

1.jQuery概述

1.1 JavaScript库

仓库:可以把很多东西放到这个仓库里面。找东西只需要到仓库里面查找到就可以了。

JavaScript库:即library,是一个**封装**好的特定的**集合**(方法和函数).从封装一大堆函数的角度理解库,就是在这个库中,封装了很多预先定义好的函数在里面,比如动画animate. hide. show,比如获取元素等。

简单理解:就是一个JS文件,里面对我们原生js代码进行了封装,存放到这里面。这样我们可以快速高效的使用这些封装好的功能了。

比如jQuery,就是为了快速方便的操作DOM,里面基本都是函数(方法)。

常见的JavaScript库

- jQuery
- Prototype
- YUI
- Dojo
- ExtJS
- 移动端的zepto

这些库都是对原生JavaScript的封装,**内部都是用JavaScript实现的**,我们**主要学习的是jQuery**。

1.2 jQuery的概念

jQuery是一个快速、简洁的JavaScript库,其设计的宗旨是"write Less, Do More",即倡导写更少的代码,做更多的事情。

j就是JavaScript; Query 查询;意思就是查询js,把js中的DOM操作做了封装,我们可以快速的查询使用里面的功能。

jQuery封装了JavaScript 常用的功能代码,优化了DOM操作、事件处理、动画设计和Ajax交互。

学习jQuery本质:就是学习调用这些函数(方法)。

jQuery出现的目的是加快前端人员的开发速度,我们可以非常方便的调用和使用它,从而提高开发效率。

优点

- 轻量级。核心文件才几+kb,不会影响页面加载速度;跨浏览器兼容。基本兼容了现在主流的浏览器
- 链式编程、隐式迭代
- 对事件、样式、动画支持,大大简化了DOM操作
- 支持插件扩展开发。有着丰富的第三方的插件,例如:树形菜单、日期控件、轮播图等
- 免费、开源

2.jQuery的基本使用

2.1 jQuery 的下载

官网地址: <https://jquery.com/>

版本:

- 1x :兼容IE 678等低版本浏览器，官网不再更新
- 2x :不兼容IE 678等低版本浏览器，官网不再更新
- 3x:不兼容IE 678等低版本浏览器，是官方主要更新维护的版本

各个版本的下载: <https://code.jquery.com/>

2.2 jQuery 的使用步骤

1.引入jQuery文件

2.使用即可

[Download the compressed, production jQuery 3.6.0](#)

[Download the uncompressed, development jQuery 3.6.0](#)

compressed 压缩过的，uncompressed未压缩过的，选择一种将里面的代码复制到创建的js文件中即可，然后在Html中引用后就可使用

2.3 jQuery 的入口函数

有两种页面加载方法，效果一致

```
$ (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  
}) ;
```

1.等着DOM结构渲染完毕即可执行内部代码,不必等到所有外部资源加载完成, jQuery帮我们完成了封装。

2.相当于原生js中的DOMContentLoaded.

3.不同于原生js中的load事件是等页面文档、外部的js文件、css文件、图片加载完毕才执行内部代码。

4.更推荐使用第一种方式。

示例:

```

<script>
    // $('div').hide();//隐藏盒子
    //等着页面DOM加载完毕再去执行js代码
    //1)
    /* $(document).ready(function () {
        $('div').hide();
    }) */
    //2)
    $(function () {
        $('div').hide();
    })
</script>
<div></div>

```

2.4 jQuery 的顶级对象\$

1.\$是jQuery的别称,在代码中可以使用jQuery代替\$,但一般为了方便,通常都直接使用\$.

2.\$是jQuery的顶级对象,相当于原生JavaScript中的window。把元素利用\$包装成jQuery对象,就可以调用jQuery的方法。

2.5 jQuery对象和DOM对象

1.用原生JS获取来的对象就是DOM对象

2.jQuery方法获取的元素就是jQuery对象。

3.jQuery对象本质是:利用\$对DOM对象包装后产生的对象(伪数组形式存储)。

示例:

```

//1.DOM对象: 使用原生js获取过来的对象就是DOM对象
var myDiv = document.querySelector('div');//myDiv是DOM对象
console.log(myDiv);
//2.jQuery对象: 使用jQuery方式获取过来的对象就是jQuery对象。 本质: 通过$把DOM元素
进行了包装
$('div');
console.dir($('div'))
//3.jQuery对象只能用jQuery方法, DOM对象只能使用原生的JavaScript属性和方法, 两者不
能混用

// myDiv.style.display = 'none'//✓
$('div').style.display = 'none'//× $('div')是jQuery对象不能使用原生的
JavaScript属性和方法

```

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

```
► S.fn.init(1)
```

```
✖ Uncaught TypeError: Cannot set property 'display' of undefined
at 02-jQuery对象和DOM对象.html:30
```

注意:

1.DOM对象和jQuery对象两者并不相同

2.jQuery对象只能用jQuery方法, DOM对象只能使用原生的JavaScript属性和方法, **两者不能混用**

2.6 jQuery 对象和DOM对象相互转换

DOM对象与jQuery对象之间是可以相互转换的。

因为原生js比jQuery更大，原生的一些属性和方法jQuery没有给我们封装，要想使用这些属性和方法需要jQuery对象转换为DOM对象才能使用。

1.DOM对象转换为jQuery对象: **\$(DOM对象)**

```
$('#div')
```

2.jQuery对象转换为DOM对象(两种方式)

```
$('#div')[index] //index是索引号  
$('#div').get(index) //index 是索引号
```

注意第一种方法索引号外是中括号[]，第二种方法索引号外是()

示例：

```
//1.将DOM对象转换为jQuery对象  
//1)用jQuery方式获取就是jQuery对象  
// $('#video')  
//2)用原生js获取就是DOM对象  
var myVideo = document.querySelector('video')  
  
//2.将jQuery对象转换为DOM对象  
$('#video')[0].play()  
// $('#video').get(0).play()
```

3.jQuery 选择器

3.1 jQuery 基础选择器

原生JS获取元素方式很多,很杂,而且兼容性情况不一致,因此jQuery给我们做了封装,使获取元素统一标准。

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

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

3.2 jQuery 层级选择器

名称	用法	描述
子代选择器	<code>\$("ul>li");</code>	使用>号,获取亲儿子层级的元素;注意,并不会获取孙子层级的元素
后代选择器	<code>\$("ul li");</code>	使用空格, 代表后代选择器, 获取ul下的所有li元素, 包括孙子等

3.3隐式迭代(重要)

遍历内部DOM元素(伪数组形式存储)的过程就叫做隐式迭代。

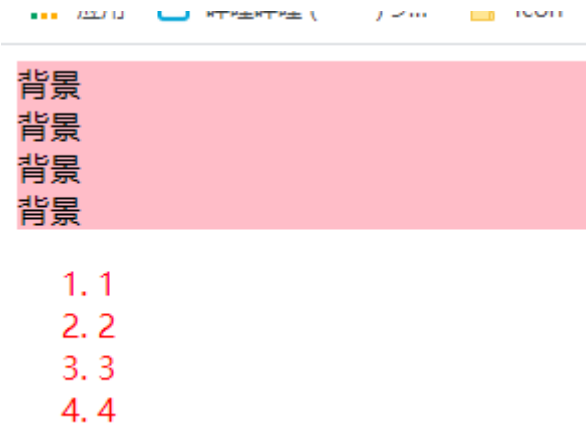
简单理解:给匹配到的所有元素进行循环遍历,执行相应的方法,而不用我们再进行循环,简化我们的操作,方便我们调用。

知识铺垫
jQuery设置样式

```
$('div').css('属性','值')
```

示例:

```
$(function () {  
    //让div背景颜色都变成粉色  
    $('div').css('background-color', 'pink')  
    //让ol下的li字体都变为红色  
    $('ol li').css('color', 'red')  
})
```



3.4 jQuery筛选选择器

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

注意: 后三种选择器是以索引号为准, 索引号从0开始

示例:

```
$(function () {  
    //让ul第一个li字体变红色  
    $('ul li:first').css('color', 'red')  
    //让ul第三个li字体变绿色, 对应索引号为2  
    $('ul li:eq(2)').css('color', 'green')  
    //让ol的索引号为偶数的li字变粉色  
    $('ol li:even').css('color', 'pink')  
    //让ol的索引号为奇数的li变灰色  
    $('ol li:odd').css('color', 'gray')  
})
```

- 保持谨慎
- 保持谨慎
- 保持谨慎
- 保持谨慎
- 保持谨慎

1. 不放纵
2. 不放纵
3. 不放纵
4. 不放纵
5. 不放纵
6. 不放纵

3.5 jQuery筛选方法(重点)

语法	用法	说明
parent()	<code>\$("li").parent();</code>	查找父级
children(selector)	<code>\$("ul").children("li")</code>	相当于 <code>\$("ul>li")</code> ,最近一级(亲儿子)
find(selector)	<code>\$("u1"). find("li");</code>	相当于 <code>\$("ul li")</code> ,后代选择器
siblings(selector)	<code>\$(" .first").siblings("li");</code>	查找兄弟节点, 不包括自己本身
nextAll([expr])	<code>\$(" .first").nextAll()</code>	查找当前元素之后所有的同辈元素
prevAll([expr])	<code>\$(" . last"). prevAll()</code>	查找当前元素之前所有的同辈元素
hasClass(class)	<code>\$('#div').hasClass("protected")</code>	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)	<code>\$("li").eq(2);</code>	相当于 <code>\$("li:eq(2)")</code> , index从0开始

重点记住: parent() children() find() siblings() eq()

示例1: 父子方法 parent() children() find()

```

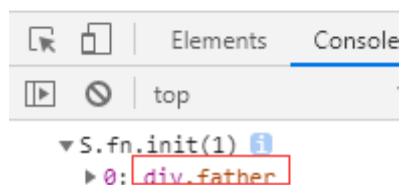
<div class="grandfather">
  <div class="father">
    <div class="son">儿子</div>
  </div>
</div>
<div class="box">
  <p>子</p>
  <div class="boxs">
    <p>孙</p>
  </div>
</div>
<script>
  //注意都是方法，需要带括号
  $(function () {
    //1.父 parent() 返回的是最近以及的父级元素 亲爸爸
    console.log($('.son').parent()); //father
    //2.子
    //1) 亲儿子 children('') 类似于子代选择器ul>li
    // $('.box').children('p').css('color', 'red')
    //2) 所有孩子，包括儿子、孙子 find('') 类似于后代选择器
    $('.box').find('p').css('color', 'red')
  })
</script>

```

儿子

子

孙



示例2:

```

<ul>
  <li>兄弟</li>
  <li class="bro">兄弟</li>
  <li>兄弟</li>
  <li>兄弟</li>
  <li>兄弟</li>
</ul>
<ol>
  <li>姐妹</li>
  <li>姐妹</li>
  <li>姐妹</li>
  <li>姐妹</li>
  <li>姐妹</li>
</ol>
<div class="current">我有</div>
<div>我没有</div>

```

```

<script>
    $(function () {
        //1.兄弟元素siblings 除了自己之外的所有亲兄弟
        $('ul .bro').siblings('li').css('color', 'red')
        //2.选择第n个元素
        //1) 选择器的方式
        $('ol li:eq(2)').css('color', 'blue')
        //2) 选择方法的方式 更建议使用, 里面的数字可以使用变量来变化
        $('ol li').eq(3).css('color', 'pink')
        //3.判断是否有某个类名, 有返回true
        console.log($('div:first').hasClass('current'));
    })
</script>

```

- 兄弟
- 兄弟
- 兄弟
- 兄弟
- 兄弟

1. 姐妹
2. 姐妹
3. 姐妹
4. 姐妹
5. 姐妹

我有
我没有

案例：新浪下拉菜单

```

$(function () {
    //1.鼠标经过li, 显示ul
    $('.nav>li').mouseover(function () {
        //$(this) jQuery当前元素, this不用加引号
        //show()显示元素    hide()隐藏元素
        $(this).children('ul').show()
    })
    $('.nav>li').mouseout(function () {
        $(this).children('ul').hide()
    })
})

```

登录 微博 博客 邮箱

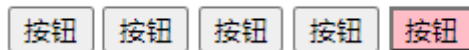
评论

私信

@我

案例：jQuery的排他思想

```
$(function () {  
    //1.隐式迭代，给所有button添加点击事件  
    $('button').click(function () {  
        //2.当前元素变化背景颜色  
        $(this).css('background', 'pink')  
        //3.隐式迭代，让其他兄弟去掉背景颜色  
        $(this).siblings('button').css('background', '')  
    })  
})
```



案例: 淘宝服饰精品案例分析



①核心原理:鼠标经过左侧盒子某个小li,就让内容区盒子相对应图片显示,其余的图片隐藏。

②需要得到当前小li的索引号,就可以显示对应索引号的图片

③jQuery得到当前元素索引号`$(this).index()` amazing! 都不需要定义索引号就能获得，注意index后面有括号，是个方法

④中间对应的图片,可以通过`eq(index)` 方法去选择

⑤显示元素`show()`隐藏元素`hide()`

```
$(function () {  
    //1.鼠标经过左侧的li  
    $('.left li').mouseover(function () {  
        //2.得到当前li的索引号  
        var index = $(this).index();  
        //3.让右侧盒子相应索引号的图片显示  
        $('.right div').eq(index).show()  
        //4.让其余图片（其他兄弟）隐藏  
        $('.right div').eq(index).siblings().hide()  
    })  
})
```



