前端 - 7

jQuery 基础

JavaScript 的 jQuery 库

jQuery 对象 和 DOM 对象

jQuery 选择器

jQuery 的常用 API（ 样式、属性、效果和元素操作）

jQuery 事件

jQuery 插件

jQuery 尺寸位置操作

黑马程序员课程
Pink 老师

目录

一. jQuery 概述.....9

1.jQuery 下载与引入.....9

2.jQuery 的使用.....10

顶级对象 $.....10

多库共存.....10

入口函数.....11

3.jQuery 对象.....12

jQuery 对象转换为 DOM 对象.....12

3.DOM 对象.....13

DOM 对象转换为 jQuery 对象.....13

二. jQuery 获取元素.....14

2. jQuery 筛选方法.....14

- parents（“选择器”） 父级筛选器.....15
- find（“选择器”） 子级筛选器.....15

案例 – 用 jQuery 做下拉菜单.....16

案例 – 用 jQuery 做简易 tab 栏切换.....17

1. jQuery 选择器.....18

3. jQuery 筛选选择器.....19

- jQuery 隐式迭代.....20
- jQuery 排他思想.....20
- jQuery 链式编程.....21

三．jQuery 元素样式.....22

1.单一样式.....22

1.1 只写属性名，不写值.....22

1.2 写属性，也写值.....22

2.多样式.....23

3.类名 常用.....23

- addClass( ) 添加类.....24
- removeClass( ) 删除类.....24
- toggleClass( ) 切换类.....25

案例 – 用增删类名改善简易 tab 栏切换.....26

四．jQuery 元素效果.....27

1.隐藏删除和切换.....27

- hide( ) 隐藏元素.....27
- show( ) 显示元素.....27
- toggle( ) 切换元素的显示隐藏.....28

案例 - 显示隐藏下拉菜单.....28

2.下拉上拉和切换.....29

- slideDown( ) 下拉显示元素.....29
- slideUp( ) 上拉隐藏元素.....29
- slideToggle( ) 切换元素的下拉和上拉.....30

案例 - 下拉上拉菜单.....30

3.淡入淡出和切换.....31

- fadeIn( ) 淡入显示元素.....31
- fadeOut( ) 淡出隐藏元素.....31
- fadeToggle( ) 切换元素的淡入和淡出.....32

案例 - 盒子缓慢消失和显示.....32

- fadeTo( ) 渐进调整元素透明度.....33

案例 - 切换高亮显示.....33

- stop( ) 停止动画队列.....34

案例 - 用 stop 停止下拉菜单的动画排队.....34

4.自定义动画 animate.....35

案例 - 折叠卡片/手风琴效果.....36

五. jQuery 元素属性.....37

1. 获得/修改元素 固有属性.....37

- prop( ) 获得固有属性的值.....37
- prop( ) 修改固有属性的值.....37

案例 - 复选框和单选框的全选/全不选.....38

- :checked 筛选选择器.....38

2. 获得/修改元素 自定义属性.....39

- attr( ) 获得自定义属性.....39
- attr( ) 修改自定义属性.....39

3. 获得/添加元素 数据缓存属性.....40

- data( ) 获得 H5 新增 data-开头属性.....40
- data( ) 数据缓存 添加属性.....40

六．jQuery 元素内容.....41

1.获得/修改元素 HTML 结构+内容.....41

- html( ) 获得结构+文本内容.....41
- html( ) 添加/修改结构+文本内容.....41

2.获得/修改元素 文本内容.....42

- text( ) 仅获得文本内容.....42
- text( ) 添加/修改文本内容.....42

3.获得/修改 input 表单 内容.....43

- val( ) 获得表单内容.....43
- val( ) 添加/修改表单内容.....43

案例 - 增添商品的数量.....44

案例 - 根据修改的商品数量修改总价.....45

- toFixed(number) 保留小数点后位数.....45

七．jQuery 操作元素.....46

1.遍历元素.....46

- each( ) 遍历 DOM 元素.....46

案例 - 商品总数量和总价.....47

- $.each( ) 遍历数据.....49

2.创建元素.....50

- $(” 标签+文本内容 “).....50

3.内部添加元素 做子元素.....51

- append( ) 添加到元素内部最后.....51

- prepend( ) 添加到元素内部最前..... 51

4.外部添加元素 做兄弟元素..... 52

- after( ) 添加到元素外部最后..... 52
- before( ) 添加到元素外部最前..... 52

5.删除元素..... 53

- remove( ) 删除元素自身..... 53
- empty( ) 删除元素的内容..... 53
- html(“”) 删除元素的内容..... 53

案例 - 删除商品..... 54

6.拷贝/合并 对象..... 55

- $.extend( ) 拷贝对象..... 55

1.拷贝给的对象为空..... 55

2.拷贝给的对象原有内容..... 55

3.浅拷贝..... 56

4.深拷贝..... 56

八. jQuery 事件..... 57

1.事件注册/解绑..... 57

元素.事件(执行程序) 单个事件注册..... 57

- on( ) 注册事件 推荐..... 58

1.on( ) 注册单个事件..... 58

2.on( ) 注册多个事件..... 58

3.on( ) 实现事件委派(事件委托)..... 59

3.on( ) 给动态生成的元素绑定事件..... 59

案例 - 简易留言板..... 60

- off( ) 解绑事件..... 61
- one( ) 仅执行一次事件..... 62

2.自动触发事件..... 62

元素.事件名( )..... 62

- trigger( )..... 62
- triggerHandler( )..... 62

3.事件对象..... 63

阻止默认行为..... 63

阻止事件冒泡..... 63

案例 - 判断键盘按下..... 63

3.常用事件..... 64

- hover( ) 鼠标经过和离开..... 64
- change( ) input 表单变化事件..... 64

九．jQuery 尺寸、位置操作..... 65

1.元素尺寸 高度宽度..... 65

2.元素位置..... 66

- offset( ) 元素距离文档的位偏移 可读写..... 66
- position( ) 元素距离定位父元素的位偏移 只读..... 67
- scrollTop( )/scrollLeft 页面滚动后元素卷去的部分..... 68

案例 - 滚动页面显示导航条..... 69

案例 - 页面直接跳转到指定位置.....70

案例 - 页面动画滚动到指定位置.....70

用$(“body，html”）代替 document.....70

一. jQuery 概述

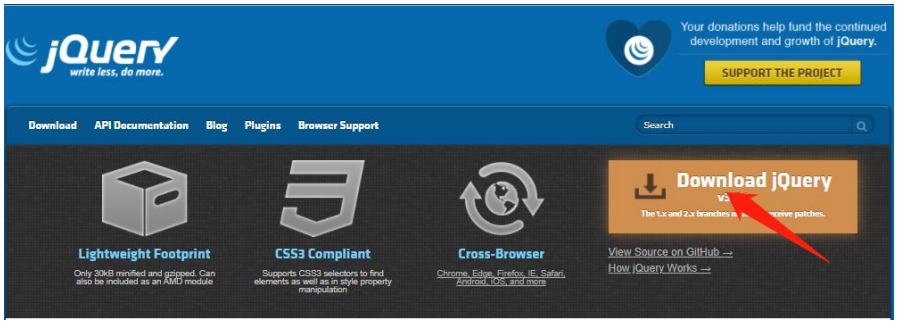
jQuery 就是一个 JS 库(library)，**里面封装的是 JS 里的常用函数（方法）。**

jQuery 的使用就是**调用函数。**通过调用这些封装好的函数实现了快速方便开发。

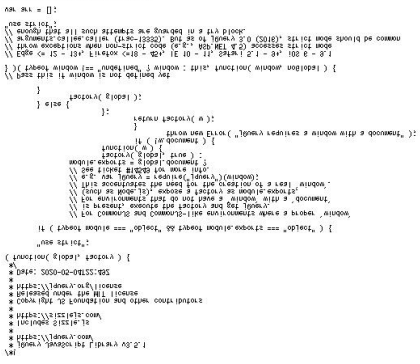
1.jQuery 下载与引入

下载

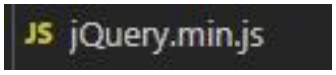
jQuery 官网下载　https://jquery.com/



分为 development 版本　　**production 版本（压缩版）** **常用**



复制全选 production 版本的所有代码，存入一个 JS 文件。命名 **jQuery.min.js**



引入

jQuery 被放入了一个 JS 文件，需要引入 HTML 页面文档。

在<head>内用**<script>引入 jQuery.min.js 文件。**

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../js/jQuery.min.js"></script>
</head>
```

2.jQuery 的使用

```
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <script>
        jQuery('div').click(function() {
            jQuery(this).css("background", "red").siblings().css("background", "green")
        });

        $('div').click(function() {
            $(this).css("background", "red").siblings().css("background", "green")
        });
    </script>
</body>
```

原生 JS 需要 function 函数名（）{ } 声明函数，函数调用时： 函数名（）。

使用 jQuery 时候，要**写** jQuery( ) 或 $（）调用 jQuery 的函数。

顶级对象 $

$ 是 **jQuery** 的别称，一般为了方便，直接使用 **$**。

$ 是 jQuery 的顶级对象，相当于原生 JS 中的顶级对象 window。

多库共存

解决 jQuery 的 $ 和其他 JS 库的 $ 冲突，

方法一：在 jQuery 中不使用 $,而是用 jQuery

方法二：释放权限，用 noConflict( )方法 让用户自定义

var 自定义名 = $.noConflict( )

```
<div></div>
<script>
    jQuery("div");
    $("div");

    var jq = $.noConflict();
    jq("div")

    console.log(jq("div"));
```

入口函数

和原生 JS 一样，若代码不在目标元素的后面，JS 从上到下执行会找不到该元素，无法执行。

原生 JS 用到了 **load / DOMContentLoaded** 窗口加载事件：

```
<script>
    //  先加载完所有页面DOM，再执行里面的JS
    document.addEventListener('DOMContenLoaded', function() {

    });
    //先加载完所有页面元素，再执行里面的JS
    window.addEventListener('load', function() {

    });
</script>
```

jQuery 需要一个入口函数，类似原生的 DOMContentLoaded 窗口加载事件。

jQuery 的入口函数是等仅所有 DOM 结果渲染完毕即可执内部代码。

```
$(function () {
    ...  // 此处是页面 DOM 加载完成的入口
 }) ;
```

```
$(document).ready(function(){
    ...  // 此处是页面DOM加载完成的入口
});
```

```
<script>
    $(function() {
        $('div').hide();
    })

    $(document).ready(function() {    //不常用
        $('div').hide();
    })
</script>
<div></div>
```

3.jQuery 对象

用 jQuery 获取的元素对象，就是 jQuery 对象。

jQuery 对象只能调用 jQuery 的方法函数，不能用 DOM 的属性和方法。

jQuery( ) 或 $( )

用 jQuery 获得的元素对象是伪数组形式。类似 querySelectorAll

