小程序: WEB问题速查

1. 问题清单: 小程序->首页->DOM->day04

2. 高清无码小视频: 小程序->在线->DOM->day04

今日扩展视频:

小程序->在线->DOM->day04

3. 利用冒泡 事件委托 ... 优化购物车, 标签页, 表格删除行, 。。。

5. 鼠标坐标... 原生DOM拖拽效果

所有以"作业"和"扩展"开头的视频

赠送: 原生DOM轮播图

不用今天都看完！！！！！！\*\*\*两个月后\*\*\*鄙视面试前\*\*\*看完就行\*\*\*！

3. 单词列表: 小程序->单词->DOM->day04

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 拼写错误！自己找！让我帮你找，你需要手抄20遍，给我拍照！

3. 现在正处在照猫画虎的阶段！每个DOM效果至少写3遍:

3.1 注释+代码抄一遍！

3.2 保留注释，删除代码，把代码再填回来！

3.3 删除注释和代码，试着用自己的话把注释写回来，根据自己的注释，把代码填回来

一个DOM效果，只有自己能写出人话注释，才算自己会！——标准！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻微信私信问我！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻微信私信问我！

3. 个人项目中哪个功能不会做！立刻微信私信问我！

4. 刷笔试题时，遇到不会的笔试题！立刻微信私信问我！

下载源代码: 右键->链接另存为！

正课:

1. 打开新链接4种方式

2. history

3. location

4. navigator

5. \*\*\*\*\*事件\*\*\*\*\*

一. 打开新链接四种方式:

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. 在当前窗口打开新连接, 可后退:

html: <a href="url" target="\_self">

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

2. 在当前窗口打开新连接，禁止后退:

js: location.replace("新url")

3. 在新窗口打开，可打开多个:

html: <a href="url" target="\_blank">

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

4. 在新窗口打开，只能打开一个:

html: <a href="url" target="自定义的窗口名">

js: window.open("url", "自定义的窗口名");

5. 原理:

(1). 其实在浏览器中打开的每个窗口，在内存中都有一个唯一的名字，我们默认看不到，保存在window.name属性中。

强调: 因为name是bom中有特殊意义的词，所以我们自己定义变量名或属性名时不能用name作为名称！

(2). 我们也可以通过a元素的target属性和open()的第二个实参值来修改新打开的窗口的name属性。

(3). 浏览器中规定，相同名称的窗口，只能打开一个。后打开的会把先打开的覆盖掉！

(4). 内置的预定义的窗口名:

a. \_self: 获得当前窗口的名称，赋值给新窗口，结果，因为新窗口与旧窗口同名，所以新窗口会覆盖旧窗口。产生的效果就好像是在当前窗口打开的新页面。

b. \_blank: 不指定新窗口名。但是浏览器不会让窗口名空着。为了避免重复，浏览器会自动为新窗口随机生成新窗口名。因为随机的窗口名是不重复的，所以可以打开多个窗口！

6. 示例: 以四种方式打开新链接:

day03剩余/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","abc");      }    </script>  </head>  <body>    <h3>1. 在当前窗口打开，可后退</h3>    <a href="http://tmooc.cn" target="\_self">go to tmooc</a><br/>    <button onclick="open1()">go to tmooc</button>    <h3>2. 在当前窗口打开，不可后退</h3>    <button onclick="open2()">go to tmooc</button>    <h3>3. 在新窗口打开，可打开多个</h3>    <a href="http://tmooc.cn" target="\_blank">go to tmooc</a><br/>    <button onclick="open3()">go to tmooc</button>    <h3>4. 在新窗口打开，只能打开一个</h3>    <a href="http://tmooc.cn" target="abc">go to tmooc</a><br/>    <button onclick="open4()">go to tmooc</button>  </body>  </html>  运行结果： |

对应小程序视频: 小程序->在线->DOM->day04 0. history, location, navigator

二. history:

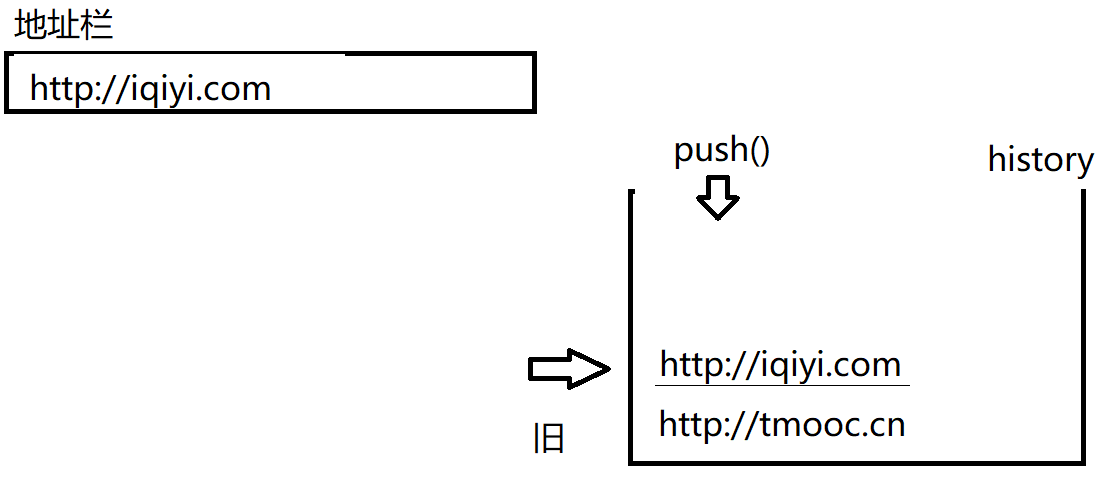
1. 什么是: 专门保存一个窗口打开后，成功访问过的所有url的历史记录数组。

2. 原理：

(1). 只要当前窗口成功访问了一个url，history就会把这个新的url追加到history数组中保存！

(2). history对象中有一个"指针"，默认指向最新的url。但是，也可以通过前进、后退来移动指针在history中的位置

(3). 如果指针当前位置下方有旧url，则可以后退；如果指针当前位置上方有新url，则可以前进。



3. 用程序控制前进后退: history.go(i)

