

O'REILLY®

2-е издание



React

Быстрый старт

Создаем веб-приложения



Стоян Стефанов

ББК 32.988.02-018
УДК 004.738.5
С79

Стефанов С.

С79 React. Быстрый старт, 2-е изд. — СПб.: Питер, 2023. — 304 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-4461-2115-1

Незаменимая книга по React — технологии с открытым исходным кодом для быстрого создания многофункциональных веб-приложений. Второе издание, обновленное с учетом последней версии React, показывает, как создавать компоненты React и организовывать их в удобные для сопровождения крупномасштабные приложения. Если вы знаете синтаксис JavaScript, то сразу можете приступить к работе.

По ходу чтения разработчики и программисты создадут полноценное приложение. Вы быстро поймете, почему многие выбирают React для разработки веб-приложений.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988.02-018
УДК 004.738.5

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др.

Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1492051466 англ.

Authorized Russian translation of the English edition of React: Up & Running 2E,

ISBN 9781492051466

© 2022 Stoyan Stefanov. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-5-4461-2115-1

© Перевод на русский язык ООО «Прогресс книга», 2023

© Издание на русском языке, оформление ООО «Прогресс книга», 2023

© Серия «Бестселлеры O'Reilly», 2023

Оглавление

Предисловие	16
Об этой книге.....	18
Условные обозначения.....	19
Использование примеров кода.....	19
Благодарности	20
От издательства	22
Глава 1. Hello World	23
Установка.....	23
Hello React World	24
Так что же сейчас произошло?	26
Метод React.createElement()	28
Синтаксис JSX	30
Установка Babel	31
Привет, мир JSX	32
О транспиляции	33
Далее: настраиваемые компоненты.....	35

Глава 2. Жизнь компонента	36
Настраиваемый функциональный компонент.....	36
JSX-версия	37
Настраиваемый классовый компонент	38
Какой синтаксис использовать	39
Свойства	40
Свойства в функциональных компонентах	42
Свойства по умолчанию	43
Состояние	44
Компонент текстовой области	45
Компонент <code>textarea</code> , отслеживающий свое состояние.....	47
Немного о DOM-событиях	50
Обработка событий в прежние времена	50
Обработка событий в React.....	52
Синтаксис обработки событий	53
Сравнение свойств и состояния.....	55
Свойства в исходном состоянии: антипаттерн	55
Доступ к компоненту извне	56
Методы управления жизненным циклом	58
Примеры управления жизненным циклом: тотальная регистрация	59
Параноидальная защита состояния.....	62
Пример жизненного цикла: применение дочернего компонента	63

Выигрыш в производительности: предотвращение обновлений компонентов	66
Что случилось с функциональными компонентами.....	68
Глава 3. Excel: причудливый компонент таблицы.....	69
Начнем с данных.....	69
Цикл создания заголовков таблицы.....	70
Цикл заголовков таблиц, упрощенная версия	72
Отладка для избавления от предупреждения в консоли.....	75
Добавление содержимого <td>.....	76
Типы свойств.....	79
Можете ли вы улучшить компонент.....	81
Сортировка	82
Можете ли вы улучшить компонент.....	85
Сортировка подсказок пользовательского интерфейса	85
Редактирование данных	87
Редактируемая ячейка	88
Поле ввода ячейки	91
Сохранение	91
Выводы и определение различий в виртуальной DOM.....	92
Поиск.....	94
Состояние и пользовательский интерфейс	96
Фильтрация содержимого.....	99

Обновление метода save()	102
Как усовершенствовать поиск.....	103
Мгновенное воспроизведение	103
Очистка обработчиков событий.....	106
Решение задачи очистки	108
Как усовершенствовать воспроизведение	109
Возможна ли альтернативная реализация	110
Скачивание данных таблицы	110
Получение данных.....	113
Глава 4. Функциональный Excel	115
Кратко освежим знания: функциональные и классовые компоненты.....	115
Отображение данных	116
Хук состояния.....	118
Сортировка таблицы.....	120
Редактирование данных	123
Поиск.....	125
Жизненные циклы в мире хуков	126
Проблемы с методами жизненного цикла.....	126
Хук useEffect()	127
Устранение побочных эффектов	128
Безаварийный жизненный цикл.....	130
Хук useLayoutEffect().....	131

