Встроенные объекты Error, Math и RegExp

В этой главе приведены описания встроенных объектов **Error**, **Math** и **RegExp**.

3.12.1. Исключения: объект Error

Объекты **Error** создаются при возникновении ошибок в процессе выполнения сценария и содержат информацию об ошибке, которая используется операторами обработки исключений. К сожалению, реализации этого класса объектов не вполне соответствуют стандарту, поэтому мы сначала описываем объект **Error** согласно ECMAScript, а затем его реализацию в JScript (JavaScript, реализованный в обозревателях Netscape, вообще не поддерживает объекты **Error**).

3.12.1.1. Объект Error в ECMAScript

Все исключения подразделяются на *системные* и *пользовательские*. Системные исключения генерируются исполняющей системой в процессе выполнения сценария, пользовательские — самим сценарием с помощью оператора **throw**. Стандарт ECMAScript предусматривает следующие виды системных исключений:

Таблица 3.9. Исключения ECMAScript

Название	Описание
EvalError	Недопустимое обращение к функции **eval**.
RangeError	Числовое значение вне допустимого диапазона.
ReferenceError	Недопустимое значение ссылки.
SyntaxError	Синтаксическая ошибка.
TypeError	Несовместимые типы операндов.
URIError	Недопустимое обращение к функции работы с URI.

Объекты **Error** соответственно также могут быть созданы исполняющей системой или сценарием. Конструктор пользовательского объекта **Error** имеет вид `new Error(message)`, где `message` — текст сообщения об ошибке.

Свойства объекта Error

Свойство	Описание	Член прототипа
`constructor`	Конструктор, который создал объект.	Да
`message`	Текст сообщения об ошибке.	Да
`name`	Название исключения.	Да
`prototype`	Ссылка на прототип класса объектов.	Да

Методы объекта Error		
Свойство	**Описание**	**Член прототипа**
toString	Возвращает строку "[object Error]".	Да

Свойство message

Синтаксис: *объект*.message
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **message** является сообщение об ошибке данного *объекта*. Это свойство является свойством *объекта* по умолчанию.

Свойство name

Синтаксис: *объект*.name
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **name** является название данного *объекта*. В пользовательских объектах оно содержит строку "Error", а в системных объектах — строку, содержащую тип исключения согласно Таблицы 3.9.

3.12.1.2. Объект Error в JScript

В JScript, реализованном в обозревателях Internet Explorer 5.x, объект **Error** имеет нестандартный конструктор `new Error(number [, description]?)`. Здесь `number` — номер ошибки, `description` — текст сообщения о ней. При этом `number` является свойством по умолчанию и содержит в в старшем слове код источника ошибки (facility code), а в младшем слове — ее номер. Перечень всех системных ошибок JScript приведен в Приложении 14.

Для совместимости с ECMAScript объект **Error** в Internet Explorer 5.5 получил свойства `message` и `name`. При этом свойство `message` является синонимом свойства `description`, а `name` соответствует приведенному выше описанию.

Пример генерации исключения в JScript и вывода информации о нем:

```
function showErrorInfo(e) {
  document.write(e, ":<BR>");
  document.write("Источник ошибки: ", (e.number >> 16) & 0x1FFF, "<BR>");
  document.write("Номер ошибки: ", e.number & 0xFFFF, "<BR>");
  document.write("Описание ошибки: ", e.description);
}

var x;
try {
  x = y;                // Ошибка: переменная y не определена
}
catch (e) {             // Создает локальный объект e класса Error
  showErrorInfo(e);
}
```

Этот сценарий выведет на экран:

```
[object Error]:
Источник ошибки: 10
Номер ошибки: 5009
Описание ошибки: 'y' - определение отсутствует
```

3.12.2. Математические функции и константы: объект Math

Объект **Math** обеспечивает доступ к различным математическим константам и функциям. Он существует в единственном экземпляре и потому не имеет конструктора. Соответственно все его свойства и методы являются статическими и должны вызываться обращением к объекту **Math**, а не его реализациям. Прототипа объект **Math** не имеет.

