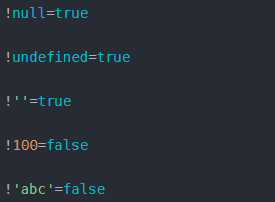
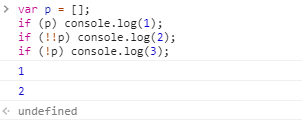
!与!!



前者用于求布尔值

后者判断布尔值后再进行判断，判断非空



Promise、async、await

promise可以更好的处理回调问题



[Promise 对象](http://es6.ruanyifeng.com/" \l "docs/promise)

async/await



[async 函数的含义和用法](http://www.ruanyifeng.com/blog/2015/05/async.html)

Js数组的join()方法，指定的字符分隔

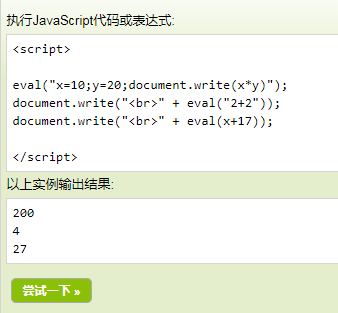


**当button标签在from表单中，而button又用到了ajax，那么可以请求成功，但是不回调，因为优先执行了form表单，可以提交。**

**解决方式：**

1. **在button标签中加入type=”button”属性**
2. **Ajax请求的时候把数据序列化data.serialize()。未尝试**
3. **表单使用ajax提交时，用该方法序列化**

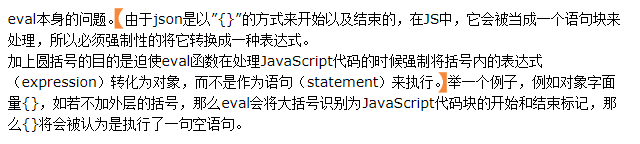
Js的全局函数，eval() 用于计算string字符串



也可以将json字符串转换成json对象。注意写法！



这样就把msg(json字符串)转成了json对象。

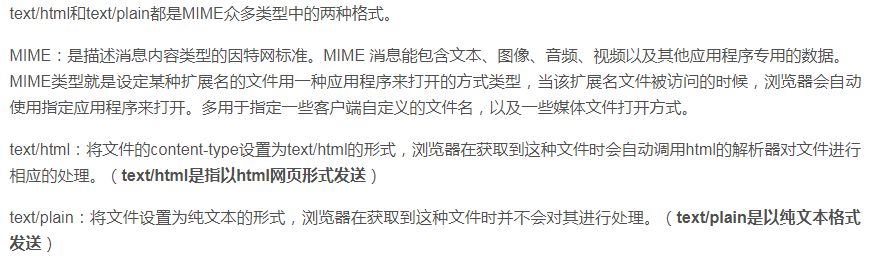


但是json字符串转json对象推荐，

[JSON.parse(“json字符串”)](https://www.runoob.com/json/json-parse.html)

[JSON.stringify(“JSON对象”);](https://www.runoob.com/json/json-stringify.html)

script type="text/plain"



**Js的push方法，用于向数组的后边添加元素**

**Js中拼接html代码时使用单引号，避免过多冗余，如**

特殊情况一：



特殊情况二：



在前段把json字符串(这里的data参数)转换成json对象，通过以下方法



或者



Js的toFixed函数，用于四舍五入，截取

str.toFixed(1)表示保留小数点后一位

Js的splice函数，用于截取数组

下标为i，截取一位

**window.history.go(-1)**

**`用于页面后退，-1表示后退至上一步**

**window.history.back()**

**`这个也是后退上一步**

**两者区别在于前者后退时刷新，后者后退时不刷新**

**要前进一页，只需要使用正数，例如**

**history.go(1)；**

页面关闭（卸载）时执行的方法

**window.onbeforeunload的return和没return的区别**

**如果return则出现弹窗，return的值不影响弹窗的内容**

**不return 则不出现弹窗。**

Jquery通过类名获取原生的DOM对象后应该如何取出对应的属性等信息？

直接 obj.attr 这样获取。既然是对象，当然这样获取 :-)

window.parent ,window.top,window.self

A页面中使用Iframe标签打开B页面，那么window.parent就是指页面A

B页面或者A页面中使用window.top就是指窗口即浏览器

B页面使用window.self指的时当前窗口

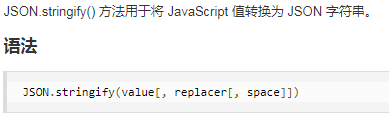
window.opener指的是通过open方法打开的页面

<https://blog.csdn.net/zdwzzu2006/article/details/6047632>

**在js中JSON对象与JSON字符串间的互转**

**jsonObject = JSON.parse(jsonString)**

**jsonString = JSON.stringify(jsonObject)**



<http://www.runoob.com/js/javascript-json-stringify.html>

**Js判断中文字符，可以通过unicode码配合正则来进行**

**中文的unicode码范围[\u4E00-\u9FA5]**

**判断一个字符是否为中文 /^[\u4E00-\u9FA5]$/.test(char)**

**区别正则中?、+、\*三个字符区别**

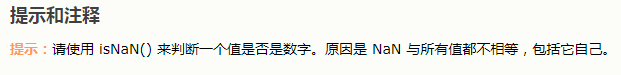
**?:出现一次或零次，最多一次**

**+:出现一次或多次，最少出现一次**

**\*:出现零次或多次，可以不出现**

**NaN属性，代表非数字值的特殊值。可用于判断是否是数字**

**通过全局函数isNaN()来进行判别**



<http://www.w3school.com.cn/jsref/jsref_nan_number.asp>

不是数值但进行数值的运算。



<https://www.imooc.com/qadetail/62358>

在同一个子页面下，最好不要使用相同的方法名。如，

A页面中嵌套了页面B、页面C，其中页面B、C的编辑操作方法名相同，那么这个时候执行C的编辑方法却是B的编辑方法。因为页面B、C对页面A来说，他们同属一层次，js文件的顺序影响了正常的方法(如果不同页面的方法名称相同)。

**Js的setTimeout**



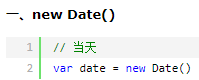
创建Date对象时safari的不兼容性(safari 5及以下)

<http://www.cnblogs.com/snandy/p/3992443.html>

**因为Safari(Safari5及以下版本)创建Date对象不支持new Date(“yyyy-MM-dd HH:mm”)的格式**

**所以更换其他的构造方法来进行构造Date对象，如**

**·直接new**



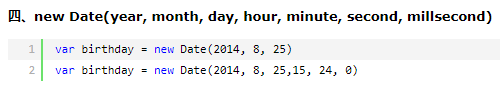
**·时间戳**

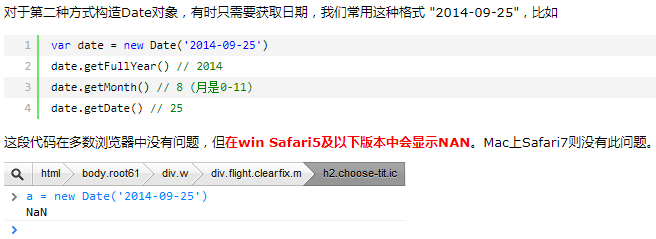


**·**

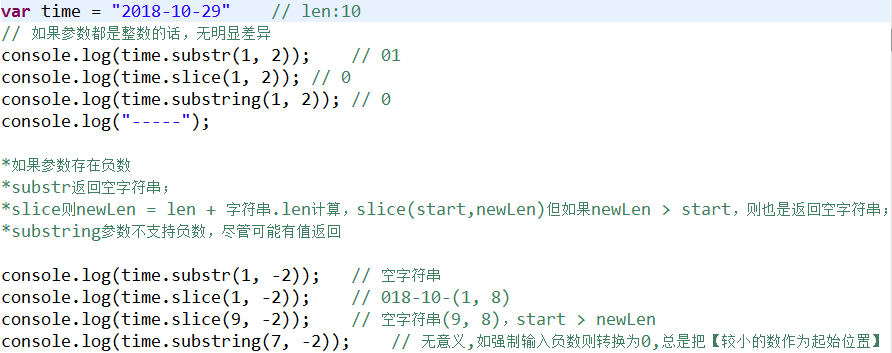


**·**



·

**截取字符串，substr(start,len)、slice(start,end)、substring(start,end)**

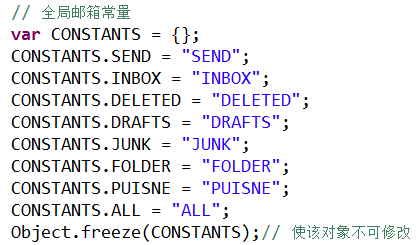
****

**slice相对其他几个更加灵活**

<http://www.w3school.com.cn/jsref/jsref_substr.asp>

<https://www.cnblogs.com/lmsblogs/p/5876384.html>

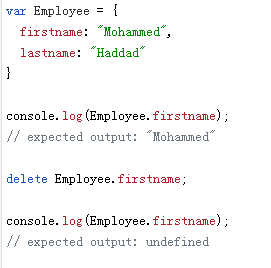
**Object.freeze()，这个可用于把一个变量变成常量，不可再次更改值。**



