Оглавление

Часть II. Компоненты и данные в React

1 Что такое React

- Знакомство с React.
- Некоторые концепции и парадигмы React.
- Виртуальная объектная модель документа.
- Компоненты в React.
- React в командной работе.
- Компромиссы использования React.

Если вы работаете веб-разработчиком, то, скорее всего, слышали о React. Возможно, читали публикации в Интернете, например в Twitter или Reddit. Или друг (коллега) упомянул о библиотеке, или вы услышали о ней на конференции. Где бы и как это ни произошло, наверняка то, что вы слышали о React, было одобрительным или немного скептическим. Большинство людей хотят составить собственное мнение о таких технологиях, как React. Эффективные и удачные технологии, как правило, получают положительные отзывы. Зачастую немногие понимают эти технологии до того, как они выходят на более широкую аудиторию. Библиотека React тоже начинала с этого, но теперь пользуется огромной популярностью и широко распространена в мире веб-разработки. И ее известность заслуженна, так как она может многое предложить и позволяет внедрять, обновлять или даже трансформировать ваши идеи пользовательских интерфейсов.

1.1. Знакомство с React

React — это JavaScript-библиотека для создания пользовательских интерфейсов на разных платформах. Она предоставляет мощную ментальную модель для работы и помогает создавать декларативные и ориентированные на компоненты пользовательские интерфейсы. Это то, что библиотека React представляет собой в самом широком и общем смысле. Я раскрою эти идеи и многое другое по ходу книги.

Как библиотека React вписывается в более широкий мир веб-разработки? Часто говорят, что она стоит в одном ряду с такими проектами, как Vue, Preact, Angular, Ember, Webpack, Redux, и другими известными JavaScript-библиотеками

и фреймворками. React часто является важной частью клиентского приложения, и ее функции похожи на присущие библиотекам и фреймворкам, которые я только что упомянул. Фактически сейчас многие популярные интерфейсные технологии относятся к React в большей степени, чем в прошлом. Было время, когда принципы React были новинкой, но с тех пор часть технологий подпала под влияние инновационного подхода, основанного на ее компонентах. Библиотека продолжает поддерживать дух переосмысления установленных лучших практик, главная цель которого — предоставить разработчикам выразительную ментальную модель и эффективную технологию для создания приложений пользовательского интерфейса.

Почему ментальная модель React такая мощная? Она опирается на глубокие области информатики и техники разработки программного обеспечения. Ментальная модель React широко использует функциональные, а также объектно-ориентированные концепции программирования и фокусируется на компонентах как единой основе для разработки. В React-приложениях вы создаете из них интерфейсы. Система рендеринга React управляет ими и автоматически синхронизирует представление приложения. Компоненты часто соответствуют элементам пользовательского интерфейса, например календарям, заголовкам, панелям навигации и т. п., они также могут отвечать за маршрутизацию на стороне клиента, форматирование данных, стилизацию и другие функции клиентского приложения.

Компоненты в React должны быть такими, чтобы их можно было легко понять и интегрировать с другими компонентами. Они должны развиваться в соответствии с предсказуемым жизненным циклом, поддерживать собственное внутреннее состояние и работать со старым добрым JavaScript. В дальнейшем мы изучим эти идеи, а сейчас рассмотрим их вкратце. Обзор основных «ингредиентов», которые входят в React-приложение, представляет рис. 1.1. Кратко рассмотрим их.

- *Компоненты*. Это инкапсулированные блоки функциональности, которые являются основой в React. Они используют данные (*свойства* и *состояние*) для рендеринга пользовательских интерфейсов (мы рассмотрим, как компоненты React работают с данными, в главе 2 и далее). Некоторые их типы также предоставляют набор методов жизненного цикла, которые вы можете перехватывать (hook), иначе — *перехватчики жизненного цикла*. *Процесс рендеринга* (вывод и обновление пользовательского интерфейса на основе ваших данных) в React предсказуем, и компоненты могут подключиться к нему через API React.
- *Библиотеки React*. React содержит набор основных библиотек. Основная библиотека React работает с интерактивными библиотеками `react-dom` и `react-native` и ориентирована на спецификацию и определение компонентов. Она позволяет создавать дерево компонентов, которые можно использовать для средств визуализации (рендеринга) браузера или другой платформы. `react-dom` — одно из таких средств, предназначенное для рендеринга в браузере и на стороне сервера. Библиотеки React Native сфокусированы на нативных платформах и позволяют создавать React-приложения для iOS, Android и других платформ.

