0613 jQuery

用更少的代码实现更多的功能

What？

一个优秀的函数库，在行业内叫做框架，一个优秀的js函数库

库是量词，比如有一车苹果，把苹果放仓库，这个仓库就是库

jQuery定义了很多方法，要用就直接调，是提前写好的方法

Why？

HTML元素选取，这是jQuery里面最强大的

HTML元素操作

css操作

HTML事件处理，跟js是一样

js动画效果

链式调用box.fn1().fn2()

读写合一

既能读，又能写

box.html()，获取元素的标签体文本，如果想设置里面的文本，括号里面加字符串，不传值为读，传值为写

浏览器兼容，解决了很多ie不兼容的问题

How？

本质就是个js文件，作为单独的js文件，要用它就要引入

引入js库

$是核心函数，如果报这个错就是js没引入

使用jQuery

jQuery核心函数

区别2中js库库文件

开发版(测试版)

生产版(压缩版)，真正上线用压缩的

引入2种方式

本地的路径

开发测试时使用，加重了服务器负担，上线一般不使用这种方法

CDN远程库，企业付费给CDN运行商，需要保密的放在本地引，不重要的放CDN，服务器会优先访问离得近的服务器

区别jQuery的不同版本

1.x

兼容老版本ie

文件更大

2.x

部分ie8及以下不支持

为了文件更小，执行效率更高

3.x

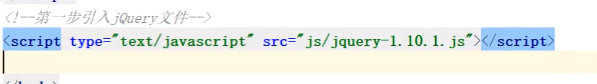
完全不在支持ie8及以下版本

提供了一些新的API

如果有人问3版本，就说没用过

## **初识jquery**

第一步引入jQuery文件，我们写的文件依赖于js文件



加载完成的回调

$()函数调用，这个就约等于window.onload



获取按钮绑定单击事件

$('')，跟css选择器一样

=function，全是传function

前边是对象，后面是方法

IMG_258

获取输入框的value

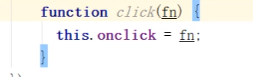
IMG_259

在jQuery中，没用DOM获取就不是DOM对象，而是jQuery对象

DOM是浏览器和js之间的纽带，有了DOM，js才能操作元素

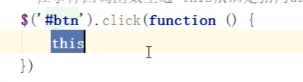
jQuery是js写的，不能超过js，核心底层是依赖原生js写的，我们所看到的通过选择器查询或者是事件的方法依然原生的技术实现的

这里是把DOM单独包装 一下



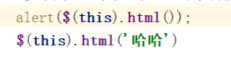
$就是jQuery

在事件里面，this依然指向DOM对象



$(this)把dom对象转化成jQuery对象

上面是读，下面是写

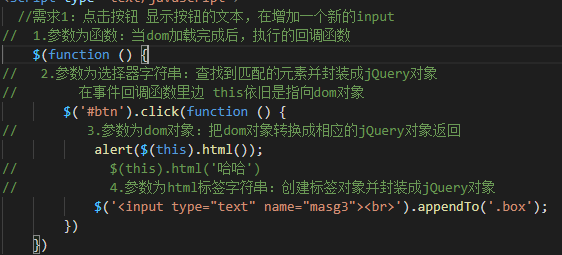


创建标签对象封装成jQuery对象



appendTo()在结尾增加

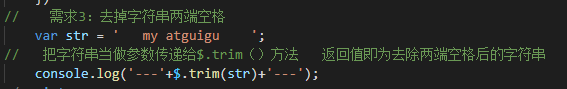
jQuery4种传参方式



jQuery遍历数组



jQuery去掉两端空格，返回结果是去除空格的字符串



有一个方法，原生没有，但是要用， 就用jQuery对象，看什么属性和方法决定前面的操作者是谁

所有通过jQuery获取的元素对应的变量，需要以$开头，用以标记是jQuery对象和dom对象区分IMG_267

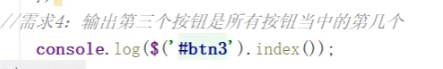
jQuery对象和dom对象的互转

jQuery =>dom 下标取值或者遍历

dom => jQuery $(this)

$.each()//通用遍历方法，用来遍历数组或者对象的  
$btns.each()//是专门用来遍历jQuery对象的  
​  
this.innerHTML//this指向每一次拿出来的dom元素  
​

index()是用来获取当前兄弟元素当中的索引



jQuery的方法 内置隐式遍历

过滤选择器先找第一个集合$('div:first')

最后一个$('div:last')

$('div:not(.box)')否定选择器

$('li:eq(1)') 索引为1的li，获取第N个元素

$('li:gt(0):lt(2)大于第一个小于最后一个就剩下中间的元素了

:gt和:lt都是不包含给定索引的，jQuery的选择器是从左往右依次执行，用上一次的执行结果来执行下一个选择器

选择内容为BBB的li

$('li:hidden').show(时间)隐藏的li

$('li[title]').css('background,颜色')选择有title属性的li元素

$('li[title=hello]')选择title属性为Hellow的li元素

## **jQuery核心函数**

<body>  
    <div class="box">  
        <button>测试</button><br />  
        <input type="text"><br />  
        <input type="text">  
    </div>  
    <!-- 引入jQuery文件 -->  
    <script src="../js/jquery-1.10.1.js"></script>  
    <script>  
        // 1.函数当参数传  
        $(function(){  
            // 2.参数为选择器字符串，封装成jQuery对象，给按钮绑定单击事件  
            $('button').click(function(){  
                // 3.参数为dom对象，把dom对象转化成响应的jQuery  
                // alert($(this).html());//读  
                alert($(this).html('我是1'))//写  
                // 4.参数为html标签字符串，  
                $('<input type="text">').appendTo('.box');  
           })  
       })  
​  
        //遍历数组  
        var arr = [3,2,34,2,54];  
        //$作为对象调用each方法  
        $.each(arr,function(index,item){  
            console.log(index,item);  
       })  
​  
        // 去除两端空格  
        var str = '   helloword   ';  
        console.log('--' + $.trim(str) + '--');  
    </script>  
</body>

