# 基本格式

可以将 JavaScript 代码放置在 HTML 文档的任何地方。但放置的地方，会对 JavaScript 代码的正常执行会有一定影响

## 1.1放置于<head></head>之间

由于 HTML 文档是由浏览器从上到下依次载入的，将 JavaScript 代码放置于<head></head> 标签之间，可以确保在需要使用脚本之前，它已经被载入了。

eg：

<html>  
<head>  
<script type="text/javascript">  
……  
JavaScript 代码  
……  
</script>  
</head>  
....

## 1.2放置于<body></body>之间

如果我们有一段 JavaScript 代码需要操作 HTML 元素。但由于 HTML 文档是由浏览器从上到下依次载入的，为避免 JavaScript 代码操作 HTML 元素时，HTML 元素还未载入而报错（对象不存在），因此需要将这段代码写到 HTML 元素后面

## 1.3外部引用JavaScript代码

将 JavaScript 代码（不包括<script></script>标签）单独形成一个文档，并以 js 后缀命名，如 myscript.js ，并在 HTML 文档 <script></script> 标签中使用 src 属性来引用该文件

eg：

<html>  
<head>  
<script type="text/javascript" src="myscript.js"></script>  
</head>  
....

**注意**：如果 HTML 文档声明为 XHTML ，<script></script> 标签必须在 CDATA 部分内声明，否则 XHTML 将把 <script></script> 标签解析为另一个 XML 标签，里面的 JavaScript 代码可能不会正常执行。

**eg：**  
<html>  
<head>  
<script type="text/javascript">  
<![CDATA[  
JavaScript 代码  
]]>  
</script>  
</head>  
....

# 变量

## 2.1变量声明

javascript的所有变量声明都使用var进行声明，声明不指定变量类型，但变量有自己的变量声明，eg：

<head>

<title>helloWorld</title>

<meta charset="utf-8">

<script type="text/javascript">

var num\_1=1;

var num\_2=1;

var result=num\_1+num\_1;

window.alert("结果是"+result);

</script>

</head>

**注意！！！不使用var定义的变量默认为全局变量，如：**

**function f() {**

**i = 1;**

**};**

**f();**

**alert(i);**

**这里的i被声明为全局变量，调用函数之后，创建的i还可以被外部访问到**

## 2.2数据类型

javascript是弱数据类型脚本语言，即在定义变量时，可以不声明其变量类型，其实际的变量类型由javascript引擎决定，如：

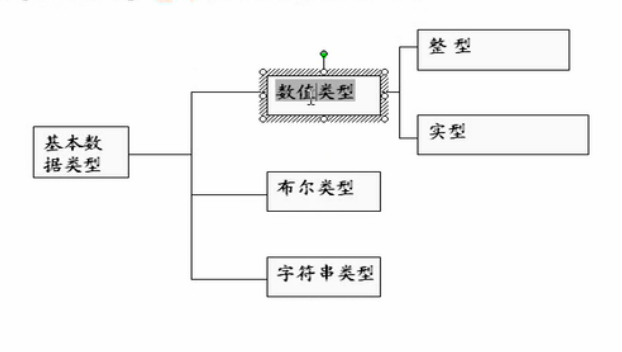
var str=”hello!”;//str是字符串类型

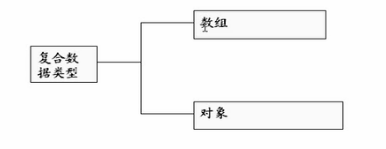
var num=2;//num是数值类型

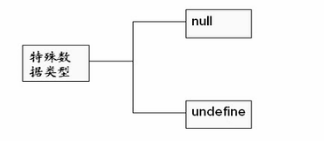
var und;//und是undefined

name=23;//此时name变为数值类型

**javascript的三大数据类型：**







整型

整型的特殊类型有：NaN(not a number)表示不是一个数，Infinity表示无穷大

eg var num\_2="abc";

window.alert(parseInt(num\_2));



window.alert(-2/0);



使用isNaN()和isFinite()**(是否为有限的)**来判断是一个数是否不是数或者不是无穷大，当是参数是NaN（非数字），或者是正、负无穷大的数，isFinite()返回 false，若参数是有理数，则返回true

## 2.3类型检测

**注意：**

**== 等同，===恒等。**

**==， 两边值类型不同的时候，要先进行类型转换，再比较。   
===，不做类型转换，类型不同的一定不等。**

**2.3.1.typeof可以帮助我们查看变量的数据类型。**

eg：

<script type="text/javascript">

var num\_1=1;

var num\_2="abc";

window.alert(typeof(num\_1));

window.alert(typeof num\_2);

</script>

**注意：typeof null 返回”object”**

**2.3.2.instanceof可以帮助我们查看对象属于哪个构造器。**

**语法:**

**obj instanceof function;**

instanceof通过查找function是否在object的原型链上进行判断，eg：

[1,0] instanceof Array;

# 数组和字符串

### **3.1数组概述**

对于java，{}代表一个无元素的数组，对于javascript，[]代表无元素数组，数组长度最长为2^23-1

只需使用下一个未用的索引，任何时刻可以不断向数组增加新元素

语法：

var myarray= new Array(8); //创建数组，存储8个数据。其实只是把myarray的length设为8而已，用0 in myarray;来看，该索引0并不存在

var myarray = new Array(66,80,90,77,59);//创建数组同时赋值

var myarray = [66,80,90,77,59];//直接输入一个数组（称 “字面量数组”）

**注意：如果数组未赋初值，输出显示undefined，若使用了数组长度之外的元素，也是可以使用的**

和JAVA一样，javascript的数组也具有length属性，该属性也是可变的，即：arr.length=10;是合法的，随数组元素增加，其长度也是变化的

javascript也有二维数组，要分开声明

eg：

var myarr=new Array(); //先声明一维

for(var i=0;i<2;i++){ //一维长度为2

myarr[i]=new Array(); //再声明二维

}

**注意：用in操作符对数组进行操作时，index in obj，第一个参数是数组下标，in对对象操作时，第一个参数是对象属性的key。**

### **3.2数组函数操作**

**push(var)**方法，在数组的尾部添加一个元素var，该函数返回数组长度

**eg: arr.push(3);**

**unshift(var)**方法，在数组头部添加一个元素var，该函数返回数组长度

**eg：var arr=[1,2,3];**

**arr.unshift(0);//[0,1,2,3]**

**pop()**方法，移除数组的最后一个元素，该函数返回值为该元素

**eg：var arr=[1,2,3];**

**document.write(arr.pop());//输出3**

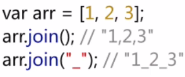
**shift()**方法，删除数组的第一个元素，该函数返回被删除的第一个元素

**eg：var arr = [1, 2, 3];**

**document.write(arr.push(4)); //输出4**

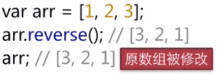
**join(var)**方法，将数组转化为字符串，

**eg：**



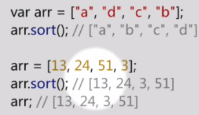
**reverse()**方法，将数组颠倒顺序，返回颠倒后的数组，需要注意的是原数组也会被修改

**eg：**



**sort()**方法，将数组按字符排序，该方法也会修改原数组，是原址操作

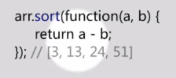
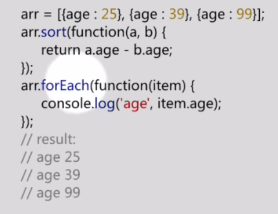
**eg：**



