事件

前提: 基本电脑知识, 对HTML和CSS的基本了解,及 JavaScript first steps.

目标: 了解事件的基本理论,它们怎么在浏览器上运行的, 以及在不同的编程环境下事件怎么不同.

# 事件基本概念

事件是在编程时系统内发生的动作或者发生的事情— 系统会在事件出现的时候触发某种信号并且会提供一个自动加载某种动作（列如：运行一些代码）的机制

在Web中, 事件在浏览器窗口中被触发并且通常被绑定到窗口内部的特定部分 — 可能是一个元素、一系列元素、被加载到这个窗口的 HTML 代码或者是整个浏览器窗口。举几个可能发生的不同事件：

* 用户在某个元素上点击鼠标或悬停光标。
* 用户在键盘中按下某个按键。
* 用户调整浏览器的大小或者关闭浏览器窗口。
* 一个网页停止加载。
* 提交表单。
* 播放、暂停、关闭视频。
* 发生错误。

事件参考：

<https://developer.mozilla.org/zh-CN/docs/Web/Events>

# 使用网页事件的方式

## 事件处理器属性

var btn = document.querySelector('button');

btn.onclick = function() {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

document.body.style.backgroundColor = rndCol;

}

这个 [onclick](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onclick) 是被用在这个情景下的事件处理器的属性，它就像 button 其他的属性（如 [btn.textContent](https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent), or [btn.style](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style)), 但是有一个特别的地方——将一些代码赋值给它的时候，只要事件触发代码就会运行。

也可以将一个有名字的函数赋值给事件处理参数

var btn = document.querySelector('button');

function bgChange() {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

document.body.style.backgroundColor = rndCol;

}

btn.onclick = bgChange;

练习：

将btn.onclick 依次换成其他值，在浏览器中观察效果。

* [btn.onfocus](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onfocus)及[btn.onblur](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onblur) — 颜色将于按钮被置于焦点或解除焦点时改变（尝试使用Tab移动至按钮上，然后再移开）。这些通常用于显示有关如何在置于焦点时填写表单字段的信息，或者如果表单字段刚刚填入不正确的值，则显示错误消息。
* [btn.ondblclick](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/ondblclick) — 颜色将仅于按钮被双击时改变。
* [window.onkeypress](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onkeypress), [window.onkeydown](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onkeydown), [window.onkeyup](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onkeyup) — 当按钮被按下时颜色会发生改变. keypress 指的是通俗意义上的按下按钮 (按下并松开), 而 keydown 和 keyup 指的是按键动作的一部分,分别指按下和松开. 注意如果你将事件处理器添加到按钮本身，它将不会工作 — 我们只能将它添加到代表整个浏览器窗口的 [window](https://developer.mozilla.org/en-US/docs/Web/API/Window)对象中。
* [btn.onmouseover](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onmouseover) 和 [btn.onmouseout](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onmouseout) — 颜色将会在鼠标移入按钮上方时发生改变, 或者当它从按钮移出时.

一些事件非常通用，几乎在任何地方都可以用（比如 onclick 几乎可以用在几乎每一个元素上），然而另一些元素就只能在特定场景下使用，比如我们只能在 video 元素上使用 [onplay](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/GlobalEventHandlers.onplay) 。

## 行内事件处理器 - 请勿使用

<button onclick="bgChange()">Press me</button>

function bgChange() {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

document.body.style.backgroundColor = rndCol;

}

在Web上注册事件处理程序的最早方法是类似于上面所示的**事件处理程序HTML属性**(也称为内联事件处理程序)—属性值实际上是当事件发生时要运行的JavaScript代码。上面的例子中调用一个在[<script>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/script)元素在同一个页面上，但也可以直接在属性内插入JavaScript，例如：

<button onclick="alert('你好，这是个老式事件处理器');">Press me</button>

**注释**: 将编程逻辑与内容分离也会使站点对搜索引擎更加友好。

## addEventListener()和removeEventListener()

新的事件触发机制[addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener)。这个函数和事件处理属性是类似的，但是语法略有不同。重写上面的随机颜色背景代码：

var btn = document.querySelector('button');

function bgChange() {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

document.body.style.backgroundColor = rndCol;

}

btn.addEventListener('click', bgChange);

在addEventListener() 函数中, 我们具体化了两个参数——事件名称，回应事件的函数的代码。也可以将这些代码全部放到一个匿名函数中。

btn.addEventListener('click', function() {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

document.body.style.backgroundColor = rndCol;

});

使用addEventListener 添加事件侦听器可以通过[removeEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/removeEventListener)，方法移除事件监听器。例如，下面的代码将会移除上个代码块中的事件监听器：

btn.removeEventListener('click', bgChange);

在这个简单的、小型的项目中可能不是很有用，但是在大型的、复杂的项目中就非常有用了，可以非常高效地清除不用的事件处理器，另外在其他的一些场景中也非常有效——比如在不同环境下运行不同的事件处理器，只需要恰当地删除或者添加事件处理器即可。

