# JavaScript

## 基础

我们可以从几个方面去说JavaScript是什么：

* **基于对象**
  + javaScript中**内置了许多对象**供我们使用【String、Date、Array】等等
  + javaScript也**允许我们自己自定义对象**
* **事件驱动**
  + 当用户触发执行某些动作的时候【鼠标单机、鼠标移动】，**javaScript提供了监听这些事件的机制**。当用户触发的时候，就执行我们自己写的代码。
* **解释性语言**
  + javaScript代码是**由浏览器解析的**，并不需要编译。
* **基于浏览器的动态交互技术**
  + 既然javaScript是由浏览器解析的，那么它肯定要基于浏览器。 javaScript让网页变得更加“灵活""
* **弱类型**
  + 像java、c++等编译型语言，要先定义变量，后使用。javaScript**能够直接使用，不需要先定义**

typeof 运算符有一个参数，即要检查的变量或值。alert (typeof sTemp); //输出 "string"

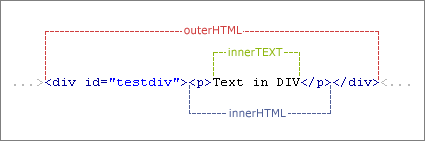
对变量或值调用typeof运算符将返回下列值之一：

* **undefined - 如果变量是 Undefined 类型的**
* **boolean - 如果变量是 Boolean 类型的**
* **number - 如果变量是 Number 类型的**
* **string - 如果变量是 String 类型的**
* **object - 如果变量是一种引用类型或 Null 类型的**

在读模式下，**innerHTML 属性**返回与**调用元素的所有子节点（包括元素、注释和文本节点）对应的 HTML 标记**。在写模式下，innerHTML 会根据指定的值创建新的 DOM 树，然后用这个 DOM 树完全替换调用元素原先的所有子节点.

在读模式下，outerHTML 返回**调用它的元素及所有子节点的 HTML 标签。**在写模式下，outerHTML 会根据指定的 HTML 字符串创建新的 DOM 子树完全替换调用元素。

通过 innerText 属性可以操作元素中包含的所有文本内容，包括子文档树中的文本。在通过 innerText 写入值时，结果会删除元素的所有子节点，插入包含相应文本值的文本节点。



## JS中的数据类型

JavaScript包含两种不同数据类型：**基本数据类型和引用数据类型。基本类型**指的是简单的数据，**引用类型**指由多个值构成的对象。

常见的五种基本数据类型：

1. Boolean
2. Number
3. String
4. Undifined
5. Null

这五种基本数据类型可以直接操作保存在变量中的**实际值。**

这也证明了一点：**基本类型虽然也可以添加属性，也不会报错，经测试添加完之后却是无法访问的。**

**引用类型：对象、数组、函数。**

**not defined**

**一个未定义的变量是没有声明的变量**

**undefined 是声明了变量但未对其初始化时赋予该变量的值**

**null 则用于表示尚未存在的对象（typeof 运算符对于 null 值会返回 "Object"。）。**如果函数或方法要返回的是对象，那么找不到该对象时，返回的通常是 null。

在JavaScript中，**0，"",false,null,undefined,NaN均表示false。**

**var obj = / / 表示正则表达式对象，是Perl风格.**

**js数组中提供了以下几个方法可以让我们很方便实现堆栈：**

**shift:从数组中把第一个元素删除，并返回这个元素的值。  
unshift: 在数组的开头添加一个或更多元素，并返回新的长度  
push:在数组的中末尾添加元素，并返回新的长度  
pop:从数组中把最后一个元素删除，并返回这个元素的值。**

**列举三种强制类型转换和两种隐式类型转换**

**parseInt(),parseFloat(),Number()**

**==,!!**

## 逻辑符号

**逻辑 NOT 运算符返回的一定是 Boolean 值**。

**逻辑 NOT 运算符的行为如下：**

* **如果运算数是对象，返回 false**
* **如果运算数是数字 0，返回 true**
* **如果运算数是 0 以外的任何数字，返回 false**
* **如果运算数是 null，返回 true**
* **如果运算数是 NaN，返回 true**
* **如果运算数是 undefined，发生错误**

**判断JavaScript变量的Boolean 值时，也可以使用逻辑NOT运算符。这样做需要在一行代码中使用两个 NOT 运算符。无论运算数是什么类型，第一个NOT运算符返回 Boolean值，第二个NOT将对该Boolean值取反，从而给出变量真正的Boolean值。**

JavaScript的**逻辑 OR 运算也是简便运算，对于逻辑 OR 运算符来说，如果第一个运算数值为 true，就不再计算第二个运算数**，**如果某个运算数不是 Boolean 值，逻辑 OR 运算并不一定返回 Boolean 值**，逻辑||的运算规则如下：

1. **如果一个运算数是对象，另一个是 Boolean 值，返回该对象。**
2. **如果两个运算数都是对象，返回第一个对象。**
3. **如果某个运算数是 null，返回 null。**
4. **如果某个运算数是 NaN，返回 NaN。**
5. **如果某个运算数是 undefined，发生错误。**

**对于这些运算规则，没有必要死记硬背，因为在JavaScript中，可以使用逻辑Not运算符来判断JavaScript变量的Boolean值，判断的方式就是"!!变量名"**

**JavaScript"=="的作用**

1. **当==两边的内容是字符串时，则比较字符串的内容是否相等。**
2. **当==两边的内容是数字时，则比较数字的大小是否相等。**
3. **当==两边的内容是对象或者是对象的函数属性时，则比较内存地址是否相等**。

**==和===的区别**

**==用于一般比较，===用于严格比较，==在比较的时候可以转换数据类型，===严格比较，只要类型不匹配就返回flase**。

**总结：**

**==和===的区别："==" 只要求值相等; "===" 要求值和类型都相等**

## 数组

　　常规方式声明：

**1、var arrName = new Array();//创建一个数组**

**2、var arrName = new Array([size]);　//创建一个数组并指定长度，注意不是上限，是长度**

**3、var arrName =new Array("孤傲苍狼","白虎神皇","灭世魔尊");//创建一个数组，并初始化数组的内容**

**4、var array = []**

**5.** **Array.of(1,2) //[1,2]**

**注意：虽然var arrName = new Array([size]);指定了长度，但实际上所有情况下数组都是变长的，也就是说即使指定了长度为2，仍然可以将元素存储在规定长度以外的，注意：这时长度会随之改变。\**

**判断是否为数组的方法**

* **console.log(arr instanceof Array)**
* **console.log(arr.construct === Array)**
* **console.log(Array.isArray(arr))**