第二个例子中先把数字转化为了字符串，然后再排序，所以顺序为13,24,3,51

sort方法有个可选参数，该参数必须是个有两个参数的返回值为正或负数值的函数，当函数返回一个正值，就交换两个参数在数组中的位置。实际上是冒泡排序。

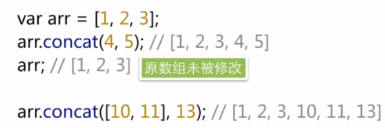
**eg：**

将函数改为b-a之后则变为从大到小排序

**concat()**方法，合并数组，只能通过返回值获取新数组

**eg：**



### **3.3字符串操作**

**1、字符串分割split()**

**split(char，[length])**

**第二个参数可选，第一个参数是用于分割的字符，第二个表示返回的字符串数组的最大长度eg：**

**var myStr = "I,Love,You,Do,you,love,me";**

**var substrArray = myStr .split(",");**

**// ["I", "Love", "You", "Do", "you", "love", "me"];**

**var arrayLimited = myStr .split(",", 3);**

**// ["I", "Love", "You"];**

**2字符串长度length属性**

**eg：**

alert(str.length);

# 语句和运算符

### **4.1javascript中不存在{}块语句**

也就是说：

for(var i=0;i<arr.length;i++){}这个循环结束后，i的值仍然存在，不会被销毁，不存在块语句，当然也就不存在块中的局部变量。但要注意，一般来说，只有if和循环使用的{}是块语句，函数使用的{}不是，因此函数的{}在任何语言中都不是块语句(不受此影响)，也就是说

function fun(){

var a=1;

alert(“helloworld!”);

}

alert(typeof a); //输出undefined

### 4.2注意不要写var a=b=1;这样的语句,如

function fun() {

var a = b = 1;

}

fun();

alert(typeof a); //输出undefined,因为a使用了var声明，是局部变量不可被外部访问

alert(typeof b);//b没有被var声明，是全局变量

### 4.3JavaScript也有try-catch-finally语句块

### 4.4with语句

**with用来引用某个特定对象中已有的属性，但是不能用来给对象添加属性**

eg：

document.forms[].name.value = "lee king";

document.forms[].address.value = "Peking";

document.forms[].zipcode.value = "";

可以简化为：

with(document.forms[]){

name.value = "lee king";

address.value = "Peking";

zipcode.value = "";

}

with语句可以简化代码，减少代码量，但由于每次执行前先要检查对象中是否有相应属性，降低了运行效率，不建议使用，严格模式禁止出现该段代码

### 4.5delete运算符

在JavaScript中，delete运算符通常用于删除对象的属性，但它只能**删除属性的引用**，它并不能像c++那样删除属性引用指向的对象，只能解除引用对该对象的指向，并将引用赋值为undefined，但一般由于没有任何引用指向的对象将会被作为垃圾回收，所以也就等于变相删除了对象。

**(1)delete的返回值**

当被delete的对象的属性存在并且拥有DontDelete时 返回false，否则返回true。 这里的一个特点就是，对象属性不存在时也返回true，所以**返回值并非完全等同于删除成功与否。**

**(2)无法被删除的属性**

**所有被prototype声明的属性都不能被删除,但对他们用delete会返回true**

**对象的预定义属性也无法删除。 可以认为这类属性带有DontDelete的特性**

**eg：**

function F() {

this.a = 1;

};

F.prototype.b = 2;

var f = new F();

document.write("delete a的返回值:"+delete f.a+"<br>");

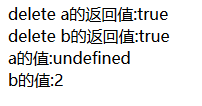
document.write("delete b的返回值:" + delete f.b + "<br>");

//这样就可以删除了

document.write("delete b的返回值:" + delete f.\_\_proto\_\_.b + "<br>");

document.write("a的值:" + f.a + "<br>");

document.write("b的值:" + f.b + "<br>");



**注意：Javascript 在执行脚本之前 会创建一个Global对象，所有的全局变量都是这个Global对象的属性， 执行函数时也会创建一个Activation对象，所有的局部变量都是这个Activation对象的属性**

**(3)无法被删除的变量**

**通过var声明的变量和通过function声明的函数拥有DontDelete特性，无法被删除。没有被var声明的变量可以被删除且返回值为true**

1. **严格模式**

严格模式是一种代码执行模式，它修复了语言上的不足，提供更强的错误检查，并增强安全性。

我们可以通过在函数中加**'use strict';**来实现严格模式的启动，也可以在js文件的最开头加**'use strict';**，指定整个文件用严格模式执行

严格模式和普通模式的区别：

1. **不允许使用with语句，否则报错**
2. **定义变量必须使用var，防止定义了不被期望出现的全局变量(不加var声明出来的变量都是全局变量)，否则报错**
3. **arguments变为静态副本**

在正常模式下，可以通过每个函数内部都可以使用的arguments参数列表数组来访问函数的参数，如：

function f(a) {

alert(arguments[0]);

};

f(2);

这里的arguments[0]在正常模式下是与形参a绑定的，即改变任何一个，另一个也跟着改变

**需要注意的是如果不给f传参，即f();这时改变arguments[0]的值也没用，a依然是undefined**

但在严格模式下，arguments不存在这种绑定关系，arguments只是参数列表的一个静态副本，**但如果参数传入的是一个对象的引用，仍然可以通过arguments访问该对象并对该对象进行修改**

1. **正常模式下delete参数和函数没有什么意义，会返回false，但严格模式下则会报错，严格模式下delete不可配置的属性也会报错**
2. **通过对象字面量方式定义类时，若属性名重复，严格模式下会报错，正常模式下只会使用最后一次定义的值，eg：**

var obj = {

a: 2,

b: 3,

a:1

}

alert(obj.a);//正常模式下值为1

1. **严格模式下eval和arguments变为关键字，不能作为变量和函数名，同时严格模式还禁止出现八进制字面量**

eg: var obj = {

a: 2,

}

alert(0743); //正常模式下值为483

1. **在严格模式下，eval将拥有独立作用域eg：**

eval(‘var myEval=3;’);

alert(myEval); //正常模式下myEval有值，严格模式下为undefined

1. **严格模式下一般函数调用时(即不是对象函数调用，也不实用apply/call/bind修改this)，this指向null，而不是全局变量**
2. **试图修改不可写属性或在不可拓展对象上添加属性，报错而不是忽略**

# 5.函数

### **5.1函数对象**

**函数也是对象，可以有自己的方法和属性，但要注意，如果只是声明函数，而没有new创建和调用，是不产生函数对象的，在未创建函数前，不通过函数原型链添加的属性不属于该函数，而应该属于函数的局部变量**

语法：

function 函数名(参数列表)

{

代码

}

**注意：参数列表直接写出参数即可，不能加var！**

**eg：**

**function getMax(var num\_1,var num\_2){**

**}//错误**

**function getMax(num\_1,num\_2){**

**}**

eg：

function sayHello() {

alert("hello");

}

sayHello.alternate = function() {

alert("hi");

}

sayHello(); //输出 "hello"

sayHello.alternate(); //输出 "hi"

