

Содержание

Об авторах	11
О технических рецензентах	14
Предисловие	15
Глава 1. Добро пожаловать в платформу Node.js	21
Философия Node.js.....	21
Небольшое ядро.....	22
Небольшие модули.....	22
Небольшая общедоступная область.....	23
Простота и прагматизм.....	23
Введение в Node.js 6 и ES2015.....	24
Ключевые слова let и const.....	24
Стрелочные функции.....	26
Синтаксис классов.....	28
Расширенные литералы объектов.....	29
Коллекции Map и Set.....	30
Коллекции WeakMap и WeakSet.....	31
Литералы шаблонов.....	32
Другие особенности ES2015.....	33
Шаблон Reactor.....	33
Медленный ввод/вывод.....	33
Блокирующий ввод/вывод.....	34
Неблокирующий ввод/вывод.....	35
Демультимплексирование событий.....	36
Введение в шаблон Reactor.....	37
Неблокирующий движок libuv платформы Node.js.....	38
Рецепт платформы Node.js.....	39
Итоги.....	40
Глава 2. Основные шаблоны Node.js	41
Шаблон Callback.....	41
Стиль передачи продолжений.....	42
Синхронный или асинхронный?.....	44
Соглашения Node.js об обратных вызовах.....	48
Система модулей и ее шаблоны.....	51
Шаблон Revealing Module.....	51
Пояснения относительно модулей Node.js.....	52
Шаблоны определения модулей.....	58
Шаблон Observer.....	63
Класс EventEmitter.....	63
Создание и использование класса EventEmitter.....	64

Распространение ошибок	65
Создание произвольного наблюдаемого объекта	66
Синхронные и асинхронные события.....	67
Класс EventEmitter и обратные вызовы	68
Комбинирование EventEmitter и обратных вызовов.....	68
Итоги	69

Глава 3. Шаблоны асинхронного выполнения с обратными вызовами.....

с обратными вызовами.....	70
Сложности асинхронного программирования	70
Создание простого поискового робота	71
Ад обратных вызовов.....	72
Использование обычного JavaScript	73
Дисциплина обратных вызовов	74
Применение дисциплины обратных вызовов	74
Последовательное выполнение.....	76
Параллельное выполнение	80
Ограниченное параллельное выполнение.....	85
Библиотека async	88
Последовательное выполнение.....	89
Параллельное выполнение	91
Ограниченное параллельное выполнение.....	92
Итоги	93

Глава 4. Шаблоны асинхронного выполнения с использованием спецификации ES2015, и не только

с использованием спецификации ES2015, и не только	94
Promise	94
Что представляет собой объект Promise?	95
Реализации Promises/A+.....	97
Перевод функций в стиле Node.js на использование объектов Promise	98
Последовательное выполнение.....	99
Параллельное выполнение	101
Ограниченное параллельное выполнение.....	102
Обратные вызовы и объекты Promise в общедоступных программных интерфейсах	103
Генераторы.....	105
Введение в генераторы	105
Асинхронное выполнение с генераторами.....	108
Последовательное выполнение.....	110
Параллельное выполнение	112
Ограниченное параллельное выполнение.....	114
Async/await с использованием Babel	117
Установка и запуск Babel.....	118
Сравнение	119
Итоги	119

Глава 5. Программирование с применением потоков данных	121
Исследование важности потоков данных.....	121
Буферизация и потоковая передача данных	121
Эффективность с точки зрения памяти.....	122
Эффективность с точки зрения времени.....	124
Способность к объединению.....	126
Начало работы с потоками данных.....	127
Анатомия потоков данных	128
Потоки данных для чтения	128
Потоки данных для записи	132
Дуплексные потоки данных	135
Преобразующие потоки данных	136
Соединение потоков с помощью конвейеров.....	138
Управление асинхронным выполнением с помощью потоков данных.....	140
Последовательное выполнение.....	140
Неупорядоченное параллельное выполнение	142
Неупорядоченное ограниченное параллельное выполнение	145
Шаблоны конвейерной обработки.....	147
Объединение потоков данных.....	147
Ветвление потоков данных.....	150
Слияние потоков данных	151
Мультиплексирование и демультимплексирование	153
Итоги	158
Глава 6. Шаблоны проектирования	159
Фабрика.....	160
Универсальный интерфейс для создания объектов.....	160
Механизм принудительной инкапсуляции	161
Создание простого профилировщика кода	162
Составные фабричные функции.....	164
Реальное применение	167
Открытый конструктор	168
Генератор событий, доступный только для чтения	168
Реальное применение	169
Прокси	170
Приемы реализации прокси.....	171
Сравнение различных методов	172
Журналирование обращений к потоку для записи.....	173
Место прокси в экосистеме – ловушки для функций и АОП.....	174
Прокси в стандарте ES2015.....	174
Реальное применение	176
Декоратор.....	176
Приемы реализации декораторов	176
Декорирование базы данных LevelUP	177
Реальное применение	179

