JQuery

A **元素选择 $()**

eg: $('.demo')　$('div')　$('#id')　$('ul>li')　$('#demo p')

$()作为元素选择而存在，括号内的写法与css选择一个标签的写法完全一样，它等同于document.getElementsByClassName()；但是它与原生方法不同的是，它能集中操作多个元素，例如当需要为多个div绑定事件时，通过原生方法选出一堆div后，需要通过循环遍历，给每一个div绑定事件，而在jquery中通过$()的方法 不需要循环遍历，便可一次性给所有的div都绑定事件.

$()所返回的结果为一个类数组，它继承了jquery上的方法，我们称该类数组为jquery对象，继承了jquery上的方法指的是jquery系统下规定好的关于一个jquery对象的方法，但这些方法不一定是jquery独有，该方法可能在原生中也有，但jquery把他们统筹到jquery对象的原型链上。

$() == jquery()　就其本质而言$是一个方法(函数)，$()等同于该函数的执行，在jquery中有一个单独的模块sizzli,它实现了$方法sizzle主要应用正则表达式，实现元素的匹配。

B **$(适用范围)**

$()不仅适用于通过css选择器获取dom元素，它还可以通过原生dom、jquery对象、null/undefined、函数function(){}、selector/content等，获取元素

C **原生dom**

var oDiv = document.getElementsByTagName('div')[0]　这里所说的原生dom，指通过原生js方法获取的dom元素
$(oDiv)

D **jquery对象**

```
var oDiv = $('.demo')
$(oDiv).click( function(){ console.log(1) } )
```

对于需要被反复操作或内部内容多变的dom元素，用一个变量去存储它，不需要每次都获取，提高了代码执行的效率

E **布尔值为false的原始值** (undefined,null,'',false)

$(''), $(undefined)返回一个空的jquery对象，需要注意的是这里的‘空’仅指它没有自身的属性，但它依然继承了jquery上的属性

F **函数function(){}**

```
$(function(){console.log( '等同于立即执行函数' ) } )
```

G **指定某一执行期上下文内的某一元素**

$('li','ul')　ul执行期上下文内的li元素
$('li','ul','div')　div执行期上下文内的ul执行器上下文内的li元素

H **jquery特有的选择规则**

```
$('ul>li:first');　　//ul内的第一个li
$('ul>li:eq(2)');　//ul内的第三个li.eq代表要选择的li的索引值
$('ul>li:odd/even');　//odd为基，even为偶，它们查找的是ul中索引值为基数的或偶数的li
```

I **$().filter()**

$('li').filter('.demo');　//把在li中class名为demo的li选择出来
就其字面而言，filter是过滤的意思，所以我们可以理解为,把class名为demo的li留下，没有demo的li全部过滤掉

```
$('li').filter(function(){      $('li').filter(function(index,ele){
    return true                     return index%3 == 0
})                              })
```

filter()会循环遍历通过$('li')选择出来的类数组，每循环一遍都会执行一次function(){}，看返回值是否为true,是true的话，留下，不是的话，也就是为false的情况的话当前被遍历的元素将会被过滤掉，它数组上的filter方法功能一样。

原生filter与jquery的filter的区别

1）原生JS中的filter的返回值为一个数组，而$().filter()返回的是类数组
2）在jquery中的filter执行时的回调函数的第一个参数为索引值，第二个参数为遍历的当前值，而在原生JS中，filter执行时的回调函数的第一个值为遍历的当前值，第二个值才为索引值
3）在jquery中filter的回调函数内部的this指向为当前遍历至的元素，也就是说当前遍历至哪一个元素，那么此时的this便是该元素，而在原生js中假如filter(function(){}, ...)不传第二个参数的话那么回调函数中的this指向window, 传值了的话那么this指向传入的对象

J **$().not()**

not与filter仅有一点差别即它所return的值与filter正好相反，也就是说filter是把符合要求的留下，不符合要求的去掉，而not是把符合要求的去掉，不符合要求的留下，如$('li').not('.demo');　class名不为demo的li元素将会被选出