## **jQuery核心对象**

<body>  
    <p>hahaha</p>  
    <button>按钮一</button>  
    <button>按钮二</button>  
    <button data-index="2">按钮四</button>  
    <button id="btn3">按钮三</button>  
    <!-- 引入jQuery文件 -->  
    <script src="../js/jquery-1.10.1.js"></script>  
    <script>  
         //需求1：统计一共多少个按钮  
         var $btns = $('button');  
         console.log($btns.length);  
        console.log($btns.size());  
          
        //需求2：取出第二个button的文本  
        console.log($btns[1].innerHTML);//此时的btns[1]是dom对象  
​  
        //需求3：输出所有button的文本  
        // 先遍历所有button  
        $btns.each(function(){  
            console.log(this.innerHTML);  
       })  
​  
        //需求4：输出第三个按钮是所有按钮当中的第几个  
        console.log($('#btn3').index());  
​  
    </script>  
</body>

## **基本选择器**

<body>  
    <div id="div1" class="box">div1(class=box)</div>  
    <div id="div2" class="box">div2(class=box)</div>  
    <div class="div3">div3</div>  
    <span class="box">span(class=box)</span>  
    <script src="../js/jquery-1.10.1.js"></script>  
    <script>  
        //选择div1  
        // $('#div1').css('background','pink');  
​  
        // //选择所有的div  
        // $('div').css('background','pink');  
​  
        // // 选择所有class属性为box的元素  
        // $('.box').css('background','#bfa');  
​  
        // 选择所有的div和span元素  
        // $('div,span').css('background','pink');  
​  
        // 选择class属性为box的div  
        $('div.box').css('background','pink');  
    </script>  
</body>

## **层级选择器**

<body>  
    <ul>  
        <li>AAAAA</li>  
        <li class="box">BBBBB</li>  
        <li title="hello">CCCCC</li>  
        <li title="hello">DDDDD</li>  
        <span>EEEEE</span>  
    </ul>  
    <script src="../js/jquery-1.10.1.js"></script>  
    <script>  
        //选择ul下面所有的li  
        // $('ul li').css('background','pink');  
​  
        // 选中ul下所有的子元素span  
        $('ul>span').css('background','pink');  
​  
        // 选中class为box的下一个li  
        $('.box+li').css('background','pink');  
​  
        // 选中class为box后边所有的兄弟元素  
        $('.box~').css('background','pink');  
​  
​  
    </script>  
</body>

## **过滤选择器**

<body>  
    <ul>  
        <li class="box">list item 1</li>  
        <li class="box">list item 2</li>  
        <li>list item 3</li>  
        <li>list item 4</li>  
        <li>list item 5</li>  
        <li title="two">BBBBB</li>  
        <li title="threer">ccccc</li>  
        <li title="four">eeeee</li>  
        <li title="hello">DDDDDD</li>  
        <li style="display: none;">我本来是隐藏的</li>  
        <span class="box">class为box的span</span>  
    </ul>  
    <script src="../js/jquery-1.10.1.js"></script>  
    <script>  
        //选择第一个li，:first  
        // $('li').first().css('background','pink');  
        // $('li:first').css('background','pink');  
​  
        // //选择最后一个li，:last  
        // $('li:last').css('background','pink');  
​  
        // 选择所有class属性不为box的li :not()  
        // $('li:not(.box)').css('background','pink');  
​  
        // 选择第二个和第三个li元素 :eq()  
        // $('li').eq(1,2).css('background','pink');  
        // $('li:eq(1),li:eq(2)').css('background','pink');  
​  
        //获取>0 <3的索引，只有1,2  
        // $('li:gt(0)li:lt(2)').css('background','pink');  
​  
        //选择内容为BBBBB的li，:contains  
        // $('li:contains(BBBBB)').css('background','pink');  
​  
        // 选择隐藏的li  
        $('li:hidden').show(1000);  
​  
        // 选择有title属性的li元素  
        // $('li[title]').css('background','pink');  
​  
        // 选择title属性为hello的li元素  
        // $('li[title=hello]').css('background','pink');  
​  
        // 选择所有有title属性且title属性不为hello的li元素  
        //方法1  
        // $('li[title][title!=hello]').css('background','pink');  
        // 方法2  
        $('li[title]:not([title=hello])').css('background','pink');  
​  
          