## [面向(基于)对象编程](https://www.cnblogs.com/xdp-gacl/p/3698862.html)

**定义在方法内【也就是function内部，也可以看作成构造函数】的变量，就是私有变量。**

**js的几种继承方式？**

1.使用对象冒充实现继承

2.采用call、Apply方法改变函数上下文实现继承

call和apply都是改变this指向的方法，区别在于call可以写多个参数，而apply只能写两个参数，第二个参数是一个数组，用于存放要传的参数。

**3.原型链方式继承**

### 创建类

**使用function来模拟创建类，function充当了构造函数. function Teacher() { }**

**用一个变量记住一个匿名的function当做是类，function充当了构造函数. var Teacher = function () {}**

**使用JSON语法来创建类:**

**var obj = {  
 age: 20,  
 str: "zhongfucheng",  
 method:function () {  
 alert("aaa");  
 }}**

### 函数的声明

方式一：常规方式

1 function sum1(num1,num2){

2 return num1+num2

3 }

解析器会率先读取函数声明，并使其置于任何代码之前；而对于函数表达式，则必须等到解析器执行到它所在的代码行，才别执行。

方式二：函数表达式

1 var sum2=function(num1,num2){

2 return num1+num2;

3 };

方式三：动态创建函数(这种方式用得不多)

1 var sum3=new Function("num1","num2","return num1+num2");//动态创建函数

**函数重载**

那么后面定义的就会覆盖前面定义，调用方法时永远是调用后定义的那个。

**JavaScript函数的特殊性**

JavaScript中的函数名本身就是变量，所以函数也可以当作普通变量来使用。

### 创建对象（new）

1、创建一个空对象

var obj=new Object();

2、设置原型链

obj.\_\_proto\_\_= Func.prototype;

3、让Func中的this指向obj，并执行Func的函数体。

var result =Func.call(obj);

4、判断Func的返回值类型：

如果是值类型，返回obj。如果是引用类型，就返回这个引用类型的对象。

### 增加属性，访问属性

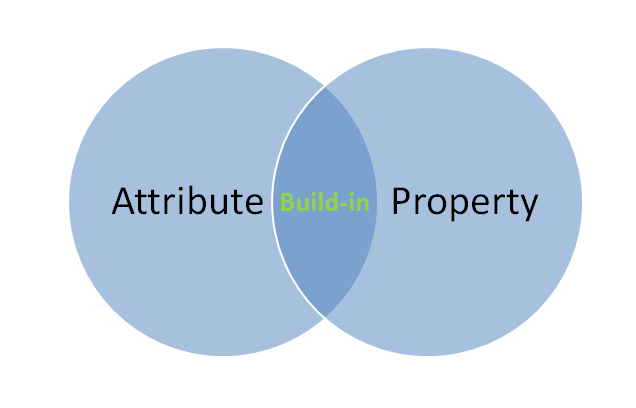
使用.操作符增加属性、访问属性

**var** aa = [obj["age"]];

### attribute与property是一样的吗？

其实不尽然，真实情况是attribute与property引用了相同的数据来源。

1. 于build-in属性，attribute和property共享数据，attribute更改了会对property造成影响，反之亦然，但是两者的自定义属性是独立的数据，即使name一样，也互不影响，看起来是下面这张图，但是IE6、7没有作区分，依然共享自定义属性数据。



2. 并不是所有的attribute与对应的property名字都一致。比如attribute中的class，在property中对应的名称为className。

3. 对于值是true/false的property，类似于input的checked等，attribute取得值是HTML文档里面的值（checked），property是取得计算结果（true/false），property改变并不影响attribute字面量，但attribute改变会影响property计算。

4. 对于一些和路径相关的属性，两者取得值也不尽相同，但是同样attribute取得是字面量，property取得是计算后的完整路径，就像上面例子中src的返回值。

