Node.js

Разработка серверных веб-приложений на JavaScript

Дэвид Хэррон

ДМК ПРЕСС

УДК 004.738.5:004.45Node
ББК 32.973.202-018.2
Х99

Хэррон Д.
Х99 Node.js. Разработкасерверных веб-приложений в JavaScript: Пер. с англ. Слинкина А. А. – М.: ДМК Пресс, 2012. – 144 с.: ил.
ISBN 978-5-94074-809-0

Книга посвящена разработке веб-приложений в Node.js – платформе, которая выводит язык JavaScript за пределы браузера и позволяет использовать его в серверных приложениях. В основе платформы лежит исключительно быстрый движок JavaScript, заимствованный из браузера Chrome, к которому добавлена быстрая и надежная библиотека асинхронного сетевого ввода/вывода. Основной упор в Node.js делается на создании высокопроизводительных, хорошо масштабируемых клиентских и серверных приложений.

На практических примерах вы научитесь пользоваться серверным и клиентским объектами HTTP, каркасами Connect и Express, освоите алгоритмы асинхронного выполнения и узнаете, как работать с базами данных на основе SQL и с MongoDB.

Начав с практических рекомендаций по установке и настройке Node.js в режиме разработки и эксплуатации, вы научитесь разрабатывать клиентские и серверные HTTP-приложения; познакомитесь с применяемой в Node.js системой организации модулей на основе спецификации CommonJS, позволяющей реализовать подмножество технологии объектно-ориентированного проектирования.

Издание предназначено для программистов, знакомых с основами JavaScript и веб-разработки.

УДК 004.738.5:004.45Node
ББК 32.973.202-018.2

ISBN 978-1-849515-14-6 (анг.)
ISBN 978-5-94074-809-0 (рус.)

Содержание

Об авторе

Дэвид Хэррон вот уже больше 20 лет занимается созданием программного обеспечения, работая в Кремниевой долине в роли разработчика и инженера по контролю качества. Его последнее место работы – архитектор по организации контроля качества в компании Yahoo!, где ведутся работы по созданию новой платформы для веб-приложений на основе Node.

В компании Sun Microsystems Дэвид занимал должность архитектора по организации контроля качества Java SE и работал главным образом над средствами автоматизации тестирования, в том числе написанным на базе AWT классом Robot, который ныне широко применяется в программах автоматизации тестирования графического интерфейса пользователя (ГИП). Он принимал участие в запуске проектов OpenJDK и JDK-Distros, а также отвечал за организацию конкурса Mustang Regressions Contest, идея которого состояла в том, чтобы обратиться к сообществу разработчиков на Java с просьбой поискать ошибки в версии 1.6.

До перехода в Sun Дэвид работал в компании VXtreme, где занимался разработкой программных средств потокового видео. После покупки этой компании корпорацией Microsoft созданный ей продукт лег в основу Windows Media Player. В компании The Wollongong Group Дэвид работал над клиентским и серверным ПО электронной почты и принимал участие в деятельности нескольких рабочих групп IETF, направленной на совершенствование протоколов электронной почты.

Дэвида интересуют транспортные средства с электрическими двигателями, мировые запасы энергии, изменение климата и вопросы охраны окружающей среды. Он сооснователь компании Transition Silicon Valley. В качестве сетевого журналиста он ведет раздел Green Transportation Examiner на сайте examiner.com, пишет на темы экологии в блоге на сайте 7gen.com, организовал дискуссионный форум по электромобилям на сайте visforvoltage.org, а также обсуждает различные программы, в том числе Node.js, Drupal и Doctor Who на сайте davidherron.com.

Предисловие

Добро пожаловать в мир разработки ПО на базе Node (другое название – Node.js). Node – это недавно появившаяся платформа, которая выводит язык JavaScript за пределы браузера и позволяет использовать его в серверных приложениях. В основе платформы лежит исключительно быстрый движок JavaScript, заимствованный из браузера Chrome, V8, к которому добавлена быстрая и надежная библиотека асинхронного сетевого ввода/вывода. Основной упор в Node делается на создании высокопроизводительных, хорошо масштабируемых клиентских и серверных приложений для «веб реального времени».

