

jQuery®

专业人士笔记

jQuery®

Notes for Professionals



50+ 页
专业提示和技巧

50+ pages
of professional hints and tricks

目录

关于

第1章：jQuery入门

第1.1节：入门

第1.2节：避免命名空间冲突

第1.3节：jQuery命名空间（“jQuery”和“\$”）

第1.4节：在没有jQuery的页面上通过控制台加载jQuery

第1.5节：在HTML页面的头部包含脚本标签

第1.6节：jQuery对象

第2章：选择器

第2.1节：概述

第2.2节：选择器的类型

第2.3节：选择器缓存

第2.4节：组合选择器

第2.5节：DOM元素作为选择器

第2.6节：HTML字符串作为选择器

第3章：各个函数

第3.1节：jQuery each 函数

第4章：属性

第4.1节：attr() 和 prop() 的区别

第4.2节：获取 HTML 元素的属性值

第4.3节：设置HTML属性的值

第4.4节：移除属性

第5章：文档就绪事件

第5.1节：什么是文档就绪以及我该如何使用它？

第5.2节：jQuery 2.2.3及更早版本

第5.3节：jQuery 3.0

第5.4节：在ready()中绑定事件和操作DOM

第5.5节：\$(document).ready()与\$(window).load()的区别

第5.6节：jQuery(fn)与在</body>之前执行代码的区别

第6章：事件

第6.1节：委托事件

第6.2节：附加和移除事件处理程序

第6.3节：通过jQuery开启和关闭特定事件。（命名监听器）

第6.4节：originalEvent

第6.5节：不使用ID的重复元素事件

第6.6节：文档加载事件.load()

第7章：DOM操作

第7.1节：创建DOM元素

第7.2节：操作元素类

第7.3节：其他API方法

第8章：DOM遍历

第8.1节：选择元素的子元素

第8.2节：获取下一个元素

第8.3节：获取上一个元素

第8.4节：筛选选择项

第8.5节：find() 方法

关于	1
第1章：jQuery入门	2
第1.1节：入门	2
第1.2节：避免命名空间冲突	3
第1.3节：jQuery命名空间（“jQuery”和“\$”）	4
第1.4节：在没有jQuery的页面上通过控制台加载jQuery	5
第1.5节：在HTML页面的头部包含脚本标签	5
第1.6节：jQuery对象	7
第2章：选择器	8
第2.1节：概述	8
第2.2节：选择器的类型	8
第2.3节：选择器缓存	10
第2.4节：组合选择器	11
第2.5节：DOM元素作为选择器	13
第2.6节：HTML字符串作为选择器	13
第3章：各个函数	15
第3.1节：jQuery each 函数	15
第4章：属性	16
第4.1节：attr() 和 prop() 的区别	16
第4.2节：获取 HTML 元素的属性值	16
第4.3节：设置HTML属性的值	17
第4.4节：移除属性	17
第5章：文档就绪事件	18
第5.1节：什么是文档就绪以及我该如何使用它？	18
第5.2节：jQuery 2.2.3及更早版本	18
第5.3节：jQuery 3.0	19
第5.4节：在ready()中绑定事件和操作DOM	19
第5.5节：\$(document).ready()与\$(window).load()的区别	20
第5.6节：jQuery(fn)与在</body>之前执行代码的区别	21
第6章：事件	22
第6.1节：委托事件	22
第6.2节：附加和移除事件处理程序	23
第6.3节：通过jQuery开启和关闭特定事件。（命名监听器）	24
第6.4节：originalEvent	25
第6.5节：不使用ID的重复元素事件	25
第6.6节：文档加载事件.load()	26
第7章：DOM操作	27
第7.1节：创建DOM元素	27
第7.2节：操作元素类	27
第7.3节：其他API方法	29
第8章：DOM遍历	31
第8.1节：选择元素的子元素	31
第8.2节：获取下一个元素	31
第8.3节：获取上一个元素	31
第8.4节：筛选选择项	32
第8.5节：find() 方法	33

Contents

About

Chapter 1: Getting started with jQuery

Section 1.1: Getting Started

Section 1.2: Avoiding namespace collisions

Section 1.3: jQuery Namespace (“jQuery” and “\$”)

Section 1.4: Loading jQuery via console on a page that does not have it

Section 1.5: Include script tag in head of HTML page

Section 1.6: The jQuery Object

Chapter 2: Selectors

Section 2.1: Overview

Section 2.2: Types of Selectors

Section 2.3: Caching Selectors

Section 2.4: Combining selectors

Section 2.5: DOM Elements as selectors

Section 2.6: HTML strings as selectors

Chapter 3: Each function

Section 3.1: jQuery each function

Chapter 4: Attributes

Section 4.1: Differece between attr() and prop()

Section 4.2: Get the attribute value of a HTML element

Section 4.3: Setting value of HTML attribute

Section 4.4: Removing attribute

Chapter 5: document-ready event

Section 5.1: What is document-ready and how should I use it?

Section 5.2: jQuery 2.2.3 and earlier

Section 5.3: jQuery 3.0

Section 5.4: Attaching events and manipulating the DOM inside ready()

Section 5.5: Difference between \$(document).ready() and \$(window).load()

Section 5.6: Difference between jQuery(fn) and executing your code before </body>

Chapter 6: Events

Section 6.1: Delegated Events

Section 6.2: Attach and Detach Event Handlers

Section 6.3: Switching specific events on and off via jQuery. (Named Listeners)

Section 6.4: originalEvent

Section 6.5: Events for repeating elements without using ID's

Section 6.6: Document Loading Event .load()

Chapter 7: DOM Manipulation

Section 7.1: Creating DOM elements

Section 7.2: Manipulating element classes

Section 7.3: Other API Methods

Chapter 8: DOM Traversing

Section 8.1: Select children of element

Section 8.2: Get next element

Section 8.3: Get previous element

Section 8.4: Filter a selection

Section 8.5: find() method

About	1
Chapter 1: Getting started with jQuery	2
Section 1.1: Getting Started	2
Section 1.2: Avoiding namespace collisions	3
Section 1.3: jQuery Namespace (“jQuery” and “\$”)	4
Section 1.4: Loading jQuery via console on a page that does not have it	5
Section 1.5: Include script tag in head of HTML page	5
Section 1.6: The jQuery Object	7
Chapter 2: Selectors	8
Section 2.1: Overview	8
Section 2.2: Types of Selectors	8
Section 2.3: Caching Selectors	10
Section 2.4: Combining selectors	11
Section 2.5: DOM Elements as selectors	13
Section 2.6: HTML strings as selectors	13
Chapter 3: Each function	15
Section 3.1: jQuery each function	15
Chapter 4: Attributes	16
Section 4.1: Differece between attr() and prop()	16
Section 4.2: Get the attribute value of a HTML element	16
Section 4.3: Setting value of HTML attribute	17
Section 4.4: Removing attribute	17
Chapter 5: document-ready event	18
Section 5.1: What is document-ready and how should I use it?	18
Section 5.2: jQuery 2.2.3 and earlier	18
Section 5.3: jQuery 3.0	19
Section 5.4: Attaching events and manipulating the DOM inside ready()	19
Section 5.5: Difference between \$(document).ready() and \$(window).load()	20
Section 5.6: Difference between jQuery(fn) and executing your code before </body>	21
Chapter 6: Events	22
Section 6.1: Delegated Events	22
Section 6.2: Attach and Detach Event Handlers	23
Section 6.3: Switching specific events on and off via jQuery. (Named Listeners)	24
Section 6.4: originalEvent	25
Section 6.5: Events for repeating elements without using ID's	25
Section 6.6: Document Loading Event .load()	26
Chapter 7: DOM Manipulation	27
Section 7.1: Creating DOM elements	27
Section 7.2: Manipulating element classes	27
Section 7.3: Other API Methods	29
Chapter 8: DOM Traversing	31
Section 8.1: Select children of element	31
Section 8.2: Get next element	31
Section 8.3: Get previous element	31
Section 8.4: Filter a selection	32
Section 8.5: find() method	33

第8.6节：遍历jQuery元素列表	34
第8.7节：选择兄弟元素	34
第8.8节：closest() 方法	34
第9章：CSS 操作	36
第9.1节：CSS – 获取器和设置器	36
第9.2节：数值属性的递增/递减	36
第9.3节：设置CSS属性	37
第9.4节：获取CSS属性	37
第10章：元素可见性	38
第10.1节：概述	38
第10.2节：切换可能性	38
第11章：追加	40
第11.1节：高效的连续.append()使用	40
第11.2节：jQuery追加	43
第11.3节：向容器追加元素	43
第12章：前置操作	45
第12.1节：向容器前置元素	45
第12.2节：前置方法	45
第13章：获取和设置元素的宽度和高度	47
第13.1节：获取和设置宽度与高度（忽略边框）	47
第13.2节：获取和设置innerWidth和innerHeight（忽略内边距和边框）	47
第13.3节：获取和设置outerWidth和outerHeight（包括内边距和边框）	47
第14章：jQuery .animate() 方法	48
第14.1节：带回调的动画	48
第15章：jQuery Deferred对象和Promise	50
第15.1节：jQuery ajax() 的 success、error 与 .done()、.fail()	50
第15.2节：基本的 Promise 创建	50
第16章：Ajax	52
第16.1节：使用 \$.ajax() 处理 HTTP 响应代码	52
第16.2节：使用Ajax提交表单	53
第16.3节：综合示例	53
第16.4节：Ajax文件上传	55
第17章：复选框全选及其他复选框自动选中/取消选中 更改	58
第17.1节：两个全选复选框与对应分组复选框	58
第18章：插件	59
第18.1节：插件 - 入门	59
鸣谢	61
你可能也喜欢	64

Section 8.6: Iterating over list of jQuery elements	34
Section 8.7: Selecting siblings	34
Section 8.8: closest() method	34
Chapter 9: CSS Manipulation	36
Section 9.1: CSS – Getters and Setters	36
Section 9.2: Increment/Decrement Numeric Properties	36
Section 9.3: Set CSS property	37
Section 9.4: Get CSS property	37
Chapter 10: Element Visibility	38
Section 10.1: Overview	38
Section 10.2: Toggle possibilities	38
Chapter 11: Append	40
Section 11.1: Efficient consecutive .append() usage	40
Section 11.2: jQuery append	43
Section 11.3: Appending an element to a container	43
Chapter 12: Prepend	45
Section 12.1: Prepending an element to a container	45
Section 12.2: Prepend method	45
Chapter 13: Getting and setting width and height of an element	47
Section 13.1: Getting and setting width and height (ignoring border)	47
Section 13.2: Getting and setting innerWidth and innerHeight (ignoring padding and border)	47
Section 13.3: Getting and setting outerWidth and outerHeight (including padding and border)	47
Chapter 14: jQuery .animate() Method	48
Section 14.1: Animation with callback	48
Chapter 15: jQuery Deferred objects and Promises	50
Section 15.1: jQuery ajax() success, error VS .done()、.fail()	50
Section 15.2: Basic promise creation	50
Chapter 16: Ajax	52
Section 16.1: Handling HTTP Response Codes with \$.ajax()	52
Section 16.2: Using Ajax to Submit a Form	53
Section 16.3: All in one examples	53
Section 16.4: Ajax File Uploads	55
Chapter 17: Checkbox Select all with automatic check/uncheck on other checkbox change	58
Section 17.1: 2 select all checkboxes with corresponding group checkboxes	58
Chapter 18: Plugins	59
Section 18.1: Plugins – Getting Started	59
Credits	61
You may also like	64

请随意免费分享此PDF，
本书最新版本可从以下网址下载：

<https://goalkicker.com/jQueryBook>

本jQuery® 专业笔记一书汇编自[Stack Overflow](#)

[文档](#)，内容由Stack Overflow的优秀人士撰写。

文本内容采用知识共享署名-相同方式共享许可协议发布，详见本书末尾对各章节贡献者的致谢。图片版权归各自所有者所有，除非另有说明。

本书为非官方免费书籍，旨在教育用途，与官方jQuery®组织或公司及Stack Overflow无关。所有商标及注册商标均为其各自公司所有。

本书所提供的信息不保证正确或准确，使用风险自负。

请将反馈和更正发送至web@petercv.com

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:

<https://goalkicker.com/jQueryBook>

This *jQuery® Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official jQuery® group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：jQuery入门

版本	笔记	发布日期
1.0	第一个稳定版本	2006-08-26
1.1		2007-01-14
1.2		2007-09-10
1.3	Sizzle 引入核心	2009-01-14
1.4		2010-01-14
1.5	延迟回调管理, ajax模块重写	2011-01-31
1.6	在attr()和val()方法中显著提升性能	2011-05-03
1.7	新的事件API：on()和off()。	2011-11-03
1.8	Sizzle 重写, 改进动画和\$(html, props)的灵活性。	2012-08-09
1.9	移除已废弃接口并进行代码清理	2013-01-15
1.10	合并了来自1.9和2.0测试版周期报告的错误修复和差异	2013-05-24
1.11		2014-01-24
1.12		2016-01-08
2.0	放弃对 IE 6-8 的支持, 以提升性能并减小体积	2013-04-18
2.1		2014-01-24
2.2		2016-01-08
3.0	对某些 jQuery 自定义选择器进行了大幅加速	2016-06-09
3.1	不再有静默错误	2016-07-07
3.2	不再有静默错误	2017-03-16
3.3	不再有静默错误	2018-01-19

第1.1节：入门

创建一个文件 hello.html, 内容如下：

```
<!DOCTYPE html>
<html>
<head>
  <title>你好，世界！</title>
</head>
<body>
  <div>
    <p id="hello">一些随机文本</p>
  </div>
  <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
  <script>
$(document).ready(function() {
  $('#hello').text('Hello, World!');
});
</script>
</body>
</html>
```

[JSBin 在线演示](#)

在网页浏览器中打开此文件。结果你将看到一个页面，显示文本：Hello, World!

代码说明

Chapter 1: Getting started with jQuery

Version	Notes	Release Date
1.0	First stable release	2006-08-26
1.1		2007-01-14
1.2		2007-09-10
1.3	Sizzle introduced into core	2009-01-14
1.4		2010-01-14
1.5	Deferred callback management, ajax module rewrite	2011-01-31
1.6	Significant performance gains in the attr() and val() methods	2011-05-03
1.7	New Event APIs: on() and off() .	2011-11-03
1.8	Sizzle rewritten, improved animations and \$(html, props) flexibility.	2012-08-09
1.9	Removal of deprecated interfaces and code cleanup	2013-01-15
1.10	Incorporated bug fixes and differences reported from both the 1.9 and 2.0 beta cycles	2013-05-24
1.11		2014-01-24
1.12		2016-01-08
2.0	Dropped IE 6-8 support for performance improvements and reduction in size	2013-04-18
2.1		2014-01-24
2.2		2016-01-08
3.0	Massive speedups for some jQuery custom selectors	2016-06-09
3.1	No More Silent Errors	2016-07-07
3.2	No More Silent Errors	2017-03-16
3.3	No More Silent Errors	2018-01-19

Section 1.1: Getting Started

Create a file `hello.html` with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello, World!</title>
</head>
<body>
  <div>
    <p id="hello">Some random text</p>
  </div>
  <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
  <script>
$(document).ready(function() {
  $('#hello').text('Hello, World!');
});
</script>
</body>
</html>
```

[Live Demo on JSBin](#)

Open this file in a web browser. As a result you will see a page with the text: Hello, World!

Explanation of code

1. 从 jQuery CDN 加载 jQuery 库：_____

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
```

这会引入 \$ 全局变量，它是 jQuery 函数和命名空间的别名。

请注意，包含 jQuery 时最常见的错误之一是未能在任何可能依赖或使用它的其他脚本或库之前加载该库。

2. _____ 延迟执行一个函数，当 DOM（文档对象模型）被检测为 "ready" 时由 jQuery:

```
// 当 `document` 处于 `ready` 状态时，执行此函数 `...`
$(文档).准备(函数() { ... });

// 一个常用的简写版本（行为与上述相同）
$(函数() { ... });
```

3. 一旦 DOM 准备就绪，jQuery 会执行上面显示的回调函数。在我们的函数内部，只有一个调用，它完成两个主要任务：

- 1. 获取属性id等于hello的元素（我们的选择器#hello）。使用选择器作为_____ 传入的参数是 jQuery 功能和命名的核心；整个库本质上是从扩展 document.querySelectorAllMDN 发展而来的。
- 2. 将选定元素内的text()设置为Hello, World!。

```
#    ↓ - 传递一个 `selector` 给 `$` jQuery，返回我们的元素
$('#hello').text('Hello, World!');
#           ↑ - 设置元素上的文本
```

更多信息请参阅jQuery - 文档页面。_____

第1.2节：避免命名空间冲突

除 jQuery 外，其他库也可能使用 \$ 作为别名。这可能导致这些库与

jQuery 之间发生冲突。

要释放\$以供其他库使用：

```
jQuery.noConflict();
```

调用此函数后，\$ 不再是 jQuery 的别名。但是，你仍然可以使用变量 jQuery 本身来访问 jQuery 函数：

```
jQuery('#hello').text('Hello, World!');
```

你也可以选择将另一个变量赋值为 jQuery 的别名：

```
var jqy = jQuery.noConflict();
```

1. Loads the jQuery library from the jQuery [CDN](#):

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
```

This introduces the \$ global variable, an alias for the jQuery function and namespace.

Be aware that one of the most common mistakes made when including jQuery is failing to load the library BEFORE any other scripts or libraries that may depend on or make use of it.

2. Defers a function to be executed when the DOM (Document Object Model) is detected to be "ready" by jQuery:

```
// When the `document` is `ready`, execute this function `...`
$(document).ready(function() { ... });

// A commonly used shorthand version (behaves the same as the above)
$(function() { ... });
```

3. Once the DOM is ready, jQuery executes the callback function shown above. Inside of our function, there is only one call which does 2 main things:

- 1. Gets the element with the id attribute equal to hello (our selector #hello). Using a selector as the passed argument is the core of jQuery's functionality and naming; the entire library essentially evolved from extending document.querySelectorAllMDN.
- 2. Set the text() inside the selected element to Hello, World!.

```
#    ↓ - Pass a `selector` to `$` jQuery，returns our element
$('#hello').text('Hello, World!');
#           ↑ - Set the Text on the element
```

For more refer to the [jQuery - Documentation](#) page.

Section 1.2: Avoiding namespace collisions

Libraries other than jQuery may also use \$ as an alias. This can cause interference between those libraries and jQuery.

To release \$ for use with other libraries:

```
jQuery.noConflict();
```

After calling this function, \$ is no longer an alias for jQuery. However, you can still use the variable jQuery itself to access jQuery functions:

```
jQuery('#hello').text('Hello, World!');
```

Optionally, you can assign a different variable as an alias for jQuery:

```
var jqy = jQuery.noConflict();
```

```
jqy('#hello').text('Hello, World!');
```

相反地，为了防止其他库干扰 jQuery，你可以将 jQuery 代码包裹在一个立即调用的函数表达式（IIFE）中，并传入 jQuery 作为参数：

```
(function($) {
    $(document).ready(function() {
        $('#hello').text('Hello, World!');
    });
})(jQuery);
```

在这个立即调用的函数表达式（IIFE）内部，\$ 仅是 jQuery 的别名。

另一种简单的方法来保护 jQuery 的\$别名并确保 DOM 已准备好：

```
jQuery(function( $ ) { // DOM 已准备好
    // 现在你可以自由使用 $ 别名
    $('#hello').text('Hello, World!');
});
```

总结一下，

- jQuery.noConflict() ：\$ 不再指向 jQuery，而变量 jQuery 仍然指向。
- var jQuery2 = jQuery.noConflict() - \$ 不再指向 jQuery，而变量 jQuery 仍然指向，变量 jQuery2 也指向。

现在，有第三种情况——如果我们只想让 jQuery 在 jQuery2 中可用怎么办？使用，

```
var jQuery2 = jQuery.noConflict(true)
```

这将导致 \$ 和 jQuery 都不指向 jQuery。

当需要在同一页面加载多个版本的 jQuery 时，这非常有用。

```
<script src='https://code.jquery.com/jquery-1.12.4.min.js'></script>
<script>
    var jQuery1 = jQuery.noConflict(true);
</script>
<script src='https://code.jquery.com/jquery-3.1.0.min.js'></script>
<script>
    // 这里，jQuery1 指的是 jQuery 1.12.4，而 $ 和 jQuery 指的是 jQuery 3.1.0。
</script>
```

<https://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries/>

第1.3节：jQuery命名空间（“jQuery”和“\$”）

jQuery 是编写任何 jQuery 代码的起点。它可以用作函数 jQuery(...) 或变量 jQuery.foo。

\$ 是 jQuery 的别名，二者通常可以互换使用（除非 jQuery.noConflict(); 已被使用——参见避免命名空间冲突）。

假设我们有以下这段 HTML 代码——

```
<div id="demo_div" class="demo"></div>
```

```
jqy('#hello').text('Hello, World!');
```

Conversely, to prevent other libraries from interfering with jQuery, you can wrap your jQuery code in an immediately invoked function expression (IIFE) and pass in jQuery as the argument:

```
(function($) {
    $(document).ready(function() {
        $('#hello').text('Hello, World!');
    });
})(jQuery);
```

Inside this IIFE, \$ is an alias for jQuery only.

Another simple way to **secure jQuery's \$ alias and make sure DOM is ready**:

```
jQuery(function( $ ) { // DOM is ready
    // You're now free to use $ alias
    $('#hello').text('Hello, World!');
});
```

To summarize,

- jQuery.noConflict() ：\$ no longer refers to jQuery, while the variable jQuery does.
- **var** jQuery2 = jQuery.noConflict() - \$ no longer refers to jQuery, while the variable jQuery does and so does the variable jQuery2.

Now, there exists a third scenario - What if we want jQuery to be available **only in jQuery2**? Use,

```
var jQuery2 = jQuery.noConflict(true)
```

This results in neither \$ nor jQuery referring to jQuery.

This is useful when multiple versions of jQuery are to be loaded onto the same page.