**Js的Date.parse()，用于把date字符串转换成Date对象**

**js的delete操作符**

**用于向删除对象的某个属性**



<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/delete>

事件冒泡：

父元素绑定了事件，子元素也绑定了事件，点击子元素的时候子元素的事件触发了，父元素不该触发的但也触发了。

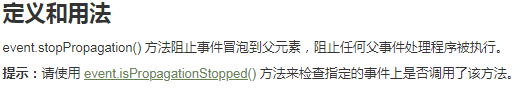
阻止事件冒泡方式之一：stopPropagation();

$("span").click(function(event){

event.stopPropagation();

alert("The span element was clicked.");

});

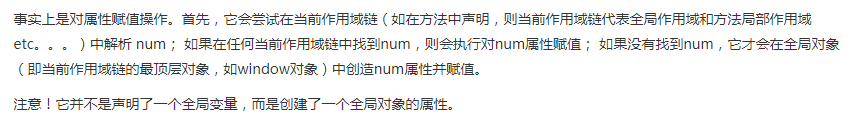


<http://www.runoob.com/jquery/event-stoppropagation.html>

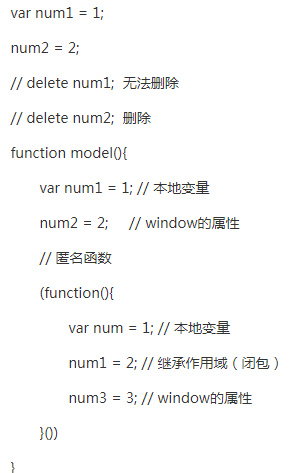
详解篇：

<http://caibaojian.com/javascript-stoppropagation-preventdefault.html>

JS加var和不加var

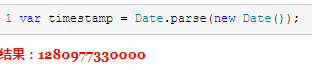


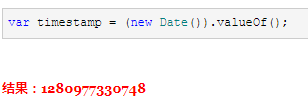




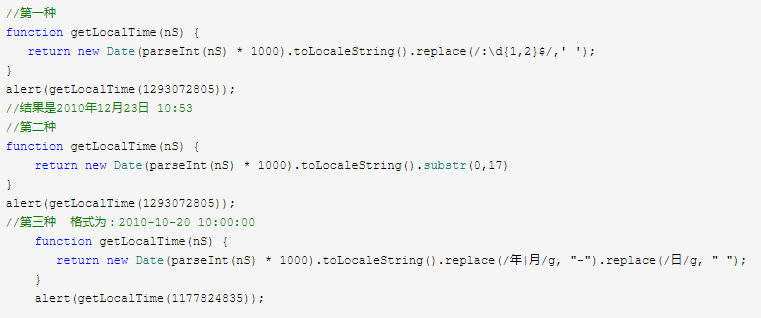
<https://www.cnblogs.com/liuna/p/6140901.html>

[js获取时间戳](https://www.cnblogs.com/lipcblog/p/6725347.html)



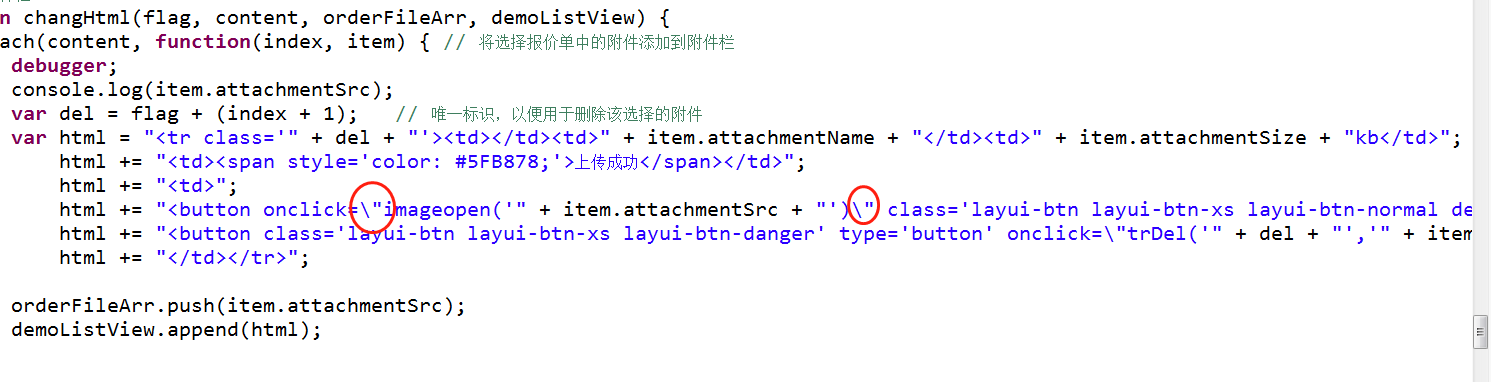


[时间戳转日期格式](https://www.cnblogs.com/lipcblog/p/6725347.html)



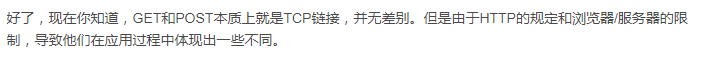
js中拼接字符串问题

class拼接时加入了引号(单引号|双引号)那么需要绑定的事件也需要加入引号(单引号|双引号)，以防解析错误。



[GET & POST](https://www.cnblogs.com/logsharing/p/8448446.html)

(前者通过url传值，后者把值放在Request body)



数据返回null时不要拼接成字符串拼接成字符串就是字符串了。之前是null也不例外

[js定时任务](http://www.runoob.com/jsref/met-win-settimeout.html)

setInterval，循环执行相应的方法（清除，clearInterval）



setTimeout，规定时间后执行（清除，clearTimeout）



[逻辑运算符&&、||，位运算符|，&](https://www.cnblogs.com/yuanxinghuo/p/7881366.html)

// 如果没有，则定义（||的后边必须为值，不能是表达式）

moduleCache || (moduleCache = {});

||，不存在则修改值

&&，存在则修改值

var a = null;

a || (a = 2);

console.log(a); // 2

var aa = null;

aa && (aa = 5); // null

console.log(aa);

var a1 = 1;

a1 || (a1 = 2);

console.log(a1); // 1

var aa1 = 4;

aa1 && (aa1 = 5);

console.log(aa1); // 5

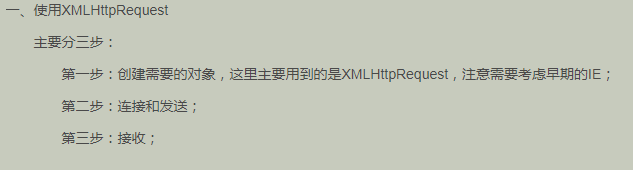
[页面刷新（手动刷新&自动刷新）](https://www.cnblogs.com/Chen-XiaoJun/p/6219880.html)

var a = “a”;

!!a;

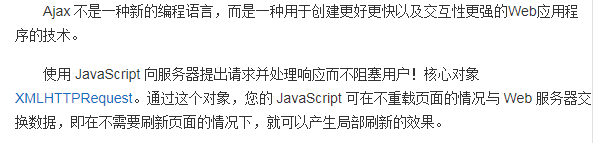
如果a有值则返回true，无值返回false

[JS发送GET、POST请求](https://www.cnblogs.com/Im-Victor/p/9405974.html)





AJAX：



[https://baike.baidu.com/item/ajax/8425?fr=aladdin#3](https://baike.baidu.com/item/ajax/8425?fr=aladdin" \l "3)

JS原生请求



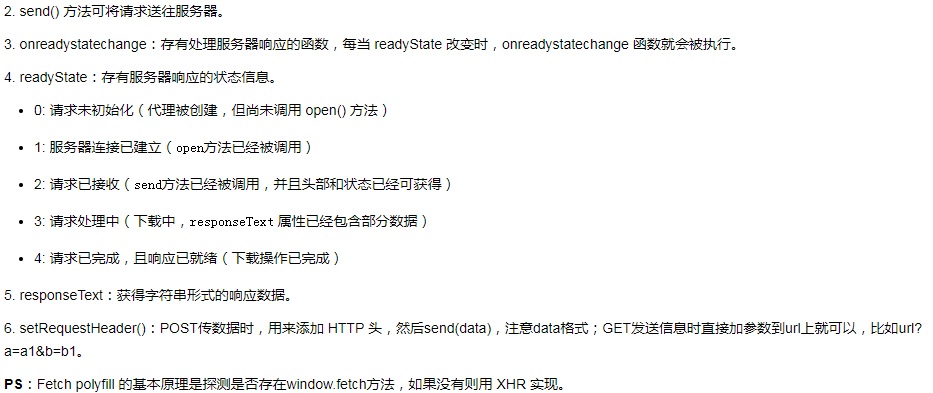
1. open(method, url, async) 方法需要三个参数:

**method**：发送请求所使用的方法（GET或POST）；与POST相比，GET更简单也更快，并且在大部分情况下都能用；然而，在以下情况中，请使用POST请求：

* 无法使用缓存文件（更新服务器上的文件或数据库）
* 向服务器发送大量数据（POST 没有数据量限制）
* 发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

**url**：规定服务器端脚本的 URL(该文件可以是任何类型的文件，比如 .txt 和 .xml，或者服务器脚本文件，比如 .asp 和 .php （在传回响应之前，能够在服务器上执行任务）)；

**async**：规定应当对请求进行异步（true）或同步（false）处理；true是在等待服务器响应时执行其他脚本，当响应就绪后对响应进行处理；false是等待服务器响应再执行。



<https://www.cnblogs.com/colima/p/5339227.html>

简述

<https://blog.csdn.net/u013953289/article/details/32335591>

HTTP协议异步通信，

1、创建对象

2、调用对象open方法

2.1如果时POST请求则需要设置头(setRequestHeader)