```
<div></div>
<script>
    $('div');
    console.log($('div'));

    jQuery('div');
    console.log(jQuery('div'));
</script>
```

jQuery 对象转换为 DOM 对象

jQuery 里封装的是常用 JS 函数但不是全部，

所以要使用没封装的方法和属性时，需要转换为 DOM 对象。

因为 jQuery 获取的元素是伪数组形式，用序号指定元素。

$(元素)[序号]

$('div') [index]　　index 是索引号

$('div').get(index)　index 是索引号

```
<div>1</div>
<div>2</div>
<script>
    $('div')[0].style.backgroundColor = 'green';
    $('div')[1].style.display = 'none';
</script>
```

3.DOM 对象

用原生 JS 获取的元素对象，就是 DOM 对象。

DOM 对象只能用原生 JS 自带的 DOM 属性和方法，不能用 jQuery 的方法。

```
<div>1</div>
<div>2</div>
<script>
    var div = document.querySelector('div');
    var divs = document.querySelectorAll('div');
    console.log(div);
    console.log(divs);
</script>
```

```
<div>1</div>
NodeList(2) [div, div]
  0: div
  1: div
  length: 2
  __proto__: NodeList
```

querySelector 获得的是第一个该元素，

querySelectorAll 是以**伪数组形式**获取所有元素

```
<div>1</div>
<div>2</div>
<script>
    var div = document.querySelector('div');
    var divs = document.querySelectorAll('div');
    div.style.display = 'none';
    divs[1].style.display = 'none';
</script>
```

DOM 对象转换为 jQuery 对象

jQuery 的调用的方法写起来简单

若 DOM 对象想使用 jQuery 的方法，需要转换位 jQuery 对象。

$(通过 DOM 获得的 DOM 对象)

```
<div>1</div>
<div>2</div>
<script>
    var mydiv = document.querySelector('div');
    var mydivs = document.querySelectorAll('div');

    $(mydiv).hide();
    $(mydivs[1]).hide();
```

二．jQuery 获取元素

2. jQuery 筛选方法

类似 DOM 的节点操作。可以获得父子兄元素

语法	用法	说明
parent()	$("li").parent();	查找父级
children(selector)	$("ul").children("li")	相当于 $("ul>li")，最近一级（亲儿子）
find(selector)	$("ul").find("li");	相当于$("ul li"),后代选择器
siblings(selector)	$(".first").siblings("li");	查找兄弟节点，不包括自己本身
nextAll([expr])	$(".first").nextAll()	查找当前元素之后所有的同辈元素
prevtAll([expr])	$(".last").prevAll()	查找当前元素之前所有的同辈元素
hasClass(class)	$('div').hasClass("protected")	检查当前的元素是否含有某个特定的类，如果有，则返回true
eq(index)	$("li").eq(2);	相当于 $("li:eq(2)"),index 从0开始

```
<div class="father">father
    <div class="son">son1
        <div class="grandson">gd0</div>
        <div class="grandson gd1">gd1</div>
        <div class="grandson gd2">gd2</div>
        <div class="grandson">gd3</div>
        <span>span1</span>
    </div>
    <span>span2</span>
</div>
<script>
    $(".son").parent().hide();      // .father

    $(".father").children().hide();         //.father所有的子元素
    $(".father").children("span").hide(); //.father下的亲儿子span2

    $(".gd1").siblings().hide();  // 除.gd1自身以外的所有同级兄弟

    $(".gd2").prevAll().hide();   // 除.gd1自身以外的前面的所有同级元素
    $(".gd1").nextAll().hide();   // 除.gd1自身以外的后面的所有同级元素

    $(".grandson").eq(2).hide();  // 所有.grandson中序号是2的 常用

    console.log($(".grandson").hasClass("gd1"));   //有 true
    console.log($(".grandson").hasClass("papa"));  //没有 false
```

■ parents（“选择器”） 父级筛选器

元素 . parent（ ）只能获得最近一层的父元素（亲爸爸）

parent( ) 要一层一层向上找父元素。会很麻烦。

元素 . parents（ ）返回当前元素的全部祖先元素。

元素 . parents（“指定选择器” ）返回当前元素的全部祖先元素中的指定祖先元素。

■ find（“选择器”） 子级筛选器

元素 .children( ) 只能获得最近一层的所有子元素（所有亲儿子）

元素 .children(“指定选择器”）只能获得最近一层子元素中的指定元素。

children( ) 要一层一层向上找父元素。会很麻烦。

元素 .find（“指定选择器” ）返回当前元素的全部子元素中的指定子元素。

```
<body>
    <div class="grandpa">
        <div class="father">
            <div class="brother"></div>
            <ul>
                <li>1</li>
                <li>2</li>
                <li>3</li>
                <li>4</li>
            </ul>
        </div>
        <div class="uncle"></div>
    </div>
    <script>
        console.log($("li").parent());
        console.log($("li").parents());

        console.log($("li").parent().parent().parent());
        console.log($("li").parents(".grandpa"));


        console.log($(".grandpa").children());
        console.log($(".grandpa").children(".uncle"));

        console.log($(".grandpa").children().children().children());
        console.log($(".grandpa").find("li"));

        console.log($(".grandpa").find()); //空
    </script>
```

```
▸ S.fn.init [ul, prevObject: S.fn.init(4)]
▸ S.fn.init(5) [ul, div.father, div.grandpa, body, html, prevObject: S.fn.init(4)]

▸ S.fn.init [div.grandpa, prevObject: S.fn.init(1)]
▸ S.fn.init [div.grandpa, prevObject: S.fn.init(4)]

▸ S.fn.init(2) [div.father, div.uncle, prevObject: S.fn.init(1)]
▸ S.fn.init [div.uncle, prevObject: S.fn.init(1)]

▸ S.fn.init(4) [li, li, li, li, prevObject: S.fn.init(2)]
▸ S.fn.init(4) [li, li, li, li, prevObject: S.fn.init(1)]

▸ S.fn.init [prevObject: S.fn.init(1)]空的伪数组
```

案例 – 用 jQuery 做下拉菜单


```
<ul class="nav">
    <li>
        <a href="">大一</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
    <li>
        <a href="">大二</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
</ul>
<script>
    $(function() {
        $(".nav>li").mouseover(function() {
            $(this).children("ul").show()
        });
        $(".nav>li").mouseleave(function() {
            $(this).children("ul").hide()
        });
    });
</script>
```

利用 jQuery 的隐式迭代，给每个<li>绑定鼠标经过和鼠标离开事件。

利用 **$(this)** 指向当前触发事件的<li>的子元素里的

案例 – 用 jQuery 做简易 tab 栏切换


```
        .bgc {
            opacity: 1;
        }
    </style>
</head>
<body>
    <ul class="nav ">
        <li class="bgc">选项一</li>
        <li>选项二</li>
        <li>选项三</li>
        <li>选项四</li>
    </ul>
    <div class="content">
        <div style="display: block;">选项一 的内容</div>
        <div>选项二 的内容</div>
        <div>选项三 的内容</div>
        <div>选项四 的内容</div>
    </div>
    <script>
        $(function() {
            $(".nav li").click(function() {
                console.log($(this).css("opacity"));
                $(this).css("opacity", 0.8).siblings().css("opacity", 1)

                var index = $(this).index();
                $(".content div").eq(index).show().siblings().hide();
            });
        });
    </script>
```

1.利用隐式迭代，给每个<li>绑定点击事件。

2.获得当前被点击的元素的序号：

$(this).index( ); 获得当前元素的序号

3.在点击事件内，获得下面内容盒子的匹配序号的元素显示。

4.利用**链式编程**，把当前元素的其余兄弟都隐藏。

1. jQuery 选择器

```
$( "选择器" ) // 里面选择器直接写 CSS 选择器即可，但是要加引号
```

类似 CSS 与原生 JS。可以用 **$( ) 里直接写选择器** 来获得元素。

名称	用法	描述
ID选择器	$("#id")	获取指定ID的元素
全选选择器	$('*')	匹配所有元素
类选择器	$(".class")	获取同一类class的元素
标签选择器	$("div")	获取同一类标签的所有元素
并集选择器	$("div,p,li")	选取多个元素
交集选择器	$("li.current")	交集元素

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

    <div class="nav">nav</div>

    <div id="item">item</div>

    <div class="father">father
        <div class="son">son</div>
        <div class="brother">brother</div>
    </div>
    <script>
        $("div").hide();        //选中所有 div

        $(".nav").hide();       //选中所有 类名是nav的元素

        $("#item").hide();      //选中所有 id是item的元素

        $(".father .son").hide();    //选中类名father 下的 类名son

        $(".nav,#item").hide();      //选中所有类名nav 和 id名item
    </script>
</body>
```

3. jQuery 筛选选择器

类似 H5 的结构伪类选择器。可以获得指定元素。

语法	用法	描述
:first	$('li:first')	获取第一个li元素
:last	$('li:last')	获取最后一个li元素
:eq(index)	$("li:eq(2)")	获取到的li元素中，选择索引号为2的元素，索引号index从0开始。
:odd	$("li:odd")	获取到的li元素中，选择索引号为奇数的元素
:even	$("li:even")	获取到的li元素中，选择索引号为偶数的元素

```
<body>
    <div>0</div>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <script>
        $("div").hide();          //所有的 div

        $("div:first").hide();    //第一个 div

        $("div:last").hide();     //最后一个 div

        $("div:odd").hide();      //所有奇数序号的 div

        $("div:even").hide();     //所有偶数序号的 div

        $("div:eq(2)").hide();    //序号是2的 div
    </script>
</body>
```

● jQuery 隐式迭代

jQuery 自动给**同一元素做相同操作。**

自动遍历了所有匹配的元素

自动给所有匹配的元素调用了方法

自动给所有匹配的元素添加了事件

相当于省略了写 for 循环的一步

● jQuery 排他思想

```
<button>按钮0</button>
<button>按钮1</button>
<button>按钮2</button>
<script>
    //元素JS
    var btns = document.querySelectorAll('button');
    for (var i = 0; i < btns.length; i++) {
        btns[i].addEventListener('click', function() {
            for (var i = 0; i < btns.length; i++) {
                btns[i].style.backgroundColor = '';
            };
            this.style.backgroundColor = 'red';
        })
    }

    //jQuery
    $(function() {
        $("button").click(function() {
            $(this).css("background", "red");
            $(this).siblings().css("background", "");
        });
    })

    //jQuery 链式编程
    $(function() {
        $("button").click(function() {
            $(this).css("background", "red").siblings().css("background", "")
        });
    })
```

