ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Прежде не было более подходящего времени для программирования на JavaScript. За последние несколько лет на наших глазах JavaScript прошел путь от языка, с которым никто не хотел иметь дело, до языка, которым все непременно желают овладеть. Разработка больших и сложных приложений для браузеров сегодня подталкивает развитие JavaScript , как никогда раньше. Фреймворки и полностью новые подходы к созданию приложений породили новые потребности в разработке на стороне клиента, и сообщество пришло к соглашению о необходимости их удовлетворения.

Спецификация ECMAScript 2015 или ES6, как ее обычно называют, наконец-то, привела язык в соответствие с нашими высокими требованиями. Значительными приобретениями являются встроенная поддержка отложенных вычислений и модульной организации, но присутствуют также и небольшие дополнения, которые делают повседневную разработку более приятной. Познакомившись поближе с приемом деструктурирования объектов, вы будете удивляться, как прежде удавалось обходиться без него, а впервые воспользовавшись стрелочной функцией, вы никогда не захотите вновь использовать «function». С помощью функции «let» вы избавитесь от сложностей с областями видимости функций и от утечки переменных, и вам реже придется биться головой о стену.

ES6 – великолепный язык, совершивший невероятный скачок относительно ES5, а упорный труд многих членов сообщества сделал возможным его использование уже сегодня, не дожидаясь полной реализации в браузерах. Существуют инструменты преобразования кода ES6 в код, совместимый со спецификацией ES5, а это значит, что будущее уже здесь, его не надо ждать еще 5 лет, как это часто бывало с JavaScript.

Эта книга познакомит вас с наиболее полезными нововведениями в JavaScript и всей, доступной уже сейчас, функциональностью. Научит, как конструировать модульные приложения, используя встроенную систему модулей ES6, и как сделать код чище, лаконичнее и

более приятным для работы. Изучение нового стандарта является сложной задачей для любого разработчика, и я рад внести свой вклад, написав предисловие к книге, которая значительно упростит эту задачу.

Эта книга поможет вам сделать первые шаги в новом удивительном мире JavaScript, клиентских приложений и фреймворков, основанном на спецификации ES6. Я надеюсь, что дочитав эту книгу до конца, вы будете так же взволнованы, как и я.

Джек Франклин (Jack Franklin)
Программист JavaScript из GoCardless
@Jack_Franklin
http://www.jackfranklin.co.uk

ОБ АВТОРЕ

Нараян Прасти (Narayan Prusty) разработчик веб- и мобильных приложений. Специализируется на WordPress, HTML5, JavaScript, PHP, Solr и Cordova. Изучал эти технологии и создавал приложения с их помощью в течение многих лет.

Является основателем сайта QScutter.com, который проводит курсы по различным темам разработки приложений и имеет более 10 000 подписчиков по всему миру. Его личный блог http://www.QNimate.com один из самых популярных в рейтинге блогов Intel XDK и WordPress. Он также работает консультантом и внештатным разработчиком во многих компаниях по всему миру.

Посетите его личную страничку http://www.twitter.com/narayanprusty.

В первую очередь хочу выразить благодарность веб-сообществу. Без их блестящих успехов в документировании и щедрого обмена решениями, я не смог бы написать эту книгу. И, наконец, я благодарен моей семье за поддержку.

О ТЕХНИЧЕСКИХ РЕЦЕНЗЕНТАХ

Андреа Чиарелли (Andrea Chiarelli) имеет более чем 20 летний опыт работы инженером-программистом и техническим писателем. В своей профессиональной карьере использовал различные технологии: от C# до JavaScript, от ASP.NET до AngularJS, от REST до PhoneGap/Cordova.

Сотрудничал со многими электронными и печатными журналами, такими как *Computer Programming* и *ASP Today*, и был соавтором нескольких книг, изданных Wrox Press.

В настоящее время работает старшим инженером-программистом в итальянском офисе компании Apparound Inc., основанной в самом центре Силиконовой долины и занимающейся разработкой программ для мобильных устройств, а также является постоянным автором итальянского электронного журнала *HTML.it*, специализирующегося на веб-технологиях.

Филипп Реневье Гонен (Philippe Renevier Gonin) с 2005 года работает ассистентом профессора в Университете Софии Антиполис (Ницца, Франция). Преподает веб-технологии, программную инженерию (проектирование и разработку), и предмет «Взаимодействие человека с компьютером» (Human Computer Interaction, сокращено HCI). Как исследователь, Филипп занимается связями между интерактивным пользовательским дизайном (например, моделями пользователей и задач) и программным обеспечением (например, компонентной архитектурой и разработкой пользовательского интерфейса). В своих проектах часто использует программное обеспечение и инструменты Javascript, HMTL, CSS, Java (Android).

