提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题清单都已经整理在小程序->首页->DOM->day04里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->DOM->day04里

3. 今天的重点视频、扩展高频鄙视题视频已经整理在小程序->在线->DOM->day04里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

今天开始！每个案例三遍:

1. 连注释+代码抄一遍

2. 保留注释，删掉代码，自己试着填回来

3. 删除注释和代码，自己用人话把注释写回来！——才算自己会！

正课:

1. 什么是BOM:

2. window对象

3. 打开新窗口

4. history

5. location

6. navigator

7. \*\*\*\*\*事件

一.什么是BOM: Browser Object Model

1. 什么是: 一批专门操作浏览器软件或浏览器窗口的对象或方法的集合

2. 何时: 今后只要需要获得浏览器软件或窗口的信息，或要操作浏览器窗口时，都用BOM

3. 问题: BOM没有标准！严重的兼容性问题！所以现在用的越来越少了！

4. BOM由一系列对象组成，每个对象各自掌管一个功能，包括:

(1). window: 最大的对象，包含其它对象

(2). window下还有：history, location, navigator, document, event, screen

二. window对象:有三个角色:

1. 代替global充当ES标准中固定的全局作用域对象，来保存我们自己定义的全局变量和全局函数

2. 包含所有ES、DOM、BOM的所有可直接使用的对象和函数。

3. 代表当前正在打开的这个浏览器窗口

(1). 用window，可以设置和监控窗口大小！

a. 获取窗口的大小: 2组

1). 窗口完整大小: window.outerWidth window.outerHeight

2). 窗口中文档显示区大小: window.innerWidth window.innerHeight

文档显示区是指浏览器窗口中专门显示网页内容的区域

b. 响应窗口大小改变事件: window.onresize事件

c. 示例: 当窗口大小改变时，自动执行操作:

|  |
| --- |
| <body>  <h3 id="h3"></h1>  <script>    var h3=document.getElementById("h3");    //     当重新改变大小    window.onresize=function(){      h1.innerHTML=`      窗口总大小: ${window.outerWidth} x ${window.outerHeight}      <br/>      窗口文档显示区部分大小: ${window.innerWidth} x ${window.innerHeight}`    }  </script>  </body> |

(2). 监控和操作窗口中页面的滚动:

a. 页面滚动事件:

当窗口中的滚动条发生滚动时自动执行

window.onscroll=function(){

var scrollTop=document.documentElement.scrollTop

||document.body.scrollTop;

}



b. 还可主动控制页面滚动到一个具体的位置:

1). window.scrollTo(水平方向滚动到的位置, 垂直方向滚动到的位置)

通常水平方向不滚动:0

2). window.scrollBy() //看作业中扩展视频！

c. 示例: 监控页面滚动，并主动滚动到指定位置:

|  |
| --- |
| <body style="height:2500px">  <h3 id="h3" style="position:fixed; top:0; left:0"></h3>  <h4 style="position:fixed; top:0; right:0">    <p id="h4"></p>    <button onclick="window.scrollTo(0,500)">滚动到500</button>    <button onclick="window.scrollTo(0,1000)">滚动到1000</button>  </h4>  <script>    var h3=document.getElementById("h3");    var h4=document.getElementById("h4");    //     当重新改变大小    window.onresize=function(){      h3.innerHTML=`      窗口总大小: ${window.outerWidth} x ${window.outerHeight}      <br/>      窗口文档显示区部分大小: ${window.innerWidth} x ${window.innerHeight}`    }    //当滚动条发生滚动时    window.onscroll=function(){      var scrollTop=document.documentElement.scrollTop                ||document.body.scrollTop;      h4.innerHTML=`滚动过的距离是: ${scrollTop}`    }  </script>  </body> |

(3). 还可以打开和关闭窗口:

a. 打开新窗口: window.open()

b. 关闭当前窗口: window.close();

三. 打开新窗口：4种需求

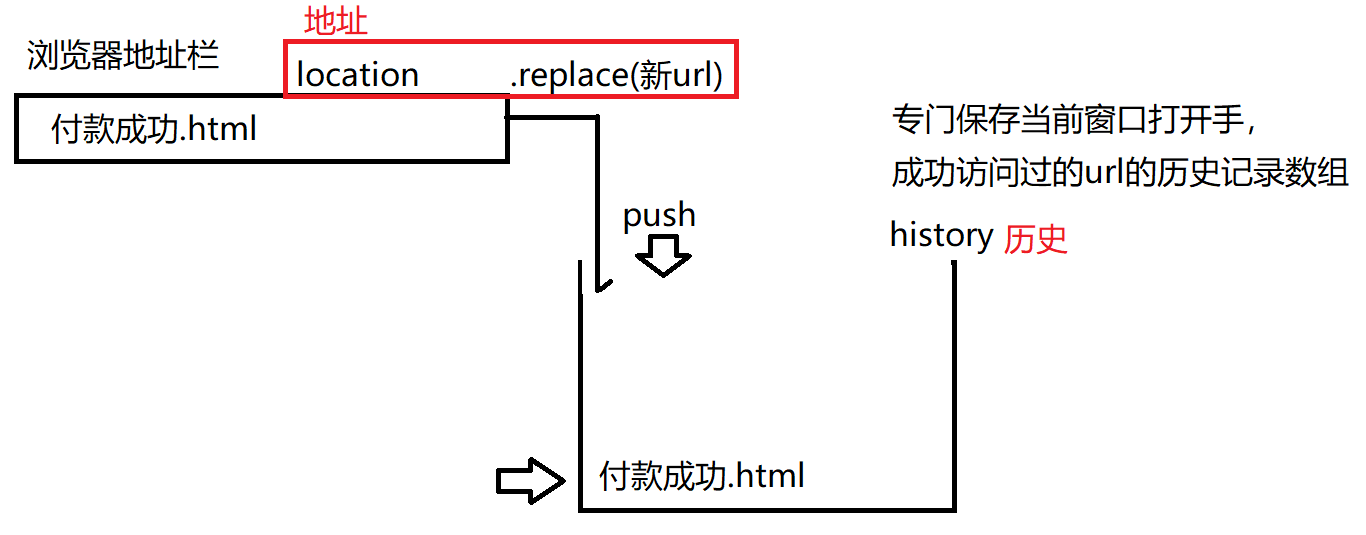
1. 在当前窗口打开新链接，可后退

(1). html中: <a href=”新链接url” target=”\_self”>

(2). js中: window.open(”新链接url”, ”\_self”)

2. 在当前窗口打开新链接，禁止后退

(1). js中: location.replace(“新url”)



3. 在新窗口打开新链接，可打开多个

(1). html中: <a href=”新链接url” target=”\_blank”>

(2). js中: window.open(“新链接url”,”\_blank”)

4. 在新窗口打开新链接，只能打开一个

(1). html中: <a href=”新链接url” target=”自定义窗口名”>

(2). js中: window.open(”新链接url”, ” 自定义窗口名”)

5. 原理:

(1). 在浏览器中每打开一个新窗口，新窗口都会有一个唯一的名字

(2). 浏览器规定，同名的窗口只能打开一个。再打开另一个同名窗口，新窗口会覆盖旧同名窗口

(3). 包括:

a. 其实可以在打开新窗口时自定义每个窗口的名字:

1).<a href=”新链接url” target=”自定义窗口名”>

2).window.open(”新链接url”, ”自定义窗口名”)

3). 结果: target中的自定义窗口名会成为新窗口的name属性名

4). 强调: 今后我们自己再声明全局变量时，不能使用”name”。

b. 预定义窗口名:

1). \_self: 将当前窗口自己的名字，给新窗口——新窗口覆盖旧窗口，实现在当前窗口打开，可后退

自己

2). \_blank: 不给新窗口指定任何名称。但是浏览器不会让新窗口名空着的！浏览器会在底层随机为窗口分配不重复的名称！——用\_blank打开的新窗口永远不重名！所以，可打开多个！

6. 示例: 四种打开新链接的方式:

|  |
| --- |
| <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","ank");  }  </script>  </head>  <body>  <h3>1. 在当前窗口打开新链接，可后退</h3>  <a href="http://tmooc.cn" target="\_self">go to tmooc</a><br/>  <a href="javascript:open1()">go to tmooc(js)</a>  <h3>2. 在当前窗口打开新链接，禁止后退</h3>  <a href="javascript:open2()">go to tmooc(js)</a>  <h3>3. 在新窗口打开新链接，可打开多个</h3>  <a href="http://tmooc.cn" target="\_blank">go to tmooc</a><br/>  <a href="javascript:open3()">go to tmooc(js)</a>  <h3>4. 在新窗口打开新链接，只能打开一个</h3>  <a href="http://tmooc.cn" target="ank">go to tmooc</a><br/>  <a href="javascript:open4()">go to tmooc(js)</a><br/>  <a href="http://tmooc.cn" target="tm">go to tmooc</a>  </body> |

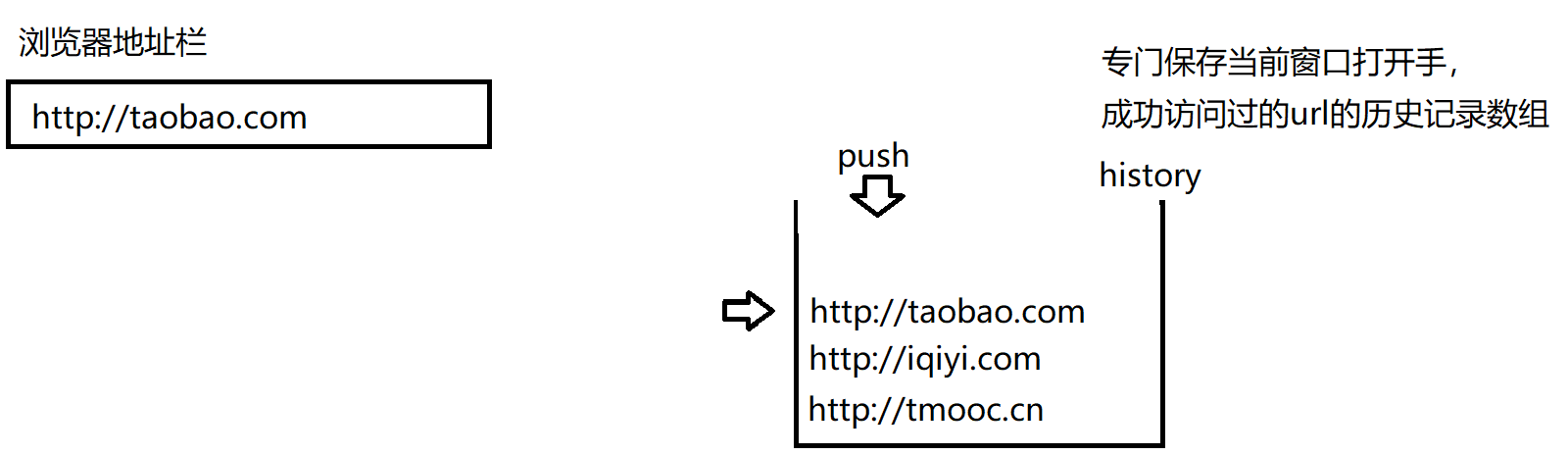
四. history:

1. 什么是： 浏览器内存中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: 当前窗口每成功打开一个新url，就会将新的url 压入history数组中保存。新加入的url在上，旧的url被压在下边。

3. history还控制着前进后退:

1). 如果浏览器当前正在显示的url下方有旧的url，则可以后退  
 2). 如果浏览器当前正在显示的url上方有新的url，则可以前进



5. history提供了专门用程序执行前进后退操作的方法:

(1). 前进一步: history.go(1)

(2). 后退一步: history.go(-1)

有时后退一步可能不管用，需要后退2步 ：history.go(-2)

(3). 刷新: history.go(0)

五: location

1. 什么是：专门保存当前浏览器正在打开的url的对象，并提供了各种跳转页面的方法。

2. 何时: 获取当前浏览器正在打开的url的信息时，或执行一些跳转操作时

3. location提供了分段获取url各个部分的属性:

location.href 获取当前浏览器正在打开的完整的url

location.protocol 协议部分

location.host 主机名+端口号

location.hostname 仅主机名

location.port 仅端口号

location.pathname 相对路径

location.search ?及其之后的查询字符串参数列表

location.hash #及其之后的锚点地址

4. location还提供了一些执行跳转新页面的方法:

(1). 也可以实现在当前窗口打开可后退: location.href=”新链接”

(2). 只有location可以实现在当前窗口打开，禁止后退: location.replace(“新链接”)

(3). 也提供了刷新页面的方法: location.reload()

六. navigator(领航者(浏览器))

1. 什么是: 专门保存浏览器配置信息的对象

2. 何时: 今后只要想获取浏览器的配置信息，都可用navigator

3. 如何:

(1). 可以判断是否安装了某个插件：

a. 所有已安装的插件信息都放在了：navigator.plugins集合中，插件的全称就是这个插件在集合中的下标。

b. 如何判断当前浏览器是否安装某个插件:

if(navigator[“插件全称”]!==undefined){

已安装该插件

}else{

未安装该插件

}

c. 示例: 检查浏览器是否安装PDF插件和Flash插件

|  |
| --- |
| console.log(navigator.plugins);  //想判断当前浏览器是否安装PDF插件  if(navigator.plugins["Chrome PDF Viewer"]!==undefined){    document.write(`<h1>已安装PDF插件，可以浏览PDF电子书</h1>`)  }else{    document.write(`<h1>未安装PDF插件，<a href="javascript:;">点此下载安装</a></h1>`)  }  //想判断当前浏览器是否安装Flash插件  if(navigator.plugins["Shockwave Flash"]!==undefined){    document.write(`<h1>已安装Flash插件，可以播放Flash动画</h1>`)  }else{    document.write(`<h1>未安装Flash插件，<a href="javascript:;">点此下载安装</a></h1>`)  } |

(2). 判断浏览器的名称和版本号:

a. 浏览器的名称和版本号是放在navigator.userAgent字符串中的

用户 代理 (浏览器)

b. 如何根据navigator.userAgent字符串中的内容，判断PC端浏览器的名称和版本号：见作业中小程序视频  
七. 事件

1. 什么是事件: 浏览器自己触发的或用户手动触发的页面内容的状态改变

2. 什么是事件处理函数属性: 每个元素上都有一批on开头的特殊属性。这些属性负责提前保存一个函数。当这个元素上发生了某个事件时，浏览器会自动找到这个元素上的对应的on开头的事件处理函数属性，自动执行提前保存的处理函数。

3. 何时使用事件处理函数属性: 只要希望当发生一个事件时，网页能自动执行一项任务时，就要用事件处理函数属性

4. 如何绑定事件处理函数: 3种方式:

(1). 在HTML中元素的开始标签中绑定事件处理函数属性: ——淘汰了

<元素 on事件名=”js表达式或语句”>

问题: 不符合内容与行为分离的原则，不便于维护和重用

(2). 在js中用赋值方式绑定:

a. 元素对象.on事件名=function(){ … }

b. 因为当事件发生时，浏览器实际执行的代码是: 元素对象.on事件名() this->当前正在触发事件的元素对象本身

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。无法同时绑定多个处理函数

d. 何时：所以如果一个元素的一个事件属性只绑定一个处理函数时，才可用赋值方式

(3). 最灵活的事件绑定方式: 添加事件监听对象的方式:

a. 何时: 希望一个元素的一个事件属性上可以同时绑定多个处理函数时，就用换成用添加事件监听的方式.

b. 元素.addEventListener(“事件名”, 事件处理函数)

添加 事件 监听对象

c. 强调:

1). 事件名不要带on前缀！因为DOM标准的事件名，都是没有前缀的，比如: click, mouseover, change,…

2). 事件处理函数，既可以是匿名函数，又可以是有名称的函数

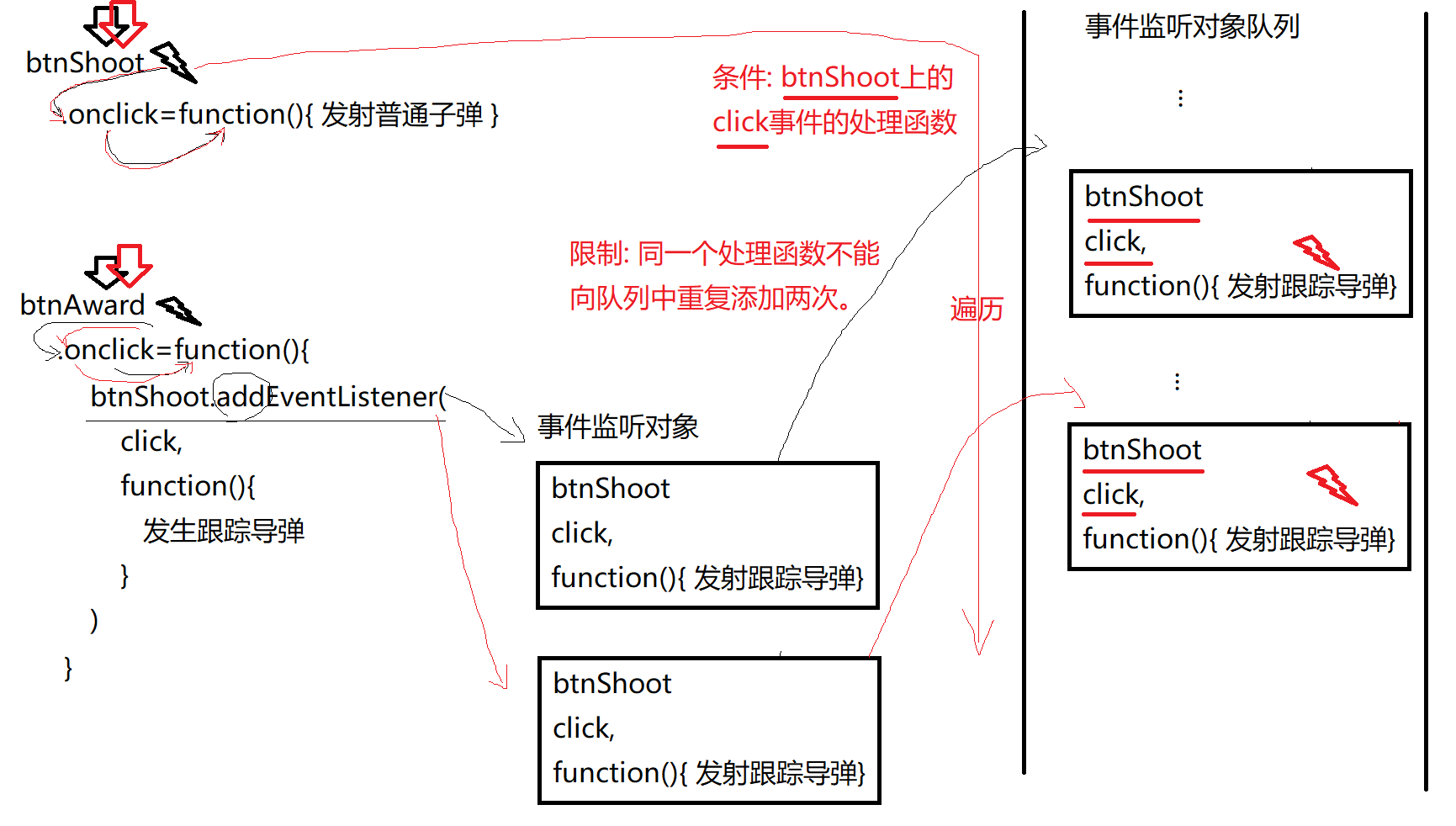
d. 原理：

1). 其实浏览器的内存中有一个巨大的事件对象队列

2). 每执行一次addEventListener()就会创建一个事件对象，加入到浏览器的事件队列中

每个事件对象中包含三个属性： 元素是谁？什么事件? 处理函数是什么？

3). 当事件发生时，浏览器会遍历事件队列中每个事件对象，找出符合条件的事件对象，执行其中的处理函数



e. 还可以移除事件监听:

1). 如果移除用onclick=function(){}方式绑定的处理函数，可用=null

2). 如果移除用addEventListener()方式绑定的处理函数，应该用:

元素.removeEventListener(“事件名”, 原处理函数)

3). 问题: 移除事件处理函数时，即使写的匿名处理函数和绑定时一模一样，也无法移除！

因为: function(){}等效于new Function()，所以两次匿名函数虽然写的一样，但是相当于先后创建了两个函数对象。所以，移除时用的不是原函数对象。而是新建的函数。当然移除不了！

4). 解决: 今后，只要一个事件监听对象有可能被移除，则绑定时，就必须用有名称的函数保存处理函数的地址。然后移除时才能用函数名获得原函数的地址。

5). 新问题: 一旦采用函数名添加事件监听，则同一个处理函数，不允许重复添加两次到事件队列。

(4). 示例: 使用赋值方式和添加事件监听方式绑定事件处理函数

|  |
| --- |
| <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");  console.dir(btnShoot);  btnShoot.onclick=function(){     console.log("发射普通子弹......")  }  function shoot2(){//new Function()     alert("发射跟踪导弹=>=>=>")  }  //希望点获得跟踪导弹按钮时，给btnShoot按钮多添加一种导弹  //希望再点btnShoot时，能同时发射两种子弹  btnAward.onclick=function(){     btnShoot.addEventListener("click",shoot2)//不加()，因为不是立刻调用  }  //希望点btnBreak，将跟踪导弹从btnShoot上移除  btnBreak.onclick=function(){     btnShoot.removeEventListener("click",shoot2)  }  </script>  </body> |

5. 事件模型:

(1). 什么是: 从事件发生到所有处理函数执行完，所经历的过程

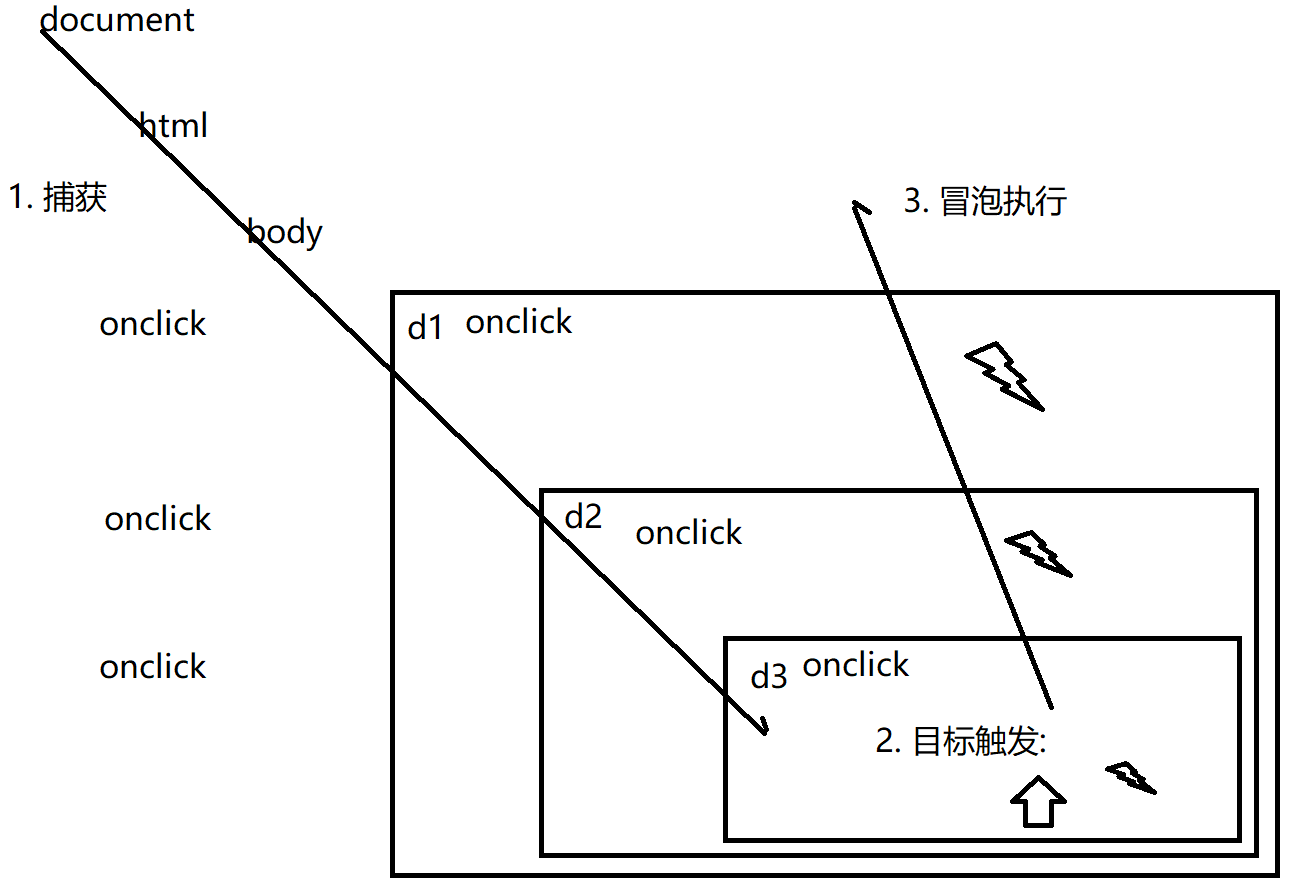
(2). DOM标准事件模型包含三个阶段:

a. 捕获阶段: 当事件发生时，浏览器从根节点document开始，由外向内浏览当前触发事件的元素的各级父元素，并记录各级父元素上绑定的相同事件处理函数。但是，该阶段只记录，不调用！

b. 目标触发: 优先触发当前点击的这个目标元素:

什么是目标元素: 本意想点的那个元素，就称为目标元素

c. 冒泡执行: 浏览器会按捕获阶段记录的各级父元素上绑定的事件的反向顺序，又内向外执行各级父元素上的事件处理函数。



6. 事件对象:

(1). 什么是事件对象: 当事件发生时自动创建的保存事件信息的对象

(2). 何时: 2种:

a. 希望获得事件信息时:

b. 希望改变事件默认的行为时:

(3). 如何获得事件对象: 不用自己创建！伸手等着要就行了！

事件对象总是会作为事件处理函数的第一个参数传入处理函数——信任！

event对象

btn.onclick=function( e ){ //所有形参都可以改名

}

(4). 拿到事件对象后，可以做什么:

a. 取消冒泡: e.stopPropagation()

停止 蔓延/冒泡

强调: e.stopPropagation()只阻止外层事件的执行，不阻止当前元素自己事件的执行。所以，在当前元素处理函数中，e.stopPropagation()写前写后无所谓！对当前元素的事件执行都没有影响！

