jQuery1.8.0帮助文档

12xue研发

速查表

核心

- jQuery 核心函数
- [jQuery([sel,[context]])](#)
- jQuery(html,[ownerDoc])1.8*
- jQuery(callback)
- jQuery.holdReady(hold)1.6+
- jQuery 对象访问
- each(callback)
- size()
- length
- selector
- context
- get([index])
- index([selector|element])
- 数据缓存
- data([key],[value])
- removeData([name|list])1.7*
- ~~$.data(ele,[key],[val])~~1.8-
- 队列控制
- queue(element,[queueName])
- dequeue([queueName])
- clearQueue([queueName])
- 插件机制

- jQuery.fn.extend(object)
- jQuery.extend(object)

- 多库共存

- jQuery.noConflict([ex])

属性

- 属性

- attr(name|pro|key,val|fn)
- removeAttr(name)
- prop(name|pro|key,val|fn)1.6+
- removeProp(name)1.6+

- CSS 类

- addClass(class|fn)
- removeClass([class|fn])
- toggleClass(class|fn[,sw])

- HTML代码/文本/值

- html([val|fn])
- text([val|fn])
- val([val|fn|arr])

CSS

- CSS

- css(name|pro|[,val|fn])1.8*

- 位置

- offset([coordinates])
- position()
- scrollTop([val])

- scrollLeft([val])

- 尺寸

- heigh([val|fn])
- width([val|fn])
- innerHeight()
- innerWidth()
- outerHeight([soptions])
- outerWidth([options])

选择器

- 基本

- #id
- element
- .class
- *
- selector1,selector2,selectorN

- 层级

- ancestor descendant
- parent > child
- prev + next
- prev ~ siblings

- 基本

- :first
- :last
- :not(selector)
- :even

- :odd
- :eq(index)
- :gt(index)
- :lt(index)
- :header
- :animated
- :focus1.6+

- 内容

- :contains(text)
- :empty
- :has(selector)
- :parent

- 可见性

- :hidden
- :visible

- 属性

- [attribute]
- [attribute=value]
- [attribute!=value]
- [attribute^=value]
- [attribute$=value]
- [attribute*=value]
- [attrSel1][attrSel2][attrSelN]

- 子元素

- :nth-child
- :first-child

- :last-child
- :only-child

表单

- :input
- :text
- :password
- :radio
- :checkbox
- :submit
- :image
- :reset
- :button
- :file
- :hidden

表单对象属性

- :enabled
- :disabled
- :checked
- :selected

文档处理

内部插入

- append(content|fn)
- appendTo(content)
- prepend(content|fn)
- prependTo(content)

- 外部插入
- after(content|fn)
- before(content|fn)
- insertAfter(content)
- insertBefore(content)
- 包裹
- wrap(html|ele|fn)
- unwrap()
- wrapall(html|ele)
- wrapInner(html|ele|fn)
- 替换
- replaceWith(content|fn)
- replaceAll(selector)
- 删除
- empty()
- remove([expr])
- detach([expr])
- 复制
- [clone([Even[,deepEven]])](#)

筛选

- 过滤
- eq(index|-index)
- first()
- last()
- hasClass(class)

- filter(expr|obj|ele|fn)
- is(expr|obj|ele|fn)1.6*
- map(callback)
- has(expr|ele)
- not(expr|ele|fn)
- slice(start,[end])

查找

- children([expr])
- closest(expr,[con]|obj|ele)1.6*
- find(expr|obj|ele)1.6*
- next([expr])
- nextall([expr])
- nextUntil([exp|ele][,fil])1.6*
- offsetParent()
- parent([expr])
- parents([expr])
- parentsUntil([exp|ele][,fil])1.6*
- prev([expr])
- prevall([expr])
- prevUntil([exp|ele][,fil])1.6*
- siblings([expr])

串联

- add(expr|ele|html|obj[,con])
- andSelf()
- contents()
- end()

事件

- 页面载入
- ready(fn)
- 事件处理
- on(eve,[sel],[data],fn)1.7+
- off(eve,[sel],[fn])1.7+
- bind(type,[data],fn)
- one(type,[data],fn)
- trigger(type,[data])
- triggerHandler(type, [data])
- unbind(type,[data|fn])
- 事件委派
- live(type,[data],fn)
- die(type,[fn])
- delegate(sel,[type],[data],fn)
- [undelegate([sel,[type],fn])](#)1.6*
- 事件切换
- hover([over,]out)
- toggle(fn, fn2, [fn3, fn4, ...])
- 事件
- [blur([[data],fn])](#)
- [change([[data],fn])](#)
- [click([[data],fn])](#)
- [dblclick([[data],fn])](#)
- [error([[data],fn])](#)

- [focus([[data],fn])](#)
- focusin([data],fn)
- focusout([data],fn)
- [keydown([[data],fn])](#)
- [keypress([[data],fn])](#)
- [keyup([[data],fn])](#)
- [mousedown([[data],fn])](#)
- [mouseenter([[data],fn])](#)
- [mouseleave([[data],fn])](#)
- [mousemove([[data],fn])](#)
- [mouseout([[data],fn])](#)
- [mouseover([[data],fn])](#)
- [mouseup([[data],fn])](#)
- [resize([[data],fn])](#)
- [scroll([[data],fn])](#)
- [select([[data],fn])](#)
- [submit([[data],fn])](#)
- [unload([[data],fn])](#)

效果

基本

- [show([speed,[easing],[fn]])](#)
- [hide([speed,[easing],[fn]])](#)
- toggle([speed],[easing],[fn])

滑动

- slideDown([spe],[eas],[fn])
- [slideUp([speed,[easing],[fn]])](#)

- slideToggle([speed],[easing],[fn])

- 淡入淡出

- fadeIn([speed],[eas],[fn])

- fadeOut([speed],[eas],[fn])

- [fadeTo([[spe],opa,[eas],[fn]])](#)

- [fadeToggle([speed,[eas],[fn]])](#)

- 自定义

- animate(par,[spe],[e],[fn])1.8*

- stop([cle],[jum])1.7*

- delay(duration,[queueName])

- 设置

- jQuery.fx.off

- jQuery.fx.interval

ajax

- ajax 请求

- $.ajax(url,[settings])

- load(url,[data],[callback])

- $.get(url,[data],[fn],[type])

- $.getJSON(url,[data],[fn])

- $.getScript(url,[callback])

- $.post(url,[data],[fn],[type])

- ajax 事件

- ajaxComplete(callback)

- ajaxError(callback)

- ajaxSend(callback)

- ajaxStart(callback)
- ajaxStop(callback)
- ajaxSuccess(callback)

其它

- $.ajaxSetup([options])
- serialize()
- serializearray()

工具

浏览器及特性检测

- $.support
- $.browser
- $.browser.version
- $.boxModel

数组和对象操作

- $.each(object,[callback])
- $.extend([d],tgt,obj1,[objN])
- $.grep(array,fn,[invert])
- $.sub()
- $.when(deferreds)
- $.makearray(obj)
- $.map(arr|obj,callback)1.6*
- $.inarray(val,arr,[from])
- $.toarray()
- $.merge(first,second)
- $.unique(array)
- $.parseJSON(json)

- 函数操作
- $.noop
- $.proxy(function,context)
- 测试操作
- $.contains(container,contained)
- $.type(obj)
- $.isarray(obj)
- $.isFunction(obj)
- $.isEmptyObject(obj)
- $.isPlainObject(obj)
- $.isWindow(obj)
- $.isNumeric(value)1.7+
- 字符串操作
- $.trim(str)
- URL
- $.param(obj,[traditional])
- 插件编写
- $.error(message)

关于

- 关于此jQuery中文文档
- 提交bug及获取更新

Deferred

- def.done(donCal,[donCal])
- def.fail(failCallbacks)
- ~~def.isRejected()~~1.8-
- ~~def.isResolved()~~1.8-

- def.reject(args)
- def.rejectWith(context,[args])
- def.resolve(args)
- def.resolveWith(context,[args])
- def.then(doneCal,failCals)1.7*
- def.promise([type],[target])1.6+
- def.pipe([donl],[fai],[pro])1.7*
- def.always(alwCal,[alwCal])1.6+
- def.notify(args)1.7+
- def.notifyWith(con,[args])1.7+
- def.progress(proCal)1.7+
- def.state()1.7+

Callbacks

- cal.add(callbacks)1.7+
- cal.disable()1.7+
- cal.empty()1.7+
- cal.fire(arguments)1.7+
- cal.fired()1.7+
- cal.fireWith([context] [, args])1.7+
- cal.has(callback)1.7+
- cal.lock()1.7+
- cal.locked()1.7+
- cal.remove(callbacks)1.7+
- $.callbacks(flags)1.7+

其它

- HTML5速查表
- 正则表达式速查表

核心

jQuery(selector,[context])

返回值:jQueryjQuery([selector,[context]])

概述

这个函数接收一个包含 CSS 选择器的字符串，然后用这个字符串去匹配一组元素。

jQuery 的核心功能都是通过这个函数实现的。 jQuery中的一切都基于这个函数，或者说都是在以某种方式使用这个函数。这个函数最基本的用法就是向它传递一个表达式（通常由 CSS 选择器组成），然后根据这个表达式来查找所有匹配的元素。

默认情况下, 如果没有指定context参数，$()将在当前的 HTML document中查找 DOM 元素；如果指定了 context 参数，如一个 DOM 元素集或 jQuery 对象，那就会在这个 context 中查找。在jQuery 1.3.2以后，其返回的元素顺序等同于在context中出现的先后顺序。

参考文档中 选择器 部分获取更多用于 expression 参数的 CSS 语法的信息。

参数

selector,[context]String,Element,/jQuery *V1.0*

selector:用来查找的字符串

context:作为待查找的 DOM 元素集、文档或 jQuery 对象。

elementElement *V1.0*

一个用于封装成jQuery对象的DOM元素

objectobject *V1.0*

一个用于封装成jQuery对象

elementArrayElement *V1.0*

一个用于封装成jQuery对象的DOM元素数组。

jQuery objectobject *V1.0*

一个用于克隆的jQuery对象。

jQuery() *V1.4*

返回一个空的jQuery对象。

示例

描述:

找到所有 p 元素，并且这些元素都必须是 div 元素的子元素。

jQuery 代码:

```
$("div > p");
```

描述:

设置页面背景色。

jQuery 代码:

```
$(document.body).css( "background", "black" );
```

描述:

隐藏一个表单中所有元素。

jQuery 代码:

```
$(myForm.elements).hide()
```

描述:

在文档的第一个表单中，查找所有的单选按钮(即: type 值为 radio 的 input 元素)。

jQuery 代码:

```
$("input:radio", document.forms[0]);
```

描述:

在一个由 AJAX 返回的 XML 文档中，查找所有的 div 元素。

jQuery 代码:

```
$("div", xml.responseXML);
```

jQuery(html,[ownerDocument])

返回值:jQueryjQuery(html,*[ownerDocument]*)

概述

根据提供的原始 HTML 标记字符串，动态创建由 jQuery 对象包装的 DOM 元素。同时设置一系列的属性、事件等。

你可以传递一个手写的 HTML 字符串，或者由某些模板引擎或插件创建的字符串，也可以是通过 AJAX 加载过来的字符串。但是在你创建 input 元素的时会有限制，可以参考第二个示例。当然这个字符串可以包含斜杠 (比如一个图像地址)，还有反斜杠。当你创建单个元素时，请使用闭合标签或 XHTML 格式。例如，创建一个 span ，可以用$("") 或 $("") ，但不推荐 $("")。在jQuery 中，这个语法等同于 $(document.createElement("span")) 。

在jQuery 1.8中，通过$(html,props)， 您可以使用任何jQuery对象的方法或插件。在此之前，你只能使用一个方法名的短名单，并有没有成文的方式添加到列表中。现在并不需要是一个列表，在所有！然而，请注意，这可能会导致你的代码的行为改变，如果插件添加后，有相同的名称作为HTML属性。

参数

html,[ownerDocument]String,Document *V1.0*

html:用于动态创建DOM元素的HTML标记字符串

ownerDocument:创建DOM元素所在的文档

html,props String,Map *V1.4*

html:用于动态创建DOM元素的HTML标记字符串

props:用于附加到新创建元素上的属性、事件和方法

示例

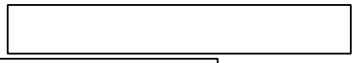
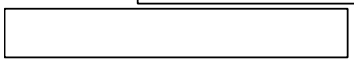
描述:

动态创建一个 div 元素（以及其中的所有内容），并将它追加到 body 元素中。在这个函数的内部，是通过临时创建一个元素，并将这个元素的 innerHTML 属性设置为给定的标记字符串，来实现标记到 DOM 元素转换的。所以，这个函数既有灵活性，也有局限性。

jQuery 代码:

```
$("<div><p>Hello</p></div>").appendTo("body");
```

描述:

创建一个  元素必须同时设定 type 属性。因为微软规定  元素的 type 只能写一次。

jQuery 代码:

```
// 在 IE 中无效:
$("<input>").attr("type", "checkbox");
// 在 IE 中有效:
$("<input type='checkbox'>");
```

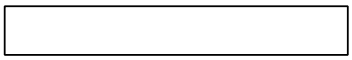
描述:

动态创建一个 div 元素（以及其中的所有内容），并将它追加到 body 元素中。在这个函数的内部，是通过临时创建一个元素，并将这个元素的 innerHTML 属性设置为给定的标记字符串，来实现标记到 DOM 元素转换的。所以，这个函数既有灵活性，也有局限性。

jQuery 代码:

```
$("<div>", {
  "class": "test",
  text: "Click me!",
  click: function(){
    $(this).toggleClass("test");
  }
}).appendTo("body");
```

描述:

创建一个  元素，同时设定 type 属性、属性值，以及一些事件。

jQuery 代码:

```
$("<input>", {
  type: "text",
  val: "Test",
  focusin: function() {
    $(this).addClass("active");
  },
  focusout: function() {
    $(this).removeClass("active");
  }
}).appendTo("form");
```

jQuery(callback)

返回值:jQueryjQuery(callback)

概述

$(document).ready()的简写。

允许你绑定一个在DOM文档载入完成后执行的函数。这个函数的作用如同$(document).ready()一样，只不过用这个函数时，需要把页面中所有需要在 DOM 加载完成时执行的$()操作符都包装到其中来。从技术上来说，这个函数是可链接的 - - 但真正以这种方式链接的情况并不多。你可以在一个页面中使用任意多个$(document).ready事件。参考 ready(Function) 获取更多 ready 事件的信息。

参数

callbackFunction *V1.0*

当DOM加载完成后要执行的函数

示例

描述:

当DOM加载完成后，执行其中的函数。

jQuery 代码:

```
$(function(){
  // 文档就绪
});
```

描述:

使用 $(document).ready() 的简写，同时内部的 jQuery 代码依然使用 $ 作为别名，而不管全局的 $ 为何。

jQuery 代码:

```
jQuery(function($) {
  // 你可以在这里继续使用$作为别名...
});
```

jQuery.holdReady(hold)

返回值:BooleanjQuery.holdReady(hold)

概述

暂停或恢复.ready() 事件的执行。

在$.holdReady()方法允许调用者延迟jQuery的ready事件。这种先进的功能，通常是使用动态脚本加载器，要加载的JavaScript比如 jQuery插件，如额外的ready事件才让事件发生，即使在DOM可能准备。此方法必须早在文件调用，在这样 jQuery脚本后，立即标记。ready事件之后调用此方法即使已经发射也将没有任何效果。

要延迟ready事件，第一次调用的$.holdReady(true)。当ready事件应该执行时，调用$.holdReady(false) 。请注意，多个holds可以放在ready事件上，每个$.holdReady(true)逐一调用。ready事件将不会执行，直到所有已发布了相应的一个$.holdReady(false) *和*正常的文件准备好条件得到满足。（见ready的更多信息。））

参数

hold *V1.6*

指示是否暂停或恢复被请求的ready事件

示例

描述:

延迟就绪事件，直到已加载的插件。

jQuery 代码:

```
$.holdReady(true);
$.getScript("myplugin.js", function() {
    $.holdReady(false);
});
```

each(callback)

返回值:jQueryeach(callback)

概述

以每一个匹配的元素作为上下文来执行一个函数。

意味着，每次执行传递进来的函数时，函数中的this关键字都指向一个不同的DOM元素（每次都是一个不同的匹配元素）。而且，在每次执行函数时，都会给函数传递一个表示作为执行环境的元素在匹配的元素集合中所处位置的数字值作为参数（从零开始的整型）。 返回 'false' 将停止循环 (就像在普通的循环中使用 'break')。返回 'true' 跳至下一个循环(就像在普通的循环中使用'continue')。

参数

callbackFunction *V1.0*

对于每个匹配的元素所要执行的函数

示例

描述:

迭代两个图像，并设置它们的 src 属性。注意:此处 this 指代的是 DOM 对象而非 jQuery 对象。

HTML 代码:

```
<img/><img/>
```

jQuery 代码:

```
$("img").each(function(i){
  this.src = "test" + i + ".jpg";
});
```

结果:

```
[ <img src="test0.jpg" />, <img src="test1.jpg" /> ]
```

描述:

如果你想得到 jQuery对象，可以使用 $(this) 函数。

HTML 代码:

```
<button>Change colors</button>
<span></span>
<div></div>
<div></div>

<div></div>
<div></div>
<div id="stop">Stop here</div>
<div></div>

<div></div>
<div></div>
```

jQuery 代码:

```
$("img").each(function(){
  $(this).toggleClass("example");
});
```

描述:

你可以使用 'return' 来提前跳出 each() 循环。

HTML 代码:

```
<button>Change colors</button>
<span></span>
<div></div>
<div></div>

<div></div>
<div></div>
<div id="stop">Stop here</div>
<div></div>

<div></div>
<div></div>
```

jQuery 代码:

```
$("button").click(function () {
$("div").each(function (index, domEle) {
  // domEle == this
  $(domEle).css("backgroundColor", "yellow");
  if ($(this).is("#stop")) {
  $("span").text("Stopped at div index #" + index);
  return false;
  }
});
});
```

size()

返回值:Numbersize()

*V1.0*概述

jQuery 对象中元素的个数。

当前匹配的元素个数。 size 将返回相同的值。

示例

描述:

计算文档中所有图片数量

HTML 代码:

```
<img src="test1.jpg"/> <img src="test2.jpg"/>
```

jQuery 代码:

```
$("img").size();
```

结果:

```
2
```

length

返回值:Numberlength

*V1.0*概述

jQuery 对象中元素的个数。

这个函数的返回值与 jQuery 对象的'length' 属性一致。

示例

描述:

计算文档中所有图片数量

HTML 代码:

```
<img src="test1.jpg"/> <img src="test2.jpg"/>
```

jQuery 代码:

```
$("img").length;
```

结果:

```
2
```

selector

返回值:Striingselector

*V1.3*概述

返回传给jQuery()的原始选择器。

换句话说，就是返回你用什么选择器来找到这个元素的。可以与context一起使用，用于精确检测选择器查询情况。这两个属性对插件开发人员很有用。

示例

描述:

确定查询的选择器

jQuery 代码:

```
$("ul")
  .append("<li>" + $("ul").selector + "</li>")
  .append("<li>" + $("ul li").selector + "</li>")
  .append("<li>" + $("div#foo ul:not([class])").selector + "</li>");
```

结果:

```
ul
ul li
div#foo ul:not([class])
```

context

返回值:Elementcontext

*V1.3*概述

返回传给jQuery()的原始的DOM节点内容，即jQuery()的第二个参数。如果没有指定，那么context指向当前的文档(document)。

可以与selector一起使用，用于精确检测选择器查询情况。这两个属性对插件开发人员很有用。

示例

描述:

检测使用的文档内容

jQuery 代码:

```
$("ul")
  .append("<li>" + $("ul").context + "</li>")
  .append("<li>" + $("ul", document.body).context.nodeName + "</li>");
```

结果:

```
[object HTMLDocument]  //如果是IE浏览器，则返回[object]
BODY
```

get([index])

返回值:Elementget([index])

概述

取得其中一个匹配的元素。 num表示取得第几个匹配的元素。

这能够让你选择一个实际的DOM 元素并且对他直接操作，而不是通过 jQuery 函数。$(this).get(0)与$(this)[0]等价。

参数

[index]Number

取得第 index 个位置上的元素

get()

取得所有匹配的 DOM 元素集合。

示例

描述:

HTML 代码:

```
<img src="test1.jpg"/> <img src="test2.jpg"/>
```

jQuery 代码:

```
$("img").get(0);
```

结果:

```
[ <img src="test1.jpg"/> ]
```

描述:

选择文档中所有图像作为元素数组，并用数组内建的 reverse 方法将数组反向。

HTML 代码:

```
<img src="test1.jpg"/> <img src="test2.jpg"/>
```

jQuery 代码:

```
$("img").get().reverse();
```

结果:

```
[ <img src="test2.jpg"/> <img src="test1.jpg"/> ]
```

index([selector|element])

返回值:Numberindex([selector|element])

概述

搜索匹配的元素，并返回相应元素的索引值，从0开始计数。

如果不给 .index() 方法传递参数，那么返回值就是这个jQuery对象集合中第一个元素相对于其同辈元素的位置。

如果参数是一组DOM元素或者jQuery对象，那么返回值就是传递的元素相对于原先集合的位置。

如果参数是一个选择器，那么返回值就是原先元素相对于选择器匹配元素中的位置。如果找不到匹配的元素，则返回-1。 具体请参考示例。

参数

index() *V1.4*

selectorSelector *V14*

一个选择器，代表一个jQuery对象，将会从这个对象中查找元素。

elementElement *V1.0*

获得 index 位置的元素。可以是 DOM 元素或 jQuery 选择器。

示例

描述:

查找元素的索引值

HTML 代码:

```
<ul>
  <li id="foo">foo</li>
  <li id="bar">bar</li>
  <li id="baz">baz</li>
</ul>
```

jQuery 代码:

```
$('li').index(document.getElementById('bar')); //1，传递一个DOM对象，返回这个对象在原先集合中的索引位置
$('li').index($('#bar')); //1，传递一个jQuery对象
$('li').index($('li:gt(0)')); //1，传递一组jQuery对象，返回这个对象中第一个元素在原先集合中的索引位置
$('#bar').index('li'); //1，传递一个选择器，返回#bar在所有li中的做引位置
$('#bar').index(); //1，不传递参数，返回这个元素在同辈中的索引位置。
```

data([key],[value])

返回值:jQuerydata([key],[value])

概述

在元素上存放数据,返回jQuery对象。

如果jQuery集合指向多个元素，那将在所有元素上设置对应数据。这个函数不用建立一个新的expando，就能在一个元素上存放任何格式的数据，而不仅仅是字符串。

V1.4.3 新增用法NEWdata(obj)可传入key-value形式的数据。

参数

keyString *V1.23*

存储的数据名.

key,value String,Any *V1.2.3*

key:存储的数据名

value:将要存储的任意数据

objobject *V1.4.3*

一个用于设置数据的键/值对

data() *V1.4.3*

示例

描述:

在一个div上存取数据

HTML 代码:

```
<div></div>
```

jQuery 代码:

```
$("div").data("blah");  // undefined
$("div").data("blah", "hello");  // blah设置为hello
$("div").data("blah");  // hello
$("div").data("blah", 86);  // 设置为86
$("div").data("blah");  //  86
$("div").removeData("blah");  //移除blah
$("div").data("blah");  // undefined
```

描述:

在一个div上存取名/值对数据

HTML 代码:

```
<div></div>
```

jQuery 代码:

```
$("div").data("test", { first: 16, last: "pizza!" });
$("div").data("test").first  //16;
$("div").data("test").last  //pizza!;
```

removeData([name|list])

返回值:jQueryremoveData([name|list])

概述

在元素上移除存放的数据

与$(...).data(name, value)函数作用相反

参数

[name]String *V1.2.3*

存储的数据名

[list]Array,String *V1.7*

移除数组或以空格分开的字符串

示例

描述:

从元素中删除之前添加的数据：

jQuery 代码:

```
$("#btn2").click(function(){
  $("div").removeData("greeting");
  alert("Greeting is: " + $("div").data("greeting"));
});
```

jQuery.data(element,[key],[value])

返回值:jQueryjQuery.data(element,[key],[value])

概述

在元素上存放数据,返回jQuery对象。

注意：这是一个底层方法。你应当使用.data()来代替。

此方法在jQuery 1.8中删除，但你仍然可以通过$._data(element, "events")调试事件数据。请注意，这是不支持的公共接口;实际的数据结构可能会改变从版本之间不兼容。

参数

element,key,valueString,String,Any *V1.2.3*

element:要关联数据的DOM对象

key:存储的数据名

value:将要存储的任意数据

element,key String,String *V1.2.3*

element:要查询数据的DOM对象

key:存储的数据名

elementString *V1.4*

要查询数据的DOM对象

示例

jQuery 代码:

```
jQuery.data(document.body, 'foo', 52);
jQuery.data(document.body, 'bar', 'test');
```

queue(element,[queueName])

返回值:Array/jQueryqueue(element,[queueName])

概述

显示或操作在匹配元素上执行的函数队列

参数

element,[queueName]Element,String *V1.3*

element:检查附加列队的DOM元素

queueName:字符串值，包含序列的名称。默认是 fx, 标准的效果序列。

**element,queueName,newQueue* *Element,String,Array *V1.3*

element:检查附加列队的DOM元素

queueName:字符串值，包含序列的名称。默认是 fx, 标准的效果序列。

newQueue:替换当前函数列队内容的数组

element,queueName,callback() Element,String *V1.3*

element:检查附加列队的DOM元素

queueName:字符串值，包含序列的名称。默认是 fx, 标准的效果序列。

callback():要添加进队列的函数

示例

描述:

显示队列长度

HTML 代码:

```
<style>
  div { margin:3px; width:40px; height:40px;
        position:absolute; left:0px; top:30px;
        background:green; display:none; }
  div.newcolor { background:blue; }
  span { color:red; }
</style>
<button id="show">Show Length of Queue</button>
<span></span>
<div></div>
```

jQuery 代码:

```
$("#show").click(function () {
      var n = $("div").queue("fx");
      $("span").text("Queue length is: " + n.length);
});
function runIt() {
      $("div").show("slow");
      $("div").animate({left:'+=200'},2000);
      $("div").slideToggle(1000);
      $("div").slideToggle("fast");
      $("div").animate({left:'-=200'},1500);
      $("div").hide("slow");
      $("div").show(1200);
      $("div").slideUp("normal", runIt);
}
runIt();
```

描述:

通过设定队列数组来删除动画队列

HTML 代码:

```
<style>
 div { margin:3px; width:40px; height:40px;
       position:absolute; left:0px; top:30px;
       background:green; display:none; }
 div.newcolor { background:blue; }
 </style>

 <button id="start">Start</button>
 <button id="stop">Stop</button>
 <div></div>
```

jQuery 代码:

```
$("#start").click(function () {
      $("div").show("slow");
      $("div").animate({left:'+=200'},5000);
      $("div").queue(function () {
          $(this).addClass("newcolor");
          $(this).dequeue();
      });
      $("div").animate({left:'-=200'},1500);
      $("div").queue(function () {
          $(this).removeClass("newcolor");
          $(this).dequeue();
      });
      $("div").slideUp();
 });
 $("#stop").click(function () {
      $("div").queue("fx", []);
      $("div").stop();
 });
```

描述:

插入一个自定义函数 如果函数执行后要继续队列，则执行 jQuery(this).dequeue();

HTML 代码:

```
<style>
 div { margin:3px; width:40px; height:40px;
       position:absolute; left:0px; top:30px;
       background:green; display:none; }
 div.newcolor { background:blue; }
 </style>
 Click here...
 <div></div>
```

jQuery 代码:

```
$(document.body).click(function () {
      $("div").show("slow");
      $("div").animate({left:'+=200'},2000);
      $("div").queue(function () {
         $(this).addClass("newcolor");
         $(this).dequeue();
      });
      $("div").animate({left:'-=200'},500);
      $("div").queue(function () {
         $(this).removeClass("newcolor");
         $(this).dequeue();
      });
      $("div").slideUp();
});
```

dequeue([queueName])

返回值:jQuerydequeue([queueName])

概述

从队列最前端移除一个队列函数，并执行他。

参数

[queueName]String *V1.2*

队列名，默认为fx

示例

描述:

使用 dequeue() 终止一个自定义的队列函数

jQuery 代码:

```
$("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
});
```

描述:

用dequeue来结束自定义队列函数，并让队列继续进行下去。

HTML 代码:

```
<style>
 div { margin:3px; width:50px; position:absolute;
       height:50px; left:10px; top:30px;
       background-color:yellow; }
 div.red { background-color:red; }
 </style>

 <button>Start</button>
 <div></div>
```

jQuery 代码:

```
$("button").click(function () {
      $("div").animate({left:'+=200px'}, 2000);
      $("div").animate({top:'0px'}, 600);
      $("div").queue(function () {
          $(this).toggleClass("red");
          $(this).dequeue();
      });
      $("div").animate({left:'10px', top:'30px'}, 700);
  });
```

clearQueue([queueName])

返回值:jQueryclearQueue(*[queueName]*)

概述

清空对象上尚未执行的所有队列

如果不带参数，则默认清空的是动画队列。这跟stop(true)类似，但stop()只能清空动画队列，而这个可以清空所有通过 .queue() 创建的队列。

参数

queueName *Boolean*1.4*

含有队列名的字符串。默认是"Fx"，动画队列。

示例

描述:

停止当前正在运行的动画：

jQuery 代码:

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

jQuery.noConflict([extreme])

返回值:jQueryjQuery.noConflict([extreme])

概述

运行这个函数将变量$的控制权让渡给第一个实现它的那个库。

这有助于确保jQuery不会与其他库的$对象发生冲突。在运行这个函数后，就只能使用jQuery变量访问jQuery对象。例如，在要用到$("div p")的地方，就必须换成jQuery("div p")。'''注意:'''这个函数必须在你导入jQuery文件之后，并且在导入另一个导致冲突的库'''之前'''使用。当然也应当在其他冲突的库被使用之前，除非jQuery是最后一个导入的。

参数

extremeBoolean *V1.0*

传入 true 来允许彻底将jQuery变量还原

示例

描述:

将$引用的对象映射回原始的对象。

jQuery 代码:

```
jQuery.noConflict();
// 使用 jQuery
jQuery("div p").hide();
// 使用其他库的 $()
$("content").style.display = 'none';
```

描述:

恢复使用别名$，然后创建并执行一个函数，在这个函数的作用域中仍然将$作为jQuery的别名来使用。在

这个函数中，原来的$对象是无效的。这个函数对于大多数不依赖于其他库的插件都十分有效。

jQuery 代码:

```
jQuery.noConflict();
(function($) {
  $(function() {
    // 使用 $ 作为 jQuery 别名的代码
  });
})(jQuery);
// 其他用 $ 作为别名的库的代码
```

描述:

创建一个新的别名用以在接下来的库中使用jQuery对象。

jQuery 代码:

```
var j = jQuery.noConflict();
// 基于 jQuery 的代码
j("div p").hide();
// 基于其他库的 $() 代码
$("content").style.display = 'none';
```

描述:

完全将 jQuery 移到一个新的命名空间。

jQuery 代码:

```
var dom = {};
dom.query = jQuery.noConflict(true);
```

结果:

```
// 新 jQuery 的代码
dom.query("div p").hide();
// 另一个库 $() 的代码
$("content").style.display = 'none';
// 另一个版本 jQuery 的代码
jQuery("div > p").hide();
```

选择器

#id

返回值:Array#id

概述

根据给定的ID匹配一个元素。

如果选择器中包含特殊字符，可以用两个斜杠转义。参见示例。

参数

idString *V1.0*

用于搜索的，通过元素的 id 属性中给定的值

示例

描述:

查找 ID 为"myDiv"的元素。

HTML 代码:

```
<div id="notMe"><p>id="notMe"</p></div>
<div id="myDiv">id="myDiv"</div>
```

jQuery 代码:

```
$("#myDiv");
```

结果:

```
[ <div id="myDiv">id="myDiv"</div> ]
```

描述:

查找含有特殊字符的元素

HTML 代码:

```
<span id="foo:bar"></span>
<span id="foo[bar]"></span>
<span id="foo.bar"></span>
```

jQuery 代码:

```
#foo\\:bar
#foo\\[bar\\]
#foo\\.bar
```

element

返回值:Arrayelement

概述

根据给定的元素名匹配所有元素

参数

elementString *V1.0*

一个用于搜索的元素。指向 DOM 节点的标签名。

示例

描述:

查找一个 DIV 元素。

HTML 代码:

```
<div>DIV1</div>
<div>DIV2</div>
<span>SPAN</span>
```

jQuery 代码:

```
$("div");
```

结果:

```
[ <div>DIV1</div>, <div>DIV2</div> ]
```

.class

返回值:Array.class

概述

根据给定的类匹配元素。

参数

classString *V1.0*

一个用以搜索的类。一个元素可以有多个类，只要有一个符合就能被匹配到。

示例

描述:

查找所有类是 "myClass" 的元素.

