# JAVASCRIPT

## JavaScript

### JavaScript

JavaScript ：基于对象和事件驱动的脚本语言，应用于客户端。JavaScript诞生于1995年，它的出现主要是用于处理网页中的前端验证。所谓的前端验证，就是指检查用户输入的内容是否符合一定的规则。比如：用户名的长度，密码的长度，邮箱的格式等。将这些验证放到浏览器中，让浏览器区验证这些信息，而不用传到服务器中，节省了响应的时间。网景公司基于这点问题，开发出了LiveScript，后面由于sun公司的介入，改为JavaScript。目前，的网速足以支持响应时间，现在的JavaScript不仅支持前端验证，还支持各种动态效果。

微软开发的Jscript与JavaScript的功能相同，网景公司将JavaScript捐给了ECMA欧洲计算机协会，后改为ECMAScript，ECMAScript是一个标准，而这个标准需要由各个厂商去实现。不同的浏览器厂商对该标准会有不同的实现。

|  |  |
| --- | --- |
| 浏览器 | JavaScript实现方式(引擎) |
| FireFo | SpiderMonkey |
| Internet ExplorerI | JScript/Chakra |
| Safari | JavaScriptCore |
| Chrorme | v8(执行最快的引擎) |
| Carakan | Carakan |

我们已经知道ECMAScript是JavaScript标准，所以一般情况下这两个词我们认为是一个意思。但是实际上JavaScript的含义却要更大一些。

一个完整的JavaScript实现应该由以下三个部分构成：

ECMAScript 标准

BOM 浏览器对象模型，提供一组对象操作浏览器。

DOM 文档对象模型，提供一组对象操作文档。

### Js的特点

-解释型语言：不用编译，可以

-类似于C和Java的语法结构

-动态语言：保存变量的数据，可以是任何类型。

-基于原型的面向对象

### Js的作用

1.实现网页特效

2.读写页面元素

3.表单验证

4.响应事件

5.控制浏览器中的cookies

6.制作游戏,动画

### js的组成（三部分）

ECMAscript：由ECMA组织制定的js的语法，语句。(ECMA：欧洲计算机协会)

由ECMA-262定义的ECMASctipt与Web浏览器没有依赖关系.实际上,这门语言本身并不包含输入和输出定义.ECMA-262定义的只是这门语言的基础,而在此基础之上可以构建更完善的脚步语言.它主要规定了这个语言的下列组成部分:语法,类型,语句,关键字,保留字，操作符,对象.

DOM: document object model 文档对象模型。提供访问和操作网页内容的方法和接口。

DOM(Document Object Model)是针对XML但经过扩展用于HTML的应用程序编程接口。DOM把整个页面映射为一个多层节点结构.借助DOM提供的API,开发人员可以轻松自如的删除，添加，替换或者修改任何节点.DOM并不只是针对于javaScript的,很多别的语言也都实现了DOM。

BOM browser object modele浏览器对象模型：提供与浏览器交互的方法和接口。

BOM 实现对浏览器窗口进行访问和操作。例如弹出新的浏览器窗口，移动、改变和关闭浏览器窗口，提供详细的网络浏览器信息（navigator object），详细的页面信息（location object），详细的用户屏幕分辨率的信息（screen object），对cookies的支持等等。BOM作为JavaScript的一部分并没有相关标准的支持，每一个浏览器都有自己的实现，虽然有一些非事实的标准，但还是给开发者带来一定的麻烦。

### 第一个js代码

|  |
| --- |
| <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  //以下的代码顺序执行  //控制浏览器弹出一个警告框  alert("haha");  //让计算机在页面输出一个内容  //document.write("");在body输出一个内容  document.write("让计算机在页面输出一个内容");  //向控制台输出一个内容  //console.log("");任何浏览器都可以，在开发者界面可以看到。  console.log("向控制台输出一个内容");  </script>  </head> |

## js和html的结合方式（两种）

### 直接写在标签

|  |
| --- |
| <!--可以将js代码编写到标签的onclick属性中当我们点击按钮时，js代码才会执-->  <button onclick="alert('js代码');">点我</button>  <!--可以将js代码写在超链接的href属性中，这样当点击超链接时，会执行js代码-->  <a href="javascript:alert('超链接写js代码');"> 超链接js代码</a>  <!--点完超链接没有任何功能-->  <a href="javascript:;"> 超链接js代码</a>  <a href="javascript:void(0);"> 超链接js代码</a> |

虽然可以写在标签的属性中，但是他们属于结构与行为耦合，不方便维护，不推荐使用

### 使用script标签引入一个外部的js代码

格式：<script type=”text/javascript” src=“js文件的相对路径”></script>

特点：可以是实现js代码的重复使用，避免了代码的冗余。

注：

1.script一旦引入外部文件，就不要再script标签里写js的代码了，不会执行。可以在写一个script写在内部。浏览器都会按照<script>元素在页面中出现的先后顺序对它们依次进行解析

2.引文js在html文档中是一门单独的语言，所以可以声名在文档中的任意位置，一般情况下声名在head标签中。页面先显示出来，再执行js代码，可以将js代码放到body标签后。

3.写到外部文件中可以在不同的页面中同时引用，也可以利用到浏览器的缓存机制,推荐使用的方式。

## Js中的语法

JS中的注释，不会被执行，可以在源代码中查看，可以使用注释进行简单的调试。

### 注释

单行注释

//注释内容

多行注释

/\* 注释内容 \*/

### JS中语法

1.JS中严格区分大小写

2.JS每一条语句以分号(;)结尾。如果不写分号，浏览器会自动添加，但是会消耗一些系统资源，而且有些时候，浏览器会加错分号，所以在开发中分号必须写。

3.JS中会忽略多个空格和换行，所以我们利用空格和换行对代码进行格式化。

# js的数据类型

## JS基础

### 标识符

标识符在JS中所有的可以由我们自主命名的都可以称为是标识符

例如：变量名、函数名、属性名都属于标识符，或者是循环语句中跳转位置的标记。

### 标识符命名规则：

1.标识符中可以含有字母、数字、、$。

2.标识符不能以数字开头。

3.标识符不能是javascript中的关键字或保留字。

4.标识符一般采用驼峰命名法:（规范，不强迫执行）

首字母小写，每个单词的开头字母大写，其余字母小写

例如：helloworld xxxYyyZzz

JS底层保存标识符时实际上是采用的Unicode编码，所以理论上讲，所有的utf-8中含有的内容都可以作为标识。

## JS的直接量

javaScript中不去分整数值和浮点数字，所有数字均用浮点数值表示。

### 数字直接量

十进制：0,33，100000,

十六进制：0xff：15\*16+15=255

八进制：有些js支持，最好不用。

### 浮点型直接量

一个实数由整数部分、小数点和小数部分组成。还可以使用e来表达。

例如：3.14、.33333、6.02e23

### 文本

## 变量

### 变量

变量可以用来保存字面量，而且变量的值是可以任意改变的变量更加方便我们使用，所以在开发中都是通过变量去保存一个字面量，而很少直接使用字面量。可以通过变量对字面量进行描述。

### 声明变量

javascript 中变量通过var关键字(variable)来声明的。

例如：

var a = 1 ;

var school = "beijingyizhong" , diqu = "beijing" , age = 100;

|  |
| --- |
| 同一变量，赋值不同类型  var school = "beijing";  document.write(school);  school = 132134;  document.write(school);  不声明变量也是可以的。  var test;  var \_beijing;  var $djas; |

用var声明和不用var声明的区别：

1、在函数内部如果用var声明变量和不用时有很大差别，用var声明的是局部变量，在函数外部访问这个变量是访问不到的，没var声明的是全局变量。在函数外部是可以访问到的。

|  |
| --- |
| function ff(){  var a=1;  console.log(a); //1  }  ff();  console.log(a); //a is not defined |

如果去掉var来声明变量a：

|  |
| --- |
| function ff(){  a=1;  console.log(a); //1  }  ff();  console.log(a); //1 |

2.全局作用域内声明变量

在这里用var声明的变量我们之所以认为声明的是全局变量是因为它现在处于的作用域范围是全局，实际上它声明也是局部变量，只是现在它的局部变量是全局而已，所以就相当于起着全局变量的作用

|  |
| --- |
| var num1 = 1;  num2 = 2;  console.log(num1);//1  console.log(num2);//2  delete num1; //无法删除,会报错  delete num2; //删除  console.log(num2);//num2 is not defined |

前者是变量声明，带不可删除属性，因此无法被删除；后者为全局变量的一个属性，因此可以从全局变量中删除。

### 变量赋值

例如：a=”123”;

声明和赋值同时

例如：var a = 789;

## 数据类型（6种）

### Java的基本数据类型

String 字符串类型 var str=“abc”;

Number 数字类型 var num=123;

Boolean 布尔类型 var flag = true;

var str ="123";

alert(str);

Object类型

Date 对象类型 Var date =new Date();

alert（typeof date）// Object类型

Null：显示的赋值为bull，指定这个var只能引用对象。

获取对象的引用，null表示对象的引用为空，所有对象的引用都object。

Undifined:变量声名不赋值的情况下，默认值时Undifined。

var a;

alert(a);// undifined

typeof

提供了一个运算符 typeof 查看当签变量的数据类型

例如： alert（typeof（num）） 打印出 number

其中String Number Boolean Null Undefined属于基本数据类型而Object属于引用数据类型。

注：

1.js中的代码所有变量声明只有var关键字。

2.js中的变量名严格区分大小写的。

3.js中的字符串可以使用双引号和的单引号。

4.js中可以声名同名变量，不会报错，后声名的变量会将前面的覆盖。

5.java中变量的类型决定变量的使用方式，而js变量只有在赋值时才能决定变量的使用方式。两种的都时数据决定数据运算方式。

### String 字符串类型

字符串（string)是一组由16位值组成的不可变的有序序列，每个字符通常来自于Unicode字符集。JavaScript通过字符串类型来表示文本。字符串的长度（length）是其所

含16位值的个数。JavaScript字符串（和其数组)的索引从零开始：第一个字符的位置是0，第二个字符的位置是1，以此类推。空字符串（empty string)长度为0，JavaScript中并没有表示单个字符的“字符型”。要表示一个16位值，只需将其赋值给字符串变量即可，这个字符串长度为1。

String：-在JS中字符串需要使用引号引起来，使用双引号或单引号都可以。但是要混着用。

var str = “String”;

alert(str);//输出变量

alert(“str”);/输出字符串

注：

1、字符串连接：使用+，运算作用域两个字符串，效果是将第二个字符串拼接到第一个字符串之后。

2、确定字符串的长度，可以使用字符串你的length属性。

3、javascrit的字符串长度永远是不会变得，对字符串的操作，返回的是一个新的字符串。

### Number

在JS中所有的数值都是Number类型，包括整数和浮点数（小数）。

var a = 123;

var b = 456.65;

JS中可以表示的数字的最大值Number.MAX VALUE=1.7976931348623157e+308。

Number.MIN\_VALUE 大于0的最小值5e-324。

注：

1.如果使用Number表示的数字超过了最大值，则会返回个Infinity表示正无穷。

2. 使用typeof检查Infinity也会返回Number。

例如：var a = Number.MAX\_VALUE\* Number.MAX\_VALUE;//最大值Infinity

var b= -Number.MAX\_VALUE\* Number.MAX\_VALUE;//最小值-Infinity

var c = infinity; //表示正无穷

3. NaN 是一个特殊的数字表示Not A Number，使用typeof检查一个NaN也会返回number。

var b = NAN;//表示的数字类型为number;

var c = “NAN”;//返回的类型为String;

4. 在JS中整数的运算基本可以保证精确, 如果使用JS进行浮点元素，可能得到一个不精确的结果，所以使用进行精确运算。

例如：var c = 0.1+0.2;//0.3000000000000000004;

### 布尔值(Boolean)

Boolean 布尔值。布尔值只有两个，主要用来做逻辑判断

true-表示真 false-表示假

例如：var bool = false;

使用typeof检查一个布尔值时，会返回boolean

转成false的值：undefined、null、0、-0、NaN、” ” 空串 都会转为false，其他的值都为true。

### Null值

Nu11类型的值只有一个，就是nu11.nu11这个值专门用来表示一个为空的对象。

var a = null;

注：使用typeof检查一个nu11值时，会返回object。

### Undefined

Undefined(未定义)类型的值只有一个，就undefind。

注：1.当声明一个变量，但是并不给变量赋值时，它的值就是undefined。

2.使用typeof检查一个undefined时也会返回undefined。

### 转义字符

转义字符：在字符串中我们可以使用\作为转义字符，当表示一些特殊符号时可以使用\进行转义。  
 \”表示”

\’表示’

\n表示换行

\t制表符

[\\表示\\](file:///\\表示\\)

## Js数据类型转换

强制类型转换：指将一个数据类型强制转换为其他的数据类型。

类型转换主要指，将其他的数据类型，转换为String Number Boolean。

### 使用Number(其他类型的数据);

string 类型的数字字符:对应的number数字

string 非数字字符:NaN(number类型)

boolean true: 1

boolean false:0

object 有具体的值:数字

object null: 0

### 其他类型转换为boolean 类型,使用Boolean(其他类型的数据);

number 正数或者负数：true

number 0:false

string 非空字符串 : true

string 空字符串 "":false

object 具体的对象 : true

object null : false

声明不赋值的变量: false

### 其他数据类型转为String

方式一：调用被转换数据类型的toString()方法,该方法不会影响原变量，它会将结果返回。

var a = 12;

a = a.toString();//a. toString()返回的值赋给a

alert(typeof a)

注：null和undefined这两个值没有toString如果调用他们的方法，会报错。

方法二：调用String函数 var b = String(value)

使用String()函数做强制类型转换时，对于Number和Boolean实际上就是调用的tostring()方法但是对于nu11和undefined，就不会调用toString()方法。它会将null直接转换为“null"，将undefined 直接转换为“undefined"

例如：

var a= 12.5;

var b =String(a);//使用String()方法。

alert(typeof(b));

方法三：其他类型转换为字符串,只需要将其与空字符串拼接即可。

### 将其他的数据类型转换为Number

转换方式一：使用Number()函数

1.字符串-->数字

1.如果是纯数字的字符串，则直接将其转换为数字

2.如果字符串中有非数字的内容，则变量的内容变为NaN。

3.如果字符串是一个空串或者是一个全是空格的字符，则为0。

4.如果字符串中有空格也有连续的数字，则键自动将字符串去掉。（小数也可以，但是不能超过有两个点。）

2. -布尔-->数字

true转成1

false转成。

3. Nu11-->数字0

4.undefined-->\*数字NaN

注：如果转化的内容有非法的字符，会变为nan。

转换方式一：使用parseInt()和parseFloat()函数

1、parseInt()可以将一个字符串中的有效的整数内容去出来，然后转换为Number。以整数数字开头，非整数数字结尾。例如：a123，123.35ad都是不可以的。

a ="a123px";

a = parseInt(a);

console.log(a)

2、parseFloat()可以将一个字符串中的有效的小数内容去出来，然后转换为Number。例如：a12.3，是不可以的。123.456.789abc，取出来的数字是123.456。

a ="123.456asdfv";

a = parseFloat(a);

console.log(a)//123.456

3、如果对非String使用parseInt()或parseFloat()它会先将其转换为String，然后在操作。

例如：对boolean进行转换

a = true;

a = parseInt(a);//NAN

### 转换布尔值

将其他的数据类型转换为Boolean，使用Boolean()函数。

1、数字转boolean，除了0和NAN，其余都为true。

var a = 123;//true

var a = -123; //true

var a = Infinity; //true

var a = 0// false

var a = NAN;//false

a = Boolean(a);

console.log(a)

2、字符串转boolean,除了空串，其余都为true。

var a = “hello”;

var a = “false”;

var a = “true”;

var a = “”;//false

a = Boolean(a);

console.log(a);

3、null和undefined都会转为false

4、对象也会转为true。

# 算术运算符

运算符也叫操作符，通过运算符可以对一个或多个值进行运算，并获取运算结果、

## typeof运算符

typeof就是运算符，可以来获得一个值的类型，它会将该值的类型以字符串的形式返number string boolean undefined object。

例如：var a = 123;

var result = typeof a;

console.log(result);//number

## 算术运算符

注：不会对元变量产生影响。

加法：+; 可以对两个值进行加法运算，并将结果返回。

例如：var a = 123; var b = 234; alert(a+b);//357

注：1任何值与NAN做运算都为NAN

a = NAN+a;//NAN

2.字符串类型相加，为拼串。

例如：js中一个字符歘能不能写在多行，但可以使用+来将多个字符串拼接。

var str=\"锄禾日当午，”+

“汗滴禾下土，”+

“谁知盘中餐，”+

“粒粒皆辛苦”;

3. 任何的值和字符串做加法运算，我们只需要为任意的数据类型+一个””即可将其转换为String这是一种隐式的类型转换，由浏览器自动完成，实际上它也是调用String函数。

例如：result = true +1;//true1

作用：将数值变为字符串，是一个

var c = 123;

result = c+””// 同c.String()相同

4.案例：算术的执行顺序，从左向右。

result = 1+2+'3';

alert(result);//330

result = "1"+2+3;

alert(result);//123

减法：-：可以对两个值进行减法运算，并将结果返回。

例如：result=\"1\"+2+3;//123

result=100-5;//95

result=100-true;//99

result=100-"1";//99

乘法：\* 对象个数进行乘法运算。

除法：/ 对象个数进行乘法运算。

余数：% 取模操作

例如：result = 9%3;//0

任何值做-\*/运算时都会自动转换为Number我们可以利用这一特点做隐式的类型转换可以通过为一个值-0\*1/1来将其转换为Number原理和Number()函数一样，使用起来更加简单。

