第3章 小程序开发基础

本章将从架构层面介绍小程序的开发框架，包括视图层描述语言 WXML（WeiXin Markup Language，微信标记语言）和 WXSS（WeiXin Style Sheets，微信样式表），以及基于 JavaScript 的逻辑层等。

本章学习目标：

- 了解小程序的生命周期与页面的生命周期。
- 了解小程序框架的基本功能。
- 了解接口和组件。
- 熟悉小程序注册和页面注册的方法。
- 熟悉模板，样式导入，模块化的操作方法。
- 掌握数据绑定的用法。
- 掌握列表渲染和条件渲染的使用方法。
- 掌握事件的绑定与处理方法。
- 掌握小程序样式文件的使用方法。

3.1 认识小程序的生命周期

【任务要求】

修改示例项目中的 app.js 文件，要求当小程序在执行以下操作时在调试器的 Console 面板中输出对应的信息。

（1）当小程序第一次启动时，输出当前系统的时间。

（2）当小程序被放置到后台时，输出“小程序已被隐藏”。

（3）当小程序从后台被唤醒时，输出进入的场景值。

【任务分析】

本次任务主要涉及对小程序生命周期中三种状态的处理，一个是第一次启动，然后是隐藏和显示。其中比较重要的一点是当小程序从后台被唤醒时，携带的包含场景值的参数，它可以用来帮助开发者更好地响应用户的操作。

【任务操作】

（1）打开示例项目的 app.js 文件，在 onLaunch()函数内部的开始处，新增如下所示的两行代码。

```
//引用 util 文件夹中的公共函数
```

```
const util = require('utils/util.js')
//将当前系统时间格式化输出
console.log(util.formatTime(new Date(Date.now())))
```


（2）在 app.js 文件的 globalData 参数前，新增如下用于处理小程序被隐藏和唤醒事件的函数。

```
onShow:function(opts) {
//输出 opts 中表示场景值的 scene 变量
  console.log(opts.scene)
},
onHide:function() {
  console.log('小程序已被隐藏')
},
```

完成后的 app.js 文件内容结构大致如下。

```
App({
  onLaunch:function(opts) {
    const util = require('utils/util.js')
    console.log(util.formatTime(new Date(Date.now())))
    // 展示本地存储能力
    …
  },
  onShow:function(opts) {
    console.log(opts.scene)
  },
  onHide:function() {
    console.log('小程序已被隐藏')
  },
  globalData: {
    userInfo: null
  }
})
```

注意：代码中加粗部分内容仅作阅读重点提示用，无其他特殊含义。下同。

（3）保存文件，编译项目，在 Console 面板中查看小程序启动时输出的时间信息，如图 3-1 所示。

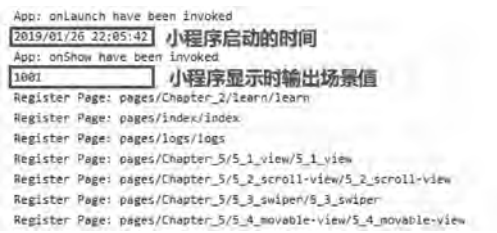

图 3-1　小程序启动时的输出

可以看到，小程序的第一次启动触发了 onLaunch()函数和 onShow()函数。

（4）在工具栏中单击“切后台”按钮，让小程序进入后台状态，然后在模拟器中选择

“1011：扫描二维码”，唤醒小程序到前台。在这个过程中观察到的输出如图 3-2 所示。

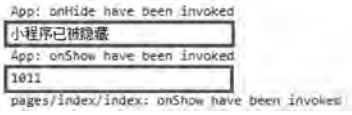

图 3-2　小程序转入后台和切换前台时输出

【相关知识】

小程序主要有前台和后台两种运行状态。当小程序处于前台时，它可以调用所有的 API(Application Programming Interface，应用程序编程接口)，为用户提供服务；当用户单击小程序运行页面左上角“关闭”按钮，或者按了设备 Home 键离开微信，小程序就转入了后台，此时小程序只能调用部分 API，并随时可能被销毁。小程序只有在进入后台一定时间后，或者系统资源占用过高时，才会被真正销毁。当用户再次进入微信或再次打开小程序，又会从后台变为前台状态。

场景值表示的是小程序是通过什么途径从后台切换到前台的。目前小程序支持的场景值说明见表 3-1。

表 3-1　小程序场景值说明

场景值 ID	说　明	场景值 ID	说　明	场景值 ID	说　明
1001	发现栏小程序主入口，“最近使用”列表(基础库 2.2.4 版本起包含“我的小程序”列表)	1032	手机相册选取一维码	1064	微信连 WiFi 状态栏
1005	顶部搜索框的搜索结果页	1034	微信支付完成页	1067	公众号文章广告
1006	发现栏小程序主入口搜索框的搜索结果页	1035	公众号自定义菜单	1068	附近小程序列表广告
1007	单人聊天会话中的小程序消息卡片	1036	App 分享消息卡片	1069	移动应用
1008	群聊会话中的小程序消息卡片	1037	小程序打开小程序	1071	钱包中的银行卡列表页
1011	扫描二维码	1038	从另一个小程序返回	1072	二维码收款页面
1012	长按图片识别二维码	1039	摇电视	1073	客服消息列表下发的小程序消息卡片
1013	手机相册选取二维码	1042	添加好友搜索框的搜索结果页	1074	公众号会话下发的小程序消息卡片
1014	小程序模板消息	1043	公众号模板消息	1077	摇周边
1017	前往体验版的入口页	1044	带 shareTicket 的小程序消息卡片 详情	1078	连 WiFi 成功页
1019	微信钱包	1045	朋友圈广告	1079	微信游戏中心
1020	公众号 profile 页相关小程序列表	1046	朋友圈广告详情页	1081	客服消息下发的文字链

续表

场景值 ID	说　明	场景值 ID	说　明	场景值 ID	说　明
1022	聊天顶部置顶小程序入口	1047	扫描小程序码	1082	公众号会话下发的文字链
1023	安卓系统桌面图标	1048	长按图片识别小程序码	1084	朋友圈广告原生页
1024	小程序 profile 页	1049	手机相册选取小程序码	1089	微信聊天主界面下拉，“最近使用”栏（基础库 2.2.4 版本起包含“我的小程序”栏）
1025	扫描一维码	1052	卡券的适用门店列表	1090	长按小程序右上角菜单唤出最近使用历史
1026	附近小程序列表	1053	搜一搜的结果页	1091	公众号文章商品卡片
1027	顶部搜索框搜索结果页“使用过的小程序”列表	1054	顶部搜索框小程序快捷入口	1092	城市服务入口
1028	我的卡包	1056	音乐播放器菜单	1095	小程序广告组件
1029	卡券详情页	1057	钱包中的银行卡详情页	1096	聊天记录
1030	自动化测试下打开小程序	1058	公众号文章	1097	微信支付签约页
1031	长按图片识别一维码	1059	体验版小程序绑定邀请页	1099	页面内嵌插件
1102	公众号 profile 页服务预览				

注意：由于 Android 系统限制，目前还无法获取到按 Home 键退出到桌面，然后从桌面再次进小程序的场景值，对于这种情况，会保留上一次的场景值。

监听小程序生命周期变化的函数见表 3-2。

表 3-2　小程序生命周期函数

属　性	类　型	描　述	触发时机
onLaunch()	Function	生命周期回调——监听小程序初始化	小程序初始化完成时（全局只触发一次）
onShow()	Function	生命周期回调——监听小程序显示	小程序启动或从后台进入前台显示时
onHide()	Function	生命周期回调——监听小程序隐藏	小程序从前台进入后台时

其中，onLaunch()函数的回调参数说明见表 3-3。

表 3-3　onLaunch()函数回调参数说明

属　性	类　型	说　明
path	String	启动小程序的路径
scene	Number	启动小程序的场景值
query	Object	启动小程序的 query 参数

续表

属　性	类　型	说　明
shareTicket	String	当其他用户通过分享的小程序卡片打开小程序时，该字段包含转发的信息
referrerInfo	Object	来源信息。从另一个小程序、公众号或 App 进入小程序时返回。否则返回{}

onShow()函数的回调参数说明见表 3-4。

表 3-4　onShow 函数回调参数说明

属　性	类　型	说　明
path	String	小程序切前台的路径
scene	Number	小程序切前台的场景值
query	Object	小程序切前台的 query 参数
ShareTicket	String	当其他用户通过分享的小程序卡片打开小程序时，该字段包含转发的信息
referrerInfo	Object	来源信息。从另一个小程序、公众号或 App 进入小程序时返回。否则返回{}

referrerInfo 的结构说明见表 3-5。

表 3-5　referrerInfo 结构说明

属　性	类　型	说　明
AppID	String	来源小程序、公众号或 App 的 AppID
extraData	Object	来源小程序传过来的数据，scene＝1037 或 1038 时支持

当从表 3-6 中的场景进入小程序时，返回的 referrerInfo 才是有效的。

表 3-6　返回有效 referrerInfo 的场景

场　景　值	场　景	AppID 含义
1020	公众号 profile 页相关小程序列表	来源公众号
1035	公众号自定义菜单	来源公众号
1036	App 分享消息卡片	来源 App
1037	小程序打开小程序	来源小程序
1038	从另一个小程序返回	来源小程序
1043	公众号模板消息	来源公众号

3.2　认识小程序页面的生命周期

【任务要求】

为当前示例项目的首页和查看日志页添加监听页面生命周期变化的函数，并在对应的生命周期处理函数中编写向调试器 Console 面板输出页面状态变化的代码。在首页跳转到

日志页面时，需要携带名为 message1，值为 Hello 以及名为 message2，值为 Logs 的两个参数，并在日志页面输出获取到的参数。完成后在两个页面之间跳转，观察控制台输出，理解小程序页面生命周期的变化。

【任务分析】

除了小程序本身的生命周期外，小程序的每个页面也有自己的生命周期。相较于小程序的生命周期来说，每个页面的生命周期包含加载、准备、显示、隐藏和卸载这几个阶段。本次任务使用现有的两个页面之间的跳转，通过观察对应生命周期函数的输出来理解页面生命周期的变化。

