### 基础特性

1. 可见性

:visible

* display:none
* 元素占据文档中一定的空间，被认为是可见的。（审查元素HTML页面里显示的都为可见）
* 元素的visibility: hidden 或 opacity: 0被认为是可见的，因为他们仍然占用空间布局。
* 所有 option 元素是被认为是隐藏的， 不管他们是否selected（选中）状态.
* 如果一个元素只要有任何布局盒，即使宽度和/或高度为0，那么它将被视为:visible。

:hidden

与visible相反

1. 子元素

注意：:前的元素的父元素下的所有子元素中选择

先找它的父元素，再找每个父元素下的某个子元素

|  |
| --- |
| $("ul li:first-child") //ul父元素下所有的子元素li中的第一个  结果：<li>John</li>，<li>Glen</li>  $("ul li:first) //:first只能选择一个，即把ul下的所有li放一起，找第一个 |

* :first-child

虽然:first只匹配一个单独的元素，但是:first-child选择器可以匹配多个：即为每个父级元素匹配第一个子元素。这相当于:nth-child(1)。

* first-of-type

匹配冒号前面元素的同类元素的第一个

:first :first-child :first-of-child的区别

|  |
| --- |
| <body>  <button>:first</button>  <button>:first-child</button>  <button>:first-of-type</button><br><br>  <p>p1</p>  <div style="border:1px solid;">  <p>p2</p>  <p>p3</p>  </div><br>  <div style="border:1px solid;">  <span>span1</span>  <p>p4</p>  <p>p5</p>  <span>span2</span>  </div><br>  <div style="border:1px solid">  <p>p6</p>  <p>p7</p>  </div>    <p>p8</p>  </body> |

$(“p:first”)

先找p标签的集合，然后再这个集合里找第一个p标签

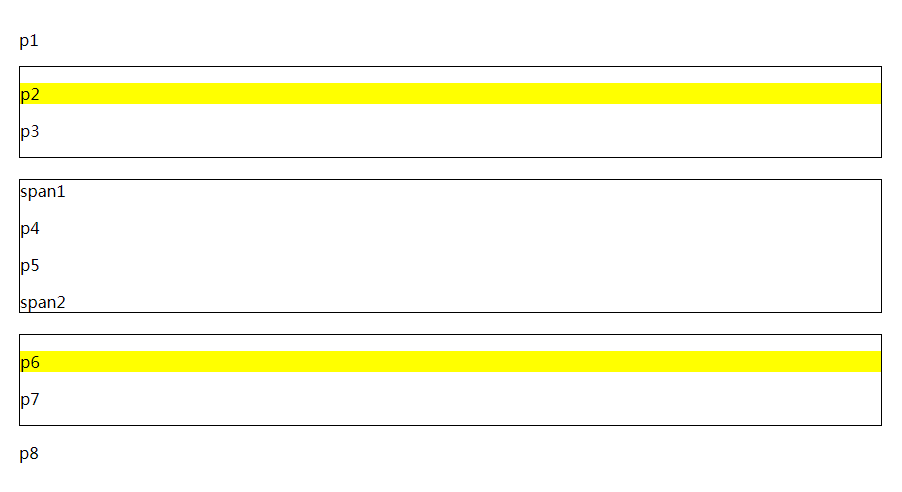
结果：

$(“p:first-child”)

先找所有p标签的所有父标签，再在每个父标签下找各自的第一个子元素，并且这个第一个子元素必须是p标签，不是p标签也匹配不到。

所以，p1不是body父元素的第一个子元素，匹配不到，p4不是第二个div里面的第一个子元素，匹配不到；

结果：



$(“p:first-of-type”)

先找所有p标签的所有父标签，然后在每个父标签下找p标签中的第一个，他并不一定是子元素中的第一个，而是p标签中的第一个。

结果：

* :nth-child(n)

选择的他们所有父元素的第n个子元素。

比如：a:nth-child(1),先找a标签的父元素，若父元素的第一个子元素是a标签的话，匹配，若不是，则匹配不到。

n可以是index索引，从1开始；也可以是even或odd，或方程式4n；

* :nth-of-type(n)

与first-of-type同理，例如： a:nth-of-type(1)先找a标签的父元素，在父元素里找所有a标签集合中的第1个

1. 表单

* :input

选择所有 input, textarea, select 和 button 元素.