## [JS 触发事件大全](https://www.cnblogs.com/wd1984215/articles/1348145.html)

|  |  |  |
| --- | --- | --- |
| 事件 | 浏览器 |  |
| 一般事件 |  |  |  |
| onclick | IE3、N2 | 鼠标点击时触发此事件 |  |
| ondblclick | IE4、N4 | 鼠标双击时触发此事件 |  |
| onmousedown | IE4、N4 | 按下鼠标时触发此事件 |  |
| onmouseup | IE4、N4 | 鼠标按下后松开鼠标时触发此事件 |  |
| onmouseover | IE3、N2 | 当鼠标移动到某对象范围的上方时触发此事件 |  |
| onmousemove | IE4、N4 | 鼠标移动时触发此事件 |  |
| onmouseout | IE4、N3 | 当鼠标离开某对象范围时触发此事件 |  |
| onkeypress | IE4、N4 | 当键盘上的某个键被按下并且释放时触发此事件. |  |
| onkeydown | IE4、N4 | 当键盘上某个按键被按下时触发此事件 |  |
| onkeyup | IE4、N4 | 当键盘上某个按键被按放开时触发此事件 |  |
| 页面相关事件 |  |  |  |
| onabort | IE4、N3 | 图片在下载时被用户中断 |  |
| onbeforeunload | IE4、N | 当前页面的内容将要被改变时触发此事件 |  |
| onerror | IE4、N3 | 出现错误时触发此事件 |  |
| onload | IE3、N2 | 页面内容完成时触发此事件 |  |
| onmove | IE、N4 | 浏览器的窗口被移动时触发此事件 |  |
| onresize | IE4、N4 | 当浏览器的窗口大小被改变时触发此事件 |  |
| onscroll | IE4、N | 浏览器的滚动条位置发生变化时触发此事件 |  |
| onstop | IE5、N | 浏览器的停止按钮被按下时触发此事件或者正在下载的文件被中断 |  |
| onunload | IE3、N2 | 当前页面将被改变时触发此事件 |  |
| 表单相关事件 |  |  |  |
| onblur | IE3、N2 | 当前元素失去焦点时触发此事件 |  |
| onchange | IE3、N2 | 当前元素失去焦点并且元素的内容发生改变而触发此事件 |  |
| onfocus | IE3 、N2 | 当某个元素获得焦点时触发此事件 |  |
| onreset | IE4 、N3 | 当表单中RESET的属性被激发时触发此事件 |  |
| onsubmit | IE3 、N2 | 一个表单被递交时触发此事件 |  |
| 滚动字幕事件 |  |  |  |
| onbounce | IE4、N | 在Marquee内的内容移动至Marquee显示范围之外时触发此事件 |  |
| onfinish | IE4、N | 当Marquee元素完成需要显示的内容后触发此事件 |  |
| onstart | IE4、 N | 当Marquee元素开始显示内容时触发此事件 |  |
| 编辑事件 |  |  |  |
| onbeforecopy | IE5、N | 当页面当前的被选择内容将要复制到浏览者系统的剪贴板前触发此事件 |  |
| onbeforecut | IE5、 N | 当页面中的一部分或者全部的内容将被移离当前页面[剪贴]并移动到浏览者的系统剪贴板时触发此事件 |  |
| onbeforeeditfocus | IE5、N | 当前元素将要进入编辑状态 |  |
| onbeforepaste | IE5、 N | 内容将要从浏览者的系统剪贴板传送[粘贴]到页面中时触发此事件 |  |
| onbeforeupdate | IE5、 N | 当浏览者粘贴系统剪贴板中的内容时通知目标对象 |  |
| oncontextmenu | IE5、N | 当浏览者按下鼠标右键出现菜单时或者通过键盘的按键触发页面菜单时触发的事件 |  |
| oncopy | IE5、N | 当页面当前的被选择内容被复制后触发此事件 |  |
| oncut | IE5、N | 当页面当前的被选择内容被剪切时触发此事件 |  |
| ondrag | IE5、N | 当某个对象被拖动时触发此事件 [活动事件] |  |
| ondragdrop | IE、N4 | 一个外部对象被鼠标拖进当前窗口或者帧 |  |
| ondragend | IE5、N | 当鼠标拖动结束时触发此事件，即鼠标的按钮被释放了 |  |
| ondragenter | IE5、N | 当对象被鼠标拖动的对象进入其容器范围内时触发此事件 |  |
| ondragleave | IE5、N | 当对象被鼠标拖动的对象离开其容器范围内时触发此事件 |  |
| ondragover | IE5、N | 当某被拖动的对象在另一对象容器范围内拖动时触发此事件 |  |
| ondragstart | IE4、N | 当某对象将被拖动时触发此事件 |  |
| ondrop | IE5、N | 在一个拖动过程中，释放鼠标键时触发此事件 |  |
| onlosecapture | IE5、N | 当元素失去鼠标移动所形成的选择焦点时触发此事件 |  |
| onpaste | IE5、N | 当内容被粘贴时触发此事件 |  |
| onselect | IE4、N | 当文本内容被选择时的事件 |  |
| onselectstart | IE4、N | 当文本内容选择将开始发生时触发的事件 |  |
| 数据绑定 |  |  |  |
| onafterupdate | IE4、N | 当数据完成由数据源到对象的传送时触发此事件 |  |
| oncellchange | IE5、N | 当数据来源发生变化时 |  |
| ondataavailable | IE4、N | 当数据接收完成时触发事件 |  |
| ondatasetchanged | IE4、N | 数据在数据源发生变化时触发的事件 |  |
| ondatasetcomplete | IE4、N | 当来子数据源的全部有效数据读取完毕时触发此事件 |  |
| onerrorupdate | IE4、N | 当使用onBeforeUpdate事件触发取消了数据传送时，代替onAfterUpdate事件 |  |
| onrowenter | IE5、N | 当前数据源的数据发生变化并且有新的有效数据时触发的事件 |  |
| onrowexit | IE5、N | 当前数据源的数据将要发生变化时触发的事件 |  |
| onrowsdelete | IE5、N | 当前数据记录将被删除时触发此事件 |  |
| onrowsinserted | IE5、N | 当前数据源将要插入新数据记录时触发此事件 |  |
| 外部事件 |  |  |  |
| onafterprint | IE5、N | 当文档被打印后触发此事件 |  |
| onbeforeprint | IE5、N | 当文档即将打印时触发此事件 |  |
| onfilterchange | IE4、N | 当某个对象的滤镜效果发生变化时触发的事件 |  |
| onhelp | IE4、N | 当浏览者按下F1或者浏览器的帮助选择时触发此事件 |  |
| onpropertychange | IE5、N | 当对象的属性之一发生变化时触发此事件 |  |
| onreadystatechange | IE4、N | 当对象的初始化属性值发生变化时触发此事件 |  |

## 闭包（Closure）

**一、变量的作用域**

在JavaScript中，变量的作用域分两种：全局变量和局部变量。

在Javascript中，在函数内部可以直接读取全局变量。

在函数外部无法读取函数内的局部变量。

这里有一个地方需要注意，函数内部声明变量的时候，**一定要使用var命令。如果不用的话，实际上是声明了一个全局变量！**

**JavaScript中的函数名本身就是变量，所以函数也可以当作普通变量来使用。也就是说，不仅可以像传递参数一样把一个函数传递给另一个函数，而且可以将一个函数作为另一个函数的返回值返回。**

**闭包的概念**

闭包定义："JavaScript闭包就是在另一个作用域中保存了一份它从上一级函数或者作用域得到的变量，而这些变量是不会随上一级函数的执行完成而销毁".

