?:$(myForm.elements).hide()

// 在 IE 中无效:

$("<input>").attr("type", "checkbox");

// 在 IE 中有效:

$("<input type='checkbox'>");

? $("<div>", {

"class": "test",

text: "Click me!",

click: function(){

$(this).toggleClass("test");

}

}).appendTo("body");

? $("<input>", {

type: "text",

val: "Test",

focusin: function() {

$(this).addClass("active");

},

focusout: function() {

$(this).removeClass("active");

}

}).appendTo(document.forms);

?:index([selector|element])

<ul>

<li id="foo">foo</li>

<li id="bar">bar</li>

<li id="baz">baz</li>

</ul>

$('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])

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。

NEW data(obj) 可传入key-value形式的数据。

// $('em').data("blabla");

$("em").data("blabla","haha");

alert($("em").data("blabla"));

$("em").data("blabla",66);

alert($("em").data("blabla"));

$("em").removeData("blabla");

alert($("em").data("blabla"));

$("em").data("array",{'a':'a','b':'b','c':"c"});

alert($("em").data("array"));

alert($("em").data("array").a);

alert($("em").data("array").b);

removeData([name|list])

在元素上移除存放的数据,与$(...).data(name, value)函数作用相反

alert($("em").removeData("array").b);

alert($("em").data("array").b);

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

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。

jQuery.data(document.body,"foo","heihei");

alert($(document.body).data('foo'));

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

$("#show").click(function(){

var n=$("code").queue("fx");

$("i").text("queue length is:"+n.length);

});

function runit(){

$("code").show("slow");

$("code").animate({left:"+=200"},2000);

$("code").slideToggle(1000);

$("code").slideToggle("fast");

$("code").animate({left:"-=200"},1500);

$("code").hide("slow");

$("code").show(1200);

$("code").slideUp("normal",runit);

}

runit();

参数：element,queueName,newQueue

newQueue:替换当前函数列队内容的数组

通过设定队列数组来删除动画队列

$("#stop").click(function () {

$("div").queue("fx", []);

$("div").stop();

});

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();

$(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:jQuery,队列名，默认为fx

从队列最前端移除一个队列函数，并执行他。

使用 dequeue() 终止一个自定义的队列函数

$("div").queue(function(){

$(this).toggleClass("red");

$(this).dequeue();

});

用dequeue来结束自定义队列函数，并让队列继续进行下去。

$("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:jQuery

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。

queueName:含有队列名的字符串。默认是"Fx"，动画队列。

停止当前正在运行的动画：

$("#stop").click(function(){

$("#box").clearQueue();

});

返回值:jQueryjQuery.noConflict([extreme])

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。

# #选择器

$("\*")找到每一个元素，多用于上下文搜索

$("div,span,p.myClass")找打每一个匹配的元素合并到一个结果内

$("div p")

$("div > p")

$("prevp + p")选择同辈元素的紧挨着下一个选择器，且该选择器必须是p

$("prevp ~ p")选择知之后同辈指定元素的所有选择器，上面的是一个，这里是所有

$("li:first")在指定区域(这里为li)搜索第一个，搜索到即停止搜索

$("ol li:first")指定区域(这里为ul li)

$("li:last")在指定区域搜索最后一个，搜索到即停止搜索，相当于在区域从后向前搜索

$("ol li:last")指定区域(这里为ol li)

？？：$("input:not(:text)")//不明白时尽量少用

$("li:even")

$("li:odd")

$("li:eq(5)")

$("li:lt(3)")不包括3

$("li:gt(7)")不包括7

$(":header")匹配所有h1-h6元素

$("span:animated")匹配所有正在执行动画的元素

$("span:not(:animated)")

？？？$("div:focus");

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

:empty 匹配所有不包含子元素或者文本的空元素

$("div:empty")

$("div:has('p')") $("div:has('.p')")

:parent 匹配含有子元素或者文本的元素

$("div:parent"),匹配的是div

:hidden 匹配所有不可见元素，或者type为hidden的元素

$(":hidden").css("display","block");

:visible 匹配所有的可见元素

示例代码：

<html>

<head>

<meta charset="utf-8">

<title>选择器</title>

<script type="text/javascript" src="../jquery-3.2.1.min.js"></script>

<style>

#box,.mybox{

width:200px;

height:100px;

border:1px solid red;

}

</style>

</head>

<body>

<h1>jquery api</h1>

<h2>选择器</h2>

<div id="box">

<div>

<p>box-p-1</p>

</div>

<p>box-p-2</p>

<p>box-p-3</p>

</div>

<form action="">

<input type="text">

<input type="text">

</form>

<div class="mybox">

</div>

<span>span</span>

<b>这是粗体</b>

<div class="another\_box"></div>

<div class="prev-footer">

<a href="">box</a>

</div>

<div id='footer'></div>

<script>

$(function(){

//alert(1);

//id选择器，返回一个数组，通过索引引用具体的dom对象

console.log($("#box")[0]);

//标签选择器，返回一个数组，通过索引引用具体的dom对象

let divs=$('div');

console.log(divs[0]);

console.log(divs[1]);

//class选择器，返回一个数组，通过索引引用具体的dom对象

console.log($('.mybox')[0]);

//通配符选择器，匹配整个dom中的所有元素，返回一个数组，可以通过索引引用具体的dom对象

let all=$('\*');

console.log(all);

//组合选择器，同时运行多个选择器并将结果一起返回，返回一个数组

let multi=$('span,.mybox,#box,b');

console.log(multi);

//祖先后代选择器，类似css的祖先后代样式,返回一个数组

let p\_in\_box=$('#box p');

console.log(p\_in\_box);

console.log(p\_in\_box.length);//3

//父亲儿子选择器，类似css的父亲儿子样式，返回一二数组

let box\_p=$('#box>p');

console.log(box\_p);

console.log(box\_p.length);//2

//第一兄弟选择器,类似css的第一兄弟样式，返回一个数组

let box\_next=$('#box+div');

console.log(box\_next);

console.log(box\_next.length);//1

//所有之后兄弟选择器，类似css的所有之后兄弟样式，返回一个数组

let box\_siblings=$('#box ~ div');

console.log(box\_siblings);

console.log(box\_siblings.length);//2

//获取第一个元素，类似css的第一个元素样式，尽管是一个元素但是也返回一个数组，就像id选择器一样

let div\_first=$('div:first');

console.log(div\_first);

console.log(div\_first.length);

