06 jQuery

你可能听说过jQuery，它名字起得很土，但却是JavaScript世界中使用最广泛的一个库。

江湖传言，全世界大约有80~90%的网站直接或间接地使用了jQuery。鉴于它如此流行，又如此好用，所以每一个入门JavaScript的前端工程师都应该了解和学习它。

jQuery这么流行，肯定是因为它解决了一些很重要的问题。实际上，jQuery能帮我们干这些事情：

- 消除浏览器差异：你不需要自己写冗长的代码来针对不同的浏览器来绑定事件，编写AJAX等代码；
- 简洁的操作DOM的方法：写`$('#test')`肯定比`document.getElementById('test')`来得简洁；
- 轻松实现动画、修改CSS等各种操作。

jQuery的理念“Write Less, Do More“，让你写更少的代码，完成更多的工作！

jQuery版本

目前jQuery有1.x和2.x两个主要版本，区别在于2.x移除了对古老的IE 6、7、8的支持，因此2.x的代码更精简。选择哪个版本主要取决于你是否想支持IE 6~8。

从jQuery官网可以下载最新版本。jQuery只是一个`jquery-xxx.js`文件，但你会看到有compressed（已压缩）和uncompressed（未压缩）两种版本，使用时完全一样，但如果你想深入研究jQuery源码，那就用uncompressed版本。

使用jQuery

使用jQuery只需要在页面的``引入jQuery文件即可：

```
<html>
<head>
    <script src="//code.jquery.com/jquery-1.11.3.min.js">
</script>
    ...
</head>
<body>
    ...
</body>
</html>
```

好消息是，当你在学习这个教程时，由于网站本身已经引用了jQuery，所以你可以直接使用：

```
'use strict';
console.log('jQuery版本：' + $.fn.jquery);
```

$符号

`$`是著名的jQuery符号。实际上，jQuery把所有功能全部封装在一个全局变量`jQuery`中，而`$`也是一个合法的变量名，它是变量`jQuery`的别名：

```
window.jQuery; // jQuery(selector, context)
window.$; // jQuery(selector, context)
$ === jQuery; // true
typeof($); // 'function'
```

`$`本质上就是一个函数，但是函数也是对象，于是`$`除了可以直接调用外，也可以有很多其他属性。

注意，你看到的`$`函数名可能不是`jQuery(selector, context)`，因为很多JavaScript压缩工具可以对函数名和参数改名，所以压缩过的jQuery源码`$`函数可能变成`a(b, c)`。

绝大多数时候，我们都直接用`$`（因为写起来更简单嘛）。但是，如果`$`这个变量不幸地被占用了，而且还不能改，那我们就只能让`jQuery`把`$`变量交出来，然后就只能使用`jQuery`这个变量：

```
$; // jQuery(selector, context)
jQuery.noConflict();
$; // undefined
jQuery; // jQuery(selector, context)
```

这种黑魔法的原理是jQuery在占用`$`之前，先在内部保存了原来的`$`,调用`jQuery.noConflict()`时会把原来保存的变量还原。

选择器

选择器是jQuery的核心。一个选择器写出来类似`$('#dom-id')`。

为什么jQuery要发明选择器？回顾一下DOM操作中我们经常使用的代码：

```
// 按ID查找：
var a = document.getElementById('dom-id');

// 按tag查找：
var divs = document.getElementsByTagName('div');

// 查找<p class="red">：
var ps = document.getElementsByTagName('p');
// 过滤出class="red":
// TODO:
```

```
// 查找<table class="green">里面的所有<tr>:
var table = ...
for (var i=0; i<table.children; i++) {
    // TODO: 过滤出<tr>
}
```

这些代码实在太繁琐了，并且，在层级关系中，例如，查找`里面的所有，一层循环实际上是错的，因为`的标准写法是：

```
<table>
    <tbody>
        <tr>...</tr>
        <tr>...</tr>
    </tbody>
</table>
```

很多时候，需要递归查找所有子节点。

jQuery的选择器就是帮助我们快速定位到一个或多个DOM节点。

按ID查找

如果某个DOM节点有`id`属性，利用jQuery查找如下：

```
// 查找<div id="abc">:
var div = $('#abc');
```

注意，`#abc`以`#`开头。返回的对象是jQuery对象。

什么是jQuery对象？jQuery对象类似数组，它的每个元素都是一个引用了DOM节点的对象。

以上面的查找为例，如果`id`为`abc`的``存在，返回的jQuery对象如下：

```
[<div id="abc">...</div>]
```

如果`id`为`abc`的``不存在，返回的jQuery对象如下：

```
[]
```

总之jQuery的选择器不会返回`undefined`或者`null`，这样的好处是你不必在下一行判断`if (div === undefined)`。

jQuery对象和DOM对象之间可以互相转化：

```
var div = $('#abc'); // jQuery对象
var divDom = div.get(0); // 假设存在div，获取第1个DOM元素
var another = $(divDom); // 重新把DOM包装为jQuery对象
```

通常情况下你不需要获取DOM对象，直接使用jQuery对象更加方便。如果你拿到了一个DOM对象，那可以简单地调用`$(aDomObject)`把它变成jQuery对象，这样就可以方便地使用jQuery的API了。

按tag查找

按tag查找只需要写上tag名称就可以了：

```
var ps = $('p'); // 返回所有<p>节点
ps.length; // 数一数页面有多少个<p>节点
```

按class查找

按class查找注意在class名称前加一个`.`：

```
var a = $('.red'); // 所有节点包含`class="red"`都将返回
// 例如:
// <div class="red">...</div>
// <p class="green red">...</p>
```

通常很多节点有多个class，我们可以查找同时包含`red`和`green`的节点：

```
var a = $('.red.green'); // 注意没有空格!
// 符合条件的节点:
// <div class="red green">...</div>
// <div class="blue green red">...</div>
```

按属性查找

一个DOM节点除了`id`和`class`外还可以有很多属性，很多时候按属性查找会非常方便，比如在一个表单中按属性来查找：

```
var email = $('[name=email]'); // 找出<??? name="email">
var passwordInput = $('[type=password]'); // 找出<???
type="password">
var a = $('[items="A B"]'); // 找出<??? items="A B">
```

当属性的值包含空格等特殊字符时，需要用双引号括起来。

按属性查找还可以使用前缀查找或者后缀查找：

```
var icons = $('[name^=icon]'); // 找出所有name属性值以icon开
头的DOM
// 例如: name="icon-1", name="icon-2"
var names = $('[name$=with]'); // 找出所有name属性值以with结
尾的DOM
// 例如: name="startswith", name="endswith"
```

这个方法尤其适合通过class属性查找，且不受class包含多个名称的影响：

```
var icons = $('[class^="icon-"]'); // 找出所有class包含至少一个以`icon-`开头的DOM
// 例如: class="icon-clock", class="abc icon-home"
```

组合查找

组合查找就是把上述简单选择器组合起来使用。如果我们查找`$('[name=email]')`，很可能把表单外的`也找出来，但我们只希望查找`，就可以这么写：

```
var emailInput = $('input[name=email]'); // 不会找出<div name="email">
```

