#jQueryday01:

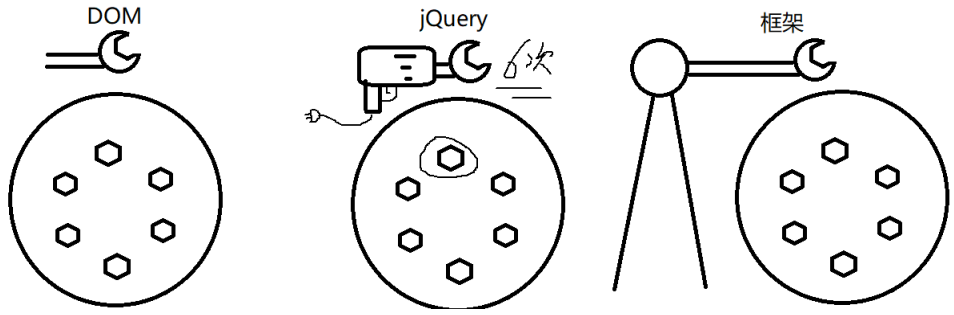
张景元

正课:

1. 什么是 jQuery
2. 如何使用 jQuery
3. jQuery 原理
4. 查找元素
5. 修改元素

一. 什么是 jQuery

1. 什么是 jQuery:
   (1). 第三方开发的: 不是原生 js，需要先下载才能使用，比如 bootstrap
   (2).执行 DOM 操作的: jQuery 还是在做 DOM 增删改查+事件绑定
   学习 jQuery 其实还是在学习 DOM5 件事
   (3).极简化的: 因为原生 DOM 繁琐，所以 jQuery 对 DOM 五件事进行了简化
   jQuery 是简化版的 DOM
   (4).函数库: jQuery 中都是用函数来解决一切问题！jQuery 是没有属性的！
   jQuery 中一切操作都要加()
2. 为什么: 2 个原因:
   (1). 简单: jQuery 将 DOM 操作的每一步几乎都进行了终极的简化
   (2). 解决了大部分浏览器兼容性问题：凡是 jquery 让用的，都没有兼容性问题
3. 何时: 在前端 js 框架出来之前，旧的项目和框架几乎都是用 jquery 做的！
   但是，前端 js 框架出来之后，jquery 使用越来越少！
4. 问题:
   (1). 只有 PC 端，没有移动端！
   (2). 仅仅对 DOM 的每一步进行了简化，但是没有彻底简化 DOM 开发的步骤！



二. 如何使用 jQuery:

1. 下载: 官网: www.jquery.com
   你们不用下载，在每天压缩包的 js 文件夹下已经下载好了
2. 版本:
   (1). 1.x 唯一支持旧浏览器(IE8)
   a. 未压缩版: 包含完备的注释，代码结构和见名知意的变量名，所以适合学习和开发之用。但是体积大，不便于快速下载运行，所以不适合生产环境使用。
   我们上课用 jquery-1.11.3.js
   b. 压缩版: 去掉了所有注释和代码结构，极简化了变量名，所以体积小，便于快速下载

和运行，适合生产环境使用。但是可读性差，不适合学习和开发之用。

(2). 2.x 不再支持旧浏览器，也不支持 ES6，比如不能用 for of 遍历 jquery 对象

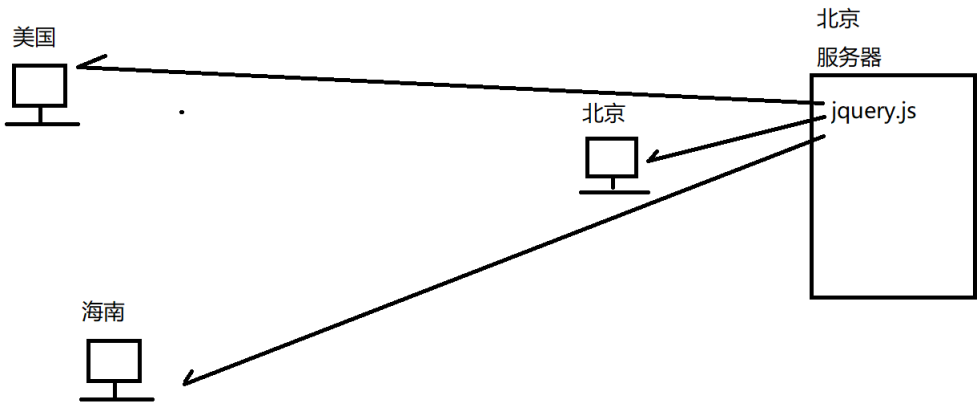
(3). 3.x 也不再支持旧浏览器，但是，开始支持 ES6，比如可以用 for of 遍历 jquery 对象

3. 必须引入网页中才能使用: 2 种:

(1). 将 js 文件下载到项目本地和项目文件放在一起使用相对路径加载到页面中

a. <script src="js/jquery-1.11.3.js">

b. 问题: jquery.js 文件只能部署在一个地点的服务器上。距离服务器位置不同的用户，访问时，下载时间会受距离的影响!

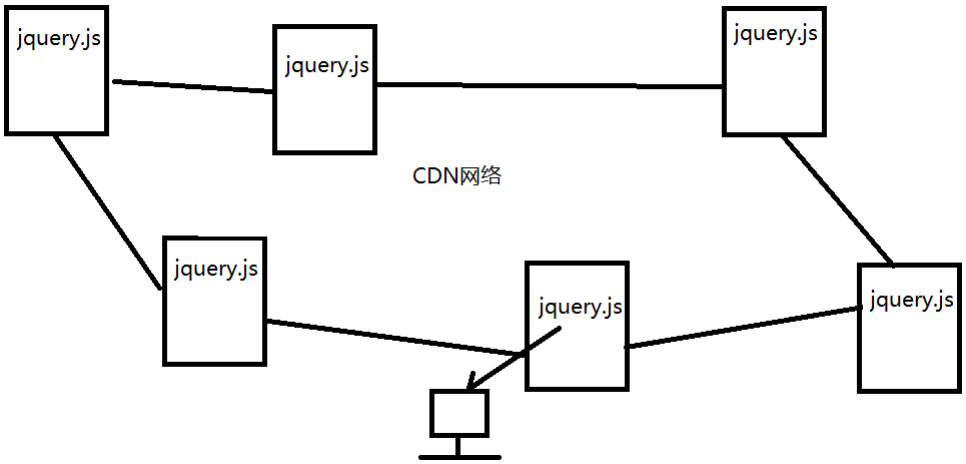


(2). 使用 CDN 网络中共享的 jquery.js 文件

a. CDN: (智能)内容分发网络

1). 可在全球多个地点的服务器上共享通用的第三方文件

2). 客户端访问 CDN 网络中的文件时，CDN 网络会自动根据用户所在位置距离全球哪台服务器网络状况最优，来动态为客户端选择下载文件的服务器。



b. 优点: 无论身处何地的客户端下载共享的文件时，都能获得差不多一致的下载速度。

c. 如何: 其实所有第三方的框架或库，都提供了 CDN 网址

1). 不用下载 js 文件到本地

2). <script src="官方提供的 CDN 网址">

4. 示例: 第一个 jQuery 程序: 点击按钮显示点击次数

1_DOM.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <button id="btn1">click me(0)</button>
  <script>
    //DOM 4 步:
    //1. 查找触发事件的元素
    //本例中: 用户点 id 为 btn1 的按钮触发变化
    var btn1=document.getElementById("btn1");
    //2. 绑定事件处理函数
    //本例中: 用户单击按钮触发变化
    btn1.onclick=function(){
      //3. 查找要修改的元素
      //本例中: 就是要修改当前按钮自己，所以不用找
      //就是 this->btn1
      //4. 修改元素
      //本例中: 修改元素的内容,3 步
      //4.1 取出旧内容
      //本例中: 取出按钮的内容，并截取()内的数字，转为整数
      var n=parseInt(this.innerHTML.slice(9,-1));
```

parseInt(
btn1 <button id="btn1">click me(0)</button>
.innerHTML "click me(10000)"
0123456789 -1
.slice(9, -1)
"10000"

) 10000

```
      //4.2 计算新值: 本例中: n++
      n++;
      //4.3 将新值放元素内容中
      //本例中: 将新值 n，拼接上原来完整的字符串，再放回按钮内容中
      this.innerHTML=`click me(${n})`;
    }
  </script>
</body>
</html>
```

运行结果:

1_jQuery.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <button id="btn1">click me(0)</button>
  <!--0. 必须先引入 jquery.js-->
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //DOM 4 步:
    //1. 查找触发事件的元素
    //本例中: 用户点 id 为 btn1 的按钮触发变化
    //var btn1=document.getElementById("btn1");
    //            选择器
    var $btn1=$("#btn1");
    //2. 绑定事件处理函数
    //本例中: 用户单击按钮触发变化
    //btn1.onclick=function(){
    $btn1.click(function(){
      //3. 查找要修改的元素
      //本例中: 就是要修改当前按钮自己，所以不用找
      //就是 this->btn1
      var $this=$(this);//先不要问!
      //4. 修改元素
      //本例中: 修改元素的内容,3 步
      //4.1 取出旧内容
      //本例中: 取出按钮的内容，并截取()内的数字，转为整数
      //var n=parseInt(this.innerHTML.slice(9,-1));
      var n=parseInt($this.html().slice(9,-1))
      //4.2 计算新值: 本例中: n++
      n++;
      //4.3 将新值放元素内容中
      //本例中: 将新值 n，拼接上原来完整的字符串，再放回按钮内容中
      //this.innerHTML=`click me(${n})`;
      $this.html(`click me(${n})`)
    //}
    })
  </script>
```

```
</body>
</html>
```

运行结果:

三. jQuery 原理:

1. 当我们引入<script src="js/jquery-1.11.3.js">时，其实是给内存中添加了一种新的类型，包含两部分:

(1). 构造函数: 专门负责创建该类型的子对象

function jQuery(){ … }

(2). 原型对象: 替该类型所有子对象集中保存共有的方法

在 jQuery.prototype 中，集中保存的就是所有简化版的 DOM 操作的函数

(3). 只有 jQuery 构造函数创建的子对象才能使用 jQuery 原型对象中保存的简化版函数别人家的子对象，无权使用!

2. 所以，要想使用 jQuery 家简化版函数，必须先创建 jQuery 家的子对象。如何创建:

(1). var jq 子对象=new jQuery(…)

(2). 问题: jq 子对象调用 jQuery 原型对象中的方法是为了对 DOM 元素执行操作!

所以，jq 对象应该提前找到并保存住要操作的 DOM 元素才行

(3). 所以，创建 jQuery 对象时，必须提前放入一个要操作的 DOM 元素对象，2 种:

a. 先用选择器查找 DOM 元素，再装入 jq 子对象中备用:

var jq 子对象=new jQuery("选择器")

b. 如果已经获得一个 DOM 元素(this 或 e.target)，则无需再查找，就可直接放入 jq 子对象中:

var jq 子对象=new jQuery(DOM 元素对象)

(4). 问题: 每次操作一个 DOM 元素时，都需要重复写 new jQuery()，太长了!

(5). 解决: 将 new jQuery()封装进了 jQuery()构造函数内部!

```
function jQuery(  选择器或 DOM 元素对象  ){

    return new jQuery 的真正构造函数(   …   )
}
```

结果: 将来调用 jQuery(xxx)等效于调用 new jQuery(xxx)，省略了 new!

(6). 问题: 作者连"jQuery"都懒得写

(7). 解决: 给 jQuery 起了个别名/外号: $=jQuery

(8). 总结: 今后创建 jQuery 子对象，只要写:

var jq 子对象=$("选择器"或 DOM 元素对象)

$()=>jQuery()=>new jQuery()

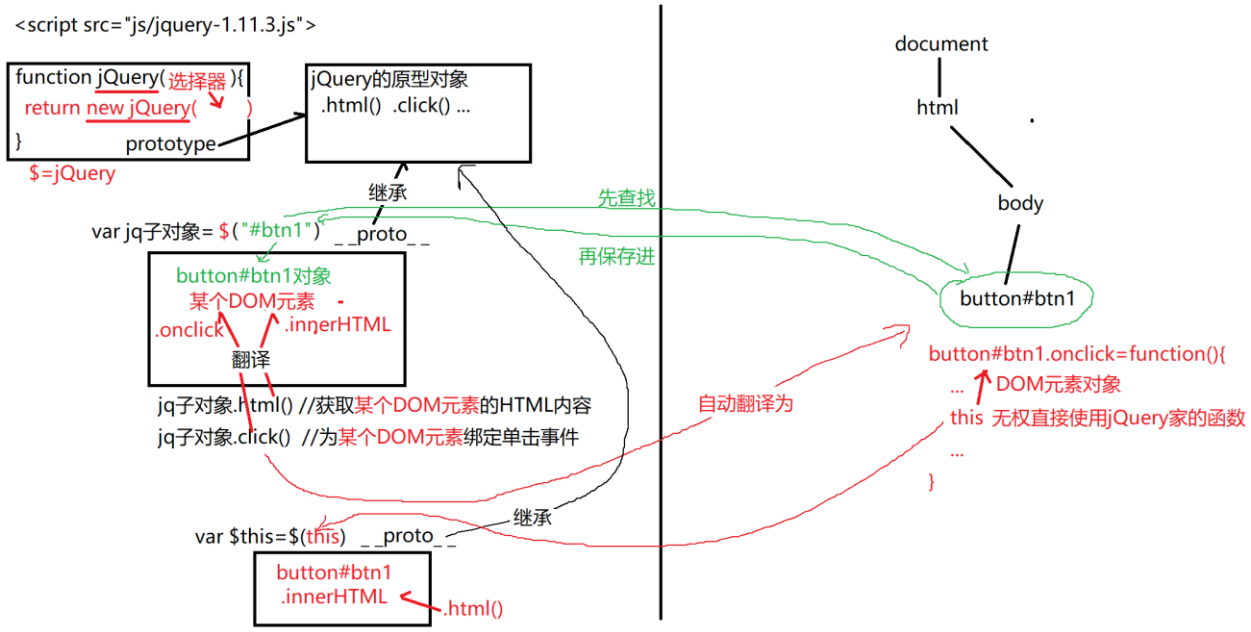
3. 结果: 获得了一个装有 DOM 元素对象的 jQuery 家子对象，简称 jq 对象

4. 对整个 jq 子对象调用简化版函数时，jq 子对象会自动将简化版函数，翻译为原生的 DOM 对应的函数，作用到 jq 子对象中保存的 DOM 元素对象上

5. 变量命名习惯: 今后为了和 DOM 家的元素对象区分，所有保存 jQuery 子对象的变量名，习惯上都要以$开头! 但是，不是必须!

今后在 jquery 程序中，只要看到$开头的变量，几乎都是 jQuery 家的子对象，其中都保

存着 DOM 元素对象。



6. jQuery 子对象/查找结果对象的本质: 类数组对象

(1). 选择器找到几个 DOM 元素，jQuery 子对象中就包含几个 DOM 元素对象。

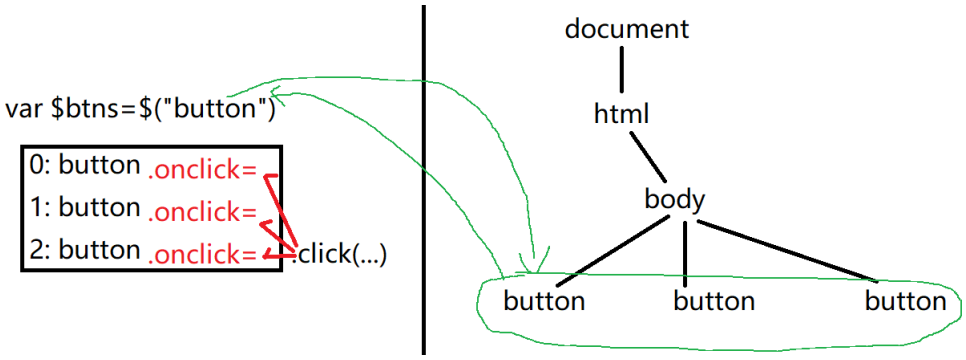
(2). 每个 DOM 元素对象都有下标。

```
jQuery.fn.init(3) [button#btn1, button#btn2, button#btn3, prevObject: jQuery.fn.init(1), context: document, selector: "button"]
  0: button#btn1
  1: button#btn2
  2: button#btn3
  context: document
  length: 3
  prevObject: jQuery.fn.init [document, context: document]
  selector: "button"
  __proto__: Object(0)
```

7. jQuery 简化版函数的三大通用特点:

(1). 每个 jQuery 简化版函数都自带 for 循环:

只要对整个 jQuery 子对象调用一次简化版函数，就相当于对 jQuery 子对象内部保存的每个 DOM 元素都分别调用了一次对应的原生方法或属性!



(2). 凡是和修改相关的函数，都一个函数两用: —— 重载!

a. 如果没有给新值，则该函数默认执行读取旧值的操作

比如: $btn1.html()　获取按钮的 innerHTML 内容

b. 如果给了新值，则该函数改为执行修改值的操作

比如: $btn1.html("hello")　修改按钮的内容为"hello"

(3). （碰到在讲...）

(4). 示例: 三个按钮，点击每个按钮，分别记录每个按钮的点击次数

2_jQueryAPI.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <h1>jQueryAPI 特点</h1>
  <button id="btn1">click me(0)</button>
  <button id="btn2">click me(0)</button>
  <button id="btn3">click me(0)</button>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //DOM 4 步
    //1. 查找触发事件的元素
    //本例中: 三个按钮都能点
    //        css 元素选择器
    var $btns=$("button");
    console.log($btns);//包含三个 button 元素对象的类数组对象
    //2. 绑定事件处理函数
    //本例中: 虽然三个按钮都能点击，但是 jQuery 简化版函数自带 for 循环，只要调用一次 click()就可给三个按钮都绑定上单击事件处理函数
    $btns.click(function(){
      //3. 查找要修改的元素
      //本例中: 就是要修改当前按钮自己，所以不用找
      var $this=$(this);//this->DOM 按钮对象，不能直接使用 jQuery 家函数.
      //必须 new jQuery(this)，转为 jq 子对象才有资格使用 jQuery 家的简化版函数。
      //4. 修改元素
      //本例中: 修改元素的内容,3 步
      //4.1 取出旧内容
      //本例中: 取出按钮的内容，并截取()内的数字，转为整数
      var n=parseInt($this.html().slice(9,-1))
      //4.2 计算新值: 本例中: n++
      n++;
      //4.3 将新值放元素内容中
      //本例中: 将新值 n，拼接上原来完整的字符串，再修改回按钮内容中
      $this.html(`click me(${n})`)
    })
```

```
    //自动翻译为三个 onclick:
    //$btns[0].onclick=function(){ ... }
    //$btns[1].onclick=function(){ ... }
    //$btns[2].onclick=function(){ ... }
  </script>
 </body>
 </html>
```

运行结果:

四. 查找元素: jQuery 支持用所有 CSS3 选择器查找，并添加了少量 jQuery 独有选择器

1. 回顾: CSS 中的子元素过滤选择器:

(1). 什么是: 根据元素在其父元素下的相对位置选择元素

(2). 包括:

a. :first-child 选择所有作为其所在父元素下的第一个孩子的元素

b. :last-child 选择所有作为其所在父元素下的最后一个孩子的元素

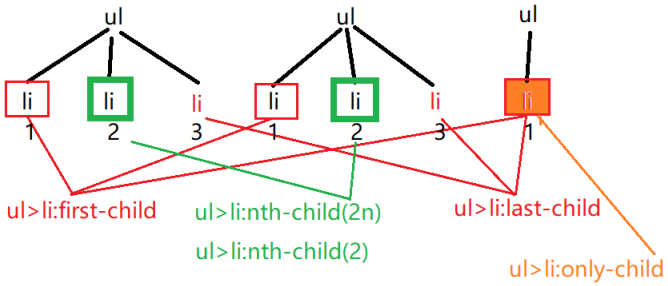
c. :nth-child(i) 选择所有作为其所在父元素下第 i 个子孩子的元素

css 中的 i 都是从 1 开始!

d. :only-child 选择所有作为其所在父元素下的唯一一个孩子的元素

(3). 示例: 使用子元素过滤，选择指定的元素

```
<ul>
  <li>child1-basic0</li>
  <li>child2-basic1</li>
  <li>child3-basic2</li>
</ul>
<ul>
  <li>child1-basic3</li>
  <li>child2-basic4</li>
  <li>child3-basic5</li>
</ul>
<ul>
  <li>child1-basic6</li>
</ul>
```



5_child filter.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <title></title>
</head>
<body>
  <h3>子元素过滤选择器.</h3>
  <ul>
```

```
    <li>child1-basic0</li>
    <li>child2-basic1</li>
    <li>child3-basic2</li>
  </ul>
  <ul>
    <li>child1-basic3</li>
    <li>child2-basic4</li>
    <li>child3-basic5</li>
  </ul>
  <ul>
    <li>child1-basic6</li>
  </ul>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //查找每个 ul 中第一个 li,设置他们的边框为红色
    $("ul>li:first-child")
    .css("border","1px solid red");
    //查找每个 ul 中最后一个 li,设置他们的字体为红色
    $("ul>li:last-child")
    .css("color","red");
    //查找每个 ul 中处于偶数位置的，设置他们的阴影为绿色
    $("ul>li:nth-child(2n)")
    .css("box-shadow","0 0 5px green")
    //查找每个 ul 中第 2 个 li，设置其 padding
    $("ul>li:nth-child(2)")
    .css("padding","10px 0")
    //查找作为 ul 下唯一子元素的 li，设置其背景色为橙色
    $("ul>li:only-child")
    .css("background-color","orange")
  </script>
</body>
</html>
```

运行结果:

子元素过滤选择器.