HTML 代码:

```
<div class="notMe">div class="notMe"</div>
<div class="myClass">div class="myClass"</div>
<span class="myClass">span class="myClass"</span>
```

jQuery 代码:

```
$(".myClass");
```

结果:

```
[ <div class="myClass">div class="myClass"</div>, <span class="myClass">span class="myClass"</s
pan> ]
```

*

返回值:Array*

*V1.0*概述

匹配所有元素

多用于结合上下文来搜索。

示例

描述:

找到每一个元素

HTML 代码:

```
<div>DIV</div>
<span>SPAN</span>
<p>P</p>
```

jQuery 代码:

```
$("*")
```

结果:

```
[ <div>DIV</div>, <span>SPAN</span>, <p>P</p> ]
```

selector1,selector2,selectorN

返回值:Arrayselector1,selector2,selectorN

*V1.0*概述

将每一个选择器匹配到的元素合并后一起返回。

你可以指定任意多个选择器，并将匹配到的元素合并到一个结果内。

参数

selector1 Selector

一个有效的选择器

selector2 Selector

另一个有效的选择器

selectorN**Selector

任意多个有效选择器

示例

描述:

找到匹配任意一个类的元素。

HTML 代码:

```
<div>div</div>
<p class="myClass">p class="myClass"</p>
<span>span</span>
<p class="notMyClass">p class="notMyClass"</p>
```

jQuery 代码:

```
$("div,span,p.myClass")
```

结果:

```
[ <div>div</div>, <p class="myClass">p class="myClass"</p>, <span>span</span> ]
```

ancestor descendant

返回值:Arrayancestor descendant

概述

在给定的祖先元素下匹配所有的后代元素

参数

ancestorSelector *V1.0*

任何有效选择器

descendantSelector *V1.0*

用以匹配元素的选择器，并且它是第一个选择器的后代元素

示例

描述:

找到表单中所有的 input 元素

HTML 代码:

```
<form>
  <label>Name:</label>
  <input name="name" />
  <fieldset>
      <label>Newsletter:</label>
      <input name="newsletter" />
 </fieldset>
</form>
<input name="none" />
```

jQuery 代码:

```
$("form input")
```

结果:

```
[ <input name="name" />, <input name="newsletter" /> ]
```

parent > child

返回值:Arrayparent > child

概述

在给定的父元素下匹配所有的子元素

参数

parentSelector *V1.0*

任何有效选择器

childSelector *V1.0*

用以匹配元素的选择器，并且它是第一个选择器的子元素

示例

描述:

匹配表单中所有的子级input元素。

HTML 代码:

```
<form>
  <label>Name:</label>
  <input name="name" />
  <fieldset>
      <label>Newsletter:</label>
      <input name="newsletter" />
 </fieldset>
</form>
<input name="none" />
```

jQuery 代码:

```
$("form > input")
```

结果:

```
[ <input name="name" /> ]
```

prev + next

返回值:Arrayprev + next

概述

匹配所有紧接在 prev 元素后的 next 元素

参数

prevSelector *V1.0*

任何有效选择器

nextSelector *V1.0*

一个有效选择器并且紧接着第一个选择器

示例

描述:

匹配所有跟在 label 后面的 input 元素

HTML 代码:

```
<form>
  <label>Name:</label>
  <input name="name" />
  <fieldset>
      <label>Newsletter:</label>
      <input name="newsletter" />
 </fieldset>
</form>
<input name="none" />
```

jQuery 代码:

```
$("label + input")
```

结果:

```
[ <input name="name" />, <input name="newsletter" /> ]
```

prev ~ siblings

返回值:Arrayprev ~ siblings

概述

匹配 prev 元素之后的所有 siblings 元素

参数

prevSelector *V1.0*

任何有效选择器

siblingsSelector *V1.0*

一个选择器，并且它作为第一个选择器的同辈

示例

描述:

找到所有与表单同辈的 input 元素

HTML 代码:

```
<form>
  <label>Name:</label>
  <input name="name" />
  <fieldset>
     <label>Newsletter:</label>
     <input name="newsletter" />
 </fieldset>
</form>
<input name="none" />
```

jQuery 代码:

```
$("form ~ input")
```

结果:

```
[ <input name="none" /> ]
```

:first

返回值:jQuery:first

*V1.0*概述

获取第一个元素

示例

描述:

获取匹配的第一个元素

HTML 代码:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li>list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

jQuery 代码:

```
$('li:first')
```

;

结果:

```
[ <li>list item 1</li> ]
```

:last

返回值:jQuery:last()

*V1.0*概述

获取最后个元素

示例

描述:

获取匹配的最后个元素

HTML 代码:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li>list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

jQuery 代码:

```
$('li:last')
```

结果:

```
[ <li>list item 5</li> ]
```

:not(selector)

返回值:Array:not(selector)

概述

去除所有与给定选择器匹配的元素

在jQuery 1.3中，已经支持复杂选择器了（例如:not(div a) 和 :not(div,a)）

参数

selectorSelector *V1.0*

用于筛选的选择器

示例

描述:

查找所有未选中的 input 元素

HTML 代码:

```
<input name="apple" />
<input name="flower" checked="checked" />
```

jQuery 代码:

```
$("input:not(:checked)")
```

结果:

```
[ <input name="apple" /> ]
```

:even

返回值:Array:even

*V1.0*概述

匹配所有索引值为偶数的元素，从 0 开始计数

示例

描述:

查找表格的1、3、5...行（即索引值0、2、4...）

HTML 代码:

```
<table>
  <tr><td>Header 1</td></tr>
  <tr><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:even")
```

结果:

```
[ <tr><td>Header 1</td></tr>, <tr><td>Value 2</td></tr> ]
```

:odd

返回值:Array:odd

概述

匹配所有索引值为奇数的元素，从 0 开始计数

示例

描述:

查找表格的2、4、6行（即索引值1、3、5...）

HTML 代码:

```
<table>
  <tr><td>Header 1</td></tr>
  <tr><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:odd")
```

结果:

```
[ <tr><td>Value 1</td></tr> ]
```

:eq(index)

返回值:Array:eq(index)

概述

匹配一个给定索引值的元素

参数

indexNumber *V1.0*

从 0 开始计数

示例

描述:

查找第二行

HTML 代码:

```
<table>
  <tr><td>Header 1</td></tr>
  <tr><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:eq(1)")
```

结果:

```
[ <tr><td>Value 1</td></tr> ]
```

:gt(index)

返回值:Array:gt(index)

概述

匹配所有大于给定索引值的元素

参数

indexNumber *V1.0*

从 0 开始计数

示例

描述:

查找第二第三行，即索引值是1和2，也就是比0大

HTML 代码:

```
<table>
  <tr><td>Header 1</td></tr>
  <tr><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:gt(0)")
```

结果:

```
[ <tr><td>Value 1</td></tr>, <tr><td>Value 2</td></tr> ]
```

:lt(index)

返回值:Array:lt(index)

概述

匹配所有小于给定索引值的元素

参数

indexNumber *V1.0*

从 0 开始计数

示例

描述:

查找第一第二行，即索引值是0和1，也就是比2小

HTML 代码:

```
<table>
  <tr><td>Header 1</td></tr>
  <tr><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:lt(2)")
```

结果:

```
[ <tr><td>Header 1</td></tr>, <tr><td>Value 1</td></tr> ]
```

:header

返回值:Array:header

*V1.2*概述

匹配如 h1, h2, h3之类的标题元素

示例

描述:

给页面内所有标题加上背景色

HTML 代码:

```
<h1>Header 1</h1>
<p>Contents 1</p>
<h2>Header 2</h2>
<p>Contents 2</p>
```

jQuery 代码:

```
$(":header").css("background", "#EEE");
```

结果:

```
[ <h1 style="background:#EEE;">Header 1</h1>, <h2 style="background:#EEE;">Header 2</h2> ]
```

:animated

返回值:Array:animated

*V1.2*概述

匹配所有正在执行动画效果的元素

示例

描述:

只有对不在执行动画效果的元素执行一个动画特效

HTML 代码:

```
<button id="run">Run</button><div></div>
```

jQuery 代码:

```
$("#run").click(function(){
  $("div:not(:animated)").animate({ left: "+=20" }, 1000);
});
```

:focus

返回值:jQuery:focus

概述

匹配当前获取焦点的元素。

如同其他伪类选择器（那些以":"开始），建议:focus前面用标记名称或其他选择;否则，通用选择(" *")是不言而喻的。换句话说，$(':focus')等同为$(*':focus')。如果你正在寻找当前的焦点元素，$(document.activeElement)将检索，而不必搜索整个DOM树。

示例

描述:

添加一个"focused"的类名给那些有focus方法的元素

css 代码:

```
.focused {    background: #abcdef;}
```

html 代码:

```
<div id="content">    <input tabIndex="1">    <input tabIndex="2">    <select tabIndex="3">       <
option>select menu</option>    </select>    <div tabIndex="4">       a div    </div></div>
```

jQuery 代码:

```
$( "#content" ).delegate( "*", "focus blur", function( event ) {
    var elem = $( this );
    setTimeout(function() {
      elem.toggleClass( "focused", elem.is( ":focus" ) );
    }, 0);
});
```

:contains(text)

返回值:Array:contains(text)

概述

匹配包含给定文本的元素

参数

textString *V1.1.4*

一个用以查找的字符串

示例

描述:

查找所有包含 "John" 的 div 元素

HTML 代码:

```
<div>John Resig</div>
<div>George Martin</div>
<div>Malcom John Sinclair</div>
<div>J. Ohn
```

jQuery 代码:

```
$("div:contains('John')")
```

结果:

```
[ <div>John Resig</div>, <div>Malcom John Sinclair</div> ]
```

:empty

返回值:Array:empty

*V1.0*概述

匹配所有不包含子元素或者文本的空元素

示例

描述:

查找所有不包含子元素或者文本的空元素

HTML 代码:

```
<table>
  <tr><td>Value 1</td><td></td></tr>
  <tr><td>Value 2</td><td></td></tr>
</table>
```

jQuery 代码:

```
$("td:empty")
```

结果:

```
[ <td></td>, <td></td> ]
```

:has(selector)

返回值:Array:has(selector)

概述

匹配含有选择器所匹配的元素的元素

参数

selectorSelector *V1.1.4*

一个用于筛选的选择器

示例

描述:

给所有包含 p 元素的 div 元素添加一个 text 类

HTML 代码:

```
<div><p>Hello</p></div>
<div>Hello again!</div>
```

jQuery 代码:

```
$("div:has(p)").addClass("test");
```

结果:

```
[ <div class="test"><p>Hello</p></div> ]
```

:parent

返回值:Array:parent

*V1.0*概述

匹配含有子元素或者文本的元素

示例

描述:

查找所有含有子元素或者文本的 td 元素

HTML 代码:

```
<table>
  <tr><td>Value 1</td><td></td></tr>
  <tr><td>Value 2</td><td></td></tr>
</table>
```

jQuery 代码:

```
$("td:parent")
```

结果:

```
[ <td>Value 1</td>, <td>Value 2</td> ]
```

:hidden

返回值:Array:hidden

*V1.0*概述

匹配所有不可见元素，或者type为hidden的元素

示例

描述:

查找隐藏的 tr

HTML 代码:

```
<table>
  <tr style="display:none"><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:hidden")
```

结果:

```
[ <tr style="display:none"><td>Value 1</td></tr> ]
```

描述:

匹配type为hidden的元素

HTML 代码:

```
<form>
  <input type="text" name="email" />
  <input type="hidden" name="id" />
</form>
```

jQuery 代码:

```
$("input:hidden")
```

结果:

```
[ <input type="hidden" name="id" /> ]
```

:visible

返回值:Array:visible

*V1.0*概述

匹配所有的可见元素

示例

描述:

查找所有可见的 tr 元素

HTML 代码:

```
<table>
  <tr style="display:none"><td>Value 1</td></tr>
  <tr><td>Value 2</td></tr>
</table>
```

jQuery 代码:

```
$("tr:visible")
```

结果:

```
[ <tr><td>Value 2</td></tr> ]
```

[attribute]

返回值:Array[attribute]

概述

匹配包含给定属性的元素。注意，在jQuery 1.3中，前导的@符号已经被废除！如果想要兼容最新版本，只需要简单去掉@符号即可。

参数

attributeString *V1.0*

属性名

示例

描述:

查找所有含有 id 属性的 div 元素

HTML 代码:

```
<div>
  <p>Hello!</p>
</div>
<div id="test2"></div>
```

jQuery 代码:

```
$("div[id]")
```

结果:

```
[ <div id="test2"></div> ]
```

[attribute=value]

返回值:Array[attribute=value]

概述

匹配给定的属性是某个特定值的元素

参数

attributeString *V1.0*

属性名

value **String*V1.0*

属性值。引号在大多数情况下是可选的。但在遇到诸如属性值包含"]"时，用以避免冲突。

示例

描述:

查找所有 name 属性是 newsletter 的 input 元素

HTML 代码:

```
<input type="checkbox" name="newsletter" value="Hot Fuzz" />
<input type="checkbox" name="newsletter" value="Cold Fusion" />
<input type="checkbox" name="accept" value="Evil Plans" />
```

jQuery 代码:

```
$("input[name='newsletter']").attr("checked", true);
```

结果:

```
[ <input type="checkbox" name="newsletter" value="Hot Fuzz" checked="true" />, <input type="ch
eckbox" name="newsletter" value="Cold Fusion" checked="true" /> ]
```

[attribute!=value]

返回值:Array[attribute!=value]

概述

匹配所有不含有指定的属性，或者属性不等于特定值的元素。

此选择器等价于:not([attr=value])

要匹配含有特定属性但不等于特定值的元素，请使用[attr]:not([attr=value])

参数

attributeString *V1.0*

属性名

value**String*V1.0*

属性值。引号在大多数情况下是可选的。但在遇到诸如属性值包含"]"时，用以避免冲突。

示例

描述:

查找所有 name 属性不是 newsletter 的 input 元素

HTML 代码:

```
<input type="checkbox" name="newsletter" value="Hot Fuzz" />
<input type="checkbox" name="newsletter" value="Cold Fusion" />
<input type="checkbox" name="accept" value="Evil Plans" />
```

jQuery 代码:

```
$("input[name!='newsletter']").attr("checked", true);
```

结果:

```
[ <input type="checkbox" name="accept" value="Evil Plans" checked="true" /> ]
```

[attribute^=value]

返回值:Array[attribute^=value]

概述

匹配给定的属性是以某些值开始的元素

参数

attributeString *V1.0*

属性名

value *String* V1.0*

属性值。引号在大多数情况下是可选的。但在遇到诸如属性值包含"]"时，用以避免冲突。

示例

描述:

查找所有 name 以 'news' 开始的 input 元素

HTML 代码:

```
<input name="newsletter" />
<input name="milkman" />
<input name="newsboy" />
```

jQuery 代码:

```
$("input[name^='news']")
```

结果:

```
[ <input name="newsletter" />, <input name="newsboy" /> ]
```

[attribute$=value]

返回值:Array[attribute$=value]

概述

匹配给定的属性是以某些值结尾的元素

参数

attribute String *V1.0*

属性名

value *String* V1.0*

属性值。引号在大多数情况下是可选的。但在遇到诸如属性值包含"]"时，用以避免冲突。

示例

描述:

查找所有 name 以 'letter' 结尾的 input 元素

HTML 代码:

```
<input name="newsletter" />
<input name="milkman" />
<input name="jobletter" />
```

jQuery 代码:

```
$("input[name$='letter']")
```

结果:

```
[ <input name="newsletter" />, <input name="jobletter" /> ]
```

[attribute*=value]

返回值:Array[attribute*=value]

概述

匹配给定的属性是以包含某些值的元素

参数

attributeString *V1.0*

属性名

value **String*V1.0*

属性值。引号在大多数情况下是可选的。但在遇到诸如属性值包含"]"时，用以避免冲突。

示例

描述:

查找所有 name 包含 'man' 的 input 元素

HTML 代码:

```
<input name="man-news" />
<input name="milkman" />
<input name="letterman2" />
<input name="newmilk" />
```

jQuery 代码:

```
$("input[name*='man']")
```

结果:

```
[ <input name="man-news" />, <input name="milkman" />, <input name="letterman2" /> ]
```

[selector1][selector2][selectorN]

返回值:Array[selector1][selector2][selectorN]

*V1.0*概述

复合属性选择器，需要同时满足多个条件时使用。

参数

selector1 Selector

属性选择器

selector2 Selector

另一个属性选择器，用以进一步缩小范围

selectorNSelector

任意多个属性选择器

示例

描述:

找到所有含有 id 属性，并且它的 name 属性是以 man 结尾的

HTML 代码:

```
<input id="man-news" name="man-news" />
<input name="milkman" />
<input id="letterman" name="new-letterman" />
<input name="newmilk" />
```

jQuery 代码:

```
$("input[id][name$='man']")
```

结果:

```
[ <input id="letterman" name="new-letterman" /> ]
```

:nth-child

返回值:Array:nth-child

概述

匹配其父元素下的第N个子或奇偶元素

':eq(index)' 只匹配一个元素，而这个将为每一个父元素匹配子元素。:nth-child从1开始的，而:eq()是从0算起的！可以使用:
nth-child(even)
:nth-child(odd)
:nth-child(3n)
:nth-child(2)
:nth-child(3n+1)
:nth-child(3n+2)

参数

indexNumber *V1.1.4*

要匹配元素的序号，从1开始

示例

描述:

在每个 ul 查找第 2 个li

HTML 代码:

```
<ul>
  <li>John</li>
  <li>Karl</li>
  <li>Brandon</li>
</ul>
<ul>
  <li>Glen</li>
  <li>Tane</li>
  <li>Ralph</li>
</ul>
```

jQuery 代码:

```
$("ul li:nth-child(2)")
```

结果:

```
[ <li>Karl</li>,  <li>Tane</li> ]
```

:first-child

返回值:Array:first-child

*V1.1.4*概述

匹配第一个子元素

':first' 只匹配一个元素，而此选择符将为每个父元素匹配一个子元素

示例

描述:

在每个 ul 中查找第一个 li

HTML 代码:

```
<ul>
  <li>John</li>
  <li>Karl</li>
  <li>Brandon</li>
</ul>
<ul>
  <li>Glen</li>
  <li>Tane</li>
  <li>Ralph</li>
</ul>
```

jQuery 代码:

```
$("ul li:first-child")
```

结果:

```
[ <li>John</li>, <li>Glen</li> ]
```

:last-child

返回值:Array:last-child

*V1.1.4*概述

匹配最后一个子元素

':last'只匹配一个元素，而此选择符将为每个父元素匹配一个子元素

示例

描述:

在每个 ul 中查找最后一个 li

HTML 代码:

```
<ul>
  <li>John</li>
  <li>Karl</li>
  <li>Brandon</li>
</ul>
<ul>
  <li>Glen</li>
  <li>Tane</li>
  <li>Ralph</li>
</ul>
```

jQuery 代码:

```
$("ul li:last-child")
```

结果:

```
[ <li>Brandon</li>, <li>Ralph</li> ]
```

:only-child

返回值:Array:only-child

*V1.1.4*概述

如果某个元素是父元素中唯一的子元素，那将会被匹配

如果父元素中含有其他元素，那将不会被匹配。

示例

描述:

在 ul 中查找是唯一子元素的 li

HTML 代码:

```
<ul>
  <li>John</li>
  <li>Karl</li>
  <li>Brandon</li>
</ul>
<ul>
  <li>Glen</li>
</ul>
```

jQuery 代码:

```
$("ul li:only-child")
```

结果:

```
[ <li>Glen</li> ]
```

:input

返回值:Array:input

*V1.0*概述

匹配所有 input, textarea, select 和 button 元素

示例

描述:

查找所有的input元素，下面这些元素都会被匹配到。

HTML 代码:

```
<form>
  <input type="button" value="Input Button"/>
  <input type="checkbox" />

  <input type="file" />
  <input type="hidden" />
  <input type="image" />

  <input type="password" />
  <input type="radio" />
  <input type="reset" />

  <input type="submit" />
  <input type="text" />
  <select><option>Option</option></select>

  <textarea></textarea>
  <button>Button</button>

</form>
```

jQuery 代码:

```
$(":input")
```

结果:

```
[
  <input type="button" value="Input Button"/>,
  <input type="checkbox" />,

  <input type="file" />,
  <input type="hidden" />,
  <input type="image" />,

  <input type="password" />,
  <input type="radio" />,
  <input type="reset" />,

  <input type="submit" />,
  <input type="text" />,
  <select><option>Option</option></select>,

  <textarea></textarea>,
  <button>Button</button>,
]
```

:text

返回值:Array:text

*V1.0*概述

匹配所有的单行文本框

示例

描述:

查找所有文本框

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":text")
```

结果:

```
[ <input type="text" /> ]
```

:password

返回值:Array:password

*V1.0*概述

匹配所有密码框

示例

描述:

查找所有密码框

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":password")
```

结果:

```
[ <input type="password" /> ]
```

:radio

返回值:Array:radio

*V1.0*概述

匹配所有单选按钮

示例

描述:

查找所有单选按钮

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":radio")
```

结果:

```
[ <input type="radio" /> ]
```

:checkbox

返回值:Array:checkbox

*V1.0*概述

匹配所有复选框

示例

描述:

查找所有复选框

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":checkbox")
```

结果:

```
[ <input type="checkbox" /> ]
```

:submit

返回值:Array:submit

*V1.0*概述

匹配所有提交按钮

示例

描述:

查找所有提交按钮

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":submit")
```

结果:

```
[ <input type="submit" /> ]
```

:image

返回值:Array:image

*V1.0*概述

匹配所有图像域

示例

描述:

匹配所有图像域

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":image")
```

结果:

```
[ <input type="image" /> ]
```

:reset

返回值:Array:reset

*V1.0*概述

匹配所有重置按钮

示例

描述:

查找所有重置按钮

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":reset")
```

结果:

```
[ <input type="reset" /> ]
```

:button

返回值:Array:button

*V1.0*概述

匹配所有按钮

示例

描述:

查找所有按钮.

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":button")
```

结果:

```
[ <input type="button" />,<button></button> ]
```

:file

返回值:Array:file

*V1.0*概述

匹配所有文件域

示例

描述:

查找所有文件域

HTML 代码:

```
<form>
  <input type="text" />
  <input type="checkbox" />
  <input type="radio" />
  <input type="image" />
  <input type="file" />
  <input type="submit" />
  <input type="reset" />
  <input type="password" />
  <input type="button" />
  <select><option/></select>
  <textarea></textarea>
  <button></button>
</form>
```

jQuery 代码:

```
$(":file")
```

结果:

```
[ <input type="file" /> ]
```

:enabled

返回值:Array:enabled

*V1.0*概述

匹配所有可用元素

示例

描述:

查找所有可用的input元素

HTML 代码:

```
<form>
  <input name="email" disabled="disabled" />
  <input name="id" />
</form>
```

jQuery 代码:

```
$("input:enabled")
```

结果:

```
[ <input name="id" /> ]
```

:disabled

返回值:Array:disabled

*V1.0*概述

匹配所有不可用元素

示例

描述:

查找所有不可用的input元素

HTML 代码:

```
<form>
  <input name="email" disabled="disabled" />
  <input name="id" />
</form>
```

jQuery 代码:

```
$("input:disabled")
```

结果:

```
[ <input name="email" disabled="disabled" /> ]
```

:checked

返回值:Array:checked

*V1.0*概述

匹配所有选中的被选中元素(复选框、单选框等，不包括select中的option)

示例

描述:

查找所有选中的复选框元素

HTML 代码:

```
<form>
  <input type="checkbox" name="newsletter" checked="checked" value="Daily" />
  <input type="checkbox" name="newsletter" value="Weekly" />
  <input type="checkbox" name="newsletter" checked="checked" value="Monthly" />
</form>
```

jQuery 代码:

```
$("input:checked")
```

结果:

```
[ <input type="checkbox" name="newsletter" checked="checked" value="Daily" />, <input type="ch
eckbox" name="newsletter" checked="checked" value="Monthly" /> ]
```

:selected

返回值:Array:selected

*V1.0*概述

匹配所有选中的option元素

示例

描述:

查找所有选中的选项元素

HTML 代码:

```
<select>
  <option value="1">Flowers</option>
  <option value="2" selected="selected">Gardens</option>
  <option value="3">Trees</option>
</select>
```

jQuery 代码:

```
$("select option:selected")
```

结果:

```
[ <option value="2" selected="selected">Gardens</option> ]
```

属性

attr(name|pro|key,val|fn)

返回值:Stringattr(name|properties|key,value|fn)

概述

设置或返回被选元素的属性值。

参数

nameString *V1.0*

属性名称

properties **Map*V1.0*

作为属性的“名/值对”对象

key,value **String,Object*V1.0*

属性名称，属性值

key,function(index, attr) **String,Function*V1.1*

1:属性名称。

2:返回属性值的函数,第一个参数为当前元素的索引值，第二个参数为原先的属性值。

示例

参数name 描述:

返回文档中所有图像的src属性值。

jQuery 代码:

```
$("img").attr("src");
```

参数properties 描述:

为所有图像设置src和alt属性。

jQuery 代码:

```
$("img").attr({ src: "test.jpg", alt: "Test Image" });
```

参数key,value 描述:

为所有图像设置src属性。

jQuery 代码:

```
$("img").attr("src","test.jpg");
```

参数key,回调函数 描述:

把src属性的值设置为title属性的值。

jQuery 代码:

```
$("img").attr("title", function() { return this.src });
```

removeAttr(name)

返回值:jQueryremoveAttr(name)

概述

从每一个匹配的元素中删除一个属性

1.6以下版本在IE6使用JQuery的removeAttr方法删除disabled是无效的。解决的方法就是使用
$("XX").prop("disabled",false);
1.7版本在IE6下已支持删除disabled。

参数

nameString *V1.0*

要删除的属性名

示例

描述:

将文档中图像的src属性删除

HTML 代码:

```
<img src="test.jpg"/>
```

jQuery 代码:

```
$("img").removeAttr("src");
```

结果:

```
[ <img /> ]
```

prop(name|pro|key,val|fn)

返回值:jQueryprop(name|properties|key,value|fn)

概述

获取在匹配的元素集中的第一个元素的属性值。

随着一些内置属性的DOM元素或window对象，如果试图将删除该属性，浏览器可能会产生错误。jQuery第一次分配undefined值的属性，而忽略了浏览器生成的任何错误

参数

name *StringV1.6*

属性名称

properties *MapV1.6*

作为属性的“名/值对”对象

key,value *String,ObjectV1.6*

属性名称，属性值

key,function(index, attr) *String,FunctionV1.6*

1:属性名称。

2:返回属性值的函数,第一个参数为当前元素的索引值，第二个参数为原先的属性值。

示例

参数name 描述:

选中复选框为true，没选中为false

jQuery 代码:

```
$("input[type='checkbox']").prop("checked");
```

参数properties 描述:

禁用页面上的所有复选框。

jQuery 代码:

```
$("input[type='checkbox']").prop({
  disabled: true
});
```

参数key,value 描述:

禁用和选中所有页面上的复选框。

jQuery 代码:

```
$("input[type='checkbox']").prop("disabled", false);
$("input[type='checkbox']").prop("checked", true);
```

参数key,回调函数 描述:

通过函数来设置所有页面上的复选框被选中。

jQuery 代码:

```
$("input[type='checkbox']").prop("checked", function( i, val ) {
  return !val;
});
```

removeProp(name)

返回值:jQueryremoveProp(name)

概述

用来删除由.prop()方法设置的属性集

随着一些内置属性的DOM元素或window对象，如果试图将删除该属性，浏览器可能会产生错误。jQuery第一次分配undefined值的属性，而忽略了浏览器生成的任何错误

参数

propertyNameString *V1.6*

要删除的属性名

示例

描述:

设置一个段落数字属性，然后将其删除。

HTML 代码:

```
<p> </p>
```

jQuery 代码:

```
var $para = $("p");
$para.prop("luggageCode", 1234);
$para.append("The secret luggage code is: ", String($para.prop("luggageCode")), ". ");
$para.removeProp("luggageCode");
$para.append("Now the secret luggage code is: ", String($para.prop("luggageCode")), ". ");
```

结果:

```
The secret luggage code is: 1234. Now the secret luggage code is: undefined.
```

addClass(class|fn)

返回值:jQueryaddClass(class|fn)

概述

为每个匹配的元素添加指定的类名。

参数

classString *V1.0*

一个或多个要添加到元素中的CSS类名，请用空格分开

function(index, class) Function *V1.4*

此函数必须返回一个或多个空格分隔的class名。接受两个参数，index参数为对象在这个集合中的索引值，class参数为这个对象原先的class属性值。

示例

参数class 描述:

为匹配的元素加上 'selected' 类

jQuery 代码:

```
$("p").addClass("selected");
$("p").addClass("selected1 selected2");
```

回调函数 描述:

给li加上不同的class

HTML 代码:

```
<ul>
    <li>Hello</li>
    <li>Hello</li>
    <li>Hello</li>
</ul>
```

jQuery 代码:

```
$('ul li:last').addClass(function() {
  return 'item-' + $(this).index();
});
```

removeClass([class|fn])

返回值:jQueryremoveClass([class|fn])

概述

从所有匹配的元素中删除全部或者指定的类。

参数

class *String*V1.0*

一个或多个要删除的CSS类名，请用空格分开

function(index, class) *Function*V1.4*

此函数必须返回一个或多个空格分隔的class名。接受两个参数，index参数为对象在这个集合中的索引值，class参数为这个对象原先的class属性值。

示例

参数class 描述:

从匹配的元素中删除 'selected' 类

jQuery 代码:

```
$("p").removeClass("selected");
```

参数class 描述:

删除匹配元素的所有类

jQuery 代码:

```
$("p").removeClass();
```

回调函数描述:

删除最后一个元素上与前面重复的class

jQuery 代码:

```
$('li:last').removeClass(function() {
    return $(this).prev().attr('class');
});
```

toggleClass(class|fn[,sw])

返回值:jQuerytoggleClass(class|fn[,sw])

概述

如果存在（不存在）就删除（添加）一个类。

参数

classString *V1.0*

CSS类名

class,switch String,Boolean *V1.3*

1:要切换的CSS类名.

