jQuery 的初认识

定义：是一个优秀的 JS 函数库（Write Less, Do More!!!）

功能：封装简化 DOM 操作，增删改查

强大的选择器:HTML、元素选择器，操作

CSS 的操作

HTML 事件的处理

JS 动画效果

隐式遍历：一次操作多个元素

链式调用

读写合一：读写数据都一个函数

浏览器的兼容

易扩展插件，ajax 封装

引入：直接像引入 js 一样就行

使用方式：

核心函数$(xxx)/jQury

核心对象:$.xxx(执行$()得到)

区别 js 库文件

测试版：在开发时候用

压缩版：在生产的时候用：没有任何空格和跨行

区别 2 种引入 js 库的方式：

1.服务器本地库（开发测试时使用）

加重了服务器负担, 上线时一般不使用这种方式

2.CDN 远程库

项目上线时, 一般使用比较靠谱的 CDN 资源库

减轻服务器负担

jQery 的不同版本的区别（2-3 版本不兼容 IE8）

1.x（常用）

兼容老版本 IE

文件更大

2.x

部分 IE8 及以下不支持

文件小, 执行效率更高

3.x

完全不再支持 IE8 及以下版本

提供了一些新的 API

提供不包含 ajax/动画 API 的版本

当一般函数来用：$(param)

param 是 function:相当于 window.onload =

param 是选择器字符：查找所有匹配的 DOM 对象，返回包含 DOM 元素的 jQuery 对象

param 是 DOM 元素：将 DOM 元素对象包装成 jQuery 对象返回$(this)

param 是标签字符串：创建标签 DOM 元素对象并包装 jQuery 对象返回

当对象来用

jQuery 对象：包含所有匹配的 n 个 DOM 元素的伪数组

执行$()返回的就是 jQuery 对象

基本行为：length，index(),each()

选择器：基础，层级，过滤，表单

jquery 的核心函数：

window.jquery = window.$ =jquery

（在 jQuery 中把 jQuery 的函数值都给了）

一般用即: $ 或 jQuery），所以一般只写$就好了

当他调用方法时 $就是一个工具对象

定义了这个全局的函数供我们调用

它既是一般函数调用，且传递的函数类型不同，格式不同就完全不同

也可作为对象调用其定义好的方法，

得到的都是伪数组，伪数组是只有数组的长度和下标取元素的功能

1.作为函数调用

1. 参数为函数

   $(fun)

2. 参数为选择器(selector)字符串

$("#div1") $('#btn').click(function () {

alert($(this).html());//这里是 jq 的对象，a 要$()吧 this 给括起来})

3. 参数为 DOM 对象

   $(div1Ele)

4. 参数为 html 标签字符串

   $(" ")

作为对象调用：

其它工具方法

$.each()

$.trim()

$.parseJSON()

.....

2.jQuery 作为核心对象

理解

即执行 jQuery 核心函数返回的对象

jQuery 对象内部包含的是 dom 元素对象的伪数组

jQuery 对象拥有很多有用的属性和方法,能方便的操作 dom

.属性/方法

基本行为

size()/length

```
console.log($btns.length);
```

[index]/get(index)

```
console.log($btns[2].innerHTML);
```

each()

```
$.each($btns,function () {
    console.log(this.innerHTML); })
```

$index()

```
console.log($('#btn3').index())//返回这个的下标位置
```

属性：操作内部标签的属性或值

CSS：操作标签的样式

文档：对标签进行增删改操作

筛选：根据指定的规则过滤内部的标签

事件：处理事件监听相关

效果：实现一些动画效果

使用 jQuery 核心函数

选择器

选择器本身只是一个有特定语法规则的字符串，没有实质用处它的基本语法规则使用的就是 CSS 的选择器语法，对基进行了扩展

只有调用$()，并将选择器作为参数传入才能起作用

$(selector)作用 ：

根据选择器规则在整个文档中查找所有匹配的标签的数组，并封装成 jQuery 对象返回

但是里面的值单独拿出来就是 dom，要用 dom 的方法。

分类

基本选择器

最基本最常用的选择器

#id（id 选择器）：$('#div1').css('background', 'red')

element（元素选择器）$('div').css('background', 'red')

.class (　属性选择器) $('.box').css('background', 'red')

selector1,selector2,selectorN[取多选择器的并集(组合选择器)]

$('div,span').css('background', 'red')

selector1selector2selectorN[取多个选择器的交集(相交选择器)]

$('div.box').css('background', 'red')