3.6链式编程

链式编程是为了节省代码量,看起来更优雅。

```
$(this).css('color', 'red').siblings().css('color', ''); //我的样式文字颜色为红，我兄弟的颜色为默认
```

使用链式编程一定注意是哪个对象执行样式

示例：

```
$(function () {
    $('button').click(function () {
        // $(this).css('background', 'pink')
        // $(this).siblings('button').css('background', '')
        //上面代码优化，链式编程
        $(this).css('color', 'red').siblings().css('color', 'blue')
        //我的颜色为红色，我的兄弟颜色为蓝色
    })
})
```

按钮

按钮

按钮

按钮

按钮

4.jQuery样式操作

4.1操作css方法

jQuery可以使用CSS方法来修改简单元素样式;也可以操作类,修改多个样式。

1.参数只写属性名,则是返回属性值

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

2.参数是属性名,属性值,逗号分隔,是设置一组样式,属性必须加引号,值如果是数字可以不用跟单位和引号

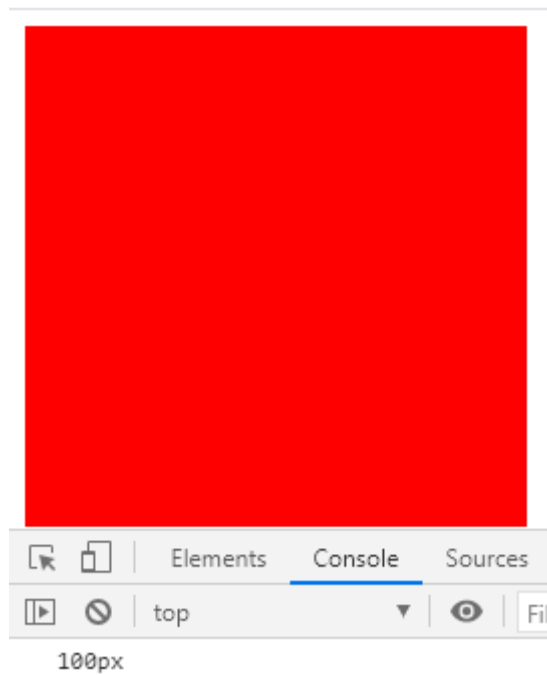
```
$('#div').css('width', 200)
```

3.参数可以是对象形式,方便设置多组样式。属性名和属性值用冒号隔开,属性可以不用加引号,

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

示例:

```
$(function () {  
    console.log($('#div').css('width')); //100px  
    // $('#div').css('background-color', 'blue') //属性名必须带引号  
    // $('#div').css('width', 200) //属性值是数字可以不加引号, 不带单位  
    //对象形式设置多种样式  
    $('#div').css({  
        width: 250,  
        height: 250,  
        backgroundColor: 'red' //如果是复合属性必须采取驼峰命名法  
    })  
})
```



注意:

- 1.如果是复合属性必须采取驼峰命名法
- 2.如果值是数字, 可以不带单位, 不加引号; 否则**属性名和属性值都需要加引号** (参数非对象时)

4.2设置类样式方法

作用等同于以前的classList, 可以操作类样式, **注意操作类里面的参数不要加.**

1.添加类

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

2.移除类

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

3.切换类 不断添加类和删除类

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

示例:

```
$(function () {  
    //1.添加类  addClass  
    /* $('#div').click(function () {  
        $(this).addClass('current')  
    }) */  
    //2.移除类  removeClass  
    /* $('#div').click(function () {  
        $(this).removeClass('current')  
    }) */  
    //3.切换类  
    $('#div').click(function () {  
        $(this).toggleClass('current')  
    })  
})
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100



来回切换



案例：tab栏切换

①点击上部的li ,当前li添加current类,其余兄弟移除类。

②点击的同时,得到当前li的索引号

③让下部里面相应索引号的item显示,其余的item隐藏

```
$(function () {  
    //1.点击上部的li ,当前li添加current类,其余兄弟移除类  
    $('#.tab_list li').click(function () {  
        $(this).addClass('current').siblings().removeClass('current')  
    })  
    //2.点击的同时,得到当前li的索引号  
    var index = $(this).index()  
    //3.让下部里面相应索引号的item显示,其余的item隐藏  
    $('#.tab_con .item').eq(index).show().siblings().hide()  
})
```

商品介绍 规格与包装 售后保障 商品评价(1.1万+) 手机社区

规格与包装模块

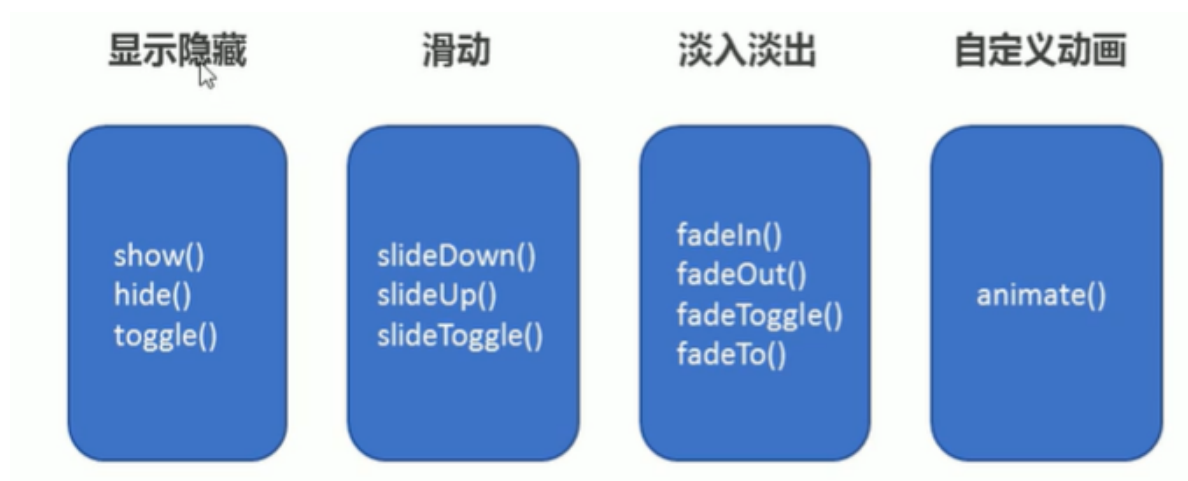
4.3类操作与className区别

原生JS中className会覆盖元素原先里面的类名。

jQuery里面类操作(添加、删除、更换类)只是对指定类进行操作,不影响原先的类名。

5.jQuery效果

jQuery给我们封装了很多动画效果,最为常见的如下:



5.1显示隐藏效果

1.显示语法规范

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

2.显示参数

(1)参数都可以省略,无动画直接显示。

(2)speed:三种预定速度之一的字符串("slow", "normal", or "fast")或表示动画时长的毫秒数值(如:1000)。

(3) easing : (Optional)用来指定切换效果,默认是"swing" (慢快慢),可用参数"linear"。

(4) fn:回调函数,在动画完成时执行的函数,每个元素执行一次。

切换 (toggle)、隐藏(hide)与显示语法规范和参数一模一样

一般情况下,我们都不加参数,直接显示隐藏就可以了

5.2滑动效果

1.下滑效果语法规范

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

2.下滑效果参数

(1)参数都可以省略,无动画直接显示。

(2)speed:三种预定速度之一的字符串("slow", "normal", or "fast")或表示动画时长的毫秒数值(如:1000)。

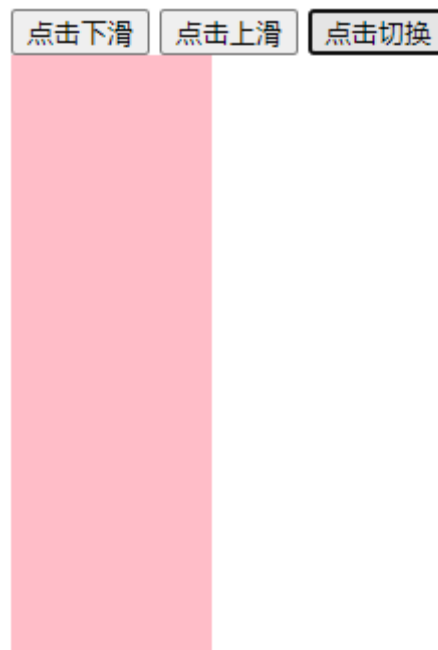
(3) easing : (Optional)用来指定切换效果,默认是"swing" (慢快慢) ,可用参数"linear" .

(4) fn:回调函数,在动画完成时执行的函数,每个元素执行一次。

上滑 (slideUp)、隐藏(slidToggle)与下滑 (slideDown) 语法规范和参数一模一样

示例:

```
$(function () {
    $('button').eq(0).click(function () {
        //上滑
        $('div').slideDown()
    })
    $('button').eq(1).click(function () {
        //下滑
        $('div').slideUp()
    })
    $('button').eq(2).click(function () {
        //切换
        $('div').slideToggle()
    })
})
```



5.3事件切换

```
hover([over,]out)
```

(1) over:鼠标移到元素上要触发的函数(相当于mouseenter)

(2) out:鼠标移出元素要触发的函数(相当于mouseleave)

示例：将新浪下拉菜单js部分改进

//hover事件切换将代码改进1

```
//1.hover事件切换 鼠标经过和离开的复合写法
/* $('nav>li').hover(function () {
    $(this).children('ul').slideDown(200)
}, function () {
    $(this).children('ul').slideUp(200)
}) */
```

//改进2

```
//2.如果只写一个函数，那么鼠标经过和鼠标离开都会触发这个函数
$('nav>li').hover(function () {
    $(this).children('ul').slideToggle(200)
})
```

都能实现下拉效果

注意：

- 1.hover事件切换是鼠标经过和离开的复合写法
- 2.如果写两个参数，第一个参数里放鼠标经过的函数，第二个参数里放鼠标离开函数。

```
$('#nav>li').hover(function() {}, function() {})
```

- 3.如果只写一个函数，那么鼠标经过和鼠标离开都会触发这个函数(可以尝试放入切换效果toggle、slideToggle)

5.4动画队列及其停止排队方法

1.动画或效果队列

动画或者效果一旦触发就会执行,如果多次触发,就造成多个动画或者效果排队执行。

如上个案例新浪下拉菜单，若多次鼠标经过、离开会造成下图情况，鼠标离开后菜单仍在上下滑动。



2.停止排队

```
stop()
```

- (1) stop()**方法**（带括号）用于停止动画或效果。
- (2) 注意: stop()写到动画或者效果的前面,相当于停止结束上一次的动画，一般写了动画就要添加stop()。

新浪下拉菜单案例改进

