

第 25 章 事件对象

学习要点:

- 1.事件对象
- 2.鼠标事件
- 3.键盘事件
- 4.W3C 与 IE

主讲教师: 李炎恢

合作网站: <http://www.ibeifeng.com>

讲师博客: <http://hi.baidu.com/李炎恢>

JavaScript 事件的一个重要方面是它们拥有一些相对一致的特点, 可以给你的开发提供更多的强大功能。最方便和强大的就是事件对象, 他们可以帮你处理鼠标事件和键盘敲击方面的情况, 此外还可以修改一般事件的捕获/冒泡流的函数。

一. 事件对象

事件处理函数的一个标准特性是, 以某些方式访问的事件对象包含有关于当前事件的上下文信息。

事件处理三部分组成: 对象.事件处理函数=函数。例如: 单击文档任意处。

```
document.onclick = function () {  
    alert('Lee');  
};
```

PS: 以上程序的名词解释: click 表示一个事件类型, 单击。onclick 表示一个事件处理函数或绑定对象的属性(或者叫事件监听器、侦听器)。document 表示一个绑定的对象, 用于触发某个元素区域。function()匿名函数是被执行的函数, 用于触发后执行。

除了用匿名函数的方法作为被执行的函数, 也可以设置成独立的函数。

```
document.onclick = box; //直接赋值函数名即可, 无须括号  
function box() {  
    alert('Lee');  
}
```

this 关键字和上下文

在面向对象那章我们了解到: 在一个对象里, 由于作用域的关系, this 代表着离它最近对象。

```
var input = document.getElementsByTagName('input')[0];  
input.onclick = function () {  
    alert(this.value); //HTMLInputElement, this 表示 input 对象  
};
```

从上面的拆分, 我们并没有发现本章的重点: 事件对象。那么事件对象是什么? 它在哪里呢? 当触发某个事件时, 会产生一个事件对象, 这个对象包含着所有与事件有关的信息。

包括导致事件的元素、事件的类型、以及其它与特定事件相关的信息。

事件对象，我们一般称作为 event 对象，这个对象是浏览器通过函数把这个对象作为参数传递过来的。那么首先，我们就必须验证一下，在执行函数中没有传递参数，是否可以得到隐藏的参数。

```
function box() { //普通空参函数
    alert(arguments.length); //0, 没有得到任何传递的参数
}
```

```
input.onclick = function () { //事件绑定的执行函数
    alert(arguments.length); //1, 得到一个隐藏参数
};
```

通过上面两组函数中,我们发现,通过事件绑定的执行函数是可以得到一个隐藏参数的。说明,浏览器会自动分配一个参数,这个参数其实就是 event 对象。

```
input.onclick = function () {
    alert(arguments[0]); //MouseEvent, 鼠标事件对象
};
```

上面这种做法比较累,那么比较简单的做法是,直接通过接收参数来得到即可。

```
input.onclick = function (evt) { //接受 event 对象, 名称不一定非要 event
    alert(evt); //MouseEvent, 鼠标事件对象
};
```

直接接收 event 对象,是 W3C 的做法,IE 不支持,IE 自己定义了一个 event 对象,直接在 window.event 获取即可。

```
input.onclick = function (evt) {
    var e = evt || window.event; //实现跨浏览器兼容获取 event 对象
    alert(e);
};
```

二. 鼠标事件

鼠标事件是 Web 上面最常用的一类事件,毕竟鼠标还是最主要的定位设备。那么通过事件对象可以获取到鼠标按钮信息和屏幕坐标获取等。

1. 鼠标按钮

只有在主鼠标按钮被单击时(常规一般是鼠标左键)才会触发 click 事件,因此检测按钮的信息并不是必要的。但对于 mousedown 和 mouseup 事件来说,则在其 event 对象存在一个 button 属性,表示按下或释放按钮。

非 IE(W3C)中的 button 属性

值	说明
0	表示主鼠标按钮(常规一般是鼠标左键)
1	表示中间的鼠标按钮(鼠标滚轮按钮)

2	表示次鼠标按钮(常规一般是鼠标右键)
---	--------------------

IE 中的 button 属性

值	说明
0	表示没有按下按钮
1	表示主鼠标按钮(常规一般是鼠标左键)
2	表示次鼠标按钮(常规一般是鼠标右键)
3	表示同时按下了主、次鼠标按钮
4	表示按下了中间的鼠标按钮
5	表示同时按下了主鼠标按钮和中间的鼠标按钮
6	表示同时按下了次鼠标按钮和中间的鼠标按钮
7	表示同时按下了三个鼠标按钮