K **$().has()　$().find()**

$('li').has('p')　把子元素中有p的li选出，返回子元素中有p的li元素
$('li').find('p')　查找li中是否有p，有p的话把它选出，返回p元素

L **$().eq()**

与$('ul>li:eq(2)')相同

M **$().is()**

是与不是，例 $('li').eq(2).is('.demo')　li中的第三个li是否带有demo的class名，是的话返回true，否的话返回false，
一般用于事件委托，判断当前点击的是不是demo是的话触发事件

N **$().css()写法**

链式写法：$('li').css('width','100px').css('height','100px').css('background','orange')
对象式写法：　$('li').css({width:'100px',height:'100px',background:'orange'})
在为元素设置css属性时，除仅仅只需要设置一个属性的情况除外，在需要设置多个属性时，对象式写法将更加高效
css方法不传第二个参数是，$('li').css('width') 他能取值，但不可以任何参数的不传，否则会报错

O **$().text()　$().html()**

与原生JS方法中的ele.innerText和ele.innerHTML相同，但略有不同的是在JS中这两个方法为属性，需要以赋值的形式使用，
而在jquery中这两个方法为函数，需要以函数执行的形式进行使用
同样地$().text()不识别标签，括号内所输入的所有信息将作文本内容处理
$().html()既识别标签同时也识别文本内容$('li').html('<div>123</div>')
同时在$('li').html()中，括号内有值的话那么html(有值)将起赋值的作用，而当html()无值时，它将起取值作用把li元素内的内容取出(注意这里的内容包含子标签)，$().text()与$().html一样，当有值时为赋值，无值时为取值，取值时将统一只取文本内容，两者所取出的值统一为字符串类型

P **$().attr()　$().prop()**

为dom元素设置行间属性，与原生JS中的ele.setAttribute('class', 'demo')和ele.getAttribute('class')原理相同，attr()可设置自定义属性如$('li').attr('data-log', 'demo')，而$('li').prop('class', 'demo')只可以设置HTML中原有的属性，
$().attr('class')，$().prop('class')假如只传第一个参数的话，他们能获取该行间属性的属性值，但区别在于，与传参一样prop()不能获取自定义的行间属性，而attr()可以获取自定义的行间属性

Q **对于行间属性中的一些状态值 checked、selected、disabled**

<input type="checkbox" checked> checked代表了当前选项默认被选中，同时可以表示为checked="checked"

```
  $('input').attr('checked')   //返回checked 假如没有checked属性的话返回undefined
☑ $('input').prop('checked')   //返回 true   无论有无checked属性对prop()的结果不产生任何影响，
☐ $('input').prop('checked')   //返回 false  它只基于当前选项有没有被选中而作判断
```

通过以上例子可以得知prop()方法对行间属性中的一些状态值是敏感的而attr()对行间属性中的一些状态值时不敏感的

[3 ▼] 下拉菜单

```
<select>                                   $('option:eq(2)').attr('selected')   //返回selected
    <option value="">1</option>            $('option:eq(1)').attr('selected')   //返回undefined
    <option value="">2</option>            $('option:eq(2)').prop('selected')   //返回true
    <option value="" selected>3</option>   $('option:eq(1)').prop('selected')   //返回false
</select>
```

input框是否可用

```
<input type="text" disabled>    $('input').prop('disabled')  //返回true
<input type="text" disabled>    $('input').attr('disabled')  //返回disabled
<input type="text">             $('input').prop('disabled')  //返回false
<input type="text">             $('input').attr('disabled')  //返回undefined
```

R **$().next()　$().prev()　$().index()**

next();　返回当前元素节点的下一个兄弟节点
prev();　返回当前元素节点的上一个兄弟节点
index();　返回当前元素节点在同类节点中的索引值

应用：通过获得当前被点击的li的索引值，在轮播图中可以操作当前要展示的图片

```
$('li').click(function(){
    $('li').css('background','#888');
    $(this).css('background','red');
    console.log($(this).index())
})
```