Пользовательский хук.....	134
Завершение воспроизведения	137
Хук useReducer	139
Функции редюсера	139
Действия	140
Пример редюсера	141
Модульное тестирование редюсера	144
Компонент Excel с редюсером.....	145
Глава 5. JSX	149
Несколько инструментов.....	150
Пробельные символы в JSX.....	151
Комментарии в JSX	153
Элементы HTML	154
Anti-XSS.....	156
Распространяемые атрибуты.....	157
Атрибуты, распространяемые от родительского компонента к дочернему	158
Возвращение в JSX нескольких узлов	161
Оболочка	161
Фрагмент	162
Массив	162
Отличия JSX от HTML	163
Просто class использовать нельзя, а как насчет for?	164

Атрибут style — объект	164
Закрывающие теги	165
Атрибуты в верблюжьем регистре	166
Компоненты с пространством имен	166
JSX и формы	168
Обработчик события onChange	168
Сравнение value и defaultValue	170
Значение компонента <textarea>	171
Значение компонента <select>	172
Управляемые и неуправляемые компоненты	173

Глава 6. Настройки, необходимые для разработки

приложения	181
Создание React-приложения	182
Node.js	182
Привет, CRA	183
Сборка и развертывание	184
Были допущены ошибки	186
Файл package.json и папка node_modules	186
Рассмотрим код подробнее	187
Индексы	187
Модернизированный JavaScript	188
CSS	189
Идем дальше	190

Глава 7. Создание компонентов приложения	191
Настройка	192
Приступим к программированию	192
Рефакторинг компонента Excel	193
Версия 0.0.1 нового приложения.....	196
CSS.....	197
Локальное хранилище	198
Компоненты	199
Исследование.....	201
Логотип и тело	203
Логотип.....	203
Тело	204
Исследование компонентов.....	204
Компонент <button>.....	205
Файл Button.js.....	206
Пакет classnames.....	207
Формы	209
Компонент <Suggest>	209
Компонент <Rating>	211
Компонент <FormInput>	215
Компонент <Form>	218
Компонент <Actions>.....	224
Диалоги	226
Компонент <Header>	232

Конфигурация приложения.....	233
<Excel>: новый и усовершенствованный	235
Общая структура.....	238
Отображение	239
Строгий режим React и редюсеры.....	244
Небольшие вспомогательные функции Excel.....	247
Глава 8. Готовое приложение	254
Обновленный App.js.....	257
Компонент DataFlow	258
Тело компонента DataFlow	260
Работа выполнена	263
Whinepad v2	266
Контекст	266
Следующие шаги.....	267
Циклические данные	268
Предоставление контекста	270
Потребление контекста	273
Контекст в заголовке.....	274
Контекст в таблице данных	279
Обновление Discovery.....	281
Маршрутизация	284
Контекст маршрута	285
Использование URL фильтра.....	288

Использование контекста маршрута в заголовке	290
Использование контекста маршрута в таблице данных	292
Хук <code>useCallback()</code>	294
Заключение	298
Об авторе	299
Иллюстрация на обложке	300

Hello World

Приступим к освоению разработки приложений с использованием React. В этой главе вы узнаете, как установить React и написать свое первое веб-приложение Hello World.

Установка

Прежде всего вам нужно получить копию библиотеки React. Это можно сделать разными способами. Остановимся на самом простом из них. Он не требует никаких специальных инструментов и может помочь вам получить и освоить React в кратчайшие сроки.

Создайте папку для всего кода в книге в месте, где вы сможете его найти. Например:

```
$ mkdir ~/reactbook
```

Создайте папку `/react`, чтобы хранить код библиотеки React отдельно.

```
$ mkdir ~/reactbook/react
```

Далее необходимо добавить два файла: один — сам React, второй — пакет ReactDOM. Вы можете получить последние версии 17.* этих двух файлов с хоста unpkg.com, как показано ниже:

```
$ curl -L https://unpkg.com/react@17/umd/react.development.js >
~/reactbook/react/react.js
$ curl -L https://unpkg.com/react-dom@17/umd/react-dom.
development.js > ~/reactbook/react/react-dom.js
```

Обратите внимание, что React не навязывает какую-либо структуру каталогов, вы можете свободно переместить файлы в другой каталог или переименовать `react.js` так, как считаете нужным.

Вам не обязательно скачивать библиотеки — вы можете пользоваться ими напрямую с сайта unpkg.com. Однако их наличие в локальной сети позволяет учиться в любом месте, не подключаясь к Интернету.