同样的，根据tag和class来组合查找也很常见：

```
var tr = $('tr.red'); // 找出<tr class="red ...">...</tr>
```

多项选择器

多项选择器就是把多个选择器用`,`组合起来一块选：

```
$('p,div'); // 把<p>和<div>都选出来
$('p.red,p.green'); // 把<p class="red">和<p class="green">都选出来
```

要注意的是，选出来的元素是按照它们在HTML中出现的顺序排列的，而且不会有重复元素。例如，`` `不会被上面的$('p.red,p.green')`选择两次。

练习

使用jQuery选择器分别选出指定元素：

- 仅选择JavaScript
- 仅选择Erlang
- 选择JavaScript和Erlang
- 选择所有编程语言
- 选择名字input
- 选择邮件和名字input

```
<!-- HTML结构 -->
<div id="test-jquery">
    <p id="para-1" class="color-red">JavaScript</p>
    <p id="para-2" class="color-green">Haskell</p>
    <p class="color-red color-green">Erlang</p>
    <p name="name" class="color-black">Python</p>
    <form class="test-form" target="_blank" action="#0" onsubmit="return false;">
```

```
        <legend>注册新用户</legend>
        <fieldset>
            <p><label>名字: <input name="name"></label>
</p>
            <p><label>邮件: <input name="email"></label>
</p>
            <p><label>口令: <input name="password"
type="password"></label></p>
            <p><button type="submit">注册</button></p>
        </fieldset>
    </form>
</div>
```

运行查看结果：

```
'use strict';

var selected = null;
selected = ???;
// 高亮结果:
if (!(selected instanceof jQuery)) {
    return console.log('不是有效的jQuery对象!');
}
$('#test-jquery').find('*').css('background-color', '');
selected.css('background-color', '#ffd351');
```

层级选择器

除了基本的选择器外，jQuery的层级选择器更加灵活，也更强大。

因为DOM的结构就是层级结构，所以我们经常要根据层级关系进行选择。

层级选择器（Descendant Selector）

如果两个DOM元素具有层级关系，就可以用`$('ancestor descendant')`来选择，层级之间用空格隔开。例如：

```
<!-- HTML结构 -->
<div class="testing">
    <ul class="lang">
        <li class="lang-javascript">JavaScript</li>
        <li class="lang-python">Python</li>
        <li class="lang-lua">Lua</li>
    </ul>
</div>
```

要选出JavaScript，可以用层级选择器：

```
$('ul.lang li.lang-javascript'); // [<li class="lang-javascript">JavaScript</li>]
$('div.testing li.lang-javascript'); // [<li class="lang-javascript">JavaScript</li>]
```

因为和都是的祖先节点，所以上面两种方式都可以选出相应的节点。

要选择所有的``节点，用：

```
$('ul.lang li');
```

这种层级选择器相比单个的选择器好处在于，它缩小了选择范围，因为首先要定位父节点，才能选择相应的子节点，这样避免了页面其他不相关的元素。

例如：

```
$('form[name=upload] input');
```

就把选择范围限定在name属性为upload的表单里。如果页面有很多表单，其他表单的``不会被选择。

多层选择也是允许的：

```
$('form.test p input'); // 在form表单选择被<p>包含的<input>
```

子选择器（Child Selector）

子选择器$('parent>child')类似层级选择器，但是限定了层级关系必须是父子关系，就是节点必须是节点的直属子节点。还是以上面的例子：

```
$('ul.lang>li.lang-javascript'); // 可以选出[<li class="lang-javascript">JavaScript</li>]
$('div.testing>li.lang-javascript'); // [], 无法选出，因为<div>和<li>不构成父子关系
```

过滤器（Filter）

过滤器一般不单独使用，它通常附加在选择器上，帮助我们更精确地定位元素。观察过滤器的效果：

```
$('ul.lang li'); // 选出JavaScript、Python和Lua 3个节点

$('ul.lang li:first-child'); // 仅选出JavaScript
$('ul.lang li:last-child'); // 仅选出Lua
$('ul.lang li:nth-child(2)'); // 选出第N个元素，N从1开始
$('ul.lang li:nth-child(even)'); // 选出序号为偶数的元素
$('ul.lang li:nth-child(odd)'); // 选出序号为奇数的元素
```

表单相关

针对表单元素，jQuery还有一组特殊的选择器：

- `:input`：可以选择`<input>`，`<select>`，`<button>`和`<textarea>`；
- `:file`：可以选择`<input type="file">`，和input[type=file]`一样；
- `:checkbox`：可以选择复选框，和`input[type=checkbox]`一样；
- `:radio`：可以选择单选框，和`input[type=radio]`一样；
- `:focus`：可以选择当前输入焦点的元素，例如把光标放到一个`<input>`上，用`$('input:focus')`就可以选出；
- `:checked`：选择当前勾上的单选框和复选框，用这个选择器可以立刻获得用户选择的项目，如`$('input[type=radio]:checked')`；
- `:enabled`：可以选择可以正常输入的`<input>`、`<select>` 等，也就是没有灰掉的输入；
- `:disabled`：和`:enabled`正好相反，选择那些不能输入的。

此外，jQuery还有很多有用的选择器，例如，选出可见的或隐藏的元素：

```
$('div:visible'); // 所有可见的div
$('div:hidden'); // 所有隐藏的div
```

练习

针对如下HTML结构：

```
<!-- HTML结构 -->

<div class="test-selector">
    <ul class="test-lang">
        <li class="lang-javascript">JavaScript</li>
        <li class="lang-python">Python</li>
        <li class="lang-lua">Lua</li>
    </ul>
    <ol class="test-lang">
        <li class="lang-swift">Swift</li>
        <li class="lang-java">Java</li>
        <li class="lang-c">C</li>
    </ol>
</div>
```

选出相应的内容并观察效果：

```
'use strict';
var selected = null;
// 分别选择所有语言，所有动态语言，所有静态语言，JavaScript，Lua，
C等：
selected = ???
// 高亮结果：
if (!(selected instanceof jQuery)) {
    return console.log('不是有效的jQuery对象!');
}
$('#test-jquery').find('*').css('background-color', '');
selected.css('background-color', '#ffd351');
```

- JavaScript
- Python
- Lua

1. Swift
2. Java
3. C

查找和过滤

通常情况下选择器可以直接定位到我们想要的元素，但是，当我们拿到一个jQuery对象后，还可以以这个对象为基准，进行查找和过滤。

最常见的查找是在某个节点的所有子节点中查找，使用`find()`方法，它本身又接收一个任意的选择器。例如如下的HTML结构：

- JavaScript
- Python
- Swift
- Scheme
- Haskell

```
<!-- HTML结构 -->
<ul class="lang">
    <li class="js dy">JavaScript</li>
    <li class="dy">Python</li>
    <li id="swift">Swift</li>
    <li class="dy">Scheme</li>
    <li name="haskell">Haskell</li>
</ul>
```

