# 介绍

## 什么是JQ

一个优秀的JS库，大型开发必备

jQuery由美国人John Resig于2006年创建  
jQuery是目前最流行的JavaScript程序库，它是对JavaScript对象和函数的封装  
它的设计思想是write less,do more

## 为什么要使用库

一些JavaScript开发人员强烈建议编写自己的代码而不是使用库， 主要理由包括：

>使用库时只是调用其他人编写的算法和函数，所以我们不能确切了解库里的代码是如何运行的。

>JavaScript库里包含很多不会用到的代码， 但用户仍然需要下载它们。

## jQuery能做什么

访问和操作DOM元素  
控制页面样式  
对页面事件进行处理  
扩展新的jQuery插件  
与Ajax技术完美结合

## jQuery的优势

体积小，压缩后只有100KB左右  
强大的选择器  
出色的DOM封装  
可靠的事件处理机制  
出色的浏览器兼容性  
使用隐式迭代简化编程  
丰富的插件支持

简化JS的复杂操作  
不再需要关心兼容性  
提供大量实用方法

## 网站

### 官网

进入jQuery官网：<http://jquery.com>

### jq中文网

<https://www.jquery123.com/>

## jQuery库文件

jQuery库分开发版和发布版

### 无压缩版

jquery-3.4.1.js  
完整无压缩版本，主要用于测试、学习和开发

### 压缩版

jquery-3.4.1.min.js  
经过工具压缩或经过服务器开启Gzip压缩，主要应用于发布的产品和项目

## 在页面中引入jQuery

### 本地引入

<script src="js/jquery-3.3.1.min.js" type="text/javascript"></script>

### CDN引入

CDN就是“内容分发网络”，也就是CDN的方式引用它。

优点：不必下载jQuery； 当浏览器需要使用jQuery时，它很可能已经在缓存里了；CDN通常能够保证从最近地理位置的服务器提供文件，从而减少加载时间。

<script src="https://cdn.bootcss.com/jquery/3.4.1/jquery.min.js"></script>

## 网页加载完再执行

### JS方法

<script>  
 *window*.onload = function() {  
 alert("我欲奔赴沙场征战JavaScript，势必攻克之！");  
 };  
</script>

### JQ方法

ready() 其实就是为页面加载事件绑定方法

<script>  
  
 *// JQ方法一* $(*document*).ready(function() {  
 alert("我欲奔赴沙场征战jQuery，势必攻克之！");  
 });  
  
 *// JQ方法二* $(function(){  
 alert("HELLO JQUERY!");  
 });  
  
</script>

$(function(){});其实这个就是JQ ready()的简写，他等价于：$(document).ready(function(){})

### 区别

$(document).ready()与window.onload类似，但也有区别：

window.onload   
执行时机：必须等待网页中所有的内容加载完毕后（包括图片、flash、视频等）才能执行  
同一页面不能同时编写多个，如果有多个，那么后一个生效

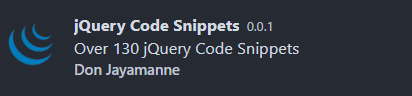
$(document).ready()  
网页中所有DOM文档结构绘制完毕后即刻执行，可能与DOM元素关联的内容（图片、flash、视频等）并没有加载完。也就是等DOM加载完就执行了，性能要好，对改善性能略有帮助。（原理：JQ使用了DOMContentLoaded事件）  
同一页面能同时编写多个

## jQuery 基本语法

jQuery 语法是为 HTML 元素的选取编制的，可以对元素执行某些操作

基础语法是：$(selector).action()

## VSCODE插件



# 获取节点

元素就是节点

在JQ中，一种效果可以使用很多种方法写出来，不一定非得用一种。

$(selector)：通过选择器选取节点

## 基本选择器

### 全局

\* 表示选择到所有网页元素

### 标签

p ，a ，img ，h1，span

### class

也叫类选择器，一个class可以用在多个元素上；一个元素也可以使用多个class，之间用空格隔开

### id

一个id只能用在一个元素上

### 交集

也叫指定选择器，把元素与class或id选择器结合起来用。

如span.red{ }

div#top{ }

### 并集

也叫群选择器，多个不同的选择器 设置 相同的样式，不同的选择器之间用逗号分隔。

#box,p,.class1{...}

### 示例

<script>  
 $(function () {  
 $("\*").css("color", "red"); *// 通配符（全局）选择器* $("p").css("color", "blue"); *// 标签选择器* $(".list").css("color", "green"); *// class选择器* $("#box").css("color", "purple"); *// id选择器* $("p.aaa").css("color", "pink"); *// 交集选择器* $("h1,p,a,.test").css("text-decoration", "underline"); *// 并集（群）选择器* });  
</script>

## 层次选择器

### 后代

div p 选择div里面所有的p段落

### 子代

div>p 选择div里面的所有子代p元素，不包括孙代p元素

### 相邻

div+p 选择在div后面的第一个p元素（如果div后面第一个元素不是p，则无效）

### 同辈

div~p 选择在div后面的所有的p元素

### 示例

<script>  
 *// 后代选择器* $("form input").css("border", "2px dotted blue");  
 $("form fieldset input").css("backgroundColor", "yellow");  
  
 *// 子代选择器* $("ul.topnav > li").css("border", "3px solid red");  
  
 *// 相邻兄弟选择器* $("label + input").css("color", "blue").val("Labeled!");  
  
 *// 通用兄弟选择器* $("#box ~ p").css("color", "blue");  
  