在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

它的最大用处有两个，**一个是前面提到的可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。**

nAdd=function(){n+=1}

nAdd代表的就是在f1函数内部声明的一个匿名函数，nAdd()就是在调用匿名函数

## 编写类的扩展方法

**1.使用prototype属性扩展**

String.prototype.quote = function(quotestr)

Number.prototype.add=function(n)

Array.prototype.findVal=function(val)

语法格式：

**类名.prototype.方法名 = function([param1],[param2],....[paramn]) {**

**.................**

**}**

**[param1],[param2],....[paramn]这些参数都是可选的**

**使用这种方式给类添加的扩展方法都是动态的，动态方法是针对类的实例对象的，所以调用必须要用"对象.方法名"的形式去调用，不能用"类名.方法名"的形式去调用！**

**2.扩展JavaScript内置类，添加静态方法**

　　JavaScript是可以给类添加静态的扩展方法的，语法格式如下：

**类名.方法名 = function([param1],[param2],....[paramn]) {**

**.................**

**}**

**[param1],[param2],....[paramn]这些参数都是可选的**

**使用这种方式给类添加的扩展方法都是静态的，动态方法是针对类的实例对象的，所以调用必须要用"对象.方法名"的形式去调用，而静态方法是针对类的，用"类名.方法名"的形式去调用！**

**String.Format = function ()**

## 原型链

在JavaScript中,一共有两种类型的值,原始值和对象值.每个对象都有一个内部属性[[prototype]],我们通常称之为原型.原型的值可以是一个对象,也可以是null.如果它的值是一个对象,则这个对象也一定有自己的原型.这样就形成了一条线性的链,我们称之为原型链.

由于JavaScript规定，任何类都继承自Object类，所以"**Person.prototype =** **new Object();**//让Person类继承Object类"即使我们不写，我**猜想**JavaScript引擎也会自动帮我们加上这句话.

我们先看看这样一段代码：

1 var p = new Person("孤傲苍狼",24);//创建一个人，名字是孤傲苍狼

　　很简单的一段代码，我们来看看这个new究竟做了什么？我们可以把new的过程拆分成以下三步：

　　1.var p={}; 初始化一个对象p。

　　2. p.\_\_proto\_\_=Person.prototype;，将对象p的 \_\_proto\_\_ 属性设置为Person.prototype

　　3.Person.call(p,"孤傲苍狼",24);调用构造函数Person来初始化p。

　那么\_\_proto\_\_是什么？在这里简单地说下。每个对象都会在其内部初始化一个属性，就是 \_\_proto\_\_，当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去\_\_proto\_\_里找这个属性，这个 \_\_proto\_\_又会有自己的\_\_proto\_\_，于是就这样一直找下去，也就是我们平时所说的原型链的概念。

其实prototype只是一个假象，他在实现原型链中只是起到了一个辅助作用，换句话说，他只是在new的时候有着一定的价值，而原型链的本质，其实在于\_\_proto\_\_。

实现圆角： 使用 border-radius

实现阴影：使用 box-shadow

call apply bind : 改变this的指向，

## DOM,BOM

javascript由ECMAScript,DOM,BOM三部分组成，

* ECMAScript也是一种语言，也就是对规定的语法，操作，关键字，语句等的一个描述，javascript实现了ECMAScript
* DOM是文档对象模型，包括了获取元素，修改样式，操作元素三方面内容，也是我们进行最多的操作，有很多兼容性写法
* BOM是浏览器对象模型，包括浏览器的一些操作，window.onload,window.open等还有浏览器事件，监听窗口的改变onresize,监听滚动事件onscroll等

## 移动端是怎么做适配的？

1. 使用<mtea name="viewport">  
   <meta>标签包含 name、 content 、 http-equiv 、 charset 、 scheme 五个属性，当 name 的值为 viewport 时可以使页面适应移动端设备，常用的meta viewport标签如下： <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1">  
   其中：  
   width：控制viewport宽度的大小，可以指定的一个值（或特殊值），如500，或device-width //特殊值  
   height：控制viewport宽度的大小，同上  
   initial-scale：控制页面第一次加载时的缩放比例  
   maximum-scale：允许用户缩放的最大比例  
   minimum-scale：允许用户缩放的最小比例  
   user-scalable：用户是否可以手动缩放
2. 媒体查询  
   @media有四种媒体类型： all （适用于所有设备）、 screen （用于屏幕）、 print （用于打印设备）、speech （用于发声设备），默认情况下使用 all 类型

<style>

@media (max-width: 500px) {

body {

color: green;

}

}

</style>

上述代码默认使用 all 类型，当窗口小于500px时，字体颜色会变成原谅色

1. 使用动态 rem  
   rem代表根元素的 font-size 的大小，当设置元素的 font-size 为1rem时，就表示该元素的字体大小为根元素字体大小的1倍，2rem就为根元素字体大小的2倍...以此类推；同时也可以通过修改根元素的字体大小来改变元素字体的大小。  
   使用rem可以成比例地修改元素的字体大小，还可以避免添加的字体样式与继承来的样式相冲突。

## js垃圾回收方式

**内存泄露**指任何对象在不再拥有或不再需要它之后依然存在

* setTimeout第一个参数是字符串而不是函数的时候就会造成内存泄露
* 闭包, **外部变量没释放，所以指向的大函数内的小函数也释放不了.**
* 控制台日志
* 循环（两个对象彼此引用且彼此保留）

**防止内存泄露：**

1、不要动态绑定事件；  
2、不要在动态添加，或者会被动态移除的dom上绑事件，用事件冒泡在父容器监听事件；  
3、如果要违反上面的原则，必须提供destroy方法，保证移除dom后事件也被移除，这点可以参考Backbone的源代码，做的比较好；  
4、单例化，少创建dom，少绑事件。

* 标记清除：这是js最常用的垃圾回收方法，当一个变量进入执行环境时，例如函数中声明一个变量，将其标记为进入环境，当变量离开环境时，（函数执行结束），标记为离开环境
* 引用计数: 跟踪记录每个值被引用的次数，声明一个变量，并将引用 类型赋值给这个变量，则这个值的引用次数+1，当变量的值变成了另一个，则这个值的引用次数-1，当值的引用次数为0的时候，就回收

# jQuery

## 一、初识jQuery

### 什么是jQuery？

**jQuery是一个优秀的JavaScript库**，是一个凭借简洁的语法和跨平台的兼容性，极大地简化了JavaScript开发人员遍历HTML文档，操作DOM，执行动画和开发Ajax的操作。jQuery封装了很多预定义的对象和函数。其理念：write less,do more.