### 类型转化的运算

number类型的和number类型的

number类型和boolean类型(true--1,false--0)

number类型和string类型(\*,/,%,-)

string类型和string类型的数字(\*,/,%,-)

string类型数字和boolean类型(\*,/,%,-)

在算术运算中如果两边的数据类型不是number的话，会使用Number()强转后在进行运算，字符串的加法会作为字符连接符，不会运算。在字符串中”+”符合代表的是字符串的连接符，不会参与运算。

## 一元运算符：只需要一个运算符

### 1+正号：正号不会对数字产生任何影响。

例如： var a = 123;

a = +a;

### ﹣号：负号可以对数字进行负号的取反。

例如： var a = 123;

a = +a;

### 对于非Number

对于非Number类型的值，它会将先转换为Number，然后在运算可以对一个其他的数据类型使用十，来将其转换为number，它的原理和Number()函数一样。

例如：var a = “123”;

a = +a;//此时a的类型为Number类型。

var a = “true”;

a = +a;//a 为1此时a的类型为Number类型。

### 自增++：通过自增可以使变量在自身的基础上增加1，对于一个变量自增以后，原变量的值会立即自增1

自增分成两种：后++（a++）和前++（++a)

注：无论是a++还是++a，都会立即使原变量的值自增1，但是不同的是a++和++a的值不同。

a++的值等于原变量的值（自增前的值）

++a的值等于原变量新值（自增后的值）

例如：var a = 1;

console.log(a++);//1

console.log(++a);//2

例如：d = 20;

//20+22+22

//var result=d++ + ++d+d;

d = d++;d=21;

### 自减--

自减减：通过自减可以使变量在自身的基础上减1

自减分成两种：后--(a--)和前--（--a)

无论是a--还是--a都会立即使原变量的值自减1，不同的是a--和--a的值不同

a--是变量的原值（自减前的值）

--a是变量的新值（自减以后的值）

## 逻辑运算

! 、 & 、&&、 | 、|| (与java中的一致);

var a=true;

var b=true;//false

alert(a&b);

if(a&b){//

alert("haha");

}

### ！非：就是值对一个布尔值进行取反操作。

true变false,false变true

注：对非boolean进行去反，先转为boolean型，再进行运算。

例如：var a = 10;

a = !!a;//a = true;先将a转为boolean，然后取反，再取反。

可以为一个任意数据类型取两次反，来将其转换为布尔值，原理和Boolean()函数一样。

### &&与运算

&&可以对符号两侧的值进行与运算并返回结果

运算规则：

1.两个值中只要有一个值为false就返回false，只有两个值都为true时，才会返回true。

例如：//如果两个值都是true则返回true

var result=true && true;

var result=false&& true;

console.1og（"result="+result)

2.JS中的“与”属于短路的与，如果第一个值为false，则不会看第二个值”。

例如：true&&alert(“短路与”);//可以弹出短路与

false&&alert(“短路与”);//不能弹出短路与

### ||或运算

||可以对符号两侧的值进行或运算并返回结果

运算规则：

1.两个值中只要有一个值为true就返回true，只有两个值都为false时，才会返回false。

例如：//如果两个值都是true则返回true

var result=true && true;

var result=false&& true;

console.1og（"result="+result)

2.JS中的“或”属于短路的与，如果第一个值为true，则不会看第二个值”。

例如：true||alert(“短路与”);//不能弹出短路与

false||alert(“短路与”);//能弹出短路与

### 非布尔的与或运算

与运算：

-如果第一个值为true，则必然返回第二个值

-如果第一个值为false，则直接返回第一个值

或运算

-如果第一个值为true，则直接返回第一个值

-如果第一个值为false，则返回第二个值

&&

规则：与非布尔值进行与或运算时，会向将其转换为布尔值，然后再运算。

例如：//true && true

//与运算：如果两个值都为true，则返回后边的

var result=5&&6;

例如：与运算：如果两个值中有false，则返回靠前的falsel

//false && true

result=0&&2;

result=2&&0;

//false && false

result=NaN &&e;

result= && NaN;

||

例如：//如果第一个值为true，则直接返回第一个值

result=2 || 1;

result=2 || NaN;

result=2 ||0;

//如果第一个值为false，则直接返回第二个值

result=NaN ||1;

result=NaN ||0;

## 赋值运算符=

=：可以将符号右侧的值赋值给符号左侧的变量。

例如：var a = 10;

可以将符号右侧的值赋值给符号左侧的变量

+=

a +=5等价于a=a+5

-=

a-=5等价于a=a-5

\*=

a\*=5等价于a=a\*5

/=

a/=5等价于a=a/5

%=

a %=5等价于a=a%5

## 关系运算符

关系运算符：通过关系运算符可以比较两个值之间的大小关系，如果关系成立它会返回true，如果关系不成立则返回false。

>大于号

-判断符号左侧的值是否大于右侧的

-如果关系成立，返回true，如果关系不成立则返回false

例如：var result = 5>10;//false

>=大于等于

-判断符号左侧的值是否大于或等于右侧的值

<小于号 同理

<=小于等于 同理

### 非数值的情况关系运算符

对于非数值进行比较时，要将非数值转为数值，然后进行比较。如果符号两侧的值都为字符串，不会将其转为数字进行比较，而会分别转为unicode编码进行比较，且比较字符编码时，一位一位的比较。

例如：var a = 1>true ;//false

var a = 1>false;//true

var a = 1>”0”;//true

var a = 10>null;//true null会转为0

//任何值和NAN做任何比较都是false

console.log(10<=”hello”);//false

console.log(“11”<=”5”);//true

所以,注意：在比较两个字符串型的数字时，一定一定一定要转型。

console.log(“113232532323”<=”5”);//true 第一位1<5 所以直接出判断，后边的任何值都不会进行判断。

### ==运算符

==：相等运算符用来比较两个值是否相等，如果相等会返回true，否则返回false。

例如：var a = 4

console(a==4);//true

console(a==10);//false

注：1、则会自动进行类型转换，将其转换为相同的类型然后在比较。

console.log(a==4);//false

console.log("1"==1);//true

console.log(true=="1");//true

console.log(nul1==0);//false

2、undefined 衍生自null，所以这两个值做相等判断时，会返回true。

例如：console.log(undefined==null);//true

3、NaN不和任何值相等，包括他本身。

console.log(NaN==NAN);//false

例如：判断b的值是否为NAN

可以通过isNaN()函数值判断一个值是否为NaN，如果该值是NaN返则回true，否则返回false。

### ！=

！=;不相等运算符用来比较两个值是否不相等，如果相等会返回false，否则返回true。

### ===

全等：===

注：1.不相等也会对变量进行自动的类型转换，如果转换后相等它也会返回false。

2.它和相等类似，不同的是它不会做自动的类型转换。如果两个值得类型不同，直接返回false。

例：console.log(“123”===123);//false

### ==和===运算符

等值运算符:==

先判断类型，类型一致则直接比较，类型不一致，则先使用Number()进行强转后再进行比较。

等同运算符:===

先判断类型，类型一致则再比较内容，内容一致则返回true，内容不一致则返回false。

注意：

1.null和undefined在做==判断时候返回true

2.当用==去比较基础类型和对象类型时，对象会调用valueOf()方法或者是toString方法。对象和对象类型比较比较的是地址符。

## 条件运算符(三元运算符)

语法：条件表达式？语句1：语句2;

执行流程：条件运算符在执行时，首先对条件表达式进行求值，如果该值为true，则执行语句1，并返回执行结果。如果该值为false，则执行语句2，并返回执行结果。

例如：true？alert(“语句一”):alert(”语句二”);//语句一

var a = 10;

var b = 20;

var max = a>b?ab;//语句二

## 运算符的优先级

在JS中有一个运算符优先级的表，在表中越靠上优先级越高，优先级越高越优先计算，如果优先级一样，则从左往右计算。可以通过()来修改表达式的优先级。

# Js的逻辑结构

## 代码块

我们的程序是由一条一条语句构成的语句是按照自上向下的顺序一条一条执行的在JS中可以使用{}来为语句进行分组。

同一个{}中的语句我们称为是一组语句，它们要么都执行，要么都不执行，一个{}中的语句我们也称为叫一个代码块。在代码块的后边就不用再编写;了。

JS中的代码块，只具有分组的的作用，没有其他的用途。代码块内容的内容，在外部是完全可见。

例如：{

alert(\"hello\");

console.log(“你好\"）;

document.write(\"语句\"）;

}

## if语句

通过流程控制语句可以控制程序执行流程，使程序可以根据一定的条件来选择执行。

语句的分类：

1.条件判断语句

2.条件分支语句

3.循环语句

### 条件判断语句

条件判断语句：使用条件判断语句可以在执行某个语句之前进行判断，如果条件成立才会执行语句，条件不成立则语句不执行。

格式：if(判断条件){执行体}

例如：if(true){alert("单分支结构");}

if语句在执行时，会先对条件表达式进行求值判断，如果条件表达式的值为true，则执行if后的语句，如果条件表达式的值为false，则不会执行if后的语句。

如果希望if语句可以控制多条语句，可以将这些语句统一放到代码块中。if语句后的代码块不是必须的，但是在开发中尽量写上代码块，即使if后只有一条语句。

### if…else…语句

格式：if(判断条件){执行体}else{执行体}

当该语句执行时，会先对if后的条件表达式进行求值判断，如果该值为true，则执行if后的语句。如果该值为false,则执行else后的语句。

### 多分支语句

格式if(判断条件){执行体}else if(判断条件){执行体}……else{执行体}

当该语句执行时，会从上到下依次对条件表达式进行求值判断如果值为true,则执行当前语句。如果值为false，则继续向下判断。如果所有的条件都不满足，则执行最后一个else后的语句。该语句中，只会有一个代码块被执行，一旦代码块执行了，则直接结束语句。

## swicth选择结构

格式： switch (a){这里是全等比较。

case "1":

alert("第一项选择");

break;

case "2":

alert("第二项选择");

break;

default:

alert("没有对应的选项");

break;

}

在执行时会依次将case后的表达式的值和switch后的条件表达式的值进行全等比较，

如果比较结果为true，则从当前case处开始执行代码。

当前case后的所有的代码都会执行，我们可以在case的后边跟着一个break关键字，这样可以确保只会执行当前case后的语句，而不会执行其他的case语句

如果比较结果为false，则继续向下比较。如果所有的比较结果都为false，则只执行defau1t后的语句。

注：判断的变量可以是number类型也可以是string类型，但是不要混用。

## 循环结构

### for }循环

格式：for(变量;条件;迭代条件){循环体}循环

执行过程：

1.执行初始化表达式，初始化变量。(初始化表达式只执行一次)

2.执行条件表达式，判断是否执行循环。

如果为true，则执行循环。

如果为false，终止循环。

3.执行更新表达式，更新表达式执行完毕继续重复2。

例：九九乘法表

for(var i=1;i<=9;i++){

for(var j=1;j<=i;j++){

document.write(i+"\*"+j+"="+i\*j+"&nbsp;&nbsp;&nbsp;&nbsp;");

}

document.write("<br />");

}

### while循环

格式：while(循环条件){循环体}

执行流程：先对条件表达式进行求值判断，如果值为true，则执行循环体，循环体执行完毕以后，继续对表达式进行判断，如果为true，则继续执行循环体，以此类推。如果值为false，则终止循环。

### do-while循环

格式：do{循环体} while(循环条件)

执行流程：

do...while语句在执行时，会先执行循环体，循环体执行完毕以后，在对while后的条件表达式进行判断，如果结果为true，则继续执行循环体，执行完毕继续判断以此类推，如果结果为false，则终止循环。

### do-while和while的区别

# 对象

## Js的数据类型

### 引用数据类型

object对象

### 基本数据类型

String 字符串 Number数值Boolean 布尔值Null空值Undefined 未定义

-以上这五种类型属于基本数据类型，以后我们看到的值只要不是上边的5种，全都是对象

## 对象

在ECMA-262中把对象定义为：“无序属性的集合，其属性可以包含基本值、对象、或者函数。”严格来说，这就是相当于说对象是一组没有特定序列的值。对象的每一个属性或方法都有一个名字，而每一个名字都映射到一个值。（如：“女朋友”[对象]喜欢运动[对象的属性或方法]--打篮球[属性对应的值]，我门可以试着这样去理解，但实际可能有所区别）

基本数据类型的值都是单一的值”hello”，123,true.。值和值之间没有联系。

在JS中来表示一个人的信息（name gender age)：

var name="孙悟空";

VSrSGRdeT,5."男”;

var age=18;

如果使用基本数据类型的数据，我们所创建的变量都是独立，不能成为一个整体。 对象属于一种复合的数据类型，在对象中可以保存多个不同数据类型的属性。

## 对象的分类

### 内建对象

-由ES标准中定义的对象，在任何的ES的实现中都可以使用

-比如:Math String Number Boolean Function Object...

### 宿主对象

-由JS的运行环境提供的对象，目前来讲主要指由浏览器提供的对象

-比如BOM 一组对象、DOM一组对象。

### 自定义对象

-由开发人员自己创建的对象

## 创建对象

1、使用new关键字调用的函数，是构造函数constructor构造函数是专门用来创建对象的函数。

var obj = new Object();

2、使用typeof检查一个对象时，会返回object。

console.log(typeof obj);

### 变量的声明提前

变量的声明提前：使用var关键字声明的变量，会在所有的代码执行之前被声明。

例如：consloe.log(a);

var a = 5;

上述代码出现结果为a 没有被赋值，即undefined。声明在前，但是赋值仍然在后。如果没有写var关键字，则a不会被声明。此时使用windows.a创建属性。

## 属性

在对象中保存的值称为属性向对象添加属性。

语法：对象.属性名=属性值；

### 添加属性

属性的添加有两种方式：

对象名.属性名=值；

对象名[属性名]=值

|  |
| --- |
| //添加属性  var obj = {};  obj.username = 'foodoir';  obj.age = 21;  obj.sex = '男';  obj.addr = '湖南';  obj['test'] = 'hello world';  console.log(obj.username + " " + obj.age + " " + obj.sex + " " + obj.addr + " " + obj['test']); //foodoir 21 男 湖南 hello world |

### 修改

属性的修改也有两种方式：

对象名.属性名=值；

对象名[属性名]=值

|  |
| --- |
| //修改指定属性  obj.username = 'chenwu';  console.log("修改之后的名字：" + obj.username); //修改之后的名字：chenwu  obj['test'] = 'hello javascript';  console.log("修改之后的test：" + obj['test']); //修改之后的test：hello javascript |

### 删除

属性的删除也有两种方式：delete 对象名.属性名；delete 对象名[属性名] 。在delete删除指定的属性时应该注意：（后面讲对象的结构的时候会详细介绍）

delete只能删除自身属性，不能删除继承属性

要删除继承属性，只能从定义它属性的原型对象上删除它，而且这会影响到所有继承自这个原型的对象

delete只是断开属性和宿主对象的联系，而不会去操作属性的属性

delete不能删除哪些可配制性为false的属性

|  |
| --- |
| //通过delete删除指定的属性  delete obj.sex;  console.log(obj.sex); //undefined  delete obj['test'];  console.log(obj['test']); //undefined |

## 属性名和属性值

|  |
| --- |
| var person = {  username: "foodoir",  age: 21,  sex: "男",  addr: "湖南",  salary: 123456,  };  console.log("姓名：" + person.username + "\n" + "性别：" + person.sex); //姓名：foodoir 性别：男  console.log("年龄：" + person['age'] + "\n" + "薪水：" + person["salary"]); //年龄：21 薪水：123456  //如果属性不确定，需要使用[]  var key = 'username';  console.log(person.key); //undefined 此种方法不能得到key的值，通过[]方法可以得到  console.log(person[key]); //foodoir 添加一个变量相当于person['username']方式。  console.log(person['key']); //undefined key是变量不能使用引号，这个相当于寻找person中key属性 |

### 属性名

1、对象的属性名不强制要求遵守标识符的规范，什么乱七八槽的名字都可以使用，但是我们使用是还是尽量按照标识符的规范去做。

2、如果要使用特殊的属性名，不能采用.的方式来操作需要使用另一种方式：

语法：对象[”属性名”]=属性值 //这里没有点

例如：obj.123 = 789;//错误方式

obj[“123”] = 789; //这种方式是正确的。

console.log(obj[“123”]);//查询该属性时也要这种格式。

3、使用[]这种形式去操作属性，更加的灵活，在[]中可以直接传递一个变量，这样变量值是多少就会读取那个属性。相当于多条，没有规定死查询的值得名字。有利于代码的复用。

### 属性值

属性值：JS的属性值可以是任意的数据类型，甚至也可以是一个对象。

例如：obj.test = true;

obj.test = null;

obj.test = undefined;

obj.test = obj2;

如果要查处找obj中的属性

obj.test.name

### in运算符

in：通过该运算符可以检查一个对象中是否含有指定的属性。如果有则返回true，没有则返回false。

语法：”属性名”in对象

## 内存结构

### 基本数据类型

JS中的变量都是保存到栈内存中的，基本数据类型的值直接在栈内存中存储，值与值之间是独立存在，修改一个变量不会影响其他的变量。

### 引用数据类型

对象是保存到堆内存中的，每创建一个新的对象，就会在堆内存中开群出一个新的空间，而变量保存的是对象的内存地址（对象的引用），如果两个变量保存的是同一个对象引用，当一个通过一个变量修改属性时，另一个也会受到影响。

## 对象创建

### new创建对象(不经常用)

语法:var obj = new object();

|  |
| --- |
| var obj4 = new Object();//创建一个空对象  var arr = new Array();//创建一个空数组对象  var date = new Date();//创建一个空时间对象 |