```
$( '.nav>li' ).hover(function () {  
    //stop()方法必须写在动画的前面  
    $(this).children('ul').stop().slideToggle(200)  
})
```

就不会出现上图的情况了

5.5淡入淡出效果

1.淡入效果语法规范

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

淡出(fadeOut)、切换(fadeToggle)与淡入参数一样

2.渐进方式调整到指定的不透明度

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

2.效果参数

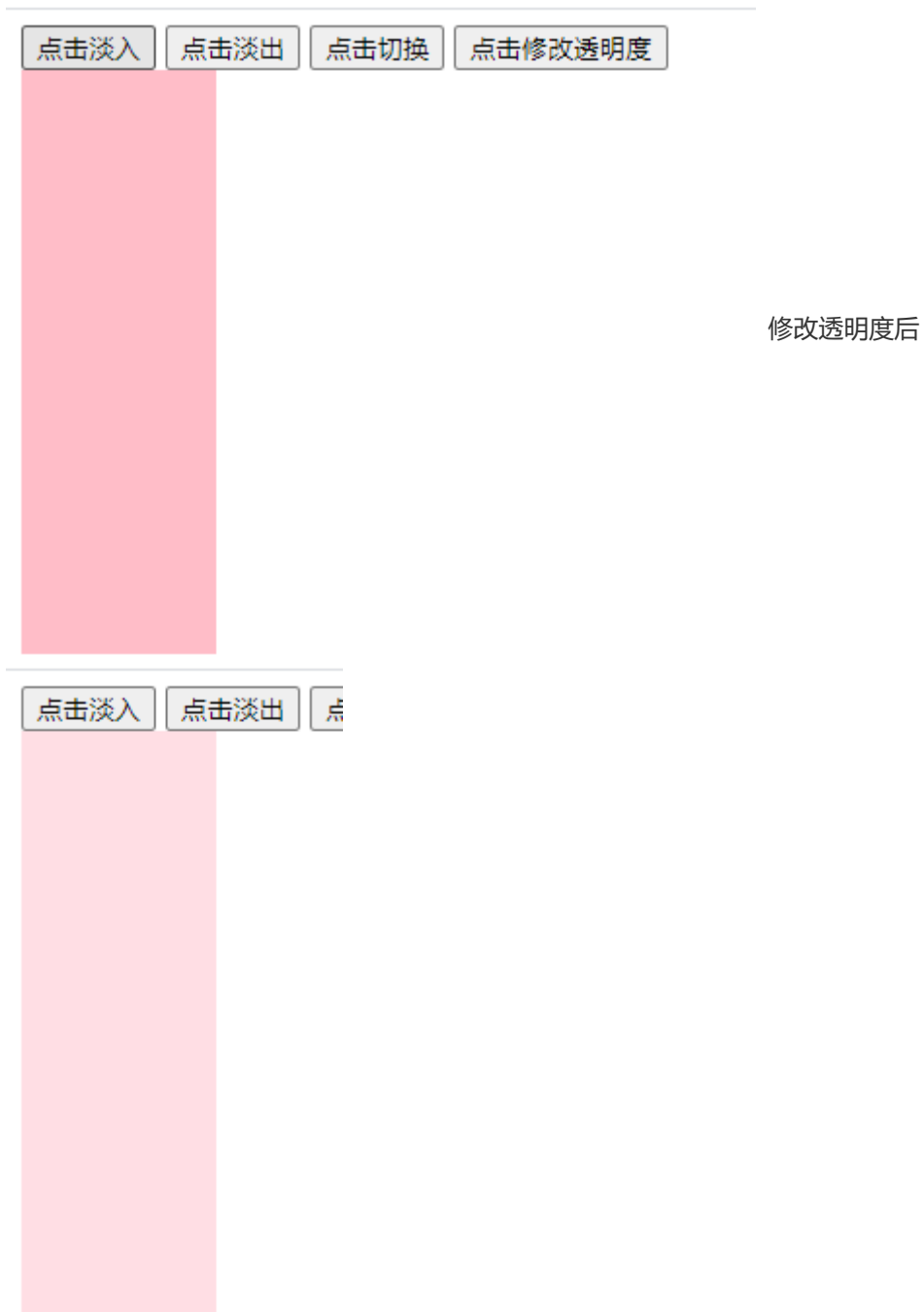
(1) **speed :必须写**,三种预定速度之一的字符串("slow" , "normal" ,or "fast")或表示动画时长的毫秒数值(如: 1000)。

(2) **opacity:必须写,透明度,取值0~1之间**。透明度1为不透明, 原来的样子

(3) easing : (Optional)用来指定切换效果,默认是"swing",可用参数"linear"。

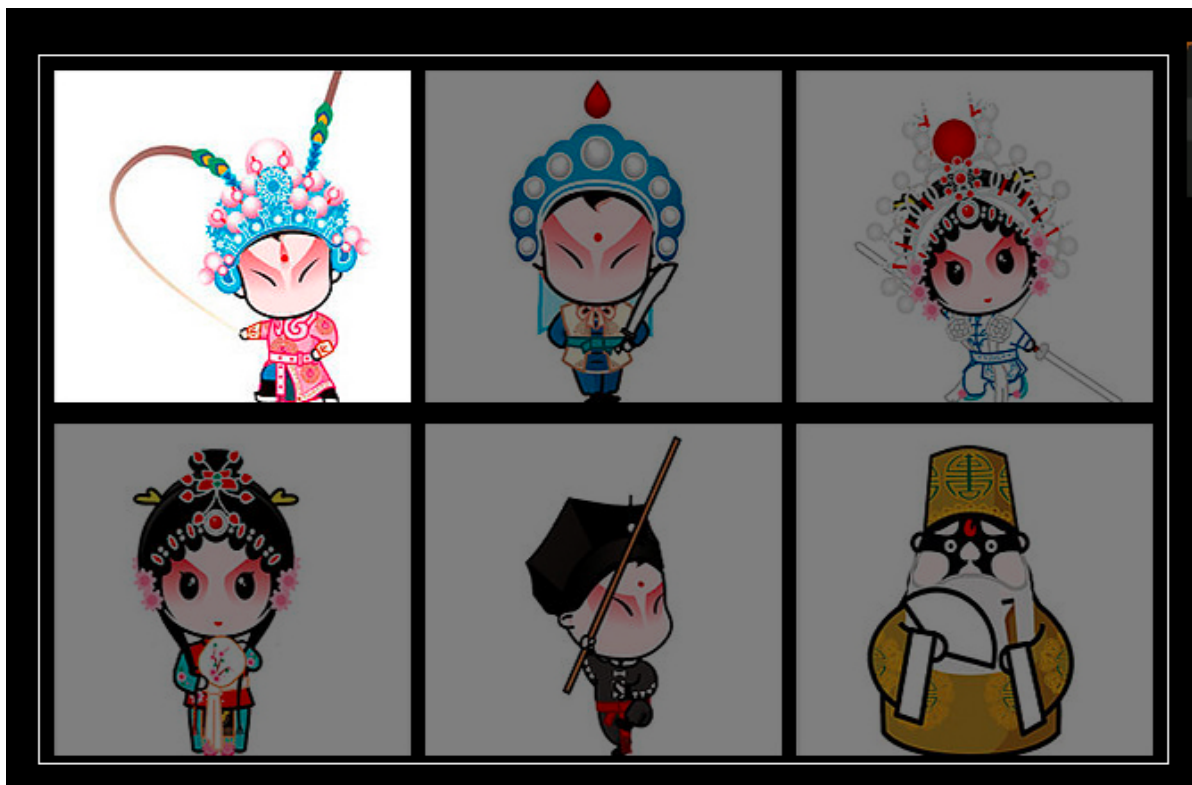
示例:

```
$(function () {  
    $('button').eq(0).click(function () {  
        //淡入  
        $('div').fadeIn()  
    })  
    $('button').eq(1).click(function () {  
        //淡出  
        $('div').fadeOut()  
    })  
    $('button').eq(2).click(function () {  
        //切换  
        $('div').fadeToggle()  
    })  
    $('button').eq(3).click(function () {  
        //修改透明度 速度和透明度必须写  
        $('div').fadeTo(1000, .5)  
    })  
})
```

案例：高亮显示

实现：鼠标经过哪个图片，哪个图片高亮其余图片变暗（使用fadeTo降低透明度）

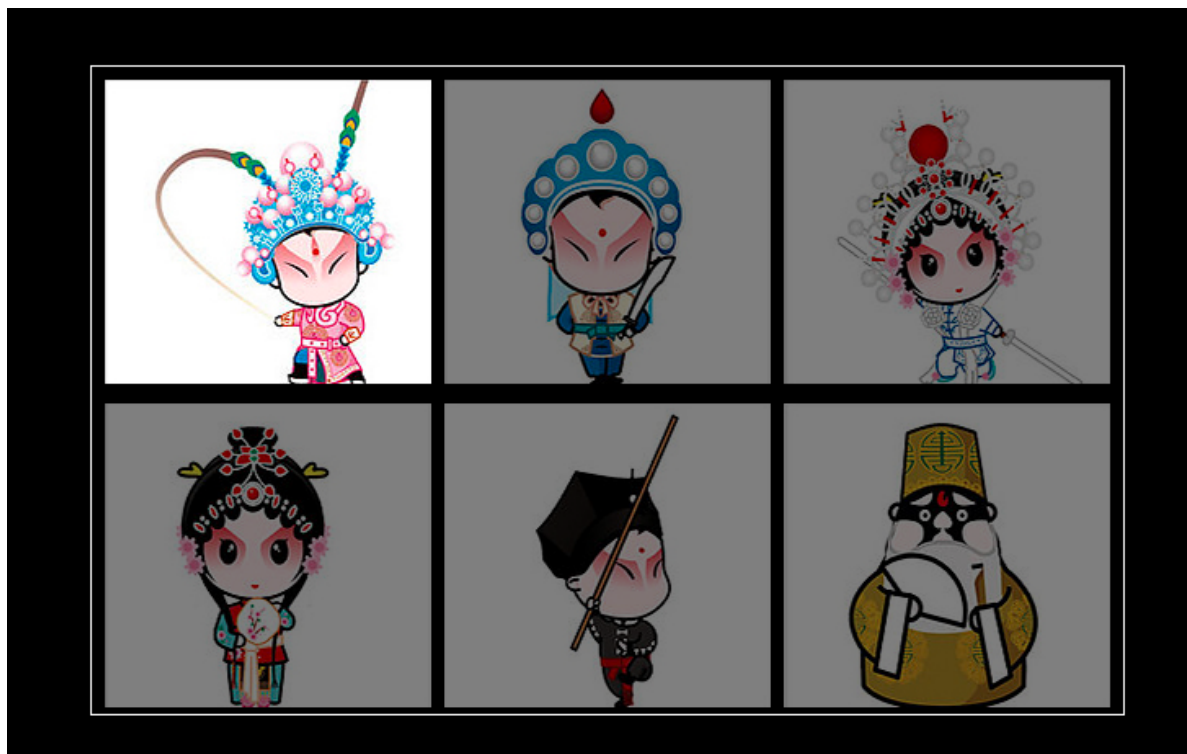


```
$(function () {
    $('ul li').hover(function () {
        //鼠标经过，让其他兄弟透明度变0.5
        $(this).siblings().stop().fadeTo(400, .5)
    }, function () {
        //鼠标离开，让其他兄弟透明度为1
        //注意给动画添加stop()
        $(this).siblings().stop().fadeTo(400, 1)
    })
})
```

默认状态:



鼠标经过第一个图片：



5.6自定义动画animate

1.语法

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

2.参数

(1) **params**:想要更改的样式属性,以对象形式传递,必须写, 属性名可以不用带引号,如果是复合属性则需要采取驼峰命名法borderLeft.其余参数都可以省略。

(2) **speed**:三种预定速度之一的字符串("slow", "normal", or "fast")或表示动画时长的毫秒数值(如:1000)。

(3) **easing**: (Optional)用来指定切换效果,默认是"swing",可用参数"linear".

(4) **fn**:回调函数,在动画完成时执行的函数,每个元素执行一次。

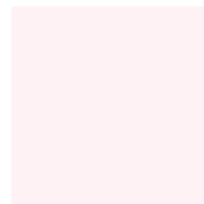
注意动画不仅可以左走，还可以上下右走，但仍需要给运动的元素添加定位

示例：

```
$(function () {  
    $('button').click(function () {  
        $('div').animate({  
            top: 300,  
            left: 400,  
            opacity: .2  
        }, 100)  
    })  
})
```

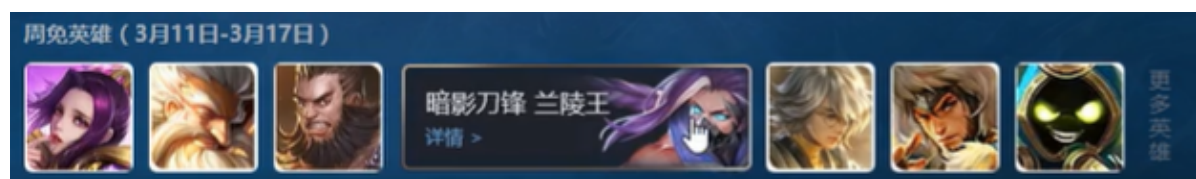


点击按钮后:



案例:王者荣耀手风琴效果

实现：鼠标经过小图，显示大图，离开后又变为大图模式



模块划分：

一个大盒子box，里面使用ul li a放置图片



注意小图和大图的实现，让小图通过定位到大图上。

html+css部分

```
<div class="box">
  <ul>
    <li class="current"><a href="javascript:;">
      
      
    </a>
  </li>
  <li><a href="javascript:;">
    
    
  </a>
</li>
  <li><a href="javascript:;">
    
    
  </a>
</li>
  <li><a href="javascript:;">
    
    
  </a>
</li>
  <li><a href="javascript:;">
    
    
  </a>
</li>
  <li><a href="javascript:;">
    
    
  </a>
</li>
  <li><a href="javascript:;">
    
    
  </a>
</li>
</ul>
</div>
```

```
* {
  padding: 0;
  margin: 0;
}

.box {
  width: 852px;
  background: url(images/bg.png) no-repeat;
  margin: 100px auto;
  padding: 10px;
  overflow: hidden;
}
```

```

    .box ul li {
        position: relative;
        float: left;
        /* 注意要给li设置宽度和高度，否则大图全部隐藏后无法根据每个li进行定位而叠在一起 */

        width: 69px;
        height: 69px;
        list-style: none;
        margin-right: 10px;
    }

