ГЛАВА

4

Основные языковые компоненты

В ЭТОЙ ГЛАВЕ...

- Версии JavaScript
- Лексемы
- Переменные
- Константы
- Типы данных
- Выражения
- Комментарии
- Функции
- Объекты JavaScript
- События

В этой главе рассматриваются фундаментальные языковые компоненты языка JavaScript в том виде, в каком они предстают перед нами сегодня. Глава начинается с конспективного обзора версий JavaScript, после чего мы перейдем к изучению компонентов языка, начиная с дескрипторов и заканчивая анализом механизма обработки событий.

Версии JavaScript

Хотя сама версия JavaScript не является фундаментальным языковым компонентом, она, тем не менее, очень важна и может рассматриваться как фундаментальный аспект, особенно, если ваш JavaScript-код должен выполняться в устаревших браузерах.

По мере того, как компании Netscape, Microsoft и ряд других совершенствовали основной стандарт ECMAScript и свои отдельные реализации языка, технология JavaScript обзаводилась новыми версиями языка, равно как и получала поддержку в новых версиях браузеров. В данной книге рассматриваются все версии JavaScript вплоть до 1.5. Полный список поддерживаемых платформ приведен в табл. 4.1.

Следующей версией языка JavaScript будет 2.0. В версии 2.0 язык будет переделан, однако при этом должна сохраняться обратная совместимость. Ожидается, что в версии 2.0 появятся такие понятия, как классы, типы и другие свойства, которые в большей степени присущи статическим языкам.

Таблица 4.1. Список платформ, которые поддерживают JavaScript

Версия	*Описание*
JavaScript 1.0	Поддерживается в браузере Netscape Navigator 2.
JavaScript 1.1	Поддерживается в браузерах Netscape Navigator 2, Internet Explorer 3, Opera 3 и Netscape Enterprise Server 2.
JavaScript 1.2	Поддерживается в браузерах Netscape Navigator версий 4 – 4.05 и Internet Explorer 4.
JavaScript 1.3	Поддерживается в браузерах Netscape Navigator версий 4.06 – 4.7 и Internet Explorer 5.
JavaScript 1.4	Поддерживается в некоторых версиях браузеров Mozilla, предшествующих его альфа-версиям (до M12), HotJava 3.0, Netscape Enterprise Server 4.0.
JavaScript 1.5	Поддерживается в браузерах Netscape Navigator версии 6 (Mozilla) и более новых версиях.

Чтобы понять, как соотносятся друг с другом версии JavaScript, JScript и ECMAScript, просмотрите табл. 4.2, 4.3 и 4.4. Отношения между JavaScript, JScript и ECMAScript, которые выходят за рамки информации, представленной в таблицах, не обязательно такие же, поскольку дополнительно существуют небольшие различия.

Таблица 4.2. Отношения между JavaScript и JScript

JavaScript	*JScript*
1.0	1.0
1.1	1.0 – 2.0
1.2	3.0 – 4.0
1.3	5.0 – 5.1
1.4	5.0 – 5.1
1.5	5.5

Таблица 4.3. Отношения между JavaScript и ECMAScript

JavaScript	*ECMAScript*
1.0	Нет соответствия.
1.1	Нет соответствия – основание для внедрения стандарта.
1.2	Нет соответствия – содержит большую часть ECMAScript 1.0.
1.3 – 1.4	1.0.
1.5	Редакция 1.0 Edition 3.

Таблица 4.4. Отношения между JScript и ECMAScript

JScript	*ECMAScript*
1.0	Нет соответствия.
2.0	Нет соответствия – основание для внедрения стандарта.
3.0 – 3.1	1.0.
4.0 – 5.0	1.0.
5.5	Редакция 1.0 Edition 3.

В недавнем прошлом, вы как разработчик на JavaScript должны были хорошо изучить платформу, для которой намеревались создавать конечный продукт, дабы правильно выбрать версию JavaScript. После того, как выбор был сделан, вы должны были описать используемую версию с помощью атрибута `language` дескриптора `<script>`. Но, к сожалению, в HTML 4.01 и XHTML 1.0 этот дескриптор был официально объявлен нерекомендованным к использованию. Браузер Microsoft Internet Explorer спроектирован таким образом, что он игнорирует версии JavaScript выше 1.3. В браузере Netscape Navigator реализован другой подход, который обеспечивает совместимость со стандартом HTML. Теперь в нем используется конструкция `<script type="text/javascript; version=1.5">`. В результате стандартизованный дескриптор, позволяющий указать версию языка, больше не используется.

Теперь, когда вы узнали, какие сложности выбора соответствующей версии браузера сопровождают JavaScript, рассмотрим другой ключевой компонент JavaScript – лексемы.

Лексемы

Лексемы (**token**) – это отдельные короткие слова, фразы или символы, которые умеет распознавать JavaScript. Во время интерпретации JavaScript-кода браузер выполняет синтаксический разбор, разбивая при этом сценарий на лексемы и игнорируя комментарии, а также все пробельные символы.

Лексемы JavaScript подразделяются на четыре категории: идентификаторы, ключевые слова, литералы и операции. Как и в случае любых других языков программирования, в вашем распоряжении имеется множество способов упорядочения этих лексем, которые заставляют компьютер выполнить конкретную задачу. **Синтаксис** (**syntax**) языка есть набор правил и ограничений в отношении способов, посредством которых можно строить различные комбинации лексем.

Идентификаторы

Идентификаторы (**identifiers**) – это просто имена, которые обозначают переменные, методы и объекты. Идентификаторы представляют собой различные комбинации символов или символов и цифр. Некоторые имена являются встроенными в язык JavaScript и, следовательно, зарезервированными.

Помимо этих ключевых слов вы можете определять свои собственные оригинальные идентификаторы, наделенные конкретным смыслом. Разумеется, при этом существует набор правил, обязательных для исполнения:

- Все идентификаторы должны начинаться либо с буквы, либо с символа подчеркивания (_).
- После обязательной буквы или символа подчеркивания в идентификаторах можно использовать любые буквы, цифры, а также символы подчеркивания.
- Буквами считаются все символы верхнего регистра от A до Z и все символы нижнего регистра от a до z (стандарт ECMAScript допускает использование также и символов Unicode, однако в этом случае необходимо соблюдать осторожность, поскольку браузеры ранних версий могут не поддерживать Unicode).
- Последовательность символов, образующих идентификатор, не должна содержать пробелов.
- Цифрами считаются символы от 0 до 9.

В табл. 4.5 приводятся примеры допустимых и недопустимых идентификаторов.

Таблица 4.5. Примеры пользовательских идентификаторов JavaScript

Допустимый	*Недопустимый*
`current_WebSite`	`current WebSite`
`numberOfHits`	`#ofIslands`
`n`	`2bOrNotToBe`
`N`	`return`

Обратите внимание та тот факт, что идентификатор `current WebSite` является недопустимым по той причине, что он содержит пробел. JavaScript пытается интерпретировать его как два идентификатора. Если в идентификаторе по ряду причин должен присутствовать пробел, то в этом случае вместо него употребляется символ подчеркивания.

Идентификатор `#ofIslands` недопустим, поскольку символ фунта не входит в список символов, которые разрешено использовать для построения идентификаторов. Идентификатор `2bOrNotToBe` не является допустимым, так как он начинается с цифры. Идентификатор `return` уже используется в JavaScript для других целей. Попытка его применения в качестве пользовательского идентификатора приводит к ошибке во время запуска сценария на выполнение.

Следует отметить, что оба идентификатора `n` и `N` являются допустимыми, однако разными. JavaScript является языком, чувствительным к регистру, и трактует идентификаторы, представленные символами разных регистров, как различные, даже если они произносятся совершенно одинаково.

