Регулярные выражения и объекты RegExp

ГЛАВА

38

В этой главе...

Web-разработчики, работавшие на Perl (и других языках создания Web-приложений), знакомы с тем, насколько мощным средством являются регулярные выражения (regular expressions) для обработки входящих данных или их форматирования для вывода в формате HTML или сохранения в базе данных на сервере. Любая задача, требующая операций поиска и замены текста, может быть красиво решена благодаря гибкости и выразительности регулярных выражений. Броузеры Navigator 4 и Internet Explorer 4 (и IE5.5 в более полной мере) наделили этим мощным средством и язык JavaScript.

В наибольшей степени достоинства применения регулярных выражений в JavaScript проявляются для тех, кто разрабатывает CGI-программы на сервере. Однако не стоит недооценивать приложения этого "языка внутри языка" и на стороне клиента. Если в сценариях производится проверка данных или другая синтаксическая обработка текста, предпочтительнее обращаться к регулярным выражениям, а не сложным функциям JavaScript.

Регулярные выражения и шаблоны

В нескольких главах ранее в этой книге выражения были определены как любые последовательности идентификаторов, ключевых слов и/или операторов, результатом выполнения которых является определенное значение. Это описание подходит и для регулярных выражений, однако за ними скрыто и нечто большее. В сущности, в регулярном выражении используется последовательность символов, определяющая шаблон текста. Такой шаблон используется для поиска блока текста в строке путем их сравнения.

Опытный программист на JavaScript может указать на наличие методов `string.indexOf()` и `string.lastIndexOf()`, позволяющих установить, где в строке содержится требуемая подстрока. Эти методы замечательно работают, когда требуется точное (посимвольное) совпадение. Если же требуется произвести более сложное сравнение (например, определить, содержит ли строка пятизначный почтовый код?), приходится отказываться от этих удобных методов и создавать функции синтаксического анализа. В этом и состоит прелесть регулярных выражений: они позволяют определить подстроку поиска, указывающую, что должно, а что не должно совпадать.

Простейшим шаблоном регулярного выражения является тот, что используется в методе `string.indexOf()`. Это не более, чем текст, совпадение с которым требуется найти. Одним из способов создания регулярного выражения в JavaScript является его окружение символами косой черты. Рассмотрим, например, строку:

```
Oh, hello, do you want to play Othello in the school play?
```

Предположим, что задачей сценария является замена формальных терминов неформальными, например, слова `hello` на `hi`. Стандартная функция поиска и замены использует простейший шаблон требуемой подстроки. В JavaScript шаблон (регулярное выражение) определяется с помощью символов косой черты. Для удобства и хорошей читаемости кода я обычно присваиваю регулярное выражение переменной, как в следующем примере:

```
var myRegExpression = /hello/
```

В сочетании с некоторыми методами объекта регулярных выражений или строкового объекта этот шаблон совпадает со строкой `hello`, где бы ни присутствовала эта последовательность букв. Проблема возникает в цикле поиска и замены: метод находит не только отдельное слово `hello`, но и часть `hello`, например в слове `Othello`.

Написание другой процедуры поиска и замены, которая будет обрабатывать только отдельные слова, представляет собой сложную задачу. Здесь недостаточно простого расширения шаблона пробелами с обеих сторон слова `hello`, поскольку до или после букв в строке могут находиться символы пунктуации — запятая, дефис, двоеточие или что-нибудь еще. К счастью, регулярные выражения обеспечивают простой способ указания общих характеристик, включая и параметр, называемый границей слова. Для этого используется символ `\b` (символ обратной косой черты и строчная буква `b`). Если переопределить шаблон с учетом этих спецификаций, регулярное выражение будет выглядеть следующим образом:

```
var myRegExpression = /\bhello\b/
```

Когда JavaScript использует это регулярное выражение как параметр специального метода строкового объекта, производящего операции поиска и замены, он заменяет на `hi` только отдельное слово `hello`, оставляя без изменений слово `Othello`.

Если вы все еще изучаете JavaScript и не имеете достаточного опыта работы с регулярными выражениями в других языках, за это упущение вам придется заплатить определенную цену: изучение “жаргона” регулярных выражений отнюдь не простое занятие, а сами выражения, наполненные большим количеством специальных символов, выглядят иногда как полуночный бред. Цель этой главы — познакомить вас с синтаксисом регулярных выражений, реализованных в JavaScript, а не представить руководства по обучению этому “языку в языке”. Более важным является понимание того, как JavaScript управляет регулярными выражениями как объектами, а также различий между копиями объектов регулярных выражений и статическим объектом `RegExp`. Примеры в данной главе продемонстрируют широкие возможности регулярных выражений.

Основы языка

Чтобы глубже познакомить читателя с синтаксисом регулярных выражений, я разбил эту тему на три раздела. В первом рассказано о простейших выражениях (примеры некоторых из них уже встречались здесь). Второй посвящен большому диапазону специальных символов, используемых в спецификациях строк поиска. И, наконец, третий раздел знакомит с использованием скобок, которые не просто помогают группировать выражения для изменения порядка операций (как это происходит в обычных математических выражениях), но позволяют также временно хранить промежуточные результаты более сложных выражений для воссоздания строк после их синтаксического анализа.

Простейшие шаблоны

В простейшем регулярном выражении для определения строки поиска никакие специальные символы не используются. Поэтому, если, например, все пробелы в строке требуется заменить символами подчеркивания, шаблон пробела будет выглядеть так:

```
var re = / /
```

Пробел заключен между символами косой черты. Проблема этого выражения заключается в том, что в нем указан поиск лишь первого экземпляра пробела в строке. Однако регулярные выражения можно "инструктировать" о том, что строка поиска применяется глобально. Для этого используется модификатор `g`:

```
var re = / /g
```

Когда переменная `re` указывается как параметр метода `replace()`, который использует регулярные выражения (он описан далее в этой главе), замена выполняется во всей строке, а не только в первом экземпляре заменяемой подстроки (символа пробела). Отметьте, что модификатор указывается после закрывающего символа косой черты в операторе, задающем регулярное выражение.