    </script>  
</body>

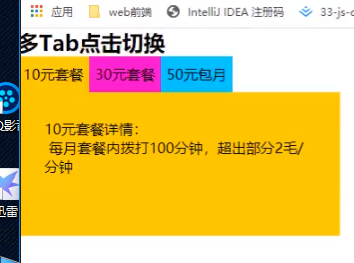
## **表单选择器**

:input//表单选择器，选择所有的表单元素  
:input:text:disabled//text类型并且禁用状态

作业

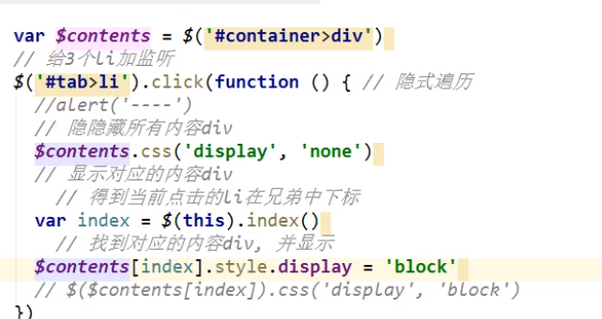
点击显示的需要隐藏的实际是上一次显示的那个

一个函数的结尾，相当于是下一次执行的开始，一个函数的开始，也相当于是上一次的结尾

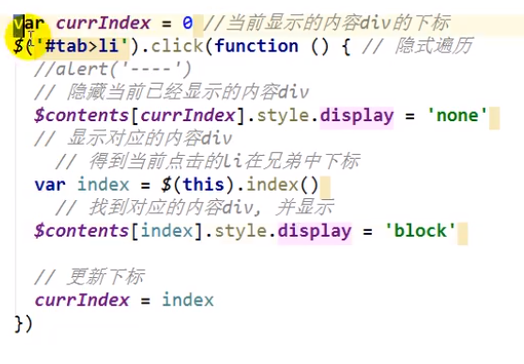




方法1



方法2 这种适用于多个li，比较灵活



# **0615 属性**

attr方法也是读写合一

attr用于操作标签自定义属性（属性值可以自己随意写的），操作非布尔值的属性

prop用于操作标签固有属性，操作布尔值

haslayout相当于Chrome的BFC，没事多了解一些普及的东西

过滤选择是在查找dom的过程中进行筛选，把匹配的dom元素封装进jQuery对象

筛选过滤的方法是在已经有jQuery对象的前提下，要对当前这个对象进行筛选，这时只能通过方法来解决，不能使用选择器了

filter()，根据传递的选择器，筛选已有的jQuery对象，筛选出与指定表达式匹配的元素集合。

has()这个方法只要内部有就行

prevAll()就是前面所有的，括号内可以写选择器，查到的元素会有倒序存入jQuery对象

nextAll() 后面所有的，括号内可以写选择器

perv() 前面一个

next()后面一个

siblings()除了我以外的前后所有兄弟元素

## **筛选-查找-子级-父级-兄弟**

children() 用来获取子元素   
​  
find()   用来获取后代元素  
​  
parent()和获取父元素

## **工具方法**

把json数组或对象转成正常的数组或者对象用parseJSON();

body在ie里有效

html在Chrome有效

如果是老版本的jQuery用第一个方法，新版本的用第二个



设置滚动条坐标时和读取一样，html在Chrome中生效，body在ie中生效

解决方案：html和body同时设置就行了，用并集选择器

IMG_274

如果版本没问题，document也是可以用的



# **0616 文档增删改**

empty 掏空节点，用于删除所有子元素，保留父元素

offset和position

offset是个对象，left和top两个属性，元素相当于页面左上角的位置

position 返回的也是一个对象，left和top两个属性，元素相当于父元素左上角的位置，和绝对定位的逻辑相同

以上两个方法返回值都不带单位，但是以像素来计

offset传一个对象，可以设置元素的位置，设置时不需要带单位，不过一般不使用，一般是通过css来修改

## **元素尺寸**

三种尺寸，四种情况

width和height的方法，返回的是你设置给当前元素的宽高的值

内部尺寸，包含width+padding，也就是content+padding

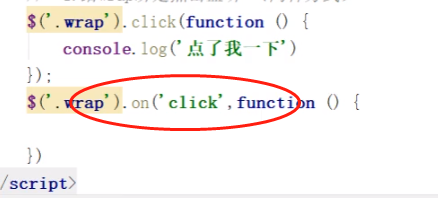
外部尺寸，包含width+padding+border

outerWidth和outerHeight方法传递参数true，则在原有基础上添加margin的值

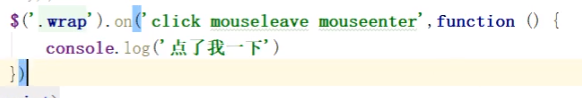
## **事件的绑定与解绑**

eventName：编码方便，但是只能加一个监听，且有的监听不支持

on这个方法，只要js原生有的，传进来直接使用



多个事件响应一个回调，用on直接可以解决



hover(移入的回调，移出的回调)，如果只传递了一个参数，那么移入和移出都是使用这一个函数，hover底层使用mouseenter和mouseleave实现的

## **事件的解绑**

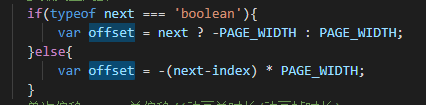
off()默认清除所有事件监听

off()方法允许传递参数，参数为事件名称的字符串

可用于单独清除某个事件，如果想删除多个，使用空格分开

## **轮播图**

一个函数就解决一个问题



if里边儿直接判断next是不行的，因为next可能是false，false的话直接走else了，那个位置判断它的类型啊，布尔值true和false都是布尔值，只有不是true，传索引的时候才走else

html()拿的是标签体文本，val()拿的是文本框用户输入的值

# **0617 事件委托 自定义动画**

delegate方法：由父元素调用，传递三个参数(子元素(选择器)，事件名称，回调函数)

jQuery帮你修改了this指向

undelegate 解绑事件

fadeOut() 淡出，就是慢慢消失

fadeIn() 淡入，慢慢显示

fadeToggle() 切换，相当于开关

slideUp() 滑出，高度的过渡，就像卷帘门

slideDown() 滑入

hide()隐藏 show()显示，如果不传递时间，是没有任何过渡的，toggle()切换

渐变不能过渡，本质是绘制了一个背景图片

宽度如果不写，默认值是auto

opacity默认值是1

这相当于写两个过渡，前者是简写，后者是完整写法

IMG_279

## **自定义动画**

animate()，自定义动画，两个参数，参数1是一个对象，设置要过渡的样式，参数2是单位时间

stop()，参数1：是否清空动画队列，参数2：当前动画是否执行完成，不传参默认两个参数都是false

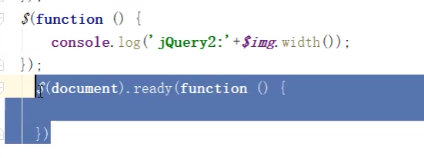
## **onload和ready的区别**

window.onload是需要等所有资源加载完成才会触发回调函数，html，css，js，图片等

jQuery的加载时间是document.ready 指的是文档（dom）加载完成后就会触发，不包含其他资料

window.onload只能指定一个回调函数，document.ready可以指定多次

前者是简写，后者是完整版



## **todolist**

遍历数组里的对象



要数据绑定

页面初始化

数组有四个对象，就得放四个li

定义函数，更新任务总数

ul当中有多少li就有多少任务，所以直接使用li长度即可

定义数越单一越好，复用性越好



# **键盘编码查询**

<body onkeyup = "unicode(event)">  
    <script>  
        function unicode(event){  
            alert(event.keyCode)  
       }  
    </script>  
</body>

