第 3 章　小程序框架结构介绍

3.1　MINA 框架

微信团队为小程序提供的框架命名为 MINA 应用框架(在微信官方文档中现已取消该名称),它的核心是一个响应的数据绑定系统。

如图 3.1 所示,MINA 框架分为两大部分,分别是页面视图层与 AppService 应用逻辑层。在页面视图层,开发者使用 WXML 文件搭建页面的基本视图结构,使用 WXSS 文件控制页面的展现样式。小程序自己开发了一套 WXML 标签语言和 WXSS 样式语言,并非直接使用标准的 HTML5+CSS3。

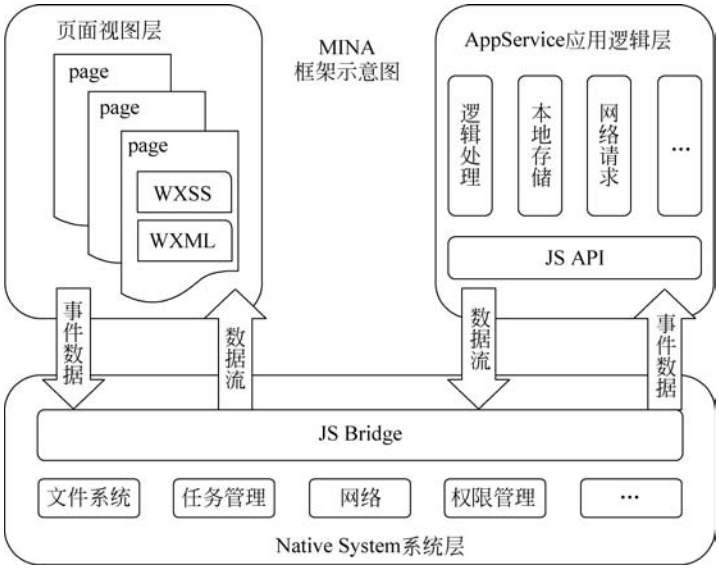

图 3.1　微信小程序框架结构图

AppService 应用逻辑层是 MINA 的服务中心,由微信客户端启用异步线程单独加载运行。页面渲染所需的数据、页面交互处理逻辑都在 AppService 中实现。AppService 使用 JavaScript 编写交互逻辑、网络请求、数据处理,小程序中的各个页面可以通过 AppService 实现数据管理、网络通信、应用生命周期管理和页面路由。

MINA 框架提供视图层与逻辑层之间的数据绑定和事件绑定系统,该系统实现了视图层和逻辑层的基于数据绑定和事件机制的数据交互:在逻辑层传递数据到视图层的过程

中，它让数据与视图非常简单地保持同步；当作数据修改的时候，只需要在逻辑层修改数据，视图层就会做相应的更新。

文件结构

MINA 框架程序包含一个描述整体程序的 App 和多个描述各个页面的 pages，一个 MINA 程序的主体部分由 3 个文件组成，这 3 个文件必须放在项目的根目录下，如图 3.2 所示。

图 3.2　MINA 程序结构

pages 用于创建页面，图 3.2 中的 index 页面和 logs 页面，用于展示欢迎页和小程序启动日志页面。小程序中的每个页面的“路径＋页面名”都要写在 app.json 的 pages 中，且 pages 中的第一个页面是小程序的首页即 index 页面。表 3.1 和表 3.2 分别给出了程序主体部分的 3 个文件以及构成一个页面的 4 个文件。

表 3.1　组成 MINA 程序主体的 3 个文件

文件名称	是否必需	作　用
app.js	是	小程序逻辑
app.json	是	小程序公共设置
app.wxss	否	小程序公共样式表

app.js 是小程序的脚本代码，用于监听并处理小程序的生命周期函数、声明全局变量等。app.json 用于配置小程序由哪些页面组成，配置小程序的窗口背景色，配置导航条样式，以及配置默认标题；需要注意的是该文件不可添加任何注释，示例代码为。

```
{
  "pages": [
      "pages/index/index",
      "pages/logs/logs"
   ],
   "window": {
      "backgroundTextStyle": "light",
      "navigationBarBackgroundColor": "#fff",
```

```
        "navigationBarTitleText": "WeChat",
        "navigationBarTextStyle": "black"
    },
    "sitemapLocation": "sitemap.json"
}
```

小程序中每新增或减少页面时，都需要对 app.json 文件中的 pages 数组进行修改。

app.wxss 是整个小程序的公共样式表，可以在页面组件的 class 属性上直接使用 app.wxss 中定义的样式规则，示例代码为：

```
/** app.wxss **/
.container {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-between;
  padding: 200rpx 0;
  box-sizing: border-box;
}
```

表 3.2 给出了微信规定的小程序的界面组织模式中的 4 个文件构成，其中，.wxml 文件（页面结构文件）类似 HTML，负责页面结构，可绑定数据；.wxss 文件（样式表文件）类似 CSS；.json 文件（配置文件）用于配置样式，如选项卡、窗口样式等；.js 文件（脚本文件）用于运行页面逻辑，使用 JS 语言。