Сравнение данных с помощью регулярных выражений (как и многие аспекты JavaScript) чувствительно к регистру. Это поведение можно отменить, воспользовавшись модификатором, задающим сравнение без учета регистра. Следующее выражение

```
var re = /web/i
```

соответствует `web`, `Web` или любой другой комбинации строчных и прописных букв в этом слове (указанных, естественно, в том же порядке). В заключительной части регулярного выражения можно комбинировать модификаторы. Например, следующее выражение производит глобальный поиск без учета регистра:

```
var re = /web/gi
```

В соответствии с третьим изданием стандарта ECMA-262, броузеры IE5.5 и NN6 позволяют применять атрибут регулярного выражения, указывающий на обработку нескольких строк (т.е. строки, где в качестве разделителя используется символ возврата каретки). Таким модификатором является символ `m`.

Специальные символы

Большая часть словаря регулярных выражений JavaScript позаимствована из языка Perl. В некоторых случаях JavaScript предлагает альтернативу для упрощения синтаксиса, но при этом поддерживает и версии Perl для тех, кто уже имеет опыт работы с его регулярными выражениями.

Сами по себе регулярные выражения могут включать краткое представление таких параметров, как тип символов искомой строки, способ окружения искомой строки в тексте, а также количество символов отдельного типа в просматриваемом тексте. Это значительно расширяет возможности поиска. Типы символов задаются, в основном, с помощью специальных знаков перехода (т.е. букв с предшествующим им символом обратной косой черты), а частота их появления и диапазон — с помощью символов пунктуации и группировки.

Как вы видели в примере, приведенном ранее, `\b` задает границу слова в строке поиска. Модификаторы регулярных выражений JavaScript перечислены в табл. 38.1. Частью словаря являются так называемые метасимволы — символы выражений, которые сами по себе не используются для поиска совпадения, но служат командами языка регулярных выражений.

Таблица 38.1. Метасимволы поиска совпадения в регулярных выражениях JavaScript

Символ	Совпадение	Пример
`\b`	Граница слова	`/\bor/` совпадает с `origami` и `or`, но не с `normal`
		`/or\b/` совпадает с `traitor` и `or`, но не `perform`
		`/\bor\b/` совпадает только с целым словом `or`
`\B`	Не граница слова	`/\Bor/` совпадает с `normal`, но не с `origami`
		`/or\B/` совпадает с `normal` и `origami`, но не с `traitor`
		`/\Bor\B/` совпадает с `normal`, но не с `origami` или `traitor`
`\d`	Цифра от 0 до 9	`/\d\d\d/` совпадает с `212` и `415`, но не с `B17`
`\D`	Не цифра	`/\D\D\D/` совпадает с `ABC`, но не с `212` или `B17`
`\s`	Одиночный пустой символ	`/over\sbite/` совпадает с `over bite`, но не с `overbite` или `over  bite`
`\S`	Одиночный непустой символ	`/over\Sbite/` совпадает `c over-bite`, но не с `overbite` или `over bite`
`\w`	Буква, цифра или символ подчеркивания	`/A\w/` совпадает с `A1` и `AA`, но не с `A+`
`\W`	Не буква, цифра или символ подчеркивания	`/A\W/` совпадает с `A+`, но не с `A1` или `AA`
`.`	Любой символ, кроме символа новой строки	`/.../` совпадает с `ABC`, `1+3`, `A 3` и любыми тремя символами
`[...]`	Набор символов	`/[AN]BC/` совпадает с `ABC` и `NBC`, но не с `BBC`
`[^...]`	Набор неподходящих символов	`/[^AN]BC/` совпадает с `BBC` и `CBC`, но не с `ABC` и `NBC`

С символами, перечисленными в табл. 38.1, не следует путать последовательности знаков перехода, используемые в строках: табуляции (`\t`), новой строки (`\n`), возврата каретки (`\r`), прокрутки страницы (`\f`) и вертикальной табуляции (`\v`).

Остановимся подробнее на метасимволах `[...]` и `[^...]`. В квадратных скобках можно указать или отдельные символы (как показано в табл. 38.1), или диапазоны символов, или и то, и другое. Например, метасимвол `\d` можно определить как `[0-9]`, т.е. любая цифра от нуля до девяти. Если требуется совпадение только с цифрой 2 и цифрами от 6 до 8, применяется спецификация `[26-8]`. Аналогично, метасимвол `\w` представляет собой `[A-Za-z0-9_]`, напоминая о том, что по умолчанию при сравнении в регулярных выражениях учитывается регистр.

Каждый символ, перечисленный в табл. 38.1 (за исключением наборов символов, заключенных в квадратные скобки), соответствует одному символу в регулярном выражении. Однако в большинстве случаев неизвестно, как отформатированы входные данные, какова длина слова или количество цифр в числе. Для указания частоты появления определенного символа или типа символов (заданных, как показано в табл. 38.1) используется дополнительный набор метасимволов. Если вы знакомы с командной строкой, применяемой для управления операциями во многих операционных системах, то наверняка сталкивались с символами-заместителями, используемыми и в регулярных выражениях. Метасимволы такого типа приведены в табл. 38.2.

Таблица 38.2. Количественные метасимволы в регулярных выражениях JavaScript

Символ	Количество совпадений	Пример
*	Ноль и большее количество раз	/Ja*vaScript/ совпадает с JvaScript, JavaScript и JaaavaScript, но не с JovaScript
?	Ноль и большее количество раз	/Ja?vaScript/ совпадает с JvaScript или JavaScript, но не с JaaavaScript
+	Один и большее количество раз	/Ja+vaScript/ совпадает с JavaScript или JaavaScript, но не с JvaScript
{n}	Точно n раз	/Ja{2}vaScript/ совпадает с JaavaScript, но не с JvaScript или JavaScript
{n,}	n или большее количество раз	/Ja{2,}vaScript/ совпадает с JaavaScript или JaaavaScript, но не с JavaScript
{n,m}	По крайней мере, n раз, но не более, чем m раз	/Ja{2,3}vaScript/ совпадает с JaavaScript или JaaavaScript, но не с JavaScript

Каждый метасимвол из табл. 38.2 применяется к символу, непосредственно предшествующему ему в регулярном выражении, например, метасимволу из табл. 38.1. Приведенное ниже выражение совпадает с подстрокой, содержащей две цифры, разделенные одной или более гласными:

```
/\d[aeiouy]+\d/
```

Последний набор метасимволов предназначен для поиска подстроки в определенной позиции строки. Под позицией здесь понимается не смещение, о последнем позволяет узнать функциональность самих регулярных выражений, а, скорее, то, где искать подстроку — в начале или конце строки (если это важно). Метасимволы позиционирования перечислены в табл. 38.3.