Свойства объекта Math

Свойство	Описание
E	Основание натуральных логарифмов e.
LN10	Число ln 10.
LN2	Число ln 2.
LOG10E	Число lg e.
LOG2E	Число $\log_2 e$.
PI	Число π.
SQRT1_2	Квадратный корень из 1/2.
SQRT2	Квадратный корень из 2.

Методы объекта Math

Метод	Описание
abs	Возвращает абсолютную величину аргумента.
acos	Возвращает арккосинус аргумента.
asin	Возвращает арксинус аргумента.
atan	Возвращает арктангенс аргумента.
atan2	Возвращает арктангенс частного от деления аргументов.
ceil	Возвращает наименьшее целое число, большее или равное аргументу.
cos	Возвращает косинус аргумента.
exp	Возвращает экспоненту аргумента.
floor	Возвращает наибольшее целое число, меньшее или равное аргументу.
log	Возвращает натуральный логарифм аргумента.
max	Возвращает наибольший из аргументов.
min	Возвращает наименьший из аргументов.
pow	Возводит первый аргумент в степень, заданную вторым.
random	Генерирует случайное число в диапазоне от 0 до 1.
round	Округляет аргумент до ближайшего целого числа.

sin	Возвращает синус аргумента.
sqrt	Возвращает квадратный корень из аргумента.
tan	Возвращает тангенс аргумента.

Свойство E

Синтаксис: Math.E
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **E** является основание натуральных логарифмов e, которое приблизительно равно 2.718281828459045. Пример: `var x = Math.E`.

Свойство LN10

Синтаксис: Math.LN10
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **LN10** является натуральный логарифм числа 10 (ln 10), который приблизительно равен 2.302585092994046. Пример: `var x = Math.LN10`.

Свойство LN2

Синтаксис: Math.LN2
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **LN2** является натуральный логарифм числа 2 (ln 2), который приблизительно равен 0.6931471805599453. Пример: `var x = Math.LN2`.

Свойство LOG10E

Синтаксис: Math.LOG10E
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **LOG10E** является десятичный логарифм числа e (lg e), который приблизительно равен 0.4342944819032518. Пример: `var x = Math.LOG10E`.

Свойство LOG2E

Синтаксис: Math.LOG2E
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **LOG2E** является двоичный логарифм числа e ($\log_2 e$), который приблизительно равен 1.4426950408889634. Пример: `var x = Math.LOG2E`.

Свойство PI

Синтаксис: Math.PI
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **PI** является число π (отношение длины окружности к диаметру круга), которое приблизительно равно 3.1415926535897932. Пример: `var x = Math.PI`.

Свойство SQRT1_2

Синтаксис: Math.SQRT1_2
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **SQRT1_2** является квадратный корень из 1/2, который приблизительно равен 0.7071067811865476. Пример: `var x = Math.SQRT1_2`.

Свойство SQRT2

Синтаксис: Math.SQRT2
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **SQRT2** является квадратный корень из 2, который приблизительно равен 1.4142135623730951. Пример: `var x = Math.SQRT2`.

Метод abs

Синтаксис: Math.abs(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **abs** возвращает абсолютную величину *числа*. Примеры:

```
var x = Math.abs(2);   // x равно 2
var x = Math.abs(-2);  // x равно 2
```

Метод acos

Синтаксис: Math.acos(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **acos** возвращает арккосинус *числа*. Если *число* находится в диапазоне от -1 до 1 включительно, то результат находится в диапазоне от 0 до π. В противном случае результат равен **NaN**. Примеры:

```
var x;
with (Math) {
  x = acos(-1);// x равно 3.141592653589793
  x = acos(0); // x равно 1.5707963267948965
  x = acos(1); // x равно 0
  x = acos(2); // x равно NaN
}
```

Метод asin

Синтаксис: Math.asin(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **asin** возвращает арксинус *числа*. Если *число* находится в диапазоне от -1 до 1 включительно, то результат находится в диапазоне от -π/2 до +π/2. В противном случае результат равен **NaN**. Примеры:

```
var x;
with (Math) {
  x = asin(-1);// x равно -1.5707963267948965
  x = asin(0); // x равно 0
  x = asin(1); // x равно 1.5707963267948965
  x = asin(2); // x равно NaN
}
```