2:用于决定元素是否包含class的布尔值。

switchBoolean *V1.4*

用于决定元素是否包含class的布尔值。

**function(index, class,switch)[, switch] **Function,Boolean *V1.4*

1:用来返回在匹配的元素集合中的每个元素上用来切换的样式类名的一个函数。接收元素的索引位置和元素旧的样式类作为参数。

2: 一个用来判断样式类添加还是移除的 boolean 值。

示例

参数class 描述:

为匹配的元素切换 'selected' 类

jQuery 代码:

```
$("p").toggleClass("selected");
```

参数class,switch 描述:

每点击三下加上一次 'highlight' 类

HTML 代码:

```
<strong>jQuery 代码:</strong>
```

jQuery 代码:

```
var count = 0;
$("p").click(function(){
    $(this).toggleClass("highlight", count++ % 3 == 0);
});
```

回调函数 描述:

根据父元素来设置class属性

jQuery 代码:

```
$('div.foo').toggleClass(function() {
  if ($(this).parent().is('.bar') {
    return 'happy';
  } else {
    return 'sad';
  }
});
```

html([val|fn])

返回值:Stringhtml([val|fn])

概述

取得第一个匹配元素的html内容。这个函数不能用于XML文档。但可以用于XHTML文档。

在一个 HTML 文档中, 我们可以使用 .html() 方法来获取任意一个元素的内容。 如果选择器匹配多于一个的元素，那么只有第一个匹配元素的 HTML 内容会被获取。

参数

valString *V1.0*

用于设定HTML内容的值

function(index, html) Function *V1.4*

此函数返回一个HTML字符串。接受两个参数，index为元素在集合中的索引位置，html为原先的HTML值。

示例

无参数 描述:

返回p元素的内容。

jQuery 代码:

```
$('p').html();
```

参数val 描述:

设置所有 p 元素的内容

jQuery 代码:

```
$("p").html("Hello <b>world</b>!");
```

回调函数描述:

使用函数来设置所有匹配元素的内容。

jQuery 代码:

```
$("p").html(function(n){
    return "这个 p 元素的 index 是：" + n;
    });
```

text([val|fn])

返回值:Stringtext([val|fn])

概述

取得所有匹配元素的内容。

结果是由所有匹配元素包含的文本内容组合起来的文本。这个方法对HTML和XML文档都有效。

参数

valString *V1.0*

用于设置元素内容的文本

function(index, text) Function *V1.4*

此函数返回一个字符串。接受两个参数，index为元素在集合中的索引位置，text为原先的text值。

示例

无参数 描述:

返回p元素的文本内容。

jQuery 代码:

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

参数val 描述:

设置所有 p 元素的文本内容

jQuery 代码:

```
$("p").text("Hello world!");
```

回调函数 描述:

使用函数来设置所有匹配元素的文本内容。

jQuery 代码:

```
$("p").text(function(n){
    return "这个 p 元素的 index 是：" + n;
    });
```

val([val|fn|arr])

返回值:String,Arrayval([val|fn|arr])

概述

获得匹配元素的当前值。

在 jQuery 1.2 中,可以返回任意元素的值了。包括select。如果多选，将返回一个数组，其包含所选的值。

参数

valString *V1.0*

要设置的值。

function(index, value) Function *V1.4*

此函数返回一个要设置的值。接受两个参数，index为元素在集合中的索引位置，text为原先的text值。

arrayArray *V1.0*

用于 check/select 的值

示例

无参数 描述:

获取文本框中的值

jQuery 代码:

```
$("input").val();
```

参数val 描述:

设定文本框的值

jQuery 代码:

```
$("input").val("hello world!");
```

回调函数 描述:

设定文本框的值

jQuery 代码:

```
$('input:text.items').val(function() {
  return this.value + ' ' + this.className;
});
```

参数array 描述:

设定一个select和一个多选的select的值

HTML 代码:

```
<select id="single">
  <option>Single</option>
  <option>Single2</option>
</select>
<select id="multiple" multiple="multiple">
  <option selected="selected">Multiple</option>
  <option>Multiple2</option>
  <option selected="selected">Multiple3</option>
</select><br/>
<input type="checkbox" value="check1"/> check1
<input type="checkbox" value="check2"/> check2
<input type="radio" value="radio1"/> radio1
<input type="radio" value="radio2"/> radio2
```

jQuery 代码:

```
$("#single").val("Single2");
$("#multiple").val(["Multiple2", "Multiple3"]);
$("input").val(["check2", "radio1"]);
```

筛选

eq(index|-index)

返回值:jQueryeq(index|-index)

概述

获取第N个元素

参数

indexInteger *V1.1.2*

一个整数，指示元素基于0的位置,这个元素的位置是从0算起。

-index Integer *V1.4*

一个整数，指示元素的位置，从集合中的最后一个元素开始倒数。(1算起)

示例

参数index描述:

获取匹配的第二个元素

HTML 代码:

```
<p> This is just a test.</p> <p> So is this</p>
```

jQuery 代码:

```
$("p").eq(1)
```

结果:

```
[ <p> So is this</p> ]
```

参数-index描述:

获取匹配的第二个元素

HTML 代码:

```
<p> This is just a test.</p> <p> So is this</p>
```

jQuery 代码:

```
$("p").eq(-2)
```

结果:

```
[ <p> This is just a test.</p> ]
```

first()

返回值:jQueryfirst()

*V1.4*概述

获取第一个元素

示例

描述:

获取匹配的第一个元素

HTML 代码:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li>list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

jQuery 代码:

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

结果:

```
[ <li>list item 1</li> ]
```

last()

返回值:jQuerylast()

*V1.4*概述

获取最后个元素

示例

描述:

获取匹配的最后个元素

HTML 代码:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

jQuery 代码:

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

结果:

```
[ <li>list item 5</li> ]
```

hasClass(class)

返回值:BooleanhasClass(class)

概述

检查当前的元素是否含有某个特定的类，如果有，则返回true。

这其实就是 is("." + class)。

参数

classString *V1.2*

用于匹配的类名

示例

描述:

给包含有某个类的元素进行一个动画。

HTML 代码:

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

jQuery 代码:

```
$("div").click(function(){
  if ( $(this).hasClass("protected") )
    $(this)
      .animate({ left: -10 })
      .animate({ left: 10 })
      .animate({ left: -10 })
      .animate({ left: 10 })
      .animate({ left: 0 });
});
```

filter(expr|obj|ele|fn)

返回值:jQueryfilter(expr|obj|ele|fn)

概述

筛选出与指定表达式匹配的元素集合。

这个方法用于缩小匹配的范围。用逗号分隔多个表达式

参数

exprString *V1.0*

字符串值，包含供匹配当前元素集合的选择器表达式。

jQuery objectobject *V1.0*

现有的jQuery对象，以匹配当前的元素。

*element *Expression *V1.4*

一个用于匹配元素的DOM元素。

*function(index) *Function *V1.4*

一个函数用来作为测试元素的集合。它接受一个参数index，这是元素在jQuery集合的索引。在函数，this指的是当前的DOM元素。

示例

参数selector描述:

保留带有select类的元素

HTML 代码:

```
<p>Hello</p><p>Hello Again</p><p class="selected">And Again</p>
```

jQuery 代码:

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

结果:

```
[ <p class="selected">And Again</p> ]
```

参数selector描述:

保留第一个以及带有select类的元素

HTML 代码:

```
<p>Hello</p><p>Hello Again</p><p class="selected">And Again</p>
```

jQuery 代码:

```
$("p").filter(".selected, :first")
```

结果:

```
[ <p>Hello</p>, <p class="selected">And Again</p> ]
```

回调函数 描述:

保留子元素中不含有ol的元素。

HTML 代码:

```
<p><ol><li>Hello</li></ol></p><p>How are you?</p>
```

jQuery 代码:

```
$("p").filter(function(index) {
  return $("ol", this).length == 0;
});
```

结果:

```
[ <p>How are you?</p> ]
```

is(expr|obj|ele|fn)

返回值:Booleanis(expr|obj|ele|fn)

概述

根据选择器、DOM元素或 jQuery 对象来检测匹配元素集合，如果其中至少有一个元素符合这个给定的表达式就返回true。

如果没有元素符合，或者表达式无效，都返回'false'。'''注意：'''在jQuery 1.3中才对所有表达式提供了支持。在先前版本中，如果提供了复杂的表达式，比如层级选择器（比如 +，~ 和 > ），始终会返回true

参数

exprString *V1.0*

字符串值，包含供匹配当前元素集合的选择器表达式。

jQuery objectobject *V1.6*

现有的jQuery对象，以匹配当前的元素。

*element *Expression V1.6

一个用于匹配元素的DOM元素。

*function(index) *Function V1.6

一个函数用来作为测试元素的集合。它接受一个参数index，这是元素在jQuery集合的索引。在函数，this指的是当前的DOM元素。

示例

参数expr 描述:

由于input元素的父元素是一个表单元素，所以返回true。

HTML 代码:

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

jQuery 代码:

```
$("input[type='checkbox']").parent().is("form")
```

结果:

```
true
```

回调函数 描述:

判断点击li标签增加背景色为红色，如果点击的是第2个strong,当前的li增加背景色为绿色，

HTML 代码:

```
<ul>
  <li><strong>list</strong> item 1 - one strong tag</li>
  <li><strong>list</strong> item <strong>2</strong> - two <span>strong tags</span></li>
  <li>list item 3</li>
</ul>
```

jQuery 代码:

```
$("li").click(function() {
  var $li = $(this),
    isWithTwo = $li.is(function() {
      return $('strong', this).length === 2;
    });
  if ( isWithTwo ) {
    $li.css("background-color", "green");
  } else {
    $li.css("background-color", "red");
  }
});
```

结果:

```
  list item 1 - one strong tag
  list item 2 - two strong tags
  list item 3
  list item 4
  list item 5
```

map(callback)

返回值:jQuerymap(callback)

概述

将一组元素转换成其他数组（不论是否是元素数组）

你可以用这个函数来建立一个列表，不论是值、属性还是CSS样式，或者其他特别形式。这都可以用'$.map()'来方便的建立。

参数

callbackFunction *V1.2*

给每个元素执行的函数

示例

描述:

把form中的每个input元素的值建立一个列表。

HTML 代码:

```
<p><b>Values: </b></p>
<form>
  <input type="text" name="name" value="John"/>
  <input type="text" name="password" value="password"/>
  <input type="text" name="url" value="http://ejohn.org/"/>
</form>
```

jQuery 代码:

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

结果:

```
[ <p>John, password, http://ejohn.org/</p> ]
```

has(expr|ele)

返回值:jQueryhas(expr|ele)

概述

保留包含特定后代的元素，去掉那些不含有指定后代的元素。

.has()方法将会从给定的jQuery对象中重新创建一组匹配的对象。提供的选择器会一一测试原先那些对象的后代，含有匹配后代的对象将得以保留。

参数

exprString *V1.4*

一个选择器字符串。

elementDOMElement *V1.4*

一个DOM元素

示例

描述:

给含有ul的li加上背景色

HTML 代码:

```
<ul>
  <li>list item 1</li>
  <li>list item 2
    <ul>
      <li>list item 2-a</li>
      <li>list item 2-b</li>
    </ul>
  </li>
  <li>list item 3</li>
  <li>list item 4</li>
</ul>
```

jQuery 代码:

```
$('li').has('ul').css('background-color', 'red');
```

not(expr|ele|fn)

返回值:jQuerynot(expr|ele|fn)

概述

删除与指定表达式匹配的元素

参数

exprString *V1.0*

一个选择器字符串。

elementDOMElement *V1.0*

一个DOM元素

function(index) Function *V1.4*

一个用来检查集合中每个元素的函数。this是当前的元素。

示例

描述:

从p元素中删除带有 select 的ID的元素

HTML 代码:

```
<p>Hello</p><p id="selected">Hello Again</p>
```

jQuery 代码:

```
$("p").not( $("#selected")[0] )
```

结果:

```
[ <p>Hello</p> ]
```

slice(start, [end])

返回值:jQueryslice(start, *[end]*)

概述

选取一个匹配的子集

与原来的slice方法类似

参数

startInteger *V1.1.4*

开始选取子集的位置。第一个元素是0.如果是负数，则可以从集合的尾部开始选起。

end **Integer*V1.1.4*

结束选取自己的位置，如果不指定，则就是本身的结尾。

示例

描述:

选择第一个p元素

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("p").slice(0, 1).wrapInner("<b></b>");
```

结果:

```
[ <p><b>Hello</b></p> ]
```

描述:

选择前两个p元素

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("p").slice(0, 2).wrapInner("<b></b>");
```

结果:

```
[ <p><b>Hello</b></p>,<p><b>cruel</b></p> ]
```

描述:

只选取第二个p元素

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("p").slice(1, 2).wrapInner("<b></b>");
```

结果:

```
[ <p><b>cruel</b></p> ]
```

描述:

只选取第二第三个p元素

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("p").slice(1).wrapInner("<b></b>");
```

结果:

```
[ <p><b>cruel</b></p>, <p><b>World</b></p> ]
```

描述:

选取第最后一个p元素

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("p").slice(-1).wrapInner("<b></b>");
```

结果:

```
[ <p><b>World</b></p> ]
```

children([expr])

返回值:jQuerychildren(*[expr]*)

概述

取得一个包含匹配的元素集合中每一个元素的所有子元素的元素集合。

可以通过可选的表达式来过滤所匹配的子元素。注意：parents()将查找所有祖辈元素，而children()只考虑子元素而不考虑所有后代元素。

参数

expr *String*V1.0*

用以过滤子元素的表达式

示例

描述:

查找DIV中的每个子元素。

HTML 代码:

```
<p>Hello</p><div><span>Hello Again</span></div><p>And Again</p>
```

jQuery 代码:

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

结果:

```
[ <span>Hello Again</span> ]
```

描述:

在每个div中查找 .selected 的类。

HTML 代码:

```
<div><span>Hello</span><p class="selected">Hello Again</p><p>And Again</p></div>
```

jQuery 代码:

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

结果:

```
[ <p class="selected">Hello Again</p> ]
```

closest(expr,[con]|obj|ele)

返回值:jQueryclosest(expr,[con]|obj|ele)

概述

jQuery 1.3新增。从元素本身开始，逐级向上级元素匹配，并返回最先匹配的元素。。

closest会首先检查当前元素是否匹配，如果匹配则直接返回元素本身。如果不匹配则向上查找父元素，一层一层往上，直到找到匹配选择器的元素。如果什么都没找到则返回一个空的jQuery对象。

closest和parents的主要区别是：1，前者从当前元素开始匹配寻找，后者从父元素开始匹配寻找；2，前者逐级向上查找，直到发现匹配的元素后就停止了，后者一直向上查找直到根元素，然后把这些元素放进一个临时集合中，再用给定的选择器表达式去过滤；3，前者返回0或1个元素，后者可能包含0个，1个，或者多个元素。

closest对于处理事件委托非常有用。

参数

expr *String,Array*1.3*

用以过滤元素的表达式。jQuery 1.4开始，也可以传递一个字符串数组，用于查找多个元素。

expr,[context] *String*1.4*

expr：用以过滤子元素的表达式

context：DOM元素在其中一个匹配的元素可以被发现。如果没有上下文在当时的情况下通过了jQuery设置将被使用。

jQuery object *object*1.6*

一个用于匹配元素的jQuery对象

element *DOMElement*1.6*

一个用于匹配元素的DOM元素。

示例

描述:

展示如何使用clostest查找多个元素

HTML 代码:

```
<ul><li></li><li></li></ul>
```

jQuery 代码:

```
$("li:first").closest(["ul", "body"]);
```

结果:

```
[ul, body]
```

描述:

展示如何使用clostest来完成事件委托。

HTML 代码:

```
<ul>
  <li><b>Click me!</b></li>
  <li>You can also <b>Click me!</b></li>
</ul>
```

jQuery 代码:

```
$(document).bind("click", function (e) {
  $(e.target).closest("li").toggleClass("hilight");
});
```

find(expr|obj|ele)

返回值:jQueryfind(expr|obj|ele)

概述

搜索所有与指定表达式匹配的元素。这个函数是找出正在处理的元素的后代元素的好方法。

所有搜索都依靠jQuery表达式来完成。这个表达式可以使用CSS1-3的选择器语法来写。

参数

exprString *V1.0*

用于查找的表达式

jQuery object **object*V1.6**

一个用于匹配元素的jQuery对象

elementDOMElement *V1.6*

一个DOM元素

示例

描述:

从所有的段落开始，进一步搜索下面的span元素。与$("p span")相同。

HTML 代码:

```
<p><span>Hello</span>, how are you?</p>
```

jQuery 代码:

```
$("p").find("span")
```

结果:

```
[ <span>Hello</span> ]
```

next([expr])

返回值:jQuerynext(*[expr]*)

概述

取得一个包含匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合。

这个函数只返回后面那个紧邻的同辈元素，而不是后面所有的同辈元素（可以使用nextAll）。可以用一个可选的表达式进行筛选。

参数

expr *String*V1.0*

用于筛选的表达式

示例

描述:

找到每个段落的后面紧邻的同辈元素。

HTML 代码:

```
<p>Hello</p><p>Hello Again</p><div><span>And Again</span></div>
```

jQuery 代码:

```
$("p").next()
```

结果:

```
[ <p>Hello Again</p>, <div><span>And Again</span></div> ]
```

描述:

找到每个段落的后面紧邻的同辈元素中类名为selected的元素。

HTML 代码:

```
<p>Hello</p><p class="selected">Hello Again</p><div><span>And Again</span></div>
```

jQuery 代码:

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

结果:

```
[ <p class="selected">Hello Again</p> ]
```

nextAll([expr])

返回值:jQuerynextAll(*[expr]*)

概述

查找当前元素之后所有的同辈元素。

可以用表达式过滤

参数

expr **String*V1.2*

用来过滤的表达式

示例

描述:

给第一个div之后的所有元素加个类

HTML 代码:

```
<div></div><div></div><div></div><div></div>
```

jQuery 代码:

```
$("div:first").nextAll().addClass("after");
```

结果:

```
[ <div class="after"></div>, <div class="after"></div>, <div class="after"></div> ]
```

nextUntil([exp|ele][,fil])

返回值:jQuerynextUntil([exp|ele][,fil])

概述

查找当前元素之后所有的同辈元素，直到遇到匹配的那个元素为止。

如果提供的jQuery代表了一组DOM元素，.nextUntil()方法也能让我们找遍所有元素所在的DOM树，直到遇到了一个跟提供的参数匹配的元素的时候才会停下来。这个新jQuery对象里包含了下面所有找到的同辈元素，但不包括那个选择器匹配到的元素。

如果没有选择器匹配到，或者没有提供参数，那么跟在后面的所有同辈元素都会被选中。这就跟用没有提供参数的 .nextAll()效果一样。

参数

[expr][,filter] *String,String*1.4*

expr: 用于筛选祖先元素的表达式。

filter: 一个字符串，其中包含一个选择表达式匹配元素。

[element][,filter] *DOMElement,String*1.6*

element: 用于筛选祖先元素的DOM元素。

filter: 一个字符串，其中包含一个选择表达式匹配元素。

示例

描述:

给#term-2后面直到dt前的元素加上红色背景

HTML 代码:

```
<dl>
  <dt>term 1</dt>
  <dd>definition 1-a</dd>
  <dd>definition 1-b</dd>
  <dd>definition 1-c</dd>
  <dd>definition 1-d</dd>

  <dt id="term-2">term 2</dt>
  <dd>definition 2-a</dd>
  <dd>definition 2-b</dd>
  <dd>definition 2-c</dd>

  <dt>term 3</dt>
  <dd>definition 3-a</dd>
  <dd>definition 3-b</dd>
</dl>
```

jQuery 代码:

```
$('#term-2').nextUntil('dt').css('background-color', 'red');

var term3 = document.getElementById("term-3");
$("#term-1").nextUntil(term3, "dd").css("color", "green");
```

结果:

```
term 1definition 1-adefinition 1-bdefinition 1-cdefinition 1-dterm 2definition 2-adefinition 2-bdefiniti
on 2-cterm 3definition 3-adefinition 3-b
```

parent([expr])

返回值:jQueryparent(*[expr]*)

概述

取得一个包含着所有匹配元素的唯一父元素的元素集合。

你可以使用可选的表达式来筛选。

参数

expr *String*V1.0*

用来筛选的表达式

示例

描述:

查找每个段落的父元素

HTML 代码:

```
<div><p>Hello</p><p>Hello</p></div>
```

jQuery 代码:

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

结果:

```
[ <div><p>Hello</p><p>Hello</p></div>]
```

描述:

查找段落的父元素中每个类名为selected的父元素。

HTML 代码:

```
<div><p>Hello</p></div><div class="selected"><p>Hello Again</p></div>
```

jQuery 代码:

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

结果:

```
[ <div class="selected"><p>Hello Again</p></div> ]
```

parents([expr])

返回值:jQueryparents(*[expr]*)

概述

取得一个包含着所有匹配元素的祖先元素的元素集合（不包含根元素）。可以通过一个可选的表达式进行筛选。

参数

expr *String*V1.0*

用于筛选祖先元素的表达式

示例

描述:

找到每个span元素的所有祖先元素。

HTML 代码:

```
<html><body><div><p><span>Hello</span></p><span>Hello Again</span></div></body></html>
```

jQuery 代码:

```
$("span").parents()
```

描述:

找到每个span的所有是p元素的祖先元素。

jQuery 代码:

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

parentsUntil([exp|ele][,fil])

返回值:jQueryparentsUntil([exp|ele][,fil])

概述

查找当前元素的所有的父辈元素，直到遇到匹配的那个元素为止。

如果提供的jQuery代表了一组DOM元素，.parentsUntil()方法也能让我们找遍所有元素的祖先元素，直到遇到了一个跟提供的参数匹配的元素的时候才会停下来。这个返回的jQuery对象里包含了下面所有找到的父辈元素，但不包括那个选择器匹配到的元素。

参数

[expr][,filter] *String,String*1.4*

expr: 用于筛选祖先元素的表达式

filter: 一个字符串，其中包含一个选择表达式匹配元素。

[element][,filter] *DOMElement,String*1.6*

element:用于筛选祖先元素的DOM元素

filter: 一个字符串，其中包含一个选择表达式匹配元素。

示例

描述:

查找item-a的祖先，但不包括level-1

HTML 代码:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

jQuery 代码:

```
$('li.item-a').parentsUntil('.level-1')
    .css('background-color', 'red');
```

prev([expr])

返回值:jQueryprev(*[expr]*)

概述

取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合。

可以用一个可选的表达式进行筛选。只有紧邻的同辈元素会被匹配到，而不是前面所有的同辈元素。

参数

expr *String*V1.0*

用于筛选前一个同辈元素的表达式

示例

描述:

找到每个段落紧邻的前一个同辈元素。

HTML 代码:

```
<p>Hello</p><div><span>Hello Again</span></div><p>And Again</p>
```

jQuery 代码:

```
$("p").prev()
```

结果:

```
[ <div><span>Hello Again</span></div> ]
```

描述:

找到每个段落紧邻的前一个同辈元素中类名为selected的元素。

HTML 代码:

```
<div><span>Hello</span></div><p class="selected">Hello Again</p><p>And Again</p>
```

jQuery 代码:

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

结果:

```
[ <p class="selected">Hello Again</p> ]
```

prevAll([expr])

返回值:jQueryprevAll(*[expr]*)

概述

查找当前元素之前所有的同辈元素

可以用表达式过滤。

参数

expr ***String***V1.2*

用于过滤的表达式

示例

描述:

给最后一个之前的所有div加上一个类

HTML 代码:

```
<div></div><div></div><div></div><div></div>
```

jQuery 代码:

```
$("div:last").prevAll().addClass("before");
```

结果:

```
[ <div class="before"></div>, <div class="before"></div>, <div class="before"></div> ]
```

prevUntil([exp|ele][,fil])

返回值:jQueryprevUntil([exp|ele][,fil])

概述

查找当前元素之前所有的同辈元素，直到遇到匹配的那个元素为止。

如果提供的jQuery代表了一组DOM元素，.prevUntil()方法也能让我们找遍所有元素所在的DOM树，直到遇到了一个跟提供的参数匹配的元素的时候才会停下来。这个新jQuery对象里包含了前面所有找到的同辈元素，但不包括那个选择器匹配到的元素。

如果没有选择器匹配到，或者没有提供参数，那么排在前面的所有同辈元素都会被选中。这就跟用没有提供参数的 .prevAll()效果一样。

参数

[expr][,filter] *String,String*1.4*

expr: 用于筛选祖先元素的表达式

filter: 一个字符串，其中包含一个选择表达式匹配元素。

[element][,filter] *DOMElement,StringV1.6*

element:用于筛选祖先元素的DOM元素

filter: 一个字符串，其中包含一个选择表达式匹配元素。

示例

描述:

给#term-2前面直到dt前的元素加上红色背景

HTML 代码:

```
<dl>
  <dt>term 1</dt>
  <dd>definition 1-a</dd>
  <dd>definition 1-b</dd>
  <dd>definition 1-c</dd>
  <dd>definition 1-d</dd>

  <dt id="term-2">term 2</dt>
  <dd>definition 2-a</dd>
  <dd>definition 2-b</dd>
  <dd>definition 2-c</dd>

  <dt>term 3</dt>
  <dd>definition 3-a</dd>
  <dd>definition 3-b</dd>
</dl>
```

jQuery 代码:

```
$('#term-2').prevUntil('dt').css('background-color', 'red');
```

siblings([expr])

返回值:jQuerysiblings(*[expr]*)

概述

取得一个包含匹配的元素集合中每一个元素的所有唯一同辈元素的元素集合。可以用可选的表达式进行筛选。

参数

expr *_String_ 1.0*

用于筛选同辈元素的表达式

示例

描述:

找到每个div的所有同辈元素。

HTML 代码:

```
<p>Hello</p><div><span>Hello Again</span></div><p>And Again</p>
```

jQuery 代码:

```
$("div").siblings()
```

结果:

```
[ <p>Hello</p>, <p>And Again</p> ]
```

描述:

找到每个div的所有同辈元素中带有类名为selected的元素。

HTML 代码:

```
<div><span>Hello</span></div><p class="selected">Hello Again</p><p>And Again</p>
```

jQuery 代码:

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

结果:

```
[ <p class="selected">Hello Again</p> ]
```

add(expr|ele|html|obj[,con])

返回值:jQueryadd(expr|ele|html|obj*[,con]*)

概述

把与表达式匹配的元素添加到jQuery对象中。这个函数可以用于连接分别与两个表达式匹配的元素结果集。

jQuery 1.4 中， .add()方法返回的结果将始终以元素在HTML文档中出现的顺序来排序，而不再是简单的添加。

参数

expr *String*1.0*

一个用于匹配元素的选择器字符串。

elements *DOMElement*1.0*

DOM元素。

html *String*1.0*

HTML片段添加到匹配的元素。

jQuery object *object*1.3.2*

一个jqeruy对象增加到匹配的元素

expr,context *Element, jQuery*1.4*

expr:用于匹配元素并添加的表达式字符串，或者用于动态生成的HTML代码，如果是一个字符串数组则返回多个元素

context:作为待查找的 DOM 元素集、文档或 jQuery 对象。

示例

描述:

添加一个新元素到一组匹配的元素中，并且这个新元素能匹配给定的表达式。

HTML 代码:

```
<p>Hello</p><span>Hello Again</span>
```

jQuery 代码:

```
$("p").add("span")
```

结果:

```
[ <p>Hello</p>, <span>Hello Again</span> ]
```

描述:

动态生成一个元素并添加至匹配的元素中

HTML 代码:

```
<p>Hello</p>
```

jQuery 代码:

```
$("p").add("<span>Again</span>")
```

结果:

```
[ <p>Hello</p>, <span>Hello Again</span> ]
```

描述:

为匹配的元素添加一个或者多个元素

HTML 代码:

```
<p>Hello</p><p><span id="a">Hello Again</span></p>
```

jQuery 代码:

```
$("p").add(document.getElementById("a"))
```

结果:

```
[ <p>Hello</p>, <p><span id="a">Hello Again</span></p>, <span id="a">Hello Again</span> ]
```

andSelf()

返回值:jQueryandSelf()

*V1.2*概述

加入先前所选的加入当前元素中

对于筛选或查找后的元素，要加入先前所选元素时将会很有用。

示例

描述:

选取所有div以及内部的p，并加上border类

HTML 代码:

```
<div><p>First Paragraph</p><p>Second Paragraph</p></div>
```

jQuery 代码:

```
$("div").find("p").andSelf().addClass("border");
```

结果:

```
<div class="border">
  <p class="border">First Paragraph</p>
  <p class="border">Second Paragraph</p>
</div>
```

contents()

返回值:jQuerycontents()

*V1.2*概述

查找匹配元素内部所有的子节点（包括文本节点）。如果元素是一个iframe，则查找文档内容

示例

描述:

查找所有文本节点并加粗

HTML 代码:

```
<p>Hello <a href="http://ejohn.org/">John</a>, how are you doing?</p>
```

jQuery 代码:

```
$("p").contents().not("[nodeType=1]").wrap("<b/>");
```

结果:

```
<p><b>Hello</b> <a href="http://ejohn.org/">John</a>, <b>how are you doing?</b></p>
```

描述:

往一个空框架中加些内容

HTML 代码:

```
<iframe src="/index-blank.html" width="300" height="100"></iframe>
```

jQuery 代码:

```
$("iframe").contents().find("body")
  .append("I'm in an iframe!");
```

end()

返回值:jQueryend()

*V1.0*概述

回到最近的一个"破坏性"操作之前。即，将匹配的元素列表变为前一次的状态。

如果之前没有破坏性操作，则返回一个空集。所谓的"破坏性"就是指任何改变所匹配的jQuery元素的操作。这包括在 Traversing 中任何返回一个jQuery对象的函数--'add', 'andSelf', 'children', 'filter', 'find', 'map', 'next', 'nextAll', 'not', 'parent', 'parents', 'prev', 'prevAll', 'siblings' and 'slice'--再加上 Manipulation 中的 'clone'。

示例

描述:

选取所有的p元素，查找并选取span子元素，然后再回过来选取p元素

HTML 代码:

```
<p><span>Hello</span>,how are you?</p>
```

jQuery 代码:

```
$("p").find("span").end()
```

结果:

```
[ <p><span>Hello</span> how are you?</p> ]
```

文档处理

append(content|fn)

返回值:jQueryappend(content|fn)

概述

向每个匹配的元素内部追加内容。

这个操作与对指定的元素执行appendChild方法，将它们添加到文档中的情况类似。

参数

contentString, Element, jQuery *V1.0*

要追加到目标中的内容

function(index, html) Function *V1.4*

返回一个HTML字符串，用于追加到每一个匹配元素的里边。接受两个参数，index参数为对象在这个集合中的索引值，html参数为这个对象原先的html值。

示例

描述:

向所有段落中追加一些HTML标记。

HTML 代码:

```
<p>I would like to say: </p>
```

jQuery 代码:

```
$("p").append("<b>Hello</b>");
```

结果:

```
[ <p>I would like to say: <b>Hello</b></p> ]
```

appendTo(content)

返回值:jQueryappendTo(content)

*V1.0*概述

把所有匹配的元素追加到另一个指定的元素元素集合中。

实际上，使用这个方法是颠倒了常规的$(A).append(B)的操作，即不是把B追加到A中，而是把A追加到B中。

在jQuery 1.3.2中，appendTo, prependTo, insertBefore, insertAfter, 和 replaceAll这个几个方法成为一个破坏性操作，返回值是所有被追加的内容，而不仅仅是先前所选中的元素。所以，要选择先前选中的元素，需要使用end()方法，参见例二。

参数

contentString

用于被追加的内容

示例

描述:

把所有段落追加到ID值为foo的元素中。

HTML 代码:

```
<p>I would like to say: </p>
<div></div><div></div>
```

jQuery 代码:

```
$("p").appendTo("div");
```

结果:

```
<div><p>I would like to say: </p></div>
<div><p>I would like to say: </p></div>
```

描述:

新建段落追加div中并加上一个class

HTML 代码:

```
<div></div><div></div>
```

jQuery 代码:

```
$("<p/>")
  .appendTo("div")
  .addClass("test")
  .end()
  .addClass("test2");
```

结果:

```
<div><p class="test test2"></p></div>
<div><p class="test"></p></div>
```

prepend(content|fn)

返回值:jQueryprepend(content)

概述

向每个匹配的元素内部前置内容。

这是向所有匹配元素内部的开始处插入内容的最佳方式。

参数

contentString, Element, jQuery *V1.0*

要插入到目标元素内部前端的内容

function(index, html) Function *V1.4*

返回一个HTML字符串，用于追加到每一个匹配元素的里边。接受两个参数，index参数为对象在这个集合中的索引值，html参数为这个对象原先的html值。

示例

描述:

向所有段落中前置一些HTML标记代码。

HTML 代码:

```
<p>I would like to say: </p>
```

jQuery 代码:

```
$("p").prepend("<b>Hello</b>");
```

结果:

```
[ <p><b>Hello</b>I would like to say: </p> ]
```

描述:

将一个DOM元素前置入所有段落

HTML 代码:

```
<p>I would like to say: </p>
<p>I would like to say: </p>
<b class="foo">Hello</b>
<b class="foo">Good Bye</b>
```

jQuery 代码:

```
$("p").prepend( $(".foo")[0] );
```

结果:

```
<p><b class="foo">Hello</b>I would like to say: </p>
<p><b class="foo">Hello</b>I would like to say: </p>
<b class="foo">Good Bye</b>
```

描述:

向所有段落中前置一个jQuery对象(类似于一个DOM元素数组)。

HTML 代码:

```
<p>I would like to say: </p><b>Hello</b>
```

jQuery 代码:

```
$("p").prepend( $("b") );
```

结果:

```
<p><b>Hello</b>I would like to say: </p>
```

prependTo(content)

返回值:jQueryprependTo(content)

概述

把所有匹配的元素前置到另一个、指定的元素元素集合中。

实际上，使用这个方法是颠倒了常规的$(A).prepend(B)的操作，即不是把B前置到A中，而是把A前置到B中。

在jQuery 1.3.2中，appendTo, prependTo, insertBefore, insertAfter, 和 replaceAll这个几个方法成为一个破坏性操作，要选择先前选中的元素，需要使用end()方法，参见 appendTo 方法的例二。

参数

contentString

用于匹配元素的jQuery表达式

示例

描述:

把所有段落追加到ID值为foo的元素中。

HTML 代码:

```
<p>I would like to say: </p><div id="foo"></div>
```

jQuery 代码:

```
$("p").prependTo("#foo");
```

结果:

```
<div id="foo"><p>I would like to say: </p></div>
```

after(content|fn)

返回值:jQueryafter(content|fn)

概述

在每个匹配的元素之后插入内容。

参数

contentString, Element, jQuery *V1.0*

插入到每个目标后的内容

functionFunction *V1.4*

函数必须返回一个html字符串。

示例

描述:

在所有段落之后插入一些HTML标记代码。

HTML 代码:

```
<p>I would like to say: </p>
```

jQuery 代码:

```
$("p").after("<b>Hello</b>");
```

结果:

```
<p>I would like to say: </p><b>Hello</b>
```

描述:

在所有段落之后插入一个DOM元素。

HTML 代码:

```
<b id="foo">Hello</b><p>I would like to say: </p>
```

jQuery 代码:

```
$("p").after( $("#foo")[0] );
```

结果:

```
<p>I would like to say: </p><b id="foo">Hello</b>
```

描述:

在所有段落中后插入一个jQuery对象(类似于一个DOM元素数组)。

HTML 代码:

```
<b>Hello</b><p>I would like to say: </p>
```

jQuery 代码:

```
$("p").after( $("b") );
```

结果:

```
<p>I would like to say: </p><b>Hello</b>
```

before(content|fn)

返回值:jQuerybefore(content|fn)

概述

在每个匹配的元素之前插入内容。

参数

contentString, Element, jQuery *V1.0*

插入到每个目标后的内容

functionFunction *V1.4*

函数必须返回一个html字符串。

示例

描述:

在所有段落之前插入一些HTML标记代码。

HTML 代码:

```
<p>I would like to say: </p>
```

jQuery 代码:

```
$("p").before("<b>Hello</b>");
```

结果:

```
[ <b>Hello</b><p>I would like to say: </p> ]
```

描述:

在所有段落之前插入一个元素。

HTML 代码:

```
<p>I would like to say: </p><b id="foo">Hello</b>
```

jQuery 代码:

```
$("p").before( $("#foo")[0] );
```

结果:

```
<b id="foo">Hello</b><p>I would like to say: </p>
```

描述:

在所有段落中前插入一个jQuery对象(类似于一个DOM元素数组)。

HTML 代码:

```
<p>I would like to say: </p><b>Hello</b>
```

jQuery 代码:

```
$("p").before( $("b") );
```

结果:

```
<b>Hello</b><p>I would like to say: </p>
```

insertAfter(content)

返回值:jQueryinsertAfter(content)

概述

把所有匹配的元素插入到另一个、指定的元素元素集合的后面。

实际上，使用这个方法是颠倒了常规的$(A).after(B)的操作，即不是把B插入到A后面，而是把A插入到B后面。

在jQuery 1.3.2中，appendTo, prependTo, insertBefore, insertAfter, 和 replaceAll这个几个方法成为一个破坏性操作，要选择先前选中的元素，需要使用end()方法，参见 appendTo 方法的例二。

参数

contentString *V1.0*

用于匹配元素的jQuery表达式

示例

描述:

把所有段落插入到一个元素之后。与 $("#foo").after("p")相同

HTML 代码:

```
<p>I would like to say: </p><div id="foo">Hello</div>
```

jQuery 代码:

```
$("p").insertAfter("#foo");
```

结果:

```
<div id="foo">Hello</div><p>I would like to say: </p>
```

insertBefore(content)

返回值:jQueryinsertBefore(content)

概述

把所有匹配的元素插入到另一个、指定的元素元素集合的前面。

实际上，使用这个方法是颠倒了常规的$(A).before(B)的操作，即不是把B插入到A前面，而是把A插入到B前面。

在jQuery 1.3.2中，appendTo, prependTo, insertBefore, insertAfter, 和 replaceAll这个几个方法成为一个破坏性操作，要选择先前选中的元素，需要使用end()方法，参见 appendTo 方法的例二。

参数

contentString *V1.0*

用于匹配元素的jQuery表达式

示例

描述:

把所有段落插入到一个元素之前。与 $("#foo").before("p")相同。

HTML 代码:

```
<div id="foo">Hello</div><p>I would like to say: </p>
```

jQuery 代码:

```
$("p").insertBefore("#foo");
```

结果:

```
<p>I would like to say: </p><div id="foo">Hello</div>
```

wrap(html|ele|fn)

返回值:jQuerywrap(html|ele|fn)

概述

把所有匹配的元素用其他元素的结构化标记包裹起来。

这种包装对于在文档中插入额外的结构化标记最有用，而且它不会破坏原始文档的语义品质。这个函数的原理是检查提供的第一个元素（它是由所提供的HTML标记代码动态生成的），并在它的代码结构中找到最上层的祖先元素 - - 这个祖先元素就是包裹元素。当HTML标记代码中的元素包含文本时无法使用这个函数。因此，如果要添加文本应该在包裹完成之后再行添加。

参数

htmlString *V1.0*

HTML标记代码字符串，用于动态生成元素并包裹目标元素

elemElement *V1.0*

用于包装目标元素的DOM元素

fnFunction *V1.4*

生成包裹结构的一个函数

示例

html参数描述:

把所有的段落用一个新创建的div包裹起来

jQuery 代码:

```
$("p").wrap("<div class='wrap'></div>");
```

elem参数描述:

用ID是"content"的div将每一个段落包裹起来

jQuery 代码:

```
$("p").wrap(document.getElementById('content'));
```

回调函数 描述:

用原先div的内容作为新div的class，并将每一个元素包裹起来

HTML 代码:

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

jQuery 代码:

```
$('.inner').wrap(function() {
  return '<div class="' + $(this).text() + '" />';
});
```

结果:

```
<div class="container">
  <div class="Hello">
    <div class="inner">Hello</div>
  </div>
  <div class="Goodbye">
    <div class="inner">Goodbye</div>
  </div>
</div>
```

unwrap()

返回值:jQueryunwrap()

概述

这个方法将移出元素的父元素。这能快速取消 .wrap()方法的效果。匹配的元素（以及他们的同辈元素）会在DOM结构上替换他们的父元素。

示例

描述:

用ID是"content"的div将每一个段落包裹起来

HTML 代码:

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

jQuery 代码:

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

结果:

```
  <p>Hello</p>
  <p>cruel</p>
  <p>World</p>
```

wrapAll(html|ele)

返回值:jQuerywrapAll(html|ele)

概述

将所有匹配的元素用单个元素包裹起来

这于 '.wrap()' 是不同的，'.wrap()'为每一个匹配的元素都包裹一次。这种包装对于在文档中插入额外的结构化标记最有用，而且它不会破坏原始文档的语义品质。这个函数的原理是检查提供的第一个元素并在它的代码结构中找到最上层的祖先元素 - - 这个祖先元素就是包装元素。

参数

htmlString *V1.2*

TML标记代码字符串，用于动态生成元素并包装目标元素

elemElement *V1.2*

用于包装目标元素的DOM元素

示例

html描述:

用一个生成的div将所有段落包裹起来

jQuery 代码:

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

elem描述:

用一个生成的div将所有段落包裹起来

jQuery 代码:

```
$("p").wrapAll(document.createElement("div"));
```

wrapInner(html|ele|fn)

返回值:jQuerywrapInner(htm|ele|fnl)

概述

将每一个匹配的元素的子内容(包括文本节点)用一个HTML结构包裹起来

这个函数的原理是检查提供的第一个元素（它是由所提供的HTML标记代码动态生成的），并在它的代码结构中找到最上层的祖先元素 - - 这个祖先元素就是包装元素。

参数

htmlString *V1.2*

HTML标记代码字符串，用于动态生成元素并包装目标元素

elemElement *V1.2*

用于包装目标元素的DOM元素

fnFunction *V1.4*

生成包裹结构的一个函数

示例

参数html描述:

把所有段落内的每个子内容加粗

jQuery 代码:

```
$("p").wrapInner("<b></b>");
```

参数elem描述:

把所有段落内的每个子内容加粗

jQuery 代码:

```
$("p").wrapInner(document.createElement("b"));
```

回调函数 描述:

用原先div的内容作为新div的class，并将每一个元素包裹起来

HTML 代码:

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

jQuery 代码:

```
$('.inner').wrapInner(function() {
  return '<div class="' + $(this).text() + '" />';
});
```

结果:

```
<div class="container">
  <div class="inner">
    <div class="Hello">Hello</div>
  </div>
  <div class="inner">
    <div class="Goodbye">Goodbye</div>
  </div>
</div>
```

replaceWith(content|fn)

返回值:jQueryreplaceWith(content|fn)

概述

将所有匹配的元素替换成指定的HTML或DOM元素。

参数

contentString, Element, jQuery, Function *V1.2*

用于将匹配元素替换掉的内容。如果这里传递一个函数进来的话，函数返回值必须是HTML字符串。

fnFunction *V1.4*

返回THML字符串，用来替换的内容。

示例

描述:

把所有的段落标记替换成加粗的标记。

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("p").replaceWith("<b>Paragraph. </b>");
```

结果:

```
<b>Paragraph. </b><b>Paragraph. </b><b>Paragraph. </b>
```

描述:

用第一段替换第三段，你可以发现他是移动到目标位置来替换，而不是复制一份来替换。

HTML 代码:

```
<div class="container">
  <div class="inner first">Hello</div>
  <div class="inner second">And</div>
  <div class="inner third">Goodbye</div>
</div>
```

jQuery 代码:

```
$('.third').replaceWith($('.first'));
```

结果:

```
<div class="container">
  <div class="inner second">And</div>
  <div class="inner first">Hello</div>
</div>
```

replaceAll(selector)

返回值:jQueryreplaceAll(selector)

概述

用匹配的元素替换掉所有 selector匹配到的元素。

在jQuery 1.3.2中，appendTo, prependTo, insertBefore, insertAfter, 和 replaceAll这个几个方法成为一个破坏性操作，要选择先前选中的元素，需要使用end()方法，参见 appendTo 方法的例二。

参数

selector选择器 *V1.2*

用于查找所要被替换的元素

示例

描述:

把所有的段落标记替换成加粗标记

HTML 代码:

```
<p>Hello</p><p>cruel</p><p>World</p>
```

jQuery 代码:

```
$("<b>Paragraph. </b>").replaceAll("p");
```

结果:

```
<b>Paragraph. </b><b>Paragraph. </b><b>Paragraph. </b>
```

empty()

返回值:jQueryempty()

*V1.0*概述

删除匹配的元素集合中所有的子节点。

示例

描述:

把所有段落的子元素（包括文本节点）删除

HTML 代码:

```
<p>Hello, <span>Person</span> <a href="#">and person</a></p>
```

jQuery 代码:

```
$("p").empty();
```

结果:

```
<p></p>
```

remove([expr])

返回值:jQueryremove(*[expr]*)

概述

从DOM中删除所有匹配的元素。

这个方法不会把匹配的元素从jQuery对象中删除，因而可以在将来再使用这些匹配的元素。但除了这个元素本身得以保留之外，其他的比如绑定的事件，附加的数据等都会被移除。

参数

exprString *V1.0*

用于筛选元素的jQuery表达式

示例

描述:

从DOM中把所有段落删除

HTML 代码:

```
<p>Hello</p> how are <p>you?</p>
```

jQuery 代码:

```
$("p").remove();
```

结果:

```
how are
```

描述:

从DOM中把带有hello类的段落删除

HTML 代码:

```
<p class="hello">Hello</p> how are <p>you?</p>
```

jQuery 代码:

```
$("p").remove(".hello");
```

结果:

```
how are <p>you?</p>
```

detach([expr])

返回值:jQuerydetach(*[expr]*)

概述

从DOM中删除所有匹配的元素。

这个方法不会把匹配的元素从jQuery对象中删除，因而可以在将来再使用这些匹配的元素。与remove()不同的是，所有绑定的事件、附加的数据等都会保留下来。

参数

exprString *V1.0*

用于筛选元素的jQuery表达式

示例

描述:

从DOM中把所有段落删除

HTML 代码:

```
<p>Hello</p> how are <p>you?</p>
```

jQuery 代码:

```
$("p").detach();
```

结果:

```
how are
```

描述:

从DOM中把带有hello类的段落删除

HTML 代码:

```
<p class="hello">Hello</p> how are <p>you?</p>
```

jQuery 代码:

```
$("p").detach(".hello");
```

结果:

```
how are <p>you?</p>
```

clone([Even[,deepEven]])

返回值:jQueryclone([Even[,deepEven]])

概述

克隆匹配的DOM元素并且选中这些克隆的副本。

在想把DOM文档中元素的副本添加到其他位置时这个函数非常有用。

参数

EventsBoolean *V1.0*

一个布尔值（true 或者 false）指示事件处理函数是否会被复制。V1.5以上版本默认值是：false

Events[,deepEvents]Boolean,Boolean *V1.5*

1:一个布尔值（true 或者 false）指示事件处理函数是否会被复制。

2:一个布尔值，指示是否对事件处理程序和克隆的元素的所有子元素的数据应该被复制。

示例

描述:

克隆所有b元素（并选中这些克隆的副本），然后将它们前置到所有段落中。

HTML 代码:

```
<b>Hello</b><p>, how are you?</p>
```

jQuery 代码:

```
$("b").clone().prependTo("p");
```

结果:

```
<b>Hello</b><p><b>Hello</b>, how are you?</p>
```

描述:

创建一个按钮，他可以复制自己，并且他的副本也有同样功能。

HTML 代码:

```
<button>Clone Me!</button>
```

jQuery 代码:

```
$("button").click(function(){
  $(this).clone(true).insertAfter(this);
});
```

CSS

css(name|pro|[,val|fn])

返回值:Stringcss(name|pro|[,val|fn])

概述

访问匹配元素的样式属性。

jQuery 1.8中，当你使用CSS属性在css()或animate()中，我们将根据浏览器自动加上前缀(在适当的时候)，比如("user-select", "none"); 在Chrome/Safari浏览器中我们将设置为"-webkit-user-select", Firefox会使用"-moz-user-select", IE10将使用"-ms-user-select".

参数

nameString *V1.0*

要访问的属性名称

propertiesMap *V1.0*

要设置为样式属性的名/值对

name,value String, Number *V1.4*

属性名,属性值

name,function(index, value) String,Function *V1.0*

1:属性名

2:此函数返回要设置的属性值。接受两个参数，index为元素在对象集合中的索引位置，value是原先的属性值。

示例

参数name 描述:

取得第一个段落的color样式属性的值。

jQuery 代码:

```
$("p").css("color");
```

参数properties 描述:

将所有段落的字体颜色设为红色并且背景为蓝色。

jQuery 代码:

```
$("p").css({ color: "#ff0011", background: "blue" });
```

参数name,value 描述:

将所有段落字体设为红色

jQuery 代码:

```
$("p").css("color","red");
```

参数name,回调函数 描述:

逐渐增加div的大小

jQuery 代码:

```
$("div").click(function() {
  $(this).css({
    width: function(index, value) {
      return parseFloat(value) * 1.2;
    },
    height: function(index, value) {
      return parseFloat(value) * 1.2;
    }
  });
});
```

offset([coordinates])

返回值:Object{top,left}offset([coordinates])

概述

获取匹配元素在当前视口的相对偏移。

返回的对象包含两个整型属性：top 和 left。此方法只对可见元素有效。

参数

coordinatesObject{top,left}, function(index, coords) *V1.2*

一个对象，必须包含top和left属性，作为元素的新坐标。这个参数也可以是一个返回一对坐标的函数，函数的第一个参数是元素的索引，第二个参数是当前的坐标。

示例

无参数描述:

获取第二段的偏移

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );
```

结果:

```
<p>Hello</p><p>left: 0, top: 35</p>
```

参数coordinates 描述:

获取第二段的偏移

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
$("p:last").offset({ top: 10, left: 30 });
```

position()

返回值:Object{top,left}position()

*V1.2*概述

获取匹配元素相对父元素的偏移。

返回的对象包含两个整型属性：top 和 left。为精确计算结果，请在补白、边框和填充属性上使用像素单位。此方法只对可见元素有效。

示例

描述:

获取第一段的偏移

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
var position = p.position();
$("p:last").html( "left: " + position.left + ", top: " + position.top );
```

结果:

```
<p>Hello</p><p>left: 15, top: 15</p>
```

scrollTop([val])

返回值:IntegerscrollTop([val])

概述

获取匹配元素相对滚动条顶部的偏移。

此方法对可见和隐藏元素均有效。

参数

valString, Number*V1.2.6*

设定垂直滚动条值

示例

无参数描述:

获取第一段相对滚动条顶部的偏移

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
$("p:last").text( "scrollTop:" + p.scrollTop() );
```

结果:

```
<p>Hello</p><p>scrollTop: 0</p>
```

参数val 描述:

设置相对滚动条顶部的偏移

jQuery 代码:

```
$("div.demo").scrollTop(300);
```

scrollLeft([val])

返回值:IntegerscrollLeft([val])

概述

获取匹配元素相对滚动条左侧的偏移。

此方法对可见和隐藏元素均有效。

参数

valString, Number*V1.2.6*

设定水平滚动条值

示例

无参数描述:

获取第一段相对滚动条左侧的偏移

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
$("p:last").text( "scrollLeft:" + p.scrollLeft() );
```

结果:

```
<p>Hello</p><p>scrollLeft: 0</p>
```

参数val 描述:

设置相对滚动条左侧的偏移

jQuery 代码:

```
$("div.demo").scrollLeft(300);
```

height([val|fn])

返回值:Integerheight([val|fn])

概述

取得匹配元素当前计算的高度值（px）。

在 jQuery 1.2 以后可以用来获取 window 和 document 的高

参数

valString, Number, Function *V1.0*

设定CSS中 'height' 的值，可以是字符串或者数字，还可以是一个函数，返回要设置的数值。函数接受两

个参数，第一个参数是元素在原先集合中的索引位置，第二个参数为原先的高度。

function(index, height) String, Number, Function *V1.4.1*

返回用于设置高度的一个函数。接收元素的索引位置和元素旧的高度值作为参数。

示例

无参数描述:

获取第一段的高

jQuery 代码:

```
$("p").height();
```

参数val 描述:

把所有段落的高设为 20:

jQuery 代码:

```
$("p").height(20);
```

参数function(index, height) 描述:

以 10 像素的幅度增加 p 元素的高度

jQuery 代码:

```
$("button").click(function(){
  $("p").height(function(n,c){
  return c+10;
  });
});
```

width([val|fn])

返回值:Integerwidth([val|fn])

概述

取得第一个匹配元素当前计算的宽度值（px）。

在 jQuery 1.2 以后可以用来获取 window 和 document 的宽

参数

valString, Number, Function *V1.0*

设定CSS中 'width' 的值，可以是字符串或者数字，还可以是一个函数，返回要设置的数值。函数接受两个参数，第一个参数是元素在原先集合中的索引位置，第二个参数为原先的宽度。

function(index, height) String, Number, Function *V1.4.1*

返回用于设置宽度的一个函数。接收元素的索引位置和元素旧的宽度值作为参数。

示例

无参数描述:

获取第一段的宽

jQuery 代码:

```
$("p").width();
```

参数val 描述:

把所有段落的宽设为 20:

jQuery 代码:

```
$("p").width(20);
```

参数function(index, height) 描述:

以 10 像素的幅度增加 p 元素的宽度

jQuery 代码:

```
$("button").click(function(){
  $("p").width(function(n,c){
  return c+10;
  });
});
```

innerHeight()

返回值:IntegerinnerHeight()

*V1.2.6*概述

获取第一个匹配元素内部区域高度（包括补白、不包括边框）。

此方法对可见和隐藏元素均有效。

示例

描述:

获取第一段落内部区域高度。

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
$("p:last").text( "innerHeight:" + p.innerHeight() );
```

结果:

```
<p>Hello</p><p>innerHeight: 16</p>
```

innerWidth()

返回值:IntegerinnerWidth()

*V1.2.6*概述

获取第一个匹配元素内部区域宽度（包括补白、不包括边框）。

此方法对可见和隐藏元素均有效。

示例

描述:

获取第一段落内部区域宽度。

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
$("p:last").text( "innerWidth:" + p.innerWidth() );
```

结果:

```
<p>Hello</p><p>innerWidth: 40</p>
```

outerHeight([options])

返回值:IntegerouterHeight([options])

概述

获取第一个匹配元素外部高度（默认包括补白和边框）。

此方法对可见和隐藏元素均有效。

参数

optionsBoolean*默认值:'false'V1.2.6*

设置为 true 时，计算边距在内。

示例

描述:

获取第一段落外部高度。

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
$("p:last").text( "outerHeight:" + p.outerHeight() + " , outerHeight(true):" + p.outerHeight(true) );
```

结果:

```
<p>Hello</p><p>outerHeight: 35 , outerHeight(true):55</p>
```

outerWidth([options])

返回值:IntegerouterWidth([options])

概述

获取第一个匹配元素外部宽度（默认包括补白和边框）。

此方法对可见和隐藏元素均有效。

参数

optionsBoolean*默认值:'false'V1.2.6*

设置为 true 时，计算边距在内。

示例

描述:

获取第一段落外部宽度。

HTML 代码:

```
<p>Hello</p><p>2nd Paragraph</p>
```

jQuery 代码:

```
var p = $("p:first");
$("p:last").text( "outerWidth:" + p.outerWidth() + " , outerWidth(true):" + p.outerWidth(true) );
```

结果:

```
<p>Hello</p><p>outerWidth: 65 , outerWidth(true):85</p>
```

事件

ready(fn)

返回值:jQueryready(fn)

概述

当DOM载入就绪可以查询及操纵时绑定一个要执行的函数。

这是事件模块中最重要的一个函数，因为它可以极大地提高web应用程序的响应速度。

简单地说，这个方法纯粹是对向window.load事件注册事件的替代方法。通过使用这个方法，可以在DOM载入就绪能够读取并操纵时立即调用你所绑定的函数，而99.99%的JavaScript函数都需要在那一刻执行。

有一个参数 - - 对jQuery函数的引用 - - 会传递到这个ready事件处理函数中。可以给这个参数任意起一个名字，并因此可以不再担心命名冲突而放心地使用$别名。

请确保在 元素的onload事件中没有注册函数，否则不会触发+$(document).ready()事件。

可以在同一个页面中无限次地使用$(document).ready()事件。其中注册的函数会按照（代码中的）先后顺序依次执行。

参数

fnFunction *V1.0*

要在DOM就绪时执行的函数

示例

描述:

在DOM加载完成时运行的代码，可以这样写：

jQuery 代码:

```
$(document).ready(function(){
  // 在这里写你的代码...
});
```

描述:

使用 $(document).ready() 的简写，同时内部的 jQuery 代码依然使用 $ 作为别名，而不管全局的 $ 为何。

jQuery 代码:

```
$(function($) {
  // 你可以在这里继续使用$作为别名...
});
```

on(events,[selector],[data],fn)

返回值:jQueryon(events,[selector],[data],fn)

概述

在选择元素上绑定一个或多个事件的事件处理函数。

on()方法绑定事件处理程序到当前选定的jQuery对象中的元素。在jQuery 1.7中，.on()方法 提供绑定事件处理程序所需的所有功能。帮助从旧的jQuery事件方法转换，see .bind(), .delegate(), 和 .live(). 要删除的.on()绑定的事件，请参阅.off()。要附加一个事件，只运行一次，然后删除自己， 请参阅.one()

参数

events,[selector],[data],fn *V1.7*

events:一个或多个用空格分隔的事件类型和可选的命名空间，如"click"或"keydown.myPlugin" 。

selector:一个选择器字符串用于过滤器的触发事件的选择器元素的后代。如果选择的

data:当一个事件被触发时要传递event.data给事件处理函数。

fn:该事件被触发时执行的函数。 false 值也可以做一个函数的简写，返回false。

events-map,[selector],[data] *V1.7*

events-map:个用字符串表示的，一个或多个空格分隔的事件类型和可选的命名空间，值表示事件绑定的处理函数。

selector:一个选择器字符串过滤选定的元素，该选择器的后裔元素将调用处理程序。如果选择是空或被忽略，当它到达选定的元素，事件总是触发。

data:当一个事件被触发时要传递event.data给事件处理函数。

示例

描述:

Display a paragraph's text in an alert when it is clicked:

```
$("p").on("click", function(){
alert( $(this).text() );
});
```

Pass data to the event handler, which is specified here by name:

```
function myHandler(event) {
alert(event.data.foo);
}
$("p").on("click", {foo: "bar"}, myHandler)
```

Cancel a form submit action and prevent the event from bubbling up by returning false:

```
$("form").on("submit", false)
```

Cancel only the default action by using .preventDefault().

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

Stop submit events from bubbling without preventing form submit, using .stopPropagation().

```
$("form").on("submit", function(event) {
  event.stopPropagation();
});
```

off(events,[selector],[fn])

返回值:jQueryoff(events,[selector],[fn])

概述

在选择元素上移除一个或多个事件的事件处理函数。

off() 方法移除用.on()绑定的事件处理程序。有关详细信息，请参阅该网页上delegated和directly绑定事件。特定的事件处理程序可以被移除元素上提供事件的名称，命名空间，选择器，或处理函数名称的组合。**当有多个过滤参数，所提供的参数都必须匹配的事件处理程序被删除。**

如果一个简单的事件名称，比如提供"click"，*所有*这种类型的事件（包括直接和委派）从jQuery设置的元素上删除。当编写代码，将作为一个插件使用，或者干脆当一个大的代码基础工作，最好的做法是安装和取下使用命名空间的事件，从而使代码不会无意中删除其他代码附加事件处理程序。在一个特定的命名空间中的所有类型的所有事件，可以从一个元素中删除，只是提供了一个命名空间，比如 ".myPlugin"。至少，无论是命名空间或事件名称必须提供。

要删除特定的委派事件处理程序，提供一个selector 的参数。选择器字符串必须是完全匹配递到.on()事件处理程序附加的选择器。要删除非委托元素上的所有事件，使用特殊值 "**" 。

处理程序也可以删除handler参数指定名称的函数。当jQuery的绑定一个事件处理程序，它分配一个唯一的ID给处理函数。函数用jQuery.proxy()代理或类似有相同的唯一ID机制（代理函数），因此，通过代理处理程序.off 可能会删除比预期更多的处理程序。在这些情况下，最好是附加和移除事件处理程序，使用命名空间。

和.on()一样，你可以传递一个 events-map>参数明确的指定而不是用events 和 handler作为单独参数。键事件和/或命名空间;值是处理函数或为false的特殊价值。

参数

events,[selector],[fn] *V1.7*

events:一个或多个空格分隔的事件类型和可选的命名空间，或仅仅是命名空间，比如"click", "keydown.myPlugin", 或者 ".myPlugin".

selector:一个最初传递到.on()事件处理程序附加的选择器。

fn:事件处理程序函数以前附加事件上，或特殊值false.

events-map,[selector] *V1.7*

events-map:一个用字符串表示的，一个或多个空格分隔的事件类型和可选的命名空间，值表示先前事件绑定的处理函数。

selector:一个最初传递到.on()事件处理程序附加的选择器。

示例

描述:

Remove all event handlers from all paragraphs:

```
$("p").off()
```

Remove all delegated click handlers from all paragraphs:

```
$("p").off( "click", "**" )
```

Remove just one previously bound handler by passing it as the third argument:

```
var foo = function () {
  // code to handle some kind of event
};

// ... now foo will be called when paragraphs are clicked ...
$("body").on("click", "p", foo);


// ... foo will no longer be called.
$("body").off("click", "p", foo);
```

Unbind all delegated event handlers by their namespace:

```
var validate = function () {
  // code to validate form entries
};

// delegate events under the ".validator" namespace
$("form").on("click.validator", "button", validate);

$("form").on("keypress.validator", "input[type='text']", validate);

// remove event handlers in the ".validator" namespace

$("form").off(".validator");
```

bind(type,[data],fn)

返回值:jQuerybind(type,[data],fn)

概述

为每个匹配元素的特定事件绑定事件处理函数。

参数

type,[data],function(eventObject)String,Object,Function*V1.0*

type:含有一个或多个事件类型的字符串，由空格分隔多个事件。比如"click"或"submit"，还可以是自定义事件名。

data:作为event.data属性值传递给事件对象的额外数据对象

fn:绑定到每个匹配元素的事件上面的处理函数

type,[data],false *String,Object,bool *V1.4.3*

type:含有一个或多个事件类型的字符串，由空格分隔多个事件。比如"click"或"submit"，还可以是自定义事件名。

data:作为event.data属性值传递给事件对象的额外数据对象

false: 将第三个参数设置为false会使默认的动作失效。

eventsString *V1.4*

一个或多个事件类型的字符串和函数的数据映射来执行他们。

示例

描述:

当每个段落被点击的时候，弹出其文本。

jQuery 代码:

```
$("p").bind("click", function(){
  alert( $(this).text() );
});
```

描述:

同时绑定多个事件类型

jQuery 代码:

```
$('#foo').bind('mouseenter mouseleave', function() {
  $(this).toggleClass('entered');
});
```

描述:

同时绑定多个事件类型/处理程序

jQuery 代码:

```
$("button").bind({
  click:function(){$("p").slideToggle();},
  mouseover:function(){$("body").css("background-color","red");},
  mouseout:function(){$("body").css("background-color","#FFFFFF");}
});
```

描述:

你可以在事件处理之前传递一些附加的数据。

jQuery 代码:

```
function handler(event) {
  alert(event.data.foo);
}
$("p").bind("click", {foo: "bar"}, handler)
```

描述:

通过返回false来取消默认的行为并阻止事件起泡。

jQuery 代码:

```
$("form").bind("submit", function() { return false; })
```

描述:

通过使用 preventDefault() 方法只取消默认的行为。

jQuery 代码:

```
$("form").bind("submit", function(event){
  event.preventDefault();
});
```

描述:

通过使用 stopPropagation() 方法只阻止一个事件起泡。

jQuery 代码:

```
$("form").bind("submit", function(event){
  event.stopPropagation();
});
```

one(type,[data],fn)

返回值:jQueryone(type,[data],fn)

概述

为每一个匹配元素的特定事件（像click）绑定一个一次性的事件处理函数。

在每个对象上，这个事件处理函数只会被执行一次。其他规则与bind()函数相同。这个事件处理函数会接收到一个事件对象，可以通过它来阻止（浏览器）默认的行为。如果既想取消默认的行为，又想阻止事件起泡，这个事件处理函数必须返回false。

多数情况下，可以把事件处理函数定义为匿名函数（见示例一）。在不可能定义匿名函数的情况下，可以传递一个可选的数据对象作为第二个参数（而事件处理函数则作为第三个参数），见示例二。

参数

type,[data],fnString,Object,Function *V1.1*

type:添加到元素的一个或多个事件。由空格分隔多个事件。必须是有效的事件。

data:将要传递给事件处理函数的数据映射

fn:每当事件触发时执行的函数。

示例

描述:

当所有段落被第一次点击的时候，显示所有其文本。

jQuery 代码:

```
$("p").one("click", function(){
  alert( $(this).text() );
});
```

trigger(type,[data])

返回值:jQuerytrigger(type,[data])

概述

在每一个匹配的元素上触发某类事件。

这个函数也会导致浏览器同名的默认行为的执行。比如，如果用trigger()触发一个'submit'，则同样会导致浏览器提交表单。如果要阻止这种默认行为，应返回false。

你也可以触发由bind()注册的自定义事件而不限于浏览器默认事件。

事件处理函数会收到一个修复的(规范化的)事件对象，但这个对象没有特定浏览器才有的属性，比如keyCode。

jQuery也支持 命名空间事件。这允许你触发或者解除绑定一组特定的事件处理函数，而无需一一个指定。你可以在事件类型后面加上感叹号！来只触发那些没有命名空间的事件处理函数。

jQuery 1.3中新增：

所有触发的事件现在会冒泡到DOM树上了。举例来说，如果你在一个段落p上触发一个事件，他首先会在这个元素上触发，其次到父元素，在到父元素的父元素，直到触发到document对象。这个事件对象有一个 .target 属性指向最开始触发这个事件的元素。你可以用 stopPropagation() 来阻止事件冒泡，或者在事件处理函数中返回false即可。

事件对象构造器现在已经公开，并且你可以自行创建一个事件对象。这个事件对象可以直接传递给trigger所触发的事件处理函数。事件对象的完整属性列表可以在 jQuery.Event 的文档里找到。

你可以有三种方式指定事件类型：

- 你可以传递字符串型的事件名称(type参数)。
- 你可以使用jQuery.Event对象。可以将数据放进这个对象，并且这个对象可以被触发的事件处理函数获取到。
- 最后，你可以传递一个带有数据的字面量对象。他将被复制到真正的jQuery.Event对象上去。 注意在这种情况下你'''必须'''指定一个 *type* 属性。

参数

type,[data]String|Event,Array *V1.0*

type:一个事件对象或者要触发的事件类型

data:传递给事件处理函数的附加参数

event **Object*V1.3**

事件发生时运行的函数

示例

描述:

提交第一个表单，但不用submit()

jQuery 代码:

```
$("form:first").trigger("submit")
```

描述:

给一个事件传递参数

jQuery 代码:

```
$("p").click( function (event, a, b) {
  // 一个普通的点击事件时，a和b是undefined类型
  // 如果用下面的语句触发，那么a指向"foo",而b指向"bar"
} ).trigger("click", ["foo", "bar"]);
```

描述:

下面的代码可以显示一个"Hello World"

jQuery 代码:

```
$("p").bind("myEvent", function (event, message1, message2) {
  alert(message1 + ' ' + message2);
});
$("p").trigger("myEvent", ["Hello","World!"]);
```

triggerHandler(type,[data])

返回值:jQuerytriggerHandler(type, *[data]*)

概述

这个特别的方法将会触发指定的事件类型上所有绑定的处理函数。但不会执行浏览器默认动作，也不会产生事件冒泡。

这个方法的行为表现与trigger类似，但有以下三个主要区别：

- 第一，他不会触发浏览器默认事件。
- 第二，只触发jQuery对象集合中第一个元素的事件处理函数。
- 第三，这个方法的返回的是事件处理函数的返回值，而不是据有可链性的jQuery对象。此外，如果最

开始的jQuery对象集合为空，则这个方法返回 undefined 。

参数

type,[data]String,Array *V1.2*

type:要触发的事件类型

data:传递给事件处理函数的附加参数

示例

描述:

如果你对一个focus事件执行了 .triggerHandler() ，浏览器默认动作将不会被触发，只会触发你绑定的动作。

HTML 代码:

```
<button id="old">.trigger("focus")</button>
<button id="new">.triggerHandler("focus")</button><br/><br/>
<input type="text" value="To Be Focused"/>
```

jQuery 代码:

```
$("#old").click(function(){
  $("input").trigger("focus");
});
$("#new").click(function(){
  $("input").triggerHandler("focus");
});
$("input").focus(function(){
  $("<span>Focused!</span>").appendTo("body").fadeOut(1000);
});
```

unbind(type,[data|fn])

返回值:jQueryunbind(type,[data|fn]])

概述

bind()的反向操作，从每一个匹配的元素中删除绑定的事件。

如果没有参数，则删除所有绑定的事件。

你可以将你用bind()注册的自定义事件取消绑定。

如果提供了事件类型作为参数，则只删除该类型的绑定事件。

如果把在绑定时传递的处理函数作为第二个参数，则只有这个特定的事件处理函数会被删除。

参数

type,[fn]String,Function *V1.0*

type:删除元素的一个或多个事件,由空格分隔多个事件值。

fn:要从每个匹配元素的事件中反绑定的事件处理函数

type,falseString,bool *V1.4.3*

type:删除元素的一个或多个事件,由空格分隔多个事件值

false:设置为false会使默认的动作失效。

eventObjString *V1.0*

事件对象。这个 eventObj 参数来自事件绑定函数

示例

描述:

把所有段落的所有事件取消绑定

jQuery 代码:

```
$("p").unbind()
```

描述:

将段落的click事件取消绑定

jQuery 代码:

```
$("p").unbind( "click" )
```

描述:

删除特定函数的绑定，将函数作为第二个参数传入

jQuery 代码:

```
var foo = function () {
  // 处理某个事件的代码
};

$("p").bind("click", foo); // ... 当点击段落的时候会触发 foo

$("p").unbind("click", foo); // ... 再也不会被触发 foo
```

live(type,[data],fn)

返回值:jQuerylive(type, *[data]*, fn)

概述

jQuery 给所有匹配的元素附加一个事件处理函数，即使这个元素是以后再添加进来的也有效。

这个方法是基本是的 .bind() 方法的一个变体。使用 .bind() 时，选择器匹配的元素会附加一个事件处理函数，而以后再添加的元素则不会有。为此需要再使用一次 .bind() 才行。比如说

```
<body>
  <div class="clickme">Click here</div>
</body>
```

可以给这个元素绑定一个简单的click事件：

```
$('.clickme').bind('click', function() {
  alert("Bound handler called.");
});
```

当点击了元素，就会弹出一个警告框。然后，想象一下这之后有另一个元素添加进来了。

```
$('body').append('<div class="clickme">Another target</div>');
```

尽管这个新的元素也能够匹配选择器 ".clickme" ，但是由于这个元素是在调用 .bind() 之后添加的，所以点击这个元素不会有任何效果。

.live() 就提供了对应这种情况的方法。如果我们是这样绑定click事件的：

```
$('.clickme').live('click', function() {
  alert("Live handler called.");
});
```

然后再添加一个新元素：

```
$('body').append('<div class="clickme">Another target</div>');
```

然后再点击新增的元素，他依然能够触发事件处理函数。

事件委托

.live() 方法能对一个还没有添加进DOM的元素有效，是由于使用了事件委托：绑定在祖先元素上的事件处理函数可以对在后代上触发的事件作出回应。传递给 .live() 的事件处理函数不会绑定在元素上，而是把他作为一个特殊的事件处理函数，绑定在 DOM 树的根节点上。在我们的例子中，当点击新的元素后，会依次发生下列步骤：

1. 生成一个click事件传递给
   来处理 由于没有事件处理函数直接绑定在 事件不断冒泡一直到DOM树的根节点，默认情况下上面绑定了这个特殊的事件处理函数。
   执行由 .live() 绑定的特殊的 click 事件处理函数。
   这个事件处理函数首先检测事件对象的 target 来确定是不是需要继续。这个测试是通过检测 $(event.target).closest('.clickme') 能否找到匹配的元素来实现的。
   如果找到了匹配的元素，那么调用原始的事件处理函数。

由于只有在事件发生时才会在上面的第五步里做测试，因此在任何时候添加的元素都能够响应这个事件。

附加说明

.live() 虽然很有用，但由于其特殊的实现方式，所以不能简单的在任何情况下替换 .bind()。主要的不同有：

- 在jQuery 1.4中，.live()方法支持自定义事件，也支持所有的 JavaScript 事件。在jQuery 1.4.1中，甚至也支持 focus 和 blue 事件了（映射到更合适，并且可以冒泡的focusin和focusout上）。另外，在jQuery 1.4.1中，也能支持hover（映射到"mouseenter mouseleave"）。然而在jQuery 1.3.x中，只支持支持的JavaScript事件和自定义事件：click, dblclick, keydown, keypress, keyup, mousedown, mousemove, mouseout, mouseover, 和 mouseup.
- .live() 并不完全支持通过DOM遍历的方法找到的元素。取而代之的是，应当总是在一个选择器后面直接使用 .live() 方法，正如前面例子里提到的。
- 当一个事件处理函数用 .live() 绑定后，要停止执行其他的事件处理函数，那么这个函数必须返回 false。 仅仅调用 .stopPropagation() 无法实现这个目的。

参考 .bind() 方法可以获得更多关于事件绑定的信息。

在jQuery 1.4.1 中，.live() 能接受多个，用空间分隔事件，在提供类似.bind()的功能 。例如，我们可以“live ” 同时绑定mouseover和mouseout事件,像这样：

```
$('.hoverme').live('mouseover mouseout', function(event) {
  if (event.type == 'mouseover') {
    // do something on mouseover
  } else {
    // do something on mouseout
  }
});
```

在jQuery 1.4.3中:你可以绑定一个或多个事件类型的字符串和函数的数据映射来执行他们

```
$("a").live({
  click: function() {
    // do something on click
  },
  mouseover: function() {
    // do something on mouseover
  }
});
```

在jQuery 1.4 中，data参数可以用于把附加信息传递给事件处理函数。一个很好的用处是应付由闭包导致的问题。可以参考 .bind() 的讨论来获得更多信息。

在jQuery 1.4 中， live事件可以绑定到“context”DOM元素，而不是默认的文档的根。要设置此背景下，我们通过在一个单一的DOM元素(而不是一个jQuery集合或选择器)使用jQuery() function's second argument。

```
$('div.clickme', $('#container')[0]).live('click', function() {
  // Live handler called.
});
```

参数

type,[fn]String,Function *V1.3*

type:一个或多个事件类型，由空格分隔多个事件。

fn:要从每个匹配元素的事件中反绑定的事件处理函数

type,[data],falseString,Array,bool *V1.4*

type:一个或多个事件类型，由空格分隔多个事件。

data:传递给事件处理函数的附加参数

false:设置为false会使默认的动作失效。

eventString *V1.4.3*

一个或多个事件类型的字符串和函数的数据映射来执行他们

示例

描述:

点击生成的p依然据有同样的功能。

HTML 代码:

```
<p>Click me!</p>
```

jQuery 代码:

```
$("p").live("click", function(){
    $(this).after("<p>Another paragraph!</p>");
});
```

描述:

阻止默认事件行为和事件冒泡，返回 false

jQuery 代码:

```
$("a").live("click", function() { return false; });
```

描述:

仅仅阻止默认事件行为

jQuery 代码:

```
$("a").live("click", function(event){
  event.preventDefault();
});
```

die(type,[fn])

返回值:jQuerydie(type, *[fn]*)

概述

从元素中删除先前用.live()绑定的所有事件.(此方法与live正好完全相反。)

如果不带参数，则所有绑定的live事件都会被移除。

你可以解除用live注册的自定义事件。

如果提供了type参数，那么会移除对应的live事件。

如果也指定了第二个参数function,则只移出指定的事件处理函数。

参数

type[,fn] *String,Function*V1.3*

type:要移除的一个或多个事件处理程序。由空格分隔多个事件值。必须是有效的事件。

fn:要移除的函数。。

type *String*V1.4.3*

要移除的一个或多个事件处理程序。由空格分隔多个事件值。必须是有效的事件。

示例

描述:

给按钮解除click事件

jQuery 代码:

```
function aClick() {
    $("div").show().fadeOut("slow");
 }
 $("#unbind").click(function () {
    $("#theone").die("click", aClick)
 });
```

delegate(sel,[type],[data],fn)

返回值:jQuery delegate(selector,[type],[data],fn)

概述

指定的元素（属于被选元素的子元素）添加一个或多个事件处理程序，并规定当这些事件发生时运行的函数。

使用 delegate() 方法的事件处理程序适用于当前或未来的元素（比如由脚本创建的新元素）。

参数

selector,[type],fnString,String,Function *V1.4.2*

selector:选择器字符串，用于过滤器触发事件的元素。

type:附加到元素的一个或多个事件。由空格分隔多个事件值。必须是有效的事件。

fn:当事件发生时运行的函数

selector,[type],[data],fnString,String,Object,Function *V1.4.2*

selector:选择器字符串，用于过滤器触发事件的元素。

type:附加到元素的一个或多个事件。由空格分隔多个事件值。必须是有效的事件。

data:传递到函数的额外数据

fn:当事件发生时运行的函数

selector,eventsString,String *V1.4.3*

selector:选择器字符串，用于过滤器触发事件的元素。

events:一个或多个事件类型的字符串和函数的数据映射来执行他们。

示例

描述:

当点击鼠标时，隐藏或显示 p 元素：

HTML 代码:

```
<div style="background-color:red">
<p>这是一个段落。</p>
<button>请点击这里</button>
</div>
```

jQuery 代码:

```
$("div").delegate("button","click",function(){
  $("p").slideToggle();
});
```

描述: delegate这个方法可作为live()方法的替代，使得每次事件绑定到特定的DOM元素。

以下两段代码是等同的:

```
$("table").delegate("td", "hover", function(){    $(this).toggleClass("hover");
});
```

```
$("table").each(function(){    $("td", this).live("hover", function(){          $(this).toggleClass("hover");    });

});
```

undelegate([sel,[type],fn])

返回值:jQuery undelegate([sel,[type],fn])

概述

删除由 delegate() 方法添加的一个或多个事件处理程序。

参数

selector,[type]String,String *1.4.2*

selector:需要删除事件处理程序的选择器。

type:需要删除处理函数的一个或多个事件类型。由空格分隔多个事件值。必须是有效的事件。

selector,[type],fnString,String,Function *V1.4.2*

selector:需要删除事件处理程序的选择器。

type:需要删除处理函数的一个或多个事件类型。由空格分隔多个事件值。必须是有效的事件。

fn:要删除的具体事件处理函数。

selector,eventsString,String *V1.4.3*

selector:需要删除事件处理程序的选择器。

events:一个或多个事件类型的字符串和函数的数据映射来执行他们。

namespaceString *1.6*

namespace:一个字符串，其中包含一个命名空间取消绑定的所有事件。

示例

描述:

从p元素删除由 delegate() 方法添加的所有事件处理器：

jQuery 代码:

```
$("p").undelegate();
```

从p元素删除由 delegate() 方法添加的所有click事件处理器：

jQuery 代码:

```
$("p").undelegate( "click" )
```

从form元素删除由 delegate() 方法添加的".whatever"命名空间：

jQuery 代码:

```
var foo = function () {
  //.....
};
//用delegate() 方法给click事件增加".whatever"命名空间
$("form").delegate(":button", "click.whatever", foo);
$("form").delegate("input[type='text']", "keypress.whatever", foo);

// 用undelegate()方法删除delegate()方法增加的".whatever"命名空间
$("form").undelegate(".whatever");
```

hover([over,]out)

返回值:jQueryhover([over,]out)

概述

一个模仿悬停事件（鼠标移动到一个对象上面及移出这个对象）的方法。这是一个自定义的方法，它为频繁使用的任务提供了一种“保持在其中”的状态。

当鼠标移动到一个匹配的元素上面时，会触发指定的第一个函数。当鼠标移出这个元素时，会触发指定的第二个函数。而且，会伴随着对鼠标是否仍然处在特定元素中的检测（例如，处在div中的图像），如果是，则会继续保持“悬停”状态，而不触发移出事件（修正了使用mouseout事件的一个常见错误）。

参数

over,out Function,Function *V1.0*

over:鼠标移到元素上要触发的函数

out:鼠标移出元素要触发的函数

outObject *V1.4*

当鼠标移到元素上或移出元素时触发执行的事件函数

示例

over,out 描述:

鼠标悬停的表格加上特定的类

jQuery 代码:

```
$("td").hover(
  function () {
    $(this).addClass("hover");
  },
  function () {
    $(this).removeClass("hover");
  }
);
```

out 描述:

hovre()方法通过绑定变量"handlerInOut"来切换方法。

jQuery 代码:

```
$("
```

td

```
").bind("mouseenter mouseleave",handlerInOut);
$("
```

td

```
").hover(handlerInOut);
```

toggle(fn, fn2, [fn3, fn4, ...])

返回值:jQuerytoggle([speed],[easing],[fn])

概述

用于绑定两个或多个事件处理器函数，以响应被选元素的轮流的 click 事件。

如果元素是可见的，切换为隐藏的；如果元素是隐藏的，切换为可见的。

参数

fn,fn2,[fn3,fn4,...]Function,.... *V1.0*

fn:第一数次点击时要执行的函数。

fn2:第二数次点击时要执行的函数。

fn3,fn4,....:更多次点击时要执行的函数。

[speed] [,fn]String,Function *V1.0*

speed: 隐藏/显示 效果的速度。默认是 "0"毫秒。可能的值：slow，normal，fast。"

fn:在动画完成时执行的函数，每个元素执行一次。

*[speed], [easing], [fn] *String,String,Function *V1.4.3*

speed: 隐藏/显示 效果的速度。默认是 "0"毫秒。可能的值：slow，normal，fast。"

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

*switch*Boolean V1.3*

用于确定显示/隐藏的开关。如：true - 显示元素，false - 隐藏元素

示例

无参数描述:

对表格切换显示/隐藏

jQuery 代码:

```
$('td).toggle();
```

fn,fn2描述:

对表格的切换一个类

jQuery 代码:

```
$("td").toggle(
  function () {
    $(this).addClass("selected");
  },
  function () {
    $(this).removeClass("selected");
  }
);
```

speed 描述:

用600毫秒的时间将段落缓慢的切换显示状态

jQuery 代码:

```
$("p").toggle("slow");
```

speed,fn 描述:

用200毫秒将段落迅速切换显示状态，之后弹出一个对话框。

jQuery 代码:

```
$("p").toggle("fast",function(){
   alert("Animation Done.");
 });
```

switch参数描述:

如果这个参数为true ，那么匹配的元素将显示;如果false ，元素是隐藏的

jQuery 代码:

```
$('#foo').toggle(showOrHide);

//相当于
if (showOrHide) {
  $('#foo').show();
} else {
  $('#foo').hide();
}
```

blur([[data],fn])

返回值:jQueryblur([[data],fn])

概述

触发每一个匹配元素的blur事件。

这个函数会调用执行绑定到blur事件的所有函数，包括浏览器的默认行为。可以通过返回false来防止触发浏览器的默认行为。blur事件会在元素失去焦点的时候触发，既可以是鼠标行为，也可以是按tab键离开的

参数

fnFunction *V1.0*

在每一个匹配元素的blur事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:blur([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的blur事件中绑定的处理函数。

示例

描述:

触发所有段落的blur事件

jQuery 代码:

```
$("p").blur();
```

描述:

任何段落失去焦点时弹出一个 "Hello World!"在每一个匹配元素的blur事件中绑定的处理函数。

jQuery 代码:

```
$("p").blur( function () { alert("Hello World!"); } );
```

change([[data],fn])

返回值:jQuerychange([[data],fn])

概述

触发每个匹配元素的change事件

这个函数会调用执行绑定到change事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。change事件会在元素失去焦点的时候触发，也会当其值在获得焦点后改变时触发。

参数

fnFunction *V1.0*

在每一个匹配元素的change事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:change([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的change事件中绑定的处理函数。

示例

描述:

触发被选元素的 change 事件。

jQuery 代码:

```
$(selector).change();
```

描述:

给所有的文本框增加输入验证

jQuery 代码:

```
$("input[type='text']").change( function() {
  // 这里可以写些验证代码
});
```

click([[data],fn])

返回值:jQueryclick([[data],fn])

概述

触发每一个匹配元素的click事件。

这个函数会调用执行绑定到click事件的所有函数。

参数

fnFunction *V1.0*

在每一个匹配元素的click事件中绑定的处理函数。

[data],fnString,Function *V1.4.3*

data:click([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的click事件中绑定的处理函数。

示例

描述:

触发页面内所有段落的点击事件

jQuery 代码:

```
$("p").click();
```

描述:

将页面内所有段落点击后隐藏。

jQuery 代码:

```
$("p").click( function () { $(this).hide(); });
```

dblclick([[data],fn])

返回值:jQuerydblclick([[data],fn])

概述

触发每一个匹配元素的dblclick事件。

这个函数会调用执行绑定到dblclick事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数

中返回false来防止触发浏览器的默认行为。dblclick事件会在元素的同一点双击时触发。

参数

fnFunction *V1.0*

在每一个匹配元素的dblclick事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:dblclick([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的dblclick事件中绑定的处理函数。

示例

描述:

给页面上每个段落的双击事件绑上 "Hello World!" 警告框

jQuery 代码:

```
$("p").dblclick( function () { alert("Hello World!"); });
```

error([[data],fn])

返回值:jQueryerror([[data],fn])

概述

触发每一个匹配元素的error事件。

这个函数会调用所有绑定到error事件上的函数，包括在对应元素上的浏览器默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。error事件通常可以在元素由于点击或者tab导航失去焦点时触发。

对于error事件，没有一个公众的标准。在大多数浏览器中，当页面的JavaScript发生错误时，window对象会触发error事件;当图像的src属性无效时，比如文件不存在或者图像数据错误时，也会触发图像对象的error事件。

如果异常是由window对象抛出，事件处理函数将会被传入三个参数：

1. 描述事件的信息 ("varName is not defined", "missing operator in expression", 等等.),
2. 包含错误的文档的完整URL

3. 异常发生的行数 如果事件处理函数返回true，则表示事件已经被处理，浏览器将认为没有异常。

更多相关信息:

msdn - onerror Event

Gecko DOM Reference - onerror Event

Gecko DOM Reference - Event object

Wikipedia: DOM Events<

参数

fnFunction *V1.0*

在每一个匹配元素的error事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:error([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的error事件中绑定的处理函数。

示例

描述:

在服务器端记录JavaScript错误日志:

jQuery 代码:

```
$(window).error(function(msg, url, line){
  jQuery.post("js_error_log.php", { msg: msg, url: url, line: line });
});
```

描述:

隐藏JavaScript错误:

jQuery 代码:

```
$(window).error(function(){
  return true;
});
```

描述:

给你IE的用户隐藏无效的图像:

jQuery 代码:

```
$("img").error(function(){
  $(this).hide();
});
```

focus([[data],fn])

返回值:jQueryfocus([[data],fn])

概述

触发每一个匹配元素的focus事件。

可以通过鼠标点击或者键盘上的TAB导航触发。这将触发所有绑定的focus函数，注意，某些对象不支持focus方法。

参数

fnFunction *V1.0*

在每一个匹配元素的focus事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:focus([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的focus事件中绑定的处理函数。

示例

描述:

当页面加载后将 id 为 'login' 的元素设置焦点:

jQuery 代码:

```
$(document).ready(function(){
  $("#login").focus();
});
```

描述:

使人无法使用文本框:

jQuery 代码:

```
$("input[type=text]").focus(function(){
  this.blur();
});
```

focusout([data],fn)

返回值:jQueryfocusout([data],fn)

概述

在每一个匹配元素的focusout事件中绑定一个处理函数。

当一个元素，或者其内部任何一个元素失去焦点的时候会触发这个事件。这跟blur事件区别在于，他可以在父元素上检测子元素失去焦点的情况。

参数

fnFunction *V1.0*

在每一个匹配元素的focusout事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:focusout([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的focusout事件中绑定的处理函数。

示例

描述:

失去焦点后会触发动画:

HTML 代码:

```
<p><input type="text" /> <span>focusout fire</span></p>
<p><input type="password" /> <span>focusout fire</span></p>
```

jQuery 代码:

```
$("p").focusout(function() {
  $(this).find("span").css('display','inline').fadeOut(1000);
});
```

keydown([[data],fn])

返回值:jQuerykeydown([[data],fn])

概述

触发每一个匹配元素的keydown事件

这个函数会调用执行绑定到keydown事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。keydown事件会在键盘按下时触发。

参数

fnFunction *V1.0*

在每一个匹配元素的keydown事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:keydown([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的keydown事件中绑定的处理函数。

示例

描述:

在页面内对键盘按键做出回应，可以使用如下代码:

jQuery 代码:

```
$(window).keydown(function(event){
  switch(event.keyCode) {
    // ...
    // 不同的按键可以做不同的事情
    // 不同的浏览器的keycode不同
    // 更多详细信息:     http://unixpapa.com/js/key.html
    // ...
  }
});
```

keypress([[data],fn])

返回值:jQuerykeypress([[data],fn])

概述

触发每一个匹配元素的keypress事件

T这个函数会调用执行绑定到keydown事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。keydown事件会在键盘按下时触发

参数

fnFunction *V1.0*

在每一个匹配元素的keypress事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:keypress([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的keypress事件中绑定的处理函数。

示例

描述:

计算在输入域中的按键次数：

jQuery 代码:

```
$("input").keydown(function(){  $("span").text(i+=1);});
```

keyup([[data],fn])

返回值:jQuerykeyup([[data],fn])

概述

触发每一个匹配元素的keyup事件

这个函数会调用执行绑定到keyup事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数

中返回false来防止触发浏览器的默认行为。keyup事件会在按键释放时触发。

参数

fnFunction *V1.0*

在每一个匹配元素的keyup事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:keyup([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的keyup事件中绑定的处理函数。

示例

描述:

当按下按键时，改变文本域的颜色：

jQuery 代码:

```
$("input").keyup(function(){
  $("input").css("background-color","#D6D6FF");
});
```

mousedown([[data],fn])

返回值:jQuerymousedown([[data],fn])

概述

在每一个匹配元素的mousedown事件中绑定一个处理函数。

mousedown事件在鼠标在元素上点击后会触发

参数

fnFunction *V1.0*

在每一个匹配元素的mousedown事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mousedown([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mousedown事件中绑定的处理函数。

示例

描述:

当按下鼠标按钮时，隐藏或显示元素：

jQuery 代码:

```
$("button").mousedown(function(){
  $("p").slideToggle();
});
```

mouseenter([[data],fn])

返回值:jQuerymouseenter([[data],fn])

概述

当鼠标指针穿过元素时，会发生 mouseenter 事件。该事件大多数时候会与mouseleave 事件一起使用。

与 mouseover 事件不同，只有在鼠标指针穿过被选元素时，才会触发 mouseenter 事件。如果鼠标指针穿过任何子元素，同样会触发 mouseover 事件。

参数

fnFunction *V1.0*

在每一个匹配元素的mouseenter事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mouseenter([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mouseenter事件中绑定的处理函数。

示例

描述:

当鼠标指针进入（穿过）元素时，改变元素的背景色：

jQuery 代码:

```
$("p").mouseenter(function(){
  $("p").css("background-color","yellow");
});
```

mouseleave([[data],fn])

返回值:jQuerymouseleave([[data],fn])

概述

当鼠标指针离开元素时，会发生 mouseleave 事件。该事件大多数时候会与mouseenter 事件一起使用。

与 mouseout 事件不同，只有在鼠标指针离开被选元素时，才会触发 mouseleave 事件。如果鼠标指针离开任何子元素，同样会触发 mouseout 事件。

参数

fnFunction *V1.0*

在每一个匹配元素的mouseleave事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mouseleave([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mouseleave事件中绑定的处理函数。

示例

描述:

当鼠标指针离开元素时，改变元素的背景色：：

jQuery 代码:

```
$("p").mouseleave(function(){
  $("p").css("background-color","#E9E9E4");
});
```

mousemove([[data],fn])

返回值:jQuerymousemove([[data],fn])

概述

在每一个匹配元素的mousemove事件中绑定一个处理函数。

mousemove 事件通过鼠标在元素上移动来触发。事件处理函数会被传递一个变量¯事件对象，其.clientX 和 .clientY 属性代表鼠标的坐标

参数

fnFunction *V1.0*

在每一个匹配元素的mousemove事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mousemove([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mousemove事件中绑定的处理函数。

示例

描述:

获得鼠标指针在页面中的位置：

jQuery 代码:

```
$(document).mousemove(function(e){
  $("span").text(e.pageX + ", " + e.pageY);
});
```

mouseout([[data],fn])

返回值:jQuerymouseout([[data],fn])

概述

在每一个匹配元素的mouseout事件中绑定一个处理函数。

mouseout事件在鼠标从元素上离开后会触发

参数

fnFunction *V1.0*

在每一个匹配元素的mouseout事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mouseout([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mouseout事件中绑定的处理函数。

示例

描述:

当鼠标从元素上移开时，改变元素的背景色：

jQuery 代码:

```
$("p").mouseout(function(){
  $("p").css("background-color","#E9E9E4");
});
```

mouseover([[data],fn])

返回值:jQuerymouseover([[data],fn])

概述

在每一个匹配元素的mouseover事件中绑定一个处理函数。

mouseover事件会在鼠标移入对象时触发

参数

fnFunction *V1.0*

在每一个匹配元素的mouseover事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mouseover([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mouseover事件中绑定的处理函数。

示例

描述:

当鼠标指针位于元素上方时时，改变元素的背景色：

jQuery 代码:

```
$("p").mouseover(function(){
  $("p").css("background-color","yellow");
});
```

mouseup([[data],fn])

返回值:jQuerymouseup([[data],fn])

概述

在每一个匹配元素的mouseup事件中绑定一个处理函数。

mouseup事件会在鼠标点击对象释放时

参数

fnFunction *V1.0*

在每一个匹配元素的mouseup事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:mouseup([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的mouseup事件中绑定的处理函数。

示例

描述:

当松开鼠标按钮时，隐藏或显示元素：

jQuery 代码:

```
$("button").mouseup(function(){
  $("p").slideToggle();
});
```

resize([[data],fn])

返回值:jQueryresize([[data],fn])

概述

在每一个匹配元素的resize事件中绑定一个处理函数。

当文档窗口改变大小时触发

参数

fnFunction *V1.0*

在每一个匹配元素的resize事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:resize([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的resize事件中绑定的处理函数。

示例

描述:

让人每次改变页面窗口的大小时很郁闷的方法:

jQuery 代码:

```
$(window).resize(function(){
  alert("Stop it!");
});
```

描述:

对浏览器窗口调整大小进行计数：

jQuery 代码:

```
$(window).resize(function() {
  $('span').text(x+=1);
});
```

scroll([[data],fn])

返回值:jQueryscroll([[data],fn])

概述

在每一个匹配元素的scroll事件中绑定一个处理函数。

当滚动条发生变化时触发

参数

fnFunction *V1.0*

在每一个匹配元素的scroll事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:scroll([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的scroll事件中绑定的处理函数。

示例

描述:

当页面滚动条变化时，执行的函数:

jQuery 代码:

```
$(window).scroll( function() { /* ...do something... */ } );
```

描述:

对元素滚动的次数进行计数：

jQuery 代码:

```
$("div").scroll(function() {
  $("span").text(x+=1);
});
```

select([[data],fn])

返回值:jQueryselect([[data],fn])

概述

触发每一个匹配元素的select事件

这个函数会调用执行绑定到select事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。

参数

fnFunction *V1.0*

在每一个匹配元素的select事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:select([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的select事件中绑定的处理函数。

示例

描述:

触发所有input元素的select事件:

jQuery 代码:

```
$("input").select();
```

描述:

当文本框中文本被选中时执行的函数:

jQuery 代码:

```
$(":text").select( function () { /* ...do something... */ } );
```

submit([[data],fn])

返回值:jQuerysubmit([[data],fn])

概述

触发每一个匹配元素的submit事件。

这个函数会调用执行绑定到submit事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。

参数

fnFunction *V1.0*

在每一个匹配元素的submit事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:submit([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的submit事件中绑定的处理函数。

示例

描述:

提交本页的第一个表单:

jQuery 代码:

```
$("form:first").submit();
```

描述:

如果你要阻止表单提交:

jQuery 代码:

```
$("form").submit( function () {
  return false;
} );
```

unload([[data],fn])

返回值:jQueryunload([[data],fn])

概述

在每一个匹配元素的unload事件中绑定一个处理函数。

参数

fnFunction *V1.0*

在每一个匹配元素的unload事件中绑定的处理函数。

[data],fn String,Function *V1.4.3*

data:unload([Data], fn) 可传入data供函数fn处理。

fn:在每一个匹配元素的unload事件中绑定的处理函数。

示例

描述:

页面卸载的时候弹出一个警告框:

jQuery 代码:

```
$(window).unload( function () { alert("Bye now!"); } );
```

效果

show([speed,[easing],[fn]])

返回值:jQueryshow([speed,[easing],[fn]])

概述

显示隐藏的匹配元素。

这个就是 'show(speed, [callback])' 无动画的版本。如果选择的元素是可见的，这个方法将不会改变任何东西。无论这个元素是通过hide()方法隐藏的还是在CSS里设置了display:none;，这个方法都将有效。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

显示所有段落

HTML 代码:

```
<p style="display: none">Hello</p>
```

jQuery 代码:

```
$("p").show()
```

描述:

用缓慢的动画将隐藏的段落显示出来，历时600毫秒。

HTML 代码:

```
<p style="display: none">Hello</p>
```

jQuery 代码:

```
$("p").show("slow");
```

描述:

用迅速的动画将隐藏的段落显示出来，历时200毫秒。并在之后执行反馈！

HTML 代码:

```
<p style="display: none">Hello</p>
```

jQuery 代码:

```
$("p").show("fast",function(){
   $(this).text("Animation Done!");
 });
```

描述:

将隐藏的段落用将近4秒的时间显示出来。。。并在之后执行一个反馈。。。

HTML 代码:

```
<p style="display: none">Hello</p>
```

jQuery 代码:

```
$("p").show(4000,function(){
   $(this).text("Animation Done...");
 });
```

hide([speed,[easing],[fn]])

返回值:jQueryhide([speed,[easing],[fn]])

概述

隐藏显示的元素

这个就是 'hide(speed, [callback])' 的无动画版。如果选择的元素是隐藏的，这个方法将不会改变任何东西。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

隐藏所有段落

jQuery 代码:

```
$("p").hide()
```

描述:

用600毫秒的时间将段落缓慢的隐藏

jQuery 代码:

```
$("p").hide("slow");
```

描述:

用200毫秒将段落迅速隐藏，之后弹出一个对话框。

jQuery 代码:

```
$("p").hide("fast",function(){
  alert("Animation Done.");
 });
```

slideDown([speed],[easing],[fn])

返回值:jQueryslideDown([speed],[easing],[fn])

概述

通过高度变化（向下增大）来动态地显示所有匹配的元素，在显示完成后可选地触发一个回调函数。

这个动画效果只调整元素的高度，可以使匹配的元素以“滑动”的方式显示出来。在jQuery 1.3中，上下的padding和margin也会有动画，效果更流畅。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn在动画完成时执行的函数，每个元素执行一次。

示例

描述:

以滑动方式显示隐藏的

元素：

jQuery 代码:

```
$(".btn2").click(function(){
  $("p").slideDown();
});
```

描述:

用600毫秒缓慢的将段落滑下

jQuery 代码:

```
$("p").slideDown("slow");
```

描述:

用200毫秒快速将段落滑下，之后弹出一个对话框

jQuery 代码:

```
$("p").slideDown("fast",function(){
   alert("Animation Done.");
 });
```

slideUp([speed,[easing],[fn]])

返回值:jQueryslideUp([speed,[easing],[fn]])

概述

通过高度变化（向上减小）来动态地隐藏所有匹配的元素，在隐藏完成后可选地触发一个回调函数。

这个动画效果只调整元素的高度，可以使匹配的元素以“滑动”的方式隐藏起来。在jQuery 1.3中，上下的padding和margin也会有动画，效果更流畅。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

用600毫秒缓慢的将段落滑上

jQuery 代码:

```
$("p").slideUp("slow");
```

描述:

用200毫秒快速将段落滑上，之后弹出一个对话框

jQuery 代码:

```
$("p").slideUp("fast",function(){
   alert("Animation Done.");
 });
```

slideToggle([speed],[easing],[fn])

返回值:jQueryslideToggle([speed],[easing],[fn])

概述

通过高度变化来切换所有匹配元素的可见性，并在切换完成后可选地触发一个回调函数。

这个动画效果只调整元素的高度，可以使匹配的元素以“滑动”的方式隐藏或显示。在jQuery 1.3中，上下的padding和margin也会有动画，效果更流畅。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

用600毫秒缓慢的将段落滑上或滑下

jQuery 代码:

```
$("p").slideToggle("slow");
```

描述:

用200毫秒快速将段落滑上或滑下，之后弹出一个对话框

jQuery 代码:

```
$("p").slideToggle("fast",function(){
   alert("Animation Done.");
 });
```

fadeIn([speed],[easing],[fn])

返回值:jQueryfadeIn([speed],[easing],[fn])

概述

通过不透明度的变化来实现所有匹配元素的淡入效果，并在动画完成后可选地触发一个回调函数。

这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

用600毫秒缓慢的将段落淡入

jQuery 代码:

```
$("p").fadeIn("slow");
```

描述:

用200毫秒快速将段落淡入，之后弹出一个对话框

jQuery 代码:

```
("p").fadeIn("fast",function(){
   alert("Animation Done.");
 });
```

fadeOut([speed],[easing],[fn])

返回值:jQueryfadeOut([speed],[easing],[fn])

概述

通过不透明度的变化来实现所有匹配元素的淡出效果，并在动画完成后可选地触发一个回调函数。

这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

用600毫秒缓慢的将段落淡出

jQuery 代码:

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

描述:

用200毫秒快速将段落淡出，之后弹出一个对话框

jQuery 代码:

```
$("p").fadeOut("fast",function(){
   alert("Animation Done.");
 });
```

fadeTo([[speed],opacity,[easing],[fn]])

返回值:jQueryfadeTo([[speed],opacity,[easing],[fn]])

概述

把所有匹配元素的不透明度以渐进方式调整到指定的不透明度，并在动画完成后可选地触发一个回调函数。

这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化。

参数

speed,opacity,[fn]Number/String,Number,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

opacity:一个0至1之间表示透明度的数字。

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],opacity,[easing],[fn]Number/String,String,Function *V1.4.3*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

opacity:一个0至1之间表示透明度的数字。

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

使用淡入效果来显示一个隐藏的

元素：

jQuery 代码:

```
$(".btn2").click(function(){
  $("p").fadeIn();
});
```

描述:

用600毫秒缓慢的将段落的透明度调整到0.66，大约2/3的可见度

jQuery 代码:

```
$("p").fadeTo("slow", 0.66);
```

描述:

用200毫秒快速将段落的透明度调整到0.25，大约1/4的可见度，之后弹出一个对话框

jQuery 代码:

```
$("p").fadeTo("fast", 0.25, function(){
   alert("Animation Done.");
 });
```

fadeToggle([speed,[easing],[fn]])

返回值:jQueryfadeToggle([speed,[easing],[fn]])

概述

通过不透明度的变化来开关所有匹配元素的淡入和淡出效果，并在动画完成后可选地触发一个回调函数。

这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化。

参数

speed[,fn]Number/String,Function *V1.0*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

fn:在动画完成时执行的函数，每个元素执行一次。

[speed],[easing],[fn]Number/String,String,Function *V1.4.4*

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:(Optional) 用来指定切换效果，默认是"swing"，可用参数"linear"

fn:在动画完成时执行的函数，每个元素执行一次。

示例

描述:

用600毫秒缓慢的将段落淡入

jQuery 代码:

```
$("p").fadeToggle("slow","linear");
```

描述:

用200毫秒快速将段落淡入，之后弹出一个对话框

jQuery 代码:

```
$("p").fadeToggle("fast",function(){
   alert("Animation Done.");
 });
```

animate(param,[spe],[e],[fn])

返回值:jQueryanimate(params,[speed],[easing],[fn])

概述

用于创建自定义动画的函数。

这个函数的关键在于指定动画形式及结果样式属性对象。这个对象中每个属性都表示一个可以变化的样式属性（如“height”、“top”或“opacity”）。注意：所有指定的属性必须用骆驼形式，比如用marginLeft代替margin-left.

