html绑定事件的三种方式：

1.直接在DOM中绑定：

<any onclick='函数名()'></any>

<script>

function 函数名(){

...//事件处理程序

}

/\*</scri

优势：最方便，没有之一，代码量少，逻辑清楚

劣势：太多！

1.存在时差问题，因为用户可能会在html元素一出现在页面上就直接开始点击， 而此时因为js还在加载(不管是在网页中嵌入的脚本代码，还是通过script标签的src 属性引入的外部脚本文件，网页解析到这都会暂停解析开始加载外部资源文件，造成延 迟),尚不具备执行条件，导致事件触发错误，用户体验贼差。

2.html元素和js代码的耦合度过高。如果要更换事件处理程序，就要同时在html 代码和js代码两处修改，造成不必要的麻烦，不利于后期的代码维护

2.在js中绑定(DOM一级事件处理程序)

var 变量名 = document.get系列 ||.query系列方法获取元素;

变量名.on事件名 = function(){

...//事件处理程序

}

优势：简单，兼容几乎所有浏览器

劣势：只能给元素绑定一个事件

疑问：如何给多个元素同时绑定事件？

例如，当前文档有四个a标签，我想一次性给四个a标签都绑定上事件，该如何 实现？

var a1 = document.querySelectorAll('a');

for(var i = 0;i < a1.length;i++){

a1[i].onclick = function(){

...//事件处理程序

}

}

追加疑问：我想四个a标签点击的时候都给我输出其各自对应的数组下标，该如 何实现？能直接添加alert(i)么？

当然不行，如果这样，无论你点击哪个a标签，其弹出值都为4，因为在你触发事 件之前，事件处理函数已经绑定完了，因为伴随着文档流的加载，事件绑定代码被解析， 一定是先绑定好事件，再跳出循环，而不是你先触发事件再绑定的。而此时因为跳出循 环，i只能为4，所以不管你点击哪个a标签，其值都为4.

如何解决？

var a1 = document.querySelectorAll('a');

for(var i = 0;i < a1.length;i++){

a1[i].onclick = function(){

...//事件处理程序

}

}

在以上代码中，我们让：

a1[i].onclick = out\_(i);

事件去绑定一个外部的函数，这个外部的函数我们写在循环之外：

function out\_(index){

return function(){

alert(index);

}

}

通过一个闭包结构，保护好传入的位置形参(局部变量index)，进而保证第一个元 素点击完，调用完这个外部函数后，函数不会被垃圾回收，局部变量index不会被释放， 进而能够等待下一个元素触发事件函数进行使用。

3.DOM二级事件处理程序

1.什么是一级二级：一级，是指，当前元素绑定事件之后处理完，没了，没啥事了。 二级，是指，当前元素绑定事件之后触发事件所造成的其他一系列连带的操作和影 响。

2.如何绑定：

1.W3C标准中

使用addEventListener(‘事件名称’,处理函数,捕获||冒泡阶段)事件监听函数进行绑定

标准语法：obj.addEventListener(event,function,bCaptrue);

2.IE浏览器中：

使用attachEvent()事件监听函数进行绑定

标准语法：obj.attachEvent(event,function);

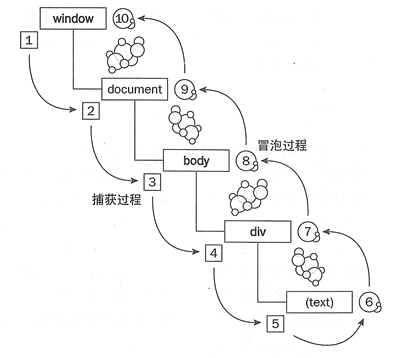
3.参数解析：

Event：事件名称

Function：事件处理程序

bCaptrue:捕获还是冒泡阶段执行事件，默认为true

3.DOM事件处理机制



我们知道，window对象是一切BOM对象的顶级父对象。从window开始，会存在document，也就是文档结构，文档里面(在这里我们以HTML文档为例)包含各种标签，按照DOM tree的顺序，依次是：html,body,...any元素。假设，当前body中，有一个div，div里有一个p标签，p标签中有一个a标签，文档结构如下：

<body>

<div>

<p>

<a href=’#’></a>

</p>

</div>

</body>

我们通过js代码先获取到这些所有元素：

Var bodys = document.getElementsByTagName(‘body’);