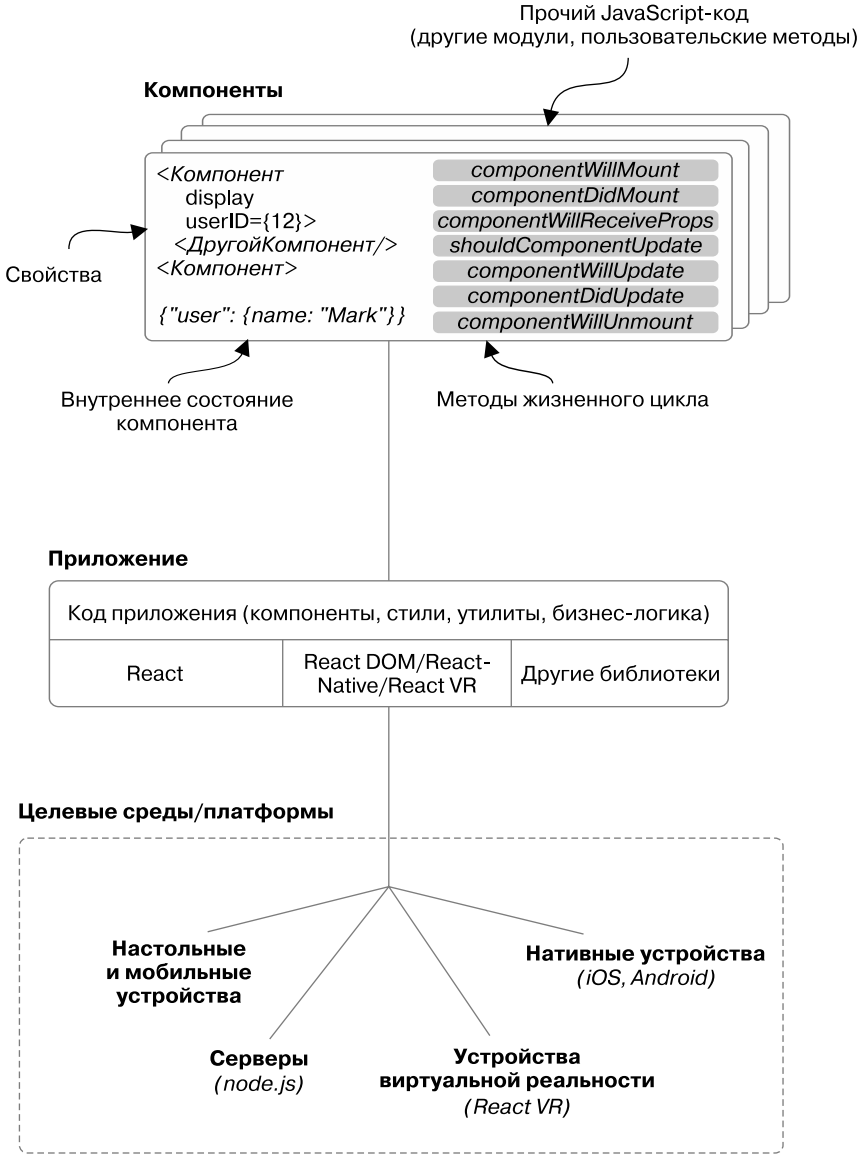

Рис. 1.1. React позволяет создавать пользовательские интерфейсы из компонентов. Последние поддерживают свое собственное состояние, написаны и работают с «ванильным» JavaScript-кодом и наследуют ряд полезных API от React. Большинство React-приложений разработаны для браузеров, но могут использоваться и в таких средах, как iOS и Android. Для получения дополнительной информации о React Native см. книгу React Native in Action, выпущенную издательством Manning

- *Сторонние библиотеки*. React не поставляется с инструментами моделирования данных, HTTP-вызовов, библиотеками стилей или другими распространенными элементами интерфейсного приложения. Поэтому задействуйте в своем приложении дополнительный код, модули или другие инструменты. И хотя этими общими технологиями React не комплектуется, широкая экосистема, окружающая ее, состоит из невероятно полезных библиотек. Далее будем использовать некоторые из них и даже отведем главы 10 и 11 на то, чтобы изучить Redux — библиотеку для управления состоянием.
- *Запуск React-приложения*. React-приложение запускается на платформе, для которой было разработано. В книге мы сосредоточимся на веб-платформе и создадим браузерное и серверное приложения, но другие проекты, такие как React Native и React VR, дают возможность запускать приложения на других платформах.