Доменико Лучиани (Domenico Luciani**),** увлеченный 22-летний программист. В настоящее время работает инженером-программистом в нескольких компаниях и получает степень в области компьютерных наук в Университете Палермо.

С энтузиазмом занимается задачами распознавания образов. Отдает предпочтение компьютерной безопасности и пентестам, принимал участие в премиальных программах ряда компаний. Работал со многими технологиями: MongoDB, Node.js, PHP, PostgreSQL и C.

Пишет модули Node.js, которые публикует на сайте NPM. Не раз выступал в роли технического рецензента и в настоящее время изучает язык программирования GoLang, просто для удовольствия.

Является членом сообщества Maker и любит работать на своем одноплатным компьютере Raspberry Pi. Обожает писать код в текстовом редакторе Vim и управлять исходными текстами с помощью Git. Пишет тесты и участвует в проектах с открытым исходным кодом в Интернете.

В свободное время занимается бегом, любит паркур. Вы можете найти более подробную информацию о нем на сайте http://www.dlion.it.

Михир Мone (Mihir Mone), аспирант из Университета Монаша, Австралия. Окончил курс обучения по распределенным вычислениям, но сейчас занимается разработкой веб- и мобильных приложений. Повозившись, некоторое время, с маршрутизаторами и коммутаторами, он решил реализовать свою тягу к веб-разработке, не дизайну, а именно разработке. Его интересует и привлекает создание веб-систем и приложений, а не веб-сайтов со всей их фантастической флеш-анимацией. Он даже вернулся в свою альма-матер, чтобы преподавать веб-разработку, вернуть полученные им знания.

Теперь он работает в небольшой инженерно-программной фирме в Мельбурне, где занимается веб-разработкой и генерирует новые увлекательные идеи в области визуализации данных и взаимодействий между человеком и машиной.

Он также большой поклонник JavaScript и принимал участие в рецензировании нескольких книг о JQuery и JavaScript. Энтузиаст Linux и большой сторонник движения за открытое программное обеспечение. Считает, что программное обеспечение всегда должны быть свободным, чтобы раскрыть весь свой потенциал.

Как убежденный компьютерщик, тратит часть своего свободного времени на написание кода, в надежде, что это поможет другим. Вы можете найти более подробную информацию о нем на сайте http://mihirmone.apphb.com.

Такехару Ошида (Takeharu Oshida) (https://github.com/georgeOsdDev) работает в недавно запущенном проекте Mobilus (http://mobilus.

co.jp/), занимающемся созданием коммуникационной платформы реального времени и SDK под названием Konnect.

Как программист на JavaScript, разрабатывает JavaScript-библиотеки и ES6-совместимые веб-приложения на React.JS.

Он также участвует в разработке веб-фреймворка Xitrum (http://xitrum-framework.github.io/). В рамках этого проекта изучает функциональное программирование на языке Scala, создавая примеры приложений и занимаясь переводом документации.

Был техническим рецензентом книги *Learning Behavior-driven Development Javascript*, опубликованной издательством Packt Publishing.

Юрий Струмпфлохнер (Juri Strumpflohner) увлеченный разработчик, любит программировать, следит за последними тенденциями в веб-разработке и делится своими выводами с другими. Работает программным архитектором и техническим руководителем в компании поддержки электронного правительства, где отвечает за подготовку разработчиков, внедрение инноваций и контроль качества программного обеспечения.

В свободное время участвует в проектах с открытым исходным кодом, занимается рецензированием книг (подобных этой), переписывается в Twitter (@juristr) или пишет о последних новостях технологий веб-разработки в своем блоге на http://juristr.com. В настоящее время его особенно интересуют ES 2015 (ES6), AngularJS, React, Babel и все связанное с современной веб-разработкой.

Если Юрий не программирует, то учится или тренируется, занимаясь боевым искусством Йосейкан Будо (Yoseikan Budo), где достиг черного пояса второго дана. Вы можете связаться с ним в Twitter (@juristr) или посетить его блог http://juristr.com.

ВВЕДЕНИЕ

ECMAScript – это язык сценариев, стандартизированный организацией Ecma International в спецификациях ECMA-262 и ISO/IEC 16262. Языки сценариев, такие как JavaScript, JScript и ActionScript, являются надмножествами ECMAScript. Хотя в JavaScript, JScript, и ActionScript дают больше возможностей, чем в ECMAScript, определяя множество дополнительных объектов и методов, основные черты этих языков совпадают с чертами ECMAScript.