层次选择器

查找子元素，后代元素，兄弟元素的选择器

1. 选中 ul 下所有的的 span

$('ul span').css('background', 'red')

在给定的祖先元素下的后代元素中匹配元素

2. 选中 ul 下所有的子元素 span

$('ul>span').css('background', 'red')

在给定的父元素下的子元素中匹配元素

3. 选中 class 为 box 的下一个 li

$('.box+li').css('background', 'red')

匹配所有紧接在 prev 元素后的 next 元素

4. 选中 ul 下的 class 为 box 的元素后面的兄弟元素

$('ul .box~ ').css('background', 'red')

匹配 prev 元素之后的所有 siblings 元素

5. 选中 ul 下的 class 为 box 的元素后面的兄弟元素

$('ul .box~*').css('background', 'red')

过滤选择器

在原有选择器匹配的元素中进一步进行过滤的选择器

基本

1. 选择第一个 div

$('div:first').css('background', 'red')

2. 选择最后一个 class 为 box 的元素

$('.box:last').css('background', 'red')

3. 选择所有 class 属性不为 box 的 div

$('div:not(.box)').css('background', 'red')

4. 选择第二个和第三个 li 元素

$('li:gt(0):lt(2)').css('background', 'red') //索引起始位置发生变化，重新开始计算

$('li:lt(3):gt(0)').css('background', 'red') //正确索引位置

内容

5. 选择内容为 BBBBB 的 li

$('li:contains(BBBBB)').css('background', 'red')

可见性

6. 选择隐藏的 li

$('li:hidden ').show()

属性

7. 选择有 title 属性的 li 元素

$('li[title]').css('background', 'red')

8. 选择所有属性 title 为 hello 的 li 元素

$('li[title=hello]').css('background', 'red')

表单选择器

表单对象属性

工具

$.each()

遍历数组或对象中的数据

$.trim()

去除字符串两边的空格

$.type(obj)

得到数据的类型

$.isarray(obj)

判断是否是数组

$.isFunction(obj)

判断是否是函数(返回真假就行)

$.parseJSON(json) :

解析 json 字符串转换为 js 对象/数组

.....

ajax

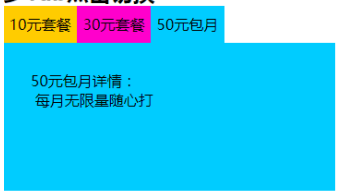
ajax()

get()

post()

.....

多Tab点击切换




```
$(function () {
        //记录上一条的记录
        var currIndex = 0;
        //获取要点击的上面的 div
        $('#tab>li').click(function () {
         //获取点击后面的那个显示和隐藏的 div 数组
            var $lis = $('#container>div');
         //获取当前的 index，就是点击的 this，要用$(this);
             var index = $(this).index();
         //判断当前的位置是不是已经处于的位置，不是就执行，是就不再执行，减少缓存
             if(index !==currIndex){
                  //将上一个的设置为 none，就不用多次循环了
             $lis[currIndex].style.display = 'none';
             //将点击的这个设置为显示
             $lis[index].style.display = 'block';
             //将上一个点击的有恢复到 none 的属性
currIndex = index;}//就是点一个，上一个就恢复到原来的，就一直是两个变化。
})})
```

属性:

1. 操作任意属性（$(元素).att()/r/p）

attr()
removeAttr()
prop()

1）. 读取第一个 div 的 title 属性
$('div:first').attr('title')

2）. 给所有的 div 设置 name 属性(value 为 atguigu)
$('div').attr('name', 'atguigu')

3）. 移除所有 div 的 title 属性
$('div').removeAttr('title')

4）. 给所有的 div 设置 class='guiguClass'
$('div').attr('class', 'guiguClass')

2. 操作 class 属性

addClass()
removeClass()

5. 给所有的 div 添加 class 值(abc)
$('div').addClass('abc')

6. 移除所有 div 的 guiguClass 的 class
$('div').removeClass('guiguClass')

3. 操作 HTML 代码/文本/值

html()
val()

7. 得到最后一个 li 的标签体文本

```
console.log($('li:last').html())
```

8. 设置第一个 li 的标签体为"<h1>mmmmmmmmm</h1>"

```
$('li:first').html('<h1>mmmmmmmmm</h1>')
```

9. 得到输入框中的 value 值

```
console.log($(':text').val())
```

10. 将输入框的值设置为 atguigu

```
$(':text').val('atguigu')
```

设置 css 样式：

设置：元素.css('属性','该属性的值')
添加：元素.css({里面加属性})

```
$('p:eq(1)').css({
    color: '#ff0011',
    background: 'blue',
    width: '300px',
    height: '30px'
})
```