1. 表单对象属性

:enabled

:disabled

:checked

:selected

1. 内容

:contains 匹配包含指定文本的元素（文本可以在后代中）

:has(selector) 后代匹配selector的父元素

:empty 匹配没有子元素的元素（子元素包括文本节点）

:parent匹配含有子元素的元素（子元素包括文本节点）

1. 属性

* .attr(attrName) 获取匹配的元素集合中的第一个元素的属性的值。

.attr(attr,value) 为每一个元素设置属性值

1）.attr( attributeName, value )

.attr(‘id’,’act’)

2）.attr( attributes ) //传对象

.attr({alt:”xxx”,title:”xxx”})

3）.attr( attributeName, function )

|  |
| --- |
| $('#greatphoto').attr('title', function(i, oldVal) { //i代表索引，oldVal代表原来的属性值  return val + ' - photo by Kelly Clark'  }); |

.removeAttr(attrName) 为匹配到的每一个元素移除一个属性

* .prop(propName) 获取匹配元素集中第一个元素的property值
* .prop(propName,value)
* .prop({})
* .prop( propertyName, function(index, oldPropertyValue) )

注意：Properties 属性一般影响 DOM 元素的动态状态并不会改变序列化的 HTML attribute 属性。所以input的disabled和checked属性，应该使用prop()而不是使用attr();

但是在3.xx的jquery，使用attr()也能达到效果，只是attr()改变的是html的checked属性。

html的属性值，相当于初始值。

HTML标签上设置的属性就是Attribute, 而直接在JavaScript中用点运算符操作的DOM对象属性就是Property; Attribute的数据类型永远都是字符串，而Property就可以非常丰富。

1. HTML代码/文本/值

html() 获取（第一个）/设置（所有匹配）的html

text() 获取（第一个）/设置（所有匹配）的文本

val() 获取input select textarea的值

1. css()

通过.css()方法处理的是内联样式，直接通过元素的style属性附加到元素上的

addClass()方法是通过增加class名的方式，那么这个样式是在外部文件或者内部样式中先定义好的，等到需要的时候在附加到元素上

所以，css()的优先级大于addClass()

1. 内部插入

选择器.append(追加的内容[,more]) 在匹配到的每个元素的内部追加

注意：追加的内容可以是新创建的，也可以是页面上已存在的；若是选择页面已存在的，是移动而不是复制；

1. 外部插入

选择器.after(插入的内容) 在每个匹配到的元素后面插入，作为其兄弟元素

插入的内容.insertAfter(选择器) 在每个匹配到的元素后面插入，作为其兄弟元素

1. 删除节点

empty() 移除匹配元素的所有子节点，他本身并没有被移除；它移除了元素身上的所有事件，不可恢复；不能传参数

remove([selector]) 除了移除所有子节点外，还将自身移除，并移除所有绑定的事件。

detach([selector]) 删除节点（包括自身），但保留事件

1. 克隆

clone()

clone(true) 连事件一起克隆，默认false

1. 替换

oldContent.replaceWith(newContent) 使用新内容替换匹配内容

注意：用来替换的内容若是页面上原有的元素，它是被移动了，并非复制。

会删除与节点相关联的所有数据和事件

返回被替换掉的元素的集合

newContent.replaceAll(oldContent) 目标和源相反

1. 串联

contents() 获得匹配元素集合中每个元素的子元素，包括文字和注释节点。

等同于js里的childNodes

children() 只获取子元素，不含文本节点及注释节点，只是子代，不含后代

1. 包裹

wrap(wrappingElement) 用wrappingElement包裹匹配到的每一个元素（一个个单独包裹）

wrapAll( wrappingElement ) 用wrappingElement包裹匹配到的元素的集合，把匹配的集合凑一块，用一个东西包裹。匹配元素间的任何元素都被排除在外。