### 用对象字面量创建一个对象

var obj1 = {};//没有任何属性的对象

onsole.log(typeof obj); //object

注：

1、用该方法创建对象与new操作创建对象完全相同。

2、用对象字面量创建对象时可以直接指定对象中的属性。对象的属性名可以加引号，也可以不加。

例：var obj2 = {

‘name’:”猪八戒”，//使用单引号

age:28， //什么都不适用

“gender”:”男”， //使用双引号

“for”:”关键字”， //特殊字符，需要使用放在引号中间。

test:{ name:”沙和尚”}//对象中值的部分可以是任何的类型。

'first-name':'foodoir', //-是特殊字符，也需要放在引号中间

};

属性名和属性值是一组一组的名值对结构，名和值之间使用：连接，多个名值对之间使用，隔开。如果一个属性之后没有其他的属性了，就不要写逗号。|

### 通过构造函数的形式创建对象

语法：function Person(){};或者var Person=function(){};

|  |
| --- |
| var obj5=new Test1();  function Test1(num1,num2){  this.n1=num1;  this.n2=num2;  }  var obj6=new Test1(5,6);  console.log(typeof obj6); //object  console.log(obj6 instanceof Object); //true |

注意点：

a.使用的时候通过new操作符得到对象var person1=new Person()

b.用构造器创建对象的时候可以接收参数

c.构造器函数的首字母最好大写，区别其他的一般函数

### type和instanceof的区别

typeof 和 instanceof 常用来判断一个变量是否为空，或者是什么类型的。但它们之间还是有区别的：

typeof 是一个一元运算，放在一个运算数之前，运算数可以是任意类型。 它返回值是一个字符串，该字符串说明运算数的类型。typeof 一般只能返回这几个结果：number,boolean,string,function,object,undefined。我们可以使用 typeof 来获取一个变量是否存在，如 if(typeof a!="undefined"){alert("ok")}，而不要去使用 if(a) 因为如果 a 不存在（未声明）则会出错，对于 Array,Null 等特殊对象使用 typeof 一律返回 object，这正是 typeof 的局限性。

instanceof 用于判断一个变量是否某个对象的实例在上面的例子中，obj6 instanceof Object判断的为true，则说明创建的obj6也是Object类型

## 对象的遍历

对象中的遍历有两种，一种是for/in遍历，一种是通过Object.keys(obj)函数进行遍历。

|  |
| --- |
| var obj = {  x: 1,  y: 2,  test: "helloworld",  edu: "javascript",  };  //通过for/in遍历属性  for(var i in obj) {  console.log(i); //x y test edu输出属性值。  console.log(obj[i]); //输出的是属性名对应的属性值  }  //通过Object.keys(obj)遍历属性  console.log(Object.keys(obj)); //["x", "y", "test", "edu"] |

虽然两种方法都可以对对象的属性进行遍历，但是二者之间还是有区别的。Object.keys() 方法会返回一个由给定对象的所有可枚举自身属性的属性名组成的数组，数组中属性名的排列顺序和使用for-in循环遍历该对象时返回的顺序一致（两者的主要区别是 for-in 还会遍历出一个对象从其原型链上继承到的可枚举属性）。

## 案例

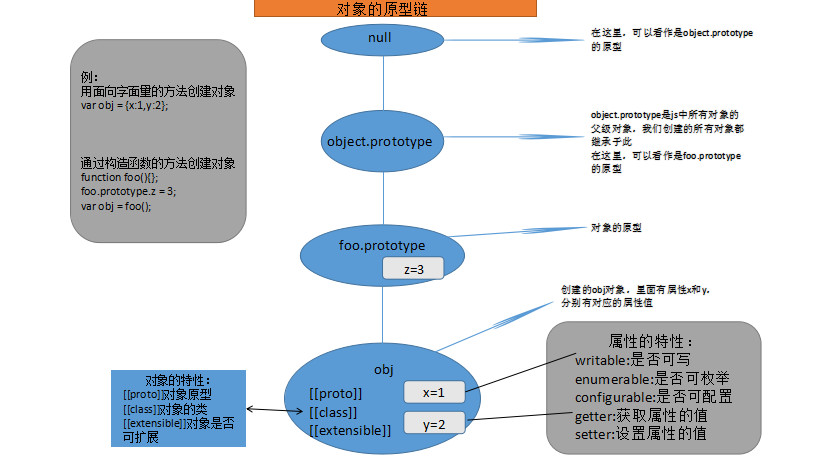
### 案例1

|  |
| --- |
| var obj = {  username: 'foodoir',  age: 21,  sex: '男',  sayHi: function() {  return 'say hi';  },  info: function() {  return obj.username + obj['sex'];  }  }  console.log(obj.sayHi()); //say hi  console.log(obj.info); //在info的后面要加(),不然结果是：function(){ return obj.username+obj['sex']; }  console.log(obj.info()); //foodoir男 |

### 案例2

|  |
| --- |
| function Person(username, age, sex) {  this.username = username;  this.age = age;  this.sex = sex;  this.info = function() {  return this.name + this.age;  }  }  var person1 = new Person('foodoir', 21, '男');  console.log(person1.info()); //NaN String+Number两种类型相加的结果为NaN  console.log(Person.person1); //undefined person1不是Person里面的属性，故返回undefined  console.log(person1.username); //foodoir  //向对象中添加新属性  person1.test = 'this is a test'; //this is a test  console.log(person1.test);  //向对象中添加新的方法  person1.fun = function() {  return 'hello world';  }  console.log(person1.fun()); //hello world |

## 对象在内存中的结构



首先从局部来看我在途中举的两个例子，我们通过面向字面量的方法var obj = {x:1,y:2};可以创建出我们想要的对象（如图：最大的椭圆），在obj对象中x,y分别是obj的属性，它们对应的属性值分别是1和2，当然除了属性对应的属性值之外，属性还有右侧的几个属性特性（writable:是否可写、 enumerable:是否可枚举、 configurable:是否可配置、 getter:获取属性的值 、setter:设置属性的值），这些属性的特性是可以设置和更改的（后面会介绍），只要是对象中的属性，都有这些特性，但是这些特性的值要柑橘具体情况来分析。属性有自己的特性，我们的obj是否也有自己的特性呢？从图中我们可以知道，obj对象也有自己的特性（[[proto]]对象原型、 [[class]]对象的类、 [[extensible]]对象是否可扩展），这些特性我们和面都会介绍。然后，我们通过构造函数的方法创建对象function foo(){}; foo.prototype.z = 3; var obj = foo();可以创建出我们想要的对象（如图：第二大的椭圆），它依然具有我们之前说的那些特性，但是具体特性要根据具体情况来分析，属性的特性也是一样。

接下来为了理解的方便，我们只看四个椭圆，每上一个椭圆代表着下一个椭圆中的对象的原型，比如说foo.prototype是obj对象的原型，foo.prototype的原型是object.prototype（值得注意的是object.prototype是javascript中所有对象的父级对象，我们创建的所有对象都继承于此），object.prototype往上，它的原型对象是null。这些话说起来有点拗口，我们还是通过具体的例子来了解它。

# 函数

## 函数

函数也是一个对象，函数中可以封装一些功能（代码），在需要时可以执行这些功能（代码）。函数中可以保存一些代码在需要的时候调用。

## 创建函数对象和调用

### 使用构造器创建函数对象

格式：var fun = new Function();

使用typeof检查一个函数对象的类型时，会返回function。

例如：console.log(typeof fun);// function

注：

1、可以将要封装的代码以字符串的形式传递给构造函数

var fun=new Function("console.log('hello 这是我的第一个函数’);");

fun();调用函数，函数中的代码会顺序执行。

封装到函数中的代码不会立即执行,函数中的代码会在函数调用的时候执行。一般在实际开发中，很少使用构造器创建一个函数对象。

2、函数对象也可以添加属性，具备普通对象的所有功能，并且能够封装可执行的代码。

### 使用函数声明创建函数对象

语法：function 函数名([形参1，形参2，…,形参N]){

语句;

}

例如：function fun2(){

console.log(“这是我的第一个函数”);

}

调用：fun2();

注：函数调用可以在函数声明之前。

### 匿名函数创建对象

匿名函数创建函数对象，将函数分装到了对象。

var fun3 = function([形参1，形参2,…,形参2 ]){

console.log(“这是我的第一个函数”);

}

调用该方法：

fun3();

## 函数的调用

### 调用函数对象

语法：函数对象();

例如：var fun=new Function("console.log('hello 这是我的第一个函数’);");

fun();此时会执行fun中形参的代码。在控制台输出('hello 这是我的第一个函数; fun()

//函数对象也可以添加属性

fun.name =”sx”;

### 函数声明提前

函数的声明提前

1、使用函数声明形式创建的函数function函数(){}它会在所有的代码执行之前就被创建，所以我们可以在函数声明前来调用函数.

function fun(){//可以提前调用该函数。

console.log("我是一个fun函数");

}

2、使用函数表达式创建的函数，不会被声明提前，所以不能在声明前调用。

var fun = function(){//var fun 可以提前声明，但是没有函数值。不可以提前调用该函数。

console.log("我是一个fun函数");

}

## 函数的形参

形参：可以在函数的()中来指定一个或多个形参（形式参数）。多个形参之间使用，隔开，声明形参就相当于在函数内部声明了对应的变量，但是并不赋值。

例如; function fun(a,b){

函数体;

}

在调用函数时，可以在()中指定实参（实际参数）实参将会赋值给函数中对应的形参。

例如：fun(3，4);

注：

1、调用函数时解析器不会检查实参的类型，所以要注意，是否有可能会接收到非法的参数，如果有可能则需要对参数进行类型的检查。

2、调用函数时，解析器也不会检查实参的数量，多余实参不会被赋值。如果实参的数量少于形参的数量，没有对象实参的形参将是undefined。

### 将对象传入形参。

var obj = {

name:"sx",

age:17

};

function fun(obj){

console.log(obj.name+obj.age);

}

fun(obj);

### 将函数传入形参

function fun1(b){};

function fun2(fun1);//传入fun1函数对象

{

a =fun1; 此时a引用函数的方法。

}

注：

1、fun1为函数对象，fun(b)为调用函数方法,此时返回函数的返回值。

## 函数的返回值

可以使用return设置函数的返回值,可以定义一个变量接收者个返回值。

格式：function fun(a,b){

函数体;

retrun 值;

}

注：

1、return之后的语句将不会执行。

2、return;之后没有跟值，返回一个undefined。函数中不写内容也返回一个undefined

3、return之后可返回任意类型的值。

案例1：判断是否为偶数

function even(num){

return num%2 == 0;

}

## 立即执行函数（匿名函数）

立即执行函数：函数定义完，函数立即被执行。一般执行一次。例如：

|  |
| --- |
| (function(a,b){//使用圆括括起来  console.log(“立即执行函数”);  })();  (function(a,b){//使用圆括括起来  console.log(a+b);  })(2,3); |

## 方法

函数也可以称为对象的属性，如果一个函数作为一个对象的属性保存，那么我们这个函数时这个对象的方法，调用这个函数就说调用对象的方法（method)。

例如：consloe.log()//这个是调用console的log方法。方法与对象中的其他的属性时同级的。

对象.属性//通过这种方法调用的函数称为函数，它是将一个方法名赋给了对象中的属性。函数本身与其他的属性不是同一级别。

## 对象的枚举

需求：拿到一个位置的对象，需要了解其中的属性。

### for-in语句

语法：for(var n in obj)

例如：for(var n in obj){

console.log(“属性名”+n);

console.log(“属性值”+obj[n]);

//这里不能是obj.n，此时表示的是去obj里边找n这个属性，但是n没有。使用变量查找，使用obj[]。

}

For...in语句对象中有几个属性，循环体就会执行几次,每次执行时，会将对象中的一个属性的名字赋值给变量。

## 作用域

作用域：指一个变量的作用范围。

在JS中一共有两种作用域：全局作用域和局部作用域。

### 全局作用域

1.直接编写在script标签中的JS代码，都在全局作用域。

2.全局作用域在页面打开时创建，在页面关闭时销毁。

3.在全局作用域中有一个全局对象window，它代表的是一个浏览器的窗口，它由浏览器创建我们可以直接使用。

例如：window.alert()；window.属性; window.方法;

4.在全局作用域中，创建的变量都会作为window对象的属性保存。

5.创建的函数都会作为window对象的方法保存。

6.全局作用域的变量都是全局变量，在页面的任意的部分都可以访问的到。

### 局部作用域(函数作用域)

局部作用域：调用函数时创建函数作用域，函数执行完毕后，函数作用域销毁。

1、每调用一次函数就会创建一个新的函数作用域，他们之间是互相独立的

2、在函数作用域中可以访问到全局作用域的变量

3、在全局作用域中无法访问到函数作用域的变量

4、当在函数作用域操作一个变量时，它会先在自身作用域中寻找，如果有就直接使用，如果没有则向上一级作用域中寻找。上一级作用域不一定为全局，可能为上一级函数。如果没有则向上一级作用域中寻找，直到找到全局作用域，如果全局作用域中依然没有找到，则会报错ReferenceError。

5、在函数中如果全局与局部有相同的属性，想要调用全局中的属性，可以使用window.属性来调用。

6、如果在局部作用域中没有写var关键字，定义的变量为全局变量。

例如：function fun(){

function fun1(){

a = 12;

}

b = 11; //相当于window.b;

fun1();

}

fun();

console.log(a)

console.log(b)

7、定义形参相当于在函数作用域中定义变量。

### 函数作用域声明提前

在函数作用域也有声明提前的特性，使用var关键字声明的变量，会在函数中所有的代码执行之前被声明。

1、属性

function fun(){

console.log(a);//此时a显示为undefined

var a = 35;

}

2函数

function fun(){

fun();//可以在函数声明之前调用该函数。

function fun(){

console.log(“fun函数”);

}

}

3

var c = 33;

function fun(){

console.log(c);//此时c显示为undefined,就近原则。

var c = 35;

}

4

var c = 33;

function fun(){

console.log(c);//此时c显示为35。

c = 35;//这里没有var 此时c不会被声明，c被定义为全局变量。

}

## this

解析器在调用函数每次都会向函数内部传递进一个隐含的参数，这个隐含的参数就是this，this指向的是一个对象，这个对象我们称为函数执行的上下文对象。

1、根据函数的调用方式的不同，this会指向不同的对象。

1、以函数的调用方式，this指向的是window

function fun(){ console.log(this)};//调用该函数，此时的this是window对象。

2、以方法的形式调用，this指向调用该方法的对象。

obj.sayname();//此时的this是obj

2功能：利用this实现代码的复用

|  |
| --- |
| var name = "全局";  function fun(){  console.log(this.name);//不要去指定确定的对象的name，使用this指当前对象的name  }  var obj = {  name :"孙悟空",  sayName:fun  };  var obj1 = {  name :"山河上",  sayName:fun  };  fun();  obj.sayName();  obj1.sayName(); |

## 工厂方法使用创建对象

工厂方法：相当于构造器中创建属性并赋值。

|  |
| --- |
| function createPerason(name,age,gender){  var obj = new Object();  obj.name = name;  obj.age = age;  obj.gender =gender;  return obj;  }  var obj1 = createPerason("孙悟空",12,"男");  var obj2 = createPerason("猪八戒",12,"男");  var obj3 = createPerason("沙和尚",12,"男"); |

缺点：使用工厂方法创建的对象，使用的构造函数都是Object，所以创建的对象都是Object这个类型，就导致我们无法区分出多种不同类型的对象。

## 构造函数（类）

构造函数：创建某一个类型的对象。

构造函数和与普通函数的区别：

1、构造函数的首字母大写

例如：function Person(){}

2、调用方式不同，普通方式直接调用，构造函数通过new关键字调用。

例如: var person = new Person();

### 构造函数的执行流程

1.立刻创建一个新的对象。

2.将新建的对象设置为函数中this，this就是新创建的对象。

3.逐行执行函数中的代码，可以在函数的代码块中添加属性，然后通过形参传递进来。

function Person(name，age，gender){

this.name = name;//必须使用this来调用，否则创建的对象为全局变量。

this.age = age;

this.gender = gender;

}

4.将新建的对象作为返回值返回。//此时返回的对象类型为Person

var obj1 = createPerason("孙悟空",12,"男");

var obj2 = createPerason("猪八戒",12,"男");

var obj3 = createPerason("沙和尚",12,"男");

### 优化1

|  |
| --- |
| function Perason(name,age,gender){  var obj = new Object();  this.name = name;  this.age = age;  this.gender =gender;  this.sayName = fun();  }  function fun(){//将fun函数定义到全局域中。  alert("我是"+this.sayName());  }  var obj1 = Perason("孙悟空",12,"男");  var obj2 = Perason("猪八戒",12,"男");  var obj3 = Perason("沙和尚",12,"男"); |

引入：上述创建的person对象，也就是构造函数每执行一次就会创建一个新的sayName方法，也是所有实例的sayName都是唯一的。这样就导致了构造函数执行一次就会创建一个新的方法，执行10000次就会创建10000个新的方法，而10000个方法都是一摸一样的

这是完全没有必要，完全可以使所有的对象共享同一个方法。

解决方法：可以将fun函数定义到全局域中，这样，创建的每个对象就共享着个方法。

### 优化2(prototype)

|  |
| --- |
| function Perason(name,age,gender){  var obj = new Object();  this.name = name;  this.age = age;  this.gender =gender;  this.sayName = fun();  }  function fun(){//将fun函数定义到全局域中。  alert("我是"+this.sayName());  }  var obj1 = Perason("孙悟空",12,"男");  var obj2 = Perason("猪八戒",12,"男");  var obj3 = Perason("沙和尚",12,"男"); |

引入：将sayName方法在全局作域中定义将函数定义在全局作用减，污染了全同作用域的命名空间而且定义在全局作用域中也很不安全。

prototype: 我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype对象。这个属性指向原型对象。如果函数通过普通方式调用prototype没有任何作用。

