目录

1.什么是JQuery

- 一个快速、简洁的`JavaScript`框架
- `JQuery`兼容各种主流浏览器, 如IE6.0+、FF1.5+、Sofari2.0+、Opera 9.0+ 等
- `JQuery`的好处
  1. 简化`JS`的复杂操作
  2. 不需要关心兼容性
  3. 提供大量实用方法
- 如何学习`JQ`
  1. https://jquery.com/ `JQ`的官方文档
  2. `JQ`只是辅助工具, 要正确面对
  3. 需要分阶段学习
- 下载`JQ` : https://jquery.com/download/#downloading-jquery
  - 和`require.js`一样 拷贝复制
- `JQuery`版本区别
  - `1.0` 可以兼容`IE8`以下
  - `2.0` 只兼容`IE8`以后
  - `3.0`
- 接下来所有的操作, 在三个版本中都是相同的, `JQuery` 很多, 可以去查阅中文文档

2.JQuery设计思想 - 选择网页元素 (模拟css选择元素、独有表达式、多种筛选方式)

1. 模拟`css`选择元素
   - 获取节点: `$("css选择器")`

选择器	写法	描述
id	`$("#box")`	选择所有`id`为`box`的节点
class	`$(".box")`	选择所有`class`为`box`的节点
tagName	`$("div")`	选择所有`div`节点
name	`$("[name=hello]")`	选择所有`name = hello`的节点 `$("[id = box]")` 选择`id=box`的所有元素

   - `input`选择器

写法	描述
`$(":input")`	所有 `<input>` 元素
`$(":text")`	所有 `type="text"` 的 `<input>` 元素
`$(":password")`	所有 `type="password"` 的 `<input>` 元素

    - 还有`radio、checkbox、submit、reset、button、file、image`

- `伪元素` 选择器

写法	描述
`$("p:first")`	第一个 `<p>` 元素
`$("p:last")`	最后一个 `<p>` 元素
`$("tr:even")`	所有偶数 `<tr>` 元素
`$("tr:odd")`	所有奇数 `<tr>` 元素
`$(":empty")`	无子（元素）节点的所有元素
`$("p:hidden")`	所有隐藏的 `<p>` 元素

- `nth-of-type` -> `eq`

写法	描述
`$("ul li:eq(3)")`	列表中的第四个元素（`index` 从 `0` 开始）
`$("ul li:gt(3)")`	列出 `index` 大于 `3` 的元素
`$("ul li:lt(3)")`	列出 `index` 小于 `3` 的元素
`$("input:not(:empty)")`	所有不为空的 `input` 元素

- `$(this)` ： 选取当前 `HTML` 元素
- 更多选择器写法请参照: 或百度 `JQ` 选择器
  https://www.w3school.com.cn/jquery/jquery_ref_selectors.asp

- 修改样式:
    - `$("css选择器").css("样式名称", "值")`
    - `$(".box").css("backgroundColor", "red")`
    - 修改的是行间`style`, 可以获取值, 也可以传入对象

2. 独有表达式选择

- 参考上方 -> `nth-of-type` 和 `伪元素`

  `:eq()` `:gt()` `:lt()`

  `:first` `:even` `:odd`

3. 多种筛选方法
   1. 获取下标为 `2` 的 `li` 标签 -> 可以将 `eq()` 单独取放在 `$()` 之后

      ```
      $("ul li:eq(2)").css("backgroundColor","red")
      $("ul li").eq(2).css("backgroundColor","red")
      ```

   2. 将 `li` 中 `class=box` 修改为蓝色

      ```
      $("li.box").css("backgroundColor","blue")
      $("li").filter('.box').css("backgroundColor","blue")
      ```

3.JQuery设计思想 - JQuery的写法 （方法函数化、链式操作、取值赋值一体化）

1. 方法函数化
   - `window.onload = function(){}`
     - `JQ`：`$(function(){})` === `window.onload`
   - 事件函数 `onclick` ...
     - `JQ`: 去掉了 `on`, 将事件函数化： `$("节点").click( 函数 )`

       ```
       $("h1").click( function(){
           alert('我被点击了');
       } )

