1 Сильные стороны

...привлекательная внешность, мужественная осанка — вот и все мое колдовство.

Уильям Шекспир. Виндзорские насмешницы

В начале своей карьеры программиста я хотел досконально изучить языки, на которых писал, чтобы в полной мере использовать все их возможности. Думаю, это был отличный способ проявить себя, и это сработало, я стал тем, к кому можно обратиться, чтобы узнать, как использовать ту или иную функцию.

Со временем я понял, что использование некоторых функций приносит больше вреда, чем пользы. Одни функции были плохо определены, что, вероятнее всего, объяснялось проблемами переносимости. Другие превращали код в нечитабельный и плохо модифицируемый, что делало его чрезмерно сложным и было чревато ошибками. Кроме того, некоторые функции были плохо спроектированы. Иногда разработчики языка тоже совершают ошибки.

Любой язык программирования имеет свои сильные и слабые стороны. Я понял, что мог бы стать хорошим программистом, используя только достоинства и избегая недостатков. В конце концов, как получить что-то хорошее из плохих деталей?

Иногда комитеты по стандартизации могут убрать из языка неудачные фрагменты, способные привести к сбою программ, в которых эти несовершенные фрагменты используются. Но, как правило, они лишь добавляют новые функции к многочисленным уже существующим и не слишком удачным, причем эти новые функции не всегда гармонично сочетаются с существующими, что порождает новые проблемы.

Тем не менее у *вас* есть возможность определять собственные подмножества и создавать хорошие программы, основываясь исключительно на удачных фрагментах.

Язык JavaScript почти полностью состоит из недостатков. Все началось с того, что основы были заложены в небывало короткий промежуток времени. У раз-

работчиков так и не нашлось времени, чтобы все опробовать и отладить. То же произошло и с Netscape Navigator 2 — все было сделано довольно грубо. Идея создания Java™-приложений провалилась, и язык JavaScript занял место «основного языка для Веб». Хотя популярность JavaScript почти не зависит от его качеств как языка программирования.

К счастью, у JavaScript есть и довольно сильные стороны. Это красивый, элегантный и очень выразительный язык, похороненный под грудой добрых намерений и ошибок. Лучшие черты JavaScript так старательно скрыты, что на протяжении многих лет бытует мнение о том, что JavaScript — это просто невзрачная сломанная игрушка. Моя цель — открыть сильные стороны JavaScript, как выдающегося из динамических языков программирования. Будем считать, что JavaScript — это кусок мрамора, и я, отсекая все лишнее, попытаюсь показать его истинную суть. Я считаю, что мне удалось выделить более элегантное подмножество, чем JavaScript в целом, более надежное, удобное для чтения и сопровождения.

Эта книга не является детальным описанием JavaScript. Основное внимание здесь сосредоточено на сильных сторонах этого языка и советах, помогающих избежать слабых. Упомянутые в этой книге конструкции могут быть использованы для построения надежных и понятных программ любого объема. Концентрируясь только на хороших конструкциях, можно сократить время обучения, повысить надежность и спасти несколько деревьев (которые иначе пришлось бы извести на лишнюю бумагу).

Возможно, наибольшая польза от изучения сильных сторон языка состоит в том, что это не требует отучиваться от использования его слабых сторон. Отучиться применять плохие программные шаблоны очень трудно. Большинство из нас сталкивается с этой непростой задачей с крайней неохотой. Иногда языки подразделяют на подмножества, чтобы облегчить работу студентов. Однако в данном случае я выделил такие подмножества JavaScript, которые способны облегчить работу профессионалам.

Почему JavaScript?

JavaScript важен, потому что это язык веб-браузера. Его связь с браузером делает JavaScript одним из самых популярных языков программирования в мире, хотя в то же время он один из самых презираемых. API браузера, объектная модель документов (Document Object Model, DOM) просто ужасны, но обвинять JavaScript несправедливо. На любом языке нелегко работать с DOM, так как эта модель плохо определена и непоследовательно реализована. Эта книга вскользь касается DOM. Я думаю, что написать книгу о сильных сторонах DOM было бы крайне сложно.

JavaScript презирают потому, что этот язык не похож на какой-либо другой. Если вы разбираетесь в каком-нибудь другом языке, но вам приходится рабо-

тать в программной среде, которая поддерживает только JavaScript, это раздражает. Большинство людей в такой ситуации даже не стараются вначале изучить JavaScript, а потом удивляются, когда оказывается, что JavaScript отличается от других языков, которые они предпочли бы использовать, и что эти различия существенны.

