JavaScript对象

Array数组对象

数组对象的构建

字面量方式

```
var b = [1,true,2];
```

new方式

通过数组的构造函数**Array()**来创建一个数组对象。

```
new Array();
new Array(size);
new Array(element0, element1, ..., elementN);
```

使用无参数构造函数创建数组时会返回一个空数组，数组长度为0。

例子：

```
var goodsTypes = new Array();
goodsTypes[0]="男装";
window.alert(goodsTypes[0]);
```

注意：

当把构造函数当作函数调用，不使用new运算符时，其行为与使用new运算符调用时的行为完全一致。

数组对象的属性

Array对象的属性包括constructor、length和prototype。

属性	描述
constructor	返回对创建此对象的构造函数的引用
length	数组的长度
prototype	对对象添加属性和方法

例子：

```
var movies=new Array("分歧者2:绝地反击","疯狂的麦克斯：狂暴之路",
                "复仇者联盟2:奥创纪元","飓风营救4");

if(movies.constructor==String){
        alert("movies是个字符串对象");
}else if(movies.constructor==Array){
       alert("movies是个数组对象，数组中元素的个数是："+movies.length
        +"\n------------------------\n"+movies.constructor);
}else if(movies.constructor==Date){
        alert("movies是个日期对象");
    }
```

数组对象的常用方法

方 法	描 述
concat()	用于连接两个或多个数组
join()	用于把数组中的所有元素放入一个字符串，并用指定的分隔符隔开
push()	可向数组的末尾添加一个或多个元素，并返回新的长度
pop()	用于删除并返回数组的最后一个元素
shift()	用于删除并返回数组的第一个元素
reverse()	在原有数组的基础上，颠倒数组中元素的顺序，不会创建新的数组
slice()	可从已有的数组中返回选定的元素
sort()	用于对数组的元素进行排序
splice()	向数组中添加或删除一个或多个元素，然后返回被删除的元素

concat()方法

concat()方法用于连接两个或多个数组，返回合并后的新数组，而原数组保持不变。

```
arrayObject.concat(param1, param2, ......, paramX)
```

- 多个参数之间使用逗号（,）隔开；
- concat()方法返回的是合并后的新数组，原数组保持不变。

join()方法

join()方法用于把数组中的所有元素放入一个字符串中，并通过指定的分隔符隔开。

```
arrayObject.join(separator);
```

- separator是可选的，作为数组元素之间的分隔符，默认为逗号。
- join()方法的返回形式是字符串。

slice()方法

slice()方法用于从数组中返回选定的元素。

```
arrayObject.slice(start,[end]);
```

- 参数start是必需的，表示元素选取的开始位置；
- 参数end可选，表示元素选取的结束位置（不包括end）；**当参数end省略时，将选取从start开始到数组末尾的所有元素；**
- start和end允许取负数；-1表示字符串的最后一个字符，-2表示倒数第二个字符，其他以此类推。

splice()方法

splice()方法用于向数组中添加1~n个元素或从数组中删除元素。

```
arrayObject.splice(index,howmany,[item1,.....,itemX]);
```

- 参数index必须，规定添加或删除元素的位置。当为负数时从末尾计数；
- 参数howmany必需，表示要删除元素的数量，0代表不删除数据；
- 参数列表item1, ..., itemX可选，表示向数组中添加或替换的新元素；
- 该方法在原数组基础上实现，不会生成新的副本数组。

二维数组

在JavaScript中，没有二维或多维数组，不能通过new Array()()方式来创建二维数组。
通过在一维数组中存放另一个数组，来模拟实现二维数组。

数组中可以包含不同的数据类型。

```
var array=new Array();
array[0]=new Array("科幻","《2012》",80);
array[1]=["爱情","《何以笙箫默》",6];
array[2]="从1992年到2012年这二十年是本次太阳纪的最后一个周期…";
array[3]=88;
```

String字符串对象

字符串的创建

字符串对象的创建有以下两种方式：字面量方式和new方式

字面量方式

在Javascript中，可以隐式地将一个字符串转换成字符串对象。

```
var name="漫步时尚广场";       //类型为string类型
var address='中国·青岛·高新区';         //类型为string类型
```