//获取最后一个元素，类似css的最后一个元素样式，尽管是一个元素但是也返回一个数组，就像id选择器一样

let div\_last=$('div:last');

console.log(div\_last);

//反选选择器，即取出给定的所有，返回一个数组

let not\_div=$(':not(div)');

console.log(not\_div);

console.log(not\_div.length);//13

//在选择器的基础上匹配索引值为偶数的元素，返回一个数组

let div\_even=$('div:even');

console.log(div\_even);

console.log(div\_even.length);

//在选择器的基础上匹配索引值为奇数的元素，返回一个数组

let div\_odd=$('div:odd');

console.log(div\_odd);

console.log(div\_odd.length);

//在选择器的基础上匹配指定索引的元素，返回一个数组，尽管是一个

let div\_2=$('div:eq(2)');

console.log(div\_2);

//在选择器的基础上匹配所有索引大于指定索引的元素，返回一个数组

let div\_gt\_1=$('div:gt(0)');

console.log(div\_gt\_1);

//在选择器的基础上匹配所有索引大于指定索引的元素，返回一个数组

let div\_lt\_1=$('div:lt(1)');

console.log(div\_lt\_1);

//匹配所有的标题元素，即h1~h6，也可以使用多选择器来实现相同的功能，返回一个数组

let h\_all=$(':header');

console.log(h\_all);

let h\_all\_2=$('h1,h2,h3,h4,h5,h6');

console.log(h\_all\_2);

//匹配所有在执行动画效果的元素

let div\_not\_animated=$('div:not(:animated)');

console.log(div\_not\_animated);

//匹配当前获取焦点的元素

let curr\_focus=$('input:not(:focus)');

console.log(curr\_focus);

//获取文档根元素，即html元素，返回一个数组

let root\_elem=$(':root');

console.log(root\_elem);

//匹配包含给定文本的元素，如果指定元素的子元素包含该文本，爱元素也被选择，返回一个数组

let div\_contain\_box=$('div:contains(box)');

console.log(div\_contain\_box);

//匹配所有不包含子元素或者文本的空元素

let empty=$(':empty');

console.log(empty);

//匹配含有选择器所匹配的元素的元素,返回一个数组

let div\_has\_a=$('div:has(a)');

console.log(div\_has\_a);

//匹配所有含有子元素或者文本元素的元素，返回一个数组

let div\_parent=$('div:parent');

console.log(div\_parent);

//匹配所有的不可见元素或者type为hidden的元素

let div\_hidden=$('div:hidden');

console.log(div\_hidden);

//匹配所有的可见元素

let div\_visible=$('div:visible');

console.log(div\_visible);

});

</script>

</body>

</html>

=================== [attribute] =================

[attribute] 匹配包含给定属性的元素

$("span[id]") id不要加引号

[attribute=value]

匹配给定的属性是某个特定值的元素

$("input[type=text]") 或者属性值加引号$("input[type='text']")

[attribute!=value]

匹配所有不含有指定的属性和有指定属性但属性值不等于特定值的元素

$("div[id!='hello']") 匹配所有不包含id属性且有该属性但值不等于hello的div

$("div[id!='hellow']")

[attribute^=value]

匹配给定的属性是以某些值开始的元素

$("div[id^='he']") 匹配到id=hello的div

[attribute$=value]

匹配给定的属性是以某些值结尾的元素

[attribute\*=value]

匹配给定的属性是以包含某些值的元素

$("div[id\*='ll']")

[selector1][selector2][selectorN]

复合属性选择器，需要同时满足多个条件时使用。

$("div[id\*='ll'][class]")

代码示例：

<html>

<head>

<meta charset="utf-8">

<title>选择器</title>

<script type="text/javascript" src="../jquery-3.2.1.min.js"></script>

<style>

</style>

</head>

<body>

<div id="box">#box

<span id="span-id1">

span with span-id1

</span>

<span id="span-id2">

span with span-id2

</span>

</div>

<div class="mybox">.mybox</div>

<ul>

<li class="list">1</li>

<li class="list">2</li>

<li class="list">3</li>

<li class="list">4</li>

</ul>

<a href="file" class="pa">file</a>

<div>

<p class="p1">p1</p>

<p class="p2">p2</p>

<p class="p3">p3</p>

<span class="p-span"></span>

</div>

<a href="file" class="a1">file</a>

<a href="#">#</a>

<a href="http">http</a>

<script>

$(function(){

//alert(1);

//匹配包含给定属性的元素,返回一个数组

let box\_id=$('div[id]');

console.log(box\_id[0]);

let span\_ids=$('span[id]');

console.log(span\_ids);

console.log(span\_ids.length);//2

console.log($('span[id]:eq(0)')[0]);

console.log($('span[id]:last')[0]);

//匹配给定的属性是某个特定的值的元素

let li\_list=$('li[class=list]');

console.log(li\_list.length);//4

console.log(li\_list[1]);

//匹配所有不包含指定的属性和有指定属性但是属性值不等于特定值的元素

let li\_not\_cc=$('li[class!=cc]');

console.log(li\_not\_cc);

//匹配给定的属性是以某些值开头的元素

let start\_p=$('[class^=p]');

console.log(start\_p);

let p\_start\_p=$('p[class^=p]:lt(2)');

console.log(p\_start\_p);

//匹配所有给定属性是以某些值结尾的元素

let end\_1=$('[class$=1]');

console.log(end\_1);

//匹配给定的属性包含某些值的元素

let has\_a=$('[class\*=a]');

console.log(has\_a);

//符合属性选择器，需要多个属性条件同时成立

let multi\_attr=$('a[class\*=a][href^=file]');

console.log(multi\_attr[0]);

});

</script>

</body>

</html>

=================== 子元素 =================

:nth-child

匹配作为子元素的第N个子或奇偶元素：即该元素首先是某个父元素的子元素，然后还要满足某个索引条件

:eq(index)只匹配一个元素（尽管如此也返回一个包含一个元素的数组），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(n)

:nth-child(3n+1)

:nth-child(3n+2)

$("div dd:nth-child(2n+1)")

$("div dd:nth-child(even)")

$("div dd:nth-child(3n+2)")