У JavaScript есть удивительная особенность — этот язык позволяет решать поставленные задачи, даже не имея особого представления ни о самом языке, ни о программировании вообще. Он обладает огромной выразительной силой. Тем не менее лучше, если вы знаете, что делаете. Программирование — дело трудное. И без должных знаний браться за него не стоит.

Анализ JavaScript

В основу JavaScript положены как очень хорошие, так и очень плохие идеи.

Очень хорошие идеи касаются функций, нестрогой типизации, динамических объектов, а также нотации литералов объектов. Плохие же идеи связаны с моделью программирования на основе глобальных переменных.

Функции в JavaScript представляют собой различного рода объекты (в основном) в лексическом контексте. JavaScript — один из основных языков, использующих лямбда-выражения. По сути, JavaScript имеет гораздо больше общего с Lisp и Scheme, нежели чем с Java. JavaScript — это Lisp в «шкуре» C, что делает его удивительно мощным языком.

Сегодня в большинстве языков программирования мода требует строгой типизации. Существует теория, что строгая типизация позволяет компилятору обнаруживать больше ошибок во время компиляции. Чем скорее мы сможем найти и исправить ошибки, тем дешевле они нам обойдутся. В JavaScript нет строгой типизации, поэтому компиляторы не в состоянии обнаружить ошибки типизации. Это вызывает беспокойство людей, которые пришли в JavaScript из языков со строгой типизацией. Однако оказывается, что строгая типизация не избавляет от необходимости тщательного тестирования. Я обнаружил, что некоторые из ошибок, которые компилятор определяет как ошибки типизации, вовсе не являются ошибками, и это настораживает. Я вообще считаю, что решать подобные проблемы поможет нестрогая типизация. Совершенно не обязательно выстраивать сложные иерархии классов и ломать голову над системой типов, чтобы достигнуть желаемого результата.

В JavaScript реализована очень мощная нотация литералов объектов. Объекты создаются простым указанием их компонентов. Эта нотация привела к появлению популярного формата обмена данными JSON. (Подробнее о JSON рассказано в приложении Д.)

Прототипическое наследование в JavaScript — спорная концепция. Объекты в JavaScript не привязаны к классам и наследуют свойства непосредственно от

других объектов. На самом деле, это довольно мощный инструмент, но он не знаком программистам с классическим образованием. Если вы попытаетесь применять классические шаблоны разработки непосредственно в JavaScript, вы будете разочарованы. Однако если вы научитесь работать с прототипами в контексте JavaScript, ваши усилия будут вознаграждены.

Язык JavaScript сильно пострадал из-за клеветы, касающийся его основных идей. По большей части, эти идеи хороши, если не необычны. Но кое-что действительно ужасно: JavaScript зависит от глобальных переменных. Все переменные высшего уровня собраны вместе в общем пространстве имен, названном глобальным объектом. Это плохо, так как использование глобальных переменных — не лучшее решение, но в JavaScript они являются основополагающими. К счастью, в JavaScript предусмотрены инструменты, способные решить эту проблему.

Некоторые недостатки невозможно игнорировать. Иногда они неизбежно приводят к ужасным последствиям, о чем рассказывается в приложении А. Однако избежать большинства из них можно, обобщая написанное в приложении Б. Обратитесь к любой другой книге по JavaScript, если захотите побольше узнать о недостатках.

Стандарт, определяющий JavaScript (известный как JScript) — третье издание языка ECMAScript, доступен по адресу http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf. В этой книге описывается не весь язык, обладающий массой недостатков, а лишь подмножество ECMAScript. Приводимые рекомендации не являются исчерпывающими, но помогают в крайних случаях, которых следует избегать.

В приложении В описан программный продукт под названием JSLint. Это анализатор JavaScript, который проверяет код, написанный на JavaScript, и сообщает о его недостатках. JSLint обеспечивает точность, которой не хватает для развития JavaScript, и дает вам уверенность в том, что ваши программы состоят только из хороших частей.

JavaScript — это язык контрастов. В нем много ошибок и сложностей, которые заставляют задаться вопросом: «Почему я должен использовать JavaScript?» Есть два ответа. Во-первых, у вас всегда есть выбор. Сеть стала важной площадкой для разработки приложений, а JavaScript — единственный язык, который поддерживается всеми браузерами. Очень жаль, что идея Java провалилась, она могла бы стать альтернативой для тех, кому нужен строго типизированный классический язык. Но Java потерпел неудачу, а JavaScript процветает, так что еще не известно, так ли плох JavaScript.

Во-вторых, несмотря на все свои недостатки, JavaScript действительно хорош. Это легкий и выразительный язык. И приобретя некоторые навыки, вы поймете, что функциональное программирование — дело довольно веселое.