(1). 前进一步: history.go(1)

(2). 后退一步: history.go(-1)

问题: 如果后退一步不管用，可以后退两步history.go(-2)

(3). 刷新: history.go(0)

4. 示例:用程序实现前进后退:

(1). 说明:

a. 先打开9-1\_history.html

b. 先点2，再点3，在同一窗口中先后打开1,2,3三个页面。

结果: 当前窗口的history对象中就有了三个url历史记录

c. 补全: 9-1\_history.html、9-2\_history.html、9-3\_history.html中空缺的语句。

d. 点三个页面中的前进后退链接体会前进后退。

(2). 示例: 使用history.go()在三个页面间前进后退

day03剩余/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: |

day03剩余/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>  运行结果: |

day03剩余/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: |

三. location对象:

1. 什么是: 专门保存当前地址栏中的url信息，并提供网页跳转操作的对象。

2. 属性: location对象可以分段获得url中各个部分的不同属性

(1). location.href 获得地址栏中完整的url地址

(2). location.protocol 协议

(3). location.host 主机名+端口号

(4). location.hostname 主机名

(5). location.port 端口号

(6). location.pathname 相对路径

(7). location.search 获得?及其之后的查询字符串参数部分

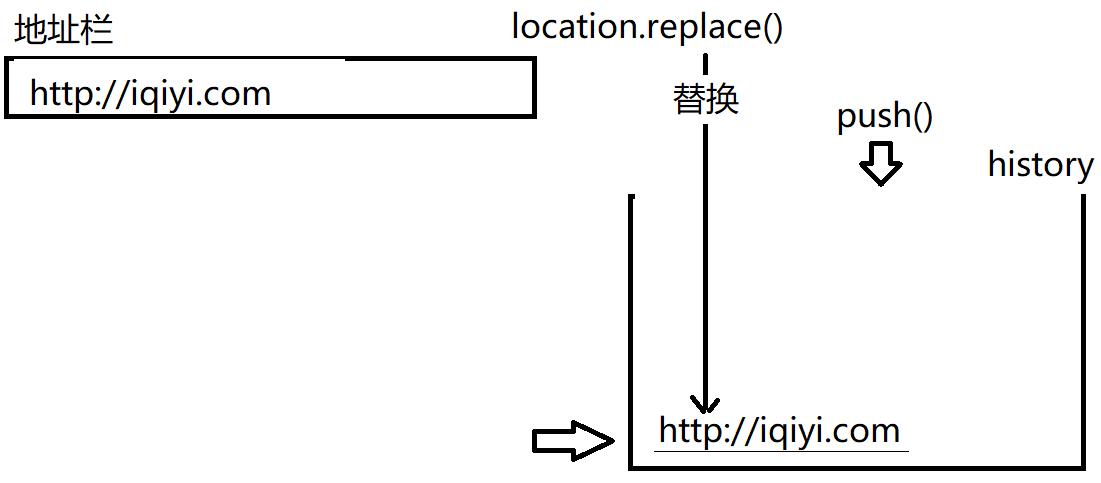
(8). location.hash 获得#及其之后的锚点地址

3. 方法:

(1). location.replace("新url"): 专门实现在当前窗口打开新连接，并且禁止后退！

a. 在当前窗口打开新url

b. 将新url以替换的方式加入history对象中，替换旧url。因为旧url被替换，所以没有后退的余地了！



(2). location.assign("新url"): 也能实现在当前窗口打开新连接，可以后退。等效于window.open("新url","\_self");

(3). location.reload(): 刷新 等效于history.go(0);

4. 示例: 分别获得当前url的各个部分

day03剩余/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);      console.log(location.search);      console.log(location.hash);    </script>  </body>  </html>  运行结果: |

四. navigator

1. 什么是: 专门保存浏览器配置信息的对象

2. 问题: navigator很多属性兼容性极差，已经很少使用了！

3. 判断浏览器是否安装了某个插件:

(1). 浏览器中安装的所有插件的信息都集中保存在navigator.plugins集合中。

(2). 如何判断浏览器是否安装某个插件？通过插件全名，强行访问

navigator.plugins["完整插件名称"]!==undefined 说明装了该插件

(3). 示例: 判断浏览器是否安装了某个插件:

day03剩余/11\_navigator.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>navigator对象常用属性</title>  <meta charset="utf-8" />  </head>  <body>    <script>      //判断当前浏览器是否安装了Chrome PDF Viewer插件      if(navigator.plugins["Chrome PDF Viewer"]!==undefined){        document.write(`<h3 style="color:green">已安装PDF插件，可浏览电子书</h3>`)      }else{        document.write(`<h3 style="color:red">未安装PDF插件，<a href="javascript:;">点此下载安装</a></h3>`)      }      //判断当前浏览器是否安装了Shockwave Flash插件      if(navigator.plugins["Shockwave Flash"]!==undefined){        document.write(`<h3 style="color:green">已安装Flash插件，可浏览电子书</h3>`)      }else{        document.write(`<h3 style="color:red">未安装Flash插件，<a href="javascript:;">点此下载安装</a></h3>`)      }      document.write(navigator.userAgent);    </script>  </body>  </html>  运行结果: |