$("div dd:nth-child(5n+2)")

:first-child

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第一个子元素

$("ul li:first-child")匹配每一ul下的第一个li

:last-child

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

$("ul li:last-child") 匹配到每一个ul下的最后一个子元素

:only-child

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。

$("div p:only-child")

示例：

<html>

<head>

<meta charset="utf-8">

<title>选择器</title>

<script type="text/javascript" src="../jquery-3.2.1.min.js"></script>

<style>

</style>

</head>

<body>

<div id='box'>

<div class="header">

<div class="bg">

<h1>title</h1>

<div class="nav">

<ul>

<li class="tab">nav1</li>

<li class="tab">nav2</li>

<li class="tab">nav3</li>

<li class="tab">nav4</li>

<li class="tab">nav5</li>

</ul>

</div>

</div>

</div>

<div class="container">

<div class="sidebar">

<div class="sidebar-top">

<span>top</span>

</div>

<div class="sidebar-middle">

<span>middle</span>

</div>

<div class="sidebar-bottom">

<span>bottom</span>

</div>

</div>

<div class="content">content</div>

</div>

<div class="footer">

<div class="footer-ad">footer-ad</div>

<div class="footer-copyright">footer-copyright</div>

<div class="footer-contact">footer-contact</div>

</div>

</div>

<script>

$(function(){

//alert(1);

//匹配作为子元素的第N个子或奇偶元素，nth-child从1开始起

let div\_child\_1=$('div:nth-child(1)');

console.log(div\_child\_1);

console.log(div\_child\_1[1]);

let li\_child\_even=$('li:nth-child(even)');

console.log(li\_child\_even);

console.log(li\_child\_even[1]);

let li\_child\_odd=$('li:nth-child(odd)');

console.log(li\_child\_odd);

console.log(li\_child\_odd[0]);

let li\_child\_1=$('li:nth-child(3n+1)');

console.log(li\_child\_1);

console.log(li\_child\_1[1]);

//first-child 等价于nth-child(1)

let div\_first\_child=$('div:first-child');

console.log(div\_first\_child);

//last-child 等价于nth-child(1)

let div\_last\_child=$('div:last-child');

console.log(div\_last\_child);

//匹配唯一子元素

let div\_only\_child=$('div:only-child');

console.log(div\_only\_child);

console.log(div\_only\_child[0]);

let span\_only\_child=$('span:only-child');

console.log(span\_only\_child);

console.log(span\_only\_child[0]);

});

</script>

</body>

</html>

=========== form - input =================

:input

匹配所有 input, textarea, select 和 button 元素

$("span :input") 这个的冒号和前边一定要有空格

之所有空格是因为，jquery使用:input来代替了input、textarea、select和button等表单元素，将:input当做标签来使用就可以。

:text

匹配所有的单行文本框即input，且type为text

:password

匹配所有密码框，即匹配input且type为password

:radio

匹配所有单选按钮，即匹配input且type为radio

:checkbox

匹配所有复选框，即匹配input且type为checkbox

:submit

匹配所有提交按钮，即匹配input且type为submit和button标签元素

:image

匹配所有图像域，指的是input且type为image的而非img标签

:reset

匹配所有重置按钮，即匹配input且type为reset

:button

匹配所有按钮，即匹配input且type为button

:file

匹配所有文件域

:enabled

匹配所有可用元素，即匹配input且disabled属性设置为enabled

:disabled

匹配所有不可用元素，即匹配input且disabled属性设置为disabled

:checked

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

:selected

匹配所有select中被选中的option元素

示例代码：

<html>  
 <head>  
 <meta charset="utf-8">  
 <title>表单选择器</title>  
 <script type="text/javascript" src="../jquery-3.2.1.min.js"></script>  
 <style>  
  
 </style>  
 </head>  
 <body>  
 <div id="box">  
 <form action="#">  
 <label for="input-user">UserName:</label>  
 <input id="input-user" type="text" value="">  
 <label for="input-password">Password</label>  
 <input id="input-password" type="password" value="">  
 <label for="male">男</label>  
 <input type="radio" name="sex" id="male">  
 <label for="female">女</label>  
 <input type="radio" name="sex" id="female">  
 <span>  
 <input type="text" value="inner-span"/>  
 </span>  
 <label for="Math">Math</label>  
 <input type="checkbox" value="Math" id="Math" name="">  
 <label for="English">English</label>  
 <input type="checkbox" value="English" id="English" name="">  
 <label for="Chinese">Chinese</label>  
 <input type="checkbox" value="Chinese" id="Chinese" name="">  
 <textarea name="intro" id="intro" cols="30" rows="10"></textarea>  
 <select name="job" id="">  
 <option value="a">ajob</option>  
 <option value="b">bjob</option>  
 </select>  
 <input type="image" id="image-1">  
 <input type="image" id="image-2">  
 <input type="reset" value="reset">  
 <input type="file" name="file">  
 <input type="text" disabled="disabled" value="disabled-input1">  
 <input type="text" disabled="disabled" value="disabled-input2">  
 <button type="submit">submit</button>  
 </form>  
 <div>  
 <ul>  
 <li><img src="#" alt="img1"></li>  
 <li><img src="#" alt="img2"></li>  
 <li><img src="#" alt="img3"></li>  
 <li><img src="#" alt="img4"></li>  
 </ul>  
 </div>  
 </div>  
 <script type="text/javascript">  
 $(function(){  
 //alert(1);  
 //匹配所有的表单元素  
 let form\_elems=$(':input');  
 console.log(form\_elems.length);  
 console.log(form\_elems[0]);//input  
  
 let input\_in\_span=$('span :input');  
 console.log(input\_in\_span[0]);  
  
 //匹配所有的单行文本框,即input的类型为text的  
 let text=$(':text');  
 console.log(text[1]);  
  
 //匹配所有的密码框  
 let password=$('#box :password');  
 console.log(password)  
  
 //匹配所有的button，即input且type为button的和button标签元素  
 let buttons=$('#box :button');  
 console.log(buttons[0]);  
  
 //匹配所有的submit，仅仅匹配type为submit的表单元素  
 let submits=$(':submit');  
 console.log(submits[0]);  
  
 //匹配所有的radio元素  
 let radios=$(':radio');  
 console.log(radios[0]);  
  