表 3.2　构成一个 MINA 页面的 4 个文件

文件类型	是否必需	作　用
wxml	是	页面结构
wxss	否	页面样式表
json	否	页面配置
js	是	页面逻辑

3.2　WXML 标签语言

3.2.1　WXML 简介

WXML 用于指定界面的框架结构，而 WXSS 则用于指定界面的框架及元素的显示样式。

WXML 是微信小程序团队设计的一套标签语言，可以构建出页面的结构，开发者通过借助 WXML 提供的各种组件，可以很方便地实现文字的嵌入、图片的嵌入、视频的嵌入等。

我们知道，在开发基于 Web 的网络系统时，网页编程通常采用 HTML＋CSS＋JS 组合，HTML 用来描述当前这个页面的结构，CSS 用来描述页面的显示样式，JS 用来处理这

个页面和用户的交互。

在小程序中，WXML 充当的就是类似 HTML 的角色。打开 pages/index/index.wxml，其中代码如下。

```
<!-- index.wxml -->
<view class="container">
  <view class="userinfo">
    <button wx:if="{{!hasUserInfo && canIUse}}" open-type="getUserInfo"
            bindgetuserinfo="getUserInfo">获取头像昵称</button>
    <block wx:else>
      <image bindtap="bindViewTap" class="userinfo-avatar"
            src="{{userInfo.avatarUrl}}" mode="cover"></image>
      <text class="userinfo-nickname">{{userInfo.nickName}}</text>
    </block>
  </view>
  <view class="usermotto">
    <text class="user-motto">{{motto}}</text>
  </view>
</view>
```

从视图层的角度来说，WXML 由标签、属性等构成，但也存在如下与 HTML 的不同之处。

1. 开发工具不同

常见的 HTML5 的开发工具有 Adobe Dreamweaver CS6、Adobe Edge、DevExtreme、JetBrains WebStorm 等；小程序开发有自己的开发工具，名为“微信开发者工具”，可以实现同步本地文件＋开发调试＋编译＋预览＋上传＋发布等一整套流程。

2. 开发语言不同

小程序自己开发了一套 WXML 标签语言和 WXSS 样式语言，并非直接使用标准的 HTML5＋CSS3。

常见的 HTML 有，定义声音内容的<audio>标签，定义粗体字的<b>标签，定义表格标题的<caption>标签，定义文档中的节的 标签等；HTML5 中又新增了一些标签，如装载非正文内容附属信息的<aside>标签，对标题元素（h1～h6）进行组合的<hgroup>标签，用于描述独立的流内容（图像、图表、照片、代码等）的<figure>（<figcaption>）标签等。

小程序的 WXML 用的标签在语法上更接近 XML，遵循 SGML 规范，区别于 HTML 随意的标签闭合方式，WXML 必须包括开始标签和结束标签，如<view></view>、<text></text>、<navigator url="#" redirect></navigator>等，这些标签都是小程序给开发者包装好的基本能力，还提供了地图、视频、音频等组件能力。

3. 组件封装不同

小程序独立出来了很多原生 App 的组件，在 HTML5 中需要模拟才能实现的功能，在小程序中可以用直接调用组件的形式来实现。

4. 多了一些 wx：if 这样的属性以及{{}}这样的表达式

在一般的网页开发流程中，通常会通过 JS 操作 DOM，以引起界面的一些变化来响应用户的行为。例如，用户单击某个按钮的时候，JS 会记录一些状态到 JS 变量里，同时通过

DOM API 操控 DOM 的属性或者行为，从而引起视图层的一些变化。当项目越来越大的时候，代码中将会充斥着非常多的界面交互逻辑和程序的各种状态变量，显然这不是一个很好的开发模式，因此就有了 MVVM 的开发模式（例如 React、Vue），提倡把渲染和逻辑分离。简单来说，就是不要再让 JS 直接操控 DOM，JS 只需要管理状态即可，然后再通过一种模板语法来描述状态和界面结构的关系即可。

小程序的框架也是用到了这个思路，如果需要把一个 Hello World 的字符串显示在界面视图上，WXML 的写法是：

```
<text>{{msg}}</text>
```

JS 只需要管理状态即可：

```
this.setData({ msg: "Hello World" })
```

通过{{}}的语法把一个变量绑定到界面视图上，称为数据绑定。仅通过数据绑定还不能够完整地描述状态和界面的关系，还需要 if/else、for 等控制能力，在小程序里，这些控制能力都用以“wx:”开头的属性来表达，例如：

```
<view wx:if="{{boolean == true}}">
    <view class="bg_black"></view>
</view>
<view wx:elif="{{boolean == false}}">
    <view class="bg_red"></view>
</view>
```

3.2.2 基础知识

1. 语法规则

（1）所有的元素都需要闭合标签，例如：

```
<text>Hello World</text>;
```

（2）所有的元素都必须正确嵌套，例如：

```
<view>
    <text>Hello World</text>
</view>;
```

（3）属性值必须用引号包围，例如：

```
<text id = "myText">myText</text>;
```

（4）标签必须用小写。

（5）WXML 中连续多个空格会合并成一个空格。

2. WXML 的共同属性

由于某些属性被几乎所有的组件使用，这些属性被抽离出来，形成 WXML 组件的共同属性，如表 3.3 所示。

表 3.3　WXML 组件的共同属性

属　性　名	描　　述	注　　解
id	组件唯一标识	页面内唯一
class	组件样式类	在对应的 WXSS 内定义
style	组件的内联样式	常用语动态样式
hidden	组件是否显示	默认显示
data-*	自定义数据	触发时会进行上报
hide/catch	组件事件	

