# javascript基础

## 一、简介

**JavaScript**

**BOM**

**DOM**

**ECMAScript**

#### ECMAScript是核心 是基础

* + 1. 语法
    2. 类型
    3. 语句
    4. 关键字
    5. 保留字
    6. 操作符
    7. 对象

JavaScript实现了ECMAScript

#### DOM 文档对象模型

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节点结构，可以做到无需重新加载页面就可以做到修改页面内容，可以做到轻松自如的添加，删除，修改，替换任何节点

由ECMAScript实现，很多语言都可以，但ECMAScript已经成为主流

#### 1.3、 BOM浏览器对象模型

访问和操作浏览器窗口的浏览器对象模型

## 二、 在HTML中使用

#### 2.1、 嵌入代码

<script type=”text/javascript”>

alert(“Hello World”);

</script>

#### 2.2、 外部文件

<script type=”text/javascript” src=”example.js”></script>

加载：由脚本的位置决定

统一放在head标签中是为了，方便管理

放到body结尾前，则可以先加载整个文档，在下载js文件

延迟脚本：只适用于外部脚本

defer=“defer”

这个属性表示：脚本会延迟到整个页面解析完毕后执行。但不会影响其下载顺序。

<script type=”text/javascript”src=”example.js” defer=”defer”></script>

优势：可维护性强，可缓存，适应未来

## ECMAScript

### 三、 基本概念

#### 3.1 语法

* 区分大小写
* 标识符
* 注释
* 严格模式
* 语句

#### 3.2 关键字

#### 3.3 变量

#### 3.4 数据类型

java的数据类型

* 简单数值类型：用typeof 检测 ，Undefined,Number,String,Boolean。 他们创建的不是对象，时数值字面量。
* 引用数据类型：Object,Null,Function。用instanceof检测。他们创建的都是对象
* Number
* Boolean
* String
* Object
* Undefined
* Null

type of 返回

* Number
* Boolean
* String
* Object 对象或Null
* Undefined 如果变量未定义
* function 如果变量是函数

#### 3.5 操作符

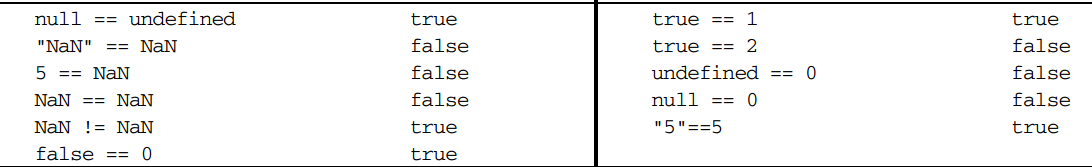
##### 3.5.7 相等操作符

* 相等和不想等 == ！=

先转换再比较

字符串与数字比较会转换成数字

对象与字符串比较，会调用对象的valueof方法



* 全等和不全等 === ！==

不转换，直接比较。

var value = "5";  
var value1 = 5;  
console.log(value == value1);  
console.log(value === value1);

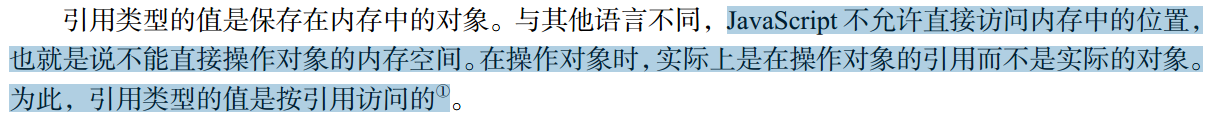
#### 3.6 语句

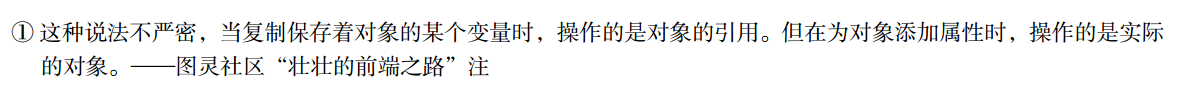
##### 3.6.8 with语句

#### 3.7 函数

### 四、 变量、作用域和内存问题？作用域链

JS变量：只是特点时间保存特定值的一个名字而已，由于不存在定义某个变量必须保存某种数据类型的值的规则，变量的值及其数据类型可以在脚本的生命周期内动态改变，强大，但是容易出问题。