*** offset():** 相对页面左上角的坐标

```
var offset1 =$('.div1').offset();
console.log(off1.left,off1.top);
```

*** position():** 相对于父元素左上角的坐标**;**

```
var position1 = $('.div1').position();
console.log(position1.top,position1.left
```

给属性添加值：

```
$('.div2').offset({
            top:300,
            left:300})
```

*scrollTop():读取/设置滚动条的 Y 坐标

```
$('div').scrollTop()    //读取 div 的滚动条
```

读取页面滚动条的 Y 坐标(兼容 chrome 和 IE)

```
$('body').scrollTop() + $('html').scrollTop()
$('body,html').scrollTop(60);  滚动到 60 位置(兼容 chrome 和 IE)
```

*内容尺寸
height(): height
width(): width
* 内部尺寸
innerHeight(): height+padding
innerWidth(): width+padding
* 外部尺寸
outerHeight(false/true): height+padding+border 如果是 true, 加上 margin
outerWidth(false/true): width+padding+border 如果是 true, 加上 margin

例题：

```
var $div = $('div')
// 内容尺寸
console.log($div.width(),$div.height());
// 内部尺寸
console.log($div.innerHeight(),$div.innerWidth())
//外部尺寸
console.log($div.outerHeight(true),$div.outerWidth(true))
console.log($div.outerWidth(true), $div.outerHeight(true))
```

筛选_过滤

在 jQuery 对象中的元素对象数组中过滤出一部分元素来

1. first()ul 下 li 标签第一个

```
$lis.first().css('background', 'red')
```

2. last()ul 下 li 标签的最后一个

```
$lis.last().css('background', 'red')
```

3. eq(index|-index) ul 下 li 标签的第二个

```
$lis.eq(1).css('background', 'red')
```

4. filter(selector) ul 下 li 标签中 title 属性为 hello 的

```
$lis.filter('[title=hello]').css('background', 'red')
```

5. not(selector) ul 下 li 标签中 title 属性不为 hello 的

```
$lis.not('[title=hello]').css('background', 'red')
$lis.filter('[title][title!=hello]').css('background', 'red')
```

6. has(selector) ul 下 li 标签中有 span 子标签的

```
$lis.has('span').css('background', 'red')
```

父母/兄弟标签

在已经匹配出的元素集合中根据选择器查找孩子/父母/兄弟标签

1. children(): 子标签中找

ul 标签的第 2 个 span 子标签

```
$ul.children('span').eq(1).css('background')
$ul.children('span:eq(1)').css('background', 'red')
```

2. find() : 后代标签中找

ul 标签的第 2 个 span 后代标签

```
$ul.find('span:eq(1)').css('background','pink')
$ul.find('span:eq(1)').css('background', 'red')
```

3. parent() : 父标签

ul 标签的父标签

```
$ul.parent().css('background', 'red')
$ul.parent().css('background', 'red')
```

4. prevAll() : 前面所有的兄弟标签

id 为 cc 的 li 标签的前面的所有 li 标签

```
$ul.children('#cc').prevAll('li').css('background', 'blue')
$ul.children('#cc').prevAll('li').css('background', 'red')
```

5. nextAll() : 后面所有的兄弟标签

6. siblings() : 前后所有的兄弟标签

id 为 cc 的 li 标签的所有兄弟 li 标签

```
$ul.children('#cc').siblings('li').css('background', 'pink')
$ul.children('#cc').siblings('li').css('background', 'red')
```

文档增删改

添加/替换元素

*** append(content) 向 id 为 ul1 的 ul 下添加一个 span(最后)**

```
$('#ul1').append('<span>append()添加的 span</span>')
$('<span>appendTo()添加的 span</span>').appendTo('#ul1')
```

*** prepend(content) 向当前匹配的所有元素内部的最后插入指定内容**

向 id 为 ul1 的 ul 下添加一个 span(最前)

```
$('#ul1').prepend('<span>prepend()添加的 span</span>')
$('<span>prependTo()添加的 span</span>').prependTo('#ul1')
```

*** before(content) 向当前匹配的所有元素内部的最前面插入指定内容**

在 id 为 ul1 的 ul 下的 li(title 为 hello)的前面添加 span

```
$('#ul1>li[title=hello]').before('<span>before()添加的 span</span>')
```

*** after(content) 将指定内容插入到当前所有匹配元素的前面**

在 id 为 ul1 的 ul 下的 li(title 为 hello)的后面添加 span

```
$('#ul1>li[title=hello]').after('<span>after()添加的 span</span>')
```