S **addClass()　removeClass()　toggleClass()　参数可以是function**

$('li').addClass('active')　添加class类名, 返回该元素的jquery对象
$('li').removeClass(.active')　删除class类名，假如不填具体值，将把class类名全删除掉，返回关于该元素的jquery对象

```
$('li').addClass(function(){
    if(index % 3 == 0){
        return 'aaa'
    }
})
```

当需要满足一定条件时才进行addClass()或removeClass()的话，我们可以通过传一个回调函数进行实现，其次在函数体内必须有具体的return值，因为所return的值代表了表添加上的Class属性名

$('li')).toggleClass('.active')　toggleClass()方法会先判断当前元素是否有active的class类名，有的话删除，没的话添上

应用：按钮状态标记
可以把它想成是电器按钮，假如按钮被按下那么它发光呈现active的样式，此时电器是通电状态，假如再按的话不发光，
把为active的class类名删除掉，按钮等同于回复之前状态，此时电器是不同点的状态

```
$('li').click(function(){
    $(this).toggleClass('active')
})
```

T **insertBefore()　insertAfter()　before()　强调的是前后**

$('li:eq(5)').insertBefore('li:eq(0)')　把第六位的li剪切下来插入至第一个li的前面
$('li:eq(0)').before('li:eq(5)')　第一个li的前面是第六个li
区别两者，要从语法去进行分析，在insertBefore()中强调的是把A插入至B，用于插入的A是重点，自然它返回的jquery对象是A
而before()强调的是B前面有A，B是被描述的主体，所以返回的Jquery对象为B

U **appendTo()　append()　插入至里面的最后面**

$('p').appendTo($('li:eq(2)'))　把p标添加至第三个li的里面，返回的是为p的jquery对象
$('li:eq(2)').append($('p'))　在第三个li里面添加p标签，返回的是第三个为li的jquery对象

V **prependTo()　prepend()　插入至里面的最前面**

$('p').prependTo($('li:eq(2)'))　把p标添加至第三个li的里面，返回的是为p的jquery对象
$('p').prepend($('li:eq(2)'))　在第三个li里面添加p标签，返回的是第三个为li的jquery对象

W **remove()　detach()　把元素删除并返回**

```
var item1,item2
$('.item1').click(function(){
    item1 = $('.item1').remove()
})
$('.item2').click(function(){
    item2 = $('.item2').detach()
})
$('p').click(function(){
    $('body').append(item1).append(item2)
})
```

区别：remove()与detach()的区别在于remove()不会把事件返回，
而detach()会把元素返回之余把关于该元素的事件返回，同时append(),
appendTo(),prepend(),prependTo()假如需要被插入的元素在存在于该区域，
那么它将覆盖前一个，也就是说$('body').append(item1).item1已经存在于
body内的话，插入依然会被执行，只是item1会覆盖原有的item1，这样的情况
只存在于要插入的item1与在body内已存在的item1为同一个item1时，
假如两者仅仅是长得一样但却是不同的item1的话，这样的情况将不生效，
html结构内将存在两个一模一样的item1

X **on()**(新版本) **注册事件**

事件注册方式：　$('div').on('click',function(){console.log(1)})

可传参数

```
$('div').on('click', 事件源 , 传至回调函数的参数 ,function(){.....})
$('div').on('click',  'p'  ,  [1,2,3]   ,function(){.....})
$('div').on('click', 非字符串 ,function(){.....})
```

在on()中总共可以传四个参数第一个为事件类型，最后一个是事件处理函数，第二个参数假如为非字符串的话，那么它将作为回调函数的参数传入至回调函数内部，假如为字符串的话该字符串将作为事件源存在，作用为判断当前是否点击的是p元素，是的话触发事件，不是的话不触发事件，也就是等同于限定div上的事件只能由规定的事件源对象触发，这里的事件源不能传自身也就是div

```
$('div').on('click', 'p' , 'p' ,function(e){.....})   //e.data为'p'
$('div').on('click',  ''  , 'p' ,function(){.....})   //既可以不指定事件源对象，又可以传字符串入回调函数的方法
```