也可以给同一个监听器注册多个处理器，下面这种方式不能实现这一点：

myElement.onclick = functionA;

myElement.onclick = functionB;

第二行会覆盖第一行，但是下面这种方式就会正常工作了：

myElement.addEventListener('click', functionA);

myElement.addEventListener('click', functionB);

当元素被点击时两个函数都会工作：

## 该使用哪种事件注册机制？

在三种机制中,不使用HTML事件处理程序属性 - 这些属性已经过时了，而且也是不好的做法。另外两种在简单应用中可以互换相对可互换的。

# 事件进阶

本我们将简要介绍一些与事件相关的高级概念。在这一点并不需要完全理解透彻，但它可能有助于你解释一些经常会遇到的代码模式。

## 事件对象

在事件处理函数内部，使用一个固定指定名称的参数，例如event，evt或简单的e。 这被称为**事件对象**，它被自动传递给事件处理函数，以提供额外的功能和信息。随机颜色示例：

function bgChange(e) {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

e.target.style.backgroundColor = rndCol;

console.log(e);

}

btn.addEventListener('click', bgChange);

在这里，事件触发函数中包括一个事件对象e，并在函数中设置背景颜色样式在e.target上 - 它指的是按钮本身。 事件对象 e 的target属性始终是事件刚刚发生的元素的引用。 所以在这个例子中，是按钮上设置一个随机的背景颜色，而不是页面。

注释: 可以使用任意标识符作为事件对象 -通常常使用 e / evt / event。

要在多个元素上设置相同的事件处理程序时，e.target非常有用，并且在发生事件时对所有元素执行某些操作.  例如：为每一个元素都添加一个onclick单击事件，每当它们点击时就会为背景添加一个随机颜色。

var divs = document.querySelectorAll('div');

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

divs[i].onclick = function(e) {

e.target.style.backgroundColor = bgChange();

}

}

## 阻止默认行为-- e.preventDefault();

有时希望事件不执行它的默认行为。 最常见的例子是Web表单，例如自定义注册表单。 当填写详细信息并按提交按钮时，自然行为是将数据提交到服务器上的指定页面进行处理，并将浏览器重定向到某种“成功消息”页面（或 相同的页面，如果另一个没有指定。）

当用户没有正确提交数据时，希望停止提交信息给服务器，并一个错误提示，告诉用户什么做错了，以及需要做些什么来修正错误。

<form>

<div>

<label for="userName">姓名: </label>

<input id="userName" type="text">

</div>

<div>

<label for="psw">密码: </label>

<input id="psw" type="text">

</div>

<div>

<input id="submit" type="submit">

</div>

</form>

<p></p>

这里我们用一个onsubmit事件处理程序（在提交的时候，在一个表单上发起submit事件）来实现一个非常简单的检查，用于测试文本字段是否为空。 如果是，我们在事件对象上调用preventDefault()函数，这样就停止了表单提交，然后在我们表单下面的段落中显示一条错误消息，告诉用户什么是错误的：

var form = document.querySelector('form');

var fname = document.getElementById('fname');

var lname = document.getElementById('lname');

var submit = document.getElementById('submit');

var para = document.querySelector('p');

form.onsubmit = function(e) {

if (fname.value === '' || lname.value === '') {

e.preventDefault();

para.textContent = 'You need to fill in both names!';

}

}

显然，这是一种非常弱的表单验证——例如，用户输入空格或数字提交表单，表单验证并不会阻止用户提交——这不是我们例子想要达到的目的。输出如下:

**Note**: 查看完整的源代码 [preventdefault-validation.html](https://github.com/mdn/learning-area/blob/master/javascript/building-blocks/events/preventdefault-validation.html) (也可以 [running live](http://mdn.github.io/learning-area/javascript/building-blocks/events/preventdefault-validation.html) )

## 事件冒泡及捕获

事件冒泡和捕捉是两种机制，主要描述当在一个元素上有两个相同类型的事件处理器被激活会发生什么。

一个非常简单的例子，它显示和隐藏一个包含<video>元素的<div>元素：

<button>Display video</button>

<div class="hidden">

<video>

<source src="rabbit320.mp4" type="video/mp4">

<source src="rabbit320.webm" type="video/webm">

<p>Your browser doesn't support HTML5 video. Here is a <a href="rabbit320.mp4">link to the video</a> instead.</p>

</video>

</div>

当‘’button‘’元素按钮被单击时，将显示视频，它是通过将改变<div>的class属性值从hidden变为showing(这个例子的CSS包含两个class，它们分别控制这个<div>盒子在屏幕上显示还是隐藏。)：

btn.onclick = function() {

videoBox.setAttribute('class', 'showing');

}

然后我们再添加几个onclick事件处理器，第一个添加在<div>元素上，第二个添加在<video>元素上。这个想法是当视频(<video>）外 <div>元素内这块区域被单击时，这个视频盒子应该再次隐藏；当单击视频(<video>）本身，这个视频将开始播放。

videoBox.onclick = function() {

videoBox.setAttribute('class', 'hidden');