用`find()`查找：

```
var ul = $('ul.lang'); // 获得<ul>
var dy = ul.find('.dy'); // 获得JavaScript, Python, Scheme
var swf = ul.find('#swift'); // 获得Swift
var hsk = ul.find('[name=haskell]'); // 获得Haskell
```

如果要从当前节点开始向上查找，使用`parent()`方法：

```
var swf = $('#swift'); // 获得Swift
var parent = swf.parent(); // 获得Swift的上层节点<ul>
var a = swf.parent('.red'); // 获得Swift的上层节点<ul>，同时传入过滤条件。如果ul不符合条件，返回空jQuery对象
```

对于位于同一层级的节点，可以通过`next()`和`prev()`方法，例如：

当我们已经拿到`Swift`节点后：

```
var swift = $('#swift');

swift.next(); // Scheme
swift.next('[name=haskell]'); // 空的jQuery对象，因为Swift的下一个元素Scheme不符合条件[name=haskell]

swift.prev(); // Python
swift.prev('.dy'); // Python，因为Python同时符合过滤器条件.dy
```

过滤

和函数式编程的map、filter类似，jQuery对象也有类似的方法。

`filter()`方法可以过滤掉不符合选择器条件的节点：

```
var langs = $('ul.lang li'); // 拿到JavaScript, Python, Swift, Scheme和Haskell
var a = langs.filter('.dy'); // 拿到JavaScript, Python, Scheme
```

或者传入一个函数，要特别注意函数内部的`this`被绑定为DOM对象，不是jQuery对象：

```
var langs = $('ul.lang li'); // 拿到JavaScript, Python, Swift, Scheme和Haskell
langs.filter(function () {
    return this.innerHTML.indexOf('S') === 0; // 返回S开头的节点
}); // 拿到Swift, Scheme
```

`map()`方法把一个jQuery对象包含的若干DOM节点转化为其他对象：

```
var langs = $('ul.lang li'); // 拿到JavaScript, Python, Swift, Scheme和Haskell
var arr = langs.map(function () {
    return this.innerHTML;
}).get(); // 用get()拿到包含string的Array：['JavaScript', 'Python', 'Swift', 'Scheme', 'Haskell']
```

此外，一个jQuery对象如果包含了不止一个DOM节点，`first()`、`last()`和`slice()`方法可以返回一个新的jQuery对象，把不需要的DOM节点去掉：

```
var langs = $('ul.lang li'); // 拿到JavaScript, Python, Swift, Scheme和Haskell
var js = langs.first(); // JavaScript，相当于$('ul.lang li:first-child')
var haskell = langs.last(); // Haskell, 相当于$('ul.lang li:last-child')
var sub = langs.slice(2, 4); // Swift, Scheme, 参数和数组的slice()方法一致
```

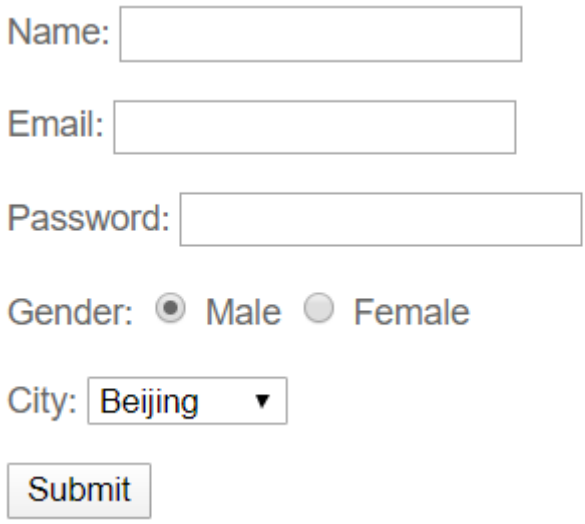
练习

对于下面的表单：

```
<form id="test-form" action="#0" onsubmit="return false;">
    <p><label>Name: <input name="name"></label></p>
    <p><label>Email: <input name="email"></label></p>
    <p><label>Password: <input name="password" type="password"></label></p>
    <p>Gender: <label><input name="gender" type="radio" value="m" checked> Male</label> <label><input name="gender" type="radio" value="f"> Female</label></p>
    <p><label>City: <select name="city">
        <option value="BJ" selected>Beijing</option>
        <option value="SH">Shanghai</option>
        <option value="CD">Chengdu</option>
        <option value="XM">Xiamen</option>
    </select></label></p>
    <p><button type="submit">Submit</button></p>
</form>
```

输入值后，用jQuery获取表单的JSON字符串，key和value分别对应每个输入的name和相应的value，例如：`{"name":"Michael","email":...}`

```
'use strict';
var json = null;
json = ???;
// 显示结果:
if (typeof(json) === 'string') {
    console.log(json);
}
else {
    console.log('json变量不是string!');
}
```



操作DOM

jQuery的选择器很强大，用起来又简单又灵活，但是搞了这么久，我拿到了jQuery对象，到底要干什么？

答案当然是操作对应的DOM节点啦！

回顾一下修改DOM的CSS、文本、设置HTML有多么麻烦，而且有的浏览器只有innerHTML，有的浏览器支持innerText，有了jQuery对象，不需要考虑浏览器差异了，全部统一操作！

修改Text和HTML

jQuery对象的`text()`和`html()`方法分别获取节点的文本和原始HTML文本，例如，如下的HTML结构：

```
<!-- HTML结构 -->
<ul id="test-ul">
    <li class="js">JavaScript</li>
    <li name="book">Java &amp;amp; JavaScript</li>
</ul>
```

分别获取文本和HTML：

```
$('#test-ul li[name=book]').text(); // 'Java & JavaScript'
$('#test-ul li[name=book]').html(); // 'Java &amp;
JavaScript'
```

如何设置文本或HTML？jQuery的API设计非常巧妙：无参数调用`text()`是获取文本，传入参数就变成设置文本，HTML也是类似操作，自己动手试试：

```
'use strict';
var j1 = $('#test-ul li.js');
var j2 = $('#test-ul li[name=book]');
j1.html('<span style="color: red">JavaScript</span>');
j2.text('JavaScript & ECMAScript');
```

- JavaScript
- Java & JavaScript

一个jQuery对象可以包含0个或任意个DOM对象，它的方法实际上会作用在对应的每个DOM节点上。在上面的例子中试试：

```
$('#test-ul li').text('JS'); // 是不是两个节点都变成了JS?
```

所以jQuery对象的另一个好处是我们可以执行一个操作，作用在对应的一组DOM节点上。即使选择器没有返回任何DOM节点，调用jQuery对象的方法仍然不会报错：

```
// 如果不存在id为not-exist的节点:
$('#not-exist').text('Hello'); // 代码不报错，没有节点被设置为'Hello'
```

这意味着jQuery帮你免去了许多`if`语句。

修改CSS

jQuery对象有“批量操作”的特点，这用于修改CSS实在是太方便了。考虑下面的HTML结构：

```
<!-- HTML结构 -->
<ul id="test-css">
    <li class="lang dy"><span>JavaScript</span></li>
    <li class="lang"><span>Java</span></li>
    <li class="lang dy"><span>Python</span></li>
    <li class="lang"><span>Swift</span></li>
    <li class="lang dy"><span>Scheme</span></li>
</ul>
```