    .box .current {
        width: 224px;
    }

    .current .big {
        display: block;
    }

    .current .small {
        display: none;
    }

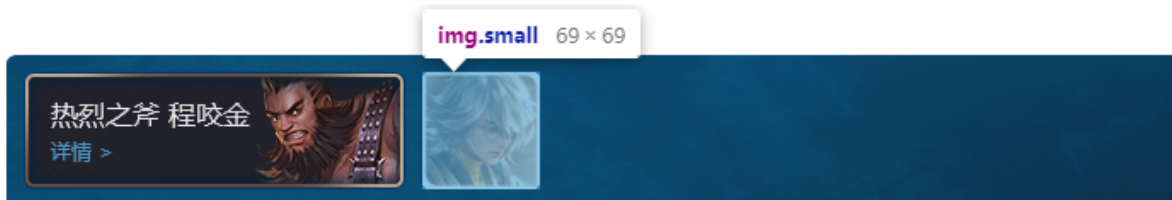
    .big {
        width: 224px;
        display: none;
    }

    .small {
        position: absolute;
        top: 0;
        left: 0;
        width: 69px;
        height: 69px;
        border-radius: 5px;
    }

```

注意:

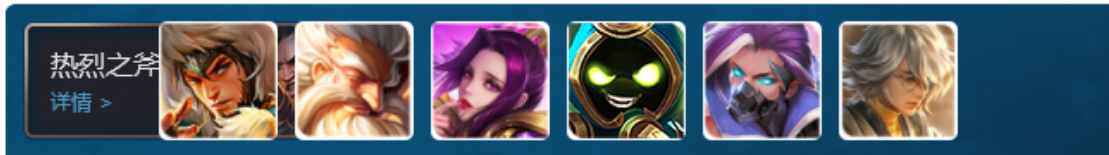
问题1: 如下图，小图层叠在一起导致只显示放在最后的小图



原因: 小图根据li进行定位，而未给li设置宽度和高度，导致小图层叠在一起。

解决方法: 给li设置宽度和高度，以小图的大小为准，69x69

问题2: 如下图，小图都显示出来了，但盖住了当前大图的一部分



原因：上面设置了每个li宽69，所以大图所处的li宽度也为69，因此出现上图情况

解决方法：将大图所在的li改为大图的宽度，注意需要权重大于之前的li

```
.box .current {  
    width: 224px;  
}
```

案例分析

- ①鼠标经过某个小li有两步操作：
- ②当前小li宽度变为224px，同时里面的小图片淡出，其兄弟大图片淡入
- ③其余兄弟小li宽度变为69px，小图片淡入，大图片淡出

```
$(function () {  
    //鼠标经过某个小li有两步操作：  
    $('.box li').mouseenter(function () {  
        //1.当前小li宽度变为224px，同时里面的小图片淡出，大图片淡入  
        $(this).stop().animate({  
            width: 224 //注意该出写成动画更好，有动画效果  
        }).find('.small').stop().fadeOut().siblings('.big').stop().fadeIn()  
        //2.其余兄弟小li宽度变为69px，小图片淡入，大图片淡出  
        $(this).siblings('li').stop().animate({  
            width: 69  
        })  
        .find('.small').stop().fadeIn().siblings('.big').stop().fadeOut()  
    })  
})
```

注意：

- 1.使用动画animate修改li的宽度，有动画，宽度变化更流畅
- 2.找到当前li里面的小图片添加淡出效果，注意小图片是li的孙子，不能通过children来查找，那么可以通过find查找符合要求的所有孩子
- 3.链式编程里哪个对象执行样式

```
$(this).stop().animate({  
    width: 224 //注意该出写成动画更好，有动画效果  
}).find('.small').stop().fadeOut().siblings('.big').stop().fadeIn()  
})//我添加动画，我找到所有有small类的孩子添加淡出动画，找到small类的兄弟big为其添加淡入动画，  
我-我的small孩子-small孩子的兄弟
```

- 4.里面出现了动画，需要给每个动画前都添加stop()方法

6.jQuery属性操作

6.1设置或获取元素固有属性值prop()

所谓元素固有属性就是元素本身自带的属性,比如<a>元素里面的href , 比如<input>元素里面的type.

1.获取属性语法

```
prop("属性")
```

2.设置属性语法

```
prop("属性", "属性值")
```

6.2设置或获取元素自定义属性值attr()

用户自己给元素添加的属性,我们称为自定义属性。比如给 div添加index = "1"。

1.获取属性语法

```
attr("属性") // 类似原生getAttribute()
```

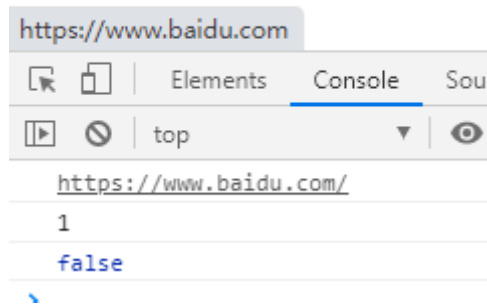
2.设置属性语法

```
attr("属性", "属性值") // 类似原生setAttribute()
```

示例:

```
<a href="https://www.baidu.com" title="百度">百度</a>
<input type="checkbox" name="" id="" checked>
<div index='1'></div>
<script>
    $(function () {
        //1.element.prop('属性名') 获取元素固有属性值
        console.log($('a').prop('href'));
        //element.prop('属性名','属性值') 改变属性值
        $('a').prop('title', '都挺好')
        //change事件, 发生改变就触发
        $('input').change(function () {
            console.log($(this).prop('checked')); //不勾选输出为false
        })
        //2.attr('属性名') 获取元素的自定义属性
        console.log($('div').attr('index')); //1
    })
</script>
```


百度



该方法也可以获取H5自定义属性（data-index）

```
<div index='1' data-index='2'></div>

//也可获取h5自定义属性
console.log($('#div').attr('data-index'));
```

1
2

6.3数据缓存data()

data()方法可以在指定的元素上存取数据,并不会修改DOM元素结构。一旦页面刷新,之前存放的数据都将被移除。

1.附加数据语法

```
data("name", "value") // 向被选元素附加数据
```

2.获取数据语法

```
date("name")//向被选元素获取数据
```

同时,还可以读取HTML5自定义属性data-index,得到的是数字型

示例:

```
//3.数据缓存data() 里面的数据存放在元素的内存里
$('#span').data('uname', 'auue')
console.log($('#span').data('uname'));
//该方法获取data-index h5自定义属性不用写data-且返回的是数字型
console.log($('#span').data('index'));
```

```
<span data-index="5">111</span>
```

span盒子内属性并未改变，类似于存储的是变量'uname'

auue

5

输出自定义属性得到的是数字型

案例：购物车全选

功能：全选和全不选

- ①全选思路:里面3个小的复选框按钮(j-checkbox)选中状态(checked)跟着全选按钮(checkall)走。
- ②因为checked是复选框的固有属性,此时我们需要利用prop()方法获取和设置该属性。
- ③把全选按钮状态赋值给3小复选框就可以了。

通过prop获取全选按钮的状态，再通过prop将其状态赋值给3小复选框

选用change事件

//实现全选和取消全选

```
$('.checkall').change(function () {  
    //全选的状态赋值给小复选框状态,以及下方的全选框  
    $('.j-checkbox,.checkall').prop('checked', $(this).prop('checked'))  
})
```

功能：根据小复选框全被勾选，勾选全选，否则不勾全选

④当我们每次点击小的复选框按钮,就来判断:使用change事件

⑤如果小复选框被选中的个数等于3（直接改成复选框个数，不定死）就应该把全选按钮选上,否则全选按钮不选。

⑥:checked 选择器 :checked查找被选中的表单元素。

可以通过\$('li:checked').length来知道被选中的复选框个数

```
▶ w.fn.init(2) [input.j-checkbox, input.j-checkbox  
▼ w.fn.init(3) [input.j-checkbox, input.j-checkbox  
  ▶ 0: input.j-checkbox  
  ▶ 1: input.j-checkbox  
  ▶ 2: input.j-checkbox  
  length: 3  
  ▶ prevObject: w.fn.init [document]  
  ▶ proto: Object(0)
```

//小复选框改变时

```
$('.j-checkbox').change(function () {  
    //小复选框勾选个数等于复选框个数，则勾选全选框，否则不勾选  
    if ($('.j-checkbox:checked').length == $('.j-checkbox').length) {  
        $('.checkall').prop('checked', true)  
    } else {  
        $('.checkall').prop('checked', false)  
    }  
})
```

<input checked="" type="checkbox"/> 全选	商品
<input checked="" type="checkbox"/>	 【5本2版八十
<input checked="" type="checkbox"/>	 【2000年贴语
<input checked="" type="checkbox"/>	 唐诗三外书 集 选 外 阅
<input checked="" type="checkbox"/> 全选	删除选中的商品

注意：

- 1.复选框事件选用change更合适
- 2.使用:checked选择器查找被选中的表单元素，其长度length就是被选中的个数

7.jQuery内容文本值

7.1基本内容

主要针对**元素的内容**还有**表单的值**操作。

- 1.普通元素内容html() (相当于原生innerHTML)

```
html() //获取元素的内容 包括标签
html("内容") // 设置元素的内容
```

- 2.普通元素文本内容text() (相当与原生 innerText)

```
text() //获取元素的文本内容 纯文本
text("文本内容") // 设置元素的文本内容
```

主要针对元素的内容还有**表单的值**操作。

- 3.表单的值val() (相当于原生value)

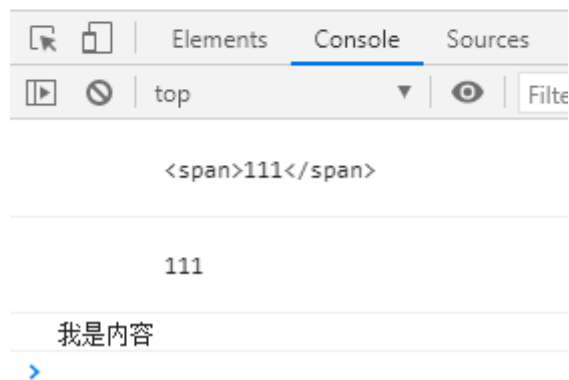
```
val() //获取表单的文本内容
val("文本内容") // 设置表单的文本内容
```

示例:

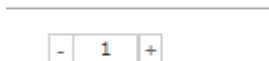
```
<div>
  <span>111</span>
</div>
<input type="text" value="我是内容">
<script>
  $(function () {
    //1. 获取/设置元素内容 html 包括标签
    console.log($('div').html());
    // $('div').html('222')
    //2. 获取/设置元素内容 text 纯文本
    console.log($('div').text());
    $('div').text('333')
    //3. 获取/设置表单值 val
    console.log($('input').val());
    $('input').val('文本')
  })
</script>
```

333

文本

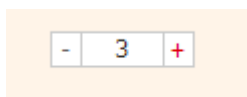


案例:购物车案例模块增减商品数量分析


A UI element consisting of three boxes: a minus sign (-), a text input containing the number 1, and a plus sign (+). The entire element is highlighted with a red border.A UI element consisting of three boxes: a minus sign (-), a text input containing the number 1, and a plus sign (+).A UI element consisting of three boxes: a minus sign (-), a text input containing the number 1, and a plus sign (+).

- ①核心思路:首先声明一个变量,当我们点击+号(increment) , 就让这个值++ ,然后赋值给文本框。
- ②**注意1** :只能增加本商品的数量, 就是当前+号的兄弟文本框(itxt)的值。
- ③修改表单的值是val()方法
- ④**注意2**:这个变量初始值应该是这个文本框的值,在这个值的基础上++,要**先获取表单的值再修改值**
- ⑤减号(decrement)思路同理,但是如果文本框的值是1 ,就**不能再减了, 直接return false, 不执行后面的代码。**

