

Н. А. Прохоренко



Bootstrap и CSS-препроцессор Sass

САМОЕ
НЕОБХОДИМОЕ



Адаптивный дизайн

Форматирование веб-страницы с помощью стилей

Флекс-контейнеры

Система сеток в библиотеке Bootstrap

Формы и элементы управления

Адаптивная панель навигации

Карточки, панели с вкладками, карусель

Всплывающие подсказки и уведомления

Модальные диалоговые окна

Синтаксис SCSS-файлов



Материалы
на www.bhv.ru

Н. А. Прохоренок

Bootstrap и CSS-препроцессор Sass

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.438
ББК 32.973.26-018.1
П84

Прохоренок Н. А.

П84 Bootstrap и CSS-препроцессор Sass. Самое необходимое. —
СПб.: БХВ-Петербург, 2021. — 496 с.: ил. — (Самое необходимое)

ISBN 978-5-9775-6769-5

Рассмотрена разработка адаптивных веб-сайтов, одинаково хорошо отображающихся на всех типах устройств, с использованием библиотеки Bootstrap и CSS-препроцессора Sass. Описана система сеток на основе flex-контейнера, позволяющая задавать ширину колонок, их количество и порядок следования. Рассмотрено стилевое оформление стандартных элементов и большое количество готовых компонентов Bootstrap: адаптивная панель навигации, карточки, панели с вкладками, карусель, всплывающие подсказки и уведомления, модальные диалоговые окна и др. Описана сборка SCSS-файлов библиотеки Bootstrap под свой проект с использованием CSS-препроцессора Sass, а также создание собственного проекта без участия Bootstrap. Большое количество практических примеров помогает начать работу самостоятельно. Материал тщательно подобран, хорошо структурирован и компактно изложен, что позволяет использовать книгу как удобный справочник.

Электронный архив с примерами находится на сайте издательства.

Для веб-разработчиков

УДК 004.438
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Людмила Гауль</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Дизайн серии	<i>Марины Дамбиевой</i>
Оформление обложки	<i>Карины Соловьевой</i>

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Оглавление

Введение	11
Глава 1. Форматирование веб-страницы с помощью стилей	13
1.1. Первые шаги.....	13
1.1.1. Установка библиотек Bootstrap и jQuery	14
1.1.2. Шаблон HTML-документа	18
1.1.3. Инструменты разработчика и консоль в веб-браузере Firefox.....	22
1.1.4. Адаптивный дизайн	26
1.1.5. Базовые контейнеры	28
1.1.6. Цвет фона	29
1.2. Форматирование шрифта	32
1.2.1. Имя шрифта.....	33
1.2.2. Стиль шрифта.....	34
1.2.3. Размер шрифта	34
1.2.4. Цвет текста	35
1.2.5. Жирность шрифта.....	36
1.2.6. Вертикальное расстояние между строками	37
1.3. Форматирование текста.....	37
1.3.1. Горизонтальное выравнивание текста	37
1.3.2. Вертикальное выравнивание текста	38
1.3.3. Подчеркивание и зачеркивание текста	39
1.3.4. Изменение регистра символов.....	39
1.3.5. Обработка переноса строк	40
1.3.6. Создание нижних и верхних индексов.....	41
1.3.7. Выделение фрагментов кода.....	42
1.3.8. Выделение важного фрагмента текста и аббревиатуры	44
1.3.9. Выделение цитат	45
1.3.10. Заголовки	46
1.3.11. Разделение на абзацы	46
1.3.12. Тег <i><details></i>	47
1.3.13. Горизонтальная линия	47
1.3.14. Гиперссылки.....	48

1.4. Отступы	51
1.4.1. Внешние отступы	52
1.4.2. Внутренние отступы	59
1.5. Рамки.....	64
1.5.1. Отображение рамки	64
1.5.2. Сокрытие рамки	65
1.5.3. Цвет линии рамки	65
1.5.4. Рамки со скругленными углами.....	66
1.6. Списки.....	67
1.6.1. Нумерованные списки	67
1.6.2. Маркированные списки.....	67
1.6.3. Вложенные списки.....	68
1.6.4. Списки без маркеров	68
1.6.5. Компонент <i>list-group</i> : список (введение).....	69
1.6.6. Списки определений.....	70
1.7. Таблицы	71
1.7.1. Рамки таблицы и ячеек.....	72
1.7.2. Компактное отображение содержимого таблицы	73
1.7.3. Зебра.....	73
1.7.4. Выделение строки при наведении указателя мыши.....	74
1.7.5. Изменение цвета фона для таблицы, строки и ячеек	74
1.7.6. Местоположение и выравнивание заголовка таблицы	75
1.8. Графика и видео	75
1.8.1. Работа с изображениями	76
1.8.2. Готовые значки	78
1.8.3. Добавление описания к изображению	79
1.8.4. Вставка видео	82
1.9. Форматирование блоков.....	82
1.9.1. Указание типа блока	83
1.9.2. Указание размеров.....	85
1.9.3. Атрибут <i>overflow</i>	86
1.9.4. Управление обтеканием	88
1.9.5. Позиционирование блока.....	88
1.9.6. Управление отображением элемента	92
1.9.7. Создание тени для блока	92
1.10. Управление выделением текста.....	93
1.11. Атрибут <i>pointer-events</i>	93
Глава 2. Размещение элементов внутри окна	94
2.1. Flex-контейнеры.....	94
2.1.1. Направление выравнивания элементов внутри контейнера	94
2.1.2. Перенос на новую строку.....	99
2.1.3. Размеры элемента	100
2.1.4. Растяжение элементов	100
2.1.5. Сжатие элементов	101
2.1.6. Одновременное указание характеристик элементов.....	102
2.1.7. Выравнивание элементов внутри контейнера	103
2.1.8. Выравнивание элементов внутри строки.....	107
2.1.9. Порядок следования элементов внутри контейнера	110

2.2. Система сеток в библиотеке Bootstrap	110
2.2.1. Создание контейнера для строки.....	111
2.2.2. Колонки одинаковой ширины.....	113
2.2.3. Перенос колонок на новую строку	115
2.2.4. Указание количества колонок в строке.....	116
2.2.5. Колонки автоматической ширины	118
2.2.6. Колонки относительной ширины	119
2.2.7. Смещение колонок	123
2.2.8. Горизонтальное выравнивание колонок внутри строки	125
2.2.9. Вертикальное выравнивание всех колонок внутри строки	126
2.2.10. Вертикальное выравнивание одной колонки внутри строки	127
2.2.11. Порядок следования колонок внутри контейнера.....	127
2.2.12. Вложенные сетки	129
Глава 3. Формы и элементы управления.....	131
3.1. Элементы управления.....	131
3.1.1. Командные кнопки	131
3.1.2. Поля для ввода данных.....	135
3.1.3. Поле, доступное только для чтения.....	138
3.1.4. Вывод пояснительной надписи.....	138
3.1.5. Списки автодополнения	144
3.1.6. Списки со значениями	144
3.1.7. Флажки	152
3.1.8. Выключатели.....	157
3.1.9. Переключатели.....	159
3.1.10. Поле выбора файла	161
3.1.11. Шкала с ползунком	167
3.1.12. Элемент для выбора цвета	168
3.2. Выравнивание и группировка элементов формы	168
3.2.1. Выравнивание элементов по вертикали.....	169
3.2.2. Выравнивание элементов по горизонтали.....	170
3.2.3. Выравнивание элементов по сетке	172
3.2.4. Группировка элементов формы.....	179
3.2.5. Выравнивание кнопок	180
3.2.6. Группа с кнопками.....	182
3.2.7. Группа с кнопками-переключателями	183
3.3. Проверка данных формы.....	190
3.4. Установка фокуса ввода	198
3.5. Порядок обхода элементов формы.....	198
3.6. Индикатор хода процесса.....	198
3.7. Компонент <i>spinner</i> : индикатор состояния загрузки	201
3.7.1. Класс <i>spinner-border</i>	201
3.7.2. Класс <i>spinner-grow</i>	202
3.7.3. Выравнивание компонента по центру или по правой стороне	203
3.7.4. Кнопка с компонентом <i>spinner</i>	203
3.8. Компонент <i>dropdown</i> : кнопка с выпадающим меню	204
3.8.1. Класс <i>dropdown-toggle</i> : кнопка или ссылка с меню.....	204
3.8.2. Класс <i>dropdown-toggle-split</i> : кнопка с двумя зонами и меню	205
3.8.3. Выпадающее меню с отдельными пунктами.....	207

3.8.4. Активные и недоступные пункты меню	208
3.8.5. Выпадающее меню с произвольным содержимым.....	209
3.8.6. Направление выпадения меню.....	209
3.8.7. Положение меню	210
3.8.8. Управление компонентом из программы	212
3.8.9. Обработка событий.....	213
Глава 4. Готовые компоненты	217
4.1. Компонент <i>jumbotron</i> : выделение важного содержимого	217
4.2. Компонент <i>alert</i> : уведомления	218
4.2.1. Создание уведомления	218
4.2.2. Закрытие уведомления	219
4.2.3. Обработка событий.....	221
4.3. Компонент <i>badge</i> : выделение фрагментов текста.....	223
4.4. Компонент <i>card</i> : карточки	225
4.4.1. Разделы карточки.....	225
4.4.2. Тело карточки	227
4.4.3. Ширина и высота карточки.....	229
4.4.4. Изменение цветовой схемы карточки	230
4.4.5. Изображение внутри карточки	231
4.4.6. Группа из карточек без отступов.....	233
4.4.7. Группа из карточек с отступами.....	235
4.4.8. Выравнивание карточек по сетке	236
4.4.9. Размещение карточек в трех колонках.....	238
4.5. Компонент <i>collapse</i> : сворачивание и разворачивание области с содержимым	240
4.5.1. Переключение состояния с помощью кнопки.....	241
4.5.2. Переключение состояния с помощью ссылки.....	241
4.5.3. Переключение состояния сразу нескольких областей.....	242
4.5.4. Панель «Аккордеон».....	243
4.5.5. Управление компонентом из программы	246
4.5.6. Обработка событий.....	247
4.6. Компонент <i>nav</i> : контейнер со ссылками или ярлыками вкладок.....	249
4.6.1. Горизонтальное размещение ссылок	249
4.6.2. Выравнивание ссылок внутри контейнера	250
4.6.3. Вертикальное размещение ссылок	251
4.6.4. Активное состояние ссылки.....	252
4.6.5. Контейнер с ярлыками вкладок	253
4.6.6. Ссылка или ярлык вкладки с выпадающим меню.....	254
4.7. Компонент <i>tab</i> : панель с вкладками	255
4.7.1. Создание компонента	256
4.7.2. Вертикальное размещение ярлыков вкладок.....	258
4.7.3. Компонент <i>list-group</i> в качестве ярлыков вкладок.....	259
4.7.4. Карточки с панелью вкладок	260
4.7.5. Управление компонентом из программы	262
4.7.6. Обработка событий.....	263
4.8. Компонент <i>list-group</i> : список с пунктами, ссылками или кнопками	267
4.8.1. Список с пунктами, содержащими текст	267
4.8.2. Размещение пунктов по горизонтали.....	268

4.8.3. Список со ссылками или кнопками	269
4.8.4. Изменение цветовой схемы	270
4.8.5. Список с пунктами, содержащими произвольные элементы	271
4.9. Компонент <i>breadcrumb</i> : «хлебные крошки»	272
4.10. Компонент <i>pagination</i> : постраничная навигация	273
4.11. Компонент <i>navbar</i> : панель навигации	276
4.11.1. Создание панели и вывод простого текста	277
4.11.2. Вывод названия фирмы или проекта	278
4.11.3. Добавление блока со ссылками	279
4.11.4. Добавление ссылки с выпадающим меню	280
4.11.5. Сворачивание и разворачивание блока со ссылками	281
4.11.6. Добавление формы	282
4.11.7. Позиционирование панели навигации	283
4.12. Компонент <i>carousel</i> : циклическое повторение слайдов	285
4.12.1. Создание компонента	286
4.12.2. Управление компонентом пользователем	287
4.12.3. Добавление надписей	289
4.12.4. Способы смены слайдов и указание интервала	290
4.12.5. Управление компонентом из программы	291
4.12.6. Обработка событий	293
4.13. Компонент <i>scrollspy</i> : отслеживание прокрутки	297
4.13.1. Создание компонента	297
4.13.2. Управление компонентом из программы	299
4.13.3. Обработка событий	300
4.14. Компонент <i>tooltip</i> : всплывающие подсказки	302
4.14.1. Добавление всплывающей подсказки к элементу	302
4.14.2. Местоположение всплывающей подсказки	304
4.14.3. Управление компонентом из программы	305
4.14.4. Обработка событий	308
4.15. Компонент <i>popover</i> : всплывающие информеры	312
4.15.1. Добавление всплывающего информера к элементу	312
4.15.2. Местоположение всплывающего информера	315
4.15.3. Управление компонентом из программы	315
4.15.4. Обработка событий	318
4.16. Компонент <i>toast</i> : всплывающие уведомления	323
4.16.1. Создание и отображение уведомления	323
4.16.2. Управление компонентом из программы	325
4.16.3. Обработка событий	326
4.17. Компонент <i>modal</i> : модальные диалоговые окна	330
4.17.1. Создание окна	331
4.17.2. Отображение и закрытие окна	332
4.17.3. Изменение размеров окна	333
4.17.4. Размещение содержимого внутри окна	334
4.17.5. Вертикальное выравнивание окна по центру	336
4.17.6. Полноэкранный режим	336
4.17.7. Управление компонентом из программы	338
4.17.8. Обработка событий	339

Глава 5. CSS-препроцессор Sass	344
5.1. Первые шаги.....	344
5.1.1. Установка Node.js	344
5.1.2. Работа с командной строкой.....	348
5.1.3. Установка Sass с помощью NPM.....	351
5.1.4. Создание пакета и добавление файла <code>package.json</code>	351
5.1.5. Создание CSS-файла из SCSS-файла	354
5.1.6. Отслеживание изменений SCSS-файлов.....	357
5.1.7. Интерактивный режим	358
5.1.8. Комментарии.....	358
5.1.9. Директивы <code>@debug</code> , <code>@warn</code> и <code>@error</code>	359
5.2. Переменные и типы данных.....	360
5.2.1. Именованние переменных в Sass.....	360
5.2.2. Области видимости переменных	361
5.2.3. Инструкция <code>!global</code>	362
5.2.4. Инструкция <code>!default</code>	362
5.2.5. Типы данных	363
5.2.6. Подстановка значений переменных	365
5.2.7. Проверка существования переменной	367
5.3. Операторы и циклы	368
5.3.1. Математические операторы.....	368
5.3.2. Приоритет выполнения операторов	370
5.3.3. Операторы для работы со строками.....	371
5.3.4. Операторы сравнения.....	372
5.3.5. Оператор ветвления <code>@if</code> и функция <code>if()</code>	375
5.3.6. Цикл <code>@for</code>	376
5.3.7. Цикл <code>@while</code>	377
5.3.8. Цикл <code>@each</code> : перебор элементов списка или ассоциативного массива	378
5.4. Числа	380
5.4.1. Математические константы	381
5.4.2. Основные функции для работы с числами	381
5.4.3. Округление чисел.....	383
5.4.4. Тригонометрические функции.....	384
5.4.5. Работа с единицами измерения.....	385
5.4.6. Преобразование числа в строку.....	386
5.4.7. Генерация псевдослучайных чисел	386
5.5. Списки.....	387
5.5.1. Создание списка.....	387
5.5.2. Определение количества элементов.....	389
5.5.3. Получение и изменение значения элемента	389
5.5.4. Перебор элементов списка.....	390
5.5.5. Добавление элемента в конец списка.....	391
5.5.6. Объединение списков	392
5.5.7. Поиск значения в списке	394
5.5.8. Сравнение списков.....	395
5.6. Ассоциативные массивы	395
5.6.1. Создание ассоциативного массива	396
5.6.2. Определение количества элементов.....	397

5.6.3. Получение значения по ключу.....	397
5.6.4. Проверка существования ключа.....	397
5.6.5. Добавление элементов и изменение значения.....	398
5.6.6. Удаление элементов	399
5.6.7. Перебор элементов	399
5.6.8. Преобразование ассоциативного массива в список.....	400
5.6.9. Сравнение ассоциативных массивов.....	400
5.7. Строки.....	401
5.7.1. Создание строки.....	401
5.7.2. Кодировка файлов.....	403
5.7.3. Определение количества символов в строке	403
5.7.4. Изменение регистра символов.....	404
5.7.5. Получение фрагмента строки	404
5.7.6. Вставка фрагмента в строку.....	405
5.7.7. Поиск в строке	405
5.7.8. Сравнение строк.....	406
5.7.9. Создание уникального идентификатора	406
5.8. Работа с цветом.....	406
5.8.1. Способы указания значения.....	407
5.8.2. Получение значений компонентов цвета	410
5.8.3. Изменение значений компонентов цвета	412
5.8.4. Изменение насыщенности цвета	415
5.8.5. Изменение яркости цвета	416
5.8.6. Изменение прозрачности цвета	417
5.8.7. Преобразование цвета в оттенки серого	419
5.8.8. Смешивание цветов	419
5.8.9. Инвертирование цвета.....	420
5.8.10. Получение значения в формате <i>#AARRGGBB</i>	420
5.9. Пользовательские функции.....	420
5.9.1. Создание функции и ее вызов.....	421
5.9.2. Расположение определений функций	423
5.9.3. Способы передачи значений в функцию	423
5.9.4. Необязательные параметры	425
5.9.5. Передача произвольного количества значений.....	426
5.9.6. Передача ссылки на функцию и обратный вызов	426
5.9.7. Проверка существования функции.....	428
5.10. Модули.....	428
5.10.1. Директива <i>@import</i>	429
5.10.2. Вложенные директивы <i>@import</i>	430
5.10.3. Пути поиска модулей.....	430
5.10.4. Индексные файлы	432
5.10.5. Директива <i>@use</i>	432
5.10.6. Изменение названия пространства имен	433
5.10.7. Импорт всех идентификаторов из модуля.....	434
5.10.8. Частные идентификаторы внутри модуля	434
5.10.9. Переопределение значений переменных из модуля	435
5.10.10. Директива <i>@forward</i>	436
5.10.11. Миксин <i>meta.load-css()</i>	439

5.10.12. Подключение CSS-файлов	441
5.10.13. Получение всех переменных внутри модуля.....	442
5.10.14. Получение всех функций внутри модуля.....	442
5.11. Работа с селекторами и атрибутами стилей.....	443
5.11.1. Основные селекторы	444
5.11.2. Привязка к элементам и вложенные правила	445
5.11.3. Директива <i>@at-root</i>	448
5.11.4. Оператор <i>&</i> : список с родительскими селекторами.....	451
5.11.5. Вложенные атрибуты	452
5.11.6. Генерация названий селекторов и атрибутов	453
5.11.7. Вставка атрибута в зависимости от условия	454
5.11.8. Функции для работы с селекторами.....	455
5.12. Шаблоны и миксины	457
5.12.1. Директива <i>@extend</i> и шаблонные селекторы	457
5.12.2. Директива <i>@extend</i> и простые селекторы.....	459
5.12.3. Инструкция <i>!optional</i>	460
5.12.4. Миксины (примеси).....	460
5.12.5. Расположение определений миксинов.....	463
5.12.6. Способы передачи значений	464
5.12.7. Необязательные параметры	465
5.12.8. Передача произвольного количества значений.....	466
5.12.9. Директива <i>@content</i> и блоки содержимого	466
5.12.10. Проверка существования миксина	468
5.13. Отличия SASS-файлов от SCSS-файлов	468
5.14. Программа node-sass (LibSass).....	470
5.14.1. Установка node-sass	470
5.14.2. Создание CSS-файла из SCSS-файла	471
5.14.3. Отслеживание изменений SCSS-файлов.....	475
5.14.4. Различия между node-sass и Sass	475
5.15. Сборка SCSS-файлов библиотеки Bootstrap под свой проект.....	477
Заключение	482
Приложение. Описание электронного архива	483
Предметный указатель	484



Введение

Добро пожаловать в мир Bootstrap, Sass и адаптивного веб-дизайна!

Благодаря доступному мобильному Интернету существенно расширился список устройств, с которых пользователи заходят на сайт. Это не только обычные компьютеры, но и ноутбуки, смартфоны и планшеты. Ширина экрана у этих устройств сильно различается. Сайт, сделанный только для мобильных устройств, не очень удобно просматривать на большом мониторе. В то же время сайт, сделанный только для больших мониторов, становится очень неудобным для просмотра на мобильных устройствах.

Ранее для решения этой проблемы многие разработчики делали несколько версий сайта — например, полную и мобильную. При заходе пользователя со смартфона на полную версию сайта обычно осуществляется автоматическое перенаправление на его мобильную версию. Однако разработчики часто забывают обновлять и проверять работоспособность мобильной версии, в результате чего у пользователей возникают проблемы. Чтобы избежать таких проблем, следует сделать сайт *адаптивным*, т. е. изменяющим свою структуру в зависимости от ширины экрана.

Создавать адаптивные сайты, одинаково хорошо отображающиеся на всех типах устройств, независимо от ширины экрана, позволяет библиотека Bootstrap. Благодаря предоставляемой ею адаптивной системе сеток на основе flex-контейнера, можно для различных точек останова задавать ширину колонок и их количество в одном ряду, управлять видимостью колонок, а также менять их порядок следования. И все это только с помощью CSS — без использования скриптов. Простота и удобство использования этой системы сеток сделали библиотеку Bootstrap очень популярной.

Помимо адаптивной системы сеток библиотека Bootstrap содержит стилевое оформление для стандартных элементов веб-страницы и предоставляет большое количество готовых компонентов: адаптивную панель навигации, карточки, панели с вкладками, карусель, всплывающие подсказки и уведомления, модальные диалоговые окна и др.

Существует возможность выполнить сборку библиотеки Bootstrap под свой проект. Для этого потребуются знания основ CSS-препроцессора Sass, используемого разработчиками Bootstrap для создания CSS-файла. Sass — это технология для профессионалов в CSS, причем очень логичная и удобная, простая в изучении. Знание Sass позволит:

- ◆ уменьшить размер CSS-файла библиотеки Bootstrap за счет отключения неиспользуемых модулей (например, модулей со стилями компонентов);

- ◆ изменить значения некоторых переменных (например, поменять цвет);
- ◆ добавить свои цветовые темы (достаточно указать название темы и ее цвет, и весь код будет сгенерирован автоматически при компиляции);
- ◆ добавить свои произвольные стили или переопределить стили библиотеки Bootstrap;
- ◆ создать собственный проект без участия Bootstrap. Этот пункт особенно важен, если вы являетесь профессиональным разработчиком или хотите им стать.

Давайте рассмотрим структуру книги.

- ◆ В *главе 1* мы загрузим и подключим к шаблону HTML-документа библиотеки Bootstrap и jQuery, изучим основные стили библиотеки Bootstrap, познакомимся с точками останова и базовыми контейнерами.
- ◆ *Глава 2* посвящена размещению элементов внутри окна с помощью flex-контейнеров и адаптивной системы сеток библиотеки Bootstrap.
- ◆ В *главе 3* мы познакомимся со стилевыми классами, позволяющими сделать стандартные элементы управления внутри формы удобными и красивыми, а также рассмотрим несколько нестандартных элементов: выключатели, кнопки-переключатели и кнопки с выпадающим меню.
- ◆ В *главе 4* приведено описание большого количества нестандартных готовых компонентов: адаптивной панели навигации, карточек, панелей с вкладками, всплывающих подсказок и уведомлений, модальных диалоговых окон и др.
- ◆ *Глава 5* полностью посвящена CSS-препроцессору Sass, который позволит заполнить сборку SCSS-файлов библиотеки Bootstrap под свой проект, а также создать собственный проект без участия Bootstrap.

Для полного понимания материала книги потребуется знание основ HTML 5 и CSS 3 в объеме первых двух глав из 5-го издания книги «HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера»¹. Более глубокие знания не требуются, т. к. Bootstrap изменяет стилизацию всех основных HTML-элементов и преобразует атрибуты стилей из CSS в стилевые классы. Поэтому многие теги и атрибуты будут рассматриваться в этой книге, но только в случае изменения их свойств библиотекой Bootstrap.

Кроме того, для управления компонентами и обработки событий может потребоваться знание языка программирования JavaScript и умение пользоваться библиотекой jQuery. Описание JavaScript приведено в третьей главе книги «HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера», а мою книгу по jQuery можно найти на сайте <http://прохоронок.рф/>. Впрочем, при использовании Bootstrap 5 знание библиотеки jQuery не потребуется.

Все листинги из этой книги вы найдете в файле Listings.doc, электронный архив с которым можно загрузить с FTP-сервера издательства «БХВ» по ссылке: <ftp://ftp.bhv.ru/9785977567695.zip> или со страницы книги на сайте <https://bhv.ru/> (см. приложение).

¹ См. <https://bhv.ru/product/html-javascript-php-i-mysql-dzhentelmenskij-nabor-web-mastera-5-izd/>.



ГЛАВА 1

Форматирование веб-страницы с помощью стилей

Прежде чем мы начнем рассматривать библиотеки Bootstrap и Sass, необходимо сделать замечание. Не забывайте, что книги по программированию нужно не только читать, но и выполнять все примеры, а также экспериментировать, изменяя что-нибудь в примерах. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить библиотеки! Сядьте с книгой перед компьютером и выполняйте все примеры! Помните, что книга может содержать только черно-белые изображения, поэтому многие иллюстрации вставлять в книгу не имеет смысла. Если вы не будете запускать примеры из книги, то просто не увидите результат и в голове ничего не отложится.

Чем больше вы будете делать самостоятельно, тем большему научитесь. Обычно после первого прочтения многое непонятно, после второго прочтения некоторые вещи становятся понятнее, после третьего — еще лучше, ну а после N-го прочтения не понимаешь, как могло быть что-то непонятно после первого прочтения. Повторение — мать учения. Наберитесь терпения, и вперед.

1.1. Первые шаги

Итак, приступим к изучению библиотеки Bootstrap и начнем с установки необходимых программ. Перед установкой библиотек создадим следующую структуру каталогов:

```
book\  
  css\  
  img\  
  js\  

```

Каталог book лучше разместить в корне какого-либо диска. В моем случае это будет диск C:, следовательно, путь к содержимому каталога: C:\book. Можно создать каталог book и в любом другом удобном месте, т. к. мы будем указывать относительный путь до библиотек, а не абсолютный.

Внутри каталога `C:\book` нужно создать три вложенных каталога:

- ◆ `css` — в этом каталоге мы разместим файлы каскадных таблиц стилей (CSS);
- ◆ `img` — в этом каталоге мы станем сохранять изображения и значки;
- ◆ `js` — в этом каталоге мы разместим файлы программ на языке JavaScript.

1.1.1. Установка библиотек Bootstrap и jQuery

Теперь приступим к загрузке всех необходимых файлов. Для загрузки библиотеки Bootstrap переходим на сайт <https://getbootstrap.com/> и нажимаем кнопку **Download**. На открывшейся странице представлены все варианты получения библиотеки. Переходим в раздел **Source files** и нажимаем кнопку **Download source**, чтобы загрузить все исходные файлы библиотеки в виде архива. Мы будем изучать возможности версии 4.5.2, поэтому лучше скачать именно эту версию, а не последнюю доступную, т. к. она может отличаться от версии, рассматриваемой в книге. Прямая ссылка на архив библиотеки: <https://github.com/twbs/bootstrap/archive/v4.5.2.zip>. Если нужны только сжатые файлы, то можно скачать архив из раздела **Compiled CSS and JS**. Распаковываем загруженный архив в какой-либо каталог.

Сжатые файлы в архиве с исходными файлами расположены в каталоге `dist`. Если скачать архив из раздела **Compiled CSS and JS**, то его содержимое и есть содержимое каталога `dist`. В каталоге `dist\css` расположены следующие файлы:

- ◆ `bootstrap.css` — основной файл со стилями без сжатия;
- ◆ `bootstrap.min.css` — основной файл со стилями в сжатом виде;
- ◆ `bootstrap-reboot.css` — файл с нормализацией (сбросом) стилей без сжатия;
- ◆ `bootstrap-reboot.min.css` — файл с нормализацией (сбросом) стилей в сжатом виде;
- ◆ `bootstrap-grid.css` — стили для использования системы сеток без сжатия;
- ◆ `bootstrap-grid.min.css` — стили для использования системы сеток в сжатом виде.

Кроме того, в каталоге `dist\css` расположены также файлы с расширением `map`:

`bootstrap.css.map`

`bootstrap.min.css.map`

`bootstrap-reboot.css.map`

`bootstrap-reboot.min.css.map`

`bootstrap-grid.css.map`

`bootstrap-grid.min.css.map`

Если расположить файл с расширением `map` рядом с одноименным файлом, имеющим расширение `css` или `js`, то веб-браузер в окне консоли будет показывать название файла с исходным кодом, а не название подключенного файла в сжатом виде.

В каталоге `dist\js` расположены следующие файлы:

- ◆ `bootstrap.js` — основной файл с кодом на языке JavaScript без сжатия;
- ◆ `bootstrap.min.js` — основной файл с кодом на языке JavaScript в сжатом виде;
- ◆ `bootstrap.bundle.js` — основной файл с кодом на языке JavaScript без сжатия, дополнительно включающий код библиотеки `Popper.js`;
- ◆ `bootstrap.bundle.min.js` — основной файл с кодом на языке JavaScript в сжатом виде, дополнительно включающий код библиотеки `Popper.js`.

Файлы с кодом на языке JavaScript потребуются при работе с готовыми компонентами, которые мы рассмотрим в *главах 3 и 4*. Кроме того, в каталоге `dist\js` расположены файлы с расширением `map`:

```
bootstrap.js.map
```

```
bootstrap.min.js.map
```

```
bootstrap.bundle.js.map
```

```
bootstrap.bundle.min.js.map
```

Очень редко на странице мы используем сразу все компоненты. Если нужно сократить объем подключаемых файлов, то обратите внимание на содержимое каталога `js\dist`, в котором расположены файлы с JavaScript кодом каждого компонента по отдельности. Исходный код для каждого компонента можно найти в каталоге `js\src`.

Итак, копируем файл `bootstrap.min.css` из каталога `dist\css` архива в каталог `C:\book\css`. Код для подключения файла стилей в файле `C:\book\test.html` будет выглядеть следующим образом (код размещаем в разделе `HEAD` HTML-документа):

```
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
```

Далее копируем файлы `bootstrap.min.js` и `bootstrap.bundle.min.js` из каталога `dist\js` архива в каталог `C:\book\js`. Код для подключения файла `bootstrap.min.js` в файле `C:\book\test.html` будет выглядеть следующим образом (код размещаем в самом конце HTML-документа после подключения библиотеки `jQuery`):

```
<script src="js/bootstrap.min.js"></script>
```

Файл `bootstrap.bundle.min.js` содержит код файла `bootstrap.min.js`, а также код библиотеки `Popper.js`. Поэтому подключать одновременно эти файлы нельзя. Библиотека `Popper.js` понадобится при работе с некоторыми компонентами, реализующими различные всплывающие элементы, — например, подсказки. Можно скачать библиотеку `Popper.js` с сайта <https://popper.js.org/> и подключить ее отдельно перед подключением файла `bootstrap.min.js`, но мы будем просто подключать файл `bootstrap.bundle.min.js` вместо файла `bootstrap.min.js`:

```
<script src="js/bootstrap.bundle.min.js"></script>
```

Помимо версии 4.5.2 библиотеки `Bootstrap`, мы рассмотрим возможности ее версии 5.0.0-alpha1. Для загрузки этой версии переходим на сайт <https://v5.getbootstrap.com/> и нажимаем кнопку **Download**. На открывшейся странице представлены все варианты получения библиотеки. Переходим в раздел **Source files** и нажимаем кнопку

Download source, чтобы загрузить все исходные файлы библиотеки в виде архива. Прямая ссылка на архив библиотеки: <https://github.com/twbs/bootstrap/archive/v5.0.0-alpha1.zip>. Распаковываем загруженный архив в какой-либо каталог.

Переименовываем файл `bootstrap.min.css`, расположенный в каталоге `dist\css` архива, в `bootstrap5.min.css` и копируем его в каталог `C:\book\css`. Далее переименовываем файлы `bootstrap.min.js` и `bootstrap.bundle.min.js`, расположенные в каталоге `dist\js` архива, в `bootstrap5.min.js` и `bootstrap5.bundle.min.js` соответственно и копируем их в каталог `C:\book\js`.

Библиотека Bootstrap версии 4 зависит от библиотеки jQuery. В Bootstrap версии 5 такой зависимости нет, но если библиотека jQuery подключена, то можно, как и прежде, использовать интерфейс доступа через библиотеку jQuery. Для работы компонентов достаточно использовать версию `slim`, в которую не входят модули для работы с анимацией и технологией AJAX. Однако мы подключим полную версию библиотеки jQuery.

Для загрузки библиотеки jQuery переходим на страницу <https://jquery.com/download/> и скачиваем файл `jquery-3.5.1.min.js` или сразу загружаем файл по ссылке <https://code.jquery.com/jquery-3.5.1.min.js>. Переименовываем файл `jquery-3.5.1.min.js` в `jquery.min.js` и сохраняем в каталоге `C:\book\js`. Подключение библиотеки jQuery в файле `C:\book\test.html` будет выглядеть следующим образом (подключение выполняется перед файлом `bootstrap.min.js` или `bootstrap.bundle.min.js`):

```
<script src="js/jquery.min.js"></script>
```

Таким образом, окончательная структура файлов и каталогов будет выглядеть так:

```
C:\book\  
  css\  
    bootstrap.min.css  
    bootstrap5.min.css  
  img\  
  js\  
    bootstrap.min.js  
    bootstrap5.min.js  
    bootstrap.bundle.min.js  
    bootstrap5.bundle.min.js  
    jquery.min.js
```

Вместо загрузки архивов можно выполнить подключение файлов библиотек через CDN. Если посетитель ранее заходил на другой сайт, на котором библиотеки также подгружались с этого же CDN, то веб-браузер не будет повторно загружать библиотеку, а использует данные, сохраненные в кеше. Таким образом, скорость работы вашего сайта может увеличиться. В этом и заключается преимущество данного

метода. Однако если CDN будет недоступен, то возможны проблемы. Код подключения файлов для версии 4.5.2 выглядит так:

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css" integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpsSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
crossorigin="anonymous">
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-
DfXdz2htPH0lsSS5nCTpuj/zy4C+OGpamoFVy38MVBne+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.
min.js" integrity="sha384-
9/reFTGAW83EW2RDu2S0VkaIzap3H66lZ81PoYlFhbGU+6BZp6G7niu735Sk7lN"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.
min.js" integrity="sha384-
B4gt1jrGC7Jh4AgTPSdU0Bvf08shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
crossorigin="anonymous"></script>
```

Вот список поддерживаемых библиотекой Bootstrap версии 4.5.2 веб-браузеров (полный их список приведен на странице: <https://getbootstrap.com/docs/4.5/getting-started/browsers-devices/>):

- ◆ Chrome >= 45
- ◆ Firefox >= 38
- ◆ Edge >= 12
- ◆ Internet Explorer >= 10
- ◆ iOS >= 9
- ◆ Safari >= 9
- ◆ Android >= 4.4
- ◆ Opera >= 30

Список веб-браузеров, поддерживаемых библиотекой Bootstrap версии 5.0.0-alpha1 (полный их список приведен на странице: <https://v5.getbootstrap.com/docs/5.0/getting-started/browsers-devices/>):

- ◆ Chrome >= 60
- ◆ Firefox >= 60
- ◆ Edge >= 16.16299
- ◆ iOS >= 10
- ◆ Safari >= 10
- ◆ Android >= 6

- ◆ not Internet Explorer <= 11
- ◆ not ExplorerMobile <= 11

Обратите внимание: веб-браузер Internet Explorer в Bootstrap 5 не поддерживается.

1.1.2. Шаблон HTML-документа

Давайте создадим шаблон HTML-документа, которым будем пользоваться в дальнейших примерах, и подключим все нужные файлы библиотек (листинг 1.1).

Листинг 1.1. Шаблон HTML-документа

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Шаблон HTML-документа</title>
</head>
<body>

  <!-- Сюда вставляем HTML-код -->

<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.bundle.min.js"></script>
<script>
// Сюда вставляем код на JavaScript
console.log('jQuery v' + $.fn.jquery);
console.log('Bootstrap v' + $.fn.tooltip.Constructor.VERSION);
</script>
</body>
</html>
```

Рассмотрим структуру нашего шаблона. В первой строке содержится тег `<!doctype>`, который позволяет веб-браузеру определить формат файла и правильно отобразить все его инструкции. Мы будем работать с форматом HTML 5, для которого инструкция выглядит очень просто:

```
<!doctype html>
```

Если тег `<!doctype>` не указан, то некоторые веб-браузеры переходят в режим совместимости. В этом режиме может отличаться тип блочной модели. Поэтому при отсутствии тега `<!doctype>` разные веб-браузеры могут по-разному отображать веб-страницу.

Весь текст HTML-документа расположен между тегами `<html>` и `</html>`. Тег `<html>` содержит параметр `lang`, задающий код языка веб-страницы. Значение `ru` параметра `lang` означает русский язык:

```
<html lang="ru">
```

HTML-документ состоит из двух разделов — *заголовка* (между тегами `<head>` и `</head>`) и *содержательной части* (между тегами `<body>` и `</body>`).

Раздел `HEAD` содержит техническую информацию о странице:

- ◆ кодировку (мы работаем с кодировкой UTF-8):

```
<meta charset="utf-8">
```

- ◆ параметры для адаптивного дизайна:

```
<meta name="viewport" content="width=device-width, initial-scale=1,
                               shrink-to-fit=no">
```

- ◆ подключение файла со стилями библиотеки Bootstrap (указываем относительный путь к файлу `bootstrap.min.css`):

```
<link rel="stylesheet" type="text/css"
      href="css/bootstrap.min.css">
```

- ◆ текст, отображаемый в строке заголовка вкладки:

```
<title>Шаблон HTML-документа</title>
```

В разделе `BODY` располагается все содержимое документа. В нашем шаблоне присутствует только комментарий:

```
<!-- Сюда вставляем HTML-код -->
```

В самом конце раздела `BODY` производится подключение JavaScript-библиотек jQuery, Popper.js и Bootstrap (указываем относительный путь к файлам):

```
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.bundle.min.js"></script>
```

Файл `bootstrap.bundle.min.js` содержит библиотеки Popper.js и Bootstrap. Вместо этого файла можно подключить два файла по отдельности: `popper.min.js` и `bootstrap.min.js`:

```
<script src="js/jquery.min.js"></script>
<script src="js/popper.min.js"></script>
<script src="js/bootstrap.min.js"></script>
```

Сначала нужно подключить библиотеку jQuery, затем Popper.js, а самой последней библиотеку Bootstrap. При использовании только CSS эти строки можно убрать, но, скорее всего, вы захотите воспользоваться всеми возможностями этих библиотек. Мы будем изучать все возможности, поэтому в шаблоне подключаем все библиотеки.

Подключение библиотек производится в самом конце раздела `BODY`. Структура страницы к этому моменту уже доступна для просмотра и изменения, поэтому мы можем обращаться к элементам HTML-документа из программы на языке JavaScript напрямую. Инструкции на языке JavaScript следует размещать внутри тегов `<script>` и `</script>`, после комментария:

```
<script>
// Сюда вставляем код на JavaScript
console.log('jQuery v' + $.fn.jquery);
console.log('Bootstrap v' + $.fn.tooltip.Constructor.VERSION);
</script>
```

В этом примере мы выводим версии библиотек jQuery и Bootstrap в окно консоли:

jQuery v3.5.1

Bootstrap v4.5.2

Чтобы иметь возможность изучать Bootstrap 5, создайте еще один файл в формате HTML, вставьте в него код из листинга 1.1, затем замените в нем подключаемые файлы bootstrap.min.css и bootstrap.bundle.min.js файлами bootstrap5.min.css и bootstrap5.bundle.min.js. После запуска этого файла в окне консоли получим следующий результат:

jQuery v3.5.1

Bootstrap v5.0.0-alpha1

В целях экономии места в книге мы будем использовать шаблон из листинга 1.1, опуская описание структуры HTML-документа. HTML-код из примеров следует вставлять после этого комментария внутри шаблона:

```
<!-- Сюда вставляем HTML-код -->
```

Инструкции на языке JavaScript следует вставлять после этого комментария:

```
// Сюда вставляем код на JavaScript
```

Предыдущие вставленные фрагменты при этом следует удалять, если явным образом не указано иное. В противном случае возможны проблемы — например, конфликт имен.

Для создания HTML-документа можно воспользоваться любым текстовым редактором — например, Блокнотом. Однако мы будем работать с многобайтовой кодировкой UTF-8, а Блокнот при сохранении в этой кодировке добавляет *метку порядка байтов* (сокращенно BOM). Эта метка может стать причиной ошибок при формировании заголовков ответа сервера из программы, написанной на языке PHP. Чтобы этого избежать, в качестве редактора кода на протяжении всего обучения мы будем пользоваться программой Notepad++. Она позволяет корректно работать как с однобайтовой кодировкой windows-1251, так и с многобайтовой кодировкой UTF-8, а также имеет подсветку синтаксиса HTML, JavaScript, PHP и др.

Скачать программу Notepad++ можно абсолютно бесплатно со страницы: <https://notepad-plus-plus.org/>. Из двух вариантов (архив и инсталлятор) советую выбрать именно инсталлятор, т. к. при установке можно будет указать язык интерфейса программы. Установка Notepad++ предельно проста и в комментариях не нуждается.

Чтобы открыть какой-либо файл на редактирование, в меню **Файл** выбираем пункт **Открыть** или щелкаем правой кнопкой мыши на значке файла в Проводнике Windows и из контекстного меню выбираем пункт **Edit with Notepad++**.

Открываем редактор Notepad++ и создаем новый документ (меню **Файл** | **Новый**). В меню **Кодировки** устанавливаем флажок **Кодировать в UTF-8 (без BOM)**. Набираем код, представленный в листинге 1.1, а затем в меню **Файл** выбираем пункт **Сохранить как**. В открывшемся окне выбираем каталог — например, C:\book, в строке **Имя файла** вводим test.html, а из списка **Тип файла** выбираем пункт **All types (*.*)**. Нажимаем кнопку **Сохранить**. В заголовке окна редактора должен быть

прописан путь C:\book\test.html. Если после фрагмента html стоит точка и какое-либо другое расширение (например, txt), то при сохранении была допущена ошибка. Такая ошибка очень часто возникает при сохранении файла в редакторе Блокнот.

Чтобы посмотреть результат, нужно открыть HTML-документ в веб-браузере. В этой книге мы будем пользоваться веб-браузером Mozilla Firefox. Можно пользоваться и другим веб-браузером, но процесс работы с другим веб-браузером будет отличаться, и вы станете путаться, не зная, что делать. Поэтому на время обучения установите веб-браузер Mozilla Firefox последней версии, а уже после обучения можете вернуться к своему любимому веб-браузеру.

Запускаем веб-браузер Mozilla Firefox и переходим в главное меню. Выбираем пункт **Открыть файл** (или в меню **Файл** выбираем пункт **Открыть файл** или нажимаем комбинацию клавиш <Ctrl>+<O>), в открывшемся окне находим наш файл test.html и нажимаем кнопку **Открыть**. Если все сделано правильно, то в заголовке вкладки вы увидите надпись **Шаблон HTML-документа**.

Если веб-браузер Firefox является веб-браузером по умолчанию, то открыть HTML-документ можно с помощью двойного щелчка левой кнопкой мыши на значке файла. Если это не так, то щелкаем правой кнопкой мыши на значке файла и из контекстного меню выбираем пункт **Открыть с помощью | Выбрать программу**. В открывшемся окне находим веб-браузер Mozilla Firefox и устанавливаем флажок **Использовать это приложение для всех файлов html**. Либо в настройках веб-браузера делаем его браузером по умолчанию.

Можно также в адресной строке веб-браузера набрать команду:

```
file:///C:/book/test.html
```

и нажать клавишу <Enter>.

Мы ничего не выводили в окно веб-браузера, поэтому оно пустое. Попробуем что-нибудь вывести. Открываем файл с HTML-документом в программе Notepad++ и сразу после строки:

```
<!-- Сюда вставляем HTML-код -->
```

вставляем следующий код:

```
<div class="container bg-success text-white text-center p-2">  
    Привет, мир!  
</div>
```

Сохраняем файл (меню **Файл**, пункт **Сохранить**). Теперь вернемся в веб-браузер и обновим веб-страницу. Обновить страницу в Firefox можно следующими способами:

- ◆ нажимаем кнопку **Обновить текущую страницу** в адресной строке;
- ◆ щелкаем правой кнопкой мыши на заголовке текущей вкладки и из контекстного меню выбираем пункт **Обновить вкладку**;
- ◆ на клавиатуре нажимаем комбинацию клавиш <Ctrl>+<R> или клавишу <F5>.

В результате в окне веб-браузера отобразится прямоугольный контейнер (класс `container`) зеленого цвета (класс `bg-success`), внутри которого расположено сообщение «Привет, мир!» белого цвета (класс `text-white`), которое выровнено горизонтально

по центру контейнера (класс `text-center`). Чтобы сообщение не прижималось к границам контейнера со всех сторон, были добавлены внутренние отступы размером `0.5 rem` (класс `p-2`). В современных веб-браузерах относительная величина `1 rem` равна `16 px`, следовательно, мы добавили отступы по `8 px` со всех сторон.

Таким способом, изменяя что-либо в исходном коде, можно визуально оценивать результаты произведенных действий. Алгоритм такой: открываем исходный код, вносим корректировку, сохраняем, а затем обновляем веб-страницу.

1.1.3. Инструменты разработчика и консоль в веб-браузере Firefox

Веб-браузер Mozilla Firefox содержит панель **Инструменты разработчика**, которая позволяет не только просматривать исходный код, но и изменять значения различных параметров тегов, атрибутов стиля и т. д. При этом все изменения сразу отображаются в окне веб-браузера.

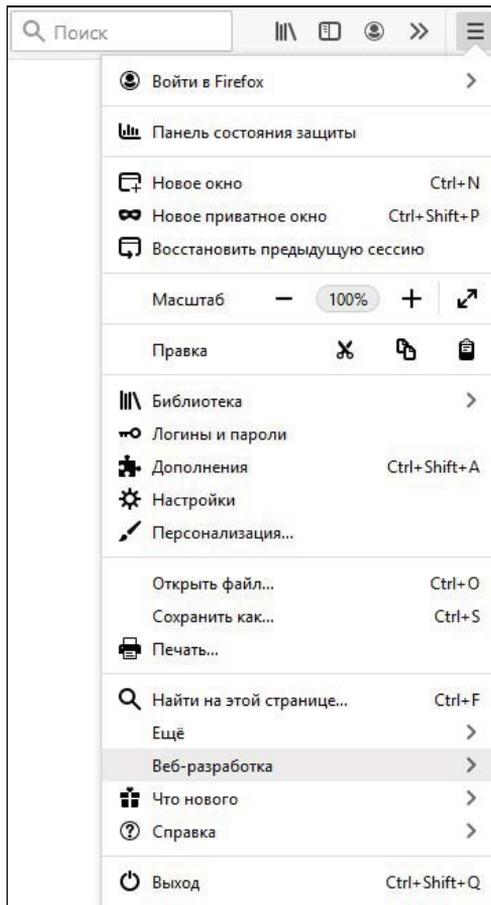


Рис. 1.1. Главное меню в веб-браузере Firefox

Чтобы открыть панель **Инструменты разработчика**, нужно в главном меню веб-браузера (рис. 1.1) выбрать пункт **Веб-разработка | Инструменты разработчика** (рис. 1.2) или нажать комбинацию клавиш <Shift>+<Ctrl>+<I>.

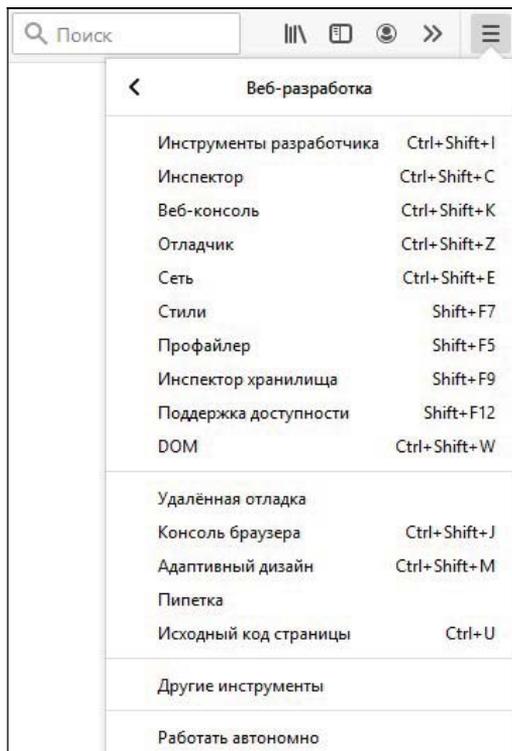


Рис. 1.2. Главное меню: пункт **Веб-разработка**

В открывшейся панели нас сейчас будет интересовать вкладка **Инспектор**. Чтобы ее сразу отобразить, в главном меню выбираем пункт **Веб-разработка | Инспектор** (см. рис. 1.2) или нажимаем комбинацию клавиш <Shift>+<Ctrl>+<C>. Можно также в меню **Инструменты** выбрать пункт **Веб-разработка | Инспектор** (рис. 1.3).

По умолчанию панель **Инструменты разработчика** (рис. 1.4) отображается в нижней части окна веб-браузера. Это поведение можно изменить, щелкнув левой кнопкой мыши на значке в виде троеточия в заголовке панели и из открывшегося меню выбрав нужный вариант. Например, можно отобразить панель в отдельном окне или прикрепить ее к боковой части окна. В меню содержится также пункт **Параметры**. Щелчок на нем или нажатие клавиши <F1> откроет панель с различными настройками.

На вкладке **Инспектор** панели **Инструменты разработчика** (см. рис. 1.4) отображается структура HTML-документа в виде дерева, а на вкладке **Разметка** можно увидеть блочную структуру каждого элемента. При наведении указателя мыши на какой-либо элемент он подсвечивается в окне веб-браузера, позволяя определить его местоположение визуально. Чтобы сразу перейти к элементу внутри HTML-

кода, щелкаем на значке **Выбрать элемент со страницы**  в заголовке панели, а затем наводим указатель мыши на элемент в окне веб-браузера и щелкаем на нем левой кнопкой мыши. Строка HTML-кода, соответствующая этому элементу, будет выделена на панели **Инспектор**.

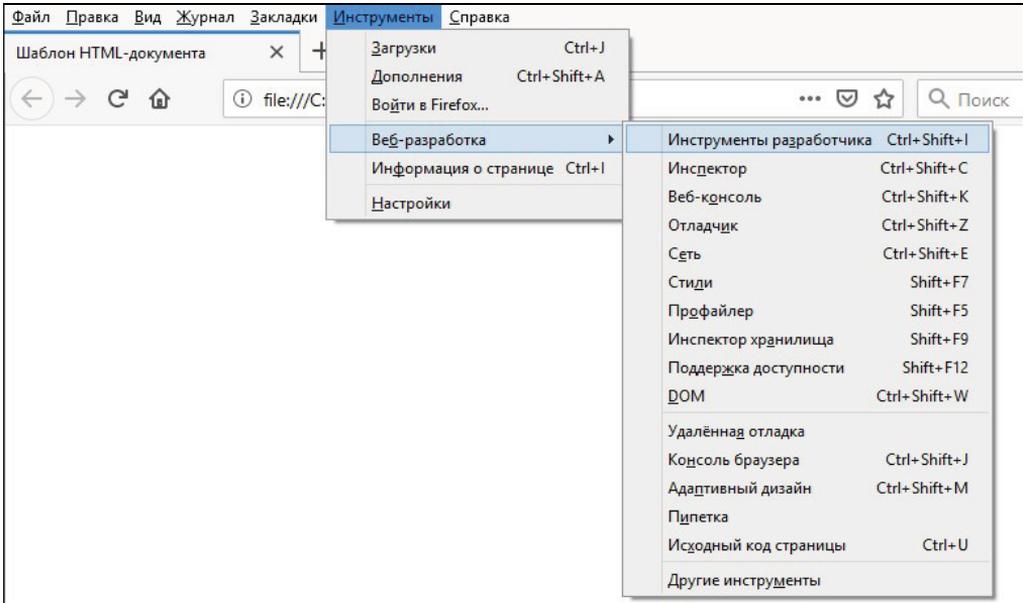


Рис. 1.3. Меню Инструменты: пункт Веб-разработка

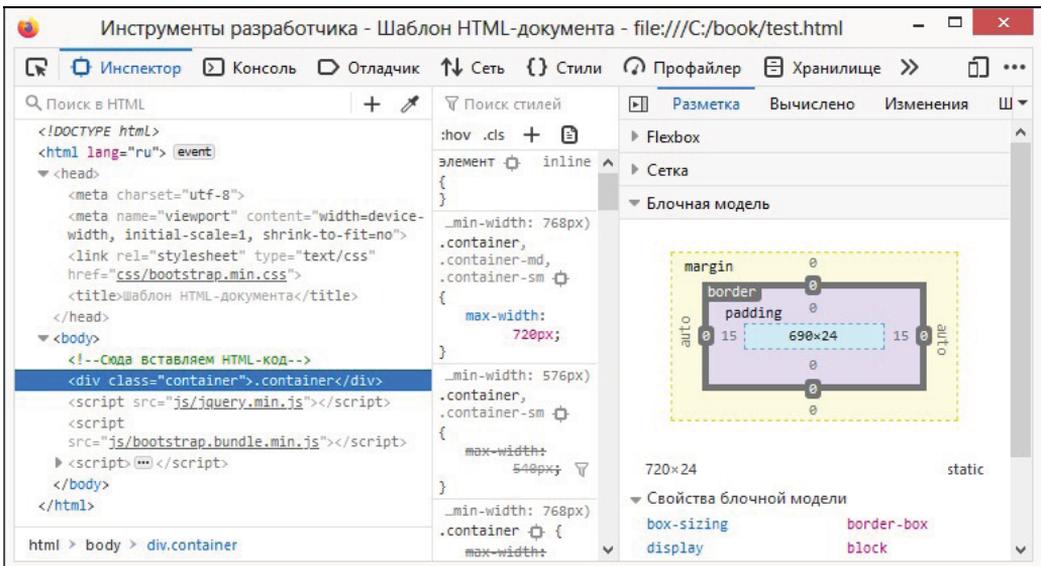


Рис. 1.4. Инструменты разработчика в веб-браузере Firefox

Все значения на этой панели можно менять и сразу видеть результат в окне. Например, изменим текст сообщения. Для этого находим элемент `div` и делаем двойной щелчок левой кнопкой мыши на тексте — текст станет доступен для редактирования. Вводим любой другой текст, нажимаем клавишу `<Enter>` и любуемся сделанной работой в окне веб-браузера.

Если нужно изменить не только текст элемента, но и поменять HTML-код, то щелкаем правой кнопкой мыши на элементе и из контекстного меню выбираем пункт **Править как HTML**. После этого станет доступным поле, в котором можно проинформировать изменения. Например, изменим код следующим образом:

```
<p><i>Текст выделен курсивом</i></p>
```

Точно таким же способом можно отобразить весь HTML-код в виде текста. Для этого достаточно щелкнуть правой кнопкой мыши на открывающем теге `<html>` и из контекстного меню выбрать пункт **Править как HTML**. Чтобы скопировать весь код, вначале выделяем его, а затем нажимаем комбинацию клавиш `<Ctrl>+<C>`. Код будет скопирован в буфер обмена, и его можно вставить в любое другое место — например, сохранить в файл. Аналогичное действие позволяет выполнить пункт из контекстного меню **Копировать | Внешний HTML**.

Пользоваться панелью **Инструменты разработчика** мы будем очень часто, поэтому способы отображения панели нужно знать наизусть.

В состав панели **Инструменты разработчика** входит **Веб-консоль**, в которую выводятся различные сообщения — например, о наличии ошибок в программе. Чтобы открыть консоль, переходим в главное меню веб-браузера и выбираем пункт **Веб-разработка | Веб-консоль** (см. рис. 1.2) или нажимаем комбинацию клавиш `<Shift>+<Ctrl>+<K>`. Можно также в меню **Инструменты** выбрать пункт **Веб-разработка | Веб-консоль** (см. рис. 1.3). В результате отобразится содержимое вкладки **Консоль** (рис. 1.5).

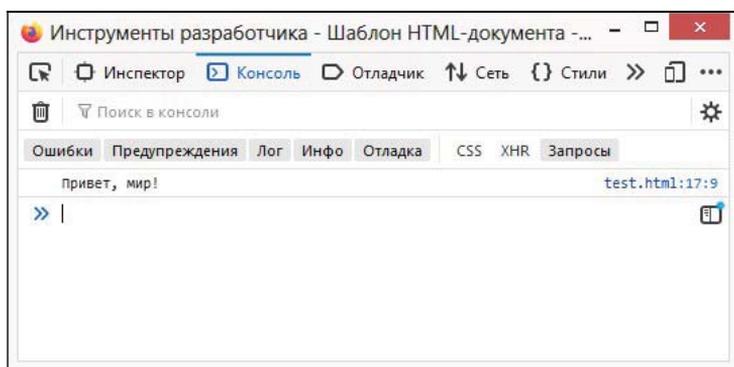


Рис. 1.5. Инструменты разработчика: вкладка **Консоль**

В верхней части вкладки **Консоль** расположены переключатели **Ошибки**, **Предупреждения**, **Лог**, **Инфо**, **Отладка**, **CSS**, **XHR** и **Запросы**, с помощью которых можно включить или отключить вывод сообщений в окно консоли. Убедитесь, что все переключатели подсвечены серым цветом. Если фон переключателя белого

цвета, то вывод сообщений отключен. Держите консоль всегда открытой при написании и отладке программы и сразу заметите наличие ошибок в программе.

Консоль можно также использовать для вывода отладочной информации. Например, в листинге 1.1 мы выводим в окно консоли версии библиотек:

```
console.log('jQuery v' + $.fn.jquery);  
console.log('Bootstrap v' + $.fn.tooltip.Constructor.VERSION);
```

Этим способом мы будем очень часто пользоваться для вывода результатов выполнения учебных скриптов. Для вывода сообщения в консоль используется следующий код:

```
console.log(<Текст сообщения или какие-либо данные>);
```

Например:

```
console.log('Привет, мир!');
```

Добавьте этот код сразу после комментария (см. листинг 1.1):

```
// Сюда вставляем код на JavaScript
```

Сообщение «Привет, мир!» отобразится на вкладке **Консоль**, если переключатель **Лог** подсвечен серым цветом.

1.1.4. Адаптивный дизайн

В библиотеке Bootstrap прописаны медиазапросы для различных размеров экрана, заданных с помощью *точек останова*. Используя специальные классы, можно создать *адаптивный дизайн* страницы. Такая страница будет подстраивать содержимое под различную ширину экрана и учитывать другие характеристики устройства — например, ориентацию экрана смартфона. Помните, что мобильный Интернет стал доступным всем, поэтому не забывайте про владельцев смартфонов и планшетов, т. к. их становится все больше и больше.

Веб-браузер Firefox содержит инструмент, который позволяет наглядно увидеть, как отображается содержимое страницы при различных характеристиках устройства. Переходим в главное меню браузера и выбираем пункт **Веб-разработка | Адаптивный дизайн** (см. рис. 1.2) или нажимаем комбинацию клавиш <Shift>+<Ctrl>+<M>. Можно также в меню **Инструменты** выбрать пункт **Веб-разработка | Адаптивный дизайн** (см. рис. 1.3). В окне веб-браузера отобразится содержимое страницы внутри рамки (рис. 1.6). Размер этой рамки можно задать, введя ширину и высоту в поля ввода, расположенные над рамкой, или взявшись мышью за правую или нижнюю ее границу. В заголовке рамки содержится список доступных устройств с уже настроенными характеристиками, а также значок, с помощью которого можно изменить ориентацию экрана.

ПРИМЕЧАНИЕ

Медиазапросы поддерживают также теги и <picture>.

При использовании адаптивного дизайна не забудьте в раздел HEAD добавить следующий код:

```
<meta name="viewport"  
      content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

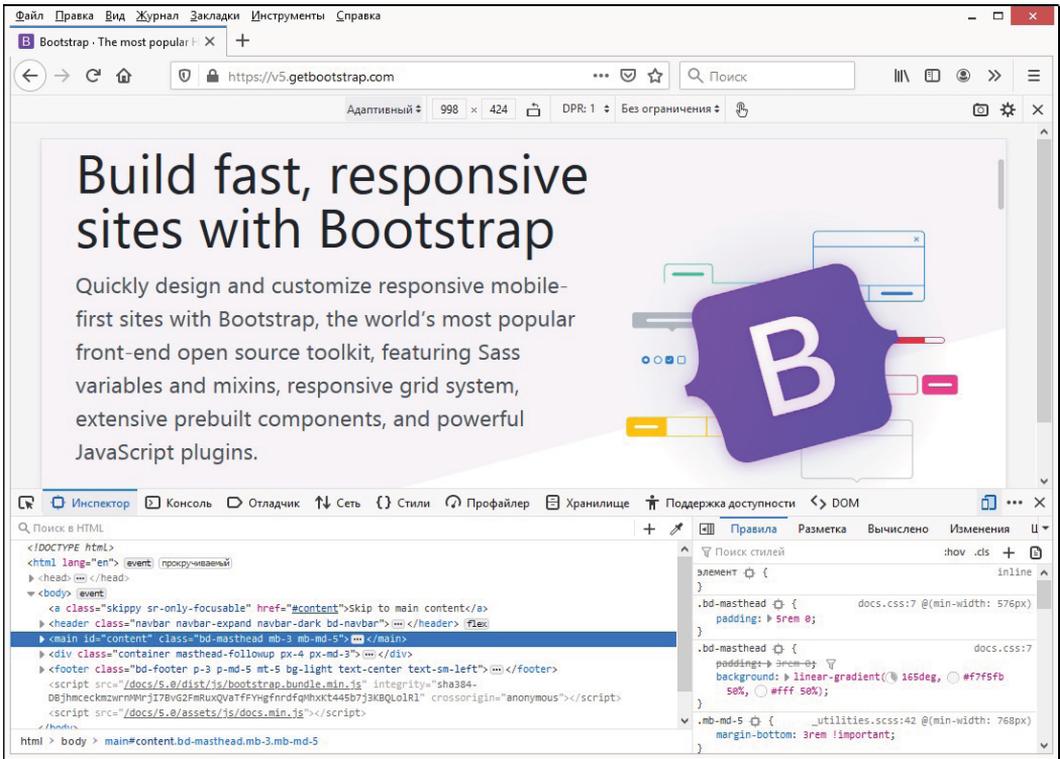


Рис. 1.6. Инструменты разработчика: режим адаптивного дизайна

В библиотеке Bootstrap используются следующие сокращения для различных устройств: sm, md, lg, xl и xxl (в Bootstrap 5). Эти сокращения (точки останова) присутствуют в названиях стилевых классов. Вот минимальная ширина экранов для различных сокращений:

```
@media (min-width: 576px) {
    /* sm (Small; 576px и более) */
}
@media (min-width: 768px) {
    /* md (Medium; 768px и более) */
}
@media (min-width: 992px) {
    /* lg (Large; 992px и более) */
}
@media (min-width: 1200px) {
    /* xl (Extra large; 1200px и более) */
}
@media (min-width: 1400px) {
    /* Bootstrap 5 */
    /* xxl (Extra extra large; 1400px и более) */
}
```

Значения точек останова доступны через следующие CSS-переменные в Bootstrap 4 (в Bootstrap 5 эти переменные отсутствуют):

```
--breakpoint-xs: 0;          /* Extra small / phone      */
--breakpoint-sm: 576px;     /* Small          / phone      */
--breakpoint-md: 768px;     /* Medium        / tablet     */
--breakpoint-lg: 992px;     /* Large         / desktop    */
--breakpoint-xl: 1200px;    /* Extra large   / wide desktop */
```

Рассмотрим несколько примеров. Сначала отобразим текст по центру внутри рамки для всех точек останова:

```
<div class="border text-center">Текст</div>
```

При ширине от 0 до 1199 px выполним выравнивание текста по центру, а при большей ширине — по правому краю:

```
<div class="border text-center text-xl-right">Текст</div>
```

Выполним разное выравнивание для всех точек останова в Bootstrap 4:

```
<div class="border text-center text-sm-right text-md-center text-lg-left text-xl-right">Текст</div>
```

Внимательный читатель наверняка заметил, что мы не использовали в названиях стилевых классов точку останова `xs`. Дело в том, что стилевой класс, предназначенный для точки останова `xs`, не содержит в названии точку останова. Иными словами, библиотека Bootstrap изначально предназначена для мобильных устройств (принцип `mobile-first`). Если нет других стилевых классов с названием точки останова, то правила стилей действуют для всех устройств.

1.1.5. Базовые контейнеры

В библиотеке Bootstrap существуют следующие стилевые классы, описывающие характеристики базовых контейнеров:

- ◆ `container` — максимальная ширина контейнера 1140 px в Bootstrap 4 и 1320 px в Bootstrap 5.

Классы для адаптивной верстки: `container-sm`, `container-md`, `container-lg`, `container-xl`, `container-xxl` (Bootstrap 5);

- ◆ `container-fluid` — ширина контейнера всегда 100%.

В отличие от контейнера по умолчанию, для базовых контейнеров, помимо ширины, дополнительно прописываются значения внутренних и внешних отступов справа и слева. Стили в Bootstrap 4:

```
padding-right: 15px;
padding-left: 15px;
margin-right: auto;
margin-left: auto;
```

В Bootstrap 5 размеры внутренних отступов были изменены:

```
padding-right: 1rem;
padding-left: 1rem;
```

Пример:

```
<div class="container bg-warning">.container</div>
<div class="container-sm bg-success">.container-sm</div>
<div class="container-md bg-warning">.container-md</div>
<div class="container-lg bg-success">.container-lg</div>
<div class="container-xl bg-warning">.container-xl</div>
<div class="container-fluid bg-success">.container-fluid</div>
<div class="bg-warning">По умолчанию</div>
<!-- Bootstrap 5 -->
<div class="container-xxl bg-success">.container-xxl</div>
```

В этом примере классы `bg-warning` и `bg-success` задают цвет фона контейнера. Это сделано, чтобы можно было различить контейнеры визуально. Попробуйте в режиме **Адаптивный дизайн** изменить ширину экрана, переместив правую границу с помощью мыши, — ширина контейнеров будет подстраиваться под текущую ширину экрана. Ширина контейнеров при разной ширине экрана приведена в табл. 1.1.

Таблица 1.1. Ширина базовых контейнеров в зависимости от ширины экрана

	Extra small (es) <576px	Small (sm) >=576px	Medium (md) >=768px	Large (lg) >=992px	Extra large (xl) >=1200px	Extra extra large (xxl) >=1400px
.container	100%	540px	720px	960px	1140px	1320px
.container-sm	100%	540px	720px	960px	1140px	1320px
.container-md	100%	100%	720px	960px	1140px	1320px
.container-lg	100%	100%	100%	960px	1140px	1320px
.container-xl	100%	100%	100%	100%	1140px	1320px
.container-xxl	100%	100%	100%	100%	100%	1320px
.container-fluid	100%	100%	100%	100%	100%	100%

ПРИМЕЧАНИЕ

В Bootstrap 4 нет точки останова `xxl` и отсутствует стилевой класс `container-xxl`.

1.1.6. Цвет фона

Задать цвет фона контейнера или элемента позволяют следующие стилевые классы (аналоги CSS-атрибута `background-color`):

- ◆ `bg-white` — белый цвет;
- ◆ `bg-dark` — темно-серый цвет;
- ◆ `bg-light` — светло-серый цвет;
- ◆ `bg-success` — цвет успешно выполненной операции (зеленый);
- ◆ `bg-info` — цвет информационного сообщения;
- ◆ `bg-warning` — цвет предупреждающего сообщения (желтый);

- ◆ `bg-danger` — цвет опасности (красный);
- ◆ `bg-primary` — синий цвет;
- ◆ `bg-secondary` — серый цвет;
- ◆ `bg-transparent` — прозрачный фон;
- ◆ `bg-body` — цвет фона, указанный для элемента `BODY` (белый цвет в Bootstrap 5);
- ◆ `bg-gradient` — градиентный фон (Bootstrap 5):

```
.bg-gradient {
  background-image: var(--bs-gradient) !important;
}
```

Значение CSS-переменной `--bs-gradient` в Bootstrap 5:

```
--bs-gradient: linear-gradient(180deg, rgba(255, 255, 255, 0.15),
  rgba(255, 255, 255, 0));
```

Пример:

```
<div class="container bg-warning">
  <div class="bg-white border">.bg-white</div>
  <div class="bg-dark text-white">.bg-dark</div>
  <div class="bg-light">.bg-light</div>
  <div class="bg-success">.bg-success</div>
  <div class="bg-info">.bg-info</div>
  <div class="bg-warning">.bg-warning</div>
  <div class="bg-danger">.bg-danger</div>
  <div class="bg-primary">.bg-primary</div>
  <div class="bg-secondary text-white">.bg-secondary</div>
  <div class="bg-transparent border">.bg-transparent</div>
</div>
```

Стилевые классы, приведенные далее, в Bootstrap 4 можно указывать и для фона ссылок. В этом случае при наведении указателя мыши на ссылку, а также при получении фокуса, цвет фона ссылки станет темнее:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <p><a href="#" class="bg-dark text-white">.bg-dark</a></p>
  <p><a href="#" class="bg-light text-dark">.bg-light</a></p>
  <p><a href="#" class="bg-success text-white">.bg-success</a></p>
  <p><a href="#" class="bg-info text-white">.bg-info</a></p>
  <p><a href="#" class="bg-warning text-white">.bg-warning</a></p>
  <p><a href="#" class="bg-danger text-white">.bg-danger</a></p>
  <p><a href="#" class="bg-primary text-white">.bg-primary</a></p>
  <p><a href="#" class="bg-secondary text-white">.bg-secondary</a></p>
</div>
```

Стилевые классы, приведенные далее, в Bootstrap 4 можно указывать и для фона кнопок (тег `<button>`). В этом случае при наведении указателя мыши на кнопку, а также при получении фокуса цвет фона кнопки станет темнее:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
```

```

<button class="bg-dark text-white">.bg-dark</button><br>
<button class="bg-light text-dark">.bg-light</button><br>
<button class="bg-success text-white">.bg-success</button><br>
<button class="bg-info text-white">.bg-info</button><br>
<button class="bg-warning text-white">.bg-warning</button><br>
<button class="bg-danger text-white">.bg-danger</button><br>
<button class="bg-primary text-white">.bg-primary</button><br>
<button class="bg-secondary text-white">.bg-secondary</button>
</div>

```

Задать цвет можно также через следующие CSS-переменные в Bootstrap 4, которые следует передать функции `var()`:

```

--blue: #007bff;
--cyan: #17a2b8;
--gray: #6c757d;
--gray-dark: #343a40;
--green: #28a745;
--indigo: #6610f2;
--orange: #fd7e14;
--pink: #e83e8c;
--purple: #6f42c1;
--red: #dc3545;
--teal: #20c997;
--white: #fff;
--yellow: #ffc107;

--dark: #343a40;
--light: #f8f9fa;
--success: #28a745;
--info: #17a2b8;
--warning: #ffc107;
--danger: #dc3545;
--primary: #007bff;
--secondary: #6c757d;

```

Пример:

```

<p style="background-color: var(--red)">Текст абзаца1</p>
<p style="background-color: var(--success)">Текст абзаца2</p>

```

В Bootstrap 5 были изменены названия CSS-переменных и их значения:

```

--bs-blue: #0d6efd;
--bs-cyan: #17a2b8;
--bs-gray: #6c757d;
--bs-gray-dark: #343a40;
--bs-green: #28a745;
--bs-indigo: #6610f2;
--bs-orange: #fd7e14;
--bs-pink: #d63384;
--bs-purple: #6f42c1;
--bs-red: #dc3545;

```

```
--bs-teal: #20c997;
--bs-white: #fff;
--bs-yellow: #ffc107;
--bs-gradient: linear-gradient(180deg, rgba(255, 255, 255, 0.15),
  rgba(255, 255, 255, 0));

--bs-dark: #343a40;
--bs-light: #f8f9fa;
--bs-success: #28a745;
--bs-info: #17a2b8;
--bs-warning: #ffc107;
--bs-danger: #dc3545;
--bs-primary: #0d6efd;
--bs-secondary: #6c757d;
```

Пример:

```
<p style="background-color: var(--bs-red)">Текст абзаца1</p>
<p style="background-color: var(--bs-success)">Текст абзаца2</p>
<p style="background-image: var(--bs-gradient);
background-color: var(--bs-success)">Текст абзаца3</p>
```

1.2. Форматирование шрифта

В библиотеке Bootstrap 4 для элемента BODY задаются следующие характеристики шрифта и фона:

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
    "Helvetica Neue", Arial, "Noto Sans", sans-serif, "Apple Color Emoji",
    "Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji";
  font-size: 1rem;
  font-weight: 400;
  line-height: 1.5;
  color: #212529;
  text-align: left;
  background-color: #fff;
}
```

Стили в Bootstrap 5:

```
body {
  margin: 0;
  font-family: var(--bs-font-sans-serif);
  font-size: 1rem;
  font-weight: 400;
  line-height: 1.5;
  color: #212529;
  background-color: #fff;
  -webkit-text-size-adjust: 100%;
  -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
}
```

Значение переменной `--bs-font-sans-serif`:

```
--bs-font-sans-serif: system-ui, -apple-system, "Segoe UI", Roboto,
"Helvetica Neue", Arial, "Noto Sans", sans-serif, "Apple Color Emoji",
"Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji";
```

В Bootstrap 4 некоторые характеристики шрифта прописаны и для элемента HTML:

```
html {
  font-family: sans-serif;
  line-height: 1.15;
  -webkit-text-size-adjust: 100%;
  -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
}
```

В зависимости от элемента характеристики шрифта меняются.

1.2.1. Имя шрифта

В Bootstrap 4 с помощью стилевого класса `text-monospace` можно отобразить текст моноширинным шрифтом:

```
<!-- Только в Bootstrap 4 -->
<p>Слово1 <span class="text-monospace">слово2</span>
<code>слово3</code></p>
```

При использовании тега `<code>` текст также отображается моноширинным шрифтом, но меньшего размера и выделяется красным цветом.

В Bootstrap 5 следует воспользоваться стилевым классом `font-monospace`:

```
<!-- Только в Bootstrap 5 -->
<p>Слово1 <span class="font-monospace">слово2</span>
<code>слово3</code></p>
```

Списки названий шрифтов доступны также через следующие CSS-переменные в Bootstrap 4, которые можно передать функции `var()`:

- ◆ `--font-family-sans-serif` — обычный шрифт;
- ◆ `--font-family-monospace` — моноширинный шрифт.

Пример:

```
<!-- Только в Bootstrap 4 -->
<p style="font-family: var(--font-family-sans-serif)">
Текст абзаца1</p>
<p style="font-family: var(--font-family-monospace)">
Текст абзаца2</p>
```

В Bootstrap 5 используются следующие CSS-переменные:

- ◆ `--bs-font-sans-serif` — обычный шрифт;
- ◆ `--bs-font-monospace` — моноширинный шрифт.

Пример:

```
<!-- Только в Bootstrap 5 -->
<p style="font-family: var(--bs-font-sans-serif)">
Текст абзаца1</p>
<p style="font-family: var(--bs-font-monospace)">
Текст абзаца2</p>
```

1.2.2. Стиль шрифта

Отобразить текст курсивным шрифтом позволяет стилевой класс `font-italic` (аналог CSS-атрибута `font-style`). Пример:

```
<p class="font-italic">font-style: italic</p>
```

Можно также воспользоваться тегом `<i>` или тегом логического форматирования ``, с помощью которого делается акцент на выделенном фрагменте:

```
<p class="p-2">
  Обычный шрифт<br>
  <i>Текст, выделенный курсивом</i><br>
  <em>Текст, выделенный курсивом</em>
</p>
```

В Bootstrap 5 доступен стилевой класс `font-normal`, задающий нормальный стиль шрифта (`font-style: normal`):

```
<p class="font-normal">font-style: normal</p>
```

1.2.3. Размер шрифта

По умолчанию в библиотеке Bootstrap для элемента `BODY` задается размер шрифта равный `1 rem`. Как уже отмечалось ранее, в современных веб-браузерах относительная величина `1 rem` равна `16 px`. Этот размер наследуется вложенными элементами и переопределяется для некоторых элементов — например, для заголовков.

Для вывода текста шрифтом меньшего размера применяется тег `<small>`:

```
<p class="p-2">Текст <small>меньшего</small> размера</p>
```

Можно также воспользоваться стилевым классом `small`:

```
<p class="p-2">
  Текст <span class="small">меньшего</span> размера
</p>
```

Стиль для тега `<small>` в Bootstrap 4:

```
small, .small {
  font-size: 80%;
  font-weight: 400;
}
```

Стиль для тега `<small>` в Bootstrap 5:

```
small, .small {
  font-size: 0.875em;
}
```

Если добавить стилевой класс `lead`, то текст будет выделен более крупным шрифтом:

```
<div class="container">
  <p>Обычный абзац</p>
  <p class="lead">Выделенный абзац</p>
  <p>Текст <span class="lead">большого</span> размера</p>
</div>
```

1.2.4. Цвет текста

Задать цвет текста позволяют следующие стилевые классы (аналоги CSS-атрибута `color`):

- ◆ `text-body` — цвет текста, заданный для элемента `BODY`:
`<p class="text-body">.text-body color: #212529</p>`
- ◆ `text-white` — белый цвет:
`<p class="text-white bg-warning">color: #fff</p>`
- ◆ `text-white-50` — белый полупрозрачный цвет:
`<p class="text-white-50 bg-dark">color: rgba(255, 255, 255, 0.5)</p>`
- ◆ `text-black-50` — черный полупрозрачный цвет:
`<p class="text-black-50">color: rgba(0, 0, 0, 0.5)</p>`
- ◆ `text-dark` — темно-серый цвет:
`<p class="text-dark">.text-dark color: #343a40</p>`
- ◆ `text-muted` — серый цвет:
`<p class="text-muted">.text-muted color: #6c757d</p>`
- ◆ `text-light` — светло-серый цвет:
`<p class="text-light bg-dark">.text-light color: #f8f9fa</p>`
- ◆ `text-success` — цвет успешно выполненной операции (зеленый):
`<p class="text-success">.text-success color: #28a745</p>`
- ◆ `text-info` — цвет информационного сообщения:
`<p class="text-info">.text-info color: #17a2b8</p>`
- ◆ `text-warning` — цвет предупреждающего сообщения (желтый):
`<p class="text-warning">.text-warning color: #ffc107</p>`
- ◆ `text-danger` — цвет текста с сообщением об опасности (красный):
`<p class="text-danger">.text-danger color: #dc3545</p>`
- ◆ `text-primary` — синий цвет:
`<p class="text-primary">.text-primary color: #007bff`
или `#0d6efd</p>`
- ◆ `text-secondary` — серый цвет:
`<p class="text-secondary">.text-secondary color: #6c757d</p>`
- ◆ `text-reset` — цвет текста наследуется от родителя (`color: inherit`).

Стилевые классы, приведенные далее, в Bootstrap 4 можно указывать и для ссылок. В этом случае при наведении указателя мыши на ссылку, а также при получении фокуса цвет текста ссылки станет темнее:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <p><a href="#" class="text-dark">.text-dark</a></p>
  <p><a href="#" class="text-light bg-dark">.text-light</a></p>
  <p><a href="#" class="text-success">.text-success</a></p>
  <p><a href="#" class="text-info">.text-info</a></p>
  <p><a href="#" class="text-warning">.text-warning</a></p>
```

```
<p><a href="#" class="text-danger">.text-danger</a></p>
<p><a href="#" class="text-primary">.text-primary</a></p>
<p><a href="#" class="text-secondary">.text-secondary</a></p>
</div>
```

В Bootstrap 5 следует использовать классы семейства link-* (см. разд. 1.3.14):

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <p><a href="#" class="link-dark">.link-dark</a></p>
  <p><a href="#" class="link-light bg-dark">.link-light</a></p>
  <p><a href="#" class="link-success">.link-success</a></p>
  <p><a href="#" class="link-info">.link-info</a></p>
  <p><a href="#" class="link-warning">.link-warning</a></p>
  <p><a href="#" class="link-danger">.link-danger</a></p>
  <p><a href="#" class="link-primary">.link-primary</a></p>
  <p><a href="#" class="link-secondary">.link-secondary</a></p>
</div>
```

Задать цвет можно также через CSS-переменные (см. разд. 1.1.6), которые следует передать функции var():

```
<!-- Только в Bootstrap 4 -->
<p style="color: var(--red)">Текст абзаца1</p>
<p style="color: var(--success)">Текст абзаца2</p>
<!-- Только в Bootstrap 5 -->
<p style="color: var(--bs-red)">Текст абзаца1</p>
<p style="color: var(--bs-success)">Текст абзаца2</p>
```

Чтобы скрыть текст (сделать его прозрачным), в Bootstrap 4 можно воспользоваться стилевым классом text-hide:

```
<!-- Только в Bootstrap 4 -->
<p>Слово1 <span class="text-hide">слово2</span> слово3</p>
```

1.2.5. Жирность шрифта

Управлять жирностью шрифта позволяют следующие стилевые классы (аналоги CSS-атрибута font-weight):

- ◆ font-weight-light — соответствует значению 300;
- ◆ font-weight-normal — нормальный шрифт. Соответствует значению 400;
- ◆ font-weight-bold — полужирный шрифт. Соответствует значению 700;
- ◆ font-weight-lighter — менее жирный, чем у родительского элемента;
- ◆ font-weight-bolder — более жирный, чем у родительского элемента.

Пример:

```
<p class="font-weight-light">font-weight: 300</p>
<p class="font-weight-normal">font-weight: 400</p>
<p class="font-weight-bold">font-weight: 700</p>
<p class="font-weight-lighter">font-weight: lighter</p>
<p class="font-weight-bolder">font-weight: bolder</p>
```

Тег `` и тег логического форматирования ``, который определяет важный фрагмент текста, отображают текст полужирным шрифтом:

```
<p class="p-2">  
  Обычный шрифт<br>  
  <b>Полужирный шрифт</b><br>  
  <strong>Важный фрагмент текста</strong>  
</p>
```

Стиль для тегов `` и ``:

```
b, strong {  
  font-weight: bolder;  
}
```

1.2.6. Вертикальное расстояние между строками

Следующие стилевые классы в Bootstrap 5 задают вертикальное расстояние между базовыми линиями двух строк (аналоги атрибута `line-height`):

- ◆ `lh-1` — `line-height: 1;`
- ◆ `lh-sm` — `line-height: 1.25;`
- ◆ `lh-base` — `line-height: 1.5.` Это значение указано для элемента `body`;
- ◆ `lh-lg` — `line-height: 2.`

Пример:

```
<!-- Только в Bootstrap 5 -->  
<p class="lh-1">.lh-1 Строка 1<br>Строка 2</p>  
<p class="lh-sm">.lh-sm Строка 1<br>Строка 2</p>  
<p class="lh-base">.lh-base Строка 1<br>Строка 2</p>  
<p class="lh-lg">.lh-lg Строка 1<br>Строка 2</p>
```

1.3. Форматирование текста

Для текстовых фрагментов, кроме указания характеристик шрифтов, можно задать некоторые дополнительные параметры: вертикальное и горизонтальное выравнивание, способ изменения регистра символов и т. д.

1.3.1. Горизонтальное выравнивание текста

Задать горизонтальное выравнивание текста позволяют следующие стилевые классы (аналоги CSS-атрибута `text-align`):

- ◆ `text-center` — выравнивание по центру.
Классы для адаптивной верстки: `text-sm-center`, `text-md-center`, `text-lg-center`, `text-xl-center`, `text-xxl-center` (Bootstrap 5);
- ◆ `text-left` — выравнивание по левому краю.
Классы для адаптивной верстки: `text-sm-left`, `text-md-left`, `text-lg-left`, `text-xl-left`, `text-xxl-left` (Bootstrap 5);

- ◆ `text-right` — выравнивание по правому краю.

Классы для адаптивной верстки: `text-sm-right`, `text-md-right`, `text-lg-right`, `text-xl-right`, `text-xxl-right` (Bootstrap 5);

- ◆ `text-justify` — выравнивание по ширине (по двум сторонам). Класс доступен только в Bootstrap 4.

Пример:

```
<p class="text-center">Абзац с выравниванием по центру</p>
<p class="text-left">Абзац с выравниванием по левому краю</p>
<p class="text-right">Абзац с выравниванием по правому краю</p>
<p class="text-justify">Абзац с выравниванием по ширине</p>
```

1.3.2. Вертикальное выравнивание текста

Задать вертикальное выравнивание текста относительно элемента-родителя (например, ячейки таблицы) позволяют следующие стилевые классы (аналоги CSS-атрибута `vertical-align`):

- ◆ `align-baseline` — по базовой линии;
- ◆ `align-middle` — по центру;
- ◆ `align-top` — по верху;
- ◆ `align-bottom` — по низу.

Эти стили работают со строчными элементами и ячейками таблицы, поэтому если попробовать выполнить вертикальное выравнивание текста внутри элемента `DIV` или другого блочного элемента, то ничего не получится.

Со строчными элементами можно также использовать следующие стили:

- ◆ `align-text-top` — выравнивание по верху текстовой строки;
- ◆ `align-text-bottom` — выравнивание по низу текстовой строки.

Пример:

```
<p>

Строка
<span style="font-size: 0.6rem" class="align-text-top">
align-text-top</span>
<span style="font-size: 0.6rem" class="align-text-bottom">
align-text-bottom</span>
<span style="font-size: 0.6rem" class="align-baseline">
align-baseline</span>
<span style="font-size: 0.6rem" class="align-middle">
align-middle</span>
<span style="font-size: 0.6rem" class="align-top">
align-top</span>
<span style="font-size: 0.6rem" class="align-bottom">
align-bottom</span>
Строка</p>
```

1.3.3. Подчеркивание и зачеркивание текста

Тег `<u>` отображает текст подчеркнутым, а тег `<s>` — перечеркнутым:

```
<p class="p-2">
  <u>Подчеркнутый текст</u><br>
  <s>Перечеркнутый текст</s>
</p>
```

В Bootstrap 5 доступны следующие стилевые классы (аналоги CSS-атрибута `text-decoration`):

- ◆ `text-decoration-underline` — подчеркивает текст;
- ◆ `text-decoration-line-through` — зачеркивает текст.

Пример:

```
<!-- Только в Bootstrap 5 -->
<p class="text-decoration-underline">Подчеркнутый текст</p>
<p class="text-decoration-line-through">Перечеркнутый текст</p>
```

С помощью стилевого класса `text-decoration-none` можно отменить подчеркивание, надчеркивание или зачеркивание текста (аналог CSS-атрибута `text-decoration` со значением `none`):

```
<p class="p-2">
  <u class="text-decoration-none">Не подчеркнутый текст</u><br>
  <s class="text-decoration-none">Не перечеркнутый текст</s>
</p>
```

Отметить фрагмент текста как удаленный позволяет тег ``. Текст отображается перечеркнутым. Текст, вставленный вместо удаленного, отмечается тегом `<ins>`. Текст при этом отображается подчеркнутым. Пример:

```
<p class="p-2">
  <del>Старый текст</del> <ins>новый текст</ins>
</p>
```

1.3.4. Изменение регистра символов

Для изменения регистра символов предназначены следующие стилевые классы (аналоги CSS-атрибута `text-transform`):

- ◆ `text-capitalize` — делает первую букву каждого слова прописной;
- ◆ `text-uppercase` — преобразует все буквы в прописные;
- ◆ `text-lowercase` — преобразует все буквы в строчные.

Пример:

```
<h1 class="text-capitalize">заголовок из нескольких слов</h1>
<h1 class="text-uppercase">заголовок2</h1>
<h1 class="text-lowercase">ЗАГОЛОВОК3</h1>
```

Результат:

```
Заголовок Из Несколько Слов
ЗАГОЛОВОК2
заголовок3
```

1.3.5. Обработка переноса строк

Следующие стилевые классы позволяют задать способ переноса строк:

- ◆ `text-wrap` — текст выводится стандартным образом (по умолчанию несколько пробелов подряд выводятся в окне веб-браузера как один пробел):

```
<p class="text-wrap border" style="width: 150px">
Строка      1
Строка 2<br>
Строка 3<br>
текст оченьдлинныйтекст<br>
текст оченьоченьоченьдлинныйтекст
</p>
```

Результат в окне Web-браузера (последняя строка выходит за пределы блока):

```
Строка 1 Строка 2
Строка 3
текст
оченьдлинныйтекст
текст
оченьоченьоченьдлинныйтекст
```

Чтобы последняя строка не выходила за пределы блока, а переносилась на новую строку, следует добавить стилевой класс `text-break`:

```
<p class="text-wrap text-break border" style="width: 150px">
Строка      1
Строка 2<br>
Строка 3<br>
текст оченьдлинныйтекст<br>
текст оченьоченьоченьдлинныйтекст
</p>
```

Результат в окне веб-браузера:

```
Строка 1 Строка 2
Строка 3
текст
оченьдлинныйтекст
текст
оченьоченьоченьдл
инныйтекст
```

- ◆ `text-nowrap` — переносы строк в HTML-коде игнорируются. Если внутри строки содержится тег `
`, то он вставляет перенос строки. Текст автоматически на новую строку не переносится. Пример:

```
<p class="text-nowrap border" style="width: 150px">
Строка      1
Строка 2<br>
Строка 3<br>
текст оченьдлинныйтекст<br>
текст оченьоченьоченьдлинныйтекст
</p>
```

Результат в окне веб-браузера (две последние строки выходят за пределы блока):

Строка 1 **Строка 2**

Строка 3

текст оченьдлинныйтекст

текст оченьоченьоченьдлинныйтекст

- ◆ `text-truncate` — аналог `text-nowrap`, но дополнительно выполняется обрезка строки, если она не помещается в пределы блока, и вставляется троеточие. Пример:

```
<p class="text-truncate border" style="width: 150px">
Строка      1
Строка 2<br>
Строка 3<br>
текст оченьдлинныйтекст<br>
текст оченьоченьоченьдлинныйтекст
</p>
```

Результат в окне Web-браузера (две последние строки обрезаются):

Строка 1 **Строка 2**

Строка 3

текст оченьдлинн...

текст оченьоченьо...

При использовании тега `<pre>` (подробнее о нем рассказано в *разд. 1.3.7*) сохраняются все пробелы и переносы строк. Текст выводится моноширинным шрифтом. Если текст не помещается внутри блока, то автоматически будут добавлены полосы прокрутки.

1.3.6. Создание нижних и верхних индексов

Тег `<sub>` сдвигает текст ниже уровня строки и уменьшает размер шрифта. Он используется для создания нижних индексов. Тег `<sup>` сдвигает текст выше уровня строки и уменьшает размер шрифта. Этот тег используется чаще всего для создания степеней. Пример:

```
<p class="p-2">
  Формула воды — H<sub>2</sub>O<br>
  Единица измерения площади — м<sup>2</sup>
</p>
```

Стили для этих тегов:

```
sub, sup {
  position: relative;
  font-size: 75%;      /* В Bootstrap 5: font-size: 0.75em; */
  line-height: 0;
  vertical-align: baseline;
}
```

```
sub {
  bottom: -.25em;
}
sup {
  top: -.5em;
}
```

1.3.7. Выделение фрагментов кода

Для вывода текста в том же виде, что и в исходном коде, можно воспользоваться парным блочным тегом `<pre>`:

```
<pre class="p-2 bg-light">
Пункт1
  Пункт1_1
  Пункт1_2
</pre>
```

При использовании тега `<pre>` сохраняются все пробелы и переносы строк. Текст выводится моноширинным шрифтом. Если текст не помещается внутри блока, то автоматически будут добавлены полосы прокрутки:

```
<pre class="border p-2 bg-light" style="width: 150px">
Строка      1
Строка 2<br>
Строка 3
текст оченьдлинныйтекст
текст оченьоченьоченьдлинныйтекст</pre>
```

Результат в окне веб-браузера (отображается горизонтальная полоса прокрутки):

Строка 1

Строка 2

Строка 3

текст оченьдлинныйтекст

текст оченьоченьоченьдлинныйтекст

Если нужно ограничить высоту контейнера, то в Bootstrap 4 можно добавить стиливой класс `pre-scrollable`:

```
.pre-scrollable { /* Только в Bootstrap 4 */
  max-height: 340px;
  overflow-y: scroll;
}
```

Если высота контейнера будет больше 340 px, то станет доступной вертикальная полоса прокрутки. Пример указания класса в Bootstrap 4:

```
<pre class="border p-2 bg-light pre-scrollable">
Пункт1
  Пункт1_1
  Пункт1_2
</pre>
```

Стили для тега `<pre>`:

```
pre {
  display: block;
  font-size: 87.5%; /* В Bootstrap 5: font-size: 0.875em; */
  color: #212529; /* В Bootstrap 5 наследуется от body */
  margin-top: 0;
  margin-bottom: 1rem;
  overflow: auto;
  font-family: SFMono-Regular, Menlo, Monaco, Consolas,
  "Liberation Mono", "Courier New", monospace;
  /* В Bootstrap 5: font-family: var(--bs-font-monospace); */
}
pre code {
  font-size: inherit;
  color: inherit;
  word-break: normal;
}
```

Для выделения фрагментов кода можно воспользоваться следующими тегами логического форматирования:

◆ `<code>` — служит для отметки фрагментов программного кода. Стили:

```
code {
  font-family: SFMono-Regular, Menlo, Monaco, Consolas,
  "Liberation Mono", "Courier New", monospace;
  /* В Bootstrap 5: font-family: var(--bs-font-monospace); */
  font-size: 87.5%; /* В Bootstrap 5: font-size: 0.875em; */
  color: #e83e8c; /* В Bootstrap 5: color: #d63384; */
  word-wrap: break-word;
}
a > code {
  color: inherit;
}
```

Пример:

```
<p class="p-2">Библиотека <code>Bootstrap</code></p>
```

◆ `<samp>` — применяется для отметки результата, выдаваемого программой. Текст выделяется моноширинным шрифтом:

```
<p class="p-2">Результат: <samp>x = 10</samp></p>
```

◆ `<kbd>` — отмечает фрагмент как вводимый пользователем с клавиатуры. Стили:

```
kbd {
  font-family: SFMono-Regular, Menlo, Monaco, Consolas,
  "Liberation Mono", "Courier New", monospace;
  /* В Bootstrap 5: font-family: var(--bs-font-monospace); */
  padding: 0.2rem 0.4rem;
  font-size: 87.5%; /* В Bootstrap 5: font-size: 0.875em; */
  color: #fff;
  background-color: #212529;
```

```
border-radius: 0.2rem;
}
kbd kbd {
padding: 0;
font-size: 100%; /* В Bootstrap 5: font-size: 1em; */
font-weight: 700;
}
```

Текст отображается белым цветом на темном фоне. Пример:

```
<p class="p-2">Выполняем команду <kbd>cd C:\book</kbd></p>
```

◆ `<var>` — отмечает имена переменных:

```
<p class="p-2">Переменная <var>title</var></p>
```

1.3.8. Выделение важного фрагмента текста и аббревиатуры

Тег `<mark>` и стилевой класс `mark` помечают важный фрагмент текста:

```
<p class="p-2">
  <mark>Важный фрагмент</mark> текста<br><br>
  <span class="mark">Важный фрагмент</span> текста
</p>
```

Для такого фрагмента устанавливается желтый цвет фона (фрагмент текста как бы выделяется маркером). Стили:

```
mark, .mark {
padding: 0.2em;
background-color: #fcf8e3;
}
```

Тег `<abbr>` используется для отметки аббревиатур. Расшифровка аббревиатуры задается в параметре `title`. Текст подчеркивается пунктирной линией. При наведении указателя мыши отображается всплывающая подсказка с расшифровкой аббревиатуры. Пример:

```
<p class="p-2">
  <abbr title="Hypertext Transfer Protocol">HTTP</abbr>
</p>
```

Чтобы аббревиатура отображалась уменьшенным шрифтом и всегда в верхнем регистре нужно добавить стилевой класс `initialism`:

```
<p class="p-2">
  <abbr title="Hypertext Transfer Protocol"
    class="initialism">http</abbr> - это протокол
    передачи данных
</p>
```

Стили в Bootstrap 4:

```
.initialism {
font-size: 90%;
text-transform: uppercase;
}
```

Стили в Bootstrap 5:

```
.initialism {
  font-size: 0.875em;
  text-transform: uppercase;
}
```

1.3.9. Выделение цитат

Парный тег `<blockquote>` создает абзац с длинной цитатой. Пример:

```
<blockquote class="bg-light">Длинная цитата</blockquote>
```

Стиль:

```
blockquote {
  margin: 0 0 1rem;
}
```

Можно дополнительно воспользоваться стилевыми классами `blockquote` и `blockquote-footer` (подпись под цитатой — например, источник цитаты):

```
<blockquote class="blockquote bg-light">
  <p class="mb-0">Длинная цитата</p>
  <footer class="blockquote-footer">.blockquote-footer</footer>
</blockquote>
```

По умолчанию производится выравнивание по левому краю. Для выравнивания по центру или по правому краю нужно добавить стилевые классы `text-center` или `text-right` соответственно:

```
<blockquote class="blockquote bg-light text-center">
  <p class="mb-0">Длинная цитата с выравниванием по центру</p>
  <footer class="blockquote-footer">.blockquote-footer</footer>
</blockquote>
```

В Bootstrap 5 рекомендуется такая разметка для цитат:

```
<figure class="bg-light text-center">
  <blockquote class="blockquote">
    <p>Длинная цитата с выравниванием по центру</p>
  </blockquote>
  <figcaption class="blockquote-footer">
    .blockquote-footer
  </figcaption>
</figure>
```

Для цитат внутри текста используются следующие теги:

- ◆ `<cite>` — применяется для отметки цитат, а также названий произведений и сносок. Фрагмент выделяется курсивом;
- ◆ `<q>` — используется для отметки коротких цитат. Фрагмент отображается в кавычках.

Пример:

```
<p class="p-2"><cite>цитата</cite> <q>цитата</q></p>
```

1.3.10. Заголовки

Заголовки могут иметь шесть различных размеров:

```
<hx>Заголовок</hx>
```

где x — число от 1 до 6. Заголовок с номером 1 является самым крупным, а заголовок с номером 6 — самым мелким:

```
<h1>Заголовок h1</h1>
<h2>Заголовок h2</h2>
<h3>Заголовок h3</h3>
<h4>Заголовок h4</h4>
<h5>Заголовок h5</h5>
<h6>Заголовок h6</h6>
```

Можно также воспользоваться одноименными стилевыми классами:

```
<p class="h1">Заголовок h1</p>
<p class="h2">Заголовок h2</p>
<p class="h3">Заголовок h3</p>
<p class="h4">Заголовок h4</p>
<p class="h5">Заголовок h5</p>
<p class="h6">Заголовок h6</p>
```

Для отображения заголовка более крупным шрифтом следует воспользоваться стилевыми классами `display-1`, `display-2`, `display-3` или `display-4`:

```
<h1 class="display-1">.display-1</h1>
<h1 class="display-2">.display-2</h1>
<h1 class="display-3">.display-3</h1>
<h1 class="display-4">.display-4</h1>
<h1>Заголовок h1</h1>
```

В Bootstrap 5 дополнительно доступны стилевые классы `display-5` и `display-6`:

```
<!-- Только в Bootstrap 5 -->
<h1 class="display-5">.display-5</h1>
<h1 class="display-6">.display-6</h1>
<h1>Заголовок h1</h1>
```

Чтобы создать подзаголовок, можно воспользоваться тегом `<small>` совместно со стилевым классом, задающим цвет текста:

```
<h1>Заголовок h1
  <small class="text-muted">дополнительный заголовок</small>
</h1>
```

По умолчанию производится выравнивание заголовка по левому краю. Для выравнивания заголовка по центру или по правому краю нужно добавить стилевые классы `text-center` или `text-right` соответственно:

```
<h1 class="text-center">Заголовок по центру</h1>
```

1.3.11. Разделение на абзацы

Тег `<p>` позволяет разделить текст на отдельные абзацы. При этом после абзаца добавляется пустое пространство:

```
<div class="container">
  <h1>Заголовок</h1>
```

```
<p>Абзац с выравниванием по левому краю</p>
<p class="text-center">Абзац с выравниванием по центру</p>
<p class="text-left">Абзац с выравниванием по левому краю</p>
<p class="text-right">Абзац с выравниванием по правому краю</p>
<p class="text-justify">Абзац с выравниванием по ширине</p>
</div>
```

Если добавить стилевой класс `lead`, то текст абзаца будет выделен более крупным шрифтом, а если `small` — то уменьшенным шрифтом:

```
<div class="container">
  <p>Обычный абзац</p>
  <p class="small">Абзац с уменьшенным шрифтом</p>
  <p class="lead">Выделенный абзац</p>
</div>
```

Для абзацев в библиотеке Bootstrap прописаны следующие правила:

```
p {
  margin-top: 0;
  margin-bottom: 1rem;
}
```

1.3.12. Тег `<details>`

Тег `<details>` позволяет скрыть или отобразить какой-либо фрагмент страницы. По умолчанию содержимое элемента скрыто. Чтобы отобразить это содержимое, нужно щелкнуть левой кнопкой мыши на заголовке, реализуемом с помощью тега `<summary>`, или добавить параметр `open`. Чтобы опять скрыть содержимое, нужно повторно щелкнуть мышью на заголовке. Пример:

```
<details open>
  <summary>Развернуть или свернуть</summary>
  <p class="bg-success">Скрытый текст</p>
</details>
```

Стили для тега `<summary>`:

```
summary {
  display: list-item;
  cursor: pointer;
}
```

1.3.13. Горизонтальная линия

Одинарный тег `<hr>` позволяет провести горизонтальную линию. По умолчанию линия занимает всю ширину родительского элемента и выравнивается по центру. Пример:

```
<div class="container">
  <p>Со значениями по умолчанию</p>
  <hr>
  <p>Зеленый цвет фона</p>
  <hr class="bg-success">
  <!-- Только в Bootstrap 4 -->
```

```

    <p>Граница синего цвета</p>
    <hr class="border-primary">
</div>

```

Стили в Bootstrap 4:

```

hr {
  box-sizing: content-box;
  height: 0;
  overflow: visible;
  margin-top: 1rem;
  margin-bottom: 1rem;
  border: 0;
  border-top: 1px solid rgba(0, 0, 0, 0.1);
}

```

Стили в Bootstrap 5:

```

hr {
  margin: 1rem 0;
  color: inherit;
  background-color: currentColor;
  border: 0;
  opacity: 0.25;
}
hr:not([size]) {
  height: 1px;
}

```

1.3.14. Гиперссылки

Для гиперссылок в библиотеке Bootstrap 4 прописаны следующие правила:

```

a {
  color: #007bff;
  text-decoration: none;
  background-color: transparent;
}
a:hover {
  color: #0056b3;
  text-decoration: underline;
}
a:not([href]):not([class]) {
  color: inherit;
  text-decoration: none;
}
a:not([href]):not([class]):hover {
  color: inherit;
  text-decoration: none;
}
a > code {
  color: inherit;
}

```

В Bootstrap 5 правила изменены — теперь ссылки всегда подчеркнуты:

```
a {
  color: #0d6efd;
  text-decoration: underline;
}
a:hover {
  color: #024dbc;
}
a:not([href]):not([class]), a:not([href]):not([class]):hover {
  color: inherit;
  text-decoration: none;
}
a > code {
  color: inherit;
}
```

Пример:

```
<div class="container">
  <p>Текст абзаца <a href="#">текст ссылки</a></p>
  <p>Текст абзаца <a>текст ссылки (нет параметра href)</a></p>
  <p>Текст абзаца <a href="#" target="_blank">ссылка</a></p>
</div>
```

Стилевые классы, приведенные далее, в Bootstrap 4 можно указывать для ссылок. В этом случае при наведении указателя мыши на ссылку, а также при получении фокуса, цвет текста ссылки станет темнее:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <p><a href="#" class="text-dark">.text-dark</a></p>
  <p><a href="#" class="text-light bg-dark">.text-light</a></p>
  <p><a href="#" class="text-success">.text-success</a></p>
  <p><a href="#" class="text-info">.text-info</a></p>
  <p><a href="#" class="text-warning">.text-warning</a></p>
  <p><a href="#" class="text-danger">.text-danger</a></p>
  <p><a href="#" class="text-primary">.text-primary</a></p>
  <p><a href="#" class="text-secondary">.text-secondary</a></p>
</div>
```

В Bootstrap 5 для изменения цвета ссылок следует использовать следующие классы (при наведении указателя мыши на ссылку, а также при получении фокуса, цвет текста ссылки станет темнее):

- ◆ `link-dark` — темно-серый цвет;
- ◆ `link-light` — светло-серый цвет;
- ◆ `link-success` — цвет успешно выполненной операции (зеленый);
- ◆ `link-info` — цвет информационного сообщения;
- ◆ `link-warning` — цвет предупреждающего сообщения (желтый);
- ◆ `link-danger` — цвет текста с сообщением об опасности (красный);

- ◆ link-primary — синий цвет;
- ◆ link-secondary — серый цвет.

Пример:

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <p><a href="#" class="link-dark">.link-dark</a></p>
  <p><a href="#" class="link-light bg-dark">.link-light</a></p>
  <p><a href="#" class="link-success">.link-success</a></p>
  <p><a href="#" class="link-info">.link-info</a></p>
  <p><a href="#" class="link-warning">.link-warning</a></p>
  <p><a href="#" class="link-danger">.link-danger</a></p>
  <p><a href="#" class="link-primary">.link-primary</a></p>
  <p><a href="#" class="link-secondary">.link-secondary</a></p>
</div>
```

Стилевые классы, приведенные далее, в Bootstrap 4 можно указывать для фона ссылок. В этом случае при наведении указателя мыши на ссылку, а также при получении фокуса, цвет фона ссылки станет темнее:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <p><a href="#" class="bg-dark text-white">.bg-dark</a></p>
  <p><a href="#" class="bg-light text-dark">.bg-light</a></p>
  <p><a href="#" class="bg-success text-white">.bg-success</a></p>
  <p><a href="#" class="bg-info text-white">.bg-info</a></p>
  <p><a href="#" class="bg-warning text-white">.bg-warning</a></p>
  <p><a href="#" class="bg-danger text-white">.bg-danger</a></p>
  <p><a href="#" class="bg-primary text-white">.bg-primary</a></p>
  <p><a href="#" class="bg-secondary text-white">.bg-secondary</a></p>
</div>
```

По умолчанию ссылкой является только содержимое между тегами `<a>` и `` — например, текст или изображение. Если нужно, чтобы ссылка занимала всю ширину родительского элемента, то следует элемент `a` сделать блочным, добавив стилевой класс `d-block`. Если же нужно, чтобы ссылка занимала вообще весь блок, включая все его содержимое, то для родительского блока указываем относительное позиционирование (стилевой класс `position-relative`), а к ссылке добавляем стилевой класс `stretched-link`. В результате к ссылке добавляется элемент с абсолютным позиционированием, который занимает всю область блока:

```
.stretched-link::after {
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 1;
  pointer-events: auto; /* Только в Bootstrap 4 */
  content: "";
  background-color: rgba(0, 0, 0, 0); /* Только в Bootstrap 4 */
}
```

Пример:

```

<div class="container">
  <div class="bg-light p-2" style="width: 400px">
    <p>Текст абзаца ссылкой не является</p>
    <a href="link1.html">Обычная ссылка</a>
  </div>
  <div class="position-relative bg-warning p-2 my-3"
    style="width: 400px">
    <p>Текст абзаца ссылкой не является</p>
    <a href="link2.html" class="d-block text-white">
      Ссылка занимает всю ширину блока</a>
  </div>
  <div class="position-relative bg-success p-2"
    style="width: 400px">
    <p>Текст абзаца является ссылкой</p>
    <a href="link3.html" class="stretched-link text-white">
      Ссылка занимает весь блок</a>
  </div>
</div>

```

1.4. Отступы

Любой элемент веб-страницы занимает в окне веб-браузера некоторую прямоугольную область. Причем эта область имеет как внутренние, так и внешние отступы. *Внутренний отступ* — это расстояние между элементом страницы и реальной или воображаемой границей области. *Внешний отступ* — это расстояние между реальной или воображаемой границей и другим элементом веб-страницы, точнее сказать, между границей и крайней точкой внешнего отступа другого элемента веб-страницы.

По умолчанию нижний внешний отступ одного блочного элемента может объединяться с верхним внешним отступом другого блочного элемента, т. е. мы получим не сумму внешних отступов, а наибольшее значение. Этот эффект можно наблюдать при организации абзацев. Если объединения не будет, то отступ между абзацами увеличится в два раза. Справа и слева внешние отступы никогда не объединяются.

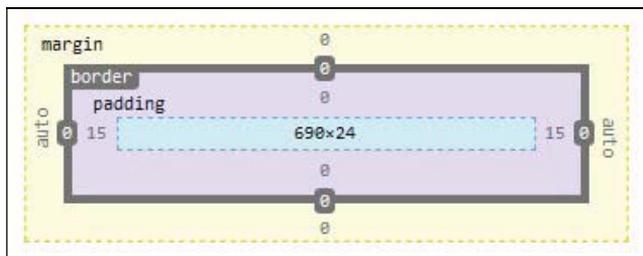


Рис. 1.7. Структура блочной модели

Чтобы увидеть структуру блочной модели, воспользуемся инструментами разработчика веб-браузера Firefox. Открываем вкладку **Инспектор** и справа переходим на вкладку **Разметка** (см. рис. 1.4). Как можно видеть на рис. 1.7, вначале идут размеры элемента, затем внутренние отступы (`padding`), далее граница (`border`) и внешние отступы (`margin`).

1.4.1. Внешние отступы

Задать размер внешних отступов позволяют следующие стилевые классы:

- ◆ `m-auto` — `margin: auto`.

Классы для адаптивной верстки: `m-sm-auto`, `m-md-auto`, `m-lg-auto`, `m-xl-auto`, `m-xxl-auto` (Bootstrap 5);

- ◆ `mt-auto` — `margin-top: auto`.

Классы для адаптивной верстки: `mt-sm-auto`, `mt-md-auto`, `mt-lg-auto`, `mt-xl-auto`, `mt-xxl-auto` (Bootstrap 5);

- ◆ `mb-auto` — `margin-bottom: auto`.

Классы для адаптивной верстки: `mb-sm-auto`, `mb-md-auto`, `mb-lg-auto`, `mb-xl-auto`, `mb-xxl-auto` (Bootstrap 5);

- ◆ `ml-auto` — `margin-left: auto`.

Классы для адаптивной верстки: `ml-sm-auto`, `ml-md-auto`, `ml-lg-auto`, `ml-xl-auto`, `ml-xxl-auto` (Bootstrap 5);

- ◆ `mr-auto` — `margin-right: auto`.

Классы для адаптивной верстки: `mr-sm-auto`, `mr-md-auto`, `mr-lg-auto`, `mr-xl-auto`, `mr-xxl-auto` (Bootstrap 5).

Пример горизонтального выравнивания блоков по правой и левой сторонам:

```
<div class="border w-25 ml-auto">.ml-auto</div>
<div class="border w-25 mr-auto">.mr-auto</div>
```

- ◆ `mx-auto` — `margin-left: auto; margin-right: auto`.

Классы для адаптивной верстки: `mx-sm-auto`, `mx-md-auto`, `mx-lg-auto`, `mx-xl-auto`, `mx-xxl-auto` (Bootstrap 5).

Пример горизонтального выравнивания блока по центру:

```
<div class="border w-25 mx-auto">.mx-auto</div>
```

- ◆ `my-auto` — `margin-top: auto; margin-bottom: auto`.

Классы для адаптивной верстки: `my-sm-auto`, `my-md-auto`, `my-lg-auto`, `my-xl-auto`, `my-xxl-auto` (Bootstrap 5);

- ◆ `m-0` — `margin: 0`.

Классы для адаптивной верстки: `m-sm-0`, `m-md-0`, `m-lg-0`, `m-xl-0`, `m-xxl-0` (Bootstrap 5);

- ◆ m-1 — margin: 0.25rem.

Классы для адаптивной верстки: m-sm-1, m-md-1, m-lg-1, m-xl-1, m-xxl-1 (Bootstrap 5);

- ◆ m-2 — margin: 0.5rem.

Классы для адаптивной верстки: m-sm-2, m-md-2, m-lg-2, m-xl-2, m-xxl-2 (Bootstrap 5);

- ◆ m-3 — margin: 1rem.

Классы для адаптивной верстки: m-sm-3, m-md-3, m-lg-3, m-xl-3, m-xxl-3 (Bootstrap 5);

- ◆ m-4 — margin: 1.5rem.

Классы для адаптивной верстки: m-sm-4, m-md-4, m-lg-4, m-xl-4, m-xxl-4 (Bootstrap 5);

- ◆ m-5 — margin: 3rem.

Классы для адаптивной верстки: m-sm-5, m-md-5, m-lg-5, m-xl-5, m-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border m-0">.m-0</div>  
<div class="border m-1">.m-1</div>  
<div class="border m-2">.m-2</div>  
<div class="border m-3">.m-3</div>  
<div class="border m-4">.m-4</div>  
<div class="border m-5">.m-5</div>
```

- ◆ mt-0 — margin-top: 0.

Классы для адаптивной верстки: mt-sm-0, mt-md-0, mt-lg-0, mt-xl-0, mt-xxl-0 (Bootstrap 5);

- ◆ mt-1 — margin-top: 0.25rem.

Классы для адаптивной верстки: mt-sm-1, mt-md-1, mt-lg-1, mt-xl-1, mt-xxl-1 (Bootstrap 5);

- ◆ mt-2 — margin-top: 0.5rem.

Классы для адаптивной верстки: mt-sm-2, mt-md-2, mt-lg-2, mt-xl-2, mt-xxl-2 (Bootstrap 5);

- ◆ mt-3 — margin-top: 1rem.

Классы для адаптивной верстки: mt-sm-3, mt-md-3, mt-lg-3, mt-xl-3, mt-xxl-3 (Bootstrap 5);

- ◆ mt-4 — margin-top: 1.5rem.

Классы для адаптивной верстки: mt-sm-4, mt-md-4, mt-lg-4, mt-xl-4, mt-xxl-4 (Bootstrap 5);

- ◆ mt-5 — margin-top: 3rem.

Классы для адаптивной верстки: mt-sm-5, mt-md-5, mt-lg-5, mt-xl-5, mt-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border mt-0">.mt-0</div>
<div class="border mt-1">.mt-1</div>
<div class="border mt-2">.mt-2</div>
<div class="border mt-3">.mt-3</div>
<div class="border mt-4">.mt-4</div>
<div class="border mt-5">.mt-5</div>
```

- ◆ mb-0 — margin-bottom: 0.

Классы для адаптивной верстки: mb-sm-0, mb-md-0, mb-lg-0, mb-xl-0, mb-xxl-0 (Bootstrap 5);

- ◆ mb-1 — margin-bottom: 0.25rem.

Классы для адаптивной верстки: mb-sm-1, mb-md-1, mb-lg-1, mb-xl-1, mb-xxl-1 (Bootstrap 5);

- ◆ mb-2 — margin-bottom: 0.5rem.

Классы для адаптивной верстки: mb-sm-2, mb-md-2, mb-lg-2, mb-xl-2, mb-xxl-2 (Bootstrap 5);

- ◆ mb-3 — margin-bottom: 1rem.

Классы для адаптивной верстки: mb-sm-3, mb-md-3, mb-lg-3, mb-xl-3, mb-xxl-3 (Bootstrap 5);

- ◆ mb-4 — margin-bottom: 1.5rem.

Классы для адаптивной верстки: mb-sm-4, mb-md-4, mb-lg-4, mb-xl-4, mb-xxl-4 (Bootstrap 5);

- ◆ mb-5 — margin-bottom: 3rem.

Классы для адаптивной верстки: mb-sm-5, mb-md-5, mb-lg-5, mb-xl-5, mb-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border mb-0">.mb-0</div>
<div class="border mb-1">.mb-1</div>
<div class="border mb-2">.mb-2</div>
<div class="border mb-3">.mb-3</div>
<div class="border mb-4">.mb-4</div>
<div class="border mb-5">.mb-5</div>
```

- ◆ ml-0 — margin-left: 0.

Классы для адаптивной верстки: ml-sm-0, ml-md-0, ml-lg-0, ml-xl-0, ml-xxl-0 (Bootstrap 5);

- ◆ ml-1 — margin-left: 0.25rem.

Классы для адаптивной верстки: ml-sm-1, ml-md-1, ml-lg-1, ml-xl-1, ml-xxl-1 (Bootstrap 5);

- ◆ ml-2 — margin-left: 0.5rem.

Классы для адаптивной верстки: ml-sm-2, ml-md-2, ml-lg-2, ml-xl-2, ml-xxl-2 (Bootstrap 5);

- ◆ ml-3 — margin-left: 1rem.