Адаптер	180
Использование LevelUP через интерфейс файловой системы.....	180
Реальное применение	183
Стратегия	183
Объекты для хранения конфигураций в нескольких форматах.....	184
Реальное применение	186
Состояние	187
Реализация простого сокета, защищенного от сбоев.....	188
Макет.....	191
Макет диспетчера конфигурации.....	192
Реальное применение	193
Промежуточное программное обеспечение.....	194
Промежуточное программное обеспечение в Express.....	194
Промежуточное программное обеспечение как шаблон.....	195
Создание фреймворка промежуточного программного обеспечения для ØMQ.....	196
Промежуточное программное обеспечение, использующее генераторы Кoa	201
Команда	204
Гибкость шаблона.....	205
Итоги	208
Глава 7. Связывание модулей.....	210
Модули и зависимости.....	211
Наиболее типичные зависимости в Node.js	211
Сцепленность и связанность.....	212
Модули с поддержкой состояния.....	212
Шаблоны связывания модулей.....	214
Жесткие зависимости.....	214
Внедрение зависимостей.....	218
Локаатор служб.....	222
Контейнер внедрения зависимостей	227
Связывание плагинов.....	230
Плагины как пакеты.....	230
Точки расширения	232
Расширение, управляемое плагинами и приложением	232
Реализация плагина выхода из системы.....	235
Итоги	242
Глава 8. Универсальный JavaScript для веб-приложений	243
Использование кода совместно с браузером	244
Совместное использование модулей.....	244
Введение в Webpack	248
Знакомство с волшебством Webpack	248
Преимущества использования Webpack.....	250
Использование ES2015 с помощью Webpack.....	250

Основы кросс-платформенной разработки	252
Ветвление кода во время выполнения.....	252
Ветвление кода в процессе сборки.....	253
Замена модулей	255
Шаблоны проектирования для кросс-платформенной разработки.....	257
Введение в React.....	258
Первый компонент React	259
Что такое JSX?!	260
Настройка Webpack для транскомпиляции JSX.....	262
Отображение в браузере.....	263
Библиотека React Router.....	264
Создание приложений на универсальном JavaScript.....	268
Создание многократно используемых компонентов.....	268
Отображение на стороне сервера	271
Универсальное отображение и маршрутизация.....	274
Универсальное извлечение данных	275
Итоги	282
Глава 9. Дополнительные рецепты асинхронной обработки.....	284
Подключение модулей, инициализируемых асинхронно.....	284
Канонические решения.....	285
Очереди на инициализацию.....	286
Реальное применение	289
Группировка асинхронных операций и кэширование.....	290
Реализация сервера без кэширования и группировки операций.....	290
Группировка асинхронных операций.....	292
Кэширование асинхронных запросов	294
Группировка и кэширование с использованием объектов Promise.....	297
Выполнение вычислительных заданий.....	299
Решение задачи выделения подмножеств с заданной суммой.....	299
Чередование с помощью функции setImmediate	302
Чередование этапов алгоритма извлечения подмножеств с заданной суммой.....	302
Использование нескольких процессов.....	304
Итоги	310
Глава 10. Шаблоны масштабирования и организации архитектуры	311
Введение в масштабирование приложений	312
Масштабирование приложений на платформе Node.js	312
Три измерения масштабируемости.....	312
Клонирование и распределение нагрузки.....	314
Модуль cluster	315
Взаимодействия с сохранением состояния.....	322
Масштабирование с помощью обратного проксирования	325
Использование реестра служб	328

Одноранговое распределение нагрузки	333
Декомпозиция сложных приложений	336
Монолитная архитектура	336
Архитектура на микрослужбах	338
Шаблоны интеграции в архитектуре на микрослужбах	341
Итоги	346
Глава 11. Шаблоны обмена сообщениями и интеграции	348
Введение в системы обмена сообщениями	349
Шаблоны однонаправленного обмена и вида «Запрос/ответ»	349
Типы сообщений	350
Асинхронный обмен сообщениями и очереди	351
Обмен сообщениями, прямой и через брокера	352
Шаблон «Публикация/подписка»	353
Минимальное приложение для общения в режиме реального времени	354
Использование Redis в качестве брокера сообщений	357
Прямая публикация/подписка с помощью библиотеки ØMQ	359
Надежная подписка	362
Шаблоны конвейеров и распределения заданий	369
Шаблон распределения/слияния в ØMQ	370
Конвейеры и конкурирующие потребители в AMQP	374
Шаблоны вида «Запрос/ответ»	378
Идентификатор корреляции	378
Обратный адрес	382
Итоги	386
Предметный указатель	387

Об авторах

Марио Каскиаро (Mario Casciaro) – программный инженер и предприниматель, увлекающийся технологиями, наукой и разработкой программного обеспечения с открытым исходным кодом. Получив степень магистра в области разработки программного обеспечения, Марио начал карьеру в компании IBM, где несколько лет занимался созданием различных корпоративных продуктов, таких как Tivoli Endpoint Manager, Cognos Insight и SalesConnect. Затем перешел в компанию D4H Technologies, специализирующуюся на SaaS, чтобы руководить разработкой нового важного проекта по управлению операциями в режиме реального времени. В настоящее время Марио является соучредителем и генеральным директором компании [Sponsorama.com](https://www.sponsorama.com), разрабатывающей, обеспечивающей корпоративное спонсорство развития интернет-проектов.