PS: 在绝大部分情况下, 我们最多只使用主次中三个单击键, IE 给出的其他组合键一般无法使用上。所以, 我们只需要做上这三种兼容即可。

```
function getButton(evt) { //跨浏览器左中右键单击相应
    var e = evt || window.event;
    if (evt) { //Chrome 浏览器支持 W3C 和 IE
        return e.button; //要注意判断顺序
    } else if (window.event) {
        switch(e.button) {
            case 1 :
                return 0;
            case 4 :
                return 1;
            case 2 :
                return 2;
        }
    }
}

document.onmouseup = function (evt) { //调用
    if (getButton(evt) == 0) {
        alert('按下了左键! ');
    } else if (getButton(evt) == 1) {
        alert('按下了中键! ');
    } else if (getButton(evt) == 2) {
        alert('按下了右键! ');
    }
};
```

2. 可视区及屏幕坐标

事件对象提供了两组来获取浏览器坐标的属性，一组是页面可视区左边，另一组是屏幕坐标。

坐标属性

属性	说明
clientX	可视区 X 坐标，距离左边框的位置
clientY	可视区 Y 坐标，距离上边框的位置
screenX	屏幕区 X 坐标，距离左屏幕的位置
screenY	屏幕区 Y 坐标，距离上屏幕的位置

```
document.onclick = function (evt) {  
    var e = evt || window.event;  
    alert(e.clientX + ',' + e.clientY);  
    alert(e.screenX + ',' + e.screenY);  
};
```

3. 修改键

有时，我们需要通过键盘上的某些键来配合鼠标来触发一些特殊的事件。这些键为：Shift、Ctrl、Alt 和 Meta(Windows 中就是 Windows 键，苹果机中是 Cmd 键)，它们经常被用来修改鼠标事件和行为，所以叫修改键。

修改键属性

属性	说明
shiftKey	判断是否按下了 Shift 键
ctrlKey	判断是否按下了 ctrlKey 键
altKey	判断是否按下了 alt 键
metaKey	判断是否按下了 windows 键，IE 不支持

```
function getKey(evt) {  
    var e = evt || window.event;  
    var keys = [];  
  
    if (e.shiftKey) keys.push('shift'); //给数组添加元素  
    if (e.ctrlKey) keys.push('ctrl');  
    if (e.altKey) keys.push('alt');  
  
    return keys;  
}
```

```
document.onclick = function (evt) {  
    alert(getKey(evt));  
};
```

三. 键盘事件

用户在使用键盘时会触发键盘事件。“DOM2 级事件”最初规定了键盘事件，结果又删除了相应的内容。最终还是使用最初的键盘事件，不过 IE9 已经率先支持“DOM3”级键盘事件。

1. 键码

在发生 `keydown` 和 `keyup` 事件时，`event` 对象的 `keyCode` 属性中会包含一个代码，与键盘上一个特定的键对应。对数字字母字符集，`keyCode` 属性的值与 ASCII 码中对应小写字母或数字的编码相同。字母中大小写不影响。

```
document.onkeydown = function (evt) {  
    alert(evt.keyCode); //按任意键，得到相应的 keyCode  
};
```

不同的浏览器在 `keydown` 和 `keyup` 事件中，会有一些特殊的情况：

在 Firefox 和 Opera 中，分号键时 `keyCode` 值为 59，也就是 ASCII 中分号的编码；而 IE 和 Safari 返回 186，即键盘中按键的键码。

PS：其他一些特殊情况由于浏览器版本太老和市场份额太低，这里不做补充。

2. 字符编码

Firefox、Chrome 和 Safari 的 `event` 对象都支持一个 `charCode` 属性，这个属性只有在发生 `keypress` 事件时才包含值，而且这个值是按下的那个键所代表字符的 ASCII 编码。此时的 `keyCode` 通常等于 0 或者也可能等于所按键的编码。IE 和 Opera 则是在 `keyCode` 中保存字符的 ASCII 编码。

```
function getCharCode(evt) {  
    var e = evt || window.event;  
    if (typeof e.charCode == 'number') {  
        return e.charCode;  
    } else {  
        return e.keyCode;  
    }  
}
```

PS：可以使用 `String.fromCharCode()` 将 ASCII 编码转换成实际的字符。

`keyCode` 和 `charCode` 区别如下：比如当按下“a 键（重视是小写的字母）时，在 Firefox 中会获得

keydown: keyCode is 65 charCode is 0

keyup: keyCode is 65 charCode is 0

keypress: keyCode is 0 charCode is 97

在 IE 中会获得

keydown: keyCode is 65 charCode is undefined

keyup: keyCode is 65 charCode is undefined

keypress: keyCode is 97 charCode is undefined

而当按下 shift 键时，在 Firefox 中会获得

keydown: keyCode is 16 charCode is 0

keyup: keyCode is 16 charCode is 0

在 IE 中会获得

keydown: keyCode is 16 charCode is undefined

keyup: keyCode is 16 charCode is undefined

keypress: 不会获得任何的 charCode 值，因为按 shift 并没输入任何的字符，并且也不会触发 keypress 事务

PS: 在 keydown 事务里面，事务包含了 keyCode - 用户按下的按键的物理编码。

在 keypress 里，keyCode 包含了字符编码，即默示字符的 ASCII 码。如许的情势实用于所有的浏览器 - 除了火狐，它在 keypress 事务中的 keyCode 返回值为 0。