3、设置回调方法

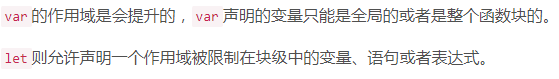
4、send

[js Date对象](http://www.w3school.com.cn/jsref/jsref_obj_date.asp)

获取一个月中的某天（1~31）getDate()

获取一周中的某天（0~6）getDay()

[js的let关键字](https://blog.csdn.net/w157z0372x5580/article/details/75331754)







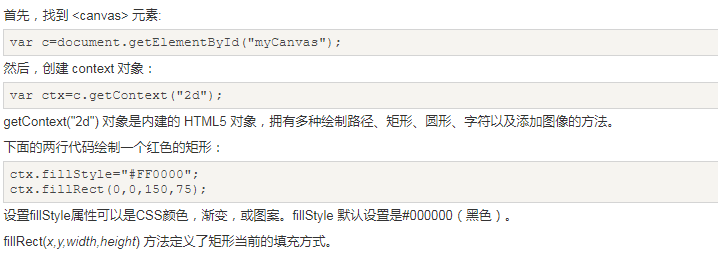


<http://www.runoob.com/jsref/met-document-queryselector.html>

[canvas](http://www.runoob.com/html/html5-canvas.html)

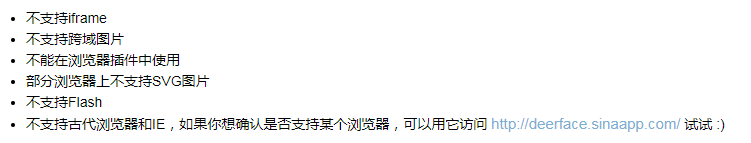
这只是一个容器





[html2canvas用于把特定的html画进画布](https://www.cnblogs.com/yanweidie/p/5203943.html)

注意事项：



实例：



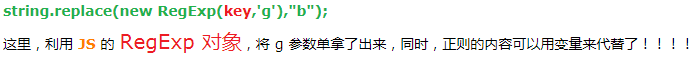
[替换某串字符](https://www.cnblogs.com/jasonlam/p/7070604.html)

构造正则

注：参数g表示全文搜索



这里的key表示一个变量，可以是某个字符串





<http://www.runoob.com/js/js-obj-regexp.html>

[清空数组](https://www.jb51.net/article/109170.htm)

var arr = new Array();

arr = 0;

arr= []; // 推荐

Arr.split(0, arr.length);

[原生获取DOM属性](http://www.runoob.com/jsref/prop-node-attributes.html)

先拿到DOM对象，如TagName

1 先获取节点的属性集

TagName.attributes

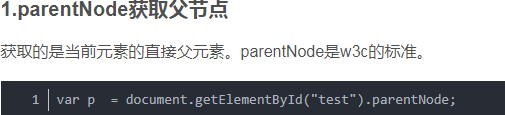
2 再从属性集中取相应的属性

TagName.attributes.title

3 接着取里边的值

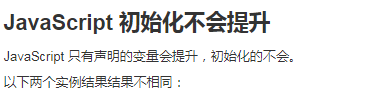
TagName.attributes.title.nodeValue

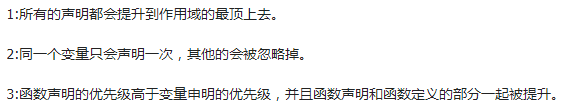
[js获取子节点、父节点和兄弟节点的若干种方式](https://blog.csdn.net/laok_/article/details/75760572)



**变量提升（**函数声明和变量声明总是会被解释器悄悄地被"提升"到方法体的最顶部**）**

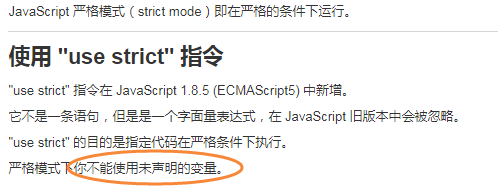


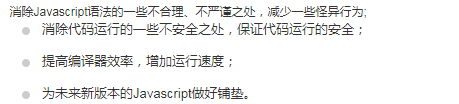


**变量提升**<http://www.runoob.com/js/js-hoisting.html>

<https://www.jb51.net/article/140720.htm>

**严格模式**

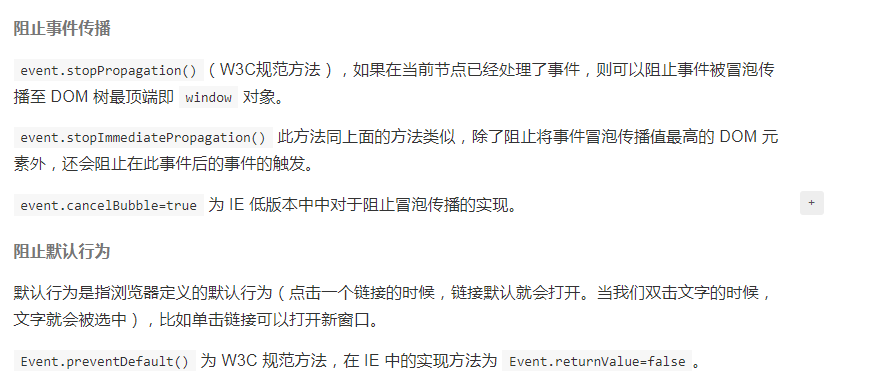




<http://www.runoob.com/js/js-strict.html>

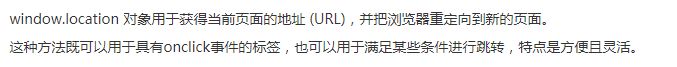
事件传播(冒泡)、默认行为

每一个事件发生时都会有一个Event对象，但各浏览器对它的支持不一致。

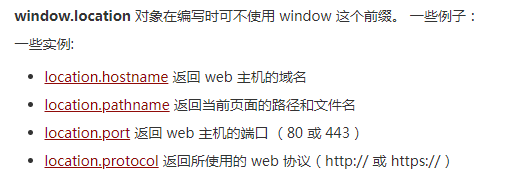


[https://li-xinyang.gitbooks.io/frontend-notebook/content/chapter3/05\_events.html#window](https://li-xinyang.gitbooks.io/frontend-notebook/content/chapter3/05_events.html" \l "window)

Window Location对象



比如：window.location.href=”newUrl”



<https://www.w3cschool.cn/javascript/js-window-location.html>

application/x-www-form-urlencoded

在form元素的语法中，encType表明提交数据的格式 用 enctype 属性指定将数据**回发到服务器**时浏览器使用的**编码类型**。

application/x-www-form-urlencoded：窗体数据被编码为名称/值对。这是标准的编码格式。（默认）

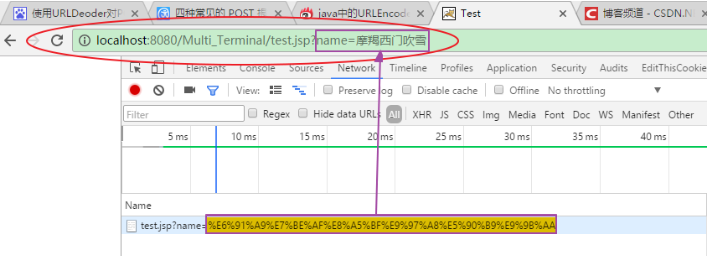
multipart/form-data：窗体数据被编码为一条消息，页上的每个控件对应消息中的一个部分。

当action为get时候，浏览器用x-www-form-urlencoded的编码方式把form数据转换成一个字串（name1=value1& amp; amp;name2=value2...），然后把这个字串append到url后面，用?分割，加载这个新的url。

当action为post时候，浏览器把form数据封装到http body中，然后发送到server。如果没有type=file的控件，用默认的application/x-www-form-urlencoded就可以了。但是如果有 type=file的话，就要用到multipart/form-data了。浏览器会把整个表单以控件为单位分割，并为每个部分加上 Content-Disposition(form-data或者file),Content-Type(默认为text/plain),name(控件 name)等信息，并加上分割符(boundary)。

**[ajax中的application/x-www-form-urlencoded中的使用](https://www.cnblogs.com/renzh/articles/5413090.html)**

使用 URLDecoder 和 URLEncoder 对中文字符进行编码和解码



当URL地址里包含非西欧字符的字符串时，浏览器都会将这些非西欧字符串转换成application/x-www-form-urlencoded MIME 字符串。

那么在Java中可以对其进行编码及解码。

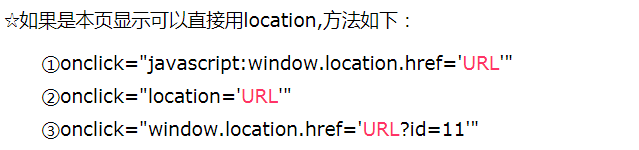
URLDecoder类包含一个decode(String s,String enc)静态方法，它可以将application/x-www-form-urlencoded MIME字符串转成普通字符串；

URLEncoder类包含一个encode(String s,String enc)静态方法，它可以将普通字符串转换成application/x-www-form-urlencoded MIME字符串。