# **0619 多库共存**

如果你哥哥库都有$，就存在冲突

jQuery可以释放$的使用权，让另一个库正常使用，此时jQuery就只能使用jQuery了

noConflict解除$命名冲突

# **电商详情页**

如果没有很强的空间感，先搭色块

\*zoom:1;是在ie里面清除浮动的

&:就代表元素本身，相当于this

混合的方式 .混合明(){}，混合调用的结尾必须写; 否则编译报错

字跟边框之间只能用padding

导航是否用ul取决于页面是动态还是静态，如果是动态就用复杂的结构ul，比如移动端页面的拖拽，如果是动态的就直接使用a标签

给共有的祖先元素设置行高就可以都实现垂直居中

font-size：0，换行符就没了，再重置超链接font-size

& + a，相当于把第一个排除在外

如何锁定问题，样式出问题第一件事先到控制台看元素身上看有没有写的样式

写了浮动想着给父级清除浮动

## **作业**

鼠标移入的蒙板动起来

getBoundingClientRect()

这个方法返回一个矩形对象，包含四个属性：left、top、right和bottom。分别表示元素各边与页面上边和左边的距离。

## **[getBoundingClientRect()](https://www.cnblogs.com/leejersey/p/4127714.html)**

var box=document.getElementById('box');     // 获取元素  
​  
alert(box.getBoundingClientRect().top);     // 元素上边距离页面上边的距离  
​  
alert(box.getBoundingClientRect().right);    // 元素右边距离页面左边的距离  
​  
alert(box.getBoundingClientRect().bottom);    // 元素下边距离页面上边的距离  
​  
alert(box.getBoundingClientRect().left);     // 元素左边距离页面左边的距离

如果想在一个指定的元素的后边添加内容，用after(content);

$(A).insertAfter(B)表示将A插入到B以后

$(A).appendTo(B)表示把A追加到B中

## **电商页面逻辑**

html部分

最外层容器wrap

头部容器header

最上面导航容器topContainer

版心container

左边登录、注册容器loginContainer

p标签，"尚品汇欢迎您！"

p标签

请

a标签，登录

a标签，免费注册

右侧导航列表，rightList

8个超链接

logo和搜索框的容器middleSearch

h1标签里

a标签

logo

搜索框区域search

form创建表单

输入框

提交按钮

商品导航分类typeNav

商品导航容器menuContainer

h2标签，'全部商品分类'

导航列表

8个超链接

主要内容区域main

路径导航conPoin

3个超链接，一个文本

内容区mainContent

左侧放大镜容器previewWrap

放大镜区域zoom

小图

蒙板

大图

缩略图，specScroll

a标签，左按钮

轮播图容器items

ul

随便几个li，每个里面放图片地址

a标签右按钮

## **创建图片**

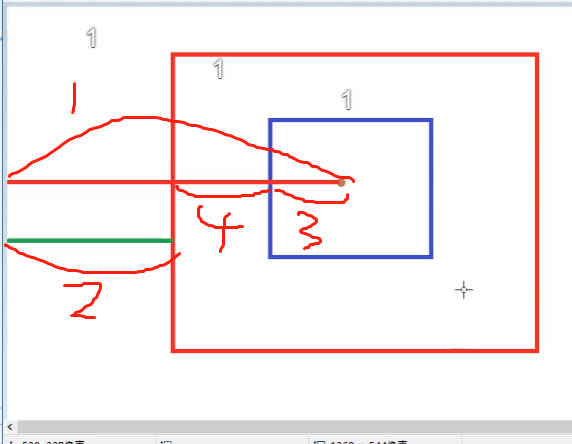
名字=new 构造函数();  
名字=document.createElement('img');  
这两者效果是一样的



## **放大镜逻辑**

鼠标的位置动不了，只能修正元素的位置

1-2-3（元素宽度一半）



计算比例：蒙板总共可以移动的距离/大图能够移动的距离

剩余距离=总距离-已经走过的距离

比较碎的用div，操作比较方便

三个空格相当于一个字

dl列表，描述列表,dt，dd是块

<dl>  
    <dt></dt>  
    <dd></dd>  
</dl>

## **放大镜逻辑**

创建放大镜索引 magnifierIndex = 0;

调用 magnifier()

提取放大镜效果的函数 function magnifier(){}

模拟后台数据，从js文件里面引入数据，保存变量imgSrc

(定义一个数组imgSrcArr，里面放大图和小图的图片路径，从js文件引入，这里就不用写了)

获取小图容器zoom，保存变量名zoomNode

获取左侧放大镜容器previewWrap

初始化蒙板，定义变量maskNode=null

初始化大图容器，变量名bigImgContainer=null

初始化大图，定义变量bigImg=null

绑定鼠标移入移出事件

判断蒙板状态，进入判断就表示蒙板为空

在js里面往页面创建蒙板，变量名maskNode，不定义，赋值操作

给蒙板写个类名，提示maskNode.className = 'mask';

把蒙板插入到小图容器当中

(如果蒙板存在，大图和大图容器也存在，所以直接同时创建即可)

创建大图容器，变量名bigImgContainer，不定义，赋值操作

给大图容器写个类名，bigImgContainer

创建大图，不定义，给大图赋值

把放大镜生成的大图路径改为动态，大图路径就是从js文件引入的数据

把大图追加到大图容器当中

把大图容器添加到放大镜容器中

(if判断外面)

在小图容器里面的移动事件，需要用到事件对象ev

这里是移动的逻辑

创建一个对象表示蒙板相对父元素的位置，变量名maskPosition

left:鼠标横向坐标 - 蒙板实际宽度的一半 - 小图容器左边框距离屏幕的距离，getBoundingClientRect().left

top:鼠标纵向坐标 - 蒙板实际宽度的一半 - 小图容器上边框距离屏幕的距离， getBoundingClientRect().top

判断边界 右侧边界的判断小图容器宽度 - 蒙版宽度

判断如果maskPosition.left<=0

就=最小值

否则，如果maskPosition.left>=最大值（小图容器的宽度 - 蒙板的实际宽度）

就=最大值

如果maskPosition.top<=0,

就=最小值

否则，如果maskPosition.top>=最大值

就=最大值

对蒙版进行赋值

蒙板样式的left值 = 蒙板相对父元素的left +单位

蒙板样式的top值 = 蒙板相对父元素的top +单位

计算比例： 蒙版总共可以移动的距离 / 大图能够移动的距离

定义变量scale = (小图容器clientWith-蒙板offsetWith) / (大图的offsetWith - 大图容器的clientWith)