？没看懂

#### 4.1 变量：

1. 基本数据类型：指的是简单的数据段 number、boolean、string、null、undefined
2. 引用数据类型：指那些可能由多个值构成的对象

##### 4.1.1 动态属性

* 引用类型可以动态的为对象添加、删除、修改属性，
* 基本数据类型不可以为其动态添加、删除属性，但是其数据类型可以在脚本的生命周期中动态改变，一个变量可以一会是基本类型一会是引用类型，自由度非常高，所以控制起来要非常严谨，没有编译，不会较错。

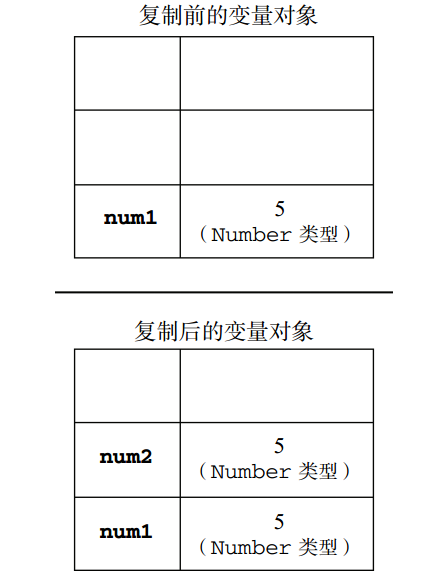
##### 4.1.2 复制变量的值

* 基本数据类型：用一个对象为另外一个对象初始化的时候，其实是创建了另外一个副本，因为没有什么缓存池的设计

var defined = 5;

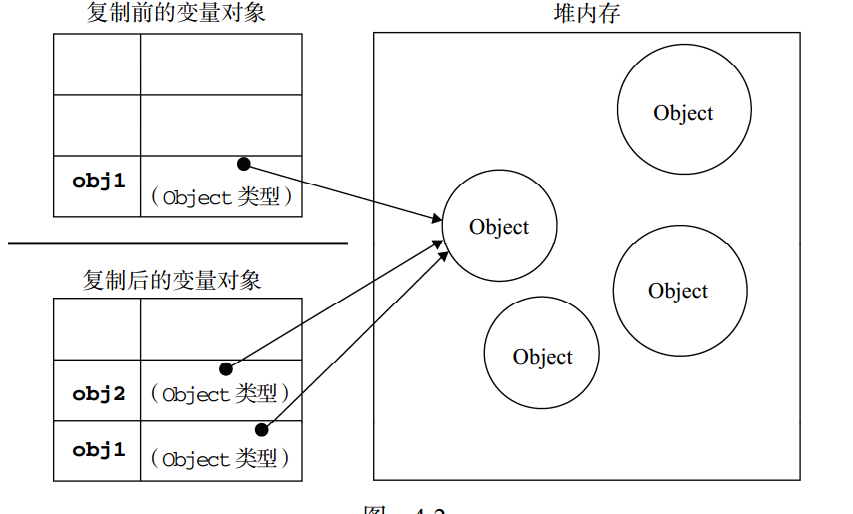
var defined1 = defined;

在堆中其实，有两个5的对象，一个由defined引用，另外一个有defined1引用。



* 引用数据类型：

同样也是复制一份，只不过这个值副本保存的是指针，指针指向的还是原来的那个对象



内存模型 栈里面是键值对?

* 参数传递

都是值传递，

基本类型，传入函数内的参数对原参数是没有影响的。

引用类型的参数，由于只是复制了一个引用的副本，副本中的指针指向的还是原来的对象，所以函数内引用类型的属性改变，函数外部的引用类型的属性也会随之改变。

* 检测类型

基本类型的值用typeof检查

引用类型的值用instanceof 检查

#### 4.2 执行环境

全局执行环境 声明的是全局变量和方法，全局变量和方法属于window对象，该环境中的对象和属性，当全部代码执行完毕，环境被销毁后保存在其中的变量和属性也随之销毁。