- child1-basic0
- child2-basic1
- child3-basic2
- child1-basic3
- child2-basic4
- child3-basic5
- child1-basic6

2. 基本过滤选择器: （jQuery 独有！）

补: 这些 jQuery 新增的独有选择器其实底层使用 js 程序模拟的！所以下标从 0 开始！且效率不如原生的 css 选择器高！

(1). 什么是: 先将所有符合条件的元素放在一个大的集合中，统一编号。按元素在大的结果集合中的下标位置选择对应的元素。

(2). 强调: 和元素在其父元素下的相对位置毫无关系！

(3). 何时: 今后希望按照元素的大排名位置选择元素，而不关心元素在其父元素下的相对位置时，都选择基本过滤选择器。

(4). 包括:

a. :first 所有符合条件的元素中，第一个元素

b. :last 所有符合条件的元素中，最后一个元素

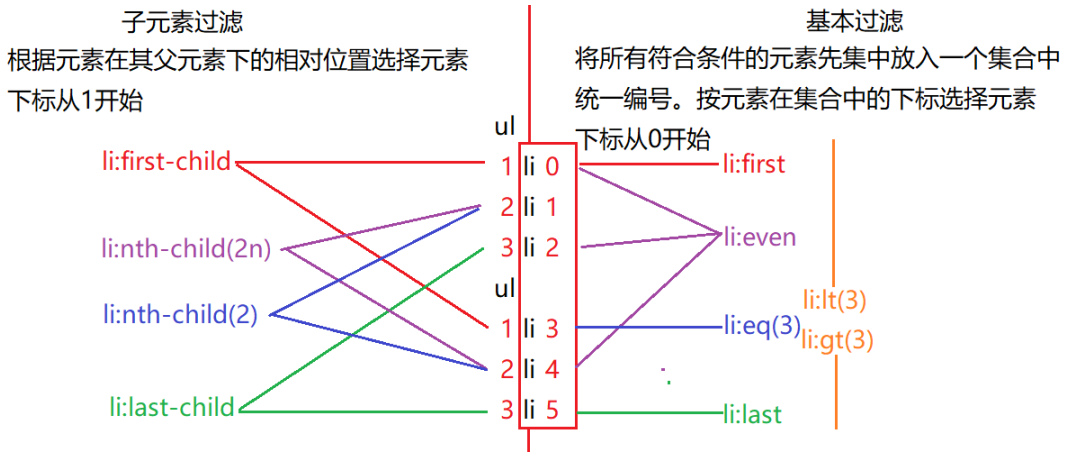
c. :eq(i) equal 所有符合条件的元素中，下标等于 i 位置的一个元素(i 从 0 开始)

d. :lt(i) less than 所有符合条件的元素中，下标<i 位置的元素(i 从 0 开始)

e. :gt(i) greater than 所有符合条件的元素中，下标>i 位置的元素(i 从 0 开始)

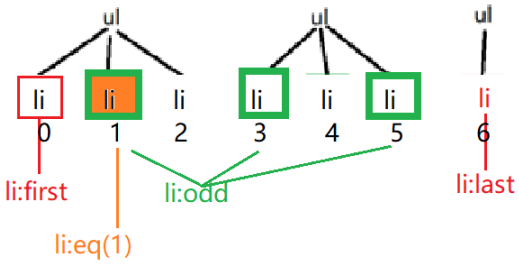
f. :even 所有符合条件的元素中，下标为偶数位置的元素

g. :odd 所有符合条件的元素中，下标为奇数位置的元素



(5). 示例: 使用基本过滤选择器，选择指定的元素

```
<ul>
  <li>child1-basic0</li>
  <li>child2-basic1</li>
  <li>child3-basic2</li>
</ul>
<ul>
  <li>child1-basic3</li>
  <li>child2-basic4</li>
  <li>child3-basic5</li>
</ul>
<ul>
  <li>child1-basic6</li>
</ul>
```



6_basic filter.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <title></title>
</head>
<body>
  <h3>基本过滤选择器.</h3>
  <ul>
    <li>child1-basic0</li>
    <li>child2-basic1</li>
    <li>child3-basic2</li>
  </ul>
  <ul>
    <li>child1-basic3</li>
    <li>child2-basic4</li>
    <li>child3-basic5</li>
  </ul>
  <ul>
    <li>child1-basic6</li>
  </ul>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //查找所有 li 中第一个 li，加红边框
    $("li:first").css("border","1px solid red");
    //查找所有 li 中最后一个 li，字体变为红色
    $("li:last").css("color","red");
    //查找处于偶数位置的 li，加绿色阴影
```


```
    $("li:odd").css("box-shadow","0 0 5px green")
    //查找第 2 个 li,背景变为橙色
    $("li:eq(1)").css("background-color","orange")
  </script>
</body>
</html>
```

运行结果:

基本过滤选择器.

- child1-basic0
- child2-basic1
- child3-basic2

- child1-basic3
- child2-basic4
- child3-basic5

- child1-basic6

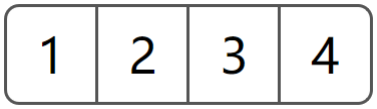
(6). 示例: 分别使用 css 和 jquery 实现按钮组效果

7_basic filter.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>...</title>
  <style>
    ul{list-style:none}
    ul>li{
      padding:5px 10px;
      border:1px solid #555;
      float:left;
    }
    /*第一个 li, 左上, 左下圆角*/
    ul>li:first-child{
```

```
      border-radius:5px 0 0 5px;
    }
    /*最后一个 li，右上，右下圆角*/
    ul>li:last-child{
      border-radius:0 5px 5px 0;
    }
    /*去掉每个 li 右边相邻的 li 的左边框*/
    ul>li+li{
      border-left:0;
    }
  </style>
</head>

<body>
  <h1>实现按钮组效果</h1>
  <ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
  </ul>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    // //第一个 li 左上左下圆角
    // $("ul>li:first")
    // .css("border-radius","5px 0 0 5px");
    // //最后一个 li 右上右下圆角
    // $("ul>li:last")
    // .css("border-radius","0 5px 5px 0");
    // //第一个 li 之后所有 li 去掉左边框
    // $("ul>li:gt(0)").css("border-left",0)
  </script>
</body>

</html>
```

运行结果:



3. 内容过滤: 根据元素内容的不同查找不同的元素
   (1). :contains(关键词)　根据元素开始标签到结束标签之间的内容中的关键词查找元素
   (2). :has(选择器)　根据元素内的子元素具有的特征查找父元素
   (3). :parent　选择所有内容不为空的元素

(4). :empty　选择所有内容为空的元素

说明:empty 只能匹配开始标签和结束标签之间没有任何字符的元素

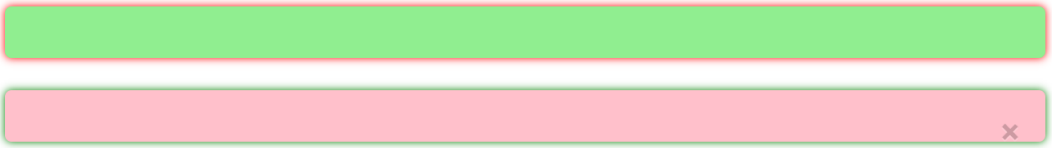
(5). 示例: 使用内容过滤选择器选择符合条件的元素

10_content filter.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <link rel="stylesheet" href="css/bootstrap.css"/>
  <style>
  </style>
</head>
<body>
<div class="container">
<h1>jQuery 中的选择器——内容过滤选择器</h1>

  <button>提交订单</button>
  <button>Submit 注册信息</button>
  <button>马上提交</button>
  <button>清空重填</button>

  <hr/>
  <div class="alert" id="alert1"></div>
  <div class="alert" id="alert2">
    <span class="close">×</span>
  </div>

</div>
<script src="js/jquery-1.11.3.js"></script>
<script>
  //选择包含"提交"二字的按钮，变为绿色按钮
  $("button:contains(提交)")
  .css("background-color","lightGreen")
  //选中包含.close 按钮的.alert 元素，让它们变为红色的警告框
  $(".alert:has(.close)")
  .css("background-color","pink");
  //选中不包含.close 按钮的.alert 元素，让它们变为绿色的警告框
  $(".alert:not(:has(.close))")
  .css("background-color","lightGreen");

  //空的.alert，加红阴影
  $(".alert:empty")
  .css("box-shadow","0 0 5px red")
```

```
  //非空的.alert，加绿阴影
  $(".alert:parent")
  .css("box-shadow","0 0 5px green")
</script>
</body>
</html>
```

运行结果:



4. 可见性过滤: 根据元素是否可见来选择元素
   (1). :visible　选择所有可见的元素
   (2). :hidden　选择所有不可见的元素
   强调: :hidden 只能选择两种隐藏的元素: display:none 和 input type="hidden"
   ~~不能选择 visibility:hidden 和 opacity:0~~
5. 表单元素过滤选择器: 专门选择表单元素的选择器
   (1). :input　选择所有表单元素(input　textarea　select　button)
   vs　input　CSS 元素选择器，仅选择 input 元素，不选择其他元素
   (2). 其实每种 input type 都对应着一个专门的选择器:
   a. :text　专门选择 input type="text"的文本框
   b. :password　专门选择 input type="password"的密码框
   c. :radio　专门选择 input type="radio"的单选按钮
   d. :checkbox　专门选择 input type="checkbox"的复选框
   ...
   (3). 示例: 点同意，控制其他表单元素的启用和禁用状态
   13_form state selector.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>.....</title>
  <meta charset="utf-8"/>
  <style>
  </style>
 </head>
 <body>
  <form>
    用户名:<input disabled></br>
    密码:<input type="password" disabled></br>
    <input type="checkbox">我同意本站的使用条款<br>
    <button type="submit" disabled>提交注册信息</button>
```

```
  </form>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //DOM 4步
    //1. 查找触发事件的元素
    //本例中：用户点checkbox元素触发变化
    var $chb=$(":checkbox");
    //console.log($chb);
    //2. 绑定事件处理函数
    //本例中：单击checkbox触发变化
    $chb.click(function(){
      //3. 查找要修改的元素
      //本例中：点checkbox是为了修改除checkbox以外其他表单元素的状态
      var $others=$(":input:not(:checkbox)");
      //console.log($others);
      //4. 修改元素
      //如果当前checkbox是选中的（checked==true），说明同意，其他表单元素
启用(disabled=false)
      //否则如果当前checkbox是未选中（checked==false）的，说明不同意，其
他表单元素就禁用(disabled=true)
      //总结规律:
      //如果checked==true,则disabled=false
      //如果checked==false,则disabled=true
      //总结：disabled=!checked 唱反调
      //只要让其他元素的disabled属性=!当前checkbox的checked属性
      //语法错误:jquery中全是函数，不能用属性!
      //$others.disabled=!this.checked;
      //jq中:
      var $this=$(this);
      $others.prop("disabled",!$this.prop("checked"));
      //                        ←--------------------
      //                      取出当前checkbox的checked属性值
      //                       ←
      //                      取反
      //←----------------------
      //赋值给其他表单元素的disabled属性
    })
  </script>
 </body>
</html>
```

运行结果：

用户名:
密码:
我同意本站的使用条款
提交注册信息

用户名:
密码:
我同意本站的使用条款
提交注册信息

五. 修改: 3 种:

修改中的所有函数，都是一个函数两用!

1. 内容: 3 种:

(1). 原始 HTML 内容: $元素.html("新 HTML 内容") 被翻译为-> .innerHTML

(2). 纯文本内容: $元素.text("纯文本内容") 被翻译为-> .textContent

(3). 表单元素的值: $元素.val("新值") 被翻译为-> .value

(4). 示例: 使用修改内容方式，实现表单验证

14_html_val.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
 </head>
 <body>
  <h1>操作元素的内容和值</h1>
  <form action="">
    用户名:<input name="uname">
          <span></span><br>
    密码:<input type="password" name="upwd">
          <span></span><br>
    <input type="submit" value="提交注册信息">
  </form>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //正确时，使用图片:"<img src='img/ok.png'>"
    //姓名错误时: "<img src='img/err.png'>用户名必须介于 3~9 位之间!"
    //密码错误时: "<img src='img/err.png'>密码必须介于 6~8 位之间!"
    //DOM 4 步
    //1. 查找触发事件的元素
    //本例中: 文本框失去焦点时，触发验证和变化
```

```
    $(":text")
    //2. 绑定事件处理函数
    //本例中: 失去焦点触发变化
    .blur(function(){
      //为了防止再使用原生 DOM 属性和方法，一开始就把 this 包装一下
      var $this=$(this);
      //3. 查找要修改的元素
      //本例中: 修改当前文本框旁边的 span 元素
      var $span=$this.next();//.nextElementSibling 找的是下一个元素对象！所以返回结果也是一个包含找到的元素对象的 jquery 对象
      //4. 修改元素
      //获得当前文本框的内容
      var value=$this.val();//.value -> 字符串
      //如果验证当前文本框的内容通过
      if(value.length>=3&&value.length<=9){
        //则修改 span 的内容为<img src='img/ok.png'>
        $span.html(`<img src='img/ok.png'>`);
      }else{//否则如果验证文本框的内容不通过
        //则修改 span 的内容为<img src='img/err.png'>用户名必须介于 3~9 位之间!
        $span.html(`<img src='img/err.png'>用户名必须介于 3~9 位之间!`)
      }
    })
  </script>
 </body>
</html>
```

运行结果:

用户名: 用户名必须介于3~9位之间!
密码:
提交注册信息

用户名: dingding
密码:
提交注册信息

2. 属性: 3 种:
   (1). 字符串类型的 HTML 标准属性:（待续...）
   (2). bool 类型的 HTML 标准属性:
   a. DOM 中只有一种操作方式: 元素.属性名
   b. 问题: jquery 是函数库，不能用属性
   c. jq 中也只有一种操作方式: 提供了一个专门的函数来代替.操作:
   $元素.prop("属性名", bool 值)
   property 的缩写，属性，意为获取或修改一个属性的值。

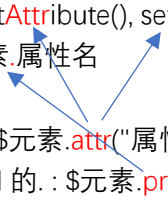
(3). 自定义扩展属性:

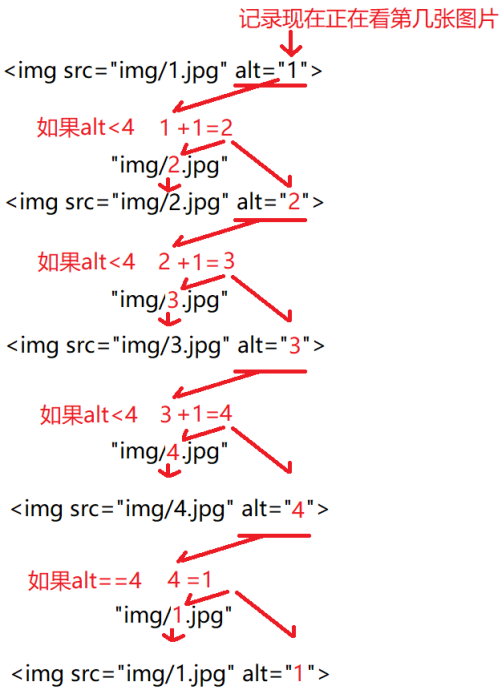
3. 样式:
   (1). 获取或修改内联样式:
   a. DOM 中: 元素.style.css 属性="属性值"
      自动翻译为↑
   b. jq 中: $元素.css("css 属性名", "属性值")

#jQueryday02:

正课:
1. 修改元素
2. 按节点间关系查找
3. 添加删除替换克隆
4. 事件绑定

一. 修改元素: 3 种
强调: 以下所有函数都是一个函数两用
1. 内容: (已讲过)
2. 属性: 3 种:
   (1). 字符串类型的 HTML 标准属性
   a. DOM 中: 2 种:
      1). 旧核心 DOM: getAttribute(), setAttribute(), hasAttribute(), removeAttribute()
      2). HTML DOM: 元素.属性名
   b. jq 中: 2 种:
      1). 代替核心 DOM: $元素.attr("属性名","新属性值")
      2). 代替 HTML DOM 的. : $元素.prop("属性名", "新属性值")
   c. 示例: 点图片, 切换下一张





2_attr.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
 </head>
 <body>
  <h1>操作元素的属性</h1>
  <img src="img/1.jpg" alt="1">
  <script src="js/jquery-1.11.3.js"></script>
  <script>
  //DOM 4步
  //1. 查找触发事件的元素
  //本例中：用户点 img 元素触发变化？
  $("img")
  //2. 绑定事件处理函数
  //本例中：单击切换下一张
  .click(function(){
    //3. 查找要修改的元素
    //本例中：就是要修改自己
    var $this=$(this);
    //4. 修改元素
    //本例中：切换 img 的 src 属性为下一张图片
    //4.1 先取出当前 img 的 alt 属性值，转为整数
    var alt=parseInt($this.attr("alt"));
```

```
                        //.prop("alt")
    //4.2 如果 alt<4，就 alt+1，否则就 alt=1
    // if(alt<4){
    //   alt++
    // }else{
    //   alt=1
    // }
    //一个条件，两件事，二选一执行，应该用三目
    alt<4?(alt++):(alt=1);//复习第一阶段三目运算
    //4.3 将新 alt 值放回当前 img 的 alt 属性中，并且将 alt 值拼接成新的 src，
保存到当前 img 的 src 属性上
    //$this.attr("alt",alt);
       //.prop("alt",alt);
    //$this.attr("src",`img/${alt}.jpg`);
       //.prop("alt",alt);
    // $this.attr({//.prop({
    //   alt:alt,
    //   src:`img/${alt}.jpg`
    // });
    $this.attr({alt,src:`img/${alt}.jpg`});//复习 ES6
       //.prop
  })
  </script>
 </body>
</html>
```

运行结果：

操作元素的属性

(2). bool 类型的 HTML 标准属性:

a. DOM 中: 只有一种方式: 元素.属性名

代替

b. jq 中: 也只有一种方式: $元素.prop("属性名", bool 值)

(3). 自定义扩展属性:

a. DOM 中: 不考虑 HTML5，则只能用核心 DOM: getAttribute()和 setAttribute()