Марио является автором первого издания книги *Node.js Design Patterns*.

Благодарности

Работая над первым изданием этой книги, я и не предполагал, что она будет иметь такой успех. Я очень благодарен всем, кто прочел первое издание книги, приобрел его, оставил отзыв и рекомендовал его своим друзьям в Twitter и других интернет-форумах. И конечно же, я выражаю благодарность читателям второго издания, всем вам, кто читает его сейчас. Это значит, что наши усилия не пропали даром. Я также прошу вас присоединиться к моему дружескому поздравлению Лучано, соавтору второго издания, проделавшему огромную работу по обновлению и добавлению новых бесценных сведений в эту книгу. Все лавры по созданию этого издания принадлежат ему, потому что я выступал только в роли советчика. Работа над книгой – нелегкая задача, но Лучано удивил меня и всех сотрудников издательства Packt своей преданностью делу, профессионализмом и техническими навыками, демонстрируя способность достичь любой задуманной им цели. Работать вместе с Лучано было приятно и почетно, и я с нетерпением жду продолжения сотрудничества. Также хочу поблагодарить всех, кто работал над книгой, сотрудников издательства Packt, технических рецензентов – Тане (Tane) и Джоэл (Joel) – и всех друзей, внесших ценные предложения и идеи: Энтони Уолли (Anton Whalley, [@dhigit9](https://twitter.com/dhigit9)), Алессандро Синелли (Alessandro Cinelli, [@cirpo](https://twitter.com/cirpo)), Андреа Джулиано (Andrea Giuliano, [@bit_shark](https://twitter.com/bit_shark)) и Андреа Мангано (Andrea Mangano, [@ManganoAndrea](https://twitter.com/ManganoAndrea)). Спасибо всем дарящим мне свою любовь друзьям, моей семье и, самое главное, моей подруге Мириам, партнеру во всех моих приключениях, наполняющей любовью и радостью каждый день моей жизни. Нас еще ждут сотни тысяч новых приключений.

Лучано Маммино (Luciano Mammino) – инженер, родившийся в том же 1987 году, в котором компания Nintendo выпустила в Европе игру Super Mario Bros, совершенно случайно ставшую его любимой видеоигрой. Программировать он начал в возрасте 12 лет, на старом отцовском компьютере с процессором Intel 386, операционной системой DOS и интерпретатором qBasic.

После получения степени магистра в области компьютерных наук развивал навыки программирования в основном как веб-разработчик, работающий в качестве фрилансера с компаниями и стартапами, разбросанными по всей Италии. Будучи в течение трех лет техническим директором и соучредителем сайта Sbaam.com в Италии и Ирландии, он решил переехать в Дублин, где поступил на работу в компанию Smartbox старшим РНР-инженером.

Ему нравится разрабатывать библиотеки с открытым исходным кодом и работать на таких платформах, как Symfony и Express. Он убежден, что JavaScript все еще находится в самом начале своего славного пути, и в будущем его влияние на веб- и мобильные технологии будет только расти. По этой причине он проводит большую часть своего свободного времени, совершенствуя знания JavaScript и экспериментируя с Node.js.

Благодарности

Прежде всего хочу сказать огромное спасибо Марио за предоставленную мне возможность и оказанное доверие поработать вместе с ним над новым изданием этой книги. Это был интересный эксперимент, и я надеюсь, что он станет началом дальнейшего нашего сотрудничества.

Появление этой книги стало возможным только благодаря невероятно эффективной работе команды издательства Packt, в частности неустанным усилиям и терпению Онкара (Onkar), Решма (Reshma) и Прякта (Prajakta). А также благодаря помощи технических рецензентов Тана Пайпера (Tane Piper) и Джоэла Пурра (Joel Purra), их опыт работы с Node.js имел решающее значение для повышения качества этой книги.

Моя огромная благодарность (и море пива) моим друзьям Энтони Уолли (Anton Whalley, @dhigit9), Алессандро Синелли (Andrea Giuliano, @cirpo), Андреа Джулиано (Andrea Giuliano, @bit_shark) и Андреа Мангано (Andrea Mangano, @ManganoAndrea), поддерживавшим меня всю дорогу, делившимися со мной своим опытом и значительно повлиявшим на эту книгу.

Также большое спасибо хочу сказать Рикардо (Ricardo), Хосе (Jose), Альберто (Alberto), Марцин (Marcin), Начо (Nacho), Давиду (David), Артуру (Arthur) и всем моим коллегам из Smartbox за радость работы с ними и мою мотивацию к совершенствованию как программного инженера. Я не могу представить себе лучшую команду.

Выражаю глубочайшую благодарность моей семье, поддерживавшей меня все это время. Спасибо, мама, ты для меня постоянный источник вдохновения и силы. Спасибо, папа, за все уроки, похвалы и советы, я скучаю по нашим разговорам, я действительно скучаю по тебе. Спасибо моему брату Давиду и моей сестре Алессии, державшим меня в курсе грустных и радостных моментов, что позволяло мне чувствовать себя частью большой семьи.