 //匹配所有的checkbox  
 let checkbox=$('#box :checkbox');  
 console.log(checkbox);  
  
 //匹配所有的图像域，指的是input且type为image的而非img标签  
 let images=$(':image');  
 console.log(images[0]);  
  
 //匹配所有的reset表单域  
 let resets=$(':reset');  
 console.log(resets[0]);  
  
 //匹配所有的file表单域  
 let file=$(':file');  
 console.log(file[0]);  
  
 //匹配所有enabled的input的元素  
 let enableds=$(':enabled');  
 console.log(enableds);  
  
 //匹配所有disabled的input元素  
 let disableds=$(':disabled');  
 console.log(disableds);  
  
 //匹配所有的checked元素  
 let checkeds=$(':checked');  
 console.log(checkeds[0]);  
  
 //匹配所有被选中的option元素  
 let options=$(':selected');  
 console.log(options[0]);  
 });  
 </script>  
 </body>  
</html>

# #属性

重要：\*\*\*这些访问属性或者设置属性的方法都是jquery对象的，一定要确定调用者为jquery对象，如果是js的dom对象可以将它包含进$来转换为jquery对象。

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

设置或返回和设置被选元素的属性值。

$('img').attr('src')返回img的src属性，默认仅返回第一个值但选中了所有，

但是如果为元素设置属性值的话会为所有的匹配项设置属性。

如果是设置样式的话，如height、widht等时也会对所有的匹配项生效。

综上，无论是设置属性还是设置样式都会对所有的匹配项生效，但是当获取属性值的时候只会但会匹配项的第一个元素的属性值。

$("img").attr({'src':"test1.jpg",'title':'test'});//出入一个对象以为多个属性设置值

$("img").attr("src","test2.jpg");

$("img").attr("title", function() { return this.src });//某些属性能够使用this引用，应该是js规定的

$("img").attr("title",function(){

return $(this).attr("src");//某些属性不能使用this引用，必须使用$将this包含进来变为一个jquery对象然后访问和设置属性

});

removeAttr(name)//从每一个匹配的元素中删除一个属性

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

Prop相当于简化了操作，它既获取匹配集合的第一个元素又访问或者设置元素的属性值。

prop(name|properties|key,value|fn)//获取在匹配的元素集合中的第一个元素的属性值。

$('img').prop('src')

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

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

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

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

$("input[type='checkbox']").prop("checked", function( i, val ) {

return 1;

});

不考虑使用函数，下面设置属性值的方式是等价的

$(checkboxes[1]).prop('disabled',false);//相当于是removeAttr  
$(checkboxes[1]).removeAttr('disabled');//这句与上句功能上是等价的  
$(checkboxes[1]).attr('disabled',false);//这句有上面两句是等价的

removeProp(name)

用来删除由.prop()方法设置的属性集

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

示例代码：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>属性操作</title>  
 <script type="text/javascript" src="../jquery-3.2.1.min.js"></script>  
 <style>  
 #wrapper{  
 width:980px;  
 height:200px;  
 border:1px solid red;  
 }  
  
 ul{  
 border:1px solid blue;  
 }  
  
 ul li{  
 border:1px solid red;  
 }  
 </style>  
</head>  
<body>  
 <div id="box">  
 <div id="wrapper" name="bg-wrapper">  
 <img src="#" alt="背景图片" id="bg-img">  
 <div id="header">  
 <div class="nav">  
 <ul>  
 <li style="width:120px"><a href="#1">#1</a></li>  
 <li style="width:50px"><a href="#2">#2</a></li>  
 <li style="width:150px"><a href="#3">#3</a></li>  
 <li style="width:100px"><a href="#4">#4</a></li>  
 </ul>  
 </div>  
 </div>  
 </div>  
 <div id="container">  
 <img src="##" alt="container 背景图片">  
 <div class="sidebar"></div>  
 <div class="content">  
 <h2>表单</h2>  
 <form action="">  
 <input type="text" name="user"><br />  
 <input type="password" name="password"><br />  
 男:<input type="checkbox" name="male"><br />  
 女:<input type="checkbox" name="female">  
 </form>  
 </div>  
 </div>  
 <div id="footer"></div>  
 </div>  
 <script>  
 $(function(){  
// alert(1);  
 //获取元素的属性值,返回字符串类型,属性包括  
 //name/properties/key/value/fn  
 //而其他的如宽、高等为样式  
 let wrapper=$("#wrapper");  
 console.log(wrapper.attr("id"));  
 console.log(wrapper.attr("name"));  
  
 console.log(wrapper.width());//980  
 console.log(wrapper.height());//200  
  
 let bg\_img=$('#bg-img');  
  
 //获取属性  
 console.log(bg\_img.attr("src"));  
  
 //设置元素的属性  
 bg\_img.attr('src','http://shifone.qiniudn.com/images/top-img2\_708014e.jpg');  
  
 //如果有多个匹配项的话，这样只会得到第一个匹配元素的属性值，当然设置的话也只对第一个  
 //匹配的生效  
 $('li').attr('title','a list');  
  
 let li\_width=$('li').width();//设置样式的时候回对所有的匹配项生效  
 console.log(li\_width);  
 $('ul li').height('20px');  
  
 //使用函数设置属性  
 let ul=$('ul')[0];  
 console.log(ul);  
 $(ul).attr('alt','a ul');  
 $(ul).attr({a:'aa',b:'bb'});//传入一个对象以为多个属性设置值  
 $(ul).attr('title',function(){  
// console.log($(this).attr('alt'));  
 return $(this).attr('alt')+" used by title";  
 });  
  
 $('img').attr('src',function(){  
 console.log(this.src);  
 return this.src;  
 });  
  
 //移除所有匹配元素中的属性  
 $(ul).removeAttr("alt");  
  
 //指定获取第一个匹配元素的指定属性值  
 console.log($('ul li').prop("title"));//a list  
 console.log($('img').prop("src"));//a list  
  
 let checkboxes=$('input[checked!=checked][type=checkbox]');  
 $(checkboxes[0]).attr('disabled','disabled');  
  
