# jQuery样式

## 1.1 jQuery对象

jQuery对象是将JavaScript的DOM对象进行包装处理得到的一个类数组对象，他可以帮助我们完美地解决浏览器的兼容问题

### 1.1.1jQuery对象转化为DOM对象

jQuery是一个类数组对象，而DOM对象就是一个单独的DOM元素，前者是对后者进行了包装处理的对象，那么如何将jQuery对象转化为DOM对象呢？

1. **通过数组下标的方式**

<div>元素一</div>

<div>元素二</div>

<div>元素三</div>

var $div = $('div') //jQuery对象

var div = $div[0] //转化成DOM对象

div.style.color = 'red' //操作dom对象的属性

获取的jQuery对象是一个数组，通过数组操作将其取出，即为DOM对象

1. **通过jQuery对象自带的get方法**

jQuery对象自身提供一个.get() 方法允许我们直接访问jQuery对象中相关的DOM节点

var $div = $('div') //jQuery对象

var div = $div.get(0) //通过get方法，转化成DOM对象

div.style.color = 'red' //操作dom对象的属性

该方法其实和第一种基本一样

### 1.2DOM对象转化为jQuery对象

如果传递给**$(DOM)**函数的参数是一个DOM对象，jQuery方法会把这个DOM对象给包装成一个新的jQuery对象

<div>元素一</div>

<div>元素二</div>

<div>元素三</div>

var div = document.getElementsByTagName('div'); //dom对象

var $div = $(div); //jQuery对象

var $first = $div.first(); //找到第一个div元素

$first.css('color', 'red'); //给第一个元素设置颜色

## jQuery选择器

### 1.2.1基本选择器

jQuery提供了一系列的选择器帮助开发者进行节点查找，让开发者可以更少的处理复杂选择过程与性能优化，更多专注业务逻辑的编写。同时，还能有效兼容不同的浏览器，如：

1.IE会将注释节点实现为元素，所以在IE中调用getElementsByTagName里面会包含注释节点，这个通常是不应该的

2.getElementById的参数在IE8及较低的版本不区分大小写

3.IE7及较低的版本中，表单元素中，如果表单A的name属性名用了另一个元素B的ID名并且A在B之前，那么getElementById会选中A

4.IE8及较低的版本，浏览器不支持getElementsByClassName

但是有了jQuery，就能很好地兼容不同的浏览器

#### 1.2.1.1id选择器

**语法：**

**$( "#id" )**

该选择器相当于JavaScript原生代码中的document. getElementById()方法，但是写起来更简洁

eg：<div id="imooc">

<p>id="imooc"</p>

<p>jQuery选中</p>

</div>

**$("#imooc").css("border", "3px solid red");**

#### 1.2.1.2类选择器

类选择器，相对id选择器来说，效率相对会低一点，但是优势就是可以多选

**语法：**

**$( ".class" )**

**注意：jQuery的类选择器内部带了一个循环操作，因此对该选择器所得的jQuery对象进行过操作实际上是对所有使用了该类选择器的标签的DOM对象进行操作**

**eg：**

<div class="imooc">

<p>class="imooc"</p>

<p>jQuery选中</p>

</div>

<div class="imooc">

<p>class="imooc"</p>

<p>jQuery选中</p>

</div>

**var $divs=$(".imooc");**

**$divs.css("border", "3px solid red");**

#### 1.2.1.3元素选择器

**语法：**

**$( "element" )**

**注意：该选择器同类选择器，也具有循环操作**

**eg：**

<div class="imooc">

<p>class="imooc"</p>

<p>jQuery选中</p>

</div>

<div class="imooc">

<p>class="imooc"</p>

<p>jQuery选中</p>

</div>

**var $p=$("p");**

**$p.css("border", "3px solid red");**

#### 1.2.1.4全选择器

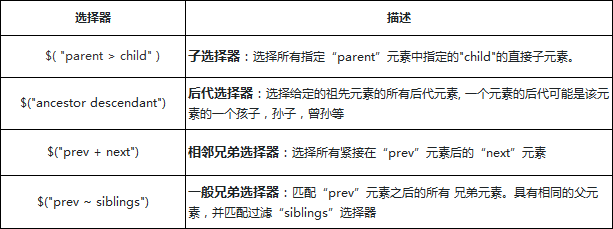
**语法：**

**$( "\*" )**

该选择器将选择所有的元素

### 1.2.3层级选择器

每个html文档都有一个文档树，因此就有了树节点，节点和节点之间就存在父子，兄弟，祖孙的关系，因此就有了层级选择器来处理这种关系



eg：

<div class="aaron">

<p>div下的第一个p元素</p>

</div>

<div class="imooc">

<article>

<p>div下的article下的p元素</p>

</article>

</div>

**$('div > p').css("border", "1px groove red");//设置所有div的所有直接子节点p**

**$('div p').css("border", "1px groove red");//设置所有div的所有后代子节点p**

<div class="bottom">

<div>兄弟节点div, +~选择器不能向前选择</div>

<span class="prev">选择器span元素</span>

<div>span后第一个兄弟节点div</div>

<div>兄弟节点div

<div class="small">子元素div</div>

</div>

<span>兄弟节点span,不可选</span>

<div>兄弟节点div</div>

</div>

**$(".prev + div").css("border", "3px groove blue");//设置+符号前面节点(prev)的下一个div兄弟节点**

**$(".prev ~ div").css("border", "3px groove red");//设置~符号前面节点(prev)的下面所有div兄弟节点**

**1.层级选择器都有一个参考节点**

**2.后代选择器包含子选择器的选择的内容**

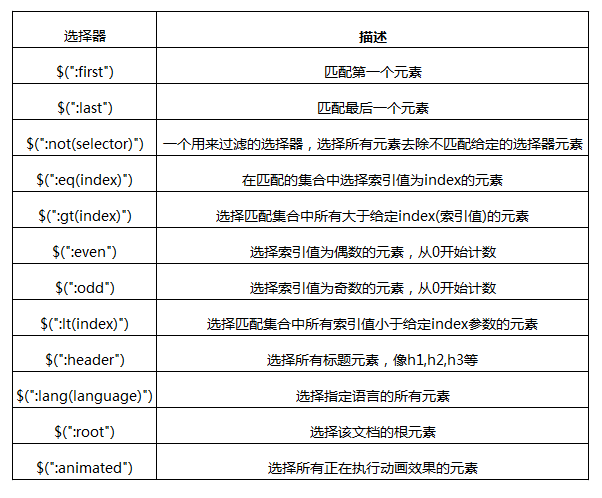
**3.一般兄弟选择器包含相邻兄弟选择的内容**

**4.相邻兄弟选择器和一般兄弟选择器所选择到的元素，必须在同一个父元素下**

**注意：$('img.feedbackImg')*中间没空格*是选择所有class中有feedbackImg的img**

### 1.2.3基本筛选选择器

基本筛选选择器与css中的伪元素类似，选择器用冒号“：”开头。



$(“:nth-child()”) 选择器匹配属于其父元素的第 N 个子元素，不论元素的类型

eg

p:nth-child(3n+0)

//指定了下标(下标从1开始，n从1开始)是 3 的倍数的所有 p 元素

**注意：**

**1. :eq(), :lt(), :gt(), :even, :odd 用来筛选他们前面的匹配表达式的集合元素，根据之前匹配的元素在进一步筛选。注意jQuery合集都是从0开始索引**

**2.** **gt是一个段落筛选，从指定索引的下一个开始，gt(1) 实际从2开始**

**eg：**

<div class="left">

<div class="div">

<p>div:first</p>

<p>:even</p>

</div>

<div class="div">

<p>:odd</p>

</div>

<div class="div">

<p>:even</p>

</div>

<div class="div">

<p>:odd</p>

</div>

<div class="div">

<p>:even</p>

</div>

<div class="div">

<p>div:last</p>

<p>:odd</p>

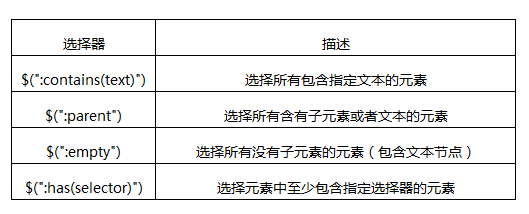
</div>

</div>

**$(".div:first").css("color", "#CD00CD");//找到第一个div**

**$(".div:even").css("border", "3px groove red"); //:even 选择所引值为偶数的元素，从 0 开始计数**

### 1.2.4内容筛选选择器



**注意：contains里的text可以只给一部分**

eg：

<h3>:contains/:has</h3>

<div class="left">

<div class="div">

<p>:contains</p>

</div>

<div class="div">

<p>:contains</p>

</div>

<div class="div">

<p>

<span>:has</span>

</p>

</div>

<div class="div">

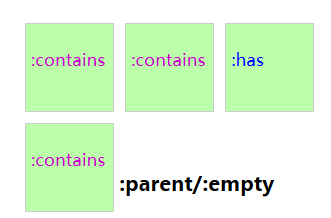
<p>:contains</p>

</div>

</div>

**$(".div:contains(':contains')").css("color", "#CD00CD");//查找文本中包含”:contains”的节点**

**$(".div:has('span')").css("color", "blue");//查找div子孙节点中包含<span>节点的div**



<h3>:parent/:empty</h3>

<div class="left">

<div class="aaron">

<a>:parent</a>

</div>

<div class="aaron">

<a>:parent</a>

</div>

<div class="aaron">

<a>:parent</a>

</div>

<div class="aaron">

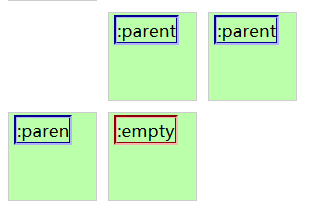
<a></a>

</div>

</div>

**$("a:empty").text(":empty").css("border", "3px groove red"); //选择所有内容为空的a标签**

**$("a:parent").css("border", "3px groove blue");//选择所有有内容的a标签**



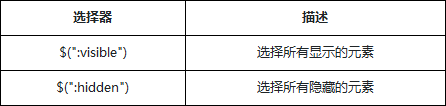
**1. :contains与:has都有查找的意思，但是contains查找包含“指定文本”的元素，has查找包含“指定元素”的元素**