当函数以构造函数的形式调用时，它所创建的对象中都会有一个隐含的属性，指向该构造函数的原型对象，我们可以通过\_\_\_proto\_来访问该属性。

原型对象就相当于一个公共的区域，所有同一个类的实例都可以访问到这个原型对象，我们可以将对象中共有的内容，统一设置到原型对象中

|  |
| --- |
| function Person(name,age,gender){  this.name = name;  this.age = age;  this.gender =gender;  }  Person.prototype.a = 123;//原型中添加属性  Person.prototype.sayName=function(){//原型中添加方法  alert("我是"+this.name);  };    var obj1 =new Person("孙悟空",12,"男");  var obj2 =new Person("猪八戒",12,"男");  var obj3 =new Person("沙和尚",12,"男");  console.log(obj1.name);  console.log(obj1.sayName()); |

以后我们创建构造函数时，可以将这些对象共有的属性和方法，统一添加到构造函数的原型对象中，这样不用分别为每一个对象添加，也不会影响到全局作用域，就可以使每个对象都具有这些属性和方法了。

### hasOwnProerty 检查是否含有属性

引入：使用in检查对象中是否含有某个属性时，如果对象中没有但是原型中有，也会返回true

例如：console.log(“a” in obj1);//返回true

解决方法：hasOwnProerty(“属性值”)

例如：console.logm(obj1. hasOwnProerty(“a”));//会返回false

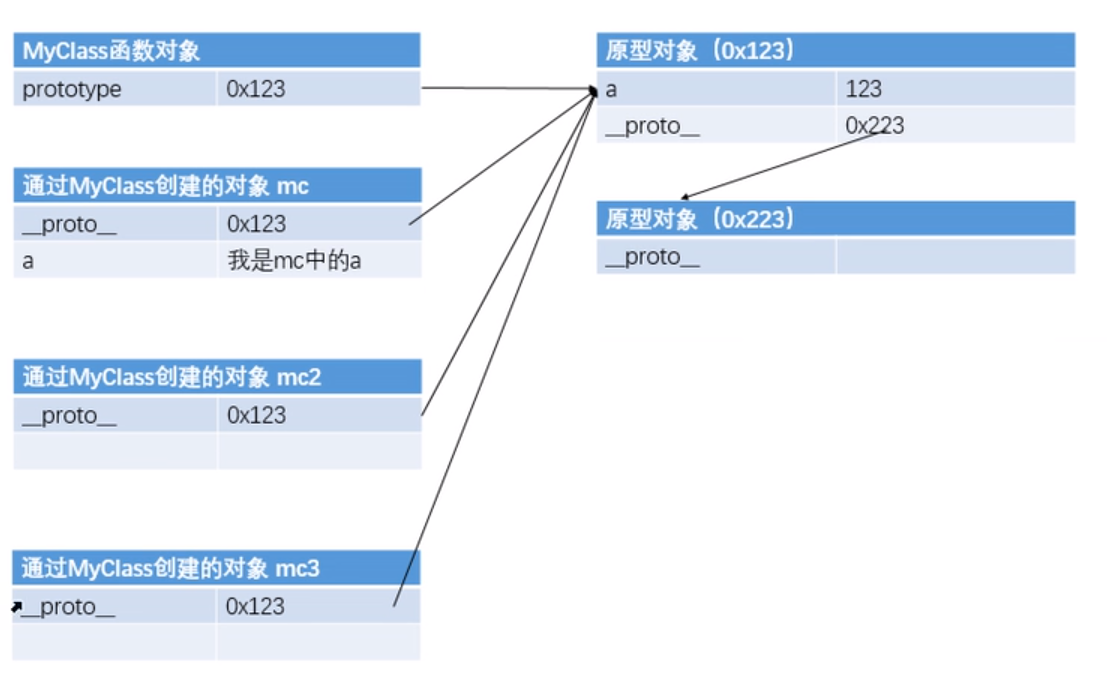
hasOwnProerty(“属性值”)：

原型对象也是对象，所以它也有原型，当我们使用一个对象的属性或方法时，会现在自身中寻找，自身中如果有，则直接使用，如果没有则去原型对象中寻找，如果原型对象中与，则使用，如果没有则去原型的原型中寻找。知道找到object对象的原型，但是object没有原型，所以找到object。object中没有原型，所以返回true。

console.log(obj1.\_\_proto\_\_.\_\_proto\_\_.hasOwnProperty("hasOwnProperty"));

查找obj原型的原型的hasOwnProperty方法是否存在，这里返回true。

这个方法是原型中方法，只要是自己没有添加的方法，都是原型中的方法。



## 常用方法

### instanceof

instanceof：判断一个对象是否为一个类的实例。

语法：对象instanceof构造函数名

返回值为boolean型

例如：comnsole.log(per instanceof Person) true

comnsole.log(dog instanceof Person) true

所有类的父类都是object。

### toString

|  |
| --- |
| function Person(name,age,gender){  this.name = name;  this.age = age;  this.gender =gender;  }  var p = new Person("sx",12,"女");  console.log(p);//这里返回的是"[object Object]" |

当我们直接在页面中打印一个对象时，事件上是输出的对象的toString()方法的返回值。

console.log(obj1.\_\_proto\_\_.\_\_proto\_\_.hasOwnProperty("toString"));//true

如果我们希望在输出对象时不输出"[object Object]"，可以为对象添加一个toString()方法。所以重写p对象的toString方法。

例如：p.toString = function(){

return “name:”+this.name+”age:”+this.age;

}

修改原型的toString，所有的Person对象都适用

Person.prototype.toString = function(){

return “name:”+this.name+”age:”+this.age;

### call和apply

ca11()和apply()这两个方法都是函数对象的方法，需要通过函数对象来调用。当对函数调用ca11()和apply()都会调用函数执行。

在调用ca11（）和apply（）可以将一个对象指定为第一个参数，此时这个对象将会成为函数执行时的this。

# 垃圾回收

垃圾回收（GC）就像人生活的时间长了会产生垃圾一样，程序运行过程中也会产生垃圾，这些垃圾积搂过多以后，会导致程序运行的速度过慢，所以我们需要一个垃圾回收的机制，来处理程序运行过程中产生垃圾。

当一个对象没有任何的变量或属性对它进行引月，此时我们将永远无法操作该对象，此时这种对象就是一个垃圾，这种对象过多会占用大量的内存空间，导致程序运行变慢，所以这种应圾必须进行清理。

在JS中拥有自动的垃圾回收机制，会自动将这些垃圾对象从内存中销毁，我们不需要也不能进行垃圾回收的操作。

我们需要做的只是要将不再使用的对象设置nu11即可。

# 数组：

数组（Array)数组也是一个对象，它和我们普通对象功能类似，也是用来存储一些值的。不同的是普通对象是使用字符串作为属性名的，而数组时使用数字作为索引操作元素。

数组的存储性能比普通对象要好，在开发中我们经常使用数组来存储一些数据。

注；

1、数组中的元素可以时任何类型,也可以是对象,也可以是函数。

var arr = [ “hello”,1,null,undefined,true,obj];

var arr1 =[function(){},function()];

调用函数arr1[0]()

## 数组的声明

var arr=new Array();//声明一个空数组对象

var arr=new Array(length)//声明一个指定长度的数组

var arr=[元素]//声明数组(最常用);<br />

注意：js中的数组声明不用指定长度，js的数组长度是不固定的，会随着元素的数量改变而改变。

## 数组的赋值和取值

数组可以存储任意类型的数据

数组名[角标]=值;//角标可以是任意的正整数或者是0

例如:arr[0] = 10;

arr[1] = 10;

arr[2] = 10;

数组的取出：

数组名[角标]//返回当前角标对应存储的值

console.log(arrr[0]);

如果角标不存在，返回undefined;

## 数组的length属性

length:设置或返回数组的长度。

作用1：数组名.length//返回当前数组的长度。

arr[0] = 1;

arr[2] = 1;

arr[3] = 1;

arr[4] = 1;

arr[100] = 1;

console.log(arr)

返回的结果1,,1,1,1,,,,,,1//没有添加元素的位置用逗号隔开。

注：1、对于连续的数组，使用1ength可以获取到数组的长度（元素的个数），对于非连续的数组，使用1ength会获取到数组的最大的索引+1。

2、尽量不要创建非连续的数组。

作用2：数组名.length=新的值//动态的改变数组的长度

注意：length>原有长度，则使用空进行填充。

length<原有长度，则从后面进行截取，最后的数据会被删除。

应用：

1、arr[arr.length] = 70;可以通过arr.length想数组的最后一个位置添加元素。

## 数组的遍历

### 普通for循环：

for(var i=0;i<arr.length;i++){

alert(arr[i]);

}

|  |
| --- |
| 大于三的输出  var arr =[1,2,3,4,5];  var newArr = getAdult(arr);  function getAdult(arr){  var newArr = [];  for(var i = 0;i<arr.length;i++){  var p = arr[i];  if(p>3){  newArr.push(arr[i]);  }  }  return newArr;  }  for(var i = 0;i<newArr.length;i++){  console.log(newArr[i]);  } |

### for-in：

for(var i in arr){

alert(i);//获取的是角标

}

### for-each

forEach()方法需要一个函数作为参数，像这种函数，由我们创建但是不由我们调用的，我们称为回调函数。数组中有几个元素函数就会执行几次，每次执行时，浏览器会将遍历到的元素，以实参的形式传递进来，我们可以来定义形参，来读取这些内容。

浏览器会在回调函数中传递三个参数：

第一个参数，就是当前正在遍历的元素。

第二个参数，就是当前正在遍历的元素的索引。

第三个参数，就是正在遍历的数组。

arr.forEach(function(value,index,array){

console.log(value);通过上述的形参，对其进行操作。

})

注:该方法只支持ie8以上浏览器。

## 数组的字面量(常采用的用法)

1、使用字面量创建一个数组

语法：[ ]

例如：var arr =[]

2、创建一个数组添加元素。

例如：var arr = [1,2,3,4,5,6];

3、使用造函数创建数组时，也可以同时添加元素，将要添加的元素作文构造函数的参数传递元素之间使用，隔开。

例如:var arr = new arr(20,30,40);

### 区别

var arr = new arr(20);//当传递一个值时，表示创建一个长度为20的数组。

var arr =[20];表示有一个元素，元素的值为20；

## 数组存储的类型

1、数组中的元素可以时任何类型,也可以是对象,也可以是函数。

var arr = [ “hello”,1,null,undefined,true,obj];

存函数：

var arr1 =[function(){alert(1)}, function(){alert(1)];

调用函数arr1[0]()；

存数组

var arr = [ [1,2,3],[4,5,6],[7,8,9,]];

console.log(arr);//打印数组。

## 数组的方法

### push

push：该方法可以向数组的末尾添加一个或多个元素，并返回教组的新的长度。可以将要添加的元素作为方法的参数传递，这样这些元素将会自动添加到数组的末尾。该方法会将数组新的长度作为返回值返回。

例如:var arr =new Array();

var result = arr.push(“sx”,”ss”,”sz”);

### pop

pop：该方法可以删除数组的最后一个元素，并将被删除的元素作为返回值返回

例如:var arr =new Array();

var result = arr.push(“sx”,”ss”,”sz”);

arr.pop();

### unshift

unshift:向数组开头添加一个或多个元素，并返回新的数组长度。向前边添加元素之后，元素的顺序会发生改变。

:var arr =new Array();

var result = arr. unshift(“sx”,”ss”,”sz”);

### shift

shift:可以删除数组的第一个元素，并将被删删除的元素作为返回值返回

例如: :var arr =new Array();

var result = arr. unshift(“sx”,”ss”,”sz”);

var result = arr.shift();

### slice方法

slice()：可以用来从数组提取指定元素，该方法不会改变元素数组，而是将截取到的元素封装到一个新数组中返回。

参数：

1.截取开始的位置的索引，包含开始索引。

2.截取结束的位置的索引，不包含结束索引。

3.第二个参数可以省略不写，此时会截取从开始索引往后的所有元素。

4. 索引可以传递一个负值，如果传递一个负值，则从后往前计算

-1倒数第一个

-2倒数第二个

例如：var arr =["孙悟空","猪八戒","沙和尚","唐僧"];

var result = arr.slice(1,3);

console.log(result);//返回"猪八戒","沙和尚"

### splice方法

splice：删除数组中的指定元素。

-使用splice（）会影响到原数组，会将指定元素从原数组中删除，并将被删除的元素作为返回值返回。会对原数组产生影响。

-参数：

第一个，表示开始位置的索引。

第二个，表示删除的数量到。

第三个及以后，可以传递一些新的元素，这些元素将会自动插入到开始位置索引前边。

例如：var arr =["孙悟空","猪八戒","沙和尚","唐僧"];

var result = arr.slice(0,2,”菩萨”);

console.log(result);//返回"孙悟空","猪八戒"

console.log(arr);// ”菩萨” "沙和尚","唐僧"

### concat

concat：可以连接两个或多个数组，并将新的数组返回，方法不会对原数组产生影响。

连接两个数组：

var arr = ["孙悟空","沙和尚","猪八戒"];

var arr1 = ["赵信","劫","蛮王"];

var arr2 = ["情女","kate","moon"];

var result = arr.concat(arr1,arr2,"vn");

console.log(result);

//"孙悟空,沙和尚,猪八戒,赵信,劫,蛮王,情女,kate,moon,vn"

### join

join():该方法可以将数组转换为一个字符串,方法不会对原数组产生影响，而是将转换后的字符串作为结果返回。

在join（)中可以指定一个字符串作为参数，这个字符串将会成为数组中元素的连接符，如果不指定连接符，则默认使用逗号作为连接符。

var arr = ["孙悟空","沙和尚","猪八戒"];

var str = arr.join(“#”);//String类型。

console.log(str);

### reverse

reverse:该方法用于反转数组。

例如：var arr = ["孙悟空","沙和尚","猪八戒"];

arr.reverse();//该方法会对原数组产生影响。

### sort

sort：对数组的元素进行排序。可以用来对数组中的元素进行排序，也会影响原数组，默认会按照Unicode编码进行排序。

即使对于纯数字的数组，使用sort()排序时，也会按照Unicode编码来排序，所以对数字进排序时，可能会得到错误的结果。

解决方法：我们可以在sort（）添加一个回调函数，来指定排序规则，回调函数中需要定义两个形参，浏览器将会分别使用数组中的元素作为实参去调用回调函数，使用哪个元素调用不确定，但是肯定的是在数组中a一定在b前边。

浏览器会根据回调函数的返回值来决定元素的顺序：

如果返回一个大于e的值，则元素会交换位置。

如果返回一个小于e的值，则元素位置不变。

如果返回一个e，则认为两个元素相等，也不交换位置。

|  |
| --- |
| arr = [5,4,11,65,41,32];  arr.sort(function(a,b){  if(a>b){  return 1;  }else if(a<b){  return -1;  }else{  return 0;  }  }  //升序排列  return a-b;  //降序排列  return b -a;  ) |

### 案例：数组的去重

|  |
| --- |
| //创建数组  var arr = [1,2,3,2,2,1,3,4,2,5];  //获取数组中的每一个元素  for(var i = 0;i<arr.length;i++){  console.log(arr[i]);  //获取当前元素的所有元素  for(var j=i+1;j<arr.length;j++){  if(arr[i]==arr[j]){  arr.splice(j,1);  //删除当前j所在的元素以后，我需要在比较一次所以在位置的元素  j--;  }  } } |

# 对象

## Date对象

### Date

1、创建date对象

var date = new Date();

如果直接使用构造函数创建一个Date对象，则会封装为当前代码执行的时间

2、创建一个指定的时间对象，需要在构造函数中传递一个表示时间的字符串作为参数。

var date = new Date("12/03/2016 11:10:30");//日期的格式月份/日/年时：分：秒

console.log(date);

### Date方法

var date = new Date("12/03/2016 11:10:30");//日期的格式月份/日/年时：分：秒

alert(d.getYear());//返回的是1900年开始距今的年分数

alert(d.getFullYear());//返回的是当前的年份

alert(d.getMonth()+1);//返回的当前月份的角标值，需要+1

alert(d.getDate());//返回的是当前的日期数

alert(d.getDay());//返回的是当前的星期数，但是周天会返回值为

alert(d.getHours());//返回当前时间的小时数

alert(d.getMinutes());//返回当前时间的分钟数

alert(d.getSeconds());//返回当前时间的秒数

### getDay()

-获取当前日期对象时周几

-会返回一个0-6的值

var date = date.getDate();//获取当前对象是周几。

## 包装类

在JS中为我们提供了三个包装类，通过这三个包装类可以将基本数据类型的数据转换为对象

String（)

-可以将基本数据类型字符串转换为String对象

Number()

-可以将基本数据类型的数字转换为Number对象

Boolean()

但是注意：我们在实际应用中不会使用基本数据类型的对象，如果使用基本数据类型的对象，在做一些比较时可能会带来一些不可预期的结果。

主要作用：方法和属性之能添加给对象，不能添加给基本数据类型。当我们对一些基本数据类型的值去调用属性和方法时，浏览器会临时使用包装类将其转换为对象，然后在调用对象的属性和方法。

例如:var s =123;

s = s.toString();这里s是，基本数据类型，不能调用方法。系统自动将它转为对应的包装类对象调用toString()

s.hello=”你好”；//这里添加属性，也是将s变为包装类对象，添加属性。

console.log(s.hello);//undefined，上述的步骤虽然添加了hello属性，但是转成的包装类对象并不是同一个。

所以，各个包装类的方法，基本数据变量都可以调用。

### Number包装类

例如：var num = new Number();

console.log(typeof num);//返回类型Number

### Boolean

var bool = new Boolean(true);

console.log(typeof bool);//返回类型Boolean

### String()

例如：var str = new String();

console.log(typeof str);//返回类型String

## String相关的方法

### 字符串底层原理

字符串在底层是以字符数组的形式保存的，可以使用数组对其实现。

例如：var str = “hello world”;

alert(str[0]);//显示为h

alert(str[3]);//显示为l

alert(str.lengh);//显示为11

### length属性

1ength属性：可以用来获取字符串的长度

### charAt

charAt():可以返回字符串中指定位置的字符。

例如：var str = “hello world”;

var str1 = str.charAt[0];