Эту платформу разработал Райан Дал (Ryan Dahl) в 2009 году, после двух лет экспериментирования с созданием серверных веб-компонентов на Ruby и других языках. В ходе своих исследований он пришел к выводу, что вместо традиционной модели параллелизма на основе потоков следует обратиться к событийно-ориентированным системам. Эта модель была выбрана за простоту (хорошо известно, что многопоточные системы трудно реализовать правильно), за низкие накладные расходы, по сравнению с идеологией «один поток на каждое соединение», и за быстродействие. Цель Node – предложить «простой способ построения масштабируемых сетевых серверов». При проектировании за образец были взяты такие системы, как Event Machine (Ruby) и каркас Twisted (Python).

В настоящей книге рассматривается в первую очередь вопрос о построении веб-приложений с помощью Node. Мы познакомимся с важными концепциями, которые необходимо понимать, чтобы повысить быстродействие приложения. Для этого мы будем писать реальные приложения, подробно анализировать их составные части и обсуждать, как применить эти идеи в своих программах. Мы установим Node и npm и научимся устанавливать и разрабатывать npm-пакеты и модули Node. Мы создадим несколько приложений, изучим, как отражаются на отзывчивости цикла обработки событий продолжительные вычисления, расскажем о двух способах распределения нагрузки между серверами, поработаем с каркасом Express и прочее.

О содержании книги

Глава 1 «Что такое Node?» содержит введение в платформу Node. Мы поговорим о том, где она применяется, об архитектурных решениях, принятых в Node, о ее истории и об истории использования JavaScript на стороне сервера, а также о том, почему JavaScript не должен быть замурован в браузере.

В главе 2 «Настройка Node» речь пойдет о настройке среды разработки для Node, в том числе о нескольких способах сборки и установки из исходного кода. Мы также коснемся вопроса о развертывании Node на производственных серверах.

Глава 3 «Модули Node» посвящена модулям – единицам модульной структуры приложений для Node. Мы разберемся в том, что такое модули, и разработаем несколько. Затем познакомимся с программой npm, Node Package Manager (менеджер пакетов Node) и рассмотрим несколько способов использования npm для управления установленными пакетами, а также для разработки и распространения npm-пакетов.

В главе 4 «Вариации на тему простого приложения» мы, уже вооруженные знанием основ, приступим к изучению процесса разработки приложений на платформе Node. Точнее, мы разработаем простое приложение, применяя саму Node, систему промежуточного уровня Connect и веб-каркас Express. Хотя приложение несложное, оно даст нам возможность изучить цикл обработки событий в Node, узнать о его адаптации к продолжительным вычислениям, познакомиться с синхронными и асинхронными алгоритмами и вынести громоздкие вычисления на вспомогательный сервер.

В главе 5 «Простой веб-сервер, объекты EventEmitter и HTTP-клиенты» объясняется, что в Node клиентские и серверные объекты – это соответственно передний план и центр. Мы подробно рассмотрим обе стороны протокола HTTP, для чего разработаем клиентское и серверное приложения на основе HTTP.

В главе 6 «Хранение и выборка данных» речь пойдет о том, что в большинстве приложений необходимо какое-то надежное хранилище для длительного хранения данных. Мы рассмотрим применение для этой цели СУБД на основе SQL и MongoDB. Попутно поговорим о применении каркаса Express для аутентификации пользователей и создания более привлекательной страницы с сообщением об ошибке.

Что необходимо для чтения этой книги

В настоящее время мы обычно собираем Node из исходного кода. Лучше всего платформа работает в системах на базе Unix или совместимых со стандартом POSIX. Предъявляемые Node требования достаточно скромны, а самым важным инструментом является то, что находится у вас между ушами.