4. 获取浏览器的名称和版本号: navigator.userAgent

五. \*\*\*\*\*事件\*\*\*\*\*

1. 复习: 事件简介:

(1). 什么是事件: 用户手动触发的或浏览器自动触发的页面内容和状态的改变

(2). 事件属性: 每个元素身上都有一批"on事件名"的属性。每当当前元素身上发生一个事件时，浏览器都会自动找到这个元素身上的对应的"on事件名"属性。自动触发该属性上提前保存的事件处理函数。

(3). 事件绑定: 提前将要执行的函数，保存到一个元素的一个事件属性上。

(4). 如何: 元素.on事件名=function(){ ... this ... }

2. 绑定事件: 3种:

对应小程序视频：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:

a. js中: 提前定义function 函数名(){ ... }

b. html中: <元素 on事件名="函数名()">

c. 问题: 不符合内容与行为分离的原则！及其不便于维护！

(2). 在js中用赋值方式绑定:

a. js中: 元素.on事件名=function(){ ... }

b. 优点: 所有事件绑定都集中定义在js中，极其便于维护！

c. 问题: 一个元素的一个事件属性上，只能绑定一个处理函数。无法同时绑定多个事件处理函数——不灵活！

(3). 在js中通过添加事件监听对象的方式来绑定:

a. js中: 元素.addEventListener("事件名", 事件处理函数)

添加 事件 监听对象

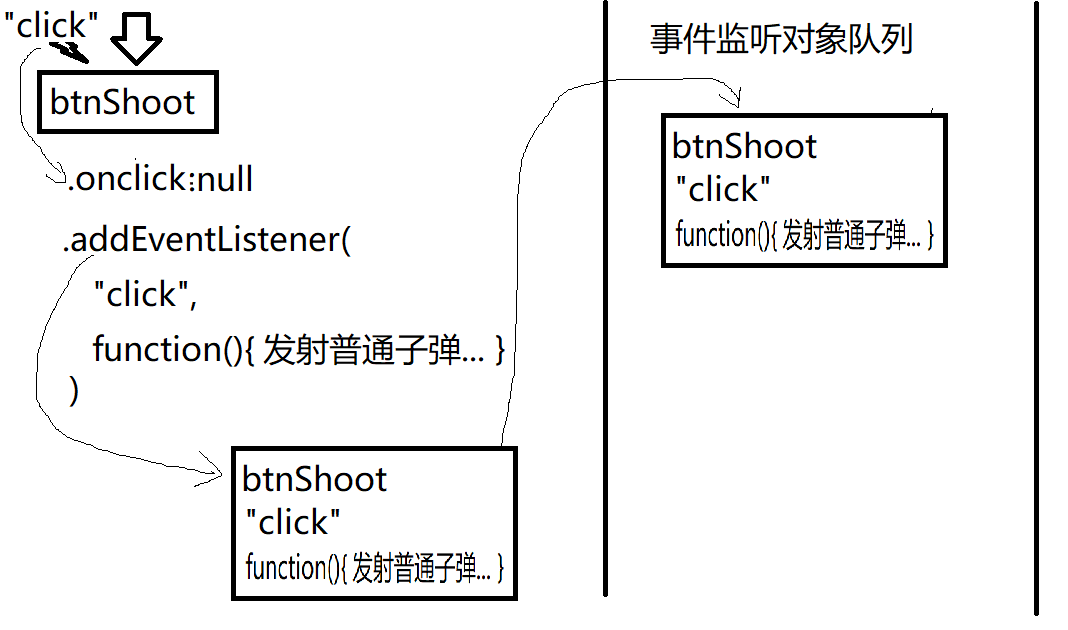
b. 意为: 创建一个事件监听对象，其中包括3样东西:

1). 哪个元素

2). 哪个事件

3). 事件处理函数是什么

然后，浏览器会自动将事件监听对象加入到浏览器内存中一个巨大的事件监听队列中

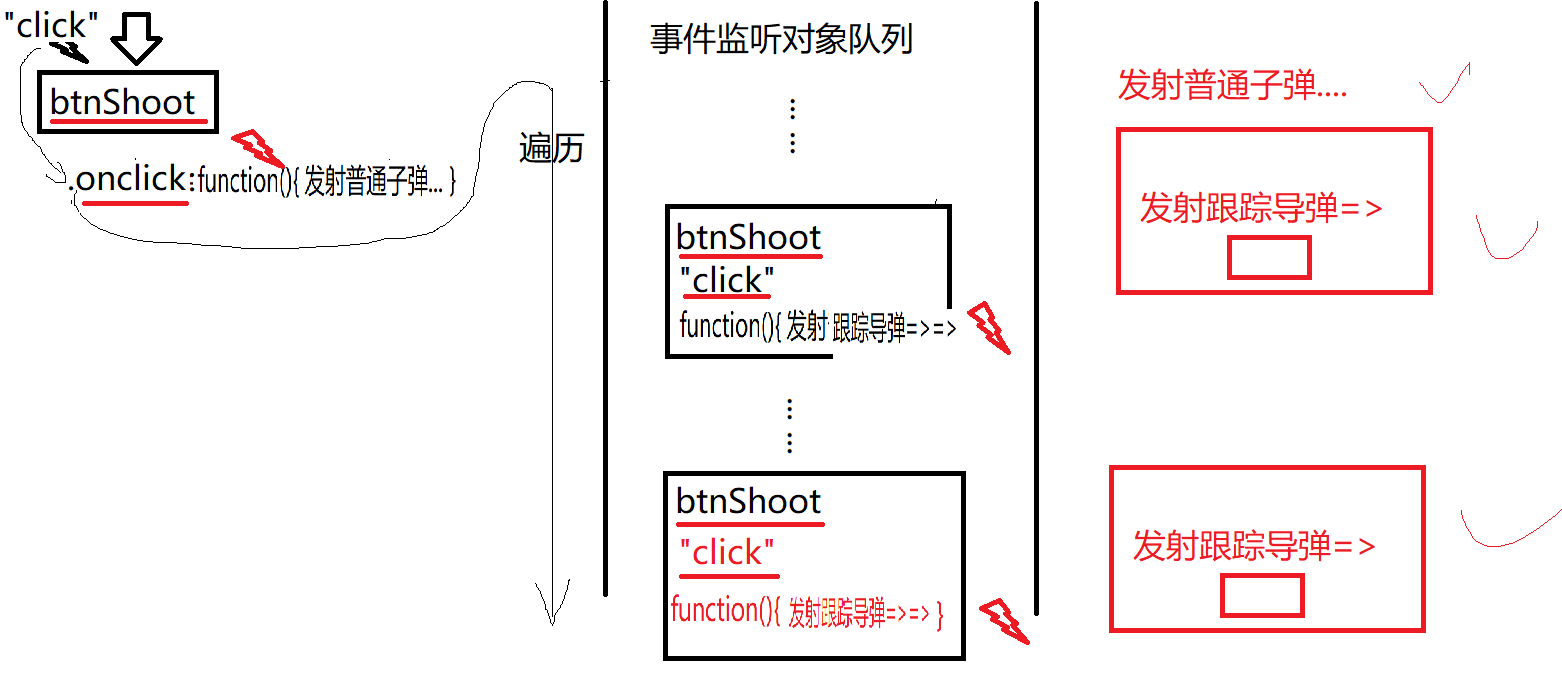


c. 结果:

1). 每当元素上发生事件时，浏览器都会自动遍历事件监听对象队列。

2). 查找和本次发生事件的元素、事件名相同的监听对象。

3). 只要找到的符合要求的监听对象，就自动触发监听对象中保存的事件处理函数



d. 强调:

1). 其实在DOM标准中的事件名称都不带on开头:

比如： click dblclick mouseover blur ...

所以. addEventListener("事件名")也不需要加on前缀

2). 但是元素对象身上的事件属性都带on

为了和元素身上其它普通的属性区分

e. 移除事件监听对象:

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

移除 事件 监听对象

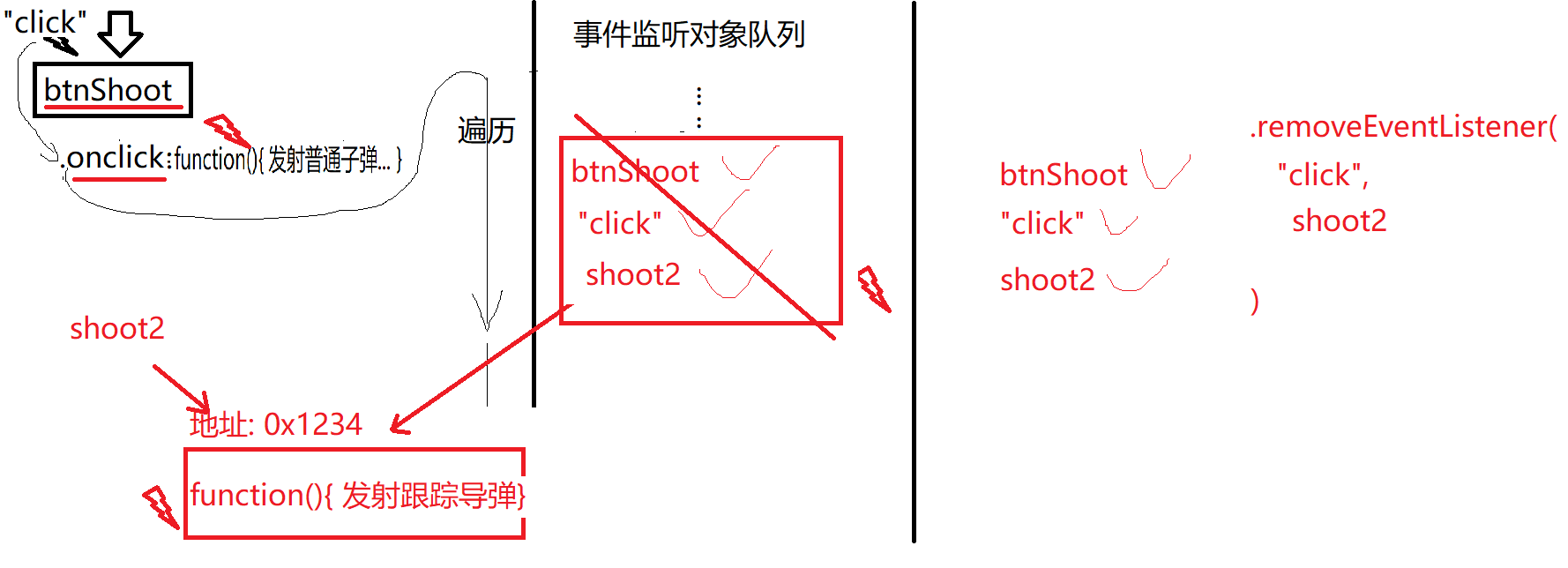
2). 坑: 移除事件监听的原理:

i. 浏览器会去事件监听对象队列中查找要移除的事件监听对象:

ii. 如果发现一个事件监听对象的元素、事件名、事件处理函数的地址与removeEventListener()提供的参数值一致，就会移除该事件监听。

iii. 但是, 如果绑定事件处理函数时，使用的是匿名函数绑定的！一旦绑定之后，旧再不可能找到原函数对象了！移除时，也就不可能用原函数对象来移除原事件监听。

3). 解决: 今后如果一个事件监听对象有可能被移除，则不要用匿名函数绑定！应该用有名称的函数绑定。



4). 坑: 一旦添加事件监听时，使用有名称的函数来添加事件监听，则不允许重复添加一模一样的监听对象!