● jQuery 链式编程

用一条链子的形式，以当前元素位目标向后写。

```
<ul>
    <li class="first">
        <a href="">
            <img src="../images/60小 老夫子.png" alt="" class="small">
            <img src="../images/180大 老夫子.png" alt="" class="big">
        </a>
    </li>
    <li>
        <a href="">
            <img src="../images/60小 程咬金.png" alt="" class="small">
            <img src="../images/180大 程咬金.png" alt="" class="big">
        </a>
    </li>
</ul>
<script>
    $("li").mouseover(function() {
        $(this).stop().animate({width: 180}).find(".small").fadeOut().siblings(".big").fadeIn();


        $(this).siblings("li").stop().animate({width: 60}).find(".small").fadeIn().siblings(".big").fadeOut();


    });
</script>
```

如上：

this 自身做动画、this 的子元素 .samll 淡出、 **.samll 的兄弟 .big 淡入**

this 的兄弟 li 做动画、li 的子元素 .samll 淡入、 **.samll 的兄弟 .big 淡出**

三．jQuery 元素样式

1.单一样式

1.1 只写属性名，不写值

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

只获得属性的值，不会修改元素的样式

```
    <style>
        div {
            height: 100px;
            width: 100px;
            background-color: crimson;
        }
    </style>
</head>
<body>
    <div></div>
    <script>
        $(function() {
            console.log($("div").css("width"));
            console.log($("div").css("background"));
        });
    </script>
```

```
100px
rgb(220, 20, 60) none repeat scroll 0% 0% / auto padding-box border-box
```

1.2 写属性，也写值

```
$('div').css('属性', '值')
```

修改元素该属性的值。

属性要加 " "，**值也要加 " "。值是数字时省略 " " 和单位 px。**

```
<div></div>
<script>
    $(function() {
        $("div").css("height", "200px");
        $("div").css("width", 200);
        $("div").css("background", "green");
    });
</script>
```

2.多样式

对象形式

```
$(this).css({ "color":"white","font-size":"20px"});
```

要修改多个属性时，可以采用对象的形式。

对象的属性是 css 样式属性，不加引号。和原生 JS 一样，符合类名采用**驼峰命名法**

对象的属性值是 css 样式属性值，要加引号，数字可以省略引号和 px。

```
<div></div>
<script>
    $(function() {
        // $("div").css("height", "200px");
        // $("div").css("width", 200);
        // $("div").css("background", "green");

        $("div").css({
            height: 200,
            width:200,
            backgroundColor:"green"
        })
    });
</script>
```

3.类名 常用

要修改多个样式时，用对象形式写起来还是麻烦。

可以在 CSS 里预先写一个类，里面写好样式，然后 jQuery **添加**/调用该样式。

■ addClass( ) 添加类

```
$( "div" ).addClass("current");
```

```
    <style>
        div {
            height: 100px;
            width: 100px;
            border: 1px solid black
        }
        .red {
            background-color: crimson;
        }
    </style>
</head>
<body>
    <div></div>
    <script>
        $(function() {
            $("div").addClass("red");
        });
    </script>
</body>
```

与原生 JS 的 className 不同，**className** 是以行内样式**覆盖元素原来的类。**

jQuery 的 **addClass 不会覆盖**元素原来的类，只**是追加新的类（多类名）**

```
<div class="one two"></div>
```

■ removeClass( ) 删除类

```
$( "div" ).removeClass("current");
```

删除元素原来自带的类名。

```
<div class="red"></div>
<script>
    $(function() {
        $("div").removeClass("red")
    });
</script>
```

■ toggleClass( ) 切换类

```
$( "div" ).toggleClass("current");
```

若现在元素没有该属性，就添加该属性。

若现在元素有该属性的话，就删除该属性。

比如，可以实现一个元素多次点击时，属性来回切换。

```
    <style>
        div {
            height: 100px;
            width: 100px;
            border: 1px solid black;
        }
        .red {
            background-color: crimson;
        }
    </style>
</head>
<body>
    <div></div>
    <script>
        $(function() {
            // $("div").click(function() {
            //     $(this).addClass("red");
            // });
            // $("div").click(function() {
            //     $(this).removeClass("red");
            // });

            $("div").click(function() {
                $(this).toggleClass("red");
            });
        });
    </script>
```

案例 – 用增删类名改善简易 tab 栏切换

给$(this)添加类，给其所有兄弟元素移除类。

链式编程写在一起。

```
        .bgc {
            opacity: 1;
        }
    </style>
</head>
<body>
    <ul class="nav ">
        <li class="bgc">选项一</li>
        <li>选项二</li>
        <li>选项三</li>
        <li>选项四</li>
    </ul>
    <div class="content">
        <div style="display: block;">选项一 的内容</div>
        <div>选项二 的内容</div>
        <div>选项三 的内容</div>
        <div>选项四 的内容</div>
    </div>
    <script>
        $(function() {
            $(".nav li").click(function() {
                console.log($(this).css("opacity"));
                $(this).addClass("bgc").siblings().removeClass("bgc");

                var index = $(this).index();
                $(".content div").eq(index).show().siblings().hide();
            });
        });
    </script>
```

四．jQuery 元素效果

1.隐藏删除和切换

■ hide( ) 隐藏元素

类似原生 JS 里的 style.display= ‘none’

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

参数可以都省略不写。**一般不写参数使用无动画的 hide( )。**

Speed：隐藏元素花费的时间。一般不写。
Easing：隐藏元素的运动曲线。一般不写。
Fn：回调函数，元素隐藏后执行。一般不写。

```
<div></div>
<script>
    $(function() {
        $("div").click(function() {
            $(this).hide(1000, "linear", function() {
                alert("hi");
            })
        });
    });
```

实际上不用参数

■ show( ) 显示元素

类似原生 JS 里的 style.display= ‘block’

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

参数可以都省略不写。**一般不写参数使用无动画的 show( )。**

Speed：显示元素花费的时间。一般不写。
Easing：显示元素的运动曲线。一般不写。
Fn：回调函数，元素显示后执行。一般不写。

```
<div style="display: none"></div>
<button>show the element</button>
<script>
    $(function() {
        $("button").click(function() {
            $("div").show(1000, "linear", function() {
                alert("hi");
            })
        });
    });
```

实际上不用参数

■ toggle( ) 切换元素的显示隐藏

若元素隐藏，则让其显示 show( )

若元素显示，则让其隐藏 hide( )

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

参数可以都省略不写。**一般不写参数使用无动画的 toggle( )。**

Speed：显示/隐藏元素花费的时间。一般不写。
Easing：显示/隐藏元素的运动曲线。一般不写。
Fn：回调函数，元素显示/隐藏后执行。一般不写。

如下：每次点击按钮，元素切换的显示和隐藏

```
<div style="display: none"></div>
<button>show/hide the element</button>
<script>
    $(function() {
        $("button").click(function() {
            $("div").toggle(1000, "linear", function() {
                alert("hi");
            })
        });
    });
```

实际不用参数

案例 - 显示隐藏下拉菜单

```
<ul class="nav">
    <li>
        <a href="">大一</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
    <li>
        <a href="">大二</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
</ul>
<script>
    $(".nav li").hover(function() {
        $(this).children("ul").toggle();
    });
```

2.下拉上拉和切换

■ slideDown( ) 下拉显示元素

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

参数可以不写，也可以写参数实现动画效果。

Speed：下拉显示元素花费的时间。可以不写。

Easing：下拉显示元素的运动曲线。可以不写。

Fn：回调函数，元素下拉显示后执行。可以不写。

■ slideUp( ) 上拉隐藏元素

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

参数可以不写，也可以写参数实现动画效果。

Speed：上拉隐藏元素花费的时间。可以不写。

Easing：上拉隐藏元素的运动曲线。可以不写。

Fn：回调函数，元素上拉隐藏后执行。可以不写。

```
<ul class="nav">
    <li>
        <a href="">大一</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
    <li>
        <a href="">大二</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
</ul>
<script>
    $(".nav li").mouseover(function() {
        $(this).children("ul").slideDown(1000, "linear", function() {
            alert("hi");
        });
    });
    $(".nav li").mouseleave(function() {
        $(this).children("ul").slideUp(1000, "linear", function() {
            alert("hi");
        });
    });
```

■ slideToggle( ) 切换元素的下拉和上拉

若元素隐藏，则让其下拉显示 slideDown( )

若元素显示，则让其上拉隐藏 slideUp( )

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

参数可以不写，也可以写参数实现动画效果。
Speed：下拉显示/上拉隐藏元素花费的时间。可以不写。
Easing：下拉显示/上拉隐藏元素的运动曲线。可以不写。
Fn：回调函数，元素下拉显示/上拉隐藏后执行。可以不写。

案例 - 下拉上拉菜单

使用鼠标经过+离开的 hover( )事件

在使用切换下拉上拉效果的 **slideToggle( )**

因为是切换的动画效果，会产生**动画队列**。

需要用到 stop( )消除当前动画效果的前一个动画效果。

```
<ul class="nav">
    <li>
        <a href="">大一</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
    <li>
        <a href="">大二</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
</ul>
<script>
    $(".nav li").hover(function() {
        $(this).children("ul").stop().slideToggle(500)
    });
</script>
```

3.淡入淡出和切换

■ fadeIn( ) 淡入显示元素

比 show( )的显示效果多了慢慢淡入的动画。

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

参数可以不写，也可以写参数实现动画效果。

Speed：淡入显示元素花费的时间。可以不写。

Easing：淡入显示元素的运动曲线。可以不写。

Fn：回调函数，元素下淡入显示后执行。可以不写。

```
<button>show/hide the element</button>
<div style="display: none"></div>
<script>
    $(function() {
        $("button").click(function() {
            $("div").fadeIn(1000, "linear", function() {
                alert("hi");
            });
        });
    })
```

■ fadeOut( ) 淡出隐藏元素

比 hide( )的隐藏效果多了慢慢淡出的动画。

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

参数可以不写，也可以写参数实现动画效果。

Speed：淡出隐藏元素花费的时间。可以不写。

Easing：淡出隐藏元素的运动曲线。可以不写。

Fn：回调函数，元素淡出隐藏后执行。可以不写。

```
<button>show/hide the element</button>
<div></div>
<script>
    $(function() {
        $("button").click(function() {
            $("div").fadeOut(1000, "linear", function() {
                alert("hi");
            });
        });
    })
```

■ fadeToggle( ) 切换元素的淡入和淡出

若元素隐藏，则让其淡入显示 fadeIn( )

若元素显示，则让其淡出隐藏 fadeOut( )

