1、DOM事件模型是什么？

答：

DOM MDN

DOM（Document Object Model——文档对象模型）是用来呈现以及与任意HTML或XML交互的API文档。Dom是载入到浏览器中的文档模型，它用节点树的形式来表现文档，每个节点代表文档的构成部分（例如：element——页面元素、字符串或注释等等）。

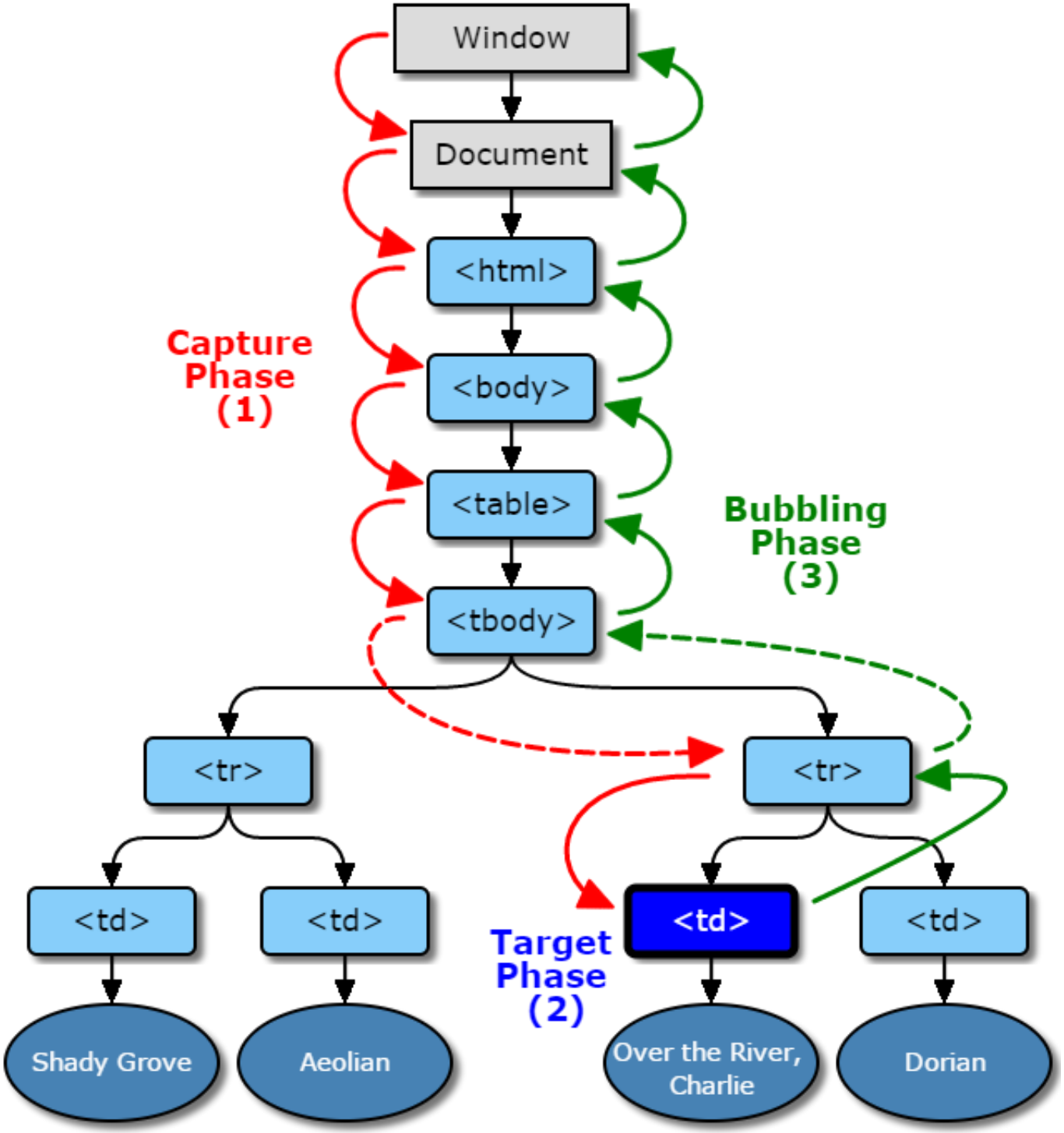
DOM是Web——万维网上使用最为广泛的API之一，因为它允许运行在浏览器中的代码访问文件中的节点并与之交互。节点可以被创建、移动或修改。事件监听器可以被添加到节点上并在给定事件发生时触发。