计算大图偏移

定义对象bigImgPosition{

left:蒙板相对小图容器的left / scale

top:蒙板相对小图容器的top / scale

}

给大图修改位置，表示大图的偏移方向与小图蒙板的偏移方向是相反的

大图样式的marginLeft = -bigImgPosition.left 加单位

大图样式的marginTop = -bigImgPosition.Top 加单位

小图移动事件外面

放大镜区域(previewWrap)的鼠标移出事件

小图容器的移除子元素的方法，删除蒙板

放大镜区域的移除子元素的方法，删除大图容器

把蒙版元素的变量初始化，为了下一次判断移入显示，移出删除

蒙板设为null

大图容器设为null

大图设为null

解绑小图容器移动事件，设为null

解绑放大镜区域(preview)的移出事件，设为null

放大镜效果的函数外面

获取元素非行内样式，创建函数getStyle

​  
function getStyle(obj, attr) {  
                if (obj.currentStyle) {  
                    return obj.currentStyle(attr);//兼容ie的写法  
               } else {  
                    return window.getComputedStyle(obj)[attr];//兼容非ie的写法  
             }  
           }

缩略图的滑动效果函数 thumbnail()

function thumbnail(){

获取左按钮

获取右按钮

获取ul容器(图片容器)list，变量名imgList

获取每一张图片的容器li，变量名liNodes，是为了获取距离

定义每次滑动的数量，变量名moveNum =2;

定义容器中默认显示的数量 showNum = 5;

(一次移动两张图片，两张图片的宽度+两倍的marginRight)

(移动规则，如果剩余距离够移动则正常移动，如果不够，直接走完剩余距离)

计算总共可以移动的长度 （li总数 - 显示li的个数）\* （li的宽度+ li的marginRight）

定义变量countLength = 。。。

提示

var countLength = (liNodes.length - viewNum) \* (liNodes[0].offsetWidth + parseInt(getStyle(liNodes[0], 'marginRight')));

每次移动的长度=(一个元素的宽度（带margin）) \* 每次移动的数量

var moveLength = (liNodes[0].offsetWidth + parseInt(getStyle(liNodes[0], 'marginRight'))) \* moveNum;

初始化已经移动的距离

变量名tempLength = 0;

右边按钮点击事件

判断如果已经走过的距离 < 总共可以走的距离 才能继续移动

(剩余距离 大于每次可以移动的长度 则每次都该走的距离)

(剩余的距离 = 总共可以走的距离 - 已经移动的距离)

判断如果总共可以走的距离 - 已经走过的距离 > 每次移动的距离

就让已经走过的距离 += 每次移动的距离

并且图片容器样式的left = -已经走过的距离+单位

否则

已经走过的距离 += (总共可以走的距离 - 已经走过的距离)

图片容器样式的left = -已经走过的距离+单位

右边按钮点击事件外面

左边按钮点击事件

相对于prev按钮来说 可以动的距离 就是tempLenght

判断如果已经走过的距离是否>0

(如果可以走的距离 大于每次移动的距离 则走该走的长度 否则直接为0)

判断，如果已经走过的距离大于每次移动的距离

就 已经走过的距离 -= 每次移动的距离

并且图片容器样式的left = -已经走过的距离+单位

否则

初始化已经走过的距离，让已经走过的距离=0

并且图片容器样式的left = 已经走过的距离+单位

缩略图滑动效果外面

缩略图点击效果

thumbnailClick()

function thumbnailClick(){

获取ul容器中图片的集合

变量名imgItems

获取小图容器里面的图片

变量名zoomImg

先遍历图片集合imgItems

给所有图片绑定单击事件

遍历可能被点击的数量

判断，如果所有图片里面可能被点击的图片j==当前点击的图片this

就让放大镜的索引 = 点击的图片索引j

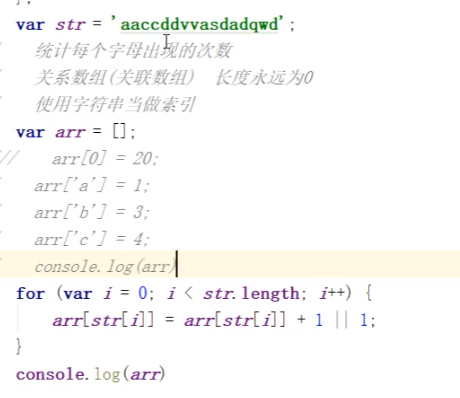
(在遍历可能被点击的数量循环外面)

修改小图容器里的图片的src=this.src，目的是让缩略图点击的图片跟小图容器的图片保持一致

# **关联数组**

字符串的字符可以当做索引来读取

案例



# **0622 电商页面**

模拟数据

对象有几个决定了有几个dl

title决定了dt的数量

data决定了dd的数量

<mark></mark>是个标记标签，默认有个黄色背景色，如果一行文字特殊颜色的话就用它

nth-of-type()？

创建一个数组，一一对应



数组的长度由dl来决定

goodData.path

模板字符串 ${}就可以在字符串中正常解析，就是省略了字符串拼接的过程，在模板字符串里面可以随意回车换行没有影响

## **01商品筛选动态加载逻辑**

(缩略图点击效果函数外面)

调用商品内容筛选的函数

function screening(){

模拟数据

从导入的data数据里面获取，演示

var crumbData = goodData.goodsDetail.crumbData;

获取选择区域容器chooseArea

forEach遍历crumbData，形参里面传element

每次遍历进来 先创建一个dl标签，变量名dlNode

每一个dl当中都只有一个dt 所以在这里同步创建即可，变量名dtNode

通过element的title属性，给每一个dt设置文本innerHTML

把dt插入到dl中

forEach遍历每一个对象element当中的data属性，动态创建dd，形参传element

创建dd，变量名ddNode

给dd插入内容element，(innerHTML插入)

给生成的dd添加自定义属性 保存对价格的修改

把dd插入到dl中

(遍历data属性外面)

把当前遍历生成的dl插入到选择区域的容器中

}

## **02商品内容筛选功能逻辑**

(遍历模拟数据外面)

以下是点击筛选内容的交互

获取dl集合，变量名dlList(动态获取)

定义一个数组，数组中保存筛选条件，需要的时候 按顺序保存和获取里边的内容

并且 保证数组中只有dlList.length(crumbData.length)的值 当更换数据时 直接替换原有位置即可，提示

var arr = new Array(crumbData.length);