### charCodeAt

charCodeAt():获取指定位置字符的字符编码（Unicode编码）

例如var str = “hello world”;

var str1 = str. charCodeAt [0];

### formCharCode()

formCharCode()可以根据字符编码去获取字符,该方法通过String去调用。

例如：var result = String.formCharCode(73);//通过字符编码去获取字符。

### concat

concat：可以用来连接两个或多个字符串。

例如：var result = str.concat(“你好”,”再见”);

### indexOf

indexOf:该方法可以检索一个字符串中是否含有指定内容。如果字符串中含有该内容，则会返回其第一次出现的索引,如果没有找到指定的内容，则返回-1。

例如：var str = “hello world”;

var result = str.indexOf(“l”);//返回2

var result = str.indexOf(“l”,2);//指定开始查找的位置。

### lastIndexOf()

lastIndexof():该方法的用法和indexOf（)一样，的是indexOf是从前往后找，而lastIndexOf是从后往前找

### slice

slice:可以从字符串中截取指定的内容，不会影响原字符串。

-参数：

第一个，开始位置的索引（包括开始位置）

第二个，结束位置的索引（不包括结束位置）

如果省略第二个参数，则会截取后边所有的。

-也可以传递一个负数作为参数，负数的话将会从后边计算。

例如：var str = “hello world”;

str.slice(0,2);//

### substring

subString：截取字符串

第一个：开始截取位置的索引（包括开始位置）

第二个：结束位置的索引（不包括结束位置）

不同的是这个方法不能接受负值作为参数，如果传递了一个负值，则默认使用0，而且他还自动调整参数的位置，如果第二个参数小于第一个，则自动交换。

例如：var str = “hello world”;

var result = str.substring(0,2);

### split

split()：可以将一个字符串拆分为一个数组。

注：如果传递一个空串，每个字符将被查分。

例如：var str = “abc,bcd,efg,hig”;

result = str.split(“,”);//使用逗号来分隔字符串，然后返回一个数组。

console.log(Array.isArray(result));//Array

console.log(result[0]);

### 转大小写

toUpperCase()：将一个字符串转换为大写并返回。

toLowerCase()：将一个字符串转换为大写并返回

# 正则表达式

正则表达式用于定义一些字符串的规则，计算机可以根据正则表达式，来检查一个字符串是否符合规则，获取将字符串中符合规则的内容提取出来。

注：正则表达式严格区分大小写。

## 正则对象的创建

语法：var 变量 = new RegExp(“正则表达式”，”匹配模式”);

类型：对象 typeOf：返回的是object。

匹配模式：

在构造函数中可以传递一个匹配模式作为第二个参数，可以是

i忽略大小写

g全局匹配模式

## test方法

test():使用这个方法可以用来检查一个字符串是否符合正则表达式的规则，如果符合则返回true，否则返回false。

var reg = new RegExp(“a”);

var str = “a”;

var result = reg.test(str);//true，检查是否含有a

### 忽略大小写

var reg = new RegExp(“a”,”i”);//这里添加i参数，忽略大小写。

var str = “Abdc”;

var result = reg.test(str);//true，检查是否含有a

## 字面量创建正则

使用字面量来创建正则表达式,创建简单，但是不灵活。

语法：var变量=/正则表达式/匹配模式

例如：var reg = /a/i

案例：创建一个正则表达式，检查一个字符串中是否有a或b

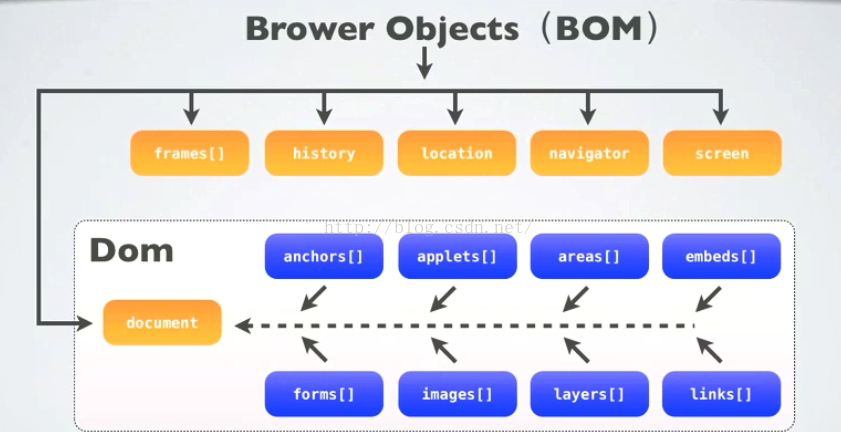
var reg = /a|b/;

var str = “Abdc”;

var result = reg.test(str);

# windows对象

## BOM(Browser Object Model)



BOM浏览器对象模型：是规范浏览器对js语言的支持(js调用浏览器本身的功能)，提供了独立于内容而与浏览器窗口进行交互的对象。BOM的具体实现是window对象。

Window 对象表一个浏览器窗口或一个框架。在客户端JavaScript中，Window 对象是全局对象，要引用当前窗口根本不需要特殊的语法，可以把那个窗口的属性作为全局变量来用。在使用window对象的方法和属性的时候，可以不带window，比如window.alert(“韩顺平！”），可以直接写成alert（“韩顺平”）;

作用：在BOM中为我们提供了一组对象，用来完成对浏览器的操作。

Window：代表的是整个浏览器的窗口，同时window也是网页中的全局对象。

Window对象包含属性：document、location、navigator、screen、history、frames

Navigator：代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器。网景王思的第一款浏览器，航海家。

Location：代表当前浏览器的地址栏信息，通过Location可以获取地址栏信息，或者操作浏览器跳转页面。

History：代表浏览器的历史记录，可以通过该对象来操作浏览器的历史记录。由于隐私原因，该对象不能获取到具体的历史记录，只能操作浏览器向前或向后翻页，而且该操作只在当次访问时有效。

Screen：代表用户的屏幕的信息，通过该对象可以获取到用户的显示器的相关的信息。

这些BOM对象在浏览器中都是作为window对象的属性保存的，可以通过window对象来使用，也可以直接使用。

例如：

## windon对象

Window 对象是 JavaScript 层级中的顶层对象。

Window 对象代表一个浏览器窗口或一个框架。

Window 对象会在 <body> 或 <frameset> 每次出现时被自动创建。

## 框体方法

1. alert:警告框，提示一个警告信息，没有返回。调用的是浏览器封装的方法。

调用：window.alert("我是警告框");

2 confirm:确认框，提示用户选择一项操作（确定/取消）

点击确定 返回true

点击取消 返回false

代码window.confirm("你确定要删除吗?"); 弹出提示框是否删除

3. prompt:提示框，提示用某个信息的录入或者说收集。

点击确定，返回当前用书录入的数据，默认返回空字符串。

点击取消,返回null。

代码：window.prompt("请输入昵称：");例如游戏中输入游戏名称等。

4.定时执行：setTimeout(code,millisec)

参数1：函数对象，参数2：时间，单位毫秒。返回值：返回当前间隔器的id

作用：方法用于在指定的毫秒数后调用函数或计算表达式。

代码：function testSetTimeout(){ //alert(会立即执行。)

window.setTimeout(function testdelete(){

alert(window.confirm("您确定要删除吗"));

},3000);

}

5间歇调用：setInterval (code,millisec)

参数1：函数对象，参数2：时间，单位毫秒。返回值：返回当前间隔器的id

作用：每间隔指定的时间执行指定的函数。

代码：function test(){

alert("间隔两秒执行");

}

function testInterval(){

window.setInterval(function test(){

alert("间隔两秒执行");

},2000);

}

<input type="button" name="" id="" value="2s之后执行" onclick="testInterval()"/>

6.clearTimeout(number id):

参数：定时器的id，此id可以在js中创建一个全局变量，变量接收来自setTimeout返回的id值。

作用：在还没执行前，将该定时器结束，

代码：function testClearTimeout(){

window.clearTimeout(idi);

}

7.clearInterval:用来停止指定的间隔器

参数：间隔器的id，需要创建全局变量来接收来自定时器的id。

作用：用来停止指定的间隔器

代码：function testClearInterval(){

window.clearInterval(ids);

### window.onload()

window.onload出现的原因？

 我们都知道页面的代码顺序是从上往下进行加载，很多时候我们要对页面中的某一个模块进行操作，这时候我们常常使用javascript代码来进行操作。为了能够保证操作的模块或对象在js代码之前就加载了，我们不得不把js代码放在页面的底端。但是我们在设计页面的时候，为了把js代码放在一起，或者一个让页面更加简洁的位置，那就有可能出现代码中操作的对象未被加载的情况，那么我们该如何去解决呢？这时候window.onload就被有了存在的意义了。

window.onload是一个事件，在文档加载完成后能立即触发，并且能够为该事件注册事件处理函数。将要对对象或者模块进行操作的代码存放在处理函数中。即：window.onload =function (){这里写操作的代码};

|  |
| --- |
| <script type="text/javascript">  document.getElementById("s").style.color="green";  </script>  </head>  <body>  <span id="s">要坚强，当然是选择原谅他啊！！！</span>  </body>  </html> |

使用window.onload()方法

|  |
| --- |
| <title>使用window.onload()，操作对象 已被加载</title>  <script type="text/javascript">  window.onload = function(){  document.getElementById("s").style.color="green";  };  </script>  </head>  <body>  <span id="s">要坚强，当然是选择原谅他啊！！！</span>  </body>  </html> |

### 子窗口方法

如果文档包含框架（frame 或 iframe 标签），浏览器会为 HTML 文档创建一个 window 对象，并为每个框架创建一个额外的 window 对象。

1、子页面方法

格式：window.open(pageURL,name,parameters)

参数pageURL为子窗口路径，name为子窗口句柄，parameters为窗口参数(各参数用逗号分隔)。

作用：打开一个新的窗口

function testOpen(){

window.open('son.html','newwindow','height=400, width=600,top=100px,left=320px,toolbar=yes,menubar=yes,scrollbars=yes,resizable=yes,location=no, status=yes');}、

2、window.close()

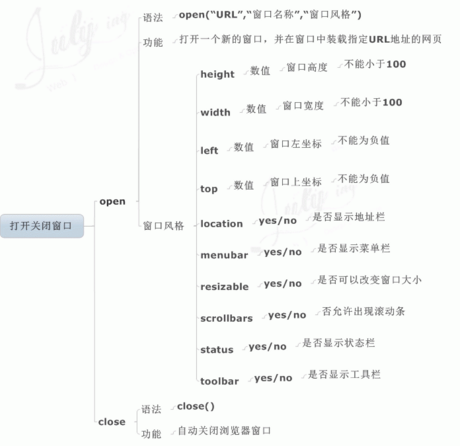
作用：关闭子页面，

注：关闭子页面的方法window.close(),但是此方法只能关闭open方法打开的子页面。

案例：倒计时功能

代码：

|  |
| --- |
| <script type="text/javascript">  /\* 倒计时功能\*/  function testTime(){  window.setInterval(function(){  var span = document.getElementById("timeSpan");  span.innerHTML = span.innerHTML-1;  if(span.innerHTML==0){  window.close();//实际是资源的跳转。  }  },1000);  }  </script>  <body onload="testTime()"> <!--当页面加载成功后触发该子页面。-->  <br />  <b>欢迎来到子页面，<span id="timeSpan">5</span>秒后自动关闭</b>  </body> |



3、子页面调用父页面的函数

格式：window.opener.父页面的函数

作用：当子页面添加完信息之后，点击确认，重新加载父页面的资源，显示更新的内容。

代码：

### js的window对象的常用属性

windows对象还有其他对象，其他的属性。

地址栏相当于一个文本框，

1.地址栏属性:location对象

1、window.location.href="新的资源路径（相对路径/URL）"

让当前页面的资源跳转。

修改当前资源路径的三种方式

(1)直接修改浏览器地址。

(2)超链接

(3)通过js来动态的改变资源地址。

2、window.location.reload()重新加载页面资源

功能：刷新此页面

3、历史记录属性

window.history.forward() 页面资源前进，历史记录的前进。

作用：跳转到历史记录的下个页面

window.history.back() 页面资源后退，历史记录后退

作用：跳转到历史记录的上个页面

window.history.go(index)

作用：跳转到指定的历史记录资源

注意window.history.go(0)相当于刷新。

4、屏幕属性

window.srceen.width;//获取屏幕的宽度分辨率

window.screen.height;//获取屏幕的高度分辨率

作用：根据浏览器的大小设置屏幕的大小。

function test(){

var widhth = window.screen.width;

var height = window.screen.width;

alert(widhth+":"+height);

}

5、浏览器配置属性

window.navigator.userAgent

获取浏览器的属性

function testNa(){

alert(window.navigator.userAgent);

}

显示内容：Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.12 Safari/537.36

6、主体面板属性(document)

对象里面分装对象，将html中的所有标签都变为对象，父节点封装子节点。

### 定时器

如果希望一段程序，可以每间隔一段时间执行一次，可以使用定时调用。

setInterval()

-定时调用

-可以将一个函数，每隔一段时间执行一次

-参数：

1.回调函数，该函数会每隔一段时间被调用一次

2.每次调用间隔的时间，单位是毫秒

返回值：

返回一个Number类型的数据。

这个数字用来作为定时器的唯一标识。

clearInterval（)可以用来关闭一个定时器，方法中需要一个定时器的标识作为参数，这样将关闭标识对应的定时器。

clearInterval()可以接收任意参数，如果参数是一个有效的定时器的标识，则停止对应的定时器。如果参数不是一个有效的标识，则什么也不做。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //获取count  var count = document.getElementById("count");  //使count中的内容自动切换  /\*for(var i = 0;i<10;i++){  //间隔一段时间执行一次代码  count.innerHTML = i;  }\*/  var num = 1;  var timer = setInterval(function(){  count.innerHTML = ++num;  if(num == 1000)  clearInterval(timer);  },1);  };  </script>  <style type="text/css">  \*{margin:0; padding: 0;}  </style>  </head>  <body>  <h1 id="count">1</h1>  </body>  </html> |

### 切换图片

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //获取img标签  var img1 = document.getElementById("img");  var timer = null;  //获取开始标签  var btn01 =document.getElementById("btn01");    btn01.onclick = function(){  //目前，我们每点击一次按钮，就会开启一个定时器，  //点击多次就会开启多个定时器，这就导致图片的切换速度过快，  //并且我们只能关闭最后一次开启的定时器  //开启前关闭定时器，就不会有多个定时器在跑  clearInterval(timer);  //创建哪一个数组来保存图片  var imgArr = ["img/1.jpg","img/2.jpg","img/3.jpg","img/4.jpg","img/5.jpg"];  //创建一个变量，用来保存当前图片的索引  var index = 0;  //开启一个定时器自动切换图片  timer = setInterval(function(){  //使索引自增  index++;  //判断索引是否超过最大索引  /\*if(index>=imgArr.length){  index =0;  }\*/  index = index%imgArr.length  img1.src = imgArr[index];  },300);    }  //暂停操作  //获取暂停标签  var btn02 =document.getElementById("btn02");  btn02.onclick = function(){  clearInterval(timer);  }  }    </script>  </head>  <body>  <img id = "img" src="img/1.jpg"/>  <button id="btn01">开始</button>  <button id="btn02">暂停</button>  </body>  </html> |

## Navigator

Navigator：代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器。由于历史原因，Navigator对象中的大部分属性都已经不能帮助我们识别浏览器了。一般我们只会使用userAgent来判断浏览器的信息。userAgent是一个字符串，这个字符串中包含有用来描述浏览器信息的内容，不同的浏览器会有不同的userAgent。

### 属性

appName

alert(navigator.appName)//Net

userAgent：可以通过该值判段浏览器。

alert(navigator.userAgent)

返回结果："Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.138 Safari/537.36"

在IE11中已经将微软和IE相关的标识都已经去除了，所以我们基本已经不能通过UserAgent来识别一个浏览器是否是IE了。

如果通过UserAgent不能判断，还可以通过一些浏览器中特有的对象，来判断浏览器的信息。例如:ActiveXObject 对象只存在在ie中。

判断浏览器

|  |
| --- |
| var bro = navigator.userAgent;  if(/firfox/i.test(bro)){  alert("火狐");  }else if(/chrome/i.test(bro)){  alert("你是chrom")  }else if(/msie/i.test(bro)){  alert("你是ie");  }else if("ActiveXObject" in window){//不能直接用window.ActiveXObject判断  alert("你是ie11");//微软对window.ActiveXObject动过手脚，会返回false、  } |

## History

History：对象可以用来操作浏览器向前或向后翻页。

### 属性和方法

length：用来获取当前访问的访问过的页面数。

例如：history.length

back()：可以用来回退到上一个页面，作用和浏览器的回退按钮一样。

forward()：可以跳转下一个页面，作用和浏览器的前进按钮一样。‘

go()：可以用来跳转到指定的页面。

-它需要一个整数作为参数。

1：表示向前跳转一个页面相当于forward()。

2：表示向前跳转两个页面。

-1：表示向后跳转一个页面。

-2：表示向后跳转两个页面。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  var btn = document.getElementById("btn");  btn.onclick = function(){  history.back();  history.forward();  history.go(2);  history.go(3);  }  }  </script>  </head>  <body>  <button id="btn">点一下 </button>  <a href="History.html">连接</a>  </body>  </html> |

