触摸事件

触摸操作概述

浏览器的触摸 API 由三个部分组成。

- Touch：一个触摸点
- TouchList：多个触摸点的集合
- TouchEvent：触摸引发的事件实例

`Touch`接口的实例对象用来表示触摸点（一根手指或者一根触摸笔），包括位置、大小、形状、压力、目标元素等属性。有时，触摸动作由多个触摸点（多根手指）组成，多个触摸点的集合由`TouchList`接口的实例对象表示。`TouchEvent`接口的实例对象代表由触摸引发的事件，只有触摸屏才会引发这一类事件。

很多时候，触摸事件和鼠标事件同时触发，即使这个时候并没有用到鼠标。这是为了让那些只定义鼠标事件、没有定义触摸事件的代码，在触摸屏的情况下仍然能用。如果想避免这种情况，可以用`event.preventDefault`方法阻止发出鼠标事件。

Touch 接口

Touch 接口概述

Touch 接口代表单个触摸点。触摸点可能是一根手指，也可能是一根触摸笔。

浏览器原生提供`Touch`构造函数，用来生成`Touch`实例。

```javascript
var touch = new Touch(touchOptions);
```

`Touch`构造函数接受一个配置对象作为参数，它有以下属性。

- `identifier`：必需，类型为整数，表示触摸点的唯一 ID。
- `target`：必需，类型为元素节点，表示触摸点开始时所在的网页元素。
- `clientX`：可选，类型为数值，表示触摸点相对于浏览器窗口左上角的水平距离，默认为0。
- `clientY`：可选，类型为数值，表示触摸点相对于浏览器窗口左上角的垂直距离，默认为0。
- `screenX`：可选，类型为数值，表示触摸点相对于屏幕左上角的水平距离，默认为0。
- `screenY`：可选，类型为数值，表示触摸点相对于屏幕左上角的垂直距离，默认为0。
- `pageX`：可选，类型为数值，表示触摸点相对于网页左上角的水平位置（即包括页面的滚动距离），默认为0。
- `pageY`：可选，类型为数值，表示触摸点相对于网页左上角的垂直位置（即包括页面的滚动距离），默认为0。

- `radiusX`：可选，类型为数值，表示触摸点周围受到影响的椭圆范围的 X 轴半径，默认为0。
- `radiusY`：可选：类型为数值，表示触摸点周围受到影响的椭圆范围的 Y 轴半径，默认为0。
- `rotationAngle`：可选，类型为数值，表示触摸区域的椭圆的旋转角度，单位为度数，在0到90度之间，默认值为0。
- `force`：可选，类型为数值，范围在`0`到`1`之间，表示触摸压力。`0`代表没有压力，`1`代表硬件所能识别的最大压力，默认为`0`。

Touch 接口的实例属性

（1）Touch.identifier

`Touch.identifier`属性返回一个整数，表示触摸点的唯一 ID。这个值在整个触摸过程保持不变，直到触摸事件结束。

```javascript
someElement.addEventListener('touchmove', function (e) {
  for (var i = 0; i < e.changedTouches.length; i++) {
    console.log(e.changedTouches[i].identifier);
  }
}, false);
```

（2）Touch.screenX，Touch.screenY，Touch.clientX，Touch.clientY，pageX，pageY

`Touch.screenX`属性和`Touch.screenY`属性，分别表示触摸点相对于屏幕左上角的横坐标和纵坐标，与页面是否滚动无关。

`Touch.clientX`属性和`Touch.clientY`属性，分别表示触摸点相对于浏览器视口左上角的横坐标和纵坐标，与页面是否滚动无关。

`Touch.pageX`属性和`Touch.pageY`属性，分别表示触摸点相对于当前页面左上角的横坐标和纵坐标，包含了页面滚动带来的位移。

（3）Touch.radiusX，Touch.radiusY，Touch.rotationAngle

`Touch.radiusX`属性和`Touch.radiusY`属性，分别返回触摸点周围受到影响的椭圆范围的 X 轴半径和 Y 轴半径，单位为像素。乘以 2 就可以得到触摸范围的宽度和高度。