使用单引号（'）或双引号（"）均可生成一个字符串。

new方式

new方式创建字符串对象是通过调用**String()**构造函数来完成，并返回一个String对象。

```
var movieName=new String("何以笙箫默");  //类型为String对象
var director=String("刘俊杰");     //类型为string类型
```

在JavaScript中，string和String的区别如下：

- String是string的包装类；
- string是一种基本的数据类型，没有提供substring()等方法；
- String是构造函数用于创建字符串对象，使用new创建的对象具有substring()等方法；
- string没有提供prototype原型对象，而String对象具有prototype原型对象，通过浏览器的端点调试方式进行查看该区别；
- 使用typeof()函数查看类型时，string变量返回“string”，String对象返回“Object”，而String返回“function”；
- 使用= =比较时，string类型判断其值是否相等，而String对象则判断是否对同一对象进行引用；
- 二者的生命周期不同，使用new创建的对象一直存在，而string类型自动生成的会在代码执行后立即销毁。

字符串对象的方法

方法	描 述
indexOf(searchValue,[fromIndex])	返回searchValue在字符串中首次出现的位置
lastIndexOf(searchValue,[fromIndex])	从后向前进行检索，返回searchValue在字符串中首次出现的位置
slice(start,[end])	抽取从start开始（包括start）到end结束（不包括end）为止的所有字符
substring(start,[stop])	抽取从start处到stop-1处的所有字符
split()	用于把一个字符串分割成字符串数组

indexOf()方法

indexOf()方法用于检索子串在字符串中首次出现的位置。

```
stringObject.indexOf(searchValue,[fromIndex])
```

- 参数searchValue表示被检索的子串
- 参数fromIndex可选，表示字符串开始检索的位置，取值范围0~stringObject.length-1，默认值为0
- 当检索到子串时，返回子串在字符串中的位置；否则返回-1
- 字符串的开始下标是从0开始的

lastIndexOf()方法

lastIndexOf()方法用于**从后向前**对字符串进行检索，并返回子串在字符串中首次出现的位置。

slice()方法

slice()方法用于从字符串中抽取一部分内容。

```
stringObject.slice(start,[end])
```

- 抽取范围从start位置开始（包括start）到end结束（不包括end）
- 参数start必选，表示要抽取子串的起始下标
- 参数end可选，表示要抽取子串的结束下标；当参数end省略时，表示提取范围从start位置到字符串的结尾
- start和end允许取负数；-1表示字符串的最后一个字符，-2表示倒数第二个字符，其他以此类推

substring()方法

```
stringObject.substring(start,[stop])
```

- substring()方法与slice()方法相似，也是从字符串中抽取一部分
- 与slice()方法不同，substring()方法不接受负参数

split()方法

split()方法用于把一个字符串分割成一个字符串数组。

```
stringObject.split(separator,[howmany])
```

- 参数separator（必需）是一个字符串或正则表达式，使用该参数指定的规则对字符串进行分割；
- 参数howmany可选，用于指定返回的数组的最大长度；当参数howmany存在时，返回的子串个数不应大于howmany；当参数howmany省略时，对整个字符串进行分割，而不考虑结果的数量

转义字符

转义字符	实现方式	转义字符	实现方式
双引号	"	换行	\n
单引号	'	回车	\r
Tab	\t	反斜杠	\
退格	\b	换页符	\f

Date日期对象

JavaScript通过日期对象（Date）来操作日期和时间。通过日期对象的构造函数创建一个系统当前时间或指定时间的日期对象。

```
var myDate1=new Date();
var myDate2=new Date(1218253167595);
var myDate3=new Date(2015,9,2);

var myDate4=new Date(2015,9,2,12,08,16);
var myDate5=new Date("9/25/2015 9:25:38");
var myDate6=new Date("April 25,2048");
var myDate7=new Date("Apr 25,2048 17:32:16");
```

方法	描 述
getDate()	返回一个月中的某一天(1~31)
getDay()	返回一周中的某一天(0~6)
getMonth()	返回月份(0~11)
getFullYear()	返回4位数字的年份
getHours()	返回Date对象的小时(0~23)
getMinutes()	返回Date对象的分钟(0~59)
getSeconds()	返回Date对象的秒数(0~59)
getTime()	返回1970年1月1日至今的毫秒数
setXxx()	用于设置日期对象的年月日等信息