Классы для адаптивной верстки: ml-sm-3, ml-md-3, ml-lg-3, ml-xl-3, ml-xxl-3 (Bootstrap 5);

- ◆ ml-4 — margin-left: 1.5rem.

Классы для адаптивной верстки: ml-sm-4, ml-md-4, ml-lg-4, ml-xl-4, ml-xxl-4 (Bootstrap 5);

- ◆ ml-5 — margin-left: 3rem.

Классы для адаптивной верстки: ml-sm-5, ml-md-5, ml-lg-5, ml-xl-5, ml-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border ml-0">.ml-0</div>
<div class="border ml-1">.ml-1</div>
<div class="border ml-2">.ml-2</div>
<div class="border ml-3">.ml-3</div>
<div class="border ml-4">.ml-4</div>
<div class="border ml-5">.ml-5</div>
```

- ◆ mr-0 — margin-right: 0.

Классы для адаптивной верстки: mr-sm-0, mr-md-0, mr-lg-0, mr-xl-0, mr-xxl-0 (Bootstrap 5);

- ◆ mr-1 — margin-right: 0.25rem.

Классы для адаптивной верстки: mr-sm-1, mr-md-1, mr-lg-1, mr-xl-1, mr-xxl-1 (Bootstrap 5);

- ◆ mr-2 — margin-right: 0.5rem.

Классы для адаптивной верстки: mr-sm-2, mr-md-2, mr-lg-2, mr-xl-2, mr-xxl-2 (Bootstrap 5);

- ◆ mr-3 — margin-right: 1rem.

Классы для адаптивной верстки: mr-sm-3, mr-md-3, mr-lg-3, mr-xl-3, mr-xxl-3 (Bootstrap 5);

- ◆ mr-4 — margin-right: 1.5rem.

Классы для адаптивной верстки: mr-sm-4, mr-md-4, mr-lg-4, mr-xl-4, mr-xxl-4 (Bootstrap 5);

- ◆ mr-5 — margin-right: 3rem.

Классы для адаптивной верстки: mr-sm-5, mr-md-5, mr-lg-5, mr-xl-5, mr-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border mr-0">.mr-0</div>
<div class="border mr-1">.mr-1</div>
<div class="border mr-2">.mr-2</div>
<div class="border mr-3">.mr-3</div>
<div class="border mr-4">.mr-4</div>
<div class="border mr-5">.mr-5</div>
```

- ◆ `mx-0` — `margin-left: 0; margin-right: 0.`

Классы для адаптивной верстки: `mx-sm-0`, `mx-md-0`, `mx-lg-0`, `mx-xl-0`, `mx-xxl-0` (Bootstrap 5);

- ◆ `mx-1` — `margin-left: 0.25rem; margin-right: 0.25rem.`

Классы для адаптивной верстки: `mx-sm-1`, `mx-md-1`, `mx-lg-1`, `mx-xl-1`, `mx-xxl-1` (Bootstrap 5);

- ◆ `mx-2` — `margin-left: 0.5rem; margin-right: 0.5rem.`

Классы для адаптивной верстки: `mx-sm-2`, `mx-md-2`, `mx-lg-2`, `mx-xl-2`, `mx-xxl-2` (Bootstrap 5);

- ◆ `mx-3` — `margin-left: 1rem; margin-right: 1rem.`

Классы для адаптивной верстки: `mx-sm-3`, `mx-md-3`, `mx-lg-3`, `mx-xl-3`, `mx-xxl-3` (Bootstrap 5);

- ◆ `mx-4` — `margin-left: 1.5rem; margin-right: 1.5rem.`

Классы для адаптивной верстки: `mx-sm-4`, `mx-md-4`, `mx-lg-4`, `mx-xl-4`, `mx-xxl-4` (Bootstrap 5);

- ◆ `mx-5` — `margin-left: 3rem; margin-right: 3rem.`

Классы для адаптивной верстки: `mx-sm-5`, `mx-md-5`, `mx-lg-5`, `mx-xl-5`, `mx-xxl-5` (Bootstrap 5).

Пример:

```
<div class="border mx-0">.mx-0</div>
<div class="border mx-1">.mx-1</div>
<div class="border mx-2">.mx-2</div>
<div class="border mx-3">.mx-3</div>
<div class="border mx-4">.mx-4</div>
<div class="border mx-5">.mx-5</div>
```

- ◆ `my-0` — `margin-top: 0; margin-bottom: 0.`

Классы для адаптивной верстки: `my-sm-0`, `my-md-0`, `my-lg-0`, `my-xl-0`, `my-xxl-0` (Bootstrap 5);

- ◆ `my-1` — `margin-top: 0.25rem; margin-bottom: 0.25rem.`

Классы для адаптивной верстки: `my-sm-1`, `my-md-1`, `my-lg-1`, `my-xl-1`, `my-xxl-1` (Bootstrap 5);

- ◆ `my-2` — `margin-top: 0.5rem; margin-bottom: 0.5rem.`

Классы для адаптивной верстки: `my-sm-2`, `my-md-2`, `my-lg-2`, `my-xl-2`, `my-xxl-2` (Bootstrap 5);

- ◆ `my-3` — `margin-top: 1rem; margin-bottom: 1rem.`

Классы для адаптивной верстки: `my-sm-3`, `my-md-3`, `my-lg-3`, `my-xl-3`, `my-xxl-3` (Bootstrap 5);

- ◆ `my-4` — `margin-top: 1.5rem; margin-bottom: 1.5rem.`

Классы для адаптивной верстки: `my-sm-4`, `my-md-4`, `my-lg-4`, `my-xl-4`, `my-xxl-4` (Bootstrap 5);

- ◆ my-5 — margin-top: 3rem; margin-bottom: 3rem.

Классы для адаптивной верстки: my-sm-5, my-md-5, my-lg-5, my-xl-5, my-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border my-0">.my-0</div>
<div class="border my-1">.my-1</div>
<div class="border my-2">.my-2</div>
<div class="border my-3">.my-3</div>
<div class="border my-4">.my-4</div>
<div class="border my-5">.my-5</div>
```

Следующие стилевые классы доступны только в Bootstrap 4:

- ◆ m-n1 — margin: -0.25rem.

Классы для адаптивной верстки: m-sm-n1, m-md-n1, m-lg-n1, m-xl-n1;

- ◆ m-n2 — margin: -0.5rem.

Классы для адаптивной верстки: m-sm-n2, m-md-n2, m-lg-n2, m-xl-n2;

- ◆ m-n3 — margin: -1rem.

Классы для адаптивной верстки: m-sm-n3, m-md-n3, m-lg-n3, m-xl-n3;

- ◆ m-n4 — margin: -1.5rem.

Классы для адаптивной верстки: m-sm-n4, m-md-n4, m-lg-n4, m-xl-n4;

- ◆ m-n5 — margin: -3rem.

Классы для адаптивной верстки: m-sm-n5, m-md-n5, m-lg-n5, m-xl-n5;

- ◆ mt-n1 — margin-top: -0.25rem.

Классы для адаптивной верстки: mt-sm-n1, mt-md-n1, mt-lg-n1, mt-xl-n1;

- ◆ mt-n2 — margin-top: -0.5rem.

Классы для адаптивной верстки: mt-sm-n2, mt-md-n2, mt-lg-n2, mt-xl-n2;

- ◆ mt-n3 — margin-top: -1rem.

Классы для адаптивной верстки: mt-sm-n3, mt-md-n3, mt-lg-n3, mt-xl-n3;

- ◆ mt-n4 — margin-top: -1.5rem.

Классы для адаптивной верстки: mt-sm-n4, mt-md-n4, mt-lg-n4, mt-xl-n4;

- ◆ mt-n5 — margin-top: -3rem.

Классы для адаптивной верстки: mt-sm-n5, mt-md-n5, mt-lg-n5, mt-xl-n5;

- ◆ mb-n1 — margin-bottom: -0.25rem.

Классы для адаптивной верстки: mb-sm-n1, mb-md-n1, mb-lg-n1, mb-xl-n1;

- ◆ mb-n2 — margin-bottom: -0.5rem.

Классы для адаптивной верстки: mb-sm-n2, mb-md-n2, mb-lg-n2, mb-xl-n2;

- ◆ mb-n3 — margin-bottom: -1rem.

Классы для адаптивной верстки: mb-sm-n3, mb-md-n3, mb-lg-n3, mb-xl-n3;

- ◆ `mb-n4` — `margin-bottom: -1.5rem`.

Классы для адаптивной верстки: `mb-sm-n4`, `mb-md-n4`, `mb-lg-n4`, `mb-xl-n4`;

- ◆ `mb-n5` — `margin-bottom: -3rem`.

Классы для адаптивной верстки: `mb-sm-n5`, `mb-md-n5`, `mb-lg-n5`, `mb-xl-n5`;

- ◆ `ml-n1` — `margin-left: -0.25rem`.

Классы для адаптивной верстки: `ml-sm-n1`, `ml-md-n1`, `ml-lg-n1`, `ml-xl-n1`;

- ◆ `ml-n2` — `margin-left: -0.5rem`.

Классы для адаптивной верстки: `ml-sm-n2`, `ml-md-n2`, `ml-lg-n2`, `ml-xl-n2`;

- ◆ `ml-n3` — `margin-left: -1rem`.

Классы для адаптивной верстки: `ml-sm-n3`, `ml-md-n3`, `ml-lg-n3`, `ml-xl-n3`;

- ◆ `ml-n4` — `margin-left: -1.5rem`.

Классы для адаптивной верстки: `ml-sm-n4`, `ml-md-n4`, `ml-lg-n4`, `ml-xl-n4`;

- ◆ `ml-n5` — `margin-left: -3rem`.

Классы для адаптивной верстки: `ml-sm-n5`, `ml-md-n5`, `ml-lg-n5`, `ml-xl-n5`.

Пример:

```
<!-- Только в Bootstrap 4 -->
<div class="container bg-warning">
  <div class="border text-center w-50 ml-n1">.ml-n1</div>
  <div class="border text-center w-50 ml-n2">.ml-n2</div>
  <div class="border text-center w-50 ml-n3">.ml-n3</div>
  <div class="border text-center w-50 ml-n4">.ml-n4</div>
  <div class="border text-center w-50 ml-n5">.ml-n5</div>
</div>
```

- ◆ `mr-n1` — `margin-right: -0.25rem`.

Классы для адаптивной верстки: `mr-sm-n1`, `mr-md-n1`, `mr-lg-n1`, `mr-xl-n1`;

- ◆ `mr-n2` — `margin-right: -0.5rem`.

Классы для адаптивной верстки: `mr-sm-n2`, `mr-md-n2`, `mr-lg-n2`, `mr-xl-n2`;

- ◆ `mr-n3` — `margin-right: -1rem`.

Классы для адаптивной верстки: `mr-sm-n3`, `mr-md-n3`, `mr-lg-n3`, `mr-xl-n3`;

- ◆ `mr-n4` — `margin-right: -1.5rem`.

Классы для адаптивной верстки: `mr-sm-n4`, `mr-md-n4`, `mr-lg-n4`, `mr-xl-n4`;

- ◆ `mr-n5` — `margin-right: -3rem`.

Классы для адаптивной верстки: `mr-sm-n5`, `mr-md-n5`, `mr-lg-n5`, `mr-xl-n5`.

Пример:

```
<!-- Только в Bootstrap 4 -->
<div class="border text-center mr-n1">.mr-n1</div>
<div class="border text-center mr-n2">.mr-n2</div>
```

```
<div class="border text-center mr-n3">.mr-n3</div>
<div class="border text-center mr-n4">.mr-n4</div>
<div class="border text-center mr-n5">.mr-n5</div>
```

- ◆ mx-n1 — margin-left: -0.25rem; margin-right: -0.25rem.

Классы для адаптивной верстки: mx-sm-n1, mx-md-n1, mx-lg-n1, mx-xl-n1;

- ◆ mx-n2 — margin-left: -0.5rem; margin-right: -0.5rem.

Классы для адаптивной верстки: mx-sm-n2, mx-md-n2, mx-lg-n2, mx-xl-n2;

- ◆ mx-n3 — margin-left: -1rem; margin-right: -1rem.

Классы для адаптивной верстки: mx-sm-n3, mx-md-n3, mx-lg-n3, mx-xl-n3;

- ◆ mx-n4 — margin-left: -1.5rem; margin-right: -1.5rem.

Классы для адаптивной верстки: mx-sm-n4, mx-md-n4, mx-lg-n4, mx-xl-n4;

- ◆ mx-n5 — margin-left: -3rem; margin-right: -3rem.

Классы для адаптивной верстки: mx-sm-n5, mx-md-n5, mx-lg-n5, mx-xl-n5.

Пример:

```
<!-- Только в Bootstrap 4 -->
<div class="container bg-warning">
  <div class="border text-center w-50 mx-n1">.mx-n1</div>
  <div class="border text-center w-50 mx-n2">.mx-n2</div>
  <div class="border text-center w-50 mx-n3">.mx-n3</div>
  <div class="border text-center w-50 mx-n4">.mx-n4</div>
  <div class="border text-center w-50 mx-n5">.mx-n5</div>
</div>
```

- ◆ my-n1 — margin-top: -0.25rem; margin-bottom: -0.25rem.

Классы для адаптивной верстки: my-sm-n1, my-md-n1, my-lg-n1, my-xl-n1;

- ◆ my-n2 — margin-top: -0.5rem; margin-bottom: -0.5rem.

Классы для адаптивной верстки: my-sm-n2, my-md-n2, my-lg-n2, my-xl-n2;

- ◆ my-n3 — margin-top: -1rem; margin-bottom: -1rem.

Классы для адаптивной верстки: my-sm-n3, my-md-n3, my-lg-n3, my-xl-n3;

- ◆ my-n4 — margin-top: -1.5rem; margin-bottom: -1.5rem.

Классы для адаптивной верстки: my-sm-n4, my-md-n4, my-lg-n4, my-xl-n4;

- ◆ my-n5 — margin-top: -3rem; margin-bottom: -3rem.

Классы для адаптивной верстки: my-sm-n5, my-md-n5, my-lg-n5, my-xl-n5.

1.4.2. Внутренние отступы

Задать размер внутренних отступов позволяют следующие стилевые классы:

- ◆ p-0 — padding: 0.

Классы для адаптивной верстки: p-sm-0, p-md-0, p-lg-0, p-xl-0, p-xxl-0 (Bootstrap 5);

- ◆ p-1 — padding: 0.25rem.

Классы для адаптивной верстки: p-sm-1, p-md-1, p-lg-1, p-xl-1, p-xxl-1 (Bootstrap 5);

- ◆ p-2 — padding: 0.5rem.

Классы для адаптивной верстки: p-sm-2, p-md-2, p-lg-2, p-xl-2, p-xxl-2 (Bootstrap 5);

- ◆ p-3 — padding: 1rem.

Классы для адаптивной верстки: p-sm-3, p-md-3, p-lg-3, p-xl-3, p-xxl-3 (Bootstrap 5);

- ◆ p-4 — padding: 1.5rem.

Классы для адаптивной верстки: p-sm-4, p-md-4, p-lg-4, p-xl-4, p-xxl-4 (Bootstrap 5);

- ◆ p-5 — padding: 3rem.

Классы для адаптивной верстки: p-sm-5, p-md-5, p-lg-5, p-xl-5, p-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border p-0">.p-0</div>
<div class="border p-1">.p-1</div>
<div class="border p-2">.p-2</div>
<div class="border p-3">.p-3</div>
<div class="border p-4">.p-4</div>
<div class="border p-5">.p-5</div>
```

- ◆ pt-0 — padding-top: 0.

Классы для адаптивной верстки: pt-sm-0, pt-md-0, pt-lg-0, pt-xl-0, pt-xxl-0 (Bootstrap 5);

- ◆ pt-1 — padding-top: 0.25rem.

Классы для адаптивной верстки: pt-sm-1, pt-md-1, pt-lg-1, pt-xl-1, pt-xxl-1 (Bootstrap 5);

- ◆ pt-2 — padding-top: 0.5rem.

Классы для адаптивной верстки: pt-sm-2, pt-md-2, pt-lg-2, pt-xl-2, pt-xxl-2 (Bootstrap 5);

- ◆ pt-3 — padding-top: 1rem.

Классы для адаптивной верстки: pt-sm-3, pt-md-3, pt-lg-3, pt-xl-3, pt-xxl-3 (Bootstrap 5);

- ◆ pt-4 — padding-top: 1.5rem.

Классы для адаптивной верстки: pt-sm-4, pt-md-4, pt-lg-4, pt-xl-4, pt-xxl-4 (Bootstrap 5);

- ◆ pt-5 — padding-top: 3rem.

Классы для адаптивной верстки: pt-sm-5, pt-md-5, pt-lg-5, pt-xl-5, pt-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border pt-0">.pt-0</div>
<div class="border pt-1">.pt-1</div>
<div class="border pt-2">.pt-2</div>
<div class="border pt-3">.pt-3</div>
<div class="border pt-4">.pt-4</div>
<div class="border pt-5">.pt-5</div>
```

- ◆ pb-0 — padding-bottom: 0.

Классы для адаптивной верстки: pb-sm-0, pb-md-0, pb-lg-0, pb-xl-0, pb-xxl-0 (Bootstrap 5);

- ◆ pb-1 — padding-bottom: 0.25rem.

Классы для адаптивной верстки: pb-sm-1, pb-md-1, pb-lg-1, pb-xl-1, pb-xxl-1 (Bootstrap 5);

- ◆ pb-2 — padding-bottom: 0.5rem.

Классы для адаптивной верстки: pb-sm-2, pb-md-2, pb-lg-2, pb-xl-2, pb-xxl-2 (Bootstrap 5);

- ◆ pb-3 — padding-bottom: 1rem.

Классы для адаптивной верстки: pb-sm-3, pb-md-3, pb-lg-3, pb-xl-3, pb-xxl-3 (Bootstrap 5);

- ◆ pb-4 — padding-bottom: 1.5rem.

Классы для адаптивной верстки: pb-sm-4, pb-md-4, pb-lg-4, pb-xl-4, pb-xxl-4 (Bootstrap 5);

- ◆ pb-5 — padding-bottom: 3rem.

Классы для адаптивной верстки: pb-sm-5, pb-md-5, pb-lg-5, pb-xl-5, pb-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border pb-0">.pb-0</div>
<div class="border pb-1">.pb-1</div>
<div class="border pb-2">.pb-2</div>
<div class="border pb-3">.pb-3</div>
<div class="border pb-4">.pb-4</div>
<div class="border pb-5">.pb-5</div>
```

- ◆ pl-0 — padding-left: 0.

Классы для адаптивной верстки: pl-sm-0, pl-md-0, pl-lg-0, pl-xl-0, pl-xxl-0 (Bootstrap 5);

- ◆ pl-1 — padding-left: 0.25rem.

Классы для адаптивной верстки: pl-sm-1, pl-md-1, pl-lg-1, pl-xl-1, pl-xxl-1 (Bootstrap 5);

- ◆ pl-2 — padding-left: 0.5rem.

Классы для адаптивной верстки: pl-sm-2, pl-md-2, pl-lg-2, pl-xl-2, pl-xxl-2 (Bootstrap 5);

- ◆ pl-3 — padding-left: 1rem.

Классы для адаптивной верстки: pl-sm-3, pl-md-3, pl-lg-3, pl-xl-3, pl-xxl-3 (Bootstrap 5);

- ◆ pl-4 — padding-left: 1.5rem.

Классы для адаптивной верстки: pl-sm-4, pl-md-4, pl-lg-4, pl-xl-4, pl-xxl-4 (Bootstrap 5);

- ◆ pl-5 — padding-left: 3rem.

Классы для адаптивной верстки: pl-sm-5, pl-md-5, pl-lg-5, pl-xl-5, pl-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border pl-0">.pl-0</div>
<div class="border pl-1">.pl-1</div>
<div class="border pl-2">.pl-2</div>
<div class="border pl-3">.pl-3</div>
<div class="border pl-4">.pl-4</div>
<div class="border pl-5">.pl-5</div>
```

- ◆ pr-0 — padding-right: 0.

Классы для адаптивной верстки: pr-sm-0, pr-md-0, pr-lg-0, pr-xl-0, pr-xxl-0 (Bootstrap 5);

- ◆ pr-1 — padding-right: 0.25rem.

Классы для адаптивной верстки: pr-sm-1, pr-md-1, pr-lg-1, pr-xl-1, pr-xxl-1 (Bootstrap 5);

- ◆ pr-2 — padding-right: 0.5rem.

Классы для адаптивной верстки: pr-sm-2, pr-md-2, pr-lg-2, pr-xl-2, pr-xxl-2 (Bootstrap 5);

- ◆ pr-3 — padding-right: 1rem.

Классы для адаптивной верстки: pr-sm-3, pr-md-3, pr-lg-3, pr-xl-3, pr-xxl-3 (Bootstrap 5);

- ◆ pr-4 — padding-right: 1.5rem.

Классы для адаптивной верстки: pr-sm-4, pr-md-4, pr-lg-4, pr-xl-4, pr-xxl-4 (Bootstrap 5);

- ◆ pr-5 — padding-right: 3rem.

Классы для адаптивной верстки: pr-sm-5, pr-md-5, pr-lg-5, pr-xl-5, pr-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border pr-0 text-right">.pr-0</div>
<div class="border pr-1 text-right">.pr-1</div>
<div class="border pr-2 text-right">.pr-2</div>
<div class="border pr-3 text-right">.pr-3</div>
<div class="border pr-4 text-right">.pr-4</div>
<div class="border pr-5 text-right">.pr-5</div>
```

- ◆ px-0 — padding-left: 0; padding-right: 0.

Классы для адаптивной верстки: px-sm-0, px-md-0, px-lg-0, px-xl-0, px-xxl-0 (Bootstrap 5);

- ◆ px-1 — padding-left: 0.25rem; padding-right: 0.25rem.

Классы для адаптивной верстки: px-sm-1, px-md-1, px-lg-1, px-xl-1, px-xxl-1 (Bootstrap 5);

- ◆ px-2 — padding-left: 0.5rem; padding-right: 0.5rem.

Классы для адаптивной верстки: px-sm-2, px-md-2, px-lg-2, px-xl-2, px-xxl-2 (Bootstrap 5);

- ◆ px-3 — padding-left: 1rem; padding-right: 1rem.

Классы для адаптивной верстки: px-sm-3, px-md-3, px-lg-3, px-xl-3, px-xxl-3 (Bootstrap 5);

- ◆ px-4 — padding-left: 1.5rem; padding-right: 1.5rem.

Классы для адаптивной верстки: px-sm-4, px-md-4, px-lg-4, px-xl-4, px-xxl-4 (Bootstrap 5);

- ◆ px-5 — padding-left: 3rem; padding-right: 3rem.

Классы для адаптивной верстки: px-sm-5, px-md-5, px-lg-5, px-xl-5, px-xxl-5 (Bootstrap 5).

Пример:

```
<div class="border px-0">.px-0</div>
<div class="border px-1">.px-1</div>
<div class="border px-2">.px-2</div>
<div class="border px-3">.px-3</div>
<div class="border px-4">.px-4</div>
<div class="border px-5">.px-5</div>
```

- ◆ py-0 — padding-top: 0; padding-bottom: 0.

Классы для адаптивной верстки: py-sm-0, py-md-0, py-lg-0, py-xl-0, py-xxl-0 (Bootstrap 5);

- ◆ py-1 — padding-top: 0.25rem; padding-bottom: 0.25rem.

Классы для адаптивной верстки: py-sm-1, py-md-1, py-lg-1, py-xl-1, py-xxl-1 (Bootstrap 5);

- ◆ py-2 — padding-top: 0.5rem; padding-bottom: 0.5rem.

Классы для адаптивной верстки: py-sm-2, py-md-2, py-lg-2, py-xl-2, py-xxl-2 (Bootstrap 5);

- ◆ `py-3` — `padding-top: 1rem; padding-bottom: 1rem.`

Классы для адаптивной верстки: `py-sm-3`, `py-md-3`, `py-lg-3`, `py-xl-3`, `py-xxl-3` (Bootstrap 5);

- ◆ `py-4` — `padding-top: 1.5rem; padding-bottom: 1.5rem.`

Классы для адаптивной верстки: `py-sm-4`, `py-md-4`, `py-lg-4`, `py-xl-4`, `py-xxl-4` (Bootstrap 5);

- ◆ `py-5` — `padding-top: 3rem; padding-bottom: 3rem.`

Классы для адаптивной верстки: `py-sm-5`, `py-md-5`, `py-lg-5`, `py-xl-5`, `py-xxl-5` (Bootstrap 5).

Пример:

```
<div class="border py-0">.py-0</div>
<div class="border py-1">.py-1</div>
<div class="border py-2">.py-2</div>
<div class="border py-3">.py-3</div>
<div class="border py-4">.py-4</div>
<div class="border py-5">.py-5</div>
```

1.5. Рамки

Любой элемент веб-страницы занимает в окне веб-браузера некоторую прямоугольную область. Содержимое этой области может быть окружено рамками (см. рис. 1.7). Иными словами, рамки могут иметь не только таблицы, но и любые элементы веб-страницы — например, абзацы.

1.5.1. Отображение рамки

Отобразить рамку позволяют следующие стилевые классы:

- ◆ `border` — рамка со всеми границами, отображаемыми сплошными линиями толщиной в один пиксел серого цвета;
- ◆ `border-top` — граница только сверху;
- ◆ `border-bottom` — граница только снизу;
- ◆ `border-left` — граница только слева;
- ◆ `border-right` — граница только справа.

Пример:

```
<div class="container">
  <p class="border">.border</p><br>
  <p class="border-top">.border-top</p><br>
  <p class="border-bottom">.border-bottom</p><br>
  <p class="border-left">.border-left</p><br>
  <p class="border-right">.border-right</p>
</div>
```

Чтобы отобразить две границы следует добавить два стилевых класса:

```
<div class="container">
  <p class="border-left border-right">.border-left .border-right</p>
</div>
```

1.5.2. Соккрытие рамки

Отключить отображение рамки позволяют следующие стилевые классы:

- ◆ `border-0` — отключает отображение всех границ рамки;
- ◆ `border-top-0` — отключает отображение верхней границы рамки;
- ◆ `border-bottom-0` — отключает отображение нижней границы рамки;
- ◆ `border-left-0` — отключает отображение левой границы рамки;
- ◆ `border-right-0` — отключает отображение правой границы рамки.

Пример:

```
<style>
p { border: 1px solid #dee2e6; }
</style>
<div class="container">
  <p>border: 1px solid #dee2e6</p><br>
  <p class="border-0">.border-0</p><br>
  <p class="border-top-0">.border-top-0</p><br>
  <p class="border-bottom-0">.border-bottom-0</p><br>
  <p class="border-left-0">.border-left-0</p><br>
  <p class="border-right-0">.border-right-0</p>
</div>
```

1.5.3. Цвет линии рамки

Задать цвет линии рамки позволяют следующие стилевые классы (аналоги CSS-атрибута `border-color`):

- ◆ `border-white` — белый цвет;
- ◆ `border-dark` — темно-серый цвет;
- ◆ `border-light` — светло-серый цвет;
- ◆ `border-success` — цвет успешно выполненной операции (зеленый);
- ◆ `border-info` — цвет информационного сообщения;
- ◆ `border-warning` — цвет предупреждающего сообщения (желтый);
- ◆ `border-danger` — цвет опасности (красный);
- ◆ `border-primary` — синий цвет;
- ◆ `border-secondary` — серый цвет.

Пример:

```

<div class="container bg-dark text-white"><br>
  <p class="border border-white">.border-white</p><br>
</div><br>
<div class="container">
  <p class="border border-dark">.border-dark</p><br>
  <p class="border border-light">.border-light</p><br>
  <p class="border border-success">.border-success</p><br>
  <p class="border border-info">.border-info</p><br>
  <p class="border border-warning">.border-warning</p><br>
  <p class="border border-danger">.border-danger</p><br>
  <p class="border border-primary">.border-primary</p><br>
  <p class="border border-secondary">.border-secondary</p>
</div>

```

1.5.4. Рамки со скругленными углами

За скругление углов рамок отвечают следующие стилевые классы:

- ◆ `rounded` — скругление всех углов на величину `0.25 rem`;
- ◆ `rounded-sm` — скругление всех углов на величину `0.2 rem`;
- ◆ `rounded-lg` — скругление всех углов на величину `0.3 rem`;
- ◆ `rounded-circle` — скругление всех углов на величину `50%`;
- ◆ `rounded-pill` — скругление всех углов на величину `50 rem`;
- ◆ `rounded-top`, `rounded-bottom`, `rounded-left` и `rounded-right` — скругление углов с одной из сторон на величину `0.25 rem`;
- ◆ `rounded-0` — отключает скругление углов.

Пример:

```

<style>
.container p {
  height: 100px;
  padding-left: 10px;
  border: 5px solid #ffc107;
}
</style>
<div class="container">
  <p class="border-0 bg-info rounded">border-0 .rounded</p><br>
  <p class="rounded">.rounded</p><br>
  <p class="rounded-sm">.rounded-sm</p><br>
  <p class="rounded-lg">.rounded-lg</p><br>
  <p class="rounded-circle text-center" style="width: 100px">
    .rounded-circle</p><br>
  <p class="rounded-pill text-center">.rounded-pill</p><br>
  <p class="rounded-top">.rounded-top</p><br>
  <p class="rounded-bottom">.rounded-bottom</p><br>

```

```
<p class="rounded-left">.rounded-left</p><br>
<p class="rounded-right">.rounded-right</p><br>
<p class="rounded-pill rounded-0">.rounded-0</p>
</div>
```

ПРИМЕЧАНИЕ

Если рамка не задана, то происходит только скругление фона.

1.6. Списки

Список — это набор упорядоченных абзацев текста, помеченных специальными значками (*маркированные списки*) или цифрами (*нумерованные списки*). Кроме того, существуют списки определений, которые состоят из пар «термин/определение». Рассмотрим каждый из вариантов по отдельности.

1.6.1. Нумерованные списки

Нумерованный список помещают внутри парного тега ``. Перед каждым пунктом списка необходимо поместить тег ``. Закрывающий тег `` не обязателен. Пример:

```
<div class="container">
  <p>Нумерованный список</p>
  <ol>
    <li>Первый пункт</li>
    <li>Второй пункт</li>
    <li>Третий пункт</li>
  </ol>
  <p>Текст абзаца</p>
</div>
```

Для нумерованного списка прописаны следующие правила:

```
ol {
  margin-top: 0;
  margin-bottom: 1rem;
}
```

В Bootstrap 5 прописано дополнительное правило:

```
ol {
  padding-left: 2rem;
}
```

1.6.2. Маркированные списки

Маркированный список помещают внутри парного тега ``. Перед каждым пунктом списка необходимо поместить тег ``. Закрывающий тег `` не обязателен.

Пример:

```
<div class="container">
  <p>Маркированный список</p>
  <ul>
```

```

    <li>Первый пункт</li>
    <li>Второй пункт</li>
    <li>Третий пункт</li>
  </ul>
<p>Текст абзаца</p>
</div>

```

Для маркированного списка прописаны следующие правила:

```

ul {
  margin-top: 0;
  margin-bottom: 1rem;
}

```

В Bootstrap 5 прописано дополнительное правило:

```

ul {
  padding-left: 2rem;
}

```

1.6.3. Вложенные списки

Списки можно вкладывать друг в друга. Для вложенных списков значение атрибута `margin-bottom` равно нулю:

```

<div class="container">
  <p>Вложенные списки</p>
  <ul>
    <li>Первый пункт
      <ol>
        <li>Пункт 1.1</li>
        <li>Пункт 1.2</li>
        <li>Пункт 1.3</li>
      </ol>
    </li>
    <li>Второй пункт</li>
    <li>Третий пункт</li>
  </ul>
</div>

```

1.6.4. Списки без маркеров

Если к списку добавить стилевой класс `list-unstyled`, то список будет отображаться без маркеров (стиль `list-style: none`) и без внутреннего отступа слева (стиль `padding-left: 0`). На вложенные списки класс `list-unstyled` не влияет. Пример:

```

<div class="container">
  <p>Списки без маркеров</p>
  <ul class="list-unstyled">
    <li>Первый пункт
      <ul>
        <li>Пункт 1.1</li>
        <li>Пункт 1.2</li>
        <li>Пункт 1.3</li>
      </ul>
    </li>
  </ul>
</div>

```

```

    </ul>
  </li>
  <li>Второй пункт</li>
  <li>Третий пункт</li>
</ul>
</div>

```

Результат в окне веб-браузера:

Списки без маркеров

Первый пункт

о Пункт 1.1

о Пункт 1.2

о Пункт 1.3

Второй пункт

Третий пункт

Аналогичное действие оказывает и стилевой класс `list-inline`, но он используется совместно с классом `list-inline-item` для представления элементов списка на одной строке, а не на отдельных строках. Такой список очень часто используется для строки меню. Пример использования классов `list-inline` и `list-inline-item`:

```

<div class="container">
  <p>Списки без маркеров inline-block</p>
  <ul class="list-inline">
    <li class="list-inline-item">Первый пункт</li>
    <li class="list-inline-item">Второй пункт</li>
    <li class="list-inline-item">Третий пункт</li>
  </ul>
</div>

```

Результат в окне веб-браузера:

Списки без маркеров `inline-block`

Первый пункт Второй пункт Третий пункт

Правила для класса `list-inline-item`:

```

.list-inline-item {
  display: inline-block;
}
.list-inline-item:not(:last-child) {
  margin-right: 0.5rem;
}

```

1.6.5. Компонент *list-group*: список (введение)

При использовании стилевого класса `list-group` пункты списка без маркеров выводятся друг под другом внутри рамок. Для каждого пункта такого списка нужно добавить стилевой класс `list-group-item`:

```

<div class="container">
  <ul class="list-group">
    <li class="list-group-item">Первый пункт</li>

```

```

    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>

```

Если дополнительно добавить стилевой класс `list-group-flush`, то пункты списка будут разделены линиями, а не помещены внутри рамок:

```

<div class="container">
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>

```

ПРИМЕЧАНИЕ

Функционал компонента `list-group` очень большой, поэтому мы рассмотрим его подробно немного позже (см. *разд. 4.8*), когда закончим изучение всех необходимых стилевых классов и других компонентов.

1.6.6. Списки определений

Списки определений состоят из пар «термин/определение». Описываются они с помощью парного тега `<dl>`. Для вставки термина применяется тег `<dt>`, а для вставки определения — тег `<dd>`. Закрывающие теги `</dt>` и `</dd>` не обязательны. Пример использования списков определений:

```

<div class="container">
  <p>Списки определений</p>
  <dl>
    <dt>HTML (HyperText Markup Language)</dt>
    <dd>
      Язык разметки документа, описывающий форму отображения
      информации на экране компьютера
    </dd>
    <dt>CSS (Cascading Style Sheets)</dt>
    <dd>Каскадные таблицы стилей</dd>
  </dl>
</div>

```

Стили:

```

dl {
  margin-top: 0;
  margin-bottom: 1rem;
}
dt {
  font-weight: 700;
}
dd {
  margin-bottom: .5rem;
  margin-left: 0;
}

```

1.7. Таблицы

Таблица вставляется в HTML-документ с помощью парного тега `<table>`. Отдельная ячейка таблицы описывается тегом `<td>`, а ряд ячеек — с помощью тега `<tr>`. Ячейка заголовка описывается с помощью тега `<th>` (текст такой ячейки выделяется полужирным шрифтом). Тег `<caption>` позволяет задать название таблицы, которое в библиотеке Bootstrap отображается снизу таблицы.

Для логического форматирования таблицы предназначены теги `<thead>`, `<tbody>` и `<tfoot>`. Тег `<thead>` описывает заголовок столбцов таблицы, тег `<tbody>` — основное содержимое таблицы, а тег `<tfoot>` — «подвал» таблицы.

Правила, прописанные для таблиц в библиотеке Bootstrap по умолчанию:

```
table {
  border-collapse: collapse;
}
caption {
  padding-top: 0.75rem;      /* В Bootstrap 5: padding-top: 0.5rem; */
  padding-bottom: 0.75rem; /* В Bootstrap 5: padding-bottom: 0.5rem; */
  color: #6c757d;
  text-align: left;
  caption-side: bottom; /* В Bootstrap 5 правило прописано для table */
}
th {
  text-align: inherit;
  text-align: -webkit-match-parent; /* В Bootstrap 5 */
}
thead, tbody, tfoot, tr, td, th { /* В Bootstrap 5 */
  border-color: inherit;
  border-style: solid;
  border-width: 0;
}
```

Создадим таблицу с шестью ячейками, с заголовками столбцов и названием таблицы (листинг 1.2). Все ячейки таблицы пронумеруем от 1 до 6. Поместим таблицу внутри родительского элемента, для которого добавим рамку и отступы, а также зададим ширину в 50%. В дальнейшем эту таблицу будем использовать как шаблон для примеров, сокращая объем кода в книге.

Листинг 1.2. Шаблон таблицы

```
<div class="border w-50 m-3 p-3">
  <table>
    <caption>Название таблицы</caption>
    <thead>
      <tr>
        <th>Столбец 1</th>
        <th>Столбец 2</th>
      </tr>
    </thead>
```

```
|  |  |
| --- | --- |
| Ячейка 1 | Ячейка 2 |
| Ячейка 3 | Ячейка 4 |
| Ячейка 5 | Ячейка 6 |

```

1.7.1. Рамки таблицы и ячеек

По умолчанию таблица выводится без рамки и без какой-либо стилизации. Это сделано специально, чтобы не было конфликтов со сторонними компонентами, которые используют таблицы. Чтобы сделать таблицу красивой, следует добавить к тегу `<table>` стилевой класс `table`:

```
<table class="table">
```

В результате таблица займет всю ширину родительского элемента, добавятся внутренние отступы для ячеек и внешний отступ для таблицы снизу. Содержимое ячеек выровняется по горизонтали по левой стороне, а по вертикали — по верхней стороне. Для ячеек заголовков внутри раздела `thead` вертикальное выравнивание осуществляется по нижней стороне. Строки таблицы разделяются горизонтальными линиями, а внешние рамки таблицы отсутствуют. Нижняя граница ячеек заголовков внутри раздела `thead` в Bootstrap 4 выделяется удвоенной толщиной, а в Bootstrap 5 — линией черного цвета.

Чтобы отобразить все границы для таблицы и ячеек, следует добавить стилевой класс `table-bordered`:

```
<table class="table table-bordered">
```

Для изменения цвета линий рамки таблицы в Bootstrap 5 можно использовать стилевые классы `border-*` (см. *разд. 1.5.3*):

```
<!-- Только в Bootstrap 5 -->
```

```
<table class="table table-bordered border-success">
```

По умолчанию в библиотеке Bootstrap для таблиц атрибут `border-collapse` имеет значение `collapse`, поэтому рисуются лишь рамки, разделяющие ячейки. Каждая ячейка таблицы будет заключена в одинарную рамку. Значение атрибута `border-spacing` при этом игнорируется.

Скрыть все границы позволяет стилевой класс `table-borderless`:

```
<table class="table table-borderless">
```

1.7.2. Компактное отображение содержимого таблицы

Если указать стилевой класс `table-sm`, то таблица станет более компактной — за счет уменьшения величины внутренних отступов для ячеек:

```
<table class="table table-sm">
```

Если содержимое таблицы будет превосходить по ширине размеры родительского контейнера, то таблица выйдет за его рамки. Чтобы этого избежать, следует для родительского контейнера добавить стилевой класс `table-responsive`, который в случае необходимости для контейнера отображает горизонтальную полосу прокрутки:

```
<div class="border w-50 m-3 p-3">
  <div class="table-responsive">
    <table class="table">
...
    </table>
  </div>
</div>
```

Классы для адаптивной верстки: `table-responsive-sm`, `table-responsive-md`, `table-responsive-lg`, `table-responsive-xl`, `table-responsive-xxl` (Bootstrap 5):

```
@media (max-width: 575.98px) {
  /* .table-responsive-sm */
}
@media (max-width: 767.98px) {
  /* .table-responsive-md */
}
@media (max-width: 991.98px) {
  /* .table-responsive-lg */
}
@media (max-width: 1199.98px) {
  /* .table-responsive-xl */
}
@media (max-width: 1399.98px) {
  /* .table-responsive-xxl Bootstrap 5 */
}
```

Например, при добавлении стилевого класса `table-responsive-sm`, до точки останова `sm` таблица может при необходимости содержать горизонтальную полосу прокрутки, а для остальных точек останова таблица будет отображаться полностью:

```
<div class="table-responsive-sm">
```

1.7.3. Зебра

Чтобы таблица лучше читалась, цвет фона соседних строк нужно сделать разным. Выполнить такое чередование для раздела `TBODY` позволяет стилевой класс `table-striped`:

```
<table class="table table-striped">
```

1.7.4. Выделение строки при наведении указателя мыши

Если к таблице добавить стилевой класс `table-hover`, то при наведении указателя мыши строка таблицы из раздела `TBODY` будет подсвечиваться изменением цвета фона ячеек:

```
<table class="table table-hover">
```

1.7.5. Изменение цвета фона для таблицы, строки и ячеек

Задать цвет фона для таблицы, строки или ячеек позволяют следующие стилевые классы:

- ◆ `table-active` — цвет активной строки или ячейки;
- ◆ `table-dark` — темно-серый цвет;
- ◆ `table-light` — светло-серый цвет;
- ◆ `table-success` — цвет успешно выполненной операции (зеленый);
- ◆ `table-info` — цвет информационного сообщения;
- ◆ `table-warning` — цвет предупреждающего сообщения (желтый);
- ◆ `table-danger` — цвет опасности (красный);
- ◆ `table-primary` — синий цвет;
- ◆ `table-secondary` — серый цвет.

Сделаем цвет фона всех ячеек строки зеленым:

```
<tr class="table-success">
```

Изменим цвет фона только для одной ячейки:

```
<td class="table-success">Ячейка</td>
```

В Bootstrap 4 все ячейки заголовка можно сделать темными или светлыми, указав классы `thead-dark` (текст отображается белым цветом) или `thead-light` соответственно для тега `<thead>`:

```
<!-- Только в Bootstrap 4 -->
<table class="table table-striped">
  <thead class="thead-dark">
```

Или:

```
<!-- Только в Bootstrap 4 -->
<table class="table table-striped">
  <thead class="thead-light">
```

В Bootstrap 5 следует использовать классы `table-dark` и `table-light`:

```
<table class="table table-striped">
  <thead class="table-dark">
```

Для применения темной темы ко всей таблице, следует добавить стилевой класс `table-dark` к тегу `<table>`:

```
<table class="table table-dark table-striped">
```

Совместно с темной темой можно использовать и другие стилевые классы, которые мы уже рассмотрели ранее:

```
<table class="table table-sm table-dark table-hover">
```

Однако для изменения цвета фона строки или ячейки при работе с темной темой в Bootstrap 4 лучше воспользоваться стилевыми классами `bg-success`, `bg-info`, `bg-warning`, `bg-danger` или `bg-primary` (см. *разд. 1.1.6*):

```
<!-- Только в Bootstrap 4 -->
<tr class="bg-success">
<td class="bg-primary">Ячейка</td>
```

1.7.6. Местоположение и выравнивание заголовка таблицы

Отобразить заголовок таблицы позволяет тег `<caption>`. По умолчанию в библиотеке Bootstrap атрибут `caption-side` имеет значение `bottom`, поэтому заголовок таблицы отображается под таблицей. Чтобы вывести заголовок над таблицей, в Bootstrap 4 следует указать значение `top` для атрибута `caption-side` явным образом:

```
<caption style="caption-side: top">
  Название таблицы
</caption>
```

В Bootstrap 5 можно воспользоваться стилевым классом `caption-top`:

```
<!-- Только в Bootstrap 5 -->
<table class="table caption-top">
```

По умолчанию производится выравнивание заголовка по левому краю. Для выравнивания заголовка по центру или по правому краю нужно добавить стилевые классы `text-center` или `text-right` соответственно:

```
<caption class="text-right">
  Название таблицы
</caption>
```

1.8. Графика и видео

Давайте создадим в каталоге `C:\book\img` следующие изображения, которые будем использовать в дальнейших примерах:

- ◆ `img_w200h200.jpg` — размер 200×200 пикселей;
- ◆ `img_sm.jpg` — размер 510×200 пикселей;
- ◆ `img_md.jpg` — размер 690×200 пикселей;
- ◆ `img_lg.jpg` — размер 930×200 пикселей;
- ◆ `img_xl.jpg` — размер 1110×200 пикселей;
- ◆ `photo.jpg` — любая фотография шириной более 1200 пикселей.

1.8.1. Работа с изображениями

Изображения вставляются с помощью одинарного тега ``. В параметре `src` задается URL-адрес файла, в параметре `alt` — строка текста, которая будет выводиться на месте появления изображения до его загрузки или при отключенной графике, а также если изображение загрузить не удалось. Размеры изображения можно также указать в параметрах `width` и `height`. Пример:

```
<div class="container bg-warning">
  Текст
</div>
```

Стили для тега `` в Bootstrap 4:

```
img {
  vertical-align: middle;
  border-style: none;
}
```

По умолчанию изображение внутри родительского контейнера выравнивается по левому краю, а текст обтекает его справа. Для выравнивания по центру можно добавить для контейнера стилевой класс `text-center`, а для выравнивания по правой стороне — стилевой класс `text-right`:

```
<div class="container bg-warning text-center">
  Текст
</div>
```

Однако в этом случае по центру выравнивается не только изображение, но и текст. Чтобы выравнивалось только изображение, а текст выводился ниже изображения, нужно сделать тег `` блочным (стилевой класс `d-block`) и добавить стилевой класс `mx-auto` для вставки равных отступов слева и справа:

```
<div class="container bg-warning">
  Текст
</div>
```

Если параметры `width` и `height` отсутствуют и ширина изображения больше ширины контейнера, то изображение выйдет за его границы. Чтобы этого избежать, следует добавить стилевой класс `img-fluid`:

```
<div class="container bg-warning">
  Текст
</div>
```

Стили:

```
.img-fluid {
  max-width: 100%;
  height: auto;
}
```

Таким способом можно создать изображение большого размера, добавить стилевой класс `img-fluid`, и изображение будет масштабироваться под любые точки останова, занимая всю ширину контейнера. Однако большое изображение обычно имеет большой размер файла и будет долго загружаться на мобильных устройствах. Можно взять изображение среднего размера и дополнительно добавить стилевой класс `w-100`, который задает ширину равной 100%:

```
<div class="container bg-warning">
  Текст
</div>
```

Но в этом случае мы получим другую проблему — потеряем качество изображения при его увеличении. Правильнее будет создать несколько изображений разной ширины и дать возможность веб-браузеру выбрать нужное изображение в зависимости от ширины экрана пользователя. Для этого тег `` имеет параметры `srcset` и `sizes`:

```
<div class="container bg-warning">
  Текст
</div>
```

Если веб-браузер не поддерживает параметры `srcset` и `sizes`, то он для всех размеров экранов будет использовать изображение `img_xl.jpg`. В противном случае:

- ◆ для точек останова `xs` и `sm` (ширина от 0 до 767 px) будет использовано изображение `img_sm.jpg`;
- ◆ для точки останова `md` (ширина до 991 px) — изображение `img_md.jpg`;
- ◆ для точки останова `lg` (ширина до 1199 px) — изображение `img_lg.jpg`;
- ◆ в противном случае — изображение `img_xl.jpg`.

Помимо тега `` в этом случае удобнее дополнительно использовать тег `<picture>` и несколько вложенных тегов `<source>` с указанием пути к файлам и медиазапросов с описанием критериев использования этих файлов:

```
<div class="container bg-warning">
  <picture>
    <source srcset="img/img_sm.jpg" media="(max-width: 767.98px)">
    <source srcset="img/img_md.jpg" media="(max-width: 991.98px)">
    <source srcset="img/img_lg.jpg" media="(max-width: 1199.98px)">
    <source srcset="img/img_xl.jpg" media="(min-width: 1200px)">
    
  </picture>Текст
</div>
```

Если для изображения добавить стилевой класс `img-thumbnail`, то вокруг изображения будет рисоваться рамка со скругленными углами:

```
<div class="container p-2">
  Текст
</div>
```

Заметьте, что стилевой класс `img-thumbnail` включает правила из стилевого класса `img-fluid`, поэтому изображение никогда не выйдет за границы родительского контейнера:

```
.img-thumbnail {
  padding: 0.25rem;
  background-color: #fff;
  border: 1px solid #dee2e6;
  border-radius: 0.25rem;
  max-width: 100%;
  height: auto;
}
```

Для скругления углов изображения можно воспользоваться стилевыми классами из *разд. 1.5.4* (например, классами `rounded`, `rounded-circle` и `rounded-pill`):

```
<div class="container p-2">
  
  
  
</div>
```

1.8.2. Готовые значки

Библиотека Bootstrap имеет собственную коллекцию значков в формате SVG, которые можно использовать в проектах. Перейдите на сайт <https://icons.getbootstrap.com/>, чтобы увидеть доступные значки. Значки не входят в базовый состав библиотеки Bootstrap, и их нужно скачивать отдельно. Для загрузки значков переходим на сайт <https://github.com/twbs/icons/releases> и скачиваем архив. Распаковываем архив и копируем файлы значков в каталог `C:\book\img\icons`.

Все значки сделаны в векторном формате, поэтому их можно масштабировать без потери качества. Подключить значок можно несколькими способами:

- ◆ с помощью тега ``. Размер значка задается в параметрах `width` и `height`:

```
<div class="container">
  
</div>
```

- ◆ с помощью тега `<svg>`. Просто откройте файл со значком текстовым редактором, скопируйте содержимое файла и вставьте его в HTML-документ. Размер значка

задается в параметрах `width` и `height`. В этом случае мы можем дополнительно стилизовать значок (например, сделав значок синего цвета, добавив стилевой класс `text-primary`):

```
<div class="container">
  <svg class="bi bi-bootstrap text-primary"
    width="64" height="64"
    viewBox="0 0 16 16" fill="currentColor"
    xmlns="http://www.w3.org/2000/svg">
    ...
  </svg>
</div>
```

Стили для тега `<svg>` в Bootstrap 4:

```
svg {
  overflow: hidden;
  vertical-align: middle;
}
```

◆ с помощью CSS-атрибутов `background` и `background-image`:

```
<style>
.icon-exclamation-triangle::before {
  content: "";
  display: inline-block;
  background-image: url('img/icons/exclamation-triangle.svg');
  background-repeat: no-repeat;
  background-size: 2rem 2rem;
  width: 2rem;
  height: 2rem;
  vertical-align: bottom;
}
</style>
<div class="container">
  <i class="icon-exclamation-triangle"></i> Текст
</div>
```

1.8.3. Добавление описания к изображению

Если нужно к изображению добавить текстовое описание, то удобно воспользоваться тегами `<figure>` и `<figcaption>`. Для тега `<figure>` следует добавить стилевой класс `figure`, для тега `` — стилевой класс `figure-img`, а для тега `<figcaption>` — стилевой класс `figure-caption`:

```
<div class="container">
  <figure class="figure">
    
    <figcaption class="figure-caption text-center">
      Подпись к изображению
    </figcaption>
  </figure>
</div>
```

Стили:

```
figure {
  margin: 0 0 1rem;
}
.figure {
  display: inline-block;
}
.figure-img {
  margin-bottom: 0.5rem;
  line-height: 1;
}
.figure-caption {
  font-size: 90%; /* В Bootstrap 5: font-size: 0.875em; */
  color: #6c757d;
}
```

Если нужно справа или слева от изображения вывести большой блок текста, то в Bootstrap 4:

- ◆ создаем родительский контейнер и добавляем для него стилевой класс `media`, который преобразует элемент в блочный flex-контейнер с горизонтальным выравниванием;
- ◆ если изображение должно быть расположено слева от текста, то добавляем тег `` в начало контейнера с классом `media`, а если справа — то в конец контейнера с классом `media`. Чтобы изображение вплотную не прилегало к тексту, следует в зависимости от местоположения изображения добавить стилевой класс `mr-3` или `ml-3`;
- ◆ описание изображения вкладываем в контейнер со стилевым классом `media-body` и добавляем его внутрь контейнера с классом `media`.

Выведем изображение слева от текста в Bootstrap 4:

```
<!-- Только в Bootstrap 4 -->
<div class="container mb-3">
  <div class="media">
    
    <div class="media-body">
      <h5 class="mt-0">Заголовок</h5>
      <p style="height: 100px" class="bg-light">Какое-то описание.</p>
      <a href="#">Текст ссылки</a>
    </div>
  </div>
</div>
```

По умолчанию ссылкой является только содержимое между тегами `<a>` и ``. Если нужно, чтобы ссылка занимала весь блок с описанием и изображением, то для контейнера с классом `media` указываем относительное позиционирование (стилевой класс `position-relative`), а к ссылке добавляем стилевой класс `stretched-link`.

Выведем изображение справа от текста в Bootstrap 4 и сделаем ссылкой весь контейнер с классом `media`, а заодно добавим рамку со скругленными углами и внутренние отступы:

```
<!-- Только в Bootstrap 4 -->
<div class="container mb-3">
  <div class="media position-relative border rounded p-3">
    <div class="media-body">
      <h5 class="mt-0">Заголовок</h5>
      <p style="height: 100px" class="bg-light">Какое-то описание.</p>
      <a href="#" class="stretched-link">Текст ссылки</a>
    </div>
    
  </div>
</div>
```

По умолчанию изображение выравнивается по верху контейнера с классом `media`. Для выравнивания по центру или нижнему краю следует к изображению добавить стилевой класс `align-self-center` или `align-self-end` соответственно. Пример выравнивания изображения по центру в Bootstrap 4:

```
<!-- Только в Bootstrap 4 -->
<div class="container mb-3">
  <div class="media border rounded p-3">
    
    <div class="media-body">
      <h5 class="mt-0">Заголовок</h5>
      <p style="height: 300px" class="bg-light">Какое-то описание.</p>
      <a href="#">Текст ссылки</a>
    </div>
  </div>
</div>
```

Если изображений с описанием несколько, то можно стилевой класс `media` добавить к пунктам списка, а содержимое вставлять внутрь пунктов. Чтобы убрать стандартное оформление списка, следует добавить стилевой класс `list-unstyled`. Для вставки отступов между пунктами списка можно использовать стилевой класс `mb-3` или `mb-4`:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <ul class="list-unstyled">
    <li class="media mb-3">
      
      <div class="media-body">
        <h5 class="mt-0">Заголовок</h5>
        <p style="height: 100px" class="bg-light">Какое-то описание.</p>
        <a href="#">Текст ссылки</a>
      </div>
    </li>
    <li class="media mb-3">
```

```


<div class="media-body">
  <h5 class="mt-0">Заголовок</h5>
  <p style="height: 100px" class="bg-light">Какое-то описание.</p>
  <a href="#">Текст ссылки</a>
</div>
</li>
</ul>
</div>

```

ПРИМЕЧАНИЕ

В Bootstrap 5 стилевые классы `media` и `media-body` отсутствуют.

1.8.4. Вставка видео

Чтобы видео с YouTube, вставленное на веб-страницу, могло адаптироваться под различные размеры экрана, следует вложить тег `<iframe>` в контейнер со стилевым классом `embed-responsive`. Этот класс делает элемент блочным, с шириной 100% и относительным позиционированием. Кроме того, нужно задать соотношение сторон видео с помощью стилевых классов: `embed-responsive-1by1`, `embed-responsive-4by3`, `embed-responsive-16by9` или `embed-responsive-21by9`.

Для тега `<iframe>` можно добавить стилевой класс `embed-responsive-item`, хотя это и необязательно, т. к. правила явным образом прописаны для тегов `<iframe>`, `<embed>`, `<video>` и `<object>`, вложенных в контейнер с классом `embed-responsive`. Класс `embed-responsive-item` добавляет абсолютное позиционирование и привязку к сторонам родительского контейнера, задает ширину и высоту равную 100%, а также в Bootstrap 4 убирает рамку, поэтому параметр `frameborder` со значением 0 указывать не нужно.

Пример вставки видео с YouTube:

```

<div class="container mb-3">
  <div class="embed-responsive embed-responsive-16by9">
    <iframe src="https://www.youtube.com/embed/X_GJAvZ4_M?rel=0"
      class="embed-responsive-item" allowfullscreen></iframe>
  </div>
</div>

```

1.9. Форматирование блоков

Любой элемент веб-страницы занимает в окне веб-браузера некоторую прямоугольную область. Эта область имеет как внутренние, так и внешние отступы, а также содержит реальную или воображаемую границу. Тип блочной модели зависит от формата документа. Если тег `<!doctype>` указан, то блочная модель соответствует стандартам консорциума W3C. Реальные размеры элемента (см. рис. 1.7) вычисляются так:

$$\text{Реальная ширина} = \text{margin-left} + \text{border-left-width} + \text{padding-left} + \text{width} + \text{padding-right} + \text{border-right-width} + \text{margin-right}$$

Реальная высота = $\text{margin-top} + \text{border-top-width} + \text{padding-top} + \text{height} + \text{padding-bottom} + \text{border-bottom-width} + \text{margin-bottom}$

Если тег `<!doctype>` не указан, то некоторые веб-браузеры переходят в режим совместимости. Поэтому при отсутствии тега `<!doctype>` разные веб-браузеры могут по-разному отображать веб-страницу.

1.9.1. Указание типа блока

Для указания типа блока предназначены следующие стилевые классы (аналоги CSS-атрибута `display`):

- ◆ `d-none` — содержимое блока не отображается.

Классы для адаптивной верстки: `d-sm-none`, `d-md-none`, `d-lg-none`, `d-xl-none`, `d-xxl-none` (Bootstrap 5).

Класс для вывода на печать: `d-print-none`;

- ◆ `d-block` — блок занимает всю ширину родительского элемента (блочные теги `<div>` и `<p>`).

Классы для адаптивной верстки: `d-sm-block`, `d-md-block`, `d-lg-block`, `d-xl-block`, `d-xxl-block` (Bootstrap 5).

Класс для вывода на печать: `d-print-block`;

- ◆ `d-inline` — блок занимает только необходимое для отображения содержимого пространство (строчные теги ``, `` и др.). Задать размеры блока нельзя.

Классы для адаптивной верстки: `d-sm-inline`, `d-md-inline`, `d-lg-inline`, `d-xl-inline`, `d-xxl-inline` (Bootstrap 5).

Класс для вывода на печать: `d-print-inline`;

- ◆ `d-inline-block` — аналогично `d-inline`, но дополнительно можно задать размеры и другое форматирование, применяемое для блочного элемента. Результат аналогичен встраиванию тега `` в строку.

Классы для адаптивной верстки: `d-sm-inline-block`, `d-md-inline-block`, `d-lg-inline-block`, `d-xl-inline-block`, `d-xxl-inline-block` (Bootstrap 5).

Класс для вывода на печать: `d-print-inline-block`;

- ◆ `d-table` — блочная таблица (тег `<table>`).

Классы для адаптивной верстки: `d-sm-table`, `d-md-table`, `d-lg-table`, `d-xl-table`, `d-xxl-table` (Bootstrap 5).

Класс для вывода на печать: `d-print-table`;

- ◆ `d-table-row` — строка таблицы (тег `<tr>`).

Классы для адаптивной верстки: `d-sm-table-row`, `d-md-table-row`, `d-lg-table-row`, `d-xl-table-row`, `d-xxl-table-row` (Bootstrap 5).

Класс для вывода на печать: `d-print-table-row`;

- ◆ `d-table-cell` — ячейка таблицы (теги `<th>` и `<td>`).

Классы для адаптивной верстки: `d-sm-table-cell`, `d-md-table-cell`, `d-lg-table-cell`, `d-xl-table-cell`, `d-xxl-table-cell` (Bootstrap 5).

Класс для вывода на печать: `d-print-table-cell`;

- ◆ `d-flex` — блочный флекс-контейнер.

Классы для адаптивной верстки: `d-sm-flex`, `d-md-flex`, `d-lg-flex`, `d-xl-flex`, `d-xxl-flex` (Bootstrap 5).

Класс для вывода на печать: `d-print-flex`;

- ◆ `d-inline-flex` — строчный флекс-контейнер.

Классы для адаптивной верстки: `d-sm-inline-flex`, `d-md-inline-flex`, `d-lg-inline-flex`, `d-xl-inline-flex`, `d-xxl-inline-flex` (Bootstrap 5).

Класс для вывода на печать: `d-print-inline-flex`.

Различные варианты указания типа блока приведены в листинге 1.3.

Листинг 1.3. Указание типа блока

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <style>
    div.b div { border: 2px solid #333; }
  </style>
  <title>Указание типа блока</title>
</head>
<body>

<div class="container">
  <h1>Различные типы блоков</h1>

  <div class="b">
    <div class="d-inline">d-inline (display: inline)</div>
    <div class="d-inline-block" style="width: 400px">
      d-inline-block (display: inline-block)
    </div>
    <div class="d-block">d-block (display: block)</div>
    <div class="d-none">Этого блока не видно</div>
  </div>

  <h2>Выравнивание элементов формы</h2>
  <div>
    <p>Для элемента LABEL в Bootstrap прописано правило:
    <code>display: inline-block</code></p>
```

```
<label for="login" style="width: 100px">Логин:</label>
<input type="text" name="login" id="login">
</div>
<div>
  <label for="passw" style="width: 100px">Пароль:</label>
  <input type="password" name="passw" id="passw">
</div>

<h2>Указание типа блока для ссылки</h2>
<div class="b">
  <div style="width: 400px">
    <a href="link1.html">Обычная ссылка</a>
  </div>
  <div style="width: 400px">
    <a href="link2.html" class="d-block">Ссылка занимает всю
      ширину блока</a>
  </div>
</div>
</div>
</body>
</html>
```

В библиотеке Bootstrap 4 для всех новых блочных элементов, добавленных в HTML 5, в целях совместимости со старыми веб-браузерами дополнительно прописывается правило `display: block`. Если веб-браузер не поддерживает стандарт HTML 5, то новые элементы будут просто блочными:

```
article, aside, figcaption, figure, footer, header, hgroup, main, nav,
section {
  display: block;
}
```

1.9.2. Указание размеров

В библиотеке Bootstrap для всех элементов значение атрибута `box-sizing` устанавливается равным `border-box`, поэтому размеры будут задаваться для самого контейнера с учетом его внутренних отступов и рамки (без учета внешних отступов):

```
*,
*::before,
*::after {
  box-sizing: border-box;
}
```

Обычно в CSS по умолчанию используется значение `content-box`, которое не учитывает внутренние отступы и рамку, поэтому, если возникают проблемы, то нужно вручную прописать значение `content-box` для конкретного элемента.

Указать размеры блока позволяют следующие стилевые классы:

- ◆ `w-auto` — `width: auto;`
- ◆ `w-25` — `width: 25%;`

◆ w-50 — width: 50%;

◆ w-75 — width: 75%;

◆ w-100 — width: 100%. **Пример:**

```
<div class="w-25 border">.w-25</div>
<div class="w-50 border">.w-50</div>
<div class="w-75 border">.w-75</div>
<div class="w-100 border">.w-100</div>
<div class="w-auto border">.w-auto</div>
```

◆ h-auto — height: auto;

◆ h-25 — height: 25%;

◆ h-50 — height: 50%;

◆ h-75 — height: 75%;

◆ h-100 — height: 100%. **Пример:**

```
<div class="container bg-warning clearfix"
  style="height: 300px">
  <div class="w-25 h-25 border float-left">.h-25</div>
  <div class="w-25 h-50 border float-left">.h-50</div>
  <div class="w-25 h-75 border float-left">.h-75</div>
  <div class="w-25 h-100 border float-left">.h-100</div>
</div>
```

◆ vw-100 — width: 100vw;

◆ vh-100 — height: 100vh. **Пример заполнения всей доступной области:**

```
<div class="bg-warning vw-100 vh-100">.vw-100 .vh-100</div>
```

◆ mw-100 — max-width: 100%;

◆ mh-100 — max-height: 100%;

◆ min-vw-100 — min-width: 100vw;

◆ min-vh-100 — min-height: 100vh. **Пример заполнения всей доступной области:**

```
<div class="bg-warning min-vw-100 min-vh-100">
.min-vw-100 .min-vh-100</div>
```

1.9.3. Атрибут *overflow*

Следующие стилевые классы задают поведение блока, чье содержимое выходит за его границы (аналоги CSS-атрибута *overflow*):

◆ overflow-hidden — то, что не помещается в блоке, скрывается;

◆ overflow-auto — если содержимое не помещается в блок, то добавляются полосы прокрутки. Если же содержимое полностью помещается, то полосы прокрутки не отображаются.

Различные варианты использования атрибута *overflow* приведены в листинге 1.4.

Листинг 1.4. Атрибут overflow

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Атрибут overflow</title>
</head>
<body>
<div class="container">
  <h1>Атрибут overflow</h1>

  <p>Класс overflow-hidden</p>
  <div class="border overflow-hidden bg-warning mb-3"
    style="width: 100px; height: 100px">
    это очень длинная строка без пробелов
    То, что не влезает в границы блока, скрывается
  </div>

  <p>Класс overflow-auto</p>
  <div class="border overflow-auto bg-warning mb-3"
    style="width: 100px; height: 100px">
    это очень длинная строка без пробелов
    overflow-auto. Если содержимое не помещается в блок, то
    добавляются полосы прокрутки
  </div>

  <p>overflow: visible (по умолчанию). Высота не задана</p>
  <div class="border bg-warning mb-3" style="width: 100px">
    это очень длинная строка без пробелов
    overflow: visible. Блок расширяется так, чтобы все его
    содержимое отобразилось полностью
  </div>

  <p>overflow: visible (по умолчанию). Высота задана</p>
  <div class="border bg-warning"
    style="width: 100px; height: 100px">
    это очень длинная строка без пробелов
    overflow: visible. Если размеры заданы, то содержимое будет
    выходить за границы блока
  </div>
</div>
</body>
</html>
```

1.9.4. Управление обтеканием

Следующие стилевые классы определяют, по какой стороне производится выравнивание блока (аналоги CSS-атрибута `float`):

- ◆ `float-left` — блок выравнивается по левой стороне, а другие элементы обтекают его справа (`float: left`).

Классы для адаптивной верстки: `float-sm-left`, `float-md-left`, `float-lg-left`, `float-xl-left`, `float-xxl-left` (Bootstrap 5);

- ◆ `float-right` — блок выравнивается по правой стороне, а другие элементы обтекают его слева (`float: right`).

Классы для адаптивной верстки: `float-sm-right`, `float-md-right`, `float-lg-right`, `float-xl-right`, `float-xxl-right` (Bootstrap 5);

- ◆ `float-none` — выравнивание отсутствует (`float: none`).

Классы для адаптивной верстки: `float-sm-none`, `float-md-none`, `float-lg-none`, `float-xl-none`, `float-xxl-none` (Bootstrap 5).

Стилевой класс `clearfix` запрещает обтекание по обеим сторонам (аналог CSS-атрибута `clear` со значением `both`):

```
.clearfix::after {
  display: block;
  clear: both;
  content: "";
}
```

Пример:

```
<div class="container bg-warning clearfix"
  style="height: 300px">
  <div class="w-25 h-25 border float-right">.h-25</div>
  <div class="w-25 h-50 border float-right">.h-50</div>
  <div class="w-25 h-75 border float-right">.h-75</div>
  <div class="w-25 h-100 border float-right">.h-100</div>
</div>
```

1.9.5. Позиционирование блока

Следующие стилевые классы позволяют задать способ позиционирования блока (аналоги CSS-атрибута `position`):

- ◆ `position-static` — статическое позиционирование (значение по умолчанию). Положение элемента в окне веб-браузера определяется его положением в HTML-документе;
- ◆ `position-relative` — относительное позиционирование. Координаты отсчитываются относительно позиции, в которую веб-браузер поместил бы элемент, будь он статически позиционированным;
- ◆ `position-absolute` — абсолютное позиционирование. Координаты отсчитываются относительно левого верхнего угла ближайшего родительского элемента, который имеет позиционирование, отличное от статического;

- ◆ `position-fixed` — фиксированное позиционирование. Координаты отсчитываются относительно левого верхнего угла окна веб-браузера. При прокрутке содержимого окна блок не смещается;
- ◆ `fixed-top` — фиксированное позиционирование с привязкой к верху окна веб-браузера:

```
.fixed-top {
  position: fixed;
  top: 0;
  right: 0;
  left: 0;
  z-index: 1030;
}
```
- ◆ `fixed-bottom` — фиксированное позиционирование с привязкой к низу окна веб-браузера:

```
.fixed-bottom {
  position: fixed;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 1030;
}
```
- ◆ `position-sticky` — сочетание `relative` и `fixed`. Вначале используется относительное позиционирование. Как только элемент доходит до указанной позиции, применяется фиксированное позиционирование;
- ◆ `sticky-top` — аналог класса `position-sticky`, для которого дополнительно задано значение атрибута `top`, равное нулю, и значение атрибута `z-index`, равное 1020. Этот класс удобно использовать для закрепления панели навигации. В Bootstrap 5 можно воспользоваться следующими классами для адаптивной верстки: `sticky-sm-top`, `sticky-md-top`, `sticky-lg-top`, `sticky-xl-top`, `sticky-xxl-top`.

Давайте рассмотрим все это на примере (листинг 1.5).

Листинг 1.5. Позиционирование блоков

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <style>
    .div1 { width: 10px; height: 2000px; left: 900px; top: 0; }
    .div3 { top: 30px; }
    .div4 { width: 180px; height: 180px; top: 180px;
      left: 600px; background-color: green; }
    .div5 { width: 300px; height: 300px; top: 180px; left: 280px; }
    .div6 { width: 100px; height: 100px; top: 100px; left: 50px; }
```

```

    .div7 { width: 180px; height: 300px; top: 180px;
           left: 50px; background-color: #FF9600; }
  </style>
  <title>Позиционирование блоков</title>
</head>
<body class="mt-5">
  <div class="border position-absolute div1"></div>
  <div class="border position-static p-2 div2">
    Статическое позиционирование
  </div>
  <div class="border position-relative p-2 div3">
    Относительное позиционирование
  </div>
  <div class="border position-absolute text-white p-2 div4">
    Абсолютное позиционирование
  </div>
  <div class="border position-absolute p-2 div5">
    Абсолютное позиционирование внутри родительского блока
    <div class="border position-absolute bg-light div6">
      top: 100px; left: 50px;
    </div>
  </div>
  <div class="border position-fixed p-2 div7">
    Фиксированное позиционирование
  </div>
  <div class="fixed-top bg-warning p-2 div8"
        style="height: 50px;">
    Фиксированное позиционирование относительно
    верхней границы
  </div>
  <div class="fixed-bottom bg-warning p-2 div9"
        style="height: 50px;">
    Фиксированное позиционирование относительно
    нижней границы
  </div>
</body>
</html>

```

Итак, на странице девять блоков.

Блок `div1` имеет абсолютное позиционирование и смещен на 900 px относительно левой границы веб-страницы. Для блока также указана большая высота (2000 px). Это позволит увидеть эффект фиксированного позиционирования для блоков `div7`, `div8` и `div9`, т. к. веб-браузер отобразит вертикальную полосу прокрутки.

Блок `div2` имеет статическое позиционирование, а блок `div3` — относительное. Блок `div3` сдвинут на 30 px вниз относительно блока `div2`, а не от верхней границы веб-страницы.

Блоки `div4`, `div5` и `div6` имеют абсолютное позиционирование. Блок `div4` сдвинут на 600 px относительно левой границы веб-страницы и на 180 px — относительно

верхней. Внутри блока `div5` расположен блок `div6`. Смещения этого блока отсчитываются относительно блока `div5`, а не границ веб-страницы.

Блоки `div7`, `div8` и `div9` имеют фиксированное позиционирование. Блок `div7` демонстрирует возможность размещения панели навигации в определенном месте, блок `div8` — прикрепление блока к верхней границе окна веб-браузера, а `div9` — прикрепление блока к нижней границе окна веб-браузера. Чтобы увидеть отличие от других видов позиционирования, переместите вертикальную полосу прокрутки вниз. Эти блоки всегда остаются на своих местах и не перемещаются при прокрутке.

Пример использования стилевых классов `sticky-top` и `position-sticky` приведен в листинге 1.6. Сначала нужно уменьшить высоту окна веб-браузера таким образом, чтобы появилась вертикальная полоса прокрутки. При перемещении бегунка полосы прокрутки вниз, вначале панель навигации будет сдвигаться вверх, как обычный элемент страницы. Как только до верха страницы останется 90 пикселей, панель навигации остановится, а все остальное содержимое страницы будет сдвигаться вверх и дальше. Таким образом панель навигации всегда будет видна, независимо от положения полосы прокрутки. Заметьте также, что заголовок всегда остается на своем месте — прикрепленным к верхней части окна, благодаря стилевому классу `sticky-top`.

Листинг 1.6. Классы `sticky-top` и `position-sticky`

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <style>
    nav {
      width: 150px;
      min-height: 150px;
      top: 90px;
    }
    section {
      margin: 10px 10px 10px 185px;
      min-height: 800px;
    }
  </style>
  <title>Классы sticky-top и position-sticky</title>
</head>
<body>
  <header class="sticky-top">
    <div class="container-fluid bg-warning text-center p-3">
      <h1>Зароловок</h1>
    </div>
  </header>
  <div class="container-fluid clearfix p-2">
```

```

<nav class="float-left position-sticky p-2 m-2">
  Панель навигации
  <ul class="list-unstyled">
    <li><a href="page2.html">Страница 2</a></li>
    <li><a href="page3.html">Страница 3</a></li>
  </ul>
</nav>
<section class="border p-2">
  <h2>Основное содержимое страницы</h2>
  <p>Какой-то текст</p>
</section>
</div>
<footer>
  <div class="container-fluid bg-dark text-light p-3">
    Информация об авторских правах
  </div>
</footer>
</body>
</html>

```

1.9.6. Управление отображением элемента

Управлять отображением элемента в окне веб-браузера позволяют следующие стилевые классы (аналоги CSS-атрибута `visibility`):

- ◆ `visible` — элемент отображается независимо от видимости родителя;
- ◆ `invisible` — элемент скрывается независимо от видимости родителя.

Невидимый элемент при использовании стилевого класса `invisible` все равно занимает место на веб-странице. Для того чтобы скрыть элемент и убрать его с веб-страницы, можно воспользоваться стилевым классом `d-none`:

```

<div class="container bg-warning">
  <p class="visible border">Абзац 1</p>
  <p class="invisible">Абзац 2</p>
  <p class="border">Абзац 3</p>
  <p class="d-none">Абзац 4</p>
</div>

```

Для сокрытия элемента можно также воспользоваться универсальным параметром `hidden`, появившемся в HTML 5:

```
<div hidden>Текст</div>
```

Стиль:

```

[hidden] {
  display: none !important;
}

```

1.9.7. Создание тени для блока

Классы стилей `shadow` (средняя тень), `shadow-sm` (небольшая тень) и `shadow-lg` (большая тень) позволяют добавить тень для блока (аналоги CSS-атрибута `box-shadow`).

Для отключения тени используется стилевой класс `shadow-none`. Пример добавления тени для блоков:

```
<div style="width: 200px; height: 50px; margin: 50px"
  class="shadow border">.shadow</div>
<div style="width: 200px; height: 50px; margin: 100px"
  class="shadow-sm border">.shadow-sm</div>
<div style="width: 200px; height: 50px; margin: 50px"
  class="shadow-lg border">.shadow-lg</div>
```

1.10. Управление выделением текста

Для управления выделением текста предназначены следующие стилевые классы (аналоги CSS-атрибута `user-select`):

- ◆ `user-select-auto` — если элемент является редактируемым, то используется значение `contain`. Если для родительского элемента указано значение `all` (или класс `user-select-all`), то используется значение `all`. Если для родительского элемента указано значение `none` (или класс `user-select-none`), то используется значение `none`. В противном случае используется значение `text`, при котором пользователь может выделить текст. Пример:

```
<p class="user-select-auto">.user-select-auto текст абзаца</p>
<div class="user-select-none">
  <p class="user-select-auto">.user-select-auto
    текст абзаца выделить нельзя</p>
</div>
```

- ◆ `user-select-all` — текст можно выделить только целиком:
<p class="user-select-all">.user-select-all текст абзаца</p>
- ◆ `user-select-none` — пользователь не может выделить текст:
<p class="user-select-none">.user-select-none
 текст абзаца выделить нельзя</p>

1.11. Атрибут *pointer-events*

В Bootstrap 5 доступны стилевые классы `pe-none` и `pe-auto`, задающие значения для CSS-атрибута `pointer-events`:

```
.pe-none {
  pointer-events: none !important;
}
.pe-auto {
  pointer-events: auto !important;
}
```

Если добавить стилевой класс `pe-none` к тегу `<a>` или к родительскому контейнеру, то переход по ссылке становится невозможным:

```
<a href="#" class="pe-none">Текст ссылки</a>
<p class="pe-none">
  <a href="#">Текст ссылки</a>
</p>
```



ГЛАВА 2

Размещение элементов внутри окна

Библиотека Bootstrap позволяет создавать сайты, одинаково хорошо отображающиеся на всех типах устройств, независимо от ширины экрана. Благодаря адаптивной системе сеток на основе flex-контейнеров можно для различных точек останова задавать ширину колонок и их количество в одном ряду, управлять видимостью колонок, а также менять их порядок следования. И все это только с помощью CSS без использования скриптов. Простота и удобство использования этой системы сделали библиотеку Bootstrap очень популярной.

2.1. Flex-контейнеры

Система сеток основана на flex-контейнерах, поэтому предварительно рассмотрим их возможности. При изучении типов блоков (см. *разд. 1.9.1*) мы упомянули следующие стилевые классы, но оставили их без внимания:

- ◆ `d-flex` — блочный flex-контейнер.

Классы для адаптивной верстки: `d-sm-flex`, `d-md-flex`, `d-lg-flex`, `d-xl-flex`, `d-xxl-flex` (Bootstrap 5);

- ◆ `d-inline-flex` — строчный flex-контейнер.

Классы для адаптивной верстки: `d-sm-inline-flex`, `d-md-inline-flex`, `d-lg-inline-flex`, `d-xl-inline-flex`, `d-xxl-inline-flex` (Bootstrap 5).

Элементы, для которых указан стилевой класс `d-flex`, являются контейнерами, внутри которых можно задавать направление выравнивания дочерних элементов, управлять растяжением или сжатием элементов, переносом элементов на новую строку и т. д.

2.1.1. Направление выравнивания элементов внутри контейнера

Задать направление выравнивания элементов внутри flex-контейнера позволяют следующие стилевые классы (аналоги CSS-атрибута `flex-direction`):

- ◆ `flex-row` — выравнивание по горизонтали слева направо (значение по умолчанию).

Классы для адаптивной верстки: `flex-sm-row`, `flex-md-row`, `flex-lg-row`, `flex-xl-row`, `flex-xxl-row` (Bootstrap 5);

- ◆ `flex-row-reverse` — выравнивание по горизонтали справа налево.

Классы для адаптивной верстки: `flex-sm-row-reverse`, `flex-md-row-reverse`, `flex-lg-row-reverse`, `flex-xl-row-reverse`, `flex-xxl-row-reverse` (Bootstrap 5);

- ◆ `flex-column` — выравнивание по вертикали сверху вниз.

Классы для адаптивной верстки: `flex-sm-column`, `flex-md-column`, `flex-lg-column`, `flex-xl-column`, `flex-xxl-column` (Bootstrap 5);

- ◆ `flex-column-reverse` — выравнивание по вертикали снизу вверх.

Классы для адаптивной верстки: `flex-sm-column-reverse`, `flex-md-column-reverse`, `flex-lg-column-reverse`, `flex-xl-column-reverse`, `flex-xxl-column-reverse` (Bootstrap 5).

Если тег имеет параметр `dir` со значением `rtl`, то направление выравнивания по горизонтали меняется на противоположное.

Произведем выравнивание трех элементов `DIV` по горизонтали и добавим для `flex`-контейнера внутренний отступ с помощью стилевого класса `p-1` (рис. 2.1):

```
<div class="container">
  <div class="d-flex flex-row p-1">
    <div>div1</div>
    <div>div2</div>
    <div>div3</div>
  </div>
</div>
```

Все добавленные элементы по горизонтали следуют друг за другом без внешних отступов. Пробелы между элементами игнорируются. Учитывая, что значение, задаваемое стилевым классом `flex-row`, используется по умолчанию, мы можем опустить упоминание класса `flex-row`:

```
<div class="container">
  <div class="d-flex p-1">
    <div>div1</div>
    <div>div2</div>
    <div>div3</div>
  </div>
</div>
```

При добавлении элемента внутрь `flex`-контейнера с горизонтальным выравниванием его ширина по умолчанию соответствует содержимому. Если содержимое не помещается, то выполняется перенос на новую строку (рис. 2.2). По высоте элемент занимает всю доступную высоту `flex`-контейнера. Причем по умолчанию высота всех элементов внутри строки одинаковая (рис. 2.3):

```
<div class="container">
  <div class="d-flex p-1" style="height: 150px">
    <div class="p-2">div1</div>
    <div class="p-2">div2 Lorem ipsum dolor sit amet.</div>
    <div class="p-2">div3<br>Lorem ipsum dolor sit amet.</div>
  </div>
</div>
```

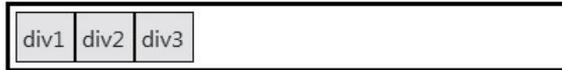


Рис. 2.1. Размещение элементов по горизонтали

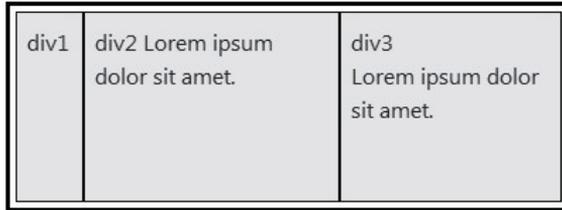


Рис. 2.2. Ширина элемента зависит от содержимого

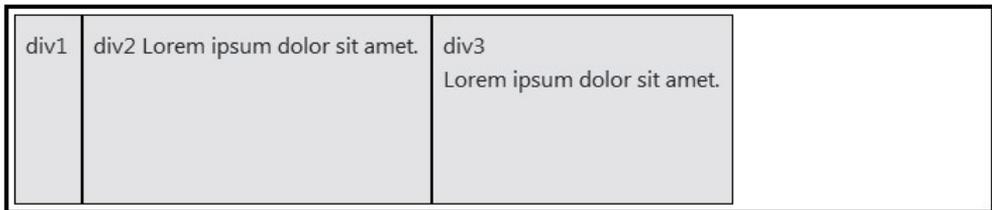


Рис. 2.3. Все элементы при горизонтальном выравнивании имеют одинаковую высоту

С помощью CSS-атрибута `width` или стилевых классов $w-*$ (см. *разд. 1.9.2*) можно явным образом задать ширину элемента. В качестве примера создадим две колонки одинаковой ширины, указав стилевой класс `w-50` (рис. 2.4):

```
<div class="container">
  <div class="d-flex p-1">
    <div class="p-2 w-50">div1</div>
    <div class="p-2 w-50">div2</div>
  </div>
</div>
```

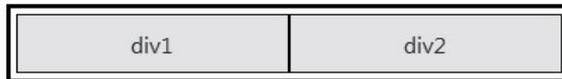


Рис. 2.4. Две колонки одинаковой ширины

Вместо стилевого класса `w-50` можно было указать класс `w-100`, который задает ширину 100%. По умолчанию, если элементы не помещаются на одной строке, то они сжимаются с одинаковыми пропорциями, поэтому ширина также станет одинаковой. Создадим три колонки одинаковой ширины, указав стилевой класс `w-100` (рис. 2.5):

```
<div class="container">
  <div class="d-flex p-1">
    <div class="p-2 w-100">div1</div>
```

```

    <div class="p-2 w-100">div2</div>
    <div class="p-2 w-100">div3</div>
  </div>
</div>

```



Рис. 2.5. Три колонки одинаковой ширины

Однако если мы включим режим переноса элементов на новую строку с помощью класса `flex-wrap`, то при указании стилевого класса `w-100` элемент займет всю ширину контейнера (рис. 2.6):

```

<div class="container">
  <div class="d-flex p-1 flex-wrap">
    <div class="p-2 w-100">div1</div>
    <div class="p-2 w-100">div2</div>
    <div class="p-2 w-100">div3</div>
  </div>
</div>

```

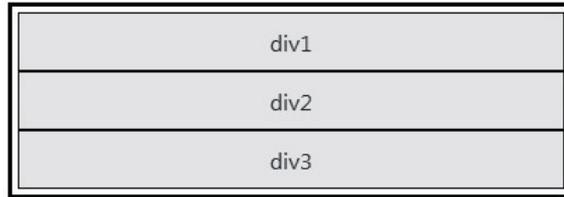


Рис. 2.6. Перенос элементов на новую строку при указании класса `w-100`



Рис. 2.7. Кнопки внутри flex-контейнера с горизонтальным выравниванием

Внутри flex-контейнера можно добавлять не только элементы `DIV`, но и любые другие элементы. В качестве примера добавим три кнопки внутри flex-контейнера с горизонтальным выравниванием (рис. 2.7):

```

<div class="container">
  <div class="d-flex">
    <button type="button" class="btn btn-outline-success">
      Кнопка 1</button>
    <button type="button" class="btn btn-outline-info">
      Кнопка 2</button>
    <button type="button" class="btn btn-outline-danger">
      Кнопка 3</button>
  </div>
</div>

```

Произведем выравнивание трех элементов `DIV` по вертикали и добавим для флекс-контейнера внутренний отступ с помощью стилевого класса `p-1`:

```
<div class="container">
  <div class="d-flex flex-column p-1">
    <div>div1</div>
    <div>div2</div>
    <div>div3</div>
  </div>
</div>
```

Все добавленные элементы по вертикали следуют друг за другом без внешних отступов. Пробелы между элементами игнорируются. При добавлении элемента внутрь флекс-контейнера с вертикальным выравниванием он занимает всю доступную ширину. Если содержимое не помещается, то выполняется перенос на новую строку. Высота элемента соответствует содержимому (рис. 2.8):

```
<div class="container">
  <div class="d-flex flex-column p-1" style="height: 200px">
    <div class="p-2">div1</div>
    <div class="p-2">div2 Lorem ipsum dolor sit amet.</div>
    <div class="p-2">div3<br>Lorem ipsum dolor sit amet.</div>
  </div>
</div>
```

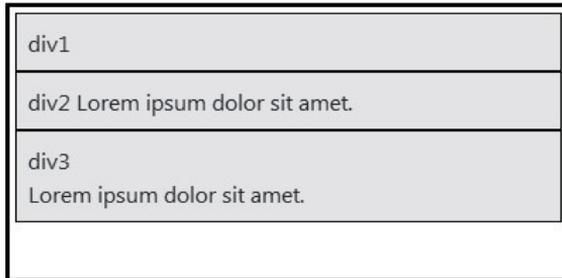


Рис. 2.8. Высота элементов зависит от содержимого

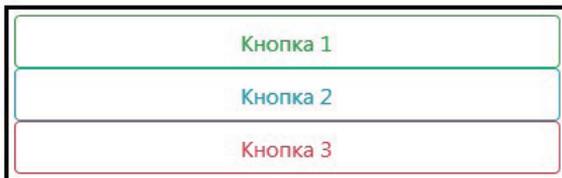


Рис. 2.9. Кнопки внутри флекс-контейнера с вертикальным выравниванием

Добавим три кнопки внутри флекс-контейнера с вертикальным выравниванием (рис. 2.9):

```
<div class="container">
  <div class="d-flex flex-column p-1">
    <button type="button" class="btn btn-outline-success">
      Кнопка 1</button>
```

```

<button type="button" class="btn btn-outline-info">
  Кнопка 2</button>
<button type="button" class="btn btn-outline-danger">
  Кнопка 3</button>
</div>
</div>

```

2.1.2. Перенос на новую строку

Управлять переносом элементов на новую строку позволяют следующие стилевые классы (аналоги CSS-атрибута `flex-wrap`):

- ◆ `flex-nowrap` — элементы выстраиваются в одну линию. Если элементы не помещаются внутри контейнера, то производится уменьшение размеров элементов (значение по умолчанию).

Классы для адаптивной верстки: `flex-sm-nowrap`, `flex-md-nowrap`, `flex-lg-nowrap`, `flex-xl-nowrap`, `flex-xxl-nowrap` (Bootstrap 5);

- ◆ `flex-wrap` — если элементы не помещаются внутри контейнера, то они переносятся на новую строку в направлении, заданном атрибутом `flex-direction`.

Классы для адаптивной верстки: `flex-sm-wrap`, `flex-md-wrap`, `flex-lg-wrap`, `flex-xl-wrap`, `flex-xxl-wrap` (Bootstrap 5);

- ◆ `flex-wrap-reverse` — если элементы не помещаются внутри контейнера, то они переносятся на новую строку в направлении, противоположном значению атрибута `flex-direction`.

Классы для адаптивной верстки: `flex-sm-wrap-reverse`, `flex-md-wrap-reverse`, `flex-lg-wrap-reverse`, `flex-xl-wrap-reverse`, `flex-xxl-wrap-reverse` (Bootstrap 5).

В этом примере элементы будут отображаться на двух строках:

```

<div class="container">
  <div class="d-flex flex-row flex-wrap"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px; height: 100px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px; height: 100px">div3</div>
  </div>
</div>

```

А в этом примере — на одной строке с уменьшением размеров элементов:

```

<div class="container">
  <div class="d-flex flex-row flex-nowrap"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px; height: 100px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
  </div>
</div>

```

```

    <div class="bg-success text-white p-2"
        style="width: 100px; height: 100px">div3</div>
  </div>
</div>

```

2.1.3. Размеры элемента

Атрибут `flex-basis` задает предпочтительную ширину элемента при горизонтальном выравнивании или предпочтительную высоту элемента при вертикальном выравнивании. Если свободного места достаточно, то элемент будет иметь указанный размер, а если нет, то его размер станет определяться значением атрибута `flex-shrink`. Если атрибут не указан, то он по умолчанию примет значение `auto`, означающее, что размер определяется содержимым элемента.

Пример:

```

<div class="container">
  <div class="d-flex flex-row"
      style="height: 400px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
        style="flex-basis: auto">div1</div>
    <div class="bg-danger text-white p-2"
        style="flex-basis: 100px">div2</div>
    <div class="bg-success text-white p-2"
        style="flex-basis: 200px">div3</div>
  </div>
</div>

```

2.1.4. Растяжение элементов

Атрибут `flex-grow` задает фактор растяжения при наличии в контейнере свободного места. По умолчанию атрибут имеет значение `0`, означающее, что размер элемента будет соответствовать предпочтительным размерам — например, заданным с помощью атрибута `flex-basis`. Значение атрибута `flex-grow` можно сравнить с жесткостью пружины, вставленной внутрь элемента. Чем больше значение, тем больше места будет занимать элемент. Если для двух элементов задано одинаковое значение, то их размеры будут равны.

Можно воспользоваться следующими стилевыми классами:

◆ `flex-grow-0` — `flex-grow: 0`.

Классы для адаптивной верстки: `flex-sm-grow-0`, `flex-md-grow-0`, `flex-lg-grow-0`, `flex-xl-grow-0`, `flex-xxl-grow-0` (Bootstrap 5);

◆ `flex-grow-1` — `flex-grow: 1`.

Классы для адаптивной верстки: `flex-sm-grow-1`, `flex-md-grow-1`, `flex-lg-grow-1`, `flex-xl-grow-1`, `flex-xxl-grow-1` (Bootstrap 5).

Пример:

```

<div class="container">
  <div class="d-flex flex-column"
      style="height: 400px; border: 3px black solid">

```

```
<div class="bg-secondary text-white p-2"
  style="flex-basis: 50px">div1</div>
<div class="bg-danger text-white p-2"
  style="flex-basis: 50px">div2</div>
<div class="bg-success text-white p-2 flex-grow-1"
  style="flex-basis: 50px">div3</div>
</div>
</div>
```

В этом примере элементы выстраиваются по вертикали сверху вниз. Для всех элементов задана предпочтительная высота 50 пх. Для элемента `div3` дополнительно указан фактор растяжения 1 (стилевой класс `flex-grow-1`). Все остальные элементы имеют фактор растяжения, равный 0. Свободной высоты контейнера достаточно, поэтому первые два элемента будут иметь высоту 50 пх, а элемент `div3` будет растянут на всю оставшуюся высоту (рис. 2.10), т. к. имеет фактор растяжения (пружину внутри себя).

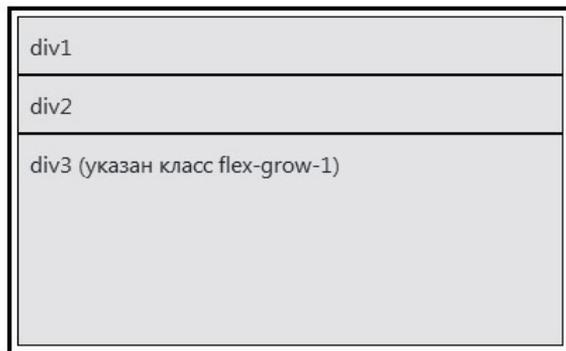


Рис. 2.10. Растяжение элементов

2.1.5. Сжатие элементов

Атрибут `flex-shrink` управляет сжатием размеров элемента при отсутствии свободного пространства. По умолчанию атрибут имеет значение 1, означающее, что при недостатке места размер элемента будет уменьшаться. Значение больше 1 задает фактор сжатия. Чем больше значение, тем сильнее сжимается элемент, освобождая пространство для других элементов. Если нужно, чтобы размеры элемента были равны предпочтительным, то следует указать значение 0.

Можно воспользоваться следующими стилиевыми классами:

◆ `flex-shrink-0` — `flex-shrink: 0`.

Классы для адаптивной верстки: `flex-sm-shrink-0`, `flex-md-shrink-0`, `flex-lg-shrink-0`, `flex-xl-shrink-0`, `flex-xxl-shrink-0` (Bootstrap 5);

◆ `flex-shrink-1` — `flex-shrink: 1`.

Классы для адаптивной верстки: `flex-sm-shrink-1`, `flex-md-shrink-1`, `flex-lg-shrink-1`, `flex-xl-shrink-1`, `flex-xxl-shrink-1` (Bootstrap 5).

Пример:

```

<div class="container">
  <div class="d-flex flex-row"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2 flex-shrink-0"
      style="flex-basis: 100px">div1</div>
    <div class="bg-danger text-white p-2 flex-shrink-1"
      style="flex-basis: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="flex-basis: 100px; flex-shrink: 3">div3</div>
  </div>
</div>

```

В этом примере элемент `div1` будет иметь предпочтительную ширину 100 пх, т. к. атрибут `flex-shrink` имеет значение 0 (указан стилевой класс `flex-shrink-0`). Оставшееся пространство будет разделено между элементами `div2` и `div3`. Причем ширина элемента `div2` будет больше ширины элемента `div3`, т. к. фактор сжатия у него меньше (рис. 2.11).



Рис. 2.11. Сжатие элементов

2.1.6. Одновременное указание характеристик элементов

Атрибут `flex` позволяет за один раз указать все характеристики элементов внутри `flex`-контейнера по следующей схеме:

```
flex: [<grow>[ <shrink> ]][<basis>]
```

Если указано одно значение с единицей измерения, то оно задает значение атрибуту `flex-basis`. Если указано одно значение без единицы измерения, то оно задает значение атрибуту `flex-grow`. Если указаны два значения, и второе с единицей измерения, то первое задает значение атрибуту `flex-grow`, а второе — атрибуту `flex-basis`. Если указаны два значения без единиц измерения, то первое задает значение атрибуту `flex-grow`, а второе — атрибуту `flex-shrink`. Можно также указать одно значение `none`, соответствующее значению `0 0 auto`.

Для значения `1 1 auto` в библиотеке Bootstrap существует стилевой класс `flex-fill`, а также классы для адаптивной верстки: `flex-sm-fill`, `flex-md-fill`, `flex-lg-fill`, `flex-xl-fill` и `flex-xxl-fill` (Bootstrap 5). В этом примере все элементы имеют одинаковые размеры и делят весь контейнер поровну (рис. 2.12), но лишь до тех пор, пока размер содержимого одинаков:

```

<div class="container">
  <div class="d-flex flex-row"
    style="height: 400px; border: 3px black solid">

```

```
<div class="bg-secondary text-white p-2 flex-fill">div1</div>  
<div class="bg-danger text-white p-2 flex-fill">div2</div>  
<div class="bg-success text-white p-2 flex-fill">div3</div>  
</div>  
</div>
```



Рис. 2.12. Результат указания стиливого класса flex-fill

2.1.7. Выравнивание элементов внутри контейнера

Задать выравнивание элементов внутри flex-контейнера по ширине (рис. 2.13) для горизонтального контейнера или по высоте для вертикального контейнера позволяют следующие стиливые классы (аналоги CSS-атрибута justify-content):

- ◆ justify-content-start — выравнивание по началу контейнера (значение по умолчанию).

Классы для адаптивной верстки: justify-content-sm-start, justify-content-md-start, justify-content-lg-start, justify-content-xl-start, justify-content-xxl-start (Bootstrap 5);

- ◆ justify-content-end — выравнивание по концу контейнера.

Классы для адаптивной верстки: justify-content-sm-end, justify-content-md-end, justify-content-lg-end, justify-content-xl-end, justify-content-xxl-end (Bootstrap 5);

- ◆ justify-content-center — выравнивание по центру контейнера.

Классы для адаптивной верстки: justify-content-sm-center, justify-content-md-center, justify-content-lg-center, justify-content-xl-center, justify-content-xxl-center (Bootstrap 5);

- ◆ justify-content-between — первый элемент прижат к началу контейнера, последний — к концу, а остальные равномерно распределяются внутри свободного пространства.

Классы для адаптивной верстки: justify-content-sm-between, justify-content-md-between, justify-content-lg-between, justify-content-xl-between, justify-content-xxl-between (Bootstrap 5);

- ◆ justify-content-around — равномерное выравнивание элементов, но перед первым и после последнего расстояние в два раза меньше, чем между остальными элементами.

Классы для адаптивной верстки: justify-content-sm-around, justify-content-md-around, justify-content-lg-around, justify-content-xl-around, justify-content-xxl-around (Bootstrap 5);

- ◆ `justify-content-evenly` — равномерное выравнивание элементов с одинаковыми пустыми пространствами. Класс доступен, начиная с Bootstrap 5.

Классы для адаптивной верстки в Bootstrap 5: `justify-content-sm-evenly`, `justify-content-md-evenly`, `justify-content-lg-evenly`, `justify-content-xl-evenly`, `justify-content-xxl-evenly`.

Пример:

```
<div class="container">
  <div class="d-flex flex-row justify-content-between"
    style="height: 400px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px; height: 100px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px; height: 100px">div3</div>
  </div>
</div>
```

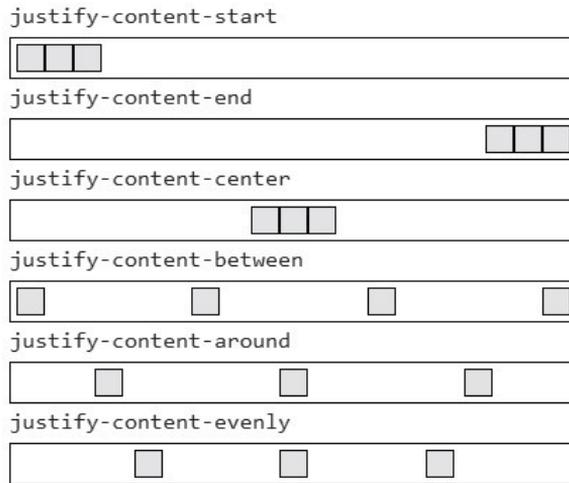


Рис. 2.13. Выравнивание элементов по ширине

Задать выравнивание элементов внутри flex-контейнера по высоте (рис. 2.14) для горизонтального контейнера или по ширине для вертикального (при условии, что атрибут `flex-wrap` имеет значение, отличное от `nowrap`) позволяют следующие стилевые классы (аналоги CSS-атрибута `align-content`):

- ◆ `align-content-stretch` — `align-content: stretch` (значение по умолчанию).

Классы для адаптивной верстки: `align-content-sm-stretch`, `align-content-md-stretch`, `align-content-lg-stretch`, `align-content-xl-stretch`, `align-content-xxl-stretch` (Bootstrap 5);

- ◆ `align-content-start` — выравнивание по началу контейнера.

Классы для адаптивной верстки: `align-content-sm-start`, `align-content-md-start`, `align-content-lg-start`, `align-content-xl-start`, `align-content-xxl-start` (Bootstrap 5);

- ◆ `align-content-end` — выравнивание по концу контейнера.

Классы для адаптивной верстки: `align-content-sm-end`, `align-content-md-end`, `align-content-lg-end`, `align-content-xl-end`, `align-content-xxl-end` (Bootstrap 5);

- ◆ `align-content-center` — выравнивание по центру контейнера.

Классы для адаптивной верстки: `align-content-sm-center`, `align-content-md-center`, `align-content-lg-center`, `align-content-xl-center`, `align-content-xxl-center` (Bootstrap 5);

- ◆ `align-content-between` — первый элемент прижат к началу контейнера, последний — к концу, а остальные равномерно распределяются внутри свободного пространства.

Классы для адаптивной верстки: `align-content-sm-between`, `align-content-md-between`, `align-content-lg-between`, `align-content-xl-between`, `align-content-xxl-between` (Bootstrap 5);

- ◆ `align-content-around` — равномерное выравнивание элементов, но перед первым и после последнего расстояние в два раза меньше, чем между остальными элементами.

Классы для адаптивной верстки: `align-content-sm-around`, `align-content-md-around`, `align-content-lg-around`, `align-content-xl-around`, `align-content-xxl-around` (Bootstrap 5).

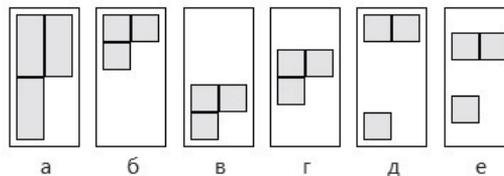


Рис. 2.14. Выравнивание элементов по высоте: *а* — `align-content-stretch`; *б* — `align-content-start`; *в* — `align-content-end`; *г* — `align-content-center`; *д* — `align-content-between`; *е* — `align-content-around`

Пример горизонтального выравнивания по центру и равномерного выравнивания по вертикали для горизонтального контейнера:

```
<div class="container">
  <div class="d-flex flex-row flex-wrap
    justify-content-center align-content-around"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px; height: 100px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
```

```

    <div class="bg-success text-white p-2"
      style="width: 100px; height: 100px">div3</div>
  </div>
</div>

```

Если для элемента горизонтального контейнера не задана высота, то при указании стилевого класса `align-content-stretch` элемент будет растягиваться по высоте:

```

<div class="container">
  <div class="d-flex flex-row flex-wrap align-content-stretch"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px">div3</div>
  </div>
</div>

```

Для выравнивания элементов можно также воспользоваться стилевыми классами, предназначенными для создания внешних отступов (см. *разд. 1.4.1*), — в частности, классами `mr-auto`, `ml-auto`, `mt-auto` и `mb-auto`. Произведем выравнивание первого элемента по левому краю горизонтального контейнера, а двух остальных элементов — по правому краю контейнера (рис. 2.15):

```

<div class="container">
  <div class="d-flex flex-row p-1">
    <div class="p-2 mr-auto">div1</div>
    <div class="p-2">div2</div>
    <div class="p-2">div3</div>
  </div>
</div>

```



Рис. 2.15. Выравнивание элементов с помощью стилевого класса `mr-auto`



Рис. 2.16. Выравнивание элементов с помощью стилевого класса `ml-auto`

Произведем выравнивание двух первых элементов по левому краю горизонтального контейнера, а третьего элемента — по правому краю контейнера (рис. 2.16):

```

<div class="container">
  <div class="d-flex flex-row p-1">
    <div class="p-2">div1</div>
    <div class="p-2">div2</div>
  </div>

```

```
<div class="p-2 ml-auto">div3</div>
</div>
</div>
```

Произведем выравнивание первого элемента по верхнему краю вертикального контейнера, а двух остальных элементов — по нижнему краю контейнера:

```
<div class="container">
  <div class="d-flex flex-column"
    style="height: 500px; border: 3px black solid">
    <div class="bg-secondary text-white p-2 mb-auto"
      style="width: 100px; height: 100px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px; height: 100px">div3</div>
  </div>
</div>
```

2.1.8. Выравнивание элементов внутри строки

Задать выравнивание элементов внутри строки по высоте (рис. 2.17) для горизонтального контейнера или по ширине для вертикального контейнера позволяют следующие стилевые классы (аналоги CSS-атрибута `align-items`):

- ◆ `align-items-stretch` — `align-items: stretch` (значение по умолчанию).
Классы для адаптивной верстки: `align-items-sm-stretch`, `align-items-md-stretch`, `align-items-lg-stretch`, `align-items-xl-stretch`, `align-items-xxl-stretch` (Bootstrap 5);
- ◆ `align-items-start` — выравнивание по началу.
Классы для адаптивной верстки: `align-items-sm-start`, `align-items-md-start`, `align-items-lg-start`, `align-items-xl-start`, `align-items-xxl-start` (Bootstrap 5);
- ◆ `align-items-end` — выравнивание по концу.
Классы для адаптивной верстки: `align-items-sm-end`, `align-items-md-end`, `align-items-lg-end`, `align-items-xl-end`, `align-items-xxl-end` (Bootstrap 5);
- ◆ `align-items-center` — выравнивание по центру.
Классы для адаптивной верстки: `align-items-sm-center`, `align-items-md-center`, `align-items-lg-center`, `align-items-xl-center`, `align-items-xxl-center` (Bootstrap 5);
- ◆ `align-items-baseline` — выравнивание по базовой линии.
Классы для адаптивной верстки: `align-items-sm-baseline`, `align-items-md-baseline`, `align-items-lg-baseline`, `align-items-xl-baseline`, `align-items-xxl-baseline` (Bootstrap 5).

Обратите внимание: атрибут `align-items` задает выравнивание внутри строки, а атрибут `align-content` — внутри flex-контейнера.

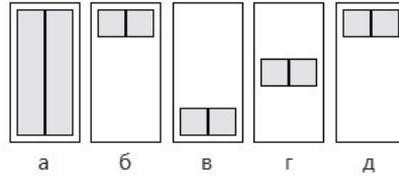


Рис. 2.17. Выравнивание элементов по высоте внутри строки:
 а — align-items-stretch; б — align-items-start; в — align-items-end;
 г — align-items-center; д — align-items-baseline

Пример выравнивания элементов по центру:

```
<div class="container">
  <div class="d-flex flex-row flex-wrap justify-content-center
    align-content-center align-items-center"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px; height: 50px">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px; height: 80px">div3</div>
  </div>
</div>
```

Если для элемента горизонтального контейнера не задана высота, то при указании стилевого класса `align-items-stretch` элемент будет растягиваться по высоте, что позволяет делать столбцы одинаковой высоты независимо от содержимого:

```
<div class="container">
  <div class="d-flex flex-row flex-nowrap
    justify-content-center align-items-stretch"
    style="height: 400px; border: 3px black solid">
    <div class="bg-secondary text-white p-2 m-2"
      style="width: 33%">div1</div>
    <div class="bg-danger text-white p-2 m-2"
      style="width: 33%">div2</div>
    <div class="bg-success text-white p-2 m-2"
      style="width: 33%">div3</div>
  </div>
</div>
```

Задать выравнивание отдельного элемента внутри строки по высоте (рис. 2.18 и 2.19) для горизонтального контейнера или по ширине для вертикального контейнера позволяют следующие стилевые классы (аналоги CSS-атрибута `align-self`, переопределяют значение атрибута `align-items`):

- ◆ `align-self-auto` — `align-self: auto` (значение по умолчанию).

Классы для адаптивной верстки: `align-self-sm-auto`, `align-self-md-auto`, `align-self-lg-auto`, `align-self-xl-auto`, `align-self-xxl-auto` (Bootstrap 5);

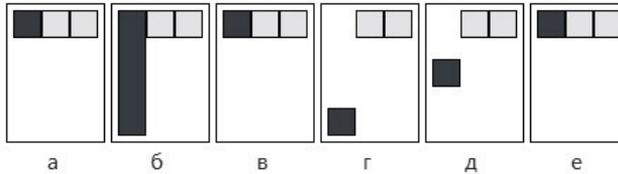


Рис. 2.18. Выравнивание первого элемента внутри строки по высоте (`align-items-start`):
 а — `align-self-auto`; б — `align-self-stretch`; в — `align-self-start`;
 г — `align-self-end`; д — `align-self-center`; е — `align-self-baseline`

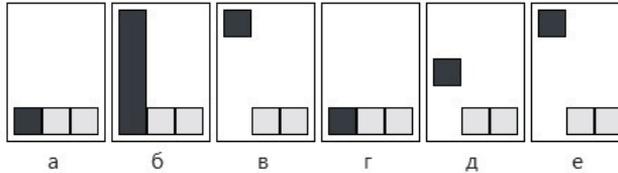


Рис. 2.19. Выравнивание первого элемента внутри строки по высоте (`align-items-end`):
 а — `align-self-auto`; б — `align-self-stretch`; в — `align-self-start`;
 г — `align-self-end`; д — `align-self-center`; е — `align-self-baseline`

◆ `align-self-stretch` — `align-self: stretch`.

Классы для адаптивной верстки: `align-self-sm-stretch`, `align-self-md-stretch`, `align-self-lg-stretch`, `align-self-xl-stretch`, `align-self-xxl-stretch` (Bootstrap 5);

◆ `align-self-start` — выравнивание по началу.

Классы для адаптивной верстки: `align-self-sm-start`, `align-self-md-start`, `align-self-lg-start`, `align-self-xl-start`, `align-self-xxl-start` (Bootstrap 5);

◆ `align-self-end` — выравнивание по концу.

Классы для адаптивной верстки: `align-self-sm-end`, `align-self-md-end`, `align-self-lg-end`, `align-self-xl-end`, `align-self-xxl-end` (Bootstrap 5);

◆ `align-self-center` — выравнивание по центру.

Классы для адаптивной верстки: `align-self-sm-center`, `align-self-md-center`, `align-self-lg-center`, `align-self-xl-center`, `align-self-xxl-center` (Bootstrap 5);

◆ `align-self-baseline` — выравнивание по базовой линии.

Классы для адаптивной верстки: `align-self-sm-baseline`, `align-self-md-baseline`, `align-self-lg-baseline`, `align-self-xl-baseline`, `align-self-xxl-baseline` (Bootstrap 5).

Пример:

```
<div class="container">
  <div class="d-flex flex-row flex-wrap justify-content-center
    align-content-center align-items-center"
    style="height: 400px; width: 250px; border: 3px black solid">
    <div class="bg-secondary text-white p-2 align-self-end"
      style="width: 100px; height: 50px">div1</div>
```

```

    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px; height: 80px">div3</div>
  </div>
</div>

```

2.1.9. Порядок следования элементов внутри контейнера

Атрибут `order` позволяет задать порядок следования элементов внутри контейнера. В качестве значения можно указать положительное или отрицательное число. Чем больше число, тем дальше будет расположен элемент. При равных значениях атрибута элементы отображаются в порядке добавления в контейнер.

Переместим первый добавленный элемент в самый конец контейнера:

```

<div class="container">
  <div class="d-flex flex-row"
    style="height: 400px; border: 3px black solid">
    <div class="bg-secondary text-white p-2"
      style="width: 100px; height: 100px; order: 3">div1</div>
    <div class="bg-danger text-white p-2"
      style="width: 100px; height: 100px; order: 1">div2</div>
    <div class="bg-success text-white p-2"
      style="width: 100px; height: 100px; order: 2">div3</div>
  </div>
</div>

```

В библиотеке Bootstrap можно воспользоваться стиливыми классами вида:

```
order[-{sm|md|lg|xl|xxl}]-{0...12|first|last}
```

Подробнее эти классы мы рассмотрим в *разд. 2.2.11*. Пример:

```

<div class="container">
  <div class="d-flex flex-row"
    style="height: 400px; border: 3px black solid">
    <div class="bg-secondary text-white p-2 order-3"
      style="width: 100px; height: 100px">div1</div>
    <div class="bg-danger text-white p-2 order-1"
      style="width: 100px; height: 100px">div2</div>
    <div class="bg-success text-white p-2 order-2"
      style="width: 100px; height: 100px">div3</div>
  </div>
</div>

```

2.2. Система сеток в библиотеке Bootstrap

Для размещения элементов внутри веб-страницы библиотека Bootstrap предоставляет свою собственную адаптивную систему сеток на основе флекс-контейнеров (см. *разд. 2.1*). Простота и удобство использования этой системы сделали библиотеку Bootstrap очень популярной. Ранние версии Bootstrap, в частности версия 3, вместо флекс-контейнеров в основе использовали блоки с CSS-атрибутом `float`. Поэтому ес-

ли нужна поддержка старых веб-браузеров, то лучше воспользоваться предыдущими версиями библиотеки Bootstrap.

НА ЗАМЕТКУ

Если вы хотите использовать только систему сеток и базовые контейнеры, то достаточно подключить файл `bootstrap-grid.min.css`.

2.2.1. Создание контейнера для строки

Чтобы создать контейнер со строкой, достаточно добавить к элементу `div` стилевой класс `row`. При этом элемент `div` нужно вложить в базовый контейнер (см. *разд. 1.1.5*).

Пример:

```
<div class="container">
  <div class="row">

  </div>
</div>
```

Стили в Bootstrap 4:

```
.row {
  display: flex;
  flex-wrap: wrap;
  margin-right: -15px;
  margin-left: -15px;
}
```

Стили в Bootstrap 5:

```
.row {
  --bs-gutter-x: 1.5rem;
  --bs-gutter-y: 0;
  display: flex;
  flex: 1 0 100%;
  flex-wrap: wrap;
  margin-top: calc(var(--bs-gutter-y) * -1);
  margin-right: calc(var(--bs-gutter-x) / -2);
  margin-left: calc(var(--bs-gutter-x) / -2);
}
```

Обратите внимание на отрицательные значения внешних отступов слева и справа. Именно поэтому строку нужно вложить в базовый контейнер, который имеет внутренние отступы точно таких же размеров. В итоге строка займет всю ширину базового контейнера. Если использовать обычный элемент `div`, то строка выйдет за рамки контейнера. Чтобы убрать отрицательные внешние отступы в Bootstrap 4, следует добавить стилевой класс `no-gutters`:

```
<div class="bg-warning">
  <div class="row no-gutters" style="height: 200px">

  </div>
</div>
```

Учитывайте, что стилевой класс `no-gutters` также убирает внутренние отступы слева и справа у колонок. Стили в Bootstrap 4:

```
.no-gutters {
  margin-right: 0;
  margin-left: 0;
}
.no-gutters > .col, .no-gutters > [class*="col-"] {
  padding-right: 0;
  padding-left: 0;
}
```

В Bootstrap 5 вместо класса `no-gutters` следует использовать следующие классы:

- ◆ `g-0` — `--bs-gutter-x: 0; --bs-gutter-y: 0.`
Классы для адаптивной верстки: `g-sm-0, g-md-0, g-lg-0, g-xl-0, g-xxl-0;`
- ◆ `g-1` — `--bs-gutter-x: 0.25rem; --bs-gutter-y: 0.25rem.`
Классы для адаптивной верстки: `g-sm-1, g-md-1, g-lg-1, g-xl-1, g-xxl-1;`
- ◆ `g-2` — `--bs-gutter-x: 0.5rem; --bs-gutter-y: 0.5rem.`
Классы для адаптивной верстки: `g-sm-2, g-md-2, g-lg-2, g-xl-2, g-xxl-2;`
- ◆ `g-3` — `--bs-gutter-x: 1rem; --bs-gutter-y: 1rem.`
Классы для адаптивной верстки: `g-sm-3, g-md-3, g-lg-3, g-xl-3, g-xxl-3;`
- ◆ `g-4` — `--bs-gutter-x: 1.5rem; --bs-gutter-y: 1.5rem.`
Классы для адаптивной верстки: `g-sm-4, g-md-4, g-lg-4, g-xl-4, g-xxl-4;`
- ◆ `g-5` — `--bs-gutter-x: 3rem; --bs-gutter-y: 3rem.`
Классы для адаптивной верстки: `g-sm-5, g-md-5, g-lg-5, g-xl-5, g-xxl-5;`
- ◆ `gx-0` — `--bs-gutter-x: 0.`
Классы для адаптивной верстки: `gx-sm-0, gx-md-0, gx-lg-0, gx-xl-0, gx-xxl-0;`
- ◆ `gx-1` — `--bs-gutter-x: 0.25rem.`
Классы для адаптивной верстки: `gx-sm-1, gx-md-1, gx-lg-1, gx-xl-1, gx-xxl-1;`
- ◆ `gx-2` — `--bs-gutter-x: 0.5rem.`
Классы для адаптивной верстки: `gx-sm-2, gx-md-2, gx-lg-2, gx-xl-2, gx-xxl-2;`
- ◆ `gx-3` — `--bs-gutter-x: 1rem.`
Классы для адаптивной верстки: `gx-sm-3, gx-md-3, gx-lg-3, gx-xl-3, gx-xxl-3;`
- ◆ `gx-4` — `--bs-gutter-x: 1.5rem.`
Классы для адаптивной верстки: `gx-sm-4, gx-md-4, gx-lg-4, gx-xl-4, gx-xxl-4;`
- ◆ `gx-5` — `--bs-gutter-x: 3rem.`
Классы для адаптивной верстки: `gx-sm-5, gx-md-5, gx-lg-5, gx-xl-5, gx-xxl-5;`
- ◆ `gy-0` — `--bs-gutter-y: 0.`
Классы для адаптивной верстки: `gy-sm-0, gy-md-0, gy-lg-0, gy-xl-0, gy-xxl-0;`

◆ gy-1 — --bs-gutter-y: 0.25rem.

Классы для адаптивной верстки: gy-sm-1, gy-md-1, gy-lg-1, gy-xl-1, gy-xxl-1;

◆ gy-2 — --bs-gutter-y: 0.5rem.

Классы для адаптивной верстки: gy-sm-2, gy-md-2, gy-lg-2, gy-xl-2, gy-xxl-2;

◆ gy-3 — --bs-gutter-y: 1rem.

Классы для адаптивной верстки: gy-sm-3, gy-md-3, gy-lg-3, gy-xl-3, gy-xxl-3;

◆ gy-4 — --bs-gutter-y: 1.5rem.

Классы для адаптивной верстки: gy-sm-4, gy-md-4, gy-lg-4, gy-xl-4, gy-xxl-4;

◆ gy-5 — --bs-gutter-y: 3rem.

Классы для адаптивной верстки: gy-sm-5, gy-md-5, gy-lg-5, gy-xl-5, gy-xxl-5.

Уберем отрицательные внешние отступы в Bootstrap 5:

```
<div class="bg-warning">
  <div class="row g-0" style="height: 200px">

    </div>
  </div>
```

Учитывайте, что стилевые классы семейства g-* также изменяют внутренние отступы слева и справа у колонок, а также внешний отступ сверху. Стили в Bootstrap 5:

```
.row > * {
  flex-shrink: 0;
  width: 100%;
  max-width: 100%;
  padding-right: calc(var(--bs-gutter-x) / 2);
  padding-left: calc(var(--bs-gutter-x) / 2);
  margin-top: var(--bs-gutter-y);
}
```

Чтобы видеть границы строки и колонок в примерах этого раздела, добавьте в раздел HEAD следующий код со стилями:

```
<style>
.row { border: 3px red solid; }
.row div { border: 1px black solid; }
</style>
```

2.2.2. Колонки одинаковой ширины

Содержимое, добавляемое в строку, описывается с помощью элемента DIV, к которому прикрепляется стилевой класс col или классы для адаптивной верстки: col-sm, col-md, col-lg, col-xl, col-xxl (Bootstrap 5) и др. Стили в Bootstrap 4.5.2:

```
.col {
  flex-basis: 0;
  flex-grow: 1;
  max-width: 100%;
```

```

position: relative;
width: 100%;
padding-right: 15px;
padding-left: 15px;
}

```

Стили в Bootstrap 5:

```

.col {
  flex: 1 0 0%;
}
.row > * {
  flex-shrink: 0;
width: 100%;
max-width: 100%;
padding-right: calc(var(--bs-gutter-x) / 2);
padding-left: calc(var(--bs-gutter-x) / 2);
margin-top: var(--bs-gutter-y);
}

```

Добавим одну колонку (рис. 2.20):

```

<div class="container bg-warning">
  <div class="row">
    <div class="col">div</div>
  </div>
</div>

```



Рис. 2.20. Одна колонка в строке

В итоге колонка заняла все доступное пространство базового контейнера по ширине и может свободно расширяться по высоте в зависимости от содержимого. Добавим еще одну колонку:

```

<div class="container bg-warning">
  <div class="row">
    <div class="col">div1</div>
    <div class="col">div2</div>
  </div>
</div>

```

Обе колонки заполнили всю доступную ширину строки и поделили ее поровну (рис. 2.21).



Рис. 2.21. Две колонки в строке

Используя классы для адаптивной верстки, можно задать точку останова, начиная с которой колонки будут размещаться по горизонтали. До этой точки останова контейнер с колонкой будет занимать всю ширину строки. Например, воспользуемся стилевым классом `col-md`:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-md">div1</div>
    <div class="col-md">div2</div>
    <div class="col-md">div3</div>
  </div>
</div>
```

До точки останова `md` (768 пх) все три колонки будут размещаться по вертикали друг под другом (рис. 2.22). Каждая колонка размещается на отдельной строке и занимает всю ее ширину. Начиная с точки останова `md`, все три колонки будут размещаться по горизонтали, разделяя всю ширину строки поровну (рис. 2.23).