```
<script src='https://code.jquery.com/jquery-1.12.4.min.js'></script>
<script>
    var jQuery1 = jQuery.noConflict(true);
</script>
<script src='https://code.jquery.com/jquery-3.1.0.min.js'></script>
<script>
    // Here, jQuery1 refers to jQuery 1.12.4 while, $ and jQuery refers to jQuery 3.1.0.
</script>
```

<https://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries/>

Section 1.3: jQuery Namespace ("jQuery" and "\$")

jQuery is the starting point for writing any jQuery code. It can be used as a function jQuery(...) or a variable jQuery.foo.

\$ is an alias for jQuery and the two can usually be interchanged for each other (except where jQuery.noConflict(); has been used - see Avoiding namespace collisions).

Assuming we have this snippet of HTML -

```
<div id="demo_div" class="demo"></div>
```

我们可能想使用 jQuery 向这个 div 添加一些文本内容。为此，我们可以使用 jQuery 的 text() 函数。这个函数可以用 jQuery 或 \$ 来编写。即 -

```
jQuery("#demo_div").text("演示文本！");
```

或者 -

```
$("#demo_div").text("演示文本！");
```

两者都会产生相同的最终HTML -


```
<div id="demo_div" class="demo">演示文本！</div>
```

由于\$比jQuery更简洁，通常更推荐使用这种方式编写jQuery代码。

jQuery使用CSS选择器，上例中使用了ID选择器。有关jQuery中选择器的更多信息，请参见选择器类型。

第1.4节：在没有jQuery的页面上通过控制台加载jQuery

有时需要处理不使用jQuery的页面，而大多数开发者习惯于随时使用jQuery。

在这种情况下，可以使用Chrome开发者工具控制台（ F12）手动在已加载页面上添加 jQuery，方法是运行以下命令：

```
var j = document.createElement('script');
j.onload = function(){ jQuery.noConflict(); };
j.src = "https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js";
document.getElementsByTagName('head')[0].appendChild(j);
```

你想要的版本可能与上述（1.12.4）不同，你可以在这里获取所需版本的链接。[_____](#)

第1.5节：在HTML页面的head中包含script标签

要从官方CDN加载jQuery，请访问jQuery网站。你会看到一系列不同版本和格式可用。

We might want to use jQuery to add some text content to this div. To do this we could use the jQuery text() function. This could be written using either jQuery or \$. i.e. -

```
jQuery("#demo_div").text("Demo Text!");
```

Or -

```
$("#demo_div").text("Demo Text!");
```

Both will result in the same final HTML -


```
<div id="demo_div" class="demo">Demo Text!</div>
```

As \$ is more concise than jQuery it is the generally the preferred method of writing jQuery code.

jQuery uses CSS selectors and in the example above an ID selector was used. For more information on selectors in jQuery see types of selectors.

Section 1.4: Loading jQuery via console on a page that does not have it

Sometimes one has to work with pages that are not using jQuery while most developers are used to have jQuery handy.

In such situations one can use Chrome Developer Tools console ( F12) to manually add jQuery on a loaded page by running following:

```
var j = document.createElement('script');
j.onload = function(){ jQuery.noConflict(); };
j.src = "https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js";
document.getElementsByTagName('head')[0].appendChild(j);
```

Version you want might differ from above(1.12.4) you can get the link for [one you need here](#).

Section 1.5: Include script tag in head of HTML page

To load jQuery from the official CDN, go to the jQuery website. You'll see a list of different versions and formats available.

jQuery CDN – Latest Stable Versions

Powered by [MaxCDN](#)

jQuery Core

Showing the latest stable release in each major branch. [See all versions of jQuery Core.](#)

jQuery 3.x

- jQuery Core 3.1.0 - [uncompressed](#), [minified](#), [slim](#), [slim minified](#)

jQuery 2.x

- jQuery Core 2.2.4 - [uncompressed](#), [minified](#)

jQuery 1.x

- jQuery Core 1.12.4 - [uncompressed](#), [minified](#)

现在，复制你想加载的jQuery版本的源代码。假设你想加载**jQuery 2.X**，点击**未压缩或压缩**标签，会显示类似如下内容：



复制完整代码（或点击复制图标），并将其粘贴到 html 的<head>或<body>中。

最佳做法是在head标签中使用async属性加载任何外部JavaScript库。以下是一个演示：

```
<!DOCTYPE html>
<html>
  <head>
    <title>加载 jquery-2.2.4</title>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>
  </head>
  <body>
    <p>此页面已加载jquery。</p>
  </body>
</html>
```

jQuery CDN – Latest Stable Versions

Powered by [MaxCDN](#)

jQuery Core

Showing the latest stable release in each major branch. [See all versions of jQuery Core.](#)

jQuery 3.x

- jQuery Core 3.1.0 - [uncompressed](#), [minified](#), [slim](#), [slim minified](#)

jQuery 2.x

- jQuery Core 2.2.4 - [uncompressed](#), [minified](#)

jQuery 1.x

- jQuery Core 1.12.4 - [uncompressed](#), [minified](#)

Now, copy the source of the version of jQuery, you want to load. Suppose, you want to load **jQuery 2.X**, click **uncompressed** or **minified** tag which will show you something like this:



Copy the full code (or click on the copy icon) and paste it in the <head> or <body> of your html.

The best practice is to load any external JavaScript libraries at the head tag with the async attribute. Here is a demonstration:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading jquery-2.2.4</title>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>
  </head>
  <body>
    <p>This page is loaded with jquery.</p>
  </body>
</html>
```

使用async属性时需注意，因为JavaScript库会异步加载并在可用时立即执行。如果包含两个库，且第二个库依赖于第一个库，那么如果第二个库在第一个库之前加载并执行，可能会抛出错误并导致应用程序崩溃。

第1.6节：jQuery对象

每次调用jQuery时，无论是使用\$()还是jQuery()，内部都会创建一个new的jQuery实例。以下是显示新实例的源代码：

```
// 定义jQuery的本地副本
jQuery = function( selector, context ) {

    // jQuery 对象实际上只是经过“增强”的 init 构造函数
    // 如果调用 jQuery 则需要 init (如果未包含则允许抛出错误)
    return new jQuery.fn.init( selector, context );
}
```

jQuery 内部将其原型称为 .fn，这里使用的内部实例化 jQuery 对象的方式允许在调用者不显式使用 new 的情况下暴露该原型。

除了设置实例（这就是 jQuery API，如 .each、children、filter 等被暴露的方式），jQuery 内部还会创建一个类似数组的结构来匹配选择器的结果（前提是传入的参数不是 undefined、null 或类似的空值）。如果只有一个元素，这个类似数组的结构将只包含该元素。

一个简单的示例是通过 id 查找元素，然后访问 jQuery 对象以返回底层的 DOM 元素（当匹配或存在多个元素时也适用）。

```
var $div = $("#myDiv");// 使用 id 选择器的结果填充 jQuery 对象
var div = $div[0];// 访问 jQuery 对象的类似数组结构以获取 DOM 元素
```

When using async attribute be conscious as the javascript libraries are then asynchronously loaded and executed as soon as available. If two libraries are included where second library is dependent on the first library is this case if second library is loaded and executed before first library then it may throw an error and application may break.

Section 1.6: The jQuery Object

Every time jQuery is called, by using \$() or jQuery(), internally it is creating a new instance of jQuery. This is the [source code](#) which shows the new instance:

```
// Define a local copy of jQuery
jQuery = function( selector, context ) {

    // The jQuery object is actually just the init constructor 'enhanced'
    // Need init if jQuery is called (just allow error to be thrown if not included)
    return new jQuery.fn.init( selector, context );
}
```

Internally jQuery refers to its prototype as .fn, and the style used here of internally instantiating a jQuery object allows for that prototype to be exposed without the explicit use of new by the caller.

In addition to setting up an instance (which is how the jQuery API, such as .each, children,filter, etc. is exposed), internally jQuery will also create an array-like structure to match the result of the selector (provided that something other than nothing, undefined, null, or similar was passed as the argument). In the case of a single item, this array-like structure will hold only that item.

A simple demonstration would be to find an element with an id, and then access the jQuery object to return the underlying DOM element (this will also work when multiple elements are matched or present).

```
var $div = $("#myDiv");//populate the jQuery object with the result of the id selector
var div = $div[0];//access array-like structure of jQuery object to get the DOM Element
```

第2章：选择器

jQuery 选择器用于在 HTML 文档中选择或查找 DOM（文档对象模型）元素。它基于 id、名称、类型、属性、类等选择 HTML 元素，基于现有的 CSS 选择器。

第2.1节：概述

可以使用 jQuery 选择器 来选择元素。该函数返回单个元素或元素列表。

基本选择器

```
$( "*")           // 所有元素
$( "div")         // 所有 <div> 元素
$( ".blue")       // 所有 class=blue 的元素
$( ".blue.red")   // 所有同时具有 class=blue 和 class=red 的元素
$( ".blue, .red") // 所有具有 class=blue 或 class=red 的元素
$( "#headline")   // id=headline 的（第一个）元素
$( "[href]")      // 所有具有 href 属性的元素
$( "[href='example.com']") // 所有 href=example.com 的元素
```

关系运算符

```
$( "div span")    // 所有 <span>, 它们是 <div> 的后代
$( "div > span")  // 所有 <span>, 它们是 <div> 的直接子元素
$( "a ~ span")    // 所有 <span>, 它们是 <a> 之后的兄弟元素
$( "a + span")    // 所有 <span>, 它们紧跟在 <a> 之后
```

第2.2节：选择器类型

在 jQuery 中，你可以使用元素的多种属性来选择页面中的元素，包括：

- 类型
- 类
- ID
- 属性的拥有
- 属性值
- 索引选择器
- [伪状态](#)

如果你了解CSS选择器，你会注意到jQuery中的选择器是相同的（有少量例外）。

以以下HTML为例：

```
<a href="index.html"></a>           <!-- 1 -->
<a id="second-link"></a>             <!-- 2 -->
<a class="example"></a>              <!-- 3 -->
<a class="example" href="about.html"></a> <!-- 4 -->
<span class="example"></span>        <!-- 5 -->
```

按类型选择：

下面的jQuery选择器将选择所有<a>元素，包括1、2、3和4。

```
$( "a")
```

Chapter 2: Selectors

A jQuery selectors selects or finds a DOM (document object model) element in an HTML document. It is used to select HTML elements based on id, name, types, attributes, class and etc. It is based on existing CSS selectors.

Section 2.1: Overview

Elements can be selected by jQuery using [jQuery Selectors](#). The function returns either an element or a list of elements.

Basic selectors

```
$( "*")           // All elements
$( "div")         // All <div> elements
$( ".blue")       // All elements with class=blue
$( ".blue.red")   // All elements with class=blue AND class=red
$( ".blue, .red") // All elements with class=blue OR class=red
$( "#headline")   // The (first) element with id=headline
$( "[href]")      // All elements with an href attribute
$( "[href='example.com']") // All elements with href=example.com
```

Relational operators

```
$( "div span")    // All <span>s that are descendants of a <div>
$( "div > span")  // All <span>s that are a direct child of a <div>
$( "a ~ span")    // All <span>s that are siblings following an <a>
$( "a + span")    // All <span>s that are immediately after an <a>
```

Section 2.2: Types of Selectors

In jQuery you can select elements in a page using many various properties of the element, including:

- Type
- Class
- ID
- Possession of Attribute
- Attribute Value
- Indexed Selector
- [Pseudo-state](#)

If you know CSS selectors you will notice selectors in jQuery are the same (with minor exceptions).

Take the following HTML for example:

```
<a href="index.html"></a>           <!-- 1 -->
<a id="second-link"></a>             <!-- 2 -->
<a class="example"></a>              <!-- 3 -->
<a class="example" href="about.html"></a> <!-- 4 -->
<span class="example"></span>        <!-- 5 -->
```

Selecting by Type:

The following jQuery selector will select all <a> elements, including 1, 2, 3 and 4.

```
$( "a")
```

按类选择

下面的jQuery选择器将选择所有类为example的元素（包括非a元素），即3、4和5。

```
$(".example")
```

按ID选择

以下jQuery选择器将选择具有给定ID（为2）的元素。

```
$("#second-link")
```

按属性拥有选择

以下jQuery选择器将选择所有定义了 href属性的元素，包括1和4。

```
$("[href]")
```

按属性值选择

以下jQuery选择器将选择所有存在 href属性且值为index.html的元素，即只有1。

```
$("[href='index.html']")
```

按索引位置选择（索引选择器）

以下 jQuery 选择器将只选择第 2 个 <a>，即 第二个-链接，因为提供的索引是 1，类似于 eq(1)（注意索引从 0 开始，因此这里选择了第二个！）。

```
$("#a:eq(1)")
```

使用索引排除进行选择

要通过索引排除某个元素 :not(:eq())

以下选择所有 <a> 元素，除了类名为 example 的那个，它的索引是 1

```
$("#a").not(":eq(0)")
```

使用排除进行选择

要从选择中排除某个元素，使用 :not()

以下选择所有 <a> 元素，除了类名为 example 的那些，它们的索引是 1 和 2。

```
$("#a:not(.example)")
```

通过伪状态选择

你也可以使用伪状态在 jQuery 中进行选择，包括 :first-child、:last-child、:first-of-type、:last-of-type 等。

以下 jQuery 选择器只会选择第一个 <a> 元素：编号 1。

Selecting by Class

The following jQuery selector will select all elements of class example (including non-a elements), which are 3, 4 and 5.

```
$(".example")
```

Selecting by ID

The following jQuery selector will select the element with the given ID, which is 2.

```
$("#second-link")
```

Selecting by Possession of Attribute

The following jQuery selector will select all elements with a defined href attribute, including 1 and 4.

```
$("[href]")
```

Selecting by Attribute Value

The following jQuery selector will select all elements where the href attribute exists with a value of index.html, which is just 1.

```
$("[href='index.html']")
```

Selecting by Indexed Position (Indexed Selector)

The following jQuery selector will select only 1, the second <a> ie. the second-link because index supplied is 1 like eq(1) (Note that the index starts at 0 hence the second got selected here!).

```
$("#a:eq(1)")
```

Selecting with Indexed Exclusion

To exclude an element by using its index :not(:eq())

The following selects <a> elements, except that with the class example, which is 1

```
$("#a").not(":eq(0)")
```

Selecting with Exclusion

To exclude an element from a selection, use :not()

The following selects <a> elements, except those with the class example, which are 1 and 2.

```
$("#a:not(.example)")
```

Selecting by Pseudo-state

You can also select in jQuery using pseudo-states, including :first-child, :last-child, :first-of-type, :last-of-type, etc.

The following jQuery selector will only select the first <a> element: number 1.


```
$("a:first-of-type")
```

组合 jQuery 选择器

你也可以通过组合多个 jQuery 选择器来增加你的特异性；你可以组合任意数量的选择器，或者组合所有选择器。你还可以同时选择多个类、属性和状态。

```
$("a.class1.class2.class3#someID[attr1][attr2='something'][attr3='something']:first-of-type:first-child")
```

这将选择一个 <a> 元素，该元素：

- 具有以下类：class1、class2 和 class3
- 具有以下 ID：someID
- 具有以下属性：attr1
- 具有以下属性及其值：attr2 值为 something, attr3 值为 something
- 具有以下状态：first-child 和 first-of-type

你也可以用逗号分隔不同的选择器：

```
$("a, .class1, #someID")
```

这将选择：

- 所有<a>元素
- 所有具有类class1
- 具有ID#someID的元素

子元素和兄弟元素选择

jQuery选择器通常遵循与CSS相同的约定，这使你可以以相同的方式选择子元素和兄弟元素。

- 选择非直接子元素时，使用空格
- 选择直接子元素时，使用>
- 选择紧邻第一个元素的相邻兄弟元素时，使用+
- 选择非紧邻第一个元素的兄弟元素时，使用~

通配符选择

有时我们想选择所有元素，但没有共同的属性可供选择（类、属性等）。在这种情况下，我们可以使用*选择器，它会简单地选择所有元素：

```
$('#wrapper *') // 选择#wrapper元素内的所有元素
```

第2.3节：缓存选择器

每次在 jQuery 中使用选择器时，都会在 DOM 中搜索匹配查询的元素。过于频繁或重复地执行此操作会降低性能。如果你多次引用同一个选择器，应该将其缓存到变量中：

```
var nav = $('#navigation');
nav.show();
```

```
$("a:first-of-type")
```

Combining jQuery selectors

You can also increase your specificity by combining multiple jQuery selectors; you can combine any number of them or combine all of them. You can also select multiple classes, attributes and states at the same time.

```
$("a.class1.class2.class3#someID[attr1][attr2='something'][attr3='something']:first-of-type:first-child")
```

This would select an <a> element that:

- Has the following classes: class1, class2, and class3
- Has the following ID: someID
- Has the following Attribute: attr1
- Has the following Attributes and values: attr2 with value something, attr3 with value something
- Has the following states: first-child and first-of-type

You can also separate different selectors with a comma:

```
$("a, .class1, #someID")
```

This would select:

- All <a> elements
- All elements that have the class class1
- An element with the id #someID

Child and Sibling selection

jQuery selectors generally conform to the same conventions as CSS, which allows you to select children and siblings in the same way.

- To select a non-direct child, use a space
- To select a direct child, use a >
- To select an adjacent sibling following the first, use a +
- To select a non-adjacent sibling following the first, use a ~

Wildcard selection

There might be cases when we want to select all elements but there is not a common property to select upon (class, attribute etc). In that case we can use the * selector that simply selects all the elements:

```
$('#wrapper *') // Select all elements inside #wrapper element
```

Section 2.3: Caching Selectors

Each time you use a selector in jQuery the DOM is searched for elements that match your query. Doing this too often or repeatedly will decrease performance. If you refer to a specific selector more than once you should add it to the cache by assigning it to a variable:

```
var nav = $('#navigation');
nav.show();
```

这将替代：

```
$('#navigation').show();
```

如果你的网站需要频繁显示/隐藏该元素，缓存此选择器会很有帮助。如果有多个元素使用相同的选择器，变量将成为这些元素的数组：

```
<div class="parent">
  <div class="child">子元素 1</div>
  <div class="child">子元素 2</div>
</div>

<script>
var children = $('.child');
var firstChildText = children[0].text();
console.log(firstChildText);

// 输出: "子元素 1"
</script>
```

注意：该元素必须在赋值给变量时存在于DOM中。如果DOM中没有名为child的类元素，则该变量中将存储一个空数组。

```
<div class="parent"></div>

<script>
var parent = $('.parent');
var children = $('.child');
console.log(children);

// 输出：[]

parent.append('<div class="child">Child 1</div>');
children = $('.child');
console.log(children[0].text());

// 输出："Child 1"
</script>
```

记得在DOM中添加或移除该选择器对应的元素后，重新将选择器赋值给变量。

注意：在缓存选择器时，许多开发者会在变量名前加上\$，以表示该变量是一个jQuery对象，如下所示：

```
var $nav = $('#navigation');
$nav.show();
```

第2.4节：组合选择器

考虑以下DOM结构

```
<ul class="parentUI">
  <li> 一级
    <ul class="childUI">
      <li>一级-1 <span> 项目 - 1 </span></li>
      <li>一级-1 <span> 项目 - 2 </span></li>
    </ul>
  </li>
```

This would replace:

```
$('#navigation').show();
```

Caching this selector could prove helpful if your website needs to show/hide this element often. If there are multiple elements with the same selector the variable will become an array of these elements:

```
<div class="parent">
  <div class="child">Child 1</div>
  <div class="child">Child 2</div>
</div>

<script>
var children = $('.child');
var firstChildText = children[0].text();
console.log(firstChildText);

// output: "Child 1"
</script>
```

NOTE: The element has to exist in the DOM at the time of its assignment to a variable. If there is no element in the DOM with a class called child you will be storing an empty array in that variable.

```
<div class="parent"></div>

<script>
var parent = $('.parent');
var children = $('.child');
console.log(children);

// output: []

parent.append('<div class="child">Child 1</div>');
children = $('.child');
console.log(children[0].text());

// output: "Child 1"
</script>
```

Remember to reassign the selector to the variable after adding/removing elements in the DOM with that selector.

Note: When caching selectors, many developers will start the variable name with a \$ to denote that the variable is a jQuery object like so:

```
var $nav = $('#navigation');
$nav.show();
```

Section 2.4: Combining selectors

Consider following DOM Structure

```
<ul class="parentUI">
  <li> Level 1
    <ul class="childUI">
      <li>Level 1-1 <span> Item - 1 </span></li>
      <li>Level 1-1 <span> Item - 2 </span></li>
    </ul>
  </li>
```

```
<li> 二级
  <ul class="childUI">
    <li>二级-1 <span> 项目 - 1 </span></li>
    <li>二级-1 <span> 项目 - 1 </span></li>
  </ul>
</li>
</ul>
```

后代选择器和子选择器

给定一个父元素 - parentUI，查找其后代元素 () ，

1. 简单的 \$('parent child')

```
>> $('ul.parentUI li')
```

这会获取指定祖先的所有匹配后代元素，所有层级。

2. > - \$('parent > child')

```
>> $('ul.parentUI > li')
```

这会查找所有匹配的子元素（仅第一级）。

3. 基于上下文的选择器 - \$('child','parent')

```
>> $('li','ul.parentUI')
```

这与上面第1条的作用相同。

4. find() - \$('parent').find('child')

```
>> $('ul.parentUI').find('li')
```

这与上面第1条的作用相同。

5. children() - \$('parent').find('child')

```
>> $('ul.parentUI').children('li')
```

这与上面第2条的作用相同。

其他组合选择器

组选择器 ： ","

选择所有元素和所有元素以及所有元素：

```
$('ul, li, span')
```

多重选择器 ： ""（无字符）

```
<li> Level 2
  <ul class="childUI">
    <li>Level 2-1 <span> Item - 1 </span></li>
    <li>Level 2-1 <span> Item - 1 </span></li>
  </ul>
</li>
</ul>
```

Descendant and child selectors

Given a parent - parentUI find its descendants (),

1. Simple \$('parent child')

```
>> $('ul.parentUI li')
```

This gets all matching descendants of the specified ancestor *all levels down*.

