准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->DOM-> day03、04

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->DOM-> day03、04

(学有余力!!!的同学或自己认为自己基础好的同学必须看!!!!!!)：

今日扩展视频：

小程序->在线->DOM->day03

2. 选择省份和城市 级联下拉列表

作业: 使用navigator.userAgent鉴别浏览器名称和版本号

小程序->在线->DOM->day04

3. 利用冒泡 事件委托 delegate 。。。

5. 鼠标坐标x y 原生DOM拖拽效果

作业: 原生DOM页面滚动动画效果

作业: 高频笔试题nodejs浏览器不同事件循环

赠送：原生DOM轮播图

3. 今日课程中包含的英语单词的翻译：小程序->单词->DOM->day03、04

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例/视频，至少写三遍: (如果需要画图，必须亲手画图)

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩10天）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

谁问的问题越多！谁的培训费交的越值！

正课:

1. HTML DOM

\*\*\*\*BOM\*\*\*\*

1. 什么是BOM

2. window对象

3. 打开新链接

4. history

5. location

6. navigator

7. userAgent

8. 事件

一. HTML DOM:

1. <form>元素:

(1). 获取页面中一个form元素: var form=document.forms[i]

a. 因为浏览器已经将一个网页中所有的<form>元素都保存在document的forms集合中

b. 可以用下标[i]，访问forms集合中保存的第i个form元素对象.

c. 如果网页中只有一个表单元素: var form=document.forms[0]

(2). 获取<form>元素内的表单元素时:

a. 标准: form.elements[id/i/name]

1). 因为form已经将表单中的所有表单元素都集中保存在了form的elements集合中。

2). 可通过下标[i]或[id]或[name]方式来获取<form>内的一个或多个表单元素

3). 返回值:

i. 如果要找的表单元素只有一个，则返回一个元素对象，不放在类数组对象中

ii. 如果要找的表单元素时多个，则放在一个类数组对象中返回。

b. 简写: 如果这个元素有name属性，则可以简写为:

form.name名

(3). 让表单元素自动获得焦点: 表单元素.focus()

(4). 示例: 简化带样式的表单验证案例，并让姓名文本框自动获得焦点

day03 valiWithCSS/0\_valiWithCss.html

|  |
| --- |
| <!doctype html>  <html>  <head>  <meta charset="UTF-8">  <title>实现带样式的表单验证</title>  <style>  table{width:700px}  td:first-child{width:60px}  td:nth-child(2){width:200px}  td:first-child+td{width:200px}  td span{color:red}  .vali\_info{  display:none;  }  .txt\_focus{  border-top:2px solid black;  border-left:2px solid black;  }  .vali\_success,.vali\_fail{  background-repeat:no-repeat;  background-position:left center;  display:block;  }  .vali\_success{  background-image:url("images/ok.png");  padding-left:20px;  width:0px;height:20px;  overflow:hidden;  }  .vali\_fail{  background-image:url("images/err.png");  border:1px solid red;  background-color:#ddd;  color:Red;  padding-left:30px;  }  </style>  </head>  <body>  <form id="form1">  <h2>增加管理员</h2>  <table>  <tr>  <td>姓名：</td>  <td>  <input name="username"/>  <span>\*</span>  </td>  <td>  <div class="vali\_info">  10个字符以内的字母、数字或下划线的组合  </div>  </td>  </tr>  <tr>  <td>密码：</td>  <td>  <input type="password" name="pwd"/>  <span>\*</span>  </td>  <td>  <div class="vali\_info">6位数字</div>  </td>  </tr>  <tr>  <td></td>  <td colspan="2">  <input type="submit" value="保存"/>  <input type="reset" value="重填"/>  </td>  </tr>  </table>  </form>  <script>  //DOM 4步  //1. 查找触发事件的元素  //本例中: 用户操作姓名文本框时会触发变化  // var txtName=document.getElementsByName("username")[0];  //[0]: 从查询结果的类数组对象中取出找到的唯一的文本框元素对象  // var txtPwd=document.getElementsByName("pwd")[0];  //先找<form>元素：  var form=document.forms[0];  //再在form元素中用简写查找表单元素  var txtName=form.username;  var txtPwd=form.pwd;  //希望开局就让姓名文本框自动获得焦点  txtName.focus();  console.log(txtName);  //2. 绑定事件处理函数  //本例中: 当文本框获得焦点时，旁边的提示框，显示出来  txtName.onfocus=function(){  //3. 查找要修改的元素  //本例中: 当文本款获得焦点时，应该查找文本框的爹的下一个兄弟td中的div  var div=this.parentElement.nextElementSibling.children[0];  //4. 修改元素  //本例中: 当文本款获得焦点时，让旁边的div显示,脱掉div的隐身衣class  div.className="";  }  //2. 绑定事件处理函数  //本例中: 当文本框失去焦点时，就要验证文本框中用户输入的内容，并根据验证结果修改旁边提示框的样式类  txtName.onblur=function(){  //3. 查找要修改的元素  //本例中：当文本框失去焦点时，还是要修改文本框旁边的div的class  var div=this.parentElement.nextElementSibling.children[0];  //4. 修改元素  //先定义正则表达式来规定姓名的格式  var reg=/^\w{1,10}$/;  //验证当前文本框的内容是否符合格式要求  var bool=reg.test(this.value);  //如果验证通过  if(bool==true){  //就修改div的className为vali\_success  div.className="vali\_success";  }else{//否则如果验证不通过  //就修改div的className为vali\_fail  div.className="vali\_fail";  }  }  </script>  </body>  </html>  运行结果: |

\*\*\*\*BOM\*\*\*\*Browser Object Model

浏览器 对象 模型

一. 什么是BOM:

1. 什么是BOM: 专门操作浏览器窗口的对象和函数

2. 何时: 今后只要希望操作浏览器窗口，或获取浏览器软件相关信息，都用BOM~~而不是DOM~~

3. 问题: BOM没有标准，兼容性极差！用的越来越少！

4. 包括:

(1). window: 3个角色

(2). history: 控制前进后退和刷新

(3). location: 控制页面跳转，也可以获取当前地址栏里的url信息

(4). navigator: 保存着所有浏览器的配置信息

(5). screen: 保存着当前浏览器所在的硬件设备的显示设备信息

(6). event: 保存着浏览器中触发的事件的相关信息

(7). document: 属于DOM，专门保存和操作网页的内容——前边已学过

二. window： 3个角色:

