版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。

◎从零开始打造自己的微信小程序，创建一种全新的连接用户与服务的方式
◎详细介绍使用JavaScript、WXML与Flexbox综合开发微信小程序
◎由浅入深，全面详细地为初学者提供开发指导，为开发者答疑解惑
◎让初学者读懂每一行代码

Broadview
www.broadview.com.cn

实战微信小程序

JavaScript、WXML与Flexbox综合开发

荣蓉 穆心驰 何金刚 于连林 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

实战微信小程序

JavaScript、WXML与Flexbox综合开发

荣蓉 穆心驰 何金刚 于连林 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

本书从零开始介绍微信小程序的开发，内容从逻辑上可以分为 4 部分。第一部分（第 1~ 第 2 章）主要介绍什么是微信小程序、微信小程序的发展前景等，让初学者快速了解微信小程序。第二部分（第 3~ 第 4 章）主要介绍 JavaScript 和 WXML，让零基础或者没有学习过 JavaScript 的读者在学习微信小程序的时候不会有语言障碍，能够读懂每一行代码，为学习微信小程序开发打下基础。第三部分（第 5~ 第 9 章）是本书的精髓，重点介绍了微信小程序的布局、组件、API 接口等，详细地阐述了微信小程序的所有功能实现。希望读者能够通过这部分内容的学习，熟练掌握微信小程序开发。第四部分（第 10~ 第 11 章）通过详细的案例分析，可以使读者从项目开始到最后发布有一个系统的学习过程。

本书内容深入浅出，适合零基础、对微信小程序开发有兴趣的人员，移动平台开发人员，JavaScript 开发人员，有编程经验想转行做微信小程序的开发人员及计算机专业的学生。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

实战微信小程序：JavaScript、WXML 与 Flexbox 综合开发 / 荣蓉等著.—北京：电子工业出版社，2017.7

ISBN 978-7-121-31314-1

I. ①实…II. ①荣…III. ①移动终端 – 应用程序 – 程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字（2017）第 072697 号

责任编辑：王　静

印　　刷：北京天宇星印刷厂

装　　订：北京天宇星印刷厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱　　邮编：100036

开　　本：787×980　1/16　　印张：25.25　　字数：520 千字

版　　次：2017 年 7 月第 1 版

印　　次：2017 年 7 月第 1 次印刷

定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

推荐序

依稀记得是在2014年年初认识了于连林老师，那时候我还在中关村软件园负责黑马程序员的软件开发培训业务，当时负责Android课程的教学总监和我说："咱们团队新引入了一位老师，非常优秀，喜欢教育工作，并且技术非常好，相信这位老师的加盟会让咱们的Android后期项目课程上一个台阶。"当时我有些将信将疑，后来随着连林的课程研发、教学实施等工作的开展，我对连林的学术、人品以及对技术的执着研究精神，也有了更深的认识。在一起共事一年半左右的时间后，2015年年底连林由于家庭原因，不得不离开北京回天津发展，当时我听到这个消息后甚是遗憾。分手之际，还和连林约定，若是黑马程序员去天津开分公司，连林不管采用哪种形式，一定要在黑马程序员这个平台上继续分享更多的知识给同学们。

一晃两年过去了，2017年3月中旬连林给我发来消息，说他写了一本关于微信小程序的书，并发来书稿，请我为之作序。

荣幸之余，我也有些担心，因为"事非亲历不知难"，只有写过书的人才知道写书的不易，想在一本书中把一门技术讲清楚，无论是案例的选取，还是技术的逻辑组织，都是一个让人彻夜难眠的苦差事，这也就导致一本通俗、易读、实在的书，在市面上少之又少。而"微信小程序"作为最近刚刚兴起的热点，市面上还没有一套成熟的书籍成功案例，这就让一本具备这些特质的书更加难以出现。值得惊讶的是，连林写的这本书，恰好具备了这方面的特质，虽然没有接触过微信小程序的开发，我仅随意翻阅了几章便对微信小程序有了一个清晰的认知，甚至有想开发几个属于自己的小程序的冲动。

这本书里的每一个文字，都透露着连林希望把知识分享给读者的意愿。所以我也希望这本实在、易读、有血有肉的书，能满足连林的初心，帮助更多的人。也期待能有更多的小程序开发者从中受益。

方立勋
黑马程序员创始人
2017年4月7日夜

前言

移动应用开发的前景

随着科技的发展，手机已经成为每个人随时随地都要使用的“便携式电脑”。近年来，手机硬件配置越来越高，功能越来越丰富，系统越来越完善（并且一直被 Android 和 iOS 两大系统称霸），价格越来越平易近人，这些优势使得手机越来越普及，相应地，市场对移动应用开发的需求必然会越来越高。

本书写作目的

现在，市场上的应用基本可以分成三种开发方式：Android 原生开发、iOS 原生开发和跨平台开发。前两种开发方式都是原生开发，一般情况下会同时进行两个版本的原生开发，或者直接使用第三种——跨平台开发。近年来，跨平台开发陆续萌生出很多种框架，例如 PhoneGap、Titanium、React Native 等。开发者们也对这些框架有过一些争论，我们暂且不说这些框架孰好孰坏。

2016 年，“微信之父”张小龙时隔多年公开亮相并称微信正在研究一种新的形态，叫“微信小程序”，经过一年的研究测试，终于在 2017 年 1 月正式上线。微信小程序是一种全新的连接用户与服务的方式，它可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

微信小程序上线不到一天，便迅速引爆网络，一百多个微信小程序的名单出炉，各种互联网公司也争相投入开发，笔者从 2016 年起一直关注微信小程序的发展，也开发了一些应用。笔者希望可以把自己的开发经验分享给读者，也希望可以通过本书，由浅入深，为初学者提供开发指导，为开发者答疑解惑。同时希望借此书来认识更多志同道合的朋友。

如何阅读本书

本书在内容逻辑上可以分为 4 部分。

第一部分（第 1~ 第 2 章）主要介绍了什么是微信小程序、微信小程序的发展前景等，让初学者很快了解微信小程序。

第二部分（第 3~ 第 4 章）主要介绍了 JavaScript 和 WXML，可以让零基础或者没有学习过 JavaScript 的读者在学习微信小程序时不会有语言障碍，能够读懂每一行代码，为学习微信小程序开发打下基础。

第三部分（第 5~ 第 9 章）是本书的精髓，重点介绍了微信小程序的布局、组件、API 接口等，详细地阐述了微信小程序的所有功能实现。希望读者能够通过这部分内容的学习，熟练掌握微信小程序的开发。

第四部分（第 10~ 第 11 章）通过详细的案例分析，可以使读者从项目开始到最后发布有一个系统的学习过程，至此，读者完全有能力开发出自己的微信小程序。

读者对象

- 对微信小程序开发有兴趣的人员。
- 移动平台开发人员。
- JavaScript 开发人员。
- 有编程经验想转行做微信小程序的开发人员。
- 计算机专业的学生。

代码下载

本书配套源代码的下载地址：https://pan.baidu.com/s/1nvyjsQt；密码：wqtr，若下载有问题或有什么疑问，欢迎发邮件到：yll@520wcf.com 或 verobook@163.com。

致谢

感谢微信团队，创造了这一伟大的产品。

感谢电子工业出版社，审校此书，以及为本书能够快速出版而付出的巨大努力。

感谢方立勋老师，在百忙之中为本书题序，帮助我们完善本书。

感谢徐明华老师，在百忙之中给予指导和支持。

感谢工作和生活中帮助过我们的所有人，有你们的鼓励和支持才有本书的面世，谢谢你们。

欢迎访问作者的博客（http://520wcf.com）和公众号。

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在 下载资源 处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/31314

目录

第一部分　认识微信小程序

1　微信小程序介绍 1

1.1　微信小程序是什么 1

1.2　微信小程序的前景 2

1.3　微信小程序与订阅号、服务号的区别 3

1.4　如何创建一个微信小程序 4

1.4.1　成为微信小程序开发者 5

1.4.2　安装开发者编辑器 7

1.5　运行一个微信小程序 7

1.6　本章小结 9

2　体验微信小程序 10

2.1　界面与操作 10

2.2　编辑功能 11

2.3　调试功能 12

2.4　项目功能 16

2.5　常用快捷键 18

2.6　项目的目录与文件结构 18

2.7　本章小结 21

第二部分 读懂每一行代码

3 JavaScript 语法 22

3.1 JavaScript 简介 22

3.2 基础语法 24

3.2.1 语句和语句块 25

3.2.2 注释 26

3.2.3 变量 26

3.2.4 常量 27

3.2.5 数据类型 27

3.2.6 运算符 38

3.2.7 条件判断 42

3.2.8 循环语句 43

3.2.9 Map 和 Set 46

3.3 JavaScript 函数 48

3.3.1 函数定义和调用 48

3.3.2 变量作用域 53

3.3.3 方法 54

3.3.4 高阶函数 56

3.3.5 箭头函数 57

3.4 JavaScript 标准对象 58

3.4.1 Date 58

3.4.2 RegExp 59

3.4.3 JSON 60

3.4.4 Math 61

3.5 本章小结 62

4 熟练掌握 WXML 和 HTML 63

4.1 WXML 和 HTML 的差异 63

4.2 WXML 语法 65

4.2.1 数据绑定 65

4.2.2 条件渲染 70
4.2.3 列表渲染 72
4.2.4 模板 75
4.2.5 事件绑定 77
4.2.6 引用 81
4.3 WXSS 语法 83
4.3.1 语法规则 83
4.3.2 注释 83
4.3.3 选择器 84
4.4 WXSS 基本属性 85
4.5 CSS 和 WXSS 的区别 86
4.5.1 尺寸单位 86
4.5.2 样式导入 87
4.5.3 内联样式 87
4.5.4 全局样式与局部样式 87
4.6 本章小结 87

第三部分 精通微信小程序开发

5 微信小程序开发基础 88
5.1 全局配置 88
5.2 页面配置 93
5.3 注册程序 94
5.4 注册页面 96
5.4.1 页面的生命周期 98
5.4.2 页面的事件处理 99
5.4.3 页面的数据处理 101
5.4.4 页面的栈 103
5.4.5 页面的路由 103
5.5 简单封装与调用 104
5.6 本章小结 106

6 Flexbox 布局 107
6.1 基本要素 107
6.2 容器属性 108
6.2.1 display 109
6.2.2 flex-direction 109
6.2.3 flex-wrap 110
6.2.4 flex-flow 110
6.2.5 justify-content 110
6.2.6 align-item 111
6.2.7 align-content 112
6.3 子元素属性 112
6.4 position 属性 115
6.5 边框、空隙与填充 120
6.6 本章小结 121

7 组件的开发应用 122
7.1 视图容器组件 124
7.1.1 view 124
7.1.2 scroll-view 126
7.1.3 swiper 129
7.2 基础内容组件 131
7.2.1 icon 132
7.2.2 text 132
7.2.3 progress 135
7.3 表单组件 135
7.3.1 button 136
7.3.2 checkbox 138
7.3.3 radio 139
7.3.4 input 142
7.3.5 textarea 146
7.3.6 form 149

7.3.7 label 151
7.3.8 picker 154
7.3.9 picker-view 158
7.3.10 slider 160
7.3.11 switch 162
7.4 多媒体组件 164
7.4.1 image 164
7.4.2 audio 167
7.4.3 video 170
7.5 地图组件 174
7.6 导航组件 177
7.7 画布组件 179
7.8 客服会话按钮 182
7.9 本章小结 183

8 API 接口 184

8.1 网络相关 185
8.1.1 发送请求 185
8.1.2 上传和下载 187
8.1.3 WebSocket 189
8.2 多媒体 193
8.2.1 图片 193
8.2.2 录音 196
8.2.3 音频 202
8.2.4 背景音乐 204
8.2.5 音频组件控制 208
8.2.6 视频 209
8.2.7 视频组件控制 212
8.3 文件 214
8.4 数据缓存 218
8.5 位置 224
8.5.1 获取与查看位置 224

8.5.2 地图组件控制 227
8.6 设备 229
8.6.1 系统信息 229
8.6.2 网络状态 231
8.6.3 重力感应 232
8.6.4 罗盘 232
8.6.5 拨打电话 233
8.6.6 扫描二维码 233
8.7 界面 234
8.7.1 交互反馈 234
8.7.2 设置导航条 239
8.7.3 导航 241
8.7.4 动画 246
8.7.5 绘画 254
8.7.6 下拉刷新 256
8.8 开放接口 257
8.8.1 登录 257
8.8.2 用户信息 263
8.8.3 微信支付 266
8.8.4 模板消息 267
8.8.5 客服消息 272
8.8.6 分享 275
8.8.7 获取二维码 276
8.9 本章小结 276

9 组件进阶 277

9.1 九宫格 278
9.2 页脚 280
9.3 加载更多 283
9.4 导航条 287
9.5 搜索条 290
9.6 字母列表导航条 295

9.7 日历 299
9.8 本章小结 305

第四部分 自己动手开发微信小程序

10 综合案例——音乐播放小程序 306
10.1 项目需求 306
10.2 项目结构 307
10.3 配置项目文件 308
10.4 首页 index 308
10.4.1 推荐页 311
10.4.2 排行页 317
10.4.3 检索页（上） 325
10.4.4 检索页（中） 332
10.4.5 检索页（下） 340
10.5 列表页 346
10.5.1 获取列表页数据 346
10.5.2 页面渲染 353
10.5.3 完成相似页面 357
10.6 音乐播放页 364
10.7 本章小结 376

11 发布微信小程序 378
11.1 设置服务器域名 378
11.2 上传与审核步骤 380
11.3 微信小程序数据分析 382
11.4 常见问题及注意事项 384
11.5 微信小程序审核不通过原因整理汇总 386
11.6 本章小结 388