**2. 如果:contains匹配的文本包含在元素的子元素中，同样认为是符合条件的。**

**3. :parent与:empty是相反的，两者所涉及的子元素，包括文本节点**

### 1.2.5可见性筛选选择器

元素有显示状态与隐藏状态，jQuery根据元素的状态扩展了可见性筛选选择器:visible与:hidden



我们有几种方式可以隐藏一个元素：

**1.CSS display的值是none。 (属于hidden)**

**2.type="hidden"的表单元素。**

**3.宽度和高度都显式设置为0。 (属于hidden)**

**4.一个祖先元素是隐藏的，该元素是不会在页面上显示**

**5.CSS visibility的值是hidden (属于visible)**

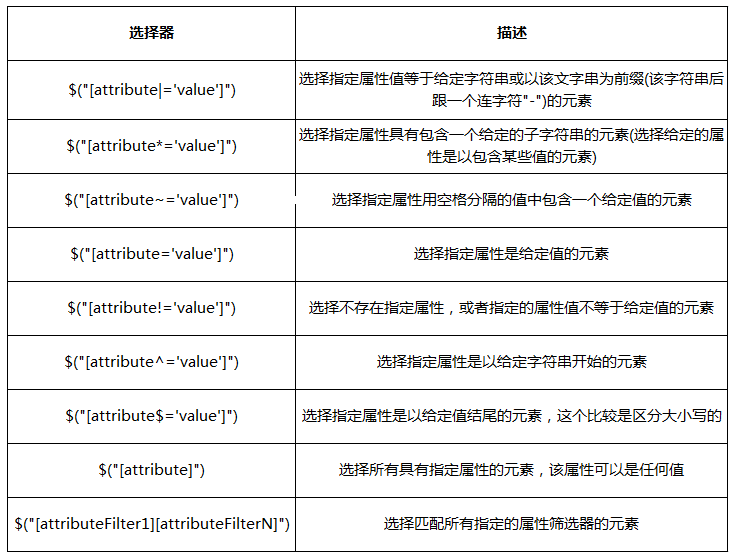
**6.CSS opacity的值是0 (属于visible)**

**如果元素中占据文档中一定的空间,元素被认为是可见的。**

**可见元素的宽度或高度，是大于零。(因此高宽都为0的元素是hidden)**

**元素的visibility: hidden 或 opacity: 0被认为是可见的，因为他们仍然占用空间布局。**

### 1.2.6属性筛选选择器



[attr="value"]和[attr\*="value"]是最实用的

eg：

<h2>属性筛选选择器</h2>

<div class="left" testattr="true" >

<div class="div" testattr="true" name='p1'>

<a>[att=val]</a>

</div>

<div class="div" testattr="true" p2>

<a>[att]</a>

</div>

<div class="div" testattr="true" name="-">

<a>[att|=val]</a>

</div>

<div class="div" testattr="true" name="a b">

<a>[att~=val]</a>

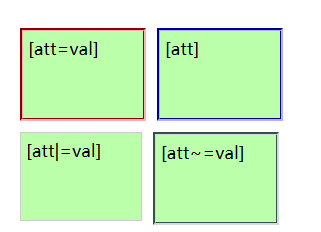
</div>

</div>

**$("div[name='p1']").css("border", "3px groove red");//查找所有div中，属性name=p1的div元素，这是值，下面那个是属性名**

**$("div[p2]").css("border", "3px groove blue");//查找所有div中，有属性p2的div元素**

**$("div[name~='b']").css("border", "3px groove #668B8B"); //查找所有div中，有属性name中的值包含一个连字符“空格”和“b”的div元素**



<div class="div" testattr="true" name='imooc-aaorn'>

<a>[att^=val]</a>

</div>

<div class="div" testattr="true" name='aaorn-imooc'>

<a>[att$=val]</a>

</div>

<div class="div" testattr="true" name="attr-test-selector">

<a>[att\*=val]</a>

</div>

<div class="div" name="a b">

<a>[att!=val]</a>

</div>

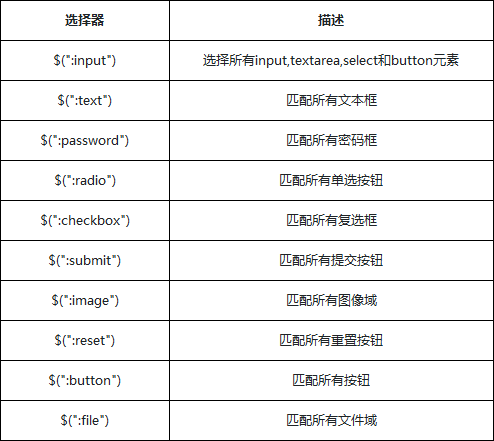
**$('div[name^=imooc]').css("border", "3px groove red");//查找所有div中，属性name的值是用imooc开头的**

**$("div[name$=imooc]").css("border", "3px groove blue");//查找所有div中，属性name的值是用imooc结尾的**

**$("div[name\*=test]").css("border", "3px groove #00FF00"); //查找所有div中，有属性name中的值包含一个test字符串的div元素**

**$("div[testattr!=true]").css("border", "3px groove #668B8B"); //查找所有div中，有属性testattr中的值没有包含"true"的div**

### 1.2.7表单元素选择器



eg:

<form>

<input type="text" value="text类型"/>

<input type="password" value="password"/>

<input type="radio"/>

<input type="checkbox"/>

</form>

**$(":input").css("border", "1px groove red");**

**$("input:checkbox").attr('checked','true');**

**$("input:text").css("background", "#A2CD5A");**

**$("input:password").css("background", "yellow");**

**$("input:radio").attr('checked','true');**

### 1.2.8表单对象属性筛选选择器

****

1.选择器适用于复选框和单选框，对于下拉框元素, 使用 :selected 选择器

2.在某些浏览器中，选择器:checked可能会错误选取到<option>元素，所以保险起见换用选择器input:checked，确保只会选取<input>元素

**eg:**

<form>

<input type="text" value="未设置disabled" />

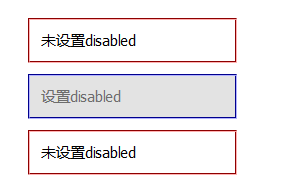
<input type="text" value="设置disabled" disabled="disabled" />

<input type="text" value="未设置disabled" />

</form>

**$("input:enabled").css("border", "2px groove red"); //查找所有input所有可用的（未被禁用的元素）input元素。**

**$("input:disabled").css("border", "2px groove blue"); //查找所有input所有不可用的（被禁用的元素）input元素。**



<form>

<input type="checkbox" checked="checked">

<input type="checkbox">

<input type="radio" checked>

<input type="radio">

<select name="garden" multiple="multiple">

<option>imooc</option>

<option selected="selected">慕课网</option>

<option>aaron</option>

<option selected="selected">博客园</option>

</select>

</form>

**$("input:checked").removeAttr('checked') //查找所有input所有勾选的元素(单选框,复选框),移除input的checked属性**

**$("option:selected").removeAttr('selected') //查找所有option元素中,有selected属性被选中的选项, 移除option的selected属性**

### 1.2.9this选择器

this，表示当前的上下文对象是一个html对象，可以调用html对象所拥有的属性和方法。

$(this),代表的上下文对象是一个jquery的上下文对象，可以调用jQuery的方法和属性值。

eg:

p.addEventListener('click',function(){

//this === p

this.style.color = "red";

},false);

等价于

$('p').click(function(){

//把p元素转化成jQuery的对象

var $this= $(this)

$this.css('color','red')

})

## jQuery对象的属性和方法

### 1.3.1对象属性操作.attr()和.removeAttr()

**语法：**

**attr(传入属性名)：获取属性的值**

**attr(属性名, 属性值)：设置属性的值**

**attr(属性名,函数值)：设置属性的函数值**

**attr(attributes)：给指定元素设置多个属性值，即：{属性名一: “属性值一” , 属性名二: “属性值二” , … … }**

**.removeAttr( attributeName ) : 为匹配的元素集合中的每个元素中移除一个属性（attribute）**

**需要注意：**

**Attr就是dom节点自带的属性，例如：html中常用的id、class、title、align等**

**Property是这个DOM元素作为对象，其附加的内容，例如,tagName, nodeName, nodeType,, defaultChecked, 和 defaultSelected 使用.prop()方法进行取值或赋值等**

**eg：**

<form>

<input type="text" value="设置value" />

<input type="text" value="获取value"/>

<input type="text" value="回调拼接value" />

<input type="text" value="删除value" />

</form>

**$("input:first").attr('value','新的值 ');**

**$("input:eq(1)").attr('value');**

**$("input:eq(2)").attr('value',function(i, val){**

**return '通过function设置' +val;**

**})//通过当前属性val值设置新值，该函数的第一个参数为当前元素在元素集合中的index，第二个参数为当前元素的属性值**

**$("input:last").removeAttr('value')；**

### 1.3.2修改html内容

**1..html()方法**

获取集合中第一个匹配元素的HTML内容 或 设置每一个匹配元素的html内容，具体有3种用法：

.html() 不传入值，就是获取集合中第一个匹配元素的HTML内容

.html( htmlString ) 设置每一个匹配元素的html内容

.html( function(index, oldhtml) ) 用来返回设置HTML内容的一个函数

**注意：在设置与获取上需要注意的一个最重要的问题，这个操作是针对整个HTML内容（不仅仅只是文本内容）**

**2..text()方法**

得到匹配元素集合中每个元素的文本内容结合，包括他们的后代，或设置匹配元素集合中每个元素的文本内容为指定的文本内容。，具体有3种用法：

.text() 得到匹配元素集合中每个元素的合并文本，包括他们的后代

.text( textString ) 用于设置匹配元素内容的文本

.text( function(index, text) ) 用来返回设置文本内容的一个函数

注意：

1.html方法是对html元素进行操作，text是对元素的文本进行操作

2.如果处理的对象只有一个子文本节点，那么html处理的结果与text是一样的

**eg：**

<div class="left first-div">

<div class="div">

<a>:first-child</a>

<a>第二个元素</a>

<a>:last-child</a>

</div>

<div class="div">

<a>:first-child</a>

<a>第二个元素</a>

<a>:last-child</a>

</div>

</div>

<h4>显示通过html方法获取到的内容</h4>

<p></p>

<h4>显示通过text方法获取到的内容</h4>

<p></p>

**$('p:first').text( $(".first-div").html() );**

**$('p:last').text( $(".first-div").text() );**

### 1.3.3修改表单value值

该方法用于修改<select>中option的文本,<input>中的value以及<textarea>

**.val()方法**

1. **.val()无参数，获取匹配的元素集合中第一个元素的当前值**
2. **.val( value )，设置匹配的元素集合中每个元素的值**
3. **.val( function ) ，一个用来返回设置值的函数**

**注意：**

1. **通过.val()处理select元素， 当没有选择项被选中，它返回null**
2. **.val()方法多用来设置表单的字段的值**
3. **如果select元素有multiple（多选）属性，并且至少一个选择项被选中， .val()方法返回一个数组，这个数组包含每个选中选择项的值**

**html(),text(),val()的区别**

1 .html()是用来读取元素的html内容（包括html标签），.text()用来读取元素的纯文本内容，包括其后代元素，.val()是用来读取表单元素的"value"值。其中.html()和.text()方法不能使用在表单元素上,而.val()只能使用在表单元素上；另外.html()方法使用在多个元素上时，只读取第一个元素；.val()方法和.html()相同，如果其应用在多个元素上时，只能读取第一个表单元素的"value"值，但是.text()和他们不一样，如果.text()应用在多个元素上时，将会读取所有选中元素的文本内容。

2 .html(htmlString),.text(textString)和.val(value)三种方法都是用来替换选中元素的内容，如果三个方法被运用在多个选中元素上时，那么三个方法都将会替换所有选中元素的内容。

**3.一定要注意：通过这三个方法取出来的都是string类型，如果想要转换为其他类型，如number，需要通过parseInt()等函数**

eg：

<select id="single">

<option>慕课网</option>

<option>博客园</option>

</select>

<select id="multiple" multiple="multiple">

<option selected="selected">imocc</option>

<option>慕课网</option>

<option selected="selected">博客园</option>

</select>

<input type="text" value="click a button" />

<p></p>

**$("p").text( $("#single").val() );//显示慕课网**

**$("p").text( $("#multiple").val());//显示被选中的项，即imocc,博客园**

**$("input[type='text']").val('修改表单的字段');//修改input的value**

### 1.3.4增加样式addClass()

当一个节点（或称为一个标签）含有多个class时，DOM元素响应的className属性获取的不是class名称的数组，而是一个含有空格的字符串，这就使得多class操作变得很麻烦。因此jQuery提供了一个addClass方法用于动态增加类名

**addClass( className )方法**

1. **.addClass( className ) : 为每个匹配元素所要增加的一个或多个样式名**
2. **(元素或元素集合).addClass( function(index, currentClass) ) : 这个函数返回一个或更多用空格隔开的要增加的样式名，其中currentClass代表当前被选中元素的所有类名，用空格隔开，index表示当前调用function的元素在元素集合中的index**

注意：addClass方法不会替换样式，他只能添加一个新的样式

eg：

<p class="orgClass">

**$("p").addClass("newClass");**

**$("div").addClass(function(index,className) {**

**//判断已有的class中是否存在旧的名为imooc的类名，若未找到，则返回值==-1，不做处理，否则返回值!==-1，再加入新的类名用以替换**

**if(-1 !== className.indexOf('imooc')){**

**//this指向匹配元素集合中的当前元素**

**$(this).addClass('imoocClass')**

**}**

**});**

### 1.3.5删除样式removeClass()

从匹配的元素中删除全部或者指定的class

**.removeClass( )方法**

1. **.removeClass( [className ] )：每个匹配元素移除的一个或多个用空格隔开的样式名**
2. **.removeClass( function(index, className) ) ： 一个函数，返回一个或多个将要被移除的样式名，函数应有返回值，返回值即是被删除的类；**

**注意：如果一个类名作为一个参数,只有这样该类会被从匹配的元素集合中删除 。 如果没有样式名作为参数，那么所有的类将被移除**

**eg：**

<div class="left">

<div class="aaron newClass">

<p>newClass</p>

</div>

<div class="aaron newClass">

<p>newClass</p>

</div>

</div>

<div class="right">

<div class="aa bb imoocClass">

<article>

<p>imoocClass</p>

</article>

</div>

<div>

<article>

<p>imoocClass</p>

</article>

</div>

</div>

**$('.left div').removeClass('newClass');**

**$('.right > div:first').removeClass(function(index,className){**

**//className = aa bb imoocClass**

**$(this).next().addClass(className)//将自身的所有类附加给下一个元素**

**return 'aa';//返回被删除的类**

**})**

### 1.3.6切换样式.toggleClass()

在做某些效果的时候，可能会针对同一节点的某一个样式不断的切换，也就是addClass与removeClass的互斥切换，比如隔行换色效果

**.toggleClass( )方法：在匹配的元素集合中的每个元素上添加或删除一个或多个样式类,取决于这个样式类是否存在或值切换属性。即：如果存在（不存在）就删除（添加）一个类**

**1 .toggleClass( className )：在匹配的元素集合中的每个元素上用来切换的一个或多个（用空格隔开）样式类名**

**2 .toggleClass( className, boolean)：boolean一个布尔值，true为添加类，false为删除类**

**3 .toggleClass( [switch ] )：一个用来判断样式类添加还是移除的 布尔值**

**4 .toggleClass( function(index, class, switch) [, switch ] )：用来返回在匹配的元素集合中的每个元素上用来切换的样式类名的一个函数。接收元素的索引位置和元素旧的样式类作为参数**

**注意：**

**toggleClass是一个互斥的逻辑，也就是通过判断对应的元素上是否存在指定的Class名，如果有就删除，如果没有就增加**

**toggleClass会保留原有的Class名后新增，通过空格隔开**

<table id="table" width="50%" border="0" cellpadding="3" cellspacing="1">

<tr>

<td>慕课jQuery入门</td>

<td>慕课jQuery入门</td>

</tr>

<tr>

<td>慕课jQuery入门</td>

<td>慕课jQuery入门</td>

</tr>

<tr>

<td>慕课jQuery入门</td>

<td>慕课jQuery入门</td>

</tr>

<tr>

<td>慕课jQuery入门</td>

<td>慕课jQuery入门</td>

</tr>

<tr>

<td>慕课jQuery入门</td>

<td>慕课jQuery入门</td>

</tr>

</table>

**//给所有的tr元素加一个class="c"的样式**

**$("#table tr").toggleClass("c");**

**//所有偶数行的样式被删除**

**$("#table tr:odd").toggleClass("c");**

**$("#table tr:even").toggleClass("c", true); //这个操作没有变化，给所有的奇数行增加样式c**

### 1.3.7样式操作.css()

**.css() 方法：获取元素样式属性的计算值或者设置元素的CSS属性**

**获取：**

1. **.css( propertyName ) ：获取匹配元素集合中的第一个元素的样式属性的计算值**
2. **.css( propertyNames )：传递一组属性名数组，返回一个属性值对象结果**

**设置：**

1. **.css(propertyName, value )：设置CSS**
2. **.css( propertyName, function )：可以传入一个回调函数，返回取到对应的值进行处理**
3. **.css( properties )：可以传一个对象，同时设置多个样式，但要注意，对象的属性和值都应为string类型**

**eg：**

<div class="first">获取颜色</div>

<p></p>

<div class="second">获取文字尺寸</div>

<p></p>

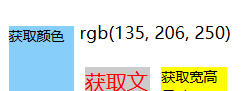
<div class="third">获取宽高尺寸</div>

<p></p>

**//颜色都会转化成统一的rgb标示**

**$('p:eq(0)').text( $('.first').css("background-color") )**

**var value = $('.first').css(["width","height"]);//value的值为一个对象{width:××,height:××}**



<div class="fourth">设置颜色设置文字尺寸</div>

<div class="fifth">设置颜色设置文字尺寸</div>

<div class="sixth">通过回调设置新的值</div>

<div class="seventh">同时设置多少个样式</div>

**$('.sixth').css("width",function(index,value){**

**//value带单位，先分解**

**value = value.split('px');**

**//返回一个新的值，在原有的值上，增加50px，注意，此处要先转换成number，否则值会变成字符串连接，即60+50=6050；**

**return (Number(value[0]) + 50);**

**})**

**$('.seventh').css({"color":"green",**

**"font-size":"30px"**

**})**



### 1.3.8.data()数据存储

该数据存储方式可用来进行数据预设或存储

**jQuery提供的存储接口**

**jQuery.data( element, key, value ) //静态接口,存数据**

**jQuery.data( element, key ) //静态接口,取数据**

**.data( key, value ) //实例接口,存数据，提倡**

**.data( key ) //实例接口,取数据，提倡**

同样的也提供2个对应的删除接口：

**jQuery.removeData( element [, name ] )**

**.removeData( [name ] )//提倡**

**eg：**

<div class="left">

<div class="aaron">

<p>点击看结果</p>

<p>jQuery.data</p>

</div>

<div><span></span></div>

</div>

<div class="right">

<div class="aaron">

<p>点击看结果</p>

<p>.data</p>

</div>

<div><span></span></div>

</div>

<script type="text/javascript">

**//给整个.left div添加监听**

**$('.left').click(function() {**

**var ele = $(this);**

**//通过jQuery.data方式设置数据**

**$.data(ele, "a", "data test")**

**$.data(ele, "b", {**

**name : "慕课网"**

**})**

**var reset = $.data(ele, "a") + "</br>" + $.data(ele, "b").name**

**ele.find('span').append(reset)**

**})**

**$('.right').click(function() {**

**var ele = $(this);**

**//通过.data方式设置数据**

**ele.data("a", "data test")**

**ele.data("b", {**

**name: "慕课网"**

**})**

**//通过.data方式取出数据**

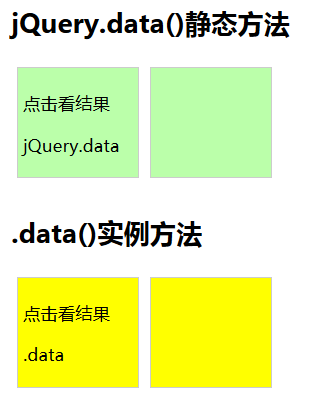
**var reset = ele.data("a") + "</br>" + ele.data("b").name**

**ele.find('span').append(reset)**

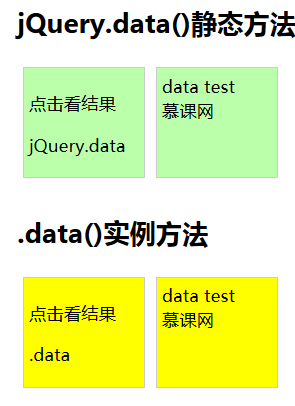
**})**

</script>

**注意：单个$符号代表整个jQuery因此，$.data()就是jQuery.data()**



**点击后**



# jQuery DOM操作

原生JavaScript代码在动态处理节点上不够灵活，不同浏览器的兼容性较差，无法批量地对节点进行创建和操作

## 2.1 DOM节点的创建

原生的JavaScript创建节点复杂又繁琐，jQuery提供简便的方法来创建节点

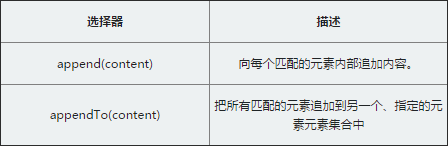
**eg：**

**$("<div class='right'><div class='aaron'>动态创建DIV元素节点</div></div>")**

## 2.2插入节点

### 2.2.1插入到节点内部的尾部

**动态创建了节点，还需要append()和appendTo()添加进文档**



append()方法相当于原生JavaScript中的appedChild()方法。$(“A”).appendTo($(“B”))方法则是将A集合的所有元素添加到B集合的每个元素中

**注意： 1.append()方法支持多个参数传递，**

**eg：**

**$(".content").append('<div class="append">aaa</div>',"<p>ssss</p>")**

**2.若A是文档已有元素，appendTo()方法会将该已有元素从原位置删除**

eg：

<span>

<a style="color:red">xixi</a>

<a>haha</a>

</span>

<button id="bt1">点击通过jQuery的append添加元素</button>

<button id="bt2">点击通过jQuery的appendTo添加元素</button>

<div class="content">

<p></p>

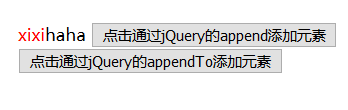
<p></p>

</div>

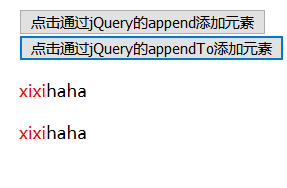
**$("#bt2").on('click', function() {**

**$('span>a').appendTo($(".content>p"))**

**})**

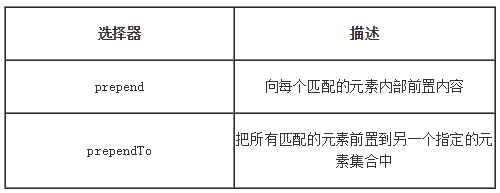


**点击后**



### 2.2.2插入到节点内部的头部prepend()

.prepend()方法将指定元素插入到匹配元素里面作为它的第一个子元素 (如果要作为最后一个子元素插入用.append()).



**注意：append和prepend的参数都只能是string不能是jQuery选择器筛选的对象**

### 2.2.3插入到节点外部after(),before()

之前的append方法都是新节点插入到选中节点的内部，作为父子节点，jQuery提供了一种将新节点插入选中节点外部，作为兄弟节点的方法



**1.before与after都是用来对相对选中元素外部增加相邻的兄弟节点**

**2. 2个方法都支持多个参数传递after(div1,div2,....)**

### 2.2.4插入到节点外部insertAfter()与insertBefore()



**1..before()和.insertBefore()实现同样的功能。主要的区别是： 对于before()，目标元素在函数前面，内容作为参数，而.insertBefore()刚好相反，内容在方法前面，它将被放在参数里目标元素的前面**

**2..after()和.insertAfter() 实现同样的功能。主要的不同是语法：对于after()选择表达式在函数的前面，参数是将要插入的内容。对于 .insertAfter(), 刚好相反，内容在方法前面，它将被放在参数里目标元素的后面**

**3.before、after与insertBefore。insertAfter后面的不支持多参数处理**

**4.若插入的内容是源文档中的元素，则该元素被删除**

## 2.2删除节点

### 2.2.1清空方法empty()

empty()可以清空元素的所有子节点，包括文本节点，但不会清除该元素，eg：

<div id="test">

<p>p元素1</p>

<p>p元素2</p>

</div>

**$("#test").empty();//只删除了两个p节点，本身id=test的div元素没有被删除**

### 2.2.2移除元素remove()

remove与empty一样，都是移除元素的方法，但是remove会将元素自身移除，同时也会移除元素内部的一切

remove可以通过传递一个选择器作为参数来过滤将被移除的匹配元素集合，可以选择性的删除指定的节点

eg：

<div class="test1">

<p>p元素1</p>

<p>p元素2</p>

</div>

<div class="test2">

<p>p元素3</p>

<p>p元素4</p>

</div>

**$(".test1").remove();**

**$("p").remove(":contains('3')");//相当于$("p").filter(":contains('3')").remove()**

**区别：**

**empty方法**

严格地讲，empty()方法并不是删除节点，而是清空节点，它能清空元素中的所有后代节点

empty不能删除自己本身这个节点

**remove方法**

该节点与该节点所包含的所有后代节点将同时被删除

提供传递一个筛选的表达式作为参数，删除指定合集中的元素

二者都可以通过过滤器的方式进行筛选，效果等同于有参数的remove，eg：

**$("p").filter(":contains('3')").remove()**

**$("p").filter(":contains('3')").empty()**

### 2.2.3暂时删除detach()

如果我们希望临时删除页面上的节点，但是又不希望节点上的数据与事件丢失，并且能在下一个时间段让这个删除的节点重新显示到页面，这时候就可以使用detach方法来处理

该方法可以暂时将元素移出文档流，但内存中仍保留该元素的内存模型对象，detach方法的返回值用于存储内存中的该元素

eg：

<p>P元素1，默认给绑定一个点击事件</p>

<p>P元素2，默认给绑定一个点击事件</p>

**var p = $("p").detach();//移除**

**$("body").append(p);//重新添加到body的尾部(不在原位)**

注意:detach也可以通过参数进行筛选**(只有empty不行)**

## 2.3复制和替换节点

### 2.3.1拷贝方法clone()

.clone()方法：复制所有匹配的元素集合，包括所有匹配元素、匹配元素的下级元素、文字节点。如果需要连自身和子元素的事件、数据一起复制，应添加一个true参数，该方法返回被复制元素的jQuery对象

clone()方法时，在将它插入到文档之前，我们可以修改克隆后的元素或者元素内容，如： $(this).clone().css('color','red') 增加了一个颜色

eg：

<div class="test1">

<p>嘻嘻</p>

<button id="bt1">点击显示</button>

</div>

<button id="bt2">clone</button>

<div class="test2"></div>

**$("#bt1").click(function() {**

**alert("嘻嘻");**

**})**

**$("#bt2").click(function(){**

**var myCloneNode=$(".test1").clone();**

**$(".test2").append(myCloneNode);**

**})**

### 2.3.2替换replaceWith()和relpaceAll()

**.replaceWith( newContent )**：用提供的内容替换集合中所有匹配的元素并且返回被删除元素的集合

**.replaceAll()**和.replaceWith()功能类似，但是目标和源相反，用上述的HTML结构，我们用replaceAll处理

注意:

eg:

<div class="right">

<div>

<p>第一段</p>

<p>第二段</p>

<p>第三段</p>

</div>

<div>

<p>第四段</p>

<p>第五段</p>

<p>第六段</p>

</div>

</div>

**$(".right > div:first p:eq(1)").replaceWith('<a style="color:red">replaceWith替换第二段的内容</a>');**

**$('<a style="color:red">replaceAll替换第六段的内容</a>').replaceAll('.right > div:last p:last');**

### 2.3.3包裹方法wrap()

如果要将元素用其他元素包裹起来，也就是给它增加一个父元素，针对这样的处理，JQuery提供了一个wrap方法

**.wrap( wrappingElement )：在集合中匹配的每个元素周围包裹一个HTML结构**

**.wrap( function ) ：一个回调函数，返回用于包裹匹配元素的 HTML 内容或 jQuery 对象**

eg：

**$('p').wrap('<div></div>')//结果为<div><p></p></div>**

**$('p').wrap(function() {**

**return '<div></div>';**

**})//与上一种类似，只是写法不一样**

<a>a元素</a>

**$('a').wrap(function() {**

**return '<div class="' + $(this).text() + '" />';**

**})//结果为**

**<div class=”a元素”>**

**<a>a元素</a>**

**</div>**

### 2.3.4删除父元素unwrap()

unwarp()方法 ，作用与wrap方法是相反的。将匹配元素集合的父级元素删除，保留自身和兄弟元素。该函数不需要参数

**eg：**

<div>

<p>p元素</p>

</div>

**$('p').unwarp();**//结果为<p>p元素</p>

### 2.3.5全部包裹wrapAll()

wrap是针对单个dom元素处理，如果要将集合中所有的元素用一个指定的html元素包裹起来(wrap会将集合的每个元素都用该html元素包裹一次)，也就是给他们增加一个共同的父元素，可以使用wrapAll方法

**.wrapAll( wrappingElement )：**给集合中匹配的元素增加一个外面包裹HTML结构

**eg：**

<p>p元素</p>

<p>p元素</p>

**$('p').wrapAll('<div></div>');//结果为:**

**<div>**

**<p>p元素</p>**

**<p>p元素</p>**

**<div>**

### 2.3.6内部包裹wrapInner()

**.wrapInner( wrappingElement )：**给集合中匹配的元素的内部，增加包裹的HTML结构

eg：

<div>p元素</div>

<div>p元素</div>

**$('div').wrapInner('<p></p>')//结果为:**

**<div>**

**<p>p元素</p>**

**</div>**

**<div>**

**<p>p元素</p>**

**</div>**

## 2.4遍历节点

### 2.4.1返回子元素children()

如果想快速查找合集里面的第一级子元素，此时可以用children()方法。这里需要注意：.children(selector) 方法是返回匹配元素集合中每个元素的所有子元素（仅儿子辈，这里可以理解为就是父亲-儿子的关系）

**该对象允许给定一个选择器参数，eg:**

<div class="div">

<ul class="level-1">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

<div class="div">

<ul class="level-2">

<li class="item-1">1</li>

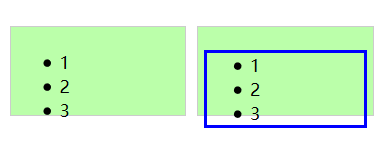
<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

**$('.div').children(':last').css('border', '3px solid blue');**



### 2.4.2查找后代元素find()

如果想快速查找DOM树中的元素的后代元素，此时可以用find()方法，注意children是父子关系查找，find是后代关系（包含父子关系）

**注意：**

* 1. **选择器表达式对于 .find() 是必需的参数。如果我们需要实现对所有后代元素的取回，可以传递通配选择器 '\*'**
  2. **find只在后代中遍历，不包括自己。**

**eg：**

<div class="div">

<ul class="level-1">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

<div class="div">

<ul class="level-2">

<li class="item-1">1</li>

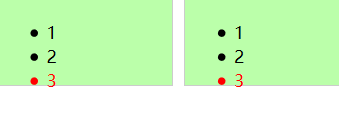
<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

**$(".div").find("li:contains('3')").css("color","red");**



### 2.4.3查找父元素parent ()

如果想快速查找合集里面的每一个元素的父元素（这里可以理解为就是父亲-儿子的关系），此时可以用parent(selector)方法

该方法有个可选的选择器参数

eg：

<div class="div">

<ul class="son">

<li class="grandson">1</li>

</ul>

</div>

查找ul的父元素div, $(ul).parent()

### 2.4.4查找祖辈元素parents ()

如果想快速查找合集里面的每一个元素的所有祖辈元素，此时可以用parents(selector)方法(查找结果包括父元素)

该方法有个可选的选择器参数

eg：

<div class="div">

<ul class="son">

<li class="grandson">1</li>

</ul>

</div>

**$("li").parents(".son").addClass("test1");**



### 2.4.5向上最近查找closest ()

closest (selector),从元素本身开始，在DOM 树上逐级向上级元素匹配，并返回最先匹配的祖先元素

**.parents()和.closest()的区别：**

** 1.起始位置不同：.closest开始于当前元素 .parents开始于父元素**

** 2.遍历的目标不同： closest向上查找，直到找到一个匹配的就停止查找，parents一直查找到根元素，并将匹配的元素加入集合**

**3.** **结果不同：.closest返回的是包含零个或一个元素的jquery对象，parents返回的是包含零个或一个或多个元素的jquery对象**

**eg：**

<ul class="level-2">

<li class="item-a">A</li>

<li class="item-b">B

<ul class="level-3">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

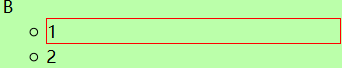
</ul>

</li>

<li class="item-c">C</li>

</ul>

**$(".item-1").closest("li").css("border","1px red solid");**



### 2.4.6查找下一个兄弟节点next()

如果想快速查找指定元素集合中每一个元素紧邻的后面同辈元素的元素集合，此时可以用next(selector)方法

**eg:**

<div class="div">

<ul class="level-1">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

<div class="div">

<ul class="level-2">

<li class="item-1">1</li>

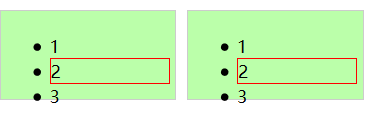
<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

**$(".div>ul>li:nth-child(3n-2)").next().css("border","1px red solid");**



### 2.4.7查找前一个兄弟节点prev()

如果想快速查找指定元素集合中每一个元素紧邻的前面同辈元素的元素集合，此时可以用prev(selector)方法

<div class="div">

<ul class="level-1">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

<div class="div">

<ul class="level-2">

<li class="item-1">1</li>

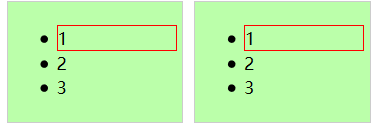
<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

**$(".item-2").prev().css("border","1px red solid");**



### 2.4.8查找前一个兄弟节点siblings()

如果想快速查找指定元素集合中每一个元素的同辈元素，此时可以用siblings(selector)方法

**注意：该方法选择的同辈元素不包括它自己**

**eg：**

<div class="div">

<ul class="level-1">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

<div class="div">

<ul class="level-2">

<li class="item-1">1</li>

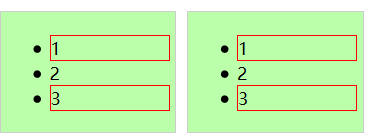
<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

</div>

**$(".item-2").siblings().css("border","1px red solid");**



### 2.4.9向jQuery对象添加节点add()

往这个jQuery对象集合中添加新的元素可使用add(节点)方法

**注意：add()的参数可以几乎接受任何的$()，包括一个jQuery选择器表达式，DOM元素，或HTML片段引用。**

有三种添加节点的方式：

**第一种：传递选择器**

**$('li').add('p');**

**第二种：传递DOM元素**

**$('li').add(document.getElementsByTagName('p')[0]);**

**第三种：临时创建标签加入jQuery对象**

**$('li').add('<p>新的p元素</p>').appendTo(目标位置);**

**eg：**

<div class="div">

<ul>

<li>list item 1</li>

<li>list item 2</li>

<li>list item 3</li>

</ul>

<p>新的p元素</p>

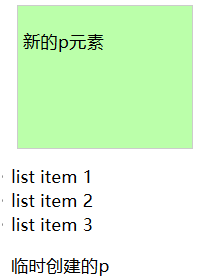
</div>

<div class="right"></div>

**$('li').add('p').css('background', 'red');**



**$('li').add('<p>临时创建的p</p>').appendTo($('.right'));**



### 2.4.10遍历jQuery对象each()

jQuery对象是一组节点的集合，对jQuery对象进行操作，就是对集合的每一个节点进行操作，其中隐式地使用了each()方法

each是一个for循环的包装迭代器，参数是一个回调方法，该回调方法有2个固定的实参，索引与元素，each回调方法中的this指向当前迭代的dom元素

**.each(function(index, element) {**

**index 当前元素的位置索引**

**element是对应的是当前节点元素**

**this 指向的是当前迭代的dom元素**

**})**

**eg：**

<ul>

<li>list item 1</li>

<li>list item 2</li>

<li>list item 3</li>

<li>list item 4</li>

<li>list item 5</li>

<li>list item 6</li>

</ul>

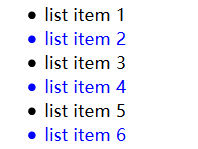
**$("li").each(function(index, element) {**

**if (index % 2) {**

**$(this).css('color','blue')**

**}**

**})**



# jQuery事件

## 3.1 鼠标事件

### 3.1.1点击事件click()和dbclick()

click方法用于监听用户单击操作，dbclick方法用于监听用户双击操作。

**方法一：$ele.click()不带参数**

**该方法一般不用于添加事件，而是用于触发元素绑定的事件，这里是用于触发$ele元素已经绑定的事件**

**方法二：$ele.click( function(event){**

**//this指向触发事件的元素，即ele**

**//event.target也指向触发事件的元素，但this的指向可以改变，该属性的指向不可改变**

**event.target.textContent获取事件触发节点的文本内容**

**//event.** **currentTarget指向当前处理事件的元素，但this的指向可以改变，该属性的指向不可改变**

**请注意！火狐不支持event事件，IE不支持currentTarget属性，最稳妥的写法是：**

**var e = event || arguments.callee.caller.arguments[0] || window.event;**

**url = $(e.currentTarget || e.srcElement).children("p").text();**

**} )**

**方法三：$ele.click(data, function(event){**

**//该方法和第二种的区别是多了一个data，该参数在绑定事件监听事传递给元素，通过event.data调用**

**} )**

**dbclick()方法同上，只不过一个是单机一个是双击**

**注意：**

**1.click事件其实是由mousedown与mouseup 2个动作构成，所以点击的动作只有在松手后才触发**

**2.不可以在同一个元素中同时绑定click()和dbclick()方法**

### 3.1.2鼠标按下释放事件mousedown()和mouseup()

mousedown的快捷方法可以监听用户鼠标按下的操作，mouseup快捷方法可以监听用户鼠标弹起的操作

**具体语法同click事件(三种)**

**注意：**

**1.** **mousedown强调是按下触发，只要按下就触发，mouseup强调是松手触发，如果在一个元素按住了鼠标不放，并且拖动鼠标离开这个元素，并释放鼠标键，这只能是算作一个mousedown事件**

**2.** **mouseup与mousedown组合起来就是click事件**

**3.对于mousedown和mouseup，作为函数参数的事件event，有个独有的属性event.which,** **敲击鼠标左键which的值是1，敲击鼠标中键which的值是2，敲击鼠标右键which的值是3**

**eg:**

<button>弹出回调中的鼠标键</button>

**$("button:eq(0)").mousedown(function(e) {**

**alert('e.which: ' + e.which);**

**})**

### 3.1.3鼠标移动事件mousemove ()

mousemove()方法可以监听用户移动鼠标的的操作

**具体语法同click事件(三种)**

**注意：**

**1.mousemove事件是当鼠标指针移动时触发的，即使是一个像素**

**2.如果处理器做任何重大的处理，或者如果该事件存在多个处理函数，该方法都可能造成浏览器的严重的性能问题(因为每次鼠标移动都要触发事件)**

**3. 对于mousemove ()，作为函数参数的事件event，具有属性event.pageX和event.pageY，分别是鼠标当前的x轴坐标和y轴坐标**

**eg：**

<div class="aaron1">

<p>鼠标在绿色区域移动触发mousemove</p>

<p>移动的X位置：</p>

</div>

**$(".aaron1").mousemove(function(e) {**

**$(this).find('p:last').html('移动的X位置：' + e.pageX)**

**})**

**function data(e) {**

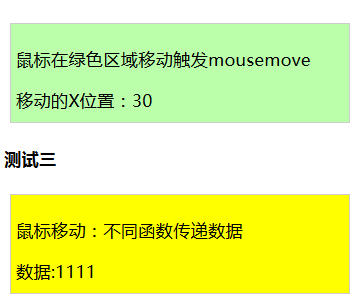
**$(this).find('p:last').html('数据:' + e.data)**