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

参数可以不写，也可以写参数实现动画效果。

Speed：淡入显示/淡出隐藏元素花费的时间。可以不写。

Easing：淡入显示/淡出隐藏元素的运动曲线。可以不写。

Fn：回调函数，元素淡入显示/淡出隐藏后执行。可以不写。

案例 - 盒子缓慢消失和显示

使用 click 点击事件。每次点击<button>，盒子 box 就切换的显示和隐藏。

利用 **fadeToggle( )**切换淡入淡出效果。

因为是切换的动画效果，会产生**动画队列。**

需要用到 stop( )消除当前动画效果的前一个动画效果。

```
<button>show/hide the box</button>
<div></div>
<script>
    $(function() {
        $("button").click(function() {
            $("div").stop().fadeToggle(1000);
        });
    })
</script>
```

■ fadeTo( ) 渐进调整元素透明度

可理解为 CSS 属性 opacity 的动画版。

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

Speed：调整元素透明度花费的时间。**必须写。**

Opacity：和 CSS 一样。调整元素透明度，值为 1~0。**必须写**

Easing：调整元素透明度的运动曲线。可以不写。

Fn：回调函数，调整元素透明度后执行。可以不写。

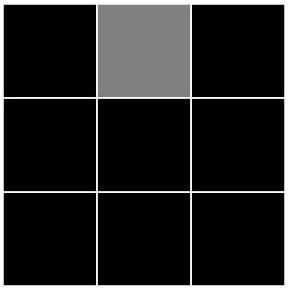
```
<button>show/hide the box</button>
<div></div>
<div></div>
<script>
    $(function() {
        $("button").click(function() {
            $("div").fadeTo(1000, 0.5);

            $("div").eq(1).fadeTo(1000, 0.3, "linear", function() {
                alert("hi");
            });
        });
    });
</script>
```

案例 - 切换高亮显示

用 hover(函数，函数) 实现鼠标经过的效果切换。

要用 stop( ) 消除动画队列

```
<ul>
    <li></li> <li></li> <li></li>
    <li></li> <li></li> <li></li>
    <li></li> <li></li> <li></li>
</ul>
<script>
    $(function() {
        $("li").hover(function() {
            $(this).stop().fadeTo(300, 0.5);
        }, function() {
            $(this).stop().fadeTo(300, 1)
        });
    });
</script>
```

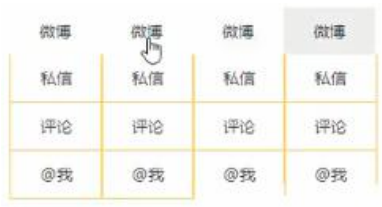


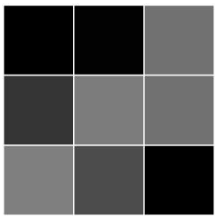
● stop( ) 停止动画队列

动画一旦执行就一定要执行完。

若是短时间内多次触发，就会造成多个**动画/效果**的**排队执行**。影响美观。

尤其是鼠标经过事件 hover 最明显。如下





可以使用 stop 来停止一个动画的上一个动画的执行。停止动画的排队执行。

```
stop()
```

是停止该动画的上一个动画，要**写在该动画/效果的链式编程的前面。**

案例 - 用 stop 停止下拉菜单的动画排队

```
<ul class="nav">
    <li>
        <a href="">大一</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
    <li>
        <a href="">大二</a>
        <ul>
            <li>小一</li>
            <li>小二</li>
            <li>小三</li>
        </ul>
    </li>
</ul>
<script>
    $(".nav li").hover(function() {
        $(this).children("ul").stop().slideToggle(500)
    });
</script>
```

4.自定义动画　animate

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

Params：要修改的属性样式，以**对象方式**传递，**必须写。**

Speed：动画花费的时间。可以不写。

Easing：动画的运动曲线。可以不写。

Fn：回调函数，动画结束后执行。可以不写。

css( ) 和 addClass( ) 是一下子瞬间修改样式

动画 animate( ) 在修改的过程显示时间和运动曲线。并且动画的属性是同时改变。如下：

案例 - 折叠卡片/手风琴效果

布局是一个盒子里用定位叠放两个图。

然后鼠标经过，盒子和两个图片分别做动画和显示隐藏


```
<ul>
    <li>
        <a href="">
            <img src="" alt="">
            <img src="" alt="">
        </a>
    </li>
</ul>
```

```
<style>
    li {
        list-style: none;
        position: relative;
        float: left;
        margin-right: 2px;
        height: 60px;
        width: 60px;
    }
    .first {
        width: 180px;
    }
    .first .small {
        display: none;
    }
    .first .big {
        display: block;
    }
    .small {
        position: absolute;
    }
    .big {
        display: none;
    }
</style>
```

```
<ul>
    <li class="first">
        <a href="">
            <img src="../images/60小 老夫子.png" alt="" class="small">
            <img src="../images/180大 老夫子.png" alt="" class="big">
        </a>
    </li>
    <li>
        <a href="">
            <img src="../images/60小 程咬金.png" alt="" class="small">
            <img src="../images/180大 程咬金.png" alt="" class="big">
        </a>
    </li>
    <li>
        <a href="">
            <img src="../images/60小 兰陵王.png" alt="" class="small">
            <img src="../images/180大 兰陵王.png" alt="" class="big">
        </a>
    </li>
</ul>
<script>
    $("li").mouseover(function() {
        $(this).stop().animate({
            width: 180
        }).find(".small").fadeOut().siblings(".big").fadeIn();

        $(this).siblings("li").stop().animate({
            width: 60
        }).find(".small").fadeIn().siblings(".big").fadeOut();
    });
</script>
```

五．jQuery 元素属性

1．获得/修改元素 固有属性

■ prop( ) 获得固有属性的值

类似原生 JS 的 **element. 属性**

```
prop("属性")
```

只写属性不写值，是获得该属性的值。属性名加引号。

```
<a href="http://www.baidu.com" title="百度">跳转百度</a>

<input type="checkbox" name="" id="" checked>
<input type="text">
<script>
    $(function() {
        console.log($("a").prop("href"));

        $("input").eq(0).change(function() {
            console.log($(this).prop("checked"));
        });

        $("input").eq(1).change(function() {
            console.log($(this).prop("value"));
        });
    });
</script>
```


```
http://www.baidu.com/
false
true
hi
```

■ prop( ) 修改固有属性的值

类似原生 JS 的 **element. 属性= ‘值’**

```
prop("属性","属性值")
```

写了属性又写了属性值，是把该属性修改位该值。

属性名加引号，属性名和属性值用逗号隔开。

```
<input type="checkbox" name="" id="" checked>
<input type="text">
<script>
    $(function() {
        $("input").eq(0).prop("checked", false);

        $("input").eq(1).prop("value", "hi")
    });
</script>
```

案例 - 复选框和单选框的全选/全不选


```
//全选按钮
$(".cbAll").change(function() {
    $(".cbBtns,.cbAll").prop("checked", $(this).prop("checked"))
});

//单选框控制多选框
$(".cbBtns").change(function() {
    if ($(".cbBtns:checked").length === $(".cbBtns").length) {
        $(".cbAll").prop("checked", true)
    } else {
        $(".cbAll").prop("checked", false)
    };
});
```

1. 复选框控制单选框

利用单击复选框后，所有单选框的 checked 属性跟随复选框的 checked 属性。

上下两个全选框，类名相同。

点击任何一个都会修改所有单选框的 checked 属性并一同修改另一个复选框。

用并集选择器。

2. 单选框控制复选框

判断获得所有**被选中的单选框的个数** 是否等于 **所有单选框的个数。**

$(" 元素 ")是以**数组形式**获得所有该元素，该**元素的个数**就是**数组的长度 length**

■ :checked 筛选选择器

$(" input 元素：checked ") 可以筛选出并获得**被选中的表单元素。**

2. 获得/修改元素 自定义属性

■ attr( ) 获得自定义属性

类似原生 JS 的 **getAttribute('属性')**

```
attr("属性")
```

只写属性不写值，是获得该自定义属性。自定义属性名加引号。

也可获得 **H5 新增的 data- 开头**的自定义属性

```
<div index="1" data-index="22"></div>
<script>
    $(function() {
        console.log($("div").attr("index"));
        console.log($("div").attr("data-index"));
    })
</script>
```

```
1
22
```

■ attr( ) 修改自定义属性

类似原生 JS 的 **setAttribute('属性' , '值')**

```
attr("属性","属性值")
```

写了属性名又写值，是修改该自定义属性的值。

属性名加引号，属性名和属性值用逗号隔开。

也可修改 **H5 新增的 data- 开头**的自定义属性。

```
<div index="1" data-index="22"></div>
<script>
    $(function() {
        $("div").attr("index", 2)
        console.log($("div").attr("index"));

        $("div").attr("data-index", 33)
        console.log($("div").attr("data-index"));
    })
</script>
```

```
2
33
```

3. 获得/添加元素 数据缓存属性

■ data( ) 获得 H5 新增 data-开头属性

```
date("name")
```

用 attr( )可以获取普通自定义属性，也可获取 data-开头的自定义属性。

获得的都是字符串，

用 data( )只能获取 data-开头的自定义属性。

获得的会看数据本身返回字符串 或数字型。

```
<div index="1" data-index="22"></div>
<script>
    $(function() {
        console.log($("div").data("index"));

        console.log($("div").attr("data-index"));
    })
</script>
```

```
22
22
```

■ data( ) 数据缓存 添加属性

```
data("name","value")
```

给元素的缓存中添加属性，**添加的该属性不会显示在 HTML 结构中。**

在缓存中添加的该元素**只能再被 data( )获取。**

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

属性名省略 data-开头。

```
<div></div>
<script>
    $(function() {
        $("div").data("index", 33)
        console.log($("div").data("index"));

        console.log($("div").attr("data-index"));
        console.log($("div").attr("index"));

        console.log($("div").data("index"));
    })
</script>
```

```
33
undefined
undefined
33
```

六．jQuery 元素内容

1.获得/修改元素 HTML 结构+内容

■ html( ) 获得结构+文本内容

类似原生 JS 的 **innerHTML**

```
html()          // 获取元素的内容
```

括号里不写值，是获取该元素的**保留 HTML 结构**的内容。

以字符串形式获取 HTML 标签+文本内容。

```
<div class="father">father
    <div class="son">son<br>123
    </div>
</div>

<script>
    console.log($(".father").html());
</script>
```

```
father
        <div class="son">son<br>123
        </div>
```

■ html( ) 添加/修改结构+文本内容

类似原生 JS 的 **innerHTML= ‘标签+文本内容’**

```
html("内容") // 设置元素的内容
```

括号里要写 HTML 标签和文本内容，加引号。

括号里写内容是给该元素添加 / 修改（覆盖）内容。

新内容会覆盖原先的结构内容。

```
<div class="father">father
    <div class="son">son<br>123
    </div>
</div>
<script>
    $(".father").html("<span>新内容</span>")
</script>
```

```
<div class="father">
  <span>新内容</span>
</div>
```

