jQuery的加载

一般采用下面的写法，在网页中加载jQuery。

```
<script type="text/javascript"
  src="//code.jquery.com/jquery-1.11.0.min.js">
</script>
<script>
window.jQuery ||
  document.write(
    '<script src="js/jquery-1.11.0.min.js" type="text/javascript">
<\/script>'
  );
</script>
```

上面代码有两点需要注意。一是采用CDN加载。如果CDN加载失败，则退回到本地加载。二是采用协议无关的加载网址（使用双斜线表示），同时支持http协议和https协议。
目前常用的jQuery CDN有以下这些。

- Google CDN
- Microsoft CDN
- jQuery CDN
- CDNJS CDN
- jsDelivr CDN

上面这段代码最好放到网页尾部。如果需要支持IE 6/7/8，就使用jQuery 1.x版，否则使用最新版。

jQuery基础

jQuery对象

jQuery最重要的概念，就是`jQuery`对象。它是一个全局对象，可以简写为美元符号`$`。也就是说，`jQuery`和`$`两者是等价的。
在网页中加载jQuery函数库以后，就可以使用jQuery对象了。jQuery的全部方法，都定义在这个对象上面。

```
var listItems = jQuery('li');
// or
var listItems = $('li');
```

上面两行代码是等价的，表示选中网页中所有的`li`元素。

jQuery构造函数

`jQuery`对象本质上是一个构造函数，主要作用是返回`jQuery`对象的实例。比如，上面代码表面上是选中`li`元素，实际上是返回对应于`li`元素的`jQuery`实例。因为只有这样，才能在DOM对象上使用jQuery提供的各种方法。

```
$('body').nodeType
// undefined

$('body') instanceof jQuery
// true
```

上面代码表示，由于jQuery返回的不是DOM对象，所以没有DOM属性`nodeType`。它返回的是jQuery对象的实例。

jQuery构造函数可以多种参数，返回不同的值。

（1）CSS选择器作为参数

jQuery构造函数的参数，主要是CSS选择器，就像上面的那个例子。下面是另外一些CSS选择器的例子。

```
$("*")
$("#lastname")
$(".intro")
$("h1,div,p")
$("p:last")
$("tr:even")
$("p:first-child")
$("p:nth-of-type(2)")
$("div + p")
$("div:has(p)")
$(":empty")
$("[title^='Tom']")
```

本书不讲解CSS选择器，请读者参考有关书籍和jQuery文档。

除了CSS选择器，jQuery还定义了一些自有的选择器，比如`contains`选择器用来选择包含特定文本的元素。下面是一个例子。

```
var search = $('#search').val();
$('div:not(:contains("' + search + '"))').hide();
```

上面代码用来选中包含搜索框输入文本的元素。

（2）DOM对象作为参数

jQuery构造函数的参数，还可以是DOM对象。它也会被转为jQuery对象的实例。

```
$(document.body) instanceof jQuery
// true
```

上面代码中，jQuery的参数不是CSS选择器，而是一个DOM对象，返回的依然是jQuery对象的实例。

如果有多个DOM元素要转为jQuery对象的实例，可以把DOM元素放在一个数组里，输入jQuery构造函数。

```
$([document.body, document.head])
```

（3）HTML字符串作为参数

如果直接在jQuery构造函数中输入HTML字符串，返回的也是jQuery实例。

```
$('<li class="greet">test</li>')
```

上面代码从HTML代码生成了一个jQuery实例，它与从CSS选择器生成的jQuery实例完全一样。唯一的区别就是，它对应的DOM结构不属于当前文档。

上面代码也可以写成下面这样。

```
$( '<li>', {
  html: 'test',
  'class': 'greet'
});
```

上面代码中，由于`class`是javaScript的保留字，所以只能放在引号中。

通常来说，上面第二种写法是更好的写法。

```
$('<input class="form-control" type="hidden" name="foo" value="bar" />')

// 相当于
$('<input/>', {
  'class': 'form-control',
```

```
  type: 'hidden',
  name: 'foo',
  value: 'bar'
})

// 或者
$('<input/>')
.addClass('form-control')
.attr('type', 'hidden')
.attr('name', 'foo')
.val('bar')
```

由于新增的DOM节点不属于当前文档，所以可以用这种写法预加载图片。

```
$.preloadImages = function () {
  for (var i = 0; i < arguments.length; i++) {
    $('<img>').attr('src', arguments[i]);
  }
};

$.preloadImages('img/hover-on.png', 'img/hover-off.png');
```

（4）第二个参数

默认情况下，jQuery将文档的根元素（`html`）作为寻找匹配对象的起点。如果要指定某个网页元素（比如某个`div`元素）作为寻找的起点，可以将它放在jQuery函数的第二个参数。

```
$('li', someElement);
```

上面代码表示，只寻找属于someElement对象下属的li元素。someElement可以是jQuery对象的实例，也可以是DOM对象。

jQuery构造函数返回的结果集

jQuery的核心思想是“先选中某些网页元素，然后对其进行某种处理”（find something, do something），也就是说，先选择后处理，这是jQuery的基本操作模式。所以，绝大多数jQuery操作都是从选择器开始的，返回一个选中的结果集。

（1）length属性

jQuery对象返回的结果集是一个类似数组的对象，包含了所有被选中的网页元素。查看该对象的length属性，可以知道到底选中了多少个结果。

```
if ( $('li').length === 0 ) {
 console.log('不含li元素');
}
```

上面代码表示，如果网页没有li元素，则返回对象的length属性等于0。这就是测试有没有选中的标准方法。所以，如果想知道jQuery有没有选中相应的元素，不能写成下面这样。

```
if ($('div.foo')) { ... }
```

因为不管有没有选中，jQuery构造函数总是返回一个实例对象，而对象的布尔值永远是true。使用length属性才是判断有没有选中的正确方法。

```
if ($('div.foo').length) { ... }
```

（2）下标运算符

jQuery选择器返回的是一个类似数组的对象。但是，使用下标运算符取出的单个对象，并不是jQuery对象的实例，而是一个DOM对象。

```
$('li')[0] instanceof jQuery // false
```

```
$('li')[0] instanceof Element // true
```

上面代码表示，下标运算符取出的是Element节点的实例。所以，通常使用下标运算符将jQuery实例转回DOM对象。

（3）is方法

is方法返回一个布尔值，表示选中的结果是否符合某个条件。这个用来验证的条件，可以是CSS选择器，也可以是一个函数，或者DOM元素和jQuery实例。

```
$('li').is('li') // true

$('li').is($('.item'))

$('li').is(document.querySelector('li'))

$('li').is(function() {
      return $("strong", this).length === 0;
});
```

（4）get方法

jQuery实例的get方法是下标运算符的另一种写法。

```
$('li').get(0) instanceof Element // true
```

（5）eq方法

如果想要在结果集取出一个jQuery对象的实例，不需要取出DOM对象，则使用eq方法，它的参数是实例在结果集中的位置（从0开始）。

```
$('li').eq(0) instanceof jQuery // true
```

由于eq方法返回的是jQuery的实例，所以可以在返回结果上使用jQuery实例对象的方法。

（6）each方法，map方法

这两个方法用于遍历结果集，对每一个成员进行某种操作。

each方法接受一个函数作为参数，依次处理集合中的每一个元素。

```
$('li').each(function( index, element) {
  $(element).prepend( '<em>' + index + ': </em>' );
});


// <li>Hello</li>
// <li>World</li>
// 变为
// <li><em>0: </em>Hello</li>
// <li><em>1: </em>World</li>
```

从上面代码可以看出，作为each方法参数的函数，本身有两个参数，第一个是当前元素在集合中的位置，第二个是当前元素对应的DOM对象。

map方法的用法与each方法完全一样，区别在于each方法没有返回值，只是对每一个元素执行某种操作，而map方法返回一个新的jQuery对象。

```
$("input").map(function (index, element){
    return $(this).val();
})
.get()
.join(", ")
```

上面代码表示，将所有input元素依次取出值，然后通过get方法得到一个包含这些值的数组，最后通过数组的join方法返回一个逗号分割的字符串。

（8）内置循环

jQuery默认对当前结果集进行循环处理，所以如果直接使用jQuery内置的某种方法，each和map方法是不必

要的。

```
$(".class").addClass("highlight");
```

上面代码会执行一个内部循环，对每一个选中的元素进行addClass操作。由于这个原因，对上面操作加上each方法是不必要的。

```
$(".class").each(function(index,element){
  $(element).addClass("highlight");
});

// 或者

$(".class").each(function(){
 $(this).addClass("highlight");
});
```

上面代码的each方法，都是没必要使用的。
由于内置循环的存在，从性能考虑，应该尽量减少不必要的操作步骤。

```
$(".class").css("color", "green").css("font-size", "16px");

// 应该写成

$(".class").css({
  "color": "green",
  "font-size": "16px"
});
```

链式操作

jQuery最方便的一点就是，它的大部分方法返回的都是jQuery对象，因此可以链式操作。也就是说，后一个方法可以紧跟着写在前一个方法后面。

```
$('li').click(function (){
    $(this).addClass('clicked');
})
.find('span')
.attr( 'title', 'Hover over me' );
```

$(document).ready()

$(document).ready方法接受一个函数作为参数，将该参数作为document对象的DOMContentLoaded事件的回调函数。也就是说，当页面解析完成（即下载完</html>标签）以后，在所有外部资源（图片、脚本等）完成加载之前，该函数就会立刻运行。

```
$( document ).ready(function() {
  console.log( 'ready!' );
});
```

上面代码表示，一旦页面完成解析，就会运行ready方法指定的函数，在控制台显示“ready!”。
该方法通常作为网页初始化手段使用，jQuery提供了一种简写法，就是直接把回调函数放在jQuery对象中。

```
$(function() {
  console.log( 'ready!' );
});
```

上面代码与前一段代码是等价的。

$.noConflict方法

jQuery使用美元符号（$）指代jQuery对象。某些情况下，其他函数库也会用到美元符号，为了避免冲突，$.noConflict方法允许将美元符号与jQuery脱钩。

```
<script src="other_lib.js"></script>
<script src="jquery.js"></script>
<script>$.noConflict();</script>
```

上面代码就是$.noConflict方法的一般用法。在加载jQuery之后，立即调用该方法，会使得美元符号还给前面一个函数库。这意味着，其后再调用jQuery，只能写成jQuery.methond的形式，而不能用$.method了。为了避免冲突，可以考虑从一开始就只使用jQuery代替美元符号。

jQuery实例对象的方法

除了上一节提到的is、get、eq方法，jQuery实例还有许多其他方法。

结果集的过滤方法

选择器选出一组符合条件的网页元素以后，jQuery提供了许多方法，可以过滤结果集，返回更准确的目标。

（1）first方法，last方法

first方法返回结果集的第一个成员，last方法返回结果集的最后一个成员。

```
$("li").first()


$("li").last()
```

（2）next方法，prev方法

next方法返回紧邻的下一个同级元素，prev方法返回紧邻的上一个同级元素。

```
$("li").first().next()
$("li").last().prev()


$("li").first().next('.item')
$("li").last().prev('.item')
```

如果next方法和prev方法带有参数，表示选择符合该参数的同级元素。

（3）parent方法，parents方法，children方法

parent方法返回当前元素的父元素，parents方法返回当前元素的所有上级元素（直到html元素）。

```
$("p").parent()
$("p").parent(".selected")


$("p").parents()
$("p").parents("div")
```

children方法返回选中元素的所有子元素。

```
$("div").children()
$("div").children(".selected")


// 下面的写法结果相同，但是效率较低


$('div > *')
$('div > .selected')
```

上面这三个方法都接受一个选择器作为参数。

（4）siblings方法，nextAll方法，prevAll方法

siblings方法返回当前元素的所有同级元素。

```
$('li').first().siblings()
$('li').first().siblings('.item')
```

nextAll方法返回当前元素其后的所有同级元素，prevAll方法返回当前元素前面的所有同级元素。

```
$('li').first().nextAll()
```

```
$('li').last().prevAll()
```

（5）closest方法，find方法

closest方法返回当前元素，以及当前元素的所有上级元素之中，第一个符合条件的元素。find方法返回当前元素的所有符合条件的下级元素。

```
$('li').closest('div')
$('div').find('li')
```

上面代码中的find方法，选中所有div元素下面的li元素，等同于$('li', 'div')。由于这样写缩小了搜索范围，所以要优于$('div li')的写法。

（6）find方法，add方法，addBack方法，end方法

add方法用于为结果集添加元素。

```
$('li').add('p')
```

addBack方法将当前元素加回原始的结果集。

```
$('li').parent().addBack()
```

end方法用于返回原始的结果集。

```
$('li').first().end()
```

（7）filter方法，not方法，has方法

filter方法用于过滤结果集，它可以接受多种类型的参数，只返回与参数一致的结果。

```
// 返回符合CSS选择器的结果
$('li').filter('.item')

// 返回函数返回值为true的结果
$("li").filter(function(index) {
    return index % 2 === 1;
})

// 返回符合特定DOM对象的结果
$("li").filter(document.getElementById("unique"))

// 返回符合特定jQuery实例的结果
$("li").filter($("#unique"))
```

not方法的用法与filter方法完全一致，但是返回相反的结果，即过滤掉匹配项。

```
$('li').not('.item')
```

has方法与filter方法作用相同，但是只过滤出子元素符合条件的元素。

```
$("li").has("ul")
```

上面代码返回具有ul子元素的li元素。

DOM相关方法

许多方法可以对DOM元素进行处理。

（1）html方法和text方法

html方法返回该元素包含的HTML代码，text方法返回该元素包含的文本。

假定网页只含有一个p元素。

```
<p><em>Hello World!</em></p>
```

html方法和text方法的返回结果分别如下。

```
$('p').html()
// <em>Hello World!</em>

$('p').text()
```

```
// Hello World!
```

jQuery的许多方法都是取值器（getter）与赋值器（setter）的合一，即取值和赋值都是同一个方法，不使用参数的时候为取值器，使用参数的时候为赋值器。

上面代码的html方法和text方法都没有参数，就会当作取值器使用，取回结果集的第一个元素所包含的内容。如果对这两个方法提供参数，就是当作赋值器使用，修改结果集所有成员的内容，并返回原来的结果集，以便进行链式操作。

```
$('p').html('<strong>你好</strong>')
// 网页代码变为<p><strong>你好</strong></p>

$('p').text('你好')
// 网页代码变为<p>你好</p>
```

下面要讲到的jQuery其他许多方法，都采用这种同一个方法既是取值器又是赋值器的模式。

html方法和text方法还可以接受一个函数作为参数，函数的返回值就是网页元素所要包含的新的代码和文本。这个函数接受两个参数，第一个是网页元素在集合中的位置，第二个参数是网页元素原来的代码或文本。

```
$('li').html(function (i, v){
 return (i + ': ' + v);
})

// <li>Hello</li>
// <li>World</li>
// 变为
// <li>0: Hello</li>
// <li>1: World</li>
```

（2）addClass方法，removeClass方法，toggleClass方法

addClass方法用于添加一个类，removeClass方法用于移除一个类，toggleClass方法用于折叠一个类（如果无就添加，如果有就移除）。

```
$('li').addClass('special')
$('li').removeClass('special')
$('li').toggleClass('special')
```

（3）css方法

css方法用于改变CSS设置。

该方法可以作为取值器使用。

```
$('h1').css('fontSize');
```

css方法的参数是css属性名。这里需要注意，CSS属性名的CSS写法和DOM写法，两者都可以接受，比如font-size和fontSize都行。

css方法也可以作为赋值器使用。

```
$('li').css('padding-left', '20px')
// 或者
$('li').css({
  'padding-left': '20px'
});
```

上面两种形式都可以用于赋值，jQuery赋值器基本上都是如此。

（4）val方法

val方法返回结果集第一个元素的值，或者设置当前结果集所有元素的值。

```
$('input[type="text"]').val()
$('input[type="text"]').val('new value')
```

（5）prop方法，attr方法

首先，这里要区分两种属性。

一种是网页元素的属性，比如a元素的href属性、img元素的src属性。这要使用attr方法读写。

```
// 读取属性值
$('textarea').attr(name)

//写入属性值
$('textarea').attr(name, val)
```

下面是通过设置a元素的target属性，使得网页上的外部链接在新窗口打开的例子。

```
$('a[href^="http"]').attr('target', '_blank');
$('a[href^="//"]').attr('target', '_blank');
$('a[href^="' + window.location.origin + '"]').attr('target', '_self');
```

另一种是DOM元素的属性，比如tagName、nodeName、nodeType等等。这要使用prop方法读写。

```
// 读取属性值
$('textarea').prop(name)

// 写入属性值
$('textarea').prop(name, val)
```

所以，attr方法和prop方法针对的是不同的属性。在英语中，attr是attribute的缩写，prop是property的缩写，中文很难表达出这种差异。有时，attr方法和prop方法对同一个属性会读到不一样的值。比如，网页上有一个单选框。

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

对于checked属性，attr方法读到的是checked，prop方法读到的是true。

```
$(input[type=checkbox]).attr("checked") // "checked"

$(input[type=checkbox]).prop("checked") // true
```

可以看到，attr方法读取的是网页上该属性的值，而prop方法读取的是DOM元素的该属性的值，根据规范，element.checked应该返回一个布尔值。所以，判断单选框是否选中，要使用prop方法。事实上，不管这个单选框是否选中，attr(“checked”)的返回值都是checked。

```
if ($(elem).prop("checked")) { /*... */ };

// 下面两种方法亦可

if ( elem.checked ) { /*...*/ };
if ( $(elem).is(":checked") ) { /*...*/ };
```

（6）removeProp方法，removeAttr方法

removeProp方法移除某个DOM属性，removeAttr方法移除某个HTML属性。

```
$("a").prop("oldValue",1234).removeProp('oldValue')
$('a').removeAttr("title")
```

（7）data方法

data方法用于在一个DOM对象上储存数据。

```
// 储存数据
$("body").data("foo", 52);

// 读取数据
$("body").data("foo");
```

该方法可以在DOM节点上储存各种类型的数据。

添加、复制和移动网页元素的方法

jQuery方法提供一系列方法，可以改变元素在文档中的位置。

（1）append方法，appendTo方法

append方法将参数中的元素插入当前元素的尾部。

```
$("div").append("<p>World</p>")


// <div>Hello </div>
// 变为
// <div>Hello <p>World</p></div>
```

appendTo方法将当前元素插入参数中的元素尾部。

```
$("<p>World</p>").appendTo("div")
```

上面代码返回与前一个例子一样的结果。

（2）prepend方法，prependTo方法

prepend方法将参数中的元素，变为当前元素的第一个子元素。

```
$("p").prepend("Hello ")


// <p>World</p>
// 变为
// <p>Hello World</p>
```

如果prepend方法的参数不是新生成的元素，而是当前页面已存在的元素，则会产生移动元素的效果。

```
$("p").prepend("strong")


// <strong>Hello </strong><p>World</p>
// 变为
// <p><strong>Hello </strong>World</p>
```

上面代码运行后，strong元素的位置将发生移动，而不是克隆一个新的strong元素。不过，如果当前结果集包含多个元素，则除了第一个以后，后面的p元素都将插入一个克隆的strong子元素。

prependTo方法将当前元素变为参数中的元素的第一个子元素。

```
$("<p></p>").prependTo("div")


// <div></div>
// 变为
// <div><p></p></div>
```

（3）after方法，insertAfter方法

after方法将参数中的元素插在当前元素后面。

```
$("div").after("<p></p>")


// <div></div>
// 变为
// <div></div><p></p>
```

insertAfter方法将当前元素插在参数中的元素后面。

```
$("<p></p>").insertAfter("div")
```

上面代码返回与前一个例子一样的结果。

（4）before方法，insertBefore方法

before方法将参数中的元素插在当前元素的前面。

```
$("div").before("<p></p>")
```

```
// <div></div>
// 变为
// <p></p><div></div>
```

insertBefore方法将当前元素插在参数中的元素的前面。

```
$("<p></p>").insertBefore("div")
```

上面代码返回与前一个例子一样的结果。

（5）wrap方法，wrapAll方法，unwrap方法，wrapInner方法

wrap方法将参数中的元素变成当前元素的父元素。

```
$("p").wrap("<div></div>")

// <p></p>
// 变为
// <div><p></p></div>
```

wrap方法的参数还可以是一个函数。

```
$("p").wrap(function() {
  return "<div></div>";
})
```

上面代码返回与前一个例子一样的结果。

wrapAll方法为结果集的所有元素，添加一个共同的父元素。

```
$("p").wrapAll("<div></div>")

// <p></p><p></p>
// 变为
// <div><p></p><p></p></div>
```

unwrap方法移除当前元素的父元素。

```
$("p").unwrap()

// <div><p></p></div>
// 变为
// <p></p>
```

wrapInner方法为当前元素的所有子元素，添加一个父元素。

```
$("p").wrapInner('<strong></strong>')

// <p>Hello</p>
// 变为
// <p><strong>Hello</strong></p>
```

（6）clone方法

clone方法克隆当前元素。

```
var clones = $('li').clone();
```

对于那些有id属性的节点，clone方法会连id属性一起克隆。所以，要把克隆的节点插入文档的时候，务必要修改或移除id属性。

（7）remove方法，detach方法，replaceWith方法

remove方法移除并返回一个元素，取消该元素上所有事件的绑定。detach方法也是移除并返回一个元素，但是不取消该元素上所有事件的绑定。

```
$('p').remove()
$('p').detach()
```

replaceWith方法用参数中的元素，替换并返回当前元素，取消当前元素的所有事件的绑定。

```
$('p').replaceWith('<div></div>')
```

动画效果方法

jQuery提供一些方法，可以很容易地显示网页动画效果。但是，总体上来说，它们不如CSS动画强大和节省资源，所以应该优先考虑使用CSS动画。

如果将jQuery.fx.off设为true，就可以将所有动画效果关闭，使得网页元素的各种变化一步到位，没有中间过渡的动画效果。

（1）动画效果的简便方法

jQuery提供以下一些动画效果方法。

- show：显示当前元素。
- hide：隐藏当前元素。
- toggle：显示或隐藏当前元素。
- fadeIn：将当前元素的不透明度（opacity）逐步提升到100%。
- fadeOut：将当前元素的不透明度逐步降为0%。
- fadeToggle：以逐渐透明或逐渐不透明的方式，折叠显示当前元素。
- slideDown：以从上向下滑入的方式显示当前元素。
- slideUp：以从下向上滑出的方式隐藏当前元素。
- slideToggle：以垂直滑入或滑出的方式，折叠显示当前元素。

上面这些方法可以不带参数调用，也可以接受毫秒或预定义的关键字作为参数。

```
$('.hidden').show();
$('.hidden').show(300);
$('.hidden').show('slow');
```

上面三行代码分别表示，以默认速度、300毫秒、较慢的速度隐藏一个元素。

jQuery预定义的关键字是在jQuery.fx.speeds对象上面，可以自行改动这些值，或者创造新的值。

```
jQuery.fx.speeds.fast = 50;
jQuery.fx.speeds.slow = 3000;
jQuery.fx.speeds.normal = 1000;
```

上面三行代码重新定义fast、normal、slow关键字对应的毫秒数。

你还可以定义自己的关键字。

```
jQuery.fx.speeds.blazing = 30;

// 调用
$('.hidden').show('blazing');
```

这些方法还可以接受一个函数，作为第二个参数，表示动画结束后的回调函数。

```
$('p').fadeOut(300, function() {
  $(this).remove();
});
```

上面代码表示，p元素以300毫秒的速度淡出，然后调用回调函数，将其从DOM中移除。

使用按钮控制某个元素折叠显示的代码如下。

```
// Fade
$('.btn').click(function () {
  $('.element').fadeToggle('slow');
});

// Toggle
$('.btn').click(function () {
  $('.element').slideToggle('slow');
```

```
});
```

（2）animate方法

上面这些动画效果方法，实际上都是animate方法的简便写法。在幕后，jQuery都是统一使用animate方法生成各种动画效果。

```
$('a.top').click(function (e) {
  e.preventDefault();
  $('html, body').animate({scrollTop: 0}, 800);
});
```

上面代码是点击链接，回到页面头部的写法。其中，animate方法接受两个参数，第一个参数是一个对象，表示动画结束时相关CSS属性的值，第二个参数是动画持续的毫秒数。需要注意的是，第一个参数对象的成员名称，必须与CSS属性名称一致，如果CSS属性名称带有连字号，则需要用“骆驼拼写法”改写。

animate方法还可以接受第三个参数，表示动画结束时的回调函数。

```
$('div').animate({
    left: '+=50', // 增加50
    opacity: 0.25,
    fontSize: '12px'
  },
  300, // 持续时间
  function() { // 回调函数
     console.log('done!');
  }
);
```

上面代码表示，动画结束时，在控制台输出“done！”。

（3）stop方法，delay方法

stop方法表示立即停止执行当前的动画。

```
$("#stop").click(function() {
  $(".block").stop();
});
```

上面代码表示，点击按钮后，block元素的动画效果停止。

delay方法接受一个时间参数，表示暂停多少毫秒后继续执行。

```
$("#foo").slideUp(300).delay(800).fadeIn(400)
```

上面代码表示，slideUp动画之后，暂停800毫秒，然后继续执行fadeIn动画。

其他方法

jQuery还提供一些供特定元素使用的方法。

serialize方法用于将表单元素的值，转为url使用的查询字符串。

```
$( "form" ).on( "submit", function( event ) {
  event.preventDefault();
  console.log( $( this ).serialize() );
});

// single=Single&multiple=Multiple&check=check2&radio=radio1
```

serializeArray方法用于将表单元素的值转为数组。

```
$("form").submit(function (event){
  console.log($(this).serializeArray());
  event.preventDefault();
});
```

```
// [
//  {name : 'field1', value : 123},
//  {name : 'field2', value : 'hello world'}
// ]
```

事件处理

事件绑定的简便方法

jQuery提供一系列方法，允许直接为常见事件绑定回调函数。比如，click方法可以为一个元素绑定click事件的回调函数。

```
$('li').click(function (e){
  console.log($(this).text());
});
```

上面代码为li元素绑定click事件的回调函数，点击后在控制台显示li元素包含的文本。
这样绑定事件的简便方法有如下一些：

- click
- keydown
- keypress
- keyup
- mouseover
- mouseout
- mouseenter
- mouseleave
- scroll
- focus
- blur
- resize
- hover

如果不带参数调用这些方法，就是触发相应的事件，从而引发回调函数的运行。

```
$('li').click()
```

上面代码将触发click事件的回调函数。
需要注意的是，通过这种方法触发回调函数，将不会引发浏览器对该事件的默认行为。比如，对a元素调用click方法，将只触发事先绑定的回调函数，而不会导致浏览器将页面导向href属性指定的网址。
下面是一个捕捉用户按下escape键的函数。

```
$(document).keyup(function(e) {
  if (e.keyCode == 27) {
    $('body').toggleClass('show-nav');
    // $('body').removeClass('show-nav');
  }
});
```

上面代码中，用户按下escape键，jQuery就会为body元素添加/去除名为show-nav的class。
hover方法需要特别说明。它接受两个回调函数作为参数，分别代表mouseenter和mouseleave事件的回调函数。

```
$(selector).hover(handlerIn, handlerOut)
// 等同于
$(selector).mouseenter(handlerIn).mouseleave(handlerOut)
```

下面是一个例子，当按钮发生hover事件，添加一个class样式，当hover事件结束时，再取消这个class。

```
$('.btn').hover(function () {
  $(this).addClass('hover');
}, function () {
  $(this).removeClass('hover');
});
```

使用toggleClass可以简化上面的代码。

```
$('.btn').hover(function () {
  $(this).toggleClass('hover');
});
```

on方法，trigger方法，off方法

除了简便方法，jQuery还提供事件处理的通用方法。

（1）on方法

on方法是jQuery事件绑定的统一接口。事件绑定的那些简便方法，其实都是on方法的简写形式。

on方法接受两个参数，第一个是事件名称，第二个是回调函数。

```
$('li').on('click', function (e){
  console.log($(this).text());
});
```