代替

b. jq 中: 也只能用$元素.attr("data-自定义属性名","新属性值")

总结: .attr()代替了 getAttribute()和 setAttribute()

.prop()代替了.操作

c. 示例: 点击小图片，切换大图片

3_attr2.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
    body {
      text-align: center;
    }
  </style>
</head>
<body>

<img src="img/1.jpg" data-target="img/1-l.jpg" class="my-small">
<img src="img/2.jpg" data-target="img/2-l.jpg" class="my-small">
<img src="img/3.jpg" data-target="img/3-l.jpg" class="my-small">
<img src="img/4.jpg" data-target="img/4-l.jpg" class="my-small">
<hr/>
<img src="img/1-l.jpg" class="my-big">


<script src="js/jquery-1.11.3.js"></script>
<script>
//DOM 4 步
//1. 查找触发事件的元素
//本例中: 点所有 class 为 my-small 的小图片，触发变化
$(".my-small")
//2. 绑定事件处理函数
//本例中: 单击小图片触发变化
.click(function(){
  //3. 查找要修改的元素
  //本例中: 要修改 class 为 my-big 的大图片
  var $big=$(".my-big");
  //4. 修改元素
  //本例中:
  //4.1 取出当前小图片的 data-target 属性值中保存的对应大图片路径
  var src=$(this).attr("data-target");
```

```
                // .prop() ×
  //4.2 将 src 设置到大图片的 src 属性上
  // $big.attr({src:src});
  $big.attr({src}); //复习 ES6
    //.prop({src}); √
})
</script>
</body>
</html>
```

运行结果:

3. 样式:

(1). DOM 中:

a. 修改样式: 元素对象.style.css 属性="新值"

b. 获取样式: var style=getComputedStyle(元素对象)

囊括

(2). jq 中: 无论获取还是修改样式都用: $元素.css("css 属性名","新属性值")

a. 当.css()给了新属性值，则执行修改 css 属性的操作，相当于.style

b. 当.css()没给新属性值，则执行获取 css 属性值的操作，相当于 getComputedStyle()

(3). 批量修改一个元素的多个 css 属性，最好用 class 方式修改:

a. DOM 中: 元素.className="新 class"

b. 问题: 只能批量获取或替换整个 class 字符串。如果只希望操作一个元素上众多 class 中的一个 class，就极其不方便!

c. jq 中: 4 个函数:

1). $元素.addClass("class 名")

2). $元素.removeClass("class 名")

3). $元素.hasClass("class 名")

4). $元素.toggleClass("class 名") 原理: 在有和没有一个 class 名之间来回切换

切换

a. 如果当前元素上有这个 class，就移除 class

b. 否则如果当前元素上没有这个 class 名，就添加 class

d. 问题：能不能用 toggleClass()彻底代替 addClass()和 removeClass()?

e. 答：绝对不行！

因为：addClass()永远是添加 class 的意思，到什么时候都不会变

removeClass()永远是移除 class 的意思，到什么时候都不会变

但是 toggleClass() 是一次添加，一次移除，下次又变成添加，没准！

只有在确定要在有和没有一个 class 之间来回切换时，才用 toggleClass()

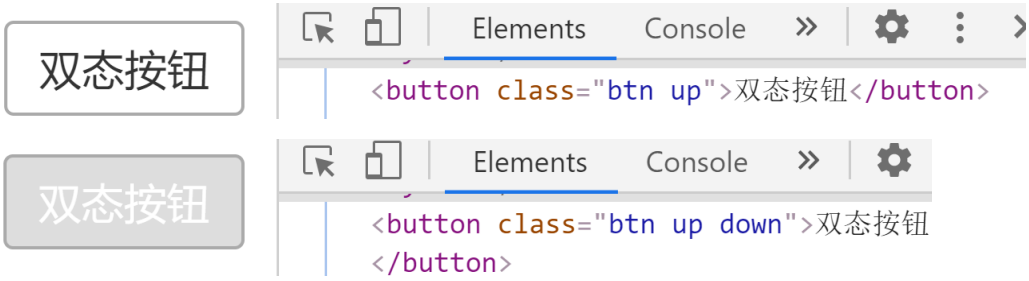
f. 示例：双态按钮

4_class.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
    .btn {
      padding: 5px 10px;
      border-radius: 3px;
      border: 1px solid #aaa;
      outline: none;
    }
    /* 默认抬起时的样子 */
    .up {
      background: #fff;
      color: #333;
    }
    /* 按下后的样子 */
    .down {
      background: #ddd;
      color: #fff;
    }
  </style>
</head>
<body>

  <button class="btn up">双态按钮</button>


<script src="js/jquery-1.11.3.js"></script>
<script>
//DOM 4 步
//1. 查找触发事件的元素
//本例中：单击按钮触发变化
$("button")
```

```
//2. 绑定事件处理函数
.click(function(){
  //3. 查找要修改的元素
  //本例中: 就是要修改自己:
  //var $this=$(this);
  //4. 修改元素
  //如果按钮自己有 class down，就移除 class down
  // if($this.hasClass("down")){
  //   $this.removeClass("down")
  // }else{//否则如果按钮自己没有 class down，就添加 class down
  //   $this.addClass("down")
  // }
  $(this).toggleClass("down");
})
</script>
</body>
</html>
```

运行结果:



同时修改多个属性值: 修改相关函数的简写:

```
$元素.attr 或 prop 或 css({
    属性名:"属性值",
        ... : ...
})
```

二. 按节点间关系查找: jq 中: 2 大类关系，8 个函数

1. 父子关系: 4 个属性 —— jq 中变为 3 个函数(parent(), children(), find())

(1). 获取元素的父元素: 元素.parentNode 或 parentElement

jq 中简写: $元素.parent()

(2). 获取元素的所有直接子元素: 元素.children

a. jq 中简写: $元素.children("选择器") ——增强: 可以用选择器只获取我想要的个别子元素。

b. 问题: children()只能查找直接子元素，无法在所有后代中查找

c. jq 中新增了一个函数: $元素.find("选择器")

意为: 在所有后代中查找符合选择器要求的元素。

(3). 获取元素的第一个直接子元素: 元素.firstElementChild

a. 没有提供专门的函数与之对应

b. 但是可用 children()变通实现: $元素.children(":first-child")

(4). 获取元素的最后一个直接子元素: 元素.lastElementChild

a. 没有提供专门的函数与之对应

b. 但是可用 children()变通实现: $元素.children(":last-child")

2. 兄弟关系: 2 个属性 —— 5 个函数(prev() prevAll() next() nextAll() siblings())

(1). 前一个相邻的兄弟元素: 元素.previousElementSibling

a. jq 中: $元素.prev()

b. 问题: prev()只能找相邻的前一个兄弟元素，无法获得之前所有兄弟元素

c. jq 中新增: $元素.prevAll("选择器")

如果提供了选择器参数，则只获取之前所有兄弟中符合选择器要求的兄弟

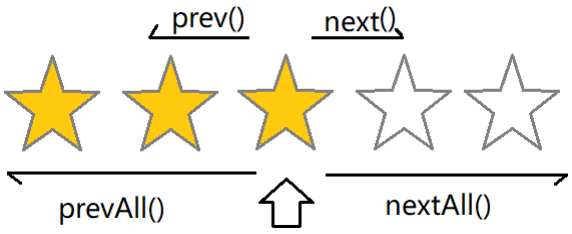
(2). 后一个相邻的兄弟元素: 元素.nextElementSibling

a. jq 中: $元素.next()

b. 问题: next()只能找相邻的下一个兄弟元素，无法获得之后所有兄弟元素

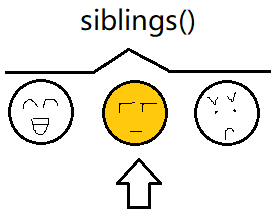
c. jq 中新增: $元素.nextAll("选择器")

如果提供了选择器参数，则只获取之后所有兄弟中符合选择器要求的兄弟



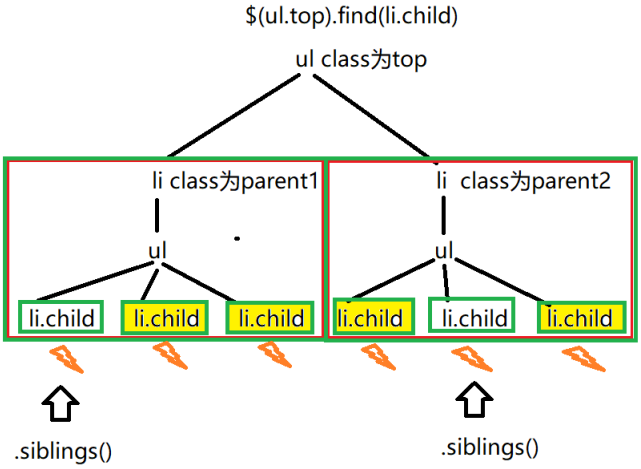
(3). jq 中新增: $元素.siblings("选择器") 选择除当前元素自己之外的其余所有兄弟元素。

如果提供了选择器参数，则只获取其余兄弟中符合选择器要求的兄弟



3. 示例: 使用节点间关系查找，查找指定元素

```
<ul class="top">
    <li class="parent1">parent1
        <ul>
            <li class="child">child1</li>
            <li class="child">child2</li>
            <li class="child">child3</li>
        </ul>
    </li>
    <li class="parent2">parent2
        <ul>
            <li class="child">child1</li>
            <li class="child">child2</li>
            <li class="child">child3</li>
        </ul>
    </li>
</ul>
```



6_traverse.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>...</title>
  <script>
  </script>
</head>

<body>
  <!--    ul.top>(li.parent>ul>li.child*3)*2 -->
  <ul class="top">
    <li class="parent1">parent1
      <ul>
        <li class="child">child1</li>
        <li class="child">child2</li>
        <li class="child">child3</li>
      </ul>
    </li>
    <li class="parent2">parent2
      <ul>
        <li class="child">child1</li>
        <li class="child">child2</li>
        <li class="child">child3</li>
      </ul>
    </li>
  </ul>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //修改 class 为 top 的 ul 的所有直接子元素
    $("ul.top").children()
               .css("border","1px solid red");
    //修改 class 为 top 的 ul 的所有后代 li
    $("ul.top").find("li")
               .css("box-shadow","0 0 5px green");
    //为 class 为 child 的 li 绑定单击事件
    $("ul.top").find("li.child").click(function(){
      //alert("疼!")
      //选择当前元素的下一个元素/前一个元素/之前所有/之后所有/除自己之外所
有
      $(this)
      //.next()
      //.prev()
```

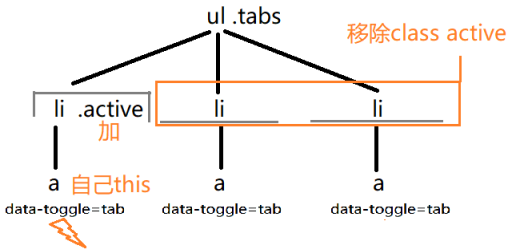
```
      //.prevAll()
      //.nextAll()
      .siblings()
      .css("background-color","yellow")
    })

  </script>
</body>

</html>
```

运行结果:

- parent1
  - child1
  - child2
  - child3
- parent2
  - child1
  - child2
  - child3

4. 示例: 标签页

```
<ul class="tabs">
 <li class="active">
   <a data-toggle="tab" href="#">十元套餐</a>
 </li>
 <li>
   <a data-toggle="tab" href="#">二十元套餐</a>
 </li>
 <li>
   <a data-toggle="tab" href="#">三十元套餐</a>
 </li>
</ul>
```



7_tabs.html

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <style>
    .tabs{ list-style:none; padding:0 }
    .tabs a{
      text-decoration:none;
      color:#000;
      padding:6px 12px;
      display:inline-block;
    }
```

```
    .tabs>li{
      float:left;
      border-bottom:1px solid #000;
    }
    .tabs>.active{
      border:1px solid #000;
      border-bottom:0;
    }
  </style>
 </head>
 <body>
    <h1>使用属性选择器实现标签页头的切换</h1>
    <ul class="tabs">
      <li class="active"><a data-toggle="tab" href="#">十元套餐
</a></li>
      <li><a data-toggle="tab" href="#">二十元套餐</a></li>
      <li><a data-toggle="tab" href="#">三十元套餐</a></li>
    </ul>
    <script src="js/jquery-1.11.3.js"></script>
    <script>
      //DOM 4 步
      //1. 查找触发事件的元素
      //本例中：用户点击 class 为 tabs 的 li 下的 a 触发变化
      $("ul.tabs>li>a")
      //2. 绑定事件处理函数
      //本例中：单击触发变化/换成鼠标移入变化
      .click(function(){ //mouseover(function(){
        //3. 查找要修改的元素
        //本例中：既改自己，又改别人
        //4. 修改元素
        //先给自己的爹加 active class
        $(this).parent()
        //当前 a  当前 li
              .addClass("active") //本没有返回值
        //默认返回.前的当前 li 的 jquery 对象
        //再去掉自己的爹的所有兄弟的 active class
        //下一句刚好再次需要用到自己的爹
        //于是就可以链式操作
              .siblings()
        //    除当前 li 之外的其它所有兄弟 li
              .removeClass("active")
      })
```

三. 添加删除替换克隆:

1. 添加/替换:

(1). 回顾 DOM 中:

a. 只添加一个元素: 3 步:

1). 创建新元素: var 新元素=document.createElement("标签名")

2). 设置关键属性: 元素.属性名=属性值

3). 将新元素添加到 DOM 树: 3 种:

i. 末尾追加: 父元素.appendChild(新元素)

ii. 在现有元素前插入: 父元素.insertBefore(新元素, 现有元素)

iii. 替换现有元素: 父元素.replaceChild(新元素, 现有元素)

b. 同时添加多个平级元素:

1). 创建文档片段:

var frag=document.createDocumentFragment();

2). 将多个新子元素添加到文档片段中
frag.appendChild(新子元素)
3). 将文档片段一次性添加到 DOM 树
父元素.appendChild(frag);

(2). jq 中: 2 步:

a. 创建新元素: var $新元素=$(`HTML 片段`)

b. 也需要将新元素添加到 DOM 树: 5 种方式，10 个函数

1). 末尾追加: $父元素.append($新元素) 代替 appendChild()

兄弟函数: $新元素.appendTo($父元素)
上下两个函数.前的主语对象不同！导致返回值不同！便于各种情况链式操作
如果下一步想继续对父元素执行操作，应该选择以父元素开头的 append
如果下一步想继续对新元素执行操作，应该选择以新元素开头的 appendTo

2). 开头插入: 新增: $父元素.prepend($新元素)

兄弟函数: $新元素.prependTo($父元素)
如果下一步想继续对父元素执行操作，应该选择以父元素开头的 prepend
如果下一步想继续对新元素执行操作，应该选择以新元素开头的 prependTo

3). 插入到一个现有元素之前: $现有元素.before($新元素) 代替 insertBefore()

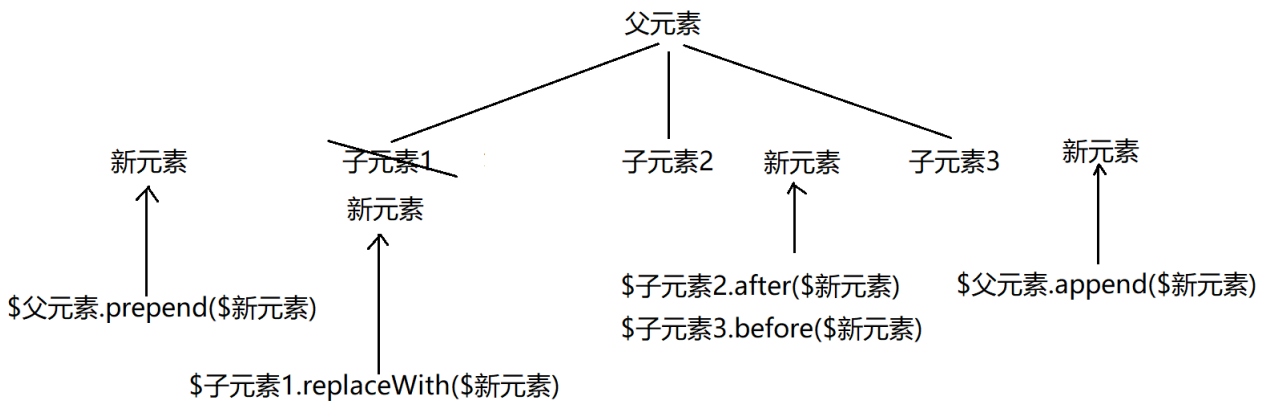
兄弟函数: $新元素.insertBefore($现有元素)
如果下一步想继续对现有元素执行操作，应该选择以现有元素开头的 before
如果下一步想继续对新元素执行操作，应该选择以新元素开头的 insertBefore

4). 插入到一个现有元素之后: $现有元素.after($新元素)

兄弟函数: $新元素.insertAfter($现有元素)
如果下一步想继续对现有元素执行操作，应该选择以现有元素开头的 after
如果下一步想继续对新元素执行操作，应该选择以新元素开头的 insertAfter

5). 替换现有元素: $现有元素.replaceWith($新元素)

兄弟函数: $新元素.replaceAll($现有元素)
如果下一步想继续对现有元素执行操作，应该选择以现有元素开头的 replaceWith
如果下一步想继续对新元素执行操作，应该选择以新元素开头的 replaceAll



c. 强调: jquery 所有将元素添加到 DOM 树的函数中，都内置了文档片段优化，不用再自己写。当添加多个平级元素时，jquery 简化版函数自动使用文档片段优化添加操作。

2. 删除元素: $任意元素.remove();

3. 示例: 点按钮添加方块，点 x 删除方块

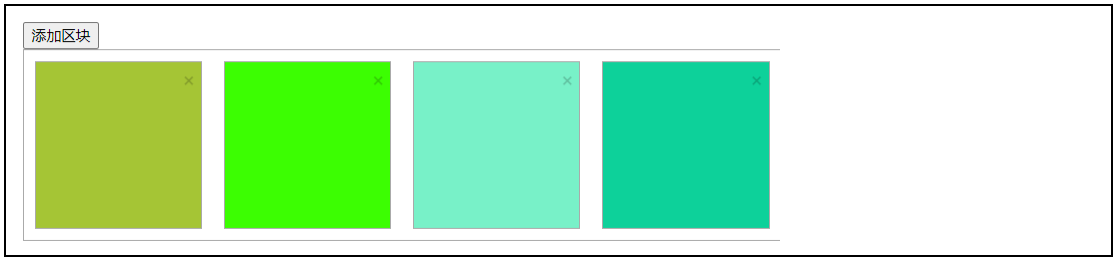
9_append.html

```
<!DOCTYPE html>
<html>

<head>
  <title> new document </title>
  <meta charset="utf-8">
  <style>
    .container {
      border: 1px solid #aaa;
      overflow: hidden;
    }

    .block {
      float: left;
      margin: 10px;
      border: 1px solid #aaa;
      background: #faa;
      width: 150px;
      height: 150px;
    }

    .block:hover {
      box-shadow: 0 5px 6px #000;
    }

    .close {
      float: right;
      padding: 5px;
      font-weight: bold;
      opacity: .2;
      cursor: pointer;
    }

    .close:hover {
      opacity: .5;
    }
  </style>
</head>
```

```
<body>
  <h1>添加/删除节点</h1>
  <button id="add-block">添加区块</button>

  <div class="container">
    <!-- <div class="block">
      <span class="close">×</span>
    </div> -->
  </div>

  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //DOM 4步
    //1. 查找触发事件的元素
    //本例中：点按钮触发单击
    $("#add-block")
    //2. 绑定事件处理函数
    .click(function(){
      //3. 查找要修改的元素
      //本例中：新添加的元素都放在class为container的div下
      // var $div=$(".container");
      // //4. 修改元素
      // //4.1新创建一个方块元素
      // var $block=$(`<div class="block">
      //  <span class="close">×</span>
      // </div>`);
      // //4.2设置新方块元素的背景色为一种随机色
      // $block.css(
      //  "background-color",
      //  `rgb(${
      //     parseInt(Math.random()*256)
      //   },${
      //     parseInt(Math.random()*256)
      //   },${
      //     parseInt(Math.random()*256)
      //   })`
      // );
      // //4.3将新方块插入到父元素div的开头
      // $div.prepend($block);

      $(//1. 创建新元素
        `<div class="block">
          <span class="close">×</span>
```

```
        </div>`
      )
      .css(//2. 为新元素设置随机背景色
        "background-color",
        `rgb(${
            parseInt(Math.random()*256)
          },${
            parseInt(Math.random()*256)
          },${
            parseInt(Math.random()*256)
          })`
      )//返回 新元素
      //3. 将新元素插入到父元素开头
      //.prependTo($(".container"));
      //福利：查找父元素时，根本不用我们自己找，我们只要提供选择器，prependTo 这一类函数自带查找功能！
      .prependTo(".container")
      //因为 prependTo.前就是新元素，所以 prependTo 在操作完之后，返回的依然是新元素
      //刚好下一步是查找新元素下的 span,刚好需要新元素
      //4. 查找新元素中 class 为 close 的 span 元素，绑定单击事件
      .children("span.close")
      .click(function(){
        //删除当前 span 的父元素
        $(this).parent().remove();
        //span    div
      })
    });

    //不要写在事件处理函数外部，因为页面刚加载时，一个方块和 span 元素都没有，这里肯定什么都找不到！
    //查找所有 class 为 close 的 span 元素,绑定单击事件
    console.log($("span.close"));
  </script>

</body>

</html>
```

运行结果:



4. 克隆: var $一模一样的新元素=$元素.clone() ——不是我们自己定义的，而是 jq 内置的，可直接使用!