       $("h1").mouseover(function(){
           this.style.backgroundColor = "red"
       })
       $("h1").mouseout(function(){
           this.style.backgroundColor = "blue"
       })
       ```

       - 在JQ中 基本见不到 等于号, 所有的赋值操作都是函数传参的操作
2. 链式操作: 在操作函数完成后,跟上另外一个函数

   ```
   $("h1").click(function(){
     alert("点击")
   })
   .css("backgroundColor","orange")
   .mouseover(function(){
     this.style.backgroundColor = "red"
   })
   ```

3. 取值赋值合体 `.html()`
   - 获取标签中的所有内容, 文本形式
     - `js`：`innerHTML`

    - JQ：.html()
        - 获取标签间的内容：$("#div1").html()
        - 设置标签间的内容：$("#div1").html("<h2>我是新赋值的内容<h2>")
  - 获取input中的value值
    - js：value
    - JQ：.val()
        - 获取/设置 input的value值

          $(":text").val()

          $(":text").val("这是新赋值的内容")
  - 如果是 节点数组 取值时只返回第一个节点间的内容

4.JQuery和JS的关系

- 可以共存,不能混用

```
$("#div1")      //JQ对象

alert($("#div1").innerHtml) // 错误
alert($("#div1").html())    // 正确
```

  - 不能一半是JQuery,一半是JS。

5.JQuery常用方法-1 filter('.box') not('.box') has('.box')

- 对已经获取到的网页元素进行筛选
- filter：过滤，选择符合条件的节点

```
// 选择 div的class为box的节点
$("div").filter('.box').css("backgroundColor","orange")
```

- not： filter的反义词，去除符合条件的节点

```
// 选择 div的class不为box的节点
$("div").not('.box').css("backgroundColor","orange")
```

- has：拥有，选择拥有指定类型子节点的元素。

```
// 选择子节点 的 class需要为box的 div节点
$("div").has('.box').css("backgroundColor","orange")
```

5.2 JQuery常用方法-2 prev() next() find('.box')

- prev：查找当前兄弟节点中的上一个节点

```
$("h3").prev().css('backgroundColor',"red")
```

- next：查找当前兄弟节点中的下一个节点

```
$("h3").next().css('backgroundColor',"blue")
```

- `find`：查找符合条件子节点 -> 使用`$("ul li")`也可以,符合`JQ`多种写法

```
$("ul").find("li").css("backgroundColor","red")
```

5.3 JQuery常用方法-3 `eq(下标)` `index()`

- `index`：获取当前节点在兄弟节点中的下标

```
$("h3").index()
```

- `eq(下标)`：通过下标获取指定节点(`nth-of-type()`)

```
$("li").eq(3).css("backgroundColor","orange")
$("li:eq(4)").css("backgroundColor","yellow")
```

5.4 JQuery常用方法-4 `attr('属性')` `attr('属性','值')` `attr({属性: '值'})`

- `attr()`：设置和修改行间属性, 可以设置自定义属性
  - `node.attr('属性')`：取值

    ```
    $("#div1").attr("title")
    ```

  - `node.attr('属性','值')`：修改，**可以设置自定义属性**

    ```
    $("#div1").attr("title",'dd')
    ```

  - `node.attr({})`：一次修改多个值，**可以设置自定义属性**

    ```
    $("#div1").attr({
      title:'world',
      "class": 'ddd',
      yyy:"zzz"
    })
    ```

    - `.css()` 是 修改 `style` 中的单个属性, `attr` 是修改属性的值(`style` 属于节点属性)
    - `.css()` 用法相同, 修改的是行间 `style` , 可以获取值, 也可以传入对象

5.5 选项卡案例重置

```
$(function(){
  $('#div1').find("button").click(function(){
    // 重置样式
    for(var i = 0; i<=$("#div1 div").length;i++){
      $('#div1 button').eq(i).attr('class',null)
      $("#div1 div").eq(i).css("display","none")
    }
    $(this).attr('class',"active")
    $("#div1 div").eq($(this).index()).css("display",'block')
  })
})
```