5). 所以:

i. 如果使用匿名函数添加事件监听:

好处: 可反复添加多个; 不好: 无法移除

ii. 如果使用有名称的函数添加事件监听:

好处: 可移除; 不好: 只能加一个

6). 解决: jQuery

（基础好的同学可以看小程序中视频DOM如何解决）

f. 示例: 添加移除事件监听对象:

1\_addEventListener.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>     <meta charset="utf-8" />     <title>...</title>     <script>     </script>  </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");        //给btnShoot绑定单击事件处理函数，可发射普通子弹        btnShoot.onclick=function(){           console.log(`发射普通子弹......`)        }        //先定义一个发射跟踪导弹的有名称的函数        function shoot2(){           alert("发射跟踪导弹=>=>=>")        }        //当单击btnAward时，不是发生导弹，而是给btnShoot按钮再多绑定一种跟踪导弹        btnAward.onclick=function(){           //不能再用onclick=function(){ ... }           // btnShoot.onclick=function(){           //    alert("发射跟踪导弹=>=>=>");           // }           //想浏览器的事件监听对象的队列中添加了一个新的监听对象，元素:btnShoot, 事件名:click。           btnShoot.addEventListener(              "click",              shoot2           )        }        //单击btnBreak时，从btnShoot上移除发射跟踪导弹的事件监听对象        btnBreak.onclick=function(){           btnShoot.removeEventListener(              "click",              shoot2           )        }     </script>  </body>  </html>  运行结果: |

3. 事件模型: 高频鄙视题

对应小程序视频: 小程序->在线->DOM->day04 2. 事件模型

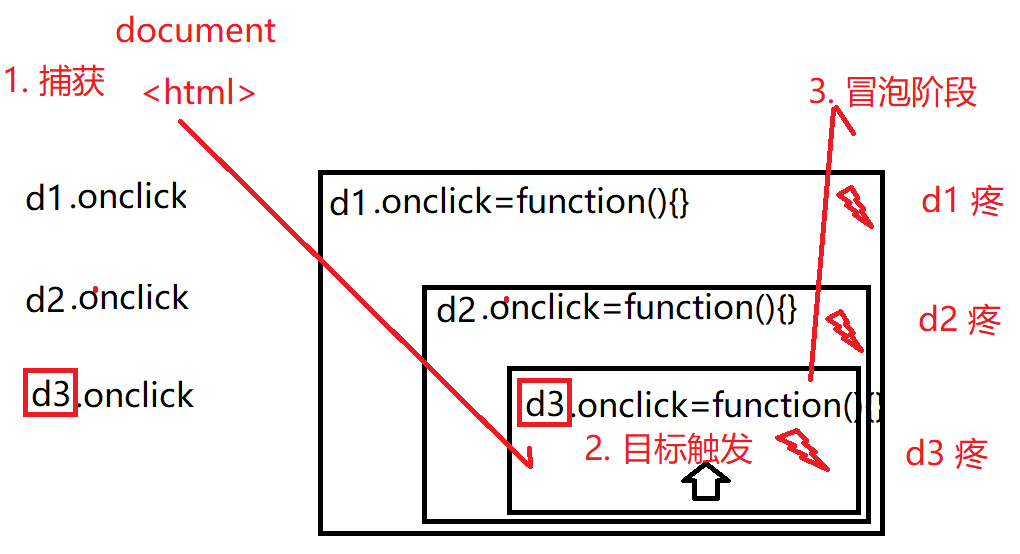
(1). 什么是: 从刚触发事件，到所有事件处理函数执行完，所经历的完整过程。

(2). DOM标准中的事件模型:3个阶段

a. 捕获: 浏览器会从顶级父元素document->html->...开始由外向内依次遍历当前点击的元素的各级父元素。在遍历过程中，记录哪些父元素上绑定了相同的事件处理函数。——只记录不执行！

b. 目标触发: 浏览器总是优先触发实际点击的这个元素上的事件处理函数。实际点击的元素，也被称为"目标元素"(target)

c. 冒泡: 由内向外，依次触发各级父元素上的事件处理函数



4. 事件对象:

(1). 什么是: 专门保存事件发生时的信息，并提供了修改属性默认行为的方法，的对象。

(2). 何时: 2个

a. 想获得事件发生时的信息时

b. 想改变事件默认的执行过程时

(3). 如何获取:

a. 事件对象是浏览器在事件发生时自动创建的，所以不用我们自己创建

b. 所有事件对象都默认作为事件处理函数的第一个实参值自动传入

c. 所以，要想接住事件对象，只要在事件处理函数中定义第一个形参变量等着即可！

浏览器自动创建event事件对象

↓

元素.onclick=function(e){

... ...

}

或

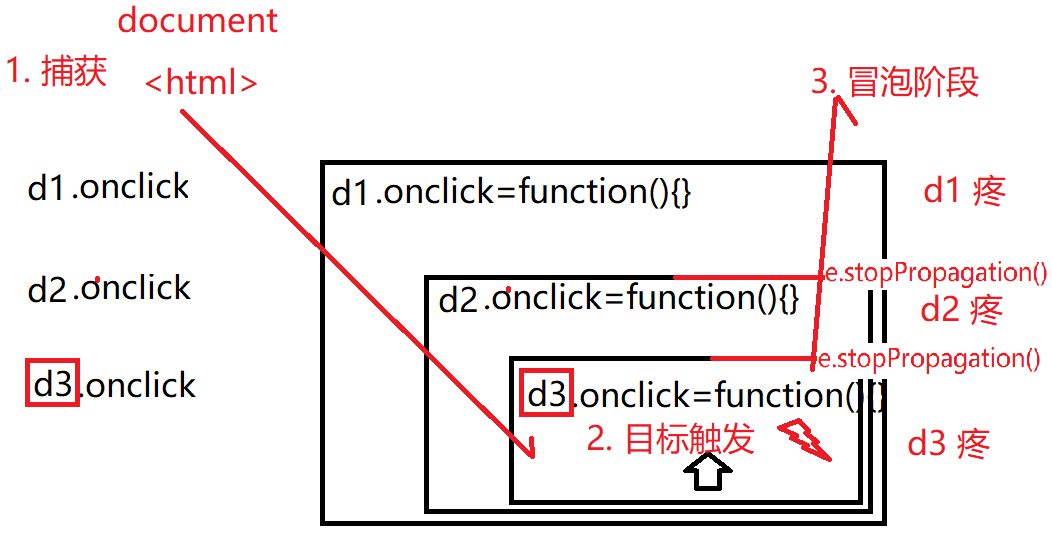
元素.addEventListener("click",function(e){ ... })

(4). 接住事件对象后能干什么:

a. 取消冒泡/停止漫延:

1). e.stopPropagation()

停止 漫延



2). 说明：因为e.stopPropagation()不拦自己，只拦别人，所以写前写后与自己无关！自己的代码都会执行！

3). 示例: 演示冒泡和阻止冒泡:

2\_bubble.html

|  |
| --- |
| <!DOCTYPE HTML>  <html>    <head>      <title>事件处理</title>      <meta charset="utf-8"/>      <link rel="stylesheet" href="css/2.css"/>      </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");        //希望点哪个div，就只有哪个div喊疼！        d1.onclick=function(){          alert(`d1 疼!`)        }        d2.onclick=function(e){          e.stopPropagation();          alert(`d2 疼!`);        }        d3.onclick=function(e){          //停止事件继续向上层冒泡          e.stopPropagation();          alert(`d3 疼!`);        }      </script>    </body>  </html>  运行结果: |

b. 事件委托/利用冒泡:

1). 问题: 如果页面中绑定太多事件监听，会导致监听队列中监听对象很多，遍历查找就会慢，就会影响事件响应的速度。

2). 解决: 优化: 今后，尽量减少绑定事件监听的个数

3). 利用冒泡/事件委托: 3步

a. 今后只要多个平级子元素都要绑定相同的事件处理函数时，其实只要在父元素上绑定一个处理函数，所有子元素就都可以通过冒泡机制共用父元素上的事件处理函数！

问题: 因为事件绑定在父元素上，所以事件处理函数中的this不再指当前点的按钮，而是指事件绑定时.前的父元素div

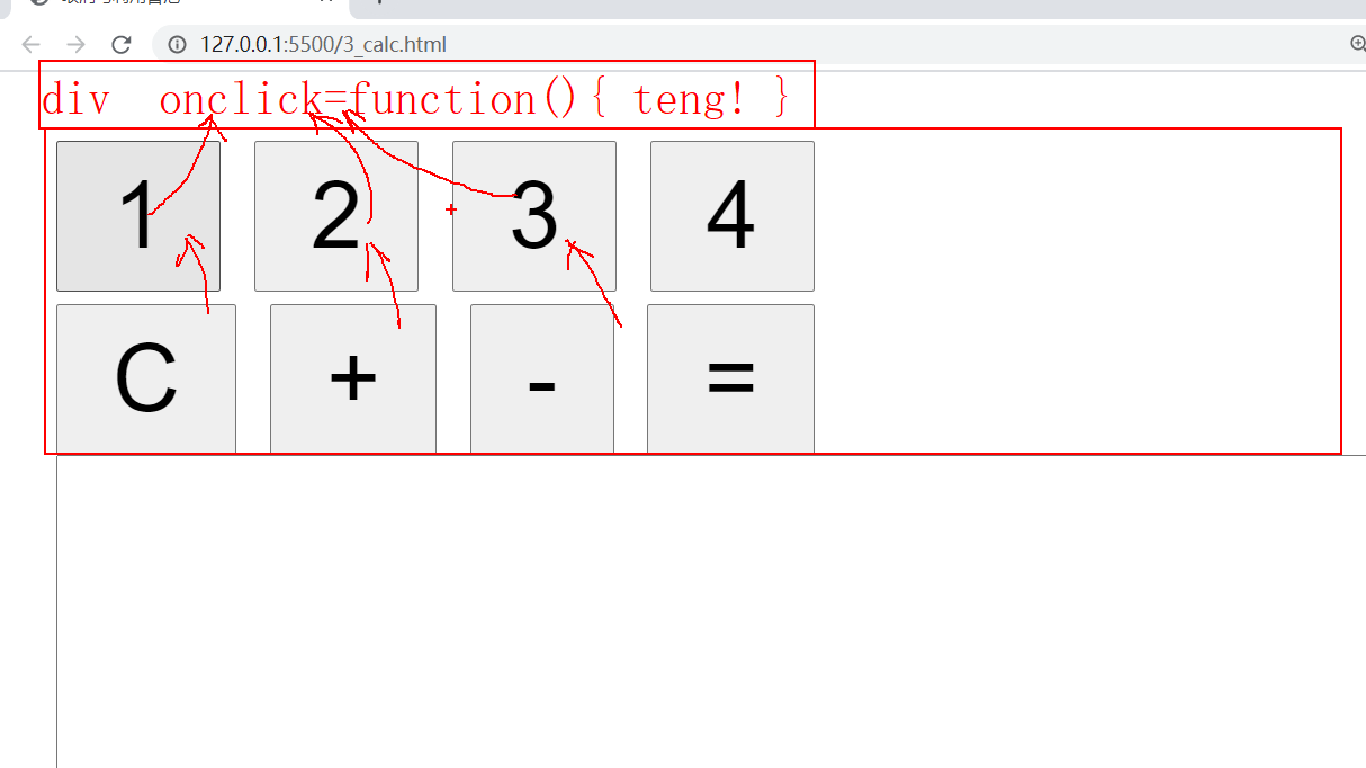
b. 今后只要利用冒泡，都要用e.target代替this获得实际点击的目标元素。

