# 1 JavaScript简介

Javascript是一种由Netscape的LiveScript发展而来的原型化继承的面向对象的动态类型的区分大小写的客户端脚本语言，主要目的是为了解决服务器端语言，比如Perl，遗留的速度问题，为客户提供更流畅的浏览效果。当时服务端需要对数据进行验证，由于网络速度相当缓慢，只有28.8kbps，验证步骤浪费的时间太多。于是Netscape的浏览器Navigator加入了Javascript，提供了数据验证的基本功能。

一个完整的 JavaScript 实现是由以下 3 个不同部分组成的：核心（ECMAScript）、文档对象模型（Document Object Model，简称DOM）、浏览器对象模型（Browser Object Model，简称BOM）

## 1.1 JavaScript特点

* 编程风格：函数式和面向对象编程风格。
* 弱类型面向对象语言。
* JavaScript程序是用Unicode字符集编写的。

# 2 ECMAScript

ECMAScript描述了语法、类型、语句、关键字、保留字、运算符

## 2.1词法结构

### 2.1.1 语法

* **区分大小写**：变量、函数名、运算符都是区分大小写的。
* **变量是弱类型**：ECMAScript变量无特定的类型，定义变量时只用var运算符，可将它初始化为任意值。
* **每行结尾的分号可有可无**：ECMAScript允许开发者自行决定是否以分号结束一行代码。根据好的编码习惯，最好都加上分号（有些浏览器就不能正确运行）。
* **注释**：与JAVA相同。

// 单行注释

/\*

\*多行注释 \*/

* 大括号表示代码块：

### 2.1.2 变量

ECMAScript中的变量是使用var运算符加变量名定义的。

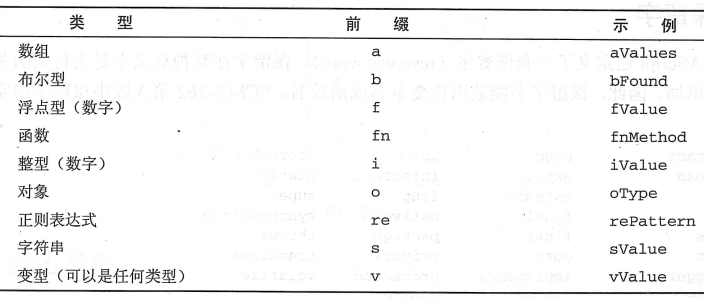
例如：var sTest = “hello world!”;

使用变量时，好的编程习惯是始终存放相同类型的值。变量名遵循以下规则：

* 第一个字符必须是字母、下划线或美元符号($)
* 余下的字符可以是下划线、字母、数字或美元符号
* 首字母小写，接下来的单词都以大写字母开头
* 匈牙利类型标志法：在变量前附加一个小写字母，说明该变量的类型。

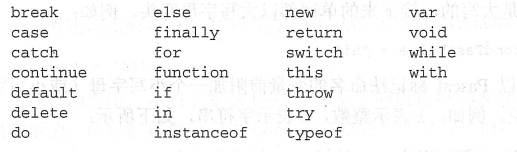
例如：i表示整数，s表示字符串

如下图为区别变量类型的前缀：



### 2.1.3 关键字

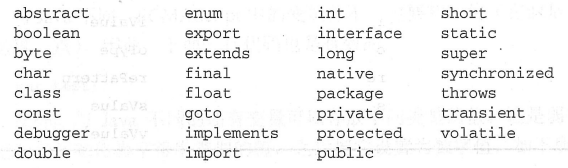
ECMA-262规范定义了ECMAScript支持的一套关键字，关键字是保留的，不能作为变量名或函数名。下面是ECMAScript关键字的完整列表:



如果将关键字用作变量名或函数名，将可能会得到“Identitier expected”这样的错误消息。

### 2.1.4 保留字

ECMAScript定义了一套保留字(reserved word)，为将来关键字而保留的单词。不能作为变量名或函数名。



## 2.2 数据类型和值

### 2.2.1 原始值和引用值

ECMAScript中的变量可存放两种类型的值，原始值和引用值。

* 原始值：是存在栈中的简单数据段，是值直接存储在变量访问的位置。
* 引用值：是存在堆中的对象，存在变量处的值是一个指针(point)，指向存储对象的内存处。

### 2.2.2 原始类型

ECMAScript的原始类型为Undefined、Null、String、Boolean。*注意：其它语言将String设为引用类型，这点ECMAScript打破了这一传统*