填充数组，用传入的值填充整个数组，fill() 方法用于将一个固定值替换数组的元素。

arr.fill(0);

遍历所有的dl标签(dl集合)，然后给当前的dl标签中的dd绑定事件

给每一个dl添加索引

获取当前dl集合所有的dd，变量名ddList，用匿名函数

动态获取所有的dd

遍历dd集合

对所有的dd绑定单击事件

绑定单击事件

dd高亮切换

先恢复所有dd的默认颜色，遍历dd集合

每一个dd样式的颜色改为#666

this样式的颜色设置红色

用户点击了dd，根据当前的dd找到对于的dl获取索引，然后修改数组内容为this

更新了新的选择结果 重新计算价钱，调用计算价格的函数priceNum()(计算价格的函数完善再补上)

获取选择区域保存结果的容器，变量名choosed

每次创建元素之前先把结果清空，否则点击dd就会出现数组里所有的元素

forEach遍历数组，生成对应的选择结果，形参element，index

检测数组，if判断当当前元素不为0的时候，创建mark标签

创建mark标签

把数组的内容放到mark标签中，演示

markNode.innerHTML = element.innerHTML;

创建mark标签中的a标签

创建a标签时用一个自定义属性保存当前a所在数组的下标，提示

aNode.setAttribute('num',index);

设置a标签的文本设为X

把a标签放到mark中

把mark标签放到存放选择结果的容器中(choosed)

## **03商品内容筛选删除功能逻辑**

(遍历数组 生成对应的选择结果外面)

获取mark标签中所有的a标签（获取集合）

遍历所有的a标签

绑定单击事件

获取当前点击的a标签的索引，变量名num， 去操作对应的dl元素当中dd高亮

演示var num = parseInt(this.getAttribute('num'));

(在创建mark标签中的a标签下面，自定义a标签的num属性为index)

删掉当前点击的mark标签，找到mark的祖先元素再删除mark的父元素

获取dl集合里面的num索引的dd集合

遍历dd集合

dd集合里面的每一个dd样式的颜色设置为#666

dd里面的第一个索引的样式的颜色属性设置为默认的红色

每一次删除一个内容的时候要把当前数组当中对应的元素清空

arr[num] = 0;

点击的是dl，对应的dd的值

IMG_287

## **04、路径导航动态数据生成逻辑**

(商品筛选动态加载逻辑函数外面)

调用函数pathUrl()

提取路径导航动态生成的函数，函数名pathUrl

获取路径导航容器conPoin，保存变量名conPoinNode

获取数据path，保存给变量名path，因为在外部重写了一个js文件，是引入的，所以在文档里面找不到变量，语法参考var path = goodData.path;

遍历路径的数据

forEach遍历，形参把元素和索引都定义

遍历的时候创建a标签，这里的a标签代表的是子页面的url

判断数据有没有url属性

如果形参定义的索引的变量==数据的长度-1

最后一个a标签不是超链接，把a标签的文本设置为形参的元素变量的title的值

把a标签放到路径导航容器里面

否则

a标签的href属性设置为元素的url属性

a标签的文本设置为元素的title属性

把a标签放到路径导航容器里面

创建一个i标签，用来放/的

把i标签的文本设置为'/'

把i标签放到路径导航容器里面

## **05商品详情页动态数据生成逻辑**

(路径导航动态生成的函数外面)

调用函数goodsDetail

创建函数goodsDetail

获取商品信息容器goodsInfo，保存给变量名goodsInfo

获取右侧所有详细商品数据goodsDetail

定义一个变量保存要插入的右侧动态数据的html结构，变量名infoGoodsContent

参考

var infoGoodsContent = `<h3 class="infoName">${goodsDetail.title}</h3>

<p class="news">${goodsDetail.recommend}</p>

<div class="priceArea">

<div class="priceArea1">

<div class="title">价&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;格</div>

<div class="price">

<i>¥</i>

<em>${goodsDetail.price}</em>

<span>降价通知</span>

</div>

<div class="remark">

<i>累计评价</i>

<span>3283</span>

</div>

</div>

<div class="priceArea2">

<div class="title">促&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;销</div>

<div class="fixWidth">

<i>${goodsDetail.promoteSales.type}</i>

<em>${infoData.promoteSales.content}</em>

</div>

</div>

</div>

<div class="support">

<div class="supportArea">

<div class="title">支&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;持</div>

<p>${goodsDetail.support}</p>

</div>

<div class="supportArea">

<div class="title">配&nbsp;送&nbsp;至</div>

<p>${goodsDetail.address}</p>

</div>

</div>`;

商品信息容器goodsInfo的文本设置为 js往页面里面添加的html格式的字符串(infoGoodsContent)的

# **0623 电商详情页**

点击dd的时候和删除mark的时候会对价格有影响

这个this是每一个ddIMG_288

好处就是创建mark的文本取到dd里面的文本

IMG_289

侧边栏标签<aside></aside>

在迷糊的地方把this打印一下就清晰了

子元素有高度设置了浮动，父级就不用清除浮动

## **01 商品价格计算逻辑**

(遍历所有的dl标签(dl集合)，然后给当前的dl标签中的dd绑定事件外面)

计算价钱的函数，只要保存筛选条件的数组被操作了 则调用该函数 所以可以把数组当做参数传递过来

函数名priceSum

定义变量newPrice表示最新的价格，格式var newPrice = goodData.goodsDetail.price;

forEach遍历arr数组(里面保存的是dl集合)

先判断当前元素是否有值，在"screening函数"里面，在遍历data里面加一个自定义属性changePrice，是为了现在能获取到

如果有值，就把最新价格 += parseInt(element.getAttribute('changePrice'))(里边每一个dd的changePrice)

获取价钱容器保存变量名priceNode

价钱容器的文本设置为最新的价格newprice

(下面是内容详情选择搭配价格逻辑加的)

只要计算价钱 就改变选择搭配的价钱

获取原价，变量名choosePriceNode

choosePriceNode的文本设置为：人民币符号+最新价格

计算搭配完成的总价，获取搭配后的总价元素price，变量名chooseAllPrice

获取搭配区域所有的复选框'input'，变量名chooseAllCheckBox

forEach遍历搭配区域所有的复选框，形参传element(代表每一个复选框)