Y **off()　解除绑定**

解除事件绑定只能是解除绑定了事件的元素的事件，而不能是元素节点，假如off()内不传值，默认解绑该元素的所有事件

```
$('div').on('click','p',function(){.....})
$('div').off('click')
```

有区别解绑

```
$('div').on('click','p',function(){.....})       function a() {console.log(1)};    function b() {console.log(2)};
$('div').on('click','span',function(){.....})    $('div').on('click',a);       $('div').on('click',b);
$('div').off('click', 'span')                    $('div').off('click',a);
```

解除span元素作为div的事件源对象的身份　　　取消b作为div元素的click事件的事件处理函数身份

Z **one()　只触发一次便自动解绑**

```
$('div').one('click',function(){.....})
```

A1 **scroll()　绑定滚轮事件　scrollTop()当前滚动出去的高度**

```
$(window).scroll(function(){
    $('div').css( { top: $(window).scrollTop() } )
})
```

B1 **innerWidth()　outerWidth()**

```
$('div').innerWidth()   //div的content + padding
$('div').outerWidth()   //div的content + padding + border
$('div').outerWidth(true)   //div的content + padding + border + margin
```

innerHeight()　outerHeight()的用法与特性与innerWidth()　outerWidth()相同

C1 **创建dom**

$('<div>div</div>')　　$('<div>'),$('<div/>') --->单标签写法,单标签写法无法插入内容
需要注意的是<img/>为单标签，而</div>为尾标签，把它写入括号内将不生成dom元素

```
$('<div><span><a href="#">123</a></span></div>').appendTo($('body'))
```

通过转义字符，可以实现所创建的dom结构与实际书写时的格式相同

```
$('<div>\
        <span>\
            <a href="#">123</a>\
        </span>\
    </div>').appendTo($('body'))
```

D1 **事件类属性**

e.pageX，e.pageY　相对于文档的鼠标的位置
e.clientX　e.clientY　相对于浏览器窗口的鼠标的位置
e.screenX　e.screenY　相对于电脑屏幕的鼠标的位置
e.which　哪个键或按钮　e.button　哪个鼠标按键
e.preventDefault()　取消默认事件
e.stopPropagation()　取消冒泡

```
$('div').on('mousedown contextmenu',function(e){
    e.preventDefault(); })
$('div').on('click',function(e){
    e.stopPropagation(); })
$('div').click(,function(e){ return false; })同时去取消冒泡与默认
```

E1 **offset().left／top　position().left／top**

$('div').offset().left　相对于文档的定位
$('div').position().left　相对于最近的有定位的父级的定位
当div的父级元素都没有定位时 position().left与offset().left所返回的值相同

F1 **parent()　offsetParent()　parents()　closest()**

$('p').parent()　返回p标签的直接父元素
$('p').parent('div')　看p标签的直接父元素是不是div,是的话把div返回出来，不是的话返回一个没有父元素的jquery对象
$('p').offsetParent()　返回最近的有定位的父级，假如父级上都没有有定位的元素，返回html，不接受参数
$('p').parents('p')　返回所所有指定的父元素，假如不传参返回全部的父元素
$('p').closest('div')　返回所指定的元素中离当前元素最近的那个元素，可以指定自身，那么返回自身，必须传参

G1 **val()**　打印input框中的value值

```
$('input').blur(function(){ console.log($(this).val()) })
```

H1 **each()**　用于遍历jquery对象与数组

```
$('li').each(function(index){   console.log(index,this)  })
```

each()中回调函数的第一个参数为当前元素的索引值，第二个参数为当前元素，可以用this替代，
与forEach()中回调函数的参数略有不同，在forEach(function(ele,index){})第一个参数为数组内遍历至的当前值，
第二个参数为当前值的索引，在forEach()方法中this只代表调用this的主体，即数组，而在jquery中从代码可以看出，
是每一个li均调用一次each()，所以在each的回调函数中，this可以代表每一个li