附录 A 389

1 微信小程序介绍

随着互联网的高速发展，国内互联网应用走向多元化，各行各业都在转型加入互联网计划。李克强总理也提出制定“互联网 +”行动计划，推动移动互联网、云计算、大数据、物联网等与现代制造业结合，促进电子商务、工业互联网和互联网金融健康发展，引导互联网企业拓展国际市场。但是传统企业的融合、转型与移动化该如何相结合呢？转型成功后用户流量和使用率的问题如何解决呢？

这时，微信小程序这款为移动时代而生的产品出现了，并有效地解决了中国传统企业在转型过程中出现用户流量和使用率的问题，同时也让世界对中国创新能力刮目相看。

1.1 微信小程序是什么

微信小程序，简称小程序，其英文名为 Mini Program。如今，几乎每台智能手机上都会安装微信应用程序，微信的用户也遍布全球。而微信小程序实现了在微信内部不需要下载安装，用户扫一扫或搜索一下即可打开应用程序。

2016 年 9 月 21 日，微信小程序正式开始内测。在微信生态下，这个不需要下载安装、用完即走的微信小程序引起了大家广泛的关注。腾讯云也正式上线了微信小程序的解决方案，提供了微信小程序在云端的技术方案。

2016年1月11日，微信之父张小龙公开亮相，他解读了微信四大价值观。张小龙指出，越来越多的产品注册了公众号，因为在这里开发、推广传播产品的成本比较低。但拆分出来的服务号功能做得却并不理想，所以微信内部正在研究一个新的形态，即微信小程序。

2017年1月9日0点，微信小程序正式低调上线，其为开发人员提供了全新的组件和API技术，为用户开启了不一样的体验模式。

张小龙曾说："让创造发挥价值，好产品应该是用完即走，微信应该是给用户提供便利，而非浪费时间。希望用户在微信里看到的都是自己愿意看到的东西，也希望用户能够留出更多的时间去做其他的事情"。从张小龙说的话我们能看出，微信小程序的目的就是把用户使用微信的动作都集中在微信上，而微信小程序也就将是另一个App Store。

1.2 微信小程序的前景

其实，早在前两年，这种轻应用的模式就一直被大家所关注，不管是百度、UC还是Google，都在尝试基于Web模式做一种小成本、跨平台性好的轻应用，由于各种原因，最后都没有实现。而一直沉默的微信在2017年发布了新产品——微信小程序，打破了这个一直不能解决的问题。如今的微信掌握着数亿用户，也成为目前国内用户量最大的App应用。统计数据表明，微信用户日常在微信上花费的时间平均超过4个小时以上，这也是微信小程序之所以会成功的关键吧。可以看出，微信早已不满足于其在社交层面上的这点流量了，现在已经上升到要与操作系统分庭抗礼了。在未来，这个昔日免费的社交应用也将成为集社交、游戏、购物、生活服务于一身的"超级App"，从而进一步影响我们的生活，具体影响包括以下几个方面。

(1) 对于开发者的影响。微信小程序推出后，开发者也许会发现，微信小程序的语法和前端语法没什么区别，目前来看，对前端开发者来说，从网页迁移到微信小程序成本很低。可见在将来，各行各业对小程序的需求会让前端开发人员迅速增长，而对于只会原生App的开发人员，就会增加一些压力了。

(2) 对于互联网创业企业的影响。近年来，原生APP的开发和推广成本日益增高，从而造成创业企业早期用户积累的成本也越来越高。而通过微信小程序开发，依托于微信本身数亿的用户量，可以以更低的成本来完成早期用户的积累。在开发App之前，可以先用微信小程序开发，这样可以以更快、成本更低的方式进行试错和更新。

(3) 对于小城市企业的影响。一直以来，小城市缺乏技术人员的问题难以解决，微信小程序出现后，这个问题就可以尝试解决了。小城市的企业培养一个前端技术开发人员远比培养一个后端开发人员、架构师、APP 开发人员要容易得多，而且成本也降低了很多。

(4) 对于互联网推广的影响。笔者记得在某贷宝盛行的时期，经常可以看到这样的场景：在一些超市的门口摆放着一张桌子，桌子上摆放了一堆礼品，旁边立着一个易拉宝，两个地推人员在宣传，让用户下载 App 即可送礼物。对于做市场和地推的朋友，这种场景可能并不陌生，这是线下推广时使用的老手段。而有了微信小程序后再去做地推时，只需要携带海报、宣传册等工具即可，这样成本会少很多。

(5) 对于想转型的传统企业的影响。现在，各行各业都在互联网化，传统企业大批转型为互联网模式，所以眼下传统企业面临的问题就是技术不成熟、成本高。而微信小程序能为传统企业降低成本，为传统企业转型铺平道路，给传统企业试错和迭代更新的机会。传统企业通过这种模式把线上、线下和用户群体融合后，再考虑是否需要组建一支全方位的互联网团队。

上面介绍的是微信小程序对未来的一些影响，可能读者还是不能理解。其实微信小程序在很多场景中都可以应用，比如在餐馆点餐、在医院挂号、叫出租车、办理会员卡等。下面以在医院挂号场景来举例。在将来，我们在家里或者在医院可以使用微信小程序在线查询自己需要挂号的科室，并直接在线付款。付款成功后会生成二维码，在医院指定的机器前即可扫描手机二维码，自动打印挂号条，这样大大节约了患者等待排队的时间。如果能全面打通医院系统，以后我们甚至可以在医生开完药以后在线直接付款，再也不用排队了。所以微信小程序在未来的前景、影响难以估计，笔者认为，虽然微信小程序来得有些晚，改变不了过去已经饱和的移动互联网市场，但是足以影响移动互联网的未来。

1.3 微信小程序与订阅号、服务号的区别

微信的订阅号、服务号和微信小程序都属于微信生态下的产品，所以很多人将它们进行比较，那么到底它们之间到底有什么区别呢？

首先来看一下微信的订阅号和服务号的区别：订阅号主要的功能是帮助企业或个人在微信中向用户传达资讯消息，为企业或个人提供一种新的信息传播方式。服务号可以帮助企业和组织在微信中向用户提供服务，为企业和组织提供了更强大的业务服务与用户管理能力，同时还提供了让用户直接与企业的客服进行沟通等功能。

综上所述，我们大致能看出，订阅号和服务号都是以微信的主体聊天界面为基础，所以它们的界面和功能早就约定好了规则。而微信小程序允许开发者自定义界面，还可以根据开发者的产品需求自定义功能，从而打破了订阅号和服务号一成不变的规则。

订阅号与服务号作为腾讯线上对线上模式的重要组成部分，我们已经感受到了其对我们生活的影响。低成本的推广模式配合微信庞大的用户群，使订阅号与服务号成为中小企业和个人商户的首选。微信小程序同样继承了微信的用户广、成本低的优势，但是其更主要的作用是打开线上对线下的大门，让人们在日常生活中经常接触的东西通过一种非常简易的模式在互联网上连接起来，达到更便捷的目的，这也是物联网发展的体现。微信小程序在上线之初并没有像订阅号与服务号那样迅速被大家所接受，但我们也能看到像摩拜单车这种非常适合微信小程序的公司通过与微信团队合作，利用微信小程序更好地为老百姓提供服务。相信随着物联网精神的普及以及腾讯对微信小程序的推广，微信小程序一定会创造出独特的未来。

1.4 如何创建一个微信小程序

学习任何一门语言都是从最基本的“Hello World”开始的，所以，在学习微信小程序开发之前，我们先来创建一个自己的小程序。同时，微信公众平台为我们提供了微信小程序接入指南（https://mp.weixin.qq.com/debug/wxadoc/introduction/index.html?t=2017117），其中列出了以下几项：

(1) 产品定位及功能介绍

(2) 微信小程序注册

(3) 微信小程序信息完善及开发前准备

(4) 开发者工具的使用

(5) 代码审核与发布

(6) 微信小程序申请微信认证

(7) 微信小程序申请微信支付

(8) 微信小程序绑定微信开放平台账号

以上操作在本书中会进行简单的介绍，下面主要介绍如何注册成为开发者和安装工具。

1.4.1 成为微信小程序开发者

我们可以通过微信公众平台里的微信小程序入口（https://mp.weixin.qq.com/cgi-bin/wx）来注册成为微信小程序开发者，注册页面如图 1-1 所示。

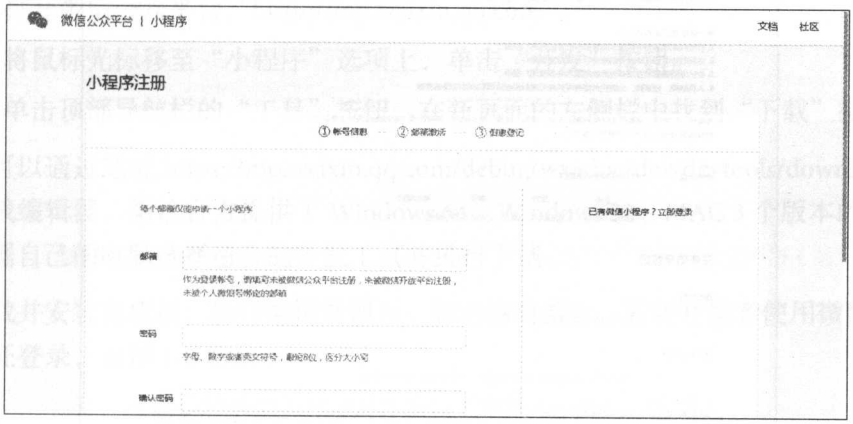

图 1-1 小程序注册页面

填写完邮箱、密码等信息后，单击“注册”按钮，激活邮箱，如图 1-2 所示。

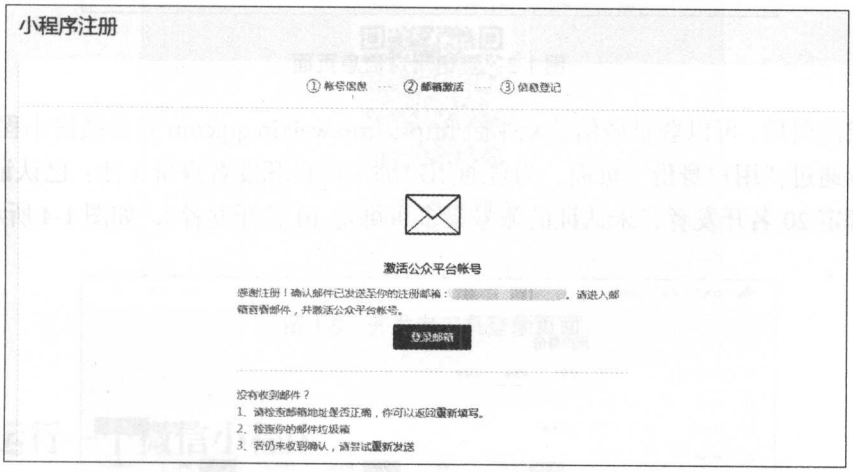

图 1-2 激活邮箱页面

最后登记用户信息，如图 1-3 所示，但是微信公众平台为了真实性和安全性，目前只针对企业用户开放注册（2017 年 3 月 27 日微信公众平台发布消息称，开放对个人开发者申请微信小程序的权限）。

图 1-3 登记用户信息页面

完成注册后，可以登录微信公众平台 https://mp.weixin.qq.com 完善微信小程序信息，并且可以通过“用户身份”页面，为普通用户绑定项目开发者身份（注：已认证的账号最多可绑定 20 名开发者，未认证的账号最多可绑定 10 名开发者），如图 1-4 所示。

图 1-4 用户绑定页面

1.4.2 安装开发者编辑器

要开发微信小程序，需要用到“微信 Web 开发者工具”，通过以下方式可以找到工具的下载地址：

(1) 打开微信公众平台“https://mp.weixin.qq.com”。

(2) 将鼠标光标移至“小程序”选项上，单击“开发”按钮。

(3) 单击顶部导航栏的“工具”按钮，在新页面的左侧栏中找到“下载”按钮。

也可以通过地址 https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/download.html 直接下载编辑器，微信官方提供了 Windows 64、Windows 32、MAC 3 个版本的编辑器，可以根据自己的电脑选择适合的开发工具并进行下载。

下载并安装完成后，运行编辑器即可。启动编辑器后，需要开发者使用微信扫描二维码验证登录，如图 1-5 所示。

图 1-5 开发者工具登录页面

1.5 运行一个微信小程序

登录成功后，会出现如图 1-6 所示的页面，在这里选择调试类型：可以选择本地微信小程序项目或公众号网页开发。这里选择“本地小程序项目”，然后添加 AppID、项目名称和项目目录，如图 1-7 所示（要获取 AppID，可以登录 https://mp.weixin.qq.com，在网站的“设置”—“开发设置”页面中，可以查看到微信小程序的 AppID，如图 1-8 所示）。