**注意：若函数的return语句中返回了别的函数，在使用构造器new创建该函数对象时，实际创建的是另一个函数，eg：**

**function fun() { }**

**fun.prototype.y = 4;**

**function foo() {**

**return new fun();**

**}**

**foo.prototype.x = 1;**

**var obj = new foo();**

**document.write(obj.x);//返回undefined，若为obj.y返回4**

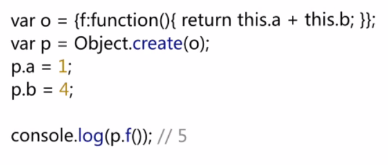
### **5.2this指针**

在全局环境下，this指向BOM的window对象

在一般函数中，this指向全局对象，在js中是window对象，在node.js中是 global

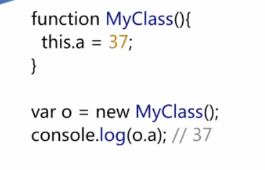
当函数作为对象的方法时，this指向该对象

当函数处于对象的原型链上时，this仍指向该对象，eg：



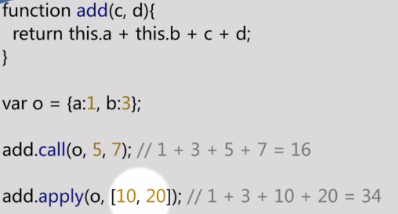
这里的o中只有f函数，p的原型链指向o，p上有两个属性a，b，通过p调用o，o的函数中的this仍指向p

构造器中的this，声明函数时并未创建对象，此时并没有this，用new创建函数对象，此时this指向创建的对象

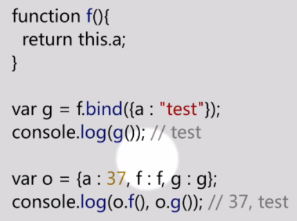


### **5.3函数对象的方法**

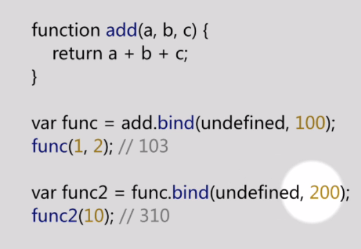
call()方法和apply()方法，都可以给函数传入this的值，区别是call()方法是将函数的参数列表作为参数传入，而apply是将函数的参数列表作为数组传入，eg：



上述方法每次调用时都需要重新绑定this，还有一个bind()方法只需要绑定一次this，可重复使用，每次使用的this都指向上一次通过bind()绑定的this。eg：



bind()方法后面还有几个可选参数，如果加上就是给函数添加默认参数，eg：



默认a=100,b=200

### **5.4函数属性**

function foo(x,y,z){

arguments.length;//实际传参个数,只传了1和3，所以这里是2

}

foo(1,3);

foo.length;//获取形参个数，这里是3

foo.name;//返回函数的名称，这里是foo

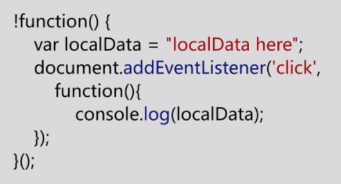
**注意：这里有几个实参arguments数组就有多长，若只像上面那样传入两个实参，然后强行赋值arguments[2] = 9;是不行的，z依然是undefined**

**严格模式下不可修改**

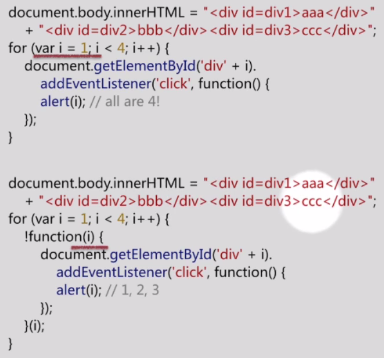
### **5.5函数闭包**

JavaScript支持闭包，最常见的是在一个函数中定义了局部变量，该局部变量本该在函数调用后被销毁，但是通过将本函数中的子函数传出的方法调用能够调用这个本该销毁的变量

eg：



总的来说，就是一个本该被销毁的函数，有一部分与外面还有关联，就不会将其销毁，所依一般退出函数时，会将不用的局部变量全部删除(将其置为null，delete没用)



上述例子中，第一个不存在闭包，由于for循环没有块语句，所以循环结束i还存在，为4，点击之后结果都是4，第二个给循环代码加了个函数，巧妙运用了闭包，每次执行循环体，都将i传入，分别为1,2,3，每次执行函数，由于匿名function被外部点击事件绑定，所以函数执行(添加监听)完之后不会将i删除，而是保存在内存之中，点击触发运行匿名函数时，就是使用了匿名函数的父函数的形参i

### **5.6回调函数**

回调函数是一个作为变量传递给另外一个函数的函数，它在主体函数执行完之后执行。作为参数传入的函数是回调函数

代码如下：

 function a(callbackFunction){  
   alert("这是parent函数a");  
   var m =1;  
   var n=3;  
  return callbackFunction(m,n);  
 }  
 function b(m,n){  
   alert("这是回调函数B");  
   return m+n;  
 }  
 $(function(){  
   var result = a(b);  
   alert("result = "+ result);  
 });

执行顺序为：

这是parent函数a

这是回调函数B

result = 4

**函数首先执行了主题函数a，之后调用了回调函数b，最后返回函数a的返回值。**

# 6.for in声明

语法：

for(index in array){

代码

}

for in类似于java的foreach，但又有区别，二者都可遍历数组的所有元素，但for in是通过下标遍历，foreach是直接遍历

eg：

var x

var mycars = new Array()

mycars[0] = "宝马"

mycars[1] = "奔驰"

mycars[2] = "宾利"

for (x in mycars)

{

document.write(mycars[x] + "<br />")

}

for in使用需要谨慎，比如对getElementsByTagName返回的数组进行for in时，返回的数组除了该有的元素，还有item()和namedItem()以及length等属性，所以会多循环三次

**注意： for in的输出顺序不固定，它不是按照创建对象属性的顺序访问的，所以要求按顺序遍历时建议不要使用for in而使用普通for**，eg：

var obj = {

"first": "first",

"zoo": "zoo",

"2": "2",

"34": "34",

"1": "1",

"second": "second"

};//用对象字面量方式创建对象obj

for (var i in obj) {

alert( obj[i]);

};

# 7.对象

## 7.1对象概述

### 7.1.1对象字面量

对于javascript来说，一切都是对象，包括基本数据类型和函数，对象中包含一系列的**无序**属性，每个属性都有一个字符串key(属性名)和对应的value(值),在给出key之后，JavaScript会对key进行toString处理然后再最为属性名

eg: var obj={x:1,y:2};

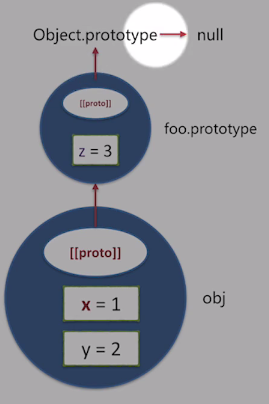
obj[{}]=3;

alert(obj[{}]);//输出3

### 7.1.2对象原型链prototype

每个函数对象内部除了正常属性外，还有一个prototype对象(只有函数有prototype，对象有一个**\_ \_proto\_ \_指向**构造函数的prototype)可以通过给这个prototype对象添加属性来给obj添加属性，在访问类的属性时，优先访问正常属性，即图中的x，y；然后再去prototype中查找，也即是当prototype和正常属性出现重名时，正常属性会覆盖prototype