Для установки из исходного кода необходима система на базе Unix/POSIX (Linux, Mac, FreeBSD, OpenSolaris и т. д.), современный компилятор C/C++, библиотеки OpenSSL и Python версии не ниже 2.4.

Программы для Node можно набирать в любом текстовом редакторе, но лучше использовать такой, который понимает синтаксис JavaScript, HTML, CSS и т. д.

Хотя эта книга посвящена разработке веб-приложений, иметь веб-сервер необязательно. В состав Node входит собственный веб-сервер.

На кого рассчитана эта книга

Данная книга рассчитана на любого программиста, ищущего приключений, сопутствующих новой программной платформе, построенной на основе новой парадигмы программирования.

Разработчики серверных компонентов, возможно, придут к выводу, что свежие идеи позволили им по-новому взглянуть на процесс создания веб-приложений. JavaScript – мощный язык, а асинхронная природа Node подчеркивает его сильные стороны.

Разработчикам, имеющим опыт работы с JavaScript в браузере, будет интересно применить свои знания на новой территории и посмотреть, что значит писать на JavaScript без доступа к DOM. (Раз нет браузера, то нет и DOM, если только вы не установите JSDom.)

Хотя главы зависят друг от друга, порядок чтения выбираете вы сами.

Мы предполагаем, что вы уже умеете писать программы и знакомы с современными языками программирования, такими как JavaScript.

Графические выделения

В этой книге используются различные шрифты для обозначения типа информации. Ниже приведено несколько примеров с пояснениями.

Фрагменты кода внутри абзаца выделяются следующим образом: «Объект `http` инкапсулирует протокол HTTP, а его метод `http.createServer` создает полноценный веб-сервер, прослушивающий порт, указанный в методе `.listen`».

Блок кода выделяется следующим образом:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

Если мы хотим привлечь внимание к конкретной части кода, то выделяем ее полужирным шрифтом:

```
var util = require('util');
var A = "a different value A";
var B = "a different value B";
var m1 = require('./module1');
util.log('A='+A+' B='+B+' values='+util.inspect(m1.values()));
```

Команды и выводимая ими информация записываются так:

```
$ sudo /usr/sbin/update-rc.d node defaults
```

Глава 1. ЧТО ТАКОЕ NODE?

Node – это захватывающая новая платформа для разработки веб-приложений, серверов приложений, произвольных сетевых серверов и клиентов, да и вообще для программирования. Она спроектирована так, чтобы обеспечить высочайшую масштабируемость сетевых приложений – за счет хитроумного сочетания асинхронного ввода/вывода, использования JavaScript на стороне сервера, изобретательного использования анонимных функций JavaScript и однопоточной событийно-ориентированной архитектуры.

Принятая в Node модель принципиально отличается от распространенных платформ для построения серверов приложений, в которых масштабируемость достигается за счет многопоточности. Утверждается, что благодаря событийно-ориентированной архитектуре снижается потребление памяти, повышается пропускная способность и упрощается модель программирования. Сейчас платформа Node быстро развивается, и многие считают ее привлекательной альтернативой традиционному подходу к разработке веб-приложений – на базе Apache, PHP, Python и т. п.

В основе Node лежит автономная виртуальная машина JavaScript с расширениями, делающими ее пригодной для программирования общего назначения с упором на разработку серверов приложений. Платформу Node не имеет смысла напрямую сравнивать ни с языками программирования, которые обычно используются для создания веб-приложений (PHP/Python/Ruby/Java и прочие), ни с контейнерами, реализующими протокол HTTP (Apache/Tomcat/Glassfish и т. д.). В то же время многие считают, что потенциально она может заменить традиционные стеки веб-приложений.

В основе реализации лежит цикл обработки событий неблокирующего ввода/вывода и библиотеки файлового и сетевого ввода/вывода, причем все это построено поверх движка V8 JavaScript (заимствованного из веб-браузера Chrome). Библиотека ввода/вывода обладает достаточной общностью для реализации любого протокола на базе TCP или UDP: DNS, HTTP, IRC, FTP и др. Но хотя она поддерживает разработку серверов и клиентов произвольного протокола, чаще всего применяется для создания обычных веб-сайтов, где заменяет Apache/PHP или Rails.