</script>

## 属性选择器

属性选择器都是放在中括号里面的

### [attr]

选择带有attr属性的元素

### [attr="value"]

属性值等于value

### [attr^="value"]

属性值以value开头

### [attr$="value"]

属性值以value结尾

### [attr\*="value"]

属性值包含value

### [attr~="value"]

属性值中含有value的，value是独立的，或用空格隔开才行，而\*只要包含就行，比~范围广

### [attr|="value"]

属性值为value或value-开头

### [attr!="value"]

jQuery多了一个：[attr!="value"]  
选取属性值不等于value的元素，CSS3没有这个选择器

## 伪类

除了4种鼠标状态伪类选择器，除了target，jQuery都支持，在jQuery中也叫过滤选择器

### first-child

E:first-child 第一个E元素（这个E元素必须是父元素的第一个子元素）

### first-of-type

E:first-of-type

第一个E类型的元素（这个E元素不一定是父元素的第一个子元素）

### last-child

E:last-child 最后一个E元素（这个E元素必须是父元素的最后一个子元素）

### last-of-type

E:last-of-type 最后一个E类型的元素（这个E元素不一定是父元素的最后一个子元素）

### nth-child(n)

选择第n个子元素（从前往后数）

n是从1开始

E:nth-child(n) 选择第n个E元素。  
li:nth-child(2n) {color:red}  
li:nth-child(2n+1) {color:red}  
li:nth-child(even){color:#f00;} /\* 偶数 \*/  
li:nth-child(odd){color:purple;} /\* 奇数 \*/

### nth-of-type(n)

E:nth-of-type(n) 选择第n个E类型的元素

### nth-last-child(n)

E:nth-last-child(n)

选择第n个子元素（从后往前数）

### nth-last-of-type(n)

E:nth-last-of-type(n)

### only-child

就是父盒子里面只有它一个子元素

<style>  
 ul li:only-child {  
 color: red;  
 }  
</style>  
<ul>  
 <li>1111111111</li>  
</ul>  
<ul>  
 <li>2222222</li>  
 <li>33333333</li>  
 <li>4444</li>  
</ul>

### only-of-type

就是父盒子里面只有它一个E类型的子元素，父盒子里面可以有多个子元素

<style>  
 ul li:only-of-type {  
 color: red;  
 }  
</style>  
<ul>  
 <p>pppp</p>  
 <p>ppppp</p>  
 <li>1111111111</li>  
</ul>

### empty

选择没有任何子元素（包括text节点）的元素E

<style>  
 div p:empty {  
 height: 25px;  
 border: 1px solid #ccc;  
 background: red;  
 }  
</style>  
<div>  
 <p>11111</p>  
 <p></p>  
 <p>33333</p>  
</div>

### :enabled

选择表单中处于可用状态的元素

input:enabled{color:red}

### :disabled

选择表单中处于禁用状态的元素

input:disabled{color:red}

<style>  
 input[type="text"]:enabled {  
 border: 2px solid blue;  
 background: greenyellow;  
 color: #000;  
 }  
  
 input[type="text"]:disabled {  
 border: 2px solid black;  
 background: orangered;  
 color: #fff;  
 }  
</style>  
<input type="text" value="可用状态" />  
<input type="text" value="禁用状态" disabled="disabled" />

### :checked

选择表单中被选中的radio或checkbox元素

input:checked{color:red}

<style>  
 input:checked+span {  
 background: #f00;  
 }  
  
 input:checked+span:after {  
 content: " 我被选中了";  
 }  
</style>  
  
<label><input type="radio" name="colour-group" value="0" /><span>蓝色</span></label>  
<label><input type="radio" name="colour-group" value="1" /><span>红色</span></label>

### :focus

选择获得焦点的 input 元素

<style>  
 input:focus {  
 background: #f6f6f6;  
 color: #f60;  
 border: 1px solid #f60;  
 outline: none;  
 }  
</style>  
<input value="姓名" />  
<input value="单位" />

### :seclected

选中的

### E:not(selector)

匹配不含有selector选择符的元素E

<style>  
 .test :not(p) {  
 color: red;  
 }  
</style>  
<div class="test">  
 <p>11111</p>  
 <p>22222</p>  
 <span>爱学吧</span>  
</div>

<style>  
 ul li:not(:last-child) {  
 border-bottom: 1px solid red;  
 }  
</style>  
<ul>  
 <li>11111</li>  
 <li>2222</li>  
 <li>333333</li>  
</ul>

<style>  
 div p:not(.abc) {  
 color: red;  
 }  
</style>  
<div>  
 <p>1111</p>  
 <p class="abc">222</p>  
 <p>33333</p>  
</div>

### E:lang(fr)

匹配使用特殊语言的E元素

<style>  
 div p:lang(en) {  
 color: #090;  
 }  
</style>  
<div>  
 <p lang="kr">大段测试文字</p>  
 <p lang="en">english</p>  
</div>

### :root

匹配根元素。在HTML中，根元素永远是HTML

### :contains(text)

选取包含特定文本的元素 $('td:contains("学习")') 返回包含 “学习”的td元素

### :has(selector)

选取包含selector元素的元素 $('tr:has("th")') 选取包含th元素的tr元素

## 过滤选择器

### :first

选取第一个元素

$(" li:first" )选取所有<li>元素中的第一个<li>元素

### :last

选取最后一个元素

$(" li:last" )选取所有<li>元素中的最后一个<li>元素

### :not(selector)

选取去除所有与给定选择器匹配的元素

$(" li:not(.three)" )选取class不是three的元素

### :even

选取索引是偶数的所有元素（index从0开始）

$(" li:even" )选取索引是偶数的所有<li>元素

### :odd

选取索引是奇数的所有元素（index从0开始）

$(" li:odd" )选取索引是奇数的所有<li>元素

### :eq(index)

选取索引等于index的元素（index从0开始）

$("li:eq(1)" )选取索引等于1的<li>元素

### :gt(index)

选取索引大于index的元素（index从0开始）

$(" li:gt(1)" )选取索引大于1的<li>元素（注：大于1，不包括1）

### :lt(index)

选取索引小于index的元素（index从0开始）

$(“li:lt(1)” )选取索引小于1的<li>元素（注：小于1，不包括1）

### :header

选取所有标题元素，如h1~h6

$(":header" )选取网页中所有标题元素

### :focus

选取当前获取焦点的元素

$(":focus" )选取当前获取焦点的元素

### :animated

选择所有动画

$(":animated" )选取当前所有动画元素

# 创建节点

使用HTML字符串创建jQuery节点，如：

    <script type="text/javascript">

        var mydiv = $("<div>");