需要注意的是，若添加代码obj.z=4;将会有有两个obj，一个是obj的属性，一个是原型链上的属性，若添加obj.z=undefined并不会删除该属性，只是将其赋值为undefined，需要通过delete obj.z来删除该属性，此时原型链上的z属性将可见



function foo(){}//函数也是对象

foo.prototype.z = 1;

var obj = new foo();

obj.x = 2;

obj.y = 3;

对于上述代码，首先声明(未创建)了一个foo函数的类，它拥有原型链，然后通过原型链添加了z属性，接着创建了foo的实例obj，最后给obj添加x和y属性，这样做，会使obj的原型链\_\_proto\_\_指向foo的原型链prototype，foo的原型链指向所有类的父类Object对象，Object的\_\_proto\_\_指向null，因此一般到foo的原型链就停止了

## 7.2创建对象

### 7.2.1对象字面量

对象字面量是一个名值对列表，每个名值对之间用逗号分隔，并用一个大括号括起。各名值对表示对象的一个属性，名和值这两部分之间用一个冒号分隔，名的部分可以加””也可以不加，因为JavaScript会对其进行一次toString，eg：

var obj = {

age: 21,

sex: “男”,

name:"DJ"

obj2:{

x:22,

y:32

}

};//该对象有四个属性，分别为x，1，name，obj2

### 7.2.2构造方法

function showFunction () {

alert(this.color);

};

function Car(sColor,iDoors,iMpg) {//函数本身即是一个对象

this.color = sColor;//函数对象添加color属性并赋值

this.doors = iDoors;

this.mpg = iMpg;

this.showColor = showFunction;

}

var oCar1 = new Car("red",4,23);

var oCar2 = new Car("blue",3,25);

### 7.2.3原型模式(构造器模式)

prototype中的属性和方法可以由各种特定类型的所有实例共享，eg:

function Car() {

}

var oCar1 = new Car();

oCar1.\_\_proto\_\_.color = "blue";

oCar1.\_\_proto\_\_.showColor = function () {

alert(this.color);

};

var oCar2 = new Car();

alert(oCar2.color);//结果blue

这种方式解决了前两种共享函数的问题，但它的构造函数没有参数，eg

function Car() {

}

Car.prototype.color = "blue";

Car.prototype.doors = 4;

Car.prototype.mpg = 25;

Car.prototype.showColor = function() {

alert(this.color);

};

var oCar1 = new Car();//只能使用无参构造函数

var oCar2 = new Car();

**注意：用new 函数名来创建的对象会执行该函数，如果函数有返回值，且返回值为对象，那么该new的返回值也为该对象，若函数返回值是数值或其他类型，仍相当于创建了一个该函数的对象，而不是该返回值的类型的对象，在构造器函数内部用var等方式创建的局部变量由于闭包的原因无法访问，如果构造器中是this.a=2;这样的语句，相当于给新创建的对象添加属性a=2;但要注意该语句不能保证每个该函数的对象都共享a属性，每个构造器创建的对象都有各自的a属性，改变其中一个，其他的并不改变，类似于static，想要共享，应使用prototype属性添加a，eg:**

function foo() {

var z=3;//通过该函数创建的对象中没有该属性

this.a = 1;

document.write("嘻嘻<br>");

return 1;

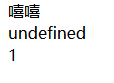
}

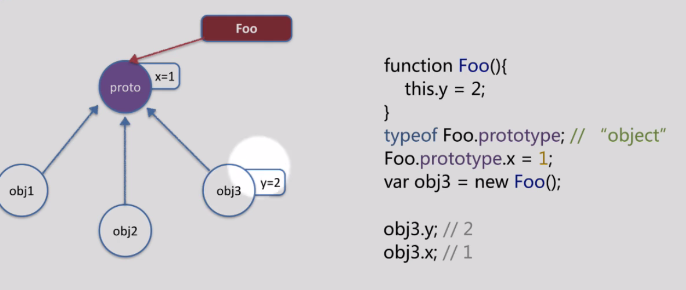
foo.prototype.x = 3;

foo1=new foo();

document.write(foo1.z + "<br>");

document.write(foo1.a + "<br>");



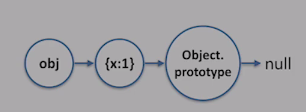


### 7.2.4Object.create()方法

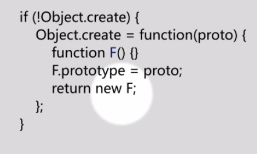
Object.create(obj)方法是系统内置的方法。它一般是创建一个空的对象，然后将对象的原型链指向参数obj，即obj是作为创建对象的原型链，eg：

var obj=Object.create({x:1});

obj.hasOwnPrototype(‘x’);//false



需要注意，create是ES5的方法，一些老版本浏览器可能不支持，所以我们可以对create进行模拟，eg：



### 7.2.5工厂方式

function showColor() {

alert(this.color);

}//为了保证同一个类的每一个对象都共享同一个函数

function createCar(sColor,iDoors,iMpg) {

var oTempCar = new Object;

oTempCar.color = sColor;

oTempCar.doors = iDoors;

oTempCar.mpg = iMpg;

oTempCar.showColor = showColor;

return oTempCar;

}

var oCar1 = createCar("red",4,23);

var oCar2 = createCar("blue",3,25);

oCar1.showColor(); //输出 "red"

oCar2.showColor(); //输出 " blue "

## 7.3属性操作

### 7.3.1正常读写

对象读写可以通过” . ”操作符或者” [] ”操作符，eg：

var obj={x:1,y:2};

obj.x;

obj[“y”];//多用于for循环，如for(i=0;i<2;i++){obj[]}

### 7.3.2异常读写

若对某对象中一个不存在的属性进行读取，会出现undefined，若对一个不存在属性进行写，会创建该属性，若属性存在，则可被隐式转化为true，若不存在，可被隐式转化为false，eg：

var obj={x:1,y:2};

if(obj&&obj.x){

alert(“x存在!”);

}

### 7.3.3属性删除

JavaScript属性删除一般使用delete关键字，可使用” . ”和” [] ”操作符，eg：

var obj={x:1,y:2};

delete obj.x;//true

delete obj[“y”]; //true

重复对一个属性进行delete，不进行任何操作，但该表达式仍返回true

### 7.3.4检测属性

JavaScript使用**in**关键字进行检测属性是否存在于对象内，每个对象内都内置了一个**Object.hasOwnPrototype()**方法，用于检测属性是否是对象自己的属性(即不在原型链上)，若是(不在原型链prototype上)，则返回true，否则返回false；**propertyIsEnumerable()**返回属性是否可枚举(只有可枚举的属性才能被for in访问，被Object.keys()和JSON.stringify()方法返回和访问)

**设置属性是否可被枚举**

Object提供了一个方法**Object.definePrototype(obj,pro,{enumerable:false,….})**方法，obj是被设置的属性的对象名，pro是被设置的属性名，第三个参数也是个对象，在里面设置属性的标签，若不设置，默认全为false(其他方法添加属性默认都为true)

eg: Object.definePrototype(cat,’price’,{enumerable:false,value:1000})

Object.definePrototype(foo.prototype,’z’,{get:function(){return 1;})

//设置z的getter方法

### 7.3.5 getter/setter方法

JavaScript对象提供getter/setter方法来按一定规则获取或设置属性，eg：

var obj={

name:”DJ”,

sex:”男”;

get age(){

return new Date().getFullYear()-1988;

}

set age(val){

alert(“age can not be set to”+val);