Таблица 38.3. Метасимволы позиционирования в регулярных выражениях JavaScript

Символ	Совпадает с расположением	Пример
^	В начале строки	/^Fred/ совпадает с Fred is OK, но не с I'm with Fred или Is Fred here?
$	В конце строки	/Fred$/ совпадает с I'm with Fred, но не с Fred is OK или Is Fred here?

Рассмотрим следующий пример: требуется найти совпадение с римской цифрой в начале строки (но не в середине). Если документ содержит нумерацию римскими цифрами, все элементы, выровненные по левому краю, можно найти с помощью следующего регулярного выражения:

```
/^[IVXMDCL]+\./
```

Это выражение совпадает с любой комбинацией римских цифр, после которой следует точка (точка является специальным символом в регулярных выражениях, как показано в табл. 38.1, поэтому ее необходимо выделять символом обратной косой черты), если римская цифра находится непосредственно в начале строки и ей не предшествуют символы табуляции или пробелы. Данный шаблон не совпадает, например, с фразой `see Part IV`, поскольку здесь цифры не находятся в начале строки.

Группировка и обратные ссылки

Регулярные выражения удовлетворяют большинству законов старшинства операторов в JavaScript с учетом группировки с помощью скобок и логического оператора ИЛИ. Единственное отличие заключается в том, что оператор ИЛИ регулярного выражения обозначается одним символом вертикальной черты (`|`) в отличие от двойного символа, применяемого в JavaScript.

Кроме изменения порядка вычислений, скобки обладают дополнительными возможностями. Любой набор скобок (т.е. открывающая правая и закрывающая левая) сохраняет результат поиска совпадения с выражением. Скобки могут быть вложенными. Сохранение происходит автоматически, при этом данные хранятся в индексированном массиве, доступном сценариям и регулярным выражениям (при этом используется разный синтаксис). Доступ к таким элементам называется *обратной ссылкой*, поскольку регулярное выражение может указывать на результат совпадения выражения, который присутствует раньше. Такие сохраняемые подкомпоненты оказываются удобными в операциях замены, как продемонстрировано далее в этой главе.

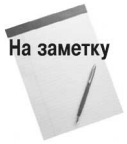
Взаимоотношения между объектами

Когда создается регулярное выражение и производятся простейшие операции с ним, JavaScript выполняет множество операций “за кулисами”. Не менее важным, чем применение самого языка регулярных выражений в сценариях, является правильное использование взаимосвязей между объектами JavaScript.

Первая концепция, которую необходимо освоить, состоит в следующем, —используется два типа объектов: объект регулярного выражения и статический объект `RegExp`. Оба они являются внутренними объектами JavaScript и не принадлежат объектной модели документа. Эти объекты работают совместно, но при этом имеют абсолютно разные наборы свойств, используемых в приложениях.

Когда создается регулярное выражение (даже с помощью синтаксиса `/ . . . /`), в JavaScript вызывается конструктор `new RegExp()` точно так же, как и конструктор `new Date()` для создания объекта определенной даты. Копия объекта регулярного выражения, возвращаемая конструктором, наделена несколькими свойствами, содержащими информацию о его данных. В то же время единичный статический объект `RegExp` поддерживает собственный набор свойств, отслеживающих действия с регулярными выражениями в текущем окне (или фрейме).

Чтобы ознакомиться с обычно “невидимыми” операциями, в этом разделе последовательно рассмотрено создание и применение регулярных выражений. В процессе рассмотрения показано, что происходит со всеми свойствами объекта, когда для поиска совпадения используется один из методов регулярных выражений.



Несколько свойств объекта регулярных выражений и статического объекта `RegExp` в приведенном далее описании недоступны в броузерах IE версии ниже 5.5. Все они присутствуют в NN4+. Сведения о совместимости приведены в таблицах отдельных свойств далее в этой главе.

В качестве строки, в которой будет осуществляться поиск, использовано начало монолога Гамлета (строка присвоена переменной mainString):

```
var mainString = "To be, or not to be: That is the question:"
```

Если требуется найти все экземпляры слова be, вначале необходимо создать регулярное выражение, которое совпадает с этим словом. Регулярное выражение создается для осуществления глобального поиска (оно присваивается переменной re):

```
var re = /\bbe\b/g
```

Чтобы совпадение происходило только для полного слова be (а не слога в другом слове), буквы окружены метасимволами границы слова. g представляет собой глобальный модификатор. Переменная re, которой присвоено выражение, представляет собой объект регулярного выражения, имеющий следующие свойства:

Объект.ИмяСвойства	Значение
re.source	"\bbe\bg"
re.global	true
re.ignoreCase	false
re.lastIndex	0

Свойство source регулярного выражения представляет собой строку, содержащую синтаксис регулярного выражения (без символов косой черты). Каждый из двух возможных модификаторов g и i имеет собственные свойства global и ignoreCase, чьи значения принадлежат булевому типу и указывают, присутствуют ли модификаторы в исходном выражении. И заключительное свойство lastIndex содержит индекс, начиная с которого будет происходить следующий поиск в главной строке. В созданном регулярном выражении это свойство имеет нулевое значение, т.е. поиск должен начинаться с первого символа строки. Это свойство предназначено для чтения/записи, поэтому его можно изменять в сценарии, если процесс требует дополнительного вмешательства. Как мы вскоре увидим, JavaScript со временем изменяет это значение, если для объекта указан модификатор глобального поиска.

Конструктор RegExp не просто создает объекты регулярных выражений. Как и объект Math, RegExp всегда доступен (один объект в каждом окне или фрейме). Он отслеживает действия с регулярными выражениями в сценарии. Его свойства показывают, какое сравнение с шаблоном регулярного выражения проводилось в сценарии. На этой стадии создания регулярного выражения в объекте RegExp установлено только одно из возможных свойств:

Объект.ИмяСвойства	Значение
RegExp.input	
RegExp.multiline	false
RegExp.lastMatch	
RegExp.lastParen	
RegExp.leftContext	
RegExp.rightContext	
RegExp.$1	
...	
RegExp.$9	

Последняя группа свойств (от $1 до $9) предназначена для хранения обратных ссылок. Поскольку определенное регулярное выражение не имеет скобок, эти свойства будут пустыми на протяжении всего примера, далее они опущены.