1). e.target 专门保存最初实际点击的那个目标元素，不随冒泡而改变。

2). 问题: 即使点在父元素上，也会触发父元素的事件处理函数——是我们不希望的.

c. 先判断当前点击的元素是不是我们想要的。只有当前点击的元素是我们想要的，才执行后续事件处理操作。

比如: 判断元素的标签名、className......



4). 示例: 使用事件委托实现计算器效果:

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>      <button>2</button>      <button>3</button>      <button>4</button><br>      <button>C</button>      <button>+</button>      <button>-</button>      <button>=</button>    </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: 因为多个平级的button都可以单击，所以，事件应该只绑定在他们共同的父元素上一份即可！      var div = document.getElementById("keys")      //2. 绑定事件处理函数      div.onclick = function (e) {        // alert("div 疼！");        //点哪个按钮，让哪个按钮变小花        //this->div        //this.innerHTML="❀";        //e.target代替this        //只有当前点击的目标元素是button时，才继续执行后续操作        if (e.target.nodeName === "BUTTON") {          //e.target.innerHTML = "❀"          //作业: 看注释实现完整计算器效果。          //3. 查找要修改的元素          //本例中:点按钮，要修改的是充当显示屏的id为sc的元素          var sc = document.getElementById("sc");          //4. 修改元素          //本例中: 根据本次点的按钮不同，执行不同的操作          //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>  运行结果： |

c. 阻止默认行为:

1). 问题: HTML中有些元素，自身带有一些自动执行的默认的行为:

比如: 点击<a href="#top">，会自动在地址栏中结尾加"#top"

但是有些默认行为使我们不想要的

2). 解决: e.preventDefault();

阻止 默认

3). 示例: 阻止a元素的默认行为:

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="#top">click me</a>    <script>      //点a时不要擅自修改地址栏结尾      var a1=document.getElementById("a1");      a1.onclick=function(e){        e.preventDefault();          //阻止  默认行为        alert(`做我规定的事情...`);      }    </script>  </body>  </html>  运行结果: |

d. 获得鼠标位置: 3组

1). 鼠标相对于整个屏幕左上角的距离

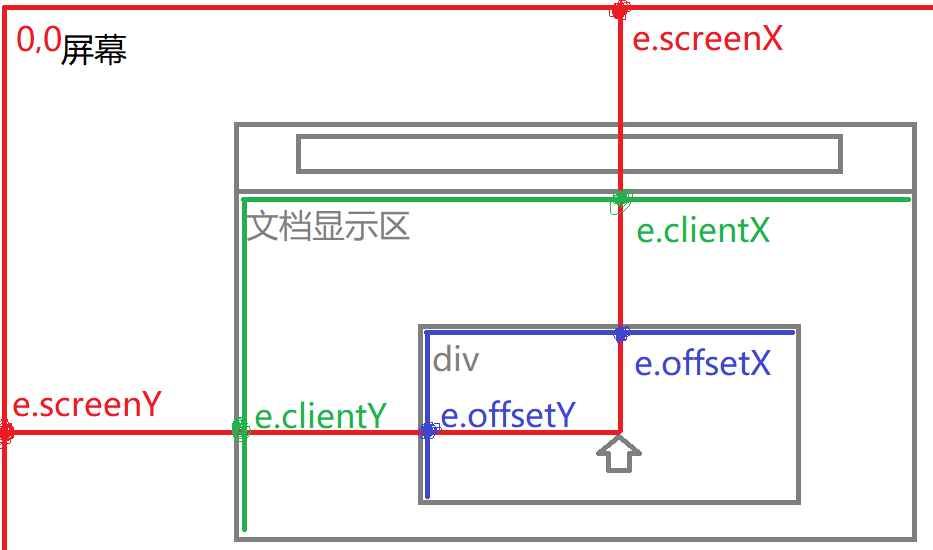
e.screenX e.screenY

2). 鼠标相对于文档显示区左上角的距离

e.clientX e.clientY

3). 鼠标相对于当前事件所在元素左上角的距离

e.offsetX e.offsetY



4). 示例: 点击元素获得鼠标位置:

5\_pop1.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;        width:100px;        height:100px;        top:50px;        left: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(e.screenX, e.screenY);        console.log(e.clientX, e.clientY);        console.log(e.offsetX, e.offsetY);      }    </script>   </body>  </html>  运行结果: |

补: 元素.nodeName:

1. 专门保存一个元素的标签名

2. 坑: 它保存的标签名是全大写字母的: DIV、BUTTON、INPUT...