wrapInner(wrappingElement) 在匹配的元素的内容外部包一层结构，即在每个匹配元素的innerHTML的外面包一层。

1. 查找

children（）只获取子元素，不含文本节点及注释节点，只是子代，不含后代

find() 在后代中查找，不包括自己

parent() 查找父元素，只会向上查一级

parents()查找所有的祖辈元素的集合,从元素上一级开始

closest() 从元素本身开始，在DOM 树上逐级向上级元素匹配，并返回最先匹配的祖先元素。

|  |
| --- |
| parents()和closest()区别：  1.起始位置不同：.closest开始于当前元素 .parents开始于父元素  2.遍历的目标不同：.closest要找到指定的目标，.parents遍历到文档根元素，closest向上查找，直到找到一个匹配的就停止查找，parents一直查找到根元素，并将匹配的元素加入集合  3.结果不同：.closest返回的是包含零个或一个元素的jquery对象，parents返回的是包含零个或一个或多个元素的jquery对象 |

next() 匹配元素集合中每个元素后面紧邻的同辈元素的一个集合

prev() 匹配元素集合中每个元素前面紧邻的同辈元素的一个集合

siblings(）匹配元素集合中每个元素的所有兄弟元素，不包括元素本身。

add()向已有选择器中添加新元素

1. 事件

mouseover 鼠标进入元素就要触发 ，存在事件冒泡，会在进入他的子元素也触发

mouseout 鼠标离开元素就要触发 ，存在事件冒泡，会在进入他的子元素也触发

mouseenter鼠标进入时触发，只在绑定该事件的元素上触发，不存在冒泡

mouseleave 鼠标离开时触发，不存在冒泡

mousemove 鼠标在元素内部移动时就触发

hover(fun1,fun2)当鼠标进入和离开时触发

focusin() 在元素（或者其内部的任何元素）获得焦点时触发 支持冒泡

focusout()在元素（或者其内部的任何元素）失去焦点时触发 支持冒泡

focus() 在元素获得焦点时触发，不支持冒泡

blur() 在元素失去焦点时触发，不支持冒泡

关于冒泡的理解：

|  |
| --- |
| <div class="aaron">  点击触发焦点(无反应)：  <input type="text" />  </div>    //没反应  $(‘.aaron’).focus(function() {  $(this).css('border', '2px solid red')  })  //有反应  $(‘.aaron’).focusin(function() {  $(this).css('border', '2px solid red')  })  解析：事件时绑定在div上的，鼠标只能让它的子元素input获得焦点，当input获得焦点时，触发了获得焦点事件（focusin），但input上没有绑focusin事件，但是focusin支持冒泡，会向上找他的父元素，如父元素绑定的有focusin事件，就会触发。所以向上找到div，并触发div的focusin事件；  而focus并不支持冒泡，所以input的聚焦事件，并不能冒泡到div上，所以focus事件没反应。 |

所以，若一个元素内部有input时，要用focusin和focusout, 要是直接作用到input上时，用focus和blur

change() 一个元素的值改变时触发，此事件只用于<input>元素、<textarea>和<select> ,支持冒泡（change可以绑到以上三种类型的父元素上）；

注：使用js改变输入值不会触发change()事件

select() 用户在一个元素中进行文本选择时触发。

keydown()和keyup() :keydown获取不到当前输入的内容，因为keydown是在文字还没敲进文本框时就触发的，所以此时获取到的文本是之前的，而keyup是在整个键盘事件完成后触发，可以获取此次输入的文本。

keypress：

* 只能捕获单个字符，不能捕获组合键
* 无法响应系统功能键（如delete，backspace）
* 不区分小键盘和主键盘的数字字符
* 主要接收字母、数字
* 而keyup和keydown可以接收不被keypress识别的字符

