

# 事件处理

讲师: 王红元  
微博: coderwhy



更多学习资源



老师答疑交流



# 事件的介绍

## ■ 什么时候会产生事件呢？

- 小程序需要经常和用户进行某种交互，比如点击界面上的某个按钮或者区域，比如滑动了某个区域；
- 这些交互都会产生各种各样的事件；

## ■ 事件时如何处理呢？

- - 事件是通过bind/catch这个属性绑定在组件上的（和普通的属性写法很相似, 以key= “value” 形式）；
- - key以bind或catch开发, 从1.5.0版本开始, 可以在bind和catch后加上一个冒号；
- - 同时当前页面的Page构造器中定义对应的事件处理函数tapName, 如果没有对应的函数, 触发事件时会报错
- - 当用户点击该button区域时，达到触发条件生成事件tap，该事件处理函数tapName会被执行，同时还会收到一个事件对象event。





# 事件的简单演练

## ■ 事件的简单演练:

The screenshot displays the HBuilderX IDE interface with a simulated WeChat environment on the left and the code editor on the right. The simulated device shows a status bar with 'WeChat', '10:45', and '100%' battery, and a page with a single button labeled '按钮'.

The code editor shows two files: `home.wxml` and `home.js`.

**home.wxml** code:

```
1 <!--home.wxml-->
2 <button size='mini'
3   bindtap="onBtnTap"
4   data-name="coderwhy">
5   按钮
6 </button>
```

**home.js** code:

```
1 // home.js
2 Page({
3   data: {
4     message: "微信"
5   },
6   onBtnTap(e) {
7     console.log('按钮被点击:', e)
8   }
9 })
```

Below the code editor, the 'Console' tab is active, showing two log entries:

```
按钮被点击: {type: "tap", timeStamp: 4246, target: {...}, currentTarget: {...}, detail: {...}, ...}
按钮被点击: {type: "tap", timeStamp: 5369, target: {...}, currentTarget: {...}, detail: {...}, ...}
```

Red arrows indicate the flow of data: one arrow points from the `onBtnTap` attribute in the WXML to the `onBtnTap` function in the JS, and another points from the `console.log` statement to the log entries in the console.



# 常见事件类型

■ 某些组件会有自己特性的事件类型，大家可以在使用组件时具体查看对应的文档

□ - 比如input有bindinput/bindblur/bindfocus等

□ - 比如scroll-view有bindscrolltoupper/bindscrolltolower等

■ 这里我们讨论几个组件都有的, 并且也比较常见的事件类型:

类型	触发条件
touchstart	手指触摸动作开始
touchmove	手指触摸后移动
touchcancel	手指触摸动作被打断，如来电提醒，弹窗
touchend	手指触摸动作结束
tap	手指触摸后马上离开
longpress	手指触摸后，超过350ms再离开，如果指定了事件回调函数并触发了这个事件，tap事件将不被触发
longtap	手指触摸后，超过350ms再离开（推荐使用longpress事件代替）



# 事件类型演练

## ■ 事件类型的演练

The screenshot shows a mobile app simulator on the left with a button labeled "按钮" and the text "常见事件类型". To the right is a code editor with two files: `home.wxml` and `home.js`.

**home.wxml** code:

```
7 </button>
8
9 <!-- 常见的事件类型 -->
10 <view bindtouchstart='onTouchStart'
11       bindtouchmove='onTouchMove'
12       bindtouchend='onTouchEnd'
13       bindtouchcancel='onTouchCancel'
14       bindtap='onTap'
15       bindlongpress='onLongTap'>
16   常见事件类型
17 </view>
18
```

**home.js** code:

```
onTouchStart(e) {
  console.log('触摸开始手势:', e)
},
onTouchMove(e) {
  console.log('触摸移动手势:', e)
},
onTouchEnd(e) {
  console.log('触摸结束手势:', e)
},
onTouchCancel(e) {
  console.log('触摸取消手势', e)
},
onTap(e) {
  console.log('轻点手势', e)
},
onLongTap(e) {
  console.log('长按手势', e)
}
```

Red arrows indicate the mapping from the `bind` attributes in `home.wxml` to the corresponding `on` functions in `home.js`:

- `bindtouchstart` to `onTouchStart`
- `bindtouchmove` to `onTouchMove`
- `bindtouchend` to `onTouchEnd`
- `bindtouchcancel` to `onTouchCancel`
- `bindtap` to `onTap`
- `bindlongpress` to `onLongTap`

## ■ 有两个注意点:

- ❑ **Touchcancel**: 在某些特定场景下才会触发 (比如来电打断等)
- ❑ **tap事件**和**longpress事件**通常只会触发其中一个





# 事件对象介绍

■ 当某个事件触发时, 会产生一个事件对象, 并且这个对象被传入到回调函数中, 事件对象有哪些常见的属性呢?

属性	类型	说明
type	String	事件类型
timeStamp	Integer	页面打开到触发事件所经过的毫秒数
target	Object	触发事件的组件的一些属性值集合
currentTarget	Object	当前组件的一些属性值集合
detail	Object	额外的信息
touches	Array	触摸事件, 当前停留在屏幕中的触摸点信息的数组
changedTouches	Array	触摸事件, 当前变化的触摸点信息的数组



# touches和changedTouches的区别

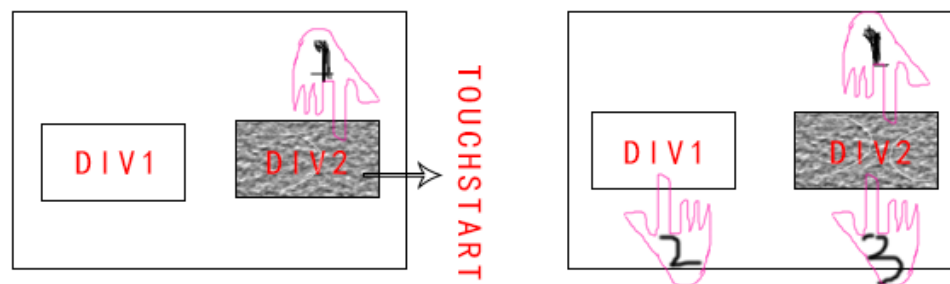
- 1.在touchend中不同
- 2.多手指触摸时不同

## touchstart

```
▼ Object {type: "touchstart", timeStamp: 2352}
  ▼ changedTouches: Array[1]
    ▼ 0: Object
      clientX: 181
      clientY: 40
      identifier: 0
      pageX: 181
      pageY: 40
      ▶ __proto__: Object
    length: 1
    ▶ __proto__: Array[0]
  ▶ currentTarget: Object
  ▶ target: Object
  timeStamp: 2352
  ▼ touches: Array[1]
    ▼ 0: Object
      clientX: 181
      clientY: 40
      identifier: 0
      pageX: 181
      pageY: 40
      ▶ __proto__: Object
    length: 1
    ▶ __proto__: Array[0]
  type: "touchstart"
  ▶ __proto__: Object
```

## touchend

```
▼ Object {type: "touchend", timeStamp: 5097}
  ▼ changedTouches: Array[1]
    ▼ 0: Object
      clientX: 187
      clientY: 37
      identifier: 0
      pageX: 187
      pageY: 37
      ▶ __proto__: Object
    length: 1
    ▶ __proto__: Array[0]
  ▶ currentTarget: Object
  ▶ target: Object
  timeStamp: 5097
  ▼ touches: Array[0]
    length: 0
    ▶ __proto__: Array[0]
    type: "touchend"
    ▶ __proto__: Object
```



touches: ①

changedTouches: ①

touches: ①②③

changedTouches: ②③

touches: 当前屏幕上所有触摸点的列表

changedTouches: 触发事件时改变的触摸点的集合



# currentTarget和target的区别

ne 6 ▾ 85% ▾ WiFi ▾ 模拟操作 ▾ 🔊 🖨️ 📱

我是外层container  
我是内层inner

- assets
- pages
  - home
  - target
    - target.js
    - target.json
    - target.wxml
    - target.wxss
- app.js
- app.json

```
1 <!--pages/target/target.wxml-->
2 <view class='container' id='container' bindtap='onContainerTap'>
3   我是外层container
4   <view class='inner' id='inner' bindtap='onInnerTap'>我是内层inner</view>
5 </view>
6
```