В этой книге мы потратим много времени на изучение всех закоулков библиотеки React, но перед началом работы у вас может возникнуть несколько вопросов. Подойдет ли вам библиотека? Кто еще применяет ее? Каковы преимущества использования React или отказа от нее? Это лишь часть важных вопросов, на которые следует получить ответы, прежде чем начать изучать React.

1.1.1. Для кого эта книга

Книга предназначена тем, кто разрабатывает пользовательские интерфейсы или планирует начать делать это. А также тем, кто интересуется библиотекой React, но непосредственно не занимается разработкой пользовательских интерфейсов. Вы получите максимальную отдачу от книги, если у вас есть опыт работы с JavaScript и создания интерфейсных приложений с его помощью.

Вы легко научитесь создавать React-приложения, если знакомы с основами языка JavaScript и имеете некоторый опыт разработки веб-приложений. Я не буду освещать основы JavaScript. Такие темы, как прототипная модель наследования, стандарт ES2015 и выше, приведение типов, синтаксис, ключевые слова, паттерны асинхронного программирования, такие как async/await, и другие фундаментальные концепции выходят за рамки этой книги. Я просто рассмотрю все, что особенно важно для разработки React-приложений, но не стану глубоко погружаться в язык программирования JavaScript.

Это не значит, что у вас не получится изучить React или вы ничего не поймете в книге, если не знаете языка JavaScript. Но вы получите гораздо больше, если предварительно уделите время его изучению. Продолжать обучение без знания этого языка может оказаться сложно. Вероятно, вы не во всем сразу разберетесь: код работает, но вы не понимаете как. Это неправильный подход, он не поможет вам как разработчику, поэтому последнее предупреждение: прежде чем изучать React, освойте основы JavaScript. Это замечательный, выразительный и гибкий язык, он вам понравится!

Возможно, вы уже хорошо знаете язык JavaScript и даже имели дело с React. Это не удивляет, учитывая популярность библиотеки. Если это так, вы лучше поймете некоторые основные концепции React. Тем не менее я не буду рассматривать

ряд узких тем, с которыми вы могли столкнуться, работая с библиотекой. Изучить их вы можете, прочитав другие книги издательства Manning, например *React Native in Action*.

Даже если вы не относитесь ни к одной из этих групп, а просто хотите получить краткий обзор библиотеки React, то эта книга и для вас. Вы изучите фундаментальные концепции React и получите доступ к демонстрационному React-приложению. Сможете проверить готовое приложение на странице social.react.sh. Увидите, как основные концепции React-приложения реализуются на практике и как вы или ваша команда можете работать с ними в следующем проекте.

1.1.2. Примечание об инструментарии

Если последние несколько лет вы много работали над клиентскими приложениями, то не удивитесь тому, что инструментарий для приложений стал такой же частью процесса разработки, как и сами фреймворки и библиотеки. Вероятно, вы используете для разработки своих приложений инструментарий типа Webpack, Babel и пр. Как эти и другие инструменты вписываются в эту книгу и что нужно о них знать?

Вам не нужно быть гуру Webpack, Babel и других инструментов, чтобы читать и понимать эту книгу. При создании демонстрационного приложения я применил несколько важных инструментов, и вы сможете просмотреть их конфигурации. Но я не буду подробно описывать их. Инструменты быстро обновляются и изменяются, поэтому их бесполезно глубоко рассматривать в рамках книги. Я обязательно отмечу, если инструменты актуальны или подходят для работы над проектом, но не буду их описывать.