6.1 JQuery方法 - `addClass('1 2 3')` `removeClass('1 2 3')`

- 操作`class`属性
- `addClass()`：写重复的`class`，会自动去重

```
$('#div1').addClass('box3 box4')
```

- `removeClass()`：只删除存在的`class`，传入不存在的`class`不会报错

```
$('#div1').removeClass('box1 box6')
```

  - 两个方法都可以传多个值，空格隔开
  - 比`attr('class','box')`更简单一些，`attr`在有多个`class`时，需要处理

6.2 JQuery方法 - width

- js获取某一节点的宽高：**返回值纯数字**
  - `offsetWidth`：`width` + `border` + `padding` 包括滚动条
  - `clientWidth`：`width` + `padding` 不包括滚动条
  - `scrollWidth`：`width` + `padding` (不溢出就是`client`,溢出后的那边`padding`不计入其中)
- `.css()`获取宽高：返回值是带字符的字符串(`'100px'`)

```
alert($(".div1").css("width"))   // 100px 带字符的字符串
```

- `JQ`方法获取宽高: **返回的是数字**
  - `width()`：`width`
  - `innerWidth()`：`width` + `padding`
  - `outerWidth()`：`width` + `padding` + `border`
    - `outerWidth(true)`：传入`true`会`margin`一起获取

```
alert($(".div1").width())       // width
alert($(".div1").innerWidth())  // width + padding
alert($(".div1").outerWidth())  // width + padding + border
alert($(".div1").outerWidth(true))  // width + padding + border + margin
```

  - 没有匹配`scrollWidth`：`width` + `padding`(根据实际情况)

6.3 JQuery方法 - 节点操作 节点.remove()

- insertBefor()

```
$("new").insertBefore($("old")) // 将new节点 插入到 old节点前面
```

- insertAfter()

```
$("new").insertAfter($("old")) // 将new节点 放入到 old节点的后面
```

- appendTo()

```
$("span").appendTo($("#div1")) // 将span 插入到 #div1的结尾
```

- prependTo()

```
$("span").prependTo($("#div1")) // 将span 插入到 #div1的开始
```

- insertBefor、insertAfter、appendTo、prependTo和insert等节点操作注意点,
  - **第一个节点是 节点数组,第一个节点数组中的所有节点, 会插入到第二个节点**
  - **第二个节点是 节点数组, 第二个节点的所有子节点中,都会 插入第一个节点**
  - JQ中取值, 只能取得第一个符合条件的元素值
  - JQ中赋值, 可以批量操作符合条件的节点
- remove() 删除节点

```
$('em').remove()
```

- before()

```
$("node1").before("node2") // node1的前面是node2
```

  - insertBefore() 和 before() 区别
    - 语文解释:
      - before() : div 前面是 span
        - $("#div1").before("span")
      - insertBefor() : 找到 span 节点 插入到 div 的前面
        - $("span").before("#div1")
    - 链式操作: 链式操作,操作的是第一个节点(节点数组), **提供两组不同的方法,是为了更好的链式操作**
    - 同类的还有 after()、append()、prepend()

6.4 JQuery方法 - 事件绑定 on off

- JQ **事件函数 的 底层是事件监听器, 可以绑定多个同类型事件**

- $().on() 事件绑定 4种写法, node.on('事件类型','触发对象选择器', e.data, function)

1. `node.on('事件类型',function)`
   - 给一个事件添加一个函数
2. `node.on('事件类型1 事件类型2', function)`
   - 同时给多个事件添加一个函数, 通过空格隔开
3. `on({ 事件类型1: F, 事件类型2: F })`

```
on({
  事件类型1: function,
  事件类型2: function
})
```

   - 给不同的事件添加不同的函数
4. `on.('事件类型','触发对象选择器',function)`
   - 触发对象选择器: this指向触发对象
     - 底层：事件委托

- 节点的创建: `$( "<li> 这里是内容 <li/>" )` 创建了一个 `li` 节点

- `$().off()` 事件取消绑定 `5种写法`
  1. `node.off()`

```
$("#div").off() // 取消所有事件上所有函数
```