 $(checkboxes[0]).removeAttr("disabled");//要移除属性而非设置属性为其他的值，这样才能生效  
  
 console.log($('img').prop('src'));//prop中的参数是必须的，而且  
 $(checkboxes[1]).prop('disabled','disabled');  
// $(checkboxes[1]).prop('disabled',false);//相当于是removeAttr  
// $(checkboxes[1]).removeAttr('disabled');//这句与上句功能上是等价的  
 $(checkboxes[1]).attr('disabled',false);//这句有上面两句是等价的  
 });  
 </script>  
</body>  
</html>

============== CSS类 =====================

这里的函数也是jquery对象的函数，需要使用jquery对象来调用这些函数；如果得到的是js的dom对象则可以使用$将它包含进来得到jquery对象。

addClass(class|fn)

为所有匹配的元素添加指定的类名,一个或多个类名时使用空格分隔。参数可以是类名的字符串，也可以是函数；当使用函数作为参数时，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。

$("ul li:last").addClass("c1 c2");

$("ul li:last").addClass(function(){

return "c3 c4";

});

removeClass([class|fn])；从所有匹配的元素中删除全部或者指定的类；当参数为空时表示移除所有的class。

$("ul li:last").removeClass("c3");

toggleClass(class|fn[,sw]) //如果存在就删除，如果不存在就添加，可以控制一个元素的类型的class有无切换。

class:要切换的CSS类名.

sw:用于决定元素是否包含class的布尔值,默认一次一切换

var count=0;

$("p").click(function(){

$(this).toggleClass("c1",count++%3==0);

});

$('div.foo').toggleClass(function() {

if ($(this).parent().is('.bar') {

return 'happy';

} else {

return 'sad';

}

});

示例代码：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>CSS demo</title>  
 <script type="text/javascript" src="../jquery-3.2.1.min.js"></script>  
 <style>  
 ul li{  
 list-style: none;  
 }  
 .fl{  
 float:left;  
 }  
  
 .fr{  
 float:right;  
 }  
  
 .li\_even{  
 width:20px;  
 height:30px;  
 border:3px solid red;  
 }  
  
 .li\_odd{  
 width:20px;  
 height:30px;  
 border:3px solid blue;  
 }  
  
 .margin-20{  
 margin:20px;  
 }  
  
 .btn-red{  
 background-color:red;  
 }  
  
 .btn-yellow{  
 background-color: yellow;  
 }  
 </style>  
</head>  
<body>  
 <div id="wrapper">  
 <div id="header">  
  
 </div>  
 <div id="container">  
 <div id="sidebar"></div>  
 <div id="content">  
 <ul>  
 <li>1</li>  
 <li>2</li>  
 <li>3</li>  
 <li>4</li>  
 <li>5</li>  
 </ul>  
 <button id="toggle">toggleClass</button>  
 </div>  
 </div>  
 <div id="footer"></div>  
 </div>  
 <script>  
 $(function(){  
 let li\_even=$('#wrapper li:even');  
 let li\_odd=$('#wrapper li:odd');  
 console.log(li\_even);  
 li\_even.addClass("li\_even");  
 li\_odd.addClass('li\_odd');  
 li\_even.addClass(function(){  
 return 'fl margin-20';  
 });  
 li\_odd.addClass(function(){  
 return 'fl margin-20';  
 });  
  
 li\_even.removeClass("li\_even");  
 // li\_odd.removeClass();//移除所有的class  
  
 //自动添加或者删除  
 $('#toggle').click(function(event){  
 // console.log(event);  
 $(this).toggleClass('btn-red');  
 });  
  
 $('#toggle').click(function(event){  
 // console.log(event);  
 $(this).toggleClass(function(){  
 return 'btn-yellow';  
 },true);//最后一个参数如果为true则只会一次变换，相当于移除了class  
 });  
 });  
 </script>  
</body>  
</html>

============ HTML代码/文本/值 ===============

## html([val|fn])

取得/设置第一个匹配元素的html内容（即内部的html部分）。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。

$("ul").html("<hr/><br/>hhahahh");改写里面的所有内容

$("ul").html(function(m){

return "这个ul元素的index是"+m;

});

## text([val|fn])

返回值:String

取得所有匹配元素的内容（即包含的文本节点的内容）。

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

alert($("p").text());//返回所匹配元素的文本组成的文本集合,不包含标签

$("p").text("<h2>haha</h2>");//设置所匹配元素的文本，包含的标签，它会定义标签进行转义，并是的标签类似文本一样显示在页面中

$("p").text(function(n){

return "这个 p 元素的 index 是：" + n;

});

## val([val|fn|arr])

获得匹配元素的当前值。元素应该具有value属性

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

$("input").val();//设定文本框的值

$("input").val("hello world!");//设定文本框的值

$('input:text.items').val(function() {

return this.value + ' ' + this.className;

});

示例代码：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>html value text ... demo</title>  
 <script type="text/javascript" src="../jquery-3.2.1.min.js"></script>  
</head>  
<body>  
 <div id="header">  
 <div class="header-content">  
 <a class="header-logo" href="http://xin.baidu.com"></a>  
 <div class="header-right"><div class="header-qr">  
 <ins></ins>  
 <img class="header-qr-img" src="//ts.bdimg.com/xin/pc/20170918/dist/common/css/img/qrcode.png">  
 <div class="header-qr-txt">扫一扫</div></div>  
 <span class="header-login-wrap">  
 <span class="header-login-icon">  
 </span><a class="header-login ZX\_LOG\_LINK" data-click="{&quot;cate&quot;: &quot;login&quot;, &quot;action&quot;: &quot;click&quot;}">登录</a>  
 </span>  
 </div>  
 </div>  
 </div>  
 <div id="container">  
 <div id="sidebar"></div>  
 <div id="content">  
 <div class="zx-detail-content">  
 <ul class="zx-detail-tab"><!-- 阿拉丁用到了webRecord，所以只能在这过滤 -->  
 <li class="zx-detail-tab-item ZX\_LOG\_TAB" data-type="basic" data-reload="0" data-click="{&quot;cate&quot;: &quot;basic&quot;, &quot;action&quot;: &quot;click&quot;}">  
 <span class="tab-item-name">基本信息</span>  
 <span class="tab-item-tag "></span>  
 </li><!-- 阿拉丁用到了webRecord，所以只能在这过滤 -->  
 <li class="zx-detail-tab-item ZX\_LOG\_TAB" data-type="changeRecord" data-reload="0" data-click="{&quot;cate&quot;: &quot;changeRecord&quot;, &quot;action&quot;: &quot;click&quot;}">  
 <span class="tab-item-name">变更记录</span>  
 <span class="tab-item-tag ">3</span>  
 </li>  
 </ul>  
 </div>  
 </div>  
 </div>  
 <div id="footer">  
 <input type="text" name="user" value="please enter your user name" />  
 <input type="password" name="password" />  
 </div>  
 <script>  
 $(function(){  
// alert(1);  
 let content=$('#content');  
 console.log(content.html());  
  