5. 示例: 选飞机

10_replace_clone.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
    body{
      text-align: center;
    }
  </style>
</head>
<body>
<h1>替换节点</h1>

<div id="chosen">
  <img src="img/p0.png">
</div>
<hr/>
<div id="list">
  <img src="img/p3.png">
  <img src="img/p4.png">
  <img src="img/p5.png">
</div>

<script src="js/jquery-1.11.3.js"></script>
<script>
  //DOM 4步
  //1. 查找触发事件的元素
  //本例中: 单击id为list下的img元素触发变化
  $("#list>img")
  //2. 绑定事件处理函数
  .click(function(){
```

```
    //3. 查找要修改的元素
    //本例中: id 为 chosen 下的 img
    //var $chosen=$("#chosen>img");
    //4. 修改元素
    //先将当前选中的飞机克隆出一个副本，用副本代替上方的飞机
    //$(this).clone().replaceAll($chosen);
    $(this).clone().replaceAll("#chosen>img");
                                //自带查找功能
  })
</script>
</body>
</html>
```

运行结果:

四. 事件绑定:

1. DOM 中: 最灵活的事件绑定:
   (1). 添加事件监听对象: 元素.addEventListener("事件名", 处理函数)
   (2). 移除事件监听对象: 元素.removeEventListener("事件名", 原处理函数对象)
2. jq 中标准写法: 用.on()代替了.addEventListener(), 用.off()代替了.removeEventListener()
   (1). 添加事件监听对象: $元素.on("事件名", 处理函数)
   (2). 移除事件监听对象: $元素.off("事件名", 原处理函数对象)
3. 差别:
   (1). DOM 中: 完全相同的事件监听对象, 不允许重复添加
        所以在 DOM 中使用有名称的函数绑定事件, 则只能绑定一个
   (2). jq 中: 即使完全相同的事件监听对象, 照样可添加多个, 还能移除。
        所以在 jq 中即使用有名称的函数绑定事件, 也能绑定多个
4. 简写: jq 对常用的个别事件提供了简写方式:
   $元素.事件名(处理函数)
        ->翻译为 $元素.on("事件名", 处理函数)
            ->再被翻译为 DOM 中 元素.addEventListener("事件名", 处理函数)
            ~~而不等效于.on 事件名=function(){ ... }~~
   比如: $btn.click(function(){ ... })
            最底层 被翻译为: btn.addEventListener("click", function(){ ... })
            ~~而不是.onclick=function(){ ... }~~
   以下列表中的常用事件都可以使用简写方式, 而不用 on

常用事件名:
blur 失去焦点
change 下拉列表选中项改变
click 单击
dblclick 双击
focus 获得焦点
keydown 键盘按键按下
keyup 键盘按键抬起
mousedown 鼠标按键按下
mouseenter 鼠标进入(jq)
mouseleave 鼠标移出(jq)
mousemove 鼠标移动
mouseout 鼠标移出(dom)
mouseover 鼠标进入(dom)
mouseup 鼠标按键抬起
resize 窗口大小改变
scroll 网页滚动

5. 示例: 点发射按钮发射普通子弹，点获得跟踪导弹为发射按钮添加跟踪导弹，点失去跟踪导弹从发射按钮上移除跟踪导弹

12_bind_unbind.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
 </head>
 <body>
  <h1>事件绑定</h1>
  <button id="btn1">发射子弹</button>
  <button id="btn2">获得奖励</button>
  <button id="btn3">失去奖励</button>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //点 btn1，发射普通子弹
    //点 btn2，给 btn1 多增加一种跟踪导弹
    //点 btn3，从 btn1 上移除跟踪导弹
    //DOM 版:
    // var btn1=document.getElementById("btn1");
    // var btn2=document.getElementById("btn2");
    // var btn3=document.getElementById("btn3");
    // btn1.onclick=function(){
    //  console.log(`发射普通子弹...`)
    // }
    // function shoot2(){
```

```
    //  alert(`发射跟踪导弹=>=>=>`)
    // }
    // btn2.onclick=function(){ //无论点多少次 btn2，也只能添加一个跟踪导
弹
    //  btn1.addEventListener("click",shoot2)
    // }
    // btn3.onclick=function(){
    //  btn1.removeEventListener("click",shoot2);
    // }

    //jq 版
    // $("#btn1").on("click",function(){
    //  console.log(`发射普通子弹......`)
    // })
    // function shoot2(){
    //  alert(`发射跟踪导弹=>=>=>`)
    // }
    // $("#btn2").on("click",function(){
    //  $("#btn1").on("click",shoot2)
    // })
    // $("#btn3").on("click",function(){
    //  $("#btn1").off("click",shoot2)
    // })

    $("#btn1").click(function(){
      console.log(`发射普通子弹......`)
    })
    function shoot2(){
      alert(`发射跟踪导弹=>=>=>`)
    }
    $("#btn2").click(function(){
      $("#btn1").click(shoot2) //可以绑定多个处理函数
    })
    $("#btn3").click(function(){
      $("#btn1").off("click",shoot2)
    })
  </script>
 </body>
</html>
```

运行结果：

jq 中$共有 4 种用法:
1. $("选择器") 查找 DOM 元素，并保存进新创建的 jquery 对象中
2. $(DOM 元素对象) 无需查找，直接将已经获得的 DOM 元素，保存进新创建的 jquery 对象中
3. $(`HTML 片段`) 新创建一个 HTML 元素，保存进新创建的 jquery 对象中
4. 待续...

总结: jQuery API 三大特点:
1. 自带遍历:
2. 一个函数两用:
3. 所有本没有返回值的函数，默认都返回当前正在操作的.前的 jQuery 对象。
   好处: 链式操作!
   链式操作: 前一个函数的返回值，刚好是下一个函数所需的.前的主语，则两个函数可以用.连续调用!

#jQueryday03:

正课:
1. 事件
2. 动画
3. 类数组对象操作
4. 添加自定义函数
5. ***封装自定义插件

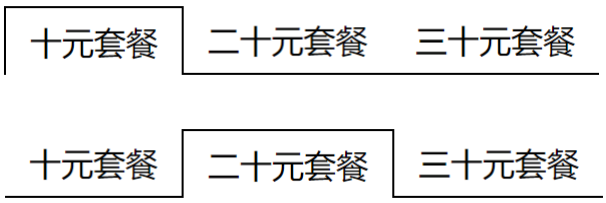
一. 事件:
1. 利用冒泡/事件委托:
   (1). DOM 中: 3 步——反而更通用
   a. 事件只绑定在父元素上一份即可
   b. 用 e.target 代替 this
   c. 判断 e.target 的特征是否是我们想要的
   d. 示例: 使用 jquery 和 DOM 事件委托 3 步优化标签页案例
   event/7_tabs.html

```
<!doctype html>
<html lang="en">
```

```
<head>
 <meta charset="UTF-8">
 <style>
   .tabs{ list-style:none; padding:0 }
   .tabs a{
     text-decoration:none;
     color:#000;
     padding:6px 12px;
     display:inline-block;
   }
   .tabs>li{
     float:left;
     border-bottom:1px solid #000;
   }
   .tabs>.active{
     border:1px solid #000;
     border-bottom:0;
   }
 </style>
</head>
<body>
   <h1>使用属性选择器实现标签页头的切换</h1>
   <ul class="tabs">
     <li class="active">
       <a data-toggle="tab" href="#">十元套餐</a>
     </li>
     <li>
       <a data-toggle="tab" href="#">二十元套餐</a>
     </li>
     <li>
       <a data-toggle="tab" href="#">三十元套餐</a>
     </li>
   </ul>
   <script src="js/jquery-1.11.3.js"></script>
   <script>
     //DOM 4步 事件委托 3步
     //1. 查找触发事件的元素
     //本例中: 因为多个子元素 a 都需要能单击，所以应该用事件委托优化，事件
应该只绑定在父元素上一份即可
     $("ul.tabs")
     //2. 绑定事件处理函数
     //本例中: 单击触发变化/换成鼠标移入变化
     .click(function(e){ //mouseover(function(e){
       //e.target 代替 this
```

```
        var $target=$(e.target);
        //只允许 data-toggle=tab 的元素触发事件
        if($target.attr("data-toggle")=="tab"){
          //3. 查找要修改的元素
          //本例中：既改自己，又改别人
          //4. 修改元素
          //先给自己的爹加 active class
          $target.parent()
          //当前 a   当前 li
                .addClass("active") //本没有返回值
          //默认返回.前的当前 li 的 jquery 对象
          //再去掉自己的爹的所有兄弟的 active class
          //下一句刚好再次需要用到自己的爹
          //于是就可以链式操作
                .siblings()
          //     除当前 li 之外的其它所有兄弟 li
                .removeClass("active")
        }
      })
    </script>
 </body>
</html>
```

运行结果：



(2). jq 中：简化 —— 仅限于 jQuery 中使用

a. 事件只绑定在父元素上一份即可，但是必须用$元素.on("事件名", ...)

b. this 又重新指向了 e.target，this 又可以使用了！

c. 不用自己写 if 判断，而是为 on 指定第二个选择器参数，由 on 自动检测当前元素是否符合要求：$元素.on("事件名", "选择器", 事件处理函数)

判断条件

1). 只有符合选择器要求的元素，才继续触发事件处理函数。

2). 如果当前元素不符合选择器要求，则不触发事件处理函数

d. 示例：使用 jquery 简化版事件委托优化标签页案例

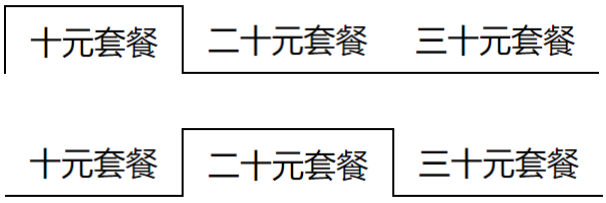
event/7_tabs_jq.html

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
```

```
  <style>
    .tabs{ list-style:none; padding:0 }
    .tabs a{
      text-decoration:none;
      color:#000;
      padding:6px 12px;
      display:inline-block;
    }
    .tabs>li{
      float:left;
      border-bottom:1px solid #000;
    }
    .tabs>.active{
      border:1px solid #000;
      border-bottom:0;
    }
  </style>
 </head>
 <body>
    <h1>使用属性选择器实现标签页头的切换</h1>
    <ul class="tabs">
      <li class="active">
        <a data-toggle="tab" href="#">十元套餐</a>
      </li>
      <li>
        <a data-toggle="tab" href="#">二十元套餐</a>
      </li>
      <li>
        <a data-toggle="tab" href="#">三十元套餐</a>
      </li>
    </ul>
    <script src="js/jquery-1.11.3.js"></script>
    <script>
      //DOM 4步 事件委托 3步
      //1. 查找触发事件的元素
      //本例中: 因为多个子元素 a 都需要能单击，所以应该用事件委托优化，事件
应该只绑定在父元素上一份即可
      $("ul.tabs")
      //2. 绑定事件处理函数
      //本例中: 单击触发变化/换成鼠标移入变化
      .on("click","[data-toggle=tab]",function(){
        //3. 查找要修改的元素
        //本例中: 既改自己，又改别人
        //4. 修改元素
```