Благодарю Франко и его семью за поддержку многих моих инициатив и за то, что они делятся со мной мудростью и жизненным опытом.

Спасибо моим «умным» друзьям Джанлуке (Gianluca), Флавио (Flavio), Антонио (Antonio), Валерио (Valerio) и Луке (Luca) за отлично проведенное вместе время и их призывы продолжать работу над этой книгой.

А также спасибо моим «менее умным» друзьям Дамиано (Damiano), Пьетро (Pietro) и Себастьяно (Sebastiano) за их дружбу, наш смех и радость при совместных прогулках по Дублину.

И напоследок, но не в последнюю очередь, спасибо моей подруге Франческе (Francesca). Спасибо за любовь и поддержку моих идей, даже сумасшедших. Я с нетерпением жду написания следующих страниц книги нашей совместной жизни.

О технических рецензентах

Тане Пайпер (Tane Piper) – опытный разработчик из Лондона (Великобритания). На протяжении более 10 лет работал в нескольких агентствах и компаниях, занимаясь разработкой программного обеспечения на разных языках, таких как Python, PHP и JavaScript. С платформой Node.js работает с 2010 года и был одним из тех, кто первым в Великобритании и Ирландии заговорил о применении JavaScript на стороне сервера в 2011–2012 годах. Также внес свой вклад в становление проекта jQuery.

В настоящее время работает консультантом по инновационным решениям в Лондоне, пишет главным образом React- и Node-приложения. В свободное время страстно увлекается дайвингом и фотографией.

Я хотел бы поблагодарить мою подругу Элину, изменившую последние два года моей жизни и вдохновившую меня на рецензирование этой книги.

Джоэл Пурра (Joel Purra) познакомился с компьютерами в раннем детстве, рассматривая их как еще один вид устройств для видеоигр. Некоторое время использовал компьютеры (иногда ломавшиеся и затем ремонтировавшиеся) только для запуска самых новых игр. Попытка изменить игру Lunar Lander заставила его в раннем подростковом возрасте начать программировать, вызвала интерес к созданию цифровых инструментов. Вскоре, после появления дома подключения к Интернету, разработал свой первый веб-сайт для электронной коммерции, начав таким образом собственный бизнес. Его карьера началась в раннем возрасте. В 17 лет Джоэл начал изучать компьютерное программирование в школе при атомной электростанции. После окончания школы поступил в военное училище в Швеции, где получил образование в области телекоммуникаций, а затем закончил университет Линчёпинга (Linköping), получив степень магистра информационных технологий и программной инженерии. Начиная с 1998 года участвовал в нескольких проектах разных компаний, как успешных, так и не очень, с 2007 года работает консультантом. Джоэл родился, вырос и получил образование в Швеции, но как внештатный разработчик в течение нескольких лет путешествовал по пяти континентам с рюкзаком и жил за границей. Постоянно учится решать новые задачи, одной из его целей является создание программного обеспечения для широкого общественного использования. Имеет свой веб-сайт <http://joelpurra.com/>.

Я хотел бы поблагодарить сообщество разработчиков открытого программного обеспечения за предоставление строительных блоков, так необходимых внештатным консультантам для создания малых и больших программных систем. Nanos gigantum humeris insidentes (Карлики на плечах гигантов). Помните, чтобы суметь, нужно пытаться!

Предисловие

Появление платформы Node.js многими рассматривается как смена правил игры – крупнейший за десятилетия сдвиг в области веб-разработки. Это определяется не только техническими возможностями платформы, но и парадигмой, вносимой ею в веб-разработку.

Во-первых, приложения на платформе Node.js написаны на языке Интернета – JavaScript, единственном языке программирования, изначально поддерживаемом большинством веб-браузеров. Эта его черта позволяет писать все компоненты приложения на одном языке и совместно использовать код клиентом и сервером. Платформа Node.js способствует развитию языка JavaScript. Разработчики, использующие JavaScript на сервере и в браузере, вскоре оценят его прагматизм и гибридный характер, определяемый промежуточным положением между объектно-ориентированным и функциональным программированием.

Вторым важным фактором является однопоточная асинхронная архитектура платформы. Помимо очевидных преимуществ с точки зрения производительности и масштабируемости, платформа меняет сам подход к реализации приемов параллельной обработки. Мьютексы заменяются очередями, потоки – обратными вызовами и событиями, а синхронизация – причинно-следственной связью.

И последний, но самый важный аспект платформы Node.js, заключается в ее экосистеме: диспетчер пакетов npm с постоянно растущей базой данных модулей, активным и дружелюбным сообществом, а самое главное, с собственной культурой, основанной на простоте, прагматизме и чрезвычайной модульности.

Но эти особенности разработки на платформе Node.js вызывают смешанные чувства, по сравнению с работой на других серверных платформах, и практически все, кто впервые сталкивается с этой парадигмой, чувствуют себя неуверенно при решении даже самых простых задач проектирования и написания кода. Постоянно возникают вопросы: «Как организовать код?», «Существует ли лучший способ решения?», «Как улучшить модульность приложения?», «Как эффективно обрабатывать набор асинхронных вызовов?», «Как можно удостовериться, что рост приложения не приведет к его краху?» или просто «Есть ли правильный способ сделать это?»