 //设置元素的内部  
 content.html('<h1>修改content的html内容</h1>');  
 content.html(function(){  
 return '<p><a href="">使用函数的返回值来更新content的html内容</a></p>';  
 });  
  
 console.log($('#content p').text());  
 console.log($('#header').text());  
 console.log($('#header').text('<a>is there anything?<a href="#">click</a></h6>'));//类似使用了html设置，但是  
 //text的设置会对标签进行转义，即页面会显示所有的标签  
  
 $("#sidebar").text(function(){  
 return '<h3>sidebar</h3>';  
 });  
  
 //获取元素的value值  
 console.log($('#footer :input'));  
 let userVal= $('#footer :input[type=text][name=user]');  
 console.log($(userVal).val());  
 $(userVal).val('my-username');  
 });  
 </script>  
</body>  
</html>

# #文档处理

append(content|fn)

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。

$(document.body).append("<span>hahah</span>");

$("div").append("<strong>强调</strong>");

$("div").append(function(){

return "hahaa";

});

appendTo(content)

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。

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

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

$("<p>这也行</p>").appendTo($("div:not('.div')"));

prepend(content)

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。

$("p").prepend("div");

$("div").prepend("<em>加载前边</em>");

prependTo(content)

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。

$("span").prependTo($(".dd"));

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

返回值:jQueryafter(content|fn)

在每个匹配的元素之后插入内容。

$("span").after("<i>加到span</i>");

$("span").after($("em"));

before(content|fn)

在每个匹配的元素之前插入内容。

insertAfter(content)

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。

insertBefore(content) 返回值:jQuery

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

$("p").wrap("<div class='c1'></div>");

$("p").wrap(document.getElementById("content"));//原来的content并没有变化，而是生成了新的id为content的包裹层

$('.inner').wrap(function() {

return '<div class="' + $(this).text() + '" />';

});

unwrap()

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。

$("p").unwrap();

？？？wrapAll(html|ele)

wrapInner(htm|ele|fnl) 返回值:jQuery

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

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

$("div").wrapInner("<b></b>");//包装div里的所有内容

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

$("b").wrapInner(function(){

return "<em></em>";

});

replaceWith(content|fn) 将所有匹配的元素替换成指定的HTML或DOM元素。

$("div").replaceWith("<b>替换成功</b>");//全部替换，包括里面的内容清空

$(".third").replaceWith($(".first"));//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。

replaceAll(selector) 用匹配的元素替换掉所有 selector匹配到的元素。与上面的方法效果相反

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

empty() 删除匹配的元素集合中所有的子节点。

$("div,p").empty();//清空p和div里的内容

remove([expr])

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。

$("p").remove();//上面是清空元素，这是删除

$("p").remove(".hello");//删除类为hello的p元素

detach([expr])

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。

clone([Even[,deepEven]])

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。

Events:一个布尔值（true 或者 false）指示事件处理函数是否会被复制

deepEvents: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

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

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。

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。

$("button").click(function(){

$(this).clone(true).insertAfter(this);

});

####筛选

#eq(index|-index)

获取第N个元素

index:一个整数，表示元素基于0的位置

-index:一个整数，表示元素的位置，从集合中的最后一个元素开始倒数,从1算起

$("p").eq(1).css("color","red");

$("p").eq(-2).css("color","red");

#first() 获取第一个元素

$("li").first().css("background","yellow");

#last() 获取最后一个元素

$("ul li").last().css("color","orange");

#hasClass(class) 返回值:Boolean

$("div").click(function(){

if ($(this).hasClass("protected")){

$(this)

.animate({ "left": "-10px" })

.animate({ "left": "10px" })

.animate({ "left": "-10px" })

.animate({ "left": "10px" })

.animate({ "left": 0 });

}

});

#filter(expr|obj|ele|fn) 返回值:jQuery

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

$("p").filter(".selected").css("background-color","orange");

$("p").filter(".selected, :first").css("background","#333333");//保留第一个或带有selected类的元素

$("p").filter(function(index) {

return $("ol", this).length == 0;

});//保留子元素中不包含ol的元素

$("p","ul li").filter(":first ").css("background","#dcd");//在ul li下选择p来匹配

#is(expr|obj|ele|fn) 返回值:Boolean

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。

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

$("ul li").click(function(){

if($(this).is(function(){ return $("strong",this).length==2})){

$(this).css("background","blue");

}else if($(this).is(function(){ return $("strong",this).length==1})){

$(this).css("background","red");

}

});

#map(callback) 返回值:jQuery

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。

$("p b").append($("input").map(function(){

return $(this).val();

}).get().join(","));

alert($("input").map(function(){

return $(this).val();

}).get());//得到数组

#has(expr|ele)

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。

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

$("p").has("span").css("background-color","#dcdcdc");

#not(expr|ele|fn) 返回值:jQuery

删除与指定表达式匹配的元素

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

#slice(start, [end]) 选取一个匹配的子集 与原来的slice方法类似

$("p").slice(0, 1).wrapInner("<b></b>");//选取第一个P元素

$("p").slice(-1).wrapInner("<b></b>");//选取最后一个P元素

$("p").slice(0, 2).wrapInner("<b></b>");//选取前两个P元素

$("p").slice(1, 2).wrapInner("<b></b>");//只选取第二个P元素

$("p").slice(1).wrapInner("<b></b>");//选取第二个P元素之后的所有p元素

$("p").slice(-1).wrapInner("<b></b>");//选取最后一个P元素

#children([expr])

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。

$("div").children()

$("div").children("span").css("color","blue");//仅接受表达式，jQuery对象不可以

$("div").children().css("color","yellow");

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

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。注意：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。

$("li:first").closest(["ul", "body"]);//传递数组字符串可以查找多个元素

#ind(expr|obj|ele)