А еще я думаю, что подобный инструментарий способен отвлечь при изучении новых технологий наподобие React. Если вам нужно обдумать несколько новых концепций и парадигм, зачем еще изучать сложный инструментарий? Именно поэтому сначала мы сосредоточимся на изучении «ванильного» React в главе 2, прежде чем переходить к возможностям JSX или JavaScript, реализация которых требует применения инструментов разработки. Но что стоит знать, так это npm. Это инструмент управления пакетами для JavaScript, и мы будем использовать его для установки зависимостей для нашего проекта и выполнения команд для работы с ним в оболочке командной строки. Вероятно, вы уже знакомы с менеджером пакетов npm, но если нет, это не помешает вам читать книгу. Потребуются лишь базовые навыки работы в оболочке командной строки и с менеджером npm. Справочные сведения, касающиеся менеджера пакетов npm, опубликованы на сайте docs.npmjs.com/getting-started/what-is-npm.

1.1.3. Кто использует React

Когда дело доходит до программного обеспечения с открытым исходным кодом, то кто ею пользуется (или не пользуется) — это больше чем просто вопрос популярности. На ответ влияет опыт работы с технологией (включая наличие поддержки, документации, исправлений безопасности), уровень инноваций в проекте и потенциальный жизненный цикл инструментария. Как правило, интереснее, проще

и удобнее работать с инструментами, которые поддерживаются, имеют надежную экосистему и учитывают опыт разработчиков и пользователей.

Библиотека React стартовала как небольшой проект, а теперь имеет широкую популярность и поддержку. Не существует совершенных проектов, и React не исключение, но, поскольку проекты с открытым исходным кодом, как правило, развиваются эффективнее прочих, у нее есть много важных компонентов для достижения успеха. Более того, проект React включает в себя множество других технологий с открытым исходным кодом. Это кажется сложным, поскольку экосистема может стать огромной, но так она формирует надежное и разнообразное сообщество. На рис. 1.2 показана карта экосистемы React. Я буду рассказывать о различных библиотеках и проектах на протяжении всей книги, но если вам интересно узнать об этом больше, вы найдете руководство по адресу ifelse.io/react-ecosystem. Я буду поддерживать и обновлять ресурс по мере развития экосистемы.

Рис. 1.2. Карта экосистемы React разноплановая настолько, что даже не вписывается в мои представления. Если вы хотите получить дополнительную информацию, я опубликовал на сайте ifelse.io/react-ecosystem руководство, которое поможет определить свой путь в экосистеме React

Основной способ прикоснуться к React — это приложения с открытым исходным кодом, которые вы используете ежедневно. Существует много компаний, работающих с React разными интересными способами. Далее перечислены лишь некоторые, применяющие React для своих продуктов: Facebook, Netflix, New Relic, Uber, Wealthfront, Heroku, PayPal, BBC, Microsoft, NFL, Asana, ESPN, Walmart, Venmo, Codecademy, Atlassian, Asana, Airbnb, Khan Academy, FloQast и др.

Эти компании не следуют слепо тенденциям в сообществе JavaScript. У них есть исключительные технические требования, которые влияют на огромное количество пользователей, и им необходимо выполнять разработку в жесткие сроки. Кто-то скажет: «Я слышал, что React — хорошая тема, поэтому мы должны все-таки “реактифицироваться”!» — и не будет прислушиваться к советам менеджеров и других разработчиков. Компаниям и разработчикам нужны хорошие инструменты, которые помогают удачно реализовывать идеи и делать это быстро, создавая высокопроизводительные, масштабируемые и надежные приложения.

1.2. Чего React не делает

До сих пор мы говорили о React в общем — кто ее использует, для кого эта книга и т. д. Мои главные цели при написании книги — научить вас создавать React-приложения и сформировать из вас веб-разработчика. React неидеальна, но с ней по-настоящему приятно работать, и я видел, как команды программистов творят с ее помощью большие дела. Мне нравится писать о React, работать с нею, обсуждать на конференциях, а иногда и вести энергичные дебаты по тому или иному паттерну.

Но я бы оказал вам плохую услугу, если бы умолчал о некоторых недостатках React и не описал, на что библиотека не способна. Понимание того, что что-то нереализуемо, так же важно, как осознание того, чего можно достичь. Почему? Лучшие технические решения обычно результат компромиссов, а не безапелляционного заявления: «React кардинально лучше, чем инструмент X, потому что нравится мне больше». По поводу последнего выражения: вы, вероятно, имеете дело не с двумя совершенно разными технологиями (COBOL против JavaScript). Надеюсь, вы даже не рассматриваете технологии, которые принципиально не подходят для решения этой задачи. И еще: создание больших проектов и решение технических задач никогда не должны зависеть от чьего-то мнения. Дело не в том, что мнения людей не имеют значения (конечно, это не так), просто они не должны влиять на работу.