Ключевые слова и зарезервированные слова

Ключевые слова (**keywords**) – это предопределенные идентификаторы, которые образуют ядро языка программирования. В JavaScript ключевые слова выполняют уникальные функции, такие как, например, объявление новых переменных и функций, принятие решения на основе текущего состояния компьютера или организации цикла внутри приложения.

Ключевые слова встроены в язык JavaScript и всегда доступны для использования программистом, но при этом необходимо следовать корректному синтаксису. Ключевое слово `var` является первым из ключевых слов, которое подробно описывается далее в этой главе. По мере вашего продвижения по материалам этой книги вы узнаете, как пользоваться другими ключевыми словами для разработки более динамичных программ.

Зарезервированные слова (**reserved words**) – это идентификаторы, которые не следует использовать в JavaScript в качестве имен переменных, функций, объектов и методов. К этой категории относятся ключевые слова, которые зарезервированы для возможного использования в будущем. Ниже приведен перечень всех зарезервированных слов JavaScript:

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

Литералы

Литералы представляют собой числа или строки, которые используются для представления значений в JavaScript. Эти значения не меняются на протяжении времени выполнения сценария. В следующих пяти разделах описаны различные типы литералов.

Целочисленные

Целочисленные величины могут быть выражены в десятичном (основание системы счисления 10), восьмеричном (основание системы счисления 8) или шестнадцатеричном (основание системы счисления 16) формате. Целочисленный литерал в десятичном формате может включать любую последовательность цифр, которые не начинаются с 0 (нуля). Ноль вначале целочисленного литерала означает восьмеричный формат представления.

Само по себе целочисленное значение может включать последовательность цифр от 0 до 9. Для обозначения шестнадцатеричных значений перед целочисленным значением ставятся `0x` (или `0X`). Шестнадцатеричные значения могут содержать цифры от 0 до 9 и буквы от A до F. Рассмотрим несколько примеров:

Десятичный формат	`33, 2139, -33`
Восьмеричный формат	`071, 03664, 37777777737`
Шестнадцатеричный формат	`0x7b8, 0X395, 0xffffffdf`

С плавающей точкой

Литералы с плавающей точкой определяют десятичные числа с дробной частью. Они могут быть представлены в стандартной форме или в экспоненциальном представлении. В экспоненциальном представлении для обозначения порядка используется символ `e` или `E`. И десятичная мантисса, и порядок могут быть со знаком или без знака (положительным или отрицательным), как показано в следующих примерах:

```
3405.673
-1.958
8.3200e+11
8.3200e11
9.98E-12
```

Булевские

В JavaScript реализованы булевские типы данных и в силу этого обстоятельства поддерживаются два литерала – `true` и `false`. Даже если вы новичок в программировании, все равно вы очень скоро поймете, насколько часто возникает потребность в значениях `true` и `false`. Именно потому-то они и встроены в язык. Ключевые слова `true` и `false` должны быть представлены символами нижнего регистра. В результате эти же слова, но представленные символами верхнего регистра, то есть `TRUE` и `FALSE`, можно использовать в качестве своих оригинальных идентификаторов, тем не менее, во избежание путаницы поступать подобным образом не рекомендуется.

Строки

Строковый литерал представляет собой ноль или большее число символов Unicode, заключенных в двойные (" ") или одинарные кавычки (' '). JavaScript предоставляет вам этот выбор, однако вы должны заключать отдельную строку в кавычки одного и того же вида. Далее представлены примеры строковых литералов, заключенных в кавычки.

```
"Allen's car"
'виртуальные "сообщества"'
"#12-6"
"Посмотри, что творится с облаками!"
```

Возможность использования разных видов кавычек удобна в тех случаях, если вы отдаете предпочтение тому или иному их типу. Когда вы приступите к изучению встроенных методов JavaScript, неукоснительно следуйте рекомендациям по использованию строковых литералов при передаче их в качестве параметров. В ряде случаев для правильного вызова метода, возможно, придется употреблять оба типа кавычек, в частности, когда необходимо поместить один литерал внутрь другого. Это отличается от использования управляющих кодов, которые описаны в следующем разделе.

Специальные символы

При написании сценариев иногда может возникнуть потребность дать компьютеру команду на использование специальных символов или клавиш, таких как, например, табуляция или переход на новую строку. Чтобы сделать это, поместите обратную косую черту перед одним из управляющих кодов, как показано в табл. 4.6.

Таблица 4.6. Управляющие символы

Управляющая последовательность	*Unicode-значение*	*Описание*
\b	\u0008	Возврат на одну позицию (забой)
\f	\u000C	Подача страницы
\n	\u000A	Перевод строки (новая строка)
\r	\u000D	Возврат каретки
\t	\u0009	Горизонтальная табуляция
\v	\u000B	Вертикальная табуляция
\u	\uXXXX	Управляющие Unicode-последовательности
\\	\u005C	Обратная косая черта
\'	\u0027	Одинарная кавычка
\"	\u0022	Двойная кавычка

Если требуется выполнить эмуляцию клавиши табуляции с целью выравнивания двух столбцов данных, вы должны воспользоваться символом табуляции (\t). В листинге 4.1 показано, как выровнять текст с помощью символов табуляции. Удобочитае-

мость самого сценария несколько теряется после добавления специальных символов, однако, как видно на рис. 4.1, конечные результаты выглядят намного лучше.

Листинг 4.1. Использование специальных символов в JavaScript

```
<html>
<head>
  <title>JavaScript. Полное руководство</title>
</head>
<body>
<!--
    Замечание. Специальные символы не дадут эффекта, если они не
    будут заключены в форматированный блок
-->
  <pre>
    <script type="text/javascript">
    <!--
      document.writeln("\tПерсонал");
      document.writeln("Name\t\tАдрес");
      document.writeln("Иванов\t\tivanov@company.com");
      document.writeln("Петров\t\tpetrov@company.com");
      document.writeln("Сидоров\t\tsidorov@company.com");
    // -->
    </script>
  </pre>
</body>
</html>
```

НА ЗАМЕТКУ

Специальные символы обеспечивают должный эффект, только когда используются в отформатированном текстовом блоке. Следовательно, ваш сценарий должен быть заключен в дескрипторы `<pre>` и `</pre>`.

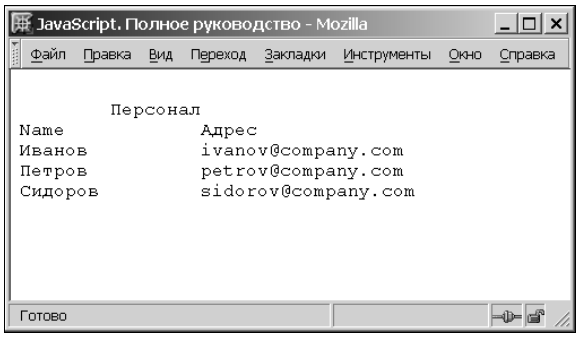

Рис. 4.1. *Выравнивание текста с помощью символов табуляции в JavaScript*

Если у вас возникла необходимость использовать кавычки внутри строкового литерала, им должен предшествовать символ обратной косой черты, например:

```
document.write("\"Фантазия важнее знания.\"");
document.write(", Альберт Эйнштейн");
```

Приведенный выше сценарий отобразит на экране следующую текстовую строку:

```
"Фантазия важнее знания.", Альберт Эйнштейн
```

Операции

Операции (**operators**) – это символы или идентификаторы, которые представляют способ вычисления или манипулирования некоторой комбинацией выражений. Наиболее известной операцией, которую вы использовали до сих пор, является операция присваивания. В простом примере `x = 10`, как `10`, так и переменная `x` представляют собой выражения. Когда интерпретатор JavaScript встречает операцию присваивания, которая находится между двумя выражениями, он действует в соответствии с правилами операции присваивания. В рассматриваемом случае значение выражения в правой части присваивается переменной, указанной в левой части. Наряду с обычными арифметическими операциями в JavaScript поддерживаются свыше 30 других операций, которые подробно рассматриваются в главе 5.