  2. `node.off('事件类型')`

```
$("#div").off('click')  // 取消某一个事件上的所有函数
```

  3. `node.off('事件类型',函数名)`

```
$("#div").off('click', show) // 取消某个事件上的某一个函数
```

  4. `node.off("事件类型1 事件类型2", show)`

```
$("#div").off('click mouseout', show) // 取消几个事件上某一个函数
```

  5. `node.off({事件类型1: F, 事件类型2: F})`

```
node.off({
  事件类型1: 函数,
  事件类型2: 函数
})   // 取消不同的事件中不同的函数
```

- 可以通过链式操作, 来增加/取消 同一个事件上的多个函数

6.5 JQ方法 - `scrollTop()` 获取页面滚动高度

- `scrollTop()`

    - 获取：`$(window).scrollTop()`
    - 设置：`$(window).scrollTop(100)`
        - 只能传数字，`100=100px`
- 练习：点击按钮 显示一个`div`， 滚动和放大缩小时，都可以使其居中
    1. 点击按钮 让`div`显示并居中，并且给`window`添加`scroll/resize`事件
    2. 关闭`div`让`div`消失， 使用`off`移除`window`的`scroll/resize`事件

6.6 JQ方法 - 阻止事件冒泡 阻止默认行为 e e.pageX e.which

- **不能为了使用`JQ`而使用`JQ`**。如果实际开发的时候`JS`一些比较简单的操作,没必要强制写成`JQ`代码
- `JS`阻止`a`链接的默认行为 和 事件冒泡

```
$("a").click(function(e){
  e.preventDefault()  // 阻止a链接的默认行为, IE:e.returnValue=false
  e.stopPropagation() // 阻止a链接，还有IE的 e.cancelBubble = true
})
```

- JQ阻止 默认行为 和 事件冒泡, 只需要 在函数最后 `return false`

```
$("a").click(function(e){
  return false;
})
```

    - JQ中的 `e` 只要在 事件函数中写形参`e`就可以

- `JS` 获取 鼠标 距离 `e.pageX`

属性	原点位置
`e.pageX`， `e.pageY`	整个页面左上角 [包含滚动距离]
`e.clientX`， `e.clientY`	可视窗口的左上角为原点
`e.screenX`， `e.screenY`	电脑屏幕的左上角 `window.screenX/Y` 获取的是浏览器距离屏幕左上的位置
`e.offsetX`， `e.offsetY`	获取鼠标距离`target`元素的位置

    - 返回值：纯数字.
    - 以上都是鼠标事件(`e`)的属性
    - `Top/Left`
        - `node.offsetTop` ： 距离第一个有定位的父节点距离
        - `node.clientTop` ： 边框宽度
        - `node.scrollTop` ： 滚动的距离
- `JQ`中的`e.which`

- `e.which` 在各个不同类型事件中有不同含义
  - 鼠标事件: 是 `JS` 的 `e.button`
    - `e.which`：`1` 左键 `2` 滚轮 `3` 右键
    - `e.button`：`0` 左键
  - `keydown`：`keyCode` 键码
  - `keypress`：`charCode` 字符码

  `e.which` 在不同的事件函数中, 有着不同的定义

6.7 JQ方法 - `node.offset()` 和 `node.position()`

- `JS`：`this.offsetTop` 获取节点距离第一个有定位的父节点距离, `margin` 不算在节点内
  - 返回值：距离第一个有定位的父节点 + 节点的 `margin`

- `JQ`：`offset().left/top` `position().left/top`
  - `node.offset().left/top`：获取当前元素,距离页面 最左/上边的距离 `margin` 不算在节点内
    - 距离最左/上边的距离 + 节点的 `margin`

    ```
    $("#div2").offset().left
    ```

  - `node.position().left/top`：获取当前元素，与第一个有定位的父节点之间的距离，`margin` 算在节点内

    ```
    $("#div2").position().left
    ```

- `node.offsetParent()`：找到第一个有定位的父节点, 可以操作这个节点

7.1 JQ特效函数 `val()` `size()` ->length `each(function(index,item){})`

- **`JQ` 中取值, 只能取得第一个符合条件的元素值**
- **`JQ` 中赋值, 可以批量操作符合条件的节点**
  - 以上对于 `.css()` `.attr()` `.html()` `.scrollTop()` ... 同理