```
//2. 增减商品数量
//1)增加商品数量, 只让当前商品数量增加
$('.increment').click(function () {
    //先获取表单内值再修改
    var n = $(this).siblings('.itxt').val()//
    // console.log(n);
    n++
    $(this).siblings('.itxt').val(n)
})
//2)减少商品数量
$('.decrement').click(function () {
    //先获取表单内值再修改
    var n = $(this).siblings('.itxt').val()
    if (n == 1) {
        return false //数量到1就不能再减了, 使用return结束代码
    }
    n--
    $(this).siblings('.itxt').val(n)
})
```

A UI element consisting of three boxes: a minus sign (-), a text input containing the number 3, and a plus sign (+).

案例: 购物车案例模块修改商品小计分析

全选	商品	单价	数量	小计	操作
	 【5本26.8元】经典儿童文学彩图青少年版八十天环游地球中学生语文教学大纲	¥12.60	- 1 +	¥12.60	删除

①核心思路:每次点击+号或者-号,根据文本框的值乘以当前商品的价格就是商品的小计

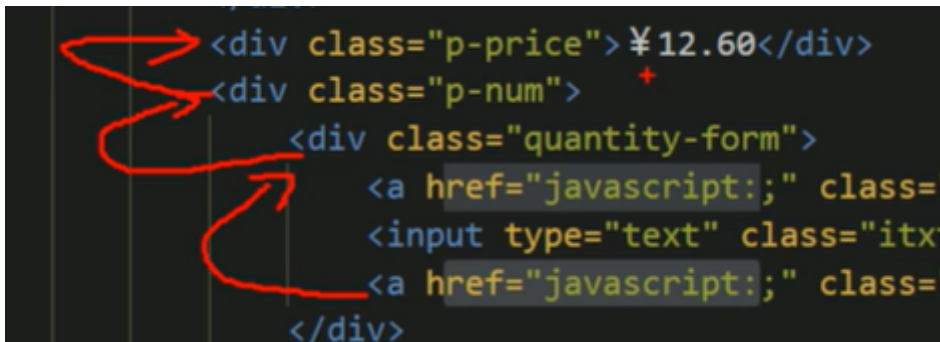
②注意1 :只能增加本商品的小计,就是当前商品的小计模块(p-sum)

当前商品的单价: 如下图1, 是+-标签的爸爸的爸爸的兄弟'p-price'

当前商品的小计模块:如下图2, 是+-标签的爸爸的兄弟'p-sum'

③修改普通元素的内容是text()或html()方法

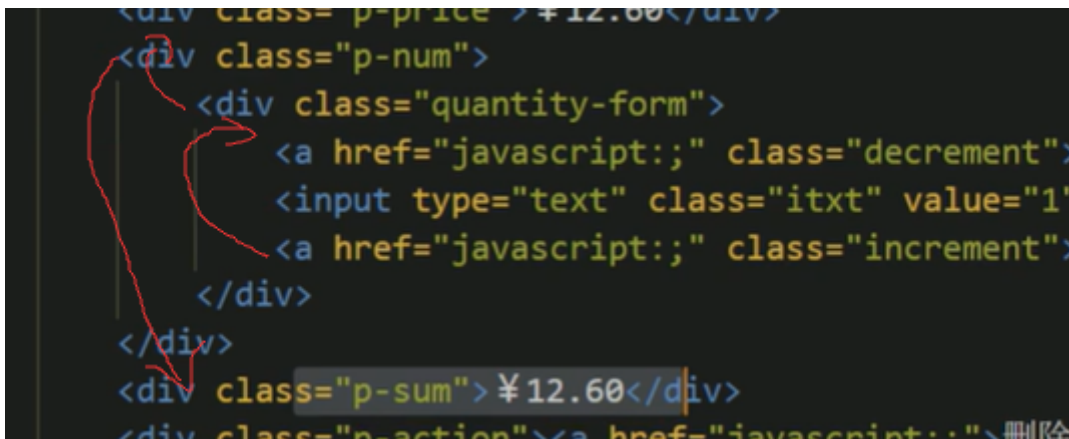
④注意2:当前商品的价格,要把¥符号去掉再相乘截取字符串substr(1), 放入文本内容时需要再加¥符号



```

<div class="p-price">¥12.60</div>
<div class="p-num">
  <div class="quantity-form">
    <a href="javascript:;" class="
    <input type="text" class="itxt
    <a href="javascript:;" class="
  </div>
</div>

```



```

<div class="p-price">¥12.60</div>
<div class="p-num">
  <div class="quantity-form">
    <a href="javascript:;" class="decrement">
    <input type="text" class="itxt" value="1"
    <a href="javascript:;" class="increment">
  </div>
</div>
<div class="p-sum">¥12.60</div>
<div class="p-action"><a href="javascript:;">删除

```

//3. 小计模块

//文本框的值乘以当前商品的价格就是商品的小计

//获取当前商品的单价, 使用爸爸的爸爸的兄弟方式获取到

```
var price = $(this).parent().parent().siblings('.p-price').html()
```

price = price.substr(1)//通过字符截取¥后面的数字

```
// console.log(price);
```

//小计算后赋值给当前商品的小计, 爸爸的爸爸的兄弟

```
$(this).parent().parent().siblings('.p-sum').text('¥' + n * price)
```

¥12.60	- 3 +	¥37.8
¥24.80	- 2 +	¥49.6
¥29.80	- 2 +	¥59.6

祖先选择器和保留小数方法完善小计案例

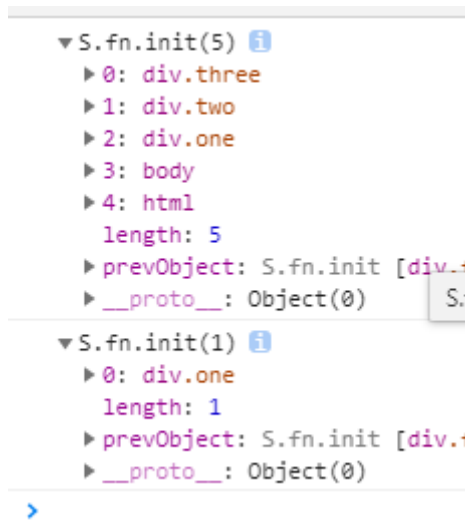
⑤parents('选择器') 可以返回指定祖先元素

⑥最后计算的结果如果想要保留2位小数通过toFixed(2)方法

parents('选择器')使用示例

```
<div class="one">
  <div class="two">
    <div class="three">
      <div class="four">子</div>
    </div>
  </div>
</div>
<script>
  $(function () {
    //返回所有祖先
    console.log($('.four').parents());
    //返回指定祖先
    console.log($('.four').parents('.one'));

  })
</script>
```



使用parents和toFixed()改进案例：

```
// var price = $(this).parent().parent().siblings('.p-price').html()
//使用祖先选择器完善上一行代码
var price = $(this).parents('.p-num').siblings('.p-price').html()

// $(this).parent().parent().siblings('.p-sum').text('¥' + n * price)
//toFixed(x) 方法，保留x位小数
$(this).parents('.p-num').siblings('.p-sum').text('¥' + (n * price).toFixed(2))
```

用户修改文本框的值计算小计

⑦用户也可以直接修改表单里面的值,同样要计算小计。用**表单change事件**

⑧用最新的表单内的值乘以单价即可但是还是当前商品小计

```
//4.用户修改文本框的值计算小计
$('.itxt').change(function () {
    //小计=文本框的值x当前商品单价
    //文本框的值
    var n = $(this).val()
    //当前商品单价
    var price = $(this).parents('.p-num').siblings('.p-price').html()
    price = price.substr(1)
    //小计
    $(this).parents('.p-num').siblings('.p-sum').text('¥' + (n *
price).toFixed(2))
})
```

¥12.60	- 4 +	¥50.40
--------	-------	--------

8.jQuery元素操作

主要是**遍历、创建、添加、删除**元素操作。

8.1遍历元素

jQuery隐式迭代是对同一类元素做了同样的操作。如果想要给同一类元素做不同操作,就需要用到遍历。

语法1：

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

- 1.each()方法遍历匹配的每一个元素。 **主要用DOM处理**。each每一个
- 2.里面的回调函数有2个参数: **index 是每个元素的索引号; domEle是每个DOM元素对象,不是jquery对象**
- 3.所以要想使用jquery方法,需要**将这个dom元素转换为jquery对象\$(domEle)**
- 4.index,domEle是函数参数,可以自己指定名字,经常也为i,ele

示例;

```
<div>1</div>
<div>2</div>
<div>3</div>
<script>
    $(function () {
        //对同一类元素做不同操作，使用遍历
        //1.each()方法遍历元素
        var arr = ['red', 'green', 'blue']
        var sum = 0;
        $('#div').each(function (index, domEle) {
            //回调函数第一个参数是索引号，自己可以指定索引号名称
            console.log(index); //0 1 2
            //回调函数第二个参数是dom元素对象，因此我们需要转化成jQuery对象才能用css方法

            //给三个div添加不同的字体颜色
            $(domEle).css('color', arr[index])
            //将div里面的数字相加，记得先转化为数字型
            sum += parseInt($(domEle).text())
        })
        console.log(sum);
    })
</script>
```

1
2
3

Elements
top
0
1
2
6
.

语法2:

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

- 1.\$each()方法可用于遍历任何对象。主要用于数据处理,比如数组,对象
- 2.里面的函数有2个参数: index 是每个元素的索引号; element遍历内容

```
//2. $.each(arr, function (i, ele){})  
//遍历数组  
$.each(arr, function (i, ele) {  
    console.log(i);  
    console.log(ele);  
})
```

0
red
1
green
2
blue

```
//遍历对象  
$.each({  
    name: 'auue',  
    age: 18  
}, function (i, ele) {  
    console.log(i); //属性名  
    console.log(ele); //属性值  
})
```

name
auue
age
18

案例:购物车案例模块计算总计和总额

- ①核心思路:把所有文本框里面的值相加就是总计数量。总额同理, **使用each遍历**
- ②文本框里面的值不相同,如果想要相加需要用到**each遍历**。声明一个变量即可
- ③点击+号-号,会改变总计和总额; 如果用户修改了文本框里面的值,同样会改变总计和总额; 如果改变复选框也要重新计算金额
- ④因此可以封装一个函数求总计和总额的, 以上2个操作调用这个函数即可.
- ⑤注意1 :总计是文本框里面的值相加用val(), 总额是普通元素的内容用text()
- ⑥**要注意普通元素里面的内容要去掉¥ 并且转换为数字型才能相加**

注意: 添加只有商品被勾选了, 商品件数和金额才被计算

```
//5.计算总额
function getSum() {
    var count = 0//总件数
    var money = 0 //总金额
    //将表单内的数字相加
    $('itxt').each(function (i, ele) {
        // 只计算复选框被勾选的商品件数
        if ($(ele).parents('.cart-item').find('.j-checkbox').prop('checked')) {
            count += parseInt($(ele).val())
        }
    })
    $('.amount-sum em').text(count)
    //将金额相加
    $('.p-sum').each(function (i, ele) {
        //只计算复选框被勾选的商品总金额
        if ($(ele).parents('.cart-item').find('.j-checkbox').prop('checked')) {
            money += parseFloat($(ele).text().substr(1))//去掉¥符号, 转化为浮点型
        }
    })
    $('.price-sum em').text(money.toFixed(2))//, 保留2位小数
}
```

<input checked="" type="checkbox"/>		【5本26.8元】经典儿童文学彩图青少年版八十天环游地球中学生语文教学大纲	¥12.60	- 1 +	¥12.60	删除
<input checked="" type="checkbox"/>		【2000张贴纸】贴纸书 3-6岁 贴画儿童贴画书全卷12册 贴画 贴纸儿童 汽	¥24.80	- 2 +	¥49.60	删除
<input type="checkbox"/>		唐诗三百首+成语故事全2册 一年级课外书 精美注音儿童版 小学生二三年级课外阅读书籍	¥29.80	- 2 +	¥59.60	删除

☐ 全选
 ☐ 删除选中的商品

已经选 3 件商品 总价: **62.20**

8.2创建元素

语法:

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

动态的创建了一个li

8.3添加元素

1.内部添加

```
element.append("内容")
```

把内容放入匹配元素内部**最后面**,类似原生appendChild.

```
element.prepend("内容")
```

把内容放入匹配元素内部**最前面**。

示例:

```
//1.创建元素
var li = $('<li>后来的li</li>')
//2.添加元素
//1) 内部添加
// $('ul').append(li)//放后面
$('ul').prepend(li)//放前面
```

- 后来的li
- 原有的li

2.外部添加

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

①内部添加元素,生成之后,它们是父子关系。

②外部添加元素,生成之后,他们是兄弟关系。

示例:

```
//2) 外部添加
var div = $('<div>后来的div</div>')
// $('div').after(div)//放后面
$('div').before(div)//放前面
```

- 后来的li
- 原有的li

后来的div
原有的div

8.4删除元素

```
element.remove() // 删除匹配的元素(本身)
element.empty() // 删除匹配的元素集合中所有的子节点
element.html('') // 清空匹配的元素内容
```

示例:

```
$('#ul').remove()//删除匹配的元素(本身) 自杀
```

```
▶ <head>...</head>
▼ <body>
  <div>后来的div</div>
  <div>原有的div</div>
```

ul整个都没了

```
// $('#ul').empty()//删除匹配的元素集合中所有的子节点
$('#ul').html('')//清空匹配的元素内容，效果和上一个一样
```

后来的div

ul 1350 × 0

ul还在，里面什么都没有了

```
Elements Console Sources
<!DOCTYPE html>
<html lang="en"> == $0
  <head>...</head>
  <body>
    <ul></ul>
    <div>后来的div</div>
    ... 原有的 div ...
```

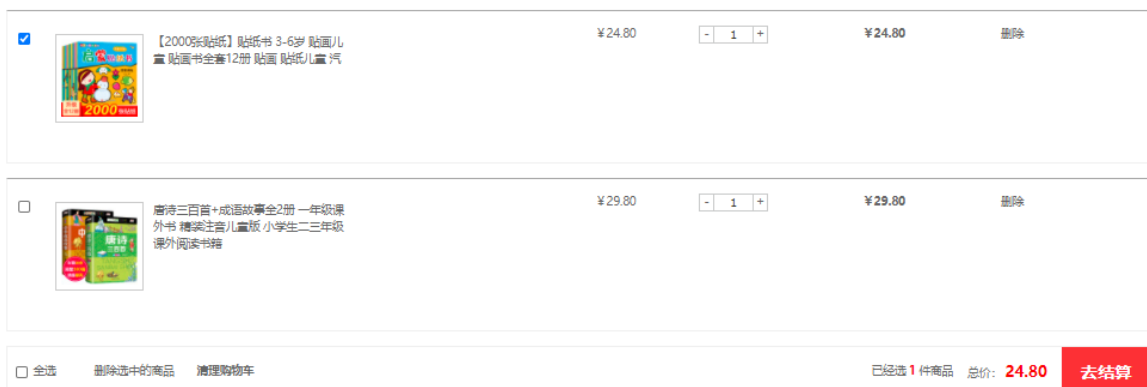
案例:购物车案例模块删除商品模块

①核心思路:把商品remove()删除元素即可

②有三个地方需要删除: 1. 商品后面的删除按钮2.删除选中的商品3.清理购物车, 删除后也要计算商品总件数和总金额

③商品后面的删除按钮: 一定是删除当前的商品,所以从\$(this)出发

