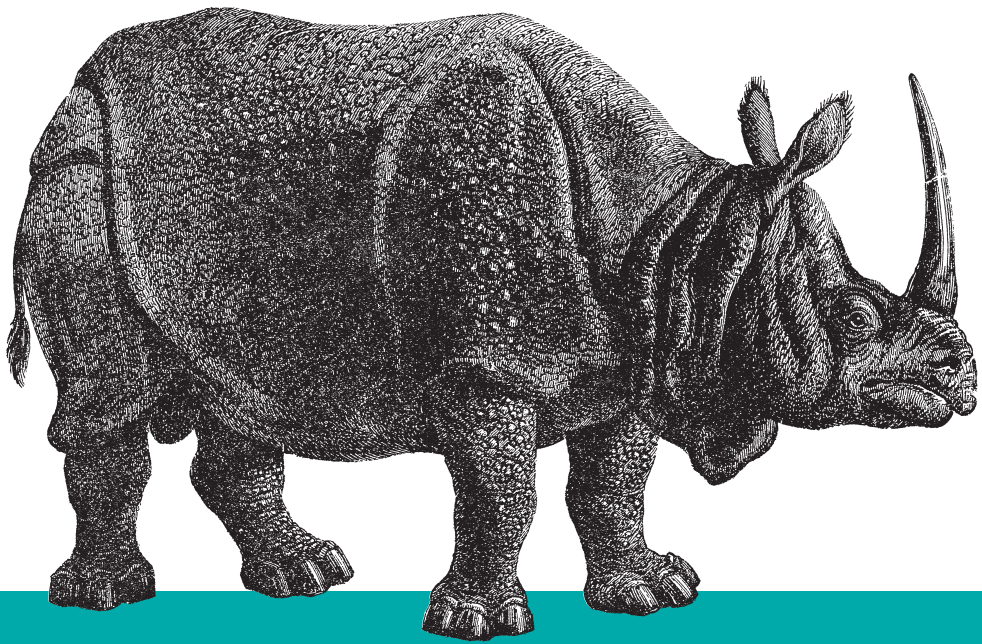
Создание активных веб-страниц

6-е издание
Включает ECMAScript 5 и HTML5



JavaScript

Подробное руководство

СИМВОЛ®
O'REILLY®

Дэвид Флэнаган

JavaScript

The Definitive Guide

Sixth Edition

David Flanagan

O'REILLY®

JavaScript

Подробное руководство

Шестое издание

Дэвид Флэнаган

Санкт-Петербург — Москва
2012

Дэвид Флэнаган

JavaScript. Подробное руководство, 6-е издание

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*Ю. Бочина*
Корректор	*Т. Школьная*
Верстка	*Д. Орлова*

Флэнаган Д.
JavaScript. Подробное руководство, 6-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2012. – 1080 с., ил.

ISBN 978-5-93286-215-5

Шестое издание бестселлера «JavaScript. Подробное руководство» полностью пересмотрено и дополнено сведениями о JavaScript в соответствии с современным положением дел в разработке приложений для Web 2.0. Эта книга – одновременно и руководство программиста с большим числом практических примеров, и полноценный справочник по базовому языку JavaScript и клиентским прикладным интерфейсам, предоставляемым веб-броузерами.

Издание охватывает стандарты ECMAScript 5 и HTML5. Многие главы переписаны заново, другие дополнены новой информацией, появились и новые главы с описанием библиотеки jQuery и поддержки JavaScript на стороне сервера.

Часть I знакомит с основами JavaScript. В части II описывается среда разработки сценариев, предоставляемая веб-броузерами. Основное внимание уделяется разработке сценариев с применением методики ненавязчивого JavaScript и модели DOM. Часть III – обширный справочник по базовому языку JavaScript, включающий описания всех классов, объектов, конструкторов, методов, функций, свойств и констант, определенных в JavaScript 1.8, V8 3.0 и ECMAScript 5. Часть IV – справочник по клиентскому JavaScript. Здесь описываются API веб-броузеров, стандарт DOM API Level 3 и недавно вошедшие в стандарт HTML5 технологии WebSockets и WebWorkers, объекты localStorage и sessionStorage, а также теги <audio> и <video>.

Издание рекомендуется программистам, которым потребовалось изучить язык программирования для Веб, а также программистам, использующим язык JavaScript и желающим овладеть им в совершенстве.

ISBN 978-5-93286-215-5
ISBN 978-0-596-80552-4 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 30.05.2012. Формат 70×100 1/16 .
Печать офсетная. Объем 67,5 печ. л.

Эта книга посвящается всем,
кто учит жить в мире и противостоит насилию.

Оглавление

Предисловие

Эта книга охватывает язык программирования JavaScript и прикладные интерфейсы JavaScript, реализованные в веб-броузерах. Я писал ее для тех, кто уже имеет некоторый опыт программирования и желает изучить JavaScript, а также для программистов, уже использующих JavaScript, но стремящихся подняться на более высокий уровень мастерства и по-настоящему овладеть языком и веб-платформой. Моя цель состояла в том, чтобы максимально полно и подробно описать JavaScript и платформу. В результате получилась эта объемная и подробная книга. Однако смею надеяться, что вы будете вознаграждены за внимательное изучение книги и время, потраченное на ее чтение, будет компенсировано более высокой производительностью труда.

Книга делится на четыре части. Часть I охватывает сам язык JavaScript. Часть II охватывает клиентский JavaScript: прикладные программные интерфейсы JavaScript, определяемые стандартом HTML5 и сопутствующими ему стандартами и реализованные в веб-броузерах. Часть III представляет собой справочник по базовому языку, а часть IV – справочник по клиентскому JavaScript. Глава 1 включает краткий обзор глав первой и второй частей книги (раздел 1.1).

Это шестое издание книги охватывает стандарты ECMAScript 5 (последняя версия спецификации базового языка) и HTML5 (последняя версия спецификации веб-платформы). Положения стандарта ECMAScript 5 будут рассматриваться на протяжении всей первой части. Нововведения, появившиеся в HTML5, в основном будут обсуждаться в конце части II, но мы будем рассматривать их и в других главах. Совершенно новыми в этом издании являются глава 11 «Подмножества и расширения JavaScript», глава 12 «Серверный JavaScript», глава 19 «Библиотека jQuery» и глава 22 «Прикладные интерфейсы HTML5».

Читатели предыдущих изданий могут заметить, что в этом издании я полностью переписал многие главы. Главы первой части книги, посвященные основам языка и охватывающие объекты, массивы, функции и классы, были переписаны заново и приведены в соответствие с современными приемами программирования. Ключевые главы второй части, описывающие документы и события, точно так же были полностью переписаны, чтобы привести их к современному уровню.

Несколько слов о пиратстве

Если вы читаете электронную версию этой книги, за которую вы (или ваш работодатель) ничего не платили (или позаимствовали ее у третьего лица, не заплатившего за книгу), то, скорее всего, эта копия является пиратской. Работа над шестым изданием продолжалась более года, и мне приходилось трудиться над книгой полный рабочий день в течение всего этого времени. Оплату за свой труд я получаю, лишь когда кто-то покупает эту книгу. И единственным источником дохода, который позволит мне продолжить работу над седьмым изданием, является гонорар от продажи шестого издания.

Я не приветствую пиратство, но, если вы читаете пиратскую копию, прочитайте несколько глав. Это позволит вам убедиться, что данная книга является ценным источником информации о JavaScript, лучше организованным и более качественным, чем бесплатные (и законные) источники информации, доступные в Веб. Если вы согласитесь с тем, что эта книга является ценным источником информации, пожалуйста, заплатите за эту ценность, приобретя легальную копию книги (электронную или бумажную). Если же вы посчитаете, что эта книга ничуть не лучше открытых источников информации в Веб, пожалуйста, уничтожьте пиратскую копию и пользуйтесь открытыми источниками информации.

Типографские соглашения

В этой книге приняты следующие типографские соглашения:

Курсив

Обозначает первое вхождение термина. *Курсив* также применяется для выделения адресов электронной почты, адресов URL и имен файлов и каталогов.