1. trigger和triggerHandler
   1. triggerHandler不会触发浏览器的默认行为,只会触发绑定了该事件的处理函数
   2. .trigger() 会影响所有与 jQuery 对象相匹配的元素，而 .triggerHandler() 仅影响第一个匹配到的元素。
   3. 使用 .triggerHandler() 触发的事件，并不会在 DOM 树中向上冒泡。 如果它们不是由目标元素直接触发的，那么它就不会进行任何处理
2. hide 实际是将元素的css的display设为none，当隐藏之后，元素会从页面布局中移除。

show

toggle()

1. toggle() 无参数，不带动画，若元素最初是显示状态，则隐藏，若最初是隐藏装填，则显示
2. .toggle( [duration ] [, complete ] ) 若最初元素时隐藏的，则会动态显示，若最初元素是显示的，则会动态隐藏
3. .toggle(display) 直接提供一个参数true或false，指定要改变的元素的最终效果

用法：

鼠标点击，可以切换div的显示和隐藏。

|  |
| --- |
| $('button').click(function () {  $('div').toggle(2000);  }) |

1. slideDown slideUp slideToggle 下拉，上翻（上下动作），两者切换，用法跟toggle类似

注意：若有多个匹配的元素一起执行，回调函数会在每个元素执行完都执行一遍，而不是所有元素执行完再执行。

1. fadeIn（speed,fn） 淡入

fadeOut（speed,fn） 淡出

fadeTo（speed,opacity,fn） 可以控制淡入或淡出到某个透明度,opacity是0~1的值。

fadeToggle（speed,fn）元素显示时，是淡出；元素隐藏时，是淡入

1. animate(properties, [, duration ] [, easing ] [, complete ])

properties是css属性和值的对象，能使用的属性，其值必须是数字；

或者使用‘show’ ‘hide’ ‘toggle’快捷显示和隐藏。

也可以使用+=5px作为值

每个属性的值表示这个样式属性到多少时动画结束

1. stop()

stop() 无参数，当前动画立即停止，直接跳到后面队列里的下个动画

stop(true) 当前动画立即停止，后面队列里的动画也都不执行了

stop(true,true)直接跳到当前动画的最终形态，后面队列的都不执行了

stop(‘字符串’[,][,])若第一个参数是字符串，则只停止该参数代表的动画

1. $.each(arr,callback) 遍历数组

$.each(obj,callback) 遍历对象

callback:function(index,value){}或function(key,value){}

在循环里面，如果return false，则跳出循环，相当于break；若返回其他非false值，则相当于continue，跳到下个循环。

1. $(selector).each(function(index, Element)) 专门用于遍历一个jQuery对象

index指索引，Element指当前的DOM元素，等同于this

1. $.inArray(value,array[,fromIndex]) 检查array中有没有value值，有就返回索引，没有返回-1
2. $.trim(str) 去掉开始和结束处的所有换行符，空格(包括连续的空格)和制表符（tab）
3. get([index]) 通过jQuery对象获取一个对应的DOM元素。

get() 无参数，将匹配的所有元素转化为DOM元素组成的一个数组；

每个 jQuery 对象也被当成一个数组，所以我们也可以直接用数组的取值运算符来获得列表项：

$(‘input’)[0] 也可以直接获取DOM元素，

1. .index() 返回匹配元素在同辈中的索引位置

若匹配元素是个集合，则返回集合中第一个的索引

.index(element) 参数可以是DOM对象，亦可以是jQuery对象，返回这个对象element在原先集合中的索引

.index(selector) 参数是选择器，返回前面的元素在选择器匹配的元素中的索引。这个最特殊，参数是较多的一方，上面两个参数是较少一方

1. .load() 从服务器载入数据并且将返回的HTML代码并插入至匹配的元素中。

.load( url [, data ] [, complete(responseText, status, XMLHttpRequest) ] )