```
        //先给自己的爹加 active class
        $(this).parent()
        //当前 a  当前 li
              .addClass("active") //本没有返回值
        //默认返回.前的当前 li 的 jquery 对象
        //再去掉自己的爹的所有兄弟的 active class
        //下一句刚好再次需要用到自己的爹
        //于是就可以链式操作
              .siblings()
        //    除当前 li 之外的其它所有兄弟 li
              .removeClass("active")
    })
  </script>
 </body>
</html>
```

运行结果：同上



2. 页面加载后自动执行:

(1). 问题: js 代码应该写在页面的什么位置才能保证正确执行?

答: 应该写在 HTML 内容之后，body 的结尾

(2). 问题: 按理说根据内容与行为分离的原则，为了便于维护，HTML 文件中就不应该包含 js 代码

解决: 所有 js 代码都应该放在 HTML 外的独立的 js 文件中保存，然后用<script>标签引入页面中使用

(3). 问题: 将来公司里什么人都有，有人愿意把 js 在 head 里引入，有人愿意把 js 在 body 结尾引入。如何保证无论 js 在开头还是在结尾引入都能正确执行?

解决: 利用页面加载完成事件

(4). 页面加载完成事件:

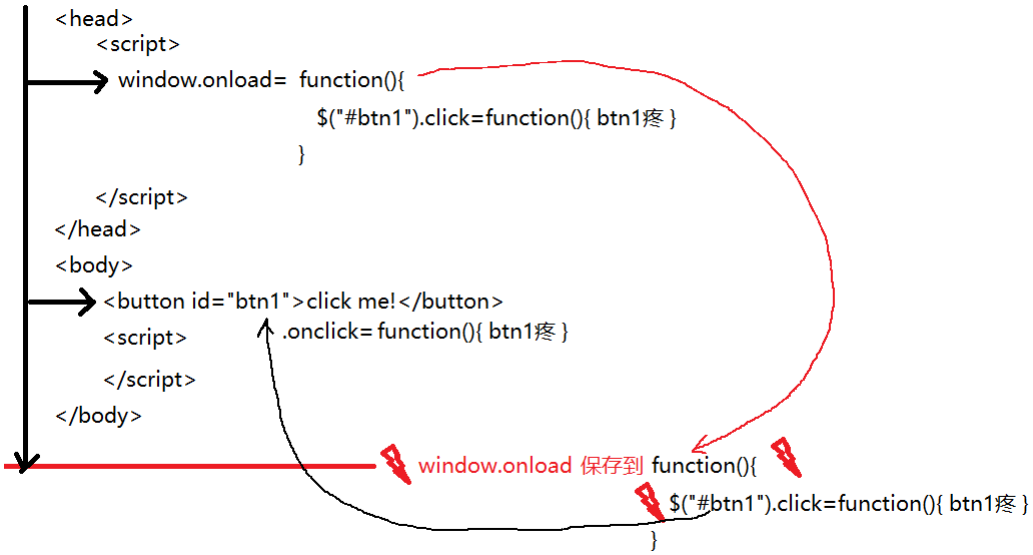
a. 其实在一个网页内容加载完成后，浏览器会触发一个事件。

b. 如果把我们的代码提前写在这个页面加载完成事件的处理函数中，提前保存在事件属性上。则无论任何情况下，我们的代码注定会随着事件在整个页面加载完成后才自动执行。

c. 如何: window.onload=function(){ … 注定只会在整个页面加载完成后才自动执行…}

d. 强调: 只是事件绑定代码，不会立刻执行! 只在事件发生时才自动执行!

(5). 总结: 将 js 代码放在 window.onload=function(){}中，无论 js 代码在页面开头引用，还是在页面结尾引用，注定只会在整个页面加载完成后才自动执行，且都能正确执行

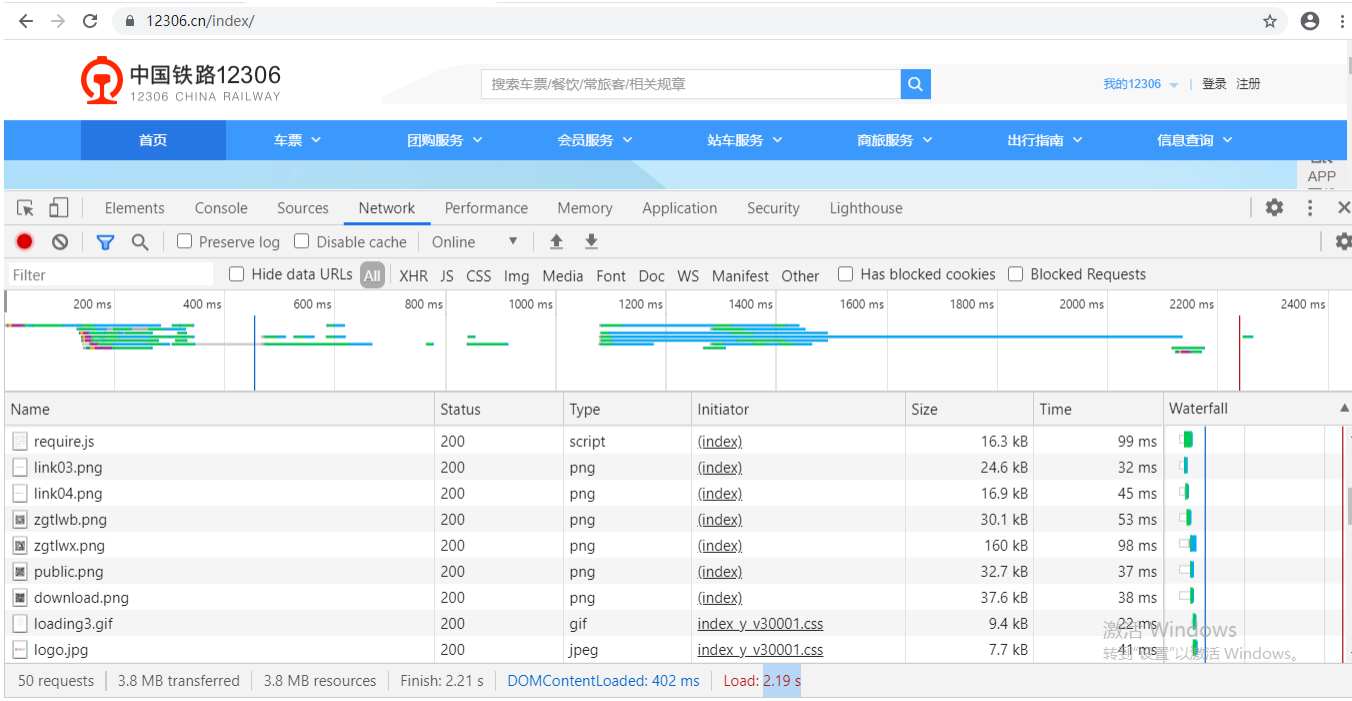


(6). 问题:不想让 DOM 和 jQuery 混搭

解决: 可将 DOM 的 window.onload 改为 jq 写法

window.onload=function(){} 改为 $(window).load(function(){ ... })

(7). 问题: window.onload 必须等待所有页面内容（HTML+CSS+JS+图片）都加载完才能执行——有点儿晚！但是，有些情况下，用户着急使用功能，而不关心界面的好看难看！此时，就不应该强迫用户必须等待图片和 css 加载完才能使用网页！——比如: 春运抢票时 12306



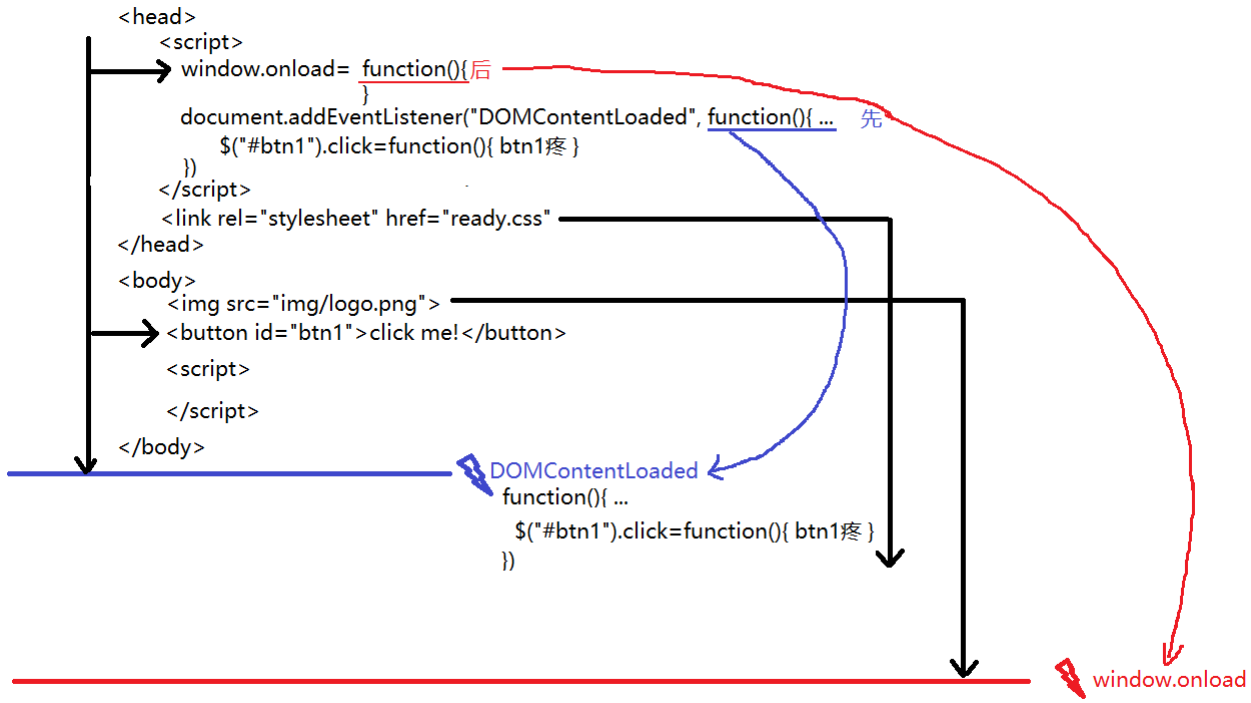
(8). 解决: 其实在 window.onload 之前，还会先触发一次另一个事件——仅 DOM 内容加载后自动执行。学名 DOMContentLoaded。仅等待 HTML+JS，无需等待 CSS 和图片。所以，DOMContentLoaded 比 window.onload 要早触发!

(9). 如何绑定仅 DOM 内容加载完成就提前触发

a. DOM 中: document.addEventListener("DOMContentLoaded",事件处理函数)

b. jq 中: $(document).on("DOMContentLoaded",事件处理函数)

(10). 结论: 今后为了让用户尽早用上功能，而不用关心 css 和图片，大部分 js 代码都应该放在 DOMContentLoaded 中。



(11). 特例: 如果 js 代码中页面一加载就用到了 css 的样式或用到了图片，则只能放在 window.onload 中!

(12). 问题: 因为今后几乎所有的 js 代码都要放在 DOMContentLoaded 中，但是每次都写$(document).on("DOMContentLoaded",事件处理函数)，太麻烦!

(13). 解决: 简写:

a. $(document).ready(function(){ ... })
已准备好

b. $().ready(function(){ ... })

c. $(function(){ ... })

d. ES6: $(()=>{ ... })

(14). 大结局: 今后几乎所有 jq 代码都要放在$(function(){ ... })或$(()=>{ ... })中!

强调: 结尾一定不要画蛇添足加()!!! 因为这是事件绑定，不是匿名函数自调!

(15). 示例: 演示 window.onload 和 DOMContentLoaded 事件的先后顺序

event/14_ready.js

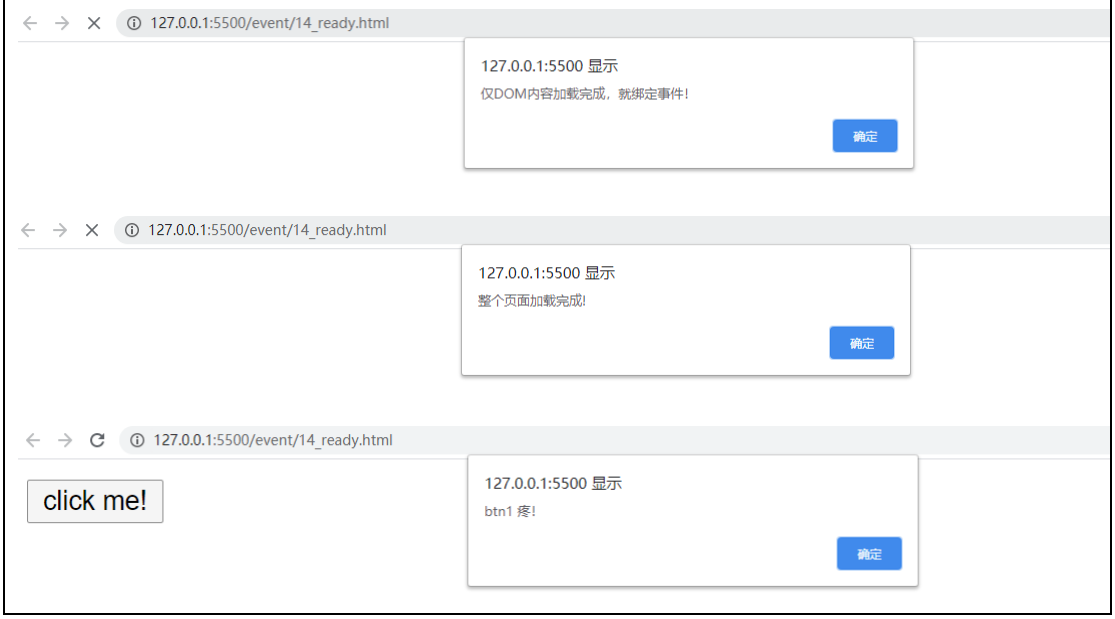
```
//仅绑定整个页面加载完成才自动执行的事件处理函数
//而不是立刻执行!
//window.onload=function(){ //DOM
$(window).load(function(){ //jq
  alert("整个页面加载完成!");
});

//$(document).on("DOMContentLoaded",function(){
//$(function(){
$(()=>{
```

```
  alert("仅 DOM 内容加载完成，就绑定事件！ ");
  //事件绑定代码应该放在 DOMContentLoaded 中,才能早执行!
  $("#btn1").click(function(){
    alert("btn1 疼！ ")
  })
})
```

event/14_ready.html

```
<!DOCTYPE html>
<html>

<head>
  <title> new document </title>
  <meta charset="utf-8">
  <script src="js/jquery-1.11.3.js"></script>
  <script src="14_ready.js"></script>
</head>

<body>
  <button id="btn1">click me!</button>

</body>

</html>
```

运行结果:



3. 鼠标事件:

(1). 问题: DOM 中的鼠标事件 mouseover（鼠标进入）和 mouseout（鼠标移出）两个事件有问题，反复进出子元素，同样会反复触发父元素上的鼠标进出事件——与现实不符!

(2). 解决: 今后 jquery 中以及其他框架中，都用:

a. mouseenter 代替 mouseover 表示鼠标进入

b. mouseleave 代替 mouseout 表示鼠标离开

(3). 优点: 即使反复进出子元素，也不会反复触发父元素上的的进出事件处理函数——与现实相符!

(4). 简写: 如果同时绑定 mouseenter 和 mouseleave，可简写为绑定 hover

$元素.hover(//相当于.mouseenter(...).mouseleave(...)

function(){ ... },

function(){ ... }

)

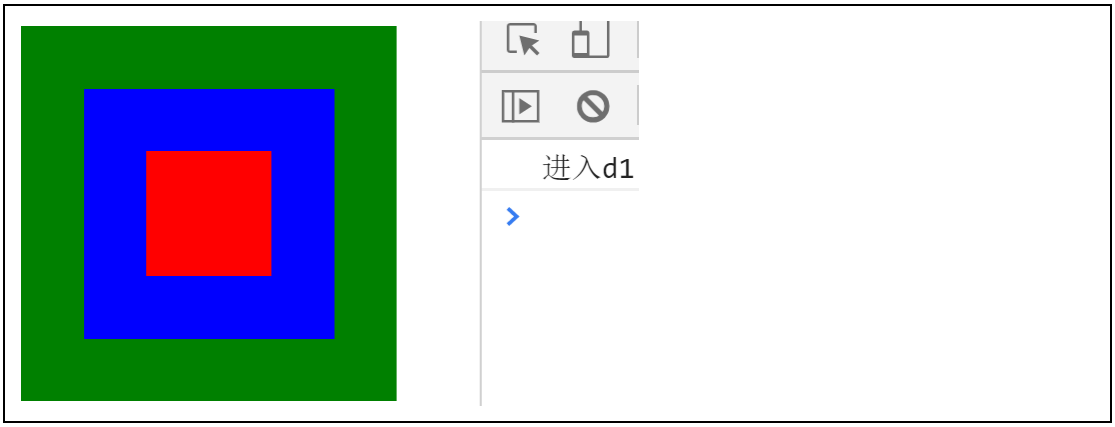
(5). 示例: 使用 mouseenter 和 mouseleave 代替 mouseover 和 mouseout，并用 hover 简写

event/15_mouse.html

```
<!DOCTYPE HTML>
<html>

<head>
  <title>事件处理</title>
  <meta charset="utf-8" />
  <style>
    #d1 #d2 #d3 {
      cursor: pointer
    }

    #d1 {
      background-color: green;
      position: relative;
      width: 150px;
      height: 150px;
      text-align: center;
      cursor: pointer;
    }

    #d2 {
      background-color: blue;
      position: absolute;
      top: 25px;
      left: 25px;
      width: 100px;
      height: 100px;
    }

    #d3 {
      background-color: red;
      position: absolute;
```

```
      top: 25px;
      left: 25px;
      width: 50px;
      height: 50px;
      line-height: 50px;
    }
  </style>
</head>

<body>
  <div id="d1">
    <div id="d2">
      <div id="d3">
      </div>
    </div>
  </div>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    $("#d1")
    //想只要鼠标进入外层 d1 的范围，就输出进入 d1
    //.mouseover(function(){
    // .mouseenter(function(){
    //  console.log(`进入 d1`)
    // })
    //只要鼠标离开外层 d1 的范围，就输出离开 d1
    //.mouseout(function(){
    // .mouseleave(function(){
    //  console.log(`离开 d1`)
    // })
    .hover(//.mouseenter().mouseleave()
      function(){ console.log(`进入 d1`) },
      function(){ console.log(`离开 d1`) }
    );
  </script>
</body>

</html>
```

运行结果:



(6). 特殊: 当两个事件处理函数刚好可以修改为完全相同的两个函数时，其实 hover 中也可以只写一个函数。但是这个函数既给 mouseenter，又给 mouseleave

$元素.hover(function(){ ... })

.mouseenter(...) .mouseleave(...)

(7). 示例: 使用 hover 简写

event/16_hover.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
  <style>
    #target {
      border: 1px solid #eee;
      border-radius: 6px;
      padding: 10px;
      transition: all .5s linear;
    }
    #target.hover {
      border: 1px solid #aaa;
      box-shadow: 0 0 6px #aaa;
      background-color:red;
      color:#fff;
    }
  </style>
 </head>
 <body>
  <h1>使用 hover(fn,fn)</h1>

  <h3>鼠标悬停在 div 上方，则突出显示；移出则取消突出显示</h3>
  <div id="target">
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Elig
endi neque quae voluptatum ducimus culpa itaque maxime distinctio sol
```

```
uta cum cupiditate esse tenetur deserunt fuga perferendis sed veritat
is asperiores. Numquam officia.</p>
  </div>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //希望鼠标进入 id 为 target 的 div，让 div 高亮显示(添加自定义
class hover)
    //鼠标离开 id 为 target 的 div，让 div 恢复正常显示（移除自定义
class hover）
    $("#target")
    .hover(
      function(){//进入时+.hover
        //$(this).addClass("hover")
        $(this).toggleClass("hover")
      },
      // function(){//离开时-.hover
      //  //$(this).removeClass("hover");
      //  $(this).toggleClass("hover")
      // }
    )
  </script>
 </body>
</html>
```

运行结果：

鼠标悬停在div上方，则突出显示；移出则取消突出显示

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eligendi neque quae voluptatum ducimus culpa itaque cum cupiditate esse tenetur deserunt fuga perferendis sed veritatis asperiores. Numquam officia.

鼠标悬停在div上方，则突出显示；移出则取消突出显示

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eligendi neque quae voluptatum ducimus culpa itaque cum cupiditate esse tenetur deserunt fuga perferendis sed veritatis asperiores. Numquam officia.

4. 模拟触发:

(1). 什么是: 即使没有点到按钮，也能触发按钮的单击事件，执行相同的操作

(2). 比如: 百度: 点按钮"百度一下"可以执行搜索，同时在文本框按回车，也可以执行相同的搜索操作。

(3). 如何:

a. 标准: $元素.trigger("事件名")
触发

b. 简写: 如果要触发的事件属于常用事件列表中
可简写为: $元素.事件名()

(4). 示例: 点按钮可以搜索，在文本框按回车也能搜索

event/17_trigger.html

```
<!DOCTYPE html>
<html>

<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
  </style>
</head>

<body>
  <input id="kw"><button>百度一下</button>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //点百度一下按钮，获取文本框中的内容并执行搜索操作
    $("button").click(function(){
      var kw=$("#kw").val().trim();
                              //复习 js 高级第二天
      if(kw!==""){
        console.log(`查找 ${kw} 相关的内容...`)
      }
    })

    //希望在文本框上按下回车键，执行按钮的单击事件操作
    //    键盘按键按下——屌丝可能按住不放!
    //$("#kw").keydown(function(e){
    //    键盘按键抬起
    $("#kw").keyup(function(e){
      //只希望本次按的是回车键时才执行查找操作
      //e.keyCode 属性保存着本次按键键盘号
      //只希望当按的是回车键(13 号)时，才执行后续操作
      if(e.keyCode==13){
        //$("button").trigger("click");
        $("button").click();
      }
    })
  </script>
</body>

</html>
```

运行结果:

二. 动画:

1. 简单动画: 固定的写死的三种动画效果:

(1). 显示隐藏: .show() .hide() .toggle()
显示 隐藏 在显示和隐藏之间来回切换

a. 问题: 以上三个函数, 不加参数时, 默认使用 display:none 或 block 实现显示隐藏, 所以是瞬间显示隐藏, 没有动画效果!

b. 解决: 其实以上三个函数中可以指定动画持续时间参数, 只要提供动画持续时间, 则三个函数才带动画效果

(2). 淡入淡出: .fadeIn() .fadeOut() .fadeToggle()
淡入 淡出 在淡入淡出之间来回切换

(3). 上滑下滑: .slideUp() .slideDown() .slideToggle()
上滑 下滑 在上滑和下滑之间来回切换

简单动画两个致命缺点:
1. 动画效果和变化过程都是在函数中写死的, 整个项目共用一套简单动画函数——几乎不可维护!
2. 可以在 f12 中 element 中看到动画变化过程, 说明这套函数使用 js 定时器和频繁 DOM 操作, 模拟实现的动画效果——效率远不如 css
结论: 以上简单动画函数, 尽量少用!
今后, 凡是简单的过渡效果, 都用 css+transition 实现! 2 个好处:
1. 极其便于维护
2. 效率远比 js 定时器和 DOM 要高的多!
特例: .show() .hide() .toggle()这三个函数在没有实参时, 相当于 display:none 和 block, 没有性能损失的。所以, 不需要动画效果的瞬间显示隐藏, 反而强烈建议用.show() .hide() .toggle()简写

(4). 示例: 测试简单动画各种效果:

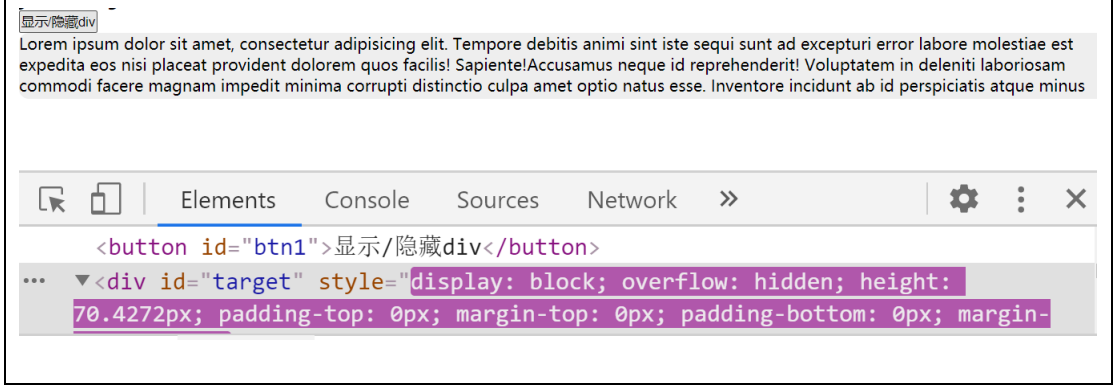
1_show_hide.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
  <style>
```

```
    *{margin:0; padding:0;}
    #target{
      border-radius:10px;
      background:#eee;
    }
    .fade{/*动画起始状态*/
      height:104px; width:970px; opacity:1;
      padding: 10px; overflow:hidden;
      border: 1px solid #aaa;

    }
    .out{/*动画结束状态*/

    }
  </style>
 </head>
 <body>
  <h1>jQuery 动画函数—显示隐藏动画</h1>
  <button id="btn1">显示/隐藏 div</button>
  <div id="target">
    <p><span>Lorem ipsum dolor sit amet, consectetur adipisicing elit
. Tempore debitis animi sint iste sequi sunt ad excepturi error labor
e molestiae est expedita eos nisi placeat provident dolorem quos faci
lis! Sapiente!</span><span>Accusamus neque id reprehenderit! Voluptat
em in deleniti laboriosam commodi facere magnam impedit minima corrup
ti distinctio culpa amet optio natus esse. Inventore incidunt ab id p
erspiciatis atque minus magnam tempore harum.</span></p>
  </div>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    $("#btn1").click(function(){
      // var $div=$("#target");
      //想如果 div 是隐藏的
      // if($div.css("display")=="none"){
      //  $div.show(2000);//就让 div 显示
      // }else{//否则如果 div 是显示的
      //  $div.hide(2000);//就让 div 隐藏
      // }

      //$("#target").toggle(2000) //在显示和隐藏之间来回切换
      //$("#target").fadeToggle(2000);
      $("#target").slideToggle(2000)
    });
  </script>
```

```
 </body>
</html>
```

运行结果：



2. 万能动画：能对多种 css 属性应用动画效果

(1). 如何：

$元素.animate({

要变化的 css 属性：希望变化到的目标值，

... : ...

}, 动画持续时间 ms)

(2). 原理: animate()自动获得元素当前的状态，然后根据我们指定的目标状态求出差值。然后在指定持续时间内，均匀改变元素的状态。

(3). 强调：

a. animate 只支持属性值为单个数值的 css 属性.

比如: width height

opacity

top left right bottom

font-size

margin padding

... ...

b. 与 transition 相比：

1). animate()不支持颜色动画，transition 支持

2). animate()不支持 CSS3 变换, transition 支持

(4). 优缺点：

a. 优点：相对灵活，但是远不如 css transition

b. 缺点：底层用 js 定时器和频繁操作 DOM 树模拟动画——效率极低

(5). 示例：测试 animate()支持和不支持哪些 css 属性

4_animate.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
  <style>
    #d1{
```

```
      border:1px solid #aaa;
      border-radius:6px;
      background:#eee;
      width:50px; height:50px;
      position:absolute; top:120px; left:0;
    }
  </style>
 </head>
 <body>
  <h1>animate</h1>
  <button id="btn1">启动动画</button>
  <div id="d1">abcd</div>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    $("#btn1").click(function(){
      $("#d1").animate({
        //测试支持哪些 css 属性
        //width:300, //height
        //opacity:0,
        left:300, //top right bottom
        //fontSize:32
        //padding:60, //margin

        //不支持哪些 css 属性
        //backgroundColor:"red" //rgb(r,g,b)
        //transform:`rotate(60deg)`//css 中的函数调用语法
      },2000)
    })
  </script>
 </body>
</html>
```

运行结果:

3. 排队和并发:

(1). 并发: 多个 css 属性同时变化

在一个 animate()中写的多个 css 属性，默认是并发变化

(2). 排队: 多个 css 属性先后依次变化

对同一个元素先后调用多个 animate()，先后调用的多个 animate()中的 css 是排队变化

(3). 原理:

a. 其实每个元素对象都有一个自己的动画队列

b. 每次调用 animate()时，不是立刻执行动画的意思，而是将动画加入到元素的动画队列中保存

c. 如果元素的动画队列中之前没有其它正在执行的动画，则新加入的动画才立刻执行

d. 如果元素的动画队列中还有未执行完的动画，则新加入的动画需要等待队列中之前的动画都执行完，才能执行。

4. 停止动画: 随时可停止动画，是 animate()相比 transition 的唯一优点

(1). $元素.stop()

(2). 问题: .stop()默认只能停止动画队列中正在播放的一个动画。队列中之后的动画，依然会触发播放

(3). 解决: $元素.stop(true)

停止当前并清空队列中剩余动画的意思

5. jq 中独有选择器: :animated 匹配正在播放动画的元素

6. 动画结束后自动执行:

(1). 错误: 直接在$元素.aniamte()之后写下一步代码

(2). 因为 animate()底层是 js 定时器，是异步的，之后的代码不会等待定时器执行完，就已经提前执行了

(3). 正确: 其实每个动画函数都有最后一个参数——回调函数:

凡是写在动画函数最后一个回调函数参数中的代码, 注定只能在动画结束后才被自动调用执行!

(4). $元素.animte(

{ css 样式 },

动画持续时间 ms,

function(){

//回调函数中的代码，注定只能在动画结束后才被自动调用执行

}

)

7. 示例: 四颗小星星，点击后，执行不同的动画效果

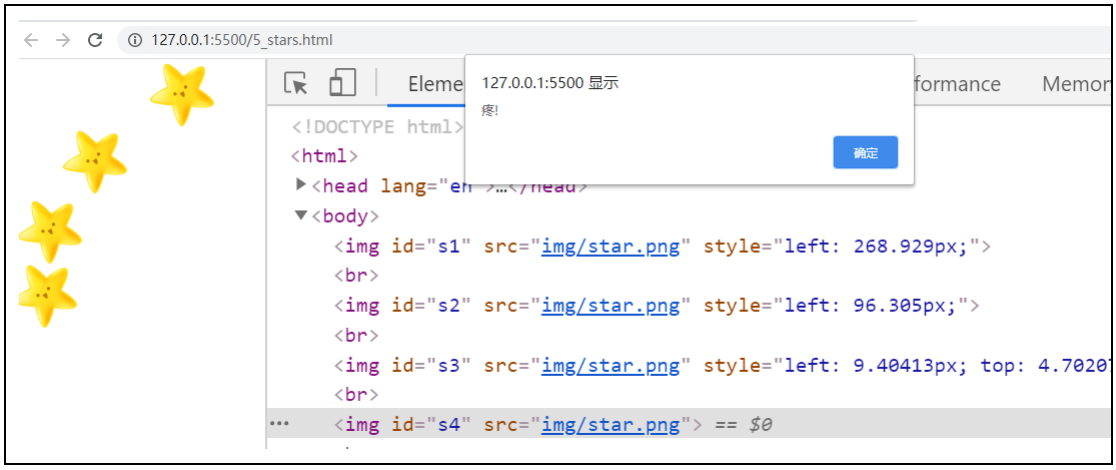
5_stars.html

```
<!DOCTYPE html>
<html>

<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
    img {
      position: relative;
    }
```

```
  </style>
</head>

<body>
  <img id="s1" src="img/star.png"><br />
  <img id="s2" src="img/star.png"><br />
  <img id="s3" src="img/star.png"><br />
  <img id="s4" src="img/star.png"><br />

  <script src="js/jquery-1.11.3.js"></script>
  <script>
    /*
    s1 在屏幕左上角的小星星，  点击后从左移动到屏幕右边
    s2 在屏幕左上角的小星星，点击后从左移动到屏幕右边，再移动到下边——走直角
    s3 在屏幕左上角的小星星，点击后从左上角移动到屏幕右下边，走斜线
    s4 点击小星星，变大、变淡....  直至消失
    */
    $("#s1").click(function(){
      var $this=$(this);
      //点一下启动，点一下停止，再点再启动...
      //如果当前元素正在播放动画
      if($this.is(":animated")){
        $this.stop()//就停止
      }else{//否则如果当前元素没有播放动画
        //才启动动画
        $this.animate({
          left:400
        },2000)
      }
    });
    $("#s2").click(function(){
      var $this=$(this);
      //点一下启动，点一下停止，再点再启动...
      //如果当前元素正在播放动画
      if($this.is(":animated")){
        $this.stop(true)//就停止
      }else{//否则如果当前元素没有播放动画
        //才启动动画
        $this
        .animate({
          left:400
        },2000)
        .animate({
          top:200
```

```
        },1000)
      }
    });
    $("#s3").click(function(){
      var $this=$(this);
      //点一下启动，点一下停止，再点再启动...
      //如果当前元素正在播放动画
      if($this.is(":animated")){
        $this.stop()//就停止
      }else{//否则如果当前元素没有播放动画
        //才启动动画
        $this
        .animate({
          left:400,
          top:200
        },3000)
      }
    })
    $("#s4").click(function(){
      alert("疼!");
      $(this)
      .animate(//底层 js 定时器模拟的动画效果——异步
        {//参数 1
          width:256,
          height:256,
          opacity:0
        },
        2000,//参数 2
        function(){//参数 3
          //animate()回调函数中的 this 指 animate().前的 jquery 对象。
          $(this).hide();//不会等之前异步代码执行完才执行，而是立刻执行
          //$(this).css("display","none");
        }
      );
    })
  </script>
</body>

</html>
```

运行结果:



三. 类数组对象操作:

1. 问题: jQuery 查找结果返回的都是类数组对象。但是 js 中类数组对象非常受歧视。因为类数组对象虽然长的像数组，但是不能使用数组家任何函数！如果这样，jquery 的功能就会收到极大的制约。

2. 解决: jQuery 为了便于自己类数组对象的操作，仿着 js 的数组家，定义了两个几乎一模一样的 jquery 函数

3. jq 中遍历操作:

(1). js 中数组家: forEach //依次取出数组中每个元素值，调用相同的回调函数，执行相同的处理。

遍历每一个　　元素值　下标　当前数组对象

arr.forEach(function(elem,　i,　arr){

...

})

(2). jq 中新增了一个遍历函数: each //依次取出 jquery 结果对象中保存的每个 DOM 元素对象，调用回调函数，对找到的每个 DOM 元素对象执行相同的操作

每一个　　下标　DOM 元素对象

$查找结果.each(function(i,　domElem){

...

})

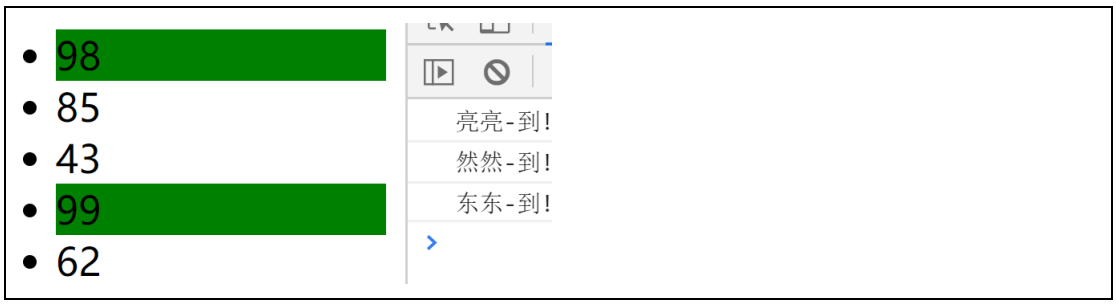
(3). 示例: 使用 forEach 点名，使用 each 遍历 ul 下每个 li

9_each.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
  </style>
</head>
<body>

<ul id="list">
```

```
  <li>98</li>
  <li>85</li>
  <li>33</li>
  <li>99</li>
  <li>52</li>
</ul>

<script src="js/jquery-1.11.3.js"></script>
<script>
  //回顾:forEach
  var arr=["亮亮","然然","东东"];
  arr.forEach(function(elem){
    console.log(`${elem}-到!`)
  })
  //请给每个不足 60 分的成绩+10 分，并将超过 90 分的成绩用绿色背景标识出来
  $("#list>li").each(function(i, domElem){
    //domElem 是一个 DOM 家的元素对象，不是 jq 家孩子
    //如果想用 jq 家简化版函数，得用$()包一下
    var $domElem=$(domElem);
    //如果当前元素的内容<60，就+10 分
    if($domElem.html()<60){
      //先取出旧成绩转为整数
      var score=parseInt($domElem.html())
        //成绩  转换为整数          获取 innerHTML 内容
      //成绩+10
      score+=10;
      //将新成绩放回当前元素内容中
      $domElem.html(score);
    }else if($domElem.html()>=90){
      //否则如果当前元素的内容>=90，就背景变为绿色
      $domElem.css("background-color","green")
          //修改 css 属性  背景      色       绿色
    }
  })
</script>
</body>
</html>
```

运行结果:



4. jq 中查找操作:

(1). js 中数组家: indexOf() 查找一个指定的元素值在数组中的下标位置

var i=arr.indexOf(要找的元素值)

(2). jq 中新增了一个查找函数: index() 查找一个指定的 DOM 元素在查找结果中的下标位置

var i=$查找结果.index(要找的 DOM 元素对象)

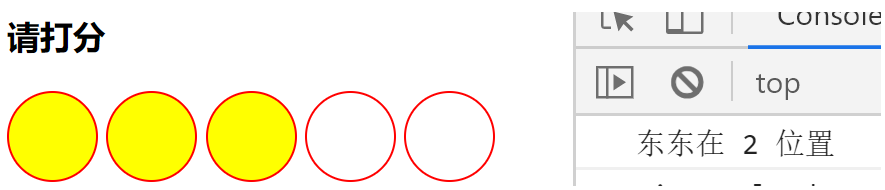
(3). 示例: 评分

10_index.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <style>
    .score {
      list-style: none;
      margin: 0;
      padding: 0;
    }
    .score li {
      display: inline-block;
      width: 50px;
      height: 50px;
      border: 1px solid #f00;
      border-radius: 50%;
      cursor: pointer;
    }
  </style>
</head>
<body>

<h3>请打分</h3>
<ul class="score">
  <li></li>
  <li></li>
  <li></li>
  <li></li>
```

```
  <li></li>
</ul>

<script src="js/jquery-1.11.3.js"></script>
<script>
  //回顾:js 中的 index
  var arr=["亮亮","然然","东东"];
  //在数组 arr 中查找东东的下标位置
  var i=arr.indexOf("东东")
        //  位置 的
  console.log(`东东在 ${i} 位置`)

  //DOM 4 步
  //1. 查找触发事件的元素
  //本例中: class 为 ul 下的每个 li 都能点
  $("ul>li")
  //2. 绑定事件处理函数
  .click(function(){
    //获得当前点击的 li
    var $this=$(this);
    //3. 查找要修改的元素
    //4. 修改元素
    //先将自己改为黄色
    $this.css("background-color","yellow");
    //  修改 css 属性 背景   颜色     黄色
    //获得自己在所有 li 中的下标位置
    // 在 ul 下的所有 li 中 找  当前 li 的位置
    var i=$("ul>li").index($this);
    //                位置
    //alert(i);
    //让<i 位置的都变为黄色
    $(`ul>li:lt(${i})`)
          // less than
          //   小  于
      .css("background-color","yellow");
    //让>i 位置的都变为白色
    $(`ul>li:gt(${i})`)
          //greater than
          //  大     于
      .css("background-color","#fff")
  })
</script>
</body>
</html>
```

运行结果:



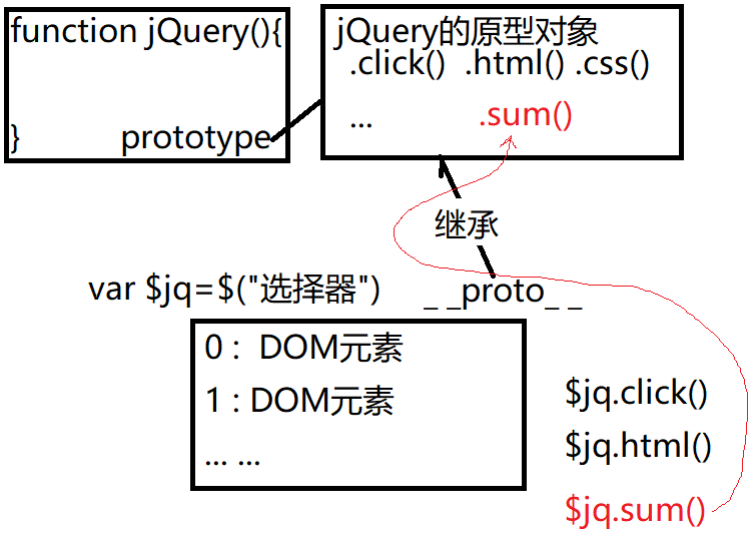
补: 判断当前元素是否符合 xxx 条件
var bool=$元素.is("任意选择器")
是 条件
以为判断$元素是否符合"选择器"的要求!

jq 中$共有 4 种用法:
1. $("选择器") 查找 DOM 元素，并保存进新创建的 jquery 对象中
2. $(DOM 元素对象) 无需查找，直接将已经获得的 DOM 元素，保存进新创建的 jquery 对象中
3. $(`HTML 片段`) 新创建一个 HTML 元素，保存进新创建的 jquery 对象中
4. $(function(){ ... }) 绑定 DOMContentLoaded 事件!

#jQueryday04:

正课:
1. 添加自定义函数
2. 封装自定义插件
3. ajax
4. ***跨域

一. 添加自定义函数:
1. 如果我们经常使用的一个功能，jQuery 中没有提供，其实，我们也可以自定义函数，加入到 jquery 中
2. 如何: 只需要定义一个函数，放入 jQuery 的原型对象中。所有 jquery 家孩子都可以使用这个自定义的函数



3. 示例：为 jquery 添加 sum()函数，可以对找到的所有 DOM 元素求和

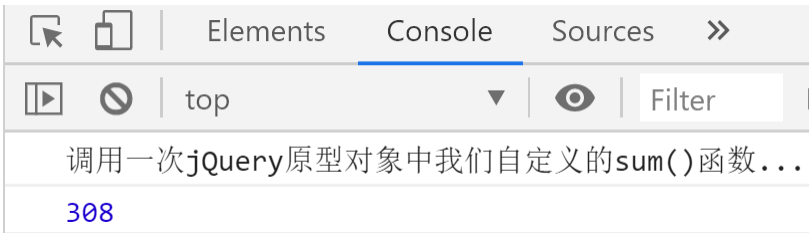
11_sum.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <ul>
    <li>85</li>
    <li>91</li>
    <li>73</li>
    <li>59</li>
  </ul>
  <script src="js/jquery-1.11.3.js"></script>
  <script>
    //为 jQuery 家原型对象中添加一个 sum()函数
    //jQuery.prototype.sum=function(){
    //jQuery.fn=jQuery.prototype
    jQuery.fn.sum=function(){
      console.log(`调用一次 jQuery 原型对象中我们自定义的 sum()函数...`);
      //对一个数组中的内容求和的固定套路:
      //1. 自定义一个变量准备保存累加的和
      var result=0;
      //2. 遍历数组中每个元素
      //问题: 想对找到的所有 DOM 元素内容求和，应该遍历谁?
      //答: 应该遍历本次$("选择器")找到的查找结果对象。因为其中包含了找到的
所有 DOM 元素
      //问题: 如何获得将来调用 sum()函数的.前的$("选择器")查找结果对象?
      //答: 只有一种办法: this
```

```
        //问题：this 是否需要$()包裹？
        //答：不用。因为.前已经是$("选择器")的一个 jq 对象了。所以，不需要再重复$()
        for(var i=0;i<this.length;i++){
          //3. 每遍历数组中一个元素，就将元素值累加到自定义变量 result 上
          //错误：this[i]取出的是 jq 查找结果中某一个 DOM 元素对象，而不是元素的内容
          //result+=this[i]
          //正确：
          //先获得当前查找结果中 i 位置的元素对象
          //再元素对象.innerHTML 获得元素的内容
          //保险起见，将字符串类型的元素内容转为数字
          //parseFloat 既可转整数，又可转小数
          //复习第一阶段 parseFloat 和 parseInt
          result+=parseFloat(this[i].innerHTML)
        }
        //当循环结束后，自定义变量 result 上就累加了所有元素的值
        //4. 返回求和的结果
        return result;
      }



      //比如：经常需要对找到的所有 DOM 元素的内容求和
      //问题：jq 中没有提供对元素内容求和的函数
      console.log($("ul>li").sum());
      //          查找结果
      //          包含了找到的所有 DOM 元素对象
    </script>
</body>
</html>
```

运行结果:

- 85
- 91
- 73
- 59



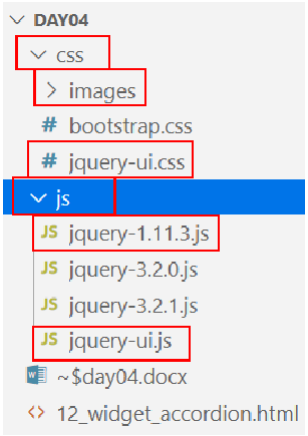
二. 封装自定义插件:

1. 什么是插件: 包含专属的 HTML+JS+CSS 的可重用的独立的页面功能区域
2. 为什么: 重用——公司极其重视重用!

3. 何时: 今后只要出现一个功能需要多处反复使用时，都要先将功能封装为插件，再反复使用插件。

4. jQuery 官方提供了一套插件库: jQuery UI

(1). 官网: jqueryui.com



(2). 在页面中引入 jqueryui 的 css 和 js

<script src="js/jquery-1.11.3.js">

<script src="js/jquery-ui.js">

<link rel="stylesheet" href="css/jquery-ui.css">

(3). 如何使用:

a. 先在页面中按插件的要求编写 HTML 内容

b. 再在自定义 js 中，用 jquery 查找到插件的父元素，调用 jqueryui 提供的插件函数

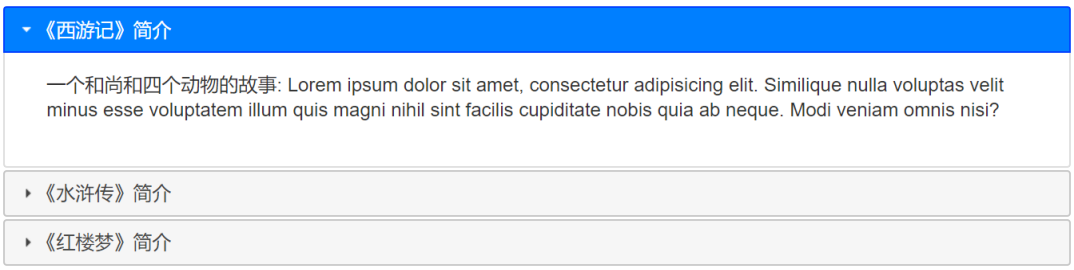
5. 示例: 使用 jqueryui 实现手风琴效果:

12_widget_accordion.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
  <script src="js/jquery-1.11.3.js"></script>
  <script src="js/jquery-ui.js"></script>
  <link rel="stylesheet" href="css/jquery-ui.css">
 </head>
 <body>
  <h1>jQueryUI：Widgets —— Accordion</h1>
  <!--复制 14_accordion.html 中<div class="accordion">的内容到此-->
  <div id="my-accordion">
    <div>《西游记》简介</div>
    <div>一个和尚和四个动物的故
事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Similiqu
e nulla voluptas velit minus esse voluptatem illum quis magni nihil s
int facilis cupiditate nobis quia ab neque. Modi veniam omnis nisi? <
/div>
    <div>《水浒传》简介</div>
```

```
    <div>105 个男人和三个女人的故
事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Omnis pr
ovident sapiente aperiam reprehenderit repellat rem magnam vel odio q
uia harum hic impedit dolorem similique ea est consequatur adipisci a
t nemo!</div>
    <div>《红楼梦》简介</div>
    <div>一个男人和一群女人的故
事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Delectus
 minima quidem aspernatur eligendi optio cupiditate minus nam expedit
a? Aliquid veritatis doloribus maxime vel dicta illo unde iusto qui q
uasi doloremque.</div>
  </div>
  <script>
    $("#my-accordion").accordion();
  </script>
 </body>
</html>
```

运行结果:



6. jquery ui 2 个问题:

(1). 只有 PC 端，没有移动端!