Рис. 2.22. Размещение колонок до точки останова `md`



Рис. 2.23. Размещение колонок, начиная с точки останова `md`

2.2.3. Перенос колонок на новую строку

Можно добавлять сколько угодно колонок, и все они будут делить строку поровну. Учитывая, что для контейнера со строкой атрибут `flex-wrap` имеет значение `wrap`, то колонки, не помещающиеся на одной строке, будут автоматически перетекать на новую строку (рис. 2.24). Однако в Bootstrap 4 для элементов лучше явным образом задать минимальную ширину, иначе колонки будут сжиматься, т. к. минимальная ширина равна нулю (в версии 4.5.2 атрибут `min-width` со значением `0` был удален).

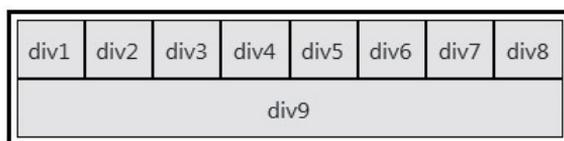


Рис. 2.24. Автоматический перенос колонок на новую строку

Чтобы принудительно перенести колонки на следующую строку, достаточно добавить элемент с классом `w-100` (`width: 100%`):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col">div1</div>
    <div class="col">div2</div>

    <div class="w-100"></div>

    <div class="col">div3</div>
    <div class="col">div4</div>
  </div>
</div>
```

Первые две колонки поделят поровну всю ширину первой строки, а последние две колонки поделят поровну всю ширину второй строки (рис. 2.25).

div1	div2
div3	div4

Рис. 2.25. Перенос колонок на новую строку с помощью элемента с классом `w-100`

2.2.4. Указание количества колонок в строке

Следующие стилевые классы позволяют задать количество колонок в строке (обратите внимание: классы указываются для контейнера со строкой, а не для отдельных колонок):

- ◆ `row-cols-1` — одна колонка в строке.

Классы для адаптивной верстки: `row-cols-sm-1`, `row-cols-md-1`, `row-cols-lg-1`, `row-cols-xl-1`, `row-cols-xxl-1` (Bootstrap 5);

- ◆ `row-cols-2` — две колонки в строке.

Классы для адаптивной верстки: `row-cols-sm-2`, `row-cols-md-2`, `row-cols-lg-2`, `row-cols-xl-2`, `row-cols-xxl-2` (Bootstrap 5);

- ◆ `row-cols-3` — три колонки в строке.

Классы для адаптивной верстки: `row-cols-sm-3`, `row-cols-md-3`, `row-cols-lg-3`, `row-cols-xl-3`, `row-cols-xxl-3` (Bootstrap 5);

- ◆ `row-cols-4` — четыре колонки в строке.

Классы для адаптивной верстки: `row-cols-sm-4`, `row-cols-md-4`, `row-cols-lg-4`, `row-cols-xl-4`, `row-cols-xxl-4` (Bootstrap 5);

- ◆ `row-cols-5` — пять колонок в строке.

Классы для адаптивной верстки: `row-cols-sm-5`, `row-cols-md-5`, `row-cols-lg-5`, `row-cols-xl-5`, `row-cols-xxl-5` (Bootstrap 5);

- ◆ `row-cols-6` — шесть колонок в строке.

Классы для адаптивной верстки: `row-cols-sm-6`, `row-cols-md-6`, `row-cols-lg-6`, `row-cols-xl-6`, `row-cols-xxl-6` (Bootstrap 5).

Выведем по две колонки в строке:

```
<div class="container bg-warning">
  <div class="row row-cols-2">
    <div class="col">div1</div>
    <div class="col">div2</div>
    <div class="col">div3</div>
    <div class="col">div4</div>
  </div>
</div>
```

Первые две колонки поделят поровну всю ширину первой строки, а последние две колонки поделят поровну всю ширину второй строки (см. рис. 2.27).

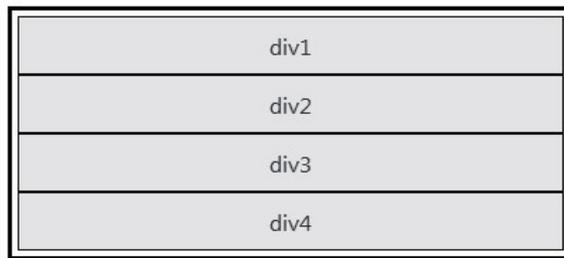


Рис. 2.26. Размещение колонок до точки останова `sm`

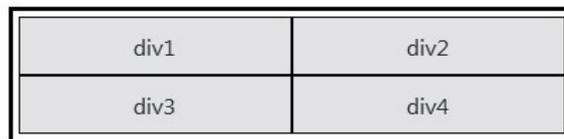


Рис. 2.27. Размещение колонок, начиная с точки останова `sm`

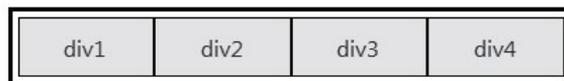


Рис. 2.28. Размещение колонок, начиная с точки останова `md`

До точки останова `sm` отобразим каждую колонку на отдельной строке (рис. 2.26). Начиная с точки останова `sm`, выведем по две колонки на строке (рис. 2.27).

Начиная с точки останова `md`, выведем по четыре колонки на строке (рис. 2.28):

```
<div class="container bg-warning">
  <div class="row row-cols-1 row-cols-sm-2 row-cols-md-4">
    <div class="col">div1</div>
    <div class="col">div2</div>
    <div class="col">div3</div>
    <div class="col">div4</div>
  </div>
</div>
```

2.2.5. Колонки автоматической ширины

Если вместо стилевого класса `col` указать класс `col-auto` или классы для адаптивной верстки: `col-sm-auto`, `col-md-auto`, `col-lg-auto`, `col-xl-auto`, `col-xxl-auto` (Bootstrap 5), то ширина колонки будет соответствовать ее содержимому (рис. 2.29):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col">div1</div>
    <div class="col-auto">Колонка автоматической ширины</div>
    <div class="col">div3</div>
  </div>
</div>
```



Рис. 2.29. Колонки автоматической ширины

В Bootstrap 5 с помощью стилевого класса `row-cols-auto` или классов для адаптивной верстки: `row-cols-sm-auto`, `row-cols-md-auto`, `row-cols-lg-auto`, `row-cols-xl-auto`, `row-cols-xxl-auto` можно указать, что все колонки внутри строки должны иметь ширину, соответствующую содержимому:

```
<!-- Только в Bootstrap 5 -->
<div class="container bg-warning">
  <div class="row row-cols-auto">
    <div class="col">Колонка автоматической ширины</div>
    <div class="col">Колонка автоматической ширины</div>
    <div class="col">Колонка автоматической ширины</div>
  </div>
</div>
```

Стили:

```
.row-cols-auto > * {
  flex: 0 0 auto;
  width: auto;
}
```

2.2.6. Колонки относительной ширины

В системе сеток Bootstrap одна строка по умолчанию содержит ровно 12 виртуальных ячеек. Используя следующие стилевые классы можно указать, сколько ячеек должна занимать колонка:

- ◆ `col-1` — колонка занимает одну ячейку.

Классы для адаптивной верстки: `col-sm-1`, `col-md-1`, `col-lg-1`, `col-xl-1`, `col-xxl-1` (Bootstrap 5);

- ◆ `col-2` — две ячейки.

Классы для адаптивной верстки: `col-sm-2`, `col-md-2`, `col-lg-2`, `col-xl-2`, `col-xxl-2` (Bootstrap 5);

- ◆ `col-3` — три ячейки.

Классы для адаптивной верстки: `col-sm-3`, `col-md-3`, `col-lg-3`, `col-xl-3`, `col-xxl-3` (Bootstrap 5);

- ◆ `col-4` — четыре ячейки.

Классы для адаптивной верстки: `col-sm-4`, `col-md-4`, `col-lg-4`, `col-xl-4`, `col-xxl-4` (Bootstrap 5);

- ◆ `col-5` — пять ячеек.

Классы для адаптивной верстки: `col-sm-5`, `col-md-5`, `col-lg-5`, `col-xl-5`, `col-xxl-5` (Bootstrap 5);

- ◆ `col-6` — шесть ячеек.

Классы для адаптивной верстки: `col-sm-6`, `col-md-6`, `col-lg-6`, `col-xl-6`, `col-xxl-6` (Bootstrap 5);

- ◆ `col-7` — семь ячеек.

Классы для адаптивной верстки: `col-sm-7`, `col-md-7`, `col-lg-7`, `col-xl-7`, `col-xxl-7` (Bootstrap 5);

- ◆ `col-8` — восемь ячеек.

Классы для адаптивной верстки: `col-sm-8`, `col-md-8`, `col-lg-8`, `col-xl-8`, `col-xxl-8` (Bootstrap 5);

- ◆ `col-9` — девять ячеек.

Классы для адаптивной верстки: `col-sm-9`, `col-md-9`, `col-lg-9`, `col-xl-9`, `col-xxl-9` (Bootstrap 5);

- ◆ `col-10` — десять ячеек.

Классы для адаптивной верстки: `col-sm-10`, `col-md-10`, `col-lg-10`, `col-xl-10`, `col-xxl-10` (Bootstrap 5);

- ◆ `col-11` — одиннадцать ячеек.

Классы для адаптивной верстки: `col-sm-11`, `col-md-11`, `col-lg-11`, `col-xl-11`, `col-xxl-11` (Bootstrap 5);

◆ `col-12` — все двенадцать ячеек.

Классы для адаптивной верстки: `col-sm-12`, `col-md-12`, `col-lg-12`, `col-xl-12`, `col-xxl-12` (Bootstrap 5).

Добавим две колонки, занимающие по 12 ячеек:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-12">div1</div>
    <div class="col-12">div2</div>
  </div>
</div>
```

В результате получим две строки по одной колонке (колонки размещены по вертикали), т. к. одна строка содержит максимум 12 ячеек (рис. 2.30).

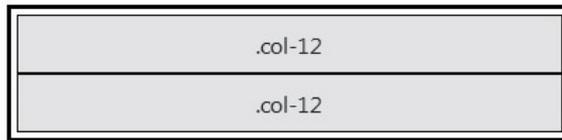


Рис. 2.30. Две колонки, занимающие по 12 ячеек

Для первой колонки зададим ширину в 4 ячейки, а для второй — в 8 ячеек:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-4">div1</div>
    <div class="col-sm-8">div2</div>
  </div>
</div>
```

В этом примере мы воспользовались классами для адаптивной верстки, поэтому до точки останова `sm` (576 px) колонки будут занимать всю ширину строки и размещаться по вертикали друг под другом. После точки останова `sm` обе колонки будут размещаться на одной строке. При этом первая колонка займет 4 ячейки, а вторая — 8 ячеек (рис. 2.31). В сумме обе колонки занимают все 12 (4 + 8) ячеек строки.



Рис. 2.31. Первая колонка занимает 4 ячейки, а вторая — 8 ячеек

Для первой колонки зададим ширину в 3 ячейки, для второй — в 6 ячеек, а для третьей — в 2 ячейки (рис. 2.32):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-3">div1</div>
    <div class="col-6">div2</div>
    <div class="col-2">div3</div>
  </div>
</div>
```



Рис. 2.32. Первая колонка занимает 3 ячейки, вторая — 6 ячеек, а третья — 2 ячейки

Мы заняли лишь 11 (3 + 6 + 2) ячеек из 12, поэтому последняя ячейка осталась пустой. При уменьшении ширины экрана колонки будут наезжать одна на другую, поэтому для маленьких экранов разумно использовать другую компоновку с помощью классов для адаптивной верстки:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-3 col-md-3">div1</div>
    <div class="col-sm-6 col-md-6">div2</div>
    <div class="col-sm-3 col-md-2">div3</div>
  </div>
</div>
```

В этом примере до точки останова `sm` колонки будут размещаться по вертикали друг под другом, занимая всю ширину строки. От точки останова `sm` до точки `md` первая колонка будет шириной в 3 ячейки, вторая — в 6 ячеек, а третья — в 3 ячейки. Для остальных точек первая колонка будет шириной в 3 ячейки, вторая — в 6 ячеек, а третья — в 2 ячейки.



Рис. 2.33. Строка содержит ровно 12 виртуальных ячеек

Добавим 14 колонок шириной в 1 ячейку (рис. 2.33). Попробуйте изменить ширину экрана, чтобы убедиться, что одна строка содержит только 12 ячеек. Две последние колонки переместятся на вторую строку. До точки останова `sm` все колонки будут размещаться по вертикали друг под другом, занимая всю ширину строки:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-1">1</div>
    <div class="col-sm-1">2</div>
    <div class="col-sm-1">3</div>
    <div class="col-sm-1">4</div>
    <div class="col-sm-1">5</div>
    <div class="col-sm-1">6</div>
    <div class="col-sm-1">7</div>
    <div class="col-sm-1">8</div>
    <div class="col-sm-1">9</div>
    <div class="col-sm-1">10</div>
    <div class="col-sm-1">11</div>
    <div class="col-sm-1">12</div>
    <div class="col-sm-1">13</div>
```

```

    <div class="col-sm-1">14</div>
  </div>
</div>

```

Надеюсь, что принцип понятен. Берем 12 ячеек строки и делим их между колонками произвольным образом так, чтобы в сумме ширина колонок была равна двенадцати. Если сумма больше, то колонки переместятся на новую строку. В следующем примере мы получим две строки. В первой строке будут заняты только 6 ячеек, а во второй — 9 ячеек, т. к. в первой строке остались пустыми только 6 ячеек (рис. 2.34):

```

<div class="container bg-warning">
  <div class="row">
    <div class="col-6">div1</div>
    <div class="col-9">div2</div>
  </div>
</div>

```

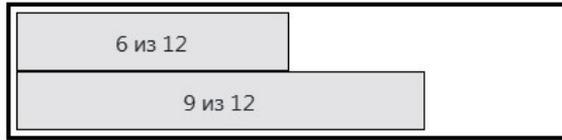


Рис. 2.34. Первая колонка занимает 6 ячеек, а вторая — 9 ячеек

Допускается смешивание внутри строки колонок разных типов. Например, для первой колонки зададим ширину в 6 ячеек, а две следующие колонки разделяют оставшееся пространство поровну (рис. 2.35):

```

<div class="container bg-warning">
  <div class="row">
    <div class="col-6">div1</div>
    <div class="col">div2</div>
    <div class="col">div3</div>
  </div>
</div>

```

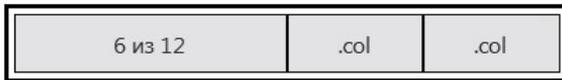


Рис. 2.35. Смешивание внутри строки колонок разных типов

Можно также с помощью CSS-атрибутов указать фиксированные размеры колонки, но тогда теряется все преимущество системы сеток:

```

<div class="container bg-warning">
  <div class="row">
    <div style="width: 100px">div1</div>
    <div class="col">div2</div>
    <div class="col">div3</div>
  </div>
</div>

```

2.2.7. Смещение колонки

Колонки внутри строки выводятся друг за другом без внешних отступов. Добавить внешний отступ слева от колонки позволяют следующие стилевые классы:

- ◆ `offset-1` — внешний отступ слева шириной в 1 ячейку.
Классы для адаптивной верстки: `offset-sm-1`, `offset-md-1`, `offset-lg-1`, `offset-xl-1`, `offset-xxl-1` (Bootstrap 5);
- ◆ `offset-2` — внешний отступ слева шириной в 2 ячейки.
Классы для адаптивной верстки: `offset-sm-2`, `offset-md-2`, `offset-lg-2`, `offset-xl-2`, `offset-xxl-2` (Bootstrap 5);
- ◆ `offset-3` — внешний отступ слева шириной в 3 ячейки.
Классы для адаптивной верстки: `offset-sm-3`, `offset-md-3`, `offset-lg-3`, `offset-xl-3`, `offset-xxl-3` (Bootstrap 5);
- ◆ `offset-4` — внешний отступ слева шириной в 4 ячейки.
Классы для адаптивной верстки: `offset-sm-4`, `offset-md-4`, `offset-lg-4`, `offset-xl-4`, `offset-xxl-4` (Bootstrap 5);
- ◆ `offset-5` — внешний отступ слева шириной в 5 ячеек.
Классы для адаптивной верстки: `offset-sm-5`, `offset-md-5`, `offset-lg-5`, `offset-xl-5`, `offset-xxl-5` (Bootstrap 5);
- ◆ `offset-6` — внешний отступ слева шириной в 6 ячеек.
Классы для адаптивной верстки: `offset-sm-6`, `offset-md-6`, `offset-lg-6`, `offset-xl-6`, `offset-xxl-6` (Bootstrap 5);
- ◆ `offset-7` — внешний отступ слева шириной в 7 ячеек.
Классы для адаптивной верстки: `offset-sm-7`, `offset-md-7`, `offset-lg-7`, `offset-xl-7`, `offset-xxl-7` (Bootstrap 5);
- ◆ `offset-8` — внешний отступ слева шириной в 8 ячеек.
Классы для адаптивной верстки: `offset-sm-8`, `offset-md-8`, `offset-lg-8`, `offset-xl-8`, `offset-xxl-8` (Bootstrap 5);
- ◆ `offset-9` — внешний отступ слева шириной в 9 ячеек.
Классы для адаптивной верстки: `offset-sm-9`, `offset-md-9`, `offset-lg-9`, `offset-xl-9`, `offset-xxl-9` (Bootstrap 5);
- ◆ `offset-10` — внешний отступ слева шириной в 10 ячеек.
Классы для адаптивной верстки: `offset-sm-10`, `offset-md-10`, `offset-lg-10`, `offset-xl-10`, `offset-xxl-10` (Bootstrap 5);
- ◆ `offset-11` — внешний отступ слева шириной в 11 ячеек.
Классы для адаптивной верстки: `offset-sm-11`, `offset-md-11`, `offset-lg-11`, `offset-xl-11`, `offset-xxl-11` (Bootstrap 5).

Произведем выравнивание колонки шириной 6 ячеек по правому краю строки (рис. 2.36):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-6 offset-6">div</div>
  </div>
</div>
```



Рис. 2.36. Выравнивание колонки шириной 6 ячеек по правому краю строки

Начиная от точки останова `sm`, между двумя первыми колонками добавим отступ в 1 ячейку, а между двумя последними колонками — отступ в 2 ячейки (рис. 2.37):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-3">div1</div>
    <div class="col-sm-3 offset-sm-1">div2</div>
    <div class="col-sm-3 offset-sm-2">div3</div>
  </div>
</div>
```



Рис. 2.37. Добавление отступов между колонками

Убрать внешний отступ слева на отдельных точках останова позволяют стилевые классы `offset-sm-0`, `offset-md-0`, `offset-lg-0`, `offset-xl-0` и `offset-xxl-0` (Bootstrap 5). Начиная от точки останова `lg`, увеличим ширину средней колонки на одну ячейку и уберем внешний отступ:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-3">div1</div>
    <div class="col-sm-3 offset-sm-1 col-lg-4 offset-lg-0">div2</div>
    <div class="col-sm-3 offset-sm-2">div3</div>
  </div>
</div>
```

Вместо стилевых классов семейства `offset-*` для создания смещения можно воспользоваться семейством стилевых классов `ml-auto`, `ml-*-auto`, `mr-auto`, `mr-*-auto` (см. *разд. 1.4.1*). Начиная от точки останова `sm`, произведем выравнивание двух колонок по разным сторонам строки (рис. 2.38):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-3 mr-sm-auto">div1</div>
    <div class="col-sm-3">div2</div>
  </div>
</div>
```



Рис. 2.38. Выравнивание двух колонок по разным сторонам строки

Произведем выравнивание колонки шириной 6 ячеек по правому краю строки (рис. 2.39):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-6 ml-auto">div</div>
  </div>
</div>
```



Рис. 2.39. Выравнивание колонки шириной 6 ячеек по правому краю строки

2.2.8. Горизонтальное выравнивание колонок внутри строки

Для горизонтального выравнивания колонок внутри строки можно воспользоваться стилевыми классами семейства `justify-content-*` (см. *разд. 2.1.7*). Произведем выравнивание колонки шириной 6 ячеек по правому краю строки (см. рис. 2.39):

```
<div class="container bg-warning">
  <div class="row justify-content-end">
    <div class="col-6">div</div>
  </div>
</div>
```

Пример выравнивания колонок по центру строки (рис. 2.40):

```
<div class="container bg-warning">
  <div class="row justify-content-center">
    <div class="col-4">div1</div>
    <div class="col-4">div2</div>
  </div>
</div>
```



Рис. 2.40. Выравнивание колонок по центру строки

Начиная от точки останова `sm`, произведем выравнивание двух колонок по разным сторонам строки (рис. 2.41), а начиная от точки останова `md`, выполним равномерное выравнивание (рис. 2.42):

```
<div class="container bg-warning">
  <div class="row justify-content-sm-between
    justify-content-md-around">
```

```

<div class="col-sm-4">div1</div>
<div class="col-sm-4">div2</div>
</div>
</div>

```



Рис. 2.41. Выравнивание двух колонок по разным сторонам строки



Рис. 2.42. Равномерное выравнивание колонок

2.2.9. Вертикальное выравнивание всех колонок внутри строки

Для вертикального выравнивания всех колонок внутри строки можно воспользоваться стилевыми классами семейства `align-items-*` (см. *разд. 2.1.8*). Произведем выравнивание колонки шириной 4 ячейки по центру строки и по вертикали, и по горизонтали (рис. 2.43):

```

<div class="container bg-warning">
  <div class="row justify-content-center align-items-center"
    style="min-height: 200px">
    <div class="col-4">div</div>
  </div>
</div>

```

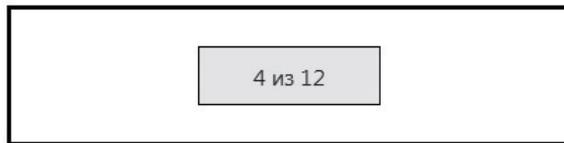


Рис. 2.43. Выравнивание колонки по центру

Начиная от точки останова `sm`, произведем горизонтальное выравнивание двух колонок по правой стороне строки, а вертикальное их выравнивание — по низу строки (рис. 2.44):

```

<div class="container bg-warning">
  <div class="row justify-content-sm-end align-items-sm-end"
    style="min-height: 200px">
    <div class="col-sm-4">div1</div>
    <div class="col-sm-4">div2</div>
  </div>
</div>

```

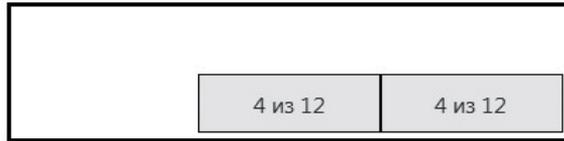


Рис. 2.44. Горизонтальное выравнивание двух колонок по правой стороне строки, а вертикальное их выравнивание — по низу строки

2.2.10. Вертикальное выравнивание одной колонки внутри строки

Для вертикального выравнивания одной колонки внутри строки можно воспользоваться стилевыми классами семейства `align-self-*` (см. *разд. 2.1.8*). Начиная от точки останова `sm`, произведем вертикальное выравнивание первой колонки по верху строки, второй — по центру строки, а третьей — по низу строки (рис. 2.45):

```
<div class="container bg-warning">
  <div class="row" style="min-height: 200px">
    <div class="col-sm-4 align-self-sm-start">div1</div>
    <div class="col-sm-4 align-self-sm-center">div2</div>
    <div class="col-sm-4 align-self-sm-end">div3</div>
  </div>
</div>
```

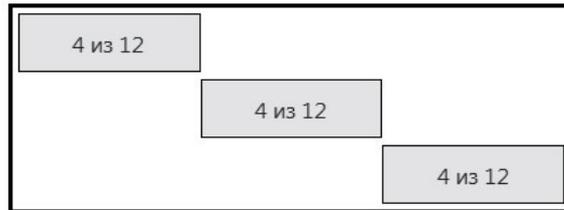


Рис. 2.45. Вертикальное выравнивание каждой колонки по отдельности

2.2.11. Порядок следования колонок внутри контейнера

По умолчанию порядок следования колонок совпадает с порядком их добавления в строку. Следующие стилевые классы позволяют изменить порядок следования:

- ◆ `order-first` — в начало строки (`order: -1`).

Классы для адаптивной верстки: `order-sm-first`, `order-md-first`, `order-lg-first`, `order-xl-first`, `order-xxl-first` (Bootstrap 5);

- ◆ `order-last` — в конец строки (`order: 13` в Bootstrap 4 или `order: 6` в Bootstrap 5).

Классы для адаптивной верстки: `order-sm-last`, `order-md-last`, `order-lg-last`, `order-xl-last`, `order-xxl-last` (Bootstrap 5);

- ◆ `order-0` — `order: 0`.

Классы для адаптивной верстки: `order-sm-0`, `order-md-0`, `order-lg-0`, `order-xl-0`, `order-xxl-0` (Bootstrap 5);

- ◆ `order-1` — `order: 1`.

Классы для адаптивной верстки: `order-sm-1`, `order-md-1`, `order-lg-1`, `order-xl-1`, `order-xxl-1` (Bootstrap 5);

- ◆ `order-2` — `order: 2`.

Классы для адаптивной верстки: `order-sm-2`, `order-md-2`, `order-lg-2`, `order-xl-2`, `order-xxl-2` (Bootstrap 5);

- ◆ `order-3` — `order: 3`.

Классы для адаптивной верстки: `order-sm-3`, `order-md-3`, `order-lg-3`, `order-xl-3`, `order-xxl-3` (Bootstrap 5);

- ◆ `order-4` — `order: 4`.

Классы для адаптивной верстки: `order-sm-4`, `order-md-4`, `order-lg-4`, `order-xl-4`, `order-xxl-4` (Bootstrap 5);

- ◆ `order-5` — `order: 5`.

Классы для адаптивной верстки: `order-sm-5`, `order-md-5`, `order-lg-5`, `order-xl-5`, `order-xxl-5` (Bootstrap 5);

- ◆ `order-6` — `order: 6` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-6`, `order-md-6`, `order-lg-6`, `order-xl-6`;

- ◆ `order-7` — `order: 7` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-7`, `order-md-7`, `order-lg-7`, `order-xl-7`;

- ◆ `order-8` — `order: 8` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-8`, `order-md-8`, `order-lg-8`, `order-xl-8`;

- ◆ `order-9` — `order: 9` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-9`, `order-md-9`, `order-lg-9`, `order-xl-9`;

- ◆ `order-10` — `order: 10` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-10`, `order-md-10`, `order-lg-10`, `order-xl-10`;

- ◆ `order-11` — `order: 11` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-11`, `order-md-11`, `order-lg-11`, `order-xl-11`;

- ◆ `order-12` — `order: 12` (только в Bootstrap 4).

Классы для адаптивной верстки в Bootstrap 4: `order-sm-12`, `order-md-12`, `order-lg-12`, `order-xl-12`.

Чем больше число, тем дальше будет расположена колонка. По умолчанию при равных значениях CSS-атрибута `order` колонки отображаются в порядке добавления в строку. Переместим первую добавленную колонку в самый конец строки (рис. 2.46):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-4 order-last">div1</div>
    <div class="col-4">div2</div>
    <div class="col-4">div3</div>
  </div>
</div>
```



Рис. 2.46. Изменение порядка следования колонок

Начиная от точки останова `md`, изменим порядок следования колонок на противоположный (рис. 2.47 и 2.48):

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-4 order-md-3">div1</div>
    <div class="col-sm-4 order-md-2">div2</div>
    <div class="col-sm-4 order-md-1">div3</div>
  </div>
</div>
```



Рис. 2.47. Порядок следования колонок до точки останова `md`



Рис. 2.48. Порядок следования колонок, начиная от точки останова `md`

2.2.12. Вложенные сетки

Одну сетку можно вложить в колонку другой сетки (рис. 2.49), создавая вложенные структуры:

```
<div class="container bg-warning">
  <div class="row">
    <div class="col-sm-2">Level 1: div1</div>
    <div class="col-sm-2">Level 1: div2</div>
    <div class="col-sm-8">
      Вложенная сетка
      <div class="row">
```

```

<div class="col-sm-3">Level 2: div3</div>
<div class="col-sm-9">Level 2: div4</div>
</div>
</div>
</div>
</div>

```

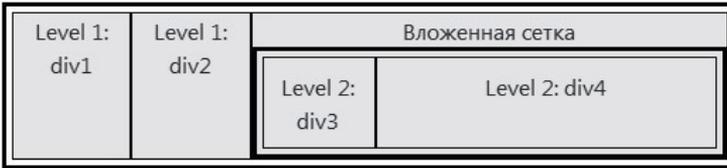


Рис. 2.49. Вложенные сетки

Комбинируя несколько отдельных строк с различным размещением колонок (рис. 2.50) можно получить дизайн страницы произвольной сложности. Если дополнительно указать классы с точками останова, то дизайн страницы будет адаптироваться под ширину экрана пользователя. Согласитесь, что система сеток библиотеки Bootstrap очень проста и удобна! В последующих разделах мы еще не раз используем ее в примерах, требующих выравнивания элементов, — например, для выравнивания элементов формы в следующей главе.

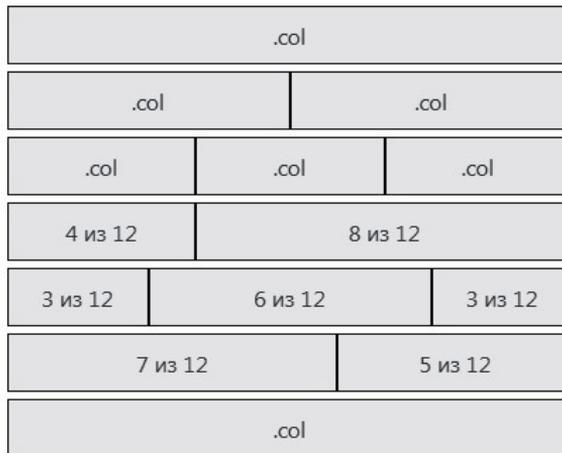


Рис. 2.50. Комбинация нескольких отдельных строк с различной шириной колонок



ГЛАВА 3

Формы и элементы управления

Для взаимодействия с пользователем предназначены элементы управления — такие как кнопки, текстовые поля, списки со значениями, флажки, переключатели и др. Элементы управления могут использоваться как отдельно, так и в составе формы, предназначенной для отправки на сервер данных, введенных пользователем. Библиотека Bootstrap предоставляет множество стилевых классов, позволяющих сделать стандартные элементы удобными и красивыми, а также несколько нестандартных элементов: выключатели, кнопки-переключатели и кнопки с выпадающим меню.

3.1. Элементы управления

Форма добавляется в HTML-документ при помощи парного тега `<form>`. Внутри тегов `<form>` и `</form>` могут располагаться теги `<input>`, `<textarea>`, `<select>` и др., вставляющие в форму элементы управления.

3.1.1. Командные кнопки

Кнопку можно вставить с помощью тега `<input>`, в параметре `type` которого указаны значения `button` (обычная командная кнопка), `reset` (кнопка сброса) или `submit` (кнопка, при нажатии которой происходит отправка данных формы):

```
<input type="button" value="OK" onclick="alert('OK')">
<input type="reset" value="Очистить">
<input type="submit" value="Отправить">
```

Помимо тега `<input>` для вставки кнопки применяется также тег `<button>`, причем в библиотеке Bootstrap он используется гораздо чаще:

```
<button type="button" onclick="alert('OK')">OK</button>
<button type="reset">Очистить</button>
<button type="submit">Отправить</button>
```

Внутри тегов `<button>` и `</button>` могут быть различные элементы или просто текст, задающие содержимое, отображаемое на кнопке. Пример вывода значка и текста:

```
<button type="submit">
  
  Отправить
</button>
```

По умолчанию кнопки имеют стандартный вид, который зависит от веб-браузера. Чтобы применить стилизацию из библиотеки Bootstrap, следует добавить стилевой класс `btn` (его лучше использовать с тегом `<button>`):

```
<button type="button" class="btn">Текст</button>
```

Класс `btn` является лишь базовым, поэтому кнопка имеет прозрачный фон. Задать цвет фона кнопки и цвет надписи позволяют следующие стилевые классы:

- ◆ `btn-dark` — темно-серый цвет фона и белый цвет надписи;
- ◆ `btn-light` — светло-серый цвет фона и черный цвет надписи;
- ◆ `btn-success` — зеленый цвет фона и белый цвет надписи;
- ◆ `btn-info` — информационная кнопка с белым цветом надписи;
- ◆ `btn-warning` — желтый цвет фона и черный цвет надписи;
- ◆ `btn-danger` — красный цвет фона и белый цвет надписи;
- ◆ `btn-primary` — синий цвет фона и белый цвет надписи (рис. 3.1);
- ◆ `btn-secondary` — серый цвет фона и белый цвет надписи.

При наведении указателя мыши на кнопку цвет фона становится темнее. Пример:

```
<input type="button" value="OK" class="btn btn-success">
<input type="reset" value="Очистить" class="btn btn-info">
<input type="submit" value="Отправить" class="btn btn-primary">
<button type="button" class="btn btn-dark">.btn-dark</button>
<button type="button" class="btn btn-light">.btn-light</button>
<button type="button" class="btn btn-success">.btn-success</button>
<button type="button" class="btn btn-info">.btn-info</button>
<button type="button" class="btn btn-warning">.btn-warning</button>
<button type="button" class="btn btn-danger">.btn-danger</button>
<button type="button" class="btn btn-primary">.btn-primary</button>
<button type="button" class="btn btn-secondary">.btn-secondary</button>
```

При получении фокуса ввода цвет фона становится темнее, а вокруг кнопки отображается рамка и тень. С помощью стилевого класса `focus` в Bootstrap 4 можно явно указать, что кнопка находится в фокусе ввода (хотя на самом деле фокус может иметь другой элемент):

```
<!-- Только в Bootstrap 4 -->
<button type="button" class="btn btn-dark focus">.btn-dark focus</button>
```

Для имитации нажатого состояния нужно добавить стилевой класс `active`:

```
<button type="button"
  class="btn btn-outline-success active">Кнопка</button>
```

Если указан параметр `disabled`, то доступ к кнопке будет запрещен. При наличии этого параметра кнопка имеет более бледные (полупрозрачные) цвета и не может получить фокус ввода. С помощью стилевого класса `disabled` можно имитировать запрет доступа к кнопке:

```
<button type="button"
  class="btn btn-success" disabled>.btn-success</button>
<button type="button" class="btn btn-info disabled">.btn-info</button>
```

Обратите внимание: первая кнопка недоступна для нажатия, а вот вторая, хоть и отображается как недоступная, все равно будет работать.



Рис. 3.1. Кнопки со стилевым классом btn-primary

Следующие стилевые классы позволяют отобразить только контур кнопки (цветная рамка с прозрачным фоном и цветной надписью):

- ◆ btn-outline-dark — темно-серый цвет рамки и надписи;
- ◆ btn-outline-light — светло-серый цвет рамки и надписи;
- ◆ btn-outline-success — зеленый цвет рамки и надписи;
- ◆ btn-outline-info — информационная кнопка (рис. 3.2);
- ◆ btn-outline-warning — желтый цвет рамки и надписи;
- ◆ btn-outline-danger — красный цвет рамки и надписи;
- ◆ btn-outline-primary — синий цвет рамки и надписи;
- ◆ btn-outline-secondary — серый цвет рамки и надписи.

При наведении указателя мыши на кнопку цвет фона будет соответствовать цвету рамки, при этом цвет текста становится или белым, или черным. Пример:

```
<input type="button" value="OK" class="btn btn-outline-success">
<input type="reset" value="Очистить" class="btn btn-outline-info">
<input type="submit" value="Отправить" class="btn btn-outline-primary">
<button type="button"
  class="btn btn-outline-dark">.btn-outline-dark</button>
<button type="button"
  class="btn btn-outline-light">.btn-outline-light</button>
<button type="button"
  class="btn btn-outline-success">.btn-outline-success</button>
<button type="button"
  class="btn btn-outline-info">.btn-outline-info</button>
<button type="button"
  class="btn btn-outline-warning">.btn-outline-warning</button>
<button type="button"
  class="btn btn-outline-danger">.btn-outline-danger</button>
<button type="button"
  class="btn btn-outline-primary">.btn-outline-primary</button>
<button type="button"
  class="btn btn-outline-secondary">.btn-outline-secondary</button>
```



Рис. 3.2. Кнопки со стилевым классом btn-outline-info

Следующие стилевые классы позволяют указать размеры кнопки (рис. 3.3):

- ◆ `btn-sm` — высота меньше, чем у обычной кнопки;
- ◆ `btn-lg` — высота больше, чем у обычной кнопки;
- ◆ `btn-block` — кнопка занимает всю доступную ширину. Если несколько кнопок со стилем `btn-block` располагаются подряд, то добавляется внешний отступ сверху, равный `0.5 rem`.

Пример:

```
<button type="button" class="btn btn-success btn-sm">Маленькая</button>
<button type="button" class="btn btn-info">Обычная</button>
<button type="button"
  class="btn btn-danger btn-lg mb-2">Большая</button>
<button type="button"
  class="btn btn-primary btn-block">.btn-block</button>
<button type="button"
  class="btn btn-primary btn-block">.btn-block</button>
```



Рис. 3.3. Размеры кнопок

Если нужно изменить высоту сразу для нескольких кнопок, то следует вложить их в контейнер и добавить к нему стилевой класс `btn-group-sm` или `btn-group-lg`:

```
<div class="btn-group-sm mb-2">
  <button type="button" class="btn btn-success">Маленькая</button>
  <button type="button" class="btn btn-info">Маленькая</button>
  <button type="button" class="btn btn-danger">Маленькая</button>
</div>
<div class="btn-group-lg">
  <button type="button" class="btn btn-success">Большая</button>
  <button type="button" class="btn btn-info">Большая</button>
  <button type="button" class="btn btn-danger">Большая</button>
</div>
```

Если добавить стилевой класс `btn-link`, то кнопка превратится в гиперссылку:

```
<button type="button" class="btn btn-link">.btn-link</button>
<button type="button"
  class="btn btn-link" disabled>.btn-link:disabled</button>
```

Если же добавить к гиперссылке стилевой класс `btn`, то она превратится в кнопку:

```
<a class="btn btn-success" href="#" role="button">btn</a>
```

3.1.2. Поля для ввода данных

Вставить поле для ввода данных позволяет тег `<input>`. В параметре `type` указываются следующие значения:

- ◆ `text` — текстовое поле ввода:
`<input type="text">`
- ◆ `password` — текстовое поле для ввода пароля, в котором все вводимые символы заменяются точками:
`<input type="password">`
- ◆ `url` — поле ввода URL-адреса. Значение автоматически проверяется. Форма отправляется только в случае, если поле не заполнено или содержит корректное значение URL-адреса. Существование URL-адреса не проверяется. Если нужно исключить пустое значение, то следует дополнительно указать параметр `required`:
`<input type="url" required>`
- ◆ `email` — поле ввода адреса электронной почты. Значение автоматически проверяется. Форма отправляется только в случае, если поле не заполнено или содержит корректное значение адреса. Существование E-mail не проверяется. Если нужно исключить пустое значение, то следует дополнительно указать параметр `required`:
`<input type="email" required>`
- ◆ `tel` — поле для ввода телефона. Значение автоматически не проверяется, т. к. форматы номеров сильно различаются. Мобильные веб-браузеры для этого поля могут отобразить специальную клавиатуру. В остальных веб-браузерах элемент выглядит как обычное текстовое поле:
`<input type="tel" placeholder="+0 (000) 000-00-00">`
- ◆ `number` — поле для ввода числа. Справа от поля веб-браузер отображает две кнопки, с помощью которых можно увеличить или уменьшить значение на шаг, указанный в параметре `step`. Мобильные веб-браузеры для этого поля отображают цифровую клавиатуру. Значение автоматически проверяется. Форма отправляется только в случае, если поле не заполнено или содержит положительное или отрицательное число. Если нужно исключить пустое значение, то следует дополнительно указать параметр `required`:
`<input type="number" step="5" required>`
`<input type="number" step="0.5" required>`
- ◆ `search` — поле ввода подстроки для поиска:
`<input type="search">`
- ◆ `date` — поле для ввода даты (элемент не поддерживается Firefox и Internet Explorer):
`<input type="date">`
- ◆ `time` — поле для ввода времени (элемент не поддерживается Firefox и Internet Explorer):
`<input type="time">`

- ◆ `datetime-local` — поле для ввода локальной даты и времени (элемент не поддерживается Firefox и Internet Explorer):
`<input type="datetime-local">`
- ◆ `month` — поле для ввода месяца и года (элемент не поддерживается Firefox и Internet Explorer):
`<input type="month">`
- ◆ `week` — поле для ввода номера недели и года (элемент не поддерживается Firefox и Internet Explorer):
`<input type="week">`

ПРИМЕЧАНИЕ

Получить полную информацию о текущей поддержке тегов и параметров веб-браузерами можно на сайте <https://caniuse.com/>.

Парный тег `<textarea>` создает внутри формы поле для ввода многострочного текста. В окне веб-браузера поле отображается в виде прямоугольной области с полными прокрутки:

```
<textarea>
Текст по умолчанию
</textarea>
```

Чтобы применить стилизацию из библиотеки Bootstrap к полям для ввода данных, следует добавить стилевой класс `form-control` (рис. 3.4):

```
<div class="container">
  <input type="text" class="form-control my-2">
  <input type="password" class="form-control mb-2">
  <input type="email" class="form-control mb-2" required>
  <textarea class="form-control"></textarea>
</div>
```

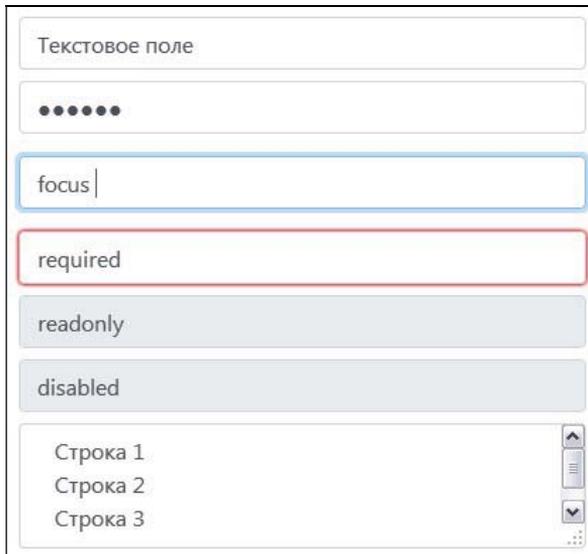


Рис. 3.4. Текстовые поля со стилевым классом `form-control`

В результате поля займут всю ширину родительского контейнера. При получении фокуса ввода границы поля будут подсвечиваться. Если поле, обязательное для заполнения (имеет параметр `required`), не содержит данных или данные введены неправильно, то границы поля станут красного цвета.

В предыдущем примере мы добавили внешние отступы с помощью стилевых классов `my-2` и `mb-2`, чтобы поля по вертикали не соприкасались друг с другом. Однако в Bootstrap 4 лучше вложить каждое поле в контейнер и добавить к нему стилевой класс `form-group`, который задает внешний отступ снизу:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <div class="form-group">
    <input type="text" class="form-control">
  </div>
  <div class="form-group">
    <input type="password" class="form-control">
  </div>
  <div class="form-group">
    <input type="email" class="form-control" required>
  </div>
  <div class="form-group">
    <textarea class="form-control"></textarea>
  </div>
</div>
```

В Bootstrap 5 класс `form-group` был удален, вместо него нужно использовать класс `mb-3`:

```
<div class="container">
  <div class="mb-3">
    <input type="text" class="form-control">
  </div>
  <div class="mb-3">
    <input type="password" class="form-control">
  </div>
  <div class="mb-3">
    <input type="email" class="form-control" required>
  </div>
  <div class="mb-3">
    <textarea class="form-control"></textarea>
  </div>
</div>
```

Если к стилевому классу `form-control` добавить класс `form-control-sm`, то текст внутри поля будет выводиться уменьшенным шрифтом, а если добавить класс `form-control-lg` — то увеличенным шрифтом (рис. 3.5):

```
<div class="container">
  <div class="mb-3">
    <input type="text" value=".form-control-sm"
      class="form-control form-control-sm">
  </div>
```

```

<div class="mb-3">
  <input type="text" value=".form-control"
    class="form-control">
</div>
<div class="mb-3">
  <input type="text" value=".form-control-lg"
    class="form-control form-control-lg">
</div>
</div>

```

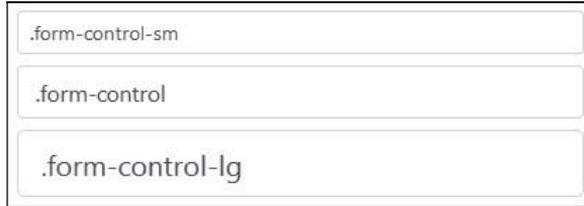


Рис. 3.5. Размеры текстовых полей

3.1.3. Поле, доступное только для чтения

Поле можно сделать доступным только для чтения. Для этого нужно добавить параметр `readonly` и параметр `value` со значением, отображаемым внутри поля:

```

<input type="text" value="Текст только для чтения"
  class="form-control" readonly>

```

Чтобы содержимое поля, доступного только для чтения, отображалось просто в виде текста без рамки вокруг, следует вместо стилевого класса `form-control` использовать класс `form-control-plaintext`:

```

<input type="text" value="Текст только для чтения"
  class="form-control-plaintext" readonly>

```

Если к стилевому классу `form-control-plaintext` добавить класс `form-control-sm`, то текст будет выводиться уменьшенным шрифтом, а если добавить класс `form-control-lg` — то увеличенным шрифтом:

```

<input type="text" value=".form-control-sm"
  class="form-control-plaintext form-control-sm" readonly>
<input type="text" value=".form-control-plaintext"
  class="form-control-plaintext" readonly>
<input type="text" value=".form-control-lg"
  class="form-control-plaintext form-control-lg" readonly>

```

3.1.4. Вывод пояснительной надписи

С помощью параметра `placeholder` можно внутри поля отобразить текст подсказки, который будет выводиться прямо в поле ввода, пока оно не заполнено:

```

<div class="container">
  <div class="mb-3">
    <input type="password" class="form-control"
      placeholder="Введите пароль">

```

```

</div>
<div class="mb-3">
  <textarea class="form-control"
    placeholder="Введите сообщение"></textarea>
</div>
</div>

```

Указать пояснительную надпись для поля ввода (и любого другого элемента формы) позволяет тег `<label>`. В параметре `for` нужно задать идентификатор элемента, к которому привязана надпись. Точно такой же идентификатор должен быть указан в параметре `id` элемента формы:

```

<!-- Только в Bootstrap 4 -->
<div class="form-group">
  <label for="passwd">Пароль:</label>
  <input type="password" class="form-control"
    name="passwd" id="passwd" placeholder="Введите пароль">
</div>

```

Стили для тега `<label>`:

```

label {
  display: inline-block;
  margin-bottom: 0.5rem; /* Только в Bootstrap 4 */
}

```

Чтобы добавить внешний отступ для пояснительной надписи снизу в Bootstrap 5, следует использовать стилевой класс `form-label`:

```

<div class="mb-3">
  <label for="passwd" class="form-label">Пароль:</label>
  <input type="password" class="form-control"
    name="passwd" id="passwd" placeholder="Введите пароль">
</div>

```

Если пояснительная надпись не помещается, например по соображениям дизайна, то ее можно скрыть, но оставить доступной для различных программ чтения с экрана. Для этого следует добавить стилевой класс `sr-only`:

```

<div class="mb-3">
  <label for="passwd" class="sr-only">Пароль:</label>
  <input type="password" class="form-control"
    name="passwd" id="passwd" placeholder="Введите пароль">
</div>

```

Ниже поля можно отобразить дополнительную информацию шрифтом меньшего размера (рис. 3.6), добавив после поля тег `<small>` со стилевыми классами `form-text` и `text-muted` (задает серый цвет шрифта):

```

<div class="mb-3">
  <label for="passwd">Пароль:</label>
  <input type="password" class="form-control"
    name="passwd" id="passwd" placeholder="Введите пароль">
  <small class="form-text text-muted">От 6 до 16 символов</small>
</div>

```

Стили в Bootstrap 4:

```
.form-text {
  display: block;
  margin-top: 0.25rem;
}
```

Стили в Bootstrap 5 (обратите внимание: дополнительно задан цвет текста, поэтому стилевой класс `text-muted` можно не добавлять):

```
.form-text {
  margin-top: 0.25rem;
  font-size: 0.875em;
  color: #6c757d;
}
```



Рис. 3.6. Вывод пояснительной надписи

Пояснительную надпись, а также кнопку можно добавить слева или справа от поля (рис. 3.7). Для этого в Bootstrap 4 нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если нужно запретить перенос на новую строку, то следует дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать вложенный контейнер со стилевым классом `input-group-prepend` (текст слева от поля) или `input-group-append` (текст справа от поля).
3. Внутри вложенного контейнера создать элемент `SPAN` со стилевым классом `input-group-text` и текстом надписи.

Если используется стилевой класс `input-group-prepend`, то поле добавляется после контейнера с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <div class="input-group-prepend">
    <span class="input-group-text">Пароль:</span>
  </div>
  <input type="password" class="form-control">
</div>
```

Если используется стилевой класс `input-group-append`, то поле добавляется перед контейнером с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <input type="number" step="5" value="100" class="form-control">
  <div class="input-group-append">
    <span class="input-group-text">${</span>
  </div>
</div>
```

Если вы хотите дополнительно использовать тег `<label>`, то его следует вынести за пределы контейнера с классом `input-group`:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="passwd">Пароль:</label>
  <div class="input-group">
    <div class="input-group-prepend">
      <span class="input-group-text">[a-zA-Z0-9]+</span>
    </div>
    <input type="password" name="passwd" id="passwd"
      class="form-control">
  </div>
</div>
```

Если к стилевому классу `input-group` добавить класс `input-group-sm`, то текст надписи и текст внутри поля будут выводиться уменьшенным шрифтом, а если добавить класс `input-group-lg` — то увеличенным шрифтом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group input-group-sm">
  <div class="input-group-prepend">
    <span class="input-group-text">.input-group-sm</span>
  </div>
  <input type="text" class="form-control">
</div>
<div class="mb-3 input-group input-group-lg">
  <div class="input-group-prepend">
    <span class="input-group-text">.input-group-lg</span>
  </div>
  <input type="text" class="form-control">
</div>
```

Можно добавить несколько полей в группу:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group flex-nowrap">
  <div class="input-group-prepend">
    <span class="input-group-text">Надпись</span>
  </div>
  <input type="text" class="form-control">
  <input type="text" class="form-control">
</div>
```

Кроме того, допускается указывать несколько надписей или комбинировать надпись с каким-либо элементом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group flex-nowrap">
  <div class="input-group-prepend">
    <span class="input-group-text">Надпись1</span>
    <span class="input-group-text">Надпись2</span>
  </div>
  <input type="text" class="form-control">
</div>
```

Вместо надписи можно добавить одну или несколько кнопок. Выведем кнопку после текстового поля:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group flex-nowrap">
  <input type="text" class="form-control">
  <div class="input-group-append">
    <button class="btn btn-outline-success" type="button">
      Текст на кнопке</button>
  </div>
</div>
```

Чтобы пояснительную надпись добавить слева или справа от поля в Bootstrap 5 (рис. 3.7), следует выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если нужно запретить перенос на новую строку, то надо дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать тег `` со стилевым классом `input-group-text` и текстом надписи. Чтобы надпись отобразилась слева от поля, ее нужно добавить перед полем, а чтобы справа — после поля. Для программ чтения с экрана указываем параметр `id` с уникальным идентификатором и связываем его с параметром `aria-describedby` текстового поля.

Пример вывода пояснительной надписи перед полем:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <span class="input-group-text" id="lbl1">Пароль:</span>
  <input type="password" class="form-control"
    aria-describedby="lbl1">
</div>
```

Пример вывода пояснительной надписи после поля:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <input type="text" class="form-control"
    aria-describedby="lbl1">
  <span class="input-group-text" id="lbl1">${</span>
</div>
```

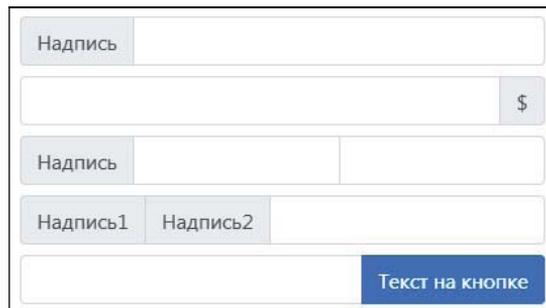


Рис. 3.7. Компонент `input-group` с текстовым полем

Если вы хотите дополнительно использовать тег `<label>`, то его следует вынести за пределы контейнера с классом `input-group`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <label for="passwd" class="form-label">Пароль:</label>
  <div class="input-group">
    <span class="input-group-text" id="lbl1">[a-zA-Z0-9]+</span>
    <input type="password" name="passwd" id="passwd"
      class="form-control" aria-describedby="lbl1">
  </div>
</div>
```

Если к стилевому классу `input-group` добавить класс `input-group-sm`, то текст надписи и текст внутри поля будут выводиться уменьшенным шрифтом, а если добавить класс `input-group-lg` — то увеличенным шрифтом:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group input-group-sm">
  <span class="input-group-text" id="lbl1">.input-group-sm</span>
  <input type="text" class="form-control" aria-describedby="lbl1">
</div>
<div class="mb-3 input-group input-group-lg">
  <span class="input-group-text" id="lbl2">.input-group-lg</span>
  <input type="text" class="form-control" aria-describedby="lbl2">
</div>
```

Можно добавить несколько полей в группу:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group flex-nowrap">
  <span class="input-group-text">Надпись</span>
  <input type="text" class="form-control">
  <input type="text" class="form-control">
</div>
```

Кроме того, допускается указывать несколько надписей или комбинировать надпись с каким-либо элементом:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group flex-nowrap">
  <span class="input-group-text">Надпись1</span>
  <span class="input-group-text">Надпись2</span>
  <input type="text" class="form-control">
</div>
```

Вместо надписи можно добавить одну или несколько кнопок. Выведем кнопку после текстового поля:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group flex-nowrap">
  <input type="text" class="form-control">
  <button class="btn btn-outline-success" type="button">
    Текст на кнопке</button>
</div>
```

3.1.5. Списки автодополнения

Для текстового поля можно подключить список автодополнения. При наборе первых букв под текстовым полем тогда отобразится список с возможными значениями, содержащими набранные буквы. При выборе пункта из списка его значение вставляется в поле.

Создать список автодополнения позволяет парный тег `<datalist>`, в параметре `id` которого нужно обязательно указать уникальный идентификатор. Пункты списка добавляются с помощью тегов `<option>`, расположенных внутри тега `<datalist>`. Чтобы прикрепить список автодополнения к полю, нужно указать идентификатор списка в параметре `list` тега `<input>`:

```
<div class="mb-3">
  <label for="txtCar">Марка автомобиля:</label>
  <input type="text" class="form-control" id="txtCar" list="cars">
  <datalist id="cars">
    <option value="ВАЗ"></option>
    <option value="ГАЗ"></option>
    <option value="Москвич"></option>
    <option value="BMW"></option>
    <option value="Opel"></option>
    <option value="Audi"></option>
  </datalist>
</div>
```

Вместо стилевого класса `form-control` в Bootstrap 4 можно использовать класс `custom-select`, благодаря которому внутри поля справа отобразится значок со стрелками вверх и вниз. Этот значок послужит подсказкой пользователю.

3.1.6. Списки со значениями

Тег `<select>` создает внутри формы список с возможными значениями. Пункт списка, выбранный изначально, должен содержать параметр `selected`:

```
<select>
  <option>Пункт1</option>
  <option>Пункт2</option>
  <option selected>Пункт3</option>
</select>
```

По умолчанию отображается только один пункт списка. С помощью параметра `size` можно задать количество одновременно видимых пунктов:

```
<select size="3">
  <option>Пункт1</option>
  <option>Пункт2</option>
  <option selected>Пункт3</option>
</select>
```

По умолчанию можно выбрать только один пункт из списка. Изменить это поведение позволяет параметр `multiple`, который указывает, что из списка можно выбрать

сразу несколько элементов одновременно, дополнительно удерживая при выборе нажатой клавишу <Ctrl> (выбор одного пункта) или <Shift> (выбор диапазона):

```
<select size="3" multiple>
  <option>Пункт1</option>
  <option>Пункт2</option>
  <option selected>Пункт3</option>
</select>
```

Чтобы сделать пункт списка недоступным, следует добавить параметр `disabled`:

```
<select size="3">
  <option>Пункт1</option>
  <option disabled>Пункт2</option>
  <option selected>Пункт3</option>
</select>
```

С помощью тега `<optgroup>` можно объединить несколько пунктов в группу (рис. 3.8). Название группы указывается в параметре `label`:

```
<select>
  <optgroup label="Отечественные">
    <option>ВАЗ</option>
    <option>ГАЗ</option>
    <option>Москвич</option>
  </optgroup>
  <optgroup label="Зарубежные">
    <option>BMW</option>
    <option>Opel</option>
    <option selected>Audi</option>
  </optgroup>
</select>
```

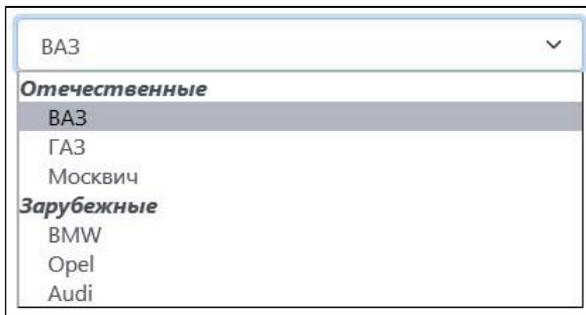


Рис. 3.8. Объединение нескольких пунктов в группу

Если в теге `<optgroup>` указать параметр `disabled`, то вся группа станет недоступной:

```
<optgroup label="Отечественные" disabled>
```

Чтобы к списку применить стилизацию из библиотеки Bootstrap 4, следует добавить стиливой класс `form-control` (рис. 3.9). В результате список займет всю ширину родительского контейнера. При получении фокуса ввода границы списка будут

подсвечиваться. Пример добавления списка и вывода пояснительных надписей до и после списка:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="list1">Надпись:</label>
  <select size="3" name="list1" id="list1" class="form-control">
    <option value="1">Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3" selected>Пункт3</option>
  </select>
  <small class="form-text text-muted">Пояснение</small>
</div>
```

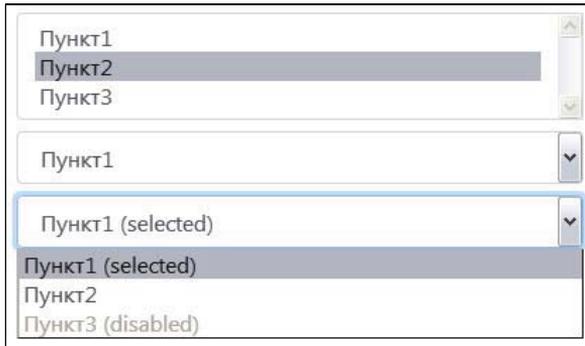


Рис. 3.9. Списки со стилевым классом `form-control` в Bootstrap 4



Рис. 3.10. Списки со стилевым классом `form-select` в Bootstrap 5

В Bootstrap 5 вместо класса `form-control` следует добавить класс `form-select` (рис. 3.10):

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <label for="list1" class="form-label">Надпись:</label>
  <select size="3" name="list1" id="list1" class="form-select">
    <option value="1">Пункт1</option>
```

```

    <option value="2">Пункт2</option>
    <option value="3" selected>Пункт3</option>
  </select>
  <small class="form-text text-muted">Пояснение</small>
</div>

```

Если к стилевому классу `form-control` в Bootstrap 4 добавить класс `form-control-sm`, то текст пунктов внутри списка будет выводиться уменьшенным шрифтом, а если добавить класс `form-control-lg` — то увеличенным шрифтом:

```

<!-- Только в Bootstrap 4 -->
<div class="container">
  <div class="mb-3">
    <select size="3" class="form-control form-control-sm">
      <option value="1">Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3" selected>Пункт3</option>
    </select>
  </div>
  <div class="mb-3">
    <select size="3" class="form-control form-control-lg">
      <option value="1">Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3">Пункт3</option>
    </select>
  </div>
</div>

```

В Bootstrap 5 к стилевому классу `form-select` следует добавлять классы `form-select-sm` (текст пунктов внутри списка будет выводиться уменьшенным шрифтом) или `form-select-lg` (текст пунктов внутри списка будет выводиться увеличенным шрифтом):

```

<!-- Только в Bootstrap 5 -->
<div class="container">
  <div class="mb-3">
    <select size="3" class="form-select form-select-sm">
      <option value="1">Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3" selected>Пункт3</option>
    </select>
  </div>
  <div class="mb-3">
    <select size="3" class="form-select form-select-lg">
      <option value="1">Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3">Пункт3</option>
    </select>
  </div>
</div>

```

Если в Bootstrap 4 вместо стилевого класса `form-control` указать класс `custom-select`, то список будет похож на текстовое поле. Внутри поля справа отображается значок со стрелками вверх и вниз (рис. 3.11):

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="list1">Надпись</label>
  <select name="list1" id="list1" class="custom-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
  <small class="form-text text-muted">Пояснение</small>
</div>
```

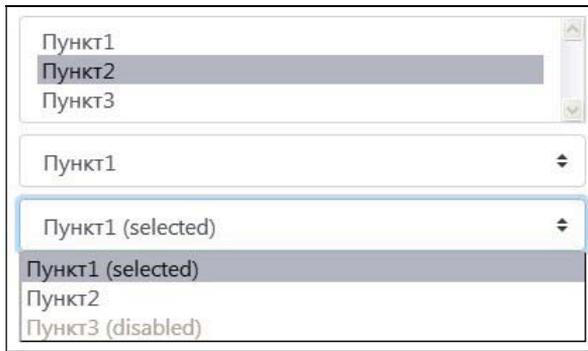


Рис. 3.11. Списки со стилевым классом `custom-select` в Bootstrap 4

Если в Bootstrap 4 к стилевому классу `custom-select` добавить класс `custom-select-sm`, то текст пунктов внутри списка будет выводиться уменьшенным шрифтом, а если добавить класс `custom-select-lg` — то увеличенным шрифтом:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <div class="mb-3">
    <select class="custom-select custom-select-sm">
      <option value="1" selected>Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3">Пункт3</option>
    </select>
  </div>
  <div class="mb-3">
    <select class="custom-select custom-select-lg">
      <option value="1" selected>Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3">Пункт3</option>
    </select>
  </div>
</div>
```

Пояснительную надпись, а также кнопку можно добавить слева или справа от списка со значениями (рис. 3.12). Для этого в Bootstrap 4 нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если надо запретить перенос на новую строку, то следует дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать вложенный контейнер со стилевым классом `input-group-prepend` (текст слева от списка) или `input-group-append` (текст справа от списка).
3. Внутри вложенного контейнера создать элемент `SPAN` со стилевым классом `input-group-text` и текстом надписи.

Если используется стилевой класс `input-group-prepend`, то список добавляется после контейнера с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <div class="input-group-prepend">
    <span class="input-group-text">Надпись</span>
  </div>
  <select class="custom-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
</div>
```

Если используется стилевой класс `input-group-append`, то список добавляется перед контейнером с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <select class="custom-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
  <div class="input-group-append">
    <span class="input-group-text">Надпись</span>
  </div>
</div>
```

Если вы хотите дополнительно использовать тег `<label>`, то его следует вынести за пределы контейнера с классом `input-group`:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="list1">Надпись</label>
  <div class="input-group">
    <div class="input-group-prepend">
      <span class="input-group-text">Надпись</span>
    </div>
```

```

    <select name="list1" id="list1" class="custom-select">
      <option value="1" selected>Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3">Пункт3</option>
    </select>
  </div>
</div>

```

Если к стилевому классу `input-group` добавить класс `input-group-sm`, то текст надписи и текст внутри списка будут выводиться уменьшенным шрифтом, а если добавить класс `input-group-lg` — то увеличенным шрифтом:

```

<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group input-group-sm">
  <div class="input-group-prepend">
    <span class="input-group-text">Надпись</span>
  </div>
  <select class="custom-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
</div>
<div class="mb-3 input-group input-group-lg">
  <div class="input-group-prepend">
    <span class="input-group-text">Надпись</span>
  </div>
  <select class="custom-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
</div>

```

Вместо надписи можно добавить одну или несколько кнопок. Выведем кнопку после списка:

```

<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <select class="custom-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
  <div class="input-group-append">
    <button class="btn btn-outline-success" type="button">
      Текст на кнопке</button>
  </div>
</div>

```

Чтобы пояснительную надпись добавить слева или справа от списка со значениями в Bootstrap 5 (рис. 3.12), нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если надо запретить перенос на новую строку, то следует дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать тег `<label>` или `` со стилевым классом `input-group-text` и текстом надписи. Чтобы надпись отобразилась слева от списка, ее надо добавить перед списком, а чтобы справа — после списка.
3. Для списка указать стилевой класс `form-select`.

Пример вывода пояснительной надписи перед списком:

```
<!-- Только в Bootstrap 5 -->
<div class="input-group mb-3">
  <label class="input-group-text" for="list1">Надпись</label>
  <select id="list1" class="form-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
</div>
```

Пример вывода пояснительной надписи после списка:

```
<!-- Только в Bootstrap 5 -->
<div class="input-group mb-3">
  <select id="list1" class="form-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
  <label class="input-group-text" for="list1">Надпись</label>
</div>
```

Если вы хотите дополнительно задействовать тег `<label>`, то его следует вынести за пределы контейнера с классом `input-group`, а для надписи использовать тег ``:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <label for="list1" class="form-label">Надпись</label>
  <div class="input-group">
    <span class="input-group-text">Надпись</span>
    <select name="list1" id="list1" class="form-select">
      <option value="1" selected>Пункт1</option>
      <option value="2">Пункт2</option>
      <option value="3">Пункт3</option>
    </select>
  </div>
</div>
```

Если к стилевому классу `input-group` добавить класс `input-group-sm`, то текст надписи и текст внутри списка будут выводиться уменьшенным шрифтом, а если добавить класс `input-group-lg` — то увеличенным шрифтом:

```
<!-- Только в Bootstrap 5 -->
<div class="input-group input-group-sm mb-3">
  <label class="input-group-text" for="list1">Надпись</label>
  <select id="list1" class="form-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
</div>
<div class="input-group input-group-lg mb-3">
  <label class="input-group-text" for="list2">Надпись</label>
  <select id="list2" class="form-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
</div>
```

Вместо надписи можно добавить одну или несколько кнопок. Выведем кнопку после списка:

```
<!-- Только в Bootstrap 5 -->
<div class="input-group mb-3">
  <select id="list1" class="form-select">
    <option value="1" selected>Пункт1</option>
    <option value="2">Пункт2</option>
    <option value="3">Пункт3</option>
  </select>
  <button class="btn btn-outline-success" type="button">
    Текст на кнопке</button>
</div>
```



Рис. 3.12. Компонент `input-group` со списком

3.1.7. Флажки

Вставить флажок позволяет тег `<input>`, в параметре `type` которого указывается значение `checkbox`:

```
<input type="checkbox" name="check1" id="check1">
```

Для установки флажка по умолчанию следует добавить параметр `checked`:

```
<input type="checkbox" name="check1" id="check1" checked>
```

По умолчанию доступны только два состояния флажка: установлен и сброшен. С помощью JavaScript можно добавить неопределенное состояние, присвоив свойству `indeterminate` значение `true`:

```
var elem = document.getElementById('check1');  
if (elem) elem.indeterminate = true;
```

В этом примере мы находим флажок по идентификатору (`id="check1"`) и присваиваем свойству `indeterminate` значение `true`. Напомню, что код на языке JavaScript нужно вставлять внутри тега `<script>` после комментария в шаблоне из листинга 1.1:

```
// Сюда вставляем код на JavaScript  
var elem = document.getElementById('check1');  
if (elem) elem.indeterminate = true;
```

Аналогичный код на jQuery:

```
$('#check1').prop('indeterminate', true);
```

Чтобы к флажку применить стилизацию из библиотеки Bootstrap (рис. 3.13 и 3.14), следует выполнить следующие шаги:

1. Для родительского контейнера добавить стилевой класс `form-check`.
2. Для флажка добавить стилевой класс `form-check-input`.
3. Для тега `<label>` с пояснительным текстом добавить стилевой класс `form-check-label`.

Пример:

```
<div class="mb-3 form-check">  
  <input type="checkbox" class="form-check-input"  
    name="check1" id="check1">  
  <label class="form-check-label" for="check1">Надпись</label>  
</div>
```

Если флажок недоступен (указан параметр `disabled`), то флажок и пояснительная надпись отображаются серым цветом:

```
<div class="mb-3 form-check">  
  <input type="checkbox" class="form-check-input"  
    name="check1" id="check1" disabled>  
  <label class="form-check-label" for="check1">Надпись</label>  
</div>
```

Чтобы отобразить флажок без пояснительной надписи, в Bootstrap 4 следует добавить стилевой класс `position-static`:

```
<!-- Только в Bootstrap 4 -->  
<div class="mb-3 form-check">  
  <input class="form-check-input position-static" type="checkbox">  
</div>
```

В Bootstrap 5 надо опустить указание класса `form-check`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <input class="form-check-input" type="checkbox">
</div>
```

Для отображения сразу нескольких флажков на одной строке помимо стилевого класса `form-check` нужно добавить класс `form-check-inline`:

```
<div class="mb-3">
  <div class="form-check form-check-inline">
    <input type="checkbox" class="form-check-input"
      name="check1" id="check1">
    <label class="form-check-label" for="check1">Надпись1</label>
  </div>
  <div class="form-check form-check-inline">
    <input type="checkbox" class="form-check-input"
      name="check2" id="check2">
    <label class="form-check-label" for="check2">Надпись2</label>
  </div>
</div>
```

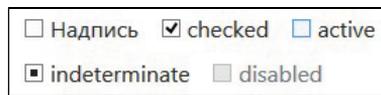


Рис. 3.13. Флажки со стилевым классом `form-check-input` в Bootstrap 4

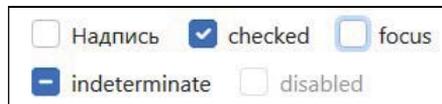


Рис. 3.14. Флажки со стилевым классом `form-check-input` в Bootstrap 5

Применить стилизацию в Bootstrap 4 (рис. 3.15) можно также с помощью следующих действий:

1. Для родительского контейнера добавить стилевые классы `custom-control` и `custom-checkbox`.
2. Для флажка добавить стилевой класс `custom-control-input`.
3. Для тега `<label>` с пояснительным текстом добавить стилевой класс `custom-control-label`.

Пример:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-checkbox">
    <input type="checkbox" class="custom-control-input"
      name="check1" id="check1">
    <label class="custom-control-label" for="check1">Надпись</label>
  </div>
</div>
```

Для отображения сразу нескольких флажков на одной строке помимо стилевых классов `custom-control` и `custom-checkbox` нужно добавить класс `custom-control-inline`:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-checkbox custom-control-inline">
    <input type="checkbox" class="custom-control-input"
      name="check1" id="check1">
    <label class="custom-control-label" for="check1">Надпись1</label>
  </div>
  <div class="custom-control custom-checkbox custom-control-inline">
    <input type="checkbox" class="custom-control-input"
      name="check2" id="check2">
    <label class="custom-control-label" for="check2">Надпись2</label>
  </div>
</div>
```

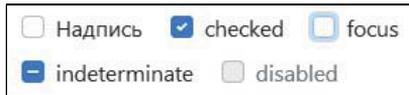


Рис. 3.15. Флажки со стилевым классом `custom-control-input` в Bootstrap 4

Флажок можно добавить слева или справа от текстового поля (рис. 3.16). Для этого в Bootstrap 4 нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если надо запретить перенос на новую строку, то следует дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать вложенный контейнер со стилевым классом `input-group-prepend` (флажок слева от поля) или `input-group-append` (флажок справа от поля).
3. Внутри вложенного контейнера создать контейнер со стилевым классом `input-group-text` и флажком.

Если используется стилевой класс `input-group-prepend`, то поле добавляется после контейнера с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <div class="input-group-prepend">
    <div class="input-group-text">
      <input type="checkbox">
    </div>
  </div>
  <input type="text" class="form-control">
</div>
```

Если используется стилевой класс `input-group-append`, то поле добавляется перед контейнером с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <input type="text" class="form-control">
  <div class="input-group-append">
    <div class="input-group-text">
      <input type="checkbox">
    </div>
  </div>
</div>
```

Чтобы флажок добавить слева или справа от текстового поля в Bootstrap 5 (рис. 3.16), нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если надо запретить перенос на новую строку, то следует дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать тег `<div>` со стилевым классом `input-group-text` и вложить в него флажок со стилевым классом `form-check-input`.



Рис. 3.16. Компонент `input-group` с флажком

Чтобы флажок отобразился перед текстовым полем, следует поле добавить после контейнера с классом `input-group-text`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <div class="input-group-text">
    <input type="checkbox" class="form-check-input">
  </div>
  <input type="text" class="form-control">
</div>
```

Чтобы флажок отобразился после текстового поля, следует поле добавить перед контейнером с классом `input-group-text`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <input type="text" class="form-control">
  <div class="input-group-text">
    <input type="checkbox" class="form-check-input">
  </div>
</div>
```

3.1.8. Выключатели

С помощью следующих действий в Bootstrap 4 можно превратить стандартный флажок в нестандартный компонент — выключатель (рис. 3.17):

1. Для родительского контейнера добавить стилевые классы `custom-control` и `custom-switch`.
2. Для флажка добавить стилевой класс `custom-control-input`.
3. Для тега `<label>` с пояснительным текстом добавить стилевой класс `custom-control-label`.

Выключатель имеет два основных состояния: включен и выключен. Для переключения состояний следует щелкнуть на компоненте мышью. При включенном состоянии фон компонента будет синего цвета, а при выключенном состоянии — белого цвета. Пример вставки выключателя в Bootstrap 4:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-switch">
    <input type="checkbox" class="custom-control-input"
      name="check1" id="check1">
    <label class="custom-control-label" for="check1">Надпись</label>
  </div>
</div>
```

По умолчанию компонент находится в выключенном состоянии. Чтобы включить его из программы, нужно добавить параметр `checked`:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-switch">
    <input type="checkbox" class="custom-control-input"
      name="check1" id="check1" checked>
    <label class="custom-control-label" for="check1">Надпись</label>
  </div>
</div>
```

Выключатель можно сделать неактивным. Для этого следует добавить параметр `disabled`. В неактивном состоянии выключатель и текст надписи отображаются серым цветом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-switch">
    <input type="checkbox" class="custom-control-input"
      name="check1" id="check1" disabled>
    <label class="custom-control-label" for="check1">Надпись</label>
  </div>
</div>
```



Рис. 3.17. Выключатели в Bootstrap 4

В Bootstrap 5 для создания выключателя (рис. 3.18) нужно выполнить следующие действия:

1. Для родительского контейнера добавить стилевые классы `form-check` и `form-switch`.
2. Для флажка добавить стилевой класс `form-check-input`.
3. Для тега `<label>` с пояснительным текстом добавить стилевой класс `form-check-label`.

Выключатель имеет два основных состояния: включен и выключен. Для переключения состояний следует щелкнуть мышью на компоненте или на пояснительной надписи. При включенном состоянии фон компонента будет синего цвета, а при выключенном состоянии — белого цвета. Пример вставки выключателя в Bootstrap 5:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <div class="form-check form-switch">
    <input type="checkbox" class="form-check-input"
      name="check1" id="check1">
    <label class="form-check-label" for="check1">Надпись</label>
  </div>
</div>
```

По умолчанию компонент находится в выключенном состоянии. Чтобы включить его из программы, нужно добавить параметр `checked`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <div class="form-check form-switch">
    <input type="checkbox" class="form-check-input"
      name="check1" id="check1" checked>
    <label class="form-check-label" for="check1">Надпись</label>
  </div>
</div>
```

Выключатель можно сделать неактивным. Для этого следует добавить параметр `disabled`. В неактивном состоянии выключатель и текст надписи отображаются серым цветом:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <div class="form-check form-switch">
    <input type="checkbox" class="form-check-input"
      name="check1" id="check1" disabled>
    <label class="form-check-label" for="check1">Надпись</label>
  </div>
</div>
```



Рис. 3.18. Выключатели в Bootstrap 5

3.1.9. Переключатели

Вставить переключатель позволяет тег `<input>`, в параметре `type` которого указывается значение `radio`:

```
<input type="radio">
```

Для выбора переключателя по умолчанию следует добавить параметр `checked`:

```
<input type="radio" checked>
```

Переключатель может существовать только в составе группы подобных элементов, из которых может быть выбран лишь один. Для объединения переключателей в группу необходимо установить одинаковое значение параметра `name` и разное значение параметра `value`:

```
<input type="radio" name="sex" value="male"> Мужской  
<input type="radio" name="sex" value="female"> Женский
```

Чтобы к переключателю применить стилизацию из библиотеки Bootstrap (рис. 3.19), нужно выполнить следующие шаги:

1. Для родительского контейнера добавить стилевой класс `form-check`.
2. Для переключателя добавить стилевой класс `form-check-input`.
3. Для тега `<label>` с пояснительным текстом добавить стилевой класс `form-check-label`.

Пример:

```
<div class="mb-3">  
  <div class="form-check">  
    <input type="radio" name="sex" id="radio1" value="male"  
      class="form-check-input">  
    <label class="form-check-label" for="radio1">Мужской</label>  
  </div>  
  <div class="form-check">  
    <input type="radio" name="sex" id="radio2" value="female"  
      class="form-check-input">  
    <label class="form-check-label" for="radio2">Женский</label>  
  </div>  
</div>
```

Если переключатель недоступен (указан параметр `disabled`), то переключатель и пояснительная надпись отображаются серым цветом:

```
<div class="form-check">  
  <input type="radio" name="sex" id="radio1" value="male"  
    class="form-check-input" disabled>  
  <label class="form-check-label" for="radio1">Мужской</label>  
</div>
```

Чтобы отобразить переключатель без пояснительной надписи в Bootstrap 4, следует добавить стилевой класс `position-static`:

```
<!-- Только в Bootstrap 4 -->  
<div class="form-check">  
  <input type="radio" class="form-check-input position-static">  
</div>
```

В Bootstrap 5 следует опустить указание класса `form-check`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <input type="radio" class="form-check-input">
</div>
```

Для отображения сразу нескольких переключателей на одной строке помимо стилового класса `form-check` нужно добавить класс `form-check-inline`:

```
<div class="mb-3">
  <div class="form-check form-check-inline">
    <input type="radio" name="sex" id="radio1" value="male"
      class="form-check-input">
    <label class="form-check-label" for="radio1">Мужской</label>
  </div>
  <div class="form-check form-check-inline">
    <input type="radio" name="sex" id="radio2" value="female"
      class="form-check-input">
    <label class="form-check-label" for="radio2">Женский</label>
  </div>
</div>
```



Рис. 3.19. Переключатели со стиливым классом `form-check-input` в Bootstrap 5

Применить стилизацию в Bootstrap 4 (рис. 3.20) можно также с помощью следующих действий:

1. Для родительского контейнера добавить стиливые классы `custom-control` и `custom-radio`.
2. Для переключателя добавить стиливой класс `custom-control-input`.
3. Для тега `<label>` с пояснительным текстом добавить стиливой класс `custom-control-label`.

Пример:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-radio">
    <input type="radio" name="sex" id="radio1" value="male"
      class="custom-control-input">
    <label class="custom-control-label"
      for="radio1">Мужской</label>
  </div>
  <div class="custom-control custom-radio">
    <input type="radio" name="sex" id="radio2" value="female"
      class="custom-control-input">
    <label class="custom-control-label"
      for="radio2">Женский</label>
  </div>
</div>
```

Для отображения сразу нескольких переключателей на одной строке помимо стилевых классов `custom-control` и `custom-radio` нужно добавить класс `custom-control-inline`:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-control custom-radio custom-control-inline">
    <input type="radio" name="sex" id="radio1" value="male"
      class="custom-control-input">
    <label class="custom-control-label"
      for="radio1">Мужской</label>
  </div>
  <div class="custom-control custom-radio custom-control-inline">
    <input type="radio" name="sex" id="radio2" value="female"
      class="custom-control-input">
    <label class="custom-control-label"
      for="radio2">Женский</label>
  </div>
</div>
```



Рис. 3.20. Переключатели со стилевым классом `custom-control-input` в Bootstrap 4

3.1.10. Поле выбора файла

Поле выбора файла реализуется с помощью тега `<input>`, в параметре `type` которого указано значение `file`:

```
<input type="file">
```

Поддерживаются следующие основные параметры:

- ◆ `multiple` — если параметр указан, то можно выбрать сразу несколько файлов:


```
<input type="file" multiple>
```
- ◆ `accept` — задает поддерживаемые MIME-типы или расширения файлов (значения указываются через запятую):


```
<input type="file" accept="image/jpeg, image/png, image/gif">
<input type="file" accept="image/*">
<input type="file" accept=".gif, .jpg, .jpeg">
```

ПРИМЕЧАНИЕ

При пересылке файлов параметр `method` в теге `<form>` должен иметь значение `POST`, а параметр `enctype` — значение `multipart/form-data`.

Если в Bootstrap 4 к полю добавить стилевой класс `form-control-file`, то поле займет всю ширину родительского элемента:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="file1">Выберите файл</label>
  <input type="file" class="form-control-file" id="file1">
</div>
```

Чтобы к полю выбора файла применить стилизацию из библиотеки Bootstrap 4 (рис. 3.21), нужно выполнить следующие шаги:

1. Для родительского контейнера добавить стилевой класс `custom-file`.
2. Для поля выбора файла добавить стилевой класс `custom-file-input`.
3. Для тега `<label>` с пояснительным текстом добавить стилевой класс `custom-file-label`. По умолчанию справа от поля отображается текст «Browse», но если добавить параметр `data-browse`, то будет отображаться текст, указанный в качестве значения этого параметра, — например, «Обзор...».

Пример:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-file">
    <input type="file" class="custom-file-input" id="file1">
    <label class="custom-file-label" for="file1"
      data-browse="Обзор...">Файл не выбран</label>
  </div>
</div>
```



Рис. 3.21. Поле выбора файла в Bootstrap 4

Вместо использования параметра `data-browse` можно указать текст надписи в зависимости от языка. Это может быть язык всего документа или явно указанный язык с помощью параметра `lang` для поля выбора файла. Пример указания правила для русского языка:

```
.custom-file-input:lang(ru) ~ .custom-file-label::after {
  content: "Обзор...";
}
```

Пример явного указания параметра `lang` для русского языка:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <div class="custom-file">
    <input type="file" class="custom-file-input" id="file1" lang="ru">
    <label class="custom-file-label" for="file1">Файл не выбран</label>
  </div>
</div>
```

Чтобы после выбора файла его название отобразилось внутри поля, нужно подключить дополнительную библиотеку `bs-custom-file-input`. Для этого после инструкции:

```
<script src="js/bootstrap.bundle.min.js"></script>
```

вставляем инструкцию (без символа переноса строки):

```
<script src="https://cdn.jsdelivr.net/npm/bs-custom-file-input/dist/
bs-custom-file-input.min.js"></script>
```

Далее после комментария:

```
// Сюда вставляем код на JavaScript
```

вставляем инструкцию:

```
bsCustomFileInput.init();
```

Библиотека `bs-custom-file-input` по умолчанию работает со стилевыми классами из библиотеки `Bootstrap 4` и не имеет зависимости от `jQuery`, поэтому ничего дополнительно подключать и настраивать не нужно. Библиотека позволяет отобразить название одного выбранного файла, а также поддерживает возможность множественного выбора файлов. Кроме того, можно выбрать файл путем перетаскивания ярлыка файла на поле.

ПРИМЕЧАНИЕ

Дополнительную информацию о библиотеке `bs-custom-file-input` можно найти на странице <https://www.npmjs.com/package/bs-custom-file-input>.

Чтобы к полю выбора файла применить стилизацию из библиотеки `Bootstrap 5` (рис. 3.22), нужно выполнить следующие шаги:

1. Для родительского контейнера добавить стилевой класс `form-file`.
2. Для поля выбора файла добавить стилевой класс `form-file-input`.
3. Для тега `<label>` добавить стилевой класс `form-file-label`
4. Внутри тега `<label>` вложить тег `` со стилевым классом `form-file-text` и текстом надписи внутри поля — например, «Файл не выбран». Если текст слишком длинный, то он будет обрезан, а в конец добавится многоточие.
5. Внутри тега `<label>` вложить тег `` со стилевым классом `form-file-button` и текстом надписи на кнопке — например, «Обзор...».

Пример:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <div class="form-file">
    <input type="file" class="form-file-input" id="file1">
    <label for="file1" class="form-file-label">
      <span class="form-file-text">Файл не выбран</span>
      <span class="form-file-button">Обзор...</span>
    </label>
  </div>
</div>
```

На момент подготовки этой книги библиотека `bs-custom-file-input` работала только с разметкой из `Bootstrap 4`. Чтобы отобразить название первого выбранного файла внутри поля в `Bootstrap 5`, давайте самостоятельно напишем программу на `jQuery`:

```
$('.form-file-input').change( function() {
  if (this.files.length == 0) {
    $(this).parent().find('.form-file-text').text('Файл не выбран');
  }
}
```

```

else {
    $(this).parent().find('.form-file-text').text(this.files[0].name);
}
});

```

В этом примере для всех элементов со стилевым классом `form-file-input` мы назначаем обработчик для события `onchange`, которое возникает при изменении значения. Внутри обработчика указатель `this` ссылается на поле для выбора файла, являющееся источником события. Все выбранные файлы доступны через свойство `files`. Если файл не выбран, то свойство `length` будет равно нулю. В этом случае выводим в поле текст «Файл не выбран». Если свойство `length` не равно нулю, то выводим в поле название первого выбранного файла, получая доступ к объекту файла с индексом 0. Название файла доступно через свойство `name`. Для поиска элемента с классом `form-file-text`, в котором выводится название выбранного файла, мы сначала находим текущий элемент, передавая указатель `this` функции `$()`. Далее с помощью метода `parent()` находим родительской элемент. Затем с помощью метода `find()` находим вложенный элемент со стилевым классом `form-file-text`. С помощью метода `text()` выводим текст сообщения.

Если к стилевому классу `form-file` добавить класс `form-file-sm`, то размер поля и кнопки будет меньше стандартного размера, а если добавить класс `form-file-lg` — то больше:

```

<!-- Только в Bootstrap 5 -->
<div class="form-file form-file-sm mb-3">
  <input type="file" class="form-file-input" id="file1">
  <label for="file1" class="form-file-label">
    <span class="form-file-text">.<span class="form-file-button">Обзор...</span>
  </label>
</div>
<div class="form-file form-file-lg mb-3">
  <input type="file" class="form-file-input" id="file2">
  <label for="file2" class="form-file-label">
    <span class="form-file-text">.<span class="form-file-button">Обзор...</span>
  </label>
</div>

```



Рис. 3.22. Поля выбора файла в Bootstrap 5

Пояснительную надпись, а также кнопку можно добавить слева или справа от поля выбора файла (рис. 3.23). Для этого в Bootstrap 4 нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`. Если нужно запретить перенос на новую строку, то следует дополнительно указать стилевой класс `flex-nowrap`.
2. Внутри этого контейнера создать вложенный контейнер со стилевым классом `input-group-prepend` (текст слева от поля) или `input-group-append` (текст справа от поля).
3. Внутри вложенного контейнера создать элемент `SPAN` со стилевым классом `input-group-text` и текстом надписи.

Если используется стилевой класс `input-group-prepend`, то поле добавляется после контейнера с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <div class="input-group-prepend">
    <span class="input-group-text">Надпись</span>
  </div>
  <div class="custom-file">
    <input type="file" class="custom-file-input" id="file1">
    <label class="custom-file-label" for="file1"
      data-browse="Обзор...">Файл не выбран</label>
  </div>
</div>
```

Если используется стилевой класс `input-group-append`, то поле добавляется перед контейнером с этим классом:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <div class="custom-file">
    <input type="file" class="custom-file-input" id="file1">
    <label class="custom-file-label" for="file1"
      data-browse="Обзор...">Файл не выбран</label>
  </div>
  <div class="input-group-append">
    <span class="input-group-text">Надпись</span>
  </div>
</div>
```

Вместо надписи можно добавить одну или несколько кнопок. Выведем кнопку после поля:

```
<!-- Только в Bootstrap 4 -->
<div class="mb-3 input-group">
  <div class="custom-file">
    <input type="file" class="custom-file-input" id="file1">
    <label class="custom-file-label" for="file1"
      data-browse="Обзор...">Файл не выбран</label>
```

```

</div>
<div class="input-group-append">
  <button class="btn btn-outline-success" type="button">
    Текст на кнопке</button>
</div>
</div>

```

Чтобы пояснительную надпись добавить слева или справа от поля выбора файла в Bootstrap 5 (рис. 3.23), нужно выполнить следующие шаги:

1. Создать контейнер и добавить для него стилевой класс `input-group`.
2. Внутри этого контейнера создать тег `` со стилевым классом `input-group-text` и текстом надписи. Чтобы надпись отобразилась слева от поля, ее надо добавить перед полем, а чтобы справа — после поля.

Пример вывода пояснительной надписи перед полем:

```

<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <span class="input-group-text">Надпись</span>
  <div class="form-file">
    <input type="file" class="form-file-input" id="file1">
    <label for="file1" class="form-file-label">
      <span class="form-file-text">Файл не выбран</span>
      <span class="form-file-button">Обзор...</span>
    </label>
  </div>
</div>

```

Пример вывода пояснительной надписи после поля:

```

<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <div class="form-file">
    <input type="file" class="form-file-input" id="file1">
    <label for="file1" class="form-file-label">
      <span class="form-file-text">Файл не выбран</span>
      <span class="form-file-button">Обзор...</span>
    </label>
  </div>
  <span class="input-group-text">Надпись</span>
</div>

```

Вместо надписи можно добавить одну или несколько кнопок. Выведем кнопку после поля:

```

<!-- Только в Bootstrap 5 -->
<div class="mb-3 input-group">
  <div class="form-file">
    <input type="file" class="form-file-input" id="file1">
    <label for="file1" class="form-file-label">
      <span class="form-file-text">Файл не выбран</span>
      <span class="form-file-button">Обзор...</span>
    </label>

```

```

</div>
<button class="btn btn-outline-success" type="button">
  Текст на кнопке</button>
</div>

```

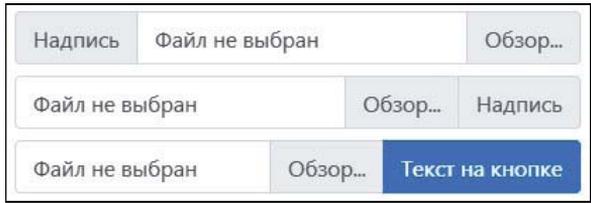


Рис. 3.23. Компонент input-group с полем выбора файла

3.1.11. Шкала с ползунком

Шкала с ползунком вставляется с помощью тега `<input>`, в параметре `type` которого указывается значение `range`. Минимальное значение диапазона задается параметром `min`, максимальное — параметром `max`, текущее — параметром `value`, а шаг — параметром `step`:

```
<input type="range" min="0" max="100" value="10" step="5">
```

По умолчанию значение изменяется на целое число, т. к. параметр `step` имеет значение 1. Чтобы значение могло быть вещественным числом, следует соответствующим образом изменить значение параметра `step`:

```
<input type="range" min="0" max="10" value="2.5" step="0.1">
```

Если в Bootstrap 4 добавить стилевой класс `form-control-range`, то шкала займет всю ширину родительского элемента:

```

<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="range1">Выберите значение</label>
  <input type="range" min="0" max="100" value="10"
    class="form-control-range" id="range1">
</div>

```

Чтобы к ползунку применить стилизацию из библиотеки Bootstrap 4 (рис. 3.24), достаточно добавить стилевой класс `custom-range`:

```

<!-- Только в Bootstrap 4 -->
<div class="mb-3">
  <label for="range1">Выберите значение</label>
  <input type="range" min="0" max="100" value="10"
    class="custom-range" id="range1">
</div>

```



Рис. 3.24. Шкала с ползунком в Bootstrap

В Bootstrap 5 следует добавить стилевой класс `form-range`:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <label for="range1" class="form-label">Выберите значение</label>
  <input type="range" min="0" max="100" value="10"
    class="form-range" id="range1">
</div>
```

3.1.12. Элемент для выбора цвета

Чтобы создать элемент для выбора цвета, нужно в параметре `type` тега `<input>` указать значение `color`. При щелчке на элементе отображается диалоговое окно (рис. 3.25), в котором можно выбрать цвет или ввести его значение вручную. Элемент не поддерживается Internet Explorer. Текущее значение задается с помощью параметра `value` в формате `#RRGGBB`:

```
<input type="color" value="#FF0000">
```

Чтобы к элементу применить стилизацию из библиотеки Bootstrap 5, достаточно добавить стилевые классы `form-control` и `form-control-color`:

```
<!-- Только в Bootstrap 5 -->
<input type="color" value="#FF0000"
  class="form-control form-control-color">
```

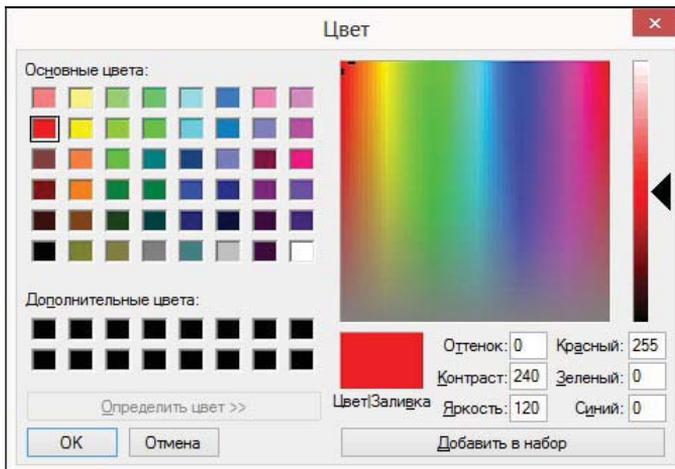


Рис. 3.25. Диалоговое окно для выбора цвета

3.2. Выравнивание и группировка элементов формы

В предыдущем разделе мы рассмотрели отдельные элементы управления. Сейчас же научимся выравнивать элементы по вертикали, горизонтали и по сетке, а также объединять элементы в группы.

3.2.1. Выравнивание элементов по вертикали

В библиотеке Bootstrap все элементы управления объявлены блочными, и они занимают всю ширину родительского контейнера. Поэтому для выравнивания элементов по вертикали достаточно вложить каждое поле с пояснительной надписью в контейнер и добавить к нему в Bootstrap 4 стилевой класс `form-group`, который задает внешний отступ снизу. В Bootstrap 5 класс `form-group` был удален, и вместо него нужно использовать класс `mb-3`.

Пример выравнивания элементов формы по вертикали в Bootstrap 4 приведен в листинге 3.1. Результат выполнения показан на рис. 3.26.

Листинг 3.1. Выравнивание элементов формы по вертикали в Bootstrap 4

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <form action="#" method="GET">
    <div class="form-group">
      <label for="login">Логин:</label>
      <input type="text" class="form-control"
        name="login" id="login" placeholder="Введите логин">
    </div>
    <div class="form-group">
      <label for="passwd">Пароль:</label>
      <input type="password" class="form-control"
        name="passwd" id="passwd" placeholder="Введите пароль">
      <small class="form-text text-muted">От 6 до 16 символов</small>
    </div>
    <div class="form-group">
      <div><a href="#">Правила участия</a></div>
      <div class="form-check form-check-inline">
        <input type="radio" name="rule" id="radio1" value="yes"
          class="form-check-input">
        <label class="form-check-label" for="radio1">Согласен</label>
      </div>
      <div class="form-check form-check-inline">
        <input type="radio" name="rule" id="radio2" value="no"
          class="form-check-input" checked>
        <label class="form-check-label" for="radio2">Нет</label>
      </div>
    </div>
    <button type="submit" class="btn btn-primary">Отправить</button>
  </form>
</div>
```

В Bootstrap 5 вместо класса `form-group` нужно указать класс `mb-3`. Кроме того, для пояснительной надписи следует добавить стилевой класс `form-label`:

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <form action="#" method="GET">
```

```

<div class="mb-3">
  <label for="login" class="form-label">Логин:</label>
  <input type="text" class="form-control"
    name="login" id="login" placeholder="Введите логин">
</div>
<div class="mb-3">
  <label for="passwd" class="form-label">Пароль:</label>
  <input type="password" class="form-control"
    name="passwd" id="passwd" placeholder="Введите пароль">
  <small class="form-text text-muted">От 6 до 16 символов</small>
</div>
<div class="mb-3">
  <div class="mb-2"><a href="#">Правила участия</a></div>
  <div class="form-check form-check-inline">
    <input type="radio" name="rule" id="radio1" value="yes"
      class="form-check-input">
    <label class="form-check-label" for="radio1">Согласен</label>
  </div>
  <div class="form-check form-check-inline">
    <input type="radio" name="rule" id="radio2" value="no"
      class="form-check-input" checked="">
    <label class="form-check-label" for="radio2">Нет</label>
  </div>
</div>
<button type="submit" class="btn btn-primary">Отправить</button>
</form>
</div>

```

Логин:

Введите логин

Пароль:

Введите пароль

От 6 до 16 символов

[Правила участия](#)

Согласен Нет

Отправить

Рис. 3.26. Выравнивание элементов формы по вертикали в Bootstrap 4

3.2.2. Выравнивание элементов по горизонтали

Если к тегу `<form>` добавить стилевой класс `form-inline`, то форма превратится во flex-контейнер с выравниванием элементов по горизонтали, поэтому мы можем применять к нему все стилевые классы, рассмотренные в *разд. 2.1*. Например, ис-

пользовать стилевой класс `justify-content-center` для выравнивания всех элементов по центру флекс-контейнера. Стили для класса `form-inline` в Bootstrap 4:

```
.form-inline {
  display: flex;
  flex-flow: row wrap;
  align-items: center;
}
```

Начиная от точки останова `sm` (576 px), элементы с классом `form-group` становятся вложенными флекс-контейнерами, а для элементов с классом `form-control` применяются следующие стили в Bootstrap 4:

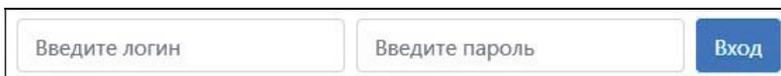
```
.form-inline .form-control {
  display: inline-block;
  width: auto;
  vertical-align: middle;
}
```

По умолчанию все элементы следуют друг за другом без внешних отступов. Используйте стилевые классы из *разд. 1.4.1*, чтобы их добавить. Если место ограничено и нет возможности использовать пояснительные надписи, то лучше оставить их в коде, но не отображать пользователю, добавив к тегу `<label>` стилевой класс `sr-only`.

Пример выравнивания элементов формы по горизонтали в Bootstrap 4 приведен в листинге 3.2. Результат выполнения показан на рис. 3.27.

Листинг 3.2. Выравнивание элементов формы по горизонтали в Bootstrap 4

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <form action="#" method="GET"
    class="form-inline justify-content-md-center">
    <div class="form-group mr-sm-2 mb-2">
      <label for="login" class="sr-only">Логин:</label>
      <input type="text" class="form-control"
        name="login" id="login" placeholder="Введите логин">
    </div>
    <div class="form-group mr-sm-2 mb-2">
      <label for="passwd" class="sr-only">Пароль:</label>
      <input type="password" class="form-control"
        name="passwd" id="passwd" placeholder="Введите пароль">
    </div>
    <button type="submit" class="btn btn-primary mb-2">Вход</button>
  </form>
</div>
```



The image shows a rendered HTML form. It consists of two input fields and a submit button, all horizontally aligned. The first input field is labeled 'Введите логин' (Enter login) and is a light gray box. The second input field is labeled 'Введите пароль' (Enter password) and is also a light gray box. To the right of these two fields is a blue button with white text that says 'Вход' (Login).

Рис. 3.27. Выравнивание элементов по горизонтали

В Bootstrap 5 классы `form-inline` и `form-group` были удалены. Вместо этих классов можно использовать классы, предназначенные для работы с флекс-контейнерами (см. *разд. 2.1*):

```
<div class="container">
  <form action="#" method="GET"
    class="d-md-flex flex-md-wrap justify-content-md-center">
    <div class="mr-sm-2 mb-2">
      <label for="login" class="sr-only">Логин:</label>
      <input type="text" class="form-control"
        name="login" id="login" placeholder="Введите логин">
    </div>
    <div class="mr-sm-2 mb-2">
      <label for="passwd" class="sr-only">Пароль:</label>
      <input type="password" class="form-control"
        name="passwd" id="passwd" placeholder="Введите пароль">
    </div>
    <button type="submit" class="btn btn-primary mb-2">Вход</button>
  </form>
</div>
```

Можно также воспользоваться системой сеток Bootstrap 5, указав для формы стили-**вые классы** `row`, `row-cols-md-auto` и `g-2`, а для колонок — класс `col-12`:

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <form action="#" method="GET"
    class="row row-cols-md-auto g-2 justify-content-md-center">
    <div class="col-12">
      <label for="login" class="sr-only">Логин:</label>
      <input type="text" class="form-control"
        name="login" id="login" placeholder="Введите логин">
    </div>
    <div class="col-12">
      <label for="passwd" class="sr-only">Пароль:</label>
      <input type="password" class="form-control"
        name="passwd" id="passwd" placeholder="Введите пароль">
    </div>
    <div class="col-12">
      <button type="submit" class="btn btn-primary mb-2">Вход</button>
    </div>
  </form>
</div>
```

3.2.3. Выравнивание элементов по сетке

Используя систему сеток библиотеки Bootstrap (см. *разд. 2.2*), можно произвести выравнивание элементов формы по ячейкам сетки. Чтобы уменьшить внутренние отступы (с 15 px до 5 px), следует в Bootstrap 4 вместо класса `row` добавлять класс `form-row`. Описание колонок производится обычным образом.

Рассмотрим пример (листинг 3.3). До точки останова `sm` (576 px) все три элемента будут размещаться по вертикали друг под другом (рис. 3.28). Каждая колонка размещается на отдельной строке и занимает всю ее ширину. Начиная с точки останова `sm`, элементы будут размещаться по горизонтали (рис. 3.29). Для кнопки укажем ширину в две ячейки и добавим для нее стилевой класс `btn-block`, а две колонки с текстовыми полями будут разделять всю оставшуюся ширину поровну.

Листинг 3.3. Выравнивание элементов формы по сетке в Bootstrap 4

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <form action="#" method="GET">
    <div class="form-row">
      <div class="col-sm mb-2">
        <label for="login" class="sr-only">Логин:</label>
        <input type="text" class="form-control"
          name="login" id="login" placeholder="Логин">
      </div>
      <div class="col-sm mb-2">
        <label for="passwd" class="sr-only">Пароль:</label>
        <input type="password" class="form-control"
          name="passwd" id="passwd" placeholder="Пароль">
      </div>
      <div class="col-sm-2 mb-2">
        <button type="submit"
          class="btn btn-primary btn-block">Вход</button>
      </div>
    </div>
  </form>
</div>
```

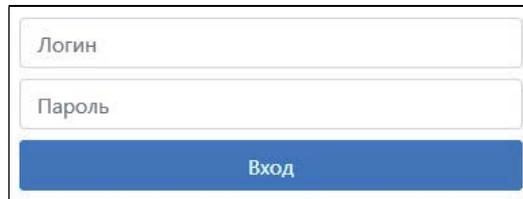


Рис. 3.28. Выравнивание элементов формы по сетке (ширина меньше 576 px)

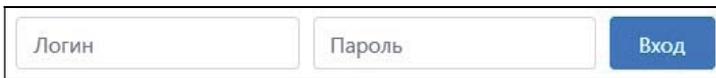


Рис. 3.29. Выравнивание элементов формы по сетке (ширина больше или равна 576 px)

В Bootstrap 5 класс `form-row` отсутствует. Вместо него следует использовать класс `row`, а размер отступов задавать с помощью классов `g-*` (см. *разд. 2.2.1*):

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <form action="#" method="GET">
```

```

<div class="row gx-sm-2">
  <div class="col-sm mb-2">
    <label for="login" class="sr-only">Логин:</label>
    <input type="text" class="form-control"
      name="login" id="login" placeholder="Логин">
  </div>
  <div class="col-sm mb-2">
    <label for="passwd" class="sr-only">Пароль:</label>
    <input type="password" class="form-control"
      name="passwd" id="passwd" placeholder="Пароль">
  </div>
  <div class="col-sm-2 mb-2">
    <button type="submit"
      class="btn btn-primary btn-block">Вход</button>
  </div>
</div>
</form>
</div>

```

Можно комбинировать строки с сеткой и строки с группой, создавая тем самым очень сложные комбинации выравнивания элементов формы. При этом весьма удобно описывать разное расположение элементов при различных размерах экрана.

Давайте рассмотрим это на примере (листинг 3.4). До точки останова `sm` (576 px) все элементы будут размещаться по вертикали друг под другом. Начиная с точки останова `sm`, поля **Логин** и **Пароль** будут размещаться по горизонтали, разделяя всю ширину строки поровну (рис. 3.30). Поля **Сообщение** и **URL** займут всю ширину строки, а поля **E-mail** и **Код** поделят строку. Причем поле **E-mail** займет 8 ячеек, а поле **Код** — 4 ячейки. Начиная с точки останова `md`, поля **URL**, **E-mail** и **Код** будут размещаться по горизонтали на одной строке (рис. 3.31). Причем поле **URL** займет 5 ячеек, поле **E-mail** — 5 ячеек, а поле **Код** — 2 ячейки. Поле **Сообщение** и кнопка всегда будут на отдельных строках.

Листинг 3.4. Комбинация стилей выравнивания в Bootstrap 4

```

<!-- Только в Bootstrap 4 -->
<div class="container">
  <form action="#" method="GET">
    <!-- Строка сетки -->
    <div class="form-row">
      <div class="form-group col-sm">
        <label for="login">Логин:</label>
        <input type="text" class="form-control"
          name="login" id="login" placeholder="Логин">
      </div>
      <div class="form-group col-sm">
        <label for="passwd">Пароль:</label>
        <input type="password" class="form-control"
          name="passwd" id="passwd" placeholder="Пароль">
      </div>
    </div>
  </form>
</div>

```

```

<!-- Обычная группа -->
<div class="form-group">
  <label for="msg">Сообщение:</label>
  <textarea class="form-control" name="msg" id="msg"></textarea>
</div>
<!-- Строка сетки -->
<div class="form-row">
  <div class="form-group col-sm-12 col-md-5">
    <label for="url">URL:</label>
    <input type="url" class="form-control"
      name="url" id="url" placeholder="URL">
  </div>
  <div class="form-group col-sm-8 col-md-5">
    <label for="email">E-mail:</label>
    <input type="email" class="form-control"
      name="email" id="email" placeholder="E-mail">
  </div>
  <div class="form-group col-sm-4 col-md-2">
    <label for="code">Код:</label>
    <input type="text" class="form-control"
      name="code" id="code" placeholder="Код">
  </div>
</div>
</div>
<!-- Кнопка на строке -->
<button type="submit"
  class="btn btn-primary mb-2">Отправить</button>
</form>
</div>

```

The screenshot shows a web form with a grid layout. The form is contained within a rectangular border. At the top, there are two input fields: "Логин:" (Login) and "Пароль:" (Password). Below these is a "Сообщение:" (Message) field, which is a large text area. Underneath is a "URL:" field. At the bottom, there are two input fields: "E-mail:" and "Код:" (Code). A blue "Отправить" (Submit) button is located at the bottom left of the form.

Рис. 3.30. Комбинация стилей выравнивания (ширина больше или равна 576 px)

Рис. 3.31. Комбинация стилей выравнивания (ширина больше или равна 768 px)

Пример комбинации стилей выравнивания для Bootstrap 5:

```

<!-- Только в Bootstrap 5 -->
<div class="container">
  <form action="#" method="GET">
    <!-- Строка сетки -->
    <div class="row gx-sm-2">
      <div class="mb-3 col-sm">
        <label for="login" class="form-label">Логин:</label>
        <input type="text" class="form-control"
          name="login" id="login" placeholder="Логин">
      </div>
      <div class="mb-3 col-sm">
        <label for="passwd" class="form-label">Пароль:</label>
        <input type="password" class="form-control"
          name="passwd" id="passwd" placeholder="Пароль">
      </div>
    </div>
    <!-- Обычная группа -->
    <div class="mb-3">
      <label for="msg" class="form-label">Сообщение:</label>
      <textarea class="form-control" name="msg" id="msg"></textarea>
    </div>
    <!-- Строка сетки -->
    <div class="row gx-sm-2">
      <div class="mb-3 col-sm-12 col-md-5">
        <label for="url" class="form-label">URL:</label>
        <input type="url" class="form-control"
          name="url" id="url" placeholder="URL">
      </div>
      <div class="mb-3 col-sm-8 col-md-5">
        <label for="email" class="form-label">E-mail:</label>
        <input type="email" class="form-control"
          name="email" id="email" placeholder="E-mail">
      </div>
    </div>
  </form>
</div>

```

```

<div class="mb-3 col-sm-4 col-md-2">
  <label for="code" class="form-label">Код:</label>
  <input type="text" class="form-control"
    name="code" id="code" placeholder="Код">
</div>
</div>
<!-- Кнопка на строке -->
<button type="submit"
  class="btn btn-primary mb-2">Отправить</button>
</form>
</div>

```

Если надо надпись разместить слева, а элемент справа на одной строке, то нужно выполнить следующие действия:

1. Создать родительский контейнер и добавить к нему стилевые классы `row` и `form-group` (в Bootstrap 4) или `mb-3` (в Bootstrap 5).
2. Вложить в родительский контейнер тег `<label>` с текстом надписи и добавить к нему стилевые классы `col-form-label` и `col-*`.
3. После тега `<label>` создать контейнер со стилевым классом `col-*` и вложить в него элемент управления.

Давайте рассмотрим это на примере (листинг 3.5). До точки останова `sm` (576 px) все элементы будут размещаться по вертикали друг под другом. Начиная с точки останова `sm`, колонка с текстом надписи будет занимать 3 ячейки, а колонка с элементом управления — 9 ячеек. Начиная с точки останова `md`, колонка с текстом надписи будет занимать 2 ячейки, а колонка с элементом управления — 10 ячеек (рис. 3.32). Для кнопки ситуация такая же, только вместо колонки с надписью слева выведем смещение, чтобы кнопка выравнивалась по элементам управления.

Листинг 3.5. Вывод надписи слева от элемента управления

```

<div class="container">
  <form action="#" method="GET">
    <div class="mb-3 row">
      <label for="login" class="col-sm-3 col-md-2 col-form-label">
        Логин:</label>
      <div class="col-sm-9 col-md-10">
        <input type="text" class="form-control"
          name="login" id="login" placeholder="Логин">
      </div>
    </div>
    <div class="mb-3 row">
      <label for="passwd" class="col-sm-3 col-md-2 col-form-label">
        Пароль:</label>
      <div class="col-sm-9 col-md-10">
        <input type="password" class="form-control"
          name="passwd" id="passwd" placeholder="Пароль">
      </div>
    </div>
  </form>
</div>

```

```

<label for="msg" class="col-sm-3 col-md-2 col-form-label">
Сообщение:</label>
<div class="col-sm-9 col-md-10">
  <textarea rows="5" class="form-control"
    name="msg" id="msg"></textarea>
</div>
</div>
<div class="mb-3 row">
  <div class="offset-sm-3 offset-md-2 col-sm-9 col-md-10">
    <button type="submit"
      class="btn btn-primary">Отправить</button>
  </div>
</div>
</form>
</div>

```

The image shows a web form with three input fields and a submit button. The first field is labeled "Логин:" and contains the text "Логин". The second field is labeled "Пароль:" and contains the text "Пароль". The third field is labeled "Сообщение:" and is empty. Below the fields is a blue button labeled "Отправить".

Рис. 3.32. Вывод надписи слева от элемента управления

Чтобы размер шрифта надписи и размер шрифта внутри элемента управления был одинаковым, нужно выполнить следующие действия:

4. При указании класса `form-control-sm` (уменьшенный шрифт) нужно к надписи добавить класс `col-form-label-sm`.
5. При указании класса `form-control-lg` (увеличенный шрифт) нужно к надписи добавить класс `col-form-label-lg`.