Указав @17 в URL, показанных в предыдущем примере, вы получите копию последней версии React 17, которая актуальна на момент написания этой книги. Опустите @17, чтобы получить последнюю доступную версию React. Кроме того, вы можете явно указать требуемую версию, например @17.0.2.

Hello React World

Начнем с простой страницы в вашем рабочем каталоге (`~/reactbook/01.01.hello.html`):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello React</title>
    <meta charset="utf-8">
  </head>
  <body>
    <div id="app">
```

```
    <!-- здесь будет отображено мое приложение -->
  </div>
  <script src="react/react.js"></script>
  <script src="react/react-dom.js"></script>
  <script>
    // код моего приложения
  </script>
</body>
</html>
```



Весь представленный в книге код можно найти в соответствующем репозитории по адресу <https://github.com/stoyan/reactbook2>.

В этом файле внимания заслуживают лишь две особенности:

- включение библиотеки React и дополнение ее объектной модели документа (Document Object Model, DOM) с помощью тегов `<script src>`;
- определение места, где должно находиться ваше приложение на странице (`<div id="app">`).



С React-приложением всегда можно смешивать обычный HTML-контент, а также другие библиотеки JavaScript. Вы также можете иметь несколько приложений React на одной странице. Все, что вам нужно, — это место в DOM, на которое вы можете нацелить библиотеку React и сказать ей: «Твори здесь свое волшебство».

Теперь добавим код, который говорит hello. Для этого изменим содержимое файла `01.01.hello.html`, заменив строку `// код моего приложения` следующими строками:

```
ReactDOM.render(
  React.createElement('h1', null, 'Hello world!'),
  document.getElementById('app')
);
```

Загрузите файл `01.01.hello.html` в браузер — и увидите ваше новое приложение в действии (рис. 1.1).

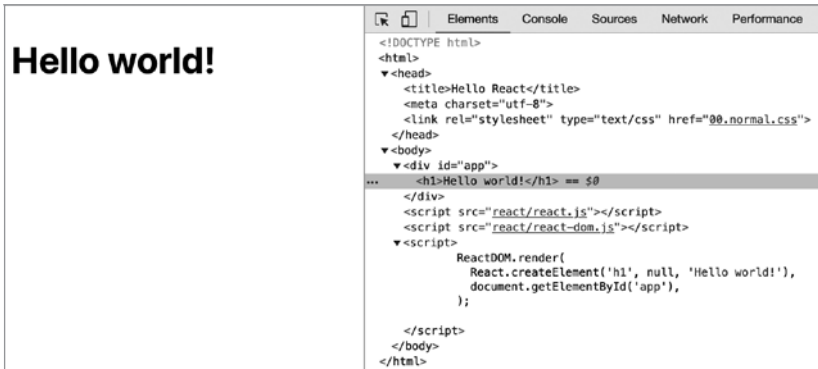


Рис. 1.1. Hello world в действии

Поздравляем, вы только что создали свое первое React-приложение!

На рис. 1.1 также показано, в каком виде *сгенерированный* код отображается в области инструментов разработчика DevTools браузера Chrome: видно, что содержимое контейнера `<div id="app">` было заменено содержимым, сгенерированным вашим React-приложением.

Так что же сейчас произошло?

В коде, который заставил работать ваше первое приложение, есть несколько интересных моментов.

В первую очередь видно, что использовался объект `React`. Обращение ко всем доступным вам API происходит с его помощью. API намеренно сделан минимальным, чтобы не пришлось запоминать слишком много имен методов.

Можно также посмотреть на объект ReactDOM. В нем всего несколько методов, наиболее полезным из которых является `render()`. ReactDOM отвечает за отображение приложения в браузере. На самом деле вы можете создавать приложения React и отображать их в различных средах вне браузера — например, на холсте (canvas) или прямо в Android или iOS.

Затем следует обратить внимание на концепцию *компонентов*. Вы создаете свой пользовательский интерфейс с помощью компонентов и комбинируете их так, как считаете нужным. В своих приложениях вы, конечно же, будете создавать собственные компоненты, но для начала React предоставляет вам надстройки над HTML DOM-элементами. Вы используете эти надстройки с помощью функции `React.createElement`. В нашем первом примере показано использование компонента `h1`. Он соответствует `<h1>` в HTML и доступен вам с помощью вызова `React.createElement('h1')`.

И наконец, вы можете заметить хорошо известный DOM-доступ к контейнеру, осуществляемый с помощью метода `document.getElementById('app')`. Он используется для того, чтобы подсказать React, где именно на странице должно быть расположено приложение. Это своеобразный мост, перекинутый между манипуляцией с известными вам DOM-объектами и React-территорией.

После переброски моста от DOM к React вам больше не придется беспокоиться о манипуляциях с DOM, поскольку React переводит свои компоненты на базовую платформу (DOM браузера, canvas, нативное приложение). На самом деле то, что не нужно беспокоиться о DOM, — одна из замечательных особенностей React. Вы заботитесь о компоновке компонентов и их данных — сути приложения — и позволяете React позаботиться о наиболее эффективном обновлении DOM. Больше не нужно искать узлы DOM, `firstChild`, `appendChild()` и т. д.



Вам не нужно заниматься DOM, но это не значит, что вы не можете этого делать. React предоставляет вам «резервные переходы», если возникнет необходимость вернуться к источникам DOM по любой причине, которую вы сочтете важной.

Теперь, выяснив роль каждой строки, взглянем на общую картину. Произошло следующее: вы отображали один React-компонент в выбранном вами месте DOM. Вы всегда отображаете один компонент верхнего уровня, и он может иметь любое необходимое вам количество дочерних (и производных от них) компонентов. Фактически даже в этом простом примере у компонента `h1` имеется дочерний компонент — текст `Hello world!`.

Метод `React.createElement()`

Как вам теперь известно, вы можете использовать несколько элементов HTML в качестве компонентов React с помощью метода `React.createElement()`. Рассмотрим этот API более подробно.

Вспомним, что приложение `Hello world!` выглядело следующим образом:

```
ReactDOM.render(  
  React.createElement('h1', null, 'Hello world!'),  
  document.getElementById('app')  
);
```

Первый параметр `createElement` — это тип создаваемого элемента. Второй параметр (который в данном случае имеет значение `null`) является объектом, указывающим на любые свойства (подумайте об атрибутах DOM), которые вы хотите передать вашему элементу. Например, вы можете сделать следующее:

```
React.createElement(  
  'h1',  
  {  
    id: 'my-heading',
```

```
  },  
  'Hello world!'  
)
```

Код HTML, созданный в этом примере, показан на рис. 1.2.

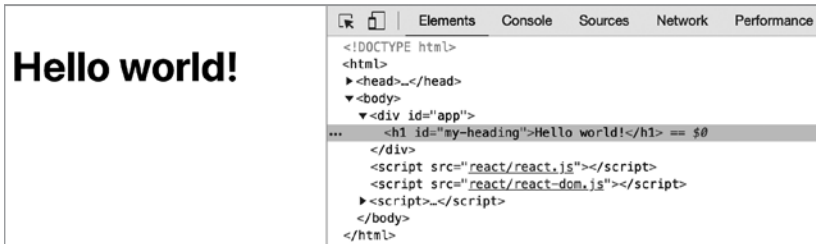


Рис. 1.2. HTML, сгенерированный вызовом React.createElement()

Третий параметр (в данном примере "Hello world!") определяет дочерний элемент компонента. В простейшем случае это всего лишь текстовый дочерний элемент (узел Text в DOM-терминологии), как вы можете видеть в предыдущем коде. Но у вас может быть сколько угодно вложенных дочерних компонентов, которые вы передаете в качестве дополнительных параметров функции. Например:

```
React.createElement(  
  'h1',  
  {id: 'my-heading'},  
  React.createElement('span', null, 'Hello'),  
  ' world!'  
)
```

А вот еще один пример, на этот раз с вложенными компонентами (результат показан на рис. 1.3), который выглядит следующим образом:

```
React.createElement(  
  'h1',  
  {id: 'my-heading'},
```

```

React.createElement(
  'span',
  null,
  'Hello',
  React.createElement('em', null, 'Wonderful'),
),
' world!'
),

```

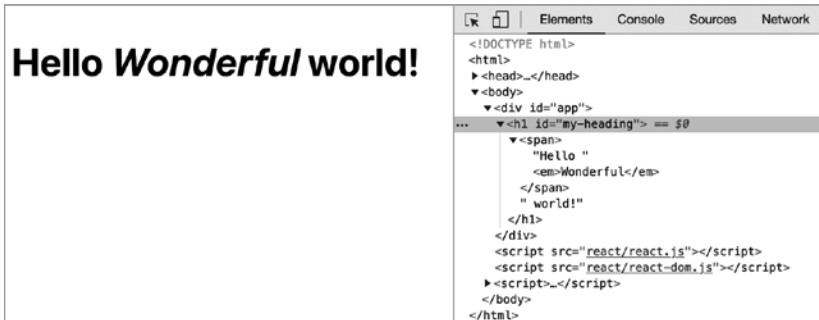


Рис. 1.3. HTML, созданный путем вложения вызовов `React.createElement()`

На рис. 1.3 видно, что в DOM, сгенерированном React, элемент `` является дочерним элементом элемента ``, который, в свою очередь, является дочерним элементом элемента `<h1>` (и дочерним элементом текстового узла `world!`).

Синтаксис JSX

Начиная вкладывать компоненты, вы быстро получаете множество вызовов функций и круглых скобок, которые нужно отслеживать. Чтобы упростить задачу, вы можете использовать *синтаксис JSX*. Он является немного спорным: люди часто поначалу находят

его отталкивающим («тьфу, XML в моем JavaScript!»), но впоследствии незаменимым.



Не совсем понятно, что означает аббревиатура JSX, но, скорее всего, это JavaScriptXML или JavaScript Syntax eXtension. Официальный сайт данного проекта с открытым исходным кодом — <https://facebook.github.io/jsx>.

Предыдущий фрагмент, на этот раз с использованием синтаксиса JSX, выглядит так:

```
ReactDOM.render(  
  <h1 id="my-heading">  
    <span> Hello<em> Wonderful</em></span> world!  
  </h1>,  
  document.getElementById('app')  
)
```

Такой код намного удобнее читать. Этот синтаксис очень похож на уже известный вам HTML. Есть только одно затруднение: данный код не является допустимым синтаксисом JavaScript, поэтому его нельзя запустить в браузере в неизменном виде. Вам нужно преобразовать (*транспилировать*) код в чистый JavaScript, который может быть запущен браузером. Более того, в целях обучения вы можете сделать это без специальных инструментов. Вам понадобится библиотека Babel, которая переводит современный JavaScript (и JSX) в традиционный JavaScript, работающий в старых браузерах.

Установка Babel

Как и в случае с React, получите локальную копию Babel:

```
$ curl -L https://unpkg.com/babel-standalone/babel.min.js >  
~/reactbook/react/ babel.js
```

Затем вам нужно обновить свой учебный шаблон, включив в него Babel. Создайте файл `01.04.hellojsx.html` следующим образом:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Hello React+JSX</title>
    <meta charset="utf-8">
  </head>
  <body>
    <div id="app">
      <!-- мое приложение отображается здесь -->
    </div>
    <script src="react/react.js"></script>
    <script src="react/react-dom.js"></script>
    <script src="react/babel.js"></script>
    <script type="text/babel">
      // код моего приложения
    </script>
  </body>
</html>
```



Обратите внимание, как `<script>` превращается в `<script type="text/babel">`. Это прием, при котором, указав недопустимый тип, браузер игнорирует код. Это дает Babel возможность разобрать и преобразовать синтаксис JSX в то, что браузер может запустить.

Привет, мир JSX

С этой небольшой настройкой попробуем JSX. Замените часть `// код моего приложения` в предыдущем HTML на:

```
ReactDOM.render(
  <h1 id="my-heading">
    <span> Hello <em>JSX</em></span> world!
  </h1>,
  document.getElementById('app')
);
```

Результат выполнения этого кода в браузере показан на рис. 1.4.

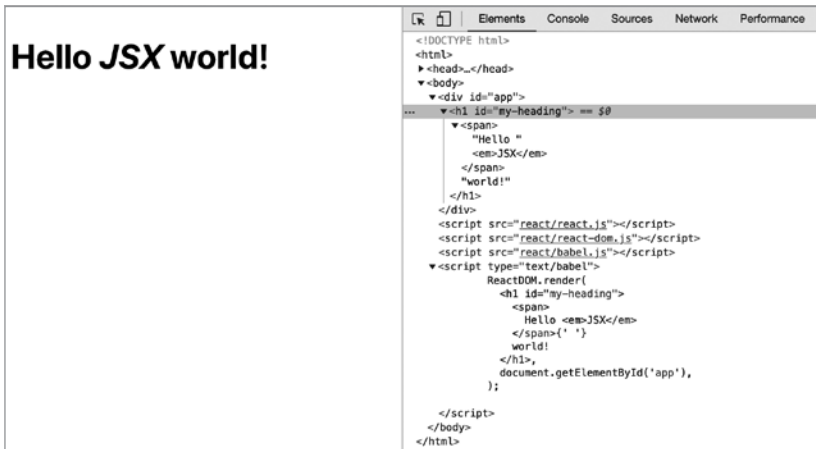


Рис. 1.4. Hello JSX world

0 транпиляции

Итак, JSX и Babel работают, и это замечательно. Но, возможно, вам будут полезны еще некоторые разъяснения, особенно если вы новичок в Babel и процессе транпиляции. Если вы уже знакомы с ним, то смело пропускайте этот фрагмент, в котором мы немного познакомимся с терминами *JSX*, *Babel* и «*транпиляция*».