因为过于简单，所以小程序中未配视频，回看TMOOC或看笔记正文复习即可。

1. 代替ECMAScript中的global充当js语言的全局作用域对象

我们在全局声明的变量和函数默认都保存在window中

2. 集中保存了所有原生的直接可用的对象和函数：

(1). 原生: 所有浏览器自带的，不需要我们自己定义，就可直接使用的对象或函数

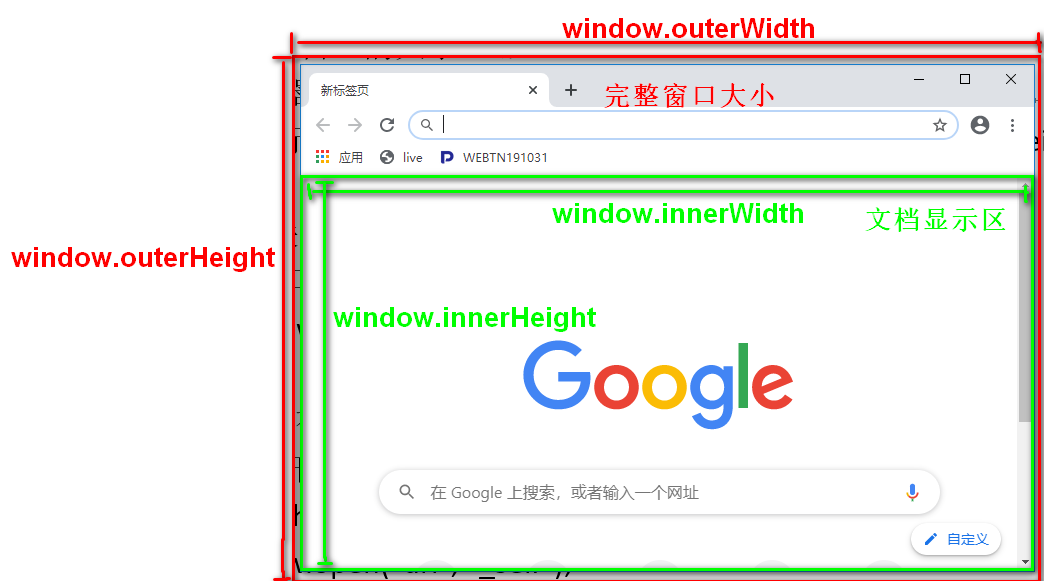
(2). 包括: ECMAScript+DOM+BOM=原生

3. window代表当前浏览器窗口对象:

(1). 获取当前窗口的大小: 2组:

a. 完整浏览器窗口大小: window.outerWidth window.outerHeight

b. 仅文档显示区部分的大小: window.innerWidth window.innerHeight



(2). 打开和关闭窗口:

a. 打开新窗口: window.open()

b. 关闭窗口: window.close()

三. 打开新链接共有几种方式：4种:

对应小程序视频: 小程序->在线->DOM->day03 4. 打开新链接4种方式

1. 在当前窗口打开新链接，可以后退

(1). HTML: <a href="url" target="\_self" >

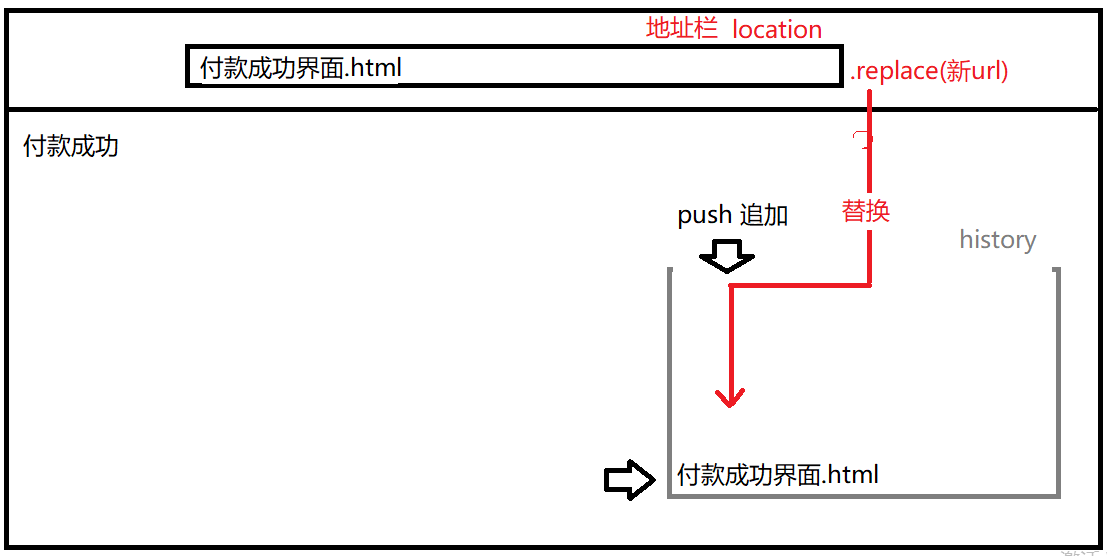
(2). JS: window.open("url", "\_self");

2. 在当前窗口打开新链接，禁止后退

~~(1). HTML:~~

(2). JS: location.replace("新url");

用新url代替history中旧的一条url，无法再后退到旧url了！



3. 在新窗口打开新链接，可以同时打开多个

(1). HTML: <a href="url" target="\_blank" >

(2). JS: window.open("url", "\_blank");

4. 在新窗口打开新链接，同时只能打开一个！

(1). HTML: <a href="url" target="自定义窗口名" > -> 新窗口的window.name值

(2). JS: window.open("url", "自定义窗口名"); -> 新窗口的window.name值

5. 原理:

(1). 每个窗口在内存中都有一个唯一的name属性，标记当前窗口的名字。

(2). 浏览器规定相同name属性值的窗口，在内存中同时只能打开一个

(3). 后打开的同名窗口，会覆盖先打开的同名窗口！

(4). 其实: <a>的target属性和window.open()的第二个实参值，都是在给新的窗口指定name名！

(5). 强调: 禁止使用"name"当做变量名或属性名！因为name是BOM中的保留字，有特殊用途！

(6). 预定义窗口名:

a. \_self，不自定义窗口名，而是自动将当前窗口自己的名字，作为新窗口的name名，

自己的

后果: 因为新窗口和旧窗口自己同名，所以新窗口覆盖旧窗口，不会额外打开新窗口了。

b. \_blank: 也不自定义窗口名，而是让窗口名空白！但是浏览器不允许窗口名空着，浏览器就会在底层自动为新窗口分配新随机名称（人看不见！）。

blank: 空白

后果: 因为每次打开的新窗口随机名称都不一样！所以，不会覆盖，而是可以反复打开多个！

6. 示例: 以四种不同的情况打开新链接

day03 BOM/4\_open.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8"/>  <title>打开新链接方式总结</title>  <script>  /\*打开新链接方式总结：  1. 在当前窗口打开，可后退  2. 在当前窗口打开，不可后退  3. 在新窗口打开，可打开多个  4. 在新窗口打开，只能打开一个  \*/  function open1(){  window.open("http://tmooc.cn","\_self");  }  function open2(){  location.replace("http://tmooc.cn");  }  function open3(){  window.open("http://tmooc.cn","\_blank");  }  function open4(){  window.open("http://tmooc.cn","tmooc");  }  </script>  </head>  <body>  <h3>1. 在当前窗口打开，可后退</h3>  <a href="http://tmooc.cn" target="\_self">欢迎访问tmooc</a><br>  <button onclick="open1()">欢迎访问tmooc</button>  <h3>2. 在当前窗口打开，禁止后退</h3>  <button onclick="open2()">欢迎访问tmooc</button>  <h3>3. 在新窗口打开，可同时打开多个</h3>  <a href="http://tmooc.cn" target="\_blank">欢迎访问tmooc</a><br>  <button onclick="open3()">欢迎访问tmooc</button>  <h3>4. 在新窗口打开，同时只能打开一个</h3>  <a href="http://tmooc.cn" target="tmooc">欢迎访问tmooc</a><br>  <button onclick="open4()">欢迎访问tmooc</button>  </body>  </html>  运行结果: |

四. history:

1. 什么是: 当前窗口中保存当前窗口打开后成功访问过得所有url地址的历史记录数组

2. 何时: history只给我们开放了前进后退和刷新的功能。其内容，我们既不可见，又不能修改！

3. 如何: 只能做三件事:

(1). 前进一步: history.go(1)

(2). 后退一步: history.go(-1)

有时如果后退一步不好用，还可以后退两步: history.go(-2)

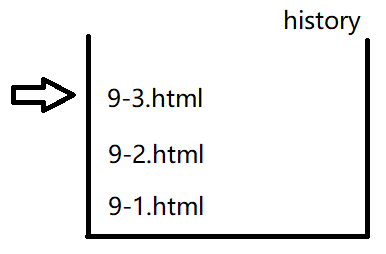
(3). 刷新: history.go(0)