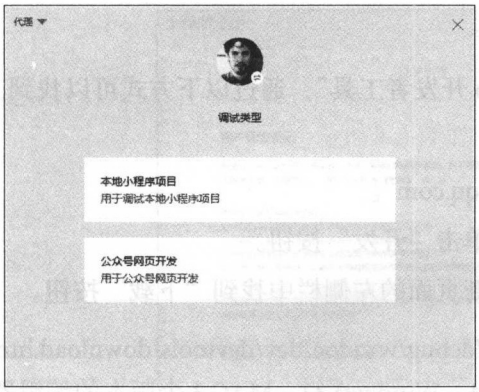

图 1-6 选择调试类型页面

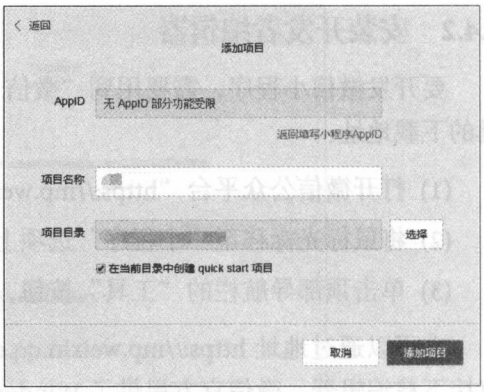

图 1-7 添加项目页面

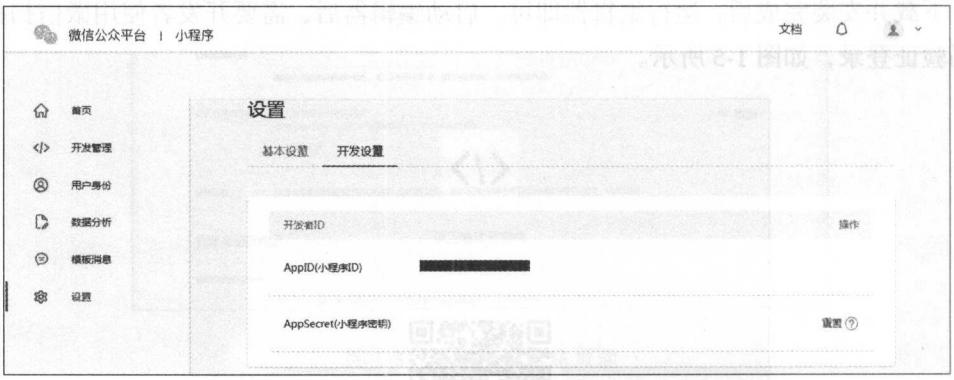

图 1-8 在“开发设置”页面中查看 AppID

勾选“在当前目录中创建 quick start 项目”选项，会自动生成一个简单的微信小程序结构，不勾选此选项的话，则打开空白文件夹。如果是刚开始学习微信小程序，则建议勾选此选项，在此基础上编写小程序。

全部填写完成后，单击“添加项目”按钮，会自动跳转到编辑器的主页面，如图 1-9 所示。

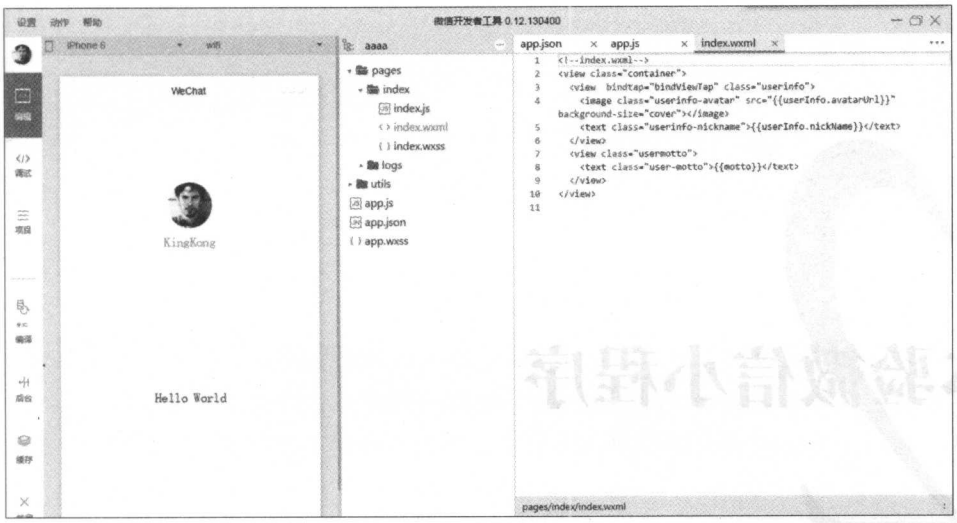

图 1-9 编辑器页面

1.6 本章小结

本章主要介绍了微信小程序的简史，以及发展前景，而且简单地讲解了微信小程序开发的前期准备工作和工作软件的安装与运行，后面会详细介绍微信小程序的组件、API、布局方式、丰富的案例，以及在开发中可能遇到的关于 JavaScript 和 CSS 的语法等。

2 体验微信小程序

第 1 章简单地介绍了一下微信小程序，同时也讲解了创建并运行一个微信小程序的案例。本章会详细介绍微信小程序工具以及微信小程序结构，包括微信小程序工具操作、目录结构、常用快捷键等。

2.1 界面与操作

微信小程序开发工具的功能非常强大和便捷，其工具内部集成了代码编辑、开发调试及程序发布等功能，微信小程序开发工具界面如图 2-1 所示。

微信小程序开发工具界面基本划分为四大区域：顶部菜单区域、导航菜单区域、目录文件 & 模拟运行区域、编辑 & 调试开发区域。顶部菜单区域和导航菜单区域是固定的，目录文件 & 模拟运行区域和编辑 & 调试开发区域会根据选择导航菜单的不同功能而有所不同。

顶部菜单区域在开发小程序的过程中使用的机会并不多，其中“设置”页面用于配置开发模拟器运行程序时连接的网络代理（见图 2-2）。“动作”页面用于刷新、后退、前进等操作。“帮助”页面用于显示开发工具的版本和版权声明等信息。导航菜单区域是开发者经常使用的功能区域。特别是其中的编辑和调试的功能对开发者来说是非常重要的功能。下面重点介绍这两个功能。

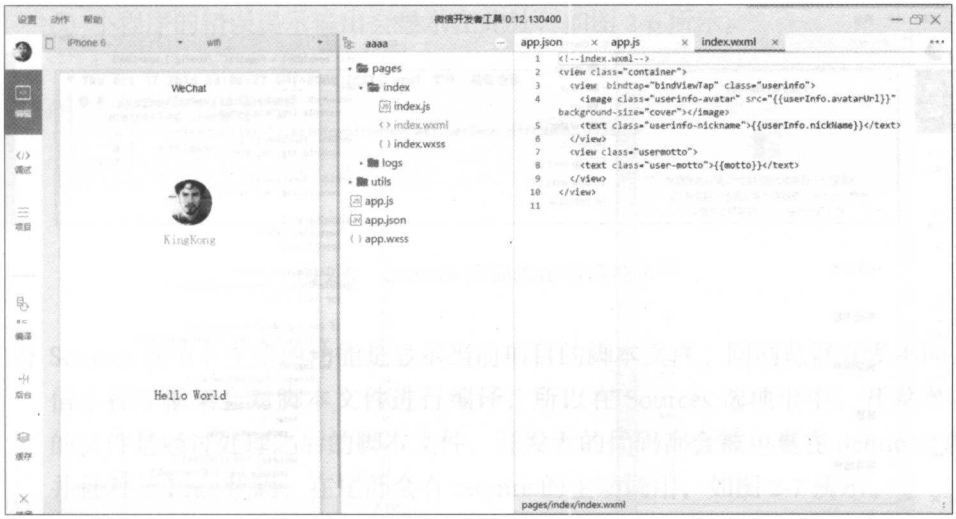

图 2-1 开发工具主界面

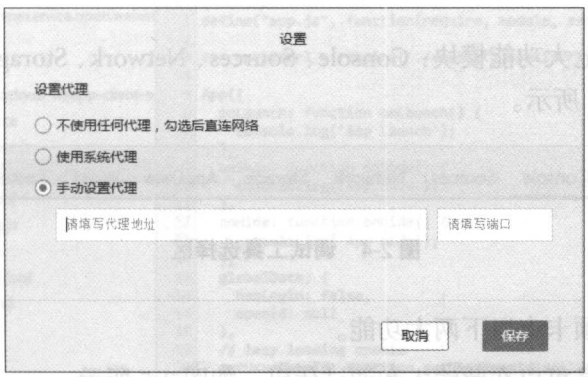

图 2-2 设置页面

2.2 编辑功能

编辑区分为两个部分，在右侧的编辑区域可以对当前项目进行编写、文件添加和删除以及重命名等基本操作。在左侧的模拟器区域可以实时预览编辑的情况，如图 2-3 所示。编辑器同时也提供了比较完善的自动补全和自动保存功能，编写代码后，工具会自动帮助用户保存当前代码为编辑状态，如果直接关闭工具或者切换项目，也不会丢失当前代码的编辑状态。但是要注意，处于编辑状态的代码只保存到工具内部，并没有写到硬盘上，只有手动保存文件后，修改的内容才会写到硬盘上，并触发实时预览。

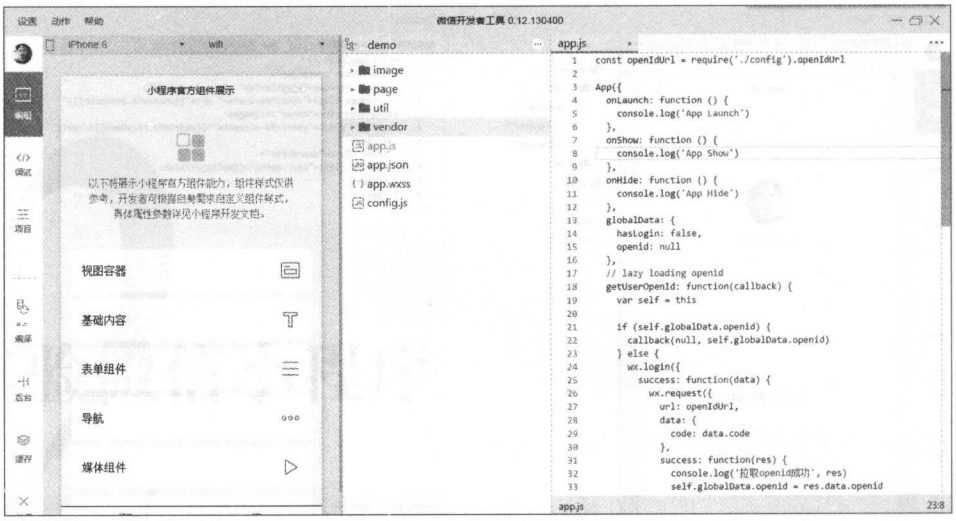

图 2-3 编辑区页面

2.3 调试功能

调试工具分为七大功能模块：Console、Sources、Network、Storage、AppData、Wxml 和 Sensor，如图 2-4 所示。

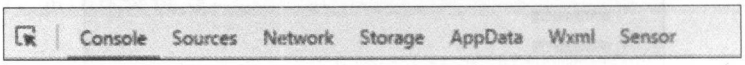

图 2-4 调试工具选择区

(1) Console 选项卡有以下两大功能。

① 开发人员可以在此输入和调试代码，如图 2-5 所示。

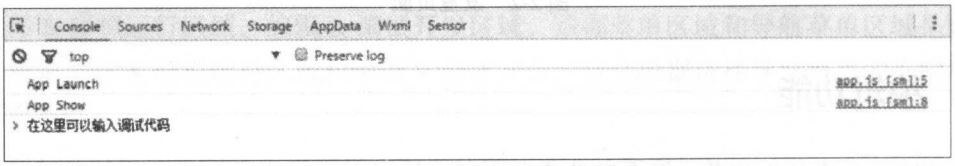

图 2-5 Console 页面调试

② 小程序的错误提示输出会显示在此处，如图 2-6 所示。

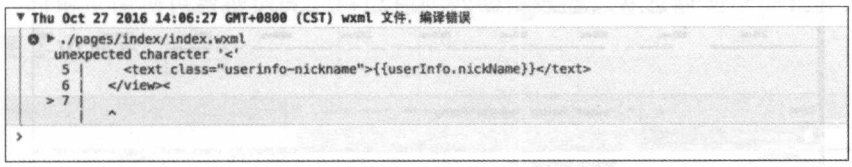

图 2-6 Console 页面输出错误提示

(2) Sources 选项卡主要的功能是显示当前项目的脚本文件。同浏览器开发不同，微信小程序框架会对脚本文件进行编译，所以在 Sources 选项卡中，开发者看到的文件是经过处理之后的脚本文件，开发者的代码都会被包裹在 define 函数中，并且对于 Page 代码，在尾部会有 require 的主动调用，如图 2-7 所示。