# [使用 URLDecoder 和 URLEncoder 对中文字符进行编码和解码](https://blog.csdn.net/justloveyou_/article/details/57156039)

onclick跳转url



<https://www.cnblogs.com/JuneZhang/archive/2010/11/25/1887575.html>

JavaScript splice() 方法

array.splice(index,howmany,item1,.....,itemX)

index规定从何处添加/删除元素

howmany不为0时，则表示对数组进行删除，数值表示删除的个数

item1…就对数组进行了新增

[typeof和instanceof](https://www.cnblogs.com/double405/p/5326311.html)

 typeof用以获取一个变量或者表达式的类型，typeof一般只能返回如下几个结果：

number,boolean,string,function（函数）,object（NULL,数组，对象）,undefined。

如，console.log(typeof Function (){} === “function”);

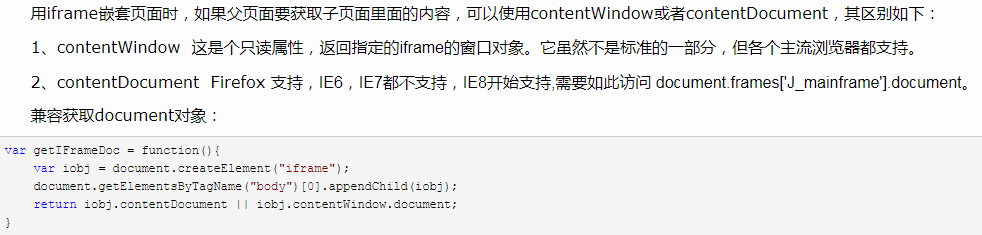
instanceof用于判断一个变量是否某个对象的实例，如var a=new Array();alert(a instanceof Array);会返回true，

同时alert(a instanceof Object)也会返回true;这是因为Array是object的子类。

contentDocument

**contentDocument 属性能够以 HTML 对象来返回 iframe 中的文档，可以通过所有标准的 DOM 方法来处理被返回的对象**。

语法：frameObject.contentWindow或者 iframeObject.contentWindow（不是jquery对象）



操作iframe中的对象

1、**先获取iframe里面的window对象，再通过这个对象，获取到里面的DOM元素**

例子：

var ifr = document.getElementById("iframe");

ifr.contentWindow.document.getElementById("XXXXX")

<iframe src="a.html" id=""></iframe>



Iframe操作父页面的DOM

二、在iframe本页面，要操作这个iframe的父页面的DOM元素（即嵌套这个iframe的页面）可以用：

　　window.parent、window.top(这里的TOP是获取的顶层，即有多层嵌套iframe的时候使用)

var ifr = document.getElementByTagName("iframe");

ifr.parent.document.getElementById("XXXXX")

<iframe src="a.html" id=""></iframe>

实例：

top.$("iframe[name='iframeWindow']")[0].**contentWindow**.$("#inside\_tableElement")，这样才能获取到iframe里的元素，

　　注意：top.$("iframe[name='iframeWindow']").eq(0).$("#inside\_tableElement")，是获取不到的

<https://www.cnblogs.com/goloving/p/7828070.html>

立即执行函数(function (){})();

顾名思义就是声明后立即执行。前置条件，了解函数声明、函数表达式、匿名函数

* 函数声明：使用function声明函数，并指定函数名。

function setFn() {

// coding

}

* 函数表达式：使用function声明函数，但未指定函数名，将匿名函数赋予一个变量。

var setFn = function() {

// coding

}

* 匿名函数：使用function声明函数，但未指定函数名。匿名函数属于函数表达式，匿名函数有很多作用。

1. 赋予一个变量则创建函数；2、赋予一个事件则成为事件处理程序或创建闭包（函数里边嵌函数）等等。

function() {

// coding

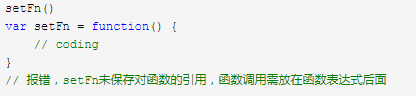
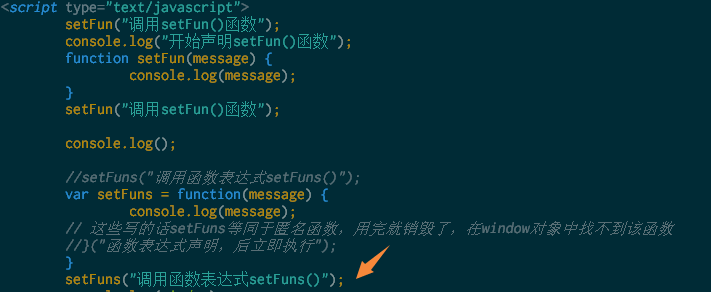
}

函数声明 & 函数表达式区别：

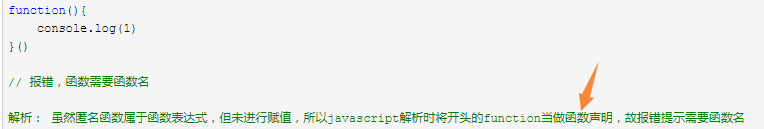
前者可以调用顺序跟声明顺序无关，只要声明了。如，先调用后声明、先声明后调用。

后者只能是声明了再调用，也就是执行了该表达式才能执行，不能提前调用。

实例：



匿名函数声明后立即调用



立即执行函数里面的函数**必须是**函数表达式，所以由var setFn = function(){}();可以理解为在匿名函数前加了‘=’运算符后，将函数声明**转化为**函数表达式，所以拿‘!’‘+’‘-’‘( )’...等运算符来测试下是否如此。

!function(){

console.log(1)

}()// 1

+function(){

console.log(2)

}()// 2

-function(){

console.log(3)

}()// 3

(function(){

console.log(4)

})()// 4

推荐使用括号，觉得可以划出函数边界，能避免不必要的麻烦。

<https://www.cnblogs.com/yanzp/p/6371292.html>

js立即执行函数的作用

##### 立即执行函数会形成一个单独的作用域，我们可以封装一些临时变量或者局部变量，避免污染全局变量

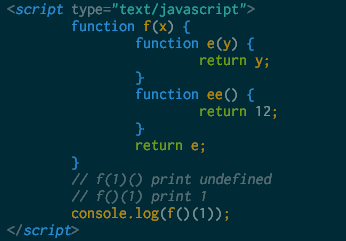
<https://www.jianshu.com/p/b10b6e93ddec>

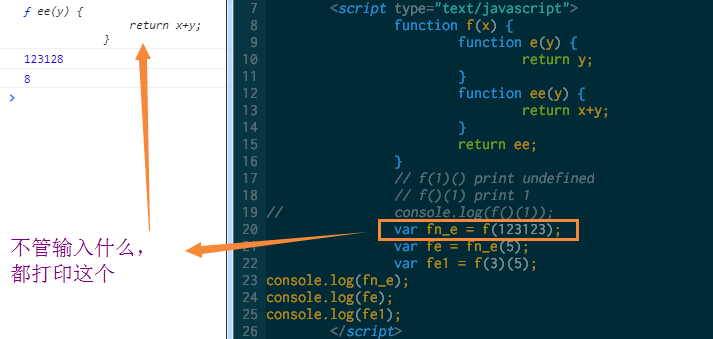
在ES6中，声明局部作用域的话，只需要{ // 一对花括号就行了 }

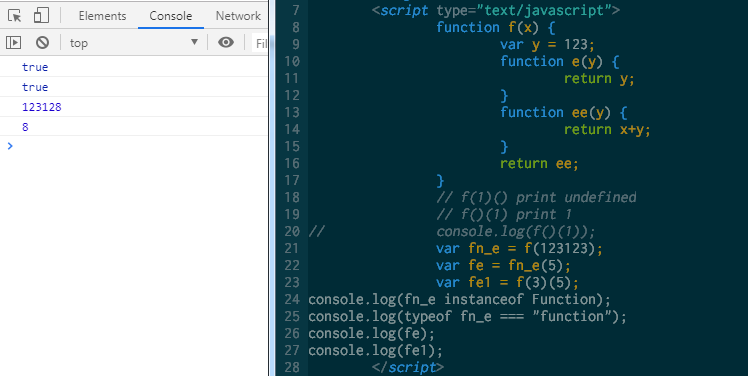
[为什么要用闭包(closure)？](https://www.jianshu.com/p/11fb0195dc9d)

闭包的使用









而后面的var fe = fn\_e(5);则对fn\_e()进行再次调用

<https://www.cnblogs.com/litings/p/7817146.html>

<https://www.cnblogs.com/leezhxing/p/4103893.html>

jquery的ready()与window.onload()

尽管两者都是等到页面加载完成后再执行，但是，前者只需要加载完DOM结构即可执行，后者则是加载时更彻底些(包括图片等)。



[jquery 的ready() 与window.onload()的区别](https://www.cnblogs.com/wsun/p/3916487.html)