2.获得/修改元素 文本内容

■ text( ) 仅获得文本内容

类似原生 JS 的 **innerText**

```
text()                    // 获取元素的文本内容
```

括号里不写值，是以字符串形式获取文本内容。

会分开显示父级和子级 的内容，但是不会返回 HTML 标签，只返回文本。

```
<div class="father">father
    <div class="son">son<br>123
    </div>
</div>
<script>
    console.log($(".father").text());
</script>
```

father
son123

换行标签没显示

■ text( ) 添加/修改文本内容

类似原生 JS 的 **innerText=”内容“**

```
text("文本内容") // 设置元素的文本内容
```

括号里写内容是给该元素添加 / 修改（覆盖）内容。

新内容会覆盖原先的结构内容。

```
<div class="father">father
    <div class="son">son<br>123
    </div>
</div>
<script>
    $(".father").text("新内容");
    console.log($(".father").text());
</script>
```

新内容

3.获得/修改 input 表单 内容

■ val( ) 获得表单内容

类似原生 JS 的 **表单. value**

括号里不写值，是获取该表单的内容。

```
<input type="text" name="" id="" value="请输入内容">
<script>
    console.log($("input").val());

    $("input").change(function() {
        console.log($("input").val());
    })
</script>
```

请输入内容
12
asd

■ val( ) 添加/修改表单内容

类似原生 JS 的 **表单. value= '值'**

括号里写值，是修改/添加该表单的内容。

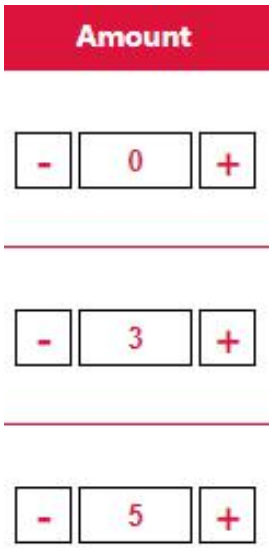
新内容会覆盖原先的结构内容。

```
<input type="text" name="" id="" value="请输入内容">
<script>
    $("input").val("新内容");
    console.log($("input").val());
</script>
```

新内容

案例 - 增添商品的数量

```
<div class="i_amount">
    <i class="subtract">-</i>
    <input type="text" name="" id="" class="item_input" value="1">
    <i class="plus"> +</i>
</div>
```




```
// 输入框+-按钮
//减号按钮
$(".subtract").click(function() {
    var value = $(this).siblings(".item_input").val();
    value--;
    if (value < 0) {
        return false;
    } else {
        $(this).siblings(".item_input").val(value);
    };
});
//加号按钮
$(".plus").click(function() {
    var value = $(this).siblings(".item_input").val();
    value++;
    $(this).siblings(".item_input").val(value);
});
```

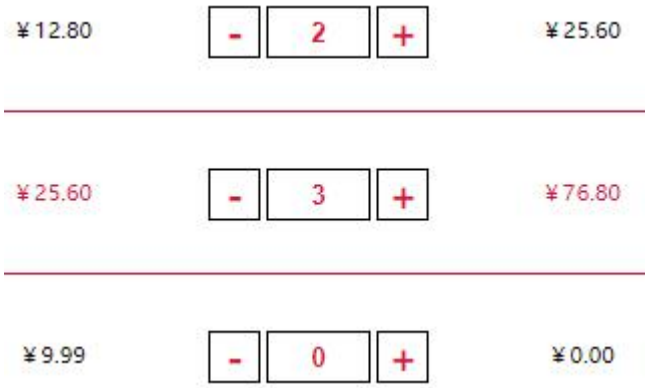
点击表单两侧的 + - 会增减表单的内容。

注意商品的个数不能减到 0 以下。

注意 + - 增减的是**他们的兄弟元素** input 表单的内容。

return false 是直接结束判断。

案例 - 根据修改的商品数量修改总价



■ toFixed(number) 保留小数点后位数

浮点的数字型数据 . toFixed(2) 保留小数点后第 2 位数字。

利用 change 表单变化事件，在表单内数字（商品个数）变化时，

获得表单的内容（个数）和单价（字符串变为数字型并去掉 ¥ 字符），相乘

再拼接加上 ¥ 字符后，赋值给标签内容。

注意分清 this 指向、父级元素、父级的兄弟元素。

```
//减号按钮
$(".subtract").click(function() {
    var value = $(this).siblings(".item_input").val();
    value--;
    if (value < 0) {
        return false;
    } else {
        $(this).siblings(".item_input").val(value);
    };

    //Price 和 sum 计算
    var price = parseFloat($(this).parent().siblings(".i_price").text().substr(1));
    var sum = (price * value).toFixed(2);
    $(this).parent().siblings(".i_sum").text("¥" + sum);
});

//加号按钮
$(".plus").click(function() {
    var value = $(this).siblings(".item_input").val();
    value++;
    $(this).siblings(".item_input").val(value);

    //Price 和 sum 计算
    var price = parseFloat($(this).parent().siblings(".i_price").text().substr(1));
    var sum = (price * value).toFixed(2);
    $(this).parent().siblings(".i_sum").text("¥" + sum);
});
```

七．jQuery 操作元素

1.遍历元素

jQuery 虽然还有隐式迭代，但是**隐式迭代**是自动遍历**同类元素执行相同操作**。

jQuery 的遍历元素，可以对同类元素进行不同操作，隐式迭代无法实现。

■ each( ) 遍历 DOM 元素

主要用作 DOM 处理，比如修改元素的样式、获取元素的内容等。

```
$("div").each(function (index, domEle) { xxx; })
```

index：是每个元素的序号，可以自定义名为 index 或 i

domEle：是遍历的每个 DOM 元素。可以自定义名

是 DOM 对象，不能用 jQuery 的方法函数。

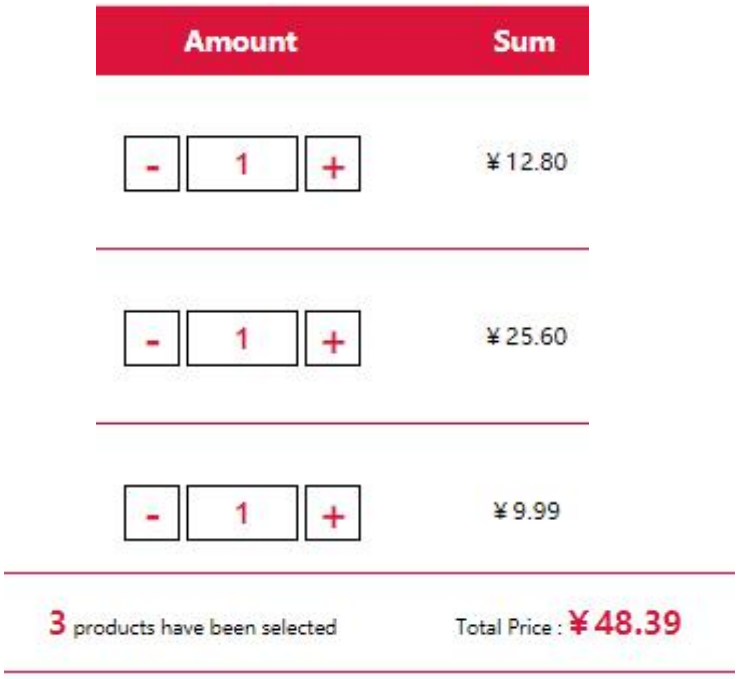
若要对每个对象使用 jQuery 方法，要转换为 jQuery 对象 **$(domEle)**

function：是回调函数。每遍历一个元素就执行一次回调函数。。

```
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
<script>
    $("div").each(function(i, domEle) {
        console.log(i);
        console.log(domEle);
    });
                                    利用序号 i

    var color = ["red", "blue", "green", "pink"];
    $("div").each(function(i, domEle) {
        $(domEle).css("color", color[i])
    });
                        把DOM对象domEle转换为
                        jQuery对象
    var sum = 0;
    var str = [];
    $("div").each(function(i, domEle) {
        str += $(domEle).text() + ",";
        sum += parseInt($(domEle).text());
    });
    console.log(sum);
    console.log(str);
</script>
```

```
0
  <div style="color: red;">1</div>
1
  <div style="color: blue;">2</div>
2
  <div style="color: green;">3</div>
3
  <div style="color: pink;">4</div>
10
1,2,3,4,
```

案例 - 商品总数量和总价



在+ - 计算完商品数量和价格总计后，调用该封装的函数。

分别遍历相加商品的 个数和单价 ，然后分别相加，最后把结果分别赋值底部的标签内。

```
<li class="car_item">
    <a href="javascript:;">
        <div class="i_checkBox">
            <input type="checkbox" name="" id="" class="cbBtns">
        </div>
        <div class="i_gods">Products</div>
        <div class="i_price">￥9.99</div>
        <div class="i_amount">
            <i class="subtract">-</i>
            <input type="text" name="" id="" class="item_input" value="1">
            <i class="plus"> +</i>
        </div>
        <div class="i_sum">￥9.99</div>
        <div class="i_option"><a href="javascript:;">Delete</a></div>
    </a>
</li>
```

```
<div class="car_footerBar">
    <div class="f_checkBox">
        <input type="checkbox" name="" id="" class="cbAll"><em> Select All</em>
    </div>
    <div class="f_delete"><a href="javascript:;">Delete Selected Products</a></div>
    <div class="f_clear"><a href="javascript:;">Clear the Shopping Car</a></div>
    <div class="f_btn">
        <a href="javascript:;">BUY</a>
    </div>
    <div class="f_sum">Total Price :<i> ￥00.00</i></div>
    <div class="f-amount"><i>3</i> products have been selected</div>
</div>
```

```
// 页面刷新时，也要计算商品数量和价格总计 赋值底部
// 必须写在 + -计算的最前面
fn();

// 直接修改输入框内容，计算并修改商品数量和价格总计
$(".item_input").change(function() {
    var n = parseInt($(this).val());
    var price = parseFloat($(this).parent().siblings(".i_price").text().substr(1));
    var sum = (price * n).toFixed(2);
    $(this).parent().siblings(".i_sum").text("¥" + sum);
    //每商品数量和价格总计 计算结束后，赋值底部
    fn();
});

// 输入框+-按钮
//减号按钮
$(".subtract").click(function() {
    var value = $(this).siblings(".item_input").val();
    value--;
    if (value < 0) {
        return false;
    } else {
        $(this).siblings(".item_input").val(value);
    };
    //Price 和 sum 计算
    var price = parseFloat($(this).parent().siblings(".i_price").text().substr(1));
    var sum = (price * value).toFixed(2);
    $(this).parent().siblings(".i_sum").text("¥" + sum);
    //每商品数量和价格总计 计算结束后，赋值底部
    fn();
});

//加号按钮
$(".plus").click(function() {
    var value = $(this).siblings(".item_input").val();
    value++;
    $(this).siblings(".item_input").val(value);
    //Price 和 sum 计算
    var price = parseFloat($(this).parent().siblings(".i_price").text().substr(1));
    var sum = (price * value).toFixed(2);
    $(this).parent().siblings(".i_sum").text("¥" + sum);
    //每商品数量和价格总计 计算结束后，赋值底部
    fn();
});
```