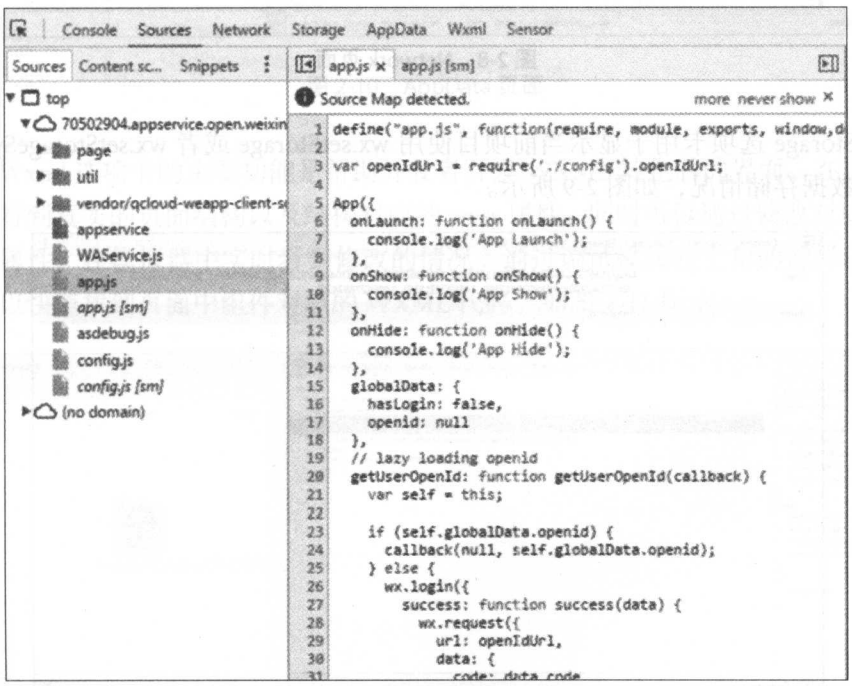

图 2-7 Sources 页面

(3) Network 选项卡的主要功能是观察和显示 request 和 socket 的请求情况，如图 2-8 所示。

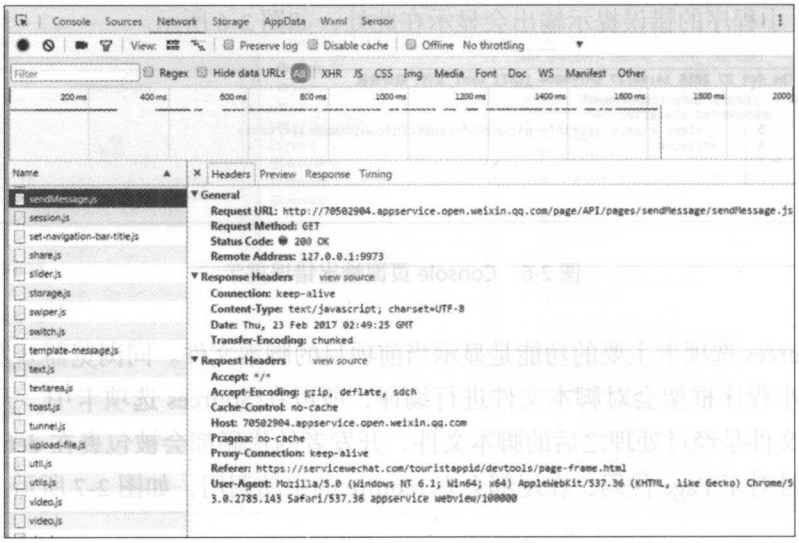

图 2-8 Network 页面

(4) Storage 选项卡用于显示当前项目使用 wx.setStorage 或者 wx.setStorageSync 后的数据存储情况，如图 2-9 所示。

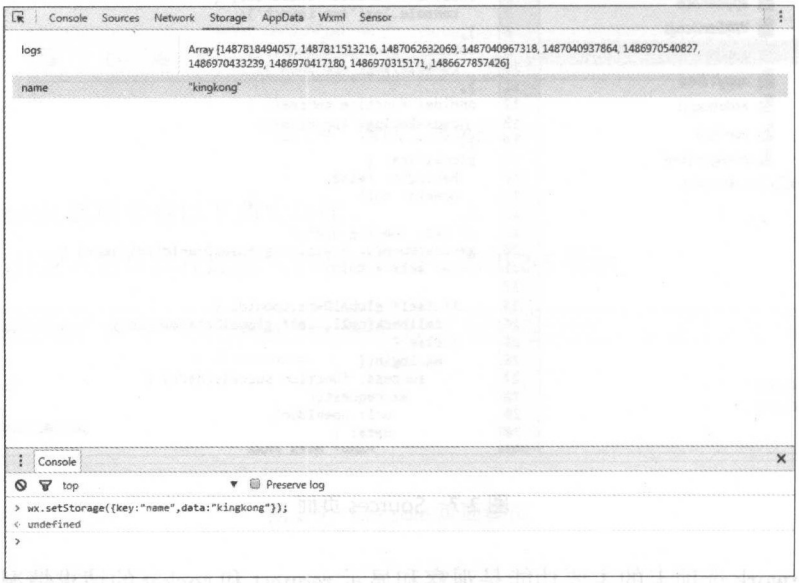

图 2-9 Storage 页面

(5) AppData 选项卡的主要功能是显示当前项目在当前时刻 AppData 的具体数据，实时地反馈项目数据情况，可以在此处编辑数据，并及时地反馈到界面上，如图 2-10 所示。

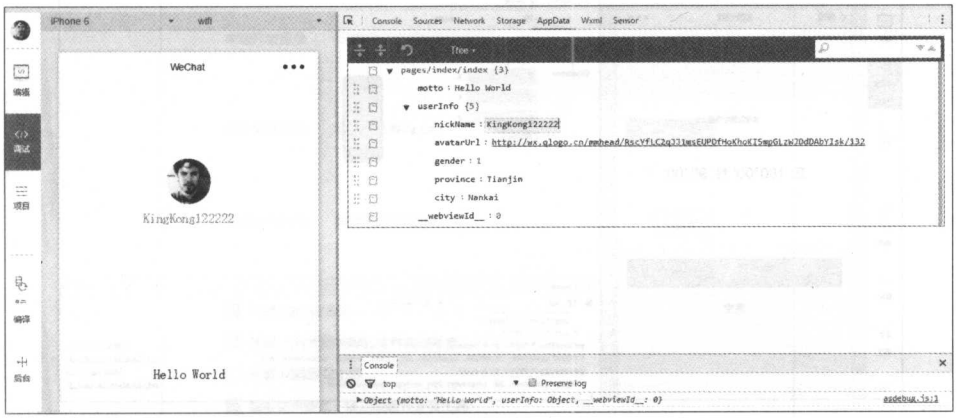

图 2-10 AppData 页面

(6) Wxml 选项卡的主要功能是帮助开发者开发 WXML 转化后的界面。在这里可以看到真实的页面结构以及结构对应的 wxss 属性，同时可以通过修改对应的 wxss 属性，在模拟器中实时看到修改的情况。通过调试模块左上角的选择器，还可以快速找到页面中组件对应的 WXML 代码，如图 2-11 所示。

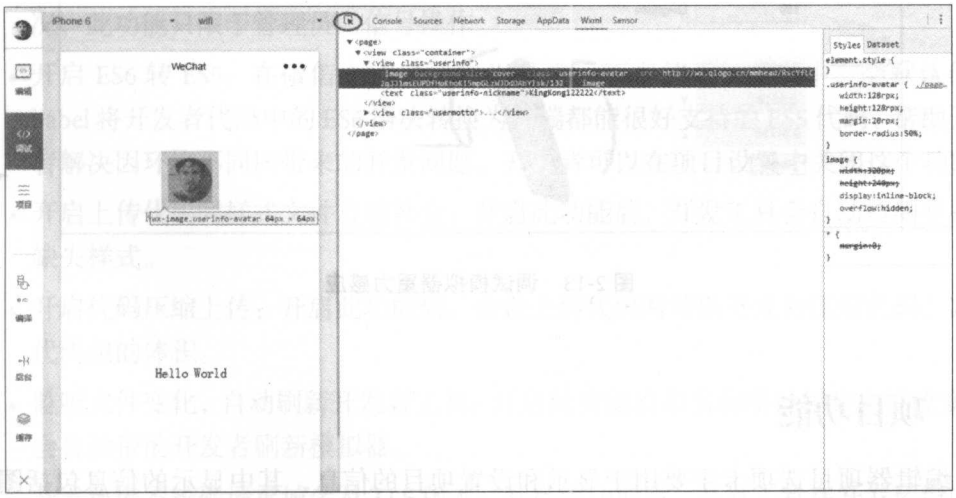

图 2-11 Wxml 页面

(7) Sensor 选项卡有两大功能。

① 开发人员可以在这里选择模拟地理位置，如图 2-12 所示。

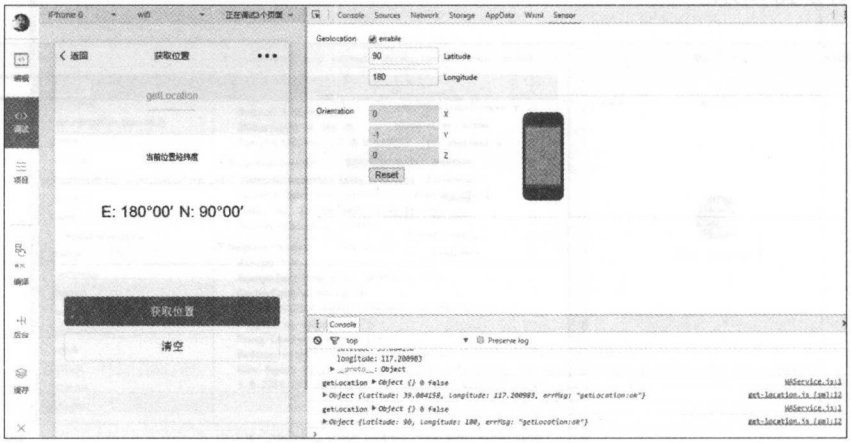

图 2-12 设置模拟地理位置

② 开发人员可以在这里模拟移动设备表现，用于调试重力感应 API，如图 2-13 所示。

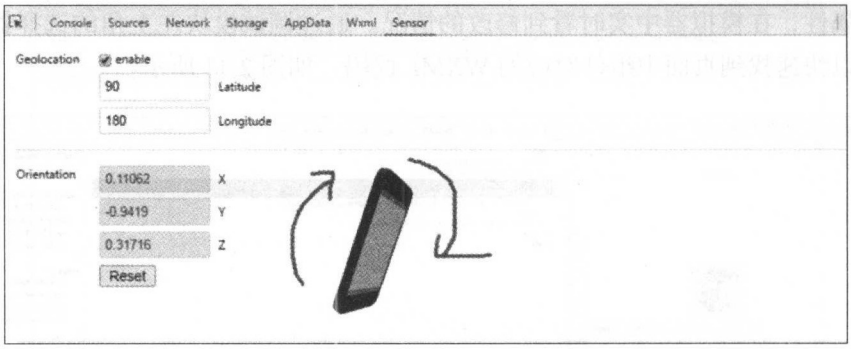

图 2-13 调试模拟器重力感应

2.4 项目功能

编辑器项目选项卡主要用于显示和设置项目的信息，其中显示的信息包括图标、AppID、本地开发目录、最新更新时间、最近上传时间以及代码包大小，如图 2-14 所示。

图 2-14 项目设置页面

- 预览：指上传源码到微信服务器并成功后会生成一个二维码，开发者用最新版本的微信扫描二维码后，即可在手机上体验到最新的效果。
- 上传：指上传源码到微信服务器后，开发者可以在微信管理后台看到本地提交情况，此功能只限于管理员微信号操作。
- 开启 ES6 转 ES5：在微信 0.10.101000 以及之后版本的开发工具中，会默认使用 babel 将开发者代码中的 ES6 语法转换为三端都能很好支持的 ES5 代码，帮助开发者解决因环境不同所带来的开发问题。开发者可以在项目设置中关闭这个功能。
- 开启上传代码时样式文件自动补全：开启此功能后，开发工具会自动检测并补全缺失样式。
- 开启代码压缩上传：开启此功能后，会在上传代码时帮助开发者压缩代码，减小代码包的体积。
- 监听文件变化，自动刷新开发者工具：开启此功能后和当前项目相关文件改变时，会自动帮助开发者刷新模拟器。
- 开发环境不校验请求域名及 TLS 版本：开启此功能后，开发工具在开发环境中将不会校验安全域名和 TLS 版本。

2.5 常用快捷键

为了提高开发效率，微信小程序的编辑器中设置了很多快捷键供开发者使用，下面介绍一些常用的快捷键的功能，如表 2-1 所示。

表 2-1

键　盘	作　用
Ctrl+S	保存文件
Ctrl+[，Ctrl+]	代码行缩进
Ctrl+Shift+[，Ctrl+Shift+]	折叠打开代码块
Ctrl+C，Ctrl+V	复制并粘贴，如果没有选中任何文字，则复制并粘贴一行
Shift+Alt+F	代码格式化
Alt+Up 或 Alt+Down	向上或向下移动一行
Shift+Alt+Up 或 Shift+Alt+Down	向上或向下复制一行
Ctrl+Shift+Enter	在当前行上方插入一行
Ctrl+Shift+F	全局搜索
Ctrl+End	移动到文件结尾
Ctrl+Home	移动到文件开头
Ctrl+I	选中当前行
Shift+End	选择从光标到行尾
Shift+Home	选择从行首到光标处
Ctrl+Shift+L	选中所有匹配
Ctrl+D	选中匹配
Ctrl+U	光标回退
Ctrl+\	隐藏侧边栏
Ctrl+M	打开或者隐藏模拟器

2.6 项目的目录与文件结构

在创建小程序项目时，小程序开发工具会默认生成一些文件，这些文件就是小程序的基本结构。除此之外，在开发过程中，可以根据产品需求自定义小程序的目录和文件，本章会简单介绍一下微信小程序的结构。

项目目录

微信小程序的主体部分由三个文件组成，这些文件必须放到项目的根目录下，如表2-2所示。

表 2-2

文　件	是否必填	作　用
app.js	是	微信小程序逻辑
app.json	是	微信小程序公共设置
app.wxss	否	微信小程序公共样式表

微信小程序页面主要有四种文件类型，如表2-3所示。

表 2-3

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

文件结构

app.json文件是微信框架的配置文件，可以在里面对微信小程序进行全局配置，如窗口背景色、导航条样式、默认标题等。

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

app.js 文件是脚本文件，可以在这个文件中监听处理小程序的生命周期和声明全局变量等，微信小程序提供了丰富的 API，详细的 API 介绍请参考第 8 章内容。

```
App({
  onLaunch: function () {
    //调用API从本地缓存中获取数据
    var logs = wx.getStorageSync('logs') || []
    logs.unshift(Date.now())
    wx.setStorageSync('logs', logs)
  },
  getUserInfo:function(cb){
    var that = this;
    if(this.globalData.userInfo){
      typeof cb == "function" && cb(this.globalData.userInfo)
    }else{
      //调用登录接口
      wx.login({
        success: function () {
          wx.getUserInfo({
            success: function (res) {
              that.globalData.userInfo = res.userInfo;
              typeof cb == "function" && cb(that.globalData.userInfo)
            }
          })
        }
          });
    }
  },
  }
})
```

app.wxss 文件定义的是全局样式，而在 page 中定义的.wxss 文件为局部样式，只作用于局部，局部样式中定义的样式会覆盖全局样式中定义的样式。

```
**app.wxss**/
.container {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  box-sizing: border-box;
}
```

2.7 本章小结

本章主要讲解了开发工具的操作、使用方式。古语有云：“工欲善其事，必先利其器。”能否熟练使用开发工具，决定了我们的开发效率，所以读者要多多练习，熟能生巧。

3 JavaScript 语法

问题来了，为什么我们要学 JavaScript?