Метод atan

Синтаксис: Math.atan(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **atan** возвращает арктангенс *числа*. Результат находится в диапазоне от -π/2 до +π/2. Примеры:

```
var x;
with (Math) {
  x = atan(-Infinity); // x равно -1.5707963267948965
  x = atan(0);         // x равно 0
  x = atan(Infinity);  // x равно 1.5707963267948965
}
```

Метод atan2

Синтаксис: Math.atan2(*число1*, *число2*)
Аргументы: *число1*, *число2* — числовые выражения
Результат: числовое значение

Метод **atan2** возвращает арктангенс частного от деления *числа1* на *число2*. Результат находится в диапазоне от -π до +π и соответствует величине угла в радианах между осью абсцисс и вектором до точки с координатами (*число2*, *число1*). Примеры:

```
var x;
with (Math) {
  x = atan2(-0, -1);   // x равно -3.141592653589793
  x = atan2(-1, 0);    // x равно -1.5707963267948965
  x = atan2(0, 1);     // x равно 0
  x = atan2(1, 1);     // x равно 0.7853981633974483
  x = atan2(1, 0);     // x равно 1.5707963267948965
  x = atan2(0, -1);    // x равно 3.141592653589793
}
```

Метод ceil

Синтаксис: Math.ceil(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **ceil** возвращает наименьшее целое число, большее или равное *числа*. Примеры:

```
var x = Math.ceil(-2.95);      // x равно -2
var x = Math.ceil(2.95);       // x равно 3
```

Метод cos

Синтаксис: Math.cos(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **cos** возвращает косинус *числа*. Результат находится в диапазоне от -1 до +1. Примеры:

```
var x;
with (Math) {
  x = cos(0);          // x равно 1
  x = cos(PI/2);       // x равно 6e-17 (почти 0)
  x = cos(PI);         // x равно -1
}
```

Метод exp

Синтаксис: Math.exp(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **exp** возвращает экспоненту *числа* ($e^{число}$, где e — основание натуральных логарифмов). Если число больше 709.78, то возвращается **Inifinity**. Пример:

```
var x = Math.exp(1);   // x равно 2.718281828459045
```

Метод floor

Синтаксис: Math.floor(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **floor** возвращает наибольшее целое число, меньшее или равное *числа*. Примеры:

```
var x = Math.floor(-2.95);     // x равно -3
var x = Math.floor(2.95);      // x равно 2
```

Метод log

Синтаксис: Math.log(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **log** возвращает натуральный логарифм *числа*. Если *число* отрицательно, то возвращается **NaN**. Примеры:

```
var x = Math.log(Math.E);      // x равно 1
```

```
var x = Math.log(0);          // x равно -Infinity
var x = Math.log(-1);         // x равно NaN
```

Метод max

Синтаксис: Math.max(*число1*, …, *числоN*)
Аргументы: *число1*, …, *числоN* — числовые выражения
Результат: числовое значение

Поддержка: Соответствует стандарту.

Возвращает большее из значений первых двух аргументов, остальные аргументы игнорируются. Если аргументов меньше двух, возвращает NaN.

Метод **max** возвращает наибольшее из значений своих аргументов. Если аргументы не заданы, то он возвращает -**Inifinity**. Например, следующий сценарий

```
document.write(Math.max(1, 2, 3));
```

выведет на экран Internet Explorer число 3, а на экран Netscape Navigator число 2.

Метод min

Синтаксис: Math.min(*число1*, …, *числоN*)
Аргументы: *число1*, …, *числоN* — числовые выражения
Результат: числовое значение

Поддержка: Соответствует стандарту.

Возвращает меньшее из значений первых двух аргументов, остальные аргументы игнорируются. Если аргументов меньше двух, возвращает NaN.

Метод **min** возвращает наименьшее из значений своих аргументов. Если аргументы не заданы, то он возвращает **Inifinity**. Например, следующий сценарий

```
document.write(Math.min(3, 2, 1));
```

выведет на экран Internet Explorer число 1, а на экран Netscape Navigator число 2.