■ $.each( ) 遍历数据

主要用于处理数据。**可以遍历 DOM 元素、对象、数组。**

```
$.each(object , function (index, element) { xxx; } )
```

object ： 是遍历对象，要遍历谁就写谁。

index：是要遍历元素的索引号。索引号名可以自定义。

element ： 是遍历的内容。

```
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
<script>
    //遍历DOM元素
    $.each($("div"), function(i, e) {
        console.log(i);
        console.log(e);
    })
</script>
<script>
    //遍历数组
    var arr = ["red", "blue", "green", "pink"];
    $.each(arr, function(i, e) {
        console.log(i);
        console.log(e);
    });

    //遍历对象
    var obj = {
        name: "Andy",
        sex: "male",
        age: 18
    }
    $.each(obj, function(i, e) {
        console.log(i);
        console.log(e);
    });
</script>
```

```
0
  <div>1</div>
1
  <div>2</div>
2
  <div>3</div>
3
  <div>4</div>
```

```
0
red
1
blue
2
green
3
pink
```

```
name
Andy
sex
male
age
18
```

2.创建元素

■ $(“ 标签+文本内容 “)

相当于原生 JS 的节点操作 **document.createElement（‘标签名’）**和 **innerHTML**

```
$("<li></li>");
```

和原生 JS 一样，只创建元素不添加进某个父元素的话，新建元素不会再页面中显示。

```
<body>
    <div class="father">father</div>
    <script>
        var son = $("<div class='son'>son</div>");
    </script>
</body>
```


```
<body>
  <div class="father">father</div>
  <script>
          var son = $("<div class='son'>son</div>");
      </script>
</body>
```

3.内部添加元素 做子元素

■ append( ) 添加到元素内部最后

相当于原生 JS 的节点操作 **父元素.appendChild（ 子节点 ）**

```
element.append("内容")
```

放到目标元素的**内部的最后面**

■ prepend( ) 添加到元素内部最前

```
element.prepend("内容")
```

放到目标元素的**内部的最前面**

```
<ul>
    <li>1</li>
    <li>2</li>
</ul>
<script>
    var li_3 = $("<li>3</li>");
    var li_0 = $("<li>0</li>");

    //添加到父元素内部的后面
    $("ul").append(li_3);

    //添加到父元素内部的前面
    $("ul").prepend(li_0);

    console.log($("ul li"));
</script>
```

```
<ul>
  <li>0</li>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

```
▼S.fn.init(4) [li, li, li, li, prevObject: S.fn.init(1)]
  ▶0: li
  ▶1: li
  ▶2: li
  ▶3: li
    length: 4
  ▶prevObject: S.fn.init [document]
  ▶__proto__: Object(0)
```

4.外部添加元素 做兄弟元素

■ after( ) 添加到元素外部最后

```
element.after("内容")      // 把内容放入目标元素后面
```

放到父元素的前面，作为父元素的兄弟元素

■ before( ) 添加到元素外部最前

```
element.before("内容")  // 把内容放入目标元素前面
```

放到父元素的后面，作为父元素的兄弟元素

```
<ul>
    <li>1</li>
    <li>2</li>
</ul>
<script>
    var li_3 = $("<li>3</li>");
    var li_0 = $("<li>0</li>");

    //添加到父元素外部的后面
    $("ul").after(li_3);

    //添加到父元素外部的前面
    $("ul").before(li_0);

    console.log($("ul li"));
</script>
```

```
<li>0</li>
<ul>
  <li>1</li>
  <li>2</li>
</ul>
<li>3</li>
```

```
▼S.fn.init(2) [li, li, prevObject: S.fn.init(1)]
  ▶0: li
  ▶1: li
    length: 2
  ▶prevObject: S.fn.init [document]
  ▶__proto__: Object(0)
```

5.删除元素

■ remove( )　删除元素自身

```
element.remove()  // 删除匹配的元素（本身）
```

删除元素自身+该元素的内容（子元素）。

■ empty( )　删除元素的内容

```
element.empty()  // 删除匹配的元素集合中所有的子节点
```

仅删除元素的内容（子元素），**保留该元素 HTML 结构**，页面中仍可见。

■ html(“”)　删除元素的内容

```
element.html("") // 清空匹配的元素内容
```

就是用 html( )给元素内容赋值为空。

仅删除元素的内容（子元素），**保留该元素 HTML 结构**，页面中仍可见。

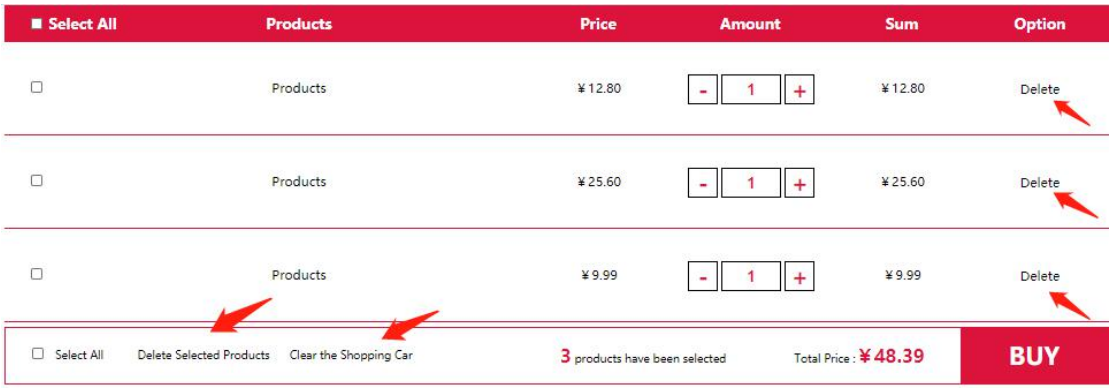
```
<ul>我是ul
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
<script>
    //删除元素自身+其子元素（删除结构）
    $("ul").remove();


    //仅删除元素的元素（保留结构）
    $("ul").empty();


    //删除元素自身内容+其子元素（保留结构）
    $("ul").html("")
</script>
```

```
<body>
▶<script>…</script>
</body>
```

```
<body>
  <ul></ul>
▶<script>…</script>
</body>
```

```
<body>
  <ul></ul>
▶<script>…</script>
</body>
```

案例 - 删除商品




```
<li class="car_item">
    <a href="javascript:;">
        <div class="i_checkBox">
            <input type="checkbox" name="" id="" class="cbBtns">
        </div>
        <div class="i_gods">Products</div>
        <div class="i_price">￥25.60</div>
        <div class="i_amount">
            <i class="subtract">-</i>
            <input type="text" name="" id="" class="item_input" value="1">
            <i class="plus"> +</i>
        </div>
        <div class="i_sum">￥25.60</div>
        <div class="i_option"><a href="javascript:;">Delete</a></div>
    </a>
</li>
```

```
<div class="f_delete"><a href="javascript:;">Delete Selected Products</a></div>
<div class="f_clear"><a href="javascript:;">Clear the Shopping Car</a></div>
```

```
//  删除按钮
//  items 删除按钮
$(".i_option").click(function() {
    $(this).parents(".car_item").remove();
    //等商品数量和价格总计 计算结束后，赋值底部
    fn();
});

//  clear car 删除按钮
$(".f_clear").click(function() {
    $(".car_item").remove();

    //等商品数量和价格总计 计算结束后，赋值底部
    //该例子因为标签自带value=1，此处不能调用函数
    var sumAmount = 0;
    var sumPrice = 0;
    $(".f-amount i").text(sumAmount);
    $(".f_sum i").text("￥" + sumPrice.toFixed(2));
});

//  删除 checked的按钮
$(".f_delete").click(function() {
    $(".cbBtns:checked").parents(".car_item").remove();
    //等商品数量和价格总计 计算结束后，赋值底部
    fn();
});
```

6.拷贝/合并 对象

■ $.extend( ) 拷贝对象

```
$.extend([deep], target, object1, [objectN])
```

$.extend(深拷贝/浅拷贝 拷贝给谁， 拷贝谁)

true 深拷贝，不写的话**默认浅拷贝。**

1.拷贝给的对象为空

被拷贝的对象会完全复制给该空对象

```
var obj_2 = {
    name: "red",
    age: 25
};
var obj = {};

console.log(obj_2);
console.log(obj);

$.extend(obj, obj_2)
console.log(obj);
```

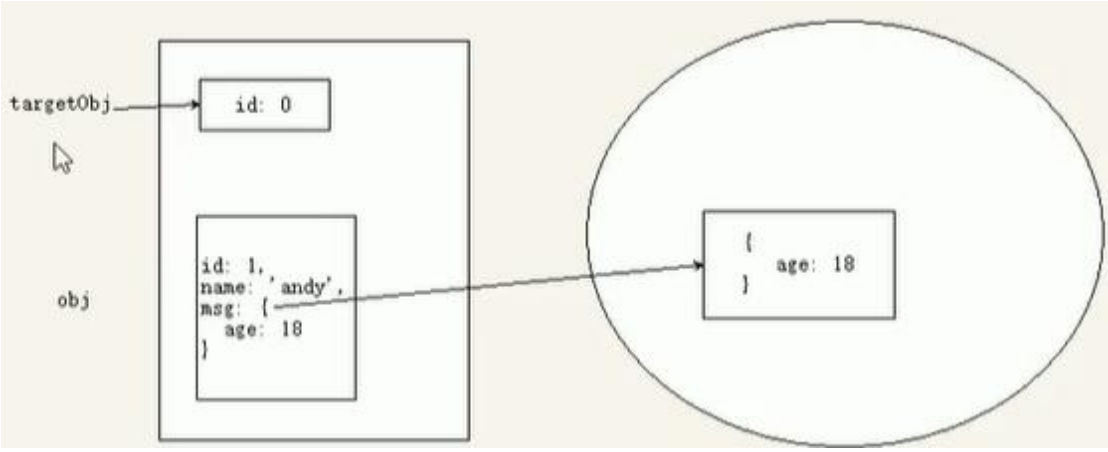
```
▼{name: "red", age: 25}
    age: 25
    name: "red"
  ▶__proto__: Object
▶{}
▼{name: "red", age: 25}
    age: 25
    name: "red"
  ▶__proto__: Object
```

2.拷贝给的对象原有内容

被拷贝的对象会**复制**给该拷贝给的对象里**没有的内容**，并**覆盖** 拷贝给的对象里**相同的内容。**