```
//6.删除商品
//1) 商品后面的删除按钮
$('.p-action a').click(function () {
    //删除当前的商品,所以从$(this)出发
    $(this).parents('.cart-item').remove()
    //删除后需要重新计算商品总件数和金额
    getSum()
})
//2)删除选中的商品
$('.remove-batch').click(function () {
    //需要删除的是被勾选的小复选框
    $('.j-checkbox:checked').parents('.cart-item').remove()
    getSum()
})
//3)清理购物车, 删除所有商品
$('.clear-all').click(function () {
    $('.cart-item').remove()
    getSum()
})
```



案例:购物车案例模块选中商品添加背景

①核心思路:选中的商品添加背景,不选中移除背景即可

②全选按钮点击:如果全选是选中的,则所有的商品添加背景,否则移除背景

③小的复选框点击:如果是选中状态,则当前商品添加背景,否则移除背景,使用this

④这个背景,可以通过类名修改,添加类和删除类

```
.check-cart-item {
    background: #fff4e8;
}
```

```
//全选，给所有的商品添加背景颜色，添加check-cart-item类，否则移除类
if ($('#.checkall').prop('checked')) {
    $('#.cart-item').addClass('check-cart-item')
} else {
    $('#.cart-item').removeClass('check-cart-item')
}
```

```
if ($(this).prop('checked')) {
    //给当前选中的商品添加背景颜色，添加check-cart-item类，否则移除类
    $(this).parents('.cart-item').addClass('check-cart-item')
} else {
    $(this).parents('.cart-item').removeClass('check-cart-item')
}
```

全部商品

<input checked="" type="checkbox"/> 全选	商品	单价	数量	小计	操作
<input checked="" type="checkbox"/>	 【5本26.8元】经典儿童文学彩图青少年版八十天环游地球中学生语文教学大纲	¥12.60	- 1 +	¥12.60	删除
<input checked="" type="checkbox"/>	 【2000张贴纸】贴纸书 3-6岁 贴画儿童 贴纸书全册12册 贴画 贴纸儿童 汽	¥24.80	- 1 +	¥24.80	删除
<input checked="" type="checkbox"/>	 唐诗三百首+成语故事全2册 一年级课外书 精美注音儿童版 小学生二三年级课外阅读书籍	¥29.80	- 1 +	¥29.80	删除

9.jQuery尺寸、位置操作

9.1 jQuery 尺寸

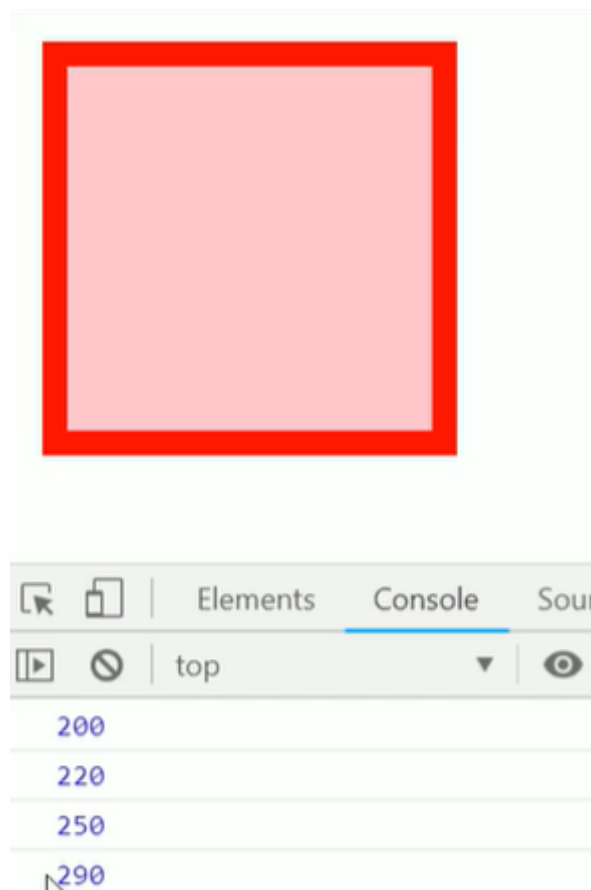
语法	用法
width()/ height()	取得匹配元素宽度和高度值只算width / height
innerWidth() / innerHeight()	取得匹配元素宽度和高度值包含padding
outerWidth() / outerHeight()	取得匹配元素宽度和高度值包含padding、border
outerWidth(true) / outerHeight(true)	取得匹配元素宽度和高度值包含padding、border、margin

- 以上参数为空,则是获取相应值,返回的是数字型。
- 如果参数为数字,则是修改相应值。
- 参数可以不必写单位。

示例：

```
div {  
  width: 200px;  
  height: 200px;  
  background-color: pink;  
  padding: 10px;  
  border: 15px solid red;  
  margin: 20px;  
}
```

```
// 1. width() / height() 获取设置元素width和height大小  
console.log($(".div").width());  
// $(".div").width(300);  
  
// 2. innerwidth() / innerHeight() 获取设置元素width和height + padding 大小  
console.log($(".div").innerwidth());  
  
// 3. outerwidth() / outerHeight() 获取设置元素width和height + padding + border 大小  
console.log($(".div").outerwidth());  
  
// 4. outerwidth(true) / outerHeight(true) 获取设置width和height + padding +  
border +margin  
console.log($(".div").outerwidth(true));
```



9.2 jQuery 位置

位置主要有3个: offset()、 position()、 scrollTop() /scrollLeft ()

1.offset(设置或获取元素偏移)

①offset() 方法**设置或返回**被选元素，相对于**文档**的偏移坐标，跟父级没有关系。

②该方法有2个属性left、top。offset().top 用于获取距离文档顶部的距离, offset().left用于获取距离文档左侧的距离。

③可以设置元素的偏移: offset({ top: 10, left: 30 }); **以对象的形式设置**

可读写

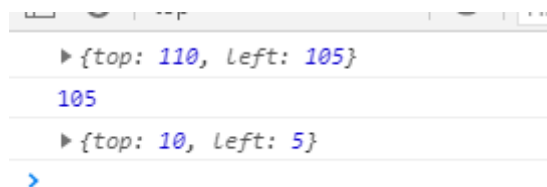
2.position(获取元素偏移)

①position() 方法用于返回被选元素相对于带有定位的父级偏移坐标,如果父级都没有定位,则以文档为准。 **只读**

示例:

```
//1. 获取设置距离文档的位置（偏移） offset
//可读写
console.log($('.son').offset());
console.log($('.son').offset().left);
// $('.son').offset({
//     top: 200,
//     left: 200
// })
//2. 获取距离带有定位父级位置（偏移） position 如果没有带有定位的父级，则以文档为准

//注意只能获取，不能设置，只读
console.log($('.son').position());
```



3.scrollTop()/scrollLeft()(设置或获取元素被卷去的头部和左侧)

①scrollTop()方法设置或返回被选元素被卷去的头部。

②scrollLeft()方法设置或返回被选元素被卷去的左侧。

可读写

示例:

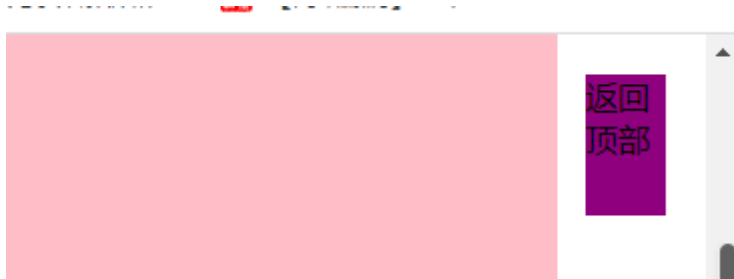
页面被卷去多少就显示返回顶部，否则隐藏

```

//页面滚动事件
$(document).scrollTop(100)//还可以设置被卷去的头部，这样默认刷新页面就被卷去
100

//scrollTop()被卷去的头部 scrollLeft()被卷去的左侧
var boxTop = $('#main').offset().top
$(window).scroll(function () {
    //$(document).scrollTop()页面被卷去的头部
    console.log($(document).scrollTop());
    if ($(document).scrollTop() >= boxTop) {
        $('.back').fadeIn()
    } else {
        $('.back').fadeOut()
    }
})

```



滚完main上方区域再显示，否则隐藏

案例:带有动画的返回顶部

- ①核心原理:使用animate动画返回顶部。
- ②animate动画函数里面有个scrollTop 属性,可以设置位置
- ③但是只能是元素做动画,因此\$("body,html").animate(scrollTop: 0)), 不能写document

```

//返回顶部
$('.back').click(function () {
    // $(document).scrollTop(0)//可以返回顶部，但没有返回动画
    //使用animate动画函数返回顶部，注意是元素做动画，不能写document
    $('body,html').animate({
        scrollTop: 0
    })
})

```

案例:品优购电梯导航,导航控制内容

- ①当我们滚动到今日推荐模块,就让电梯导航显示出来
- ②点击电梯导航页面可以滚动到相应内容区域
- ③核心算法:因为电梯导航模块和内容区模块——对应的，通过小li的索引号求当前索引号内容区模块它的offset().top即移动距离
- ④当我们点击电梯导航某个小模块,就可以拿到当前小模块的索引号
- ⑤就可以把animate要移动的距离求出来:当前索引号内容区模块它的offset().top

```

//电梯导航
//1. 滚动到今日推荐模块才显示电梯导航，否则隐藏

```

```

var toolTop = $('.recom').offset().top
$(window).scroll(function () {
    if ($(document).scrollTop() >= toolTop) {
        $('.fixedtool').fadeIn()
    } else {
        $('.fixedtool').fadeOut()
    }
})
//点击电梯导航页面可以滚动到相应内容区域
$('.fixedtool li').click(function () {
    //获得当前li的索引号，来选出对应内容区模块，并得到移动的距离offset().top 思路类似
    //tab栏切换
    var top = $('.floor .w').eq($(this).index()).offset().top
    //页面滚动效果
    $('body,html').stop().animate({
        scrollTop: top
    })
})

```



小bug: 页面刷新后导航栏消失

原因: 当初设置导航栏隐藏和显示是在页面滚动后才触发的，页面刷新并没有滚动页面

解决方法: 将页面滚动事件内的代码封装成函数，在滚动事件前先调用一次

```

toggleTool() //先调用一次，防止页面刷新后导航消失
function toggleTool() {
    if ($(document).scrollTop() >= toolTop) {
        $('.fixedtool').fadeIn()
    } else {
        $('.fixedtool').fadeOut()
    }
}
$(window).scroll(function () {
    toggleTool()
})

```

案例:品优购电梯导航，内容控制导航

- ①当我们点击电梯导航某个小li,当前小li添加current类,兄弟移除类名
- ②当我们页面滚动到内容区域某个模块,左侧电梯导航,相对应的小li模块,也会添加current类,兄弟移除current类。
- ③触发的事件是页面滚动,因此**这个功能要写到页面滚动事件里面**。
- ④需要用到each,遍历内容区域大模块。each里面能拿到内容区域每一个模块元素和索引号
- ⑤判断的条件:被卷去的头部大于等于内容区域里面每个模块的offset().top
- ⑥就利用这个索引号找到相应的电梯导航小li添加类。

```
$(window).scroll(function () {
    toggleTool()
    //3. 页面滚动到某个内容区域，左侧电梯导航li相应添加和删除current类
    $('.floor .w').each(function (i, ele) {
        if ($(document).scrollTop() >= $(ele).offset().top) {
            // console.log(i); //i记录索引号
            $('.fixedtool
li').eq(i).addClass('selected').siblings('li').removeClass('selected')
        }
    })
})
```

小bug: 点击第一个li后, 第三个li红色消失, 第二个li会变红再消失, 然后第一个li变红。而不是直接第一个li变红



原因: 点击li后页面必定会滚动, 滚动就会触发滚动事件中途就会触发上方案例里的代码给li添加背景

解决方法: 当我们点击了小li此时不需要执行页面滚动事件里面的li的背景选择添加current。使用节流阀控制滚动事件里内容区域对应li显示背景颜色功能, 点击事件触发后关闭水龙头, 在animate动画中添加回调函数再打开水龙头