Эта книга представляет собой введение в платформу Node. Мы предполагаем, что вы уже умеете писать программы, знакомы с языком JavaScript и знаете, как разрабатываются веб-приложения на других языках. Мы напишем несколько работоспособных приложений и убедимся, что учиться лучше всего, копаясь в коде.

Что позволяет делать Node?

Node – платформа для написания JavaScript-приложений вне веб-браузера. Это не тот JavaScript, с которым все мы знакомы по опыту работы с браузерами. В Node не встроена ни объектная модель документа (DOM), ни какие-либо еще возможности браузера. Именно язык JavaScript в сочетании с асинхронным вводом/выводом делает Node мощной платформой для разработки приложений.

Но вот для чего Node непригодна, так это для разработки персональных приложений с графическим интерфейсом пользователя (ГИП). На сегодняшний день в Node нет встроенного эквивалента Swing (или SWT, если вам больше нравится эта библиотека). Нет и подключаемой библиотеки ГИП для Node, и внедрить Node в браузер тоже нельзя. Если бы для Node существовала библиотека ГИП, то на этой платформе можно было строить и персональные приложения. Недавно появилось несколько проектов по созданию интерфейса между Node и GTK, итогом которых должна стать кросс-платформенная библиотека ГИП. В состав движка V8, используемого в Node, входят API-расширения, позволяющие писать на C/C++ код для расширения JavaScript или интеграции движка с платформенными библиотеками.

Помимо встроенного умения исполнять код на JavaScript, включенные в состав дистрибутива модули предоставляют и другие возможности:

- утилиты командной строки (для включения в скрипты оболочки);
- средства написания интерактивных консольных программ (цикл «чтение – выполнение – печать»);
- великолепные функции управления процессами для наблюдения за дочерними процессами;
- объект Buffer для работы с двоичными данными;
- механизм для работы с сокетами TCP и UDP с полным комплектом обратных вызовов в ответ на события;
- поиск в системе DNS;
- средства для создания серверов и клиентов протоколов HTTP и HTTPS, построенные на основе библиотеки TCP-сокетов;
- средства доступа к файловой системе;
- встроенная рудиментарная поддержка автономного тестирования с помощью утверждений.

Сетевой слой Node находится на низком уровне, но работать с ним все равно просто. Например, модули HTTP позволяют реализовать HTTP-сервер (или клиент), написав всего несколько строк кода, но тем не менее на этом уровне программист работает очень близко к реальным запросам по протоколу и может точно указать, какие HTTP-заголовки следует включать в ответ на запрос. Если программист на PHP обычно не интересуется заголовками, то для программиста на Node они существенны.

Иными словами, написать на Node HTTP-сервер очень просто, но типичному разработчику веб-приложений нет нужды работать на таком низком уровне. Например, кодируя на PHP, программист предполагает, что Apache уже присутст-

вует, так что реализовывать серверную часть стека ему не нужно. Сообщество, сложившееся вокруг Node, создало широкий спектр каркасов для разработки веб-приложений, в том числе Connect, которые позволяют быстро сконфигурировать HTTP так, чтобы предоставлялось все, к чему мы привыкли, – сеансы, куки, обслуживание статических файлов, протоколирование и т. д. – и пусть разработчик занимается бизнес-логикой приложения.

Серверный JavaScript

Хватит чесать в затылке. Ведь именно этим вы сейчас и занимаетесь, правда? – чешете затылок и бормочете: «Что браузерному языку делать на сервере?» Но на самом-то деле у JavaScript есть долгая и мало кому известная история применения вне браузера. JavaScript – язык программирования, такой же, как любой другой, поэтому правильнее было бы спросить: «Почему JavaScript должен быть замурован в браузере?»