Метод pow

Синтаксис: Math.pow(*число1*, *число2*)
Аргументы: *число1*, *число2* — числовые выражения
Результат: числовое значение

Метод **pow** возвращает *число1*, возведенное в степень *число2* (*число1*^{число2}). Примеры:

```
var x;
with (Math) {
  x = pow(2, 3);       // x равно 8
  x = pow(100, 0);     // x равно 1
  x = pow(10, 0.5);    // x равно 3.162277660168379
}
```

Метод random

Синтаксис: Math.random()
Результат: числовое значение

Метод **random** генерирует случайное число в диапазоне от 0 включительно до 1 исключительно. Пример: `var x = Math.random()`.

Метод round

Синтаксис: Math.round(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **round** округляет *число* до ближайшего целого числа и возвращает его. Примеры:

```
var x;
with (Math) {
  x = round(2.49);     // x равно 2
  x = round(2.5);      // x равно 3
  x = round(-2.5);     // x равно -2
  x = round(-2.51);    // x равно -3
}
```

Метод sin

Синтаксис: Math.sin(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **sin** возвращает синус *числа*. Результат находится в диапазоне от -1 до +1. Примеры:

```
var x;
with (Math) {
  x = sin(0);          // x равно 0
  x = sin(PI/2);       // x равно 1
  x = sin(PI);         // x равно 1e-16 (почти 0)
}
```

Метод sqrt

Синтаксис: Math.sqrt(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **sqrt** возвращает квадратный корень из *числа*. Если *число* отрицательно, то возвращается **NaN**. Примеры:

```
var x = Math.sqrt(2);  // x равно 1.4142135623730951
var x = Math.sqrt(1);  // x равно 1
var x = Math.sqrt(-1); // x равно NaN
```

Метод tan

Синтаксис: Math.tan(*число*)
Аргументы: *число* — числовое выражение
Результат: числовое значение

Метод **tan** возвращает тангенс *числа*. Примеры:

```
var x;
with (Math) {
  x = tan(0);          // x равно 0
  x = tan(PI/2);       // x равно 16331778728383844
  x = tan(PI);         // x равно -1e-16 (почти 0)
}
```

3.12.3. Регулярные выражения: объект RegExp

Объект **RegExp** используется для создания регулярных выражений, подробно описанных в гл. 3.5. Там же описаны способы создания этих объектов.

Все описанные ниже свойства объекта **RegExp** являются статическими. Это означает, что они хранятся в единственном экземпляре и изменяются при каждой операции сопоставления с регулярным выражением.

Свойства объекта RegExp

Свойство	Описание	Член прототипа
`$1, ..., $9`	Хранят запомненные подстроки.	Нет
`$01, ..., $99`	Хранят запомненные подстроки.	Нет
`$&`	См. `lastMatch`.	Да
`&``	См. `leftContext`.	Да
`&'`	См. `rightContext`.	Да
`&+`	См. `lastParen`.	Да
`&*`	См. `multiline`.	Да
`constructor`	Конструктор, который создал объект.	Да
`global`	Значение опции глобального поиска.	Нет
`ignoreCase`	Значение опции не различать строчные и прописные буквы.	Нет
`input`	Последняя исходная строка.	Да
`lastIndex`	Номер позиции в строке для следующего сопоставления с образцом.	Нет
`lastMatch`	Последняя найденная подстрока.	Да
`lastParen`	Последняя запомненная подстрока.	Да
`leftContext`	Подстрока, предшествующая последней найденной подстроке.	Да
`multiline`	Значение опции многострочного поиска.	Нет
`prototype`	Ссылка на прототип класса объектов.	Нет
`rightContext`	Подстрока, следующая за последней найденной подстрокой.	Да
`source`	Текст регулярного выражения.	Нет

Методы объекта RegExp		
Метод	**Описание**	**Член прототипа**
compile	Компилирует регулярное выражение.	Да
exec	Выполняет сопоставление строки с образцом и заносит результаты в массив.	Да
test	Проверяет успешность сопоставления строки с образцом.	Да
toString	Преобразует регулярное выражение в строку.	Да
valueOf	Возвращает примитивное значение объекта.	Нет