Пример:

```

<div class="mb-3 row">
  <label for="login"
    class="col-sm-3 col-md-2 col-form-label col-form-label-sm">
    Логин:</label>
  <div class="col-sm-9 col-md-10">
    <input type="text" class="form-control form-control-sm"
      name="login" id="login" placeholder="Логин">
  </div>
</div>

```

При добавлении группы переключателей и других элементов в качестве текста надписи можно использовать содержимое тега `<legend>`, вложенного в тег `<fieldset>`. К тегу `<fieldset>` добавляем стилевой класс `mb-3` и внутрь него вкладываем контейнер со стилевым классом `row`. Внутри контейнера создаем тег `<legend>` с надписью и добавляем к нему стилевые классы `col-form-label`, `col-*` и `pt-0`, чтобы убрать внутренний отступ сверху. После тега `<legend>` добавляем контейнер со стилевым классом `col-*` и внутри него размещаем группу переключателей:

```
<fieldset class="mb-3">
  <div class="row">
    <legend class="col-form-label col-sm-3 col-md-2 pt-0">Пол:</legend>
    <div class="col-sm-9 col-md-10">
      <div class="form-check">
        <input type="radio" name="sex" id="radio1" value="male"
              class="form-check-input">
        <label class="form-check-label" for="radio1">Мужской</label>
      </div>
      <div class="form-check">
        <input type="radio" name="sex" id="radio2" value="female"
              class="form-check-input">
        <label class="form-check-label" for="radio2">Женский</label>
      </div>
    </div>
  </div>
</fieldset>
```

3.2.4. Группировка элементов формы

Парный тег `<fieldset>` позволяет сгруппировать элементы формы. Веб-браузеры обычно вокруг группы отображают рамку, но в библиотеке Bootstrap рамка и отступы полностью убраны. На линии рамки обычно с помощью тега `<legend>` можно поместить надпись, но в библиотеке Bootstrap 4 элемент сделан блочным и он занимает всю ширину родительского элемента, а текст надписи отображается увеличенным шрифтом. В Bootstrap 5 для тега `<legend>` прописаны следующие правила:

```
legend {
  float: left;
  width: 100%;
  padding: 0;
  margin-bottom: 0.5rem;
  font-size: calc(1.275rem + 0.3vw);
  line-height: inherit;
  white-space: normal;
}
@media (min-width: 1200px) {
  legend {
    font-size: 1.5rem;
  }
}
```

```

legend + * {
  clear: left;
}

```

Как видите, стили для этих элементов в библиотеке Bootstrap изменены. Тем не менее тег `<fieldset>` все также полезен для объединения элементов в группу. Если добавить параметр `disabled`, то все элементы внутри группы станут недоступными:

```

<div class="container">
  <form action="#" method="GET">
    <fieldset disabled>
      <legend>Название группы</legend>
      <div class="mb-3">
        <label for="login">Логин:</label>
        <input type="text" class="form-control"
          name="login" id="login" placeholder="Введите логин">
      </div>
      <div class="mb-3">
        <label for="passwd">Пароль:</label>
        <input type="password" class="form-control"
          name="passwd" id="passwd" placeholder="Введите пароль">
      </div>
      <button type="submit" class="btn btn-primary">Отправить</button>
    </fieldset>
  </form>
</div>

```

3.2.5. Выравнивание кнопок

Кнопки являются наиболее часто используемыми элементами управления, поэтому давайте рассмотрим варианты выравнивания кнопок. По умолчанию кнопки следуют друг за другом по горизонтали и выравниваются по левому краю родительского контейнера:

```

<div class="mb-3">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button" class="btn btn-outline-info">Кнопка 2</button>
  <button type="button" class="btn btn-outline-danger">Кнопка 3</button>
</div>

```

Для выравнивания кнопок по центру нужно для родительского контейнера добавить стилевой класс `text-center`:

```

<div class="mb-3 text-center">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button" class="btn btn-outline-info">Кнопка 2</button>
  <button type="button" class="btn btn-outline-danger">Кнопка 3</button>
</div>

```

Чтобы произвести выравнивание по правому краю — класс `text-right`:

```
<div class="mb-3 text-right">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button" class="btn btn-outline-info">Кнопка 2</button>
  <button type="button" class="btn btn-outline-danger">Кнопка 3</button>
</div>
```

Если кнопка одна, то ее можно сделать блочной (класс `d-block`) и добавить к ней класс для создания внешнего отступа (см. *разд. 1.4.1*):

```
<!-- Выравнивание по центру -->
<button type="button"
  class="btn btn-outline-info d-block mx-auto">Кнопка</button>
<!-- Выравнивание по правому краю -->
<button type="button"
  class="btn btn-outline-success d-block ml-auto">Кнопка</button>
```

Между кнопками по умолчанию существует пустое пространство, т. к. перед тегом `<button>` у нас есть символ переноса строки и пробелы, которые преобразуются в один пробел. Чтобы убрать этот пробел, можно указать все теги вплотную друг к другу или преобразовать родительский блок во flex-контейнер:

```
<div class="mb-3 d-flex">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button" class="btn btn-outline-info">Кнопка 2</button>
  <button type="button" class="btn btn-outline-danger">Кнопка 3</button>
</div>
```

Добавив стилевой класс `justify-content-center`, можно произвести выравнивание кнопок по центру. Для выравнивания кнопок по правому краю следует использовать класс `justify-content-end`:

```
<div class="mb-3 d-flex justify-content-end">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button" class="btn btn-outline-info">Кнопка 2</button>
  <button type="button" class="btn btn-outline-danger">Кнопка 3</button>
</div>
```

Используя классы для создания внешнего отступа (см. *разд. 1.4.1*), можно задать расстояние между кнопками, а также произвести выравнивание по разным сторонам. Например, первую кнопку отобразим около левого края, а две остальные — около правого края, причем между этими кнопками добавим отступ:

```
<div class="mb-3 d-flex">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button"
    class="btn btn-outline-info ml-auto">Кнопка 2</button>
  <button type="button"
    class="btn btn-outline-danger ml-1">Кнопка 3</button>
</div>
```

Произведем выравнивание кнопок по правому краю, причем первую кнопку сдвинем влево на величину `3 rem` относительно двух остальных, начиная с точки отсчета `sm`:

```
<div class="mb-3 d-sm-flex justify-content-sm-end">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button"
    class="btn btn-outline-info ml-sm-5">Кнопка 2</button>
  <button type="button"
    class="btn btn-outline-danger ml-sm-1">Кнопка 3</button>
</div>
```

С помощью системы сеток Bootstrap можно произвести любое другое выравнивание. Например, в листинге 3.5 мы выравнивали кнопку по элементам управления, а не по надписям.

3.2.6. Группа с кнопками

Для объединения кнопок в группу следует добавить стилевой класс `btn-group`, а также параметр `role` со значением `group` и параметр `aria-label` с описанием:

```
<div class="btn-group" role="group" aria-label="Группа">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button"
    class="btn btn-outline-info">Кнопка 2</button>
  <button type="button"
    class="btn btn-outline-danger">Кнопка 3</button>
</div>
```

При добавлении класса `btn-group` кнопки выстраиваются по горизонтали. Чтобы кнопки располагались по вертикали, следует использовать стилевой класс `btn-group-vertical`:

```
<div class="btn-group-vertical" role="group" aria-label="Группа">
  <button type="button"
    class="btn btn-outline-success">Кнопка 1</button>
  <button type="button"
    class="btn btn-outline-info">Кнопка 2</button>
  <button type="button"
    class="btn btn-outline-danger">Кнопка 3</button>
</div>
```

Если нужно изменить высоту всех кнопок внутри группы, то следует дополнительно добавить стилевой класс `btn-group-sm` (маленькие кнопки) или `btn-group-lg` (большие кнопки):

```
<div class="btn-group btn-group-sm"
  role="group" aria-label="Группа 1">
  <button type="button" class="btn btn-success">Маленькая</button>
  <button type="button" class="btn btn-info">Маленькая</button>
  <button type="button" class="btn btn-danger">Маленькая</button>
</div>
```

```
<div class="btn-group btn-group-lg ml-3"
  role="group" aria-label="Группа 2">
  <button type="button" class="btn btn-success">Большая</button>
  <button type="button" class="btn btn-info">Большая</button>
  <button type="button" class="btn btn-danger">Большая</button>
</div>
```

Обратите внимание: обе группы отобразились на одной строке, т. к. для класса `btn-group` CSS-атрибут `display` имеет значение `inline-flex`. Расстояние между группами можно регулировать с помощью стилевых классов для создания внешних отступов (см. *разд. 1.4.1*). Для выравнивания групп следует вложить их в контейнер и добавить к нему стилевой класс `btn-toolbar`, а также параметр `role` со значением `toolbar` и параметр `aria-label` с описанием. Для класса `btn-toolbar` CSS-атрибут `display` имеет значение `flex`, поэтому мы можем использовать стилевые классы для выравнивания групп внутри панели инструментов (см. *разд. 2.1*).

Создадим панель инструментов и добавим в нее две группы с кнопками. Зададим величину внешнего отступа между группами и выравнивание по правому краю панели инструментов:

```
<div class="btn-toolbar justify-content-end"
  role="toolbar" aria-label="Панель">
  <div class="btn-group" role="group" aria-label="Группа 1">
    <button type="button" class="btn btn-outline-success"> 1 </button>
    <button type="button" class="btn btn-outline-info"> 2 </button>
    <button type="button" class="btn btn-outline-danger"> 3 </button>
  </div>
  <div class="btn-group ml-3" role="group" aria-label="Группа 2">
    <button type="button" class="btn btn-outline-success"> 4 </button>
    <button type="button" class="btn btn-outline-info"> 5 </button>
  </div>
</div>
```

3.2.7. Группа с кнопками-переключателями

Кнопку-переключатель можно создать на основе командной кнопки, флажка или стандартного переключателя. Такой переключатель выглядит как обычная командная кнопка и имеет два состояния: включен и выключен (рис. 3.33). Во включенном состоянии к кнопке добавляется стилевой класс `active`, поэтому кнопка отображается в нажатом состоянии. При выключенном состоянии стилевой класс `active` удаляется, и переключатель выглядит как обычная командная кнопка.

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.button.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Button.VERSION);
```



Рис. 3.33. Кнопки-переключатели

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы переключателя требуется подключение файлов *jquery.min.js* и *bootstrap.min.js* (или *bootstrap.bundle.min.js*), а также добавление к переключателю параметра `data-toggle`. В Bootstrap 5 можно не подключать файл *jquery.min.js*, если не используется интерфейс доступа через jQuery.

Чтобы преобразовать командную кнопку в кнопку-переключатель, нужно выполнить следующие действия:

1. Добавить к командной кнопке стилевой класс `btn`, а также параметр `data-toggle` со значением `button`.
2. Если переключатель изначально должен находиться во включенном состоянии, то надо добавить стилевой класс `active` и параметр `aria-pressed` со значением `true`.
3. Для выключенного состояния надо добавить параметр `aria-pressed` со значением `false`.

Для определения состояния переключателя можно проверить наличие стилевого класса `active`, а также значение параметра `aria-pressed`. Для переключения состояния из программы с помощью jQuery, следует воспользоваться методом `button()` со значением `toggle`:

```
$('#btnToggle').button('toggle');
```

Добавим кнопку-переключатель и две командные кнопки:

```
<button type="button" id="btnToggle" data-toggle="button"
  aria-pressed="false"
  class="btn btn-outline-primary">Переключатель</button>
<button type="button" class="btn btn-info" id="btnTest">
  Проверить состояние переключателя</button>
<button type="button" class="btn btn-info" id="btnChange">
  Переключить состояние</button>
```

По нажатию кнопки с идентификатором `btnTest` выведем состояние переключателя в окно консоли, а по нажатию кнопки с идентификатором `btnChange` изменим состояние переключателя противоположным:

```
$('#btnTest').click( function() {
  var btn = $('#btnToggle');
  if (btn.attr('aria-pressed') === 'true')
    console.log('Включен');
  else
    console.log('Выключен');
  if ( btn.hasClass('active') )
    console.log('Включен (.active)');
  else
    console.log('Выключен (нет .active)');
});
$('#btnChange').click( function() {
  $('#btnToggle').button('toggle');
  console.log('toggle');
});
```

В этом примере мы воспользовались возможностями JavaScript-библиотеки jQuery. Для поиска элемента нужно передать CSS-селектор функции `$()`. Мы хотим найти кнопку по идентификатору, поэтому внутри кавычек указываем идентификатор, перед которым добавляется символ `#`. Все точно так же, как и при доступе к элементу из CSS. Функция `$()` возвращает объект jQuery с найденными элементами (в нашем случае — с одним элементом). Назначение обработчика события нажатия кнопки производится с помощью метода `click()`. В качестве параметра метод `click()` принимает функцию, которая будет вызвана при наступлении события.

Внутри обработчика нажатия первой кнопки находим переключатель по его идентификатору и сохраняем ссылку на него в переменной. Далее проверяем значение параметра `aria-pressed`. Получить это значение позволяет метод `attr()`. Если параметр имеет значение `'true'`, то переключатель находится во включенном состоянии, в противном случае — в выключенном. Обратите внимание: значение `'true'` является строкой, а не логическим значением `true`. Вместо проверки значения параметра `aria-pressed`, можно проверить наличие стилевых классов `active`. Выполнить проверку позволяет метод `hasClass()`, возвращающий логическое значение `true`, если стиливый класс, указанный в круглых скобках, существует. Сообщения о состоянии переключателя выводим в окно консоли с помощью метода `log()`.

Внутри обработчика нажатия второй кнопки находим переключатель по его идентификатору и сразу применяем метод `button()` для переключения состояния переключателя. Точно также мы могли бы поступить и в предыдущем случае: не сохранять ссылку на элемент в переменной, а сразу применить метод `attr()`. Сохранение ссылки в переменной удобно в случае многократного доступа к элементу. Сообщение о факте переключения выводим в окно консоли.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Button`:

```
<Объект> = new bootstrap.Button(<Элемент>)
```

Пример создания объекта:

```
var btn = document.getElementById('btnToggle');
var btnInstance = new bootstrap.Button(btn);
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект и создания объекта при отсутствии ссылки:

```
var btn = document.getElementById('btnToggle');
var btnInstance = bootstrap.Button.getInstance(btn);
if ( !btnInstance ) {
    btnInstance = new bootstrap.Button(btn);
}
```

Для переключения состояния кнопки из программы, следует воспользоваться методом `toggle()`:

```
btnInstance.toggle();
```

По нажатию кнопки с идентификатором `btnTest` выведем состояние переключателя в окно консоли, а по нажатию кнопки с идентификатором `btnChange` изменим состояние переключателя противоположным:

```
// Bootstrap 5 без jQuery
var btnTest = document.getElementById('btnTest');
btnTest.addEventListener('click', function() {
  var btn = document.getElementById('btnToggle');
  if (btn.getAttribute('aria-pressed') === 'true')
    console.log('Включен');
  else
    console.log('Выключен');
  if ( btn.classList.contains('active') )
    console.log('Включен (.active)');
  else
    console.log('Выключен (нет .active)');
}, false);

var btnChange = document.getElementById('btnChange');
btnChange.addEventListener('click', function() {
  var btn = document.getElementById('btnToggle');
  var btnInstance = bootstrap.Button.getInstance(btn);
  if ( !btnInstance ) {
    btnInstance = new bootstrap.Button(btn);
  }
  btnInstance.toggle();
  console.log('toggle');
}, false);
```

В Bootstrap 4 стандартный флажок можно преобразовать в кнопку-переключатель. Для этого нужно выполнить следующие действия:

1. Создать контейнер и добавить к нему стилевой класс `btn-group-toggle`. Чтобы состояние переключалось при нажатии кнопки, следует добавить параметр `data-toggle` со значением `buttons`.
2. Внутри контейнера создать тег `<label>` и добавить для него класс `btn`, а также класс для оформления кнопки — например, `btn-outline-primary`. Если переключатель изначально находится во включенном состоянии, то нужно добавить стилевой класс `active`.
3. Внутри тега `<label>` создать тег `<input>`, параметр `type` которого содержит значение `checkbox`, и вставить после него текст надписи. Добавить параметр `autocomplete` со значением `off`, чтобы веб-браузер автоматически не подставлял значение (например, при нажатии кнопки **Обновить**), и параметр `checked`, если переключатель изначально находится во включенном состоянии.

Контейнер может содержать несколько кнопок-переключателей, но состояние каждой из них не зависит от состояния других кнопок-переключателей.

Пример создания двух кнопок-переключателей:

```
<!-- Только в Bootstrap 4 -->
<div class="btn-toolbar mb-2" role="toolbar" aria-label="Панель">
  <div class="btn-group btn-group-toggle" role="group"
    data-toggle="buttons">
    <label class="btn btn-outline-primary active" id="btnToggle1">
      <input type="checkbox" id="check1"
        autocomplete="off" checked> Переключатель 1
    </label>
    <label class="btn btn-outline-primary" id="btnToggle2">
      <input type="checkbox" id="check2"
        autocomplete="off"> Переключатель 2
    </label>
  </div>
</div>
<button type="button" class="btn btn-info" id="btnTest">
  Проверить состояние переключателей</button>
```

По нажатию кнопки с идентификатором `btnTest` выведем текущее состояние кнопок-переключателей (проверим наличие стилевого класса `active`):

```
$('#btnTest').click( function() {
  if ( $('#btnToggle1').hasClass('active') )
    console.log('1 - Включен');
  else
    console.log('1 - Выключен');
  if ( $('#btnToggle2').hasClass('active') )
    console.log('2 - Включен');
  else
    console.log('2 - Выключен');
});
```

Если надо, чтобы из группы можно было включить только один переключатель, то в Bootstrap 4 нужно выполнить следующие действия:

1. Создать контейнер и добавить к нему стилевой класс `btn-group-toggle`. Чтобы состояние переключалось при нажатии кнопки, следует добавить параметр `data-toggle` со значением `buttons`.
2. Внутри контейнера создать тег `<label>` и добавить для него класс `btn`, а также класс для оформления кнопки — например, `btn-outline-primary`. Если переключатель изначально находится во включенном состоянии, то нужно добавить стилевой класс `active`.
3. Внутри тега `<label>` создать тег `<input>`, параметр `type` которого содержит значение `checkbox`, и вставить после него текст надписи. Параметр `name` у всех переключателей внутри группы должен иметь одинаковое значение. Добавить параметр `autocomplete` со значением `off`, чтобы веб-браузер автоматически не подставлял значение (например, при нажатии кнопки **Обновить**), и параметр `checked`, если переключатель изначально находится во включенном состоянии.

Пример создания двух зависимых друг от друга кнопок-переключателей:

```
<!-- Только в Bootstrap 4 -->
<div class="btn-toolbar mb-2" role="toolbar" aria-label="Панель">
  <div class="btn-group btn-group-toggle" role="group"
    data-toggle="buttons">
    <label class="btn btn-outline-primary active" id="btnToggle1">
      <input type="radio" name="radioToggle" id="radio1"
        autocomplete="off" checked> Переключатель 1
    </label>
    <label class="btn btn-outline-primary" id="btnToggle2">
      <input type="radio" name="radioToggle" id="radio2"
        autocomplete="off"> Переключатель 2
    </label>
  </div>
</div>
<button type="button" class="btn btn-info" id="btnTest">
  Проверить состояние переключателей</button>
```

Найдем включенный переключатель внутри группы:

```
$('#btnTest').click( function() {
  if ( $('#btnToggle1').hasClass('active') )
    console.log('Включен переключатель 1');
  else if ( $('#btnToggle2').hasClass('active') )
    console.log('Включен переключатель 2');
  else
    console.log('Не включены');
});
```

Если переключателей много, то можно воспользоваться следующим кодом:

```
$('#btnTest').click( function() {
  var elem = $("label.active [name='radioToggle']:checked");
  if (elem.length === 1)
    console.log('Включен переключатель ' + elem.prop('id'));
});
```

Здесь мы находим элемент, параметр `name` которого равен `radioToggle`, причем он выбран (содержит псевдокласс `:checked`) и вложен в тег `<label>` со стилевым классом `active`. Далее проверяем его наличие, сравнивая значение свойства `length` с единицей. Свойство `length` содержит количество найденных элементов, и если ничего не найдено, то оно будет содержать значение `0`. Далее выводим в окно консоли идентификатор найденного переключателя, передавая название свойства `id` (идентификатор элемента) методу `prop()`.

Чтобы в Bootstrap 5 стандартный флажок преобразовать в кнопку-переключатель, нужно выполнить следующие действия:

1. Создать тег `<input>`, параметр `type` которого содержит значение `checkbox`, со стилевым классом `btn-check`. Добавить к тегу параметр `autocomplete` со значением `off`, чтобы веб-браузер автоматически не подставлял значение (например, при нажатии кнопки **Обновить**). Если переключатель изначально находится во включенном состоянии, то нужно добавить параметр `checked`.

2. После тега `<input>` создать тег `<label>` с текстом надписи и добавить для него стилевой класс `btn`, а также класс для оформления кнопки — например, `btn-outline-primary`.

Группа может содержать несколько кнопок-переключателей, но состояние каждой из них не зависит от состояния других кнопок-переключателей. Пример создания двух кнопок-переключателей:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <input type="checkbox" class="btn-check" id="check1"
    autocomplete="off" checked>
  <label class="btn btn-outline-primary" for="check1">
    Переключатель 1</label>

  <input type="checkbox" class="btn-check" id="check2"
    autocomplete="off">
  <label class="btn btn-outline-primary" for="check2">
    Переключатель 2</label>
</div>
<button type="button" class="btn btn-info" id="btnTest">
  Проверить состояние переключателей</button>
```

Компонент `btn-check` использует для переключения состояния только CSS и не требует подключения JavaScript-файлов. Программа может быть нужна только для определения текущего состояния переключателя. В качестве примера выведем текущее состояние кнопок-переключателей при нажатии кнопки с идентификатором `btnTest` (проверим наличие псевдокласса `:checked`):

```
$('#btnTest').click( function() {
  if ( $('#check1').is(':checked') )
    console.log('1 - Включен');
  else
    console.log('1 - Выключен');

  if ( $('#check2').is(':checked') )
    console.log('2 - Включен');
  else
    console.log('2 - Выключен');
});
```

Если надо, чтобы из группы можно было включить только один переключатель, то в Bootstrap 5 нужно выполнить следующие действия:

1. Создать тег `<input>`, параметр `type` которого содержит значение `radio`, со стилевым классом `btn-check`. Добавить к тегу параметр `autocomplete` со значением `off`, чтобы веб-браузер автоматически не подставлял значение (например, при нажатии кнопки **Обновить**). Если переключатель изначально находится во включенном состоянии, то нужно добавить параметр `checked`.
2. После тега `<input>` создать тег `<label>` с текстом надписи и добавить для него стилевой класс `btn`, а также класс для оформления кнопки — например, `btn-outline-primary`.

3. Повторить процедуру несколько раз в зависимости от нужного количества переключателей. Параметр `name` у всех переключателей внутри группы должен иметь одинаковое значение.

Пример создания двух зависимых друг от друга кнопок-переключателей:

```
<!-- Только в Bootstrap 5 -->
<div class="mb-3">
  <div class="btn-group">
    <input type="radio" class="btn-check" id="radio1"
      name="radioToggle" autocomplete="off" checked>
    <label class="btn btn-outline-primary" for="radio1">
      Переключатель 1</label>

    <input type="radio" class="btn-check" id="radio2"
      name="radioToggle" autocomplete="off">
    <label class="btn btn-outline-primary" for="radio2">
      Переключатель 2</label>
  </div>
</div>
<button type="button" class="btn btn-info" id="btnTest">
  Проверить состояние переключателей</button>
```

Найдем включенный переключатель внутри группы:

```
$('#btnTest').click( function() {
  if ( $('#radio1').is(':checked') )
    console.log('Включен переключатель 1');
  else if ( $('#radio2').is(':checked') )
    console.log('Включен переключатель 2');
  else
    console.log('Не включены');
});
```

3.3. Проверка данных формы

Многие элементы управления, добавленные в HTML 5, поддерживают автоматическую проверку введенного значения — например, поля для ввода числа (`type="number"`), адреса электронной почты (`type="email"`), URL (`type="url"`) и др. Если данные введены неправильно, то при потере фокуса или при попытке отправить форму элемент обводится рамкой красного цвета, и отправка формы прерывается. При попытке отправки формы рядом с первым элементом с неправильно введенными данными отображается всплывающая подсказка с информацией о проблеме.

Однако если поле не заполнено, то проверка автоматически не выполняется. Чтобы указать, что поле обязательно для заполнения, следует добавить параметр `required`:

```
<input type="text" class="form-control" required
  name="login" id="login" placeholder="Логин">
```

С помощью параметра `pattern` можно задать шаблон вводимого значения в виде регулярного выражения из языка JavaScript.

Зададим шаблон для номера мобильного телефона и сделаем поле обязательным для заполнения:

```
<input type="tel" class="form-control" name="phone" id="phone"
      placeholder="+0 (000) 000-00-00" required
      pattern="\+[1-9] \([0-9]{3}\) [0-9]{3}-[0-9]{2}-[0-9]{2}">
```

Если для тега `<form>` добавить параметр `novalidate`, то значения, занесенные в поля ввода, проверяться не будут:

```
<form action="#" method="GET" novalidate>
```

Для изменения стилей можно воспользоваться следующими псевдоклассами:

- ◆ `:optional` — необязательный элемент управления (тег которого не имеет параметра `required`);
- ◆ `:required` — обязательный элемент управления (тег которого имеет параметр `required`);
- ◆ `:valid` — элемент управления, в котором указано корректное значение;
- ◆ `:invalid` — элемент управления, в котором указано некорректное значение.

ПРИМЕЧАНИЕ

Следует учитывать, что злоумышленник может изменить HTML-код, поэтому на веб-сервере нужно в обязательном порядке выполнить проверку полученных данных, а не полагаться на веб-браузер. Тем не менее проверка на стороне клиента позволит снизить нагрузку на веб-сервер, поэтому отказываться от такой проверки не следует. Данные нужно проверять и в веб-браузере, и на веб-сервере.

Чтобы воспользоваться стилевым оформлением для валидации формы из библиотеки Bootstrap, следует отключить стандартную проверку веб-браузером, добавив к тегу `<form>` параметр `novalidate`. Обратите внимание: добавление параметра `novalidate` не отключает возможность проверки с помощью JavaScript. Запросить такую проверку позволяет метод `checkValidity()` объекта формы. Если все элементы формы успешно прошли проверку, то метод вернет значение `true`, в противном случае — значение `false`.

В зависимости от условия (например, при неудачной попытке отправки данных формы) следует добавить к тегу `<form>` стилевой класс `was-validated`. Элементы управления, для которых применен псевдокласс `:valid`, при этом обводятся рамкой зеленого цвета, а справа отображается значок в виде галочки зеленого цвета (рис. 3.34). Элементы управления, для которых применен псевдокласс `:invalid`, обводятся рамкой красного цвета, а справа отображается значок красного цвета в виде круга с восклицательным знаком внутри (рис. 3.35).



Рис. 3.34. Текстовое поле с псевоклассом `:valid`



Рис. 3.35. Текстовое поле с псевоклассом `:invalid`

Рядом с элементом можно отобразить различные сообщения, которые будут показываться в зависимости от статуса проверки данных. Чтобы создать сообщение, отображаемое при корректных данных, следует поместить его в контейнер со стилевым классом `valid-feedback`. Цвет текста такого сообщения будет зеленым (рис. 3.36):

```
<div class="valid-feedback">
  Данные введены правильно!
</div>
```

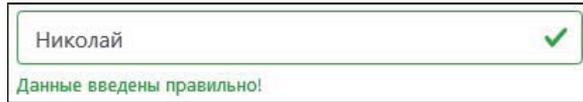


Рис. 3.36. Сообщение со стилевым классом `valid-feedback`

Чтобы создать сообщение, отображаемое при некорректных данных, следует поместить его в контейнер со стилевым классом `invalid-feedback`. Цвет текста такого сообщения будет красным (рис. 3.37):

```
<div class="invalid-feedback">
  Пожалуйста, заполните поле!
</div>
```



Рис. 3.37. Сообщение со стилевым классом `invalid-feedback`

Давайте создадим форму с одним текстовым полем и кнопкой для отправки данных:

```
<div class="container">
  <form action="#" method="GET" id="form1" novalidate>
    <div class="mb-3">
      <label for="login">Логин:</label>
      <input type="text" class="form-control" required
        name="login" id="login" placeholder="Логин">
      <div class="valid-feedback">
        Данные введены правильно!
      </div>
      <div class="invalid-feedback">
        Пожалуйста, заполните поле!
      </div>
    </div>
    <button type="submit"
      class="btn btn-primary mb-2">Отправить</button>
  </form>
</div>
```

Теперь напишем программу на jQuery, которая назначает обработчик события отправки данных формы:

```
$('#form1').submit( function(e) {
  if ( !this.checkValidity() ) {
```

```

        this.classList.add('was-validated');
        return false;
    }
    return true;
});

```

Сначала находим объект формы по идентификатору, а затем вызываем метод `submit()`, который назначает обработчик отправки формы. Внутри обработчика проверяется значение, возвращаемое методом `checkValidity()`. Если метод возвращает значение `true`, то все данные корректные, и можно разрешить отправку формы, возвращая из обработчика значение `true`. Если метод возвращает значение `false`, то добавляем к объекту формы стиливой класс `was-validated` и запрещаем отправку формы, возвращая из обработчика значение `false`. Обратите внимание: объект формы доступен внутри обработчика через указатель `this`, а объект события — через параметр `e`.

Вместо контейнеров с сообщениями можно использовать всплывающие подсказки. Обратите внимание: для корректного отображения подсказки родительский контейнер должен иметь относительное позиционирование (например, содержать стиливой класс `position-relative`). Чтобы создать подсказку, отображаемую при корректных данных, следует поместить сообщение в контейнер со стиливым классом `valid-tooltip`. Цвет фона такого сообщения будет зеленым (рис. 3.38):

```

<div class="valid-tooltip">
    Данные введены правильно!
</div>

```



Рис. 3.38. Подсказка со стиливым классом `valid-tooltip`

Чтобы создать подсказку, отображаемую при некорректных данных, следует поместить сообщение в контейнер со стиливым классом `invalid-tooltip`. Цвет фона такого сообщения будет красным (рис. 3.39):

```

<div class="invalid-tooltip">
    Пожалуйста, заполните поле!
</div>

```



Рис. 3.39. Подсказка со стиливым классом `invalid-tooltip`

Пример использования подсказок (подключите программу из предыдущего примера и нажмите кнопку **Отправить**, чтобы увидеть результат):

```

<div class="container">
    <form action="#" method="GET" id="form1" novalidate>
        <div class="mb-5 position-relative">

```

```

<label for="login">Логин:</label>
<input type="text" class="form-control" required
      name="login" id="login" placeholder="Логин">
<div class="valid-tooltip">
  Данные введены правильно!
</div>
<div class="invalid-tooltip">
  Пожалуйста, заполните поле!
</div>
</div>
<button type="submit"
        class="btn btn-primary mb-2">Отправить</button>
</form>
</div>

```

Если проверка данных производится, например, на стороне сервера, то вместо использования стилевого класса `was-validated` и псевдоклассов `:valid` и `:invalid` можно воспользоваться стилевыми классами `is-valid` и `is-invalid`. Если данные корректны, то к элементу добавляем стилевой класс `is-valid`, в противном случае — класс `is-invalid`. Рассмотрим пример:

```

<div class="container">
  <form action="" method="GET" id="form1" novalidate>
    <div class="mb-3">
      <label for="login">Логин:</label>
      <input type="text" class="form-control"
            name="login" id="login" placeholder="Логин">
      <div class="invalid-feedback">
        Пожалуйста, заполните поле!
      </div>
    </div>
    <button type="submit"
          class="btn btn-primary mb-2">Отправить</button>
  </form>
</div>

```

Это обычная форма с отключенной проверкой данных. Если сейчас нажать кнопку **Отправить**, то форма в любом случае будет отправлена. Давайте напишем программу на jQuery, которая при попытке отправить форму проверяет заполнение поля **Логин** и, если оно не заполнено, добавляет стилевой класс `is-invalid` и прерывает отправку формы. Если поле заполнено, то разрешим отправку формы. При получении полей фокуса ввода удалим стилевой класс `is-invalid`:

```

$('#form1').submit( function(e) {
  var elem = $('#login')[0];
  if (elem && elem.value.length === 0) {
    elem.classList.add('is-invalid');
    return false;
  }
  return true;
});

```

```
$('#login').focus( function() {  
    $(this).removeClass('is-invalid');  
});
```

Обратите внимание на инструкцию:

```
var elem = $('#login')[0];
```

Она почти соответствует следующей инструкции на языке JavaScript (различие между инструкциями будет в возвращаемом значении при отсутствии элемента: в первом случае получим значение `undefined`, а во втором — значение `null`):

```
var elem = document.getElementById('login');
```

Функция `$()` возвращает коллекцию элементов jQuery. Если же мы указываем индекс внутри квадратных скобок, то получаем ссылку на DOM-элемент, с которым можно выполнять действия на языке JavaScript. Например, получать значение из поля с помощью свойства `value`.

Назначение обработчика для события получения фокуса ввода производится с помощью метода `focus()`. Внутри обработчика мы находим элемент (`$(this)`) и удаляем стилевой класс `is-invalid` с помощью метода `removeClass()`. На языке JavaScript эту инструкцию можно написать следующим образом:

```
this.classList.remove('is-invalid');
```

Использовать стилевое оформление библиотеки Bootstrap 4 при проверке данных формы мы можем со следующими элементами:

- ◆ поля `<input>` и `<textarea>` со стилевым классом `form-control`;
- ◆ тег `<select>` со стилевым классом `custom-select`;
- ◆ флажки и переключатели со стилевыми классами `form-check`, `custom-checkbox` и `custom-radio`. При использовании флажков и переключателей изменяется цвет надписи (зеленый или красный);
- ◆ поля для выбора файла со стилевым классом `custom-file`.

Отобразим сообщения только при ошибках для списка, флажка и переключателей в Bootstrap 4:

```
<!-- Только в Bootstrap 4 -->  
<div class="container">  
    <form action="#" method="GET" id="form2"  
        class="was-validated" novalidate>  
        <div class="mb-3">  
            <label for="list1">Надпись:</label>  
            <select name="list1" id="list1" class="custom-select" required>  
                <option value="">...</option>  
                <option value="1">Пункт1</option>  
                <option value="2">Пункт2</option>  
                <option value="3">Пункт3</option>  
            </select>  
            <div class="invalid-feedback">  
                Пожалуйста, выберите пункт из списка!  
            </div>  
        </div>
```

```

</div>
<div class="mb-3">
  <div class="custom-control custom-checkbox">
    <input type="checkbox" class="custom-control-input"
      name="check1" id="check1" required>
    <label class="custom-control-label" for="check1">Надпись</label>
    <div class="invalid-feedback">
      Пожалуйста, установите флажок!
    </div>
  </div>
</div>
</div>
<div class="mb-3">
  <div class="custom-control custom-radio">
    <input type="radio" name="sex" id="radio1" value="male"
      class="custom-control-input" required>
    <label class="custom-control-label"
      for="radio1">Мужской</label>
  </div>
  <div class="custom-control custom-radio">
    <input type="radio" name="sex" id="radio2" value="female"
      class="custom-control-input" required>
    <label class="custom-control-label"
      for="radio2">Женский</label>
    <div class="invalid-feedback">
      Пожалуйста, выберите переключатель!
    </div>
  </div>
</div>
</div>
<button type="submit"
  class="btn btn-primary mb-2">Отправить</button>
</form>
</div>

```

Теперь напишем программу на jQuery, которая назначает обработчик события отправки данных формы:

```

$('#form2').submit( function(e) {
  return this.checkValidity();
});

```

Использовать стилевое оформление библиотеки Bootstrap 5 при проверке данных формы мы можем со следующими элементами:

- ◆ поля `<input>` и `<textarea>` со стилевым классом `form-control`;
- ◆ тег `<select>` со стилевым классом `form-select`;
- ◆ флажки и переключатели со стилевым классом `form-check`. При использовании флажков и переключателей изменяется цвет надписи (зеленый или красный);
- ◆ поля для выбора файла со стилевым классом `form-file`.

Отообразим сообщения только при ошибках для списка, флажка и переключателей в Bootstrap 5:

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <form action="#" method="GET" id="form2"
    class="was-validated" novalidate>
    <div class="mb-3">
      <label for="list1" class="form-label">Надпись:</label>
      <select name="list1" id="list1" class="form-select" required>
        <option value="">...</option>
        <option value="1">Пункт1</option>
        <option value="2">Пункт2</option>
        <option value="3">Пункт3</option>
      </select>
      <div class="invalid-feedback">
        Пожалуйста, выберите пункт из списка!
      </div>
    </div>
    <div class="mb-3">
      <div class="form-check">
        <input type="checkbox" class="form-check-input"
          name="check1" id="check1" required>
        <label class="form-check-label" for="check1">Надпись</label>
        <div class="invalid-feedback">
          Пожалуйста, установите флажок!
        </div>
      </div>
    </div>
    <div class="mb-3">
      <div class="form-check">
        <input type="radio" name="sex" id="radio1" value="male"
          class="form-check-input" required>
        <label class="form-check-label"
          for="radio1">Мужской</label>
      </div>
      <div class="form-check">
        <input type="radio" name="sex" id="radio2" value="female"
          class="form-check-input" required>
        <label class="form-check-label"
          for="radio2">Женский</label>
        <div class="invalid-feedback">
          Пожалуйста, выберите переключатель!
        </div>
      </div>
    </div>
    <button type="submit"
      class="btn btn-primary mb-2">Отправить</button>
  </form>
</div>
```

3.4. Установка фокуса ввода

Если нужно, чтобы после загрузки страницы элемент оказался в фокусе ввода, то следует добавить параметр `autofocus`:

```
<input type="text" class="form-control" autofocus
      name="login" id="login" placeholder="Логин">
```

3.5. Порядок обхода элементов формы

Перемещение между элементами формы осуществляется с помощью клавиши `<Tab>` (в прямом направлении) или комбинации клавиш `<Shift>+<Tab>` (в обратном направлении). По умолчанию порядок обхода элементов формы совпадает с порядком следования элементов. Для изменения порядка обхода нужно к элементам добавить параметр `tabindex` с уникальным номером:

```
<div class="container">
  <div class="row">
    <div class="mb-3 col-sm">
      <input type="text" class="form-control" tabindex="2"
            name="login" id="login" placeholder="Логин">
    </div>
    <div class="mb-3 col-sm">
      <input type="password" class="form-control" tabindex="1"
            name="passwd" id="passwd" placeholder="Пароль">
    </div>
  </div>
</div>
```

В этом примере при нажатии клавиши `<Tab>` первым в фокусе ввода окажется поле **Пароль**, т. к. у него номер в параметре `tabindex` имеет меньшее значение. При повторном нажатии клавиши `<Tab>` фокус ввода переместится на поле **Логин**.

Если нужно исключить элемент из порядка обхода, то следует добавить параметр `tabindex` со значением `-1`:

```
<input type="text" class="form-control" tabindex="-1"
      name="login" id="login" placeholder="Логин">
```

3.6. Индикатор хода процесса

Отобразить текущее состояние хода длительного процесса в графической форме позволяет тег `<progress>`, который имеет следующие параметры:

- ◆ `value` — текущее значение;
- ◆ `max` — максимальное значение.

Пример:

```
<progress max="100" value="25"></progress>
<progress max="100" value="50"></progress>
<progress max="100" value="75"></progress>
```

Можно также воспользоваться тегом `<meter>`, который имеет следующие параметры:

- ◆ `value` — текущее значение (обязательный параметр);
- ◆ `min` — минимальное значение;
- ◆ `max` — максимальное значение;
- ◆ `low` — низкое значение;
- ◆ `optimum` — оптимальное значение;
- ◆ `high` — высокое значение.

Пример:

```
<meter min="0" max="100" optimum="50"
  low="50" high="75" value="40"></meter>
<meter min="0" max="100" optimum="50"
  low="50" high="75" value="60"></meter>
<meter min="0" max="100" optimum="50"
  low="50" high="75" value="90"></meter>
```

Вместо тегов `<progress>` и `<meter>` лучше воспользоваться компонентом `progress`, входящим в состав библиотеки `Bootstrap` (рис. 3.40). Для этого нужно выполнить следующие действия:

1. Создать родительский контейнер со стилевым классом `progress`, который будет исполнять роль индикатора максимального значения и фона компонента. Контейнер растягивается на всю доступную ширину и по умолчанию имеет высоту `1 rem`. Для изменения высоты следует использовать CSS-атрибут `height`.
2. Вложить в родительский контейнер элемент со стилевым классом `progress-bar`, который будет исполнять роль полосы, изменяющей свою длину в зависимости от текущего значения. Дополнительно следует задать следующие параметры:
 - `style` — в этом параметре указывается текущее значение в диапазоне от 0 до 100 процентов с помощью CSS-атрибута `width`. Вместо этого параметра можно воспользоваться стилевыми классами `w-25`, `w-50`, `w-75` или `w-100` (см. *разд. 1.9.2*);
 - `role` — со значением `"progressbar"`;
 - `aria-valuenow` — текущее значение;
 - `aria-valuemin` — минимальное значение;
 - `aria-valuemax` — максимальное значение.

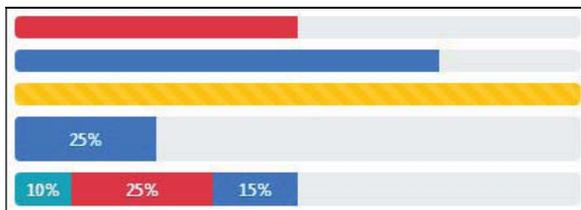


Рис. 3.40. Компонент `progress`: индикатор хода процесса

Пример указания значения 75%:

```
<div class="progress">
  <div class="progress-bar" role="progressbar"
    style="width: 75%" aria-valuenow="75"
    aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

Зададим значение 25% с помощью стилевого класса `w-25` и укажем высоту компонента, равной значению 5 px:

```
<div class="progress" style="height: 5px">
  <div class="progress-bar w-25" role="progressbar"
    aria-valuenow="25"
    aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

По умолчанию фон полосы отображается синим цветом. Для изменения цвета фона полосы следует воспользоваться стилевыми классами семейства `bg-*` (см. *разд. 1.1.6*). Укажем красный цвет (стилевой класс `bg-danger`) и значение 50%:

```
<div class="progress">
  <div class="progress-bar bg-danger" role="progressbar"
    style="width: 50%" aria-valuenow="50"
    aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

Если добавить стилевой класс `progress-bar-striped`, то вместо сплошного фона будут отображаться градиентные полосы:

```
<div class="progress">
  <div class="progress-bar progress-bar-striped bg-warning"
    role="progressbar" style="width: 100%" aria-valuenow="100"
    aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

Чтобы к градиентным полосам применялась CSS-анимация смещения фона, нужно дополнительно указать стилевой класс `progress-bar-animated`:

```
<div class="progress">
  <div class="progress-bar progress-bar-striped progress-bar-animated"
    role="progressbar" style="width: 100%" aria-valuenow="100"
    aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

По середине полосы можно указать надпись с текущим значением. Для этого вставляем значение внутрь элемента со стилевым классом `progress-bar`:

```
<div class="progress" style="height: 2rem">
  <div class="progress-bar" role="progressbar"
    style="width: 25%" aria-valuenow="25"
    aria-valuemin="0" aria-valuemax="100">25</div>
</div>
```

Однако при нулевом значении и небольших значениях надпись видна не будет, т. к. для полосы атрибут `overflow` по умолчанию имеет значение `hidden`. Поэтому над-

пись скрывается. Чтобы надпись всегда отображалась, нужно явным образом задать значение `visible` для атрибута `overflow`:

```
<div class="progress" style="height: 2rem">
  <div class="progress-bar" role="progressbar"
    style="width: 1%; overflow: visible" aria-valuenow="1"
    aria-valuemin="0" aria-valuemax="100">1%</div>
</div>
```

Родительский элемент является flex-контейнером, поэтому мы можем добавить в него сразу несколько полос с различными значениями, и они будут отображаться друг за другом:

```
<div class="progress" style="height: 1.5rem">
  <div class="progress-bar bg-info" role="progressbar"
    style="width: 10%" aria-valuenow="10"
    aria-valuemin="0" aria-valuemax="100">10%</div>
  <div class="progress-bar bg-danger" role="progressbar"
    style="width: 25%" aria-valuenow="25"
    aria-valuemin="0" aria-valuemax="100">25%</div>
  <div class="progress-bar" role="progressbar"
    style="width: 15%" aria-valuenow="15"
    aria-valuemin="0" aria-valuemax="100">15%</div>
</div>
```

3.7. Компонент *spinner*: индикатор состояния загрузки

Для индикации неопределенного по времени состояния загрузки библиотека Bootstrap предоставляет компонент `spinner`, который выглядит либо в виде бесконечно вращающейся границы круга с одним отсутствующим сегментом, либо в виде постоянно растущего круга без границы. Компонент не использует JavaScript, но для его отображения и сокрытия он, скорее всего, понадобится.

3.7.1. Класс *spinner-border*

Чтобы создать компонент `spinner`, который выглядит в виде бесконечно вращающейся границы круга с одним отсутствующим сегментом, следует создать контейнер и добавить к нему стилевой класс `spinner-border` и параметр `role` со значением `status`. Внутри контейнера вкладываем тег `` со стилевым классом `sr-only` (скрывает элемент) и текстом для программ чтения с экрана:

```
<div class="spinner-border" role="status">
  <span class="sr-only">Загрузка...</span>
</div>
```

По умолчанию ширина и высота равна значению `2 rem`. Для уменьшения размеров в два раза нужно добавить стилевой класс `spinner-border-sm`:

```
<div class="spinner-border spinner-border-sm" role="status">
  <span class="sr-only">Загрузка...</span>
</div>
```

С помощью атрибутов `width` и `height` можно указать любой другой размер:

```
<div class="spinner-border spinner-border-sm" role="status"
  style="width: 3rem; height: 3rem">
  <span class="sr-only">Загрузка...</span>
</div>
```

По умолчанию компонент отображается цветом текущего шрифта. Для изменения цвета границы следует использовать стилевые классы, предназначенные для изменения цвета текста (см. *разд. 1.2.4*):

```
<div class="spinner-border text-dark" role="status"></div>
<div class="spinner-border text-light" role="status"></div>
<div class="spinner-border text-success" role="status"></div>
<div class="spinner-border text-info" role="status"></div>
<div class="spinner-border text-warning" role="status"></div>
<div class="spinner-border text-danger" role="status"></div>
<div class="spinner-border text-primary" role="status"></div>
<div class="spinner-border text-secondary" role="status"></div>
```

3.7.2. Класс *spinner-grow*

Чтобы создать компонент `spinner`, который выглядит в виде постоянно растущего круга без границы, следует создать контейнер и добавить к нему стилевой класс `spinner-grow` и параметр `role` со значением `status`. Внутри контейнера вкладываем тег `` со стилевым классом `sr-only` (скрывает элемент) и текстом для программ чтения с экрана:

```
<div class="spinner-grow" role="status">
  <span class="sr-only">Загрузка...</span>
</div>
```

По умолчанию ширина и высота равна значению `2 rem`. Для уменьшения размеров в два раза нужно добавить стилевой класс `spinner-grow-sm`:

```
<div class="spinner-grow spinner-grow-sm" role="status">
  <span class="sr-only">Загрузка...</span>
</div>
```

С помощью атрибутов `width` и `height` можно указать любой другой размер:

```
<div class="spinner-grow spinner-grow-sm" role="status"
  style="width: 3rem; height: 3rem">
  <span class="sr-only">Загрузка...</span>
</div>
```

По умолчанию компонент отображается цветом текущего шрифта. Для изменения цвета круга следует использовать стилевые классы, предназначенные для изменения цвета текста (см. *разд. 1.2.4*):

```
<div class="spinner-grow text-dark" role="status"></div>
<div class="spinner-grow text-light" role="status"></div>
<div class="spinner-grow text-success" role="status"></div>
<div class="spinner-grow text-info" role="status"></div>
<div class="spinner-grow text-warning" role="status"></div>
```

```
<div class="spinner-grow text-danger" role="status"></div>
<div class="spinner-grow text-primary" role="status"></div>
<div class="spinner-grow text-secondary" role="status"></div>
```

3.7.3. Выравнивание компонента по центру или по правой стороне

Для компонента `spinner` атрибут `display` имеет значение `inline-block`. Чтобы произвести выравнивание компонента по центру или по правому краю, следует вложить его в родительский блочный контейнер и добавить для контейнера стилевые классы `text-center` или `text-right` соответственно:

```
<div class="text-center">
  <div class="spinner-border" role="status">
    <span class="sr-only">Загрузка...</span>
  </div>
</div>
<div class="text-right">
  <div class="spinner-border" role="status">
    <span class="sr-only">Загрузка...</span>
  </div>
</div>
```

Можно также преобразовать родительский контейнер в блочный флекс-контейнер и воспользоваться стилевыми классами `justify-content-center` и `justify-content-end`:

```
<div class="d-flex justify-content-center">
  <div class="spinner-border" role="status">
    <span class="sr-only">Загрузка...</span>
  </div>
</div>
<div class="d-flex justify-content-end">
  <div class="spinner-border" role="status">
    <span class="sr-only">Загрузка...</span>
  </div>
</div>
```

Пример выравнивания текста и компонента по разным сторонам контейнера:

```
<div class="d-flex align-items-center">
  <span>Загрузка...</span>
  <div class="spinner-border ml-auto" role="status"
    aria-hidden="true"></div>
</div>
```

3.7.4. Кнопка с компонентом *spinner*

Компонент `spinner` со стилевым классом `spinner-*-sm` можно добавить на кнопку после ее нажатия перед текстом надписи или вместо нее, для индикации выполнения операции. Чтобы пользователь не мог нажать кнопку при выполнении операции, следует сделать кнопку недоступной, добавив параметр `disabled`:

```
<button type="button" class="btn btn-primary" disabled>
  <span class="spinner-grow spinner-grow-sm" role="status"
    aria-hidden="true"></span>
```

```

    Загрузка...
</button>
<button type="button" class="btn btn-primary" disabled>
  <span class="spinner-border spinner-border-sm" role="status"></span>
  <span class="sr-only">Загрузка...</span>
</button>

```

3.8. Компонент *dropdown*: кнопка с выпадающим меню

Библиотека Bootstrap позволяет при нажатии кнопки или щелчке на ссылке отобразить выпадающее меню с отдельными пунктами или другими элементами — например, формой. Для отображения меню можно также установить фокус ввода на кнопку и нажать клавишу <Пробел>. Перемещение по пунктам меню возможно с помощью мыши или нажатия клавиш со стрелками. Чтобы закрыть меню без выбора, достаточно щелкнуть мышью вне меню или нажать клавишу <Esc>.

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.dropdown.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Dropdown.VERSION);
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов *jquery.min.js*, *popper.min.js* и *bootstrap.min.js*. Вместо двух последних файлов можно использовать файл *bootstrap.bundle.min.js*. В Bootstrap 5 можно не подключать файл *jquery.min.js*, если не используется интерфейс доступа через jQuery.

3.8.1. Класс *dropdown-toggle*: кнопка или ссылка с меню

Чтобы при нажатии обычной кнопки отображалось выпадающее меню (рис. 3.41), надо выполнить следующие действия:

1. Добавить к кнопке стилевой класс `dropdown-toggle` и уникальный идентификатор, а также параметры: `data-toggle` со значением `dropdown`, `aria-haspopup` со значением `true` и `aria-expanded` со значением `false`.
2. Вложить кнопку в родительский контейнер с относительным позиционированием (`position: relative`). Например, можно добавить стилевые классы `dropdown` (вниз), `dropup` (вверх), `dropleft` (влево) или `dropright` (вправо), задающие направление выпадения меню, или использовать их совместно с классом `btn-group`.
3. Добавить в родительский контейнер выпадающее меню, в параметре `aria-labelledby` которого указать идентификатор кнопки.

Пример:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" aria-haspopup="true"

```

```

        aria-expanded="false"> Отобразить меню
    </button>
    <div class="dropdown-menu" aria-labelledby="btn1">
        <a class="dropdown-item" href="#">Пункт 1</a>
        <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
</div>

```

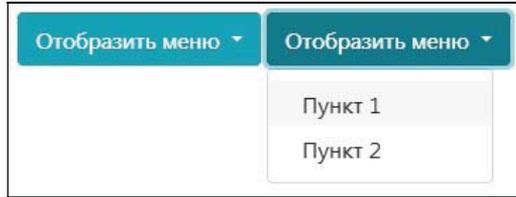


Рис. 3.41. Класс `dropdown-toggle`: кнопка с меню

Для уменьшения или увеличения размера кнопки с меню можно использовать классы `btn-sm` и `btn-lg` соответственно, а также классы `btn-group-sm` и `btn-group-lg`. Уменьшим размер кнопки:

```

<div class="btn-group dropdown">
    <button type="button" class="btn btn-sm btn-info dropdown-toggle"
        id="btn1" data-toggle="dropdown" aria-haspopup="true"
        aria-expanded="false"> Отобразить меню
    </button>
    <div class="dropdown-menu" aria-labelledby="btn1">
        <a class="dropdown-item" href="#">Пункт 1</a>
        <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
</div>

```

Вместо кнопки можно использовать гиперссылку:

```

<div class="btn-group dropdown">
    <a class="btn btn-info dropdown-toggle" href="#" role="button"
        id="btn1" data-toggle="dropdown" aria-haspopup="true"
        aria-expanded="false"> Отобразить меню
    </a>
    <div class="dropdown-menu" aria-labelledby="btn1">
        <a class="dropdown-item" href="#">Пункт 1</a>
        <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
</div>

```

3.8.2. Класс `dropdown-toggle-split`: кнопка с двумя зонами и меню

Можно также создать кнопку с двумя зонами и выпадающим меню, которое отображается при нажатии кнопки из второй зоны (рис. 3.42). На самом деле такая кнопка состоит из двух кнопок. Первая кнопка работает как обычная командная кнопка. Вторая кнопка примыкает вплотную к первой кнопке и вместо надписи со-

держит значок со стрелкой, указывающей направление выпадения меню. Для второй кнопки, помимо класса `dropdown-toggle`, следует добавить стиливой класс `dropdown-toggle-split`, который уменьшает внутренние отступы слева и справа от кнопки, а также убирает внешние отступы слева и справа. Вместо текста надписи нужно использовать тег `` с описанием для программ чтения с экрана и стиливым классом `sr-only`, скрывающим это описание:

```
<div class="btn-group dropdown">
  <button type="button" class="btn btn-info">Надпись</button>
  <button type="button" id="btn2" data-toggle="dropdown"
    class="btn btn-info dropdown-toggle dropdown-toggle-split"
    aria-haspopup="true" aria-expanded="false">
    <span class="sr-only">Отображение меню</span>
  </button>
  <div class="dropdown-menu" aria-labelledby="btn2">
    <a class="dropdown-item" href="#">Пункт 1</a>
    <a class="dropdown-item" href="#">Пункт 2</a>
  </div>
</div>
```

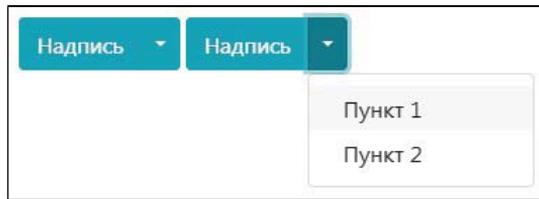


Рис. 3.42. Класс `dropdown-toggle-split`: кнопка с двумя зонами и меню

Для уменьшения или увеличения размера кнопки можно использовать классы `btn-sm` и `btn-lg` соответственно, а также классы `btn-group-sm` и `btn-group-lg`. Увеличим размер кнопки:

```
<div class="btn-group btn-group-lg dropdown">
  <button type="button" class="btn btn-info">Надпись</button>
  <button type="button" id="btn2" data-toggle="dropdown"
    class="btn btn-info dropdown-toggle dropdown-toggle-split"
    aria-haspopup="true" aria-expanded="false">
    <span class="sr-only">Отображение меню</span>
  </button>
  <div class="dropdown-menu" aria-labelledby="btn2">
    <a class="dropdown-item" href="#">Пункт 1</a>
    <a class="dropdown-item" href="#">Пункт 2</a>
  </div>
</div>
```

По умолчанию меню выравнивается по второй кнопке. Если нужно произвести выравнивание по первой кнопке, то следует добавить параметр `data-reference` со значением `parent`:

```
<div class="btn-group dropdown">
  <button type="button" class="btn btn-info">Надпись</button>
```

```

<button type="button" id="btn2" data-toggle="dropdown"
        class="btn btn-info dropdown-toggle dropdown-toggle-split"
        aria-haspopup="true" aria-expanded="false"
        data-reference="parent">
  <span class="sr-only">Отображение меню</span>
</button>
<div class="dropdown-menu" aria-labelledby="btn2">
  <a class="dropdown-item" href="#">Пункт 1</a>
  <a class="dropdown-item" href="#">Пункт 2</a>
</div>
</div>

```

3.8.3. Выпадающее меню с отдельными пунктами

Выпадающее меню (рис. 3.43) состоит из двух основных частей:

- ◆ родительского контейнера со стилевым классом `dropdown-menu` и параметром `aria-labelledby`, содержащим идентификатор кнопки;
- ◆ тегами `<a>` внутри родительского контейнера со стилевым классом `dropdown-item` и текстом пункта меню или другими элементами — например, кнопками.

Пример:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
          id="btn1" data-toggle="dropdown" aria-haspopup="true"
          aria-expanded="false"> Отобразить меню
</button>
<div class="dropdown-menu" aria-labelledby="btn1">
  <a class="dropdown-item" href="#">Пункт 1</a>
  <a class="dropdown-item" href="#">Пункт 2</a>
  <a class="dropdown-item" href="#">Пункт 3</a>
</div>
</div>

```

Для создания пунктов меню вместо ссылок можно также использовать кнопки:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
          id="btn1" data-toggle="dropdown" aria-haspopup="true"
          aria-expanded="false"> Отобразить меню
</button>
<div class="dropdown-menu" aria-labelledby="btn1">
  <button type="button" class="dropdown-item">Пункт 1</button>
  <button type="button" class="dropdown-item">Пункт 2</button>
  <button type="button" class="dropdown-item">Пункт 3</button>
</div>
</div>

```

Помимо обычных пунктов, меню может содержать заголовки (стилевой класс `dropdown-header`), разделительные линии (стилевой класс `dropdown-divider`), а также простой текст (стилевой класс `dropdown-item-text`):

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"

```

```

        id="btn1" data-toggle="dropdown" aria-haspopup="true"
        aria-expanded="false"> Отобразить меню
    </button>
    <div class="dropdown-menu" aria-labelledby="btn1">
        <h6 class="dropdown-header">Заголовок 1</h6>
        <a class="dropdown-item" href="#">Пункт 1</a>
        <a class="dropdown-item" href="#">Пункт 2</a>
        <div class="dropdown-divider"></div>
        <h6 class="dropdown-header">Заголовок 2</h6>
        <a class="dropdown-item" href="#">Пункт 3</a>
        <div class="dropdown-divider"></div>
        <span class="dropdown-item-text">Просто текст</span>
    </div>
</div>

```

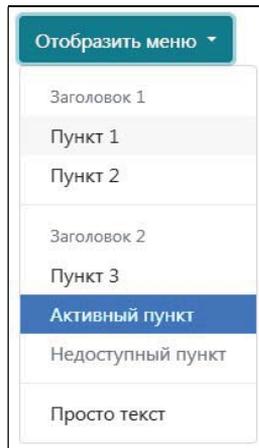


Рис. 3.43. Выпадающее меню

3.8.4. Активные и недоступные пункты меню

Для отображения меню можно добавить стилевой класс `show` вручную:

```
<div class="dropdown-menu show">...</div>
```

Чтобы имитировать активное (нажатое) состояние пункта меню, добавляем к пункту стилевой класс `active`:

```
<a class="dropdown-item active" href="#">Активный пункт</a>
```

Если нужно сделать пункт меню недоступным, то к ссылке добавляем стилевой класс `disabled`, а к кнопке — одноименный параметр:

```
<a class="dropdown-item disabled" href="#">Недоступный пункт</a>
```

Пример постоянно отображающегося меню с обычными, активными и недоступными пунктами:

```

<div class="dropdown">
    <div class="dropdown-menu show">
        <a class="dropdown-item" href="#">Обычный пункт</a>
        <a class="dropdown-item disabled" href="#">Недоступный пункт</a>
        <a class="dropdown-item active" href="#">Активный пункт</a>
    </div>

```

```

    <button type="button" class="dropdown-item" disabled>
      Недоступный пункт</button>
  </div>
</div>

```

3.8.5. Выпадающее меню с произвольным содержимым

Вместо пунктов в меню допускается вкладывать произвольное содержимое, но в этом случае нужно ограничить максимальную ширину меню и самостоятельно управлять отступами. Добавим в меню абзац с текстом:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" aria-haspopup="true"
    aria-expanded="false"> Отобразить меню
  </button>
  <div class="dropdown-menu p-2"
    style="max-width: 250px" aria-labelledby="btn1">
    <p class="mb-0">Просто абзац с текстом</p>
  </div>
</div>

```

Добавим в меню форму с полем для ввода пароля и кнопкой, а после нее вставим разделительную линию и обычный пункт меню:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" aria-haspopup="true"
    aria-expanded="false"> Отобразить меню
  </button>
  <div class="dropdown-menu" aria-labelledby="btn1">
    <form action="#" method="GET" class="px-3 py-1"
      style="max-width: 200px">
      <div class="mb-3">
        <label for="passwd">Пароль:</label>
        <input type="password" class="form-control"
          name="passwd" id="passwd">
      </div>
      <button type="submit"
        class="btn btn-primary btn-block">Войти</button>
    </form>
    <div class="dropdown-divider"></div>
    <a class="dropdown-item" href="#">Пункт меню</a>
  </div>
</div>

```

3.8.6. Направление выпадения меню

По умолчанию меню отображается ниже кнопки. Следующие классы позволяют явно указать предпочтительное направление выпадения меню:

◆ dropdown — ВНИЗ;

- ◆ `dropdown` — вверх;
- ◆ `dropleft` — влево;
- ◆ `dropright` — вправо.

В качестве примера отобразим меню справа от кнопки:

```
<div class="btn-group dropright">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" aria-haspopup="true"
    aria-expanded="false"> Отобразить меню
</button>
<div class="dropdown-menu" aria-labelledby="btn1">
  <a class="dropdown-item" href="#">Пункт 1</a>
  <a class="dropdown-item" href="#">Пункт 2</a>
</div>
</div>
```

Следует учитывать, что при недостаточном количестве места направление выпадения меню может поменяться на противоположное. Например, меню выпадает вниз. Но если в процессе прокрутки меню коснется нижней границы окна, то меню будет отображаться выше кнопки. Если это поведение по умолчанию нужно отключить, то следует добавить параметр `data-flip` со значением `false`:

```
<div class="container" style="margin: 500px">
  <div class="btn-group dropdown">
    <button type="button" class="btn btn-info dropdown-toggle"
      id="btn1" data-toggle="dropdown" aria-haspopup="true"
      aria-expanded="false" data-flip="false"> Отобразить меню
    </button>
    <div class="dropdown-menu" aria-labelledby="btn1">
      <a class="dropdown-item" href="#">Пункт 1</a>
      <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
  </div>
</div>
```

Уменьшите высоту окна таким образом, чтобы появилась вертикальная полоса прокрутки. Отобразите меню и попробуйте переместить полосу прокрутки так, чтобы меню коснулось границы окна, — меню просто уйдет за границы окна. Укажите в параметре `data-flip` значение `true` и повторите эксперимент. При касании границы окна меню из-под кнопки переместится наверх.

3.8.7. Положение меню

По умолчанию меню выравнивается по левому краю кнопки. Можно указать это явным образом, добавив стилевой класс `dropdown-menu-left`. Для выравнивания по правому краю следует добавить стилевой класс `dropdown-menu-right`:

```
<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" aria-haspopup="true"
    aria-expanded="false"> Отобразить выпадающее меню
```

```

</button>
<div class="dropdown-menu dropdown-menu-right" aria-labelledby="btn1">
  <a class="dropdown-item" href="#">Пункт 1</a>
  <a class="dropdown-item" href="#">Пункт 2</a>
</div>
</div>

```

В зависимости от ширины экрана можно воспользоваться следующими классами:

- ◆ `dropdown-menu-sm-left`, `dropdown-menu-md-left`, `dropdown-menu-lg-left`, `dropdown-menu-xl-left`, `dropdown-menu-xxl-left` (Bootstrap 5);
- ◆ `dropdown-menu-sm-right`, `dropdown-menu-md-right`, `dropdown-menu-lg-right`, `dropdown-menu-xl-right`, `dropdown-menu-xxl-right` (Bootstrap 5).

Обратите внимание: для кнопки в этом случае нужно добавить параметр `data-display` со значением `static`, а для контейнера — стилевой класс `btn-group`, иначе меню будет выравниваться относительно родительского контейнера, выходя за границы кнопки. До точки останова `md` выполним выравнивание меню по левому краю кнопки, а после нее — по правому краю:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" data-display="static"
    aria-haspopup="true" aria-expanded="false">
    Отобразить выпадающее меню
  </button>
  <div class="dropdown-menu dropdown-menu-md-right"
    aria-labelledby="btn1">
    <a class="dropdown-item" href="#">Пункт 1</a>
    <a class="dropdown-item" href="#">Пункт 2</a>
  </div>
</div>

```

Если необходимо сместить меню, то следует добавить параметр `data-offset` и в качестве значения указать одно число или два числа через запятую. Смысл этих чисел зависит от направления выпадения меню. Если меню выпадает вниз (значение по умолчанию), то одно число задает смещение меню вправо на указанное количество пикселей. Если указаны два числа через запятую, то первое число задает смещение вправо, а второе — смещение вниз. Сместим меню на 5 px вправо и 15 px вниз:

```

<div class="btn-group dropdown">
  <button type="button" class="btn btn-info dropdown-toggle"
    id="btn1" data-toggle="dropdown" aria-haspopup="true"
    aria-expanded="false" data-offset="5, 15">
    Отобразить меню
  </button>
  <div class="dropdown-menu" aria-labelledby="btn1">
    <a class="dropdown-item" href="#">Пункт 1</a>
    <a class="dropdown-item" href="#">Пункт 2</a>
  </div>
</div>

```

С помощью параметра `data-reference` можно указать, относительно какого элемента будет выравниваться меню. По умолчанию параметр имеет значение `toggle`. При использовании кнопки с двумя зонами это значение выравнивает меню по второй кнопке. Если указать значение `parent`, то меню будет выравниваться по первой кнопке:

```
<div class="btn-group dropdown">
  <button type="button" class="btn btn-info">Надпись</button>
  <button type="button" id="btn2" data-toggle="dropdown"
    class="btn btn-info dropdown-toggle dropdown-toggle-split"
    aria-haspopup="true" aria-expanded="false"
    data-reference="parent">
    <span class="sr-only">Отображение меню</span>
  </button>
  <div class="dropdown-menu" aria-labelledby="btn2">
    <a class="dropdown-item" href="#">Пункт 1</a>
    <a class="dropdown-item" href="#">Пункт 2</a>
  </div>
</div>
```

3.8.8. Управление компонентом из программы

Для управления компонентом из программы на jQuery надо вызвать метод `dropdown()` и передать ему одно из следующих значений:

- ◆ `'toggle'` — отображает или скрывает меню;
- ◆ `'show'` — отображает меню;
- ◆ `'hide'` — скрывает меню;
- ◆ `'update'` — обновляет позицию меню;
- ◆ `'dispose'` — удаляет данные, связанные с компонентом.

Пример:

```
$('#btn1').dropdown('toggle');
```

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Dropdown`:

```
<Объект> = new bootstrap.Dropdown(<Элемент>[, <Объект с опциями>])
```

Полный список опций, которые можно передать во втором параметре, вы найдете в документации. Пример создания объекта:

```
var btn1 = document.getElementById('btn1');
var btnInstance = new bootstrap.Dropdown(btn1, { flip: true });
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект и создания объекта при отсутствии ссылки:

```
var btn1 = document.getElementById('btn1');
var btnInstance = bootstrap.Dropdown.getInstance(btn1);
if ( !btnInstance ) {
  btnInstance = new bootstrap.Dropdown(btn1, { flip: true });
}
```

Класс `Dropdown` содержит следующие методы:

- ◆ `toggle()` — отображает или скрывает меню;
- ◆ `show()` — отображает меню;
- ◆ `hide()` — скрывает меню;
- ◆ `update()` — обновляет позицию меню;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример:

```
btnInstance.toggle();
```

3.8.9. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.dropdown` — событие перед отображением меню. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то меню отображено не будет:

```
var flagShow = false;
$('#div1').on('show.bs.dropdown', function(e) {
  if (!flagShow) e.preventDefault();
});
```

Пример для Bootstrap 5 без jQuery:

```
var flagShow = false;
var div1 = document.getElementById('div1');
div1.addEventListener('show.bs.dropdown', function(e) {
  if (!flagShow) e.preventDefault();
}, false);
```

- ◆ `shown.bs.dropdown` — событие после отображения меню;
- ◆ `hide.bs.dropdown` — событие перед сокрытием меню. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то меню скрыто не будет:

```
var flagHide = false;
$('#div1').on('hide.bs.dropdown', function(e) {
  if (!flagHide) e.preventDefault();
});
```

- ◆ `hidden.bs.dropdown` — событие после сокрытия меню.

Обработчики нужно назначать для родительского элемента, внутри которого расположено меню со стилевым классом `dropdown-menu`. Внутри обработчика через параметр доступен объект события, который содержит свойство `relatedTarget`. Значением этого свойства будет ссылка на кнопку с меню, которое является источником события.

Пример управления компонентом `dropdown` из программы и обработки событий приведен в листинге 3.6. Нажмите следующие клавиши:

- ◆ `<t>` — для отображения или сокрытия меню;

- ◆ `<s>` — для отображения меню;
- ◆ `<h>` — для сокрытия меню.

Листинг 3.6. Компонент `dropdown`: кнопка с выпадающим меню

```

<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Компонент dropdown: кнопка с выпадающим меню</title>
</head>
<body>
<div class="container">
  <div class="btn-group dropdown my-3" id="div1">
    <button type="button" class="btn btn-info dropdown-toggle"
      id="btn1" data-toggle="dropdown" aria-haspopup="true"
      aria-expanded="false"> Отобразить меню
    </button>
    <div class="dropdown-menu" aria-labelledby="btn1">
      <a class="dropdown-item" href="#">Пункт 1</a>
      <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
  </div>
  <p>Нажмите клавиши t (toggle), s (show) или h (hide).</p>
  <div class="btn-group dropdown my-5">
    <button type="button" class="btn btn-info">Надпись</button>
    <button type="button" id="btn2" data-toggle="dropdown"
      class="btn btn-info dropdown-toggle dropdown-toggle-split"
      aria-haspopup="true" aria-expanded="false">
      <span class="sr-only">Отображение меню</span>
    </button>
    <div class="dropdown-menu" aria-labelledby="btn2">
      <a class="dropdown-item" href="#">Пункт 1</a>
      <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
  </div>
  <p>Результат обработки событий см. в окне консоли.</p>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.bundle.min.js"></script>
<script>
$(function() {
  $('body').keydown( function(e) {
    if (e.which === 84) { // t
      $('#btn1').dropdown('toggle');
    }
  }
}

```

```
    else if (e.which === 83) {          // s
        $('#btn1').dropdown('show');
    }
    else if (e.which === 72) {          // h
        $('#btn1').dropdown('hide');
    }
});
var div1 = $('#div1');
div1.on('show.bs.dropdown', function(e) {
    console.log('show.bs.dropdown');
});
div1.on('shown.bs.dropdown', function(e) {
    console.log('shown.bs.dropdown');
});
div1.on('hide.bs.dropdown', function(e) {
    console.log('hide.bs.dropdown');
});
div1.on('hidden.bs.dropdown', function(e) {
    console.log('hidden.bs.dropdown');
});
$('.dropdown').on('show.bs.dropdown', function(e) {
    console.log(e.relatedTarget);
});
});
</script>
</body>
</html>
```

В этом примере мы воспользовались двумя новыми методами из библиотеки jQuery. Метод `keydown()` позволяет назначить обработчик события нажатия клавиши. В качестве параметра указывается ссылка на функцию обратного вызова. Внутри функции через параметр доступен объект, с помощью которого можно получить дополнительную информацию о событии. Например, чтобы получить код нажатой клавиши, нужно воспользоваться свойством `which`. Метод `on()` позволяет назначить обработчик для события, указанного в первом параметре. Во втором параметре передается ссылка на функцию обратного вызова.

Изменим содержимое тега `<script>` из листинга 3.6 так, чтобы программа работала в Bootstrap 5 без jQuery:

```
var btn1 = document.getElementById('btn1');
var btnInstance = new bootstrap.Dropdown(btn1, { flip: true });

document.body.addEventListener('keydown', function(e) {
    if (e.keyCode === 84) {          // t
        btnInstance.toggle();
    }
    else if (e.keyCode === 83) {     // s
        btnInstance.show();
    }
});
```

```
    }
    else if (e.keyCode === 72) {           // h
        btnInstance.hide();
    }
}, false);

var div1 = document.getElementById('div1');
div1.addEventListener('show.bs.dropdown', function(e) {
    console.log('show.bs.dropdown');
}, false);
div1.addEventListener('shown.bs.dropdown', function(e) {
    console.log('shown.bs.dropdown');
}, false);
div1.addEventListener('hide.bs.dropdown', function(e) {
    console.log('hide.bs.dropdown');
}, false);
div1.addEventListener('hidden.bs.dropdown', function(e) {
    console.log('hidden.bs.dropdown');
}, false);
document.querySelectorAll('.dropdown').forEach(
    function(elem) {
        elem.addEventListener('show.bs.dropdown', function(e) {
            console.log(e.relatedTarget);
        }, false);
    });
```



ГЛАВА 4

Готовые компоненты

Библиотека Bootstrap предоставляет большое количество нестандартных готовых компонентов: адаптивную панель навигации, карточки, панели с вкладками, всплывающие подсказки и уведомления, модальные диалоговые окна и др. Для правильной работы некоторых компонентов, помимо файла со стилями `bootstrap.min.css`, нужно дополнительно подключить файл `bootstrap.min.js` (или `bootstrap.bundle.min.js`). В Bootstrap 4 предварительно следует подключить файл `jquery.min.js`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery. Если же библиотека jQuery подключена, но нужно запретить ее использование в Bootstrap 5, то достаточно к тегу `<body>` добавить параметр `data-no-jquery`.

4.1. Компонент *jumbotron*: выделение важного содержимого

Компонент `jumbotron` в Bootstrap 4 предназначен для выделения важного содержимого, которое пользователь должен обязательно увидеть, — например, выгодного коммерческого предложения. Информационный блок выделяется цветом фона и большими внутренними отступами. Внутри блока можно вкладывать любые элементы — например, заголовки, абзацы, разделительные линии, кнопки и др. Для увеличения размера шрифта заголовка используются стилевые классы семейства `display-*` (см. *разд. 1.3.10*), для абзацев — стилевой класс `lead` (см. *разд. 1.3.11*), а для увеличения кнопок — стилевой класс `btn-lg` (см. *разд. 3.1.1*).

Для создания компонента следует обернуть важное содержимое в тег `<div>` и добавить к нему стилевой класс `jumbotron`:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <div class="jumbotron">
    <h1 class="display-4">Распродажа</h1>
    <p class="lead">Скидка 30% на ноутбуки</p>
    <hr class="mb-4">
    <p class="lead"><a class="btn btn-primary btn-lg" href="#"
      role="button">Узнать больше</a></p>
  </div>
</div>
```

Если нужно занять всю доступную ширину, убрать внутренние отступы слева и справа, а также запретить скругление углов, то следует дополнительно добавить стилевой класс `jumbotron-fluid`. Такой контейнер лучше использовать вместо базового, а контейнер со стилевым классом `container` и важным содержимым вложить в него:

```
<!-- Только в Bootstrap 4 -->
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-4">Распродажа</h1>
    <p class="lead">Скидка 30% на ноутбуки</p>
    <hr class="mb-4">
    <p class="lead"><a class="btn btn-primary btn-lg" href="#"
      role="button">Узнать больше</a></p>
  </div>
</div>
```

ПРИМЕЧАНИЕ

В Bootstrap 5 компонент `jumbotron` отсутствует. Для имитации этого компонента используйте стилевые классы: `bg-*` (см. *разд. 1.1.6*) — для указания цвета фона и `p-*` (см. *разд. 1.4.2*) — для добавления внутренних отступов.

4.2. Компонент *alert*: уведомления

Компонент `alert` используется для создания уведомлений, расположенных внутри контейнера с цветным фоном и рамкой со скругленными углами. С помощью специальной кнопки пользователь может закрыть уведомление (при условии, что программист добавил эту кнопку).

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.alert.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Alert.VERSION);
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента при наличии кнопки **Закрыть** требуется подключение файлов `jquery.min.js` и `bootstrap.min.js` (или `bootstrap.bundle.min.js`), а также добавление к кнопке параметра `data-dismiss` со значением `alert`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

4.2.1. Создание уведомления

Для создания компонента следует обернуть уведомление в тег `<div>` и добавить к нему стилевой класс `alert`:

```
<div class="alert" role="alert">.alert</div>
```

Дополнительно нужно указать стилевой класс, задающий цвет фона, текста, рамки и разделительной линии:

- ◆ `alert-dark` — темно-серый цвет фона;
- ◆ `alert-light` — светло-серый цвет фона;

- ◆ `alert-success` — цвет успешно выполненной операции (зеленый фон);
- ◆ `alert-info` — цвет информационного сообщения;
- ◆ `alert-warning` — цвет предупреждающего сообщения (желтый фон);
- ◆ `alert-danger` — цвет опасности (красный фон);
- ◆ `alert-primary` — синий цвет фона;
- ◆ `alert-secondary` — серый цвет фона.

Пример:

```
<div class="container">
  <div class="alert alert-dark" role="alert">.alert-dark</div>
  <div class="alert alert-light" role="alert">.alert-light</div>
  <div class="alert alert-success" role="alert">.alert-success</div>
  <div class="alert alert-info" role="alert">.alert-info</div>
  <div class="alert alert-warning" role="alert">.alert-warning</div>
  <div class="alert alert-danger" role="alert">.alert-danger</div>
  <div class="alert alert-primary" role="alert">.alert-primary</div>
  <div class="alert alert-secondary" role="alert">.alert-secondary</div>
</div>
```

Внутри контейнера может располагаться как обычный текст, так и произвольное содержимое — например, заголовки, абзацы, разделительные линии и др. Для заголовков нужно добавить стилевой класс `alert-heading`, а для стилизации ссылок, следует к ссылке добавить стилевой класс `alert-link`:

```
<div class="container">
  <div class="alert alert-success" role="alert">
    <h3 class="alert-heading">Заголовок</h3>
    <p>Текст до разделительной линии.</p>
    <hr>
    <p class="mb-0">Текст после разделительной линии.
      <a href="#" class="alert-link">Текст ссылки</a>
    </p>
  </div>
</div>
```

4.2.2. Закрытие уведомления

Чтобы пользователь мог закрыть уведомление с помощью кнопки **Закрыть** (рис. 4.1), следует выполнить следующие действия:

1. Добавить к родительскому контейнеру стилевой класс `alert-dismissible`, который создаст внутренний отступ справа для размещения кнопки. Если нужно, чтобы уведомление закрывалось с анимацией, то следует дополнительно добавить стилевые классы `fade` и `show`.
2. Вложить в родительский контейнер кнопку с символом «крестик» (HTML-эквивалент `×`) и добавить к кнопке стилевой класс `close`, а также параметр `data-dismiss` со значением `alert`. Кнопка отобразится в правом верхнем углу контейнера. По нажатию кнопки уведомление будет закрыто, а компонент полностью удален из структуры документа.

Пример:

```

<div class="container">
  <div class="alert alert-success alert-dismissible fade show"
    role="alert" id="myAlert">
    <h3 class="alert-heading">Заголовок</h3>
    <p>Текст до разделительной линии.</p>
    <hr>
    <p class="mb-0">Текст после разделительной линии.</p>
    <button type="button" class="close" data-dismiss="alert"
      aria-label="Закреть" title="Закреть">
      <span aria-hidden="true">&times;</span>
    </button>
  </div>
</div>

```

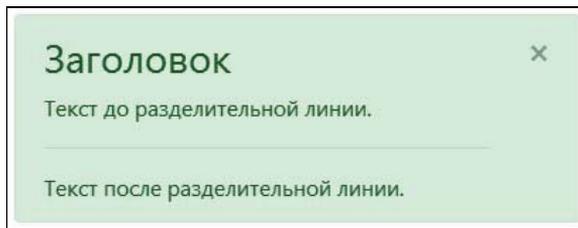


Рис. 4.1. Компонент alert: уведомление с кнопкой **Закреть**

Для закрытия уведомления из программы на jQuery, следует вызвать метод `alert()` и передать ему значение `'close'` (см. далее листинг 4.1). После закрытия компонент удаляется из документа.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Alert`:

```
<Объект> = new bootstrap.Alert(<Элемент>)
```

Пример создания объекта и сохранения его в элементе:

```
document.querySelectorAll('.alert').forEach( function(alert) {
  new bootstrap.Alert(alert);
});
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект и создания объекта при отсутствии ссылки:

```
var myAlert = document.getElementById('myAlert');
var myAlertInstance = bootstrap.Alert.getInstance(myAlert);
if ( !myAlertInstance ) {
  myAlertInstance = new bootstrap.Alert(myAlert);
}
```

Класс `Alert` содержит следующие методы:

- ◆ `close()` — закрывает уведомление. После закрытия компонент удаляется из документа;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример закрытия уведомления:

```
myAlertInstance.close();
```

4.2.3. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `close.bs.alert` — событие перед закрытием уведомления (сразу после нажатия кнопки **Закрыть** или вызова метода, предназначенного для закрытия уведомления). Если внутри обработчика через объект события вызвать метод `preventDefault()`, то уведомление закрыто не будет:

```
var flagClose = false;
$('#myAlert').on('close.bs.alert', function(e) {
  if (!flagClose) e.preventDefault();
});
```

Пример для Bootstrap 5 без jQuery:

```
var flagClose = false;
var myAlert = document.getElementById('myAlert');
myAlert.addEventListener('close.bs.alert', function(e) {
  if (!flagClose) e.preventDefault();
}, false);
```

- ◆ `closed.bs.alert` — событие после закрытия уведомления.

Пример управления компонентом `alert` из программы и обработки событий приведен в листинге 4.1.

Листинг 4.1. Компонент `alert`: уведомления

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Компонент alert: уведомления</title>
</head>
<body>
<div class="container mt-2">
  <div class="alert alert-success alert-dismissible fade show"
    role="alert" id="myAlert">
    <h3 class="alert-heading">Заголовок</h3>
    <p>Текст до разделительной линии.</p>
    <hr>
    <p class="mb-0">Текст после разделительной линии.</p>
    <button type="button" class="close" data-dismiss="alert">
```

```

        aria-label="Заккрыть" title="Заккрыть">
        <span aria-hidden="true">&times;</span>
    </button>
</div>
<button type="button" class="btn btn-primary" id="btnClose">
    Заккрыть</button>
<p>Результат обработки событий см. в окне консоли.</p>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
    $('#myAlert').on('close.bs.alert', function(e) {
        console.log('close.bs.alert');
    }).on('closed.bs.alert', function(e) {
        console.log('closed.bs.alert');
        $('#btnClose').prop('disabled', true);
    });
    $('#btnClose').click( function() {
        var elem = $('#myAlert');
        if (elem.length === 1) elem.alert('close');
        else console.log('Компонент не найден');
    });
});
</script>
</body>
</html>

```

Изменим содержимое тега `<script>` из листинга 4.1 так, чтобы программа работала в Bootstrap 5 без jQuery:

```

var myAlert = document.getElementById('myAlert');
var btnClose = document.getElementById('btnClose');

myAlert.addEventListener('close.bs.alert', function(e) {
    console.log('close.bs.alert');
}, false);
myAlert.addEventListener('closed.bs.alert', function(e) {
    console.log('closed.bs.alert');
    btnClose.disabled = true;
}, false);

btnClose.addEventListener('click', function() {
    var myAlert = document.getElementById('myAlert');
    if ( !myAlert ) return;
    var myAlertInstance = bootstrap.Alert.getInstance(myAlert);
    if ( !myAlertInstance ) {
        myAlertInstance = new bootstrap.Alert(myAlert);
    }
    myAlertInstance.close();
}, false);

```

4.3. Компонент *badge*: выделение фрагментов текста

Компонент `badge` используется для выделения фрагментов текста внутри строки путем изменения цвета фона и цвета текста. В Bootstrap 4 размер шрифта задан в процентах (75%), поэтому он всегда будет подстраиваться под размер шрифта родительского элемента. Для создания компонента следует обернуть фрагмент внутри строки в тег `` и добавить к нему стилевой класс `badge`:

```
<p>Абзац с <span class="badge">выделенным</span> фрагментом</p>
```

Дополнительно нужно указать стилевой класс, задающий цветовую схему. Классы в Bootstrap 4:

- ◆ `badge-dark` — темно-серый цвет фона и белый цвет надписи;
- ◆ `badge-light` — светло-серый цвет фона и черный цвет надписи;
- ◆ `badge-success` — зеленый цвет фона и белый цвет надписи;
- ◆ `badge-info` — информационный фрагмент с белым цветом надписи;
- ◆ `badge-warning` — желтый цвет фона и черный цвет надписи;
- ◆ `badge-danger` — красный цвет фона и белый цвет надписи;
- ◆ `badge-primary` — синий цвет фона и белый цвет надписи;
- ◆ `badge-secondary` — серый цвет фона и белый цвет надписи.

Пример указания цветовой схемы в Bootstrap 4:

```
<!-- Только в Bootstrap 4 -->
<p class="lead">
  <span class="badge badge-dark">.badge-dark</span>
  <span class="badge badge-light">.badge-light</span>
  <span class="badge badge-success">.badge-success</span>
  <span class="badge badge-info">.badge-info</span>
  <span class="badge badge-warning">.badge-warning</span>
  <span class="badge badge-danger">.badge-danger</span>
  <span class="badge badge-primary">.badge-primary</span>
  <span class="badge badge-secondary">.badge-secondary</span>
</p>
```

В Bootstrap 5 размер шрифта равен 0.75 em, а цвет текста белый. Для указания цветовой схемы следует использовать стилевые классы `bg-*` (см. *разд. 1.1.6*) и `text-*` (см. *разд. 1.2.4*):

```
<!-- Только в Bootstrap 5 -->
<p class="lead">
  <span class="badge bg-dark">.bg-dark</span>
  <span class="badge bg-light text-body">.bg-light</span>
  <span class="badge bg-success">.bg-success</span>
  <span class="badge bg-info">.bg-info</span>
  <span class="badge bg-warning text-body">.bg-warning</span>
</p>
```

```

<span class="badge bg-danger">.bg-danger</span>
<span class="badge bg-primary">.bg-primary</span>
<span class="badge bg-secondary">.bg-secondary</span>
</p>

```

В Bootstrap 4 стилевые классы `badge` и `badge-*` можно указывать для ссылок. В этом случае при наведении указателя мыши на ссылку, а также при получении фокуса, цвет фона ссылки станет темнее:

```

<!-- Только в Bootstrap 4 -->
<p class="lead">
  <a href="#" class="badge badge-dark">.badge-dark</a>
  <a href="#" class="badge badge-light">.badge-light</a>
  <a href="#" class="badge badge-success">.badge-success</a>
  <a href="#" class="badge badge-info">.badge-info</a>
  <a href="#" class="badge badge-warning">.badge-warning</a>
  <a href="#" class="badge badge-danger">.badge-danger</a>
  <a href="#" class="badge badge-primary">.badge-primary</a>
  <a href="#" class="badge badge-secondary">.badge-secondary</a>
</p>

```

По умолчанию выделенный фрагмент расположен внутри прямоугольной области с немного скругленными краями. Чтобы края сделать полукруглыми, нужно в Bootstrap 4 добавить стилевой класс `badge-pill`:

```

<!-- Только в Bootstrap 4 -->
<p class="lead">
  <span class="badge badge-pill badge-dark">.badge-dark</span>
  <span class="badge badge-pill badge-light">.badge-light</span>
  <span class="badge badge-pill badge-success">.badge-success</span>
  <span class="badge badge-pill badge-info">.badge-info</span>
  <span class="badge badge-pill badge-warning">.badge-warning</span>
  <span class="badge badge-pill badge-danger">.badge-danger</span>
  <span class="badge badge-pill badge-primary">.badge-primary</span>
  <span class="badge badge-pill badge-secondary">.badge-secondary</span>
</p>

```

В Bootstrap 5 вместо класса `badge-pill` следует использовать класс `rounded-pill`:

```

<!-- Только в Bootstrap 5 -->
<p class="lead">
  <span class="badge rounded-pill bg-dark">.bg-dark</span>
  <span class="badge rounded-pill bg-light text-body">.bg-light</span>
  <span class="badge rounded-pill bg-success">.bg-success</span>
  <span class="badge rounded-pill bg-info">.bg-info</span>
  <span
    class="badge rounded-pill bg-warning text-body">.bg-warning</span>
  <span class="badge rounded-pill bg-danger">.bg-danger</span>
  <span class="badge rounded-pill bg-primary">.bg-primary</span>
  <span class="badge rounded-pill bg-secondary">.bg-secondary</span>
</p>

```

Пример выделения фрагмента надписи на кнопке в Bootstrap 4 (рис. 4.2):

```
<!-- Только в Bootstrap 4 -->
<button type="button" class="btn btn-primary">
  Сообщений <span class="badge badge-pill badge-light">5</span>
</button>
```

Пример выделения фрагмента надписи на кнопке в Bootstrap 5:

```
<!-- Только в Bootstrap 5 -->
<button type="button" class="btn btn-primary">
  Сообщений
  <span class="badge rounded-pill bg-light text-body">5</span>
</button>
```

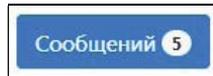


Рис. 4.2. Выделение фрагмента надписи на кнопке

4.4. Компонент *card*: карточки

Компонент `card` реализует контейнер с содержимым и рамкой в виде карточки, состоящей из нескольких разделов. Карточка может содержать изображение, заголовки, текст, список и любые другие элементы. Для создания карточки следует обернуть ее содержимое в тег `<div>` и добавить к нему стилевой класс `card`:

```
<div class="card">
  Текст внутри карточки
</div>
```

Элемент со стилевым классом `card` является flex-контейнером с вертикальным выравниванием, имеющим относительное позиционирование, внутрь которого мы можем вкладывать какое-либо содержимое или контейнеры с разделами. В нашем примере мы увидим текст без каких-либо отступов внутри рамки с границами серого цвета со скругленными углами.

4.4.1. Разделы карточки

Карточка может содержать следующие разделы, описываемые с помощью стилевых классов:

- ◆ `card-header` — заголовок карточки. Контейнер с заголовком имеет рамку серого цвета и светло-серый почти прозрачный цвет фона, а также внутренние отступы. Используйте теги заголовков, если нужно увеличить размер текста;
- ◆ `card-body` — основное содержимое карточки. Контейнер с телом карточки содержит внутренние отступы и наследует белый цвет фона от родительского контейнера. Рамку тело карточки не содержит. Контейнер будет максимально расширяться или сжиматься и иметь размер, соответствующий содержимому;
- ◆ `card-footer` — «подвал» карточки. Этот контейнер имеет рамку серого цвета и светло-серый почти прозрачный цвет фона, а также внутренние отступы. Раздел

размещается в самом низу карточки и обычно имеет менее важное значение, поэтому для изменения размера и цвета текста лучше вложить тег `<small>` и добавить для него стилевой класс `text-muted`.

Пример карточки со всеми разделами:

```
<div class="container">
  <div class="card">
    <div class="card-header">
      Заголовок карточки
    </div>
    <div class="card-body">
      Текст внутри карточки
    </div>
    <div class="card-footer">
      <small class="text-muted">&quot;Подвал&quot; карточки</small>
    </div>
  </div>
</div>
```

Ни один из разделов не является обязательным. Кроме того, разделов может не быть вовсе. Например, внутрь карточки мы можем вставить компонент `list-group`:

```
<div class="card">
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>
```

Если присутствует заголовок, то верхняя граница у компонента `list-group` убирается, — при условии, что компонент следует сразу после заголовка:

```
<div class="card">
  <div class="card-header">Заголовок карточки</div>
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>
```

Если нужно увеличить размер текста в разделе заголовка, то стилевой класс `card-header` следует указать для тега заголовка, а не для тега `<div>`:

```
<div class="card">
  <h5 class="card-header">Заголовок карточки</h5>
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>
```

Разделов с телом может быть несколько. Если нужно разделить их линией, то следует воспользоваться тегом `<hr>`:

```
<div class="card">
  <div class="card-body">Текст 1 внутри карточки</div>
  <hr class="my-0">
  <div class="card-body">Текст 2 внутри карточки</div>
</div>
```

Между двумя разделами с телом можно вставить другие элементы — например, компонент `list-group`:

```
<div class="card">
  <div class="card-body">Текст 1 внутри карточки</div>
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
  <div class="card-body">Текст 2 внутри карточки</div>
</div>
```

По умолчанию производится выравнивание содержимого карточки по левому краю. Для выравнивания по центру или по правому краю нужно добавить стилевые классы `text-center` или `text-right` соответственно. Пример выравнивания содержимого всех разделов по центру карточки:

```
<div class="card text-center">
  <div class="card-header">Заголовок карточки</div>
  <div class="card-body">Текст внутри карточки</div>
  <div class="card-footer">
    <small class="text-muted">&quot;Подвал&quot; карточки</small>
  </div>
</div>
```

4.4.2. Тело карточки

Все основное содержимое карточки нужно размещать внутри контейнера со стилевым классом `card-body`. Например, вставим в карточку текст цитаты (рис. 4.3):

```
<div class="card">
  <div class="card-header">Цитата</div>
  <div class="card-body">
    <blockquote class="blockquote mb-0">
      <p>Найди себе дело по душе и тебе не придется трудиться ни
        одного дня в жизни.</p>
      <footer class="blockquote-footer">Конфуций</footer>
    </blockquote>
  </div>
</div>
```



Рис. 4.3. Карточка с цитатой

Внутри тела карточки можно использовать следующие стилевые классы:

- ◆ `card-title` — используется для заголовков. Уменьшает внешний отступ снизу;
- ◆ `card-subtitle` — дополнительный заголовок. Убирает внешний отступ сверху (имеет отрицательное значение) и снизу;
- ◆ `card-text` — используется для абзацев с текстом. Убирает внешний отступ снизу при условии, что абзац является последним внутри контейнера;
- ◆ `card-link` — применяется для ссылок. Убирает подчеркивание при наведении указателя мыши на ссылку. Если две ссылки следуют друг за другом, то добавляет внешний отступ слева для второй ссылки.

Пример указания стилевых классов:

```
<div class="card">
  <div class="card-body">
    <h5 class="card-title">Заголовок</h5>
    <h6 class="card-subtitle text-muted mb-2">
      Дополнительный заголовок</h6>
    <p class="card-text">Текст внутри карточки</p>
    <a href="#" class="card-link">Текст ссылки 1</a>
    <a href="#" class="card-link">Текст ссылки 2</a>
  </div>
</div>
```

Создадим карточку с разделом заголовка, дополнительным заголовком внутри тела, абзацем и ссылкой, переделанной под кнопку:

```
<div class="card">
  <div class="card-header">Заголовок карточки</div>
  <div class="card-body">
    <h5 class="card-title">Дополнительный заголовок</h5>
    <p class="card-text">Текст внутри карточки</p>
    <a href="#" class="btn btn-primary">Текст на кнопке</a>
  </div>
</div>
```

В этом примере ссылкой является только область кнопки. Если хочется сделать ссылкой все содержимое карточки, то к тегу `<a>` нужно добавить стилевой класс `stretched-link`:

```
<div class="card">
  <div class="card-header">Заголовок карточки</div>
```

```

<div class="card-body">
  <h5 class="card-title">Дополнительный заголовок</h5>
  <p class="card-text">Текст внутри карточки</p>
  <a href="#" class="btn btn-primary stretched-link">
    Текст на кнопке</a>
</div>
</div>

```

4.4.3. Ширина и высота карточки

По умолчанию карточка занимает всю доступную ширину родительского элемента. Для указания ширины можно воспользоваться стилевыми классами `w-25`, `w-50` или `w-75` (см. *разд. 1.9.2*), а также CSS-атрибутами `width` или `max-width`:

```

<div class="card" style="max-width: 370px">
  <div class="card-header">Заголовок карточки</div>
  <div class="card-body">
    <h5 class="card-title">Дополнительный заголовок</h5>
    <p class="card-text">Текст внутри карточки</p>
    <a href="#" class="btn btn-primary">Текст на кнопке</a>
  </div>
</div>

```

Высота карточки зависит от ее содержимого, поэтому несколько карточек в одной строке могут иметь разную высоту. Чтобы высота была одинаковой, нужно явным образом добавить CSS-атрибут `height` с одинаковым значением:

```

<div class="clearfix">
  <div class="card float-left" style="width: 370px; height: 200px">
    <div class="card-body">
      <h5 class="card-title">Дополнительный заголовок</h5>
      <p class="card-text">Текст внутри карточки</p>
      <a href="#" class="btn btn-primary">Текст на кнопке</a>
    </div>
  </div>
  <div class="card float-left" style="width: 370px; height: 200px">
    <div class="card-body">Текст внутри карточки</div>
  </div>
</div>

```

Однако можно ошибиться с указанной высотой, и содержимое карточки будет вылезать за ее границы. Поэтому лучше поместить карточки внутри системы сеток библиотеки Bootstrap (см. *разд. 2.2* и *4.4.8*) и для каждой карточки указать стилевой класс `h-100` (см. *разд. 1.9.2*), задающий для высоты значение 100%:

```

<div class="container">
  <div class="row no-gutters g-0">
    <div class="col-sm mb-2">
      <div class="card h-100">
        <div class="card-body">
          <h5 class="card-title">Дополнительный заголовок</h5>
          <p class="card-text">Текст внутри карточки</p>
        </div>
      </div>
    </div>
  </div>

```

```

        <a href="#" class="btn btn-primary">Текст на кнопке</a>
    </div>
</div>
</div>
<div class="col-sm mb-2">
    <div class="card h-100">
        <div class="card-body">Текст внутри карточки</div>
    </div>
</div>
</div>
</div>

```

4.4.4. Изменение цветовой схемы карточки

Если нужно изменить цветовую схему карточки, то к родительскому контейнеру добавляем стилевые классы: семейства `bg-*` — для изменения цвета фона (см. *разд. 1.1.6*) и семейства `text-*` — для изменения цвета текста (см. *разд. 1.2.4*). Контейнеры с заголовком и «подвалом» карточки имеют полупрозрачный фон, поэтому изменятся цвет сразу всех разделов, причем цвет фона контейнера с телом будет светлее. Сделаем цвет фона карточки зеленым, а цвет текста белым:

```

<div class="card bg-success text-white">
    <div class="card-header">Заголовок карточки</div>
    <div class="card-body">Текст внутри карточки</div>
    <div class="card-footer text-white-50">
        &quot;Подвал&quot; карточки</div>
</div>

```

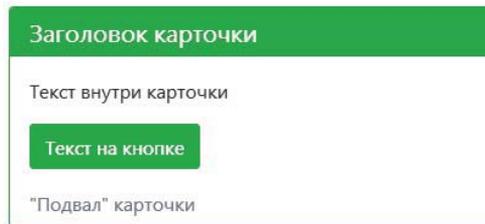


Рис. 4.4. Изменение цветовой схемы карточки

Чтобы изменить цвет линии рамки, следует воспользоваться стилевыми классами семейства `border-*` (см. *разд. 1.5.3*). Следует учитывать, что рамку имеет родительский контейнер, а также контейнеры заголовка и «подвала». Зададим зеленый цвет для линий рамки всей карточки, отключим рамку для контейнера с «подвалом» и уберем заливку фона. Кроме того, увеличим размер шрифта и сделаем цвет фона заголовка зеленым, а цвет текста белым (рис. 4.4):

```

<div class="card border-success">
    <h5 class="card-header bg-success text-white">
        Заголовок карточки</h5>
    <div class="card-body">
        <p class="card-text">Текст внутри карточки</p>

```

```

    <a href="#" class="btn btn-success">Текст на кнопке</a>
  </div>
  <div class="card-footer text-muted bg-transparent pt-0 border-0">
    &quot;Подвал&quot; карточки</div>
</div>

```

4.4.5. Изображение внутри карточки

Карточка может содержать изображение, которое чаще всего размещается выше или ниже всех разделов. Чтобы поместить изображение в верхней части карточки (рис. 4.5), нужно вставить изображение до разделов и добавить к нему стилевой класс `card-img-top` (устанавливает ширину 100% и добавляет скругление углов сверху):

```

<div class="card" style="max-width: 370px">
  
  <div class="card-body">
    <h5 class="card-title">Заголовок</h5>
    <p class="card-text">Текст внутри карточки</p>
    <a href="#" class="btn btn-primary">Текст на кнопке</a>
  </div>
</div>

```

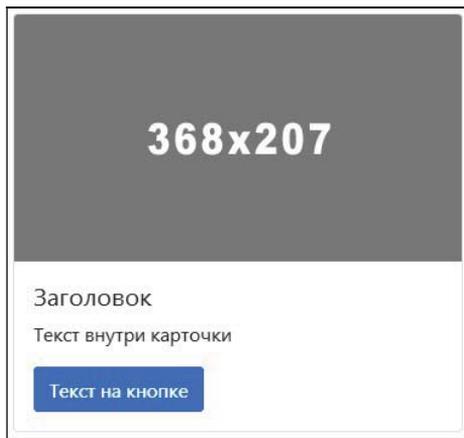


Рис. 4.5. Изображение в верхней части карточки

ПРИМЕЧАНИЕ

Предварительно создайте в каталоге `C:\book\img` изображение с названием `img_w368h207.jpg` и размерами 368×207 пикселей.

Чтобы поместить изображение в нижней части карточки (рис. 4.6), нужно вставить изображение после разделов и добавить к нему стилевой класс `card-img-bottom` (устанавливает ширину 100% и добавляет скругление углов снизу):

```

<div class="card" style="max-width: 370px">
  <div class="card-body">
    <h5 class="card-title">Заголовок</h5>

```

```

<p class="card-text">Текст внутри карточки</p>
<a href="#" class="btn btn-primary">Текст на кнопке</a>
</div>

</div>

```

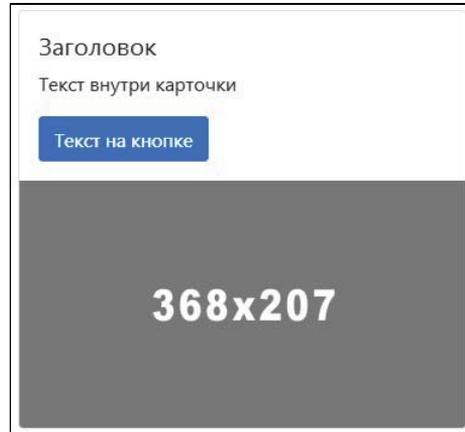


Рис. 4.6. Изображение в нижней части карточки

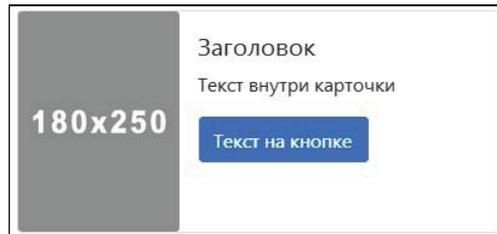


Рис. 4.7. Изображение в левой части карточки

Если нужно поместить изображение в левой или правой части карточки, следует воспользоваться системой сеток библиотеки Bootstrap (рис. 4.7). Для изображения в этом случае добавляем стиливой класс `card-img` (устанавливает ширину 100% и добавляет скругление для всех углов изображения):

```

<div class="card" style="max-width: 542px">
  <div class="row no-gutters g-0">
    <div class="col-4">
      
    </div>
    <div class="col-8">
      <div class="card-body">
        <h5 class="card-title">Заголовок</h5>
        <p class="card-text">Текст внутри карточки</p>
        <a href="#" class="btn btn-primary">Текст на кнопке</a>
      </div>
    </div>
  </div>

```

```

    </div>
  </div>
</div>

```

ПРИМЕЧАНИЕ

Предварительно создайте в каталоге *C:\book\img* изображение с названием *img_w180h250.jpg* и размерами 180×250 пикселей.

Существует также возможность использования изображения в качестве фона карточки. Содержимое карточки в этом случае будет накладываться на изображение, поэтому придется предусмотреть изменение цвета текста в зависимости от яркости фона. Для изображения в этом случае добавляем стилевой класс *card-img* (устанавливает ширину 100% и добавляет скругление для всех углов изображения), а для тела карточки — стилевой класс *card-img-overlay* (устанавливает абсолютное позиционирование и привязку ко всем сторонам, а также добавляет внутренние отступы):

```

<div class="card text-white" style="max-width: 546px">
  
  <div class="card-img-overlay">
    <h5 class="card-title">Заголовок</h5>
    <p class="card-text">Текст внутри карточки</p>
    <a href="#" class="btn btn-primary">Текст на кнопке</a>
  </div>
</div>

```

ПРИМЕЧАНИЕ

Предварительно создайте в каталоге *C:\book\img* темное изображение с названием *img_w544h306.jpg* и размерами 544×306 пикселей.

4.4.6. Группа из карточек без отступов

Самый простой способ выравнивания нескольких карточек заключается в добавлении их в родительский контейнер со стилевым классом *card-group* (описывает флек-контейнер с горизонтальным выравниванием). До точки останова *sm* (576 px), все карточки будут располагаться по вертикали друг под другом. Начиная с точки останова *sm*, карточки станут поровну делить горизонтальное пространство внутри строки без отступов между карточками. Если карточки не помещаются внутри одной строки, то они будут переноситься на следующую строку.

Высота всех карточек внутри строки одинаковая, причем расширяется раздел с телом, поэтому разделы с «подвалом» в разных карточках окажутся на одном уровне (рис. 4.8), — при условии, что их содержимое занимает одинаковое количество строк текста. Дополнительно для всей группы скругляются углы, а для карточек добавляются отступы снизу и удаляются границы между соседними карточками.

Пример создания группы из двух карточек приведен в листинге 4.2.

Листинг 4.2. Группа из карточек без отступов

```

<!doctype html>
<html lang="ru">

```

```

<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Группа из карточек без отступов</title>
</head>
<body>
<div class="container my-2">
  <div class="card-group">
    <div class="card">
      
      <div class="card-body">
        <p class="card-text">Текст внутри карточки</p>
      </div>
      <div class="card-footer text-muted">&quot;Подвал&quot;</div>
    </div>
    <div class="card">
      
      <div class="card-body">
        <h5 class="card-title">Заголовок</h5>
        <p class="card-text">Текст внутри карточки</p>
        <a href="#" class="btn btn-primary">Текст на кнопке</a>
      </div>
      <div class="card-footer text-muted">&quot;Подвал&quot;</div>
    </div>
  </div>
</div>
</body>
</html>

```

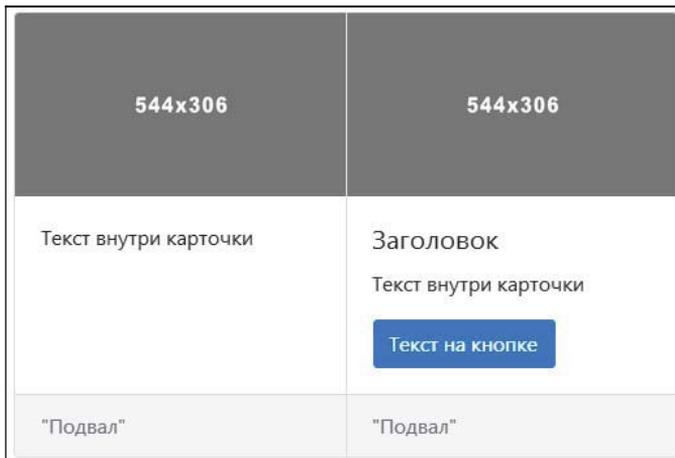


Рис. 4.8. Группа из карточек без отступов

4.4.7. Группа из карточек с отступами

При использовании стилевого класса `card-group`, карточки одинаковой высоты примыкают вплотную друг к другу внутри строки. Если нужно, чтобы между карточками было пустое расстояние, то в Bootstrap 4 достаточно заменить стилевой класс `card-group` классом `card-deck` (описывает flex-контейнер с горизонтальным выравниванием).

До точки останова `sm` (576 px), все карточки будут располагаться по вертикали друг под другом. Начиная с точки останова `sm`, карточки станут поровну делить горизонтальное пространство внутри строки. Если карточки не помещаются внутри одной строки, то они будут переноситься на следующую строку. Высота всех карточек внутри строки одинаковая, причем расширяется раздел с телом, поэтому разделы с «подвалом» в разных карточках окажутся на одном уровне (рис. 4.9), — при условии, что их содержимое занимает одинаковое количество строк текста.

Пример создания группы из двух карточек в Bootstrap 4 приведен в листинге 4.3.

Листинг 4.3. Группа из карточек с отступами (только в Bootstrap 4)

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Группа из карточек с отступами</title>
</head>
<body>
<div class="container my-3">
  <div class="card-deck">
    <div class="card">
      
      <div class="card-body">
        <p class="card-text">Текст внутри карточки</p>
      </div>
      <div class="card-footer text-muted">&quot;Подвал&quot;</div>
    </div>
    <div class="card">
      
      <div class="card-body">
        <h5 class="card-title">Заголовок</h5>
        <p class="card-text">Текст внутри карточки</p>
        <a href="#" class="btn btn-primary">Текст на кнопке</a>
      </div>
      <div class="card-footer text-muted">&quot;Подвал&quot;</div>
    </div>
  </div>
</div>
</body>
</html>
```

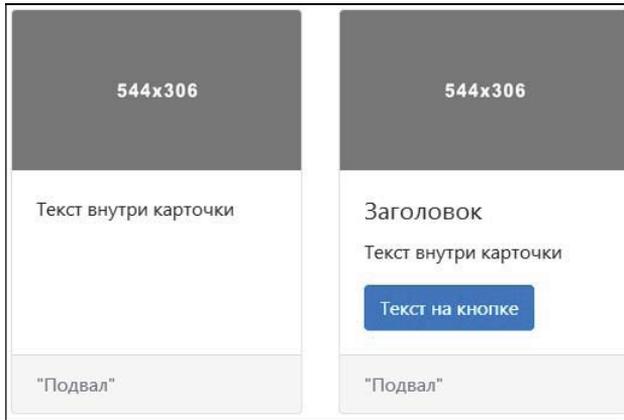


Рис. 4.9. Группа из карточек с отступами

4.4.8. Выравнивание карточек по сетке

Для выравнивания карточек удобно также использовать систему сеток библиотеки Bootstrap (см. *разд.* 2.2). Однако высота карточек внутри строки может быть разной. Чтобы этого избежать, следует для каждой карточки добавить стилевой класс `h-100`, задающий для высоты значение `100%`. По умолчанию между карточками добавляется пустое пространство. Чтобы этого избежать, нужно помимо стилевого класса `row` добавить класс `no-gutters` в Bootstrap 4 или `g-0` в Bootstrap 5.

Рассмотрим пример выравнивания двух карточек по сетке (листинг 4.4). До точки останова `md` (768 px), все карточки будут располагаться по вертикали друг под другом. Начиная с точки останова `md`, карточки станут поровну делить горизонтальное пространство внутри строки без пустого пространства между ними. Чтобы добавить пустое пространство между карточками, достаточно удалить стилевые классы `no-gutters` и `g-0`. Высоту всех карточек внутри строки сделаем одинаковой.

Листинг 4.4. Выравнивание карточек по сетке

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Выравнивание карточек по сетке</title>
</head>
<body>
<div class="container my-3">
  <div class="row no-gutters g-0">
    <!-- Удалите классы no-gutters и g-0, если нужны отступы -->
    <div class="col-md-6 mb-2">
      <div class="card h-100">
```

```

    
    <div class="card-body">
      <p class="card-text">Текст внутри карточки</p>
    </div>
    <div class="card-footer text-muted">&quot;Подвал&quot;</div>
  </div>
</div>
<div class="col-md-6 mb-2">
  <div class="card h-100">
    
    <div class="card-body">
      <h5 class="card-title">Заголовок</h5>
      <p class="card-text">Текст внутри карточки</p>
      <a href="#" class="btn btn-primary">Текст на кнопке</a>
    </div>
    <div class="card-footer text-muted">&quot;Подвал&quot;</div>
  </div>
</div>
</div>
</div>
</body>
</html>

```

Если количество карточек внутри группы может быть разным, то при выравнивании удобно указывать количество колонок в строке в зависимости от ширины экрана. Лишние колонки будут автоматически переноситься на новую строку, причем ширина всех карточек будет одинаковой, даже в том случае, если карточка окажется одна на строке, состоящей из нескольких колонок. Для указания количества колонок в строке предназначены стиливые классы семейства `row-cols-*` (см. *разд. 2.2.4*).

Рассмотрим пример выравнивания карточек по сетке (листинг 4.5). До точки останова `md` (768 px), все карточки станут располагаться по вертикали друг под другом, т. к. в строке будет только одна колонка (укажем стилевой класс `row-cols-1`). Начиная с точки останова `md`, карточки станут поровну делить горизонтальное пространство внутри строки. Между точками останова `md` и `lg` организуем две колонки в строке (укажем стилевой класс `row-cols-md-2`). Начиная с точки останова `lg`, количество колонок в строке увеличим до трех (укажем стилевой класс `row-cols-lg-3`). Высоту всех карточек внутри строки сделаем одинаковой (укажем стилевой класс `h-100`), а между карточками оставим пустое пространство. Чтобы удалить пустое пространство, достаточно добавить стилевой класс `no-gutters` в Bootstrap 4 или `g-0` в Bootstrap 5.

Листинг 4.5. Выравнивание карточек по сетке

```

<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"

```

```

    content="width=device-width, initial-scale=1, shrink-to-fit=no">
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
<title>Выравнивание карточек по сетке</title>
</head>
<body>
<div class="container my-3">
  <div class="row row-cols-1 row-cols-md-2 row-cols-lg-3">
    <div class="col mb-3">
      <div class="card h-100">
        
        <div class="card-body">
          <p class="card-text">Текст внутри карточки</p>
        </div>
        <div class="card-footer text-muted">&quot;Подвал&quot;</div>
      </div>
    </div>
    <div class="col mb-3">
      <div class="card h-100">
        
        <div class="card-body">
          <h5 class="card-title">Заголовок</h5>
          <p class="card-text">Текст внутри карточки</p>
        </div>
        <div class="card-footer text-muted">&quot;Подвал&quot;</div>
      </div>
    </div>
    <div class="col mb-3">
      <div class="card h-100">
        
        <div class="card-body">
          <h5 class="card-title">Заголовок</h5>
          <p class="card-text">Текст внутри карточки</p>
          <a href="#" class="btn btn-primary">Текст на кнопке</a>
        </div>
        <div class="card-footer text-muted">&quot;Подвал&quot;</div>
      </div>
    </div>
  </div>
</div>
</body>
</html>

```

4.4.9. Размещение карточек в трех колонках

Если есть множество карточек разной высоты, то в Bootstrap 4 их можно разместить внутри трех колонок (рис. 4.10), обернув карточки в родительский контейнер со стилевым классом `card-columns`:

```

<div class="card-columns">
  <div class="card">...</div>

```

```
...
<div class="card">...</div>
</div>
```

В этом примере используется технология многоколоночного текста (CSS-атрибуты `column-count` и `column-gap`), а не система сеток Bootstrap, поэтому карточки распределяются внутри трех колонок сверху вниз и слева направо. Между колонками и карточками снизу есть отступы. Обратите внимание: до точки останова sm (576 пх), карточки будут размещаться по вертикали друг под другом с отступом. Пример размещения пяти карточек в трех колонках в Bootstrap 4 приведен в листинге 4.6.