常用方法

setFullYear()

setFullYear()方法用于设置年份（包括月份和日期）。

```
dateObject.setFullYear(year[,month[,day]])
```

setHours()

setHours()方法用于设置指定时间的小时（包括分钟、秒、毫秒）。

```
dateObject.setHours(hour[,min,sec,millisec])
```

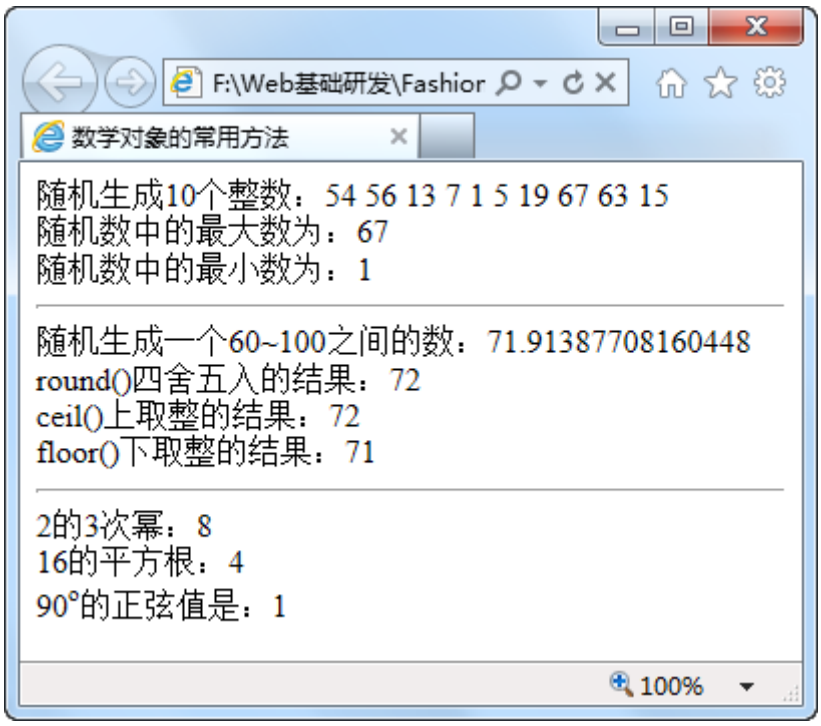
Math数学对象

数学对象（Math）提供了一些数学运算中的常数及数学计算方法，在数学运算时非常有用。
与String、Date不同，Math对象**没有提供构造方法**，可以直接使用Math对象。
在Math对象中，提供了一些常用的数学常数，如圆周率、自然对数的底数等。

属性	描 述
E	返回算术常量e，即自然对数的底数（约等于2.718）
LN2	返回2的自然对数（约等于0.693）
LN10	返回10的自然对数（约等于2.302）
LOG2E	返回以2为底的e的对数（约等于1.442）
LOG10E	返回以10为底的e的对数（约等于0.434）
PI	返回圆周率（约等于3.14159）
SORT2	返回2的平方根（约等于1.414）
SQRT1_2	返回2的平方根的倒数（约等于0.7071）

示例代码：

```
var num=10;
        document.write("随机生成"+num+"个整数：");
        var array=[];
        var maxNum=0;
        var minNum=0;
        for(var i=0;i<num;i++){
            var tmp=Math.random()*100
            array[i]=Math.floor(tmp);
            document.write(array[i]+"\t");
            if(i==0){
                maxNum=array[0];
                minNum=array[0];
            }else{
                maxNum=Math.max(maxNum,array[i]);
                minNum=Math.min(minNum,array[i]);
            }
        }
        document.write("<br/>随机数中的最大数为："+maxNum);
        document.write("<br/>随机数中的最小数为："+minNum);
        document.write("<hr/>");
        var randomNum=60+Math.random()*40;
        document.write("随机生成一个60~100之间的数："+randomNum);
        document.write("<br/>round()四舍五入的结果："+Math.round(randomNum));
        document.write("<br/>ceil()上取整的结果："+Math.ceil(randomNum));
        document.write("<br/>floor()下取整的结果："+Math.floor(randomNum));
        document.write("<hr/>");
        document.write("2的3次幂："+Math.pow(2,3));
        document.write("<br/>16的平方根："+Math.sqrt(16));
        document.write("<br/>90<sup>o</sup>的正弦值是："+Math.sin(90*Math.PI/180));
```