- `val()`：获取/设置表单属性的值(`value`)
  - 获取： 只能取得第一个符合条件的元素值

    ```
    $("input").val()
    ```

  - 赋值： 可以批量操作符合条件的节点

    ```
    $("input").val('1111')
    ```

- `size()`：获取节点数组 中 节点的个数, `length` 属性也可以

  ```
  $("input").size() == $("input").length
  ```

    - 此处的`length`是`JQ`封装的`length`
- `each( function(index, item){} )`：循环 节点/节点数组

```
$("input").each(function(index,item){
  console.log(index,item);
  item.value = index; // js赋值
  $(item).val(index)  // jq赋值
  // index(当前节点下标), item(节点本身)
})
```

7.2 JQ特效函数 `hover(F1, F2)` `hide(毫秒,回调)` `show()` `slideUp/Down()` `fadeIn/Out/To()`

- `hover(移入函数, 移出函数)`：添加移入移出事件

- `hide(毫秒, 回调函数)` 节点隐藏
    - 毫秒: 动画持续的毫秒数
    - 回调函数: 回调函数, 动画结束执行
- `show()`：节点显示
    - 动画效果是从左上角收起和展开
    - 可以不传参, 动画默认有 持续时间
    - 所有动画都有回调函数, 大致格式如下
      `hide(持续时间-毫秒, 回调函数-动画播放完毕会执行)`

- `slideDown(毫秒, 回调函数)`
- `slideUp()`：动画效果, 卷闸效果

- `fadeIn(毫秒, 回调函数)`：淡入
- `fadeOut(毫秒, 回调函数)`：淡出
- `fadeTo(时间, 透明度0~1, 回调函数)`
    - **叠加触发动画, 会排队依次执行**

```
$("#div1").hover(function(){
  // hover的移入函数
  $("#div2").show(500,function(){
    $("#div1").html('移入')
  })
},function(){
  // hover的移出函数
  $("#div2").hide(500,function(){
    $("#div1").html('移出')
  })
})
```

7.3 JQ特效函数 `animate(obj, duration,["linear"], 回调)`

- `animate(obj, duration,["linear"], 回调)`
  - `obj: {属性: 值,width: 100}` ： 要变换的属性, 值为数字,默认为 `px`
  - `duration`： 动画持续时间
  - `linear`： 匀速, 不传参默认 慢快慢
  - 回调函数: 在动画执行完毕后执行

  - 动画 是 属于异步执行, 多个动画会进行等待,但是非动画(`css()`)不会等待前方动画执行
    - `node.animate().animate().css()`
      - 第二个 `animate` 会等待第一个 `animate` 执行完毕后执行
      - `.css()` 不会等待动画执行完毕, 而是和动画异步执行

  - 默认的运动形式是 慢快慢
    - 匀速: 在第三个参数传入 `"linear"`
  - 扩展更多 `animate` 运动形式
    - 引入 `jquery-ui` (`jq` 扩展版本，基本被淘汰)
    - 下载: https://code.jquery.com/ui/1.10.4/jquery-ui.js
    - 文档: https://www.jqueryui.org.cn/demo/5735.html
    - 使用了 `JQ-UI` 后 `addClass()` 和 `removeClass()` 就变成了增强版
      - 可以添加动画持续时间 `addClass("box",400)`

7.4 JQ停止动画函数 `stop()` `finish()` `delay()`

- 为了**解决**: **叠加动画次数, 会依次执行**
  - **动画之间是同步, 动画和其他函数是异步操作**
- `stop()`
  1. 无参数, 停止当前运动的动画, 后续链式操作动画不受影响