【任务操作】

(1) 打开示例项目，在 index 页面的 index.js 文件中，修改 onLoad 函数，使其可以在调试器中输出运行的信息。新增用于监听页面生命周期变化的 onShow()、onReady()、onHide()和 onUnload()函数。同时修改 bindViewTap()函数中的 URL 部分，使其带着参数 message1 和 message2 跳转到日志页面。完成后的 index.js 文件内容大致如下。

```
//index.js
Page({
  data: {
    …
  },
  //事件处理函数
  bindViewTap: function() {
    wx.navigateTo({
      url: '../logs/logs?message1 = Hello&message2 = Logs'
    })
  },
  onLoad: function () {
    console.log("首页加载")
    if (app.globalData.userInfo) {
      …
    }
  },
  getUserInfo: function(e) {
    …
  },
  onShow: function () {
    console.log("首页显示")
  },
  onReady: function () {
    console.log("首页渲染完成")
  },
  onHide: function () {
    console.log("首页隐藏")
  },
  onUnload: function () {
    console.log("首页卸载")
  }
})
```

(2) 打开 logs.js,在 onLoad()函数中新增一行代码用于输出首页传递过来的参数信息。同样新增用于监听页面生命周期变化的 onShow(),onReady(),onHide()和 onUnload()函数。完成后的 logs.js 函数大致如下。

```
//logs.js
Page({
  data: {
    …
  },
  onLoad: function (query) {
    console.log("日志页加载", query)
    this.setData({
      …
      })
    })
  },
  onShow: function () {
    console.log("日志页显示")
  },
  onReady: function () {
    console.log("日志页渲染完成")
  },
  onHide: function () {
    console.log("日志页隐藏")
  },
  onUnload: function () {
    console.log("日志页卸载")
  }
})
```

(3) 保存文件,编译项目。在首页上单击用户头像跳转到日志页面,观察如图 3-3 所示调试器的输出。

图 3-3　页面生命周期函数调用示例

【相关知识】

小程序页面的生命周期，会经历加载、显示、渲染完成、隐藏和卸载这几个过程，涉及的生命周期回调函数说明见表 3-7。

表 3-7　页面生命周期回调函数

函　数　名	描　　述
onLoad()	页面加载时触发。一个页面只会调用一次，可以在 onLoad()的参数中获取打开当前页面路径中的参数
onShow()	页面显示/切入前台时触发
onReady()	页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互
onHide()	页面隐藏/切入后台时触发。如 navigateTo 或底部 Tab 切换到其他页面，小程序切入后台等
onUnload()	页面卸载时触发。如 redirectTo 或 navigateBack 到其他页面时

在小程序中，所有页面的路由以栈的形式维护。当发生页面切换时，页面栈的变化见表 3-8 的说明。

表 3-8　各种情况下页面栈的表现

路 由 方 式	页面栈表现
初始化	新页面入栈
打开新页面	新页面入栈
页面重定向	当前页面出栈，新页面入栈
页面返回	页面不断出栈，直到目标返回页
Tab 切换	页面全部出栈，只留下新的 Tab 页面
重加载	页面全部出栈，只留下新的页面

在上述路由方式中，每种情况下的路由触发方式以及对应页面响应的生命周期函数见表 3-9。

表 3-9　路由触发时机和页面生命周期函数

路由方式	触 发 时 机	路由前页面	路由后页面
初始化	小程序打开的第一个页面		onLoad()，onShow()
打开新页面	调用 wx. navigateTo 接口或使用组件< navigator open-type="navigateTo"/>	onHide()	onLoad()，onShow()
页面重定向	调用 wx. redirectTo 接口或使用组件< navigator open-type="redirectTo"/>	onUnload()	onLoad()，onShow()
页面返回	调用 wx. navigateBack 接口或使用组件< navigator open-type="navigateBack">或用户单击左上角返回按钮	onUnload()	onShow()
Tab 切换	调用 wx. switchTab 接口或使用组件< navigator open-type="switchTab"/>或用户切换 Tab		参考表 3-10
重启动	调用 wx. reLaunch 接口或使用组件 < navigator open-type="reLaunch"/>	onUnload	onLoad()，onShow()

Tab 切换时对应页面的生命周期变化见表 3-10(以 A、B 页面为 tabBar 页面,C 是从 A 页面打开的页面,D 页面是从 C 页面打开的页面为例)。

表 3-10　Tab 切换页面的生命周期变化

当 前 页 面	路由后页面	触发的生命周期(按顺序)
A	A	
A	B	A. onHide(), B. onLoad(), B. onShow()
A	B(再次打开)	A. onHide(), B. onShow()
C	A	C. onUnload(), A. onShow()
C	B	C. onUnload(), B. onLoad(), B. onShow()
D	B	D. onUnload(), C. onUnload(), B. onLoad(), B. onShow()
D(从转发进入)	A	D. onUnload(), A. onLoad(), A. onShow()
D(从转发进入)	B	D. onUnload(), B. onLoad(), B. onShow()

关于页面的路由,需要注意以下几点。

(1) navigateTo(),redirectTo()只能打开非 tabBar 页面。

(2) switchTab()只能打开 tabBar 页面。

(3) reLaunch()可以打开任意页面。

(4) 页面底部的 tabBar 由页面决定,即只要是定义为 tabBar 的页面,底部都有 tabBar。

tabBar 部分可以参见 2.1 节内容,涉及的接口可以参见 14.7 节的内容,组件可以参考 9.1 节的内容。

在本次任务中,可以看到,加载首页的时候依次执行的函数是 onLoad(),onShow()和 onReady()。在单击首页头像后,首页被隐藏,首页的 onHide()函数被执行,然后跳转到日志页面。日志页面的 onLoad()函数运行,同时也获取到了首页传递过来的参数 message1 和 message2。紧接着,日志页面的 onShow()和 onReady()函数被执行。

在 index.js 文件中,为要跳转到的 logs 页面地址添加了 message1 和 message2 两个参数。路径携带参数的格式要求为:参数与路径之间使用"?"分隔,参数键与参数值之间用"="相连,不同参数用"&"分隔,例如'path?key=value&key2=value2'。

结合小程序的生命周期和页面的生命周期,在小程序运行的过程中,各个生命周期函数的调用顺序可参考如图 3-4 所示说明。

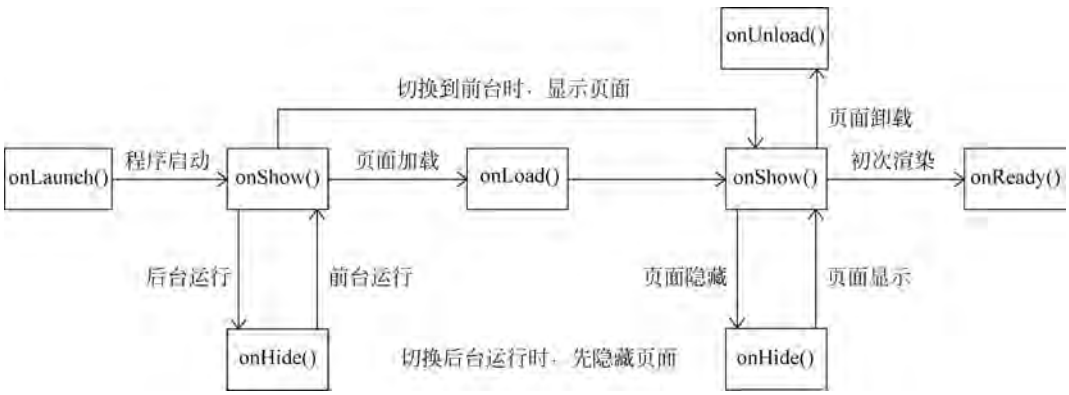

图 3-4　小程序和页面生命周期变化

3.3 概览 MINA 框架

【任务要求】

在 pages 目录下新建一个 Chapter_3 目录，在 Chapter_3 目录下新建一个名为 mina 的页面。要求在页面上显示文本“This is a text”和一个 Change Text 按钮，并实现单击按钮后将文本更改为“This is another text”的功能。

【任务分析】

MINA 是小程序使用的框架，其核心是一个响应的数据绑定系统。本次任务通过实现一个简单的单击按钮更改文字的功能，来体会框架中视图层和逻辑层的联系。

【任务操作】

(1) 打开示例项目，在 app.json 文件的 pages 数组中，新增第一项"pages/Chapter_3/mina/mina"。保存文件，编译项目，可以看到开发者工具已经建好了 mina 页面，模拟器中正在运行的也是 mina 页面。完成后的目录结构如图 3-5 所示。

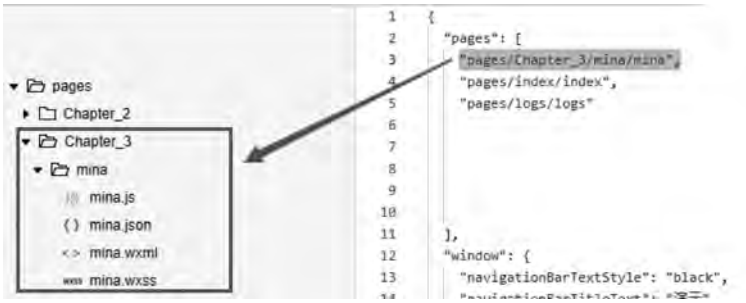

图 3-5 新建 mina 页面后的项目目录结构

(2) 打开 mina.js 文件，在 Page 函数的页面初始数据处，添加一个名为 text，值为“This is a text”的初始数据。完成后的 mina.js 文件内容大致结构如下。

```
// pages/Chapter_3/mina/mina.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    text:"This is a text"
  },
  ...
})
```

(3) 打开 mina.wxml 文件，删除原文件内容，新增用于显示 mina.js 文件中 text 变量的代码，同时增加一个按钮，在按钮的属性中，为单击事件绑定一个名为 changeText()的函数。完成后的 mina.wxml 文件内容如下。

```
<!--pages/Chapter_3/mina/mina.wxml-->
<text>{{text}}</text>
```

```
<button bindtap = 'changeText'> Change Text </button>
```

(4) 回到 mina.js 文件,需要为按钮单击事件的处理函数 changeText()编写代码逻辑,实现更改文字的功能。完成后的 mina.js 文件内容结构大致如下。

```
// pages/Chapter_3/mina/mina.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    text:"This is a text"
  },
  ...
  /**
   * 用户单击右上角分享
   */
  onShareAppMessage: function () {

  },
  changeText:function(){
    this.setData({
      text:"This is another text"
    })
  }
})
```

(5) 保存所有文件,编译项目,在模拟器中查看 mina 页面表现,单击按钮,观察文字是否被改变。正常情况下,页面表现应如图 3-6 所示。

图 3-6　mina 页面单击按钮前(左)和单击按钮后(右)

【相关知识】

MINA 框架(后文简称为“框架”)是微信小程序的开发框架。框架的目标是通过尽可能简单、高效的方式让开发者可以在微信中开发具有原生应用体验的服务。

框架中,整个小程序的内容分为两个层次:视图层(View)和逻辑层(App Service)。视图层是小程序的“外观”,它规定了小程序页面的结构和样式,决定了小程序内容的展示;逻辑层是小程序的“内涵”,它控制了小程序的生命周期和数据处理等。除了处理本地的业务逻辑之外,在开发小程序的过程中逻辑层往往还需要通过 HTTP 同远程服务器进行数据

交流。

框架提供了自己的视图层描述语言规范：WXML 和 WXSS，以及基于 JavaScript 的逻辑层，并在视图层与逻辑层间提供了数据传输和事件系统，可以让开发者将重点放在数据与视图上。

整个框架中最核心的部分就是视图层和逻辑层之间的数据绑定和事件响应系统。视图层中的信息通过事件绑定携带参数向逻辑层传递，逻辑层的数据通过数据绑定向视图层传递。框架可以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。

在本次任务中，通过框架将逻辑层（mina.js 文件）数据中的 text 与视图层（mina.wxml 文件）的 text 进行了绑定，所以在页面一打开的时候会显示“This is a text”。当单击按钮的时候，触发了按钮的单击事件，视图层会发送单击事件的处理函数 changeText()给逻辑层，逻辑层找到并执行对应的函数。changeText()函数触发后，逻辑层执行 setData 的操作，将 data 中的 text 从“This is a text”变为“This is another text”，因为该数据和视图层已经绑定了，从而视图层上显示的内容会自动改变为“This is another text”。

框架除了最为核心的数据绑定和事件响应系统外，还管理了整个小程序的页面路由，可以做到页面间的无缝切换，并给予页面完整的生命周期。开发者需要做的只是将页面的数据、方法、生命周期函数注册到框架中，其他的一切复杂的操作都交由框架处理。有关页面生命周期和路由的介绍，可以参见 3.2 节内容。

框架提供了一套基础的组件，这些组件自带微信风格的样式以及特殊的逻辑，开发者可以通过组合基础组件，创建出强大的微信小程序。大量的预置组件如文本组件< text ></text >、轮播组件< swiper ></swiper >等使开发者能够专注业务逻辑的编写，快速开发出需要的小程序。

框架还提供了丰富的微信原生 API，使用这些 API 可以方便地调用微信提供的能力，如获取用户信息、本地存储、支付等。

3.4 逻 辑 层

小程序开发框架的逻辑层使用 JavaScript 引擎为小程序提供开发者 JavaScript 代码的运行环境以及微信小程序的特有功能。

逻辑层将数据进行处理后发送给视图层，同时接受视图层的事件反馈。开发者写的所有代码最终将会打包成一份 JavaScript 文件，并在小程序启动的时候运行，直到小程序销毁。这一行为类似 Service Worker，所以逻辑层也称为 App Service。

在 JavaScript 的基础上，微信小程序还增加了一些新的功能，以方便小程序的开发。

（1）增加 App()和 Page()方法，进行程序和页面的注册。

（2）增加 getApp()和 getCurrentPages()方法，分别用来获取 App 实例和当前页面栈。

（3）提供丰富的 API，如微信用户数据、扫一扫、支付等微信特有能力。

（4）每个页面有独立的作用域，并提供模块化能力。

需要注意的是，小程序框架的逻辑层并非运行在浏览器中，因此 JavaScript 在 Web 中的一些能力都无法使用，如 window、document 等。

3.4.1 注册程序

【任务要求】

新建一个临时的项目，要求在建立的时候，不勾选任何启动模板，直接建立一个完全空白的项目，然后手动新建必要的 app.js 以及 app.json 文件。除了需要在 app.js 文件中注册小程序所有的生命周期函数以外，还需要注册错误监听函数以及页面不存在监听函数，同时，还需要在 app.js 中设置一个名为 globalData，值为“This is global data”字符串的变量。在 app.json 文件中，注册一个名为 demo 的页面，并让该页面显示 globalData 的值。

【任务分析】

注册程序，意即让开发者工具知道当前的目录和文件是一个小程序并将其作为小程序进行处理的步骤。之前的任务，我们一直都是使用基于“普通快速启动模板”（详见 1.3 节）建立的小程序项目。本次任务，需要自己手动从一个空白的目录建立小程序，并熟悉声明注册一个小程序的过程，体会哪些文件是小程序必需的。

【任务操作】

（1）新建一个名为 Blank_Project 的项目，具体操作如图 3-7 所示。注意不要勾选图中的“建立普通快速启动模板”复选框。

图 3-7　新建空白小程序项目

（2）新建 app.js 文件和 app.json 文件。app.js 的文件内容如下。

```
//app.js
App({
  onLaunch: function (options) {
    console.log("小程序启动",options)
  },
  onShow: function (options) {
    console.log("小程序显示",options)
```

```
  },
  onHide: function (options) {
    console.log("小程序隐藏",options)
  },
  onError: function (msg) {
    console.log("小程序发生错误",msg)
  },
  onPageNotFound:function(msg){
    console.log("小程序要打开的页面不存在",msg)
  },
  globalData: 'This is global data'
})
```

app.json 文件内容如下。

```
{
  "pages": [
    "Demo/demo"
  ]
}
```

保存文件,编译项目,完成后的项目目录结构如图 3-8 所示。

图 3-8　Blank_Project 项目目录

(3) 打开 demo.js 文件,在第一行使用 getApp()函数获取小程序实例,在 data 中,将小程序实例中的 globalData 数据赋给页面的 message 变量。随后打开 demo.wxml 文件,在 < text >< /text >标签中使用{{message}}来显示 demo.js 中 message 变量的值。完成后的 demo.js 文件大致如下。

```
// Demo/demo.js
const appInstance = getApp()
Page({

  /**
   * 页面的初始数据 */
  data: {
    message:appInstance.globalData
  },
  ...
})
```

demo.wxml 文件内容如下。

```
<! -- Demo/demo.wxml -- >
< text >{{message}}</text >
```

(4) 编译运行,可以看到在模拟器中显示出了 demo 页面的内容(如图 3-9 所示),调试区输出了小程序生命周期的相关内容(如图 3-10 所示)。

图 3-9　模拟器页面输出

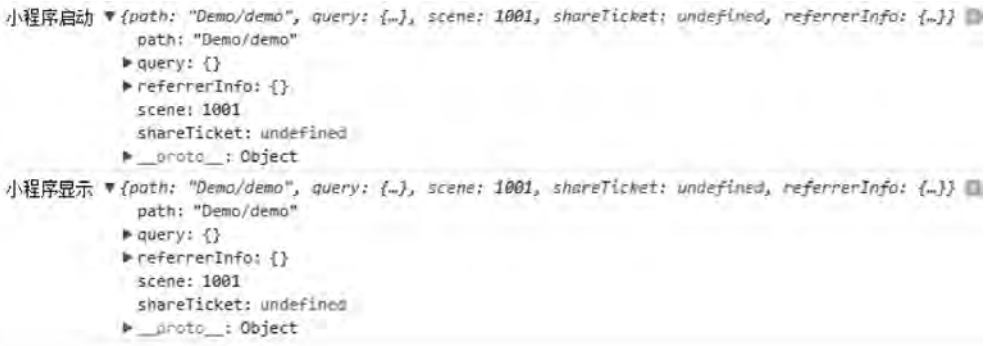

图 3-10　调试区输出

【相关知识】

在本次任务中,最为重要的便是 app.js 文件中的 App(Object)函数。App()函数用来注册一个小程序。接受一个 Object 参数,用于指定小程序的生命周期回调等。App()必须在 app.js 中调用,必须调用且只能调用一次,不然会出现无法预期的后果。

Object 参数说明见表 3-11。

表 3-11　App 函数参数说明

属　性	类　型	描　述	触发时机
onLaunch()	Function	生命周期回调—监听小程序初始化	小程序初始化完成时(全局只触发一次)
onShow()	Function	生命周期回调—监听小程序显示	小程序启动或从后台进入前台显示时
onHide()	Function	生命周期回调—监听小程序隐藏	小程序从前台进入后台时
onError()	Function	错误监听函数	小程序发生脚本错误,或者 API 调用失败时触发,会带上错误信息
onPageNotFound()	Function	页面不存在监听函数	小程序要打开的页面不存在时触发,会带上页面信息回调该函数
其他	Any	开发者可以添加任意的函数或数据到 Object 参数中,用 this 可以访问	

其中,有关小程序生命周期函数的相关内容可以参考 3.1 节。

onError(String error)函数会在小程序发生脚本错误或 API 调用报错时触发。string error 表示错误信息,包含堆栈信息。

onPageNotFound(Object)函数会在小程序要打开的页面不存在时触发。其参数说明见表 3-12。

表 3-12 onPageNotFound 函数参数说明

属 性	类 型	说 明
path	String	不存在页面的路径
query	Object	打开不存在页面的 query 参数
isEntryPage	Boolean	是否本次启动的首个页面(例如从分享等入口进来,首个页面是开发者配置的分享页面)

一个简单的使用示例如下。

```
App({
  onPageNotFound(res) {
    wx.redirectTo({
      url: 'pages/...'
    }) // 如果是 tabBar 页面,请使用 wx.switchTab
  }
})
```

当小程序页面不存在时,需要注意以下几点。

(1) 开发者可以在回调中进行页面重定向,但必须在回调中同步处理,异步处理(例如 setTimeout 异步执行)无效。

(2) 若开发者没有处理页面不存在的情况,当跳转页面不存在时,将打开微信客户端原生的页面不存在提示页面。

(3) 如果回调中又重定向到另一个不存在的页面,将打开微信客户端原生的页面不存在提示页面,并且不再第二次回调。

getApp(Object)是一个全局函数,可以用来获取到小程序 App 实例。一个简单的使用 getApp 函数的示例如下。

```
// other.js
const appInstance = getApp()
console.log(appInstance.globalData)     // I am global data
```

使用 getApp()函数需要注意以下两点。

(1) 不要在定义于 App()内的函数中调用 getApp(),使用 this 就可以得到 App 实例。

(2) 通过 getApp()获取实例之后,不要私自调用生命周期函数。

3.4.2 注册页面

【任务要求】

打开之前的示例项目(非上个任务的空白项目),在 pages/Chapter_3 文件夹下新建一个名为 page 的文件夹,并在里面新建一个名为 page 的页面,要求如下。

(1) 当页面显示时,在调试器的 Console 面板输出当前页面的路径;
(2) 添加页面的初始数据,分别包括文本、数字、JSON 对象和数组四种数据类型;
(3) 在页面上显示所有的初始数据,同时添加 4 个按钮用于修改其内容;
(4) 添加一个按钮用于新增一个数据并在页面上显示出来;
(5) 设置自定义分享的标题,将当前页面分享给他人。
完成后的页面示例如图 3-11 所示。

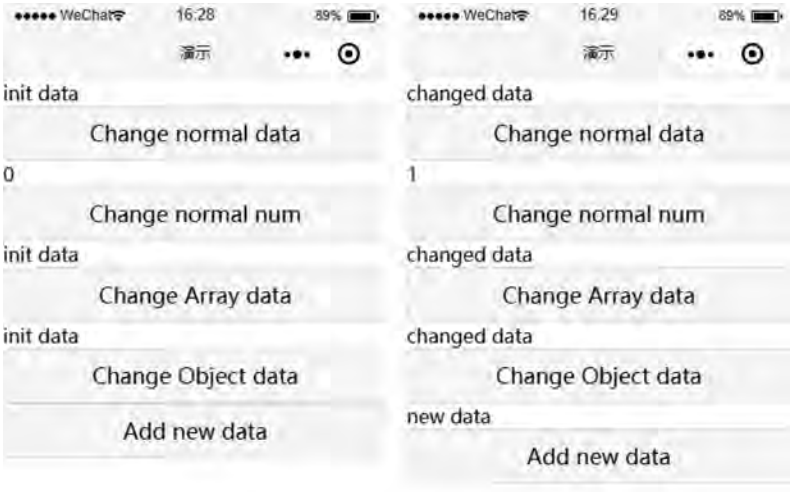

图 3-11　初始页面(左)和单击按钮后页面(右)

【任务分析】

对页面的相关处理,可以说是在开发小程序的时候打交道最多的操作了。同注册程序需要一个 App()函数一样,注册页面也需要一个 Page()函数。通过设置 Page()函数中的参数,可以实现对页面生命周期的处理,监听页面的事件和处理组件事件。同时,注册程序里面还有一个非常重要的用于逻辑层和视图层数据同步的机制。这些都是十分重要的。

【任务操作】

(1) 打开示例项目,在 app.json 文件的 pages 数组中,新增第一项"pages/Chapter_3/page/page"。保存文件,编译项目,让开发者工具自动生成 page 页面所需的文件。

(2) 打开 page.wxml,将其中的内容替换成如下代码。

```
<!-- pages/Chapter_3/page/page.wxml -->
<view>{{text}}</view>
<button bindtap="changeText">Change normal data</button>
<view>{{num}}</view>
<button bindtap="changeNum">Change normal num</button>
<view>{{array[0].text}}</view>
<button bindtap="changeItemInArray">Change Array data</button>
<view>{{object.text}}</view>
<button bindtap="changeItemInObject">Change Object data</button>
<view>{{newField.text}}</view>
<button bindtap="addNewField">Add new data</button>
```

(3) 打开 page.js,将其中的内容替换成如下代码。

```
// pages/Chapter_3/page/page.js
Page({
  data: {
    text: 'init data',
    num: 0,
    array: [{ text: 'init data' }],
    object: {
      text: 'init data'
    }
  },
  onShow(){
    console.log(this.route)
  },
  changeText() {
    // this.data.text = 'changed data' // 不要直接修改 this.data
    // 应该使用 setData
    this.setData({
      text: 'changed data'
    })
  },
  changeNum() {
    // 或者,可以修改 this.data 之后马上用 setData 设置修改了的字段
    this.data.num = 1
    this.setData({
      num: this.data.num
    })
  },
  changeItemInArray() {
    // 对于对象或数组字段,可以直接修改一个其下的子字段,这样做通常比修改整个对象或数组更好
    this.setData({
      'array[0].text': 'changed data'
    })
  },
  changeItemInObject() {
    this.setData({
      'object.text': 'changed data'
    })
  },
  addNewField() {
    this.setData({
      'newField.text': 'new data'
    })
  },
  onShareAppMessage(res){
    if (res.from === 'menu') {
      // 来自右上角转发菜单
      console.log(res.target)
    }
    return {
```

```
    title: '注册页面示例',
    path: this.route
  }
 }
})
```

(4) 保存所有文件,编译项目并运行。可以看到页面的表现和按钮的作用如图 3-11 所示。观察调试器的 Console 面板,可以看到如图 3-12 所示输出的当前页面路径信息。

图 3-12　onShow 函数中使用 this.route 输出当前页面路径

(5) 在模拟器区单击右上角的“菜单”按钮,在弹出的选项中选择“转发”(如图 3-13 所示),可以看到带自定义标题的用当前页面截图生成的转发卡片(如图 3-14 所示),同时可以看到在 Console 面板中多了一行如图 3-15 所示的输出。

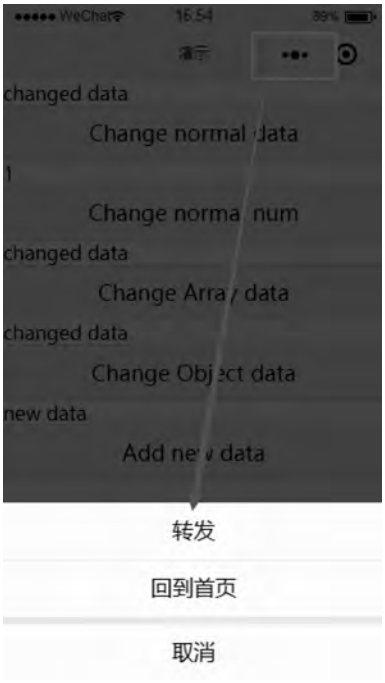

图 3-13　“转发”选项

图 3-14　转发卡片预览

图 3-15　输出转发信息

【相关知识】

Page(Object)函数用来注册一个页面。它接受一个 Object 类型参数,用于指定页面的初始数据、生命周期回调、事件处理函数等。其中,Object 参数内容说明见表 3-13。

表 3-13 Page 函数参数说明

属　性	类　型	描　述
data	Object	页面的初始数据
onLoad()	Function	生命周期回调—监听页面加载
onShow()	Function	生命周期回调—监听页面显示
onReady()	Function	生命周期回调—监听页面初次渲染完成
onHide()	Function	生命周期回调—监听页面隐藏
onUnload()	Function	生命周期回调—监听页面卸载
onPullDownRefresh()	Function	监听用户下拉动作
onReachBottom()	Function	页面上拉触底事件的处理函数
onShareAppMessage()	Function	用户单击右上角“转发”菜单
onPageScroll()	Function	页面滚动触发事件的处理函数
onResize()	Function	页面尺寸改变时触发,详见响应显示区域变化
onTabItemTap()	Function	当前是 tab 页时,单击 tab 时触发
其他	Any	开发者可以添加任意的函数或数据到 Object 参数中,在页面的函数中用 this 可以访问

其中,data 是页面第一次渲染使用的初始数据。页面加载时,data 将会以 JSON 字符串的形式由逻辑层传至渲染层,因此 data 中的数据必须是可以转成 JSON 的类型:字符串,数字,布尔值,对象,数组。在渲染层的 wxml 文件中,可以使用“{{var}}”的方式绑定数据。一个简单的示例片段如下。

```
<!-- example.wxml -->
<view>{{text}}</view>
<view>{{array[0].msg}}</view>

//example.js
Page({
  data: {
    text: 'init data',
    array: [{msg: '1'}, {msg: '2'}]
  }
})
```

在上面这个示例片段中,最终页面上会输出字符串 init data 和 1。

对页面的生命周期函数的介绍和页面的路由跳转,请参见表 3-7 的相关内容。此处需要单独提到的一点是,onLoad()函数带有一个 Object 类型的名为 query 的参数,该参数包含打开当前页面路径中的参数信息。例如,一个页面通过在路径中使用“url/?key=value”的方式携带了参数跳转到当前页面,那么在当前页面的 onLoad 函数加载时,便可以在其 query 参数中获取到以“key : value”形式存储的 JSON 数据对象。

onPullDownRefresh 函数用于监听用户的下拉刷新事件。需要注意的是,下拉刷新的

功能并不是默认启用的，如果要允许用户使用下拉刷新，可以在 app.json 的 window 选项中或页面配置中设置 enablePullDownRefresh 的值为 true。也可以使用 wx.startPullDownRefresh 触发下拉刷新，调用后触发下拉刷新动画，效果与用户手动下拉刷新一致。当处理完数据刷新后，调用 wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。

onReachBottom()函数用于监听用户上拉触底事件。开发者可以在 app.json 的 window 选项中或页面配置中设置触发距离 onReachBottomDistance，设置好后，用户滑到距离页面底部指定距离时，便会执行 onReachBottom 函数中的内容。用户在触发距离内滑动期间，本事件只会被触发一次。

onPageScroll(Object)用于监听用户滑动页面事件。其 Object 参数说明见表 3-14。

表 3-14　onPageScroll 函数参数说明

属　性	类　型	说　明
scrollTop	Number	页面在垂直方向已滚动的距离(单位为 px)

需要注意的是，只在需要的时候才在 page 中定义此方法，不要定义空方法，以减少不必要的事件派发对渲染层和逻辑层通信的影响。同时需要避免在 onPageScroll 中过于频繁地执行 setData 等引起逻辑层到渲染层通信的操作，尤其是每次传输大量数据时，这样会影响通信耗时。

onShareAppMessage(Object) 用于监听用户单击页面内“转发”按钮(< button > 组件 open-type="share")或右上角菜单“转发”按钮的行为，并自定义转发内容。只有定义了此事件处理函数，右上角菜单才会显示“转发”按钮。其参数说明见表 3-15。

表 3-15　onShareAppMessage 函数参数说明

参　数	类　型	说　明
from	String	转发事件来源。button 表示页面内“转发”按钮，menu 表示右上角的“转发”菜单
target	Object	如果 from 值是 button，则 target 是触发这次转发事件的 button，否则为 undefined
webViewUrl	String	页面中包含< web-view >组件时，返回当前< web-view >的 URL

此事件需要返回一个 Object，用于自定义转发内容。返回内容见表 3-16。

表 3-16　自定义转发内容

字　段	说　明	默 认 值
title	转发标题	当前小程序名称
path	转发路径	当前页面 path，必须是以/开头的完整路径
imageUrl	自定义图片路径，可以是本地文件路径、代码包文件路径或者网络图片路径。支持 PNG 及 JPG 格式。显示图片长宽比是 5∶4	使用默认截图

onTabItemTap(Object)在单击 tab 时触发,其参数说明见表 3-17。

表 3-17　onTabItemTap 参数说明

参　数	类　型	说　明	最低版本
index	String	被单击 tabItem 的序号,从 0 开始	1.9.0
pagePath	String	被单击 tabItem 的页面路径	1.9.0
text	String	被单击 tabItem 的按钮文字	1.9.0

一个使用的简单示例如下。

```
Page({
  onTabItemTap(item) {
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
  }
})
```

在 Page 中还可以定义组件事件处理函数。在渲染层的组件中加入事件绑定,当事件被触发时,就会执行 Page 中定义的事件处理函数。有关事件响应的详细信息,可以参照 3.5.1 节的内容。在本次任务中,就为多个按钮的单击事件绑定了相应的事件处理函数,用于实现每次单击,就更改数据的功能。

Page.route 是一个 String 类型的成员变量,表示当前页面的路径。在对应页面的 Page 函数中,可以使用 this.route 的方式来获取当前页面的路径信息。

Page.prototype.setData(Object data, Function callback) 函数用于将数据从逻辑层发送到视图层(异步),同时改变对应的 this.data 的值(同步)。其参数说明见表 3-18。

表 3-18　setData()函数参数说明

字　段	类　型	必　填	描　述	最低版本
data	Object	是	这次要改变的数据	
callback	Function	否	setData()引起的界面更新渲染完毕后的回调函数	1.5.0

其中,Object 以 key : value 的形式表示,将 this.data 中的 key 对应的值改变成 value。key 可以以数据路径的形式给出,支持改变数组中的某一项或对象的某个属性,如 array[2].message,a.b.c.d,并且不需要在 this.data 中预先定义。

使用 setData 函数需要注意以下几点。

(1) 直接修改 this.data 而不调用 this.setData 是无法改变页面的状态的,还会造成数据不一致;

(2) 仅支持设置可 JSON 化的数据;

(3) 单次设置的数据不能超过 1024kB,请尽量避免一次设置过多的数据;

(4) 请不要把 data 中任何一项的 value 设为 undefined,否则这一项将不被设置并可能遗留一些潜在问题。

3.4.3 模块化

在 JavaScript 文件中声明的变量和函数只在该文件中有效，不同的文件中可以声明相同名字的变量和函数，不会互相影响。

通过全局函数 getApp()可以获取全局的应用实例，如果需要全局的数据可以在 App()中设置，例如：

```
// app.js
App({
  globalData: 1
})

// a.js
// localValue 只能在 a.js 这个文件中使用
const localValue = 'a'
//获取应用实例
const app = getApp()
//获取全局变量并更改其值
app.globalData++

// b.js
//在 b.js 中再次定义 localValue 不会对 a.js 文件中的 localValue 值造成影响
const localValue = 'b'
//如果 b.js 文件在 a.js 文件后面运行，那么此时的 globalData 的值为 2
console.log(getApp().globalData)
```

小程序也支持将一些公共的代码抽离成为一个单独的 js 文件，使其成为一个模块。模块只有通过 module.exports 或者 exports 才能对外暴露接口。例如，有一个公共的模块 common.js：

```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}

module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

在需要使用这些模块的文件中，使用 require(path) 将公共代码引入，并调用已经暴露出来的接口。

```
const common = require('common.js')
Page({
  helloMINA() {
    common.sayHello('MINA')
  },
```

```
  goodbyeMINA() {
    common.sayGoodbye('MINA')
  }
})
```


需要注意的是，exports 是 module.exports 的一个引用，在模块中随意更改 exports 的指向会造成未知的错误，所以更推荐采用 module.exports 暴露模块接口。同时，require 暂时不支持绝对路径。

3.4.4 接口

小程序开发框架提供丰富的微信原生 API，可以方便地调用微信提供的功能，如获取设备信息、本地存储、分享转发等。

通常，小程序的 API 有事件监听 API、同步 API 和异步 API 这三种类型。

一般来说，以 on 开头的 API 用来监听某个事件是否触发，如 wx.onSocketOpen（监听 WebSocket 连接打开事件），wx.onCompassChange（监听罗盘数据变化事件）等。这类 API 接受一个回调函数作为参数，当事件触发时会调用这个回调函数，并将相关数据以参数形式传入。一个简单的示例如下：

```
wx.onCompassChange(function (res) {
  console.log(res.direction)                    //输出当前设备面对的方向度数
})
```

一般来说，以 Sync 结尾的 API 都是同步 API，如 wx.setStorageSync，wx.getSystemInfoSync 等。此外，也有一些其他的同步 API，如 wx.createWorker，wx.getBackgroundAudioManager 等，具体的见后文对应 API 详细说明。同步 API 的执行结果可以通过函数返回值直接获取，如果执行出错会抛出异常。一个简单的代码示例如下。

```
try {
  wx.setStorageSync('key', 'value')          //设置本地数据缓存
} catch (e) {
  console.error(e)
}
```

除了以上两种外，大多数 API 都是异步 API，如 wx.request，wx.login 等。这类 API 通常都接受一个 Object 类型的参数，这个参数都支持按需指定不同的字段来接收接口调用结果。其参数说明见表 3-19。

表 3-19 异步 API 的参数说明

参 数 名	类 型	必 填	说 明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）
其他	Any	-	接口定义的其他参数

其中，success，fail 和 complete 这三个回调函数在调用时会传入一个 Object 参数，包含的字段说明见表 3-20。

表 3-20　回调函数参数说明

属　性	类　型	说　明
errMsg	String	错误信息，如果调用成功返回 ${apiName}:ok
errCode	Number	错误码，仅部分 API 支持，成功时为 0
其他	Any	接口返回的其他数据

异步 API 的执行结果需要通过 Object 类型的参数中传入的对应回调函数获取。部分异步 API 也会有返回值，可以用来实现更丰富的功能，如 wx.request，wx.connectSockets 等。一个使用异步 API 的简单示例如下。

```
wx.login({
  success(res) {
    console.log(res.code)
  }
})
```

3.5　视　图　层

小程序框架的视图层由 WXML（WeiXin Markup Language，微信标记语言）与 WXSS（WeiXin Style Sheet，微信样式表）编写，由组件来进行展示。视图层负责将逻辑层的数据显示在页面上，同时将视图层的事件发送给逻辑层。WXML 用来描述页面的结构，WXSS 用于描述页面的样式，组件是视图的基本组成单元。这三者的关系可以类比为 HTML，CSS 与 HTML 里面各种标签的关系。除了这三者之外，还有一套用于小程序的脚本语言——WXS（WeiXin Script）。WXS 和 WXML 结合起来，可以构建出页面结构。

3.5.1　WXML

【任务要求】

新建一个页面，在页面上显示九九乘法表。要求每次只显示一组数据，每单击一次按钮，加载下一组数据，直到 9×9＝81。也就是先显示 1×1＝1，1×2＝2，…，1×9＝9，单击按钮后，再显示 2×2＝4，2×3＝6，…，2×9＝18。示例效果如图 3-16 所示。

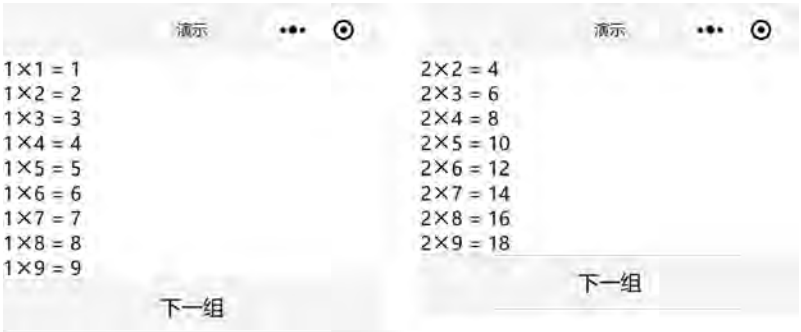

图 3-16　显示第一组（左）和单击按钮后显示第二组（右）

【任务分析】

本次任务针对 WXML 的相关功能进行设计，包含数据绑定、列表渲染(可以类比 for 循环)、条件渲染(可以类比 if 判断)和事件处理等知识点。通过对这个任务的练习和讲解，可以了解 WXML 的绝大部分功能和使用方法。

【任务操作】

(1) 打开示例项目，在 app.json 文件的 pages 数组中，新增第一项"pages/Chapter_3/WXML/WXML"。保存文件，编译项目，让开发者工具自动生成 WXML 页面所需的文件。

(2) 打开 pages/Chapter_3/WXML 目录下的 WXML.wxml 文件，将其中的内容替换为以下代码。

```
<!-- pages/Chapter_3/WXML/WXML.wxml -->
<view wx:for="{{array}}" wx:for-item="i">
  <view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
    </view>
  </view>
</view>
<button bindtap="addNewMultiplier">下一组</button>
```

(3) 打开该目录下的 WXML.js 文件，将里面的内容修改为如下代码。

```
// pages/Chapter_3/WXML/WXML.js
Page({
  data: {
    array:[1]
  },
  addNewMultiplier:function(){
    this.setData({
      array:[this.data.array[this.data.array.length-1]+1]
    })
  }
})
```

(4) 保存所有文件，编译运行。在模拟器中单击“下一组”按钮，观察运行效果。

【相关知识】

WXML 是框架设计的一套标签语言，结合基础组件、事件系统，可以构建出页面的结构。它包含数据绑定、列表渲染、条件渲染、模板、事件和引用这几个部分的功能。

1. 数据绑定

WXML 中的动态数据，均来自于对应 Page()函数的 data 对象。如何将 data 中的数据送到前端页面中去显示，这就涉及数据绑定的问题。

1) 简单绑定

数据绑定使用 Mustache 语法(双大括号)将变量括起来，可以作用于以下四种情况。

(1) 内容。直接在页面上显示数据内容，一个简单的示例如下。

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

```
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

将变量 message 用{{ }}括起来，即表示需要使用 data 中的数据来显示。该示例会在页面上显示“Hello MINA1”字样。

（2）组件属性。用后端变量来设置前端部分组件的属性。注意由双大括号括起来的变量需要在属性的双引号内。一个简单的示例如下。

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

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

该示例表示为 view 标签新增一个值为“item-0”的 id 属性。

（3）控制属性。用后端变量来控制前端组件的显示效果。由双大括号括起来的变量需要在属性的双引号内。一个简单的示例如下。

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

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

该示例表示判断的逻辑条件成立，“Hello MINA”的字样会被渲染显示到屏幕上。如果 condition 的值为 false，则页面上什么也不会显示。有关 wx:if 的用法可以参见后文条件渲染部分内容。

（4）关键字。主要用于逻辑判断。具体指“true”和“false”这两个关键字，分别是 boolean 类型的 true 和 false，表示真和假。一个简单的示例如下。

```
<view>勾选框不被选中</view>
<checkbox checked = "{{false}}"></checkbox>
<view>勾选框被选中</view>
<checkbox checked = "{{true}}"></checkbox>
```

该示例表示复选框处于选中和未选中的两种状态。显示的效果如图 3-17 所示。

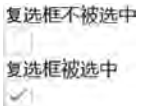

图 3-17　在 WXML 中设置关键字控制复选框状态

需要特别注意的是，设置复选框未被选中时不能直接写 checked＝"false"，因为这样的方式会将“false”当作字符串看待，转成 boolean 类型后代表真值。

2）运算

可以在{{ }}内进行简单的运算，支持如下几种方式。

（1）三元运算。可以在双大括号内进行三元运算，一个简单的示例如下。

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

该示例表示会根据条件表达式 flag 的情况来决定 hidden 属性的值是 true 还是 false，进而决定是否要在页面上显示“Hidden”字符串。

（2）算术运算。在双大括号内，可以进行基本的算术运算，会直接显示运算后的结果。一个简单的示例如下。

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

Page({
  data: {
    a: 1,
    b: 2,
    c: 3
  }
})
```

该示例会在页面上显示“3 + 3 + d”。其中，第一个 3 来自 a+b 的运算结果，第二个 3 来自变量 c 的值，d 因为没有被包含在双大括号内，作为一个字符原样输出。

（3）逻辑判断。可以在双大括号内进行逻辑运算，返回 boolean 类型的 true 或者 false，可以用于某些属性的控制。一个简单的示例如下。

```
<view wx:if="{{length > 5}}">{{length}}</view>
```

该示例表示，如果变量 length 的值大于 5，则显示 length 的值，否则不显示。

（4）字符串运算。可以在双大括号内做字符串的拼接运算。一个简单的示例如下。

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

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

该示例会在页面上显示出拼接好的字符串“hello MINA”。

（5）数据路径运算。对于数组和 JSON 对象类型的数据，在双大括号内也可以通过索引的方式取其值。一个简单的示例如下。

```
<view>{{object.key}} {{array[0]}}</view>

Page({
  data: {
    object: {
      key: 'Hello '
```

```
    },
    array: ['MINA','!']
  }
})
```

该示例最终会在页面上显示“Hello MINA”。因为{{array[0]}}是取 array 这个数组的第一个元素,因此“!”并不会被输出。

3) 组合

可以在双大括号内直接进行组合,构成新的数组或者对象。

(1) 数组。可以将 data 中的数据在 WXML 中组合成为一个新的数组。一个简单的示例如下。

```
<view wx:for="{{[zero, 1, 2, 3, 4]}}">{{item}}</view>

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

该示例表示将 data 中的变量 zero 加到 WXML 的数组中去,组成数组[0, 1, 2, 3, 4],最后在页面上输出 0,1,2,3,4。有关示例里面用到的 wx:for 的用法,可以参考后文列表渲染的相关内容。

(2) 对象。在双大括号里面,可以对对象进行组合、展开等操作。几个简单的示例如下。

```
<template is="objectCombine" data="{{for: a, bar: b}}"></template>

Page({
  data: {
    a: 1,
    b: 2
  }
})
```

最终组合成的对象是{for: 1, bar: 2}。

也可以使用扩展运算符“...”来将一个对象展开:

```
<template is="objectCombine" data="{{...obj1, ...obj2, e: 5}}"></template>

Page({
  data: {
    obj1: {
      a: 1,
      b: 2
    },
    obj2: {
      c: 3,
      d: 4
```

```
    }
  }
})
```


最终组合成的对象是{a：1，b：2，c：3，d：4，e：5}。

如果对象的 key 和 value 相同，也可以间接地表达。例如：

```
<template is="objectCombine" data="{{foo, bar}}"></template>

Page({
  data: {
    foo: 'my-foo',
    bar: 'my-bar'
  }
})
```

最终组合成的对象是{foo：'my-foo'，bar：'my-bar'}。

上述几种情况可以随意组合，但是如有存在变量名相同的情况，后面的会覆盖前面，例如：

```
<template is="objectCombine" data="{{...obj1, ...obj2, a, c: 6}}"></template>

Page({
  data: {
    obj1: {
      a: 1,
      b: 2
    },
    obj2: {
      b: 3,
      c: 4
    },
    a: 5
  }
})
```

最终组合成的对象是{a：5，b：3，c：6}。

注意：如果双大括号和引号之间有空格，则表达式最终将会被解析成为字符串。例如：

```
<view wx:for="{{[1,2,3]}} ">
  {{item}}
</view>
```

等同于：

```
<view wx:for="{{[1,2,3] + ''}}">
  {{item}}
</view>
```

2. 列表渲染

1）wx:for

在组件上使用 wx:for 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该

组件。默认情况下,数组当前项的下标变量名为 index,数组当前项的变量名为 item。一个简单的示例如下。

```
<view wx:for="{{array}}">
  {{index}}: {{item.message}}
</view>

Page({
  data: {
    array: [{
      message: 'foo',
    }, {
      message: 'bar'
    }]
  }
})
```

该示例最终会在页面上输出 0: foo 和 1: bar 两行结果。

当然,也可以使用 wx:for-item 来指定数组当前元素的变量名,使用 wx:for-index 来指定数组当前下标的变量名。例如,前面的例子也可以这样写:

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName.message}}
</view>
```

也可以将 wx:for 用在< block />标签上,以渲染一个包含多节点的结构块。例如:

```
<block wx:for="{{[1, 2, 3]}}">
  <view>{{index}}:</view>
  <view>{{item}}</view>
</block>
```

< block/>标签本身不含有任何的默认样式,也不会在页面上有具体的展现,仅仅是作为设计 WXML 页面的结构而出现。这个例子最终会渲染出 6 个 view 组件,每两个 view 组件为一组,分别输出数组里每一项的序号和值。

如果 wx:for 的值为一个字符串,那么该字符串将被解析成为字符串数组。例如:

```
<view wx:for="array">
  {{item}}
</view>
```

等同于:

```
<view wx:for="{{['a','r','r','a','y']}}">
  {{item}}
</view>
```

2) wx:key

如果列表中项目的位置会动态改变或者有新的项目添加到列表中,与此同时还希望列表中已有的项目保持自己的特征和状态(如 < input />中的输入内容,< switch />的选中状

态),这个时候需要使用 wx:key 来指定列表中项目的唯一的标识符。有关< input />和< switch />的介绍可以参见 7.4 节和 7.10 节的内容。

wx:key 的值以如下两种形式提供。

(1) 字符串。代表在 wx:for 循环的数组中某一项的某个属性,该属性的值需要是列表中唯一的字符串或数字,且不能动态改变。

(2) 保留关键字 * this。代表在 wx:for 循环中的某一项本身,这种表示需要这一项本身是一个唯一的字符串或者数字。

当数据改变触发渲染层重新渲染的时候,框架会校正带有 key 的组件,让它们被重新排序,而不是重新创建,以确保使组件保持自身的状态,并且提高列表渲染时的效率。一个使用 wx:key 的示例如下。

```
<!-- pages/Chapter_3/WXKEY/WXKEY.wxml -->
<switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;">
  {{item.id}}
</switch>
<button bindtap="switch">重新排序</button>
<button bindtap="addToFront">在列表顶端新增一项</button>
<switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;">
  {{item}}
</switch>
<button bindtap="addNumberToFront">在列表顶端新增一项</button>

// pages/Chapter_3/WXKEY/WXKEY.js
Page({
  data: {
    objectArray: [
      { id: 5, unique: 'unique_5' },
      { id: 4, unique: 'unique_4' },
      { id: 3, unique: 'unique_3' },
      { id: 2, unique: 'unique_2' },
      { id: 1, unique: 'unique_1' },
      { id: 0, unique: 'unique_0' },
    ],
    numberArray: [1, 2, 3, 4]
  },
  switch(e) {
    const length = this.data.objectArray.length
    for (let i = 0; i < length; ++i) {
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.objectArray[x]
      this.data.objectArray[x] = this.data.objectArray[y]
      this.data.objectArray[y] = temp
    }
    this.setData({
      objectArray: this.data.objectArray
    })
  },
```

```
  addToFront(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{ id: length, unique: 'unique_' + length }].concat(this.data.
objectArray)
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addNumberToFront(e) {
    this.data.numberArray = [this.data.numberArray.length + 1].concat(this.data.numberArray)
    this.setData({
      numberArray: this.data.numberArray
    })
  }
})
```

该示例会在页面上显示若干个开关组件，可以更改部分开关组件的状态（如图 3-18 所示），在单击“重新排序”或者是“在列表顶端新增一项”之后，原有的开关状态并不会被改变，而是会一直保持（如图 3-19 所示）。

图 3-18　设定开关状态

图 3-19　插入新的元素或者是重新排序后

一般情况下，如果不提供 wx:key，调试器会给出一个警告。如果自己明确知道该列表是静态的，或者不必关注其顺序，可以选择忽略。

3. 条件渲染

1）wx:if

wx:if 是一个控制属性，用于控制它所作用的标签是否要被渲染。使用的格式为：wx:if="{{condition}}"。condition 需要是一个可以转换为 Boolean 类型的值或者表达式。还

可以结合使用 wx:elif 和 wx:else 这两个控制属性组合成 if else 代码块。一个简单的示例如下。

```
<view wx:if="{{score>90}}">A</view>
<view wx:elif="{{score>70}}">B</view>
<view wx:else>C</view>
```

和前面的 wx:if 类似，如果需要一次性判断多个组件标签，可以使用一个 <block/> 标签将多个组件包装起来，并在上边使用 wx:if 控制属性。例如：

```
<block wx:if="{{true}}">
  <view>view1</view>
  <view>view2</view>
</block>
```

同样地，<block />仅仅是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

2）wx:if 对比 hidden

考虑到 wx:if 之中的模板也可能包含数据绑定，所以当 wx:if 的条件值切换时，框架会对 wx:if 包含的代码块进行销毁或者是重新进行局部渲染。同时 wx:if 也是惰性的。这意味着，如果初始渲染条件为 false，那么框架什么也不会做。框架只有在 wx:if 的条件第一次变成真的时候才开始渲染 wx:if 控制的代码块内容。

相比之下，hidden 就简单得多，组件始终会被渲染，只是简单地控制显示与隐藏。

一般来说，wx:if 有更高的切换消耗而 hidden 有更高的初始渲染消耗。因此，如果需要频繁切换，用 hidden 更好，如果在运行时条件不大可能改变则使用 wx:if 较好。

4. 事件

事件是视图层到逻辑层的通信方式，它可以将用户的行为反馈到逻辑层进行处理。事件一般绑定在组件上，当设定监听的事件被触发时，视图层会将携带了 id，dataset，touches 等信息的事件对象发送到逻辑层中，此时框架就会执行逻辑层中对应的事件处理函数，来响应用户的操作。

1）事件的使用方式

以 bindtap 这样的一个监听用户单击事件的使用方式举一个简单的例子。

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

在相应的 Page 定义中写上相应的事件处理函数，参数是 event。

```
Page({
  tapEvent(event) {
    console.log(event)
  }
})
```

在调试器的 Console 面板中可以看到输出的信息大致如下。

```
{
  "type": "tap",
  "timeStamp": 895,
  "target": {
```

```
      "id": "tapTest",
      "dataset": {
        "hi": "WeChat"
      }
    },
    "currentTarget": {
      "id": "tapTest",
      "dataset": {
        "hi": "WeChat"
      }
    },
    "detail": {
      "x": 53,
      "y": 14
    },
    "touches": [
      {
        "identifier": 0,
        "pageX": 53,
        "pageY": 14,
        "clientX": 53,
        "clientY": 14
      }
    ],
    "changedTouches": [
      {
        "identifier": 0,
        "pageX": 53,
        "pageY": 14,
        "clientX": 53,
        "clientY": 14
      }
    ]
  }
```


在这个例子中，为 view 组件绑定了一个值为“tapEvent”，名为“bindtap”的属性。它表示需要监听用户的单击(tap)事件，该事件由逻辑层的 tapEvent 函数处理。同时还在 view 组件里设置了事件需要传递的数据，由 data-hi 属性给出，表示需要传递的数据名为“hi”，值为“WeChat”。在逻辑层接收到的事件对象 event 里，也找到了对应的数据。

2）事件的分类

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。一般来说，大多数组件的事件都是非冒泡事件。除了非冒泡事件，WXML 里的冒泡事件见表 3-21。

表 3-21　WXML 里的冒泡事件

类　　型	触 发 条 件	最低版本
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	1.5.0
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	
animationend	会在一个 WXSS animation 动画完成时触发	
touchforcechange	在支持 3D Touch 的 iPhone 设备，重按时会触发	1.9.90

3）事件的绑定和冒泡

事件绑定的写法同组件的属性写法一样，均以（key，value）键值对的形式。

key 以 bind 或 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，在非原生组件中，bind 和 catch 后可以紧跟一个冒号，其含义不变，如 bind:tap、catch:touchstart。使用 bind 绑定的事件不会阻止冒泡事件向上冒泡，而使用 catch 绑定的事件可以阻止冒泡事件向上冒泡。

value 是一个字符串，指的是对应的 Page 中定义的同名函数。如果没有在 Page 中定义该函数，在事件被触发时，调试器会报错。

有关冒泡和阻止冒泡，可以参考下面这个简单的例子。

```
<view id="outer" bindtap="handleTap1">
  outer view
  <view id="middle" catchtap="handleTap2">
    middle view
    <view id="inner" bindtap="handleTap3">
      inner view
    </view>
  </view>
</view>
```

在上面的这个例子中，单击 inner view 会先后调用 handleTap3 和 handleTap2（因为 tap 事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，不再向父节点传递），单击 middle view 只会触发 handleTap2，单击 outer view 会触发 handleTap1。

4）事件的捕获阶段

自基础库版本 1.5.0 起，触摸类事件支持捕获阶段，也可以理解为监听事件的发生。捕获阶段位于冒泡阶段之前，且在捕获阶段中，事件到达节点的顺序与冒泡阶段恰好相反。需

要在捕获阶段监听事件时，可以采用 capture-bind、capture-catch 关键字。其中，capture-catch 将中断捕获阶段和取消冒泡阶段。

一个使用事件捕获阶段的简单示例如下。

```
<view id="outer" bindtouchstart="handleTap1" capture-bindtouchstart="handleTap2">
  outer view
  <view id="inner" bindtouchstart="handleTap3" capture-bindtouchstart="handleTap4">
    inner view
  </view>
</view>
```

在该示例中，如果单击了 inner view，则会先后触发 handleTap2、handleTap4、handleTap3、handleTap1 这四个事件。

如果将上面例子中的第一个 capture-bindtouchstart 改为 capture-catchtouchstart，由于 capture-catch 会中断后面的事件捕获以及冒泡，因此在单击 inner view 后将只会触发 handleTap2。

5）事件对象

如无特殊说明，当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。基础事件（BaseEvent）对象的属性说明见表 3-22。

表 3-22　基础事件对象属性说明

属　性	类　型	说　明
type	String	事件类型
timeStamp	Integer	事件生成时的时间戳，记录的是页面打开到触发事件所经过的毫秒数
target	Object	触发事件的源组件的一些属性值集合
currentTarget	Object	事件绑定的当前组件的一些属性值集合

由基础事件（BaseEvent）对象派生出自定义事件对象（CustomEvent）和触摸事件对象（TouchEvent），这两个对象除了具有基础事件对象的所有属性外，还具有各自的一些属性，分别见表 3-23 和表 3-24。

表 3-23　自定义事件对象属性说明

属　性	类　型	说　明
detail	Object	额外的信息，由各个组件自己定义。比如表单提交的数据、媒体的错误信息等

表 3-24　触摸事件对象属性说明

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

对基础事件对象的 target 属性所包含的属性值集合的说明见表 3-25。

表 3-25　target 所包含的属性值

属　　性	类　　型	说　　明
id	String	事件源组件的 id
tagName	String	当前组件的类型
dataset	Object	事件源组件上由 data-开头的自定义属性组成的集合

对基础事件对象的 currentTarget 属性所包含的属性值集合的说明见表 3-26。

表 3-26　currentTarget 所包含的属性值

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

有关 target 和 currentTarget 的使用，可以参考下面这个简单的例子。

```
<view id = "outer" bindtap = "handleTap1">
  outer view
  <view id = "middle" catchtap = "handleTap2">
    middle view
    <view id = "inner" bindtap = "handleTap3">
      inner view
    </view>
  </view>
</view>
```

在这个例子中，我们单击 inner view，会依次触发 handleTap3 和 handleTap2 这两个处理函数。其中，handleTap3 收到的事件对象的 target 和 currentTarget 都是 inner，而 handleTap2 收到的事件对象的 target 是 inner，currentTarget 则是 middle。

在 target 和 currentTarget 属性值的集合中，dataset 属性表示组件中自定义数据的集合。在组件中定义的数据，会通过事件传递给逻辑层。正如上面例子中，在 view 组件中自定义了 data-hi＝"WeChat"的属性，那么在 dataset 这个属性里，就可以找到一个 key 为 hi，value 为“WeChat”的值。在组件中自定义数据的方式是：以“data-”开头，多个单词由连字符“-”连接，不能有大写，如果有大写，则会自动转成小写。例如，data-element-type，最终在 event.currentTarget.dataset 中会将连字符转成驼峰 elementType；data-elementType，最终在 event.currentTarget.dataset 中会转换成 elementtype。

在触摸事件对象中，touches 属性是一个数组，数组的每一项是一个 Touch 对象，表示当前停留在屏幕上的触摸点。Touch 对象的属性说明见表 3-27。

表 3-27　Touch 对象属性说明

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
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

触摸事件对象的 changedTouches 属性数据格式同 touches，表示有变化的触摸点。如从无变有（touchstart），位置变化（touchmove），从有变无（touchend，touchcancel）。

需要单独说明的是，在 Canvas 中，触摸事件对象的 touches 属性里面携带的则是 CanvasTouch 对象。CanvasTouch 对象的属性说明见表 3-28。

表 3-28　CanvasTouch 对象属性说明

属　性	类　型	说　明
identifier	Number	触摸点的标识符
x，y	Number	距离 Canvas 左上角的距离，Canvas 的左上角为原点，横向为 X 轴，纵向为 Y 轴

5. 模板

WXML 提供模板（template）功能。开发者可以在模板中定义代码片段，然后在不同的地方调用。方便在小程序页面的部分固定结构中使用。

要使用模板，首先需要定义模板。定义模板需要使用< template/>标签，然后在标签内声明 name 属性。模板的代码写在< template/>标签内。一个简单的示例如下。

```
<template name="msgItem">
  <view>
    <text>{{index}}: {{msg}}</text>
    <text>Time: {{time}}</text>
  </view>
</template>
```

定义好之后，在使用的时候依然是用< template />标签，然后通过指定其 is 属性来确定使用哪个模板。在上个例子中，模板还需要一些数据，我们需要指定< template />标签的 data 属性来将数据传递给模板。使用示例如下。

```
<template is="msgItem" data="{{...item}}" />

Page({
  data: {
    item: {
      index: 0,
      msg: 'this is a template',
      time: '2019-01-01'
    }
  }
})
```

is 属性可以使用双大括号语法，从而动态决定具体需要渲染哪个模板。一个简单的示例如下。

```
<template name="odd">
  <view>odd</view>
</template>
<template name="even">
  <view>even</view>
</template>

<block wx:for="{{[1, 2, 3, 4, 5]}}">
  <template is="{{item % 2 == 0 ? 'even' : 'odd'}}" />
</block>
```

在上面这个例子中，会交替使用 odd 和 even 这两个模板。

6. 引用

WXML 提供两种文件引用方式：import 和 include。

import 可以在该文件中使用目标文件定义的模板。例如，在 item.wxml 中定义了一个叫作 item 的模板：

```
<!-- item.wxml -->
<template name="item">
  <text>{{text}}</text>
</template>
```

在 index.wxml 中引用了 item.wxml，就可以使用 item 模板。

```
<import src="item.wxml" />
<template is="item" data="{{text: 'forbar'}}" />
```

import 有作用域的概念，即只会导入目标文件中定义的模板，而不会导入目标文件导入的模板。

例如：C import B，B import A，在 C 中可以使用 B 定义的 template，在 B 中可以使用 A 定义的 template，但是 C 不能使用 A 定义的 template。

include 可以将目标文件除了 <template /> <wxs /> 外的整个代码引入，相当于复制到 include 位置。一个简单的示例如下。

```
<!-- index.wxml -->
<include src="header.wxml" />
<view>body</view>
<include src="footer.wxml" />