ECMAScript 6 является шестой версией и седьмой редакцией языка ECMAScript. Короткое его название «ES6».

Хотя JavaScript чрезвычайно мощный и гибкий язык, его часто критикуют за ненужную избыточность. Поэтому, разработчики часто используют абстракции JavaScript, такие как CoffeeScript и Typescript, имеющие упрощенный синтаксис, мощные функции и компилирующиеся в JavaScript. Спецификация ES6 была введена для такого улучшения JavaScript, которое убедит разработчиков не обращаться к абстракциям или другим методам для написания качественного кода, увеличивающим время выполнения.

Особенности ES6 унаследованы от других популярных абстрагированных языков, таких как CoffeeScript. То есть, особенности языка ES6 схожи с особенностями других языков и не новы в мире программирования, в тоже время они являются новыми для JavaScript.

Эта книга содержит разъяснения, сопровождаемые примерами, всех особенностей новой версии ECMAScript – ECMAScript 6. Она посвящена реализации стандарта ECMAScript 6 в JavaScript. Все функции и примеры в этой книге работают во всех окружениях JavaScript, таких как браузеры, Node.js, Cordova и так далее.

О чем рассказывается в этой книге

Глава 1 «Игры с синтаксисом», описывает новые способы создания переменных и параметров функций. В этой главе подробно обсуждаются новые объекты и функции.

Глава 2 «Знакомство с библиотекой», знакомит с новыми методами существующих объектов, основанными на прототипах.

Глава 3 «Использование итераторов», описывает различные типы итераторов, доступных в ES6, и приемы создания пользовательских итераторов. Она также рассматривает оптимизацию хвостового вызова в ES6.

Глава 4 «Асинхронное программирование», рассказывает, как объекты Promise помогают упростить создание кода, выполняемого асинхронно.

Глава 5 «Реализация Reflect API», служит подробным руководством по отражению объектов в ES6.

Глава 6 «Использование прокси-объектов», объясняет, как определить нестандартное поведение основных операций над объектами с помощью прокси-объектов ES6.

Глава 7 «Прогулка по классам», описывает приемы объектно-ориентированного программирования с использованием классов ES6. Объясняет такие понятия, как наследование, конструкторы, абстракции, сокрытие информации и другие.

Глава 8 «Модульное программирование», рассматривает разные способы создания модулей в JavaScript. Охватывает технологи создания модулей, такие как IIFE, CommonJS, AMD, UMD и ES6.

Что понадобится при чтении этой книги

Если вы читаете эту книгу после того, как стандарт ES6 стал полностью поддерживаться всеми движками JavaScript, то вам не понадобится никакой специальной среды для тестирования примеров из нее. Вы сможете просто проверить примеры на любом выбранном вами движке.

Если вы читаете эту книгу, до того как ES6 стал полностью поддерживается всеми механизмами JavaScript, тогда для опробования фрагментов кода, приведенных в книге, можно использовать транскомпилятор ES6. Чтобы опробовать примеры в браузере, используйте шаблон веб-страницы, с присоединенным к ней транскомпилятором Traceur для преобразования кода ES6 в код ES5, для каждой загружаемой страницы:

```
<!doctype html>
<html>
```

```
<head>...</head>
<body>
...
<script src="traceur.js"></script>
<script src="bootstrap.js"></script>
<script type="module">
          // Поместите программный код на ES6 сюда
</script>
</body>
</html>
```

Загрузите сценарии `traceur.js` (https://google.github.io/traceur-compiler/bin/traceur.js) и `bootstrap.js` (https://google. github.io/traceur-compiler/src/bootstrap.js). Поместите их в папку, где находится HTML-файл, содержащий приведенный выше код.

В прилагаемых к книге файлах (пакет кода), транскомпилятор Traceur и упомянутые сценарии уже подключены. Эти файлы предназначены для опробования примеров в браузерах.

Опробовать примеры из *главы 4 «Асинхронное программирование»* можно только в браузере, так как они используют JQuery и AJAX. Здесь вам также понадобится веб-сервер.

Чтобы опробовать примеры из *главы 8 «Модульное программирование»* в браузере, вам понадобится веб-сервер. Но если вы используете среду Node.js, то веб-сервер вам не потребуется.

Совместимость с ECMAScript 6

Эта книга была написана до того, как все реализации JavaScript начали поддерживать все функции ES6.

Подготовка спецификаций ES6 уже завершена. Но еще не все реализации JavaScript поддерживают все особенности ES6. Я уверен, что к концу 2016 года все реализации JavaScript будут поддерживать ES6.

