# 事件处理

事件被看作是JavaScript与网页之间交互的桥梁，当事件发生时，可以通过JavaScript代码执行相关的操作。例如，用户可以通过鼠标拖拽窗口，改变窗口的位置；或者在阅读文章时，选中文本后自动弹出分享、复制选项等。

## 事件概述

事件可以理解为是JavaScript侦测到的行为。例如当用户鼠标单击页面、鼠标滑过某个 区域等动作。它对实现网页的交互起着非常重要的作用。与事件有关的相关概念如下：

1. 事件处理程序：

事件处理程序指的是JavaScript响应用户行为所执行的程序代码。通常是一个函数。

2）事件驱动式：

事件驱动式是指，在Web页面中JavaScrip的事件，侦测到的用户行为（如鼠标单击、鼠标移入等），并执行相应的事件处理程序的过程。

1. 事件流：

事件流指的是，事件发生时，会在元素节点与DOM根节点之间按照特定的顺序进行传播，这个事件传播的过程就是事件流。网景（Netscape）和微软（Microsoft）IE浏览器对于事件流的传播顺序提供了两种不同的解决方案。如下：

1. 事件捕获方式（网景）：它指的是事件流传播的顺序从DOM树的根节点到发生事件的元素节点。
2. 事件冒泡方式（微软）：它指的是事件流传播的顺序应该是从发生事件的元素节点到DOM树的根节点。

W3C对网景和微软提出的方案进行了中和处理，规定了事件发生后，首先实现事件捕获，但不会对事件进行处理；然后进行到目标阶段，执行当前元素对象的事件处理程序，但它会被看成是冒泡阶段的一部分；最后实现事件的冒泡，逐级对事件进行处理。

## 事件的绑定方式

事件绑定指的是为某个元素对象的事件绑定事件处理程序。在JavaScript中提供了3种 事件的绑定方式，分别是行内绑定式、动态绑定式和事件监听的方式。

1. **行内绑定式：**

事件的行内绑定式是通过HTML标签的属性设置实现的。语法格式为：

**<标签 事件=”事件处理程序”>**

其中，标签可以是任意的HTML标签；事件由on和事件名称组成，如单击事件对应onclick；事件处理程序是JavaScript中的函数。

由于在开发中提倡JavaScript和HTML代码的分离，所以不建议使用此种方式。

1. **动态绑定式：**

动态绑定式很好的解决了JavaScript代码与HTML代码混合编写的问题。动态绑定式是在JavaScript中获取相应标签的DOM元素对象，为DOM对象添加事件和事件处理程序。其语法格式为：

**DOM元素对象.事件=事件处理程序 ;**

·事件处理程序通常是一个匿名函数或有名函数。

·行内绑定式与动态绑定式除了实现的语法不同以外，在事件处理程序中关键字

this的指向也不同，行内绑定式的事件处理程序中this关键字指向window对象，

而动态绑定式的事件处理程序中this关键字指向当前正在操作的DOM元素对象。

·除此之外，行内绑定式和动态绑定是最原始的事件模型提供的事件绑定方式，在

该模型中没有事件流的概念，也就是说事件不能够传播。因此，同一个DOM对

象的同一个事件只能有一个事件处理程序。

1. **事件监听：**

行内绑定式与动态绑定式没有事件流的概念，事件不能传播。因此DOM对象 的同一个事件只能有一个事件处理程序。为了给同一个事件添加多个事件处理程序，事件模型中引入了事件流的概念，可以让DOM对象通过事件监听的方式实现事件的绑定。由于不同浏览器采用的事件流实现方式不同，事件监听的实现存在兼容性问题。通常根据浏览器的内核可以划分为两大类，一类是早期版本的IE浏览器（如IE6~8），一类遵循W3C标准的浏览器。