RegExp正则表达式对象

正则表达式（Regular Expression）最早出现于20世纪40年代，在数学与计算机科学理论中用于反映"正则性"的数学表达式特征。直到70年代末，才真正在程序设计领域中得到应用。

正则表达式是一种**字符串匹配的模式**，通过单个字符串来描述和匹配一系列符合某个句法的规则。JavaScript提供了一个RegExp对象来完成有关正则表达式的匹配功能。

正则表达式类似于通配符，是计算机帮助人们去匹配指定规则的字符串。

对象创建

创建一个RegExp对象有两种方式：直接量方式和构造函数方式。

```
var reg=/pattern/attributes;                              //直接量方式
var regExp=new RegExp(pattern,attributes);       //构造函数方式
```

- 参数pattern是一个字符串或表达式，表示正则表达式的模式；
- 参数attributes是一个可选的字符串，取值包括"g"、"i"和"m"，分别用于指定**全局匹配**、**区分大小写的匹配**和**多行匹配**。

常用元字符

正则表达式中的pattern部分可以包括**元字符**、**括号表达式**以及**量词**等。

元字符	描 述
.	用于查找单个字符，除了换行和行结束符
\w	匹配包括下划线的任何单词字符， 等价于[A-Za-z0-9_]
\W	匹配任何非单词字符，等价于[^A-Za-z0-9_]
\d	查找数字
\D	查找非数字字符
\s	查找空白字符
\S	查找非空白字符
\n	查找换行符
\r	查找回车符
\xxx	查找以八进制数xxx规定的字符
\xdd	查找以十六进制数dd规定的字符

表达式	描 述
[abc]	查找括号内的任意字符
[^abc]	查找除了括号内的其他任意字符
[0-9]	查找0-9之间的任意数字
[a-z]	查找a-z之间的任意字符
[A-Z]	查找A-Z之间的任意字符
[A-z]	查找A-z之间的任意字符
(boy|girl|baby)	查找括号内的某一项（或关系）

量词	描 述
n+	匹配任何包含至少一个n的字符串
n*	匹配任何包含零个或多个n的字符串
n?	匹配任何包含零个或一个n的字符串
n{x}	匹配包含x个n的序列的字符串
n{x,y}	匹配包含x或y个n的序列的字符串
n{x,}	匹配包含至少x个n的序列的字符串
n$	匹配任何结尾为n的字符串
^n	匹配任何开头为n的字符串
?=n	匹配任何其后紧接指定字符串n的字符串
?!n	匹配任何其后没有紧接指定字符串n的字符串

自定义对象

用户还可以自定义对象，定义对象有五种方式：

- 原始方式
- 构造函数方式
- 原型方式
- 混合方式
- JSON方式

原始方式

使用原始方式创建一个JavaScript对象，步骤如下：

```
var object = new Object();
object.porpertyName = value;
object.methodName = functionName | function(){};
```

- 使用Object类创建一个对象object；
- propertyName表示为object对象添加属性名
- methodName表示为object对象添加方法名

示例：

```
var goods=new Object();
goods.name="男士白领衬衣";
goods.type="男装";
goods.price="580";
goods.color="white";

goods.showInfo=function(){
    alert("商品名称："+goods.name+"\n商品类型："+goods.type+"\n商品价格："
        +goods.price+"\n商品颜色："+goods.color);
```

```
}

goods.showColor=showColor;
function showColor(){
    alert("商品颜色："+goods.color);
}
```

构造函数方式

通过构造函数创建一个JavaScript对象，步骤如下：

- 创建构造函数
- 用new运算符和构造函数来创建一个对象

```
function ClassName([param1][,param2]...){
    this.propertyName=value;
    //其他属性...
    this.methodName=functionName | function (){...};
    //其他方法...
}
```