**}**

**!(function a() {**

**$(".right").mousemove(1111, data)**

**})();**



### 3.1.4鼠标移入移出事件mouseover与mouseout

mouseover()与mouseout()事件分别被用来监听鼠标的移入和移出的操作

**具体语法同click事件(三种)**

eg：

<div class="aaron1">

<p>鼠标移进此区域触发mouseover事件</p>

<a>进入元素内部,mouseover事件触发次数：</a>

</div>

**$(".aaron1 p:first").mouseover(function(e) {**

**$(".aaron1 a").html('进入元素内部,mouseover事件触发次数：' + (++n))**

**})**

### 3.1.5鼠标进入离开事件mouseenter与mouseleave

mouseenter和mouseleave可以监听鼠标是否有移到元素内部或是元素外部

**注意：他们和mouseover()与mouseout()的区别是：mouseenter事件只会在绑定它的元素上被调用，而不会在后代节点上被触发**

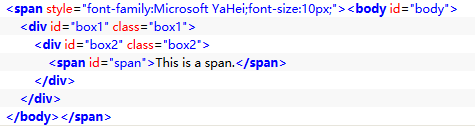
**具体语法同click事件(三种)**

**该方法主要是为了处理上一个小节的方法可能产生事件冒泡的问题，尽量使用本节方法**

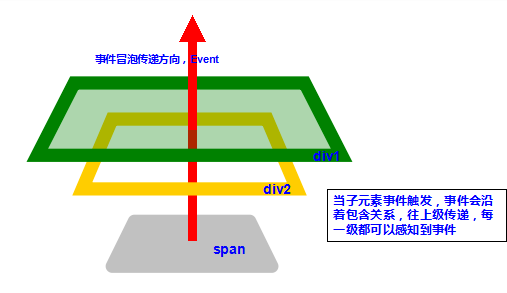
### 3.1.6事件冒泡

当监听器的作用域出现重叠时，将会出现事件冒泡，由最内层向外层冒泡，

**eg：**



若给每个节点都添加一个click事件，当点击span时



事件会一层一层地向外层传播，那么每个节点都会相应一次，如何终止事件冒泡呢？

**方法一：戳破气泡**

**通过触发事件的节点的事件处理函数添加参数e，然后调用方法e.stopPropagation()终止事件广播分发，这样事件就不会继续向外层传递了**

**eg：**

<div class="aaron2">

<p>鼠标离开此区域触发mouseleave事件</p>

</div>

<script type="text/javascript">

$("div").mouseout(function(e){

alert("移出div");

e.stopPropagation();

})

$("p").mouseout(function(e){

alert("移出p");

e.stopPropagation();

})

</script>

**方法二：通过事件冒泡触发的事件广播，触发事件的元素和处理事件的元素可能不同，我们可以通过比较event.target和event. currentTarget属性是否相同来判断是否执行事件处理**

**eg：**

<div class="aaron2">

<p>鼠标离开此区域触发mouseleave事件</p>

</div>

<script type="text/javascript">

$("div").mouseout(function(e){

if(e.target==e.currentTarget){

alert("移出div");

}

})

$("p").mouseout(function(e){

if(e.target==e.currentTarget){

alert("移出p");

}

})

</script>

**方法三：事件委托**

方法一的缺点是如果一个节点出了问题，该节点终止广播就会失效，方法很脆弱，方法二代码冗余度很高，我们可以给每个节点添加监听，通过让某个父节点统一处理，判断事件发生节点e.target属性的相关子属性的方式来判断是哪个后代节点发生了该事件，再进行相应的处理(建议使用switch判断)的方式处理事件冒泡

eg:

<div class="left">

<ul id="ul">

<li>111</li>

<li>222</li>

<li>333</li>

<li>444</li>

</ul>

</div>

**$("div").on("click", function(e) {**

**switch (e.target.textContent) {**

**case "111":**

**alert("111");**

**break;**

**case "222":**

**alert("222");**

**break;**

**case "333":**

**alert("333");**

**break;**

**case "444":**

**alert("444");**

**break;**

**}**

**});**

### 3.1.7切换事件hover()

hover()可以给元素做一个切换效果，当鼠标移入移出分别做什么操作

**语法：**

**$(selector).hover(handlerIn, handlerOut)**

handlerIn(eventObject)：当鼠标指针进入元素时触发执行的事件函数

handlerOut(eventObject)：当鼠标指针离开元素时触发执行的事件函数

eg：

<div class="aaron1">

<p>触发hover事件</p>

</div>

**$("p").hover(**

**function() {**

**$(this).css("background", 'red');**

**},**

**function() {**

**$(this).css("background", '#bbffaa');**

**}**

**)**

## 表单事件

### 3.2.1焦点事件focusin()，focusout()

当一个元素，或者其内部任何一个元素获得或失去焦点的时候触发focusin、focusout事件

**具体语法同click事件(三种)**

### 3.2.2焦点事件focus ()和blur()

focus()和blur()也用于监听捕获和失去焦点，它们和focusin与focusout()的区别在于前者只能监听自身，后者可以监听自身和后代元素

eg：

<div class="aaron">

点击聚焦：<input type="text" />

</div>

<div class="aaron1">

点击聚焦并传递参数：<input type="text" />

</div>

**$(".aaron").focusin(function() {**

**$(this).find(":input").css('border','2px solid red')**

**})**

**/\*--------------------\*/**

**function fn(e) {**

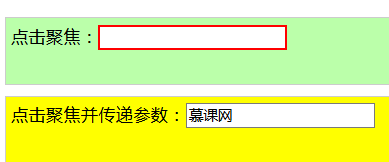
**$(this).find(":input").val(e.data)**

**}**

**!(function a() {**

**$(".aaron1").focusin('慕课网', fn)**

**})();**



**$("input:eq(0)").focus(function() {**

**$(this).css('border','2px solid red')**

**})**

**/\*--------------------\*/**

**function fn(e) {**

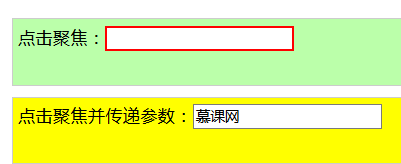
**$(this).val(e.data)**

**}**

**!(function a() {**

**$("input:eq(1)").blur('慕课网', fn)**

**})();**



### 3.2.3文本改变事件change ()

<input>元素，<textarea>和<select>这三种元素的值都是可以发生改变的， change事件可以去监听这些改变的动作(文本改动并失去焦点才触发)

**注意:对于<input>，通过监听value的值实现监听change()事件**

eg：

<div class="aaron">input：

<input class="target1" type="text" value="监听input的改变" />

</div>

<div class="aaron1">

<select class="target2">select：

<option value="option1" selected="selected">Option 1</option>

<option value="option2">Option 2</option>

</select>

</div>

<div class="aaron3">textarea：

<textarea class="target2" rows="3" cols="20">文本输入</textarea>

</div>

<div id="result"></div>

**$('.target1').change(function(e) {**

**$("#result").html(e.target.value)**

**});**

**$(".target2").change(function(e) {**

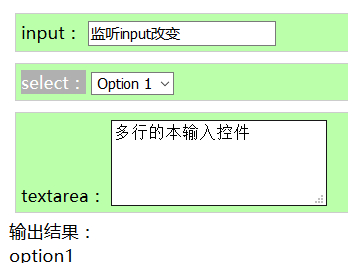
**$("#result").html(e.target.value)**

**})**

**$(".target3").change(function(e) {**

**$("#result").html(e.target.value)**

**})**



### 3.2.4文本选中事件select ()

当 textarea 或文本类型的 input 元素中的文本被选择时，会发生 select 事件。

**注意：select事件只能用于<input>元素与<textarea>元素**

**具体语法同click事件(三种)**

**如果想要获取被选中的文字，event.target和event.currentTarget属性中都有selectionStart和selectionEnd属性，因此可以通过**

**alert(e.target.value.substring(e.currentTarget.selectionStart,e.currentTarget.selectionEnd));方法获取被选中字符**

### 3.2.5提交事件submit ()

可以通过在<form>元素上绑定submit事件，监听提交表单这个动作

**以下行为都可以触发submit事件**

**<input type="submit">**

**<input type="image">**

**<button type="submit">**

**当某些表单元素获取焦点时，敲击Enter（回车键）**

**form提交表单分为submit提交和ajax提交，前者在form标签中添加action(提交的地址)和method(默认为get)，如：**

<form action=’http://www.123.com/postValue’ method=’post’>

<input type=’text’ name=’username’ />

<input type=’password’ name=’password’/>

<input type=’submit’ value=’登陆'/>

</form>

该方式提交会转向action指向的页面，**若要阻止跳转，应在submit事件中添加return false语句**

## 键盘事件

### 3.3.1按下释放事件keydown()与keyup()

keydown与keyup方法被用来监听键盘按下与松手的事件

**具体语法同click事件(三种)**

**注意：**

**1.keydown是在键盘按下就会触发**

**2.keyup是在键盘松手就会触发**

**3.理论上它可以绑定到任何元素，但keydown/keyup事件只是发送到具有焦点的元素上，不同的浏览器中，可获得焦点的元素略有不同，但是表单元素总是能获取焦点，所以对于此事件类型表单元素是最合适的。**

**4.当按键按下并没有释放时，按下的键并没有生效，只有当释放之后，输入(按下的键)才生效，即keydown事件触发在文字还没敲进文本框**

### 3.3.2按下事件keypress()

KeyPress主要用来接收字母、数字等ANSI字符，类似于keydown，而 KeyDown事件过程可以处理任何不被 KeyPress 识别的击键。诸如：功能键（F1-F12）、编辑键、定位键以及任何这些键和键盘换档键的组合等。