}

}

当获取obj.age时，调用get方法，当设置其值如：obj.age=3;时，调用set方法

## 7.4属性标签

属性标签描述了属性是否可枚举，是否可写等性质

### 7.4.1 查看属性标签

Object对象提供了**Object.getOwnPropertyDescriptor(对象名,属性名);**方法，用于输出属性的标签

### 7.4.2 属性标签

**writable代表属性是否可写**

**enumerable代表属性是否可被枚举**

**configurable代表属性是否可被再修改，如果该值为false，不可被删除不可被修改**

## 7.5对象标签

对象具有一个名为extensible标签，标明对象是否可被扩展添加新属性，通过Object.isExtensible(obj);方法可查看obj的extensible标签，Object.preventExtensions();方法可将extensible标签设为false，此时再添加属性会添加失败，不报错，但无该属性；Object.seal()方法不止将extensible设为false，还将每个属性的configurable标签设为false；Object.freeze()方法则在seal方法的基础上设置writable为false

**注意：这里的所有方法都只针对对象自己的属性，不会影响对象的原型链上的属性**

## 7.6对象序列化

### 7.6.1对象转换为数据

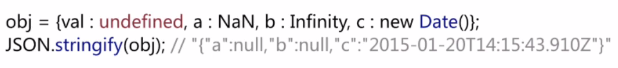
JSON对象提供了对象序列化功能，将对象进行相应处理后提交给后台，eg：

var f = { x: 1, y: 2 };

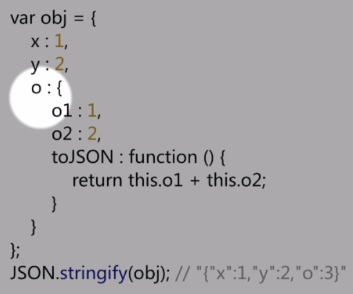
document.write(JSON.stringify(f));



注意：对象中值为undefined的属性在该方法处理后不出现在返回的字符串中和值为NaN、Indinity的属性处理后为null，eg

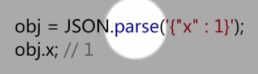


有时候可能有一些较为复杂的转化，因此我们可以自定义toJSON方法来定义JSON转化为字符串的规则，eg：



### 7.6.2后端数据转换为对象

通过JSON.parse()方法可将字符串的对象转化为object，eg：



## 7.7销毁对象

javascript有垃圾收集机制，当没有任何引用指向一个对象时，该对象在下次收集时就会被销毁，因此可以通过将对象的所有引用置为null来销毁对象

## 7.8访问权限

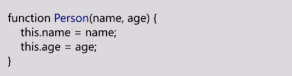
javascript所有属性和方法都是public，但有一个约定，约定前后加下划线的属性和方法应作为私有的使用，eg：

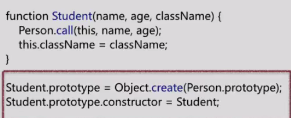
object.\_color\_=”red”;

javascript具有this指针

# 8.继承

## 8.1继承

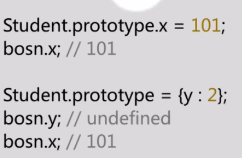




上述Student继承了Person，Person.call()方法将Student设置为子类的this指针，create方法创建了一个空的原型链指向Person的方法，这样再给Student添加函数(给类添加函数一般在原型链上添加，因为函数是共有的)时不会给父类Person也加上

**注意：对已创建的子类对象的类，修改其prototype的指向，不会对已创建的对象造成影响，eg：**

**var bosn=new Student(‘bosn’,20,’StudentName’);**



但当重新创建一个新的Student对象时，此时x为undefined，y=2

## 8.2实现继承的方式

function Person(){}

function Student(){}

方法一：

Student.prototype=new Person();//该方法用得少，因为如果是给Person方法本身(不是prototype)添加属性，该属性无法被Student访问，eg：

Student.prototype=new Person();

Person.x=1;

Person.prototype.y=2;

var stu=new Student();

document.write(stu.x);//undefined

document.write(stu.y);//2

方法二：

## 8.3apply()和call()

Function.apply(obj,args)和Function.call(obj,arg1,arg2….)都可以用来将function中的this指向参数obj，arg用于提供参数给function，不同的是当参数为数组形式时，应使用apply()，当参数为一个一个的时候，应使用call，根据这个特性，我们可以对Math.max()函数进行改进，max函数要求提供一个一个的参数，但有时我们需要对数组进行max操作，因此，我们可以Math.max.apply(null,array),将参数以数组形式提供给函数，但不改变this指向

# 9.事件



1. **onclick事件**

鼠标点击事件

1. **onmouseover事件**

鼠标经过事件，当鼠标移到一个对象上时，该对象就触发onmouseover事件

1. **onmouseout事件**

鼠标移开事件，当鼠标移开当前对象时，该对象就触发onmouseout事件

1. **onfocus事件**

当网页中的对象获得聚点时，执行onfocus调用的程序就会被执行

1. **onblur失焦事件**

onblur事件与onfocus是相对事件，当光标离开当前获得聚焦对象的时候，触发onblur事件

1. **onchange事件**

文本框内容改变事件，改变文本框的内容则触发onchange事件。

1. **onload事件**

事件会在页面加载完成后，立即发生，同时执行被调用的程序

**注意：该事件一般写在<body>内**

1. **onunload事件**

当用户退出页面时（页面关闭、页面刷新等），触发onUnload事件

**注意：不同浏览器对该事件的支持不同，最好不要使用**

**注意：可以通过调用this指针来对触发事件的html元素进行函数操作，eg：**

**<img onmouseover="bigImg(this)"/>**

# 10.javascript内置对象

## 10.1Date对象

日期对象可以储存任意一个日期，并且可以精确到毫秒

如：

var Udate=new Date();

var d = new Date(2012, 10, 1); //2012年10月1日

此时Udate已经是一个日期对象，且该对象的初始值为创建对象时的系统时间



**getDay()**返回星期，返回的是0-6的数字，0 表示星期天，可通过数组 var weekday=["星期日","星期一","星期二","星期三","星期四","星期五","星期六"];来获取相应的值。

**get/setTime()** 返回/设置时间，单位毫秒数，计算从 1970 年 1 月 1 日零时到日期对象所指的日期的毫秒数

## 10.2string字符串对象

1.**toUpperCase()**和**toLowerCase()**分别能将字符串转换为全大写和全小写

2.**charAt(index)** 方法可返回指定位置的字符。返回的字符是长度为 1 的字符串，若参数 index 不在 0 与 string.length-1 之间，该方法将返回一个空字符串

3. **indexOf(substring, startpos)**从startpos位置开始查找substring，如果没有此参数将从stringObject的开始位置查找，如果要检索的字符串值没有出现，则该方法返回 -1

4. **split(separator,limit)** 方法将字符串分割为数个子串组成的数组，并返回此数组，separator是分割字符，limit是可选参数，若设置该参数，则返回的数组长度最大为limit

**注意：如果把空字符串 ("") 用作 separator，那么 stringObject 中的每个字符之间都会被分割**

5.**substring(start，stop)** 方法用于提取字符串中介于两个指定下标之间的字符，返回的内容是从 start开始(包含start位置的字符)到 stop-1 处的所有字符，其长度为 stop 减start。

