

EventTarget 接口

事件的本质是程序各个组成部分之间的一种通信方式，也是异步编程的一种实现。DOM 支持大量的事件，本章开始介绍 DOM 的事件编程。

概述

DOM 的事件操作（监听和触发），都定义在`EventTarget`接口。所有节点对象都部署了这个接口，其他一些需要事件通信的浏览器内置对象（比如，`XMLHttpRequest`、`AudioNode`、`AudioContext`）也部署了这个接口。

该接口主要提供三个实例方法。

- `addEventListener`：绑定事件的监听函数
- `removeEventListener`：移除事件的监听函数
- `dispatchEvent`：触发事件

EventTarget.addEventListener()

`EventTarget.addEventListener()`用于在当前节点或对象上，定义一个特定事件的监听函数。一旦这个事件发生，就会执行监听函数。该方法没有返回值。

```
```javascript
target.addEventListener(type, listener[, useCapture]);
```
```

该方法接受三个参数。

- `type`：事件名称，大小写敏感。
- `listener`：监听函数。事件发生时，会调用该监听函数。
- `useCapture`：布尔值，表示监听函数是否在捕获阶段（capture）触发（参见后文《事件的传播》部分），默认为`false`（监听函数只在冒泡阶段被触发）。该参数可选。

下面是一个例子。

```
```javascript
function hello() {
 console.log('Hello world');
}

var button = document.getElementById('btn');
button.addEventListener('click', hello, false);
```
```

上面代码中，`button`节点的`addEventListener`方法绑定`click`事件的监听函数`hello`，该函数只在冒泡阶段触发。

关于参数，有两个地方需要注意。

首先，第二个参数除了监听函数，还可以是一个具有`handleEvent`方法的对象。

```
```javascript
buttonElement.addEventListener('click', {
 handleEvent: function (event) {
 console.log('click');
 }
});
```
```

上面代码中，`addEventListener`方法的第二个参数，就是一个具有`handleEvent`方法的对象。

其次，第三个参数除了布尔值`useCapture`，还可以是一个属性配置对象。该对象有以下属性。

- > - `capture`：布尔值，表示该事件是否在捕获阶段触发监听函数。
- > - `once`：布尔值，表示监听函数是否只触发一次，然后就自动移除。
- > - `passive`：布尔值，表示监听函数不会调用事件的`preventDefault`方法。如果监听函数调用了，浏览器将忽略这个要求，并在监控台输出一行警告。

如果希望事件监听函数只执行一次，可以打开属性配置对象的`once`属性。

```
```javascript
element.addEventListener('click', function (event) {
 // 只执行一次的代码
}, {once: true});
```
```

`addEventListener`方法可以为针对当前对象的同一个事件，添加多个不同的监听函数。这些函数按照添加顺序触发，即先添加先触发。如果为同一个事件多次添加同一个监听函数，该函数只会执行一次，多余的添加将自动被去除（不必使用`removeEventListener`方法手动去除）。

```
```javascript
function hello() {
 console.log('Hello world');
}

document.addEventListener('click', hello, false);
document.addEventListener('click', hello, false);
```
```

执行上面代码，点击文档只会输出一行`Hello world`。

如果希望向监听函数传递参数，可以用匿名函数包装一下监听函数。

```
```javascript
```

```
function print(x) {
 console.log(x);
}

var el = document.getElementById('div1');
el.addEventListener('click', function () { print('Hello'); }, false);

```

上面代码通过匿名函数，向监听函数`print`传递了一个参数。

监听函数内部的`this`，指向当前事件所在的那个对象。

```
```javascript
// HTML 代码如下
// <p id="para">Hello</p>
var para = document.getElementById('para');
para.addEventListener('click', function (e) {
  console.log(this.nodeName); // "P"
}, false);
```

```

上面代码中，监听函数内部的`this`指向事件所在的对象`para`。

## EventTarget.removeEventListener()

`EventTarget.removeEventListener`方法用来移除`addEventListener`方法添加的事件监听函数。该方法没有返回值。

```
```javascript
div.addEventListener('click', listener, false);
div.removeEventListener('click', listener, false);
```

```

`removeEventListener`方法的参数，与`addEventListener`方法完全一致。它的第一个参数“事件类型”，大小写敏感。

注意，`removeEventListener`方法移除的监听函数，必须是`addEventListener`方法添加的那个监听函数，而且必须在同一个元素节点，否则无效。

```
```javascript
div.addEventListener('click', function (e) {}, false);
div.removeEventListener('click', function (e) {}, false);
```

```

上面代码中，`removeEventListener`方法无效，因为监听函数不是同一个匿名函数。

```
```javascript
element.addEventListener('mousedown', handleMouseDown, true);
element.removeEventListener("mousedown", handleMouseDown, false);
```

```

上面代码中，`removeEventListener`方法也是无效的，因为第三个参数不一样。

```
EventTarget.dispatchEvent()
```

`EventTarget.dispatchEvent`方法在当前节点上触发指定事件，从而触发监听函数的执行。该方法返回一个布尔值，只要有一个监听函数调用了`Event.preventDefault()`，则返回值为`false`，否则为`true`。

```
```javascript
target.dispatchEvent(event)
```
```

`dispatchEvent`方法的参数是一个`Event`对象的实例（详见《Event 对象》章节）。

```
```javascript
para.addEventListener('click', hello, false);
var event = new Event('click');
para.dispatchEvent(event);
```
```

上面代码在当前节点触发了`click`事件。

如果`dispatchEvent`方法的参数为空，或者不是一个有效的事件对象，将报错。

下面代码根据`dispatchEvent`方法的返回值，判断事件是否被取消了。

```
```javascript
var canceled = !cb.dispatchEvent(event);
if (canceled) {
  console.log('事件取消');
} else {
  console.log('事件未取消');
}
```
```