Компромиссы. Итак, компромиссы необходимы при оценке и обсуждении программного обеспечения. На какие же компромиссы можно пойти относительно React? Во-первых, библиотеку иногда называют *простым представлением*. Это может быть неверно истолковано, если вы подумаете, что React — просто система паттернов типа Handlebars или Pug (née Jade) или что эта библиотека — часть архитектуры MCV (модель — контроллер — представление). Все эти утверждения неверны. Библиотека React не только сочетает оба определения, она — нечто намного большее. Для упрощения опишу React как ответ на вопрос «Что это такое?», а не «Чем она не является?» (например, просто представлением). React — это *декларативная*,

основанная на компонентах библиотека для создания пользовательских интерфейсов, поддерживаемых на различных платформах (Всемирная паутина, нативные устройства, мобильные устройства, серверная платформа, настольные устройства и даже устройства виртуальной реальности в будущем (React VR)).

Это приводит нас к первому компромиссу: React в первую очередь касается особенностей *представления* пользовательского интерфейса. Это означает, что она создавалась не для выполнения многих задач более сложных фреймворка или библиотеки. Сравните с технологией типа Angular. В своем последнем релизе Angular имеет гораздо больше общего с React, чем раньше, с точки зрения концепций и строения, но и предоставляет гораздо больше функций, чем React. Angular включает в себя следующие решения:

- HTTP-вызовы;
- создание и проверку веб-форм;
- маршрутизацию;
- форматирование строк и чисел;
- интернационализацию;
- внедрение зависимостей;
- базовые элементы для моделирования данных;
- настраиваемый фреймворк тестирования (хотя это и не так важно, как другие возможности);
- служебные скрипты (SW) включены по умолчанию (подход с поддержкой обслуживания к выполнению JavaScript-сценариев).

Это очень много, и, судя по моему опыту, пользователи, как правило, реагируют на все эти функции во фреймворке двумя способами. Либо «Ничего себе! Мне не нужны все эти функции!», либо «Ничего себе! Я не могу выбрать, как мне что-то сделать!». Суть таких фреймворков, как Angular, Ember и т. п., состоит в том, что обычно существует четко определенный способ сделать что-то. Например, маршрутизация в Angular реализуется с помощью встроенного роутера Angular Router, все задачи по протоколу HTTP выполняются с помощью встроенных HTTP-процедур и т. д.

Нет ничего принципиально неправильного в этом подходе. Я трудился в таких командах, где использовали подобные технологии, и в таких, где пошли более гибким путем и выбирали технологии, которые хорошо помогают сделать что-то конкретное. Мы отлично поработали с обоими подходами, они хорошо зарекомендовали себя. Я предпочитаю принцип «одно, но качественно», но тут важен компромисс. В библиотеке React нет непосредственных решений для HTTP, маршрутизации, моделирования данных (хотя наверняка есть понятие о потоке данных в ваших представлениях, к которым мы стремимся) или другого функционала, схожего с имеющимся у платформ наподобие Angular. Если ваша команда считает, что в ходе работы над проектом одним фреймворком не обойтись, возможно, React не лучший выбор. Но, по моему опыту, большинство команд одобряют гибкость React, сочетающуюся с ментальной моделью и интуитивно понятными API.

Гибкость React, в частности, заключается в том, что вы можете выбрать наилучшие инструменты для работы. Не нравится, как работает HTTP-библиотека X? Нет проблем, замените ее чем-то другим. Предпочитаете работать с веб-формами иначе? В чем дело, реализуйте свой способ. React предоставляет набор мощных элементов для работы. Справедливости ради стоит сказать: другие фреймворки, типа Angular, как правило, также позволяют заменить что-то, но реально поддерживаемый сообществом способ выполнить поставленную задачу обычно будет встроенным и задействованным.