**注意：如果 start 比 stop 大，那么该方法在提取子串之前会先交换这两个参数。**

eg：字符串”I love JavaScript”,想获取love应给参数2,6

6.**substr(startPos,length)**从字符串中提取从 startPos位置开始的指定数目的字符串

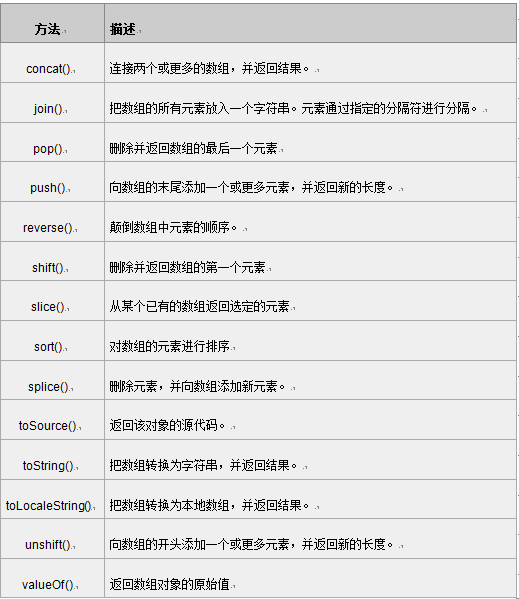
**注意：如果参数startPos是负数，从字符串的尾部开始算起的位置。也就是说，-1 指字符串中最后一个字符，-2 指倒数第二个字符，以此类推，如果startPos为负数且绝对值大于字符串长度，startPos为0**

## 10.3Math对象





## 10.4Array对象

****

1. **concat()** 方法用于连接两个或多个数组。此方法返回一个新数组，不改变原来的数组。

# 11.BOM浏览器对象模型

## 11.1window对象



### 计时器

计时器可以在设定的时间间隔之后来执行代码，分为一次性计时器和间隔性触发计时器



这里所有的set方法，后面的参数都为**(代码,交互时间)**，他们的返回值都是一个可以传递给对应的clear函数从而取消对"代码"的周期性执行的值。

eg：

var stopRetVal =window.setInterval("clock()",1000);//每隔1s调用一个名为clock()的函数

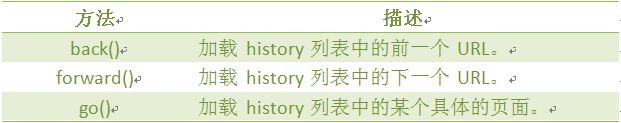
clearInterval(stopRetVal);//停止计时器

### History对象

history对象记录了用户曾经浏览过的页面(URL)，并可以实现浏览器前进与后退相似导航的功能

**注意:从浏览器被打开的那一刻开始记录，每个浏览器窗口、每个标签页乃至每个框架，都有自己的history对象与特定的window对象关联。**





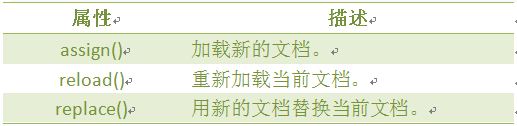
**back()**和**forward()**方法都是无参的函数，**go(number)**跳number个页面，若number为0，则为本页面，为1等同于forward()，为-1等同于back();

### Location对象

location用于获取或设置窗体的URL，并且可以用于解析URL，还可以对页面进行重定向





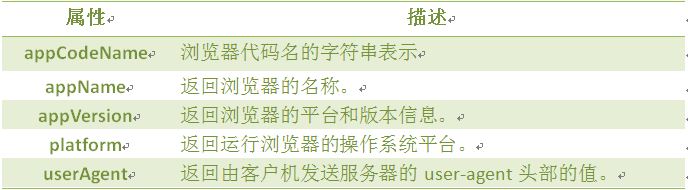


eg:

window.location.href;

### Navigator对象

Navigator 对象包含有关浏览器的信息，通常用于检测浏览器与操作系统的版本



userAgent返回用户代理头的字符串表示(就是包括浏览器版本信息等的字符串),可以通过该属性判断使用的浏览器，eg：

var u\_agent = navigator.userAgent;

var B\_name="Failed to identify the browser";

if(u\_agent.indexOf("Firefox")>-1){

B\_name="Firefox";

}else if(u\_agent.indexOf("Chrome")>-1){

B\_name="Chrome";

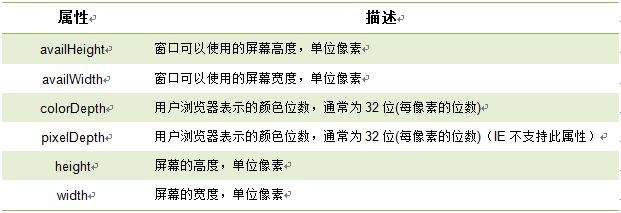
}else if(u\_agent.indexOf("MSIE")>-1&&u\_agent.indexOf("Trident")>-1){

B\_name="IE(8-10)";

}

### screen对象

screen对象用于获取用户的屏幕信息



screen.availWidth属性和screen.availHeight属性返回可用的屏幕的宽度和高度，以像素计，减去界面特性，比如任务栏。

### 消息框

1.警告框

语法：

alert("文本");

eg：alert("我是警告框！！");

1. 验证框

语法：

confirm(“文本”);

该函数具有返回值，当用户点击确认，返回值为 true，若点击取消，返回值为 false。

eg：

var r=confirm("Press a button!");

if (r==true)

{

alert("You pressed OK!");

}

else

{

alert("You pressed Cancel!");

}

1. 提问框

语法：

prompt("文本","默认值");

提示框通常具有输入功能，其函数具有返回值，若用户点击确认，那么返回值为输入的值。如果用户点击取消，那么返回值为 null

eg:

var name=prompt("请输入您的名字","Bill Gates")

if (name!=null && name!="")

{

document.write("你好！" + name + " 今天过得怎么样？")

}

# 12.核心DOM(DOM树操作)

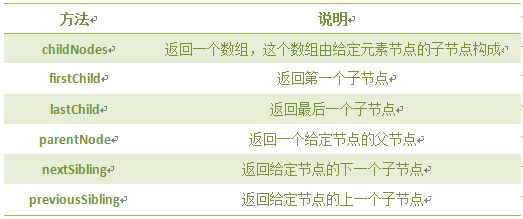
DOM 将HTML文档呈现为带有元素、属性和文本的树结构（节点树），即HTML 文档中的所有内容都是节点，整个文档是一个文档节点，每个 HTML 元素是元素节点，HTML 元素内的文本是文本节点，每个 HTML 属性是属性节点，注释是注释节点，如：



**节点属性**



**遍历节点树:**





### 12.1获取元素节点

**语法:**

**document.getElementById(“id”) //通过ID获取元素**

**document.getElementsByName(“name”) //该方法获取的是数组，因为一个文档中可以存在许多相同name的元素**

**getElementsByTagName(Tagname) //该Tagname是标签的名称，如p、a、img等标签名。同样是返回一个数组**

**注意：通过标签名获取时，标签名作为参数不区分大小写**

eg:

<p id="con">JavaScript</p>

var mychar=document.getElementById("con");