**引入库**：<script src="jquery-3.3.1/jquery-3.3.1.min.js"></script>

### 对象转换

**JQuery对象 -> DOM对象**

两种转换方式将一个JQuery对象转换成DOM对象：[index]和.get(index);

1. jQuery对象是一个数据对象，可以通过[index]的方法，来得到相应的DOM对象。

|  |  |
| --- | --- |
| 1  2 | var v1 = $("#test") ; //jQuery对象  var v2 = $v1[0]; //DOM对象 |

1. 通过.get(index)方法，得到相应的DOM对象。

|  |  |
| --- | --- |
| 1  2 | var v1 = $("#test"); //jQuery对象  var v2 = v1.get(0); //DOM对象 |

**DOM对象 -> JQuery对象**

对于已经是一个DOM对象，只需要用$()把DOM对象包装起来，就可以获得一个jQuery对象了。如$(document.getElementById("test"))

|  |  |
| --- | --- |
| 1  2 | var v1=document.getElementById("test"); //DOM对象  var v2=$(v1); //jQuery对象 |

DOM对象转换为JQuery对象之后就可以使用JQuery的方法了。

**需要再次强调注意的是：DOM对象才能使用DOM中的方法，jQuery对象是不可以用DOM中的方法。**

### 讲解$(function(){})

* $是jQuery别名。如$()也可jQuery()这样写,相当于**页面初始化函数**，当页面加载完毕，会执行jQuery()。
* 希望在做所有事情之前，JQuery操作DOM文档。必须确保在DOM载入完毕后开始执行，应该用ready事件做处理HTML文档的开始
* $(document).ready(function(){});
  + 类似于js的window.onload事件函数，但是ready事件要先于onload事件执行
  + window.onload = function(){};
* 为方便开发，jQuery简化这样的方法，直接用$()表示
* **JQuery的ready事件不等于Js的load：**
  + 执行时机不同：load需要等外部图片和视频等全部加载才执行。ready是**DOM绘制完毕后执行，先与外部文件**
  + 用法不同：load只可写一次，ready可以多次

### jQuery选择器

1）选择器（字符串）

jQuery函数通过该选择器获取对应的DOM，然后将这些DOM封装到一个人jQuery对象中并返回。

所有选择器 \*。$("\*")

标签选择器 标签名。document.getElementsByTagName(tagName)与$("tagname")对比---多个标签，返回数组

ID选择器 #id。document.getElementById(id)与$("#id")

类选择器 .className。$(".className")--多个classname（改变背景图片）

群组选择器 .one,.two 多个选择器使用都好分隔，取并集

复合选择器 .one.two 多个选择器组合使用，取交集

后代选择器 .one .two 两个选择器使用空格隔开，表示可以获取当前元素的子代以及孙子代等等后代元素。

子代选择器 .one>.two 两个选择器使用>隔开，表示只能获取当前选中元素的子代元素。$('div>p'); // 选择所有div标签下的子集标签为p的DOM元素

### jQuery过滤器

jQuery的过滤器必须用在jQuery选择器后，表示对通过前面的jQuery选择器选择到的内容的过滤。是建立在前面选择器已经选择到的元素的基础之上。 语法：selector:过滤器

1.基本过滤器：

selector:first 获取所有已选择到的元素中的第一个元素

$('.info:first').val();

selector:last 获取所有已选择到的元素中的最后一个元素

selector:even 获取所有已选择到的元素中的索引为偶数的元素

selector:odd 获取所有已选择到的元素中的索引为奇数的元素

selector:eq(index) 获取所有已选择到的元素中的索引为index的元素

selector:lt(num) 获取所有已选择到的元素中的索引值小于num的元素

selector:gt(num) 获取所有已选择到的元素中的索引值大于num的元素

selector1:not(selector2) 获取所有已选择到的元素中的除了selector2的元素

selector:header 获取所有已选择到的元素中的标题元素(h1~h6)

2.内容过滤器：

selector:contains(text)

获取所有已选择到的元素中文本包含text的元素

selector:empty

获取所有已选择到的元素中的空元素(没有子节点)

selector:parent

获取所有已选择到的元素中的非空元素(有子节点)，如$("div:parent");

selector1:has(selector2)

获取所有已选择到的元素中包含selector2的元素，如$("div:has('span')");

3.可见性过滤器：

隐藏类型分两种：

1）不占据屏幕空间

display:none;

<input type="hidden">

2）占据屏幕空间

visibility:hidden;

opacity:0;//透明度为0

使用：

:visible 选择所有占据屏幕空间的元素

:hidden 选择所有不占据屏幕空间的元素

4.属性过滤器：

selector[attrKey]

获取所有已选择到的元素中具有属性attrKey的元素

selector[attrKey=attrVal]

获取所有已选择到的元素中具有属性attrKey，并且属性值为attrVal的元素

selector[attrKey^=attrVal]

获取所有已选择到的元素中具有属性attrKey，并且属性值为以attrVal开头的元素

selector[attrKey$=attrVal]

获取所有已选择到的元素中具有属性attrKey，并且属性值为以attrVal结尾的元素

selector[attrKey\*=attrVal]

获取所有已选择到的元素中具有属性attrKey，并且属性值为包含attrVal的元素

selector[attrKey!=attrVal]

获取所有已选择到的元素中具有属性attrKey，并且属性值不为以attrVal的元素或者没有属性attrVal的元素

5.后代选择器：

selector:nth-child(index)

获取每个selector元素中索引为index的子元素。【注意】index从1开始

selector:first-child

获取每一个selector元素中的第一个子元素（每个父元素的第一个子元素）

selector:last-child

获取每一个selector元素中的最后一个子元素（每个父元素的最后一个子元素）

selector:only-child

获取每一个selector元素中的独生子子元素（每个父元素如果只有一个孩子元素，获取该元素）

selector:first-of-type

获取每个selector元素中每种类型子元素中的第一个

selector:last-of-type

获取每个selector元素中每种类型子元素中的最后一个

6.表单过滤器：

:checked 选取所有被选中的元素，用于复选框、单选框、下拉框

:selected 选取所有被选中的元素，该选择器只适用于<option>

:focus 选取当前获取焦点的元素

:text 选取所有的单行文本框(<input type="text">)

:password 选取所有的密码框

:input 选取所有的<input>,<textarea>,<select>,<button>元素。

\*注意，$(":input")是选中可以让用户输入的标签元素；而$("input")是选择名字为input的标签元素。\*

:enable 选取所有可用元素，该选择器仅可用于支持disable属性的html元素。(<button>,<input>,<optgruop>,<option>,<select>,<textarea>)