<url:向此地址发送请求>

data：向服务器发送的请求数据，形如{name:’jame’,age:25}

complete:当请求成功，且html被插入完时被调用

注意：默认GET方式请求，若传data数据，则自动改成POST请求；

1）这个函数有个隐含的回调函数，当他检查到一个成功的请求时，（当textStatus是‘success’或‘notmodified’时），load()方法就将返回的html内容设置到相匹配的节点当中去。

所以，一般可以简写为：.load(‘ajax.html’)

2）url地址后面可以跟选择器，以空格隔开，决定从返回的html里筛选选择器匹配的内容加载。

例如：$('#result').load('ajax/test.html #container');

3）关于回调函数

|  |
| --- |
| $('#go').click(function () {  $('.box').load('ajax.php .son',function (responseText,status,xhr) {  // if(status=='error'){  var msg ='出现错误：';  $('.error').html(msg+status+' '+xhr.status+' '+xhr.statusText+' '+xhr.responseText);  //}  });  });  参数解释：  status：成功返回success，失败返回error  responseText：请求的url返回的内容，php直接echo的内容  xhr： XMLHttpRequest对象  xhr.status :xhr的属性，响应code，成功返回200  xhr.statusText:响应状态的说明，成功返回OK |

1. $.getJSON(url[,data] [,success(data,status,jqXHR)]

返回：jqXHR对象，后面可以链式操作，跟.done() .fail() .always()

使用一个HTTP GET请求从服务器加载JSON编码的数据。

第二个参数，会转成字符串并进行url编码，附加到url后，发送给服务器

回调函数success里：

data：返回的数据，一般是javascript对象或数组定义的json结构

|  |
| --- |
| {"gender":"male","height":177} 这种形式可以直接在回调里用data.gender  或[{"gender":"male"},{"height":177}] 这种形式需要在回调里用$.each(data,function(index,value){  value.gender  })遍历 |

status:成功返回‘success’，失败返回‘error’

jqXHR：jQuery XHR对象，该对象实现了promise接口，拥有promise所有的属性、方法，有以下三个：

jqXHR.done(function(){})成功时调用

jqXHR.fail(function(){}) 失败时调用

jqXHR.always(function(){}) 总是调用

此方法相当于：

|  |
| --- |
| $.ajax({  dataType: "json", //返回json数据类型  url: url,  data: data,  success: success  }); |

|  |
| --- |
| $('#go').click(function () {  $.getJSON('ajax.php',{'name':'ju','age':26},function (*data*,*status*,*xhr*) {  alert(*data*.gender+' '+*status*+' '+*xhr*.status+ ' '+*xhr*.responseText);  }); });  回调函数的参数解释：  data:后台返回的js对象  status：成功返‘success’ 失败返‘error’  *xhr*.status 成功返回200  *xhr*.responseText：后台返回的json字符串，可以使用$.parseJSON(*xhr*.responseText)转化成js对象，就等同于data |

1. $.getScript(url[,success(script, textStatus, jqXHR)])

返回：jqXHR对象，后面可以链式操作，跟.done() .fail() .always()

使用一个HTTP GET请求从服务器加载并执行一个 JavaScript 文件

此方法相当于：

|  |
| --- |
| $.ajax({  url: url,  dataType: "script",  success: success  }); |

回调函数参数：

script:请求的js代码字符串

textStatus：成功返回‘success’ 失败返回‘error’

其他参数同以上连个方法类似

1. $.get(url[,data][,success][,dataType])