/pages/target/target.wxml 208 B 04\_event\* 行 6

Console Sources Network Security AppData Audits Sensor Storage Trace Wxml

top Filter Default levels 1 item

inner: {type: "tap", timeStamp: 1108, target: {...}, currentTarget: {...}, detail: {...}, ...}

- changedTouches: [...]
- currentTarget: {id: "inner", offsetLeft: 0, offsetTop: 25, dataset: {...}}
- detail: {x: 65.20000457763672, y: 80}
- target: {id: "inner", offsetLeft: 0, offsetTop: 25, dataset: {...}}
- timeStamp: 1108
- touches: [...]
- type: "tap"
- \_\_proto\_\_: Object

container: {type: "tap", timeStamp: 1108, target: {...}, currentTarget: {...}, detail: {...}, ...}

- changedTouches: [...]
- currentTarget: {id: "container", offsetLeft: 0, offsetTop: 0, dataset: {...}}
- detail: {x: 65.20000457763672, y: 80}
- target: {id: "inner", offsetLeft: 0, offsetTop: 25, dataset: {...}}
- timeStamp: 1108
- touches: [...]
- type: "tap"
- \_\_proto\_\_: Object





# 事件参数的传递

■ 当视图层发生事件时，某些情况需要事件携带一些参数到执行的函数中，这个时候就可以通过 **data-属性** 来完成：

□ 格式：data-属性的名称

□ 获取：e.currentTarget.dataset.属性的名称

事件参数

```
<!--pages/argument/argument.wxml-->
<view bind:tap="onTap1"
      data-name="coderwhy"
      data-age="18"
      data-height="1.88">
  事件参数
</view>
```

```
data: {
},
onTap1(e) {
  // 1.打印整个事件
  console.log(e)
  // 2.从事件中获取传递的参数
  const data = e.currentTarget.dataset;
  console.log(data)
},
/**
```

Console

```
{type: "tap", timeStamp: 2112, target: {...}, currentTarget: {...}, detail: {...}, ...}
{age: "18", height: "1.88", name: "coderwhy"}
```



# 参数传递的练习

Phone 6 80% WIFI 模拟操作

11:51 100%

事件参数

衣服 裤子 鞋子

```
/* pages/argument/argument.wxss */
.title {
  display: flex;
  background: purple;
}

.item {
  flex: 1;
  color: #fff;
  height: 88rpx;
  line-height: 88rpx;
  text-align: center;
}

.active {
  color: red;
}
```

```
<view class='title'>
  <block wx:for="{{titles}}" wx:key="*this">
    <view class='item' {{currentIndex === index ? "active": ""}}
      bind:tap="onItemTap"
      data-index="{{index}}">
      {{item}}
    </view>
  </block>
</view>
```

```
onItemTap(e) {
  // 1.获取当前的index
  const index = e.currentTarget.dataset.index;

  // 2.修改currentIndex
  this.setData({
    currentIndex: index
  })
}
```