        mydiv.appendTo($('body'))

        var newNode1 = $("<li></li>");

        var newNode2 = $("<li>你喜欢哪些冬季运动项目？</li>");

        var newNode3 = $("<li title='last'>北京申办冬奥会是再合适不过了！</li>");

    </script>

# 添加节点

## 内部添加

### appendTo()

$(A).appendTo(B) 把A添加到B的末尾（$(A).append(B) 把B添加到A的末尾）

### prependTo()

$(A). prependTo (B) 把A添加到B的开头（$(A). prepend (B) 把A添加到B的开头）

## 外部添加

### insertBefore()

$(A). insertBefore (B) 把A添加到B的前面 （$(A). before (B) 把B添加到A的前面）

### insertAfter()

$(A). insertAfter (B) 把A添加到B的后面 （$(A).after (B) 把B添加到A的后面）

## 示例

    <div class="box">

        <p>我是段落</p>

    </div>

    <span>我是span</span>

    <script>

*// $('span').appendTo($('.box'))*

*// $('span').prependTo($('.box'))*

*// $('span').insertBefore($('.box'))*

        $('.box').insertAfter($('span'))

    </script>

# 删除节点

## remove()

删除整个节点

remove()方法带来一个返回值，就是已删除那个节点

    <script>

        $("h1").click(function(){

            alert('hi')

        })

        var myh1 = $('h1').remove();

        $('body').append(myh1); *// 失去了绑定的事件*

    </script>

## empty()

清空节点里面的内容

## detach()

删除整个节点，保留元素的绑定事件、附加的数据

    <script>

        $("h1").click(function(){

            alert('hi')

        })

        var myh1 = $('h1').detach();

        $('body').append(myh1); *// 保留了绑定的事件*

    </script>

# 替换节点

replaceWith()

replaceAll()

两者的关系类似于append()和appendTo()

# 复制节点

clone()

用于复制某个节点，参数ture或flase, true复制事件处理，flase时反之

# 节点的内容

## html()

获取和设置节点的html内容使用的是同一个方法： html()；没有参数就是获取，有参数就是设置

### 获取

$(selector).html();

### 设置

$(selector).html(“<h1>aaa</h1>”);  
会解析html代码

### 注意

alert( $('li').html() );  
当一组元素的时候，获取的是一组中的第一个

$('li').html("hello")  
当一组元素的时候，设置的是所有元素

<div class="box">  
 <p>000</p>  
 <p>111</p>  
 <p>222</p>  
</div>  
<script>  
 var ***a*** = $('.box p').html();  
 ***console***.log(***a***); // 000  
  
 $('.box p').html('hello');  
</script>

## text()

获取和设置节点的文本内容使用的是同一个方法：text()；没有参数就是获取，有参数就是设置

### 获取

$(selector).text();

不会获取元素中的html标签，也就是会忽略html标签

### 设置

$(selector).text(“<h1>aaa</h1>”);  
不会解析html代码，当成文本直接输出

### 注意

当一组元素的时候，获取的是所有元素的内容，设置的也是所有元素

    <div class="box">

        <p>000</p>

        <p>111</p>

        <p>222</p>

    </div>

    <script>

*// var a = $('.box p').text()*

*// console.log(a); // 000111222*

        $('.box p').text('hello')

    </script>

## val()

获取和设置表单元素的value值使用的是同一个方法：val()；没有参数就是获取，有参数就是设置

### 获取

$(selector).val();

### 设置

$(selector).val(“hello”);

# 节点的属性

## attr()

获取和设置节点的属性使用的是同一个方法：attr()；没有参数就是获取，有参数就是设置

### 获取

$(selector).attr('属性名称')

### 设置

$(selector).attr('属性名称','属性值')  
多个属性使用JSON格式: $(selector).attr({'属性名称':'属性值','属性名称':'属性值'})

    <script>

        $("a").attr("target",function () {

            if (*this*.host == location.host) {

                return "\_self";*// 使站内链接在本窗口中打开，并且让*

            } else{

                return "\_blank";*// 非站内链接在新窗口中打开*

            }

        });

    </script>

## removeAttr()

移除节点的属性

# 节点的CSS

## css()方法

获取和设置节点的css使用的是同一个方法：css()；没有参数就是获取，有参数就是设置

在获取css属性时，css()返回的是元素当前样式，返回值可能来自style属性也可能来自样式表  
在设置css属性时，css()方法会将样式添加到该元素的style属性中

### 获取

css('css属性名称')

### 设置

css('css属性名称','css属性值')

添加多个css属性，用的是json形式：  
css({'css属性名称':'css属性值','css属性名称':'css属性值'})

## 节点的class操作

### addClass()

说明：为每个匹配的元素添加指定的样式类名

语法1：

$(selector).addClass( "className1 className2" )  
小括号里面是由一个或多个class名称组成的字符串

<style>  
 .selected {  
 color: red;  
 }  
  
 .highlight {  
 background-color: yellow;  
 }  
</style>  
<p>Hello</p>  
<p>and</p>  
<p>Goodbye</p>  
<script>  
 $("p").last().addClass("selected");  
 $("p:first").addClass("selected highlight");  
</script>