На заре эры веб инструменты для создания веб-приложений находились еще в зачаточном состоянии. Кто-то экспериментировал с написанием CGI-скриптов на Perl или на TCL, языки PHP и Java еще только разрабатывались, и даже JavaScript применялся на стороне сервера. Одним из первых серверов приложений был LiveWire от компании Netscape, и в нем использовался JavaScript. В некоторых версиях технологии Microsoft ASP использовался язык JScript, версия JavaScript от Microsoft. Из относительно недавних серверных проектов, в которых используется JavaScript, назовем каркас разработки приложений RingoJS, популярный в мире Java. Он построен на базе Rhino – реализации JavaScript, написанной на Java.

Node дополняет картину ранее невиданной комбинацией – сочетанием быстрого событийно-ориентированного механизма ввода/вывода и быстрого движка JavaScript, такого как V8, используемого в браузере Google Chrome.

Почему имеет смысл использовать Node?

Язык JavaScript очень популярен благодаря присутствию в любом веб-браузере. Он ни в чем не уступает другим языкам, но при этом поддерживает многие современные представления о том, каким должен быть язык программирования. Благодаря широкому распространению имеется немало опытных программистов на JavaScript.

Это динамический язык со слабо типизированными, динамически расширяемыми объектами, которые неформально объявляются по мере необходимости. Функции в нем являются полноценными объектами и обычно используются в виде анонимных замыканий. Это делает JavaScript более мощным языком, по сравнению с некоторыми другими, часто применяемыми для разработки веб-приложений. Теоретически наличие подобных возможностей должно повышать продуктивность программистов. Но скажем откровенно: споры между сторонни-

ками динамических и статических языков, а также строгой и слабой типизации до сих пор не утихли и вряд ли когда-нибудь утихнут.

Один из основных недостатков JavaScript – Глобальный Объект. Все переменные верхнего уровня «сваливаются» в Глобальный Объект, и при использовании одновременно нескольких модулей это может привести к неуправляемому хаосу. Поскольку веб-приложения обычно состоят из множества объектов, возможно, создававшихся разными организациями, то может возникнуть опасение, будто программирование для Node сродни хождению по минному полю, нашпигованному конфликтующими между собой глобальными объектами. Однако это не так. На самом деле в Node используется система организации модулей CommonJS, а это означает, что локальные переменные некоторого модуля так и будут локальными в нем, пусть даже выглядят как глобальные. Такое четкое разграничение между модулями решает проблему Глобального Объекта.

Использование единого языка программирования на сервере и на клиенте давно уже было мечтой разработчиков для веб. Своими корнями эта мечта уходит в период становления Java, когда апплеты представлялись клиентским интерфейсом к написанным на Java серверным приложениям, а JavaScript первоначально виделся как облегченный скриптовый язык взаимодействия с апплетами. Но что-то на этом пути не сложилось, и в результате не Java, а JavaScript стал основным языком на стороне клиента-браузера. С появлением Node мы наконец сможем реализовать мечту – сделать JavaScript языком, используемым по обе стороны веб – на стороне клиента и сервера.

У единого языка есть несколько потенциальных плюсов:

- одни и те же программисты могут работать над обеими сторонами приложения;
- код проще переносить с сервера на клиент и обратно;
- общий для клиента и сервера формат данных (JSON);
- общий программный инструментарий;
- общие для клиента и сервера средства тестирования и контроля качества;
- на обеих сторонах веб-приложения можно использовать общие шаблоны представлений;
- общий язык общения между группами, работающими над клиентской и серверной частью.

Node упрощает реализацию этих (и других) достоинств, предлагая основательную платформу и активное сообщество разработчиков.

Архитектура: потоки или асинхронный ввод/вывод с управлением по событиям

Говорят, что именно благодаря асинхронной событийно-ориентированной архитектуре Node демонстрирует столь высокую производительность. Так и есть, только к этому надо добавить еще стремительность движка V8 JavaScript. В тра-

диционной модели сервера приложений параллелизм обеспечивается за счет использования блокирующего ввода/вывода и нескольких потоков. Каждый поток должен дожидаться завершения ввода/вывода, перед тем как приступить к обработке следующего запроса.