Переменные

Переменная (**variable**) – это имя, присвоенное ячейке памяти компьютера, в которой хранятся данные. Первые программисты тратили уйму времени на перевод данных наподобие сообщения "Здравствуй, мир!" в двоичный формат. Затем им приходилось отыскивать свободное пространство в памяти компьютера, дабы поместить туда полученную совокупность нулей и единиц, к тому же запоминать, где конкретно начинались, а где заканчивались соответствующие данные. Зная расположение (адрес), программисты могли находить, обновлять и извлекать данные, когда по ходу программы в этом возникала необходимость. В результате приходилось отслеживать весьма немалый объем данных.

Переменные существенно облегчили для современных программистов процесс хранения, обновления и извлечения информации. Пользуясь переменными, вы можете назначать осмысленные имена ячейкам, в которых хранятся данные, а все остальное берет на себя компьютер.

Именование

Имя JavaScript-переменной образуется из одной или нескольких букв, цифр и символов подчеркивания. Оно не может начинаться с цифры (от 0 до 9). К буквам относятся не только буквы английского алфавита от A до Z и от a до z, но также и другие символы, которые Unicode классифицирует как буквы других алфавитов. Например, в немецком алфавите имеются буквы ä и ü, а во французском – буква é. Язык JavaScript чувствителен регистру и в силу этого рассматривает следующие два имени переменных как разные:

```
internetAddress
internetaddress
```

Приведенные ниже имена также являются допустимыми:

```
_lastName
n
number_2
```

Во время присвоения имен переменным в JavaScript в виде одного слова обычной практикой является использование букв нижнего регистра. Когда для представления имени переменной выбираются два и больше слов, как правило, то для первого слова применяются буквы нижнего регистра, а для первых букв всех последующих слов используются заглавные буквы. Хорошим тоном является применение для имен переменных двух и более слов. В этом случае как самому программисту, так и другим будет понятно, для чего была создана та или иная переменная.

Например, предположим, что вы хотите, чтобы переменная хранила булевские значения (`true` или `false`), на основе которого можно было бы узнать, завершил ли посетитель Web-страницы ввод своего имени в соответствующем текстовом поле. Если в качестве имени переменной выбрано слово `finish` (завершение), то другой программист (или вы сами несколько месяцев спустя) взглянет на него, и его начнут одолевать сомнения: "Может, это флаг, который нужно проверить, чтобы знать, что посетитель завершил ввод? Или, может быть, это строка, определяющая, *что* необходимо вывести после того, как посетитель завершит ввод, например, благодарственное сообщение?"

Для упомянутого случая гораздо лучше подойдет имя `isDone` (завершено ли?). Слово "is" в префиксе позволяет указать, что эта переменная ставит вопрос типа "да-нет", из чего следует, что переменная хранит булевское значение. Если посетитель завершил ввод своего имени, переменной `isDone` присваивается значение `true`; в противном случае она получит значение `false`. Несмотря на то что длина имени JavaScript-переменной ограничивается только объемом памяти компьютера, желательно, чтобы эта длина была практически оправданной. Рекомендуется использовать от 1 до 20 символов, или от одного до трех слов. При написании сценариев старайтесь не заходить за край строки. В случае чересчур длинных имен переменных подобное случается довольно-таки часто, при этом искажается внешний вид и структура программного кода.

Некоторые традиционно используемые однословные и даже односимвольные переменные представляют внутрипрограммные или математические значения. Чаще других используются такие имена переменных: `n` для представления чисел; `x`, `y` и `z` для представления координат и `i` для представления заполнителя в рекурсивной функции или счетчика в цикле. Подчеркнем еще раз, что эти методы использования переменных просто являются традиционными; их можно применять для любых нужных вам целей.

Очень часто возникают ситуации, особенно если вы трудитесь на ниве профессионального программирования, когда на некоторой стадии с вашим кодом должны ознакомиться другие люди. Подбор непротиворечивых имен со смысловой нагрузкой может оказать неоценимую помощь тем, кто будет в дальнейшем сопровождать ваши сценарии. Плохо продуманные соглашения по именованию переменных могут причинить сильную головную боль компании-разработчику и существенно снизить ценность практических результатов.

Объявление

Объявление переменной означает для JavaScript, что вы намереваетесь использовать конкретный идентификатор в качестве переменной. Чтобы объявить переменную в JavaScript, воспользуйтесь ключевым словом `var`, за которым укажите имя новой переменной. Такое действие зарезервирует это имя как переменную, которая будет использоваться в качестве области хранения необходимых данных. Применение ключевого слова `var` вовсе не обязательно, однако его присутствие является признаком хорошего стиля программирования. В приведенных ниже примерах обратите внимание на то, что за один раз можно объявить сразу несколько переменных, при этом для отделения переменных друг от друга используется запятая.

```
var internetAddress;
var n;
var i, j, k;
var isMouseOverLink, helloMessage;
```

После того, как переменная будет объявлена, она готова к заполнению ее первым значением. Подобного рода **инициализация** (**initializing**) выполняется с помощью операции присваивания (=).

НА ЗАМЕТКУ

Знак равенства (=) используется для присвоения значения переменной. Более подробную информацию об операции присваивания можно найти в главе 5.

Вы можете инициализировать переменную в своем сценарии как в момент ее объявления, так и в любой другой момент позже. Присвоение значения переменной в момент ее объявления может помочь вам запомнить, какой тип значений вы первоначально намеревались хранить в этой переменной. Ниже приведены предыдущие примеры, переписанные с учетом инициализации.

```
var internetAddress = "name@company.com";
var n = 0.00;
var i = 0, j = 0, k;
var isMouseOverLink = false;
var helloMessage = "Благодарим за посещение!";
k = 0;
```

Обратите внимание на то, что все переменные были объявлены и инициализированы в один и тот же момент, за исключением переменной `k`, которая бала инициализирована несколько позднее. JavaScript производит чтение кода сверху вниз, переходя от строки к строке кода и выполняя инструкции по порядку. До тех пор, пока переменная не достигнет стадии инициализации, говорят, что она **не определена** (**undefined**). JavaScript позволяет проверить, было ли переменной присвоено значение с помощью операции `typeof`. Эти и другие вопросы рассматриваются в главе 5. JavaScript предлагает еще один способ объявления переменной – путем ее инициализации без использования ключевого слова `var`. Если вы присваиваете значение новой переменной до ее объявления с помощью `var`, JavaScript автоматически объявит ее.

НА ЗАМЕТКУ

Объявление переменной без использования ключевого слова `var` автоматически объявляет эту переменную как **глобальную** (**global**) в смысле области действия. И хотя это можно рассматривать как некий ускоренный способ объявления, хороший стиль программирования требует явного объявления всех переменных по отдельности. Использование ключевого слова `var` поддерживает концепцию области действия переменных.

Типы

При сохранении очередной порции данных (часто такую порцию данных называют **значением** (**value**)) JavaScript автоматически классифицирует ее как подпадающую под один из пяти существующих типов данных. В табл. 4.7 перечислены различные типы данных, которые поддерживаются в JavaScript.

Таблица 4.7. Типы данных JavaScript

Тип	*Примеры*
`number`	`-19, 3.14159`
`boolean`	`true, false`
`string`	`"Элементарно, Ватсон!", ""`
`function`	`unescape, write`
`object`	`window, document, null`
`undefined`	`undefined`

Переменная типа `number` хранит либо целое, либо вещественное число. Переменная типа `boolean` содержит либо значение `true`, либо значение `false`. Переменная типа `string` может содержать любое строковое значение (литерал или выражение, вычисление которого дает строку), которое было ей присвоено, включая пустую строку. В табл. 4.7 было показано, как можно представить пустую строку с помощью двух двойных кавычек. Функции (тип `function`) либо определены пользователем, либо являются встроенными. Например, `unescape` – это встроенная функция JavaScript. В главе 7 вы узнаете, как создавать свои собственные функции.