Однако для того чтобы использовать язык хорошо, нужно быть хорошо информированным об ограничениях, которые порой ставят в тупик. Но пусть это вам не мешает. Сильные стороны с лихвой компенсируют слабые.

Простая проверка

Веб-браузер и любой текстовый редактор — это все, что необходимо для запуска программ на JavaScript. Во-первых, создайте HTML-файл и назовите его, к примеру, program.html:

```
<html><body><pre><script src="program.js">
</script></pre></body></html>
```

Затем создайте в том же каталоге файл с именем program.js:

```
document.writeln('Hello, world!');
```

Теперь, чтобы увидеть результат, откройте ваш HTML-файл в браузере. На протяжении всей книги метод `method` используется для определения новых методов. Он определяется следующим образом:

```
Function.prototype.method = function (name, func) {
    this.prototype[name] = func;
    return this;
};
```

Подробнее этот метод описан в главе 4.

2 Грамматика

... знаком он мне: В грамматике
читал его когда-то.

Уильям Шекспир. Тит Андроник

Описанная в этой главе грамматика сильных сторон JavaScript дает общее представление о структуре языка. Описать грамматику можно с помощью синтаксических диаграмм.

Правила интерпретации таких диаграмм довольно просты:

- Синтаксические диаграммы читаются слева направо.
- Овалами обозначены символьные значения, прямоугольниками — правила или описания.
- Верная последовательность может быть получена из элементов, стоящих друг за другом.
- Любая другая последовательность не верна.
- Синтаксическая диаграмма, имеющая на концах одинарные штрихи, может быть вставлена между двумя другими диаграммами, но с диаграммами, имеющими на концах двойные штрихи, это невозможно.

Грамматика сильных сторон JavaScript, описанная в этой главе, значительно проще грамматики всего языка.

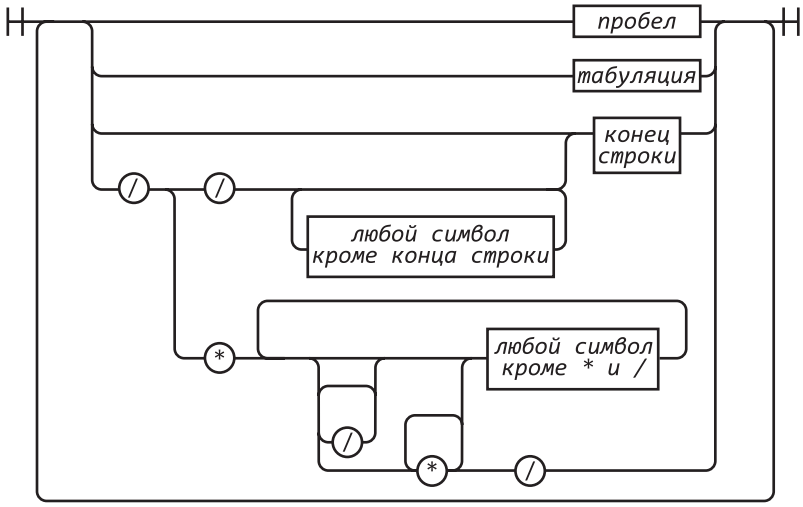
Пробельные символы

Пробелы применяются для разделения символов или комментариев. Пробельные символы, как правило, не важны, но иногда их приходится использовать для разделения последовательностей символов, которые могли бы быть объединены. Например:

```
var that = this;
```

Пробел между `var` и `that` удалять не следует, хотя другие пробелы могут быть удалены.

пробел



В JavaScript существует два способа задания комментариев: блочные комментарии начинаются с `/*` и заканчиваются `*/`, строчный комментарий начинается с `//`. Чтобы облегчить чтение кода, принято использовать комментарии, точно описывающие программный код. Нет ничего хуже, чем устаревшие комментарии.

Блочное задание комментариев символами `/* */` пришло из языка PL/1. В PL/1 такое сочетание было выбрано потому, что оно вряд ли могло бы встретиться в программе, разве что внутри строки. В JavaScript эти сочетания могут возникнуть и в регулярных выражениях, поэтому блочные комментарии не совсем безопасны. Например:

```
/*
    var rm_a = /a*/.match(s);
*/
```

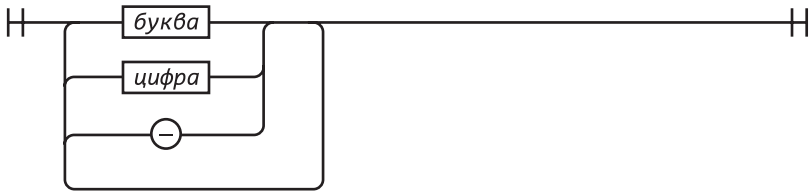
Приведенный код содержит ошибку, поэтому рекомендуется избегать символов `/* */` в комментариях, а использовать для комментариев символы `//`. В этой книге применяется исключительно второй способ.