К счастью, Node.js – уже достаточно зрелая платформа, и на большинство этих вопросов можно легко ответить с помощью шаблонов проектирования, проверенных приемов программирования и рекомендуемых методик. Цель этой книги – познакомить вас с существующими шаблонами, методами и рекомендациями, предоставляющими проверенные временем решения часто возникающих проблем, и научить вас их использованию для решения конкретных задач.

Прочтя эту книгу, вы узнаете следующее:

- «подход Node»: рекомендуемая точка зрения на задачи проектирования при использовании платформы Node.js. Например, как выглядят на платформе Node.js различные традиционные шаблоны проектирования или как создавать модули, решающие только одну задачу;
- набор шаблонов для решения общих проблем проектирования при использовании платформы Node.js: вам будет предоставлено что-то вроде «швейцарского

армейского ножа» из готовых к использованию шаблонов, для эффективного решения повседневных задач разработки и проектирования;

- порядок написания эффективных модульных приложений для Node.js: понимание основных строительных блоков и принципов создания больших и хорошо организованных приложений для Node.js, обеспечивающее применение этих принципов к новым задачам, к которым не применимы существующие шаблоны.

В книге будет представлено несколько библиотек и технологий, таких как LevelDb, Redis, RabbitMQ, ZMQ, Express и многие другие. Они используются для демонстрации шаблонов или технологий, делая примеры более полезными, а также знакомят с экосистемой платформы Node.js и ее набором решений.

Если вы уже используете или планируете использовать платформу Node.js в работе над сторонним проектом или проектом с открытым исходным кодом, применение широко распространенных шаблонов и методов позволит вам быстро найти общий язык при совместном использовании кода и проектных решений, а также поможет понять будущее платформы Node.js и узнать, как внести свой вклад в ее развитие.

Какие темы охватывает книга

Глава 1 «Добро пожаловать в платформу Node.js» служит введением в проектирование приложений на платформе Node.js, знакомя с шаблонами, лежащими в основе самой платформы. Она охватывает экосистему и философию платформы Node.js, содержит краткое введение в версию Node.js 6, спецификацию ES2015 и шаблон «Реактор» (Reactor).

Глава 2 «Основные шаблоны Node.js» знакомит с основами асинхронного программирования и шаблонами проектирования Node.js, описывая и сравнивая обратные вызовы и генерацию событий – шаблон «Наблюдатель» (Observer). В этой главе также рассматриваются система модулей Node.js и соответствующий шаблон «Модуль» (Module).

Глава 3 «Шаблоны асинхронного выполнения с обратными вызовами» содержит набор шаблонов и методов эффективного выполнения асинхронных операций в Node.js. Эта глава знакомит со способами смягчения проблемы «ада обратных вызовов», основанными на обычном JavaScript и библиотеке `async`.

Глава 4 «Шаблоны асинхронного выполнения с использованием спецификации ES2015, и не только», продолжает исследование способов выполнения асинхронных операций с применением объектов `Promise`, генераторов и библиотеки `Async/Await`.

Глава 5 «Программирование с применением потоков данных» подробно рассматривает один из самых важных шаблонов Node.js: «Поток данных» (Stream). Знакомит с обработкой данных путем преобразования и объединения их потоков.

Глава 6 «Шаблоны проектирования» посвящена вызывающей споры теме традиционных шаблонов платформы Node.js. Охватывает наиболее популярные традиционные шаблоны проектирования и демонстрирует, насколько нетрадиционно они могут выглядеть в Node.js. Здесь также приводятся некоторые новые шаблоны проектирования, применимые только к JavaScript и Node.js.

Глава 7 «Связывание модулей» содержит анализ различных способов связывания модулей приложения в единое целое. Здесь представлены такие шаблоны проектирования, как «Внедрение зависимостей» (Dependency Injection) и «Локатор служб» (Service Locator).

Глава 8 «Универсальный JavaScript для веб-приложений» рассматривает одну из наиболее интересных черт современных веб-приложений на JavaScript – возможность совместного использования кода приложения клиентом и сервером. В этой главе демонстрируются основные принципы универсальности JavaScript путем создания простого веб-приложения с помощью React, Webpack и Babel.

Глава 9 «Дополнительные рецепты асинхронной обработки» предлагает подход, помогающий исключить несколько общих проблем разработки и проектирования с помощью готовых к использованию решений.

Глава 10 «Шаблоны масштабирования и организации архитектуры» описывает основные методы и шаблоны масштабирования приложений на платформе Node.js.

Глава 11 «Шаблоны обмена сообщениями и интеграции» посвящена наиболее важным шаблонам организации обмена сообщениями, создания и интегрирования сложных распределенных систем с использованием использующих ZMQ и AMQP.