В Node имеется единственный поток выполнения, без какого-либо контекстного переключения или ожидания ввода/вывода. При любом запросе ввода/вывода задаются функции обработки, которые впоследствии вызываются из цикла обработки событий, когда станут доступны данные или произойдет еще что-то значимое. Модель цикла обработки событий и обработчика событий – вещь распространенная, именно так исполняются написанные на JavaScript скрипты в браузере. Ожидается, что программа быстро вернет управление циклу обработки, чтобы можно было вызвать следующее стоящее в очереди задание.

Чтобы повернуть наши мысли в нужном направлении, Райан Дал (в презентации «*Cinco de Node*») спрашивает, что происходит при выполнении такого кода:

```
result = query('SELECT * from db');
```

Разумеется, программа в этой точке приостанавливается на время, пока слой доступа к базе данных отправляет запрос базе, которая вычисляет результат и возвращает данные. В зависимости от сложности запроса его выполнение может занять весьма заметное время. Это плохо, потому что пока поток простаивает, может прийти другой запрос, а если заняты все потоки (не забывайте, что ресурсы компьютера конечны), то запрос будет просто отброшен. Расточительно это както. Да и контекстное переключение обходится не бесплатно; чем больше запущено потоков, тем больше времени процессор тратит на сохранение и восстановление их состояния. Кроме того, стек каждого потока занимает место в памяти. И просто за счет асинхронного событийно-ориентированного ввода/вывода Node устраняет большую часть этих накладных расходов, привнося совсем немного собственных.

Часто рассказ о реализации параллелизма с помощью потоков сопровождается предостережениями типа «дорого и чревато ошибками», «ненадежные примитивы синхронизации в Java» или «проектирование параллельных программ может оказаться сложным и не исключены ошибки» (фразы взяты из результатов, выданных поисковой системой). Причиной этой сложности являются доступ к разделяемым переменным и различные стратегии предотвращения взаимоблокировок и состязаний между потоками. «Примитивы синхронизации в Java» – один из примеров такой стратегии, и, очевидно, многие программисты считают, что пользоваться ими трудно. Чтобы как-то скрыть сложность, присущую многопоточному параллелизму, создаются каркасы типа java.util.concurrent, но все равно некоторые считают, что попытка упрятать сложность подальше не делает проблему проще.

Node призывает подходить к параллелизму по-другому. Обратные вызовы из цикла обработки событий – гораздо более простая модель параллелизма как для понимания, так и для реализации.

Чтобы пояснить необходимость асинхронного ввода/вывода, Райан Дал напоминает об относительном времени доступа к объектам. Доступ к объектам в памя-

ти (порядка наносекунд) производится быстрее, чем к объектам на диске или по сети (миллисекунды или секунды). Время доступа к внешним объектам измеряется несметным количеством тактовых циклов и может оказаться вечностью, если ваш клиент, не дождавшись загрузки страницы в течение двух секунд, устанет пялиться на окно браузера и отправится в другое место.

В Node вышеупомянутый запрос следовало бы записать так:

```
query('SELECT * from db', function (result) {
  // произвести какие-то операции с результатом
});
```

Разница в том, что теперь результат запроса не возвращается в качестве значения функции, а передается функции обратного вызова, которая будет вызвана позже. Таким образом, возврат в цикл обработки событий происходит почти сразу, и сервер может перейти к обслуживанию других запросов. Одним из таких запросов будет ответ на запрос, отправленный базе данных, и тогда будет вызвана функция обратного вызова. Такая модель быстрого возврата в цикл обработки событий повышает степень использования ресурсов серверов. Это прекрасно для владельца сервера, но еще больший выигрыш получает пользователь, перед которым быстрее открывается содержимое страницы.