总结: 不要背英文名字！反而应该记中文能做哪些事儿！

3. 添加/删除元素:

(1). 只添加一个新元素: 3步

a. 创建一个新元素:

var 新元素=document.createElement("标签名")

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DOM树: 3种:

1). 末尾追加:

父元素.appendChild(新元素)

2). 在某个元素前插入:

父元素.insertBefore(新元素, 现有元素)

3). 替换某个元素:

父元素.replaceChild(新元素, 现有元素)

(2). 优化: 尽量减少操作DOM树的次数，2种:

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，最后再将父元素一次性添加到DOM树

b. 如果父元素已经在页面上，要添加多个平级子元素。应该利用文档片段对象

1). 创建文档片段对象:

var frag=document.createDocumentFragment()

2). 将子元素添加到文档片段对象中:

frag.appendChild(子元素)

3). 最后将文档片段对象一次性添加到DOM树上父元素下  
 父元素.appendChild(frag);

(3). 删除元素: 父元素.removeChild(子元素)

4. HTML DOM常用对象:（了解即可）

(1). var img=new Image()

(2). table  
a. table管着行分组:

1). 添加行分组:

var thead=table.createTHead()

var tbody=table.createTBody()

var tfoot=table.createTFoot()

2) 删除行分组:

table.deleteTHead(); table.deleteTFoot()

3). 获取行分组:

table.tHead table. tFoot table.tBodies[i]

b. 行分组管着行:

1). 添加行:

i. 任意行插入新行: var tr=行分组.insertRow(i);

ii. 开头插入新行: var tr=行分组.insertRow(0)

iii. 末尾追加新行: var tr=行分组.insertRow()

2). 删除行: table.deleteRow(tr.rowIndex)

3). 获取行: 行分组.rows[i]

c. 行管着格:

1). 添加格: var td=tr.insertCell()

2). 删除格: tr.deleteCell(i)

3). 获取格: tr.cells[i]

(3). form:

a. 获取form元素: document.forms[i]

b. 获取form中的表单元素:

1). 标准: form.elements[i或id或name名]

2). 简写: 如果有name属性: form.name名

c. 让表单元素自动获得焦点: 表单元素.focus()

总结:

BOM:

1. window:

(1). 获得窗口大小:

a. 获得完整窗口大小:

window.outerWidth和window.outerHeight

b. 获得文档显示区大小:

window.innerWidth和window.innerHeight

(2). 打开和关闭窗口:

window.open()和window.close()

2. 打开新链接4种方式:

(1). 在当前窗口打开，可后退

a. html: <a href="url" target="\_self">

b. js: window.open("url", "\_self");

(2). 在当前窗口打开，禁止后退

a. js: location.replace("新url")

(3). 在新窗口打开，可同时打开多个

a. html: <a href="url" target="\_blank">

b. js: window.open("url", "\_blank");

(4). 在新窗口打开，只能打开一个

a. html: <a href="url" target="自定义窗口名">

b. js: window.open("url", "自定义窗口名")

3. history:

(1). 前进: history.go(n)

(2). 后退: history.go(-n)

(3). 刷新: history.go(0)

4. location:

(1). 属性: 分段获得url中各个部分:

a. location.href 完整url

b. location.protocol 协议

c. location.host 主机名+端口号

d. location.hostname 主机名

e. location.port 端口号

f. location.pathname 相对路径

g. location.search ?及其之后的查询字符串参数列表

h. location.hash #锚点地址

(2). 方法:

a. 在当前窗口打开，可后退：

location.assign("新url") 或 location.href="新url"

b. 在当前窗口打开，禁止后退:

location.replace("新url")

c. 刷新: location.reload();

5. navigator

(1). 查看浏览器的名称和版本号: navigator.userAgent

(2). 查看浏览器中安装的插件列表: navigator.plugins

总结: 事件:

1. 绑定事件: js中:

(1). 一个事件只绑定一个处理函数

元素.on事件名=function(){ ... }

(2). 一个事件绑定多个处理函数

元素.addEventListener("事件名", 事件处理函数)

(3). 移除一个事件监听:

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

2. 事件模型: 捕获，目标触发，冒泡

3. 事件对象:

(1). 获得事件对象:

元素.on事件名=function(e){ ... }

(2). 阻止冒泡: e.stopPropagation()

(3). 当多个子元素都要绑定相同事件时，利用冒泡/事件委托3步:

a. 事件只在父元素上绑定一次

b. e.target代替this

c. 判断e.target的任意特征是否是我们想要的元素

(4). 阻止元素默认行为:

e.preventDefault()

(5). 获取鼠标位置:

a. 相对于屏幕左上角的x，y坐标:

e.screenX, e.screenY

b. 相对于文档显示区左上角的x，y用坐标:

e.clientX, e.clientY

c. 相对于事件所在元素左上角的x，y坐标:

e.offsetX e.offsetY

今日对应小程序视频列表:

小程序->在线->DOM->day03 4. 打开新链接4种方式

小程序->在线->DOM->day04 0. history, location, navigator

1. 添加事件监听，移除事件监听 ...

2. 事件模型

3. 利用冒泡 事件委托

4. 阻止默认行为

5. 鼠标坐标x y 原生DOM

作业:

1. 复习今日问题清单: 小程序->首页->DOM->day03、day04

2. 看小程序视频，实现完整计算器效果:

小程序->在线->DOM->day04 3. 利用冒泡 事件委托 delegate 计算器...

3. 看小程序串讲视频，总结DOM所有知识点:

小程序->在线->DOM->day04 总结. 1小时DOM知识点串讲

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