`Моноширинный шрифт`

Применяется для форматирования программного кода на языке JavaScript, листингов CSS и HTML и вообще всего, что непосредственно набирается на клавиатуре при программировании.

`Моноширинный курсив`

Обозначает аргументы функций и любые другие элементы, которые в программе необходимо заменить реальными значениями.

Использование программного кода примеров

Примеры для этой книги доступны в электронном виде. Соответствующие ссылки можно найти на странице книги на веб-сайте издательства:

http://oreilly.com/catalog/9780596805531/

Данная книга призвана оказать помощь в решении ваших задач. Вы можете свободно использовать примеры программного кода из этой книги в своих приложениях и в документации. Вам не нужно обращаться в издательство за разрешением,

если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и задействуете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам *необходимо* получить разрешение от издательства O'Reilly. При цитировании данной книги или примеров из нее и при ответе на вопросы получение разрешения не требуется. При включении существенных объемов программного кода примеров из этой книги в вашу документацию вам *необходимо* получить разрешение издательства.

Если вы соберетесь использовать программный код из этой книги, я приветствую, но не требую добавлять ссылку на первоисточник при цитировании. Под ссылкой подразумевается указание авторов, издательства и ISBN. Например: «JavaScript: The Definitive Guide, by David Flanagan (O'Reilly). Copyright 2011 David Flanagan, 978-0-596-80552-4».

Дополнительную информацию о порядке использования программного кода примеров можно найти на странице *http://oreilly.com/pub/a/oreilly/ask_tim/2001/codepolicy.html*. За получением разрешения на использование значительных объмов программного кода примеров из этой книги обращайтесь по адресу *permissions@oreilly.com*.

Ошибки и контактная информация

Издательство на своем сайте публикует список ошибок, найденных в книге. Вы можете ознакомиться со списком и отправить свои замечания об обнаруженных вами ошибках, посетив веб-страницу:

http://oreilly.com/catalog/9780596805531

С вопросами и предложениями технического характера, касающимися этой книги, обращайтесь по адресу:

bookquestions@oreilly.com

Дополнительную информацию о книгах, обсуждения, Центр ресурсов издательства O'Reilly вы найдете на веб-сайте:

http://www.oreilly.com

Можно найти нас на сайте Facebook: *http://facebook.com/oreilly*

Следить за последними новостями на сайте Twitter:

http://twitter.com/oreillymedia.

Просматривать видеоматериалы на сайте YouTube:

http://www.youtube.com/oreillymedia.

Благодарности

Работая над этой книгой, я получал помощь от многих людей. Я хотел бы поблагодарить моего редактора, Майка Лукидеса (Mike Loukides), который старался удержать меня в рамках планов и вносил полезные комментарии. Спасибо также моим техническим редакторам: Захару Кессину (Zachary Kessin), просмотрев-

шему многие главы в первой части, и Рафаэлю Цекко (Raffaele Cecco), который редактировал главу 19 и материал с описанием тега <canvas> в главе 21. Как обычно, блестяще справился со своей работой производственный отдел издательства O'Reilly: Дэн Фокссмит (Dan Fauxsmith), руководивший производственным процессом, Тереза Элси (Teresa Elsey), выполнявшая литературную правку, Роб Романо (Rob Romano), готовивший иллюстрации, и Элен Тротман Цайг (Ellen Troutman Zaig), создавшая алфавитный указатель.

В нашу эпоху широкого развития средств электронной коммуникации практически невозможно перечислить всех, кто оказывал помощь в том или ином виде. И мне хотелось бы поблагодарить всех, кто отвечал на мои вопросы, касающиеся ECMAScript 5, w3c, всех участников списков рассылки и всех, кто делился со мной своими идеями, касающимися программирования на JavaScript. Я глубоко сожалею, что не могу перечислить всех поименно, но хочу сказать, что мне было приятно работать с таким ярким сообществом программистов на JavaScript.

В работе над предыдущими изданиями книги принимали участие следующие редакторы и рецензенты: Эндрю Шульман (Andrew Schulman), Анжело Сиригос (Angelo Sirigos), Аристотель Пагальцис (Aristotle Pagaltzis), Брендан Эйх (Brendan Eich), Кристиан Хейльманн (Christian Heilmann), Дэн Шейфер (Dan Shafer), Дэйв С. Митчелл (Dave C. Mitchell), Деб Камерон (Deb Cameron), Дуглас Крокфорд (Douglas Crockford), д-р Танкред Хиршманн (Dr. Tankred Hirschmann), Дилан Шиман (Dylan Schiemann), Френк Уиллисон (Frank Willison), Джефф Штернс (Geoff Stearns), Герман Вентер (Herman Venter), Джей Ходжес (Jay Hodges), Джефф Ятс (Jeff Yates), Джозеф Кесселман (Joseph Kesselman), Кен Купер (Ken Cooper), Ларри Салливан (Larry Sullivan), Линн Роллинс (Lynn Rollins), Нил Беркман (Neil Berkman), Ник Томпсон (Nick Thompson), Норрис Бойд (Norris Boyd), Паула Фергюсон (Paula Ferguson), Питер-Пауль Кох (Peter-Paul Koch), Филипп Ле Хегарет (Philippe Le Hegaret), Ричард Якер (Richard Yaker), Сандерс Клейнфельд (Sanders Kleinfeld), Скотт Фурман (Scott Furman), Скотт Иссакс (Scott Issacs), Шон Каценбергер (Shon Katzenberger), Терри Аллен (Terry Allen), Тодд Дихендорф (Todd Ditchendorf), Вайдур Аппарао (Vidur Apparao) и Валдемар Хорват (Waldemar Horwat).

Это издание книги было в значительной степени переписано заново, из-за чего моя семья провела массу вечеров без меня. Хочу выразить им мою любовь и благодарность за то, что терпели мое отсутствие.

Дэвид Флэнаган (*http://www.davidflanagan.com*), март 2011

1

Введение в JavaScript

JavaScript – это язык программирования для Веб. Подавляющее большинство веб-сайтов используют JavaScript, и все современные веб-броузеры – для настольных компьютеров, игровых приставок, электронных планшетов и смартфонов – включают интерпретатор JavaScript, что делает JavaScript самым широкоприменимым языком программирования из когда-либо существовавших в истории. JavaScript входит в тройку технологий, которые должен знать любой веб-разработчик: язык разметки HTML, позволяющий определять содержимое веб-страниц, язык стилей CSS, позволяющий определять внешний вид веб-страниц, и язык программирования JavaScript, позволяющий определять поведение веб-страниц. Эта книга поможет вам овладеть языком программирования.

Если вы знаете другие языки программирования, вам может оказаться полезна информация, что JavaScript является высокоуровневым, динамическим, нетипизированным и интерпретируемым языком программирования, который хорошо подходит для программирования в объектно-ориентированном и функциональном стилях. Свой синтаксис JavaScript унаследовал из языка Java, свои первоклассные функции – из языка Scheme, а механизм наследования на основе прототипов – из языка Self. Но вам не требуется знать все эти языки или быть знакомыми с их терминологией для чтения этой книги и изучения JavaScript.

Название языка «JavaScript» может вводить в заблуждение. За исключением поверхностной синтаксической схожести, JavaScript полностью отличается от языка программирования Java. JavaScript давно перерос рамки языка сценариев, превратившись в надежный и эффективный универсальный язык программирования. Последняя версия языка (смотрите врезку) определяет множество новых особенностей, позволяющих использовать его для разработки крупномасштабного программного обеспечения.

JavaScript: названия и версии

JavaScript был создан в компании Netscape на заре зарождения Веб. Название «JavaScript» является торговой маркой, зарегистрированной компанией Sun Microsystems (ныне Oracle), и используется для обозначения реализации языка, созданной компанией Netscape (ныне Mozilla). Компания Netscape представила язык для стандартизации европейской ассоциации производителей компьютеров ECMA (European Computer Manufacturer's Association), но из-за юридических проблем с торговыми марками стандартизованная версия языка получила несколько неуклюжее название «ECMAScript». Из-за тех же юридических проблем версия языка от компании Microsoft получила официальное название «JScript». Однако на практике все эти реализации обычно называют JavaScript. В этой книге мы будем использовать название «ECMAScript» только для ссылки на стандарт языка.