DOM并不是天生就被规范好了的，它是浏览器开始实现JS时才出现的。这个传统的DOM有时会被成为DOM 0。现在WHATWG维护DOM生活标准。

DOM事件模型 MDN

简介

这一章节介绍了DOM事件模型（DOM Event Model）。主要描述了事件（Event）接口本身以及DOM节点中的事件注册接口、事件监听接口，以及几个展示了多种事件接口之间相互关联的较长示例。



注册事件监听器

这里有三种方法来为一个DOM元素注册事件回调 EventTarget.addEventListener:

```
//假设myButton是一个按钮
myButton.addEventListener('click', function() {
    alert('Hello world');
}, false);
```

你应该在现代Web技术的页面中使用这个方法。

注：IE6-8不支持这一方法，但提供了类似的API即element.attachEvent用以替代。考虑到跨浏览器兼容性问题请使用有效的JS代码库。

HTML属性：

```
<button onclick="alert('Hello world!')">
```

属性中的JS代码触发时通过event参数将Event类型对象传递过去的。其返回值以特殊的方式来处理，已经在HTML规范中被描述。

应该尽量避免这种书写方式，这将使HTML变大并减少可读性。考虑到内容/结构及行为不能很好的分享开，这将造成bug很难被找到。

DOM元素属性

```
//假设myButton是一个按钮
myButton.onclick = function(event){
    alert('Hello world');
};
```

带有event参数的函数可以这样被定义。其返回值以特殊的方式来处理，已经在HTML规范中被描述。

这种方式的问题是每个事件及每个元素只能被设置一个回调。

访问事件接口

事件回调可以被绑定到包括DOM元素、文档、窗口等多种对象上。当一个事件被触发时，一个event对象将被创建并顺序的传递给事件监听者们。

Event接口可以在回调函数内被访问到，通过被传递进来作为第一个参数的事件对象。以下这个简单例子展示了如何将事件对象传递给事件回调函数，同时可以在这个函数中使用。

```
function foo(evt){
    //evt参数自动分配事件对象
    alert(evt);
}
table_el.onclick = foo;
```

DOM事件模型——博文

DOM

首先，DOM全程是Document Object Model，即文档对象模型。DOM是W3C的标准，定义了访问HTML和XML文档的标准。

```
W3C文档对象模型（DOM）是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。
```

DOM事件

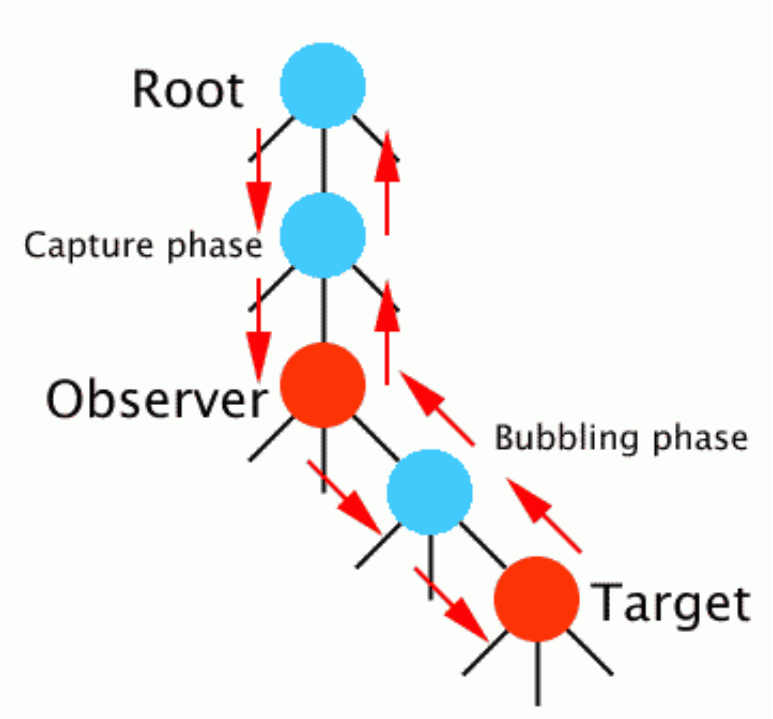
DOM使JS有能力对HTML上的事件做出反应。这些事件包括鼠标键盘的点击事件、移动事件以及页面中内容的变化等。HTML元素事件是浏览器内在自动产生的，当有事件发生时html元素会向外界（这里主要指元素事件的订阅者）发出各种事件，如click、onmouseover、onmouseout等等。

DOM事件流

DOM的结构是一个树形，每当HTML元素产生事件时，该事件就会在树的根节点和元素节点之间传播，所有经过的节点都会收到该事件。

DOM事件模型

DOM事件模型分为两类：一类是IE所使用的冒泡型事件（Buddling）；另一类是DOM标准定义的冒泡性与捕获型（Capture）的事件。除IE外的其他浏览器都支持标准的DOM事件处理模型。