4. 示例: 用js程序实现前进后退

步骤: 先用live server运行起来9-1.html

然后，依次点击超链接2和3跳转到9-2.html和9-3.html

结果: 让当前窗口的history中，有三个历史记录url，才可以测试前进后退功能



day03 BOM/

9-1\_history.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用 history 对象</title>  <meta charset="utf-8" />  </head>  <body>  <h2>9-1.html</h2>  <div>  <a href="9-1\_history.html">1</a>&nbsp;&nbsp;  <a href="9-2\_history.html">2</a>&nbsp;&nbsp;  <a href="9-3\_history.html">3</a>&nbsp;&nbsp;  </div><br/>  <div>  <a href="javascript: history.go(1)">前进一次</a>&nbsp;&nbsp;  <a href="javascript: history.go(2)">前进二次</a>&nbsp;&nbsp;  </div>  </body>  </html>  运行结果: |

9-2\_history.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用 history 对象</title>  <meta charset="utf-8" />  </head>  <body>  <h2>9-2.html</h2>  <div>  <a href="9-1\_history.html">1</a>&nbsp;&nbsp;  <a href="9-2\_history.html">2</a>&nbsp;&nbsp;  <a href="9-3\_history.html">3</a>&nbsp;&nbsp;  </div><br/>  <div>  <a href="javascript: history.go(-1)">后退一次</a>&nbsp;&nbsp;  <a href="javascript: history.go(1)">前进一次</a>&nbsp;&nbsp;  </div>  </body>  </html>  运行效果: |

9-3\_history.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>使用 history 对象</title>  <meta charset="utf-8" />  </head>  <body>  <h2>9-3.html</h2>  <div>  <a href="9-1\_history.html">1</a>&nbsp;&nbsp;  <a href="9-2\_history.html">2</a>&nbsp;&nbsp;  <a href="9-3\_history.html">3</a>&nbsp;&nbsp;  </div><br/>  <div>  <a href="javascript: history.go(-2)">后退二次</a>&nbsp;&nbsp;  <a href="javascript: history.go(-1)">后退一次</a>&nbsp;&nbsp;  </div>  </body>  </html>  运行结果 |

五. location

地址/位置

1. 什么是: 保存当前地址栏中的地址信息，并提供页面跳转操作的函数 的对象

2. 何时: 2种:

(1). 获得地址栏中的包含的相关信息时  
 (2). 执行页面跳转操作时

3.. 如何:

(1). 分段获得地址栏中url中各个部分的内容

a. location.href 获得完整url地址

b. location.protocol 获得协议部分

c. location.host 获得主机名+端口号

d. location.hostname 仅获得主机名

e. location.port 获得端口号

f. location.pathname 获得相对路径

g. location.search 获得?及其之后的查询字符串  
 h. location.hash 获得#及其之后的锚点地址

电话上的#

(2). 提供了一些函数:

a. location.assign("新url") 在当前窗口打开新链接，跳转到新url

等效于: window.open("新url","\_self")

b. location.replace("新url") 在当前窗口打开新链接，禁止后退

c. location.reload() 刷新页面

等效于: history.go(0)

4. 示例: 获取location各个部分的内容

day03 BOM/10\_location.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>事件处理</title>  <meta charset="utf-8" />  </head>  <body>  <form>  姓名:<input name="username"/><br>  密码:<input type="password" name="pwd"/><br>  爱好:<input type="checkbox" name="favs" value="running"/>跑步  <input type="checkbox" name="favs" value="swimming"/>游泳  <input type="checkbox" name="favs" value="basketball"/>篮球  <br>  <input type="submit"/>  </form>  <a href="#top">返回顶部</a>  <script>  console.log(location.href);  console.log(location.protocol);  console.log(location.host);  console.log(location.hostname);  console.log(location.port);  console.log(location.pathname); //在js程序中判断，根据pathname不同，动态改变主导航菜单中哪个菜单高亮显示，其余菜单恢复原样  console.log(location.search);  console.log(location.hash);  </script>  </body>  </html>  运行结果: |

六. navigator

1. 什么是: 保存浏览器软件和配置信息的对象

2. 何时: 只要希望获得浏览器配置信息时，都可用navigator

3. 如何:

(1). 查看浏览器已经安装的插件列表，比如判断是否安装了QQMusic插件或flash插件

navigator.plugins

(2). 查看浏览器的名称、内核和版本号

navigator.userAgent

七. 事件:

1. 什么是事件: 浏览器自动触发的或用户手动触发的页面内容和状态的变化

2. 什么是事件处理函数: 在事件发生前，提前保存在元素的事件属性上的函数。当事件发生时，浏览器会自动调用元素的事件属性上的事件处理函数。

3. 何时: 只要希望当事件发生时，自动执行一项任务时，都要提前绑定事件处理函数

4. 如何绑定事件处理函数: 3种:

对一个小程序视频: 小程序->在线->DOM->day04 1. 添加事件监听 移除事件监听

(1). 在HTML中绑定: ——几乎不用

a. HTML中: <元素 on事件名="fun()">

b. js中: 定义一个全局函数:function fun(){ ... }

c. 问题: 不符合内容与行为分离的原则，极其不便于维护。

(2). 在js中用赋值方式绑定:

a. 元素对象.on事件名=function(){ ... }

b. 问题: 一个事件属性上只能赋值一个事件处理函数，不灵活

无法当一个事件发生时，同时执行多个事件处理函数！  
 (3). 在js中通过添加事件监听对象的方式:

a. 元素对象.addEventListener("事件名", 事件处理函数)

添加 事件 监听对象

b. 原理:

1). 在浏览器内存中有一个巨大的事件监听对象队列

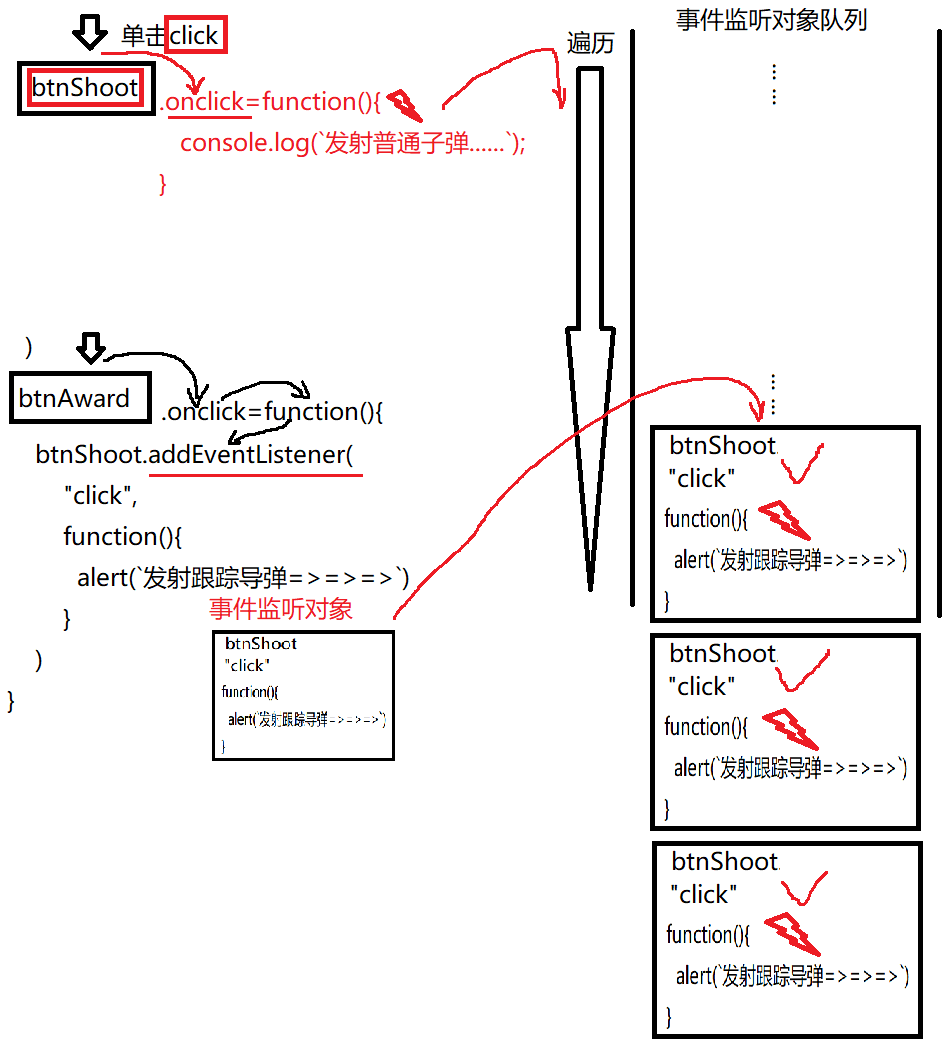
2). 每次addEventListener()时，都会:2件事

i. 先创建一个事件监听对象:包含三项内容:

①哪个元素②什么事件③事件处理函数

ii. 将新创建的事件对象添加到浏览器的事件队列中保存

3). 当事件在一个元素上发生时，浏览器会遍历事件队列中每个事件监听对象，找到和当前元素当前事件名匹配的一个或多个监听对象，自动执行其中的事件处理函数。找到集合符合条件的监听对象，就执行几个事件处理函数。



c. 移除事件监听对象:

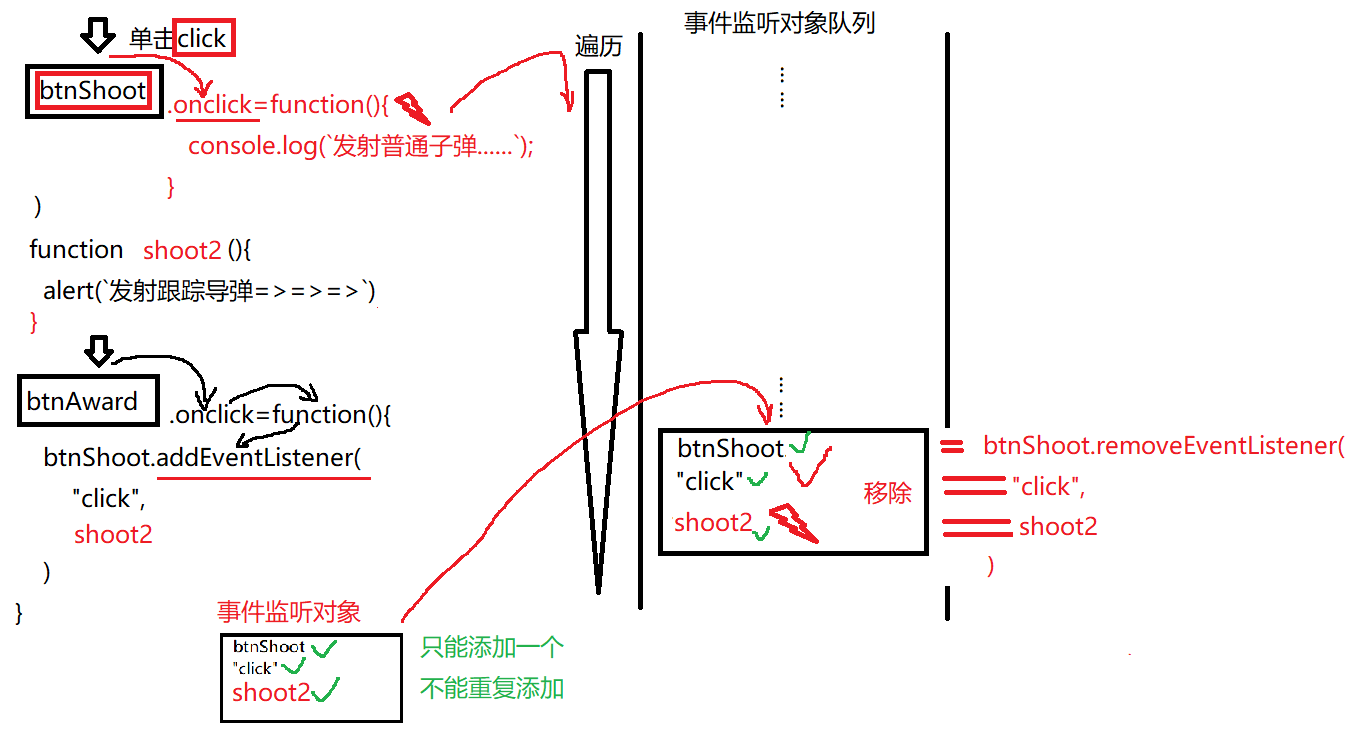
1) 元素对象.removeEventListener("事件名",原事件处理函数)

2). 问题: 移除事件处理函数时，如果只是把事件处理函数写的和原来事件处理函数一模一样，依然移除不掉！

3). 原因: 浏览器在移除事件监听对象时，会比较元素对象是否相同，事件名是否相同，事件处理函数是否是原事件处理函数对象（比的是地址！）。三个比较都相同，才能移除这个事件监听对象。如果绑定时和移除时都用匿名函数，则两次function的地址绝对不同！只要地址不同！就无法移除原事件监听对象。

4). 解决: 今后，只要一个事件处理函数有可能被移除，就不能用匿名函数。因为匿名函数创建后，就再也找不到了！应该用有名称的函数！移除时还可通过函数名获得原函数对象的地址!

5). 强调: 一旦使用有名称的函数添加事件监听，则完全相同的事件监听对象只能添加一个到队列中，不予许重复添加！



(4). 总结:

a. 如果一个元素的一个事件上，只需要一个处理函数，不会增加新的处理函数，就依然可以使用.onclick=function(){ ... }

b. 只有一个元素的一个事件上，有可能同时执行多个事件处理函数时，才被迫改为用addEventListener()

(5). 示例: 点按钮发射多种子弹

day04 1\_addEventListener.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8"/>  <title>...</title>  </head>  <body>  <button id="btnShoot">shoot</button><br>  <button id="btnAward">获得跟踪导弹</button><br>  <button id="btnBreak">失去跟踪导弹</button><br>  <script>  var btnShoot=document.getElementById("btnShoot");  var btnAward=document.getElementById("btnAward");  var btnBreak=document.getElementById("btnBreak");  //希望开局点shoot按钮，只能发射一种普通子弹  btnShoot.onclick=function(){  console.log(`发射普通子弹......`);  }  //希望点获得跟踪导弹之后，为shoot按钮多添加一种新的跟踪导弹，希望再点shoot时，能同时发射两种子弹！  //因为跟踪导弹的事件处理函数有可能被移除  //所以必须定义为有名称的函数  function shoot2(){  alert(`发射跟踪导弹=>=>=>`)  }  btnAward.onclick=function(){  // btnShoot.onclick=function(){  // alert(`发射跟踪导弹=>=>=>`)  // }  //因为跟踪导弹的事件处理函数有可能被移除  //所以绑定时就要用函数名绑定  btnShoot.addEventListener("click",shoot2)  }  //希望点失去跟踪导弹，从shoot按钮上移除跟踪导弹的事件处理函数  btnBreak.onclick=function(){  //移除时必须用原函数名移除  btnShoot.removeEventListener("click",shoot2)  }  </script>  </body>  </html>  运行结果： |

5. 事件模型:

对应小程序视频: 小程序->在线->DOM->day04 2. 事件模型 event model ...