*** replaceWith(content) 指定内容插入到当前所有匹配元素的后面替换节点**

将在 id 为 ul2 的 ul 下的 li(title 为 hello)全部替换为 p

```
$('#ul1>li[title=hello]').replaceWith('<p>replaceWith()替换的 p</p>')
```

*** empty()删除元素用指定内容替换所有匹配的标签删除节点**

删除所有匹配元素的子元素

*** remove()删除所有匹配的元素**

移除 id 为 ul2 的 ul 下的所有 li

```
$('#ul2').empty()
```

事件绑定(2 种):

*** eventName(function(){})绑定对应事件名的监听**

例如：$('#div').click(function(){});

*** on(eventName, funcion(){})通用的绑定事件监听**

例如：$('#div').on('click', function(){})

*** 优缺点:**

*eventName: 编码方便，但只能加一个监听，且有的事件监听不支持

*on: 编码不方便, 可以添加多个监听, 且更通用

事件解绑:

* off(eventName)

事件的坐标

* event.clientX, event.clientY 相对于视口的左上角
* event.pageX, event.pageY 相对于页面的左上角
* event.offsetX, event.offsetY 相对于事件元素左上角

事件相关处理

* 停止事件冒泡 : event.stopPropagation()
* 阻止事件默认行为 : event.preventDefault()

区别 mouseover 与 mouseenter

* mouseover: 在移入子元素时也会触发, 对应 mouseout
* mouseenter: 只在移入当前元素时才触发, 对应 mouseleave

hover()使用的就是 mouseenter()和 mouseleave()

例题：

```
$('.inner').hover(function () {
    console.log('进入...')
}, function () {
    console.log('离开...')
})
```

事件的委托：

1. 事件委托:

* 将多个子元素(li)的事件监听委托给父辈元素(ul)处理
* 监听回调是加在了父辈元素上
* 当操作任何一个子元素(li)时, 事件会冒泡到父辈元素(ul)
* 父辈元素不会直接处理事件, 而是根据 event.target 得到发生事件的子元素(li), 通过这个子元素调用事件回调函数

2. 事件委托的 2 方:

* 委托方: 业主 li
* 被委托方: 中介 ul

3. 使用事件委托的好处

* 添加新的子元素, 自动有事件响应处理
* 减少事件监听的数量: n==>1

4. jQuery 的事件委托 API

*设置事件委托: $(父元素).delegate('子元素,' 'eventName,多个', function())

* 移除事件委托: $(parentSelector).undelegate(eventName)

```
$('ul').delegate('li', 'click', function () {
    this.style.background = 'red'
})
$('#btn1').click(function () {
    $('ul').append('<li>xxxxxxxxx</li>')
})
$('#btn2').click(function () {
    // 移除事件委托
    $('ul').undelegate()})
```

淡入淡出：不断改变元素的透明度来实现的

1. fadeIn(): 带动画的显示
2. fadeOut(): 带动画隐藏
3. fadeToggle(): 带动画切换显示/隐藏

```
$("#btn1").click(function () {
    $('.div1').fadeOut(1000);
})
```

滑动动画

1. slideDown(): 带动画的展开
2. slideUp(): 带动画的收缩
3. slideToggle(): 带动画的切换展开/收缩

显示隐藏，默认没有动画

1. show(): (不)带动画的显示
2. hide(): (不)带动画的隐藏
3. toggle(): (不)带动画的切换显示/隐藏

jQuery 动画本质 ： 在指定时间内不断改变元素样式值来实现的

1. animate(): 自定义动画效果的动画
2. stop(): 停止动画

```
$('.div1')
   .animate({
      width: '500px'
   }, 2 * 1000)

 .animate({
     height: 200
  }, 2 * 1000) })
```

自定义插件(谁调用 this 就指向谁)

1. 扩展 jQuery 的工具方法
   $.extend(object)
2. 扩展 jQuery 对象的方法
   $.fn.extend(object)

```
$.fn.extend({//加 fn 是 jquery 对象的方法，不加是普通的方法；
    checkAll:function () {//加粗是方法名字，直接调用，和调用方法一样
        this.prop('checked',true);
    },
     UncheckAll:function () {
        this.prop('checked',false);
    },
```

问题 ： 如果有 2 个库都有$, 就存在冲突

解决 ： jQuery 库可以释放$的使用权, 让另一个库可以正常使用, 此时 jQuery 库只能使用 jQuery 了,引用文件要写在后面

API : jQuery.noConflict()