第二章：基础核心

一．代码风格

在 jQuery 程序中，不管是页面元素的选择、内置的功能函数，都是美元符号“$”来起始的。而这个“$”就是 jQuery 当中最重要且独有的对象：jQuery 对象，所以我们在页面元素选择或执行功能函数的时候可以这么写：

```
$(function () {}); //执行一个匿名函数
$( ‘#box’ ); //进行执行的 ID 元素选择
$( ‘#box’ ).css( ‘color’ , ‘red’ ); //执行功能函数
```

由于$本身就是 jQuery 对象的缩写形式，那么也就是说上面的三段代码也可以写成如下形式：

```
jQuery(function () {});
jQuery( ‘#box’ );
jQuery( ‘#box’ ).css( ‘color’ , ‘red’ );
```

在执行功能函数的时候，我们发现.css()这个功能函数并不是直接被“$”或 jQuery 对象调用执行的，而是先获取元素后，返回某个对象再调用.css()这个功能函数。那么也就是说，这个返回的对象其实也就是 jQuery 对象。

```
$().css( ‘color’ , ’ red’ ); //理论上合法，但实际上缺少元素而报错
```

值得一提的是，执行了.css()这个功能函数后，最终返回的还是 jQuery 对象，那么也就是说，jQuery 的代码模式是采用的连缀方式，可以不停的连续调用功能函数。

```
$('#box').css('color', 'red').css('font-size', '50px'); //连缀
```

jQuery 中代码注释和 JavaScript 是保持一致的，有两种最常用的注释：单行使用“//...” ，多行使用“/* ... */” 。

```
//$('#box').css('color', 'red');
```

二．加载模式

我们在之前的代码一直在使用$(function () {});这段代码进行首尾包裹，那么为什么必须要包裹这段代码呢？原因是我们 jQuery 库文件是在 body 元素之前加载的，我们必须等待所有的 DOM 元素加载后，延迟支持 DOM 操作，否则就无法获取到。

在延迟等待加载，JavaScript 提供了一个事件为 load，方法如下：

```
window.onload = function () {}; //JavaScript 等待加载
$(document).ready(function () {}); //jQuery 等待加载
```

load和ready区别

	window.onload	$(document).ready()
执行时机	必须等待网页全部加载完毕（包括图片等），然后再执行包裹代码	只需要等待网页中的DOM结构加载完毕，就能执行包裹的代码
执行次数	只能执行一次，如果第二次，那么第一次的执行会被覆盖	可以执行多次，第N次都不会被上一次覆盖
简写方案	无	$(function () { });

在实际应用中，我们都很少直接去使用 window.onload，因为他需要等待图片之类的大型元素加载完毕后才能执行 JS 代码。所以，最头疼的就是网速较慢的情况下，页面已经全面展开，图片还在缓慢加载，这时页面上任何的 JS 交互功能全部处在假死状态。并且只能执行单次在多次开发和团队开发中会带来困难。

执行次数实例：

```
window.onload = function () {
    alert(1);
}
window.onload = function () {
    alert(2);
}
$(document).ready(function () {
    alert(1);
});
$(document).ready(function () {
    alert(2);
});
```

三．对象互换

jQuery 对象虽然是 jQuery 库独有的对象，但它也是通过 JavaScript 进行封装而来的。 我们可以直接输出来得到它的信息。

```
alert($); //jQuery 对象方法内部
alert($()); //jQuery 对象返回的对象，还是 jQuery
alert($('#box')); //包裹 ID 元素返回对象，还是 jQuery
```

从上面三组代码我们发现：只要使用了包裹后，最终返回的都是 jQuery 对象。这样的好处显而易见，就是可以连缀处理。但有时，我们也需要返回原生的 DOM 对象，比如：

```
alert(document.getElementById('box')); //[object HTMLDivElement]
```

jQuery 想要达到获取原生的 DOM 对象，可以这么处理：

```
alert($('#box').get(0)); //ID 元素的第一个原生 DOM
```

从上面 get(0)，这里的索引看出，jQuery 是可以进行批量处理 DOM 的，这样可以在很多需要循环遍历的处理上更加得心应手。

四. 多个库之间的冲突

当一个项目中引入多个第三方库的时候，由于没有命名空间的约束（命名空间就好比同一个目录下的文件夹一样，名字相同就会产生冲突），库与库之间发生冲突在所难免。那么，既然有冲突的问题，为什么要使用多个库呢？原因是 jQuery 只不过是 DOM 操作为主的库，方便我们日常 Web 开发。但有时，我们的项目有更多特殊的功能需要引入其他的库，比如用户界面 UI 方面的库，游戏引擎方面的库等等一系列。

而很多库，比如 prototype、还有我们 JavaScript 课程开发的 Base 库，都使用“$”作为基准起始符，如果想和 jQuery 共容有两种方法：

1.将 jQuery 库在 Base 库之前引入，那么“$”的所有权就归 Base 库所有，而 jQuery 可以直接用 jQuery 对象调用，或者创建一个“$$”符给 jQuery 使用。

```
var $$ = jQuery; //创建一个$$的 jQuery 对象
$(function () { //这是 Base 的$
    alert($('#box').ge(0)); //这是 Base 的$
    alert($$('#box').width()); //这是 jQuery 的$$
});
```

2.如果将 jQuery 库在 Base 库之后引入，那么“$”的所有权就归 jQuery 库所有，而 Base 库将会冲突而失去作用。这里，jQuery 提供了一个方法：

```
jQuery.noConflict(); //将$符所有权剔除
var $$ = jQuery;
$(function () {
```

```
    alert($('#box').ge(0));
    alert($$('#box').width());
});
```

第三章：常规选择器

一. 简单选择器

在使用 jQuery 选择器时，我们首先必须使用“$()”函数来包装我们的 CSS 规则。而 CSS 规则作为参数传递到 jQuery 对象内部后，再返回包含页面中对应元素的 jQuery 对象。随后，我们就可以对这个获取到的 DOM 节点进行行为操作了。

```
#box { //使用 ID 选择器的 CSS 规则
    color:red; //将 ID 为 box 的元素字体颜色变红
}
```

在 jQuery 选择器里，我们使用如下的方式获取同样的结果：

```
$('#box').css('color', 'red'); //获取 DOM 节点对象，并添加行为
```

那么除了 ID 选择器之外，还有两种基本的选择器，分别为：元素标签名和类(class)：

选择器	CSS 模式	jQuery 模式	描述
元素名	div {}	$('div')	获取所有 div 元素的 DOM 对象
ID	#box {}	$('#box')	获取一个 ID 为 box 元素的 DOM 对象
类(class)	.box{}	$('.box')	获取所有 class 为 box 的所有 DOM 对象

```
$('div').css('color', 'red'); //元素选择器，返回多个元素
$('#box').css('color', 'red'); //ID 选择器，返回单个元素
$('.box').css('color', 'red'); //类(class)选择器，返回多个元素
```

为了证明 ID 返回的是单个元素，而元素标签名和类(class)返回的是多个，我们可以采用 jQuery 核心自带的一个属性 length 或 size()方法来查看返回的元素个数。原因：在DOM 元素中ID是唯一的

```
alert($('div').size()); //3 个
alert($('#box').size()); //1 个，后面两个失效了
```

```
alert($('.box').size()); //3 个
```

同理，你也可以直接使用 jQuery 核心属性来操作：

```
alert($('#box').length); //1 个，后面失效了
```

警告：有个问题特别要注意，ID 在页面只允许出现一次，我们一般都是要求开发者要遵守和保持这个规则。但如果你在页面中出现三次，并且在 CSS 使用样式，那么这三个元素还会执行效果。但如果，你想在 jQuery 这么去做，那么就会遇到失效的问题。所以，开发者必须养成良好的遵守习惯，在一个页面仅使用一个 ID。

```
$('#box').css('color', 'red'); //只有第一个 ID 变红，后面两个失效
```

jQuery 选择器的写法与 CSS 选择器十分类似，只不过他们的功能不同。CSS 找到元素后添加的是单一的样式,而 jQuery 则添加的是动作行为。最重要的一点是:CSS 在添加样式的时候,高级选择器会对部分浏览器不兼容,而 jQuery 选择器在添加 CSS 样式的时候却不必为此烦恼。

```
#box > p { //CSS 子选择器，IE6 不支持    （区别 #box p 选择器会包括孙子节点）
    color:red;
}
$('#box > p').css('color','red'); //jQuery 子选择器，兼容了 IE6
```

jQuery 选择器支持 CSS1、CSS2 的全部规则，支持 CSS3 部分实用的规则，同时它还有少量独有的规则。所以，对于已经掌握 CSS 的开发人员，学习 jQuery 选择器几乎是零成本。而 jQuery 选择器在获取节点对象的时候不但简单， 还内置了容错功能， 这样避免像 JavaScript 那样每次对节点的获取需要进行有效判断。

```
$('#pox').css('color', 'red'); //不存在 ID 为 pox 的元素，也不报错（容错）
document.getElementById('pox').style.color = 'red'; //报错了
```

因为 jQuery 内部进行了判断，而原生的 DOM 节点获取方法并没有进行判断，所以导致了一个错误，原生方法可以这么判断解决这个问题：

```
if (document.getElementById('pox')) { //先判断是否存在这个对象
    document.getElementById('pox').style.color = 'red';
}
```

那么对于缺失不存在的元素，我们使用 jQuery 调用的话，怎么去判断是否存在呢？因为本身返回的是 jQuery 对象，可能会导致不存在元素存在与否，都会返回 true。

```
if ($('#pox').length > 0) { //判断元素包含数量即可
    $('#pox').css('color', 'red');
}
```

除了这种方式之外，还可以用转换为 DOM 对象的方式来判断，例如：

```
if ($('#pox').get(0)) {}
```

二．进阶选择器

在简单选择器中，我们了解了最基本的三种选择器：元素标签名、ID 和类(class)。那么在基础选择器外，还有一些进阶和高级的选择器方便我们更精准的选择元素。

选择器	CSS 模式	jQuery 模式	描述
群组选择器	span,em,.box {}	$('span,em,.box')	获取多个选择器的 DOM 对象
后代选择器	ul li a {}	$('ul li a')	获取追溯到的多个 DOM 对象
通配选择器	* {}	$('*')	获取所有元素标签的 DOM 对象

```
//群组选择器
span, em, .box { //多种选择器添加红色字体
    color:red;
}
$('span, em, .box').css('color', 'red'); //群组选择器 jQuery 方式
//后代选择器
ul li a { //层层追溯到的元素添加红色字体
color:red;
}
$('ul li a').css('color', 'red'); //群组选择器 jQuery 方式
//通配选择器
* { //页面所有元素都添加红色字体
    color:red;
}
$('*').css('color', 'red'); //通配选择器
```

目前介绍的六种选择器，在实际应用中，我们可以灵活的搭配，使得选择器更加的

精准和快速：

```
$('#box p, ul li *').css('color', 'red'); //组合了多种选择器
```

警告：在实际使用上，通配选择器一般用的并不多，尤其是在大通配上，比如：$('*')，这种使用方法效率很低，影响性能，建议竟可能少用。还有一种选择器，可以在 ID 和类(class)中指明元素前缀，比如：

```
$('div.box'); //限定必须是.box 元素获取必须是 div
$('p#box div.side'); //同上
```

类(class)有一个特殊的模式，就是同一个 DOM 节点可以声明多个类(class)。那么对于这种格式，我们有多 class 选择器可以使用，但要注意和 class 群组选择器的区别。

```
.box.pox { //双 class 选择器，IE6 出现异常
    color:red;
}
$('.box.pox').css('color', 'red'); //兼容 IE6，解决了异常
```

多 class 选择器是必须一个 DOM 节点同时有多个 class，用这多个 class 进行精确限定。而群组 class 选择器，只不过是多个 class 进行选择而已。

```
$('.box, .pox').css('color', 'red'); //加了逗号，体会区别
```

警告：在构造选择器时，有一个通用的优化原则：只追求必要的确定性。当选择器筛选越复杂，jQuery 内部的选择器引擎处理字符串的时间就越长。比如：

```
$('div#box ul li a#link'); //让 jQuery 内部处理了不必要的字符串
$('#link'); //ID 是唯一性的，准确度不变，性能提升
```

三．高级选择器

在前面我们学习六种最常规的选择器， 一般来说通过这六种选择器基本上可以解决所有 DOM 节点对象选择的问题。但在很多特殊的元素上，比如父子关系的元素，兄弟关系的元素，特殊属性的元素等等。在早期 CSS 的使用上，由于 IE6 等低版本浏览器不支持，所以这些高级选择器的使用也不具备普遍性，但随着 jQuery 兼容，这些选择器的使用频率也越来越高。

层次选择器

选择器	CSS 模式	jQuery 模式	描述
后代选择器	ul li a {}	$('ul li a')	获取追溯到的多个 DOM 对象
子选择器	div > p {}	$('div p')	只获取子类节点的多个 DOM 对象
next 选择器	div + p {}	$('div + p')	只获取某节点后一个同级 DOM 对象
nextAll 选择器	div ~ p {}	$('div ~ p')	获取某节点后面所有同级 DOM 对象

在层次选择器中， 除了后代选择器之外， 其他三种高级选择器是不支持 IE6 的，而 jQuery 却是兼容 IE6 的。

```
//后代选择器
$('#box p').css('color', 'red'); //全兼容
```

jQuery 为后代选择器提供了一个等价 find()方法

```
$('#box').find('p').css('color', 'red'); //和后代选择器等价
//子选择器，孙子后失效
#box > p { //IE6 不支持
    color:red;
}
$('#box > p').css('color', 'red'); //兼容 IE6
```

jQuery 为子选择器提供了一个等价 children()方法：

```
$('#box').children('p').css('color', 'red'); //和子选择器等价
```

选择器 CSS 模式 jQuery 模式 描述

后代选择器 ul li a {} $('ul li a') 获取追溯到的多个 DOM 对象

子选择器 div > p {} $('div p') 只获取子类节点的多个 DOM 对象

next 选择器 div + p {} $('div + p') 只获取某节点后一个同级 DOM 对象

nextAll 选择器 div ~ p {} $('div ~ p') 获取某节点后面所有同级 DOM 对象

```
//next 选择器(下一个同级节点)
#box + p { //IE6 不支持
    color:red;
}
$('#box+p').css('color', 'red'); //兼容 IE6
```

jQuery 为 next 选择器提供了一个等价的方法 next()：

```
$('#box').next('p').css('color', 'red'); //和 next 选择器等价
```

```
//nextAll 选择器(后面所有同级节点)
#box ~ p { //IE6 不支持
    color:red;
}
$('#box ~ p').css('color', 'red'); //兼容 IE6
```

jQuery 为 nextAll 选择器提供了一个等价的方法 nextAll()：

```
$('#box').nextAll('p').css('color', 'red'); //和 nextAll 选择器等价
```

层次选择器对节点的层次都是有要求的，比如子选择器，只有子节点才可以被选择到，孙子节点和重孙子节点都无法选择到。next 和 nextAll 选择器，必须是同一个层次的后一个和后 N 个，不在同一个层次就无法选取到了。

在 find()、next()、nextAll()和 children()这四个方法中，如果不传递参数，就相当于传递了“*” ，即任何节点，我们不建议这么做，不但影响性能，而且由于精准度不佳可能在复杂的 HTML 结构时产生怪异的结果。

```
$('#box').next(); //相当于$('#box').next('*');
```

为了补充高级选择器的这三种模式，jQuery 还提供了更加丰富的方法来选择元素：

```
$('#box').prev('p').css('color', 'red'); //同级上一个元素
$('#box').prevAll('p').css('color', 'red'); //同级所有上面的元素
```

nextUntil()和 prevUnitl()方法是选定同级的下面或上面的所有节点，选定非指定的所有元素，一旦遇到指定的元素就停止选定。

```
//同级上非指定元素选定，遇到则停止
$('#box').prevUntil('p').css('color', 'red');
//同级下非指定元素选定，遇到则停止
$('#box').nextUntil('p').css('color', 'red');
```

siblings()方法正好集成了 prevAll()和 nextAll()两个功能的效果，及上下相邻的所有元素进行选定：

```
$('#box').siblings('p').css('color', 'red'); //同级上下所有元素选定
```

//等价于下面：

```
$('#box').prevAll('p').css('color', 'red'); //同级上所有元素选定
$('#box').nextAll('p').css('color', 'red'); //同级下所有元素选定
```

警告：切不可写成“$('#box').prevAll('p').nextAll('p').css('color', 'red');”这种形式，因为 prevAll('p')返回的是已经上方所有指定元素，然后再 nextAll('p')选定下方所有指定元素，这样必然出现错误。

理论上来讲，jQuery 提供的方法 find()、next()、nextAll()和 children()运行速度要快于使用高级选择器。因为他们实现的算法有所不同，高级选择器是通过解析字符串来获取节点对象，而 jQuery 提供的方法一般都是单个选择器，是可以直接获取的。但这种快慢的差异，对于客户端脚本来说没有太大的实用性， 并且速度的差异还要取决了浏览器和选择的元素内容。比如，在 IE6/7 不支持 querySelectorAll()方法，则会使用“Sizzle”引擎，速度就会慢，而其他浏览器则会很快。有兴趣的可以了解这个方法和这个引擎。

选择器快慢分析：

```
// 这条最快，会使用原生的 getElementById、ByName、ByTagName 和 querySelectorAll()
$('#box').find('p');
//jQuery 会自动把这条语句转成$('#box').find('p')，这会导致一定的性能损失。它比最快的形式慢了 5%-10%
$('p', '#box');
//这条语句在 jQuery 内部，会使用$.sibling()和 javascript 的 nextSibling()方法，一个个遍历节点。它比最快的形式大约慢 50%
$('#box').children('p');
//jQuery 内部使用 Sizzle 引擎，处理各种选择器。Sizzle 引擎的选择顺序是从右到左，所以这条语句是先选 p，然后再一个个过滤出父元素#box，这导致它比最快的形式大约慢 70%
$('#box > p');
//这条语句与上一条是同样的情况。但是，上一条只选择直接的子元素，这一条可以于选择多级子元素，所以它的速度更慢，大概比最快的形式慢了 77%。
$('#box p');
```

//jQuery 内部会将这条语句转成$('#box').find('p')，比最快的形式慢了 23%。

```
$('p', $('#parent'));
```

综上所属，最快的是 find()方法，最慢的是$('#box p')这种高级选择器。如果一开始将$('#box')进行赋值，那么 jQuery 就对其变量进行缓存，那么速度会进一步提高。

```
var box = $('#box');
var p = box.find('p');
```

注意：我们应该推荐使用哪种方案呢？其实，使用哪种都差不多。这里，我们推荐使用 jQuery 提供的方法。因为不但方法的速度比高级选择器运行的更快，并且它的灵活性和扩展性要高于高级选择器。使用“+”或“~”从字面上没有 next 和 nextAll 更加语义化，更加清晰，jQuery 的方法更加丰富，提供了相对的 prev 和 prevAll。毕竟 jQuery 是编程语言，需要能够灵活的拆分和组合选择器，而使用 CSS 模式过于死板。所以，如果 jQuery 提供了独立的方法来代替某些选择器的功能，我们还是推荐优先使用独立的方法。

属性选择器

CSS 模式	jQuery 模式	描述
a[title]	$('a[title]')	获取具有这个属性的 DOM 对象
a[title=num1]	$('a[title=num1]')	获取具有这个属性=这个属性值的 DOM 对象
a[title^=num]	$('a[title^=num]')	获取具有这个属性且开头属性值匹配的 DOM 对象
a[title|=num]	$('a[title|=num]')	获取具有这个属性且等于属性值或开头属性值匹配后面跟一个“-”号的 DOM 对象
a[title$=num]	$('a[title$=num]')	获取具有这个属性且结尾属性值匹配的 DOM 对象
a[title!=num]	$('a[title!=num]')	获取具有这个属性且不等于属性值的 DOM 对象
a[title~=num]	$('a[title~=num]')	获取具有这个属性且属性值是以一个空格分割的列表，其中包含属性值的 DOM 对象
a[title*=num]	$('a[title*=num]')	获取具有这个属性且属性值含有一个指定字串的 DOM 对象
a[bbb][title=num1]	$('a[bbb][title=num1]')	获取具有这个属性且属性值匹配的 DOM 对象

属性选择器也不支持 IE6， 所以在 CSS 界如果要兼容低版本， 那么也是非主流。 但 jQuery 却不必考虑这个问题。

```
//选定这个属性的
a[title] { //IE6 不支持
    color:red;
}
$('a[title]').css('color', 'red'); //兼容 IE6 了
    //选定具有这个属性=这个属性值的
    a[title=num1] { //IE6 不支持
    color:red;
}
$('a[title=num1]').css('color', 'red'); //兼容 IE6 了
//选定具有这个属性且开头属性值匹配的
a[title^=num] { //IE6 不支持
    color:red;
}
$('a[title=^num]').css('color', 'red'); //兼容 IE6 了
//选定具有这个属性且等于属性值或开头属性值匹配后面跟一个“-”号
a[title|=num] { //IE6 不支持
    color:red;
}
$('a[title|="num"]').css('color', 'red'); //兼容 IE6 了
//选定具有这个属性且结尾属性值匹配的
a[title$=num] { //IE6 不支持
    color:red;
}
$('a[title$=num]').css('color','red'); //兼容 IE6 了
//选定具有这个属性且属性值不想等的
a[title!=num1] { //不支持此 CSS 选择器
    color:red;
}
$('a[title!=num1]').css('color','red'); //jQuery 支持这种写法
```

```
//选定具有这个属性且属性值是以一个空格分割的列表，其中包含属性值的
a[title~=num] { //IE6 不支持
    color:red;
}
$('a[title~=num1]').css('color','red'); //兼容 IE6
//选定具有这个属性且属性值含有一个指定字串的
a[title*=num] { //IE6 不支持
    color:red;
}
$('a[title*=num]').css('color','red'); //兼容 IE6
//选定具有多个属性且属性值匹配成功的
a[bbb][title=num1] { //IE6 不支持
    color:red;
}
$('a[bbb][title=num1]').css('color','red'); //兼容 IE6
```

第四章过滤选择器

过滤选择器简称：过滤器。它其实也是一种选择器，而这种选择器类似与 CSS3 (http://t.mb5u.com/css3/)里的伪类，可以让不支持 CSS3 的低版本浏览器也能支持。和常规选择器一样，jQuery 为了更方便开发者使用，提供了很多独有的过滤器。

一. 基本过滤器

过滤器主要通过特定的过滤规则来筛选所需的 DOM 元素，和 CSS 中的伪类的语法类似：使用冒号(:)开头。

过滤器名	jQuery 语法	说明	返回
:first	$('li:first')	选取第一个元素	单个元素
:last	$('li:last')	选取最后一个元素	单个元素
:not(selector)	$('li:not(.red)')	选取 class 不是 red 的 li 元素	集合元素
:even	$('li.even')	选择索引(0 开始)是偶数的所有元素	集合元素
:odd	$('li:odd')	选择索引(0 开始)是奇数的所有元素	集合元素
:eq(index)	$('li:eq(2)')	选择索引(0 开始)等于 index 的元素	单个元素
:gt(index)	$('li:gt(2)')	选择索引(0 开始)大于 index 的元素	集合元素
:lt(index)	$('li.lt(2)')	选择索引(0 开始)小于 index 的元素	集合元素
:header	$(':header')	选择标题元素，h1～h6	集合元素
:animated	$(':animated')	选择正在执行动画的元素	集合元素
:focus	$(':focus')	选择当前被焦点的元素	集合元素

```
$('li:first').css('background', '#ccc'); //第一个元素
$('li:last).css('background', '#ccc'); //最后一个元素
$('li:not(.red)).css('background', '#ccc'); //非 class 为 red 的元素
$('li:even').css('background', '#ccc'); //索引为偶数的元素
$('li:odd).css('background', '#ccc'); //索引为奇数的元素
$('li:eq(2)).css('background', '#ccc'); //指定索引值的元素
$('li:gt(2)').css('background', '#ccc'); //大于索引值的元素
$('li:lt(2)').css('background', '#ccc'); //小于索引值的元素
$(':header').css('background', '#ccc'); //页面所有 h1 ~ h6 元素
```

注意： :focus 过滤器，必须是网页初始状态的已经被激活焦点的元素才能实现元素获取。而不是鼠标点击或者 Tab 键盘敲击激活的。

```
$('input').get(0).focus(); //先初始化激活一个元素焦点
$(':focus').css('background', 'red'); //被焦点的元素
```

jQuery 为最常用的过滤器提供了专用的方法，已达到提到性能和效率的作用：

```
//元素 li 的第三个元素，负数从后开始
$('li').eq(2).css('background', '#ccc');
$('li').first().css('background', '#ccc'); //元素 li 的第一个元素
$('li').last().css('background', '#ccc'); //元素 li 的最后一个元素
$('li').not('.red').css('background', '#ccc'); //元素 li 不含 class 为 red
```

的元素

注意：:first、:last 和 first()、last()这两组过滤器和方法在出现相同元素的时候，first 会实现第一个父元素的第一个子元素，last 会实现最后一个父元素的最后一个子元素。所以，如果需要明确是哪个父元素，需要指明：

```
$('#box li:last').css('background', '#ccc'); //#box 元素的最后一个 li
//或
$('#box li).last().css('background', '#ccc'); //同上
```

二．内容过滤器

过滤器名	jQuery 语法	说明	返回
:contains(text)	$(':contains("ycku .com")')	选取含有"ycku.com"文本的元素	元素集合
:empty	$(':empty')	选取不包含子元素或空文本的元素	元素集合
:has(selector)	$(':has(.red)')	选取含有 class 是 red 的元素	元素集合
:parent	$(':parent')	选取含有子元素或文本的元素	元素集合

```
//选择元素文本节点含有 ycku.com 文本的元素
$('div:contains("ycku.com")').css('background', '#ccc');
$('div:empty').css('background', '#ccc'); //选择空元素
//选择子元素含有 class 是 red 的元素
$('ul:has(.red)').css('background', '#ccc');
$(':parent').css('background', '#ccc'); //选择非空元素
```

jQuery 提供了一个 has()方法来提高:has 过滤器的性能：

```
//选择子元素含有 class 是 red 的元素
$('ul').has('.red').css('background', '#ccc');
```

jQuery 提供了一个名称和:parent 相似的方法， 但这个方法并不是选取含有子元素或文本的元素，而是获取当前元素的父元素，返回的是元素集合。

```
$('li').parent().css('background', '#ccc'); //选择当前元素的父元素
$('li').parents().css('background', '#ccc');
//选择当前元素的父元素及祖先元素
```

```
$('li').parentsUntil('div').css('background', '#ccc');
```

//选择当前元素遇到 div 父元素停止

三．可见性过滤器

可见性过滤器根据元素的可见性和不可见性来选择相应的元素。

过滤器名	jQuery 语法	说明	返回
:hidden	$(':hidden')	选取所有不可见元素	集合元素
:visible	$(':visible')	选取所有可见元素	集合元素

```
$('p:hidden).size(); //元素 p 隐藏的元素
$('p:visible').size(); //元素 p 显示的元素
```

注意：:hidden 过滤器一般是包含的内容为：CSS 样式为 display:none、input 表单类型为 type="hidden"和 visibility:hidden 的元素。

四．子元素过滤器

子元素过滤器的过滤规则是通过父元素和子元素的关系来获取相应的元素。

过滤器名	jQuery 语法	说明	返回
:first-child	$('li:first-child')	获取每个父元素的第一个子元素	集合元素
:last-child	$('li:last-child')	获取每个父元素的最后一个子元素	集合元素
:only-child	$('li:only-child')	获取只有一个子元素的元素	集合元素
:nth-child(odd/even/eq(index))	$('li:nth-child(even)')	获取每个自定义子元素的元素	集合元素

```
$('li:first-child').css('background', '#ccc'); //每个父元素第一个 li 元素
$('li:last-child').css('background', '#ccc'); //每个父元素最后一个 li 元素
$('li:only-child').css('background', '#ccc'); //每个父元素只有一个 li 元素
$('li:nth-child(odd)').css('background', '#ccc'); //每个父元素奇数 li 元素
$('li:nth-child(even)').css('background', '#ccc'); //每个父元素偶数 li 元素
$('li:nth-child(2)').css('background', '#ccc'); //每个父元素第三个 li 元素
```

五．其他方法

jQuery 在选择器和过滤器上，还提供了一些常用的方法，方便我们开发时灵活使用。

方法名	jQuery 语法	说明	返回
is(s/o/e/f)	$('li').is('.red')	传递选择器、DOM、jquery 对象	集合元素
		或是函数来匹配元素结合	
hasClass(class)	$('li').eq(2).hasClass('red')	其实就是 is("." + class)	集合元素
slice(start, end)	$('li').slice(0,2)	选择从 start 到 end 位置的元素，如果是负数，则从后开始	集合元素
filter(s/o/e/f)	$('li').filter('.red')		
end()	$('div').find('p').end()	获取当前元素前一次状态	集合元素
contents()	$('div').contents()	获取某元素下面所有元素节点，包括文本节点，如果是 iframe，则可以查找文本内容	集合元素

```
$('.red').is('li'); //true，选择器，检测 class 为是否为 red
$('.red').is($('li')); //true，jQuery 对象集合，同上
$('.red').is($('li').eq(2)); //true，jQuery 对象单个，同上
$('.red').is($('li').get(2)); //true，DOM 对象，同上
$('.red').is(function () { //true，方法，同上
return $(this).attr('title') == '列表 3'; //可以自定义各种判断
}));
$('li').eq(2).hasClass('red'); //和 is 一样，只不过只能传递 class
$('li').slice(0,2).css('color', 'red'); //前三个变成红色
```

注意：这个参数有多种传法和 JavaScript 的 slice 方法是一样的比如：slice(2)，从第三个开始到最后选定；slice(2,4)，第三和第四被选定；slice(0,-2)，从倒数第三个位置，向前选定所有；slice(2,-2)，前两个和末尾两个未选定。

```
$("div").find("p").end().get(0); //返回 div 的原生 DOM
$('div').contents().size(); //返回子节点(包括文本)数量
//选择 li 的 class 为 red 的元素
$('li').filter('.red').css('background','#ccc');
```

```
//增加了首尾选择
$('li').filter('.red, :first, :last').css('background','#ccc');
 //特殊要求函数返回
$('li').filter(function () {
return $(this).attr('class') == 'red' && $(this).attr('title') == '列表 3';
}).css('background', '#ccc');
```

第五章：基础 DOM 和 CSS 操作

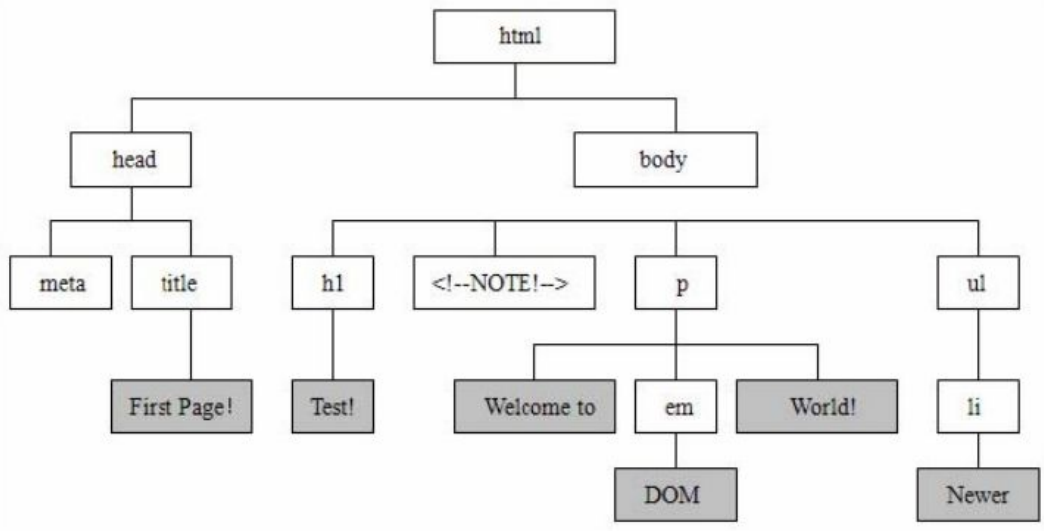
一. DOM 简介

由于课程是基于 JavaScript 基础上完成的， 这里我们不去详细的了解 DOM 到底是什么。只需要知道几个基本概念：

1.D 表示的是页面文档 Document、O 表示对象，即一组含有独立特性的数据集合、M 表示模型，即页面上的元素节点和文本节点。

2.DOM 有三种形式，标准 DOM、HTML DOM、CSS DOM，大部分都进行了一系列的封装，在 jQuery 中并不需要深刻理解它。

3.树形结构用来表示 DOM，就非常的贴切，大部分操作都是元素节点操作，还有少部分是文本节点操作。



二．设置元素及内容

我们通过前面所学习的各种选择器、过滤器来得到我们想要操作的元素。这个时候，我们就可以对这些元素进行 DOM 的操作。 那么， 最常用的操作就是对元素内容的获取和修改。

html()和 text()方法

方法名	描述
html()	获取元素中 HTML 内容
html(value)	设置元素中 HTML 内容
text()	获取元素中文本内容
text(value)	设置原生中文本内容
val()	获取表单中的文本内容
val(value)	设置表单中的文本内容

在常规的 DOM 元素中，我们可以使用 html()和 text()方法获取内部的数据。html()方法可以获取或设置 html 内容，text()可以获取或设置文本内容。

```
$('#box').html(); //获取 html 内容
$('#box').text(); //获取文本内容，会自动清理 html 标签
$('#box').html('<em>www.li.cc</em>'); //设置 html 内容
$('#box').text('<em>www.li.cc</em>'); //设置文本内容，会自动转义 html 标签
```

注意：当我们使用 html()或 text()设置元素里的内容时，会清空原来的数据。而我们期望能够追加数据的话，需要先获取原本的数据。

```
$('#box').html($('#box').html() + '<em>www.li.cc</em>'); //追加数据
```

如果元素是表单的话，jQuery 提供了 val()方法进行获取或设置内部的文本数据。

```
$('input').val(); //获取表单内容
$('input').val('www.li.cc'); //设置表单内容
```

如果想设置多个选项的选定状态，比如下拉列表、单选复选框等等，可以通过数组传递操作。

```
$("input").val(["check1","check2", "radio1" ]); //value
```

三．元素属性操作

除了对元素内容进行设置和获取，通过 jQuery 也可以对元素本身的属性进行操作，包括获取属性的属性值、设置属性的属性值，并且可以删除掉属性。

attr()和 removeAttr()

方法名	描述
attr(key)	获取某个元素 key 属性的属性值
attr(key, value)	设置某个元素 key 属性的属性值
attr({key1:value2, key2:value2...})	设置某个元素多个 key 属性的属性值
attr(key, function (index, value) {})	设置某个元素 key 通过 fn 来设置

```
$('div').attr('title'); //获取属性的属性值
$('div').attr('title', '我是域名'); //设置属性及属性值
$('div').attr('title', function () { //通过匿名函数返回属性值
    return '我是域名';
});
$('div').attr('title', function (index) { //index 是选中对象的集合的索引值
        return '我是'+index+'域名';
});
$('div').attr('title', function (index, value) { //可以接受两个参数
    return value + (index+1) + '，我是域名';//value 是属性原本的值
});
```

注意：attr()方法里的 function() {}，可以不传参数。可以只传一个参数 index，表示当前元素的索引(从 0 开始)。也可以传递两个参数 index、value，第二个参数表示属性原本的值。

注意：jQuery 中很多方法都可以使用 function() {}来返回出字符串，比如 html()、text()、val()和上一章刚学过的 is()、filter()方法。而如果又涉及到多个元素集合的话，还可以传递 index 参数来获取索引值，并且可以使用第二个参数 value(并不是所有方法都适合，有兴趣可以自己逐个尝试)。

```
$('div').html(function (index) { //通过匿名函数赋值，并传递 index
    return '我是' + (index+1) + '号 div';
});
```

```
$('div').html(function (index, value) { //还可以实现追加内容
    return '我是' + (index+1) + '号 div：'+value ;
});
```

注意：我们也可以使用 attr()来创建 id 属性，但我们强烈不建议这么做。这样会导致整个页面结构的混乱。 当然也可以创建 class 属性， 但后面会有一个语义更好的方法来代替 attr()方法，所以也不建议使用。

删除指定的属性，这个方法就不可以使用匿名函数，传递 index 和 value 均无效。

```
$('div').removeAttr('title'); //删除指定的属性
$('div').removeAttr(function () {          //不可以传递 function
    return 'title';
});
```

四. 元素样式操作

元素样式操作包括了直接设置 CSS 样式、增加 CSS 类别、类别切换、删除类别这几种操作方法。而在整个 jQuery 使用频率上来看，CSS 样式的操作也是极高的，所以需要重点掌握。

CSS 操作方法

方法名	描述
css(name)	获取某个元素行内的 CSS 样式
css([name1, name2, name3])	获取某个元素行内多个 CSS 样式
css(name, value)	设置某个元素行内的 CSS 样式
css(name, function (index, value))	设置某个元素行内的 CSS 样式

css({name1 : value1, name2 : value2})	设置某个元素行内多个 CSS 样式
addClass(class)	给某个元素添加一个 CSS 类
addClass(class1 class2 class3...)	给某个元素添加多个 CSS 类
removeClass(class)	删除某个元素的一个 CSS 类
removeClass(class1 class2 class3...)	删除某个元素的多个 CSS 类
toggleClass(class)	来回切换默认样式和指定样式
toggleClass(class1 class2 class3...)	同上
toggleClass(class, switch)	来回切换样式的时候设置切换频率
toggleClass(function () {})	通过匿名函数设置切换的规则
toggleClass(function () {}, switch)	在匿名函数设置时也可以设置频率
toggleClass(function (i, c, s) {}, switch)	在匿名函数设置时传递三个参数

```
$('div').css('color'); //获取元素行内 CSS 样式的颜色
$('div').css('color', 'red'); //设置元素行内 CSS 样式颜色为红色
```

在 CSS 获取上，我们也可以获取多个 CSS 样式，而获取到的是一个对象数组，如果用传统方式进行解析需要使用 for in 遍历。

```
//得到多个 CSS 样式的数组对象
var box = $('div').css(['color', 'height', 'width']);
for (var i in box) { //逐个遍历出来
    alert(i + ':' + box[i]);
}
```

jQuery 提供了一个遍历工具专门来处理这种对象数组，$.each()方法，这个方法可以轻松的遍历对象数组。

```
$.each(box, function (index, value) { //遍历 JavaScript 原生态的对象数组
    alert(index + ':' + value);
});
```

使用$.each()可以遍历原生的 JavaScript 对象数组，如果是 jQuery 对象的数组怎么使用.each()方法呢？

```
$('div').each(function (index, element) { //index 为索引，element 为元素 DOM
    alert(index + ':' + element);
});
```

在需要设置多个样式的时候，我们可以传递多个 CSS 样式的键值对（对象）即可。

```
$('div').css({
    'background-color' : '#ccc',
    'color' : 'red',
    'font-size' : '20px'
});
```

如果想设置某个元素的 CSS 样式的值，但这个值需要计算我们可以传递一个匿名函数。

```
$('div').css('width', function (index, value) {
    return (parseInt(value) - 500) + 'px'; //局部操作
});
```

除了行内 CSS 设置，我们也可以直接给元素添加 CSS 类，可以添加单个或多个，并且也可以删除它。

```
$('div').addClass('red'); //添加一个 CSS 类
$('div').addClass('red bg'); //添加多个 CSS 类
$('div').removeClass('bg'); //删除一个 CSS 类
$('div').removeClass('red bg'); //删除多个 CSS 类
```

我们还可以结合事件来实现 CSS 类的样式切换功能。

```
$('div').click(function () { //当点击后触发
    $(this).toggleClass('red size'); //单个样式多个样式均可
});
```

.toggleClass()方法的第二个参数可以传入一个布尔值， true 表示执行切换到 class 类， false 表示执行回默认 class 类(默认的是空 class)，运用这个特性，我

们可以设置切换的频率。

```
var count = 0;
$('div').click(function () { //每点击两次切换一次 red
    $(this).toggleClass('red', count++ % 3 == 0);
});
```

默认的 CSS 类切换只能是无样式和指定样式之间的切换，如果想实现样式 1 和样式 2 之间的切换还必须自己写一些逻辑。

```
$('div').click(function () {
    $(this).toggleClass('red size'); //一开始切换到样式 2
    if ($(this).hasClass('red')) { //判断样式 2 存在后
        $(this).removeClass('blue'); //删除样式 1
    } else {
        $(this).toggleClass('blue'); //添加样式 1，这里也可以 addClass
    }
});
```

上面的方法较为繁琐，.toggleClass()方法提供了传递匿名函数的方式，来设置你所需要切换的规则。

```
$('div').click(function () {
    $(this).toggleClass(function () { //传递匿名函数，返回要切换的样式
        return $(this).hasClass('red') ? 'blue' : 'red size';
    });
});
```

注意：上面虽然一句话实现了这个功能，但还是有一些小缺陷，因为原来的 class 类没有被删除，只不过被替代了而已。所以，需要改写一下。

```
$('div').click(function () {
    $(this).toggleClass(function () {
        if ($(this).hasClass('red')) {
            $(this).removeClass('red');
            return 'green';
```

```
        } else {
            $(this).removeClass('green');
            return 'red';
        }
    });
});
```

我们也可以在传递匿名函数的模式下，增加第二个频率参数。

```
var count = 0;
$('div').click(function () {
    $(this).toggleClass(function () {
        return $(this).hasClass('red') ? 'blue' : 'red size';
    },count++ % 3 == 0); //增加了频率
});
```

对于.toggleClass()传入匿名函数的方法，还可以传递 index 索引、class 类两个参数以及频率布尔值，可以得到当前的索引、class 类名和频率布尔值。

```
var count = 0;
$('div').click(function () {
    $(this).toggleClass(function (index, className, switchBool) {
        alert(index + ':' + className + ':' + switchBool); //得到当前值
        return $(this).hasClass('red') ? 'blue' : 'red size';
    },count++ % 3 == 0);
});
```

五. CSS 方法

jQuery 不但提供了 CSS 的核心操作方法，比如.css()、.addClass()等。还封装了一些特殊功能的 CSS 操作方法，我们分别来了解一下。

width()方法

方法名	描述
width()	获取某个元素的长度

width(value)	设置某个元素的长度
width(function (index, width) {})	通过匿名函数设置某个元素的长度

```
$('div').width(); //获取元素的长度，返回的类型为 number
$('div').width(500); //设置元素长度，直接传数值，默认加 px
$('div').width('500pt'); //同上，设置了 pt 单位
$('div').width(function (index, value) { //index 是索引，value 是原本值
    return value - 500; //无须调整类型，直接计算
});
//alert(typeof parseInt($('div').css('width'))); //string 需要手动转换
//alert(typeof $('div').width());//内部自动转换为 number
```

height()方法

方法名	描述
height()	获取某个元素的长度
height(value)	设置某个元素的长度
height(function (index, width) {})	通过匿名函数设置某个元素的长度

```
$('div').height(); //获取元素的高度，返回的类型为 number
$('div').height(500); //设置元素高度，直接传数值，默认加 px
$('div').height('500pt'); //同上，设置了 pt 单位
$('div').height(function (index, value) { //index 是索引，value 是原本值
    return value - 1; //无须调整类型，直接计算
});
```

内外边距和边框尺寸方法

方法名	描述
innerWidth()	获取元素宽度，包含内边距 padding
innerHeight()	获取元素高度，包含内边距 padding
outerWidth()	获取元素宽度，包含边框 border 和内边距 padding
outerHeight()	获取元素高度，包含边框 border 和内边距 padding
outerWidth(ture)	同上，且包含外边距
outerHeight(true)	同上，且包含外边距

```
alert($('div').width()); //不包含
alert($('div').innerWidth()); //包含内边距 padding
alert($('div').outerWidth()); //包含内边距 padding+边框 border
//包含内边距 padding+边框 border+外边距 margin
alert($('div').outerWidth(true));
```

元素偏移方法

方法名	描述
offset()	获取某个元素相对于视口的偏移位置
position()	获取某个元素相对于父元素的偏移位置
scrollTop()	获取垂直滚动条的值
scrollTop(value)	设置垂直滚动条的值
scrollLeft()	获取水平滚动条的值
scrollLeft(value)	设置水平滚动条的值

```
$('strong').offset().left; //相对于视口的偏移
$('strong').position().left; //相对于父元素的偏移
$(window).scrollTop(); //获取当前滚动条的位置
$(window).scrollTop(300); //设置当前滚动条的位置
```

```
<div title="aaa" style="position:relative;">
    <strong style="position:absolute;top:1px;">www.ycku.com</strong>
```

```
</div>
//alert($('strong').offset().top);     //9
//alert($('strong').position().top);   //1
```

第六章：DOM 节点操作

DOM 中有一个非常重要的功能，就是节点模型，也就是 DOM 中的“M” 。页面中的元素结构就是通过这种节点模型来互相对应着的，我们只需要通过这些节点关系，可以创建、插入、替换、克隆、删除等等一些列的元素操作。

一．创建节点

为了使页面更加智能化，有时我们想动态的在 html 结构页面添加一个元素标签，那么在插入之前首先要做的动作就是：创建节点。

```
var box = $('<div id="box">节点</div>'); //创建一个节点
$('body').append(box); //将节点插入到<body>元素内部
```

二．插入节点

在创建节点的过程中，其实我们已经演示怎么通过.append()方法来插入一个节点。但除了这个方法之余呢，jQuery 提供了其他几个方法来插入节点。

内部插入节点方法

方法名	描述
append(content)	向指定元素内部后面插入节点 content
append(function (index, html) {})	使用匿名函数向指定元素内部后面插入节点
appendTo(content)	将指定元素移入到指定元素 content 内部后面
prepend(content)	向指定元素 content 内部的前面插入节点
prepend(function (index, html) {})	使用匿名函数向指定元素内部的前面插入节点
prependTo(content)	将指定元素移入到指定元素 content 内部前面

```
$('div').append('<strong>节点</strong>'); //向 div 内部后面插入 strong 节点
```

```
//使用匿名函数插入节点，value 是原本节点内容，index 是索引
$('div').append(function (index, value) {
    return '<strong>节点</strong>';//经过处理在返回
});
$('span').appendTo('div'); //先创建到页面中再将 span 节点移入到 div 节点内
$('<span>节点</span>').appendTo('div'); //将 span 节点插入到 div 节点内
$('span').appendTo($('div')); //同上
$('div').prepend('<span>节点</span>'); //将 span 插入到 div 内部的前面
$('div').append(function (index, html) { //使用匿名函数，同上
    return '<span>节点</span>';
});
$('span').prependTo('div'); //将 span 移入 div 内部的前面
$('span').prependTo($('div')); //同上
```

外部插入节点方法

方法名	描述
after(content)	向指定元素的外部后面插入节点 content
after(function (index, html) {})	使用匿名函数向指定元素的外部后面插入节点
before(content)	向指定元素的外部前面插入节点 content
before(function (index, html) {})	使用匿名函数向指定元素的外部前面插入节点
insertAfter(content)	将指定节点移到指定元素 content 外部的后面
insertBefore(content)	将指定节点移到指定元素 content 外部的前面

```
$('div').after('<span>节点</span>'); //向 div 的同级节点后面插入 span
$('div').after(function (index, html) { //使用匿名函数，同上
    return '<span>节点</span>';
});

$('div').before('<span>节点</span>'); //向 div 的同级节点前面插入 span
```

```
$('div').before(function (index, html) { //使用匿名函数，同上
    return '<span>节点</span>';
});

$('span').insertAfter('div'); //将 span 元素移到 div 元素外部的后面
$('span').insertBefore('div'); //将 span 元素移到 div 元素外部的前面
```

三．包裹节点

jQuery 提供了一系列方法用于包裹节点， 那包裹节点是什么意思呢？其实就是使用字符串代码将指定元素的代码包含着的意思。

包裹节点

方法名	描述
wrap(html)	向指定元素包裹一层 html 代码
wrap(element)	向指定元素包裹一层 DOM 对象节点
wrap(function (index) {})	使用匿名函数向指定元素包裹一层自定义内容
unwrap()	移除一层指定元素包裹的内容
wrapAll(html)	用 html 将所有元素包裹到一起
wrapAll(element)	用 DOM 对象将所有元素包裹在一起
wrapInner(html)	向指定元素的子内容包裹一层 html
wrapInner(element)	向指定元素的子内容包裹一层 DOM 对象节点
wrapInner(function (index) {})	用匿名函数向指定元素的子内容包裹一层

```
$('div').wrap('<strong></strong>'); //在 div 外层包裹一层 strong
$('div').wrap('<strong>123</strong>'); //包裹的元素可以带内容
$('div').wrap('<strong><em></em></strong>'); //包裹多个元素
$('div').wrap($('strong').get(0)); //也可以包裹一个原生 DOM
$('div').wrap(document.createElement('strong')); //临时的原生 DOM
$('div').wrap(function (index) { //匿名函数
    return '<strong>'+index+'</strong>';
```

```
});
$('div').unwrap(); //移除最近一层包裹内容，多个需移除多次

$('div').wrapAll('<strong></strong>'); //所有 div 外面只包一层 strong
$('div').wrapAll($('strong').get(0)); //同上

$('div').wrapInner('<strong></strong>'); //包裹子元素内容
$('div').wrapInner($('strong').get(0)); //DOM 节点
$('div').wrapInner(function () { //匿名函数
    return '<strong></strong>';
});
```

注意：.wrap()和.wrapAll()的区别在前者把每个元素当成一个独立体，分别包含一层外层；后者将所有元素作为一个整体作为一个独立体，只包含一层外层。这两种都是在外层包含，而.wrapInner()在内层包含。

四. 节点操作

除了创建、插入和包裹节点，jQuery 还提供了一些常规的节点操作方法：复制、替换和删除节点。

```
//复制节点
$('body').append($('div').clone(true)); //复制一个节点添加到 HTML 中
```

注意：clone(true)参数可以为空，表示只复制元素和内容，不复制事件行为。而加上 true 参数的话，这个元素附带的事件处理行为也复制出来。

```
//删除节点
$('div').remove(); //直接删除 div 元素
```

注意：.remove()不带参数时，删除前面对象选择器指定的元素。而.remove()本事也可以带选择符参数的，比如：$('div').remove('#box');只删除 id=box 的 div。

```
//保留事件的删除节点
$('div').detach(); //保留事件行为的删除
```

注意：.remove()和.detach()都是删除节点，而删除后本身方法可以返回当前被删除的节点对象，但区别在于前者在恢复时不保留事件行为，后者则保留。

```
//清空节点
```

```
$('div').empty(); //删除掉节点里的内容
//替换节点
$('div').replaceWith('<span>节点</span>'); //将 div 替换成 span 元素
$('<span>节点</span>').replaceAll('div'); //同上
```

注意：节点被替换后，所包含的事件行为就全部消失了。

第七章：表单选择器

表单作为 HTML 中一种特殊的元素，操作方法较为多样性和特殊性，开发者不但可以使用之前的常规选择器或过滤器，也可以使用 jQuery 为表单专门提供的选择器和过滤器来准确的定位表单元素。

一．常规选择器

我们可以使用 id、 类(class)和元素名来获取表单字段， 如果是表单元素， 都必须含有 name 属性，还可以结合属性选择器来精确定位。

```
$('input').val(); //元素名定位，默认获取第一个
$('input').eq(1).val(); //同上，获取第二个 //这种写法语义不清晰，
$('input[type=password]').val();//选择 type 为 password 的字段，语义清晰一点
$('input[name=user]').val(); //选择 name 为 user 的字段，语义更清晰一点
```

那么对于 id 和类(class)用法比较类似，也可以结合属性选择器来精确的定位，在这里我们不在重复。对于表单中的其他元素名比如：textarea、select 和 button 等，原理一样，不在重复。

二．表单选择器

虽然可以使用常规选择器来对表单的元素进行定位， 但有时还是不能满足开发者灵活多变的需求。所以，jQuery 为表单提供了专用的选择器。

表单选择器

方法名	描述	返回
:input	选取所有 input、textarea、select 和 button 元素	集合元素
:text	选择所有单行文本框，即 type=text	集合元素
:password	选择所有密码框，即 type=password	集合元素
:radio	选择所有单选框，即 type=radio	集合元素
:checkbox	选择所有复选框，即 type=checkbox	集合元素
:submit	选取所有提交按钮，即 type=submit	集合元素
:reset	选取所有重置按钮，即 type=reset	集合元素
:image	选取所有图像按钮，即 type=image	集合元素
:button	选择所有普通按钮，即 button 元素	集合元素
:file	选择所有文件按钮，即 type=file	集合元素
:hidden	选择所有不可见字段，即 type=hidden	集合元素

```
$(':input').size(); //获取所有表单字段元素
$(':text).size(); //获取单行文本框元素
$(':password').size(); //获取密码栏元素
$(':radio).size(); //获取单选框元素
$(':checkbox).size(); //获取复选框元素
$(':submit).size(); //获取提交按钮元素
$(':reset).size(); //获取重置按钮元素
$(':image).size(); //获取图片按钮元素
$(':file).size(); //获取文件按钮元素
$(':button).size(); //获取普通按钮元素
$(':hidden).size(); //获取隐藏字段元素 （限定范围 from）
```

注意：这些选择器都是返回元素集合，如果想获取某一个指定的元素，最好结合一下属性选择器。比如：

```
alert($(':input[name=city]').size());
alert($(':password[name=pass]').size());
```

```
$(':text[name=user]).size(); //获取单行文本框 name=user 的元素
```

三．表单过滤器

jQuery 提供了四种表单过滤器，分别在是否可以用、是否选定来进行表单字段的筛选过滤。

表单过滤器

方法名	描述	返回
:enabled	选取所有可用元素	集合元素
:disabled	选取所有不可用元素	集合元素
:checked	选取所有被选中的元素，单选和复选字段	集合元素
:selected	选取所有被选中的元素，下拉列表	集合元素

```
$(':enabled').size(); //获取可用元素
$(':disabled).size(); //获取不可用元素
$(':checked).size(); //获取单选、复选框中被选中的元素
$(':selected).size(); //获取下拉列表中被选中的元素
```

第八章：基础事件

JavaScript 有一个非常重要的功能，就是事件驱动。当页面完全加载后，用户通过鼠标或键盘触发页面中绑定事件的元素即可触发。jQuery 为开发者更有效率的编写事件行为， 封装了大量有益的事件方法供我们使用。

一．绑定事件

在 JavaScript 课程的学习中， 我们掌握了很多使用的事件， 常用的事件有：click、 dblclick、mousedown、mouseup、mousemove、 mouseover、mouseout、change、select、 submit、keydown、keypress、keyup、blur、focus、load、resize、scroll、error。那么，还有更多的事件可以参考手册中的事件部分。

jQuery 通过.bind()方法来为元素绑定这些事件。可以传递三个参数：bind(type, [data], fn)，type 表示一个或多个类型的事件名字符串；[data]是可选的，作为 event.data 属性值传递一个额外的数据，这个数据是一个字符串、一个数字、一个数组或一个对象；fn 表示绑定到指定元素的处理函数。

```
//使用点击事件
$('input').bind('click', function () { //点击按钮后执行匿名函数
    alert('点击！');
});
//普通处理函数
$('input').bind('click', fn); //执行普通函数式无须圆括号
function fn() {
    alert('点击！');
}
//可以同时绑定多个事件
$('input').bind('mouseout mouseover', function () { //移入和移出分别执行一次
    $('div').html(function (index, value) {
        return value + '1';
    });
});
//通过对象键值对绑定多个参数
$('input').bind({ //传递一个对象
    'mouseout' : function () { //事件名的引号可以省略
    alert('移出');
    },
    'mouseover' : function () {
    alert('移入');
    }
});
//使用 unbind 删除绑定的事件
$('input').unbind(); //删除所有当前元素的事件
//使用 unbind 参数删除指定类型事件
$('input').unbind('click'); //删除当前元素的 click 事件
//使用 unbind 参数删除指定处理函数的事件
function fn1() {
```

```
    alert('点击 1');
}
function fn2() {
    alert('点击 2');
}
$('input').bind('click', fn1);
$('input').bind('click', fn2);
$('input').unbind('click', fn1); //只删除 fn1 处理函数的事件
```

二. 简写事件

为了使开发者更加方便的绑定事件，jQuery 封装了常用的事件以便节约更多的代码。 我们称它为简写事件。

简写事件绑定方法

方法名	触发条件	描述
click(fn)	鼠标	触发每一个匹配元素的 click(单击)事件
dblclick(fn)	鼠标	触发每一个匹配元素的 dblclick(双击)事件
mousedown(fn)	鼠标	触发每一个匹配元素的 mousedown(点击后)事件
mouseup(fn)	鼠标	触发每一个匹配元素的 mouseup(点击弹起)事件
mouseover(fn)	鼠标	触发每一个匹配元素的 mouseover(鼠标移入)事件
mouseout(fn)	鼠标	触发每一个匹配元素的 mouseout(鼠标移出)事件

mousemove(fn)	鼠标	触发每一个匹配元素的 mousemove(鼠标移动)事件
mouseenter(fn)	鼠标	触发每一个匹配元素的 mouseenter(鼠标穿过)事件
mouseleave(fn)	鼠标	触发每一个匹配元素的 mouseleave(鼠标穿出)事件
keydown(fn)	键盘	触发每一个匹配元素的 keydown(键盘按下)事件
keyup(fn)	键盘	触发每一个匹配元素的 keyup(键盘按下弹起)事件
keypress(fn)	键盘	触发每一个匹配元素的 keypress(键盘按下)事件
unload(fn)	文档	当卸载本页面时绑定一个要执行的函数
resize(fn)	文档	触发每一个匹配元素的 resize(文档改变大小)事件
scroll(fn)	文档	触发每一个匹配元素的 scroll(滚动条拖动)事件
focus(fn)	表单	触发每一个匹配元素的 focus(焦点激活)事件
blur(fn)	表单	触发每一个匹配元素的 blur(焦点丢失)事件
focusin(fn)	表单	触发每一个匹配元素的 focusin(焦点激活)事件
focusout(fn)	表单	触发每一个匹配元素的 focusout(焦点丢失)事件
select(fn)	表单	触发每一个匹配元素的 select(文本选定)事件
change(fn)	表单	触发每一个匹配元素的 change(值改变)事件
submit(fn)	表单	触发每一个匹配元素的 submit(表单提交)事件

注意：这里封装的大部分方法都比较好理解，我们没必要一一演示确认，重点看几个需要注意区分的简写方法。

.mouseover()和.mouseout()表示鼠标移入和移出的时候触发。那么 jQuery 还封装了另外一组：.mouseenter()和.mouseleave()表示鼠标穿过和穿出的时候触发。那么这两组本质上有什么区别呢？手册上的说明是：.mouseenter()和.mouseleave()这组穿过子元素不会触发，而.mouseover()和.mouseout()则会触发。

```
//一般 unload 卸载页面新版浏览器应该是不支持的，获取要设置一个。
$(window).unload(function () {
        alert('1');     //一般用于清理工作。
});
$('form').submit(function () {
        alert('表单提交！');  //是表单提交才触发
```

```
 });

//HTML 页面设置
<div style="width:200px;height:200px;background:green;">
    <p style="width:100px;height:100px;background:red;"></p>
</div>
<strong></strong>

//mouseover 移入    (mouseover 会触发子节点)
$('div').mouseover(function () { //移入 div 会触发, 移入 p 再触发
    $('strong').html(function (index, value) {
        return value+'1';
    });
});

//mouseenter 穿过   (mouseenter 不会触发子节点)
$('div').mouseenter(function () { //穿过 div 或者 p
    $('strong').html(function (index, value) { //在这个区域只触发一次
        return value+'1';
    });
});

//mouseout 移出
$('div').mouseout(function () { //移出 p 会触发, 移出 div 再触发
    $('strong').html(function (index, value) {
        return value+'1';
    });
});
//mouseleave 穿出
$('div').mouseleave(function () { //移出整个 div 区域触发一次
```

```
    $('strong').html(function (index, value) {
        return value+'1';
    });
});
```

.keydown()、.keyup()返回的是键码，而.keypress 返回的是字符编码。

```
$('input').keydown(function (e) {
    alert(e.keyCode); //按下 a 返回 65
});
$('input').keypress(function (e) {
    alert(e.charCode); //按下 a 返回 97
});
```

注意：e.keyCode 和 e.charCode 在两种事件互换也会产生不同的效果，除了字符还有一些非字符键的区别。更多详情可以了解 JavaScript 事件处理那章。.focus()和.blur()分别表示光标激活和丢失，事件触发时机是当前元素。而.focusin()和.focusout()也表示光标激活和丢失，但事件触发时机可以是子元素。

```
//HTML 部分
<div style="width:200px;height:200px;background:red;">
    <input type="text" value="" />
</div>
<strong></strong>

//focus 光标激活
$('input').focus(function () { //当前元素触发
    $('strong').html('123');
});
//focusin 光标激活
$('div').focusin(function () { //绑定的是 div 元素， 子类 input 触发
    $('strong').html('123');
});
```

注意：.blur()和.focusout()表示光标丢失，和激活类似，一个必须当前元素触发，一个可以是子元素触发。

```
$('div').focus(function () {   //focus 和 blur 必须是当前元素才能激活
    alert('光标激活');
});
$('div').focusin(function () {     //focusin 和 focusout 可以是子元素激活
    alert('光标激活');
});
```

三．复合事件

jQuery 提供了许多最常用的事件效果，组合一些功能实现了一些复合事件，比如切换功能、智能加载等。

复合事件

方法名	描述
ready(fn)	当 DOM 加载完毕触发事件
hover([fn1,]fn2)	当鼠标移入触发第一个 fn1，移出触发 fn2
toggle(fn1,fn2[,fn3..])	已废弃，当鼠标点击触发 fn1,再点击触发 fn2...

```
//背景移入移出切换效果
$('div').hover(function () {
$(this).css('background', 'black'); //mouseenter 效果
}, function () {
$(this).css('background', 'red'); //mouseleave 效果，可省略
});
```

注意：.hover()方法是结合了.mouseenter()方法和.mouseleva()方法，并非.mouseover()和.mouseout()方法。.toggle()这个方法比较特殊，这个方法有两层含义，第一层含义就是已经被 1.8 版废用、1.9 版删除掉的用法，也就是点击切换复合事件的用法。第二层函数将会在动画那章讲解到。既然废弃掉了，就不应该使用。被删除的原因是：以减少混乱和提高潜在的模块化程度。但你又非常想用这个方法，并且不想自己编写类似的功能，可以下载 jquery-migrate.js 文件，来向下兼容已被删除掉的方法。

```
//背景点击切换效果(1.9 版删除掉了)
```

```
<script type="text/javascript" src="jquery-migrate-1.2.1.js"></script>
$('div').toggle(function () { //第一次点击切换
        $(this).css('background', 'black');
    }, function () { //第二次点击切换
        $(this).css('background', 'blue');
    }, function () { //第三次点击切换
        $(this).css('background', 'red');
});
```

注意：由于官方已经删除掉这个方法，所以也是不推荐使用的，如果在不基于向下兼容的插件 JS。我们可以自己实现这个功能。

```
var flag = 1; //计数器
$('div').click(function () {
    if (flag == 1) { //第一次点击
    $(this).css('background', 'black');
    flag = 2;
} else if (flag == 2) { //第二次点击
    $(this).css('background', 'blue');
    flag = 3
} else if (flag == 3) { //第三次点击
    $(this).css('background', 'red');
    flag = 1
}
```

第 9 章 事件对象

JavaScript 在事件处理函数中默认传递了 event 对象，也就是事件对象。但由于浏览器的兼容性，开发者总是会做兼容方面的处理。jQuery 在封装的时候，解决了这些问题，并且还创建了一些非常好用的属性和方法。

一. 事件对象

事件对象就是 event 对象，通过处理函数默认传递接受。之前处理函数的 e 就是

event 事件对象，event 对象有很多可用的属性和方法，我们在 JavaScript 课程中已经详细的了解过这些常用的属性和方法，这里，我们再一次演示一下。

```
//通过处理函数传递事件对象
$('input').bind('click', function (e) { //接受事件对象参数
    alert(e);
});
```

event 对象的属性

属性名	描述
type	获取这个事件的事件类型，例如：click
target	获取绑定事件的 DOM 元素
data	获取事件调用时的额外数据
relatedTarget	获取移入移出目标点离开或进入的那个 DOM 元素
currentTarget	获取冒泡前触发的 DOM 元素，等同与 this
pageX/pageY	获取相对于页面原点的水平/垂直坐标
screenX/screenY	获取显示器屏幕位置的水平/垂直坐标(非 jQuery 封装)
clientX/clientY	获取相对于页面视口的水平/垂直坐标(非 jQuery 封装)
result	获取上一个相同事件的返回值
result	获取上一个相同事件的返回值
timeStamp	获取事件触发的时间戳
which	获取鼠标的左中右键(1,2,3)，或获取键盘按键
altKey/shiftKey/ ctrlKey/metaKey	获取是否按下了 alt、shift、ctrl(这三个非 jQuery 封装)或 meta 键(IE 原生 meta 键，jQuery 做了封装)

```
//通过 event.type 属性获取触发事件名
$('input').click(function (e) {
    alert(e.type);
});
//通过 event.target 获取绑定的 DOM 元素
$('input').click(function (e) {
    alert(e.target);
```

```
});
//通过 event.data 获取额外数据，可以是数字、字符串、数组、对象
$('input').bind('click', 123, function () { //传递 data 数据
    alert(e.data); //获取数字数据
});
```

注意：如果字符串就传递：'123'、如果是数组就传递：[123,'abc']，如果是对象就传递：{user : 'Lee', age : 100}。数组的调用方式是：e.data[1]，对象的调用方式是：e.data.user。

```
//event.data 获取额外数据，对于封装的简写事件也可以使用
$('input').click({user : 'Lee', age : 100},function (e) {
    alert(e.data.user);
});
```

注意：键值对的键可以加上引号，也可以不加；在调用的时候也可以使用数组的方式：alert(e.data['user']);

```
//获取移入到 div 之前的那个 DOM 元素
$('div').mouseover(function (e) {
    alert(e.relatedTarget);//移入之前的元素
});

//获取移出 div 之后到达最近的那个 DOM 元素
$('div').mouseout(function (e) {
    alert(e.relatedTarget);
});
//获取绑定的那个 DOM 元素，相当于 this，区别与 event.target
$('div').click(function (e) {
    alert(e.currentTarget);
});
```

注意：event.target 得到的是触发元素的 DOM，event.currentTarget 得到的是监听元素的 DOM。而 this 也是得到监听元素的 DOM。

```
$('div').bind('click', function (e) {
```

1.this和event.target的区别：

js中事件是会冒泡的，所以this是可以变化的，但event.target不会变化，它永远是直接接受事件的目标DOM元素；

2.this和event.target都是dom对象，如果要使用jquey中的方法可以将他们转换为jquery对象：$(this)和$(event.target)

```
    alert(e.target);    //单击 div 获取 div 对象 ， 单击 span 是获取 span 对象
});
$('div').bind('click', function (e) {
    alert(e.currentTarget);//单击 div,span 都值获取事件监听的对象 div
});
<div style="width:200px;height:200px;background:#ccc;">
    <span style="width:100px;height:100px;background:#eee;display:block;">
    </span>
</div>
//target 是获取触发元素的 DOM，触发元素，就是你点了哪个就是哪个
//currentTarget 得到的是监听元素的 DOM，你绑定的哪个就是那个

//获取上一次事件的返回值
$('div').click(function (e) {
    return '123';
});
$('div').click(function (e) {
    alert(e.result);
});
//获取当前的时间戳
$('div').click(function (e) {
    alert(e.timeStamp);
});
//获取鼠标的左中右键
$('div').mousedown(function (e) {
    alert(e.which);
});

//获取键盘的按键
$('input').keyup(function (e) {
```

```
    alert(e.which);
});
//获取是否按下了 ctrl 键，meta 键不存在，导致无法使用
$('input').click(function (e) {
    alert(e.ctrlKey);
});
//获取触发元素鼠标当前的位置
$(document).click(function (e) {
    alert(e.screenY+ ',' + e.pageY + ',' + e.clientY);
});
```

二. 冒泡和默认行为

如果在页面中重叠了多个元素，并且重叠的这些元素都绑定了同一个事件，那么就会出现冒泡问题。

```
//HTML 页面
<div style="width:200px;height:200px;background:red;">
    <input type="button" value="按钮" />
</div>
//三个不同元素触发事件
$('input').click(function () {
    alert('按钮被触发了！');
});
$('div').click(function () {
    alert('div 层被触发了！');
});
$(document).click(function () {
    alert('文档页面被触发了！');
});
```

注意:当我们点击文档的时候,只触发文档事件;当我们点击 div 层时,触发了 div 和文档两个；当我们点击按钮时，触发了按钮、div 和文档。触发的顺序是从小范围到大范围。这就是所谓的冒泡现象，一层一层往上。（三个元素绑定相同的事件，出现冒

泡会影响元素事件的触发）

jQuery 提供了一个事件对象的方法：event.stopPropagation()；这个方法设置到需要触发的事件上时，所有上层的冒泡行为都将被取消。

```
$('input').click(function (e) {
    alert('按钮被触发了！');
    e.stopPropagation();
});
```

网页中的元素，在操作的时候会有自己的默认行为。比如：右击文本框输入区域，会弹出系统菜单、点击超链接会跳转到指定页面、点击提交按钮会提交数据。

```
$('a').click(function (e) {
    e.preventDefault();
});
//禁止提交表单跳转
$('form').submit(function (e) {
    e.preventDefault();
});
```

注意：如果想让上面的超链接同时阻止默认行为且禁止冒泡行为，可以把两个方法同时写上： event.stopPropagation()和 event.preventDefault()。 这两个方法如果需要同时启用的时候，

还有一种简写方案代替，就是直接 return false。

```
$('a').click(function (e) {
    return false;
});
```

冒泡和默认行为的一些方法

方法名	描述
preventDefault()	取消某个元素的默认行为
isDefaultPrevented()	判断是否调用了 preventDefault()方法
stopPropagation()	取消事件冒泡
isPropagationStopped()	判断是否调用了 stopPropagation()方法
stopImmediatePropagation()	取消事件冒泡，并取消该事件的后续事件处理函数
isImmediatePropagationStopped()	判断是否调用了 stopImmediatePropagation()方法

```
//判断是否取消了元素的默认行为
$('input').keyup(function (e) {
    e.preventDefault();
    alert(e.isDefaultPrevented());
});
//取消冒泡并取消后续事件处理函数
$('input').click(function (e) {
    alert('input');
    e.stopImmediatePropagation();
});
$('input').click(function () {
    alert('input2');
});
$(document).click(function () {
    alert('document');
});
//判断是否调用了 stopPropagation()方法
$('input').click(function (e) {
    e.stopPropagation();
    alert(e.isPropagationStopped());
});
```

```
//判断是否执行了 stopImmediatePropagation()方法
$('input').click(function (e) {
    e.stopImmediatePropagation();
    alert(e.isImmediatePropagationStopped());
});
```

第十章：高级事件

jQuery 不但封装了大量常用的事件处理，还提供了不少高级事件方便开发者使用。比如模拟用户触发事件、事件委托事件、和统一整合的 on 和 off，以及仅执行一次的 one 方法。这些方法大大降低了开发者难度，提升了开发者的开发体验。

一. 模拟操作

在事件触发的时候，有时我们需要一些模拟用户行为的操作。例如：当网页加载完毕后自行点击一个按钮触发一个事件，而不是用户去点击。

```
//点击按钮事件
$('input').click(function () {
    alert('我的第一次点击来自模拟！');
});
//模拟用户点击行为
$('input').trigger('click');
//可以合并两个方法
$('input').click(function () {
    alert('我的第一次点击来自模拟！');
}).trigger('click');
```

有时在模拟用户行为的时候， 我们需要给事件执行传递参数， 这个参数类似与 event.data 的额外数据，可以可以是数字、字符串、数组、对象。

```
$('input').click(function (e, data1, data2) {
alert(data1 + ',' + data2);
```

```
}).trigger('click', ['abc', '123']);
```

注意：当传递一个值的时候，直接传递即可。当两个值以上，需要在前后用中括号包含起来。但不能认为是数组形式，下面给出一个复杂的说明。（参数数目保持一致）

```
$('input').click(function (e, data1, data2) {
    alert(data1.a + ',' + data2[1]);
}).trigger('click', [{'a' : '1', 'b' : '2'}, ['123','456']]);
```

除了通过 JavaScript 事件名触发，也可以通过自定义的事件触发，所谓自定义事件其实就是一个被.bind()绑定的任意函数。

```
$('input').bind('myEvent', function () {
    alert('自定义事件！');
}).trigger('myEvent');
```

.trigger()方法提供了简写方案，只要想让某个事件执行模拟用户行为，直接再调用一个空的同名事件即可。

```
$('input').click(function () {
    alert('我的第一次点击来自模拟！');
}).click(); //空的 click()执行的是 trigger()
```

这种便捷的方法，jQuery 几乎个所有常用的事件都提供了。

blur	focusin	mousedown	resize
change	focusout	mousenter	scroll
click	keydown	mouseleave	select
dblclick	keypress	mousemove	submit
error	keyup	mouseout	unload
focus	load	mouseover	

jQuery 还提供了另外一个模拟用户行为的方法：.triggerHandler()；这个方法的使用和.trigger()方法一样。

```
$('input').click(function () {
    alert('我的第一次点击来自模拟！');
}).triggerHandler('click');
```

在常规的使用情况下，两者几乎没有区别，都是模拟用户行为，也可以可以传递额外参数。但在某些特殊情况下，就产生了差异：

1..triggerHandler()方法并不会触发事件的默认行为，而.trigger()会。

```
$('form').trigger('submit'); //模拟用户执行提交，并跳转到执行页面
$('form').triggerHandler('submit'); //模拟用户执行提交，并阻止的默认行为
```

如果我们希望使用.trigger()来模拟用户提交，并且阻止事件的默认行为，则需要这么写：

```
$('form').submit(function (e) {
    e.preventDefault(); //阻止默认行为
}).trigger('submit');
```

2..triggerHandler()方法只会影响第一个匹配到的元素，而.trigger()会影响所有。

3..triggerHandler()方法会返回当前事件执行的返回值，如果没有返回值，则返回undefined；而.trigger()则返回当前包含事件触发元素的 jQuery 对象(方便链式连缀调用)。

```
alert($('input').click(function () {
    return 123;
}).triggerHandler('click')); //返回 123，没有 return 返回

$('input').click(function () {
    alert('我将要使用模拟用户操作来触发！');
}).trigger('click').css('color', 'red');  //返回 jQuery 对象，可以连缀
$('input').click(function () {
    alert('我将要使用模拟用户操作来触发！');
    return 123;
}).triggerHandler('click').css('color', 'red');//返回 return 值，或 undefined
```

4..trigger()在创建事件的时候，会冒泡。但这种冒泡是自定义事件才能体现出来，是 jQuery 扩展于 DOM 的机制，并非 DOM 特性。而.triggerHandler()不会冒泡。

```
var index = 1;
```

```
$('div').bind('myEvent',function(){
    alert('自定义事件' + index);
    index++;
});
$('.div3').trigger("myEvent");
```

HTML 页面

```
<div class="d1">
    <div class="d2">
        <div class="d3">
            div 内容
        </div>
    </div>
</div>
```

二. 命名空间

有时，我们想对事件进行移除。但对于同名同元素绑定的事件移除往往比较麻烦，这个时候，可以使用事件的命名空间解决。（之前我们处理方式会将匿名函数定义成普通函数以函数名区分去移除）

```
$('input').bind('click.abc', function () {
    alert('abc');
});
$('input').bind('click.xyz', function () {
    alert('xyz');
});
$('input').bind('mouseover.abc', function () {
        alert('abc');
    });
$('input').unbind('click.abc'); //移除 click 事件中命名空间为 abc 的
//$('input').unbind('.abc');//移除命名空间为 abc 的所有绑定的事件
```

定义事件的命名空间：事件.名称

注意：也可以直接使用('.abc')，这样的话，可以移除相同命名空间的不同事件。对于模拟操作.trigger()和.triggerHandler()，用法也是一样的。

```
$('input').trigger('click.abc');
```

三. 事件委托

什么是事件委托？用现实中的理解就是：有 100 个学生同时在某天中午收到快递，但这 100 个学生不可能同时站在学校门口等，那么都会委托门卫去收取，然后再逐个交给学生。而在 jQuery 中，我们通过事件冒泡的特性，让子元素绑定的事件冒泡到父元素(或祖先元素)上，然后再进行相关处理即可。

如果一个企业级应用做报表处理，表格有 2000 行，每一行都有一个按钮处理。如果用之前的.bind()处理，那么就需要绑定 2000 个事件，就好比 2000 个学生同时站在学校门口等快递，不断会堵塞路口，还会发生各种意外。这种情况放到页面上也是一样，可能导致页面极度变慢或直接异常。而且，2000 个按钮使用 ajax 分页的话，.bind()方法无法动态绑定尚未存在的元素。就好比，新转学的学生，快递员无法验证他的身份，就可能收不到快递。

```
//HTML 部分
<div style="background:red;width:200px;height:200px;" id="box">
    <input type="button" value="按钮" class="button" />
    <input type="button" value="按钮" class="button" />
    <input type="button" value="按钮" class="button" />
</div>
//.bind 绑定了三个 click 事件
$('.button').bind('click', function () {
    alert('事件不委托！');
});
//使用 live 绑定的是 document，而非 button 所以，永远只会绑定一次事件，
$('.button').live('click', function () {
    alert('事件委托！');
});
```

```
//使用.bind()不具备动态绑定功能，只有点击原始按钮才能生成
$('.button').bind('click', function () {
    $(this).clone().appendTo('#box');
});
//使用.live()具备动态绑定功能，jQuery1.3 使用，jQuery1.7 之后废弃，jQuery1.9 删除，.live 可以动态绑定事件，因为事件绑定在 document 上
$('.button').live('click', function () {
    $(this).clone().appendTo('#box');
});
```

.live()原理就是把 click 事件绑定到祖先元素$(document)上，而只需要给$(document)绑定一次即可（冒泡行为），而非 2000 次。然后就可以处理后续动态加载的按钮的单击事件。在接受任何事件时，$(document)对象都会检查事件类型(event.type)和事件目标(event.target)，如果 click 事件是.button，那么就执行委托给它的处理程序。.live()方法已经被删除，无法使用了。需要测试使用的话，需要引入向下兼容插件。.live 绑定在 document 上,而点击. button 其实是冒泡到 document 上，并且点击 document 时候，需要验证 event.type 和 event.target 才能触发

```
//.live()无法使用链接连缀调用，因为参数的特性导致
$('#box').children(0).live('click', function () {
    $(this).clone().appendTo('#box');
});
```

在上面的例子中，我们使用了.clone()克隆。其实如果想把事件行为复制过来，我们只需要传递 true 即可：.clone(true)。这样也能实现类似事件委托的功能，但原理却截然不同。一个是复制事件行为，一个是事件委托。而在非克隆操作下，此类功能只能使用事件委托。

```
$('.button').live('click', function () {
$('<input type="button" value="复制的" class="button" />').appendTo('#box');
});
```

当我们需要停止事件委托的时候，可以使用.die()来取消掉。

```
$('.button').die('click');
```

由于.live()和.die()在 jQuery1.4.3 版本中废弃了， 之后推出语义清晰、 减少冒泡传播层次、又支持链接连缀调用方式的方法：.delegate()和.undelegate()。但这个方法在 jQuery1.7 版本中被.on()方法整合替代了。

```
$('#box').delegate('.button', 'click', function () {
    $(this).clone().appendTo('#box');
}); //#box 是绑定的元素，.button 是触发的元素
$('#box').undelegate('.button','click');
//支持连缀调用方式
$('div').first().delegate('.button', 'click', function () {
    $(this).clone().appendTo('div:first');
});
```

注意：.delegate()需要指定父元素，然后第一个参数是当前元素，第二个参数是事件方式，第三个参数是执行函数。和.bind()方法一样，可以传递额外参数。.undelegate()和.unbind()方法一样可以直接删除所有事件，比如：.undelegate('click')。也可以删除命名空间的事件，

比如：.undelegate('click.abc')。

注意：.live()和.delegate()和.bind()方法一样都是事件绑定，那么区别也很明显，用途上遵循两个规则：1.在 DOM 中很多元素绑定相同事件时；2.在 DOM 中尚不存在即将生成的元素绑定事件时；我们推荐使用事件委托的绑定方式，否则推荐使用.bind()的普通绑定。

四. on、off 和 one

目前绑定事件和解绑的方法有三组共六个。由于这三组的共存可能会造成一定的混乱，为此 jQuery1.7 以后推出了.on()和.off()方法彻底摒弃前面三组。

HTML 页面

```
<body>
<div style="width:200px;height:200px;background:green;" id="box">
    <input type="button" class="button" value="按钮" />
</div>
</body>
```

```
//替代.bind()方式
$('.button').on('click', function () {
    alert('替代.bind()');
});
//替代.bind()方式，并使用额外数据和事件对象
$('.button').on('click', {user : 'Lee'}, function (e) {
    alert('替代.bind()' + e.data.user);
});
//替代.bind()方式，并绑定多个事件
$('.button').on('mouseover mouseout', function () {
    alert('替代.bind()移入移出！');
});
//替代.bind()方式，以对象模式绑定多个事件
$('.button').on({
    mouseover : function () {
        alert('替代.bind()移入！');
    },
    mouseout : function () {
        alert('替代.bind()移出！');
    }
});
//替代.bind()方式，阻止默认行为并取消冒泡
$('form').on('submit', function () {
    return false;
});
```

或

```
$('form').on('submit', false);
//替代.bind()方式，阻止默认行为
$('form').on('submit', function (e) {
    e.preventDefault();
```

```
});
//替代.bind()方式，取消冒泡
$('form').on('submit', function (e) {
    e.stopPropagation();
});
//替代.unbind()方式，移除事件
$('.button').off('click');
$('.button').off('click', fn);
$('.button').off('click.abc');
//替代.live()和.delegate()，事件委托
$('#box').on('click', '.button', function () {
    $(this).clone().appendTo('#box');
});
//替代.die()和.undelegate()，取消事件委托
$('#box').off('click', '.button');
```

注意：和之前方式一样，事件委托和取消事件委托也有各种搭配方式，比如额外数据、命名空间等等，这里不在赘述。

不管是.bind()还是.on()，绑定事件后都不是自动移除事件的，需要通过.unbind()和.off()来手工移除。jQuery 提供了.one()方法，绑定元素执行完毕后自动移除事件，可以方法仅触发一次的事件。

```
//类似于.bind()只触发一次
$('.button').one('click', function () {
    alert('one 仅触发一次！');
});
```

第 十一 章 动画效果

在以前很长一段时间里，网页上的各种特效还需要采用 flash 在进行。但最近几年里，我们已经很少看到这种情况了，绝大部分已经使用 JavaScript 动画效果来取代 flash。这里说的取代是网页特效部分，而不是动画。网页特效比如：渐变菜单、渐进显示、图片轮播等；而动画比如：故事情节广告、MV 等等。

一. 显示、隐藏

jQuery 中显示方法为：.show()，隐藏方法为：.hide()。在无参数的时候，只是硬性的显示内容和隐藏内容。

```
$('.show').click(function () { //显示
    $('#box').show();
});
$('.hide').click(function () { //隐藏
    $('#box').hide();
});
```

注意：.hide()方法其实就是在行内设置 CSS 代码：display:none; 而.show()方法要根据原来元素是区块还是内联来决定，如果是区块，则设置 CSS 代码：display:block; 如果是内联，则设置 CSS 代码：display:inline;。

在.show()和.hide()方法可以传递一个参数，这个参数以毫秒(1000 毫秒等于 1 秒钟)来控制速度。并且里面富含了匀速变大变小，以及透明度变换。

```
$('.show').click(function () {
    $('#box').show(1000); //显示用了 1 秒
});
$('.hide').click(function () {
    $('#box').hide(1000); //隐藏用了 1 秒
});
```

除了直接使用毫秒来控制速度外，jQuery 还提供了三种预设速度参数字符串：slow、normal 和 fast，分别对应 600 毫秒、400 毫秒和 200 毫秒。

```
$('.show').click(function () {
    $('#box').show('fast'); //200 毫秒
});
$('.hide').click(function () {
    $('#box').hide('slow'); //600 毫秒
});
```

注意： 不管是传递毫秒数还是传递预设字符串， 如果不小心传递错误或者传递空字符串。那么它将采用默认值：400 毫秒。

```
$('.show').click(function () {
    $('#box').show(''); //默认 400 毫秒
});
//使用.show()和.hide()的回调函数，可以实现列队动画效果。
$('.show').click(function () {
    $('#box').show('slow', function () {
        alert('动画持续完毕后，执行我！');//显示之后执行
    });
});
<div class="test">你</div>
<div class="test">好</div>
<div class="test">吗</div>
<div class="test">? </div>
//同步动画，四个区块同时显示出来
$('.show').click(function () {
    $('.test').show('slow');
});
//列队动画 （处理起来比较麻烦，不提倡）
$('.show').click(function () {
    $('.test').eq(0).show('fast', function () {
        $('.test').eq(1).show('fast', function () {
            $('.test').eq(2).show('fast', function () {
```

```
                $('.test').eq(3).show('fast');
            });
        });
    });
});
//列队动画，使用函数名调用自身 （递归自调用）
$('.show').click(function () {
    $('div').first().show('fast', function showSpan() {
        $(this).next().show('fast', showSpan);
    });
});
//列队动画，使用 arguments.callee 匿名函数自调用
$('.hide').click(function () {
    $('div').last().hide('fast', function() {
        $(this).prev().hide('fast', arguments.callee);
    });
});
```

我们在使用.show()和.hide()的时候，如果需要一个按钮切换操作，需要进行一些条件判断。而 jQuery 提供给我们一个类似功能的独立方法：.toggle()。

```
$('.toggle').click(function () {
    $(this).toggle('slow');
});
```

二. 滑动、卷动

jQuery 提供了一组改变元素高度的方法：.slideUp()、.slideDown()和.slideToggle()。顾名思义，向上收缩(卷动)和向下展开(滑动)。

```
$('.down').click(function () {
    $('#box').slideDown();
});
```

```
$('.up').click(function () {
    $('#box').slideUp();
});
$('.toggle').click(function () {
    $('#box').slideToggle();
});
```

注意：滑动、卷动效果和显示、隐藏效果一样，具有相同的参数。

三．淡入、淡出

jQuery 提供了一组专门用于透明度变化的方法：.fadeIn()和.fadeOut()，分别表示淡入、淡出，当然还有一个自动切换的方法：.fadeToggle()。

```
$('.in').click(function () {
    $('#box').fadeIn('slow');
});
$('.out').click(function () {
    $('#box').fadeOut('slow');
});
$('.toggle').click(function () {
    $('#box').fadeToggle();
});
```

上面三个透明度方法只能是从 0 到 100，或者从 100 到 0，如果我们想设置指定值就没有办法了。而 jQuery 为了解决这个问题提供了.fadeTo()方法。

```
$('.toggle').click(function () {
    $('#box').fadeTo('slow', 0.33); //0.33 表示值为 33
});
```

注意：淡入、淡出效果和显示、隐藏效果一样，具有相同的参数。对于.fadeTo()方法，如果本身透明度大于指定值，会淡出，否则相反。

四．自定义动画

jQuery 提供了几种简单常用的固定动画方面我们使用。但有些时候，这些简单动画无法满足我们更加复杂的需求。这个时候，jQuery 提供了一个.animate()方法来创建我们的自定义动画，满足更多复杂多变的要求。

```
//同步动画
$('.animate').click(function () {
    $('#box').animate({
        'width' : '300px',
        'height' : '200px',
        'font-Size' : '50px',
        'opacity' : 0.5
    });
});
```

注意：一个 CSS 变化就是一个动画效果，上面的例子中，已经有四个 CSS 变化，已经实现了多重动画同步运动的效果。必传的参数只有一个，就是一个键值对 CSS 变化样式的对象。还有两个可选参数分别为速度和回调函数。

```
//参数，回调函数
$('.animate').click(function () {
    $('#box').animate({
        'width' : '300px',
        'height' : '200px'
    }, 1000, function () {
        alert('动画执行完毕再来执行我！');//回调函数
    });
});
```

到目前位置， 我们都是创建的固定位置不动的动画。 如果想要实现运动状态的位移动画，那就必须使用自定义动画，并且结合 CSS 的绝对定位功能。

```
#box{position:absolute;}
$('.animate').click(function () {
    $('#box').animate({
```

```
        'top' : '300px', //先必须设置 CSS 绝对定位
        'left' : '200px'
    });
});
```

自定义动画中，每次开始运动都必须是初始位置或初始状态，而有时我们想通过当前位置或状态下再进行动画。jQuery 提供了自定义动画的累加、累减功能。

```
$('.animate').click(function () {
    $('#box').animate({
        'left' : '+=100px',
    });
});
```

自定义实现列队动画的方式，有两种：1.在回调函数中再执行一个动画；2.通过连缀或顺序来实现列队动画。

```
//通过依次顺序实现列队动画
$('.animate').click(function () {
$('#box').animate({'left' : '100px'});
$('#box').animate({'top' : '100px'});
$('#box').animate({'width' : '300px'});
});
```

注意：如果不是同一个元素，就会实现同步动画

```
//通过连缀实现列队动画 $('.animate').click(function () {
$('#box').animate({
        'left' : '100px'
    }).animate({
        'top' : '100px'
    }).animate({
        'width' : '300px'
    });
});
```

```
//通过回调函数实现列队动画
$('.animate').click(function () {
    $('#box').animate({
        'left' : '100px'
    }, function () {
        $('#box').animate({
            'top' : '100px'
        }, function () {
            $('#box').animate({
                'width' : '300px'
            });
        });
    });
});
```

五. 列队动画方法

之前我们已经可以实现列队动画了，如果是同一个元素，可以依次顺序或连缀调用。如果是不同元素，可以使用回调函数。但有时列队动画太多，回调函数的可读性大大降低。 为此， jQuery 提供了一组专门用于列队动画的方法。

```
//连缀无法实现按顺序列队 （会先执行 CSS 方法）
$('#box').slideUp('slow').slideDown('slow').css('background', 'orange');
```

注意：如果动画方法，连缀可以实依次列队，而.css()方法不是动画方法，会在一开始传入列队之前。那么，可以采用动画方法的回调函数来解决。

```
//使用回调函数，强行将.css()方法排队到.slideDown()之后
$('#box').slideUp('slow').slideDown('slow', function () {
    $(this).css('background', 'orange');
});
```

但如果这样的话，当列队动画繁多的时候，可读性不但下降，而原本的动画方法不够清晰。所以，我们的想法是每个操作都是自己独立的方法。那么 jQuery 提供了一个类似于回调函数的方法：.queue()。

```
//使用.queue()方法模拟动画方法跟随动画方法之后
$('#box').slideUp('slow').slideDown('slow').queue(function () {
    $(this).css('background', 'orange');
});
```

现在，我们想继续在.queue()方法后面再增加一个隐藏动画，这时发现居然无法实现。这是.queue()特性导致的。有两种方法可以解决这个问题，jQuery 的.queue()的回调函数可以传递一个参数，这个参数是 next 函数，在结尾处调用这个 next()方法即可再连缀执行列队动画。

```
//使用 next 参数来实现继续调用列队动画
$('#box').slideUp('slow').slideDown('slow').queue(function (next) {
    $(this).css('background', 'orange');
    next();
}).hide('slow');
```

因为 next 函数是 jQuery1.4版本以后才出现的，而之前我们普遍使用的是.dequeue()方法。意思为执行下一个元素列队中的函数。

```
//使用.dequeue()方法执行下一个函数动画
$('#box').slideUp('slow').slideDown('slow').queue(function () {
    $(this).css('background', 'orange');
    $(this).dequeue();
}).hide('slow');
```

如果采用顺序调用，那么使用列队动画方法，就非常清晰了，每一段代表一个列队， 而回调函数的嵌套就会杂乱无章。

```
//使用顺序调用的列队，逐个执行，非常清晰
$('#box').slideUp('slow');
$('#box').slideDown('slow');
$('#box').queue(function () {
    $(this).css('background', 'orange');
    $(this).dequeue();
})
$('#box').hide('slow');
```

.queue()方法还有一个功能，就是可以得到当前动画个列队的长度。当然，这个用法在普通 Web 开发中用的比较少，我们这里不做详细探讨。

```
//获取当前列队的长度，fx 是默认列队的参数
function count() {
    return $("#box").queue('fx').length;
}
//在某个动画处调用
$('#box').slideDown('slow', function () {alert(count());});
```

jQuery 还提供了一个清理列队的功能方法：.clearQueue()。把它放入一个列队的回调函数或.queue()方法里，就可以把剩下为执行的列队给移除。

```
//清理动画列队
$('#box').slideDown('slow', function () {$(this).clearQueue()});
```

六. 动画相关方法

很多时候需要停止正在运行中的动画，jQuery 为此提供了一个.stop()方法。它有两个可选参数：.stop(clearQueue, gotoEnd)；clearQueue 传递一个布尔值，代表是否清空未执行完的动画列队，gotoEnd 代表是否直接将正在执行的动画跳转到末状态。

```
//强制停止运行中的
$('.stop').click(function () {
    $('#box').stop();
});
//带参数的强制运行
$('.animate').click(function (){
       $('#box').animate({
            'left' : '300px'
        }, 1000);
        $('#box').animate({
            'bottom' : '300px'
        }, 1000);
        $('#box').animate({
```

```
            'width' : '300px'
        }, 1000);
        $('#box').animate({
            'height' : '300px'
        }, 1000);
    });
$('.stop').click(function () {
$('#box').stop(true ,true);
});
//如果在列队动画中停止，就会停止掉第一个列队动画，然后继续执行后面的列队动画
//第一个参数为 true,就会停止并清除后面的列队动画,动画完全停止,默认值为 false
//第二个参数，如果为 true，停止后会跳转到末尾的位置上。，默认值为 false
```

注意：第一个参数表示是否取消列队动画，默认为 false。如果参数为 true，当有列队动画的时候，会取消后面的列队动画。第二参数表示是否到达当前动画结尾，默认为 false。如果参数为 true，则停止后立即到达末尾处。

有时在执行动画或列队动画时，需要在运动之前有延迟执行，jQuery 为此提供了.delay()方法。这个方法可以在动画之前设置延迟，也可以在列队动画中间加上。

```
//开始延迟 1 秒钟，中间延迟 1 秒
$('.animate').click(function () {
$('#box').delay(1000).animate({
    'left' : '300px'
}, 1000);
$('#box').animate({
    'bottom' : '300px'
}, 1000);
$('#box').delay(1000).animate({
    'width' : '300px'
}, 1000);
$('#box').animate({
    'height' : '300px'
```

```
    }, 1000);
});
```

在选择器的基础章节中，我们提到过一个过滤器:animated，这个过滤器可以判断出当前运动的动画是哪个元素。 通过这个特点， 我们可以避免由于用户快速在某个元素执行动画时，由于动画积累而导致的动画和用户的行为不一致。

```
//递归执行自我，无限循环播放
$('#box').slideToggle('slow', function () {
    $(this).slideToggle('slow', arguments.callee);
});
//停止正在运动的动画，并且设置红色背景
$('.button').click(function(){
    $('div:animated').stop().css('background', 'red');
});
```

六．动画全局属性

jQuery 提供了两种全局设置的属性， 分别为： $.fx.interval， 设置每秒运行的帧数； $.fx.off，关闭页面上所有的动画。$.fx.interval 属性可以调整动画每秒的运行帧数，默认为 13 毫秒。数字越小越流畅，但可能影响浏览器性能。

```
//设置运行帧数为 1000 毫秒
$.fx.interval = 1000; //默认为 13
$('.button').click(function () {
    $('#box').toggle(3000);
});
```

$.fx.off 属性可以关闭所有动画效果，在非常低端的浏览器，动画可能会出现各种异常问题导致错误。而 jQuery 设置这个属性，就是用于关闭动画效果的。

```
//设置动画为关闭 true
$.fx.off = true; //默认为 false
```

补充：在.animate()方法中，还有一个参数，easing 运动方式，这个参数，大部分参数值需要通过插件来使用，在后面的课程中，会详细讲解。自带的参数有两个：swing(缓动)、linear(匀速)，默认为 swing。

```
<div id="box">box</div>
<div id="pox">pox</div>
#box {width:100px;height:100px;background:red;position:absolute;}
#pox {
    width:100px;height:100px;background:green;position:absolute;top:150px;
}
$('.button').click(function () {
    $('#box').animate({
    left : '800px'
}, 'slow', 'swing');
        $('#pox').animate({
        left : '800px'
    }, 'slow', 'linear');
});
```

第 12 章 Ajax

Ajax 全称为："Asynchronous JavaScript and XML"（异步 JavaScript 和 XML），它并不是 JavaScript 的一种单一技术，而是利用了一系列交互式网页应用相关的技术所形成的结合体。使用 Ajax，我们可以无刷新状态更新页面，并且实现异步提交，提升了用户体验。

一. Ajax 概述

Ajax 这个概念是由 Jesse James Garrett 在 2005 年发明的。它本身不是单一技术，是一串技术的集合，主要有：

1. JavaScript，通过用户或其他与浏览器相关事件捕获交互行为；

2. XMLHttpRequest 对象，通过这个对象可以在不中断其它浏览器任务的情况下向服务器发送请求；

3. 服务器上的文件，以 XML、HTML 或 JSON 格式保存文本数据；

4.其它 JavaScript，解释来自服务器的数据（比如 PHP 从 MySQL 获取的数据）并将其呈现到页面上。由于 Ajax 包含众多特性，优势与不足也非常明显。

优势主要以下几点：

1.不需要插件支持（一般浏览器且默认开启 JavaScript 即可）；

2.用户体验极佳（不刷新页面即可获取可更新的数据）；

3.提升 Web 程序的性能（在传递数据方面做到按需放松，不必整体提交）；

4.减轻服务器和带宽的负担（将服务器的一些操作转移到客户端）；

而 Ajax 的不足由以下几点：

1.不同版本的浏览器度 XMLHttpRequest 对象支持度不足(比如 IE5 之前)；

2.前进、后退的功能被破坏（因为 Ajax 永远在当前页，不会几率前后页面）；

3.搜索引擎的支持度不够（因为搜索引擎爬虫还不能理解 JS 引起变化数据的内容）；

4.开发调试工具缺乏(相对于其他语言的工具集来说 JS 或 Ajax 调试开发少的可怜）。

异步和同步：

使用 Ajax 最关键的地方，就是实现异步请求、接受响应及执行回调。那么异步与同步有什么区别呢？我们普通的 Web 程序开发基本都是同步的，意为执行一段程序才能执行下一段，类似电话中的通话，一个电话接完才能接听下个电话；而异步可以同时执行多条任务，感觉有多条线路，类似于短信，不会因为看一条短信而停止接受另一条短信。Ajax 也可以使用同步模式执行，但同步的模式属于阻塞模式，这样会导致多条线路执行时又必须一条一条执行，会让 Web 页面出现假死状态，所以，一般 Ajax 大部分采用异步模式。

二．load()方法

jQuery 对 Ajax 做了大量的封装，我们使用起来也较为方便，不需要去考虑浏览器兼容性。对于封装的方式，jQuery 采用了三层封装：最底层的封装方法为：$.ajax()，而通过这层封装了第二层有三种方法：.load()、$.get()和$.post()，最高层是$.getScript()和$.getJSON()方法。

.load()方法可以参数三个参数： url(必须， 请求 html 文件的 url 地址，参数类型为 String)、data(可选，发送的 key/value 数据，参数类型为 Object)、callback(可选，成功或失败的回调函数，参数类型为函数 Function)。如果想让 Ajax 异步载入一段 HTML 内容，我们只需要一个 HTML 请求的 url 即可。

```
//HTML
<input type="button" value="异步获取数据" />
<div id="box"></div>
//jQuery
$('input').click(function () {
    $('#box').load('test.html');
});
```

如果想对载入的 HTML 进行筛选，那么只要在 url 参数后面跟着一个选择器即可。

```
//带选择器的 url
$('input').click(function () {
    $('#box').load('test.html .my');//过滤 HTML 元素（加载后再过滤）
});
```

如果是服务器文件，比如.php。一般不仅需要载入数据，还需要向服务器提交数据，那么我们就可以使用第二个可选参数 data。向服务器提交数据有两种方式：get 和 post。

```
//不传递 data，则默认 get 方式
$('input').click(function () {
    $('#box').load('test.php?url=ycku');
});
//get 方式接受的 PHP
<?php
if ($_GET['url'] == 'ycku') {
    echo '瓢城 Web 俱乐部官网';
} else {
    echo '其他网站';
}
?>
//传递 data，则为 post 方式
$('input').click(function () {
    $('#box').load('test.php', {
```

```
    url : 'ycku'  //键值对形式默认提交方式是 POST
    });
});
//post 方式接受的 PHP
<?php
if ($_POST['url'] == 'ycku') {
    echo '瓢城 Web 俱乐部官网';
} else {
    echo '其他网站';
}
?>
```

在 Ajax 数据载入完毕之后，就能执行回调函数 callback，也就是第三个参数。回调函数也可以传递三个可选参数： responseText （请求返回） 、 textStatus（请求状态）、 XMLHttpRequest （XMLHttpRequest 对象） 。

```
$('input').click(function () {
    $('#box').load('test.php', {
    url : 'ycku'
}, function (response, status, xhr) {
    alert('返回的值为：' + response + '，状态为：' + status + '，
    状态是：' + xhr.statusText);
    });
});
```

注意：status 得到的值， 如果成功返回数据则为： success， 否则为： error。XMLHttpRequest 对象属于 JavaScript 范畴，可以调用一些属性如下：

属性名	说明
responseText	作为响应主体被返回的文本
responseXML	如果响应主体内容类型是"text/xml"或"application/xml"，则返回包含响应数据的 XML DOM 文档
status	响应的 HTTP 状态
statusText	HTTP 状态的说明

如果成功返回数据，那么 xhr 对象的 statusText 属性则返回'OK'字符串。除了'OK'的状态字符串，statusText 属性还提供了一系列其他的值，如下：

HTTP 状态码	状态字符串	说明
200	OK	服务器成功返回了页面
400	Bad Request	语法错误导致服务器不识别
401	Unauthorized	请求需要用户认证
404	Not found	指定的 URL 在服务器上找不到
500	Internal Server Error	服务器遇到意外错误，无法完成请求
503	ServiceUnavailable	由于服务器过载或维护导致无法完成请求

三．$.get()和$.post()

.load()方法是局部方法，因为他需要一个包含元素的 jQuery 对象作为前缀。而$.get()和$.post()是全局方法，无须指定某个元素。对于用途而言，.load()适合做静态文件的异步获取，而对于需要传递参数到服务器页面的，$.get()和$.post()更加合适。

第二个参数形式：

1.通过直接在 url 问号紧跟传参 （get）

```
$.get('test.php?url=ycku',function (response, status, xhr){})
```

2.字符串形式的键值对传参，然后自动转换为问号紧跟传参 （post，get）

```
$.get('test.php', 'url=ycku',function (response, status, xhr) {
        $('#box').html(response);
});
```

3.对象形式的键值对传参，然后自动转换为问号紧跟传参 **（post，get）**

```
$.get('test.php', {
    url : 'ycku'
},function (response, status, xhr) {
    $('#box').html(response);
});
```

$.get()方法有四个参数，前面三个参数和.load()一样，多了一个第四参数 type，即服务器返回的内容格式：包括 xml、html、script、json、jsonp 和 text。第一个参数为必选参数，后面三个为可选参数。

第四参数 type：

```
//使用$.get()异步返回 html 类型
$('input').click(function () {
$.get('test.php', {
    url : 'ycku'
    }, function (response, status, xhr) {
        if (status == 'success') {
            $('#box').html(response);
        }
    }) //type 自动转为 html
});
```

//后端处理本身是纯文本，如果强行按照 xml 或者 json 数据格式将无法解析数据

注意：第四参数 type 是指定异步返回的类型。一般情况下 type 参数是智能判断，并不需要我们主动设置，如果主动设置，则会强行按照指定类型格式返回。

```
//使用$.get()异步返回 xml
$('input').click(function () {
    $.get('test.xml', function (response, status, xhr) {
        $('#box').html($(response).find('root').find('url').text());
    }); //type 自动转为 xml
});
```

注意：如果载入的是 xml 文件，type 会智能判断。如果强行设置 html 类型返回，则会把 xml 文件当成普通数据全部返回，而不会按照 xml 格式解析数据。

```
//使用$.get()异步返回 json
$.get('test.json', function (response, status, xhr) {
    alert(response[0].url);
});

$('input').click(function () {
    $.post('test.php', {
        url : 'ycku'
    },function (response, status, xhr) {
        $('#box').html(response);
    }, 'json');//如果是 json 字符串一定要写上，是 json 对象默认不写
});
```

后端数据 JSON 对象格式：（不是 json 字符串）

```
[
    {
        "url" : "www.ycku.com"
    }
]
```

$.post()方法的使用和$.get()基本上一致，他们之间的区别也比较隐晦，基本都是背后的不同，在用户使用上体现不出。具体区别如下：

1. GET 请求是通过 URL 提交的，而 POST 请求则是 HTTP 消息实体提交的；
2. GET 提交有大小限制（2KB），而 POST 方式不受限制；
3. GET 方式会被缓存下来，可能有安全性问题，而 POST 没有这个问题；
4. GET 方式通过$_GET[]获取，POST 方式通过$_POST[]获取。

```
//使用$.post()异步返回 html
$.post('test.php', {
    url : 'ycku'
```

```
}, function (response, status, xhr) {
    $('#box').html(response);
});
```

四. $.getScript()和$.getJSON()

jQuery 提供了一组用于特定异步加载的方法：$.getScript()，用于加载特定的 JS 文件；$.getJSON()，用于专门加载 JSON 文件。有时我们希望能够特定的情况再加载 JS 文件，而不是一开始把所有 JS 文件都加载了，这时课时使用$.getScript()方法。

```
//点击按钮后再加载 JS 文件（需要的时候加载 js 文件）
$('input').click(function () {
    $.getScript('test.js');
});
```

$.getJSON()方法是专门用于加载 JSON 文件的，使用方法和之前的类似。

```
$('input').click(function () {
    $.getJSON('test.json', function (response, status, xhr) {
        alert(response[0].url);
    });
});
```

五. $.ajax()

$.ajax()是所有 ajax 方法中最底层的方法，所有其他方法都是基于$.ajax()方法的封装。这个方法只有一个参数，传递一个各个功能键值对的对象。

$.ajax()方法对象参数表

参数	类型	说明
url	String	发送请求的地址
type	String	请求方式：POST 或 GET，默认 GET

timeout	Number	设置请求超时的时间（毫秒）
data	Object 或 String	发送到服务器的数据，键值对字符串或对象
dataType	String	返回的数据类型，比如 html、xml、json 等
beforeSend	Function	发送请求前可修改 XMLHttpRequest 对象的函数
complete	Function	请求完成后调用的回调函数
success	Function	请求成功后调用的回调函数
error	Function	请求失败时调用的回调函数

global	Boolean	默认为 true，表示是否触发全局 Ajax
cache	Boolean	设置浏览器缓存响应，默认为 true。如果 dataType 类型为 script 或 jsonp 则为 false。
content	DOM	指定某个元素为与这个请求相关的所有回调函数的上下文。
contentType	String	指定请求内容的类型。默认为 application/x-www-form-urlencoded。
async	Boolean	是否异步处理。默认为 true，false 为同步处理
processData	Boolean	默认为 true，数据被处理为 URL 编码格式。如果为 false，则阻止将传入的数据处理为 URL 编码的格式。
dataFilter	Function	用来筛选响应数据的回调函数。

ifModified	Boolean	默认为 false，不进行头检测。如果为 true，进行头检测，当相应内容与上次请求改变时，请求被认为是成功的。
jsonp	String	指定一个查询参数名称来覆盖默认的 jsonp 回调参数名 callback。
username	String	在 HTTP 认证请求中使用的用户名
password	String	在 HTTP 认证请求中使用的密码
scriptCharset	String	当远程和本地内容使用不同的字符集时，用来设置 script 和 jsonp 请求所使用的字符集。
xhr	Function	用来提供 XHR 实例自定义实现的回调函数
traditional	Boolean	默认为 false，不使用传统风格的参数序列化。如为 true，则使用。

```
//$.ajax 使用
$('input').click(function () {
    $.ajax({
        type : 'POST', //这里可以换成 GET  //1 方式
        url : 'test.php', //   2 请求地址
        data : {           //3 发送数据
                url : 'ycku'
        },
        success : function (response, stutas, xhr) {
            $('#box').html(response);  //4 返回显示数据
        }
    });
});
```

注意：对于 data 属性，如果是 GET 模式，可以使用三种之前说所的三种形式。如果是 POST 模式可以使用之前的两种形式。

六．表单序列化

Ajax 用的最多的地方莫过于表单操作，而传统的表单操作是通过 submit 提交将数据传输到服务器端。如果使用 Ajax 异步处理的话，我们需要将每个表单元素逐个获取才方能提交。这样工作效率就大大降低。

```
//常规形式的表单提交
$('form input[type=button]').click(function () {
    $.ajax({
        type : 'POST',
        url : 'test.php',
        data : {
            user : $('form input[name=user]').val(),
            email : $('form input[name=email]').val()
        },
        success : function (response, status, xhr) {
```

```
            alert(response);
        }
    });
});
```

使用表单序列化方法.serialize()，会智能的获取指定表单内的所有元素。这样，在面对大量表单元素时，会把表单元素内容序列化为字符串，然后再使用 Ajax 请求。

```
//使用.serialize()序列化表单内容
$('form input[type=button]').click(function () {
    $.ajax({
        type : 'POST',
        url : 'test.php',
        data : $('form').serialize(),//得到字符串键值对
        success : function (response, status, xhr) {
            alert(response);
        }
    });
});
```

.serialize()方法不但可以序列化表单内的元素，还可以直接获取单选框、复选框和下拉列表框等内容。（decodeURIComponent 汉字解码）

```
//使用序列化得到选中的元素内容
$(':radio').click(function () {
    $('#box').html(decodeURIComponent($(this).serialize()));
});
```

除了.serialize()方法，还有一个可以返回 JSON 数据的方法：.serializeArray()。这个方法可以直接把数据整合成键值对的 JSON 对象。

```
$(':radio').click(function () {
    console.log($(this).serializeArray());
    var json = $(this).serializeArray();
    $('#box').html(json[0].value);
});
```

有时，我们可能会在同一个程序中多次调用$.ajax()方法。而它们很多参数都相同，这个时候我们课时使用 jQuery 提供的$.ajaxSetup()请求默认值来初始化参数。

```
$('form input[type=button]').click(function () {
    $.ajaxSetup({
        type : 'POST',
        url : 'test.php',
        data : $('form').serialize()
    });
    $.ajax({
        success : function (response, status, xhr) {
            alert(response);
        }
    });
});
```

在使用 data 属性传递的时候，如果是以对象形式传递键值对，可以使用$.param()方法将对象转换为字符串键值对格式。

```
var obj = {a : 1, b : 2, c : 3};
var form = $.param(obj);
alert(form);
```

注意：使用$.param()将对象形式的键值对转为 URL 地址的字符串键值对，可以更加稳定准确的传递表单内容。因为有时程序对于复杂的序列化解析能力有限，所以直接传递 obj 对象要谨慎。

第十三章：Ajax 进阶

在 Ajax 课程中，我们了解了最基本的异步处理方式。本章，我们将了解一下 Ajax 的一些全局请求事件、跨域处理和其他一些问题。

一. 加载请求

在 Ajax 异步发送请求时，遇到网速较慢的情况，就会出现请求时间较长的问题。而超过一定时间的请求，用户就会变得不再耐烦而关闭页面。而如果在请求期间能给用户一些提示，比如：正在努力加载中...，那么相同的请求时间会让用户体验更加的好一些。jQuery 提供了两个全局事件，.ajaxStart()和.ajaxStop()。这两个全局事件，只要用户触发了 Ajax，请求开始时（未完成其他请求）激活.ajaxStart()，请求结束时（所有请求都结束了）激活.ajaxStop()。

```
//请求加载提示的显示和隐藏
$('.loading').ajaxStart(function () {
    $(this).show();
}).ajaxStop(function () {
    $(this).hide();
});
```

注意：以上代码在 jQuery1.8 及以后的版本不在有效，需要使用 jquery-migrate 向下兼容才能运行。新版本中，必须绑定在 document 元素上。

```
$(document).ajaxStart(function () {
    $('.loading').show();
}).ajaxStop(function () {
    $('.loading').hide();
});
//如果请求时间太长，可以设置超时
$.ajax({
    timeout : 500
});
//如果某个 ajax 不想触发全局事件，可以设置取消
$.ajax({
    global : false
});
```

二．错误处理

Ajax 异步提交时，不可能所有情况都是成功完成的，也有因为代码异步文件错误、网络错误导致提交失败的。这时，我们应该把错误报告出来，提醒用户重新提交或提示开发者进行修补。

在之前高层封装中是没有回调错误处理的，比如$.get()、$.post()和.load()。所以，早期的方法通过全局.ajaxError()事件方法来返回错误信息。而在 jQuery1.5 之后，可以通过连缀处理使用局部.error()方法即可。而对于$.ajax()方法，不但可以用这两种方法，还有自己的属性方法 error : function () {}。

```
//$.ajax()使用属性提示错误
$.ajax({
    type : 'POST',
    url : 'test1.php',
    data : $('form').serialize(),
    success : function (response, status, xhr) {
        $('#box').html(response);
    },
    error : function (xhr, ) {
        alert(xhr.status + ':' + xhr.statusText);
    }
});
//$.post()使用连缀.error()方法提示错误，连缀方法将被.fail()取代
$.post('test1.php').error(function (xhr, status, info) {
    alert(xhr.status + ':' +xhr.statusText);
    alert(status + ':' + info);
});
//$.post()使用全局.ajaxError()事件提示错误
$(document).ajaxError(function (event, xhr, settings, infoError) {
    alert(xhr.status + ':' +xhr.statusText);
    alert(settings+ ':' + info);
});
```

查看对象中属性

```
$(document).ajaxError(function (event, xhr, settings, errorType) {
        alert(event.type);
        alert(event.target);
        for (var i in event) {
            document.write(i + '<br />');
        }
        alert(settings);
        for (var i in settings) {
            document.write(i + '<br />');
        }
        alert(settings.url);
        alert(settings.type);
        alert(info);
    });
```

三. 请求全局事件

jQuery 对于 Ajax 操作提供了很多全局事件方法，.ajaxStart()、.ajaxStop()，.ajaxError()等事件方法。他们都属于请求时触发的全局事件，除了这些，还有一些其他全局事件：.ajaxSuccess()，对应一个局部方法：.success()，请求成功完成时执行。.ajaxComplete()，对应一个局部方法：.complete()，请求完成后注册一个回调函数。.ajaxSend()，没有对应的局部方法，只有属性 beforeSend，请求发送之前要绑定的数。

```
//$.post()使用局部方法.success()
$.post('test.php', $('form').serialize(), function (response, status, xhr) {
    $('#box').html(response);
}).success(function (response, status, xhr) {
    alert(response);
});
//$.post()使用全局事件方法.ajaxSuccess()
$(document).ajaxSuccess(function (event, xhr, settings) {
```

```
    alert(xhr.responseText);
});
```

注意：全局事件方法是所有 Ajax 请求都会触发到，并且只能绑定在 document 上。而局部方法，则针对某个 Ajax。对于一些全局事件方法的参数，大部分为对象，而这些对象有哪些属性或方法能调用，可以通过遍历方法得到。

```
//遍历 settings 对象的属性
$(document).ajaxSuccess(function (event, xhr, settings) {
    for (var i in settings) {
        alert(i);
    }
});
//$.post()请求完成的局部方法.complete()
$.post('test.php', $('form').serialize(), function (response, status, xhr) {
    alert('成功');
}).complete(function (xhr,status) {
    alert('完成');
});
//$.post()请求完成的全局方法.ajaxComplete()
$(document).ajaxComplete(function (event, xhr, settings) {
    alert('完成');
});
//$.post()请求发送之前的全局方法.ajaxSend()
$(document).ajaxSend(function (event, xhr, settings) {
    alert('发送请求之前');
});

//执行顺序：
$(document).ajaxSend(function () {
        alert('发送请求之前执行');   //1
    }).ajaxComplete(function () {
```

```
        alert('请求完成后，不管是否失败成功');  //3
    }).ajaxSuccess(function () {
        alert('请求成功后');  //2
    }).ajaxError(function () {
        alert('请求失败后'); //2
    });

//$.ajax()方法，可以直接通过属性设置即可。

$.ajax({
    type : 'POST',
    url : 'test.php',
    data : $('form').serialize(),
    success : function (response, status, xhr) {
        $('#box').html(response);
    },
    complete : function (xhr, status) {
        alert('完成' + ' - ' + xhr.responseText + ' - ' + status);
    },
    beforeSend : function (xhr, settings) {
        alert('请求之前' + ' - ' + xhr.readyState + ' - ' + settings.url);
    }
});
```

注意：在 jQuery1.5 版本以后，使用.success()、.error()和.complete()连缀的方法，可以用.done()、.fail()和.always()取代。

四. JSON 和 JSONP

如果在同一个域下，$.ajax()方法只要设置 dataType 属性即可加载 JSON 文件。而在非同域下，可以使用 JSONP，但也是有条件的。

```
//$.ajax()加载 JSON 文件
$.ajax({
    type : 'POST',
    url : 'test.json',
    dataType : 'json',
    success : function (response, status, xhr) {
        alert(response[0].url);//json 对象
    }
});
```

注意：php ：

```
 $_arr = array('a'=>1, 'b'=>2, 'c'=>3);$_result = json_encode($_arr);
echo $_result; //{ “a” :1, “b” :2, “c” :3 }
$.ajax({
        type : 'POST',
        url : 'jsonp.php',
        dataType : 'json',  //返回 json 字符串一定要转成 json 对象
        success : function (response, status, xhr) {
            //alert(response);
            alert(response.a);
        }
});
```

如果想跨域操作文件的话，我们就必须使用 JSONP。JSONP(JSON with Padding)是一个非官方的协议，它允许在服务器端集成 Script tags 返回至客户端，通过 javascript callback 的形式实现跨域访问（这仅仅是 JSONP 简单的实现形式） 。

```
//跨域的 PHP 端文件
<?php
$arr = array('a'=>1,'b'=>2,'c'=>3,'d'=>4,'e'=>5);$result = json_encode($arr);
$callback = $_GET['callback'];
echo $callback."($result)";
?>
```

```
//$.getJSON()方法跨域获取 JSON
$.getJSON('http://www.li.cc/test.php?callback=?', function (response) {
    console.log(response);
});
//$.ajax()方法跨域获取 JSON
$.ajax({
    url : 'http://www.li.cc/test.php',
    dataType : 'jsonp', //不带参数指定类型
    success : function (response, status, xhr) {
        console.log(response);
        alert(response.a);
    }
});
$.ajax({
        type : 'POST',
        url : 'http://www.li.cc/jsonp2.php?callback=?',
        dataType : 'json',
        success : function (response, status, xhr) {
            //alert(response);
            console.log(response);
            alert(response.a);
        }
});
```

注意：这里的 URL 如果不想后面跟着?callback=?，那么可以给$.ajax()方法增加一个属性即可。

```
//使用 jsonp 属性
$.ajax({
    url : 'http://www.li.cc/test.php',
    jsonp : 'callback'
});
```

五. jqXHR 对象

在之前，我们使用了局部方法：.success()、.complete()和.error()。这三个局部方法并不是 XMLHttpRequest 对象调用的， 而是$.ajax()之类的全局方法返回的对象调用的。 这个对象，就是 jqXHR 对象，它是原生对象 XHR 的一个超集。

```
//获取 jqXHR 对象，查看属性和方法
var jqXHR = $.ajax({
    type : 'POST',
    url : 'test.php',
    data : $('form').serialize()
});
for (var i in jqXHR) {
    document.write(i + '<br />');
}
```

注意：如果使用 jqXHR 对象的话，那么建议用.done()、.always()和.fail()代替.success()、.complete()和.error()。以为在未来版本中，很可能将这三种方法废弃取消。

```
//成功后回调函数
jqXHR.done(function (response) {
    $('#box').html(response);
});
```

使用 jqXHR 的连缀方式比$.ajax()的属性方式有三大好处：

1. 可连缀操作，可读性大大提高；
2. 可以多次执行同一个回调函数；
3. 为多个操作指定回调函数；

```
//同时执行多个成功后的回调函数
jqXHR.done().done();
```

```
//多个操作指定回调函数
var jqXHR = $.ajax('test.php');
var jqXHR2 = $.ajax('test2.php');
$.when(jqXHR, jqXHR2).done(function (r1,r2) {
    alert(r1[0]);
    alert(r2[0]);
});
```

第十四章：工具函数

工具函数是指直接依附于 jQuery 对象，针对 jQuery 对象本身定义的方法，即全局性的函数。它的作用主要是提供比如字符串、数组、对象等操作方面的遍历。

一. 字符串操作（javascript 中已经存在更多对字符串的操作，JQuery 是封装此函数）

在 jQuery 中，字符串的工具函数只有一个，就是去除字符串左右空格的工具函数：$.trim()。

```
//$.trim()去掉字符串两边空格
var str = ' jQuery ';
alert(str);
alert($.trim(str));
```

二. 数组和对象操作

jQuery 为处理数组和对象提供了一些工具函数， 这些函数可以便利的给数组或对象进行遍历、筛选、搜索等操作。

```
//$.each()遍历数组
var arr = ['张三', '李四', '王五', '马六'];
$.each(arr, function (index, value) {
```

```
    $('#box').html($('#box').html() + index + '.' + value + '<br />');

});
//$.each()遍历对象
$.each($.ajax(), function (name, fn) {
    $('#box').html($('#box').html() + name + '.' + '<br /><br />');
})
```

注意：$.each()中 index 表示数组元素的编号，默认从 0 开始。

```
//$.grep()数据筛选
var arr = [5,2,9,4,11,57,89,1,23,8];
var arrGrep = $.grep(arr, function (element, index) {
    return element < 6 && index < 5;  //index 是下标  element 元素值
});
alert(arrGrep);
```

注意：$.grep()方法的 index 是从 0 开始计算的。

```
//$.map()修改数据
var arr = [5,2,9,4,11,57,89,1,23,8];
var arrMap = $.map(arr, function (element, index) {
    if (element < 6 && index < 5) {
        return element + 1;
    }
});
alert(arrMap);
//$.inArray()获取查找到元素的下标
var arr = [5,2,9,4,11,57,89,1,23,8];
var arrInArray = $.inArray(1, arr);
alert(arrInArray);
```

注意：$.inArray()的下标从 0 开始计算。

```
//$.merge()合并两个数组
var arr = [5,2,9,4,11,57,89,1,23,8];
```

```
var arr2 = [23,2,89,3,6,7];
alert($.merge(arr, arr2));
//$.unique()删除重复的 DOM 元素
<div></div>
<div></div>
<div class="box"></div>
<div class="box"></div>
<div class="box"></div>
<div></div>
var divs = $('div').get();
divs = divs.concat($('.box').get());
alert($(divs).size());
$.unique(divs);
alert($(divs).size());
//.toArray()合并多个 DOM 元素组成数组
alert($('li').toArray());
```

三．测试操作

在 jQuery 中，数据有着各种类型和状态。有时，我们希望能通过判断数据的类型和状态做相应的操作。jQuery 提供了五组测试用的工具函数。

测试工具函数

函数名	说明
$.isArray(obj)	判断是否为数组对象，是返回 true
$.isFunction(obj)	判断是否为函数，是返回 true
$.isEmptyObject(obj)	判断是否为空对象，是返回 true
$.isPlainObjet(obj)	判断是否为纯粹对象，是返回 true
$.contains(obj)	判断 DOM 节点是否含另一个 DOM 节点，是返回 true
$.type(data)	判断数据类型
$.isNumeric(data)	判断数据是否为数值
$.isWindow(data)	判断数据是否为 window 对象

```
//判断是否为数组对象
var arr = [1,2,3];
alert($.isArray(arr));
//判断是否为函数
var fn = function () {};
alert($.isFunction(fn));
//判断是否为空对象
var obj = {}
alert($.isEmptyObject(obj));
//判断是否由{}或 new Object()创造出的对象
var obj = window;
alert($.isPlainObject(obj));
```

注意：如果使用 new Object('name');传递参数后，返回类型已不是 Object，而是字符串，所以就不是纯粹的原始对象了。

```
//判断第一个 DOM 节点是否含有第二个 DOM 节点
alert($.contains($('#box').get(0), $('#pox').get(0)));
//$.type()检测数据类型
alert($.type(window));
//$.isNumeric 检测数据是否为数值
alert($.isNumeric(5.25));
//$.isWindow 检测数据对象是否为 window 对象
alert($.isWindow(window));
```

四. URL 操作

URL 地址操作，在之前的 Ajax 章节其实已经讲到过。只有一个方法：$.param()，将对象的键值对转化为 URL 键值对字符串形式。

```
//$.param()将对象键值对转换为 URL 字符串键值对
var obj = {
    name : 'Lee',
    age : 100
```

```
};
alert($.param(obj));
```

五．浏览器检测

由于在早期的浏览器中，分 IE 和 W3C 浏览器。而 IE678 使用的覆盖率还很高，所以，早期的 jQuery 提供了$.browser 工具对象。而现在的 jQuery 已经废弃删除了这个工具对象，如果还想使用这个对象来获取浏览器版本型号的信息，可以使用兼容插件。

$.browser 对象属性

属性	说明
webkit	判断 webkit 浏览器，如果是则为 true
mozilla	判断 mozilla 浏览器，如果是则为 true
safari	判断 safari 浏览器，如果是则为 true
opera	判断 opera 浏览器，如果是则为 true
msie	判断 IE 浏览器，如果是则为 true
version	获取浏览器版本号

```
//获取火狐浏览器和版本号
alert($.browser.mozilla + ':' + $.browser.version);
```

注意：火狐采用的是 mozilla 引擎，一般就是指火狐；而谷歌 Chrome 采用的引擎是 webkit，一般验证 Chrome 就用 webkit。

还有一种浏览器检测，是对浏览器内容的检测。比如：W3C 的透明度为 opacity，而 IE 的透明度为 alpha。这个对象是$.support。

$.support 对象部分属性

属性名	说明
hrefNormalized	如果浏览器从 getAttribute("href")返回的是原封不动的结果，则返回 true。在 IE 中会返回 false，因为他

	的 URLs 已经常规化了
htmlSerialize	如果浏览器通过 innerHTML 插入链接元素的时候会序列化这些链接，则返回 true，目前 IE 中返回 false
leadingWhitespace	如果在使用 innerHTML 的时候浏览器会保持前导空白字符，则返回 true，目前在 IE 6-8 中返回 false
objectAll	如果在某个元素对象上执行 getElementsByTagName("*")会返回所有子孙元素，则为 true，目前在 IE 7 中为 false
opacity	如果浏览器能适当解释透明度样式属性，则返回 true，目前在 IE 中返回 false，因为他用 alpha 滤镜代替
scriptEval	使用 appendChild/createTextNode 方法插入脚本代码时，浏览器是否执行脚本，目前在 IE 中返回 false，IE 使用 .text 方法插入脚本代码以执行
style	如果 getAttribute("style")返回元素的行内样式，则为 true。目前 IE 中为 false，因为他用 cssText 代替
tbody	如果浏览器允许 table 元素不包含 tbody 元素，则返回 true。目前在 IE 中会返回 false，他会自动插入缺失的 tbody
Ajax	如果浏览器支持 ajax 操作，返回 true

```
//$.support.ajax 判断是否能创建 ajax
alert($.support.ajax);
//$.support.opacity 设置不同浏览器的透明度
if ($.support.opacity == true) {
    $('#box').css('opacity', '0.5');
} else {
    $('#box').css('filter', 'alpha(opacity=50)');
}
```

注意：由于 jQuery 越来越放弃低端的浏览器，所以检测功能在未来使用频率也越来越低。所以，$.brower 已被废弃删除，而$.support.boxModel 检测 W3C 或 IE 盒子也被删除。并且 http://api.jquery.com/jQuery.support/官网也不提供属性列表和解释，给出一个 Modernizr 第三方小工具来辅组检测。

六．其他操作

jQuery 提供了一个预备绑定函数上下文的工具函数：$.proxy()。这个方法，可以解决诸如外部事件触发调用对象方法时 this 的指向问题。

```
//$.proxy()调整 this 指向
var obj = {
    name : 'Lee',
    test : function () {
        alert(this.name);
    }
};
$('#box').click(obj.test); //指向的 this 为#box 元素
$('#box').click($.proxy(obj, 'test')); //指向的 this 为方法属于对象 box
```

第十五章：插件

插件(Plugin)也成为 jQuery 扩展(Extension)，是一种遵循一定规范的应用程序接口编写出来的程序。目前 jQuery 插件已超过几千种，由来自世界各地的开发者共同编写、验证和完善。而对于 jQuery 开发者而言，直接使用这些插件将快速稳定架构系统，节约项目成本。

一．插件概述

插件是以 jQuery 的核心代码为基础，编写出复合一定规范的应用程序。也就是说，插件也是 jQuery 代码，通过 js 文件引入的方式植入即可。

插件的种类很多，主要大致可以分为：UI 类、表单及验证类、输入类、特效类、Ajax 类、滑动类、图形图像类、导航类、综合工具类、动画类等等。

引入插件是需要一定步骤的，基本如下：

1.必须先引入 jquery.js 文件，而且在所有插件之前引入；

2.引入插件；

3.引入插件的周边，比如皮肤、中文包等。

二. 验证插件

Validate.js 是 jQuery 比较优秀的表单验证插件之一。这个插件有两个 js 文件，一个是主文件，一个是中文包文件。使用的时候，可以使用 min 版本；在这里，为了教学，我们未压缩版本。

验证插件包含的两个文件分别为：

jquery.validate.js 和 jquery.validate.messages_zh.js。

```
//HTML 内容
<script type="text/javascript" src="jquery.validate.js"></script>
<script type="text/javascript" src="jquery.validate.messages_zh.js">
</script>
<form><p>用 户 名： <input type="text" class="required" name="username"
minlength="2" />*</p><p>电子邮件：<input type="text" class="required email"
name="email" /> *</p>
<p>网 址：<input type="text" class="url" name="url" /></p>
<p><input type="submit" value="提交" /></p>
</form>
//jQuery 代码
$(function () {
    $('form').validate();
});
```

只要通过 form 元素的 jQuery 对象调用 validate()方法，就可以实现“必填”、“不能小于两位”、“电子邮件不正确”、“网址不正确”等验证效果。除了 js 端的 validate()方法调用，表单处也需要相应设置才能最终得到验证效果。

1. 必填项：在表单设置 class="required";
2. 不得小于两位：在表单设置 minlength="2";
3. 电子邮件：在表单中设置 class="email";
4. 网址：在表单中设置 class="url"。

注意：本章就简单的了解插件的使用，并不针对某个功能的插件进行详细讲解。比如验证插件 validate.js，它类似与 jQuery 一样，同样具有各种操作方法和功能，需要进行类似手册一样的查询和讲解。所以，我们会在项目中再去详细讲解使用到的插件。

三．自动完成插件

所谓自动完成，就是当用户输入部分字符的时候，智能的搜索出包含字符的全部内容。比如：输入 a，把匹配的内容列表展示出来。

```
//HTML 内容
<script type="text/javascript" src="jquery.autocomplete.js"></script>
<script type="text/javascript" src="jquery-migrate-1.2.1.js"></script>
<link rel="stylesheet" href="jquery.autocomplete.css" type="text/css" />
//jQuery 代码
var user = ['about', 'family', 'but', 'black'];
$('form input[name=username]').autocomplete(user, {
minChars : 0 //双击显示全部数据
});
```

注意：这个自动完成插件使用的 jQuery 版本较老，用了一些已被新版本的 jQuery 废弃

删除的方法，这样必须要向下兼容才能有效。所以，去查找插件的时候，要注意一下他坚持

的版本。

四．自定义插件

前面我们使用了别人提供好的插件，使用起来非常的方便。如果市面上找不到自己满意的插件，并且想自己封装一个插件提供给别人使用。那么就需要自己编写一个 jQuery 插件

了。

按照功能分类，插件的形式可以分为一下三类：

1.封装对象方法的插件；　（也就是基于某个 DOM 元素的 jQuery 对象，局部性）

2.封装全局函数的插件；　（全局性的封装）

3.选择器插件。(类似与.find())

经过日积月累的插件开发，开发者逐步约定了一些基本要点，以解决各种因为插件导致的冲突、错误等问题，包括如下：

1.插件名推荐使用 jquery.[插件名].js，以免和其他 js 文件或者其他库相冲突；

2.局部对象附加 jquery.fn 对象上，全局函数附加在 jquery 上；

3. 插件内部，this 指向是当前的局部对象；

4. 可以通过 this.each 来遍历所有元素；

5. 所有的方法或插件，必须用分号结尾，避免出现问题；

6. 插件应该返回的是 jQuery 对象，以保证可链式连缀；

7. 避免插件内部使用$，如果要使用，请传递 jQuery 进去。

按照以上的要点，我们开发一个局部或全局的导航菜单的插件。只要导航的<li>标签内部嵌入要下拉的<ul>，并且 class 为 nav，即可完成下拉菜单。

```
//HTML 部分
<ul class="list">
<li>导航列表
<ul class="nav">
<li>导航列表 1</li>
<li>导航列表 1</li>
<li>导航列表 1</li>
<li>导航列表 1</li>
<li>导航列表 1</li>
<li>导航列表 1</li>
</ul>
</li>
<li>导航列表
<ul class="nav">
<li>导航列表 2</li>
<li>导航列表 2</li>
<li>导航列表 2</li>
<li>导航列表 2</li>
<li>导航列表 2</li>
<li>导航列表 2</li>
</ul>
</li>
</ul>
```

```
//jquery.nav.js 部分   （局部）
;(function ($) {
    $.fn.extend({ 利用JQuery扩展插件的局部($.fn.extend)方法添加插件
        'nav' : function (color) {
            $(this).find('.nav').css({ 一定要进来标识选中的当前对象
                                       $('.nav')->$(this).find('.nav')
                listStyle : 'none',
                margin : 0,
                padding : 0,
                display : 'none',
                color : color
            }); 一定要进来标识选中的当前对象
                $('.nav')->$(this).find('.nav')
            $(this).find('.nav').parent().hover(function () {
                $(this).find('.nav').slideDown('normal');
            }, function () {
                $(this).find('.nav').stop().slideUp('normal');
            });
            return this;
        }
    });
})(jQuery);避免插件内部使用$，如果要使用，请传递 jQuery 进去。
$('.list').eq(0).nav('orange') //调用
局部只针对选中的DOM节点才会有导航下拉的效果

//jquery.nav.js 部分   （全局）
;(function ($) {
    $.extend({ 利用JQuery扩展插件的全局方法($.extend)添加插件
自定义插件名 'nav' : function (color) {
            $('.nav').css({
                'list-style' : 'none',
                'margin' : 0,
                'padding' : 0,
```

```
$(function () {
    //alert($('.list').eq(0).nav('orange'));

    //自定义
    $('.nav').css({
        listStyle: 'none',
        margin: 0,
        padding: 0,
        display: 'none',
        color: color
    });
    $('.nav').parent().hover(function() {
        $(this).find('.nav').slideDown('normal');
    },
    function() {
        $(this).find('.nav').stop().slideUp('normal');
    });
    //stop鼠标离开就停止动画,让它不再往下展开
});
```

```
                'display' : 'none',
                'color' : color
            });

            $('.nav').parent().hover(function () {
                $(this).find('.nav').slideDown('normal');
            }, function () {
                $(this).find('.nav').stop().slideUp('normal');
            });
            return this;
        }
    });
})(jQuery);

//调用
$.nav();
```

引入插件文件在调用即可，此方法为全局，缺点时主要满足定义的文档结构就会有导航下拉的效果。