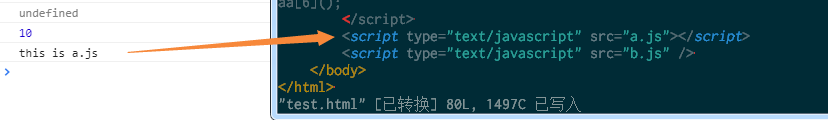
页面中引入JS

1、<script tpye=”text/javascript”>do something</script>

2、<script type=”text/javascript” src=”/test.js></script>

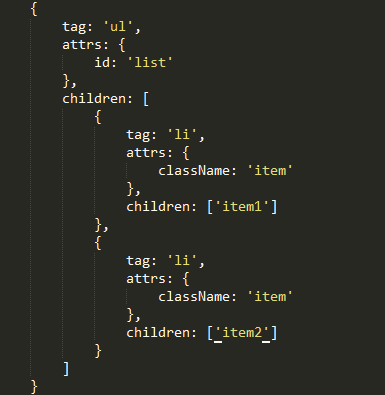
3、<script type=”text/javascript” src=”https://www.xxx.com/test/test.js”></script>





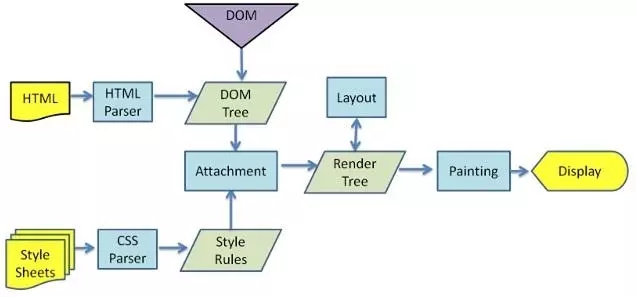
虚拟DOM（简单了解）

由facebook提出，随react的诞生而诞生。

vdom可以看作是一个使用JavaScript模拟了DOM结构的树形结构，这个树结构包含整个DOM结构的信息

为什么需要虚拟DOM（vdom：Virtual DOM）

先看下webkit引擎的处理流程：



大致五步走：创建DOM tree –> 创建Style Rules -> 构建Render tree -> 布局Layout –> 绘制Painting

第一步：用HTML分析器，分析HTML元素，构建一颗DOM树。

第二步：用CSS分析器，分析CSS文件和元素上的inline样式，生成页面的样式表。

第三步：将上面的DOM树和样式表，关联起来，构建一颗Render树。这一过程又称为Attachment。每个DOM节点都有attach方法，接受样式信息，返回一个render对象（又名renderer）。这些render对象最终会被构建成一颗Render树。

第四步：有了Render树后，浏览器开始布局，会为每个Render树上的节点确定一个在显示屏上出现的精确坐标值。

第五步：Render数有了，节点显示的位置坐标也有了，最后就是调用每个节点的paint方法，让它们显示出来。

因此，当你用传统的源生api或jQuery去操作DOM时，浏览器会从构建DOM树开始从头到尾执行一遍流程。比如当你在一次操作时，需要更新10个DOM节点，理想状态是一次性构建完DOM树，再执行后续操作。但浏览器没这么智能，收到第一个更新DOM请求后，并不知道后续还有9次更新操作，因此会马上执行流程，最终执行10次流程。显然例如计算DOM节点的坐标值等都是白白浪费性能，可能这次计算完，紧接着的下一个DOM更新请求，这个节点的坐标值就变了，前面的一次计算是无用功。所以vdom就是将操作累计起来,统计计算出所有的变化后，统一更新一次DOM。

# 减少计算机性能消耗、解耦合便于维护

# [虚拟DOM介绍](https://www.jianshu.com/p/616999666920)

**[什么是虚拟DOM](https://www.cnblogs.com/gaosong-shuhong/p/9253959.html)**

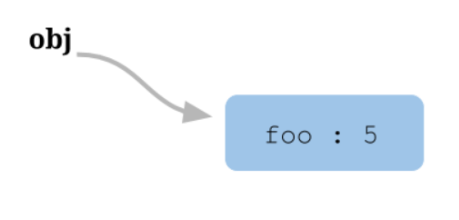
this 原理

this指的是函数运行时所在的环境。之所以有this的设计，跟内存里面的数据结构有关系。

1、内存的数据结构

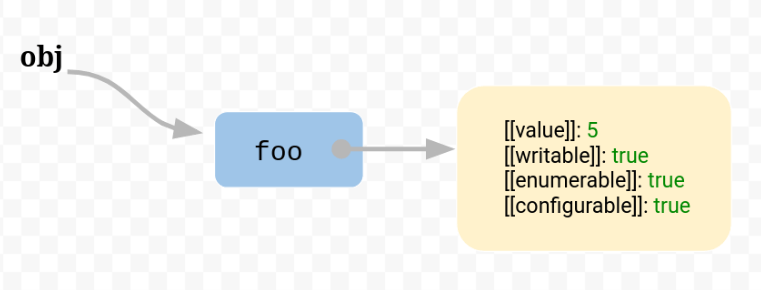


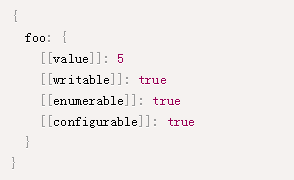
上面的代码将一个对象赋值给变量obj。JavaScript 引擎会先在内存里面，生成一个对象{ foo: 5 }，然后把这个对象的内存地址赋值给变量obj。



也就是说，变量obj是一个地址（reference）。后面如果要读取obj.foo，引擎先从obj拿到内存地址，然后再从该地址读出原始的对象，返回它的foo属性。

原始的对象以字典结构保存，每一个属性名都对应一个属性描述对象。举例来说，上面例子的foo属性，实际上是以下面的形式保存的。



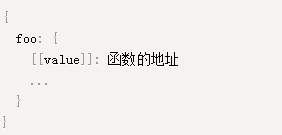
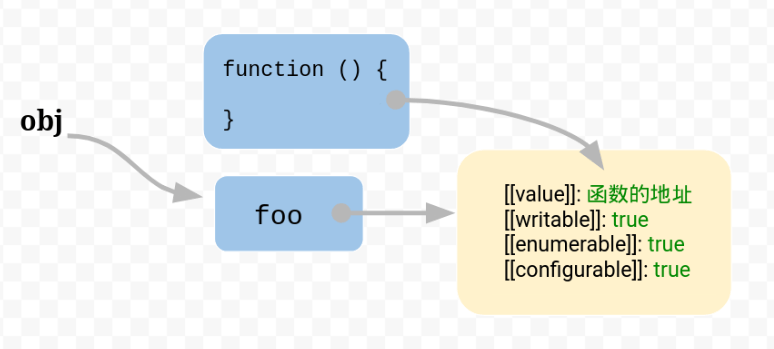


注意，foo属性的值保存在属性描述对象的value属性里面。

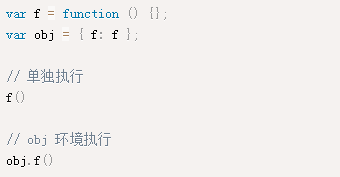
2、函数



这时，引擎会将函数单独保存在内存中，然后再将函数的地址赋值给foo属性的value属性



由于函数是一个单独的值，所以它可以在不同的环境（上下文）执行。



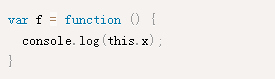
3、环境变量

JavaScript 允许在函数体内部，引用当前环境的其他变量

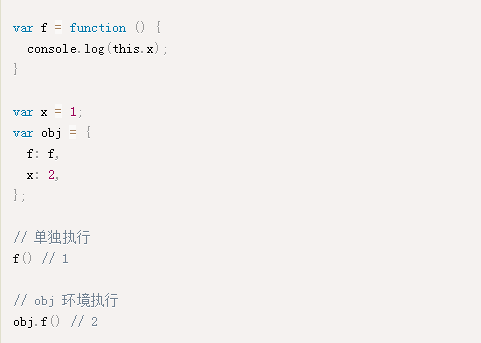


上面代码中，函数体里面使用了变量x。该变量由运行环境提供。

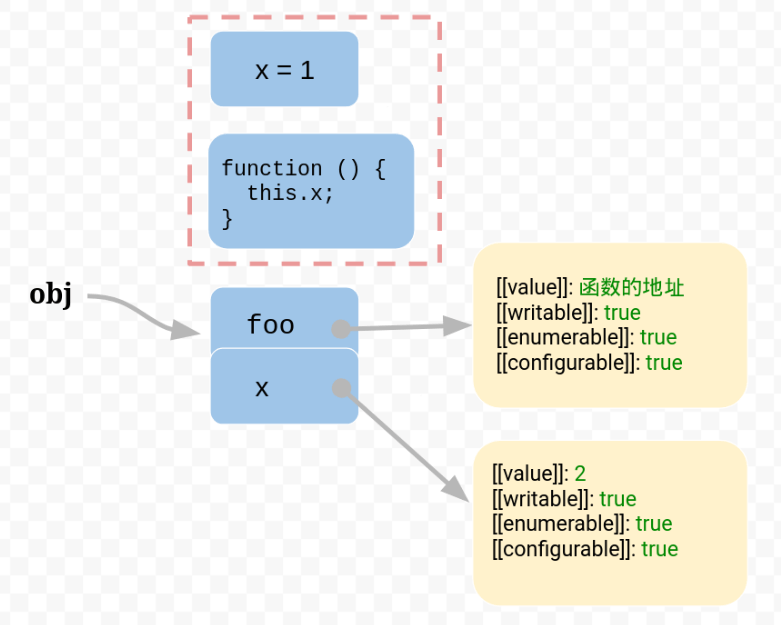
现在问题就来了，**由于函数可以在不同的运行环境执行**，所以需要有一种机制，**能够在函数体内部获得当前的运行环境（context）**。所以，**this就出现了，它的设计目的就是在函数体内部，指代函数当前的运行环境**。



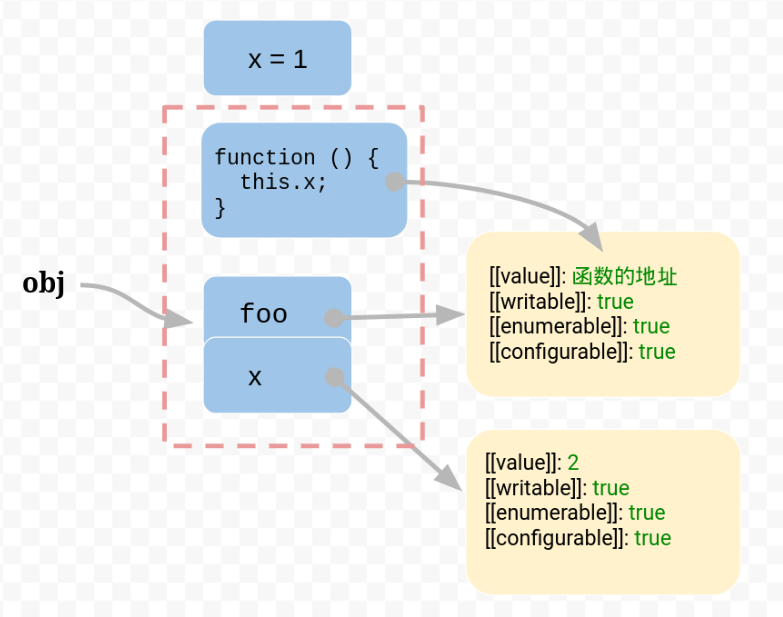
上面代码中，函数体里面的this.x就是指当前运行环境的x。



上面代码中，函数f在全局环境执行，this.x指向全局环境的x。



在obj环境执行，this.x指向obj.x。



**obj.foo()是通过obj找到foo，所以就是在obj环境执行。一旦var foo = obj.foo，变量foo就直接指向函数本身，所以foo()就变成在全局环境执行。**

[JavaScript 的 this 原理](http://www.ruanyifeng.com/blog/2018/06/javascript-this.html)

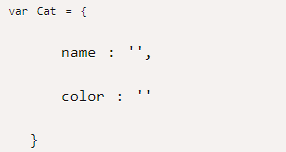
JavaScript的封装

JavaScript是一种基于对象（object-based）的语言，你遇到的所有东西几乎都是对象。但是，它又不是一种真正的面向对象编程（OOP）语言，因为它的语法中没有class（类）。

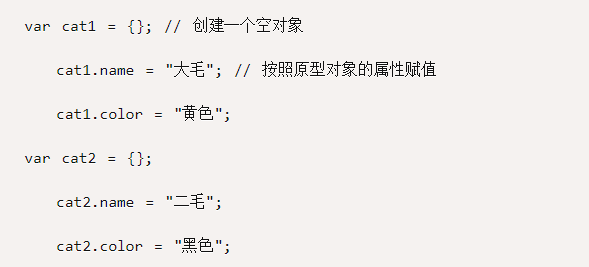
那么，如果我们要把"属性"（property）和"方法"（method），封装成一个对象，甚至要从原型对象生成一个实例对象，我们应该怎么做呢？

一、生成实例对象的原始模式

假定我们把猫（Cat）看成一个对象，它有"名字"和"颜色"两个属性。



现在，我们需要根据这个原型对象的规格（schema），生成两个实例对象。



好了，**这就是最简单的封装了**，把两个属性封装在一个对象里面。

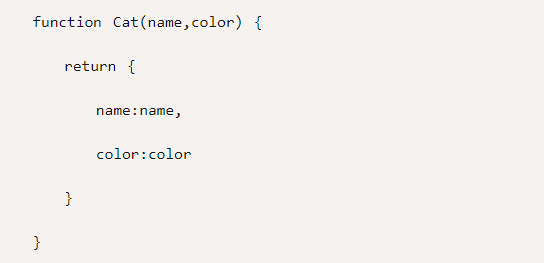
但是，这样的写法有两个缺点：

1、是如果多生成几个实例，写起来就非常麻烦；

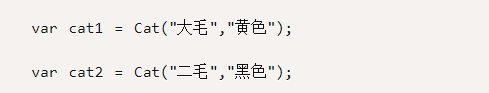
2、是实例与原型之间，没有任何办法，可以看出有什么联系。

二、原始模式的改进

我们可以写一个函数，解决代码重复的问题。



然后生成实例对象，就等于是在调用函数：



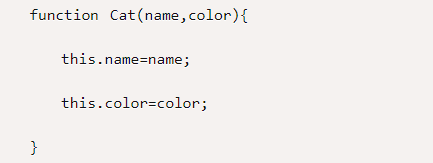
这种方法的问题依然是，cat1和cat2之间没有内在的联系，不能反映出它们是同一个原型对象的实例。

三、构造函数模式

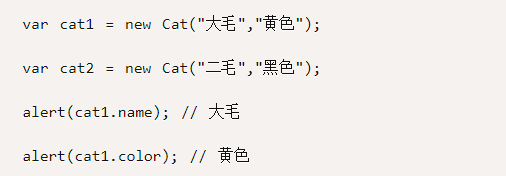
为了解决从原型对象生成实例的问题，Javascript提供了一个构造函数（**Constructor**）模式。

所谓"构造函数"，其实就是一个普通函数，但是内部使用了this变量。对构造函数使用new运算符，就能生成实例，并且this变量会绑定在实例对象上。

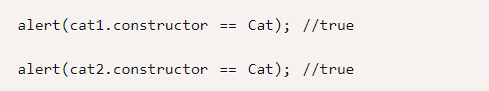
比如，猫的原型对象现在可以这样写



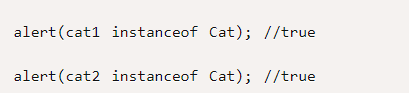
我们现在就可以生成实例对象了。



这时cat1和cat2会自动含有一个**constructor**属性，指向它们的构造函数。



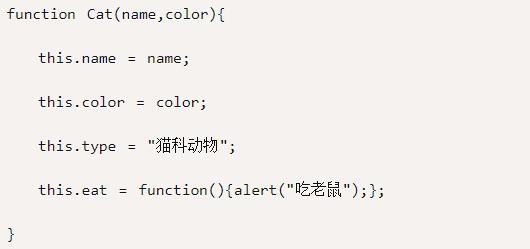
Javascript还提供了一个instanceof运算符，验证原型对象与实例对象之间的关系。



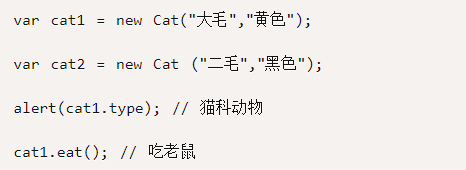
四、构造函数模式的问题

构造函数方法很好用，但是存在一个浪费内存的问题。

请看，我们现在为Cat对象添加一个不变的属性type（种类），再添加一个方法eat（吃）。那么，原型对象Cat就变成了下面这样：



还是采用同样的方法，生成实例：



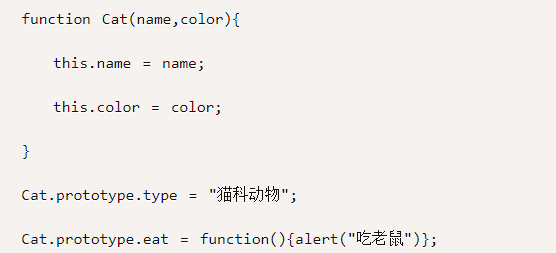
表面上好像没什么问题，但是实际上这样做，有一个很大的弊端。那就是对于每一个实例对象，type属性和eat()方法都是一模一样的内容，每一次生成一个实例，都必须为重复的内容，多占用一些内存。这样既不环保，也缺乏效率。



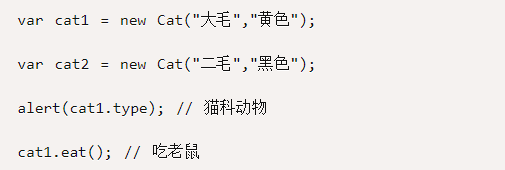
能不能让type属性和eat()方法在内存中只生成一次，然后所有实例都指向那个内存地址呢？回答是可以的。

五、Prototype模式

Javascript规定，每一个**构造函数**都有一个**prototype**属性，指向另一个对象（如Cat）。这个对象的所有属性和方法，都会被构造函数的实例继承。这意味着，我们可以把那些不变的属性和方法，直接定义在prototype对象上。



然后，生成实例。



这时所有实例的type属性和eat()方法，其实都是同一个内存地址，指向prototype对象，因此就提高了运行效率。

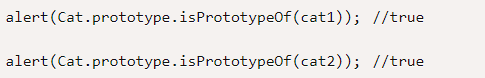


六、Prototype模式的验证方法

为了配合prototype属性，Javascript定义了一些辅助方法，帮助我们使用它。

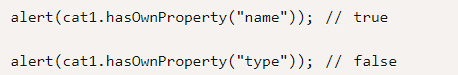
6.1 isPrototypeOf()

这个方法用来判断，某个proptotype对象和某个实例之间的关系。



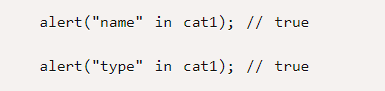
6.2 hasOwnProperty()