2. > - \$('parent > child')

```
>> $('ul.parentUI > li')
```

This finds all matching children (*only 1st level down*).

3. Context based selector - \$('child','parent')

```
>> $('li','ul.parentUI')
```

This works same as 1. above.

4. find() - \$('parent').find('child')

```
>> $('ul.parentUI').find('li')
```

This works same as 1. above.

5. children() - \$('parent').find('child')

```
>> $('ul.parentUI').children('li')
```

This works same as 2. above.

Other combinators

Group Selector : ","

Select all elements AND all elements AND all elements :

```
$('ul, li, span')
```

Multiples selector : "" (no character)

选择所有带有类名parentUI的元素：

```
$('.parentUI')
```

相邻兄弟选择器：

选择所有紧接在另一个元素后面的元素：

```
$( 'li + li' )
```

通用兄弟选择器：“~”

选择所有作为其他元素兄弟的元素：

```
$( 'li ~ li' )
```

第2.5节：作为选择器的DOM元素

jQuery接受多种参数，其中之一是实际的DOM元素。将DOM元素传递给jQuery会使jQuery对象的底层数组结构包含该元素。

jQuery通过检查其nodeType来检测参数是否为DOM元素。

DOM元素最常见的用法是在回调中，将当前元素传递给jQuery构造函数，以便访问jQuery API。

例如在teach回调中（注意：每个都是迭代器函数）。

```
$(".elements").each(function(){
    //当前元素在使用each时由jQuery内部绑定到`this`
    var currentElement = this;

    //此时，currentElement（或this）可以访问原生API

    //用currentElement(this)构造一个jQuery对象
    var $currentElement = $(this);

    //现在$currentElement可以访问jQuery API
});
```

第2.6节：作为选择器的HTML字符串

jQuery接受多种参数作为“选择器”，其中之一是HTML字符串。将HTML字符串传递给jQuery会使jQuery对象的底层类数组结构保存构造出的HTML。

jQuery使用正则表达式判断传入构造函数的字符串是否为HTML字符串，并且必须以<开头。该正则表达式定义为 rquickExpr = `/^(?:\s*(<[\w\W]+>)[^>]*|#[\w-]*)$/`（详见regex101.com）。

HTML字符串作为选择器最常见的用途是在代码中仅创建一组DOM元素，通常被库用于模态弹出等功能。

例如，一个返回用div包裹的锚点标签作为模板的函数

```
function template(href,text){
    return $("<div><a href='" + href + "'>"+ text + "</a></div>");
}
```

Select all elements with class parentUI：

```
$( 'ul.parentUI' )
```

Adjacent Sibling Selector：“+”

Select all elements that are placed immediately after another element:

```
$( 'li + li' )
```

General Sibling Selector：“~”

Select all elements that are siblings of other elements:

```
$( 'li ~ li' )
```

Section 2.5: DOM Elements as selectors

jQuery accepts a wide variety of parameters, and one of them is an actual DOM element. Passing a DOM element to jQuery will cause the underlying array-like structure of the jQuery object to hold that element.

jQuery will detect that the argument is a DOM element by inspecting its nodeType.

The most common use of a DOM element is in callbacks, where the current element is passed to the jQuery constructor in order to gain access to the jQuery API.

Such as in the each callback (note: each is an iterator function).

```
$(".elements").each(function(){
    //the current element is bound to `this` internally by jQuery when using each
    var currentElement = this;

    //at this point, currentElement (or this) has access to the Native API

    //construct a jQuery object with the currentElement(this)
    var $currentElement = $(this);

    //now $currentElement has access to the jQuery API
});
```

Section 2.6: HTML strings as selectors

jQuery accepts a wide variety of parameters as "selectors", and one of them is an HTML string. Passing an HTML string to jQuery will cause the underlying array-like structure of the jQuery object to hold the resulting constructed HTML.

jQuery uses regex to determine if the string being passed to the constructor is an HTMLstring, and also that it *must* start with <. That regex is defined as rquickExpr = `/^(?:\s*(<[\w\W]+>)[^>]*|#[\w-]*)$/` ([explanation at regex101.com](#)).

The most common use of an HTML string as a selector is when sets of DOM elements need to be created in code only, often this is used by libraries for things like Modal popouts.

For example, a function which returned an anchor tag wrapped in a div as a template

```
function template(href,text){
    return $("<div><a href='" + href + "'>"+ text + "</a></div>");
}
```



```
}
```

将返回一个包含以下内容的jQuery对象

```
<div>
  <a href="google.com">Google</a>
</div>
```

如果调用为 `template("google.com","Google")`。

belindoc.com

```
}
```

Would return a jQuery object holding

```
<div>
  <a href="google.com">Google</a>
</div>
```

if called as `template("google.com", "Google")`.

第3章：每个函数

第3.1节：jQuery each函数

HTML：

```
<ul>
  <li>芒果</li><li>
    书</li>
```

脚本：

```
$( "li" ).each(function( index ) {
  console.log( index + ": " + $( this ).text() );
});
```

因此，每个列表项都会记录一条消息：

0: 芒果

1: 书本

Chapter 3: Each function

Section 3.1: jQuery each function

HTML:

```
<ul>
  <li>Mango</li>
  <li>Book</li>
</ul>
```

Script:

```
$( "li" ).each(function( index ) {
  console.log( index + ": " + $( this ).text() );
});
```

A message is thus logged for each item in the list:

0: Mango

1: Book

第4章：属性

第4.1节：attr() 和 prop() 之间的区别

attr() 使用 DOM 函数 `getAttribute()` 和 `setAttribute()` 来获取/设置 HTML 属性。 `prop()` 通过设置 [DOM 属性](#) 而不改变属性值来工作。在许多情况下，两者可以互换，但有时需要使用其中一个。

设置复选框为选中状态：

```
$('#tosAccept').prop('checked', true); // 这里使用 attr() 无法正常工作
```

要移除属性，可以使用 `removeProp()` 方法。同样，`removeAttr()` 用于移除属性。

第4.2节：获取HTML元素的属性值

当向.attr()函数传递单个参数时，它会返回所选元素上该属性的值。

语法：

```
$([选择器]).attr([属性名]);
```

示例：

HTML：

```
<a href="/home">首页</a>
```

jQuery:

```
$('a').attr('href');
```

获取data属性：

jQuery提供了 [.data\(\)](#) 函数来处理data属性。 `.data` 函数返回所选元素上data属性的值。

语法：

```
$([选择器]).data([属性名]);
```

示例：

Html:

```
<article data-column="3"></article>
```

jQuery:

```
$("article").data("column")
```

注意：

jQuery 的 `data()` 方法可以让你访问 `data-*` 属性，但它会改变属性名的大小写。参考

Chapter 4: Attributes

Section 4.1: Differece between attr() and prop()

`attr()` gets/sets the HTML attribute using the DOM functions `getAttribute()` and `setAttribute()`. [.prop\(\)](#) works by setting the DOM property without changing the attribute. In many cases the two are interchangeable, but occasionally one is needed over the other.

To set a checkbox as checked:

```
$('#tosAccept').prop('checked', true); // using attr() won't work properly here
```

To remove a property you can use the [.removeProp\(\)](#) method. Similarly `removeAttr()` removes attributes.

Section 4.2: Get the attribute value of a HTML element

When a single parameter is passed to the [.attr\(\)](#) function it returns the value of passed attribute on the selected element.

Syntax:

```
$([selector]).attr([attribute name]);
```

Example:

HTML:

```
<a href="/home">Home</a>
```

jQuery:

```
$('a').attr('href');
```

Fetching data attributes:

jQuery offers [.data\(\)](#) function in order to deal with data attributes. `.data` function returns the value of the data attribute on the selected element.

Syntax:

```
$([selector]).data([attribute name]);
```

Example:

Html:

```
<article data-column="3"></article>
```

jQuery:

```
$("article").data("column")
```

Note:

jQuery's `data()` method will give you access to `data-*` attributes, BUT, it clobbers the case of the attribute name. [Reference](#)

第4.3节：设置HTML属性的值

如果你想给某个元素添加属性，可以使用 `attr(attributeName, attributeValue)` 函数。
例如：

```
$( 'a' ).attr( 'title', '点击我' );
```

这个例子会给页面上所有链接添加鼠标悬停文本 "点击我"。

同一个函数也用于修改属性的值。

第4.4节：移除属性

要从元素中移除属性，可以使用函数 `removeAttr(attributeName)`。例如：

```
$( '#home' ).removeAttr( 'title' );
```

这将从ID为home的元素中移除title属性。

Section 4.3: Setting value of HTML attribute

If you want to add an attribute to some element you can use the `attr(attributeName, attributeValue)` function. For example:

```
$( 'a' ).attr( 'title', 'Click me' );
```

This example will add mouseover text "Click me" to all links on the page.

The same function is used to change attributes' values.

Section 4.4: Removing attribute

To remove an attribute from an element you can use the function `removeAttr(attributeName)`. For example:

```
$( '#home' ).removeAttr( 'title' );
```

This will remove title attribute from the element with ID home.

第5章：document-ready事件

第5.1节：什么是document-ready以及如何使用？

jQuery代码通常包裹在jQuery(function(\$){ ... });中，以确保只有在DOM加载完成后才运行。

```
<script type="text/javascript">
  jQuery(function($) {
    // 这将把div的文本设置为“Hello”。
    $("#myDiv").text("Hello");
  });
</script>

<div id="myDiv">文本</div>
```

这是重要的，因为 jQuery（以及一般的 JavaScript）无法选择尚未渲染到页面上的 DOM 元素。

```
<script type="text/javascript">
// 此时不存在 id 为 "myDiv" 的元素，所以 $("#myDiv") 是一个
// 空选择，这不会产生任何效果
$("#myDiv").text("Hello");
</script>

<div id="myDiv">文本</div>
```

注意，你可以通过向 .ready() 方法传入自定义处理函数来为 jQuery 命名空间设置别名。这在另一个 JS 库使用与 jQuery 相同的简写 \$ 别名时非常有用，这会导致冲突。为避免此冲突，你必须调用 \$.noConflict(); —— 这会强制你仅使用默认的 jQuery 命名空间（而非简写的 \$ 别名）。

通过向 .ready() 处理函数传入自定义处理函数，你可以选择使用的 jQuery 别名。

```
$.noConflict();

jQuery( document ).ready(function( $ ) {
  // 在这里我们可以使用 '$' 作为 jQuery 的别名，而不会与其他
  // 使用相同命名空间的库冲突
  $('body').append('<div>Hello</div>')
});

jQuery( document ).ready(function( jq ) {
  // 这里我们使用自定义的 jQuery 别名 'jq'
  jq('body').append('<div>Hello</div>')
});
```

与其简单地将你的 jQuery 代码放在页面底部，使用 \$(document).ready 函数可以确保所有 HTML 元素都已渲染完毕，整个文档对象模型（DOM）已准备好供 JavaScript 代码执行。

第 5.2 节：jQuery 2.2.3 及更早版本

以下写法都等效，代码块内的代码将在文档准备好时运行：

```
$(function() {
  // 代码
```

Chapter 5: document-ready event

Section 5.1: What is document-ready and how should I use it?

jQuery code is often wrapped in jQuery(function(\$){ ... }); so that it only runs after the DOM has finished loading.

```
<script type="text/javascript">
  jQuery(function($) {
    // this will set the div's text to "Hello".
    $("#myDiv").text("Hello");
  });
</script>

<div id="myDiv">Text</div>
```

This is important because jQuery (and JavaScript generally) cannot select a DOM element that has not been rendered to the page.

```
<script type="text/javascript">
// no element with id="myDiv" exists at this point, so $("#myDiv") is an
// empty selection, and this will have no effect
$("#myDiv").text("Hello");
</script>

<div id="myDiv">Text</div>
```

Note that you can alias the jQuery namespace by passing a custom handler into the .ready() method. This is useful for cases when another JS library is using the same shortened \$ alias as jQuery, which create a conflict. To avoid this conflict, you must call \$.noConflict(); - This forcing you to use only the default jQuery namespace (Instead of the short \$ alias).

By passing a custom handler to the .ready() handler, you will be able to choose the alias name to use jQuery.

```
$.noConflict();

jQuery( document ).ready(function( $ ) {
  // Here we can use '$' as jQuery alias without it conflicting with other
  // libraries that use the same namespace
  $('body').append('<div>Hello</div>')
});

jQuery( document ).ready(function( jq ) {
  // Here we use a custom jQuery alias 'jq'
  jq('body').append('<div>Hello</div>')
});
```

Rather than simply putting your jQuery code at the bottom of the page, using the \$(document).ready function ensures that all HTML elements have been rendered and the entire Document Object Model (DOM) is ready for JavaScript code to execute.

Section 5.2: jQuery 2.2.3 and earlier

These are all equivalent, the code inside the blocks will run when the document is ready:

```
$(function() {
  // code
```

```
});

$.ready(function() {
    // 代码
});

$(document).ready(function() {
    // 代码
});
```

因为这些写法等效，推荐使用第一种形式，下面是使用 jQuery 关键字替代 \$ 的版本，效果相同：

```
jQuery(function() {
    // 代码
});
```

第 5.3 节：jQuery 3.0

符号说明

从 jQuery 3.0 开始，推荐仅使用这种形式：

```
jQuery(function($) {
    // 文档准备好时运行
    // $ (第一个参数) 将是对 jQuery 的内部引用
    // 永远不要依赖 $ 是全局命名空间中对 jQuery 的引用
});
```

jQuery 3.0 中所有其他的文档就绪处理程序均已弃用。

异步

从 jQuery 3.0 开始，ready 处理程序将始终异步调用。这意味着在下面的代码中，无论文档在执行时是否已准备好，日志“outside handler”总是会先显示。

```
$(function() {
    console.log("inside handler");
});
console.log("outside handler");
```

- > 外部处理程序
- > 内部处理程序

第5.4节：附加事件和操作DOM在ready()内部

示例用法 \$(document).ready():

1.附加事件处理程序

附加jQuery事件处理程序

```
$(document).ready(function() {
    $("button").click(function() {
```

```
});

$.ready(function() {
    // code
});

$(document).ready(function() {
    // code
});
```

Because these are equivalent the first is the recommended form, the following is a version of that with the jQuery keyword instead of the \$ which produce the same results:

```
jQuery(function() {
    // code
});
```

Section 5.3: jQuery 3.0

Notation

As of jQuery 3.0, only this form is recommended:

```
jQuery(function($) {
    // Run when document is ready
    // $ (first argument) will be internal reference to jQuery
    // Never rely on $ being a reference to jQuery in the global namespace
});
```

All other document-ready handlers [are deprecated in jQuery 3.0](#).

Asynchronous

As of jQuery 3.0, the ready handler [will always be called asynchronously](#). This means that in the code below, the log 'outside handler' will always be displayed first, regardless whether the document was ready at the point of execution.

```
$(function() {
    console.log("inside handler");
});
console.log("outside handler");
```

- > outside handler
- > inside handler

Section 5.4: Attaching events and manipulating the DOM inside ready()

Example uses of \$(document).ready():

1. Attaching event handlers

Attach jQuery event handlers

```
$(document).ready(function() {
    $("button").click(function() {
```

```
// 点击函数的代码
});
});
```

2. 页面结构创建后运行jQuery代码

```
jQuery(function($) {
// 设置元素的值。
$("#myElement").val("Hello");
});
```

3. 操作已加载的DOM结构

例如：在页面首次加载时隐藏一个div，并在按钮的点击事件中显示它

```
$(document).ready(function() {
    $("#toggleDiv").hide();
    $("button").click(function() {
        $("#toggleDiv").show();
    });
});
```

第5.5节：\$(document).ready()和\$(window).load()的区别

\$(window).load()在jQuery 1.8版本中被弃用（并在jQuery 3.0中完全移除），因此不应再使用。弃用的原因已在jQuery关于此事件的页面中说明

使用load事件时的注意事项（针对图片）

开发者常用.load()快捷方式来解决在图片（或图片集合）完全加载后执行函数的问题。但这存在几个已知的注意事项，应予以关注。这些包括：

- 它在不同浏览器间表现不一致且不可靠
- 如果图片的 src设置为之前相同的 src，WebKit中不会正确触发
- 它不会正确地向上冒泡到DOM树
- 对于已经存在于浏览器缓存中的图片，可能不会触发事件

如果你仍然希望使用load()，相关文档如下：

\$(document).ready() 会等待直到完整的 DOM 可用——HTML 中的所有元素都已被解析并且在文档中。然而，此时诸如图片等资源可能尚未完全加载。如果需要等待所有资源加载完成，\$(window).load() 且你了解该事件的显著限制，则可以使用以下代码：

```
$(document).ready(function() {
    console.log($("#my_large_image").height()); // 可能为0，因为图片尚不可用
});

$(window).load(function() {
    console.log($("#my_large_image").height()); // 将是正确的
});
```

```
// Code for the click function
});
});
```

2. Run jQuery code after the page structure is created

```
jQuery(function($) {
// set the value of an element.
$("#myElement").val("Hello");
});
```

3. Manipulate the loaded DOM structure

For example: hide a div when the page loads for the first time and show it on the click event of a button

```
$(document).ready(function() {
    $("#toggleDiv").hide();
    $("button").click(function() {
        $("#toggleDiv").show();
    });
});
```

Section 5.5: Difference between \$(document).ready() and \$(window).load()

\$(window).load() was **deprecated in jQuery version 1.8 (and completely removed from jQuery 3.0)** and as such should not be used anymore. The reasons for the deprecation are noted on the [jQuery page about this event](#)

Caveats of the load event when used with images

A common challenge developers attempt to solve using the .load() shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:

- It doesn't work consistently nor reliably cross-browser
- It doesn't fire correctly in WebKit if the image src is set to the same src as before
- It doesn't correctly bubble up the DOM tree
- Can cease to fire for images that already live in the browser's cache

If you still wish to use load() it is documented below:

\$(document).ready() waits until the full DOM is availble -- all the elements in the HTML have been parsed and are in the document. However, resources such as images may not have fully loaded at this point. If it is important to wait until all resources are loaded, \$(window).load() **and you're aware of the significant limitations of this event** then the below can be used instead:

```
$(document).ready(function() {
    console.log($("#my_large_image").height()); // may be 0 because the image isn't available
});

$(window).load(function() {
    console.log($("#my_large_image").height()); // will be correct
});
```

第5.6节：jQuery(fn) 与在 </body> 之前执行代码的区别

使用 document-ready 事件可能会有小的性能缺陷，执行可能延迟约300毫秒。
有时可以通过在关闭的</body>标签之前执行代码来实现相同的行为：

```
<body>
  <span id="greeting"></span> world!
  <script>
    $("#greeting").text("Hello");
  </script>
</body>
```

将产生类似的行为，但执行得更早，因为它不像以下情况那样等待文档就绪事件触发：

```
<head>
  <script>
    jQuery(function($) {
      $("#greeting").text("Hello");
    });
  </script>
</head>
<body>
  <span id="greeting"></span> world!
</body>
```

强调第一个示例依赖于你对页面的了解以及脚本放置在关闭的</body>标签之前，且特别是在span标签之后这一事实。

Section 5.6: Difference between jQuery(fn) and executing your code before </body>

Using the document-ready event can have small [performance drawbacks](#), with delayed execution of up to ~300ms. Sometimes the same behavior can be achieved by execution of code just before the closing </body> tag:

```
<body>
  <span id="greeting"></span> world!
  <script>
    $("#greeting").text("Hello");
  </script>
</body>
```

will produce similar behavior but perform sooner than as it does not wait for the document ready event trigger as it does in:

```
<head>
  <script>
    jQuery(function($) {
      $("#greeting").text("Hello");
    });
  </script>
</head>
<body>
  <span id="greeting"></span> world!
</body>
```

Emphasis on the fact that first example relies upon your knowledge of your page and placement of the script just prior to the closing </body> tag and specifically after the span tag.

第6章：事件

第6.1节：委托事件

让我们从示例开始。这是一个非常简单的HTML示例。

示例HTML

```
<html>
  <head>
  </head>
  <body>
    <ul>
      <li>
        <a href="some_url/">链接 1</a>
      </li>
      <li>
        <a href="some_url/">链接 2</a>
      </li>
      <li>
        <a href="some_url/">链接 3</a>
      </li>
    </ul>
  </body>
</html>
```

问题

现在在这个例子中，我们想给所有<a>元素添加事件监听器。问题是这个例子中的列表是动态的。元素会随着时间的推移被添加和移除。然而，页面在变化之间不会刷新，这使得我们无法使用简单的点击事件监听器来绑定链接对象（即\$('a').click()）。

我们面临的问题是如何给不断出现和消失的<a>元素添加事件。

背景信息 - 事件传播

委托事件之所以可能，是因为事件传播（通常称为事件冒泡）。每当事件被触发时，它会一直冒泡到文档根节点。它们将事件的处理委托给一个不变的祖先元素，因此称为“委托”事件。

所以在上面的例子中，点击<a>元素链接会按以下顺序触发这些元素中的“click”事件：

- a
- 李
- ul
- body
- html
- 文档根元素

解决方案

了解事件冒泡的作用后，我们可以捕获在我们的

Chapter 6: Events

Section 6.1: Delegated Events

Let's start with example. Here is a very simple example HTML.