:disable 选取所有不可用元素，该选择器也仅可用于支持disable属性的html元素。

:radio 选取所有的单选框

:checkbox 选取所有的多选框

:submit 选取所有的提交按钮

:image 选取所有的input类型为image的表单元素

:reset 选取所有的input类型为reset的表单元素

:button 选取所有的input类型为button的表单元素

:file 选取所有的input类型为file的表单元素

### jQuery中的Dom操作

DOM是一种与浏览器、平台|语言无关的接口，使用该接口可以轻松的访问 页面中的所有的标准组件。它将**HTML中的一个个标签**封装成DOM元素。

1.查找节点

通过jQuery选择器来完成

2.创建节点

$(html)：根据传递的标记字符串，创建DOM对象。

3.插入节点

1) $A.append(B)

将B追加到A的末尾处，作为它的最后一个子元素

2) $A.appendTo(B)

将A追加到B的末尾，作为它的最后一个子元素

3) prepend()

$A.prependTo(B)

将A追加到B的前面，作为它的第一个子元素

$A.after(B)

在A之后追加B，作为它的兄弟元素

$A.insertAfter(B)

在B之后追加A，作为它的兄弟元素

$A.before(B)

在A之前追加B，作为它的兄弟元素

$A.insertBefore(B)

在B之前追加A，作为它的兄弟元素

var div = document.createElement("div");

var txt = document.createTextNode("DOM");

div.appendChild(txt);

document.body.appendChild(div);

div.setAttribute("title","盒子");

var $div = = $("<div title='盒子'>DOM</div>");

$(body).append($div);

4.删除节点

remove([selector])

从DOM中删除所有匹配的元素，返回值是一个指向已经被删除的节点的引用，可以在以后再使用这些元素。

该方法会移除元素，同时也会移除元素内部的一切，包括绑定的事件及与该元素相关的jQuery数据。

detach([selector])

与remove()类似，但是detach()保存所有jQuery数据和被移走的元素的相关联事件。

empty()

无参数。从DOM中清空集合中匹配元素的所有的子节点。

5.复制节点

$("#id").clone(false);

该方法返回的是一个节点的引用，参数默认为false，为浅复制；

参数是true,为深复制，含义是：复制元素的同时复制元素中所绑定的事件。

6.替换节点

replaceWith(newContent);

用新内容替换集合中所有匹配的元素，并且返回被删除的元素的集合。

该方法会删除与节点相关联的所有数据和事件处理程序。

replaceAll(target);

用集合的匹配元素替换每个目标元素。颠倒了replaceWith()操作效果。

7.包裹节点

wrap([wrappingElement])

在每个匹配的元素外层包上一个html元素

warpAll([wrappingElement])

将所有匹配的元素用一个元素来包裹，在所有匹配元素外面包裹一层HTML结构

warpInner([wrappingElement])

每个匹配元素里面内容(子元素)都会被这种结构包裹

8.节点遍历

children([selector])

用于取得匹配元素的子元素集合(只考虑子元素而不考虑任何后代元素)

$('.content.inner')=>$('.content').children('.inner');

find(selector)

在当前对象元素中的子元素查找，和参数所匹配的所有的后代元素

next([selector])

取得匹配的元素集合中每一个元素紧邻的后面兄弟元素

nextAll([selector])

查找当前元素之后所有的同辈元素

prev([selector])

取得匹配元素前面紧邻的兄弟元素

prevAll([selector])

取得当前元素之前所有的同辈元素

silibinng([selector])

取得匹配元素的前后所有的兄弟元素

closest(selector)

取得和参数匹配的最近的元素，如果匹配不上继续向上查找父元素

filter(selector)

把当前所选择的所有元素再进行筛选过滤

parent([selector])

取得匹配元素的集合中，每个元素的父元素

parents([selector])

获得集合中每个匹配元素的祖先元素

## 二、jQuery的事件和API

### 事件

on()

在选定的元素上绑定一个或多个事件处理函数。

off()

移除一个事件处理函数。

trigger()

根据绑定到匹配元素的给定的事件类型执行所有的处理程序和行为。

click() 单击

dblclick() 双击

hover() 悬停

mousedown() 按下

mouseup() 抬起

mouseenter() 移入 不支持子元素

mouseleave() 离开 不支持子元素

mouseout() 离开 支持子元素

mouseover() 进入 支持子元素

mousemove() 移动

keypress() 按键按下

keyup() 按键抬起

keydown() 按键按下

focus() 聚焦事件

blur() 失去焦点事件

**change() 当元素的值发生改变时激发的事件**

select() 当textarea或文本类型的input元素中的文本被选择时触发的事件

submit() 表单提交事件，绑定在form上

### **事件冒泡**

* 什么是冒泡
  + 在页面上可以有多个事件，也可以多个元素影响同一个元素
  + 从里到外
  + 嵌套关系
  + 相同事件
  + 其中的某一父类没有相同事件时,继续向上查找
* **停止事件冒泡**
  + 停止事件冒泡可以阻止事件中其他对象的事件处理函数被执行
  + 在jQuery中提供了**stopPropagation()**方法
* **阻止默认行为**
  + 网页中元素有自己的默认行为，例如：单击超链接后会跳转、单击提交后表单会提交，有时需要阻止元素的默认行为
  + 在jQuery中提供了preventDefault()方法来阻止元素的默认行为

 **事件捕获**

* 事件捕获和冒泡是相反的过程，事件捕获是从最顶端往下开始触发
* 并非所有的浏览器都支持事件捕获，并且这个缺陷无法通过JavaScript来修复。jQuery不支持事件捕获，如需要用事件捕获，要用原生的JavaScript

### 事件委托

 事件委托是利用事件冒泡，只指定一个事件处理程序来管理某一类型的所有事件。

(1)、bind

定义和用法：主要用于给选择到的元素上绑定特定事件类型的监听函数；

(2)、live

定义和用法：主要用于给选择到的元素上绑定特定事件类型的监听函数；

(3)、delegate

定义和用法：将监听事件绑定在就近的父级元素上

(4)、on

定义和用法：将监听事件绑定到指定元素上。

$("#lists").on("click","li",function(event)

