□□□□

□□

- □□□□□□□□ □□□□□□
- MouseEvent □□ □□
- MouseEvent □□□□□□□□ □□□□□□
  - MouseEvent.altKey□ MouseEvent.ctrlKey□ MouseEvent.metaKey□ MouseEvent.shiftKey
  - MouseEvent.button□ MouseEvent.buttons
  - MouseEvent.clientX□ MouseEvent.clientY
  - MouseEvent.movementX□ MouseEvent.movementY
  - MouseEvent.screenX□ MouseEvent.screenY
  - MouseEvent.offsetX□ MouseEvent.offsetY
  - MouseEvent.pageX□ MouseEvent.pageY
  - MouseEvent.relatedTarget
- MouseEvent □□□□□□□□ □□□□□□
  - MouseEvent.getModifierState()
- WheelEvent □□ □□
  - □□ □□
  - □□□□□ □□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□ `MouseEvent` □□□□□□□□□

□ 1□□□□□

□□□□□□□□□□□□□

- `click` □□□□□□□□□□□□□□□□□□□□
- `dblclick` □□□□□□□□□□□□□□□□□
- `mousedown` □□□□□□□□□□
- `mouseup` □□□□□□□□□□□□□

`click`事件可以看成是两个事件组成的：用户在同一个位置先触发`mousedown`，再触发`mouseup`。因此，触发顺序是，`mousedown`首先触发，`mouseup`接着触发，`click`最后触发。

双击时，`dblclick`事件则会在`mousedown`、`mouseup`、`click`之后触发。

（2）移动事件

鼠标移动相关的有五个事件。

- `mousemove`：当鼠标在一个节点内部移动时触发。当鼠标持续移动时，该事件会连续触发。为了避免性能问题，建议对该事件的监听函数做一些限定，比如限定一段时间内只能运行一次。
- `mouseenter`：鼠标进入一个节点时触发，进入子节点不会触发这个事件（详见后文）。
- `mouseover`：鼠标进入一个节点时触发，进入子节点会再一次触发这个事件（详见后文）。
- `mouseout`：鼠标离开一个节点时触发，离开父节点也会触发这个事件（详见后文）。
- `mouseleave`：鼠标离开一个节点时触发，离开父节点不会触发这个事件（详见后文）。

`mouseover`事件和`mouseenter`事件，都是鼠标进入一个节点时触发。两者的区别是，`mouseenter`事件只触发一次，而只要鼠标在节点内部移动，`mouseover`事件会在子节点上触发多次。

```
/* HTML 代码如下
 <ul>
   <li>item 1</li>
   <li>item 2</li>
   <li>item 3</li>
 </ul>
*/

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

// 进入 ul 节点以后，mouseenter 事件只会触发一次
// 以后只要鼠标在节点内移动，都不会再触发这个事件
// event.target 是 ul 节点
ul.addEventListener('mouseenter', function (event) {
  event.target.style.color = 'purple';
  setTimeout(function () {
    event.target.style.color = '';
  }, 500);
}, false);

// 进入 ul 节点以后，只要在子节点上移动，mouseover 事件会触发多次
// event.target 是 li 节点
ul.addEventListener('mouseover', function (event) {
```

```
  event.target.style.color = 'orange';
  setTimeout(function () {
    event.target.style.color = '';
  }, 500);
}, false);
```

上面代码中，在父节点内部进入子节点，不会触发 `mouseenter` 事件，但是会触发 `mouseover` 事件。

`mouseout` 事件和 `mouseleave` 事件，都是鼠标离开一个节点时触发。两者的区别是，在父元素内部离开一个子元素时，`mouseleave` 事件不会触发，而 `mouseout` 事件会触发。

```
/* HTML 代码如下
 <ul>
   <li>item 1</li>
   <li>item 2</li>
   <li>item 3</li>
 </ul>
*/

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

// 先进入 ul 节点，然后在节点内部移动，不会触发 mouseleave 事件
// 只有离开 ul 节点时，触发一次 mouseleave
// event.target 是 ul 节点
ul.addEventListener('mouseleave', function (event) {
  event.target.style.color = 'purple';
  setTimeout(function () {
    event.target.style.color = '';
  }, 500);
}, false);

// 先进入 ul 节点，然后在节点内部移动，mouseout 事件会触发多次
// event.target 是 li 节点
ul.addEventListener('mouseout', function (event) {
  event.target.style.color = 'orange';
  setTimeout(function () {
    event.target.style.color = '';
  }, 500);
}, false);
```