Функции, которые принадлежат объектам, в JavaScript называются **методами** (**methods**) и относятся к типу данных `function`. Ключевые объекты клиентской стороны JavaScript, такие как `window` или `document`, естественно, принадлежат к типу данных `object`. Переменные типа `object` или просто **объекты** (**objects**), могут хранить в себе различные объекты. Говорят, что переменная, которая хранит значение `null`, имеет тип `object`. Это объясняется тем, что JavaScript рассматривает значение `null` как объект.

В других языках программирования предполагается, что на протяжении программы все значения, присваиваемые конкретной переменной, должны иметь тип данных, указанный при объявлении этой переменной. Более того, при попытке присвоить переменной значение другого типа возникает состояние ошибки. Подобное не случается в JavaScript, который относится к языкам со слабым контролем типов. От вас не тре-

буется определять типы данных, равно как вам не запрещается присваивать данные различных типов одной и той же переменной. JavaScript-переменная в любой момент времени может принять данные нового типа, что, в свою очередь, меняет тип этой переменной. Следующие примеры показывают, как корректно использовать переменные в JavaScript.

```
var carLength;
carLength = 4 + 5;
document.writeln(carLength);
carLength = "Длиннее не бывает";
document.writeln(carLength);
```

Сразу после объявления переменная `carLength` получает значение `4 + 5`. JavaScript сохраняет число 9 как значение типа `number`. Однако если затем присвоить переменной `carLength` значение `"Длиннее не бывает"`, JavaScript позволит хранить в переменной `carLength` значения другого тип а – `string`. Такой подход позволяет обойтись без дополнительных действий, которые обычно нужно выполнить в других языках программирования, дабы уведомить компьютер об изменении типа данных.

Область действия переменных

Область действия переменной (**scope**) означает область или области программы, в рамках которых можно ссылаться на эту переменную. Предположим, что вы внедрили один сценарий в заголовок HTML-документа, а другой сценарий (с использованием другого набора дескрипторов `<script>`) – в тело того же HTML-документа. JavaScript рассматривает любые переменные, объявленные в этих двух областях как находящиеся в одной и той же области действия. Эти переменные рассматриваются как **глобальные** (**global**), при этом к ним возможен доступ со стороны любого сценария в текущем документе. Далее в этой главе будет дано более подробное описание функций, которые представляют собой отдельные блоки кода. Переменные, объявленные в этих блоках, рассматриваются как локальные, и доступ к ним со стороны того или иного сценария не всегда возможен.

Локальные переменные

Переменная, объявленная внутри функции, является **локальной** (**local**) в контексте некоторой области действия. Только эта функция имеет доступ к значению, которое хранится в переменной. Каждый раз, когда данная функция вызывается, локальная переменная создается заново. Точно так же, каждый раз, когда функция завершается, локальная переменная уничтожается., объявляющая Переменная с таким же именем, но объявленная в другой функции, трактуется JavaScript как отличная от рассматриваемой переменной. Каждая из таких переменных ссылается на собственный блок памяти. Исключением из этого правила является ситуация, когда функция рассматривается как объект. В этом случае переменная типа функции может продолжать существовать. Более подробную информацию по этому поводу можно найти в главе 7.

Глобальные переменные

Если требуется, чтобы переменная совместно использовалась в более чем одной функции, необходимо объявить эту переменную за пределами всех функций (но, есте-

ственно, внутри дескрипторов <script>). В результате появляется возможность совместного использования такой переменной в любой части приложения, в том числе и внутри всех функций.

НА ЗАМЕТКУ

Рекомендуется объявлять глобальные переменные в заголовке (разделе <head>) HTML-страницы, что обеспечит их загрузку до загрузки любых других частей приложения.

В листинге 4.2 продемонстрировано, как объявлять и пользоваться глобальными и локальными переменными. Чтобы подчеркнуть различие между областями действия этих переменных, в программу была специально включена функция. Вам не обязательно понимать, как работают функции, достаточно знать, что они подобны отдельным частям сценария, заключенным в пару фигурных скобок ({}). Если вы пока не знакомы с механизмом функций, то более подробную информацию о них можете получить в главе 7.

Листинг 4.2. Сравнение областей действия глобальных и локальных переменных

```
<html>
<head>
<title>Спецификации на автомобили</title>

<script type="text/javascript" language="JavaScript">
//Инициализация глобальных переменных
var color = "зеленый";
var numDoors= 4;

//Объявление функции вывода спецификации на автомобиль
function carSpecs()
{
  //Объявление и установка переменных внутри функции
  color = "красный";
  price = "$25 000";
  var numDoors = 2;
  document.write(numDoors,"-дверный ",color);
  document.write(" автомобиль стоит ",price);
}

</script>
</head>

<body>
<h2><u>Спецификации на автомобили</u></h2>
<script type="text/javascript" language="JavaScript">

//Вывод результатов выполнения функции carSpecs()
carSpecs();

//Вывод значений переменных за пределами функции
document.write("<br>",numDoors,"-дверный ",color);
```

```
document.write(" автомобиль стоит ",price);

</script>
</body>
</html>
```

В этом примере глобальные переменные `color` (цвет) и `numDoors` (число дверей) объявлены в заголовке HTML-файла вместе с функцией `carSpecs` (спецификация автомобиля). Программа начинается с вызова функции `carSpec`, которая отображает содержимое переменных. Одна из локальных переменных, `numDoors`, объявлена внутри этой функции. Обратите внимание, что эта переменная имеет имя, совпадающее с одной из глобальных переменных, которая была объявлена заголовке HTML-файла, при этом значение локальной переменной равно 2, а не 4. Кроме того, в этой же функции объявлена еще и новая глобальная переменная с именем `price`. В результате выполнения оператора присваивания глобальная переменная `price` получила значение `"$25 000"`.

В начале тела функции значение глобальной переменной `color` изменяется с `"зеленый"` на `"красный"`, а новой глобальной переменной `price` присваивается строковое значение `"$25 000"`. Когда значения этих переменных отображаются внутри функции, используется локальная переменная `numDoors`, следовательно, выводится спецификация двухдверного автомобиля.

По завершении вызова функции значения переменных отображаются снова, однако, на этот раз уже за пределами функции. Поскольку это делается вне функции `carSpec`, используется глобальная переменная `numDoors`, поэтому выводится спецификация четырехдверного автомобиля. Несмотря на то что переменная `price` объявлена внутри функции, она объявлена как глобальная, так что она видна за и пределами этой функции. На рис. 4.2 показаны результаты выполнения кода в листинге 4.2.

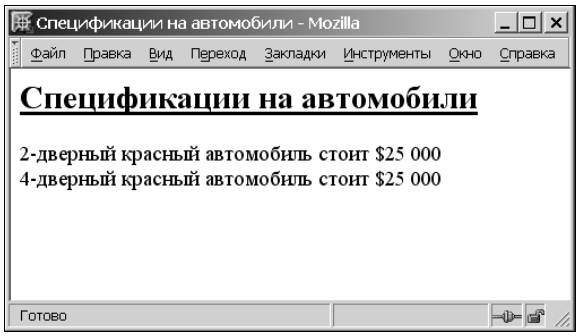

Рис. 4.2. *Различия между областями действия локальных и глобальных переменных*

Константы

Константа – это, по сути, переменная, которая сохраняет одно и то же значение на протяжении всей программы. В JavaScript предусмотрен набор встроенных констант для представления значений, используемых в общих математических вычислениях, например, `pi`. Доступ к этим константам осуществляется через объект `math`. Более подробное описание встроенных констант и объектов можно найти в главе 8 этой книги.