<!-- header.wxml -->
<view>header</view>

<!-- footer.wxml -->
<view>footer</view>
```

最终形成的 index.wxml 文件内容如下。

```
<!-- index.wxml -->
<view>header</view>
<view>body</view>
<view>footer</view>
```

3.5.2 WXSS

【任务要求】

在 pages/Chapter_3 目录下新建一个 WXSS 目录，在 WXSS 目录下新建一个名为 WXSS 的页面。页面上横向放置绿、蓝、灰三个色块，并在上面分别标记 A，B，C。在单击该区域时，三个色块的颜色能在绿、蓝、灰之间随机切换。效果如图 3-20 所示。

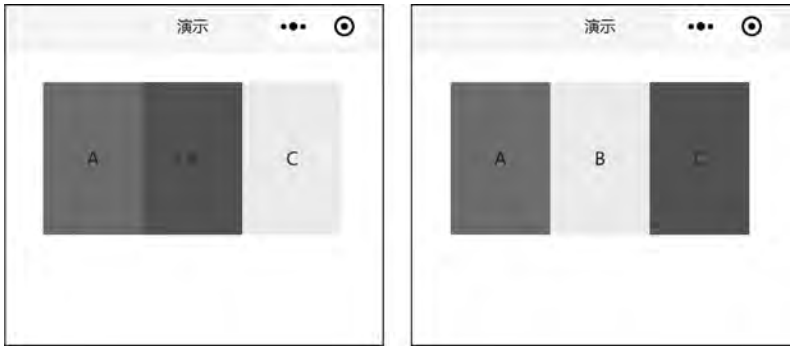

图 3-20　三个色块(左)和单击后随机切换颜色效果(右)

【任务分析】

本次任务主要是练习微信的样式语言 WXSS 的使用，包括设定颜色，设定大小，控制位置，如何使用选择器等。其中还涉及样式的动态渲染，需要使用到事件的知识。WXSS 的样式属性大部分都可以和 CSS 类比。本次任务中所涉及的具体的样式属性可以暂时不用太关注，重点学习选择器的使用。

【任务操作】

(1) 打开示例项目，在 app.json 中新增第一项"pages/Chapter_3/WXSS/WXSS"。保存并编译项目，让开发者工具自动生成需要的文件。

(2) 打开 pages/Chapter_3/WXSS 目录下的 WXSS.wxml 文件，将其中的内容替换为以下代码。

```
<!-- pages/Chapter_3/WXSS/WXSS.wxml -->
<viewid="container">
  <view class="flex-wrp" bindtap="changeColor">
    <view class="flex-item demo-text-1" style="background-color:{{colorArray[0]}}">
</view>
    <view class="flex-item demo-text-2" style="background-color:{{colorArray[1]}}">
</view>
    <view class="flex-item demo-text-3" style="background-color:{{colorArray[2]}}">
</view>
  </view>
</view>
```

(3) 打开该目录下的 WXSS.js 文件,为色块设定初始颜色和添加改变颜色事件的处理函数。

```
// pages/Chapter_3/WXSS/WXSS.js
Page({
  data: {
    colorArray: ["#1AAD19", "#2782D7", "#F1F1F1"],
  },
  changeColor:function() {
    const length = this.data.colorArray.length
    for (let i = 0; i < length; ++i) {
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.colorArray[x]
      this.data.colorArray[x] = this.data.colorArray[y]
      this.data.colorArray[y] = temp
    }
    this.setData({
      colorArray: this.data.colorArray
    })
  }
})
```

(4) 打开该目录下的 WXSS.wxss 文件,为三个色块设定需要的样式效果。

```
/* pages/Chapter_3/WXSS/WXSS.wxss */
#container{
  box-sizing: border-box;
  padding: 0 80rpx;
}
.flex-wrp{
  margin-top: 60rpx;
  display:flex;
  flex-direction:row;
}
.flex-item{
  width: 200rpx;
  height: 300rpx;
  font-size: 26rpx;
  color: #353535;
}
.demo-text-1{
  position: relative;
  align-items: center;
  justify-content: center;
  font-size: 36rpx;
}
.demo-text-1:before{
  content: 'A';
```

```
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
.demo-text-2{
  position: relative;
  align-items: center;
  justify-content: center;
  font-size: 36rpx;
}
.demo-text-2:before{
  content: 'B';
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
.demo-text-3{
  position: relative;
  align-items: center;
  justify-content: center;
  font-size: 36rpx;
}
.demo-text-3:before{
  content: 'C';
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

(5) 保存所有文件,编译项目。在模拟器中单击三个色块所在的区域,观察是否随着单击,三个色块的颜色都在蓝、绿、灰之间改变。

【相关知识】

WXSS是一套样式语言,用于描述WXML的组件样式。它被用来决定WXML的组件应该怎么显示。和CSS相比,WXSS具有CSS的大部分特性。同时为了更适合开发微信小程序,在尺寸单位和样式导入这两方面,WXSS对CSS进行了扩充以及修改。

1. 尺寸单位

WXSS使用了rpx(responsive pixel,响应像素)作为尺寸单位。它规定屏幕宽为750rpx,然后可以根据屏幕宽度进行自适应。如在iPhone 6上,屏幕宽度为375px,共有750个物理像素,则750rpx=375px=750物理像素,即1rpx=0.5px=1物理像素。一般来说,建议小程序的设计师以iPhone 6作为视觉稿的标准。需要注意的是,在较小的屏幕上不可避免地会有一些毛刺,请在开发时尽量避免这种情况。部分设备的尺寸大小换算见表3-29。

表 3-29　rpx 和 px 在部分设备上的换算关系

设　　备	rpx 换算为 px(屏幕宽度/750)	px 换算为 rpx(750/屏幕宽度)
iPhone 5	1rpx = 0.42px	1px = 2.34rpx
iPhone 6	1rpx = 0.5px	1px = 2rpx
iPhone 6 Plus	1rpx = 0.552px	1px = 1.81rpx

2. 样式导入

使用@import 语句可以导入外联样式表。@import 后跟需要导入的外联样式表的相对路径,用“;”表示语句结束。一个简单的示例如下。

```
/** common.wxss **/
.small-p {
  padding:5px;
}

/** app.wxss **/
@import "common.wxss";
.middle-p {
  padding:15px;
}
```

则在 app.wxss 中,包含对 small-p 和 middle-p 这两个样式类的定义。

3. 内联样式

框架组件上支持使用 style、class 属性来控制组件的样式。

style:用于接收动态样式,在运行时会进行解析。一个简单的示例如下。

```
<view style="color:{{color}};" />
```

该示例表示 view 的颜色由 color 这个变量动态定义。

class:用于指定样式规则。其值是样式规则中类选择器名(样式类名)的集合。一般将静态样式写到对应样式类名的定义中。多个样式类名之间用空格分隔。一个简单的示例如下。

```
<!--example.wxml-->
<view class="normal_view content_view" />

/** example.wxss **/
.normal_view{
  height: 300rpx;
  width: 200rpx;
}
.content _view{
  font-size: 26rpx;
  color: #353535;
}
```

该示例表示为 view 组件设置了宽为 200rpx,高为 300rpx,里面的内容字体大小为 26rpx,颜色为#353535。

4. 选择器

和 CSS 一样，WXSS 也需要使用选择器来决定样式的作用对象。WXSS 目前支持的选择器见表 3-30。

表 3-30　WXSS 支持的选择器

选　择　器	样　　例	样 例 描 述
.class	.intro	选择所有拥有 class="intro"的组件
#id	#firstname	选择拥有 id="firstname"的组件
element	view	选择所有 view 组件
element, element	view,checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
::after	view::after	在 view 组件后边插入内容
::before	view::before	在 view 组件前边插入内容

5. 全局样式与局部样式

定义在 app.wxss 中的样式为全局样式，作用于每一个页面。在每个页面自己的 wxss 文件中定义的样式为局部样式，只作用在对应的页面，并且会覆盖 app.wxss 中相同的选择器所定义的样式。

3.5.3　基础组件

组件是视图层的基本组成单元，自带一些功能与微信风格一致的样式。一个组件通常包括开始标签和结束标签，以及用来修饰这个组件的属性。组件的内容被包含在开始和结束这两个标签之内。组件的使用方式示例如下。

```
<tagname property="value">
  Content goes here ...
</tagname>
```

框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行快速开发。

组件属性的数据类型说明见表 3-31。

表 3-31　组件属性的数据类型

类　　型	描　　述	注　　解
Boolean	布尔值	组件写上该属性，不管是什么值都被当作 true；只有组件上没有该属性时，属性值才为 false。如果属性值为变量，变量的值会被转换为 Boolean 类型
Number	数字	1,2.5
String	字符串	"string"
Array	数组	[1,"string"]
Object	对象	{key:value}
EventHandler	事件处理函数名	"handlerName"是 Page 中定义的事件处理函数名
Any	任意属性	

所有组件都具有的公共属性见表 3-32。

表 3-32　组件的公共属性

属　性　名	类　　型	描　　述	注　　解
id	String	组件的唯一标识	保持整个页面唯一
class	String	组件的样式类	在对应的 WXSS 中定义的样式类
style	String	组件的内联样式	可以动态设置的内联样式
hidden	Boolean	组件是否显示	所有组件默认显示
data-*	Any	自定义属性	组件上的事件被触发时，会把数据发送给事件处理函数
bind* / catch*	EventHandler	组件的事件	详见 3.5.1 节中对事件的介绍

组件各自的特殊属性，详见后文对于每个组件对应的介绍。

练　习　题

1. 小程序的运行状态有哪几种？在切换时分别会调用的生命周期函数是什么？

2. 页面 B 通过 switchTab 的方式打开了页面 A，页面 A 通过 redirectTo 打开了页面 C，页面 C 再通过 navigateTo 打开了页面 B。请按顺序描述在这个过程中各个页面的生命周期函数的调用情况。

3. 简要描述 MINA 框架的主要功能。

4. 注册程序和注册页面的 App()函数以及 Page()函数分别可以包含哪些参数？每个参数的大致作用是什么？

5. WXML 里面的数据绑定有几种使用方式？

6. 冒泡事件和非冒泡事件的区别在什么地方？

7. 在事件发生时，如何向事件处理函数发送数据？

8. 在示例项目的 pages/Chapter_3 目录下新建一个 exercise_01 文件夹，然后在该目录下新建一个名为“01”的页面。要求：如果当前页面不是从转发进入的，则在页面上显示文字“请将该页面转发给朋友。”，并且设置转发的标题为“您的好友{{nickname}}觉得这个小程序不错，快来看看吧！”。其中，{{nickname}}为占位符，需要替换成使用当前小程序的用户的微信昵称。如果当前页面是由转发进入的，则在页面上显示“您的好友{{nickname}}邀请您访问了这个页面。”，其中，{{nickname}}为占位符，需要替换成转发者的微信昵称。

9. 在示例项目的 pages/Chapter_3/exercise 目录下新建名为“02”的页面。在页面上用一个数字显示该数字被单击的次数。意即，每单击一次该数字，该数字便会加 1。