中介者模式

中介者模式：通过中介者对象封装一系列对象之间的交互，是对象之间不在相互引用，降低他们之间的耦合。有时中介者对象也可改变对象之间的交互。

导航设置分析；

创建中介者对象

<script>  
 //中介者对象  
 var *Mediator*=function(){  
 // 消息对象  
 var \_msg={ };  
 return{  
 /\*\*\*  
 \* 订阅消息方法  
 \* 参数Type 消息名称  
 \* 参数 action 消息回调函数】  
 \* \*/  
 register:function(type,action){  
 //如果该消息存在  
 if(\_msg[type])   
 {  
 //存入回调函数  
 \_msg[type].push(action);  
 }else  
 {  
 //不存在，则建立消息容器  
 \_msg[type]=[];  
 //存入消息回调函数  
 \_msg[type].push(action);  
 }  
 },  
 /\*\*  
 \* 发布消息方式  
 \* 参数type消息名称  
 \* \*\*/  
 send:function(type){  
 //如果消息已经被订阅  
 if(\_msg[type]){  
 //遍历已存在的消息回调函数  
 for(var i= 0, len=\_msg[type].length;i<len;i++){  
 \_msg[type][i]&&\_msg[type][i]();  
 }  
 }  
  
 }  
 }  
 }();

测试中介者对象

//单元测试  
*Mediator*.register('demo',function(){  
 console.log('first');  
});  
  
*Mediator*.register('demo',function(){  
 console.log('second');  
});  
  
*Mediator*.send('demo');

导航示例：

订阅消息

/\*\*\*  
 \* 显隐导航小组件  
 \* 参数 mod 模块  
 \* 参数 tag 处理的标签（消息提示b，网址 span）  
 \* 参数 showorhide 显示还是隐藏  
 \* \*\*\*/  
 var showHideNavWidget=function(mod,tag,showOrHide){  
 //获取导航模块  
 var mod=document.getElementById(mod);  
 //获取下面的标签名为tag的元素  
 var tag=mod.getElementsByTagName(tag);  
 //如果设置为false或者hide则值为hidden，否则为visible；  
 showOrHide=(!showOrHide||showOrHide=='hide')?'hidden':'visible';  
 //站位隐藏这些标签  
 for(var i=tag.length-1;i>=0;i--){  
 tag[i].style.visibility=showOrHide;  
 }  
};

//用户收藏导航模块里面的导航有消息提示与网址功能

//用户 收藏导航模块  
(function(){  
 //……其他交互逻辑  
 //订阅隐藏用户收藏导航消息提醒消息  
 *Mediator*.register('hideAllNavNum',function(){  
 showHideNavWidget('collection\_nav','b',false);  
 });  
 //订阅显示用户收藏导航消息提醒消息  
 *Mediator*.register('showAllNavNum',function(){  
 showHideNavWidget('collection\_nav','b',true);  
 });  
 //订阅隐藏用户收藏导航网址信息  
 *Mediator*.register('hideAllNavUrl',function(){  
 showHideNavWidget('collection\_nav','span',false);  
 });  
 //订阅显示用户收藏导航网址信息  
 *Mediator*.register('showAllNavUrl',function(){  
 showHideNavWidget('collection\_nav','span',true);  
 });  
  
})();  
  
//推荐用户导航  
(function(){  
 //订阅隐藏推荐用户导航消息提醒消息  
 *Mediator*.register('hideAllNavNum',function(){  
 showHideNavWidget('recommend\_nav','b',false);  
 });  
 //订阅显示推荐用户导航消息提醒消息  
 *Mediator*.register('showAllNavNum',function(){  
 showHideNavWidget('recommend\_nav','b',true);  
 });  
  
})();  
  
//最常用导航  
(function(){  
 //订阅隐藏最近常用导航网址消息  
 *Mediator*.register('hideAllNavUrl',function(){  
 showHideNavWidget('recently\_nav','span','hide');  
 });  
 //订阅显示最近常用导航网址消息  
 *Mediator*.register('showAllNavUrl',function(){  
 showHideNavWidget('recently\_nav\_nav','span','show');  
 });  
})();

发布消息

这些模块订阅中介者提供的消息，剩下的只需要监听每位用户对设置层内的导航展现样式的设置，并发送样式消息。

//设置层模块  
(function(){  
 //消息提醒选框  
 var hideNum=document.getElementById('hide\_num');  
 //网址选框  
 var hideUrl= document.getElementById('hide\_url');  
 //消息提醒选框事件  
 hideNum.onchange=function(){  
 //如果勾选  
 if(hideNum.checked){  
 *Mediator*.send('hideAllNavNum');  
 }else  
 {  
 *Mediator*.send('showAllNavNum');  
 }  
 };  
  
 //网址选框事件  
 hideUrl.onchange=function(){  
 //如果勾选  
 if(hideUrl.checked){  
 *Mediator*.send('hideAllNavUrl');  
 }else  
 {  
 *Mediator*.send('showAllNavUrl');  
 }  
 };  
})();