Что потребуется для работы с книгой

Для работы с примерами, приведенными в книге, потребуется установить платформу Node.js версии 6 (или выше) и диспетчер пакетов npm версии 3 (или выше). Некоторые примеры требуют использования транскомпилятора, такого как Babel. Необходимы также навыки работы с командной строкой, умение установить npm и запустить приложение на Node.js. Также понадобятся текстовый редактор для работы с кодом и современный веб-браузер.

Кому адресована эта книга

Эта книга адресована разработчикам, уже знакомым с платформой Node.js и желающим получить максимальную отдачу от ее использования с точки зрения производительности, качества проектирования и масштабируемости. От вас требуется лишь понимание основ технологии, поскольку книга включает описание основных понятий. Разработчики со средним опытом работы в Node.js также найдут в книге полезные для себя методы.

Наличие знаний в области теории проектирования программного обеспечения станет преимуществом при изучении некоторых из представленных концепций.

Эта книга предполагает наличие у читателя знаний о разработке веб-приложений, языке JavaScript, веб-службах, базах и структурах данных.

Соглашения

В этой книге используется несколько разных стилей оформления текста для выделения разных видов информации. Ниже приведены примеры этих стилей с объяснением их назначения.

Программный код в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, фиктивные адреса URL, пользовательский ввод и ссылки в Twitter будут выглядеть так: «Спецификация ES2015 вводит ключевое слово `let` для объявления переменных, с областью видимости внутри блока».

Блоки программного кода оформляются так:

```
const zmq = require('zmq')
const sink = zmq.socket('pull');
sink.bindSync("tcp://*:5001");

sink.on('message', buffer => {
  console.log(`Message from worker: ${buffer.toString()}`);
});
```

Чтобы обратить ваше внимание на определенный фрагмент блока кода, соответствующие строки или элементы будут выделены жирным:

```
function produce() {
  //...
  variationsStream(alphabet, maxLength)
    .on('data', combination => {
      //...
      const msg = {searchHash: searchHash, variations: batch};
      channel.sendToQueue('jobs_queue', new Buffer(JSON.stringify(msg)));
      //...
    })
  //...
}
```

Ввод и вывод в командной строке будут выделены так:

```
node replier
node requestor
```

Новые термины и важные слова будут выделены жирным. Слова, которые выводятся на экран, например в меню или диалоговых окнах, будут оформляться так: «Для иллюстрации задачи создадим небольшое приложение **web spider**, запускаемое из командной строки и принимающее на входе URL-адрес, которое загружает страницу в локальный файл».



Предупреждения и важные сообщения будут выделены так.



Подсказки и советы будут выглядеть так.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут вам максимально полезны. Отзывы и пожелания можно посылать по адресу feedback@packtpub.com, указав название книги в теме письма. Если существует область, в которой вы хорошо разбираетесь, и у вас есть желание написать книгу, загляните в руководство для авторов: www.packtpub.com/authors.

Поддержка пользователей

Если вы – счастливый обладатель книги, изданной в Packt, мы готовы предоставить вам дополнительные услуги, чтобы ваша покупка принесла вам максимальную пользу.

Загрузка примеров исходного кода

С помощью своей учетной записи на <http://www.packtpub.com> вы сможете загрузить файлы с примерами кода для всех книг издательства Packt, купленных вами. Где бы вы ни купили эту книгу, вы сможете посетить страницу <http://www.packtpub.com/support> и зарегистрироваться для получения файлов по электронной почте.

Для скачивания файлов кода необходимо выполнить следующие действия:

- 1) войдите или зарегистрируйтесь на веб-сайте, указав свой адрес электронной почты и пароль;
- 2) наведите указатель мыши на вкладку **SUPPORT** (Поддержка) вверху;
- 3) щелкните на ссылке **Code Downloads & Errata** (Загрузка примеров кода и опечатки);
- 4) введите название книги в поле **Search** (Поиск);
- 5) выберите книгу, для которой хотите скачать файлы с исходным кодом;
- 6) выберите в раскрывающемся меню, где вы приобрели эту книгу;
- 7) щелкните на ссылке **Code Download** (Загрузить код), чтобы запустить загрузку.

После загрузки архива извлеките файлы с помощью последней версии одной из программ:

- WinRAR / 7-Zip для Windows;
- Zipreg / iZip / UnRarX для Mac;
- 7-zip / PeaZip для Linux.

Кроме того, пакет примеров кода к книге хранится в GitHub, и его можно найти на странице http://bit.ly/node_book_code. Пакеты с кодом примеров для других книг и видеоуроки можно найти на странице <https://github.com/PacktPublishing/>. Загляните туда!

Список опечаток

Мы приняли все возможные меры, чтобы гарантировать качество наших книг, но ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – в тексте или в коде, мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги. Если вы нашли опечатку, пожалуйста, сообщите о ней, для этого посетите страницу <http://www.packtpub.com/submit-errata>, выберите название книги, щелкните на ссылке **Errata Submission Form** (Форма отправки сообщения об ошибке) и опишите найденную ошибку. Как только ваше сообщение будет проверено, оно будет принято и добавлено в общий список.