而每个属性的值表示这个样式属性到多少时动画结束。如果是一个数值，样式属性就会从当前的值渐变到指定的值。如果使用的是“hide”、“show”或“toggle”这样的字符串值，则会为该属性调用默认的动画形式。

在 jQuery 1.2 中，你可以使用 em 和 % 单位。另外，在 jQuery 1.2 中，你可以通过在属性值前面指定 "*+=*" 或 "*-=*" 来让元素做相对运动。

jQuery 1.3中，如果duration设为0则直接完成动画。而在以前版本中则会执行默认动画。

jQuery 1.8中，当你使用CSS属性在css()或animate()中，我们将根据浏览器自动加上前缀(在适当的时候)，比如("user-select", "none"); 在Chrome/Safari浏览器中我们将设置为"-webkit-user-select", Firefox会使用"-moz-user-select", IE10将使用"-ms-user-select".

参数

params,[speed],[easing],[fn]Options,Number/String,String,Function *V1.0*

params:一组包含作为动画属性和终值的样式属性和及其值的集合

speed:三种预定速度之一的字符串("slow","normal", or "fast")或表示动画时长的毫秒数值(如：1000)

easing:要使用的擦除效果的名称(需要插件支持).默认jQuery提供"linear" 和 "swing".

fn:在动画完成时执行的函数，每个元素执行一次。

params,optionsString,String *V1.0*

params::一组包含作为动画属性和终值的样式属性和及其值的集合

options:动画的额外选项。如：speed - 设置动画的速度,easing - 规定要使用的 easing 函数,callback - 规定动画完成之后要执行的函数,step - 规定动画的每一步完成之后要执行的函数,queue - 布尔值。指示是否在效果队列中放置动画。如果为 false，则动画将立即开始,specialEasing - 来自 *styles* 参数的一个或多个 CSS 属性的映射，以及它们的对应 easing 函数

示例

描述:

点击按钮后div元素的几个不同属性一同变化

HTML 代码:

```
<button id="go"> Run</button>
<div id="block">Hello!</div>
```

jQuery 代码:

```
// 在一个动画中同时应用三种类型的效果
$("#go").click(function(){
  $("#block").animate({
    width: "90%",
    height: "100%",
    fontSize: "10em",
    borderWidth: 10
  }, 1000 );
});
```

描述:

让指定元素左右移动

HTML 代码:

```
<button id="left">«</button> <button id="right">»</button>
<div class="block"></div>
```

jQuery 代码:

```
$("#right").click(function(){
  $(".block").animate({left: '+50px'}, "slow");
});

$("#left").click(function(){
  $(".block").animate({left: '-50px'}, "slow");
});
```

描述:

在600毫秒内切换段落的高度和透明度

jQuery 代码:

```
$("p").animate({
   height: 'toggle', opacity: 'toggle'
 }, "slow");
```

描述:

用500毫秒将段落移到left为50的地方并且完全清晰显示出来（透明度为1）

jQuery 代码:

```
$("p").animate({
   left: 50, opacity: 'show'
 }, 500);
```

描述:

一个使用“easein”函数提供不同动画样式的例子。只有使用了插件来提供这个“easein”函数，这个参数才起作用。

jQuery 代码:

```
$("p").animate({
   opacity: 'show'
 }, "slow", "easein");
```

stop([cle],[jum])

返回值:jQuerystop([clearQueue],[jumpToEnd])

概述

停止所有在指定元素上正在运行的动画。

如果队列中有等待执行的动画(并且clearQueue没有设为true)，他们将被马上执行

参数

[clearQueue],[gotoEnd] *Boolean,Boolean*V1.2*

clearQueue:如果设置成true，则清空队列。可以立即结束动画。

gotoEnd:让当前正在执行的动画立即完成，并且重设show和hide的原始样式，调用回调函数等。

[queue],[clearQueue],[jumpToEnd]Boolean *V1.7*

queue:用来停止动画的队列名称

clearQueue:如果设置成true，则清空队列。可以立即结束动画。

jumpToEnd:如果设置成true，则完成队列。可以立即完成动画。

示例

描述:

停止当前正在运行的动画：

HTML 代码:

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

描述:

点击Go之后开始动画,点Stop之后会在当前位置停下来

HTML 代码:

```
<button id="go">Go</button> <button id="stop">STOP!</button>
<div class="block"></div><button id="go">Go</button> <button id="stop">STOP!</button>
<div class="block"></div>
```

jQuery 代码:

```
// 开始动画
$("#go").click(function(){
  $(".block").animate({left: '+200px'}, 5000);
});

// 当点击按钮后停止动画
$("#stop").click(function(){
  $(".block").stop();
});
```

delay(duration,[queueName])

返回值:jQuerydelay(duration,*[queueName]*)

概述

设置一个延时来推迟执行队列中之后的项目。

jQuery 1.4新增。用于将队列中的函数延时执行。他既可以推迟动画队列的执行，也可以用于自定义队列。

参数

duration,[queueName]Integer,String *V1.4*

duration:延时时间，单位：毫秒

queueName:队列名词，默认是Fx，动画队列。

示例

描述:

在.slideUp() 和 .fadeIn()之间延时800毫秒。

HTML 代码:

```
<div id="foo /">
```

jQuery 代码:

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

jQuery.fx.off

返回值:BooleanjQuery.fx.off

*V1.3*概述

关闭页面上所有的动画。

把这个属性设置为true可以立即关闭所有动画(所有效果会立即执行完毕)。有些情况下可能需要这样，比如：

- 你在配置比较低的电脑上使用jQuery。
- 你的一些用户由于动画效果而遇到了 可访问性问题

当把这个属性设成false之后，可以重新开启所有动画。

示例

描述:

执行一个禁用的动画

jQuery 代码:

```
jQuery.fx.off = true;
$("input").click(function(){
    $("div").toggle("slow");
});
```

jQuery.fx.interval

返回值:NumberjQuery.fx.interval

*V1.4.3*概述

设置动画的显示帧速。

示例

描述: 帧速设置为100ms

jQuery 代码:

```
<!DOCTYPE html><html><head>  <style>    div { width:50px; height:30px; margin:5px; float:left;
 background:green; }    </style>  <script src="http://code.jquery.com/jquery-1.5.2.js"></script></hea
d><body>  <p><input type="button" value="Run"/></p><div></div><script>jQuery.fx.interval =
100;$("input").click(function(){  $("div").toggle( 3000 );});  </script></body></html>
```

Ajax

jQuery.ajax(url,[settings])

返回值:XMLHttpRequestjQuery.ajax(url,[settings])

概述

通过 HTTP 请求加载远程数据。

jQuery 底层 AJAX 实现。简单易用的高层实现见 $.get, $.post 等。$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该函数，除非你需要操作不常用的选项，以获得更多的灵活性。

最简单的情况下，$.ajax()可以不带任何参数直接使用。

注意，所有的选项都可以通过$.ajaxSetup()函数来全局设置。

回调函数

如果要处理$.ajax()得到的数据，则需要使用回调函数。beforeSend、error、dataFilter、success、complete。

- beforeSend 在发送请求之前调用，并且传入一个XMLHttpRequest作为参数。
- error 在请求出错时调用。传入XMLHttpRequest对象，描述错误类型的字符串以及一个异常对象（如果有的话）
- dataFilter 在请求成功之后调用。传入返回的数据以及"dataType"参数的值。并且必须返回新的数据（可能是处理过的）传递给success回调函数。
- success 当请求之后调用。传入返回后的数据，以及包含成功代码的字符串。
- complete 当请求完成之后调用这个函数，无论成功或失败。传入XMLHttpRequest对象，以及一个包含成功或错误代码的字符串。

数据类型

$.ajax()函数依赖服务器提供的信息来处理返回的数据。如果服务器报告说返回的数据是XML，那么返回的结果就可以用普通的XML方法或者jQuery的选择器来遍历。如果见得到其他类型，比如HTML，则数据就以文本形式来对待。

通过dataType选项还可以指定其他不同数据处理方式。除了单纯的XML，还可以指定 html、json、jsonp、script或者text。

其中，text和xml类型返回的数据不会经过处理。数据仅仅简单的将XMLHttpRequest的responseText或responseHTML属性传递给success回调函数，

'''注意'''，我们必须确保网页服务器报告的MIME类型与我们选择的dataType所匹配。比如说，XML的话，服务器端就必须声明 text/xml 或者 application/xml 来获得一致的结果。

如果指定为html类型，任何内嵌的JavaScript都会在HTML作为一个字符串返回之前执行。类似的，指定script类型的话，也会先执行服务器端生成JavaScript，然后再把脚本作为一个文本数据返回。

如果指定为json类型，则会把获取到的数据作为一个JavaScript对象来解析，并且把构建好的对象作为结果返回。为了实现这个目的，他首先尝试使用JSON.parse()。如果浏览器不支持，则使用一个函数来构建。JSON数据是一种能很方便通过JavaScript解析的结构化数据。如果获取的数据文件存放在远程服务器上（域名不同，也就是跨域获取数据），则需要使用jsonp类型。使用这种类型的话，会创建一个查询字符串参数 callback=? ，这个参数会加在请求的URL后面。服务器端应当在JSON数据前加上回调函数名，以便完成一个有效的JSONP请求。如果要指定回调函数的参数名来取代默认的callback，可以通过设置$.ajax()的jsonp参数。

注意，JSONP是JSON格式的扩展。他要求一些服务器端的代码来检测并处理查询字符串参数。更多信息可以参阅 最初的文章。

如果指定了script或者jsonp类型，那么当从服务器接收到数据时，实际上是用了<script>标签而不是XMLHttpRequest对象。这种情况下，$.ajax()不再返回一个XMLHttpRequest对象，并且也不会传递事件处理函数，比如beforeSend。

发送数据到服务器

默认情况下，Ajax请求使用GET方法。如果要使用POST方法，可以设定type参数值。这个选项也会影响data选项中的内容如何发送到服务器。

data选项既可以包含一个查询字符串，比如 key1=value1&key2=value2 ，也可以是一个映射，比如 {key1: 'value1', key2: 'value2'} 。如果使用了后者的形式，则数据再发送器会被转换成查询字符串。这个处理过程也可以通过设置processData选项为false来回避。如果我们希望发送一个XML对象给服务器时，这种处理可能并不合适。并且在这种情况下，我们也应当改变contentType选项的值，用其他合适的MIME类型来取代默认的 application/x-www-form-urlencoded 。

高级选项

global选项用于阻止响应注册的回调函数，比如.ajaxSend，或者ajaxError，以及类似的方法。这在有些时候很有用，比如发送的请求非常频繁且简短的时候，就可以在ajaxSend里禁用这个。更多关于这些方法的详细信息，请参阅下面的内容。

如果服务器需要HTTP认证，可以使用用户名和密码可以通过username和password选项来设置。

Ajax请求是限时的，所以错误警告被捕获并处理后，可以用来提升用户体验。请求超时这个参数通常就保留其默认值，要不就通过jQuery.ajaxSetup来全局设定，很少为特定的请求重新设置timeout选项。

默认情况下，请求总会被发出去，但浏览器有可能从他的缓存中调取数据。要禁止使用缓存的结果，可以设置cache参数为false。如果希望判断数据自从上次请求后没有更改过就报告出错的话，可以设置ifModified为true。

scriptCharset允许给<script>标签的请求设定一个特定的字符集，用于script或者jsonp类似的数据。当脚本和页面字符集不同时，这特别好用。

Ajax的第一个字母是asynchronous的开头字母，这意味着所有的操作都是并行的，完成的顺序没有前后关系。$.ajax()的async参数总是设置成true，这标志着在请求开始后，其他代码依然能够执行。强烈不建议把这个选项设置成false，这意味着所有的请求都不再是异步的了，这也会导致浏览器被锁死。

$.ajax函数返回他创建的XMLHttpRequest对象。通常jQuery只在内部处理并创建这个对象，但用户也可以通过xhr选项来传递一个自己创建的xhr对象。返回的对象通常已经被丢弃了，但依然提供一个底层接口来观察和操控请求。比如说，调用对象上的.abort()可以在请求完成前挂起请求。

参数

url,[settings]Object *V1.5*

url:一个用来包含发送请求的URL字符串。

settings:AJAX 请求设置。所有选项都是可选的。

*V1.0*settings:选项

acceptsMap

默认： 取决于数据类型。

内容类型发送请求头，告诉服务器什么样的响应会接受返回。如果accepts设置需要修改，推荐在$.ajaxSetup()方法中做一次。

asyncBoolean

(默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。注意，同步请求将锁住浏览器，用户其它操作必须等待请求完成才可以执行。

beforeSend(XHR)Function

发送请求前可修改 XMLHttpRequest 对象的函数，如添加自定义 HTTP 头。XMLHttpRequest 对象是唯一的参数。这是一个 Ajax 事件。如果返回false可以取消本次ajax请求。

```
function (XMLHttpRequest) {
    this; // 调用本次AJAX请求时传递的options参数
}
```

cacheBoolean

(默认: true,dataType为script和jsonp时默认为false) jQuery 1.2 新功能，设置为 false 将不缓存此页面。

complete(XHR, TS)Function

请求完成后回调函数 (请求成功或失败之后均调用)。参数： XMLHttpRequest 对象和一个描述成功请求类型的字符串。 Ajax 事件。

```
function (XMLHttpRequest, textStatus) {
    this; // 调用本次AJAX请求时传递的options参数
}
```

contentsMap *V1.5*

一个以"{字符串:正则表达式}"配对的对象，用来确定jQuery将如何解析响应，给定其内容类型。

contentTypeString

(默认: "application/x-www-form-urlencoded") 发送信息至服务器时内容编码类型。默认值适合大多数情况。如果你明确地传递了一个content-type给 $.ajax() 那么他必定会发送给服务器（即使没有数据要发送）

contextObject

这个对象用于设置Ajax相关回调函数的上下文。也就是说，让回调函数内this指向这个对象（如果不设定这个参数，那么this就指向调用本次AJAX请求时传递的options参数）。比如指定一个DOM元素作为context参数，这样就设置了success回调函数的上下文为这个DOM元素。就像这样：

```
$.ajax({ url: "test.html", context: document.body, success: function(){
    $(this).addClass("done");
}});
```

convertersmap *V1.5*

默认： {"* text": window.String, "text html": true, "text json": jQuery.parseJSON, "text xml": jQuery.parseXML}

一个数据类型对数据类型转换器的对象。每个转换器的值是一个函数，返回响应的转化值

crossDomainmap *V1.5*

默认： 同域请求为false

跨域请求为true如果你想强制跨域请求（如JSONP形式）同一域，设置crossDomain为true。这使得例如，服务器端重定向到另一个域

dataObject,String

发送到服务器的数据。将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。查看 processData 选项说明以禁止此自动转换。必须为 Key/Value 格式。如果为数组，jQuery 将自动为不同值对应同一个名称。如 {foo:["bar1", "bar2"]} 转换为 "&foo=bar1&foo=bar2"。

dataFilterFunction

给Ajax返回的原始数据的进行预处理的函数。提供data和type两个参数：data是Ajax返回的原始数据，type是调用jQuery.ajax时提供的dataType参数。函数返回的值将由jQuery进一步处理。

```
function (data, type) {
    // 对Ajax返回的原始数据进行预处理
    return data  // 返回处理后的数据
}
```

dataTypeString

预期服务器返回的数据类型。如果不指定，jQuery 将自动根据 HTTP 包 MIME 信息来智能判断，比如 XML MIME类型就被识别为XML。在1.4中，JSON就会生成一个JavaScript对象，而script则会执行这个脚本。随后服务器端返回的数据会根据这个值解析后，传递给回调函数。可用值:

"xml": 返回 XML 文档，可用 jQuery 处理。

"html": 返回纯文本 HTML 信息；包含的script标签会在插入dom时执行。

"script": 返回纯文本 JavaScript 代码。不会自动缓存结果。除非设置了"cache"参数。'''注意：'''在远程请求时(不在同一个域下)，所有POST请求都将转为GET请求。(因为将使用DOM的script标签来加载)

"json": 返回 JSON 数据 。

"jsonp": JSONP 格式。使用 JSONP 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。

"text": 返回纯文本字符串

errorFunction

(默认: 自动判断 (xml 或 html)) 请求失败时调用此函数。有以下三个参数：XMLHttpRequest 对象、错误信息、（可选）捕获的异常对象。如果发生了错误，错误信息（第二个参数）除了得到null之外，还可能是"timeout", "error", "notmodified" 和 "parsererror"。Ajax 事件。

```
function (XMLHttpRequest, textStatus, errorThrown) {
    // 通常 textStatus 和 errorThrown 之中
    // 只有一个会包含信息
    this; // 调用本次AJAX请求时传递的options参数
}
```

globalBoolean

(默认: true) 是否触发全局 AJAX 事件。设置为 false 将不会触发全局 AJAX 事件，如 ajaxStart 或 ajaxStop 可用于控制不同的 Ajax 事件。

headersmap *V1.5*

Default: {}

一个额外的"{键:值}"对映射到请求一起发送。此设置被设置之前beforeSend函数被调用;因此，消息头中的值设置可以在覆盖beforeSend函数范围内的任何设置。

ifModifiedBoolean

(默认: false) 仅在服务器数据改变时获取新数据。使用 HTTP 包 Last-Modified 头信息判断。在jQuery 1.4中，他也会检查服务器指定的'etag'来确定数据没有被修改过。

isLocalmap *V1.5.1*

默认: 取决于当前的位置协议

允许当前环境被认定为“本地”，（如文件系统），即使jQuery默认情况下不会承认它。以下协议目前公认为本地：file, *-extension, and widget。如果isLocal设置需要修改，建议在$.ajaxSetup()方法中这样做一次。

jsonpString

在一个jsonp请求中重写回调函数的名字。这个值用来替代在"callback=?"这种GET或POST请求中URL参数里的"callback"部分，比如{jsonp:'onJsonPLoad'}会导致将"onJsonPLoad=?"传给服务器。

jsonpCallbackString

为jsonp请求指定一个回调函数名。这个值将用来取代jQuery自动生成的随机函数名。这主要用来让jQuery生成度独特的函数名，这样管理请求更容易，也能方便地提供回调函数和错误处理。你也可以在想让浏览器缓存GET请求的时候，指定这个回调函数名。

mimeTypeString *V1.5.1*

一个mime类型用来覆盖XHR的 MIME类型。

passwordString

用于响应HTTP访问认证请求的密码

processDataBoolean

(默认: true) 默认情况下，通过data选项传递进来的数据，如果是一个对象(技术上讲只要不是字符串)，都会处理转化成一个查询字符串，以配合默认内容类型 "application/x-www-form-urlencoded"。如果要发送 DOM 树信息或其它不希望转换的信息，请设置为 false。

scriptCharsetString

只有当请求时dataType为"jsonp"或"script"，并且type是"GET"才会用于强制修改charset。通常只在本地和远程的内容编码不同时使用。

statusCodemap *V1.5*

默认: {}

一组数值的HTTP代码和函数对象，当响应时调用了相应的代码。例如，如果响应状态是404，将触发以下警报：

```
$.ajax({
  statusCode: {404: function() {
    alert('page not found');
  }
});
```

success(data, textStatus, jqXHR)Function,Array

请求成功后的回调函数。参数：由服务器返回，并根据dataType参数进行处理后的数据；描述状态的字符串。还有 jqXHR（在jQuery 1.4.x的中，XMLHttpRequest） 对象 。在jQuery 1.5， 成功设置可以接受一个函数数组。每个函数将被依次调用。 Ajax 事件。

```
function (data, textStatus) {
    // data 可能是 xmlDoc, jsonObj, html, text, 等等...
    this; // 调用本次AJAX请求时传递的options参数
}
```

traditionalBoolean

如果你想要用传统的方式来序列化数据，那么就设置为true。请参考工具分类下面的jQuery.param 方法。

timeoutNumber

设置请求超时时间（毫秒）。此设置将覆盖全局设置。

typeString

(默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET"。注意：其它 HTTP 请求方法，如 PUT 和 DELETE 也可以使用，但仅部分浏览器支持。

urlString

(默认: 当前页地址) 发送请求的地址。

usernameString

用于响应HTTP访问认证请求的用户名

xhrFunction

需要返回一个XMLHttpRequest 对象。默认在IE下是ActiveXObject 而其他情况下是XMLHttpRequest 。用于重写或者提供一个增强的XMLHttpRequest 对象。这个参数在jQuery 1.3以前不可用。

xhrFieldsmap *V1.5*

一对“文件名-文件值”在本机设置XHR对象。例如，如果需要的话，你可以用它来设置withCredentials为true的跨域请求。

示例

描述:

加载并执行一个 JS 文件。

jQuery 代码:

```
$.ajax({
  type: "GET",
  url: "test.js",
  dataType: "script"
});
```

描述:

保存数据到服务器，成功时显示信息。

jQuery 代码:

```
$.ajax({
   type: "POST",
   url: "some.php",
   data: "name=John&location=Boston",
   success: function(msg){
     alert( "Data Saved: " + msg );
   }
});
```

描述:

装入一个 HTML 网页最新版本。

jQuery 代码:

```
$.ajax({
  url: "test.html",
  cache: false,
  success: function(html){
    $("#results").append(html);
  }
});
```

描述:

同步加载数据。发送请求时锁住浏览器。需要锁定用户交互操作时使用同步方式。

jQuery 代码:

```
var html = $.ajax({
 url: "some.php",
 async: false
}).responseText;
```

描述:

发送 XML 数据至服务器。设置 processData 选项为 false，防止自动转换数据格式。

jQuery 代码:

```
var xmlDocument = [create xml document];
$.ajax({
  url: "page.php",
  processData: false,
  data: xmlDocument,
  success: handleResponse
});
```

load(url,[data],[callback])

返回值:jQueryload(url, *[data]*, *[callback]*)

概述

载入远程 HTML 文件代码并插入至 DOM 中。

默认使用 GET 方式 - 传递附加参数时自动转换为 POST 方式。jQuery 1.2 中，可以指定选择符，来筛选载入的 HTML 文档，DOM 中将仅插入筛选出的 HTML 代码。语法形如 "url #some > selector"。请查看示例。

参数

url,[data,[callback]]String,Map/String,Callback *V1.0*

url:待装入 HTML 网页网址。

data:发送至服务器的 key/value 数据。在jQuery 1.3中也可以接受一个字符串了。

callback:载入成功时回调函数。

示例

描述:

加载文章侧边栏导航部分至一个无序列表。

HTML 代码:

```
<b>jQuery Links:</b>
<ul id="links"></ul>
```

jQuery 代码:

```
$("#links").load("/Main_Page #p-Getting-Started li");
```

描述:

加载 feeds.html 文件内容。

jQuery 代码:

```
$("#feeds").load("feeds.html");
```

描述:

同上，但是以 POST 形式发送附加参数并在成功时显示信息。

jQuery 代码:

```
$("#feeds").load("feeds.php", {limit: 25}, function(){
  alert("The last 25 entries in the feed have been loaded");
});
```

jQuery.get(url,[data],[callback],[type])

返回值:XMLHttpRequestjQuery.get(url, *[data]*, *[callback]*, *[type]*)

概述

通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 $.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 $.ajax。

参数

url,[data],[callback],[type]String,Map,Function,String *V1.0*

url:待载入页面的URL地址

data:待发送 Key/value 参数。

callback:载入成功时回调函数。

type:返回内容格式，xml, html, script, json, text, _default。

示例

描述:

请求 test.php 网页，忽略返回值。

jQuery 代码:

```
$.get("test.php");
```

描述:

请求 test.php 网页，传送2个参数，忽略返回值。

jQuery 代码:

```
$.get("test.php", { name: "John", time: "2pm" } );
```

描述:

显示 test.php 返回值(HTML 或 XML，取决于返回值)。

jQuery 代码:

```
$.get("test.php", function(data){
  alert("Data Loaded: " + data);
});
```

描述:

显示 test.cgi 返回值(HTML 或 XML，取决于返回值)，添加一组请求参数。

jQuery 代码:

```
$.get("test.cgi", { name: "John", time: "2pm" },
  function(data){
    alert("Data Loaded: " + data);
  });
```

jQuery.getJSON(url,[data],[callback])

返回值:XMLHttpRequestjQuery.getJSON(url, *[data]*, *[callback]*)

概述

通过 HTTP GET 请求载入 JSON 数据。

在 jQuery 1.2 中，您可以通过使用JSONP形式的回调函数来加载其他网域的JSON数据，如 "myurl?callback=?"。jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。 注意：此行以后的代码将在这个回调函数执行前执行。

参数

url,[data],[callback]String,Map,Function*V1.0*

url:发送请求地址。

data:待发送 Key/value 参数。

callback:载入成功时回调函数。

示例

描述:

从 Flickr JSONP API 载入 4 张最新的关于猫的图片。

HTML 代码:

```
<div id="images"></div>
```

jQuery 代码:

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?tags=cat&tagmode=any&format=
json&jsoncallback=?", function(data){
  $.each(data.items, function(i,item){
    $("<img/>").attr("src", item.media.m).appendTo("#images");
    if ( i == 3 ) return false;
  });
});
```

描述:

从 test.js 载入 JSON 数据并显示 JSON 数据中一个 name 字段数据。

jQuery 代码:

```
$.getJSON("test.js", function(json){
  alert("JSON Data: " + json.users[3].name);
});
```

描述:

从 test.js 载入 JSON 数据，附加参数，显示 JSON 数据中一个 name 字段数据。

jQuery 代码:

```
$.getJSON("test.js", { name: "John", time: "2pm" }, function(json){
  alert("JSON Data: " + json.users[3].name);
});
```

jQuery.getScript(url,[callback])

返回值:XMLHttpRequestjQuery.getScript(url, *[callback]*)

概述

通过 HTTP GET 请求载入并执行一个 JavaScript 文件。

jQuery 1.2 版本之前，getScript 只能调用同域 JS 文件。 1.2中，您可以跨域调用 JavaScript 文件。注意：Safari 2 或更早的版本不能在全局作用域中同步执行脚本。如果通过 getScript 加入脚本，请加入延时函数。

参数

url,[callback]String,Function *V1.0*

url:待载入 JS 文件地址。

callback:成功载入后回调函数。

示例

描述:

载入 jQuery 官方颜色动画插件 成功后绑定颜色变化动画。

HTML 代码:

```
<button id="go">» Run</button>
<div class="block"></div>
```

jQuery 代码:

```
jQuery.getScript("http://dev.jquery.com/view/trunk/plugins/color/jquery.color.js", function(){
  $("#go").click(function(){
    $(".block").animate( { backgroundColor: 'pink' }, 1000)
      .animate( { backgroundColor: 'blue' }, 1000);
  });
});
```

描述:

加载并执行 test.js。

jQuery 代码:

```
$.getScript("test.js");
```

描述:

加载并执行 test.js ，成功后显示信息。

jQuery 代码:

```
$.getScript("test.js", function(){
  alert("Script loaded and executed.");
});
```

jQuery.post(url,[data],[callback],[type])

返回值:XMLHttpRequestjQuery.post(url, *[data]*, *[callback]*, *[type]*)

概述

通过远程 HTTP POST 请求载入信息。

这是一个简单的 POST 请求功能以取代复杂 $.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 $.ajax。

参数

url,[data],[callback],[type]String,Map,Function,String *V1.0*

url:发送请求地址。

data:待发送 Key/value 参数。

callback:发送成功时回调函数。

type:返回内容格式，xml, html, script, json, text, _default。

示例

1描述:

请求 test.php 网页，忽略返回值：

jQuery 代码:

```
$.post("test.php");
```

2描述:

请求 test.php 页面，并一起发送一些额外的数据（同时仍然忽略返回值）：

jQuery 代码:

```
$.post("test.php", { name: "John", time: "2pm" } );
```

3描述:

向服务器传递数据数组（同时仍然忽略返回值）：

jQuery 代码:

```
$.post("test.php", { 'choices[]': ["Jon", "Susan"] });
```

4描述:

使用 ajax 请求发送表单数据：

jQuery 代码:

```
$.post("test.php", $("#testform").serialize());
```

5描述:

输出来自请求页面 test.php 的结果（HTML 或 XML，取决于所返回的内容）：

jQuery 代码:

```
$.post("test.php", function(data){
  alert("Data Loaded: " + data);
 });
```

6描述:

向页面 test.php 发送数据，并输出结果（HTML 或 XML，取决于所返回的内容）：

jQuery 代码:

```
$.post("test.php", { name: "John", time: "2pm" },
  function(data){
    alert("Data Loaded: " + data);
  });
```

7描述:

获得 test.php 页面的内容，并存储为 XMLHttpResponse 对象，并通过 process() 这个 JavaScript 函数进行处理：

jQuery 代码:

```
$.post("test.php", { name: "John", time: "2pm" },
  function(data){
    process(data);
  }, "xml");
```

8描述:

获得 test.php 页面返回的 json 格式的内容：：

jQuery 代码:

```
$.post("test.php", { "func": "getNameAndTime" },
  function(data){
    alert(data.name); // John
    console.log(data.time); //  2pm
  }, "json");
```

ajaxComplete(callback)

返回值:jQueryajaxComplete(callback)

概述

AJAX 请求完成时执行函数。Ajax 事件。

XMLHttpRequest 对象和设置作为参数传递给回调函数。

参数

callbackFunction *V1.0*

待执行函数

示例

描述:

AJAX 请求完成时执行函数。

jQuery 代码:

```
$("#msg").ajaxComplete(function(event,request, settings){
  $(this).append("<li>请求完成.</li>");
});
```

描述:

当 AJAX 请求正在进行时显示“正在加载”的指示：

jQuery 代码:

```
$("#txt").ajaxStart(function(){
  $("#wait").css("display","block");
});
$("#txt").ajaxComplete(function(){
  $("#wait").css("display","none");
});
```

ajaxError(callback)

返回值:jQueryajaxError(callback)

概述

AJAX 请求发生错误时执行函数。Ajax 事件。

XMLHttpRequest 对象和设置作为参数传递给回调函数。捕捉到的错误可作为最后一个参数传递。

参数

callbackFunction *V1.0*

待执行函数

```
function (event, XMLHttpRequest, ajaxOptions, thrownError) {
      // thrownError 只有当异常发生时才会被传递
      this; // 监听的 dom 元素
}
```

示例

描述:

AJAX 请求失败时显示信息。

jQuery 代码:

```
$("#msg").ajaxError(function(event,request, settings){
    $(this).append("<li>出错页面:" + settings.url + "</li>");
});
```

ajaxSend(callback)

返回值:jQueryajaxSend(callback)

概述

AJAX 请求发送前执行函数。Ajax 事件。

XMLHttpRequest 对象和设置作为参数传递给回调函数。

参数

callbackFunction *V1.0*

待执行函数

示例

描述:

AJAX 请求发送前显示信息。

jQuery 代码:

```
$("#msg").ajaxSend(function(evt, request, settings){
  $(this).append("<li>开始请求: " + settings.url + "</li>");
});
```

ajaxStart(callback)

返回值:jQueryajaxStart(callback)

概述

AJAX 请求开始时执行函数。Ajax 事件。

参数

callbackFunction *V1.0*

待执行函数

示例

描述:

AJAX 请求开始时显示信息。

jQuery 代码:

```
$("#loading").ajaxStart(function(){
  $(this).show();
});
```

ajaxStop(callback)

返回值:jQueryajaxStop(callback)

概述

AJAX 请求结束时执行函数。Ajax 事件。

参数

callbackFunction *V1.0*

待执行函数

示例

描述:

AJAX 请求结束后隐藏信息。

jQuery 代码:

```
$("#loading").ajaxStop(function(){
  $(this).hide();
});
```

ajaxSuccess(callback)

返回值:jQueryajaxSuccess(callback)

概述

AJAX 请求成功时执行函数。Ajax 事件。

XMLHttpRequest 对象和设置作为参数传递给回调函数。

参数

callbackFunction *V1.0*

待执行函数

示例

描述:

当 AJAX 请求成功后显示消息。

jQuery 代码:

```
$("#msg").ajaxSuccess(function(evt, request, settings){
  $(this).append("<li>请求成功!</li>");
});
```

jQuery.ajaxSetup([options])

返回值:jQueryjQuery.ajaxSetup(*[options]*)

概述

设置全局 AJAX 默认选项。

参数见 '$.ajax' 说明。

参数

options **Object*V1.1*

选项设置。所有设置项均为可选设置。.

示例

描述:

设置 AJAX 请求默认地址为 "/xmlhttp/"，禁止触发全局 AJAX 事件，用 POST 代替默认 GET 方法。其后的 AJAX 请求不再设置任何选项参数。

jQuery 代码:

```
$.ajaxSetup({
  url: "/xmlhttp/",
  global: false,
  type: "POST"
});
$.ajax({ data: myData });
```

serialize()

返回值:Stringserialize()

*V1.0*概述

序列表表格内容为字符串。

示例

描述:

序列表表格内容为字符串，用于 Ajax 请求。

HTML 代码:

```
<p id="results"><b>Results: </b> </p>
<form>
  <select name="single">
    <option>Single</option>
    <option>Single2</option>
  </select>
  <select name="multiple" multiple="multiple">
    <option selected="selected">Multiple</option>
    <option>Multiple2</option>
    <option selected="selected">Multiple3</option>
  </select><br/>
  <input type="checkbox" name="check" value="check1"/> check1
  <input type="checkbox" name="check" value="check2" checked="checked"/> check2
  <input type="radio" name="radio" value="radio1" checked="checked"/> radio1
  <input type="radio" name="radio" value="radio2"/> radio2
</form>
```

jQuery 代码:

```
$("#results").append( "<tt>" + $("form").serialize() + "</tt>" );
```

serializeArray()

返回值:ArrayserializeArray()

*V1.2*概述

序列化表格元素 (类似 '.serialize()' 方法) 返回 JSON 数据结构数据。

'''注意''' ，此方法返回的是JSON对象而非JSON字符串。需要使用插件或者第三方库进行字符串化操作。

返回的JSON对象是由一个对象数组组成的，其中每个对象包含一个或两个名值对——name参数和value参数（如果value不为空的话）。举例来说：

```
[
    {name: 'firstname', value: 'Hello'},
    {name: 'lastname', value: 'World'},
    {name: 'alias'}, // this one was empty
 ]
```

示例

描述:

取得表单内容并插入到网页中。

HTML 代码:

```
<p id="results"><b>Results:</b> </p>
<form>
  <select name="single">
    <option>Single</option>
    <option>Single2</option>
  </select>
  <select name="multiple" multiple="multiple">
    <option selected="selected">Multiple</option>
    <option>Multiple2</option>
    <option selected="selected">Multiple3</option>
  </select><br/>
  <input type="checkbox" name="check" value="check1"/> check1
  <input type="checkbox" name="check" value="check2" checked="c
hecked"/> check2
  <input type="radio" name="radio" value="radio1" checked="check
ed"/> radio1
  <input type="radio" name="radio" value="radio2"/> radio2
</form>
```

jQuery 代码:

```
var fields = $("select, :radio").serializeArray();
jQuery.each( fields, function(i, field){
  $("#results").append(field.value + " ");
});
```

工具

jQuery.support

返回值:ObjectjQuery.support

*V1.3*概述

jQuery 1.3 新增。一组用于展示不同浏览器各自特性和bug的属性集合。

jQuery提供了一系列属性，你也可以自由增加你自己的属性。其中许多属性是很低级的，所以很难说他们能否在日新月异的发展中一直保持有效，但这这些主要用于插件和内核开发者。

所有这些支持的属性值都通过特性检测来实现，而不是用任何浏览器检测。以下有一些非常棒的资源用于解释这些特性检测是如何工作的：

- http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting
- http://yura.thinkweb2.com/cft/
- http://www.jibbering.com/faq/faq_notes/not_browser_detect.html

jQuery.support主要包括以下测试：

boxModel: 如果这个页面和浏览器是以W3C CSS盒式模型来渲染的，则等于true。通常在IE 6和IE 7的怪癖模式中这个值是false。在document准备就绪前，这个值是null。

cssFloat: 如果用cssFloat来访问CSS的float的值，则返回true。目前在IE中会返回false,他用styleFloat代替。

hrefNormalized: 如果浏览器从getAttribute("href")返回的是原封不动的结果，则返回true。在IE中会返回false，因为他的URLs已经常规化了。

htmlSerialize: 如果浏览器通过innerHTML插入链接元素的时候会序列化这些链接，则返回true，目前IE中返回false。

leadingWhitespace: 如果在使用innerHTML的时候浏览器会保持前导空白字符，则返回true，目前在IE 6-8中返回false。

noCloneEvent: 如果浏览器在克隆元素的时候不会连同事件处理函数一起复制，则返回true，目前在IE中返回false。

objectAll: 如果在某个元素对象上执行getElementsByTagName("*")会返回所有子孙元素，则为true，目前在IE 7中为false。

opacity: 如果浏览器能适当解释透明度样式属性，则返回true，目前在IE中返回false，因为他用alpha滤镜代替。

scriptEval: 使用 appendChild/createTextNode 方法插入脚本代码时，浏览器是否执行脚本，目前在IE中返回false，IE使用 .text 方法插入脚本代码以执行。

style: 如果getAttribute("style")返回元素的行内样式，则为true。目前IE中为false，因为他用cssText代替。

tbody: 如果浏览器允许table元素不包含tbody元素，则返回true。目前在IE中会返回false，他会自动插入缺失的tbody。

示例

描述:

检测浏览器是否支持盒式模型

jQuery 代码:

```
jQuery.support.boxModel
```

jQuery.browser

返回值:MapjQuery.browser

*V1.0*概述

在jQuery 1.3中不建议使用。浏览器内核标识。依据 navigator.userAgent 判断。

可用值: safari

opera

msie

mozilla 此属性在 DOM 树加载完成前即有效，可用于为特定浏览器设置 ready 事件。

浏览器对象检测技术与此属性共同使用可提供可靠的浏览器检测支持。

示例

描述:

在 Microsoft's Internet Explorer 浏览器中返回 true。

jQuery 代码:

```
$.browser.msie
```

描述:

仅在 Safari 中提示 "this is safari!" 。

jQuery 代码:

```
if ($.browser.safari) {
   alert("this is safari!");
}
```

jQuery.browser.version

返回值:StringjQuery.browser.version

*V1.1.3*概述

在jQuery 1.3中不建议使用。浏览器渲染引擎版本号。

典型结果: Internet Explorer: 6.0, 7.0

Mozilla/Firefox/Flock/Camino: 1.7.12, 1.8.1.3

Opera: 9.20

Safari/Webkit: 312.8, 418.9

示例

描述:

显示当前 IE 浏览器版本号。

jQuery 代码:

```
if ( $.browser.msie )
 alert( $.browser.version );
```

jQuery.boxModel

返回值:BooleanjQuery.boxModel

*V1.0*概述

在jQuery 1.3中不建议使用。当前页面中浏览器是否使用标准盒模型渲染页面。 建议使用 jQuery.support.boxModel 代替。W3C CSS 盒模型.

示例

描述:

在 Internet Explorer 怪癖模式（QuirksMode）中返回 False。

jQuery 代码:

```
$.boxModel
```

jQuery.each(object,[callback])

返回值:ObjectjQuery.each(object, *[callback]*)

概述

通用例遍方法，可用于例遍对象和数组。

不同于例遍 jQuery 对象的 $().each() 方法，此方法可用于例遍任何对象。回调函数拥有两个参数：第一个为对象的成员或数组的索引，第二个为对应变量或内容。如果需要退出 each 循环可使回调函数返回 false，其它返回值将被忽略。

参数

object,[callback]Object,Function *V1.0*

object:需要例遍的对象或数组。

callback:每个成员/元素执行的回调函数。

示例

描述:

例遍数组，同时使用元素索引和内容。

jQuery 代码:

```
$.each( [0,1,2], function(i, n){
  alert( "Item #" + i + ": " + n );
});
```

描述:

例遍对象，同时使用成员名称和变量内容。

jQuery 代码:

```
$.each( { name: "John", lang: "JS" }, function(i, n){
  alert( "Name: " + i + ", Value: " + n );
});
```

jQuery.extend([deep],target,object1,[objectN])

返回值:ObjectjQuery.extend(*[deep]*, target, object1, *[objectN]*)

概述

用一个或多个其他对象来扩展一个对象，返回被扩展的对象。

如果不指定target，则给jQuery命名空间本身进行扩展。这有助于插件作者为jQuery增加新方法。 如果第一个参数设置为true，则jQuery返回一个深层次的副本，递归地复制找到的任何对象。否则的话，副本会与原对象共享结构。 未定义的属性将不会被复制，然而从对象的原型继承的属性将会被复制。

参数

target,[object1],[objectN]Object,Object,Object *V1.0*

target:一个对象，如果附加的对象被传递给这个方法将那么它将接收新的属性，如果它是唯一的参数将扩展jQuery的命名空间。

object1:待合并到第一个对象的对象。

objectN:待合并到第一个对象的对象。

[deep],target,object1,[objectN]Object,Object,Object,Object *V1.1.4*

deep:如果设为true，则递归合并。

target:待修改对象。

object1:待合并到第一个对象的对象。

objectN:待合并到第一个对象的对象。

示例

描述:

合并 settings 和 options，修改并返回 settings。

jQuery 代码:

```
var settings = { validate: false, limit: 5, name: "foo" };
var options = { validate: true, name: "bar" };
jQuery.extend(settings, options);
```

结果:

```
settings == { validate: true, limit: 5, name: "bar" }
```

描述:

合并 defaults 和 options, 不修改 defaults。

jQuery 代码:

```
var empty = {};
var defaults = { validate: false, limit: 5, name: "foo" };
var options = { validate: true, name: "bar" };
var settings = jQuery.extend(empty, defaults, options);
```

结果:

```
settings == { validate: true, limit: 5, name: "bar" }
empty == { validate: true, limit: 5, name: "bar" }
```

jQuery.grep(array,callback,[invert])

返回值:ArrayjQuery.grep(array, callback, *[invert]*)

概述

使用过滤函数过滤数组元素。

此函数至少传递两个参数：待过滤数组和过滤函数。过滤函数必须返回 true 以保留元素或 false 以删除元素。

参数

array,callback,[invert]Array,Function,Boolean *V1.0*

array:待过滤数组。

callback:此函数将处理数组每个元素。第一个参数为当前元素，第二个参数而元素索引值。此函数应返回一个布尔值。另外，此函数可设置为一个字符串，当设置为字符串时，将视为“lambda-form”（缩写形式？），其中 a 代表数组元素，i 代表元素索引值。如“a > 0”代表“function(a){ return a > 0; }”。

invert:如果 "invert" 为 false 或为设置，则函数返回数组中由过滤函数返回 true 的元素，当"invert" 为 true，则返回过滤函数中返回 false 的元素集。

示例

描述:

过滤数组中小于 0 的元素。

jQuery 代码:

```
$.grep( [0,1,2], function(n,i){
  return n > 0;
});
```

结果:

```
[1, 2]
```

描述:

排除数组中大于 0 的元素，使用第三个参数进行排除。

jQuery 代码:

```
$.grep( [0,1,2], function(n,i){
  return n > 0;
}, true);
```

结果:

```
[0]
```

jQuery.makeArray(obj)

返回值:ArrayjQuery.makeArray(obj)

概述

将类数组对象转换为数组对象。

类数组对象有 length 属性，其成员索引为 0 至 length - 1。实际中此函数在 jQuery 中将自动使用而无需特意转换。

参数

objObject *V1.2*

类数组对象。

示例

描述:

过滤数组中小于 0 的元素。

HTML 代码:

```
<div>First</div><div>Second</div><div>Third</div><div>Fourth</div>
```

jQuery 代码:

```
var arr = jQuery.makeArray(document.getElementsByTagName("div"));
arr.reverse(); // 使用数组翻转函数
```

结果:

```
Fourth
Third
Second
First
```

jQuery.map(array, callback)

返回值:ArrayjQuery.map(arr|obj,callback)

概述

将一个数组中的元素转换到另一个数组中。

作为参数的转换函数会为每个数组元素调用，而且会给这个转换函数传递一个表示被转换的元素作为参数。转换函数可以返回转换后的值、null（删除数组中的项目）或一个包含值的数组，并扩展至原始数组中。

参数

array,callback Array,Function *V1.0*

array:待转换数组。

callbackArray:为每个数组元素调用，而且会给这个转换函数传递一个表示被转换的元素作为参数。函数可返回任何值。另外，此函数可设置为一个字符串，当设置为字符串时，将视为“lambda-form”（缩写形式？），其中 a 代表数组元素。如“a * a”代表“function(a){ return a * a; }”。

arrayOrObject,callback Array/Object,Function *V1.6*

arrayOrObject:数组或者对象。

为每个数组元素调用，而且会给这个转换函数传递一个表示被转换的元素作为参数。函数可返回任何值。另外，此函数可设置为一个字符串，当设置为字符串时，将视为“lambda-form”（缩写形式？），其中 a 代表数组元素。如“a * a”代表“function(a){ return a * a; }”。

示例

描述:

将原数组中每个元素加 4 转换为一个新数组。

jQuery 代码:

```
$.map( [0,1,2], function(n){
  return n + 4;
});
```

结果:

```
[4, 5, 6]
```

描述:

原数组中大于 0 的元素加 1 ，否则删除。

jQuery 代码:

```
$.map( [0,1,2], function(n){
  return n > 0 ? n + 1 : null;
});
```

结果:

```
[2, 3]
```

描述:

原数组中每个元素扩展为一个包含其本身和其值加 1 的数组，并转换为一个新数组。

jQuery 代码:

```
$.map( [0,1,2], function(n){
  return [ n, n + 1 ];
});
```

结果:

```
[0, 1, 1, 2, 2, 3]
```

jQuery.inArray(val,arr,[from])

返回值:NumberjQuery.inArray(value,array,[fromIndex])

概述

确定第一个参数在数组中的位置，从0开始计数(如果没有找到则返回 -1)。

参数

value,array,[fromIndex]Any,Array,Number*V1.2*

value:用于在数组中查找是否存在

array:待处理数组。

fromIndex:用来搜索数组队列，默认值为0。

arrayArray

待处理数组。

示例

描述:

查看对应元素的位置

jQuery 代码:

```
var arr = [ 4, "Pete", 8, "John" ];
jQuery.inArray("John", arr);  //3
jQuery.inArray(4, arr);  //0
jQuery.inArray("David", arr);  //-1
jQuery.inArray("Pete", arr, 2);  //-1
```

jQuery.toArray()

返回值:NumberjQuery.toArray()

*V1.4*概述

把jQuery集合中所有DOM元素恢复成一个数组。

示例

描述:

得到所有li的元素数组

jQuery 代码:

```
alert($('li').toArray());
```

结果:

```
[<li id="foo">, <li id="bar">]
```

jQuery.sub()

返回值:jQueryjQuery.sub()

*V1.5*概述

可创建一个新的jQuery副本，不影响原有的jQuery对像。

有两个具体使用jQuery.sub（ ）创建案例。首先是提供完全没有破坏jQuery原有一切的方法，另一个用于帮助做jQuery插件封装和基本命名空间。

请注意，jQuery.sub（ ）不会做任何特殊的隔离 - 这不是它的意图。所有关于jQuery的sub'd版本的方法将仍然指向原来的jQuery。（绑定和触发仍将通过主jQuery的事件，数据将通过主绑定的元素的jQuery，Ajax的查询和活动将通过主jQuery的运行，等等）。

请注意，如果你正在寻找使用这个开发插件，应首先*认真*考虑使用一些类似jQuery UI widget工厂，这两个状态和插件管理子方法。 使用jQuery UI widget的一些例子建立一个插件。

这种方法的具体使用情况下可以通过一些例子最好的描述。

参数

jQuery.sub()

示例

描述:

添加一个jQuery的方法，以便它不会受到外部分：

jQuery 代码:

```
(function(){
  var sub$ = jQuery.sub();

  sub$.fn.myCustomMethod = function(){
    return 'just for me';
  };

  sub$(document).ready(function() {
    sub$('body').myCustomMethod() // 'just for me'
  });
})();

typeof jQuery('body').myCustomMethod // undefined
```

描述:

改写一些jQuery的方法，以提供新的功能。

jQuery 代码:

```
(function() {
  var myjQuery = jQuery.sub();

  myjQuery.fn.remove = function() {
    // New functionality: Trigger a remove event
    this.trigger("remove");

    // Be sure to call the original jQuery remove method
    return jQuery.fn.remove.apply( this, arguments );
  };

  myjQuery(function($) {
    $(".menu").click(function() {
      $(this).find(".submenu").remove();
    });

    // A new remove event is now triggered from this copy of jQuery
    $(document).bind("remove", function(e) {
      $(e.target).parent().hide();
    });
  });
})();

// Regular jQuery doesn't trigger a remove event when removing an element
// This functionality is only contained within the modified 'myjQuery'.
```

描述:

创建一个插件，它返回插件的具体办法。

jQuery 代码:

```
(function() {
  // Create a new copy of jQuery using sub()
  var plugin = jQuery.sub();

  // Extend that copy with the new plugin methods
  plugin.fn.extend({
    open: function() {
      return this.show();
    },
    close: function() {
      return this.hide();
    }
  });

  // Add our plugin to the original jQuery
  jQuery.fn.myplugin = function() {
    this.addClass("plugin");

    // Make sure our plugin returns our special plugin version of jQuery
    return plugin( this );
  };
})();

$(document).ready(function() {
  // Call the plugin, open method now exists
  $('#main').myplugin().open();

  // Note: Calling just $("#main").open() won't work as open doesn't exist!
});
```

jQuery.when(deferreds)

返回值:DeferredjQuery.when(deferreds)

概述

提供一种方法来执行一个或多个对象的回调函数，延迟对象通常表示异步事件。

如果单一延迟传递给jQuery.when ，它是通过这个方法和延迟对象附加的其他方法可访问绑定的回调函数返回的，如defered.then 。当延迟得到解决或者拒绝，通常的代码创建了原来的延迟，适当的回调将被调用。例如，jqXHR对象返回jQuery.ajax是一个延期，可以用这种方式：

```
$.when( $.ajax("test.aspx") ).then(function(ajaxArgs){
    alert(ajaxArgs[1]); /* ajaxArgs is [ "success", statusText, jqXHR ] */
});
```

如果一个参数被传递给jQuery.when ，这不是延迟，这将被视为延迟解决，并立即将执行附加任何doneCallbacks。该doneCallbacks传递原始的参数。在这种情况下，任何failCallbacks您可能会设置是永远不会被调用，因为延迟从不拒绝。

例如：

```
$.when( { testing: 123 } ).done(
   function(x){ alert(x.testing); } /* alerts "123" */
);
```

在案例中有多个延迟对象传递jQuery.when ，该方法返回一个新的“宿主”延迟对象，跟踪所有已通过Deferreds聚集状态。该方法能够解决它的“宿主”延迟尽快解决所有延迟，或拒绝尽快将被拒绝的延迟。如果“宿主”延迟得到解决，它是通过传递给解析值，所有的延迟 jQuery.when 。例如，当延迟是jQuery.ajax()请求，参数将是jqXHR对象的要求，以便他们在名单内的说法。

在多延迟情况下，如果延迟一被拒绝，jQuery.when火灾立即掌握其推迟failCallbacks。请注意，延迟一些可能仍然在这一点没有得到解决。如果您需要执行额外的处理对于这种情况，如取消任何未完成的Ajax请求，你可以保持基本jqXHR引用对象在封闭和检查/取消在failCallback他们。

参数

deferreds *V1.5*

一个或多个延迟对象，或者普通的JavaScript对象。

示例

描述:

执行Ajax请求后两个函数是成功的。（见jQuery.ajax()对于一个成功的和错误的案件为AJAX请求的完整描述文档）。

jQuery 代码:

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php")).done(function(a1,  a2){
    /* a1 and a2 are arguments resolved for the
       page1 and page2 ajax requests, respectively */
   var jqXHR = a1[2]; /* arguments are [ "success", statusText, jqXHR ] */
   if ( /Whip It/.test(jqXHR.responseText) ) {
     alert("First page has 'Whip It' somewhere.");
   }
});
```

执行函数myfunc当两个Ajax请求是成功的，如果任一或myFailure有一个错误。

jQuery 代码:

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php"))
  .then(myFunc, myFailure);
```

jQuery.merge(first,second)

返回值:ArrayjQuery.merge(first,second)

概述

合并两个数组

返回的结果会修改第一个数组的内容——第一个数组的元素后面跟着第二个数组的元素。要去除重复项，请使用$.unique()

参数

first,second Array,Array *V1.0*

first:第一个待处理数组，会改变其中的元素。

second:第二个待处理数组，不会改变其中的元素。

示例

描述:

合并两个数组到第一个数组上。

jQuery 代码:

```
$.merge( [0,1,2], [2,3,4] )
```

结果:

```
[0,1,2,2,3,4]
```

jQuery.unique(array)

返回值:ArrayjQuery.unique(array)

概述

删除数组中重复元素。只处理删除DOM元素数组，而不能处理字符串或者数字数组。

参数

arrayArray *V1.1.3*

待处理数组。

示例

描述:

删除重复 div 标签。

jQuery 代码:

```
$.unique(document.getElementsByTagName("div"));
```

结果:

```
[<div>, <div>, ...]
```

jQuery.parseJSON(json)

返回值:StringjQuery.parseJSON(json)

概述

接受一个JSON字符串，返回解析后的对象。

传入一个畸形的JSON字符串会抛出一个异常。比如下面的都是畸形的JSON字符串：

- {test: 1}（ test 没有包围双引号 ）
- {'test': 1}（ 使用了单引号而不是双引号 ）

另外，如果你什么都不传入，或者一个空字符串、null或undefined，parseJSON都会返回 null 。

参数

jsonString *V1.4.1*

要解析的JSON字符串

示例

描述:

解析一个JSON字符串

jQuery 代码:

```
var obj = jQuery.parseJSON('{"name":"John"}');
alert( obj.name === "John" );
```

jQuery.noop

返回值:ArrayjQuery.noop

*V1.4*概述

一个空函数

当你仅仅想要传递一个空函数的时候，就用他吧。这对一些插件作者很有用，当插件提供了一个可选的回调函数接口，那么如果调用的时候没有传递这个回调函数，就用jQuery.noop来代替执行。

jQuery.proxy(function,context)

返回值:BooleanjQuery.proxy(function,context)

概述

jQuery 1.4 新增。返回一个新函数，并且这个函数始终保持了特定的作用域。

当有事件处理函数要附加到元素上，但他们的作用域实际是指向另一个对象时，这个方法最有用了。此外，最妙的是，jQuery能够确保即便你绑定的函数是经过jQuery.proxy()处理过的函数，你依然可以传递原先的函数来准确无误地取消绑定。请参考下面的例子。

这个函数还有另一种用法，jQuery.proxy(scope, name)。第一个参数是要设定的作用域对象。第二个参数是将要设置作用域的函数名（必须是第一个作用域对象的一个属性）。

参数

function,context Function,Object *V1.4*

function:将要被改变作用域的函数

context:一个object，那个函数的作用域会被设置到这个object上来。

context,nameObject,Object *V1.4*

context:一个object，那个函数的作用域会被设置到这个object上来。

name：改变上下文中的函数名(这个函数必须是前一个参数 'context' 对象的属性)

示例

描述:

强制设置函数的作用域，让this指向obj而不是#test对象。

HTML 代码:

```
<div id="test">Click Here!</div>
```

jQuery 代码:

```
var obj = {
  name: "John",
  test: function() {
    alert( this.name );
    $("#test").unbind("click", obj.test);
  }
};

$("#test").click( jQuery.proxy( obj, "test" ) );

// 以下代码跟上面那句是等价的:
// $("#test").click( jQuery.proxy( obj.test, obj ) );

// 可以与单独执行下面这句做个比较。
// $("#test").click( obj.test );
```

jQuery.contains(container, contained)

返回值:BooleanjQuery.contains(container,contained)

概述

一个DOM节点是否包含另一个DOM节点。

参数

container,containedObject,Object *V1.4*

container:DOM元素作为容器，可以包含其他元素

contained:DOM节点，可能被其他元素所包含

示例

描述:

检测一个元素是否包含另一个元素

jQuery 代码:

```
jQuery.contains(document.documentElement, document.body); // true
jQuery.contains(document.body, document.documentElement); // false
```

jQuery.isArray(obj)

返回值:BooleanjQuery.isArray(obj)

概述

jQuery 1.3 新增。测试对象是否为数组。

参数

objObject *V1.3*

用于测试是否为数组的对象

示例

描述:

检测是否为数组

jQuery 代码:

```
$("b").append( " + $.isArray([]) );
```

结果:

```
<b>true</b>
```

jQuery.isFunction(obj)

返回值:BooleanjQuery.isFunction(obj)

概述

测试对象是否为函数。

注意：jQuery 1.3以后，在IE浏览器里，浏览器提供的函数比如'alert'还有 DOM 元素的方法比如 'getAttribute' 将不认为是函数

参数

objObject *V1.2*

用于测试是否为函数的对象

示例

描述:

检测是否为函数

jQuery 代码:

```
function stub() {
   }
var objs = [
            function () {},
            { x:15, y:20 },
            null,
            stub,
            "function"
          ];
        jQuery.each(objs, function (i) {
         var isFunc = jQuery.isFunction(objs[i]);
         $("span:eq( " + i + ")").text(isFunc);
       });
```

结果:

```
[ true,false,false,true,false ]
```

jQuery.isEmptyObject(obj)

返回值:BooleanjQuery.isEmptyObject(obj)

概述

jQuery 1.4 新增。测试对象是否是空对象（不包含任何属性）。

jQuery 1.4 中，这个方法既检测对象本身的属性，也检测从原型继承的属性（因此没有使用 hasOwnProperty）。

参数

objObject *V1.4*

用于测试是否为空对象

示例

描述:

测试是否为空对象

jQuery 代码:

```
jQuery.isEmptyObject({}) // true
jQuery.isEmptyObject({ foo: "bar" }) // false
```

jQuery.isPlainObject(obj)

返回值:BooleanjQuery.isPlainObject(obj)

概述

测试对象是否是纯粹的对象（通过 "{}" 或者 "new Object" 创建的）。

参数

objObject *V1.4*

用于测试是否为纯粹的对象

示例

描述:

测试是否为纯粹的对象

jQuery 代码:

```
jQuery.isPlainObject({}) // true
jQuery.isPlainObject("test") // false
```

jQuery.isWindow(obj)

返回值:BooleanjQuery.isWindow(obj)

概述

测试对象是否是窗口（有可能是Frame）。

参数

objObject *V1.4.3*

用于测试是否为窗口的对象

示例

描述:

测试是否为窗口

jQuery 代码:

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.5.2.js"></script>
</head>
<body>
 Is 'window' a window? <b></b>
<script>$("b").append( "" + $.isWindow(window) );</script>

</body>
</html>
```

jQuery.isNumeric(value)

返回值:BooleanjQuery.isNumeric(value)

概述

确定它的参数是否是一个数字。

$.isNumeric() 方法检查它的参数是否代表一个数值。如果是这样，它返回 true。否则，它返回false。该参数可以是任何类型的

参数

value *V1.7*

用于测试的值。

示例

描述:

Sample return values of $.isNumeric with various inputs.

jQuery 代码:

```
$.isNumeric("-10");  // true
$.isNumeric(16);     // true
$.isNumeric(0xFF);   // true
$.isNumeric("0xFF"); // true
$.isNumeric("8e5");  // true (exponential notation string)
$.isNumeric(3.1415); // true
$.isNumeric(+10);    // true
$.isNumeric(0144);   // true (octal integer literal)
$.isNumeric("");     // false
$.isNumeric({});     // false (empty object)
$.isNumeric(NaN);    // false
$.isNumeric(null);   // false
$.isNumeric(true);   // false
$.isNumeric(Infinity); // false
$.isNumeric(undefined); // false
```

jQuery.type(obj)

返回值:StringjQuery.type(obj)

概述

检测obj的数据类型。

参数

objObject *V1.4.3*

用于测试类型的对象

示例

描述:

以下obj的类型如下：

```
jQuery.type(true) === "boolean"
      jQuery.type(3) === "number"jQuery.type("test") === "string"jQuery.type(function(){}) === "functi
on"jQuery.type([]) === "array"jQuery.type(new Date()) === "date"jQuery.type(/test/) === "regexp"
```

jQuery.trim(str)

返回值:StringjQuery.trim(str)

概述

去掉字符串起始和结尾的空格。

参数

strString *V1.0*

需要处理的字符串

示例

描述:

去掉字符串起始和结尾的空格。

jQuery 代码:

```
$.trim("  hello, how are you?  ");
```

结果:

```
"hello, how are you?"
```

jQuery.param(obj,[traditional])

返回值:StringjQuery.param(obj,*[traditional]*)

概述

将表单元素数组或者对象序列化。是.serialize()的核心方法。

在jQuery 1.3中，如果传递的参数是一个函数，那么用.param()会得到这个函数的返回值，而不是把这个函数作为一个字符串来返回。

在jQuery 1.4中，.param()会深度递归一个对象来满足现在脚本语言和框架，比如PHP， Ruby on Rails等。你可以通过jQuery.ajaxSettings.traditional = true; 来全局得禁用这个功能。

注意：因为有些框架在解析序列化数字的时候能力有限，所以当传递一些含有嵌套对象、数组的对象作为参数时，请务必小心！

在jQuery 1.4中，HTML5的input元素也会被序列化。

参数

objArray/jQuery/Object *V1.2*

数组或jQuery对象会按照name/value对进行序列化，普通对象按照key/value对进行序列化。

obj,[traditional]Array/jQuery/Object,Boolean *V1.4*

obj:数组或jQuery对象会按照name/value对进行序列化，普通对象按照key/value对进行序列化。

traditional:是否使用传统的方式浅层序列化。

示例

描述:

按照key/value对序列化普通对象。

jQuery 代码:

```
var params = { width:1680, height:1050 };
var str = jQuery.param(params);
$("#results").text(str);
```

结果:

```
width=1680&height=1050
```

描述:

对比两种序列化方式

jQuery 代码:

```
var myObject = {
  a: {
    one: 1,
    two: 2,
    three: 3
  },
  b: [1,2,3]
};
var recursiveEncoded = $.param(myObject);
var recursiveDecoded = decodeURIComponent($.param(myObject));

alert(recursiveEncoded);
alert(recursiveDecoded);
//a%5Bone%5D=1&a%5Btwo%5D=2&a%5Bthree%5D=3&b%5B%5D=1&b%5B%5D=2&b%5B%5D=
3
//a[one]=1&a[two]=2&a[three]=3&b[]=1&b[]=2&b[]=3

var shallowEncoded = $.param(myObject, true);
var shallowDecoded = decodeURIComponent(shallowEncoded);

alert(shallowEncoded);
alert(shallowDecoded);
//a=%5Bobject+Object%5D&b=1&b=2&b=3
//a=[object+Object]&b=1&b=2&b=3
```

jQuery.error(message)

返回值:StringjQuery.error(message)

概述

接受一个字符串，并且直接抛出一个包含这个字符串的异常。

这个方法的主要目的是提供给插件开发人员，让他们可以重载这个方法，并以更好的方式显示错误，或者提供更多信息。

参数

messageString *V1.4.1*

要抛出的消息

示例

描述:

重载jQuery.error，以便可以在Firebug里显示

jQuery 代码:

```
jQuery.error = console.error;
```

Deferred

def.done(donCal,[donCal])

返回值:Deferred Objectdeferred.done(doneCallbacks[,doneCallbacks])

概述

当延迟成功时调用一个函数或者数组函数.