В течение прошлого десятилетия все веб-броузеры предоставляли реализацию версии 3 стандарта ECMAScript, и в действительности разработчикам не было необходимости задумываться о номерах версий: стандарт языка был стабилен, а его реализации в веб-броузерах в значительной мере были совместимыми. Недавно вышла новая важная версия стандарта языка под названием ECMAScript 5, и к моменту написания этих строк производители броузеров приступили к созданию его реализации. Эта книга охватывает все нововведения, появившиеся в ECMAScript 5, а также все особенности, предусмотренные стандартом ECMAScript 3. Иногда вам будут встречаться названия этих версий языка, сокращенные до ES3 и ES5, а также название JavaScript, сокращенное до JS.

Когда речь заходит непосредственно о самом языке, при этом подразумеваются только версии 3 и 5 стандарта ECMAScript. (Четвертая версия стандарта ECMAScript разрабатывалась много лет, но из-за слишком амбициозных целей так и не была выпущена.) Однако иногда можно встретить упоминание о версии JavaScript, например: JavaScript 1.5 или JavaScript 1.8. Эти номера версий присваивались реализациям JavaScript, выпускаемым компанией Mozilla, причем версия 1.5 соответствует базовому стандарту ECMAScript 3, а более высокие версии включают нестандартные расширения (подробнее об этом рассказывается в главе 11). Наконец, номера версий также присваиваются отдельным интерпретаторам, или «механизмам» JavaScript. Например, компания Google разрабатывает свой интерпретатор JavaScript под названием V8, и к моменту написания этих строк текущей версией механизма V8 была версия 3.0.

Чтобы представлять хоть какой-то интерес, каждый язык программирования должен иметь свою платформу, или стандартную библиотеку, или API функций для выполнения таких базовых операций, как ввод и вывод. Ядро языка JavaScript определяет минимальный прикладной интерфейс для работы с текстом, массивами, датами и регулярными выражениями, но в нем отсутствуют операции ввода-вывода. Ввод и вывод (а также более сложные возможности, такие как

сетевые взаимодействия, сохранение данных и работа с графикой) перекладываются на «окружающую среду», куда встраивается JavaScript. Обычно роль окружающей среды играет веб-броузер (однако в главе 12 мы увидим два примера использования JavaScript без привлечения веб-броузера). Первая часть этой книги охватывает сам язык JavaScript и его минимальный прикладной интерфейс. Вторая часть описывает использование JavaScript в веб-броузерах и охватывает прикладной интерфейс, предоставляемый броузерами, который иногда называют «клиентским JavaScript».

Третья часть книги представляет собой справочник по базовому API языка. Например, чтобы ознакомиться с прикладным интерфейсом JavaScript для работы с массивами, вы можете отыскать и прочитать раздел «Array» в этой части книги. Четвертая часть – это справочник по клиентскому JavaScript. Например, чтобы ознакомиться с прикладным интерфейсом JavaScript для работы с графикой, определяемым стандартом HTML5 для тега `<canvas>`, можно отыскать и прочитать раздел «Canvas» в четвертой части книги.

В этой книге мы сначала рассмотрим низкоуровневые основы, а затем перейдем к базирущимся на них высокоуровневым абстракциям. Желательно читать главы, придерживаясь порядка, в котором они следуют в книге. Однако изучение нового языка программирования никогда не было линейным процессом, точно так же и описание языка трудно представить в линейном виде: каждая особенность языка тесно связана с другими особенностями, поэтому данная книга полна перекрестных ссылок – иногда назад, а иногда вперед – на сведения, с которыми вы еще не ознакомились. Эта глава являет собой первый краткий обзор основ языка и прикладного интерфейса клиентского JavaScript и представляет ключевые особенности, чем упрощает более глубокое их изучение в последующих главах.

Исследование JavaScript

Изучая новый язык программирования, очень важно стараться пробовать запускать примеры, представленные в книге, изменять их и опять запускать, чтобы проверить, насколько правильно вы понимаете особенности языка. Для этого необходим интерпретатор JavaScript. К счастью, любой веб-броузер включает интерпретатор JavaScript, а если вы читаете эту книгу, у вас, скорее всего, на компьютере установлено более одного веб-броузера.

Далее в этой главе мы увидим, что код на языке JavaScript можно встраивать в HTML-файлы, в теги `<script>`, и при загрузке HTML-файла этот код будет выполняться броузером. К счастью, нам не требуется поступать так всякий раз, когда нужно опробовать короткий фрагмент программного кода JavaScript. Появление мощного и оригинального расширения Firebug для Firefox (изображено на рис. 1.1 и доступно для загрузки на сайте *http://getfirebug.com/*) подтолкнуло производителей веб-броузеров к включению в них инструментов веб-разработчика, необходимых для отладки, проверки и изучения. Обычно эти инструменты можно отыскать в меню Tools (Инструменты или Сервис) броузера в виде пункта Developer Tools (Средства разработчика)

или Web Console (Веб-консоль). (Броузер Firefox 4 включает собственный встроенный инструмент Web Console, но к настоящему моменту расширение Firebug обладает более широкими возможностями.) Как правило, консоль можно запустить нажатием горячей комбинации клавиш, такой как F12 или Ctrl-Shift-J. Обычно эти инструменты открываются в виде отдельной панели в верхней или нижней части окна броузера, но некоторые броузеры открывают их в отдельном окне (как изображено на рис. 1.1), что часто более удобно.

Панель или окно типичного «инструмента разработчика» включает множество вкладок, позволяющих исследовать структуру HTML-документа, стили CSS, наблюдать за выполнением сетевых запросов и т. д. Среди них имеется вкладка JavaScript console (Консоль JavaScript), где можно вводить строки программного кода JavaScript и выполнять их. Это самый простой способ поэкспериментировать с JavaScript, и я рекомендую использовать его во время чтения этой книги.

В современных броузерах имеется простой переносимый API консоли. Для вывода текста в консоль можно использовать функцию `console.log()`. Зачастую такая возможность оказывается удивительно полезной при отладке, и некоторые примеры из этой книги (даже в разделе, посвященном базовому языку) используют `console.log()` для вывода простого текста. Похожий, но более навязчивый способ вывода информации или отладочных сообщений заключается в передаче строки текста функции `alert()`, которая отображает его в окне модального диалога.