语法2：

$(selector).addClass( function(index, currentClass) {} )  
小括号里面是一个函数，这个函数使用return返回一个或多个用空格隔开的要增加的样式名。  
index 参数表示元素在匹配集合中的索引位置和html 参数表示元素上原来的 HTML 内容。在函数中this指向匹配元素集合中的当前元素。

<ul>  
 <li>111</li>  
 <li>222</li>  
 <li>333</li>  
</ul>  
  
<div>这是盒子是白色的</div>  
<div class="red">  
 这个盒子将会是绿色的因为它现在同时有.green和.red这两个类,  
 如果.addClass 中的函数不起作用，它将会是红色的  
</div>  
<div>这是盒子是白色的</div>  
<p>没有绿色的盒子</p>  
  
<script>  
  
 $("li").addClass(function (index) {  
 return "item" + index;  
 });  
  
 $("div").addClass(function (index, currentClass) {  
 var addedClass;  
  
 if (currentClass === "red") {  
 addedClass = "green";  
 $("p").text("有一个绿色的盒子");  
 }  
  
 return addedClass;  
 });  
</script>

### removeClass()

移除class

### toggleClass()

切换class，模拟了addClass()与removeClass()实现样式切换的过程

### hasClass()

判断是否包含指定的样式 ,返回布尔值

## 节点的宽度和高度

### 获取

获取元素的宽度，不带单位

width() width  
innerWidth() width+padding (相当于JS的clientWidth)  
outerWidth() width+padding+border (相当于JS的offsetWidth)  
outerWidth(true) width+padding+border+margin

注：innerWidth和outerWidth可以获取隐藏元素的宽，但是JS的获取不到

<style>  
 .box {  
 width: 100px;  
 height: 100px;  
 padding: 10px;  
 border: 5px solid blue;  
 background-color: red;  
 margin: 20px;  
 }  
</style>  
  
<div class="box">  
 一个盒子  
</div>  
  
<script>  
 console.log($(".box").width());  
 console.log($(".box").innerWidth());  
 console.log($(".box").outerWidth());  
 console.log($(".box").outerWidth(true));  
</script>

### 修改或添加

$(selector).width(100)

如果没有给定明确的单位（像'em' 或者 '%'），那么默认情况下"px"会被直接添加上去（也理解为"px"是默认单位）。

### height()

设置或返回匹配元素的宽度

$(this).width(30)

如果没有给定明确的单位（像'em' 或者 '%'），那么默认情况下"px"会被直接添加上去（也理解为"px"是默认单位）。

# 节点的位置

### .offset()

元素距离浏览器边框的距离，这个方法有两个属性：left和top

和这个元素的上级元素有没有定位没有关系

示例:

$(this).offset().left

$(this).offset().top

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

    <script type="text/javascript">

        $(document).on('click',function(){

            console.log(document.querySelector(".box p").offsetLeft);

            console.log( $('.box p').offset().left )

        })

    </script>

原生JS要想获得一个元素距离浏览器边框的距离是需要自己封装一个方法，比较麻烦（通过循环，一层层的累加才能得到）

### .position()

当前元素到定位上级的距离（和原生JS的offsetLeft有点像，但是它不包括maring值，而原生JS是包括maring值的）

如果给它设置个绝对定位，发现这个属性获取的永远都是你设置的那个定位的top/left值，会忽略padding和maring值

示例:

$(this).position().left

$(this).position().top

    <style>

        \* {

            margin: 0;

            padding: 0;

        }

        .box {

            width: 100px;

            height: 100px;

            background-color: red;

            margin-left: 100px;

            padding: 50px;

            position: relative;

        }

        .box p {

            width: 50px;

            height: 50px;

            background-color: green;

            margin-left: 20px;

        }

    </style>

    <div class="box">

        <p>我是段落</p>

    </div>

    <script type="text/javascript">

        $(document).on('click',function(){

            console.log(document.querySelector(".box p").offsetLeft);

            console.log( $('.box p').offset().left )

            console.log( $('.box p').position().left )

        })

    </script>

### .scrollTop()

获取或设置元素的垂直滚动条的位置

示例:

$("p:first").scrollTop()

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

    <script type="text/javascript">

        $(document).click(function(){

            alert($(window).scrollTop())

        })

    </script>

### .scrollLeft()

获取匹配的元素集合中第一个元素的当前水平滚动条的位置。设置每个匹配元素的水平滚动条位置。

# 节点的数量

## length属性

返回的jQuery对象匹配的DOM元素的数量。

    <ul>

        <li>000</li>

        <li>111</li>

        <li>222</li>

    </ul>

    <script>

        console.log($("li").length);

    </script>

# 节点对象的转换

jQuery对象和Dom元素都是对象，但它们是不同的  
jQuery对象是Dom对象的一种包装，具有更多面向对象的方面，例如方法，属性等，

它们可以相互转换

# 遍历

## DOM节点遍历

### children()

语法：.children( [selector ] )  
selector：一个用于匹配元素的选择器字符串，CSS3和JQ选择器都行。可选

子代选择器。还要注意的是和大多数的jQuery方法一样，.children()不返回文本节点;让所有子元素包括使用文字和注释节点，建议使用.contents()。

<script>  
 // 整个页面所有的div里面的所有子代元素，只要是子代元素即可，不限类型  
 $("div").children().css('color', 'red')  
  
 // 整个页面所有的div里面的所有p类型子代元素  
 $("div").children('p').css('color', 'blue')  
 // 相当于：  
 $("div > p").css('color', 'blue')  
  