要高亮显示动态语言，调用jQuery对象的`css('name', 'value')`方法，我们用一行语句实现：

```
'use strict';
$('#test-css li.dy>span').css('background-color', '#ffd351').css('color', 'red');
```

- JavaScript
- Java
- Python
- Swift
- Scheme

注意，jQuery对象的所有方法都返回一个jQuery对象（可能是新的也可能是自身），这样我们可以进行链式调用，非常方便。

jQuery对象的`css()`方法可以这么用：

```
var div = $('#test-div');
div.css('color'); // '#000033', 获取CSS属性
div.css('color', '#336699'); // 设置CSS属性
div.css('color', ''); // 清除CSS属性
```

为了和JavaScript保持一致，CSS属性可以用`'background-color'`和`'backgroundColor'`两种格式。

`css()`方法将作用于DOM节点的`style`属性，具有最高优先级。如果要修改`class`属性，可以用jQuery提供的下列方法：

```
var div = $('#test-div');
div.hasClass('highlight'); // false, class是否包含highlight
div.addClass('highlight'); // 添加highlight这个class
div.removeClass('highlight'); // 删除highlight这个class
```

练习：分别用`css()`方法和`addClass()`方法高亮显示JavaScript：

```
<!-- HTML结构 -->
<style>
.highlight {
    color: #dd1144;
    background-color: #ffd351;
}
</style>

<div id="test-highlight-css">
    <ul>
        <li class="py"><span>Python</span></li>
        <li class="js"><span>JavaScript</span></li>
        <li class="sw"><span>Swift</span></li>
```

```
        <li class="hk"><span>Haskell</span></li>
    </ul>
</div>
```

```
'use strict';
var div = $('#test-highlight-css');
// TODO:
```

- Python
- JavaScript
- Swift
- Haskell

显示和隐藏DOM

要隐藏一个DOM，我们可以设置CSS的`display`属性为`none`，利用`css()`方法就可以实现。不过，要显示这个DOM就需要恢复原有的`display`属性，这就得先记下来原有的`display`属性到底是`block`还是`inline`还是别的值。

考虑到显示和隐藏DOM元素使用非常普遍，jQuery直接提供`show()`和`hide()`方法，我们不用关心它是如何修改`display`属性的，总之它能正常工作：

```
var a = $('a[target=_blank]');
a.hide(); // 隐藏
a.show(); // 显示
```

注意，隐藏DOM节点并未改变DOM树的结构，它只影响DOM节点的显示。这和删除DOM节点是不同的。

获取DOM信息

利用jQuery对象的若干方法，我们直接可以获取DOM的高宽等信息，而无需针对不同浏览器编写特定代码：

```
// 浏览器可视窗口大小:
$(window).width(); // 800
$(window).height(); // 600

// HTML文档大小:
$(document).width(); // 800
$(document).height(); // 3500

// 某个div的大小:
var div = $('#test-div');
div.width(); // 600
div.height(); // 300
div.width(400); // 设置CSS属性 width: 400px，是否生效要看CSS是否有效
```

```
div.height('200px'); // 设置CSS属性 height: 200px，是否生效要看CSS是否有效
```

`attr()`和`removeAttr()`方法用于操作DOM节点的属性：

```
// <div id="test-div" name="Test" start="1">...</div>
var div = $('#test-div');
div.attr('data'); // undefined, 属性不存在
div.attr('name'); // 'Test'
div.attr('name', 'Hello'); // div的name属性变为'Hello'
div.removeAttr('name'); // 删除name属性
div.attr('name'); // undefined
```

`prop()`方法和`attr()`类似，但是HTML5规定有一种属性在DOM节点中可以没有值，只有出现与不出现两种，例如：

```
<input id="test-radio" type="radio" name="test" checked value="1">
```

等价于：

```
<input id="test-radio" type="radio" name="test" checked="checked" value="1">
```

`attr()`和`prop()`对于属性`checked`处理有所不同：

```
var radio = $('#test-radio');
radio.attr('checked'); // 'checked'
radio.prop('checked'); // true
```

`prop()`返回值更合理一些。不过，用`is()`方法判断更好：

```
var radio = $('#test-radio');
radio.is(':checked'); // true
```

类似的属性还有`selected`，处理时最好用`is(':selected')`。

操作表单

对于表单元素，jQuery对象统一提供`val()`方法获取和设置对应的`value`属性：

```
/*
    <input id="test-input" name="email" value="">
    <select id="test-select" name="city">
        <option value="BJ" selected>Beijing</option>
        <option value="SH">Shanghai</option>
        <option value="SZ">Shenzhen</option>
```

```
    </select>
    <textarea id="test-textarea">Hello</textarea>
*/
var
    input = $('#test-input'),
    select = $('#test-select'),
    textarea = $('#test-textarea');

input.val(); // 'test'
input.val('abc@example.com'); // 文本框的内容已变为
abc@example.com

select.val(); // 'BJ'
select.val('SH'); // 选择框已变为Shanghai

textarea.val(); // 'Hello'
textarea.val('Hi'); // 文本区域已更新为'Hi'
```

可见，一个`val()`就统一了各种输入框的取值和赋值的问题。

修改DOM结构

直接使用浏览器提供的API对DOM结构进行修改，不但代码复杂，而且要针对浏览器写不同的代码。

有了jQuery，我们就专注于操作jQuery对象本身，底层的DOM操作由jQuery完成就可以了，这样一来，修改DOM也大大简化了。

添加DOM

要添加新的DOM节点，除了通过jQuery的`html()`这种暴力方法外，还可以用`append()`方法，例如：

```
<div id="test-div">
    <ul>
        <li><span>JavaScript</span></li>
        <li><span>Python</span></li>
        <li><span>Swift</span></li>
    </ul>
</div>
```

如何向列表新增一个语言？首先要拿到``节点：

```
var ul = $('#test-div>ul');
```

然后，调用`append()`传入HTML片段：

```
ul.append('<li><span>Haskell</span></li>');
```

除了接受字符串，`append()`还可以传入原始的DOM对象，jQuery对象和函数对象：

```
// 创建DOM对象:
var ps = document.createElement('li');
ps.innerHTML = '<span>Pascal</span>';
// 添加DOM对象:
ul.append(ps);

// 添加jQuery对象:
ul.append($('#scheme'));

// 添加函数对象:
ul.append(function (index, html) {
    return '<li><span>Language - ' + index + '</span>
</li>';
});
```

传入函数时，要求返回一个字符串、DOM对象或者jQuery对象。因为jQuery的`append()`可能作用于一组DOM节点，只有传入函数才能针对每个DOM生成不同的子节点。

`append()`把DOM添加到最后，`prepend()`则把DOM添加到最前。

另外注意，如果要添加的DOM节点已经存在于HTML文档中，它会首先从文档移除，然后再添加，也就是说，用`append()`，你可以移动一个DOM节点。

如果要把新节点插入到指定位置，例如，JavaScript和Python之间，那么，可以先定位到JavaScript，然后用`after()`方法：

```
var js = $('#test-div>ul>li:first-child');
js.after('<li><span>Lua</span></li>');
```