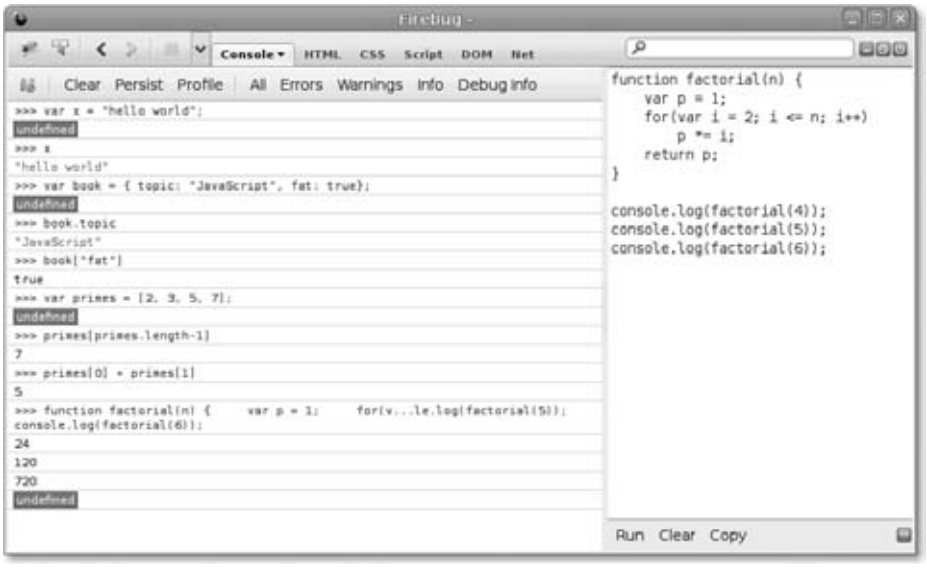


Рис. 1.1. *Отладочная консоль расширения Firebug для Firefox*

1.1. Базовый JavaScript

Этот раздел представляет собой обзор языка JavaScript, а также обзор первой части этой книги. После этой вводной главы мы опустимся на самый нижний уровень JavaScript: в главе 2 «Лексическая структура» будут описаны основные лексические конструкции JavaScript, такие как комментарии, точки с запятой и набор символов Юникода. В главе 3 «Типы данных, значения и переменные» мы начнем рассматривать более интересные темы: здесь будут описаны переменные JavaScript и значения, которые можно присваивать этим переменным. Ниже приводится пример программного кода, иллюстрирующий предмет обсуждения этих двух глав:

```
// Все, что следует за двумя символами слэша, является комментарием.
// Внимательно читайте комментарии: они описывают программный код JavaScript.

// Переменная - это символическое имя некоторого значения.
// Переменные объявляются с помощью ключевого слова var:
var x;          // Объявление переменной с именем x.

// Присваивать значения переменным можно с помощью знака =
x = 0;          // Теперь переменная x имеет значение 0
x               // => 0: В выражениях имя переменной замещается ее значением.

// JavaScript поддерживает значения различных типов
x = 1;                      // Числа.
x = 0.01;                   // Целые и вещественные числа представлены одним типом.
x = "hello world";          // Строки текста в кавычках.
x = 'JavaScript';           // Строки можно также заключать в апострофы.
x = true;                   // Логические значения.
x = false;                  // Другое логическое значение.
x = null;                   // null - особое значение, обозначающее "нет значения".
x = undefined;              // Значение undefined подобно значению null.
```

Двумя другими очень важными *типами* данных, которыми могут манипулировать программы на JavaScript, являются объекты и массивы. Они будут рассматриваться в главе 6 «Объекты» и в главе 7 «Массивы» однако они настолько важны, что вы не раз встретитесь с ними, прежде чем дойдете до этих глав.

```
// Наиболее важным типом данных в JavaScript являются объекты.
// Объект - это коллекция пар имя/значение или отображение строки в значение.
var book = {                // Объекты заключаются в фигурные скобки.
    topic: "JavaScript",    // Свойство "topic" имеет значение "JavaScript".
    fat: true               // Свойство "fat" имеет значение true.
};                          // Фигурная скобка отмечает конец объекта.

// Доступ к свойствам объектов выполняется с помощью . или []:
book.topic                  // => "JavaScript"
book["fat"]                 // => true: другой способ получить значение свойства.
book.author = "Flanagan";   // Создать новое свойство присваиванием.
book.contents = {};         // {} - пустой объект без свойств.

// JavaScript поддерживает массивы (списки с числовыми индексами) значений:
var primes = [2, 3, 5, 7]; // Массив из 4 значений, ограничивается [ и ].
primes[0]                   // => 2: первый элемент (с индексом 0) массива.
primes.length               // => 4: количество элементов в массиве.
primes[primes.length-1]     // => 7: последний элемент массива.
primes[4] = 9;              // Добавить новый элемент присваиванием.
```

```
primes[4] = 11;                 // Или изменить значение имеющегося элемента.
var empty = [];                 // [] - пустой массив без элементов.
empty.length                    // => 0

// Массивы и объекты могут хранить другие массивы и объекты:
var points = [                  // Массив с 2 элементами.
    {x:0, y:0},                 // Каждый элемент - это объект.
    {x:1, y:1}
];
var data = {                    // Объект с 2 свойствами
    trial1: [[1,2], [3,4]],     // Значение каждого свойства - это массив.
    trial2: [[2,3], [4,5]]      // Элементами массива являются массивы.
};
```

Синтаксические конструкции, представленные выше и содержащие списки элементов массивов в квадратных скобках или отображения свойств объектов в значения внутри фигурных скобок, часто называют *выражениями инициализации*, которые будут рассматриваться в главе 4 «Выражения и операторы». *Выражение* – это фраза на языке JavaScript, которую можно вычислить, чтобы получить значение. Например, применение . и [] для ссылки на значение свойства объекта или элемента массива является выражением. Возможно, вы заметили, что в листинге, приведенном выше, в строках, содержащих только выражение, комментарии начинаются со стрелки (=>), за которой следует значение выражения. Этому соглашению мы будем следовать на протяжении всей книги.

Наиболее типичным способом формирования выражений в JavaScript является использование *операторов*, подобно тому, как показано ниже:

```
// Операторы выполняют действия со значениями (операндами) и воспроизводят
// новое значение. Наиболее часто используемыми являются арифметические операторы:
3 + 2                         // => 5: сложение
3 - 2                         // => 1: вычитание
3 * 2                         // => 6: умножение
3 / 2                         // => 1.5: деление
points[1].x - points[0].x  // => 1: можно использовать более сложные операнды
"3" + "2"                     // => "32": + складывает числа, объединяет строки

// В JavaScript имеются некоторые сокращенные формы арифметических операторов
var count = 0;                // Объявление переменной
count++;                      // Увеличение значения переменной на 1
count--;                      // Уменьшение значения переменной на 1
count += 2;                   // Добавить 2: то же, что count = count + 2;
count *= 3;                   // Умножить на 3: то же, что count = count * 3;
count                         // => 6: имена переменных сами являются выражениями

// Операторы сравнения позволяют проверить два значения на равенство
// или неравенство, выяснить, какое значение меньше или больше, и т. д.
// Они возвращают значение true или false.
var x = 2, y = 3;             // Знаки = выполняют присваивание, а не сравнение
x == y                        // => false: равенство
x != y                        // => true: неравенство
x < y                         // => true: меньше
x <= y                        // => true: меньше или равно
x > y                         // => false: больше
x >= y                        // => false: больше или равно
"two" == "three"              // => false: две разных строки
```

```
"two" > "three"     // => true: при упорядочении по алфавиту строка "tw" больше, чем "th"
false == (x > y)    // => true: false равно false

// Логические операторы объединяют или инвертируют логические значения
(x == 2) && (y == 3)       // => true: оба сравнения истинны. && - "И"
(x > 3) || (y < 3)         // => false: оба сравнения ложны. || - "ИЛИ"
!(x == y)                  // => true: ! инвертирует логическое значение
```

Если фразы в языке JavaScript называются выражениями, то полные предложения называются *инструкциями*; они рассматриваются в главе 5 «Инструкции». В программном коде, приведенном выше, строки, заканчивающиеся точками с запятой, являются инструкциями. (В примере ниже можно увидеть инструкции, состоящие из нескольких строк, которые не завершаются точками с запятой.) Между инструкциями и выражениями много общего. Грубо говоря, выражение – это конструкция, которая вычисляет значение, но *ничего не делает*: она никак не изменяет состояние программы. Инструкции, напротив, не имеют значения (или их значение не представляет интереса для нас), но они изменяют состояние программы. Выше уже были показаны инструкции объявления переменных и присваивания значений. Еще одной обширной категорией инструкций являются *управляющие конструкции*, такие как условные инструкции и инструкции циклов. Примеры этих инструкций будут показаны далее, после того, как мы познакомимся с функциями.