判断如果每一个复选框为true

就让最新价格 += 每个复选框对应的价格(value)

搭配后的总价的文本设置为 ￥+最新价格

## **02、内容详情侧边栏布局**

(主要内容区域外面)

商品详情容器 类名product

侧边栏容器aside标签

选项卡容器 tabWrap

h4 "相关分类"，添加类active

h4 "推荐品牌"

选项卡内容

类名tabContent

tabItem 添加类active

ul 类名partList

8个li

ul商品列表，goodsList

4个li，

## **3详情评论区tab切换布局 ；4详情选择搭配价格布局**

(侧边栏外边)

右侧区域 类名detail

选择搭配，类名fitting

评论区域 类名intro

ul tab容器 tabWrap

5个li，第一个加个类active

商品介绍区域 类名tabContent

## **5选择搭配价钱**

( 商品筛选动态加载逻辑外边)

调用函数fittingPrice()

创建函数

获取搭配后的总价元素'price' 变量名chooseAllPrice

获取搭配区域所有的复选框'input' 变量名chooseAllCheckBox

遍历所有复选框

给所有复选框绑定单击事件

获取原价元素，变量名choosePriceNode

获取原价，用的啥表情就获取对应的文字标签，变量名choosePrice

forEach遍历所有的复选框 判断哪一个被选中，形参里面传element

如果当前的多选框被选中 则返回的checked属性为true

就让原价加等当前点击的复选框的value(是复选框对应的价格)

(遍历复选框外边)

搭配后的总价的文本设置为 ￥+原价

(商品详情的数据动态生成外边)

## **选项卡封装，构造函数逻辑**

构造函数Tab，形参定义btns,content

实例对象添加tabBtn = btns

实例对象添加content = content

//保存当前的this，用于内部事件的回调函数访问外部的this，提示加\_ 用来区分不同的this

// 给所有的btn绑定单击事件

遍历所有的btn

//需要给所有的按钮添加索引

给所有的btn绑定单击事件

调用构造函数的this的clickBtn方法传事件源的this

选项卡封装外面

构造函数的原型上添加clickBtn方法

遍历tabBtn

把所有的btn按钮的类名设为空

把所有的内容样式设为none

btn的类名设为active

事件源的内容的btn的索引的样式设为block

//侧边栏调用tab切换

调用asideNav()

asideNav函数

//获取侧边栏的"相关分类"按钮'h4'，保存变量名tabNodes

//获取选项卡容器'tabItem'，变量名contentNodes

new 构造函数，传参tabNodes, contentNodes

//评论区域tab切换

调用intro()

intro函数

代码

function Tab(btns,content) {

this.tabBtn = btns;

this.content = content;

// 保存当前的this，用于内部事件的回调函数访问外部的this

var \_this = this;

// 给所有的btn绑定单击事件

for (var i = 0; i < this.tabBtn.length; i++) {

// 需要给所有的按钮添加索引

this.tabBtn[i].index = i;

this.tabBtn[i].onclick = function () {

// this 事件源

\_this.clickBtn(this)

}

}

}

Tab.prototype.clickBtn = function (btn) {

// btn接受的就是当前点击的事件源

// 选项卡切换

for (var i = 0; i < this.tabBtn.length; i++) {

this.tabBtn[i].className = '';

this.content[i].style.display = 'none';

}

btn.className = 'active';

this.content[btn.index].style.display = 'block';

}

// 侧边栏调用tab切换

asideNav();

function asideNav() {

//获取侧边栏的"相关分类"按钮

var tabNodes = document.querySelectorAll('.wrap .product > aside .tabWrap>h4');

// 获取选项卡容器

var contentNodes = document.querySelectorAll('.wrap .product > aside .tabContent .tabItem');

new Tab(tabNodes,contentNodes);

}

// 评论区域tab切换

intro();

function intro() {

var tabNodes = document.querySelectorAll('.wrap .product .detail .intro .tabWrap li');

var contentNodes = document.querySelectorAll('.wrap .product .detail .intro .tabContent .tabItem')

new Tab(tabNodes,contentNodes);

}

如果要单独操作一个层级，给相对定位影响是最小的

## **06 数量价钱更新逻辑**

(选择搭配价钱外面)

调用函数goodsNum()

创建函数

获取数量选择容器里面的input

获取数量选择容器里面的plus(+) ，变量名plusNode

获取数量选择里面的minus(-)，变量名minusNode

加号的点击事件

(在最上面定义的位置定义一个变量changeNum，表示商品默认数量为1)

商品默认数量++

input的value设置为默认数量

调用改变数量就改变搭配总价的函数changeFittingPrice()

减号的点击事件

商品最小值为1，不能为负数，判断商品默认值是否大于1

商品默认数量--

input的value设置为商品默认数量

调用改变数量就改变搭配总价的函数changeFittingPrice()

## **改变数量就改变搭配总价的函数**

函数名changeFittingPrice

获取往右侧商品区域添加数据的em元素，表示单价，变量名priceNode

.wrap .mainContent .mainCon .infoWrap .priceArea .priceArea1 em

需要修改的价格就是 单价的文本\*数量changeNum，变量名finalPrice

改变选择搭配价钱，获取选择价钱的p元素，保存变量名choosePrice

choosePrice的文本设为 ￥ +最终价格

## **07 侧边栏布局及逻辑**

(评论区域tab切换函数外边)

标志位逻辑

//右侧边栏效果

asideShow();

function asideShow() {

var butNode = document.querySelector('.wrap .toolBar .but');

var toolBarNode = document.querySelector('.wrap .toolBar');

var flag = true;//默认是闭合

butNode.onclick = function () {

if(flag){

//原本是true代表闭合 这次点击就要执行展开的逻辑

toolBarNode.className = 'toolBar toolBarOut';

this.className = 'but open';

//flag = false

}else{

//原本是flase代表展开 这次点击就要执行闭合逻辑

toolBarNode.className = 'toolBar toolBarWrap';

this.className = 'but list';

// flag = true

}

flag = !flag;