每个实例对象都有一个hasOwnProperty()方法，用来判断某一个属性到底是本地属性，还是继承自prototype对象的属性。



6.3 in运算符

除了可以用来判断某个实例是否含有某个属性，不管是不是本地属性。



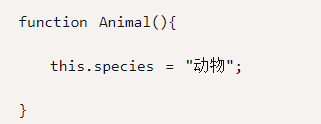
还可以用来遍历某个对象的所有属性



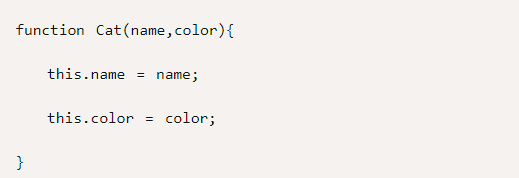
JavaScript构造函数的继承

对象之间的“继承”的五种方法

比如，现在有一个"动物"对象的构造函数。



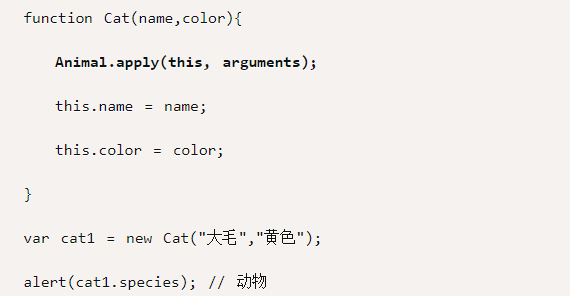
还有一个"猫"对象的构造函数。



怎样才能使"猫"继承"动物"呢？

一、 构造函数绑定

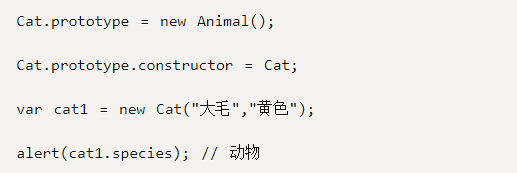
第一种方法也是最简单的方法，**使用call或apply方法，将父对象的构造函数绑定在子对象上**，即在子对象构造函数中加一行：



二、 prototype模式

第二种方法更常见，使用prototype属性。

如果"猫"的prototype对象，指向一个Animal的实例，那么所有"猫"的实例，就能继承Animal了。



代码的第一行，我们将Cat的prototype对象指向一个Animal的实例。



它相当于完全删除了prototype 对象原先的值，然后赋予一个新值。但是，第二行又是什么意思呢？



原来，**任何一个prototype对象都有一个constructor属性，指向它的构造函数**。

如果没有"Cat.prototype = new Animal();"这一行，Cat.prototype.constructor是指向Cat的；加了这一行以后，Cat.prototype.constructor指向Animal。



更重要的是，每一个实例也有一个constructor属性，默认调用prototype对象的constructor属性。



因此，在运行"Cat.prototype = new Animal();"这一行之后，cat1.constructor也指向Animal！



**这显然会导致继承链的紊乱（cat1明明是用构造函数Cat生成的），因此我们必须手动纠正，将Cat.prototype对象的constructor值改为Cat。这就是第二行的意思。**

这是很重要的一点，编程时务必要遵守。下文都遵循这一点，即如果替换了prototype对象，



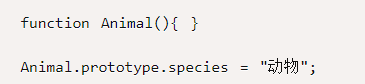
那么，下一步必然是为新的prototype对象加上constructor属性，并将这个属性指回原来的构造函数。



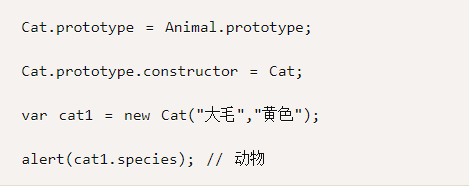
三、 直接继承prototype

第三种方法是对第二种方法的改进。由于Animal对象中，不变的属性都可以直接写入Animal.prototype。所以，我们也可以让Cat()跳过 Animal()，直接继承Animal.prototype。

现在，我们先将Animal对象改写：



然后，将Cat的prototype对象，然后指向Animal的prototype对象，这样就完成了继承。



与前一种方法相比，这样做的优点是效率比较高（不用执行和建立Animal的实例了），比较省内存。缺点是 Cat.prototype和Animal.prototype现在指向了同一个对象，那么任何对Cat.prototype的修改，都会反映到Animal.prototype。

所以，上面这一段代码其实是有问题的。请看第二行

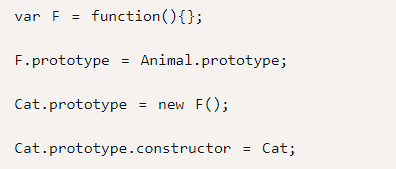


这一句实际上把Animal.prototype对象的constructor属性也改掉了！



四、 利用空对象作为中介

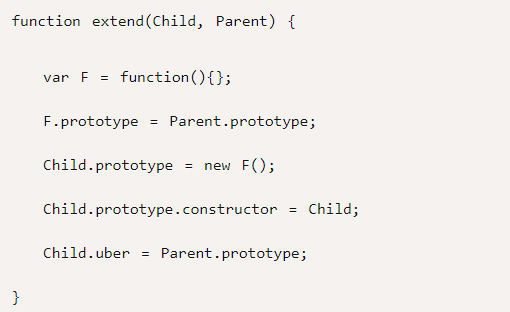
由于"直接继承prototype"存在上述的缺点，所以就有第四种方法，利用一个空对象作为中介。



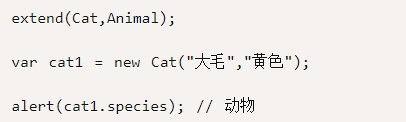
F是空对象，所以几乎不占内存。这时，修改Cat的prototype对象，就不会影响到Animal的prototype对象。



我们将上面的方法，封装成一个函数，便于使用。



使用的时候，方法如下



另外，说明一点，函数体最后一行

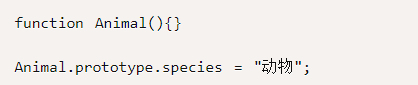


意思是为子对象设一个uber属性，这个属性直接指向父对象的prototype属性。（uber是一个德语词，意思是"向上"、"上一层"。）这等于在子对象上打开一条通道，可以直接调用父对象的方法。这一行放在这里，只是为了实现继承的完备性，纯属备用性质。

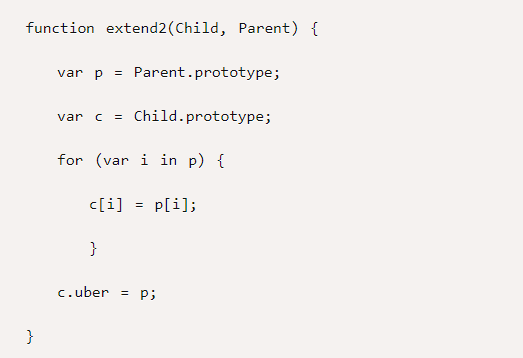
五、 拷贝继承

上面是采用prototype对象，实现继承。我们也可以换一种思路，纯粹采用"拷贝"方法实现继承。简单说，如果把父对象的所有属性和方法，拷贝进子对象，不也能够实现继承吗？这样我们就有了第五种方法。

首先，还是把Animal的所有不变属性，都放到它的prototype对象上.

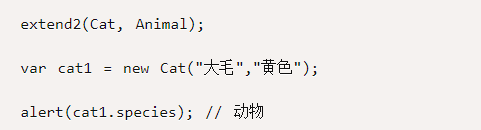


然后，再写一个函数，实现属性拷贝的目的。



这个函数的作用，就是将父对象的prototype对象中的属性，一一拷贝给Child对象的prototype对象。

使用的时候，这样写：



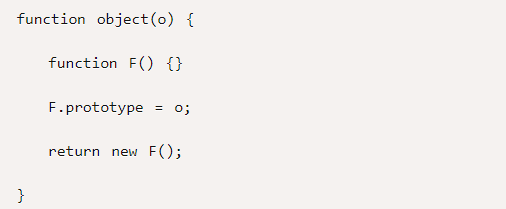
[Javascript面向对象编程（二）：构造函数的继承](http://www.ruanyifeng.com/blog/2010/05/object-oriented_javascript_inheritance.html)

什么是"非构造函数"的继承？

这两个对象都是普通对象，不是构造函数，无法使用构造函数方法实现"继承"。

一、“object()”函数（prototype链）

json格式的发明人Douglas Crockford，提出了一个object()函数，可以做到这一点。



这个object()函数，其实只做一件事，就是把子对象的prototype属性，指向父对象，从而使得子对象与父对象连在一起。

使用的时候，第一步先在父对象的基础上，生成子对象：



然后，再加上子对象本身的属性：



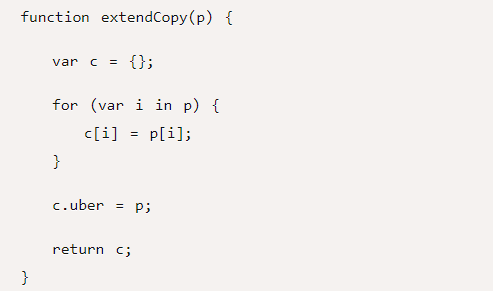
这时，子对象已经继承了父对象的属性了。



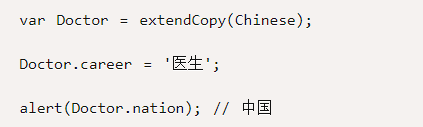
二、浅拷贝

除了使用"prototype链"以外，还有另一种思路：把父对象的属性，全部拷贝给子对象，也能实现继承。

下面这个函数，就是在做拷贝：



使用的时候，这样写：



但是，这样的拷贝有一个问题。那就是，如果父对象的属性等于数组或另一个对象，那么实际上，**子对象获得的只是一个内存地址，而不是真正拷贝，因此存在父对象被篡改的可能。**