Константы, определенные пользователем, представляют собой переменные, объявленные программистом, причем значения этих переменных меняться не могут. Константы обычно представляются словами, записанными с помощью букв верхнего регистра, и определяются в начале программы.

Поддержка констант появилась в версии JavaScript 1.5. Константы объявляются с использованием ключевого слова `const`. Например, следующая строка кода:

```
const car = "Жигули";
```

объявляет константу с именем `car`. После того, как константа `car` будет объявлена, она может использоваться в разных частях программного кода, однако ее значение изменять нельзя.

НА ЗАМЕТКУ

Ключевое слово `const` представляет собой JavaScript-расширение стандарта ECMAScript редакции 3, и оно не поддерживается в JScript (реализации ECMAScript компании Microsoft) и реализации JavaScript в браузере Opera.

Типы данных

И хотя мы уже рассматривали базовые типы данных, которые могут быть присвоены переменным, тем не менее, стоит еще раз напомнить, что функции и объекты относятся к специальным типам данных. Они предлагают интересные способы хранения и манипулирования данными в рамках вашего сценария. Особенности использования функций и объектов подробно описываются в главе 7.

Выражения

Выражение можно построить, применяя операции к операндам, простейшей формой которых являются литеральные значения и переменные. Результат вычисления выражения относится к одному из следующих типов данных JavaScript: `boolean`, `number`, `string`, `function`, `object` и `undefined`.

В простейшем случае выражение – это одно число или переменная, в то же время оно может включать множество переменных, ключевых слов и операций. Например, выражение `x = 10` присваивает значение `10` переменной `x`. Результатом вычисления этого выражения как единого целого будет `10`, следовательно, использование такого выражения в строке кода наподобие `document.writeln(x = 10)` является вполне допустимым.

JavaScript в общем случае просто видит строку в круглых скобках и отображает ее, но в рассматриваемом случае должна быть выполнена некоторая предварительная обработка. Сначала JavaScript вычисляет то, что заключено в скобки, а затем отображает результирующее значение. В данном случае на экране отображается число 10.

После того, как работа, связанная с присваиванием значения `10` переменной `x`, будет завершена, допустимым станет и следующее выражение: `x`. В этом случае единственным действием, предпринимаемым JavaScript, будет считывание соответствующего значения из памяти компьютера, при этом никакого присваивания не выполняется. Помимо операции присваивания, существует множество других операций, которыми можно пользоваться при построении выражений (см. главу 5).

Комментарии

До сих пор HTML-дескрипторы комментариев применялись для того, чтобы браузеры ранних версий не отображали внедренных сценариев, которые они не способны выполнить. А как поступать в тех случаях, когда в JavaScript-код необходимо включить комментарии, которые JavaScript будет игнорировать? Здесь возможны два варианта, при этом используется синтаксис, характерный для языков программирования C, C++ и Java. Вот как выглядит один из вариантов синтаксиса комментариев:

```
// Однострочный комментарий
```

Две косых черты (`//`) в начале строки скрывают текст, который следует за ними до конца текущей строки. Для более крупных блоков комментариев можно воспользоваться синтаксисом многострочных комментариев:

```
/* Если комментарий должен состоять из нескольких строк,
   то он должен быть представлен
   в таком вот виде. */
```

В обеих формах комментариев пробельные символы игнорируются. В силу этого обстоятельства, приведенные ниже строки кода являются допустимыми:

```
// Однострочный комментарий
/* Многострочный
комментарий */
```

В листинге 4.3 демонстрируется применение рассмотренных выше типов комментариев. Временами вы будете сталкиваться с синтаксисом `<!--` и `//-->`. Такой вид комментариев используется для сокрытия кода от браузеров ранних версий, которые могут не распознать дескрипторы `<script>`. В настоящее время это уже не является проблемой, однако не мешает отметить данную возможность на случай, если вам по ряду причин доведется разрабатывать код, предназначенный для исполнения в устаревших браузерах. Все, что помещено в комментарии в листинге 4.3, на экран не выводится; это легко заметить на рис. 4.3.

Листинг 4.3. Использование комментариев JavaScript

```
<html>
<head>
  <title>JavaScript. Полное руководство</title>
</head>
```

```
<body>
  <script type="text/javascript">
  <!--
    // Переменные
    var firstName = "Васисуалий";
    var lastName = "Лоханкин";
    var internetAddress = "vaswashtub@crowsoutskirts.com";

    /*----------------------------------------
      Вывод на экран имени и фамилии пользователя,
      а также его адреса электронной почты.
    ----------------------------------------*/

    // Комбинирование трех строк
    document.writeln(firstName + " " + lastName + "<br>");
    document.writeln("адрес электронной почты: " + internetAddress);
  // -->
  </script>
</body>
</html>
```

Возможность сокрытия кода от JavaScript предоставляет вам возможность документировать разрабатываемые сценарии. Хорошим стилем программирования считается добавление к коду создаваемой программы примечаний по проекту, дружественных напоминаний или предостережений. Это, вне всяких сомнений, поможет как вам, так и другим программистам впоследствии понять, для чего предназначен тот или иной раздел вашей программы.

Большую пользу комментарии приносят и во время отладки сценариев. Вы можете сокрыть какую-то часть кода для того, чтобы локализовать возникшую ошибку, а затем, после внесения соответствующих исправлений, восстановить сокрытый коды, просто удалив дескрипторы комментариев. В результате можно сэкономить немало драгоценного времени.

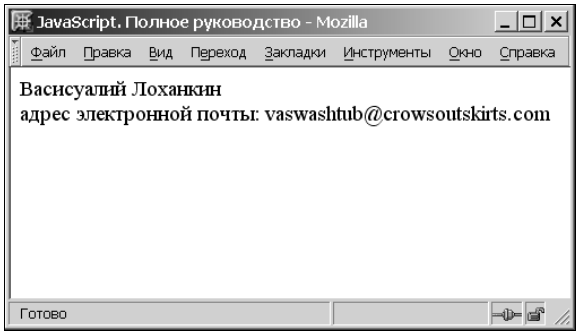

Рис. 4.3. *Комментарии в окне браузера не отображаются*

Функции

В своей простейшей форме функция представляет собой сценарий, который в любое время может быть вызван по его имени. Этот механизм совершенствует JavaScript в двух аспектах. Когда HTML-документ считывается Web-браузером, поддерживающим JavaScript, этот браузер обнаруживает все встроенные сценарии и выполняет их команды одну за другой. Все это работает прекрасно до тех пор, пока какая-то часть вашей программы или даже вся программа должна ожидать некоторое время, прежде чем наступят условия для ее выполнения. Представление такой части программы в виде функции и присвоение ей имени – это прекрасная возможность выполнить соответствующий сценарий в удобный момент в будущем.

Когда происходит специфическое событие, вы также можете выполнить этот сценарий, воспользовавшись ранее присвоенным ему именем. Еще одно преимущество функций состоит в возможности повторно использования сценария без необходимости периодического набора с клавиатуры одного и того же кода. Вместо этого вы можете просто указать имя, назначенное функции, и тем самым запустить код, содержащийся в теле данной функции.

В листинге 4.4 показано, как JavaScript выполняет функцию. Сначала выясняется, в каком месте программы данная функция была объявлена. Подобно тому, как должны быть объявлены все другие переменные, функция также должна быть объявлена. Убедитесь, что все объявления функций заключены в дескрипторы `<script>`.

НА ЗАМЕТКУ

Рекомендуется размещать объявления функций в разделе `<head>` HTML-документа. Благодаря этому функции будут загружены браузером еще до того, как дойдет очередь до их вызова в теле программы.