上面代码中，在父节点内部离开子节点，不会触发 `mouseleave` 事件，但是会触发 `mouseout` 事件。

（3）其他事件

- `contextmenu`：按下鼠标右键时（上下文菜单出现前）触发，或者按下“上下文”菜单键时触发。
- `wheel`：滚动鼠标的滚轮时触发，该事件继承的是 `WheelEvent` 接口。

MouseEvent 接口

`MouseEvent` 接口代表了鼠标相关的事件，单击（click）、双击（dblclick）、松开鼠标键（mouseup）、按下鼠标键（mousedown）等动作，所产生的事件对象都是 `MouseEvent` 实例。此外，滚轮事件和拖拉事件也是 `MouseEvent` 实例。

`MouseEvent` 接口继承了 `Event` 接口，所以拥有 `Event` 的所有属性和方法，并且还提供鼠标独有的属性和方法。

浏览器原生提供一个 `MouseEvent()` 构造函数，用于新建一个 `MouseEvent` 实例。

```
var event = new MouseEvent(type, options);
```

`MouseEvent()` 构造函数接受两个参数。第一个参数是字符串，表示事件名称；第二个参数是一个事件配置对象，该参数可选。除了 `Event` 接口的实例配置属性，该对象可以配置以下属性，所有属性都是可选的。

- `screenX`：数值，鼠标相对于屏幕的水平位置（单位像素），默认值为 0，设置该属性不会移动鼠标。
- `screenY`：数值，鼠标相对于屏幕的垂直位置（单位像素），其他与 `screenX` 相同。
- `clientX`：数值，鼠标相对于程序窗口的水平位置（单位像素），默认值为 0，设置该属性不会移动鼠标。
- `clientY`：数值，鼠标相对于程序窗口的垂直位置（单位像素），其他与 `clientX` 相同。
- `ctrlKey`：布尔值，是否同时按下了 Ctrl 键，默认值为 `false`。
- `shiftKey`：布尔值，是否同时按下了 Shift 键，默认值为 `false`。
- `altKey`：布尔值，是否同时按下 Alt 键，默认值为 `false`。
- `metaKey`：布尔值，是否同时按下 Meta 键，默认值为 `false`。
- `button`：数值，表示按下了哪一个鼠标按键，默认值为 `0`，表示按下主键（通常是鼠标的左键）或者当前事件没有定义这个属性；`1` 表示按下辅助键（通常是鼠标的中间键），`2` 表示按下次要键（通常是鼠标的右键）。
- `buttons`：数值，表示按下了鼠标的哪些键，是一个三个比特位的二进制值，默认为 `0`（没有按下任何键）。`1`（二进制 `001`）表示按下主键（通常是左键），`2`（二进制 `010`）表示按下次要键（通常是右键），`4`（二进制 `100`）表示按下辅助键（通常是中间键）。因此，如果返回 `3`（二进制 `011`）就表示同时按下了左键和右键。
- `relatedTarget`：节点对象，表示事件的相关节点，默认为 `null`。`mouseenter` 和 `mouseover` 事件时，表示鼠标刚刚离开的那个元素节点；`mouseout` 和 `mouseleave` 事件时，表示鼠标正在进入的那个元素节点。

下面是一个例子。

```
function simulateClick() {
  var event = new MouseEvent('click', {
    'bubbles': true,
    'cancelable': true
  });
  var cb = document.getElementById('checkbox');
  cb.dispatchEvent(event);
}
```

上面代码生成一个鼠标点击事件，并触发该事件。

MouseEvent 接口的实例属性

MouseEvent.altKey，MouseEvent.ctrlKey，MouseEvent.metaKey，MouseEvent.shiftKey

`MouseEvent.altKey`、`MouseEvent.ctrlKey`、`MouseEvent.metaKey`、`MouseEvent.shiftKey`