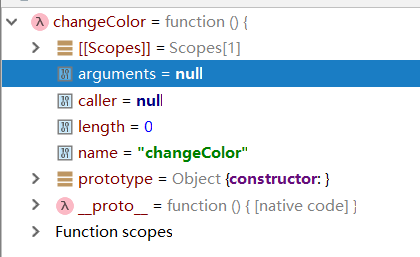
每个函数都有自己的执行环境。当执行流进入一个函数时，函数的环境就会被推入一个环境栈中。  
而在函数执行之后，栈将其环境弹出

每个函数都有自己的执行环境

js中的函数是一个非常独特的概念，函数即对象

js实际上就是由一个一个对象组成的，

标识符的解析是沿着作用域链一级一级地搜索标识符的过程。搜索过程始终从作用域链的最前端开始，然后逐级地向后回溯，直到找到标识符为止。



var color = "blue";

var changeColor = function () {

if(color === "blue"){

color = "red";

}else{

color = "blue";

}

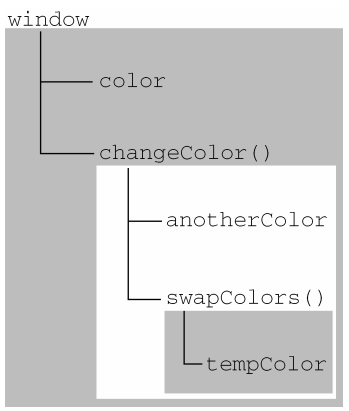
}

console.log("color is now : " + color);

##### 4.2.1 作用域链

定义：当代码在执行时，会创建对象的作用域链。

用途：保证对当前执行环境有权访问的属性和函数 有序的访问，



* 延长作用域

try catch 语句块、with（location）{}

with (location) {  
 var qs = search.substring(1);  
 var hostname = hostname;  
 var url = href;  
}

function buildurl() {  
 var qs = "?debug = true ";  
 var href = "test";  
 with (location){  
 var url = href + qs;  
 }  
 return url;  
}  
console.log(buildurl()); // http://localhost:63342/HTML-CSS-JS/js-base/WebContent/html/work/js-test.html?debug = true

with语句块，可以将location对象的作用域链 添加到with语句中，search、hostname、href等都是location作用域链中的变量，而不在全局作用域中。是将location的作用域链添加到buildurl的前端。gs在location中找不到，所以往buildurl中的作用域链中查找。

* 没有块级作用域

if(true){

var color = “blue”;

}

alert(color); //blue

语句块声明的变量会被添加到当前执行环境。

for语句块也一样。

for(var i;i<10;i++){

doSomething(i);

}

alert(i) ---这里的i同样被添加到for的执行环境中

1. 声明变量

使用var声明的变量会自动添加到作用域链最接近的环境，with，for，if由于没有块，所以其最接近的环境就是 语句块所在的环境。

如果初始化没有用var声明，则会被自动添加到全局环境中。

### 五、 引用类型，内置对象

* JS是数据类型，不是类，Object、Array、Date、RegExp是内置的几种数据类型，从内置或数据类型的层面上讲，有点像java的八种基本数据类型。
* 函数/方法是一种数据类型，可以有自己的属性和方法
* JS 的数据类型都可以动态的添加属性和方法

#### 5.1 Object类型

创建实例

1. new 的方法

var obj = new Object();

obj.name = “zhangsan”;

obj.age = 18;

1. 引用 对象字面量 的方法

var obj = {

name : zhangsan,

age : 18

};

#### 5.2 Array类型

**大小是可以动态调整的**

每一项可以保存任意类型的数据，第一项：number，第二项：string，第三项：对象

##### 创建

* var array = new Array(3);
* var array = Array(3); 或 Array(“aa”);
* var array = [“aa”,”bb”,”cc”];

##### 取值：

array[i],

长度 array.length

赋值 array(array.length) = “aa”;

##### 检测

array instanceof Array 返回 Boolean， 跨框架，传参数组参数时，有两个Array对象，跨框架至今不能相互校验

需要采用 Array.isArray(array)

##### 栈方法 ： 后进先出

ECMAScript的数组，可以表现的像栈一样

var arr = new Array();

arr.push("aa","bb");

console.log(arr);

arr.push("cc");

console.log(arr);

var value = arr.**pop**();

console.log(value);

console.log(arr);