请看，现在给Chinese添加一个"出生地"属性，它的值是一个数组。



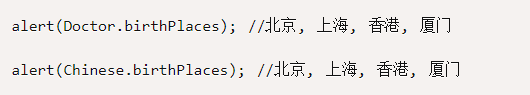
通过extendCopy()函数，Doctor继承了Chinese。



然后，我们为Doctor的"出生地"添加一个城市：



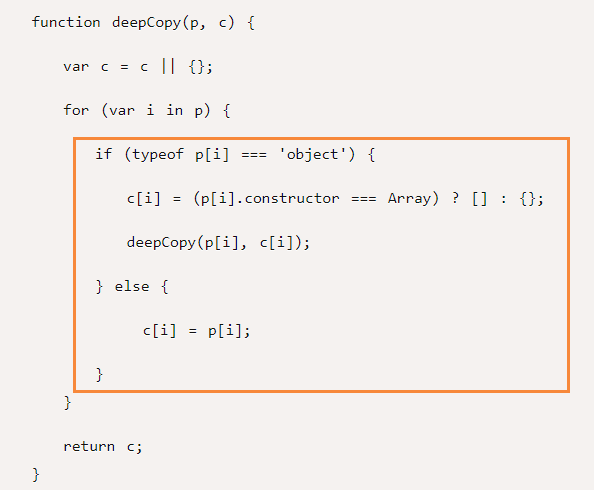
发生了什么事？Chinese的"出生地"也被改掉了！



所以，extendCopy()只是拷贝基本类型的数据，我们把这种拷贝叫做"浅拷贝"。这是早期jQuery实现继承的方式。

三、深拷贝

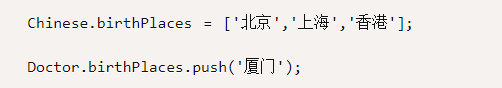
所谓"深拷贝"，就是能够实现真正意义上的数组和对象的拷贝。它的实现并不难，只要递归调用"浅拷贝"就行了。



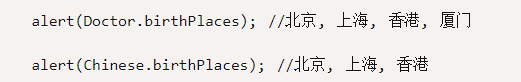
使用的时候这样写：



现在，给父对象加一个属性，值为数组。然后，在子对象上修改这个属性：



这时，父对象就不会受到影响了。



[Javascript面向对象编程（三）：非构造函数的继承](http://www.ruanyifeng.com/blog/2010/05/object-oriented_javascript_inheritance_continued.html)

Javascript中undefined与null的区别

null表示“没有对象"，即该处不应该有值”。典型用法是：

（1） 作为函数的参数，表示该函数的参数不是对象。

（2） 作为对象原型链的终点。



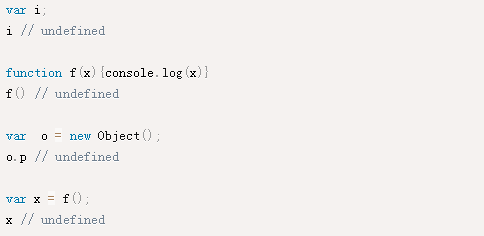
undefined表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法是：

（1）变量被声明了，但没有赋值时，就等于undefined。

（2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。

（3）对象没有赋值的属性，该属性的值为undefined。

（4）函数没有返回值时，默认返回undefined。



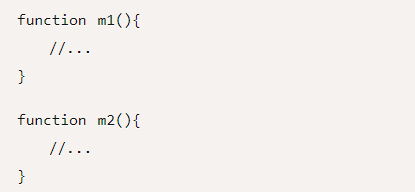
[undefined与null的区别](http://www.ruanyifeng.com/blog/2014/03/undefined-vs-null.html)

模块化的写法

Javascript模块化编程，已经成为一个迫切的需求。理想情况下，开发者只需要实现核心的业务逻辑，其他都可以加载别人已经写好的模块。

一、原始写法

**模块就是实现特定功能的一组方法。只要把不同的函数（以及记录状态的变量）简单地放在一起，就算是一个模块。**

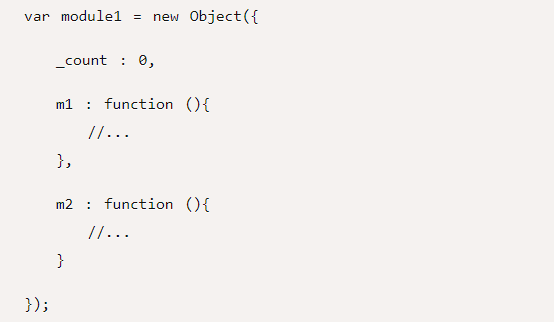


上面的函数m1()和m2()，组成一个模块。使用的时候，直接调用就行了。

这种做法的缺点很明显：**"污染"了全局变量，无法保证不与其他模块发生变量名冲突，而且模块成员之间看不出直接关系。**

二、对象写法

为了解决上面的缺点，可以把模块写成一个对象，所有的模块成员都放到这个对象里面。



上面的函数m1()和m2(），都封装在module1对象里。使用的时候，就是调用这个对象的属性。

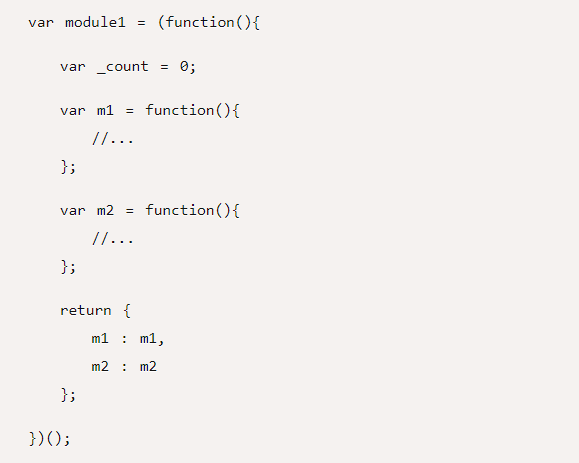


但是，这样的写法会暴露所有模块成员，内部状态可以被外部改写。比如，外部代码可以直接改变内部计数器的值。



三、立即执行函数写法

使用"立即执行函数"（Immediately-Invoked Function Expression，IIFE），可以达到不暴露私有成员的目的。



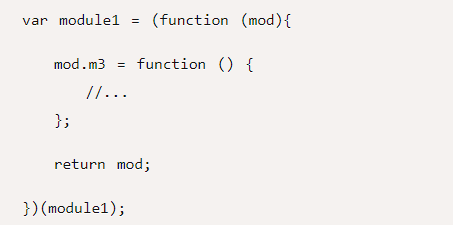
使用上面的写法，外部代码无法读取内部的\_count变量。



module1就是Javascript模块的基本写法。下面，再对这种写法进行加工。

四、放大模式

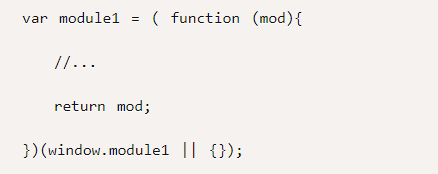
如果一个模块很大，必须分成几个部分，或者一个模块需要继承另一个模块，这时就有必要采用"放大模式"（augmentation）。



上面的代码为module1模块添加了一个新方法m3()，然后返回新的module1模块。

五、宽放大模式（Loose augmentation）

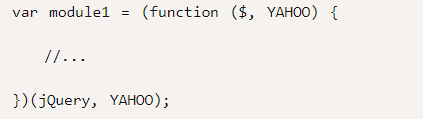
在浏览器环境中，模块的各个部分通常都是从网上获取的，有时无法知道哪个部分会先加载。如果采用上一节的写法，第一个执行的部分有可能加载一个不存在空对象，这时就要采用"宽放大模式"。



与"放大模式"相比，＂宽放大模式＂就是"立即执行函数"的参数可以是空对象。

六、输入全局变量

独立性是模块的重要特点，模块内部最好不与程序的其他部分直接交互。为了在模块内部调用全局变量，必须显式地将其他变量输入模块。



上面的module1模块需要使用jQuery库和YUI库，就把这两个库（其实是两个模块）当作参数输入module1。这样做除了保证模块的独立性，还使得模块之间的依赖关系变得明显。

[Javascript模块化编程（一）：模块的写法](http://www.ruanyifeng.com/blog/2012/10/javascript_module.html)

操作元素属性

设置属性 .setAttribute("属性","值")

var temp2= document.getElementById('test2').setAttribute('title','hello');

获取属性 .getAttribute("属性");

var temp = document.getElementById('test1').getAttribute('hello');

删除属性 .removeAttribute

var temp = document.getElementById('test1').removeAttribute('hello');

<https://www.cnblogs.com/chengxs/p/9000037.html>