## Location

window对象的location属性引用是Location对象，表示当前显示的的文档的URL。Location对象的href属性是一个字符串。Location对象的toString()方法返回href属性的值，因此会在隐式调用toString()情况下，使用location代替Location.href。

### 属性和对象

Location：该对象中封装了浏览器的地址栏的信息。

### 方法

如果直接打印localtion，则可以获取到地址栏的信息（当前页面的完整路径）。

console.log(location);

显示结果：<http://127.0.0.1:8020/Test/Location.html>

如果直接将1ocation属性修改为一个完整的路径，或相对路径则我们页面会自动跳转到该路径，并且会生成相应的历史记录location = "http://www.baidu.com";

assign():用来跳转到其他的页面，作用和直接修改location一样。

例如：location.assign("http://www.baidu.com");

reload()：用于重新加载当前页面，作用和刷新按钮一样

例如:location.reload(“true’);// 如果在方法中传递一个true，作为参数，则会强制清空缓存刷新页面。

replace():可以使用一个新的页面替换当前页面，调用完毕也会跳转页面。不会生成历史记录，不能使用回退按钮回退。

## 定时器

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //获取count  var count = document.getElementById("count");  //使count中的内容自动切换  /\*for(var i = 0;i<10;i++){  //间隔一段时间执行一次代码  count.innerHTML = i;  }\*/  var num = 1;  var timer = setInterval(function(){  count.innerHTML = ++num;  if(num == 1000)  clearInterval(timer);  },1);  };  </script>  <style type="text/css">  \*{margin:0; padding: 0;}  </style>  </head>  <body>  <h1 id="count">1</h1>  </body>  </html> |

### 切换图片

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //获取img标签  var img1 = document.getElementById("img");  var timer = null;  //获取开始标签  var btn01 =document.getElementById("btn01");    btn01.onclick = function(){  //目前，我们每点击一次按钮，就会开启一个定时器，  //点击多次就会开启多个定时器，这就导致图片的切换速度过快，  //并且我们只能关闭最后一次开启的定时器  //开启前关闭定时器，就不会有多个定时器在跑  clearInterval(timer);  //创建哪一个数组来保存图片  var imgArr = ["img/1.jpg","img/2.jpg","img/3.jpg","img/4.jpg","img/5.jpg"];  //创建一个变量，用来保存当前图片的索引  var index = 0;  //开启一个定时器自动切换图片  timer = setInterval(function(){  //使索引自增  index++;  //判断索引是否超过最大索引  /\*if(index>=imgArr.length){  index =0;  }\*/  index = index%imgArr.length  img1.src = imgArr[index];  },300);    }  //暂停操作  //获取暂停标签  var btn02 =document.getElementById("btn02");  btn02.onclick = function(){  clearInterval(timer);  }  }    </script>  </head>  <body>  <img id = "img" src="img/1.jpg"/>  <button id="btn01">开始</button>  <button id="btn02">暂停</button>  </body>  </html> |

## 延时调用（setTimeout）

延时调用：延时调用一个函数不马上执行，而是隔一段时间以后在执行，而且只会执行一次

延时调用和定时调用的区别，定时调用会执行多次，而延时调用只会执行一次。

延时调用和定时调用实际上是可以互相代替的，在开发中可以根据自己需要去选择。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //延时调用  var num = 1;  var timer = setTimeout(function(){  console.log(num);  },3000);  }  </script>  </head>  <body>  </body>  </html> |

### 案例：点击之后div向右移动

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <style type="text/css">  \*{  margin: 0px;  padding:0px;  }  #box1{  width: 100px;  height: 100px;  background-color: red;  position: absolute;  left = 0;  }    </style>  <script type="text/javascript">  window.onload = function(){  //获取box1  var box = document.getElementById("box1");  //获取btn01  var btn01 = document.getElementById("btn01");  //点击按钮之后，使box1向右移动增加(left值增大)  //定义一个变量，用来保存定时器的标识  var timer;  btn01.onclick = function(){  //开启定时器执行动画效果  //关闭上一个定时器  clearInterval(timer);  timer = setInterval(function(){  //没有设置left获得auto，获得px，需要将它转为number，使用parseInt()，获取合法数字字符  //var oldVlaue = parseInt(getComputedStyle(box1,null)["left"]);  var oldVlaue =parseInt(getComputedStyle(box1,null)["left"]);  var newValue = oldVlaue+13;  //判断newvalue是否大于800,让停的位置始终等于800  if(newValue>800){  newValue=800;  }  box1.style.left = newValue+"px";  //当元素的值为800的时候，关闭定时器    if(newValue800){  clearTimeout(timer);  }  },30);  }  };  </script>  </head>  <body>  <button id="btn01">点击按钮之后box1向右移动</button>  <div id="box1">    </div>  </body>  </html> |
|  |

# document对象

## DOM

### 引入

需求：

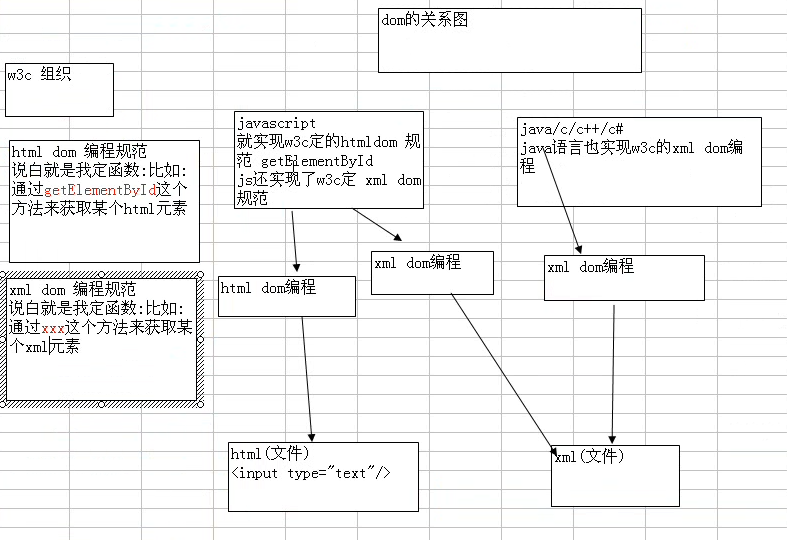
DOM：Document Object Model(文本对象模型），根据W3C DOM规范，DOM是HTML与XML的应用编程接口（APl），DOM将整个页面映射为一个由层次节点组成的文件。

js把浏览器，网页文档和网页文档中的html元素都用相应的内置对象来表示，这些对象与对象间的层次关系称为dom，针对网页（htm1，jsp，php，aspx.net)的dom就是htmldom。我们这讲的就是html dom。

在一个浏览器窗口中可能有多个 document，例如，通过 iframe 加载的页面，每一个都是一个 document。

由于BOM的window包含了document，因此可以直接使用window对象的document属性，通过document属性就可以访问、检索、修改XHTML文档内容与结构。因为document对象又是DOM（Document Object Model）模型的根节点。

可以说，BOM包含了DOM(对象)，浏览器提供出来给予访问的是BOM对象，从BOM对象再访问到DOM对象，从而js可以操作浏览器以及浏览器读取到的文档。



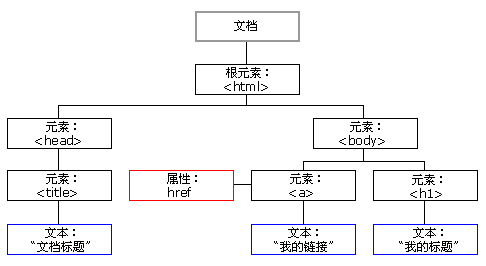
由于javascript ，xml 和 html是由不同的人开发的，javascript不能直接操作 html和xml文档，不能直接操作怎么办呢？W3C就出来制定了一个规范，也就是dom规范，W3c规定了一些函数，可以通过这些函数获取某个html元素，也可以获取某个xml元素，然后由javascript开发人实现这些函数，那么也就实现了由javascript来操作html和xml文件。由于html和xml是不一样的，所以也就出现了 html dom编程和 xml dom编程两种规范。

作用：js把浏览器，网页文档和网页文档中的html元素都用相应的内置对象来表示，这些内置对象就是dom对象，编程人员通过访问dom对象，就可以实现对浏览器本身、网页文档、网页文档中元素的操作，从而控制浏览器和网页元素的行为和外观。

文档：文档表示的就是整个的HTML网页文档。

对象：对象表示将网页中的每个部分都转换为了个对象。

模型：使用模型来表示对象之间的关系，这样方便我们获取对象。



### 代码的执行顺序

浏览器在加载一个页面时，是按照自上向下的顺序加载的，读取到一行就运行一行，如果将script标签写到页面的上边，在代码执行时，页面还没有加载。将js代码编写到页面的下部就是为了，可以在页面加载完毕以后再执行js代码。

如果想要在body前面添加js代码。需要通过事件绑定函数，函数中放要执行的代码。只有html 页面加载完成之后餐能触发该事件，可以使用onload。onload事件会在整个页面加载完成之后才触发为window绑定一个onload事件。该事件对应的响应函数将会在页面加载完成之后执行，这样可以确保我们的代码执行时所有的DOM对象已经加载完毕了。

例如：window.onload = function{ 代码内容};

最佳：写在body里面，性能好一点。写在head中加载了但是不执行。

## 节点

节点Node，是构成我们网页的最基本的组成部分，网页中的每一个部分都可以称为是一个节点。比如：html标签、属性、文本、注释、整个文档等都是一个节点。虽然都是节点，但是实际上他们的具体类型是不同的。比如：标签我们称为元素节点、属性称为属性节点、文本称为文本节点、文档称为文档节点。节点的类型不同，属性和方法也都不尽相同。

节点：Node——构成HTML文档最基本的单元。

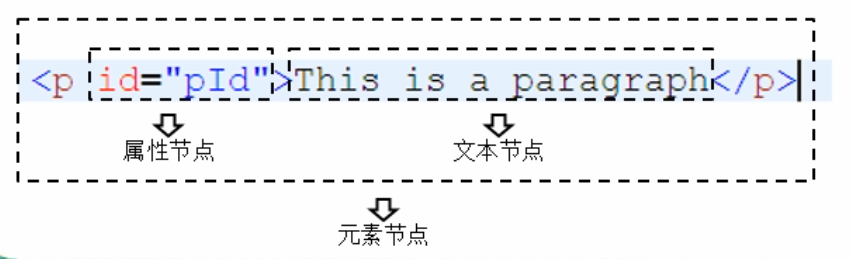
### 常用节点分为四类

文档节点：整个HTML文档

元素节点：HTML文档中的HTL标签

属性节点：元素的属性

文本节点：HTML标签中的文本内容



### 节点属性



节点属性：nodeName nodeType nodeValue

节点类型：通过节点类型可以判断节点是哪种类型。

文档节点：文档节点document，代表的是整个HTML文档，网页中的所有节点都是它的子节点。document对象作为window对象的属性存在的，我们不用获取可以直接使用。通过该对象我们可以在整个文档访问内查找节点对象，并可以通过该对象创建各种节点对象。

元素节点：HTML中的各种标签都是元素节点，这也是我们最常用的一个节点。浏览器会将页面中所有的标签都转换为一个元素节点，我们可以通过document的方法来获取元素节点。比如：

-document.getElementByld0

-根据id属性值获取一个元素节点对象。

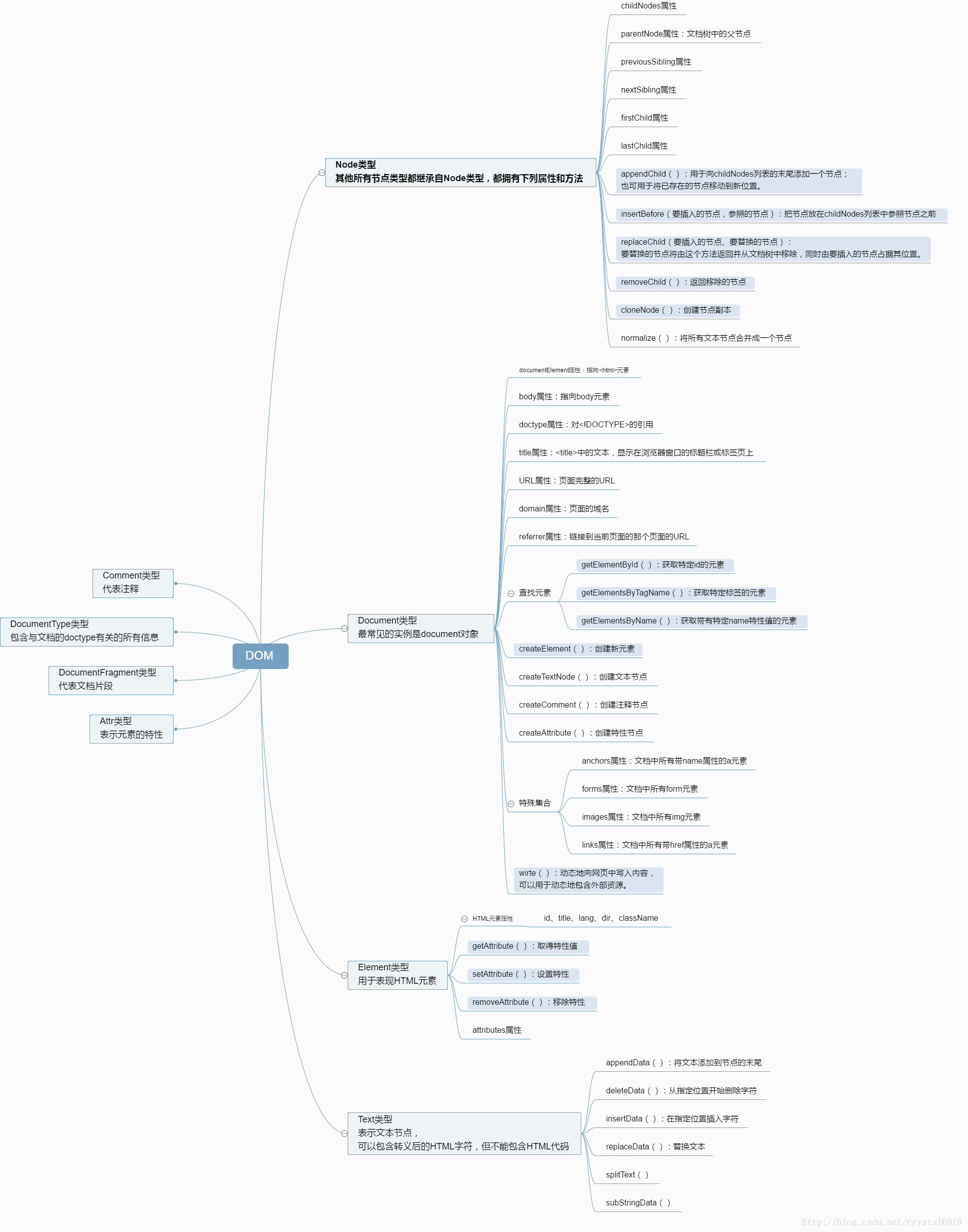
属性节点：文本节点表示的是HTML标签以外的文本内容，任意非HTML的文本都是文本节点。它包括可以字面解释的纯文本内容。文本节点一般是作为元素节点的子节点存在的。获取文本节点时，一般先要获取元素节点。在通过元素节点获取文本节点。

例如：

-元素节点firstChild；

-获取元素节点的第一个子节点，一般为文本节点

### dom对象层次图



### dom对象的加载过程

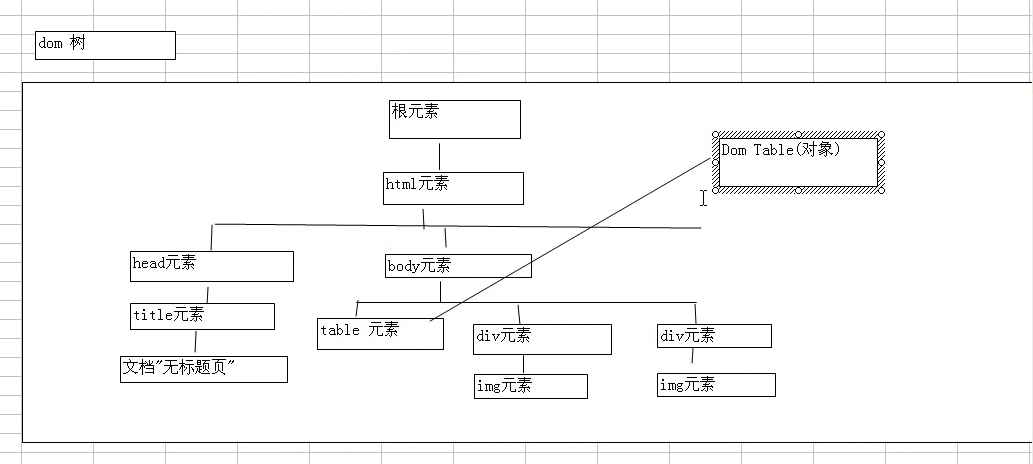
浏览器加载html页面，会在浏览器内部创建document对象，使用对象进行操作，操作的是内部创建的document对象封存的信息，最后它会更新源码。 不存在使用document修改html源码。

## document树

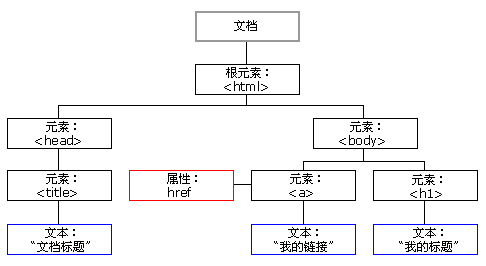
### 代码

|  |
| --- |
| <html>  <head>  </head>  <body onkeydown="move(event)">  <table border="1">  <tr>  <td></td>  <td><input type="button" value="向上走" onclick ="move(this)"/></td>  <td></td>  </tr>    <tr>  <td><input type="button" value="向左走" onclick ="move(this)"/></td>  <td></td>  <td><input type="button" value="向右走" onclick ="move(this)"/></td>  </tr>    <tr>  <td></td>  <td><input type="button" value="向下走" onclick ="move(this)"/></td>  <td></td>  </tr>  </table>  <div style="width:500px;height:400px;background-color:pink">  <img id="wugui" src="wugui.png" border="1" style="position:absolute;width:50px;height:40px;left:100px;top:120px"/>  <img id="cock" src="ji.png" border="1" style="position:absolute;width:50px;height:40px;left:200px;top:200px"/>  </div>  </body>  </html> |