##### 队列方法： 先进先出

var arr = new Array();

arr.push("aa","bb");

console.log(arr);

arr.push("cc");

console.log(arr);

var value = arr.**shift**();

console.log(value);

console.log(arr);

* 后端进前端出

var arr = new Array();

arr.unshift("aa","bb");

console.log(arr);

arr.unshift("cc");

console.log(arr);

var value = arr.pop();

console.log(value);

console.log(arr);

* push：从前面推进来
* unshift：从末端推进来
* pop：从前面取出来
* shift：从末端取出来

##### 排序方法

sort() ： 正序

reverse() ：倒序

sort(compare) ： 自定义排序

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

arr.sort();

大于10的倒序，小于10的顺序

var compare = function(value1,value2){

if(value1>value2){

if(value2>10){

return -1;

}else{

return 1;

}

}else if(value1<value2){

if(value1<10){

return -1;

}else{

return 1;

}

}else{

return 0;

}

}

var array = [12,22,66,22,1,4,8,4,9,33,256,55];

array.sort(compare);

#### 5.3 Date类型

##### 创建

var date = new Date(毫秒数); -- 得到的是本地的时间

var date = new Date(Date.UTC(2005,2,6,15,28,10)); -- 传入的数字是UTC的时间

-- 得到的并不是1971年到2005,2,6,15,28,10 的毫秒数

var lon = Date.UTC(2005,2,6,15,28,10);

console.log(lon);

var da = new Date(2005,2,6,15,28,10);

console.log(da.getTime());

这两个时间戳是不一样的，前面那个传入的是UTC的时间，

var date2 = new Date(Date.parse("Tue May 25 2004 00:00:00 GMT-0700")); --这个date.parse()中传入的也不是本地时间

**var da = new Date(2005,2,6,15,28,10); --传入的才是本地时间**

var da = new Date(); --得到本地时间对象

Date.now(); -- 本地当前时间的时间戳

new Date(Date.now()); -- 好像多此一举

##### 获取时间：

//日期格式化

Date.prototype.Format = function(fmt) {

var o = {

"M+" : this.getMonth() + 1,

"d+" : this.getDate(),

"h+" : this.getHours(),

"m+" : this.getMinutes(),

"s+" : this.getSeconds(),

"q+" : Math.floor((this.getMonth() + 3) / 3),

"S" : this.getMilliseconds()

};

if (/(y+)/.test(fmt)){

fmt = fmt.replace(RegExp.$1, (this.getFullYear() + "").substr(4 - RegExp.$1.length));

}

for (var k in o){

if (new RegExp("(" + k + ")").test(fmt)){

fmt = fmt.replace(RegExp.$1, (RegExp.$1.length == 1) ? (o[k]) : (("00" + o[k]).substr(("" + o[k]).length)));

}

}

return fmt;

}

var time = no.Format("yyyy-MM-dd hh:mm:ss");

console.log(time);

time = no.toUTCString();

console.log(time);

time = no.toLocaleDateString();

console.log(time);

time = no.toLocaleTimeString();

console.log(time);

time = no.toLocaleString();

console.log(time);

得到的都是字符串

var date = new Date();

console.log(date.getTime()); -- 时间戳

console.log(date.getFullYear()); --yyyy

console.log(date.getMonth()); -- 0 为1月

console.log(date.getDate()); --天

console.log(date.getDay()); --星期几

console.log(date.getHours()); -- 小时 24小时制

console.log(date.getMinutes()); -- 分

console.log(date.getSeconds()); -- 秒

console.log(date.getMilliseconds()); -- 毫秒

#### 5.4 RegExp类型

##### 创建：

1. new

var reg = new RegExp(“aa”,”g”);

reg.test(“String”);

1. perl 语法，**注意不要加引号**

var reg = /aa/g ;

##### 正则表达式

* 修饰符 ： i ：忽略大小写

g :全局匹配

m : 执行多行匹配

* 方括号 ：

[abc] : 查找方括号内的任何字符

[[^abc]](http://www.w3school.com.cn/jsref/jsref_regexp_charset_not.asp) ： 查找不在方括号中的字符

[0-9] [a-z] [A-Z] [A-z]

(red|blue|green) ： 查找其中的任意一项

* 元字符

|  |  |
| --- | --- |
| [\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。 |