冒泡型事件处理模型（Bubbling）：如上图所示，冒泡型事件处理模型在事件发生时，首先在最精确的元素上触发，然后向上传播，直到根节点。反映到DOM树上就是事件从叶子节点传播到根节点。
捕获型事件处理模型（Capture）：相反地，捕获型在事件发生时首先在最顶级的元素上触发，传播到最低级的元素上。在DOM树上的表现就是由根节点传播到叶子节点。

标准的DOM事件处理模型：分为三个阶段

1.父元素中所有的捕获型事件（如果有）自上而下地执行

2.目标元素的冒泡型事件（如果有）

3.父元素中所有的冒泡性事件（如果有）自下而上地执行

注册事件监听

1.传统方式的事件模型即直接在DOM元素上绑定事件处理器，例如

```
window.onload = function() {……}
obj.onmouseover = function(e){……}
obj.onclick = function(){……}
```

首先这种方式是无论在IE还是Firefox等其他浏览器上都可以成功运行的通用方式。这便是它最大的优势了，而且在Event处理函数内部的this变量无一例外的都指向被绑定的DOM元素，这使得JS程序员可以大大利用this关键字做很多事情。

至于它的缺点也很明显，就是传统方式只支持Bubbling，而不支持Capturing，并且一次只能绑定一个事件处理器在DOM元素上，无法实现多Handler绑定。最后就是function参数中的event参数只对非IE浏览器有效果（因为IE浏览器有特制的window.event）。

2.W3C（Firefox.e.g）Event Module

Firefox等浏览器很坚决的遵循W3C标准来制定浏览器事件模型，使用addEventListener和removeEventListener两个函数，看几个例子：

```
window.addEventListener('load', function(){……}, false);
document.body.addEventListener('keypress', function(){……},  false);
obj.addEventListener('mouseover', MV, true);
function MV(){……};
```

addEventListener带有三个参数，第一个参数是事件类型，就是我们熟知的那些事件名字去掉前面的‘on’，第二个参数是处理函数，可以直接给函数字面量或者函数名，第三个参数是boolean值，表示事件是否支持Capturing。

W3C的事件模型优点是Bubbling和Capturing都支持，并且可以在一个DOM元素上绑定多个事件处理器，各自并不会冲突。并且在处理函数内部，this关键字仍然可以使用只想被绑定的DOM元素。另外function参数列表的第一个位置（不管是否显示调用），都永远是event对象的引用。

至于它的缺点，只有在IE浏览器下不可使用这一点。

3.IE Event Module

IE自己的事件模型跟W3C的类似，但主要是通过attachEvent和detachEvent两个函数来实现的。依旧看几个例子吧：

```
window.attachEvent('onload', function() {……});
document.body.attachEvent('onkeypress', myKeyHandler);
```

可以发现它跟W3C的区别是没有第三个参数，而且第一个表示事件类型的参数也必须把‘on’给加上。这种方式的优点就是能绑定多个事件处理函数在同一个DOM元素上。

至于它的缺点，为什么如今在实际开发中很少见呢？首先IE浏览器本身只支持Bubbling不支持Capturing；而且在事件处理的function内部this关键字也无法使用，因为this永远都指向window object这个全局对象。要想得到event对象必须通过window.event方式，最后一点，在别的浏览器中，它显然是无法工作的。

2、移动端的触摸事件了解吗？

答：移动端相对于PC端，用触摸事件替代了点击事件。移动端也可以用点击事件，但是移动端的点击事件是对触摸事件的一重封装，会判断触摸开始到结束的时长来确定是点击事件还是长按时间等，体验上有延时效果。另外如果直接用移动端的触摸事件需要先禁用掉默认事件，避免引发一系列奇怪的问题。

3、事件委托是什么？有什么好处？

答：事件即文档或浏览器中发生的一些特定交互的瞬间，我们可以利用事件监听来预定事件，当事件发生的时候执行响应的处理程序。当事件发生在某个DOM节点上时，事件在DOM结构中进行一级一级的传递，这边形成了“流”，事件流便描述了从页面中接收事件的顺序。

DOM事件流：

可以想象画在一张纸上的一组同心圆，如果你把手指放在圆心上，那么你的手指指向的其实不是一个圆，而是纸上所有的圆。...>换句话说，在单击按钮的同时，你也单击了按钮的容器元素，甚至也单击了整个页面。