这四个属性都返回一个布尔值，表示事件发生时，是否按下对应的键。它们都是只读属性。

- `altKey`属性：Alt 键
- `ctrlKey`属性：Ctrl 键
- `metaKey`属性：Meta 键（Mac 键盘是一个四瓣的小花，Windows 键盘是 Windows 键）
- `shiftKey`属性：Shift 键

```
// HTML 代码如下
// <body onclick="showKey(event)">
function showKey(e) {
  console.log('ALT key pressed: ' + e.altKey);
  console.log('CTRL key pressed: ' + e.ctrlKey);
  console.log('META key pressed: ' + e.metaKey);
  console.log('SHIFT key pressed: ' + e.shiftKey);
}
```

上面代码中，点击网页会输出是否同时按下对应的键。

MouseEvent.button MouseEvent.buttons

`MouseEvent.button`

- 0
- 1
- 2

`mousemove`

```
// HTML
// <button onmouseup="whichButton(event)">
var whichButton = function (e) {
  switch (e.button) {
    case 0:
      console.log('Left button clicked.');
      break;
    case 1:
      console.log('Middle button clicked.');
      break;
    case 2:
      console.log('Right button clicked.');
      break;
    default:
      console.log('Unexpected code: ' + e.button);
  }
}
```

`MouseEvent.buttons`

- 1 `001`
- 2 `010`
- 4 `100`

3

011

MouseEvent.clientX MouseEvent.clientY

`MouseEvent.clientX`
`MouseEvent.clientY`

```
// HTML 代码为
// <body onmousedown="showCoords(event)">
function showCoords(evt){
  console.log(
    'clientX value: ' + evt.clientX + '\n' +
    'clientY value: ' + evt.clientY + '\n'
  );
}
```

这两个属性还分别有一个别名 `MouseEvent.x` 和 `MouseEvent.y`。

MouseEvent.movementX, MouseEvent.movementY

`MouseEvent.movementX` 属性返回当前位置与上一个 `mousemove` 事件之间的水平距离（单位像素）。数值上，它等于下面的计算公式。

```
currentEvent.movementX = currentEvent.screenX - previousEvent.screenX
```

`MouseEvent.movementY` 属性返回当前位置与上一个 `mousemove` 事件之间的垂直距离（单位像素）。数值上，它等于下面的计算公式。

```
currentEvent.movementY = currentEvent.screenY - previousEvent.screenY。
```

这两个属性都是只读属性。

MouseEvent.screenX, MouseEvent.screenY

`MouseEvent.screenX` 属性返回鼠标位置相对于屏幕左上角的水平坐标（单位像素），`MouseEvent.screenY` 属性返回垂直坐标。这两个属性都是只读属性。

```
// HTML 代码如下
// <body onmousedown="showCoords(event)">
function showCoords(evt) {
  console.log(
    'screenX value: ' + evt.screenX + '\n',
```

```
    'screenY value: ' + evt.screenY + '\n'
  );
}
```

MouseEvent.offsetX、MouseEvent.offsetY

`MouseEvent.offsetX` 属性返回事件发生时，鼠标指针与触发事件的节点的 `padding` 边缘在水平方向的偏移量。
`MouseEvent.offsetY` 属性返回垂直方向的偏移量。如果事件节点有 `padding` 属性，则返回值包括 `padding` 的大小。

```
/* HTML 代码如下
  <style>
    p {
      width: 100px;
      height: 100px;
      padding: 100px;
    }
  </style>
  <p>Hello</p>
*/
var p = document.querySelector('p');
p.addEventListener(
  'click',
  function (e) {
    console.log(e.offsetX);
    console.log(e.offsetY);
  },
  false
);
```

上面代码中，`p` 节点的内容区域宽高为 `150 150`，`padding` 为 `100`。点击内容区域的左上角，返回 `padding` 的大小 `100`。点击内容区域的中间，返回 `50`。

MouseEvent.pageX、MouseEvent.pageY

`MouseEvent.pageX` 属性返回事件发生时，显示内容的文档对象（document）的水平坐标。`MouseEvent.pageY` 属性返回垂直坐标。
。

```
/* HTML 代码如下
  <style>
    body {
      height: 2000px;
    }
  </style>
*/
document.body.addEventListener(
  'click',
  function (e) {
    console.log(e.pageX);
    console.log(e.pageY);
  },
  false
);
```