四. W3C 与 IE

在标准的 DOM 事件中，event 对象包含与创建它的特定事件有关的属性和方法。触发的事件类型不一样，可用的属性和方法也不一样。

W3C 中 event 对象的属性和方法

属性/方法	类型	读/写	说明
bubbles	Boolean	只读	表明事件是否冒泡
cancelable	Boolean	只读	表明是否可以取消事件的默认行为
currentTarget	Element	只读	其事件处理程序当前正在处理事件的那个元素
detail	Integer	只读	与事件相关的细节信息
eventPhase	Integer	只读	调用事件处理程序的阶段：1 表示捕获阶段，2 表示“处理目标”，3 表示冒泡阶段
preventDefault()	Function	只读	取消事件的默认行为。如果 cancelable 是 true，则可以使用这个方法
stopPropagation()	Function	只读	取消事件的进一步捕获或冒泡。如果 bubbles 为 true，则可以使用这个方法
target	Element	只读	事件的目标
type	String	只读	被触发的事件的类型
view	AbstractView	只读	与事件关联的抽象视图。等同于发生事件的 window 对象

IE 中 event 对象的属性

属性	类型	读/写	说明
cancelBubble	Boolean	读/写	默认值为 false, 但将其设置为 true 就可以取消事件冒泡
returnValue	Boolean	读/写	默认值为 true, 但将其设置为 false 就可以取消事件的默认行为
srcElement	Element	只读	事件的目标
type	String	只读	被触发的事件类型

在这里, 我们只看所有浏览器都兼容的属性或方法。首先第一个我们了解一下 W3C 中的 target 和 IE 中的 srcElement, 都表示事件的目标。

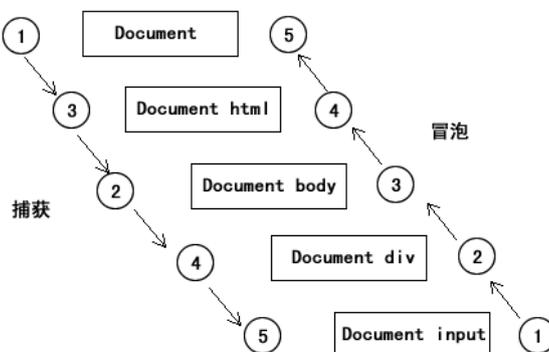
```
function getTarget(evt) {
    var e = evt || window.event;
    return e.target || e.srcElement; //兼容得到事件目标 DOM 对象
}

document.onclick = function (evt) {
    var target = getTarget(evt);
    alert(target);
};
```

事件流

事件流是描述的从页面接受事件的顺序, 当几个都具有事件的元素层叠在一起的时候, 那么你点击其中一个元素, 并不是只有当前被点击的元素会触发事件, 而层叠在你点击范围的所有元素都会触发事件。事件流包括两种模式: 冒泡和捕获。

事件冒泡, 是从里往外逐个触发。事件捕获, 是从外往里逐个触发。那么现代的浏览器默认情况下都是冒泡模型, 而捕获模式则是早期的 Netscape 默认情况。而现在的浏览器要使用 DOM2 级模型的事件绑定机制才能手动定义事件流模式。



```
document.onclick = function () {  
    alert('我是 document');  
};  
document.documentElement.onclick = function () {  
    alert('我是 html');  
};  
document.body.onclick = function () {  
    alert('我是 body');  
};  
document.getElementById('box').onclick = function () {  
    alert('我是 div');  
};  
document.getElementsByTagName('input')[0].onclick = function () {  
    alert('我是 input');  
};
```

在阻止冒泡的过程中，W3C 和 IE 采用的不同的方法，那么我们必须做一下兼容。

```
function stopPro(evt) {  
    var e = evt || window.event;  
    window.event ? e.cancelBubble = true : e.stopPropagation();  
}
```

感谢收看本次教程！

本课程是由北风网(ibeifeng.com)

瓢城 **Web** 俱乐部(yc60.com)联合提供：

本次主讲老师：李炎恢

我的博客：hi.baidu.com/李炎恢/

我的邮件：yc60.com@gmail.com