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). 因为: 浏览器触发事件，是遍历事件队列查找监听对象方式。所以，监听对象越多，遍历越慢，事件响应速度越慢。反之，监听对象少，遍历快！事件响应的速度就快！

3). 何时: 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，都要用事件委托优化！

4). 如何: 3步:

i. 只要多个平级子元素都要绑定相同的事件处理函数时，只要在父元素上绑定一次处理函数即可

问题： 事件绑定在父元素上，this不再指子元素，而是指父元素

ii. 解决: 事件委托中不要用this获得当前元素对象。而要换成e.target

e.target，是事件对象中自带的，永久保存最初的目标元素的属性。不随冒泡而改变！

问题: 也可能点在父元素上或其它不想要的子元素上，也会出发事件

iii. 解决: 事件委托中，都要先验证目标元素e.target的特征是否符合要求！只有符合要求的子元素，才能继续执行处理函数。如果发现，e.target是不符合要求的元素，就什么都不干！

4). 示例: 计算器功能

|  |
| --- |
| <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>  //当多个平级button都要绑定相同的单击事件时，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！  var keys=document.getElementById("keys");  //                  event  //                    ↓  keys.onclick=function(e){    //this->父元素keys    //错误: alert(`${this.innerHTML} 疼!`);    //正确: e.target代替this    //希望只有e.target是按钮时，才能喊疼！别的元素不能喊！    if(e.target.nodeName==="BUTTON"){//nodeName每个元素自带的标记自己元素名的属性。属性值是全大写！！！      //alert(`${e.target.innerHTML} 疼！`);      /\*接下来，计算器的正常流程...\*/      //先查找用于显示屏的id为sc的文本域元素      var sc=document.getElementById("sc");      //判断e.target的内容      switch(e.target.innerHTML){        case "C"://如果是C          //就清除显示屏sc的内容          sc.value="";          break;        case "="://如果是=          //如果显示屏sc的内容不为空          if(sc.value!==""){            try{//才尝试              //将显示屏sc的内容放入全局函数eval()中计算得到计算结果              //再将结果放回显示屏sc中              sc.value=eval(sc.value);            }catch(err){//一旦发生错误，就捕获错误              //并将错误信息显示到显示屏sc中              sc.value=err;            }          }          break;        default: //其余按钮          //将e.target的内容追加到显示屏sc的内容末尾          sc.value+=e.target.innerHTML;      }    }  }  </script>  </body> |

总结:

1. 查找元素: 4种

(1). 不需要查找就可直接获得的元素: 5个

a. document

b. <html> document.documentElement

c. <head> document.head  
 d. <body> document.body

e. <form> document.forms[i]

(2). 按节点间关系查找: 2大类关系, 6个属性

何时: 已经获得一个元素，想找它周围附近的元素时

a. 父子关系:

1). 获得一个元素的父元素: 元素.parentNode

2). 获得一个元素下的所有直接子元素: 元素.children

3). 获得一个元素下的第一个直接子元素: 元素.firstElementChild

4). 获得一个元素下的最后一个直接子元素: 元素.lastElementChild

b. 兄弟关系:

1). 获得一个元素的前一个兄弟元素: 元素.previousElementSibling

2). 获得一个元素的后一个兄弟元素: 元素.nextElementSibling

(3). 按HTML特征查找：4个

何时: 只需要一个条件就可找到想要的元素时

a. 按id查找一个元素: var 一个元素=document.getElementById(“id名”)

b. 按标签名查找多个元素: var 类数组对象=任意父元素.getElementsByTagName(“标签名”)

c. 按class名查找多个元素: var 类数组对象=任意父元素.getElementsByClassName(“class名”)

d. 按name名查找多个表单元素: var 类数组对象=document.getElementsByName(“name名”)

(4). 按选择器查找: 2个

何时: 查找条件复杂时

a. 只查找一个符合条件的元素: var 一个元素=任意父元素.querySelector(“选择器”)

b. 查找多个符合条件的元素: var 类数组对象=任意父元素.querySelectorAll(“选择器”)

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

2. 修改元素: 3种:

(1). 内容: 3种:

a. 获取或修改元素的原始的HTML内容: 元素.innerHTML

b. 获取或修改元素的纯文本内容: 元素.textContent

c. 获取或修改表单元素的值: 元素.value

(2). 属性: 3种:

a. 字符串类型的HTML标准属性: 2种:

1). 核心DOM 4个函数: 元素.getAttribute()/setAttribute()/removeAttribute()/has Attribute();

2). HTML DOM简写: 元素.属性名

b. bool类型的HTML标准属性: 只能: 元素.属性名

c. 自定义扩展属性: 2种:

1). 核心DOM4个函数: 元素.getAttribute()/setAttribute()/removeAttribute()/has Attribute();

2). 如果自定义扩展属性名以data-开头，则可用: 元素.dataset.自定义属性名 (不要加data-前缀)

(3). 样式: 3种:

a. 修改一个元素的一个css属性: 元素.style.css属性

b. 获取一个元素的css属性: var style=getComputedStyle(元素对象); style.css属性名

c. 批量修改一个元素的多个css属性: 元素.className

3. 添加新元素:

(1). 只添加一个新元素: 3步:

a. 创建一个新的空元素对象: var 新元素=document.createElement(“元素名”)

b. 为新元素添加必要属性: 新元素.属性名=属性值

c. 将新元素添加到DOM树: 3种:

1). 追加到指定父元素下的现有子元素结尾: 父元素.appendChild(新元素)

2). 插入到指定父元素下一个现有子元素之前: 父元素.insertBefore(新元素,现有子元素)

3). 替换指定父元素下一个现有子元素: 父元素.replaceChild(新元素,现有子元素)

(2). 优化: 如果要添加多个元素到页面，需要尽量减少操作DOM树的次数

a. 如果同时添加父元素和子元素时，应该先在内存中将所有子元素加入父元素中，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，需要添加多个平级子元素时，应该先将多个平级子元素加到文档片段中，最后再一次性将文档片段添加到DOM树: 3步:

1). 创建文档片段: var frag=document.createDocumentFragment();

2). 将子元素加入文档片段中: frag.appendChild(子元素)

3). 将文档片段整体加入DOM树: 父元素.appendChild(frag)

4. 删除元素: 父元素.removeChild(子元素)

HTML DOM的简写:

1. <img>: var img=new Image();

2. <table>:

(1). table管着行分组:

a. 添加行分组: var thead=table.createTHead(); var tbody=table.createTBody(); var tfoot=table.createTFoot();

b. 删除行分组: table.deleteTHead(); table.deleteTFoot();

c. 获取行分组: table.tHead table.tFoot table.tBodies[i]

(2). 行分组管着行:

a. 添加行: 3种:

1). 末尾添加一个新行: var tr=行分组.insertRow()

2). 开头插入一个新行: var tr=行分组.insertRow(0)

3). 在中间插入一个新行: var tr=行分组.insertRow(i)

b. 删除行: table.deleteRow(tr.rowIndex);

c. 获取行: 行分组.rows[i]

(3). 行管着格:

a. 添加格: var td=tr.insertCell(); //在行末尾追加一个新格

b. 删除格: tr.deleteCell();

c. 获取格: tr.cells[i]

3. <form>:

(1). 获取<form>中的表单元素:2种:

a. 标准: form.elements[i或name或id]

b. 如果表单元素有name属性可以直接: from.name

(2). 让一个表单元素自动获得焦点: 表单元素.focus();

总结: this

1. obj.fun() this->obj

2. fun() 或 (function(){…})() 或 回调函数 中的this->window

3. new Fun() this->new将来创建的某个子对象

4. 原型对象.prototype.fun=function(){ … } this->将来调用这个方法的.前的某个子对象

5. 访问器属性中的this->当前访问器属性所在的对象

Object.defineProperties(eric,{

eage:{

get(){ return this.\_eage },

set(value){ this.\_eage=value }

}

})

6. 元素.onclick=function(){ … } this->当前正在触发事件的元素

作业:

1. (必须)看小程序视频学习事件委托:

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

2. (必须)看小程序视频学习阻止元素默认行为:

小程序->在线->DOM->day04 4. 阻止默认行为 preventDefault

3. (必须)看小程序视频学习获取鼠标坐标和拖拽效果

小程序->在线->DOM->day04 5. 鼠标坐标x y 原生DOM拖拽效果

4. (必须)看小程序视频学习倒计时关闭窗口

小程序->在线->DOM->day04 作业: 倒计时关闭窗口

5. (扩展)看小程序视频学习判断浏览器名称和版本号

小程序->在线->DOM->day04 作业: 使用navigator.userAgent鉴别浏览器名称和版本号.

6. (扩展)看小程序视频学习实现页面滚动动画效果

小程序->在线->DOM->day04 作业: 原生DOM页面滚动效果

7. (扩展)看小程序视频学习高频笔试题，事件循环机制

小程序->在线->DOM->day04 作业: 高频笔试题nodejs 浏览器不同事件循环

8. (扩展)看小程序视频学习原生DOM版轮播图制作

小程序->在线->DOM->day04 赠送: 原生DOM轮播图