  
 // 整个页面所有的div里面的所有.on子代元素  
 $("div").children('.on').css('color', 'blue')  
 // 相当于：  
 $("div > .on").css('color', 'blue')  
</script>

### find()

格式：$(selector).find( 必选参数 )  
参数类型： 1，选择器字符串；2，jQuery对象；3，元素。

<div>  
 <h1>111</h1>  
 <p>222</p>  
 <p>333</p>  
</div>  
  
<div>  
 <h4>444</h4>  
 <p class="on">555</p>  
 <p class="on">666</p>  
 <h2>  
 <p class="on">777</p>  
 <p class="on">888</p>  
 </h2>  
</div>  
<script>  
  
 // 整个页面所有的div里面的所有p类型元素  
 //  $("div").find('p').css('background-color', 'red'); // 235678  
 // 相当于：  
 //  $("div p").css('background-color', 'red'); // 235678  
  
 // 整个页面所有的div里面的所有p类型元素中的第一个  
 //  $("div").find('p:first').css('background-color', 'red'); // 25  
  
 //  $("div p:first-child").css('background-color', 'red'); // 7  
 //  $("div p:first-of-type").css('background-color', 'red'); // 257  
  
 // 使用JQ对象  
 //  var myh4 = $('h4');  
 //  $("div").find(myh4).css('background-color', 'red'); // 4  
  
 // 使用DOM元素  
 var ***p1*** = $('p.on')[0];  
 $('div').find(***p1***).css('background-color', 'red'); // 5  
  
</script>

实例：

通过jQuery对象查找：  
var $allListElements = $('li');  
$('li.item-ii').find( $allListElements );

同样，一个元素可能也可以通过查找：  
var item1 = $('li.item-1')[0];  
$('li.item-ii').find( item1 ).css('background-color', 'red');

以下等同于：$( "p span" )

<p><span>Hello</span>, how are you?</p>  
<p>Me? I'm <span>good</span>.</p>  
<script>  
 $("p").find("span").css('color', 'red');  
</script>

由span标签组成的JQ集合：

<p><span>Hello</span>, how are you?</p>  
<p>Me? I'm <span>good</span>.</p>  
<div>Did you <span>eat</span> yet?</div>  
<script>  
 var *$spans* = $('span');  
 $("p").find(*$spans*).css('color', 'red');  
</script>

### prev()

上一个兄弟节点

格式：$(selector).prev(可选参数)

参数类型：字符串形式的选择器

<div>盒子一</div>  
<span>我是span</span>  
<p>我是p</p>  
<b>我是b</b>  
<script>  
 $(function(){  
 $("div").next().css("color",'red');  
 $("b").prev().css("color",'blue');  
 });  
</script>

### next()

下一个兄弟节点

格式： $(selector).next(可选参数)

参数类型：字符串形式的选择器

### prevAll()

前面的所有兄弟节点

格式：$(selector).prevAll(可选参数)

参数类型：字符串形式的选择器

### nextAll()

后面的所有兄弟节点

$(selector).nextAll(可选参数)

参数类型：字符串形式的选择器

### siblings()

所有兄弟节点

格式：$(selector).siblings(可选参数)

参数类型：字符串形式的选择器

### parent()

格式：$(selector).parent(可选参数)

参数类型：字符串形式的选择器

获取元素的父级元素

获取元素的父级元素

<div class="box">

<p>00000</p>

</div>

<script>

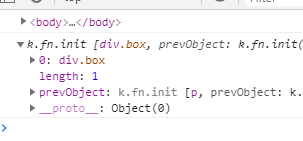
$(document).on('click', function () {

console.log(document.querySelector(".box p").offsetParent);

console.log($('.box p').parent())

})

</script>



### parents()

$(selector).parents(可选参数)

参数类型：字符串形式的选择器；参数就是筛选功能

获取当前元素的所有上级元素

    <div class="box1">

        <div class="box2">

            <p class="myp">我是p</p>

        </div>

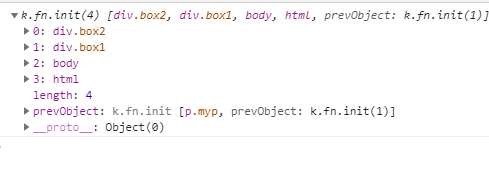
    </div>

    <script>

        console.log($('.myp').parents());

        $('.myp').parents().css("border",'2px solid red')

    </script>





可以添加一个参数，参数就是筛选功能：

    <div class="box1">

        <div class="box2">

            <p>我是p</p>

        </div>

    </div>

    <script>

        $('.box2 p').parents('.box1').css("border",'2px solid red')

    </script>

### closest( )

$(selector).closest(必选参数)

参数类型：1，选择器字符串；2，jQuery对象；3，元素。

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。只能找到一个元素；

### offsetParent()

$(selector).offsetParent()

获取有定位的上级元素。定位指的是元素的CSS position值被设置为relative、absolute或fixed的元素

## 过滤

过滤就是从匹配元素集合中找到想要的元素。（就是从一堆元素中找我们想要的那些）

### first()和last()

语法：$(selector).first()  
获取匹配元素集合中第一个元素。这个方法不接受任何参数。  
$("li").first().css() 相当于 $("li:eq(0)").css() 相当于 $("li:first").css() 相当于 $("li:first-child").css()

语法：$(selector).last()  
获取匹配元素集合中最后一个元素。这个方法不接受任何参数。

<ul>  
 <li>list item 1</li>  
 <li>list item 2</li>  
 <li>list item 3</li>  
</ul>  
<p>  
 <span>Look:</span>  
 <span>This is some text in a paragraph.</span>  
 <span>This is a note about it.</span>  
</p>  
<script>  
 $("li").first().css("background-color", "red");  
 $("p span").first().addClass('highlight');  
</script>