Функция – это именованный и параметризованный блок программного кода JavaScript, который определяется один раз, а использоваться может многократно. Формальное знакомство с функциями мы отложим до главы 8 «Функции», однако, как и в случае с объектами и массивами, мы много раз встретимся с функциями, прежде чем доберемся до этой главы. Ниже приводятся несколько примеров простых функций:

```
// Функции - это параметризованные блоки программного кода JavaScript,
// которые можно вызывать многократно.
function plus1(x) {        // Определить функцию с именем "plus1" и с параметром "x"
    return x+1;            // Вернуть значение на 1 больше полученного
}                          // Функции заключаются в фигурные скобки

plus1(y)                   // => 4: y имеет значение 3, поэтому этот вызов вернет 3+1

var square = function(x) { // Функции можно присваивать переменным
    return x*x;            // Вычислить значение функции
};                         // Точка с запятой отмечает конец присваивания.

square(plus1(y))           // => 16: вызов двух функций в одном выражении
```

При объединении функций с объектами получаются *методы*:

```
// Функции, присвоенные свойствам объектов, называются методами.
// Все объекты в JavaScript имеют методы:
var a = [];                // Создать пустой массив
a.push(1,2,3);             // Метод push() добавляет элементы в массив
a.reverse();               // Другой метод: переставляет элементы в обратном порядке

// Можно определять собственные методы. Ключевое слово "this" ссылается на объект,
// в котором определен метод: в данном случае на массив points.
points.dist = function() { // Метод вычисления расстояния между точками
    var p1 = this[0];      // Первый элемент массива, относительно которого вызван метод
```

```
    var p2 = this[1];       // Второй элемент объекта "this"
    var a = p2.x-p1.x;      // Разность координат X
    var b = p2.y-p1.y;      // Разность координат Y
    return Math.sqrt(a*a + // Теорема Пифагора
                     b*b); // Math.sqrt() вычисляет корень квадратный
};
points.dist()               // => 1.414: расстояние между 2-мя точками
```

Теперь, как было обещано, рассмотрим несколько функций, которые демонстрируют применение наиболее часто используемых управляющих инструкций JavaScript:

```
// В JavaScript имеются условные инструкции и инструкции циклов, синтаксически
// похожие на аналогичные инструкции C, C++, Java и в других языках.
function abs(x) {           // Функция, вычисляющая абсолютное значение
    if (x >= 0) {           // Инструкция if ...
        return x;           // выполняет этот код, если сравнение дает true.
    }                       // Конец предложения if.
    else {                  // Необязательное предложение else выполняет свой код,
        return -x;          // если сравнение дает значение false.
    }                       // Фигурные скобки можно опустить, если предложение
                            // содержит 1 инструкцию.
}                           // Обратите внимание на инструкции return внутри if/else.

function factorial(n) {    // Функция, вычисляющая факториал
    var product = 1;       // Начать с произведения, равного 1
    while(n > 1) {         // Повторять инструкции в {}, пока выраж. в () истинно
        product *= n;      // Сокращенная форма выражения product = product * n;
        n--;               // Сокращенная форма выражения n = n - 1
    }                      // Конец цикла
    return product;        // Вернуть произведение
}
factorial(4)               // => 24: 1*4*3*2

function factorial2(n) {   // Другая версия, использующая другой цикл
    var i, product = 1;    // Начать с 1
    for(i=2; i <= n; i++)  // i автоматически увеличивается с 2 до n
        product *= i;      // Выполнять в каждом цикле. {} можно опустить,
                           // если тело цикла состоит из 1 инструкции
    return product;        // Вернуть факториал
}
factorial2(5)              // => 120: 1*2*3*4*5
```

JavaScript – объектно-ориентированный язык, но используемая в нем объектная модель в корне отличается от модели, используемой в большинстве других языков. В главе 9 «Классы и модули» детально рассматривается объектно-ориентированное программирование на языке JavaScript на большом количестве примеров; эта глава является одной из самых больших в книге. Ниже приводится очень простой пример, демонстрирующий определение класса JavaScript для представления точек на плоскости. Объекты, являющиеся экземплярами этого класса, обладают единственным методом с методом `r()`, который вычисляет расстояние между данной точкой и началом координат:

```
// Определение функции-конструктора для инициализации нового объекта Point
function Point(x,y) { // По соглашению имя конструкторов начинается с заглавного символа
    this.x = x;       // this - ссылка на инициализируемый объект
```

```
    this.y = y;           // Сохранить аргументы в свойствах объекта
}                         // Ничего возвращать не требуется

// Чтобы создать новый экземпляр, необходимо вызвать функцию-конструктор
// с ключевым словом "new"
var p = new Point(1, 1);  // Точка на плоскости с координатами (1,1)

// Методы объектов Point определяются за счет присваивания функций свойствам
// объекта-прототипа, ассоциированного с функцией-конструктором.
Point.prototype.r = function() {
    return Math.sqrt(     // Вернуть корень квадратный от x² + y²
        this.x * this.x + // this - это объект Point, относительно которого...
        this.y * this.y   // ...вызывается метод.
    );
};

// Теперь объект p типа Point (и все последующие объекты Point) наследует метод r()
p.r()                     // => 1.414...
```

Глава 9 является кульминацией первой части, а главы, которые следуют за ней, связывают некоторые оборванные концы и завершают исследование базового языка. В главе 10 «Шаблоны и регулярные выражения» описывается грамматика регулярных выражений и демонстрируются приемы использования регулярных выражений для реализации сопоставления с текстовыми шаблонами. В главе 11 «Подмножества и расширения JavaScript» рассматриваются подмножества и расширения базового языка JavaScript. Наконец, прежде чем перейти к исследованию клиентского JavaScript в веб-броузерах, в главе 12 «Серверный JavaScript» будут представлены два способа использования JavaScript за пределами веб-броузеров.

1.2. Клиентский JavaScript

Изучение клиентского JavaScript представляет собой задачу, нелинейную из-за перекрестных ссылок в значительно меньшей мере, чем базовый язык, и поэтому вполне возможно изучать особенности использования JavaScript в веб-броузерах в линейном порядке. Возможно, вы взялись за чтение этой книги, чтобы изучить клиентский JavaScript – тему далекой второй части, поэтому здесь мы приводим краткий обзор основных приемов программирования клиентских сценариев, который сопровождается подробным примером.

Глава 13 «JavaScript в веб-броузерах» является первой главой второй части, в которой описываются детали использования JavaScript в веб-броузерах. Самое важное, что вы узнаете в этой главе, – программный код JavaScript может встраиваться в HTML-файлы с помощью тега `<script>`:

```
<html>
<head>
<script src="library.js"></script> <!-- подключить библиотеку JavaScript -->
</head>
<body>
<p>Это абзац HTML</p>
<script>
// Это некоторый программный код на клиентском JavaScript,
// встроенный непосредственно в HTML-файл
</script>
```

```
<p>Далее опять следует разметка HTML.</p>
</body>
</html>
```

Глава 14 «Объект Window» исследует приемы управления веб-броузером и описывает некоторые наиболее важные глобальные функции клиентского JavaScript. Например:

```
<script>
function moveon() {
    // Вывести модальный диалог, чтобы получить ответ пользователя
    var answer = confirm("Ready to move on?");
    // Если пользователь щелкнул на кнопке "OK", заставить броузер загрузить новую страницу
    if (answer) window.location = "http://google.com";
}
// Запустить функцию, объявленную выше, через 1 минуту (60000 миллисекунд).
setTimeout(moveon, 60000);
</script>
```

Обратите внимание, что примеры программного кода на клиентском JavaScript в этом разделе длиннее примеров на базовом языке, которые мы видели выше в этой главе. Эти примеры не предназначены для ввода в окне консоли Firebug (или в другом подобном инструменте). Однако вы можете вставлять их в HTML-файлы и затем запускать, открывая файлы в веб-броузере. Так, пример, приведенный выше, является самостоятельным HTML-файлом.