Листинг 4.6. Размещение карточек в трех колонках (только в Bootstrap 4)

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Размещение карточек в трех колонках</title>
</head>
<body>
<div class="container my-3">
  <div class="card-columns">
    <div class="card">
      
      <div class="card-body">
        <p class="card-text">Текст внутри карточки</p>
      </div>
    </div>
    <div class="card">
      
      <div class="card-body">
        <p class="card-text">Текст внутри карточки</p>
      </div>
    </div>
    <div class="card">
      
      <div class="card-body">
        <h5 class="card-title">Заголовок</h5>
        <p class="card-text">Текст внутри карточки</p>
        <a href="#" class="btn btn-primary">Текст на кнопке</a>
      </div>
    </div>
    <div class="card">
      
      <div class="card-body">
        <p class="card-text">Текст внутри карточки</p>
      </div>
    </div>
  </div>
</div>
```

```

</div>
<div class="card">
  
  <div class="card-body">
    <p class="card-text">Текст внутри карточки</p>
  </div>
</div>
</div>
</body>
</html>

```

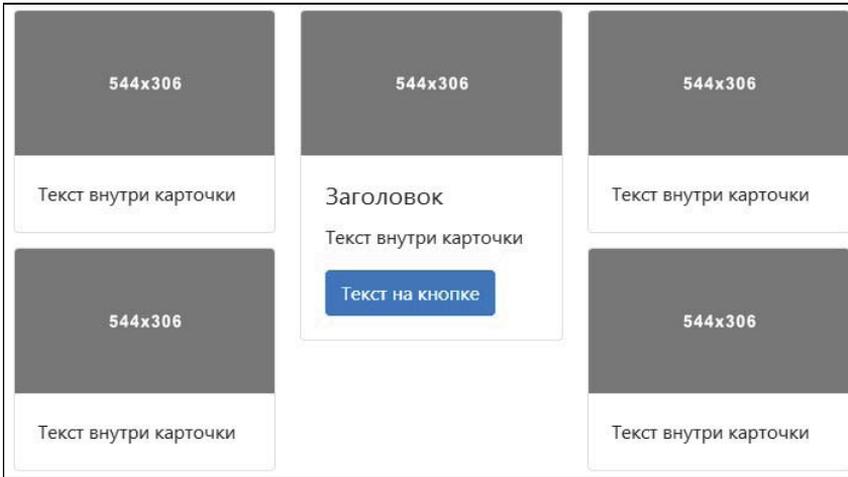


Рис. 4.10. Размещение карточек в трех колонках

ПРИМЕЧАНИЕ

Чтобы получить подобную функциональность в Bootstrap 5, следует воспользоваться библиотекой Masonry (<https://masonry.desandro.com/>). Пример можно найти на странице <https://v5.getbootstrap.com/docs/5.0/examples/masonry/>.

4.5. Компонент *collapse*: сворачивание и разворачивание области с содержимым

Компонент `collapse` позволяет переключать видимость какой-либо области с содержимым путем сворачивания или разворачивания контейнера, плавно изменяя его высоту. Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.collapse.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Collapse.VERSION);
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов *jquery.min.js* и *bootstrap.min.js* (или *bootstrap.bundle.min.js*), а также добавление к кнопке или ссылке параметра `data-toggle` со значением `collapse`. В Bootstrap 5 можно не подключать файл *jquery.min.js*, если не используется интерфейс доступа через jQuery.

4.5.1. Переключение состояния с помощью кнопки

Прежде всего, нужно вложить область с содержимым в родительский контейнер и добавить для него стилевой класс `collapse`, а также параметр `id` с уникальным идентификатором. По умолчанию содержимое скрыто. После нажатия кнопки или перехода по ссылке автоматически добавляется стилевой класс `collapsing`, который по завершении анимации удаляется. Затем, в зависимости от текущего состояния, добавляется или удаляется стилевой класс `show`, который является признаком отображения содержимого. Для изначального отображения области с содержимым следует явным образом добавить стилевой класс `show`.

Для кнопки, исполняющей роль переключателя состояния, нужно добавить следующие параметры:

- ◆ `data-toggle` — со значением `collapse`;
- ◆ `aria-expanded` — со значением `false`, если контейнер с содержимым свернут, или со значением `true`, если контейнер с содержимым отображается (в этом случае для контейнера с содержимым помимо стилевого класса `collapse` надо добавить класс `show`);
- ◆ `data-target` — должен содержать CSS-селектор, с помощью которого можно найти контейнер с содержимым;
- ◆ `aria-controls` — с идентификатором контейнера с содержимым.

Пример разворачивания и сворачивания области с содержимым по нажатию кнопки (содержимое изначально скрыто):

```
<div class="my-2">
  <button class="btn btn-primary" type="button"
    data-toggle="collapse" aria-expanded="false"
    data-target="#div1" aria-controls="div1">
    Свернуть или развернуть
  </button>
</div>
<div class="collapse" id="div1">
  <div class="card bg-success text-white">
    <div class="card-body">Текст внутри карточки 1</div>
  </div>
</div>
```

4.5.2. Переключение состояния с помощью ссылки

Вместо кнопки роль переключателя состояния может выполнять тег `<a>`, для которого действие по умолчанию (переход по ссылке) отменяется.

Для тега `<a>` в этом случае нужно добавить следующие параметры:

- ◆ `data-toggle` — со значением `collapse`;
- ◆ `aria-expanded` — со значением `false`, если контейнер с содержимым свернут, или со значением `true`, если контейнер с содержимым отображается (в этом случае для контейнера с содержимым помимо стилевого класса `collapse` надо добавить класс `show`);
- ◆ `href` — должен содержать CSS-селектор, с помощью которого можно найти контейнер с содержимым;
- ◆ `aria-controls` — идентификатор контейнера с содержимым.

Пример сворачивания и разворачивания области с содержимым по щелчку на ссылке (содержимое изначально отображается):

```
<div class="my-2">
  <a href="#div2" aria-controls="div2" role="button"
    data-toggle="collapse" aria-expanded="true">
    Свернуть или развернуть
  </a>
</div>
<div class="collapse show" id="div2">
  <div class="card bg-info text-white">
    <div class="card-body">Текст внутри карточки 2</div>
  </div>
</div>
```

4.5.3. Переключение состояния сразу нескольких областей

Параметр `data-target` кнопки и параметр `href` ссылки в качестве значения должны содержать CSS-селектор, с помощью которого можно найти контейнер с содержимым. В предыдущих примерах мы указывали поиск по идентификатору, поэтому переключалось состояние только одной области с содержимым. Если в качестве значения указать поиск по стилевому классу, то можно переключить состояние сразу нескольких областей.

Создадим три кнопки и две области. По нажатию первой кнопки переключим состояние первой области, по нажатию второй кнопки переключим состояние второй области, а по нажатию третьей кнопки переключим состояние сразу двух областей. Чтобы последнее стало возможным, добавим для контейнеров, помимо стилевого класса `collapse`, пользовательский класс `multi-collapse` и укажем его в параметре `data-target`:

```
<div class="my-2">
  <button class="btn btn-primary" type="button"
    data-toggle="collapse" aria-expanded="false"
    data-target="#div1" aria-controls="div1">
    Переключить область 1
  </button>
```

```
<button class="btn btn-primary" type="button"
  data-toggle="collapse" aria-expanded="false"
  data-target="#div2" aria-controls="div2">
  Переключить область 2
</button>
<button class="btn btn-primary" type="button"
  data-toggle="collapse" aria-expanded="false"
  data-target=".multi-collapse" aria-controls="div1 div2">
  Переключить обе области
</button>
</div>
<div class="collapse multi-collapse" id="div1">
  <div class="card bg-success text-white">
    <div class="card-body">Текст внутри карточки 1</div>
  </div>
</div>
<div class="collapse multi-collapse" id="div2">
  <div class="card bg-info text-white">
    <div class="card-body">Текст внутри карточки 2</div>
  </div>
</div>
```

4.5.4. Панель «Аккордеон»

На основе компонентов `collapse` и `card` можно создать панель «Аккордеон» — компонент с несколькими вкладками (рис. 4.11). Изначально отображается содержимое только одной вкладки, а у остальных доступны только заголовки. По щелчку мышью на заголовке вкладки она открывается, а остальные сворачиваются.



Рис. 4.11. Панель «Аккордеон»

Для создания панели «Аккордеон» выполняем следующие действия:

1. Создаем родительский контейнер для всех карточек и добавляем к нему стилевой класс `accordion`, а также уникальный идентификатор. Этот идентификатор указывается в параметре `data-parent` родительского контейнера для тела карточки (содержимого вкладки).

2. Внутри родительского контейнера вкладываем компоненты `card` с двумя разделами: заголовком (служит заголовком вкладки) и телом (содержимое вкладки), вложенным в дополнительный контейнер. Для заголовка задаем уникальный идентификатор. Этот идентификатор указывается в параметре `aria-labelledby` родительского контейнера для тела карточки.
3. Внутри заголовка карточки вкладываем тег `<a>` (или кнопку со стилизацией под ссылку) и добавляем следующие параметры:
 - `data-toggle` — со значением `collapse`;
 - `aria-expanded` — со значением `false`, если вкладка с содержимым свернута (в этом случае для ссылки нужно добавить стилевой класс `collapsed`), или со значением `true`, если вкладка с содержимым отображается (в этом случае к родительскому контейнеру для тела карточки помимо стилевого класса `collapse` нужно добавить класс `show`);
 - `href` — должен содержать CSS-селектор, с помощью которого можно найти родительский контейнер для тела карточки (для кнопки вместо параметра `href` добавляем параметр `data-target`);
 - `aria-controls` — указываем идентификатор родительского контейнера для тела карточки.
4. Тело карточки вкладываем в контейнер и для контейнера указываем следующие параметры:
 - добавляем стилевой класс `collapse` и параметр `id` с уникальным идентификатором. Если содержимое вкладки должно отображаться, то дополнительно добавляем стилевой класс `show`;
 - `aria-labelledby` — указываем идентификатор заголовка карточки;
 - `data-parent` — указываем CSS-селектор для поиска родительского контейнера для всех карточек (символ `#` и идентификатор контейнера).
5. Чтобы ссылкой служило все содержимое заголовка, нужно к заголовку карточки добавить стилевой класс `position-relative`, а к ссылке — класс `stretched-link` (см. *разд. 1.3.14*).

Пример создания панели «Аккордеон» с тремя вкладками приведен в листинге 4.7. Содержимое первой вкладки изначально отображено, а двух других — скрыто.

Листинг 4.7. Панель «Аккордеон»

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Панель Аккордеон</title>
```

```
</head>
<body>
<div class="container my-3">
  <div class="accordion" id="accordion1">

    <!-- Первая вкладка. Содержимое отображено -->
    <div class="card">
      <h5 class="card-header position-relative" id="header1">
        <a href="#tab1" class="stretched-link" aria-controls="tab1"
          role="button" data-toggle="collapse"
          aria-expanded="true">Вкладка 1</a>
      </h5>
      <div class="collapse show" id="tab1"
        aria-labelledby="header1"
        data-parent="#accordion1">
        <div class="card-body">Содержимое вкладки 1</div>
      </div>
    </div>

    <!-- Вторая вкладка. Содержимое свернуто -->
    <div class="card">
      <h5 class="card-header position-relative" id="header2">
        <a href="#tab2" aria-controls="tab2"
          class="stretched-link collapsed" role="button"
          data-toggle="collapse" aria-expanded="false">
          Вкладка 2</a>
      </h5>
      <div class="collapse" id="tab2"
        aria-labelledby="header2"
        data-parent="#accordion1">
        <div class="card-body">Содержимое вкладки 2</div>
      </div>
    </div>

    <!-- Третья вкладка. Содержимое свернуто -->
    <div class="card">
      <h5 class="card-header position-relative" id="header3">
        <a href="#tab3" aria-controls="tab3"
          class="stretched-link collapsed" role="button"
          data-toggle="collapse" aria-expanded="false">
          Вкладка 3</a>
      </h5>
      <div class="collapse" id="tab3"
        aria-labelledby="header3"
        data-parent="#accordion1">
        <div class="card-body">Содержимое вкладки 3</div>
      </div>
    </div>
  </div>
</div>
```

```

</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
</body>
</html>

```

4.5.5. Управление компонентом из программы

Для управления компонентом `collapse` из программы на jQuery, следует вызвать метод `collapse()` и передать ему одно из следующих значений:

- ◆ объект с опциями (см. документацию);
- ◆ `'toggle'` — отображает или скрывает содержимое;
- ◆ `'show'` — отображает содержимое;
- ◆ `'hide'` — сворачивает содержимое;
- ◆ `'dispose'` — удаляет данные, связанные с компонентом.

Пример переключения состояния:

```
$('#div1').collapse('toggle');
```

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Collapse`:

```
<Объект> = new bootstrap.Collapse(<Элемент>[, <Объект с опциями>])
```

При создании объекта производится переключение состояния компонента. Чтобы этого избежать, следует передать опцию `toggle` со значением `false`. Полный список опций, которые можно передать во втором параметре, вы найдете в документации.

Пример создания объекта:

```
var div1 = document.getElementById('div1');
var div1Instance = new bootstrap.Collapse(div1, { toggle: false });
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект и создания объекта при отсутствии ссылки:

```
var div1 = document.getElementById('div1');
var div1Instance = bootstrap.Collapse.getInstance(div1);
if ( !div1Instance ) {
    div1Instance = new bootstrap.Collapse(div1, { toggle: false });
}
```

Класс `Collapse` содержит следующие методы:

- ◆ `toggle()` — отображает или скрывает содержимое;
- ◆ `show()` — отображает содержимое;
- ◆ `hide()` — сворачивает содержимое;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример переключения состояния:

```
div1Instance.toggle();
```

ОБРАТИТЕ ВНИМАНИЕ!

Методы запускают переключение состояния асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

4.5.6. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.collapse` — событие перед отображением содержимого. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то содержимое отображено не будет:

```
var flagShow = false;
$('#div1').on('show.bs.collapse', function(e) {
  if (!flagShow) e.preventDefault();
});
```

Пример для Bootstrap 5 без jQuery:

```
var flagShow = false;
var div1 = document.getElementById('div1');
div1.addEventListener('show.bs.collapse', function(e) {
  if (!flagShow) e.preventDefault();
}, false);
```

- ◆ `shown.bs.collapse` — событие после отображения содержимого;
- ◆ `hide.bs.collapse` — событие перед сокрытием содержимого. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то содержимое скрыто не будет:

```
var flagHide = false;
$('#div1').on('hide.bs.collapse', function(e) {
  if (!flagHide) e.preventDefault();
});
```

- ◆ `hidden.bs.collapse` — событие после сокрытия содержимого.

Пример управления компонентом `collapse` из программы и обработки событий приведен в листинге 4.8. Нажмите следующие клавиши:

- ◆ `<t>` — для отображения или сокрытия содержимого;
- ◆ `<s>` — для отображения содержимого;
- ◆ `<h>` — для сокрытия содержимого.

Листинг 4.8. Компонент `collapse`

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

```

<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
<title>Компонент collapse</title>
</head>
<body>
<div class="container my-3">
  <p>Нажмите клавиши t (toggle), s (show) или h (hide).</p>
  <div class="my-2">
    <button class="btn btn-primary" type="button"
      data-toggle="collapse" aria-expanded="false"
      data-target="#div1" aria-controls="div1">
      Свернуть или развернуть
    </button>
  </div>
  <div class="collapse" id="div1">
    <div class="card bg-success text-white">
      <div class="card-body">Текст внутри карточки</div>
    </div>
  </div>
  <p>Результат обработки событий см. в окне консоли.</p>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
  $('body').keydown( function(e) {
    if (e.which === 84) {          // t
      $('#div1').collapse('toggle');
    }
    else if (e.which === 83) {    // s
      $('#div1').collapse('show');
    }
    else if (e.which === 72) {    // h
      $('#div1').collapse('hide');
    }
  });
  $('#div1').on('show.bs.collapse', function(e) {
    console.log('show.bs.collapse');
  }).on('shown.bs.collapse', function(e) {
    console.log('shown.bs.collapse');
  }).on('hide.bs.collapse', function(e) {
    console.log('hide.bs.collapse');
  }).on('hidden.bs.collapse', function(e) {
    console.log('hidden.bs.collapse');
  });
});
</script>
</body>
</html>

```

Изменим содержимое тега `<script>` из листинга 4.8 так, чтобы программа работала в Bootstrap 5 без jQuery:

```
var div1 = document.getElementById('div1');
var div1Instance = new bootstrap.Collapse(div1, { toggle: false });

document.body.addEventListener('keydown', function(e) {
  if (e.keyCode === 84) {          // t
    div1Instance.toggle();
  }
  else if (e.keyCode === 83) {    // s
    div1Instance.show();
  }
  else if (e.keyCode === 72) {    // h
    div1Instance.hide();
  }
}, false);

div1.addEventListener('show.bs.collapse', function(e) {
  console.log('show.bs.collapse');
}, false);
div1.addEventListener('shown.bs.collapse', function(e) {
  console.log('shown.bs.collapse');
}, false);
div1.addEventListener('hide.bs.collapse', function(e) {
  console.log('hide.bs.collapse');
}, false);
div1.addEventListener('hidden.bs.collapse', function(e) {
  console.log('hidden.bs.collapse');
}, false);
```

4.6. Компонент `nav`: контейнер со ссылками или ярлыками вкладок

Компонент `nav` описывает контейнер со ссылками или ярлыками вкладок. Ссылки внутри контейнера могут размещаться по горизонтали или по вертикали.

4.6.1. Горизонтальное размещение ссылок

Для создания контейнера с горизонтальным размещением ссылок нужно выполнить следующие действия:

1. Создать тег `` и добавить к нему стилевой класс `nav`. В результате список без маркеров становится флекс-контейнером с горизонтальным выравниванием и возможностью переноса ссылок на новую строку, если места недостаточно.
2. Вложить в тег `` теги `` со стилевым классом `nav-item`.

3. Вложить в тег `` тег `<a>` со стилевым классом `nav-link`. В результате ссылка становится блочным элементом с внутренними отступами. При наведении указателя мыши или при получении фокуса ввода ссылка подчеркнута не будет.
4. Если нужно сделать ссылку недоступной, то к ссылке следует добавить стилевой класс `disabled`.

Пример:

```
<ul class="nav">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" href="#" tabindex="-1"
      aria-disabled="true">Ссылка недоступна</a>
  </li>
</ul>
```

Вместо тега `` можно использовать тег `<nav>`, что позволяет писать меньше кода:

```
<nav class="nav">
  <a class="nav-link" href="#">Ссылка 1</a>
  <a class="nav-link" href="#">Ссылка 2</a>
  <a class="nav-link disabled" href="#" tabindex="-1"
    aria-disabled="true">Ссылка недоступна</a>
</nav>
```

4.6.2. Выравнивание ссылок внутри контейнера

По умолчанию производится выравнивание ссылок по левому краю контейнера. Чтобы произвести выравнивание по центру контейнера или по его правому краю, нужно добавить стилевые классы `justify-content-center` и `justify-content-end` соответственно:

```
<ul class="nav justify-content-center">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 2</a>
  </li>
</ul>
<nav class="nav justify-content-end">
  <a class="nav-link" href="#">Ссылка 1</a>
  <a class="nav-link" href="#">Ссылка 2</a>
</nav>
```

Вместо выравнивания по краям или центру можно равномерно распределить ссылки по всему контейнеру. Для этого нужно добавить стилевой класс `nav-fill`. Обратите внимание: в этом случае элементы могут иметь разную ширину. Если ширина

должна быть одинаковой, то вместо класса `nav-fill` следует использовать стилевой класс `nav-justified`:

```
<ul class="nav nav-fill">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 2 с длинным текстом</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 3</a>
  </li>
</ul>
<nav class="nav nav-justified">
  <a class="nav-link" href="#">Ссылка 1</a>
  <a class="nav-link" href="#">Ссылка 2 с длинным текстом</a>
  <a class="nav-link" href="#">Ссылка 3</a>
</nav>
```

На маленьких экранах стилевые классы `nav-fill` и `nav-justified` лучше не использовать. Если нужна функциональность класса `nav-fill`, но хочется, чтобы до точки останова `sm` ссылки размещались по вертикали, то для списка нужно указать стилевые классы `flex-column` и `flex-sm-row` (см. *разд. 2.1.1*), а для пунктов списка — стилевые классы `flex-sm-fill` и `text-sm-center`. Сравните два варианта, изменяя ширину экрана:

```
<ul class="nav flex-column flex-sm-row">
  <li class="nav-item flex-sm-fill text-sm-center">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item flex-sm-fill text-sm-center">
    <a class="nav-link" href="#">Ссылка 2 с длинным текстом</a>
  </li>
  <li class="nav-item flex-sm-fill text-sm-center">
    <a class="nav-link" href="#">Ссылка 3</a>
  </li>
</ul>
<nav class="nav nav-fill">
  <a class="nav-link" href="#">Ссылка 1</a>
  <a class="nav-link" href="#">Ссылка 2 с длинным текстом</a>
  <a class="nav-link" href="#">Ссылка 3</a>
</nav>
```

4.6.3. Вертикальное размещение ссылок

Для вертикального размещения ссылок достаточно к списку добавить стилевой класс `flex-column`:

```
<ul class="nav flex-column">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
```

```

</li>
<li class="nav-item">
  <a class="nav-link" href="#">Ссылка 2</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#">Ссылка 3</a>
</li>
</ul>
<nav class="nav flex-column">
  <a class="nav-link" href="#">Ссылка 1</a>
  <a class="nav-link" href="#">Ссылка 2</a>
  <a class="nav-link" href="#">Ссылка 3</a>
</nav>

```

Пример создания вложенной структуры:

```

<nav class="nav flex-column">
  <a class="nav-link" href="#">Раздел 1</a>
  <a class="nav-link" href="#">Раздел 2</a>
  <nav class="nav flex-column">
    <a class="nav-link ml-3" href="#">Раздел 2.1</a>
    <a class="nav-link ml-3" href="#">Раздел 2.2</a>
  </nav>
</nav>

```

Если хочется, чтобы до точки останова `sm` ссылки размещались по вертикали, а после — по горизонтали, то для списка нужно указать стилевые классы `flex-column` и `flex-sm-row` (см. *разд. 2.1.1*):

```

<ul class="nav flex-column flex-sm-row">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 3</a>
  </li>
</ul>

```

4.6.4. Активное состояние ссылки

Для визуального выделения ссылки с активным состоянием (рис. 4.12), следует добавить к ссылке стилевой класс `active`, а для списка — стилевой класс `nav-pills`, который изменяет цвет фона и текста ссылки:

```

<ul class="nav nav-pills">
  <li class="nav-item">
    <a class="nav-link active" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 2</a>
  </li>
</ul>

```

```

</li>
<li class="nav-item">
  <a class="nav-link" href="#">Ссылка 3</a>
</li>
</ul>
<nav class="nav nav-pills">
  <a class="nav-link" href="#">Ссылка 1</a>
  <a class="nav-link" href="#">Ссылка 2</a>
  <a class="nav-link active" href="#">Ссылка 3</a>
</nav>

```



Рис. 4.12. Активное состояние ссылки

Вместо добавления стилевых классов `active` к ссылке, можно для пункта списка указать стилевой класс `show`:

```

<ul class="nav nav-pills">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item show">
    <a class="nav-link" href="#">Ссылка 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 3</a>
  </li>
</ul>

```

4.6.5. Контейнер с ярлыками вкладок

С помощью компонента `nav` можно создать контейнер с ярлыками вкладок (рис. 4.13), который используется компонентом `tab`, реализующим панель с вкладками (см. *разд. 4.7*). Для этого к тегу ``, помимо стилевых классов `nav`, нужно добавить класс `nav-tabs`.

Ярлыки могут быть активными, неактивными и недоступными. Вокруг активного ярлыка отображается рамка со скругленными углами сверху, но без нижней границы. Чтобы сделать ярлык активным из программы, следует к ссылке добавить стилевой класс `active`. У неактивных ярлыков рамка появляется только при наведении указателя мыши. Если нужно сделать ярлык недоступным, то к ссылке добавляем стилевой класс `disabled`:

```

<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link active" href="#">Вкладка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Вкладка 2</a>
  </li>

```

```

</li>
<li class="nav-item">
  <a class="nav-link disabled" href="#" tabindex="-1"
    aria-disabled="true">Вкладка недоступна</a>
</li>
</ul>

```



Рис. 4.13. Контейнер с ярлыками вкладок

Вместо добавления стилевого класса `active` к ссылке, можно для пункта списка указать стилевой класс `show`:

```

<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link" href="#">Вкладка 1</a>
  </li>
  <li class="nav-item show">
    <a class="nav-link" href="#">Вкладка 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Вкладка 3</a>
  </li>
</ul>

```

4.6.6. Ссылка или ярлык вкладки с выпадающим меню

Чтобы в контейнер добавить ссылку с выпадающим меню, выполняем следующие действия:

1. Для списка добавляем стилевые классы `nav` и `nav-pills`. Вместо класса `nav-pills` можно использовать класс `nav-tabs`, который создает ярлыки вкладок.
2. Для пункта списка добавляем стилевые классы `nav-item` и `dropdown`.
3. К ссылке добавляем стилевые классы `nav-link` и `dropdown-toggle`, а также параметры: `data-toggle` — со значением `dropdown`, `aria-haspopup` — со значением `true`, `aria-expanded` — со значением `false` и `id` — с уникальным идентификатором.
4. После ссылки вставляем контейнер с пунктами меню (см. *разд. 3.8.3*). Добавляем к нему стилевой класс `dropdown-menu` и параметр `aria-labelledby` с идентификатором ссылки.

Пример использования ссылки с меню внутри контейнера с классом `nav-pills`:

```

<ul class="nav nav-pills">
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Ссылка 2</a>
  </li>

```

```
</li>
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" data-toggle="dropdown"
    href="#" role="button" aria-haspopup="true"
    aria-expanded="false" id="toggle1">Отобразить меню</a>
  <div class="dropdown-menu" aria-labelledby="toggle1">
    <a class="dropdown-item" href="#">Пункт 1</a>
    <a class="dropdown-item" href="#">Пункт 2</a>
  </div>
</li>
</ul>
```

Пример использования ссылки с меню внутри контейнера с классом `nav-tabs`:

```
<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link active" href="#">Вкладка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Вкладка 2</a>
  </li>
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" data-toggle="dropdown"
      href="#" role="button" aria-haspopup="true"
      aria-expanded="false" id="toggle1">Отобразить меню</a>
    <div class="dropdown-menu" aria-labelledby="toggle1">
      <a class="dropdown-item" href="#">Пункт 1</a>
      <a class="dropdown-item" href="#">Пункт 2</a>
    </div>
  </li>
</ul>
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы выпадающего меню требуется подключение файлов `jquery.min.js`, `popper.min.js` и `bootstrap.min.js`. Вместо двух последних файлов можно использовать файл `bootstrap.bundle.min.js`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

4.7. Компонент *tab*: панель с вкладками

Компонент `tab` позволяет создать панель с вкладками (рис. 4.14 и 4.15). При выборе ярлыка в области заголовка, реализуемого компонентом `nav` (см. *разд. 4.6.4* и *4.6.5*), становится видимым содержимое вкладки. Содержимое остальных вкладок при этом скрывается.

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.tab.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Tab.VERSION);
```

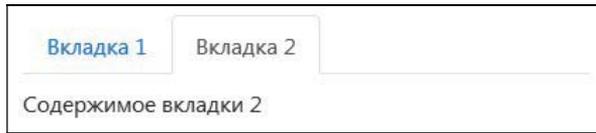


Рис. 4.14. Панель с вкладками (стилевой класс `nav-tabs`)

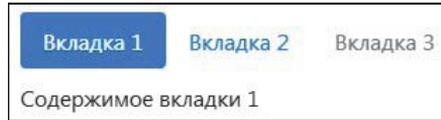


Рис. 4.15. Панель с вкладками (стилевой класс `nav-pills`)

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов `jquery.min.js` и `bootstrap.min.js` (или `bootstrap.bundle.min.js`), а также добавление к ссылкам параметра `data-toggle` со значением `tab`, `pill` или `list`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

4.7.1. Создание компонента

Компонент `tab` состоит из двух частей: контейнера с ярлыками вкладок и контейнера с содержимым вкладок. Контейнер с ярлыками вкладок реализуется с помощью компонента `nav` (см. *разд. 4.6.4* и *4.6.5*):

1. Для тега `` добавляем параметр `role` со значением `tablist`, а также стилевые классы `nav` и `nav-tabs`. Вместо класса `nav-tabs` можно использовать класс `nav-pills`.
2. Для ссылок указываем следующие параметры:
 - `data-toggle` — со значением `tab` (если используется класс `nav-tabs`), `pill` (если используется класс `nav-pills`) или `list` (если используется компонент `list-group`). Без этого параметра компонент не будет работать;
 - `role` — со значением `tab`;
 - `class` — со стилевым классом `nav-link`. Если ярлык должен быть активным, то добавляем стилевой класс `active`. Если нужно сделать ярлык недоступным, то добавляем стилевой класс `disabled`;
 - `id` — с уникальным идентификатором. Этот идентификатор нужно будет указать в параметре `aria-labelledby` контейнера с содержимым вкладки;
 - `href` — с CSS-селектором, с помощью которого можно найти контейнер с содержимым вкладки;
 - `aria-controls` — с идентификатором контейнера с содержимым вкладки;
 - `aria-selected` — со значением `true`, если вкладка активна, или значением `false` в противном случае.

Родительский контейнер с содержимым вкладок реализуется с помощью тега `<div>` со стилевым классом `tab-content`. Содержимое вкладок вставляется в тег `<div>` со следующими параметрами:

- ◆ `class` — со стилевым классом `tab-pane`. Если содержимое вкладки отображается, то нужно дополнительно добавить класс `active`. Чтобы переключение выполнялось с анимацией, а не сразу, следует добавить стилевой класс `fade`. Этот класс делает содержимое прозрачным, поэтому для активной вкладки нужно дополнительно добавить стилевой класс `show`;
- ◆ `id` — с уникальным идентификатором. Этот идентификатор указывается в параметрах `href` и `aria-controls` ссылки, служащей ярлыком вкладки;
- ◆ `role` — со значением `tabpanel`;
- ◆ `aria-labelledby` — с идентификатором ссылки, служащей ярлыком вкладки.

Пример компонента с заголовком, имеющим стилевой класс `nav-tabs`:

```
<ul class="nav nav-tabs mb-2" role="tablist">
  <li class="nav-item">
    <a class="nav-link active" id="label1" data-toggle="tab"
      href="#panel" aria-controls="panel" role="tab"
      aria-selected="true">Вкладка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" id="label2" data-toggle="tab"
      href="#pane2" aria-controls="pane2" role="tab"
      aria-selected="false">Вкладка 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" id="label3" data-toggle="tab"
      href="#pane3" aria-controls="pane3" role="tab"
      aria-selected="false" tabindex="-1"
      aria-disabled="true">Вкладка 3 (недоступна)</a>
  </li>
</ul>
<div class="tab-content">
  <div class="tab-pane fade show active" id="panel" role="tabpanel"
    aria-labelledby="label1">Содержимое вкладки 1</div>
  <div class="tab-pane fade" id="pane2" role="tabpanel"
    aria-labelledby="label2">Содержимое вкладки 2</div>
  <div class="tab-pane fade" id="pane3" role="tabpanel"
    aria-labelledby="label3">Содержимое вкладки 3</div>
</div>
```

Вместо тега `` для создания контейнера с ярлыками вкладок можно использовать тег `<nav>`:

```
<nav>
  <div class="nav nav-tabs mb-2" role="tablist">
    <a class="nav-item nav-link active" id="label1" data-toggle="tab"
      href="#panel" aria-controls="panel" role="tab"
      aria-selected="true">Вкладка 1</a>
```

```

    <a class="nav-item nav-link" id="label2" data-toggle="tab"
      href="#pane2" aria-controls="pane2" role="tab"
      aria-selected="false">Вкладка 2</a>
  </div>
</nav>
<div class="tab-content">
  <div class="tab-pane fade show active" id="panel" role="tabpanel"
    aria-labelledby="label1">Содержимое вкладки 1</div>
  <div class="tab-pane fade" id="pane2" role="tabpanel"
    aria-labelledby="label2">Содержимое вкладки 2</div>
</div>

```

Если для контейнера с заголовком указан стилевой класс `nav-pills`, то параметр `data-toggle` должен иметь значение `pill`:

```

<ul class="nav nav-pills mb-2" role="tablist">
  <li class="nav-item">
    <a class="nav-link active" id="label1" data-toggle="pill"
      href="#panel" aria-controls="panel" role="tab"
      aria-selected="true">Вкладка 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" id="label2" data-toggle="pill"
      href="#pane2" aria-controls="pane2" role="tab"
      aria-selected="false">Вкладка 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" id="label3" data-toggle="pill"
      href="#pane3" aria-controls="pane3" role="tab"
      aria-selected="false" tabindex="-1"
      aria-disabled="true">Вкладка 3 (недоступна)</a>
  </li>
</ul>
<div class="tab-content">
  <div class="tab-pane fade show active" id="panel" role="tabpanel"
    aria-labelledby="label1">Содержимое вкладки 1</div>
  <div class="tab-pane fade" id="pane2" role="tabpanel"
    aria-labelledby="label2">Содержимое вкладки 2</div>
  <div class="tab-pane fade" id="pane3" role="tabpanel"
    aria-labelledby="label3">Содержимое вкладки 3</div>
</div>

```

4.7.2. Вертикальное размещение ярлыков вкладок

Для вертикального размещения ярлыков вкладок добавляем стилевые классы `nav`, `nav-pills` и `flex-column`. Контейнеры с ярлыками вкладок и содержимым вкладок размещаем в отдельных колонках системы сеток библиотеки Bootstrap:

```

<div class="row">
  <div class="col-sm-3">
    <ul class="nav nav-pills flex-column" role="tablist"
      aria-orientation="vertical">

```

```
<li class="nav-item">
  <a class="nav-link active" id="label1" data-toggle="pill"
    href="#panel" aria-controls="panel" role="tab"
    aria-selected="true">Вкладка 1</a>
</li>
<li class="nav-item">
  <a class="nav-link" id="label2" data-toggle="pill"
    href="#pane2" aria-controls="pane2" role="tab"
    aria-selected="false">Вкладка 2</a>
</li>
</ul>
</div>
<div class="col-sm-9">
  <div class="tab-content mt-2">
    <div class="tab-pane fade show active" id="panel" role="tabpanel"
      aria-labelledby="label1">Содержимое вкладки 1</div>
    <div class="tab-pane fade" id="pane2" role="tabpanel"
      aria-labelledby="label2">Содержимое вкладки 2</div>
  </div>
</div>
</div>
```

4.7.3. Компонент *list-group* в качестве ярлыков вкладок

В качестве ярлыков вкладок может выступать компонент `list-group`. Для этого выполняем следующие действия:

1. Создаем родительский контейнер и добавляем к нему стилевой класс `list-group` и параметр `role` со значением `tablist`.
2. Внутри родительского контейнера вкладываем ссылки со следующими параметрами:
 - `data-toggle` — со значением `list`. Без этого параметра компонент не будет работать;
 - `role` — со значением `tab`;
 - `class` — со стилевыми классами `list-group-item` и `list-group-item-action`. Если ярлык должен быть активным, то добавляем стилевой класс `active`. Если нужно сделать ярлык недоступным, то добавляем стилевой класс `disabled`;
 - `id` — с уникальным идентификатором. Этот идентификатор нужно будет указать в параметре `aria-labelledby` контейнера с содержимым вкладки;
 - `href` — с CSS-селектором, с помощью которого можно найти контейнер с содержимым вкладки;
 - `aria-controls` — с идентификатором контейнера с содержимым вкладки;
 - `aria-selected` — со значением `true`, если вкладка активна, или значением `false` в противном случае.

Пример вертикального размещения ярлыков вкладок:

```
<div class="row">
  <div class="col-sm-4">
    <div class="list-group" role="tablist">
      <a class="list-group-item list-group-item-action active"
        id="label1" data-toggle="list" href="#panel"
        aria-controls="panel" role="tab"
        aria-selected="true">Вкладка 1</a>
      <a class="list-group-item list-group-item-action"
        id="label2" data-toggle="list" href="#pane2"
        aria-controls="pane2" role="tab"
        aria-selected="false">Вкладка 2</a>
    </div>
  </div>
  <div class="col-sm-8">
    <div class="tab-content mt-2">
      <div class="tab-pane fade show active" id="panel" role="tabpanel"
        aria-labelledby="label1">Содержимое вкладки 1</div>
      <div class="tab-pane fade" id="pane2" role="tabpanel"
        aria-labelledby="label2">Содержимое вкладки 2</div>
    </div>
  </div>
</div>
```

Для горизонтального размещения ярлыков вкладок нужно добавить стилевой класс `list-group-horizontal`:

```
<div class="list-group list-group-horizontal" role="tablist">
  <a class="list-group-item list-group-item-action active"
    id="label1" data-toggle="list" href="#panel"
    aria-controls="panel" role="tab"
    aria-selected="true">Вкладка 1</a>
  <a class="list-group-item list-group-item-action"
    id="label2" data-toggle="list" href="#pane2"
    aria-controls="pane2" role="tab"
    aria-selected="false">Вкладка 2</a>
</div>
<div class="tab-content mt-2">
  <div class="tab-pane fade show active" id="panel" role="tabpanel"
    aria-labelledby="label1">Содержимое вкладки 1</div>
  <div class="tab-pane fade" id="pane2" role="tabpanel"
    aria-labelledby="label2">Содержимое вкладки 2</div>
</div>
```

4.7.4. Карточки с панелью вкладок

Контейнер с ярлыками вкладок можно вложить в раздел заголовка карточки (рис. 4.16 и 4.17). В этом случае для списка нужно дополнительно указать стилевой класс `card-header-tabs`.

Контейнер с содержимым вкладок можно добавить в тело карточки:

```
<div class="card">
  <div class="card-header">
    <ul class="nav nav-tabs card-header-tabs" role="tablist">
      <li class="nav-item">
        <a class="nav-link active" id="label1" data-toggle="tab"
          href="#panel" aria-controls="panel" role="tab"
          aria-selected="true">Вкладка 1</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" id="label2" data-toggle="tab"
          href="#pane2" aria-controls="pane2" role="tab"
          aria-selected="false">Вкладка 2</a>
      </li>
    </ul>
  </div>
  <div class="card-body tab-content">
    <div class="tab-pane fade show active" id="panel" role="tabpanel"
      aria-labelledby="label1">Содержимое вкладки 1</div>
    <div class="tab-pane fade" id="pane2" role="tabpanel"
      aria-labelledby="label2">Содержимое вкладки 2</div>
  </div>
</div>
```

Если вместо стилевого класса `nav-tabs` используется класс `nav-pills`, то для списка нужно добавить стилевой класс `card-header-pills` (не забудьте в параметре `data-toggle` указать значение `pill`):

```
<div class="card">
  <div class="card-header">
    <ul class="nav nav-pills card-header-pills" role="tablist">
      <li class="nav-item">
        <a class="nav-link active" id="label1" data-toggle="pill"
          href="#panel" aria-controls="panel" role="tab"
          aria-selected="true">Вкладка 1</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" id="label2" data-toggle="pill"
          href="#pane2" aria-controls="pane2" role="tab"
          aria-selected="false">Вкладка 2</a>
      </li>
    </ul>
  </div>
  <div class="card-body tab-content">
    <div class="tab-pane fade show active" id="panel" role="tabpanel"
      aria-labelledby="label1">Содержимое вкладки 1</div>
    <div class="tab-pane fade" id="pane2" role="tabpanel"
      aria-labelledby="label2">Содержимое вкладки 2</div>
  </div>
</div>
```

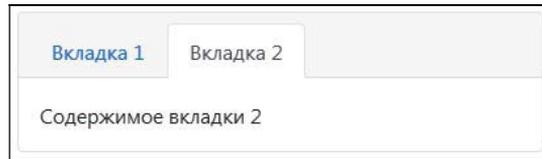


Рис. 4.16. Карточка с панелью вкладок (стилевой класс `nav-tabs`)

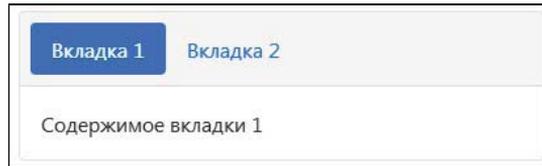


Рис. 4.17. Карточка с панелью вкладок (стилевой класс `nav-pills`)

4.7.5. Управление компонентом из программы

Для управления компонентом `tab` из программы на jQuery следует вызвать метод `tab()` и передать ему одно из следующих значений:

- ◆ `'show'` — отображает содержимое вкладки и делает ярлык вкладки активным. Содержимое других вкладок скрывается;
- ◆ `'dispose'` — удаляет данные, связанные с компонентом.

Пример отображения вкладки с указанным идентификатором:

```
$('#label2').tab('show');
```

Отообразим последнюю вкладку:

```
$('#labels li:last-child a').tab('show');
```

Пример отображения вкладки с индексом 2:

```
$('#labels li:nth-child(2) a').tab('show');
```

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Tab`:

```
<Объект> = new bootstrap.Tab(<Элемент>)
```

Пример создания объектов для ярлыков всех вкладок:

```
document.querySelectorAll('#labels a[data-toggle="tab"]').forEach(
  function(elem) {
    new bootstrap.Tab(elem);
  });
```

Получить ссылку на объект, сохраненный в ярлыке вкладки, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`.

Пример получения ссылки на объект:

```
var label1 = document.getElementById('label1');
var label1Instance = bootstrap.Tab.getInstance(label1);
```

Пример получения ссылки при доступе к первому ярлыку по селектору:

```
var label1 = document.querySelector('#labels li:first-child a');
var label1Instance = bootstrap.Tab.getInstance(label1);
```

Класс `Tab` содержит следующие методы:

- ◆ `show()` — отображает содержимое вкладки и делает ярлык вкладки активным. Содержимое других вкладок скрывается;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример отображения вкладки с указанным идентификатором:

```
var label1 = document.getElementById('label1');
var label1Instance = bootstrap.Tab.getInstance(label1);
if (label1Instance) label1Instance.show();
```

Отобразим последнюю вкладку:

```
var label2 = document.querySelector('#labels li:last-child a');
var label2Instance = bootstrap.Tab.getInstance(label2);
if (label2Instance) label2Instance.show();
```

Пример отображения вкладки с индексом 2:

```
var label2 = document.querySelector('#labels li:nth-child(2) a');
var label2Instance = bootstrap.Tab.getInstance(label2);
if (label2Instance) label2Instance.show();
```

ОБРАТИТЕ ВНИМАНИЕ!

Методы запускают переключение вкладок асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

4.7.6. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.tab` — событие перед отображением вкладки. Внутри обработчика через объект события доступны свойства `target` (ссылка на ярлык отображаемой вкладки) и `relatedTarget` (ссылка на ярлык предыдущей активной вкладки или значение `undefined`, если ранее не было активной вкладки). Если внутри обработчика через объект события вызвать метод `preventDefault()`, то вкладка отображена не будет:

```
var flagShow = false;
$('#label2').on('show.bs.tab', function(e) {
    if (!flagShow) e.preventDefault();
});
```

Пример для Bootstrap 5 без jQuery:

```
var flagShow = false;
var label2 = document.getElementById('label2');
```

```
label2.addEventListener('show.bs.tab', function(e) {
  if (!flagShow) e.preventDefault();
}, false);
```

- ◆ `shown.bs.tab` — событие после отображения вкладки. Внутри обработчика через объект события доступны свойства `target` (ссылка на ярлык активной вкладки) и `relatedTarget` (ссылка на ярлык предыдущей активной вкладки или значение `undefined`, если не было предыдущей активной вкладки);
- ◆ `hide.bs.tab` — событие перед сокрытием вкладки. Внутри обработчика через объект события доступны свойства `target` (ссылка на ярлык текущей активной вкладки) и `relatedTarget` (ссылка на ярлык следующей активной вкладки). Если внутри обработчика через объект события вызвать метод `preventDefault()`, то вкладка скрыта не будет:

```
var flagHide = false;
$('#label2').on('hide.bs.tab', function(e) {
  if (!flagHide) e.preventDefault();
});
```

- ◆ `hidden.bs.tab` — событие после сокрытия вкладки. Внутри обработчика через объект события доступны свойства `target` (ссылка на ярлык предыдущей активной вкладки) и `relatedTarget` (ссылка на ярлык новой активной вкладки).

Последовательность событий при попытке отображения вкладки выглядит следующим образом (при условии наличия предыдущей активной вкладки):

1. `hide.bs.tab` — на ярлыке текущей активной вкладки.
2. `show.bs.tab` — на ярлыке вкладки, которая должна быть отображена.
3. `hidden.bs.tab` — на ярлыке ранее активной вкладки (тот же ярлык, что и при событии `hide.bs.tab`).
4. `shown.bs.tab` — на ярлыке отображенной вкладки (тот же ярлык, что и при событии `show.bs.tab`).

Пример управления компонентом `tab` из программы и обработки событий приведен в листинге 4.9. По нажатию первой кнопки отобразим первую вкладку, а по нажатию второй кнопки — вторую вкладку. Обработаем события при смене вкладок и выведем сообщения в окно консоли.

Листинг 4.9. Компонент `tab`: панель с вкладками

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Компонент tab: панель с вкладками</title>
</head>
```

```
<body>
<div class="container my-3">
  <ul class="nav nav-tabs mb-2" role="tablist" id="labels">
    <li class="nav-item">
      <a class="nav-link active" id="label1" data-toggle="tab"
        href="#panel" aria-controls="panel" role="tab"
        aria-selected="true">Вкладка 1</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" id="label2" data-toggle="tab"
        href="#pane2" aria-controls="pane2" role="tab"
        aria-selected="false">Вкладка 2</a>
    </li>
  </ul>
  <div class="tab-content mb-3">
    <div class="tab-pane fade show active" id="panel" role="tabpanel"
      aria-labelledby="label1">Содержимое вкладки 1</div>
    <div class="tab-pane fade" id="pane2" role="tabpanel"
      aria-labelledby="label2">Содержимое вкладки 2</div>
  </div>
  <div>
    <button type="button" class="btn btn-primary" id="btn1">
      Отообразить вкладку 1</button>
    <button type="button" class="btn btn-primary" id="btn2">
      Отообразить вкладку 2</button>
  </div>
  <p>Результат обработки событий см. в окне консоли.</p>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
  $('#btn1').click( function(e) {
    $('#label1').tab('show');
    // $('#labels li:first-child a').tab('show');
    // $('#labels li:nth-child(1) a').tab('show');
  });
  $('#btn2').click( function(e) {
    $('#label2').tab('show');
    // $('#labels li:last-child a').tab('show');
    // $('#labels li:nth-child(2) a').tab('show');
  });
  $('#labels a[data-toggle="tab"]').on('show.bs.tab', function(e) {
    console.log('show.bs.tab');
  }).on('shown.bs.tab', function(e) {
    console.log('shown.bs.tab target: ' + e.target.id);
    if (e.relatedTarget)
      console.log('-- relatedTarget: ' + e.relatedTarget.id);
  });
});

```

```

    }).on('hide.bs.tab', function(e) {
        console.log('hide.bs.tab');
    }).on('hidden.bs.tab', function(e) {
        console.log('hidden.bs.tab');
    });
});
</script>
</body>
</html>

```

Изменим содержимое тега `<script>` из листинга 4.9 так, чтобы программа работала в Bootstrap 5 без jQuery:

```

document.querySelectorAll('#labels a[data-toggle="tab"]').forEach(
    function(elem) {
        new bootstrap.Tab(elem);

        elem.addEventListener('show.bs.tab', function(e) {
            console.log('show.bs.tab');
        }, false);
        elem.addEventListener('shown.bs.tab', function(e) {
            console.log('shown.bs.tab target: ' + e.target.id);
            if (e.relatedTarget)
                console.log('-- relatedTarget: ' + e.relatedTarget.id);
        }, false);
        elem.addEventListener('hide.bs.tab', function(e) {
            console.log('hide.bs.tab');
        }, false);
        elem.addEventListener('hidden.bs.tab', function(e) {
            console.log('hidden.bs.tab');
        }, false);
    });

var btn1 = document.getElementById('btn1');
btn1.addEventListener('click', function() {
    var label1 = document.getElementById('label1');
    //var label1 = document.querySelector('#labels li:first-child a');
    //var label1 = document.querySelector('#labels li:nth-child(1) a');
    var label1Instance = bootstrap.Tab.getInstance(label1);
    if (label1Instance) label1Instance.show();
}, false);

var btn2 = document.getElementById('btn2');
btn2.addEventListener('click', function() {
    var label2 = document.getElementById('label2');
    //var label2 = document.querySelector('#labels li:last-child a');
    //var label2 = document.querySelector('#labels li:nth-child(2) a');
    var label2Instance = bootstrap.Tab.getInstance(label2);
    if (label2Instance) label2Instance.show();
}, false);

```

4.8. Компонент *list-group*: список с пунктами, ссылками или кнопками

Компонент `list-group` применяется для создания списка с пунктами (содержащими текст или другие элементы), со ссылками или кнопками. Кроме того, он может выступать в качестве ярлыков вкладок при использовании компонента `tab` (см. *разд. 4.7.3*).

4.8.1. Список с пунктами, содержащими текст

При использовании стилевого класса `list-group`, пункты списка без маркеров выводятся друг под другом внутри рамок (рис. 4.18). Для каждого пункта такого списка нужно добавить стилевой класс `list-group-item`:

```
<div class="container">
  <ul class="list-group">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>
```



Рис. 4.18. Компонент `list-group`: список с пунктами

Контейнер для пунктов списка с классом `list-group` является флекс-контейнером с вертикальным выравниванием элементов. Каждый пункт списка с классом `list-group-item` имеет относительное позиционирование и занимает всю доступную ширину родительского контейнера при вертикальном выравнивании. Пункт списка содержит рамку, внутренние отступы и белый цвет фона.



Рис. 4.19. Использование стилевого класса `list-group-flush`

Если дополнительно добавить стилевой класс `list-group-flush`, то пункты списка будут разделены линиями, а не помещены внутри рамок (рис. 4.19):

```
<div class="container">
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Первый пункт</li>
```

```

    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>

```

4.8.2. Размещение пунктов по горизонтали

По умолчанию пункты размещаются друг под другом по вертикали. Если нужно выполнить выстраивание по горизонтали (рис. 4.20), то следует добавить стилевой класс `list-group-horizontal`:

```

<div class="container">
  <ul class="list-group list-group-horizontal">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>

```

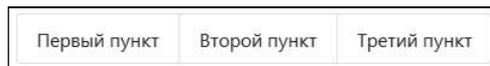


Рис. 4.20. Размещение пунктов по горизонтали

Пункты списка в приведенном примере имеют ширину, зависящую от содержимого элемента. Чтобы пункт растягивался, нужно к тегу `` добавить стилевой класс `flex-fill` или классы для адаптивной верстки: `flex-sm-fill`, `flex-md-fill`, `flex-lg-fill`, `flex-xl-fill`. В следующем примере все пункты имеют одинаковые размеры и делят весь контейнер поровну:

```

<div class="container">
  <ul class="list-group list-group-horizontal">
    <li class="list-group-item flex-fill">Первый пункт</li>
    <li class="list-group-item flex-fill">Второй пункт</li>
    <li class="list-group-item flex-fill">Третий пункт</li>
  </ul>
</div>

```

Если нужно, чтобы до определенной точки остановка список был вертикальным, а после нее — горизонтальным, то следует воспользоваться классами для адаптивной верстки: `list-group-horizontal-sm`, `list-group-horizontal-md`, `list-group-horizontal-lg`, `list-group-horizontal-xl`, `list-group-horizontal-xxl` (Bootstrap 5). Начиная с точки остановка `md`, сделаем список горизонтальным:

```

<div class="container">
  <ul class="list-group list-group-horizontal-md">
    <li class="list-group-item">Первый пункт</li>
    <li class="list-group-item">Второй пункт</li>
    <li class="list-group-item">Третий пункт</li>
  </ul>
</div>

```

4.8.3. Список со ссылками или кнопками

Список может состоять из ссылок (рис. 4.21) или кнопок. В этом случае, помимо стилевого класса `list-group-item`, нужно добавить класс `list-group-item-action`. При наведении указателя мыши на пункт списка, а также при получении фокуса ввода цвет фона пункта станет темнее. Если добавить стилевой класс `active`, то пункт будет выделен как активный (синий фон с белым текстом). Если добавить стилевой класс `disabled`, то пункт будет выделен как неактивный, а ссылка станет недоступной. Пример:

```
<div class="list-group">
  <a href="#" class="list-group-item list-group-item-action active">
    Ссылка 1 (активное состояние)</a>
  <a href="#" class="list-group-item list-group-item-action">
    Ссылка 2</a>
  <a href="#" class="list-group-item list-group-item-action">
    Ссылка 3</a>
  <a href="#"
    class="list-group-item list-group-item-action disabled">
    Недоступная ссылка</a>
</div>
```

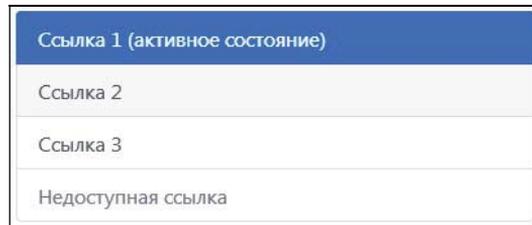


Рис. 4.21. Список со ссылками

По умолчанию пункты размещаются друг под другом по вертикали. Если нужно выполнить выстраивание по горизонтали, то следует добавить стилевой класс `list-group-horizontal` или классы для адаптивной верстки: `list-group-horizontal-sm`, `list-group-horizontal-md`, `list-group-horizontal-lg`, `list-group-horizontal-xl`, `list-group-horizontal-xxl` (Bootstrap 5). Пункты списка в этом случае имеют ширину 100%:

```
<div class="list-group list-group-horizontal-md">
  <a href="#" class="list-group-item list-group-item-action">
    Ссылка 1</a>
  <a href="#" class="list-group-item list-group-item-action">
    Ссылка 2</a>
  <a href="#" class="list-group-item list-group-item-action">
    Ссылка 3</a>
</div>
```

В список можно добавить не только ссылки, но и кнопки (без стилевого класса `btn`). Чтобы сделать кнопку недоступной, следует добавить параметр `disabled`:

```
<div class="list-group">
  <button type="button"
    class="list-group-item list-group-item-action active">
```

```

    Кнопка 1 (активное состояние)</button>
<button type="button" class="list-group-item list-group-item-action">
    Кнопка 2</button>
<button type="button" class="list-group-item list-group-item-action">
    Кнопка 3</button>
<button type="button" class="list-group-item list-group-item-action"
    disabled>Недоступная кнопка</button>
</div>

```

4.8.4. Изменение цветовой схемы

Задать цвет фона пункта списка позволяют следующие стилевые классы:

- ◆ `list-group-item-dark` — темно-серый цвет;
- ◆ `list-group-item-light` — светло-серый цвет;
- ◆ `list-group-item-success` — зеленый цвет;
- ◆ `list-group-item-info` — цвет информационного сообщения;
- ◆ `list-group-item-warning` — желтый цвет;
- ◆ `list-group-item-danger` — красный цвет;
- ◆ `list-group-item-primary` — синий цвет;
- ◆ `list-group-item-secondary` — серый цвет.

Пример:

```

<div class="container">
  <ul class="list-group">
    <li class="list-group-item">Обычный пункт</li>

    <li class="list-group-item list-group-item-dark">
      .list-group-item-dark</li>
    <li class="list-group-item list-group-item-light">
      .list-group-item-light</li>
    <li class="list-group-item list-group-item-success">
      .list-group-item-success</li>
    <li class="list-group-item list-group-item-info">
      .list-group-item-info</li>
    <li class="list-group-item list-group-item-warning">
      .list-group-item-warning</li>
    <li class="list-group-item list-group-item-danger">
      .list-group-item-danger</li>
    <li class="list-group-item list-group-item-primary">
      .list-group-item-primary</li>
    <li class="list-group-item list-group-item-secondary">
      .list-group-item-secondary</li>
  </ul>
</div>

```

Если используется стилевой класс `list-group-item-action`, то при наведении указателя мыши и получении фокуса ввода цвет фона станет темнее. Кроме того, изменится цвет фона и текста для пунктов с классом `active`:

```
<div class="list-group">
  <a href="#" class="list-group-item list-group-item-danger
    list-group-item-action active">
    Ссылка 1 (активное состояние)</a>
  <a href="#" class="list-group-item list-group-item-danger
    list-group-item-action">
    Ссылка 2</a>
</div>
```

4.8.5. Список с пунктами, содержащими произвольные элементы

Для индикации различных счетчиков (например, количества непрочитанных сообщений) справа от пункта списка можно отобразить компонент `badge` (см. *разд. 4.3*) со значением (рис. 4.22). В этом случае следует превратить пункт списка во flex-контейнер (класс `d-flex`), задать выравнивание элементов внутри строки по центру (класс `align-items-center`), а внутри контейнера — с промежутком между элементами (класс `justify-content-between`). Внутри пункта списка вставляем надпись и тег `` со стилевым классом `badge`:

```
<!-- Только в Bootstrap 4 -->
<div class="container">
  <ul class="list-group">
    <li class="list-group-item d-flex justify-content-between
      align-items-center">Входящие
      <span class="badge badge-primary badge-pill">10</span>
    </li>
    <li class="list-group-item d-flex justify-content-between
      align-items-center">Отправленные
      <span class="badge badge-primary badge-pill">15</span>
    </li>
  </ul>
</div>
```

Пример для Bootstrap 5:

```
<!-- Только в Bootstrap 5 -->
<div class="container">
  <ul class="list-group">
    <li class="list-group-item d-flex justify-content-between
      align-items-center">Входящие
      <span class="badge bg-primary rounded-pill">10</span>
    </li>
    <li class="list-group-item d-flex justify-content-between
      align-items-center">Отправленные
      <span class="badge bg-primary rounded-pill">15</span>
    </li>
  </ul>
</div>
```



Рис. 4.22. Список с компонентом `badge`

Аналогичным образом внутри пункта списка можно вставить заголовки, абзацы и другие элементы (рис. 4.23):

```
<div class="container">
  <ul class="list-group">
    <li class="list-group-item">
      <div class="d-flex w-100 justify-content-between">
        <h5 class="mb-1">Заголовок</h5>
        <small class="text-muted">текст справа</small>
      </div>
      <p class="mb-1">Описание</p>
      <small class="text-muted">Дополнительное описание</small>
    </li>
  </ul>
</div>
```

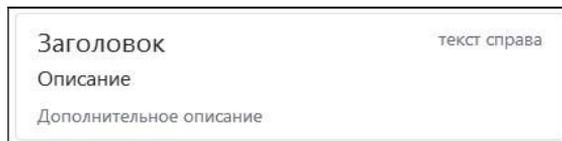


Рис. 4.23. Список с пунктами, содержащими произвольные элементы

4.9. Компонент *breadcrumb*: «хлебные крошки»

Компонент `breadcrumb` реализует цепочку навигации («хлебные крошки»), включающую все страницы на пути к текущей странице (рис. 4.24). Названия страниц в пути разделяются символом `/`, добавляемым с помощью CSS.



Рис. 4.24. Компонент `breadcrumb`: «хлебные крошки»

Для создания компонента нужно выполнить следующие действия:

1. Обернуть содержимое компонента в тег `<nav>` и добавить параметр `aria-label` со значением `breadcrumb`.
2. Внутри тега `<nav>` вложить список и добавить к нему стилевой класс `breadcrumb`, который превращает список во flex-контейнер с горизонтальным выравниванием

и переносом не помещающихся элементов на новую строку. Отображение маркеров списка отключается. Для контейнера задаются внутренние отступы и внешний отступ снизу, а также цвет фона и радиус скругления углов фона.

3. Для пунктов списка добавить стилевой класс `breadcrumb-item` и вложить в них ссылки. Исключением является пункт, описывающий текущую страницу. Для этого пункта нужно дополнительно добавить стилевой класс `active`, параметр `aria-current` со значением `page` и вместо ссылки вложить простой текст с названием текущей страницы.

Пример:

```
<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item">
      <a href="/">Главная</a>
    </li>
    <li class="breadcrumb-item">
      <a href="/shop/">Интернет-магазин</a>
    </li>
    <li class="breadcrumb-item active" aria-current="page">
      Книги
    </li>
  </ol>
</nav>
```

4.10. Компонент *pagination*: постраничная навигация

Компонент `pagination` реализует набор ссылок с номерами страниц (рис. 4.25), используемый при разбиении большого документа на отдельные страницы, — например, результатов поиска по какому-либо запросу. В начало набора можно добавить ссылку на предыдущую страницу, а в конец — ссылку на следующую страницу.

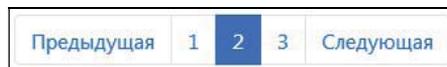


Рис. 4.25. Компонент `pagination`: постраничная навигация

Для создания компонента нужно выполнить следующие действия:

1. Обернуть содержимое компонента в тег `<nav>` и добавить параметр `aria-label` с описанием.
2. Внутри тега `<nav>` вложить список и добавить к нему стилевой класс `pagination`, который превращает список во flex-контейнер с горизонтальным выравниванием. Отображение маркеров списка отключается.
3. Для пунктов списка добавить стилевой класс `page-item` и вложить в них ссылки со стилевым классом `page-link`. При наведении указателя мыши на ссылку изме-

няется цвет фона и текста ссылки, а ссылка в этом случае не подчеркивается. При получении ссылкой фокуса ввода отображается тень рамки.

- Для пункта, описывающего текущую страницу, нужно дополнительно добавить стилевой класс `active` и параметр `aria-current` со значением `page`. Вместо ссылки в этот пункт можно вложить тег `` с номером страницы, со стилевым классом `page-link` и скрытым элементом со стилевым классом `sr-only`, содержащим описание для программ чтения с экрана.

Пример:

```
<nav aria-label="Постраничная навигация">
  <ul class="pagination">
    <li class="page-item">
      <a class="page-link" href="?id=1">Предыдущая</a>
    </li>
    <li class="page-item">
      <a class="page-link" href="?id=1">1</a>
    </li>
    <li class="page-item active" aria-current="page">
      <span class="page-link">
        2<span class="sr-only">(текущая страница)</span>
      </span>
    </li>
    <li class="page-item">
      <a class="page-link" href="?id=3">3</a>
    </li>
    <li class="page-item">
      <a class="page-link" href="?id=3">Следующая</a>
    </li>
  </ul>
</nav>
```

Если текущей страницей является первая или последняя страница, то ссылки **Предыдущая** или **Следующая** можно либо не добавлять, либо сделать эти пункты недоступными. Во втором случае для пункта списка нужно добавить стилевой класс `disabled`, а в пункт вложить тег `` со стилевым классом `page-link`:

```
<li class="page-item disabled">
  <span class="page-link">Предыдущая</span>
</li>
```

Если вместо тега `` вы используете ссылку, то нужно добавить параметр `tabindex` со значением `-1`, чтобы исключить ссылку из порядка обхода с помощью клавиши `<Tab>`, и параметр `aria-disabled` со значением `true`:

```
<li class="page-item disabled">
  <a class="page-link" href="#" tabindex="-1" aria-disabled="true">
    Предыдущая
  </a>
</li>
```

Вместо текста **Предыдущая** можно использовать символ с кодом `«` (в виде двух стрелок, указывающих влево), а вместо текста **Следующая** — символ с кодом `»` (в виде двух стрелок, указывающих вправо). В этом случае для ссылки нужно добавить параметр `aria-label` с описанием для программ чтения с экрана, а код символа вложить в тег `` с параметром `aria-hidden`, имеющим значение `true`:

```
<li class="page-item">
  <a class="page-link" href="?id=1" aria-label="Предыдущая">
    <span aria-hidden="true">&laquo;</span>
  </a>
</li>
```

Если к списку, помимо стилевого класса `pagination`, добавить класс `pagination-sm`, то размер компонента уменьшится, а если добавить класс `pagination-lg`, то, наоборот, увеличится.

По умолчанию блок со ссылками выравнивается по левому краю. Для выравнивания по центру нужно добавить стилевой класс `justify-content-center`, а для выравнивания по правому краю — класс `justify-content-end`.

Уменьшим размер компонента и произведем выравнивание блока со ссылками по центру. Вместо текстов **Предыдущая** и **Следующая** отобразим символы `«` и `»` соответственно:

```
<nav aria-label="Постраничная навигация">
  <ul class="pagination pagination-sm justify-content-center">
    <li class="page-item">
      <a class="page-link" href="?id=1" aria-label="Предыдущая">
        <span aria-hidden="true">&laquo;</span>
      </a>
    </li>
    <li class="page-item">
      <a class="page-link" href="?id=1">1</a>
    </li>
    <li class="page-item active" aria-current="page">
      <span class="page-link">
        2<span class="sr-only">(текущая страница)</span>
      </span>
    </li>
    <li class="page-item">
      <a class="page-link" href="?id=3">3</a>
    </li>
    <li class="page-item">
      <a class="page-link" href="?id=3" aria-label="Следующая">
        <span aria-hidden="true">&raquo;</span>
      </a>
    </li>
  </ul>
</nav>
```

4.11. Компонент *navbar*: панель навигации

Компонент `navbar` реализует панель навигации, которая может состоять из названия фирмы, навигационного блока с обычными ссылками и ссылками с выпадающим меню, а также формой для поиска (рис. 4.26). Кроме того, на маленьких экранах предусмотрена возможность сворачивания (рис. 4.27) и разворачивания (рис. 4.28) панели путем нажатия на кнопку.

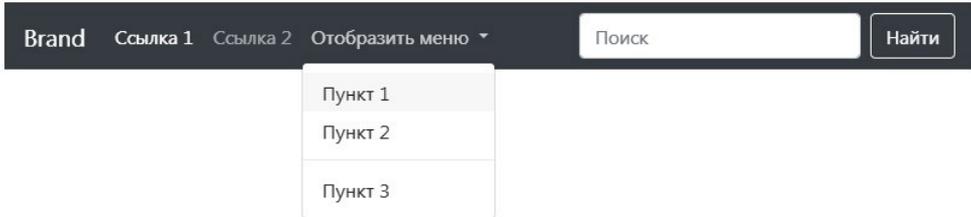


Рис. 4.26. Компонент `navbar`: панель навигации (цветовая схема `navbar-dark`)



Рис. 4.27. Панель навигации для маленьких экранов в свернутом состоянии

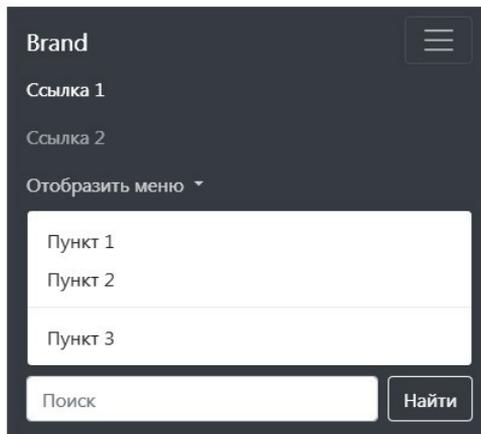


Рис. 4.28. Панель навигации для маленьких экранов в развернутом состоянии

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов `jquery.min.js` и `bootstrap.min.js` (или `bootstrap.bundle.min.js`). Внутри панели навигации выпадающее меню не требует подключения библиотеки `Popper.js`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

4.11.1. Создание панели и вывод простого текста

Панель навигации реализуется с помощью тега `<nav>` со стилевым классом `navbar`, который превращает панель во flex-контейнер с горизонтальным выравниванием, имеющий относительное позиционирование. Если элементы не помещаются на одной строке, то они будут переноситься на новую строку. По вертикали элементы выравниваются по центру, а по горизонтали — первый элемент прижат к началу контейнера, последний — к концу, а остальные равномерно распределяются внутри свободного пространства. Пример создания панели и вывода простого текста внутри нее:

```
<nav class="navbar">
  <span class="navbar-text">Простой текст внутри панели</span>
</nav>
```

В этом примере для вывода простого текста мы воспользовались стилевым классом `navbar-text`, который задает тип блока `inline-block` (только в Bootstrap 4) и добавляет внутренние отступы сверху и снизу. Если мы добавим два элемента, то первый элемент будет выравниваться по левому краю, а второй — по правому краю панели:

```
<nav class="navbar">
  <span class="navbar-text">Текст слева</span>
  <span class="navbar-text">Текст справа</span>
</nav>
```

Помимо класса `navbar`, нужно добавить стилевой класс, задающий цветовую схему для текста внутри панели, а также класс, задающий цвет фона (см. *разд. 1.1.6*):

- ◆ `navbar-light` — темный цвет текста для использования со светлым фоном (рис. 4.29):

```
<nav class="navbar navbar-light bg-light">
  <span class="navbar-text">.navbar-light .bg-light</span>
</nav>
```

- ◆ `navbar-dark` — светлый цвет текста для использования с темным фоном (см. рис. 4.26):

```
<nav class="navbar navbar-dark bg-dark">
  <span class="navbar-text">.navbar-dark .bg-dark</span>
</nav>
```

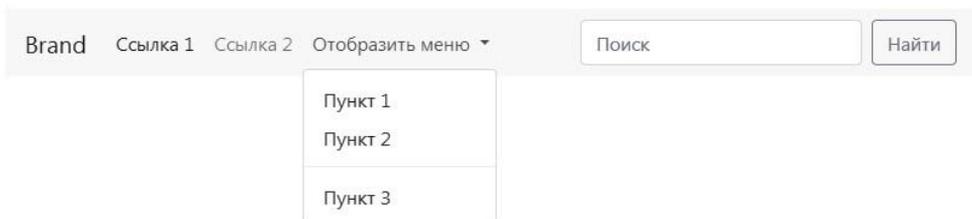


Рис. 4.29. Компонент `navbar`: панель навигации (цветовая схема `navbar-light`)

Пример указания произвольного цвета фона панели навигации с помощью CSS-атрибута `background-color`:

```
<nav class="navbar navbar-dark" style="background-color: #563d7c">
  <span class="navbar-text">.navbar-dark #563d7c</span>
</nav>
```

Указание цветовой схемы панели влияет также на ссылки внутри текста с классом `navbar-text`:

```
<nav class="navbar navbar-light bg-light">
  <span class="navbar-text">Текст <a href="#">ссылка</a></span>
</nav>
<nav class="navbar navbar-dark bg-dark">
  <span class="navbar-text">Текст <a href="#">ссылка</a></span>
</nav>
```

Обычно панель навигации занимает всю доступную ширину окна. Если нужно ограничить ширину панели и поместить ее по центру, то следует вложить панель в базовый контейнер (см. *разд. 1.1.5*):

```
<div class="container-sm">
  <nav class="navbar navbar-dark bg-dark">
    <span class="navbar-text">Текст внутри панели</span>
  </nav>
</div>
```

Чтобы ограничить только ширину для элементов внутри панели, следует вложить базовый контейнер внутрь панели:

```
<nav class="navbar navbar-dark bg-dark">
  <div class="container-sm">
    <span class="navbar-text">Текст внутри панели</span>
  </div>
</nav>
```

В Bootstrap 5 содержимое панели навигации всегда нужно вкладывать в базовый контейнер, иначе внутренние отступы слева и справа будут равны нулю.

ПРИМЕЧАНИЕ

В Bootstrap 4 по умолчанию панель навигации не выводится на печать. Чтобы это изменить, следует явным образом добавить стилевой класс из семейства `d-print-*` (см. *разд. 1.9.1*).

4.11.2. Вывод названия фирмы или проекта

Для стилизации названия фирмы или проекта следует к ссылке или тегу `` добавить класс `navbar-brand`. Текст отображается увеличенным размером шрифта:

```
<nav class="navbar navbar-dark bg-dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <span class="navbar-brand">Brand</span>
  </div>
</nav>
```

Пример дополнительного вывода значка:

```
<nav class="navbar navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">
      
      Bootstrap
    </a>
  </div>
</nav>
```

4.11.3. Добавление блока со ссылками

Блок со ссылками реализуется точно так же, как и при использовании компонента `nav` (см. *разд. 4.6*), но вместо стилевого класса `nav` следует добавлять класс `navbar-nav` (хотя можно использовать и класс `nav`, если нужно горизонтальное выравнивание ссылок). Если мы просто добавим класс `navbar-nav`, то ссылки будут размещаться друг под другом по вертикали. Чтобы использовать горизонтальное размещение, надо к тегу `<nav>` добавить стилевой класс `navbar-expand` или классы для адаптивной верстки: `navbar-expand-sm`, `navbar-expand-md`, `navbar-expand-lg`, `navbar-expand-xl`, `navbar-expand-xxl` (Bootstrap 5). Обратите внимание — класс `navbar-expand` изменяет также свойства панели: во-первых, запрещает перенос на новую строку, а во-вторых, задает выравнивание по левому краю панели (при условии, что содержимое панели навигации не вложено в базовый контейнер). Пример:

```
<nav class="navbar navbar-dark bg-dark navbar-expand">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link active" href="#">Ссылка 1</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Ссылка 2</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1"
          aria-disabled="true">Ссылка недоступна</a>
      </li>
    </ul>
  </div>
</nav>
```

Вместо использования тега `` можно добавить ссылки в родительский контейнер с классом `navbar-nav`:

```
<nav class="navbar navbar-dark bg-dark navbar-expand">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <div class="navbar-nav">
```

```

<a class="nav-link active" href="#">Ссылка 1</a>
<a class="nav-link" href="#">Ссылка 2</a>
<a class="nav-link disabled" href="#" tabindex="-1"
  aria-disabled="true">Ссылка недоступна</a>
</div>
</div>
</nav>

```

4.11.4. Добавление ссылки с выпадающим меню

Блок со ссылками может содержать ссылку с выпадающим меню. В этом случае для пункта списка, помимо стилевого класса `nav-item`, следует добавить класс `dropdown`. Для ссылки нужно добавить стилевые классы `nav-link` и `dropdown-toggle`, а также уникальный идентификатор и параметры: `data-toggle` — со значением `dropdown`, `aria-haspopup` — со значением `true` и `aria-expanded` — со значением `false`. Для выпадающего меню добавляем стилевой класс `dropdown-menu` и параметр `aria-labelledby` с идентификатором ссылки.

Напомним, что компонент `dropdown` и создание выпадающего меню мы уже рассматривали в *разд. 3.8*, поэтому не будем здесь повторяться. Однако существует несколько различий, помимо уже рассмотренных. Во-первых, внутри панели навигации выпадающее меню не требует подключения библиотеки `Popper.js`, поэтому достаточно подключить файлы `jquery.min.js` и `bootstrap.min.js`. Во-вторых, свойства выпадающего меню зависят от стилевого класса `navbar-expand` и классов для адаптивной верстки: `navbar-expand-sm`, `navbar-expand-md`, `navbar-expand-lg`, `navbar-expand-xl`, `navbar-expand-xxl` (Bootstrap 5). При наличии этих классов блок со ссылками размещается по горизонтали, и выпадающее меню имеет абсолютное позиционирование. Если эти классы для текущей точки останова не действуют, то блок со ссылками размещается по вертикали и выпадающее меню имеет статическое позиционирование.

До точки останова `sm` поместим блок со ссылками по вертикали, а после нее — по горизонтали, добавив стилевой класс `navbar-expand-sm`:

```

<nav class="navbar navbar-dark bg-dark navbar-expand-sm">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link active" href="#">Ссылка 1</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Ссылка 2</a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" role="button"
          id="link1" data-toggle="dropdown" aria-haspopup="true"
          aria-expanded="false">Отобразить меню
        </a>

```

```
<div class="dropdown-menu" aria-labelledby="link1">
  <a class="dropdown-item" href="#">Пункт 1</a>
  <a class="dropdown-item" href="#">Пункт 2</a>
  <div class="dropdown-divider"></div>
  <a class="dropdown-item" href="#">Пункт 3</a>
</div>
</li>
</ul>
</div>
</nav>
```

4.11.5. Сворачивание и разворачивание блока со ссылками

Если посмотреть на вид панели из предыдущего примера до точки останова `sm`, то согласитесь, что вертикальный блок со ссылками справа от названия фирмы выглядит не очень красиво. Давайте исправим этот недостаток. На месте блока со ссылками поместим кнопку, по нажатию на которую отобразим блок со ссылками ниже панели.

Для кнопки, исполняющей роль переключателя состояния, следует добавить следующие параметры:

- ◆ `class` — со стилевым классом `navbar-toggler`;
- ◆ `data-toggle` — со значением `collapse`;
- ◆ `aria-expanded` — со значением `false` (контейнер с блоком ссылок изначально свернут);
- ◆ `data-target` — должен содержать CSS-селектор, с помощью которого можно найти контейнер с блоком ссылок;
- ◆ `aria-controls` — с идентификатором контейнера с блоком ссылок;
- ◆ `aria-label` — с описанием для программ чтения с экрана, т. к. на кнопке будет отображаться значок.

Для добавления значка на кнопку внутри тега `<button>` вкладываем тег `` со стилевым классом `navbar-toggler-icon`.

Далее нужно вложить блок со ссылками в тег `<div>` со стилевыми классами `collapse` и `navbar-collapse`, а также добавить уникальный идентификатор, который следует указать в параметрах `data-target` (с предваряющим символом `#`) и `aria-controls` тега `<button>`.

Кроме того, с помощью стилевых классов `navbar-expand` или классов для адаптивной верстки: `navbar-expand-sm`, `navbar-expand-md`, `navbar-expand-lg`, `navbar-expand-xl`, `navbar-expand-xxl` (Bootstrap 5) надо указать точку останова. До этой точки будет видна кнопка (см. рис. 4.27), с помощью которой можно развернуть или свернуть блок со ссылками. Ссылки внутри блока размещаются по вертикали (см. рис. 4.28). Начиная с этой точки, кнопка скрывается, а блок со ссылками отображается. Ссыл-

ки внутри блока размещаются по горизонтали. Пример указания точки останова `sm` путем добавления стилового класса `navbar-expand-sm`:

```
<nav class="navbar navbar-dark bg-dark navbar-expand-sm">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
      data-target="#div1" aria-controls="div1"
      aria-expanded="false" aria-label="Свернуть или развернуть">
      <span class="navbar-toggler-icon"></span>
    </button>

  <div class="collapse navbar-collapse" id="div1">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link active" href="#">Ссылка 1</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Ссылка 2</a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" role="button"
          id="link1" data-toggle="dropdown" aria-haspopup="true"
          aria-expanded="false"> Отобразить меню
        </a>
        <div class="dropdown-menu" aria-labelledby="link1">
          <a class="dropdown-item" href="#">Пункт 1</a>
          <a class="dropdown-item" href="#">Пункт 2</a>
          <div class="dropdown-divider"></div>
          <a class="dropdown-item" href="#">Пункт 3</a>
        </div>
      </li>
    </ul>
  </div>
</div>
</nav>
```

4.11.6. Добавление формы

Чтобы на панель навигации добавить форму, нужно в Bootstrap 4 для тега `<form>` указать стиливой класс `form-inline`:

```
<!-- Только в Bootstrap 4 -->
<nav class="navbar navbar-dark bg-dark">
  <a class="navbar-brand" href="#">Brand</a>
  <form class="form-inline">
    <input class="form-control mr-sm-2" type="search"
      placeholder="Поиск" name="txt">
    <button class="btn btn-outline-light my-2 my-sm-0"
      type="submit">Найти</button>
  </form>
</nav>
```

В Bootstrap 5 для тега `<form>` следует указать стилевой класс `d-flex`:

```
<nav class="navbar navbar-dark bg-dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <form class="d-flex">
      <input class="form-control mr-2" type="search"
        placeholder="Поиск" name="txt">
      <button class="btn btn-outline-light"
        type="submit">Найти</button>
    </form>
  </div>
</nav>
```

4.11.7. Позиционирование панели навигации

По умолчанию панель навигации имеет статическое позиционирование. Это означает, что при прокрутке панель будет перемещаться вместе с содержимым страницы. Чтобы прикрепить панель навигации к верху или низу окна, нужно воспользоваться следующими стилевыми классами:

- ◆ `fixed-top` — фиксированное позиционирование с привязкой к верху окна веб-браузера (требуется добавить внутренний отступ сверху для тега `<body>`):

```
.fixed-top {
  position: fixed;
  top: 0;
  right: 0;
  left: 0;
  z-index: 1030;
}
```

- ◆ `sticky-top` — привязка к верху окна веб-браузера. Вначале используется относительное позиционирование. Как только элемент доходит до указанной позиции, применяется фиксированное позиционирование:

```
.sticky-top {
  position: sticky;
  top: 0;
  z-index: 1020;
}
```

В Bootstrap 5 можно воспользоваться следующими классами для адаптивной верстки: `sticky-sm-top`, `sticky-md-top`, `sticky-lg-top`, `sticky-xl-top`, `sticky-xxl-top`;

- ◆ `fixed-bottom` — фиксированное позиционирование с привязкой к низу окна веб-браузера (требуется добавить внутренний отступ снизу для тега `<body>`, а также изменить направление выпадения меню):

```
.fixed-bottom {
  position: fixed;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 1030;
}
```

Пример использования стилевого класса `sticky-top` приведен в листинге 4.10.

Листинг 4.10. Панель навигации с привязкой к верху окна веб-браузера

```

<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Компонент navbar: панель навигации</title>
</head>
<body>

<nav class="navbar navbar-dark bg-dark navbar-expand-lg sticky-top">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Brand</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
      data-target="#div1" aria-controls="div1"
      aria-expanded="false" aria-label="Свернуть или развернуть">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="div1">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" href="#">Ссылка 1</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Ссылка 2</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" role="button"
            id="link1" data-toggle="dropdown" aria-haspopup="true"
            aria-expanded="false"> Отобразить меню
          </a>
          <div class="dropdown-menu" aria-labelledby="link1">
            <a class="dropdown-item" href="#">Пункт 1</a>
            <a class="dropdown-item" href="#">Пункт 2</a>
            <div class="dropdown-divider"></div>
            <a class="dropdown-item" href="#">Пункт 3</a>
          </div>
        </li>
      </ul>
      <form class="d-flex my-2 my-lg-0 ml-auto" action="#">
        <input class="form-control mr-2"
          type="search" placeholder="Поиск" name="txt">
        <button class="btn btn-outline-light"
          type="submit">Найти</button>
      </form>
    </div>
  </div>
</nav>

```

```
</form>
</div>
</div>
</nav>

<div class="container bg-light" style="min-height: 1200px">
  <h1>Заголовок</h1>
  <p>Текст</p>
</div>

<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
</body>
</html>
```

4.12. Компонент *carousel*: циклическое повторение слайдов

Компонент `carousel` реализует циклическое повторение слайдов через указанный промежуток времени (по умолчанию — 5 секунд). При наведении указателя мыши на слайд смена слайдов приостанавливается до момента выведения указателя мыши из области компонента. Слайды могут сразу сменять друг друга без анимации либо с анимацией смещения по оси *x* (перелистывания) или анимацией изменения прозрачности. Существует возможность отобразить значки для перемещения к предыдущему или следующему слайду, а также индикаторы для перехода на произвольный слайд (рис. 4.30).



Рис. 4.30. Компонент `carousel`: циклическое повторение слайдов

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.carousel.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Carousel.VERSION);
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов *jquery.min.js* и *bootstrap.min.js* (или *bootstrap.bundle.min.js*), а также указание параметра `data-ride` со значением `carousel`. В Bootstrap 5 можно не подключать файл *jquery.min.js*, если не используется интерфейс доступа через jQuery.

4.12.1. Создание компонента

Для создания компонента нужно выполнить следующие действия:

1. Создать родительский контейнер и добавить к нему стилевые классы `carousel` и `slide`, а также параметр `data-ride` со значением `carousel` и уникальный идентификатор. Класс `carousel` служит маркером компонента и, кроме того, добавляет для контейнера относительное позиционирование. При добавлении класса `slide` слайды будут сменяться с CSS-анимацией смещения по оси `x` (перелистывание). Если класс `slide` не добавить, то новый слайд сразу заменит текущий слайд. Параметр `data-ride` является маркером, служащим для поиска компонента из программы на JavaScript и запуска процесса смены слайдов. Без этого параметра компонент работать не будет, пока явно не вызван метод `carousel()` (нельзя одновременно указывать параметр `data-ride` и вызывать метод `carousel()`):

```
$('#myCarousel').carousel();
```

Методу `carousel()` можно сразу передать объект с опциями. Например, установим интервал смены слайдов в 2 секунды (по умолчанию интервал составляет 5 секунд) и сделаем процесс смены слайдов бесконечным:

```
$('#myCarousel').carousel({
  interval: 2000,
  wrap: true
});
```

Пример для Bootstrap 5 без jQuery:

```
var myCarousel = document.getElementById('myCarousel');
new bootstrap.Carousel(myCarousel, {
  interval: 2000,
  wrap: true
});
```

2. Внутри контейнера со стилевым классом `carousel` добавить родительский контейнер для всех слайдов с классом `carousel-inner`. Этот класс задает относительное позиционирование, ширину `100%` и сокрытие при выходе содержимого за пределы контейнера, а также запрещает обтекание.
3. Внутри контейнера со стилевым классом `carousel-inner` добавить контейнер для отдельного слайда с классом `carousel-item`. Сколько слайдов, столько должно быть и контейнеров с классом `carousel-item`. Этот класс задает относительное позиционирование, ширину `100%`, обтекание (`float: left`) и отключает видимость слайда, а также устанавливает параметры CSS-анимации. Чтобы сделать слайд активным и включить видимость, следует дополнительно добавить стилевой класс `active` (только один слайд может иметь этот класс).

4. Внутри контейнера со стилевым классом `carousel-item` добавить изображение со стилевыми классами `d-block` и `w-100`, чтобы изображение занимало всю доступную ширину.

Пример создания компонента и добавления трех слайдов:

```
<div class="container">
  <div class="carousel slide" data-ride="carousel" id="myCarousel">
    <div class="carousel-inner">
      <div class="carousel-item active">
        
      </div>
      <div class="carousel-item">
        
      </div>
      <div class="carousel-item">
        
      </div>
    </div>
  </div>
</div>
```

ПРИМЕЧАНИЕ

Предварительно создайте в каталоге `C:\book\img` изображения с названиями `slide1.jpg`, `slide2.jpg` и `slide3.jpg` с размерами 1110×500 пикселей.

4.12.2. Управление компонентом пользователем

При наличии параметра `data-ride` со значением `carousel` или при вызове метода `carousel()` с опцией `ride`, имеющей значение `"carousel"`, после загрузки веб-страницы автоматически запускается смена слайдов с первого до последнего и далее опять по кругу. Если пользователь наведет на слайд указатель мыши, то смена слайдов приостанавливается. При выведении указателя со слайда, смена слайдов автоматически запускается снова. Приостановку при наведении указателя мыши можно отменить с помощью параметра `data-pause` со значением `false`, добавленного для элемента со стилевым классом `carousel`, или опции `pause` со значением `false` (см. разд. 4.12.5).

Указать значение можно также с помощью параметра `data-pause`, добавленного для элемента со стилевым классом `carousel`.

Чтобы пользователь мог перемещаться между слайдами вперед или назад путем нажатия на значки со стрелками, в контейнер со стилевым классом `carousel` подправляем следующие ссылки:

- ♦ для ссылки, указывающей на предыдущий слайд, добавляем стилевой класс `carousel-control-prev`, параметр `href` с CSS-селектором, с помощью которого можно найти компонент `carousel`, а также параметр `data-slide` со значением `prev`. Внутри ссылки вкладываем тег `` со стилевым классом `carousel-control-prev-icon` (значок со стрелкой влево) и параметром `aria-hidden` со значением `true`.

Дополнительно для программ чтения с экрана вкладываем скрытое описание. Перемещение к предыдущему слайду возможно с помощью щелчка левой кнопкой мыши на значке или путем нажатия стрелки влево на клавиатуре (нужно чтобы какой-либо значок внутри компонента был в фокусе ввода):

```
<a class="carousel-control-prev" href="#myCarousel"
  role="button" data-slide="prev">
  <span class="carousel-control-prev-icon"
    aria-hidden="true"></span>
  <span class="sr-only">Предыдущий слайд</span>
</a>
```

- ◆ для ссылки, указывающей на следующий слайд, добавляем стилевой класс `carousel-control-next`, параметр `href` с CSS-селектором, с помощью которого можно найти компонент `carousel`, а также параметр `data-slide` со значением `next`. Внутри ссылки вкладываем тег `` со стилевым классом `carousel-control-next-icon` (значок со стрелкой вправо) и параметром `aria-hidden` со значением `true`. Дополнительно для программ чтения с экрана вкладываем скрытое описание. Перемещение к предыдущему слайду возможно с помощью щелчка левой кнопкой мыши на значке или путем нажатия стрелки вправо на клавиатуре (нужно чтобы какой-либо значок внутри компонента был в фокусе ввода):

```
<a class="carousel-control-next" href="#myCarousel"
  role="button" data-slide="next">
  <span class="carousel-control-next-icon"
    aria-hidden="true"></span>
  <span class="sr-only">Следующий слайд</span>
</a>
```

Чтобы пользователь мог перемещаться между произвольными слайдами путем нажатия на индикаторы, в контейнер со стилевым классом `carousel` добавляем список с классом `carousel-indicators`. Для каждого пункта списка указываем параметр `data-target` с CSS-селектором, с помощью которого можно найти компонент `carousel`, а также параметр `data-slide-to`, который в качестве значения содержит индекс слайда внутри компонента `carousel` (нумерация слайдов начинается с 0). Для пункта, описывающего текущий слайд, нужно указать стилевой класс `active`.

Пример компонента со значками и индикаторами:

```
<div class="container">
  <div class="carousel slide" data-ride="carousel" id="myCarousel">
    <div class="carousel-inner">
      <div class="carousel-item active">
        
      </div>
      <div class="carousel-item">
        
      </div>
      <div class="carousel-item">
        
      </div>
    </div>
  </div>
```

```

<a class="carousel-control-prev" href="#myCarousel"
  role="button" data-slide="prev">
  <span class="carousel-control-prev-icon"
    aria-hidden="true"></span>
  <span class="sr-only">Предыдущий слайд</span>
</a>
<a class="carousel-control-next" href="#myCarousel"
  role="button" data-slide="next">
  <span class="carousel-control-next-icon"
    aria-hidden="true"></span>
  <span class="sr-only">Следующий слайд</span>
</a>

<ul class="carousel-indicators">
  <li data-target="#myCarousel" data-slide-to="0"
    class="active"></li>
  <li data-target="#myCarousel" data-slide-to="1"></li>
  <li data-target="#myCarousel" data-slide-to="2"></li>
</ul>
</div>
</div>

```

4.12.3. Добавление надписей

Поверх слайдов можно вывести какие-либо пояснительные надписи. Для этого после изображения слайда добавляем тег `<div>` со стилевым классом `carousel-caption`. Чтобы надпись отображалась только после точки останова `md`, добавляем классы `d-none` и `d-md-block`. Внутри тега `<div>` вкладываем различные элементы — например, заголовки и абзацы:

```

<div class="container">
  <div class="carousel slide" data-ride="carousel" id="myCarousel">
    <div class="carousel-inner">
      <div class="carousel-item active">
        
        <div class="carousel-caption d-none d-md-block">
          <h5>Слайд 1</h5>
          <p>Описание слайда 1</p>
        </div>
      </div>
      <div class="carousel-item">
        
        <div class="carousel-caption d-none d-md-block">
          <h5>Слайд 2</h5>
          <p>Описание слайда 2</p>
        </div>
      </div>
      <div class="carousel-item">
        

```

```

    <div class="carousel-caption d-none d-md-block">
      <h5>Слайд 3</h5>
      <p>Описание слайда 3</p>
    </div>
  </div>
</div>
</div>
</div>

```

4.12.4. Способы смены слайдов и указание интервала

При добавлении стилевых классов `slide` слайды будут сменяться с CSS-анимацией смещения по оси `x` (перелистывание). Если класс `slide` не добавить, то новый слайд сразу заменит текущий слайд. Если же помимо стилевых классов `slide` добавить класс `carousel-fade`, то слайды будут сменять друг друга с анимацией изменения прозрачности.

По умолчанию один слайд отображается в течение пяти секунд. Чтобы изменить интервал, нужно указать другое значение в миллисекундах для свойства `interval` при вызове метода `carousel()`. Укажем две секунды:

```

$('#myCarousel').carousel({
  interval: 2000
});

```

Пример для Bootstrap 5 без jQuery:

```

var myCarousel = document.getElementById('myCarousel');
new bootstrap.Carousel(myCarousel, {
  interval: 2000
});

```

Существует также возможность указать интервал в качестве значения параметра `data-interval`, добавленного для контейнера со стилевым классом `carousel`:

```

<div class="carousel slide" id="myCarousel"
  data-ride="carousel" data-interval="2000">

```

Можно задать интервал для отдельного слайда. Для этого к тегу `<div>` со стилевым классом `carousel-item` добавляем параметр `data-interval` с нужным значением в миллисекундах:

```

<div class="carousel-item" data-interval="2000">

```

Для первого слайда зададим интервал в одну секунду, для второго — в две секунды, а для третьего — в десять секунд (используется значение для всех слайдов без конкретного значения). Слайды будут сменять друг друга с анимацией изменения прозрачности:

```

<div class="container">
  <div class="carousel slide carousel-fade"
    data-ride="carousel" data-interval="10000" id="myCarousel">
    <div class="carousel-inner">
      <div class="carousel-item active" data-interval="1000">
        

```

```
</div>
<div class="carousel-item" data-interval="2000">
  
</div>
<div class="carousel-item">
  
</div>
</div>
</div>
</div>
```

4.12.5. Управление компонентом из программы

Для управления компонентом `carousel` из программы на jQuery следует вызвать метод `carousel()` и передать ему одно из следующих значений:

- ◆ без значения — запускает процесс смены слайдов со значениями опций по умолчанию (нельзя одновременно указывать параметр `data-ride` и вызывать метод `carousel()`);
- ◆ объект с опциями — можно указать следующие свойства:
 - `interval` — интервал в миллисекундах перед сменой слайда. Значение по умолчанию: `5000`. Укажите значение `false`, если хотите сменять слайды вручную, — например, с помощью значков или индикаторов. Интервал можно также указать с помощью HTML-параметра `data-interval` (см. *разд. 4.12.4*);
 - `keyboard` — если указано значение `true` (значение по умолчанию), то перемещение к предыдущему или следующему слайду возможно путем нажатия стрелки влево или вправо на клавиатуре (нужно чтобы какой-либо значок внутри компонента был в фокусе ввода). Значение `false` отключает обработку событий клавиатуры. Указать значение можно также с помощью HTML-параметра `data-keyboard`, добавленного для элемента со стилевым классом `carousel`;
 - `pause` — если указано значение `'hover'` (значение по умолчанию), то при наведении указателя мыши на слайд, смена слайдов приостанавливается. При выведении указателя со слайда смена слайдов автоматически запускается снова. Значение `false` отключает приостановку смены слайдов. Запустить смену слайдов вручную нужно будет путем вызова метода `carousel('cycle')` (`cycle()` в Bootstrap 5) или указанием значения `'carousel'` для опции `ride`. Указать значение можно также с помощью HTML-параметра `data-pause`, добавленного для элемента со стилевым классом `carousel`;
 - `ride` — если указано значение `'carousel'`, то процесс смены слайдов запустится автоматически сразу после загрузки веб-страницы (в Bootstrap 5 работает только при подключении jQuery или при использовании статического метода `carouselInterface()`). Если указано значение `false` (значение по умолчанию), то запуск смены слайдов нужно выполнить вручную с помощью метода `carousel('cycle')`. Смена слайдов запустится автоматически при наведении и

выведении указателя мыши пользователем — при условии, что опция `pause` имеет значение `'hover'`. Указать значение можно также с помощью HTML-параметра `data-ride`, добавленного для элемента со стилевым классом `carousel`;

- `wrap` — если указано значение `true` (значение по умолчанию), то будет происходить смена слайдов с первого до последнего и далее снова с первого до последнего бесконечно по кругу. Если указано значение `false`, то смена слайдов произойдет только с первого и до последнего. Указать значение можно также с помощью HTML-параметра `data-wrap`, добавленного для элемента со стилевым классом `carousel`;
- `touch` — если указано значение `true` (значение по умолчанию), то компонент будет реагировать на жесты влево и вправо на сенсорных устройствах. Значение `false` отключает эту возможность. Указать значение можно также с помощью HTML-параметра `data-touch`, добавленного для элемента со стилевым классом `carousel`;
- ◆ `'prev'` — переход к предыдущему слайду;
- ◆ `'next'` — переход к следующему слайду;
- ◆ индекс — переход к слайду с указанным индексом (нумерация слайдов начинается с 0);
- ◆ `'pause'` — останавливает смену слайдов;
- ◆ `'cycle'` — запускает смену слайдов;
- ◆ `'dispose'` — удаляет данные, связанные с компонентом.

Пример отображения второго слайда (имеет индекс 1):

```
$('#myCarousel').carousel(1);
```

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Carousel`:

```
<Объект> = new bootstrap.Carousel(<Элемент>[, <Объект с опциями>])
```

Пример создания объекта:

```
var myCarousel = document.getElementById('myCarousel');
new bootstrap.Carousel(myCarousel, {
  interval: 2000,
  wrap: true
});
```

Для создания объекта и сохранения его в элементе можно воспользоваться статическим методом `carouselInterface()`:

```
var myCarousel = document.getElementById('myCarousel');
bootstrap.Carousel.carouselInterface(myCarousel, {
  interval : 3000,
  keyboard : true,
  pause    : 'hover',
  ride     : 'carousel',
  wrap     : true,
  touch    : true
});
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект:

```
var myCarousel = document.getElementById('myCarousel');
var myCarouselInstance = bootstrap.Carousel.getInstance(myCarousel);
```

Класс `Carousel` содержит следующие методы:

- ◆ `prev()` — переход к предыдущему слайду;
- ◆ `next()` — переход к следующему слайду;
- ◆ `to(<Индекс>)` — переход к слайду с указанным индексом (нумерация слайдов начинается с 0);
- ◆ `pause()` — останавливает смену слайдов;
- ◆ `cycle()` — запускает смену слайдов;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример отображения второго слайда (имеет индекс 1):

```
myCarouselInstance.to(1);
```

ОБРАТИТЕ ВНИМАНИЕ!

Методы запускают переключение состояния асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

4.12.6. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий (события генерируются для элемента со стилевым классом `carousel`):

- ◆ `slide.bs.carousel` — событие перед сменой слайда. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то смены слайда не произойдет:

```
var flagSlide = false;
$('#myCarousel').on('slide.bs.carousel', function(e) {
  if (!flagSlide) e.preventDefault();
});
```

Пример для Bootstrap 5 без jQuery:

```
var flagSlide = false;
var myCarousel = document.getElementById('myCarousel');
myCarousel.addEventListener('slide.bs.carousel', function(e) {
  if (!flagSlide) e.preventDefault();
}, false);
```

- ◆ `slid.bs.carousel` — событие после смены слайда.

Внутри обработчика через объект события доступны следующие свойства:

- ◆ `direction` — направление смены слайдов: "left" или "right";
- ◆ `relatedTarget` — ссылка на элемент со слайдом, который будет показан;

- ◆ `from` — индекс текущего слайда;
- ◆ `to` — индекс следующего слайда.

Пример управления компонентом `carousel` из программы и обработки событий приведен в листинге 4.11.

Листинг 4.11. Компонент `carousel`: циклическое повторение слайдов

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Компонент carousel: циклическое повторение слайдов</title>
</head>
<body>

<div class="container mt-3">
  <div class="carousel slide" id="myCarousel">
    <!-- Индикаторы -->
    <ul class="carousel-indicators">
      <li data-target="#myCarousel" data-slide-to="0"
        class="active"></li>
      <li data-target="#myCarousel" data-slide-to="1"></li>
      <li data-target="#myCarousel" data-slide-to="2"></li>
    </ul>
    <!-- Слайды -->
    <div class="carousel-inner">
      <div class="carousel-item active" id="slide1">
        
      </div>
      <div class="carousel-item" id="slide2">
        
      </div>
      <div class="carousel-item" id="slide3">
        
      </div>
    </div>
    <!-- Значки -->
    <a class="carousel-control-prev" href="#myCarousel"
      role="button" data-slide="prev">
      <span class="carousel-control-prev-icon"
        aria-hidden="true"></span>
      <span class="sr-only">Предыдущий слайд</span>
    </a>
    <a class="carousel-control-next" href="#myCarousel"
      role="button" data-slide="next">
```

```
<span class="carousel-control-next-icon"
  aria-hidden="true"></span>
<span class="sr-only">Следующий слайд</span>
</a>
</div>
<div class="my-3">
  <button type="button" class="btn btn-primary mb-2" id="btnPrev">
    Предыдущий слайд</button>
  <button type="button" class="btn btn-primary mb-2" id="btnNext">
    Следующий слайд</button>
  <button type="button" class="btn btn-primary mb-2" id="btnNumber">
    К слайду 2</button>
  <button type="button" class="btn btn-primary mb-2" id="btnPause">
    Приостановить смену</button>
  <button type="button" class="btn btn-primary mb-2" id="btnCycle">
    Запустить смену</button>
</div>
<p>Результат обработки событий см. в окне консоли.</p>
</div>

<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
  $('#myCarousel').carousel({
    interval : 3000,
    keyboard : true,
    pause    : 'hover',
    ride     : 'carousel',
    wrap     : true,
    touch    : true
  });
  $('#btnPrev').click( function() {
    $('#myCarousel').carousel('prev');
  });
  $('#btnNext').click( function() {
    $('#myCarousel').carousel('next');
  });
  $('#btnNumber').click( function() {
    $('#myCarousel').carousel(1);
  });
  $('#btnPause').click( function() {
    $('#myCarousel').carousel('pause');
  });
  $('#btnCycle').click( function() {
    $('#myCarousel').carousel('cycle');
  });
  $('#myCarousel').on('slide.bs.carousel', function(e) {
    console.log('slide.bs.carousel');
  });
});

```

```

        console.log('--direction ' + e.direction);
        console.log('--relatedTarget ' + e.relatedTarget.id);
        console.log('--from ' + e.from);
        console.log('--to ' + e.to);
    }).on('slid.bs.carousel', function(e) {
        console.log('slid.bs.carousel');
    });
});
</script>
</body>
</html>

```

Изменим содержимое тега `<script>` из листинга 4.11 так, чтобы программа работала в Bootstrap 5 без jQuery:

```

var myCarousel = document.getElementById('myCarousel');
var myCarouselInstance = new bootstrap.Carousel(myCarousel, {
    interval : 3000,
    keyboard : true,
    pause    : 'hover',
    wrap     : true,
    touch    : true
});
myCarouselInstance.cycle();

var btnPrev = document.getElementById('btnPrev');
btnPrev.addEventListener('click', function() {
    myCarouselInstance.prev();
}, false);
var btnNext = document.getElementById('btnNext');
btnNext.addEventListener('click', function() {
    myCarouselInstance.next();
}, false);
var btnNumber = document.getElementById('btnNumber');
btnNumber.addEventListener('click', function() {
    myCarouselInstance.to(1);
}, false);
var btnPause = document.getElementById('btnPause');
btnPause.addEventListener('click', function() {
    myCarouselInstance.pause();
}, false);
var btnCycle = document.getElementById('btnCycle');
btnCycle.addEventListener('click', function() {
    myCarouselInstance.cycle();
}, false);

myCarousel.addEventListener('slide.bs.carousel', function(e) {
    console.log('slide.bs.carousel');
    console.log('--direction ' + e.direction);
    console.log('--relatedTarget ' + e.relatedTarget.id);

```

```
console.log('--from ' + e.from);
console.log('--to ' + e.to);
}, false);
myCarousel.addEventListener('slid.bs.carousel', function(e) {
  console.log('slid.bs.carousel');
}, false);
```

4.13. Компонент *scrollspy*: отслеживание прокрутки

Компонент `scrollspy` позволяет отслеживать положение полосы прокрутки документа или блока и делать активными соответствующие ссылки на панели навигации, реализуемой с помощью компонентов `navbar`, `nav` или `list-group`.

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.scrollspy.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.ScrollSpy.VERSION);
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов `jquery.min.js` и `bootstrap.min.js` (или `bootstrap.bundle.min.js`), а также указание параметра `data-spy` со значением `scroll`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

4.13.1. Создание компонента

Для создания компонента нужно выполнить следующие действия:

1. Задать для тега `<body>` или блока, внутри которого требуется отслеживать положение полосы прокрутки, относительное позиционирование, — например, с помощью стилевого класса `position-relative`. Для блока надо указать высоту и включить отображение полос прокрутки — например, с помощью стилевого класса `overflow-auto`. Добавить следующие параметры:
 - `data-spy` — со значением `scroll`;
 - `data-target` — с CSS-селектором, с помощью которого можно найти блок со ссылками;
 - `data-offset` — можно указать число пикселей до элемента сверху при вычислении положения. Если параметр не указан, то используется значение 10 пикселей.
2. Создать блок со ссылками с помощью компонентов `navbar`, `nav` или `list-group` и указать параметр `id` с уникальным идентификатором. Этот идентификатор с предваряющим символом `#` надо задать в качестве значения параметра `data-target` тега `<body>` или блока. Каждая ссылка в параметре `href` должна содержать CSS-селектор, с помощью которого можно найти раздел внутри тега `<body>` или блока.

Если панель навигации содержит компонент `dropdown`, то будет выделяться ссылка с вложенным меню и пункт внутри этого меню (см. далее листинг 4.12). Выделение выполняется добавлением к ссылке и пункту стилевого класса `active`. Если блок со ссылками имеет вложенную структуру, то будет выделен пункт с разделом и пункт с текущим подразделом:

```
<div class="row">
  <div class="col-sm-3">
    <nav class="nav nav-pills flex-column" id="nav1">
      <a class="nav-link" href="#section1">Раздел 1</a>
      <a class="nav-link" href="#section2">Раздел 2</a>
      <nav class="nav nav-pills flex-column">
        <a class="nav-link ml-3 my-1"
          href="#section2-1">Раздел 2.1</a>
        <a class="nav-link ml-3"
          href="#section2-2">Раздел 2.2</a>
      </nav>
    </nav>
  </div>
  <div class="col-sm-9">
    <div class="position-relative overflow-auto"
      style="height: 300px"
      data-spy="scroll" data-target="#nav1" data-offset="0">
      <div class="bg-info p-3" style="height: 400px" id="section1">
        <h1>Раздел 1</h1>
      </div>
      <div class="bg-success p-3" style="height: 400px" id="section2">
        <h1>Раздел 2</h1>
      </div>
      <div class="bg-info p-3" style="height: 600px" id="section2-1">
        <h2>Раздел 2.1</h2>
      </div>
      <div class="bg-light p-3" style="height: 600px" id="section2-2">
        <h2>Раздел 2.2</h2>
      </div>
    </div>
  </div>
</div>
```

Пример использования компонента `list-group`:

```
<div class="row">
  <div class="col-sm-3">
    <div class="list-group" id="list1">
      <a class="list-group-item list-group-item-action"
        href="#section1">Раздел 1</a>
      <a class="list-group-item list-group-item-action"
        href="#section2">Раздел 2</a>
      <a class="list-group-item list-group-item-action"
        href="#section3">Раздел 3</a>
    </div>
  </div>
```

```

</div>
<div class="col-sm-9">
  <div class="position-relative overflow-auto"
    style="height: 300px"
    data-spy="scroll" data-target="#list1" data-offset="0">
    <div class="bg-info p-3" style="height: 400px" id="section1">
      <h1>Раздел 1</h1>
    </div>
    <div class="bg-success p-3" style="height: 400px" id="section2">
      <h1>Раздел 2</h1>
    </div>
    <div class="bg-info p-3" style="height: 600px" id="section3">
      <h1>Раздел 3</h1>
    </div>
  </div>
</div>
</div>
</div>

```

4.13.2. Управление компонентом из программы

Для управления компонентом `scrollspy` из программы на jQuery, нужно вызвать метод `scrollspy()` и передать ему одно из следующих значений:

- ◆ объект с опциями — можно указать следующие свойства:
 - `offset` — число пикселей до элемента сверху при вычислении положения (по умолчанию: 10). Значение можно также указать с помощью HTML-параметра `data-offset` (см. *разд. 4.13.1*);
 - `target` — CSS-селектор, с помощью которого можно найти блок со ссылками. Значение допускается указывать с помощью HTML-параметра `data-target` (см. *разд. 4.13.1*);
 - `method` — метод вычисления позиции: "auto", "offset" или "position". Значение можно также указать с помощью HTML-параметра `data-method`.
- ◆ 'refresh' — вызывает обновление;
- ◆ 'dispose' — удаляет данные, связанные с компонентом.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `ScrollSpy`:

```
<Объект> = new bootstrap.ScrollSpy(<Элемент>[, <Объект с опциями>])
```

Пример создания объекта:

```

var scrollSpy = new bootstrap.ScrollSpy(document.body, {
  offset : 0,
  target : '#navabar1',
  method : 'auto'
});

```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект:

```
var scrollSpyInstance = bootstrap.ScrollSpy.getInstance(document.body);
```

Класс `ScrollSpy` содержит следующие методы:

- ◆ `refresh()` — вызывает обновление;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример:

```
scrollSpyInstance.refresh();
```

4.13.3. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчик для события `activate.bs.scrollspy`, которое возникает при смене активного элемента. Пример назначения обработчика для блока с полосой прокрутки с помощью jQuery:

```
$( '[data-spy="scroll"]' ).on('activate.bs.scrollspy', function(e) {
    console.log('activate.bs.scrollspy');
});
```

Пример для Bootstrap 5 без jQuery:

```
document.querySelectorAll('[data-spy="scroll"]').forEach(
    function(elem) {
        elem.addEventListener('activate.bs.scrollspy', function(e) {
            console.log('activate.bs.scrollspy');
        }, false);
    });
```

Пример управления компонентом `scrollspy` из программы и обработки событий приведен в листинге 4.12. Здесь мы используем компонент `navbar`, прикрепленный к низу окна, внутри которого расположены ссылки и компонент `dropdown`. Отслеживать прокрутку будем для всего окна веб-браузера.

Листинг 4.12. Компонент `scrollspy`: отслеживание прокрутки

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  <title>Компонент scrollspy: отслеживание прокрутки</title>
</head>
<body class="position-relative" style="padding-bottom: 56px">

<nav class="navbar navbar-light bg-light fixed-bottom" id="navbar1">
  <ul class="nav nav-pills">
```

```
<li class="nav-item">
  <a class="nav-link" href="#section1">Раздел 1</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#section2">Раздел 2</a>
</li>
<li class="nav-item dropdown dropup">
  <a class="nav-link dropdown-toggle" href="#" role="button"
    id="link1" data-toggle="dropdown" aria-haspopup="true"
    aria-expanded="false"> Разделы 3-5
  </a>
  <div class="dropdown-menu" aria-labelledby="link1">
    <a class="dropdown-item" href="#section3">Раздел 3</a>
    <a class="dropdown-item" href="#section4">Раздел 4</a>
    <a class="dropdown-item" href="#section5">Раздел 5</a>
  </div>
</li>
</ul>
</nav>

<div class="container bg-info" style="height: 400px" id="section1">
  <h1>Раздел 1</h1>
</div>
<div class="container bg-light" style="height: 400px" id="section2">
  <h1>Раздел 2</h1>
</div>
<div class="container bg-info" style="height: 600px" id="section3">
  <h1>Раздел 3</h1>
</div>
<div class="container bg-light" style="height: 600px" id="section4">
  <h1>Раздел 4</h1>
</div>
<div class="container bg-info" style="height: 800px" id="section5">
  <h1>Раздел 5</h1>
</div>

<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
  $('body').scrollspy({
    offset : 0,
    target : '#navbar1',
    method : 'auto'
  });
  $(window).on('activate.bs.scrollspy', function(e) {
    console.log('activate.bs.scrollspy');
  });
});
```

```
</script>
</body>
</html>
```

Изменим содержимое тега `<script>` из листинга 4.12 так, чтобы программа работала в Bootstrap 5 без jQuery:

```
new bootstrap.ScrollSpy(document.body, {
  offset : 0,
  target : '#navabar1',
  method : 'auto'
});
window.addEventListener('activate.bs.scrollspy', function(e) {
  console.log('activate.bs.scrollspy');
}, false);
```

4.14. Компонент *tooltip*: всплывающие подсказки

Компонент `tooltip` реализует всплывающую подсказку (рис. 4.31), отображаемую при наведении указателя мыши на ссылку, кнопку и другой элемент.

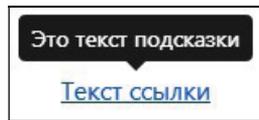


Рис. 4.31. Компонент `tooltip`: всплывающая подсказка

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.tooltip.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Tooltip.VERSION);
```

ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов `jquery.min.js`, `popper.min.js` и `bootstrap.min.js`. Вместо двух последних файлов можно использовать файл `bootstrap.bundle.min.js`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

4.14.1. Добавление всплывающей подсказки к элементу

Чтобы можно было отобразить всплывающую подсказку для ссылки при наведении указателя мыши или при получении ссылкой фокуса ввода, нужно для тега `<a>` добавить параметр `data-toggle` со значением `tooltip` и параметр `title` с текстом подсказки:

```
<a href="#" data-toggle="tooltip" title="Это текст подсказки">
  Текст ссылки</a>
```

Если сейчас навести указатель мыши на ссылку, то никакой всплывающей подсказки мы не увидим, т. к. инициализацию компонента нужно выполнить вручную. Для этого добавляем следующий код на jQuery:

```
$('#[data-toggle="tooltip"]').tooltip();
```

Код для Bootstrap 5 без jQuery:

```
document.querySelectorAll('[data-toggle="tooltip"]').forEach(
  function(elem) {
    new bootstrap.Tooltip(elem);
  });
```

Теперь при наведении указателя мыши над ссылкой отобразится черный прямоугольник со стрелкой и белым текстом подсказки из параметра `title`. Если над ссылкой недостаточно свободного пространства, то всплывающая подсказка отобразится под ссылкой. При выведении указателя мыши всплывающая подсказка автоматически скрывается.

Если текст ссылки переносится на новую строку, то всплывающая подсказка отобразится посередине строки, а не над ссылкой. Чтобы этого избежать, нужно запретить перенос текста внутри ссылки, добавив к тегу `<a>` стилевой класс `text-nowrap`:

```
<a href="#" data-toggle="tooltip" title="Это текст подсказки"
  class="text-nowrap">Текст ссылки</a>
```

Если параметр `title` не содержит значения, то всплывающая подсказка не отображается. Если к тегу `<a>` дополнительно добавить параметр `data-html` со значением `true`, то в качестве значения параметра `title` можно указать HTML-код. Отообразим текст внутри всплывающей подсказки подчеркнутым:

```
<a href="#" data-toggle="tooltip" data-html="true"
  title="<u>Текст подсказки</u>" class="text-nowrap">
  Текст ссылки</a>
```

Всплывающую подсказку можно добавить не только к ссылкам, но и к другим элементам, — например, к тегу `` или к кнопке:

```
<span data-toggle="tooltip" title="Это текст подсказки для span">
  Текст с подсказкой</span>
<button type="button" class="btn btn-info" data-toggle="tooltip"
  title="Это текст подсказки для button">
  Текст на кнопке
</button>
```

Если нужно добавить всплывающую подсказку для недоступной кнопки, то ее следует вложить в тег `` или `<div>` и уже для этих тегов добавить параметры `data-toggle` и `title`. Дополнительно можно добавить параметр `tabindex`, чтобы иметь доступ с клавиатуры:

```
<span class="d-inline-block" tabindex="0"
  data-toggle="tooltip" title="Текст подсказки">
  <button type="button" class="btn btn-info"
    style="pointer-events: none" disabled>
    Текст на кнопке
  </button>
</span>
```

Контейнер с подсказкой имеет стилевой класс `tooltip`, который задает абсолютное позиционирование и множество других настроек. Характеристики шрифта (по умолчанию белый цвет текста) и фона (по умолчанию черный цвет фона) подсказки задаются стилевым классом `tooltip-inner`. При наведении указателя на связанный элемент контейнер с подсказкой добавляется в конец документа, а при выведении указателя — удаляется из документа. Структура всплывающей подсказки в Bootstrap 4:

```
<div class="tooltip bs-tooltip-top" role="tooltip">
  <div class="arrow"></div>
  <div class="tooltip-inner">
    Текст подсказки
  </div>
</div>
```

В Bootstrap 5 класс `arrow` переименован в `tooltip-arrow`:

```
<div class="tooltip bs-tooltip-top" role="tooltip">
  <div class="tooltip-arrow"></div>
  <div class="tooltip-inner">
    Текст подсказки
  </div>
</div>
```

4.14.2. Местоположение всплывающей подсказки

По умолчанию всплывающая подсказка отображается над элементом или под ним, если свободного места сверху недостаточно. Чтобы явно указать местоположение подсказки (рис. 4.32), нужно добавить параметр `data-placement` с одним из следующих значений:

- ◆ `top` — сверху. К контейнеру добавляется стилевой класс `bs-tooltip-top`;
- ◆ `bottom` — снизу. К контейнеру добавляется стилевой класс `bs-tooltip-bottom`;
- ◆ `left` — слева. К контейнеру добавляется стилевой класс `bs-tooltip-left`;
- ◆ `right` — справа. К контейнеру добавляется стилевой класс `bs-tooltip-right`.
- ◆ `auto` — автоматическое определение местоположения.