Свойства $1, …, $9

Синтаксис: RegExp.$*n*
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Если часть регулярного выражения заключена в круглые скобки, то соответствующая ей подстрока запоминается для последующего использования. Значениями свойств **$1**, …, **$9** являются подстроки исходной строки, которые были запомнены в процессе последнего сопоставления с образцом. Регулярное выражение может содержать любое количество выражений в круглых скобках, но в объекте **RegExp** запоминаются только последние девять найденных соответствий. Пример: следующий сценарий

```
var re = new RegExp("(\\d*)\\s*(\\d*)","ig");
var arr = re.exec("111 2222 33333");
var s = "$1 = '" + RegExp.$1 + "' ";
s += "$2 = '" + RegExp.$2 + "' ";
s += "$3 = '" + RegExp.$3 + "'";
document.write(s);
```

выведет на экран обозревателя текст `$1 = '111' $2 = '2222' $3 = ''`.

Эти свойства объекта **RegExp** являются статическими и изменяются при каждой операции сопоставления с регулярным выражением. В методе String.replace они употребляются без имени объекта **RegExp**. См. пример в п. 3.5.4.

Свойства $01, …, $99

Синтаксис: RegExp.$*nn*
Атрибуты: { DontEnum, DontDelete, ReadOnly }
Поддержка: IE - Поддерживаются с версии 5.5.
NN - Не поддерживаются.

Если часть регулярного выражения заключена в круглые скобки, то соответствующая ей подстрока запоминается для последующего использования. Значениями свойств **$01**, …, **$99** являются подстроки исходной строки, которые были запомнены в процессе последнего сопоставления с образцом. Регулярное

выражение может содержать любое количество выражений в круглых скобках, но в этих свойствах запоминаются только последние 99 найденных соответствий. Пример: следующий сценарий

```
var re = new RegExp("(\\d*)\\s*(\\d*)","ig");
var arr = re.exec("111 2222 33333");
var s = "$01 = '" + RegExp.$01 + "' ";
s += "$02 = '" + RegExp.$02 + "' ";
s += "$03 = '" + RegExp.$03 + "'";
document.write(s);
```

выведет на экран обозревателя текст `$01 = '111' $02 = '2222' $03 = ''`.

Эти свойства объекта **RegExp** являются статическими и изменяются при каждой операции сопоставления с регулярным выражением. В методе `String.replace` они употребляются без имени объекта **RegExp**. См. пример в п. 3.5.4.

Свойство global

Синтаксис: *регвыр*.global
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **global** является значение опции "g" (глобальный поиск), заданное при создании объекта *регвыр*. Например, следующий сценарий

```
var re = new RegExp("a+b+c","ig");
document.write(re.global);
```

выведет на экран обозревателя текст `true`.

Свойство ignoreCase

Синтаксис: *регвыр*.ignoreCase
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **ignoreCase** является значение опции "i" (не различать строчные и прописные буквы), заданное при создании объекта *регвыр*. Например, следующий сценарий

```
var re = new RegExp("a+b+c","ig");
document.write(re.ignoreCase);
```

выведет на экран обозревателя текст `true`.

Свойство input

Синтаксис: RegExp.input
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **input** является последняя исходная строка, к которой применялось сопоставление с образцом. Это свойство объекта **RegExp** является статическим и изменяется при каждой операции сопоставления с регулярным выражением. Пример: следующий сценарий

```
var re = new RegExp("\\d+", "g");
var arr = re.exec("111 2222 33333");
document.write(RegExp.input);
```

выведет на экран обозревателя текст `111 2222 33333`.