示例：

```
//创建构造函数
function Goods(name,type,price,color){
}
Goods.name=name;
Goodstype=type;
Goods.price=price;
Goods.color=color;

Goods.showInfo=function(){
    alert("商品名称："+this.name+"\n商品类型："+this.type+"\n商品价格："
        +this.price+"\n商品颜色："+this.color);
};
//创建一个对象
var goods =new Goods("Nick","Shoes",1200,"White");
//方法的调用
goods.showInfo();
```

原型方式

```
object.prototype.name = value;
```

示例：

```
//创建构造函数
function Goods(){
}
Goods.prototype.name="耐克运动鞋";
Goods.prototype.type="鞋类";
```

```
Goods.prototype.price=1200;
Goods.prototype.color="白色";

Goods.prototype.showInfo=function(){
    alert("商品名称："+this.name+"\n商品类型："+this.type+"\n商品价格："
        +this.price+"\n商品颜色："+this.color);
};
//创建一个对象
var goods=new Goods();
//方法的调用
goods.showInfo();
```

混合方法

使用原型方式创建对象时，对象属性值采用默认值，在对象创建完成后再去改变属性的值。而构造方式在创建对象时，会重复生成方法所引用的函数。在实际应用中，将两者混合使用。

```
//创建构造函数
function Goods(name,type,price,color){
    this.name=name;
    this.type=type;
    this.price=price;
    this.color=color;
}
//原型方式添加方法
Goods.prototype.showInfo=function(){
    alert("商品名称："+this.name+"\n商品类型："+this.type+"\n商品价格："
        +this.price+"\n商品颜色："+this.color);
};
//创建对象实例
var goods1=new Goods("男士衬衣","男装",200,"白色");
var goods2=new Goods("女士花裙","女装",700,"红色");
//方法的调用
goods1.showInfo();
goods2.showInfo();
```

JSON方式

- JSON（JavaScript Object Notation）是一种基于ECMAScript的轻量级数据交换格式，采用完全独立于语言的文本格式，能够以更加简单的方式来创建对象
- 使用JSON方式无需构造函数和new关键字，直接创建所需的JavaScript对象即可
- JSON对象是以“{”开始，以“}”结束，且属性与属性值成对出现

```
{
    //对象的属性部分
    propertyName:value,
    //对象的方法部分
    methodName:function(){...}
};
```

示例：

```
//创建对象实例
var goods={
    name:"男士衬衣",
    type:"男装",
    price:200,
    color:"白色",
    showInfo:function(){
        alert("商品名称："+this.name+"\n商品类型："+this.type+"\n商品价格："
            +this.price+"\n商品颜色："+this.color);
    },
    showColor:function(){
        alert("商品颜色："+this.color);
    }
};
//方法的调用
goods.showInfo();
```

在数据传输过程中，JSON数据往往以字符串的形式进行传输，所以在页面中需要通过

- 1.将JSON字符串转成JSON对象前，需在JSON字符串两侧添加一对括号“（）”
- 2.使用eval()方法将该字符串强制转换成JSON对象

```
//JSON字符串
var movieStr='{'
    +'name:"小时代",'
    +'type:"爱情",'
    +'price:80,'
    +'showInfo:function(){'
           +'document.write("影片名称："+this.name+",影片类型："+this.type+",票
价："+this.price);'
    +'}'
+'}';
//eval()转换
var movie=eval("("+movieStr+")");
//对象方法的调用
movie.showInfo();
```

Function对象方式

```
//JSON字符串
var movieStr='{'
    +'name:"小时代",'
    +'type:"爱情",'
    +'price:80,'
    +'showInfo:function(){'
               +'document.write("影片名称："+this.name+",影片类型："+this.type+",票
价："+this.price);'
    +'}'
+'}';
//1.Function对象方式，自调用函数的方式
var movie=(new Function("","return "+movieStr))();
movie.showInfo();
//2.分解写法，与上面写法完全等价
var movieFunction=new Function("","return "+movieStr);
var otherMovie=movieFunction();
otherMovie.showInfo();
```