`Touch.rotationAngle`属性表示触摸区域的椭圆的旋转角度，单位为度数，在`0`到`90`度之间。

上面这三个属性共同定义了用户与屏幕接触的区域，对于描述手指这一类非精确的触摸，很有帮助。指尖接触屏幕，触摸范围会形成一个椭圆，这三个属性就用来描述这个椭圆区域。

下面是一个示例。

```javascript
div.addEventListener('touchstart', rotate);
div.addEventListener('touchmove', rotate);
div.addEventListener('touchend', rotate);

function rotate(e) {
  var touch = e.changedTouches.item(0);
  e.preventDefault();

  src.style.width = touch.radiusX * 2 + 'px';
  src.style.height = touch.radiusY * 2 + 'px';
  src.style.transform = 'rotate(' + touch.rotationAngle + 'deg)';
};
```

（4）Touch.force

`Touch.force`属性返回一个`0`到`1`之间的数值，表示触摸压力。`0`代表没有压力，`1`代表硬件所能识别的最大压力。

（5）Touch.target

`Touch.target`属性返回一个元素节点，代表触摸发生时所在的那个元素节点。即使触摸点已经离开了这个节点，该属性依然不变。

TouchList 接口

`TouchList`接口表示一组触摸点的集合。它的实例是一个类似数组的对象，成员是`Touch`的实例对象，表示所有触摸点。用户用三根手指触摸，产生的`TouchList`实例就会包含三个成员，每根手指的触摸点对应一个`Touch`实例对象。

它的实例主要通过触摸事件的`TouchEvent.touches`、`TouchEvent.changedTouches`、`TouchEvent.targetTouches`这几个属性获取。

它的实例属性和实例方法只有两个。

- `TouchList.length`：数值，表示成员数量（即触摸点的数量）。
- `TouchList.item()`：返回指定位置的成员，它的参数是该成员的位置编号（从零开始）。

TouchEvent 接口

概述

TouchEvent 接口继承了 Event 接口，表示由触摸引发的事件实例，通常来自触摸屏或轨迹板。除了被继承的属性以外，它还有一些自己的属性。

浏览器原生提供`TouchEvent()`构造函数，用来生成触摸事件的实例。

```javascript
new TouchEvent(type, options)
```

`TouchEvent()`构造函数可以接受两个参数，第一个参数是字符串，表示事件类型；第二个参数是事件的配置对象，该参数是可选的，对象的所有属性也是可选的。除了`Event`接口的配置属性，该接口还有一些自己的配置属性。

- `touches`：`TouchList`实例，代表所有的当前处于活跃状态的触摸点，默认值是一个空数组`[]`。
- `targetTouches`：`TouchList`实例，代表所有处在触摸的目标元素节点内部、且仍然处于活动状态的触摸点，默认值是一个空数组`[]`。
- `changedTouches`：`TouchList`实例，代表本次触摸事件的相关触摸点，默认值是一个空数组`[]`。
- `ctrlKey`：布尔值，表示 Ctrl 键是否同时按下，默认值为`false`。
- `shiftKey`：布尔值，表示 Shift 键是否同时按下，默认值为`false`。
- `altKey`：布尔值，表示 Alt 键是否同时按下，默认值为`false`。
- `metaKey`：布尔值，表示 Meta 键（或 Windows 键）是否同时按下，默认值为`false`。

实例属性

TouchEvent 接口的实例具有`Event`实例的所有属性和方法，此外还有一些它自己的实例属性，这些属性全部都是只读。

（1）TouchEvent.altKey，TouchEvent.ctrlKey，TouchEvent.shiftKey，TouchEvent.metaKey

- `TouchEvent.altKey`：布尔值，表示触摸时是否按下了 Alt 键。
- `TouchEvent.ctrlKey`：布尔值，表示触摸时是否按下了 Ctrl 键。
- `TouchEvent.shiftKey`：布尔值：表示触摸时是否按下了 Shift 键。
- `TouchEvent.metaKey`：布尔值，表示触摸时是否按下了 Meta 键（或 Windows 键）。