返回：jqXHR对象，后面可以链式操作，跟.done() .fail() .always()

data:可以为‘name=yang&age=27’这样的字符串，亦可{'name':'yang','age':26}对象形式，还可以不要；

success：回调函数

|  |
| --- |
| $('#go').click(function () {  $.get('ajax.php',{'name':'yang','age':26},function (data, textStatus, jqXHR) {  $('ul').html(data.gender+' '+textStatus+' '+jqXHR.status+' '+jqXHR.statusText+' '+jqXHR.responseText);  },'json')  });  输出：male success 200 OK {"gender":"male","height":177}  回调函数参数解释：  data：根据dataType，经过转化后的对象  textStatus和jqXHR的属性同上面相同  dataType:确定返回数据的类型，可不填，会自动识别； |

1. $.post(url [, data ] [, success ] [, dataType ])

data:POST方式传递的数据，可以为‘name=yang&age=27’这样的字符串，亦可{'name':'yang','age':26}对象，若是表单，则可以$(‘form’).serialize()

注意：data这个对象里，key不能是变量，value可以是变量，但可以直接传对象进去

1. serialize()

只有form元素检查它们所包含的输入框: $(‘form’).serialize();

在所有其他情况下，输入元素要序列化应该是集合的一部分传递给.serialize()方法:

$(‘input, textarea, select’).serialize();

1. .ajaxStart() 在ajax触发前触发

.ajaxStop() 在ajax触发后触发

在jQuery1.9之后，以上两个方法必须依附在document上，即：

$(document).ajaxStart(function(){})

1. $.extend(target [,object1][ ,objectN]) 将两个或更多的对象内容合并到第一个对象

返回：返回第一个对象target。

* + 1. 若第一个参数为true，$.extend(true,target,object1) 则object1和target递归合并；若无此参数，则后面的对象属性会覆盖前面的。
    2. 若不想改变目标对象，可以给第一个参数传个空对象

var obj = $.extend({},obj1,obj2); //这样，obj1不会改变

* + 1. 若只有一个参数，则代表目标对象被省略了，此时，jQuery对象被默认为目标对象，我们可以在jQuery的命名空间下添加新功能。

|  |
| --- |
| var *obj*={  success: function () {  alert('success');  } }; $.extend(*obj*); $.success(); 即jQuery.success(); |

1. $.fn 为jquery原型对象。

### 事件对象

事件对象是用来记录一些事件发生时的相关信息的对象。事件对象只有事件发生时才会产生，并且只能是事件处理函数内部访问，在所有事件处理函数运行结束后，事件对象就被销毁

event.target

通常用于比较 event.target 和 this 来确定事件是不是由于冒泡而触发的。

事件属性：

event.type:获取事件类型

event.pageX和event.pageY:获取鼠标当前相对于页面的坐标，简单来说是从页面左上角开始,即是以页面为参考点,不随滑动条移动而变化

event.preventDefault() 方法：阻止默认行为

event.stopPropagation() 方法：阻止事件冒泡 防止事件冒泡到DOM树上，也就是不触发的任何前辈元素上的事件处理函数

event.which：获取在鼠标单击时，单击的是鼠标的哪个键；event.which 将 event.keyCode 和 event.charCode 标准化了。event.which也将正常化的按钮按下(mousedown 和 mouseupevents)，左键报告1，中间键报告2，右键报告3

event.currentTarget : 在事件冒泡过程中的当前DOM元素

this和event.target的区别：

this是冒泡过程中触发事件的对象，会随着冒泡逐级改变。

e.target不会变，指直接触发事件的对象，即使是冒泡了，还是指最早触发的那个对象。

|  |
| --- |
| <**ul** id=**"parent-list"**>  <**li** id=**"post-1"**>**Item 1**</**li**>  <**li** id=**"post-2"**>**Item 2**</**li**> </**ul**> <**script**>  var *ul* = *document*.getElementById('parent-list');   var *li* = *document*.querySelectorAll('li');  *ul*.addEventListener('click', function (*e*) {  alert(this.textContent+' ul1');  alert(*e*.target.textContent+' ul2')  });  *li*[0].addEventListener('click',function(*e*){  alert(this.textContent+' li1');  alert(*e*.target.textContent+' li2')  }); </**script**> |