Чтобы вызвать объявленную функцию, достаточно поместить обращение к функции в требуемое место программного кода. Главная программа помещается в тело документа и заключается в собственный набор дескрипторов `<script>`. Взгляните на рис. 4.4 и вы увидите, как это все работает в браузере.

Листинг 4.4. Внедрение JavaScript-функции

```
<html>
<head>
  <title>JavaScript. Полное руководство</title>
  <script type="text/javascript">
  <!--
    function displayMessage() {
      document.write("JavaScript-функции использовать очень просто!<br>");
    }
  // -->
  </script>
</head>
<body>
  <script type="text/javascript">
  <!--
```

```
    document.write("Вызов JavaScript-функции...<p>");
    displayMessage();
    document.write("<\/p>Завершено. ");
  // -->
  </script>
</body>
</html>
```

Начиная с главного сценария, JavaScript, как обычно, выполняет первый оператор, а затем достигает вызова функции `displayMessage()`, которая отображает на экране сообщение. Эта функция отыскивается в памяти, а затем начинается выполнение первой строки ее кода. После отображения на экране фразы "JavaScript-функции использовать очень просто!" выводится символ разрыва строки. Далее достигается конец кода функции, и выполнение программы возобновляется с того места главного сценария, в котором оно было прервано.

На рис. 4.4 нетрудно видеть последовательность вывода сообщений. Если вы имеете дело с большой программой, в которой должно многократно отображаться одно и тоже сообщение, его вывод можно организовать в виде функции, а в нужных местах программы поместить обращение к этой функции. Если впоследствии возникнет необходимость обновить выдаваемое на экран сообщение, достаточно будет однократно внести соответствующие изменения, к тому же в одном месте кода. Внесите изменения в тело функции, и должным образом изменится вся программа. Дополнительные сведения о функциях и их достоинствах можно найти в главе 7 книги.

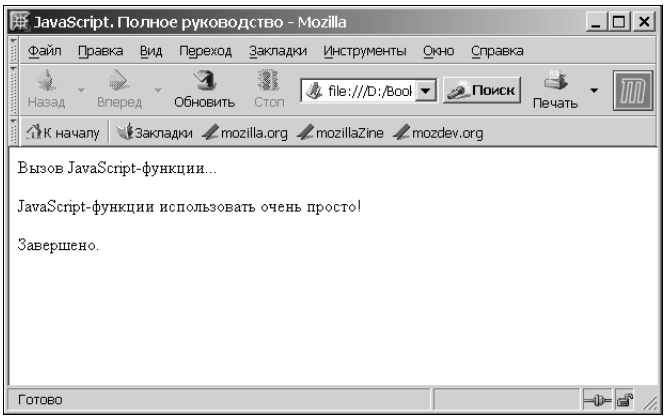

Рис. 4.4. *Вызов JavaScript-функции*

Объекты JavaScript

JavaScript не является объектно-ориентированным языком, подобным Java или C++, поскольку в нем не реализовано понятие класса. И хотя в JavaScript отсутствует наследование на основе классов, тем не менее, имеется возможность наследования на базе прототипов. В силу этого обстоятельства JavaScript попадает в категорию языков программирования, основанных на объектах. По прогнозам, язык JavaScript 2, который в

настоящее время находится в стадии разработки, в будущем будет все больше приближаться к истинному объектно-ориентированному языку, однако сейчас мы будем рассматривать объекты JavaScript в том виде, в каком они предстают перед нами в данный момент.

В этом разделе мы последовательно рассмотрим некоторые ключевые компоненты, связанные с объектами JavaScript. Разумеется, это отнюдь не исчерпывающее описание объектов JavaScript, а, скорее, обзор. Прежде всего, мы рассмотрим точечную и скобочную формы записи, которые применяются в отношении объектов JavaScript.

Точечная и скобочная формы записи

В JavaScript доступ к свойствам и методам объектов может быть осуществлен двумя способами: с использованием точечной и скобочной форм записи. Точечная и скобочная формы, примеры применения которых приведены ниже, обеспечивают метод иерархического доступа к этим свойствам и реализации объектных методов.

```
имяОбъекта.имяСвойства
имяОбъекта["имяСвойства"]
имяОбъекта.имяМетода(аргументы)
имяОбъекта["имяМетода"](аргументы)
```

Ссылка на текущий объект осуществляется с помощью специальной переменной `this`. В объявлении метода в рамках типа объекта (класса) ссылка на объект реализуется посредством переменной `this`. Собственно объект (имеется в виду текущий объект) – это объект, для которого объявляется метод.

Например, при объявлении типа объекта, представляющего комплексные числа, можно записать следующий прототип метода сложения двух комплексных чисел:

```
// объявление комплексных чисел
function Complex(real, img)
{
  this.real = real;
  this.img = img;
}

// получение вещественной части
Complex.prototype.getReal = function()
{
  return this.real;
}

// получение мнимой части
Complex.prototype.getImg = function()
{
  return this.img;
}

Complex.prototype.add = function(z)
{
  var a;
  var b;
  var rz;
```

```
    a = this.real + z.getReal();
    b = this.img + z.getImg();
    rz = new Complex(a,b);
    return rz;
}
```

Теперь можно создать два комплексных числа с помощью конструктора объекта `Complex`, как показано ниже:

```
var complexNum1 = new Complex(4,1);
var complexNum2 = new Complex(3,2);
```

Однажды созданные комплексные числа можно складывать друг с другом с помощью метода `add`, который определен в рамках объекта `Complex`. В следующей строке кода показано, как с помощью метода `add` прибавить к комплексному числу `complexNum2` комплексное число complexNum1:

```
complexNum1.add(complexNum2);
```

В методе `add` конструкция `this.real` ссылается на свойство `real` (вещественная часть) текущего объекта – в данном случае это `complexNum1.real`. Аналогично, `this.img` представляет собой адрес свойства `img` – то есть, `complexNum1.img`, которое хранит мнимую часть комплексного числа `complexNum1`.

Объектная модель JavaScript

Объекты JavaScript являются истинными объектами в том смысле, что они обладают свойствами и методами и способностью реагировать на события. Однако, как упоминалось в этой главе ранее, JavaScript не обладает таким же истинным механизмом наследования, который характерен для объектно-ориентированных языков программирования. Тем не менее, в JavaScript поддерживается наследование на базе прототипов. При анализе объектной модели JavaScript исключительно важно рассматривать ее именно в этом контексте. В отличие от иерархии классов, основанной на наследовании, объектная модель JavaScript представляет собой иерархию контейнеров, как показано на рис. 4.5. Если ранее вам доводилось иметь дело с объектно-ориентированными языками программирования, подобными Java или C++, это может оказаться наиболее важной поправкой, которую потребуется внести в свое мировоззрение, дабы успешно разрабатывать сценарии на языке JavaScript.

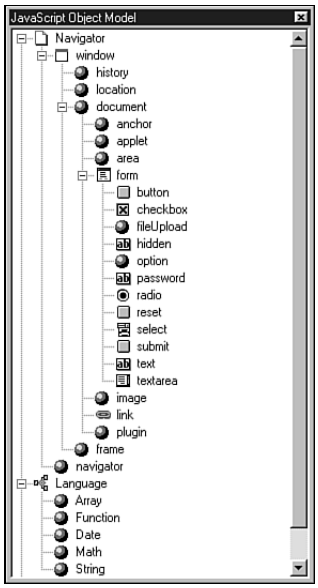

Рис. 4.5. *Иерархия встроенных JavaScript-объектов*

Иерархия контейнеров (**containership**) – это принцип, согласно которому один объект может содержать в себе другой объект. Если вы еще раз посмотрите на рис. 4.5, то обнаружите, что отношение, существующее между объектом `Form` и объектом `Radio`, не является отношением предка и потомка (или класса и подкласса), но представляет собой отношение между контейнером и его со-

держимым. Если прибегнуть к другой формулировке, то не существует генеалогического родства между этими объектами, поскольку они не происходят друг от друга.