}

//鼠标滑过小图逻辑

var pullList = document.querySelectorAll('.wrap .toolBar .pull');

var tabIconList = document.querySelectorAll('.wrap .toolBar .pull .tabIcon');

var tabTxtList = document.querySelectorAll('.wrap .toolBar .pull em');

for (var i = 0; i < pullList.length; i++) {

pullList[i].index = i;

pullList[i].onmouseenter = function () {

tabTxtList[this.index].style.left = '-59px';

tabTxtList[this.index].style.backgroundColor = '#e1251b';

tabIconList[this.index].style.backgroundColor = '#e1251b';

};

pullList[i].onmouseleave = function () {

tabTxtList[this.index].style.left = '35px';

tabTxtList[this.index].style.backgroundColor = '#7a6e6e';

tabIconList[this.index].style.backgroundColor = '#7a6e6e';

}

}

}

# **0624中级原型链 执行上下文**

构造器的地址指向的是构造函数的地址

构造器是Object构造出来的

1实例的隐式原型指向构造函数的显示原型对象 2原型对象的constructor属性指向构造函数 3原型对象是Object()构造函数的实例 4原型对象的隐式原型指向Object构造函数的显示原型对象 5Object构造函数的显示原型对象可以理解为原型链的终点，它的隐式原型为null

6所有的函数都是Function()构造函数的实例，包括Function()和Object() 7所有函数的隐式原型都指向Function()构造函数的显示原型对象 8Function()构造函数的隐式原型对象和显示原型对象是同一个 9Function()的显示原型对象也是Object()的实例 10总之沿原型链查找最终都会找到Object()构造函数的显示原型对象

所有的函数都是Function的实例

## **执行上下文**

心法的东西反复看感受会不一样

js本身没有预解析，这是人为赋予的，比如炒菜，做菜的过程有颠勺这个过程，这是人为赋予这个过程的名字

es6里面有块级作用域，es5只有函数作用域

作用域链是使用执行上下文当中变量对象所组成的链条结构

查找的时候其实真正是先去自身的变量对象中查找，如果没有，就去上级执行上下文的变量对象中去查找，直到找到全局执行上下文

全局执行环境

函数执行上下文栈

收集变量

确定this指向

创建自身执行上下文的作用域链

执行函数执行上下文，为变量真正赋值，顺着作用域链查找

call stack就是栈结构

global是全局

local是当前的作用域

scope表示作用域

closure就是闭包

undefined是有但是没有赋值

not defined就不存在

# **0626 闭包**

如何产生闭包

函数嵌套

内部函数引用外部函数的局部变量

使用(调用)外部函数

上班必然会用到模块化

闭包的作用

延长外部函数遍历对象的生命周期

让函数外部可以操作到函数内部的数据(变量/函数)，通过闭包间接的操作

注意：浏览器为了性能，后期将外部函数中不被内部函数使用的变量清除了

闭包的缺点和解决

内存泄漏，就是程序申请的内存无法释放

内存溢出，内存被撑爆了

解决方案，

闭包是作用域链存在引用关系，得到的结果

外部函数只要调用，就会生成新的闭包机制、

每一套都是独立的，没有任何关系

外部函数被调用，内部函数就会被定义

这里 this是window，访问不到name

不要被对象的大括号迷惑，访问对象的属性和方法都要通过对象来访问

都是 基于基本的那几条

外部函数就被调用一次，就只有一套闭包机制

b的写法就相当于： var a = fun(0); var a1 = a.fun(1); var a2 = a1.fun(2); a2.fun(3);

## **自定义模块**

在公司js不可能一个人写

两个人写的js文件可能变量冲突，这个几率很高

## **面向对象**

面向对象三大特征：封装，继承，多态

es5当中没有类的概念，es5我们可以把构造函数理解为类

es6语法当中添加了类的概念class，但是也是假的，class后期es6半只也是构造函数，它只是一个语法糖

原型继承，主要是让子类继承父类的方法，子类的原型等于父类的一个实例

在新的原型中，也就是父类的实例中添加构造器，手动添加

IMG_290

借助构造函数继承，专门用来继承父类中的属性

属性继承用前者，方法继承用后者



混合继承就是把两个合到一起

## **事件循环机制**

进程：程序的一次执行，它占有一片独有的内存空间

线程依赖于进程执行的

线程是一个基本调度单位

一个进程中至少有一个运行的线程（主线程）

多个进程的数据不能直接共享

js是单线程

为什么不做多线程，因为服务端都是程序员来控制的

线程正常是无序的，谁先跑完谁先来，容易出现冲突

多线程不能操作dom

单线程意味着所有任务都要排队，一个没完成，后面都得等着

js里面就出了个异步操作

web work 模拟多线程

案例

峰哥上课的时候像喝饮料，让同学去买饮料这个事情是异步操作，饮料买回来了，上课的任务一直在执行，等上课的任务完成了才能继续喝饮料

可以理解为同步的执行也是特殊的异步，只是没有异步的任务

## **webworker**

onmessage:用于接收数据

postMessage：用于传输数据

# **0627 less**

Mixin 混合

vertical-align

声明less用@变量名

查找自身变了时，不管自身大括号内有多少个同名变了，没有远近之分，只有看最后一个，当自身没有时，不会向子元素的括号内查询，还是向外查找

less默认的选择器是后代选择器

如果需要给当前元素设置伪类，伪元素，结构类等，只需要在当前大括号里面使用&，代表其自身

如果需要操作子元素，只需要在子元素前边使用 >

如果当前的父元素有id或者比较特殊的class，可以减少一定的层级的嵌套

混合允许传递参数，与生命变量的方式相同，在使用小括号内，使用@声明，都好分隔

不给定默认值，参数的个数要对等，否则报错

混合设置参数时，在参数后边直接使用：赋值，即为默认值

设置默认值后，如果传入对应的参数，按照传入的值渲染，没传的参数，使用默认值

传递参数时，顺序不重要，因为是带名字的，不过一般还是会按照定义的顺序去写

引入外部less文件，@import'mixin'，只写less文件名，不写后缀

## **Bootstrap**

里面所有的类名不用记，要用的时候再查

container容器

gutter 槽宽

栅格系统

~相当于字符串拼接

when相当于if

offset列偏移

响应式宽度1170 970 750