В рамках проекта Kangax создана таблица совместимости с ES6, где можете отслеживать поддержку различных функций ES6 разными реализациями JavaScript. Найти эту таблицу можно по адресу http://kangax.github.io/compat-table/es6/.

Запуск ECMAScript 6 в несовместимых реализациях

Если вы захотите запустить программный код ES6 в реализации JavaScript, не поддерживающей стандарт ES6, используйте полифилы ES6 или транскомпиляторы ES6.

Полифил (polyfill) – это фрагмент кода, обеспечивающий поддержку технологии, которую вы, как разработчик, ожидаете получить от реализации JavaScript. Помните, что полифилы поддерживают не все функции ES6. Список доступных полифилов и ссылок для их загрузки можно найти в на странице https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills#ecmascript-6-harmony.

Транскомпилятор ES6 принимает исходный код на ES6 и выводит исходный код на ES5, совместимый со всеми реализациями JavaScript. Транскомпиляторы поддерживают больше возможностей, чем полифилы, но также не способны обеспечить поддержку всех функций ES6. Существуют множество разных транскомпиляторов, таких как Google Traceur (https://github.com/google/traceur-compiler), Google Caja (https://developers.google.com/caja/), Babel (https://babeljs.io/), Termi ES6 Transpiler (https://github.com/termi/es6-transpiler) и многие другие. Вы всегда должны транскомпилировать код ES6 в код ES5 перед подключением его к веб-страницам, а не каждый раз при загрузке страницы, чтобы не замедлять их загрузку.

Таким образом, с помощью транскомпиляторов и/или полифилов, можно начать писать код на ES6 до того, как все реализации начнут полностью поддерживать ES6, а реализации без поддержки ES6 уйдут в прошлое.

Кому адресована эта книга

Эта адресована всем, кто знаком с JavaScript. Вы не должны быть экспертом в JavaScript, чтобы разобраться в приведенных здесь сведениях. Эта книга поможет вам поднять ваше знание JavaScript на новый уровень.

Соглашения

В этой книге вы обнаружите несколько стилей оформления текста, которые разделяют различные виды информации. Ниже приводятся примеры этих стилей и поясняется их значение.

Фрагменты кода в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, адреса страниц в Интернете, пользовательский ввод и указатели Twitter будут выглядеть так: «Мы можем подключать другие контексты посредством использования директивы `include`».

Блоки программного кода оформляются, как показано ниже:

```
var a = 12; // доступна глобально
function myFunction()
{
  console.log(a);
  var b = 13; // доступна в пределах функции
  if(true)
  {
    var c = 14; // доступна в пределах функции
    console.log(b);
  }
  console.log(c);
}
myFunction();
```

Новые термины и **важные слова** будут выделены жирным шрифтом.

Предупреждения или важные сообщения будут выделены подобным образом.

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 для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф в разделе **Читателям – Файлы к книгам**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки всё равно случаются. Если вы найдёте ошибку в одной из наших книг – возможно, ошибку 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 этой книги.

Если вы найдёте какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства ДМК Пресс и Packt очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной 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 dmkpress@gmail.com со ссылкой на подозрительные материалы.

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

ГЛАВА 1. Игры с синтаксисом

Языку **JavaScript** не доставало некоторых синтаксических форм, присутствующих в ряде других языков программирования, таких как объявление констант, объявление переменных с областью видимостью внутри блока, извлечение данных из массивов, короткий синтаксис для объявления функций и так далее. Спецификация **ES6** добавляет много новых синтаксических конструкций в JavaScript, которые позволят программистам писать более компактный и эффективный код. Также ES6 дает возможность отказаться от применения грубых приемов, отрицательно влияющих на производительность и затрудняющих чтение кода. В этой главе описаны новые синтаксические возможности, предоставляемые ES6.

В этой главе мы рассмотрим:

- создание переменных с областью видимости в пределах блока с помощью ключевого слова `let`;
- создание констант с помощью ключевого слова `const`;
- оператор расширения и дополнительные параметры;
- извлечение данных из итерируемых объектов с помощью деструктивного присваивания;
- стрелочные функции;
- новый синтаксис создания свойств объектов.

Ключевое слово let

В ES6 для объявления переменных с областью видимости в пределах блока и возможностью инициализации их значений ключевое используется слово `let`. У программистов, привыкших писать на других языках программирования и начинающих осваивать JavaScript, часто возникают ошибки, связанные с их представлениями об области видимости переменных. Почти все популярные языки исполь-

зуют единые правила определения области видимости переменных, но в JavaScript эти правила несколько отличаются из-за отсутствия переменных с областью видимости в пределах блока. В связи с этим увеличивается вероятность утечек памяти, усложняется чтение кода и его отладка.