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。

$("p").find("span");这等价于$("p span")

#next([expr])

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。

$("div").next().css("color","red");

$("div").next(".pp").css("color","red");//选择div紧跟后面的p元素且该元素带有pp类

#nextAll([expr]) 查找当前元素之后所有的同辈元素，可以使用表达式过滤

$("div").nextAll().css("color","red");

$("div").nextAll("span").css("background-color","red");

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

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。

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。

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

$("#term-2").nextUntil(document,getElementById("#term-3"),"dt").css("color","red");

#parent([expr]) 取得包含所有匹配元素的唯一父元素的元素集合

$("p").parent();

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

#parents([expr]) 取得一个包含着所有匹配元素的祖先元素的元素集合，不包括根元素

$("span").parents()

$("span").parents(".selected");

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

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

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。

$(".item-a").parentsUntil(".level-1");

#prev([expr])

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。

$("li").prev().css("background-color","#777");

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

#prevAll([expr]) 查找当前元素之前所有的同辈元素

$("p").prevAll().css("color","#663344");

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

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。

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

#siblings([expr])

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。

$("div").siblings()

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

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

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。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。

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

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

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

#andSelf() 加入先前所选的加入当前元素中

$("div").find("p").andSelf().addClass("border");//将div和内部的p都加上border类

#contents()

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

$("p").contents().not("[nodeType=1]").wrap("<b/>");//查找所有文本节点并加粗

$("iframe").contents().find("body")

.append("I'm in an iframe!");

#end()

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'。

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

$("p").find("span").end().css("background-color","#123433");

###CSS

#css(name|pro|[,val|fn]) 返回值:String

访问匹配元素的样式属性。也可以设置样式

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

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

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

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

获取匹配元素在当前视口的相对偏移;此方法只对可见元素有效。

coordinates 一个对象，用于设置元素新的位置，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

alert($("p.offset").offset().left);

alert($("p.offset").offset().top);

$("p,offset").offset({"top":123,"left":111});//设置为当前整个打开视口的位置（网页顶部左上角为0，0），这时加上position:relative属性并自动调节left 和top是的位置到达相对整个打开视口的位置

#position() 返回值:Object{top,left}

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。

alert($("p.offset").position().left);

#???/scrollTop([val]) 返回值:Integer

获取匹配元素相对滚动条顶部的偏移。

#???/scrollLeft([val]) 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。

#height([val|fn]) 返回值:Integer

取得匹配元素当前计算的高度值（px）,可以用来获取 window 和 document 的高

alert($(".height").height());//火狐中获得的是内容区的高度，即设置的width值

$(".height").height("500px");//设置高度为500px

$(".height").height("200px");

#width([val|fn])

取得第一个匹配元素当前计算的宽度值（px）。

#innerHeight()

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。

alert($(".height").innerHeight());

#innerWidth()

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。

#outerHeight([options]) 返回值:Integer

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。

alert($(".height").outerHeight());

options:Boolean 默认值:'false' 设置为 true 时，计算边距在内。

alert($(".height").outerHeight(true));

###效果

#show([speed,[easing],[fn]])

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。

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。

$("div").show();//默认600毫秒

$("div").show(4000);

$("div").show("slow","swing");

$("p").show("fast",function(){

$(this).text("Animation Done!");

});//动画执行完执行一次函数

$("div").show(4000,function(){

$(this).text("执行完动画");

});

$("div").show(4000,"linear",function(){

$(this).text("执行完动画");

});

#hide([speed,[easing],[fn]])

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。

参数解释同上

$(".show").hide(4000,"swing");

#toggle([speed],[easing],[fn])

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。

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"

switch:Boolean,用于确定显示/隐藏的开关。如：true - 显示元素，false - 隐藏元素

fn:在动画完成时执行的函数，每个元素执行一次。

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。

$(".toggle").toggle(4000);相当于hide或show 4000毫秒

#slideDown([speed],[easing],[fn])

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。

$(".hide").slideDown();

$("p").slideDown("slow");

$(".hide").slideDown(1000,function(){

$(this).text("动画执行完");

});

#slideUp([speed,[easing],[fn]])

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。

$(".show").slideUp(1000,function(){

$(this).text("动画执行完");

});

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

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。

$(".hide").slideToggle(1000,function(){

$(this).text("动画执行完");

});//相当于slideDown

$(".show").slideToggle(1000,function(){

$(this).text("动画执行完");

});//相当于slideUp

#fadeIn([speed],[easing],[fn])

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。

$(".hide").fadeIn(1000,function(){

$(this).text("动画执行完");

});

#fadeOut([speed],[easing],[fn])

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。

$(".show").fadeOut(1000,function(){

$(this).text("动画执行完");

});

#fadeTo([[speed],opacity,[easing],[fn]])

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。

opacity:一个0至1之间表示透明度的数字。

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。

$(".hide").fadeTo(4000,"0.3",function(){

$(this).text("动画执行完");

});

#fadeToggle([speed,[easing],[fn]])

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。

$(".hide").fadeToggle(4000,function(){

$(this).text("动画执行完");

});

##animate(params,[speed],[easing],[fn])

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。

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

$("#go").click(function(){

$("#block").animate({

width: "90%",

height: "100%",

fontSize: "10em",

borderWidth: 10

}, 1000 );

});

$("#right").click(function(){

$(".block").animate({left: '+50px'}, "slow");

});//以原始位置为参考

$("#left").click(function(){

$(".block").animate({left: '-50px'}, "slow");

});//以原始位置为参考

#stop([clearQueue],[jumpToEnd])

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

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。

$("#stop").click(function(){

$("#box").stop();

});

$("#go").click(function(){

$(".block").animate({left: '+200px'}, 2000);

});

// 当点击按钮后停止动画

$("#stop").click(function(){

$(".block").stop();

});

#delay(duration,[queueName])

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。

duration:延时时间，单位：毫秒

queueName:队列名词，默认是Fx，动画队列。

$('#foo').slideUp(300).delay(800).fadeIn(400);

#jQuery.fx.off 返回值:Boolean

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。

#jQuery.fx.interval 返回值:Number

设置动画的显示帧速。

jQuery.fx.interval = 100;//数值越小越流畅

###事件

#ready(fn) 返回值:jQuery

当DOM载入就绪可以查询及操纵时绑定一个要执行的函数。

$(document).ready(function(){ })等价于$(function(){ })