Когда объект регулярных выражений готов к использованию, вызывается метод exec(), который производит поиск в строке. Если метод находит совпадение, он возвращает третий объект, чьи свойства содержат информацию о найденном экземпляре (возвращаемый объект присваивается переменной foundArray):

```
var foundArray = re.exec(mainString)
```

JavaScript позволяет использовать более короткий вариант метода exec(), когда объект регулярного выражения превращается в метод:

```
var foundArray = re(mainString)
```

Обычно сценарий проверяет, не является ли массив foundArray пустым (т.е. совпадений не найдено) до того, как обрабатывать остальные объекты. В данном случае известно, что, по крайней мере, одно совпадение существует, поэтому вначале обратимся к результатам. Метод не только сгенерировал данные для массива foundArray, но и изменил некоторые свойства объекта RegExp, а также объекта регулярных выражений. Ниже показано текущее состояние свойств объекта регулярных выражений:

Объект.ИмяСвойства	Значение
re.source	"\bbe\bg"
re.global	true
re.ignoreCase	false
re.lastIndex	5

Главное и единственное изменение: значение lastIndex увеличилось до 5. Другими словами, вызов метода exec(), при котором было найдено соответствие, смещает индекс поиска для всех последующих вызовов на значение, равное сумме смещения начала подстроки и ее длины, т.е. 5 в данном случае. По этому индексу в исходной строке находится запятая, следующая после слова be. Если бы глобальный модификатор (g) не был указан, значение lastIndex оставалось бы равным нулю, поскольку последующие попытки поиска больше не предпринимались бы.

После вызова метода exec() значения некоторых свойств объекта RegExp стали содержать результаты поиска:

Объект.ИмяСвойства	Значение
RegExp.input	
RegExp.multiline	false
RegExp.lastMatch	"be"
RegExp.lastParen	
RegExp.leftContext	"To "
RegExp.rightContext	", or not to be: That is the question:"

Одно из свойств объекта содержит элемент строки, в котором было найдено совпадение с определенным регулярным выражением. Кроме того, другие свойства содержат текст исходной строки до и после совпадающего фрагмента (в этом примере свойство leftContext содержит пробел после To). И, в заключение, массив, возвращаемый методом exec(), также содержит некоторые дополнительные данные:

Объект.ИмяСвойства	Значение
foundArray[0]	"be"
foundArray.index	3
foundArray.input	"To be, or not to be: That is the question:"

Первый элемент массива, индексируемый с нуля, содержит фрагмент строки, совпадающий с регулярным выражением. Он равен значению свойства RegExp.lastMatch. Полная исходная строка содержится в свойстве input. Потенциально важным для сценария является индекс, по которому в исходной строке находится найденная подстрока. Зная его, из массива foundArray можно извлечь ту же информацию, что содержат свойства RegExp.leftContext (foundArray.input.substring(0, foundArray.index)) и RegExp.rightContext (foundArray.input.substring(foundArray.index, foundArray[0].length)).

Поскольку в регулярном выражении заложена многократная процедура поиска, метод exec() следует запускать без дополнительных изменений. Хотя оператор JavaScript будет иметь такой же вид, поиск начнется с нового значения свойства re.lastIndex. Рассмотрим, как изменятся значения всех соответствующих объектов после второго запуска метода:

```
var foundArray = re.exec(mainString)
```

Результаты запуска приведены ниже (изменения выделены полужирным шрифтом).

Объект.ИмяСвойства	Значение
re.source	"\bbe\bg"
re.global	true
re.ignoreCase	false
re.lastIndex	**19**
RegExp.input	
RegExp.multiline	false
RegExp.lastMatch	"be"
RegExp.lastParen	
RegExp.leftContext	**", or not to "**
RegExp.rightContext	**": That is the question:"**
foundArray[0]	"be"
foundArray.index	**17**
foundArray.input	"To be, or not to be: That is the question:"

Поскольку найдено второе совпадение, массив foundArray содержит новые данные. Свойство index указывает на индекс второго экземпляра подстроки, совпадающей с регулярным выражением. Значение lastIndex объекта регулярного выражения содержит индекс, с которого начнется следующий поиск (после второго слова be). А значения свойств объекта RegExp содержат новые подстроки, окружающие найденный экземпляр.

Если регулярное выражение имеет более свободный вид, чем обычно заданное слово, некоторые из опций могут различаться. Например, если регулярное выражение определено для поиска почтового индекса, значения RegExp.lastMatch и foundArray[0] будут содержать найденные в строке коды, которые могут отличаться друг от друга.

Запуск метода exec() в третий раз не находит экземпляров в исходной строке mainString. В результате массив foundArray равен null, извещая сценарию, что больше подстрок не найдено. Свойство lastIndex объекта регулярного выражения принимает нулевое значение, позволяя вести поиск в другой строке с начала. Важно, что свойства объекта RegExp сохраняют свои значения, соответствующие последней успешной попытке поиска. Поэтому, если метод exec() запускается в цикле, выход из которого осуществляется после первой неуспешной попытки поиска, объект RegExp все еще содержит данные последней успешной процедуры поиска, позволяя обрабатывать их в сценарии в дальнейшем.

Использование регулярных выражений

Несмотря на сложные внутренние действия, выполняемые при обработке регулярных выражений, JavaScript поддерживает набор методов, позволяющих с легкостью решать различные задачи (если вы в достаточной степени овладели синтаксисом регулярных выражений, чтобы создавать их корректно). В этом разделе представлены примеры синтаксиса регулярных выражений, предназначенные для решения различных задач, которые могут оказаться полезными для многих Web-страниц.

Есть ли совпадение?

Как уже говорилось ранее, для поиска подстроки в строке используются методы string.indexOf() и string.lastIndexOf(). Чтобы воспользоваться функциональными возможностями регулярных выражений, существуют два других метода:

объектРегВыражения.test(*строка*)
строка.search(*объектРегВыражения*)

Первый представляет собой метод объекта регулярного выражения, а второй — строкового объекта. Оба метода решают одну и ту же задачу и изменяют значения тех же объектов, но, при этом, возвращают различные значения: значение булевого типа в методе test() и символьное смещение в search() (или -1, если совпадения не найдено). Каким методом воспользоваться, зависит от ситуации: иногда необходимо просто определить, есть ли совпадение, а иногда — найти смещение нужной подстроки.

В листинге 38.1 продемонстрировано использование метода search(), позволяющего получить значение булевого типа и смещение при нахождении совпадения. Здесь задан определенный текст и шаблон регулярного выражения (по которому осуществляется поиск пятизначного числа). Вам рекомендуется поэкспериментировать с другими строками и регулярными выражениями. Поскольку в этом сценарии объект регулярного выражения создается с помощью конструктора new RegExp(), заключать выражение в символы косой черты нет необходимости.

Листинг 38.1. Поиск совпадения

```
<HTML>
<HEAD>
<TITLE>Got a Match?</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function findIt(form) {
      var re = new RegExp(form.regexp.value)
      var input = form.main.value
      if (input.search(re) != -1) {
         form.output[0].checked = true
```

```
      } else {
         form.output[1].checked = true
      }
}
function locateIt(form) {
      var re = new RegExp(form.regexp.value)
      var input = form.main.value
      form.offset.value = input.search(re)
}
</SCRIPT>
</HEAD>
<BODY>
<B>Use a regular expression to test for the existence of a string:</B>
<HR>
<FORM>
Enter some text to be searched:<BR>
<TEXTAREA NAME="main" COLS=40 ROWS=4 WRAP="virtual">
The most famous ZIP code on Earth may be 90210.
</TEXTAREA><BR>
Enter a regular expression to search:<BR>
<INPUT TYPE="text" NAME="regexp" SIZE=30
      VALUE="\b\d\d\d\d\d\b"><P>
<INPUT TYPE="button" VALUE="Is There a Match?"
      onClick="findIt(this.form)">
<INPUT TYPE="radio" NAME="output">Yes
<INPUT TYPE="radio" NAME="output">No <P>
<INPUT TYPE="button" VALUE="Where is it?"
      onClick="locateIt(this.form)">
<INPUT TYPE="text" NAME="offset" SIZE=4><P>
<INPUT TYPE="reset">
</FORM>
</BODY>
</HTML>
```

Получение информации о найденной подстроке

В следующем примере необходимо не просто убедиться, что дата, введенная в поле, имеет желаемый формат, но и извлечь ее компоненты и определить день недели. Здесь используется довольно сложное регулярное выражение, поскольку проверяется диапазон введенных значений (компонента месяца должна иметь значение не больше 12, а числа — не больше 31). Не учитывается только количество дней в месяце. Однако регулярное выражение и применяемый метод извлекают объект даты в таком формате, который позволяет провести дополнительную проверку диапазона. Однако помните, что это не единственный способ проверки значения даты в формах. В главе 43 предложены дополнительные способы проверки, в которых регулярные выражения не используются, позволяющие обеспечить совместимость с ранними версиями броузеров.

В листинге 38.2 представлена страница, содержащая поле для ввода даты, кнопку для ее обработки и поле для вывода длинной версии даты, включающей день недели. В начале функции, выполняющей всю работу, создается два массива (используется синтаксис создания массивов, поддерживаемый в JavaScript 1.2), содержащих названия месяцев и дней недели. Эти массивы используются лишь в том случае, если пользователь вводит корректную дату.

Затем оператор определяет регулярное выражение, которое будет сравниваться с введенными пользователем значениями. Если вы понимаете смысл выражения, то увидите, что компоненты даты разделяются дефисом или символом косой черты (`[\-\/]`). В регулярном вы-

ражении эти символы необходимо предварить символом косой черты. Определение каждой из трех компонент заключено в круглые скобки. Это важно для различных объектов, создаваемых регулярным выражением для дальнейшего управления компонентами.

Вот краткое описание элементов, искомых с помощью регулярного выражения:

- Строка, начинающаяся после разделителя слов
- Строковое значение месяца: цифра 1 плюс цифра от 0 до 2; или (необязательная) цифра 0 плюс цифра от 1 до 9
- Дефис или символ косой черты
- Строковое значение числа: цифра 0 плюс цифра от 1 до 9; или цифра 1 или 2 плюс цифра от 0 до 9; или цифра 3 плюс цифра 0 или 1
- Вновь дефис или символ косой черты
- Строковое значение года: 19 или 20 плюс две цифры

Дополнительная пара скобок должна окружать сегмент `19|20`, чтобы хотя бы одно из двух значений было присоединено к двум следующим цифрам. Без скобок по логике две последних цифры присоединяются к числу 20.

Регулярное выражение используется в методе `exec()`. Возвращаемое значение присваивается переменной `matchArray`. Здесь можно воспользоваться и методом `string.match()`. Последующая обработка даты происходит только, если совпадение является успешным (т.е. все условия регулярного выражения соблюдены).

Круглые скобки, окружающие какой-либо сегмент регулярного выражения, указывают JavaScript, что каждое найденное значение необходимо присвоить элементу объекта `matchArray`. Месяц присваивается элементу `matchArray[1]`, число — `matchArray[2]`, год — `matchArray[3]` (`matchArray[0]` содержит строку совпадения целиком). Благодаря этому сценарий может извлечь каждую компоненту даты для построения строки, содержащей текстовую версию даты (с использованием массивов, определенной в начальной части функции). Кроме того, эти значения нужны для создания нового объекта даты, который вычисляет и день недели. После того как все компоненты даты известны, они конкатенируются и результат присваивается полю отображения результата. Если строка, введенная пользователем, не соответствует регулярному выражению в методе `exec()`, сценарий выводит в этом поле сообщение об ошибке.

Листинг 38.2. Извлечение данных при нахождении совпадения

```
<HTML>
<HEAD>
<TITLE>Got a Match?</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function extractIt(form) {
     var months = ["January","February","March","April","May","June",
        "July","August","September","October","November","December"]
     var days = ["Sunday","Monday","Tuesday","Wednesday","Thursday",
        "Friday","Saturday"]
     var re = /\b(1[0-2]|0?[1-9])[\-\/](0?[1-9]|[12][0-9]|3[01])[\-
\/]((19|20)\d{2})/
     var input = form.entry.value
     var matchArray = re.exec(input)
     if (matchArray) {
        var theMonth = months[matchArray[1] —1] + " "
        var theDate = matchArray[2] + ", "
        var theYear = matchArray[3]
```

```
         var dateObj = new Date(matchArray[3],matchArray[1]-
             1,matchArray[2])
         var theDay = days[dateObj.getDay()] + " "
         form.output.value = theDay + theMonth + theDate + theYear
      } else {
         form.output.value = "An invalid date."
      }
}
</SCRIPT>
</HEAD>
<BODY>
<B>Use a regular expression to extract data from a string:</B>
<HR>
<FORM>
Enter a date in the format mm/dd/yyyy or mm-dd-yyyy:<BR>
<INPUT TYPE="text" NAME="entry" SIZE=12><P>
<INPUT TYPE="button" VALUE="Extract Date Components"
     onClick="extractIt(this.form)"><P>
The date you entered was:<BR>
<INPUT TYPE="text" NAME="output" SIZE=40><P>
<INPUT TYPE="reset">
</FORM>
</BODY>
</HTML>
```

Замена строк

Для демонстрации применения регулярных выражений в операциях поиска и замены в качестве примера я выбрал задачу, с которой сталкиваются многие разработчики страниц, которым приходится выводить и форматировать большие числа. В базах данных большие целые числа, обычно, хранятся без запятых (запятые в англоязычной нотации используются для разделения троек цифр в целых числах; для отделения дробной части от целой в числах с плавающий точкой используется символ точки — *прим. ред.*). Однако число, имеющее больше пяти-шести цифр, воспринимать трудно. Кроме того, если пользователю требуется ввести большое число, запятые помогут ввести его без ошибок.

В таких ситуациях регулярные выражения используются вместе с функцией JavaScript `string.replace()` (см. главу 34). Это метод имеет два обязательных аргумента: регулярное выражение и строку, заменяющую найденный экземпляр. Строка-заменитель может быть свойством объекта `RegExp`, если этот метод вызывается непосредственно после метода `exec()`.

Листинг 38.3 демонстрирует, как несколько строк сценария способны выполнить большой объем работы, когда в них используются регулярные выражения. Страница содержит три поля. В первом необходимо ввести любое число. Затем необходимо щелкнуть на кнопке Insert Commas (Вставить запятые), в результате чего запускается функция `commafy()`. Результат выводится во втором поле. Кроме того, во втором поле можно ввести число, разделенное запятыми и, щелкнув на кнопке Remove Commas (Удалить запятые), произвести обратное преобразование с помощью функции `decommafy()`.

Модификаторы регулярного выражения позволяют обработать любое положительное или отрицательное число. Ключом к такому действию являются круглые скобки вокруг отдельных частей регулярного выражения. Одна компонента заключает в себе все символы, не включенные во вторую группу: обязательный набор из трех цифр. Другими словами, регулярное выражение начинает обрабатывать строку с конца, выделяя трехсимвольные сегменты и вставляя запятую после каждого найденного экземпляра.

В цикле while просматривается и изменяется вся строка (на самом деле строковый объект не изменяется — генерируется новая строка и присваивается старой переменной). Применяется метод test(), так как сценарию не нужно значение, возвращаемое методом exec(). Метод test() влияет на свойства объекта регулярного выражения и объекта RegExp подобно методу exec(), но более эффективен в данном случае. После первого запуска test() часть строки, совпадающая с первым сегментом, присваивается свойству RegExp.$1, часть со вторым (если таковая найдется) — свойству RegExp.$2. Обратите внимание, что результаты метода exec() не присваиваются какой-либо переменной, поскольку в данном приложении объект массива, генерируемый этим методом, не нужен.

Далее следует несколько хитрый фрагмент кода. Метод string.replace() вызывается с текущим значением строки (num) в качестве начального. Шаблоном для поиска служит регулярное выражение, определенное в начале тела функции. Строка-заместитель выглядит несколько странно: она заменяет все, что совпадает с регулярным выражением на RegExp.$1, запятую и RegExp.$2. Объект RegExp не должен быть частью ссылок, используемых в параметрах метода replace(). Поскольку регулярное выражение совпадает со всей строкой num, метод replace() перестраивает строку из ее компонент, добавляя между ними запятую. Каждый вызов метода replace() устанавливает значение num для следующей итерации цикла while и метода test().

Цикл продолжается до тех пор, пока совпадения не закончатся, т.е. в строке больше не содержится отдельных трехсимвольных сегментов. После этого результат выводится во втором поле страницы.

Листинг 38.3. Замена строк с помощью регулярных выражений

```
<HTML>
<HEAD>
<TITLE>Got a Match?</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function commafy(form) {
      var re = /(-?\d+)(\d{3})/
      var num = form.entry.value
      while (re.test(num)) {
         num = num.replace(re, "$1,$2")
      }
      form.commaOutput.value = num
}
function decommafy(form) {
      var re = /,/g
      form.plainOutput.value = form.commaOutput.value.replace(re,"")
}
</SCRIPT>
</HEAD>
<BODY>
<B>Use a regular expression to add/delete commas from numbers:</B>
<HR>
<FORM>
Enter a large number without any commas:<BR>
<INPUT TYPE="text" NAME="entry" SIZE=15><P>
<INPUT TYPE="button" VALUE="Insert commas"
onClick="commafy(this.form)"><P>
The comma version is:<BR>
<INPUT TYPE="text" NAME="commaOutput" SIZE=20><P>
```

```
<INPUT TYPE="button" VALUE="Remove commas"
onClick="decommafy(this.form)"><P>
The un-comma version is:<BR>
<INPUT TYPE="text" NAME="plainOutput" SIZE=15><P>
<INPUT TYPE="reset">
</FORM>
</BODY>
</HTML>
```

Удаление запятых представляет собой еще более простой процесс. Регулярным выражением является просто запятая с установленным флажком глобального поиска. Метод `replace()` реагирует на него повторением процесса до тех пор, пока все экземпляры не будут заменены. В данном случае заместитель является пустой строкой. Примеры использования регулярных выражений со строковыми объектами содержатся в разделах, посвященных обсуждению методов `string.match()`, `string.replace()` и `string.split()` в главе 34.

Объект регулярного выражения

Свойства	Методы	Обработчики событий
`constructor`	`compile()`	
`global`	`exec()`	
`ignoreCase`	`test()`	
`lastIndex`		
`multilane`		
`source`		

Синтаксис

Доступ к свойствам или методам объекта регулярного выражения:

`объектРегВыражения.свойство|метод([параметры])`

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

Описание объекта

Объект регулярного выражения создается в сценарии на лету. Каждый объект содержит собственный шаблон и другие свойства. Какой стиль создания объекта использовать зависит от того, как регулярное выражение будет дальше использоваться в сценариях.