* 量词

|  |  |
| --- | --- |
| [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。 |

* RegExp对象方法

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [compile](http://www.w3school.com.cn/jsref/jsref_regexp_compile.asp) | 编译正则表达式。 | 1 | 4 |
| [exec](http://www.w3school.com.cn/jsref/jsref_exec_regexp.asp) | 检索字符串中指定的值。返回找到的值，并确定其位置。 | 1 | 4 |
| [test](http://www.w3school.com.cn/jsref/jsref_test_regexp.asp) | 检索字符串中指定的值。返回 true 或 false。 | 1 | 4 |

* 支持正则表达式的String对象的方法

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [search](http://www.w3school.com.cn/jsref/jsref_search.asp) | 检索与正则表达式相匹配的值。 | 1 | 4 |
| [match](http://www.w3school.com.cn/jsref/jsref_match.asp) | 找到一个或多个正则表达式的匹配。 | 1 | 4 |
| [replace](http://www.w3school.com.cn/jsref/jsref_replace.asp) | 替换与正则表达式匹配的子串。 | 1 | 4 |
| [split](http://www.w3school.com.cn/jsref/jsref_split.asp) | 把字符串分割为字符串数组。 | 1 | 4 |

***如何获取一个字符串中的数字字符，并按数组形式输出，如dgfhfgh254bhku289fgdhdy675gfh***

***var str = "dgfhfgh254bhku289fgdhdy675";***

***var pattern = /\d+/g;***

***console.log(str.match(pattern));***

***校验邮政编码***

***var mail = "570228";***

***var mail1 = "2323233232";***

***var mail2 = "23 232332 32";***

***// var pattern = /^\d{6}$/;***

***var pattern = /^[0-9]{6}$/;***

***console.log(pattern.test(mail));***

***console.log(pattern.test(mail1));***

***console.log(pattern.test(mail2));***

***校验用户姓名只能输入1到30个以字母开头的字串，中间后面可以为数字（不能为空格，标点等）***

***var mail = "e70228";***

***var mail1 = "2323233232";***

***var mail2 = "23 232332 32";***

***var pattern = /^\w{1}(\w|\d){0,29}$/g;***

***console.log(pattern.test(mail));***

***console.log(pattern.test(mail1));***

***console.log(pattern.test(mail2));***

***var mail = "e702\_28";  
var mail1 = "e702\_ 28";  
var mail2 = "23f38kjdjkkjdkjakjs23233232";***

***var pattern = /^(\w|\d|\_){6,20}$/g;***

***var pattern1 = /^(\w|\d|\_){6,20}$/g;***

***var pattern2 = /^(\w|\d|\_){6,20}$/g;***

***console.log(pattern.test(mail));***

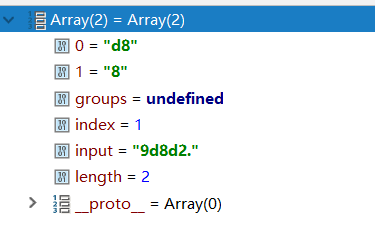
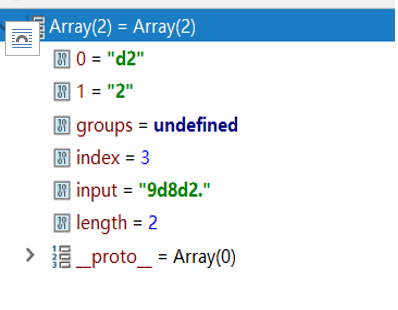
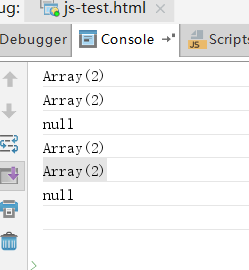
***console***.log(***pattern1***.test(***mail1***));

***console***.log(***pattern2***.test(***mail2***));

注意同一个RegExp对象，带有全局匹配的，匹配一次

没有带全局匹配的可以匹配多次，带全局的，则可以对表达式解读几次，就可以匹配几次，没匹配到就清一次lastIndex一次

***var value= "9d8d2.";  
var pattern = /\w(\d)+/g;  
var match = pattern.exec(value);  
console.log(match);  
match = pattern.exec(value);  
console.log(match);  
match = pattern.exec(value);  
console.log(match);***