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

在选择元素上绑定一个或多个事件的事件处理函数。

data:当一个事件被触发时要传递event.data给事件处理函数。

$("p").on("click",function(){

alert($(this).text());

});

$("p").on("click",{"foo":"haha"},myfunction);

function myfunction(event){

alert(event.data.foo);

}

$("form").on("submit", false);//取消form的submit提交效果

$("form").on("submit",function(event){

event.preventDefault();//也可以达到取消默认操作的效果

});

$("form").on("submit", function(event) {

event.stopPropagation();//禁止冒泡

});

/\*

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

$(".div1").on("click",function(){

alert("外层div");

});

$(".div2").on("click",function(){

alert("内层div");

});

$(".clk").on("click",function(event){

alert("点击我");

event.stopPropagation();

});

\*/

#bind(type,[data],fn)

为每个匹配元素的特定事件绑定事件处理函数。

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。

$("p").bind("click", function(){

alert( $(this).text() );

});

$("button").bind({

click:function(){$("p").slideToggle();},

mouseover:function(){$("body").css("background-color","red");},

mouseout:function(){$("body").css("background-color","#FFFFFF");}

});

function handler(event) {

alert(event.data.foo);

}

$("p").bind("click", {foo: "bar"}, handler)

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

$("form").bind("submit", function(event){

event.preventDefault();

});

$("form").bind("submit", function(event){

event.stopPropagation();

});

$("p").bind({

"mouseover":function(){

$(this).css({"font-size":"30px","color":"red"});

},

"mouseout":function(){

$(this).css({

"font-size":"14px",

"color":"black"

});

}

});

#one(type,[data],fn)

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。

$("button").one({

click:function(){$("p").slideToggle();},

mouseover:function(){$("body").css("background-color","red");},

mouseout:function(){$("body").css("background-color","#FFFFFF");}

});//仅执行一次就失效

#unbind(type,[data|fn]])

bind()的反向操作，从每一个匹配的元素中删除绑定的事件。

$("p").unbind("mouseout");//取消p的mouseout事件

$("p").unbind();//取消P的所有绑定

#live(type, [data], fn)

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。

.live() 再点击新增的元素，他依然能够触发事件处理函数。

$("a").live("click", function() { return false; });//阻止默认事件行为和事件冒泡

$("a").live("click", function(event){

event.preventDefault();

});//仅仅阻止默认事件行为

#die(type, [fn])

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。

function aClick() {

$("div").show().fadeOut("slow");

}

$("#unbind").click(function () {

$("#theone").die("click", aClick)

});

#delegate(selector,[type],[data],fn)

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。

$(".div1").delegate("p","mouseout",function(){

alert();

});

#undelegate([sel,[type],fn])

删除由 delegate() 方法添加的一个或多个事件处理程序。

#hover([over,]out)

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

$("p").hover(

function(){

$(this).css({"font-size":"30px","color":"red"});

},

function(){

$(this).css({

"font-size":"14px",

"color":"black"

});

}

);

#toggle([speed],[easing],[fn])

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。

$("button").click(function(){

$("p").toggle("slow","swing",

function(){

$(this).css({"font-size":"30px","color":"red"});

},

function(){

$(this).css({

"font-size":"14px",

"color":"black"

});

}

);

#blur([[data],fn])

触发每一个匹配元素的blur事件

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

$("input").blur(function(){

alert("hello world");

});

#change([[data],fn])

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事件会在元素获得焦点后改变时触发。

#click([[data],fn])

触发每一个匹配元素的click事件。

#dblclick([[data],fn])

#error([[data],fn])

触发每一个匹配元素的error事件。

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。

$("img").error(function(){

$(this).hide();//隐藏出错的图像

});

#focus([[data],fn])

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。

$("input").focus(function(){

alert("hello world");

});

$("input").focus(function(){

$(this).blur();//使人无法使用文本框

});

$("input").focus();//页面加载自动获得焦点

#focusin([data],fn)

在每一个匹配元素的focusin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获得焦点的时候会触发这个事件。这跟focus事件区别在于，他可以在父元素上检测子元素获取焦点的情况。当子元素获得焦点时父元素的focusin事件会被触发

$("input").focusin(function(){

$("p").find("span").fadeOut(1000);

});

#focusout([data],fn)

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。

$("div").focusout(function(){

alert();

});

#keydown([[data],fn])

触发每一个匹配元素的keydown事件,keydown事件会在键盘按下时触发。

$("input").keydown(function(){

alert();

});

#keypress([[data],fn])

触发每一个匹配元素的keypress事件

#keyup([[data],fn])

触发每一个匹配元素的keyup事件

$("input").keydown(function(){

$("p").css("color","yellow");

}).keyup(function(){

$("p").css("color","#333");

});

#mousedown([[data],fn])

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

#mouseenter([[data],fn])

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。

$(".div1").mouseenter(function(){

$("p").css("background-color","red");

});

#mouseleave([[data],fn])

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。

$(".div1").mouseleave(function(){

$("p").css("background-color","red");

});//只有在鼠标指针离开元素时才会触发，仅仅离开子元素而未离开所选元素则不会触发

#mousemove([[data],fn])

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

#mouseover([[data],fn])

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

#mouseup([[data],fn])

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

#resize([[data],fn])

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

$(window).resize(function(){

alert("禁止改变窗口大小");

});

#scroll([[data],fn])

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

#select([[data],fn])

触发每一个匹配元素的select事件

#submit([[data],fn])

触发每一个匹配元素的submit事件。

$("form").submit( function () {

return false;//禁止表单提交

} );

#unload([[data],fn])

在每一个匹配元素的unload事件中绑定一个处理函数。

$(window).unload( function () { alert("Bye now!"); } );//页面卸载时弹出警告框

///////////不懂////////////////////

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

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。

trigger(type,[data])

triggerHandler(type, [data])

命名空间

# #Ajax

## ajax 请求

### jQuery.ajax(url,[settings])

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(me：此时返回的为原生的XMLHttpRequest对象，类似使用js得到该对象)。

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（success等）。

#### 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](http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/)。

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&lt;script&gt;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&amp;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&lt;script&gt;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。

//========================20171011，使用格式化代码的部分完成