```
$(function () {
    //电梯导航
    // 当我们点击了小li此时不需要执行页面滚动事件里面的li的背景选择添加current
    var flag = true;
    //1. 滚动到今日推荐模块才显示电梯导航, 否则隐藏
    var toolTop = $('.recom').offset().top
    toggleTool() //先调用一次, 防止页面刷新后导航消失
    function toggleTool() {
        if ($(document).scrollTop() >= toolTop) {
            $('.fixedtool').fadeIn()
        } else {
            $('.fixedtool').fadeOut()
        }
    }
})
```

```

    }
    $(window).scroll(function () {
        toggleTool()
        //3. 页面滚动到某个内容区域，左侧电梯导航li相应添加和删除current类
        if (flag) {
            $('.floor .w').each(function (i, ele) {
                if ($(document).scrollTop() >= $(ele).offset().top) {
                    // console.log(i); //i记录索引号
                    $('.fixedtool
li').eq(i).addClass('selected').siblings('li').removeClass('selected')
                }
            })
        }
    })
    //2. 点击电梯导航页面可以滚动到相应内容区域
    $('.fixedtool li').click(function () {
        flag = false
        //获得当前li的索引号，来选出对应内容区模块，并得到移动的距离offset().top 思路类似
        tab栏切换
        var top = $('.floor .w').eq($(this).index()).offset().top
        //页面滚动效果
        $('body,html').stop().animate({
            scrollTop: top
        }, function () {
            flag = true
        })
    })
})

```

10.jQuery事件

10.1事件注册

单个事件注册

语法:

```

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

```

其他事件和原生基本一致。

比如mouseover. mouseout. blur. focus. change. keydown. keyup. resize. scroll等

10.2jQuery事件处理

10.2.1事件处理on()绑定事件

on()方法在匹配元素上绑定一个或多个事件的事件处理函数

语法:

```

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

```

1.events:一个或多个用空格分隔的事件类型,如"click"或"keydown".

2.selector:元素的子元素选择器。

on()方法优势1：

可以绑定多个事件,多个处理事件处理程序。

示例：

```
//事件处理on 处理多个事件
//1)对象形式处理多个事件，事件类型：过程处理函数 之间用逗号隔开
/* $('div').on({
    mouseenter: function () {
        $(this).css('backgroundColor', 'red')
    },
    click: function () {
        $(this).css('backgroundColor', 'blue')
    },
    mouseleave: function () {
        $(this).css('backgroundColor', 'green')
    }
}) */
//2)空格隔开多个事件类型，触发同一个过程处理函数 适用于切换类
$('div').on('mouseenter mouseleave', function () {
    $(this).toggleClass('current')
})
```

on()方法优势2：

可以事件委派操作。事件委派的定义就是,把原来加给子元素身上的事件绑定在父元素身上,就是把事件委派给父元素。

在此之前有bind(), live(), delegate()等方法来处理事件绑定或者事件委派,最新版本的请用on替代他们。

```
//优势2：事件委派
//将事件绑在ul父亲上，触发的对象是li孩子，通过孩子触发事件冒泡让ul父亲触发事件
//以前的做法
/* $('ul li').click(function () {
    $(this).css('background', 'red')
}) */
$('ul').on('click', 'li', function () {
    $(this).css('background', 'red')
})
```

- li孩子
- li孩子
- li孩子
- li孩子
- li孩子
- li孩子

on()方法优势3：

动态创建的元素, click()没有办法绑定事件, on()可以给未来动态生成的元素绑定事件

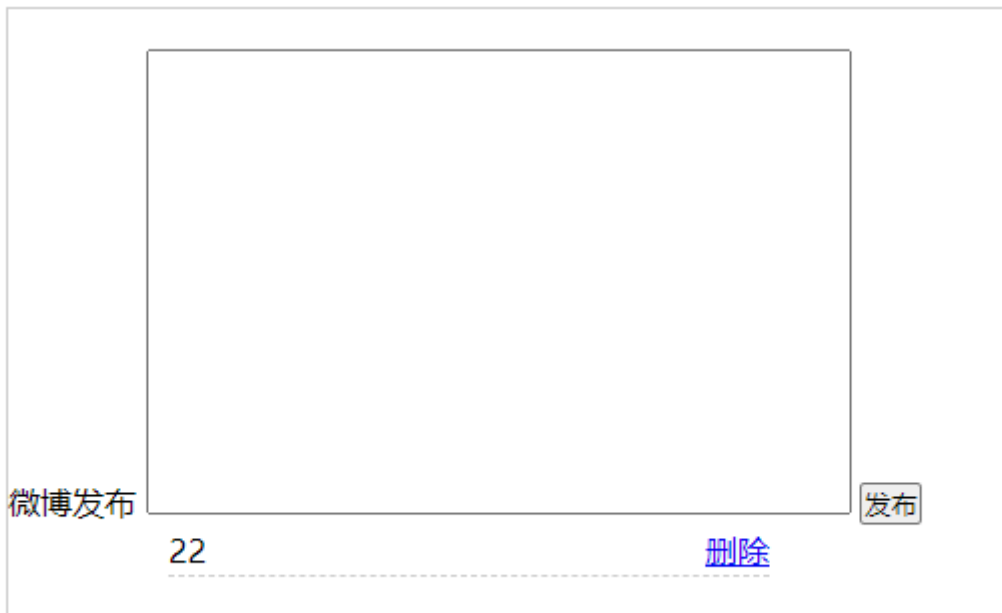
//优势3: 可以给未来动态创建的元素绑定事件

```
/* $('ol li').click(function () {
    $(this).css('color', 'red')
}) */
$('ol').on('click', 'li', function () {
    $(this).css('color', 'red')
})
var li = $('<li>后来的</li>')
$('ol').append(li)
```

1. 后来的

这样就不用给动态创建的元素再额外添加事件了, 且不需要一定创建放在绑定事件前面, 可以先绑定事件后创建元素。

案例:发布微博案例



案例分析

①点击发布按钮,动态创建一个小li,放入文本框的内容和删除按钮,并且添加到ul中。

②点击的删除按钮,可以删除当前的微博留言。

//1. 点击发布按钮,动态创建一个小li,放入文本框的内容和删除按钮,并且添加到ul中

```
$('#button').on('click', function () {
    var li = $('<li></li>')
    li.html($('textarea').val() + '<a href="javascript:;">删除</a>')
    $('ul').prepend(li)
    li.slideDown() //让li滑动出现,需要先添加li再添加滑动效果,并且需要对li事先设置隐藏

    $('textarea').val('') //清空文本域
})
```

//2. 点击删除按钮,删除当前的微博留言

//注意使用on才能给动态创建的元素绑定事件

```
$('#ul').on('click', 'a', function () {
    $(this).parent().slideUp(function () { //这个this指的是当前的a
```

```
$(this).remove();//这个this指的是当前的li，利用回调函数，让其滑动结束后再进行删除
    })//删除按钮所在的li删除，滑动只是隐藏，不能删除
  })
```

注意：

- 1.为了让发布的内容滑动出现，需要在css先对li设置隐藏display:none; 事件中先添加li再添加滑动效果
- 2.注意使用on才能给动态创建的元素绑定事件
- 3.为了让发布的内容滑动消失，需要利用滑动方法里的回调函数，在回调函数内删除
- 4.textarea获取值通过val，而不是html

```
$(this).parent().slideUp(function () { //这个this指的是当前的a
    $(this).remove();//这个this指的是当前的li，利用回调函数，让其滑动结束后再进行删除
  })//删除按钮所在的li删除，滑动只是隐藏，不能删除
```

10.2.2事件处理off()解绑事件

off()方法可以移除通过on()方法添加的事件处理程序。

如果有的事件只想触发一次，可以使用one()来绑定事件。

```
$('#div').on({
  click: function () {
    console.log('点击');
  },
  mouseenter: function () {
    console.log('经过');
  }
})
// $('#div').off()//解绑div元素所有事件处理程序
$('#div').off('click')//解绑div元素的点击事件
$('#ul').on('click', 'li', function () {
  $(this).css('color', 'red')
})
$('#ul').off('click', 'li')//解绑事件委托
$('#p').one('click', function () {
  console.log('pp');
})
```

10.2.3自动触发事件trigger()

有些事件希望自动触发比如轮播图自动播放功能跟点击右侧按钮一致。可以利用定时器自动触发右侧按钮点击事件,不必鼠标点击触发。

```
element.click() // 第一种简写形式,更常用
```

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

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


示例:

```
$('#div').on('click', function () {
    alert('11')
})
//自动触发事件
//1.元素.事件
// $('#div').click()
//2.元素.trigger('事件')
// $('#div').trigger('click')
//3.元素.triggerHandler('事件') 不会触发元素的默认行为, 如光标
// $('#div').triggerHandler('click')
$('#input').on('focus', function () {
    $(this).val('你好')
})
$('#input').focus()//会光标闪烁, 如图1
// $('#input').triggerHandler('focus')//不会光标闪烁, 如图2
```



注意: 第三种方法不会触发元素的默认行为, 如光标, 其余两种会

10.3jQuery事件对象

事件被触发,就会有事件对象的产生。

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

阻止默认行为: event.preventDefault()或者return false

阻止冒泡: event.stopPropagation()

10.4jQuery对象拷贝

如果想要把某个对象拷贝(合并)给另外一个对象使用,此时可以使用\$.extend ()方法

语法:

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

1.deep:如果设为true为深拷贝, 默认为false浅拷贝

2.target:要拷贝的目标对象

3.object1:待拷贝到第一个对象的对象。

4.objectN:待拷贝到第N个对象的对象。

5.浅拷贝是把被拷贝的对象复杂数据类型中的地址拷贝给目标对象,修改目标对象会影响被拷贝对象。

6.深拷贝,前面加true,完全克隆(拷贝的对象而不是地址),修改目标对象不会影响被拷贝对象。

深拷贝把里面的数据完全复制一份给目标对象,如果里面有不冲突的属性,会合并到一起

```
var targetobj = {
  id: 0
};
var obj={
  id: 1,
  name: 'ahdy'
};
// $. extend(target, obj);
$.extend(targetobj, obj);
console.log(targetobj); // 会覆盖targetobj里面原来的数据,输出结果就是obj对象内容
```

```
var targetobj = {
  id: 0
};
var obj={
  id: 1,
  name: "andy",
  msg: {
    age: 18
  }
}
$.extend(targetobj, obj);
//1.浅拷贝把原来对象里面的复杂数据类型地址拷贝给目标对象
targetobj.msg.age = 20;
console.log(targetobj); //对象中年龄值为20
console.log(obj); //对象中年龄值为20
```

```
//2.深拷贝把里面的数据完全复制一份给目标对象如果里面有不冲突的属性,会合并到一起
$.extend(true, targetobj, obj);
targetobj.msg.age = 20;
console.log(targetobj); //年龄值为20
console.log(obj); //年龄值为18, 不变
```

11.jQuery多库共存

问题概述:

jQuery使用\$作为标示符,随着jQuery的流行其他js库也会用这\$作为标识符,这样一起使用会引起冲突。

客观需求:

需要一个解决方案,让jQuery和其他的js库不存在冲突,可以同时存在,这就叫做多库共存。

jQuery解决方案:

1.如果\$符号冲突,就使用jQuery

把里面的\$符号统一改为jQuery.比如jQuery("div")

2.让jQuery释放对\$权限, 让用户自己定义

jQuery变量规定新的名称: \$.noConflict() var xx = \$.noConflict();