### jQuery中常用的API

1.jQuery中的html(),text(),val()方法

html()：此方法类似JavaScript中innerHTML属性（这两者并不等价），可以用来读取和设置某个元素中的HTML内容。

无参：获取html的值，$("#test").html();jQuery对象是包装DOM对象后产生的

有参数html：设置html的值

text()

无参：获取文本值

有参数text：设置文本值

val()

无参：获取value的值

有参数value：设置value的值

2.jQuery中的工具方法

1) get(); //**以数组的形式返回DOM节点。**

console.log($('div').get());

2) toArray(); //返回一个包含jQuery对象结合中的所有DOM元素数组。

console.log($('div').toArray());

3) eq(index); //**返回index位置上的jQuery对象。**

console.log($('div').eq(1).get(0));

4) filter(function(index,item){}); //过滤器函数，返回jQuery对象。

var $result = $('div').filter(function(index,item){

return index == 2;

});

5) map(function(index,item){}); //用于获取或设置元素集合中的值。

var $result = $('div').map(function(index,item){

return $(item).html()

});

6) each(function(index,item){}); //**遍历一个jQuery对象**。

$('div').each(function(index,item){

console.log(index,'--',item);

});

7) slice(0,3); //截取

var $result = $('div').slice(0,3);

console.log($result.get());

});

8) not()

9) first()

10) last()

11) is()

12) has()

3.jQuery中属性设置函数

//获取属性值

attr(key)

prop(key)

//删除属性

removeAttr(attributeName)

removeProp(propertyName)

//批量设置属性

css(key)

//添加类

addClass(className)

//判断有没有指定的类，有，返回true，否则返回false

hasClass(className)

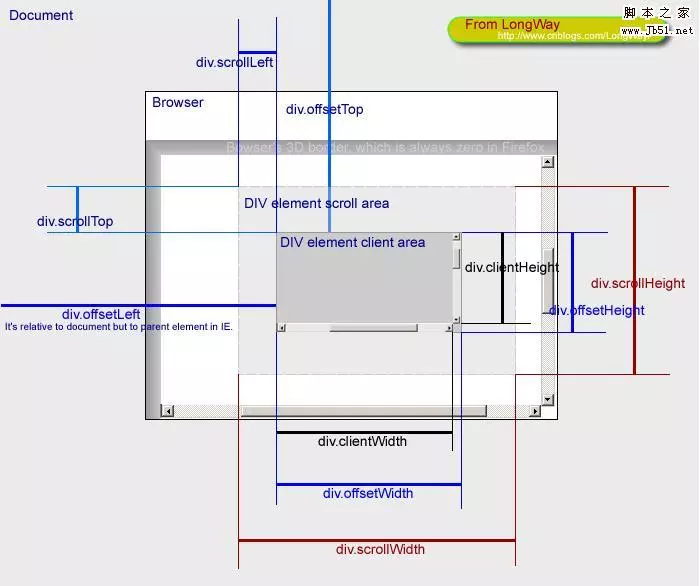
//删除类

removeClass(className)

## 三、jQuery中的动画

一、jQuery样式相关方法

宽度 = width + 2padding + 2border + 2margin



//获取视口区的宽高：

1、$(window).height()

2、$(window).width()

//获取内容区的宽高：

$('div').width();

$('div').height();

//获取内容+padding区的宽高：

3、$('div').innerHeight()

4、$('div').innerWidth()

//获取内容+padding+border区的宽高：

5、$('div').outerHeight()

6、$('div').outerWidth()

//获取内容+padding+border+margin区的宽高：

$('div').outerWidth(true);

$('div').outerHeight(true);

//获取相对于文档的偏移

7、.offset()

//获取相对于定位父元素的偏移

8、.position()

//横向滚动条左侧的偏移

9、.scrollLeft()

//横向滚动条上侧的偏移

10、.scrollTop()

//获取离它最近的父定位元素

11、.offsetParent()

二、效果

1.基本效果

1）隐藏 hide()，hide()通过改变元素的高度宽度和不透明度，直到这三个属性值到0。

2）显示 show()，

show(speed,callback)

* speed: 字符串或数字，表示动画将运行多久（slow=0.6/normal=0.4/fast=0.2）
* callback: 动画完成时执行的方法。

3）隐藏与显示 toggle()，toggle(boolean)。true为显示 false为隐藏

2.淡入淡出效果

1）淡入 fadeIn()

2）淡出 fadeOut()

3）淡入到 fadeTo()

4）淡入与淡出 fadeToggle()

3.滑动效果

1）滑下 slideDown()

2）滑上 slideUp()

3）滑上与滑下 slideToggle()

4.自定义效果

animate()

## 四、Jquery对ajax常用的API

jquery对Ajax操作进行了封装，在jquery中的$.ajax()方法属于最底层的方法，第2层是load()、$.get()、$.post();第3层是$.getScript()、$.getJSON()，第2层使用频率很高

1.load()方法是使用Jquery的对象来进行调用的，得到服务器的结果自动会把结果嵌套到所在的标签中。

load(url,[data],[callback])　　url:加载的页面地址,[data]传送的数据，[callback]加载完成时回调函数。

另外‘：url：参数还可以用于筛选要加载页面中的某部分内容。

url selector 注意：url和选择器之间有一个空格

$("div1").load("load.html #div1");

load()方法的传递方式根据参数data来自动指定，如果没有参数传递，则采用GET方式传递，反之，采用POST

2.get()方法不是使用Jquery对象来调用，因此需要手动把结果放在想要放的位置。

语法：$.get(url,[data],[callback],[type])　　url:等待加载的数据地址,[data]发送到服务器的数据,key/value形式,[callback]加载成功时执行的回调函数,[type]表示返回数据的个数。

<?xml version="1.0" encoding="utf-8" ?>

<Info>

<User id="1">

<name>刘德华</name>

<sex>男</sex>

<email>liudehua@qq.com</email>

</User>

<User id="2">

<name>张学友</name>

<sex>男</sex>

<email>zhangxueyou@qq.com</email>

</User>

</Info>

$(function () {

$.get("UserInfo.xml", function (data) {

$("#div1").empty();

var strHtml = "";

$(data).find("User").each(function () {

var $strUser = $(this);

strHtml += "姓名:" + $strUser.find("name").text() + "<br/>";

strHtml += "性别:" + $strUser.find("sex").text() + "<br/>";

strHtml += "邮箱:" + $strUser.find("email").text() + "<hr/>";