Var divs = document.getElementsByTagName(‘div’);

Var ps = document.getElementsByTagName(‘p’);

Var as = document.getElementsByTagName(‘a’);

然后我们给这获取到的四个元素全部绑定上事件，因为讲到了事件监听，我们统一

统一采用事件监听函数来绑定：

Bodys.addEventListener(‘click’,function(){

Alert(‘body被点击’);

},false);

divs.addEventListener(‘click’,function(){

Alert(‘div被点击’);

},false);

ps.addEventListener(‘click’,function(){

Alert(‘p被点击’);

},false);

as.addEventListener(‘click’,function(){

Alert(‘a被点击’);

},false);

ok,此时我们点击最里面的a标签，注意，发生了什么？

我的天了噜！！！不仅触发了a标签的事件，连带着p,div,body所绑定的事件也都被触发了！什么情况！？我只点击了a啊？我只点了一下啊？妈呀怎么全都触发了？......

接下来，我们进入DOM事件流，来看看到底怎么就一下子触发了所有的事件。

DOM2级事件规定：事件触发包括三个阶段：

1. 事件捕获阶段
2. 处于目标阶段
3. 事件冒泡阶段

当你点击某一个元素时，鼠标点击的一瞬间(我们以点击事件举例)，事件流产生。事件流的起点是DOM tree的根节点，事件流会顺着DOM tree，从根节点开始向下找，如果经过了某一个元素，这个元素绑定了事件，ok，执行事件；这个方向的事件流我们就叫做事件捕获阶段，最终捕获阶段在触发事件的元素结束，而执行事件的过程，就是目标阶段。Ok，捕获结束，到达触发事件的元素了，开始第三阶段，冒泡阶段的事件流，此时事件流会从触发事件元素开始，向根节点进行，也就是反方向了，在事件流回到根节点的过程中，也是一样，如果经过了某一个元素，这个元素绑定了事件，ok，执行事件，直至回到根节点，一次完整的事件流，结束。

在我们上面举得例子中，因为我们给每一个元素添加事件监听函数时，第三个参数我们都给的是false，代表所有事件全部在冒泡阶段执行，所以，当你点击最里面的a标签时，捕获阶段，没有事件执行，因为你规定好了，所有事件全部在冒泡阶段执行，而冒泡阶段开始，你没有给触发事件的元素设置阻止冒泡，导致你点击a，触发a的事件，连带着p,div,body都因事件流经过而触发！

那么，如何阻止事件冒泡？

4.阻止事件冒泡

1.事件对象event：BOM对象中众多的内置对象之一，属于window对象的一个子对象，该对象中封装了一系列属性及方法，其中就包括阻止事件冒泡方法stopPropagation方法，当你在某一事件处理函数中将事件对象event当做位置形参传入，并在函数内调用事件对象event的stopPropagation方法，就可以实现阻止事件冒泡。

2.通用写法：

Function 事件处理函数(e){

Var e = e || window.e; //此代码兼容IE8，因为IE8中识别event的方式 //为window.event

e.stopPropagation();

}

3.注意：

1.事件监听函数addEventListener()第三个参数bCaptrue我们一般都写false或不填，因为大多数情况下，我们都是将事件处理程序放在事件流的冒泡阶段去执行，这样做可以最大程度的兼容各种浏览器(因为IE就是在事件冒泡阶段处理事件，一般我们就是为了兼容IE浏览器)，如果不是特别需要，不建议在事件捕获阶段执行事件。

2.事件处理函数内想要调用阻止事件冒泡方法，必须将事件对象event当做位置形参传入函数内部，方可调用！

3.阻止冒泡方法在哪调用，事件流执行到这个元素之后就会终止冒泡，冒泡阶段后续元素的事件不会被触发！

5.阻止事件默认行为

有些时候，比如网页中有一个a标签，添加了一个超链接，而当前这个a标签还有添加好的事件处理函数，当你鼠标点击该a标签时，事件确实触发了，可也相当于点击了这个超链接，网页跳转了......触发事件的同时也触发了元素的默认行为，造成不必要的bug。所以有些时候，我们需要清除事件的默认行为。

1.如何清除：调用事件对象event的preventDefault()方法

2.语法：

Function 事件处理函数(e){

Var e = e || window.e;

e.preventDefault();

}