4.3 界面常见的交互反馈

用户和小程序上进行交互的时候，某些操作可能比较耗时，我们应该予以及时的反馈以舒缓用户等待的不良情绪。

4.3.1 触摸反馈

通常页面会摆放一些 button 按钮或者 view 区域，用户触摸按钮之后会触发下一步的操作。这种情况下，我们要对触摸这个行为给予用户一些响应。如图 4-17 所示，当我们手指触摸了 button 文字所在的 cell 区域时，对应的区域底色变成浅灰色，这样用户就可以知道小程序是有及时响应他的这次触摸操作，用户就不会很迷惑。

图 4-17 触摸区域底色变成灰色响应用户的触摸操作

小程序的 view 容器组件和 button 组件提供了 hover-class 属性，触摸时会往该组件加上对应的 class 改变组件的样式。

代码清单 4-1 通过 hover-class 属性改变触摸时的样式

```
/*page.wxss */

.hover{

  background-color: gray;

}



<!--page.wxml -->

<button hover-class="hover"> 点击 button </button>
```

```
<view hover-class="hover"> 点击 view</view>
```

对于用户的操作及时响应是非常优秀的体验，有时候在点击 button 按钮处理更耗时的操作时，我们也会使用 button 组件的 loading 属性，在按钮的文字前边出现一个 Loading，让用户明确的感觉到，这个操作会比较耗时，需要等待一小段时间。

图 4-18 button 文字前出现 loading

代码清单 4-2 设置 button 的 loading 属性

```
<!--page.wxml -->

<button loading="{{loading}}" bindtap="tap">操作</button>



//page.js

Page({

  data: { loading: false },

  tap: function() {

    // 把按钮的 loading 状态显示出来

    this.setData({

      loading: true

    })

    // 接着做耗时的操作

  }

})
```

4.3.2 Toast 和模态对话框

在完成某个操作成功之后，我们希望告诉用户这次操作成功并且不打断用户接下来的操作。弹出式提示 Toast 就是用在这样的场景上，Toast 提示默认 1.5 秒后自动消失，其表现形式如图 4-19 所示。

图 4-19 Toast 弹出式提示

小程序提供了显示隐藏 Toast 的接口，代码示例如下所示。

代码清单 4-3 显示/隐藏 Toast

```
Page({
  onLoad: function() {
    wx.showToast({ // 显示 Toast
      title: '已发送',
      icon: 'success',
      duration: 1500
    })
    // wx.hideToast() // 隐藏 Toast
  }
})
```

特别要注意，我们不应该把 Toast 用于错误提示，因为错误提示需要明确告知用户具体原因，因此不适合用这种一闪而过的 Toast 弹出式提示。一般需要用户明确知晓操作结果状态的话，会使用模态对话框来提示，同时附带下一步操作的指引。

图 4-20 模态对话框

代码清单 4-4 显示模态对话框

```
Page({
  onLoad: function() {
    wx.showModal({
      title: '标题',
      content: '告知当前状态，信息和解决方法',
      confirmText: '主操作',
      cancelText: '次要操作',
      success: function(res) {
        if (res.confirm) {
          console.log('用户点击主操作')
        } else if (res.cancel) {
          console.log('用户点击次要操作')
        }
      }
    })
  }
})
```

4.3.3 界面滚动

往往手机屏幕是承载不了所有信息的，所以内容区域肯定会超出屏幕区域，用户可以通过滑动屏幕来查看下一屏的内容，这是非常常见的界面滚动的交互。

为了让用户可以快速刷新当前界面的信息，一般在小程序里会通过下拉整个界面这个操作来触发，如图 4-21 所示。

图 4-21 下拉刷新

宿主环境提供了统一的下拉刷新交互，开发者只需要通过配置开启当前页面的下拉刷新，用户往下拉动界面触发下拉刷新操作时，Page 构造器的 onPullDownRefresh 回调会被触发，此时开发者重新拉取新数据进行渲染，实例代码如下所示。

代码清单 4-5 页面下拉刷新

```
//page.json
{"enablePullDownRefresh": true }


//page.js
Page({
  onPullDownRefresh: function() {
    // 用户触发了下拉刷新操作
    // 拉取新数据重新渲染界面
    // wx.stopPullDownRefresh() // 可以停止当前页面的下拉刷新。
  }
```

```
})
```

多数的购物小程序会在首页展示一个商品列表，用户滚动到底部的时候，会加载下一页的商品列表渲染到列表的下方，我们把这个交互操作叫为上拉触底。宿主环境提供了上拉的配置和操作触发的回调，如下代码所示。

代码清单 4-6 页面上拉触底

```
//page.json

// 界面的下方距离页面底部距离小于 onReachBottomDistance 像素时触发 onReachBottom 回调

{"onReachBottomDistance": 100 }



//page.js

Page({

  onReachBottom: function() {

    // 当界面的下方距离页面底部距离小于 100 像素时触发回调

  }

})
```

当然我们有些时候并不想整个页面进行滚动，而是页面中某一小块区域需要可滚动，此时就要用到宿主环境所提供的 scroll-view 可滚动视图组件。可以通过组件的 scroll-x 和 scroll-y 属性决定滚动区域是否可以横向或者纵向滚动，scroll-view 组件也提供了丰富的滚动回调触发事件，这部分我们就不再展开细节，读者可以通过 scroll-view 组件的官方文档了解到细节[1]。

最后一次编辑于 2019 年 08 月 19 日 （未经腾讯允许，不得转载）