Для просмотра общего списка замеченных опечаток перейдите на страницу <https://www.packtpub.com/books/content/support> и введите название книги в поле для поиска. Требуемая информация будет показана в разделе **Errata**.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательство Packt очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты copyright@packtpub.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Вопросы

Любые вопросы, касающиеся данной книги, вы можете присылать по адресу questions@packtpub.com. Мы постараемся решить возникшие проблемы.

Добро пожаловать в платформу Node.js

Некоторые принципы и шаблоны проектирования явным образом определяют действия разработчика при работе с платформой Node.js и ее экосистемой. Наиболее своеобразными из них являются: асинхронная обработка и стиль программирования, в значительной степени опирающийся на использование обратных вызовов. Начнем с подробного рассмотрения основополагающих принципов и закономерностей, предназначенных не только для создания корректного кода, но и для принятия эффективных решений, когда речь заходит о больших и сложных задачах.

Другим аспектом, характеризующим платформу Node.js, является ее философия. Освоение платформы Node.js на самом деле означает намного больше, чем просто изучение очередной новой технологии, оно включает ознакомление с ее культурой и сообществом. Здесь будет показано, как эти факторы влияют на способы разработки приложений и компонентов, а также порядок их взаимодействия с компонентами, созданными сообществом.

Помимо этих аспектов, следует учитывать, что в последних версиях платформы Node.js появилась поддержка многих функций, описанных в спецификации ES2015 (другое ее название ES6), делающих язык еще более выразительным и удобным в использовании. Важно освоить эти новые синтаксические и функциональные дополнения, чтобы писать более лаконичный и удобочитаемый код, а также разобраться в альтернативных подходах к реализации шаблонов проектирования, приведенных в этой книге.

В этой главе будут рассмотрены следующие вопросы:

- философия платформы Node.js, «подход Node»;
- версия 6 платформы Node.js и спецификация ES2015;
- шаблон «Реактор» (Reactor) – главный механизм асинхронной архитектуры Node.js.

Философия Node.js

Каждая платформа обладает собственной философией – набором принципов и рекомендаций, подготавливаемых сообществом, которые определяют идеологию развития платформы и порядок проектирования и разработки приложений. Некоторые из этих принципов определяются самой технологией, некоторые – экосистемой, другие берут начало в тенденциях сообщества или являются продуктом эволюции различ-

ных идеологий. Некоторые из принципов платформы Node.js заложены непосредственно ее создателем Райаном Далем (Ryan Dahl) и всеми теми, кто занимался ее ядром, другими яркими фигурами сообщества, а некоторые принципы унаследованы из культуры языка JavaScript или навеяны влиянием философии Unix.

Ни одно из этих правил не является догмой, их применение должно соотноситься со здравым смыслом, но знакомство с ними станет источником вдохновения при разработке собственных программ.



Обширный перечень различных философий разработки программного обеспечения можно найти в Википедии на странице http://en.wikipedia.org/wiki/List_of_software_development_philosophies.

Небольшое ядро

Ядро Node.js разработано в соответствии с несколькими принципами, один из которых заключается в предоставлении минимального набора функциональных возможностей, оставляя реализацию всех дополнительных возможностей за модулями экосистемы, не входящими в ядро. Этот принцип оказывает огромное влияние на платформу Node.js, поскольку дает сообществу полную свободу экспериментов в области применения пользовательских модулей, не навязывая какое-то одно, медленно развивающееся решение, встроенное в жестко контролируемое стабильное ядро. Сведение основного набора функций к минимуму удобно не только с точки зрения обслуживания, но и полезно с точки зрения позитивного воздействия на развитие всей экосистемы.

Небольшие модули

Платформа Node.js использует идею *модуля* как основное средство структурирования программного кода. Модули являются строительными блоками приложений и библиотек, часто называемых *пакетами* (термин «пакет» нередко используется взамен термина «модуль», потому что стало обычным делом, когда пакет состоит из единственного модуля). Один из самых активно продвигаемых принципов платформы Node.js заключается в разработке небольших модулей, не только с точки зрения размера, но и, что более важно, с точки зрения охватываемых возможностей.

Этот принцип исходит из философии Unix, в частности из двух следующих ее заповедей:

- чем меньше, тем лучше;
- любая программа должна делать что-то одно, но делать это хорошо.

Платформа Node.js возносит эти идеи на новый уровень. Официальный диспетчер пакетов платформы Node.js помогает решить проблему зависимостей, гарантируя каждому установленному пакету наличие собственного отдельного набора зависимостей, что позволяет программе использовать множество пакетов без появления конфликтов. Подход Node обеспечивает оптимальный уровень повторного использования кода, поскольку при его применении приложения состоят из большого числа малых, четко направленных зависимостей. Хотя это считается непрактичным или даже совершенно не осуществимым в других платформах, для разработки на платформе Node.js рекомендуется именно такая методика. Как следствие прм-пакеты нередко содержат менее 100 строк кода и предназначены для реализации только одной-единственной функции.

Кроме того, естественные преимущества многократно используемых небольших модулей заключаются в следующем:

- простота понимания и использования;
- легкость тестирования и поддержки;
- оптимальность при обмене с браузером.

