3.5 事件

3.5.1 什么是事件

UI 界面的程序需要和用户互动，例如用户可能会点击你界面上某个按钮，又或者长按某个区域，这类反馈应该通知给开发者的逻辑层，需要将对应的处理状态呈现给用户。

有些时候程序上的“行为反馈”不一定是用户主动触发的，例如我们在视频 video 播放的过程中，播放进度是会一直变化的，这种反馈也应该通知给开发者做相应的逻辑处理。

在小程序里边，我们把这种“用户在渲染层的行为反馈”以及“组件的部分状态反馈”抽象为渲染层传递给逻辑层的“事件”，如图 3-7 所示。

图 3-7 渲染层产生用户交互事件传递给逻辑层

我们给出一个简单的处理事件的小程序代码。

代码清单 3-18 事件处理示例

```
<!-- page.wxml -->
<view id="tapTest" data-hi="WeChat" bindtap="tapName"> Click me! </view>

// page.js
   Page({
  tapName: function(event) {
    console.log(event)
  }
})
```

事件是通过 bindtap 这个属性绑定在组件上的，同时在当前页面的 Page 构造器中定义对应的事件处理函数 tapName，当用户点击该 view 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。

3.5.2 事件类型和事件对象

前边说到触发事件是由“用户在渲染层的行为反馈”以及“组件的部分状态反馈”引起的，由于不同组件的状态不一致，所以我们这里不讨论组件相关的事件（组件的事件可以参考其参数说明，详情见官方文档 https://mp.weixin.qq.com/debug/wxadoc/dev/component/ ）

常见的事件类型如表 3-10 所示。

表 3-10 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

类型	触发条件
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）
transitionend	会在 WXSS transition 或 wx.createAnimation 动画结束后触发
animationstart	会在一个 WXSS animation 动画开始时触发
animationiteration	会在一个 WXSS animation 一次迭代结束时触发

类型	触发条件
animationend	会在一个 WXSS animation 动画完成时触发

当事件回调触发的时候，会收到一个事件对象，对象的详细属性如下表所示。

表 3-11 事件对象属性

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

属性	类型	说明
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

这里需要注意的是 target 和 currentTarget 的区别，currentTarget 为当前事件所绑定的组件，而 target 则是触发该事件的源头组件。

代码清单 3-19 事件对象示例

```
<!-- page.wxml -->
<view id="outer" catchtap="handleTap">
  <view id="inner">点击我</view>
</view>
// page.js
Page({
  handleTap: function(evt) {
      // 当点击 inner 节点时
    // evt.target 是 inner view 组件
      // evt.currentTarget 是绑定了 handleTap 的 outer view 组件
      // evt.type == “tap”
      // evt.timeStamp == 1542
      // evt.detail == {x: 270, y: 63}
      // evt.touches == [{identifier: 0, pageX: 270, pageY: 63, clientX: 270, clientY: 63}]
      // evt.changedTouches == [{identifier: 0, pageX: 270, pageY: 63, clientX: 270, clientY: 63}]
  }
})
```

关于 target 和 currentTarget 对象的详细参数如表 3-12 所示。

表 3-12 target 和 currentTarget 事件对象属性

属性	类型	说明
id	String	当前组件的 id
tagName	String	当前组件的类型
dataset	Object	当前组件上由 data-开头的自定义属性组成的集合

关于 touch 和 changedTouches 对象的详细参数如表 3-13 所示。

表 3-13 touch 和 changedTouches 对象属性

属性	类型	说明
identifier	Number	触摸点的标识符
pageX, pageY	Number	距离文档左上角的距离，文档的左上角为原点 ，横向为 X 轴，纵向为 Y 轴

属性	类型	说明
clientX, clientY	Number	距离页面可显示区域（屏幕除去导航条）左上角距离，横向为 X 轴，纵向为 Y 轴

3.5.3 事件绑定与冒泡捕获

事件绑定的写法和组件属性一致，以 key="value"的形式，其中：

1. key 以 bind 或者 catch 开头，然后跟上事件的类型，如 bindtap、catchtouchstart。自基础库版本 1.5.0 起，bind 和 catch 后可以紧跟一个冒号，其含义不变，如 bind:tap、catch:touchstart。同时 bind 和 catch 前还可以加上 capture-来表示捕获阶段。
2. value 是一个字符串，需要在对应的页面 Page 构造器中定义同名的函数，否则触发事件时在控制台会有报错信息。

bind *和 capture-bind* 的含义分别代表事件的冒泡阶段和捕获阶段，其触发的顺序如图 3-8 所示。

图 3-8 事件捕获和冒泡触发时序

以下示例中，点击 inner view 会先后调用 handleTap2、handleTap4、handleTap3、handleTap1。

代码清单 3-20 事件的冒泡和捕获

```
<view
  id="outer"
  bind:touchstart="handleTap1"
  capture-bind:touchstart="handleTap2"
>
  outer view
  <view
    id="inner"
```

```
    bind:touchstart="handleTap3"
    capture-bind:touchstart="handleTap4"
  >
    inner view
  </view>
</view>
```

bind 事件绑定不会阻止冒泡事件向上冒泡，catch 事件绑定可以阻止冒泡事件向上冒泡。如果将上面代码中的第一个 capture-bind 改为 capture-catch，将只触发 handleTap2(capture-catch 将中断捕获阶段和取消冒泡阶段)

代码清单 3-21 使用 catch 阻止事件的传递

```
<view
  id="outer"
  bind:touchstart="handleTap1"
  capture-catch:touchstart="handleTap2"
>
  outer view
  <view
    id="inner"
    bind:touchstart="handleTap3"
    capture-bind:touchstart="handleTap4"
  >
    inner view
  </view>
</view>
```

注意，除表 3-10 列举的事件类型之外的其他组件自定义事件，如无特殊声明都是非冒泡事件，如<form/>的 submit 事件，<input/>的 input 事件，<scroll-view/>的 scroll 事件。