3. 基本组件

组件是视图层的基本组成单元，自带一些功能与微信风格类似的样式。一个组件通常包括开始标签和结束标签，属性用来修饰这个组件，内容在两个标签之内，形如：

```
<tagname property="value"> Contents here … </tagname>
```

需要注意的是，所有组件与属性都是小写，以连字符-连接。

界面结构 WXML 主要由七大类基础组件构成，分别如表 3.4～表 3.10 所示。

表 3.4　视图容器(View Container)类组件

序　　号	组　件　名	说　　明
1	movable-view	可移动的视图容器，在页面中可以拖曳滑动
2	cover-image	覆盖在原生组件之上的图片视图
3	cover-view	覆盖在原生组件之上的文本视图
4	movable-area	movable-view 的可移动区域
5	scroll-view	可滚动视图区域
6	swiper	滑块视图容器，其中只可放置 swiper-item 组件
7	swiper-item	仅可放置在 swiper 组件中，宽高自动设置为 100%
8	view	视图容器

表 3.5　基础内容(Basic Content)类组件

序　　号	组　件　名	说　　明
1	icon	图标，长度单位默认为 px
2	text	文本
3	progress	进度条
4	rich-text	富文本

表 3.6　表单(Form)类组件

序　　号	组　件　名	说　　明
1	button	按钮
2	form	表单
3	input	输入框
4	checkbox	多选项目
5	checkbox-group	多项选择器，内部由多个 checkbox 组成

续表

序　号	组　件　名	说　明
6	editor	富文本编辑器，可以对图片、文字进行编辑
7	radio	单选项目
8	radio-group	单项选择器，内部由多个 radio 组成
9	picker	从底部弹起的滚动选择器
10	picker-view	嵌入页面的滚动选择器
11	picker-view-column	滚动选择器子项
12	slider	滑动选择器
13	switch	开关选择器
14	label	标签，用来改进表单组件的可用性
15	textarea	多行输入框

表 3.7　导航(Navigator)类组件

序　号	组　件　名	说　明
1	functional-page-navigator	仅在插件中有效，用于跳转到插件功能页
2	navigator	页面链接

表 3.8　媒体(Media)类组件

序　号	组　件　名	说　明
1	audio	音频
2	camera	系统相机
3	image	图片
4	live-player	实时音视频播放
5	live-pusher	实时音视频录制
6	video	视频

表 3.9　地图(Map)类组件

序　号	组　件　名	说　明
1	map	地图

表 3.10　画布(Canvas)类组件

序　号	组　件　名	说　明
1	canvas	画布

其他更多组件及其使用方法将在第 4 章中予以介绍。

3.2.3　WXML 主要功能

1. 实现数据绑定

WXML 中的动态数据均来自于对应 Page 的 data，数据绑定使用{{}}将变量包含起来，可以作用于以下几方面。

1）内容

如在某一个页面的 JS 文件中定义一个字符串型变量 message，给其赋值为“Hello World”，代码为：

```
Page({
  data: {
    message: 'Hello World'
  }
})
```

然后在视图层的< view >组件中显示该 message 变量，代码为：

```
< view > {{ message }} </view >
```

2）组件属性

如视图层中某一个< view >组件的 id 属性命名为：

```
< view id = "item - {{id}}"> </view >
```

其中的 id 值在 Page 中定义：

```
Page({
  data: {
    id: 0
  }
})
```

3）控制属性

如在 Page 页面中定义一个 boolean 型变量 condition，初始化为 true：

```
Page({
  data: {
     condition:true
  },
})
```

然后再在 WXML 文件中添加如下代码：

```
< view wx:if = "{{condition}}">"Hello World"</view >
```

编译后会在模拟器中显示 Hello World 字样；如将 condition 改为 false，则运行后不会显示。

4）运算

可以在{{}}内进行简单的运算，如条件表达式中的三元运算：

```
< view hidden = "{{flag ? true : false}}"> Hidden </view >
```

以及算术运算：

```
< view > {{a + b}} + {{c}} + d </view >

Page({
```

```
  data: {
    a: 2,
    b: 3,
    c: 4
  }
})
```

view 中的显示内容为：5＋4＋d。

再比如字符串运算：

```
<view>{{"hello" + name}}</view>

Page({
  data:{
    name: 'World'
  }
})
```

等。

2. 条件渲染

在 WXML 框架中，使用 wx:if=""来判断是否需要渲染该代码块，如前面代码中的

```
<view wx:if = "{{condition}}">"Hello World"</view>
```

也可以用 wx:elif 和 wx:else 来添加一个 else 块，如：

```
<view wx:if = "{{length > 5}}"> 1 </view>
<view wx:elif = "{{length > 2}}"> 2 </view>
<view wx:else > 3 </view>
```

3. 列表渲染

在 WXML 文件中输入以下代码：

```
<view wx:for = "{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for - item = "i">
    <view style = 'display:inline - block;width:35px' wx:for = "{{[1, 2, 3, 4, 5, 6, 7, 8,
                                                                  9]}}" wx:for - item = "j">
      <view wx:if = "{{j <= i}}">
        {{i}} * {{j}} = {{i * j}}
      </view>
   </view>
</view>
```

此处通过 wx:for 两重嵌套循环，实现一个九九乘法表。<view>组件的 wx:for 控制属性用于绑定一个数组，可利用数组中各项数据重复渲染该组件，wx:for-item 用于指定数组当前元素的变量名。

以上代码编译运行后，在模拟器中的显示效果如图 3.3 所示。

之所以会出现显示字符的重叠，是因为样式设置有问题，需要在相应页面的 WXSS 文件中补充相应代码，如下。

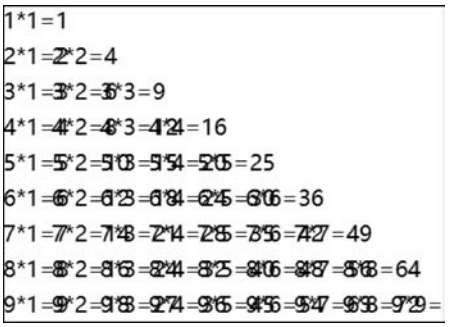

图 3.3　九九乘法表的实现效果(一)

```
.multiply{
  font-size:8px
}
```

然后在 WXML 文件中添加一个视图组件< view >,将其 class 属性设置为:

```
<view class="multiply">
```

再将上述代码置于该< view >组件内,则最终的实现效果如图 3.4 所示。

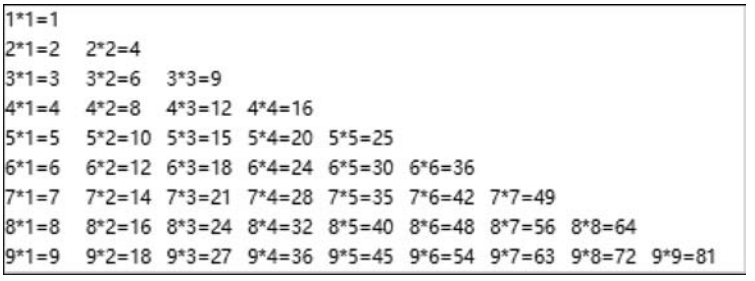

图 3.4　九九乘法表的实现效果(二)

3.3　WXSS

WXML 理解起来较为容易,但如果只是通过 WXML 搭建好了小程序页面的框架,并不能达到开发者想要的界面效果,正如同图 3.3 中的显示字符重叠一样,此时就需要用到 WXSS。WXSS(WeiXin Style Sheets)是一套样式语言,用于描述 WXML 的组件样式,用来决定 WXML 的组件该如何显示。微信提供的 WXSS 具有 CSS 大部分的特性,并做了一些扩充和修改。

1. 新增了尺寸单位

在用 CSS 布局样式时,开发者需要考虑到手机终端的屏幕显示会有不同的宽度和设备像素比,从而采用一些技巧来换算一些像素单位。WXSS 在底层支持新的尺寸单位 rpx,开发者可以免去换算的烦恼,只要交给小程序底层来换算即可,由于换算采用浮点数运算,所以运算结果会和预期结果有一点点儿偏差。

rpx(responsive pixel):可以根据屏幕宽度进行自适应。规定屏幕宽为 750rpx。如在 iPhone6

上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx＝375px＝750 物理像素，1rpx＝0.5px＝1 物理像素。微信官方文档建议在开发小程序时以 iPhone6 作为视觉设计标准。

rpx 与 px 单位换算如表 3.11 所示。

表 3.11　不同手机终端上的尺寸单位换算

序　　号	设　备　名	rpx 换算 px(屏幕宽度/750)	px 换算 rpx(750/屏幕宽度)
1	iPhone5(320×568)	1rpx＝0.42px	1px＝2.34rpx
2	iPhone6(375×667)	1rpx＝0.5px	1px＝2rpx
3	iPhone6 Plus(414×736)	1rpx＝0.552px	1px＝1.81rpx

2. 提供了全局的样式和局部样式

和前面 app.json、page.json 的作用类似，可以写一个 app.wxss 作为全局样式，它将会作用于当前小程序的所有页面，而局部页面样式 page.wxss 仅对当前页面生效。

3. 样式导入

可以使用@import 语句来导入外联样式表，其后面跟需要导入外联样式表的相对路径，并以分号结束。

例如，定义 other 页面的 WXSS 文件，在其中建立一个样式 appText，然后在 app.wxss 中直接导入该样式，代码如下。

```
/** other.wxss **/
.appText{
  margin:10px;
}
/** app.wxss **/
@import "other.wxss";
.content_text:{
  margin:15px;
}
```

如前所述，此处的 app.wxss 是全局样式，它将会作用于每一个页面；某一个 page 下的 WXSS 文件只作用于当前页面，并会覆盖全局样式中的相同属性。

微信小程序 WXSS 样式的使用大部分都和 CSS 样式一致，如表 3.12～表 3.15 所示。

表 3.12　WXSS 文本(text)的主要属性

序号	属　　性	说　　明	语法(属性值)
1	color	设置文本颜色	
2	direction	设置文本方向	ltr：从左到右。rtl：从右到左
3	letter-spacing	设置字符间距	
4	line-height	设置行高	
5	text-align	对齐元素中的文本	对齐方式有 left、right、center、justify 等
6	text-indent	缩进元素中文本的首行	
7	text-shadow	设置文本阴影	h-shadow、v-shadow、blur、color
8	vertical-align	设置元素的垂直对齐	
9	word-spacing	设置字间距	