也就是说，同级节点可以用`after()`或者`before()`方法。

删除节点

要删除DOM节点，拿到jQuery对象后直接调用`remove()`方法就可以了。如果jQuery对象包含若干DOM节点，实际上可以一次删除多个DOM节点：

```
var li = $('#test-div>ul>li');
li.remove(); // 所有<li>全被删除
```

练习

除了列出的3种语言外，请再添加Pascal、Lua和Ruby，然后按字母顺序排序节点：

```
<!-- HTML结构 -->
<div id="test-div">
    <ul>
        <li><span>JavaScript</span></li>
        <li><span>Python</span></li>
        <li><span>Swift</span></li>
    </ul>
</div>
```

```
'use strict';
// 测试:
;(function () {
    var s = $('#test-div>ul>li').map(function () {
        return $(this).text();
    }).get().join(',');
    if (s === 'JavaScript,Lua,Pascal,Python,Ruby,Swift') {
        console.log('测试通过!');
    } else {
        console.log('测试失败: ' + s);
    }
})();
```

- JavaScript
- Python
- Swift

事件

因为JavaScript在浏览器中以单线程模式运行，页面加载后，一旦页面上所有的JavaScript代码被执行完后，就只能依赖触发事件来执行JavaScript代码。

浏览器在接收到用户的鼠标或键盘输入后，会自动在对应的DOM节点上触发相应的事件。如果该节点已经绑定了对应的JavaScript处理函数，该函数就会自动调用。

由于不同的浏览器绑定事件的代码都不太一样，所以用jQuery来写代码，就屏蔽了不同浏览器的差异，我们总是编写相同的代码。

举个例子，假设要在用户点击了超链接时弹出提示框，我们用jQuery这样绑定一个`click`事件：

```
/* HTML:
 *
 * <a id="test-link" href="#0">点我试试</a>
 *
 */

// 获取超链接的jQuery对象:
var a = $('#test-link');
a.on('click', function () {
    alert('Hello!');
});
```

实测：点我试试

`on`方法用来绑定一个事件，我们需要传入事件名称和对应的处理函数。

另一种更简化的写法是直接调用`click()`方法：

```
a.click(function () {
    alert('Hello!');
});
```

两者完全等价。我们通常用后面的写法。

jQuery能够绑定的事件主要包括：

鼠标事件

click: 鼠标单击时触发； dblclick：鼠标双击时触发； mouseenter：鼠标进入时触发； mouseleave：鼠标移出时触发； mousemove：鼠标在DOM内部移动时触发； hover：鼠标进入和退出时触发两个函数，相当于mouseenter加上mouseleave。

键盘事件

键盘事件仅作用在当前焦点的DOM上，通常是`和`。

keydown：键盘按下时触发； keyup：键盘松开时触发； keypress：按一次键后触发。

其他事件

focus：当DOM获得焦点时触发； blur：当DOM失去焦点时触发； change：当、或的内容改变时触发； submit：当提交时触发； ready：当页面被载入并且DOM树完成初始化后触发。

其中，ready仅作用于document对象。由于ready事件在DOM完成初始化后触发，且只触发一次，所以非常适合用来写其他的初始化代码。假设我们想给一个`表单绑定submit`事件，下面的代码没有预期的效果：

```
<html>
<head>
    <script>
        // 代码有误:
        $('#testForm).on('submit', function () {
            alert('submit!');
        });
    </script>
</head>
<body>
    <form id="testForm">
        ...
    </form>
</body>
```

因为JavaScript在此执行的时候，`尚未载入浏览器，所以$('#testForm)返回[]`，并没有绑定事件到任何DOM上。

所以我们自己的初始化代码必须放到document对象的ready事件中，保证DOM已完成初始化：

```
<html>
<head>
    <script>
        $(document).on('ready', function () {
            $('#testForm).on('submit', function () {
                alert('submit!');
            });
        });
    </script>
</head>
<body>
    <form id="testForm">
        ...
    </form>
</body>
```

这样写就没有问题了。因为相关代码会在DOM树初始化后再执行。

由于ready事件使用非常普遍，所以可以这样简化：

```
$(document).ready(function () {
    // on('submit', function)也可以简化:
    $('#testForm).submit(function () {
        alert('submit!');
    });
});
```

甚至还可以再简化为：

```
$(function () {
    // init...
});
```

上面的这种写法最为常见。如果你遇到`$(function () {...})`的形式，牢记这是`document`对象的`ready`事件处理函数。

完全可以反复绑定事件处理函数，它们会依次执行：

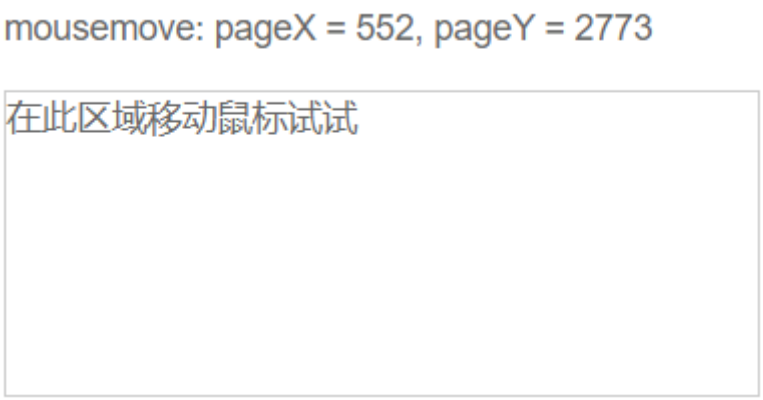
```
$(function () {
    console.log('init A...');
});
$(function () {
    console.log('init B...');
});
$(function () {
    console.log('init C...');
});
```

事件参数

有些事件，如`mousemove`和`keypress`，我们需要获取鼠标位置和按键的值，否则监听这些事件就没什么意义了。所有事件都会传入`Event`对象作为参数，可以从`Event`对象上获取到更多的信息：

```
$(function () {
    $('#testMouseMoveDiv').mousemove(function (e) {
        $('#testMouseMoveSpan').text('pageX = ' + e.pageX
+ ', pageY = ' + e.pageY);
    });
});
```

效果实测：



取消绑定

一个已被绑定的事件可以解除绑定，通过`off('click', function)`实现：

```
function hello() {
    alert('hello!');
}

a.click(hello); // 绑定事件

// 10秒钟后解除绑定:
setTimeout(function () {
    a.off('click', hello);
}, 10000);
```

需要特别注意的是，下面这种写法是无效的：

```
// 绑定事件:
a.click(function () {
    alert('hello!');
});

// 解除绑定:
a.off('click', function () {
    alert('hello!');
});
```

这是因为两个匿名函数虽然长得一模一样，但是它们是两个*不同*的函数对象，`off('click', function () {...})`无法移除已绑定的第一个匿名函数。

为了实现移除效果，可以使用`off('click')`一次性移除已绑定的`click`事件的所有处理函数。

同理，无参数调用`off()`一次性移除已绑定的所有类型的事件处理函数。

事件触发条件

一个需要注意的问题是，事件的触发总是由用户操作引发的。例如，我们监控文本框的内容改动：

```
var input = $('#test-input');
input.change(function () {
    console.log('changed...');
});
```

当用户在文本框中输入时，就会触发`change`事件。但是，如果用JavaScript代码去改动文本框的值，将*不会*触发`change`事件：

```
var input = $('#test-input');
input.val('change it!'); // 无法触发change事件
```

有些时候，我们希望用代码触发`change`事件，可以直接调用无参数的`change()`方法来触发该事件：

```
var input = $('#test-input');
input.val('change it!');
input.change(); // 触发change事件
```

`input.change()`相当于`input.trigger('change')`，它是`trigger()`方法的简写。

为什么我们希望手动触发一个事件呢？如果不这么做，很多时候，我们就得写两份一模一样的代码。

浏览器安全限制

在浏览器中，有些JavaScript代码只有在用户触发下才能执行，例如，`window.open()`函数：

```
// 无法弹出新窗口，将被浏览器屏蔽:
$(function () {
    window.open('/');
});
```

这些“敏感代码”只能由用户操作来触发：

```
var button1 = $('#testPopupButton1');
var button2 = $('#testPopupButton2');

function popupTestWindow() {
    window.open('/');
}

button1.click(function () {
    popupTestWindow();
});
```

```
button2.click(function () {
    // 不立刻执行popupTestWindow()，100毫秒后执行：
    setTimeout(popupTestWindow, 100);
});
```

当用户点击`button1`时，`click`事件被触发，由于`popupTestWindow()`在`click`事件处理函数内执行，这是浏览器允许的，而`button2`的`click`事件并未立刻执行`popupTestWindow()`，延迟执行的`popupTestWindow()`将被浏览器拦截。

效果实测：

练习

对如下的Form表单：

```
<!-- HTML结构 -->
<form id="test-form" action="test">
    <legend>请选择想要学习的编程语言：</legend>
    <fieldset>
        <p><label class="selectAll"><input
type="checkbox"> <span class="selectAll">全选</span><span
class="deselectAll">全不选</span></label> <a href="#0"
class="invertSelect">反选</a></p>
        <p><label><input type="checkbox" name="lang"
value="javascript"> JavaScript</label></p>
        <p><label><input type="checkbox" name="lang"
value="python"> Python</label></p>
        <p><label><input type="checkbox" name="lang"
value="ruby"> Ruby</label></p>
        <p><label><input type="checkbox" name="lang"
value="haskell"> Haskell</label></p>
        <p><label><input type="checkbox" name="lang"
value="scheme"> Scheme</label></p>
        <p><button type="submit">Submit</button></p>
    </fieldset>
</form>
```

绑定合适的事件处理函数，实现以下逻辑：

当用户勾上“全选”时，自动选中所有语言，并把“全选”变成“全不选”；

当用户去掉“全不选”时，自动不选中所有语言；

当用户点击“反选”时，自动把所有语言状态反转（选中的变为未选，未选的变为选中）；

当用户把所有语言都手动勾上时，“全选”被自动勾上，并变为“全不选”；

当用户手动去掉选中至少一种语言时，“全不选”自动被去掉选中，并变为“全选”。

```
'use strict';

var
    form = $('#test-form'),
    langs = form.find('[name=lang]'),
    selectAll = form.find('label.selectAll :checkbox'),
    selectAllLabel = form.find('label.selectAll
span.selectAll'),
    deselectAllLabel = form.find('label.selectAll
span.deselectAll'),
    invertSelect = form.find('a.invertSelect');

// 重置初始化状态:
form.find('*').show().off();
form.find(':checkbox').prop('checked', false).off();
deselectAllLabel.hide();
// 拦截form提交事件:
form.off().submit(function (e) {
    e.preventDefault();
    alert(form.serialize());
});
// TODO:绑定事件
// 测试:
console.log('请测试功能是否正常。');
```

动画

用JavaScript实现动画，原理非常简单：我们只需要以固定的时间间隔（例如，0.1秒），每次把DOM元素的CSS样式修改一点（例如，高宽各增加10%），看起来就像动画了。

但是要用JavaScript手动实现动画效果，需要编写非常复杂的代码。如果想要把动画效果用函数封装起来便于复用，那考虑的事情就更多了。

使用jQuery实现动画，代码已经简单得不能再简化了：只需要一行代码！

让我们先来看看jQuery内置的几种动画样式：

show / hide

直接以无参数形式调用`show()`和`hide()`，会显示和隐藏DOM元素。但是，只要传递一个时间参数进去，就变成了动画：

```
var div = $('#test-show-hide');
div.hide(3000); // 在3秒钟内逐渐消失
```

时间以毫秒为单位，但也可以是`'slow'`，`'fast'`这些字符串：

```
var div = $('#test-show-hide');
div.show('slow'); // 在0.6秒钟内逐渐显示
```

`toggle()`方法则根据当前状态决定是`show()`还是`hide()`。

效果实测：

slideUp / slideDown

你可能已经看出来了，`show()`和`hide()`是从左上角逐渐展开或收缩的，而`slideUp()`和`slideDown()`则是在垂直方向逐渐展开或收缩的。

`slideUp()`把一个可见的DOM元素收起来，效果跟拉上窗帘似的，`slideDown()`相反，而`slideToggle()`则根据元素是否可见来决定下一步动作：

```
var div = $('#test-slide');
div.slideUp(3000); // 在3秒钟内逐渐向上消失
```

效果实测：

fadeIn / fadeOut

`fadeIn()`和`fadeOut()`的动画效果是淡入淡出，也就是通过不断设置DOM元素的`opacity`属性来实现，而`fadeToggle()`则根据元素是否可见来决定下一步动作：

```
var div = $('#test-fade');
div.fadeOut('slow'); // 在0.6秒内淡出
```

自定义动画

如果上述动画效果还不能满足你的要求，那就祭出最后大招：`animate()`，它可以实现任意动画效果，我们需要传入的参数就是DOM元素最终的CSS状态和时间，jQuery在时间段内不断调整CSS直到达到我们设定的值：

```
var div = $('#test-animate');
div.animate({
    opacity: 0.25,
    width: '256px',
    height: '256px'
}, 3000); // 在3秒钟内CSS过渡到设定值
```

`animate()`还可以再传入一个函数，当动画结束时，该函数将被调用：

```
var div = $('#test-animate');
div.animate({
    opacity: 0.25,
    width: '256px',
    height: '256px'
}, 3000, function () {
    console.log('动画已结束');
    // 恢复至初始状态:
    $(this).css('opacity', '1.0').css('width',
'128px').css('height', '128px');
});
```