```
var suibian = $.noConflict();//suibian就称为新的名称
console.log(suibian("span"));
suibian.each()
```

12.jQuery插件

jQuery功能比较有限,想要更复杂的特效效果,可以借助于jQuery插件完成。

注意:这些插件也是依赖于jQuery来完成的,所以必须要先引入jQuery文件,因此也称为jQuery插件。

jQuery插件常用的网站: 免费, 且相应页面有使用教程, 更建议用第二个, 无需登录

1.jQuery插件库<http://www.jq22.com/>

2.jQuery之家<http://www.htmleaf.com/>

jQuery插件使用步骤:

1.引入相关文件。(jQuery文件和插件文件)

2.复制相关html. css. js (调用插件)。建议从demo.html里查看源代码中复制html、css文件

jQuery插件使用(简单概括使用: 赋值粘贴修改内容)

1.**图片懒加载**(图片使用延迟加载在可提高网页下载速度。它也能帮助减轻服务器负载)

当我们页面滑动到可视区域,再显示图片。

我们使用jquery插件库EasyLazyload. 注意,此时的js引入文件和js调用必须写到DOM元素(图片)最后面
根据页面中的使用方法来做

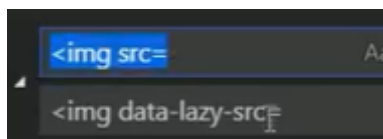
2.**全屏滚动(fullpage.js)**

gitHub : <https://github.com/alvarotio/fullPage.js>

中文翻译网站: <http://www.dowebok.com/demo/2014/77/>

VScode快捷键:

ctrl+H 替换



bootstrap JS插件:

bootstrap框架也是依赖于jQuery开发的,因此里面的js插件使用,也必须引入jQuery文件。

案例: todolist

实现:





需求分析

- ①文本框里面输入内容,按下回车,就可以生成待办事项。
- ②点击待办事项复选框,就可以把当前数据添加到已完成事项里面。
- ③点击已完成事项复选框,就可以把当前数据添加到待办事项里面。

但是本页面内容刷新页面不会丢失。 localStorage

案例分析

- ①刷新页面不会丢失数据,因此需要用到**本地存储localStorage**
- ②核心思路:不管按下回车,还是点击复选框,都是把本地存储的数据加载到页面中,这样保证刷新关闭页面不会丢失数据
- ③**存储的数据格式: `var todolist= [{ title: 'xx', done: false}]`**, 使用对象来存储数据

功能实现

1.toDoList按下回车把新数据添加到本地存储里面

- ①切记:页面中的数据,都要从**本地存储**里面获取,这样刷新页面不会丢失数据,所以先要把数据保存到本地存储里面。
- ②利用事件对象.keyCode判断用户按下回车键(13)。

③声明一个数组,保存数据。

④**先要读取本地存储原来的数据**(声明函数getData()), 放到这个数组里面。

⑤之后把最新从表单获取过来的数据,追加到数组里面。

⑥最后把数组存储给本地存储(声明函数saveDate())

关键点:

1.使用**数组-对象的格式存储数据**, 对象里存放内容与done(任务是否完成)。

由于**本地存储没有直接追加内容的方法, 通过中介数组实现**。将最新的数据追加给数组, 再将数组存储到本地存储 (实则是数据替换, 不算追加) 。

2.注意**本地存储只能存储字符串**, 存储时需要使用**JSON.stringify**转为字符串, 读取出来的数据使用**JSON.parse()**转为我们需要的对象数据格式

3.需要多次读取/存储数据, 将这些代码块封装为函数

```
$(function () {  
    // 1. todoList 按下回车把新数据添加到本地存储里面  
    // 1) 利用事件对象.keyCode判断用户按下回车键(13)。  
    $('add').on('keyup', function (e) {  
        if (e.keyCode == 13) {  
            // 2) 读取本地存储原来的数据  
            // 如果本地存储有数据, 得到一个数组, 没有数据也要得到一个数组, 为空数组  
            var local = getData()  
            // console.log(local);  
            // 由于本地存储没有直接追加内容的方法, 通过中介数组实现: (所以就算本地存储没有数据也要返回数组)  
            // (1) 对local数组进行数据更新, 把最新的数据追加给数组, 注意数据格式是对象格式  
            local.push({ title: $(this).val(), done: false }) //title为用户输入的任务内容,done为是否完成, 默认都是未完成  
            // (2) 将数组存储给本地存储 此处单独封装成函数  
            saveData(local)  
        }  
    })  
  
    // 读取本地存储的数据 由于需要多次读取, 因此单独封装成函数, 减少代码量  
    function getData() {  
        var data = localStorage.getItem('todoList')  
        // 判断本次存储下是否有数据, 有则返回相应的数组, 无则返回空数组  
        if (data != null) {  
            // 注意本地存储的数据是字符串形式, 我们需要的是对象格式, 需要进行转换  
            return JSON.parse(data)  
        } else {  
            return []  
        }  
    }  
    // 存储本地存储的数据  
    function saveData(data) {  
        // 注意: 本地存储只能存储字符串类型, 因此需要将传入的参数转为字符串类型  
        localStorage.setItem('todoList', JSON.stringify(data))  
    }  
})
```

Key	Value
todoList	[{"title":"1","done":false}, {"title":"2","done":false}]

2.todoList本地存储数据渲染加载到页面

①因为后面也会经常渲染加载操作,所以声明一个函数load,方便后面调用

②先要读取本地存储数据。

③之后遍历这个数据(\$.each()), 有几条数据,就生成几个小li添加到ol里面。

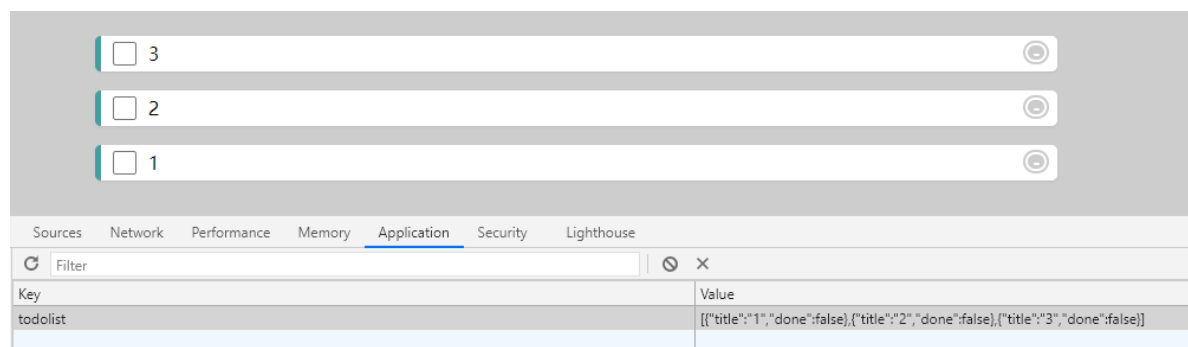
④每次渲染之前,先把原先里面ol的内容清空,然后渲染加载最新的数据。

关键点:

1.用户第一次进入页面也能看到之前列表内容, 因此需要**每次打开页面, 自动调用渲染函数**

2.ol的内容清空 \$('ol').empty()

```
// 每次打开页面都自动渲染一次数据
load()
// 2.数据渲染到页面
function load() {
  // 1)先要读取本地存储数据
  var data = getData()
  // console.log(data);
  // 4)每次渲染前要将ol内数据清空, 即清空每次打开页面渲染出来的数据
  $('ol').empty()
  // 2)遍历这个数据( $.each()) , 有几条数据,就生成几个小li添加到ol里面
  $.each(data, function (i, ele) {
    // 将最新数据放到最前面
    $('ol').prepend('<li><input type="checkbox" name="" id=""><p>' +
ele.title + '</p><a href="javascript:;"></a></li>')
  })
}
```



3.todoList删除操作

①点击里面的a链接,删除的不是li,而是删除本地存储对应的数据。

②核心原理:先获取本地存储数据,删除对应的数据,保存给本地存储,重新渲染列表li

③我们可以给链接自定义属性记录当前的索引号

④根据这个索引号删除相关的数据---数组的splice(i, 1)方法

⑤存储修改后的数据,然后存储给本地存储

⑥重新渲染加载数据列表

⑦因为a是动态创建的,我们使用on方法绑定事件

关键点:

1.删除本地存储对应的数据，本地存储的数据是不能直接删除的，通过删除数组里对应的数据，再存储到本地数据内实现。

2.点击a进行删除，可以使用事件委托给ol，而不是ol li。原因：如下代码块,a始终是li的第三个孩子，即索引号始终是2，我们需要的是当前a所在li的索引号。

3.获得当前a所在li的索引号，

`$(this).index()`获得索引号方法，只针对亲儿子。我们这里a是ol的孙子，无法直接使用

```
<ol class="todoList">
  <li>
    <input type="checkbox" name="" id="">
    <p>11</p>
    <a href=""></a>
  </li>
</ol>
```

解决方法：

1) 使用parent() ——不好用，获得的索引号与数组的索引号不对应，如[1,2,3]，数组索引为0，1，2，但获取过来的索引为2，1，0（因为li是添加到最前面）；除此之外利用索引号将任务归入“正在进行”与“已完成”列表，问题很复杂。

```
var index = $(this).parent().index()
```

2) 在生成li的时候给a添加自定义索引号id=i，通过获得自定义属性（attr）得到当前的索引号

```
$('#ol').prepend('<li><input type="checkbox" name="" id=""><p>' + ele.title +
'</p><a href="javascript:;" id=' + i + '></a></li>')
```

4.数组的splice(i, 1)方法，splice(从哪个位置开始删除-索引号，删除几个元素)

```
// 3.todoList删除操作
$('#ol').on('click', 'a', function () {
  // 1)先获取本地存储数据
  var data = getData()
  // 2)删除数组中对应的数据
  // 获得当前a所在li的索引号parent()
  var index = $(this).attr('id')
  // console.log(index);
  // splice(从哪个位置开始删除-索引号，删除几个元素)
  data.splice(index, 1)
  // 3)存储本地存储的数据
  saveData(data)
  // 4)重新渲染页面
  load()
})
```

删除“3”



4.todoList正在进行和已完成选项操作

- ①当我们点击了小的复选框,修改本地存储数据,再重新渲染数据列表。
- ②点击之后,获取本地存储数据。
- ③修改对应数据属性done为当前复选框的checked状态。
- ④之后保存数据到本地存储
- ⑤重新渲染加载数据列表
- ⑥load 加载函数里面,添加一个判断条件,通过done里面的值判断任务是否完成,如果当前数据的done为true 就是已经完成的,就把列表渲染加载到ul里面, 否则加载到ol里

关键点:

- 1.这里给勾选checkbox后,不是将正在进行的任务ol里的li移动到已经完成列表ul里。**核心思路:通过修改本地存储数据,再重新渲染数据列表实现,渲染时通过done里面的值来判断放在ol还是ul里**
- 2.在load 加载函数里的**遍历函数中判断done的值**,从而决定放入ul/ol,注意同时修改里面的checked属性,因为默认是不勾选的

```
$.each(data, function (i, ele) {  
    if (ele.done) {  
        // 将最新数据放到最前面  
        // 完成放入“已经完成”列表ul里,注意修改里面的checked  
        $('ul').prepend('<li><input type="checkbox" name="" id=""  
checked="checked"><p>' + ele.title + '</p><a href="javascript:;" id=' + i + '>-</a></li>')  
    } else {  
        // 没完成放入ol里  
        $('ol').prepend('<li><input type="checkbox" name="" id=""><p>' +  
ele.title + '</p><a href="javascript:;" id=' + i + '>-</a></li>')  
    }  
})
```

- 3.清空ul列表内的任务,否则加载的时候会重复添加

```
// 4)每次渲染前要将ol内数据清空,即清空每次打开页面渲染出来的数据  
// ul内数据也要清空  
$('ol,ul').empty()
```

- 4.需要获得当前checkbox所在li的索引号,通过兄弟a的自定义属性获取,兄弟的就是我的索引号

```
// 4.todoList正在进行和已完成选项操作  
// 使用并集选择器,两个列表里的checkbox属性都要判断
```



```

$('ul,ol').on('click', 'input', function () {
    // 1) 获取本地存储数据
    var data = getData()
    // 2) 修改数据
    // 需要获得当前checkbox所在li的索引号,通过兄弟a的自定义属性,兄弟的就是我的索引号
    var index = $(this).siblings('a').attr('id')
    // console.log('input' + index);
    // 得到并修改当前checkbox的状态
    data[index].done = $(this).prop('checked');
    // 3) 存储本地存储的数据
    saveData(data)
    // 4) 重新渲染页面
    load()
})

```

5.todoList统计正在进行个数和已经完成个数

- ① 在我们load函数里面操作
- ② 声明2个变量: todoCount待办个数doneCount已完成个数
- ③ 当进行遍历本地存储数据的时候,如果数据done为false, 则todoCount+ +,否则doneCount++
- ④ 最后修改相应的元素text()

```

// 5.todoList统计正在进行个数和已经完成个数
var todoCount = 0
var doneCount = 0
$.each(data, function (i, ele) {
    if (ele.done) {
        // 将最新数据放到最前面
        // 完成放入“已经完成”列表ul里,注意修改里面的checked
        $('ul').prepend('<li><input type="checkbox" name="" id="" checked="checked"><p>' + ele.title + '</p><a href="javascript:;" id=' + i + '>-</a></li>')
        doneCount++
        console.log(doneCount);
    } else {
        // 没完成放入ol里
        $('ol').prepend('<li><input type="checkbox" name="" id=""><p>' + ele.title + '</p><a href="javascript:;" id=' + i + '>-</a></li>')
        todoCount++
    }
});
$('.todocount').text(todoCount)
$('.donecount').text(doneCount)

```



细节完善：

输入为空回车时，弹出对话框，不为空时才加到任务列表里;回车后清空表单内容

```
if (e.keyCode == 13) {
    if ($(this).val() == '') {
        alert('请输入内容')
    } else {
        // 2) 读取本地存储原来的数据
        // 如果本地存储有数据，得到一个数组，没有数据也要得到一个数组，为空数组
        var local = getData()
        // console.log(local);
        // 由于本地存储没有直接追加内容的方法，通过中介数组实现：（所以就算本地存储没有数据也要返回数组）
        // （1）对local数组进行数据更新，把最新的数据追加给数组,注意数据格式是对象格式
        local.push({ title: $(this).val(), done: false }) //title为用户输入的任务内容,done为是否完成，默认都是未完成
        // （2）将数组存储给本地存储 此处单独封装成函数
        saveData(local)

        // 2.todolist本地存储数据渲染加载到页面
        load()

        // 清空表单内数据
        $(''.add').val('')
    }
}
```