简单粗暴的回答就是：因为你没有选择。小程序主要的逻辑功能都是通过 JavaScript 实现的。

JavaScript 简称 JS。在 IT 行业中有一句名言：“凡是能用 JS 开发的，最终都会用 JS 开发。”

上面的言论是否正确这里不做论证，但是从侧面可以看出 JS 的强大。有些读者可能没有任何编程基础，有些读者可能掌握过其他编程语言，例如 Java、C/C++ 等。无论读者是否有基础都没有关系，我们从零开始介绍 JavaScript 语法。

3.1 JavaScript 简介

要了解 JavaScript，首先要回顾一下 JavaScript 的诞生。

在 20 世纪 90 年代，当时的网景公司凭借其 Navigator 浏览器，成为开启 Web 时代最著名的互联网公司之一。

由于网景公司希望能在静态 HTML 页面中添加一些动态效果，于是让一个叫 Brendan Eich 的人在两周之内设计出了 JavaScript 语言。你没看错，他只用了 10 天时间。

为什么起名叫 JavaScript？因为当时 Java 语言非常红火，所以网景公司希望借助 Java 的名气来推广 JavaScript，但事实上，JavaScript 除了在名字上有点像 Java，其他部分基本上与 Java 没什么关系。JavaScript 和 Java 的关系类似于“雷峰塔”和“雷锋”的关系。

ECMAScript 的来历

因为网景开发了 JavaScript，一年后微软又模仿 JavaScript 开发了 JScript，为了让 JavaScript 成为全球标准，几家公司联合 ECMA（European Computer Manufacturers Association）组织定制了 JavaScript 语言的标准，被称为 ECMAScript 标准。

所以简单地说，ECMAScript 是一种语言标准，而 JavaScript 是网景公司对 ECMAScript 标准的一种实现。

不过在大多数时候，我们还是用 JavaScript 这个词。如果你遇到 ECMAScript 这个词，那么直接把它替换为 JavaScript 就行了。

JavaScript 本身有很多设计缺陷，而 JavaScript 的标准——ECMAScript 在不断发展，从而逐渐弥补了 JavaScript 一开始的设计缺陷。最新版的 ECMAScript 6 标准（简称 ES6，别名 ES2015）已经在 2015 年 6 月正式发布了，ES7 标准也已经制定了。

做微信小程序需要学什么

微信小程序一开始并不支持 ES6，后来通过 ES6 转 ES5 的功能，兼容了 ES6 语法，如图 3-1 所示。

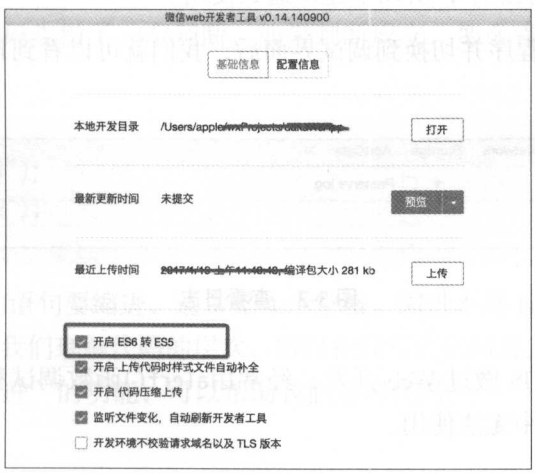

图 3-1 ES6 转 ES5 功能

不同于开发网页，开发微信小程序基本上都是使用 JS 核心语法，有些和网页开发相关的语法并不能在小程序中使用。虽然 JavaScript 一直在升级，但是其核心语法并没有什么变化。

要学习微信小程序就需要学会 JS 核心语法，这也是本书要着重介绍的内容。

3.2 基础语法

简单认识了 JavaScript，接下来我们就来学习 JavaScript（以下简称 JS）语法。为了方便演示 JS 语法，我们在空文件中创建一个新的小程序项目，并生成 quick start 项目。

微信小程序中带 js 后缀的文件都是通过 JS 编写的，在默认生成的项目中，index.js 是控制第一个界面的逻辑。

onLoad 函数是小程序生命周期方法中最先调用的，下面演示一个 JS 代码：通过 `console.log("打印日志");` 可以打印输出日志：

```
Page({
  //...
  onLoad: function () {
    //演示代码
    console.log("打印日志");
})
```

其实直接在 index.js 文件中`Page()`以外也能直接调`console.log("打印日志")`，但是这样就脱离微信小程序框架了，所以不建议这么使用。

保存代码，编译程序并切换到调试界面后，我们就可以看到日志输出了，如图 3-2 所示。

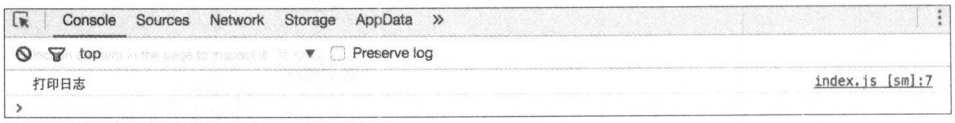

图 3-2 查看日志

有些读者可能用 JS 做过 Web 开发，经常用`alert()`函数调试程序，需要注意的是，`alert()`函数在小程序中无法使用。

3.2.1 语句和语句块

JavaScript 的语法和其他编程语言类似，每个语句以;结束，语句块用{...}。但是，JavaScript 并不强制要求在每个语句的结尾加；，JavaScript 代码的引擎会自动在每个语句的结尾补上;。

例如下面的两条语句加不加分号都是可以正常执行的。

```
onLoad: function () {
    console.log("不加分号")
    console.log("加分号");
}
```

但是为了避免潜在的风险，建议读者在语句结束时一定要加上;。

下面的一行代码就是一个完整的赋值语句，其中var是声明变量的关键字：

```
var i=1;
```

下面的一行代码是一个字符串，但仍然可以将其视为一个完整的语句：

```
'Hello, world';
```

如果一行代码包含两条语句，则每条语句都要用;表示语句结束，例如:

```
var x = 1; var y = 2;
```

按照江湖规矩，一般每条语句单独在一行。

语句块是一组语句的集合。

例如，下面的代码先做了一个判断，如果判断成立，那么将执行{...}中的所有语句：

```
if(2>1){
  console.log("语句1");
  console.log("语句2");
}
```

花括号{...}内的语句要缩进，通常是 4 个空格。缩进不是 JavaScript 语法要求必须做的，但缩进有助于我们理解代码的层次，所以在编写代码时要遵守缩进规则。小程序编辑器具有“自动缩进”的功能，可以帮助我们整理代码。

{...}还可以嵌套，形成层级结构：

```
if (2 > 1) {
    console.log("语句块1");
    if (2 < 3) {
        console.log("语句2");
    }
}
```

JavaScript 本身对嵌套的层级没有限制，但是过多的嵌套无疑会大大增加阅读者看懂代码的难度。遇到这种情况，需要把部分代码抽出来，作为函数来调用，这样可以减少代码的复杂度。

3.2.2 注释

和 Java 等编程语言一样，在 JS 代码中，以//开头直到行末的字符被视为行注释，注释是给开发人员看的，JavaScript 引擎会自动将其忽略：

```
//注释
console.log("日志");
```

块注释是用/*...*/把多行字符包裹起来，把这一大“块”内容视为一个注释：

```
/* 从这里开始是块注释
仍然是注释
仍然是注释
注释结束 */
```

3.2.3 变量

变量这个词来源于数学，变量的概念基本上和数学中的方程变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意的数据类型。

变量在 JavaScript 中用一个变量名表示。给变量命名需要注意以下三点：

(1) **变量名是大小写英文、数字、$ 和 _ 的组合，且不能用数字开头。**

(2) **变量名也不能是 JavaScript 的关键字，如if、while等。**

(3) **JavaScript 严格区分大小写，A和a表示两个不同的变量。**

虽然变量名也可以使用中文，但是如果你不想被你的项目经理批评，就最好不要这样做。

声明一个变量用var语句，比如：

```
var a; // 申明了变量a，此时a的值为undefined
```

在 JavaScript 中，使用等号 = 对变量进行赋值。可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量，但是要注意，只能用 var 声明一次，例如：

```
var a = 123; // a的值是整数123
a = 'ABC'; // a变为字符串
```

ES6 语法允许我们使用 let 对变量进行赋值，和var语法类似，区别就是var赋值的变量作用域要大很多，let赋值的变量只在花括号（{}）内生效。例如：

```
if(true){
  let a= "变量a" //如果换成var声明就不会报错
}
console.log(a);// 报错
```

3.2.4 常量

由于var和let申明的是变量，要申明一个常量，在 ES6 之前是不行的，我们通常用全部大写的变量来表示“这是一个常量，不要修改它的值”：

```
var PI = 3.14;
```

ES6 标准引入了新的关键字 const 来定义常量，const与let都具有块级作用域，其只在所在的花括号（{}）内生效。

```
const PI = 3.14;
```

3.2.5 数据类型

JavaScript 为了满足具体需求，设计了以下几种数据类型。

(1) 数字

(2) 字符串

(3) 布尔值

(4) Null

(5) Undefined

(6) 数组

(7) 对象

数字

数字，英文名称为Number。JavaScript的数字类型不区分整数和浮点数，统一用Number表示。以下都是合法的数字类型：

```
123; // 整数123
0.456; // 浮点数0.456
1.2345e3; // 科学记数法表示为1.2345x1000，等同于1234.5
-99; // 负数
NaN; // NaN表示Not a Number，当无法计算结果时用NaN表示
Infinity; // Infinity表示无限大，当数值超过了JavaScript的
          // Number所能表示的最大值时，就表示为Infinity
```

由于计算机使用二进制，所以有时候用八进制或者十六进制表示整数比较方便。

(1) 八进制用前缀0和1~7表示，例如：010等于十进制中的8。

(2) 十六进制用前缀0x和0~9，a~f表示，例如：0xff00，0xa5b4c3d2等，它们和用十进制表示的数值完全一样。

数字类型可以直接做四则运算，规则和数学一致，运算符有加（+）、减（-）、乘（*）、除（/）、求余（%）：

```
1 + 2; // 结果为3
(1 + 2) * 5 / 2; // 结果为7.5
2 / 0; // 结果为Infinity
0 / 0; // 结果为NaN
10 % 3; // 结果为1
```

字符串

字符串，英文名称为String。字符串是以单引号（'）或双引号（"）括起来的任意文本，比如'abc'，"xyz"等。

注意：''或""本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc'只有a，b，c这3个字符。如果单引号'本身也是一个字符，那就可以用引号（""）括起来，比如：

```
"I'm OK"
```

其中包含的字符是I、'、m、空格、O和K这6个字符。

转义符号

如果字符串内部既包含单引号（'），又包含双引号（"），怎么办？可以用转义字符（\）来标识，比如：

```
'I\'m \"OK\"!';
```

其表示的字符串内容是：I'm "OK"!。

转义字符（\）可以转义很多字符，比如\n表示换行，\t表示制表符，字符\本身也要转义，所以\\表示的字符就是\。

ASCII 字符可以以\x## 形式表示成十六进制，例如：

```
'\x41'; // 完全等同于 'A'
```

还可以用\u#### 表示一个 Unicode 字符：

```
'\u4e2d\u6587'; // 完全等同于 '中文'
```

多行字符串

由于多行字符串用\n写起来比较费事，所以最新的 ES6 标准新增了一种多行字符串的表示方法，用`...`表示：

```
console.log(`这是
            多行
            字符串`);
```

模板字符串

要把多个字符串连接起来，可以使用加号（+）：

```
var name = '小明';
var age = 20;
var message = '你好, ' + name + ', 你今年' + age + '岁了!';
console.log(message);// 输出 你好小明，你今年20岁了!
```

如果有很多变量需要连接，那么使用加号（+）就比较麻烦。ES6 新增了一种模板字符串，表示方法和上面的多行字符串一样，但是它会自动替换字符串中的变量：

```
var name = '小明';
var age = 20;
var message = `你好, ${name}, 你今年${age}岁了!`;
console.log(message);// 输出 您好小明，你今年20岁了!
```

操作字符串

字符串常见的操作如下，通过length可以获取字符串长度。

```
var s = 'Hello, world!';
s.length; // 13
```

要获取字符串某个指定位置的字符，可以使用下标操作，索引号从 0 开始，这和后面要介绍的数组操作类似：

```
var s = 'Hello, world!';

s[0]; // 'H'
s[6]; // ' '
s[7]; // 'w'
s[12]; // '!'
s[13]; // undefined 超出范围的索引不会报错，但一律返回undefined
```

需要特别注意的是，字符串是不可变的，如果对字符串的某个索引赋值，则在微信小程序中会报错：

```
var s = 'Test';
s[0] = 'X';  //报错
```

indexOf()