Когда регулярное выражение создается при помощи литерной системы обозначения (т.е. с двумя символами косой черты), оно автоматически компилируется для эффективной обработки при последующем выполнении оператора присваивания. То же самое справедливо и при использовании конструктора `new RegExp()` и указании шаблона (с дополнительными модификаторами) в качестве параметра. Если регулярное выражение строго определено, используйте литерную систему обозначения; если часть или все регулярное выражение поступает из внешнего источника (например текстового поля), укажите его как параметр конструк-

тора `new RegExp()`. Скомпилированное регулярное выражение можно применять в части сценария. Однако в таком виде регулярные выражения не сохраняются на диске и не существуют за пределами сценария документа.

Иногда параметры регулярного выражения изменяются с каждой итерацией цикла. Например, если его содержимое изменяется непосредственно в цикле `while`, необходимо представить выражение явно, как это показано в следующем фрагменте сценария:

```
var srchText = form.search.value
var re = new RegExp() // пустой конструктор
while (некоторое условие) {
   re.compile("\\s+" + srchText + "\\s+", "gi")
   операторы, изменяющие srchText
}
```

В каждой итерации цикла объекту присваивается новое выражение (оно конкатенируется с одним или несколькими пустыми символами с обеих сторон и текста, который изменяется при каждой итерации). Затем объект компилируется для использования с любым из соотвествующих ему методов.

Свойства

constructor

См. `string.constructor` (глава 34)

global

ignoreCase

Значение: булево Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					✓

Два этих свойства отражают модификаторы регулярного выражения `g` и `i`, если они заданы. Свойства предназначены только для чтения. Их значения задаются на этапе создания объекта и являются независимыми друг от друга.

lastIndex

Значение: целочисленные Чтение/Запись

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					✓

Свойство `lastIndex` указывает на индекс в исходной строке, начиная с которого производится следующий поиск с помощью текущего объекта регулярного выражения. При создании объекта регулярного выражения это свойство равно нулю, т.е. предварительный поиск не проводился. По умолчанию поиск начинается с начала строки.

Если указан глобальный модификатор, свойство `lastIndex` принимает большее значение после того, как объект используется для поиска в строке. Оно равно позиции в исходной строке, следующей непосредственно после экземпляра найденной подстроки (не включая символов последней). После нахождения последнего экземпляра метод обнуляет

значение lastIndex для последующих сеансов поиска. Изменяя это значение на ходу, можно влиять на поведение процедур поиска. Например, если нужно, чтобы поиск начинался с четвертого символа строки, необходимо изменить значение свойства сразу же после создания объекта, например:

```
var re = /шаблон/
re.lastIndex = 3  // четвертый символ в строке
```

См. также свойство index объекта массива результатов совпадения.

multiline

Значение: булево Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость				✓					✓

Свойство multiline указывает, производится поиск по нескольким строкам в исходной строке или нет. Этим управляет модификатор регулярного выражения m. NN4+ поддерживает свойство объекта RegExp с таким же именем (см. следующий раздел).

См. также свойство RegExp.multiline.

source

Значение: строка Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

Свойство source — строковое представление регулярного выражения, с помощью которого был создан объект. Оно предназначено только для чтения.

Методы

compile("*шаблон*", ["g" | "i" | "m"])

Возвращаемое значение: объект регулярного выражения.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

Для компиляции регулярного выражения, чье содержимое изменяется в сценарии, используется метод compile(). См. обсуждение этого объекта ранее. Другие операторы создания регулярных выражений (литерная система обозначений и конструктор new RegExp()) компилируют их автоматически. Модификатор шаблона m поддерживается в броузерах IE5.5+ и NN6+.

exec("*строка*")

Возвращаемое значение: объект массива совпадений или null.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

Метод `exec()` проводит в строке, заданной в аргументе, поиск хотя бы одного экземпляра подстроки, совпадающей с объектом регулярного выражения. Поведение этого метода сходно с действиями метода `string.match()` (хотя метод `match()` является более мощным при глобальном поиске совпадений). Обычно вызов метода `exec()` происходит непосредственно после создания объекта регулярного выражения, как в следующем примере:

```
var re = /шаблон/
var matchArray = re.exec("строка")
```

Метод `exec()` выполняет несколько действий. При успешном нахождении совпадений обновляются свойства объекта регулярного выражения объекта `RegExp`, принадлежащего окну. Кроме того, метод возвращает объект, содержащий дополнительные данные о выполненной операции. В табл. 38.4 приведены свойства возвращаемого объекта.

Таблица 38.4. Свойства объекта массива найденных совпадений

Свойство	Описание
`index`	Индекс (отсчитываемый от нуля) начала подстроки в строке
`input`	Полный текст исходной строки
`[0]`	Строка последнего совпадения
`[1],...[n]`	Совпадения с компонентами, заключенными в скобки

Некоторые из свойств возвращаемого объекта повторяют свойства объекта `RegExp`. Удобство их хранения заключается в том, что они содержатся в другом объекте, тогда как свойства объекта `RegExp` могут быть изменены при последующем вызове метода регулярного выражения. Общими для двух объектов являются свойства: `[0]` (свойство `RegExp.lastMatch`) и `[1],. . .[n]` (первые девять из них хранятся в свойствах `RegExp.$1. . .RegExp.$9`). Объект `RegExp` содержит только девять подкомпонент выражения, заключенных в скобки, а объект возвращаемого массива хранит их столько, сколько пар скобок присутствует в регулярном выражении.

Если совпадений не найдено, метод возвращает значение `null`. Пример применения этого метода приведен в листинге 38.2. Метод `exec()` позволяет использовать сокращенный вариант синтаксиса, когда регулярное выражение можно "превратить" в функцию, например,

```
var re = /шаблон/
var matchArray = re("строка")
```

См. также метод `string.match()`.

`test("строка")`

Возвращаемое значение: булево.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

Эффективным способом проверки, содержится ли в строке совпадение с регулярным выражением, является метод `test()`. Если совпадение присутствует, метод возвращает значение `true`, и `false` в ином случае. Если требуется больше информации, следует воспользоваться методом `string.search()`, который возвращает индекс найденной подстроки. Пример использования этого метода приведен в листинге 38.1.

См. также метод `string.search()`.

Объект RegExp

Свойства	Методы	Обработчики событий
input	(нет)	(нет)
lastMatch		
lastParen		
leftContext		
multilane		
prototype		
rightContext		
$1, ... $9		

Синтаксис

Доступ к свойствам объекта RegExp:

```
RegExp.свойство
```

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

Описание объекта

Начиная с Netscape Navigator 4 и Internet Explorer 4, броузеры поддерживают одну копию объекта RegExp для каждого окна или фрейма. Объект "следит" за действиями всех методов, связанными с регулярными выражениями (включая и несколько методов строкового объекта). Свойства этого объекта доступны не только в традиционной форме JavaScript, но и в краткой — в виде параметров метода string.replace() (см. листинг 38.3).