**具体语法同click事件(三种)**

**keypress事件与keydown的主要区别**

**1.只能捕获单个字符，不能捕获组合键**

**2.无法响应系统功能键（如delete，backspace）**

**3.不区分小键盘和主键盘的数字字符**

## 事件绑定和解绑

### 3.4.1多事件绑定on()

**基本用法：**

**.on( events , [ data ] ,[ function ] )**

**1.多个事件绑定一个函数**

**eg:**

**$("#elem").on("mouseover mouseout",function(){ });**

**2.多个事件绑定不同函数**

**$("#elem").on( {**

**mouseover:function(){},**

**mouseout:function(){}**

**} );**

**注意！当绑定不同函数时，参数只有一个，且为object类型，因此要在外部加上{}符号，以对象字面量形式创建对象**

**on的事件委托**

**.on( events ,[ selector ] ,[ data ], function(eventObject) )**

**eg:**

<div class="left">

<p class="aaron">

<a>目标节点</a> //点击在这个元素上

</p>

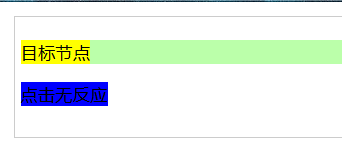
<span>点击无反应</span>

</div>

**$("div").on("click", ".aaron",123, function(e) {**

**alert(e.data);**

**})**



**事件绑定在div上，当点击div的一个后代元素时，事件触发，事件向上冒泡，当传到到选择器匹配的元素时，触发事件回调函数，进行事件处理**

### 3.4.2卸载事件off()

该方法用于卸载节点的所有事件绑定

**1.不带参数**

**$("elem").off()//卸载elem元素上所有的事件**

**2.带参数**

**$("elem").on("mousedown mouseup",function)**

**删除一个事件**

**$("elem").off("mousedown")**

**删除两个事件**

**$("elem").off("mousedown mouseup")**

## event对象

**1.** **event.target指向触发事件的节点，** **event.currentTarget是指向当前事件传播到的节点，会随事件冒泡改变，完全等同于this，currentTarget和event.target都是DOM对象，可以通过$()运算符转化为jQuery对象**

**e.target.textContent属性用于获取触发事件的元素的文本**

**2.** **event.type：获取事件的类型**

**3.** **event.pageX 和 event.pageY：获取鼠标当前相对于页面的坐标，多用于鼠标事件**

**4.** **event.preventDefault() 方法：阻止默认行为，主要用于阻止以下两种行为(不止可以阻止这两种)：**

**当点击提交按钮时阻止对表单的提交**

**阻止a标签跳转到对应URL页面**

**使用 event.isDefaultPrevented() 方法来检查是否调用了 preventDefault() 方法**

**5.** **event.stopPropagation() 方法：阻止事件冒泡，相当于把气泡戳破**

**6.** **event.which：获取在鼠标单击时，单击的是鼠标的哪个键**

## 自定义事件

### 3.6.1自定义事件的定义和触发

**$('#elem').on('Aaron', function(event,arg1,arg2) {**

**alert("自触自定义事件");**

**});//自定义事件并绑定**

**$('#elem').trigger('Aaron', arg1,arg2)//触发自定义事件,也可以通过该方法触发jQuery的原生事件**

**'Aaron'是自定义事件的事件名(标识)，arg1和arg2是传入的参数，数量不限**

**eg：**

<div class="left">

<p class="aaron">

<a>目标节点</a>

</p>

</div>

**$("a").on("myEvent", 123,function(e) {**

**alert(e.data);**

**})**

**$("a").trigger("myEvent");**

# jQuery动画

## 4.1动画基础隐藏和显示

这里的hide方法和show方法都是同时对元素的宽度、高度、不透明度进行改变来达到效果的

### 4.1.1隐藏元素hide()

让页面上的元素不可见，一般可以通过设置css的display为none属性。但是通过css直接修改是静态的布局，如果在代码执行的时候，一般是通过js控制元素的style属性，这里jQuery提供了一个快捷的方法.hide()来达到这个效果

**1.不带参数：**

**语法：**

**$elem.hide()**

**2.带参数：**

**语法：**

**$elem.hide(duration，complete)**

**option是一个对象，该对象可添加duration属性(动画所用时间，单位是毫秒)和complete属性(动画完成时调用的函数)**

**eg：**

**$("#a2").hide(3000,function() {**

**alert('执行3000ms动画完毕')**

**})**

**注意：该方法还提供了两个快捷参数 .hide("fast/slow");hide(“fast”)相当于hide(600),hide(“fast”)相当于hide(200);分别指定动画耗时为600毫秒和200毫秒**

### 4.1.2显示元素show()

jQuery同样提供了与hide相反的show方法，用于显示被隐藏的元素

**使用方法同hide()**

**eg:**

**$('elem').hide(3000).show(3000);**

**注意：**

**1.show与hide方法是修改的display属性，若要修改visibility属性，需要通过修改css实现**

**2.如果在样式中使用了display: none !important; show()方法会失效，因为加!important就表示提升这个属性的优先级，比如在其它地方又写了一个display:inline之类的 还会按照这里的none进行解释 ，这样样式中的属性none优先级较高**

**eg：**

#a1{

**display:none !important;**

}

<div id="a1">hide-show</div>

<button>直接hide-show动画</button>

**$("button").click(function() {**

**$("#a1").show(3000)**

**});//show方法失效**

### 4.1.3切换方法toggle()

jQuery提供了toggle()方法对元素进行显示隐藏的互斥切换，如果元素是最初显示，它会被隐藏，如果隐藏的，它会显示出来

**使用方法同上**

**eg：**

<div class="left">显示到隐藏</div>

**$("button:first").click(function() {**

**$(".left").toggle(3000);**

**});**

## 4.2上卷下拉效果

这里的slideDown方法和slideUp方法都是对元素的高度进行改变来达到效果的

### 4.2.1下拉动画slideDown()

.slideDown()方法将对匹配元素的高度添加动画效果，这会导致元素出现下拉滑动的效果

**语法：**

**.slideDown( duration, complete )**

**eg：**

<div id="a2">hide-show</div>

**$("button:last").click(function() {**

**$("#a2").slideDown(3000,function(){**

**alert('动画执行结束');**

**})**

**});**

持续时间（duration）是以毫秒为单位的，数值越大，动画越慢，不是越快。字符串 'fast' 和 'slow' 分别代表200和600毫秒的延时。如果提供任何其他字符串，或者这个duration参数被省略，那么默认使用400 毫秒的延时。

**注意：**

**1.下拉动画是从无到有，所以一开始元素是需要先隐藏起来的，可以设置display:none**

**2.如果多个元素一同执行该动画，那么回调函数会在每一个元素执行完动画后都执行一次，而不是这组 动画整体才执行一次**

**eg：**

<div class="b">hide-show1</div>

<div class="b">hide-show2</div>

<div class="b">hide-show3</div>

**$("button:last").click(function() {**

**$(".b").slideDown(3000,function(){**

**alert('动画执行结束')**

**})**

**});**

### 4.2.2上卷动画slideUp()

.slideDown()方法将对匹配元素的高度添加动画效果，这会导致元素出现上卷滑动的效果

**语法：**

**.slideUp( duration, easing, function)**

**duration是耗时，easing有两个可选的值，规定在动画的不同点上元素的速度，”swing”是默认的，表示在开头/结尾移动慢，在中间移动快，"linear"表示匀速移动，function是回调函数**

### 4.2.3上卷下拉切换slideToggle()

jQuery提供了slideToggle ()方法对元素进行上卷下拉的互斥切换，如果元素是最初显示，它就执行上卷，如果隐藏的，它会执行下拉，该函数对元素高度进行操作

**语法：**

**.slideToggle( duration , complete )**

**eg：**

**$("button").click(function() {**

**$("#a1").slideToggle("fast",function(){**

**alert("回调函数执行");**

**})**

**});**

## 4.3淡入淡出效果

这里的fadeOut方法和fadeIn以及fadeTo方法都是对元素的透明度进行改变来达到效果的

### 4.3.1淡出动画fadeOut()

fadeOut()函数用于隐藏所有匹配的元素，并带有淡出的过渡动画效果

**语法：**

**.fadeOut( duration , complete )**

**eg：**

<p>测试文字淡入效果</p>

<p>慕课网,专注分享</p>

淡出的隐藏效果：

<select id="animation">

<option value="1">fadeOut( )</option>

<option value="2">fadeOut( "slow" )</option>

<option value="3">fadeOut( 3000 )</option>

<option value="4">fadeOut( 2000, function )</option>

<option value="5">fadeOut( 1000, "linear" )</option>

<option value="6">fadeOut( {

duration: 1000

} )</option>

</select>

**$("#btnFadeOut").click(function() {**

**var v = $("#animation").val();**

**if (v == "1") {**

**$("p").fadeOut();**

**} else if (v == "2") {**

**$("p").fadeOut("slow");**

**} else if (v == "3") {**

**$("p").fadeOut(3000);**

**} else if (v == "4") {**

**$("p").fadeOut(2000, function() {**

**alert("隐藏完毕!");**

**});**

**} else if (v == "5") {**

**$("p").fadeOut(1000, "linear");**

**} else if (v == "6") {**

**$("p").fadeOut({**

**duration: 1000**

**});**

**}**

**});**

**注意：该方法把元素透明直到透明度为0%的时候将元素设置为display:none;**

### 4.3.2淡入动画fadeIn()

fadeOut是淡出效果，相反的还有淡入效果fadeIn,方法使用上两者都是一致的，只是结果相反

**淡入的动画原理：先将其display设为可见，然后操作元素的不透明度从0%逐渐增加到100%**

### 4.3.3淡入淡出切换fadeToggle()

fadeToggle()函数用于切换所有匹配的元素，如果元素当前是可见的，则将其隐藏(淡出)；如果元素当前是隐藏的，则使其显示(淡入)