Очевидный недостаток большей свободы заключается в том, что, если вы привыкли к фреймворкам с более широкими возможностями, например Angular или Ember, вам придется подбирать собственные решения для разных функциональных возможностей своего приложения. Это или хорошо, или плохо в зависимости от таких факторов, как уровень подготовки разработчиков в команде, предпочтения в области управления проектами и другие факторы, характерные для вашей ситуации. Существует много аргументов в пользу как универсального подхода, так и подхода в духе «одно, но качественно». Я склонен придерживаться принципа, который позволяет в каждом конкретном случае принимать гибкие решения таким образом, чтобы команда могла подобрать или создать нужные инструменты. Можно также изучить невероятно большую экосистему JavaScript — вам будет *трудно* найти что-то относящееся к задаче, которую вы решаете. Но факт остается фактом: отличные высокоэффективные команды используют оба подхода (иногда одновременно!), чтобы достичь результата.

Было бы неправильно не упомянуть об особенности, прежде чем рассказывать дальше. Фреймворки JavaScript редко совместимы между собой — это факт. Вряд ли вы создадите приложение, включающее функционал Angular, Ember, Backbone и React, по крайней мере без сегментирования каждой части или жесткого контроля за их взаимодействием. Если можно избежать такой ситуации, нужно так и сделать. Таким образом, разрабатывая приложение, вы обычно работаете с одним или двумя основными фреймворками.

Но что произойдет, если нужно внести в приложение некие изменения? Если вы применяете инструмент с широкими возможностями, например Angular, миграция приложения, скорее всего, приведет к полной переработке кода из-за глубокой идиоматической интеграции фреймворка. Вы можете переписать небольшие части кода приложения, однако, просто заменив несколько функций, не стоит ожидать, что приложение продолжит работать. Вот где проявляется преимущество React. Его использование не означает, что миграция пройдет безболезненно, но вы существенно снизите затраты по сравнению с возникающими при реализации решений с плотно интегрированным фреймворком типа Angular.

Еще один компромисс, на который вы подписываетесь, выбирая React, заключается в том, что библиотека разработана и собрана программистами прежде всего из компании Facebook и предназначена для удовлетворения потребностей пользователей сети Facebook. Вероятно, вам будет нелегко работать с React, если ваше приложение принципиально отличается от того, что нужно этой аудитории. К счастью, большинство современных веб-приложений можно разработать с помощью

React, но, безусловно, есть и такие, для которых она не годится. К ним относятся приложения, которые работают не в традиционных парадигмах пользовательского интерфейса современных веб-приложений или имеют весьма специфические требования к производительности (например, высокоскоростной биржевой тикер). Даже эти проблемы часто решаются с помощью React, но есть ситуации, требующие более специфических технологий.

Последний компромисс, который мы должны обсудить, — это реализация и строение React. Ядро React — это системы, автоматически обрабатывающие пользовательский интерфейс, когда изменяются данные в его компонентах. Они вносят изменения, которые могут перехватить, используя так называемые *методы жизненного цикла*. Мы подробно рассмотрим их в последующих главах. Системы React, поддерживающие обновление пользовательского интерфейса, значительно упрощают создание надежных модульных компонентов, применяемых приложением. То, как библиотека React выполняет бóльшую часть работы по поддержке пользовательского интерфейса с обновлением данных, — еще одна причина того, что разработчикам нравится работать с React и этот мощный инструмент — в ваших руках. Тем не менее не следует полагать, что у «движков» данной технологии нет недостатков или они не используют компромиссы.

React — это абстракция, что чревато затратами. Создаваемая вами система непрозрачна, потому что построена особым образом и управляется через API. Это означает также, что вам нужно разрабатывать пользовательские интерфейсы в режиме, присущем React. К счастью, API React создают «обусловленные выходы», которые позволяют вам спуститься на более низкие уровни абстракции. Вы все равно можете использовать другие инструменты, такие как jQuery, но делать это стоит с оглядкой на React. Это опять-таки компромисс: упрощенная ментальная модель в обмен на то, что у вас не получится делать абсолютно все, что хочется.

Вы не только теряете видимость в базовой системе, но и доверяете действиям React. Это может повлиять на более узкий срез вашего стека приложений (только представления вместо данных, специальные системы генерации веб-форм, моделирование данных и т. д.) и тем не менее имеет значение. Я надеюсь, что вы оцените преимущества React и увидите, что они существенно превышают стоимость обучения и что компромиссы, на которые вы идете при ее использовании, в целом обеспечивают вам гораздо лучшие позиции разработчика. Но я был бы неискренним, если бы сделал вид, что React волшебным образом решит все ваши технические задачи.