})

$("#div1").html(strHtml);

})

})

　HTML：

<div id="div1">

　　</div>

post()方法是用来把参数带过去给服务器的，因此我们需要在Servlet上手动设置编码。用法与get()方法一样

* **serialize()是非常好用的一个方法，不需要我们手动去拼接参数，会自动把form表单的参数封装成JSON格式的数据。**
* 至于$.ajax()方法，实际上就是集合了get()和post()方法。

# Ajax

## 基础

Ajax(Asynchronous JavaScript and XML) 异步JavaScript和XML。

他是指一种**创建交互式网页应用的网页开发技术**。

**客户端与服务器，可以在【不必刷新整个浏览器】的情况下，与服务器进行异步通讯的技术**

### 同步和异步

同步的意思就是客户端提交表单，发起请求后需要一直等待服务器端的响应，收到服务器端的响应后需要重新载入页面。

而有了ajax，则可进行异步交互，比如不会提交整个表单后等待服务器响应再刷新页面，而是填写的过程中，通过ajax的机制，就可以把已经填写的部分提交给服务器，服务器进行处理，而这个过程中可以继续填写后面的内容，服务器发送回响应只会刷新页面的部分内容（利用JavaScript操作DOM进行局部刷新），而不是重新载入整个页面。

其实就是为客户机（网页中常为JavaScript）代码提供了一种发送HTTP请求的方式。通常提交请求都是以表单的形式发送，获取响应要刷新整页，而ajax则是按需发送，只刷新返回的数据。

### 创建一个Ajax请求

**Ajax的核心是XMLHttpRequest对象，**它是Ajax实现的关键，发送异步请求、接受响应以及执行回调都是通过它来完成。

1.传统的web前端与后端的交互中，浏览器直接访问Tomcat的Servlet来获取数据。**Servlet通过转发把数据发送给浏览器**。

2.当我们使用AJAX之后，**浏览器是先把请求发送到XMLHttpRequest异步对象之中，异步对象对请求进行封装，然后再与发送给服务器。服务器并不是以转发的方式响应，而是以流的方式把数据返回给浏览器。**

XMLHttpRequest异步对象会**不停监听服务器状态的变化，得到服务器返回的数据，就写到浏览器上【因为不是转发的方式，所以是无刷新就能够获取服务器端的数据】**

<script type="text/javascript">  
    var httpRequest;  
    if(window.XMLHttpRequest) {  
        //在IE6以上的版本以及其他内核的浏览器(Mozilla)等  
        httpRequest = new XMLHttpRequest();  
    }else if(window.ActiveXObject) {  
        //在IE6以下的版本  
        httpRequest = new ActiveXObject("Microsoft.XMLHTTP");  
    }  
</script>

### 常用的方法

* **open()：该方法创建http请求**
  + **第一个参数是指定提交方式(post、get)**
  + **第二个参数是指定要提交的地址是哪**
  + **第三个参数是指定是异步还是同步(true表示异步，false表示同步)**
  + 第四和第五参数在http认证的时候会用到。是可选的
* **setRequestHeader(String header,String value)：设置消息头（使用post方式才会使用到，get方法并不需要调用该方法）**
  + **xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");**
* **send(content)：发送请求给服务器**
  + **如果是get方式，并不需要填写参数，或填写null**
  + **如果是post方式，把要提交的参数写上去**

### 属性

* **onreadystatechange：请求状态改变的事件触发器（readyState变化时会调用此方法），一般用于指定回调函数**
* **readyState：请求状态readyState一改变，回调函数被调用，它有5个状态**

0：未初始化状态：此时，已经创建了一个XMLHttpRequest对象

1： 准备发送状态：此时，已经调用了XMLHttpRequest对象的open方法，并且XMLHttpRequest对象已经准备好将一个请求发送到服务器端

2：已经发送状态：此时，已经通过send方法把一个请求发送到服务器端，但是还没有收到一个响应

3：正在接收状态：此时，已经接收到HTTP响应头部信息，但是消息体部分还没有完全接收到

4：完成响应状态：此时，已经完成了HTTP响应的接收

## Ajax的实现流程是怎样的？

* (1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.
* (2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.
* (3)设置响应HTTP请求状态变化的函数.
* (4)发送HTTP请求.
* (5)获取异步调用返回的数据.
* (6)使用JavaScript和DOM实现局部刷新.

**Ajax请求总共有八种Callback。**

**同源策略是一种安全协议，指一段脚本只能读取来自同一来源的窗口和文档的属性**。

## 如何解决跨域问题?

**理解跨域的概念：协议、域名、端口都相同才同域，否则都是跨域**

出于安全考虑，服务器不允许ajax跨域获取数据，但是可以跨域获取文件内容。

Jsonp的原理：

1. jsonp通过script标签的src可以跨域请求的特性，加载资源
2. 将加载的资源（通过一个方法名将数据进行包裹）当做是js脚本解析
3. 定义一个回调函数，获取传入的数据

* 所以基于这一点，可以动态创建script标签，**使用标签的src属性访问js文件的形式获取js脚本**，并且这个js脚本中的内容是函数调用，**该函数调用的参数是服务器返回的数据**，为了获取这里的参数数据，需要事先在页面中定义回调函数，在回调函数中处理服务器返回的数据，**【JSONP】**

**(动态添加<script>标签来调用服务器 提供的js脚本)**

缺点：只支持GET方式的jsonp实现，是一种脚本注入行为存在一定的安全隐患。如果返回的数据格式有问题或者返回失败了，并不会报错。

* 在后端上配置可跨域**【CORS方式】**
* 前端ajax请求的是本地接口，本地接口接收到请求后向实际的接口请求数据，然后再将信息返回给前端**【代理方式】**
* **如果使用的是Spring框架的话，那就只需要一个注解就能够解决跨域的问题了**：@CrossOrigin

## Ajax 解决浏览器缓存问题？

* 1、在ajax发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。
* 2、在ajax发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。
* 3、在URL后面加上一个随机数： "fresh=" + Math.random();。
* 4、在URL后面加上时间戳："nowtime=" + new Date().getTime();。
* 5、如果是使用jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。这样页面的所有ajax都会执行这条语句就是不需要保存缓存记录。