### filter()和not()和has()

filter: 把需要的过滤出来  
not： filter的反义词，把不需要的过滤出来  
has：包含

    <ul id="list">

        <li> <span class="a">我是span</span> 111111</li>

        <li class="a">2222222</li>

        <li>33333333</li>

        <li class="b">4444444</li>

        <li class="a">5555555555</li>

    </ul>

    <script>

*// $("#list li").css("background-color",'red'); // 所有li都会变*

*// $("#list li").filter('.a').css("background-color",'red'); // 带.a的li会*

*// $("#list li").not('.a').css("background-color",'red'); // 不带.a的li*

        $("#list li").has('.a').css("background-color",'red'); *// 子元素中有带.a的li*

    </script>

### eq()

下标从0开始

    <h1>我是标题00000</h1>

    <div class="box">

        <p>111111</p>

        <p>2222</p>

        <div>

            <p>333</p>

        </div>

        <p class="selected">444444</p>

        <h1>我是标题1111</h1>

    </div>

    <script>

        $("h1").eq(1).css('color','blue')

        $(".box").find("p").eq(3).css('color','red')

    </script>

### index()方法

索引就是当前元素在所有兄弟节点中的位置，从0开始（就是当前元素在一组元素中的位置，当前元素和这一组元素是平级的兄弟关系）

<div>  
 <h1>11111</h1>  
 <h2>222222222</h2>  
 <h1>333333333</h1>  
 <h2 id="hhh">444444444444</h2>  
 <h1>555555555555</h1>  
 <p>666666666</p>  
</div>  
<h2>7777777777</h2>  
<script>  
 $(function(){  
 alert($('#hhh').index()); *// 3* });  
</script>

## 其他遍历

### each()方法

遍历一个jQuery对象，为每个匹配元素执行一个函数。

.each( function(index, Element) )

index 下标

element：每个元素

<div class="box">  
 <p>苹果</p>  
 <p>橘子</p>  
 <p>西瓜</p>  
</div>  
<script>  
 $('.box p').each(function (index, element) {  
 // element == this  
 // console.log(index + ':' + $(this).text());  
  
 // 或者  
 ***console***.log(index + ':' + element.innerHTML);  
 });  
</script>

### end()方法

结束当前链条中的最近的筛选操作，并将匹配元素集还原为之前的状态

## jQuery对象转为Dom对象

var domObj = jqObj.get（index）;  
通过jQuery对象获取一个对应的DOM元素  
index从0开始计数，用来确定获取哪个元素。不写index就是指所有的都转为dom对象  
示例： var mybox = $(“.box”).get(0)

    <script>

*// 用 $('.box p').length 也可以，因为jq也有个length属性*

        for (let i = 0; i < $('.box p').get().length; i++) {

            $('.box p').get(i).style.color = 'blue'

        }

    </script>

其实通过下标也可以把jq对象转为dom对象

('.box p')[0].style.color = 'blue'

注意：

$("li:first") 是一个jQuery对象，可以使用 .html()等jq方法；  
$("li")[0] 是一个DOM元素，不可以使用 .html()等jq方法  
相当于document.getElementsByTagName(‘li’)[0].html() 会报错

## Dom对象转为jQuery对象

var jqObj = jQuery（domObj）;  
var box = document.getElementById(“box”)  
示例： var mybox = jQuery（box）

# 事件

## 事件类型

jQuery事件类型和原生JS事件类型一致

增加的有鼠标事件：

### hover()

规定当鼠标指针悬停在被选元素上时要运行的两个函数，相当于 mouseenter 和 mouseleave 事件的组合

如果只指定一个函数，则 mouseenter 和 mouseleave 都执行它

    <script>

        $('.box').hover(function () {

            $(*this*).css("backgroundColor",'blue');

        },function () {

            $(*this*).css("backgroundColor",'red');

        })

    </script>

## 绑定事件方法一

JQ中给对象添加事件使用如下格式： $(selector).eventName(function(){})

如: $(selector).click(function(){})  
$(selector).keydown(function(){})

在jQuery里可以用多种方式给单个元素或一组元素添加事件处理器。最直接的方法是这样的：

<**script type="text/javascript"**>  
 $(***document***).**ready**(**function** () {  
 $(**'button'**).click(**function** () {  
 *alert*(**"hello jquery"**);  
 });  
 });  
</**script**>

或者像下面这样使用命名的函数：

<**script type="text/javascript"**>  
 $(***document***).**ready**(**function** () {  
 **function** hello() {  
 *alert*(**"hello jquery"**);  
 }  
 $(**'button'**).click(hello);  
 });  
</**script**>

jQuery以跨浏览器的方式包装了attachEvent和addEventListener方法，从而便于添加多个事件处理器。

## 绑定事件方法二

on()方法是在jQuery1.7引入的，用于取代以前一些事件处理方法，包括bind(), delegate() 和 live()

on()方法可以给原本存在于HTML页面的元素或者动态添加DOM的元素添加处理器。

### 绑定单个事件

<script>  
 $('#box').on('click', function () {  
 alert(1);  
 });  
</script>

### 绑定多个事件

绑定多个事件执行同一个函数

<script>  
 $(".box").on("click mouseover",function(){  
 console.log(1);  
 });  
</script>

绑定多个事件，每个事件执行不同的函数；使用对象字面量格式

<script>  
 $(".box").on({  
 mouseover: function(){  
 console.log(1);  
 },  
 mouseout: function(){  
 console.log(2);  
 }  
 });  
</script>

### 绑定自定义事件

<script>

$('#box').on('myshow', function () {

alert(1);

});