### 上述diamante的dom树



### html的dom层次图



dom编程时，html文档里的table，img，title等等就是内置对象，这个可以使用window.alert();函数输出。这些内置的对象提供了很多方法，这样就很方便的使用了。

## 获取元素节点

### 获取元素节点的方式

直接获取方式：

通过id：通过id属性获取一个元素节点对象。(且这元素时唯一的)

通过name属性值

通过标签名

通过class属性值

间接获取方式：

父子关系

子父关系

兄弟关系

### 通过id方式获取

方法：getElementById(id);

注：1.id的一般只有一个，但是可以定义多个相同的id，通过这个方法获取id的标签内容时，即使有多个相同id，也只会获取一个，并且只会获取相同id的第一个id。

2. id号不要用数字开头

|  |
| --- |
| function test(){  var myhref=document.getElementById("a1");//其实它只找一个，只找第一个，找到第一个后就返回了，不往后执行了  window.alert(myhref);  }  <!-- 故意定义了三个id号相同的超链接 -->  <a id="a1" href="http://www.baidu.com">连接到baidu</a>  <a id="a1" href="http://www.sina.com">连接到sohu</a>  <a id="a1" href="http://www.163.com">连接到163</a> |

### 通过name方式获取

方法：getElementsByName(name);

注：1注意是Elements 不是 Element。

|  |
| --- |
| function test2() {//id不能重复，但是name可以重复 var hobbies=document.getElementByName("hobby"); //window.alert(hobbies.length); for(var i=0;i<hobbies.length;i++) { //如何判断是否选择 if(hobbies[i].checked){ window.alert("你的爱好是"+hobbies[i].value); } } }  <input type="checkbox" name="hobby" value="足球"/> 足球  <input type="checkbox" name="hobby" value="旅游"/> 旅游  <input type="checkbox" name="hobby" value="音乐"/> 音乐  <input type="button" value="testing" onclick="test2()"/> |

### 通过标签名方式获取

方法：getElementsByTagName(tagName); 通过标签名获取一组元素节点对象。相同标签名有多个。

|  |
| --- |
| //通过标签名来获取对象（元素）  function test3(){  var myObjs=document.getElementsByTagName("input"); for(var i=0;i<myObjs.length;i++) { window.alert("input:"+myObjs[i].value);  }}  <input type="checkbox" name="hobby" value="足球"/> 足球  <input type="checkbox" name="hobby" value="旅游"/> 旅游  <input type="checkbox" name="hobby" value="音乐"/> 音乐  <input type="button" value="获取所有input" onclick="test3()"/> |

### 通过class属性值方式获取

方法：getElementsByClassName(class)

function testGetEleByClassName(){

var inps=document.getElementsByClassName("common");

alert(inps.length);

}

缺点：ie9以上的浏览器才支持。

### dom对象的节点属性和方法

在dom编程中，一个html文档会被当作 dom 树来对待，dom会把所有的 html 元素映射成Node节点，于是你就可以使用Node节点（对象）的属性和方法。

### innerText和innerHtml属性。

innerText：返回dom元素的文本。

innerHtml：返回dom元素的标签。

相同点：都可以获取文本内容。

### 练习

|  |
| --- |
| //1、通过btn01按钮查找#bj节点  //1、为id为btn01的按钮绑定一个单机响应函数。  var btn01 = document.getElementById("btn01");  btn01.onclick = function(){//获取id为bj的节点  var bj = document.getElementById("bj");  //通过innerHTML属性可以获取到元素内部的html的代码  alert(bj.innerHTML);  }  //查找所有1i节点  //1、为id为btn02的按钮绑定一个单机响应函数。  var btn02 = document.getElementById("btn02");  btn02.onclick = function(){  //通过标签名获取标签名为li的节点,这个返回一个类数组对象。  //即使返回的值有一个，也是一个数组。  var li = document.getElementsByName("li");  //获取数组长度  alert(li.length);  //遍历数组  for(var i= 0;i<li.length;i++){  alert(li[i].innerHTML);  }  }  //查找name=gender的所有节点  var btn03 = document.getElementById("btn03");  btn03.onclick = function(){  var inputs = document.getElementsByName("gender");  alert(inputs.length);  for(var i= 0;i<inputs.length;i++){  //input这个标签是自结束标签，没有html代码，所以没有意义。  //如果需要读取元素节点属性，可以直接使用  //元素.属性 例如：元素.id 元素.name 元素.value  //但是注意：class不能采用这种方式，可以使用 元素.className  alert(inputs[i].value);  alert(inputs[i].className);  }  } |

### 案例：图片切换练习

## 获取元素节点的子节点

### 获取子节点的方法：

通过具体的元素节点调用

1.getElementsByTagName)

-方法，返回当前节点的指定标签名后代节点

2.childNodes

-属性，表示当前节点的所有子节点，会包括文本的所有节点,标签之间的空格也会被当做文本节点。注意：这个属性在ie8浏览器中不会返回空白文本。

3.firstChild

-属性，表示当前节点的第一个子节点

4.lastChild

-属性，表示当前节点的最后一个子节点

5children

属性：children属性可以获取当前元素的所有子元素。

6. firstElementChild

属性：获取当前元素的第一个子元素

注：firstElementChild不支持IE8及以下的浏览器，果需要兼容他们尽量不要使用。

### getElementsBy TagName)

|  |
| --- |
| //为id为btn04的按钮绑定一个单机响应函数  var btn04 = document.getElementById("btn04");  btn04.onclick = function(){  //获取id为city的元素  var city = document.getElementById("city");  //查找#city下所有的li节点  var lis = city.getElementsByTagName("li");  for(var i = 0li<lis.length;i++){  alert(lis[i].innerHTML);  }  } |

### childNodes

## 获取父节点和兄弟节点

### 属性：

parentNode：属性，表示当前节点的父节点

previousSibling：属性，表示当前节点的前一个兄弟节点

nextSibling：属性，表示当前节点的后一个兄弟节点

### parentNode

|  |
| --- |
| 返回id为bj的父节点  myClick("btn07",function(){  //获取id为bj的节点  var bj = document.getElementById("bj");  //返回bj的父节点  var pn = bj.parentNode;    })  回调函数//封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

### previousSibling(能够获得空白的文本)

-属性，表示当前节点的前一个兄弟节点

|  |
| --- |
| function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  }  myClick("btn",function(){  //获取id为bj的前一个节点  var bj = document.getElementById("bj");  //返回bj的父节点  var pn = bj.previousSibling;//获得id为bj的前一个节点  var pn1 = bj.previousElementSibling;//获取前一个兄弟元素。  }) |

### nextSibling

-属性，表示当前节点的后一个兄弟节点

## 其他方法

### 获取body标签

通过getElementsByTagName()获取，body标签只有一个。

var body = document.getElementsByTagName("body")[0];

console.log(body);// "[object HTMLBodyElement]"

在document中有一个属性body，它保存的是body的引用

var body = document.body;

console.log(body); // "[object HTMLBodyElement]"

### 获取根标签

在document中有一个属性documentElement，它保存的是html的引用

var html = document.documentElement;

console.log(html);//"[object HTMLHtmlElement]"

### document的all属性

all：是一个数组，存储页面中所有的元素。

|  |
| --- |
| var all =document.all;  for(var i = 0;i<all.length;i++){  console.log(all[i]);  } |

获取所有元素的第二种方式：

var all = document.getElementsByName("\*");

### 获取所有的div

var divs = document.getElementsByName("div");

### 获取class为box的div

document.querySelector()：需要一个选择器的字符串作为参数，可以根据一个CSS选择器来查询一个元素节点对象。

例如：var div1 = document.querySelector(".box div");

虽然IE8中没有getElementsByClassName()但是可以使用querySelector()代替。使用该方法总会返回唯一的一个元素，如果满足条件的元素有多个，那么它只会返回第一个。

### document.querySelectorA11()

-该方法和querySelector（)用法类似，不同的是它会将符合条件的元素封装到一个数组中返回

-即使符合条件的元素只有一个，它也会返回数组。

例如：var div1 = document.querySelectorAll(".box div");

## 操作对象

### 操作html 固有元素属性（查看和修改）

可以获取和id和name值，但一般一会修改这两个值，修改id可能会导致css样式的变化，修改name值，后台通过name获取值可能找不到。

通过这种方式不能查找和修改， 用户自定义的属性。

获取属性值：元素对象.属性

|  |
| --- |
| //  function testField(){  //获取元素对象  var inp=document.getElementById("uname");  //获取元素属性值  alert(inp.type+":"+inp.name+":"+inp.id+":"+inp.value);  } |

修改元素属性值：元素对象.属性 = 属性值。

|  |
| --- |
| function testField2(){  //获取元素对象  var inp=document.getElementById("uname");  //修改元素属性  inp.value="哈哈";  inp.type="button";  } |

2自定义属性

获取自定义属性：对象. getAttribute(自定义的属性);

|  |
| --- |
| //获取  function testOwnField(){  //获取元素对象  var inp=document.getElementById("uname");  //获取自定义属性的值  alert(inp.getAttribute("abc"));//获取到“嘿嘿”    用户名 : <input type="text" name="uname" id="uname" value="" abc="嘿嘿"/> |

修改自定义属性对象. setAttribute(自定义的属性);

|  |
| --- |
| function testOwnField2(){  //获取元素对象  var inp=document.getElementById("uname");  //修改自定义属性的值  inp.setAttribute("abc","呵呵");  }  用户名 : <input type="text" name="uname" id="uname" value="" abc="呵呵"/> |

使用自定义方式操作固有属性

|  |
| --- |
| function testOper(){  //获取元素对象  var inp=document.getElementById("uname");  //操作对象属性  alert(inp.getAttribute("type"));  alert(inp.getAttribute("value"));  } |

### 操作元素内容

获取元素对象

元素对象名.innerHTML//返回当前元素对象的所有内容，包括HTML标签

元素对象名.innerText//返回当前元素对象的文本内容，不包括HTML标签

修改

元素对象名.innerHTML="新的值"//会将原有内容覆盖，并HTML标签会被解析

元素对象名.innerHTML=元素对象名.innerHTML+"新的值"//追加效果

元素对象名.innerText="新的值"//会将原有内容覆盖，但HTML标签不会被解析，会作为普通文本显示。

### 通过js修改css的样式

需求：在浏览网页的过程中修改背景颜色或者更换某个网站的背景。

js操作元素样式：

1.获取元素对象

2.通过style属性

元素对象名.style.样式名="样式值"//添加或者修改

元素对象名.style.样式名=""//删除样式

注意:

以上操作，操作的是HTML的style属性声明中的样式。而不是其他css代码域中的样式。如果style没有了，他会从其他的样式去寻找，style的级别最高。

3通过className

元素对象名.className="新的值"//添加类选择器样式或者修改类选择器样式

元素对象名.className=""//删除类样式。

|  |
| --- |
| <script type="text/javascript">  //js操作元素样式  //js给元素操作样式---style  function testOperCss(){  //获取元素对象  var showdiv=document.getElementById("showdiv");  //添加元素样式  showdiv.style.backgroundColor="#FFA500";    //js修改元素样式  showdiv.style.border="solid 2px red";  //js删除样式  showdiv.style.border="";  删除样式是删除的style属性添加的样式，而css中的样式可能还存在。依然会有样式。  }  //js操作样式--className  function testOperCss2(){  //获取元素对象  var div01=document.getElementById("div01");  //获取  alert(div01.className);  //添加或者修改  div01.className="common2";  //删除  div01.className="";    }  </script> |

## 操作文档结构(dom增删改)

### 方法

document.createElement(0)：可以用于创建一个元素节点对象，它需要一个标签名作为参数，将会根据该标签名创建元素节点对象，并将创建好的对象作为返回值返回。

document.createTextNode()可以用来创建一个文本节点对象，需要一个文本内容作为参数，将会根据该内容创建文本节点，并将新的节点返回。

appendChild()向一个父节点中添加一个新的子节点

用法：父节点.appendChild（子节点）；

insertBefore():可以在指定的子节点前插入新的子节点

语法：父节点.insertBefore（新节点,旧节点）;

replaceChild()：可以使用指定的子节点替换已有的子节点

语法：父节点.replaceChild（新节点,旧节点）;

1、增加节点

|  |
| --- |
| //添加节点  //例如创建一个广州节点,添加到#city下  myClick("btn01",function(){  //创建广州节点<li>广州</li>  //创建li元素节点对象  var li = document.createElement("li");  //创建文本节点  var gzText = document.createTextNode("广州");  //将文本节点添加到li的子节点  li.appendChild(gzText);  //将li添加到city下  //获取city节点  var city = document.getElementById("city");  city.appendChild(li);  })    //封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

### 插入到某个节点的前面

|  |
| --- |
| //将"广州节点插入到#bj"前面  myClicek("btn",function(){  //创建一个广州  var li = document.createElement("li");  var gzText = document.createTextNode("广州");  li.appendChild(gzText);    //获取id为bj的节点  var bj = document.getElementById("bi");  //获取city  var city = document.getElementById("city");    //插入:  city.insertBefore(li,bj);    })  //封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

### 替换节点

|  |
| --- |
| //bj替换为广州  myClicek("btn",function(){  //创建一个广州  var li = document.createElement("li");  var gzText = document.createTextNode("广州");  li.appendChild(gzText);    //获取id为bj的节点  var bj = document.getElementById("bj");  //获取city  var city = document.getElementById("city");    //替换:  city.replaceChild(li,bj);  })  //封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

### 删除节点

|  |
| --- |
| //删除子节点  myClicek("btn",function(){  //获取id为bj的节点  var bj = document.getElementById("bj");  //获取city  var city = document.getElementById("city");    //删除节点:  city.removeChild(bj);  })  //封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

法二：不知道被删元素的父元素

|  |
| --- |
| //删除子节点  myClicek("btn",function(){  //获取id为bj的节点  var bj = document.getElementById("bj");  //通过父节点删除节点:  bj.parentNode.removeChid(bj);  })  //封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

### 通过innerHtml对象html中添加代码

|  |
| --- |
| //通过innerHtml中想html中添加代码，完成dom增添该。  myClick("btn",function(){  //向city中添加广州  var city = document.getElementById("city");  //使用innerHtml向dom中添加广州  city.innerHTML+="<li>广州</li>"    //通过其他方法着增删改操作时，会删除原来的节点，重新创建，动静太大。  //一般会采用两种结合的方式  var li = document.createElement("li");  //向li中设置文本  li.innerHTML = "广州";  city.appendChild(li);  })  //封装了单机响应函数  function myClick(idStr,fun){  //为id为btn的按钮绑定一个单机响应函数  var btn = document.getElementById("idStr");  btn.onclick = fun;  } |

## DOM操作CSS

### 格式

通过JS修改元素的样式：

语法：元素.style.样式名=样式值

注意：

1、如果CSS的样式名中含有-，这种名称在JS中是不合法的比如background-color

需要将这种样式名修改为驼峰命名法，去掉-，然后将-后的字母大写。

2、我们通过sty1e属性设置的样式都是内联样式，而内联样式有较高的优先级，所以通过JS修改的样式往往会立即显示，但是如果在样式中写了！important，则此时样式会有最高的优先级，即使通过JS也不能覆盖该样式，此时将会导致JS修改样式失效，所以尽量不要为样式添加！important。

### 案例获取内联中的样式

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <style type="text/css">  #box1{  width:200px;  height: 200px;  background-color: red;  }  </style>  <script type="text/javascript">  window.onload = function(){  //点击按钮以后，修改box1的大小  //获取box1  var box1 = document.getElementById("box1");  //为我们按钮绑定单机响应函数。  var btn01 = document.getElementById("btn01");  btn01.onclick = function(){  //修改box1的宽度  box1.style.width ="300px";  box1.style.height ="300px";  box1.style.backgroundColor ="yellow";  };    //注意这里读的是内联样式，如果标签中没有样式，会读到空值。  var box1 = document.getElementById("box1");  //为我们按钮绑定单机响应函数。  var btn02 = document.getElementById("btn02");  btn02.onclick = function(){  //修改box1的样式  console.log(box1.style.width);  };    };  </script>  </head>  <body>  <button id="btn01">修改样式</button>  <button id="btn02">读取样式</button>  <div id="box1">    </div>  </body>  </html> |

### 读取和设置显示中的样式

获取元素的当前显示的样式，它可以用来读取当前元素正在显示的样式。果当前元素没有设置该样式，则获取它的默认值。

法一：currentStyle只支持IE浏览器使用，其他浏览器都不支持。

语法：元素.currentStyle.样式名

此方法如果当前没有设置样式时，会返回默认值。

例如：如果没有设置width，会返回auto。

法二：在其他浏览器中可以使用getComputedStyle()这个方法来获取元素当前的样式。这个方法是window的方法，可以直接使用需要两个参数。但是该方法不支持ie8；