     ```
     $("#div1").stop()
     ```

  2. 传入 `true` , 停止所有动画

     ```
     $("#div1").stop(true)
     ```

  3. 传入两个参数, `stop(true,true)` , 停止所有动画， 并使当前正在运动的动画直接达到目的值

     ```
     $("#div1").stop(true,true)
     ```

- `finish()` ： 停止所有动画, 并使所有动画直接达到目的值

  ```
  $("#div1").finish();
  ```

  - 解决动画叠加: 每次调用 `animate` 之前 调用一次 `stop()` 停止之前的动画

```
$("div").click(function(){
  $(this).stop(true).animate()
})
```

- `delay(毫秒数)` ： 延迟, 只针对动画生效, 对后面的 `animate` 生效

```
$(this).delay(4000).animate({width:300},2000).animate({height:300},2000)
```

  - 此时会延迟 4秒才能 执行第一个 `animate`

8.1 JQ方法 `remove()` `detach()`

- `remove()` 和 `detach()` 都**会将删除的节点返回**

- `remove()` ： 删除元素节点
  - 并**不会保留**这个节点上**之前的事件和行为**
- `detach()` ： 删除元素节点
  - 会**保留这个节点上之前的事件和行为**

- 删除后节点 返回节点, 可以进行链式操作

```
$("#div1").remove().insertAfter($(this))
$("#div2").detach().insertAfter($(this))
```

8.2 JQ方法 - `ready()` 和 `$(function(){ })` 概念

- `ready()` ： 属于事件

```
$(document).ready(F) // 事件触发在当前document(文档)加载完成后执行
```

- `$(function(){ })`

```
$(function(){
  // 相当于 $(document).ready(function(){})
})
```

- `window` 包括 `location` `histort` 之类， `document` 就是 `html` 开始到结束
  - `document` 加载完毕肯定是在 `window` 加载完毕之前的
- `window.onload` 肯定不如 `$(F)` 或 `ready()` 加载快

8.3 JQ方法 - `html()` `text()`

- `JQ` 访问标签间内容
  - `html()` ： 标签间的内容 相当于 `node.innerHTML`
  - `text()` ： 标签间纯文本 相当于 `node.innerText`
    - `text()` 赋值不会解析标签

8.4 JQ方法 - 节点操作

- 节点操作方法
  - 下述所有的方法参数都是选择器, 可以二次过滤
  - `siblings()` ： 获取除当前节点外的 所有兄弟节点

```
$("#p1").siblings('h2').css('background','blue')
```

  - `nextAll()` `prevAll()` ： 获取 从此节点之 后/前 的所有兄弟节点
    - `next()` `prev()` 获取只能获取一个

```
$("#p1").nextAll('h2').css('background','blue')
$("#p1").prevAll('h2').css('background','blue')
```

  - `nextUntil()` `prevUntil()` `parentsUntil()`
    - `nextUntil()` 和 `prevUntil()` ： **获取从此节点之 后/前 到 传入参数节点中间的节点,不包括 此节点和找到的第一个节点**

```
$("#p1").nextUntil('h2').css('background','blue')
// 获取 id=p1 至 下方第一个h2标签中的所有节点, 不包括p1和h2
```

    - `parentsUntil()` ： **获取此节点 到下一节点的所有父节点**

```
$("#p1").parentsUntil('body').css('background','blue')
// 获取p1 到 body中间的所有父节点, 不包括body
```

9.1 JQ方法 -查找父节点 `parent()` `parents()` `parentsUntil()` `closest('参数')`

- `parent()` ： 获取父节点
- `parents(节点选择器)` ： 获取父节点们
  - 不传参, 获取所有父节点; 传参就筛选符合条件的父节点
- `node.parentsUntil( 父节点选择器 )` ： 获取节点到 父节点中间所有的节点，不包括传入的父节点
- `closest(选择器)` 必须传入参数, 参数也是选择器, 只获取符合条件的父元素，包括本节点
- 扩展: `$().children()` ： 获取子节点
  - `js` `node.children` ： 返回node的子节点(不返回子节点的子节点)
  - `JQ` `$().children()` ： 可以传入选择器参数, 筛选符合条件的节点