</script>

### 只执行一次事件

    <script type="text/javascript">

        $('p').one('click',function(){

            alert('hi')

        })

    </script>

## off() 移除事件

使用off()方法

当off()不带参数时，表示移除所绑定的全部事件

<script>  
 $('#box').on('click', function () {  
 alert('hello');  
 $("#box").off();  
 });  
</script>

    <script type="text/javascript">

        $(document).ready(function () {

            $("#btn").on("click", fn1);

            $("#btn").on("click", fn2);

            $("#btn").off(); *//解除所有的绑定事件*

            $("#btn").off("click"); *//解除所有的click绑定事件*

            $("#btn").off("click", fn1); *//只解除fn1的click绑定事件*

            function fn1() {

                $(".p1").hide();

            }

            function fn2() {

                $(".p2").hide();

            }

        })

    </script>

## 事件对象

### 阻止事件冒泡

jQuey阻止事件冒泡只有：event.stopPropagation() 方法，没有event.cancelBubble=true 方法

event.stopPropagation()

就是不触发任何前辈元素上的事件处理函数

event.stopImmediatePropagation()

阻止元素身上所有的事件处理函数的执行

### 阻止默认事件

e.preventDefault()

### return false

JQ中使用return false相当于 e.stopPropagation() + e.preventDefault()

阻止默认事件+阻止冒泡的操作

### 鼠标事件对象

和JS中的鼠标对象的属性是通用的。

    <script type="text/javascript">

        $(document).click(function(e){

            console.log(e.clientX);

            console.log(e.clientY);

            console.log(e.pageX);

            console.log(e.pageY);

            console.log(e.offsetX);

            console.log(e.offsetY);

            console.log(e.screenX);

            console.log(e.screenY);

        })

    </script>

### 键盘事件对象

JQ中的键值可以使用 e.which获取，也可以使用JS的e.keyCode获取

原生js中的event.button和event.keyCode 可以用jQuey中的event.which来代替

    <script type="text/javascript">

        $(document).keydown(function(e){

            console.log(e.which);

            console.log(e.keyCode);

        })

    </script>

# 特效

## show()

控制元素的显示，

默认400毫秒

## hide()

控制元素的隐藏

## toggle()

toggle()在显示和隐藏之间切换

## fadeIn()

淡入

## fadeOut()

淡出

## fadeTo()

调整匹配元素的透明度。

fadeTo()参数：

时间（默认400,或"slow", "normal", 或 "fast"），

透明度（0-1）

缓动（默认: "swing"，还有"linear"）

回调函数（在动画完成时执行的函数）

## .fadeToggle()

通过匹配的元素的不透明度动画，来显示或隐藏它们。

## .slideDown()

用滑动动画显示一个匹配元素。

## .slideUp()

用滑动动画隐藏一个匹配元素。

## .slideToggle()

用滑动动画显示或隐藏一个匹配元素。

关于参数：

时间（默认400,或"slow", "normal", 或 "fast"），

缓动（默认: "swing"，还有"linear"）

回调函数（在动画完成时执行的函数）

## .animate()

根据一组 CSS 属性，执行自定义动画。

参数：

属性：CSS属性和值的对象,动画将根据这组对象移动。所有用于动画的属性必须是数字的，比如width, height或者left可以执行动画，但是background-color不能

时间（默认400,或"slow", "normal", 或 "fast"），

缓动（默认: "swing"，还有"linear"）

回调函数（在动画完成时执行的函数）

.stop()

停止匹配元素当前正在运行的动画。

# 链式操作

链式操作也称为命令链。

jQuery的大多数方法都返回一个jQuery对象，可以用于再调用其他方法，这是jQuery的另一个方便之处。命令链的长度没有什么限制，从而可以对同一组元素连续进行很多操作：

$('.box').text('hello jquery').fadeOut().fadeIn();

上面这行代码会先淡出指定的元素，然后淡入显示它们。

## 示例

由于text()和animate()方法是对同一个元素进行操作，我们可以用命令链的方式实现：

<**style**>  
 .**box** {  
 **width**: 100**px**;  
 **height**: 400**px**;  
 **background-color**: **#f00**;  
 **position**: **absolute**;  
 **top**: 100**px**;  
 **left**: 100**px**;  
 **border**: 2**px solid black**;  
 **padding**: 20**px**;  
 }  
</**style**>  
  
<**div class="box"**></**div**>  
  
<**script type="text/javascript"**>  
 $(document).ready(**function** () {  
 $(**'.box'**).text(**'改变形状哦...'**).animate({ width: **'400px'**, height: **'200px'** }, 5000, **function** () {  
 $(**this**).text(**"淡出哦..."**).fadeOut(4000);  
 }  
 );  
 });  
</**script**>

# Ajax

## load()

如果只是需要从服务器获取一个文档并在页面元素里显示它， 那么只需要使用load() 方法就可以了。比如下面的代码片段会获取newContent.html， 并且把它的内容添加到id为“elem”的元素：

<**script type="text/javascript"**>  
 $(**function** () {  
 $(**'#elem'**).load(**"newContent.html"**);  
 });  
</**script**>

在使用load() 方法时， 除了指定URL外， 还可以传递一个选择符， 从而只返回相应的页面内容：

<**script type="text/javascript"**>  
 $(**function** () {  
 $(**'#elem'**).load(**"newContent.html #info"**);  
 });  
</**script**>

上面的范例在URL之后添加了一个j Query选择符，中间以空格分隔。这样就会返回选择符指定的容器里的内容， 本例就是id为“info”的元素。

为了弥补load() 方法的简单功能， jQuery还提供了发送GET和POST请求的方法。

## get() 和post()

这两个方法很类似，只是调用不同的请求类型而已。调用这两个方法时不需要选择某个jQuery对象(比如某个或一组页面元素) ， 而是直接调用：$.get() 或 $.post() 。在最简单的形式中， 它们只需要一个参数， 就是目标URL。