只输入5-20个以字母开头、可带数字、“\_”，“.”的字符串

/^([A-Z]|[a-z])(\w|\_|.){0,19}$/ 也可/^([a-Z])(\w|\_|.){0,19}$/

校验E-mail 地址

#### 5.4 函数类型

js的函数实际创建的也是对象，而且与其他引用类型一样拥有属性和方法，所以js中的函数与类一样，其实函数既是类，Object也是类，

函数末尾有分号，没写是因为跨行会自动补充

alert(sum(10,10));  
// var sum = function (num1,num2) {  
// return num1 + num2;  
// }  
function sum(num1,num2) {  
 return num1 + num2;  
}

代码开始执行之前，解析器就已经通过一个名为函数声明提升  
（function declaration hoisting）的过程，读取并将函数声明添加到执行环境中

JavaScript引擎会将函数的声明提升到顶部

将函数的声明，改为函数表达式，则不会提前

### 六、 面向对象的程序设计

EMCAScript没有类的概念，把对象定义为无序属性的集合，属性可以是基本值、对象（包括函数）

可以把EMCAScript的对象想象成散列表：无非就是一组无序的名值对，值可以是数据和对象。

### 七、 函数表达式

#### 7.1 递归

利用arguments.callee()（指向函数正在执行的函数的指针）

var anthorFactorial = factorial;  
factorial = null;  
console.log(anthorFactorial(2));  
console.log("success");  
// function factorial(num) {  
// // if(num > 10){  
// // return 2;  
// // }else {  
// // return num \* factorial(num);  
// // }  
// // }  
function factorial(num) {  
 if(num > 10){  
 return 2;  
 }else {  
 return num \* arguments.callee(num + 1);  
 }  
}

### 语法糖

在相同功能下，语法糖的写法会让代码更加简洁流畅，代码更加语义自然，其实真正解析出来，功能一样，只是写法简单一点

### 函数与对象

java的数据类型

* 简单数值类型：用typeof 检测 ，Undefined,Number,String,Boolean。 他们创建的不是对象，时数值字面量。
* 引用数据类型：Object,Null,Function，。用instanceof检测。他们创建的都是对象

对象：若干属性的集合

除了字面量，其他的一切都是对象

数据类型与对象有两种关系形式

* 一种像 Array，Date，他们是Object的一个子集。
* 另一种 函数与对象，函数本身是一种对象， var func = function(){}，func是对象，是对象就可以随意新增属性。
  1. 每一个函数都是Function类型的实例，即对象，所以 func instanceof Function 为true ，func是对象，是实例。与类不一样。
  2. 每一个对象都是引用类型。引用类型一定要有变量引用，否则就没法使用，会被回收。

所以 函数名func 实际是指向函数实际对象的指针。同时又具备对象的特性，可以含有属性和方法。

func是对象，指向实际函数实体。

function sum(){}

var sum = function(){}; //注意这个要加;因为他与变量赋值写法一致，都需要加;

var sum = new Function(“num1”, “num2”, “return num1 + num2”);//不推荐，但对于理解函数是Funcion类型的实例友好。

有利于理解 new Function()创建的是对象，即函数是对象。函数名称sum 是指针。

* Function不是函数，而是函数类型。用于创建函数实例。-----这才是真正的函数(对象)。函数是一种对象
* 在Javascript中每个函数都有一个Portotype属性。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 ,指向 prototype原型对象，
* 当用函数去创建对象的时候，创建的对象会自动添加\_\_proto\_\_属性，该属性指向原型对象。
* 而且访问函数创建对象的属性和方法时，首先会在自身中找，要是每找到就到原型中找。
  1. 注意这里已经又两个对象，一个sum，时Function的实例，另一个ob 是由函数创建的,是Object的实例。

var dom = function(){  
 var name = "zhangsan";  
}  
var ob = new dom();  
console.log(dom instanceof Function);  
console.log(dom);  
console.log(ob instanceof Function); //false  
console.log(ob instanceof Object); //true