————《JavaScript高级程序设计（第三版）》page 345

DOM2级事件中规定事件流包含3个阶段：*捕获阶段；*处于目标阶段；*冒泡阶段

首先发生的是事件捕获阶段，此时事件还没有传递到目标节点对象上，所以我们就有机会在这个阶段进行事件的拦截。然后是目标节点接受到事件，最后是事件冒泡阶段，可以在这个阶段对事件作出处理和响应。我们先定义一段简单的html结构：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <div class="box">
```

```
            <button type="button" name="button">click me</button>
        </div>
    </body>
</html>
```

事件捕获阶段：在事件捕获阶段中，先由不具体的节点（即上层节点）接收到事件，然后一级一级往下传递，直到最具体的目标节点接收到事件。　在DOM2级事件规范中，要求事件从document对象开始传递，但是注入Chrome、Firefox等主流浏览器却是从window开始传递的。

`addEventListener`方法的第三个参数是一个布尔值（可选），指定事件处理程序是否在捕获或冒泡阶段执行。 当为`true`时，则事件处理程序将在捕获阶段执行。

误区：无论`addEventListener`的第三个参数是否为`true`，三个阶段都会走一遍，这里的第三个参数，指的是处理程序将会在捕获或者冒泡阶段执行，好比是你想买菜，你可以在上班路上，或者下班路上完成买菜，但无论什么时候买菜，你都要把这两段路程走完。

```
document.querySelector('#btn').addEventListener('click', function() {
    console.log('btn was clicked');
}, true);
document.querySelector('body').addEventListener('click', function() {
    console.log('body was clicked');
}, true);
document.querySelector('.box').addEventListener('click', function() {
    console.log('box was clicked');
}, true);
document.addEventListener('click', function() {
    console.log('document was clicked');
}, true);
window.addEventListener('click', function() {
    console.log('window was clicked');
}, true);
```

点击click me按钮后，控制台依次打印出执行结果（捕获阶段执行）：

```
window was clicked
document was clicked
body was clicked
box was clicked
btn was clicked
```

事件冒泡阶段：事件冒泡阶段与捕获阶段恰好相反，冒泡阶段是从最具体的目标对象开始，一层一层地向上传递，直到window对象。　addEventListener方法默认就是从冒泡阶段执

行时间处理程序。

```
document.querySelector('#btn').addEventListener('click', function() {
    console.log('btn was clicked');
});
document.querySelector('body').addEventListener('click', function() {
    console.log('body was clicked');
});
document.querySelector('.box').addEventListener('click', function() {
    console.log('box was clicked');
});
document.addEventListener('click', function() {
    console.log('document was clicked');
});
window.addEventListener('click', function() {
    console.log('window was clicked');
});
```

点击click me按钮后，控制台依次打印出执行结果：

```
btn was clicked
box was clicked
body was clicked
document was clicked
window was clicked
```

组织事件冒泡：我们可以使用event.stopPropagation()方法组织事件冒泡过程，以防止事件冒泡而带来不必要的错误和困扰。

```
document.querySelector('#btn').addEventListener('click', function (event) {
    console.log("btn was clicked");
    event.stopPropagation();
});

document.querySelector('body').addEventListener('click', function () {
    console.log("body was clicked");
});

document.querySelector('.box').addEventListener('click', function () {
    console.log("box was clicked");
});

document.addEventListener('click', function () {
    console.log("document was clicked");
```

```
});

window.addEventListener('click', function () {
    console.log("window was clicked");
});
```

点击click me按钮后，控制台打印出执行结果显示，事件没有再向上冒泡传递给其他节点对象：

```
btn was clicked
```

事件委托：每个函数都是对象，都会占用内存，所以当我们的页面汇总所包含的事件数量较多时，如果给每个节点绑定一个事件，加上事件处理程序，就会造成性能很差。还有一个问题是：某个元素节点是后来通过JS动态添加进页面中的，这时候我们如果提前对它进行绑定，但此时该元素并不存在，所以会绑定事件失败。解决上述两个问题的一个常用方案，就是使用事件委托。

```
document.querySelector('.box').addEventListener(function(event){
    switch(event.target.id){
        case 'btn':
            console.log('btn was clicked');
            break;
        case 'btn-2':
            console.log('btn-2 was clicked');
            break;
        default:
            console.log('box was clicked');
            break;
    }
});
$('.box').append("<button id='btn-2'>btn-2</button>")
```

简单说，事件委托就是把本来该自己接收的事件委托给自己的上级（父级，祖父级等等）的某个节点，让自己的“长辈们”帮忙盯着，一旦有事件触发，再由“长辈们”告诉自己：“喂，小子，有人找你~~”。

格式化文本

标题1

标题2

标题3

标题4

标题5

标题6

段落
地址

预格式文本

普通