**typeof运算符**

使用typeof运算符来判断一个值是否为某种类型，及具体类型。typeof 运算符有一个参数，即要检查的变量或值。

var sTemp = “hello world”;

alert(typeof sTemp); //输出：”string”

alert(typeof 95); //输出：”number”

* “undefined”，如果变量是Undefined型。
* “boolean”，如果变量是Boolean型。
* “number”，如果变量是Number型。
* “string”，如果变量是String型。
* “object”，如果变量是一种引用类型或Null类型。

**Undefined类型**

Undefined类型只有一个值，即undefined。当声明的变量未初始化时，该变量的默认值undefined。

var sTemp;

alert(sTemp); //输出：undefined

**Null类型**

Null类型有一个专用值null，值undefined实际上是从值null派生出来的，因此ECMAScript将它们定义为相等。null值表示尚未存在的对象

alert(null == undefined); // true

**Boolean类型**

有两个值，即true和false。

**Number类型**

Number类型可以表示32位的整数，还可以表示64位的浮点数。八进制字面首数字必须是0，十六进制必须是以0x开始。

**尽管所有的整数都可以表示为八进制或十六进制的字面量，但是所有数字运算返回的都是十进制结果。**

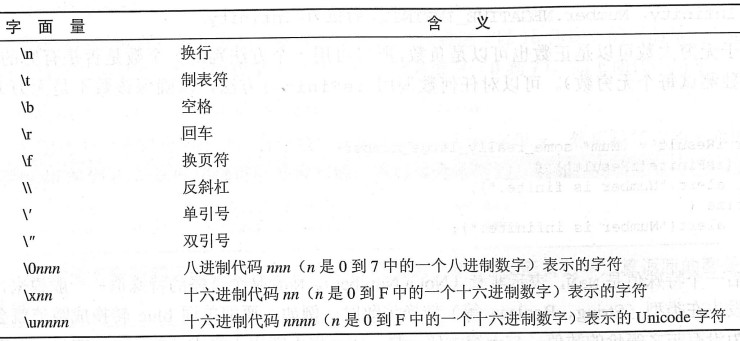
无穷大可以是整数或是负数，可以对任何数调用isFinit()方法判断，确保该数不是无穷大。

IsNaN();判断是否为数字

**String类型**

String类型是唯一没有固定大小的原始类型，首字符从位置0开始，第二个字符在位置1，以此类推。

字符字面量列表及含义如下：



### 2.2.3 引用类型

引用类型通常叫做类(Class)，即引用值就是对象。对象由new 运算符加上要实例的类名创建的。如：var obj = new Object();

**Object类**

Object类本身用处不大，ECMAScript中所有的类都由这个类继承而来，理解Object类有助于理解其他类。Object类具有以下属性：

* Constructor：对创建对象的函数的引用。对于Object类，该指针指向原始的object()函数。
* Prototype：对该对象的对象原型引用。

Object类具有几个方法：

* HasOwnProperty(property):判断对象是否有某个特性的属性。
* IsPrototypeOf(object):判断该对象是否为另一个对象的原型。

等等…

## 2.3 运算符

对值执行的原始动作在任何时候都被看作运算符。

### 2.3.1 一元运算符

**delete**

delete运算符删除对以前定义的对象属性或方法的引用。

例如： var o = new Object(); o.name = “hello word”;

alert(o.name);//输出：hello world

delete o.name; alert(o.name);//输出：undefined

delete运算符不能删除开发者未定义的属性和方法。

**void**

void运算符对任何值都返回undefined。通常用于避免输出不该输出的值。

**前增量/前减量运算符**

在使用前缀式运算符时，增量和减量都发生在计算表达式之前。

var iNum1 = 2, iNum2 =20;

var iNum3 = --Num1 + ++iNum2 ; //等于22

**后增量/后减量**

后缀式运算符是在计算过包含它们的表达式后才进行增量或减量运算。

var iNum1 = 3, iNum2 =21;

var iNum3 = Num1-- + iNum2++ ; //等于24

**new**

new运算符用来创建一个新的对象，并调用构造函数初始化。语法如下：

*new constructor(arguments);*

函数调用运算符

在JavaScript中()用于调用函数，()是一个特殊的运算符，没有固定数目的运算数。

### 2.3.2 位运算符

位运算符是在数字底层(表示数字的32个数位)进行操作的。

## 2.4 函数

函数是一组可以随时随地运行的语句，是ECMAScript的核心。函数的基本语法如下：

function name(arg0, arg1 …, argN) {

statements;