该参数可以是一个函数或一个函数的数组。当延迟成功时，doneCallbacks被调用。回调执行是依照他们添加的顺序。一旦deferred.done()返回延迟对象，延迟对象的其它方法也可以链接到了这里，包括增加.done()方法。当延迟解决，doneCallbacks执行使用参数提供给resolve或resolveWith方法依照添加的顺序调用。有关详细信息，请参阅Deferred object 。

参数

doneCallbacksFunction/Array *V1.5*

一个函数或者数组函数，延迟成功时调用

doneCallbacksFunction/Array *V1.5*

附加可选的函数或数组函数，延迟成功时调用

示例

描述:

一旦jQuery.get方法返回一个来自延迟的对象的jqXHR对象，我们可以附加一个成功回调使用.done方法。

jQuery 代码:

```
$.get("test.php").done(function() {
  alert("$.get succeeded");
});
```

def.fail(failCal)

返回值:Deferred Objectdeferred.fail(failCallbacks[,failCallbacks])

概述

当延迟失败时调用一个函数或者数组函数.。

该参数可以是一个函数或一个函数的数组。当延迟失败时，doneCallbacks被调用。回调执行是依照他们添加的顺序。一旦deferred.fail()返回延迟对象，延迟对象的其它方法也可以链接到了这里，包括增加.done()方法。当延迟解决，doneCallbacks执行使用参数提供给resolve或resolveWith方法依照添加的顺序调用。有关详细信息，请参阅Deferred object 。

参数

failCallbacksFunction/Array *V1.5*

一个函数或者数组函数，延迟失败时调用

failCallbacksFunction/Array *V1.5*

附加可选的函数或数组函数，延迟失败时调用

示例

描述:

一旦jQuery.get方法返回一个jqXHR对象，这是从一个递延所得，可以附加的成功和失败回调使用deferrred.done()和deferred.fail()方法。

jQuery 代码:

```
$.get("test.php")
  .done(function(){ alert("$.get succeeded"); })//延迟成功
  .fail(function(){ alert("$.get failed!"); });//延迟失败
```

def.isRejected()

返回值:Booleandeferred.isRejected()

*V1.5*概述

确定延迟对象是否已被拒绝。

jquery1.7API中已弃用，请用deferred.state()替代。

如果延迟对象是在被拒绝的状态则返回true，这意味着要么deferred.reject()或者deferred.rejectWith()被调用的对象和failCallbacks被访问（或称正在这一进程中的）。

请注意，延迟的对象可以有三种状态：未解决（unresolved），解决（resolved），或拒绝（rejected）；使用deferred.isResolved()来判断延迟对象是否在解决状态 。这些方法主要用于调试时非常有用，例如，以确定是否递延已经解决，即使你在代码中打算拒绝。

def.isResolved()

返回值:Deferred Objectdeferred.isResolved()

概述

确定延迟对象是否已得到解决。

jquery1.7API中已弃用，请用deferred.state()替代。

如果延迟对象是在被解决的状态则返回true，这意味着要么deferred.reject()或者deferred.rejectWith()被调用的对象和failCallbacks被访问（或称正在这一进程中的）。

请注意，延迟的对象可以有三种状态：未解决（unresolved），解决（resolved），或拒绝（rejected）；使用deferred.isResolved()来判断延迟对象是否在解决状态 。这些方法主要用于调试时非常有用，例如，以确定是否递延已经解决，即使你在代码中打算拒绝。

def.reject(args)

返回值:Deferred Objectdeferred.reject(args)

概述

拒绝延迟对象，并根据给定的参数调用任何失败的回调函数。

当延迟被拒绝，任何failCallbacks添加的deferred.then或deferred.fail被调用。回调按他们添加的顺序执行。每个回调传递的args在deferred.reject()中调用。之后添加任何failCallbacks递延被拒绝进入状态时，立即执行添加，使用的参数被传递给.reject()调用。有关详细信息，请参阅文件Deferred object 。

参数

args *V1.5*

传递给failCallbacks的可选参数。

def.rejectWith(context,[args])

返回值:Deferred Objectdeferred.rejectWith(context,[args])

概述

拒绝延迟的对象，并根据给定的上下文和参数调用任何失败的回调函数。。

当延迟被拒绝，任何doneCallbacks添加的deferred.then或deferred.fail被调用。回调按他们添加的顺序执行。每个回调传递的args在deferred.reject()中调用。之后添加任何failCallbacks递延被拒绝进入状态时，立即执行添加，使用的参数被传递给.reject()调用。有关详细信息，请参阅文件Deferred object 。

参数

context *V1.5*

上下文作为this对象传递给failCallbacks 。

args *V1.5*

传递给failCallbacks的可选参数。

def.resolve(args)

返回值:Deferred Objectdeferred.resolve(args)

概述

解决递延对象，并根据给定的参数调用任何完成的回调函数。

当递延被解决，任何failCallbacks添加的deferred.then或deferred.fail被调用。回调按他们添加的顺序执行。每个回调传递的args在deferred.reject()中调用。之后添加任何failCallbacks递延被拒绝进入状态时，立即执行添加，使用的参数被传递给.reject()调用。有关详细信息，请参阅文件Deferred object 。

参数

argsString *V1.5*

传递给doneCallbacks的可选参数

def.resolveWith(context,args)

返回值:Deferred Objectdeferred.resolveWith(context,[args])

概述

解决递延对象，并根据给定的上下文和参数调用任何完成的回调函数。

当递延被解决，任何doneCallbacks 添加的deferred.then或deferred.fail被调用。回调按他们添加的顺序执行。每个回调传递的args在deferred.reject()中调用。之后添加任何failCallbacks递延被拒绝进入状态时，立即执行添加，使用的参数被传递给.reject()调用。有关详细信息，请参阅文件Deferred object 。

参数

context *V1.5*

上下文作为this对象传递给 doneCallbacks 。

args *V1.5*

传递给doneCallbacks的可选参数

def.then(doneCal,failCal)

返回值:Deferred Objectdeferred.then(doneCallbacks,failCallbacks[,progressCallbacks])

概述

添加处理程序被调用时，递延对象得到解决或者拒绝。

所有三个参数（包括progressCallbacks ，在jQuery的1.7 ）可以是一个单独的函数或一个函数的数组。其中一个参数，也可以为空，如果没有该类型的回调是需要的。或者，使用.done()或.fail()仅设置doneCallbacks或failCallbacks。当递延解决，doneCallbacks被调用。若递延代替拒绝，failCallbacks被调用。回调按他们添加的顺序执行。一旦deferred.then返回延迟对象，延迟对象的其它方法也可以链接到了这里，包括增加.then()方法。有关详细信息，请参阅文件Deferred object 。

参数

doneCallbacks,failCallbacksString *V1.5*

doneCallbacks: 一个函数或函数数组，当延迟成功时调用。

failCallbacks: 一个函数或函数数组，当延迟失败时调用。

doneCallbacks, failCallbacks [, progressCallbacks]String *V1.7*

doneCallbacks: 一个函数或函数数组，当延迟解决时调用。

failCallbacks: 一个函数或函数数组，当延迟拒绝时调用。

progressCallbacks: 一个可选的函数，当进度递延通知呼叫。

示例

描述:

一旦jQuery.get方法返回一个来自延迟的对象的jqXHR对象，我们可以附加一个成功回调使用.then方法。

jQuery 代码:

```
$.get("test.php").then(
    function(){ alert("$.get succeeded"); },
    function(){ alert("$.get failed!"); }
);
```

def.progress([type],[target])

返回值:Deferred Objectdeferred.progress(progressCallbacks)

概述

当Deferred对象时生成进度通知时添加被访问处理程序。

这个参数可以是一个单一的功能或功能的阵列。当递延通过调用生成的进度通知notify或notifyWith，progressCallbacks 被访问。后来deferred.progress()返回Deferred对象。Deferred对象有方法可以链接到这一个。当递延解决或拒绝，进展回调将不再被调用。欲了解更多信息，请参阅文档Deferred object。

参数

progressCallbacks *V1.7*

一个函数或函数数组，被呼叫递延生成进度通知。

def.pipe([donFil],[faiFil],[proFil])

返回值:Deferred Objectdeferred.pipe([doneFilter],[failFilter],[progressFilter])

概述

筛选器和/或链Deferreds的实用程序方法。

deferred.pipe()方法返回一个新的promise，该过滤器通过一个函数有关的递延状态和价值。该doneFilter和failFilter原递延过滤功能的解决/拒绝的状态和价值。这些过滤器函数可以返回一个新的值被传递给管道承诺的done()或fail()的回调，或者他们可以返回另一个观察对象（推迟，承诺等），将通过它的解决/拒绝状态和价值，以保证管道的回调。如果使用的是过滤功能null ，或不指定，则管道的承诺将得到解决或原驳回值具有相同。

参数

doneFilter,failFilter *V1.6*

doneFilter:可选函数，当递延得到解决时调用。

failFilter:可选函数，当递延得被拒绝时调用。

doneFilter,failFilter,progressFilter *V1.7*

doneFilter:可选函数，当递延得到解决时调用。

failFilter:可选函数，当递延得被拒绝时调用。

progressFilter:一个可选的函数会在延迟调用被拒绝时被调用

示例

描述:

过滤解决值:

jQuery 代码:

```
var defer = $.Deferred(),
    filtered = defer.pipe(function( value ) {
      return value * 2;
    });

defer.resolve( 5 );
filtered.done(function( value ) {
  alert( "Value is ( 2*5 = ) 10: " + value );
});
```

描述:

过滤器拒值:

jQuery 代码:

```
var defer = $.Deferred(),
    filtered = defer.pipe( null, function( value ) {
      return value * 3;
    });

defer.reject( 6 );
filtered.fail(function( value ) {
  alert( "Value is ( 3*6 = ) 18: " + value );
});;
```

描述:

链任务:

jQuery 代码:

```
var request = $.ajax( url, { dataType: "json" } ),
    chained = request.pipe(function( data ) {
      return $.ajax( url2, { data: { user: data.userId } } );
    });

chained.done(function( data ) {
  // data retrieved from url2 as provided by the first request
});
```

def.always(alwCal,[alwCal])

返回值:Deferred Objectjdeferred.always(alwaysCallbacks, [alwaysCallbacks])

概述

当递延对象是解决或拒绝时被调用添加处理程序。

参数

alwaysCallbacks *V1.6*

一个函数，或者函数数组，当递延对象是解决或拒绝时被调用。

alwaysCallbacks *V1.6*

附加可选的一个函数，或者函数数组，当递延对象是解决或拒绝时被调用。

示例

描述:

jQuery.get（ ）方法返回一个来自一个延迟的对象的jqXHR对象，我们可以附加一个成功和错误使用deferred.always（ ）方法的回调。

jQuery 代码:

```
$.get("test.php").always( function() {
  alert("$.get completed with success or error callback arguments");
} );
```

def.notify(args)

返回值:Deferred Objectdeferred.notify(args)

概述

调用一个给定args的递延对象上的进行中的回调 （ progressCallbacks ）

通常情况下，只有一个递延的创建者，应调用此方法;你可以防止其他代码改变Deferred的状态或者通过deferred.promise()返回一个受限制的承诺对象报告状态

当 deferred.notify 被访问， 任何progressCallbacks 可以通过访问deferred.then 好或者 deferred.progress来添加。回调是在它们被添加的顺序执行。每次通过来自.notify()的args回调。任何为.notify()后递延是解决或拒绝（或之后添加任何progressCallbacks ）被忽略。欲了解更多信息，请参阅文档 Deferred object.

参数

args *V1.7*

可选参数传递到进行中的回调（progressCallbacks）

def.notifyWith(context,[args])

返回值:Deferred Objectdeferred.notifyWith(context,[args])

概述

去掉字符串起始和结尾的空格。

当deferred.notifyWith，任何doneCallbacks 添加的progressCallbacks，deferred.then或deferred.fail被调用。回调按他们添加的顺序执行。每个回调传递的args在deferred.reject()中调用。notifyWith()延迟后解决或拒绝(或添加任何progressCallbacks后 ）被忽略。有关详细信息，请参阅文件Deferred object 。

参数

contextString *V1.7*

上下文传递progressCallbacks此对象。

argsString *V1.7*

可选参数传递到progressCallbacks。

def.state()

返回值:Deferred Objectdeferred.state()

*V1.7*概述

确定一个Deferred对象的当前状态。

deferred.state（ ）方法返回一个字符串，代表Deferred对象的当前状态。 Deferred对象可以在三种状态之一：

- **pending**: Deferred对象是尚未完成状态 (不是 "rejected" 或 "resolved").
- **resolved**: Deferred对象是在解决状态，这意味着，deferred.resolve() 或者 deferred.resolveWith()被对象访问和doneCallbacks被访问（或在被调用的过程中）。

- **rejected**: Deferred对象是在被拒绝的状态，这意味着， deferred.reject() 或者 deferred.rejectWith() 被对象访问和failCallbacks被访问（或在被调用的过程中）。

  这种方法主要是有用的调试，以确定的，例如，递延是否已经得到解决，即使你打算拒绝它的内部代码。

Callbacks

callbacks.add(callbacks)

返回值:undefinedcallbacks.add(callbacks)

概述

回调列表中添加一个回调或回调的集合。

参数

callbacks *V1.7*

一个函数，或者一个函数数组用来添加到回调列表。 .

示例

描述:

使用 callbacks.add() 添加新的回调到回调列表：

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( 'foo:' + value );
}

// another function to also be added to the list
var bar = function( value ){
    console.log( 'bar:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' );
// outputs: 'foo: hello'

// add the function 'bar' to the list
callbacks.add( bar );

// fire the items on the list again
callbacks.fire( 'world' );

// outputs:
// 'foo: world'
// 'bar: world'
```

callbacks.disable()

返回值:undefinedcallbacks.disable()

*V1.7*概述

禁用回调列表中的回调

示例

描述:

使用 callbacks.disable() 禁用回调列表：

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( value );
}

var callbacks = $.Callbacks();

// add the above function to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'foo' ); // outputs: foo

// disable further calls being possible
callbacks.disable();

// attempt to fire with 'foobar' as an argument
callbacks.fire( 'foobar' ); // foobar isn't output
```

callbacks.empty()

返回值:undefinedcallbacks.empty()

*V1.7*概述

从列表中删除所有的回调.

示例

描述:

使用 callbacks.empty() 清空回调列表:

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value1, value2 ){
    console.log( 'foo:' + value1 + ',' + value2 );
}

// another function to also be added to the list
var bar = function( value1, value2 ){
    console.log( 'bar:' + value1 + ',' + value2 );
}

var callbacks = $.Callbacks();

// add the two functions
callbacks.add( foo );
callbacks.add( bar );

// empty the callbacks list
callbacks.empty();

// check to ensure all callbacks have been removed
console.log( callbacks.has( foo ) ); // false
console.log( callbacks.has( bar ) ); // false
```

callbacks.fire(arguments)

返回值:undefinedcallbacks.fire(arguments)

概述

禁用回调列表中的回调

参数

arguments *V1.7*

这个参数或参数列表传回给回调列表。

示例

描述:

使用 callbacks.fire() 用任何已传递的参数调用列表中的回调:

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'
callbacks.fire( 'world '); // outputs: 'foo: world'

// add another function to the list
var bar = function( value ){
    console.log( 'bar:' + value );
}

// add this function to the list
callbacks.add( bar );

// fire the items on the list again
callbacks.fire( 'hello again' );
// outputs:
// 'foo: hello again'
// 'bar: hello again'
```

callbacks.fired()

返回值:Booleancallbacks.fired()

*V1.7*概述

用给定的参数调用所有的回调。

示例

描述:

使用callbacks.fired() 确定，如果列表中的回调至少有一次被呼叫

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'
callbacks.fire( 'world '); // outputs: 'foo: world'

// test to establish if the callbacks have been called
console.log( callbacks.fired() );
```

callbacks.fireWith([context][,args])

返回值:undefinedcallbacks.fireWith([context][,args])

概述

访问给定的上下文和参数列表中的所有回调

参数

[context][,args] *V1.7*

context: 该列表中的回调被触发的上下文引用

args: 一个参数或参数列表传回给回调列表。

示例

描述:

使用 callbacks.fireWith() 访问给定的上下文和参数列表中的所有回调。

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var log = function( value1, value2 ){
    console.log( 'Received:' + value1 + ',' + value2 );
}

var callbacks = $.Callbacks();

// add the log method to the callbacks list
callbacks.add( log );

// fire the callbacks on the list using the context 'window'
// and an arguments array

callbacks.fireWith( window, ['foo','bar']);

// outputs: Received: foo, bar
```

callbacks.has(callback)

返回值:Booleancallbacks.has(callback)

概述

确定是否提供的回调列表

参数

callback *V1.7*

用来搜索的回调。

示例

描述:

用 callbacks.has() 检查是否回调列表中包含一个特定的回调:

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value1, value2 ){
    console.log( 'Received:' + value1 + ',' + value2 );
}

// a second function which will not be added to the list
var bar = function( value1, value2 ){
    console.log( 'foobar');
}

var callbacks = $.Callbacks();

// add the log method to the callbacks list
callbacks.add( foo );

// determine which callbacks are in the list

console.log( callbacks.has( foo ) ); // true
console.log( callbacks.has( bar ) ); // false
```

callbacks.lock()

返回值:undefinedcallbacks.lock()

*V1.7*概述

锁定在其当前状态的回调列表。

示例

描述:

用 callbacks.lock()锁定一个回调列表，以避免进一步的修改列表状态 :

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( 'foo:' + value);
}

var callbacks = $.Callbacks();

// add the logging function to the callback list
callbacks.add( foo );

// fire the items on the list, passing an argument
callbacks.fire( 'hello' );
// outputs 'foo: hello'

// lock the callbacks list
callbacks.lock();

// try firing the items again
callbacks.fire( 'world' );

// as the list was locked, no items
// were called so 'world' isn't logged
```

callbacks.locked()

返回值:Booleancallbacks.locked()

*V1.7*概述

确定是否已被锁定的回调列表。

示例

描述:

用 callbacks.locked() 确定是否已被锁定的回调列表。:

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( 'foo:' + value);
}

var callbacks = $.Callbacks();

// add the logging function to the callback list
callbacks.add( foo );

// fire the items on the list, passing an argument
callbacks.fire( 'hello' );
// outputs 'foo: hello'

// lock the callbacks list
callbacks.lock();

// test the lock-state of the list
console.log ( callbacks.locked() ); //true
```

callbacks.remove(callbacks)

返回值:undefinedcallbacks.remove(callbacks)

概述

删除回调或回调回调列表的集合。

参数

callbacks *V1.7*

一个函数或函数数组，是从回调列表中删除。

示例

描述:

用callbacks.remove() 删除回调或回调回调列表的集合。

jQuery 代码:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
    console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'

// remove 'foo' from the callback list
callbacks.remove( foo );

// fire the items on the list again
callbacks.fire( 'world' );

// nothing output as 'foo' is no longer in the list/code>
```

jQuery.callbacks(flags)

返回值:jQueryjQuery.callbacks(flags)

概述

一个多用途的回调列表对象，提供了强大的的方式来管理回调函数列表。

$.Callbacks()的内部提供了jQuery的$.ajax() 和 $.Deferred() 基本功能组件。它可以用来作为类似基础定义的新组件的功能。

$.Callbacks() 支持的方法，包括 callbacks.add(),callbacks.remove(), callbacks.fire() and callbacks.disable().

参数

flags *V1.7*

一个用空格标记分隔的标志可选列表,用来改变回调列表中的行为

示例

入门描述:

以下是两个样品的方法命名fn1 and fn2:

jQuery 代码:

```
function fn1( value ){
    console.log( value );
}

function fn2( value ){
    fn1("fn2 says:" + value);
    return false;
}
```

这些可以添加为回调函数作为一个$.Callbacks的列表，并调用如下:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );
callbacks.fire( "foo!" ); // outputs: foo!

callbacks.add( fn2 );
callbacks.fire( "bar!" ); // outputs: bar!, fn2 says: bar!
```

这样做的结果是，它使构造复杂的回调列表变得简单，输入值可以通过尽可能多的函数根据需要轻松使用。

用于以上的两个具体的方法： .add() 和 .fire() .add() 支持添加新的回调回调列表, 而.fire() 提供了一种用于处理在同一列表中的回调方法的途径.

另一种方法由$.Callbacks 的remove()，用于从回调列表中删除一个特定的回调。下面是.remove() 使用的一个例子:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );
callbacks.fire( "foo!" ); // outputs: foo!

callbacks.add( fn2 );
callbacks.fire( "bar!" ); // outputs: bar!, fn2 says: bar!

callbacks.remove(fn2);
callbacks.fire( "foobar" );

// only outputs foobar, as fn2 has been removed.
```

支持的 Flags描述:

这个 flags 参数是$.Callbacks()的一个可选参数, 结构为一个用空格标记分隔的标志可选列表,用来改变回调列表中的行为 (比如. $.Callbacks('unique stopOnFalse')).

**可用的 flags: **

- once: 确保这个回调列表只执行一次(像一个递延 Deferred).
- memory: 保持以前的值和将添加到这个列表的后面的最新的值立即执行调用任何回调 (像一个递延 Deferred).
- unique: 确保一次只能添加一个回调(所以有没有在列表中的重复).
- stopOnFalse: 当一个回调返回false 时中断调用

默认情况下，回调列表将像事件的回调列表中可以多次触发。

如何在理想情况下应该使用的flags的例子，见下文：

$.Callbacks('once'):

```
var callbacks = $.Callbacks( "once" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
*/
```

$.Callbacks('memory'):

```
var callbacks = $.Callbacks( "memory" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
fn2 says:foo
bar
fn2 says:bar
foobar
*/
```

$.Callbacks('unique'):

```
var callbacks = $.Callbacks( "unique" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn1 ); // repeat addition
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
bar
fn2 says:bar
foobar
*//code>
```

$.Callbacks('stopOnFalse'):

```
function fn1( value ){
    console.log( value );
    return false;
}

function fn2( value ){
    fn1("fn2 says:" + value);
    return false;
}

var callbacks = $.Callbacks( "stopOnFalse");
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
bar
foobar
*/
```

因为$.Callbacks() 支持一个列表的flags而不仅仅是一个，设置几个flags，有一个累积效应，类似“&&”。这意味着它可能结合创建回调名单，*unique* 和*确保如果名单已经触发，将有更多的回调调用最新的触发值* (i.e.$.Callbacks("unique memory")).

$.Callbacks('unique memory'):

```
function fn1( value ){
    console.log( value );
    return false;
}

function fn2( value ){
    fn1("fn2 says:" + value);
    return false;
}

var callbacks = $.Callbacks( "unique memory" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn1 ); // repeat addition
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.add( fn2 );
callbacks.fire( "baz" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
fn2 says:foo
bar
fn2 says:bar
baz
fn2 says:baz
foobar
*/
```

Flag结合体是使用的$.Callbacks()内部的.done() 和 .fail()一个递延容器-它们都使用 $.Callbacks('memory once').

$.Callbacks 方法也可以被分离, 为方便起见应该有一个需要定义简写版本：

```
var callbacks = $.Callbacks(),
    add = callbacks.add,
    remove = callbacks.remove,
    fire = callbacks.fire;

add( fn1 );
fire( "hello world");
remove( fn1 );
```

$.Callbacks, $.Deferred and Pub/Sub

pub / sub（ Observer模式 ）背后的一般思路 是促进应用程序的松散耦合。而比对其他对象的方法调用的单个对象，一个对象，而不是另一个对象的一个特定的任务或活动，并通知当它发生。观察家也被称为订

阅者，我们指的出版商（或主体）观察对象。出版商事件发生时通知用户

作为一个组件$.Callbacks()创造能力，它可以实现一个pub / sub系统只使用回调列表。使用$.Callbacks作为主题队列，发布和订阅的主题系统可以实现如下:

```
var topics = {};

jQuery.Topic = function( id ) {
    var callbacks,
        method,
        topic = id && topics[ id ];
    if ( !topic ) {
        callbacks = jQuery.Callbacks();
        topic = {
            publish: callbacks.fire,
            subscribe: callbacks.add,
            unsubscribe: callbacks.remove
        };
        if ( id ) {
            topics[ id ] = topic;
        }
    }
    return topic;
};
```

然后，可以很容易的使用这部分应用程序的发布和订阅感兴趣的事件：

```
// Subscribers
$.Topic( "mailArrived" ).subscribe( fn1 );
$.Topic( "mailArrived" ).subscribe( fn2 );
$.Topic( "mailSent" ).subscribe( fn1 );

// Publisher
$.Topic( "mailArrived" ).publish( "hello world!" );
$.Topic( "mailSent" ).publish( "woo! mail!" );

// Here, "hello world!" gets pushed to fn1 and fn2
// when the "mailArrived" notification is published
// with "woo! mail!" also being pushed to fn1 when
// the "mailSent" notification is published.

/*
output:
hello world!
fn2 says: hello world!
woo! mail!
*/
```

虽然这是有用的，可以采取进一步的实施。使用$.Deferreds,这是可能的，以确保发表者只为用户发布一次特别的任务已经完成（解决）通知。这可能是如何在实践中使用的一些进一步的评论，请参见下面的代

码示例：

```
// subscribe to the mailArrived notification
$.Topic( "mailArrived" ).subscribe( fn1 );

// create a new instance of Deferreds
var dfd = $.Deferred();

// define a new topic (without directly publishing)
var topic = $.Topic( "mailArrived" );

// when the deferred has been resolved, publish a
// notification to subscribers
dfd.done( topic.publish );

// Here the Deferred is being resolved with a message
// that will be passed back to subscribers. It's possible to
// easily integrate this into a more complex routine
// (eg. waiting on an ajax call to complete) so that
// messages are only published once the task has actually
// finished.
dfd.resolve( "its been published!" );
```

关于

关于jQuery API 文档

关于jQuery API 中文文档

概述

官方jQuery API ：http://api.jquery.com/
官方jQuery 源码：http://code.jquery.com/

因国内jquery中文手册更新太慢了,等了一段时间实在等不下去了,干脆自己动手做一个丰衣足食,时刻更新.

最后感谢Shawphy提供1.4.1版,jehn提供1.5.2版,julying提供1.7

hemin制作【hm@hemin.cn】

提交bug及获取更新

提交bug及获取更新

概述

如果大家使用过程中发现了什么翻译错误，可以给我Email：hm@hemin.cn来反馈。

版本信息：

$Rev: 1.8.0 $
$Author: hemin $
$Date: 2012-08-29 02:57:10 +0800 (周三)

检查更新：

检查更新

hemin制作【hm@hemin.cn】

其它

HTML5速查表

HTML5速查表

标签	描述	版本	属性
<!-- ... -->	定义注释	4 / 5	none
<!DOCTYPE>	定义文档类型	4 / 5	none
<a>	定义超链接，用于从一个页面链接到另一个页面。	4 / 5	href | hreflang | media | ping | rel | target | type

<abbr>	定义缩写	4 / 5	全局属性
<acronym>	定义首字母缩写	4	-
<address>	定义文档作者或拥有者的联系信息.	4 / 5	全局属性
<applet>	定义嵌入的 applet	4	-

<area>	定义图像映射内部的区域（图像映射指的是带有可点击区域的图像）。	4 / 5	alt | coords | href | hreflang | media | ping | rel | shape | target | type
<article>	定义独立的内容	5	全局属性
<aside>	定义页面内容之外的内容	5	全局属性
<audio>	定义声音，比如音乐或其他音频流。	5	autobuffer | autoplay | controls | loop | src
<b>	定义粗体文本，用于强调某些文本	4 / 5	全局属性

<base>	定义页面上的所有链接规定默认地址或默认目标	4 / 5	href | target
<basefont>	定义文档中所有文本的默认颜色、大小和字体	4	-
<bb>	定义文本的文本方向，使其脱离其周围文本的方向设置。	5	type
<bdi>	指的是bidi隔离,允许您设置一段文本，使其脱离其父元素的文本方向设置 .	5	dir

<bdo>	定义文本显示的方向。	4 / 5	dir
<big>	定义大号文本。	4	-
<blockquote>	定义摘自另一个源的块引用。	4 / 5	cite

<body>	定义文档的主体 。	5	全局属性
 	插入换行符。	4 / 5	全局属性
<button>	定义按钮。	4 / 5	autofocus | disabled | form | formaction | formenctype | formmethod | formnovalidate | formtarget | name | type | value
<canvas>	定义图形，比如图表和其他图像	5	height | width

<caption>	定义表格标题。	4/5	全局属性
	定义居中的文本。	4	-
<cite>	定义作品（比如书籍、歌曲、电影、电视节目、绘画、雕塑等等）的标题。	4/5	全局属性
<code>	定义计算机代码文本。	4/5	全局属性

<col>	定义为表格中的一个或多个列定义属性值。。	4 / 5	span
<colgroup>	定义对表格中的列进行组合，以便对其进行格式化。	4 / 5	span
<command>	定义命令按钮。	5	checked | default | disabled | hidden | icon | label | radiogroup | type
<datagrid>	定义可选数据的列表。datagrid 作为树列表来显示。	5	disabled | multipe

<datalist>	定义下拉列表。请与 input 元素配合使用该元素，来定义 input 可能的值。	5	全局属性
<dd>	定义一个定义列表中对项目的描述。	4 / 5	全局属性
<del>	定义删除文本。	4 / 5	cite | datetime
<details>	定义描述文档或文档某个部分的细节。	5	open

<dialog>	定义对话，比如交谈	5	全局属性
<dir>	定义目录列表。	4	-
	定义文档中的一个部分。	4 / 5	全局属性
<dfn>	定义一个定义项目。	4 / 5	title
<dl>	定义一个定义列表 .	4 / 5	全局属性
<dt>	定义一个定义列表中的一个项目。	4 / 5	全局属性
	呈现为被强调的文本。	4 / 5	全局属性

<embed>	定义嵌入的内容，比如插件	5	height | src | type | width
<fieldset>	定义用于从逻辑上将表单中的元素组合起来。	4 / 5	disabled | form | name

标签	描述	版本	属性		
	定义 fiqure 元素的标题（caption）。	5	全局属性		
	规定独立的流内容（图像、图表、照片、代码等等）。	5	全局属性		
	规定文本的字体、大小和颜色。	4	-		
	定义 section 或 document 的页脚。	5	全局属性		
	用于创建供用户输入的 HTML 表单	4 / 5	action	date	replace
	定义子窗口（框架）	4	-		

标签	描述	版本	属性
	定义框架集。	4	-
to	定义标题。 定义最大的标题。 定义最小的标题	4 / 5	全局属性
	所有头部元素的容器。位于内部的元素可以包含脚本、指引浏览器找到样式表、提供元信息，等等。	4 / 5	无
	定义文档的页眉（介绍信息）。	5	全局属性
	用于对网页或区段（section）的标题进行组合。	4 / 5	全局属性
	水平线，它应该定义内容中的主题变化	4 / 5	全局属性
	告知浏览器这是一个 HTML 文档。	4 / 5	manifest
	呈现斜体的文本。	4 / 5	全局属性

标签	描述	版本	属性

正则表达式速查表

正则表达式速查表

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个向后引用、或一个八进制转义符。例如，“n”匹配字符“n”。“\n”匹配一个换行符。串行“\\”匹配“\”而“\(”则匹配“(”。
^	匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。

字符	描述
$	匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo*能匹配“z”以及“zoo”。等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“does”或“does”中的“do”。?等价于{0,1}。
{n}	n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。

字符	描述
{n,}	n是一个非负整数。至少匹配n次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o*”。
{n,m}	m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符（*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o”。

字符	描述		
.	匹配除“\``n”之外的任何单个字符。要匹配包括“\``n”在内的任何字符，请使用像“`(.	\n)`”的模式。	
(pattern)	匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“\(”或“\)”。		
(?:pattern)	匹配pattern但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用或字符“`(	) ”来组合一个模式的各个部分是很有用。例如“industr(?:y	ies) ”就是一个industry

字符	描述		
(?=pattern)	正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，“`Windows(?=95	98	NT
(?!pattern)	正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如“`Windows(?!95	98	NT

字符	描述		
(?<=pattern)	反向肯定预查，与正向肯定预查类拟，只是方向相反。例如，“`(?<=95	98	NT
(?	反向否定预查，与正向否定预查类拟，只是方向相反。例如“`(?	98	NT
x	y	匹配x或y。例如，“`z	food”能匹配‘”或“foo”。“(z
[xyz]	字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。		
[^xyz]	负值字符集合。匹配未包含的任意字符。例如，“[^abc]”可以匹配“plain”中的“p”。		

字符	描述
[a-z]	字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。
[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。
\B	匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。
\cx	匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符。

字符	描述
\d	匹配一个数字字符。等价于[0-9]。
\D	匹配一个非数字字符。等价于[^0-9]。
\f	匹配一个换页符。等价于\x0c和\cL。
\n	匹配一个换行符。等价于\x0a和\cJ。
\r	匹配一个回车符。等价于\x0d和\cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于[^\f\n\r\t\v]。
\t	匹配一个制表符。等价于\x09和\cI。
\v	匹配一个垂直制表符。等价于\x0b和\cK。
\w	匹配包括下划线的任何单词字符。等价于“`[A-Za-z0-9_]`”。
\W	匹配任何非单词字符。等价于“`[^A-Za-z0-9_]`”。

字符	描述
\xn	匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码。.
\num	匹配num，其中num是一个正整数。对所获取的匹配的引用。例如，“(.)\1”匹配两个连续的相同字符。
\n	标识一个八进制转义值或一个向后引用。如果\n之前至少n个获取的子表达式，则n为向后引用。否则，如果n为八进制数字（0-7），则n为一个八进制转义值。

字符	描述
\nm	标识一个八进制转义值或一个向后引用。如果\nm之前至少有nm个获得子表达式，则nm为向后引用。如果\nm之前至少有n个获取，则n为一个后跟文字m的向后引用。如果前面的条件都不满足，若n和m均为八进制数字（0-7），则\nm将匹配八进制转义值nm。
\nml	如果n为八进制数字（0-3），且m和l均为八进制数字（0-7），则匹配八进制转义值nml。
\un	匹配n，其中n是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（©）。

常用正则表达式

用户名	/^[a-z0-9_-]{3,16}$/
密码	/^[a-z0-9_-]{6,18}$/
十六进制值	/^#?([a-f0-9]{6}|[a-f0-9]{3})$/
电子邮箱	/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/ /^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/

URL	/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
IP地址	/((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)/ /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
HTML标签	/^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
删除代码\\注释	(?<!http:|\S)//.*$
Unicode编码中的汉字范围	/^[\u2E80-\u9FFF]+$/