Example HTML

```
<html>
  <head>
  </head>
  <body>
    <ul>
      <li>
        <a href="some_url/">Link 1</a>
      </li>
      <li>
        <a href="some_url/">Link 2</a>
      </li>
      <li>
        <a href="some_url/">Link 3</a>
      </li>
    </ul>
  </body>
</html>
```

The problem

Now in this example, we want to add an event listener to all <a> elements. The problem is that the list in this example is dynamic. elements are added and removed as time passes by. However, the page does not refresh between changes, which would allow us to use simple click event listeners to the link objects (i.e. \$('a').click()).

The problem we have is how to add events to the <a> elements that come and go.

Background information - Event propagation

Delegated events are only possible because of event propagation (often called event bubbling). Any time an event is fired, it will bubble all the way up (to the document root). They *delegate* the handling of an event to a non-changing ancestor element, hence the name "delegated" events.

So in example above, clicking <a> element link will trigger 'click' event in these elements in this order:

- a
- li
- ul
- body
- html
- document root

Solution

Knowing what event bubbling does, we can catch one of the wanted events which are propagating up through our HTML.

在此示例中，捕获事件的一个好位置是元素，因为该元素不是动态的：

```
$( 'ul' ).on( 'click', 'a', function () {
  console.log( this.href ); // jQuery将事件函数绑定到目标DOM元素
                             // 这样`this`指向锚点而不是列表
                             // 当链接被点击时执行你想做的任何操作
});
```

上述代码中：

- 我们有一个 'ul'，它是该事件监听器的接收者
- 第一个参数（'click'）定义了我们试图检测的事件类型。
- 第二个参数（'a'）用于声明事件需要从哪里发起（在该事件监听器接收者 ul 的所有子元素中）。
- 最后，第三个参数是在满足第一个和第二个参数要求时执行的代码。

解决方案的详细工作原理

1. 用户点击<a>元素
2. 这会触发<a>元素上的点击事件。
- 3.事件开始向文档根节点冒泡。
4. 事件先冒泡到元素，然后冒泡到元素。
5. 事件监听器被触发，因为元素上绑定了该事件监听器。
- 6.事件监听器首先检测触发的事件。冒泡事件是 'click'，监听器监听的是 'click'，符合条件。
7. 监听器检查尝试将第二个参数（'a'）与气泡链中的每个项目匹配。由于链中的最后一个项目是 'a'，这符合过滤条件，因此也通过了。
8. 第三个参数中的代码使用匹配项作为它的this来运行。如果函数中不包含调用调用stopPropagation()，事件将继续向上传播到根节点（document）。

注意：如果没有合适且不变的祖先元素可用或方便，您应使用document。作为习惯，以下原因不建议使用'body'：

- body存在一个与样式相关的错误，可能导致鼠标事件无法冒泡到它。这取决于浏览器，并且当计算出的body高度为0时（例如，当所有子元素都具有绝对定位时）可能会发生。鼠标事件总是冒泡到document。
- document *always* 存在于你的脚本中，因此你可以在 document 上附加委托处理程序，即使是在 *DOM-ready handler* 之外，也能确保它们仍然有效。

第6.2节：附加和分离事件处理程序

附加事件处理程序

自版本1.7起，jQuery 拥有事件 API.on()。通过这种方式，任何标准的 JavaScript 事件或自定义事件都可以绑定到当前选中的 jQuery 元素上。虽然有诸如.click()的快捷方式，但.on()提供了更多选项。

HTML

```
<button id="foo">bar</button>
```

jQuery

```
$( "#foo" ).on( "click", function() {
  console.log( $( this ).text() ); //bar
```

A good place for catching it in this example is the element, as that element does is not dynamic:

```
$( 'ul' ).on( 'click', 'a', function () {
  console.log( this.href ); // jQuery binds the event function to the targeted DOM element
                             // this way `this` refers to the anchor and not to the list
                             // Whatever you want to do when link is clicked
});
```

In above:

- We have 'ul' which is the recipient of this event listener
- The first parameter ('click') defines which events we are trying to detect.
- The second parameter ('a') is used to declare where the event needs to *originate* from (of all child elements under this event listener's recipient, ul).
- Lastly, the third parameter is the code that is run if first and second parameters' requirements are fulfilled.

In detail how solution works

1. User clicks <a> element
2. That triggers click event on <a> element.
3. The event start bubbling up towards document root.
4. The event bubbles first to the element and then to the element.
5. The event listener is run as the element has the event listener attached.
6. The event listener first detects the triggering event. The bubbling event is 'click' and the listener has 'click', it is a pass.
7. The listener checks tries to match the second parameter ('a') to each item in the bubble chain. As the last item in the chain is an 'a' this matches the filter and this is a pass too.
8. The code in third parameter is run using the matched item as it's **this**. If the function does not include a call to stopPropagation(), the event will continue propagating upwards towards the root (document).

Note: If a suitable non-changing ancestor is not available/convenient, you should use document. As a habit do not use 'body' for the following reasons:

- body has a bug, to do with styling, that can mean mouse events do not bubble to it. This is browser dependant and can happen when the calculated body height is 0 (e.g. when all child elements have absolute positions). Mouse events always bubble to document.
- document *always* exists to your script, so you can attach delegated handlers to document outside of a *DOM-ready handler* and be certain they will still work.

Section 6.2: Attach and Detach Event Handlers

Attach an Event Handler

Since version 1.7 jQuery has the event API .on(). This way any standard javascript event or custom event can be bound on the currently selected jQuery element. There are shortcuts such as .click(), but .on() gives you more options.

HTML

```
<button id="foo">bar</button>
```

jQuery

```
$( "#foo" ).on( "click", function() {
  console.log( $( this ).text() ); //bar
```

```
});
```

移除事件处理程序

当然，你也可以从你的 jQuery 对象中移除事件。你可以使用 `.off(events [, selector] [, handler])` 来实现。

HTML

```
<button id="hello">hello</button>
```

jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off();
});
```

点击按钮时，`$(this)` 将指向当前的 jQuery 对象，并移除其上所有绑定的事件处理程序。你也可以指定要移除的具体事件处理程序。

jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off('click');
});

$('#hello').on('mouseenter', function(){
    console.log('you are about to click');
});
```

在这种情况下，mouseenter 事件在点击后仍然会生效。

第6.3节：通过jQuery开关特定事件。（命名监听器）

有时你想关闭所有之前注册的监听器。

```
// 添加一个普通的点击处理器
$(document).on("click",function(){
    console.log("文档被点击 1")
});
// 添加另一个点击处理器
$(document).on("click",function(){
    console.log("文档被点击 2")
});
// 移除所有注册的处理器。
$(document).off("click")
```

此方法的问题是，所有由其他插件等绑定在 document 上的监听器也会被移除。

我们通常希望只移除由我们绑定的所有监听器。

为此，我们可以绑定命名的监听器，如下所示，

```
// 添加命名事件监听器。
```

```
});
```

Detach an Event Handler

Naturally you have the possibility to detach events from your jQuery objects too. You do so by using `.off(events [, selector] [, handler])`.

HTML

```
<button id="hello">hello</button>
```

jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off();
});
```

When clicking the button `$(this)` will refer to the current jQuery object and will remove all attached event handlers from it. You can also specify which event handler should be removed.

jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off('click');
});

$('#hello').on('mouseenter', function(){
    console.log('you are about to click');
});
```

In this case the `mouseenter` event will still function after clicking.

Section 6.3: Switching specific events on and off via jQuery. (Named Listeners)

Sometimes you want to switch off all previously registered listeners.

```
//Adding a normal click handler
$(document).on("click",function(){
    console.log("Document Clicked 1")
});
//Adding another click handler
$(document).on("click",function(){
    console.log("Document Clicked 2")
});
//Removing all registered handlers.
$(document).off("click")
```

An issue with this method is that ALL listeners binded on document by other plugins etc would also be removed.

More often than not, we want to detach all listeners attached only by us.

To achieve this, we can bind named listeners as,

```
//Add named event listener.
```

```
$(document).on("click.mymodule",function(){
    console.log("文档被点击 1")
});
$(document).on("click.mymodule",function(){
    console.log("文档被点击 2")
});

// 移除命名事件监听器。
$(document).off("click.mymodule");
```

这确保不会无意中修改其他点击监听器。

第6.4节：originalEvent

有时jQuery事件中没有某些属性。要访问底层属性，请使用事件。`originalEvent`

获取滚动方向

```
$(文档).监听("wheel",函数(事件){
    控制台.日志(事件.原始事件.deltaY)
    // 根据滚动方向返回介于 -100 到 100 之间的值
})
```

第6.5节：针对重复元素的事件处理（不使用ID）

问题

页面中有一系列重复元素，你需要知道事件发生在哪个元素上，以便对该特定实例进行操作。

解决方案

- 给所有公共元素赋予一个公共类名
- 对该类应用事件监听器。事件处理函数中的this是事件发生的匹配选择器元素
- 从this开始，遍历到该实例最外层的重复容器
- 在该容器内使用find()来隔离该实例特定的其他元素

HTML

```
<div class="item-wrapper" data-item_id="346">
  <div class="item"><span class="person">弗雷德</span></div>
  <div class="item-toolbar">
    <button class="delete">删除</button>
  </div>
</div>
<div class="item-wrapper" data-item_id="393">
  <div class="item"><span class="person">维尔玛</span></div>
  <div class="item-toolbar">
    <button class="delete">删除</button>
  </div>
</div>
```

jQuery

```
$(function() {
    $('.delete').on('click', function() {
```

```
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 1")
});
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 2")
});

//Remove named event listener.
$(document).off("click.mymodule");
```

This ensures that any other click listener is not inadvertently modified.

Section 6.4: originalEvent

Sometimes there will be properties that aren't available in jQuery event. To access the underlying properties use Event.`originalEvent`

Get Scroll Direction

```
$(document).on("wheel",function(e){
    console.log(e.originalEvent.deltaY)
    // Returns a value between -100 and 100 depending on the direction you are scrolling
})
```

Section 6.5: Events for repeating elements without using ID's

Problem

There is a series of repeating elements in page that you need to know which one an event occurred on to do something with that specific instance.

Solution

- Give all common elements a common class
- Apply event listener to a class. **this** inside event handler is the matching selector element the event occurred on
- Traverse to outer most repeating container for that instance by starting at **this**
- Use `find()` within that container to isolate other elements specific to that instance

HTML

```
<div class="item-wrapper" data-item_id="346">
  <div class="item"><span class="person">Fred</span></div>
  <div class="item-toolbar">
    <button class="delete">Delete</button>
  </div>
</div>
<div class="item-wrapper" data-item_id="393">
  <div class="item"><span class="person">Wilma</span></div>
  <div class="item-toolbar">
    <button class="delete">Delete</button>
  </div>
</div>
```

jQuery

```
$(function() {
    $('.delete').on('click', function() {
```

```
// "this" 是事件发生的元素
var $btn = $(this);
// 遍历到包装容器
var $itemWrap = $btn.closest('.item-wrapper');
// 在包装容器内查找此按钮对应的人员名称
var person = $itemWrap.find('.person').text();
// 发送删除请求到服务器, ajax成功后从页面移除
$.post('url/string', { id: $itemWrap.data('item_id') }).done(function(response) {
    $itemWrap.remove()
}).fail(function() {
    alert("哎呀, 服务器未删除");
});
});
```

第6.6节：文档加载事件 .load()

如果你希望脚本等待某个资源加载完成，比如图片或PDF，可以使用 `.load()`，这是 `.on("load", handler)` 的快捷方式。

HTML

```

```

jQuery

```
$( "#image" ).load(function() {
    // 运行脚本
});
```

```
// "this" is element event occurred on
var $btn = $(this);
// traverse to wrapper container
var $itemWrap = $btn.closest('.item-wrapper');
// look within wrapper to get person for this button instance
var person = $itemWrap.find('.person').text();
// send delete to server and remove from page on success of ajax
$.post('url/string', { id: $itemWrap.data('item_id') }).done(function(response) {
    $itemWrap.remove()
}).fail(function() {
    alert('Ooops, not deleted at server');
});
});
```

Section 6.6: Document Loading Event .load()

If you want your script to wait until a certain resource was loaded, such as an image or a PDF you can use `.load()`, which is a shortcut for shortcut for `.on("load", handler)`.

HTML

```

```

jQuery

```
$( "#image" ).load(function() {
    // run script
});
```


第7章：DOM操作

第7.1节：创建DOM元素

jQuery函数（通常别名为\$）既可以用来选择元素，也可以用来创建新元素。

```
var myLink = $('<a href="http://stackexchange.com"></a>');
```

你可以选择传入第二个参数来设置元素属性：

```
var myLink = $('<a>', { 'href': 'http://stackexchange.com' });
```

'<a>' --> 第一个参数指定你想创建的DOM元素类型。在这个例子中是一个[锚点](#)但也可以是[此列表中的任何元素](#)。参考[规范](#)了解<a>元素的详细信息。

{ 'href': 'http://stackexchange.com' } --> 第二个参数是一个包含属性名/值对的JavaScript对象。

'name': 'value'对将出现在第一个参数的< >之间，例如<a name:value>，对于我们的例子就是

第7.2节：操作元素类

假设页面包含如下HTML元素：

```
<p class="small-paragraph">
这是一个小的<a href="https://en.wikipedia.org/wiki/Paragraph">段落</a>
内部有一个<a class="trusted" href="http://stackexchange.com">链接</a>。
</p>
```

jQuery提供了操作DOM类的有用函数，最著名的有hasClass()、addClass()、removeClass()和toggleClass()。这些函数直接修改匹配元素的class属性。

```
$( 'p' ).hasClass( 'small-paragraph' ); // true
$( 'p' ).hasClass( 'large-paragraph' ); // false

// 给所有段落内的链接添加一个类
$( 'p a' ).addClass( 'untrusted-link-in-paragraph' );

// 从a.trusted中移除该类
$( 'a.trusted.untrusted-link-in-paragraph' )
.removeClass( 'untrusted-link-in-paragraph' )
.addClass( 'trusted-link-in-paragraph' );
```

切换一个类

给定示例标记，我们可以用第一个.toggleClass()添加一个类：

```
$( ".small-paragraph" ).toggleClass( "pretty" );
```

现在这将返回true：\$(".small-paragraph").hasClass("pretty")

toggleClass提供了相同的效果，但代码更简洁，如下：

```
if( $( ".small-paragraph" ).hasClass( "pretty" ) ){
```

Chapter 7: DOM Manipulation

Section 7.1: Creating DOM elements

The jQuery function (usually aliased as \$) can be used both to select elements and to create new elements.

```
var myLink = $('<a href="http://stackexchange.com"></a>');
```

You can optionally pass a second argument with element attributes:

```
var myLink = $('<a>', { 'href': 'http://stackexchange.com' });
```

'<a>' --> The first argument specifies the type of DOM element you want to create. In this example it's an [anchor](#) but could be anything [on this list](#). See the [specification](#) for a reference of the a element.

{ 'href': 'http://stackexchange.com' } --> the second argument is a [JavaScript Object](#) containing attribute name/value pairs.

the 'name': 'value' pairs will appear between the < > of the first argument, for example <a name:value> which for our example would be

Section 7.2: Manipulating element classes

Assuming the page includes an HTML element like:

```
<p class="small-paragraph">
This is a small <a href="https://en.wikipedia.org/wiki/Paragraph">paragraph</a>
with a <a class="trusted" href="http://stackexchange.com">link</a> inside.
</p>
```

jQuery provides useful functions to manipulate DOM classes, most notably hasClass(), addClass(), removeClass() and toggleClass(). These functions directly modify the class attribute of the matched elements.

```
$( 'p' ).hasClass( 'small-paragraph' ); // true
$( 'p' ).hasClass( 'large-paragraph' ); // false

// Add a class to all links within paragraphs
$( 'p a' ).addClass( 'untrusted-link-in-paragraph' );

// Remove the class from a.trusted
$( 'a.trusted.untrusted-link-in-paragraph' )
.removeClass( 'untrusted-link-in-paragraph' )
.addClass( 'trusted-link-in-paragraph' );
```

Toggle a class

Given the example markup, we can add a class with our first .toggleClass():

```
$( ".small-paragraph" ).toggleClass( "pretty" );
```

Now this would return true: \$(".small-paragraph").hasClass("pretty")

toggleClass provides the same effect with less code as:

```
if( $( ".small-paragraph" ).hasClass( "pretty" ) ){
```

```
$(".small-paragraph").removeClass("pretty");}
else {
    $(".small-paragraph").addClass("pretty"); }
```

切换两个类：

```
$(".small-paragraph").toggleClass("pretty cool");
```

布尔值用于添加/移除类：

```
$(".small-paragraph").toggleClass("pretty",true); //不能是真值/假值

$(".small-paragraph").toggleClass("pretty",false);
```

用于切换类的函数（见下方示例以避免问题）

```
$( "div.surface" ).toggleClass(function() {
    if ( $( this ).parent().is( ".water" ) ) {
        return "wet";
    } else {
        return "dry";
    }
});
```

示例中使用：

```
// 示例中使用的函数
function stringContains(myString, mySubString) {
    return myString.indexOf(mySubString) !== -1;
}
function isOdd(num) { return num % 2;}
var showClass = true; //我们想要添加该类
```

示例：

使用元素索引切换奇数/偶数类

```
$( "div.gridrow" ).切换类(函数(索引,旧类, false), 显示类 ) {
    显示类
    如果 ( 是奇数(索引) ) {
        返回 "湿";
    } 否则 {
        返回 "干";
    }
});
```

更复杂的 切换类 示例，给出一个简单的网格标记

```
<div 类="grid">
  <div 类="gridrow">行</div>
  <div 类="gridrow">行</div>
  <div 类="gridrow">行</div>
  <div 类="gridrow">行</div>
  <div 类="gridrow">行</div>
  <div 类="gridrow gridfooter">行但我是页脚！</div>
</div>
```

我们示例的简单函数：

```
$( ".small-paragraph" ).removeClass("pretty");}
else {
    $( ".small-paragraph" ).addClass("pretty"); }
```

toggle Two classes:

```
$( ".small-paragraph" ).toggleClass("pretty cool");
```

Boolean to add/remove classes:

```
$( ".small-paragraph" ).toggleClass("pretty",true); //cannot be truthy/falsey

$( ".small-paragraph" ).toggleClass("pretty",false);
```

Function for class toggle (see example further down to avoid an issue)

```
$( "div.surface" ).toggleClass(function() {
    if ( $( this ).parent().is( ".water" ) ) {
        return "wet";
    } else {
        return "dry";
    }
});
```

Used in examples:

```
// functions to use in examples
function stringContains(myString, mySubString) {
    return myString.indexOf(mySubString) !== -1;
}
function isOdd(num) { return num % 2;}
var showClass = true; //we want to add the class
```

Examples:

Use the element index to toggle classes odd/even

```
$( "div.gridrow" ).toggleClass(function(index,oldClasses, false), showClass ) {
    showClass
    if ( isOdd(index) ) {
        return "wet";
    } else {
        return "dry";
    }
});
```

More complex toggleClass example, given a simple grid markup

```
<div class="grid">
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow gridfooter">row but I am footer!</div>
</div>
```

Simple functions for our examples:

```
函数 是奇数(数字) {  
    返回 数字 % 2;  
}  
  
function stringContains(myString, mySubString) {  
    return myString.indexOf(mySubString) !== -1;  
}  
var showClass = true; //我们想要添加该类
```

为带有gridrow类的元素添加奇数/偶数类

```
$("#div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {  
    if (isOdd(index)) {  
        return "odd";  
    } else {  
        return "even";  
    }  
    return oldClasses;  
}, showClass);
```

如果行具有gridfooter类，则移除奇数/偶数类，保留其余类。

```
$("#div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {  
    var isFooter = stringContains(oldClasses, "gridfooter");  
    if (isFooter) {  
oldClasses = oldClasses.replace('even', ' ').replace('odd', ' ');  
        $(this).toggleClass("even odd", false);  
    }  
    return oldClasses;  
}, showClass);
```

返回的类是受影响的类。在这里，如果一个元素没有gridfooter，则为偶数/奇数添加一个类。此示例说明了返回旧类列表的情况。如果移除这句else return oldClasses;，则只会添加新类，因此如果某行具有gridfooter类，且我们没有返回那些旧类，该行的所有类都会被移除——否则它们会被切换（移除）。

```
$("#div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {  
    var isFooter = stringContains(oldClasses, "gridfooter");  
    if (!isFooter) {  
        if (isOdd(index)) {  
            return "oddLight";  
        } else {  
            return "evenLight";  
        }  
    } else return oldClasses;  
}, showClass);
```

第7.3节：其他API方法

jQuery提供了多种可用于DOM操作的方法。

第一个是.empty()方法。

假设有如下标记：

```
<div id="content">  
  <div>一些文本</div>
```

```
function isOdd(num) {  
    return num % 2;  
}  
  
function stringContains(myString, mySubString) {  
    return myString.indexOf(mySubString) !== -1;  
}  
var showClass = true; //we want to add the class
```

Add an odd/even class to elements with a gridrow class

```
$("#div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {  
    if (isOdd(index)) {  
        return "odd";  
    } else {  
        return "even";  
    }  
    return oldClasses;  
}, showClass);
```

If the row has a gridfooter class, remove the odd/even classes, keep the rest.

```
$("#div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {  
    var isFooter = stringContains(oldClasses, "gridfooter");  
    if (isFooter) {  
        oldClasses = oldClasses.replace('even', ' ').replace('odd', ' ');  
        $(this).toggleClass("even odd", false);  
    }  
    return oldClasses;  
}, showClass);
```

The classes that get returned are what is effected. Here, if an element does not have a gridfooter, add a class for even/odd. This example illustrates the return of the OLD class list. If this else return oldClasses; is removed, only the new classes get added, thus the row with a gridfooter class would have all classes removed had we not returned those old ones - they would have been toggled (removed) otherwise.

```
$("#div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {  
    var isFooter = stringContains(oldClasses, "gridfooter");  
    if (!isFooter) {  
        if (isOdd(index)) {  
            return "oddLight";  
        } else {  
            return "evenLight";  
        }  
    } else return oldClasses;  
}, showClass);
```

Section 7.3: Other API Methods

jQuery offers a variety of methods that can be used for DOM manipulation.