}

JavaScript函数不必声明返回值，如没有返回值等同于void，如有返回值该函数只需使用return运算符后跟要返回的值。没有返回值可调用return用于退出函数。

**如果函数无明确的返回值，或调用了没有参数的return语句，那么真正返回值是undefined**。

* JavaScript语言不支持重载

### 2.4.1 Function类

Function类可以表示开发者定义的任何函数。利用Function类直接创建函数的语法：

var function\_name = new Function(arg1,arg2,argN,function\_body);

**尽管可用Funtion构造函数创建函数，但最好不要使用它，因为它定义函数比用传统的方式要慢的多。不过，所有函数都应看作是Function类的实例。**

### 2.4.2 闭包

闭包，是指词法表示包括不必计算的变量的函数。即是说，该函数能使用函数外定义的变量。在ECMAScript中使用全局变量是一个简单的闭包实例。

var iBaseNum;

function addNum(iNum1, iNum2){

function doAddition(){

return iNum1 + iNum2 + iBaseNum;

}

return doAddition();

}

## 2.5 定义类或对象

使用预定义对象的能力只是面向对象语言能力的一部分，可以用以下方法实现创建自己专有的类和对象。使用最广泛的的是混合的构造函数\原型方式。

### 2.5.1 工厂方式

JavaScript的对象属性可在对象创建后动态定义，可使用工厂函数(factory function)。如下：

function showColor() {

alert(this.color);

}

function createCar(sColor, iDoors, iMpg) {

var oTmpCar = new Object();

oTmpCar.color = sColor;

oTmpCar.doors = iDoors;

oTmpCar.mgp = iMpg;

oTmpCar.showColor = showColor;//通过属性指向该方法

}

### 2.5.2 构造函数方式

在构造函数内部无创建对象，而是使用this关键字。

function Car(sColor, iDoors, iMpg) {

var oTmpCar = new Object();

oTmpCar.color = sColor;

oTmpCar.doors = iDoors;

oTmpCar.mgp = iMpg;

oTmpCar.showColor = function () {

alert(this.color);

};

}

### 2.5.3 原型方式

利用对象的prototype属性，把它看成创建新对象所依赖的原型。用空构造函数代替类名。示例如下：

function Car() {

}

Car.prototype.color = “red”;

Car.prototype.doors= 4;

Car.prototype.mpg = 23;

Car.prototype.showColor = function() {

alert(this.color);

};

var car = new Car();

缺点：这个构造函数没有参数，不能通过构造函数传递参数初始化属性的值。

### 2.5.4 混合的构造函数\原型方式

联合使用构造函数和原型方式，在构造函数内定义非函数属性，而函数属性利用原型属性定义。

function Car(sColor, iDoors, iMpg) {

var oTmpCar = new Object();

oTmpCar.color = sColor;

oTmpCar.doors = iDoors;

oTmpCar.mgp = iMpg;

this.drivers = new Array(“one”,”two”);

}

Car.prototype.showColor() {

alert(this.color);

}

var car = new Car(“red”, 4, 23);

car.drivers.push(“three”);

### 2.5.5 动态原型方法

动态原型方法的基本思想与混合的构造函数\原型方式相同，即在构造函数内定义非函数属性，而函数属性利用原型属性定义。唯一区别是赋予对象方法的位置。示例如下：

function Car(sColor, iDoors, iMpg) {

var oTmpCar = new Object();

oTmpCar.color = sColor;

oTmpCar.doors = iDoors;

oTmpCar.mgp = iMpg;

this.drivers = new Array(“one”,”two”);

if(typeof Car.\_init == “undefined”) {

Car.prototype.showColor = function() {

alert(this.color);

}

Car.\_init = true;

}

}

第一次使用运算符typeof检测\_init为undefined，创建showColor方法后Car.\_init置为true，就不再创建showColor方法了。保证showColor只创建并赋值一次。

## 2.6 继承

用ECMAScript实现继承机制，首先从基类入手。所有开发者定义的类都可以作为基类。出于安全原因考虑，本地类和宿主类不能作为基类，这样可以防止公用访问编译过的浏览器的代码，因为这些代码可以被恶意攻击。

### 2.6.1 继承方式

* **对象冒充**

构造函数使用this关键字给所有的属性和方法赋值(即采用类声明的构造函数方式)。

* **call()方法**

call()方法是与经典的对象冒充方法最相似的地方。它的第一个参数用作this的对象。其他参数都直接传给函数自身。示例：

function sayColor(sPrefix, sSuffix) {

alert(sPrefix + this.color + sSuffix);

};

var obj = new Object();

obj.color = “red”;

//输出：the color is red, a very nice color

sayColor.call(obj,”The color is”, “, a very nice color”);

* **apply()方法**

apply()方法有两个参数，用作this的对象和要传递给函数的数组。示例：

function sayColor(sPrefix, sSuffix) {

alert(sPrefix + this.color + sSuffix);

};

var obj = new Object();

obj.color = “red”;

//输出：the color is red, a very nice color

sayColor. apply (obj, new Array(”The color is”, “, a very nice color”));

* **原型链**

继承这种形式在ECMAScript中原本是用于原型链的。prototype对象的属性和方法都被传递给那个类的所有实例。原型链利用这种功能实现继承机制。

function ClassA {

}

ClassA.prototype.color = “red”;

ClassA.prototype.sayColor = function() {

alert(this.color);

};

Function ClassB() {

}

ClassB.prototype = new ClassA();//通过原型实现继承

ClassB.prototype.name = “class b”;//需在ClassB.prototype = new ClassA();之后出现

**注意：调用ClassA的构造函数时，没有给它传递参数。这在原型中是标准做法，要确保构造函数没有任何参数。**

子类的所有属性和方法都必须出现在prototype属性被赋值以后，因为在它之前赋值的所有方法都会被删除。因为prototype属性被替换成了新对象，添加新的方法的原始对象将被销毁。

此外，instanceof 运算符的运行方式也很独特。

Var objB = new ClassB();

alert(objB instanceof ClassA);// true

alert(objB instanceof ClassB);// true

原型链缺点是不支持多重继承。

# 3 DOM

DOM是针对XML的基于树的API。使用DOM，解析一次来创建一个树的模型，DOM是与语言无关的API（可以是JAVA、JavaScript实现）。

Node接口定义了借点类型包含的属性和方法，如下表：

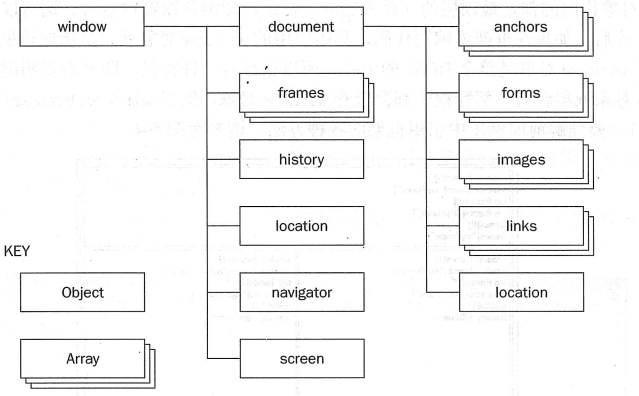


除IE浏览器外，其他浏览器都实现了标准的DOM。

# 4 BOM

Web浏览器的主要任务是在一个窗口中显示HTML文档。表示HTML文档的是Document对象，Window对象表示该文档的窗口(或框架)。Window对象是客户端程序设计中的全局对象。

BOM即是浏览器对象模型，提供了独立于内容而与浏览器窗口进行交互的对象。如下为BOM体系结构。



window对象是整个BOM的核心，所有对象和集合都以某种形式回接到window对象。

## 4.1 window对象

window对象表示整个浏览器窗口，但不必表示其中包含的内容。开发者可以移动或调整浏览器窗口的大小，如下方法实现这些操作：

* moveBy(dx, dy)：把浏览器窗口相对当前位置水平移动dx个像素，垂直移动dy个像素。dx为负数向左移动，dy为负数，向上移动。
* moveTo(x, y)：移动浏览器窗口，使它的左上角位于用户屏幕(x,y)处。可以使用负数，这样会把部分窗口移出屏幕的可视区域。
* resizeBy(dw, dh)：相对浏览器的窗口的当前大小，把它的窗口调整dw个像素，高调整为dh个像素。dw, dh为负数，则相应缩小。
* resizeTo(w, h)：把窗口的宽度调整为w，高度调整为h。h,w不能为负数。

**导航和打开新的窗口**

用windo.open()方法可打开新窗口。参考API

**系统对话框**

向用户弹出信息alert()、confirm()、prompt()

**状态栏**

状态栏是底部边界区域，用于向用户显示信息。

**时间间隔和暂停**

可使用window对象的setTimeout()方法

**历史**

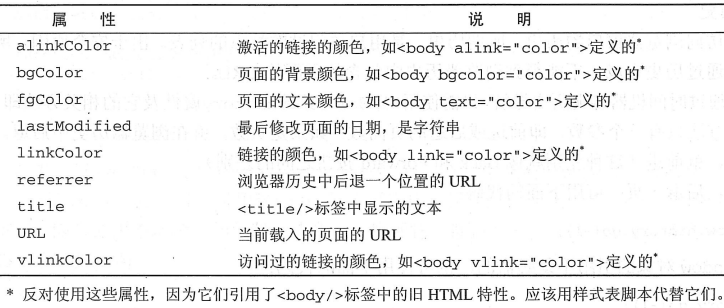
可以访问浏览器窗口的历史，即用户访问过的站点列表。出于安全原因，所有导航只能通过历史完成，不能得到浏览器历史中包含的页面的URL。

window.history.go(-1);//后退

window.history.back(1);//前进

## 4.2 document对象

document实际上是window对象的属性，这个对象的独特之处是它唯一一个既属于BOM又属于DOM的对象。从BOM的角度看来，document对象由一系列集合而成，可以访问文档的各个部分。下表列出了BOM的document对象的通用属性：



## 4.3 location对象

location对象是window对象和document对象的属性。 具体查看API

## 4.4 检测浏览器

# 5. JavaScript设计模式

## 5.1 JavaScript接口

JavaScript本身没有提供接口的定义，通常只能模仿的形式实现。JavaScript中有三种方法模仿接口：注释法、属性检查法和鸭式辨型法

### 5.1.1注释描述接口

用注释法模仿接口是最简单的方法，但效果却是最差的。模仿其他面向对象中的做法，使用了Interface 和 implements关键字，但把它们放在注释中，以免引起语法错误。示例如下：

/\*

Interface Composite {

function add(child);

function remove(child);

}

Interface FormItem{

function save();

}

\*/

var CompositeForm = function (id, method, action) {//实现Composite,FormItem接口

}

//实现Composite接口

CompositeForm.prototype.add = function(child) {

….

};

CompositeForm .prototype.remove = function(child){

….

};

//实现 FormItem 接口

CompositeForm.prototype.save = function {

…

};

这种模仿并不是很好，它没有确保CompositeForm 真正实现了正确的方法集而进行检查，也不会抛出错误以告知程序员程序中有问题。这种做法对接口的约定完全靠自觉。

优点是易于实现，不需要额外的函数或类，可以提高代码的重用性。

### 5.1.2 用属性检查模仿接口

## 5.2 JavaScript对象

JavaScript创建对象的基本模式有3种：门户大开型

# 6正则表达式

## 6.1 正则表达式支持

JavaScript对正则表达式的支持通过ECMAScript中的RegExp类实现。详细参考API。也可以使用perl风格的语法。如：

var reCat = /cat/gi;

**元字符**

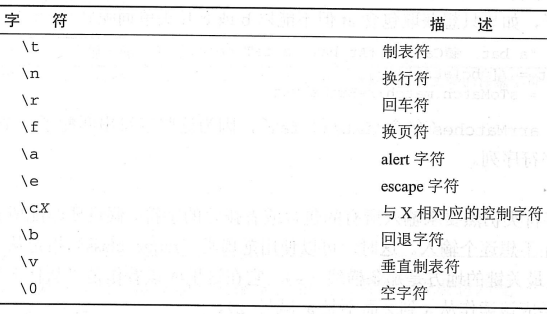
元字符包括：( [ { \ ^ $ | ) ? \* + .

在正则表达式中使用元字符，都必须对它转义。例如，匹配一个问号，正则如下表示：

var reQMark = /\?/; 或者 var reQMark = new RegExp(“\\?”);

注意：使用RegExp需使用双重转义，即\\。

**特殊字符**



## 6.2 简单模式

### 6.2.1字符类

字符类是用于测试字符的组合。通过将一些字符放入方括号中（[]），可以有效地告诉正则表达式去逐个匹配方括号中的字符。

**简单类**

要匹配字符a、b、c，字符类应该是[abc]称为简单类。示例如下：

var reMatch = /[abc]at/gi; //匹配aat,bat,cat并且不区分大小写

**负向类**

除了某些特定的一些，想要匹配所有的字符。这时可以使用负向类，可以指定要排除的字符。

var reMatch = /[^ab]t/gi; //匹配包含t但不能以a、b开头的字符序列

**范围类**

使用范围类指定从a到z之间的范围：[a-z]。横线“-”：表示从什么到什么。

**组合类**

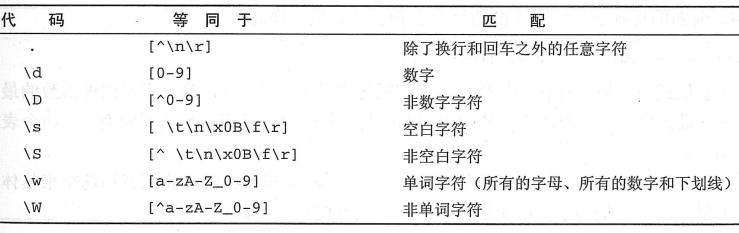
由几种其他的类组合而成的字符类。例如：匹配所有从a-m的字母及1-4的数字，以及一个换行符。

[a-m1-4\n]

注意：内部的类之间不能有空格。ECMAScript不支持联合类和交叉类，例如类似[a-m[p-z]]或者[a-m[^b-e]]。

**预定义类**

某些模式会反复用到，可以使用一组预定义字符类以更方便地指定复杂类。下表列出预定义的类。

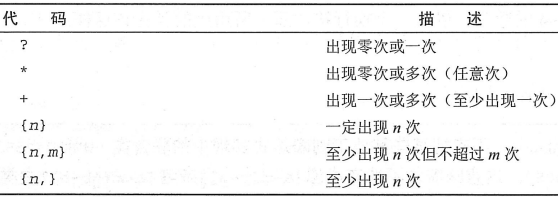


### 6.2.2 量词

量词可以指定某个特定模式出现的次数。

**简单的量词**

下表列出指定特定模式数量的几种方法：



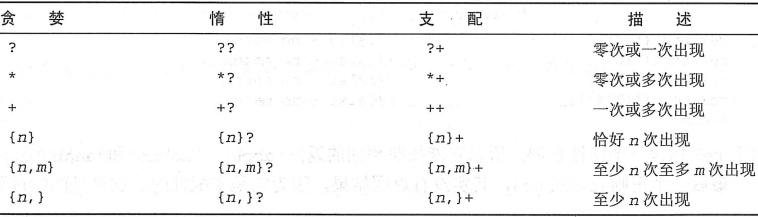
例如：

/.+/ 除换行和回车之外的任意字符至少出现一次

**贪婪的、惰性的和支配性的量词**

* 贪婪的：先看整个的字符串是否匹配。如果没有发现匹配，去掉去掉该字符串中的最后一个字符，并再次尝试。如果还是没有发现匹配，那么再次去掉最后一个字符，这个过程会一直重复直到发现一个匹配或字符串不剩任何字符串。
* 惰性的：先看字符串第一个字母是否匹配。如果单独这一个字符还不够，就读入下一个字符，组成两个字符的字符串。如果还没有匹配，继续从字符串中添加字符直到发现匹配或者整个字符串都检查过没有匹配。惰性量词和贪婪量词工作方式刚好相反。
* 支配性：只尝试匹配整个字符串。如果整个字符串不能产生匹配，不作进一步尝试。支配量词就是一刀切。

下面列出了贪婪的、惰性的和支配性的量词



*注意：浏览器对支配量词的支持还很不完善。IE和Opera不支持支配量词，如果用它，就会抛出一个错误。*

## 6.3 复杂模式

**分组**

分组是通过用一系列括号包围一系列字符、字符类以及量词来使用的。

var reSpace = /^\s+(.\*?)\s+$/; 去掉字符串前后的空格，等同trim()

String.prototype.trim = function() {

var reSpace = /^\s+(.\*?)\s+$/;

return this.replace(reSpace, “$1”);

}

**反向引用**

反向引用是按照从左到右遇到的左括号字符的顺序进行创建和编号。

**候选**

候选操作符和ECMAScript的二进制异或一样，是一个管道符(|)，把它放在两个单独的模式之间。如下例子：

var reRedOrBlack = /(red|black)/; 匹配red或black

**非捕获性分组**

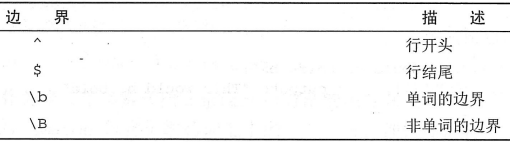
创建反向引用分组，称之为非捕获性分组。在较长的正则表达式中，存储反向引用会降低匹配速度。通过非捕获性分组可提高效率。创建非捕获性分组只要在左括号的后面加上一个问号。如下：

var reNumbers = /#(?:\d+)/;

**前瞻**

**边界**

边界用于正则表达式中模式的位置。下表是几种可能的边界



var reQQ = /^[1-9]\d{4,11}$/; 验证是否合法QQ，首位数字是1-9，总长度在5-12之间

## 6.4 RegExp对象

JavaScript中每一个正则表达式都是一个对象。

### 6.4.1 实例属性

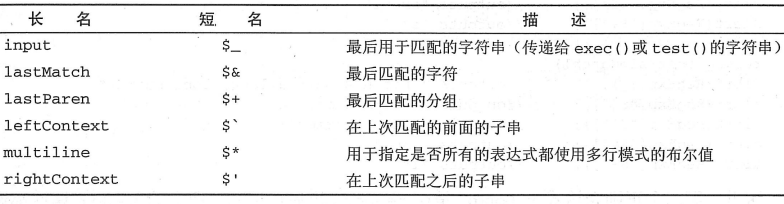
RegExp的实例属性如下：

* global：Boolean值，表示g(全局选项)是否已设置。
* ignoreCase：Boolean值，表示i(忽略大小写选项)是否已设置。
* lastIndex：整数，代表下次匹配将会从哪个字符位置开始(只有当使用exec()或test()函数才会填入，否则为0)。
* multiline：Boolean值，表示m(多行模式选项)是否已设置。
* source：正则表达式的源字符串形式。

lastIndex属性告诉你正则表达式在某个 字符串中停止了，或者匹配到哪个位置。

### 6.4.2 静态属性

静态的RegExp属性对所有的正则表达式都有效。如下表：



# 7 事件

## 7.1 事件流

IE和其他浏览器的事件流实现不一样，主要体现在事件发生的顺序上。

**冒泡事件**

IE上实现的事件流是基于冒泡型的，事件按照从最特定的事件目标到最不特定的事件目标的顺序触发。

**捕获型事件**

其他浏览器使用的捕获型事件，捕获型事件中从最不精确的对象开始触发，然后到精确。

**DOM事件流**

DOM同时支持冒泡和捕获事件，捕获型事件先发生。两种事件流会触及DOM所有对象

## 7.2 事件的处理及监听

用于响应某个事件而调用的的函数称为事件处理函数(event handler)，DOM称之为事件监听函数(event listener)

在JavaScript中分配事件处理函数，需要首先获得要处理的对象的引用，然后将函数赋值给对应的事件处理函数属性。例如：

var oDiv = document.getElementById(“div1”);

oDiv.onclick = function() { //这种方法事件处理函数名称需用小写。

// logic code

}

HTML:

<div onclick=”alert(“”);</div> //用HTML方法事件处理函数的可使用大小写

注意：如果使用XHTML代码，事件处理函数应该全部使用小写定义。

### 7.2.1 IE事件函数

IE中每个元素和window对象都有两个方法：

attachEvent()：用来给一个事件附加事件处理函数

detachEvent()：用来将事件处理函数分离出来。

### 7.2.2 DOM

DOM方法addEventListener()和removeEventListener()用来分配和移除事件处理函数。语法如下：

[Object].addEventListener(“name of event”, fnHandler, bCapture);