下面是一个示例。

```javascript
someElement.addEventListener('touchstart', function (e) {
  console.log('altKey = ' + e.altKey);
  console.log('ctrlKey = ' + e.ctrlKey);
  console.log('metaKey = ' + e.metaKey);
  console.log('shiftKey = ' + e.shiftKey);
}, false);
```

（2）TouchEvent.changedTouches

`TouchEvent.changedTouches`属性返回一个`TouchList`实例，成员是一组`Touch`实例对象，表示本次触摸事件的相关触摸点。

对于不同的时间，该属性的含义有所不同。

- `touchstart`事件：被激活的触摸点
- `touchmove`事件：发生变化的触摸点
- `touchend`事件：消失的触摸点（即不再被触碰的点）

下面是一个示例。

```javascript
someElement.addEventListener('touchmove', function (e) {
  for (var i = 0; i < e.changedTouches.length; i++) {
    console.log(e.changedTouches[i].identifier);
  }
}, false);
```

（3）TouchEvent.touches

`TouchEvent.touches`属性返回一个`TouchList`实例，成员是所有仍然处于活动状态（即触摸中）的触摸点。一般来说，一个手指就是一个触摸点。

下面是一个示例。

```javascript
someElement.addEventListener('touchstart', function (e) {
  switch (e.touches.length) {
    // 一根手指触摸
    case 1: handle_one_touch(e); break;
    // 两根手指触摸
    case 2: handle_two_touches(e); break;
    // 三根手指触摸
    case 3: handle_three_touches(e); break;
    // 其他情况
    default: console.log('Not supported'); break;
  }
}, false);
```

（4）TouchEvent.targetTouches

`TouchEvent.targetTouches`属性返回一个`TouchList`实例，成员是触摸事件的目标元素节点内部、所有仍然处于活动状态（即触摸中）的触摸点。

```javascript
```

```
function touches_in_target(ev) {
  return (ev.touches.length === ev.targetTouches.length ? true : false);
}
```

上面代码用来判断，是否所有触摸点都在目标元素内。

触摸事件的种类

触摸引发的事件，有以下几种。可以通过`TouchEvent.type`属性，查看到底发生的是哪一种事件。

- `touchstart`：用户开始触摸时触发，它的`target`属性返回发生触摸的元素节点。
- `touchend`：用户不再接触触摸屏时（或者移出屏幕边缘时）触发，它的`target`属性与`touchstart`事件一致的，就是开始触摸时所在的元素节点。它的`changedTouches`属性返回一个`TouchList`实例，包含所有不再触摸的触摸点（即`Touch`实例对象）。
- `touchmove`：用户移动触摸点时触发，它的`target`属性与`touchstart`事件一致。如果触摸的半径、角度、力度发生变化，也会触发该事件。
- `touchcancel`：触摸点取消时触发，比如在触摸区域跳出一个模态窗口（modal window）、触摸点离开了文档区域（进入浏览器菜单栏）、用户的触摸点太多，超过了支持的上限（自动取消早先的触摸点）。

下面是一个例子。

```javascript
var el = document.getElementsByTagName('canvas')[0];
el.addEventListener('touchstart', handleStart, false);
el.addEventListener('touchmove', handleMove, false);

function handleStart(evt) {
  evt.preventDefault();
  var touches = evt.changedTouches;
  for (var i = 0; i < touches.length; i++) {
    console.log(touches[i].pageX, touches[i].pageY);
  }
}

function handleMove(evt) {
  evt.preventDefault();
  var touches = evt.changedTouches;
  for (var i = 0; i < touches.length; i++) {
    var touch = touches[i];
    console.log(touch.pageX, touch.pageY);
  }
}
```