Иерархия контейнеров в JavaScript

Иерархия контейнеров – это важное понятие, с которым необходимо разобраться, чтобы успешно разрабатывать JavaScript-сценарии и приложения, причем не только с точки зрения того, как один объект соотносится с другим объектом, но и в практическом смысле, то есть каким образом ссылаться на объекты. Вспомните, что при обсуждении точечной формы записи ранее в этой главе мы выяснили, что для ссылки на свойство или метод объекта используется точка, которая обозначает отношение владения. Например, в приведенной ниже конструкции метод `write` принадлежит объекту `Document`:

```
document.write("<h1>Заседание продолжается зпт миллион поцелуев.<\/h1>");
```

В то же время это можно расширить таким образом, чтобы покрыть не только свойства и методы объекта, но также и объекты, которые может содержать данный объект. Такой способ доступа к объектам возможен благодаря объектной модели документов (document object model), которая реализована в браузере. Например, если требуется получить имя объекта `Button`, следует воспользоваться приведенным ниже оператором:

```
buttonName = document.formMain.okButton.name;
```

`document` – это имя по умолчанию объекта `Document`, а `formMain` – имя объекта `Form`, который содержит кнопку `okButton`.

При работе с объектами важно знать, когда необходимо ссылаться на объекты-контейнеры. Например, объект `Window` является по существу объектом наивысшего уровня, с которым вы имеете дело в коде на клиентской стороне. Большинство ссылок на объекты выполняются в пределах их иерархии контейнеров. Предыдущий пример `document.write` можно переписать в следующем виде:

```
window.document.write("<h1>Заседание продолжается зпт миллион поцелуев.<\/h1>");
```

Хотя в большинстве случаев ссылки на `window` можно игнорировать, эти ссылки необходимы, когда вы имеете дело со множеством окон и фреймов. Например, код в листинге 4.5 создает объект `Window` в функции `showStats()`, а затем закрывает его в функции `closeWindow()`, которая вызывается во время выгрузки главного документа. Метод `window.open` открывает новое окно и возвращает ссылку на него. Затем с помощью метода `windowObject.document.open` документ открывается для записи. Метод `window.document.close` закрывает этот поток. Если ссылка соответствует допустимому объекту типа окно, она используется для закрывания окна при помощи метода `close`, когда выгружается главный документ.

Листинг 4.5. Создание нового окна и запись в него информации

```
<html>
<head>
<title>JavaScript. Полное руководство</title>
<script type="text/javascript" language="JavaScript">
```

```
<!--
  var windowObject

  // Создание окна состояния
  function showStats() {
    windowObject = window.open("", "ViewStats",
                               "toolbar=0,width=300,height=200,resizable=1");
    windowObject.document.open();
    windowObject.document.write("<h2>В этом месяце все поставленные задачи
↳успешно выполнены. Поздравляем!</h2>");
    windowObject.document.close();
  }

  // Закрытие окна состояния, если оно существует
  function closeWindow(){
    if((typeof(windowObject) == "object") && !windowObject.closed){
      windowObject.close();
    }
  }
// -->
</script>
</head>
<body onunload="closeWindow()">
  <h1>
    Щелкните на следующей кнопке для просмотра статистики за месяц.
  </h1>
  <form>
    <input type="button" value="Просмотреть статистику" onclick="showStats()">
  </form>
</body>
</html>
```

Объект `Window` – это единственный объект, который не требует соблюдения жесткости в объектных ссылках. Например, если вы хотите сослаться на форму внутри HTML-страницы, то должны указать ее родительский объект (`Document`), чтобы можно было понять, на какой конкретно объект осуществляется ссылка в рамках объектной модели документов. Если же необходимо сослаться на первую форму в объекте `Document` и получить количество элементов в ней, следует воспользоваться таким оператором:

```
var num = document.forms[0].length;
```

Даже несмотря на то, что форма имеет атрибут имени `name`, все равно нужно добавлять ссылку на родительский объект:

```
var num = document.queryForm.length;
```

Свойства

Свойства в JavaScript напоминают атрибуты данных объекта. Свойства объекта описывают характеристики и особенности конкретного объекта. В дополнение к спе-

циальным характеристикам и идентифицирующим значениям, атрибуты объекта могут представлять состояние определенного объекта или роль, которую способен играть объект в заданный момент времени.

Решая задачу моделирования проектов, объектный тип `Project` можно определить следующим образом:

```
function Project(members, leader, currentMilestone, time){
  this.members = members;
  this.leader = leader;
  this.currentMilestone = currentMilestone;
  this.time = time;
}
```

Затем можно создать объект `myProject`, представляющий проект по разработке программного обеспечения:

```
var memberGroup = new Array();
var currTime = new Date();
var myProject = new Project(memberGroup, "Балаганов", "начало", currTime);
```

Объект `myProject` содержит в себе группу лиц, описанных в объекте `memberGroup`. Руководителем проекта является `"Балаганов"`, а текущим этапом – `"начало"`, поскольку работа над проектом только началась. Переменные `memberGroup` и `currTime` содержат, соответственно, объект `Array` и объект `Date`, которые здесь не описываются.

В дополнение к точечной форме записи существуют и другие способы доступа к свойствам объекта. Следующий пример служит иллюстрацией записи через массив:

```
имяОбъекта["имяСвойства"]
```

Приведенная ниже строка демонстрирует индексацию через порядковые числа:

```
имяОбъекта[целочисленныйИндекс]
```

В последнем примере возвращается атрибуты с номером `целочисленныйИндекс`.

Методы

Под методом понимается услуга, которую данный объект предлагает другим объектам. В общем случае методы подразделяются на следующие четыре категории:

- **Модификатор** (**modifier**). Это метод, который изменяет состояние объекта. Такой метод изменяет значение одного и более атрибутов данных объекта. Широко известным методом-модификатором является функция `set`, которая устанавливает значение одного конкретного атрибута объекта.
- **Селектор** (**selector**). Это метод, который осуществляет доступ к атрибутам данных объекта, но не изменяет их. Одним из важных селекторов является функция `get`, которая возвращает (извлекает) значение одного конкретного атрибута объекта.
- **Итератор** (**iterator**). Это метод, который осуществляет доступ ко всем частям объекта, таким как все атрибуты данных, в некотором заданном порядке. Как следует из названия метода, он многократно выполняет операции над атрибутами данных объекта.

- **Конструктор** (**constructor**). Конструктор – это метод объектного типа, который создает новый объект на основе шаблона класса. Конструктор инициализирует новый объект значениями данных, которые передаются в разделе параметров этого метода.

В JavaScript доступ к методам объектов осуществляется с помощью точечной или скобочной формы записи:

```
имяОбъекта.имяФункции(агрументы)
имяОбъекта["имяФункции"](агрументы)
```

В общем случае HTML-документ в JavaScript представлен объектом `Document`. Этот объект `Document` поддерживает метод `write()`. С помощью упомянутого метода можно динамически расширять текстовое наполнение HTML-страницы, используя для этой цели JavaScript-код. Приведенный ниже оператор выводит на экран строку "`Пример текста`":

```
document.write("Пример текста");
```

При построении новых объектных типов в JavaScript используется следующий шаблон исходного кода:

```
function ObjectType (param1, param2, ...) {
  this.property1 = param1;
  this.property2 = param2;
}
ObjectType.prototype.method1 = function (param1, param2, ...)
{
  // Здесь находятся операторы функции
}
ObjectType.prototype.method2 = function (param1, param2, ...)
{
  // Здесь находятся операторы функции
}
```

Реализация методов объекта задается далее в объявлениях функций, соответствующих правилам, которые регламентируют написание обычных JavaScript-функций. Аргументами такой функции могут быть строки, числа, булевские значения и целые объекты. В листинге 4.6 показан пример определения и использования объекта, представляющего комплексные числа.