Свойство lastIndex

Синтаксис: *регвыр*.lastIndex
Атрибуты: { DontEnum, DontDelete }

Значением свойства **lastIndex** является целое число, содержащее номер элемента строки, с которого начнется следующее сопоставление с образцом. При создании объекта *регвыр* этому свойству присваивается значение 0. Оно используется только в тех случаях, когда включена опция глобального поиска (т. е. свойство **global** имеет значение **true**). Пример:

```
var re = /(aha)/g;
var a = re.exec("aha");        // a равно ["aha", "aha"], re.lastIndex равно 3
var a = re.exec("aha");        // a равно [""], re.lastIndex по-прежнему равно
3
re.lastIndex = 0;      // повторить поиск
var a = re.exec("aha");        // a равно ["aha", "aha"], re.lastIndex равно 3
```

Свойство lastMatch ($&)

Синтаксис: RegExp.lastMatch
Атрибуты: { DontEnum, DontDelete, ReadOnly }
Поддержка: IE - Поддерживается с версии 5.5.
NN - Поддерживается с версии 4.0.

Значением свойства **lastMatch** является последняя найденная подстрока исходной строки. Это свойство объекта **RegExp** является статическим и изменяется при каждой операции сопоставления с регулярным выражением. Пример: следующий сценарий

```
var re = new RegExp("\\d+", "g");
var arr = re.exec("111 2222 33333");
document.write(RegExp.lastMatch);
```

выведет на экран обозревателя текст `111`.

Свойство lastParen (&+)

Синтаксис: RegExp.lastParen
Атрибуты: { DontEnum, DontDelete, ReadOnly }
Поддержка: IE - Поддерживается с версии 5.5.
NN - Поддерживается с версии 4.0.

Значением свойства **lastParen** является последняя запомненная подстрока исходной строки, соответствующая подвыражению регулярного выражения, заключенному в круглые скобки. Это свойство объекта **RegExp** является статическим и изменяется при каждой операции сопоставления с регулярным выражением. Пример: следующий сценарий

```
var re = new RegExp("(\\d+) (\\d+)", "g");
var arr = re.exec("111 2222 33333");
document.write(RegExp.lastParen);
```

выведет на экран обозревателя текст `2222`.

Свойство leftContext (&`)

Синтаксис: RegExp.leftContext
Атрибуты: { DontEnum, DontDelete, ReadOnly }
Поддержка: IE - Поддерживается с версии 5.5.
NN - Поддерживается с версии 4.0.

Значением свойства **leftContext** является подстрока исходной строки, предшествующая последней найденной подстроке. Это свойство объекта **RegExp** является статическим и изменяется при каждой операции сопоставления с регулярным выражением. Пример: следующий сценарий

```
var arr = / (\d+)/.exec("111 2222 33333");
document.write(RegExp.leftContext + "|" + RegExp.lastMatch + "|" +
RegExp.rightContext);
```

выведет на экран обозревателя текст `111| 2222| 33333`.

Свойство multiline ($*)

Синтаксис: *регвыр*.multiline
Атрибуты: { DontEnum, DontDelete, ReadOnly }
Поддержка: IE - Поддерживается с версии 5.5.
NN - Поддерживается с версии 4.0.

Значением свойства **multiline** является значение опции "m" (многострочный поиск), заданное при создании объекта *регвыр*. Например, следующий сценарий

```
var re = new RegExp("a+b+c","im");
document.write(re.ignoreCase);
```

выведет на экран обозревателя текст `true`.

В обозревателях Netscape опция "m" не поддерживается, а свойство **multiline** является статическим свойством объекта **RegExp**. Мы можем присваивать ему логическое значение, которое включает или выключает

многострочный поиск для всех регулярных выражений, например `RegExp.multiline = true`. Помните, что обработчики событий после своего завершения всегда сбрасывают значение этого свойства в **false**. Кроме того, Netscape поддерживает `$*` как синоним данного свойства.

Свойство rightContext (&')

Синтаксис: RegExp.rightContext
Атрибуты: { DontEnum, DontDelete, ReadOnly }
Поддержка: IE - Поддерживается с версии 5.5.
NN - Поддерживается с версии 4.0.