Имена

Имя может состоять из одной и более букв, цифр или символов подчеркивания. В качестве имени нельзя использовать одно из следующих зарезервированных слов:

```
abstract
boolean break byte
case catch char class const continue
debugger default delete do double
else enum export extends
false final finally float for function
goto
if implements import in instanceof int interface
long
native new null
package private protected public
return
short static super switch synchronized
this throw throws transient true try typeof
var volatile void
while with
```

название

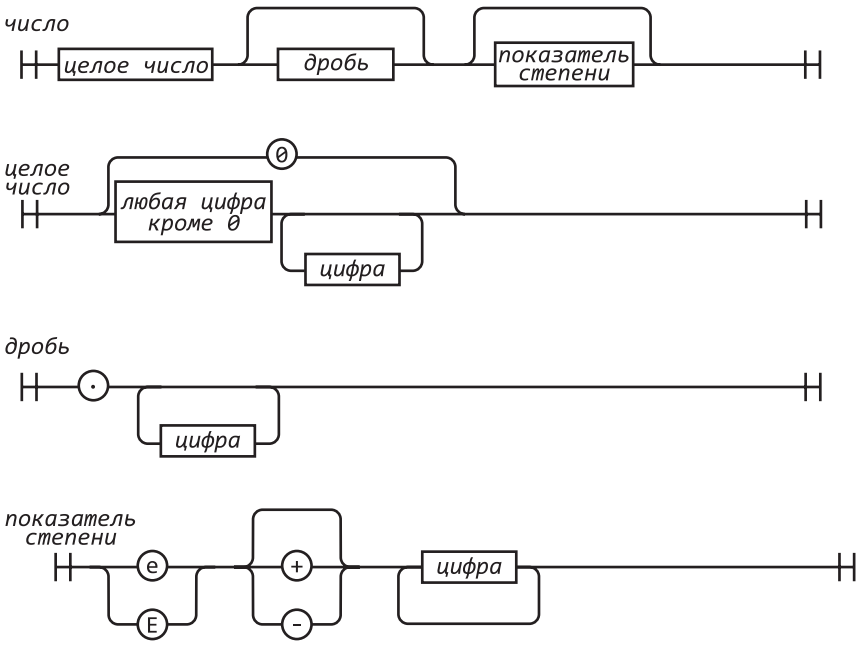


Большинство из этих зарезервированных слов в JavaScript не используются. При этом в приведенный список не включены несколько слов, которые следует зарезервировать, например `undefined`, `NaN` и `Infinity`. Не допускается использование зарезервированных слов для именования переменных или параметров. Более того, их нельзя применять в качестве имен параметров объекта, причем ни при символьном задании объекта, ни после точки.

Имена служат для объявления переменных, параметров, свойств, операторов и меток.

Числа

В JavaScript всего один числовой тип. Как и `double` в Java, он представляет собой 64-разрядное число с плавающей точкой. В отличие от большинства других языков программирования, в JavaScript нет отдельного целого типа, поэтому 1 и 1.0 — это одно и то же значение. Это довольно удобно, так как полностью исключает проблему переполнения для небольших целых чисел. Все, что вам нужно знать о числе — это то, что оно действительно число. Таким образом, удается избежать множества ошибок, связанных с числовыми типами.



Если числовое значение содержит показатель степени, то оно буквально вычисляется путем умножения на 10 части, стоящей перед `e`, столько раз, сколько указано после `e`. Так, 100 и 1e2 — это одно и то же.

Для задания отрицательных чисел используется префикс `-`.

Для обозначения результатов вычислений, в ходе которых не может быть получен нормальный результат, используется значение `NaN`. `NaN` не имеет конкретного численного значения. Функция `isNaN(number)` проверяет, имеет ли число значение `NaN`.

Бесконечностью считаются все значения больше 1.79769313486231570e +308.

К числам применяются методы, которые подробно описаны в главе 8. В JavaScript существует объект `Math`, содержащий набор методов для работы с числами. Например, метод `Math.floor(number)` преобразует значения чисел в целые.

Строки

Строка может быть заключена в одинарные или двойные кавычки. Она может содержать ноль и более символов. Обратный слэш (\) является экранирующим символом. В то время когда разрабатывался язык JavaScript, стандарт Unicode состоял из набора 16-разрядных символов, так что все символы в JavaScript являются 16-разрядными.

JavaScript не имеет символьного типа. Для того чтобы получить символ, нужно задать строку из одного символа.