上面代码为li元素绑定click事件的回调函数。

> 注意，在回调函数内部，this关键字指的是发生该事件的DOM对象。为了使用jQuery提供的方法，必须将DOM对象转为jQuery对象，因此写成$(this)。

on方法允许一次为多个事件指定同样的回调函数。

```
$('input[type="text"]').on('focus blur', function (){
  console.log('focus or blur');
});
```

上面代码为文本框的focus和blur事件绑定同一个回调函数。

下面是一个例子，当图片加载失败，使用error事件，替换另一张图片。

```
$('img').on('error', function () {
  if(!$(this).hasClass('broken-image')) {
    $(this).prop('src', 'img/broken.png').addClass('broken-image');
  }
});
```

下面是检查用户是否切换浏览器tab的例子。

```
$(document).on('visibilitychange', function (e) {
  if (e.target.visibilityState === "visible") {
    console.log('Tab is now in view!');
  } else if (e.target.visibilityState === "hidden") {
    console.log('Tab is now hidden!');
  }
});
```

on方法还可以为当前元素的某一个子元素，添加回调函数。

```
$('ul').on('click', 'li', function (e){
  console.log(this);
```

```
});
```

上面代码为ul的子元素li绑定click事件的回调函数。采用这种写法时，on方法接受三个参数，子元素选择器作为第二个参数，夹在事件名称和回调函数之间。

这种写法有两个好处。首先，click事件还是在ul元素上触发回调函数，但是会检查event对象的target属性是否为li子元素，如果为true，再调用回调函数。这样就比为li元素一一绑定回调函数，节省了内存空间。其次，这种绑定的回调函数，对于在绑定后生成的li元素依然有效。

on方法还允许向回调函数传入数据。

```
$("ul").on("click", {name: "张三"}, function (event){
 console.log(event.data.name);
});
```

上面代码在发生click事件之后，会通过event对象的data属性，在控制台打印出所传入的数据（即“张三”）。

（2）trigger方法

trigger方法用于触发回调函数，它的参数就是事件的名称。

```
$('li').trigger('click')
```

上面代码触发li元素的click事件回调函数。与那些简便方法一样，trigger方法只触发回调函数，而不会引发浏览器的默认行为。

（3）off方法

off方法用于移除事件的回调函数。

```
$('li').off('click')
```

上面代码移除li元素所有的click事件回调函数。

（4）事件的名称空间

同一个事件有时绑定了多个回调函数，这时如果想移除其中的一个回调函数，可以采用“名称空间”的方式，即为每一个回调函数指定一个二级事件名，然后再用off方法移除这个二级事件的回调函数。

```
$('li').on('click.logging', function (){
  console.log('click.logging callback removed');
});


$('li').off('click.logging');
```

上面代码为li元素定义了二级事件click.logging的回调函数，click.logging属于click名称空间，当发生click事件时会触发该回调函数。将click.logging作为off方法的参数，就会移除这个回调函数，但是对其他click事件的回调函数没有影响。

trigger方法也适用带名称空间的事件。

```
$('li').trigger('click.logging')
```

event对象

当回调函数被触发后，它们的参数通常是一个事件对象event。

```
$(document).on('click', function (e){
 // ...
});
```

上面代码的回调函数的参数e，就代表事件对象event。

event对象有以下属性：

- type：事件类型，比如click。
- which：触发该事件的鼠标按钮或键盘的键。
- target：事件发生的初始对象。
- data：传入事件对象的数据。
- pageX：事件发生时，鼠标位置的水平坐标（相对于页面左上角）。
- pageY：事件发生时，鼠标位置的垂直坐标（相对于页面左上角）。

event对象有以下方法：

- preventDefault：取消浏览器默认行为。

- stopPropagation：阻止事件向上层元素传播。

一次性事件

one方法指定一次性的回调函数，即这个函数只能运行一次。这对提交表单很有用。

```
$("#button").one( "click", function() { return false; } );
```

one方法本质上是回调函数运行一次，即解除对事件的监听。

```
document.getElementById("#button").addEventListener("click", handler);

function handler(e) {
    e.target.removeEventListener(e.type, arguments.callee);
 return false;
}
```

上面的代码在点击一次以后，取消了对click事件的监听。如果有特殊需要，可以设定点击2次或3次之后取消监听，这都是可以的。