Листинг 4.6. Определение и использование объекта, представляющего комплексные числа

```
<html>
<head>

<title>
Комплексные числа
</title>

<script type="text/javascript">
// Определение комплексных чисел
```

```
function Complex(real, img){
  this.real = real;
  this.img = img;
}

Complex.prototype.add = function (z) {
  // Прибавление к текущему числу z, результат -- rz
  var a;
  var b;
  var rz;
  a = this.real + z.real;
  b = this.img + z.img;
  rz = new Complex (a,b);
  return rz;
}

Complex.prototype.toString = function () {
  return '(' + this.real + ' + ' + this.img + 'i)';
}

</script>
</head>

<body>
<script type="text/javascript">
var a = new Complex(3, 1);
var b = new Complex(1, 2);
var c = a.add(b);
document.write(a + ' + ' + b + ' = ' + c);
</script>
</body>
</html>
```

Код в листинге 4.6 определяет объект с именем `Complex`, который представляет комплексные числа. Определение объекта включает метод конструктора, метод `add`, предназначенный для сложения двух объектов `Complex`, и метод `toString`, обеспечивающий отображение чисел, которые образуют объект `Complex`.

В этом коде объект `Complex` также используется, начиная с вызовов конструктора `Complex` для создания двух комплексных чисел `a` и `b`. Затем объект `b` типа `Complex` прибавляется к объекту `a` типа `Complex` с помощью метода `add` объекта `a`. Результат сложения сохраняется в новом объекте типа `Complex`, который представлен переменной `c`.

И, наконец, метод `toString` вызывается автоматически для каждого из объектов, когда они отображаются в HTML-документе. Результаты выполнения кода можно видеть на рис. 4.6.

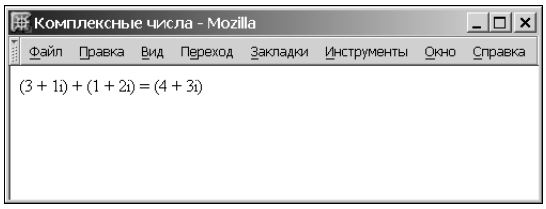

Рис. 4.6. *Сложение комплексных чисел*

События

Достаточно часто с помощью JavaScript-операторов создаются или изменяются элементы графического интерфейса пользователя, подобные формам и окнам. На рис. 4.7 представлен простой графический интерфейс пользователя.

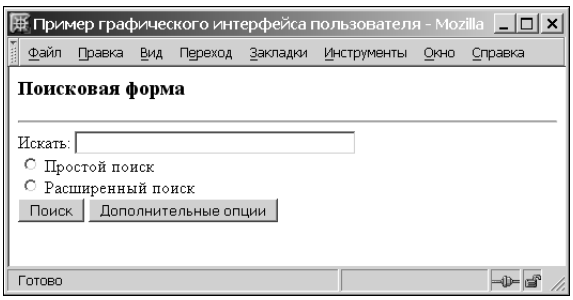

Рис. 4.7. *Графический интерфейс пользователя*

Код в листинге 4.7 служит только для создания формы, показанной на рис. 4.7. Остальные части кода пока не запрограммированы и выдают соответствующие предупредительные сообщения.

Листинг 4.7. Простой графический интерфейс пользователя

```
<html>
<head>
<title>Пример графического интерфейса пользователя</title>

<script type="text/javascript">
<!--

  // Функция поиска пока не реализована
  // here: create display of result list
  function fsearch(aForm){
    alert ("Извините, функция поиска пока не реализована");
  }

  // Отображение опций пока не реализовано
  function foptions(aForm){
    alert ("Извините, нет доступных опций");
  }

//-->
</script>
</head>

<body>
  <h3>
    Поисковая форма
  </h3>
```

```
  <hr>
  <form>
    Искать:
    <input type="text" name="tfield" size="40">
    <br>
    <input type="radio" name="search">
    Простой поиск
    <br>
    <input type="radio" name="search">
    Расширенный поиск
    <br>
    <input type="button" name="bsearch" value="Поиск"
           onclick="fsearch(this.form)">
    <input type="button" name="boptions" value="Дополнительные опции"
           onclick="foptions(this.form)">
  </form>
</body>
</html>
```

В контексте графического интерфейса пользователя событие представляет собой результат некоторого действия со стороны пользователя. Например, когда пользователь щелкает кнопкой мыши на какой-то кнопке графического интерфейса, происходит событие `click`. К другим событиям графического интерфейса пользователя относятся: щелчок на флажке, выбор строки в окне со списком, двойной щелчок на том или ином элементе, открытие и закрытие окна.

Наилучший способ управления графическим интерфейсом пользователя предполагает использование технологии программирования, управляемого событиями. События могут быть **перехвачены** и обработаны обработчиками событий JavaScript. Более подробную информацию о событиях можно найти в главе 18.

Резюме

В этой главе представлено общее описание фундаментальных компонентов языка JavaScript. Мы начали эту главу с изучения версий JavaScript и того, как они соотносятся с ECMAScript и JScript, а также с версиями наиболее популярных браузеров, доступных на рынке программных продуктов. И хотя такие стандарты, как ECMAScript, позволяют в какой-то степени сблизить браузеры друг с другом, тем не менее, внесенные в браузеры изменения несколько усложнили задачу отслеживания версий. По завершении обсуждения версий мы перешли к рассмотрению фактических компонентов JavaScript.

Начиная с первой строки кода, интерпретатор JavaScript считывает сценарий оператор за оператором, вычисляя лексемы, то есть наименьшие отдельные слова, фразы и символы, которые понимает JavaScript. Лексемами могут быть литералы, идентификаторы и операции.

Вы ознакомились с различными типами данных, реализованных в JavaScript: `number`, `boolean`, `string`, `function` и `object`. Для хранения данных одного типа за раз служат переменные. Чтобы объявить переменную, воспользуйтесь ключевым словом

var или просто инициализируйте новую переменную. Всегда старайтесь использовать осмысленные имена переменных, чтобы впоследствии было проще читать сценарий.

Вы также узнали, что JavaScript-выражения построены на основе более простых выражений, таких как, например, литералы или переменные, за счет применения соответствующих операций к этим простым выражениям. Результатом вычисления выражения является единственное значение. Самые распространенные выражения присваивают значения переменным. Операция присваивания = служит для присваивания значения правого операнда переменной, которая указывается в левом операнде. JavaScript поддерживает все стандартные математические операции либо посредством операций, определенных в синтаксисе языка, либо с помощью встроенного объекта math. Вы ознакомились с понятием функций, которые представляют собой сценарии и которые могут быть выполнены в любой момент до или после просмотра пользователем HTML-документа. С помощью механизма функций можно отложить выполнение сценария до любого удобного момента и выполнять его столько раз, сколько потребуется. Кроме того, использование функций существенно облегчает сопровождение сценариев. Более подробно функции рассматриваются в главе 7.

Вам также были даны начальные сведения о некоторых ключевых аспектах JavaScript-объектов. И хотя предложенное обсуждение не было исчерпывающим описанием JavaScript-объектов, тем не менее, вы узнали, что такое точечная и скобочная формы записи и как они используются для доступа к свойствам и методам объектов. В последующих главах вы узнаете об объектах JavaScript гораздо больше.

И, наконец, вы кратко ознакомились с событиями в JavaScript и с тем, как они используются для создания и манипулирования элементами графического интерфейса пользователя HTML-документа.