Наличие небольших, узкоспециализированных модулей обеспечивает возможность совместного многократного использования фрагментов кода. Это позволяет поднять принцип «не повторяйся» (Don't Repeat Yourself, DRY) на совершенно новый уровень.

Небольшая общедоступная область

Помимо того что модули Node.js должны иметь небольшой размер и конкретную направленность, они обычно экспортируют минимальный набор функциональных возможностей. Основным преимуществом является повышение удобства и четкости программного интерфейса, что уменьшает вероятность неправильного использования. Как правило, пользователям компонента требуется весьма ограниченный, конкретный набор функций, а не расширение его функциональности или погружение в его особенности.

В платформе Node.js широко используется шаблон определения модулей, экспортирующих только одну функциональную возможность, например функцию или конструктор, при этом доступ к более сложным аспектам или дополнительным возможностям осуществляется через свойства, экспортируемые этой функцией или конструктором. Это помогает пользователю разделить важные и вторичные возможности. Часто модули экспортируют только одну функцию и ничего кроме нее, обеспечивая явную единственную точку входа.

Еще одной характерной чертой многих модулей платформы Node.js является их предназначение для использования, а не для расширения. Скрытие внутреннего содержания модулей, запрещающее любое его расширение, может показаться лишенным гибкости подходом, но он обеспечивает сокращение вариантов использования, упрощает реализацию, облегчает обслуживание и повышает удобство использования.

Простота и прагматизм

Вы когда-нибудь слышали о принципе KISS (Keep It Simple, Stupid – делай это проще, дурачок), запрещающем использование средств более сложных, чем это необходимо, или знаменитую цитату:

«*Все гениальное просто*» (Леонардо да Винчи).

Ричард П. Гэбриел (Richard P. Gabriel), выдающийся специалист в области вычислительной техники, придумал термин «чем хуже, тем лучше» для описания модели, где предпочтение отдается программному обеспечению минимального размера и с простой функциональностью. В своем эссе *The Rise of «Worse is Better»* он пишет:

«*Архитектура должна быть простой, это касается как реализации, так и интерфейса. Реализация должна быть проще интерфейса. Простота является наиболее важным фактором при разработке*».

Простота архитектуры, как противоположность богатого возможностями программного обеспечения, является хорошей методикой по следующим причинам: тре-

бует меньше усилий для реализации, ускоряет доставку при меньших затратах, облегчает адаптацию, поддержку и понимание. Эти факторы положительно влияют на активность сообщества, что позволяет наращивать и совершенствовать программное обеспечение.

Этот принцип платформы Node.js согласуется с применением JavaScript, который является весьма прагматичным языком. В нем часто используются простые функции, замыкания и литералы объектов вместо сложной иерархии классов. В чистом виде объектно-ориентированные конструкции обычно пытаются воспроизвести реальный мир с помощью математических терминов компьютерной системы, без учета несовершенства и сложности самого реального мира. В действительности программное обеспечение всегда лишь приблизительно отражает реальность и более успешными являются попытки добиться высокого быстродействия при разумной сложности, а не создание близкого к совершенству программного обеспечения посредством огромных усилий и непомерно больших объемов кода.

На протяжении данной книги этот принцип будет использован многократно. Например, значительное число традиционных шаблонов проектирования, таких как «Одиночка» (Singleton) или «Декоратор» (Decorator), могут иметь весьма тривиальную и не всегда самую надежную реализацию, но (как правило) практический, без излишних сложностей подход обеспечивает четкую и ясную архитектуру.

Введение в Node.js 6 и ES2015

На момент написания книги последние главные версии платформы Node.js (4, 5 и 6) включали расширенную поддержку новых особенностей из стандарта ECMAScript 2015 (ES2015, или ES6), направленную на то, чтобы сделать язык JavaScript еще более гибким и удобным.

Некоторые из этих новых особенностей широко будут применяться в примерах кода в данной книге. Их идеи достаточно новы для сообщества Node.js, поэтому стоит коротко остановиться на наиболее важных чертах стандарта ES2015, уже поддерживаемых платформой Node.js в настоящее время. Здесь подразумевается использование версии Node.js 6.

В зависимости от версии Node.js корректная работа некоторых из этих особенностей гарантирует включение **строгаго режима**. Для этого достаточно добавить в начало сценария оператор "use strict". Обратите внимание, что оператор "use strict" представляет собой обычную строку, и при его записи можно использовать одинарные или двойные кавычки. Для краткости в примерах кода эта строка будет опущена, но следует помнить, что ее необходимо добавлять для правильной работы примеров.

Приведенный ниже перечень не является полным, он служит просто введением в некоторые особенности ES2015, поддерживаемые платформой Node.js, и предназначен для облегчения понимания приведенных в книге примеров кода.

Ключевые слова `let` и `const`

Исторически сложилось, что язык JavaScript поддерживает только две области видимости – область видимости функции и глобальную область – для управления временем существования и видимостью переменных. Например, если объявить переменную в теле инструкции `if`, переменная станет доступна вне ее после выполнения тела. Для большей ясности рассмотрим пример:

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru