



杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层
邮编: 215311
电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区
邮编: 200436
电话: 021-56778147

Briup High-End IT Training

第二阶段

脚本语言JavaScript

Brighten Your Way And Raise You Up.



杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

JavaScript

Brighten Your Way And Raise You Up.

学习目标

- ◆ 掌握JavaScript基本语法、注释、标识符、关键字、保留字、数据类型
- ◆ 掌握数据类型之间的转换、循环语句，分支语句
- ◆ 掌握了解对象，函数，原型
- ◆ 熟练掌握数组、函数、对象、闭包
- ◆ 熟练掌握面向对象思想、DOM、BOM
- ◆ 了解使用正则表达式



章节简介

- ◆ 第1章: JavaScript快速入门
- ◆ 第2章: 操作符及类型转换
- ◆ 第3章: 流程控制语句
- ◆ 第4章: 对象及函数
- ◆ 第5章: 数组
- ◆ 第6章: 正则表达式
- ◆ 第7章: 内置对象及内置函数
- ◆ 第8章: 面向对象的程序设计
- ◆ 第9章: 文档对象模型
- ◆ 第10章: 事件
- ◆ 第11章: 浏览器对象模型





杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

第1章: JavaScript快速入门

Brighten Your Way And Raise You Up.

学习目标

- ◆ 编写第一个**JavaScript**脚本 “Hello World”
- ◆ 掌握**JavaScript**的语法规则
- ◆ 掌握**JavaScript**中的数据类型



JavaScript简介

JavaScript诞生于1995年，当时的主要目的是处理由以前服务器语言负责的一些没有填写的必填域，是否输入了无效的值。在web日益流行的同时，人们对客户端脚本语言的需求也越来越强烈，那时绝大多数因特网用户使用的速度仅为28.8kbit/s的猫上网，但网页的大小和复杂性却不断增加，未完成简单的表单验证而与服务器交换数据只会加重用户和服务器的负担。

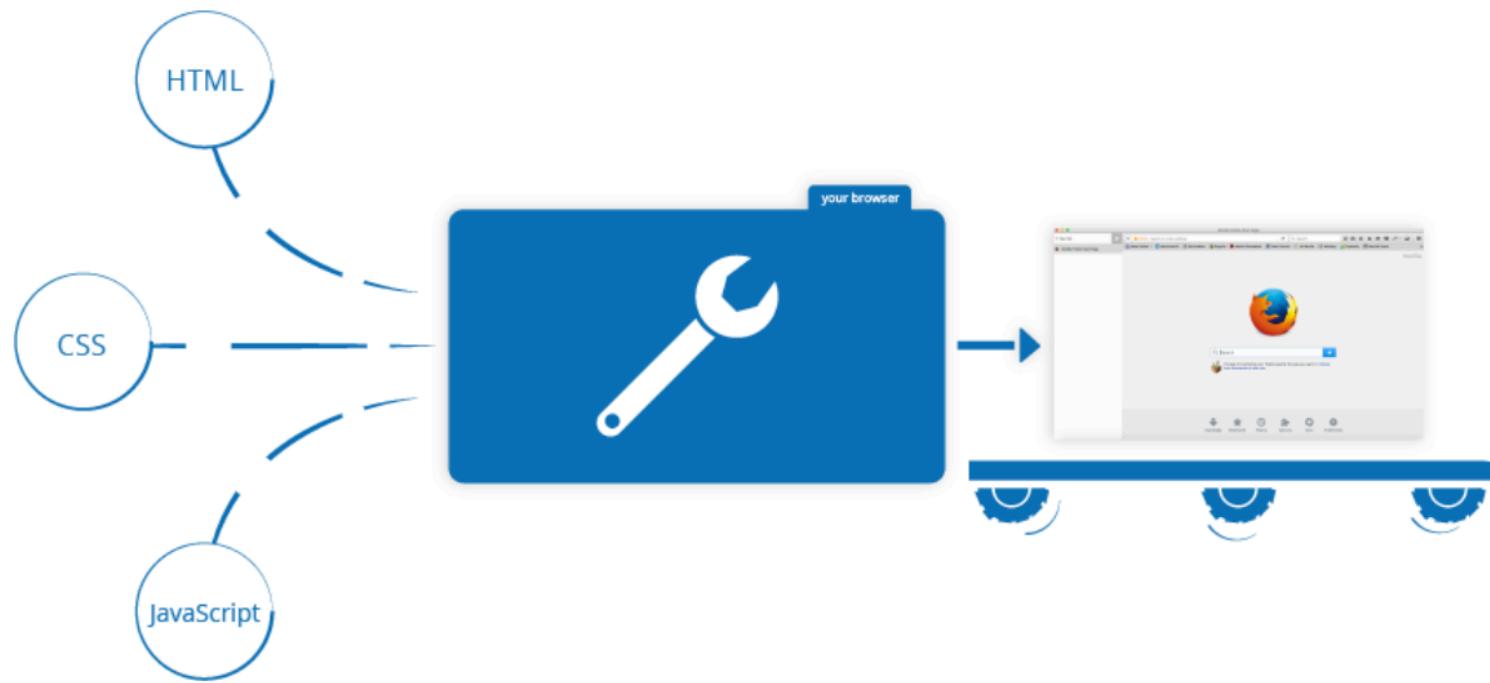
1995年2月 计划在Netscape Navigator2开发名为LiveScript的脚本语言，同时在浏览器和服务器中使用，为了赶在发布日期前完成LiveScript开发，Netscape和sun公司建立了一个开发联盟，在Netscape Navigator2发布的前夕，为了搭上媒体上热炒的java顺风车，临时把LiveScript改名为javaScript。在Navigator3发布不久，ie3就加入了名为JScript的javaScript的实现。这意味着有两个不同的javascript版本：javascript,jscript.当时并没有标准规定JavaScript的语法和特性。

1997年，JavaScript1.1 为蓝本的建议被提交给了ECMA（European Computer Manufacturers Association欧洲计算机制造商协会）。定义了ECMAScript新脚本语言的标准(ECMA-262)。第二年，ISO/IEC（International Organization for Standardization and International Electrotechnical Commission,国标标准化组织和国际电工委员会）也采用了ECMAScript作为标准（ISO/IEC-16262），自此浏览器开发商就致力于将ECMAScript作为各自JavaScript实现的基础。



◆ WHAT IS IT

JavaScript是一个编程语言，允许用户在浏览器页面上完成复杂的事情。浏览器页面并不总是静态的，往往显示一些需要动态更新的内容，交互式地图，动画，以及视频等。一个完整的Javascript包括核心(ECMAScript5)，应用程序编程接口即API(比如DOM(Document Object Model)，BOM(Browser Object Model))，以及其他第三方API。JavaScript与HTML、CSS一同配合共同完成一个复杂页面的显示。



JavaScript简介

◆ 特点

- 客户端代码，在客户机上执行

JavaScript特殊的地方在于它也可以作为服务器端代码执行，但是需要搭建Node环境

- 解释性语言

被内置于浏览器中的JS解析器解析执行，执行前无需编译

- 弱类型语言

- 从上往下顺序解析执行



JavaScript简介

◆ Hello World

● 在网页中使用JavaScript

✓ 内部JavaScript

编写好HTML，在`<head>`标签体中添加`<script>`元素，然后将JS代码填写进来即可。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Helloworld</title>
    <style type="text/css">
        body {
            margin: 0;
            padding: 0;
        }
    </style>
    <script type="text/javascript">
        var str = "hello world";
        console.log(str);
    </script>
</head>
<body>

</body>
</html>
```



JavaScript简介

◆ Hello World

● 在网页中使用JavaScript

✓ 外部JavaScript

单独新建一个后缀名为.js的JS文件，编写好HTML文件，在<head>标签体内添加<script>元素，使用script标签的src属性将JS文件导入进来。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Helloworld</title>
    <style type="text/css">
        body {
            margin: 0;
            padding: 0;
        }
    </style>
    <script type="text/javascript" src="js/hello.js"></script>
</head>
<body>

</body>
</html>
```



JavaScript简介

◆ 注释

与绝大多数语言类似，JavaScript也需要注释来说明其代码含义，或者用来进行代码调试，注释后的代码会被浏览器忽略不被执行。

- 单行注释

```
// I am a comment
```

- 多行注释

```
/*
I am also
a comment
*/
```



关键字&保留字

● 关键字:(在JS中有特殊功能)

| | | | |
|----------|---------|----------|------------|
| break | do | try | typeof |
| case | else | new | var |
| catch | finally | return | void |
| continue | for | switch | while |
| debugger | this | function | with |
| default | if | throw | instanceof |
| delete | in | | |

● 保留字: (将来可能成为关键字)

| | | | |
|----------|---------|------------|--------------|
| abstract | enum | int | short |
| boolean | export | interface | static |
| byte | extends | long | super |
| char | final | native | synchronized |
| class | float | package | throws |
| const | goto | private | transient |
| debugger | double | implements | protected |
| volatile | import | public | |



变量

变量是一个值的容器，该容器的值可以随时改变。ECMAScript的变量是弱类型（松散类型），可以用来保存任何类型的数据。定义变量时使用var关键字。

● 变量的使用

✓ 声明

```
var message;
```

✓ 初始化

```
message = "hello"
```

✓ 声明并初始化

```
var message = "hello";
```

✓ 定义多个变量

```
var message= "hello" ,found=false, age = 29;
```

● 变量名的命名规则

✓ 变量名由字母，数字，下划线以及\$组成。

✓ 不要使用下划线或者数字作为变量名的开头

✓ 变量名应该具有一定的意义，使用小驼峰命名规则

✓ 不要使用关键字或是保留字



数据类型

◆ 五种基本数据类型

✓ Undefined

未定义类型 undefined

```
var a ; var a = undefined;
```

✓ Null

空引用数据类型 null

```
var a = null;
```

✓ Boolean

布尔类型，取值为 true/false，通常用于条件判断

```
var a = false;
```

✓ Number

数字类型。整数/浮点数

✓ String

字符串类型，需要使用单引号或者双引号括起来

```
var a ="true";
```

```
var a ='hello';
```



数据类型

◆ 引用数据类型

在JS中除了以上基本数据类型，其他所有类型都可以归结为引用数据类型。

● 对象

对象是模拟现实生活的对象，对象由键值对组成，通过使用大括号将所有键值对括起来。

```
var dog = { name : 'Spot', breed : 'Dalmatian' };
```

● 数组

数组是一个特殊的对象，包含了多个值，值与值之间使用逗号分隔开，所有的值通过中括号括起来。

```
var myNameArray = ['Chris', 'Bob', 'Jim'];
```

```
var myNumberArray = [10,15,40];
```

● 函数

函数是代码执行单元，用于实现某些特殊的功能。

```
function add(a,b){  
    return a + b;  
}
```



◆ Number

数字有很多类型，按照数字精度可以分为整数(int),单精度(float),双精度(double)，按照数字的表示方法可以分为二进制(Binary)，八进制(Octal)，十进制(decimal system)，十六进制(Hexadecimal)。但是在JS中，所有的数字统一使用Number来表示。

● 表示方法

✓ 整数

十进制 55 由0~9组成

八进制 070 首位为0，其他位有0~7组成

十六进制 0x11 首位为0x，其他位为0~9，A~F

✓ 浮点数

所谓浮点数值，就是该数值中必须包含一个小数点，并且小数点后必须至少有一位数字。
浮点数值的最高精度是17位小数

普通浮点数 3.1415926

科学计数法 3.125e7 即31250000



数据类型

◆ Number

● 非数值

该数值表示一个本来要返回数值的操作数未返回数据的情况

```
var a = 10/'a'; // a为NaN
```

● 非数值检测

判断参数是否“不是数值”，当参数para不是数值的时候返回true

```
isNaN(NaN); // true
```



◆ Number

● 数值范围

由于内存的限制，ECMAScript不能保存世界上所有的数值。ECMAScript能表示的最小数值保存在Number.MIN_VALUE中，能表示的最大的数值保存在Number.MAX_VALUE中。如果某次计算的结果超过了JavaScript数值范围，将会返回Infinity(正无极)或者-Infinity(负无极)

```
var a = 9/0;           // Infinity
```

```
Number.MIN_VALUE 5e-324
```

```
Number.MAX_VALUE 1.7976931348623157e+308
```

● 数值范围检测

使用 isFinite()函数可以判断参数是否在最大值和最小值之间，如果在，返回true

```
Var a = isFinite(9/0); // false
```

数据类型

◆ Null

该类型的取值只有一个，即null。null可以表示一个空对象的指针。

```
var a = null;
```

● 使用场景

如果一个变量准备将来保存对象，可以将该变量初始化null而不是其他，这样可以通过检查null值就可以知道相应的变量是否已经保存了一个对象的引用。

```
if(car != null){ //car对象执行某些操作}
```

◆ Undefined

该类型只有一个值undefined。对未声明和未初始化的变量执行typeof操作符都返回undefined。

```
var a;  
console.log(a);          // undefined  
console.log(typeof a);  // undefined
```

◆ null vs undefined

实际上 undefined 派生自null值。undefined == null 结果为 true，null与undefined用途不同，null可以用来表示一个空对象，但是没有必要把一个变量的值显式设置为undefined。



◆ String

该类型表示由零个或者多个16位Unicode字符组成的字符序列，即字符串。字符串可以由双引号或者单引号表示

● 字符字面量

| | | | | | |
|----|-----|-----|----|----|-----|
| \n | 换行 | \t | 制表 | \b | 退格 |
| \r | 回车 | \\" | 斜杠 | ' | 单引号 |
| " | 双引号 | | | | |

● 字符长度

通过length属性获取字符长度





杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

第 2 章：操作符及类型转换

Brighten Your Way And Raise You Up.

学习目标

- ◆ 掌握**JS**中算术运算符，逻辑运算符，一元运算符，三元运算符
- ◆ 掌握**JS**中的类型转换（其他类型到**Number, Boolean, String**的转换）



操作符

◆ 算术运算符

| Operator | Name | Purpose | Example |
|----------|--|--|---|
| + | Addition | Adds two numbers together. | 6 + 9 |
| - | Subtraction | Subtracts the right number from the left. | 20 - 15 |
| * | Multiplication | Multiplies two numbers together. | 3 * 7 |
| / | Division | Divides the left number by the right. | 10 / 5 |
| % | Remainder (sometimes called modulo) | Returns the remainder left over after you've shared the left number out into a number of integer portions equal to the right number. | 8 % 3 (returns 2, as three goes into 8 twice, leaving 2 left over.) |

操作符

- ◆ 一元运算符
- 递增递减操作符

++表示每次递增1，--表示每次递减1。常用于遍历操作，比如要遍历某个数组，求所有值的和，需要将数组中的每个值逐个取出叠加，每次取值的时候都需要将索引递增1。

```
var a = 3;  
a++; //4  
a--; //3
```

- 赋值运算符

单个=表示赋值，将右侧的值赋给左侧的变量。

可以和其他算术运算符连用，常用的有*=, /=, %=, +=, -=

```
var a = 4;  
a+=3; //a = a + 3;
```



操作符

◆ 一元运算符

● 加 +

相当于调用Number()

● 减 -

✓ 将一元减应用于数值时，数值会变成负数。

✓ 将一元减应用于非数值时，遵循与一元加操作符相同的规则，最后将得到的数值转化为负数



操作符

◆ 比较运算符

| Operator | Name | Purpose | Example |
|--------------------|--------------------------|---|-------------------------|
| <code>==</code> | Strict equality | Tests whether the left and right values are identical to one another | <code>5 == 2 + 4</code> |
| <code>!=</code> | Strict-non-equality | Tests whether the left and right values not identical to one another | <code>5 != 2 + 3</code> |
| <code><</code> | Less than | Tests whether the left value is smaller than the right one. | <code>10 < 6</code> |
| <code>></code> | Greater than | Tests whether the left value is greater than the right one. | <code>10 > 20</code> |
| <code><=</code> | Less than or equal to | Tests whether the left value is smaller than or equal to the right one. | <code>3 <= 2</code> |
| <code>>=</code> | Greater than or equal to | Tests whether the left value is greater than or equal to the right one. | <code>5 >= 4</code> |

操作符

◆ 逻辑运算符

● 逻辑与&&(同真才真，有假则假)

可应用于任意数值。如果有一个操作数不是布尔类型，逻辑与就不一定返回boolean类型

- ✓ 如果第一个操作数是null,NaN,undefined,false,0,""可被转换为false的值的时候返回该值
- ✓ 如果第一个数其他，返回第二个数

```
var s1 = 8;  
var s2 = "briup";  
var s3 = "";  
var result = s1 && s2;          //briup  
var result2 = s3 && s2;        //空字符串
```

● 逻辑或||(有真则真，同假才假)

- ✓ 如果两个操作数都是null,NaN,undefined,false,0,""可被转换为false的值的时候返回该值
- ✓ 如果第一个操作数是null,NaN,undefined,false,0,"" 则返回第二个操作数



操作符

◆ 逻辑运算符

● 非 (NOT)

该操作符应用任何类型数值都返回一个【布尔值】。先将任意类型的数值转换为Boolean，然后取反

`!a ==> !Boolean(a)`

`!0` //true

`!""` //true

`!NaN` //true

`!false` //true

连用两次逻辑非，就可以将任意数据类型转化为Boolean类型

`!!a ==> Boolean(a)`

`!!"" //false`



操作符

◆ 三目运算符

```
variable = boolean_expression ? true_value : false_value;
```

如果boolean_expression为true,将true_value赋给variable, 否则将false_value赋给variable

- 例如：求任意两个数之间最大值

```
function max(m,n){  
    return m>n?m:n;    //如果m>n为true返回m,如果m>n为false,返回n  
}
```



类型转换

◆ 其他数据类型转换为**String**

● **toString()**函数

除了**null**, **undefined**, 其他三种基本数据类型的变量均有一个**toString()**函数, 该函数可以获取该变量指定值的字符串表示。

```
var a= true;  
a.toString(); // "true"
```

如果变量为**number**类型, 默认情况下**toString()**是以十进制格式返回数值的字符串表示, 通过传递参数, 可以输入以二进制, 八进制, 十六进制乃至任意有效进制格式的字符串值

```
var num = 10;  
  
num.toString(); // "10"  
  
num.toString(2); // "1010"  
  
num.toString(8); // "12"  
  
num.toString(16); // "a"
```

● **String()**函数

可以将其他任意基本数据类型的值转换为字符串, 包括**null**, **undefined**



类型转换

◆ 其他数据类型转换为Boolean

● Boolean()函数

任意其他数据类型都可以转换为布尔类型。

| Boolean | true | false |
|-----------|-------|-----------|
| String | 非空字符串 | "" |
| Number | 任何非0 | 0/NaN |
| Object | 任何对象 | null |
| Undefined | 不适用 | undefined |

类型转换

◆ 其他数据类型转换为Number

● Number()函数

- ✓ 如果转换的值是null,undefined,boolean,number

```
Number(true);           //1
Number(false);          //0
Number(null);           //0
Number(undefined);      //NaN
Number(10);             //10 如果是数字值,原样输出
```

- ✓ 如果转换的值是string

```
Number("123");         //如果仅包含数值, 转换为对应的数值
Number("234.1");       //解析为对应的小数
Number("+12.1");       //首位为符号位, 其余为为数值, 转换为对应的数值
Number("1+2.3");       //NaN 符号位出现在其他位置, 解析为NaN
Number("0xa");          //如果仅包含十六进制格式, 转为为对应的十进制的值
Number("010");          //【注意!】不会当做八进制被解析, 结果为10
Number("");              //空字符串被转换为0
Number("123ac");        //包含其他字符:  NaN
Number(" 12");           //12
```



类型转换

◆ 其他数据类型转换为Number

● parseInt()函数

- ✓ 如果转换的值是null,undefined,boolean， 均转换为NaN

- ✓ 如果转换的值是Number

```
parseInt(10);           //10 如果是整数值，原样输出  
parseInt(10.3);         //10 如果是小数，舍去小数点一级后面的内容
```

- ✓ 如果转换的值是string

```
parseInt("123");        //123; 如果仅包含数值，转换为对应的数值  
parseInt("234.1");      //234; 小数点后面的数值省略  
parseInt("+12.1");      //12; 首位为符号位，其余为为数值，转换为整数  
parseInt("1+2.3");      //1; 符号位出现在其他位置，保留符号位前面的数值  
parseInt("0xa");         //10; 如果仅包含十六进制格式，转为为对应的十进制的值  
parseInt("010");         //10; 【注意！】不会当做八进制被解析，结果为10  
parseInt("");            //NaN; 空字符串被转换为NaN  
parseInt("1+2.3");       //1;如果首位为数值，依次向后解析，找到连续的数值，直到遇到第一个非数值的，将之前获取的数值转换为Number返回  
parseInt("123ac");       //123;
```



类型转换

◆ 其他数据类型转换为Number

● **parseFloat()**函数

✓ 如果转换的值是null,undefined,boolean， 均转换为NaN

✓ 如果转换的值是Number

```
parseFloat(10);           //10 如果是整数值，原样输出  
parseFloat(10.1);        //10.1 如果是小数，保留小数点，但是如果10.0结果为10
```

✓ 如果转换的值是string

parseFloat("123"); //123; 如果仅包含数值，转换为对应的数值

parseFloat("234.1");//234.1; 保留小数点后面的数值

parseFloat("+12.1");//12.1; 首位为符号位，其余为数值，转换为整数

parseFloat("1+2.3");//1; 符号位出现在其他位置，保留符号位前的数值

parseFloat("0xa");//0; 不会当做十六进制来解析。

parseFloat("010");//10; 【注意！】不会当做八进制被解析，结果为10

parseFloat(""); //NaN; 空字符串被转换为NaN

parseFloat("1+2.3");//1;如果首位为数值，依次向后解析，找到连续的数值，直到遇到第一个非数值的，将之前获取的数值转换为Number返回

parseFloat("123.3ac");//123.3;



特别注意

- 加法 + (m + n)

- ✓ 当m,n不为String, Object类型的时候，先将m,n转换为Number类型，然后再进行计算

```
true + false;           //1; Number(true)+Number(false);
```

```
true + 1;              //2; Number(true) + 1
```

```
null + undefined;     //NaN; Number(undefined) -> NaN
```

- ✓ 当m,n有一个为String,无论另一个操作数为何（但不为对象）都要转换为String，然后再进行拼接

```
"1" + true;            // 1true
```

```
"1" + undefined;      // 1undefined
```

```
"1" + 1;               // 11
```

- ✓ 当m,n有一个为对象，如果该对象既重写toString,又重写了valueOf方法，先调用valueOf方法获取返回值，将该返回值和另外一个操作数进行运算。如果该对象没有重写valueOf方法，将调用toString方法获取返回值，将该返回值和另外一个操作数进行运算。

```
var o = {name:"briup",valueOf:function(){return "1";}}
```

```
o+1;                  //2; 1+1
```



特别注意

- 默认情况下，ECMAScript会将小数点后带有6个零以上的浮点数转化为科学计数法。

0.0000003 => 3e-7

- 在进行算术计算时，所有以八进制十六进制表示的数值都会被转换成十进制数值。
- 保存浮点数需要的内存是整数的两倍，因此ECMAScript会不失时机将浮点转换为整数

例如：

```
var a = 1.;
```

```
var b = 1.0;           //都将解析为1
```

- 避免测试某个特点的浮点数值，是使用IEEE754数值的浮点计算的通病

例如：

```
0.1+0.2 //结果不是0.3，而是0.30000000000000004
```





杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

第 3 章: 流程控制语句

Brighten Your Way And Raise You Up.

流程控制语句

◆ 分支语句

● If语句

condition表示任意表达式，该表达式求值的结果不一定是布尔类型，如果不是布尔类型，ECMAScript会调用Boolean()转换函数将这个表达式结果转换为一个布尔类型，当该值为true时，执行if代码块中的内容。

```
1 if (condition) {  
2     code to run if condition is true  
3 }  
4  
5 run some other code
```

● if-else

当condition为true时，执行if代码块中的内容，否则，执行else代码块中的内容，一般情况下，如果代码块中代码只有一行，可以省略大括号。

```
1 if (condition) code to run if condition is true  
2 else run some other code instead
```



流程控制语句

- ◆ 分支语句
- if-else if-else

多条件分支，当condition1为true时，执行statement1,否则当condition2为true时执行statement2，当condition1,condition2都为false的时候执行statement3。

```
If(condition1){  
    statement1  
} else if(condition2){  
    statement2  
} else {  
    statement3  
}
```



流程控制语句

◆ 分支语句

● switch

expression可以是变量也可以是表达式，当**expression==choice**，执行当前**case**代码块的代码。每个**case**代码块都必须包含**break;** 表示执行完当前代码块的内容跳出**switch**代码块。当所有**case**不满足情况下，执行**default**代码块的内容。**default**位置可以随意。

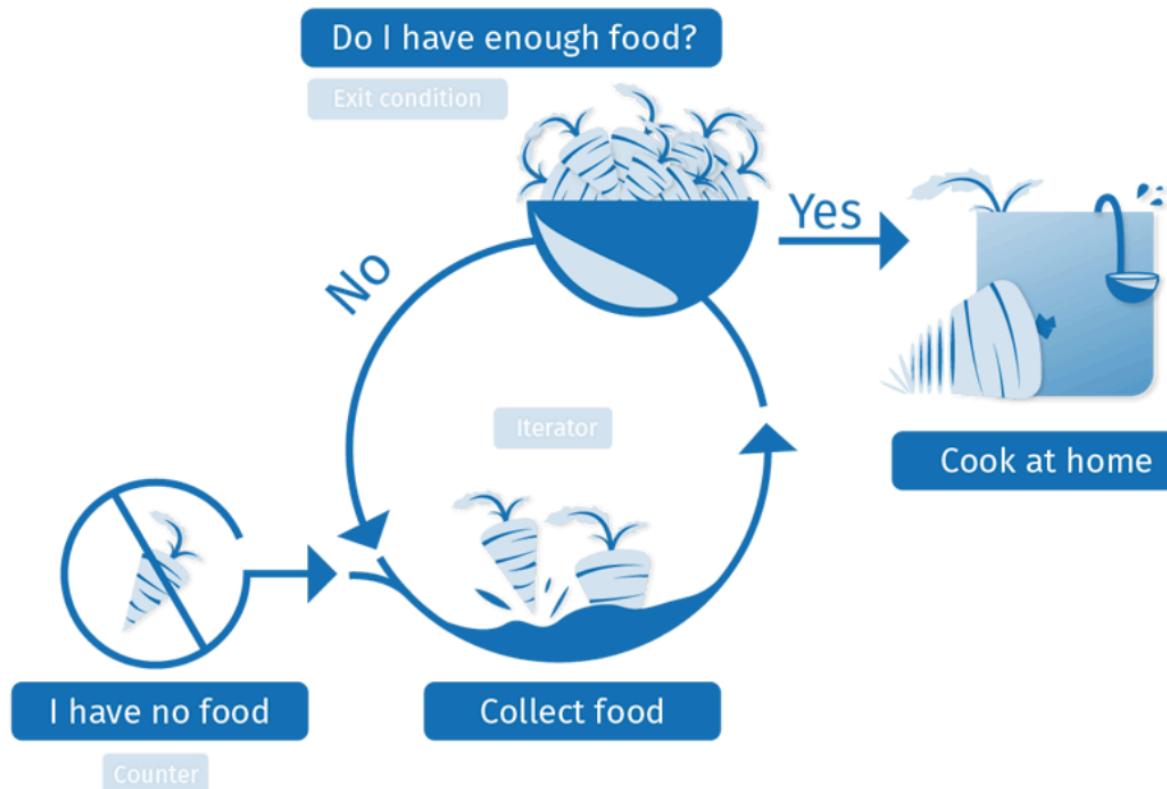
```
1  switch (expression) {  
2      case choice1:  
3          run this code  
4          break;  
5  
6      case choice2:  
7          run this code instead  
8          break;  
9  
10     // include as many cases as you like  
11  
12     default:  
13         actually, just run this code  
14 }
```



流程控制语句

◆ 循环语句

一个循环语句应该具备三要素：计数器，循环结束条件，迭代器。



流程控制语句

◆ 循环语句

● for 循环

- ✓ **Initializer**, 初始化值，一般为数字，仅会执行一次。也可以写到循环体外
- ✓ **exit-condition**, 结束条件，通常使用逻辑运算符进行结束循环判断。每次执行循环体之前均会执行该代码。
- ✓ **final-expression**, 每次执行完循环体代码后执行，通常用于迭代使其更加靠近结束条件。

```
1 | for (initializer; exit-condition; final-expression) {  
2 |     // code to run  
3 | }
```



流程控制语句

- 关键字 break

如果想在所有迭代前退出，即可使用break。当执行break后，会立即跳出循环体，执行下面的代码。

- 关键字 continue

与break不同的是，continue不会跳出循环。而是立即结束当前循环，进入下一次循环。

- 关键字 label

使用label可以在代码中添加标签，以便将来使用

```
label : for(int i=0;i<10;i++){  
    if(i == 5){  
        break label;  
    }  
}
```

➤ 注意！ECMAScript不存在块级作用域，在循环内部定义的变量也可以在外部访问到



流程控制语句

◆ 循环语句

● while 循环

前测试循环语句，即在循环体内的代码被执行之前，就会对出口条件求值。因此，循环体内的代码有可能永远不会被执行

- ✓ **Initializer**, 初始化值，一般为数字，仅会执行一次。也可以写到循环体外
- ✓ **exit-condition**, 结束条件，通常使用逻辑运算符进行结束循环判断。每次执行循环体之前均会执行该代码。
- ✓ **final-expression**, 每次执行完循环体代码后执行，通常用于迭代使其更加靠近结束条件。

```
1  initializer
2  while (exit-condition) {
3      // code to run
4
5      final-expression
6 }
```



流程控制语句

◆ 循环语句

● do-while 循环

后测试循环语句，即只有在循环体中的代码执行之后，才会测试出口条件。循环体内的代码最少被执行一次。

- ✓ **Initializer**, 初始化值，一般为数字，仅会执行一次。也可以写到循环体外
- ✓ **exit-condition**, 结束条件，通常使用逻辑运算符进行结束循环判断。每次执行循环体之前均会执行该代码。
- ✓ **final-expression**, 每次执行完循环体代码后执行，通常用于迭代使其更加靠近结束条件。

```
1 | initializer
2 | do {
3 |   // code to run
4 |
5 |   final-expression
6 | } while (exit-condition)
```



课后练习

- 分别使用**while/do-while/for**循环实现10的阶乘（使用递归算法）。
- 打印九九乘法表（四种形式）
- 有1、2、3、4个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？
- 判断101-200之间有多少个素数，并输出所有素数（只能被1和它本身整除的自然数为素数）
- 打印出所有的“水仙花数”，所谓“水仙花数”是指一个三位数，其各位数字立方和等于该数本身。例如：153是一个“水仙花数”，因为 $153=1^3+5^3+3^3$ 的三次方
- 将一个正整数分解质因数。例如：输入90,打印出 $90=2*3*3*5$ 。
- 求任意两个正整数的最大公约数和(GCD)和最小公倍数(LCM)
- 求1000以内的完全数（若一个自然数，恰好与除去它本身以外的一切因数的和相等，这种数叫做完全数。）





杰普软件科技有限公司
www.briup.com

电邮: training@briup.com
主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层
邮编: 215311
电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区
邮编: 200436
电话: 021-56778147

Briup High-End IT Training

第 4 章：对象及函数

Brighten Your Way And Raise You Up.

对象

ECMAScript中的对象其实就是一组数据(属性)和功能(方法)的集合。

● 创建Object实例：

- ✓ 使用构造函数创建，`new Object()`

```
var person = new Object();
person.name = "briup";
person.age = 22;
```

- ✓ 使用对象字面量表示法

不同的属性之间用','分割， 属性名和属性值之间用':'分割

```
var person = {
    name : "briup",
    age : 22
};
```



复杂数据类型Object

● 访问对象属性

- ✓ 点表示法，右侧必须是以属性名称命名的简单标识符

person.name

- ✓ 中括号表示法

中括号中必须是一个计算结果为字符串的表达式，可以通过变量访问属性，如果属性名中含语法错误的字符，或者属性名使用的是关键字或保留字，可以使用中括号

person["first name"]

● 删除属性

delete只是断开了属性和宿主对象的联系，而不会操作属性中的属性，并且**delete**只会删除自有属性，不能删除继承属性。在销毁对象时，为了防止内存泄露，遍历对象中的属性，依次删除所有属性。

语法：**delete** 属性访问表达式

例如：

delete stu.name //删除学生对象中的name属性



复杂数据类型Object

● 检测属性

- ✓ `in` 检测某属性是否是某对象的自有属性或者是继承属性
- ✓ `hasOwnProperty()`检测给定的属性是否是对象的自有属性，对于继承属性将返回`false`
- ✓ `propertyIsEnumerable()` 检测给定的属性是否是该对象的自有属性，并且该属性是可枚举的通常由JS代码创建的属性都是可枚举的，但是可以使用特殊的方法改变可枚举性。

例如：

```
var o = {  
    x:1  
}  
  
o.hasOwnProperty("x");          //true,x 为o的自有属性  
o.hasOwnProperty("y");          //false,o 中不存在属性y  
o.hasOwnProperty("toString");   //false,toString为继承属性  
o.propertyIsEnumerable("toString"); //false,不可枚举
```



复杂数据类型Object

● Object属性及方法

Object 类型所具有的任何属性和方法也同样存在于其他对象中，任何对象继承于 Object 对象。Object 中常用的方法：

- ✓ constructor: //保存用户创建当前对象的函数
- ✓ hasOwnProperty(propertyName); //检查给定的属性名是否是对象的自有属性，
- ✓ toString(); //返回对象的字符串表示
- ✓ valueOf(); //返回对象的字符串，数值，布尔值的表示。
- ✓ propertyIsEnumerable(propertyName); //检查给定的属性在当前对象实例中是否存在
- ✓ isPrototypeOf(object); //检查传入的对象是否是原型
 - Object.prototype.isPrototypeOf(obj) true
- ✓ toLocaleString(); //返回对象的字符串表示，该字符串与执行环境的地区对应



- 对象序列化

对象序列化是指将对象的状态转换为字符串，也可以反序列化，将字符串还原为对象函数，`RegExp`,`Error`对象，`undefined`值不能序列化和反序列化。

- ✓ `JSON.stringify(obj)` //将对象序列化为Json字符串,只能序列化对象可枚举的自有属性。
- ✓ `JSON.parse(jsonStr)` //反序列化



函数

◆ 函数介绍

函数允许我们封装一系列代码来完成特定任务。当想要完成某一任务时，只需要调用相应的代码即可。方法（method）一般为定义在对象中的函数。浏览器为我们提供了很多内置方法，我们不需要编写代码，只需要调用方法即可完成特定功能。

```
var myArray = ['I', 'love', 'chocolate', 'frogs'];
```

```
var madeAString = myArray.join(' ');
```

```
var myNumber = Math.random()
```



◆ 自定义函数

函数由**function**关键字声明，后面紧跟函数名，函数名后面为形参列表，列表后大括号括起来的内容为函数体。也可以将一个匿名函数（没有函数名的函数）赋值给一个函数变量，这种方式称为函数表达式。解析器在向执行环境中加载数据时，会率先读取函数声明，并使其在执行任何代码之前可用；当执行器执行到函数表达式的代码的时候才会真正的解释执行。

● 表示方法：

✓ 函数声明

```
function 函数名(形参列表){  
    //函数体  
}
```

✓ 函数表达式

```
var 函数名 = function 函数名(形参列表){  
    //函数体  
}
```



函数

◆ 函数的调用

函数声明好之后并不会直接运行，需要进行调用才能运行。

● 调用方法：

- ✓ 函数名(实参列表);
- ✓ 函数名.call(执行环境对象,实参列表);
- ✓ 函数名.apply(执行环境对象,实参列表数组);



◆ 函数的内部属性

只有在函数内部才能访问的属性。

- **arguments :**

是类数组对象，包含着传入函数中参数，**arguments**对象还有一个**callee**的属性，用来指向拥有这个**arguments**对象的函数

- **this:**

指向的是函数赖以执行的环境对象

函数

◆ 函数属性和方法

函数本质上也是一种对象，拥有属性和方法。

- **length:**

表示函数希望接受的命名参数的个数，即形参的个数。

- **apply():**

可以调用当前函数，并可以指定其执行环境对象

- **call():**

可以调用当前函数，并可以指定其执行环境对象



函数

◆ 函数的应用

函数本质上是一种对象，可以将其当做普通对象来使用。

● 作为参数：

由于函数名本身就是变量，所以函数可以当做值来使用（参数，返回值）。

```
function callOther(fun,args){  
    return fun(args);  
}  
  
function show(msg){  
    alert(msg);  
}.
```

● 作为返回值()：

```
function getFunction(){  
    return function(){  
        alert(hello);  
    }  
}
```



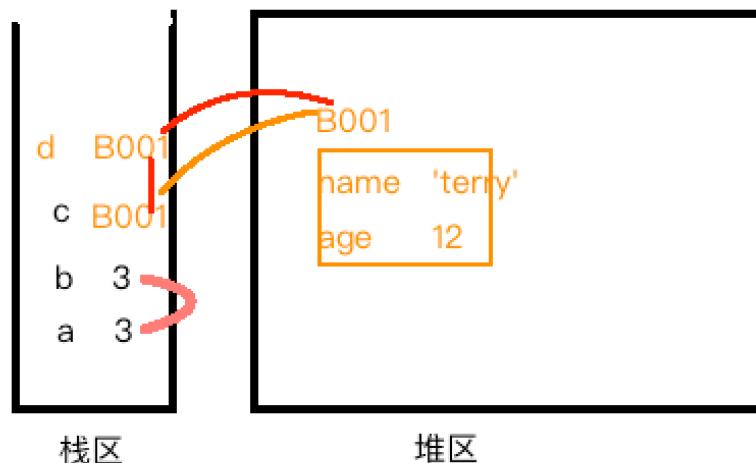
值传递与引用传递

- 基本数据类型的变量：

- ✓ 可以直接操作保存在变量中的实际的值
- ✓ 参数传递的时候传递的是实际值

- 引用数据类型的变量：

- ✓ 不能直接操作对象的内存空间，实际上是在操作对象的引用。可以为引用类型变量添加属性和方法，也可以改变和删除其属性和方法。
- ✓ 参数传递的时候传递的是引用地址。



```

var a = 3;
var b = a;
var c = {
    name: 'terry',
    age: 12
};
var d = c;

```

b=a , 实际上将a代表的值复制一份，赋值给b
c 为引用类型变量，所有c中保存的是指向对象的指针（引用）
d = c, 实际上是将c中代表的引用地址复制给d



杰普软件科技有限公司
www.briup.com

电邮: training@briup.com
主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层
邮编: 215311
电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区
邮编: 200436
电话: 021-56778147

Briup High-End IT Training

第 5 章: 数组

Brighten Your Way And Raise You Up.

数组

ECMAScript数组是有序列表，是存放多个值的集合。

有以下特性：

- 每一项都可以保存任何类型的数据。
- 数组的大小是可以动态调整。
- 数组的length属性：可读可写，可以通过设置length的值从数组的末尾移除项或向数组中添加新项



◆ 初始化

- 使用**Array**构造函数

```
var arr = new Array();
```

```
var arr = new Array(20); // 预先指定数组的大小
```

```
var arr = new Array("terry","larry","boss"); //传入参数
```

- 使用数组字面量

由一对包含数组项的方括号表示， 多个数组项之间用逗号分隔

```
var arr = ["terry","larry","boss"];
```

```
var arr = [] //空数组
```

数组

◆ 访问数组元素

数组变量名[索引]

- 如果索引小于数组的长度， 返回对应项的值

```
var arr = ["terry", "larry", "boss"];
```

```
arr[0]; //访问数组中第一个元素,返回值为terry
```

- 如果索引大于数组的长度， 数组自动增加到该索引值加1的长度

```
var arr = ["terry", "larry", "boss"];
```

```
arr[3] = "jacky"; //添加元素,数组程度变为4
```

- 注意！ 数组最多可以包含4 294 967 295个项



数组

◆ 数组检测

对于一个网页或者一个全局作用域而言，使用`instanceof`操作符即可判断某个值是否是数组。如果网页中包含多个框架，这样就存在两个不同的全局执行环境，从而存在两个不同版本的`Array`构造函数，这样就会判断不准确。为解决这个问题，`ECMAScript5`新增了`Array.isArray()`方法进行判断

```
var arr = [];
typeOf(arr);           //结果为object
arr instanceof Array   //结果为true，在同一个全局作用域下可以这么判断
Array.isArray(arr);     //结果为true，判断arr是否是数组类型
```

◆ 数组序列化

- ✓ `toString()` 在默认情况下都会以逗号分隔字符串的形式返回数组项
- ✓ `join();` 使用指定的字符串用来分隔数组字符串

➤ 例如

```
var arr = ["terry", "larry", "boss"];
arr.toString()          //terry,larry,boss
arr.join("||");         //briup||terry||jacky
```



◆ 栈，队列方法

● 栈 LIFO (Last-In-First-Out)

- ✓ `push()` 可接受任意类型的参数，将它们逐个添加到数组的末尾，并返回数组的长度
- ✓ `pop()` 从数组的末尾移除最后一项，减少数组的`length`值，返回移除的项

● 队列 FIFO (First-In-First-Out)

- ✓ `shift()` 移除数组中的第一个项并且返回该项，同时将数组的长度减一。
- ✓ `unshift()` 在数组的前端添加任意个项，并返回新数组的长度。



◆ 排序

- **reverse()** 反转数组项的顺序
- **sort()**
 - ✓ 默认排序：该方法会调用每个数组项的**toString()** 方法，然后按照字符序列排序
 - ✓ 自定义排序：
 - a. 该方法可以接受一个比较函数作为参数，比较函数有两个参数
 - b. 如果第一个参数位于第二个参数之前，返回负数
 - c. 如果第一个参数位于第二个参数之后，返回正数

```
var arr = [11,5,23,7,4,1,9,1];
console.log(arr.sort(compare));
//该比较函数适合于大多数数据类型
function compare(v1,v2){
    if(v1>v2){return -1;}
    else if( v1<v2){return 1;}
    else{return 0;}
}
```



◆ 截取方法

- **concat()**

数组拼接，先创建当前数组的一个副本，然后将接收到的参数添加到这个副本的末尾，返回副本，**不改变原数组**。

- **slice()**

数组切割，可接受一个或者两个参数（返回项的起始位置，结束位置），当接受一个参数，从该参数指定的位置开始，到当前数组末尾的所有项。当接受两个参数，起始到结束之间的项，但是不包含结束位置的项。**不改变原数组**

- **splice()**

向数组的中部插入数据将始终返回一个数组，该数组中包含从原始数组中删除的项。

- ✓ 删除：指定两个参数(删除的起始位置，要删除的项数)
- ✓ 插入：指定三个参数(起始位置，0，要插入的项任意数量的项)
- ✓ 替换：指定三个参数(起始位置，要删除的项数，要插入的任意数量的项)



◆ 索引方法

- **indexOf()**

从数组开头向后查找，使用全等操作符，找不到该元素返回-1。第一个参数为要查找的项，第二个参数（可选）为索引开始位置

- **lastIndexOf()**

从数组末尾向前查找，使用全等操作符，找不到该元素返回-1。第一个参数为要查找的项，第二个参数（可选）为索引开始位置

◆ 迭代方法

参数：每一项上运行的函数，运行该函数的作用域对象（可选）

● **every()**

对数组中的每一运行给定的函数，如果该函数对每一项都返回true，则该函数返回true

eg:

```
var arr = [11,5,23,7,4,1,9,1];
var result = arr.every(function(item,index,arr){
    return item >2;
});
console.log(result); //false
```



Array类型

- **some()**

对数组中的每一运行给定的函数，如果该函数对任一项都返回true，则返回true

eg:

```
var result = arr.every(function(item,index,arr){  
    return item >2;  
});  
console.log(result); //true
```



Array类型

- **filter()**

对数组中的每一运行给定的函数，会返回满足该函数的项组成的数组

eg:

```
var result = arr.filter(function(item,index,arr){  
    return item >2;  
});  
console.log(result); // [11, 5, 23, 7, 4, 9]
```



Array类型

● map()

对数组中的每一元素运行给定的函数,返回每次函数调用的结果组成的数组

eg:

```
var result = arr.map(function(item,index,arr){  
    return item * 2;  
});  
console.log(result); // [22, 10, 46, 14, 8, 2, 18, 2]
```



Array类型

- **forEach()**

对数组中的每一元素运行给定的函数,没有返回值, 常用来遍历元素

eg:

```
var result = arr.forEach(function(item,index,arr){  
    console.log(item);  
});
```





杰普软件科技有限公司
www.briup.com

电邮: training@briup.com
主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层
邮编: 215311
电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区
邮编: 200436
电话: 021-56778147

Briup High-End IT Training

第 6 章: 正则表达式

Brighten Your Way And Raise You Up.

正则表达式

◆ 正则表达式

是一个描述字符模式的对象.

- 正则表达式对象的创建

构造函数

第一个参数包括正则表达式的主体部分，即正则表达式直接量中两条斜线之间的文本，第二个参数指定正则表达式的修饰符。只能传入g ,i,m或者其组合，可以省略

```
var pattern =new RegExp("正则表达式","修饰符")
```

```
var pattern =new RegExp("abc","ig");
```

正则表达式字面量

```
var pattern = /正则表达式/修饰符;
```

```
var pattern = /abc/ig;
```



正则表达式

◆ 修饰符

i ignore case 不区分大小写

g global 全局

m multiline 多行



◆ 原型属性

RegExp.prototype.global 布尔值，表明这个正则表达式是否带有修饰符g

RegExp.prototype.ignoreCase 布尔值，表明这个正则表达式是否带有修饰符i

RegExp.prototype.multiline 布尔值，表明这个正则表达式是否带有修饰符m

RegExp.prototype.lastIndex 如果匹配模式带有g，这个属性存储在整个字符串中下一次检索的开始位置，这个属性会被exec(), test()方法调用到

RegExp.prototype.source 包含正则表达式文本

正则表达式

◆ 原型方法

RegExp.prototype.exec()

RegExp.prototype.test()

RegExp.prototype.toString()



正则表达式

```
var result = pattern.exec(str)
```

检索字符串中的正则表达式的匹配

- ✓ 参数 : 字符串
- ✓ 返回值 : 数组或者null

数组：匹配到的结果

- 如果正则表达式中有修饰符"g",这时，在pattern中会维护lastIndex属性，记录下一次开始的位置，当第二次执行exec的时候，从lastIndex开始检索。
- 如果正则表达式中没有修饰符"g",不会维护lastIndex属性，每次执行从开始位置检索



正则表达式

```
var result = pattern.test(str);
```

检测一个字符串是否匹配某个模式

- ✓ 参数：字符串
- ✓ 返回值：布尔类型 `true`代表有符合条件的，`false`代表没有符合条件的



◆ 字符类

[直接量]

- . (点号, 小数点) 匹配任意单个字符, 但是行结束符除外
- \d 匹配任意阿拉伯数字。等价于[0-9]
- \D 匹配任意一个不是阿拉伯数字的字符。等价于[^0-9]。
- \w 匹配任意来自基本拉丁字母表中的字母数字字符, 还包括下划线。等价于 [A-Za-z0-9_]。
- \W 匹配任意不是基本拉丁字母表中单词（字母数字下划线）字符的字符。等价于 [^A-Za-z0-9_]。
- \s 匹配一个空白符, 包括空格、制表符、换页符、换行符和其他 **Unicode** 空格。
- \S 匹配一个非空白符。
- \t 匹配一个水平制表符 (**tab**)
- \r 匹配一个回车符 (**carriage return**)
- \n 匹配一个换行符 (**linefeed**)
- \v 匹配一个垂直制表符 (**vertical tab**)
- \f 匹配一个换页符 (**form-feed**)



◆ 字符集合

[xyz] 一个字符集合，也叫字符组。匹配集合中的任意一个字符。你可以使用连字符‘-’指定一个范围。**[0-9] [a-z]**

[^xyz] 一个反义或补充字符集，也叫反义字符组。也就是说，它匹配任意不在括号内的字符。你也可以通过使用连字符 ‘-’ 指定一个范围内的字符。



正则表达式

◆ 边界

^ 匹配输入开始。如果多行（**multiline**）标志被设为 **true**，该字符也会匹配一个断行（**line break**）符后的开始处。

\$ 匹配输入结尾。如果多行（**multiline**）标志被设为 **true**，该字符也会匹配一个断行（**line break**）符的前的结尾处。

\b 匹配一个零宽单词边界（**zero-width word boundary**），如一个字母与一个空格之间。

\B 匹配一个零宽非单词边界（**zero-width non-word boundary**），如两个字母之间或两个空格之间。



正则表达式

◆ 分组

(x) 匹配 x 并且捕获匹配项。这被称为捕获括号（**capturing parentheses**）。

\n n 是一个正整数。一个反向引用（**back reference**），指向正则表达式中第 n 个括号（从左开始数）中匹配的子字符串。

例如：/\w+:\V\(\w+(.)\w+\)\w+\1\w+/-



正则表达式

◆ 数量词

x* 匹配前面的模式 **x** 0 或多次。

x⁺ 匹配前面的模式 **x** 1 或多次。等价于 {1,}。

x^{*}? 像上面的 * 一样匹配前面的模式 **x**，然而匹配是最小可能匹配。

x⁺? 像上面的 + 一样匹配前面的模式 **x**，然而匹配是最小可能匹配。

x? 匹配前面的模式 **x** 0 或 1 次。

x|y 匹配 **x** 或 **y**

x{n} **n** 是一个正整数。前面的模式 **x** 连续出现 **n** 次时匹配

x{n,} **n** 是一个正整数。前面的模式 **x** 连续出现至少 **n** 次时匹配。

x{n,m} **n** 和 **m** 为正整数。前面的模式 **x** 连续出现至少 **n** 次，至多 **m** 次时匹配。

◆ Javascript 中 String 对正则表达式的支持

- **search()**

参数为一个正则表达式。如果参数不为正则表达式，则先通过`RegExp`将其转换为构造函数。不支持全局检索，返回第一个与之匹配的子串的位置，如果找不到匹配的子串，返回-1。类似于正则表达式的 `test` 方法

- **match()**

最常用的正则表达式方法，参数为正则表达式。**返回由匹配结果组成的数组或者null**。当正则表达式中没有`g`修饰符的时候，就不是全局匹配。这时，数组的第一个元素就为匹配的字符串，剩余的元素则是由正则表达式中用圆括号括起来的子表达式。如果该正则表达式设置为修饰符`g`，则该方法返回的数组包含字符串中所有匹配结果。类似于正则表达式的 `exec` 方法

```
"1 plus 2 equals 3".match(/\d+/g) //返回["1","2","3"]
```

◆ Javascript 中 String 对正则表达式的支持

● replace()

用以执行检索和替换操作。第一个参数是正则表达式，第二个参数是要替换的字符串。

```
text.replace(/javascript/gi, "JavaScript"); //不区分大小写将所有javascript转换为JavaScript
```

● split()

将字符串转成数组。参数可以为正则表达式

```
"1, 2, 3, 4, 5".split(/\s*,\s*/);      //["1","2","3","4","5"] 允许分隔符左右两边留有空白
```





杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

第 7 章: 内置对象及内置函数

Brighten Your Way And Raise You Up.

基本包装类型

为了便于操作基本类型值，ECMAScript提供了3个特殊的引用类Boolean, Number, String。每当读取一个基本类型值的时候，后台就会创建一个对应的基本包装类型对象，从而可以调用一些方法操作这些数据。

例如：

```
var s = "briup";  
s.substring(2);
```

后台会自动完成以下操作：

- a. 创建String类型的一个实例
- b. 在实例上调用指定的方法
- c. 销毁这个实例



基本包装类型

Object构造函数会像工厂方法一些，根据传入的值的类型返回相应基本包装类型的实例

```
var obj = new Object("briup"); //obj 类型为String包装类型  
console.log(obj instanceof String);
```

使用new调用基本包装类型的构造函数，与直接调用同名的转换函数不一样

```
var s = "11";  
  
var s1 = Number(s);           //转型函数 number类型  
  
var s2 = new Number(s);       //构造函数 object类型
```



基本包装类型

- ◆ Boolean, Number, 不建议直接使用这两种包装器类型
- ◆ String 类型

length 属性，获取字符串的字符数量

charAt(i) 返回给定位置的字符

charCodeAt(i) 返回给定位置的字符的字符编码

例如：

```
var s = "helloworld";  
s.charAt(1); //e  
s.charCodeAt(1); //101
```

indexOf(); :从前往后查找指定字符所在位置

lastIndexOf();

从后往前查找字符串所在位置，可以有第二个参数，代表从字符串中哪个位置开始查找。



基本包装类型

concat()

将一个或多个字符串拼接起来，返回拼接得到的新字符串，但是大多使用“+”拼接

slice() 截取字符串（开始位置，返回字符后一个字符位置）

substr() 截取字符串（开始位置，返回字符个数）

substring() 截取字符串（开始位置，返回字符后一个字符位置，不改变原值大小）

```
var s = "helloworld";  
  
s.slice(3,7);      //lowo  
  
s.substr(3,7);    //loworld  
  
s.substring(3,7); //lowo
```

trim(): 删除前置以及后置中的所有空格，返回结果

toLowerCase(): 转换为小写

toUpperCase(): 转换为大写



Math对象

◆ 常用方法

● 比较方法

- ✓ Math.min() //求一组数中的最小值
- ✓ Math.max() //求一组数中的最大值
- ✓ Math.min(1,2,19,8,6); //1

● 将小数值舍入为整数的几个方法

- ✓ Math.ceil() 向上舍入
- ✓ Math.floor() 向下舍入
- ✓ Math.round() 四舍五入

Math.ceil(12.41); //13 Math.floor(12.41); //12

Math.round(12.3); //12 Math.round(12.5); //13

● 随机数

- ✓ Math.random() //返回大于0小于1的一个随机数



Math对象

◆ 其他方法: (了解即可, 即用即查)

| | |
|----------------|----------------|
| abs(num) | 返回num绝对值 |
| exp(num) | 返回Math.E的num次幂 |
| log(num) | 返回num的自然对数 |
| pow(num,power) | 返回num的power次幂 |
| sqrt(num) | 返回num的平方根 |
| scos(x) | 返回x的反余弦值 |
| asin(x) | 返回x的反正弦值 |
| atan(x) | 返回x的反正切值 |
| atan2(y,x) | 返回y/x的反正切值 |
| cos(x) | 返回x的余弦值 |
| sin(x) | 返回x的正弦值 |
| tan(x) | 返回x的正切值 |



Date对象

◆ 将一个字符串转换为Date对象：

```
var str = "2012-12-12";  
  
var date = new Date(str);  
  
//字符串转换为Date对象  
  
document.write(date.getFullYear());  
  
//然后就可以使用Date对象的方法输出年份了
```

◆ Date的方法

● Date.prototype.getDate()

返回是日期对象中月份中的几号。

例如：

```
var date = new Date();           //2012-12-19  
  
document.write(date.getDate()); //返回 19 是19号
```



Date对象

- **Date.prototype.getDay()**

返回日期中的星期几 星期天0-星期6

eg:

```
var date = new Date();
document.write(date.getDay()); //3 星期3
```

- **Date.prototype.getFullYear()**

返回年份 如2012。

eg:

```
var date = new Date();
document.write(date.getFullYear()); //返回2012,2012年
```

- **Date.prototype.getHours()**

返回日期中的小时， 几点了， 0-23

eg:

```
var date = new Date();
document.write(date.getHours()); //返回23, 晚上11点
```



Date对象

- **Date.prototype.getMilliseconds()**

返回日期中的毫秒数

eg:

```
var date = new Date();
document.write(date.getMilliseconds());
//返回27    当前是xx年, xx月, xx点, xx分, xx秒, xx毫秒的毫秒
```

- **Date.prototype.getMinutes()**

返回日期中的分钟数 0-59

eg:

```
var date = new Date();
document.write(date.getMinutes());
//2012-12-19 23:22    返回22, 12点22分
```



Date对象

- **Date.prototype.getMonth()**

返回日期中的月份数，返回值0(1月)-11(12月)

eg:

```
var date = new Date();
document.write(date.getMonth());
//2012-12-19    此处返回11
```

(注意此处与通常理解有些偏差，1月份返回是0,12月返回是11)

- **Date.prototype.getSeconds()**

返回一个日期的秒数

eg:

```
var date = new Date();
document.write(date.getSeconds());
```

//返回34, 2012-12-19 23:27:34 27分34秒



Date对象

- **Date.prototype.getTime()**

将一个日期对象以毫秒形式返回

eg:

```
var date = new Date();
document.write(date.getTime());
```

//返回1355930928466 返回值是1970-01-01 午夜到当前时间的毫秒数。

- **Date.prototype.getTimezoneOffset()**

GMT时间与本地时间差，用分钟表示

eg:

```
var date = new Date();
document.write(date.getTimezoneOffset());
```

//返回-480 实际上这个函数获取的是javascript运行于哪个时区。单位是分



Date对象

- **Date.prototype.getYear()**

返回Date对象中的年份值减去1900

```
var date = new Date();
```

```
document.write(date.getYear()); //2012-12-19 返回112 (2012-1900)
```

- **Date.prototype.now()**

静态方法 //返回1970-01-01午夜到现在的时间间隔，用毫秒表述

```
document.write(Date.now());
```

//静态方法，返回当前时间与1970-01-01的时间间隔，毫秒单位。

- **Date.prototype.valueOf()**

如果是一个Date对象，将一个Date对象转为毫秒的形式，否则不显示

```
var date = "";
```

```
document.write(date.valueOf()); //不是Date对象，不输出
```

```
var date1 = new Date();
```

```
document.write(date1.valueOf()); //输出1356180400916
```



Date对象

- **Date.prototype.toDateString()**

以字符串的形式返回一个Date的日期部分

```
var date = new Date();  
  
document.write(date.toDateString("yyyy-MM-dd"));
```

- **Date.prototype.toTimeString()**

以字符串的形式返回一个Date的时间部分

```
var date = new Date();  
  
document.write(date.toTimeString("yyyy-MM-dd"));
```

- **Date.prototype.toISOString()**

将一个Date对象转换为ISO-8601格式的字符串,返回的字符串格式为yyyy-mm-ddThh:mm:ssZ

```
var date = new Date();  
  
document.write(date.toISOString());
```



Date对象

- **Date.prototype.toJSON()**

JSON序列化一个对象

```
var date = new Date();
document.write(date.toJSON());
```

- **Date.prototype.toLocaleDateString()**

以本地格式的字符串返回一个Date的日期部分,返回一个本地人可读的日期格式, 日期部分

```
var date = new Date();
document.write(date.toLocaleDateString());
```

- **Date.prototype.toLocaleString()**

将一个Date转化难为一个本地格式的字符串

```
var date = new Date();
document.write(date.toLocaleString());
```



Date对象

- **Date.prototype.toLocaleTimeString()**

将一个Date转化为本地的格式的时间部分

```
var date = new Date();
document.write(date.toLocaleTimeString());
```

- **Date.prototype.toString()**

将一个Date转换为一个字符串

```
var date = new Date();          //现在是2012-12-22
document.write(date.toString());//返回Sat Dec 22 2012 19:59:17 GMT+0800
```

- **Date.prototype.toTimeString()**

以字符串的形式返回一个Date对象的时间部分

```
var date = new Date();
document.write(date.toTimeString());
```





杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

第 8 章: 面向对象的程序设计

Brighten Your Way And Raise You Up.



杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

深入理解对象

Brighten Your Way And Raise You Up.

深入理解对象

◆深入理解对象

ECMA-262对象的定义：无序属性的集合，其属性可以包含基本值，对象，或者函数。
。可以将对象想象成散列表:键值对，其中值可以是数据或者函数

● 属性类型

- ✓ 数据属性：例如：name

包含一个属性值的位置，这个位置可以读取和写入值。

[[Configurable]]:

表示是否通过**delete**删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为访问器属性直接定义在对象中，（默认为**true**）

[[Enumerable]]:

表示能否通过**for-in**循环返回属性。（直接定义在对象中，（默认为**true**）

[[Writable]]: 表示能否修改属性的值。（直接定义在对象中，（默认为**true**）

[[Value]]: 包含这个属性的数据值 name:jacky

要修改属性默认的特性，必须使用**ECMAScript5**的**Object.defineProperty()**方法

defineProperty(属性所在的对象,属性的名字,一个描述符对象);

configurable: 当为**false**时，不能修改



✓ 访问器属性 例如:`_year--> year`(访问器属性)

访问器属性不包含数据值，包含一对setter,getter方法。

`[[Get]]` 在获取属性时调用的函数，默认为`undefined`

`[[Set]]` 在写入属性时调用的函数，默认为`undefined`

```
var book = { _year :2004,edition:1}
```

```
Object.defineProperty(book,"year",{
```

```
    get:function(){
```

```
        return this._year;
```

```
    },
```

```
    set:function(year){
```

```
        this._year = _year;
```

```
    }
```

```
});
```

`_year`前面的下划线是一种常用的记号，用来表示只能通过对象方法访问的属性。而访问器属性`year`则包含一个`getter`函数和一个`setter`函数。



深入理解对象

- 定义多个属性

Object.defineProperties();

该方法接受两个对象参数，第一个是要添加或者要修改属性的对象；第二个对象的属性和第一个对象中要添加和修改的属性对应内容：

```
var book = {};
Object.defineProperties(book,{  
    _year :{  
        value:1001},  
    edition :{  
        value:1},  
    year :{  
        get:function(){  
            return this._year+1},  
        set:function(year){  
            this._year = year}  
    }  
});  
console.log(book.year);
```



深入理解对象

- 读取属性的特性

Object.getOwnPropertyDescriptor();获取指定属性的描述符该方法接受两个参数，第一个为属性所在的对象,第二个为要读取其描述符的属性名称

```
var descriptor = Object.getOwnPropertyDescriptor(book, "_year");
console.log(descriptor.value);           //1001
console.log(descriptor.configurable)    //false
```



创建对象

◆ 创建对象

- 工厂模式

```
function createPerson(name,age,job){  
    var o = new Object();  
    o.name = name;  
    o.age = age;  
    o.job = job;  
    o.sayName = function(){  
        alert(this.name);  
    }  
    return o;  
}  
  
var p1 = createPerson("terry",11,"boss");  
var p2 = createPerson("larry",12,"daBoss");
```

➤ 工厂模式的问题

```
var t1 = typeOf p1; //object 无法对象识别，即所有对象都是Object类型
```



创建对象

- 构造函数模式

js中可以自定义构造函数，从而自定义对象类型的属性和方法，构造函数本身也是函数，只不过可以用来创建对象

```
function Person(name,age,job){  
    this.name = name;  
    this.age = age;  
    this.job = job;  
    this.sayName = function(){  
        alert(this.name);  
    }  
}  
  
var p1 = new Person("terry",11,"boss");  
var p2 = new Person("larry",12,"daBoss");
```



创建对象

使用new操作符调用构造函数创建对象实际上经历了如下几个步骤

- 1) 创建一个新对象
- 2) 将构造函数的作用域赋给新对象（this指向这个新对象）
- 3) 执行构造函数中的代码
- 4) 返回新对象

这种创建对象的方法可以将实例标识为一种特定类型（例如Person类型）。

```
var t1 = typeOf p1; //t1为Person
```

1. 构造函数当做函数

```
Person("larry",12,"daBoss")
```

当在全局作用域中调用一个函数时，this总是指向Global对象（window对象）

2. 构造函数的问题

每个方法都需要在每个实例上重新创建一遍，但是毫无必要。

可以在全局范围内声明一个函数，然后将引用传递给对象中的函数属性。但是这样做会导致全局函数过多，体现不了对象的封装性

```
console.log(p1.sayName == p2.sayName); //false
```



创建对象

● 原型模式

每个函数都有一个属性: **prototype**(原型属性), 这个属性是一个指针, 指向一个对象, 该对象的用途是包含可以由特定类型的所有实例共享的属性和方法

```
function Person(){}
Person.prototype.name = "tom";
Person.prototype.age = 22;
Person.prototype.job="boss";
Person.prototype.sayName = function(){
    alert(this.name);
}
var p1 = new Person();
p1.name = "terry";
var p2 = new Person();
p2.name = "larry";
```



创建对象

创建了自定义的构造函数之后，其原型对象默认会取得**constructor**属性；当调用构造函数创建一个新实例后，该实例的内部将包含一个指针（内部属性），指向构造函数的原型对象。（指向的是原型对象而不是构造函数）

1. 属性的访问

每当代码读取某个对象的某个属性时，都会执行一次搜索，目标是具有给定名字的属性。

- 1) 首先从对象实例本身开始查找
- 2) 如果不在对象实例中，则继续搜索指针指向的原型对象。

2. 删除实例属性

当为对象实例添加一个属性时，这个属性就会屏蔽原型对象中保存的同名属性。通过**delete**操作符可以完全删除实例属性。

3. 检测属性是否存在于实例中

hasOwnProperty(p): 判断p指定的属性是否存在于实例中，如果存在返回**true**

`console.log(p1.hasOwnProperty("name")); //false` 存在于原型中而不是实例对象中



创建对象

4. 原型与in操作符

1) 在for-in 可以访问存在于实例中的属性， 以及原型中的属性

2) 单独使用

a in b; 通过b对象可以访问到a属性的时候返回true， 无论该对象在实例中还是在原型中

```
console.log("name" in p1);           //true
```

判断一个属性是否在原型

```
function hasPrototypeProperty(obj,name){  
    //不在实例中但是可以访问到的属性属于原型属性  
    return !obj.hasOwnProperty(name) && (name in obj);  
}
```



创建对象

5. 原生对象的原型

通过原生对象的原型，不仅可以取得所有默认方法的调用，而且可以定义新方法。可以向修改自定义对象的原型一样修改原生对象的原型，可以随时添加方法。

```
String.prototype.startsWith = function(text){  
    return this.indexOf(text) == 0;  
}  
  
var msg = "Hello world";  
alert(msg.startsWith("Hello")); //true
```



创建对象

6. 原型对象的问题

所有实例在默认情况下都将取得相同的属性值，这种共享对于函数来说非常合适，但是包含引用数据类型的值就不太好

```
Person.prototype = {  
    name : "briup",  
    friends : ["larry", "terry"]  
}  
  
var p1 = new Person();  
var p2 = new Person();  
p1.name = "terry";  
p1.friends.push("tom");  
p1.friends;          //["larry", "terry", "tom"]  
p2.friends;          //["larry", "terry", "tom"]
```



创建对象

7. 更简单的原型语法

将原型对象设置为等于一个对象字面量形式创建的新对象。实例对象使用效果相同，但是原型中的**constructor**属性不再指向**Person**，因为每创建一个对象，就会同时创建它的 **prototype** 对象，这个对象也自动获得**constructor**属性。这里我们重写了 **prototype** 对象因此该原型中**constructor** 属性就变成了新对象的**constructor** 属性（Object）

```
p1.prototype.constructor //Object  
function Person(){}
Person.prototype = {
    //constructor: Person, 如果constructor比较重要，可以指定它的值
    name:"tom",
    age :22,
    sayName:function(){
        alert(this.name);
    }
}
Object.defineProperty(Person.prototype,"constructor",{
    enumerable : false,
    value : Person
}); // 定义constructor属性，不可遍历
```



创建对象

● 组合使用构造函数模式和原型模式

构造函数用于定义实例属性，原型模式用于定义方法和共享属性。这种模式是目前在ECMAScript中使用最广泛，认同度最高的一种创建自定义类型的方法。

```
function Person(name,age){  
    this.name = name,  
    this.age = age,  
    this.friends = []  
}  
Person.prototype = {  
    constructor : Person,  
    sayName:function(){  
        alert(this.name);  
    }  
}  
var p1 = new Person("terry",11);  
var p2 = new Person("larry",12);  
p1.friends.push("tom");  
p2.friends.push("jacky");  
console.log(p1);  
console.log(p2);
```





杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

继承

Brighten Your Way And Raise You Up.

◆ 原型链继承

每个构造函数都有一个原型对象，原型对象中都包含一个指向构造函数的指针，而实例都包含一个指向原型对象的内部指针。当原型对象等于另外一个类型的实例即继承。调用某个方法或者属性的步骤

a. 搜索实例

b. 搜索原型

c. 搜索父类原型

//定义父类类型

```
function Animal(){
    this.name = "animal"
}
Animal.prototype = {
    sayName : function(){
        alert(this.name);
    }
}
```



继承

```
//定义子类类型  
function Dog(){  
    this.color = "灰色"  
}  
  
//通过将子对象的原型对象指向父对象的一个实例来完成继承  
Dog.prototype = new Animal();  
  
//子对象的方法其实是定义在了符类对象的实例上。  
Dog.prototype.sayColor = function(){  
    alert(this.color);  
}  
  
var dog = new Dog();  
console.log(dog);  
dog.sayColor();  
dog.sayName();
```



◆ 默认原型

所有函数默认原型都是Object的实例， 默认原型中都会包含一个内部指针， 指向Object.

◆ 确定原型和实例的关系

1) 通过使用instanceof

instance instanceof Object //true

instance instanceof SuperType //true

instance instanceof SubType //false

2) 通过使用isPrototypeOf()

只要是原型链中出现过的原型， 都可以说是该原型链所派生的实例的原型

Object.prototype.isPrototypeOf(instance) //true

SuperType.prototype.isPrototypeOf(instance) //true

SubType.prototype.isPrototypeOf(instance) //true



继承

◆ 谨慎定义方法

子类型覆盖超类型中的某个方法，或者是需要添加超类中不存在的方法，都需要将给原型添加方法的代码放在继承之后（即替换原型的语句之后）

◆ 原型链问题

- ✓ 通过原型来实现继承时，原型实际上会变成另一个类型的实例，原来的实例属性也就变成了现在的原型属性
- ✓ 在创建子类型的实例时，不能向超类型的构造函数传递参数。因此实践中很少会单独使用原型链



◆ 借用构造函数

也称 "伪造对象" 或 "经典继承", 在子类型构造函数的内部调用超类型构造函数。函数不过是在特定环境中执行代码的对象, 因此通过apply(),call()方法可以在(将来)新建对象上执行构造函数, 即 在子类型对象上执行父类型函数中定义的所有对象初始化的代码。结果每个子类实例中都具有了父类型中的属性以及方法

```
function Animal(name){  
    this.name = name;  
    this.colors = ["red","gray"];  
}  
  
function Dog(name){  
    //继承了Animal  
    Animal.call(this,name);  
    this.color = "gray";  
}  
  
Animal.prototype.sayName = function(){  
    alert(this.name);  
}
```



继承

◆组合函数

也称"伪经典继承"，将原型链和借用构造函数的技术组合在一起。原理是：使用原型链实现对原型属性和方法的继承，而通过借用构造函数实现对实例属性的继承

```
function Animal(name){  
    this.name = name;  
    this.colors = ["red","gray"];  
}  
  
function Dog(name){  
    //继承了Animal（属性）  
    Animal.call(this,name);  
    this.color = "gray";  
}  
  
Animal.prototype.sayName = function(){  
    alert(this.name);  
}  
  
//继承方法  
Dog.prototype = new Animal();  
Dog.prototype.constructor = Animal;
```





杰普软件科技有限公司
www.briup.com

电邮: training@briup.com
主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层
邮编: 215311
电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区
邮编: 200436
电话: 021-56778147

Briup High-End IT Training

第 9 章: 文档对象模型

Brighten Your Way And Raise You Up.

DOM是针对HTML和XML文档的一个API（应用程序编程接口），DOM描绘了一个层次化的节点树，允许开发人员添加，移除，修改页面的某一部分。1998年10月DOM1级规范成为W3C的推荐标准，为基本的文档结构以及查询提供了接口。但是要注意，IE中的所有DOM对象都是以COM对象的形式实现的。这意味着IE中的DOM对象与原生JavaScript对象的行为或活动特点并不一致。

DOM可以将任何HTML或XML文档描绘成一个由多层节点构成的结构。节点分为几种不同的类型，每种类型分别表示文档中不同的信息或标记。每个节点拥有各自的特点，数据和方法，另外也有与其他节点存在某种关系。节点之间的关系构成了层次，所有页面标记则表现为一个以特定节点为根节点的树形结构。



◆ Node类型

DOM1级定义为一个Node接口，该接口将由DOM中的所有节点类型实现。除了IE之外，在其他所有浏览器中都可以访问到这个类型。javascript中所有的节点类型都继承自Node类型，所有节点类型都共享着相同的基本属性和方法。

- 节点关系

属性：

- ✓ **nodeType** 表示节点类型

Document--> 9;Element -->1;TextNode -->3;Comment--> 8

document 是Document构造函数的实例

document.body 是Element构造函数的实例

document.body.firstChild 是Comment构造函数的实例

- ✓ **nodeName**

该属性取决于节点类型，如果是元素类型，值为元素的标签名



文档对象模型

✓ **nodeValue**

该属性取决于节点类型，如果是元素类型，值有null

✓ **childNodes**

属性，保存一个**NodeList**对象，**NodeList**是一种类数组对象用来保存一组有序的节点，**NodeList**是基于**DOM**结构动态执行查询的结果，**DOM**结构变化可以自动反应到**NodeList**对象中。访问时可以通过中括号访问，也可以通过**item()**方法访问。可以使用**slice**方法将**NodeList**转换为数组

```
var arr = Array.prototype.slice.call(nodes,0);
```

✓ **parentNode**

指向文档树中的父节点。包含在**childNodes**列表中所有的节点都具有相同的父节点，每个节点之间都是同胞/兄弟节点。

✓ **previousSibling** 兄弟节点中的前一个节点

✓ **nextSibling** 兄弟节点中的下一个节点

✓ **firstChild** **childNodes**列表中的第一个节点



✓ **lastChild**

childNodes列表中的最后一个节点

✓ **ownerDocument**

指向表示整个文档的文档节点。任何节点都属于它所在的文档，任何节点都不能同时存在于两个或更多个文档中。

◆ 方法:

✓ **hasChildNodes()**

在包含一个或多个子节点的情况下返回true

◆ 操作节点

以下四个方法都需要父节点对象进行调用!

✓ **appendChild()**

向**childNodes**列表末尾添加一个节点。返回新增的节点。关系更新如果参数节点已经为文档的一部分，位置更新而不插入，**dom**树可以看做是由一系列的指针连接起来的，任何**DOM**节点不能同时出现在文档中的多个位置

✓ **insertBefore()** //第一个参数：要插入的节点；第二个参数：作为参照的节点；

被插入的节点会变成参照节点的前一个同胞节点,同时被方法返回。如果第二个参数为**null**将会将该节点追加在**NodeList**后面

✓ **replaceChild()** //第一个参数：要插入的节点；第二个参数：要替换的节点；

要替换的节点将由这个方法返回并从文档树中被移除，同时由要插入的节点占据其位置

✓ **removeChild()** //一个参数，即要移除的节点。

移除的节点将作为方法的返回值。其他方法,任何节点对象都可以调用。



◆ 其他方法

✓ **cloneNode()**

用于创建调用这个方法的节点的一个完全相同的副本。有一个参数为布尔类型参数为 `true` 时，表示深复制，即复制节点以及整个子节点数。参数为 `false` 的时候，表示浅复制，只复制节点本身。该方法不会复制添加到DOM节点中的JavaScript属性，例如事件处理程序等。该方法只复制特定子节点，其他一切都不复制。但是IE中可以复制，建议标准相同，在复制之前，移除所有事件处理程序。

✓ **normalize()**

处理文档树中的文本节点，由于解析器的实现或DOM操作等原因，可能会出现文本节点不包含文本，或者接连出现两个文本节点，当在某个节点上调用了该方法，会删除空白节点，会找到相邻的两个文本节点，并将他们合并为一个文本节点。

文档对象模型

◆ Document类型

Javascript通过使用**Document**类型表示文档。在浏览器中，**document**对象是**HTMLDocument**的一个实例，表示整个**HTML**页面。**document**对象是**window**对象的一个属性，因此可以直接调用。**HTMLDocument**继承自**Document**。

● 文档子节点

可以继承**Node**中所有的属性和方法

属性：

| | |
|------------------------|---|
| documentElement | 始终指向HTML页面中的<html>元素。 |
| body | 直接指向<body>元素 |
| doctype | 访问<!DOCTYPE>, 浏览器支持不一致, 很少使用 |
| title | 获取文档的标题 |
| URL | 取得完整的URL |
| domain | 取得域名, 并且可以进行设置, 在跨域访问中经常会用到。服务器测 |
| referrer | 取得链接到当前页面的那个页面的URL, 即来源页面的URL。 |
| images | 获取所有的img对象, 返回 HTMLCollection 类数组对象 |
| forms | 获取所有的form对象, 返回 HTMLCollection 类数组对象 |
| links | 获取文档中所有带 href 属性的<a>元素 |



文档对象模型

- 查找元素

getElementById()

参数为要取得元素的**ID**，如果找到返回该元素，否则返回**null**。如果页面中多个元素的**ID**值相同，只返回文档中第一次出现的元素。如果某个表单元素的**name**值等于指定的**ID**，该元素也会被匹配。

getElementsByName()

参数为要取得元素的标签名，返回包含另一个或者多个元素的**NodeList**，在HTML文档中该方法返回的是**HTMLCollection**对象，与**NodeList**非常类似。可以通过**[index/name],item(),namedItem(name)**访问

getElementsByName()

参数为元素的**name**，返回符合条件的**HTMLCollection**

getElementsByClassName()

参数为一个字符串，可以由多个空格隔开的标识符组成。当元素的**class**属性值包含所有指定的标识符时才匹配。**HTML**元素的**class**属性值是一个以空格隔开的列表，可以为空或包含多个标识符。



◆ Element类型

● HTML元素

所有的HTML元素都由**HTMLElement**类型表示，或者其子类型表示。每个HTML元素都应具有如下一些属性以及html元素特有的属性。

| | |
|------------------------|-------------------------------------|
| <code>id</code> | 元素在文档中的唯一标识符 |
| <code>title</code> | 有关元素的附加说明信息 |
| <code>className</code> | 与元素 <code>class</code> 特性对应 |
| <code>src</code> | <code>img</code> 元素具有的属性 |
| <code>alt</code> | <code>img</code> 元素具有的属性 |
| <code>lang</code> | 元素内容的语言代码，很少使用！ |
| <code>dir</code> | 语言方向， <code>ltr,rtl</code> 左到右，右到左、 |

每个元素都有一个或者多个特性，这些特性的用途是给出相应元素或内容的附加信息。可以通过属性访问到该属性对应的值,特性的名称是不区分大小写的，即"`id`" "`ID`" 表示相同的特性，另外需要注意的是，根据**HTML5**规范，自定义特性应该加上`data-`前缀，以便验证。



文档对象模型

✓ 取得自定义属性

`getAttribute()` 参数为实际元素的属性名，`class, name, id, title, lang, dir`一般只有在取得自定义特性值的情况下，才会用该方法。大多数直接使用属性进行访问，比如`style, onclick`

✓ 设置属性

`dom.className = "one"`

`dom.setAttribute("className","one");`

`setAttribute()`：两个参数，第一个参数为要设置的特性名，第二个参数为对应的值。如果该值存在，替换

✓ 移除属性 `removeAttribute()` 移除指定的特性

✓ **attributes**属性，其中包含了一个`NamedNodeMap`,与`NodeList`类似

`getNamedItem(name)` 返回`nodeName`属性等于`name`的节点

`removeNamedItem(name)` 从列表中删除`nodeName`属性等于`name`的值

`setNamedItem(node)` 向列表中添加一个节点

`item(pos)` 返回位于数字`pos`位置处的节点



文档对象模型

✓ 创建元素

`document.createElement()` :一个参数，要创建元素的标签名。该标签名在HTML中不区分大小写，但是在XML中区分大小写

✓ 元素的子节点

```
<ul>  
    <li>item1</li>  
    <li>item2</li>  
</ul>
```

ie8及以下版本浏览器 2个子节点

其他浏览器 5个子节点

✓ 特殊特性

style 通过`getAttribute()`访问时，返回的**style**特性值中包含的是CSS文本，而通过属性来访问返回一个对象，由于**style**属性是用于以编程方式访问元素样式的，因此并没有直接映射到**style**特性

onclick类似的事件处理程序 通过`getAttribute()`访问时，返回相应代码字符串；访问`onclick`属性时，返回一个javascript函数



文档对象模型

- 作为文档树的文档

将文档看做是**Element**对象树，忽略文档**Text, Comment**节点。**Element**中的属性

children 类似于**childNodes**, 返回**NodeList**对象，但是该对象中仅包含**Element**对象

firstElementChild 第一个孩子元素节点

lastElementChild 最后一个孩子元素节点

nextElementSibling 下一个兄弟元素节点

previousElementSibling 上一个兄弟元素节点

childElementCount 子元素的数量，返回值和**children.length**值相等

- 元素内容

innerHTML 返回元素内容

textContent 元素内部的文本，不去除空格和回车

innerText 元素内部的文本，去除回车和换行

文档对象模型

◆ Text类型： 文本类型

文本节点。包含的是可以按照字面解释的纯文本内容。

| | |
|------------------------------------|--------------------------------|
| length | //文本长度 |
| appendData(text) | //追加文本 |
| deleteData(beginIndex,count) | //删除文本 |
| insertData(beginIndex,text) | //插入文本 |
| replaceData(beginIndex,count,text) | //替换文本 |
| splitText(beginIndex) | //从beginIndex位置将当前文本节点分成两个文本节点 |
| document.createTextNode() | //创建文本节点，参数为要插入节点中的文本 |
| substringData(beginIndex,count) | //从beginIndex开始提取count个子字符串 |

◆ Comment类型： 注释类型

```
<div id = "myDiv"><!--a comment--></div>
```

<!--a comment--> Comment类型



杰普软件科技有限公司

www.briup.com

电邮: training@briup.com

主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层

邮编: 215311

电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区

邮编: 200436

电话: 021-56778147

Briup High-End IT Training

第 10 章:事件

Brighten Your Way And Raise You Up.

事件

javascript与HTML之间的交互是通过事件实现的。事件就是文档或浏览器窗口中发生的一些特定的交互瞬间。

事件三要素：

- ✓ 事件目标（event target）

发生的事件与之相关联或与之相关的对象

- ✓ 事件处理程序（event handler）

处理或相应事件的函数

- ✓ 事件对象（event object）

与特定事件相关且包含有关该事件详细信息的对象



事件

◆ 事件流

描述的是从页面中接受事件的顺序

● 事件冒泡 (IE事件流)

- 从内往外

事件开始由最具体的元素接收，然后逐级向上传播到不具体的节点

```
<html>
    <head></head>
    <body>
        <div>click me</div>
    </body>
</html>
```

当点击了

元素，这个click事件会按照如下顺序传播

div->body->html->document

➤ 注意：IE8以及更早版本只支持事件冒泡。



● 事件捕获 (**Netscape**事件流)

不太具体的节点更早接收事件，具体的节点到最后接收事件。当点击了<div>元素，按照如下方式触发click事件

document->html->body->div

● DOM事件流

“DOM2级事件”规定了事件流包括三个阶段：事件捕获阶段，处理目标阶段和事件冒泡阶段。首先发生的是事件捕获，为截获事件提供了机会。然后是实际的目标接收事件。最后是事件冒泡。

事件捕获： document->html->body

处理目标： 事件处理

事件冒泡： div->body->html->document

◆ 事件处理程序

事件就是用户或浏览器自身执行的某种动作，响应某个事件的函数为事件处理程序，事件处理程序以"on"开头(`onclick, onload`)

● HTML事件处理程序

某个元素支持的每种事件，都可以使用一个与相应事件处理程序同名的HTML特性来指定。这个特性的值应该是能够执行的JavaScript代码。

```
<input type="button" value="clickMe" onclick = "alert('is clicked')">  
<input type="button" value="clickMe" onclick = "showMsg()">  
<script type="text/javascript">  
    function showMsg(){  
        alert("is clicked");  
    }  
</script>
```

点击按钮会调用`showMsg()`函数，事件处理程序的代码在执行时，有权访问全局作用域的任何代码。

缺点：1)时差问题，用户可能会在HTML元素以出现在页面上就触发相应的事件，但当时的事件处理程序有可能尚不具备执行的条件。2)这种扩展事件处理程序的作用域链在不同浏览器中会导致不同结果。3)HTML与JavaScript代码紧密耦合。



事件

● DOM0级事件处理程序

通过**javascript**指定事件处理程序的传统方式，将一个函数赋值给一个事件处理程序属性。特点是简单，跨浏览器。

```
var btn = document.getElementById("btn");
btn.onclick = function(){
    alert('cliked');
}
```

dom0级方法制定的事件处理程序被认为是元素的方法，因此这个时候时间处理程序是在元素的作用域中运行，**this**指向当前元素。

```
btn.onclick = null;           //删除事件处理程序
```

● DOM2级事件处理程序

DOM2级规范以一种符合逻辑的方式来标准化**DOM**事件，**IE9, Firefox, Opera, Safari, Chrome**全部已经实现了"**DOM2**级事件"模块的核心部分。**IE8**是最后一个仍然使用其专有事件系统的主要浏览器

事件

- 非IE事件处理程序

addEventListener() 事件绑定

参数：

要绑定的事件名

作为事件处理的函数

布尔值： **true**在捕获阶段调用事件处理程序； **false**在冒泡阶段调用

removeEventListener() 事件删除

参数：

要删除的事件名

作为事件处理的函数

布尔值： **true**在捕获阶段调用事件处理程序； **false**在冒泡阶段调用

可以添加多个事件处理程序，并且按照添加她们的顺序触发。移除事件传入的参数与添加处理程序时使用的参数相同，添加事件时如果使用匿名函数将无法删除



● IE事件处理程序

事件处理程序会在全局作用域中运行，因此this指向window对象。为一个对象添加两个相同的事件，事件处理程序的顺序是按照添加相反顺序进行处理

attachEvent() 事件绑定

参数：

事件处理程序名称

事件处理函数

detachEvent() 事件移除

参数：

事件处理程序名称

事件处理函数

事件处理程序都被添加到冒泡阶段

事件

◆ 事件对象

● DOM中的事件对象

在触发DOM上的某个事件时，会产生一个事件对象event,这个对象包含着所有与事件相关的信息，包括导致事件的元素，事件的类型以及其他与特定事件相关的信息。兼容DOM的浏览器默认会将event对象传入到事件处理函数中。

```
dom.onclick = function(event){  
    console.log(event);  
}  
dom.addEventListener("click",function(event){  
    console.log(event);  
},false);
```

➤ 事件对象的属性均为只读属性

事件

| 属性 | 类型 | 说明 |
|---------------------------|---------|-----------------------------|
| bubbles | Boolean | 事件是否冒泡 |
| cancelable | Boolean | 是否可取消事件默认行为 |
| currentTarget | Element | 事件处理程序当前正在处理事件的那个元素 |
| eventPhase | Integer | 调用事件处理程序的阶段;1捕获 2处于目标 3冒泡 |
| target | Element | 事件真正目标 |
| type | String | 事件类型，需要一个函数，处理多个事件时，可使用该属性。 |
| preventDefault() Function | | 取消事件的默认行为 |
| stopPropagation()Function | | 取消事件的进一步捕获或者冒泡 |

- 在事件处理程序内部，对象this始终等于currentTarget值，而target则只包含事件的实际目标。如果直接将事件处理程序指定给了目标元素，this,currentTarget,target包含相同的值。



事件

● IE中的事件对象

在使用DOM0级方法添加事件时，`event`对象可以作为`window`对象的一个属性存在，使用`attachEvent`添加事件处理程序的时候，`event`对象会作为参数传入事件处理函数中

```
dom.onclick = function(){
    console.log(window.event);
    window.event.returnValue = false;//阻止默认行为
    window.event.cancelBubble = true;//取消冒泡
}
dom.attachEvent("onclick",function(event){
    console.log(window.event);
});
```

| 属性 | 类型 | 说明 |
|---------------------------|----------------------|--|
| <code>cancelBubble</code> | <code>Boolean</code> | 是否取消事件冒泡,值为 <code>true</code> 取消冒泡,类似 <code>stopPropagation()</code> |
| <code>returnValue</code> | <code>Boolean</code> | 取消时间默认行为,值为 <code>false</code> 阻止,类似 <code>preventDefault()</code> |
| <code>srcEvent</code> | <code>Element</code> | 事件的目标 <code>target</code> |
| <code>type</code> | <code>String</code> | 被触发的事件 的类型 |



事件

◆ 事件类型

● UI事件

load

当页面完全加载后在window上触发，当所有框架加载完时在框架集上触发，当图像加载完毕时在img元素上触发，当嵌入的内容加载完时在<object>触发

unload

当页面完全卸载后在window上触发，当所有框架都卸载后在框架集上触发，当嵌入的内容卸载完毕后再<object>上触发,(firefox不支持)

select

当用户选择文本框（<input>,<textarea>）中的一个或多个字符时

resize

当浏览器窗口被调整到一个新的高度或者宽度时，会触发

scroll

当用户滚动带滚动条的元素中的内容时，在该元素上触发resize,scroll会在变化期间重复被激发，尽量保持代码简单



事件

● 焦点事件

blur 元素失去焦点的时候触发

focus 元素获得焦点的时候触发，不支持冒泡

//IE支持

focusin 与**focus**等价，支持冒泡

focusout 与**blur**等价，支持冒泡

● 鼠标与滚轮事件

click

点击主鼠标按钮或者按下回车按键的时候触发。只有在一个元素上相继发生
mousedown,**mouseup**事件，才会触发**click**事件

dblclick

双击主鼠标按钮时触发.只有在一个元素上相继触发两次**click**时间才会触发
dbclick事件

事件

| | |
|------------|---|
| mousedown | 任意鼠标按钮按下时触发 |
| mouseup | 释放鼠标按钮触发 |
| mousemove | 鼠标在元素内部移动的时候重发触发 |
| mousewheel | 滚轮事件 |
| mouseenter | 鼠标光标从元素外部首次移动到元素范围内激发，不冒泡。 【不支持子元素】 |
| mouseleave | 在位于元素上方的鼠标光标移动到元素范围之外时触发， 不冒泡【不支持子元素】 |
| mouseover | 鼠标位于元素外部，将其首次移入另一个元素边界之内时触发 【支持子元素】 |
| mouseout | 在位于元素上方的鼠标光标移入到另外一个元素中。 【支持子元素】在被选元素上与mouseleave效果相同 |



- 键盘与文本事件

keydown 按下键盘任意键时触发，如果按住不放会重复触发此事件

keypress 按下键盘字符键时触发，如果按住不放会重复触发此事件

keyup 释放键盘上键时触发

当键盘事件发生时，**event**对象的**keyCode**属性中会包含一个代码与键盘上的特定键对应，对数字字母键，**keyCode**属性的值与ASCII码中对应的小写字母和数字编码相同

事件

● 相关元素, **event**特殊属性

✓ 客户区坐标位置

clientX,clientY 事件发生时，鼠标指针在视口中的水平和垂直坐标位置

✓ 页面坐标位置

pageX,pageY 事件发生时，鼠标指针在页面本身而非视口的坐标，页面没有滚动的时候，**pageX**和**pageY**的值与**clientX**和**clientY**值相等

✓ 屏幕位置 **screenX,screenY**

✓ 修改键

值为**boolean**类型，用来判断对应的按键是否被按下**shiftKey**, **ctrlKey**, **altKey**, **metaKey**

✓ 鼠标按钮

mousedown,mouseup，该事件的**event**对象中包含了**button**属性，表示按下或释放的按钮。

0表示主鼠标按钮

1表示中间的滚动按钮

2表示次鼠标按钮





杰普软件科技有限公司
www.briup.com

电邮: training@briup.com
主页: <http://www.briup.com>

昆山地址: 昆山市巴城镇学院路828号昆山浦东软件园北楼4、5、8层
邮编: 215311
电话: 0512-50190298

上海地址: 上海市闸北区万荣路1188弄G栋102室-上海服务外包科技园龙软园区
邮编: 200436
电话: 021-56778147

Briup High-End IT Training

第 11 章: 浏览器对象模型

Brighten Your Way And Raise You Up.

◆ 间歇调用和超时调用

javascript是单线程语言，但是可以通过超时值和间歇时间来调度代码在特定时刻执行

● **setTimeout()**

该方法返回一个数值ID，表示超时调用，这个超时调用ID是计划执行代码的唯一标识符通过它来取消超时调用。可以通过**clearTimeout(ID);**

参数：

- 1.要执行的代码
- 2.以毫秒表示的时间

例如：一秒后调用

```
var id = setTimeout(function(){  
    alert(1000);  
},1000);  
console.log(id);  
clearTimeout(id) //清除
```



- **setInterval()**

按照指定的时间间隔重复执行代码，直到间歇调用被取消或页面被卸载。调用该方法也会返回一个间歇调用ID，该ID可以让用户在将来某个时刻取消间歇调用

参数：

- 1.要执行的代码
- 2.以毫秒表示的时间

```
clearInterval(ID); //取消间歇调用
```



- 使用超时调用来模拟间歇调用

```
var num = 0;  
var max = 10;  
function incrementNum(){  
    num++;  
    if(num < max){  
        alert(num);  
        setTimeout(incrementNum,500);  
    }else{  
        alert("Done"+num);  
    }  
}  
setTimeout(incrementNum,500);
```

- **setTimeout, setInterval**配合完成调用函数

```
function invoke(f,start,interval,end){  
    if(!start){start = 0;}  
    if(arguments.length<=2){  
        setTimeout(f,start);  
    }else {  
        function repeat(){  
            var h = setInterval(f,interval);  
            if(end){  
                setTimeout(function(){  
                    clearInterval(h);  
                },end);  
            }  
        }  
        setTimeout(repeat,start);  
    }  
}
```



◆ 系统对话框

`alert()`,`confirm()`,`prompt()`方法可以调用系统对话框向用户显示消息。显示这些对话框的时候代码会停止执行，关掉这些对话框后代码又会恢复执行。

- `alert()` 该方法接受一个字符串并将其显示给用户。该对话框会包含指定的文本和一个"OK"按钮。主要用来显示警告信息
- `confirm()` 确认对话框，显示包含指定的文本和一个"OK"按钮以及"Cancel"按钮。该方法返回布尔值，`true`表示单击了OK，`false`表示单击cancel或者关闭按钮
- `prompt()` 会话框，提示用户输入一些文本。显示包含文本，`ok`按钮,`cancel`按钮以及一个文本输入域，以供用户在其中输入内容。传入两个参数，要显示给用户的文本提示和文本输入域的默认值。

如果用户单击OK按钮，该方法返回输入域的值，如果用户单击了Cancel或者关闭对话框该方法返回`null`



◆ location对象

是最有用的BOM对象之一，提供了与当前窗口中加载的文档有关的信息，还提供一些导航功能。location是个神奇的对象，既是window的对象也是document的对象。

```
console.log(window.location == document.location); //true
```

属性：

| | |
|----------|------------------------|
| host | 返回服务器名称和端口号 |
| hostname | 返回不带端口号的服务器名称 |
| href | 返回当前加载页面的完整URL |
| pathname | 返回URL的目录和文件名 |
| port | 返回URL中指定的端口号 |
| protocol | 返回页面使用的协议 |
| search | 返回URL的查询字符串。这个字符串以问号开头 |



方法:

- | | |
|-----------|--|
| assign() | 传递一个url参数， 打开新url， 并在浏览记录中生成一条记录。 |
| replace() | 参数为一个url,结果会导致浏览器位置改变，但不会在历史记录中生成新记录 |
| reload() | 重新加载当前显示的页面， 参数可以为boolean类型， 默认为false， 表示以最有效方式重新加载， 可能从缓存中直接加载。如果参数为true， 强制从服务器中重新加载 |

为location.href; window.location 设置为一个URL值， 也会以该值调用assign()方法。以下三句话效果一样

```
window.location="http://www.baidu.com";  
location.href="http://www.baidu.com"  
location.assign("http://www.baidu.com");
```

◆ history对象

该对象保存着用户上网的历史记录。出于安全方面的考虑，开发人员无法得知用户浏览过的URL，不过借由用户访问过的页面列表，同样可以在不知道实际URL的情况下实现后退前进，注意：没有应用于History对象的公开标准，不过所有浏览器都支持该对象。

`length` 返回历史列表中的网址数

注意：IE和Opera从0开始，而Firefox、Chrome和Safari从1开始。

`back()` 加载 history 列表中的前一个 URL

`forward()` 加载 history 列表中的下一个 URL

`go()` 加载 history 列表中的某个具体页面

负数表示向后跳转，正数表示向前跳转