Объявление переменных с областью видимости в пределах функции

Переменные JavaScript, объявленные с помощью ключевого слова `var` называются **переменными с областью видимости в пределах функции**. Переменные, объявленные вне функции, становятся глобальными, то есть доступными из любого места в сценарии. Соответственно, переменные, объявленные внутри функции, доступны только внутри функции, но не за ее пределами.

Ниже приводится пример создания переменных внутри и за пределами функции:

```
var a = 12; // доступна глобально
function myFunction()
{
  console.log(a);
  var b = 13; // доступна в пределах функции
  if(true)
  {
    var c = 14; // доступна в пределах функции
    console.log(b);
  }
  console.log(c);
}
myFunction();
```

Результаты выполнения примера:

```
12
13
14
```

Загрузка примеров кода

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф в разделе **Читателям – Файлы к книгам**.

Здесь, как видите, переменная `c` доступна в пределах всей функции, но в других языках программирования она была бы недоступна за пределами оператора `if`. Следовательно, программисты, привыкшие к другим языкам программирования, ожидают, что переменная `c` не определена вне пределов оператора `if`, но это не так. Спецификация ES6 вводит ключевое слово `let` для создания переменных с областью видимости в пределах блока.

Объявление переменных с областью видимости в пределах блока

Переменные, объявленные с использованием ключевого слова `let`, называются переменными с областью видимости в пределах блока. Такие переменные, объявленные вне функции, ведут себя как обычные переменные, то есть, они доступны глобально. Но когда переменные с областью видимости в пределах блока объявляются внутри блока, они доступны только в пределах этого блока (а также во всех вложенных в него блоках).

Блок может содержать в себе ни одного или сколько угодно операторов. Ограничивает блок пара фигурных скобок, то есть {}.

Возьмем предыдущий пример и, заменив ключевые слова `var` на `let`, посмотрим на результаты его выполнения:

```
let a = 12; // доступна глобально
function myFunction()
{
  console.log(a);
  let b = 13; // доступна в пределах функции
  if(true)
  {
    let c = 14; // доступна только в инструкции "if"
    console.log(b);
  }
  console.log(c);
}
myFunction();
```

Результаты выполнения примера:

```
12
13
Reference Error Exception
```

Теперь, результаты стали именно такими, какие ожидают программисты, привыкшие к другим языкам программирования.

Повторное объявление переменных

Если с помощью ключевого слова `var` второй раз объявить переменную с тем же именем (в той же области видимости), объявленная ранее переменная будет затерта. Например:

```
var a = 0;
var a = 1;
console.log(a);
function myFunction()
{
  var b = 2;
  var b = 3;
  console.log(b);
}
myFunction();
```

Результаты выполнения примера:

```
1
3
```

Ожидаемый результат. Но переменные, созданные с использованием ключевого слова `let`, ведут себя по-другому.

Если с помощью ключевого слова `let` второй раз объявить переменную с тем же именем (в той же области видимости), будет вызвано исключение `TypeError`. Например:

```
let a = 0;
let a = 1; // TypeError
function myFunction()
{
  let b = 2;
  let b = 3; // TypeError
  if(true)
  {
    let c = 4;
    let c = 5; // TypeError
  }
}
myFunction();
```

Если объявить переменную с именем, которое уже было использовано ранее в функции (или во вложенной функции) или во вну-

треннем блоке, с использованием ключевого слова `var` или `let`, соответственно, это будет другая переменная. Следующий пример иллюстрирует это правило:

```
var a = 1;
let b = 2;
function myFunction()
{
  var a = 3; // другая переменная
  let b = 4; // другая переменная
  if(true)
  {
    var a = 5; // затрет прежнее значение
    let b = 6; // другая переменная
    console.log(a);
    console.log(b);
  }
  console.log(a);
  console.log(b);
}
myFunction();
console.log(a);
console.log(b);
```

Результаты выполнения примера:

```
5
6
5
4
1
2
```

Ключевое слово `var` или ключевое слово `let`, что использовать?

В программном коде ES6 рекомендуется использовать ключевое слово `let`, потому что это улучшает характеристики использования памяти, уменьшает вероятность ошибок области видимости, предотвращает случайные ошибки и облегчает чтение кода. Но если вы привыкли к ключевому слову `var` и чувствуете себя комфортно, можете продолжать использовать его.

Вас может удивить, почему бы просто не изменить поведение ключевого слова `var` вместо ввода нового ключевого слова `let`? Причиной тому является необходимость обратной совместимости.