Добавим всплывающие подсказки к кнопкам:

```
<div class="container mt-5 pt-5">
  <button type="button" class="btn btn-info" data-toggle="tooltip"
    data-placement="top" title="Подсказка сверху">
    Подсказка сверху
  </button>
  <button type="button" class="btn btn-info" data-toggle="tooltip"
    data-placement="bottom" title="Подсказка снизу">
    Подсказка снизу
  </button>
  <button type="button" class="btn btn-info" data-toggle="tooltip"
    data-placement="left" title="Подсказка слева">
    Подсказка слева
</div>
```

```

</button>
<button type="button" class="btn btn-info" data-toggle="tooltip"
  data-placement="right" title="Подсказка справа">
  Подсказка справа
</button>
</div>

```



Рис. 4.32. Местоположение всплывающей подсказки

4.14.3. Управление компонентом из программы

Для управления компонентом `tooltip` из программы на jQuery, нужно вызвать метод `tooltip()` и передать ему одно из следующих значений:

- ◆ без значения — выполняет инициализацию компонента со значениями опций по умолчанию. Без явного вызова метода всплывающие подсказки автоматически не добавятся. Пример инициализации сразу всех компонентов:

```
$('.[data-toggle="tooltip"]').tooltip();
```

Пример добавления всплывающей подсказки для одного элемента с идентификатором `myTooltip`:

```
$('#myTooltip').tooltip();
```

- ◆ объект с опциями — можно указать следующие основные свойства:
 - `trigger` — события, при которых отображается и скрывается всплывающая подсказка. Допустимые значения: `'hover'` (при наведении указателя мыши), `'focus'` (при получении фокуса ввода), `'click'` (при нажатии) или `'manual'` (ручной режим). Можно указать сразу несколько значений (но не `'manual'`) через пробел. Значение по умолчанию: `'hover focus'`. Значение можно также указать с помощью HTML-параметра `data-trigger`;
 - `placement` — местоположение подсказки. Допустимые значения: `'top'` (сверху; значение по умолчанию), `'bottom'` (снизу), `'left'` (слева), `'right'` (справа) или `'auto'` (автоматическое определение). Значение можно также указать с помощью HTML-параметра `data-placement` (см. *разд. 4.14.2*);
 - `html` — если указано значение `true`, то текст подсказки может содержать HTML-код. Значение по умолчанию: `false`. Значение можно также указать с помощью HTML-параметра `data-html` (см. *разд. 4.14.1*). Если надо ограничить использование тегов и параметров, то воспользуйтесь опциями `sanitize`, `sanitizeFn` и `whiteList`. Подробности вы найдете в документации;

- `title` — задает текст подсказки, если не указан HTML-параметр `title` или он имеет пустое значение. В качестве значения можно указать ссылку на функцию, возвращающую текст. Внутри функции указатель `this` ссылается на элемент, для которого отображается подсказка;
 - `animation` — если указано значение `true` (значение по умолчанию), то подсказка будет отображаться и скрываться с анимацией изменения прозрачности (добавляется стилевой класс `fade`). Значение `false` отключает анимацию. Значение можно также указать с помощью HTML-параметра `data-animation`;
 - `container` — позволяет задать элемент, внутрь которого будет добавляться контейнер с подсказкой. В качестве значения можно указать CSS-селектор, ссылку на элемент или `false` (значение по умолчанию). CSS-селектор можно также указать с помощью HTML-параметра `data-container`;
 - `selector` — добавляет всплывающую подсказку для элементов, соответствующих указанному CSS-селектору. Значение по умолчанию: `false`;
 - `delay` — задает задержку в миллисекундах перед отображением или сокрытием подсказки. Если указано число, то оно устанавливает задержку и для отображения, и для сокрытия подсказки (значение по умолчанию: 0). Если надо указать разные значения, то следует передать объект с двумя свойствами: `show` (задержка перед отображением) и `hide` (задержка перед сокрытием). Пример: `delay: { show: 300, hide: 100 }`. Значение можно также указать с помощью HTML-параметра `data-delay`;
 - `template` — позволяет изменить HTML-шаблон контейнера для всплывающей подсказки;
 - `offset` — задает смещение подсказки (значение по умолчанию: 0). В качестве значения можно указать число или строку с двумя числами через запятую. Смысл этих чисел зависит от местоположения подсказки. Если подсказка отображается сверху, то одно число задает смещение вправо на указанное количество пикселей. Если указаны два числа через запятую, то первое число задает смещение вправо, а второе — смещение вверх. Значение можно также указать с помощью HTML-параметра `data-offset`;
 - `fallbackPlacement` — если указано значение `'flip'` (по умолчанию), то при недостаточном количестве места местоположение подсказки поменяется на противоположное;
 - `boundary` — может принимать значения: `'viewport'`, `'window'`, `'scrollParent'` (значение по умолчанию) или ссылку на ограничивающий элемент. Значение можно также указать с помощью HTML-параметра `data-boundary`;
- ◆ `'toggle'` — отображает или скрывает подсказку;
 - ◆ `'show'` — отображает подсказку;
 - ◆ `'hide'` — скрывает подсказку;
 - ◆ `'enable'` — разрешает использование подсказки;
 - ◆ `'disable'` — запрещает использование подсказки;

- ◆ 'toggleEnabled' — разрешает или запрещает использование подсказки;
- ◆ 'update' — обновляет позицию подсказки;
- ◆ 'dispose' — скрывает подсказку и удаляет данные, связанные с компонентом.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Tooltip`:

```
<Объект> = new bootstrap.Tooltip(<Элемент>[, <Объект с опциями>])
```

Пример создания объектов и прикрепления их к элементам, имеющим параметр `data-toggle` со значением `tooltip`:

```
document.querySelectorAll('[data-toggle="tooltip"]').forEach(
  function(elem) {
    new bootstrap.Tooltip(elem);
  });
```

Пример создания объекта с указанием опций для кнопки с идентификатором `btn2`:

```
var btn2 = document.getElementById('btn2');
var btn2Tooltip = new bootstrap.Tooltip(btn2, {
  trigger      : 'manual',
  placement    : 'bottom',
  title        : 'Текст подсказки для кнопки 2',
  animation    : true,
  container    : document.body,
  delay        : { show: 300, hide: 100 },
  offset       : 80,
  fallbackPlacement : 'flip',
  boundary     : 'scrollParent'
});
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект:

```
var btn2 = document.getElementById('btn2');
var btn2Tooltip = bootstrap.Tooltip.getInstance(btn2);
```

Класс `Tooltip` содержит следующие методы:

- ◆ `toggle()` — отображает или скрывает подсказку;
- ◆ `show()` — отображает подсказку;
- ◆ `hide()` — скрывает подсказку;
- ◆ `enable()` — разрешает использование подсказки;
- ◆ `disable()` — запрещает использование подсказки;
- ◆ `toggleEnabled()` — разрешает или запрещает использование подсказки;
- ◆ `update()` — обновляет позицию подсказки;
- ◆ `dispose()` — скрывает подсказку и удаляет данные, связанные с компонентом.

Пример:

```
btn2Tooltip.toggle();
```

ОБРАТИТЕ ВНИМАНИЕ!

Методы запускают переключение состояния асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

4.14.4. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.tooltip` — событие перед отображением подсказки. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то подсказка отображена не будет:

```
var flagShow = false;
$('#btn2').on('show.bs.tooltip', function(e) {
  if (!flagShow) e.preventDefault();
});
```

Пример для Bootstrap 5 без jQuery:

```
var flagShow = false;
var btn2 = document.getElementById('btn2');
btn2.addEventListener('show.bs.tooltip', function(e) {
  if (!flagShow) e.preventDefault();
}, false);
```

- ◆ `inserted.bs.tooltip` — событие добавления HTML-кода подсказки в документ. Возникает после события `show.bs.tooltip`;
 - ◆ `shown.bs.tooltip` — событие после отображения подсказки;
 - ◆ `hide.bs.tooltip` — событие перед сокрытием подсказки. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то подсказка скрыта не будет:
- ```
var flagHide = false;
$('#btn2').on('hide.bs.tooltip', function(e) {
 if (!flagHide) e.preventDefault();
});
```
- ◆ `hidden.bs.tooltip` — событие после сокрытия подсказки.

Пример управления компонентом `tooltip` из программы и обработки событий приведен в листинге 4.13. Нажмите следующие клавиши:

- ◆ `<t>` — для отображения или сокрытия подсказки;
- ◆ `<s>` — для отображения подсказки;
- ◆ `<h>` — для сокрытия подсказки.

**Листинг 4.13. Компонент `tooltip`: всплывающие подсказки**

```
<!doctype html>
<html lang="ru">
<head>
```

```
<meta charset="utf-8">
<meta name="viewport"
 content="width=device-width, initial-scale=1, shrink-to-fit=no">
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
<title>Компонент tooltip: всплывающие подсказки</title>
</head>
<body>
<div class="container my-5 py-5" id="div1">
 <a href="#" data-toggle="tooltip" data-html="true"
 data-placement="top" data-animation="true"
 data-container="#div1" data-delay="100" data-offset="0"
 title="Текст подсказки" class="text-nowrap">
 Текст ссылки
 <button type="button" class="btn btn-primary ml-3" id="btn1">
 Кнопка 1
 </button>
 <button type="button" class="btn btn-primary ml-3" id="btn2">
 Кнопка 2
 </button>
</div>

<div class="container">
 <p>Нажмите клавиши t (toggle), s (show) или h (hide).</p>
 <p>Результат обработки событий см. в окне консоли.</p>
 <button type="button" class="btn btn-info" id="btnEnable">
 Enable</button>
 <button type="button" class="btn btn-info" id="btnDisable">
 Disable</button>
 <button type="button" class="btn btn-info" id="btnToggleEnabled">
 ToggleEnabled</button>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.bundle.min.js"></script>
<script>
$(function() {
 $('a[data-toggle="tooltip"]').tooltip();
 $('body').tooltip({
 title : 'Текст подсказки для кнопки 1',
 selector : '#btn1'
 });
 var btn2 = $('#btn2');
 btn2.tooltip({
 trigger : 'manual',
 placement : 'bottom',
 title : 'Текст подсказки для кнопки 2',
 animation : true,
 container : document.body,
 delay : { show: 300, hide: 100 },
 offset : 80,
 });
});
</script>
```

```

 fallbackPlacement : 'flip',
 boundary : 'scrollParent'
 });
 $('body').keydown(function(e) {
 if (e.which === 84) { // t
 $('#btn2').tooltip('toggle');
 }
 else if (e.which === 83) { // s
 $('#btn2').tooltip('show');
 }
 else if (e.which === 72) { // h
 $('#btn2').tooltip('hide');
 }
 });
 $('#btnEnable').click(function() {
 $('#btn2').tooltip('enable');
 });
 $('#btnDisable').click(function() {
 $('#btn2').tooltip('disable');
 });
 $('#btnToggleEnabled').click(function() {
 $('#btn2').tooltip('toggleEnabled');
 });
 btn2.on('show.bs.tooltip', function(e) {
 console.log('show.bs.tooltip');
 });
 btn2.on('inserted.bs.tooltip', function(e) {
 console.log('inserted.bs.tooltip');
 });
 btn2.on('shown.bs.tooltip', function(e) {
 console.log('shown.bs.tooltip');
 });
 btn2.on('hide.bs.tooltip', function(e) {
 console.log('hide.bs.tooltip');
 });
 btn2.on('hidden.bs.tooltip', function(e) {
 console.log('hidden.bs.tooltip');
 });
});
</script>
</body>
</html>

```

Изменим содержимое тега `<script>` из листинга 4.13 так, чтобы программа работала в Bootstrap 5 без jQuery:

```

document.querySelectorAll('a[data-toggle="tooltip"]').forEach(
 function(elem) {
 new bootstrap.Tooltip(elem);
 });

```

```
new bootstrap.Tooltip(document.body, {
 title : 'Текст подсказки для кнопки 1',
 selector : '#btn1'
});

var btn2 = document.getElementById('btn2');
var btn2Tooltip = new bootstrap.Tooltip(btn2, {
 trigger : 'manual',
 placement : 'bottom',
 title : 'Текст подсказки для кнопки 2',
 animation : true,
 container : document.body,
 delay : { show: 300, hide: 100 },
 offset : 80,
 fallbackPlacement : 'flip',
 boundary : 'scrollParent'
});

document.body.addEventListener('keydown', function(e) {
 if (e.keyCode === 84) { // t
 btn2Tooltip.toggle();
 }
 else if (e.keyCode === 83) { // s
 btn2Tooltip.show();
 }
 else if (e.keyCode === 72) { // h
 btn2Tooltip.hide();
 }
}, false);

var btnEnable = document.getElementById('btnEnable');
btnEnable.addEventListener('click', function() {
 btn2Tooltip.enable();
}, false);
var btnDisable = document.getElementById('btnDisable');
btnDisable.addEventListener('click', function() {
 btn2Tooltip.disable();
}, false);
var btnToggleEnabled = document.getElementById('btnToggleEnabled');
btnToggleEnabled.addEventListener('click', function() {
 btn2Tooltip.toggleEnabled();
}, false);

btn2.addEventListener('show.bs.tooltip', function(e) {
 console.log('show.bs.tooltip');
}, false);
btn2.addEventListener('inserted.bs.tooltip', function(e) {
 console.log('inserted.bs.tooltip');
}, false);
```

```
btn2.addEventListener('shown.bs.tooltip', function(e) {
 console.log('shown.bs.tooltip');
}, false);
btn2.addEventListener('hide.bs.tooltip', function(e) {
 console.log('hide.bs.tooltip');
}, false);
btn2.addEventListener('hidden.bs.tooltip', function(e) {
 console.log('hidden.bs.tooltip');
}, false);
```

## 4.15. Компонент *popover*: всплывающие информеры

Компонент `popover` расширяет компонент `tooltip` и реализует всплывающий информер, отображаемый и скрываемый при нажатии кнопки или другого элемента. В отличие от всплывающей подсказки, информер содержит два раздела: заголовок и тело, а также по умолчанию отображается справа от элемента (рис. 4.33).



Рис. 4.33. Компонент `popover`: всплывающий информер

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.popover.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Popover.VERSION);
```

### ОБРАТИТЕ ВНИМАНИЕ!

Для правильной работы компонента требуется подключение файлов `jquery.min.js`, `popper.min.js` и `bootstrap.min.js`. Вместо двух последних файлов можно использовать файл `bootstrap.bundle.min.js`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

### 4.15.1. Добавление всплывающего информера к элементу

Чтобы можно было отобразить всплывающий информер по нажатию кнопки, следует для тега `<button>` добавить параметр `data-toggle` со значением `popover`, параметр `title` с текстом для заголовка и параметр `data-content` с текстом для тела информера:

```
<button type="button" class="btn btn-info" data-toggle="popover"
 title="Заголовок информера"
 data-content="Текст внутри информера">
 Отобразить информер
</button>
```

Если сейчас нажать кнопку, то никакой всплывающий информер мы не увидим, т. к. инициализацию компонента нужно выполнить вручную. Для этого добавляем следующий код на jQuery:

```
$('.[data-toggle="popover"]').popover();
```

Код для Bootstrap 5 без jQuery:

```
document.querySelectorAll('[data-toggle="popover"]').forEach(
 function(elem) {
 new bootstrap.Popover(elem);
 });
```

Теперь при нажатии кнопки справа от нее отобразится прямоугольник со стрелкой, внутри которого имеются два раздела: заголовок и тело. Текст из параметра `title` является заголовком, который отображается в верхней части информера. Заголовок выделяется увеличенным размером шрифта и светло-серым фоном. Текст из параметра `data-content` отображается в нижней части информера на белом фоне. Если справа от кнопки недостаточно свободного пространства, то всплывающий информер отобразится слева.

При повторном нажатии кнопки всплывающий информер скрывается. Если нужно, чтобы информер скрывался автоматически при потере элементом фокуса ввода, то вместо тега `<button>` следует использовать тег `<a>`, для которого указан параметр `data-trigger` со значением `focus`:

```
<a class="btn btn-info text-white" role="button" tabindex="0"
 data-toggle="popover" data-trigger="focus"
 title="Заголовок информера"
 data-content="Текст внутри информера">
 Отобразить информер

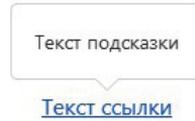
```

Если текст ссылки переносится на новую строку, то всплывающий информер отобразится посередине строки, а не над ссылкой. Чтобы этого избежать, нужно запретить перенос текста внутри ссылки, добавив к тегу `<a>` стилевой класс `text-nowrap`. Отобразим информер над ссылкой при наведении указателя мыши на ссылку и скроем его при выведении указателя:

```
<div style="height: 150px"></div>
<a href="#" data-toggle="popover" class="text-nowrap"
 data-trigger="hover focus" data-placement="top"
 title="Заголовок информера"
 data-content="Текст внутри информера">Текст ссылки
```

Если параметры `title` и `data-content` не содержат значения или не указаны, то информер не отображается вообще. Если параметр `title` не содержит значения или не указан, но существует параметр `data-content` со значением, то раздел заголовка отображаться не будет. Это позволяет превратить информер во всплывающую подсказку с черным текстом, отображенным на белом фоне (рис. 4.34):

```
<a href="#" data-toggle="popover" class="text-nowrap"
 data-trigger="hover focus" data-placement="top"
 data-content="Текст подсказки">Текст ссылки
```



**Рис. 4.34.** Информер в качестве всплывающей подсказки

Если к элементу добавить параметр `data-html` со значением `true`, то в качестве значения параметров `title` и `data-content` можно указать HTML-код. Отобразим текст заголовка информера полужирным шрифтом, а текст внутри тела — подчеркнутым:

```
<button type="button" class="btn btn-info" data-toggle="popover"
 data-html="true" title="Заголовок информера"
 data-content="<u>Текст внутри информера</u>">
```

Отобразить информер

```
</button>
```

Если нужно добавить всплывающий информер для недоступной кнопки, то ее следует вложить в тег `<span>` или `<div>` и уже для этих тегов добавить параметры `data-toggle`, `title` и `data-content`. Дополнительно можно добавить параметр `tabindex`, чтобы иметь доступ с клавиатуры:

```
<span class="d-inline-block" tabindex="0"
 data-toggle="popover" title="Заголовок информера"
 data-content="Текст внутри информера">
 <button type="button" class="btn btn-info"
 style="pointer-events: none" disabled>
 Текст на кнопке
 </button>

```

Контейнер с информером имеет стилевой класс `popover`, который задает абсолютное позиционирование и множество других настроек. Характеристики заголовка информера задаются стилевым классом `popover-header`, а тела — стилевым классом `popover-body`. По нажатию кнопки контейнер с информером добавляется в конец документа, а при повторном нажатии кнопки — удаляется из документа. Структура всплывающего информера в Bootstrap 4:

```
<div class="popover" role="tooltip">
 <div class="arrow"></div>
 <h3 class="popover-header"></h3>
 <div class="popover-body"></div>
</div>
```

В Bootstrap 5 класс `arrow` переименован в `popover-arrow`:

```
<div class="popover" role="tooltip">
 <div class="popover-arrow"></div>
 <h3 class="popover-header"></h3>
 <div class="popover-body"></div>
</div>
```

## 4.15.2. Местоположение всплывающего информера

По умолчанию всплывающий информер отображается справа от элемента или слева, если свободного места справа недостаточно. Чтобы явно указать местоположение информера, нужно добавить параметр `data-placement` с одним из следующих значений:

- ◆ `top` — сверху. К контейнеру добавляется стилевой класс `bs-popover-top`;
- ◆ `bottom` — снизу. К контейнеру добавляется стилевой класс `bs-popover-bottom`;
- ◆ `left` — слева. К контейнеру добавляется стилевой класс `bs-popover-left`;
- ◆ `right` — справа. К контейнеру добавляется стилевой класс `bs-popover-right`.
- ◆ `auto` — автоматическое определение местоположения.

Добавим всплывающие информеры к кнопкам:

```
<div class="container mt-5 pt-5">
 <button type="button" class="btn btn-info" data-toggle="popover"
 data-placement="top" title="Информер сверху"
 data-content="Текст внутри информера">
 Информер сверху
 </button>
 <button type="button" class="btn btn-info" data-toggle="popover"
 data-placement="bottom" title="Информер снизу"
 data-content="Текст внутри информера">
 Информер снизу
 </button>
 <button type="button" class="btn btn-info" data-toggle="popover"
 data-placement="left" title="Информер слева"
 data-content="Текст внутри информера">
 Информер слева
 </button>
 <button type="button" class="btn btn-info" data-toggle="popover"
 data-placement="right" title="Информер справа"
 data-content="Текст внутри информера">
 Информер справа
 </button>
</div>
```

## 4.15.3. Управление компонентом из программы

Для управления компонентом `popover` из программы на jQuery, нужно вызвать метод `popover()` и передать ему одно из следующих значений:

- ◆ без значения — выполняет инициализацию компонента со значениями опций по умолчанию. Без явного вызова метода всплывающие информеры автоматически не добавятся. Пример инициализации сразу всех компонентов:

```
$('.[data-toggle="popover"]').popover();
```

Пример добавления всплывающего информера для одного элемента с идентификатором `myPopover`:

```
$('#myPopover').popover();
```

- ◆ объект с опциями — можно указать следующие основные свойства:
- `trigger` — события, при которых отображается и скрывается всплывающий информер. Допустимые значения: `'hover'` (при наведении указателя мыши), `'focus'` (при получении фокуса ввода), `'click'` (при нажатии — значение по умолчанию) или `'manual'` (ручной режим). Допускается указание сразу несколько значений (но не `'manual'`) через пробел. Значение можно также задать с помощью HTML-параметра `data-trigger`;
  - `placement` — местоположение информера. Допустимые значения: `'top'` (сверху), `'bottom'` (снизу), `'left'` (слева), `'right'` (справа — значение по умолчанию) или `'auto'` (автоматическое определение). Значение можно также указать с помощью HTML-параметра `data-placement` (см. *разд. 4.15.2*);
  - `html` — если указано значение `true`, то текст заголовка и тела информера может содержать HTML-код. Значение по умолчанию: `false`. Значение можно также указать с помощью HTML-параметра `data-html` (см. *разд. 4.15.1*). Если нужно ограничить использование тегов и параметров, то воспользуйтесь опциями `sanitize`, `sanitizeFn` и `whiteList`. Подробности вы найдете в документации;
  - `title` — задает текст для заголовка информера, если не указан HTML-параметр `title` или он имеет пустое значение. В качестве значения можно указать ссылку на функцию, возвращающую текст. Внутри функции указатель `this` ссылается на элемент, для которого отображается информер;
  - `content` — задает текст для тела информера, если не указан HTML-параметр `data-content` или он имеет пустое значение. В качестве значения можно указать ссылку на функцию, возвращающую текст. Внутри функции указатель `this` ссылается на элемент, для которого отображается информер;
  - `animation` — если указано значение `true` (значение по умолчанию), то информер будет отображаться и скрываться с анимацией изменения прозрачности (добавляется стилевой класс `fade`). Значение `false` отключает анимацию. Значение можно также указать с помощью HTML-параметра `data-animation`;
  - `container` — позволяет задать элемент, внутрь которого будет добавляться контейнер с информером. В качестве значения можно указать CSS-селектор, ссылку на элемент или `false` (значение по умолчанию). CSS-селектор можно также указать с помощью HTML-параметра `data-container`;
  - `selector` — добавляет всплывающий информер для элементов, соответствующих указанному CSS-селектору. Значение по умолчанию: `false`;
  - `delay` — задает задержку в миллисекундах перед отображением или сокрытием информера. Если указано число, то оно устанавливает задержку и для отображения, и для сокрытия информера (значение по умолчанию: 0). Если нужно указать разные значения, то следует передать объект с двумя свойствами: `show` (задержка перед отображением) и `hide` (задержка перед сокрытием). Пример: `delay: { show: 300, hide: 100 }`. Значение можно также указать с помощью HTML-параметра `data-delay`;

- `template` — позволяет изменить HTML-шаблон контейнера для всплывающего информера;
- `offset` — задает смещение информера (значение по умолчанию: 0). В качестве значения можно указать число или строку с двумя числами через запятую. Смысл этих чисел зависит от местоположения информера. Если информер отображается сверху, то одно число задает смещение вправо на указанное количество пикселей. Если указаны два числа через запятую, то первое число задает смещение вправо, а второе — смещение вверх. Значение можно также указать с помощью HTML-параметра `data-offset`;
- `fallbackPlacement` — если указано значение `'flip'` (по умолчанию), то при недостаточном количестве места местоположение информера поменяется на противоположное;
- `boundary` — может принимать значения: `'viewport'`, `'window'`, `'scrollParent'` (значение по умолчанию) или ссылку на ограничивающий элемент. Значение можно также указать с помощью HTML-параметра `data-boundary`;
- ◆ `'toggle'` — отображает или скрывает информер;
- ◆ `'show'` — отображает информер;
- ◆ `'hide'` — скрывает информер;
- ◆ `'enable'` — разрешает использование информера;
- ◆ `'disable'` — запрещает использование информера;
- ◆ `'toggleEnabled'` — разрешает или запрещает использование информера;
- ◆ `'update'` — обновляет позицию информера;
- ◆ `'dispose'` — скрывает информер и удаляет данные, связанные с компонентом.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Popover`:

```
<Объект> = new bootstrap.Popover(<Элемент>[, <Объект с опциями>])
```

Пример создания объектов и прикрепления их к элементам, имеющим параметр `data-toggle` со значением `popover`:

```
document.querySelectorAll('[data-toggle="popover"]').forEach(
 function(elem) {
 new bootstrap.Popover(elem);
 });
```

Пример создания объекта с указанием опций для кнопки с идентификатором `btn2`:

```
var btn2 = document.getElementById('btn2');
var btn2Popover = new bootstrap.Popover(btn2, {
 trigger : 'manual',
 placement : 'right',
 title : 'Текст заголовка для кнопки 2',
 content : 'Текст внутри информера',
 animation : true,
 container : document.body,
```

```

delay : { show: 300, hide: 100 },
offset : 20,
fallbackPlacement : 'flip',
boundary : 'scrollParent'
});

```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект:

```

var btn2 = document.getElementById('btn2');
var btn2Popover = bootstrap.Popover.getInstance(btn2);

```

Класс `Popover` содержит следующие методы:

- ◆ `toggle()` — отображает или скрывает информер;
- ◆ `show()` — отображает информер;
- ◆ `hide()` — скрывает информер;
- ◆ `enable()` — разрешает использование информера;
- ◆ `disable()` — запрещает использование информера;
- ◆ `toggleEnabled()` — разрешает или запрещает использование информера;
- ◆ `update()` — обновляет позицию информера;
- ◆ `dispose()` — скрывает информер и удаляет данные, связанные с компонентом.

Пример:

```
btn2Popover.toggle();
```

#### **ОБРАТИТЕ ВНИМАНИЕ!**

Методы запускают переключение состояния асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

## 4.15.4. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.popover` — событие перед отображением информера. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то информер отображен не будет:

```

var flagShow = false;
$('#btn2').on('show.bs.popover', function(e) {
 if (!flagShow) e.preventDefault();
});

```

Пример для Bootstrap 5 без jQuery:

```

var flagShow = false;
var btn2 = document.getElementById('btn2');
btn2.addEventListener('show.bs.popover', function(e) {
 if (!flagShow) e.preventDefault();
}, false);

```

- ◆ `inserted.bs.popover` — событие добавления HTML-кода информера в документ. Возникает после события `show.bs.popover`;
- ◆ `shown.bs.popover` — событие после отображения информера;
- ◆ `hide.bs.popover` — событие перед сокрытием информера. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то информер скрыт не будет:
 

```
var flagHide = false;
$('#btn2').on('hide.bs.popover', function(e) {
 if (!flagHide) e.preventDefault();
});
```
- ◆ `hidden.bs.popover` — событие после сокрытия информера.

Пример управления компонентом `popover` из программы и обработки событий приведен в листинге 4.14. Нажмите следующие клавиши:

- ◆ `<t>` — для отображения или сокрытия информера;
- ◆ `<s>` — для отображения информера;
- ◆ `<h>` — для сокрытия информера.

#### Листинг 4.14. Компонент `popover`: всплывающие информеры

```
<!doctype html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <meta name="viewport"
 content="width=device-width, initial-scale=1, shrink-to-fit=no">
 <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
 <title>Компонент popover: всплывающие информеры</title>
</head>
<body>
<div class="container my-5 py-5" id="div1">
 <a href="#" data-toggle="popover" class="text-nowrap"
 data-trigger="hover focus" data-html="true"
 data-placement="top" data-animation="true"
 data-container="#div1" data-delay="100" data-offset="0"
 title="Заголовок информера"
 data-content="<u>Текст внутри информера</u>">
 Текст ссылки
 <button type="button" class="btn btn-primary ml-3" id="btn1">
 Кнопка 1
 </button>
 <button type="button" class="btn btn-primary ml-3" id="btn2">
 Кнопка 2
 </button>
</div>
```

```

<div class="container">
 <p>Нажмите клавиши t (toggle), s (show) или h (hide).</p>
 <p>Результат обработки событий см. в окне консоли.</p>
 <button type="button" class="btn btn-info" id="btnEnable">
 Enable</button>
 <button type="button" class="btn btn-info" id="btnDisable">
 Disable</button>
 <button type="button" class="btn btn-info" id="btnToggleEnabled">
 ToggleEnabled</button>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.bundle.min.js"></script>
<script>
$(function() {
 $('a[data-toggle="popover"]').popover();
 $('body').popover({
 title : function() {
 return 'Текст заголовка для кнопки с id=' + this.id;
 },
 content : function() {
 return 'Текст внутри информера';
 },
 placement : 'bottom',
 selector : '#btn1'
 });
 var btn2 = $('#btn2');
 btn2.popover({
 trigger : 'manual',
 placement : 'right',
 title : 'Текст заголовка для кнопки 2',
 content : 'Текст внутри информера',
 animation : true,
 container : document.body,
 delay : { show: 300, hide: 100 },
 offset : 20,
 fallbackPlacement : 'flip',
 boundary : 'scrollParent'
 });
 $('body').keydown(function(e) {
 if (e.which === 84) { // t
 $('#btn2').popover('toggle');
 }
 else if (e.which === 83) { // s
 $('#btn2').popover('show');
 }
 else if (e.which === 72) { // h
 $('#btn2').popover('hide');
 }
 });
});

```

```
$('#btnEnable').click(function() {
 $('#btn2').popover('enable');
});
$('#btnDisable').click(function() {
 $('#btn2').popover('disable');
});
$('#btnToggleEnabled').click(function() {
 $('#btn2').popover('toggleEnabled');
});
btn2.on('show.bs.popover', function(e) {
 console.log('show.bs.popover');
});
btn2.on('inserted.bs.popover', function(e) {
 console.log('inserted.bs.popover');
});
btn2.on('shown.bs.popover', function(e) {
 console.log('shown.bs.popover');
});
btn2.on('hide.bs.popover', function(e) {
 console.log('hide.bs.popover');
});
btn2.on('hidden.bs.popover', function(e) {
 console.log('hidden.bs.popover');
});
});
</script>
</body>
</html>
```

**Изменим содержимое тега `<script>` из листинга 4.14 так, чтобы программа работала в Bootstrap 5 без jQuery:**

```
document.querySelectorAll('a[data-toggle="popover"]').forEach(
 function(elem) {
 new bootstrap.Popover(elem);
 });
new bootstrap.Popover(document.body, {
 title : function() {
 return 'Текст заголовка для кнопки с id=' + this.id;
 },
 content : function() {
 return 'Текст внутри информера';
 },
 placement : 'bottom',
 selector : '#btn1'
});

var btn2 = document.getElementById('btn2');
var btn2Popover = new bootstrap.Popover(btn2, {
 trigger : 'manual',
```

```
 placement : 'right',
 title : 'Текст заголовка для кнопки 2',
 content : 'Текст внутри информера',
 animation : true,
 container : document.body,
 delay : { show: 300, hide: 100 },
 offset : 20,
 fallbackPlacement : 'flip',
 boundary : 'scrollParent'
 });
document.body.addEventListener('keydown', function(e) {
 if (e.keyCode === 84) { // t
 btn2Popover.toggle();
 }
 else if (e.keyCode === 83) { // s
 btn2Popover.show();
 }
 else if (e.keyCode === 72) { // h
 btn2Popover.hide();
 }
}, false);

var btnEnable = document.getElementById('btnEnable');
btnEnable.addEventListener('click', function() {
 btn2Popover.enable();
}, false);
var btnDisable = document.getElementById('btnDisable');
btnDisable.addEventListener('click', function() {
 btn2Popover.disable();
}, false);
var btnToggleEnabled = document.getElementById('btnToggleEnabled');
btnToggleEnabled.addEventListener('click', function() {
 btn2Popover.toggleEnabled();
}, false);

btn2.addEventListener('show.bs.popover', function(e) {
 console.log('show.bs.popover');
}, false);
btn2.addEventListener('inserted.bs.popover', function(e) {
 console.log('inserted.bs.popover');
}, false);
btn2.addEventListener('shown.bs.popover', function(e) {
 console.log('shown.bs.popover');
}, false);
btn2.addEventListener('hide.bs.popover', function(e) {
 console.log('hide.bs.popover');
}, false);
btn2.addEventListener('hidden.bs.popover', function(e) {
 console.log('hidden.bs.popover');
}, false);
```

## 4.16. Компонент *toast*: всплывающие уведомления

Компонент `toast` реализует всплывающие окна с уведомлениями (рис. 4.35), которые содержат два раздела: заголовок и тело. По умолчанию уведомления автоматически скрываются через 500 миллисекунд, но существует возможность либо увеличить время отображения, либо добавить в заголовок кнопку **Заккрыть**, по нажатию на которую уведомление скрывается.

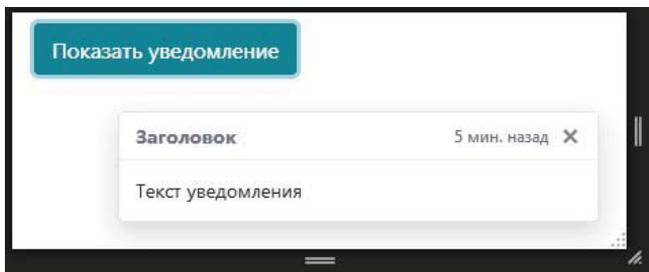


Рис. 4.35. Компонент `toast`: всплывающее уведомление

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.toast.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Toast.VERSION);
```

### **ОБРАТИТЕ ВНИМАНИЕ!**

Для правильной работы компонента требуется подключение файлов `jquery.min.js` и `bootstrap.min.js` (или `bootstrap.bundle.min.js`), а также добавление к кнопке **Заккрыть** параметра `data-dismiss` со значением `toast`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

### 4.16.1. Создание и отображение уведомления

Чтобы создать уведомление, нужно выполнить следующие действия:

1. Создать родительский контейнер и добавить для него стилевой класс `toast`. Этот класс задает максимальную ширину в 350 пикселей, размер шрифта, цвет фона (белый, частично прозрачный), добавляет рамку и тень, а также делает уведомление прозрачным, но не скрытым. Чтобы уведомление не занимало место, надо добавить стилевой класс `hide` (`display: none`). При отображении уведомления вместо класса `hide` будет добавлен класс `show`, который делает уведомление полностью непрозрачным и видимым (`display: block`). Для различных вспомогательных программ надо добавить параметр `role` со значением `alert` или `status`, параметр `aria-live` со значением `assertive` или `polite`, а также параметр `aria-atomic` со значением `true`.
2. Внутри родительского контейнера вложить контейнер со стилевым классом `toast-header`, который описывает заголовок уведомления. Этот класс делает за-

головок flex-контейнером, задает цвет текста и фона, а также добавляет разделительную линию снизу.

3. Внутри родительского контейнера вложить контейнер со стилевым классом `toast-body`, который описывает тело уведомления.

По умолчанию уведомления автоматически скрываются через 500 миллисекунд. Чтобы указать другое значение, следует к родительскому контейнеру добавить параметр `data-delay` и в качестве значения задать нужное число миллисекунд.

Если нужно запретить автоматическое сокрытие уведомления, то надо к родительскому контейнеру добавить параметр `data-autohide` со значением `false`.

Чтобы пользователь имел возможность закрыть уведомление, можно в заголовок добавить кнопку **Закреть**, по нажатию на которую уведомление скрывается. Для этого к тегу `<button>` добавляем стилевой класс `close`, а также параметр `data-dismiss` со значением `toast` и параметр `aria-label` с описанием кнопки, поскольку вместо надписи на кнопке отобразим крестик (`&times;`).

Обычно уведомления показываются либо в правом верхнем углу окна, либо в его правом нижнем углу. Чтобы это реализовать, добавляем уведомления в контейнер с фиксированным позиционированием и привязкой к нужным сторонам окна. Если добавить в этот контейнер несколько уведомлений, то они будут расположены друг под другом с отступом.

Создадим уведомление, которое автоматически скрывается через пять секунд или по нажатию пользователем кнопки **Закреть** в заголовке:

```
<div style="position: fixed; bottom: 20px; right: 20px; width: 350px">
 <div class="toast hide" id="toast1"
 data-autohide="true" data-delay="5000"
 role="alert" aria-live="assertive" aria-atomic="true">
 <div class="toast-header">
 <strong class="mr-auto">Заголовок
 <small>5 мин. назад</small>
 <button type="button" class="ml-2 mb-1 close"
 data-dismiss="toast" aria-label="Закреть">
 ×
 </button>
 </div>
 <div class="toast-body">
 Текст уведомления
 </div>
 </div>
</div>
```

Заголовок уведомления является flex-контейнером, поэтому для размещения элементов мы можем пользоваться стилевыми классами из *разд. 2.1* или классами для вставки внешних отступов (см. *разд. 1.4.1*). В нашем примере мы текст заголовка выравниваем по левому краю, а вспомогательный текст и кнопку **Закреть** — по правому краю, добавив к тексту заголовка стилевой класс `mr-auto`.

Если мы сейчас запустим этот код в веб-браузере, то ничего не увидим, ведь уведомление по умолчанию скрыто. Чтобы иметь возможность отображать уведомление, добавим кнопку:

```
<div class="container mt-2">
 <button type="button" class="btn btn-info" id="btnShow">
 Показать уведомление</button>
</div>
```

Теперь при нажатии кнопки из программы на jQuery вызовем метод `toast()` со значением `'show'`, который отображает уведомление:

```
$('#btnShow').click(function() {
 $('#toast1').toast('show');
});
```

Код для Bootstrap 5 без jQuery:

```
var btnShow = document.getElementById('btnShow');
btnShow.addEventListener('click', function() {
 var toast1 = document.getElementById('toast1');
 var toast1Instance = bootstrap.Toast.getInstance(toast1);
 if (!toast1Instance) {
 toast1Instance = new bootstrap.Toast(toast1);
 }
 toast1Instance.show();
}, false);
```

По нажатию кнопки **Показать уведомление** окно с уведомлением отобразится в правом нижнем углу окна веб-браузера. Через пять секунд уведомление будет автоматически скрыто.

## 4.16.2. Управление компонентом из программы

Для управления компонентом `toast` из программы на jQuery нужно вызвать метод `toast()` и передать ему одно из следующих значений:

- ◆ объект с опциями — можно указать такие основные свойства:
  - `delay` — задает время задержки перед сокрытием уведомления в миллисекундах. По умолчанию: 500 миллисекунд. Значение можно также задать с помощью HTML-параметра `data-delay`;
  - `autohide` — если задано значение `true` (значение по умолчанию), то уведомление будет автоматически скрываться. Значение `false` отключает эту возможность. Значение можно также задать с помощью HTML-параметра `data-autohide`;
  - `animation` — если указано значение `true` (значение по умолчанию), то уведомление будет отображаться и скрываться с анимацией изменения прозрачности (добавляется стилевой класс `fade`). Значение `false` отключает анимацию. Значение можно также указать с помощью HTML-параметра `data-animation`;
- ◆ `'show'` — отображает уведомление;

- ◆ 'hide' — скрывает уведомление;
- ◆ 'dispose' — скрывает уведомление и удаляет данные, связанные с компонентом.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Toast`:

```
<Объект> = new bootstrap.Toast(<Элемент>[, <Объект с опциями>])
```

Пример создания объекта:

```
var toast1 = document.getElementById('toast1');
var toast1Instance = new bootstrap.Toast(toast1);
```

Пример создания объекта с указанием опций:

```
var toast2 = document.getElementById('toast2');
var toast2Instance = new bootstrap.Toast(toast2, {
 animation : false,
 autohide : true,
 delay : 10000
});
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект и создания объекта при отсутствии ссылки:

```
var toast1 = document.getElementById('toast1');
var toast1Instance = bootstrap.Toast.getInstance(toast1);
if (!toast1Instance) {
 toast1Instance = new bootstrap.Toast(toast1);
}
```

Класс `Toast` содержит следующие методы:

- ◆ `show()` — отображает уведомление;
- ◆ `hide()` — скрывает уведомление;
- ◆ `dispose()` — скрывает уведомление и удаляет данные, связанные с компонентом.

Пример:

```
toast1Instance.show();
```

#### **ОБРАТИТЕ ВНИМАНИЕ!**

Методы запускают переключение состояния асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

### 4.16.3. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.toast` — событие перед отображением уведомления. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то уведомление отображено не будет:

```
var flagShow = false;
$('#toast1').on('show.bs.toast', function(e) {
```

```
 if (!flagShow) e.preventDefault();
 });
```

### Пример для Bootstrap 5 без jQuery:

```
var flagShow = false;
var toast1 = document.getElementById('toast1');
toast1.addEventListener('show.bs.toast', function(e) {
 if (!flagShow) e.preventDefault();
}, false);
```

- ◆ `shown.bs.toast` — событие после отображения уведомления;
- ◆ `hide.bs.toast` — событие перед сокрытием уведомления. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то уведомление скрыто не будет:

```
var flagHide = false;
$('#toast1').on('hide.bs.toast', function(e) {
 if (!flagHide) e.preventDefault();
});
```

- ◆ `hidden.bs.toast` — событие после сокрытия уведомления.

Пример управления компонентом `toast` из программы и обработки событий приведен в листинге 4.15.

#### Листинг 4.15. Компонент `toast`: всплывающие уведомления

```
<!doctype html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <meta name="viewport"
 content="width=device-width, initial-scale=1, shrink-to-fit=no">
 <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
 <title>Компонент toast: всплывающие уведомления</title>
</head>
<body>
<div style="position: fixed; bottom: 20px; right: 20px; width: 350px">
 <div class="toast hide" id="toast1" data-animation="true"
 data-autohide="true" data-delay="5000"
 role="alert" aria-live="assertive" aria-atomic="true">
 <div class="toast-header">
 <strong class="mr-auto">Заголовок 1
 <small>5 мин. назад</small>
 </div>
 <div class="toast-body">
 Текст уведомления 1
 </div>
 </div>

 <div class="toast hide" id="toast2"
 role="alert" aria-live="assertive" aria-atomic="true">
```

```

<div class="toast-header">

 <strong class="ml-2 mr-auto">Заголовок 2
 <small>1 мин. назад</small>
 <button type="button" class="ml-2 mb-1 close"
 data-dismiss="toast" aria-label="Закрыть">
 ×
 </button>
</div>
<div class="toast-body">
 Текст уведомления 2
</div>
</div>
</div>

<div class="container my-2">
 <button type="button" class="btn btn-primary" id="btnShow">
 Показать уведомление 1</button>
 <button type="button" class="btn btn-primary" id="btnHide">
 Скрыть уведомление 1</button>
</div>
<div class="container">
 <button type="button" class="btn btn-primary" id="btnShow2">
 Показать уведомление 2</button>
 <button type="button" class="btn btn-primary" id="btnHide2">
 Скрыть уведомление 2</button>
 <p>Результат обработки событий см. в окне консоли.</p>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
 var toast2 = $('#toast2');
 toast2.toast({
 animation : false,
 autohide : true,
 delay : 10000
 });

 $('#btnShow').click(function() {
 $('#toast1').toast('show');
 });
 $('#btnShow2').click(function() {
 $('#toast2').toast('show');
 });
 $('#btnHide').click(function() {
 $('#toast1').toast('hide');
 });
});

```

```
$('#btnHide2').click(function() {
 $('#toast2').toast('hide');
});

toast2.on('show.bs.toast', function(e) {
 console.log('show.bs.toast');
});
toast2.on('shown.bs.toast', function(e) {
 console.log('shown.bs.toast');
});
toast2.on('hide.bs.toast', function(e) {
 console.log('hide.bs.toast');
});
toast2.on('hidden.bs.toast', function(e) {
 console.log('hidden.bs.toast');
});
});
</script>
</body>
</html>
```

**Изменим содержимое тега `<script>` из листинга 4.15 так, чтобы программа работала в Bootstrap 5 без jQuery:**

```
var toast1 = document.getElementById('toast1');
var toast1Instance = new bootstrap.Toast(toast1);

var toast2 = document.getElementById('toast2');
var toast2Instance = new bootstrap.Toast(toast2, {
 animation : false,
 autohide : true,
 delay : 10000
});

var btnShow = document.getElementById('btnShow');
btnShow.addEventListener('click', function() {
 toast1Instance.show();
}, false);
var btnShow2 = document.getElementById('btnShow2');
btnShow2.addEventListener('click', function() {
 toast2Instance.show();
}, false);
var btnHide = document.getElementById('btnHide');
btnHide.addEventListener('click', function() {
 toast1Instance.hide();
}, false);
var btnHide2 = document.getElementById('btnHide2');
btnHide2.addEventListener('click', function() {
 toast2Instance.hide();
}, false);
```

```
toast2.addEventListener('show.bs.toast', function(e) {
 console.log('show.bs.toast');
}, false);
toast2.addEventListener('shown.bs.toast', function(e) {
 console.log('shown.bs.toast');
}, false);
toast2.addEventListener('hide.bs.toast', function(e) {
 console.log('hide.bs.toast');
}, false);
toast2.addEventListener('hidden.bs.toast', function(e) {
 console.log('hidden.bs.toast');
}, false);
```

## 4.17. Компонент *modal*: модальные диалоговые окна

Компонент `modal` позволяет создавать диалоговые окна (рис. 4.36), которые являются *модальными*: то есть пока пользователь не закроет окно, он не сможет взаимодействовать с элементами внутри документа. Закрыть окно можно с помощью кнопки с параметром `data-dismiss`, имеющим значение `modal`, а также щелчком левой кнопкой мыши вне диалогового окна.

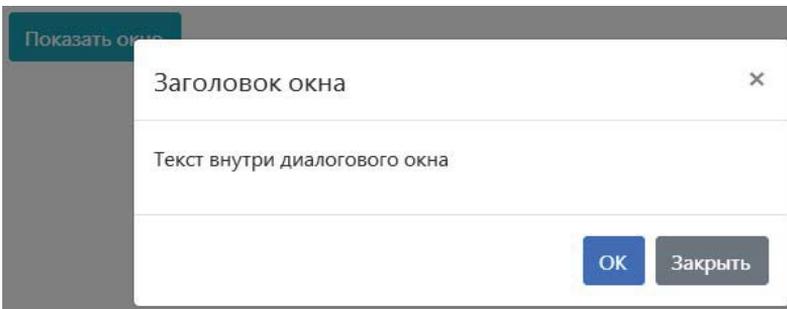


Рис. 4.36. Компонент `modal`: модальное диалоговое окно

Выведем в окно консоли версию используемого компонента с помощью jQuery:

```
console.log($.fn.modal.Constructor.VERSION);
```

Пример получения версии в Bootstrap 5 без jQuery:

```
console.log(bootstrap.Modal.VERSION);
```

### **ОБРАТИТЕ ВНИМАНИЕ!**

Для правильной работы компонента требуется подключение файлов `jquery.min.js` и `bootstrap.min.js` (или `bootstrap.bundle.min.js`), а также добавление к кнопке **Закрыть** параметра `data-dismiss` со значением `modal`. В Bootstrap 5 можно не подключать файл `jquery.min.js`, если не используется интерфейс доступа через jQuery.

### 4.17.1. Создание окна

Для создания диалогового окна нужно выполнить следующие действия:

1. Создать родительский контейнер и добавить к нему стилевой класс `modal`, а также параметры: `tabindex` — со значением `-1`, `aria-hidden` — со значением `true`, `aria-labelledby` — с идентификатором заголовка окна и `id` — с уникальным идентификатором. Начиная с версии 4.5.2, параметр `role` со значением `dialog` при отображении окна добавляется автоматически, а при его сокрытии — удаляется. Стилевой класс `modal` устанавливает фиксированное позиционирование для контейнера с привязкой к левому верхнему углу окна, задает ширину и высоту контейнера, равными `100%`, а также скрывает окно. По умолчанию диалоговое окно отображается и скрывается сразу, без анимации. Если надо, чтобы диалоговое окно отображалось и скрывалось с анимацией смещения, то следует дополнительно указать стилевой класс `fade`.
2. Внутри родительского контейнера вкладываем тег `<div>` со стилевым классом `modal-dialog` и параметром `role` со значением `document`. Начиная с точки останова `sm`, для этого элемента устанавливается максимальная ширина, равная `500` пикселей, внешний отступ сверху и снизу, а также горизонтальное выравнивание по центру.
3. Внутри элемента со стилевым классом `modal-dialog` вкладываем тег `<div>` со стилевым классом `modal-content`, который задает относительное позиционирование, белый цвет фона и рамку со скругленными углами, а также преобразует элемент во `flex`-контейнер с вертикальным размещением элементов.
4. Внутри элемента со стилевым классом `modal-content` вкладываем тег `<div>` со стилевым классом `modal-header`, который преобразует элемент во `flex`-контейнер с горизонтальным размещением элементов, а также добавляет внутренние отступы и границу снизу. Этот элемент описывает раздел с заголовком диалогового окна. Для тега с текстом заголовка следует добавить стилевой класс `modal-title`. Чтобы пользователь имел возможность закрыть окно, можно в раздел с заголовком добавить кнопку **Заккрыть**, по нажатию на которую окно скрывается. Для этого к тегу `<button>` добавляем стилевой класс `close`, а также параметр `data-dismiss` со значением `modal` и параметр `aria-label` с описанием кнопки, поскольку вместо надписи на кнопке отобразим крестик (`&times;`).
5. Внутри элемента со стилевым классом `modal-content` вкладываем тег `<div>` со стилевым классом `modal-body`, который описывает раздел с основным содержанием диалогового окна. В этот контейнер можно просто добавлять произвольные элементы или вложить в него базовый контейнер со стилевым классом `container-fluid`, содержащим систему сеток библиотеки `Bootstrap`. Если содержимое диалогового окна будет слишком велико, то появится вертикальная полоса прокрутки.
6. Внутри элемента со стилевым классом `modal-content` вкладываем тег `<div>` со стилевым классом `modal-footer`, который описывает раздел с «подвалом» диалогового окна, внутри которого обычно размещаются командные кнопки. Раздел

является flex-контейнером с горизонтальным размещением элементов и выравниванием по правому краю. Чтобы произвести выравнивание по левому краю или по центру, следует дополнительно добавить стилевой класс `justify-content-start` или `justify-content-center` соответственно. Если элементы не помещаются на строке, то они будут переноситься на новую строку. Чтобы пользователь имел возможность закрыть окно, можно добавить кнопку **Заккрыть**, по нажатию на которую окно скрывается. Для этого к тегу `<button>` добавляем параметр `data-dismiss` со значением `modal`. Обработчик нажатия кнопки будет добавлен автоматически.

### Пример:

```
<div class="modal fade" tabindex="-1" id="dialog1"
 aria-labelledby="dialog1-title" aria-hidden="true">
 <div class="modal-dialog" role="document">
 <div class="modal-content">
 <div class="modal-header">
 <h5 class="modal-title" id="dialog1-title">Заголовок окна</h5>
 <button type="button" class="close"
 data-dismiss="modal" aria-label="Заккрыть">
 ×
 </button>
 </div>
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-primary">OK</button>
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Заккрыть</button>
 </div>
 </div>
 </div>
</div>
```

## 4.17.2. Отображение и закрытие окна

Если мы сейчас запустим код из *разд. 4.17.1* в веб-браузере, то ничего не увидим, — ведь диалоговое окно по умолчанию скрыто. Чтобы иметь возможность отобразить окно, нужно добавить кнопку с параметром `data-toggle`, имеющим значение `modal`, и параметром `data-target`, содержащим CSS-селектор, с помощью которого можно найти диалоговое окно. В этом случае обработчик нажатия кнопки будет назначен автоматически:

```
<div class="container mt-2">
 <button type="button" class="btn btn-info"
 data-toggle="modal" data-target="#dialog1">
 Показать окно</button>
</div>
```

Если вместо кнопки используется ссылка, то CSS-селектор, с помощью которого можно найти диалоговое окно, указывается в параметре `href`, а не в параметре `data-target`:

```
<div class="container mt-2">
 <a class="btn btn-info" href="#dialog1" data-toggle="modal"
 role="button">Показать окно
</div>
```

По нажатию кнопки **Показать окно** диалоговое окно будет отображено на экране с анимацией смещения (окно как бы выезжает сверху). Если убрать стилевой класс `fade`, то окно будет отображаться сразу, без анимации.

При открытом окне поверх содержимого документа, но ниже окна, отображается полупрозрачный серый элемент со стилевым классом `modal-backdrop`, который добавляется в конец документа. После закрытия окна этот элемент удаляется из документа. Чтобы отключить добавление этого элемента, следует к контейнеру с классом `modal` добавить параметр `data-backdrop` со значением `false`.

При открытии диалогового окна к тегу `<body>` добавляется стилевой класс `modal-open`, который блокирует прокрутку содержимого документа. После закрытия окна этот класс удаляется.

Закрыть окно можно следующими способами:

- ◆ нажать кнопку с крестиком внутри заголовка окна;
- ◆ нажать кнопку **Закрыть** внутри «подвала» окна;
- ◆ нажать клавишу `<Esc>` на клавиатуре. Чтобы отключить эту возможность, нужно к контейнеру с классом `modal` добавить параметр `data-keyboard` со значением `false` или параметр `data-backdrop` со значением `static`;
- ◆ щелчком левой кнопкой мыши вне диалогового окна. Чтобы отключить эту возможность, нужно к контейнеру с классом `modal` добавить параметр `data-backdrop` со значением `static`. В этом случае при щелчке вне окна масштаб окна будет незначительно меняться, показывая, что сначала нужно закрыть окно. Возможность закрытия окна по нажатию клавиши `<Esc>` также отключается.

### 4.17.3. Изменение размеров окна

Начиная с точки останова `sm`, для контейнера со стилевым классом `modal-dialog` устанавливается максимальная ширина, равная 500 пикселям, внешний отступ сверху и снизу, а также горизонтальное выравнивание по центру. Добавив следующие стилевые классы, можно изменить максимальную ширину окна:

- ◆ `modal-sm` — начиная с точки останова `sm`, для контейнера устанавливается максимальная ширина, равная 300 пикселям;
- ◆ `modal-lg` — начиная с точки останова `lg`, для контейнера устанавливается максимальная ширина, равная 800 пикселям. До этой точки действуют правила для класса `modal-dialog`;

- ◆ `modal-xl` — начиная с точки останова `xl`, для контейнера устанавливается максимальная ширина, равная 1140 пикселям. До этой точки действуют правила для класса `modal-lg`.

Укажем стиливой класс `modal-sm`, а также запретим закрытие окна при щелчке вне окна и нажатии клавиши `<Esc>`. Закрыть окно можно только по нажатию кнопки **ОК**:

```
<div class="modal fade" tabindex="-1" id="dialog1"
 aria-labelledby="dialog1-title" aria-hidden="true"
 data-keyboard="false" data-backdrop="static">
 <div class="modal-dialog modal-sm" role="document">
 <div class="modal-content">
 <div class="modal-header">
 <h5 class="modal-title" id="dialog1-title">Заголовок окна</h5>
 </div>
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-primary"
 data-dismiss="modal">ОК</button>
 </div>
 </div>
 </div>
</div>
</div>
```

#### 4.17.4. Размещение содержимого внутри окна

Контейнер со стиливым классом `modal-body` описывает раздел с основным содержимым диалогового окна. Если содержимое будет слишком велико по высоте, то появится вертикальная полоса прокрутки для всего диалогового окна:

```
<div class="modal fade" tabindex="-1" id="dialog1" aria-hidden="true">
 <div class="modal-dialog" role="document">
 <div class="modal-content">
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 <div class="bg-info" style="height: 600px"></div>
 <div class="bg-dark" style="height: 600px"></div>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Закрыть</button>
 </div>
 </div>
 </div>
</div>
```

Если помимо класса `modal-dialog` добавить стиливой класс `modal-dialog-scrollable`, то в Bootstrap 4 диалоговое окно займет максимально возможное видимое про-

странство, а прокручиваться будет лишь содержимое раздела `modal-body`, а не содержимое всего окна. Полоса прокрутки появится только в том случае, когда содержимое раздела не помещается целиком в окне веб-браузера:

```
<div class="modal fade" tabindex="-1" id="dialog1" aria-hidden="true">
 <div class="modal-dialog modal-dialog-scrollable" role="document">
 <div class="modal-content">
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 <div class="bg-info" style="height: 600px"></div>
 <div class="bg-dark" style="height: 600px"></div>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Закрыть</button>
 </div>
 </div>
 </div>
</div>
```

В раздел `modal-body` можно добавить базовый контейнер со стилевым классом `container-fluid`, содержащим систему сеток библиотеки Bootstrap, и размещать элементы по ячейкам сетки. Добавим два текстовых поля с выравниванием по горизонтали, начиная с точки останова `sm`:

```
<div class="modal fade" tabindex="-1" id="dialog1"
 aria-labelledby="dialog1-title" aria-hidden="true">
 <div class="modal-dialog" role="document">
 <div class="modal-content">
 <div class="modal-header">
 <h5 class="modal-title" id="dialog1-title">Заголовок окна</h5>
 </div>
 <div class="modal-body">
 <div class="container-fluid">
 <div class="row">
 <div class="col-sm mb-2">
 <label for="login" class="sr-only">Логин:</label>
 <input type="text" class="form-control"
 id="login" placeholder="Логин">
 </div>
 <div class="col-sm mb-2">
 <label for="passwd" class="sr-only">Пароль:</label>
 <input type="password" class="form-control"
 id="passwd" placeholder="Пароль">
 </div>
 </div>
 </div>
 </div>
 </div>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-primary">OK</button>
```

```

 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Закрыть</button>
 </div>
</div>
</div>
</div>

```

Чтобы установить фокус на поле ввода после открытия окна, нужно внутри обработчика события `shown.bs.modal` получить ссылку на поле и вызвать JavaScript-метод `focus()`:

```

$('#dialog1').on('shown.bs.modal', function(e) {
 $('#login')[0].focus();
});

```

Код для Bootstrap 5 без jQuery:

```

var dialog1 = document.getElementById('dialog1');
dialog1.addEventListener('shown.bs.modal', function(e) {
 document.getElementById('login').focus();
}, false);

```

#### 4.17.5. Вертикальное выравнивание окна по центру

Диалоговое окно по горизонтали всегда выводится по центру, а вот по вертикали — нет. Чтобы выполнить вертикальное выравнивание диалогового окна по центру окна веб-браузера, следует, помимо класса `modal-dialog`, добавить стиливой класс `modal-dialog-centered`:

```

<div class="modal fade" tabindex="-1" id="dialog1" aria-hidden="true">
 <div class="modal-dialog modal-dialog-centered" role="document">
 <div class="modal-content">
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Закрыть</button>
 </div>
 </div>
 </div>
</div>

```

#### 4.17.6. Полноэкранный режим

В Bootstrap 5 модальное диалоговое окно можно открыть в полноэкранном режиме. Для этого, помимо класса `modal-dialog`, нужно добавить класс `modal-fullscreen`. В результате окно займет все доступное место на вкладке веб-браузера. Если содержимое окна будет слишком велико по высоте, то появится вертикальная полоса прокрутки для содержимого раздела `modal-body`:

```

<!-- Только в Bootstrap 5 -->
<div class="modal fade" tabindex="-1" id="dialog1" aria-hidden="true">

```

```

<div class="modal-dialog modal-fullscreen">
 <div class="modal-content">
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 <div class="bg-info" style="height: 600px"></div>
 <div class="bg-dark" style="height: 600px"></div>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Заккрыть</button>
 </div>
 </div>
</div>
</div>
<div class="container mt-2">
 <button type="button" class="btn btn-info"
 data-toggle="modal" data-target="#dialog1">
 Показать окно</button>
</div>

```

Можно также воспользоваться следующими стилевыми классами:

- ◆ `modal-fullscreen-sm-down` — полноэкранный режим до точки останова `sm`, а далее обычный режим;
- ◆ `modal-fullscreen-md-down` — полноэкранный режим до точки останова `md`, а далее обычный режим;
- ◆ `modal-fullscreen-lg-down` — полноэкранный режим до точки останова `lg`, а далее обычный режим;
- ◆ `modal-fullscreen-xl-down` — полноэкранный режим до точки останова `xl`, а далее обычный режим;
- ◆ `modal-fullscreen-xxl-down` — полноэкранный режим до точки останова `xxl`, а далее обычный режим.

Пример использования стилевого класса `modal-fullscreen-sm-down`:

```

<!-- Только в Bootstrap 5 -->
<div class="modal fade" tabindex="-1" id="dialog1" aria-hidden="true">
 <div class="modal-dialog modal-fullscreen-sm-down">
 <div class="modal-content">
 <div class="modal-body">
 <p>Текст внутри диалогового окна</p>
 <div class="bg-info" style="height: 600px"></div>
 <div class="bg-dark" style="height: 600px"></div>
 </div>
 <div class="modal-footer">
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Заккрыть</button>
 </div>
 </div>
 </div>
</div>

```

```

</div>
</div>
<div class="container mt-2">
 <button type="button" class="btn btn-info"
 data-toggle="modal" data-target="#dialog1">
 Показать окно</button>
</div>

```

### 4.17.7. Управление компонентом из программы

Для управления компонентом `modal` из программы на jQuery нужно вызвать метод `modal()` и передать ему одно из следующих значений:

- ◆ объект с опциями — можно указать следующие основные свойства:
  - `backdrop` — если указано значение `true` (значение по умолчанию) или `'static'`, то при открытом окне поверх содержимого документа, но ниже окна, отображается полупрозрачный серый элемент со стилевым классом `modal-backdrop`, который добавляется в конец документа. После закрытия окна этот элемент удаляется из документа. Чтобы отключить добавление этого элемента, следует указать значение `false`. Если указано значение `true`, то закрыть окно можно щелчком левой кнопкой мыши вне диалогового окна. Чтобы отключить эту возможность, нужно указать значение `'static'`. В этом случае при щелчке на элементе `modal-backdrop` масштаб окна будет незначительно меняться, показывая, что сначала нужно закрыть окно. Возможность закрытия окна по нажатию клавиши `<Esc>` также отключается. Значение можно также задать с помощью HTML-параметра `data-backdrop`;
  - `keyboard` — если указано значение `true` (значение по умолчанию), то закрыть окно можно с помощью нажатия клавиши `<Esc>` на клавиатуре. Чтобы отключить эту возможность, нужно указать значение `false` или задать для свойства `backdrop` значение `'static'`. Значение можно также задать с помощью HTML-параметра `data-keyboard`;
  - `focus` — если указано значение `true` (значение по умолчанию), то после вызова метода `modal()` в первый раз (при инициализации) на окно будет установлен фокус ввода. Значение `false` отключает установку фокуса,
  - `show` — если указано значение `true` (значение по умолчанию), то после вызова метода `modal()` в первый раз (при инициализации) окно будет отображено. Значение `false` отключает отображение окна;
- ◆ `'toggle'` — отображает или скрывает диалоговое окно;
- ◆ `'show'` — отображает диалоговое окно;
- ◆ `'hide'` — скрывает диалоговое окно;
- ◆ `'handleUpdate'` — обновляет позицию диалогового окна;
- ◆ `'dispose'` — удаляет данные, связанные с компонентом.

В Bootstrap 5 без jQuery создание объекта и прикрепление его к элементу выполняется с помощью конструктора класса `Modal`:

```
<Объект> = new bootstrap.Modal(<Элемент>[, <Объект с опциями>])
```

Пример создания объекта:

```
var dialog1 = document.getElementById('dialog1');
var dialog1Instance = new bootstrap.Modal(dialog1, {
 backdrop : 'static',
 keyboard : false,
 focus : false,
 show : false
});
```

Получить ссылку на объект, сохраненный в элементе, позволяет статический метод `getInstance(<Элемент>)`. Если объект не найден, то метод вернет значение `null`. Пример получения ссылки на объект:

```
var dialog1 = document.getElementById('dialog1');
var dialog1Instance = bootstrap.Modal.getInstance(dialog1);
```

Класс `Modal` содержит следующие методы:

- ◆ `toggle()` — отображает или скрывает диалоговое окно;
- ◆ `show()` — отображает диалоговое окно;
- ◆ `hide()` — скрывает диалоговое окно;
- ◆ `handleUpdate()` — обновляет позицию диалогового окна;
- ◆ `dispose()` — удаляет данные, связанные с компонентом.

Пример:

```
dialog1Instance.show();
```

#### **ОБРАТИТЕ ВНИМАНИЕ!**

Методы запускают переключение состояния асинхронно. Это означает, что код, расположенный сразу после метода, будет выполнен еще до завершения переключения. Если метод вызывается в процессе переключения, то вызов может быть проигнорирован.

### 4.17.8. Обработка событий

Чтобы обработать события компонента, нужно назначить обработчики для следующих событий:

- ◆ `show.bs.modal` — событие перед отображением диалогового окна. Внутри обработчика через объект события доступно свойство `relatedTarget`, которое содержит ссылку на кнопку (или ссылку), с помощью которой открывается окно. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то окно отображено не будет:

```
var flagShow = false;
$('#dialog1').on('show.bs.modal', function(e) {
 if (!flagShow) e.preventDefault();
});
```

### Пример для Bootstrap 5 без jQuery:

```
var flagShow = false;
var dialog1 = document.getElementById('dialog1');
dialog1.addEventListener('show.bs.modal', function(e) {
 if (!flagShow) e.preventDefault();
}, false);
```

◆ `shown.bs.modal` — событие после отображения диалогового окна. Внутри обработчика через объект события доступно свойство `relatedTarget`, которое содержит ссылку на кнопку (или ссылку), с помощью которой открывается окно;

◆ `hide.bs.modal` — событие перед сокрытием диалогового окна. Если внутри обработчика через объект события вызвать метод `preventDefault()`, то окно скрыто не будет:

```
var flagHide = false;
$('#dialog1').on('hide.bs.modal', function(e) {
 if (!flagHide) e.preventDefault();
});
```

◆ `hidden.bs.modal` — событие после сокрытия диалогового окна;

◆ `hidePrevented.bs.modal` — событие при попытке закрыть окно путем щелчка левой кнопкой мыши вне диалогового окна, при значении `'static'` свойства `backdrop`.

Пример управления компонентом `modal` из программы и обработки событий приведен в листинге 4.16.

#### Листинг 4.16. Компонент `modal`: модальные диалоговые окна

```
<!doctype html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <meta name="viewport"
 content="width=device-width, initial-scale=1, shrink-to-fit=no">
 <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
 <title>Компонент modal: модальные диалоговые окна</title>
</head>
<body>
<div class="modal fade" tabindex="-1" id="dialog1"
 aria-labelledby="dialog1-title" aria-hidden="true">
 <div class="modal-dialog modal-dialog-centered" role="document">
 <div class="modal-content">
 <div class="modal-header">
 <h5 class="modal-title" id="dialog1-title">Заголовок окна</h5>
 <button type="button" class="close"
 data-dismiss="modal" aria-label="Закрыть">
 ×
 </button>
 </div>
```

```
<div class="modal-body">
 <label for="passwd">Пароль:</label>
 <input type="password" class="form-control"
 id="passwd" placeholder="Введите пароль">
</div>
<div class="modal-footer">
 <button type="button" class="btn btn-primary"
 id="btnOK">OK</button>
 <button type="button" class="btn btn-secondary"
 data-dismiss="modal">Отмена</button>
</div>
</div>
</div>
</div>
<div class="container mt-2">
 <button type="button" class="btn btn-info"
 data-toggle="modal" data-target="#dialog1">
 Показать окно</button>
 <button type="button" class="btn btn-info" id="btnShow">
 Показать окно</button>
 <button type="button" class="btn btn-info" id="btnToggle">
 Показать или скрыть окно</button>
 <p>Результат обработки событий см. в окне консоли.</p>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script>
$(function() {
 var dialog1 = $('#dialog1');
 dialog1.modal({
 backdrop : 'static',
 keyboard : false,
 focus : false,
 show : false
 });

 $('#btnShow').click(function() {
 $('#dialog1').modal('show');
 });
 $('#btnToggle').click(function() {
 $('#dialog1').modal('toggle');
 });
 $('#btnOK').click(function() {
 var passwd = $('#passwd').val();
 if (passwd === '') {
 window.alert('Вы не ввели пароль');
 $('#passwd')[0].focus();
 }
 }
}
```

```

 else {
 console.log('Пароль: ' + passwd);
 $('#dialog1').modal('hide');
 }
 });

 dialog1.on('show.bs.modal', function(e) {
 console.log('show.bs.modal');
 if (e.relatedTarget) console.log(e.relatedTarget);
 });
 dialog1.on('shown.bs.modal', function(e) {
 console.log('shown.bs.modal');
 if (e.relatedTarget) console.log(e.relatedTarget);
 $('#passwd')[0].focus();
 });
 dialog1.on('hide.bs.modal', function(e) {
 console.log('hide.bs.modal');
 });
 dialog1.on('hidden.bs.modal', function(e) {
 console.log('hidden.bs.modal');
 $('#passwd').val('');
 });
 dialog1.on('hidePrevented.bs.modal', function(e) {
 console.log('hidePrevented.bs.modal');
 });
});
</script>
</body>
</html>

```

В этом примере мы воспользовались новым для вас методом из библиотеки jQuery. Метод `val()`, если он вызван без параметра, возвращает текущее значение элемента. Если параметр указан, то метод `val()` задает новое значение элемента.

Изменим содержимое тега `<script>` из листинга 4.16 так, чтобы программа работала в Bootstrap 5 без jQuery:

```

var dialog1 = document.getElementById('dialog1');
var dialog1Instance = new bootstrap.Modal(dialog1, {
 backdrop : 'static',
 keyboard : false,
 focus : false,
 show : false
});

var btnShow = document.getElementById('btnShow');
btnShow.addEventListener('click', function() {
 dialog1Instance.show();
}, false);
var btnToggle = document.getElementById('btnToggle');
btnToggle.addEventListener('click', function() {

```

```
 dialog1Instance.toggle();
 }, false);
var btnOK = document.getElementById('btnOK');
btnOK.addEventListener('click', function() {
 var passwdEl = document.getElementById('passwd');
 var passwd = passwdEl.value;
 if (passwd === '') {
 window.alert('Вы не ввели пароль');
 passwdEl.focus();
 }
 else {
 console.log('Пароль: ' + passwd);
 dialog1Instance.hide();
 }
}, false);

dialog1.addEventListener('show.bs.modal', function(e) {
 console.log('show.bs.modal');
 if (e.relatedTarget) console.log(e.relatedTarget);
}, false);
dialog1.addEventListener('shown.bs.modal', function(e) {
 console.log('shown.bs.modal');
 if (e.relatedTarget) console.log(e.relatedTarget);
 document.getElementById('passwd').focus();
}, false);
dialog1.addEventListener('hide.bs.modal', function(e) {
 console.log('hide.bs.modal');
}, false);
dialog1.addEventListener('hidden.bs.modal', function(e) {
 console.log('hidden.bs.modal');
 document.getElementById('passwd').value = '';
}, false);
dialog1.addEventListener('hidePrevented.bs.modal', function(e) {
 console.log('hidePrevented.bs.modal');
}, false);
```



## ГЛАВА 5

# CSS-препроцессор Sass

В предыдущих главах мы подключали файл `bootstrap.min.css`, который содержит все стили библиотеки Bootstrap. Однако как часто мы используем абсолютно все стили Bootstrap в одном проекте? Очень редко! Между тем существует возможность выполнить сборку библиотеки Bootstrap под свой проект и уменьшить размер подключаемого CSS-файла. Для этого потребуются знание основ CSS-препроцессора Sass, используемого разработчиками Bootstrap для создания такого CSS-файла. Если же штатные настройки библиотеки Bootstrap вас устраивают, то вы можете отложить изучение Sass на потом и заняться разработкой адаптивных веб-сайтов, применяя на практике изученный материал первых четырех глав книги.

## 5.1. Первые шаги

Знание CSS-препроцессора Sass позволит:

- ◆ уменьшить размер CSS-файла библиотеки Bootstrap путем отключения неиспользуемых модулей (например, модулей с кодом компонентов);
- ◆ изменить значения некоторых переменных (например, поменять цвет);
- ◆ добавить свои цветовые темы (достаточно указать название темы и ее цвет, и весь код будет сгенерирован автоматически при компиляции);
- ◆ добавить свои произвольные стили или переопределить стили библиотеки Bootstrap;
- ◆ создать собственный проект без участия Bootstrap. Этот пункт особенно важен, если вы являетесь профессиональным разработчиком или хотите им стать.

Начнем изучение CSS-препроцессора Sass и сначала установим программу Node.js.

### 5.1.1. Установка Node.js

Для установки Node.js переходим на сайт <https://nodejs.org/> и скачиваем программу установки. В моем случае она имеет название `node-v12.18.3-x64.msi`. Запускаем программу установки и на первом шаге (рис. 5.1) нажимаем кнопку **Next**. На втором шаге принимаем лицензионное соглашение (рис. 5.2) и нажимаем кнопку **Next**.

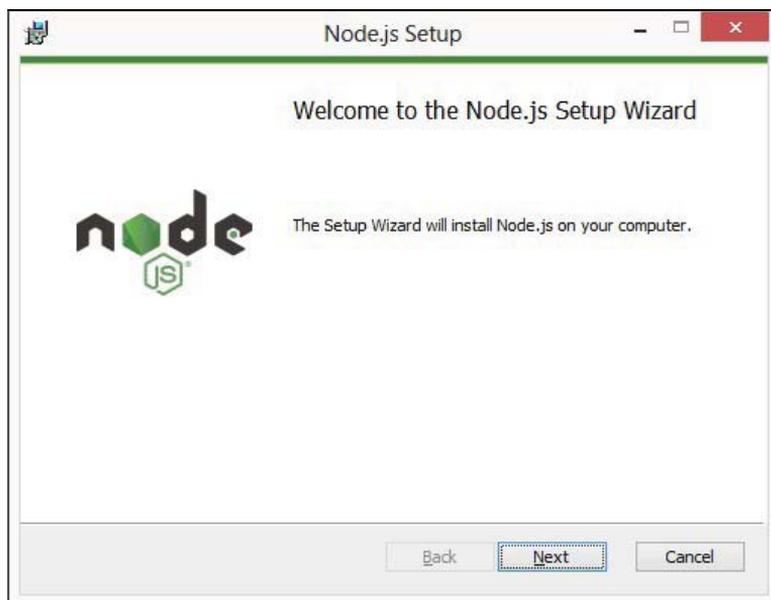


Рис. 5.1. Установка Node.js: шаг 1

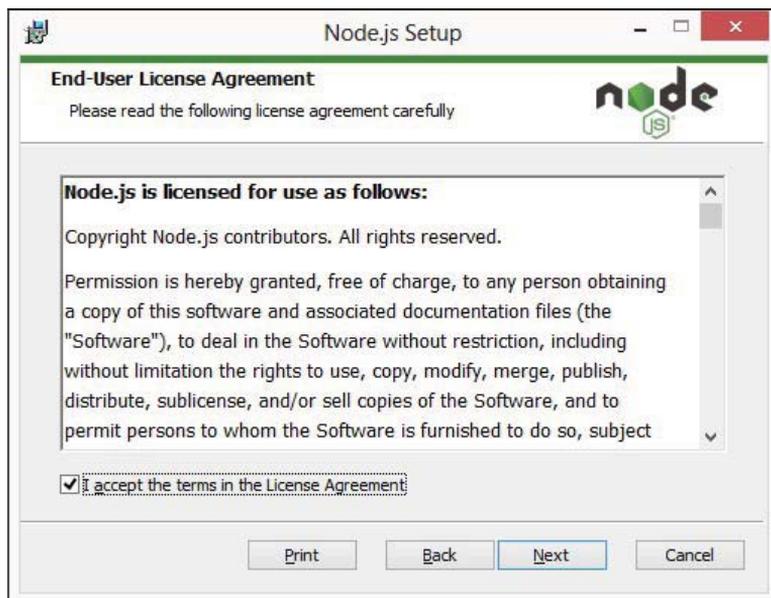


Рис. 5.2. Установка Node.js: шаг 2

На следующем шаге нужно выбрать каталог для установки Node.js. Можно оставить путь по умолчанию, но я устанавливаю Node.js в каталог `C:\nodejs\node12` (рис. 5.3), следя за тем, чтобы в записи пути не было пробелов. Выбираем каталог и

нажимаем кнопку **Next**. На следующем шаге можно выбрать устанавливаемые компоненты (рис. 5.4). Оставляем настройки по умолчанию и нажимаем кнопку **Next**.

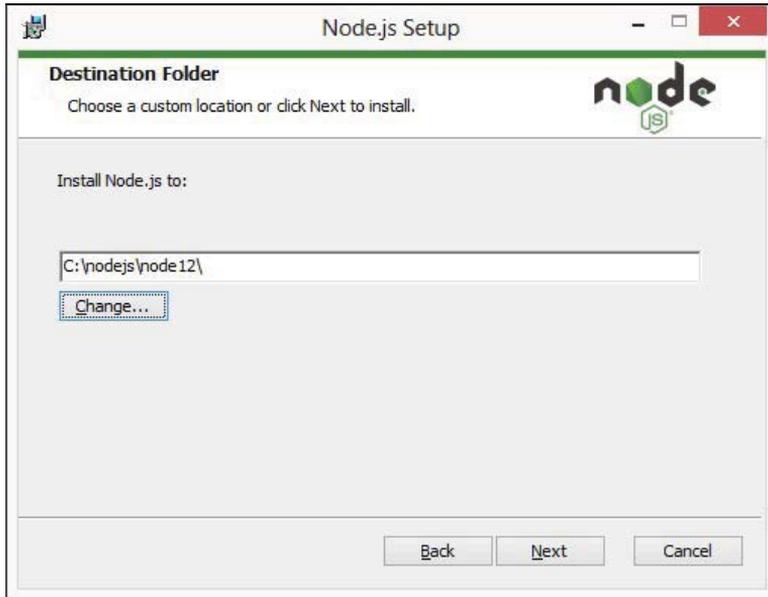


Рис. 5.3. Установка Node.js: шаг 3

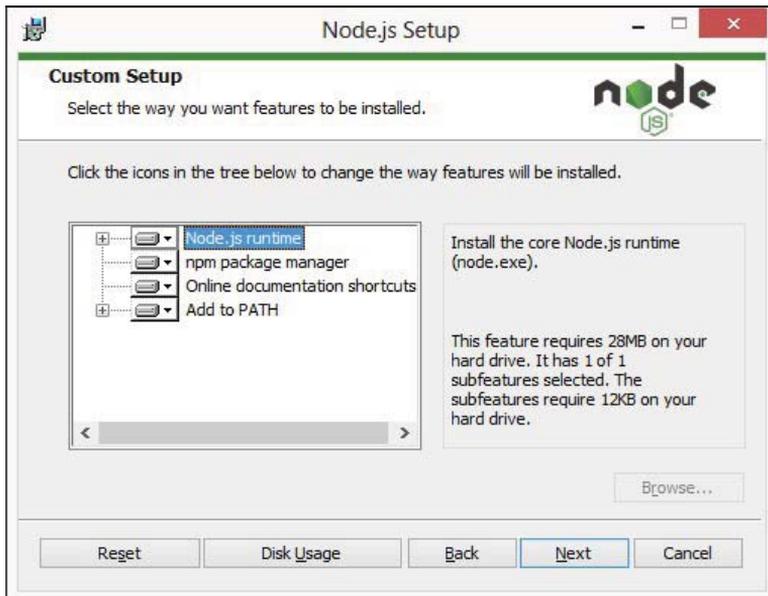


Рис. 5.4. Установка Node.js: шаг 4

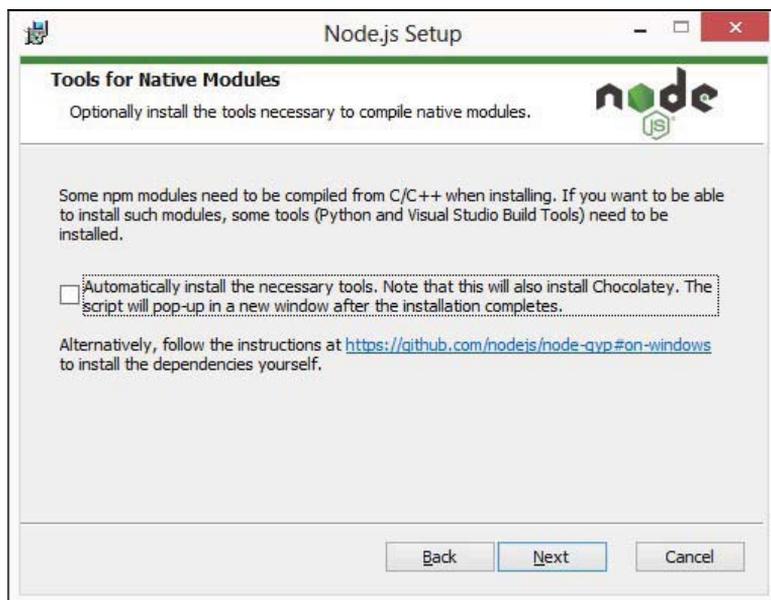


Рис. 5.5. Установка Node.js: шаг 5

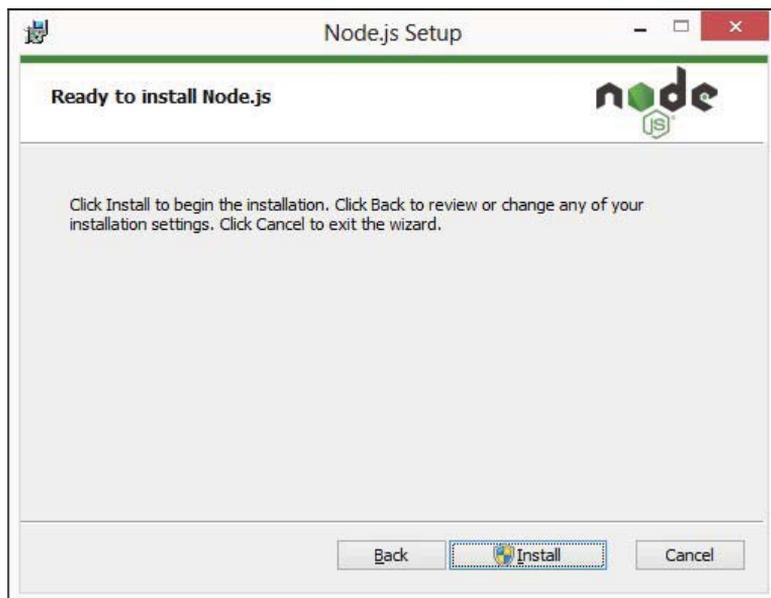


Рис. 5.6. Установка Node.js: шаг 6

На следующем шаге (рис. 5.5) также ничего не меняем и нажимаем кнопку **Next**. Далее нажимаем кнопку **Install** (рис. 5.6) для запуска процесса установки. Для установки требуются права администратора, поэтому в следующем окне нужно раз-

решить установку. В окне, открывшемся по завершении процесса установки Node.js (рис. 5.7), нажимаем кнопку **Finish** для выхода из программы установки.

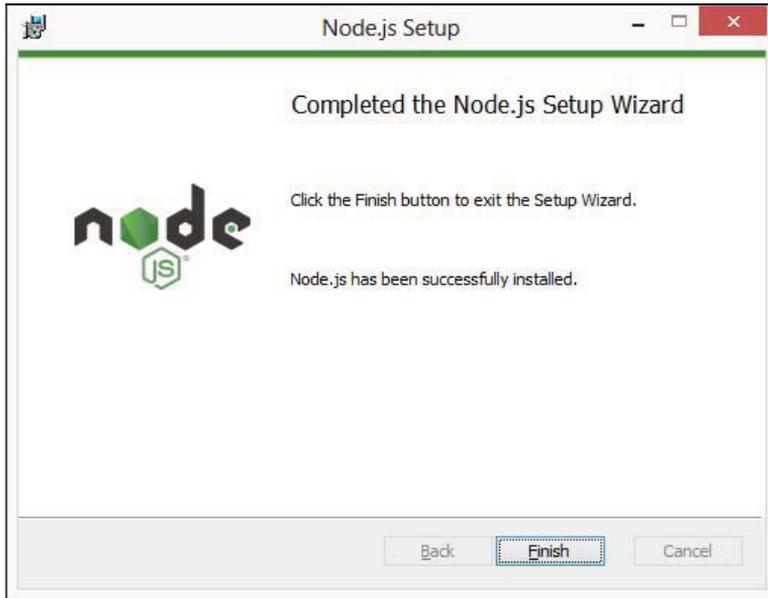


Рис. 5.7. Установка Node.js: шаг 7

### 5.1.2. Работа с командной строкой

Для проверки правильности установки Node.js и для дальнейшей работы нам понадобится приложение **Командная строка**. Вполне возможно, что вы никогда не пользовались командной строкой и не знаете, как запустить это приложение. Давайте рассмотрим некоторые способы его запуска в Windows:

- ◆ через поиск находим приложение **Командная строка**;
- ◆ нажимаем комбинацию клавиш <Windows>+<R>. В открывшемся окне вводим `cmd` и нажимаем кнопку **ОК**;
- ◆ находим файл `cmd.exe` в каталоге `C:\Windows\System32`;
- ◆ в Проводнике щелкаем правой кнопкой мыши на свободном месте списка файлов, удерживая при этом нажатой клавишу <Shift>, и из контекстного меню выбираем пункт **Открыть окно команд**;
- ◆ в Проводнике в адресной строке вводим `cmd` и нажимаем клавишу <Enter>.

В некоторых случаях для выполнения различных команд могут потребоваться права администратора. Чтобы запустить командную строку с правами администратора, через поиск находим приложение **Командная строка**, щелкаем на значке правой кнопкой мыши и из открывшегося контекстного меню выбираем пункт **Запуск от имени администратора**.

Запомните способы запуска командной строки наизусть. В дальнейшем мы просто будем говорить: «запустите командную строку» — без уточнения, как это сделать.

Итак, в результате установки Node.js требуемые файлы были загружены в каталог, указанный на третьем шаге мастера. Кроме того, если вы ничего не меняли на четвертом шаге, путь к Node.js был автоматически добавлен в переменную окружения `PATH`. Поэтому в командной строке можно просто указать название программы без явного добавления пути к ней. Давайте проверим работоспособность Node.js. Запускаем командную строку и выполняем команду `node --version`:

```
C:\book>node --version
```

### v12.18.3

Фрагмент `C:\book>` перед командой означает приглашение для ввода команды с указанием текущего рабочего каталога. Текст, выведенный после команды, является результатом ее выполнения. В нашем примере мы получили версию Node.js.

Если вы увидели следующий результат:

```
C:\book>node --version
```

**"node" не является внутренней или внешней командой, исполняемой программой или пакетным файлом.**

то программа Node.js не найдена. Вполне возможно, что путь к ней не был добавлен в переменную окружения `PATH`. Выведите значение этой переменной с помощью команды:

```
echo %Path%
```

Если путь не прописан, то его нужно добавить вручную. Чтобы изменить системную переменную `PATH` в Windows, переходим в **Параметры | Панель управления | Система и безопасность | Система | Дополнительные параметры системы**. В результате откроется окно **Свойства системы**. На вкладке **Дополнительно** нажимаем кнопку **Переменные среды**. В открывшемся окне в списке **Системные переменные** выделяем строку с переменной `Path` и нажимаем кнопку **Изменить**.

#### **ВНИМАНИЕ!**

Случайно не удалите существующее значение переменной `PATH`, иначе другие приложения перестанут запускаться.

Значение системной переменной можно временно изменить для текущего сеанса, т. е. до момента закрытия приложения **Командная строка**. Для этого перед работой с Node.js нужно выполнить следующую команду:

```
set Path=<Путь до Node.js>%Path%
```

Таким же способом можно запускать и другие версии Node.js без переустановки. Путь добавляется в самое начало системной переменной `PATH`, поэтому будет использоваться именно этот путь. У меня на компьютере уставлено несколько версий Node.js, поэтому этим способом я могу запустить, например, Node.js версии 13:

```
C:\book>set Path=C:\nodejs\node13;%Path%
```

```
C:\book>node --version
```

### v13.11.0

Если в записи пути есть пробелы, то в командной строке такой путь нужно обязательно указывать в кавычках, иначе он будет обрезан до первого пробела. Поэтому, чтобы избежать проблем, на третьем шаге установки лучше задать путь без пробелов (см. рис. 5.3). Кроме того, имена каталогов и файлов в пути не должны содержать русских букв. Допустимы только латинские буквы, цифры, дефис, подчеркивание и некоторые другие символы.

Советую также убедиться, что только латинские буквы и цифры содержит и имя пользователя компьютера. В нем не должно быть никаких русских букв и пробелов, поскольку многие программы, включая Node.js, сохраняют различные настройки и временные файлы в каталоге `C:\Users\<Имя пользователя>`. Если имя пользователя содержит русские буквы, то они могут быть искажены до неузнаваемости из-за неправильного преобразования кодировок, и программа не сможет сохранить настройки. Помните, что в разных кодировках русские буквы могут иметь разный код. Разработчики программ в основном работают с английским языком и ничего не знают о проблемах с кодировками, т. к. во всех однобайтовых кодировках и в кодировке UTF-8 коды латинских букв одинаковы. Так что, если вы хотите без проблем заниматься программированием, то от использования русских букв в имени пользователя лучше отказаться.

С помощью Node.js можно запускать на выполнение программы, написанные на языке программирования JavaScript. Причем, в отличие от программ, выполняемых в веб-браузере, здесь отсутствует «песочница». Иными словами, программа, выполняемая с помощью Node.js, имеет доступ ко всему компьютеру пользователя, точно так же, как и программы на других языках, — например, на языке Java.

Давайте напишем программу, выводящую приветствие в окно консоли. Для этого в каталоге `C:\book` создадим файл `app.js` в кодировке UTF-8 без BOM со следующим содержанием:

```
console.log('Привет, Node.js!');
```

Запускаем командную строку и выполняем следующую команду:

```
C:\Users\Имя пользователя>node C:\book\app.js
```

**Привет, Node.js!**

В результате мы получили приветствие, причем на русском языке без проблем с кодировками. Преобразование кодировок при выводе данных производится автоматически. Вы же помните, что наш файл сохранен в кодировке UTF-8? А консоль использует по умолчанию кодировку windows-866. Давайте в этом убедимся:

```
C:\Users\Имя пользователя>chcp
```

**Текущая кодовая страница: 866**

Чтобы не вводить каждый раз полный путь к программе, следует сделать каталог `C:\book` текущим с помощью следующей команды:

```
C:\Users\Имя пользователя>cd C:\book
```

Теперь достаточно указать только название программы (можно без расширения):

```
C:\book>node app
```

**Привет, Node.js!**

В этой книге мы не станем изучать возможности Node.js. Мы установили Node.js лишь для того, чтобы иметь возможность запускать различные утилиты, написанные под Node.js. Установить эти утилиты позволяет программа NPM (Node Package Manager, менеджер пакетов Node.js), входящая в состав установленных компонентов. Давайте проверим ее работоспособность, запросив вывод номера версии:

```
C:\book>npm --version
```

### 6.14.6

Если вы успешно получили версии Node.js и NPM, то можно продолжить изучение материала.

#### **НА ЗАМЕТКУ**

Все выполненные команды записываются в список истории. Перемещаться внутри этого списка позволяют клавиши со стрелками вверх и вниз. Например, чтобы выполнить последнюю команду, достаточно нажать клавишу со стрелкой вверх, а затем клавишу <Enter>. Чтобы очистить окно консоли, нужно ввести команду `cls`. Список истории при этом не очищается.

## 5.1.3. Установка Sass с помощью NPM

Установить CSS-препроцессор Sass можно несколькими способами. Первый способ заключается в загрузке архива с программой со страницы <https://github.com/sass/dart-sass/releases/tag/1.26.10>. Достаточно распаковать архив в какой-либо каталог и прописать путь к нему в переменную окружения `PATH`. Мы же сейчас воспользуемся вторым способом — установим программу Sass с помощью NPM. Для этого запускаем командную строку и выполняем следующую команду (требуется доступ в Интернет):

```
npm install -g sass
```

В этой команде `npm` является названием запускаемой программы. Флаг `install` означает, что мы хотим установить пакет, название которого указано в самом конце команды, — пакет с названием `sass`. Флаг `-g` означает, что пакет должен быть доступен для всех проектов, то есть он должен быть установлен глобально.

Если установка прошла успешно, файлы программы Sass можно найти в каталогах `C:\Users\<Имя пользователя>\AppData\Roaming\npm` и `C:\Users\<Имя пользователя>\AppData\Roaming\npm\node_modules\sass`. Первый каталог добавлен в переменную окружения `PATH`, поэтому мы можем в командной строке указывать только название программы без добавления пути к ней. Давайте выведем версию Sass:

```
C:\book>sass --version
```

### 1.26.10 compiled with dart2js 2.8.4

## 5.1.4. Создание пакета и добавление файла `package.json`

Файлы с исходным кодом для программы Sass могут иметь расширения `scss` и `sass`. Синтаксис этих файлов различается. Мы будем изучать синтаксис файлов с расши-

рением scss, т. к. он очень похож на синтаксис обычного CSS-файла и активно используется внутри библиотеки Bootstrap.

Давайте в каталоге C:\book создадим полноценный пакет Node.js под названием p1 (сокращение от package1, чтобы команды были короче). Внутри каталога C:\book\p1 создаем каталоги node\_modules, scss и dist. Внутри каталога C:\book\p1\dist создаем вложенный каталог css, в который будем сохранять создаваемые CSS-файлы. Внутри каталога C:\book\p1 создаем файл index.html (листинг 5.1), в котором подключим CSS-файл. Вместо создания CSS-файла мы в каталоге C:\book\p1\scss создадим файл main.scss (листинг 5.2), содержащий код для программы Sass. Файл должен быть сохранен в кодировке UTF-8 без BOM.

#### Листинг 5.1. Содержимое файла index.html

```
<!doctype html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <link rel="stylesheet" type="text/css" href="dist/css/main.css">
 <title>Sass</title>
</head>
<body>
 <p>Привет, Sass!</p>
</body>
</html>
```

#### Листинг 5.2. Содержимое файла main.scss

```
$black: #000000;
$white: #ffffff;

body {
 color: $white;
 background-color: $black;
}
```

Чтобы превратить каталог C:\book\p1 в полноценный пакет, нужно создать в нем файл с названием package.json. Для этого открываем командную строку и переходим в каталог пакета:

```
C:\Users\Имя пользователя>cd C:\book\p1
```

```
C:\book\p1>
```

Обратите внимание: название каталога должно быть в составе приглашения для ввода команды. Если путь будет другим, то файл будет создан в другом каталоге. Для создания файла package.json выполняем команду `npm init -y`:

```
C:\book\p1>npm init -y
Wrote to C:\book\p1\package.json:
```

```
{
 "name": "p1",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
 "test": "echo `Error: no test specified` && exit 1"
 },
 "keywords": [],
 "author": "",
 "license": "ISC"
}
```

После выполнения команды было выведено содержимое созданного файла. В этой команде мы указали флаг `-y`. Если флаг не указать, то программа будет запрашивать данные прямо в командной строке. Можно ввести данные, а можно на каждый запрос нажимать клавишу `<Enter>`. В конце диалога нужно подтвердить создание файла:

```
C:\book\p1>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible
defaults.
```

See ``npm help init`` for definitive documentation on these fields and exactly what they do.

Use ``npm install <pkg>`` afterwards to install a package and save it as a dependency in the package.json file.

Press `^C` at any time to quit.

```
package name: (p1)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\book\p1\package.json:
```

```
{
 "name": "p1",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
 "test": "echo `Error: no test specified` && exit 1"
 },
```

```
"author": "",
"license": "ISC"
}
```

### Is this OK? (yes)

Гораздо удобнее отредактировать созданный файл с помощью текстового редактора, чем что-то вводить в командной строке. Поэтому в команде мы указали флаг `-y`.

## 5.1.5. Создание CSS-файла из SCSS-файла

Теперь нам нужно преобразовать содержимое файла `main.scss` в CSS-файл, понимаемый веб-браузером. Для этого выполняем следующую команду:

```
sass <Путь к SCSS-файлу>[<Путь к CSS-файлу>]
```

Пример:

```
C:\book\p1>sass scss/main.scss dist/css/main.css
```

```
C:\book\p1>
```

Если все сделано правильно, в каталоге `C:\book\p1\dist\css` будет создан файл `main.css` со следующим кодом:

```
body {
 color: #ffffff;
 background-color: #000000;
}
```

```
/*# sourceMappingURL=main.css.map */
```

Откройте в веб-браузере файл `C:\book\p1\index.html` и убедитесь, что фон страницы стал черным, а цвет текста — белым.

Чтобы после преобразования получить CSS-файл в сжатом виде, следует в командной строке указать флаг `--style` со значением `compressed`:

```
C:\book\p1>sass --style=compressed scss/main.scss dist/css/main.min.css
```

В результате преобразования будет создан файл `main.min.css` со следующим кодом:

```
body{color:#fff;background-color:#000}/*# sourceMappingURL=
main.min.css.map */
```

Если путь к CSS-файлу не указан, то результат выводится в окно консоли:

```
C:\book\p1>sass scss/main.scss
```

```
body {
 color: #ffffff;
 background-color: #000000;
}
```

```
C:\book\p1>
```

Существует возможность преобразования всех SCSS-файлов внутри указанного каталога и создания одноименных CSS-файлов. Команда в этом случае выглядит так:

```
sass <Каталог с SCSS-файлами>:<Каталог с CSS-файлами>
```

Пример:

```
C:\book\p1>sass scss:dist/css
```

Помимо файла `main.css`, программа Sass создала еще один файл: `main.css.map`. Этот файл очень удобен для отладки. Если мы откроем **Инструменты разработчика** на вкладке **Инспектор** и выделим тег `<body>`, то при просмотре стилей сможем заметить, что веб-браузер показывает местоположение стилей в файле `main.scss` (рис. 5.8), а не в файле `main.css`. Если щелкнуть левой кнопкой мыши на ссылке, то на вкладке **Стили** откроется содержимое файла `main.scss`. Нам, как разработчикам, гораздо важнее видеть ссылки на файл с исходным кодом, а не на файл, который мы используем.

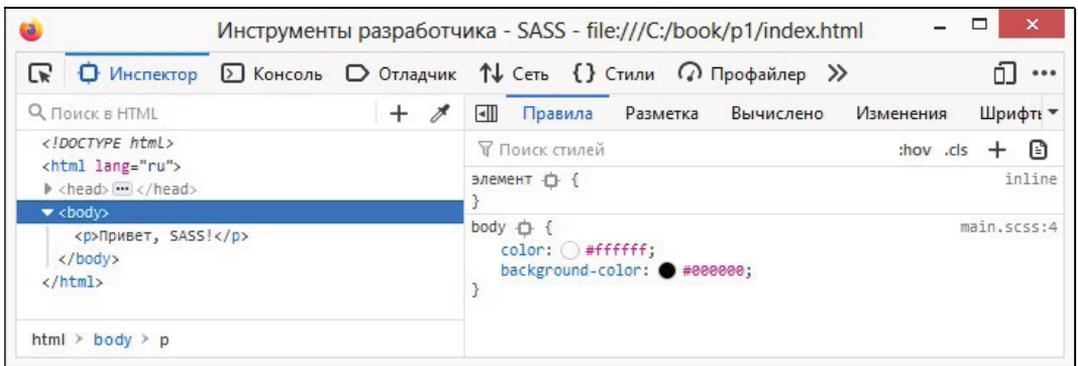


Рис. 5.8. Инструменты разработчика: вкладка **Инспектор**

С помощью флага `--source-map-urls` можно задать формат вывода пути к SCSS-файлу внутри файла с расширением `css.map`:

- ◆ `relative` — относительный путь: `"sources":["../../scss/main.scss"]` — значение по умолчанию;
- ◆ `absolute` — абсолютный путь: `"sources":["file:///c:/book/p1/scss/main.scss"]`.

Пример указания значений:

```
sass --source-map-urls=relative scss/main.scss dist/css/main.css
```

```
sass --source-map-urls=absolute scss/main.scss dist/css/main.css
```

Если указан флаг `--embed-sources`, то содержимое SCSS-файла будет вставлено внутрь файла с расширением `css.map`:

```
C:\book\p1>sass --embed-sources scss/main.scss dist/css/main.css
```

Если указан флаг `--embed-source-map`, то содержимое файла `css.map` будет вставлено внутрь CSS-файла:

```
C:\book\p1>sass --embed-source-map scss/main.scss dist/css/main.css
```

В результате преобразования будет создан только файл `main.css` со следующим кодом:

```
body {
 color: #ffffff;
 background-color: #000000;
}

/*# sourceMappingURL=data:application/json;charset=utf-8,%7B%22version%22:3,%22sourceRoot%22:%22%22,%22sources%22:%5B%22../..%2Fscss%2Fmain.scss%22%5D,%22names%22:%5B%5D,%22mappings%22:%22AAGA;EACG,OAJK;EAKL,kBAJK%22,%22file%22:%22main.css%22%7D */
```

Чтобы запретить создание файла с расширением `css.map`, следует в командной строке указать флаг `--no-source-map`:

```
C:\book\p1>sass --no-source-map scss/main.scss dist/css/main.css
```

Каждый раз вводить пути до файлов не очень удобно. Чтобы это исправить, напишем скрипт, который можно будет запустить с помощью программы NPM следующим образом:

```
npm run <Название скрипта>
```

Для этого открываем файл `package.json` с помощью программы Notepad++ и находим следующие строки:

```
"scripts": {
 "test": "echo \"Error: no test specified\" && exit 1"
}
```

Изменяем фрагмент следующим образом и сохраняем файл:

```
"scripts": {
 "createCSS": "sass scss/main.scss dist/css/main.css"
}
```

Запустим процесс преобразования:

```
C:\book\p1>npm run createCSS

> p1@1.0.0 createCSS C:\book\p1
> sass scss/main.scss dist/css/main.css
```

```
C:\book\p1>
```

Теперь нам не нужно каждый раз вводить пути в командной строке. Достаточно запомнить лишь название скрипта: `createCSS`. Если пути изменятся, то внести корректировки нужно будет только в одном месте. В случае возникновения ошибки ее описание выводится в консоли, а также записывается в CSS-файл. Кроме того, в каталоге `C:\Users\<Имя пользователя>\AppData\Roaming\npm-cache\_logs` создается файл с полным описанием ошибки.

Надеюсь, что принцип работы понятен. Пишем инструкции в SCSS-файле, затем запускаем программу Sass и получаем CSS-файл в обычном или сжатом виде.

## 5.1.6. Отслеживание изменений SCSS-файлов

Запускать команду в консоли после каждого изменения SCSS-файла очень быстро надоедает. Программисты обычно ленивые люди, поэтому если что-то можно автоматизировать, то мы с радостью это делаем. Вот и разработчики программы Sass предусмотрели способ отслеживания изменений SCSS-файла и автоматического создания CSS-файла сразу после сохранения файла с исходным кодом. Для этого в команде преобразования нужно дополнительно указать флаг `--watch`:

```
C:\book\p1>sass --watch scss/main.scss dist/css/main.css
Sass is watching for changes. Press Ctrl-C to stop.
```

После запуска команды программа Sass будет выполняться в бесконечном цикле, блокируя ввод других команд. Чтобы вводить другие команды, нужно запустить дополнительное окно командной строки. Сразу после сохранения SCSS-файла будет автоматически запущен процесс преобразования, и в командной строке появится следующее сообщение, выделенное зеленым цветом:

```
Compiled scss\main.scss to dist\css\main.css.
```

Если в файле содержится ошибка, то будет выведено сообщение о проблеме:

```
Error: Undefined variable.
```

```
| background-color: $black2;
| ^^^^^^^
```

```
scss\main.scss 6:22 root stylesheet
```

Чтобы остановить отслеживание, нужно нажать комбинацию клавиш `<Ctrl>+<C>` или закрыть окно консоли. В первом случае программа запросит подтверждение действия — путем ввода буквы `y` и нажатия клавиши `<Enter>`:

```
Завершить выполнение пакетного файла [Y(да)/N(нет)]? y
```

```
C:\book\p1>
```

Существует возможность отслеживания изменения всех SCSS-файлов внутри указанного каталога и автоматического создания одноименных CSS-файлов. Команда в этом случае выглядит так:

```
sass --watch <Каталог с SCSS-файлами>:<Каталог с CSS-файлами>
```

Пример запуска отслеживания и автоматического преобразования:

```
C:\book\p1>sass --watch scss:dist/css
Sass is watching for changes. Press Ctrl-C to stop.
```

```
Compiled scss\main.scss to dist\css\main.css.
```

Если система уведомления операционной системы не работает, то помимо флага `--watch` нужно передать флаг `--poll`, чтобы Sass самостоятельно отслеживала изменения SCSS-файлов:

```
C:\book\p1>sass --watch --poll scss:dist/css
```

Существуют также дополнительные способы упрощения и автоматизации процесса преобразования, но на данном этапе этих способов достаточно, чтобы начать изучение синтаксиса SCSS-файлов.

### 5.1.7. Интерактивный режим

Выражения SassScript можно выполнять в командной строке и сразу видеть результат, запустив интерактивный режим с помощью команды `sass -i` или `sass --interactive`:

```
C:\book\p1>sass -i
>>
```

После запуска выводится приглашение для ввода инструкции `>>`, и программа ожидает действий от нас. Инструкция или выражение вводятся без указания точки с запятой в конце. После окончания ввода следует нажать клавишу `<Enter>`. На следующей строке сразу выводится значение выражения, и программа снова ожидает наших действий. В интерактивном режиме поддерживаются выражения, переменные и инструкция `@use`:

```
>> 10px + 12px
22px
>> $size: 20px
20px
>> 10px + $size
30px
>> @use "sass:map"
>> $colors: ("red": #dc3545, "blue": #007bff)
("red": #dc3545, "blue": #007bff)
>> map.get($colors, "red")
#dc3545
>>
```

Для выхода из интерактивного режима нажимаем комбинацию клавиш `<Ctrl>+<C>`.

### 5.1.8. Комментарии

Если в SCSS-файле вставить комментарий из CSS, то он после преобразования будет добавлен в CSS-файл без изменений. Комментарий в CSS является многострочным:

```
/* Это комментарий в CSS */
```

Если внутри комментария имеется текст на русском языке, то в начало CSS-файла дополнительно будет прописано правило с указанием кодировки файла:

```
@charset "UTF-8";
/* Это комментарий в CSS */
```

При сжатии CSS-файла все комментарии удаляются. Если нужно сохранить комментарий в сжатом файле, то после открывающих комментарий символов `/*` нужно добавить восклицательный знак:

```
/*! Это многострочный комментарий в CSS
 * Он останется в сжатом CSS-файле, т. к. указан символ !
 */
```

Такой комментарий удобно использовать для вывода информации о разработчиках, названии продукта и лицензии. Если внутри комментария добавить следующую

конструкцию (название переменной указывается внутри фигурных скобок): `#{<Переменная>}`, то значение переменной будет подставлено в комментарий:

```
$version: "1.0.0";
/* Версия: #{ $version} */
```

Результат в CSS-файле:

```
/* Версия: 1.0.0 */
```

Чтобы добавить в SCSS-файл комментарий, который не попадет в итоговый CSS-файл, следует его указать после символов `//`. Такой комментарий является однострочным и действует от символов `//` до конца текущей строки:

```
// Это однострочный комментарий в Sass
```

Комментарий можно указать после инструкции:

```
$black: #000000; // Комментарий после инструкции
```

Или перед инструкцией, чтобы отключить ее:

```
// $red: #ff0000;
```

### 5.1.9. Директивы `@debug`, `@warn` и `@error`

Для вывода сообщений в консоль на этапе отладки можно использовать директиву `@debug`. В качестве примера выведем значение переменной:

```
$line-height: 1.5;
@debug "$line-height: " + $line-height;
```

После запуска преобразования сообщение отобразится в окне консоли:

```
C:\book\p1>sass scss/main.scss dist/css/main.css
scss\main.scss:2 Debug: $line-height: 1.5
```

Вывести предупреждающее сообщение позволяет директива `@warn`:

```
@warn "Предупреждающее сообщение";
```

После запуска преобразования сообщение отобразится в окне консоли:

```
C:\book\p1>sass scss/main.scss dist/css/main.css
Warning: Предупреждающее сообщение
 scss\main.scss 3:1 root stylesheet
```

С помощью флага `--quiet` или `-q` можно отключить вывод значений директив `@warn` и `@debug`:

```
C:\book\p1>sass --quiet scss/main.scss dist/css/main.css
```

Для вывода сообщения о фатальной ошибке предназначена директива `@error`:

```
@error "Описание ошибки";
```

После запуска преобразования сообщение отобразится в окне консоли, и выполнение преобразования прекратится:

```
C:\book\p1>sass scss/main.scss dist/css/main.css
Error: "Описание ошибки"
```

```
 | @error "Описание ошибки";
 | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
scss\main.scss 2:1 root stylesheet
```

Помимо вывода сообщения об ошибке в окне консоли, оно также записывается в итоговый CSS-файл в качестве значения для селектора `body::before`. Что позволяет увидеть сообщение об ошибке в окне веб-браузера. Если нужно отключить вывод сообщения в CSS-файл, то следует указать флаг `--no-error-css`:

```
C:\book\p1>sass --no-error-css scss/main.scss dist/css/main.css
```

При указании флага `--no-error-css` в случае возникновения ошибки CSS-файл может быть удален.

По умолчанию сообщение об ошибке в окне консоли выделяется красным цветом. На черном фоне сообщение красного цвета прочитать сложно. Для отключения выделения цветом нужно указать флаг `--no-color`:

```
C:\book\p1>sass --no-color scss/main.scss dist/css/main.css
```

По умолчанию в окно консоли выводятся символы из кодировки Unicode. Чтобы выводились только ASCII-символы, следует указать флаг `--no-unicode`:

```
C:\book\p1>sass --no-color --no-unicode scss/main.scss dist/css/main.css
```

## 5.2. Переменные и типы данных

Очень часто в CSS мы указываем одно и то же значение в разных местах файла. Если нужно изменить это значение, то приходится вносить корректировки в нескольких местах. В современных веб-браузерах эту проблему частично снимают *CSS-переменные*. Для создания глобальной CSS-переменной нужно объявить ее внутри блока `:root {}`. Перед названием CSS-переменной указываются два дефиса:

```
:root {
 --color-red: #ff0000;
}
```

При использовании просто указать имя CSS-переменной нельзя. Нужно передать ее название функции `var()`, что не очень удобно:

```
p {
 color: var(--color-red);
}
```

### 5.2.1. Именованние переменных в Sass

*Переменные* в SCSS-файле лишены такого недостатка. Создание переменной осуществляется по следующей схеме:

```
$<Имя переменной>: <Значение>;
```

Имя переменной начинается с символа доллара и может содержать латинские символы, цифры, дефис и подчеркивание. Регистр букв имеет значение. От использования других символов лучше отказаться. После двоеточия указывается значение переменной, а в самом конце ставится точка с запятой. Пример создания переменных:

```
$black: #000000;
$white: #ffffff;
```

Теперь эти переменные можно указывать в качестве значения CSS-атрибутов:

```
body {
 color: $white;
 background-color: $black;
}
```

После преобразования в CSS-файле останутся только значения переменных:

```
body {
 color: #ffffff;
 background-color: #000000;
}
```

Следует учитывать, что дефисы и подчеркивания в имени переменной взаимозаменяемы. Можно создать переменную с названием `$color-red`, а использовать название `$color_red`:

```
$color-red: #ff0000;
p {
 color: $color_red; // color: $color-red;
}
```

### 5.2.2. Области видимости переменных

При работе с переменными важно учитывать область видимости. Переменные, созданные вне блоков, видны внутри последующих блоков. Чтобы переменная была видна во всех блоках, она должна быть создана в самом начале файла. Такие переменные мы будем называть *глобальными*. Пример создания глобальных переменных и изменения их значений:

```
$color: #ff0000;
p {
 color: $color; // color: #ff0000;
 // Переменная $bg-color здесь не определена
 // background-color: $bg-color; // Ошибка
}
$color: #0000ff; // Изменяем значение
$bg-color: #ffffff;
span {
 color: $color; // color: #0000ff;
 background-color: $bg-color; // background-color: #ffffff;
}
```

Если переменная создана внутри блока, то ее видимость ограничена этим блоком. Такие переменные мы будем называть *локальными*. Пример создания локальной переменной:

```
p {
 $color-red: #ff0000;
 color: $color-red;
}
span {
 // Ошибка. Переменная здесь не видна
 // color: $color-red;
}
```

Обратите внимание: внутри второго блока содержатся только комментарии. Иными словами, после преобразования блок будет пустым. Пустые блоки в итоговый CSS-файл не добавляются.

### 5.2.3. Инструкция *!global*

Если внутри блока присваивается значение переменной, имя которой совпадает с именем глобальной переменной, то новое значение будет видно только внутри блока:

```
$color: #0000ff;
p {
 $color: #ff0000;
 color: $color; // color: #ff0000;
}
span {
 color: $color; // color: #0000ff;
}
```

Чтобы изменить значение глобальной переменной внутри блока, нужно после значения указать инструкцию *!global*:

```
$color: #0000ff;
p {
 $color: #ff0000 !global;
 color: $color; // color: #ff0000;
}
span {
 color: $color; // color: #ff0000;
}
```

### 5.2.4. Инструкция *!default*

Если нужно присвоить значение переменной только в том случае, если она не определена или имеет значение *null*, то после значения следует указать инструкцию *!default*:

```
$color: #0000ff;
p {
 $color: #ff0000 !default; // Значение игнорируется
 $bg-color: #ffffff !default; // ОК
 color: $color; // color: #0000ff;
 background-color: $bg-color; // background-color: #ffffff;
}
span {
 color: $color; // color: #0000ff;
}
```

#### **НА ЗАМЕТКУ**

При создании переменной всегда добавляйте инструкцию *!default*. В этом случае пользователь сможет подключить модуль в своем файле и изменить значения некоторых переменных, при этом не изменяя код модуля (см. разд. 5.10.9).