备忘录模式

备忘录模式：在不破坏对象的封装性的前提下，在对象之外捕获并保存该对象内部的状态以便日后对象使用或者对象恢复到以前的某个状态。

备忘录模式用来缓存请求过的数据，每次发生请求的时候对当前状态做一次记录，将请求下的数据以及对应的页码缓存下来。如果某一时刻想返回到某一浏览过的新网页，直接在缓存中查询即可。直接恢复记录过的状态而不必触发新的请求行为。

新闻缓存器

//创建page类

<script>  
 //page备忘录类  
 var *Page*=function() {  
 //信息缓存对象  
 var cache = {};  
 /\*\*\*  
 \* 主函数  
 \* 参数page页码  
 \* 参数fn 成功回调函数  
 \* \*\*/  
 return function(page,fn){  
 //判断该页数据是否在缓存中  
 if(cache[page]){  
 //恢复到该页状态，显示该页内容  
 showPage(page,cache[page]);  
 //执行成功回调函数  
 fn&&fn();  
 }else  
 {  
 //若缓存cache中无该页数据  
 $.post('./data/getNewsData.php',{  
 //请求携带数据page页码  
 page:page  
 },function(res){  
 //成功返回  
 if(res.errNo==0){  
 //显示该页数据  
 showPage(page,res.data);  
 //将该页数据置入缓存中  
 cache[page]=res.data;  
 //执行成功回调函数  
 }else  
 {  
 //异常处理  
 }  
 })  
 }  
 }  
 }();  
  
 //下一页按钮点击事件  
 $('#next\_page').click(function(){  
 //获取新闻内容元素  
 var $news=$('#news\_content');  
 var page=$news.data('page');  
 //获取并显示新闻  
 page(page,function(){  
 //修正新闻内容元素当前页数据  
 $news.data('page',page+1);  
 })  
 });  
</script>

迭代器模式

迭代器：在不暴露对象内部结构的同时，可以顺序地访问聚合对象内部的元素。

迭代器

<script>  
 //迭代器  
 var Iterator=function(items,container){  
 //获取父容器，若container参数存在，并且可以获取该元素则获取，  
 // 否则获取document  
 var container=container&&document.getElementById(container)||document;  
 //获取元素  
 var items=container.getElementsByTagName(items);  
 //获取元素长度  
 var length=items.length;  
 //当前索引为0  
 var index=0;  
 //缓存源生数组splice方法  
 var splice=[].splice;  
 return{  
 //获取第一个元素  
 first:function(){},  
 //获取第二个元素  
 second:function(){},  
 //获取前一个  
 pre:function(){},  
 //获取后一个  
 next:function(){},  
 //获取某一个元素  
 get:function(){},  
 //对每一个元素执行某一个方法  
 dealEach:function(){},  
 //对某一元素执行某一个方法  
 dealItem:function(){},  
 //排他方式处理某一个元素  
 exclusive:function(){}  
  
 }  
 };  
</script>

迭代器实现

<script>  
 //迭代器  
 var Iterator=function(items,container){  
 //获取父容器，若container参数存在，并且可以获取该元素则获取，  
 // 否则获取document  
 var container=container&&document.getElementById(container)||document;  
 //获取元素  
 var items=container.getElementsByTagName(items);  
 //获取元素长度  
 var length=items.length;  
 //当前索引为0  
 var index=0;  
 //缓存源生数组splice方法  
 var splice=[].splice;  
 return{  
 //获取第一个元素  
 first:function(){  
 index=0;  
 return items[index];  
 },  
 //获取第二个元素  
 second:function(){  
 index=length-1;  
 return items[index];  
 },  
 //获取前一个  
 pre:function(){  
 if(--index>0){  
 return items[index];  
 }else  
 {  
 index=0;  
 return null;  
 }  
 },  
 //获取后一个  
 next:function(){  
 if(++index<length){  
 return items[index];  
 }else  
 {  
 index=length-1;  
 return null;  
 }  
 },  
 //获取某一个元素  
 get:function(num){  
 index=num>0?num%length:num%length+length;  
 return items[index];  
 },  
 //对每一个元素执行某一个方法  
 dealEach:function(fn){  
 var args=splice.call(arguments,1);  
 for(var i=0;i<length;i++){  
 fn.apply(items[i],args);  
 }  
 },  
 //对某一元素执行某一个方法  
 dealItem:function(num,fn){  
 fn.apply(this.get(num),splice.call(arguments,2));  
 },  
 //排他方式处理某一个元素  
 exclusive:function(num,allFn,numFn){  
 this.dealEach(allFn);  
 if(Object.prototype.toString().call(num)==="[object Array]"){  
 for(var i= 0,len=num.length;i<len;i++){  
 this.dealItem(num[i],numFn);  
 }  
 }else  
 {  
 this.dealItem(num,numFn);  
 }  
 }  
  
 }  
 };  
</script>

测试：

var demo=new Iterator('li','container');

数组迭代器

//数组迭代器

var eachArray=function(arr,fn){

var i=0;

len=arr.length;

//遍历数组

for(;i<len;i++){

if(fn.call(arr[i],i,arr[i])===false)

{

break;

}

}

}

对象迭代器

//对象迭代器

var eachObject=function(obj,fn){

for(var i in obj){

if(fn.call(obj[i],i,obj[i])===false)

{

break;

}

}

}