实际上这个回调函数参数对于基本动画也是适用的。

有了`animate()`，你就可以实现各种自定义动画效果了：

串行动画

jQuery的动画效果还可以串行执行，通过`delay()`方法还可以实现暂停，这样，我们可以实现更复杂的动画效果，而代码却相当简单：

```
var div = $('#test-animates');
```

```
// 动画效果：slideDown - 暂停 - 放大 - 暂停 - 缩小
div.slideDown(2000)
    .delay(1000)
    .animate({
        width: '256px',
        height: '256px'
    }, 2000)
    .delay(1000)
    .animate({
        width: '128px',
        height: '128px'
    }, 2000);
}
</script>
```

因为动画需要执行一段时间，所以jQuery必须不断返回新的Promise对象才能后续执行操作。简单地把动画封装在函数中是不够的。

效果实测：

为什么有的动画没有效果

你可能会遇到，有的动画如`slideUp()`根本没有效果。这是因为jQuery动画的原理是逐渐改变CSS的值，如`height`从`100px`逐渐变为`0`。但是很多不是block性质的DOM元素，对它们设置`height`根本就不起作用，所以动画也就没有效果。

此外，jQuery也没有实现对`background-color`的动画效果，用`animate()`设置`background-color`也没有效果。这种情况下可以使用CSS3的`transition`实现动画效果。

练习

在执行删除操作时，给用户显示一个动画比直接调用`remove()`要更好。请在表格删除一行的时候添加一个淡出的动画效果：

```
'use strict';

function deleteFirstTR() {
    var tr = $('#test-table>tbody>tr:visible').first();
}

deleteFirstTR();
```

Name	Email	Address	Status
Bart Simpson	bart.s@primary.school	Springfield	Active
Michael Scofield	m.scofield@escape.org	Fox River	Locked
Optimus Prime	prime@cybertron.org	Cybertron	Active
Peter Parker	spider@movie.org	New York	Active
Thor Odinson	thor@asgard.org	Asgard	Active

+ Add

AJAX

用JavaScript写AJAX前面已经介绍过了，主要问题就是不同浏览器需要写不同代码，并且状态和错误处理写起来很麻烦。

用jQuery的相关对象来处理AJAX，不但不需要考虑浏览器问题，代码也能大大简化。

ajax

jQuery在全局对象`jQuery`（也就是`$`）绑定了`ajax()`函数，可以处理AJAX请求。`ajax(url, settings)`函数需要接收一个URL和一个可选的`settings`对象，常用的选项如下：

- async：是否异步执行AJAX请求，默认为`true`，千万不要指定为`false`；
- method：发送的Method，缺省为`'GET'`，可指定为`'POST'`、`'PUT'`等；
- contentType：发送POST请求的格式，默认值为`'application/x-www-form-urlencoded; charset=UTF-8'`，也可以指定为`text/plain`、`application/json`；
- data：发送的数据，可以是字符串、数组或object。如果是GET请求，data将被转换成query附加到URL上，如果是POST请求，根据contentType把data序列化成合适的格式；
- headers：发送的额外的HTTP头，必须是一个object；
- dataType：接收的数据格式，可以指定为`'html'`、`'xml'`、`'json'`、`'text'`等，缺省情况下根据响应的`Content-Type`猜测。

下面的例子发送一个GET请求，并返回一个JSON格式的数据：

```
var jqxhr = $.ajax('/api/categories', {
    dataType: 'json'
});
// 请求已经发送了
```

不过，如何用回调函数处理返回的数据和出错时的响应呢？

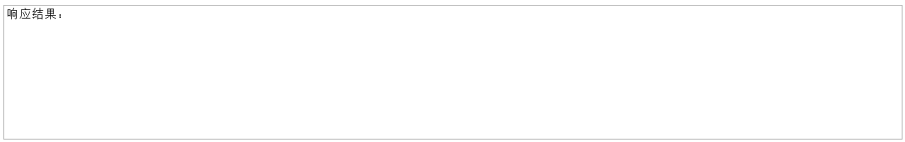
还记得Promise对象吗？jQuery的jqXHR对象类似一个Promise对象，我们可以用链式写法来处理各种回调：

```
'use strict';

function ajaxLog(s) {
```

```
    var txt = $('#test-response-text');
    txt.val(txt.val() + '\n' + s);
}

$('#test-response-text').val('');
var jqxhr = $.ajax('/api/categories', {
    dataType: 'json'
}).done(function (data) {
    ajaxLog('成功, 收到的数据: ' + JSON.stringify(data));
}).fail(function (xhr, status) {
    ajaxLog('失败: ' + xhr.status + ', 原因: ' + status);
}).always(function () {
    ajaxLog('请求完成: 无论成功或失败都会调用');
});
```



get

对常用的AJAX操作，jQuery提供了一些辅助方法。由于GET请求最常见，所以jQuery提供了`get()`方法，可以这么写：

```
var jqxhr = $.get('/path/to/resource', {
    name: 'Bob Lee',
    check: 1
});
```

第二个参数如果是object，jQuery自动把它变成query string然后加到URL后面，实际的URL是：

```
/path/to/resource?name=Bob%20Lee&check=1
```

这样我们就不用关心如何用URL编码并构造一个query string了。

post

`post()`和`get()`类似，但是传入的第二个参数默认被序列化为`application/x-www-form-urlencoded`：

```
var jqxhr = $.post('/path/to/resource', {
    name: 'Bob Lee',
    check: 1
});
```

实际构造的数据`name=Bob%20Lee&check=1`作为POST的body被发送。

getJSON

由于JSON用得越来越普遍，所以jQuery也提供了`getJSON()`方法来快速通过GET获取一个JSON对象：

```
var jqxhr = $.getJSON('/path/to/resource', {
    name: 'Bob Lee',
    check: 1
}).done(function (data) {
    // data已经被解析为JSON对象了
});
```

安全限制

jQuery的AJAX完全封装的是JavaScript的AJAX操作，所以它的安全限制和前面讲的用JavaScript写AJAX完全一样。

如果需要使用JSONP，可以在`ajax()`中设置`jsonp: 'callback'`，让jQuery实现JSONP跨域加载数据。

关于跨域的设置请参考浏览器 - AJAX一节中CORS的设置。

扩展

当我们使用jQuery对象的方法时，由于jQuery对象可以操作一组DOM，而且支持链式操作，所以用起来非常方便。

但是jQuery内置的方法永远不可能满足所有的需求。比如，我们想要高亮显示某些DOM元素，用jQuery可以这么实现：

```
$('span.hl').css('backgroundColor',
'#fffceb').css('color', '#d85030');

$('p a.hl').css('backgroundColor', '#fffceb').css('color',
'#d85030');
```

总是写重复代码可不好，万一以后还要修改字体就更麻烦了，能不能统一起来，写个`highlight()`方法？

```
$('span.hl').highlight();

$('p a.hl').highlight();
```

答案是肯定的。我们可以扩展jQuery来实现自定义方法。将来如果要修改高亮的逻辑，只需修改一处扩展代码。这种方式也称为编写jQuery插件。

编写jQuery插件

给jQuery对象绑定一个新方法是通过扩展`$.fn`对象实现的。让我们来编写第一个扩展——`highlight1()`：

```
$.fn.highlight1 = function () {
    // this已绑定为当前jQuery对象:
    this.css('backgroundColor', '#fffceb').css('color',
'#d85030');
    return this;
}
```