**注意：也可以通过某个元素调用该方法，这样就是在某个元素的后代元素中查找符合的元素**

**eg：parent. getElementsByClassName(“myClass”);**

### 12.2节点属性

1.**元素节点**属性查询和修改的方法

**elementNode.getAttribute(name)**方法获取元素节点的属性值，elementNode是使用getElementById()等方法，获取到的元素节点，name是想查询的属性的名字。

**elementNode.setAttribute(name,value)** 方法增加一个指定名称和值的新属性，或者把一个现有的属性设定为指定的值。

**eg：**(document.getElementsByTagName("li") )[0].getAttribute("title")；

2.**节点**重要属性

nodeName : 节点的名称，只读

//元素节点和属性节点的nodeName与标签名相同

//文本节点的 nodeName 永远是 #text

//文档节点的 nodeName 永远是 #document

nodeValue ：节点的值

//元素节点的 nodeValue 是 undefined 或 null  
// 文本节点的 nodeValue 是文本自身  
// 属性节点的 nodeValue 是属性的值

nodeType ：节点的类型, 只读

**元素类型    节点类型**  
  元素          1  
  属性          2  
  文本          3  
  注释          8  
  文档          9

### 12.3访问相近的DOM树节点

**1.elementNode.childNodes**//访问选定元素节点下的所有子节点的列表，返回的值可以看作是一个数组，他具有length属性

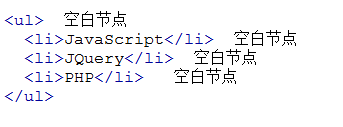
**注意：如果选定的节点没有子节点，则该属性返回不包含节点的 NodeList**



**UL子节点个数:3**

**节点类型:1**

**元素节点和元素节点之间有空白符，所以显示为 7，如下图所示:**



**即默认元素与元素标签之间有空白节点，但文本与元素标签之间没有，比如2<li></li>之间没有**

1. **firstChild和lastChild**

效果等同于用childNodes获取第一个子节点和最后一个子节点。

**注意，我们需要通过检测节点类型，过滤空白子节点。**

1. **parentNode访问父节点**

**语法：**

**elementNode.parentNode**

**注意:父节点只能有一个。**

1. **nextSibling属性访问兄弟节点，previousSibling 属性可返回某个节点之前紧跟的节点（处于同一树层级中）**

如果无此节点，则该属性返回 null，同样，该属性也可能获取空白节点，所以要注意过滤空白节点

1. **createElement(nodeName)创建节点appendChild()插入节点**

createElement(nodeName)创建新节点，nodeName是节点名，如：”p”,”div”

appendChild()在指定节点的最后一个子节点列表之后添加一个新的子节点

**eg：**

var newNode = document.createElement("li");

newNode.innerHTML = "PHP";

otest.appendChild(newNode);

1. **插入节点insertBefore()，删除节点removeChild()**

insertBefore() 方法可在父节点的某个子节点前插入一个新的子节点。

**语法:**

**父节点.insertBefore(newnode,node);**

eg：

fatherNode.insertBefore(newNode, otest.childNodes[1]);

removeChild() 方法从子节点列表中删除某个节点。如删除成功，此方法可返回被删除的节点，如失败，则返回 NULL。

**语法:**

**父节点.removeChild(node)**

**注意:** 把被删除的子节点赋值给 x，这个子节点不在DOM树中，但是还存在内存中，可通过 x 操作。如果想彻底销毁该节点，只需将其赋值为null

**还要注意别把**removeChild写成removeNode;

1. **替换元素节点replaceChild()**

replaceChild 实现子节点(对象)的替换。返回被替换对象的引用。

**语法：**

**父节点.replaceChild (newNode,oldNode )**

1. **创建文本节点createTextNode**

**语法：**

**document.createTextNode(Text)**

createTextNode() 方法创建新的文本节点,一个html节点下可能会有文本节点，如<a>百度</a>的文本节点是”百度”

1. **浏览器窗口可视区域大小**

**window.innerHeight - 浏览器窗口的内部高度**

**window.innerWidth - 浏览器窗口的内部宽度**

1. **网页尺寸**
2. **offset**

**offset共有五个属性：**

**(1)offsetWidth和offsetHeight**

只读属性，这两个属性没有兼容性问题，他们描述的是当前html元素的宽和高，包括其边框，如图，表示的是这个div的宽度，包括橘色的内部和灰色的边框



#test {

width:400px;

height:200px;

padding:20px;

background:#F60;

border:10px solid #888;

}

其test.offsetWidth为width(400)+左右padding(20+20)+左右border(10+10)=460px

**注意：offsetWidth与offsetHeight的值只与该元素有关，与周围元素（父级和子级元素无关）**

**(2)offsetParent**

只读属性，该属性没有兼容性问题，offsetParent属性返回一个对象的引用，这个对象是距离调用offsetParent的元素最近的已进行过CSS定位(position的值为relative或者absolute)的容器元素。 如果这个容器元素未进行CSS定位, 则offsetParent属性的取值为根元素(一般为body)的引用。

**1、如果当前元素的父级元素没有进行CSS定位（position为absolute或relative），offsetParent为body。**

**2、如果当前元素的父级元素或某个祖先元素有CSS定位（position为absolute或relative），offsetParent取最近的那个元素**。

eg:

<div style="width:400px;height:200px;background-color:gray;position:absolute;">

<div style="width:200px;height:100px;background-color:yellow;">

<span id="ss" style="color:red;">2342</span>

</div>

</div>

ss的offsetParent为宽400，高200的div

**(3)offsetTop和offsetLeft**

只读元素，且属性值与offsetParent有关，返回对象元素边界的左上角顶点相对于offsetParent的左上角顶点的垂直和水平**偏移量，该属性有兼容性问题**

**在火狐中，其值为该元素的offsetParent元素的内边框到该元素的外边框的距离，IE8以上的版本中也一样，但当offsetParent标签元素为html时，二者有差异**

**scrollHeight**

crollHeight和scrollWidth，获取y轴滚动条的高度和x轴滚动条的宽度, scrollHeight和scrollWidth还可获取Dom元素中内容实际占用的高度和宽度。该属性有浏览器兼容性问题

**document.documentElement.scrollWidth**

**document.documentElement.scrollHeight**

1. **网页尺寸**

**(1)offset**

**offsetWidth是指标签元素(如div)的宽度（包括div的边框）**

**offsetHeight是指标签元素(如div)的高度（包括div的边框）**

**offsetLeft是指标签元素(如div)到整个页面左边框的距离（不包括div的边框）**

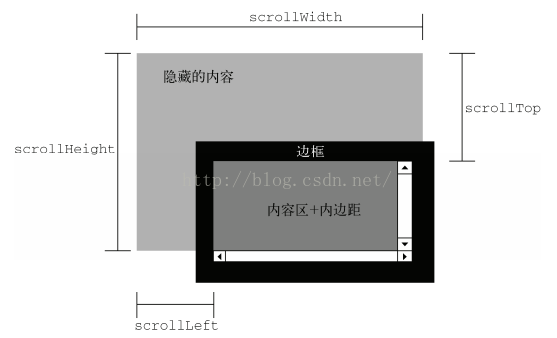
**offsetTop是指标签元素(如div)到整个页面上边框的距离（不包括div的边框）**

offset是html元素的实际宽高，包括内容物宽高，padding撑大的宽高和border宽高

**(2)scroll滚动**

**scrollHeight和scrollWidth 对象内部实际内容的高度/宽度，包括内容区和内边距，不包括边框（隐藏的内容是用滚动条查看的）**

**scrollTop和scrollLeft 被卷去部分的 顶部/左侧 到可视区域 顶部/左侧 的距离**



存在兼容问题：document.body.scrollHeight||document.documentElement. scrollHeight

document.body.scrollTop||document.documentElement.scrollTop

**(3)client   可视区、客户端**

**clientWidth和clientHight一般用来测量可视区域大小，即内容物+padding，不包括边框和滚动条，目前只发现这两个属性能测div，其他标签的值都为0，document. document Element. clientWidth是当前浏览器内部显示区域的宽度**