JSX представляет собой отдельную от React технологию, и ее не обязательно применять. Как вы уже видели, первые примеры в этой главе даже не использовали JSX. Вы можете сделать выбор в пользу того, чтобы вообще отказаться от JSX. Но все же высока вероятность того, что, один раз попробовав этот синтаксис, вы уже не захотите возвращаться к вызовам функций.

Процесс *транпиляции* заключается в следующем: исходный код переписывается, чтобы получить такие же результаты, но уже

с помощью синтаксиса, понимаемого устаревшими браузерами. Этот процесс отличается от использования *полифиллов*. Примером полифилла может служить добавление к `Array.prototype` метода, аналогичного методу `map()`, который был введен стандартом ECMAScript5, чтобы заставить его работать в браузерах, поддерживающих только стандарт ECMAScript3. Полифилл представляет собой решение из области чистого JavaScript. Это хорошее решение при добавлении новых методов к существующим объектам или реализации новых объектов (таких как JSON). Но этого недостаточно, когда в язык вводится новый синтаксис. Любой новый синтаксис для браузера, который его не поддерживает, просто недействителен и выдает ошибку разбора. Нет никакого способа его дополнить. Поэтому новый синтаксис требует проведения компиляции (транспиляции), чтобы он был преобразован *до того*, как будет представлен для обработки браузеру.

Транспиляция JavaScript становится все более распространенным процессом, поскольку программисты хотят использовать новейшие функциональные возможности JavaScript (ECMAScript), не дожидаясь выхода браузеров, поддерживающих эти возможности. Если у вас уже настроен и отработан процесс сборки (который, например, выполняет минификацию или любое другое преобразование кода), то вы можете просто добавить к нему шаг транспиляции JSX. Если же у вас *нет* процесса сборки, то далее в книге вы пройдете все необходимые шаги для его настройки.

А пока оставим транспиляцию JSX на стороне клиента (в браузере) и продолжим изучать React. Просто имейте в виду, что это делается только в образовательных и экспериментальных целях. Транспиляция на стороне клиента не предназначена для реальных производственных сайтов, поскольку она медленнее и требует больше ресурсов, чем обслуживание уже транспилированного кода.

Далее: настраиваемые компоненты

На данном этапе вы справились с созданием простейшего приложения Hello world. Теперь вы знаете, как:

- установить библиотеку React, произвести ее настройку и воспользоваться ею для экспериментов и обучения (на самом деле это всего лишь вопрос нескольких тегов `<script>`);
- вывести React-компонент в выбранном вами месте DOM (например, `ReactDOM.render(reactWhat, domWhere)`);
- использовать встроенные компоненты, которые являются оболочками для обычных DOM-элементов (например, `React.createElement(element, attributes, content, children)`).

Однако истинные возможности React проявятся, когда вы начнете использовать настраиваемые компоненты для создания (и обновления!) пользовательского интерфейса (user interface, UI) вашего приложения. Как это делать, вы узнаете в следующей главе.

ГЛАВА 2

Жизнь компонента

После того как вы узнали о том, как использовать готовые DOM-компоненты, пришло время изучить способы создания собственных компонентов.

UI можно создавать двумя способами, оба они дают одинаковый результат, но используют разный синтаксис:

- применить функцию (компоненты, созданные таким образом, называются *функциональными*);
- использовать класс, расширяющий `React.Component` (обычно называемый *классовым компонентом*).

Настраиваемый функциональный компонент

Пример функционального компонента выглядит так:

```
const MyComponent = function() {  
  return 'I am so custom';  
};
```

Но подождите, это же всего лишь функция! Да, настраиваемый функциональный компонент — просто функция, которая возвращает нужный вам пользовательский интерфейс. В данном случае UI — лишь текст, но часто вам будет требоваться что-то большее,