9.2 JQ方法 - `wrap`

- `wrap('<p></p>')` ： 每一个获取到的元素节点单独包装
- `wrapAll('<p></p>')` ： 整体包装 - 将所有节点整体包装, 放入第一个包装节点内
- `wrapInner('<p></p>')` ： 内部包装 - 节点内部塞入 创建的节点(节点里的原本的内容也会塞进包装节点)
- `unwrap()` ： 删除包装，删除上面一层包装,(删除父节点,如果父节点是`body`则不删除)

    - 除了 `unwrap()` 不用传参,其余都要传入(非 `JQ` 创建节点)
        - `JQ` 创建节点: `$("<p></p>")`
        - `warp`： `wrap( '<p></p>' )`

- `warp` 作用: 统一给某些节点 外面 或 内部添加节点/内容时

9.3 JQ方法 - `clone(true)`

- `clone()` ： 默认**克隆节点本身，不会克隆行为和事件**
- `clone(true)` ： **克隆节点和节点的事件和行为**

```
$("#div1").clone().appendTo($("#div2"))
```

9.4 JQ方法 - `add()` `slice(start, end)`

- `add("选择器")` ， 可以将选择器拼接到一起 (可在链式操作中途写入)
    - 如果两种选择器 事件只有部分相同, 可在中途拼接

```
node.click().add(node2).mouseleave()
```

        - `node` 拥有 `click` 和 `mouseleave()`
        - `node2` 只拥有 `mouseleave()`
- `slice(start,end)` 获取 `[strat,end)` 间的元素节点,类似 `eq()`
    - `JQ` 获取不同节点时, 顺序是页面从上到下

9.5 数据串联化 - `serialize()` `serializeArray()`

- 以下作用于表单节点 `from`
- `serialize()` ： 将表单中的数据拼接成 `querystring` (查询字符串) 不带 `?`
    - `name1=value1&name2=value2`
- `serializeArray()` ： 将表单数据拼接成数组
    - `[{name:value}, {name2:value2}]`

9.6 JQ事件细节 - `e.data` `e.target` `e.type` `node.trigger('事件类型')` `node.data`

- `e.data` ： 事件函数传参, 配合 `on` 使用, 在触发对象选择器后填入

```
$(node).on('事件', '触发对象选择器', {name:value}, function(){} )

$('button').on('click', {username:'钢铁侠','age':18},function(e){
    console.log(e.data) // {username:'钢铁侠','age':18}
})
```

- `e.target` ： 兼容后的触发对象
- `e.type` ： 输出事件类型

- `node.trigger('事件类型')` ： 主动触发事件，可以触发 自定义事件

```
$(this).trigger('pause')
```

  - **一次只能传入一个事件**, 可以链式操作触发多个事件

- `node.data()` ： 节点上的自定义属性
  - `html5` 规定 自定义的属性要以 `data-*` 开头(`data-cname="张三"`)
  - `JS5` 使用 `node.dataset` 属性获取节点上的所有自定义属性(返回对象)
    - `node.dataset` ： 获取 `node` 上所有的自定义属性
    - `node.dataset.cname` ： 获取、修改 `data-cname` 的值
    - 删除值: `delete node.dataset.sname`

  - `JQ` `node.data()` 方法 获取节点上的所有自定义属性 (返回对象)

```
$(this).data()      // 获取所有
$(this).data('key') // 获取key对应的值
$(this).data('key','value') // 修改
$(this).removeData('sname') // 删除
```

10 JQ工具方法 - `$.type()` 和我们自己封装的js方法没有任何区别

- `JQ` 工具方法和我们自己封装的 `js` 方法没有任何区别
  - `JQ` 工具方法基本都没用了, `ECMA5` 和 `6` 都有类似功能的函数

- `JQ` 的方法调用: `$().xxx()` ， 必须 `JQ` 对象调用这个函数
- `JQ` 的工具方法: `$.xxx()` ， `js` 获取的对象也可以调用

工具方法	描述
`$.type()`	输出当前数据类型，比 `JS` 的 `typeof` 好用，可准确输出数组、日期
`$.trim()`	`JS` ： `str.trim()` 删除字符串的前后空格,中间的空格不删除
`$.inArray()`	`JS` ： `indexOf()`
`$.proxy()`	`JS` ： `bind`
`$.noConflict()`	给 `$` 起别名 `var a = $.noConfliict();`
`$.paraseJSON()`	`JS` ： `JSON.parse()` ， `json` 转 字符串
`$.makeArray()`	`JS` ： `Array.from(arr)` ，将伪数组转为真数组