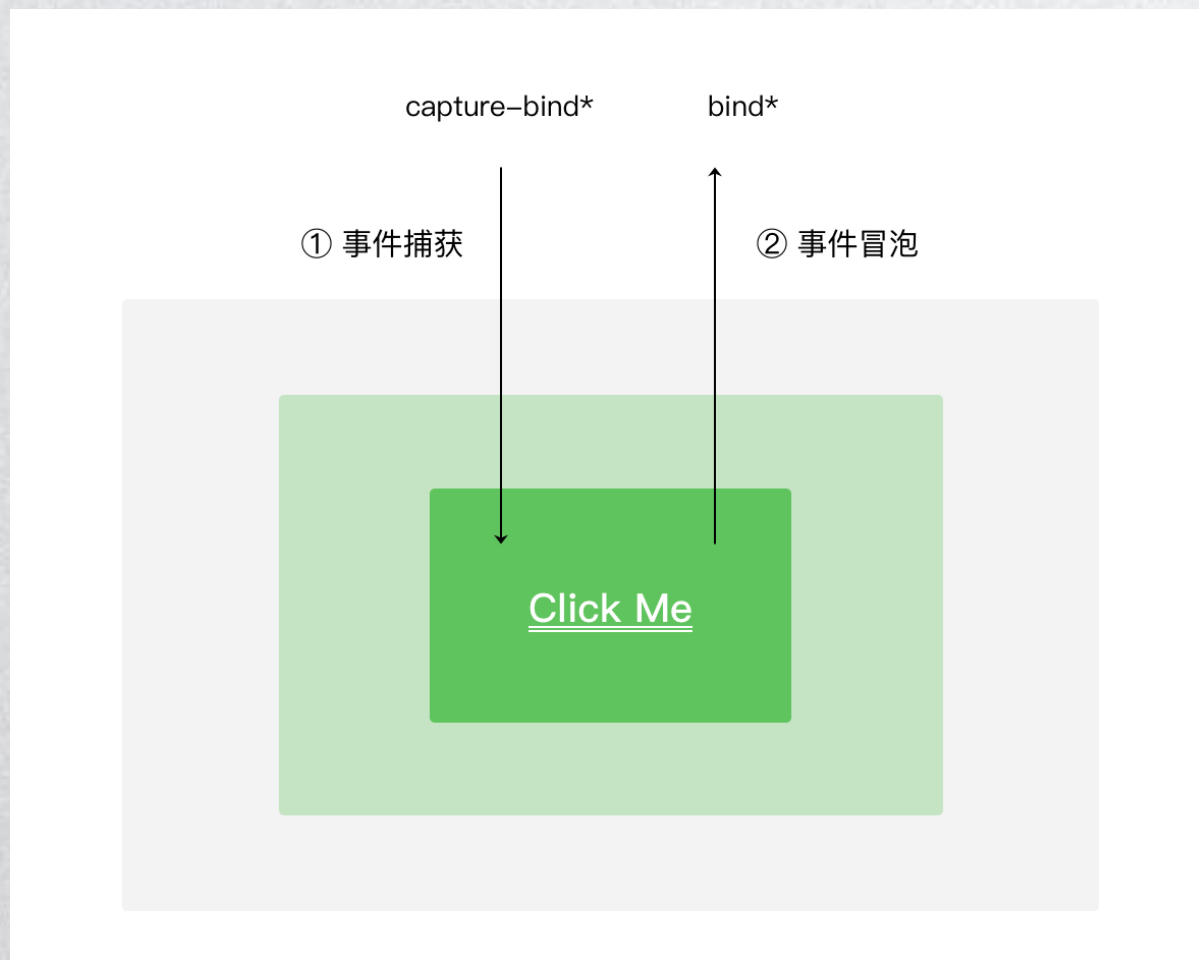
Console Sources Network Security AppData Audio Sensor Storage Trace Wxml

top Filter Default levels



# 事件冒泡和事件捕获

- 当界面产生一个事件时，事件分为了**捕获阶段**和**冒泡阶段**。







# 代码演练

100% WiFi 模拟操作

home.wxml x home.js home.wxss

```
1 <view>Hello World</view>
2
3 <view class='outer'
4   capture-bind:tap='onOuterCapture'
5   bindtap='onOuterBind'>
6   外层的view
7   <view class='middle'
8     capture-bind:tap='onMiddleCapture'
9     bindtap='onMiddleBind'>
10    中间的view
11    <view class='inner'
12      capture-bind:tap='onInnerCapture'
13      bindtap='onInnerBind'>
14      内层的view
15    </view>
16  </view>
17 </view>
```

home.wxml home.js x home.wxss

```
1 Page({
2   onOuterCapture() {
3     console.log('onOuterCapture')
4   },
5   onOuterBind() {
6     console.log('onOuterBind')
7   },
8   onMiddleCapture() {
9     console.log('onMiddleCapture')
10  },
11  onMiddleBind() {
12    console.log('onMiddleBind')
13  },
14  onInnerCapture() {
15    console.log('onInnerCapture')
16  },
17  onInnerBind() {
18    console.log('onInnerBind')
19  }
20 })
```

外层的view  
中间的view  
内层的view

Console Sources Network Security AppData Audits Sensor

top Filter Default levels 1 item

- onOuterCapture
- onMiddleCapture
- onInnerCapture
- onInnerBind
- onMiddleBind
- onOuterBind

先是从外向内捕获  
再是从内向外冒泡