1）早期版本的IE浏览器中，事件监听的语法格式如下：

**DOM对象.attachEvent( type , callback ) ;**

参数type表示的是事件类型，如onclick；参数callback表示事件的处理程序。具体示例如下：



早期版本IE浏览器的事件处理程序按照添加顺序倒序执行，因此先输出two，然后输出one。

2）标准浏览器包括IE8版本以上的IE浏览器（如IE9~11），新版的FireFox、Chrome

等浏览器。具体语法格式如下：

**DOM对象.addEventListener( type , callback , [capture] ) ;**

参数type表示事件类型，如click（不需要加上on）；参数callback表示事件的处理程序；参数capture默认值为false，表示在冒泡阶段完成事件处理，将其设置为true时，表示在捕捉阶段完成事件处理。示例如下：

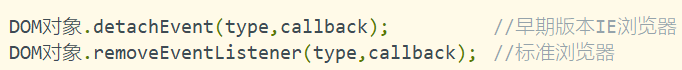


标准浏览器的事件处理程序按照添加顺序正序执行，因此输出结果依次是one和two。

3）值得一提的是，如果事件监听的处理程序是一个有名函数时，开发中可根据实

际需求移出DOM对象的事件监听。同样，事件监听处理程序的移出也需要考

虑兼容性问题，具体语法如下：



参数type的设置要与添加事件监听的事件类型相同，参数callback表示事件处理程序的名称，即函数名。

# 事件对象

在JavaScript中，当事件发生时，会产生一个事件对象event，这个对象中包含着所有与事件相关的信息，包括发生事件的DOM元素、事件的类型以及其他与特定事件相关的信息。例如，因键盘发生事件时，事件对象中会包括按下键的键值等相关信息；当发生鼠标移动事件时，事件对象中会包括鼠标位置（横纵坐标）等相关的信息。

## 获取事件对象

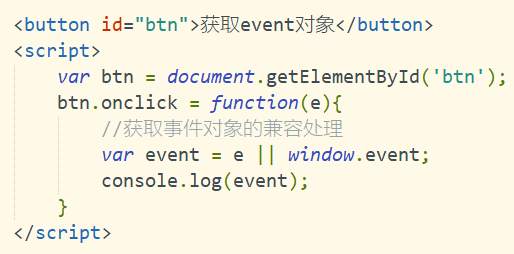
虽然所有浏览器都支持事件对象event，但是不同浏览器获取事件对象的方式不同。在标准浏览器中会将一个event对象直接传入到事件处理程序中，而早期版本的IE浏览器（IE6~8）中，仅能通过window.event才能获取事件对象。所以在开发中，一般将两种获取事件对象的方式结合使用，语法如下：

**DOM对象.事件类型 =function( e ){**

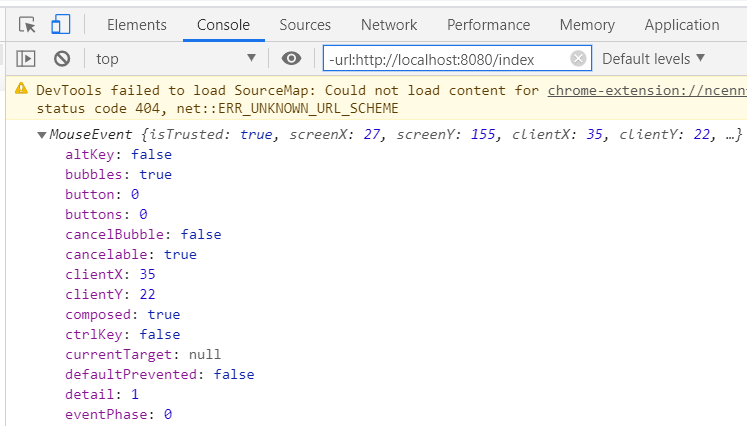
**var event = e || window . event ;**

**}**

语法中，event即获得的事件对象，以获取button按钮单击事件的事件对象为例进行演示：



上述代码中，事件处理函数中传递的参数e表示的就是事件对象event，6行通过或运算符实现不同浏览器间获取事件对象兼容的处理。若是标准浏览器，则可以直接通过e获取事件对象，否则若是早期版本的浏览器，则需要通过window.event才能获取事件对象。单击按钮触发事件处理程序，从控制台查看输出：