Глава 15 «Работа с документами» переходит к исследованию фактической работы, выполняемой с помощью JavaScript на стороне клиента, – управлению содержимым документа HTML. Она покажет вам, как выбирать определенные элементы HTML из документов, как устанавливать HTML-атрибуты этих элементов, как изменять содержимое элементов и как добавлять в документ новые элементы. Следующая функция демонстрирует некоторые из простейших приемов поиска и изменения элементов документа:

```
// Выводит сообщение в специальной области для отладочных сообщений внутри документа.
// Если документ не содержит такой области, она создается.
function debug(msg) {
    // Отыскать область для отладочных сообщений в документе, поиск по HTML-атрибуту id
    var log = document.getElementById("debuglog");

    // Если элемент с атрибутом id="debuglog" отсутствует, создать его.
    if (!log) {
        log = document.createElement("div");  // Создать элемент <div>
        log.id = "debuglog";                  // Установить атрибут id
        log.innerHTML = "<h1>Debug Log</h1>"; // Начальное содержимое
        document.body.appendChild(log);       // Добавить в конец документа
    }

    // Теперь обернуть сообщение в теги <pre> и добавить в элемент log
    var pre = document.createElement("pre");  // Создать тег <pre>
    var text = document.createTextNode(msg);  // Обернуть msg в текстовый узел
    pre.appendChild(text);                    // Добавить текст в тег <pre>
    log.appendChild(pre);                     // Добавить <pre> в элемент log
}
```

Глава 15 демонстрирует, как с помощью JavaScript можно управлять HTML-элементами, которые определяют содержимое веб-страниц. Глава 16 «Каскадные таб-

лицы стилей» демонстрирует, как с помощью JavaScript можно управлять каскадными таблицами стилей CSS, определяющими представление содержимого. Чаще всего для этой цели используется атрибут HTML-элементов `style` или `class`:

```
function hide(e, reflow) {             // Скрывает элемент e, изменяя его стиль
    if (reflow) {                      // Если 2-й аргумент true,
        e.style.display = "none"       // скрыть элемент и использовать
    }                                  // занимаемое им место
    else {                             // Иначе
        e.style.visibility = "hidden"; // сделать e невидимым, но оставить
    }                                  // занимаемое им место пустым
}

function highlight(e) { // Выделяет e, устанавливая класс CSS
    // Просто добавляет или переопределяет HTML-атрибут class.
    // Предполагается, что таблица стилей CSS уже содержит определение класса "hilite"
    if (!e.className) e.className = "hilite";
    else e.className += " hilite";
}
```

JavaScript позволяет не только управлять содержимым и оформлением HTML-документов в броузерах, но и определять поведение этих документов с помощью *обработчиков событий*. Обработчик событий – это функция JavaScript, которая регистрируется в броузере и вызывается броузером, когда возникает событие определенного типа. Таким событием может быть щелчок мышью или нажатие клавиши (или какое-то движение двумя пальцами на экране смартфона). Обработчик события может также вызываться броузером по окончании загрузки документа, при изменении размеров окна броузера или при вводе данных в элемент HTML-формы. Глава 17 «Обработка событий» описывает, как определять и регистрировать обработчики событий и как вызываются эти обработчики при появлении событий.

Простейший способ объявления обработчиков событий заключается в использовании HTML-атрибутов, имена которых начинаются с приставки «on». Обработчик «onclick» особенно удобен при создании простых тестовых программ. Предположим, что вы сохранили функции `debug()` и `hide()`, представленные выше, в файлах с именами *debug.js* и *hide.js*. В этом случае можно было бы написать простой тестовый HTML-файл, использующий элементы `<button>` с атрибутами `onclick`, определяющими обработчики событий:

```
<script src="debug.js"></script>
<script src="hide.js"></script>
Hello
<button onclick="hide(this,true); debug('hide button 1');">Hide1</button>
<button onclick="hide(this); debug('hide button 2');">Hide2</button>
World
```

Ниже приводится еще один пример программного кода на клиентском JavaScript, использующего механизм событий. Он регистрирует обработчик очень важного события «load» и дополнительно демонстрирует более сложный способ регистрации обработчика события «click»:

```
// Событие "load" возбуждается, когда документ будет полностью загружен.
// Обычно мы вынуждены ждать этого события, прежде чем можно будет запустить
// наш программный код JavaScript.
window.onload = function() { // Запустит функцию после загрузки документа
```

```
    // Отыскать все теги <img> в документе
    var images = document.getElementsByTagName("img");

    // Обойти их все в цикле, добавить к каждому обработчик события "click",
    // чтобы обеспечить сокрытие любого изображения после щелчка на нем.
    for(var i = 0; i < images.length; i++) {
        var image = images[i];
        if (image.addEventListener)    // Другой способ регистрации обработчика
            image.addEventListener("click", hide, false);
        else                           // Для совместимости с версией IE8 и ниже
            image.attachEvent("onclick", hide);
    }

    // Это функция-обработчик событий, которая регистрируется выше
    function hide(event) { event.target.style.visibility = "hidden"; }
};
```

Главы 15, 16 и 17 описывают, как с помощью JavaScript управлять содержимым (HTML), представлением (CSS) и поведением (обработка событий) веб-страниц. Прикладной интерфейс, описываемый в этих главах, является достаточно сложным, и до недавнего времени испытывал проблемы с совместимостью между броузерами. По этим причинам многие или большинство программистов на клиентском JavaScript предпочитают использовать клиентские библиотеки или фреймворки, упрощающие программирование. Наиболее популярна из этих библиотек – библиотека jQuery, которая обсуждается в главе 19 «Библиотека jQuery». Библиотека jQuery определяет простой и удобный программный интерфейс для управления содержимым документа, его представлением и поведением. Она была тщательно протестирована и может использоваться во всех основных броузерах, включая довольно старые, такие как IE6.

Программный код, использующий jQuery, легко отличить по частому использованию функции `$()`. Ниже показано, как будет выглядеть функция `debug()`, представленная выше, если переписать ее с использованием jQuery:

```
function debug(msg) {
    var log = $("#debuglog");          // Отыскать элемент для вывода msg.
    if (log.length == 0) {             // Если отсутствует, создать его...
        log = $("<div id='debuglog'><h1>Debug Log</h1></div>");
        log.appendTo(document.body);   // и вставить в конец тела документа.
    }
    log.append($("<pre/>").text(msg)); // Завернуть msg в тег <pre>
}                                      // и добавить в элемент log
```

В этих четырех главах из второй части в действительности рассматривается все, что касается *веб-страниц*. Другие четыре главы переключают внимание на *веб-приложения*. Эти главы не о том, как использовать веб-броузеры для отображения документов, содержимое, представление и поведение которых управляется с помощью JavaScript. Они рассказывают об использовании веб-броузеров как прикладной платформы и описывают прикладной интерфейс, предоставляемый современными броузерами для поддержки сложных, современных клиентских веб-приложений. Глава 18 «Работа с протоколом HTTP» описывает, как с помощью JavaScript можно управлять HTTP-запросами – своего рода сетевой прикладной интерфейс. Глава 20 «Сохранение данных на стороне клиента» описывает механизмы, позволяющие сохранять данные (и даже целые приложения) на стороне клиента для ис-

пользования в последующих сеансах работы. Глава 21 «Работа с графикой и медиафайлами на стороне клиента» охватывает клиентский прикладной интерфейс, позволяющий создавать произвольные графические изображения в HTML-теге <canvas>. И наконец, глава 22 «Прикладные интерфейсы HTML5» охватывает новые прикладные интерфейсы веб-приложений, определяемые или принятые стандартом HTML5. Сетевые взаимодействия, организация хранения данных, работа с графикой – все эти службы операционных систем, доступные посредством веб-броузеров, образуют новую, платформонезависимую среду выполнения приложений. Если вы нацелены на броузеры, которые поддерживают эти новые прикладные интерфейсы, то сейчас наступает самое интересное время для программистов на клиентском JavaScript. Здесь не приводятся примеры программного кода из этих заключительных четырех глав, однако расширенный пример, представленный ниже, использует некоторые из этих новых прикладных интерфейсов.

1.2.1. Пример: калькулятор платежей по ссуде на JavaScript

Эта глава завершается расширенным примером, объединяющим в себе многие из описанных выше приемов и демонстрирующим полноценную программу на клиентском JavaScript (плюс HTML и CSS). В примере 1.1 представлена реализация простого калькулятора для вычисления платежей по ссуде (рис. 1.2).