Значением свойства **rightContext** является подстрока исходной строки, следующая за последней найденной подстрокой. Это свойство объекта **RegExp** является статическим и изменяется при каждой операции сопоставления с регулярным выражением. Пример: следующий сценарий

```
var arr = / (\d+)/.exec("111 2222 33333");
document.write(RegExp.leftContext + "|" + RegExp.lastMatch + "|" +
RegExp.rightContext);
```

выведет на экран обозревателя текст `111| 2222| 33333`.

Свойство source

Синтаксис: *регвыр*.source
Атрибуты: { DontEnum, DontDelete, ReadOnly }

Значением свойства **source** является строка, содержащая регулярное выражение, заданное при создании объекта *регвыр*. Например, следующий сценарий

```
var re = new RegExp("a+b+c","ig");
document.write(re.source);
```

выведет на экран обозревателя текст `a+b+c`.

Метод compile

Синтаксис: *регвыр*.compile(*образец*, *опции*?)
Аргументы: *образец* — регулярное выражение
опции — необязательные опции поиска

Метод **compile** компилирует *образец* во внутренний формат хранения, что в дальнейшем ускоряет сопоставление с этим образцом. Компиляция регулярного выражения имеет смысл, если вы собираетесь использовать это выражение несколько раз. Пример:

```
var r = new RegExp("[A-Z]", "g");
r.compile("[a-z]", "g");
```

Метод exec

Синтаксис: *регвыр*.exec(*строка*)
Аргументы: *строка* — любое строковое выражение
Результат: массив результатов или **null**

Метод **exec** выполняет сопоставление *строки* с образцом, заданным *регвыр*. Если сопоставление с образцом закончилось неудачей, то возвращается значение **null**. В противном случае результатом является массив подстрок, соответствующих заданному образцу. В процессе сопоставления производится обновление всех свойств объекта *регвыр* (и тем самым всех свойств объекта **RegExp**).

Результирующий массив имеет следующие свойства:

- свойство **input** содержит исходную строку;
- свойство **index** содержит позицию найденной подстроки в исходной строке;
- свойство **length** равно n + 1, где n — количество подвыражений регулярного выражения, заключенных в круглые скобки;
- элемент 0 содержит найденную подстроку;
- элементы 1, …, n содержат подстроки, соответствующие подвыражениям регулярного выражения в круглых скобках.

Пример: следующий сценарий

```
var arr = /(\d+)\.(\d+)\.(\d+)/.exec("Я родился 21.05.1958");
document.write("Дата рождения: ", arr[0], "<br>");
document.write("День рождения: ", arr[1], "<br>");
document.write("Месяц рождения: ", arr[2], "<br>");
document.write("Год рождения: ", arr[3], "<br>");
```

выведет на экран обозревателя текст:

```
Дата рождения: 21.05.1958
День рождения: 21
Месяц рождения: 05
Год рождения: 1958
```

Включение в регулярное выражение опции глобального поиска позволяет многократно применять этот метод к исходной строке для последовательного выделения всех подстрок, соответствующих данному образцу. Например, следующий сценарий

```
var re = /\d+/g;
var s = "123 abc 456 def 789 xyz";
var result;
while (result = re.exec(s))
  document.write(result[0] + " ");
```

выведет на экран обозревателя строку `123 456 789`.

Примечание. Помните, что в подобных циклах нельзя выполнять операции с регулярными выражениями, поскольку они изменяют статические свойства объекта **RegExp**.

Метод test

Синтаксис: *регвыр*.test(*строка*)
Аргументы: *строка* — любое строковое выражение
Результат: логическое значение

Метод **test** выполняет сопоставление *строки* с образцом, заданным *регвыр* и возвращает **true**, нсли сопоставление с образцом прошло успешно, и **false** в противном случае. Этот метод эквивалентен выражению *регвыр*.exec(*строка*) != **null**.

Пример:

```
if (!/(\d+)\.(\d+)\.(\d+)/.test(str))
  document.write("Неверное значение строки!");
```

Метод toString

Синтаксис: *регвыр*.toString()
Результат: строковое значение

Метод **toString** преобразует *регвыр* в строковое значение. Например, сценарий

```
var x = new RegExp("a+b+c", "g");
document.write(x.toString());
```

выведет на экран обозревателя строку `/a+b+c/g`.