可以通过调用 indexOf() 方法找某个字符在字符串中的位置。例如:

```
var str="hello world";
var index=str.indexOf('w');
console.log(index);//输出6
```

没有找到输出 –1。

substring()

可以通过 substring() 方法截取字符串的一部分，返回新的字符串，例如：

```
var str="hello world";
var subStr1=str.substr(1);//从索引1开始到结束
console.log(subStr1);  //输出 "ello world"
var subStr2=str.substr(0,3);//从索引0开始，到索引3结束，但不包括索引3
console.log(subStr2); //输出 "hel"
```

toUpperCase()

toUpperCase()把一个字符串全部变为大写：

```
var s = 'Hello';
s.toUpperCase(); // 返回'HELLO'
```

toLowerCase()

toLowerCase()把一个字符串全部变为小写：

```
var s = 'Hello';
var lower = s.toLowerCase(); // 返回'hello'并赋值给变量lower
lower; // 'hello'
```

布尔值

一个布尔值只有 true 和 false 两种值，要么是 true，要么是 false，可以直接用 true 或 false 表示布尔值，也可以通过布尔运算计算出来：

```
true; // 这是一个true值
false; // 这是一个false值
2 > 1; // 这是一个true值
2 >= 3; // 这是一个false值
```

在条件语句中，我们还可以用非 0 值表示 true，用 0 表示 false，例如：

```
if(1){  //改成0不会输出
  console.log("非0值表示true");
}
```

除了 0，JavaScript 把null、undefined、NaN和空字符串''都视为false，其他值一概视为true。

null 和 undefined

null表示一个“空”的值，它和 0 以及空字符串''不同，0 是一个数值，''表示长度为 0 的字符串，而 null 表示“空”。

在其他语言中，也有类似 JavaScript 的 null 的表示，例如 Java 也用 null 表示，Swift 用 nil 表示，Python 用 None 表示。但是，在 JavaScript 中，还有一个和 null 类似的 undefined，它表示“未定义”。

JavaScript 的设计者希望用null表示一个空的值，而undefined表示值未定义。事实证明，这并没有什么作用，对于区分两者的意义并不大。在大多数情况下，我们都应该用null。undefined仅仅在判断函数参数是否传递的情况下有用。

数组

数组，英文名称为 Array。数组是一组按顺序排列的集合，集合中的每个值被称为元素。不同于 Java 等编程语言，JavaScript 的数组可以包括任意数据类型。例如：

```
[1, 3.14, 'Hello', null, true];
```

上述数组包含 5 个元素。数组用 [] 表示，元素之间用“,”分隔。

另一种创建数组的方法是通过 Array() 函数实现：

```
new Array(1, 2, 3); // 创建了数组[1, 2, 3]
```

然而，考虑到代码的可读性，强烈建议直接使用“[]”。数组的元素可以通过索引来访问。请注意，索引的起始值为 0：

```
var arr = [1,3.14, 'Hello', null, true];
arr[0]; // 返回索引为0的元素，即1
arr[4]; // 返回索引为4的元素，即true
arr[5]; // 索引超出了范围，返回undefined
```

要取得 Array 的长度，可以直接访问 length 属性：

```
var arr = [1,3.14, 'Hello', null, true];
console.log(arr.length); //输出5
```

注意，直接给 Array 的 length 赋一个新的值，会导致 Array 的大小发生变化：

```
var arr = [1, 2, 3];
arr.length; // 3
arr.length = 6; //长度变成6
arr; // arr变为[1, 2, 3, undefined, undefined, undefined]
arr.length = 2;
arr; // arr变为[1, 2]
```

Array可以通过索引把对应的元素修改为新的值，因此，对 Array 的索引进行赋值会直接修改这个 Array：

```
var arr = ['A', 'B', 'C'];
arr[1] = 520;
arr; // arr现在变为['A', 520, 'C']
```

如果通过索引赋值时，索引超过了范围，那么同样会引起 Array 的大小发生变化：

```
var arr = [1, 2, 3];
arr[5] = 'x';
arr; // arr变为[1, 2, 3, undefined, undefined, 'x']
```

大多数其他编程语言不允许直接改变数组的大小，越界访问索引会报错。然而，JavaScript 的 Array 却不会有任何错误。

在编写代码时，不建议直接修改 Array 的大小，访问索引时要确保索引不会越界。

indexOf()

与字符串类似，数组也可以通过 indexOf() 来搜索一个指定的元素的位置：

```
var arr = [10, 20, '30', 'xyz'];
arr.indexOf(10); // 元素10的索引为0
arr.indexOf(20); // 元素20的索引为1
arr.indexOf(30); // 元素30没有找到，返回-1
arr.indexOf('30'); // 元素'30'的索引为2
```

注意，数字 30 和字符串'30'是不同的元素。

slice()

slice() 就是对应字符串的 substring() 方法，它截取数组的部分元素，然后返回一个新的数组：

```
var arr = ['A', 'B', 'C', 'D', 'E', 'F', 'G'];

// 从索引0开始，到索引3结束，但不包括索引3: ['A', 'B', 'C']
arr.slice(0, 3);

// 从索引3开始到结束: ['D', 'E', 'F', 'G']
arr.slice(3);
```

你会注意到slice()的起止参数包括开始索引，不包括结束索引。

如果不给 slice() 传递任何参数，它就会从头到尾截取所有元素。利用这一点，我们可以很容易地复制一个数组：

```
var arr = ['A', 'B', 'C', 'D', 'E', 'F', 'G'];
var aCopy = arr.slice();
aCopy; // ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

push() 和 pop()

push()向数组的末尾添加若干元素，pop()则把数组的最后一个元素删除：

```
var arr = [1, 2];
arr.push('A', 'B'); // 返回Array新的长度: 4
arr; // [1, 2, 'A', 'B']
```

```
arr.pop(); // pop()返回'B'
arr; // [1, 2, 'A']
arr.pop(); arr.pop(); arr.pop(); // 连续pop 3次
arr; // []
arr.pop(); // 空数组继续pop不会报错，而是返回undefined
arr; // []
```

unshift() 和 shift()

如果要往数组的头部添加若干元素，则使用unshift()方法；使用shift()方法会把 Array 的第一个元素删除：

```
var arr = [1, 2];
arr.unshift('A', 'B'); // 返回Array新的长度: 4
arr; // ['A', 'B', 1, 2]
arr.shift(); // 'A'
arr; // ['B', 1, 2]
arr.shift(); arr.shift(); arr.shift(); // 连续shift 3次
arr; // []
arr.shift(); // 空数组继续shift不会报错，而是返回undefined
arr; // []
```

sort()

sort()可以对当前数组进行排序，它会直接修改当前数组的元素位置，直接调用时，按照默认顺序排序：

```
var arr = ['B', 'C', 'A'];
arr.sort();
arr; // ['A', 'B', 'C']
```

如果想按照指定的方式排序，则需要借助函数，在后面介绍函数的时候会再补充相关内容。

reverse()

reverse() 是把整个 Array 的元素给调个过儿，也就是反转：

```
var arr = ['one', 'two', 'three'];
arr.reverse();
arr; // ['three', 'two', 'one']
```

splice()

splice()方法是修改数组的“万能方法”，它可以从指定的索引开始删除若干元素，然后再从该位置添加若干元素。

该方法接收若干参数，第一个参数表示索引开始的位置，第二个参数表示删除的元素的个数，后面索引的元素表示在删除的位置添加的元素。

```
var arr = ['Microsoft', 'Apple', 'Yahoo', 'AOL', 'Excite', 'Oracle'];
// 从索引2开始删除3个元素,然后再添加两个元素:
arr.splice(2, 3, 'Google', 'Facebook'); // 返回删除的元素['Yahoo','AOL','Excite']
arr; // ['Microsoft', 'Apple', 'Google', 'Facebook', 'Oracle']
// 只删除，不添加:
arr.splice(2, 2); // ['Google', 'Facebook']
arr; // ['Microsoft', 'Apple', 'Oracle']
// 只添加，不删除:
arr.splice(2, 0, 'Google', 'Facebook'); // 返回[]，因为没有删除任何元素
arr; // ['Microsoft', 'Apple', 'Google', 'Facebook', 'Oracle']
```

concat()

concat() 方法把当前的数组和另一个数组连接起来，并返回一个新的数组。

请注意，concat() 方法并没有修改当前 Array，而是返回了一个新的 Array。

```
var arr = ['A', 'B', 'C'];
var added = arr.concat([1, 2, 3]);
added; // ['A', 'B', 'C', 1, 2, 3]
arr; // ['A', 'B', 'C']
```

实际上，concat() 方法不仅可以接收数组，还可以接收任意个元素和数组，并且自动把数组拆开，然后全部添加到新的数组里，非常的灵活：

```
var arr = ['A', 'B', 'C'];
arr.concat(1, 2, [3, 4]); // ['A', 'B', 'C', 1, 2, 3, 4]
```

join()

join() 方法是一个非常实用的方法，它把当前 Array 的每个元素都用指定的字符串连接起来，然后返回连接后的字符串：

```
var arr = ['A', 'B', 'C', 1, 2, 3];
arr.join('-'); // 'A-B-C-1-2-3'
```

map()

map() 函数对数组的每个元素调用定义的回调函数并返回包含结果的数组。举个例子，比如对数组所有的元素都加 1，修改小程序项目中的 index.js，代码如下：

```
Page({
  onLoad: function () {
      var numArr=[1,2,3,4,5];
      var newArr=numArr.map(numberAdd);
      console.log(newArr);//输出2,3,4,5,6
  }
})
//函数
function numberAdd(n){
    return n+1;
}
```

我们通过fuction关键字定义了一个函数（后面还会具体介绍函数用法），当数组调用map()函数时，数组中有几个元素就会调用几次 numberAdd 函数，把当前数组的具体元素替换numberAdd()函数中的形参 n。

学会map()和join()这两个函数，就基本能看懂微信小程序默认生成的工程中 utils.js 里的代码了。

```
function formatTime(date) {
  var year = date.getFullYear() //获取年份
  var month = date.getMonth() + 1 //获取月份 getMonth()从0开始需要加1
  var day = date.getDate() //获取日期

  var hour = date.getHours() //获取小时
  var minute = date.getMinutes() //获取分钟
  var second = date.getSeconds() //获取秒数

  //格式化数组，在年、月、日之间添加"/" 时分秒之间添加":"
  return [year, month, day].map(formatNumber).join('/') + ' ' + [hour, minute,
      second].map(formatNumber).join(':')
}

function formatNumber(n) {
  n = n.toString()
  return n[1] ? n : '0' + n  //条件运算符，数字小于两位前面补0
}
```

上面的代码如果暂时看不懂也没关系，本书介绍完函数后还会再介绍map()函数和数组的其他高阶函数。

多维数组

如果数组的某个元素又是一个 Array，则可以形成多维数组，例如：

```
var arr = [[1, 2, 3], [400, 500, 600], '-'];
```

下面的代码是把 400 这个元素取出来：

```
arr[1][0];
```

对象

JavaScript 的对象是一组由键-值组成的无序集合，一般由花括号{}包裹，例如：

```
var person = {
    name: 'Bob',
    age: 20,
    tags: ['js', 'web', 'mobile'],
    city: 'Beijing',
    hasCar: true,
    zipcode: null
};
```

JavaScript 对象的键都是字符串类型，值可以是任意数据类型。上述 person 对象一共定义了 6 个键值对，其中每个键又被称为对象的属性，例如，person 的 name 属性为'Bob'，zipcode 属性为 null。

要获取一个对象的属性，可以用对象变量.属性名的方式，访问一个不存在的属性也不会报错，而是返回undefined。

```
person.name; // 'Bob'
person.zipcode; // null
person.sex; //undefined
```

也可以通过对象变量 ["属性名"] 方式访问对象，如：

```
person["name"]; //'Bob'
```

细心的读者可能会发现，在微信小程序的 app.js 文件中，app() 方法里面的参数就是一个{}包裹的对象。在每个页面的 JS 文件的 page() 方法里面也是一个对象。

操作对象

由于 JavaScript 的对象是动态类型，所以可以自由地给一个对象添加或删除属性：

```
var xiaoming = {
    name: '小明'
};
xiaoming.age; // undefined
xiaoming.age = 18; // 新增一个age属性
xiaoming.age; // 18
delete xiaoming.age; // 删除age属性
xiaoming.age; // undefined
delete xiaoming.name; // 删除name属性
xiaoming.name; // undefined
delete xiaoming.school; // 删除一个不存在的school属性也不会报错
```

如果要检测xiaoming是否拥有某一属性，可以用 in 操作符：

```
var xiaoming = {
   name: '小明',
};
console.log('name' in xiaoming);//输出true
```

不过要小心，如果 in 判断一个属性存在，那么这个属性不一定是xiaoming的，它可能是 xiaoming 继承得到的，任何对象都会继承object。toString 就是定义在object对象中的。

```
'toString' in xiaoming; // true
```

要判断一个属性是否是 xiaoming 自身拥有的，而不是继承得到的，可以用 hasOwnProperty() 方法：

```
var xiaoming = {
    name: '小明'
};
xiaoming.hasOwnProperty('name'); // true
xiaoming.hasOwnProperty('toString'); // false
```

3.2.6 运算符

JavaScript 常见的运算符分为以下几种。

(1) 算术运算符

(2) 赋值运算符

(3) 字符串链接符

(4) 比较运算符

(5) 逻辑运算符

(6) 条件运算符

算术运算符

算术运算符主要包括加（+）、减（-）、乘（*）、除（/）、求余（%），在前面介绍数字数据类型的时候已经介绍过了。

除了这些运算符，还有累加符（++）和递减符（--）。

假设y=5，则：

```
x=++y  //结果x为6;
x=--y  //结果x为4;
```

赋值运算符

赋值运算符用于给 JavaScript 的变量赋值，核心符号就是等号=。表 3-1 解释了各个赋值运算符的作用（假设 x=10, y=5）。

表 3-1

运算符	例子	等价于	结果
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

字符串链接符

当+作用于字符串时，就变成了字符串的连接符，而不是数学中的加号了。+运算符用于把文本值或字符串变量加起来（连接起来）。

如果需要把两个或多个字符串变量连接起来，则应使用 +运算符。

```
txt1="hello";
txt2="world";
txt3=txt1+" "+txt2; //连接了txt1,空格,txt2
```

在执行以上语句后，变量 txt3 包含的值是 "hello world"。

比较运算符

当我们对 Number 做比较时，可以通过比较运算符得到一个布尔值：

```
2 > 5; // false
5 >= 2; // true
7 == 7; // true
```

>和<与数学符号一样。一个等号（=）是赋值运算，两个等号（==）才是判断是否相等。

实际上，JavaScript 允许对任意数据类型做比较：

```
false == 0; // true
false === 0; // false
```

对于上面的代码，新手可能觉得看不懂，下面解释一下。在 JavaScript 中，有两种比较运算符：

第一种是==比较运算符，它会自动转换数据类型再比较，很多时候会得到非常诡异的结果。

第二种是===比较运算符，它不会自动转换数据类型，如果数据类型不一致，则返回 false，如果一致，则再比较。

由于 JavaScript 存在这个设计缺陷，所以不要使用 == 比较运算符，要使用 === 比较运算符。

另一个例外是 NaN，这个特殊的 Number 与其他值都不相等，包括它自己：

```
NaN === NaN; // false
```

可以通过下面代码测试一下：

```
var a=0/0;
console.log(a); //输出Nan
var b=0/0;
if(a===b){ //不相等
```

```
    console.log("结果相等");//不输出
}
```

唯一能判断 NaN 的方法是通过 isNaN() 函数：

```
isNaN(NaN); // true
```

最后要注意浮点数的相等比较：

```
1 / 3 === (1 - 2 / 3); // false
```

这不是 JavaScript 的设计缺陷。浮点数在运算过程中会产生误差，因为计算机无法精确表示无限循环小数。要比较两个浮点数是否相等，则只能计算它们之差的绝对值，看是否小于某个阈值：

```
Math.abs(1 / 3 - (1 - 2 / 3)) < 0.0000001; // true
```

逻辑运算符

逻辑运算符包括||，&&和!。

&&运算是与运算，只有所有值都为 true，&&运算结果才是 true：

```
true && true; // 这个&&语句计算结果为true
true && false; // 这个&&语句计算结果为false
false && true && false; // 这个&&语句计算结果为false
```

||运算是或运算，只要其中有一个值为 true，||运算结果就是 true：

```
false || false; // 这个||语句计算结果为false
true || false; // 这个||语句计算结果为true
false || true || false; // 这个||语句计算结果为true
```

!运算是非运算，它是一个单目运算符，把 true 变成 false，把 false 变成 true：

```
! true; // 结果为false
! false; // 结果为true
! (2 > 5); // 结果为true
```

条件运算符

JavaScript 还包含了基于某些条件对变量进行赋值的条件运算符。

```
variablename=(condition)?value1:value2
```

例如：x=(y==1)?1:2

当y的值等于1时，x赋值1，否则x赋值2。

3.2.7 条件判断

和 Java 等编程语言类似，JavaScript 使用if...else...来进行条件判断。此语句的执行特点是二选一，即在多个if...else...语句中，如果某个条件成立，则后续就不再继续判断了。例如，根据年龄显示不同内容，可以用 if 语句实现：

```
var age = 20;
if (age >= 18) { // 如果age >= 18为true，则执行if语句块
   console.log("成年了");
} else { // 否则执行else语句块
   console.log("未成年");
}
```

其中else语句是可选的。如果语句块中只包含一条语句，那么可以省略{}：

```
var age = 20;
if (age >= 18) // 如果age >= 18为true，则执行if语句块
   console.log("成年了");
else // 否则执行else语句块
   console.log("未成年");
```

省略{}的危险之处在于，如果后面想添加一些语句，可能会忘了写{}，所以笔者建议，即使语句块中只包含一条语句，也要加上{}。

多条件判断

如果还要更细致地判断条件，则可以使用多个 if...else... 的组合：

```
var age = 3;
if (age >= 18) {
    console.log("成年了");
} else if (age >= 6) {
    console.log("青少年");
} else {
    console.log("儿童");
}
```

if...else... 也可以嵌套，下面的代码和上面的效果是一样的。

```
var age = 3;
if (age >= 18) {
    console.log("成年了");
} else {
```

```
    if (age >= 6) {
        console.log("青少年");
    } else {
        console.log("儿童");
    }
}
```

3.2.8 循环语句

要计算 1+2+3，则可以直接写表达式：

```
1 + 2 + 3; // 6
```

要计算 1+2+3+…+10，则也能勉强写出表达式。

但是，要计算 1+2+3+…+10000，则直接写出表达式就不可能了。

为了让计算机能计算成千上万次的重复运算，我们需要使用循环语句。

JavaScript 循环语句包括以下两种：

(1) for 循环

(2) while 循环

for 循环

下面用for循环计算 1+2+3+…+10000，代码如下：

```
var x = 0;
var i;
for (i=1; i<=10000; i++) {
    x = x + i;
}
x; // 50005000
```

让我们来分析一下for循环的控制条件：

(1) i=1 是初始条件，将变量 i 置为 1；

(2) i<=10000 是判断条件，满足时就继续循环，不满足时就退出循环；

(3) i++ 是每次循环后的递增条件，由于每次循环后变量 i 都会加 1，因此它终将在若干次循环后不满足判断条件 i<=10000 而退出循环。

for 循环最常用的地方是利用索引来遍历数组：

```
var arr = ['Apple', 'Google', 'Microsoft'];
var i, x;
for (i=0; i<arr.length; i++) {
    x = arr[i];
    console.log(x);
}
```

for循环的 3 个条件都是可以省略的，如果没有退出循环的判断条件，就必须使用 break 语句退出循环，否则就是死循环（不要在开发环境中测试死循环，界面容易卡住）：

```
var x = 0;
for (;;) { // 将无限循环下去
    if (x > 100) {
        break; // 通过if判断来退出循环
    }
    x ++;
}
```

当for循环语句内只有一行执行语句时，可以省略{}，但是不建议这么写。比如下面有一个凄美的爱情故事：

一个男孩给女孩写了一行代码：

```
for(;;) console.log("I love You!");
```

女孩回了一行代码：

```
for(;;); console.log("I love You Too!");
```

很凄美。

解释一下代码：在女孩的代码中，for 循环后多了一个 ;，输出语句属于 for 循环之外的语句，后面的内容永远执行不到。

for ... in

for循环的一个变体是for...in循环，它可以把一个对象的所有属性依次循环出来：

```
var o = {
    name: 'Jack',
    age: 20,
    city: 'Beijing'
};
```

```
for (var key in o) {
    console.log(key); // 'name', 'age', 'city'
    console.log(o[key]); // 'Jack' ,20 ,'Beijing'
}
```

由于数组也是特殊的对象，而它的每个元素的索引被视为对象的属性，因此，for ... in 循环可以直接循环出数组的索引：

```
var a = ['A', 'B', 'C'];
for (var i in a) {
    console.log(i); // '0', '1', '2'
    console.log(a[i]); // 'A', 'B', 'C'
}
```

while 循环

for循环在已知循环的初始和结束条件时非常有用。而忽略了条件的for循环容易让人看不清循环的逻辑，此时用while循环更佳。

while循环只有一个判断条件，当条件满足时，就不断循环；当条件不满足时，就退出循环。比如我们要计算 100 以内所有奇数之和，可以用while循环实现：

```
var x = 0;
var n = 99;
while (n > 0) {
    x = x + n;
    n = n - 2;
}
x; // 2500
```

在循环内部变量 n 不断自减，直到变为-1，不再满足 while 条件时，循环退出。

do ... while

do ... while循环类似while循环，它和 while 循环的唯一区别在于，不是在每次循环开始的时候判断条件，而是在每次循环完成的时候判断条件：

```
var n = 0;
do {
    n = n + 1;
} while (n < 100);
n; // 100
```

用do ... while循环要小心，循环体会至少执行1次，而for循环和while循环则可能一次都不执行。

3.2.9 Map和Set

为了满足开发需求，ES6语法新增了Map和Set两种数据类型。

Map

JavaScript的默认对象表示方式{}可以视为Java语言中的Map或Objective-C语言中Dictionary的数据结构，即一组键值对。

但是JavaScript的对象有一个小问题，就是键必须是字符串。但实际上，将数字或者其他数据类型作为键也是非常合理的。

为了解决这个问题，最新的ES6规范引入了新的数据类型Map。

Map是一组键值对的结构，具有极快的查找速度。

举个例子，假设要根据同学的名字查找对应的成绩，如果用数组实现，则需要两个数组，一个数组存放名字，另一个数组存放成绩。如果给定一个名字，要查找对应的成绩，就要先在存放名字的数组中找到对应的位置，再从存放成绩的数组中取出对应的成绩，数组越长，耗时越长。

如果用Map实现，则只需要一个“名字”-“成绩”的对照表，直接根据名字查找成绩，无论这个表有多大，查找速度都不会变慢。用JavaScript写一个Map如下：

```
var m = new Map([['小明', 95], ['张三', 75], ['李四', 85]]);
console.log(m.get('小明')); // 输出 95
```

初始化Map需要一个二维数组，或者直接初始化一个空Map。Map具有以下方法：

```
var m = new Map(); // 空Map
m.set('张三', 67); // 添加新的key-value
m.set('李四', 59);
m.has('张三'); // 是否存在key '张三': true
m.get('张三'); // 67
m.delete('张三'); // 删除key '张三'
m.get('张三'); // undefined
```

由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉：

```
var m = new Map();
m.set('张三', 67);
m.set('张三', 88);
m.get('张三'); // 88
```

Set

Set和Map类似，也是一组key的集合，但不存储value。由于key不能重复，所以，在Set中，没有重复的key。可以把Set简单理解为没有重复元素的数组。

要创建一个Set，需要提供一个数组作为输入，或者直接创建一个空Set：

```
var s1 = new Set(); // 空Set
var s2 = new Set([1, 2, 3]); // 含1, 2, 3
```

重复元素在Set中自动被过滤：

```
var s = new Set([1, 2, 3, 3, '3']);
s; // Set {1, 2, 3, "3"}
```

注意，数字3和字符串'3'是不同的元素。

通过add(key)方法可以添加元素到Set中，可以重复添加，但不会有效果，使用delete(key)方法可以删除元素。

```
var s=new Set();
s.add(1);
s.add(1);
console.log(s); //输出 Set {1}
s.delete(1);
console.log(s); //输出 Set {}
s.delete(2);  //没有相应的key也不会报错
```

iterable

遍历数组可以采用下标循环，而遍历Map和Set就无法使用下标。为了统一集合类型，ES6标准引入了新的iterable类型，数组、Map和Set都属于iterable类型。

具有iterable类型的集合可以通过新的for...of循环来遍历。

for...of循环是ES6引入的新的语法，例如：

```
var m = new Map([['小明', 95], ['张三', 75], ['李四', 85]]);
for (var x of m) {
```

```
    //依次输出 ["小明", 95] , ["张三", 75] ,["李四", 85]
    console.log(x);
}
```

3.3 JavaScript 函数

当代码出现有规律的重复时，就要当心了。这时候可以把重复的代码定义成一个函数，这样写一份代码就可以调用多次了。

函数就是最基本的一种代码抽象的方式。

3.3.1 函数定义和调用

首先学习一下如何定义函数和调用函数。

定义函数

在 JavaScript 中，定义函数用function关键字，主要方式如下：

```
function abs(x) {
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
}
```

上述 abs() 函数的定义如下：

- function指出这是一个函数定义；
- abs是函数的名称；
- (x)括号内列出函数的参数，多个参数以“,”分隔；
- { ... }之间的代码是函数体，可以包含若干语句，甚至可以没有任何语句。

请注意，函数体内部的语句在执行时，一旦执行到return语句时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有return语句，则函数执行完毕后也会返回结果，只是结果为undefined。

JavaScript 的函数其实也是一个特殊对象，上述定义的abs()函数实际上是一个函数对象，而函数名abs可以被视为指向该函数的变量。

因此，第二种定义函数的方式如下：

```
var abs = function (x) {
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
};
```

在这种方式下，function (x) { ... }是一个匿名函数，它没有函数名。但是，这个匿名函数赋值给了变量 abs，所以，通过变量abs就可以调用该函数。

上述两种定义完全等价，注意，第二种方式按照完整语法规则需要在函数体末尾加一个“;”，表示赋值语句结束。

调用函数

在调用函数时，按顺序传入参数即可。下面修改 index.js：

```
//index.js
Page({

  onLoad: function () {
    console.log(abs(-9)); //输出9
    console.log(abs(10)); //输出10
  }
})
//定义函数
var abs = function (x) {
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
};
```

由于 JavaScript 允许传入任意个参数而不影响调用，因此传入的参数比定义的参数多也没有问题，虽然函数内部并不需要这些参数：

```
abs(-9, 'haha'); // 返回9
abs(10, 'hehe', null); // 返回10
```

传入的参数比定义的少也没有问题：

```
console.log(abs()); //输出NaN
```

此时，abs(x) 函数的参数 x 将收到 undefined，计算结果为 NaN。

要避免收到 undefined，可以对参数进行检查，typeof是判断数据类型的关键字，throw是抛出异常关键字：