The first is the .empty() method.

Imagine the following markup:

```
<div id="content">  
  <div>Some text</div>
```

```
</div>
```

通过调用 `$('#content').empty();`，内部的div将被移除。这也可以通过使用 `$('#content').html('');` 来实现。

另一个方便的函数是 `closest()` 函数：[_____](#)

```
<tr id="row_1">
  <td><button type="button" class="delete">删除</button>
</tr>
```

如果你想找到点击的按钮所在的单元格中最近的行，可以这样做：

```
$('.delete').click(function() {
  $(this).closest('tr');
});
```

由于可能有多行，每行都有自己的delete按钮，我们在 `click()` 函数中使用 `$(this)` 来限制范围到我们实际点击的按钮。

如果你想获取包含你点击的删除按钮的行的id，可以这样做：

```
$('.delete').click(function() {
  var $row = $(this).closest('tr');
  var id = $row.attr('id');
});
```

通常认为给包含jQuery对象的变量加上\$（美元符号）前缀是个好习惯，这样可以清楚地表明变量的类型。

`closest()` 的另一种替代方法是 `parents()` 方法：[_____](#)

```
$('.delete').click(function() {
  var $row = $(this).parents('tr');
  var id = $row.attr('id');
});
```

并且还有一个 `parent()` 函数：[_____](#)

```
$('.delete').click(function() {
  var $row = $(this).parent().parent();
  var id = $row.attr('id');
});
```

`.parent()` 只会向上遍历DOM树一级，因此它相当不灵活，如果你将删除按钮改为包含在一个span中，例如，那么jQuery选择器将会失效。

```
</div>
```

By calling `$('#content').empty();`，the inner div would be removed. This could also be achieved by using `$('#content').html('');`

Another handy function is the [.closest\(\)](#) function:

```
<tr id="row_1">
  <td><button type="button" class="delete">Delete</button>
</tr>
```

If you wanted to find the closest row to a button that was clicked within one of the row cells then you could do this:

```
$('.delete').click(function() {
  $(this).closest('tr');
});
```

Since there will probably be multiple rows, each with their own **delete** buttons, we use `$(this)` within the [.click\(\)](#) function to limit the scope to the button we actually clicked.

If you wanted to get the id of the row containing the **Delete** button that you clicked, you could do something like this:

```
$('.delete').click(function() {
  var $row = $(this).closest('tr');
  var id = $row.attr('id');
});
```

It is usually considered good practise to prefix variables containing jQuery objects with a \$ (dollar sign) to make it clear what the variable is.

An alternative to `.closest()` is the [.parents\(\)](#) method:

```
$('.delete').click(function() {
  var $row = $(this).parents('tr');
  var id = $row.attr('id');
});
```

and there is also a [.parent\(\)](#) function as well:

```
$('.delete').click(function() {
  var $row = $(this).parent().parent();
  var id = $row.attr('id');
});
```

`.parent()` only goes up one level of the DOM tree so it is quite inflexible, if you were to change the delete button to be contained within a span for example, then the jQuery selector would be broken.

第8章：DOM遍历

第8.1节：选择元素的子元素

要选择元素的子元素，可以使用children()方法。

```
<div class="parent">
  <h2>一个标题</h2>
  <p>Lorem ipsum dolor sit amet...</p>
  <p>Praesent quis dolor turpis...</p>
</div>
```

更改所有.parent元素的子元素的颜色：

```
$('.parent').children().css("color", "green");
```

该方法接受一个可选的selector参数，可用于筛选返回的元素。

```
// 仅获取“p”子元素
$('.parent').children("p").css("color", "green");
```

第8.2节：获取下一个元素

要获取下一个元素，可以使用.next()方法。

```
<ul>
  <li>Mark</li>
  <li class="anna">Anna</li>
  <li>Paul</li>
</ul>
```

如果你当前选中的是“Anna”元素，想要获取下一个元素“Paul”，.next()方法可以帮你实现。

```
// “Paul”的文字现在变成绿色
$(".anna").next().css("color", "green");
```

该方法接受一个可选的selector参数，如果下一个元素必须是某种特定类型的元素，则可以使用该参数。

```
// 下一个元素是一个 "li", "Paul" 现在的文字是绿色的
$(".anna").next("li").css("color", "green");
```

如果下一个元素不是 selector 类型，则返回一个空集合，修改操作将不会生效。

```
// 下一个元素不是 ".mark", 在这种情况下不会执行任何操作
$(".anna").next(".mark").css("color", "green");
```

第8.3节：获取前一个元素

要获取前一个元素，可以使用 .prev() 方法。

```
<ul>
```

Chapter 8: DOM Traversing

Section 8.1: Select children of element

To select the children of an element you can use the `children()` method.

```
<div class="parent">
  <h2>A headline</h2>
  <p>Lorem ipsum dolor sit amet...</p>
  <p>Praesent quis dolor turpis...</p>
</div>
```

Change the color of *all* the children of the `.parent` element:

```
$('.parent').children().css("color", "green");
```

The method accepts an optional selector argument that can be used to filter the elements that are returned.

```
// Only get "p" children
$('.parent').children("p").css("color", "green");
```

Section 8.2: Get next element

To get the next element you can use the `.next()` method.

```
<ul>
  <li>Mark</li>
  <li class="anna">Anna</li>
  <li>Paul</li>
</ul>
```

If you are standing on the "Anna" element and you want to get the next element, "Paul", the `.next()` method will allow you to do that.

```
// "Paul" now has green text
$(".anna").next().css("color", "green");
```

The method takes an optional selector argument, which can be used if the next element must be a certain kind of element.

```
// Next element is a "li", "Paul" now has green text
$(".anna").next("li").css("color", "green");
```

If the next element is not of the type selector then an empty set is returned, and the modifications will not do anything.

```
// Next element is not a ".mark", nothing will be done in this case
$(".anna").next(".mark").css("color", "green");
```

Section 8.3: Get previous element

To get the previous element you can use the `.prev()` method.

```
<ul>
```



```
<li>Mark</li>
<li class="anna">Anna</li>
<li>Paul</li>
</ul>
```

如果你当前选中的是 "Anna" 元素，想要获取前一个元素 "Mark"，`.prev()` 方法可以实现这一点。

```
// "Mark" 现在的文字是绿色的
$(".anna").prev().css("color", "green");
```

该方法接受一个可选的selector参数，当前一个元素必须是某种特定类型的元素时可以使用该参数。

```
// 上一个元素是“li”，“Mark”的文本现在是绿色的
$(".anna").prev("li").css("color", "green");
```

如果上一个元素不是selector指定的类型，则返回一个空集合，且不会进行任何修改。

```
// 上一个元素不是“.paul”，在这种情况下不会执行任何操作
$(".anna").prev("paul").css("color", "green");
```

第8.4节：过滤选择

要过滤选择，可以使用`.filter()`方法。

该方法在一个选择集上调用并返回一个新的选择集。如果过滤条件匹配某个元素，则该元素被添加到返回的选择集中，否则被忽略。如果没有元素匹配，则返回一个空选择集。

HTML

这是我们将使用的HTML。

```
<ul>
  <li class="zero">零</li>
  <li class="one">一</li>
  <li class="two">二</li>
  <li class="three">三</li>
</ul>
```

选择器

使用选择器进行过滤是过滤选择项的较简单方法之一。

```
$("li").filter(":even").css("color", "green"); // 将偶数元素颜色设为绿色
$("li").filter(".one").css("font-weight", "bold"); // 使“.one”加粗
```

函数

使用函数过滤选择项在无法使用选择器时非常有用。

该函数会对选择中的每个元素调用。如果返回true值，则该元素将被添加到返回的选择中。

```
<li>Mark</li>
<li class="anna">Anna</li>
<li>Paul</li>
</ul>
```

If you are standing on the "Anna" element and you want to get the previous element, "Mark", the `.prev()` method will allow you to do that.

```
// "Mark" now has green text
$(".anna").prev().css("color", "green");
```

The method takes an optional selector argument, which can be used if the previous element must be a certain kind of element.

```
// Previous element is a "li", "Mark" now has green text
$(".anna").prev("li").css("color", "green");
```

If the previous element is not of the type selector then an empty set is returned, and the modifications will not do anything.

```
// Previous element is not a ".paul", nothing will be done in this case
$(".anna").prev("paul").css("color", "green");
```

Section 8.4: Filter a selection

To filter a selection you can use the `.filter()` method.

The method is called on a selection and returns a new selection. If the filter matches an element then it is added to the returned selection, otherwise it is ignored. If no element is matched then an empty selection is returned.

The HTML

This is the HTML we will be using.

```
<ul>
  <li class="zero">Zero</li>
  <li class="one">One</li>
  <li class="two">Two</li>
  <li class="three">Three</li>
</ul>
```

Selector

Filtering using selectors is one of the simpler ways to filter a selection.

```
$("li").filter(":even").css("color", "green"); // Color even elements green
$("li").filter(".one").css("font-weight", "bold"); // Make ".one" bold
```

Function

Filtering a selection using a function is useful if it is not possible to use selectors.

The function is called for each element in the selection. If it returns a **true** value then the element will be added to the returned selection.

```
var selection = $("li").filter(function (index, element) {
    // "index"是元素的位置
    // "element"与"this"相同
    return $(this).hasClass("two");
});
selection.css("color", "green"); // ".two" 将被染成绿色
```

元素

你可以通过 DOM 元素进行过滤。如果 DOM 元素在选择集中，则它们会被包含在返回的选择集中。

```
var three = document.getElementsByClassName("three");
$("li").filter(three).css("color", "green");
```

选择

你也可以通过另一个选择集来过滤选择集。如果一个元素同时存在于两个选择集中，则它会被包含在返回的选择集中。

```
var elems = $(".one, .three");
$("li").filter(elems).css("color", "green");
```

第8.5节：find() 方法

.find() 方法允许我们在 DOM 树中搜索这些元素的后代，并从匹配的元素构造一个新的 jQuery 对象。

HTML

```
<div class="parent">
  <div class="children" name="first">
    <ul>
      <li>A1</li>
      <li>A2</li>
      <li>A3</li>
    </ul>
  </div>
  <div class="children" name="second">
    <ul>
      <li>B1</li>
      <li>B2</li>
      <li>B3</li>
    </ul>
  </div>
</div>
```

jQuery

```
$('.parent').find('.children[name="second"] ul li').css('font-weight','bold');
```

输出

- A1
- A2
- A3

```
var selection = $("li").filter(function (index, element) {
    // "index" is the position of the element
    // "element" is the same as "this"
    return $(this).hasClass("two");
});
selection.css("color", "green"); // ".two" will be colored green
```

Elements

You can filter by DOM elements. If the DOM elements are in the selection then they will be included in the returned selection.

```
var three = document.getElementsByClassName("three");
$("li").filter(three).css("color", "green");
```

Selection

You can also filter a selection by another selection. If an element is in both selections then it will be included in the returned selection.

```
var elems = $(".one, .three");
$("li").filter(elems).css("color", "green");
```

Section 8.5: find() method

.find() method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements.

HTML

```
<div class="parent">
  <div class="children" name="first">
    <ul>
      <li>A1</li>
      <li>A2</li>
      <li>A3</li>
    </ul>
  </div>
  <div class="children" name="second">
    <ul>
      <li>B1</li>
      <li>B2</li>
      <li>B3</li>
    </ul>
  </div>
</div>
```

jQuery

```
$('.parent').find('.children[name="second"] ul li').css('font-weight','bold');
```

Output

- A1
- A2
- A3

- B1
- B2
- B3

第8.6节：遍历jQuery元素列表

当你需要遍历 jQuery 元素列表时。

考虑以下 DOM 结构：

```
<div class="container">
  <div class="red one">红色 1 信息</div>
  <div class="red two">红色 2 信息</div>
  <div class="red three">红色 3 信息</div>
</div>
```

打印所有 class 为 red 的 div 元素中的文本：

```
$( ".red" ).each(function(key, ele){
  var text = $(ele).text();
  console.log(text);
});
```

提示：key 是当前正在遍历的 div.red 元素在其父元素中的索引。ele 是 HTML 元素，因此我们可以使用 `$()` 或 `jQuery()` 创建一个 jQuery 对象，如：`$(ele)`。之后，我们可以在该对象上调用任何 jQuery 方法，比如 `css()` 或 `hide()` 等。在本例中，我们只是获取该对象的文本内容。

第 8.7 节：选择兄弟元素

要选择某个元素的兄弟元素，可以使用 `.siblings()` 方法。

一个典型的例子是你想修改菜单中某个元素的兄弟元素：

```
<ul class="menu">
  <li class="selected">首页</li>
  <li>博客</li>
  <li>关于</li>
</ul>
```

当用户点击菜单项时，应将selected类添加到被点击的元素上，并从其

兄弟元素中移除：

```
$( ".menu" ).on("click", "li", function () {
  $(this).addClass("selected");
  $(this).siblings().removeClass("selected");
});
```

该方法接受一个可选的selector参数，如果需要缩小选择的兄弟元素范围，可以使用该参数：

```
$(this).siblings("li").removeClass("selected");
```

第8.8节：closest()方法

返回从该元素开始向上遍历DOM树，匹配选择器的第一个元素。

- B1
- B2
- B3

Section 8.6: Iterating over list of jQuery elements

When you need to iterate over the list of jQuery elements.

Consider this DOM structure:

```
<div class="container">
  <div class="red one">RED 1 Info</div>
  <div class="red two">RED 2 Info</div>
  <div class="red three">RED 3 Info</div>
</div>
```

To print the text present in all the div elements with a class of red:

```
$( ".red" ).each(function(key, ele){
  var text = $(ele).text();
  console.log(text);
});
```

Tip: key is the index of the div.red element we're currently iterating over, within its parent. ele is the HTML element, so we can create a jQuery object from it using `$()` or `jQuery()`, like so: `$(ele)`. After, we can call any jQuery method on the object, like `css()` or `hide()` etc. In this example, we just pull the text of the object.

Section 8.7: Selecting siblings

To select siblings of an item you can use the `.siblings()` method.

A typical example where you want to modify the siblings of an item is in a menu:

```
<ul class="menu">
  <li class="selected">Home</li>
  <li>Blog</li>
  <li>About</li>
</ul>
```

When the user clicks on a menu item the selected class should be added to the clicked element and removed from its *siblings*:

```
$( ".menu" ).on("click", "li", function () {
  $(this).addClass("selected");
  $(this).siblings().removeClass("selected");
});
```

The method takes an optional selector argument, which can be used if you need to narrow down the kinds of siblings you want to select:

```
$(this).siblings("li").removeClass("selected");
```

Section 8.8: closest() method

Returns the first element that matches the selector starting at the element and traversing up the DOM tree.

HTML

```
<div id="abc" class="row">
  <div id="xyz" class="row">
  </div>
  <p id="origin">
    你好
  </p>
</div>
```

jQuery

```
var target = $('#origin').closest('.row');
console.log("最近的行:", target.attr('id') );

var target2 = $('#origin').closest('p');
console.log("最近的p:", target2.attr('id') );
```

输出

```
"最近的行: abc"
"最近的p: origin"
```

first() 方法： first 方法返回匹配元素集合中的第一个元素。

HTML

```
<div class='firstExample'>
  <p>这是 div 中的第一段。</p>
  <p>这是 div 中的第二段。</p>
  <p>这是 div 中的第三段。</p>
  <p>这是 div 中的第四段。</p>
  <p>这是 div 中的第五段。</p>
</div>
```

jQuery

```
var firstParagraph = $("div p").first();
console.log("第一段:", firstParagraph.text());
```

输出：

```
第一段: 这是 div 中的第一段。
```

HTML

```
<div id="abc" class="row">
  <div id="xyz" class="row">
  </div>
  <p id="origin">
    Hello
  </p>
</div>
```

jQuery

```
var target = $('#origin').closest('.row');
console.log("Closest row:", target.attr('id') );

var target2 = $('#origin').closest('p');
console.log("Closest p:", target2.attr('id') );
```

Output

```
"Closest row: abc"
"Closest p: origin"
```

first() method : The first method returns the first element from the matched set of elements.

HTML

```
<div class='.firstExample'>
  <p>This is first paragraph in a div.</p>
  <p>This is second paragraph in a div.</p>
  <p>This is third paragraph in a div.</p>
  <p>This is fourth paragraph in a div.</p>
  <p>This is fifth paragraph in a div.</p>
</div>
```

jQuery

```
var firstParagraph = $("div p").first();
console.log("First paragraph:", firstParagraph.text());
```

Output:

```
First paragraph: This is first paragraph in a div.
```

第9章：CSS 操作

第9.1节：CSS – 访问器和设置器

CSS 获取器

.css() getter 函数可以应用于页面上的每个 DOM 元素，如下所示：

```
// 以字符串形式返回渲染宽度（单位为像素）。例如：`150px`  
// 注意“以字符串形式”这一说明——如果需要真正的整数，  
// 请参考 `$.width()` 方法  
$("body").css("width");
```

这行代码将返回指定元素的计算宽度，您在括号中提供的每个 CSS 属性都会返回该\$("选择器") DOM 元素对
应属性的值，如果请求不存在的 CSS 属性，将返回undefined。

您也可以使用属性数组调用CSS 获取器：

```
$("body").css(["animation","width"]);
```

这将返回一个包含所有属性及其值的对象：

```
Object {animation: "none 0s ease 0s 1 normal none running", width: "529px"}
```

CSS 设置器

.css() 设置器方法也可以应用于页面上的每个DOM元素。

```
$("#selector").css("width", 500);
```

该语句将 \$("#selector") 的 width 设置为 500px，并返回jQuery对象，以便你可以对指定选择器链式调用更多方法。

.css() 设置器也可以通过传递一个CSS属性和值的对象来使用，如下所示：

```
$("#body").css({ "height": "100px", width:100, "padding-top":40, paddingBottom:"2em"});
```

设置器所做的所有更改都会附加到DOM元素的 style 属性，从而影响元素的样式（除非该样式属性值已在其他样式中
以 !important 定义）。

第9.2节：数值属性的递增/递减

数值CSS属性可以分别使用 += 和 -= 语法进行递增和递减，使用
.css() 方法：

```
// 使用 += 语法递增  
$("#target-element").css("font-size", "+=10");  
  
// 你也可以指定递增的单位  
$("#target-element").css("width", "+=100pt");  
$("#target-element").css("top", "+=30px");  
$("#target-element").css("left", "+=3em");
```

Chapter 9: CSS Manipulation

Section 9.1: CSS – Getters and Setters

CSS Getter

The .css() **getter** function can be applied to every DOM element on the page like the following:

```
// Rendered width in px as a string. ex: `150px`  
// Notice the `as a string` designation - if you require a true integer,  
// refer to `$.width()` method  
$("body").css("width");
```

This line will return the **computed width** of the specified element, each CSS property you provide in the
parentheses will yield the value of the property for this \$("selector") DOM element, if you ask for CSS attribute
that doesn't exist you will get undefined as a response.

You also can call the **CSS getter** with an array of attributes:

```
$("#body").css(["animation", "width"]);
```

this will return an object of all the attributes with their values:

```
Object {animation: "none 0s ease 0s 1 normal none running", width: "529px"}
```

CSS Setter

The .css() **setter** method can also be applied to every DOM element on the page.

```
$("#selector").css("width", 500);
```

This statement set the width of the \$("selector") to 500px and return the jQuery object so you can chain more
methods to the specified selector.

The .css() **setter** can also be used passing an Object of CSS properties and values like:

```
$("#body").css({ "height": "100px", width:100, "padding-top":40, paddingBottom:"2em"});
```

All the changes the setter made are appended to the DOM element style property thus affecting the elements'
styles (unless that style property value is already defined as !important somewhere else in styles).