  - 除了 `$.type()` 其他的都没什么用

11 JQ插件方法 - `$.extend({a:F})` `$.fn.extend({a:F})` 深拷贝

- `$.extend()` ： 拓展工具方法 `$.xxx()`
- `$.fn.extend()` ： 拓展 `JQ` 方法 `$().xxx()`
- `JQ` 插件方法， 如果我们想要给 `JQ` 新增函数， 通过以上两个插件方法拓展函数库
  - 扩展 `JQ` 方法后如果需要有链式操作， 需要在方法内返回 `$(this)`

```
$.extend({
  aaa: function(){
    console.log('这是一个工具方法')
  }
})

$.fn.extend({
  aaa: function(){
    console.log('这是一个JQ方法')
    return $(this);
  },
  drag(){
    // 拖拽看代码
  }
})
```

- `JQ` 深拷贝： `$.extend(true,{},a,b)`
  - `$.extend(true,{},a,b)`
    - `true` ： 是否深度拷贝,不加 默认为 `false` ， 浅拷贝， 加了深拷贝
    - `{}` ： 将合并结果保存到新对象， 这样原对象将不会发生改变
    - `a` ： 第一个合并的对象
    - `b` ： 第二个合并的对象, 可选

      【注】 `a` , `b` 有相同基层属性, 则覆盖; `a` 的基层属性b`没有, 则保留
      - 基层属性: 此属性值不为 引用数据类型
- JQ浅拷贝： `$.extend({},a,b)`

12 JQ的 cookie

- `$.cookie(name)` ： 通过 `name` 取值
- `$.cookie(name,value)` ： 设置 `name` 和 `value`
- `$.cookie(name, value,{ 可选项 })`

```
$.cookie(name, value,{
  // 可选项
  expires: 7,
  path: '/',
  domain: 'example.com',
  secure: true, // 开启后，只能https访问cookie
  raw: true // 不对数据进行编码
    // false(默认值)(对数据进行编码)
})
```

- `$.cookie(name, null)`：删除cookie

12.2 JQ的 ajax

```
$.ajax({
  type: "get",
  url: '',
  data: {},
  success:function(data,statusText,xhr){
    // statusText  返回下载状态 "success" "error"
    // xhr     ajax对象
  },
  error: function(msg, statusText){
    // msg : 返回xhr对象
    // statusText  返回下载状态 "success" "error"
  },

  dataType: "jsonp" // JSONP跨域请求
    // dataType -> 告诉jq,服务器端返回内容的格式,
      //包括xml、html、script、json、text和jsonp, 方便解析
})
```

- `dataType: "jsonp"` 就可以进行跨域请求

12.3 JQ的 ajax的方法 - `load('url',F)` `$.get('url',F)` `$.post('url',data,F,type)`

- `load("url [可跟节点选择器]", function(data, statusText, xhr){})`
  - 将`url`传入后, 将下载到的数据直接填充到被选中元素的`innerHTML`中(覆盖原数据)
    - `$("div").load('2.txt h2',function(){})`
      - 将 下载并且经过`JSON`转换后的数据 筛选符合条件的数据, 放入节点内
    - `load()` 可以进行`get`请求 无法进行`post`请求, 不具备`$.ajax`的`dataType`

- `$.get('url',data,回调函数,type)`
  - `get` 可以省略`data`参数, 在`url`后`?n=v&`
- `$.post('url',{u:n},回调函数,type)`
  - `type`：告诉`jq`，服务器端返回内容的格式，包括`xml`、`html`、`script`、`json`、`text`和`jsonp`， 方便解析， 和`$.ajax`中的`dataType`一致

13.JQ放大镜

14.banner图

15.购物车案例

- `JQ`的获取节点, 都是获取已有节点, 如果想 新加入的节点也拥有事件, 需要用到事件委托, `on('click','委托对象',function(){})`