В наши дни веб-страницы все чаще собирают данные из десятков источников. Каждому нужно отправить запрос и дождаться ответа на него. Асинхронные запросы позволяют выполнять эти операции параллельно – отправить сразу все запросы, задав для каждого собственный обратный вызов, и, не дожидаясь ответа, вернуться в цикл обработки событий. А когда придет ответ, будет вызвана соответствующая ему функция. Благодаря распараллеливанию данные можно собрать гораздо быстрее, чем если бы запросы выполнялись синхронно, один за другим. И пользователь по ту сторону браузера счастлив, так как страница загружается быстрее.

Производительность и использование процессора

Своей притягательностью платформа Node отчасти обязана пропускной способности (количество обслуживаемых запросов в секунду). Сравнительные тесты схожих программ, например Apache и Node, показывают фантастический выигрыш в производительности.

Один из популярных эталонных тестов – следующий простой HTTP-сервер, который всего лишь возвращает сообщение «Hello World», читаемое из памяти:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
```

```
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

Это один из самых простых веб-серверов, которые только можно построить на платформе Node. Объект `http` инкапсулирует протокол HTTP, а его метод `http.createServer` создает полнофункциональный веб-сервер, прослушивающий порт, заданный в методе `.listen`. Каждый запрос (к любому URL любого вида – хоть GET, хоть PUT) приводит к вызову указанной функции. Сервер очень простой и совсем мало «весит». В данном случае вне зависимости от URL возвращается строка «**Hello World**» типа `text/plain`.

Благодаря такому минимализму это приложение должно демонстрировать максимальную пропускную способность. Поэтому во многих опубликованных исследованиях перечень эталонных тестов начинается с этого простейшего HTTP-сервера.

Райан Дал привел пример простого теста (`http://nodejs.org/cinco_de_node.pdf`), который возвращает 1 Мб двоичных данных; Node на нем показала пропускную способность 822 запроса/с, а nginx – 708 запросов/с. Дал также отвечает, что nginx достигала пиковой производительности при памяти объемом 4 Мб, а Node – при 64 Мб.

Дастин Маккуэй (Dustin McQuay) (`http://www.synchrosinteractive.com/blog/9-nodejs/22-nodejs-has-a-bright-future`) продемонстрировал две, по его словам, очень похожие программы на платформах Node и PHP/Apache:

- PHP/Apache 3187 запросов/с;
- Node.js 5569 запросов/с.

Ханнес Вальнёфер (Hannes Wallnöfer), автор RingoJS, написал в своем блоге заметку, в которой предостерегает от принятия важных решений на основе сравнительных тестов (`http://hns.github.com/2010/09/21/benchmark.html`), а затем переходит к сравнению RingoJS с Node. RingoJS – это сервер приложений, построенный на базе движка Rhino JavaScript для Java. При некоторых сценариях производительность RingoJS и Node довольно близка. Исследования показывают, что в приложениях, где требуется быстро выделять память для буферов или строк, Node работает хуже, чем RingoJS. В последующей заметке (`http://hns.github.com/2010/09/29/benchmark2.html`) Ханнес тестировал довольно типичную задачу разбора JSON-строки и обнаружил, что RingoJS работает гораздо быстрее.

Микито Такада (Mikito Takada) рассказал в блоге о сравнительном тестировании производительности Node и Django на примере написанного им приложения «48 hour hackathon» (`http://blog.mixu.net/2011/01/17/performance-benchmarking-the-node-js-backend-of-our-48h-product-wehearvoices-net/`). Неоптимизированная версия для Node оказалась чуть медленнее (по времени отклика), но несложная оптимизация (добавление пула соединений с MySQL, кэширования и т. д.) дала значительный прирост производительности, легко обогнав Django. Окончательный график показывает значение числа запросов в секунду, близкое к вышеупомянутому серверу «Hello World».