(1). 什么是: 从触发事件到所有事件处理函数执行完所经历的过程

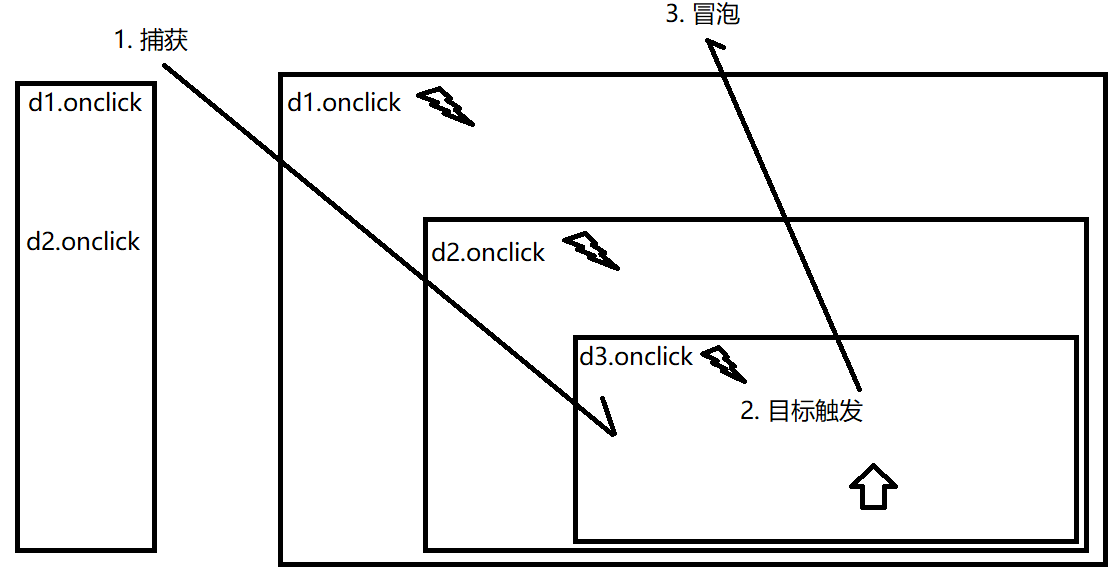
(2). 包括: 3个阶段:

a. 捕获: 由外向内依次记录单击的元素的所有父元素上绑定的事件处理函数都有哪些。

——只记录不执行！

b. 目标触发: 优先触发当前点击的元素上的事件处理函数

c. 冒泡: 由内向外依次触发捕获阶段记录的所有父元素上的事件处理函数



6. 事件对象:

(1). 什么是: 事件发生时，浏览器自动创建的，保存事件信息的对象

(2). 何时: 2种:

a. 想获得事件信息时

b. 想修改事件的默认行为时

(3). 先获得事件对象，才能使用: 事件对象总是默认作为事件处理函数的第一个实参值悄悄传入！

在绑定事件处理函数时:

//当事件发生时自动创建event对象

↓ 信任！

元素.onclick = function( e ){

//e=event

}

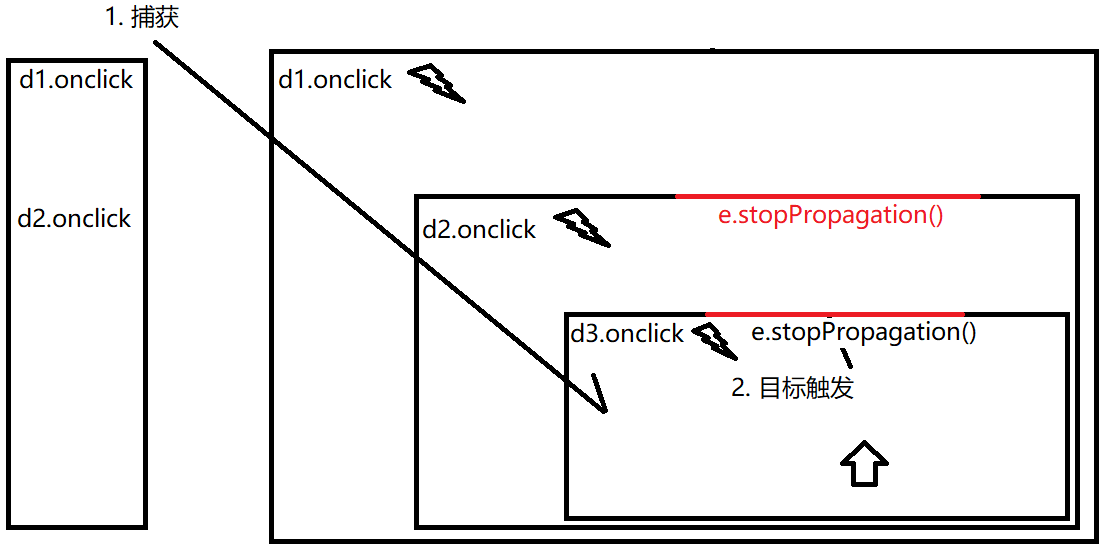
(4). 提供了:

a. 取消冒泡/停止蔓延: e.stopPropagation()

停止 蔓延

1). 强调: 在当前事件处理函数内，e.stopPropagation()写前写后无所谓！

因为: e.stopPropagation()防的是父元素触发事件，不防当前元素自己！



2). 示例: 点哪个div，哪个div喊疼！没点的！不要喊！

2\_bubble.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>  var d1=document.getElementById("d1");  var d2=document.getElementById("d2");  var d3=document.getElementById("d3");  //想点哪个div，哪个div喊疼！  d1.onclick=function(){  alert("d1疼！")  }  d2.onclick=function(e){  e.stopPropagation()  alert("d2疼！")  }  d3.onclick=function(e){  e.stopPropagation()  alert("d3疼！")  }  </script>  </body>  </html>  运行结果: |

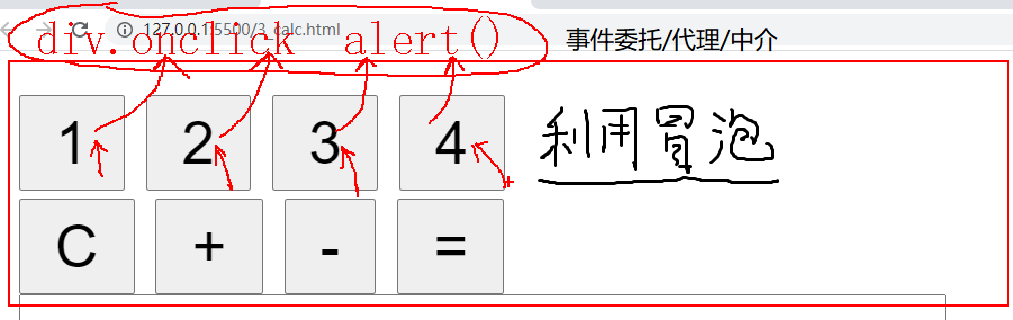
b. 利用冒泡/事件委托:

对应小程序视频: 小程序->在线->DOM->day04 3. 利用冒泡...

1). 问题: 浏览器触发事件时，是通过遍历监听对象队列的方式，查找符合要求的监听对象。所以，监听对象队列中的对象数量多少，决定了查找的速度。队列中监听对象越多，查找越慢。反之，队列中监听对象越少，查找越快

2). 优化: 尽量减少整个页面中事件监听的个数！

3). 如何: 今后只要多个平级子元素都需要绑定相同的事件时，其实只需要把事件集中绑定在父元素上一份，所有子元素就可通过冒泡共用！因为任何情况只要点击子元素，一定会冒泡触发父元素上的事件处理函数。

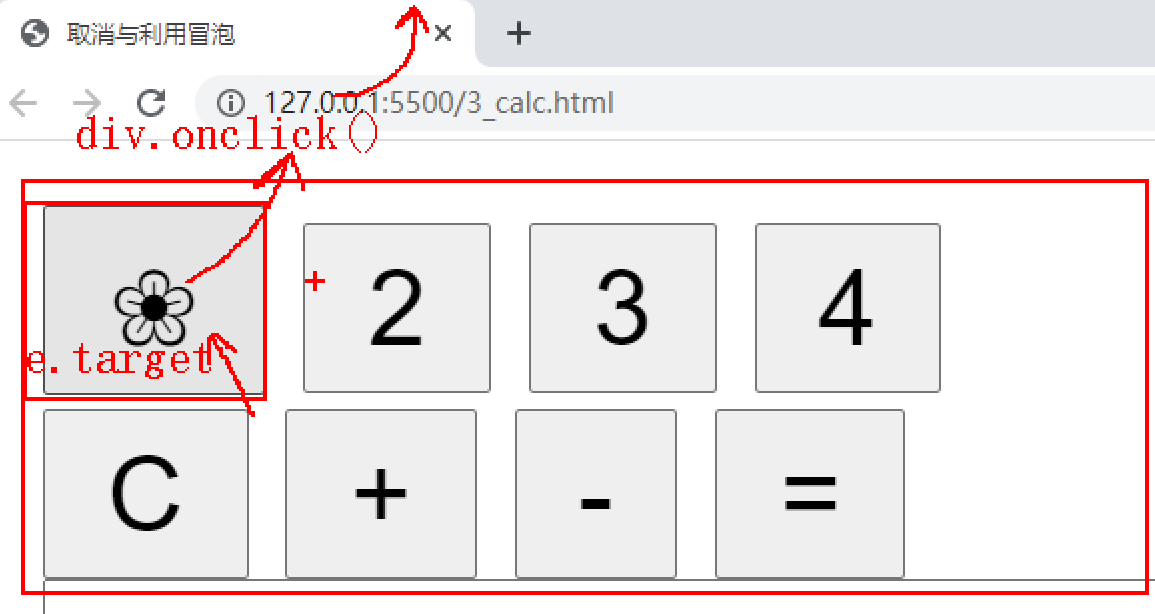


4). 问题: 当事件绑定在父元素上时，this不再指实际触发事件的子元素，而是指父元素了！事件绑定在哪个元素上，this就指谁！

5). 解决: 今后只要利用冒泡，都要用e.target代替this，用于获得最初触发事件的那个子元素！

e.target: 事件对象中自动保存的属性，专门用于保存最初触发事件的那个子元素。

vs this: e.target不会随冒泡而改变！this会随冒泡而改变！



6). 问题: 如果点的子元素不是想要的，我们就不想让它触发事件。只有点在希望的元素上才触发事件

7). 解决: 今后只要利用冒泡，都要先判断e.target是不是想要的子元素。只有e.target的特征是想要的子元素，才能继续执行后续操作。反之，如果e.target的特征不是想要的，则什么也不做！

补: 元素.nodeName 属性 记录元素的标签名！强调: 全大写！

8). 总结: 利用冒泡3步:

a. 事件只绑定在父元素上一份

b. e.target代替this，获得实际点击的子元素

c. 先判断e.target的特征符合要求，才能继续执行后续操作！

9). 示例: 实现计算器效果：

3\_calc.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>取消与利用冒泡</title>  <meta charset="utf-8"/>  </head>  <body>  <div id="keys">  <button>1</button><span>\*</span>  <button>2</button><span>\*</span>  <button>3</button><span>\*</span>  <button>4</button><span>\*</span><br>  <button>C</button><span>\*</span>  <button>+</button><span>\*</span>  <button>-</button><span>\*</span>  <button>=</button><span>\*</span>  </div>  <textarea id="sc" style="resize:none;width:200px; height:50px;" readonly></textarea>  <script>  //DOM 4步  //1. 查找触发事件的元素  //本例中: id为keys下的每个button都可点击  //因为每个按钮都能点，所以事件应该只绑定在父元素div上一份即可  var div=document.getElementById("keys")  //2. 绑定事件处理函数  //本例中: 绑定单击事件：  div.onclick=function(e){  //希望点每个按钮都能喊疼！  //alert("疼!");  //希望点哪个按钮，让哪个按钮的内容变成❀  //错误: 事件是绑定在父元素div上，所以this->父元素div  //this.innerHTML="❀";  //正确: e.target，代替this  //只希望button元素才能变成❀，别的元素即使点了，也不要变！  // 标签名  if(e.target.nodeName=="BUTTON"){  //e.target.innerHTML="❀"  //3. 查找要修改的元素  //本例中：每次点击按钮，都要修改下方的文本框  var sc=document.getElementById("sc");  //4. 修改元素  //先判断点击的按钮的内容  switch(e.target.innerHTML){  //如果是C，就清空显示屏文本框  case "C":  sc.value="";  break;  //如果是=，就将文本框内容，交给eval计算结果，再将结果放回显示屏中  case "=":  //错误处理: 复习第一阶段  try{//尝试执行  //eval: 可计算字符串类型的js表达式的值.——复习第一阶段  sc.value=eval(sc.value);  }catch(err){//如果出错  sc.value=err;//就把错误信息显示在文本框中  }  break;  //点击其余所有数字按钮和+ -号按钮，都只将按钮内容追加到文本框中算式结尾即可，不做计算  default:  sc.value+=e.target.innerHTML;  }  }    }  </script>  </body>  </html>  运行效果: |

c. 阻止元素自带的默认行为:

对应小程序视频: 小程序->在线->DOM->day04 4. 阻止元素自带默认行为

1). 问题: HTML中个别元素身上自带了一些自动执行的操作，但是，这些操作，有时我们不想用！

2). 比如:

i. <a href="#">xxx</a> 当点这个a时，会自动在地址栏结尾加#

ii. <input type="submit"/>和<button>放在<form>中时，只要点击，就自动提交！

3). 如何: e.preventDefault();

阻止 默认

4). 比如:

i. < a id="a1" href="#">xxx</a>

var a1=document.getElementById("a1");

a1.onclick=function (e){ e.preventDefault() ... }

ii. <input id="btnSubmit" type="submit">

<button id="btnSubmit">xxx</button>

var btnSubmit=document.getElementById("btnSubmit")

btnSubmit.onclick=function(e){ e.preventDefault() ... }

5). 简写:

i. <a href="javascript:;">xxx</a>

javascript: 让a点击时，不要跳转，而是执行一条js语句

; 什么也不做的空语句！

ii. <input type="button" value="提交"/>

<button type="button">提交</button>

6). 总结:

i. 今后只要想做一个什么都不做的死链接: <a href="javascript:;">

ii. 今后希望点表单中提交按钮时不要自动提交: <input type="button">

7). 示例: 验证a元素和input元素自带的默认行为，并阻止默认行为

4\_preventDefault.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>  </head>  <body>  <a id="a1" href="#">什么也不干！</a><br/>  <a href="javascript:;">什么也不干！</a>  <form>  用户名: <input name="uname"/><br/>  <!-- <input id="btnSubmit" type="submit"> -->  <!-- <button id="btnSubmit">提交</button> -->  <!-- <input type="button" value="提交"/> -->  <button type="button">提交</button>  </form>  <script>  var a1=document.getElementById("a1");  a1.onclick=function(e){  e.preventDefault();  alert("疼!")  }  var btnSubmit=document.getElementById("btnSubmit");  btnSubmit.onclick=function(e){  e.preventDefault();  alert("用ajax手动提交表单！")  }  </script>  </body>  </html>  运行结果: |

d. 获取鼠标位置: 当事件发生时，鼠标所在的位置有三组坐标

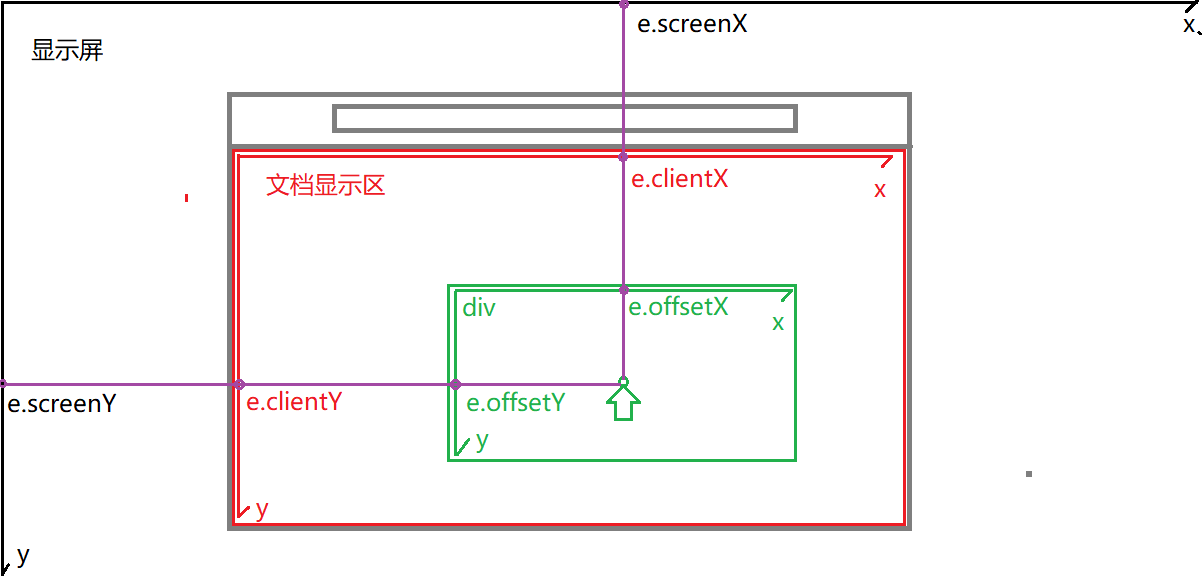
对应小程序视频: 小程序->在线->DOM->day04 5. 鼠标坐标 x y （拖拽效果选学）

(1). 鼠标位置相对屏幕左上角的距离: e.screenX e.screenY

(2). 鼠标位置相对浏览器文档显示区左上角的距离 e.clientX e.clientY

(3). 鼠标位置相对事件所在元素左上角的距离 e.offsetX e.offsetY

偏离 偏离



(4). 示例: 点击元素中某个位置，获得三组坐标:

5\_pop.html

|  |
| --- |
| <!doctype html>  <html>  <head>  <meta charset="UTF-8">  <title>在当前显示区范围内实现点不到的小方块</title>  <style>  div{  position:fixed;  top:50px;  left:100px;  width:100px;  height:100px;  background-image:url(images/xiaoxin.gif);  background-size:100%;  }  </style>  </head>  <body>  <div id="pop"></div>  <script>  var pop=document.getElementById("pop");  pop.onclick=function(e){  console.log(`鼠标距离:  屏幕左侧: ${e.screenX}, 屏幕顶部: ${e.screenY}  浏览器文档显示区左侧: ${e.clientX}, 文档显示区顶部: ${e.clientY}  div左侧: ${e.offsetX}, div顶部: ${e.offsetY}  `)  }  </script>  </body>  </html>  运行结果: |

e. 窗口滚动事件:

对应小程序视频: 小程序->在线->DOM->day04 作业: 原生DOM页面滚动（动画效果选学）

1). 其实每当窗口滚动条发生滚动时，都会自动触发window.onscroll事件

2). window.onscroll=function(){ ... }中的代码，会在窗口滚动条滚动时自动触发

3). 获得滚动条滚动过的距离: 固定套路！——信任

var scrollTop=document.body.scrollTop||document.documentElement.scrollTop;

4). 还可以主动控制滚动到哪个位置: window.scrollTo(0, 纵向滚动到的位置)

说明: scrollTo第一个实参是水平方向的滚动位置，但是网页几乎很少水平滚动，所以第一个实参值几乎总是0。

5). 示例: 实现不带动画的电梯按钮

7\_scroll.html

|  |
| --- |
| <!doctype html>  <html>  <head>  <meta charset="UTF-8">  <title>根据页面滚动位置显示浮动框</title>  <style>  body {  height: 2000px;  }  #toTop {  position: fixed;  bottom: 100px;  right: 0;  display: none;  }  </style>  </head>  <body>  <div style="height:150px;"></div>  <div style="height:150px; background-color:pink">1楼</div>  <div style="height:150px; background-color:lightGreen">2楼</div>  <div style="height:150px; background-color:lightBlue">3楼</div>  <div style="height:150px; background-color:orange">4楼</div>  <div id="toTop">  <a id="a1" href="javascript:;">1楼</a>  <a id="a2" href="javascript:;">2楼</a>  <a id="a3" href="javascript:;">3楼</a>  <a id="a4" href="javascript:;">4楼</a>  <a href="javascript:;">返回顶部</a>  </div>  <script>  var div = document.getElementById("toTop");  //只要窗口的滚动条发生滚动时，就获取滚动条滚动过的距离  window.onscroll = function () {  var scrollTop = document.body.scrollTop || document.documentElement.scrollTop;  console.log(scrollTop);  //只要滚动距离>=500，就让div显示  if (scrollTop >= 500) {  div.style.display = "block";  } else {//否则如果滚动距离不足500，就让div隐藏  div.style.display = "none";  }  }  //点击a时回到顶部  var a = document.querySelector("#toTop>a:last-child")  a.onclick = function () {  window.scrollTo(0, 0);  }  //点哪个楼层按钮，跳到哪个楼层  var a1=document.getElementById("a1");  var a2=document.getElementById("a2");  var a3=document.getElementById("a3");  var a4=document.getElementById("a4");  a1.onclick=function(){  window.scrollTo(0, 150);  }  a2.onclick=function(){  window.scrollTo(0, 300);  }  a3.onclick=function(){  window.scrollTo(0, 450);  }  a4.onclick=function(){  window.scrollTo(0, 600);  }  </script>  </body>  </html>  运行结果: |

总结: DOM操作万能4步:

1. 查找触发事件的元素

2. 为找到的元素绑定事件处理函数

3. 查找要修改的元素

4. 修改元素

总结: DOM 不用背！！！用的时候来找到即可！

1. 查找元素: 4种:

(1). 不需要查找就可直接获得的元素对象: 4个

a. document 根节点对象

b. document.documentElement <html>

c. document.head <head>

d. document.body <body>

(2). 按节点间关系查找: 如果已经获得一个元素，找周围附近的元素时，2大类关系，6个属性

a. 父子关系: 4个属性:

1). 元素.parentElement  
 2). 元素.children

3). 元素.firstElementChild

4). 元素.lastElementChild

b. 兄弟关系: 2个属性

1). 元素.previousElementSibling

2). 元素.nextElementSibling

(3). 按HTML特征查找元素: 只用一个条件就可以找到想要的元素时，4个函数

a. var 一个元素对象=document.getElementById("id名")

b. var 类数组对象=任意父元素.getElementsByTagName("标签名")

c. var 类数组对象=任意父元素.getElementsByClassName("class名")

d. var 类数组对象=document.getElementsByName("name名")

(4). 按选择器查找元素: 当查找条件复杂时，2个函数

a. 只查找一个符合要求的元素：

var 一个元素对象=任意父元素.querySelector("选择器");

b. 查找多个符合要求的元素

var 类数组对象=任意父元素.querySelectorAll("选择器");

|  |
| --- |
| 总结: 查找方法的返回值:  1. 如果原函数返回下标位置i，则如果找不到，返回-1  2. 如果原函数返回一个对象或数组，则如果找不到，返回null  3. 如果原函数返回一个类数组对象，则如果找不到，返回空类数组对象{ length:0 } |

2. 修改: 3种

(1). 内容: 3种

a. 绝大多数情况下，用.innerHTML就够了！

b. 除非在获取内容时，确实希望去掉内嵌标签，只要文本时，才用textContent

c. 获取或修改表单元素的值: 表单元素.value

(2). 属性: 3种:

a. 字符串类型的HTML标准属性: 2种:

1). 旧核心DOM 4个函数

i. 元素.getAttribute("属性名")

ii. 元素.setAttribute("属性名","属性值")

iii. 元素.hasAttrbiute("属性名")

iv. 元素.removeAttribute("属性名")

2). 新HTML DOM 简写:

i. 元素.属性名

ii. 元素.属性名="属性值"

iii. 元素.属性名!==""

iv. 元素.属性名=""

b. bool类型的HTML标准属性: 只能用 "元素.属性名" 操作，~~不能用核心DOM4个函数~~

c. 自定义扩展属性: HTML5标准

1). 添加/修改自定义扩展属性:

i. HTML中: <元素 data-自定义属性名="属性值"

ii. 旧js中: ~~不能用.,~~ 只能用 元素.setAttribute("data-自定义属性名","属性值")

iii. HTML5标准: 元素.dataset.自定义属性名=属性值

2). 获取自定义扩展属性值:

i. 旧js中: ~~不能用.,~~ 只能用 元素.getAttribute("data-自定义属性名")

ii. HTML5标准: 元素.dataset.自定义属性名

3). 查找带有自定义扩展属性的元素: 属性选择器 [data-自定义属性=属性值]

(3). 样式:

a. 只修改一个css属性时: 元素.style.css属性="属性值"

b. 批量修改多个css属性时: 元素.className="class名"

c. 获取css属性值时: var style=getComputedStyle(元素对象)

style.css属性名

3. 添加元素:

(1). 只添加一个元素: 3步

a. 创建一个空元素

var 一个元素=document.createElement("元素名")

b. 为元素设置关键属性

新元素.属性名=属性值

c. 将新元素添加到指定的父元素下: 3种:

1). 末尾追加: 父元素.appendChild(新元素)

2). 插入到一个元素前: 父元素.insertBefore(新元素, 现有元素)

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

(2). 优化: 如果添加多个元素，尽量减少操作DOM树的次数

a. 如果同时添加父元素和子元素，应该先将子元素添加到父元素，最后再一次性将父元素添加到DOM树

b. 如果父元素已经在页面上，需要同时添加多个平级子元素，应该用文档片段: 3步

1). 先创建文档片段对象

var frag=document.createDocumentFragment();

2). 先将多个平级子元素加入到文档片段中

frag.appendChild(子元素)

3). 最后将文档片段添加到DOM树

父元素.appendChild(frag)

4. 删除元素: 父元素.removeChild(要删除的元素)

5. 事件:

(1). 事件绑定: 3种:

a. <元素 on事件名="事件处理函数()">

function 事件处理函数(){ ... }

b. 一个元素的一个事件属性上，只绑定一个事件处理函数，不会动态增加或减少:

元素.on事件名=function(){ ... }

c. 一个元素的一个事件上，需要同时绑定多个事件处理函数，还可能动态添加或移除

元素.addEventListener("事件名", 事件处理函数)

元素.removeEventListener("事件名", 原事件处理函数)

(2). 事件模型: 3个阶段: 捕获、目标触发、冒泡

(3). 获得事件对象: 定义事件绑定时: 元素.on事件名=function(e){ //触发事件时e=event }

(4). 取消冒泡/停止蔓延: e.stopPropagation()

(5). 只要多个平级子元素都要绑定相同的事件时: 利用冒泡/事件委托: 3步

a. 事件绑定在父元素上一份

b. 处理函数中，用e.target代替this获得实际点击的那个子元素

c. 处理函数中，先验证e.target的特征是想要的元素，才继续执行后续操作。

(6). 阻止元素自带的默认行为: e.preventDefault();

(7). 获得鼠标位置：

a. 鼠标位置相对屏幕左上角的距离: e.screenX e.screenY

b. 鼠标位置相对浏览器文档显示区左上角的距离 e.clientX e.clientY

c. 鼠标位置相对事件所在元素左上角的距离e.offsetX e.offsetY

(8). 窗口滚动事件:

a. 当滚动条滚动时自动触发: window.onscroll=function(){ ... }

b. 获取滚动条滚动过的距离:

var scrollTop=document.body.scrollTop||document.documentElement.scrollTop;

c. 控制滚动条滚动到指定位置: window.scrollTo(0, 要滚动到的位置)

总结: this 5种:

1. obj.fun() this->obj

2. new Fun() this->new正在创建的子对象

3. 类型名.prototype.共有方法=function(){ ... } this->将来调用这个共有方法的.前的子对象

4. fun() 和 (function(){ })() 和回调函数中的this->window

5. 事件处理函数中的this->当前正在触发事件的这个元素对象

今日对应小程序视频:

小程序->在线->DOM->day03 4. 打开新链接4种方式  
 小程序->在线->DOM->day04 1. 添加事件监听 移除事件监听

2. 事件模型event model

3. 利用冒泡

4. 阻止元素自带默认行为

5. 鼠标坐标 x y（拖拽效果选学）

作业: 原生DOM页面滚动（动画效果选学）

作业:

1. 复习今日小程序问题清单(小程序->在线->DOM->day03、04)，预习明日小程序问题清单(小程序->在线->jQuery->day01)

2. 看笔记中代码完成计算器效果：3\_calc.html

3. 看小程序视频学习如何通过userAgent判断浏览器的名称和版本号

小程序->在线->DOM->day03 作业: 使用navigator.userAgent鉴别浏览器名称和版本号...

4. （凡是觉得知识点串不起来）看小程序视频学习DOM版学子商城项目(持续到下周一之前)

小程序->在线->DOM->day04 扩展: DOM版学子商城...

家里环境xampp和nodejs能用，且第一阶段学的好的同学:

从小程序中网盘地址中下载xzserver\_start.zip，看视频中day01和day02，跟着做

家里环境xampp和nodejs不能用，或者第一阶段学的不好的同学：

从小程序中网盘地址中只下载public.zip，看视频中day01和day02，跟着做

然后，所有网页用live server运行

视频中所有ajax请求的服务器端接口地址，都改为我的新浪云服务器地址，就不需要你再运行服务器端了。也不需要你了解任何服务器端知识了:

/index 改为 http://xzserver.applinzi.com/index

/details 改为 http://xzserver.applinzi.com/details?lid=xxx