I1 **end()**　回退操作

```
$('li').eq(0).css({color: 'red'}).end().eq(4).css({color: 'orange'})
```

end()之所以可以回退至之前所所选出来的一堆li的原因是，在jquery对象中存在一个prevObject属性，从字面上理解prev为previous的缩写，它的意思是之前的，也就是说假如我们的jquery后返回出来一个jquery对象时，prevObject属性都会记录下，它的前一个jquery对象是谁，并且记录下的对象与事实上的对象为同一个对象，具有同样的继承和方法。而end()就是找到该prevObject并把它返回出来，从而实现回退操作。$('li')的prevObject为document对象，通过$()的方式直接选出来的对象，因其是从doucment直接选出的，所以它的prevObject为document对象

J1 **元素节点的选择**

siblings() 选出当前元素节点的所有兄弟节点
prevAll() 选出当前元素上面的所有兄弟节点
nextAll() 选出当前元素下面的所有兄弟节点
$('li:eq(2)').prevUntil($('li:eq(0)'))　选出第三个li与第一个li之间的兄弟节点，括号内还可传原生dom，只能向上
$('li:eq(2)').prevUntil($('li:eq(5)'))　选出第三个li与第六个li之间的兄弟节点，括号内还可传原生dom，只能向下

K1 **clone()**

```
var a = $('li:eq(0)').clone(true);
a.css({color:'red'}).appendTo($('ul'))
```

这里的克隆为深度克隆，克隆出来的a与原来的li是两个li,假如clone(true)传一个值为true的参数，
那么原jquery对象上的点击事件，也会被一并克隆出来，不传参或传false时将不会克隆事件

L1 **wrap()　wrapInner()　wrapAll()　unwrap()**

$('li:eq(0)').wrap('<div></div>')　给当前li包裹一层div
$('li:eq(0)').wrap('.demo')　可在保留原有的class名为demo的元素的基础上，克隆一份用于包裹当前li
$('li:eq(0)').wrapInner('<p>')　在当前li内的包裹一个p标签，li内原有的所有元素包括文本内容将被p标签包裹

wrap()与wrapInner()均可接收一个回到函数作为自己的参数

```
$('li').wrapInner(function(index){
    if(index == 0){
        return '<div></div>'
    }else if(index == 1){
        return '<span></span>'
    }else{
        return 'p'
    }
})
```

wrapAll()

$('li)').wrapAll('<div></div>')　包裹选中所有的元素，
需要注意的是不是每一个选中的元素都相应地包裹一个元素，而是把所有选中的元素捆绑在一起在外层包裹一个元素，

wrapAll() 在特定条件下会破坏html结构

```
<ul>                                                 <ul>
    <li>1</li>                                           <div>
    <span>span</span>     $('li)').wrapAll('<div>')          <li>1</li>
    <li>2</li>            =================>                <li>2</li>
</ul>                                                        <li>外面</li>
    <li>外面</li>                                        </div>
                                                         <span>span</span>
                                                     </ul>
```

wrapAll()所可能造成的结构错乱是多变的，但可以找到其中一些规律，首先它会以第一个找的li作为标准，把往后的li都提至该li的后面，自然地假如某一li的后面原来是一个非li标签那么该标签将会被一直往后挤，同时假如在ul外还存在li的话，那么同样该位于ul外的li也会无视结构上的限制，直接排到上一li的后面，也就是说到最后需要被包裹的元素将会呈现队列的形式存在，至于这些呈队列的li在怎么一个嵌套关系上完全有wrapAll()所找到的第一个li所在的位置决定。

unwrap()　解除包装

$('li:eq(0)').unwrap()　解除在ul内的li的ul包裹，结构化标签(head,body,html)不能删除

M1 **add()**

```
$('li').eq(0).css({color: 'red',width:100}).end().eq(4).css({color: 'red,width:100})
```