1.3. Виртуальная объектная модель документа в React

Мы вкратце поговорили о некоторых возможностях библиотеки React. Я полагаю, что эта информация поможет вам и вашей команде создавать лучшие пользовательские интерфейсы. Частично она связана с ментальной моделью и API, которые предоставляет React. Что скрывается за этим? Основная концепция React — это стремление упростить задачи и исключить ненужную сложность при разработке.

Библиотека React пытается сделать все возможное, чтобы быть исполнителем, освобождая вас для того, чтобы вы могли задуматься о других функциях приложения. Этот подход заключается в том, чтобы позволить вам быть *декларативным*, а не *императивным* разработчиком. Вы объявляете, как компоненты должны вести себя и выглядеть в разных состояниях, а внутренний механизм React справляется со сложностями управления обновлениями, апгрейдом пользовательского интерфейса для отражения изменений и т. д.

Одним из основных управляющих элементов технологии является виртуальная объектная модель документа (*виртуальная DOM*, или *vDOM*). Она представляет собой структуру данных или набор структур данных, имитирующих или отражающих объектную модель документа, которая используется в браузерах. Я говорю «виртуальная объектная модель документа», потому что другие фреймворки, такие как Ember, имеют собственную реализацию подобной технологии. В целом, виртуальная DOM служит промежуточным слоем между кодом приложения и фактической DOM браузера. Как правило, виртуальная DOM позволяет скрыть сложность обнаружения изменений и управления от разработки и перейти к специализированному уровню абстракции. В следующих подразделах мы посмотрим, как этот подход реализуется в React. На рис. 1.3 показана упрощенная схема фактической DOM и ее отношений с виртуальной DOM, которые мы рассмотрим в ближайшее время.

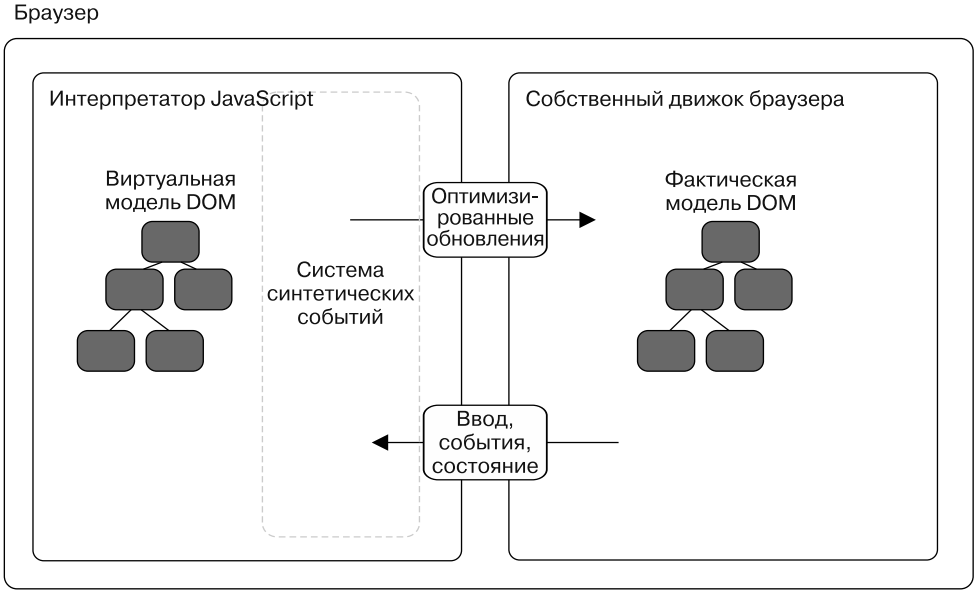

Рис. 1.3. Фактическая и виртуальная DOM. Виртуальная DOM в React обрабатывает изменения в данных, а также преобразование событий браузера в события, которые компоненты могут воспринять и на которые могут среагировать. Виртуальная DOM в React нацелена также на оптимизацию изменений, внесенных в DOM ради производительности