展开事件对象即可看到其包含的所有属性和方法。

## 常用属性和方法

当发生事件后，事件对象中不仅包含着与特定事件相关的信息，还包含一些所有事件都 有的属性和方法。常用的数学和方法如下：

|  |  |  |
| --- | --- | --- |
| **分类** | **属性/方法** | **描述** |
| 公有属性 | type | 返回事件的类型，如click |
| 标准浏览器事件对象 | target | 返回触发此事件的元素（事件的目标节点） |
| currentTarget | 返回其事件监听器触发该事件的元素 |
| bubbles | 表示事件是否是冒泡事件类型 |
| cancelable | 表示事件是否取消默认动作 |
| eventPhase | 返回事件传播的当前阶段。1表示捕获阶段，2表示处于目标节点，3表示冒泡阶段 |
| stopPropagation( ) | 阻止事件冒泡 |
| preventDefault( ) | 阻止默认行为 |
| 早期IE版本浏览器 | srcElement | 返回触发此事件的元素（事件的目标节点） |
| cancelBubble | 阻止事件冒泡，默认为false表示允许，设置true表示阻止 |
| returnValue | 阻止默认行为，默认为true表示允许，设置false表示阻止 |

type是标准浏览器和早期版本IE浏览器的事件对象的公有属性，通过该属性可以获取发生事件的类型，如click。

示例：获取触发事件的元素



从示例看出，通过事件对象的target属性或srcElement即可获取触发事件的元素相关信息。

## 阻止事件冒泡

1）HTML代码如下：

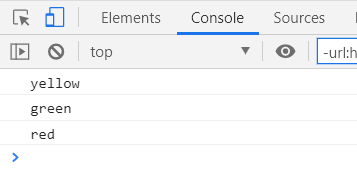


上述代码中，当用户单击不用颜色的<div>块时，则在控制台输出对应的颜色值。

1. 添加CSS样式后显示结果如下：



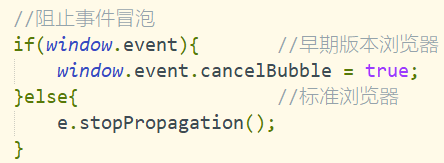
3）以单击黄色区域为例，在控制台查看输出结果：



从输出结果可以看出，与分析结果不同，这是由于默认情况下，事件是按照事件冒泡的方式进行处理的。因此，当用户单击黄色区域时，它不仅触发了黄色<div>块的单击事件，逐层向上触发了id为green和id为red的<div>块的单击事件，知道最顶层元素，所以最后的输出结果一次为yellow、green、red。

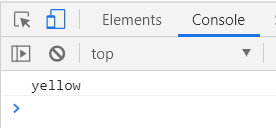
1. 开发中若要禁止事件冒泡，可以利用事件对象调用stopPropagation( )方法和

cancelBuddle属性，实现禁止所有浏览器的事件冒泡行为。例如，为上述所有单击事件的事件处理程序，添加参数e，用户获取事件对象，通过事件对象禁止事件冒泡：



 添加后代码片段如下：

修改完成后，再次点击黄色区域，控制台输出结果如下：



可以发现，事件冒泡被阻止。最后总结出，在所有浏览器器中阻止事件冒泡（在事件处理程序中）的语法如下：

**if( window.event ){**

**window.event.cancelBubble = true ;**

**} else {**

**e.stopPropagation( ) ;**

**}**

## 阻止事件默认行为

在HTML中，有些元素标签本身拥有一些特殊的行为。例如单击<a>标签后，会自动跳转到href属性指定的URL链接；单击表单的submit按钮后，会自动将表单数据提交到指定的服务器端页面处理。因此，我们把这些标签具有的行为称为默认行为。

但在实际开发中，为了使程序更加严谨，想要确定含有默认行为的标签符合要求后，才 能执行默认行为时，可利用事件对象的preventDefault( )方法和returnValue属性，禁止 所有浏览器执行元素的默认行为。其语法格式为：  
 **DOM元素.事件类型=function(e){**