**clientLeft和clientRight基本不用**

存在兼容问题：document.body.clientWidth||document.documentElement. clientWidth

# 13.HTML DOM

**1.innerHTML属性**

innerHTML 属性用于获取或替换 HTML 元素的内容。

3.标签属性

标签的html属性可以直接访问，eg：

var link = document.createElement("a");

link.href = url;

**2.样式属性**

**语法:**

**Object.style.property=new style;**

eg：

mychar.style.color="red";

mychar.style.fontSize="20";

注意：display属性设为none可不显示该元素

**3.控制类选择器**

**语法：**

**object.className = classname**

eg:

.one{}

.three{}

var p1 = document.getElementById("p1");

p1.className="one";

p1.className="three";

# 14.正则表达式

### 14.1.正则表达式的方法

**语法**

**/正则表达式主体/修饰符(可选)**

修饰符是可选的，包含属性 "g"(全局匹配, 查找所有匹配而非在找到第一个匹配后停止),"i"(区分大小写的匹配), 和 "m"(多行匹配)

eg: var patt = /runoob/i

在 JavaScript 中，正则表达式通常用于两个字符串方法 : search() 和 replace()。

1. search() 方法 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子串的起始位置。eg：

var str = "Visit Runoob!";

var n = str.search(/Runoob/i);//搜索字符串且不区分大小写，结果为6

//也可以单独传入字符串

var n = str.search("Runoob");//6

1. replace() 方法 用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。eg:

<p id="demo">请访问 Microsoft!</p> var str = document.getElementById("demo").innerHTML;

var txt = str.replace(/microsoft/i,"Runoob");//请访问Runoob!(不区分大小写)

//也可以单独传入字符串

var txt = str.replace("Microsoft","Runoob");

### 14.2.正则表达式的模式

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [abc] | 查找方括号之间的任何字符中的一个，只占一位 |
| [^abc] | 查找任何不在方括号之间的字符中的一个，同上 |
| [0-9] | 查找任何从 0 至 9 的数字的一个。同上 |
| [a-z] | 查找任何从小写 a 到小写 z 的字符的一个。同上 |
| [A-Z] | 查找任何从大写 A 到大写 Z 的字符的一个。同上 |
| [A-z] | 查找任何从大写 A 到小写 z 的字符的一个。同上 |
| (x|y) | 或 |

|  |  |  |
| --- | --- | --- |
| [.](http://www.w3school.com.cn/jsref/jsref_regexp_dot.asp) | 查找单个字符，除了换行和行结束符。 | |
| \. | 代表.号 | |
| [\w](http://www.w3school.com.cn/jsref/jsref_regexp_wordchar.asp) | 查找单词字符。 | |
| [\W](http://www.w3school.com.cn/jsref/jsref_regexp_wordchar_non.asp) | 查找非单词字符。 | |
| [\d](http://www.w3school.com.cn/jsref/jsref_regexp_digit.asp) | 查找数字。 | |
| [\D](http://www.w3school.com.cn/jsref/jsref_regexp_digit_non.asp) | 查找非数字字符。 | |
| [\s](http://www.w3school.com.cn/jsref/jsref_regexp_whitespace.asp) | 查找空白字符。 | |
| [\S](http://www.w3school.com.cn/jsref/jsref_regexp_whitespace_non.asp) | 查找非空白字符。 | |
| [\b](http://www.w3school.com.cn/jsref/jsref_regexp_begin.asp) | 匹配单词边界。 | |
| [\B](http://www.w3school.com.cn/jsref/jsref_regexp_begin_not.asp) | 匹配非单词边界。 | |
| \0 | 查找 NUL 字符。 | |
| [\n](http://www.w3school.com.cn/jsref/jsref_regexp_newline.asp) | 查找换行符。 | |
| \f | 查找换页符。 | |
| \r | 查找回车符。 | |
| \t | 查找制表符。 | |
| \v | 查找垂直制表符。 | |
| [\xxx](http://www.w3school.com.cn/jsref/jsref_regexp_octal.asp) | 查找以八进制数 xxx 规定的字符。 | |
| [\xdd](http://www.w3school.com.cn/jsref/jsref_regexp_hex.asp) | 查找以十六进制数 dd 规定的字符。 | |
| [\uxxxx](http://www.w3school.com.cn/jsref/jsref_regexp_unicode_hex.asp) | 查找以十六进制数 xxxx 规定的 Unicode 字符。 | |
| [n+](http://www.w3school.com.cn/jsref/jsref_regexp_onemore.asp) | | | 匹配任何包含至少一个 n 的字符串。 | |
| [n\*](http://www.w3school.com.cn/jsref/jsref_regexp_zeromore.asp) | | | 匹配任何包含零个或多个 n 的字符串。 | |
| [n?](http://www.w3school.com.cn/jsref/jsref_regexp_zeroone.asp) | | | 匹配任何包含零个或一个 n 的字符串。 | |
|  | | | | |
| [n{X}](http://www.w3school.com.cn/jsref/jsref_regexp_nx.asp) | | | 匹配包含 X 个 n 的序列的字符串。 | |
| [n{X,Y}](http://www.w3school.com.cn/jsref/jsref_regexp_nxy.asp) | | | 匹配包含 X 至 Y 个 n 的序列的字符串。 | |
| [n{X,}](http://www.w3school.com.cn/jsref/jsref_regexp_nxcomma.asp) | | | 匹配包含至少 X 个 n 的序列的字符串。 | |
| [n$](http://www.w3school.com.cn/jsref/jsref_regexp_ndollar.asp) | | | 匹配任何结尾为 n 的字符串。 | |
| [^n](http://www.w3school.com.cn/jsref/jsref_regexp_ncaret.asp) | | | 匹配任何开头为 n 的字符串。 | |
| [?=n](http://www.w3school.com.cn/jsref/jsref_regexp_nfollow.asp) | | | 匹配任何其后紧接指定字符串 n 的字符串。 | |
| [?!n](http://www.w3school.com.cn/jsref/jsref_regexp_nfollow_not.asp) | | | 匹配任何其后没有紧接指定字符串 n 的字符串。 | |

eg:

var objRegExp= /^\d+\.\d+$/;//是否为浮点数

var reg=/^[\u4E00-\u9FA5]{2,4}$/;//是否为2-4位中文

var reg=/^[0-9]{8}$/;//是否为8位数字

var reg= /^((0\d{2,3}-\d{7,8})|(1[3584]\d{9}))$/;//电话号码

^1(3|4|5|7|8)\d{9}$//电话号码

^[a-zA-Z0-9\_-]+@[a-zA-Z0-9\_-]+(\.[a-zA-Z0-9\_-]+)+$//邮箱

**注意：**

**1.^为匹配开头，$为匹配结尾，小括号代表是个整体，中括号代表从中选一个，只占一位，大括号代表出现次数，\_代表\_，-代表-**

**2.正则表达式可以不要加””，使用 /^…$/ 分割即可**

### 14.3. RegExp 对象

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。eg：

var patt1=new RegExp("e");  
patt1.test("The best things in life are free");//返回true

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。eg：

var patt1=new RegExp("e");

patt1.exec("The best things in life are free");//返回e

var reg=/^1[3,5,8]\d{9}$/;

reg.test(“18895673645”)

# 15.应用篇—图片缓慢缩放效果

(boxes[minIndex].offsetLeft - boxes[i].offsetLeft)