(2). 都是在 js 代码中写死的，极其不便于维护!

7. 但是 jquery ui 这种简化的思路，如果用在我们自己的项目中还是非常有好处的! 所以，如果我们自己封装插件给自己用时，还是非常建议参考 jquery ui 这种方式!

8. jquery ui 封装插件的方式:

(1). jquery ui 所有插件，其实都是一个 jquery 的自定义函数，保存在 jquery 的原型对象中。所以，任何一个 jquery 查找结果对象都可调用插件函数

(2). 问题: 页面上一个 class 都没有，css 样式怎么添加到元素上的?

答: 当我们调用 jquery 插件函数时，jquery 插件函数会根据自身的需要，自动给插件的元素及其子元素添加 class，不再需要使用插件的人手工添加任何 class

(3). 问题: 我们没有写任何事件绑定的代码，但是只调用插件函数，插件中的元素就能点击了!

答; 当我们调用 jquery 插件函数时，jquery 插件函数会根据自身的需要，自动给插件中对应的元素绑定事件处理函数。不再需要插件的使用者手工添加事件绑定代码。

9. 如何自己封装一个 jQuery 自定义插件?

(0). 前提: 封装插件的过程不是从 0 开始开发的过程，而是将一个已经用传统的

HTML+CSS+JS 实现了的界面效果，提取成一个插件而已。所以，要封装插件，必须已经用传统的 HTML+CSS+JS 实现了界面效果

(1). 将插件涉及的 css 代码提取到一个专门的独立的 css 文件中保存

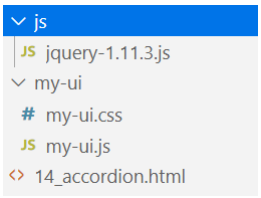
(2). 在一个专门的独立的 js 文件中，为 jquery 原型对象添加一个自定义的插件函数，做两件事:

a. 自动为插件元素及其子元素添加 class

b. 为插件中指定元素添加事件绑定

10. 如何使用自己封装好的插件: 同使用 jquery ui 插件完全相同!

11. 示例: 封装手风琴效果的自定义插件



my-ui/my-ui.css

```
.accordion{width:80%; margin:0 auto;}
.accordion>.title{
  background:#eee; border:1px soild #aaa;
  padding:6px; font-size:1.5em;
  font-weight:bold; cursor:pointer;
}
.accordion>.content{
  border-left:1px solid #eee;
  border-right:1px solid #eee;
}
.accordion>:last-child{
  border-bottom:1px solid #eee;
}
.fade{
  height:0;
  opacity:0;
  overflow:hidden;
  transition:all .5s linear;
}
.in{
  height:84px;
  opacity:1;
}
```

my-ui/my-ui.js

```
//为 jquery 的原型对象添加一个自定义插件函数 myAccordion()，希望将来用户一调用这个 myAccordion()插件函数，class 和事件绑定就自动加入到插件元素上。
jQuery.fn.myAccordion=function(){
```

```
  //this->将来调用这个插件函数的.前的 jquery 对象，也就是整个插件的父元素对象。
  //起个别名
  var $parent=this;
  //自动做 2 件事:
  //1. 为元素自动添加 class
```


```
  //1.1 给父元素自己加 class accordion
  $parent.addClass("accordion")
  //1.2 给父元素下所有奇数位置的直接子元素加 class title
  .children(":nth-child(odd)") //css 中下标从 1 开始
  .addClass("title") //return 奇数位置的子元素
  //1.3 给所以奇数位置的子元素的下一个兄弟加 class content fade
  .next().addClass("content fade") //return 所有奇数位置的子元素的下一个兄弟都返回回来
  //1.4 给所有奇数位置的子元素的下一个兄弟中第一个 div 加 class in
  //first()可获得当前查询结果中第一个位置的元素，且自动包装为 jq 对象。.first()=>$($(查找结果)[0])
  .first().addClass("in");
  //2. 为元素自动绑定事件处理函数
  //直接将页面中已经实现的事件绑定代码剪切到插件函数中，由插件函数负责自动执行事件绑定操作
  $(".accordion").on("click",".title",e=>
    $(e.target).next(".content").toggleClass("in")
      .siblings(".content").removeClass("in")
  );
}

//将来 myAccordion()函数会这样调用:
//$("#my-accordion").myAccordion()
//                       this //所以 this 不用$()包裹
//                         |
//$(整个插件的父元素) ←-点前--
```

```
//已经是$(...)了!
```

14_accordion.html

```
<!DOCTYPE html>
<html>
 <head>
  <title> new document </title>
  <meta charset="utf-8">
  <script src="js/jquery-1.11.3.js"></script>
  <script src="my-ui/my-ui.js"></script>
  <link rel="stylesheet" href="my-ui/my-ui.css">
 </head>
 <body>
  <h1>使用“高度动画”实现“手风琴”组件</h1>
  <div id="my-accordion">
    <div>《西游记》简介</div>
    <div>一个和尚和四个动物的故
事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Similiqu
e nulla voluptas velit minus esse voluptatem illum quis magni nihil s
int facilis cupiditate nobis quia ab neque. Modi veniam omnis nisi? <
/div>
    <div>《水浒传》简介</div>
    <div>105 个男人和三个女人的故
事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Omnis pr
ovident sapiente aperiam reprehenderit repellat rem magnam vel odio q
uia harum hic impedit dolorem similique ea est consequatur adipisci a
t nemo!</div>
    <div>《红楼梦》简介</div>
    <div>一个男人和一群女人的故
事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Delectus
 minima quidem aspernatur eligendi optio cupiditate minus nam expedit
a? Aliquid veritatis doloribus maxime vel dicta illo unde iusto qui q
uasi doloremque.</div>
  </div>
  <script>
    $("#my-accordion").myAccordion();
  </script>
 </body>
</html>
```

运行结果:

《西游记》简介

一个和尚和四个动物的故事: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Similique nulla voluptas velit minus esse voluptatem illum quis magni nihil sint facilis cupiditate nobis quia ab neque. Modi veniam omnis nisi?

《水浒传》简介

《红楼梦》简介

三. ajax

1. 从 jquery 开始, 今后几乎所有的函数库或框架都提供了专门发送 ajax 请求的函数。从此, 任何时候发送 ajax 请求, 只需要一句话!

2. jquery 中发送 get 和 post 类型的 ajax 请求: (固定写法!)

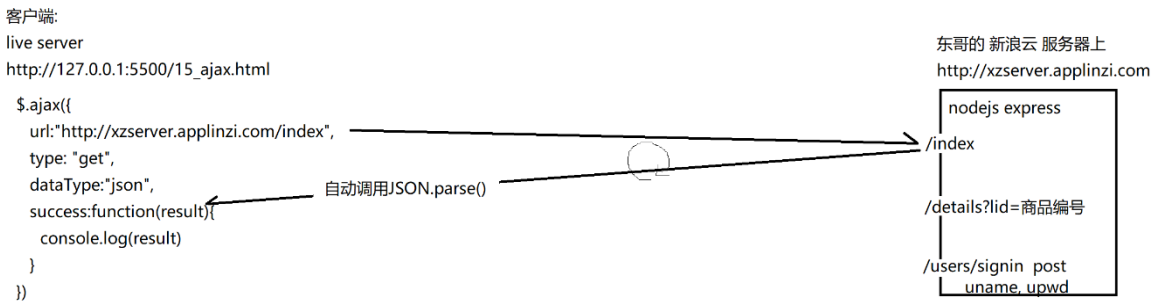
```
$.ajax({
    url:"服务器端接口地址",
    type: "get 或 post",
    data:{ 参数名:参数值, ... ... },
    dataType:"json" , //说明服务器端返回的是 json 字符串, 需要自动调用 JSON.parse()转为 js 对象或数组
    success:function( result ){ //像第二极端的 onreadystatechange, 只在请求响应成功结束后才自动执行!
        //result 就是服务器端返回的响应结果, 且已经被 JSON.parse()转为了 js 的数组或对象, 不再需要手工调用 JSON.parse()了!
    }
})
```

3. 强调:

(1). $.ajax()使用的是类似 ES6 参数结构的语法, 所以, 所有实参值必须凡在一个对象结构中。且对象结构中:前的属性名绝对不能改名! 因为必须和人家函数中规定的形参变量名保持一致才行——复习 ES6 的参数解构

(2). 因为$.ajax()是异步请求, 所以, 如果希望在 ajax 请求响应成功结束后才能执行的所有 js 代码, 必须都放在 success:function(){}内调用或执行! 不能放在 success 外部调用或执行!

4. 示例: 使用$.ajax()向学子商城服务器端发送请求, 获取响应结果



15_ajax.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script src="js/jquery-1.11.3.js"></script>
</head>
<body>
  <script>
    //想从东哥新浪云服务器中请求学子商城首页的 6 个商品对象组成的数组。
    //已知: 服务端接口地址:http://xzserver.applinzi.com/index
    //类型: get
    //不需要参数:
    $.ajax({
      //url:"http://xzserver.applinzi.com/index",
      url:"http://www.codeboy.com:9999/data/product/index.php",
      type: "get",
      dataType:"json", //说明服务器端返回的是 json 字符串，需要自动调用
JSON.parse()转为 js 对象或数组
      success:function( result ){ //只在请求响应成功结束后才自动执行!
        //result 就是服务器端返回的响应结果，且已经被 JSON.parse()转为了 js
的数组或对象，不再需要手工调用 JSON.parse()了!
        console.log(result);
      }
    })
    //想从东哥新浪云服务器/details 接口获得 5 号商品的详细信息
    //类型: get
    //参数: lid=商品编号
    $.ajax({
      // url:"http://xzserver.applinzi.com/details",
      url:"http://www.codeboy.com:9999/data/product/details.php",
      type: "get",
      data:{ lid:5 },
      dataType:"json" ,
      success:function( result ){
        console.log(result);
      }
    })
    //想用东哥新浪云服务器的登陆接口验证用户名和密码
    //类型: post
    //参数: uname=用户名  upwd=密码
    $.ajax({
      // url:"http://xzserver.applinzi.com/users/signin",
      url:"http://www.codeboy.com:9999/data/user/login.php",
```

```
      type: "post",
      data:{ uname:"dingding", upwd:"123456" },
      dataType:"json" ,
      success:function( result ){
        console.log(result);
      }
    })
  </script>
</body>
</html>
```

运行结果:

{carouselItems: Array(4), recommendedItems: Array(6), newArrivalItems: Array(6), topSaleItems: Array(6)}

carouselItems: (4) [{…}, {…}, {…}, {…}]

newArrivalItems: (6) [{…}, {…}, {…}, {…}, {…}, {…}]

recommendedItems: (6) [{…}, {…}, {…}, {…}, {…}, {…}]

topSaleItems: (6) [{…}, {…}, {…}, {…}, {…}, {…}]

__proto__: Object

{details: {…}, family: {…}}

details: {lid: "5", family_id: "2", title: "小米(MI)Air 13.3 英寸全金属超轻薄笔记本(i5-6200U 8G 256G PCIE 固态 940MX 独显 FHD WIN10)银", subtitle: "【i5 独立显卡】全高清窄边框 8G 内存 256G 固态硬盘 支持 SSD 硬盘扩容 薄至 14.8mm 轻至 1.28kg！AKG 扬声器！ ", price: "4999.00", …}

family: {fid: "2", fname: "小米 Air", laptopList: Array(4)}

__proto__: Object

{code: 200, msg: "login succ"}

code: 200

msg: "login succ"

__proto__: Object

四. ***跨域

1. 什么是: 一个网站下的网页去另一个域名下的网站请求资源(文件、数据)
2. 比如:
   <img src="其它网站的图片地址">
   <a href="其它网站的网页">
   <link rel="stylesheet" href="其它网站的 css">
   <script src="其它网站的 js">
   在淘宝或支付宝中，可以查看几乎所有快递公司的快递单进度
3. 包括:
   (1). 域名不同: 从 http://www.a.com 下的网页 请求 http://www.b.com 下的资源
   (2). 子级域名不同: 从 http://oa.tedu.com 下的网页 请求 http://hr.tedu.com 下的资源
   (3). 端口不同: 从 http://localhost:5050 下的网页 请求 http://localhost:3000 下的接

口

(4). 协议不同: 从 http://12306.cn 下的网页 请求 https://12306.cn 下的资源

(5). 即使同一台机器，域名与 IP 之间互访:

从 http://localhost 下的网页 请求 http://127.0.0.1 下的资源

4. 浏览器同源策略（CORS）:

(1). 什么是: 浏览器中有一个安全策略，要求一个网页只能用 ajax 请求来自自己网站的数据。不允许 ajax 请求其它网站的数据。

(2). 何时: 如果用 ajax 从一个网站的网页，向另一个域名下的网站发送请求，就会触发浏览器的同源策略:

(3). 结果: 报错: Access to XMLHttpRequest at 'http://localhost:3000/' from origin 'http://127.0.0.1:5500' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. (这个错误可能因为浏览器版本不同略有不同，但是大致相同！)

翻译: 从源头'http://127.0.0.1:5500' 向 'http://localhost:3000/' 发送的访问请求被 CORS 策略阻止了。因为在响应头中没有设置'Access-Control-Allow-Origin'属性!

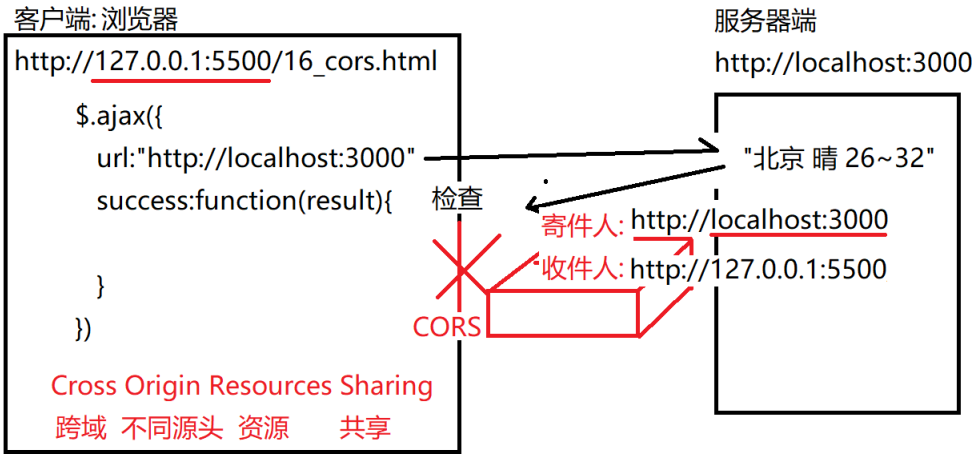
(4). 原理:

a. $.ajax 可以发送 ajax 请求到任何服务器端

b. 但是，服务器端返回结果后，客户端浏览器会检查返回结果的来源地址

c. 默认，服务器端返回结果会原原本本的记录真实服务器所在域名地址

d. 浏览器如果发现服务器端返回的结果上记录的数据来源地址与当前网页所在的域名地址不同！浏览器就会禁止 ajax 使用该返回结果!

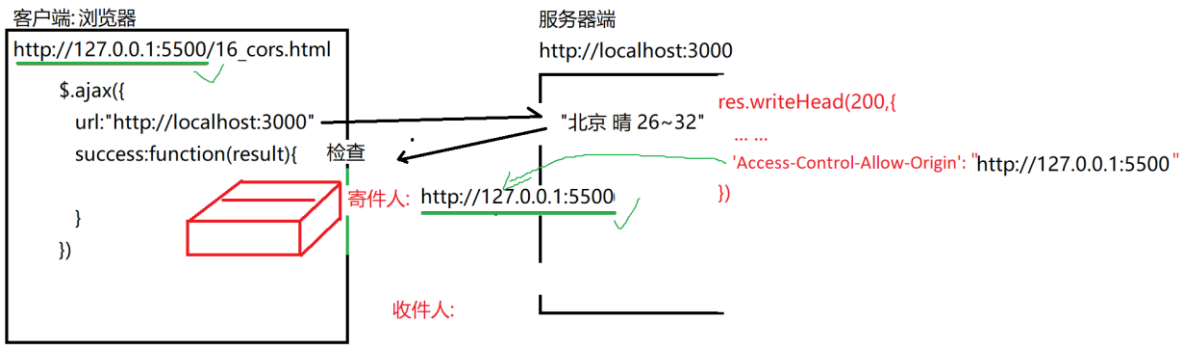


5. 解决跨域问题: 3 种方式:

(1). CORS 方式: 欺骗方式

a. 让服务器端在返回响应结果之前，在响应头中添加一个特殊属性: 'Access-Control-Allow-Origin'，属性值为服务器端要伪装的客户端地址。

```
res.writeHead(200,{
    ... ...
    'Access-Control-Allow-Origin': "客户端地址"
                寄件人
})
```



b. 结果: 服务器端返回的响应结果中的寄件人地址被伪装成了和客户端网页所在地址一样! 则就能顺利通过浏览器 CORS 策略的检查!

(2). JSONP: (略…)

(3). 服务器端中继: 纯服务器端, 我们不需要了解。

6. 示例: 使用服务器端 cors 方式解决跨域问题:

16_server.js