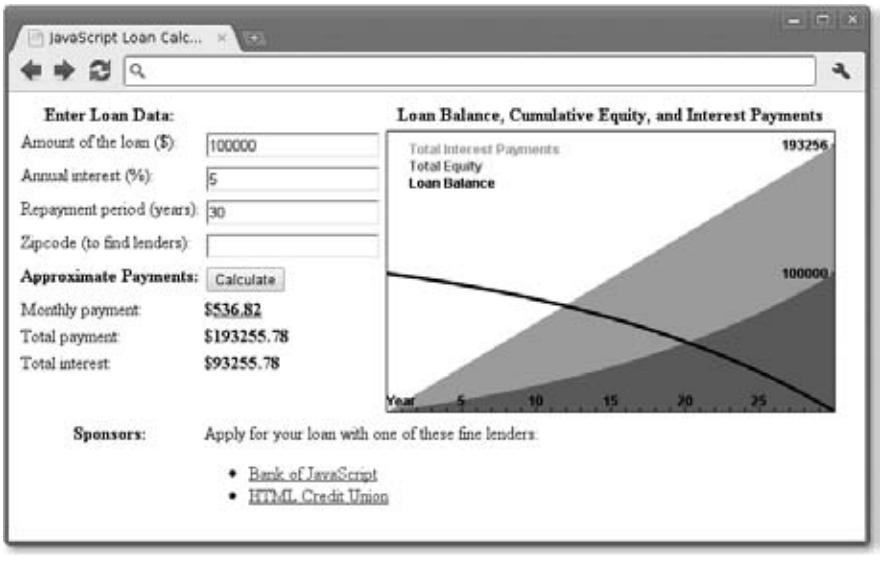


Рис. 1.2. *Веб-приложение – калькулятор платежей по ссуде*

Стоит потратить время на внимательное рассмотрение примера 1.1. Вряд ли вы сумеете досконально разобраться в нем, однако благодаря подробным комментариям вы должны по крайней мере получить общее представление о том, как действует это веб-приложение. Пример демонстрирует множество особенностей базового языка JavaScript, а также некоторые важные приемы программирования на клиентском JavaScript:

- **Поиск элементов в документе.**
- **Получение ввода пользователя с помощью элементов форм.**
- **Изменение содержимого элементов документа.**
- **Сохранение данных в броузере.**
- **Управление HTTP-запросами.**
- **Создание графики с помощью элемента** `<canvas>`**.**

Пример 1.1. Калькулятор вычисления платежей по ссуде на JavaScript

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Loan Calculator</title>
<style>  /* Таблица стилей CSS: определяет внешний вид вывода программы */
.output { font-weight: bold; }           /* Жирный шрифт для вычисленных значений */
#payment { text-decoration: underline; } /* Для элементов с id="payment" */
#graph { border: solid black 1px; }      /* Простая рамка для диаграммы */
th, td { vertical-align: top; }          /* Выравнивание в ячейках таблицы */
</style>
</head>
<body>
<!--
  Это HTML-таблица с элементами <input>, позволяющими вводить данные, и с элементами
  <span>, в которых отображаются результаты вычислений. Эти элементы имеют идентификаторы,
  такие как "interest" и "years". Данные идентификаторы используются в JavaScript-коде,
  который следует за определением таблицы. Обратите внимание, что для некоторых элементов
  ввода определены обработчики событий "onchange" и "onclick". В них заданы
  строки JavaScript-кода, выполняемого при вводе данных или щелчке на кнопке.
-->
<table>
  <tr><th>Enter Loan Data:</th>
    <td></td>
    <th>Loan Balance, Cumulative Equity, and Interest Payments</th></tr>
  <tr><td>Amount of the loan ($):</td>
    <td><input id="amount" onchange="calculate();"></td>
    <td rowspan=8>
      <canvas id="graph" width="400" height="250"></canvas></td></tr>
  <tr><td>Annual interest (%):</td>
    <td><input id="apr" onchange="calculate();"></td></tr>
  <tr><td>Repayment period (years):</td>
    <td><input id="years" onchange="calculate();"></td>
  <tr><td>Zipcode (to find lenders):</td>
    <td><input id="zipcode" onchange="calculate();"></td>
  <tr><th>Approximate Payments:</th>
    <td><button onclick="calculate();">Calculate</button></td></tr>
  <tr><td>Monthly payment:</td>
    <td>$<span class="output" id="payment"></span></td></tr>
  <tr><td>Total payment:</td>
    <td>$<span class="output" id="total"></span></td></tr>
  <tr><td>Total interest:</td>
    <td>$<span class="output" id="totalinterest"></span></td></tr>
  <tr><th>Sponsors:</th><td colspan=2>
    Apply for your loan with one of these fine lenders:
```

```
    <div id="lenders"></div></td></tr>
</table>

<!-- Остальная часть примера - JavaScript-код в теге <script> ниже. Обычно сценарии -->
<!-- помещаются в начало документа, в заголовок <head>, но в данном случае вам проще -->
<!-- будет понять пример, если JavaScript-код будет находиться ниже HTML-содержимого. -->
<script>
"use strict"; // Использовать строгий режим ECMAScript 5, если броузер поддерживает его
/*
 * Этот сценарий определяет функцию calculate(), вызываемую обработчиками событий
 * в разметке HTML выше. Функция читает значения из элементов <input>, вычисляет размеры
 * платежей по ссуде, отображает результаты в элементах <span>. Кроме того, она сохраняет
 * пользовательские данные, отображает ссылки на кредитные учреждения и рисует диаграмму.
 */
function calculate() {
    // Отыскать элементы ввода и вывода в документе
    var amount = document.getElementById("amount");
    var apr = document.getElementById("apr");
    var years = document.getElementById("years");
    var zipcode = document.getElementById("zipcode");
    var payment = document.getElementById("payment");
    var total = document.getElementById("total");
    var totalinterest = document.getElementById("totalinterest");

    // Получить ввод пользователя из элементов ввода. Предполагается, что все данные
    // являются корректными. Преобразовать процентную ставку из процентов
    // в десятичное число и преобразовать годовую ставку в месячную ставку.
    // Преобразовать период платежей в годах в количество месячных платежей.
    var principal = parseFloat(amount.value);
    var interest = parseFloat(apr.value) / 100 / 12;
    var payments = parseFloat(years.value) * 12;

    // Теперь вычислить сумму ежемесячного платежа.
    var x = Math.pow(1 + interest, payments); // Math.pow() вычисляет степень
    var monthly = (principal*x*interest)/(x-1);

    // Если результатом является конечное число, следовательно, пользователь
    // указал корректные данные и результаты можно отобразить
    if (isFinite(monthly)) {
        // Заполнить поля вывода, округлив результаты до 2 десятичных знаков
        payment.innerHTML = monthly.toFixed(2);
        total.innerHTML = (monthly * payments).toFixed(2);
        totalinterest.innerHTML = ((monthly*payments)-principal).toFixed(2);

        // Сохранить ввод пользователя, чтобы можно было восстановить данные
        // при следующем открытии страницы
        save(amount.value, apr.value, years.value, zipcode.value);

        // Реклама: отыскать и отобразить ссылки на сайты местных
        // кредитных учреждений, но игнорировать сетевые ошибки
        try { // Перехватывать все ошибки, возникающие в этих фигурных скобках
            getLenders(amount.value, apr.value, years.value, zipcode.value);
        }
        catch(e) { /* И игнорировать эти ошибки */ }

        // В заключение вывести график изменения остатка по кредиту, а также
        // графики сумм, выплачиваемых в погашение кредита и по процентам
        chart(principal, interest, monthly, payments);
```

```
    }
    else {
        // Результат не является числом или имеет бесконечное значение,
        // что означает, что были получены неполные или некорректные данные.
        // Очистить все результаты, выведенные ранее.
        payment.innerHTML = ""; // Стереть содержимое этих элементов
        total.innerHTML = ""
        totalinterest.innerHTML = "";
        chart();                // При вызове без аргументов очищает диаграмму
    }
}

// Сохранить ввод пользователя в свойствах объекта localStorage. Значения этих свойств
// будут доступны при повторном посещении страницы. В некоторых броузерах (например,
// в Firefox) возможность сохранения не поддерживается, если страница открывается
// с адресом URL вида file://. Однако она поддерживается при открытии страницы через HTTP.
function save(amount, apr, years, zipcode) {
    if (window.localStorage) { // Выполнить сохранение, если поддерживается
        localStorage.loan_amount = amount;
        localStorage.loan_apr = apr;
        localStorage.loan_years = years;
        localStorage.loan_zipcode = zipcode;
    }
}