Section 9.2: Increment/Decrement Numeric Properties

Numeric CSS properties can be incremented and decremented with the += and -= syntax, respectively, using the
.css() method:

```
// Increment using the += syntax  
$("#target-element").css("font-size", "+=10");  
  
// You can also specify the unit to increment by  
$("#target-element").css("width", "+=100pt");  
$("#target-element").css("top", "+=30px");  
$("#target-element").css("left", "+=3em");
```



```
// 递减是通过使用 -= 语法完成的
$("#target-element").css("height", "-=50pt");
```

第9.3节：设置CSS属性

只设置一个样式：

```
$('#target-element').css('color', '#000000');
```

同时设置多个样式：

```
$('#target-element').css({
  'color': '#000000',
  'font-size': '12pt',
  'float': 'left',
});
```

第9.4节：获取CSS属性

要获取元素的CSS属性，可以使用.css(propertyName)方法：

```
var color    = $('#element').css('color');
var fontSize = $('#element').css('font-size');
```

```
// Decrementing is done by using the -= syntax
$("#target-element").css("height", "-=50pt");
```

Section 9.3: Set CSS property

Setting only one style:

```
$('#target-element').css('color', '#000000');
```

Setting multiple styles at the same time:

```
$('#target-element').css({
  'color': '#000000',
  'font-size': '12pt',
  'float': 'left',
});
```

Section 9.4: Get CSS property

To get an element's CSS property you can use the .css(propertyName) method:

```
var color    = $('#element').css('color');
var fontSize = $('#element').css('font-size');
```

第10章：元素可见性

参数	详情
持续时间	传入时，.hide()、.show()和.toggle()的效果将带动画；元素将逐渐淡入或淡出。

第10.1节：概述

```
$(element).hide()           // 设置 display: none
$(element).show()           // 设置 display 为原始值
$(element).toggle()         // 在两者之间切换
$(element).是(':visible')    // 返回 true 或 false
$('element:visible')        // 匹配所有可见的元素
$('element:hidden')         // 匹配所有隐藏的元素

$('element').fadeIn();      // 显示该元素
$('element').fadeOut();     // 隐藏元素

$('element').fadeIn(1000);   // 使用定时器显示元素
$('element').fadeOut(1000);  // 使用定时器隐藏元素

// 使用定时器和回调函数显示元素
$('element').fadeIn(1000, function(){
  // 执行的代码
});

// 使用定时器和回调函数隐藏元素
$('element').fadeOut(1000, function(){
  // 执行的代码
});
```

第10.2节：切换功能

简单的 toggle()用例

```
function toggleBasic() {
  $(".target1").toggle();
}
```

指定持续时间

```
function toggleDuration() {
  $(".target2").toggle("slow"); // 也可以接受毫秒为单位的持续时间值
}
```

...和回调函数

```
function toggleCallback() {
  $(".target3").toggle("slow",function(){alert('现在执行某操作');});
}
```

...或者带有缓动效果和回调函数。

```
function toggleEasingAndCallback() {
  // 你可以使用jQueryUI, 因为核心只支持线性(linear)和摆动(swing)缓动效果
  $(".target4").toggle("slow", "linear", function(){alert('现在执行某操作');});
}
```

Chapter 10: Element Visibility

Parameter	Details
Duration	When passed, the effects of .hide(), .show() and .toggle() are animated; the element(s) will gradually fade in or out.

Section 10.1: Overview

```
$(element).hide()           // sets display: none
$(element).show()           // sets display to original value
$(element).toggle()         // toggles between the two
$(element).is(':visible')    // returns true or false
$('element:visible')        // matches all elements that are visible
$('element:hidden')         // matches all elements that are hidden

$('element').fadeIn();      // display the element
$('element').fadeOut();     // hide the element

$('element').fadeIn(1000);   // display the element using timer
$('element').fadeOut(1000);  // hide the element using timer

// display the element using timer and a callback function
$('element').fadeIn(1000, function(){
  // code to execute
});

// hide the element using timer and a callback function
$('element').fadeOut(1000, function(){
  // code to execute
});
```

Section 10.2: Toggle possibilities

Simple toggle() case

```
function toggleBasic() {
  $(".target1").toggle();
}
```

With specific duration

```
function toggleDuration() {
  $(".target2").toggle("slow"); // A millisecond duration value is also acceptable
}
```

...and callback

```
function toggleCallback() {
  $(".target3").toggle("slow",function(){alert('now do something');});
}
```

...or with easing and callback.

```
function toggleEasingAndCallback() {
  // You may use jQueryUI as the core only supports linear and swing easings
  $(".target4").toggle("slow", "linear", function(){alert('now do something');});
}
```

```
}
```

...或者带有多种选项。

```
function toggleWithOptions() {
  $(".target5").toggle(
    { // 详见所有可选项: api.jquery.com/toggle/#toggle-options
      duration:1000, // 毫秒
      easing:"linear",
      done:function(){
        alert('现在执行某操作');
      }
    }
  );
}
```

也可以使用 `slide` 作为动画效果, 配合 `slideToggle()`

```
function toggleSlide() {
  $(".target6").slideToggle(); // 从上到下动画, 而非从顶部角落
}
```

...或者通过改变透明度使用 `fadeToggle()` 实现淡入淡出

```
function toggleFading() {
  $( ".target7" ).fadeToggle("slow")
}
```

...或者使用 `toggleClass()` 切换类名

```
function toggleClass() {
  $(".target8").toggleClass('active');
}
```

一个常见的情况是使用 `toggle()` 来显示一个元素, 同时隐藏另一个 (相同类名)

```
function toggleX() {
  $(".targetX").toggle("slow");
}
```

以上所有示例都可以在 [here](#) 找到

```
}
```

...or with a variety of *options*.

```
function toggleWithOptions() {
  $(".target5").toggle(
    { // See all possible options in: api.jquery.com/toggle/#toggle-options
      duration:1000, // milliseconds
      easing:"linear",
      done:function(){
        alert('now do something');
      }
    }
  );
}
```

It's also possible to use a *slide* as animation with `slideToggle()`

```
function toggleSlide() {
  $(".target6").slideToggle(); // Animates from top to bottom, instead of top corner
}
```

...or fade in/out by changing opacity with `fadeToggle()`

```
function toggleFading() {
  $( ".target7" ).fadeToggle("slow")
}
```

...or toggle a class with `toggleClass()`

```
function toggleClass() {
  $(".target8").toggleClass('active');
}
```

A common case is to use `toggle()` in order to show one element while hiding the other (same class)

```
function toggleX() {
  $(".targetX").toggle("slow");
}
```

All the above examples can be found [here](#)

第11章：追加

参数	详情
内容	可能的类型：元素、HTML字符串、文本、数组、对象，甚至是返回字符串的函数。

第11.1节：高效的连续 .append() 使用

开始：

```
HTML
<table id='my-table' width='960' height='500'></table>
```

```
JS
var data = [
  { type: "姓名", content: "约翰·多伊" },
  { type: "出生日期", content: "1970年01月01日" },
  { type: "薪水", content: "$40,000,000" },
  // ...另外300行数据...
  { type: "喜欢的口味", content: "酸味" }
];
```

在循环中追加

你刚收到一个大型数据数组。现在是时候遍历它并在页面上渲染了。

你可能首先想到的是这样做：

```
var i; // <- 当前项编号
var count = data.length; // <- 总数
var row; // <- 用于保存行对象的引用

// 遍历数组
for ( i = 0; i < count; ++i ) {
  row = data[ i ];

  // 将整行放入你的表格中
  $('#my-table').append(
    $('<tr></tr>').append(
      $('<td></td>').html(row.type),
      $('<td></td>').html(row.content)
    )
  );
}
```

这是完全有效的，并且会准确渲染你所期望的内容，但...

不要这样做。

还记得那些300+行数据吗？

每一行都会迫使浏览器重新计算每个元素的宽度、高度和定位值，以及任何其他样式——除非它们被一个布局边界分隔开，但遗憾的是在这个例子中（因为它们是<table>元素的子元素），无法做到这一点。

在数据量小且列数少的情况下，这种性能损失肯定可以忽略不计。但我们希望每一毫秒都能发挥作用。

Chapter 11: Append

Parameters	Details
content	Possible types: Element, HTML string, text, array, object or even a function returning a string.

Section 11.1: Efficient consecutive .append() usage

Starting out:

```
HTML
<table id='my-table' width='960' height='500'></table>
```

```
JS
var data = [
  { type: "Name", content: "John Doe" },
  { type: "Birthdate", content: "01/01/1970" },
  { type: "Salary", content: "$40,000,000" },
  // ...300 more rows...
  { type: "Favorite Flavour", content: "Sour" }
];
```

Appending inside a loop

You just received a big array of data. Now it's time to loop through and render it on the page.

Your first thought may be to do something like this:

```
var i; // <- the current item number
var count = data.length; // <- the total
var row; // <- for holding a reference to our row object

// Loop over the array
for ( i = 0; i < count; ++i ) {
  row = data[ i ];

  // Put the whole row into your table
  $('#my-table').append(
    $('<tr></tr>').append(
      $('<td></td>').html(row.type),
      $('<td></td>').html(row.content)
    )
  );
}
```

This is *perfectly valid* and will render exactly what you'd expect, but...

DO NOT do this.

Remember those **300+** rows of data?

Each one will force the browser to re-calculate every element's width, height and positioning values, along with any other styles - unless they are separated by a [layout boundary](#), which unfortunately for this example (as they are descendants of a **<table>** element), they cannot.

At small amounts and few columns, this performance penalty will certainly be negligible. But we want every millisecond to count.

1. 添加到一个单独的数组中，循环完成后追加

```
/**
 * 应尽可能避免重复的DOM遍历（沿着元素树向下查找，直到找到你想要的元素——比如我们的<table>）。
 */

// 将表格缓存到变量中，然后使用它，直到你认为它已被移除
var $myTable = $('#my-table');

// 用于保存我们新的<tr> jQuery对象
var rowElements = [];

var count = data.length;
var i;
var row;

// 遍历数组
for ( i = 0; i < count; ++i ) {
    rowElements.push(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}

// 最后，一次性插入所有行
$myTable.append(rowElements);
```

在这些选项中，这个最依赖 jQuery。

2.使用现代的 Array.* 方法

```
var $myTable = $('#my-table');

// 使用 .map() 方法循环
// - 这将基于回调函数的结果给我们一个全新的数组
var rowElements = data.map(function ( row ) {

    // 创建一行
    var $row = $('<tr></tr>');

    // 创建列
    var $type = $('<td></td>').html(row.类型);
    var $content = $('<td></td>').html(row.content);

    // 将列添加到行中
    $row.append($type, $content);

    // 添加到新生成的数组中
    return $row;
});

// 最后，将所有行添加到你的表格中
$myTable.append(rowElements);
```

1. Add to a separate array, append after loop completes

```
/**
 * Repeated DOM traversal (following the tree of elements down until you reach
 * what you're looking for - like our <table>) should also be avoided wherever possible.
 */

// Keep the table cached in a variable then use it until you think it's been removed
var $myTable = $('#my-table');

// To hold our new <tr> jQuery objects
var rowElements = [];

var count = data.length;
var i;
var row;

// Loop over the array
for ( i = 0; i < count; ++i ) {
    rowElements.push(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}

// Finally, insert ALL rows at once
$myTable.append(rowElements);
```

Out of these options, this one relies on jQuery the most.

2. Using modern Array.* methods

```
var $myTable = $('#my-table');

// Looping with the .map() method
// - This will give us a brand new array based on the result of our callback function
var rowElements = data.map(function ( row ) {

    // Create a row
    var $row = $('<tr></tr>');

    // Create the columns
    var $type = $('<td></td>').html(row.type);
    var $content = $('<td></td>').html(row.content);

    // Add the columns to the row
    $row.append($type, $content);

    // Add to the newly-generated array
    return $row;
});

// Finally, put ALL of the rows into your table
$myTable.append(rowElements);
```


功能上等同于之前的，只是更易读。

3.使用HTML字符串（而非jQuery内置方法）

```
// ...
var rowElements = data.map(function ( row ) {
    var rowHTML = '<tr><td>';
    rowHTML += row.type;
    rowHTML += '</td><td>';
    rowHTML += row.content;
    rowHTML += '</td></tr>';
    return rowHTML;
});
```

// 使用 .join("") 将所有独立的字符串合并为一个
\$myTable.append(rowElements.join(""));

完全有效 但同样，**不推荐**。这会强制 jQuery 一次解析大量文本，这是不必要的。正确使用时，jQuery 非常擅长它的工作。

4.手动创建元素，追加到文档片段

```
var $myTable = $(document.getElementById('my-table'));
```

```
/**
 * 创建一个文档片段来保存我们的列
 * - 在追加到每一行后，它会清空自身
 * 这样我们可以在下一次迭代中重复使用它。
 */
```

```
var colFragment = document.createDocumentFragment();
```

```
/**
 * 这次使用 .reduce() 遍历数组。
 * 我们得到一个整洁的输出，没有任何副作用。
 * - 在此示例中，结果将是一个
 *   文档片段，包含所有的 <tr> 元素。
 */
```

```
var rowFragment = data.reduce(function ( fragment, row ) {
```

```
    // 创建一行
```

```
    var rowEl = document.createElement('tr');
```

```
    // 创建列和内部文本节点
```

```
    var typeEl = document.createElement('td');
```

```
    var typeText = document.createTextNode(row.type);
```

```
    typeEl.appendChild(typeText);
```

```
    var contentEl = document.createElement('td');
```

```
    var contentText = document.createTextNode(row.content);
```

```
    contentEl.appendChild(contentText);
```

```
    // 将列添加到列片段
```

```
    // - 如果列是单独迭代的，这将很有用
```

```
    // 但在此示例中仅用于演示。
```

```
    colFragment.appendChild(typeEl);
```

```
    colFragment.appendChild(contentEl);
```

Functionally equivalent to the one before it, only easier to read.

3. Using strings of HTML (instead of jQuery built-in methods)

```
// ...
var rowElements = data.map(function ( row ) {
    var rowHTML = '<tr><td>';
    rowHTML += row.type;
    rowHTML += '</td><td>';
    rowHTML += row.content;
    rowHTML += '</td></tr>';
    return rowHTML;
});
```

// Using .join('') here combines all the separate strings into one
\$myTable.append(rowElements.join(''));

Perfectly valid but again, **not recommended**. This forces jQuery to parse a very large amount of text at once and is not necessary. jQuery is very good at what it does when used correctly.

4. Manually create elements, append to document fragment

```
var $myTable = $(document.getElementById('my-table'));
```

```
/**
 * Create a document fragment to hold our columns
 * - after appending this to each row, it empties itself
 *   so we can re-use it in the next iteration.
 */
```

```
var colFragment = document.createDocumentFragment();
```

```
/**
 * Loop over the array using .reduce() this time.
 * We get a nice, tidy output without any side-effects.
 * - In this example, the result will be a
 *   document fragment holding all the <tr> elements.
 */
```

```
var rowFragment = data.reduce(function ( fragment, row ) {
```

```
    // Create a row
```

```
    var rowEl = document.createElement('tr');
```

```
    // Create the columns and the inner text nodes
```

```
    var typeEl = document.createElement('td');
```

```
    var typeText = document.createTextNode(row.type);
```

```
    typeEl.appendChild(typeText);
```

```
    var contentEl = document.createElement('td');
```

```
    var contentText = document.createTextNode(row.content);
```

```
    contentEl.appendChild(contentText);
```

```
    // Add the columns to the column fragment
```

```
    // - this would be useful if columns were iterated over separately
```

```
    // but in this example it's just for show and tell.
```

```
    colFragment.appendChild(typeEl);
```

```
    colFragment.appendChild(contentEl);
```

```
rowEl.appendChild(colFragment);

// 将 rowEl 添加到 fragment —— 这作为一个临时缓冲区, 用于
// 在批量插入之前累积多个 DOM 节点
fragment.appendChild(rowEl);

return fragment;
}, document.createDocumentFragment());

// 现在将整个 fragment 插入到你的表格中
$myTable.append(rowFragment);
```

我个人最喜欢的。这说明了 jQuery 在更底层所做的一般思路。

深入了解

- [jQuery 源码查看器](#)
- [Array.prototype.join\(\)](#)
- [Array.prototype.map\(\)](#)
- [Array.prototype.reduce\(\)](#)
- [document.createDocumentFragment\(\)](#)
- [document.createTextNode\(\)](#)
- [Google Web 基础 - 性能](#)

第11.2节：jQuery append

HTML

```
<p>这是一个不错的</p>
<p>我喜欢</p>

<ul>
  <li>列表项 1</li>
  <li>列表项 2</li>
  <li>列表项 3</li>
</ul>

<button id="btn-1">追加文本</button>
<button id="btn-2">追加列表项</button>
```

脚本

```
$("#btn-1").click(function(){
    $("p").append(" <b>书</b>.");
});
$("#btn-2").click(function(){
    $("ul").append("<li>追加的列表项</li>");
});
});
```

第11.3节：向容器追加元素

方案 1：

```
$('#parent').append($('#child'));
```

```
rowEl.appendChild(colFragment);

// Add rowEl to fragment - this acts as a temporary buffer to
// accumulate multiple DOM nodes before bulk insertion
fragment.appendChild(rowEl);

return fragment;
}, document.createDocumentFragment());

// Now dump the whole fragment into your table
$myTable.append(rowFragment);
```

My personal favorite. This illustrates a general idea of what jQuery does at a lower level.

Dive deeper

- [jQuery source viewer](#)
- [Array.prototype.join\(\)](#)
- [Array.prototype.map\(\)](#)
- [Array.prototype.reduce\(\)](#)
- [document.createDocumentFragment\(\)](#)
- [document.createTextNode\(\)](#)
- [Google Web Fundamentals - Performance](#)

Section 11.2: jQuery append

HTML

```
<p>This is a nice </p>
<p>I like </p>

<ul>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ul>

<button id="btn-1">Append text</button>
<button id="btn-2">Append list item</button>
```

Script

```
$("#btn-1").click(function(){
    $("p").append(" <b>Book</b>.");
});
$("#btn-2").click(function(){
    $("ul").append("<li>Appended list item</li>");
});
});
```

Section 11.3: Appending an element to a container

Solution 1:

```
$('#parent').append($('#child'));
```

方案2：

```
$('#child').appendTo($('#parent'));
```

这两种方案都是将元素#child（添加到末尾）附加到元素#parent。

之前：

```
<div id="parent">
  <span>其他内容</span>
</div>
<div id="child">

</div>
```

之后：

```
<div id="parent">
  <span>其他内容</span>
  <div id="child">
```

注意： 当你将文档中已存在的内容追加时，该内容将从其原始父容器中移除并追加到新的父容器中。因此，你不能使用 `.append()` 或 `.appendTo()` 来克隆元素。如果需要克隆，请使用 `.clone()` -> [\[http://api.jquery.com/clone/\]\[1\]](http://api.jquery.com/clone/)

Solution 2:

```
$('#child').appendTo($('#parent'));
```

Both solutions are appending the element #child (adding at the end) to the element #parent.

Before:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

After:

```
<div id="parent">
  <span>other content</span>
  <div id="child">

  </div>
</div>
```

Note: When you append content that already exists in the document, this content will be removed from its original parent container and appended to the new parent container. So you can't use `.append()` or `.appendTo()` to clone an element. If you need a clone use `.clone()` -> [\[http://api.jquery.com/clone/\]\[1\]](http://api.jquery.com/clone/)

第12章：前置插入

第12.1节：向容器前置插入元素

方案 1：

```
$('#parent').prepend($('#child'));
```

方案2：

```
$('#child').prependTo($('#parent'));
```

这两种方法都是将元素 #child 前置插入（添加到开头）到元素 #parent 中。

之前：

```
<div id="parent">
  <span>其他内容</span>
</div>
<div id="child">
  </div>
```

之后：

```
<div id="parent">
  <div id="child">
    </div>
  <span>其他内容</span>
</div>
```

第12.2节：prepend 方法

prepend() - 将参数指定的内容插入到匹配元素集合中每个元素的开头。

1.prepend(content[,content])

```
// 使用 HTML 字符串
jQuery('#parent').prepend('<span>child</span>');
// 或者你可以使用 jQuery 对象
jQuery('#parent').prepend($('#child'));
// 或者你可以使用逗号分隔的多个元素进行前置
jQuery('#parent').prepend($('#child1'),$('#child2'));
```

2.prepend(function)

jQuery 版本：从 1.4 开始，你可以使用回调函数作为参数。该函数可以获得的参数为元素在集合中的索引位置和元素的旧 HTML 值。在函数内部，this 指向集合中的当前元素。

```
jQuery('#parent').prepend(function(i,oldHTML){
  // 返回要前置的值
  return '<span>child</span>';
});
```

Chapter 12: Prepend

Section 12.1: Prepending an element to a container

Solution 1:

```
$('#parent').prepend($('#child'));
```

Solution 2:

```
$('#child').prependTo($('#parent'));
```

Both solutions are prepending the element #child (adding at the beginning) to the element #parent.

Before:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">
  </div>
```

After:

```
<div id="parent">
  <div id="child">
    </div>
  <span>other content</span>
</div>
```

Section 12.2: Prepend method

prepend() - Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

1.prepend(content [, content])

```
// with html string
jQuery('#parent').prepend('<span>child</span>');
// or you can use jQuery object
jQuery('#parent').prepend($('#child'));
// or you can use comma separated multiple elements to prepend
jQuery('#parent').prepend($('#child1'),$('#child2'));
```

2.prepend(function)

jQuery version: 1.4 onwards you can use callback function as the argument. Where you can get arguments as index position of the element in the set and the old HTML value of the element. Within the function, this refers to the current element in the set.