**if ( window.event ){**

**window.event.returnValue=false ;**

**}else{**

**e.preventDefault( ) ;**

**}**

**}**

以禁止<a>标签的链接为例进行演示：



上述代码中，通过if…else语句完成浏览器的兼容处理，然后分别调用对应的属性和方法实现禁用元素默认行为的设置。完成上述设置后，在任何浏览器中单击“百度一下”，浏览器都不会自动请求指定的URL地址<http://www.baidu.com>了。

# 事件分类

在JavaScript中的事件大致分为5类，即页面事件、焦点事件、鼠标事件、键盘事件和 表单事件。

## 页面事件

在项目开发中，经常需要JavaScript对网页中的DOM元素进行操作，而页面的加载又是按照代码的编写顺序，从上到下依次执行的。因此，若在页面还未加载完成的情况下，就使用JavaScript操作DOM元素，会出现语法错误。如：

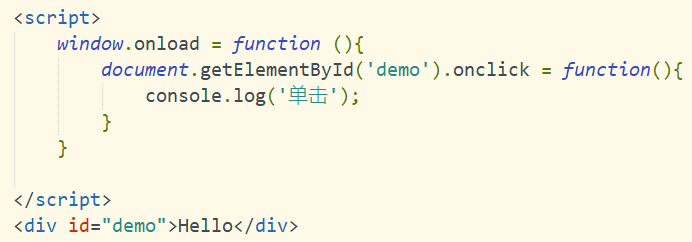


上述代码中，为id为demo的元素添加单击事件是失败的，因为id为demo的元素定义在其后面，JavaScript在执行时，后面的代码还未加载，无法获取到id为demo的元素。为了解决此类问题，JavaScript提供了页面事件，可以改变JavaScript代码的执行时机。

|  |  |
| --- | --- |
| **事件名称** | **事件触发时机** |
| load | 当页面载入完毕后触发 | |
| unload | 当页面关闭时触发 | |

load事件用于body内所有标签都加载完成后才触发，又因其无需考虑页面加载顺序的问题，常常在开发具体功能时添加；unload事件用于页面关闭时触发，开发中用于清除引用，避免内存泄漏。

load事件使用示例：



如此一来，只有当HTML文本全部加载到浏览器中时，才会触发load事件，执行其中的JavaScript代码。解决页面加载顺序问题的另一个方法是将JavaScript代码写在body标签中的最下面。

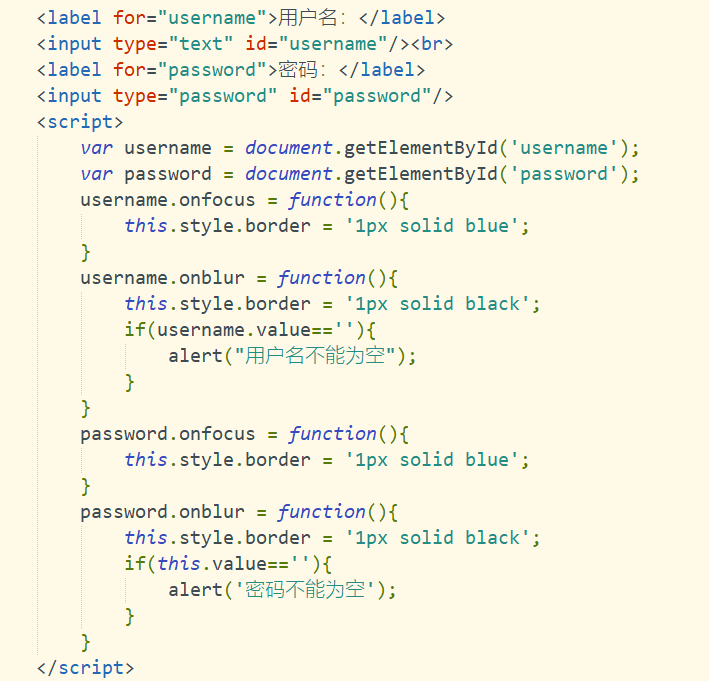
## 焦点事件

在Web开发中，焦点事件常用于表单验证功能。例如，文本框捕获焦点改变文本的样 式，文本框失去焦点时验证文本框内输入的数据是否符合要求等。常用的焦点事件如下：

|  |  |
| --- | --- |
| **事件类型** | **事件触发时机** |
| focus | 当获得焦点时触发（不会冒泡） |
| blur | 当失去焦点时触发（不会冒泡） |

示例：当用户点击输入框输入用户名或密码时，输入框样式改变，当焦点离开输入框后，

将样式还原，并对用户输入的内容进行验证，验证是否输入为空。



## 鼠标事件

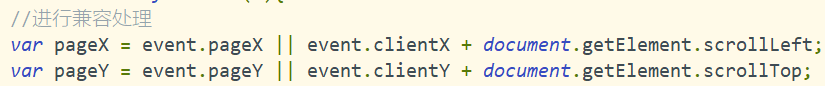
鼠标事件是Web开发中最常用的一类事件。例如，鼠标滑过时，切换Tab标签栏显示的内容、利用鼠标拖拽状态框，调整它的显示位置等。

|  |  |
| --- | --- |
| **事件类型** | **事件触发时机** |
| **click** | **当按下任意鼠标键并释放时触发（单击触发）** |
| dblclick | 当鼠标双击时触发 |
| **mouseover** | **当鼠标进入时触发** |
| **mouseout** | **当鼠标离开时触发** |
| **change** | **当内容发生改变时触发，一般多用于<select>对象** |
| mousedown | 当按下任意鼠标按键时触发 |
| mouseup | 当释放任意鼠标键时触发 |
| mousemove | 在元素内当鼠标移动时触发 |

在项目开发中还经常涉及一些常用的鼠标属性，用来获取当前鼠标的位置信息。

|  |  |
| --- | --- |
| 位置属性（只读） | 描述 |
| clientX | 鼠标指针位于浏览器页面当前窗口可视区的水平坐标（X轴坐标） |
| clientY | 鼠标指针位于浏览器页面当前窗口可视区的垂直坐标（Y坐标） |
| pageX | 鼠标指针位于文档的水平坐标（X轴坐标），IE6~8不兼容 |
| pageY | 鼠标指针位于文档的垂直坐标（Y轴坐标），IE6~8不兼容 |
| screenX | 鼠标指针位于屏幕的水平坐标（X坐标） |
| screenY | 鼠标指针位于屏幕的垂直坐标（Y坐标） |

IE6~8浏览器不兼容pageX和pageY属性。因此项目开发时需要对IE6~8浏览器进行兼容处理，如下：



从以上代码可知，鼠标在文档中的坐标等于鼠标在当前窗口中的坐标加上滚动条卷去的的文本长度。

## 键盘事件

键盘事件是指在使用键盘时触发的实际。例如，用户按ESC键关闭打开的状态栏，按Enter键直接完成光标的上下切换等。常用的键盘事件如下：

|  |  |
| --- | --- |
| **事件类型** | **事件触发时机** |
| keypress | 键盘按键（Shift、Fn、CapsLock等非字符键除外）按下时触发 |
| keydown | 键盘按键按下时触发 |
| keyup | 键盘按键弹起时触发 |

需要注意的是，keypress事件保存的按键值是ASCII码，keydown和keyup事件保存的按键值是虚拟键码。键盘的键码可以百度查找，如下。

<https://blog.csdn.net/qq_31754523/article/details/95493776>

示例：按Enter键完成光标的上下切换



## **表单事件**

表单事件指的是对Web表单操作时发生的事件。例如，表单提交前对表单的验证、表 单重置时的确认操作等。常用的表单事件如下：

|  |  |
| --- | --- |
| **事件名称** | **事件触发时机** |
| submit | 当表单提交时触发 |
| reset | 当表单重置时触发 |

submit事件的实现通常要绑定到<form>标签上，在用户单击submit按钮提交表单时触发。reset事件用于单击重置按钮时触发，这两个事件的返回值若是false则会取消默认的操作，否则将执行默认操作。

示例：在提交表单时对数据进行验证，在重置表单数据时提示是否确认重置。