通常情况下我们还需要发送一些数据， 它们是以“参数/值”对的形式出现的， 以JSON风格的字符串作为数据格式。

大多数情况下，我们会对返回的数据进行一些处理，为此还需要把回调函数作为参数。

<**script type="text/javascript"**>  
 $.get(**"serverScript.php"**, { **param1**: **'value1'**, **param2**: **'value2'** },  
 **function** (data) {  
 ***console***.log(**"服务器响应："** + data);  
 }  
 );  
</**script**>

post() 方法的语法基本上是相同的：

<**script type="text/javascript"**>  
 $.post(**"serverScript.php"**, { **param1**: **'value1'**, **param2**: **'value2'** },  
 **function** (data) {  
 ***console***.log(**"服务器响应："** + data);  
 }  
 );  
</**script**>

TIP提示：如果是从表单字段获取数据， jQuery还提供了serialize() 方法，该方法可以获取表单信息， 进行序列化，满足Ajax调用的需要：

var formdata = $('#form1').serialize()

## ajax()

ajax()方法具有很大的灵活性， 几乎可以设置关于Ajax调用及如何处理响应的各个方面。详细的介绍请见https://api.jquery.com/jQuery.ajax/的文档。

# JQ的工具方法

JQ对象的方法，只能给JQ对象用：  
$(selector).css() $(selector).html() $(selector).val()

JQ工具方法，不仅可以给JQ用，也可以给原生JS用：  
$.xxx() $.yyy() $.zzz()

## $.type()

判断数据类型

<script>  
 ***console***.log($.type('hello')); // string  
 ***console***.log($.type(123)); // number  
 ***console***.log($.type(true)); //boolean  
 ***console***.log($.type([])); // array  
 ***console***.log($.type(/box/)); // regexp  
 ***console***.log($.type({name:"qian"})); // object  
 ***console***.log($.type(null)); // null  
 ***console***.log($.type(undefined)); // undefined  
 ***console***.log($.type(function(){})); // function  
</script>

## $.trim()

去除首尾空格

<script>  
 var ***str*** = ' hello ';  
 ***console***.log('{' + ***str*** + '}'); // { hello }  
 ***console***.log('{' + $.trim(***str***) + '}'); // {hello}  
</script>

## $.inArray()

从数组中找元素，找到返回下标，找不到返回-1

<script>  
 var ***arr*** = ['a','b','c','d']  
 ***console***.log($.inArray('e',***arr***)); // -1  
 ***console***.log($.inArray('c',***arr***)); // 2  
</script>

## $.proxy()

用来改变函数内this的指向

<script>  
 function say() {  
 ***console***.log(this);  
 }  
 say(); // window  
 var ***say2*** = $.proxy(say,***document***)  
 ***say2***(); // document  
</script>

传参

<script>  
 function sum(x,y) {  
 ***console***.log(x+y);  
 }  
 var ***sum2*** = $.proxy(sum,***document***)  
 ***sum2***(8,9); // 17  
  
 // 或者：  
 // var sum2 = $.proxy(sum,document, 8, 9)  
 // sum2()  
</script>

## $.noConflict()

防止冲突的

<script>  
  
 var ***zkeji*** = ***$***.noConflict();  
 ***zkeji***(***document***).click(function (e) {  
 e.preventDefault();  
 ***zkeji***('body').css('background','red')  
 });  
  
 // 可以放心使用$了，如果有其他库使用了$,可以直接在下面引用  
 var ***$*** = 10;  
 ***console***.log(***$***);  
</script>

## $.parseJson()

接受一个标准格式的 JSON 字符串，并返回解析后的 JavaScript 对象。

传入格式有误的 JSON 字符串可能导致抛出异常。例如，下面这些 JSON 字符串格式都不对：  
{test: 1} (test 没有使用双引号包裹).  
{'test': 1} ('test' 用了单引号而不是双引号包裹).

<script>  
 var ***jsonstr*** = '{"name":"xiaoming", "age":33}'  
 ***console***.log($.parseJSON(***jsonstr***));  
 ***console***.log($.parseJSON(***jsonstr***).name);  
</script>

## $.makeArray()

转换一个类似数组的对象成为真正的JavaScript数组。

<div class="box">  
 <p>000</p>  
 <p>111</p>  
 <p>222</p>  
</div>  
<script>  
 var ***a*** = $.makeArray($('.box p'))  
 ***console***.log($.type(***a***)); // array  
 ***console***.log(***a***); // [p, p, p] 数组里面放的是具体的p节点对象  
 ***console***.log($.type(***a***[0])); // object  
 ***console***.log(***a***[0].innerHTML); // 000  
</script>

## $.each()

一个通用的迭代函数，它可以用来无缝迭代对象和数组。数组和类似数组的对象通过一个长度属性（如一个函数的参数对象）来迭代数字索引，从0到length - 1。其他对象通过其属性名进行迭代。

语法：jQuery.each( 遍历的对象或数组, callback(indexInArray, valueOfElement) )

# 注意事项

JS和JQ不能混写：

document.getElementById('box').innerHTML //纯js写法  
$('#box').html() //纯jq写法

document.getElementById('box').html() 错误写法  
$('#box').innerHTML 错误写法

------------------

this.innerHTML //纯js写法  
$(this).html() //纯jq写法

this.html() //错误写法  
$(this).innerHTML //错误写法

--------------

// 改写html标签里面的内容：  
// 元素.innerHTML=’aaa’ js写法  
// $(‘元素’).html(‘aaa’) JQ写法：