```
jQuery('#parent').prepend(function(i,oldHTML){
  // return the value to be prepend
  return '<span>child</span>';
});
```

```
});
```

belindoc.com

```
});
```


第13章：获取和设置元素的宽度和高度

第13.1节：获取和设置宽度和高度（忽略边框）

获取宽度和高度：

```
var width = $('#target-element').width();
var height = $('#target-element').height();
```

设置宽度和高度：

```
$('#target-element').width(50);
$('#target-element').height(100);
```

第13.2节：获取和设置innerWidth和innerHeight（忽略内边距和边框）

获取宽度和高度：

```
var width = $('#target-element').innerWidth();
var height = $('#target-element').innerHeight();
```

设置宽度和高度：

```
$('#target-element').innerWidth(50);
$('#target-element').innerHeight(100);
```

第13.3节：获取和设置outerWidth和outerHeight（包括内边距和边框）

获取宽度和高度（不包括外边距）：

```
var width = $('#target-element').outerWidth();
var height = $('#target-element').outerHeight();
```

获取宽度和高度（包括外边距）：

```
var width = $('#target-element').outerWidth(true);
var height = $('#target-element').outerHeight(true);
```

设置宽度和高度：

```
$('#target-element').outerWidth(50);
$('#target-element').outerHeight(100);
```

Chapter 13: Getting and setting width and height of an element

Section 13.1: Getting and setting width and height (ignoring border)

Get width and height:

```
var width = $('#target-element').width();
var height = $('#target-element').height();
```

Set width and height:

```
$('#target-element').width(50);
$('#target-element').height(100);
```

Section 13.2: Getting and setting innerWidth and innerHeight (ignoring padding and border)

Get width and height:

```
var width = $('#target-element').innerWidth();
var height = $('#target-element').innerHeight();
```

Set width and height:

```
$('#target-element').innerWidth(50);
$('#target-element').innerHeight(100);
```

Section 13.3: Getting and setting outerWidth and outerHeight (including padding and border)

Get width and height (excluding margin):

```
var width = $('#target-element').outerWidth();
var height = $('#target-element').outerHeight();
```

Get width and height (including margin):

```
var width = $('#target-element').outerWidth(true);
var height = $('#target-element').outerHeight(true);
```

Set width and height:

```
$('#target-element').outerWidth(50);
$('#target-element').outerHeight(100);
```

第14章：jQuery .animate() 方法

参数	详细信息
属性	动画将移动到的 CSS 属性和值的对象
持续时间	(默认值：400) 一个字符串或数字，决定动画运行的时长
缓动	(默认值：swing) 一个字符串，指示过渡时使用的缓动函数
完成	动画完成后调用的函数，每个匹配的元素调用一次。
开始	指定动画开始时执行的函数。
步骤	指定在动画的每一步执行的函数。
队列	一个布尔值，指定是否将动画放入效果队列中。
进度	指定在动画每一步之后执行的函数。
完成	指定动画结束时执行的函数。
失败	指定动画未能完成时执行的函数。
specialEasing	是一个映射，包含 styles 参数中的一个或多个 CSS 属性及其对应的缓动效果函数。
总是	指定如果动画在未完成时停止要执行的函数。

第14.1节：带回调的动画

有时我们需要将文字从一个位置移动到另一个位置，或缩小文字大小并自动改变文字颜色，以提升我们网站或网页应用的吸引力。JQuery 在这方面提供了很大帮助，使用fadeIn()、hide()、slideDown()等方法，但其功能有限，只能完成分配给它的特定任务。

Jquery 通过提供一个惊人且灵活的方法.animate()解决了这个问题。该方法允许设置自定义动画，使用允许超出边界的 CSS 属性。例如，如果我们给 CSS 样式属性设置为width:200;且 DOM 元素当前的位置是 50，animate 方法会将当前位置值从给定的 CSS 值中减去，并将该元素动画到 150。但我们无需担心这部分，因为动画引擎会处理它。

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script>
$("#btn1").click(function(){
    $("#box").animate({width: "200px"});
});
</script>

<button id="btn1">动画宽度</button>
<div id="box" style="background:#98bf21;height:100px;width:100px;margin:6px;"></div>
```

允许在.animate()方法中使用的 CSS 样式属性列表。

backgroundPositionX, backgroundPositionY, borderWidth, borderBottomWidth, borderLeftWidth, borderRightWidth, borderTopWidth, borderSpacing, margin, marginBottom, marginLeft, marginRight, marginTop, outlineWidth, padding, paddingBottom, paddingLeft, paddingRight, paddingTop, height, width, maxHeight, maxWidth, minHeight, minWidth, fontSize, bottom, left, right, top, letterSpacing, wordSpacing, lineHeight, textIndent,

在.animate()方法中指定的速度。

毫秒(例如: 100, 1000, 5000, 等等),
"slow",

Chapter 14: jQuery .animate() Method

Parameter	Details
properties	An object of CSS properties and values that the animation will move toward
duration	(default: 400) A string or number determining how long the animation will run
easing	(default: swing) A string indicating which easing function to use for the transition
complete	A function to call once the animation is complete, called once per matched element.
start	specifies a function to be executed when the animation begins.
step	specifies a function to be executed for each step in the animation.
queue	a Boolean value specifying whether or not to place the animation in the effects queue.
progress	specifies a function to be executed after each step in the animation.
done	specifies a function to be executed when the animation ends.
fail	specifies a function to be executed if the animation fails to complete.
specialEasing	a map of one or more CSS properties from the styles parameter, and their corresponding easing functions.
always	specifies a function to be executed if the animation stops without completing.

Section 14.1: Animation with callback

Sometimes we need to change words position from one place to another or reduce size of the words and change the color of words automatically to improve the attraction of our website or web apps. JQuery helps a lot with this concept using fadeIn(), hide(), slideDown() but its functionality are limited and it only done the specific task which assign to it.

Jquery fix this problem by providing an amazing and flexible method called .animate(). This method allows to set custom animations which is used css properties that give permission to fly over borders. for example if we give css style property as width:200; and current position of the DOM element is 50, animate method reduce current position value from given css value and animate that element to 150.But we don't need to bother about this part because animation engine will handle it.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script>
    $("#btn1").click(function(){
        $("#box").animate({width: "200px"});
    });
</script>

<button id="btn1">Animate Width</button>
<div id="box" style="background:#98bf21;height:100px;width:100px;margin:6px;"></div>
```

List of css style properties that allow in .animate() method.

backgroundPositionX, backgroundPositionY, borderWidth, borderBottomWidth, borderLeftWidth, borderRightWidth, borderTopWidth, borderSpacing, margin, marginBottom, marginLeft, marginRight, marginTop, outlineWidth, padding, paddingBottom, paddingLeft, paddingRight, paddingTop, height, width, maxHeight, maxWidth, minHeight, minWidth, fontSize, bottom, left, right, top, letterSpacing, wordSpacing, lineHeight, textIndent,

Speed specified in .animate() method.

milliseconds (Ex: 100, 1000, 5000, etc.),
"slow",

"fast"

在.animate()方法中指定的缓动效果。

"swing"
"linear"

以下是一些带有复杂动画选项的示例。

示例 1:

```
$( "#book" ).animate({
  width: [ "toggle", "swing" ],
  height: [ "toggle", "swing" ],
  opacity: "toggle"
}, 5000, "linear", function() {
  $( this ).after( "<div>动画完成。</div>" );
});
```

示例 2:

```
$( "#box" ).animate({
  height: "300px",
  width: "300px"
}, {
  duration: 5000,
  easing: "linear",
  complete: function(){
    $(this).after("<p>动画完成！</p>");
  }
});
```

"fast"

Easing specified in .animate() method.

"swing"
"linear"

Here is some examples with complex animation options.

Eg 1:

```
$( "#book" ).animate({
  width: [ "toggle", "swing" ],
  height: [ "toggle", "swing" ],
  opacity: "toggle"
}, 5000, "linear", function() {
  $( this ).after( "<div>Animation complete.</div>" );
});
```

Eg 2:

```
$( "#box" ).animate({
  height: "300px",
  width: "300px"
}, {
  duration: 5000,
  easing: "linear",
  complete: function(){
    $(this).after("<p>Animation is complete!</p>");
  }
});
```

第15章：jQuery Deferred 对象和 Promises（承诺）

jQuery 的 promises 是一种巧妙的方式，用于以构建模块的方式串联异步操作。这取代了传统的回调嵌套方式，后者不易重新组织。

第15.1节：jQuery ajax() 的 success、error 与 .done()、.fail()

success 和 error ： 一个在 Ajax 请求成功完成时调用的成功回调函数。

在请求过程中出现任何错误时调用的失败回调函数。

示例：

```
$.ajax({
  url: 'URL',
  type: 'POST',
  data: yourData,
  datatype: 'json',
  success: function (data) { successFunction(data); },
  error: function (jqXHR, textStatus, errorThrown) { errorFunction(); }
});
```

.done() 和 .fail() :

.ajax().done(function(data, textStatus, jqXHR){}); 替代了 jQuery 1.8 中已废弃的 .success() 方法。这是上述成功回调函数的另一种写法。

.ajax().fail(function(jqXHR, textStatus, errorThrown){}); 替代了 jQuery 1.8 中已废弃的 .error() 方法。这是上述完成回调函数的另一种写法。

示例：

```
$.ajax({
  url: 'URL',
  type: 'POST',
  data: yourData,
  datatype: 'json'
})
.done(function (data) { successFunction(data); })
.fail(function (jqXHR, textStatus, errorThrown) { errorFunction(); });
```

第15.2节：基本的Promise创建

这里有一个非常简单的函数示例，它“承诺在给定时间过去后继续执行”。它通过创建一个新的Deferred对象来实现，该对象稍后被解决，并返回Deferred的promise：

```
function waitPromise(毫秒){

  // 使用jQuery静态方法创建一个新的Deferred对象
  var def = $.Deferred();

  // 执行一些异步操作——在本例中是一个简单的定时器
  setTimeout(function(){
```

Chapter 15: jQuery Deferred objects and Promises

jQuery promises are a clever way of chaining together asynchronous operations in a building-block manner. This replaces old-school nesting of callbacks, which are not so easily reorganised.

Section 15.1: jQuery ajax() success, error VS .done(), .fail()

success and Error : A **success** callback that gets invoked upon successful completion of an Ajax request.

A **failure** callback that gets invoked in case there is any error while making the request.

Example:

```
$.ajax({
  url: 'URL',
  type: 'POST',
  data: yourData,
  datatype: 'json',
  success: function (data) { successFunction(data); },
  error: function (jqXHR, textStatus, errorThrown) { errorFunction(); }
});
```

.done() and .fail() :

.ajax().done(function(data, textStatus, jqXHR){}); Replaces method .success() which was deprecated in jQuery 1.8.This is an alternative construct for the success callback function above.

.ajax().fail(function(jqXHR, textStatus, errorThrown){}); Replaces method .error() which was deprecated in jQuery 1.8.This is an alternative construct for the complete callback function above.

Example:

```
$.ajax({
  url: 'URL',
  type: 'POST',
  data: yourData,
  datatype: 'json'
})
.done(function (data) { successFunction(data); })
.fail(function (jqXHR, textStatus, errorThrown) { errorFunction(); });
```

Section 15.2: Basic promise creation

Here is a very simple example of a function that "*promises* to proceed when a given time elapses". It does that by creating a new Deferred object, that is resolved later and returning the Deferred's promise:

```
function waitPromise(millisecons){

  // Create a new Deferred object using the jQuery static method
  var def = $.Deferred();

  // Do some asynchronous work - in this case a simple timer
  setTimeout(function(){
```

```
    // 工作完成.....解决Deferred, 使其promise继续执行
    def.resolve();
    }, 毫秒);

    // 立即返回一个“等待时间结束时继续执行的承诺”
    return def.promise();
}
```

并且这样使用：

```
waitPromise(2000).then(function(){
    console.log("我已经等待够久了");
});
```

```
    // Work completed... resolve the deferred, so it's promise will proceed
    def.resolve();
    }, milliseconds);

    // Immediately return a "promise to proceed when the wait time ends"
    return def.promise();
}
```

And use like this:

```
waitPromise(2000).then(function(){
    console.log("I have waited long enough");
});
```


第16章：Ajax

参数	详情
url	指定请求将发送到的URL
设置	一个包含众多影响请求行为的值的对象
类型	请求中要使用的HTTP方法
数据	请求发送的数据
成功	请求成功时调用的回调函数
错误	处理错误的回调函数
statusCode	一个包含数字HTTP状态码及对应状态码响应时调用函数的对象
数据类型	你期望从服务器返回的数据类型
contentType	发送到服务器的数据的内容类型。默认是 "application/x-www-form-urlencoded; charset=UTF-8"
context	指定回调函数中使用的上下文，通常是this，指当前目标。

第16.1节：使用 \$.ajax() 处理 HTTP 响应代码

除了基于请求是否成功触发的.done、.fail和.always承诺回调之外，还可以在服务器返回特定的HTTP 状态码时触发函数。这可以通过statusCode参数实现。

```
$.ajax({
  type: {POST 或 GET 或 PUT 等},
  url: {server.url},
  data: {someData: true},
  statusCode: {
    404: function(responseObject, textStatus, jqXHR) {
      // 未找到内容 (404)
      // 如果服务器返回404响应, 将执行此代码
    },
    503: function(responseObject, textStatus, errorThrown) {
      // 服务不可用 (503)
      // 如果服务器返回503响应, 将执行此代码
    }
  }
})
.done(function(data){
  alert(data);
})
.fail(function(jqXHR, textStatus){
  alert('出现错误：' + textStatus);
})
.always(function(jqXHR, textStatus) {
  alert('Ajax请求已完成')
});
```

正如官方jQuery文档所述：

如果请求成功，状态码函数接受与成功回调相同的参数；如果请求出错（包括3xx重定向），则接受与错误回调相同的参数。

Chapter 16: Ajax

Parameter	Details
url	Specifies the URL to which the request will be sent
settings	an object containing numerous values that affect the behavior of the request
type	The HTTP method to be used for the request
data	Data to be sent by the request
success	A callback function to be called if the request succeeds
error	A callback to handle error
statusCode	An object of numeric HTTP codes and functions to be called when the response has the corresponding code
dataType	The type of data that you're expecting back from the server
contentType	Content type of the data to sent to the server. Default is "application/x-www-form-urlencoded; charset=UTF-8"
context	Specifies the context to be used inside callbacks, usually this which refers to the current target.

Section 16.1: Handling HTTP Response Codes with \$.ajax()

In addition to .done, .fail and .always promise callbacks, which are triggered based on whether the request was successful or not, there is the option to trigger a function when a specific [HTTP Status Code](#) is returned from the server. This can be done using the statusCode parameter.

```
$.ajax({
  type: {POST or GET or PUT etc.},
  url: {server.url},
  data: {someData: true},
  statusCode: {
    404: function(responseObject, textStatus, jqXHR) {
      // No content found (404)
      // This code will be executed if the server returns a 404 response
    },
    503: function(responseObject, textStatus, errorThrown) {
      // Service Unavailable (503)
      // This code will be executed if the server returns a 503 response
    }
  }
})
.done(function(data){
  alert(data);
})
.fail(function(jqXHR, textStatus){
  alert('Something went wrong: ' + textStatus);
})
.always(function(jqXHR, textStatus) {
  alert('Ajax request was finished')
});
```

As official jQuery documentation states:

If the request is successful, the status code functions take the same parameters as the success callback; if it results in an error (including 3xx redirect), they take the same parameters as the **error** callback.

第16.2节：使用Ajax提交表单

有时你可能有一个表单，想用ajax提交它。

假设你有这样一个简单的表单 -

```
<form id="ajax_form" action="form_action.php">
  <label for="name">姓名 :</label>
  <input name="name" id="name" type="text" />
  <label for="name">电子邮件 :</label>
  <input name="email" id="email" type="text" />
  <input type="submit" value="提交" />
</form>
```

以下jQuery代码可用（在\$(document).ready调用内） -

```
$( '#ajax_form' ).submit(function(event){
  event.preventDefault();
  var $form = $(this);

  $.ajax({
type: 'POST',
url: $form.attr('action'),
data: $form.serialize(),
success: function(data) {
  // 处理响应
},
error: function(error) {
  // 处理错误
}
});
});
```

说明

- var \$form = \$(this) - 表单，缓存以便重用
- \$('#ajax_form').submit(function(event){ - 当ID为 "ajax_form" 的表单提交时运行此函数，并将事件作为参数传入。
- event.preventDefault(); - 阻止表单正常提交（或者我们可以在 ajax({}); 语句后使用 return false, 效果相同）
- url: \$form.attr('action'), - 获取表单的 "action" 属性值，并用作 "url" 属性。
- data: \$form.serialize(), - 将表单内的输入转换为适合发送到服务器的字符串。在此例中，它将返回类似 "name=Bob&email=bob@bobsemailaddress.com" 的内容

第16.3节：综合示例

Ajax 获取：

方案 1：

```
$.get('url.html', function(data){
  $( '#update-box' ).html(data);
});
```

方案2：

Section 16.2: Using Ajax to Submit a Form

Sometimes you may have a form and want to submit it using ajax.

Suppose you have this simple form -

```
<form id="ajax_form" action="form_action.php">
  <label for="name">Name :</label>
  <input name="name" id="name" type="text" />
  <label for="name">Email :</label>
  <input name="email" id="email" type="text" />
  <input type="submit" value="Submit" />
</form>
```

The following jQuery code can be used (within a \$(document).ready call) -

```
$( '#ajax_form' ).submit(function(event){
  event.preventDefault();
  var $form = $(this);

  $.ajax({
    type: 'POST',
    url: $form.attr('action'),
    data: $form.serialize(),
    success: function(data) {
      // Do something with the response
    },
    error: function(error) {
      // Do something with the error
    }
  });
});
```

Explanation

- var \$form = \$(this) - the form, cached for reuse
- \$('#ajax_form').submit(function(event){ - When the form with ID "ajax_form" is submitted run this function and pass the event as a parameter.
- event.preventDefault(); - Prevent the form from submitting normally (Alternatively we can use return false after the ajax({}); statement, which will have the same effect)
- url: \$form.attr('action'), - Get the value of the form's "action" attribute and use it for the "url" property.
- data: \$form.serialize(), - Converts the inputs within the form into a string suitable for sending to the server. In this case it will return something like "name=Bob&email=bob@bobsemailaddress.com"

Section 16.3: All in one examples

Ajax Get:

Solution 1:

```
$.get('url.html', function(data){
  $( '#update-box' ).html(data);
});
```

Solution 2:

```
$.ajax({
  type: 'GET',
  url: 'url.php',
}).done(function(data){
  $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
  alert('Error occurred: ' + textStatus);
});
```

Ajax 加载： 另一个为了简便而创建的 ajax get 方法

```
$('#update-box').load('url.html');
```

.load 也可以带额外的数据调用。数据部分可以是字符串或对象。

```
$('#update-box').load('url.php', {data: "something"});
$('#update-box').load('url.php', "data=something");
```

如果 .load 带有回调方法调用，向服务器的请求将是 post 请求

```
$('#update-box').load('url.php', {data: "something"}, function(resolve){
  //do something
});
```

Ajax Post:

方案 1：

```
$.post('url.php',
  {date1Name: data1Value, date2Name: data2Value}, //要发送的数据
  function(data){
    $('#update-box').html(data);
  }
);
```

方案2：

```
$.ajax({
  type: 'Post',
  url: 'url.php',
  data: {date1Name: data1Value, date2Name: data2Value} //要发送的数据
}).done(function(data){
  $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
  alert('发生错误: ' + textStatus);
});
```

Ajax Post JSON:

```
var postData = {
  姓名: name,
  地址: address,
  电话: phone
};

$.ajax({
  类型: "POST",
  地址: "url.php",
```

```
$.ajax({
  type: 'GET',
  url: 'url.php',
}).done(function(data){
  $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
  alert('Error occurred: ' + textStatus);
});
```

Ajax Load: Another ajax get method created for simplicity

```
$('#update-box').load('url.html');
```

.load can also be called with additional data. The data part can be provided as string or object.

```
$('#update-box').load('url.php', {data: "something"});
$('#update-box').load('url.php', "data=something");
```

If .load is called with a callback method, the request to the server will be a post

```
$('#update-box').load('url.php', {data: "something"}, function(resolve){
  //do something
});
```

Ajax Post:

Solution 1:

```
$.post('url.php',
  {date1Name: data1Value, date2Name: data2Value}, //data to be posted
  function(data){
    $('#update-box').html(data);
  }
);
```

Solution 2:

```
$.ajax({
  type: 'Post',
  url: 'url.php',
  data: {date1Name: data1Value, date2Name: data2Value} //data to be posted
}).done(function(data){
  $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
  alert('Error occurred: ' + textStatus);
});
```

Ajax Post JSON:

```
var postData = {
  Name: name,
  Address: address,
  Phone: phone
};

$.ajax({
  type: "POST",
  url: "url.php",
```

```
数据类型: "json",
    数据: JSON.stringify(postData),
    成功: 函数 (data) {
        //这里变量 data 是 JSON 格式
    }
});
```

Ajax 获取 JSON：

方案 1：

```
$.getJSON('url.php', 函数(data){
    //这里变量 data 是 JSON 格式
});
```

方案2：

```
$.ajax({
    类型: "Get",
    地址: "url.php",
    数据类型: "json",
    数据: JSON.stringify(postData),
    成功: 函数 (data) {
        //这里变量 data 是 JSON 格式
    },
    error: function(jqXHR, textStatus){
        alert('发生错误: ' + textStatus);
    }
});
```

第16.4节：Ajax文件上传

1. 一个简单完整的示例

我们可以使用此示例代码在每次用户选择新文件时上传所选文件。

```
<input type="file" id="file-input" multiple>
```

```
var files;
var fdata = new FormData();
$("#file-input").on("change", function (e) {
    files = this.files;

    $.each(files, function (i, file) {
        fdata.append("file" + i, file);
    });

    fdata.append("FullName", "John Doe");
    fdata.append("Gender", "Male");
    fdata.append("Age", "24");

    $.ajax({
        url: "/Test/Url",
        type: "post",
        data: fdata, //将 FormData 对象添加到 data 参数中
        processData: false, //告诉 jquery 不要处理数据
    });
});
```

```
dataType: "json",
data: JSON.stringify(postData),
success: function (data) {
    //here variable data is in JSON format
}
});
```

Ajax Get JSON:

Solution 1:

```
$.getJSON('url.php', function(data){
    //here variable data is in JSON format
});
```

Solution 2:

```
$.ajax({
    type: "Get",
    url: "url.php",
    dataType: "json",
    data: JSON.stringify(postData),
    success: function (data) {
        //here variable data is in JSON format
    },
    error: function(jqXHR, textStatus){
        alert('Error occurred: ' + textStatus);
    }
});
```

Section 16.4: Ajax File Uploads

1. A Simple Complete Example

We could use this sample code to upload the files selected by the user every time a new file selection is made.

```
<input type="file" id="file-input" multiple>
```

```
var files;
var fdata = new FormData();
$("#file-input").on("change", function (e) {
    files = this.files;

    $.each(files, function (i, file) {
        fdata.append("file" + i, file);
    });

    fdata.append("FullName", "John Doe");
    fdata.append("Gender", "Male");
    fdata.append("Age", "24");

    $.ajax({
        url: "/Test/Url",
        type: "post",
        data: fdata, //add the FormData object to the data parameter
        processData: false, //tell jquery not to process data
    });
});
```

```
contentType: false, //告诉 jquery 不要设置 content-type
success: function (response, status, jqxhr) {
    //处理成功
},
error: function (jqxhr, status, errorMessage) {
    //处理错误
}
});
```

现在让我们逐部分拆解并检查它。

2. 处理文件输入

这篇MDN 文档（从 Web 应用程序使用文件）是关于如何处理文件输入的各种方法的好读物。其中一些方法也将在本示例中使用。

在我们开始上传文件之前，首先需要给用户一个选择他们想上传的文件的方式。为此，我们将使用一个文件输入。multiple属性允许选择多个文件，如果你希望用户一次只选择一个文件，可以去掉它。

```
<input type="file" id="file-input" multiple>
```

我们将使用输入的change事件来捕获文件。

```
var files;
$("#file-input").on("change", function(e){
    files = this.files;
});
```

在处理函数内部，我们通过输入框的 files 属性访问文件。这会给我们一个 FileList，它是一个类似数组的对象。

3. 创建并填充 FormData

为了使用 Ajax 上传文件，我们将使用 FormData。

```
var fdata = new FormData();
```

在上一步中获得的 FileList 是一个类似数组的对象，可以使用多种方法遍历，包括 for 循环、for...of 循环 和 jQuery.each。本例中我们将使用 jQuery。

```
$.each(files, function(i, file) {
    //...
});
```

我们将使用 FormData 的 append 方法将文件添加到我们的 formdata 对象中。

```
$.each(files, function(i, file) {
    fdata.append("file" + i, file);
});
```

我们也可以用同样的方式添加其他想要发送的数据。比如说，我们想要将从用户那里获得的一些个人信息和文件一起发送，就可以将这些信息添加到我们的 formdata 对象中。

```
fdata.append("FullName", "约翰·多伊");
```

```
contentType: false, //tell jquery not to set content-type
success: function (response, status, jqxhr) {
    //handle success
},
error: function (jqxhr, status, errorMessage) {
    //handle error
}
});
```

Now let's break this down and inspect it part by part.

2. Working With File Inputs

This MDN Document (Using files from web applications) is a good read about various methods on how to handle file inputs. Some of these methods will also be used in this example.

Before we get to uploading files, we first need to give the user a way to select the files they want to upload. For this purpose we will use a file input. The multiple property allows for selecting more than one files, you can remove it if you want the user to select one file at a time.

```
<input type="file" id="file-input" multiple>
```

We will be using input's change event to capture the files.

```
var files;
$("#file-input").on("change", function(e){
    files = this.files;
});
```

Inside the handler function, we access the files through the files property of our input. This gives us a FileList, which is an array like object.

3. Creating and Filling the FormData

In order to upload files with Ajax we are going to use FormData.

```
var fdata = new FormData();
```

FileList we have obtained in the previous step is an array like object and can be iterated using various methods including for loop, for...of loop and jQuery.each. We will be sticking with the jQuery in this example.

```
$.each(files, function(i, file) {
    //...
});
```

We will be using the append method of FormData to add the files into our formdata object.

```
$.each(files, function(i, file) {
    fdata.append("file" + i, file);
});
```

We can also add other data we want to send the same way. Let's say we want to send some personal information we have received from the user along with the files. We could add this this information into our formdata object.

```
fdata.append("FullName", "John Doe");
```



```
fdata.append("Gender", "男");
fdata.append("Age", "24");
//...
```

4. 使用 Ajax 发送文件

```
$.ajax({
url: "/Test/Url",
type: "post",
  data: fdata, //将 FormData 对象添加到 data 参数中
  processData: false, //告诉 jquery 不要处理数据
  contentType: false, //告诉 jquery 不要设置内容类型
  success: function (response, status, jqxhr) {
    //处理成功
  },
  error: function (jqxhr, status, errorMessage) {
    //处理错误
  }
});
```

我们将 processData 和 contentType 属性设置为 **false**。这样做是为了让文件能够发送到服务器并被服务器正确处理。

```
fdata.append("Gender", "Male");
fdata.append("Age", "24");
//...
```

4. Sending the Files With Ajax

```
$.ajax({
  url: "/Test/Url",
  type: "post",
  data: fdata, //add the FormData object to the data parameter
  processData: false, //tell jquery not to process data
  contentType: false, //tell jquery not to set content-type
  success: function (response, status, jqxhr) {
    //handle success
  },
  error: function (jqxhr, status, errorMessage) {
    //handle error
  }
});
```

We set processData and contentType properties to **false**. This is done so that the files can be send to the server and be processed by the server correctly.

第17章：复选框全选，自动根据其他复选框的变化进行选中/取消选中

我使用了各种Stackoverflow的示例和答案，得出了这个非常简单的示例，说明如何管理“全选”复选框，并在组内任意复选框状态变化时自动勾选或取消勾选。
约束条件：“全选”ID必须与输入名称匹配，以创建全选组。在示例中，输入的全选ID是cbGroup1。输入名称也为cbGroup1

代码非常简短，没有大量的if语句（节省时间和资源）。

第17.1节：2个全选复选框及对应的分组复选框

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<p>
<input id="cbGroup1" type="checkbox">全选
<input name="cbGroup1" type="checkbox" value="value1_1">分组1 值 1
<input name="cbGroup1" type="checkbox" value="value1_2">分组1 值 2
<input name="cbGroup1" type="checkbox" value="value1_3">分组1 值 3
</p>
<p>
<input id="cbGroup2" type="checkbox">全选
<input name="cbGroup2" type="checkbox" value="value2_1">分组2 值 1
<input name="cbGroup2" type="checkbox" value="value2_2">分组2 值 2
<input name="cbGroup2" type="checkbox" value="value2_3">分组2 值 3
</p>

<script type="text/javascript" language="javascript">
    $("input").change(function() {
    $('input[name="'+this.id+'"]').not(this).prop('checked', this.checked);
        $('#'+this.name).prop('checked', $('input[name="'+this.name+'"]').length ===
    $('input[name="'+this.name+'"]').filter(':checked').length);
    });
</script>
```

Chapter 17: Checkbox Select all with automatic check/uncheck on other checkbox change

I've used various Stackoverflow examples and answers to come to this really simple example on how to manage "select all" checkbox coupled with an automatic check/uncheck if any of the group checkbox status changes.
Constraint: The "select all" id must match the input names to create the select all group. In the example, the input select all ID is cbGroup1. The input names are also cbGroup1

Code is very short, not plenty of if statement (time and resource consuming).

Section 17.1: 2 select all checkboxes with corresponding group checkboxes

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<p>
<input id="cbGroup1" type="checkbox">Select all
<input name="cbGroup1" type="checkbox" value="value1_1">Group1 value 1
<input name="cbGroup1" type="checkbox" value="value1_2">Group1 value 2
<input name="cbGroup1" type="checkbox" value="value1_3">Group1 value 3
</p>
<p>
<input id="cbGroup2" type="checkbox">Select all
<input name="cbGroup2" type="checkbox" value="value2_1">Group2 value 1
<input name="cbGroup2" type="checkbox" value="value2_2">Group2 value 2
<input name="cbGroup2" type="checkbox" value="value2_3">Group2 value 3
</p>

<script type="text/javascript" language="javascript">
    $("input").change(function() {
        $('input[name="'+this.id+'"]').not(this).prop('checked', this.checked);
        $('#'+this.name).prop('checked', $('input[name="'+this.name+'"]').length ===
    $('input[name="'+this.name+'"]').filter(':checked').length);
    });
</script>
```

第18章：插件

第18.1节：插件 - 入门

可以通过向jQuery的原型添加内容来扩展jQuery API。例如，现有的API已经有许多可用的函数，如.hide()、.fadeIn()、.hasClass()等。

jQuery原型通过\$.fn暴露，源码中包含以下代码行

```
jQuery.fn = jQuery.prototype
```

向该原型添加函数将使这些函数可以从任何构造的jQuery对象调用（每次调用jQuery或调用\$时都会隐式完成构造）。

构造的jQuery对象将根据传入的选择器持有一个内部元素数组。例如，\$('.active')将构造一个jQuery对象，该对象持有调用时具有active类的元素（即，这不是一个实时的元素集合）。

插件函数内部的this值将指向构造的jQuery对象。因此，this用于表示匹配的集合。

基础插件：

```
$.fn.highlight = function() {
    this.css({ background: "yellow" });
};

// 使用示例：
$("span").highlight();
```

[jsFiddle 示例](#)

链式调用性与可重用性

与上面的示例不同，jQuery 插件预期是支持链式调用的。这意味着可以对同一元素集合链式调用多个方法，例如 \$(".warn").append("WARNING! ").css({color:"red"})（见我们如何在.append()之后使用.css()方法，两个方法都作用于同一个.warn集合）

允许在不同集合上使用同一个插件并传入不同的自定义选项，在自定义 / 可重用性方面起着重要作用

```
(function($) {
    $.fn.highlight = function( custom ) {

        // 默认设置
        var settings = $.extend({
            color : "", // 默认使用当前文本颜色
            background : "yellow" // 默认黄色背景
        }, custom);

        return this.css({ // `return this` 保持方法链式调用
            color : settings.color,
```

Chapter 18: Plugins

Section 18.1: Plugins - Getting Started

The jQuery API may be extended by adding to its prototype. For example, the existing API already has many functions available such as .hide(), .fadeIn(), .hasClass(), etc.

The jQuery prototype is exposed through \$.fn, the source code contains the line

```
jQuery.fn = jQuery.prototype
```

Adding functions to this prototype will allow those functions to be available to be called from any constructed jQuery object (which is done implicitly with each call to jQuery, or each call to \$ if you prefer).

A constructed jQuery object will hold an internal array of elements based on the selector passed to it. For example, \$('.active') will construct a jQuery object that holds elements with the active class, at the time of calling (as in, this is not a live set of elements).

The this value inside of the plugin function will refer to the constructed jQuery object. As a result, this is used to represent the matched set.

Basic Plugin:

```
$.fn.highlight = function() {
    this.css({ background: "yellow" });
};

// Use example:
$("span").highlight();
```

[jsFiddle example](#)

Chainability & Reusability

Unlike the example above, jQuery Plugins are expected to be Chainable. What this means is the possibility to chain multiple Methods to a same Collection of Elements like \$(".warn").append("WARNING! ").css({color:"red"}) (see how we used the .css() method after the .append(), both methods apply on the same .warn Collection)

Allowing one to use the same plugin on different Collections passing different customization options plays an important role in Customization / Reusability

```
(function($) {
    $.fn.highlight = function( custom ) {

        // Default settings
        var settings = $.extend({
            color : "", // Default to current text color
            background : "yellow" // Default to yellow background
        }, custom);

        return this.css({ // `return this` maintains method chainability
            color : settings.color,
```

```
backgroundColor : settings.background
});

};
}( jQuery ));

// 使用默认设置
$("span").highlight();    // 你可以链式调用其他方法

// 使用自定义设置
$("span").highlight({
    background: "#f00",
    color: "white"
});
```

[jsFiddle 演示](#)

自由

以上示例属于理解基本插件创建的范畴。请记住，不要限制用户只能使用有限的自定义选项。

例如，假设你想构建一个.highlight()插件，可以传入一个想要高亮显示的text字符串，并允许在样式方面拥有最大的自由度：

```
//...
// 默认设置
var settings = $.extend({
    text : "",           // 要高亮的文本
    class : "highlight" // CSS类的引用
}, custom);

return this.each(function() {
    // 你的单词高亮逻辑写在这里
});
//...
```

用户现在可以传入想要的text，并通过使用自定义的CSS类完全控制添加的样式：

```
$("#content").highlight({
    text : "hello",
    class : "makeYellowBig"
});
```

[jsFiddle 示例](#)

```
        backgroundColor : settings.background
    });

    };
}( jQuery ));

// Use Default settings
$("span").highlight();    // you can chain other methods

// Use Custom settings
$("span").highlight({
    background: "#f00",
    color: "white"
});
```

[jsFiddle demo](#)

Freedom

The above examples are in the scope of understanding basic Plugin creation. Keep in mind to not restrict a user to a limited set of customization options.

Say for example you want to build a .highlight() Plugin where you can pass a desired **text** String that will be highlighted and allow maximal freedom regarding styles:

```
//...
// Default settings
var settings = $.extend({
    text : "",           // text to highlight
    class : "highlight" // reference to CSS class
}, custom);

return this.each(function() {
    // your word highlighting logic here
});
//...
```

the user can now pass a desired **text** and have complete control over the added styles by using a custom CSS class:

```
$("#content").highlight({
    text : "hello",
    class : "makeYellowBig"
});
```

[jsFiddle example](#)

鸣谢

非常感谢所有来自Stack Overflow Documentation的人员，他们帮助提供了这些内容，更多更改可以发送至web@petercv.com，以发布或更新新内容

A.J	第1章、第4章和第8章
acdcjunior	第1章和第4章
亚历克斯	第15章
亚历克斯·查尔	第10章
阿隆·艾坦	第5章
amflare	第1章和第5章
安德鲁·布鲁克	第16章
阿尼尔	第1章
阿伦·普拉萨德·E·S	第16章
阿希库扎曼	第15章
阿什坎·莫拜恩·基亚巴尼	第9、11、12、13和16章
Assimilater	第7章
阿塔福德	第16章
ban17	第4章
本·H	第16章
比蓬	第3章和第11章
布兰特·索洛维伊	第9章
布洛克·戴维斯	第7章
bwegs	第1章
卡斯特罗·罗伊	第二章
charlietfl	第6章
csbarnes	第16章
达尔沙克	第11章
大卫	第二章
DefyGravity	第7章
DelightedD0D	第二章
Deryck	第7章和第11章
德夫林·卡内特	第二章
dlsso	第8章
J. 特斯廷顿博士	第16章
伊曼纽尔·温蒂尔ă	第5章
经验的	第11章和第12章
Flyer53	第11章
咖啡驱动	第1章
去编码了	第6章和第15章
哈桑	第2章和第18章
霍斯特·雅恩斯	第6章
冰人	第二章
伊戈尔·劳什	第1章
J.F	第5章
j08691	第9章
贾特尼尔·普林斯卢	第6章
jkdev	第1章和第5章
JLF	第二章
约翰·C	第1章和第16章
约翰·斯莱格斯	第二章
乔纳森·米查利克	第9章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content, more changes can be sent to web@petercv.com for new content to be published or updated

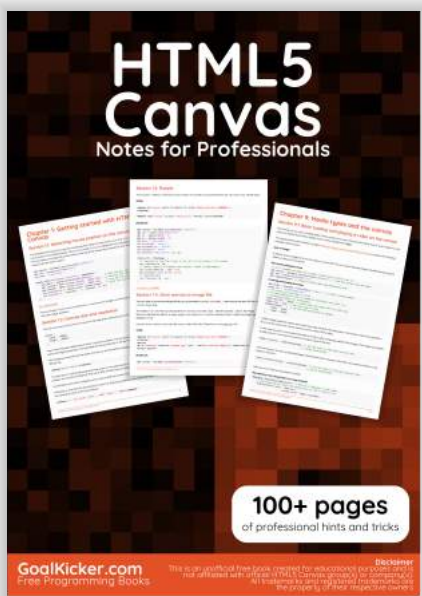
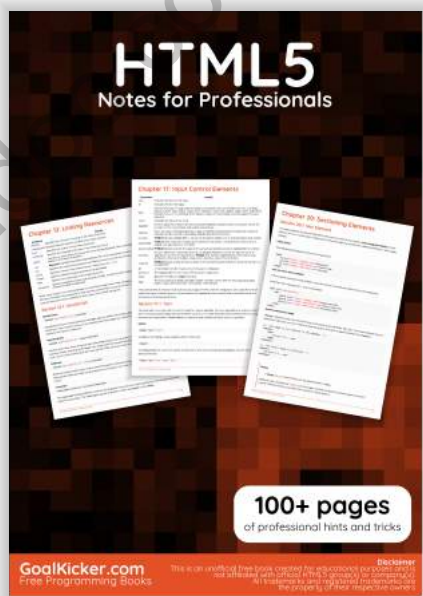
A.J	Chapters 1, 4 and 8
acdcjunior	Chapters 1 and 4
Alex	Chapter 15
Alex Char	Chapter 10
Alon Eitan	Chapter 5
amflare	Chapters 1 and 5
Andrew Brooke	Chapter 16
Anil	Chapter 1
Arun Prasad E S	Chapter 16
Ashiquzzaman	Chapter 15
Ashkan Mobayen Khiabani	Chapters 9, 11, 12, 13 and 16
Assimilater	Chapter 7
Athafoud	Chapter 16
ban17	Chapter 4
Ben H	Chapter 16
Bipon	Chapters 3 and 11
Brandt Solovij	Chapter 9
Brock Davis	Chapter 7
bwegs	Chapter 1
Castro Roy	Chapter 2
charlietfl	Chapter 6
csbarnes	Chapter 16
Darshak	Chapter 11
David	Chapter 2
DefyGravity	Chapter 7
DelightedD0D	Chapter 2
Deryck	Chapters 7 and 11
devlin carnate	Chapter 2
dlsso	Chapter 8
Dr. J. Testington	Chapter 16
Emanuel Vintilă	Chapter 5
empiric	Chapters 11 and 12
Flyer53	Chapter 11
Fueled By Coffee	Chapter 1
Gone Coding	Chapters 6 and 15
hasan	Chapters 2 and 18
Horst Jahns	Chapter 6
Iceman	Chapter 2
Igor Raush	Chapter 1
J.F	Chapter 5
j08691	Chapter 9
Jatniel Prinsloo	Chapter 6
jkdev	Chapters 1 and 5
JLF	Chapter 2
John C	Chapters 1 and 16
John Slegers	Chapter 2
Jonathan Michalik	Chapter 9

约拉姆·范登·博泽姆	第5章
卡潘扎克	第二章
凯文·卡茨克	第1章
基斯林格	第二章
拉克里奥克	第16章
利亚姆	第5章
卢卡·普楚	第1章和第6章
马克·舒尔特海斯	第5章和第7章
mark.hch	第8章
martincarlin87	第7章
马塔斯·瓦伊特凯维休斯	第1章
梅拉妮	第二章
莫蒂	第1章
尼尔	第1章
南	第5章
ni8mr	第1章
尼科·韦斯特代尔	第5章
尼拉夫·乔希	第16章
NotJustin	第6章
纳克斯	第二章
奥奇	第4章
奥赞	第16章
普拉纳夫·C·巴兰	第12章
原型	第11章
雷尼尔	第3章
阿蒙德席尔瓦	第8章
罗科·C·布尔扬	第1、9和18章
鲁帕利·佩马雷	第10章
斯基蒙斯特	第4和5章
塞塞莱特	第5章
SGS 文卡特什	第8章
肖纳克·D	第1、2和16章
谢卡尔·潘卡杰	第二章
施洛米·哈弗	第9章
简图	第14章
索朗瓦拉·阿巴萨利	第2和9章
ssb	第二章
still_learning	第7章
sucil	第8章
苏加尼亚	第1章
桑尼·R·古普塔	第6章
斯韦里·M·奥尔森	第8章
TheDeadMedic	第5章
西奥多·K.	第10章
The_Outsider	第8章和第10章
特拉维斯J	第1章、第2章和第18章
乌帕尔·罗伊	第5章
user1851673	第17章
user2314737	第10章
VJS	第14章
华盛顿·盖德斯	第6章
受伤的史蒂文·琼斯	第二章
约斯维尔·昆特罗	第1章和第16章

Joram van den Boezem	Chapter 5
kapantzak	Chapter 2
Kevin Katzke	Chapter 1
Keyslinger	Chapter 2
Lacrioque	Chapter 16
Liam	Chapter 5
Luca Putzu	Chapters 1 and 6
Mark Schultheiss	Chapters 5 and 7
mark.hch	Chapter 8
martincarlin87	Chapter 7
Matas Vaitkevicius	Chapter 1
Melanie	Chapter 2
Mottie	Chapter 1
Neal	Chapter 1
Nhan	Chapter 5
ni8mr	Chapter 1
Nico Westerdale	Chapter 5
Nirav Joshi	Chapter 16
NotJustin	Chapter 6
Nux	Chapter 2
ochi	Chapter 4
Ozan	Chapter 16
Pranav C Balan	Chapter 12
Proto	Chapter 11
Renier	Chapter 3
rmondesilva	Chapter 8
Roko C. Buljan	Chapters 1, 9 and 18
Rupali Pemare	Chapter 10
Scimonster	Chapters 4 and 5
secelite	Chapter 5
SGS Venkatesh	Chapter 8
Shaunak D	Chapters 1, 2 and 16
Shekhar Pankaj	Chapter 2
Shlomi Haver	Chapter 9
Simplans	Chapter 14
Sorangwala Abbasali	Chapters 2 and 9
ssb	Chapter 2
still_learning	Chapter 7
sucil	Chapter 8
Suganya	Chapter 1
Sunny R Gupta	Chapter 6
Sverri M. Olsen	Chapter 8
TheDeadMedic	Chapter 5
Theodore K.	Chapter 10
The_Outsider	Chapters 8 and 10
TravisJ	Chapters 1, 2 and 18
Upal Roy	Chapter 5
user1851673	Chapter 17
user2314737	Chapter 10
VJS	Chapter 14
Washington Guedes	Chapter 6
WOUNDEDStevenJones	Chapter 2
Yosvel Quintero	Chapters 1 and 16

belindoc.com

你可能也喜欢



You may also like