[Object].removeEventListener(“name of event”, fnHandler, bCapture);

三个参数：事件名称，分配的处理函数，用于冒泡阶段还是捕获阶段(true:冒泡阶段)

var fnClick = function() {

alert(“”);

};

var oDiv = document.getElementById(“div1”);

oDiv.addEventListener(“click”,fnClick, false);//可以附加多个函数

使用传统方法直接给事件处理函数属性赋值，事件处理函数将被添加到事件的冒泡阶段。例如：

oDiv.onclick = fnClick; //与上面效果是一样的

注意：直接赋值不支持多个函数，后续的赋值将会替代前面的赋值。

## 7.3 事件对象

事件对象通常是指刚刚发生的事件信息的对象，包含的信息如下：

* 引起事件的对象；
* 事件发生时鼠标的信息；
* 事件发生时键盘的信息。

事件对象只在发生事件时才被创建，且只有事件处理函数才能访问。所有事件处理函数执行完毕后，事件对象就被销毁。

### 7.3.1 定位

**IE事件对象定位**

在IE中，事件对象就是window对象的一个属性event。必须这样访问事件对象：

oDiv.onclick = function () {

var oEvent = window.event;

}

尽管它是window对象的属性，event对象还是只能在事件发生时访问。所有事件执行完毕后，事件对象就会被销毁。

**DOM标准**

DOM标准中event对象必须作为唯一的参数传给事件处理函数。两种方式获取事件对象，如下：

oDiv.onclick = function () { //第一种

var oEvent = arguments[0];

};

可以直接命名参数，第二种如下：

oDiv.onclick = function (oEvent) {

};

## 7.4 事件类型

DOM定义了标准的事件类型，分为以下几类：

* 鼠标事件：用户使用鼠标进行特定操作时触发；
* 键盘事件：用户在键盘上敲击、输入时触发；
* HTML事件：窗口发生变动或者发生特定的客户端 — 服务器端交互时触发
* 变化事件：底层的DOM结构发生改变时触发。

### 7.4.1 鼠标事件

页面上所有的元素都支持鼠标事件。

* click：用户点击鼠标左键时发生。当用户的焦点在按钮上，并按了回车键同样也会触发。
* dbclick：用户双击鼠标的左键时发生。
* mousedown：用户点击任意一个鼠标按钮时发生。
* mouseout：鼠标指针在某个元素上，且用户要移出元素的边界是发生。
* mouseover：鼠标移出某个元素，到另外一个元素上时发生。
* mouseup：用户松开任意一个按钮时发生。
* mousemove：鼠标在某个元素上时持续发生。

### 7.4.2 键盘事件

* keydown：用户在键盘上按下某按键时发生。
* keypress：用户按下一个键，并产生一个字符时发生。一直按下按键时，则会持续发生。
* keyup：用户释放按着的按键时发生。

### 7.4.3 HTML事件

* load：页面完全载人后，在window对象上触发；所有框架都载入后，在框架集上触发；<img/>完全载入后，在其上触发；或者对于<object/>元素，当其完全载入后在其上触发。
* unload：页面完全卸载后，在window对象上触发；所有框架都卸载后，在框架集触发；<img/>完全卸载后，在其上触发；或者对于<object/>元素，当其完全卸载后在其上触发。
* abort：用户停止下载过程时
* error：javascript脚本出错时，在window对象上触发。
* select：用户选择了文本框中的一个或多个字符时触发
* change：文本框失去焦点时并且在它获得焦点后内容后内容发生过改变时触发。
* submit：点击提交按钮，在<form/>上触发。
* reset：点击重置按钮，在<form/>上触发。
* scroll：用户在任何带滚动条的元素上卷动它是时触发。<body/>元素包含载入页面的滚动条。
* focus：任何元素或者窗口本身获取焦点时触发。
* blur：任何元素或窗口本身失去焦点时触发。

### 7.4.4变化事件

虽然变化事件是DOM的一部分，目前还有任何主流的浏览器实现了它。

8工具

IDE

调试、排错