表 3.13　WXSS 字体(font)的主要属性

序号	属　性	说　明	语法(属性值)
1	font-style	指定文本的字体样式	normal、italic、oblique 等
2	font-weight	指定字体的粗细	normal、bold、bolder、lighter 等
3	font-size	设置字体大小	smaller、larger、length(一个固定的值)等

表 3.14　WXSS 背景(background)的主要属性

序号	属　性	说　明	语法(属性值)
1	background-color	指定要使用的背景颜色	
2	background-position	指定背景图像的位置	center 等
3	background-size	指定背景图片的大小	如 background-size:80px 60px;
4	background-image	指定要使用的一个或多个背景图像	url('URL')、none 等

表 3.15　WXSS 显示(display)的主要属性

序号	属　性	说　明
1	flex	多栏多列布局
2	flex-direction	主轴方向(及项目的排列方向),包含水平方向 row、垂直方向 column 等
3	inline-block	行内块元素
4	inline-table	作为内联表格来显示(类似< table >),表格前后没有换行符
5	list-item	此元素会作为列表显示
6	table	会作为块级表格来显示(类似< table >),表格前后带有换行符
7	table-caption	作为一个表格标题显示(类似< caption >)
8	table-cell	作为一个表格单元格显示(类似< td >和< th >)
9	table-column	作为一个单元格列显示(类似< col >)
10	table-row	作为一个表格行显示(类似< tr >)
11	padding	内边距
12	margin	外边距

3.4　视图层和逻辑层的信息传递交互实现

一个小程序服务只有 WXML+WXSS 界面展示是不够的,还需要和用户交互,来响应用户的单击、获取用户的位置等。这里,通过编写 JS 脚本文件来举例说明视图层和逻辑层的信息传递交互逻辑的实现,如图 3.5 所示。

图 3.5　新建 demo1 页面

右击 pages,新建一个目录,命名为“demo1”,如图 3.6 所示。

依次新建 3 个文件,分别命名为 demo1.wxml、demo1.wxss 和 demo1.js,如图 3.7 所示。

图 3.6　新建 demo1 页面的相关文件

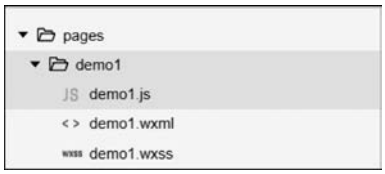

图 3.7　新建成功的 demo1 相关文件

双击 demo1.wxml 文件，输入以下代码。

```
<view>{{msg}}</view>
<button bindtap="clickMe">单击我</button>
```

现在要实现的交互逻辑是，当用户单击“单击我”按钮的时候，将界面上 msg 变量的信息显示成“Hello World”。首先需要在 button 上声明一个属性 bindtap，在 JS 文件里声明 clickMe 方法来响应这次单击操作。

双击 demo1.js 文件，在其中输入以下代码。

```
Page({
  data: {
    msg:"WeChat"
  },
  clickMe: function () {
    this.setData({msg: "Hello World" })
  }
})
```

再打开 app.json 文件，在 pages 中增加：

```
"pages/demo1/demo1"
```

然后，自定义编译条件如图 3.8 所示。

让小程序首先从 demo1 启动，编译运行，界面如图 3.9 所示。

单击“单击我”按钮后，“WeChat”文字就变成了“Hello World”，如图 3.10 所示。

在这个例子中，WXML 文件用于定义要显示的视图内容，变量 msg 在 JS 文件中定义并初始化为“WeChat”，通过定义 button 的单击事件函数 clickMe，将 msg 的内容变更为“Hello World”，并在 WXML 文件中显示。

图 3.8　自定义编译条件

图 3.9　初始运行界面

图 3.10　按钮单击后的界面

除了自定义触发函数外，还可以在 JS 中调用小程序提供的丰富的 API，利用这些 API 可以很方便地调用微信提供的能力，例如，获取用户信息、本地存储、微信支付等。第 2 章的例子程序图 2.10 中，在 pages/index/index.js 中就调用了 wx.getUserInfo 获取微信用户的头像和昵称，最后通过 setData 把获取到的信息显示到界面上，代码为：

```
wx.getUserInfo({
   success: res => {
      app.globalData.userInfo = res.userInfo
      this.setData({
        userInfo: res.userInfo,
        hasUserInfo: true
      })
   }
})
```

3.5　配置文件解析

当小程序开发工具运行后，出现如图 3.11 所示界面。其中，pages 规定了路由页面指定的页面模块，每个 pages 模块下面都包含 JS、WXML、WXSS、JSON 文件。图 3.11 中有两个页面，分别为 index 和 logs，就对应两组配置文件。utils 规定公用的方法，其中，小程序加载的时候必须以 app.js 作为入口开始运行，全局方法和变量可以放在里面。

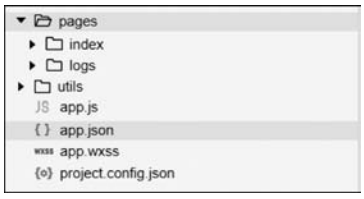

图 3.11　配置文件

3.5.1 app.json

app.json 文件必须要有，如果没有这个文件，IDE 会报错，因为微信框架把这个文件作为配置文件入口。当 IDE 报错的时候，只需新建这个文件，里面写个大括号，暂时不添加其他任何代码即可通过编译。在后续开发中，再根据项目需要继续完善该文件的配置。