```
var obj_1 = {
    name: "Andy",
    age: 18,
    sex: "male"
};
var obj_2 = {
    name: "red",
    age: 25
};

console.log(obj_2);
console.log(obj_1);

$.extend(obj_1, obj_2);
console.log(obj_1);
```

```
▼{name: "red", age: 25}
    age: 25
    name: "red"
  ▶__proto__: Object
▼{name: "Andy", age: 18, sex: "male"}
    age: 25
    name: "red"
    sex: "male"
  ▶__proto__: Object
▼{name: "red", age: 25, sex: "male"}
    age: 25
    name: "red"
    sex: "male"
  ▶__proto__: Object
```



3.浅拷贝

默认浅拷贝。

会把被拷贝对象的**复杂数据（函数）的地址** 复制过去，

若在拷贝给的对象中修改该复杂数据，会沿着地址**直接修改被拷贝对象的复杂数据**

```
var obj_1 = {};
var obj_2 = {
    name: "red",
    age: 25,
    skill: {
        dance: "moonwalk"
    }
};
$.extend(obj_1, obj_2);
obj_1.skill.dance = "rubitdance"
console.log(obj_2);
```

```
▼{name: "red", age: 25, skill: {…}}
    age: 25
    name: "red"
  ▶ skill: {dance: "rubitdance"}
  ▶ __proto__: Object
```

4.深拷贝

true 深拷贝，

不拷贝地址，是直接把被拷贝对象的复杂数据类型复制一份，然后给拷贝给的对象。

在拷贝给的对象中修改该复杂数据，只是修改自己，并**不会修改被拷贝对象的复杂数据**

```
var obj_1 = {};
var obj_2 = {
    name: "red",
    age: 25,
    skill: {
        dance: "moonwalk"
    }
};
$.extend(true, obj_1, obj_2);
obj_1.skill.dance = "rubitdance"
console.log(obj_2);
```

```
▼{name: "red", age: 25, skill: {…}}
    age: 25
    name: "red"
  ▶ skill: {dance: "moonwalk"}
  ▶ __proto__: Object
```

八．jQuery 事件

1.事件注册/解绑

元素.事件(执行程序)　单个事件注册

事件名的写法和原生 JS 的类似，

比如 mouseover、mouseout、click、keydown、keyup、blur、scroll 等等

```
element.事件(function(){})
```

```
$("div").click(function(){  事件处理程序 })
```

缺点是一次只能注册一个事件。若同一元素注册多个事件会麻烦，

```
<div></div>
<script>
    $("div").click(function() {
        $(this).css({
            backgroundColor: "yellow"
        });
    });

    $("div").mouseover(function() {
        $(this).css({
            backgroundColor: "blue"
        });
    });

    $("div").mouseleave(function() {
        $(this).css({
            backgroundColor: "red"
        });
    });
</script>
```

■ on( ) 注册事件 推荐

```
element.on(events,[selector],fn)
```

可以注册单一事件、同一元素注册多个事件、操作匹配元素的子元素

1.on( ) 注册单个事件

元素 . on(function( ){ })

```
<div></div>
<script>
    $("div").on("click", function() {
        $(this).css({
            backgroundColor: "yellow"
        });
    })
</script>
```

2.on( ) 注册多个事件

元素 . on({ function( ){ } ， function( ){ } })

以对象格式写多个事件，看作对象的方法，用逗号隔开。

事件名不加引号，键值对的形式写事件名和处理函数

若不同事件，只是切换同一样式的话，

元素 . on(" 事件 1 事件 2 " ， function( ){ })

```
<body>
    <div></div>
    <script>
        $("div").on({

            mouseenter: function() {
                $(this).css("background", "green")
            },
            click: function() {
                $(this).css("background", "yellow");
            },
            mouseleave: function() {
                $(this).css("background", "blue");
            }
        })
    </script>
```

```
        .bg {
            background-color: crimson;
        }
    </style>
</head>
<body>
    <div></div>
    <script>
        $("div").on("mouseover mouseleave", function() {
            $(this).toggleClass("bg")
        });
    </script>
```

3.on( ) 实现事件委派(事件委托)

利用事件冒泡，把事件绑定到会触发事件的子元素的父元素上。

触发事件元素的父元素. on(“事件”，” 触发事件的元素 “)

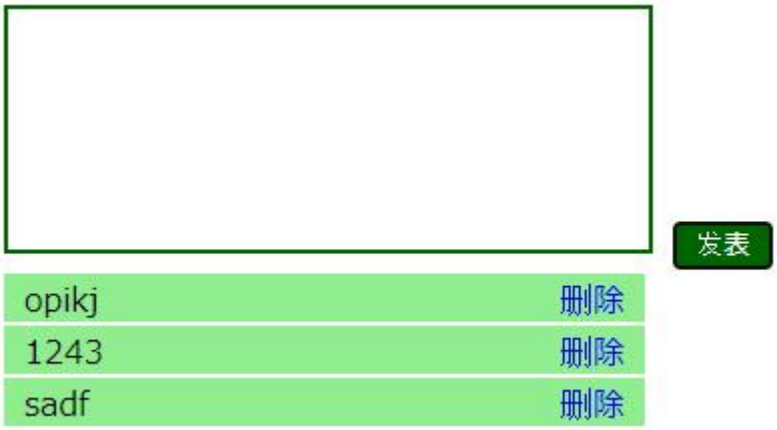
```
<ul>
    <li></li>
    <li></li>
    <li></li>
</ul>
<script>
    $("ul").on("click", "li", function() {
        $(this).css("background", "yellow");
    })
</script>
```

3.on( ) 给动态生成的元素绑定事件

当前不存在但将来后面会被创建添加的元素，要绑定事件只能用 on( ).

```
<div class="father"></div>
<script>
    //   $("div span").click(function() {          注册时没有
    //       $(this).css("background", "yellow");
    //   });

    // $("div span").on("click", function() {      注册时没有
    //     $(this).css("background", "yellow");
    // });

    $("div").on("click", "span", function() {
        $(this).css("background", "yellow");
    });
    var span = $("<span class='son'>123</span>");
    $("div").append(span);
</script>
```

案例 - 简易留言板




```
<div class="box">
    <textarea></textarea>
    <button>发表</button>
    <ul>
    </ul>
</div>
<script>
    $("button").on("click", function() {
        // jQuery3 .5 以上 $.trim(jQuery对象) 被弃用，可用原生JS的 DOM对象/str.trim()
        if ($("textarea").val().trim() == "") {
            alert("不能输入空白")
        } else {
            var value = $("textarea").val()

            var li = $("<li></li>")
            li.html(value + "<a href='javascript:;'>删除</a>");
            $("ul").prepend(li);
            li.slideDown();
            $("textarea").val(" ");

            // 利用事件冒泡，给所有a的父亲添加事件
            //又因为li是动态生成的，只能用 on( )
            $("ul").on("click", "a", function() {
                $(this).parent().remove();
            })
        }
    });
</script>
```

■ off( ) 解绑事件

可以移除 on( ) 绑定的事件。

元素. off( ) 解绑该元素的所有事件

元素. off(” 事件 “) 解绑该元素的指定事件

```
<div></div>
<script>
    $("div").on({
        mouseenter: function() {
            $(this).css("background", "green")
        },
        click: function() {
            $(this).css("background", "yellow");
        },
        mouseleave: function() {
            $(this).css("background", "blue");
        }
    });

    $("div").off();             //移除所有事件
    $("div").off("click");      //仅移除click事件
</script>
```

off( ) 解绑 on( )的事件委托

绑定事件的元素. off(” 事件 “,” 触发事件的对象”)

```
<ul>
    <li></li>
    <li></li>
    <li></li>
</ul>
<script>
    $("ul").on("click", "li", function() {
        $(this).css("background", "yellow");
    })
    $("ul").off("click", "li")
</script>
```

■ one( ) 仅执行一次事件

one( ) 绑定的事件只能触发一次，不需要解绑。

使用方法和 on( )类似。

2.自动触发事件

比如 轮播图再点击左右箭头时会切换图片，自动播放时就是自动触发了点击箭头的事件

元素.事件名( )

会自动触发这个事件

```
element.click()  // 第一种简写形式
```

■ trigger( )

元素 . trigger(“事件名”)

```
element.trigger("type") // 第二种自动触发模式
```

■ triggerHandler( )

元素 . triggerHandler(“事件名”)

```
element.triggerHandler(type)  // 第三种自动触发模式
```

不触发元素的默认行为。

比如用 triggerHandler(“focus”)不会让 input 表单的光标闪烁

3.事件对象

和原生 JS 的事件对象一样。

```
element.on(events,[selector],function(event) {})
```

```
<div></div>
<script>
    $("div").on("click", function(e) {
        console.log(e);
    })
</script>
```

阻止默认行为

和原生 JS 的事件对象一样。

比如阻止<a>的跳转，或者阻止 if{ }的继续判断、for 的继续循环.....

e.preventDefault( )

return fasle

阻止事件冒泡

和原生 JS 的事件对象一样。

e.stopPropagation( )

案例 - 判断键盘按下

利用事件对象的 event.keyCode

```
<input type="text">
<div></div>
<script>
    $("input").on("keydown", function(e) {
        if (e.keyCode == 13) {
            var value = $(this).val();
            $("div").html(value)
        }
    })
</script>
```

回车键

3.常用事件

■ hover( ) 鼠标经过和离开

可看作 **mouseover 和 mouseleave 的简写。**

该事件包含了 **经过 和 离开**，所以事件函数要写两个执行函数，分别是经过和离开的效果

直接用可以切换的 toggle 系列

hover(function( ){ toggle 系列的动画效果 })

```
<ul>
    <li></li>
    <li></li>
    <li></li>
</ul>
<script>
    $("ul").mouseover(function() {
        $(this).stop().slideUp();
    });
    $("ul").mouseleave(function() {
        $(this).stop().slideDown();
    });

    //用hover和toggle
    $("ul").hover(function() {
        $(this).stop().slideToggle();
    })
</script>
```

■ change( ) input 表单变化事件

表单内容改变了，就执行处理程序

表单的选定状态（**checked=true/false**）改变了，就就执行处理程序

```
<input type="text">
<input type="checkbox" name="" id="">
<script>
    $("input").eq(0).change(function() {
        alert("文本框 变了")
    });