};

video.onclick = function() {

video.play();

};

但是有一个问题 - 当点击video开始播放的视频时，它会在同一时间导致<div>也被隐藏。 这是因为video在<div>之内 - video是<div>的一个子元素 - 所以点击video实际上是同时也运行<div>上的事件处理程序。

### 对事件冒泡和捕捉的解释

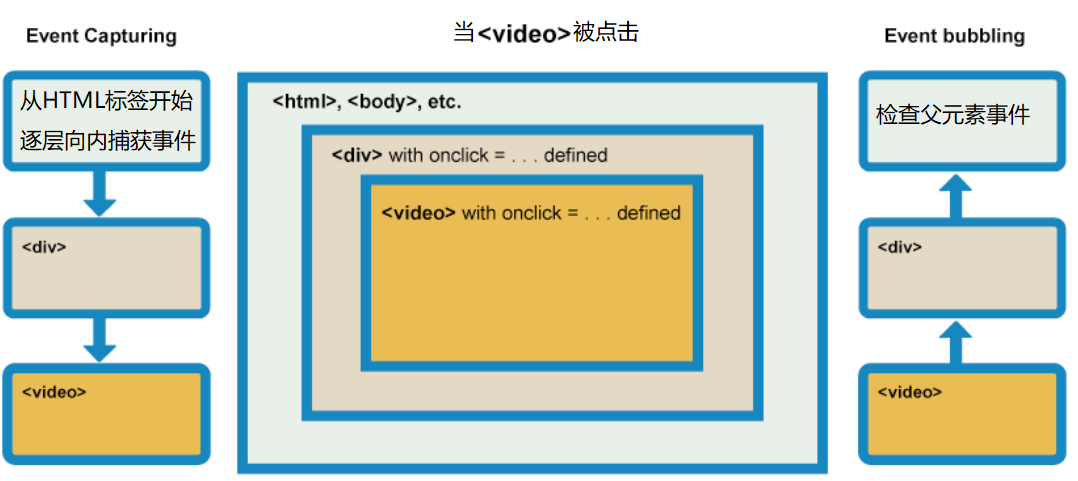
当一个事件发生在具有父元素的元素上(例如，在上个例子中是<video>元素)时，浏览器运行两个不同的阶段 - 捕获阶段和冒泡阶段。

在捕获阶段：

* 浏览器检查元素的最外层祖先<html>，是否在捕获阶段中注册了一个onclick事件处理程序，如果是，则运行它。
* 然后，它移动到<html>中的下一个元素，并执行相同的操作，然后是下一个元素，依此类推，直到到达实际点击的元素。

在冒泡阶段:

* 浏览器检查实际点击的元素是否在冒泡阶段中注册了一个onclick事件处理程序，如果是，则运行它
* 然后它移动到下一个直接的祖先元素，并做同样的事情，然后是下一个，等等，直到它到达<html>元素。



默认情况下，所有事件处理程序都在冒泡阶段进行注册。当单击视频时，这个单击事件从 <video>元素向外冒泡直到<html>元素。

沿着这个事件冒泡线路：

* 发现了video.onclick...事件处理器并且运行它，因此这个视频<video>第一次开始播放。
* 接着发现了（往外冒泡找到的） videoBox.onclick...事件处理器并且运行它，因此这个视频<video>也隐藏起来了。

### 用stopPropagation()修复问题

标准事件对象具有可用的名为 [stopPropagation()](https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation)的函数, 当在事件对象上调用该函数时，它只会让当前事件处理程序运行，但事件不会在**冒泡**链上进一步扩大，因此将不会有更多事件处理器被运行(不会向上冒泡)。

video.onclick = function(e) {

e.stopPropagation();

video.play();

};

**注解**: 为什么我们要弄清楚捕捉和冒泡呢？那是因为，在过去糟糕的日子里，浏览器的兼容性比现在要小得多，Netscape（网景）只使用事件捕获，而Internet Explorer只使用事件冒泡。当W3C决定尝试规范这些行为并达成共识时，最终被应用在现在浏览器里。

**注解**: 如上所述，默认情况下，所有事件处理程序都是在冒泡阶段注册的，这在大多数情况下更有意义。如果想在捕获阶段注册一个事件，可以通过使用[addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener)注册您的处理程序，并将可选的第三个属性设置为true。

### 事件委托

冒泡还允许我们利用事件委托——这个概念依赖于这样一个事实,如果你想要在大量子元素中单击任何一个都可以运行一段代码，您可以将事件监听器设置在其父点上，并将事件监听器气泡的影响设置为每个子点，而不是每个子点单独设置事件监听器。

让每个列表点击时弹出一条信息，可以将click单击事件监听器设置在父元素<ul>上，它将会冒泡到列表项上。

**结论**

事件并不是JavaScript的核心部分——它们是在浏览器Web APIs中定义的。

理解JavaScript在不同环境下使用不同的事件模型很重要