```
//不用写！！！
//也不要求会写！！！！
//只要会运行就行！！！

//声明常量 http，引入 nodejs 标准模块 http 模块，将引入的对象保存在 http 常量中
const http=require("http");
//创建一个服务器端的程序 server，监听一个指定的端口。
var server=http.createServer(
  //当这个端口收到从客户端发开的请求时，自动调用回调函数
  //回调函数的 req 参数，保存了本次请求相关的信息
  //回调函数的 res 参数，保存了本次响应相关的信息
  function(req,res){
    //只要服务端程序收到任何请求
    //都返回一个字符串响应结果:"北京 晴 26~32"
    //因为用的不是 express 框架，所以，不能用 res.send()只能写 res.write() res.end()
    var weather="北京 晴 26~32";
    //修改响应头中的配置信息
    //告诉客户端浏览器
    //              响应成功
    res.writeHead(200,{
    //服务器返回的内容类型为普通文本，且字符编码是 utf-8 格式
      "Content-Type":"text/plain;charset=utf-8",
    //结果: 浏览器就会自动用 utf-8 格式解析服务器端返回的内容
    //将服务器端返回结果中的寄件人地址伪装成和客户端网页地址相同
      "Access-Control-Allow-Origin":"http://127.0.0.1:5500"
    //结果: 服务器端返回的结果中寄件人被伪装成 http://127.0.0.1:5500，和客户端所在地址完全相同，就能顺利通过浏览器的 CORS 策略检查!
```

```
    })
    //将北京的天气字符串，先写入响应对象中暂存。
    res.write(weather);
    //让响应对象 res，将保存的北京天气字符串返回给客户端
    res.end();
  }
)
//让服务器端程序 server 监听 3000 端口
server.listen(3000);
//如何启动服务端程序:
//右键单击 16_server.js
//选择在 xx 终端中打开
//等弹出窗口中出现 C:\xxx\xxxx > 才能继续输入
//输入 node 16_server.js
//手工打开浏览器，地址栏输入: http://localhost:3000
//结果: 看到服务器端程序返回的北京天气字符串
//无论以上服务器端程序作了任何修改，保存后，都要重启服务端:
//  先 Ctrl+C 退出旧的服务器端程序
//  再运行 node 16_server.js
//  重新刷新浏览器中的页面
```

16_cors.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script src="js/jquery-1.11.3.js"></script>
</head>
<body>
  <script>
    $.ajax({
      url:"http://localhost:3000",
      type:"get",
      success:function(result){
        console.log(result);
      }
    })
  </script>
</body>
</html>
```

运行结果:
北京 晴 26~32

总结:

Day01

1. 引入 jQuery.js: 2 种

(1). 引入项目本地 jquery.js 文件:

<script src="js/jquery-1.11.3.js">

(2). 引入 CDN 网络中共享的 jquery.js 文件:

<script src="官方提供的 CDN 上 jquery.js 文件地址">

2. 创建 jQuery 类型子对象: 2 种

(1). 只有 jQuery 类型子对象才能使用 jQuery 家简化版函数。

DOM 家元素对象无权直接使用 jQuery 家简化版函数。

所以只要使用 jQuery 家简化版函数，都要先创建 jQuery 家子对象，其中保存一个要操作的 DOM 元素对象。

(2). 如何:2 种:

a. 查找 DOM 元素对象，并保存进新创建的 jQuery 对象中:

var $jq 子对象=$("选择器")

b. 不查找，直接将 DOM 元素对象保存进新创建的 jQuery 对象中:

var $jq 子对象=$(DOM 元素对象)

3. 原理:

(1). $=jQuery=new jQuery

(2). jq 子对象其实是一个类数组对象，可保存找到的多个 DOM 元素对象

(3). 对 jq 子对象调用简化版函数，会被自动翻译为对应的原生 DOM 的方法和属性。

所以 jQuery 中的所有简化版函数效果和 DOM 中原生方法和属性效果一样。

jQuery 中的 this、e、e.target 等，和 DOM 中的完全一样!

4. jQuery 简化版函数 3 大特点:

(1). 自带 for 循环: 对整个 jquery 子对象调用一次简化版函数, 等效于对 jQuery 子对象中保存的每个 DOM 元素对象分别调用一次对等的 DOM 原生方法或属性——不用自己写 for 循环

(2). 一个函数两用: 调用函数时:

a. 没给新值作为参数，默认执行获取旧值的操作

b. 给了新值作为参数，自动切换为执行修改操作

(3). （待续...）

5. 查找元素:

(1). jQuery 支持用所以 CSS3 选择器查找

(2). jQuery 新增选择器:

a. 基本过滤:

:first :last :eq(i) :lt(i) :gt(i) :even :odd

b. 内容过滤:

:contains(文本) :has(选择器) :parent :empty

c. 可见性过滤:

:visible :hidden

d. 表单元素过滤:

:input :text :password :radio :checkbox ... ...

(3).用节点间关系查找: 2 大类关系, 6 个属性

a. 父子关系: (待续...)

b. 兄弟关系: 2 个

1). 前一个兄弟: $元素.prev()

被自动翻译->元素.previousElementSibling

2). 下一个兄弟: $元素.next()

被自动翻译->元素.nextElementSibling

6. 修改元素: 3 种:

修改中的所有函数, 都是一个函数两用!

(1). 内容: 3 种:

a. 原始 HTML 内容:

$元素.html("新 HTML 内容")

被翻译为-> .innerHTML

b. 纯文本内容:

$元素.text("纯文本内容")

被翻译为-> .textContent

c. 表单元素的值:

$元素.val("新值")

被翻译为-> .value

(2). 属性: 3 种:

a. 字符串类型的 HTML 标准属性:（待续...）

b. bool 类型的 HTML 标准属性:

$元素.prop("属性名", bool 值)

被翻译为->元素.属性名=bool 值

c. 自定义扩展属性:（待续...）

(3). 样式:

a. 获取或修改内联样式:

$元素.css("css 属性名", "属性值")

获取属性值时被翻译为 getComputedStyle(元素)

修改属性值时被翻译为.style.css 属性=属性值

b. 使用 class 批量修改样式: （待续...）

8. 添加删除替换克隆元素: （待续...）

9. 事件绑定: $jq 对象.事件名(function(){ ... })

Day02:

1. 引入 jQuery.js: 2 种

(1). 引入项目本地 jquery.js 文件:

<script src="js/jquery-1.11.3.js">

(2). 引入 CDN 网络中共享的 jquery.js 文件:

<script src="官方提供的 CDN 上 jquery.js 文件地址">

2. 创建 jQuery 类型子对象: 2 种

(1). 只有 jQuery 类型子对象才能使用 jQuery 家简化版函数。

DOM 家元素对象无权直接使用 jQuery 家简化版函数。

所以只要使用 jQuery 家简化版函数，都要先创建 jQuery 家子对象，其中保存一个要操作的 DOM 元素对象。

(2). 如何:2 种:

a. 查找 DOM 元素对象，并保存进新创建的 jQuery 对象中:

var $jq 子对象=$("选择器")

b. 不查找，直接将 DOM 元素对象保存进新创建的 jQuery 对象中:

var $jq 子对象=$(DOM 元素对象)

3. 原理:

(1). $=jQuery=new jQuery

(2). jq 子对象其实是一个类数组对象，可保存找到的多个 DOM 元素对象

(3). 对 jq 子对象调用简化版函数，会被自动翻译为对应的原生 DOM 的方法和属性。

所以 jQuery 中的所有简化版函数效果和 DOM 中原生方法和属性效果一样。

jQuery 中的 this、e、e.target 等，和 DOM 中的完全一样!

4. jQuery 简化版函数 3 大特点:

(1). 自带 for 循环: 对整个 jquery 子对象调用一次简化版函数, 等效于对 jQuery 子对象中保存的每个 DOM 元素对象分别调用一次对等的 DOM 原生方法或属性——不用自己写 for 循环

(2). 一个函数两用: 调用函数时:

a. 没给新值作为参数，默认执行获取旧值的操作

b. 给了新值作为参数，自动切换为执行修改操作

(3). 本没有返回值的函数，默认返回当前正在操作的.前的 jQuery 对象——可链式操作

5. 查找元素:

(1). jQuery 支持用所以 CSS3 选择器查找

(2). jQuery 新增选择器:

a. 基本过滤:

:first :last :eq(i) :lt(i) :gt(i) :even :odd

b. 内容过滤:

:contains(文本) :has(选择器) :parent :empty

c. 可见性过滤:

:visible :hidden

d. 表单元素过滤:

:input :text :password :radio :checkbox ... ...

(3).用节点间关系查找: 2 大类关系, 8 个函数

a. 父子关系: 3 个函数:

1). $元素.parent()

2). $元素.children("选择器")

3). $元素.find("选择器")

b. 兄弟关系: 5 个

$元素.prev() $元素.prevAll("选择器")

$元素.next() $元素.nextAll("选择器")

$元素.siblings("选择器")

6. 修改元素: 3 种:

修改中的所有函数, 都是一个函数两用!

(1). 内容: 3 种:

a. 原始 HTML 内容:

$元素.html("新 HTML 内容") 代替.innerHTML

b. 纯文本内容:

$元素.text("纯文本内容") 代替.textContent

c. 表单元素的值:

$元素.val("新值") 代替.value

(2). 属性: 3 种:

a. 字符串类型的 HTML 标准属性:2 种:

1). $元素.attr("属性名","新属性值")

代替 元素.getAttribute()和 setAttribute()

2). $元素.prop("属性名", bool 值)

代替 元素.属性名=bool 值

b. bool 类型的 HTML 标准属性: 只有 1 种

$元素.prop("属性名", bool 值)

代替 元素.属性名=bool 值

c. 自定义扩展属性:只有一种:

$元素.attr("属性名","新属性值")

代替 元素.getAttribute()和 setAttribute()

反过来总结:

$元素.attr()可修改一切字符串类型的属性(字符串类型 HTML 标准属性+自定义扩展属性)

$元素.prop()可修改一切可用.访问的属性(所有 HTML

标准属性)

(3). 样式:

a. 获取或修改单个 css 属性: 只有 1 种

$元素.css("css 属性名", "属性值")

获取属性值时被翻译为 getComputedStyle(元素)

修改属性值时被翻译为.style.css 属性=属性值

b. 使用 class 批量修改样式:

1). $元素.addClass("class 名")

2). $元素.removeClass("class 名")

3). $元素.hasClass("class 名")

4). $元素.toggleClass("class 名")

修改相关的函数都可同时修改多个属性值:

$元素.attr 或 prop 或 css({

属性名:"属性值",

... : ...

})

8. 添加删除替换克隆元素:

a. 添加新元素: 2 步

1). 使用 HTML 片段批量创建新元素:

$(`HTML 片段`)

2). 将新元素添加到 DOM 树: 5 种方式, 10 个函数

i. 末尾追加:

$父元素.append($新元素)

$新元素.appendTo($父元素)

ii. 开头插入: 新增:

$父元素.prepend($新元素)

$新元素.prependTo($父元素)

iii. 插入到一个现有元素之前:

$现有元素.before($新元素)

$新元素.insertBefore($现有元素)

iv. 插入到一个现有元素之后:

$现有元素.after($新元素)

$新元素.insertAfter($现有元素)

v. 替换现有元素:

$现有元素.replaceWith($新元素)

$新元素.replaceAll($现有元素)

b. 删除元素: $元素.remove()

c. 克隆元素: $元素.clone()

9. 事件绑定:

a. 标准写法: $元素.on("事件名", 事件处理函数)

b. 简写: $元素.事件名(事件处理函数)

Day03:

jQuery API 三大特点:

1. 自带遍历:

2. 一个函数两用:

3. 返回当前正在操作的.前的 jQuery 对象。——链式操作!

5. 查找元素:

(1). jQuery 支持用所以 CSS3 选择器查找

(2). jQuery 新增选择器:

基本过滤、内容过滤、可见性过滤:、表单元素过滤:

(3).用节点间关系查找: 2 大类关系, 8 个函数

a. 父子关系： 3 个函数:

1). $元素.parent()

2). $元素.children("选择器")

3). $元素.find("选择器")

b. 兄弟关系： 5 个

$元素.prev()

$元素.prevAll("选择器")

$元素.next()

$元素.nextAll("选择器")

$元素.siblings("选择器")

6. 修改元素: 3 种:

修改中的所有函数，都是一个函数两用!

(1). 内容: 3 种:

(2). 属性: 3 种:

a. 字符串类型的 HTML 标准属性:2 种:

1). $元素.attr("属性名","新属性值")

代替 元素.getAttribute()和 setAttribute()

2). $元素.prop("属性名", bool 值)

代替 元素.属性名=bool 值

b. bool 类型的 HTML 标准属性: 只有 1 种

$元素.prop("属性名", bool 值)

代替 元素.属性名=bool 值

c. 自定义扩展属性:只有一种:

$元素.attr("data-属性名","新属性值")

代替 元素.getAttribute()和 setAttribute()

反过来总结:

$元素.attr()可修改一切字符串类型的属性(字符串类型 HTML 标准属性+自定义扩展属性)

$元素.prop()可修改一切可用.访问的属性(所有 HTML 标准属性)

(3). 样式:

a. 获取或修改单个 css 属性: 只有 1 种

$元素.css("css 属性名", "属性值")

获取属性值时被翻译为 getComputedStyle(元素)

修改属性值时被翻译为.style.css 属性=属性值

b. 使用 class 批量修改样式:

1). $元素.addClass("class 名")

2). $元素.removeClass("class 名")

3). $元素.hasClass("class 名")

4). $元素.toggleClass("class 名")

修改相关的函数都可同时修改多个属性值:

$元素.attr 或 prop 或 css({

属性名:"属性值",

... : ...

})

8. 添加删除替换克隆元素:

a. 添加新元素: 2 步

1). 使用 HTML 片段批量创建新元素:

$(`HTML 片段`)

2). 将新元素添加到 DOM 树: 5 种方式, 10 个函数

i. 末尾追加:

$父元素.append($新元素)

$新元素.appendTo($父元素)

ii. 开头插入: 新增:

$父元素.prepend($新元素)

$新元素.prependTo($父元素)

iii. 插入到一个现有元素之前:

$现有元素.before($新元素)

$新元素.insertBefore($现有元素)

iv. 插入到一个现有元素之后:

$现有元素.after($新元素)

$新元素.insertAfter($现有元素)

v. 替换现有元素:

$现有元素.replaceWith($新元素)

$新元素.replaceAll($现有元素)

b. 删除元素: $元素.remove()

c. 克隆元素: $元素.clone()

9. 事件绑定:

a. 标准写法: $元素.on("事件名", 事件处理函数)

b. 简写: $元素.事件名(事件处理函数)

c. 事件委托:

```
$父元素.on("事件名","选择器",function(){
    ...this 指向 e.target...
})
```

d. 页面加载后自动执行:

1). 先$(document).ready(function(){})

简写: $(function(){ ... })

2). 后$(window).load(function(){ ... })

e. 鼠标事件:

mouseenter 代替 mouseover

mouseleave 代替 mouseout

简写: .hover(处理函数 1,处理函数 2)

等于: .mouseenter(处理函数 1)

.mouseleave(处理函数 2)

f. 模拟触发:

1). 标准: $元素.trigger("事件名")

2). 如果属于常用事件列表, 可简写为:

$元素.事件名()

10. 动画:

a. 简单动画: 3 种固定效果

1). 显示隐藏:

.show() .hide() . toggle()

2). 淡入淡出:

.fadeIn() .fadeOut() .fadeToggle()

3). 上滑下滑:

.slideUp() .slideDown() .slideToggle()

b. 万能动画:

$元素.animate({

css 属性: 目标值

},动画持续时间,function(){

动画播放结束自动执行

})

c. 排队和并发:

1). 放在一个 animate()中的多个 css 属性并发变化

2). 放在一个元素的多个 animate()中的多个 css 属性排队变化

d. 停止动画:

1). 只停止当前一个动画

$元素.stop()

2). 停止队列中所有动画

$元素.stop(true)

e. 选择器匹配正在播放动画的元素: :animated

11. 类数组对象操作:

1). 遍历查找结果中每个 DOM 元素对象:

$查找结果.each(function(i, domElem){ … })

2). 查找一个 DOM 元素在整个查找结果中的下标位置

var i=$查找结果.index(要找的 DOM 元素)

Day04:

jq 中$共有 4 种用法:

1. $("选择器") 查找 DOM 元素，并保存进新创建的

jquery 对象中

2. $(DOM 元素对象) 无需查找，直接将已经获得的 DOM 元素，保存进新创建的 jquery 对象中

3. $(`HTML 片段`) 新创建一个 HTML 元素，保存进新创建的 jquery 对象中

4. $(function(){ ... }) 绑定 DOMContentLoaded 事件!

总结: jQuery API 三大特点:

1. 自带遍历:

2. 一个函数两用:

3. 返回当前正在操作的.前的 jQuery 对象。——链式操作!

总结:

9. 事件绑定:

a. 标准写法: $元素.on("事件名"，事件处理函数)

b. 简写: $元素.事件名(事件处理函数)

c. 事件委托:

```
$父元素.on("事件名","选择器",function(){
    ...this 指向 e.target...
})
```

d. 页面加载后自动执行:

1). 先$(document).ready(function(){})

简写: $(function(){ ... })

2). 后$(window).load(function(){ ... })

e. 鼠标事件:

mouseenter 代替 mouseover

mouseleave 代替 mouseout

简写: .hover(处理函数 1,处理函数 2)

等于: .mouseenter(处理函数 1)

.mouseleave(处理函数 2)

f. 模拟触发:

1). 标准: $元素.trigger("事件名")

2). 如果属于常用事件列表，可简写为:

$元素.事件名()

10. 动画:

a. 简单动画: 3 种固定效果

1). 显示隐藏:

.show() .hide() . toggle()

2). 淡入淡出:

.fadeIn() .fadeOut() .fadeToggle()

3). 上滑下滑:

.slideUp() .slideDown() .slideToggle()

b. 万能动画:

$元素.animate({

css 属性: 目标值

},动画持续时间,function(){

动画播放结束自动执行

})

c. 排队和并发:

1). 放在一个 animate()中的多个 css 属性并发变化

2). 放在一个元素的多个 animate()中的多个 css 属性排队变化

d. 停止动画:

1). 只停止当前一个动画

$元素.stop()

2). 停止队列中所有动画

$元素.stop(true)

e. 选择器匹配正在播放动画的元素: :animated

11. 类数组对象操作:

1). 遍历查找结果中每个 DOM 元素对象:

$查找结果.each(function(i, domElem){ ... })

2). 查找一个 DOM 元素在整个查找结果中的下标位置

var i=$查找结果.index(要找的 DOM 元素)

12. 添加自定义函数:

a. 添加 jQuery.fn.自定义函数=function(){ … }

b. 调用: $jq 对象.自定义函数()

13. 封装自定义插件:

a. 封装:

1). 将 css 提取到独立的插件.css 文件中

2). 在独立 js 文件中为 jquery 原型对象添加自定义插件函数

```
jQuery.fn.自定义插件函数=function(){
    //自动做 2 件事:
    //1. 为插件所在元素及其子元素自动添加 class
    //2. 将原网页中插件所需的事件绑定代码剪切到插件函数中
}
```

b. 使用自定义插件:

1). 引入 jquery.js，插件.js，插件.css

2). 按插件要求编写 HTML 内容，不用加任何 class

3). 在自定义 js 中，查找插件所在元素，调用自定义插件函数

14. 发送 ajax 请求:

```
$.ajax({
    url:"服务器端接口地址",
```

```
    type:"get 或 post",
    data:{ 参数名: 参数值, ... },
    dataType:"json",
    success:function(result){
        //result 就是服务器端返回的结果
        //不用自己调 JSON.parse()
    }
})
```

15. 跨域:

a. 同源策略(CORS): 浏览器只允许当前网页中的 ajax 请求和使用自己网站的资源，禁止 ajax 请求和使用其他网站的资源。

b. 解决: cors 方式： 只靠服务器端就能完成:

```
res.writeHead(200,{
    "Access-Control-Allow-Origin": "http://客户端网页所在网址"
})
```

c. 结果: 服务器端将响应结果中的寄件人地址伪装成和客户端网页所在地址一致，就可骗过浏览器 cors 策略的检查，让 ajax 顺利使用其他网站服务器端返回的结果数据了。张景元