// Автоматически восстановить поля ввода при загрузке документа.
window.onload = function() {
    // Если броузер поддерживает localStorage и имеются сохраненные данные
    if (window.localStorage && localStorage.loan_amount) {
        document.getElementById("amount").value = localStorage.loan_amount;
        document.getElementById("apr").value = localStorage.loan_apr;
        document.getElementById("years").value = localStorage.loan_years;
        document.getElementById("zipcode").value = localStorage.loan_zipcode;
    }
};

// Передать ввод пользователя серверному сценарию, который может (теоретически) возвращать
// список ссылок на сайты местных кредитных учреждений, готовых предоставить кредит.
// Данный пример не включает фактическую реализацию такого сценария поиска кредитных
// учреждений. Но если такой сценарий уже имеется, данная функция могла бы работать с ним.
function getLenders(amount, apr, years, zipcode) {
    // Если броузер не поддерживает объект XMLHttpRequest, не делать ничего
    if (!window.XMLHttpRequest) return;

    // Отыскать элемент для отображения списка кредитных учреждений
    var ad = document.getElementById("lenders");
    if (!ad) return;                            // Выйти, если элемент отсутствует

    // Преобразовать ввод пользователя в параметры запроса в строке URL
    var url = "getLenders.php" +                // Адрес URL службы плюс
        "?amt=" + encodeURIComponent(amount) + // данные пользователя
        "&apr=" + encodeURIComponent(apr) +    // в строке запроса
        "&yrs=" + encodeURIComponent(years) +
        "&zip=" + encodeURIComponent(zipcode);

    // Получить содержимое по заданному адресу URL с помощью XMLHttpRequest
    var req = new XMLHttpRequest();             // Создать новый запрос
    req.open("GET", url);                       // Указать тип запроса HTTP GET для url
```

```
    req.send(null);                                 // Отправить запрос без тела

    // Перед возвратом зарегистрировать обработчик события, который будет вызываться
    // при получении HTTP-ответа от сервера. Такой прием асинхронного программирования
    // является довольно обычным в клиентском JavaScript.
    req.onreadystatechange = function() {
        if (req.readyState == 4 && req.status == 200) {
            // Если мы попали сюда, следовательно, был получен корректный HTTP-ответ
            var response = req.responseText;        // HTTP-ответ в виде строки
            var lenders = JSON.parse(response);     // Преобразовать в JS-массив

            // Преобразовать массив объектов lender в HTML-строку
            var list = "";
            for(var i = 0; i < lenders.length; i++) {
                list += "<li><a href='" + lenders[i].url + "'>" +
                    lenders[i].name + "</a>";
            }

            // Отобразить полученную HTML-строку в элементе,
            // ссылка на который была получена выше.
            ad.innerHTML = "<ul>" + list + "</ul>";
        }
    }
}

// График помесячного изменения остатка по кредиту, а также графики сумм,
// выплачиваемых в погашение кредита и по процентам в HTML-элементе <canvas>.
// Если вызывается без аргументов, просто очищает ранее нарисованные графики.
function chart(principal, interest, monthly, payments) {
    var graph = document.getElementById("graph");   // Ссылка на тег <canvas>
    graph.width = graph.width;                      // Магия очистки элемента canvas

    // Если функция вызвана без аргументов или броузер не поддерживает
    // элемент <canvas>, то просто вернуть управление.
    if (arguments.length == 0 || !graph.getContext) return;

    // Получить объект "контекста" для элемента <canvas>,
    // который определяет набор методов рисования
    var g = graph.getContext("2d"); // Рисование выполняется с помощью этого объекта
    var width = graph.width, height = graph.height; // Получить размер холста

    // Следующие функции преобразуют количество месячных платежей
    // и денежные суммы в пикселы
    function paymentToX(n) { return n * width/payments; }
    function amountToY(a) { return height-(a*height/(monthly*payments*1.05));}

    // Платежи - прямая линия из точки (0,0) в точку (payments,monthly*payments)
    g.moveTo(paymentToX(0), amountToY(0));          // Из нижнего левого угла
    g.lineTo(paymentToX(payments),                  // В правый верхний
             amountToY(monthly*payments));
    g.lineTo(paymentToX(payments), amountToY(0));   // В правый нижний
    g.closePath();                                  // И обратно в начало
    g.fillStyle = "#f88";                           // Светло-красный
    g.fill();                                       // Залить треугольник
    g.font = "bold 12px sans-serif";                // Определить шрифт
    g.fillText("Total Interest Payments", 20,20);   // Вывести текст в легенде

    // Кривая накопленной суммы погашения кредита не является линейной
    // и вывод ее реализуется немного сложнее
```

```
    var equity = 0;
    g.beginPath();                                  // Новая фигура
    g.moveTo(paymentToX(0), amountToY(0));          // из левого нижнего угла
    for(var p = 1; p <= payments; p++) {
        // Для каждого платежа выяснить долю выплат по процентам
        var thisMonthsInterest = (principal-equity)*interest;
        equity += (monthly - thisMonthsInterest);  // Остаток - погашение кред.
        g.lineTo(paymentToX(p),amountToY(equity)); // Линию до этой точки
    }
    g.lineTo(paymentToX(payments), amountToY(0));  // Линию до оси X
    g.closePath();                                  // И опять в нач. точку
    g.fillStyle = "green";                          // Зеленый цвет
    g.fill();                                       // Залить обл. под кривой
    g.fillText("Total Equity", 20,35);              // Надпись зеленым цветом

    // Повторить цикл, как выше, но нарисовать график остатка по кредиту
    var bal = principal;
    g.beginPath();
    g.moveTo(paymentToX(0),amountToY(bal));
    for(var p = 1; p <= payments; p++) {
        var thisMonthsInterest = bal*interest;
        bal -= (monthly - thisMonthsInterest);     // Остаток от погаш. по кредиту
        g.lineTo(paymentToX(p),amountToY(bal));    // Линию до этой точки
    }
    g.lineWidth = 3;                                // Жирная линия
    g.stroke();                                     // Нарисовать кривую графика
    g.fillStyle = "black";                          // Черный цвет для текста
    g.fillText("Loan Balance", 20,50);              // Элемент легенды

    // Нарисовать отметки лет на оси X
    g.textAlign="center";                           // Текст меток по центру
    var y = amountToY(0);                           // Координата Y на оси X
    for(var year=1; year*12 <= payments; year++) { // Для каждого года
        var x = paymentToX(year*12);                // Вычислить позицию метки
        g.fillRect(x-0.5,y-3,1,3);                  // Нарисовать метку
        if (year == 1) g.fillText("Year", x, y-5); // Подписать ось
        if (year % 5 == 0 && year*12 !== payments) // Числа через каждые 5 лет
            g.fillText(String(year), x, y-5);
    }

    // Суммы платежей у правой границы
    g.textAlign = "right";                          // Текст по правому краю
    g.textBaseline = "middle";                      // Центрировать по вертикали
    var ticks = [monthly*payments, principal];      // Вывести две суммы
    var rightEdge = paymentToX(payments);           // Координата X на оси Y
    for(var i = 0; i < ticks.length; i++) {         // Для каждой из 2 сумм
        var y = amountToY(ticks[i]);                // Определить координату Y
        g.fillRect(rightEdge-3, y-0.5, 3,1);        // Нарисовать метку
        g.fillText(String(ticks[i].toFixed(0)),     // И вывести рядом сумму.
                   rightEdge-5, y);
    }
}
</script>
</body>
</html>
```

I

Базовый JavaScript

Данная часть книги включает главы со 2 по 12, она описывает базовый язык JavaScript и задумана как справочник по языку JavaScript. Прочитав главы этой части один раз, вы, возможно, будете неоднократно возвращаться к ним, чтобы освежить в памяти более сложные особенности языка.

- Глава 2 «Лексическая структура»
- Глава 3 «Типы данных, значения и переменные»
- Глава 4 «Выражения и операторы»
- Глава 5 «Инструкции»
- Глава 6 «Объекты»
- Глава 7 «Массивы»
- Глава 8 «Функции»
- Глава 9 «Классы и модули»
- Глава 10 «Шаблоны и регулярные выражения»
- Глава 11 «Подмножества и расширения JavaScript»
- Глава 12 «Серверный JavaScript»