    $("input").eq(1).change(function() {
        alert("复选框 变了")
    });
</script>
```

九．jQuery 尺寸、位置操作

1.元素尺寸 高度宽度

语法	用法
width() / height()	取得匹配元素宽度和高度值 只算 width / height
innerWidth() / innerHieght()	取得匹配元素宽度和高度值 包含 padding
outerWidth() / outerHeight()	取得匹配元素宽度和高度值 包含 padding 、border
outerWidth(true) / outerHeight(true)	取得匹配元素宽度和高度值 包含 padding 、borde、margin

不写参数，是获得高度宽度，返回的是数字型。

写参数，是修改元素的高宽。参数可以不写单位。

```
        div {
            height: 100px;
            width: 150px;
            padding-top: 50px;
            padding-left: 50px;
            margin-top: 50px;
            margin-left: 50px;
            border: crimson solid 10px;
            background-color: teal;
        }
    </style>
</head>
<body>
    <div></div>
    <script>
        //  width/heigth
        console.log($("div").width());
        console.log($("div").height());

        //  width/heigth + padding
        console.log($("div").innerWidth());
        console.log($("div").innerHeight());

        //  width/heigth + padding + border
        console.log($("div").outerWidth());
        console.log($("div").outerHeight());

        //  width/heigth + padding + border + margin
        console.log($("div").outerWidth(true));
        console.log($("div").outerHeight(true));
    </script>
```

```
150
100

200
150

220
170

270
220
```

2.元素位置

■ offset( ) 元素距离文档的位偏移 可读写

元素 . offset( )

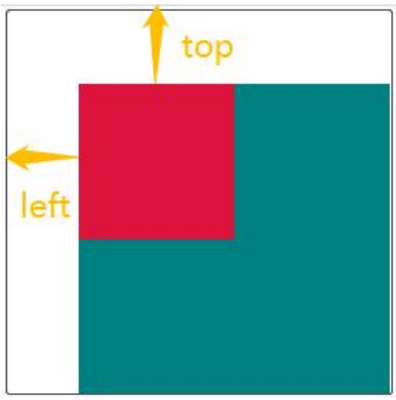
返回该元素**相对于文档 document** 的偏移坐标。

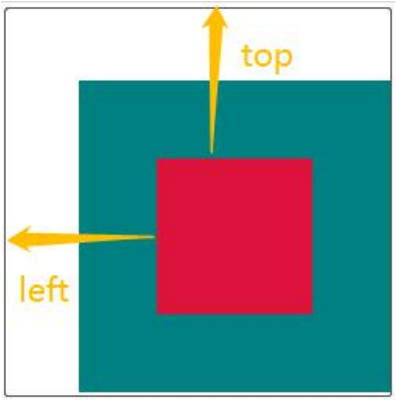
不写参数，是获得该元素**相对于文档**的 **top 和 left** 坐标。

offset( )的返回值是对象形式，top 和 left 分别是对象的属性。

写参数，是修改元素的 top 和 left。

```
        .father {
            margin-top: 50px;
            margin-left: 50px;
            height: 200px;
            width: 200px;
            background-color: teal;
        }
        .son {
            height: 100px;
            width: 100px;
            background-color: crimson;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son"></div>
    </div>
    <script>
        console.log($(".father").offset());
        console.log($(".son").offset());

        console.log($(".son").offset().top);
        console.log($(".son").offset().left);

        $(".son").offset({
            top: 100,
            left: 100
        });
        console.log($(".son").offset().top);
        console.log($(".son").offset().left);
    </script>
```






```
▶ {top: 50, left: 50}
▶ {top: 50, left: 50}

50
50

100
100
```

■ position( ) 元素距离定位父元素的位偏移　只读

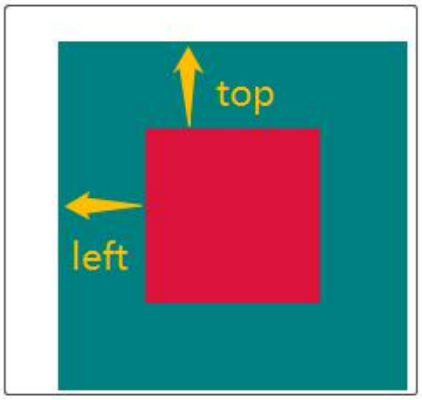
返回该元素**相对于带有定位的父元素**的偏移坐标。

position( )的返回值是对象形式，top 和 left 分别是对象的属性。

只能获得位置，不能修改位置。

和 CSS 的定位 **position** 一样，**父元素有定位则根据父元素，若父元素都没定位则根据文档。**

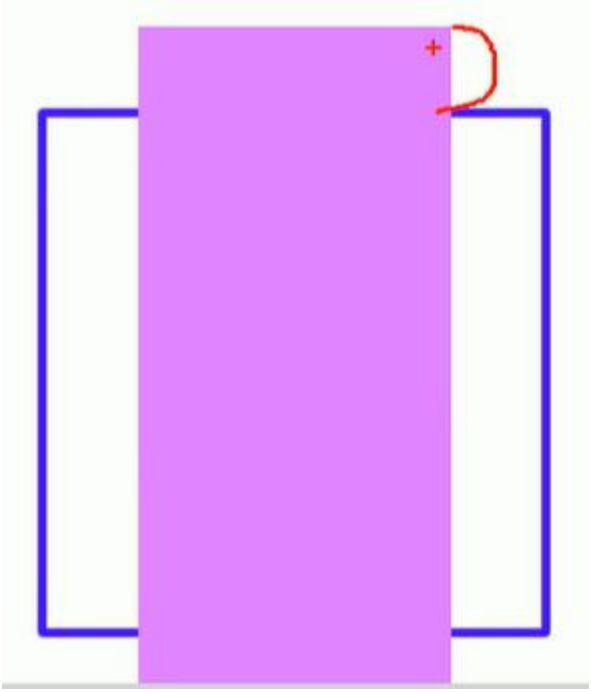
```
        .father {
            position: relative;
            margin-top: 50px;
            margin-left: 50px;
            height: 200px;
            width: 200px;
            background-color: teal;
        }
        .son {
            position: absolute;
            top: 50px;
            left: 50px;
            height: 100px;
            width: 100px;
            background-color: crimson;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son"></div>
    </div>
    <script>
        console.log($(".father").position());
        console.log($(".son").position());

        console.log($(".son").position().top);
        console.log($(".son").position().left);
    </script>
```




```
▶ {top: 0, left: 0}
▶ {top: 50, left: 50}

50
50
```

■ scrollTop( )/scrollLeft　页面滚动后元素卷去的部分

和原生 JS 的 BOOM 的 scroll 类似。



可以获取**页面 windo 滚动后，文档 document 的被卷去的部分。**

也可以设置页面打开时，页面所在的位置。

```
// 页面滚动后 获取页面被卷去的长度
$(window).scroll(function() {
    console.log($(document).scrollTop());
})

//打开页面后 自动定位到页面的所在位置
$(document).scrollTop(100);
```

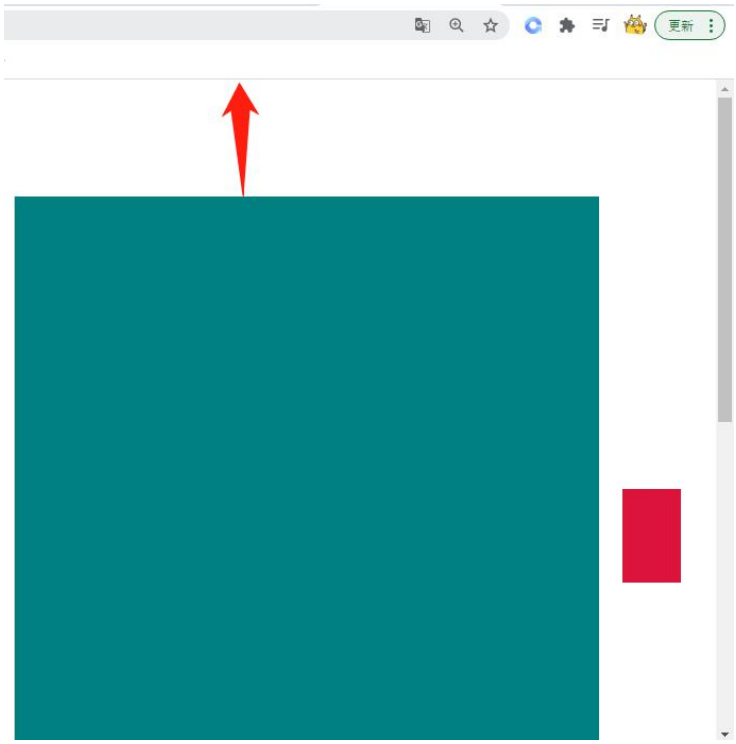
案例 - 滚动页面显示导航条

等页面滚动到指定位置后，才让红色盒子显示。

比如，指定页面滚动到 蓝色盒子的顶部位置时，红色盒子显示。

即，若页面卷去的部分 大于等于 蓝色盒子的顶部到页面距离，则显示红色盒子

若页面卷去的部分 小于 蓝色盒子的顶部到页面距离，则隐藏红色盒子



```
<div></div>
<div class="pic" style="display: none;"></div>
<script>
    $(window).scroll(function() {

        if ($(document).scrollTop() >= $("div").offset().top) {
            $(".pic").stop().fadeIn();
        } else {
            $(".pic").stop().fadeOut();
        }
    })
</script>
```

案例 - 页面**直接跳转**到指定位置

点击了指定元素后，让页面 **scrollTop(指定位置)**

```
<div></div>
<div class="pic" style="display: none;"></div>
<script>
    $(window).scroll(function() {

        if ($(document).scrollTop() >= $("div").offset().top) {
            $(".pic").stop().fadeIn();

            $(".pic").click(function() {
                $(document).scrollTop(0);
            });
        } else {
            $(".pic").stop().fadeOut();
        }
    })
</script>
```

案例 - 页面**动画滚动**到指定位置

利用 jQuery 的**动画函数 animate( )的 scrollTop 属性**

元素 . animate({ scrollTop : 指定 top 位置 })

但是 animate( )的让元素做动画，但是要滚动的是页面 document。

用$(“body，html”) 代替 document

```
<div></div>
<div class="pic" style="display: none;"></div>
<script>
    $(window).scroll(function() {
        if ($(document).scrollTop() >= $("div").offset().top) {
            $(".pic").stop().fadeIn();

            $(".pic").click(function() {
                $("html,body").stop().animate({
                    scrollTop: 0
                });
            });
        } else {
            $(".pic").stop().fadeOut();
        }
    })
</script>
```