Поскольку один объект RegExp используется всеми методами, в которых применяются регулярные выражения, необходимо проявлять определенную осторожность при чтении или изменении его свойств. Необходимо убедиться, что объект RegExp не подвергся действию какого-либо другого метода. Большинство свойств изменяется при вызове любого из методов, поддерживающих регулярные выражения. Именно это и служит причиной того, что чаще используется объект массива, возвращаемый большинством методов, а не свойства объекта RegExp. Первый связан с определенным объектом регулярного выражения, на который не действуют методы, примененные к другим подобным объектам. Свойства объекта RegExp отражают последние операции с регулярными выражениями, независимо от того, с каким объектом регулярного выражения они связаны.

В листингах используются длинные имена свойств в формате JavaScript. Однако каждое свойство можно сократить до формата Perl. Сокращенные имена можно использовать, например, в методе string.replace().

Свойства

input

Значение: строка Чтение/Запись

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					✓

Свойство `RegExp.input` содержит исходную строку, в которой производится поиск подстроки, совпадающей с регулярным выражением. Во всех примерах ранее в этой главе это свойство было равно `null`. Так происходит в том случае, когда строка является параметром метода, использующего регулярные выражения.

Однако многие объекты документа, связанные с текстом, имеют определенную скрытую взаимосвязь с объектом `RegExp`. Если объекты текста, ссылки, элементов `TEXTAREA` или `SELECT` содержат обработчик события, вызывающий функцию, в которой применяются регулярные выражения, свойство `RegExp.input` устанавливается равным соответствующим текстовым данным объекта. В вызове обработчика или функции никаких параметров указывать не нужно. Для текстового объекта или элемента `TEXTAREA` свойство `input` становится равным содержимому объекта, для объекта `SELECT` оно равно тексту (не значению) выбранной опции, а для объекта ссылки — тексту, связанному с ссылкой и выделенному в броузере подчеркиванием (что отражено в свойстве объекта ссылки `text`).

То, что JavaScript автоматически устанавливает значение свойства `RegExp.input`, значительно упрощает код сценария. Любой из методов регулярных выражений можно вызвать, не указывая исходную строку как параметр. Когда аргумент опущен, JavaScript обращается к свойству `RegExp.input`. Это свойство можно установить и в процессе выполнения сценария. Сокращенная версия имени этого свойства `$_` (знак доллара и подчерк).

См. также свойство `input` объекта массива результатов совпадения.

multiline

Значение: булево Чтение/Запись

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					

Свойство `RegExp.multiline` определяет, будет ли поиск происходить в нескольких строках исходной строки (т.е. когда в одной строке содержится несколько строк, разделенных символом конца строки). Это свойство автоматически устанавливается в значение `true`, когда обработчик события объекта `TEXTAREA` запускает функцию, содержащую регулярные выражения. Это свойство можно изменять по ходу выполнения. Сокращенная версия имени свойства: `$*`. Эта версия свойства (в отличие от свойства `multiline` объекта регулярного выражения) не определена в спецификации ECMA-262 и поддерживается только в броузерах NN4+.

См. также свойство `multiline` объекта регулярного выражения.

lastMatch

Значение: строка Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					✓

После запуска метода, связанного с регулярными выражениями, всякий текст исходной строки, совпадающий с регулярным выражением, автоматически присваивается свойству `RegExp.lastMatch`. Это же значение присваивается и свойству `[0]` объекта массива, возвращаемого методами `exec()` и `string.match()` после нахождения соответствия. Сокращенная версия имени этого свойства: `$&`.

См. также свойство `[0]` объекта массива совпадений.

lastParen

Значение: строка Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					✓

Когда регулярное выражение содержит несколько компонент, заключенных в круглые скобки, результирующие строки хранятся в свойствах `$1,...$9` объекта `RegExp`. Кроме того, значение последней совпавшей компоненты хранится и в свойстве `RegExp.lastParen`, предназначенном только для чтения. Сокращенная версия имени свойства: `$+`.

См. также свойства `RegExp.$1,...$9`.

leftContext
rightContext

Значение: строка Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓					✓

После того как одним из методов было найдено совпадение с регулярным выражением, объект `RegExp` содержит ключевую контекстную информацию о нем. В свойстве `leftContext` хранится часть исходной строки слева от найденной подстроки (не включая последнюю). Следует отметить, что строка `leftContext` начинается с того места, с которого начинался последний поиск. Поэтому при каждом последующем поиске в одной строке свойство `leftContext` принимает разные значения.

Свойство `rightContext` содержит часть строки, которая начинается после найденной подстроки и простирается до конца строки. При последующих вызовах, естественно, это значение сокращается, пока не закончатся подстроки, совпадающие с регулярным выражением. В последнем случае оба свойства принимают значение `null`. Сокращенная версия имен свойств `leftContext` и `rightContext` — `` $` `` и `$'`, соответственно.

prototype

См. `String.prototype` (глава 34).

$1...$9

Значение: строка Только для чтения

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Совместимость			✓	✓			✓	✓	✓

При запуске метода, использующего регулярные выражения, результаты совпадения с компонентами, заключенными в скобки, сохраняются в этих девяти свойствах объекта `RegExp` (их часто называют обратными ссылками). Эти же значения (а также следующие, если их больше девяти) хранятся в объекте массива, который возвращает методы `exec()` и `string.match()`. Значения хранятся в порядке появления левых (открывающих) скобок в регулярном выражении независимо от степени вложенности компонент.

Обратные ссылки можно использовать во втором аргументе метода `string.replace()`, не используя при их указании `RegExp`. Идеальное решение состоит в том, чтобы инкапсулировать компоненты, которые необходимо перестроить, с символами замены. Например, следующий сценарий переставляет порядок имени и фамилии:

```
function swapEm() {
   var re = /(\w+),\s*(\w+)/
   var input = "Lincoln, Abraham"
   return input.replace(re,"$2 $1")
}
```

В методе `replace()` вторая компонента, заключенная в скобки (имя), размещается на первом месте, а затем, после пробела — первая (фамилия). Запятая, присутствующая в исходном варианте, отбрасывается.

См. также свойства `[1]...[n]` объекта массива совпадений.