### 闭包

* JavaScript运行使用内部函数
* 内部函数可以访问它们所在的的外部函数中生命的所有局部变量、参数以及声明的其他内部变量，注意是在外部函数中声明的。

当其中的这样一个内部函数在包含它们的外部函数之外被调用时，就会形成闭包

像用return 返回一个内部函数，这样外部就可以使用，或者将内部函数赋给全局对象，外部也可以使用

### 方法调用

function persion(a,b){  
 var name = "zhangsan";  
 var age = a+b;  
 console.log(age);  
 return age;  
}  
var p1 = new persion(1,2);  
var p2 = persion(4,5);  
  
console.log(p1);  
console.log(p2);

p1是对象 Object，是函数创建的实例

p2 不是，只是函数执行的返回值，persion中的代码依然会跑。而且都会将persion对象跑一遍，但p2 赋的值是函数执行的返回值，如果没有返回值，则为undefined。

## BOM

### 八、 BOM

#### 8.1 window对象

##### 8.1.1 全局作用域

可以用直接使用、window、this 访问

* 直接定义的全局变量不能通过delete操作符删除，但是定义在window对象上的属性可以

var age1 = 29;  
window.name = "zhangsan";  
console.log(window.age1);  
console.log(window.name);  
var de = delete window.age1;  
var de1 = delete window.name;  
console.log(de); //false

console.log(de1); //true

* 尝试访问未声明的变量会报错，但是通过查询window对象的属性，可以知道某个变量是否存在。

// var newValue = oldValue; //会报错，而中断程序，  
var newValue = window.oldValue; //只是undefined  
console.log("newValue : " + newValue);

##### 8.1.2 窗口关系和框架

如果页面包含框架，则每个框架都有自己的window对象，并且保存在frames集合中

可以通过集合来访问window.frame[0]

* window:指向的都是当前框架
* top：始终指向最外层或最高层框架，也就是浏览器窗口。
* parent：始终指向当前框架的上一层框架，parent有可能等于top，没有框架的时候，parent一定等于top。
* self：一定指向自己。

#### 8.2 location对象

作用：提供了当前窗口中加载的文档有关信息。

## DOM

# BOM 和DOM

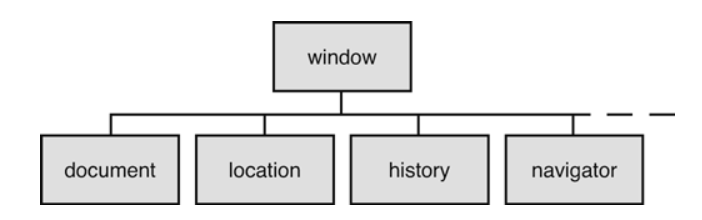
## BOM和DOM

### BOM

浏览器对象模型，获取或设置浏览器的属性、行为。  
  bom的操作对象是 浏览器。  
  例如：新建窗口、获取屏幕分辨率、浏览器版本号等。

**bom中的对象 :**

1. Window对象
2. Document对象
3. Location对象
4. History对象
5. Navigator对象
6. Screen对象



window对象模型

bom中的对象，都是浏览器内置，自动创建的，不需要程序员手动创建。

1. alert() 对象
2. confirm() 对象
3. prompt() : 是打开模态对话框的另一种方式，它允许用户输入信息  
    var name = prompt(“what is your name?”,”John”);  
    var name = prompt(“what is your name?”);
4. history.go(2) //前进2个页面，go(-3)//回退3个页面  
    history.go(“example.com”); //到达历史记录列表里面第一个包含“example.com”的URL
5. location

利用location对象导航

location.href= ‘[www.baidu.com](http://www.baidu.com/)’;使用这种方法把用户转移到新页面时，原始页面还保留在浏览器的历史记录里面。  
location.replace(‘[www.baidu.com](http://www.baidu.com/)’); 直接用新的URL代替当前页面，即把当前页面从历史记录列表里删除。

### DOM

文档对象模型，用来获取或设置文档中标签的属性  
dom 的操作对象是 文档，

### BOM之节点

在 HTML DOM （文档对象模型）中，每个部分都是节点：  
 节点(node) : DOM的最小组成单位  
 节点至少拥有nodeType、nodeName和nodeValue这三个基本属性