注意到函数内部的`this`在调用时被绑定为jQuery对象，所以函数内部代码可以正常调用所有jQuery对象的方法。

对于如下的HTML结构：

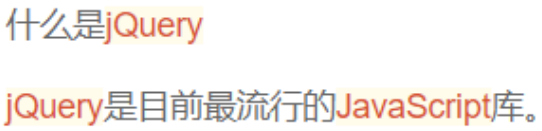
```
<!-- HTML结构 -->
<div id="test-highlight1">
    <p>什么是<span>jQuery</span></p>
    <p><span>jQuery</span>是目前最流行的
<span>JavaScript</span>库。</p>
</div>
```

来测试一下`highlight1()`的效果：

```
'use strict';
$('#test-highlight1 span').highlight1();
```

什么是jQuery

jQuery是目前最流行的JavaScript库。



细心的童鞋可能发现了，为什么最后要`return this;`？因为jQuery对象支持链式操作，我们自己写的扩展方法也要能继续链式下去：

```
$('span.hl').highlight1().slideDown();
```

不然，用户调用的时候，就不得不把上面的代码拆成两行。

但是这个版本并不完美。有的用户希望高亮的颜色能自己来指定，怎么办？

我们可以给方法加个参数，让用户自己把参数用对象传进去。于是我们有了第二个版本的`highlight2()`：

```
$.fn.highlight2 = function (options) {
    // 要考虑到各种情况:
    // options为undefined
    // options只有部分key
    var bgcolor = options && options.backgroundColor ||
'#fffceb';
    var color = options && options.color || '#d85030';
    this.css('backgroundColor', bgcolor).css('color',
color);
    return this;
}
```

对于如下HTML结构：

```
<!-- HTML结构 -->
<div id="test-highlight2">
    <p>什么是<span>jQuery</span> <span>Plugin</span></p>
    <p>编写<span>jQuery</span> <span>Plugin</span>可以用来扩
展<span>jQuery</span>的功能。</p>
</div>
```

来实测一下带参数的`highlight2()`：

```
'use strict';
$('#test-highlight2 span').highlight2({
    backgroundColor: '#00a8e6',
    color: '#ffffff'
});
```

什么是jQuery Plugin

编写jQuery Plugin可以用来扩展jQuery的功能。

对于默认值的处理，我们用了一个简单的`&&`和`||`短路操作符，总能得到一个有效的值。

另一种方法是使用jQuery提供的辅助方法`$.extend(target, obj1, obj2, ...)`，它把多个object对象的属性合并到第一个target对象中，遇到同名属性，总是使用靠后的对象的值，也就是越往后优先级越高：

```
// 把默认值和用户传入的options合并到对象{}中并返回:
var opts = $.extend({}, {
    backgroundColor: '#00a8e6',
    color: '#ffffff'
}, options);
```

紧接着用户对`highlight2()`提出了意见：每次调用都需要传入自定义的设置，能不能让我自己设定一个缺省值，以后的调用统一使用无参数的`highlight2()`？

也就是说，我们设定的默认值应该能允许用户修改。

那默认值放哪比较合适？放全局变量肯定不合适，最佳地点是`$.fn.highlight2`这个函数对象本身。

于是最终版的`highlight()`终于诞生了：

```
$.fn.highlight = function (options) {
    // 合并默认值和用户设定值:
    var opts = $.extend({}, $.fn.highlight.defaults,
options);
    this.css('backgroundColor',
opts.backgroundColor).css('color', opts.color);
    return this;
}

// 设定默认值:
$.fn.highlight.defaults = {
    color: '#d85030',
    backgroundColor: '#fff8de'
}
```

这次用户终于满意了。用户使用时，只需一次性设定默认值：

```
$.fn.highlight.defaults.color = '#fff';
$.fn.highlight.defaults.backgroundColor = '#000';
```

然后就可以非常简单地调用`highlight()`了。

对如下的HTML结构：

```
<!-- HTML结构 -->
<div id="test-highlight">
    <p>如何编写<span>jQuery</span> <span>Plugin</span></p>
    <p>编写<span>jQuery</span> <span>Plugin</span>，要设置
<span>默认值</span>，并允许用户修改<span>默认值</span>，或者运行
时传入<span>其他值</span>。</p>
</div>
```

实测一下修改默认值的效果：

```
'use strict';
$.fn.highlight.defaults.color = '#659f13';
$.fn.highlight.defaults.backgroundColor = '#f2fae3';

$('#test-highlight p:first-child span').highlight();

$('#test-highlight p:last-child span').highlight({
    color: '#dd1144'
});
```

如何编写jQuery Plugin

编写jQuery Plugin，要设置默认值，并允许用户修改默认值，或者运行时传入其他值。

最终，我们得出编写一个jQuery插件的原则：

1. 给`$.fn`绑定函数，实现插件的代码逻辑；
2. 插件函数最后要`return this;`以支持链式调用；
3. 插件函数要有默认值，绑定在`$.fn..defaults`上；
4. 用户在调用时可传入设定值以便覆盖默认值。

针对特定元素的扩展

我们知道jQuery对象的有些方法只能作用在特定DOM元素上，比如`submit()`方法只能针对`form`。如果我们编写的扩展只能针对某些类型的DOM元素，应该怎么写？

还记得jQuery的选择器支持`filter()`方法来过滤吗？我们可以借助这个方法来实现针对特定元素的扩展。

举个例子，现在我们要给所有指向外链的超链接加上跳转提示，怎么做？

先写出用户调用的代码：

```
$('#main a').external();
```

然后按照上面的方法编写一个`external`扩展：

```
$.fn.external = function () {
    // return返回的each()返回结果，支持链式调用:
    return this.filter('a').each(function () {
        // 注意: each()内部的回调函数的this绑定为DOM本身!
        var a = $(this);
```

```
        var url = a.attr('href');
        if (url && (url.indexOf('http://')===0 ||
url.indexOf('https://')===0)) {
            a.attr('href', '#0')
             .removeAttr('target')
             .append(' <i class="uk-icon-external-link">
</i>')
             .click(function () {
                if(confirm('你确定要前往' + url + '? ')) {
                    window.open(url);
                }
            });
        }
    });
}
```

对如下的HTML结构：

```
<!-- HTML结构 -->
<div id="test-external">
    <p>如何学习<a href="http://jquery.com">jQuery</a>? </p>
    <p>首先，你要学习<a
href="/wiki/1022910821149312">JavaScript</a>，并了解基本的<a
href="https://developer.mozilla.org/en-
US/docs/Web/HTML">HTML</a>。</p>
</div>
```

实测外链效果：

```
'use strict';
$('#test-external a').external();
```

如何学习jQuery?

首先，你要学习JavaScript，并了解基本的HTML。

如何学习jQuery?

首先，你要学习JavaScript，并了解基本的HTML。

小结

扩展jQuery对象的功能十分简单，但是我们要遵循jQuery的原则，编写的扩展方法能支持链式调用、具备默认值和过滤特定元素，使得扩展方法看上去和jQuery本身的方法没有什么区别。