上面代码中，页面高度为2000像素，会产生垂直滚动条。滚动到页面底部，点击鼠标输出的`pageY`值会接近2000。

MouseEvent.relatedTarget

`MouseEvent.relatedTarget`属性返回事件的相关节点。对于那些没有相关节点的事件，该属性返回`null`。该属性只读。

下表列出不同事件的`target`属性值和`relatedTarget`属性值义。

事件名称	**target** 属性	**relatedTarget** 属性
focusin	接受焦点的节点	丧失焦点的节点
focusout	丧失焦点的节点	接受焦点的节点
mouseenter	将要进入的节点	将要离开的节点
mouseleave	将要离开的节点	将要进入的节点
mouseout	将要离开的节点	将要进入的节点
mouseover	将要进入的节点	将要离开的节点
dragenter	将要进入的节点	将要离开的节点
dragexit	将要离开的节点	将要进入的节点

下面是一个例子。

```
/*
  HTML □□□□
  <div id="outer" style="height:50px;width:50px;border:1px solid black;">
    <div id="inner" style="height:25px;width:25px;border:1px solid black;"></div>
  </div>
*/

var inner = document.getElementById('inner');
inner.addEventListener('mouseover', function (event) {
  console.log('□□' + event.target.id + ' □□' + event.relatedTarget.id);
}, false);
inner.addEventListener('mouseenter', function (event) {
  console.log('□□' + event.target.id + ' □□' + event.relatedTarget.id);
});
inner.addEventListener('mouseout', function (event) {
  console.log('□□' + event.target.id + ' □□' + event.relatedTarget.id);
});
inner.addEventListener("mouseleave", function (event){
  console.log('□□' + event.target.id + ' □□' + event.relatedTarget.id);
});

// □□□ outer □□inner□□□
// □□inner □□outer
// □□inner □□outer

// □□□ inner□□ outer□□□
// □□inner □□outer
// □□inner □□outer
```

MouseEvent □□□□□□

MouseEvent.getModifierState()

`MouseEvent.getModifierState` □□

□□□ □□□□□

```
document.addEventListener('click', function (e) {
  console.log(e.getModifierState('CapsLock'));
}, false);
```

上面的代码可以了解用户是否按下了大写键。

WheelEvent 接口

概述

WheelEvent 接口继承了 MouseEvent

实例，代表鼠标滚轮事件的实例对象。目前，鼠标滚轮相关的事件只有一个 `wheel`

事件，用户滚动鼠标的滚轮，就生成这个事件的实例。

浏览器原生提供 `WheelEvent()` 构造函数，用来生成 `WheelEvent` 实例。

```
var wheelEvent = new WheelEvent(type, options);
```

`WheelEvent()`

构造函数可以接受两个参数，第一个是字符串，表示事件类型，对于滚轮事件来说，这个值目前只能是 `wheel`

。第二个参数是事件的配置对象。该对象的属性除了 `Event`、`UIEvent`

的配置属性以外，还可以接受以下几个属性，所有属性都是可选的。

- `deltaX`：数值，表示滚轮的水平滚动量，默认值是 0.0。
- `deltaY`：数值，表示滚轮的垂直滚动量，默认值是 0.0。
- `deltaZ`：数值，表示滚轮的 Z 轴滚动量，默认值是 0.0。
- `deltaMode`：数值，表示相关的滚动事件的单位，适用于上面三个属性。`0` 表示滚动单位为像素，`1` 表示单位为行，`2` 表示单位为页，默认为 `0`。

实例属性

`WheelEvent` 事件实例除了具有 `Event` 和 `MouseEvent`

的实例属性和实例方法，还有一些自己的实例属性，但是没有自己的实例方法。

下面的属性都是只读属性。

- `WheelEvent.deltaX`：数值，表示滚轮的水平滚动量。
- `WheelEvent.deltaY`：数值，表示滚轮的垂直滚动量。
- `WheelEvent.deltaZ`：数值，表示滚轮的 Z 轴滚动量。
- `WheelEvent.deltaMode`：数值，表示上面三个属性的单位，`0` 是像素，`1` 是行，`2` 是页。

□□□ #2
□ □□□ □□□ 2018-04-25 15:30:21 UTC
□ □□□ □□□ 2021-08-17 17:50:00 UTC