## 5.2.5. Типы данных

Переменные в Sass могут содержать данные следующих типов:

- ◆ **null** — означает отсутствие значения:

```
$headings-font-family: null;
@debug type-of($headings-font-family); // null
```

- ◆ **логическое значение (bool)** — переменная может содержать значение **true** (истина) или **false** (ложь):

```
$enable-shadow: true;
$enable-gradients: false;
@debug type-of($enable-shadow); // bool
@debug type-of($enable-gradients); // bool
```

- ◆ **число (number)** — значение может быть задано в целочисленном или вещественном форматах. После числа допускается указание единицы изменения:

```
$x: 10;
$y: 15.2;
$z: 12px; // Число с единицей измерения
$k: .3rem; // 0.3rem
$m: 80% !default;
@debug type-of($x); // number
@debug type-of($y); // number
@debug type-of($z); // number
@debug type-of($k); // number
@debug type-of($m); // number
```

- ◆ **строка (string)** — значение указывается внутри одинарных или двойных кавычек:

```
$s1: "bold";
$s2: 'bold';
@debug type-of($s1); // string
@debug type-of($s2); // string
```

- ◆ **строка без кавычек (string)** — значение указывается без кавычек вообще. Обратите внимание: именованный цвет также указывается как строка без кавычек, что может привести к результату, который не ожидался:

```
$s3: bold;
$color: red;
@debug type-of($s3); // string
@debug type-of($color); // color !!!
```

- ◆ **цвет (color)** — можно указать название цвета, значения в формате **#RGB** или **#RRGGBB**, а также задать значение с помощью функций: **rgb(R, G, B)**, **rgba(R, G, B, A)**, **hsl(H, S, L)** и **hsla(H, S, L, A)**:

```
$color1: red;
$color2: #f00;
$color3: #ff0000;
$color4: rgb(255, 0, 0);
$color5: rgb(100%, 0%, 0%);
$color6: rgba(255, 0, 0, 0.5);
```

```

$color7: rgba(100%, 0%, 0%, 0.5);
$color8: hsl(0, 100%, 50%);
$color9: hsla(0, 100%, 50%, 0.5);
@debug type-of($color1); // color
@debug type-of($color2); // color

```

- ◆ **список (list)** — значения, разделенные запятыми или пробелами. Дополнительно можно добавить квадратные или круглые скобки:

```

$list1: Consolas, "Courier New", monospace;
$list2: 5px 10px 15px 0;
$list3: [5px 10px 15px 0];
$list4: (5px 10px 15px 0);
$list5: [5px]; // Список из одного элемента
$list6: (10px,); // Список из одного элемента
$list7: []; // Пустой список
$list8: (); // Пустой список
@debug type-of($list1); // list
@debug type-of($list2); // list

```

- ◆ **ассоциативный массив (map)** — список пар **ключ: значение** внутри круглых скобок. Ключ должен быть уникальным. Пары разделяются запятыми:

```

$colors: (
 "blue": #007bff,
 "red": #dc3545
);
$colors2: (); // Пустой ассоциативный массив
@debug type-of($colors); // map
@debug type-of($colors2); // list

```

- ◆ **ссылка на функцию (function):**

```

@function add($x, $y) {
 @return $x + $y;
}
$func: get-function("add");
@debug $func; // get-function("add")
@debug type-of($func); // function

```

- ◆ **список аргументов (arglist):**

```

@function func($numbers...) {
 @debug type-of($numbers); // arglist
 @return 0;
}
@debug func(1, 2, 3); // 0

```

**Функции** `meta.type-of($value)` и `type-of($value)` позволяют определить тип значения или переменной. Они возвращают строку без кавычек с наименованием типа данных:

```

@use "sass:meta";

$x: 10;
$s1: "bold";
@debug type-of($x); // number

```

```
@debug meta.type-of($s1); // string
$value: meta.type-of($x);
@debug type-of($value); // string
@debug type-of(10); // number
```

Функции `meta.inspect($value)` и `inspect($value)` преобразуют любое значение в строку без кавычек в целях отладки:

```
@use "sass:meta";

@debug meta.inspect(10px); // 10px
@debug type-of(meta.inspect(10px)); // string
@debug inspect([5px 10px 15px 0]); // [5px 10px 15px 0]
@debug type-of(inspect([5px 10px 15px 0])); // string
```

### 5.2.6. Подстановка значений переменных

Значение переменной можно подставить в комментарий, строку, селектор и т. д. Для этого нужно добавить следующую конструкцию (название переменной указывается внутри фигурных скобок):

```
#{<Переменная или выражение>}
```

Пример подстановки значения переменной в комментарий:

```
$version: "1.0.0";
/* Версия: #{ $version} */
```

Результат в CSS-файле:

```
/* Версия: 1.0.0 */
```

Вставим стилевой класс в селектор из значения переменной:

```
$class-name: "cls1";
p.#{ $class-name} {
 color: red;
}
```

Результат в CSS-файле:

```
p.cls1 {
 color: red;
}
```

Подставим значение в строку и на основе строк сформируем названия атрибутов:

```
$border: "border";
$color: "color";
$border-color: "#{ $border}-#{ $color}";
p {
 #{ $border}-color: red;
}
div {
 #{ $border-color}: blue;
}
```

Результат в CSS-файле:

```
p {
 border-color: red;
}
div {
 border-color: blue;
}
```

Подстановку можно также использовать, если присутствуют операторы и не нужно вычислять значение выражения:

```
$line-height: 1.5;
p {
 font: 1rem/#{$line-height};
}
```

Результат в CSS-файле:

```
p {
 font: 1rem/1.5;
}
```

Если бы просто указали название переменной:

```
font: 1rem/$line-height;
```

то выражение было бы вычислено:

```
font: 0.6666666667rem;
```

Если нужно добавить к значению переменной единицу измерения, то не следует это делать с помощью подстановки. В этом примере мы получим строку, а не число:

```
C:\book\pl>sass --interactive
>> $x: 5
5
>> $y: #{$x}px
5px
>> $y + 1
5px1
```

Вместо сложения мы получили операцию конкатенации строки с числом. Чтобы получить именно число, следует умножить значение на 1 с единицей измерения:

```
>> $x: 5
5
>> $y: $x * 1px
5px
>> $y + 1
6px
```

Если выполняется подстановка значения `null`, то оно трактуется как пустая строка:

```
>> $value: null
null
>> "str #{$value} str"
"str str"
```

Следует учитывать, что при подстановке удаляются все кавычки:

```
$font-family-monospace: Consolas, "Courier New", monospace !default;
.monospace {
 font-family: #{font-family-monospace};
}
```

Результат в CSS-файле:

```
.monospace {
 font-family: Consolas, Courier New, monospace;
}
```

Если нужно сохранить кавычки, то следует воспользоваться функцией

`meta.inspect($value)` или `inspect($value)`:

```
@use "sass:meta";
```

```
$font-family-monospace: Consolas, "Courier New", monospace !default;
.monospace {
 font-family: #{meta.inspect($font-family-monospace)};
}
```

Результат в CSS-файле:

```
.monospace {
 font-family: Consolas, "Courier New", monospace;
}
```

## 5.2.7. Проверка существования переменной

Проверить существование переменной в текущей области видимости позволяют функции `meta.variable-exists(<ИМЯ>)` и `variable-exists(<ИМЯ>)`. Название переменной указывается в виде строки без символа `$`. Функции возвращают значение `true`, если переменная с указанным названием существует, и `false` — в противном случае:

```
@use "sass:meta";
```

```
@debug meta.variable-exists("x"); // false
$x: 10;
div {
 $width: 100px;
 width: $width;
 @debug variable-exists("x"); // true
 @debug variable-exists("width"); // true
}
@debug meta.variable-exists("x"); // true
@debug meta.variable-exists("width"); // false
```

Функции `meta.global-variable-exists()` и `global-variable-exists()` позволяют проверить существование переменной с указанным именем в глобальном пространстве имен или внутри области видимости `$module`. Форматы функций:

```
@use "sass:meta";
meta.global-variable-exists(<ИМЯ>[, $module: null])
global-variable-exists(<ИМЯ>[, $module: null])
```

Название переменной указывается в виде строки без символа `§`. Функции возвращают значение `true`, если переменная с указанным названием существует, и `false` — в противном случае:

```
@use "sass:meta";
@use "sass:math";

@debug meta.global-variable-exists("x"); // false
$x: 10;
div {
 $width: 100px;
 width: $width;
 @debug global-variable-exists("x"); // true
 @debug global-variable-exists("width"); // false
}
@debug meta.global-variable-exists("x"); // true
@debug meta.global-variable-exists("width"); // false
@debug global-variable-exists("pi", $module: "math"); // true
```

## 5.3. Операторы и циклы

*Операторы* позволяют выполнить определенные действия с данными. Например, математические операторы предназначены для арифметических вычислений, а условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнять его.

### 5.3.1. Математические операторы

Производить арифметические вычисления позволяют следующие операторы:

◆ + — сложение:

```
C:\book\pl>sass --interactive
>> 10px + 15px
25px
>> 10px + 15
25px
>> 10px + 1in
106px
```

◆ - — вычитание:

```
>> 10px - 5px
5px
>> 10px - 5
5px
```

◆ - — унарный минус:

```
>> $x: 10px
10px
>> -$x
-10px
```

## ◆ \* — умножение:

```
>> 10px * 5
50px
>> 10px * 5px
50px*px
```

## ◆ / — деление:

```
>> 10px / 5px
10px/5px
```

Как видно из результата, деление выполнено не было, т. к. символ / в CSS используется для разделения значений. Поэтому просто оператор / иногда указать недостаточно. Чтобы выполнить именно деление, нужно либо заключить выражение в круглые скобки, либо сохранить результат в переменной, либо вернуть результат из функции:

```
>> (10px / 5px)
2
>> $x: 10px / 5px
2
>> (10px / 3)
3.3333333333px
>> (10px / 3.0)
3.3333333333px
```

Если в выражении используются другие операторы или переменные, то будет выполнено деление:

```
>> 50px / 100px * 100%
50%
>> $x: 10px
10px
>> 50px / $x
5
```

Если не нужно выполнять деление, то следует выполнить подстановку значения (см. *разд. 5.2.6*):

```
>> $x: 10px
10px
>> 50px / #{ $x }
50px/10px
>> #{40px + 10px} / $x
50px/10px
```

Деление целого числа на 0 приведет к неопределенности. Деление вещественного числа на 0 приведет к ошибке:

```
>> (10px / 0)
>> (10.5px / 0)
Unexpected exception:
Unsupported operation: Infinity.round()
```

## ◆ % — остаток от деления:

```
>> 10px % 2 // 0px (10 - 10 / 2 * 2)
0px
```

```
>> 10px % 3 // 1px (10 - 10 / 3 * 3)
1px
>> 10px % 4 // 2px (10 - 10 / 4 * 4)
2px
>> 10px % 6 // 4px (10 - 10 / 6 * 6)
4px
```

В выражении должны участвовать совместимые единицы измерения. Если это не так, то будет выведено сообщение об ошибке:

```
>> 10px + 20%
^^^^^^^^^^
```

Error: Incompatible units % and px.

При умножении можно получить единицы измерения в квадрате:

```
>> 10px * 2px
20px*px
```

В интерактивном режиме ошибки не будет, но при попытке вывода в CSS-файл сообщение об ошибке обязательно отобразится:

```
C:\book\p1>sass scss/main.scss dist/css/main.css
Error: 20px*px isn't a valid CSS value.
```

Если в выражении участвуют несколько единиц измерения, то на выходе должна получиться только одна единица измерения. В этом примере на первом этапе при делении единицы измерения сокращаются и на втором этапе происходит умножение простого числа на значение в процентах:

```
>> 50px / 100px * 100%
50%
```

Чтобы получить число с единицей измерения, следует умножить значение на 1 с единицей измерения. Использовать для этого подстановку значения нельзя, т. к. вместо числа получим строку:

```
>> $x: 5
5
>> $x: $x * 1px
5px
>> $x + 1
6px
```

### 5.3.2. Приоритет выполнения операторов

Все операторы выполняются в порядке приоритета. Вначале вычисляется выражение, в котором оператор имеет наивысший приоритет, а затем выражение с меньшим приоритетом. Например, выражение с оператором умножения будет выполнено раньше выражения с оператором сложения, т. к. приоритет оператора умножения выше. Если приоритет операторов одинаковый, то используется порядок вычисления, определенный для конкретного оператора. Изменить последовательность вычисления выражения можно с помощью круглых скобок. Пример:

```
C:\book\p1>sass --interactive
>> 5 + 10 * 3 / 2 // Умножение -> деление -> сложение
20
>> (5 + 10) * 3 / 2 // Сложение -> умножение -> деление
22.5
```

### 5.3.3. Операторы для работы со строками

Если справа или слева от оператора + расположена строка, то будет выполнена операция *конкатенации* строк:

```
C:\book\p1>sass --interactive
>> "str" + "ing"
"string"
>> bo + ld
bold
>> "str" + 1
"str1"
>> 1 + "px"
"1px"
```

Если к строке в кавычках прибавить строку без кавычек, то результатом будет строка в кавычках:

```
>> "str" + ing
"string"
```

Если к строке без кавычек прибавить строку в кавычках, то результатом будет строка без кавычек:

```
>> str + "ing"
string
```

Для конкатенации строк можно также использовать подстановку значений (см. *разд. 5.2.6*):

```
>> #{"str"}#{"ing"}
string
>> #{"str"}#{1}
str1
>> #{1}#{"px"}
1px
```

При использовании со строкой оператора - возвращается строка со значениями, разделенными символом -:

```
>> font - size
font-size
>> "font" - "size"
"font"- "size"
```

Если значение указано только справа от оператора -, то возвращается строка, которая начинается с символа -, за которым следует значение:

```
>> - value
-value
>> - "value"
-"value"
```

При использовании со строкой оператора / возвращается строка со значениями, разделенными символом /:

```
>> #{10px + 2px} / 15px
12px/15px
```

Если значение указано только справа от оператора /, то возвращается строка, которая начинается с символа /, за которым следует значение:

```
>> / 5px
/5px
```

### 5.3.4. Операторы сравнения

Операторы сравнения используются в логических выражениях. Приведем операторы сравнения, доступные в Sass:

- ◆ == — равно;
- ◆ != — не равно;
- ◆ < — меньше;
- ◆ > — больше;
- ◆ <= — меньше или равно;
- ◆ >= — больше или равно.

Логические выражения возвращают только два значения: `true` (истина) или `false` (ложь). Пример вывода значения логического выражения:

```
C:\book\p1>sass --interactive
>> 10px == 10px
true
>> 10px == 5px
false
>> 10px != 5px
true
>> 10px < 5px
false
>> 10px > 5px
true
>> 10px <= 5px
false
>> 10px >= 5px
true
```

Два значения равны, если они имеют одинаковый тип данных и то же самое значение. Числа равны, если равны их значения и единицы измерения:

```
>> 1px == 1px
true
>> 1px == "1px"
false
>> 1px == 1
false
```

Строки в кавычках и без кавычек с одинаковым содержимым считаются равными. Регистр букв имеет значение:

```
>> "abc" == abc
true
```

```
>> "abc" == abcd
false
>> abc == ABC
false
```

Два цвета считаются равными, если равны значения всех каналов, включая альфа-канал:

```
>> #ff0000 == #ff0000
true
>> #ff0000 == #FF0000
true
>> rgba(255, 0, 0, 0.5) == rgba(255, 0, 0, 0.5)
true
>> rgba(255, 0, 0, 0.5) == rgba(255, 0, 0, 0.8)
false
```

Списки равны, если их элементы одинаковые:

```
>> [1 2 3] == [1 2 3]
true
>> [1 2 3] == [1 2 4]
false
```

Списки с элементами, указанными через пробел, не равны спискам с элементами, указанными через запятую:

```
>> [1 2 3] == [1, 2, 3]
false
```

Списки в квадратных скобках не равны спискам в круглых скобках и спискам без скобок:

```
>> [1 2 3] == (1 2 3)
false
>> $list: 1 2 3
1 2 3
>> [1 2 3] == $list
false
```

Ассоциативные массивы равны, если их ключи и значения равны:

```
>> $map1: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> $map2: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> $map1 == $map2
true
>> $map3: ("blue": #007bff, "red": #ff0000)
("blue": #007bff, "red": #ff0000)
>> $map1 == $map3
false
```

Значения true, false и null равны только самим себе:

```
>> true == true
true
>> false == false
true
```

```
>> true != false
true
>> null == null
true
>> $value: null
null
>> $value == null
true
```

Значение логического выражения можно инвертировать с помощью оператора `not`. В этом случае если логическое выражение возвращает `false`, то `not false` вернет значение `true`:

```
>> not true
false
>> not false
true
```

Если оператор `not` указать дважды, то любое значение преобразуется в логическое. Значения `false` и `null` трактуются как `false`, а все остальные значения (включая `0`, пустые строки и списки без элементов) трактуются как `true`:

```
>> not not false
false
>> not not null
false
>> not not 0
true
>> not not ""
true
>> not not []
true
>> not not 10px
true
```

Несколько логических выражений можно объединить в одно большое с помощью следующих операторов:

◆ `and` — логическое *и*. Логическое выражение вернет `true` только в случае, если оба подвыражения вернут `true`:

```
>> true and true
true
>> true and false
false
```

◆ `or` — логическое *или*. Логическое выражение вернет `true`, если хотя бы одно из подвыражений вернет `true`:

```
>> true or true
true
>> true or false
true
>> false or true
true
>> false or false
false
```

### 5.3.5. Оператор ветвления `@if` и функция `if()`

Оператор ветвления `@if` позволяет в зависимости от значения логического выражения выполнить отдельный блок программы или, наоборот, не выполнять его. Оператор имеет следующий формат:

```
@if <Логическое выражение> {
 <Блок, выполняемый, если условие истинно>
}
[@else {
 <Блок, выполняемый, если условие ложно>
}]
```

Если логическое выражение возвращает значение `true`, то выполняются инструкции, расположенные внутри фигурных скобок, сразу после оператора `@if`. Если логическое выражение возвращает значение `false`, то выполняются инструкции после директивы `@else`. Блок `@else` является необязательным. Пример проверки условия и вывода соответствующего сообщения в окно консоли:

```
$size: 10px;
@if $size > 5px {
 @debug "$size больше 5px"; // Выведет: $size больше 5px
}
@else {
 @debug "$size меньше 5px";
}
```

Логическое выражение может не содержать операторов сравнения вообще. В этом случае значения `false` и `null` трактуются как `false`, а все остальные значения (включая `0`, пустые строки и списки без элементов) трактуются как `true`:

```
$test: true;
@if $test {
 @debug "$test == true"; // Выведет: $test == true
}
```

Чтобы проверить несколько условий, можно воспользоваться следующим форматом:

```
@if <Условие 1> {
 <Блок 1>
} @else if <Условие 2> {
 <Блок 2>
} @else if <Условие N> {
 <Блок N>
} @else {
 <Блок else>
}
```

Если `<Условие 1>` истинно, то выполняется `<Блок 1>`, а все остальные условия пропускаются. Если `<Условие 1>` ложно, то проверяется `<Условие 2>`. Если `<Условие 2>` истинно, то выполняется `<Блок 2>`, а все остальные условия пропускаются. Если `<Условие 2>` ложно, то точно также проверяются остальные условия. Если все условия

ложны, то выполняется <Блок *else*>. В качестве примера определим, какое число от 0 до 2 содержится в переменной:

```
$x: 2;
@if $x == 0 {
 @debug "$x == 0";
} @else if $x == 1 {
 @debug "$x == 1";
} @else if $x == 2 {
 @debug "$x == 2"; // Выведет: $x == 2
} @else {
 @debug "Другое число";
}
```

Для проверки условия вместо оператора `@if` можно использовать функцию `if()`. Функция имеет следующий формат:

```
<Переменная>: if(<Логическое выражение>, <Выражение если true>,
 <Выражение если false>);
```

Если логическое выражение возвращает значение `true`, то выполняется выражение, расположенное во втором параметре. Если логическое выражение возвращает значение `false`, то выполняется выражение, расположенное в третьем параметре. Результат выполнения выражения становится результатом выполнения функции.

Пример:

```
$y: if(true, 5, 10);
@debug $y; // Выведет: 5
$z: if(false, 5, 10);
@debug $z; // Выведет: 10
```

### 5.3.6. Цикл `@for`

Цикл `@for` используется для выполнения инструкций определенное число раз. Цикл имеет следующие форматы:

```
@for <Переменная> from <Начальное значение> to <Конечное значение> {
 <Инструкции>
}
@for <Переменная> from <Начальное значение> through <Конечное значение> {
 <Инструкции>
}
```

На каждой итерации цикла в переменной сохраняется текущее значение от <Начальное значение> до <Конечное значение>. Если используется ключевое слово `to`, то конечное значение не входит в диапазон:

```
@for $i from 1 to 5 {
 @debug $i;
}
```

Результат в окне консоли:

```
scss\main.scss:2 Debug: 1
scss\main.scss:2 Debug: 2
scss\main.scss:2 Debug: 3
scss\main.scss:2 Debug: 4
```

Если используется ключевое слово `through`, то конечное значение входит в диапазон:

```
@for $i from 1 through 5 {
 @debug $i;
}
```

Результат в окне консоли:

```
scss\main.scss:2 Debug: 1
scss\main.scss:2 Debug: 2
scss\main.scss:2 Debug: 3
scss\main.scss:2 Debug: 4
scss\main.scss:2 Debug: 5
```

Если начальное значение больше конечного значения, то на каждой итерации цикла текущее значение уменьшается на единицу:

```
@for $i from 5 to 1 {
 @debug $i;
}
```

Результат в окне консоли:

```
scss\main.scss:2 Debug: 5
scss\main.scss:2 Debug: 4
scss\main.scss:2 Debug: 3
scss\main.scss:2 Debug: 2
```

### 5.3.7. Цикл `@while`

Выполнение инструкций в цикле `@while` продолжается до тех пор, пока логическое выражение истинно. Цикл имеет следующий формат:

```
<Начальное значение>
@while <Условие> {
 <Инструкции>
 <Приращение>
}
```

Последовательность работы цикла `@while`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие, и если оно истинно, выполняются инструкции внутри цикла, иначе выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Переход к п. 2.

#### **ВНИМАНИЕ!**

Если `<Приращение>` не указано, то цикл будет бесконечным. Способа досрочно остановить цикл из кода в Sass не существует. Старайтесь избегать использования цикла `@while`.

Выведем все числа от 1 до 5, используя цикл `@while`:

```
$i: 1; // <Начальное значение>
@while $i <= 5 { // <Условие>
 @debug $i; // <Инструкции>
 $i: $i + 1; // <Приращение>
}
```

Результат в окне консоли:

```
scss\main.scss:3 Debug: 1
scss\main.scss:3 Debug: 2
scss\main.scss:3 Debug: 3
scss\main.scss:3 Debug: 4
scss\main.scss:3 Debug: 5
```

Выражение, указанное в параметре *<Приращение>*, может не только увеличивать значение переменной-счетчика, но и уменьшать его. Кроме того, значение может изменяться на любую величину. Выведем числа в обратном порядке:

```
$i: 5;
@while $i > 0 {
 @debug $i;
 $i: $i - 1;
}
```

Результат в окне консоли:

```
scss\main.scss:3 Debug: 5
scss\main.scss:3 Debug: 4
scss\main.scss:3 Debug: 3
scss\main.scss:3 Debug: 2
scss\main.scss:3 Debug: 1
```

Логическое выражение может не содержать операторов сравнения вообще. В этом случае значения `false` и `null` трактуются как `false`, а все остальные значения (включая `0`, пустые строки и списки без элементов) трактуются как `true`:

```
$run: true;
$i: 1;
@while $run {
 @debug $i;
 $i: $i + 1;
 @if $i > 4 {
 $run: false;
 }
}
```

Результат в окне консоли:

```
scss\main.scss:4 Debug: 1
scss\main.scss:4 Debug: 2
scss\main.scss:4 Debug: 3
scss\main.scss:4 Debug: 4
```

### 5.3.8. Цикл `@each`: перебор элементов списка или ассоциативного массива

Цикл `@each` позволяет перебирать элементы списка или ассоциативного массива. Цикл имеет следующие форматы:

```
@each <Переменная> in <Список или ассоциативный массив> {
 <Инструкции>
}
```

```
@each <Переменная 1>, ..., <Переменная N> in
 <Список или ассоциативный массив> {
 <Инструкции>
}
```

При использовании первого формата на каждой итерации цикла переменной *<Переменная>* присваивается значение текущего элемента списка. Цикл завершится, когда будут перебраны все элементы списка:

```
@each $value in [10, 20, 30] {
 @debug $value;
}
```

Результат в окне консоли:

```
scss\main.scss:2 Debug: 10
scss\main.scss:2 Debug: 20
scss\main.scss:2 Debug: 30
```

Второй формат позволяет получить доступ к ключам и значениям ассоциативного массива. На каждой итерации в первой переменной сохраняется ключ, а во второй переменной — значение, соответствующее ключу:

```
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@each $key, $value in $colors {
 @debug "#{$key} => #{$value}";
}
```

Результат в окне консоли:

```
scss\main.scss:6 Debug: blue => #007bff
scss\main.scss:6 Debug: red => #dc3545
```

Если указана только одна переменная, то в ней сохраняется список из ключа и значения:

```
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@each $list in $colors {
 @debug $list;
}
```

Результат в окне консоли:

```
scss\main.scss:6 Debug: "blue" #007bff
scss\main.scss:6 Debug: "red" #dc3545
```

Второй формат цикла `@each` можно также использовать для получения доступа к значениям вложенных списков:

```
@each $v1, $v2, $v3 in (1, 2, 3), (4, 5, 6) {
 @debug "#{$v1} - #{$v2} - #{$v3}";
}
```

Результат в окне консоли:

```
scss\main.scss:2 Debug: 1 - 2 - 3
scss\main.scss:2 Debug: 4 - 5 - 6
```

Если какой-либо переменной не достанется значения, то она будет иметь значение null:

```
@each $v1, $v2, $v3 in (1, 2, 3), (4, 5) {
 @debug $v3;
}
```

Результат в окне консоли:

```
scss\main.scss:2 Debug: 3
scss\main.scss:2 Debug: null
```

## 5.4. Числа

В Sass числовое значение может быть задано в целочисленном или вещественном форматах:

```
C:\book\p1>sass --interactive
>> $x: 10
10
>> type-of($x)
number
>> $y: 10.0
10
>> $z: 15.2
15.2
>> $k: 3.1e5
310000
>> $n: 3e-5
0.00003
```

Sass не разделяет целые и вещественные числа. Деление целых чисел всегда возвращает вещественное число:

```
>> (10 / 3)
3.3333333333
```

После числа может быть указана единица изменения из CSS (после значения 0 единицу измерения добавлять не нужно):

```
>> $x: 12px
12px
>> $y: 1.5rem
1.5rem
```

Чтобы получить число с единицей измерения, следует умножить значение на 1 с единицей измерения. Использовать для этого подстановку значения нельзя, т. к. вместо числа получим строку:

```
>> $x: 5
5
```

```
>> $x: $x * 1px
5px
>> $x + 1
6px
```

Для работы с числами в Sass предназначен модуль `math`. Подключить модуль позволяет следующая инструкция (в интерактивном режиме точку с запятой указывать не нужно):

```
@use "sass:math";
```

Для доступа к идентификаторам внутри модуля следует указать название модуля, точку, а затем идентификатор:

```
@debug math.$pi; // 3.1415926536
```

Некоторые идентификаторы доступны в глобальной области видимости. Перед такими идентификаторами мы не будем указывать название модуля в следующих разделах.

### 5.4.1. Математические константы

В модуле `math` определены следующие математические константы:

◆ `math.$pi` — число  $\pi$ :

```
C:\book\pl>sass --interactive
>> @use "sass:math"
>> math.$pi
3.1415926536
```

◆ `math.$e` — значение константы  $e$ :

```
>> math.$e
2.7182818285
```

### 5.4.2. Основные функции для работы с числами

Приведем основные функции для работы с числами:

◆ `math.abs(<Число>)` и `abs(<Число>)` — возвращают абсолютное значение:

```
C:\book\pl>sass --interactive
>> @use "sass:math"
>> math.abs(5px)
5px
>> math.abs(-5px)
5px
>> abs(-5px)
5px
```

◆ `math.pow(x, y)` — возводит число  $x$  в степень  $y$  (числа не должны содержать единицы измерения):

```
>> math.pow(10, 2)
100
>> math.pow(3, 3)
27
```

- ◆ `math.sqrt(<Число>)` — квадратный корень (число не должно содержать единицу измерения):

```
>> math.sqrt(100)
10
>> math.sqrt(25)
5
```

- ◆ `math.log(<Число>[, $base: null])` — логарифм (числа не должны содержать единицы измерения). Если второй параметр не указан, то вычисляется натуральный логарифм. Если во втором параметре указать значение 10, то вычисляется десятичный логарифм:

```
>> math.log(10)
2.302585093
>> math.log(10, 10)
1
```

- ◆ `math.max(<Значения>...)` — максимальное значение:

```
>> math.max(10px, 3px)
10px
>> math.max(10px, 3px, 20px)
20px
```

Существует также глобальная функция `max(<Значения>...)`, но ее название совпадает с одноименной функцией из CSS. Если формат вызова совпадает с форматом функции из CSS, то мы получим строку:

```
>> max(10px, 3px)
max(10px, 3px)
>> type-of(max(10px, 3px))
string
```

Если формат вызова не совпадает с форматом функции из CSS, то получим максимальное значение:

```
>> max((10px, 3px)...)
10px
```

- ◆ `math.min(<Значения>...)` — минимальное значение:

```
>> math.min(10px, 3px)
3px
>> math.min(10px, 3px, 2px)
2px
```

Существует также глобальная функция `min(<Значения>...)`, но ее название совпадает с одноименной функцией из CSS. Если формат вызова совпадает с форматом функции из CSS, то мы получим строку:

```
>> min(10px, 3px)
min(10px, 3px)
>> type-of(min(10px, 3px))
string
```

Если формат вызова не совпадает с форматом функции из CSS, то получим минимальное значение:

```
>> min((10px, 3px)...)
3px
```

- ◆ `math.clamp(<Min>, <Число>, <Max>)` — если `<Число>` меньше `<Min>`, то возвращается значение `<Min>`. Если `<Число>` больше `<Max>`, то возвращается значение `<Max>`. В противном случае возвращается значение `<Число>`. Значения, указанные в параметрах, должны иметь совместимые единицы измерения или не иметь их вообще:

```
>> math.clamp(5, 7, 10)
7
>> math.clamp(5, 3, 10)
5
>> math.clamp(5px, 20px, 10px)
10px
```

- ◆ `math.hypot(<Значения>...)` — возвращает квадратный корень из суммы квадратов значений. Значения, указанные в параметрах, должны иметь совместимые единицы измерения или не иметь их вообще:

```
>> math.hypot(2, 3)
3.6055512755
>> math.sqrt(math.pow(2, 2) + math.pow(3, 2))
3.6055512755
>> math.hypot(2px, 3px)
3.6055512755px
```

### 5.4.3. Округление чисел

Для округления чисел предназначены следующие функции:

- ◆ `math.ceil(<Число>)` и `ceil(<Число>)` — возвращают значение, округленное до ближайшего большего значения:

```
C:\book\pl>sass --interactive
>> @use "sass:math"
>> math.ceil(1.49)
2
>> math.ceil(1.5)
2
>> math.ceil(1.51)
2
>> ceil(1.51px)
2px
```

- ◆ `math.floor(<Число>)` и `floor(<Число>)` — возвращают значение, округленное до ближайшего меньшего значения:

```
>> math.floor(1.49)
1
>> math.floor(1.5)
1
>> math.floor(1.51)
1
>> floor(1.51px)
1px
```

- ◆ `math.round(<Число>)` и `round(<Число>)` — возвращают число, округленное до ближайшего меньшего целого (для чисел с дробной частью, меньше 0.5), или значение, округленное до ближайшего большего целого (для чисел с дробной частью больше или равной 0.5):

```
>> math.round(1.49)
1
>> math.round(1.5)
2
>> math.round(1.51)
2
>> round(1.51px)
2px
```

#### 5.4.4. Тригонометрические функции

В Sass доступны следующие основные тригонометрические функции из модуля `math` (полный список функций вы найдете в документации):

- ◆ `math.sin(<Угол>)`, `math.cos(<Угол>)`, `math.tan(<Угол>)` — стандартные тригонометрические функции (синус, косинус, тангенс). Угол без единицы измерения считается заданным в радианах:

```
C:\book\pl>sass --interactive
>> @use "sass:math"
>> math.sin(90deg)
1
>> $degrees: 90.0
90
>> // Перевод градусов в радианы
>> $radians: $degrees * (math.$pi / 180.0)
1.5707963268
>> math.sin($radians)
1
>> math.cos(180deg)
-1
>> math.tan(180deg)
0
```

- ◆ `math.asin(<Число>)`, `math.acos(<Число>)`, `math.atan(<Число>)` — обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Число задается без единиц измерения. Значение возвращается в градусах с единицей измерения:

```
>> math.asin(0.5)
30deg
>> math.acos(0.5)
60deg
>> math.atan(30)
88.090847567deg
```

## 5.4.5. Работа с единицами измерения

Для работы с единицами измерения предназначены следующие функции:

- ◆ `math.unit(<Число>)` и `unit(<Число>)` — возвращают строку в кавычках, содержащую единицу измерения. Если единица измерения не задана, то возвращается пустая строка. Функция предназначена для отладки. Пример:

```
C:\book\pl>sass --interactive
>> @use "sass:math"
>> math.unit(10)
""
>> math.unit(10px)
"px"
>> math.unit(10px * 5px)
"px*px"
>> unit(10px)
"px"
```

- ◆ `math.is-unitless(<Число>)` и `unitless(<Число>)` — возвращают значение `true`, если число не содержит единиц измерения, и `false` — в противном случае:

```
>> math.is-unitless(10)
true
>> math.is-unitless(10px)
false
>> unitless(10)
true
>> unitless(10px)
false
```

Чтобы получить число с единицей измерения, следует умножить значение на 1 с единицей измерения:

```
$x: 5;
@if unitless($x) {
 $x: $x * 1px;
}
@debug $x; // 5px
@debug $x + 1px; // 6px
```

- ◆ `math.compatible(<Число1>, <Число2>)` и `comparable(<Число1>, <Число2>)` — возвращают значение `true`, если числа имеют совместимые единицы измерения, и `false` — в противном случае:

```
>> math.compatible(10px, 5px)
true
>> math.compatible(10px, 5in)
true
>> math.compatible(10px, 5%)
false
>> comparable(10px, 5px)
true
>> comparable(10px, 5mm)
```

```

true
>> comparable(10px, 5%)
false

```

- ◆ `math.percentage(<Число>)` и `percentage(<Число>)` — возвращают результат умножения числа без единицы измерения (обычно вещественное число от 0 до 1) на 100 процентов:

```

>> math.percentage(0.02)
2%
>> math.percentage(0.2)
20%
>> percentage(0.2)
20%
>> percentage(20)
2000%

```

### 5.4.6. Преобразование числа в строку

Преобразовать число в строку можно следующим образом:

```

C:\book\p1>sass --interactive
>> "" + 2px
"2px"

```

Вместо конкатенации можно использовать подстановку значения переменной (см. *разд. 5.2.6*):

```

>> $x: 2px
2px
>> "$x = #{ $x }"
"$x = 2px"

```

### 5.4.7. Генерация псевдослучайных чисел

Для генерации псевдослучайных чисел в Sass используются функции `math.random(<ЛИМИТ>)` и `random(<ЛИМИТ>)`. Если параметр не указан, то генерируется случайное вещественное число от 0 до 1:

```

C:\book\p1>sass --interactive
>> @use "sass:math"
>> math.random()
0.9886102641
>> math.random()
0.196824348
>> random()
0.8112459187
>> random()
0.4173254478

```

Если в качестве параметра указано число больше или равное 1, то генерируется целое число от 1 до `<ЛИМИТ>`:

```

>> math.random(5)
2

```

```
>> math.random(5)
1
>> math.random(5)
5
>> random(5)
1
>> random(5)
3
```

## 5.5. Списки

*Список* — это набор значений, разделенных запятыми или пробелами. Значение в списке называется *элементом*, а его позиция в списке задается *индексом*. Обратите внимание: в Sass элементы списков нумеруются с 1. Списки в Sass являются неизменяемыми, т. е. можно получить значение элемента, но изменить его нельзя. Все функции возвращают новый список, а не изменяют текущий.

Для работы со списками в Sass предназначен модуль `list`. Подключить модуль позволяет следующая инструкция (в интерактивном режиме точку с запятой указывать не нужно):

```
@use "sass:list";
```

Для доступа к идентификаторам внутри модуля, следует указать название модуля, точку, а затем идентификатор:

```
@debug list.length(10px 20px 30px); // 3
```

Некоторые идентификаторы доступны в глобальной области видимости. Перед такими идентификаторами в следующих разделах мы не будем указывать название модуля.

### 5.5.1. Создание списка

Для создания пустого списка нужно указать квадратные или круглые скобки:

```
$list1: []; // Пустой список
$list2: () !default; // Пустой список
@debug type-of($list1); // list
```

Чтобы создать список из одного элемента, следует либо указать значение в квадратных скобках, либо добавить значение и запятую внутри круглых скобок:

```
$list1: [5px]; // Список из одного элемента
$list2: (10px,); // Список из одного элемента
```

На самом деле любое одиночное значение считается списком из одного элемента:

```
$list: 5px; // Список из одного элемента
@debug length($list); // 1
```

Если список состоит из нескольких значений, то они разделяются запятыми или пробелами. Дополнительно можно добавить квадратные или круглые скобки:

```
$list1: Consolas, "Courier New", monospace;
$list2: 5px 10px 15px 0;
```

```
$list3: [5px 10px 15px 0];
$list4: (5px 10px 15px 0) !default;
```

### Пример создания списка со значениями по умолчанию:

```
$list: () !default;
$list: join((10px 20px 30px), $list);
@each $value in $list {
 @debug $value;
}
```

### Результат в окне консоли:

```
scss\main.scss:4 Debug: 10px
scss\main.scss:4 Debug: 20px
scss\main.scss:4 Debug: 30px
```

Функции `list.separator(<Список>)` и `list-separator(<Список>)` позволяют определить, какие символы являются разделителями элементов списка. Они возвращают строку без кавычек со значением `space` (пробел) или `comma` (запятая):

```
C:\book\p1>sass --interactive
>> @use "sass:list"
>> list.separator(1px 2px)
space
>> list.separator((1px, 2px))
comma
>> list.separator(5px)
space
>> list.separator([])
space
>> list-separator(1px 2px)
space
>> list-separator([1px, 2px])
comma
```

Функции `list.is-bracketed(<Список>)` и `is-bracketed(<Список>)` возвращают значение `true`, если используются квадратные скобки, и `false` — в противном случае:

```
>> list.is-bracketed(10px 20px)
false
>> list.is-bracketed([10px, 20px])
true
>> is-bracketed(10px 20px)
false
>> is-bracketed([10px 20px])
true
```

Списки могут быть вложенными. Пример создания списков из двух элементов, которые также являются списками:

```
$list1: (1, 2, 3), (4, 5, 6);
$list2: (1 2 3) (4 5);
$list3: [1, 2, 3], [4, 5, 6];
$list4: 1 2 3, 4 5;
```





Результат в окне консоли:

```
scss\main.scss:2 Debug: 3
scss\main.scss:2 Debug: null
```

Выполнить перебор значений можно также с помощью цикла `@for` (см. *разд. 5.3.6*):

```
$list: 10px 20px 30px;
$list-len: length($list);
@for $i from 1 through $list-len {
 @debug nth($list, $i);
}
```

Результат в окне консоли:

```
scss\main.scss:4 Debug: 10px
scss\main.scss:4 Debug: 20px
scss\main.scss:4 Debug: 30px
```

Выведем элементы в обратном порядке:

```
$list: 10px 20px 30px;
$list-len: length($list);
@for $i from $list-len through 1 {
 @debug nth($list, $i);
}
```

Результат в окне консоли:

```
scss\main.scss:4 Debug: 30px
scss\main.scss:4 Debug: 20px
scss\main.scss:4 Debug: 10px
```

### 5.5.5. Добавление элемента в конец списка

Функции `list.append(<Список>, <Значение>[, $separator: auto])` и `append(<Список>, <Значение>[, $separator: auto])` вставляют элемент в конец списка и возвращают новый список. Текущий список не изменяется:

```
C:\book\p1>sass --interactive
>> @use "sass:list"
>> $list1: 10px 20px
10px 20px
>> $list2: list.append($list1, 30px)
10px 20px 30px
>> $list1 // Список не изменился!
10px 20px
>> append($list1, 30px)
10px 20px 30px
```

Если вместо элемента указать список со значениями, то он станет последним элементом, и мы получим вложенный список:

```
>> append(10px 20px, (30px 40px))
10px 20px (30px 40px)
```

Если третий параметр не указан или имеет значение `auto`, то разделитель элементов будет тем же, что и у исходного списка:

```
>> append(10px 20px, 30px)
10px 20px 30px
>> append((10px, 20px), 30px, auto)
10px, 20px, 30px
>> list.append((10px, 20px), 30px, $separator: auto)
10px, 20px, 30px
```

Если в третьем параметре указано значение `space`, то элементы списка будут разделены пробелами:

```
>> list.append((10px, 20px), 30px, $separator: space)
10px 20px 30px
>> append((10px, 20px), 30px, space)
10px 20px 30px
```

Если в третьем параметре указано значение `comma`, то элементы списка будут разделены запятыми:

```
>> list.append(10px 20px, 30px, $separator: comma)
10px, 20px, 30px
>> append(10px 20px, 30px, comma)
10px, 20px, 30px
```

## 5.5.6. Объединение списков

Функции `list.join(<Список1>, <Список2>[, $separator: auto][, $bracketed: auto])` и `join(<Список1>, <Список2>[, $separator: auto][, $bracketed: auto])` объединяют два списка в один и возвращают новый список. Элементы из второго списка добавляются в конец первого списка. Текущий список не изменяется:

```
C:\book\p1>sass --interactive
>> @use "sass:list"
>> $list1: 10px 20px
10px 20px
>> $list2: list.join($list1, 30px 40px)
10px 20px 30px 40px
>> $list1 // Список не изменился!
10px 20px
>> join($list1, (30px, 40px))
10px 20px 30px 40px
```

Если параметр `$separator` не указан или имеет значение `auto`, то разделитель элементов будет тем же, что и у первого списка:

```
>> join(10px 20px, 30px 40px)
10px 20px 30px 40px
>> join((10px, 20px), 30px 40px, auto)
10px, 20px, 30px, 40px
>> list.join((10px, 20px), 30px 40px, $separator: auto)
10px, 20px, 30px, 40px
```

Если первый список не содержит разделителей, то разделитель элементов будет тем же, что и у второго списка:

```
>> list.join(10px, 20px 30px)
10px 20px 30px
>> join(10px, (20px, 30px))
10px, 20px, 30px
```

Если оба списка не содержат разделителей, то элементы списка будут разделены пробелом:

```
>> list.join(10px, 20px)
10px 20px
>> join(10px, 20px)
10px 20px
```

Если в параметре `$separator` указано значение `space`, то элементы списка будут разделены пробелами:

```
>> list.join((10px, 20px), (30px, 40px), $separator: space)
10px 20px 30px 40px
>> join((10px, 20px), (30px, 40px), space)
10px 20px 30px 40px
```

Если в параметре `$separator` указано значение `comma`, то элементы списка будут разделены запятыми:

```
>> list.join(10px 20px, 30px 40px, $separator: comma)
10px, 20px, 30px, 40px
>> join(10px 20px, 30px 40px, comma)
10px, 20px, 30px, 40px
```

Если параметр `$bracketed` не указан или имеет значение `auto`, то итоговый список будет внутри квадратных скобок, только если они есть у первого списка:

```
>> list.join((10px, 20px), [30px, 40px], $bracketed: auto)
10px, 20px, 30px, 40px
>> list.join([10px, 20px], [30px, 40px], $bracketed: auto)
[10px, 20px, 30px, 40px]
>> join([10px, 20px], [30px, 40px], space, auto)
[10px 20px 30px 40px]
```

Если в параметре `$bracketed` указано значение `true`, то итоговый список будет внутри квадратных скобок:

```
>> list.join((10px, 20px), [30px, 40px], $bracketed: true)
[10px, 20px, 30px, 40px]
>> join((10px, 20px), [30px, 40px], space, true)
[10px 20px 30px 40px]
```

Если в параметре `$bracketed` указано значение `false`, то итоговый список будет без скобок:

```
>> list.join([10px, 20px], [30px, 40px], $bracketed: false)
10px, 20px, 30px, 40px
>> join([10px, 20px], [30px, 40px], space, false)
10px 20px 30px 40px
```

Пример создания списка со значениями по умолчанию:

```
$list: () !default;
$list: join((10px 20px 30px), $list);
@each $value in $list {
 @debug $value;
}
```

Результат в окне консоли:

```
scss\main.scss:4 Debug: 10px
scss\main.scss:4 Debug: 20px
scss\main.scss:4 Debug: 30px
```

Функции `list.zip($lists...)` и `zip($lists...)` создают новый список на основе списков, переданных через запятую в качестве параметров. Каждый элемент итогового списка является вложенным списком, элементы которого расположены на равном удалении в переданных списках. Элементы в итоговом списке разделяются запятыми, а элементы вложенных списков — пробелами:

```
>> list.zip(10px 20px, 3px 4px)
10px 3px, 20px 4px
>> zip(10px 20px, 3px 4px, a b)
10px 3px a, 20px 4px b
```

Если списки содержат разное количество элементов, то размер итогового списка будет соответствовать размеру самого короткого списка:

```
>> list.zip(10px 20px 30px, 3px 4px)
10px 3px, 20px 4px
>> zip(10px 20px, 3px 4px 5px, a b c)
10px 3px a, 20px 4px b
```

## 5.5.7. Поиск значения в списке

Для поиска значения в списке предназначены следующие функции:

- ◆ `list.index(<Список>, <Значение>)` и `index(<Список>, <Значение>)` — возвращают индекс первого элемента, имеющего указанное значение. Обратите внимание: в Sass элементы списков нумеруются с 1. Если значение отсутствует в списке, то функции вернут значение `null`:

```
C:\book\pl>sass --interactive
>> @use "sass:list"
>> list.index(10px 20px 10px, 10px)
1
>> index(10px 20px 10px, 20px)
2
>> index(10px 20px 10px, 30px)
null
```

Проверить наличие значения в списке можно так (`null` трактуется как `false`):

```
$list: 10px 20px 10px;
@if index($list, 10px) {
 @debug "Значение найдено"; // Выведет: Значение найдено
}
```

```
@if index($list, 40px) {
 @debug "Значение найдено";
} @else {
 @debug "Значение НЕ найдено"; // Выведет: Значение НЕ найдено
}
```

- ◆ `math.max(<Значения>...)` — возвращает максимальное значение из списка с числовыми значениями. Для распаковки списка нужно указать три точки после переменной:

```
>> @use "sass:math"
>> $list: 10px 3px 20px
10px 3px 20px
>> math.max($list...)
20px
```

- ◆ `math.min(<Значения>...)` — возвращает минимальное значение из списка с числовыми значениями. Для распаковки списка нужно указать три точки после переменной:

```
>> $list: 10px 3px 20px
10px 3px 20px
>> math.min($list...)
3px
```

### 5.5.8. Сравнение списков

Для сравнения списков можно использовать операторы сравнения. Списки равны, если их элементы одинаковые:

```
C:\book\p1>sass --interactive
>> [1 2 3] == [1 2 3]
true
>> [1 2 3] == [1 2 4]
false
```

Списки с элементами, указанными через пробел, не равны спискам с элементами, указанными через запятую:

```
>> [1 2 3] == [1, 2, 3]
false
```

Списки в квадратных скобках не равны спискам в круглых скобках и спискам без скобок:

```
>> [1 2 3] == (1 2 3)
false
>> $list: 1 2 3
1 2 3
>> [1 2 3] == $list
false
```

## 5.6. Ассоциативные массивы

*Ассоциативный массив* — это список пар `ключ: значение`, разделенных запятыми, внутри круглых скобок. Чтобы получить значение, необходимо указать *ключ*, который использовался при сохранении значения.

Для работы с ассоциативными массивами в Sass предназначен модуль `map`. Подключить модуль позволяет следующая инструкция (в интерактивном режиме точку с запятой указывать не нужно):

```
@use "sass:map";
```

Для доступа к идентификаторам внутри модуля следует указать название модуля, точку, а затем идентификатор:

```
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@debug map.get($colors, "blue"); // #007bff
```

Некоторые идентификаторы доступны в глобальной области видимости. Перед такими идентификаторами в следующих разделах мы не будем указывать название модуля.

### 5.6.1. Создание ассоциативного массива

Для создания пустого ассоциативного массива нужно указать круглые скобки:

```
$colors: (); // Пустой ассоциативный массив
```

Внутри круглых скобок указываются пары `ключ: значение`, разделенные запятыми. Ключ должен быть уникальным:

```
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@debug type-of($colors); // map
```

Пример создания ассоциативного массива со значениями по умолчанию:

```
$colors: () !default;
$colors: map-merge(
 (
 "blue": #007bff,
 "red": #dc3545
), $colors
);
@each $key, $value in $colors {
 @debug "#{$key} => #{$value}";
}
```

Результат в окне консоли:

```
scss\main.scss:9 Debug: blue => #007bff
scss\main.scss:9 Debug: red => #dc3545
```

Наиболее часто ключи задаются в виде строк в кавычках, хотя можно использовать и другой тип данных — например, числа:

```
$map: (
 1: 10px,
 2: "str"
);
@debug map-get($map, 1); // 10px
```

или списки:

```
$map: (
 (1 2): 10px,
 (3 4): "str"
);
@debug map-get($map, 3 4); // str
```

## 5.6.2. Определение количества элементов

Рассматривайте ассоциативные массивы как списки с вложенными списками, содержащими два элемента: ключ и значение. Соответственно, можно передавать ассоциативные массивы функциям, предназначенным для работы со списками. Например, получим количество элементов ассоциативного массива с помощью функций `list.length()` и `length()`:

```
@use "sass:list";
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@debug list.length($colors); // 2
@debug length($colors); // 2
```

## 5.6.3. Получение значения по ключу

Функции `map.get(<Массив>, <Ключ>)` и `map-get(<Массив>, <Ключ>)` возвращают значение элемента по ключу, который использовался при сохранении значения. Если ключ не найден, то функции вернут значение `null`:

```
C:\book\p1>sass --interactive
>> @use "sass:map"
>> $colors: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> map.get($colors, "blue")
#007bff
>> map-get($colors, "red")
#dc3545
>> map-get($colors, "white")
null
```

## 5.6.4. Проверка существования ключа

Функции `map.has-key(<Массив>, <Ключ>)` и `map-has-key(<Массив>, <Ключ>)` возвращают значение `true`, если ключ существует, и `false` — в противном случае:

```
C:\book\p1>sass --interactive
>> @use "sass:map"
>> $colors: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> map.has-key($colors, "blue")
```

```

true
>> map.has-key($colors, "white")
false
>> map.has-key($colors, "red")
true
>> map.has-key($colors, "black")
false

```

### 5.6.5. Добавление элементов и изменение значения

Функции `map.merge(<Массив1>, <Массив2>)` и `map-merge(<Массив1>, <Массив2>)` позволяют объединить два ассоциативных массива в один. Возвращают новый ассоциативный массив, в котором элементы из второго массива добавлены в конец первого массива. Исходные массивы не изменяются:

```

C:\book\p1>sass --interactive
>> @use "sass:map"
>> $colors: ("blue": #007bff)
("blue": #007bff)
>> map.merge($colors, ("red": #dc3545))
("blue": #007bff, "red": #dc3545)
>> $colors // Массив не изменился!
("blue": #007bff)
>> map-merge($colors, ("red": #dc3545))
("blue": #007bff, "red": #dc3545)

```

Если ключ уже существует в первом массиве, то его значение будет изменено:

```

>> $colors: ("red": #dc3545)
("red": #dc3545)
>> map.merge($colors, ("red": #ff0000))
("red": #ff0000)
>> map-merge($colors, ("red": #ff0000))
("red": #ff0000)

```

Пример создания ассоциативного массива со значениями по умолчанию:

```

$colors: () !default;
$colors: map-merge(
 (
 "blue": #007bff,
 "red": #dc3545
), $colors
);
@each $key, $value in $colors {
 @debug "#{$key} => #{$value}";
}

```

Результат в окне консоли:

```

scss\main.scss:9 Debug: blue => #007bff
scss\main.scss:9 Debug: red => #dc3545

```

## 5.6.6. Удаление элементов

Функции `map.remove(<Массив>, $keys...)` и `map-remove(<Массив>, $keys...)` позволяют удалить элементы из ассоциативного массива по указанному ключу. Возвращают новый ассоциативный массив. Исходный массив не изменяется:

```
C:\book\p1>sass --interactive
>> @use "sass:map"
>> $colors: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> map.remove($colors, "red")
("blue": #007bff)
>> $colors // Массив не изменился!
("blue": #007bff, "red": #dc3545)
>> map-remove($colors, "blue")
("red": #dc3545)
```

Если ключ не существует в ассоциативном массиве, то он игнорируется:

```
>> $colors: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> map-remove($colors, "white")
("blue": #007bff, "red": #dc3545)
```

В параметре `$keys` можно указать сразу несколько ключей через запятую:

```
>> $colors: ("blue": #007bff, "red": #dc3545, "black": #000)
("blue": #007bff, "red": #dc3545, "black": #000)
>> map.remove($colors, "red", "black")
("blue": #007bff)
>> map-remove($colors, "red", "blue")
("black": #000)
```

## 5.6.7. Перебор элементов

Для перебора элементов ассоциативного массива удобно использовать цикл `@each` (см. *разд. 5.3.8*). На каждой итерации в первой переменной сохраняется ключ, а во второй переменной — значение, соответствующее ключу:

```
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@each $key, $value in $colors {
 @debug "#{$key} => #{$value}";
}
```

Результат в окне консоли:

```
scss\main.scss:6 Debug: blue => #007bff
scss\main.scss:6 Debug: red => #dc3545
```

Если указана только одна переменная, то в ней сохраняется список из ключа и значения:

```
$colors: (
 "blue": #007bff,
```

```

"red": #dc3545
);
@each $list in $colors {
 @debug $list;
}

```

Результат в окне консоли:

```

scss\main.scss:6 Debug: "blue" #007bff
scss\main.scss:6 Debug: "red" #dc3545

```

## 5.6.8. Преобразование ассоциативного массива в список

Преобразовать ассоциативный массив в список позволяют следующие функции:

- ◆ `map.keys(<Массив>)` и `map-keys(<Массив>)` — возвращают список со всеми ключами через запятую:

```

C:\book\pl>sass --interactive
>> @use "sass:map"
>> $colors: ("blue": #007bff, "red": #dc3545, "black": #000)
("blue": #007bff, "red": #dc3545, "black": #000)
>> map.keys($colors)
"blue", "red", "black"
>> map-keys($colors)
"blue", "red", "black"

```

- ◆ `map.values(<Массив>)` и `map-values(<Массив>)` — возвращают список со всеми значениями через запятую:

```

>> $colors: ("blue": #007bff, "red": #dc3545, "black": #000)
("blue": #007bff, "red": #dc3545, "black": #000)
>> map.values($colors)
#007bff, #dc3545, #000
>> map-values($colors)
#007bff, #dc3545, #000

```

Преобразуем ассоциативный массив в список:

```

>> $colors: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> $list: zip(map-keys($colors), map-values($colors))
"blue" #007bff, "red" #dc3545

```

## 5.6.9. Сравнение ассоциативных массивов

Для сравнения ассоциативных массивов можно использовать операторы сравнения. Ассоциативные массивы равны, если их ключи и значения равны:

```

C:\book\pl>sass --interactive
>> $map1: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> $map2: ("blue": #007bff, "red": #dc3545)
("blue": #007bff, "red": #dc3545)
>> $map1 == $map2
true

```

```
>> $map3: ("blue": #007bff, "red": #ff0000)
("blue": #007bff, "red": #ff0000)
>> $map1 == $map3
false
```

## 5.7. Строки

*Строка* — это последовательность символов внутри одинарных или двойных кавычек или без кавычек вообще. Для работы со строками в Sass предназначен модуль `string`. Подключить модуль позволяет следующая инструкция (в интерактивном режиме точку с запятой указывать не нужно):

```
@use "sass:string";
```

Для доступа к идентификаторам внутри модуля следует указать название модуля, точку, а затем идентификатор. Некоторые идентификаторы доступны в глобальной области видимости. Перед такими идентификаторами в следующих разделах мы не будем указывать название модуля.

### 5.7.1. Создание строки

Внутри строки в одинарных и двойных кавычках можно использовать любые символы, включая пробелы и запятые:

```
C:\book\p1>sass --interactive
>> "bold"
"bold"
>> 'bold'
"bold"
>> type-of("bold")
string
>> type-of('bold')
string
>> "word1, word2 word3"
"word1, word2 word3"
```

Символ двойной кавычки внутри строки в двойных кавычках и символ одинарной кавычки внутри строки в одинарных кавычках нужно экранировать с помощью слеша:

```
>> "\""
""
>> '\''
''
```

При использовании строки без кавычек следует помнить, что пробелы и запятые являются разделителями элементов списка. Кроме того, название цвета также задается без кавычек, поэтому можно получить результат, который совсем не ожидался.

Пример строки без кавычек:

```
>> bold
bold
```

```
>> type-of(bold)
string
>> type-of(red)
color
```

Если справа или слева от оператора + расположена строка, то будет выполнена операция конкатенации строк:

```
>> "str" + "ing"
"string"
>> bo + ld
bold
>> "str" + 1
"str1"
>> 1 + "px"
"1px"
```

Если к строке в кавычках прибавить строку без кавычек, то результатом будет строка в кавычках:

```
>> "str" + ing
"string"
```

Если к строке без кавычек прибавить строку в кавычках, то результатом будет строка без кавычек:

```
>> str + "ing"
string
```

Для конкатенации строк можно также использовать подстановку значений (см. *разд. 5.2.6*):

```
>> #{"str"}#{"ing"}
string
>> #{"str"}#{1}
str1
>> #{1}#"px"
1px
```

**Функции** `string.unquote(<Строка>)` и `unquote(<Строка>)` позволяют преобразовать строку в кавычках в строку без кавычек:

```
>> @use "sass:string"
>> string.unquote("bold")
bold
>> unquote('bold')
bold
```

**Функции** `string.quote(<Строка>)` и `quote(<Строка>)` возвращают строку в двойных кавычках:

```
>> string.quote(bold)
"bold"
>> quote(bold)
"bold"
>> quote('bold')
"bold"
```

При использовании со строкой оператора `-` возвращается строка со значениями, разделенными символом `-`:

```
>> font - size
font-size
>> "font" - "size"
"font"-"size"
```

Если значение указано только справа от оператора `-`, то возвращается строка, которая начинается с символа `-`, за которым следует значение:

```
>> - value
-value
>> - "value"
-"value"
```

При использовании со строкой оператора `/` возвращается строка со значениями, разделенными символом `/`:

```
>> #{10px + 2px} / 15px
12px/15px
```

Если значение указано только справа от оператора `/`, то возвращается строка, которая начинается с символа `/`, за которым следует значение:

```
>> / 5px
/5px
```

## 5.7.2. Кодировка файлов

При работе со строками, содержащими русские буквы, важно учитывать кодировку файлов. В Sass предполагается, что SCSS-файл сохранен в кодировке UTF-8. Советую именно так и поступать. Итоговый CSS-файл всегда сохраняется в кодировке UTF-8. Если в итоговом CSS-файле присутствуют не ASCII-символы, то в начало файла автоматически вставляется правило:

```
@charset "UTF-8";
```

## 5.7.3. Определение количества символов в строке

Получить количество символов в строке позволяют функции `string.length(<Строка>)`

и `str-length(<Строка>)`:

```
C:\book\p1>sass --interactive
>> @use "sass:string"
>> string.length("bold")
4
>> str-length('bold')
4
>> str-length(bold)
4
>> str-length("строка")
6
```

**НА ЗАМЕТКУ**

Не путайте функцию `str-length()` с функцией `length()`, предназначенной для работы со списками.

**5.7.4. Изменение регистра символов**

Для изменения регистра символов в строке предназначены следующие функции:

- ◆ `string.to-upper-case(<Строка>)` и `to-upper-case(<Строка>)` — возвращают строку, в которой все ASCII-символы указаны в верхнем регистре:

```
C:\book\pl>sass --interactive
>> @use "sass:string"
>> string.to-upper-case("bold")
"BOLD"
>> to-upper-case('bold')
"BOLD"
>> to-upper-case(bold)
BOLD
```

С русскими буквами функции не работают:

```
>> to-upper-case("строка")
"строка"
```

- ◆ `string.to-lower-case(<Строка>)` и `to-lower-case(<Строка>)` — возвращают строку, в которой все ASCII-символы указаны в нижнем регистре:

```
>> string.to-lower-case("BOLD")
"bold"
>> to-lower-case('BOLD')
"bold"
>> to-lower-case(BOLD)
bold
```

С русскими буквами функции не работают:

```
>> to-lower-case("СТРОКА")
"СТРОКА"
```

**5.7.5. Получение фрагмента строки**

Функции `string.slice(<Строка>, $start-at[, $end-at: -1])` и `str-slice(<Строка>, $start-at[, $end-at: -1])` возвращают фрагмент строки, начиная с индекса `$start-at` и заканчивая индексом `$end-at`. Символы, расположенные в строке по указанным индексам, попадут в результат:

```
C:\book\pl>sass --interactive
>> @use "sass:string"
>> string.slice("string", 4, 6)
"ing"
>> str-slice('string', 1, 3)
"str"
>> str-slice(string, 1, 6)
string
```

```
>> str-slice("строка", 1, 3)
"стр"
```

Чтобы получить один символ, следует в параметрах `$start-at` и `$end-at` указать одинаковый индекс. Выведем первый символ:

```
>> str-slice(string, 1, 1)
s
```

Если индекс `$end-at` не указан или имеет значение `-1`, то возвращается фрагмент, начиная с индекса `$start-at` до конца строки:

```
>> str-slice(string, 1)
string
>> str-slice(string, 1, -1)
string
>> str-slice(string, $start-at: 1, $end-at: 3)
str
```

### 5.7.6. Вставка фрагмента в строку

Функции `string.insert(<Строка1>, <Строка2>, $index)` и `str-insert(<Строка1>, <Строка2>, $index)` возвращают новую строку, в которой внутри строки `<Строка1>` вставлен фрагмент `<Строка2>` в позицию с индексом `$index`. Если индекс имеет отрицательное значение, то позиция отсчитывается с конца строки:

```
C:\book\p1>sass --interactive
>> @use "sass:string"
>> string.insert("string", "+++", 3)
"st+++ring"
>> str-insert("string", "+++", 1)
"+++string"
>> str-insert("string", "+++", -1)
"string+++"
```

Если значение `$index` больше длины строки, то фрагмент добавляется в конец строки. Если значение `$index` отрицательное и оно выходит за пределы строки, то фрагмент добавляется в начало:

```
>> str-insert('string', "+++", 20)
"string+++
>> str-insert('string', "+++", -20)
"+++string"
```

### 5.7.7. Поиск в строке

Функции `string.index(<Строка1>, <Строка2>)` и `str-index(<Строка1>, <Строка2>)` возвращают индекс первого вхождения фрагмента `<Строка2>` в строку `<Строка1>`. Если фрагмент не найден, то функции вернут значение `null`:

```
C:\book\p1>sass --interactive
>> @use "sass:string"
>> string.index("string", "ing")
```

```
>> str-index("string", "str")
1
>> str-index("string", "10")
null
```

### 5.7.8. Сравнение строк

Для сравнения строк можно использовать операторы сравнения. Строки в кавычках и без кавычек с одинаковым содержимым считаются равными. Регистр букв имеет значение:

```
C:\book\p1>sass --interactive
>> "abc" == abc
true
>> "abc" == abcd
false
>> abc == ABC
false
```

### 5.7.9. Создание уникального идентификатора

Функции `string.unique-id()` и `unique-id()` возвращают строку без кавычек, внутри которой символы сгенерированы случайным образом. Эта строка является допустимым идентификатором в CSS и уникальна в текущей сессии Sass:

```
C:\book\p1>sass --interactive
>> @use "sass:string"
>> string.unique-id()
u5g60e7
>> string.unique-id()
u5g60f1
>> unique-id()
u5g60fa
>> unique-id()
u5g60fb
```

## 5.8. Работа с цветом

Для работы с цветом в Sass предназначен модуль `color`. Подключить модуль позволяет следующая инструкция (в интерактивном режиме точку с запятой указывать не нужно):

```
@use "sass:color";
```

Для доступа к идентификаторам внутри модуля следует указать название модуля, точку, а затем идентификатор. Некоторые идентификаторы доступны в глобальной области видимости. Перед такими идентификаторами в следующих разделах мы не будем указывать название модуля.

## 5.8.1. Способы указания значения

Цвет можно задать одним из следующих способов:

- ◆ именем цвета — blue, green и т. д.:

```
C:\book\pl>sass --interactive
>> red
red
```

Обратите внимание: именованные цвета неотличимы от строк без кавычек, что может привести к неожиданным последствиям. Поэтому лучше использовать другие способы задания значения для цвета, а строки указывать внутри кавычек, чтобы случайно не получить цвет:

```
>> type-of(red)
color
>> type-of("red")
string
```

- ◆ значением вида #RGB, где R — насыщенность красного, G — насыщенность зеленого и B — насыщенность синего в цвете. Значения задаются одинарными шестнадцатеричными числами от 0 до F:

```
>> #f00
#f00
```

- ◆ значением вида #RRGGBB, где RR — насыщенность красного, GG — насыщенность зеленого и BB — насыщенность синего в цвете. В таком формате значения задаются двузначными шестнадцатеричными числами от 00 до FF:

```
>> #ff0000
#ff0000
```

- ◆ значением вида #RRGGBBAA, где RR — насыщенность красного, GG — насыщенность зеленого, BB — насыщенность синего и AA — уровень прозрачности цвета (альфа-канал). Значения задаются двузначными шестнадцатеричными числами от 00 до FF:

```
>> #ff000099
#ff000099
```

- ◆ значением вида rgb(R, G, B), где R, G и B — насыщенности красного, зеленого и синего цветов, которые задаются числами от 0 до 255 или в процентах от 0% до 100%:

```
>> rgb(255, 0, 0)
red
>> rgb(100%, 0%, 0%)
red
```

В Sass существуют дополнительные форматы функции rgb():

```
rgb(R G B)
rgb(R G B / A)
rgb(R, G, B[, A])
rgb(<Цвет>, A)
```

где R, G и B — насыщенности красного, зеленого и синего цветов, которые задаются числами от 0 до 255 или в процентах от 0% до 100%, A — уровень прозрачно-

сти цвета (альфа-канал), представляющий собой значение от 0.0 или 0% (цвет полностью прозрачен) до 1.0 или 100% (цвет полностью непрозрачен):

```
>> rgb(255 0 0)
red
>> rgb(100% 0% 0%)
red
>> rgb(255 0 0 / 0.5)
rgba(255, 0, 0, 0.5)
>> rgb(255 0 0 / 50%)
rgba(255, 0, 0, 0.5)
>> rgb(100% 0% 0% / 0.5)
rgba(255, 0, 0, 0.5)
>> rgb(255, 0, 0)
red
>> rgb(255, 0, 0, 0.3)
rgba(255, 0, 0, 0.3)
>> rgb(255, 0, 0, 30%)
rgba(255, 0, 0, 0.3)
>> rgb(#ff0000, 30%)
rgba(255, 0, 0, 0.3)
```

- ◆ значением вида `rgba(R, G, B, A)`, где `R`, `G` и `B` — насыщенности красного, зеленого и синего цветов, которые задаются числами от 0 до 255 или в процентах от 0% до 100%, `A` — задает уровень прозрачности цвета (альфа-канал), представляющий собой значение от 0.0 (цвет полностью прозрачен) до 1.0 (цвет полностью непрозрачен). Пример указания полупрозрачного красного цвета:

```
>> rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> rgba(100%, 0%, 0%, 0.5)
rgba(255, 0, 0, 0.5)
```

В Sass существуют дополнительные форматы функции `rgba()`:

```
rgba(R G B)
rgba(R G B / A)
rgba(R, G, B[, A])
rgba(<Цвет>, A)
```

где `R`, `G` и `B` — насыщенности красного, зеленого и синего цветов, которые задаются числами от 0 до 255 или в процентах от 0% до 100%, `A` — уровень прозрачности цвета (альфа-канал), представляющий собой значение от 0.0 или 0% (цвет полностью прозрачен) до 1.0 или 100% (цвет полностью непрозрачен):

```
>> rgba(255 0 0)
red
>> rgba(100% 0% 0%)
red
>> rgba(255 0 0 / 0.5)
rgba(255, 0, 0, 0.5)
>> rgba(255 0 0 / 50%)
rgba(255, 0, 0, 0.5)
```

```
>> rgba(100% 0% 0% / 0.5)
rgba(255, 0, 0, 0.5)
>> rgba(255, 0, 0)
red
>> rgba(255, 0, 0, 0.3)
rgba(255, 0, 0, 0.3)
>> rgba(255, 0, 0, 30%)
rgba(255, 0, 0, 0.3)
>> rgba(#ff0000, 30%)
rgba(255, 0, 0, 0.3)
```

- ◆ значением вида `hsl(H, S, L)`, где `H` — оттенок (число от 0 до 360), `S` — насыщенность (проценты от 0 до 100) и `L` — светлота (проценты от 0 до 100):

```
>> hsl(0, 100%, 50%)
red
```

В Sass существуют дополнительные форматы функции `hsl()`:

```
hsl(H S L)
hsl(H S L / A)
hsl(H, S, L[, A])
```

где `A` — уровень прозрачности цвета (альфа-канал), представляющий собой значение от 0.0 или 0% (цвет полностью прозрачен) до 1.0 или 100% (цвет полностью непрозрачен):

```
>> hsl(0 100% 50%)
red
>> hsl(25deg 100% 50%)
#ff6a00
>> hsl(0 100% 50% / 0.5)
rgba(255, 0, 0, 0.5)
>> hsl(0 100% 50% / 50%)
rgba(255, 0, 0, 0.5)
>> hsl(0, 100%, 50%, 0.3)
rgba(255, 0, 0, 0.3)
>> hsl(0, 100%, 50%, 30%)
rgba(255, 0, 0, 0.3)
```

- ◆ значением вида `hsla(H, S, L, A)`, где `H` — оттенок (число от 0 до 360), `S` — насыщенность (проценты от 0 до 100), `L` — светлота (проценты от 0 до 100) и `A` — уровень прозрачности цвета (альфа-канал), представляющий собой значение от 0.0 (цвет полностью прозрачен) до 1.0 (цвет полностью непрозрачен):

```
>> hsla(0, 100%, 50%, 0.5)
rgba(255, 0, 0, 0.5)
```

В Sass существуют дополнительные форматы функции `hsla()`:

```
hsla(H S L)
hsla(H S L / A)
hsla(H, S, L[, A])
```

где  $\alpha$  — уровень прозрачности цвета (альфа-канал), представляющий собой значение от 0.0 или 0% (цвет полностью прозрачен) до 1.0 или 100% (цвет полностью непрозрачен):

```
>> hsla(0 100% 50%)
red
>> hsla(25deg 100% 50%)
#ff6a00
>> hsla(0 100% 50% / 0.5)
rgba(255, 0, 0, 0.5)
>> hsla(0 100% 50% / 50%)
rgba(255, 0, 0, 0.5)
>> hsla(0, 100%, 50%, 0.3)
rgba(255, 0, 0, 0.3)
>> hsla(0, 100%, 50%, 30%)
rgba(255, 0, 0, 0.3)
>> hsla(0, 100%, 50%)
red
```

При выводе цвета в CSS-файл выбирается наиболее короткая запись:

```
>> rgb(255, 0, 0)
red
```

## 5.8.2. Получение значений компонентов цвета

Получить значения компонентов цвета позволяют следующие функции:

- ◆ `color.red(<Цвет>)` и `red(<Цвет>)` — возвращают значение красного канала в виде числа от 0 до 255:

```
C:\book\pl>sass --interactive
>> @use "sass:color"
>> $color: #01020304
#01020304
>> color.red($color)
1
>> red(#dc3545)
220
>> red(black)
0
>> red(white)
255
```

- ◆ `color.green(<Цвет>)` и `green(<Цвет>)` — возвращают значение зеленого канала в виде числа от 0 до 255:

```
>> $color: #01020304
#01020304
>> color.green($color)
2
>> green(#dc3545)
53
```

- ◆ `color.blue(<Цвет>)` и `blue(<Цвет>)` — возвращают значение синего канала в виде числа от 0 до 255:

```
>> $color: #01020304
#01020304
>> color.blue($color)
3
>> blue(#dc3545)
69
```

- ◆ `color.alpha(<Цвет>)`, `alpha(<Цвет>)`, `color.opacity(<Цвет>)` и `opacity(<Цвет>)` — возвращают значение альфа-канала (уровень прозрачности цвета) в виде вещественного числа от 0 до 1:

```
>> $color: #01020304
#01020304
>> color.alpha($color)
0.0156862745
>> alpha(#dc3545)
1
>> alpha(#ff000099)
0.6
>> color.opacity(#ff000099)
0.6
>> opacity(#ff000099)
0.6
```

Функция `alpha(opacity=<Значение>)` поддерживает также синтаксис указания прозрачности для веб-браузера Internet Explorer — она возвращает строку без кавычек:

```
>> alpha(opacity=50)
alpha(opacity=50)
```

- ◆ `color.hue(<Цвет>)` и `hue(<Цвет>)` — возвращают значение канала hue (оттенок) из HSL в виде числа от 0deg до 360deg:

```
>> $color: hsl(25, 100%, 50%)
#ff6a00
>> color.hue($color)
25deg
>> hue($color)
25deg
```

- ◆ `color.saturation(<Цвет>)` и `saturation(<Цвет>)` — возвращают значение канала saturation (насыщенность) из HSL в виде числа от 0% до 100%:

```
>> $color: hsl(25, 100%, 50%)
#ff6a00
>> color.saturation($color)
100%
>> saturation($color)
100%
```

- ◆ `color.lightness(<Цвет>)` и `lightness(<Цвет>)` — возвращают значение канала `lightness` (светлота) из HSL в виде числа от 0% до 100%:

```
>> $color: hsl(25, 100%, 50%)
#ff6a00
>> color.lightness($color)
50%
>> lightness($color)
50%
```

### 5.8.3. Изменение значений компонентов цвета

Изменить значения компонентов цвета позволяют следующие функции:

- ◆ `color.change()` и `change-color()` — возвращают цвет с измененными значениями. **Форматы функций:**

```
color.change(<Цвет>, $red: null, $green: null, $blue: null,
 $hue: null, $saturation: null, $lightness: null,
 $alpha: null)
change-color(<Цвет>, $red: null, $green: null, $blue: null,
 $hue: null, $saturation: null, $lightness: null,
 $alpha: null)
```

**Диапазоны значений:**

- `$red`, `$green` и `$blue` — от 0 до 255;
- `$hue` — от 0 до 359deg (единицу измерения можно не указывать);
- `$saturation` и `$lightness` — от 0 до 100% (единицу измерения можно не указывать);
- `$alpha` — вещественное число от 0 до 1.

Можно указать одну переменную или сразу несколько. При этом нельзя указывать одновременно компоненты из RGB и HSL, иначе будет выведено сообщение об ошибке. Пример изменения значений:

```
C:\book\pl>sass --interactive
>> @use "sass:color"
>> $color: #000000
#000000
>> color.change($color, $red: 1)
#010000
>> color.change($color, $red: 1, $green: 2, $blue: 3)
#010203
>> color.change($color, $red: 1, $alpha: 0.5)
rgba(1, 0, 0, 0.5)
>> color.change($color, $hue:25deg, $saturation:100%, $lightness:50%)
#ff6a00
>> change-color($color, $red: 1, $green: 2, $blue: 3)
#010203
```

```
>> change-color($color, $hue:25deg, $saturation:100%, $lightness:50%)
#ff6a00
>> change-color($color, $red: 1, $alpha: 0.5)
rgba(1, 0, 0, 0.5)
```

- ◆ `color.adjust()` и `adjust-color()` — прибавляют к компонентам цвета указанные значения и возвращают цвет с измененными значениями. Форматы функций:

```
color.adjust(<Цвет>, $red: null, $green: null, $blue: null,
 $hue: null, $saturation: null, $lightness: null,
 $alpha: null)
adjust-color(<Цвет>, $red: null, $green: null, $blue: null,
 $hue: null, $saturation: null, $lightness: null,
 $alpha: null)
```

#### Диапазоны значений:

- `$red`, `$green` и `$blue` — от -255 до 255;
- `$hue` — от -360deg до 360deg (единицу измерения можно не указывать);
- `$saturation` и `$lightness` — от -100% до 100% (единицу измерения можно не указывать);
- `$alpha` — вещественное число от -1 до 1.

Можно указать одну переменную или сразу несколько. При этом нельзя указывать одновременно компоненты из RGB и HSL, иначе будет выведено сообщение об ошибке. Пример изменения значений:

```
>> $color: rgba(10, 20, 30, 0.5)
rgba(10, 20, 30, 0.5)
>> color.adjust($color, $red: 1, $green: 2, $blue: 3)
rgba(11, 22, 33, 0.5)
>> color.adjust($color, $red: -1, $alpha: 0.2)
rgba(9, 20, 30, 0.7)
>>
>> color.adjust($color, $hue:25deg, $saturation:10%, $lightness:-5%)
rgba(3, 4, 12, 0.5)
>> adjust-color($color, $red: 1, $green: 2, $blue: 3)
rgba(11, 22, 33, 0.5)
>> adjust-color($color, $red: -1, $alpha: 0.2)
rgba(9, 20, 30, 0.7)
>> adjust-color($color, $hue: 25, $saturation: 10, $lightness: -5)
rgba(3, 4, 12, 0.5)
```

- ◆ `color.scale()` и `scale-color()` — изменяют значения компонентов цвета и возвращают новый цвет. Форматы функций:

```
color.scale(<Цвет>, $red: null, $green: null, $blue: null,
 $saturation: null, $lightness: null,
 $alpha: null)
scale-color(<Цвет>, $red: null, $green: null, $blue: null,
 $saturation: null, $lightness: null,
 $alpha: null)
```

Диапазоны значений всех переменных от -100% до 100%. Можно указать одну переменную или сразу несколько. При этом нельзя указывать одновременно компоненты из RGB и HSL, иначе будет выведено сообщение об ошибке. Пример изменения значений:

```
>> $color: rgba(10, 20, 30, 0.5)
rgba(10, 20, 30, 0.5)
>> color.scale($color, $red: 10%, $green: -20%, $blue: 50%)
rgba(35, 16, 143, 0.5)
>> color.scale($color, $red: -10%, $alpha: 20%)
rgba(9, 20, 30, 0.6)
>> scale-color($color, $red: 10%, $green: -20%, $blue: 50%)
rgba(35, 16, 143, 0.5)
>> scale-color($color, $red: -10%, $alpha: 20%)
rgba(9, 20, 30, 0.6)
>> scale-color($color, $saturation: -20%, $lightness: 10%)
rgba(26, 44, 61, 0.5)
```

- ◆ `adjust-hue(<Цвет>, $degrees)` — изменяет значение канала `hue` (оттенок) на величину `$degrees` и возвращает измененный цвет. Значение в параметре `$degrees` указывается в диапазоне от `-360deg` до `360deg` (единицу измерения можно не указывать):

```
>> $color: hsl(25, 100%, 50%)
#ff6a00
>> adjust-hue($color, 25deg)
#ffd500
>> hue(#ffd500)
50.1176470588deg
>> adjust-hue($color, -25deg)
red
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: hsl(25, 100%, 50%)
#ff6a00
>> adjust-color($color, $hue: 25deg)
#ffd500
```

- ◆ `color.complement(<Цвет>)` и `complement(<Цвет>)` — изменяют значение канала `hue` (оттенок) на величину `180deg` и возвращают измененный цвет:

```
>> $color: hsl(0, 100%, 50%)
red
>> color.complement($color)
aqua
>> complement($color)
aqua
>> hue(aqua)
180deg
>> adjust-color($color, $hue: 180deg)
aqua
```

## 5.8.4. Изменение насыщенности цвета

Указать фиксированное значение для канала `saturation` (насыщенность) позволяют функции `color.change()` и `change-color()` (см. *разд. 5.8.3*):

```
C:\book\pl>sass --interactive
>> @use "sass:color"
>> $color: hsl(25, 50%, 50%)
#bf7540
>> change-color($color, $saturation: 25%)
#9f7a60
>> saturation(#9f7a60)
24.7058823529%
```

Задать относительное значение можно с помощью следующих функций:

- ◆ `saturate(<Цвет>, $amount)` — увеличивает значение канала `saturation` (насыщенность) на величину `$amount` и возвращает измененный цвет. Значение в параметре `$amount` указывается в диапазоне от 0 до 100% (единицу измерения можно не указывать):

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> saturate($color, 25%)
#df7020
>> saturation(#df7020)
74.9019607843%
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> adjust-color($color, $saturation: 25)
#df7020
```

Для изменения насыщенности вместо этих функций лучше воспользоваться функциями `color.scale()` и `scale-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> scale-color($color, $saturation: 25%)
#cf7230
>> saturation(#cf7230)
62.3529411765%
```

- ◆ `desaturate(<Цвет>, $amount)` — уменьшает значение канала `saturation` (насыщенность) на величину `$amount` и возвращает измененный цвет. Значение в параметре `$amount` указывается в диапазоне от 0 до 100% (единицу измерения можно не указывать):

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> desaturate($color, 25%)
#9f7a60
>> saturation(#9f7a60)
24.7058823529%
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> adjust-color($color, $saturation: -25)
#9f7a60
```

Для изменения насыщенности вместо этих функций лучше воспользоваться функциями `color.scale()` и `scale-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> scale-color($color, $saturation: -25%)
#af7850
>> saturation(#af7850)
37.2549019608%
```

### 5.8.5. Изменение яркости цвета

Указать фиксированное значение для канала `lightness` (светлота, яркость) позволяют функции `color.change()` и `change-color()` (см. разд. 5.8.3):

```
C:\book\p1>sass --interactive
>> @use "sass:color"
>> $color: hsl(25, 50%, 50%)
#bf7540
>> change-color($color, $lightness: 75%)
#dfba9f
>> lightness(#dfba9f)
74.9019607843%
```

Задать относительное значение можно с помощью следующих функций:

- ◆ `lighten(<Цвет>, $amount)` — делает цвет светлее, увеличивая значение канала `lightness` (светлота) на величину `$amount`, и возвращает измененный цвет. Значение в параметре `$amount` указывается в диапазоне от 0 до 100% (единицу измерения можно не указывать):

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> lighten($color, 25%)
#dfba9f
>> lightness(#dfba9f)
74.9019607843%
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> adjust-color($color, $lightness: 25%)
#dfba9f
```

Для изменения яркости вместо этих функций лучше воспользоваться функциями `color.scale()` и `scale-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> scale-color($color, $lightness: 25%)
#cf9770
>> lightness(#cf9770)
62.5490196078%
```

- ◆ `darken(<Цвет>, $amount)` — делает цвет темнее, уменьшая значение канала `lightness` (светлота) на величину `$amount`, и возвращает измененный цвет. Значение в параметре `$amount` указывается в диапазоне от 0 до 100% (единицу измерения можно не указывать):

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> darken($color, 25%)
#603a20
>> lightness(#603a20)
25.0980392157%
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> adjust-color($color, $lightness: -25%)
#603a20
```

Для изменения яркости вместо этих функций лучше воспользоваться функциями `color.scale()` и `scale-color()`:

```
>> $color: hsl(25, 50%, 50%)
#bf7540
>> scale-color($color, $lightness: -25%)
#8f5830
>> lightness(#8f5830)
37.4509803922%
```

## 5.8.6. Изменение прозрачности цвета

Указать фиксированное значение для альфа-канала (прозрачность) позволяют функции `color.change()` и `change-color()` (см. *разд. 5.8.3*):

```
C:\book\p1>sass --interactive
>> @use "sass:color"
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> change-color($color, $alpha: 0.7)
rgba(255, 0, 0, 0.7)
```

Задать относительное значение можно с помощью следующих функций:

- ◆ `opacity(<Цвет>, $amount)` и `fade-in(<Цвет>, $amount)` — увеличивают значение альфа-канала (прозрачность) на величину `$amount` и возвращают измененный цвет. Значение в параметре `$amount` указывается в диапазоне от 0.0 до 1.0:

```
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> opacity($color, 0.2)
rgba(255, 0, 0, 0.7)
>> fade-in($color, 0.2)
rgba(255, 0, 0, 0.7)
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> adjust-color($color, $alpha: 0.2)
rgba(255, 0, 0, 0.7)
```

Для изменения прозрачности вместо этих функций лучше воспользоваться функциями `color.scale()` и `scale-color()`:

```
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> scale-color($color, $alpha: 25%)
rgba(255, 0, 0, 0.625)
```

- ◆ `transparentize(<Цвет>, $amount)` и `fade-out(<Цвет>, $amount)` — уменьшают значение альфа-канала (прозрачность) на величину `$amount` и возвращают измененный цвет. Значение в параметре `$amount` указывается в диапазоне от 0.0 до 1.0:

```
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> transparentize($color, 0.2)
rgba(255, 0, 0, 0.3)
>> fade-out($color, 0.2)
rgba(255, 0, 0, 0.3)
```

Изменить значение можно также с помощью функций `color.adjust()` и `adjust-color()`:

```
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> adjust-color($color, $alpha: -0.2)
rgba(255, 0, 0, 0.3)
```

Для изменения прозрачности вместо этих функций лучше воспользоваться функциями `color.scale()` и `scale-color()`:

```
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> scale-color($color, $alpha: -25%)
rgba(255, 0, 0, 0.375)
```

### 5.8.7. Преобразование цвета в оттенки серого

Функции `color.grayscale(<Цвет>)` и `grayscale(<Цвет>)` преобразуют цвет в оттенки серого, устанавливая для канала `saturation` (насыщенность) значение 0:

```
C:\book\p1>sass --interactive
>> @use "sass:color"
>> $color: #dc3545
#dc3545
>> saturation($color)
70.4641350211%
>> color.grayscale($color)
#898989
>> grayscale($color)
#898989
>> saturation(#898989)
0%
```

Аналогичного эффекта можно достичь с помощью функций `color.change()` и `change-color()` (см. разд. 5.8.3):

```
>> $color: #dc3545
#dc3545
>> change-color($color, $saturation: 0)
#898989
```

### 5.8.8. Смешивание цветов

Функции `color.mix($color1, $color2[, $weight: 50%])` и `mix($color1, $color2[, $weight: 50%])` позволяют смешать два цвета. Параметр `$weight` задает степень вклада того или иного цвета в результат в виде значения от 0% до 100%. По умолчанию параметр имеет значение 50%. Если в параметре `$weight` указано значение, больше 50%, то вклад цвета `$color1` в результат будет больше. Если в параметре `$weight` указано значение, меньше 50%, то вклад цвета `$color2` в результат будет больше. Значение 0% даст в результате `$color2`, а значение 100% — `$color1`. При смешивании цветов учитывается также уровень прозрачности. Пример смешивания двух цветов:

```
C:\book\p1>sass --interactive
>> @use "sass:color"
>> $color1: #010203
#010203
>> $color2: #030201
#030201
>> color.mix($color1, $color2)
#020202
>> color.mix($color1, $color2, 75%)
#020203
>> color.mix($color1, $color2, 100%)
#010203
>> mix($color1, $color2, 50%)
#020202
```

```
>> mix($color1, $color2, 25%)
#030202
>> mix($color1, $color2, 0)
#030201
```

### 5.8.9. Инвертирование цвета

Функции `color.invert($color[, $weight: 100%])` и `invert($color[, $weight: 100%])` позволяют инвертировать цвет. В параметре `$weight` указывается значение от 0% до 100%. По умолчанию параметр имеет значение 100%. Пример:

```
C:\book\p1>sass --interactive
>> @use "sass:color"
>> color.invert(white)
black
>> invert(black)
white
>> invert(black, 50%)
gray
>> invert(black, 75%)
#bfbfbf
>> invert(black, 25%)
#404040
>> invert(black, 0)
black
```

### 5.8.10. Получение значения в формате #AARRGGBB

Функции `color.ie-hex-str($color)` и `ie-hex-str($color)` возвращают значение цвета в формате #AARRGGBB в виде строки без кавычек:

```
C:\book\p1>sass --interactive
>> @use "sass:color"
>> $color: rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 0.5)
>> color.ie-hex-str($color)
#80FF0000
>> ie-hex-str($color)
#80FF0000
>> ie-hex-str(red)
#FFFF0000
```

## 5.9. Пользовательские функции

*Функция* — это фрагмент кода, который можно неоднократно вызвать из любого места программы. В предыдущих разделах мы уже не один раз использовали встроенные функции. Например, с помощью функции `darken()` делали цвет темнее. В этом разделе мы рассмотрим создание пользовательских функций, которые позволят уменьшить избыточность кода и повысить его структурированность.

### 5.9.1. Создание функции и ее вызов

Определение пользовательской функции имеет следующий формат:

```
@function <Название функции>([<Название параметра 1>
 [, ..., <Название параметра N>]])
{
 <Тело функции>
 @return <Возвращаемое значение>;
}
```

Название функции должно быть допустимым идентификатором, к которому предъявляются такие же требования, как и к названиям переменных, но без символа \$. После названия функции внутри круглых скобок через запятую указываются названия параметров, которые являются локальными переменными. Если функция не содержит параметров, то указываются только круглые скобки.

После описания параметров внутри фигурных скобок размещаются инструкции, которые будут выполняться при каждом вызове функции. Фигурные скобки указываются в любом случае, даже если тело функции состоит только из одной инструкции. Точка с запятой после закрывающей фигурной скобки не ставится.

Функция в Sass всегда что-то возвращает. Возвращаемое значение указывается после оператора @return. Создадим функцию, которая не содержит параметров и возвращает цвет:

```
@function div-color() {
 @return red;
}
```

Чтобы вызвать функцию, нужно указать название функции, после которого внутри круглых скобок через запятую передать значения. Если функция не содержит параметров, то указываются только круглые скобки. Вызов функции допускается вставлять везде, где ожидается какое-либо значение или где мы можем использовать переменную, — например, для указания значения атрибута стиля:

```
div {
 color: div-color(); // color: red;
}
```

Символы дефиса и подчеркивания в названии функции являются взаимозаменяемыми. Поэтому предыдущий код можно записать так:

```
div {
 color: div_color(); // color: red;
}
```

Возвращаемое функцией значение можно также сохранить в переменной или подставить в выражение:

```
// Определение функции
@function max-width() {
 @return 1140px; // Возвращаемое значение
}
// Вызов функции
$value: max-width(); // Сохранение значения в переменной
```

```
@debug $value; // 1140px
div {
 // Подстановка вызова функции в выражение
 width: 500px / max-width() * 100%; // width: 43.8596491228%;
}
```

Как видно из примера, вызывать функцию можно сколько угодно раз, что позволяет использовать один и тот же код многократно.

Функция может содержать один или несколько параметров через запятую. Каждый параметр является локальной переменной. Эта переменная создается при вызове функции, а после выхода из функции она удаляется. Таким образом, локальная переменная видна только внутри функции. Если название локальной переменной совпадает с названием глобальной переменной, то все операции будут производиться с локальной переменной, а значение глобальной не изменится:

```
$x: 20; // Глобальная переменная
@function sum($x, $y) {
 @debug $x; // 5
 $z: $x + $y; // Обращение к локальной переменной $x
 $x: 50; // Значение глобальной переменной не изменится
 @return $z;
}
// Вызов функции
@debug sum(5, 3); // 8
@debug $x; // 20
```

Переданные при вызове функции значения присваиваются переменным, расположенным в той же позиции в определении функции. Так, при использовании функции `sum()` переменной `$x` будет присвоено значение 5, а переменной `$y` — значение 3. Результат выполнения функции выводится в окно консоли с помощью директивы `@debug`.

Вернуть значение из функции позволяет оператор `@return`. После исполнения этого оператора выполнение функции останавливается и управление передается обратно в точку вызова функции. Это означает, что инструкции после оператора `@return` никогда не будут выполнены. При использовании оператора `@return` не должно быть неоднозначных ситуаций. Например, в этом случае возвращаемое значение зависит от условия:

```
@function sum($x, $y) {
 @if $x > 0 {
 @return $x + $y;
 }
}
```

Если переменная `$x` имеет значение больше нуля, то все будет нормально:

```
@debug sum(5, 3); // 8
```

Но если переменная `$x` равна нулю или имеет отрицательное значение, то будет выведено сообщение об ошибке:

```
@debug sum(-5, 3);
// Error: Function finished without @return.
```

Чтобы избежать подобной ситуации, следует в конце тела функции вставить оператор `@return` со значением по умолчанию:

```
@function sum($x, $y) {
 @if $x > 0 {
 @return $x + $y;
 }
 @return 0;
}
```

## 5.9.2. Расположение определений функций

Все инструкции в программе выполняются последовательно сверху вниз. Это означает, что прежде чем использовать функцию в программе, ее необходимо предварительно определить. Поэтому определение функции должно быть расположено перед вызовом функции. Обратите внимание на то, что размещать определение одной функции внутри другой нельзя. Если вызов функции расположен перед ее определением, то мы получим название функции, вместо ее вызова:

```
@debug sum(5, 3); // sum(5, 3)
```

```
@function sum($x, $y) {
 @return $x + $y;
}
```

```
@debug sum(5, 3); // 8
```

Определение функции можно разместить внутри блока, ограничив область видимости функции фигурными скобками:

```
div {
 @function sum($x, $y) {
 @return $x + $y;
 }
 width: sum(500px, 100px); // width: 600px;
}
@debug sum(5, 3); // sum(5, 3)
```

## 5.9.3. Способы передачи значений в функцию

Как вы уже знаете, после названия функции внутри круглых скобок указываются названия параметров через запятую. Если функция не содержит параметров, то указываются только круглые скобки. Название параметра является локальной переменной. Эта переменная создается при вызове функции, а после выхода из функции она удаляется. Таким образом, локальная переменная видна только внутри функции, и ее значение между вызовами не сохраняется. Если название локальной переменной совпадает с названием глобальной переменной, то все операции будут производиться с локальной переменной, а значение глобальной не изменится.

При вызове функции указывается название функции, после которого внутри круглых скобок передаются значения. Если функция не содержит параметров, то при

вызове указываются только круглые скобки. Количество параметров в определении функции должно совпадать с количеством значений при вызове. Переданные значения присваиваются переменным, расположенным в той же позиции в определении функции:

```
@function sum($x, $y) {
 @return $x + $y;
}
@debug sum(10, 20); // 30
```

При вызове функции `sum()` переменной `$x` будет присвоено значение 10, а переменной `$y` — значение 20.

Перед значением можно добавить название параметра, указанное в определении функции, и двоеточие. В этом случае именованные параметры должны следовать после обычных значений, которые присваиваются переменным, расположенным в той же позиции в определении функции:

```
@function sum($x, $y, $z) {
 @return $x + $y + $z;
}
@debug sum(10, 20, 30); // 60
@debug sum(10, 20, $z: 30); // 60
@debug sum(10, $y: 20, $z: 30); // 60
@debug sum($x: 10, $y: 20, $z: 30); // 60
// Так нельзя
// @debug sum(10, $z: 30, 20); // Error: expected "...".
```

При использовании именованных параметров их можно указывать в произвольном порядке:

```
@function sum($x, $y, $z) {
 @debug "$x = #{ $x }"; // $x = 10
 @debug "$y = #{ $y }"; // $y = 20
 @debug "$z = #{ $z }"; // $z = 30
 @return $x + $y + $z;
}
@debug sum(10, $z: 30, $y: 20); // 60
@debug sum($z: 30, $y: 20, $x: 10); // 60
```

Чтобы передать значения, расположенные внутри списка, следует после списка или переменной, содержащей список, указать три точки. Количество элементов списка должно совпадать с количеством параметров функции:

```
@function sum($x, $y, $z) {
 @debug "$x = #{ $x }"; // $x = 10
 @debug "$y = #{ $y }"; // $y = 20
 @debug "$z = #{ $z }"; // $z = 30
 @return $x + $y + $z;
}
$list: 10 20 30;
@debug sum($list...); // 60
@debug sum((10 20 30) ...); // 60
```

Мы можем передать в функцию данные всех типов и вернуть из функции данные всех типов:

```
@function test($value) {
 @debug $value;
 @return $value;
}
@debug test(null);
@debug test(true);
@debug test(10px);
@debug test("string");
@debug test(#ff0000);
@debug test(5px 10px 15px 0);
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@debug test($colors);
```

#### 5.9.4. Необязательные параметры

Чтобы сделать некоторые параметры необязательными, следует в определении функции присвоить параметру начальное значение. В этом случае если при вызове функции значение не указано, то переменной будет присвоено это начальное значение. Пример:

```
@function sum($x, $y: 20, $z: 30) {
 @debug "$x = #{ $x }";
 @debug "$y = #{ $y }";
 @debug "$z = #{ $z }";
 @return $x + $y + $z;
}
@debug sum(10); // 60
@debug sum(10, 20); // 60
@debug sum(10, 5, 3); // 18
@debug sum(10, $y: 5, $z: 6); // 21
@debug sum(10, $z: 6, $y: 5); // 21
@debug sum(10, $z: 6); // 36
```

Следует заметить, что необязательные параметры должны следовать после обязательных, чтобы можно было использовать позиционное присваивание. В противном случае необходимо указывать название параметра:

```
@function sum($x, $y: 20, $z) { // Лучше так не делать!
 @debug "$x = #{ $x }";
 @debug "$y = #{ $y }";
 @debug "$z = #{ $z }";
 @return $x + $y + $z;
}
// @debug sum(10, 3); // Error: Missing argument $z.
@debug sum(10, $z: 3); // 33
```

### 5.9.5. Передача произвольного количества значений

Количество значений, переданных функции, может быть произвольным. Для этого в определении функции после названия параметра следует указать три точки. Параметр с тремя точками должен быть расположен последним в списке параметров. В качестве примера напишем функцию суммирования произвольного количества чисел:

```
@function sum($numbers...) {
 $result: 0;
 @each $value in $numbers {
 $result: $result + $value;
 }
 @return $result;
}
@debug sum(10); // 10
@debug sum(10, 20); // 30
@debug sum(10, 20, 30); // 60
@debug sum(10, 20, 30, 40); // 100
```

Если ожидается произвольное количество значений, то их может не быть и вовсе:

```
@debug sum(); // 0
```

Если в определении функции существует параметр с тремя точками, то при вызове функции можно передать любое количество именованных параметров со значениями. Получить эти параметры позволяют функции `meta.keywords($args)` и `keywords($args)`, которые возвращают ассоциативный массив, ключами которого будут названия параметров без символа `⋄`:

```
@use "sass:meta";

@function test($args...) {
 @each $key, $value in meta.keywords($args) {
 @debug "#{$key} => #{$value}";
 }
 @return 0;
}
@debug test($x: 10, $y: 20, $z: 30);
```

Результат в окне консоли:

```
scss\main.scss:5 Debug: x => 10
scss\main.scss:5 Debug: y => 20
scss\main.scss:5 Debug: z => 30
scss\main.scss:9 Debug: 0
```

### 5.9.6. Передача ссылки на функцию и обратный вызов

Функции `meta.get-function()` и `get-function()` возвращают функцию с названием `$name`. Форматы функций:

```
@use "sass:meta";
meta.get-function($name[, $scss: false][, $module: null])
get-function($name[, $scss: false][, $module: null])
```

Параметр `$name` принимает название функции в виде строки. Это может быть название пользовательской функции или встроенной функции из Sass. Давайте подключим модуль `meta`, определим две функции и получим ссылки на них:

```
@use "sass:meta";
@use "sass:string";

@function add($x, $y) {
 @return $x + $y;
}
@function sub($x, $y) {
 @return $x - $y;
}
@debug meta.get-function("add"); // get-function("add")
@debug meta.get-function("sub"); // get-function("sub")
@debug get-function("sub"); // get-function("sub")
@debug type-of(get-function("sub")); // function
@debug get-function("darken"); // get-function("darken")
```

Если функция с указанным названием не существует, то выводится сообщение об ошибке. Однако если в параметре `$css` указать значение `true`, то считается допустимым любое название:

```
// @debug meta.get-function("sum"); // Error: Function not found: "sum"
@debug meta.get-function("sum", $css: true); // get-function("sum")
```

По умолчанию поиск функции выполняется в текущем пространстве имен. Если функция расположена внутри модуля, то в параметре `$module` следует указать название пространства имен в виде строки:

```
@debug get-function("slice", $module: "string"); // get-function("slice")
```

Чтобы вызвать функцию, следует воспользоваться функциями `meta.call()` и `call()`.  
**Форматы функций:**

```
@use "sass:meta";
meta.call($function, <Значения>...)
call($function, <Значения>...)
```

Параметр `$function` принимает значение, возвращаемое функциями `meta.get-function()` и `get-function()`. Далее через запятую указываются значения, передаваемые функции `$function`. Функции `meta.call()` и `call()` возвращают результат выполнения функции `$function`. Давайте вызовем наши функции и выведем результат:

```
$fadd: meta.get-function("add");
$fsub: meta.get-function("sub");
@debug meta.call($fadd, 5, 3); // 8
@debug call($fsub, 5, 3); // 2
```

Зачем нам так сложно вызывать функции, если мы знаем, что достаточно указать название функции и внутри круглых скобок передать значения?

```
@debug add(5, 3); // 8
@debug sub(5, 3); // 2
```

Дело в том, что этим способом можно передать ссылку на функцию в качестве параметра другой функции, а внутри этой функции вызвать ее:

```
@function test($a, $b, $func) {
 $result: call($func, $a, $b);
 @if unitless($result) {
 $result: $result * 1px;
 }
 @return $result;
}
@debug test(5, 3, get-function("add")); // 8px
@debug test(5, 3, get-function("sub")); // 2px
```

Функции, передаваемые в другие функции, называются *функциями обратного вызова*.

### 5.9.7. Проверка существования функции

Проверить существование пользовательской или встроенной функции позволяют функции `meta.function-exists($name)` и `function-exists($name)`. В параметре `$name` название функции указывается в виде строки. Если функция с названием `$name` определена, то возвращается значение `true`, в противном случае — `false`. Пример:

```
@use "sass:meta";

@debug meta.function-exists("add"); // false

@function add($x, $y) {
 @return $x + $y;
}
@debug meta.function-exists("add"); // true
@debug function-exists("darken"); // true
```

## 5.10. Модули

При увеличении размера программы определений функций становится все больше и больше. В этом случае программу разделяют на несколько отдельных файлов — *модулей*. Модули также используются для повышения удобства работы с кодом. Например, один модуль предназначен для объявления переменных, второй — для определения функций, третий — для миксинов, четвертый — для стилизации компонента и т. д. В главном файле модули подключаются по мере необходимости, и их содержимое становится частью большой программы.

Давайте вынесем функцию `add()` из *разд. 5.9.7* в модуль с названием `_functions.scss`. Файл размещаем рядом с файлом `main.scss`. Обратите внимание: название модуля начинается с подчеркивания. Это сделано для того, чтобы файл по отдельности не компилировался, если мы отслеживаем изменения всех файлов внутри каталога. Дополнительно создадим модуль `_variables.scss` и добавим в него объявления двух переменных:

```
$primary: #007bff !default;
$secondary: #6c757d !default;
```

### 5.10.1. Директива `@import`

Подключить модули можно несколькими способами. Первый способ заключается в использовании директивы `@import`:

```
@import <Модуль 1>[, ..., <Модуль N>];
```

Директива `@import` считается устаревшей и не рекомендуется к использованию, но очень часто встречается в коде. Название модуля указывается внутри кавычек без символа подчеркивания и расширения файла. Обратите внимание: нельзя вместо названия модуля указать переменную или выполнить подстановку переменной в строку. Давайте подключим модуль `_functions.scss`:

```
@import "functions";

div {
 width: add(500px, 100px); // width: 600px;
}
```

Если указано только название, то будет выполнен поиск файла с названием, начинающимся с подчеркивания и без подчеркивания, а также с расширениями файлов `scss`, `sass` и `css`. По умолчанию поиск выполняется в текущем каталоге. Если найдено несколько файлов с подходящими характеристиками, то будет выведено сообщение об ошибке.

Все содержимое модуля целиком вставляется вместо инструкции с директивой `@import`. Поэтому очень важно учитывать местоположение директивы. Чтобы функции и переменные были доступны всему коду, модули с ними следует подключать в самом начале основного файла.

Если необходимо подключить несколько модулей, то можно вставить несколько инструкций с директивой `@import`. Подключение каждого модуля по отдельности позволяет в случае необходимости закомментировать инструкцию, если она временно не нужна. Пример подключения двух модулей:

```
@import "functions";
@import "variables";
```

Можно также привести названия модулей через запятую:

```
@import "functions", "variables";
```

В следующих случаях получится вызов инструкции `@import` из CSS:

- ◆ файл указан с расширением `css` (если нужно вставить содержимое CSS-файла, то укажите только название):

```
@import "test.css";
```

- ◆ путь содержит название протокола `http` или `https`;
- ◆ используется CSS-функция `url()`;
- ◆ указан медиазапрос.

## 5.10.2. Вложенные директивы `@import`

Допускается вставка директивы `@import` внутрь блока — например, с определениями стилей:

```
div {
 @import "functions";
 width: add(500px, 100px); // width: 600px;
}
```

В этом случае следует учитывать, что стили из модуля вставляются на место директивы `@import`, что может привести к созданию вложенных правил стиля. Давайте создадим модуль `_styles.scss` со следующим содержимым:

```
p {
 color: red;
}
```

Теперь подключим этот модуль внутри блока в основном файле:

```
div {
 @import "styles";
 width: 600px;
}
```

После подстановки получим такую конструкцию с вложенным правилом:

```
div {
 p {
 color: red;
 }
 width: 600px;
}
```

При преобразовании вложенное правило будет обработано, и мы получим следующий результат в CSS-файле:

```
div {
 width: 600px;
}
div p {
 color: red;
}
```

## 5.10.3. Пути поиска модулей

Для удобной работы можно создать каталог с названием, соответствующим тематике модулей, и добавить файлы в него. Давайте в каталоге `C:\book\p1\scss` создадим каталог с названием `utilities` и внутрь него добавим файл `_test.scss` со следующим содержимым:

```
@function sub($x, $y) {
 @return $x - $y;
}
```

Чтобы подключить файл `_test.scss` в `main.scss`, следует перед названием файла указать путь. В нашем случае добавляем название каталога и прямой слеш, тем самым

указывая относительный путь. Обратные слешы в пути, принятые в Windows, использовать нельзя. Давайте подключим модуль и вызовем функцию:

```
@import "utilities/test";
```

```
@debug sub(500px, 300px); // 200px
```

Вместо указания пути внутри файла можно в командной строке передать пути поиска модулей с помощью флага `--load-path=<Путь>`:

```
sass --load-path=scss/utilities scss/main.scss dist/css/main.css
```

или с помощью флага `-I <Путь>`:

```
sass -I scss/utilities scss/main.scss dist/css/main.css
```

В этом случае в директиве `@import` достаточно указать только название модуля:

```
@import "test";
```

```
@debug sub(500px, 300px); // 200px
```

Допускается указание сразу нескольких флагов `-I`, чтобы задать несколько путей поиска модулей. Давайте установим последнюю версию библиотеки Bootstrap с помощью NPM:

```
C:\book\p1>npm i bootstrap
```

Библиотека будет загружена в каталог `C:\book\p1\node_modules`, а в файл `package.json` автоматически добавится раздел `dependencies` с зависимостью:

```
"dependencies": {
 "bootstrap": "^4.5.2"
}
```

Подключим модули `_functions.scss` и `_variables.scss` из библиотеки Bootstrap, а также наш модуль `_test.scss`:

```
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "test";
```

```
@debug $gray-100; // #f8f9fa
```

```
@debug sub(500px, 300px); // 200px
```

Укажем пути поиска модулей в командной строке:

```
C:\book\p1>sass -I scss/utilities -I node_modules scss/main.scss
dist/css/main.css
scss\main.scss:5 Debug: #f8f9fa
scss\main.scss:6 Debug: 200px
```

Мы передали путь до каталога `C:\book\p1\scss\utilities`, а также путь до каталога с установленными библиотеками `C:\book\p1\node_modules`. Благодаря этому мы теперь имеем доступ ко всем библиотекам, а не только к Bootstrap. В директиве `@import` достаточно указать название библиотеки и путь к модулю внутри нее. Благодаря этому не возникнет конфликт имен.

#### **НА ЗАМЕТКУ**

Сначала поиск модуля выполняется относительно текущего каталога. Если модуль не найден, то просматриваются пути поиска.

### 5.10.4. Индексные файлы

Если в каталоге с модулями создать файл `_index.scss` и внутри него подключить модули из каталога, то в основном файле достаточно будет в директиве `@import` указать название каталога. Давайте в каталоге `C:\book\p1\scss\utilities` создадим файл `_index.scss` со следующим содержимым:

```
@import "test";
```

Теперь подключим модуль `_test.scss` внутри файла `main.scss`, указав лишь название каталога `utilities`:

```
@import "utilities";
```

```
@debug sub(500px, 300px); // 200px
```

### 5.10.5. Директива `@use`

Второй способ подключения модулей заключается в использовании директивы `@use`:

```
@use <Модуль>[as <Пространство имен>];
```

Название модуля указывается внутри кавычек без символа подчеркивания и расширения файла. Обратите внимание: нельзя вместо названия модуля указать переменную или выполнить подстановку переменной в строку. Давайте подключим модули `_functions.scss` и `_variables.scss`:

```
@use "functions";
```

```
@use "variables";
```

```
@debug variables.$primary; // #007bff
div {
 width: functions.add(500px, 100px); // width: 600px;
}
```

Если указано только название, то будет выполнен поиск файла с названием, начинающимся с подчеркивания и без подчеркивания, а также с расширениями файлов `scss`, `sass` и `css`. По умолчанию поиск выполняется в текущем каталоге. Если найдено несколько файлов с подходящими характеристиками, то будет выведено сообщение об ошибке.

Основные отличия директивы `@use` от `@import`:

- ◆ с помощью директивы `@use` можно подключить только один модуль внутри одной инструкции;
- ◆ модуль импортируется только один раз;
- ◆ директивы `@use` должны быть расположены в самом начале файла. Раньше них могут быть расположены объявления переменных, комментарии и директивы `@forward`;
- ◆ нельзя вложить директиву `@use` внутрь блока;
- ◆ идентификаторы из модуля помещаются в именованное пространство имен, а не в глобальное пространство имен, что позволяет избежать конфликта имен. На-

званием пространства имен является последний компонент в пути без расширения файла. Чтобы получить доступ к идентификатору, необходимо указать название пространства имен, точку и название идентификатора:

```
@debug variables.$primary;
```

- ◆ импортированные идентификаторы недоступны для последующих операций импорта;
- ◆ с помощью директивы @use можно подключить встроенные модули:

```
@use "sass:math";
```

Чтобы подключить модуль, расположенный в другом каталоге, следует перед названием модуля указать путь. Например, добавляем название каталога и прямой слеш, тем самым указывая относительный путь к вложенному каталогу. Обратные слешы в пути, принятые в Windows, использовать нельзя. Давайте подключим модуль `_test.scss` и вызовем функцию `sub()`:

```
@use "utilities/test";
```

```
@debug test.sub(500px, 300px); // 200px
```

Вместо указания пути внутри файла, можно в командной строке передать пути поиска модулей с помощью флага `--load-path=<Путь>`:

```
sass --load-path=scss/utilities scss/main.scss dist/css/main.css
```

Или с помощью флага `-I <Путь>`:

```
sass -I scss/utilities scss/main.scss dist/css/main.css
```

В этом случае в директиве @use достаточно указать только название модуля:

```
@use "test";
```

```
@debug test.sub(500px, 300px); // 200px
```

#### **НА ЗАМЕТКУ**

Сначала поиск модуля выполняется относительно текущего каталога. Если модуль не найден, то просматриваются пути поиска.

Если в каталоге с модулями создать файл `_index.scss` и внутри него подключить модули из каталога, то в основном файле достаточно будет в директиве @use указать название каталога. Давайте подключим модуль `_test.scss` (см. *разд. 5.10.4*) из каталога `C:\book\p1\scss\utilities` внутри файла `main.scss`, указав лишь название каталога:

```
@use "utilities";
```

```
@debug utilities.sub(500px, 300px); // 200px
```

### **5.10.6. Изменение названия пространства имен**

При использовании директивы @use идентификаторы из модуля помещаются в именованное пространство имен, а не в глобальное пространство имен, что позволяет избежать конфликта имен. Названием пространства имен является последний ком-

понент в пути без расширения файла. Чтобы получить доступ к идентификатору необходимо указать название пространства имен, точку и название идентификатора:

```
@use "variables";
```

```
@debug variables.$primary; // #007bff
```

Каждый раз указывать длинное название пространства имен не очень удобно. С помощью инструкции `as` можно изменить название, так как вам захочется:

```
@use "variables" as v;
```

```
@debug v.$primary; // #007bff
```

### 5.10.7. Импорт всех идентификаторов из модуля

Если в конструкции `as` вместо названия пространства имен указать символ `*`, то все идентификаторы из модуля будут импортированы в глобальное пространство имен, что может привести к конфликту имен:

```
@use "variables" as *;
```

```
@debug $primary; // #007bff
```

Идентификаторы из встроенных модулей также можно импортировать в глобальное пространство имен:

```
@use "sass:math" as *;
```

```
@debug $pi; // 3.1415926536
```

### 5.10.8. Частные идентификаторы внутри модуля

Если перед идентификатором указать дефис или символ подчеркивания, то такой идентификатор будет доступен только внутри модуля, в котором создан. Импортировать частный идентификатор из модуля с помощью директивы `@use` нельзя. Давайте изменим содержимое файла `_functions.scss` следующим образом:

```
$_black: #000;
```

```
@function get-color($c: $_black) {
 @return _test($c);
}
```

```
@function _test($color) {
 @return lighten($color, 25%);
}
```

Переменная `$_black` и функция `_test()` являются частными идентификаторами. Мы можем их использовать внутри файла `_functions.scss` как обычные идентификаторы. Если попробовать подключить модуль с помощью директивы `@use`, то доступной будет только функция `get-color()`:

```
@use "functions";
```

```
@debug functions.get-color(); // #404040
```

```
@debug functions.get-color(gray); // silver
```

```
// @debug functions.$_black;
// Error: Private members can't be accessed from outside their modules.
// @debug functions._test(gray);
// Error: Private members can't be accessed from outside their modules.
```

Если подключить модуль с помощью директивы `@import`, то все частные идентификаторы будут доступны:

```
@import "functions";

@debug get-color(); // #404040
@debug get-color(gray); // silver
@debug $_black; // #000
@debug _test(gray); // silver
```

### 5.10.9. Переопределение значений переменных из модуля

Если внутри модуля при объявлении переменной указана инструкция `!default`, то можно будет изменить значение переменной при подключении модуля, не изменяя исходный код модуля. Сначала нужно объявить одноименную переменную с новым значением, а затем подключить модуль с помощью директивы `@import`. Давайте изменим значение переменной `$primary` из библиотеки Bootstrap:

```
$primary: red;
```

```
@import "bootstrap/scss/bootstrap";
```

В директиве `@import` мы указали путь к файлу `bootstrap.scss`, внутри которого подключаются все остальные модули библиотеки Bootstrap. Выполним компиляцию, указав в командной строке путь поиска модулей:

```
C:\book\p1>sass -I node_modules scss/main.scss dist/css/bootstrap.css
```

В результате в каталоге `C:\book\p1\dist\css` будет создан файл `bootstrap.css`, внутри которого изменилось значение цвета для всех стилей с темой `primary`:

```
.badge-primary {
 color: #fff;
 background-color: red;
}
```

Таким способом можно изменять значения отдельных переменных, создавая сборку библиотеки Bootstrap под свои потребности. Но не торопитесь использовать этот файл в реальном проекте, т. к. после компиляции необходимо выполнить некоторые дополнительные действия, которые мы рассмотрим немного позже (см. *разд. 5.15*).

При использовании директивы `@use` переопределить значение переменной таким же образом нельзя. В этом случае следует воспользоваться следующим форматом директивы `@use`:

```
@use <Модуль> with (
 <Переменная 1>: <Значение 1>,
 ...,
 <Переменная N>: <Значение N>);
```

Если переменная внутри модуля не содержит инструкции `!default`, то будет выведено сообщение об ошибке:

```
Error: This variable was not declared with !default in the @used module.
```

Переопределим значение переменной `$primary` из модуля `_variables.scss`:

```
@use "variables" with ($primary: red);
```

```
@debug variables.$primary; // red
@debug variables.$secondary; // #6c757d
```

Изменим значение переменной `$primary` из библиотеки Bootstrap, используя директиву `@use`:

```
@use "bootstrap/scss/bootstrap" with ($primary: red);
```

```
@debug bootstrap.$primary; // red
```

Выполним компиляцию, указав в командной строке путь поиска модулей:

```
C:\book\p1>sass -I node_modules scss/main.scss dist/css/bootstrap.css
```

### 5.10.10. Директива `@forward`

В предыдущем разделе мы смогли изменить переменную из библиотеки Bootstrap только потому, что в файле `bootstrap.scss` подключение модулей производится с помощью директивы `@import`, которая помещает все идентификаторы в глобальное пространство имен. При использовании директивы `@use` импортированные идентификаторы недоступны для последующих операций импорта, поэтому мы получим ошибку.

Давайте изменим содержимое файла `_styles.scss` следующим образом:

```
@use "variables" as v;
```

```
p {
 color: v.$primary;
}
// @debug v.$-my-color;
// Error: Private members can't be accessed from outside their modules.
```

Частные идентификаторы не импортируются из модуля, поэтому при попытке обращения к переменной `$-my-color` мы получим сообщение об ошибке. Содержимое файла `_variables.scss`:

```
$primary: #007bff !default;
$secondary: #6c757d !default;
$-my-color: red !default;
```

Теперь создадим файл `_bs.scss`, который имитирует файл `bootstrap.scss` из библиотеки Bootstrap. Внутри него подключим файлы `_variables.scss` и `_styles.scss`, а также выведем значение переменной `$primary`:

```
@use "variables";
@use "styles";
```

```
@debug variables.$primary;
```

Попробуем подключить файл `_bs.scss` внутри файла `main.scss` и обратиться к переменной `$primary`:

```
@use "bs";
```

```
@debug bs.$primary; // Error: Undefined variable.
```

В результате получим сообщение, что переменная не существует. Если попробуем переопределить значение переменной `$primary`, то опять получим сообщение об ошибке:

```
@use "bs" with ($primary: red);
// Error: This variable was not declared with !default in the
// @used module.
```

Чтобы иметь возможность переопределить значение переменной или использовать переменную внутри файла `main.scss`, нужно воспользоваться директивой `@forward`:

```
@forward <Модуль>[as <Префикс>-*];
```

Название модуля указывается внутри кавычек без символа подчеркивания и расширения файла. Директива `@forward` подключает модуль и загружает общедоступные идентификаторы в пространство имен текущего модуля. При этом идентификаторы недоступны внутри текущего модуля, но будут доступны при импорте этого модуля. Поэтому следующий код внутри файла `_bs.scss` приведет к ошибке:

```
@forward "variables";
@use "styles";
```

```
@debug variables.$primary;
// Error: There is no module with the namespace "variables".
```

Если мы добавим символы комментария перед инструкцией `@debug`, то сможем получить доступ к переменной `$primary` внутри файла `main.scss`:

```
@use "bs";
```

```
@debug bs.$primary; // #007bff
```

Кроме того, мы теперь можем переопределить значение переменной `$primary` внутри файла `main.scss`:

```
@use "bs" with ($primary: red);
```

```
@debug bs.$primary; // red
```

Результат в CSS-файле:

```
p {
 color: red;
}
```

Что же нам делать, если внутри файла `_bs.scss` мы хотим использовать переменную `$primary`? В этом случае после директивы `@forward` нужно вставить директиву `@use` с импортом того же самого модуля. При этом модуль будет импортирован только один раз:

```
@forward "variables";
@use "variables";
@use "styles";
```

```
@debug variables.$primary; // OK
```

А какое значение будет иметь переменная `$primary` в файле `_bs.scss`? Если значение переменной переопределено в файле `main.scss`, то мы получим это новое значение. Если вы все делали последовательно, как в книге, то получите значение `red`. Если убрать инструкцию `with`, то получим значение `#007bff` и в файле `_bs.scss`, и в файле `main.scss`:

```
@use "bs";
```

```
@debug bs.$primary; // #007bff
```

Директива `@forward` позволяет также переопределить значения переменных:

```
@forward <Модуль> with (
 <Переменная 1>: <Значение 1>[!default],
 ...,
 <Переменная N>: <Значение N>[!default]);
```

Переопределим значение переменной `$primary` в файле `_bs.scss`:

```
@forward "variables" with ($primary: red);
@use "variables";
@use "styles";
```

```
@debug variables.$primary; // red
```

В итоге получим красный цвет и в файле `_bs.scss`, и в файле `main.scss`. Если мы сейчас попробуем повторно переопределить переменную `$primary` в файле `main.scss`, то получим сообщение об ошибке:

```
@use "bs" with ($primary: green);
// Error: This variable was not declared with !default in the
// @used module.
```

В отличие от директивы `@use`, при переопределении значений переменных в директиве `@forward` можно использовать инструкцию `!default`. Давайте ее добавим:

```
@forward "variables" with ($primary: red !default);
@use "variables";
@use "styles";
```

```
@debug variables.$primary;
```

Переопределим значение повторно в файле `main.scss`:

```
@use "bs" with ($primary: green);
```

```
@debug bs.$primary; // green
```

В результате получим зеленый цвет и в файле `_bs.scss`, и в файле `main.scss`, а также в итоговом CSS-файле:

```
p {
 color: green;
}
```

Директива `@forward` позволяет также добавить префикс к именам всех идентификаторов из модуля. Для этого используется следующий синтаксис:

```
@forward <Модуль> as <Префикс>-*;
```

Давайте изменим содержимое файла `_bs.scss` и добавим префикс `bs-` к названиям переменных:

```
@forward "variables" as bs-*;
@use "variables";
@use "styles";

@debug variables.$primary; // #007bff
```

Теперь в файле `main.scss` мы должны указывать префикс `bs-` явным образом:

```
@use "bs" as *;

@debug $bs-primary; // #007bff
```

Импорт некоторых идентификаторов можно запретить. Сделать это позволяют следующие форматы директивы `@forward`:

```
@forward <Модуль> hide <Список идентификаторов через запятую>;
@forward <Модуль> show <Список идентификаторов через запятую>;
```

После ключевого слова `hide` указываются названия запрещенных идентификаторов через запятую. Все остальные общедоступные идентификаторы импортируются. Запретим импорт переменной `$secondary` внутри файла `_bs.scss`:

```
@forward "variables" hide $secondary;
@use "styles";
```

Теперь при импорте идентификатора из модуля `_bs.scss` не будет доступа к переменной `$secondary`:

```
@use "bs";

@debug bs.$primary; // #007bff
// @debug bs.$secondary; // Error: Undefined variable.
```

После ключевого слова `show` указываются названия разрешенных к импорту идентификаторов через запятую. Все остальные идентификаторы не импортируются. Разрешим импорт только переменной `$primary` внутри файла `_bs.scss`:

```
@forward "variables" show $primary;
@use "styles";
```

При импорте модуля не будет доступа к переменной `$secondary`:

```
@use "bs";

@debug bs.$primary; // #007bff
// @debug bs.$secondary; // Error: Undefined variable.
```

### 5.10.11. Миксин `meta.load-css()`

Директива `@use` может быть указана только в начале файла. Если нужно вставить содержимое модуля или CSS-файла в произвольное место, то она нам не поможет. В этом случае нужно воспользоваться миксином `meta.load-css()`:

```
@use "sass:meta";
meta.load-css(<Модуль>[, $with: null]);
```

В первом параметре указывается путь и/или название модуля без расширения. Модуль загружается только один раз. Вместо названия модуля можно передать путь к CSS-файлу. Второй параметр позволяет переопределить значения переменных с инструкцией `!default`. В параметре `$with` названия переменных и их значения задаются в виде ассоциативного массива. Причем название переменной указывается без символа `$` внутри кавычек. Если модуль был загружен ранее, то переопределить значения переменных нельзя.

В отличие от директивы `@use`, миксин `meta.load-css()`:

- ◆ не делает доступными идентификаторы из загруженного модуля внутри текущего пространства имен;
- ◆ может быть расположен в любом месте, включая вложение внутрь блока;
- ◆ допускает динамическое формирование пути и названия модуля с помощью переменной или подстановки значения переменной в строку.

Содержимое модуля можно вставить с помощью директивы `@include`:

```
@include meta.load-css("../");
```

Давайте изменим содержимое файла `_styles.scss` следующим образом:

```
$color: black !default;
$bg-color: white !default;
```

```
body {
 color: $color;
 background-color: $bg-color;
}
```

Теперь подключим этот модуль внутри файла `main.scss`:

```
@use "sass:meta";
```

```
@include meta.load-css("styles");
```

Результат в CSS-файле после компиляции:

```
body {
 color: black;
 background-color: white;
}
```

Переопределим значения переменных внутри файла `main.scss`:

```
@use "sass:meta";
```

```
@include meta.load-css("styles",
 $with: ("color": red, "bg-color": silver));
```

Результат в CSS-файле после компиляции:

```
body {
 color: red;
 background-color: silver;
}
```

## 5.10.12. Подключение CSS-файлов

Помимо модулей, директивы `@import`, `@use` и `@forward` позволяют подключать CSS-файлы. В этом случае название файла указывается без расширения. Давайте в каталоге `C:\book\p1\scss` создадим файл `mystyle.css` со следующим содержимым:

```
div {
 width: 600px;
}
```

Подключим его в файле `main.scss` с помощью директивы `@import`:

```
@import "mystyle";
```

Результат в CSS-файле после компиляции:

```
div {
 width: 600px;
}
```

```
/*# sourceMappingURL=main.css.map */
```

Если указать расширение файла:

```
@import "mystyle.css";
```

то в итоговом CSS-файле получим инструкцию импорта из CSS, а не содержимое файла:

```
@import "mystyle.css";
```

```
/*# sourceMappingURL=main.css.map */
```

Пример подключения CSS-файла с помощью директивы `@use`:

```
@use "mystyle";
```

Подключить CSS-файл можно также с помощью миксина `meta.load-css()` (см. *разд. 5.10.11*), причем его вызов может быть выполнен в любом месте файла, включая блоки. Пример подключения CSS-файла внутри блока:

```
@use "sass:meta";
```

```
.container {
 @include meta.load-css("mystyle");
 color: black;
}
```

Результат в CSS-файле:

```
.container {
 color: black;
}
.container div {
 width: 600px;
}
```

### 5.10.13. Получение всех переменных внутри модуля

Функция `meta.module-variables(<Пространство имен>)` возвращает ассоциативный массив, ключами которого являются названия доступных переменных (без символа `$`) из указанного пространства имен, а значениями — значения этих переменных. Перед использованием функции не забудьте подключить модуль `meta`:

```
@use "sass:meta";
```

Получим все доступные переменные из модуля `_variables.scss`:

```
@use "sass:meta";
```

```
@use "variables";
```

```
@debug meta.module-variables("variables");
// ("primary": #007bff, "secondary": #6c757d)
```

Содержимое файла `_variables.scss`:

```
$primary: #007bff !default;
$secondary: #6c757d !default;
$-my-color: red !default;
```

Частные идентификаторы не импортируются директивой `@use`, поэтому в результате нет переменной `$-my-color`.

### 5.10.14. Получение всех функций внутри модуля

Чтобы получить все доступные функции из указанного пространства имен, следует воспользоваться функцией `meta.module-functions(<Пространство имен>)`. Функция возвращает ассоциативный массив, ключами которого являются названия функций в виде строки, а значениями — ссылки на функции (см. *разд. 5.9.6*), с помощью которых можно вызвать функцию. Перед использованием функции не забудьте подключить модуль `meta`:

```
@use "sass:meta";
```

Получим все доступные функции из модуля `_functions.scss` и вызовем функцию `add()`:

```
@use "sass:meta";
```

```
@use "sass:map";
```

```
@use "functions";
```

```
@debug meta.module-functions("functions");
// ("add": get-function("add"), "sub": get-function("sub"))
```

```
$func: map.get(meta.module-functions("functions"), "add");
```

```
@debug meta.call($func, 5, 3); // 8
```

Содержимое файла `_functions.scss`:

```
@function add($x, $y) {
 @return $x + $y;
}
@function sub($x, $y) {
```

```

 @return $x - $y;
 }
 @function _test() {
 @return 0;
 }

```

Частные идентификаторы не импортируются директивой @use, поэтому в результате нет функции \_test().

## 5.11. Работа с селекторами и атрибутами стилей

В SCSS-файле можно использовать те же самые конструкции, что и в CSS-файле:

```

<Селектор> {
 <Атрибут 1>: <Значение 1>;
 ...;
 <Атрибут N>: <Значение N>;
}

```

Перед названием атрибута обычно добавляют два пробела. После определения стиля указывается точка с запятой. Исключением может быть последнее определение стиля, после которого точку с запятой указывать необязательно, но лучше все-таки ее ставить, т. к. при добавлении новых правил можно просто забыть добавить точку с запятой:

```

p {
 color: #000000;
 background-color: #ffffff;
}

```

Если точка с запятой отсутствует, то при преобразовании будет выведено сообщение об ошибке:

```

C:\book\p1>sass scss/main.scss dist/css/main.css
Error: expected ";".

```

Основное отличие от обычного CSS-файла заключается в том, что можем использовать переменные и другие конструкции Sass, которые мы уже рассмотрели ранее и будем рассматривать в следующих разделах. При преобразовании выполняется форматирование и добавляется точка с запятой после последнего правила, даже если она отсутствовала:

```

$color: #000000 !default;
$bg-color: #ffffff !default;

p {
 color: $color; background-color: $bg-color;
}

```

Результат в CSS-файле:

```

p {
 color: #000000;
 background-color: #ffffff;
}

```

Стили можно привязать сразу к нескольким селекторам, в этом случае селекторы указываются через запятую:

```
p, div {
 color: #000000;
 background-color: #ffffff;
}
```

Для работы с селекторами в Sass предназначен модуль `selector`. Подключить модуль позволяет следующая инструкция (в интерактивном режиме точку с запятой указывать не нужно):

```
@use "sass:selector";
```

Для доступа к идентификаторам внутри модуля, следует указать название модуля, точку, а затем идентификатор. Некоторые идентификаторы доступны в глобальной области видимости. Перед такими идентификаторами в следующих разделах мы не будем указывать название модуля.

### 5.11.1. Основные селекторы

Давайте вспомним основные селекторы:

- ◆ `*` — все теги. Уберем все внешние и внутренние отступы:

```
* {
 margin: 0;
 padding: 0;
}
```

- ◆ `Тег` — все теги, имеющие указанное имя:

```
p {
 color: #000000;
 background-color: #ffffff;
}
```

- ◆ `.Класс` — все теги, имеющие указанный класс (название стилевого класса задается в параметре `class`):

```
.card-title {
 margin-bottom: 0;
}
```

- ◆ `Тег.Класс` — все теги, имеющие указанное имя и класс:

```
div.card-title {
 margin-bottom: 0;
}
```

- ◆ `#Идентификатор` — элемент с указанным идентификатором (идентификатор задается в параметре `id`):

```
#txt1 {
 color: red;
}
```

Можно указывать любые другие селекторы из CSS — например, селекторы с привязкой к параметрам тегов, а также псевдоэлементы и псевдоклассы. Далее мы рассмотрим только различия и возможности, добавляемые Sass.

## 5.11.2. Привязка к элементам и вложенные правила

Привязаться к элементам можно следующими способами:

- ◆ *Селектор1 Селектор2* — все элементы, соответствующие параметру *Селектор2*, которые располагаются внутри контейнера, соответствующего параметру *Селектор1*:

```
div a {
 color: red;
}
```

Цвет текста ссылки станет красным, если тег `<a>` находится внутри тега `<div>`:

```
<div>Ссылка</div>
```

Чтобы привязка выглядела более наглядно, можно в SCSS-файле создать *вложенное правило*:

```
$bg-color: #ffffff !default;
$link-color: red !default;

div {
 background-color: $bg-color;

 a {
 color: $link-color;
 }
}
```

Результат в CSS-файле:

```
div {
 background-color: #ffffff;
}
div a {
 color: red;
}
```

- ◆ *Селектор1 > Селектор2* — все элементы, соответствующие параметру *Селектор2*, которые являются дочерними для контейнера, соответствующего параметру *Селектор1*:

```
div > a {
 color: red;
}
```

Цвет текста ссылки станет красным, если тег `<a>` находится внутри тега `<div>` и не вложен в другой тег. В следующем примере только первая ссылка станет красного цвета, т. к. вторая ссылка расположена внутри тега `<span>`:

```
<div>
 Ссылка 1

 Ссылка 2
</div>
```

Чтобы привязка выглядела более наглядно, можно в SCSS-файле создать вложенное правило:

```
div {
 background-color: #ffffff;

 > a {
 color: red;
 }
}
```

Результат в CSS-файле:

```
div {
 background-color: #ffffff;
}
div > a {
 color: red;
}
```

- ◆ *Селектор1 + Селектор2* — элемент, соответствующий параметру *Селектор2*, который является соседним для элемента, соответствующего параметру *Селектор1*, и следует сразу после него:

```
div + p {
 color: red;
}
```

Цвет текста абзаца станет красным, если тег `<p>` следует сразу после тега `<div>`:

```
<div>Текст</div><p>Текст</p>
```

Чтобы привязка выглядела более наглядно, можно в SCSS-файле создать вложенное правило:

```
div {
 background-color: #ffffff;

 + p {
 color: red;
 }
}
```

Результат в CSS-файле:

```
div {
 background-color: #ffffff;
}
div + p {
 color: red;
}
```

- ◆ *Селектор1 ~ Селектор2* — элемент, соответствующий параметру *Селектор2*, который является соседним для элемента, соответствующего параметру *Селектор1*, и следует после него, причем необязательно непосредственно:

```
div ~ h6 {
 color: red;
}
```

Цвет текста заголовка станет красным, если тег `<h6>` следует за тегом `<div>` и, возможно, отделяется от него другими элементами:

```
<div>Текст 1</div>
<p>Текст 2</p>
<h6>Красный заголовок</h6>
```

Чтобы привязка выглядела более наглядно, можно в SCSS-файле создать вложенное правило:

```
div {
 background-color: #ffffff;

 ~ h6 {
 color: red;
 }
}
```

Результат в CSS-файле:

```
div {
 background-color: #ffffff;
}
div ~ h6 {
 color: red;
}
```

Можно также воспользоваться следующими форматами при вложении:

```
<Селектор> {
 >|+|~ {
 <Селектор> { ... }
 ...
 <Селектор> { ... }
 }
}
<Селектор> >|+|~ {
 <Селектор> { ... }
 ...
 <Селектор> { ... }
}
```

**Пример:**

```
div {
 > {
 a { color: red; }
 span { color: blue; }
 }
}
p + {
 a { color: red; }
 span { color: blue; }
}
```

Результат в CSS-файле:

```
div > a {
 color: red;
}
div > span {
 color: blue;
}
p + a {
 color: red;
}
p + span {
 color: blue;
}
```

При необходимости можно составлять выражения из нескольких селекторов:

```
div span a {
 color: red;
}
```

Цвет текста ссылки станет красным, если тег `<a>` расположен внутри тега `<span>`, а тот в свою очередь вложен в тег `<div>`. В этом примере только *ссылка 2* будет красного цвета:

```
<div>
 Ссылка 1

 Ссылка 2

</div>
```

Чтобы привязки выглядели более наглядно, можно в SCSS-файле создать вложенные правила:

```
$link-color: red !default;

div {
 span {
 a {
 color: $link-color;
 }
 }
}
```

Результат в CSS-файле:

```
div span a {
 color: red;
}
```

### 5.11.3. Директива `@at-root`

Директива `@at-root` сообщает Sass, что для вложенного правила не нужно указывать привязку к родительским селекторам. Основные форматы директивы:

```
@at-root <Селектор> { ... }
@at-root {
```

```
<Селектор> { ... }
...
<Селектор> { ... }
}
```

### Пример:

```
div {
 background-color: #ffffff;

 @at-root a {
 color: red;
 }
}
```

### Результат в CSS-файле:

```
div {
 background-color: #ffffff;
}
a {
 color: red;
}
```

Если мы уберем директиву @at-root, то результат будет таким:

```
div {
 background-color: #ffffff;
}
div a {
 color: red;
}
```

Директива @at-root может быть указана на любом уровне вложенности. Пример использования второго формата:

```
div {
 p {
 @at-root {
 a {
 color: red;
 }
 span {
 color: blue;
 }
 }
 }
}
```

### Результат в CSS-файле:

```
a {
 color: red;
}
span {
 color: blue;
}
```

Директива `@at-root` позволяет также убрать привязку к директивам — например, `@media` и `@supports`. Для этого используются следующие форматы:

```
@at-root (without: ...) { ... }
@at-root (with: ...) { ... }
```

Директивы (без символа `@`) указываются после ключевых слов `without` и `with` через пробел. Уберем привязку к директиве `@media`:

```
@media print {
 div {
 background-color: #ffffff;
 @at-root (without: media) {
 color: red;
 }
 }
}
```

Результат в CSS-файле:

```
@media print {
 div {
 background-color: #ffffff;
 }
}
div {
 color: red;
}
```

После ключевых слов `without` и `with` допускается указание специальных значений:

◆ `all` — все селекторы и директивы:

```
@media print {
 div {
 background-color: #ffffff;
 @at-root (without: all) {
 span { color: red; }
 }
 }
}
```

Результат в CSS-файле:

```
@media print {
 div {
 background-color: #ffffff;
 }
}
span {
 color: red;
}
```

◆ `rule` — только все селекторы:

```
@media print {
 div {
 background-color: #ffffff;
```

```

 @at-root (without: rule) {
 span { color: red; }
 }
 }
}

```

Результат в CSS-файле:

```

@media print {
 div {
 background-color: #ffffff;
 }
 span {
 color: red;
 }
}

```

Ключевое слово `without` исключает указанные правила, тогда как `with` исключает все, кроме указанных правил. Уберем привязку к директиве `@media`:

```

@media print {
 div {
 background-color: #ffffff;
 @at-root (with: rule) {
 span { color: red; }
 }
 }
}

```

Результат в CSS-файле:

```

@media print {
 div {
 background-color: #ffffff;
 }
}
div span {
 color: red;
}

```

#### 5.11.4. Оператор `&`: список с родительскими селекторами

Оператор `&` содержит список с родительскими селекторами:

```

body {
 @debug &; // (body,)
}
p span, div span {
 @debug &; // p span, div span
}
p {
 span {
 @debug &; // (p span,)
 }
}

```

Пример изменения значений при наведении указателя мыши на элемент:

```
a {
 text-decoration: none;
 &:hover { text-decoration: underline; }
}
```

Результат в CSS-файле:

```
a {
 text-decoration: none;
}
a:hover {
 text-decoration: underline;
}
```

Если указать символ & в начале селектора, то к родительскому селектору будет добавлен указанный суффикс:

```
#test {
 color: red;
 &-color { color: blue; }
}
```

Результат в CSS-файле:

```
#test {
 color: red;
}
#test-color {
 color: blue;
}
```

### 5.11.5. Вложенные атрибуты

Некоторые CSS-атрибуты содержат одинаковый префикс — например: `margin-left`, `margin-right`, `margin-top` и `margin-bottom`. В Sass префикс и двоеточие можно указать перед открывающей фигурной скобкой, а внутри скобок обращаться к атрибутам без префикса:

```
div {
 margin: {
 left: 5px;
 right: 10px;
 top: 15px;
 bottom: 20px;
 }
}
```

Результат в CSS-файле:

```
div {
 margin-left: 5px;
 margin-right: 10px;
 margin-top: 15px;
 margin-bottom: 20px;
}
```

После двоеточия можно указать значение для обобщенного атрибута:

```
div {
 margin: auto {
 top: 15px;
 bottom: 20px;
 }
}
```

Результат в CSS-файле:

```
div {
 margin: auto;
 margin-top: 15px;
 margin-bottom: 20px;
}
```

### 5.11.6. Генерация названий селекторов и атрибутов

Переменную Sass мы можем указать в качестве значения атрибута стиля, но попытка вставки переменной вместо селектора или атрибута приведет к ошибке. Если нужно динамически генерировать название селектора или атрибута на основе переменной, то следует выполнить подстановку значения переменной (см. *разд. 5.2.6*):

```
$prefix: "background";
$property: "color";
$item1: ".item-1" !default;
$n: 2 !default;
$color: red !default;
$bg-color: #ffffff !default;

#{ $item1 } {
 #{ $prefix }-#{ $property }: $bg-color;
}
.item-#{ $n } {
 #{ $property }: $color;
}
```

Результат в CSS-файле:

```
.item-1 {
 background-color: #ffffff;
}
.item-2 {
 color: red;
}
```

Вставим стилевой класс в селектор из значения переменной:

```
$class-name: "cls1";
p.#{ $class-name } {
 color: red;
}
```

Результат в CSS-файле:

```
p.cls1 {
 color: red;
}
```

Благодаря подстановке значения переменной и циклам мы можем генерировать произвольное количество правил на основе значений ассоциативного массива:

```
$colors: (
 "blue": #007bff,
 "red": #dc3545
);
@each $name, $color in $colors {
 .item-#{$name} {
 background-color: $color;
 }
}
```

Результат в CSS-файле:

```
.item-blue {
 background-color: #007bff;
}
.item-red {
 background-color: #dc3545;
}
```

### 5.11.7. Вставка атрибута в зависимости от условия

В некоторые случаи не нужно вставлять атрибут в итоговый CSS-файл. Для этого можно проверить некое условие с помощью оператора `@if` (см. *разд. 5.3.5*):

```
$enable-border-radius: true !default;
```

```
div {
 border: 1px solid red;
 @if $enable-border-radius {
 border-radius: 10px;
 }
}
```

Если переменная `$enable-border-radius` имеет значение `true`, то получим следующий результат:

```
div {
 border: 1px solid red;
 border-radius: 10px;
}
```

Если переменная `$enable-border-radius` имеет значение `false`, то результат будет таким:

```
div {
 border: 1px solid red;
}
```

Если значением атрибута является пустая строка или значение `null`, то такой атрибут не попадет в итоговый CSS-файл. Зная этот факт, можно сократить код, используя функцию `if()` (см. *разд. 5.3.5*):

```
$enable-border-radius: true !default;

div {
 border: 1px solid red;
 border-radius: if($enable-border-radius, 10px, null);
}
```

Результат будет таким же, как и в предыдущем примере.

### 5.11.8. Функции для работы с селекторами

Для работы с селекторами предназначены следующие функции:

- ◆ `selector.is-supersselector()` и `is-supersselector()` — возвращают значение `true`, если селектор `$super` входит в селектор `$sub`, и `false` — в противном случае. Форматы функций:

```
@use "sass:selector";

selector.is-supersselector($super, $sub)
is-supersselector($super, $sub)
```

Пример:

```
@use "sass:selector";

@debug selector.is-supersselector("a", "a"); // true
@debug is-supersselector("a", "a:hover"); // true
@debug is-supersselector("a:hover", "a"); // false
```

- ◆ `selector.append($selectors...)` и `selector-append($selectors...)` — объединяют селекторы без пробелов между ними. Если указан составной селектор, то его части объединяются по отдельности:

```
@debug selector.append("a", ":hover");
// (a:hover,)
@debug selector-append("a", ".link1", ".link2");
// a.link1, a.link2
@debug selector-append("a", ".link1", ".link2");
// (a.link1.link2,)
```

- ◆ `selector.nest($selectors...)` и `selector-nest($selectors...)` — объединяют селекторы с пробелами между ними. Если указан составной селектор, то его части объединяются по отдельности:

```
@debug selector.nest("p", ".link1");
// (p .link1,)
@debug selector-nest("p", ".link1", ".link2");
// p .link1, p .link2
@debug selector-nest("p", ".link1", ".link2");
// (p .link1 .link2,)
```

Если указан символ `&`, то он будет заменен и объединение будет выполнено без пробела:

```
@debug selector.nest("a", "&:hover"); // (a:hover,)
```

- ◆ `selector.unify()` и `selector-unify()` — объединяют `$selector1` и `$selector2` и возвращают селектор, который будет соответствовать обоим селекторам. Если объединение невозможно, то функции вернут значение `null`. Форматы функций:

```
@use "sass:selector";
selector.unify($selector1, $selector2)
selector-unify($selector1, $selector2)
```

Пример:

```
@debug selector.unify("a", ":hover");
// (a:hover,)
@debug selector-unify("a.link1", "a.link2");
// (a.link1.link2,)
@debug selector-unify(".cls1 .cls2", "input");
// (.cls1 input.cls2,)
@debug selector-unify(".a .b", ".n .k");
// .a .n .b.k, .n .a .b.k
@debug selector-unify("a", "h1"); // null
```

- ◆ `selector.extend()` и `selector-extend()` — расширяют `$extendee` с `$extender` в пределах `$selector` по правилам директивы `@extend`. Форматы функций:

```
@use "sass:selector";
selector.extend($selector, $extendee, $extender)
selector-extend($selector, $extendee, $extender)
```

Пример:

```
@debug selector.extend("a:hover", "a", ".link1");
// a:hover, .link1:hover
@debug selector-extend(".a .b", ".b", ".c p.d");
// .a .b, .a .c p.d, .c .a p.d
@debug selector-extend("a:hover", "h1", "h2");
// (a:hover,)
```

- ◆ `selector.simple-selectors()` и `simple-selectors()` — разделяют составной селектор `$selector` на простые селекторы и возвращают список с элементами, разделенными запятыми. В составе `$selector` не могут присутствовать пробелы и запятые, иначе будет выведено сообщение об ошибке. Форматы функций:

```
@use "sass:selector";
selector.simple-selectors($selector)
simple-selectors($selector)
```

Пример:

```
@debug selector.simple-selectors("a:hover"); // a, :hover
@debug simple-selectors(".a.b.c"); // .a, .b, .c
```

- ◆ `selector.parse($selector)` и `selector-parse($selector)` — разбирают строку с селектором и возвращают список с отдельными элементами:

```
@debug selector.parse("p span"); // (p span,)
$result: selector-parse("p span, div span");
```

```
@debug $result; // p span, div span
@debug type-of($result); // list
@each $value in $result {
 @debug $value;
}
// p span
// div span
```

- ◆ `selector.replace()` и `selector-replace()` — выполняют замену всех вхождений `$original` на `$replacement` внутри `$selector` и возвращают новое значение. Форматы функций:

```
@use "sass:selector";
selector.replace($selector, $original, $replacement)
selector-replace($selector, $original, $replacement)
```

Пример:

```
@debug selector.replace("p span", "p", "div"); // (div span,)
@debug selector-replace("p span", "span", "a"); // (p a,)
```

Учитывайте, что при замене используются правила директивы `@extend`:

```
@debug selector-replace("p span", "span", ".cls1 a.link1");
// p .cls1 a.link1, .cls1 p a.link1
```

## 5.12. Шаблоны и миксины

*Шаблоны* позволяют создавать определения стилей, которые можно повторно использовать. *Миксины* (*примеси*) очень похожи на шаблоны, они также позволяют вставить одно или несколько определений стиля сколько угодно раз. В отличие от шаблонов, миксины могут содержать параметры, что позволяет настраивать значения атрибутов стиля или генерировать названия атрибутов.

### 5.12.1. Директива `@extend` и шаблонные селекторы

Вместо CSS-селектора можно указать конструкцию `%Название`. Такая конструкция называется *шаблонным селектором* или *селектором-заполнителем*:

```
%red-color {
 color: red;
}
```

Если мы сейчас попробуем выполнить преобразование, то в CSS-файле этот код не найдем вообще. Чтобы вставить шаблон, нужно воспользоваться директивой `@extend %Название`:

```
%red-color {
 color: red;
}
p {
 @extend %red-color;
}
```

Результат в CSS-файле:

```
p {
 color: red;
}
```

Шаблон может содержать множество стилей, и его допускается вставлять в нескольких местах:

```
%red-white {
 color: red;
 background-color: #ffffff;
}
p {
 @extend %red-white;
}
div {
 @extend %red-white;
 border: 1px solid black;
}
```

Результат в CSS-файле:

```
div, p {
 color: red;
 background-color: #ffffff;
}
div {
 border: 1px solid black;
}
```

Шаблонный селектор можно также указать в составе обычных селекторов. В этом случае при вставке родительский селектор заменит шаблонный селектор:

```
#div1 p%red-color {
 color: red;
}
.cls1 {
 @extend %red-color;
}
#id1 {
 @extend %red-color;
}
```

Результат в CSS-файле:

```
#div1 p#id1, #div1 p.cls1 {
 color: red;
}
```

#### **НА ЗАМЕТКУ**

Если название шаблонного селектора начинается с дефиса или подчеркивания, то его область видимости ограничена текущей таблицей стилей. Это правило не относится к подключению модулей с помощью директивы `@import`.

## 5.12.2. Директива `@extend` и простые селекторы

С помощью директивы `@extend` можно вставлять стили не только из блока с шаблонным селектором (см. *разд. 5.12.1*), но и из блока с простым селектором, — таким как `Тег` или `.Класс`. Например, когда один стиливой класс должен содержать все стили другого класса и при этом имеет свои дополнительные стили:

```
.cls1 {
 color: red;
}
.cls1.cls2 {
 background-color: #ffffff;
}
.cls3 {
 @extend .cls1;
 border: 1px solid black;
}
```

Результат в CSS-файле:

```
.cls1, .cls3 {
 color: red;
}
.cls1.cls2, .cls2.cls3 {
 background-color: #ffffff;
}
.cls3 {
 border: 1px solid black;
}
```

Обратите внимание: стиливой класс `cls3` получил все стили класса `cls1`, включая стили из составных селекторов. При этом Sass автоматически создал составной селектор `.cls2.cls3`. Если такие составные селекторы противоречат правилам CSS (например, `#id1#id2`), то они не создаются.

В директиве `@extend` можно указать несколько простых селекторов через запятую, но нельзя использовать составные селекторы:

```
.cls1 {
 color: red;
}
.cls2 {
 background-color: #ffffff;
}
.cls3 {
 @extend .cls1, .cls2;
 border: 1px solid black;
}
```

Результат в CSS-файле:

```
.cls1, .cls3 {
 color: red;
}
```

```
.cls2, .cls3 {
 background-color: #ffffff;
}
.cls3 {
 border: 1px solid black;
}
```

### 5.12.3. Инструкция *!optional*

Если шаблонный селектор или простой селектор, указанные в директиве `@extend`, не найдены, то будет выведено сообщение об ошибке. Если нужно, чтобы такие директивы игнорировались в случае ошибки, то в конце следует добавить инструкцию `!optional`:

```
%red-color {
 color: red;
}
p {
 @extend %red-color !optional; // color: red;
 @extend %blue-color !optional; // Игнорируется
}
```

Если шаблонный селектор найден, то его содержимое блока будет вставлено вместо директивы `@extend`. Если не найден, но он будет проигнорирован.

### 5.12.4. Миксины (примеси)

*Миксины (примеси)* очень похожи на шаблоны, которые мы рассмотрели в предыдущих разделах. Они также позволяют вставить одно или несколько правил стиля вместо директивы `@include` с названием миксина сколько угодно раз. В отличие от шаблонов миксины могут содержать параметры, но, в отличие от функций, они не возвращают значение.

Определение миксина имеет следующий формат:

```
@mixin <Название>([(<Название параметра 1>
 [, ..., <Название параметра N>]))]
{
 <Тело миксина>
}
```

Название миксина должно быть допустимым идентификатором, к которому предъявляются такие же требования, как и к названиям переменных, но без символа `$`. После названия миксина внутри круглых скобок через запятую указываются названия параметров, которые являются локальными переменными. Если миксин не содержит параметров, то можно добавить пустые круглые скобки или вообще их не добавлять.

Внутри фигурных скобок размещаются различные инструкции и определения стилей, которые будут выполняться при каждом вызове миксина. Фигурные скобки указываются в любом случае, даже если тело миксина состоит только из одной инструкции. Точка с запятой после закрывающей фигурной скобки не указывается.

Чтобы вызвать миксин, нужно указать директиву `@include` и название миксина, после которого внутри круглых скобок через запятую можно передать значения. Если миксин не содержит параметров, то указывается только название миксина без круглых скобок или с пустыми круглыми скобками. Пример создания миксина без параметров и его вызов:

```
@mixin red-color {
 color: red;
}
p {
 @include red-color;
 background-color: #ffffff;
}
div {
 @include red-color;
}
```

Результат в CSS-файле:

```
p {
 color: red;
 background-color: #ffffff;
}
div {
 color: red;
}
```

Как видно из примера, вызывать миксин можно сколько угодно раз, что позволяет использовать один и тот же шаблон определений стиля многократно.

Символы дефиса и подчеркивания в названии миксина являются взаимозаменяемыми. Поэтому инструкцию вызова можно записать так:

```
@include red_color;
```

Миксин может содержать один или несколько параметров через запятую. Каждый параметр является локальной переменной. Эта переменная создается при вызове миксина, а после выхода из миксина она удаляется. Таким образом, локальная переменная видна только внутри миксина. Если название локальной переменной совпадает с названием глобальной переменной, то все операции будут производиться с локальной переменной, а значение глобальной не изменится:

```
$color: red; // Глобальная переменная

@mixin test($selector, $color, $bg-color) {
 @debug $color;
 #{$selector} {
 color: $color;
 background-color: $bg-color;
 }
 $color: blue; // Значение глобальной переменной не изменится
}

@include test("p", black, white);
@include test("div", white, black);
@debug $color; // red
```

Результат в CSS-файле:

```
p {
 color: black;
 background-color: white;
}
div {
 color: white;
 background-color: black;
}
```

Переданные при вызове миксина значения присваиваются переменным, расположенным в той же позиции в определении миксина, как и при его вызове.

Обратите внимание: внутри фигурных скобок мы можем указывать не только определения отдельных стилей, но и создавать целые правила. Причем мы генерируем их автоматически, передавая в первом параметре название селектора. Сгенерировать таких правил можно сколько угодно — например, столько, сколько элементов содержит ассоциативный массив. Вызов миксина также может быть расположен как внутри блока, так и вне любых блоков.

Оператор `&` внутри миксина ссылается на родительский селектор блока, внутри которого был выполнен вызов миксина. Поэтому предыдущий пример можно записать следующим образом:

```
@mixin test($color, $bg-color) {
 & {
 color: $color;
 background-color: $bg-color;
 }
}
p { @include test(black, white); }
div { @include test(white, black); }
```

Результат будет таким же.

Внутри определения миксина можно вызывать другие миксины:

```
@mixin text-color($color) {
 color: $color;
}
@mixin bg-color($color) {
 background-color: $color;
}
@mixin colors($color, $bg-color) {
 @include text-color($color);
 @include bg-color($bg-color);
}
div {
 @include colors(black, white);
}
```

Результат в CSS-файле:

```
div {
 color: black;
 background-color: white;
}
```

### 5.12.5. Расположение определений миксинов

Все инструкции в программе выполняются последовательно сверху вниз. Это означает, что прежде чем использовать миксин в программе, его необходимо предварительно определить. Поэтому определение миксина должно быть расположено перед его вызовом. Обратите внимание на то, что размещать определение одного миксина внутри другого нельзя. Если вызов миксина расположен перед его определением или миксина с указанным названием не существует, то мы получим сообщение об ошибке.

Определение миксина можно поместить внутри блока, ограничив область видимости миксина фигурными скобками:

```
div {
 @mixin red-color() {
 color: red;
 }
 @include red-color();
}
```

Результат в CSS-файле:

```
div {
 color: red;
}
```

В больших проектах миксины помещают в отдельный модуль. Например, в библиотеке Bootstrap миксины собраны в модули и помещены в отдельный каталог с названием `mixins`. Все эти модули подключаются в файле `_mixins.scss`, который в свою очередь подключается в файле `bootstrap.scss`.

Давайте создадим файл `_mixins.scss` со следующим содержимым:

```
@mixin _text-color($color) {
 color: $color;
}
@mixin _bg-color($color) {
 background-color: $color;
}
@mixin colors($color, $bg-color) {
 @include _text-color($color);
 @include _bg-color($bg-color);
}
```

Если мы подключим модуль с помощью директивы `@import`, то сможем воспользоваться всеми миксинами:

```
@import "mixins";

div {
 @include colors(black, white);
}
```

```
p {
 @include _bg-color(black);
}
```

Если же подключение выполнить, например, с помощью директивы `@use`, то вызов миксинов с названиями, начинающимися с дефиса или подчеркивания, приведет к ошибке. Частные идентификаторы из модуля не импортируются. Они доступны только внутри модуля, в котором определены. В нашем примере мы можем воспользоваться только миксином `colors()`:

```
@use "mixins";

div {
 @include mixins.colors(black, white);
}
```

### 5.12.6. Способы передачи значений

Как вы уже знаете, после названия миксина внутри круглых скобок через запятую указываются названия параметров. Если миксин не содержит параметров, то можно добавить пустые круглые скобки или вообще их не добавлять. Название параметра является локальной переменной. Эта переменная создается при вызове миксина, а после выхода из него она удаляется. Таким образом, локальная переменная видна только внутри миксина и ее значение между вызовами не сохраняется. Если название локальной переменной совпадает с названием глобальной переменной, то все операции будут производиться с локальной переменной, а значение глобальной не изменится.

Чтобы вызвать миксин, нужно указать директиву `@include` и название миксина, после которого внутри круглых скобок через запятую можно передать значения. Если миксин не содержит параметров, то указывается только название миксина без круглых скобок или с пустыми круглыми скобками. Количество параметров в определении миксина должно совпадать с количеством значений при вызове. Переданные значения присваиваются переменным, расположенным в той же позиции в определении миксина:

```
@mixin colors($color, $bg-color) {
 color: $color;
 background-color: $bg-color;
}

div {
 @include colors(black, white);
}
```

При вызове миксина `colors()` переменной `$color` будет присвоено значение `black`, а переменной `$bg-color` — значение `white`.

Перед значением можно добавить название параметра, указанное в определении миксина, и двоеточие. В этом случае именованные параметры должны следовать

после обычных значений, которые присваиваются переменным, расположенным в той же позиции в определении миксина:

```
div {
 @include colors($color: black, $bg-color: white);
}
```

При использовании именованных параметров их можно указывать в произвольном порядке:

```
div {
 @include colors($bg-color: white, $color: black);
}
```

Чтобы передать значения, расположенные внутри списка, следует после списка или переменной, содержащей список, указать три точки. Количество элементов списка должно совпадать с количеством параметров миксина:

```
$list: white red;
div {
 @include colors($list...);
}
p {
 @include colors((black, white)...);
}
```

### 5.12.7. Необязательные параметры

Чтобы сделать некоторые параметры необязательными, следует в определении миксина присвоить параметру начальное значение. В этом случае если при вызове миксина значение не указано, то переменной будет присвоено это начальное значение. Пример:

```
@mixin colors($color, $bg-color: white) {
 color: $color;
 background-color: $bg-color;
}
div { @include colors(black); }
p { @include colors(white, red); }
```

Результат в CSS-файле:

```
div {
 color: black;
 background-color: white;
}
p {
 color: white;
 background-color: red;
}
```

Следует заметить, что необязательные параметры должны следовать после обязательных, чтобы можно было использовать позиционное присваивание. В противном случае необходимо указывать название параметра.

### 5.12.8. Передача произвольного количества значений

Количество значений, переданных миксину, может быть произвольным. Для этого в определении миксина после названия параметра следует указать три точки. Параметр с тремя точками должен быть расположен последним в списке параметров:

```
@mixin pad($values...) {
 padding: join($values, (), space);
}
div { @include pad(10px, 15px); }
p { @include pad(5px, 10px, 15px, 20px); }
```

Результат в CSS-файле:

```
div {
 padding: 10px 15px;
}
p {
 padding: 5px 10px 15px 20px;
}
```

Если в определении миксина существует параметр с тремя точками, то при вызове миксина можно передать любое количество именованных параметров со значениями. Получить эти параметры позволяют функции `meta.keywords($args)` и `keywords($args)`, которые возвращают ассоциативный массив, ключами которого будут названия параметров без символа `$`:

```
@use "sass:meta";

@mixin test($args...) {
 @each $key, $value in meta.keywords($args) {
 @debug "#{$key} => #{$value}";
 }
}
@include test($x: 10, $y: 20, $z: 30);
```

Результат в окне консоли:

```
scss\main.scss:5 Debug: x => 10
scss\main.scss:5 Debug: y => 20
scss\main.scss:5 Debug: z => 30
```

### 5.12.9. Директива `@content` и блоки содержимого

Миксин может принимать целый блок с определениями стилей, который называется *блоком содержимого*. В этом случае при вызове миксина в конце указываются фигурные скобки, внутри которых передаются определения стилей. Место вставки блока содержимого помечается с помощью директивы `@content` внутри определения миксина:

```
@mixin test($color, $bg-color) {
 color: $color;
 background-color: $bg-color;
 @content;
}
```

```

}
div {
 @include test(black, white) {
 border: 1px solid red;
 padding: 15px;
 }
}

```

Результат в CSS-файле:

```

div {
 color: black;
 background-color: white;
 border: 1px solid red;
 padding: 15px;
}

```

Директив `@content` внутри определения миксина может быть несколько. Блок содержимого будет подставлен вместо каждой директивы.

Внутри блока содержимого мы не имеем доступа к переменным, объявленным внутри миксина или в параметрах. Если нужно передать значения в блок содержимого, то следует указать их через запятую внутри круглых скобок после директивы `@content`:

```
@content(<Значения через запятую>);
```

Дополнительно нужно описать параметры блока содержимого внутри круглых скобок после ключевого слова `using`:

```

@mixin test($color, $bg-color) {
 color: $color;
 background-color: $bg-color;
 @content($color);
}
div {
 @include test(black, white) using ($c) {
 border: 1px solid $c;
 }
}

```

Результат в CSS-файле:

```

div {
 color: black;
 background-color: white;
 border: 1px solid black;
}

```

Функции `meta.content-exists()` и `content-exists()` возвращают значение `true`, если при вызове миксина был передан блок содержимого, и `false` — в противном случае:

```
@use "sass:meta";
```

```

@mixin test() {
 div {
 @debug meta.content-exists();
 }
}

```

```

 @debug content-exists();
 @content;
 }
}
@include test; // false
@include test() { // true
 border: 1px solid red;
}

```

### 5.12.10. Проверка существования миксина

Проверить существование миксина позволяют функции `meta.mixin-exists($name[, $module: null])` и `mixin-exists($name[, $module: null])`. В параметре `$name` название миксина указывается в виде строки. Если миксин с названием `$name` определен, то возвращается значение `true`, в противном случае — `false`. В параметре `$module` можно дополнительно указать пространство имен. Пример:

```

@use "sass:meta";
@use "mixins";

@mixin test {
 border: 1px solid red;
}

@debug meta.mixin-exists("test"); // true
@debug mixin-exists("test2"); // false
@debug mixin-exists("colors", mixins); // true
@debug mixin-exists("test", mixins); // false
@debug mixin-exists("load-css", meta); // true

```

## 5.13. Отличия SASS-файлов от SCSS-файлов

Sass содержит два синтаксиса: `scss` (Sassy CSS) и `sass` (Syntactically Awesome Stylesheets). Синтаксис SCSS-файлов мы уже изучили ранее в этой главе. Теперь рассмотрим основные отличия SASS-файлов от SCSS-файлов:

- ◆ файлы имеют расширение `sass`;
- ◆ точка с запятой в конце инструкций не добавляется;
- ◆ фигурные скобки не указываются;
- ◆ перед инструкциями внутри блоков нужно вставлять одинаковый отступ, как в языке программирования Python;
- ◆ отличается формат объявления и вызова миксинов.

Давайте создадим файл `C:\book\p1\scss\style.sass` со следующим содержимым:

```

$color: #000 !default
$bg-color: #fff !default

div
 color: $color

```

```

background-color: $bg-color
margin:
 left: 5px
 right: 10px
 top: 15px
 bottom: 20px

a
 text-decoration: none
 &:hover
 text-decoration: underline

```

```

p
 color: $color
 background-color: $bg-color

```

Теперь создадим CSS-файл, выполнив следующую команду:

```
C:\book\p1>sass scss/style.sass dist/css/style.css
```

Sass определяет синтаксис по расширению файла. Если нет расширения файла, то следует в составе команды указать флаг `--indented`:

```
sass --indented scss/style.sass dist/css/style.css
```

Результат в CSS-файле:

```

div {
 color: #000;
 background-color: #fff;
 margin-left: 5px;
 margin-right: 10px;
 margin-top: 15px;
 margin-bottom: 20px;
}
div a {
 text-decoration: none;
}
div a:hover {
 text-decoration: underline;
}
p {
 color: #000;
 background-color: #fff;
}
/*# sourceMappingURL=style.css.map */

```

При объявлении миксина вместо директивы `@mixin` нужно добавлять символ `=`, а при вызове миксина вместо директивы `@include` следует использовать символ `+`:

```

=colors($color, $bg-color)
 color: $color
 background-color: $bg-color

```

```

p
 +colors(#000, #fff)
 padding-left: 15px

```

Результат в CSS-файле:

```
p {
 color: #000;
 background-color: #fff;
 padding-left: 15px;
}
```

Синтаксис `scss` более близок к обычным CSS-файлам, чем синтаксис `sass`. Можно просто переименовать CSS-файл в `SCSS`-файл, и он будет работать, как ни в чем не бывало. Поэтому в этой книге мы используем его по умолчанию. Если синтаксис `sass` нравится вам больше, то можете пользоваться им, но помните, что разработчики библиотеки `Bootstrap` используют синтаксис `scss`.

## 5.14. Программа `node-sass` (LibSass)

Существует несколько реализаций `Sass`: `Dart Sass`, `LibSass` и `Ruby Sass`. `Dart Sass` является основной реализацией `Sass`. В этой главе мы изучали привязку `Dart Sass` к `Node.js`. Изначально `Sass` был написан на языке `Ruby`, но позднее он был переписан на `C/C++` и получил название `LibSass`. В свою очередь существуют привязки `LibSass` для различных языков — например, для `Node.js` привязка называется `node-sass`. Именно эту привязку разработчики библиотеки `Bootstrap` используют для компиляции `SCSS`-файлов, а не `Dart Sass`. Хотя компиляция с помощью `Dart Sass` также возможна.

### 5.14.1. Установка `node-sass`

Давайте рассмотрим основные отличия `node-sass` от `Dart Sass`. Сначала установим `node-sass`, но не глобально, как мы это делали раньше, а как зависимость на этапе разработки для нашего пакета `C:\book\p1`:

```
C:\book\p1>npm install --save-dev node-sass
```

После установки в каталоге `C:\book\p1\node_modules` будет создан каталог `node-sass` с файлами, необходимыми для компиляции. Дополнительно будут установлены все зависимости, а в файл `package.json` за счет наличия флага `--save-dev` добавлен раздел `devDependencies`:

```
"devDependencies": {
 "node-sass": "^4.14.1"
}
```

Благодаря разделам `dependencies` (этап работы) и `devDependencies` (этап разработки) все пользователи нашего пакета знают о зависимостях и версиях необходимых сторонних пакетов. В отличие от глобальной установки, где о зависимостях знаем только мы.

## 5.14.2. Создание CSS-файла из SCSS-файла

Если сейчас в командной строке попробовать вывести версию `node-sass`, то ничего не получится:

```
C:\book\p1>node-sass --version
```

**"node-sass" не является внутренней или внешней командой, исполняемой программой или пакетным файлом.**

При глобальной установке мы бы получили версию, но мы установили `node-sass`, как зависимость для пакета. В нашем случае мы имеем программу `node-sass` на языке JavaScript, расположенную в каталоге `C:\book\p1\node_modules\node-sass\bin`. Чтобы запустить ее на выполнение, нужно указать программу `node`, а затем передать ей путь к программе `node-sass`:

```
C:\book\p1>node C:\book\p1\node_modules\node-sass\bin\node-sass --version
node-sass 4.14.1 (Wrapper) [JavaScript]
libsass 3.5.5 (Sass Compiler) [C/C++]
```

Получается слишком длинная команда. Чтобы было удобно запускать процесс преобразования, добавим несколько скриптов в раздел `scripts` файла `package.json`:

```
"scripts": {
 "createCSS": "sass scss/main.scss dist/css/main.css",
 "node-sass-version": "node-sass --version",
 "node-sass": "node-sass --output-style compressed scss/main.scss
 dist/css/main.min.css",
 "node-sass-dev": "node-sass --output-style expanded --source-map true
 -o dist/css/ scss/main.scss"
}
```

Скриптом `createCSS` мы уже пользовались ранее. Чтобы вывести версию `node-sass`, выполняем скрипт `node-sass-version`:

```
C:\book\p1>npm run node-sass-version
```

```
> pl@1.0.0 node-sass-version C:\book\p1
> node-sass --version
```

```
node-sass 4.14.1 (Wrapper) [JavaScript]
libsass 3.5.5 (Sass Compiler) [C/C++]
```

Скрипт `node-sass` позволяет преобразовать файл `main.scss` в сжатый файл `main.min.css` без создания дополнительного файла с расширением `css.map`. Давайте добавим в файл `main.scss` содержимое листинга 5.2 и выполним скрипт:

```
C:\book\p1>npm run node-sass
```

```
> pl@1.0.0 node-sass C:\book\p1
> node-sass --output-style compressed scss/main.scss
dist/css/main.min.css
```

```
Rendering Complete, saving .css file...
Wrote CSS to C:\book\p1\dist\css\main.min.css
```

```
C:\book\p1>
```

Скрипт `node-sass-dev` позволяет преобразовать файл `main.scss` в файл `main.css` без сжатия, с дополнительным созданием файла с расширением `css.map`:

```
C:\book\p1>npm run node-sass-dev

> p1@1.0.0 node-sass-dev C:\book\p1
> node-sass --output-style expanded --source-map true -o dist/css/
scss/main.scss

Rendering Complete, saving .css file...
Wrote CSS to C:\book\p1\dist\css\main.css
Wrote Source Map to C:\book\p1\dist\css\main.css.map
```

```
C:\book\p1>
```

Как видно из примеров, синтаксис команд `node-sass` отличается от синтаксиса `Dart Sass`. Команда в `node-sass` имеет следующий формат:

```
node-sass [<Опции>] <SCSS-файл или каталог>[<CSS-файл>]
```

Если указать только название `SCSS`-файла, то результат будет выведен в окно консоли в стиле `nested`:

```
> node-sass scss/main.scss
```

```
body {
 color: #ffffff;
 background-color: #000000; }
```

Если указан параметр `<CSS-файл>`, то результат будет записан в файл:

```
node-sass scss/main.scss dist/css/main.css
```

С помощью флага `--output` (или `-o`) можно указать путь до каталога, в котором будут сохраняться `CSS`-файлы. Названия `CSS`-файлов будут совпадать с названиями файлов, содержащими исходный код. Скомпилируем все файлы из каталога `C:\book\p1\scss`:

```
> node-sass --output dist/css/ scss/

Rendering Complete, saving .css file...
Wrote CSS to C:\book\p1\dist\css\main.css
Rendering Complete, saving .css file...
Wrote CSS to C:\book\p1\dist\css\style.css
Wrote 2 CSS files to C:\book\p1\dist\css\
```

В результате были скомпилированы все файлы с расширениями `scss` и `sass`, названия которых не содержат символ подчеркивания в начале.

В параметре `<Опции>` можно указать следующие основные флаги:

- ◆ `--output <Путь>` (или `-o <Путь>`) — путь до каталога, в котором будут сохраняться `CSS`-файлы;
- ◆ `--output-style <Стиль>` — задает стиль итогового `CSS`-файла:
  - `nested` — вложенный стиль (с помощью отступов показывает вложенность правил — значение по умолчанию);

- `expanded` — развернутый стиль (файл `bootstrap.css` имеет именно этот стиль);
- `compact` — компактный стиль (все определения стилей внутри блока на одной строке);
- `compressed` — сжатый стиль (все стили на одной строке без пробелов. Файл `bootstrap.min.css` имеет именно этот стиль).

Пример указания сжатого стиля:

```
node-sass --output-style compressed --output dist/css/ scss/
```

- ◆ `--include-path <Путь>` — задает путь поиска модулей:

```
node-sass -o dist/css/ --include-path node_modules/
scss/main.scss
```

- ◆ `--source-map true` — если флаг указан со значением `true`, то рядом с CSS-файлом будет создан файл с расширением `css.map` (по умолчанию файл не создается):

```
node-sass -o dist/css/ --source-map true scss/main.scss
```

Вместо значения `true` можно передать путь к каталогу, в котором будут сохраняться файлы с расширением `css.map`:

```
node-sass -o dist/css/ --source-map dist/css/ scss/main.scss
```

- ◆ `--source-map-contents true` — если флаг указан со значением `true`, то внутри файла с расширением `css.map` будет вставлено содержимое файла с исходным кодом:

```
node-sass -o dist/css/ --source-map true
--source-map-contents true scss/main.scss
```

- ◆ `--source-map-embed true` — если флаг указан со значением `true`, то внутри CSS-файла будет вставлено закодированное содержимое файла `css.map`:

```
node-sass -o dist/css/ --source-map-embed true scss/main.scss
```

- ◆ `--source-map-root <Путь>` — задает значение для свойства `sourceRoot` внутри файла с расширением `css.map`;

- ◆ `--omit-source-map-url true` (или `-x true`) — если флаг указан со значением `true`, то внутри CSS-файла не будет вставляться комментарий:

```
/*# sourceMappingURL=<Название>.css.map */
```

Пример указания флага:

```
node-sass -o dist/css/ --source-map true
--omit-source-map-url true scss/main.scss
```

- ◆ `--source-comments true` — если флаг указан со значением `true`, то внутри CSS-файла будут вставлены комментарии с номером строки и названием файла с исходным кодом:

```
/* line 4, scss/main.scss */
body {
 color: #ffffff;
 background-color: #000000; }
```

- ◆ `--precision <Число>` — задает максимальное количество цифр после точки для вещественных чисел в результате вычисления. Значение по умолчанию: 5. Укажем шесть цифр:

```
node-sass -o dist/css/ --precision 6 scss/main.scss
```

- ◆ `--indent-type <space | tab>` — позволяет указать символ, который будет использоваться внутри CSS-файла для добавления отступа. После флага можно передать значения `space` (пробел; значение по умолчанию) или `tab` (табуляция):

```
node-sass -o dist/css/ --indent-type tab scss/main.scss
```

- ◆ `--indent-width <Число>` — задает количество символов для добавления отступа внутри CSS-файла. После флага можно передать число до 10. Значение по умолчанию: 2. Укажем три пробела:

```
node-sass -o dist/css/ --indent-type space --indent-width 3
scss/main.scss
```

- ◆ `--linefeed <cr | crlf | lf | lfcr>` — задает комбинацию для перевода строки внутри CSS-файла. Значение по умолчанию: `lf (\n)`. Укажем комбинацию `\r\n`:

```
node-sass -o dist/css/ --linefeed crlf scss/main.scss
```

- ◆ `--quiet true` (или `-q true`) — отключает вывод информации о процессе преобразования в окне консоли, кроме вывода сообщений об ошибках. По умолчанию вывод выглядит так:

```
> node-sass -o dist/css/ --quiet false scss/main.scss
```

```
Rendering Complete, saving .css file...
```

```
Wrote CSS to C:\book\pl\dist\css\main.css
```

```
C:\book\pl>
```

Если мы укажем флаг со значением `true`, то получим такой результат:

```
> node-sass -o dist/css/ --quiet true scss/main.scss
```

```
C:\book\pl>
```

- ◆ `--indented-syntax true` (или `-i true`) — задает синтаксис SASS-файлов при чтении из `stdin`. При указании файла синтаксис задает расширение файла;

- ◆ `--recursive` (или `-r`) — со значением `true` задает рекурсивный просмотр всех вложенных каталогов. Это значение по умолчанию, поэтому флаг можно не добавлять. Если нужно просматривать только указанный каталог, то следует добавить флаг со значением `false`:

```
node-sass -r false -o dist/css/ scss/
```

После выполнения этой команды файлы с исходным кодом, расположенные во вложенных каталогах, преобразованы не будут.

Для получения полного списка опций укажите флаг `--help`:

```
node-sass --help
```

### 5.14.3. Отслеживание изменений SCSS-файлов

Чтобы изменения файлов с исходным кодом отслеживались и автоматически производилось преобразование в CSS-файлы, следует в составе команды указать флаг `-watch` (или `-w`). Добавим еще один скрипт в файл `package.json`:

```
"node-sass-watch": "node-sass --watch --output-style expanded
--source-map true -o dist/css/ scss/"
```

В этой команде мы отслеживаем все изменения файлов с исходным кодом внутри каталога `C:\book\p1\scss` и во всех вложенных каталогах. Результат преобразования записывается в каталог `C:\book\p1\dist\css` в развернутом стиле. Дополнительно создается файл с расширением `css.map`. Запустим команду:

```
C:\book\p1>npm run node-sass-watch
```

```
> p1@1.0.0 node-sass-watch C:\book\p1
> node-sass --watch --output-style expanded --source-map true
-o dist/css/ scss/
```

После запуска команды программа будет выполняться в бесконечном цикле, блокируя ввод других команд. Чтобы вводить другие команды, нужно запустить дополнительное окно командной строки. Сразу после сохранения файла с исходным кодом будет автоматически запущен процесс преобразования, и в командной строке появятся следующие сообщения:

```
=> changed: C:\book\p1\scss\main.scss
Rendering Complete, saving .css file...
Wrote Source Map to C:\book\p1\dist\css\main.css.map
Wrote CSS to C:\book\p1\dist\css\main.css
```

Если в файле содержится ошибка, то будет выведено сообщение о проблеме.

Чтобы остановить отслеживание, нужно нажать комбинацию клавиш `<Ctrl>+<C>` или закрыть окно консоли. В первом случае программа запросит подтверждение действия путем ввода буквы `y` и нажатия клавиши `<Enter>`:

```
Завершить выполнение пакетного файла [Y(да)/N(нет)]? y
```

```
C:\book\p1>
```

### 5.14.4. Различия между node-sass и Sass

Программа `node-sass` является привязкой `LibSass` к `Node.js`. `LibSass` написана на низкоуровневых языках `C/C++`, поэтому скорость преобразования файлов обычно выше, чем у программы `Dart Sass`, написанной, как видно из ее названия, на языке `Dart`. Программа на языке `Dart` может быть автоматически преобразована в программу на языке `JavaScript`. Программа `Sass`, которую мы установили в *разд. 5.1.3*, как раз и есть результат такого автоматического преобразования. Программа на `JavaScript` выполняется обычно гораздо медленнее, чем программа на языке `C++`. Таким образом, по скорости преобразования файлов предпочтение стоит отдать `node-sass`.

Однако существует обратная сторона медали. Программу на низкоуровневом C++ писать гораздо сложнее, чем на языке Dart. По этой причине Dart Sass развивается быстрее, чем LibSass, и содержит большее число возможностей. Давайте рассмотрим основные различия между node-sass и Sass:

- ◆ синтаксис запуска преобразования в командной строке разный, поэтому мы отдельно рассмотрели основные опции в *разд. 5.1.5* и *5.14.2*;
- ◆ в node-sass отсутствует интерактивный режим (см. *разд. 5.1.7*);
- ◆ при выводе русских букв с помощью директив @debug и @warn мы получим искаженные символы, т. к. символы выводятся в кодировке UTF-8, а консоль по умолчанию работает с кодировкой windows-866;
- ◆ в node-sass отсутствуют встроенные модули, поэтому нельзя воспользоваться функциями с предварительным указанием пространства имен, но можно пользоваться глобальными функциями. Например, вместо функции meta.variable-exists() следует пользоваться глобальной функцией variable-exists();
- ◆ в node-sass импорт модулей производится только с помощью директивы @import. Директивы @use и @forward не поддерживаются;

- ◆ в node-sass пути поиска модулей задаются с помощью флага --include-path <Путь>:

```
node-sass -o dist/css/ --include-path node_modules/
scss/main.scss
```

- ◆ в node-sass индексные файлы (см. *разд. 5.10.4*) внутри каталога не работают;
- ◆ в node-sass деление на ноль приводит к значению Infinity (бесконечность);
- ◆ в node-sass сравнение числа, содержащего единицу измерения, с таким же числом без единицы измерения возвращает значение true:

```
@debug 1px == 1; // true
@debug 100% == 100; // true
```

В Sass возвращается значение false:

```
@debug 1px == 1; // false
@debug 100% == 100; // false
```

- ◆ в node-sass функции max() и min() возвращают значения, а не CSS-функции:

```
@debug max(10px, 3px); // 10px
@debug min(10px, 3px); // 3px
```

- ◆ в node-sass дополнительные форматы функций rgb(), rgba(), hsl() и hsla() не поддерживаются;
- ◆ в node-sass директива @at-root совместно с ключевым словом with работает не так, как в Sass:

```
@media print {
 div {
 background-color: #ffffff;
 @at-root (with: rule) {
 span { color: red; }
 }
 }
}
```

```

 }
 }
}

```

Результат в node-sass:

```

@media print {
 div {
 background-color: #ffffff;
 }
}
span {
 color: red;
}

```

**Результат в Sass:**

```

@media print {
 div {
 background-color: #ffffff;
 }
}
div span {
 color: red;
}

```

- ◆ в node-sass после директивы @content нельзя указать параметры. Ключевое слово using также не поддерживается;
- ◆ в node-sass функцию content-exists() нельзя вставлять внутрь вложенного в миксин блока. Этот код приведет к ошибке:

```

@mixin test() {
 div {
 @debug content-exists(); // Ошибка!
 @content;
 }
}

```

А этот код будет успешно преобразован:

```

@mixin test() {
 @debug content-exists(); // OK
 div {
 @content;
 }
}

```

Как видите, различий много. Учитывайте их, если хотите, чтобы исходный код компилировался и в Sass, и в node-sass.

## 5.15. Сборка SCSS-файлов библиотеки Bootstrap под свой проект

Итак, изучение Sass закончено, и мы можем рассмотреть способы изменения значений отдельных атрибутов, а также выполнить сборку SCSS-файлов библиотеки Bootstrap под свой проект.

Если нужно изменить значения отдельных атрибутов стиля, то достаточно после подключения файла `bootstrap.min.css` добавить тег `<style>` и внутри него переопределять значения, либо подключить свой CSS-файл. Учитывайте, что после значений некоторых атрибутов расположена инструкция `!important`, которая может не дать переопределить значение.

Теперь, когда вы уже знаете Sass, лучше воспользоваться возможностями CSS-препроцессора. В *разд. 5.10.9* мы рассматривали возможность переопределения значений переменных из модуля. Если внутри модуля при объявлении переменной указана инструкция `!default`, то можно будет изменить значение переменной при подключении модуля, не изменяя исходный код модуля. Сначала нужно объявить одноименную переменную с новым значением, а затем подключить модуль с помощью директивы `@import`. Таким способом можно изменять значения отдельных переменных, создавая сборку библиотеки Bootstrap под свои потребности.

В том же разделе я просил вас не торопиться использовать скомпилированный файл в реальном проекте, т. к. после компиляции необходимо выполнить некоторые дополнительные действия. Что же нужно дополнительно сделать? Во-первых, мы собирали библиотеку Bootstrap с помощью Sass, а разработчики используют `node-sass`. Различий здесь не так много, но все-таки они есть. Во-вторых, после сборки разработчики библиотеки выполняют дополнительную обработку с помощью пакета `PostCSS` — например, добавляют вендорные префиксы и сжимают файл. Рассмотрим процесс сборки подробно.

Давайте создадим следующую структуру каталогов и файлов:

```
C:\book\bs4\
 build\
 postcss.config.js
 dist\
 css\
 scss\
 mybootstrap.scss
 .browserslistrc
 package.json
```

В *разд. 1.1.1* нужно было скачать архив с исходными кодами библиотеки Bootstrap 4. Сейчас нам этот архив понадобится. Копируем из архива каталог `build` со всем содержимым (или только с файлом `postcss.config.js`), а также файл `.browserslistrc`. Создаем каталоги `dist/css` и `scss`, как и в пакете `p1`, который мы использовали в примерах этой главы. Содержимое файла `package.json` приведено в листинге 5.3.

**Листинг 5.3. Содержимое файла `package.json`**

```
{
 "name": "bs4",
 "version": "1.0.0",
 "description": "",
```

```

"main": "index.js",
"scripts": {
 "css": "npm-run-all sass css-prefix css-minify",
 "sass": "node-sass --output-style expanded --source-map true
--source-map-contents true --precision 6 --include-path node_modules/
-o dist/css/ scss/",
 "css-prefix": "postcss --config build/postcss.config.js
--replace \"dist/css/*.css\" \"!dist/css/*.min.css\"",
 "css-minify": "cleancss --level 1 --format breakWith=lf --source-map
--source-map-inline-sources --output dist/css/mybootstrap.min.css
dist/css/mybootstrap.css"
},
"keywords": [],
"author": "",
"license": "MIT",
"dependencies": {
 "bootstrap": "^4.5.2"
},
"devDependencies": {
 "node-sass": "^4.14.1",
 "npm-run-all": "^4.1.5",
 "autoprefixer": "^9.7.6",
 "clean-css-cli": "^4.3.0",
 "postcss-cli": "^7.1.1"
}
}

```

**ВНИМАНИЕ!**

Команды в скриптах указывайте без символов переноса строки.

В командной строке переходим в каталог `C:\book\bs4` и устанавливаем все зависимости, указанные в файле `package.json`, выполняя команду `npm install`:

```
C:\book\bs4>npm install
```

Обратите внимание: всего лишь одной командой мы установили в каталог `C:\book\bs4\node_modules` все необходимые пакеты, а также все зависимости этих пакетов. В числе этих пакетов была установлена библиотека Bootstrap и CSS-препроцессор `node-sass`. Переходим в каталог `C:\book\bs4\node_modules\bootstrap\scss` и копируем файл `bootstrap.scss`. Размещаем эту копию в каталоге `C:\book\bs4\scss` и переименовываем файл в `mybootstrap.scss`.

Внутри файла `mybootstrap.scss` подключаются все остальные модули библиотеки Bootstrap, но мы изменили местонахождение файла, а значит нужно изменить и пути к модулям. Во всех директивах `@import` перед названиями модулей добавляем фрагмент `bootstrap/scss/`:

```
/*!
```

```

* Bootstrap v4.5.2 (https://getbootstrap.com/)
* Copyright 2011-2020 The Bootstrap Authors
* Copyright 2011-2020 Twitter, Inc.

```

```
* Licensed under MIT
* (https://github.com/twbs/bootstrap/blob/master/LICENSE)
*/
```

```
// Сюда вставляем переменные для переопределения значений
```

```
// Обязательные модули
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "bootstrap/scss/mixins";
```

```
// Другие модули
// Наш модуль
```

В команде компиляции (см. скрипт `sass`) мы указали путь поиска модулей:

```
--include-path node_modules/
```

Если путь поиска модулей не указан, то нужно прописывать относительные пути:

```
@import "../node_modules/bootstrap/scss/functions";
```

Итак, внутри файла `mybootstrap.scss` подключаются все модули библиотеки Bootstrap. Модули `functions`, `variables` и `mixins` являются обязательными, т. к. в них расположены функции, переменные и миксины, используемые в других модулях. Остальные модули отвечают за какое-либо направление — например, каждый компонент имеет свой собственный модуль. Если мы не используем компонент, то можно просто закомментировать инструкцию импорта:

```
//@import "bootstrap/scss/carousel";
```

После сборки код модуля не попадет в итоговый CSS-файл, что позволит уменьшить размер подключаемого файла. Таким образом, с помощью добавления символов комментария перед директивами `@import`, можно создать конфигурацию библиотеки под свои потребности.

Перед инструкциями импорта можно переопределять значения переменных из библиотеки Bootstrap или добавлять свои значения в ассоциативные массивы. Давайте изменим значение переменной `$secondary` из библиотеки Bootstrap и добавим тему `purple`:

```
// Сюда вставляем переменные для переопределения значений
$secondary: #e83e8c;
$theme-colors: ("purple": #6f42c1);
```

Для компиляции следует пользоваться скриптами из файла `package.json`. Чтобы выполнить только сборку, запускаем скрипт `sass`:

```
C:\book\bs4>npm run sass
```

```
> bs4@1.0.0 sass C:\book\bs4
> node-sass --output-style expanded --source-map true
--source-map-contents true
--precision 6 --include-path node_modules/ -o dist/css/ scss/
```

```
Rendering Complete, saving .css file...
Wrote Source Map to C:\book\bs4\dist\css\mybootstrap.css.map
Wrote CSS to C:\book\bs4\dist\css\mybootstrap.css
Wrote 1 CSS files to C:\book\bs4\dist\css\
```

```
C:\book\bs4>
```

Что же мы изменили? Во-первых, переопределив значение переменной `$secondary`, мы изменили цвет во всех темах с участием этой переменной:

```
<button class="btn btn-secondary mb-3">Кнопка</button>
<div class="alert alert-secondary">.alert-secondary</div>
```

Во-вторых, добавив новый элемент в ассоциативный массив `$theme-colors`, мы создали новую тему, которая стала доступной везде, где используются цветовые темы:

```
<button class="btn btn-purple mb-3">Кнопка</button>
<div class="alert alert-purple">.alert-purple</div>
```

Посмотрите — мы добавили всего один элемент в ассоциативный массив, а получили целую коллекцию новых стилевых классов: `btn-purple`, `btn-outline-purple`, `table-purple`, `badge-purple`, `alert-purple`, `list-group-item-purple`, `bg-purple`, `text-purple` и `border-purple`. Вот оно, преимущество использования CSS-препроцессора Sass.

Скрипт `css-prefix` следует запускать после скрипта `sass`. Он с помощью пакета `Autoprefixer` добавляет вендорные префиксы в зависимости от выбранных браузеров из файла конфигурации `.browserslistrc`:

```
C:\book\bs4>npm run css-prefix
```

В результате помимо атрибута `display` со значением `flex`:

```
display: flex;
```

будет добавлен атрибут `display` со значением `-ms-flexbox`:

```
display: -ms-flexbox;
```

```
display: flex;
```

На последнем этапе следует вызвать скрипт `css-minify`:

```
C:\book\bs4>npm run css-minify
```

В результате будет создан сжатый файл `mybootstrap.min.css`, который мы можем использовать в своих проектах.

Скрипт `css` позволяет выполнить все эти скрипты последовательно:

```
C:\book\bs4>npm run css
```

Благодаря пакету `npm-run-all` можно с помощью одной команды запускать сразу несколько скриптов, причем как последовательно, так и параллельно. В нашем случае возможно только последовательное выполнение.

Вот теперь вы можете смело использовать скомпилированный файл `mybootstrap.min.css` в реальных проектах.



## Заключение

Вот и закончилось наше путешествие в мир Bootstrap и Sass. Материал книги описывает лишь основы этих замечательных технологий. А здесь мы уточним, где найти дополнительную информацию.

Самым важным источником информации по Bootstrap 4 является официальный сайт библиотеки: <https://getbootstrap.com/>. На этом сайте вы найдете новости, документацию, а также ссылки на все другие ресурсы в Интернете, посвященные Bootstrap. Не следует также забывать о существовании сайта <https://v5.getbootstrap.com/>, на котором доступна для загрузки самая последняя версия библиотеки Bootstrap 5.

На странице <https://getbootstrap.com/docs/4.5/examples/>, а также в каталоге `bootstrap\site\docs\4.5\examples` архива с исходными кодами, расположены примеры использования стилей и компонентов из библиотеки Bootstrap. В состав примеров также входят некоторые экспериментальные стили. Обязательно изучите эти примеры.

На официальном сайте Sass: <https://sass-lang.com/> можно найти документацию, которая обновляется в режиме реального времени, а также ссылки на все другие ресурсы в Интернете, посвященные Sass. Не следует забывать, что исходные коды библиотеки Bootstrap являются великолепным примером использования Sass. Обязательно изучите содержимое каталога `bootstrap\scss` из архива с исходными кодами.

На странице [https://ru.wikipedia.org/wiki/Bootstrap\\_\(фреймворк\)](https://ru.wikipedia.org/wiki/Bootstrap_(фреймворк)) вы найдете историю развития библиотеки Bootstrap, а также действующие ссылки на все другие ресурсы в Интернете, посвященные этой библиотеке. Аналогичная информация по Sass доступна на странице <https://ru.wikipedia.org/wiki/Sass>.

Если в процессе изучения у вас возникнут какие-либо недопонимания, то не следует забывать, что Интернет предоставляет множество ответов на самые разнообразные вопросы. Достаточно в строке запроса поискового портала (например, <https://www.google.com/>) набрать свой вопрос. Наверняка уже кто-то сталкивался с подобной проблемой и описал ее решение на каком-либо сайте.

Свои замечания и пожелания вы можете оставить на странице книги на сайте издательства «БХВ»: <https://bhv.ru/>. Все замеченные опечатки и неточности прошу присылать на e-mail: [mail@bhv.ru](mailto:mail@bhv.ru) — не забудьте только указать название книги и имя автора.



# ПРИЛОЖЕНИЕ

## Описание электронного архива

По ссылке <ftp://ftp.bhv.ru/9785977567695.zip> можно скачать электронный архив с исходными кодами примеров к книге. Ссылка доступна также со страницы книги на сайте <https://bhv.ru/>.

Структура архива представлена в табл. П1.

*Таблица П1. Структура электронного архива*

Файл	Описание
Listings.doc	Содержит все пронумерованные листинги из книги, а также некоторые полезные фрагменты кода
Readme.txt	Описание электронного архива

# Предметный указатель

## !

!default 362, 435, 436, 438, 440, 478  
!global 362  
!important 478  
!optional 460

## #

#AARRGGBB 420  
#RGB 407  
#RRGGBB 407  
#RRGGBBAA 407

## \$

\$() 185  
\$e 381  
\$pi 381

## @

@at-root 448, 449, 450, 476  
@content 466, 467, 477  
@debug 359, 476  
@each 378, 379, 390, 399  
@else 375  
@error 359  
@extend 456, 457, 459  
@for 376, 391  
@forward 432, 437, 438, 439, 476  
@function 421  
@if 375, 454  
@import 429, 430, 432, 435, 436, 441, 463, 476, 478, 479  
@include 440, 460, 461, 464, 469  
@media 27, 450, 451  
@mixin 460, 469  
@return 421, 422, 423  
@supports 450

@use 432, 433, 434, 435, 436, 464, 476

@warn 359, 476

@while 377

## <

<!doctype> 18, 82  
<a> 48  
<abbr> 44  
<b> 37  
<blockquote> 45  
<body> 19  
<br> 40  
<button> 30, 131, 132  
<caption> 71, 75  
<cite> 45  
<code> 43  
<datalist> 144  
<dd> 70  
<del> 39  
<details> 47  
<dl> 70  
<dt> 70  
<em> 34  
<fieldset> 179, 180  
<figcaption> 79  
<figure> 79  
<form> 131, 161  
<h1> 46  
<h2> 46  
<h3> 46  
<h4> 46  
<h5> 46  
<h6> 46  
<head> 19  
<hr> 47  
<html> 18  
<i> 34  
<iframe> 82  
<img> 76, 77, 78, 79  
<input> 131, 135, 152, 159, 161, 167, 168  
<ins> 39  
<kbd> 43  
<label> 139, 141, 143, 149, 151  
<legend> 179  
<li> 67  
<mark> 44  
<meter> 199  
<ol> 67  
<optgroup> 145  
<option> 144  
<p> 46  
<picture> 77  
<pre> 41, 42, 43  
<progress> 198, 199  
<q> 45  
<s> 39  
<samp> 43  
<select> 131, 144  
<small> 34, 46, 139  
<source> 77  
<strong> 37  
<sub> 41  
<summary> 47  
<sup> 41  
<svg> 78, 79  
<table> 71  
<tbody> 71  
<td> 71  
<textarea> 131, 136  
<tfoot> 71  
<th> 71  
<thead> 71  
<tr> 71  
<u> 39  
<ul> 67  
<var> 44

## A

abs() 381  
absolute 355  
accept 161

accordion 243  
acos() 384  
active 132, 183, 184, 186, 187,  
208, 252, 253, 254, 256, 257,  
259, 269, 271, 273, 274, 288,  
298  
adjust() 413  
adjust-color() 413, 414, 415,  
416, 417, 418  
adjust-hue() 414  
Alert 218, 219, 220, 221  
alert() 220  
alert-danger 219  
alert-dark 218  
alert-dismissible 219  
alert-heading 219  
alert-info 219  
alert-light 218  
alert-link 219  
alert-primary 219  
alert-secondary 219  
alert-success 219  
alert-warning 219  
align-baseline 38  
align-bottom 38  
align-content 104  
align-content-around 105  
align-content-between 105  
align-content-center 105  
align-content-end 105  
align-content-start 104  
align-content-stretch 104, 106  
align-items 107  
align-items-baseline 107  
align-items-center 107  
align-items-end 107  
align-items-start 107  
align-items-stretch 107, 108  
align-middle 38  
align-self 108  
align-self-auto 108  
align-self-baseline 109  
align-self-center 109  
align-self-end 109  
align-self-start 109  
align-self-stretch 109  
align-text-bottom 38  
align-text-top 38  
align-top 38  
all 93, 450  
alpha() 411  
alt 76

and 374  
animation 306, 316, 325  
append() 391, 455  
arglist 364  
aria-atomic 323  
aria-controls 241, 242, 244,  
256, 259, 281  
aria-current 273, 274  
aria-disabled 274  
aria-expanded 241, 242, 244,  
254, 280, 281  
aria-haspopup 254, 280  
aria-hidden 275  
aria-label 281  
aria-labelledby 244, 280  
aria-live 323  
aria-pressed 184  
aria-selected 256, 259  
aria-valuemax 199  
aria-valuemin 199  
aria-valuenow 199  
arrow 304, 314  
as 434  
asin() 384  
atan() 384  
attr() 185  
auto 304, 315, 392, 393  
autocomplete 186, 187, 188,  
189  
autofocus 198  
autohide 325  
Autoprefixer 481

## B

backdrop 338  
background 79  
background-color 29  
background-image 79  
badge 223, 224, 271  
badge-danger 223  
badge-dark 223  
badge-info 223  
badge-light 223  
badge-pill 224  
badge-primary 223  
badge-secondary 223  
badge-success 223  
badge-warning 223  
bg-body 30  
bg-danger 30  
bg-dark 29  
bg-gradient 30  
bg-info 29  
bg-light 29  
bg-primary 30  
bg-secondary 30  
bg-success 29  
bg-transparent 30  
bg-warning 29  
bg-white 29  
blockquote 45  
blockquote-footer 45  
blue() 411  
bool 363  
Bootstrap 13, 435, 478, 480  
border 52, 64  
border-0, 65  
border-bottom 64  
border-bottom-0, 65  
border-box 85  
border-collapse 72  
border-color 65  
border-danger 65  
border-dark 65  
border-info 65  
border-left 64  
border-left-0, 65  
border-light 65  
border-primary 65  
border-right 64  
border-right-0, 65  
border-secondary 65  
border-spacing 72  
border-success 65  
border-top 64  
border-top-0, 65  
border-warning 65  
border-white 65  
both 88  
bottom 75, 304, 315  
boundary 306, 317  
box-shadow 92  
box-sizing 85  
breadcrumb 272  
breadcrumb-item 273  
browserslistrc 478  
btn 132, 134, 184, 186, 187,  
189  
btn-block 134, 173  
btn-check 188, 189  
btn-danger 132  
btn-dark 132  
btn-group 182, 183, 204, 211

- btn-group-lg 134, 182, 205, 206
  - btn-group-sm 134, 182, 205, 206
  - btn-group-toggle 186, 187
  - btn-group-vertical 182
  - btn-info 132
  - btn-lg 134, 205, 206, 217
  - btn-light 132
  - btn-link 134
  - btn-outline-danger 133
  - btn-outline-dark 133
  - btn-outline-info 133
  - btn-outline-light 133
  - btn-outline-primary 133, 186, 187, 189
  - btn-outline-secondary 133
  - btn-outline-success 133
  - btn-outline-warning 133
  - btn-primary 132
  - btn-secondary 132
  - btn-sm 134, 205, 206
  - btn-success 132
  - btn-toolbar 183
  - btn-warning 132
  - button 131, 184
  - Button 185
  - button() 184, 185
  - buttons 186, 187
- C**
- call() 427
  - caption-side 75
  - caption-top 75
  - card 225, 243, 244
  - card-body 225, 227
  - card-columns 238
  - card-deck 235
  - card-footer 225
  - card-group 233, 235
  - card-header 225, 226
  - card-header-pills 261
  - card-header-tabs 260
  - card-img 232, 233
  - card-img-bottom 231
  - card-img-overlay 233
  - card-img-top 231
  - card-link 228
  - card-subtitle 228
  - card-text 228
  - card-title 228
  - Carousel 285, 286, 287, 288, 292, 293, 294
  - carousel() 286, 287, 290, 291
  - carousel-caption 289
  - carousel-control-next 288
  - carousel-control-next-icon 288
  - carousel-control-prev 287
  - carousel-control-prev-icon 287
  - carousel-fade 290
  - carousel-indicators 288
  - carousel-inner 286
  - carouselInterface() 291, 292
  - carousel-item 286, 290
  - CDN 16
  - ceil() 383
  - change() 412
  - change-color() 412, 415, 416, 417, 419
  - checkbox 152, 186, 188
  - checked 153, 157, 158, 159, 186, 187, 188, 189
  - checkValidity() 191, 193
  - clamp() 383
  - clear 88
  - clearfix 88
  - click() 185
  - close 219, 324, 331
  - close() 220
  - cls 351
  - cmd 348
  - col 113, 118
  - col-1, 119
  - col-10, 119
  - col-11, 119
  - col-12, 120, 172
  - col-2, 119
  - col-3, 119
  - col-4, 119
  - col-5, 119
  - col-6, 119
  - col-7, 119
  - col-8, 119
  - col-9, 119
  - col-auto 118
  - col-form-label 177, 179
  - col-form-label-lg 178
  - col-form-label-sm 178
  - collapse 72, 240, 241, 242, 243, 244, 247, 281
  - Collapse 246
  - collapse() 246
  - collapsing 241
  - col-md 115
  - color 35, 168, 363, 406
    - ◇ adjust() 413
    - ◇ alpha() 411
    - ◇ blue() 411
    - ◇ change() 412
    - ◇ complement() 414
    - ◇ grayscale() 419
    - ◇ green() 410
    - ◇ hue() 411
    - ◇ ie-hex-str() 420
    - ◇ invert() 420
    - ◇ lightness() 412
    - ◇ mix() 419
    - ◇ opacity() 411
    - ◇ red() 410
    - ◇ saturation() 411
    - ◇ scale() 413
  - column-count 239
  - column-gap 239
  - comma 388, 392, 393
  - compact 473
  - comparable() 385
  - compatible() 385
  - complement() 414
  - compressed 354, 473
  - contain 93
  - container 28, 218, 306, 316
  - container-fluid 28, 331, 335
  - content 316
  - content-box 85
  - content-exists() 467, 477
  - cos() 384
  - CSS-переменные 31, 360
  - CSS-препроцессор 344
  - CSS-файл: подключение 441
  - custom-checkbox 154, 155, 195
  - custom-control 154, 155, 157, 160, 161
  - custom-control-inline 155, 161
  - custom-control-input 154, 157, 160
  - custom-control-label 154, 157, 160
  - custom-file 162, 195
  - custom-file-input 162
  - custom-file-label 162
  - custom-radio 160, 161, 195
  - custom-range 167
  - custom-select 148, 195

custom-select-lg 148  
custom-select-sm 148  
custom-switch 157  
cycle() 293

## D

darken() 417  
Dart Sass 470  
data-animation 306, 316, 325  
data-autohide 324, 325  
data-backdrop 333, 338  
data-boundary 306, 317  
data-browse 162  
data-container 306, 316  
data-content 312, 313, 314, 316  
data-delay 306, 316, 324, 325  
data-dismiss 218, 219, 323, 324, 330, 331, 332  
data-display 211  
data-flip 210  
data-html 303, 305, 314, 316  
data-interval 290, 291  
data-keyboard 291, 333, 338  
data-method 299  
data-no-jquery 217  
data-offset 211, 297, 299, 306, 317  
data-parent 243, 244  
data-pause 287, 291  
data-placement 304, 305, 315, 316  
data-reference 206, 212  
data-ride 286, 287, 292  
data-slide 287, 288  
data-slide-to 288  
data-spy 297  
data-target 241, 242, 281, 288, 297, 299, 332, 333  
data-toggle 184, 186, 187, 204, 241, 242, 244, 254, 256, 258, 259, 261, 280, 281, 302, 303, 307, 312, 314, 317, 332  
data-touch 292  
data-trigger 305, 313, 316  
data-wrap 292  
date 135  
datetime-local 136  
d-block 50, 76, 83, 181, 287  
delay 306, 316, 325  
dependencies 431, 470  
desaturate() 415

devDependencies 470  
d-flex 84, 94, 271, 283  
d-inline 83  
d-inline-block 83  
d-inline-flex 84, 94  
dir 95  
direction 293  
disable() 307, 318  
disabled 132, 145, 153, 157, 158, 159, 180, 203, 208, 250, 253, 256, 259, 269, 274  
display 83  
display-1, 46  
display-2, 46  
display-3, 46  
display-4, 46  
display-5, 46  
display-6, 46  
dispose() 213, 220, 246, 263, 293, 300, 307, 318, 326, 339  
d-md-block 289  
d-none 83, 92, 289  
d-print-block 83  
d-print-flex 84  
d-print-inline 83  
d-print-inline-block 83  
d-print-inline-flex 84  
d-print-none 83  
d-print-table 83  
d-print-table-cell 84  
d-print-table-row 83  
Dropdown 204, 209, 212, 213, 254, 280, 298, 300  
dropdown() 212  
dropdown-divider 207  
dropdown-header 207  
dropdown-item 207  
dropdown-item-text 207  
dropdown-menu 207, 213, 254, 280  
dropdown-menu-left 210  
dropdown-menu-right 210  
dropdown-toggle 204, 206, 254, 280  
dropdown-toggle-split 206  
dropleft 204, 210  
dropright 204, 210  
dropup 204, 210  
d-table 83  
d-table-cell 84  
d-table-row 83

## E

email 135  
embed-responsive 82  
embed-responsive-16by9, 82  
embed-responsive-1by1, 82  
embed-responsive-21by9, 82  
embed-responsive-4by3, 82  
embed-responsive-item 82  
embed-source-map 355  
embed-sources 355  
enable() 307, 318  
expanded 473  
extend() 456

## F

fade 219, 257, 331  
fade-in() 418  
fade-out() 418  
fallbackPlacement 306, 317  
false 363, 372, 373  
figure 79  
figure-caption 79  
figure-img 79  
file 161  
files 164  
file-text 164  
find() 164  
fixed-bottom 89, 283  
fixed-top 89, 283  
flex 102  
flex-basis 100, 102  
flex-column 95, 251, 252, 258  
flex-column-reverse 95  
flex-direction 94  
flex-fill 102, 268  
flex-grow 100, 102  
flex-grow-0, 100  
flex-grow-1, 100  
flex-nowrap 99, 140, 142  
flex-row 94  
flex-row-reverse 95  
flex-shrink 100, 101  
flex-shrink-0, 101  
flex-shrink-1, 101  
flex-sm-fill 251  
flex-sm-row 251, 252  
flex-wrap 97, 99, 104, 115  
flex-wrap-reverse 99  
Flex-контейнеры 94  
float 88, 110

float-left 88  
float-none 88  
float-right 88  
floor() 383  
focus 132, 313, 338  
focus() 195, 336  
font-italic 34  
font-monospace 33  
font-normal 34  
font-style 34  
font-weight 36  
font-weight-bold 36  
font-weight-bolder 36  
font-weight-light 36  
font-weight-lighter 36  
font-weight-normal 36  
for 139  
form-check 153, 154, 158, 159, 160, 195, 196  
form-check-inline 154, 160  
form-check-input 153, 156, 158, 159  
form-check-label 153, 158, 159  
form-control 136, 137, 138, 145, 147, 148, 195, 196  
form-control-color 168  
form-control-file 161  
form-control-lg 137, 138, 147, 178  
form-control-plaintext 138  
form-control-range 167  
form-control-sm 137, 138, 147, 178  
form-file 163, 164, 196  
form-file-button 163  
form-file-input 163, 164  
form-file-label 163  
form-file-lg 164  
form-file-sm 164  
form-file-text 163, 164  
form-group 137, 169, 172, 177  
form-inline 170, 172, 282  
form-label 139, 169  
form-range 168  
form-row 172, 173  
form-select 146, 147, 151, 196  
form-select-lg 147  
form-select-sm 147  
form-switch 158  
form-text 139  
from 294  
function 364, 420  
function-exists() 428

## G

g-0, 112, 236, 237  
g-1, 112  
g-2, 112, 172  
g-3, 112  
g-4, 112  
g-5, 112  
get() 397  
get-function() 426, 427  
getInstance() 185, 212, 220, 246, 262, 293, 300, 307, 318, 326, 339  
global-variable-exists() 367  
grayscale() 419  
green() 410  
gx-0, 112  
gx-1, 112  
gx-2, 112  
gx-3, 112  
gx-4, 112  
gx-5, 112  
gy-0, 112  
gy-1, 113  
gy-2, 113  
gy-3, 113  
gy-4, 113  
gy-5, 113

## H

h1, 46  
h-100, 86, 229, 236, 237  
h-25, 86  
h-50, 86  
h-75, 86  
handleUpdate() 339  
hasClass() 185  
has-key() 397  
h-auto 86  
height 76, 78, 79, 86  
help 474  
hidden 92  
Hide 323, 439  
hide() 213, 246, 307, 318, 326, 339  
high 199  
href 242, 244, 333  
HSL 411, 412  
hsl() 409, 476  
hsla() 409, 476  
html 305, 316

hue() 411  
hypot() 383

## I

ie-hex-str() 420  
if() 376, 455  
img-fluid 76, 77  
img-thumbnail 78  
include-path 473, 476, 480  
indented 469  
indented-syntax 474  
indent-type 474  
indent-width 474  
indeterminate 153  
index() 394, 405  
init 352  
initialism 44  
input-group 140, 141, 142, 143, 149, 150, 151, 152, 155, 156, 165, 166  
input-group-append 140, 149, 155, 156, 165  
input-group-lg 141, 143, 150, 152  
input-group-prepend 140, 149, 155, 165  
input-group-sm 141, 143, 150, 152  
input-group-text 140, 142, 149, 151, 155, 156, 165, 166  
insert() 405  
inspect() 365, 367  
install 351, 479  
interactive 358  
interval 290, 291  
invalid-feedback 192  
invalid-tooltip 193  
invert() 420  
invisible 92  
is-bracketed() 388  
is-invalid 194  
is-superslector() 455  
is-unitless() 385  
is-valid 194

## J

join() 392  
jumbotron 217  
jumbotron-fluid 218  
justify-content 103

justify-content-around 103  
 justify-content-between 103  
 justify-content-center 103, 181  
 justify-content-end 103, 181  
 justify-content-evenly 104  
 justify-content-start 103

## K

keyboard 291, 338  
 keydown() 215  
 keys() 400  
 keywords() 426, 466

## L

label 145  
 lang 18, 162  
 lead 34, 47, 217  
 left 304, 315  
 length 164  
 length() 389, 397, 403  
 lh-1, 37  
 lh-base 37  
 lh-lg 37  
 lh-sm 37  
 LibSass 470  
 lighten() 416  
 lightness 416  
 lightness() 412  
 linefeed 474  
 line-height 37  
 link-danger 49  
 link-dark 49  
 link-info 49  
 link-light 49  
 link-primary 50  
 link-secondary 50  
 link-success 49  
 link-warning 49  
 list 256, 259, 364, 387
 

- ◇ append() 391
- ◇ index() 394
- ◇ is-bracketed() 388
- ◇ join() 392
- ◇ length() 389, 397
- ◇ nth() 389
- ◇ separator() 388
- ◇ set-nth() 389
- ◇ zip() 394

list-group 69, 70, 226, 227,  
 256, 259, 267, 297, 298  
 list-group-flush 70, 267  
 list-group-horizontal 260, 268,  
 269  
 list-group-item 69, 259, 267,  
 269  
 list-group-item-action 259,  
 269, 271  
 list-group-item-danger 270  
 list-group-item-dark 270  
 list-group-item-info 270  
 list-group-item-light 270  
 list-group-item-primary 270  
 list-group-item-secondary 270  
 list-group-item-success 270  
 list-group-item-warning 270  
 list-inline 69  
 list-inline-item 69  
 list-separator() 388  
 list-unstyled 68, 81  
 load-css() 439, 441  
 load-path 431, 433  
 log() 185, 382  
 low 199

## M

m-0, 52  
 m-1, 53  
 m-2, 53  
 m-3, 53  
 m-4, 53  
 m-5, 53  
 map 355, 364, 395, 396
 

- ◇ get() 397
- ◇ has-key() 397
- ◇ keys() 400
- ◇ merge() 398
- ◇ remove() 399
- ◇ values() 400

 map-get() 397  
 map-has-key() 397  
 map-keys() 400  
 map-merge() 398  
 map-remove() 399  
 map-values() 400  
 margin 52, 57  
 margin-bottom 52, 54, 57  
 margin-left 52, 54, 58  
 margin-right 52, 55, 58

margin-top 52, 53, 57  
 Masonry 240  
 math 381

- ◇ \$e 381
- ◇ \$pi 381
- ◇ abs() 381
- ◇ acos() 384
- ◇ asin() 384
- ◇ atan() 384
- ◇ ceil() 383
- ◇ clamp() 383
- ◇ compatible() 385
- ◇ cos() 384
- ◇ floor() 383
- ◇ hypot() 383
- ◇ is-unitless() 385
- ◇ log() 382
- ◇ max() 382, 395
- ◇ min() 382, 395
- ◇ percentage() 386
- ◇ pow() 381
- ◇ random() 386
- ◇ round() 384
- ◇ sin() 384
- ◇ sqrt() 382
- ◇ tan() 384
- ◇ unit() 385

m-auto 52  
 max 167, 198, 199  
 max() 382, 395, 476  
 max-height 86  
 max-width 86  
 mb-0, 54  
 mb-1, 54  
 mb-2, 54  
 mb-3, 54, 137, 169, 177  
 mb-4, 54  
 mb-5, 54  
 mb-auto 52, 106  
 mb-n1, 57  
 mb-n2, 57  
 mb-n3, 57  
 mb-n4, 58  
 mb-n5, 58  
 media 80, 81, 82  
 media-body 80, 82  
 merge() 398  
 meta
 

- ◇ call() 427
- ◇ content-exists() 467

- ◇ function-exists() 428
  - ◇ get-function() 426, 427
  - ◇ global-variable-exists() 367
  - ◇ inspect() 365, 367
  - ◇ keywords() 426, 466
  - ◇ load-css() 439, 441
  - ◇ mixin-exists() 468
  - ◇ module-functions() 442
  - ◇ module-variables() 442
  - ◇ type-of() 364
  - ◇ variable-exists() 367
  - method 299
  - mh-100, 86
  - min 167, 199
  - min() 382, 395, 476
  - min-height 86
  - min-vh-100, 86
  - min-vw-100, 86
  - min-width 86
  - mix() 419
  - mixin-exists() 468
  - ml-0, 54
  - ml-1, 54
  - ml-2, 54
  - ml-3, 55
  - ml-4, 55
  - ml-5, 55
  - ml-auto 52, 106
  - ml-n1, 58
  - ml-n2, 58
  - ml-n3, 58
  - ml-n4, 58
  - ml-n5, 58
  - m-n1, 57
  - m-n2, 57
  - m-n3, 57
  - m-n4, 57
  - m-n5, 57
  - Modal 330, 331, 332, 333, 339, 340
  - modal() 338
  - modal-backdrop 333
  - modal-body 331, 334, 335, 336
  - modal-content 331
  - modal-dialog 331, 333, 336
  - modal-dialog-centered 336
  - modal-dialog-scrollable 334
  - modal-footer 331
  - modal-fullscreen 336
  - modal-fullscreen-lg-down 337
  - modal-fullscreen-md-down 337
  - modal-fullscreen-sm-down 337
  - modal-fullscreen-xl-down 337
  - modal-fullscreen-xxl-down 337
  - modal-header 331
  - modal-lg 333
  - modal-open 333
  - modal-sm 333, 334
  - modal-title 331
  - modal-xl 334
  - module-functions() 442
  - module-variables() 442
  - month 136
  - mr-0, 55
  - mr-1, 55
  - mr-2, 55
  - mr-3, 55
  - mr-4, 55
  - mr-5, 55
  - mr-auto 52, 106
  - mr-n1, 58
  - mr-n2, 58
  - mr-n3, 58
  - mr-n4, 58
  - mr-n5, 58
  - mt-0, 53
  - mt-1, 53
  - mt-2, 53
  - mt-3, 53
  - mt-4, 53
  - mt-5, 53
  - mt-auto 52, 106
  - mt-n1, 57
  - mt-n2, 57
  - mt-n3, 57
  - mt-n4, 57
  - mt-n5, 57
  - multiple 144, 161
  - mw-100, 86
  - mx-0, 56
  - mx-1, 56
  - mx-2, 56
  - mx-3, 56
  - mx-4, 56
  - mx-5, 56
  - mx-auto 52
  - mx-n1, 59
  - mx-n2, 59
  - mx-n3, 59
  - mx-n4, 59
  - mx-n5, 59
  - my-0, 56
  - my-1, 56
  - my-2, 56
  - my-3, 56
  - my-4, 56
  - my-5, 57
  - my-auto 52
  - my-n1, 59
  - my-n2, 59
  - my-n3, 59
  - my-n4, 59
  - my-n5, 59
- ## N
- name 159
  - nav 249, 253, 254, 255, 256, 258, 279, 297
  - navbar 276, 277, 297, 300
  - navbar-brand 278
  - navbar-collapse 281
  - navbar-dark 277
  - navbar-expand 279, 280, 281
  - navbar-expand-sm 280, 282
  - navbar-light 277
  - navbar-nav 279
  - navbar-text 277, 278
  - navbar-toggler 281
  - navbar-toggler-icon 281
  - nav-fill 250
  - nav-item 249, 254, 280
  - nav-justified 251
  - nav-link 250, 254, 256, 280
  - nav-pills 252, 254, 256, 258, 261
  - nav-tabs 253, 254, 255, 256, 257, 261
  - nest() 455
  - nested 472
  - next 288
  - next() 293
  - no-color 360
  - Node.js 344, 350
  - node-sass 470
    - ◇ версия 471
    - ◇ отличия 475
    - ◇ отслеживание изменений 475
    - ◇ создание CSS-файла 471
    - ◇ установка 470
  - no-error-css 360

no-gutters 111, 112, 236, 237  
none 93  
no-source-map 356  
not 374  
Notepad++ 20  
no-unicode 360  
novalidate 191  
NPM 351  
npm-run-all 481  
nth() 389  
null 363, 366, 373  
number 135, 363, 380

## O

offset 299, 306, 317  
offset-1, 123  
offset-10, 123  
offset-11, 123  
offset-2, 123  
offset-3, 123  
offset-4, 123  
offset-5, 123  
offset-6, 123  
offset-7, 123  
offset-8, 123  
offset-9, 123  
offset-lg-0, 124  
offset-md-0, 124  
offset-sm-0, 124  
offset-xl-0, 124  
offset-xxl-0, 124  
omit-source-map-url 473  
on() 215  
opacity() 418  
opacity() 411  
open 47  
optimum 199  
or 374  
order 110, 129  
order-0, 127  
order-1, 128  
order-10, 128  
order-11, 128  
order-12, 128  
order-2, 128  
order-3, 128  
order-4, 128  
order-5, 128  
order-6, 128  
order-7, 128  
order-8, 128

order-9, 128  
order-first 127  
order-last 127  
output 472  
output-style 472  
overflow 86  
overflow-auto 86  
overflow-hidden 86

## P

p-0, 59  
p-1, 60  
p-2, 60  
p-3, 60  
p-4, 60  
p-5, 60  
package.json 352, 431, 471, 478  
padding 52, 59  
padding-bottom 61  
padding-left 61  
padding-right 62  
padding-top 60  
page-item 273  
page-link 273, 274  
pagination 273, 275  
pagination-lg 275  
pagination-sm 275  
parent 206, 212  
parent() 164  
parse() 456  
password 135  
PATH 349  
pattern 190  
pause 287, 291  
pause() 293  
pb-0, 61  
pb-1, 61  
pb-2, 61  
pb-3, 61  
pb-4, 61  
pb-5, 61  
pe-auto 93  
pe-none 93  
percentage() 386  
pill 256, 258, 261  
pl-0, 61  
pl-1, 61  
pl-2, 62  
pl-3, 62  
pl-4, 62  
pl-5, 62  
placeholder 138  
placement 305, 316  
pointer-events 93  
poll 357  
popover 312, 314, 317, 319  
Popover 317, 318  
popover() 315  
popover-arrow 314  
popover-body 314  
popover-header 314  
Popper.js 15  
position 88  
position-absolute 88  
position-fixed 89  
position-relative 88, 297  
position-static 88, 153, 159  
position-sticky 89, 91  
PostCSS 478  
pow() 381  
pr-0, 62  
pr-1, 62  
pr-2, 62  
pr-3, 62  
pr-4, 62  
pr-5, 62  
precision 474  
pre-scrollable 42  
prev 287  
prev() 293  
preventDefault() 213, 221, 247, 263, 293, 308, 318, 326, 339  
progress 199  
progress-bar 199, 200  
progress-bar-animated 200  
progress-bar-striped 200  
prop() 188  
pt-0, 60, 179  
pt-1, 60  
pt-2, 60  
pt-3, 60  
pt-4, 60  
pt-5, 61  
px-0, 63  
px-1, 63  
px-2, 63  
px-3, 63  
px-4, 63  
px-5, 63  
py-0, 63  
py-1, 63  
py-2, 63

py-3, 64  
 py-4, 64  
 py-5, 64

## Q

quiet 359, 474  
 quote() 402

## R

radio 159, 187, 189  
 random() 386  
 range 167  
 readonly 138  
 recursive 474  
 red() 410  
 refresh() 300  
 relatedTarget 213, 263, 293,  
 339  
 relative 355  
 remove() 399  
 removeClass() 195  
 replace() 457  
 required 137, 190  
 reset 131  
 rgb() 407, 476  
 rgba() 408, 476  
 ride 287, 291  
 right 304, 315  
 round() 384  
 rounded 66, 78  
 rounded-0, 66  
 rounded-bottom 66  
 rounded-circle 66, 78  
 rounded-left 66  
 rounded-lg 66  
 rounded-pill 66, 78, 224  
 rounded-right 66  
 rounded-sm 66  
 rounded-top 66  
 row 111, 172, 173, 177, 179,  
 236  
 row-cols-1, 116, 237  
 row-cols-2, 116  
 row-cols-3, 116  
 row-cols-4, 116  
 row-cols-5, 117  
 row-cols-6, 117  
 row-cols-auto 118  
 row-cols-lg-3, 237  
 row-cols-md-2, 237

row-cols-md-auto 172  
 Ruby Sass 470  
 rule 450

## S

sanitize 305, 316  
 sanitizeFn 305, 316  
 SASS 344, 468
 

- ◊ версия 351
- ◊ интерактивный режим 358
- ◊ отслеживание изменений 357
- ◊ создание CSS-файла 354
- ◊ установка 351

 saturate() 415  
 saturation 415, 419  
 saturation() 411  
 save-dev 470  
 scale() 413  
 scale-color() 413, 415, 416,  
 417, 418  
 scripts 471  
 scroll 297  
 scrollspy 297, 300  
 ScrollSpy 299, 300  
 scrollspy() 299  
 SCSS 354, 468  
 search 135  
 selected 144  
 selector 306, 316, 444
 

- ◊ append() 455
- ◊ extend() 456
- ◊ is-sup-selector() 455
- ◊ nest() 455
- ◊ parse() 456
- ◊ replace() 457
- ◊ simple-selectors() 456
- ◊ unify() 456

 selector-append() 455  
 selector-extend() 456  
 selector-nest() 455  
 selector-parse() 456  
 selector-replace() 457  
 selector-unify() 456  
 separator() 388  
 set-nth() 389  
 shadow 92  
 shadow-lg 92  
 shadow-none 93  
 shadow-sm 92  
 Show 208, 219, 241, 244, 253,  
 254, 257, 323, 338, 439  
 show() 213, 246, 263, 307,  
 318, 326, 339  
 simple-selectors() 456  
 sin() 384  
 size 144  
 sizes 77  
 slice() 404  
 slide 286, 290  
 small 34, 47  
 source-comments 473  
 source-map 473  
 source-map-contents 473  
 source-map-embed 473  
 source-map-root 473  
 source-map-urls 355  
 sourceRoot 473  
 space 388, 392, 393  
 spinner 201, 202, 203  
 spinner-border 201  
 spinner-border-sm 201  
 spinner-grow 202  
 spinner-grow-sm 202  
 sqrt() 382  
 src 76  
 srcset 77  
 sr-only 139, 171, 201, 202,  
 206, 274  
 static 211, 333  
 step 167  
 sticky-top 89, 91, 283, 284  
 stretched-link 50, 80, 228, 244  
 str-index() 405  
 string 363, 401
 

- ◊ index() 405
- ◊ insert() 405
- ◊ length() 403
- ◊ quote() 402
- ◊ slice() 404
- ◊ to-lower-case() 404
- ◊ to-upper-case() 404
- ◊ unique-id() 406
- ◊ unquote() 402

 str-insert() 405  
 str-length() 403  
 str-slice() 404  
 style 354  
 submit 131

submit() 193  
SVG 78

## T

tab 253, 255, 256, 264  
Tab 262, 263  
tab() 262  
tab-content 257  
tabindex 198, 274, 303, 314  
table 72  
table-active 74  
table-bordered 72  
table-borderless 72  
table-danger 74  
table-dark 74, 75  
table-hover 74  
table-info 74  
table-light 74  
table-primary 74  
table-responsive 73  
table-secondary 74  
table-sm 73  
table-striped 73  
table-success 74  
table-warning 74  
tab-pane 257  
tan() 384  
target 263, 299  
tel 135  
template 306, 317  
text 93, 135  
text() 164  
text-align 37  
text-black-50, 35  
text-body 35  
text-break 40  
text-capitalize 39  
text-center 37  
text-danger 35  
text-dark 35  
text-decoration 39  
text-decoration-line-through 39  
text-decoration-none 39  
text-decoration-underline 39  
text-hide 36  
text-info 35  
text-justify 38  
text-left 37  
text-light 35  
text-lowercase 39  
text-monospace 33  
text-muted 35, 139, 226  
text-nowrap 40, 303, 313  
text-primary 35  
text-reset 35  
text-right 38  
text-secondary 35  
text-sm-center 251  
text-success 35  
text-transform 39  
text-truncate 41  
text-uppercase 39  
text-warning 35  
text-white 35  
text-white-50, 35  
text-wrap 40  
thead-dark 74  
thead-light 74  
through 377  
time 135  
title 44, 302, 303, 306, 312,  
313, 314, 316  
to 294, 376  
to() 293  
Toast 323, 324, 326, 327  
toast() 325  
toast-body 324  
toast-header 323  
toggle 212, 246  
toggle() 185, 213, 246, 307,  
318, 339  
toggleEnabled() 307, 318  
to-lower-case() 404  
tooltip 302, 304, 307, 308, 312  
Tooltip 307  
tooltip() 305  
tooltip-arrow 304  
tooltip-inner 304  
top 75, 89, 304, 315  
touch 292  
to-upper-case() 404  
transparentize() 418  
trigger 305, 316  
true 363, 372, 373  
type 135, 152, 159, 161, 167,  
168  
type-of() 364

## U

unify() 456  
unique-id() 406  
unit() 385  
unitless() 385

unquote() 402  
update() 213, 307, 318  
url 135  
url() 429  
user-select 93  
user-select-all 93  
user-select-auto 93  
user-select-none 93  
using 467, 477

## V

val() 342  
valid-feedback 192  
valid-tooltip 193  
value 159, 167, 198, 199  
values() 400  
var() 31, 360  
variable-exists() 367  
version 349, 351  
vertical-align 38  
vh-100, 86  
viewport 26  
visibility 92  
visible 92  
vw-100, 86

## W

w-100, 77, 86, 116, 287  
w-25, 85, 200  
w-50, 86  
w-75, 86  
was-validated 191, 193, 194  
watch 357, 475  
w-auto 85  
week 136  
which 215  
whiteList 305, 316  
width 76, 78, 79, 85  
with 450, 451, 476  
without 450, 451  
wrap 115, 292

## Y

YouTube 82

## Z

zip() 39

**А**

- Абзац 46
- Абсолютное значение 381
- Автодополнение 144
- Адаптивный дизайн 26
- Аккордеон 243
- Альфа-канал 411, 417
- Арккосинус 384
- Арксинус 384
- Арктангенс 384
- Ассоциативные массивы 395
  - ◇ добавление элементов 398
  - ◇ изменение значения 398
  - ◇ количество элементов 397
  - ◇ перебор элементов 399
  - ◇ получение значения 397
  - ◇ преобразование в список 400
  - ◇ проверка существования ключа 397
  - ◇ создание 396
  - ◇ сравнение 400
  - ◇ удаление элементов 399
- Атрибуты стилей 443
  - ◇ в зависимости от условия 454
  - ◇ вложенные 452
  - ◇ генерация названий 453

**Б**

Блок 82

**В**

- Версии библиотек 20
- Видео 82
- Вложенные правила 445
- Возведение в степень 381
- Всплывающие
  - ◇ информеры 312
  - ◇ подсказки 302
  - ◇ уведомления 323
- Выделение фрагментов текста 223
- Выключатель 157
- Выпадающее меню 207

Выравнивание:

- ◇ вертикальное 38
- ◇ горизонтальное 37

Вычитание 368

**Г**

- Гиперссылка 48
- Горизонтальная линия 47

**Д**

- Деление 369
- Диалоговые окна 330

**Е**

Единицы измерения 385

**З**

- Заголовок 46
- Зебра 73
- Значки 78

**И**

- Идентификатор уникальный 406
- Изменение регистра символов 39
- Изображение 76
- Индекс 41
- Индексные файлы 432
- Индикатор состояния загрузки 201
- Инструменты разработчика 22
- Интерактивный режим 358
- Интерполяция 365
- Информеры 312

**К**

- Карточки 225
- Квадратный корень 382
- Кнопка 131
  - ◇ с выпадающим меню 204
- Кодировка файлов 403
- Командная строка 348
- Комментарии 358

- Консоль 25
- Контейнер 82
  - ◇ базовый 28
- Косинус 384
- Курсив 34

**Л**

- Линия 47
- Логарифм 382

**М**

- Маркированный список 67
- Медиазапросы 26
- Меню 207
- Миксины 460, 469
  - ◇ @content 466
  - ◇ @include 461
  - ◇ @mixin 460
  - ◇ блоки содержимого 466
  - ◇ вызов 461
  - ◇ необязательные параметры 465
  - ◇ определение 460
  - ◇ передача значений 464
  - ◇ проверка существования 468
  - ◇ расположение 463
  - ◇ создание 460
- Минус 368
- Модальные диалоговые окна 330
- Модули 428
  - ◇ @forward 436
  - ◇ @import 429
  - ◇ @use 432
  - ◇ load-css() 439
  - ◇ импорт всех идентификаторов 434
  - ◇ переопределение значений переменных 435
  - ◇ подключение 429, 432, 436
  - ◇ получение всех переменных 442
  - ◇ получение всех функций 442

- ◇ пути поиска 430
- ◇ частные идентификаторы 434
- Моноширинный шрифт 33

## Н

- Насыщенность 411, 415, 419
- Нумерованный список 67

## О

- Области видимости переменных 361
- Окно 330
- Операторы 368
  - ◇ ветвления 375
  - ◇ для работы со строками 371
  - ◇ математические 368
  - ◇ приоритет выполнения 370
  - ◇ сравнения 372
- Остаток от деления 369
- Отображение элементов 92
- Отслеживание прокрутки 297
- Отступ 51
  - ◇ внешний 52
  - ◇ внутренний 59
- Оттенок цвета 411

## П

- Панель навигации 276
- Панель с вкладками 255
- Переключатель 159
- Переменные:
  - ◇ в CSS 360
  - ◇ в SASS 360
  - ◇ глобальные 361, 362
  - ◇ локальные 361
  - ◇ области видимости 361
  - ◇ переопределение значений 435
  - ◇ подстановка значений 365
  - ◇ проверка существования 367
  - ◇ типы данных 363

- Подключение CSS-файлов 441
- Подключение библиотеки
  - ◇ Bootstrap 14
  - ◇ jQuery 16
- Подсказки 302
- Подстановка значений переменных 365
- Позиционирование 88
- Поиск в строке 405
- Поле выбора файла 161
- Полужирный шрифт 37
- Поля для ввода данных 135
- Постраничная навигация 273
- Примеси 460
- Приоритет выполнения операторов 370
- Прозрачность цвета 411, 417
- Прокрутка 297
- Пространство имен 433

## Р

- Раздел HTML-документа
  - ◇ BODY 19
  - ◇ HEAD 19
- Размеры 85
- Рамка 64
- Регистр символов 39

## С

- Светлота 412, 416
- Селекторы 443
  - ◇ @at-root 448
  - ◇ вложенные правила 445
  - ◇ генерация названий 453
  - ◇ заполнители 457
  - ◇ оператор & 451
  - ◇ основные 444
  - ◇ привязка к элементам 445
  - ◇ родительские 451
  - ◇ функции для работы с селекторами 455
  - ◇ шаблонные 457
- Синус 384
- Система сеток в библиотеке Bootstrap 110
- Скругление углов 66
- Сложение 368

- Списки 387
  - ◇ добавление элемента 391
  - ◇ изменение значения 389
  - ◇ количество элементов 389
  - ◇ объединение 392
  - ◇ перебор элементов 390
  - ◇ поиск значения 394
  - ◇ получение значения 389
  - ◇ создание 387
  - ◇ сравнение 395
- Списки определений 70
- Список 67, 267
  - ◇ элемент управления 144
- ◇ автодополнения 144
- Сравнение
  - ◇ ассоциативных массивов 400
  - ◇ операторы 372
  - ◇ списков 395
  - ◇ строк 406
- Строки 401
  - ◇ вставка фрагмента 405
  - ◇ кодировка файлов 403
  - ◇ количество символов 403
  - ◇ конкатенация 371
  - ◇ операторы 371
  - ◇ подстановка значений 365
  - ◇ поиск 405
  - ◇ регистр символов 404
  - ◇ создание 401
  - ◇ сравнение 406
  - ◇ фрагмент строки 404

## Т

- Таблица 71
- Тангенс 384
- Текстовая область 136
- Типы данных 363
- Точки останова 26, 27
- Тригонометрические функции 384

## У

- Уведомления 218, 323
- Умножение 369

Унарный минус 368

Уникальный идентификатор  
406

## Ф

Флажок 152

Фон 29, 74

Форма 131

Функции 420

- ◇ возвращаемое значение 421
- ◇ вызов 421
- ◇ необязательные параметры 425
- ◇ обратный вызов 426
- ◇ определение 421
- ◇ передача значений 423
- ◇ передача ссылки на функцию 426
- ◇ проверка существования 428
- ◇ расположение 423
- ◇ создание 421

## Х

Хлебные крошки 272

## Ц

Цвет 406

- ◇ альфа-канал 417

- ◇ изменение значений компонентов 412
- ◇ инвертирование 420
- ◇ линии рамки 65
- ◇ насыщенность 415
- ◇ негатив 420
- ◇ оттенки серого 419
- ◇ получение значений компонентов 410
- ◇ прозрачность 417
- ◇ смешивание 419
- ◇ текста 35
- ◇ указание значения 407
- ◇ фона 29
- ◇ яркость 416

Циклы:

- ◇ @each 378
- ◇ @for 376
- ◇ @while 377

Цитата 45

## Ч

Числа 380

- ◇ возведение в степень 381
- ◇ квадратный корень 382
- ◇ логарифм 382
- ◇ максимизация 483+13
- ◇ малое значение 382
- ◇ математические константы 381

- ◇ минимальное значение 382
- ◇ округление 383
- ◇ основные функции 381
- ◇ преобразование в строку 386
- ◇ псевдослучайные 386
- ◇ тригонометрические функции 384

## Ш

Шаблон

HTML-документа 18

Шаблонные селекторы 457

Шаблоны 457

Шкала с ползунком 167

Шрифт 32

- ◇ гарнитура 33
- ◇ зачеркнутый 39
- ◇ курсивный 34
- ◇ моноширинный 33
- ◇ перечеркнутый 39
- ◇ подчеркнутый 39
- ◇ полужирный 36, 37
- ◇ стиль 34
- ◇ цвет 35

## Я

Яркость 412, 416