1. 文档本身是文档节点
2. 所有 HTML 元素是元素节点
3. 所有 HTML 属性是属性节点
4. HTML 元素内的文本是文本节点
5. 注释是注释节点

| **节点** | **名称** |
| --- | --- |
| Node.ELEMENT\_NODE(1)\* | 元素节点 |
| Node.ATTRIBUTE\_NODE(2) \* | 属性节点 |
| Node.TEXT\_NODE(3) \* | 文本节点 |
| Node.CDATA\_SECTION\_NODE(4) | CDATA节点 |
| Node.ENTRY\_REFERENCE\_NODE(5) | 实体引用名称节点 |
| Node.ENTITY\_NODE(6) | 实体名称节点 |
| Node.PROCESSING\_INSTRUCTION\_NODE(7) | 处理指令节点 |
| Node.COMMENT\_NODE(8) | 注释节点 |
| Node.DOCUMENT\_NODE(9) \* | 文档节点 |
| Node.DOCUMENT\_TYPE\_NODE(10) | 文档类型节点 |
| Node.DOCUMENT\_FRAGMENT\_NODE(11) | 文档片段节点 |
| Node.NOTATION\_NODE(12) | DTD声明节点 |

像 Document（9），Element（1），Attr（2），Text（3） 等类型节点都 实现了Node接口  
内置了文档节点，即document 对象。  
**document 是浏览器对象模型中内置的对象，直接使用就可以，  
文档节点由 元素节点组成，所以是document.getElementBy...来获取下一个节点。  
元素节点中 包含了文本节点和属性节点。**

## prototype

### 原型prototype（js-base项目中)

创建一个构造函数  
问题，Person 构造函数中，为每一个对象添加了一个 sayName的方法。  
  也就是每创建一个对象，都会创建一个sayName的方法。  
  per的sayName 与per1的sayName 不是同一个对象。  
  但是把它移出构造函数，又容易与其他方法混淆，此方法只属于Person对象。  
  所以引入 原型。

function Person(name,age,gender) {

this.name = name;

this.age = age;

this.gender = gender;

this.sayName = function () {

console.log("Hello 大家好，我是：" + this.name);

}

}

var per = new Person("孙悟空",28,"男");

var per1 = new Person("猪八戒",28,"男");

console.log(per);

console.log(per1);

console.log(per.sayName == per1.sayName);

原型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 ,指向 prototype原型对象  
这个属性对应这一个对象，这个对象就是我们所谓的原型对象  
如果函数作为普通函数调用，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，  
执行构造函数的原型对象，我我们通过proto来访问该属性。

就是我们创建每一个函数的时候，都会为其添加一个prototype 属性，  
然后创建函数对象的时候，对象会有一个proto的属性，与 函数的prototype属性 指向相同的 原型对象。

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。  
我们可以将对象中具有共有的内容，统一设置到原型对象中。  
当我们访问一个对象的属性和方法时，会在对象自身中查找，找到了就直接使用。如果没找到，则会到原型对象中查找

因此对于 某个方法 只属于 某个函数对象的时候，可以考虑 用原型替代。

# jQuery 源码

## 1

(function(window, undefined) {

//do something

})(window);

* function(window,undefined) – 这是一个匿名函数，这里的window和undefined 只是两个参数，即传入两个变量，不具备任何意义。
* ()() – 这种是匿名函数自调用。 --匿名函数怎么调用，把自己括起来表示函数名后面的括号传递参数
* js没有重写。

function sum(num1,num2) {  
 return num1 + num2;  
}  
  
function sum(num1,num2,num3) {

console.log(num3);//undefined  
 return num1 + num2 + num3;  
}  
console.log(sum(1,2));  
  
var summ = function(num1,num2) {  
 return num1 + num2;  
};  
summ = function(num1,num2,num3) {  
 return num1 + num2 + num3;  
}

这里的函数名sum和summ只能指向一个函数对象。

* 不要被java坑了，函数的传参个数，与调用的时候并不用一致，未传的参数，都是undefined。

所以上面自调用传了一个window全局变量，（由于js的变量都是一层一层往外查找的，所以传变量window = window 作为局部变量，使用的时候不会因为在局部变量中找不到window，而需要到再往上查找一层。）

* 此处的undefined 并非未定义的undefined。而仅仅是一个变量/传参，由于自调用没有传参，所以此变量=未定义的undefined。

<https://blog.csdn.net/u013390913/article/details/53956618> --这个讲的不错。