小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->DOM->day03、day04

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->DOM->day03、day04

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频:

小程序->在线->DOM->day03

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 计算器，优化购物车shoppingcart, 标签页tabs

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window.onscroll

作业: 高频笔试题 nodejs 浏览器不同 事件循环 ...

扩展: 笔试题: 4x4格子 点击次数

赠送: 原生DOM轮播图

3. 单词列表: 小程序->单词->DOM->day03、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. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问: （vip待遇的剩余时间8天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 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. BOM

2. \*\*\*事件绑定\*\*\*

一. BOM: Browser Object Model

1. 什么是: 一套专门操作浏览器窗口和软件信息的对象、属性和方法的集合。

2. vs DOM: DOM专门操作网页内容； BOM专门操作浏览器窗口和软件

3. 何时: 只要想操作浏览器窗口或想获得浏览器窗口和软件的信息时，都用BOM

4. 问题: 没有标准的！每种浏览器可能都不一样！——极大的兼容性问题！——用的越来越少了！

5. 包括:

window, location, history, navigator, event, document, screen

6. window对象: 3个角色:

(1). 代替ES标准中的global，充当全局作用域对象

(2). 集中包含所有原生的对象和函数: ES标准的内置对象+DOM+BOM

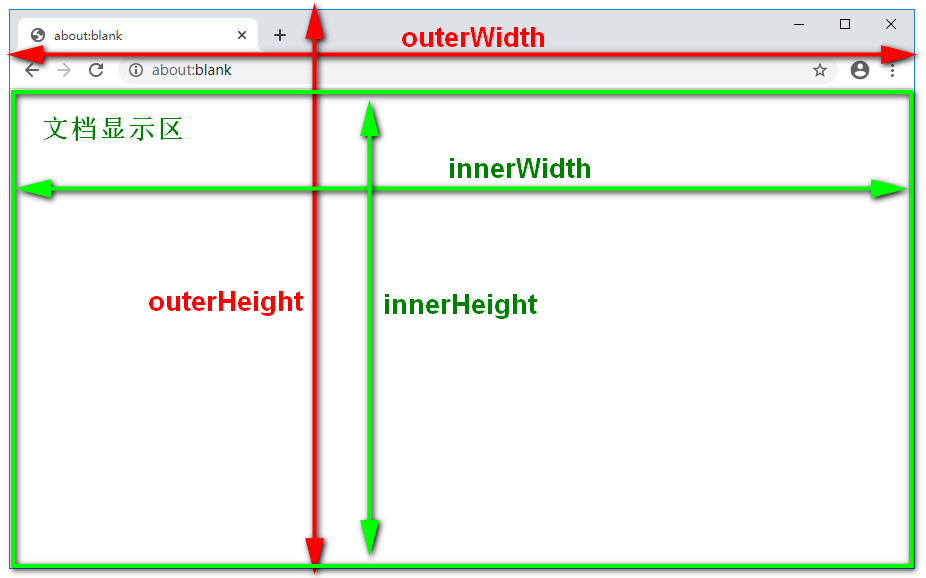
(3). 代表当前浏览器窗口:

a. window可获得当前浏览器窗口的大小

1). 完整窗口大小: window.outerWidth window.outerHeight

2). 仅内部文档显示区的大小: window.innerWidth window.innerHeight

文档显示区: 浏览器窗口中，专门用于显示网页的区域



b. 用window还可打开和关闭窗口

1). 打开一个新窗口: window.open()

2). 关闭当前窗口: window.close()

7. 打开和关闭新链接: 4种情况:

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，可后退

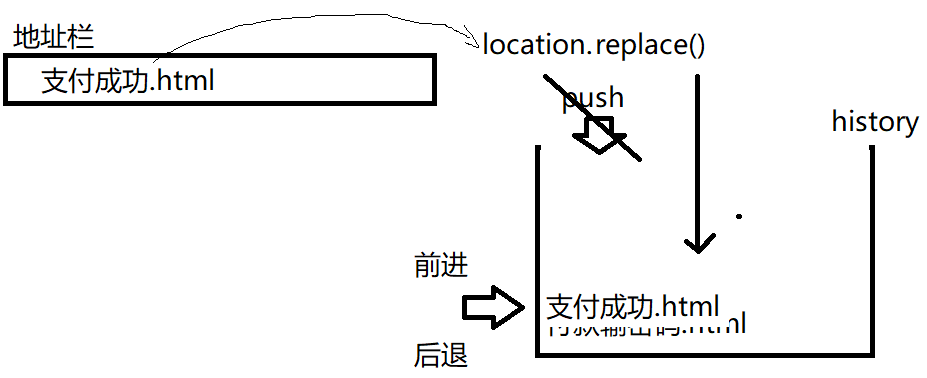
a. html: <a href="url" **target="\_self"**>

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

(2). 在当前窗口打开，禁止后退

a. js: **location.replace("新url")**

b. 原理: 用新url代替history中旧url，因为没有旧url了，所以无法后退！



(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")**

(5). 原理:

a. 每个打开的浏览器窗口在内存中都有一个name名来唯一表示这个窗口对象——一般用户看不见！

b. 浏览器规定，相同name名的窗口只能打开一个。后打开的同name名的窗口会覆盖先打开的同name名的窗口。

c. 在我们用a元素或window.open()打开新窗口时，target属性值和open()的第二个参数其实都是在为新窗口指定一个自定义的窗口名

d. 强调: 今后自己起变量名或属性名时，**禁止使用"name"作为变量名或属性名！**

e. 预定义窗口名:

1). \_self: 表示将当前旧窗口自己的name名，设置给新打开窗口的name名——新窗口覆盖/替换同名旧窗口——在当前窗口打开

2). \_blank: 表示不给新窗口指定name名，但是浏览器不会让name名空着，而会在底层自动为新窗口随机生成name名！因为随机生成，所以每个新窗口的name名各不相同！——同时打开多个而不冲突！

(6). 示例: 打开新链接4种方式:

day03/4\_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，禁止后退        3. 在新窗口打开，可同时打开多个        4. 在新窗口打开，只能打开一个        \*/      </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 id="btn1">go to tmooc</button>      <h3>2. 在当前窗口打开，禁止后退</h3>      <button id="btn2">go to 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">go to tmooc</a><br/>      <button id="btn3">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 id="btn4">go to tmooc</button>      <script>        var btn1=document.getElementById("btn1");        btn1.onclick=function(){          //window.open("http://tmooc.cn","\_self");          //location.assign("http://tmooc.cn");          location.href="http://tmooc.cn";        }        var btn2=document.getElementById("btn2");        btn2.onclick=function(){          location.replace("http://tmooc.cn");        }        var btn3=document.getElementById("btn3");        btn3.onclick=function(){          window.open("http://tmooc.cn","\_blank");        }        var btn4=document.getElementById("btn4");        btn4.onclick=function(){          window.open("http://tmooc.cn","abc");        }      </script>    </body>  </html>  运行结果: |

8. history:

(1). 什么是: 每个窗口中自带的记录当前窗口打开后成功访问过的所有url的历史记录数组。

(2). 原理:

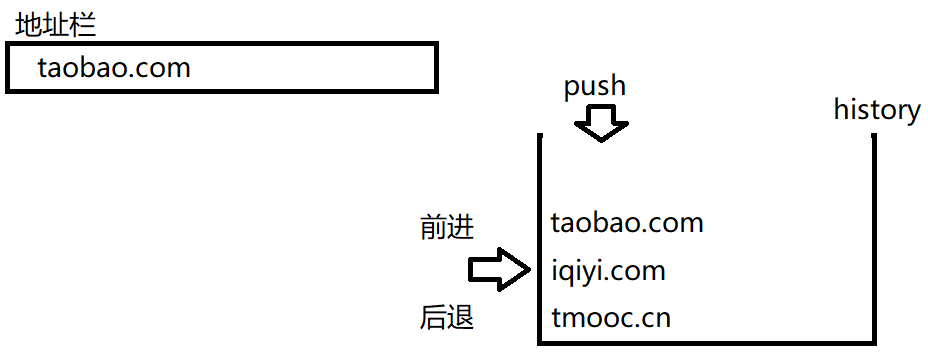
a. 首次打开一个url时，将url压入(push)history数组中保存

b. 之后每打开一个新url时，都会再次将新url压入history中上层保存

(3). 前进后退的原理:

a. 能否前进取决于当前正在看的页面的url在history中前边有没有比它新的url

b. 能否后退取决于当前正在看的页面的url在history中后边有没有比它旧的url



(5). history只能前进，后退，刷新：

a. 前进: history.go(1) 前进一步

b. 后退: history.go(-1) 后退一步

history.go(-2)

c. 刷新: history.go(0)

(6). 示例: 演示前进后退  
 a. 如何:

1). 先从9-1\_history.html运行起来

—— 此时history中只有一个url: 9-1 , 既不能前进又不能后退

2). 再点击页面中的2、3两个链接，让history中增加两个url

—— 此时history中有了三个url: 9-1 9-2 9-3 就可以验证前进后退

3). 再测试前进后退！

b. 代码:

day03/9-1\_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元素的href属性以javascript:开头，就不再执行跳转操作，而是执行一条js语句-->      <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> |

day03/9-2\_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> |

day03/9-3\_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: |

9. location:

(1). 什么是: 专门保存当前窗口正在打开的url信息的对象

(2). 何时: 要获得url中相关信息时，或者希望执行跳转操作时

(3). 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

(4). 示例: 输出location的各个部分:

day03/10\_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: |

(5). 方法:

a. 也可以实现在当前窗口打开，可后退：

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

b. 只有location可以实现在当前窗口打开，禁止后退:

**location.replace("新url")**

c. 也可以实现刷新: **location.reload();**

10. navigator:

(1). 什么是: 保存浏览器配置信息的对象

(2). 何时: 今后只要想获得浏览器中的配置信息，都用navigator对象

(3). 常用:

a. 查看浏览器的名称和版本号: **navigator.userAgent**

b. 查看浏览器中安装的插件列表: **navigator.plugins**

二. 事件:

1. 事件绑定: 3种:

(1). 在HTML中绑定:

a. html中: <元素 on事件名="事件处理函数()">

b. js中: function 事件处理函数(){ ... }

c. 问题: 因为事件绑定随元素分散在网页的各个角落，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. 好处: 所有事件绑定都集中在js中，非常便于维护

c. 问题: 一个元素的一个事件属性上，只能保留一个事件处理函数。无法同时保留多个事件处理函数。重复给一个元素的一个属性上赋值多个事件处理函数，结果只有最后一个事件处理函数才能留下来！

(3). 在js中用添加事件监听对象的方式绑定:

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

添加 事件 监听对象

b. 强调: 如果使用addEventListener，事件名之前不用加"on"。因为原本DOM标准中规定的事件名都是没有on的！比如: click, focus, change, ...

c. 原理:

1). 在浏览器内存中有一个巨大的事件队列

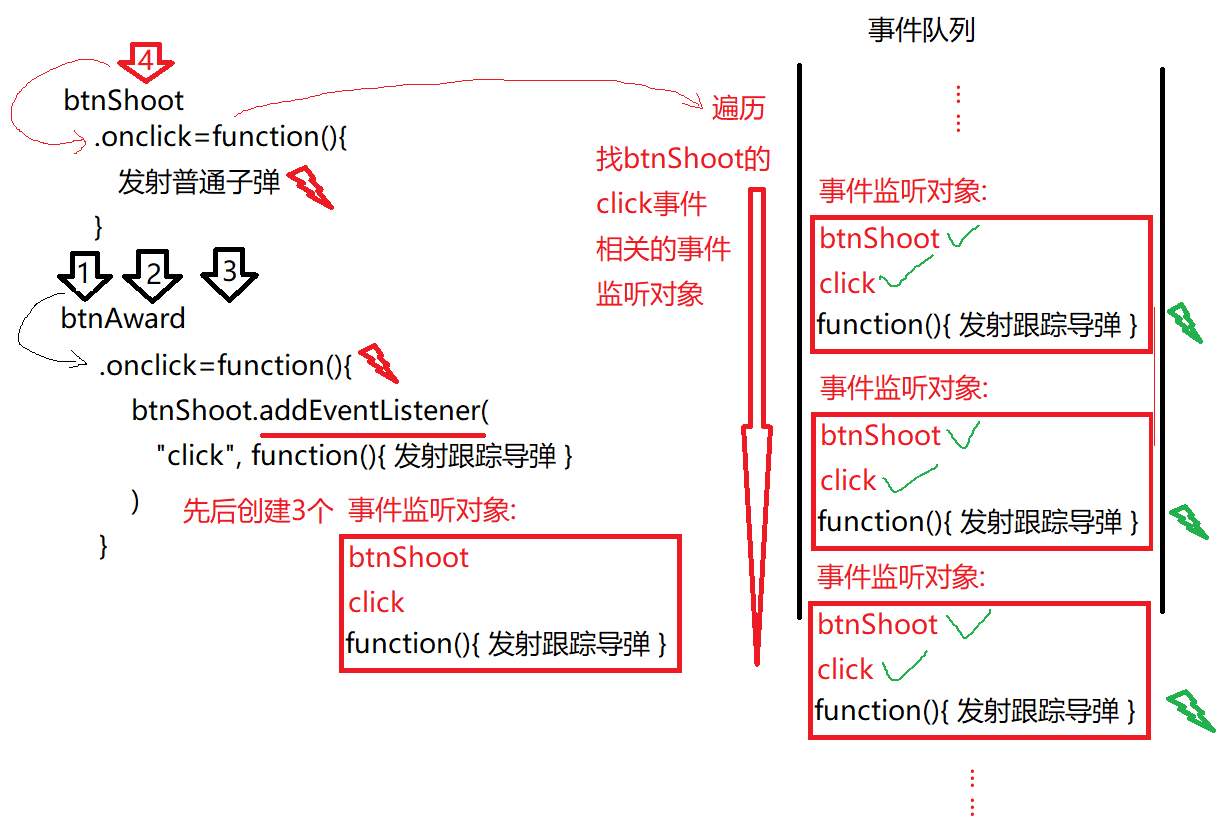
2). addEventListener() 会做2件事:

i. 创建一个事件监听对象，其中保存三个内容:

当前元素+当前事件名+事件处理函数对象

ii. 将事件监听对象添加到浏览器的事件队列中保存起来

3). 当某个元素上发生某个事件时，浏览器会通过遍历事件队列的方式查找到符合条件的事件监听对象，自动执行其中的事件处理函数。找到几个事件监听对象，就执行几个事件处理函数.

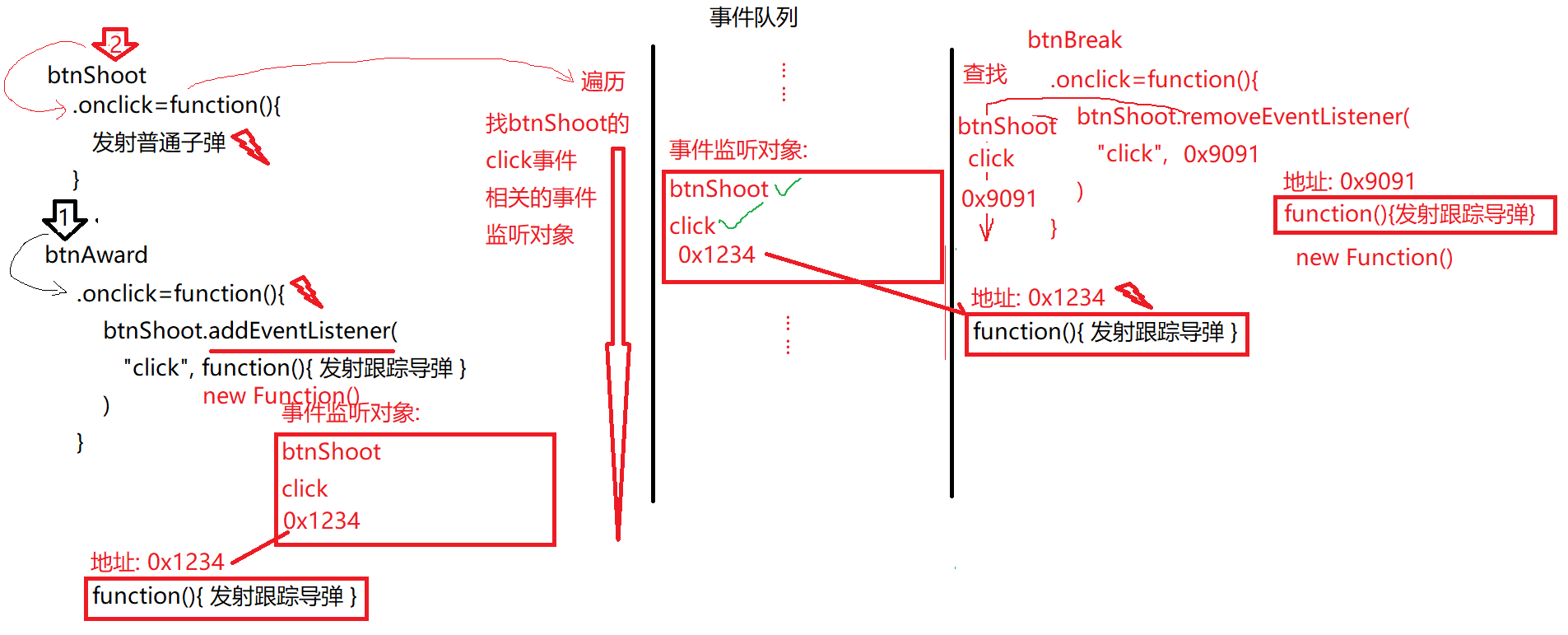


d. 移除事件监听:

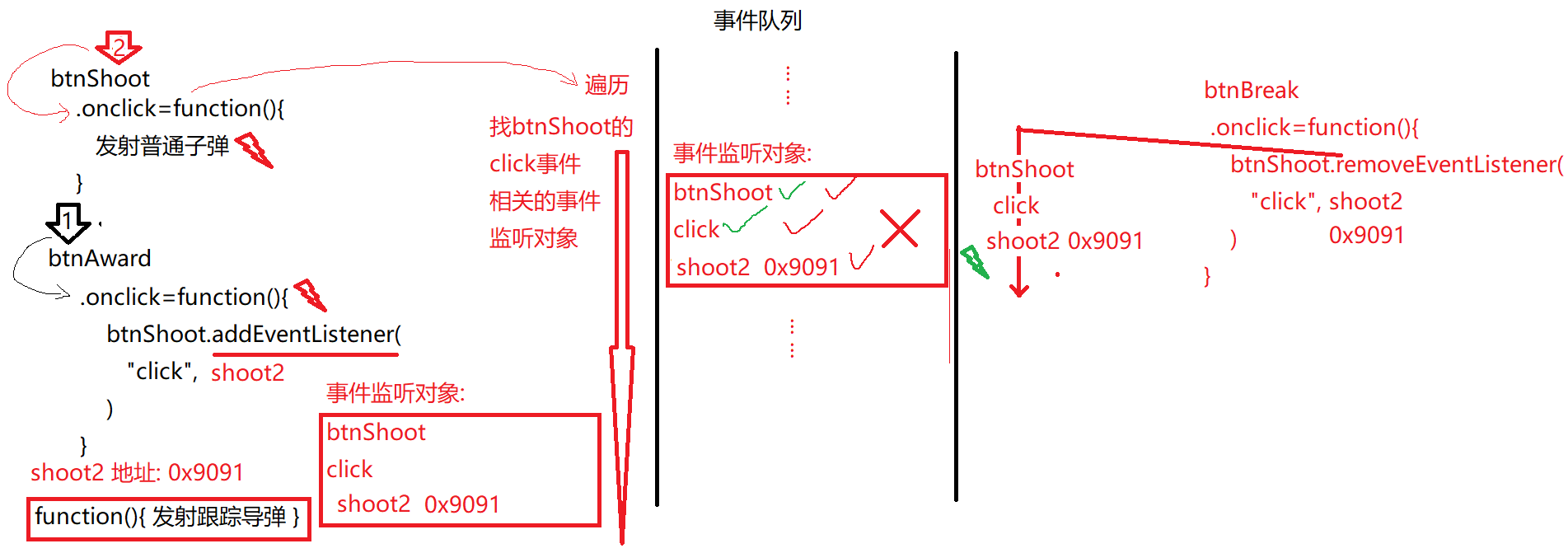
1). **元素.removeEventListener("事件名", 原事件处理函数对象)**

2). 坑: 移除事件监听对象时，仅仅将事件处理函数写的和添加事件监听时一模一样，是无法移除原事件处理函数:

3). 原因: function 是 new Function , 是创建新函数对象的意思。绑定时function(){}创建了一个函数，移除时function(){}又会创建一个新函数，两个函数地址绝不相同！所以，在查找要删除的事件监听对象时，不可能用新函数对象地址，匹配旧的函数对象的。——所以，**只要绑定时使用匿名函数绑定，移除时也使用匿名函数移除，则都移除不成功！**



4). 解决: 今后如果一个函数有可能被移除，则**绑定时就不能用匿名函数**，应该用有名称的函数绑定。移除时，才可以用函数名变量获得原处理函数的地址，移除原处理函数。



5). 问题: 事件监听队列规定完全相同的事件监听对象(元素相同，事件名相同，事件处理函数对象的地址也相同)，只能添加一个，**不能重复添加！**——结果，如果我们使用有名称函数绑定事件，无论点几次添加事件监听，永远只有一个事件监听！

e. 示例: 实现为发射按钮添加多种子弹并能移除子弹

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");        //需求:        //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......`);        }        //如果一个事件处理函数有可能被移除，则绑定时必须用有名称的函数        function shoot2(){           alert(`发射跟踪导弹=>=>=>`);        }        //点获得跟踪导弹按钮: 给shoot按钮再添加一种跟踪导弹        btnAward.onclick=function(){           btnShoot.addEventListener("click",shoot2)        }           //结果: 再点shoot按钮时，可以发射两种子弹！        //点失去跟踪导弹按钮: 从shoot按钮上移除跟踪导弹        btnBreak.onclick=function(){           btnShoot.removeEventListener("click",shoot2)        }           //结果: 再点shoot按钮时，恢复成只发射一种子弹     </script>  </body>  </html>  运行结果: |

f. 扩展示例：添加多个跟踪导弹，并能移除多个跟踪导弹

1\_addEventListener2.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");        //需求:        //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......`);        }        //定义数组保存所有添加的跟踪导弹函数:        var funs=[];        //点获得跟踪导弹按钮: 给shoot按钮再添加一种跟踪导弹        btnAward.onclick=function(){           //每点击一次添加就创建一个新函数，加入数组中           var shoot=function(){              alert(`发射跟踪导弹=>=>=>`);           }           funs.push(shoot)           btnShoot.addEventListener("click",shoot)        }           //结果: 再点shoot按钮时，可以发射两种子弹！        //点失去跟踪导弹按钮: 从shoot按钮上移除跟踪导弹        btnBreak.onclick=function(){           for(var fun of funs){              btnShoot.removeEventListener(                 "click", fun              )           }        }           //结果: 再点shoot按钮时，恢复成只发射一种子弹     </script>  </body>  </html>  运行结果: 同上 |

2. 事件模型:

(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

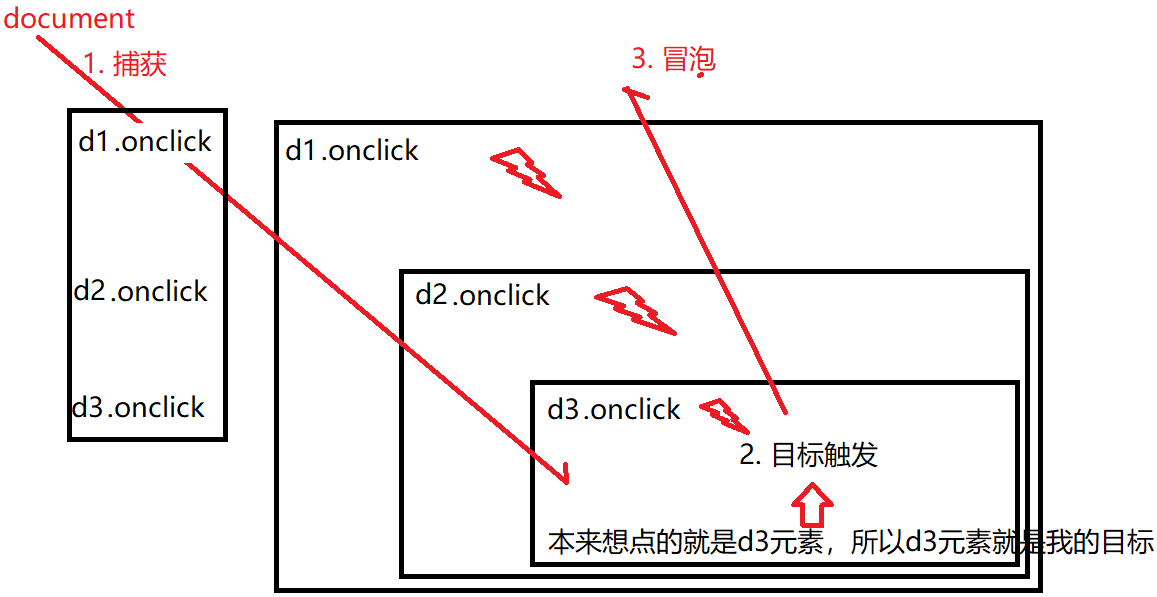
(2). DOM标准事件模型包括: 3个阶段:

a. **捕获**: 从document开始，**从外向内依次记录**实际触发事件的元素的所有父级元素的事件处理函数——只记录，不执行

b. **目标触发**: 首先触发最早触发事件的那个元素上的事件处理函数

最早触发事件的元素——称为目标元素(target)

c. **冒泡**: 然后依次**由内向外触发**捕获阶段记录的各级父元素上的事件处理函数



3. 事件对象:

(1). 事件对象: 事件发生时，浏览器自动创建的(不用我们自己创建)保存事件信息的对象

(2). 何时: 2种:

a. 想获得事件相关的信息时，比如鼠标坐标

b. 想改变事件默认的行为时

(3). 如何获得事件对象: 事件对象总是在事件发生时，自动作为事件处理函数的第一个参数自动传入。所以，我们只要在事件处理函数上，**提前定义一个形参e**，等着接就行了！——信任

事件发生时自动创建event对象

↓

**元素.onclick=function( e ){ ... }**

(4). 能做什么:

a. 阻止冒泡:

1).  **e.stopPropagation()**

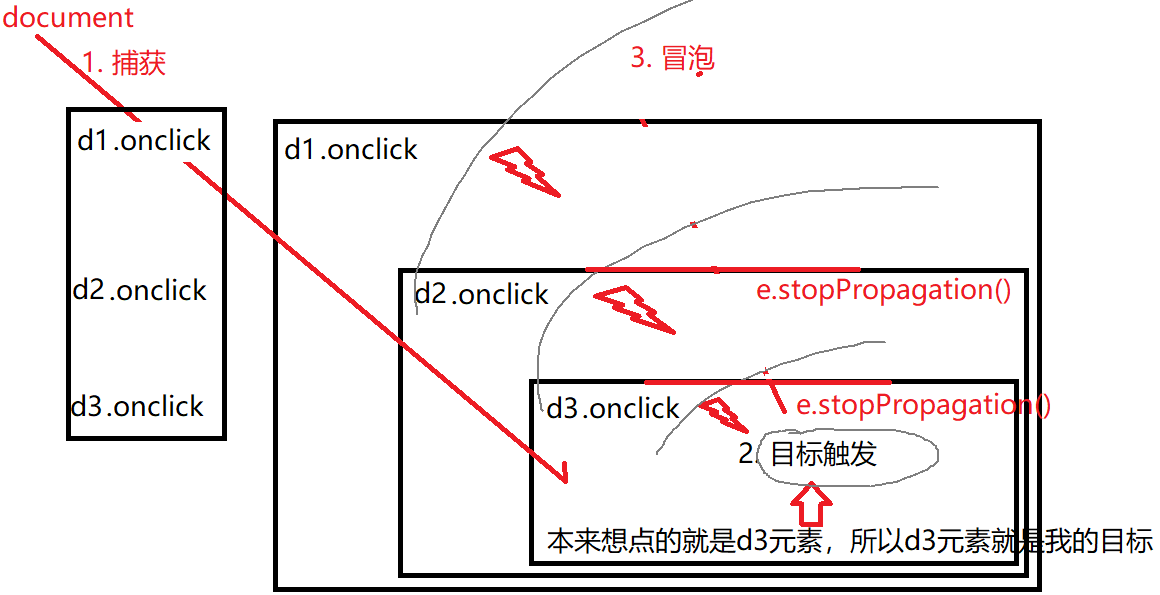
停止 漫延

2). 强调:

i. 最外层的元素上不用阻止冒泡

ii. 在一个事件处理函数中，e.stopPropagation()写前写后无所谓，因为e.stopPropagation()**阻止的是父元素**的事件继续触发，而**不是阻止自己**！

3). 示例: 使用e.stopPropagation()停止漫延



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: 175px;          width: 100px;          height: 100px;        }        #d3 {          background-color: red;          position: absolute;          top: 25px;          left: 225px;          width: 50px;          height: 50px;          line-height: 50px;        }      </style>      </head>    <body>      <div id="d1">        <div id="d2">          <div id="d3">          </div>        </div>      </div>      <script>        //需求: 为每个div都绑定单击事件，希望点哪个div，哪个div就喊疼！        var d1=document.getElementById("d1");        var d2=document.getElementById("d2");        var d3=document.getElementById("d3");        d1.onclick=function(){          //因为d1外边没有东西了，所以没必要阻止冒泡了！          alert("d1 疼")        }        d2.onclick=function(e){          //e.stopPropagation();          alert("d2 疼");          e.stopPropagation();        }        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:

1). 优化: **尽量减少事件监听对象的个数**

2). 原因: 因为浏览器触发事件时，是用遍历的方式查找事件监听对象的。所以，事件监听对象多，则查找速度慢。事件监听对象少，查找速度就快！

3). 何时: 多个平级的子元素都需要绑定相同的事件时，都要利用冒泡优化

4). 如何: 3步:

**i. 第一步: 事件只在父元素上集中绑定一个！**

所有子元素触发事件时，都通过冒泡自动触发父元素的事件处理函数，共享使用——所以这个技巧也称为"事件委托"，意为，所以子元素将自己想做的事儿托付父元素集中代为保管！

ii. 问题: 当事件冒泡到父元素上才执行时，this也跟着指向了实际触发事件处理函数的父元素，不再指向最初点击的子元素——在事件委托中，this用不了了！

**iii. 第二步: 事件委托中所有this都要用e.target代替**

e.target: 是事件对象中专门保存最初触发事件的元素的特殊属性

优点: 不随冒泡而改变！

iv. 问题: 事件绑定在父元素上，万一点父元素也会触发事件——我们不希望看到的

v. 第三步: 如果使用事件委托，则**事件处理函数中必须先判断触发事件的目标元素是不是想要的元素。**只有判断通过的是我们想要的元素，才能继续执行事件处理函数中后续代码！

**nodeName属性专门保存一个元素全大写的标签名**

5). 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><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>        //1. 只给父元素div绑定单击事件        var div=document.getElementById("keys");        div.onclick=function(e){          //需求1: 无论点哪个子元素按钮，都喊一样的疼！          //alert("疼!");          //需求2: 点哪个按钮，就让当前按钮的内容变为❀          //错误: this->父元素          //this.innerHTML="❀";          //正确: e.target          //需求3: 点父元素或其他无关的子元素，不允许触发事件          //判断: 只希望button元素触发事件          //如果当前目标元素是button元素          //           不能改名          if(e.target.nodeName=="BUTTON"){            //才修改内容为❀            //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;            }          }          //nodeName属性专门保存一个元素全大写的标签名        }      </script>    </body>  </html>  运行结果: |

c. 阻止默认行为

1). 问题: 个别元素身上带有一些默认的行为，而这些默认行为是我们不想要的！

2). 比如: <a href="#xxx">点击这个a，总是去自动修改地址栏里的url——和今后框架中的路由发生冲突，所以，我们不希望a元素擅自修改地址栏中的url

3). 如何: **e.preventDefault()**

阻止 默认

4). 针对a，还有另外一种方法:

将来只要只想使用一个a当按钮用时，href="javascript:;"即可

javascript: 让a不要跳转，转而执行一条js语句

; 是空语句的意思，什么也不执行

5). 示例: 使用两种方法阻止a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="#">click me</a><br/>    <a id="a2" href="javascript:;">click me</a>    <script>      //只想拿a当做按钮:      var a1=document.getElementById("a1");      a1.onclick=function(e){        e.preventDefault();        alert("疼!");      }      var a2=document.getElementById("a2");      a2.onclick=function(){        alert("疼!");      }    </script>  </body>  </html>  运行结果: |

d. 获取鼠标位置: 当事件发生时，我们可以获得三组鼠标位置

1). 相对于**屏幕**左上角的x，y坐标: **e.screenX, e.screenY**

屏幕 屏幕

2). 相对于**文档显示区**左上角的x，y用坐标: **e.clientX, e.clientY**

客户端 客户端

浏览器 浏览器

3). 相对于**事件所在元素**左上角的x，y坐标: **e.offsetX e.offsetY**

偏移 偏移

  
 e. 页面滚动事件:

1). 在窗口发生滚动时触发一个事件

**window.onscroll=function(){**

2). 可获得当前窗口滚动过的距离

Var **scrollTop=document.documentElement.scrollTop||**

**document.body.scrollTop**



3). 根据滚动过的距离执行不同的操作

}

4). 示例: 页面滚动到500位置时，显示返回顶部

7\_scrollTop.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 id="toTop">      <a href="#">返回顶部</a>    </div>    <script>      window.onscroll=function(){        var scrollTop=document.documentElement.scrollTop||document.body.scrollTop;        console.log(scrollTop);        var toTop=document.getElementById("toTop");        //希望当滚动距离>=500时，让toTop显示        if(scrollTop>=500){          toTop.style.display="block";        }else{//否则如果滚动距离<500时，让toTop隐藏          toTop.style.display="none";        }      }    </script>   </body>  </html>  运行结果: |

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

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

(6). 页面滚动事件:

window.onscroll=function(){

var scrollTop=document.documentElement.scrollTop||

document.body.scrollTop

//如果scrollTop>多少，就执行xx操作

//否则就恢复原样

}

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 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. 看小程序视频完成计算器效果:

小程序->在线->DOM->day04 3. 利用冒泡 事件委托

3. 看小程序串讲视频，总结DOM所有知识点:

小程序->在线->DOM->day04 0. 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