Таблица 3.1. Управляющие последовательности JavaScript

Последовательность	Представляемый символ
\0	Символ NUL (\u0000)
\b	«Забой» (\u0008)
\t	Горизонтальная табуляция (\u0009)
\n	Перевод строки (\u000A)
\v	Вертикальная табуляция (\u000B)
\f	Перевод страницы (\u000C)
\r	Возврат каретки (\u000D)
\"	Двойная кавычка (\u0022)
\'	Одинарная кавычка (\u0027)
\\	Обратный слэш (\u005C)
\x*XX*	Символ Latin-1, заданный двумя шестнадцатеричными цифрами *XX*
\u*XXXX*	Символ Unicode, заданный четырьмя шестнадцатеричными цифрами *XXXX*

3.2.3. Работа со строками

Одной из встроенных возможностей JavaScript является способность *конкатенировать* строки. Если оператор + применяется к числам, они складываются, а если к строкам – они объединяются, при этом вторая строка добавляется в конец первой. Например:

```
msg = "Hello, " + "world";  // Получается строка "Hello, world"
greeting = "Добро пожаловать на мою домашнюю страницу," + " " + name;
```

Для определения длины строки – количества содержащихся в ней 16-битных значений – используется свойство строки `length`. Например, длину строки `s` можно получить следующим образом:

```
s.length
```

Кроме того, в дополнение к свойству `length` строки имеют множество методов (как обычно, более полную информацию ищите в справочном разделе):

```
var s = "hello, world"      // Начнем с того же текста.
s.charAt(0)                 // => "h": первый символ.
s.charAt(s.length-1)        // => "d": последний символ.
s.substring(1,4)            // => "ell": 2-й, 3-й и 4-й символы.
s.slice(1,4)                // => "ell": то же самое
s.slice(-3)                 // => "rld": последние 3 символа
s.indexOf("l")              // => 2: позиция первого символа l.
s.lastIndexOf("l")          // => 10: позиция последнего символа l.
s.indexOf("l", 3)           // => 3: позиция первого символа "l", следующего
                            //       за 3 символом в строке
s.split(", ")               // => ["hello", "world"] разбивает на подстроки
s.replace("h", "H")         // => "Hello, world": замещает все вхождения подстроки
s.toUpperCase()             // => "HELLO, WORLD"
```

Не забывайте, что строки в JavaScript являются неизменяемыми. Такие методы, как `replace()` и `toUpperCase()` возвращают новые строки: они не изменяют строку, относительно которой были вызваны.

В стандарте ECMAScript 5 строки могут интерпретироваться как массивы, доступные только для чтения, и вместо использования метода `charAt()` к отдельным символам (16-битным значениям) строки можно обращаться с помощью индексов в квадратных скобках:

```
s = "hello, world";
s[0]           // => "h"
s[s.length-1]  // => "d"
```

Веб-броузеры, основанные на движке Mozilla, такие как Firefox, уже давно предоставляют такую возможность. Большинство современных броузеров (заметным исключением из которых является IE) последовали за Mozilla еще до того, как эта особенность была утверждена в стандарте ECMAScript 5.

3.2.4. Сопоставление с шаблонами

В языке JavaScript определен конструктор `RegExp()`, предназначенный для создания объектов, представляющих текстовые шаблоны. Эти шаблоны описываются с помощью *регулярных выражений*, синтаксис которых был заимствован языком JavaScript из языка Perl. И строки, и объекты `RegExp` имеют методы, позволяющие выполнять операции сопоставления с шаблоном и поиска с заменой при помощи регулярных выражений.

`RegExp` не относится к числу фундаментальных типов данных языка JavaScript. Подобно объектам `Date`, они просто являются специализированной разновидностью объектов с удобным прикладным интерфейсом. Грамматика регулярных выражений и прикладной интерфейс отличаются повышенной сложностью. Они подробно описываются в главе 10. Однако поскольку объекты `RegExp` обладают широкими возможностями и часто используются на практике, мы коротко познакомимся с ними в этом разделе.

Несмотря на то что объекты `RegExp` не относятся к фундаментальным типам данных языка, они имеют синтаксис литералов и могут вставляться непосредственно в текст программы на языке JavaScript. Текст, заключенный в пару символов слэша, интерпретируется как литерал регулярного выражения. За вторым символом слэша из этой пары может следовать один или более символов, которые модифицируют поведение шаблона. Например:

```
/^HTML/           // Соответствует символам H T M L в начале строки
/[1-9][0-9]*/     // Соответствует цифре, кроме нуля, за которой следует любое число цифр
/\bjavascript\b/i // Соответствует подстроке "javascript"
                  // как отдельному слову, учитывает регистр символов
```

Объекты `RegExp` обладают множеством полезных методов. Кроме того, строки также обладают методами, которые принимают объекты `RegExp` в виде аргументов. Например:

```
var text = "testing: 1, 2, 3"; // Образец текста
var pattern = /\d+/g           // Соответствует всем вхождениям одной или более цифр
pattern.test(text)             // => true: имеется совпадение
text.search(pattern)           // => 9: позиция первого совпадения
```