注：

第一个：要获取样式的元素

第二个：可以传递一个伪元素，一般都传nu11

该方法会返回一个对象，对象中封装了当前元素对应的样式可以通过对象。样式名来读取样式。如果获取的样式没有设置，则会获取到真实的值，而不是默认值

比如：没有设置width，它不会获取到auto，而是一个长度

注：通过currentstyle和getcomputedStyle（)读取到的样式都是只读的，不能修改，如果要修改必须通过style属性。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <style type="text/css">  #box1{  width: 100px;  height: 100px;  background-color: yellow;  }  </style>  <script type="text/javascript">  window.onload = function(){  var box1 = document.getElementById("box1");  var btn01 = document.getElementById("btn01");  btn01.onclick = function(){  //读取box的宽度  //正常浏览器方式  /\*alert(getComputedStyle(box1,null).width);  //ie8的方式  alert(box1.currentStyle.width);\*/    };  /\*自己定义一个所有浏览器都支持的方法  参数：  obj 要获取的元素  name要获取的样式名  \*/    function getStyle(obj,name){  //处理兼容性问题  //这里判断的原则是判断这个方法对象在浏览器是否存在，  //以此来判断是什么浏览器，这里不能写.getComputedStyle对象，  //如果浏览器没找到该对象，会到全局域中去找，找不到会报错，使用属性的  //方式可以避免报错，而返回undefined  if(window.getComputedStyle){  //正常浏览器方式  return getComputedStyle(obj,null)[name];  }else{  //ie8的方式  return obj.currentStyle[name];  }    }    }  </script>  </head>  <body>  <button id = "btn01">点我一下</button>  <br/>  <div id="box1">    </div>  </body>  </html> |

### 多个属性

clientwidth和lientHeight：两个属性可以获取元素的可见宽度和高度

1、这些属性都是不带px的，返回都是一个数字，可以直接进行计算。

2、会获取元素宽度和高度，包括内容区和内边距。

3、这些属性都是只读的，不能修改。数据都是相加起来的，无法修改该属性。

offsetwidth和offsetHeight获取元素的整个的宽度和高度，包括内容区、内边距和边框

offsetParent可以用来获取当前元素的定位父元素。会获取到离当前元素最近的开启了定位的祖先元素，如果所有的祖先元素都没有开启定位，则返回body

offsetLeft：当前元素相对于其定位父元素的水平偏移量

offsetTop：当前元素相对于其定位父元素的垂直偏移量

scrollWidth和scrollHeight：可以获取元素整个滚动区域的高度。当内部的div大于外外部div时，会溢出，设置外部的div，overflow:hidden。两个方法属性显示的宽高为里面滚动的宽高。

scrollLeft：可以获取水平滚动条滚动的距离。

scrollTop：可以获取垂直滚动条滚动的距离

当满足scrollHeight-scrollTop==clientHeight：说明垂直滚动条滚动到底了。

当满足scrollWidth-scrollLeft==clientwidth说明水平滚动条滚动到底。

例如：注册东西时，必须读了写协议才能确定。滚动条到底。

|  |
| --- |
| window.onload = function() {  var box1 = document.getElementById("box1");  var btn1 = document.getElementById("btn1");  btn1.onclick = function() {  //alert(box1.clientHeight);//返回100  /\* alert(box1.clientWidth);  alert(box1.clientHeight);  //不能修改属性，只读  box1.clientHeight = 300;  //该方法包括了边框  alert(box1.offsetHeight); //120  //该方法包括了边框  alert(box1.offsetWidth); //120  //获取box1的定位父元素  var box = box1.offsetParent;  alert(box.id);//box  //获取开启了定位元素的偏移量\*/    }  } |

## 阅读协议案例

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">    <title></title>  <style type="text/css">  #info{  width: 200px;  height: 200px;  background-color: red;  overflow: auto;  }  </style>  <script type="text/javascript">  window.onload = function(){    //获取id为info的p元素  var info = document.getElementById("info");  //获取两个input  var inputs = document.getElementsByTagName("input");  //为info绑定一个滚动条滚动时间  info.onscroll = function(){  //检查滚动条是否滚动到底  if(info.scrollHeight-info.scrollTop==info.clientHeight){  //滚动条滚动到底，是表单项可用  //disabled属性可以设置一个元素是否禁用。  //如果设置为true，可用，设置为false，禁用。  inputs[0].disabled =false;  inputs[1].disabled= false;  }  }    }  </script>  </head>  <body>  <h3>欢迎用户注册</h3>  <p id="info">  亲爱的用户,请仔细阅读协议。  </p>  <input type="checkbox" disabled="disabled"/>我已仔细阅读  <!--如果为表单项添加 disabled="disabled"则表达项变成不可用的状态-->  <input type="submit" value="注册" disabled="disabled"/>  </body>  </html> |

# JS的事件机制

## 事件

事件，就是用户和浏览器之间的交互行为，比如：点击按钮，鼠标移动、关闭窗口等。

解释：一般情况下当我们的行为动作满足了一定的条件后，会触发某类事务的执行。

作用：我们可以在事件对应的属性中设置一些js代码，这样当事件被触发时，这些代码将会执行主要是结合js的函数来使用，通过事件机制调用js函数。可以为按钮的对应事件绑定处理函数的形式来响应事件。这样当事件被触发时，其对应的函数将会被调用。

例如：var btn=document.getElementById("btn")；获取button对象。

btn.onclick=function(){//单机绑定函数。

alert(“你还点~~~”）；

};

### 单双击事件

只要是单击或者可以触发js中的函数。

单击：onclick 当鼠标单击的时候会触发

<input type="button" name="" id="" value="测试单机" onclick="testOnclick()" />

双击：ondblclick 当鼠标双击的时候会被触发

<input type="button" name="" id="" value="测试单机" ondblclick="testOnclick()" />

### 鼠标事件

onmouseover 当鼠标悬停在某个HTML元素上的时候触发

<div id="showdiv" onmouseover="textOnmouse()">鼠标悬停事件</div>

onmousemove 当鼠标在某个HTML元素上移动的时候触发，游戏中鼠标的移动。

<div id="showdiv" onmousemove="textOnmouse()">鼠标悬停事件</div>

onmouseout 当鼠标在某个HTML元素上移出的时候触发

<div id="showdiv" onmouseout="textOnmouse()">鼠标悬停事件</div>

### 键盘事件

onkeyup 当键盘在某个HTML元素上弹起的时候触发，游戏中技能操作，wasd。

<input type="text" name="" id="" value="" onkeyup="testOnkeyup();"/>

onkeydown 当键盘在某个HTML元素上下压的时候触发,

<input type="text" name="" id="" value="" onkeydown="testOnkeydown()"/>

### 焦点事件

onfocus 当某个HTML元素获取焦点的时候触发

<input type="text" name="" id="" value="" onfocus="testOnfocus();"/>

onblur 当某个HTML元素失去焦点的时候触发

<input type="text" name="" id="" value="" onblur="testOnblur();"/>

### 页面加载事件

作用：

onload

当页面加载成功后触发。

### 注意:

js中添加事件的第一种方式：

在HTML上直接使用事件属性进行添加，属性值为所监听执行的函数。

js中添加事件的第一种方式：document 对象来执行

## 事件对象

事件对象：当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数，

在事件对象中封装了当前事件相关的一切信息，比如：鼠标的坐标键盘哪个按键被按下鼠标滚轮滚动的方向。

在IE8中，响应函数被乘触发时，浏览器不会传递事件对象，在IE8及以下的浏览器中，是将事件对象作为window对象的属性保存的。

事件对象：event

对象属性：clientX：事件属性返回当事件被触发时鼠标指针向对于浏览器页面（或客户区）的水平坐标。

clientY：事件属性返回当事件被触发时鼠标指针向对于浏览器页面（客户区）的垂直坐标。

### onmouseover (获取鼠标的坐标)

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <style type="text/css">  #areaDiv{  width: 200px;  height: 200px;  border: black 1px solid;  }  #showMsg{  width: 200px;  height: 200px;  border: black 1px solid;    }  </style>  <script type="text/javascript">    window.onload = function(){  //当鼠标在areadiv中一定是，在showMsg中显示鼠标的坐标  //获取两个div  var areaDiv = document.getElementById("areaDiv");  var showMsg = document.getElementById("showMsg");  //onmousemove  areaDiv.onmousemove = function(enevt){//接收浏览器传进来的时间对象，关于时间对象的信息。  //在showMsg来线束鼠标的坐标  var x = event.clientX;  var y = event.clientY;  alert("x = "+x+","+"y = "+y);  showMsg.innerHTML = "x = "+x+","+"y = "+y;  }  }  </script>  </head>  <body>  <div id="areaDiv">    </div>  <div id="showMsg">    </div>  </body>  </html> |

### 案例：div跟随鼠标移动

### 冒泡

事件的冒泡（Bubble)：所谓的冒泡指的就是事件的向上传导，当后代元素上的事件被触发时，其祖先元素的相同事件也会被触发。

在开发中大部分情况冒泡都是有用的，如果不希望发生事件冒泡可以通过事件对象来取消冒泡。

例子

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <style type="text/css">  #box1{  width:200px;  height:200px;  background-color: yellowgreen;  }  #s1{  background-color: yellow;  }  </style>  <script type="text/javascript">  //当我点击s1时，其他的事件也会响应  window.onload = function(){  //为s1绑定一个单机响应函数  var s1 = document.getElementById("s1");  s1.onclick = function(event){  alert("我是s1");  //取消冒泡  //可以将时间对象的cancelBubble设置为true  event.cancelBubble = true;  }    //为box1绑定一个单机响应函数  var box1 = document.getElementById("box1");  box1.onclick = function(){  alert("我是box1");  }    //为body绑定一个单机响应函数  document.body.onclick = function(){  alert("我是body");  };  }  </script>  </head>  <body>  <div id="box1">  我是box1  <span id="s1" >  我是span  </span>    </div>  </body>  </html> |

### 事件的委派

事件的委派：指将事件统一绑定给元素的共同的祖先元素，这样当后代元素上的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件。

事件委派是利用了冒泡，通过委派可以减少事件绑定的次数，提高程序的性能

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //点击按钮后添加超链接  var btn01 = document.getElementById("btn01");  var ul = document.getElementById("u1");  btn01.onclick = function(){  var li = document.createElement("li");  li.innerHTML = "<a href='javascript:;'>新建的超链接</a>";  //将例添加到ul中  ul.appendChild(li);    }  /\*为每一个超链接都绑定一个单击响应函数  \*这里我们为每一个超链接都绑定了一个单击响应函数，这种操作比较麻烦，  而且这些操作只能为已有的超链接设置事件，而新添加的超链接必须重新绑定\*/  //为每个链接绑定一个单机响应函数  var allA = document.getElementsByTagName("a");  /\*//遍历  for(var i= 0;i<allA.length;i++){  alert("我是a的单机响应函数");  };\*/    //我们希望，只绑定一次事件，即可应用到多个的元素上，即使元素是后添加的  //我们可以尝试将其绑定给元素的共同的祖先元素|  //为ul绑定一个单机响应函数  u1.onclick = function(event){    //target:event中的target表示触发事件的对象  //alert(event.target);  //如果触发的对象使我们希望的元素，则执行。  if(event.target.className =="link"){  alert("我是u1的单击响应函数");  }  };  }  </script>  </head>  <body>  <button id="btn01">按钮</button>  <ul id="u1">  <li><a href="javascript:;" class="link">超链接一</a></li>  <li><a href="javascript:;" class="link">超链接二</a></li>  <li><a href="javascript:;" class="link">超链接三</a></li>    </ul>  </body>  </html> |

### 事件的绑定

addEventListener();通过这个方法也可以为元素绑定响应函数。

-参数：

1.事件的字符串，不要on。

2.回调函数，当事件触发时该函数会被调用。

3.是否在捕获阶段触发事件，需要一个布尔值，一般都传false。

4.不支持ie8以下的浏览器。

使用addEventListener()可以同时为一个元紊的相同事件同时绑定多个响应函数，这样当事件被触发时，响应函数将会按照函数的绑定顺序执行。

attachEvent()

-在IE8中可以使用attachEvent（)来绑定事件

-参数：

1.事件的字符串，要on。

2.回调函数。

不同的是它是后绑定先执行，执行顺序和addEventListener()相反。

例子：

|  |
| --- |
|  |

### 事件的传播

事件的传播

-关于事件的传播网景公司和微软公司有不同的理解

-微软公司认为事件应该是由内向外传播，也就是当事件触发时，应该先触发当前元素上的事件，

然后再向当前元素的祖先元素上传播，也就说事件应该在冒泡阶段执行。

-网景公司认为事件应该是由外向内传播的，也就是当前事件触发时，应该先触发当前元素的最外层的祖先元素的事件，

然后在向内传播给后代元素

-W3C综合了两个公司的方案，将事件传播分成了三个阶段

1.捕获阶段

-在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件

2.目标阶段

-事件捕获到目标元素，捕获结束开始在目标元素上触发事件

3.冒泡阶段

-事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件

如果希望在捕获阶段就触发事件，可以将addEventListener（)的第三个参数设置为true一般情况下我们不会希望在捕获阶段触发事件，所以这个参数一般都是false

### 事件的拖拽

### 事件的滚动

onmousewhee1鼠标滚轮滚动的事件，会在滚轮滚动时触发，但是火狐不支持该属性。在火狐中需要使用DOMMouseScrol1来绑定滚动事件，注意该事件需要通过addEventListene()函数来绑定。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <style type="text/css">  #box1{  width: 200px;  height: 300px;  background-color: red;  }    </style>  <script type="text/javascript">  window.onload = function(){  //当鼠标滚轮向下滚动时，bo×1变长  //当滚轮向上滚动时，bo×1变短    //获取box1  var box1 = document.getElementById("box1");  //为box1绑定滚轮事件  box1.onmousewheel = function(event){  //判断鼠标滚动的方向  event = event||window.event;  //event.wheelDelta;可以获取鼠标滚轮滚动的方向  //向上滚120，向下滚-120 我们只看大小，只看  //wheelDelta这个属性火狐中不支持  //在火狐中使用event.detail来获取滚动的方向  //向上滚-3向下滚3  if(event.wheelDelta>0 || event.detail<0){  //向上滚 box1变短  box1.style.height = box1.clientHeight - 10+"px";  }else{  //向下滚box1边长  box1.style.height = box1.clientHeight + 10+"px";  }    //当滚轮滚动时，如果浏览器有滚动条，滚动条会随之滚动，  //这是浏览器的默认行为，如果不希望发生，则可以取消默认行为  return false;    //使用addEventListener（）方法绑定响应函数，取消默认行为时不能使用return false  //需要使用event来取消默认行为event.preventDefault();  //但是IE8不支持event.preventDefault();这个玩意，如果直接调用会报错    };  };  </script>  </head>  <body>  <div id="box1">    </div>    </div>  </body>  </html> |

### 键盘移动div(onkeydown)

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script type="text/javascript">  window.onload = function(){  //使div可以根据不同的方向键向不同的方向移动  //按左键，div向左移  //按右键，div向右移 //为document绑定一个按键按下的事件    //开启一个定时器，来控制div的移动,通过修改dir改变方向  var dir = 0;  setInterval(function(){  /\*  \* 37左  \* 38上  \* 39右  \* 40下  \* \*/  //定义一个变量，每次移动的距离  var speed = 20;  switch(dir){  case 37://alert("向左")；left值减小  box1.style.left = box1.offsetLeft-speed+"px";break;  case 38:  box1.style.top = box1.offsetTop-speed+"px";break;  case 39:  box1.style.left = box1.offsetLeft+speed+"px";break;  case 40:  box1.style.top = box1.offsetTop+speed+"px";break;  };    },200);  var box1 = document.getElementById("box1");  document.onkeydown = function(event){  event = event||window.event;  dir = event.keyCode;    };    //当按键从松开，div不在移动  document.onkeyup = function(){  dir=0;  };  }  </script>  <style type="text/css">  #box1{  width: 100px;  height: 100px;  background-color: red;  position: absolute;  }  </style>  </head>  <body>  <div id="box1">    </div>  </body>  </html> |

# BOM

# 知识店

## 提交表单

|  |
| --- |
| **function** changePage(pageNum){  $("#currentPage").val(pageNum);  //通过选择器选择到id为searchForm表单,通过js的方式提交表单  $('#searchForm').submit();  } |

## javascript:void(0)或者javascript:;

### 为什么要使用href=”javascript:void(0);”?

href=”javascript:void(0);”这个的含义是，让超链接去执行一个js函数，而不是去跳转到一个地址，而void(0)表示一个空的方法，也就是不执行js函数。

javascript:是伪协议，表示url的内容通过javascript执行。void(0)表示不作任何操作，这样会防止链接跳转到其他页面。这么做往往是为了保留链接的样式，但不让链接执行实际操作。

<a href="javascript：void(0)" onClick="window.open()"> 点击链接后，页面不动，只打开链接

<a href="#" onclick="javascript:return false;"> 作用一样，但不同浏览器会有差异。

### href=”javascript:void(0);”与href=”#"的区别

<a href="javascript:void(0)">点击</a> 点击链接后不会回到网页顶部

<a href="#">点击</a> 点击后会回到网面顶部

"＃"其实是包含了位置信息，例如默认的锚点是＃top 也就是网页的上端,而javascript:void(0) 仅仅表示一个死链接这就是为什么有的时候页面很长浏览链接明明是＃可是跳动到了页首而javascript:void(0) 则不是如此所以调用脚本的时候最好用void(0)。

### javascript的方法

<a href="#" onclick="javascript:方法;return false;">文字</a>

<a href="javascript:void(0)" onclick="javascript:方法;return false;">文字</a>

例如：<a href="javascript:hanshu();"这样点击a标签就可以执行hanshu()函数了。

### 主要作用

在网页编程中，一般让一个超链接点击后不链接到任何地方，而鼠标移上去仍然显示手指形状的图标，就用javascript:void(0)，然后真实执行的操作，是在这个a标签后面加onclick="xxxxx"，就是鼠标单击后执行某个Javascript函数进行具体的操作，这样可以做更多的事情，比如根据某个值进行判断跳转到不同的页面等等。