有时候end()方法会显得稍微笨重，代码并不够简洁，像以上这种两个li只需要设置上同样的css样式时，可以使用add()使代码更简洁

```
$('li:eq(0)').add($('li:eq(2)')).add($('li:eq(4)')).css({color: 'red'})
```

add()方法可以实现以一个元素为起始点任意选中一个或多个元素进行集中操作，它比end()方法的灵活性更高

N1 **slice()**

```
$('li').slice(0,2).css({color: 'red'})
```

将需要的元素按传入的索引位的范围截取出来，方便集中操作，算头不算尾 0<= x <2

O1 **empty()**

$('ul').empty()　清空ul内的所有内容　$('ul').empty() == $('ul').html('')

P1 **serialize()　serializeArray()**

在需要进行提交表单数据的操作中 seriazili()可以起到串联表单数据的作用 而serializeArray() 可以串联数据成数组

```
<form action="">
    <input type="text" name="username">
    <input type="text" name="age">
    <input type="submit">
</form>
```

[yjx] [25] [提交]

```
$('input:eq(2)').click(function(e){
    e.preventDefault()
    $('from').serialize()
})  返回username=yjx&age=25
```

在拼接完成后可以通过ajax进行数据传输

```
$('input:eq(2)').click(function(e){
    e.preventDefault()
    $('from').serializeArray()
})
```

返回[{name: "username", value: "yjx"},{name: "age", value: "25"}]
返回的数组可以方便我们进行取值或相关操作

Q1 **animate()**

$('div').animate(target,持续时间，速度变化方式(速率)，回调函数)
$('div').animate({left:300,top:300,width:300},1000,'swing')
系统默认自带的速度变化方式只有两种linear与swing，可以通过下载jquery.easing.js插件，实现更多的速度变化模式

animate()的第四个参数为回调函数，该回调函数将在本次运动结束后被执行

```
$('div').animate({ left: 300 }, 500, 'swing', function () {
    $('div').animate({ top: 300 }, 500, 'swing', )
})
```

stop()　一般与某一事件一起出现

$('div').stop()阻止当前运动进行后续运动，
stop(true)阻止所有运动，
stop(true，true)阻止所有运动，并停在当前目标点
stop(false，true)阻止当前运动，并停在当前目标点后，继续后续运动
第一个参数规定是否停止后续所有运动，false: 不停止，继续运动　true: 停止后续所有运动
第二个参数规定是否停在当前目标点上，　false: 不到达　true: 到达
finish()　停止所有运动 并且到达最终目标点(终点)

R1 **delay()　延迟多长时间**

```
$('div').animate({ left: 300 }, 500, 'swing', function () {
    $('div').delay(2000)  //延迟两秒钟
    $('div').animate({ top: 300 }, 500, 'swing', )
})
```

S1 **slideUp(**time，回调函数**)　slideDown()　slideToggle()**

slideUp() 如何元素已显示出来，向上滑动着隐藏该元素
slideDown() 通过向下滑动的方式把隐藏的元素显示出来

slideToggle()　假如元素已显示，滑动着让它隐藏，假如元素已隐藏，滑动着让它出现

```
$('.dengsao').click(function(){
    $('.dengge').slideToggle(500,function(){
    })
})
```

fadeIn()　fadeOut()　fadeToggle()　淡入淡出，原理与滑动隐藏一样

这些方法统一可以接受两个参数，第一个参数为完成这项动作所需的时间，第二个参数为回调函数，在运动结束后执行

T1 **trigger()**

trigger()的作用是主动触发事件，事件类型包括系统默认事件与自定义事件，也就是说trigger()可以代替用户的行为去触发事件
同时trigger()可以往事件处理函数里传参，trigger()解决了事件处理函数无法往里传参的形式，假如我们要应用到某些变量时，
那么我们只能通过闭包或每次触发事件时都生成一个相同的变量的形式，这样的话不是造成内存泄露就是降低了执行效率，
而trigger()很好地解决了这个问题，trigger() 等同于事件处理函数的变量名为trigger,然后自执行，假如事件本身存在冒泡，
那么由它所触发的事件也存在冒泡

```
$('div').on('eat',function(event,a){
    $('div').css({width: $(this).width() + a})
})
var timer = setInterval(function(){
    $('div').trigger('eat',100)
},500)
```