Чтобы в полной мере раскрыть потенциальные возможности Node, крайне важно быстро возвращать управление в цикл обработки событий. Мы вернемся

к этой теме в главе 4 «Вариации на тему простого приложения», но уже сейчас заметим, что если обработчик обратного вызова выполняется «слишком долго», то Node перестает быть тем сверхбыстрым сервером, каким был задуман. В одной из ранних статей о проекте Node (`http://four.livejournal.com/963421.html`) Райан Дал обсуждал требование о том, что обработчики событий должны выполняться не дольше 5 мс. Большая часть идей, высказанных в той статье, так и не была реализована, но Алекс Пэйн (Alex Payne) написал по этому поводу крайне любопытную статью (`http://al3x.net/2010/07/27/node.html`), в которой проводится различие между «масштабированием в малом» и «масштабированием в большом».

В случае небольших веб-приложений (*«масштабирование в малом»*) реализация на Node, а не на языках 'P' (Perl, PHP, Python и т. д.), должна давать выигрыш в производительности. JavaScript – мощный язык, а среда Node со своей современной быстрой виртуальной машиной имеет преимущества в части производительности и организации параллелизма по сравнению с интерпретируемыми языками типа PHP.

Далее Пэйн говорит, что *«масштабирование в большом»*, то есть создание приложений корпоративного уровня, всегда будет трудным и сложным делом. Чтобы обслужить огромное количество пользователей по всему миру, обеспечив при этом необходимую скорость загрузки страниц, обычно приходится включать в систему балансировщики нагрузки, кэширующие серверы, многочисленные резервные машины, расположенные в территориально разнесенных точках. Поэтому платформа разработки, наверное, не так важна, как система в целом.

Мы не узнаем, насколько хороша платформа Node в действительности, пока не увидим пример долгосрочного развертывания в крупных производственных средах.

Использование серверов, экономия затрат и экологичный Интернет

Смысл борьбы за максимальную эффективность (увеличение числа обрабатываемых запросов в секунду) – не только в том, чтобы получить удовольствие от хорошо проделанной технической работы. Имеются также реальные плюсы с точки зрения бизнеса и окружающей среды. Присущая Node способность обрабатывать больше запросов в секунду означает, что можно приобрести меньше серверов. То есть сделать больше меньшими средствами.

Грубо говоря, чем больше серверов, тем выше затраты и тем сильнее воздействие на окружающую среду, и наоборот. Существует целая наука о сокращении затрат и вреда окружающей среде, причиняемого инфраструктурой веб-серверов, и эта грубая рекомендация не может охватить всех деталей. Но общая цель очевидна – сократить количество серверов, снизить затраты и уменьшить ущерб, наносимый окружающей среде.

В опубликованной корпорацией Intel статье «Increasing Data Center Efficiency with Server Power Measurements» (`http://download.intel.com/it/pdf/Server_`

Power_Measurement_final.pdf) приводится методика оценки эффективности и стоимости центров обработки данных. Следует учитывать много факторов, в том числе конструкцию здания, систему охлаждения и проект вычислительной системы. Их эффективная реализация (эффективность ЦОД, плотность ЦОД и плотность СХД) может сократить затраты и вред окружающей среде. Но все это можно свести на нет, развернув неэффективную программную систему, которая заставляет приобретать больше серверов. И наоборот, эффективная программная система позволяет усилить преимущества эффективной организации ЦОД.

Как правильно: Node, Node.js или Node.JS?

Правильно платформа называется Node.js, но в этой книге мы пишем Node, следуя традиции, принятой на сайте nodejs.org, где говорится, что торговый знак – Node.js (js строчными буквами), но в опубликованных материалах употребляется название Node.

Резюме

В этой главе мы:

- узнали, что JavaScript живет не только внутри браузеров;
- поведали о различии между асинхронным и блокирующим вводом/выводом;
- получили первое представление о платформе Node;
- поговорили о производительности Node.

Теперь мы готовы приступить к практической работе с Node. В главе 2 «Настройка Node» мы поговорим о конфигурировании окружения Node. Готовьтесь!