如图 3.11 所示，app.json 为当前小程序的全局配置，包括小程序的所有页面路径、界面表现、网络超时时间、底部选项卡等。

打开 app.json 文件，其中代码如下。

```
{
  "pages": [
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window": {
    "backgroundTextStyle": "light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "WeChat",
    "navigationBarTextStyle": "black"
  },
  "sitemapLocation": "sitemap.json"
}
```

这里的 pages 字段，用于描述当前小程序所有页面路径，用于让微信客户端知道当前小程序页面定义在哪个目录。window 字段，定义小程序所有页面的顶部背景颜色、文字颜色等。navigationBarBackgroundColor 用于指定导航栏背景色，如改为“#0f0”，则效果如图 3.12 所示。

图 3.12　导航栏背景色

3.5.2 project.config.json

project.config.json 文件为每个开发者在本地生成的一些配置。

3.5.3 app.wxss

app.wxss 文件用于配置一些公用的样式设置，例如：

```
.container {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-between;
  padding: 200rpx 0;
  box-sizing: border-box;
}
```

3.5.4 app.js

app.js 文件也必须要有，没有也会报错。但是这个文件创建一下就行，什么都不需要写，以后可以在这个文件中监听并处理小程序的生命周期函数、声明全局变量。

3.5.5 app.wxml

app.wxml 文件不是必须要有，它只是规定了全局页面视图文件。

3.6 小程序的启动

微信客户端在打开小程序之前，将其代码包下载到本地手机终端，然后通过 app.json 的 pages 字段获取当前小程序的所有页面路径，例如：

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ]
}
```

上述代码配置说明在当前小程序项目中定义了两个页面，分别位于 pages/index/index 和 pages/logs/logs。其中，pages 字段的第一个页面就是这个小程序的首页，即打开小程序看到的第一个页面。根据该页面设置信息，微信客户端将首页代码装载进来，进而渲染出这个首页。

小程序启动之后，在 app.js 中定义的 App 实例的 onLaunch 回调会被执行：

```
App({
  onLaunch: function () {
  ...
  }
})
```

整个小程序只有一个 App 实例，是全部页面共享的。

另外一个页面是 pages/logs/logs，它包含 4 个文件，如图 3.13 所示，分别为 logs.js、logs.json、logs.wxml、logs.wxss。

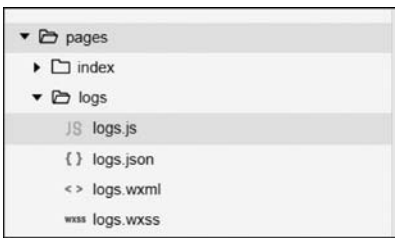

图 3.13 logs 页面包含的 4 个文件

微信客户端会根据 logs.json 配置生成一个界面，顶部的颜色和文字都可以在这个 JSON 文件中定义好，其代码为：

```
{
  "navigationBarTitleText": "查看启动日志",
  "usingComponents": {}
}
```

紧接着，微信客户端就会装载这个页面的 WXML 结构和 WXSS 样式，其中，logs.wxml 的代码为：

```
<!-- logs.wxml -->
<view class="container log-list">
  <block wx:for="{{logs}}" wx:for-item="log">
    <text class="log-item">{{index + 1}}. {{log}}</text>
  </block>
</view>
```

logs.wxss 的代码为：

```
.log-list {
   display: flex;
   flex-direction: column;
   padding: 40rpx;
}
.log-item {
   margin: 10rpx;
}
```

最后客户端会装载 logs.js，可以看到 logs.js 的大体内容如下。

```
//logs.js
const util = require('../../utils/util.js')

Page({
  data: {
    logs: []
  },
  onLoad: function () {
    this.setData({
      logs: (wx.getStorageSync('logs') || []).map(log => {
        return util.formatTime(new Date(log))
      })
    })
  }
})
```

Page 是一个页面构造器，这个构造器就生成了一个页面。在生成页面的时候，小程序框架会把 data 数据和 index.wxml 一起渲染出最终的结构。在渲染完界面之后，页面实例就会收到一个 onLoad 的回调，可以在这个回调中处理程序逻辑。

3.7 事件绑定

事件是一种用户的行为，如长按某一种图片（识别二维码，或进行编辑处理等操作）是一个事件，单击某个按钮是另外一个事件，还有其他很多不同的事件。事件也是一种视图层到逻辑层的通信方式，如当用户单击按钮的时候，UI 层会把一些信息发送给程序中的逻辑代码；事件可以绑定在组件上，当触发事件发生时，就会执行逻辑层中对应的事件处理函数；事件对象可以携带额外信息，如 id、dataset、touches。

3.7.1 事件的类别

每个控件都有自己的事件，事件分为四种类型，如表 3.16 所示。

表 3.16 事件类型

序 号	事件类型	详细信息
1	单击事件 tap	
2	长按事件 longtab	
3	触摸事件 touch	touchstart 开始触摸
		touchend 结束触摸
		touchmove 移动触摸
		touchcancel 取消触摸
4	其他的触摸事件 submit	

3.7.2 事件的使用方式

首先需在组件中绑定一个事件处理函数，如以下代码：

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

该代码定义该 view 组件的 id 为“tapTest”，自定义属性为 data-hi，绑定一个 bindtap 事件“tapName”，当用户单击该组件的时候会在该页面对应的 Page 中找到相应的事件处理函数。

接下来，就需要在相应的 Page 定义中添加相应的事件处理函数，参数为 event。

```
tapName:function(event){
    console.log(event)
}
```

当单击“Click me!”时，在控制台显示出来的信息大致为：

```
{
  "type":"tap",
  "timeStamp":4117,
  "target": {
       "dataset": {
          "hi":"WeChat"
       }
       "id": "tapTest",
   },
   "currentTarget": {
     "dataset": {
          "hi":"WeChat"
       }
       "id": "tapTest",
   },
   "detail": {
      "x":48.79999542236328,
```

```
        "y":656.6000366210938
      },
      "touches":[{
        clientX: 49.60000228881836
        clientY: 491.20001220703125
        force: 1
        identifier: 0
        pageX: 49.60000228881836
        pageY: 657.6000366210938
    }],
    "changedTouches":[{
        clientX: 49.60000228881836
        clientY: 491.20001220703125
        force: 1
        identifier: 0
        pageX: 49.60000228881836
        pageY: 657.6000366210938
    }]
}
```

3.7.3 冒泡事件与非冒泡事件

事件分为冒泡事件和非冒泡事件。冒泡事件是指，当一个组件上的事件被触发后，该事件会向父节点传递；非冒泡事件是指，当一个组件上的事件被触发后，该事件不会向父节点传递。

这里举例说明什么是冒泡事件。

首先打开 index.wxss 文件，定义 3 个 view 组件的样式，在该 WXSS 文件中增加以下代码。

```
.view1{
  height:500rpx;
  width:100%;
  background-color:cyan
}
.view2{
  height:300rpx;
  width:80%;
  background-color:greenyellow
}
.view3{
  height:100rpx;
  width:60%;
  background-color:red
}
```

然后在启动界面 index.wxml 中，创建 3 个逐层嵌套的 view 组件，代码如下。

```
<view class="view1" bindtap="clickView1">
    HelloWorld1
    <view class="view2" bindtap="clickView2">
        HelloWorld2
```

```
        <view class = "view3" bindtap = "clickView3">
            HelloWorld3
        </view>
    </view>
</view>
```

再在 index.js 文件中定义单击事件处理函数，增加以下代码，在控制台输出对应的 log 信息。

```
clickView1:function(){
    console.log("您单击了 view1")
},
clickView2: function () {
    console.log("您单击了 view2")
},
clickView3: function () {
    console.log("您单击了 view3")
},
```

以上代码完成以后，进行“编译”，观看运行效果。可以发现，当单击 HelloWorld1 的时候，在控制台显示如图 3.14 所示信息。

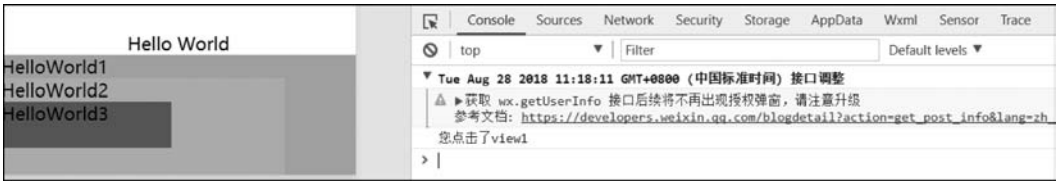

图 3.14　单击 view1 时 console 输出的信息

当单击 HelloWorld2 的时候，在控制台显示如图 3.15 所示信息。

当单击 HelloWorld3 的时候，在控制台显示如图 3.16 所示信息。

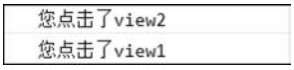

图 3.15　单击 view2 时 console 输出的信息

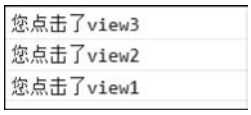

图 3.16　单击 view3 时 console 输出的信息

为什么会出现这种情况？首先来分析一下 index.wxml 文件中 3 个 view 组件的嵌套层次关系，如图 3.17 所示。第 1 个 view 组件(view1)是第 2 个 view 组件(view2)的父节点，第 2 个 view 组件(view2)又是第 3 个 view 组件(view3)的父节点。

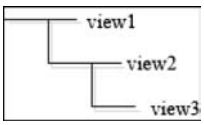

图 3.17　view 组件的层次关系

也就是说，当单击子 view 组件的时候，同时也触发了所有父 view 节点的事件，这种触发事件就称为冒泡事件；如果不触发父节点的事件，则即为非冒泡事件。

WXML 的冒泡事件主要如表 3.17 所示。

表 3.17　冒泡事件主要列表

序号	事 件 类 型	触 发 条 件
1	touchstart	手指触摸动作开始
2	touchmove	手指触摸后移动
3	touchcancel	手指触摸动作被打断，如来电提醒、弹窗等
4	touchend	手指触摸动作结束
5	tap	手指触摸后马上离开
6	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
7	longtap	手指触摸后，超过 350ms 再离开

除表 3.17 中所列事件类型之外的其他组件自定义事件都是非冒泡事件，如< form >的 submit 事件，< input >的 input 事件，< scroll-view >的 scroll 等。

3.7.4　事件绑定和冒泡

事件绑定的写法同组件的属性，为 key 和 value 的形式。key 以 bind 或 catch 开头，然后跟上事件的类型，如 bindtap、catchtouchstart。value 是一个字符串，需要在对应的 Page 中定义同名的函数。

需要注意的一点是：bind 事件绑定不会阻止冒泡事件向上冒泡，但 catch 事件绑定可以阻止冒泡事件向上冒泡。

如将上例中 index.wxml 中组件的事件绑定由 bind 改为 catch，具体如下。

```
< view class = "view1" catchtap = "clickView1">
   HelloWorld1
   < view class = "view2" catchtap = "clickView2">
      HelloWorld2
      < view class = "view3" catchtap = "clickView3">
         HelloWorld3
      </view >
  </view >
</view >
```

则在运行界面中，依次单击 3 个 view 组件，尽管 tap 类型组件为冒泡组件，均不会触发父节点的事件，显示结果如图 3.18 所示。

图 3.18　catch 事件绑定的显示效果

下面在这个例子的基础上增加一点儿程序设计的逻辑，即如何让文字的背景及颜色随着单击事件进行变化。

首先，在 index.wxss 中添加两个样式，一个用于显示购物车信息的 view 组件，另一个用于 text 组件。

```
.cart{
  float: center;
  text-align: center;
  padding: 5rpx 20rpx;
}
.cart-text{
  font-style: normal;
  font-weight: bold;
  font-size: 30rpx;
  line-height: 50rpx;
}
```

然后打开 index.wxml 文件，添加如下代码。

```
<view class="cart" style="background-color:{{BgColor}}" bindtap='add2Cart'>
    <text class="cart-text" style="color:{{TextColor}}">{{add2Cart}}</text>
</view>
```

其中，view 组件中的 style="background-color:{{BgColor}}"用于定义背景颜色，其中的变量 BgColor 在 index.js 中的 data 里面定义，'add2Cart'为该 view 组件的事件触发函数；text 组件中的 style="color:{{TextColor}}"用于定义该组件显示的文字颜色，add2Cart 为 index.js 文件中 data 里面定义的变量。

以上几个 data 数据域中定义的变量的相应代码为：

```
//index.js
Page({
  data: {
    …
    BgColor: '#5cb85c',
    TextColor:"red",
    add2Cart: "添加至购物车",
  },
  …
})
```

接下来定义 view 组件单击触发的事件函数代码，如下。

```
add2Cart: function () {
    var bgColor;              //定义局部变量,用于存放 view 组件的背景色
    var textColor;            //定义局部变量,用于存放 text 组件的文本颜色
    var add2cart;             //定义局部变量,用于改变 text 组件的文本信息

    if(flag){                 //此处的 flag 变量是一个 boolean 变量,用于改变显示信息
      bgColor = "blue";
      textColor = "gray";
```

```
      add2cart = "是否添加至购物车?";
      flag = false;
    }
    else{
      bgColor = "gray";
      textColor = "blue";
      add2cart = "已添加至购物车!";
      flag = true;
    }
    this.setData({            //将逻辑层的数据变化信息传送到视图层进行渲染显示
      BgColor: bgColor,
      TextColor: textColor,
      add2Cart: add2cart
    });
  },
```

关于上述代码中的 flag 变量的定义，如果放在 Page 中的 data 域来定义，编译就会报错，此时的一个处理方法，是将其放在 Page 的外面作为全局变量来定义，例如：

```
//index.js
//获取应用实例
const app = getApp()
var flag = true

Page({
  data: {
  …
```

则会通过系统编译，初始运行界面如图 3.19(a)所示，初始背景色 BgColor 为 #5cb85c，TextColor 颜色为 red，显示文本变量 add2Cart 初始化为“添加至购物车”；当第一次单击后，由于 flag 初始化为 true，因此执行 if-else 中的 if 分支，通过代码：

```
bgColor = "blue";
textColor = "gray";
add2cart = "是否添加至购物车?";
flag = false;
```

分别改变背景色、文本色、显示文本，以及 boolean 变量 flag 的信息，从 true 变为 false，这样当再一次单击的时候，就会执行到 if-else 中的 else 分支，通过代码：

```
bgColor = "gray";
textColor = "blue";
add2cart = "已添加至购物车!";
flag = true;
```

再次改变显示信息以及 flag 变量的值，这样当下一次单击的时候，又会交替执行 if-else 中的 if 分支，如此不断变化下去。

最后部分的代码非常关键，它完成逻辑层与渲染层的交互，代码如下：

```
this.setData({
     BgColor: bgColor,
```

```
        TextColor: textColor,
        add2Cart: add2cart
    });
```


通过更新 data 中的变量，将逻辑层的数据变化信息传送到视图层进行渲染显示，最终才会得到图 3.19 的显示效果。

图 3.19　不同执行阶段触发渲染的不同界面

思　考　题

1. WXML 与 WXSS 各自在前端页面开发中分别起什么作用？二者缺其一会怎么样？
2. 微信小程序主要目录和文件的作用分别是什么？
3. 请谈谈 WXML 与标准的 HTML 的异同。