**语法：**

**.fadeToggle( duration ,complete )**

### 4.3.4渐变动画fadeTo()

fadeIn与fadeOut都是修改元素样式的opacity属性，但是他们都有个共同的特点，透明度的变化要么是从0到1，要么是从1到0，jQuery提供了一个fadeTo()方法，可以将透明度渐变到指定的值

**语法**

**.fadeTo( duration, opacity ,function)**

**注意：通过该方法将透明度设置到0后，并不会设置display:none;**

eg：

<p>测试文字透明度效果</p>

<p>慕课网,专注分享</p>

透明度的设置效果：

<select id="animation">

<option value="1">fadeTo( "slow" ,0.5 )</option>

<option value="2">fadeTo( 1000 ,0.2 )</option>

<option value="3">fadeTo( 1000 ,0.9 ,complete)</option>

</select>

**$("#btnFadeSwitch").click(function() {**

**var v = $("#animation").val();**

**if (v == "1") {**

**$("p").fadeTo("slow", 0.5);**

**} else if (v == "2") {**

**$("p").fadeTo(1000, 0);**

**} else if (v == "3") {**

**$("p").fadeTo(1000, 0.9, function() {**

**alert('完成')**

**});**

**}**

**});**

## 4.4自定义动画

### 4.4.1 animate ()动画

animate方法相对灵活，可以精确的控制样式属性从而执行动画

**语法1：**

**.animate( properties ,duration , easing , function)**

**properties是一组css属性的对象字面量，duration是动画执行时间，easing有两个可选的值，规定在动画的不同点上元素的速度，”swing”是默认的，表示在开头/结尾移动慢，在中间移动快，"linear"表示匀速移动，function是回调函数**

**eg：**

.animate({

left: 50,

width: '50px'

opacity: 'show',

fontSize: "10em",

}, 500);

**属性值要用引号括起来！**

**注意：**

**1.可以通过给某属性设置'show', 'hide', 和 'toggle'来控制元素的显示，隐藏和切换效果**

eg：

.animate({

width: "toggle"

});

.animate({

opacity: "toggle"

});

**2.CSS 样式使用 DOM 名称（比如 "fontSize"）来设置，而非 CSS 名称（比如 "font-size"）。**

**3. 如果提供一个以+= 或 -=开始的值，那么目标值就是以这个属性的当前值加上或者减去给定的数字来计算的**

eg:

.animate({

left: “+=50px”

}, "slow");

**语法2：**

**.animate( properties, options)**

**options是一个对象字面量，应具有以下属性**

**duration - 设置动画执行的时间**

**easing - 规定要使用的 easing 函数，过渡使用哪种缓动函数**

**step：规定每个动画的每一步完成之后要执行的函数**

**progress：每一次动画调用的时候会执行这个回调，就是一个进度的概念**

**complete：动画完成回调**

**注意：step和progress都是某个动画每执行一个阶段后调用的函数，个动画执行的进度，可以用2个指标来描述，一个是动画属性的变动值，另一个是动画在整个执行过程的进度，如：一个div向左移动100px，用时200毫秒，我们可以说：div向左每移动1个px，可以看成动画执行了一步；也可以说：把200毫秒分成100个时间段，动画开始后，把每个时间段的流逝，看作动画执行了一步。在实际运行中，，每一步具体是怎么分解的由系统决定**

**step和progress都具有两个参数，now和fx，now是指当前属性值，若当前动画变化的属性值不止一个，则交替输出属性值，fx是二者的不同之处：**

**step： fx返回的是和css相关的属性，有很多相关属性，比如elem 表示前正在执行动画的元素，start和end分别为动画属性的初始和结束的值，prop为进行中的动画属性。当有多个属性参与变化时，fx只能关联我们在代码中设置的最后一条CSS属性值**

**eg：**

**$("div").animate({**

**left:100,**

**opacity:0.2**

**},{**

**duration:1000,**

**step:function(now,fx){**

**console.log("返回的CSS属性是："+fx.prop);**

**console.log("属性初始值："+fx.start);**

**console.log("属性结束值："+fx.end);**

**console.log("属性当前值："+fx.now);**

**}**

**})**

**以上代码只能返回 opacity 属性的相关值，而无法返回left的值，虽然这个值也参与了动画效果**

**progress： fx返回的是动画进度本身的属性；**

### 4.4.2 停止动画stop()

动画在执行过程中是允许被暂停的，当一个元素调用.stop()方法，当前正在运行的动画（如果有的话）立即停止

**语法:**

**stop(clearQueue , jumpToEnd )**

**stop有两个boolean类型的可选参数:**

**1.stop() 停止当前动画，并继续向下执行代码**

**2.** **stop(true) 如果同一元素调用多个动画方法，尚未被执行的动画被放置在元素的效果队列中，依次准备执行，若调用stop(true)，会暂停动画并清空效果队列，该元素的所有动画效果都不会执行**

**3.** **.stop(true,true) 当前动画将停止，并清空效果队列(后续动画也不会执行)，但该元素上的 CSS 属性会被立刻修改成被停止动画的目标值**

**eg:**

<div id="aaron">内部动画</div>

<input id="exec" type="button" value="执行动画"><br /><br />

点击观察动画效果：

<select id="animation">

<option value="1">stop()</option>

<option value="2">stop(true)</option>

<option value="3">stop(true,true)</option>

</select>

<input id="stop" type="button" value="停止动画">

**$("#exec").click(function(){**

**$("#aaron").animate({**

**height: 300**

**}, 5000)**

**$("#aaron").animate({**

**width: 300**

**}, 5000)**

**$("#aaron").animate({**

**opacity: 0.1**

**}, 2000)**

**})**

**$("#stop").click(function() {**

**var v = $("#animation").val();**

**var $aaron = $("#aaron");**

**if (v == "1") {**

**//当前当前动画**

**$aaron.stop()**

**} else if (v == "2") {**

**//停止所以队列**

**$aaron.stop(true)**

**} else if (v == "3") {**

**//停止动画，直接跳到当前动画的结束**

**$aaron.stop(true,true)**

**}**

**});**

## 4.5 jQuery核心

本节的前三个方法都是通过$符号调用的(类似静态方法)

### 4.5.1 each()方法

jQuery中有个很重要的核心方法each，大部分jQuery方法在内部都会调用each，其主要的原因的就是jQuery的大部分方法都是针元素合集的操作，所以jQuery会提供$(selector).each()来遍历jQuery对象

如：找到所有的div，并且都设置样式，css只是一个方法，所以内部会调用each处理这个div的合集，给每个div都设置style属性

$('div').css(“color”,”red”);

**语法：**

**$.each(["Aaron", "慕课网"], function(index, value) {**

**//index是索引,也就是数组的索引**

**//value就是数组中的值了， this指针也指向value(注意！只有这里的this是指向value)**

**});**

**注意： 1.若给回调函数添加返回值false，则代表停止循环，相当于break**

**2.each()方法还可以遍历对象的字面量属性**

**eg：**

**$.each({**

**name: "张三",**

**age: 18**

**}, function(property, value) {**

**$aaron.append("属性名=" + property + ";<br> 属性值=" + value);**

**});**

**$.each(['Aaron', '慕课网'], function(i, item) {**

**$aaron.append("索引=" + i + "; 元素=" + item);**

**});**

### 4.5.2 查找索引inArray()

jQuery封装了inArray()函数判断元素是否存在数组中

**语法：**

**jQuery.inArray( value, array ,[ fromIndex ] )**

**jQuery.inArray()函数用于在数组中搜索指定的值，并返回其索引值。如果数组中不存在该值，则返回 -1。**

**value是要查找的值，array是被查找的数组，fromIndex规定查找的起始值，默认是0**

**eg：**

$.inArray(5,[1,2,3,4,5,6,7]); //返回对应的索引：4

### 4.5.3 去除空格trim()方法

页面中，通过input可以获取用户的输入值，例如常见的登录信息的提交处理。用户的输入不一定是标准的，输入一段密码：'  1123456  "，注意了： 密码的前后会留空，这可能是用户的无心的行为，但是密码确实又没错，针对这样的行为，开发者应该去除多余的空白

**jQuery.trim()函数用于去除字符串两端的空白(包括空格，制表符，换行)**

**注意：**

**该函数不会去除字符串中间的空白**

### 4.5.4 获取DOM元素get()方法

jQuery是一个合集对象，如果需要单独操作合集中的的某一个元素，可以通过**.get()**方法获取到

**语法：**

**.get( [index ] )**

**注意：**

1. **get方法是获取的dom对象，也就是通过document.getElementById获取的对象**
2. **get方法是从0开始索引**
3. **get方法还可以从后往前索引，传递一个负索引值，注意的负值的索引起始值是-1**

**eg：**

<a>1</a>

<a>2</a>

<a>3</a>

**$(“a”).get(1);//获取第二个a标签**

**$(“a”).get(-2);//获取第二个元素**

### 4.5.4 获取DOM元素在jQuery对象中的索引index()

.index()方法，从匹配的元素中搜索给定元素的索引值，从0开始计数。

**语法：**

**参数接受一个jQuery或者dom对象作为查找的条件**

**jQuery集合.index() 返回jQuery集合中第一个元素相对于同辈元素的索引**

**jQuery集合.index( JQuerySelector )**

**jQuery集合.index( DOMelement )**

eg:

<ul>

<a></a>

<a></a>

<li id="test1">1</li>

<li id="test2">2</li>

<li id="test3">3</li>

</ul>

**$("li").index()//结果：2(前面有两个a标签)**

**$("li").index(document.getElementById("test2")) //结果：1**

**$("li").index($("#test2")) //结果:1**