```
function abs(x) {
    if (typeof x !== 'number') {
        throw 'Not a number';//抛出异常,终止运行
    }
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
}
```

我们还可以把在一个文件中定义的函数暴露出去，让别的文件可以引入调用。比如微信小程序默认生成的项目utils.js：

```
//暴露format方法
module.exports = {
  formatTime: formatTime
}
```

在logs.js中导入和使用：

```
//导入util
var util = require('../../utils/util.js')

//使用
utils.formatTime(...)
```

其中导入还可以换一种写法：

```
import util from '../../utils/util';
```

arguments 参数

JavaScript 还有一个免费赠送的关键字arguments，它只在函数内部起作用，并且永远指向当前函数的调用者传入的所有参数。arguments 类似数组，但它不是一个数组：

```
//index.js
Page({
  onLoad: function () {
    foo("haha", "hehe", "heihei");
  }
})

function foo(x) {
    for (var i=0; i<arguments.length; i++) {
        //依次输出// haha, hehe, heihei
        console.log(arguments[i]);
    }
}
```

利用 arguments，可以获得调用者传入的所有参数。也就是说，即使函数不定义任何形式参数，还是可以拿到参数的值：

```
function abs() {
    if (arguments.length === 0) {
        return 0;
    }
    var x = arguments[0];
    return x >= 0 ? x : -x;
}
abs(); // 0
abs(10); // 10
abs(-9); // 9
```

实际上，arguments常用于判断传入参数的个数。你可能会看到这样的写法：

```
// foo(a[, b], c)
// 接收2~3个参数，b是可选参数，如果只传两个参数，则b默认为null：
function foo(a, b, c) {
    if (arguments.length === 2) {
        // 实际拿到的参数是a和b，c为undefined
        c = b; // 把b赋给c
        b = null; // b变为默认值
    }
    // ...
}
```

要把中间的参数b变为“可选”参数，就只能通过arguments判断，然后重新调整参数并赋值。

rest 参数

ES6 标准引入了rest参数，多余的参数以数组形式交给变量rest，所以，我们不再需要arguments就获取了全部参数。

rest参数只能写在最后，前面用...标识，如果传入的参数连正常定义的参数都没填满，那么也不要紧，rest参数会接收一个空数组（注意不是undefined）。

示例代码:

```
//index.js
Page({
  onLoad: function () {
    foo("haha", "hehe", "heihei");
  }
})

function foo(x, ...rest) {
    console.log(x); //输出haha
    console.log(rest);//输出["hehe", "heihei"]
}
```

常见问题

JavaScript 引擎有一个在行末自动添加分号的机制，这本来是一件好事儿，但是这可能让你栽到 return 语句的一个大“坑”里：

```
//index.js
Page({
  onLoad: function () {
    console.log(test());//输出Object {name: "test"}
  }
})

function test() {
    return { name: 'test' };
}
```

上面的代码没有问题，但是如果把 return 语句拆分成两行:

```
function test() {
    return
        { name: 'test' };
}
```

那么这时候就出问题了，由于 JavaScript 引擎具有在行末自动添加分号的机制，上面的代码实际上变成了：

```
function test() {
    return;//自动添加了分号，相当于return undefined;
        { name: 'test' };// 这行语句已经没法执行到了
}
```

正确的多行语句的写法是：

```
function test() {
    return { // 这里不会自动加分号，因为{表示语句尚未结束
        name: 'test'
    };
}
```

3.3.2 变量作用域

在 JavaScript 中，用 var 申明的变量实际上是有作用域的。

如果一个变量在函数体内部声明，则该变量的作用域为整个函数体，在函数体外不可引用该变量：

```
function foo() {
    var x = 1;
    x = x + 1;
}

x = x + 2; // ReferenceError! 无法在函数体外引用变量x
```

如果两个不同的函数各自声明了同一个变量，那么该变量只在各自的函数体内起作用。换句话说，不同函数内部的同名变量互相独立，互不影响：

```
function foo() {
    var x = 1;
    x = x + 1;
}
```

```
function bar() {
    var x = 'A';
    x = x + 'B';
}
```

由于JavaScript的函数可以嵌套，此时，内部函数可以访问外部函数定义的变量，反过来则不行：

```
function foo() {
    var x = 1;
    function bar() {
        var y = x + 1; // bar可以访问foo的变量x!
    }
    var z = y + 1; // ReferenceError! foo不可以访问bar的变量y!
}
```

JavaScript的函数在查找变量时从自身函数定义开始，从“内”向“外”查找。如果内部函数定义了与外部函数重名的变量，则内部函数的变量将“屏蔽”外部函数的变量。

不在任何函数内定义的变量具有全局作用域，在当前文件中都能使用。

3.3.3 方法

在一个对象中绑定函数，可以把函数称为这个对象的方法。下面的代码给zhangsan对象绑定了一个方法age()，即返回zhangsan的年龄。

```
//index.js
Page({
  onLoad: function () {
    console.log(zhangsan.age);//输出方法 function age(){ return 18 }
    console.log(zhangsan.age()); //输出 18
  }
})

var zhangsan = {
    age: function () {
        return 18;
    }
};
```

一定要注意，当调用zhangsan.age时，其实得到的是该方法，后面加上括号，即zhangsan.age()，才是得到方法的结果。

在微信小程序中，app() 和page() 方法接收的参数都是对象，对象中定义了很多生命周期的方法，只不过这些方法是小程序框架调用的，不需要我们调用。

绑定到对象上的函数被称为方法，和普通函数没什么区别，但是它在内部使用了一个 this 关键字。

在一个方法内部，this 是一个特殊变量，它始终指向当前对象。往往新手在使用它时也会出现一些问题。来看下面的代码：

```
//index.js
Page({

  onLoad: function () {
    console.log(lisi.age()); // 27, 正常结果
    console.log(getAge());//报错
  }
})

function getAge() {
    //写书时年份是2017年
    var y = new Date().getFullYear();
    return y - this.birth;
}

var lisi = {
    name: 'lisi',
    birth: 1990,
    age: getAge
};
```

lisi对象age属性绑定了getAge函数，当调用lisi.age()时，getAge函数中的this关键字代表lisi对象。this.birth的结果是 1990。

如果像上面的代码一样，直接调用getAge()，此时，该函数的 this 指向的是Page()接收的对象，并没有birth属性。

JavaScript 的函数内部如果调用了 this，那么这个 this 到底指向谁？答案是，视情况而定！

不过，还是可以通过apply()函数来控制this的指向，如下：

```
console.log(getAge.apply(lisi,[]));//27 正常结果
```

apply()函数接收两个参数，第一个参数就是需要绑定的 this 变量，第二个参数是 Array，表示函数本身的参数。

3.3.4 高阶函数

函数可以接收任意类型的参数，如果一个函数接收另一个函数作为参数，那么这种函数就被称为高阶函数。

常用的高阶函数有：

(1) map 函数

(2) reduce 函数

(3) filter 函数

(4) sort 函数

map 函数

在前面介绍数组的时候提到过map()函数。map()方法定义在 JavaScript 的数组中，我们调用数组的map()方法，传入自己的函数，就得到了一个新的数组作为结果：

```
function pow(x) {
    return x * x;
}

var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
arr.map(pow); // [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

reduce 函数

reduce()也是数组使用的函数。数组的reduce()把一个函数作用在这个数组的元素上，这个函数必须接收两个参数，reduce()把结果继续和序列的下一个元素做累积计算，其效果就是：

```
[x1, x2, x3, x4].reduce(f) = f(f(f(x1, x2), x3), x4)
```

例如对一个数组求和，就可以用reduce实现：

```
var arr = [1, 3, 5, 7, 9];
arr.reduce(function (x, y) {
    return x + y;
}); // 25
```

filter 函数

filter也是一个常用的操作，它用于把数组的某些元素过滤，然后返回剩下的元素，和map()类似，数组的filter()也接收一个函数。

filter()把传入的函数依次作用于每个元素，然后根据返回值是true还是false决定是保留还是丢弃该元素。

例如在一个数组中，删掉偶数只保留奇数，可以这么写：

```
var arr = [1, 2, 4, 5, 6, 9, 10, 15];
var r = arr.filter(function (x) {
    return x % 2 !== 0;
});
r; // [1, 5, 9, 15]
```

sort 函数

前面介绍数组的时候讲过sort()函数，其实sort()函数也是一个高阶函数，接收一个函数作为参数，函数的作用就是指定排序规则。比如按照数字大小排序，可以这么写：

```
var arr = [10, 20, 1, 2];
arr.sort(function (x, y) {
    if (x < y) {
        return -1;
    }
    if (x > y) {
        return 1;
    }
    return 0;
}); // [1, 2, 10, 20]
```

3.3.5 箭头函数

ES6标准新增了一种新的函数：Arrow Function（箭头函数）。为什么叫Arrow Function？因为它的定义用的就是一个箭头：

```
x => x * x
```

上面的箭头函数相当于：

```
function (x) {
    return x * x;
}
```

3.4 JavaScript 标准对象

在 JavaScript 语言中，你可以认为一切都是对象，我们通常用typeof操作符获取对象类型，它总是返回一个字符串：

```
typeof 123; // 'number'
typeof NaN; // 'number'
typeof 'str'; // 'string'
typeof true; // 'boolean'
typeof undefined; // 'undefined'
typeof Math.abs; // 'function'
typeof null; // 'object'
typeof []; // 'object'
typeof {}; // 'object'
```

可见，number、string、boolean、function和undefined有别于其他对象类型。特别要注意，null的对象类型是object，Array的对象类型也是object，如果我们用typeof，将无法区分出null、Array和通常意义上的object——{}。

JavaScript 本身还提供了常用的对象，方便我们开发：

(1) Date

(2) RegExp

(3) JSON

(4) Math

3.4.1 Date

Date 对象用于处理日期和时间。创建 Date 对象的语法：

```
var myDate=new Date()
```

如果要创建一个指定日期和时间的 Date 对象，则可以用：

```
// 传入2017年 1月 9日 8点 15分 30秒 100毫秒
var d = new Date(2017, 0, 9, 8, 15, 30,100);
console.log(d);//输出 Mon Jan 09 2017 08:15:30 GMT+0800 (CST)
```

可以看到，我们传的参数除了月份都比较合理，而月份是从 0 开始计数，也就是范围是 0~11，0 表示 1 月，这一点要注意一下。

还可以传入距离 1970 年 1 月 1 日 0 点的毫秒值。

```
var d = new Date(1483920930100);
console.log(d); //输出Mon Jan 09 2017 08:15:30 GMT+0800 (CST)
```

Date 的对象方法有很多，常用的有下面这些:

```
var now = new Date();
now.getFullYear(); // 获取年
now.getMonth(); //获取月份,0表示1 范围为0~11
now.getDate(); // 获取日期，比如9号
now.getDay(); // 获取星期数，如星期三，返回3
now.getHours(); //获取小时数，24小时制
now.getMinutes(); // 获取分钟数
now.getSeconds(); // 获取秒数
now.getMilliseconds(); // 获取秒
now.getTime(); // 以number形式表示的时间戳，如:1483920930100
```

3.4.2 RegExp

字符串是编程时涉及最多的一种数据结构，对字符串进行操作的需求几乎无处不在。

比如在开发中经常需要判断手机号或者邮箱等字符串是否合法。最简单的方法就是用正则表达式进行判断。

正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言给字符串定义一个规则，凡是符合规则的字符串，就认为它“匹配”了，否则该字符串就是不合法的。

在正则表达式中，如果直接给出字符，就是精确匹配。用\d可以匹配一个数字，\w可以匹配一个字母或数字，所以：

- '00\d'可以匹配'007'，但无法匹配'00A';
- '\d\d\d'可以匹配'010';
- '\w\w'可以匹配'JS'。

还有很多匹配规则，本书就不一一列举了，读者可以查阅文档。http://www.w3school.com.cn/jsref/jsref_obj_regexp.asp。

有了准备知识，我们就可以在 JavaScript 中使用正则表达式了。

JavaScript 有两种方式创建一个正则表达式：

- 第一种方式是直接通过/正则表达式/写出来

- 第二种方式是通过new RegExp('正则表达式')创建一个RegExp对象。

这两种写法是一样的：

```
var re1 = /ABC\-001/;
var re2 = new RegExp('ABC\\-001');

re1; // /ABC\-001/
re2; // /ABC\-001/
```

注意，如果使用第二种写法，因为字符串的转义问题，则字符串的两个\\实际上是一个\。

RegExp对象的test()方法用于测试给定的字符串是否符合条件，下面是判断正则表达式是否匹配：

```
var re = /^\d{3}\-\d{3,8}$/;
re.test('010-12345'); // true
re.test('010-1234x'); // false
re.test('010 12345'); // false
```

3.4.3 JSON

JSON，即 JavaScript Object Notation，它是一种数据交换格式。我们在进行网络请求传输数据的时候，基本上都用 JSON 传输了。

JSON 实际上是 JavaScript 的一个子集。在 JSON 中，一共就以下几种数据类型：

- number：和 JavaScript 的 number 完全一致；
- boolean：就是 JavaScript 的true或false；
- string：就是 JavaScript 的string；
- null：就是 JavaScript 的null；
- array：就是 JavaScript 的 Array 表示方式——[]；
- object：就是 JavaScript 的{ ... }表示方式。

以及上面的任意组合。

并且，JSON 还定死了字符集必须是 UTF-8，表示多语言就没有问题了。为了统一解析，JSON 的字符串规定必须用双引号（""），Object 的键也必须用双引号（""）。

将任何 JavaScript 对象变成 JSON，就是把这个对象序列化成一个 JSON 格式的字符串，这样才能够通过网络传递给其他计算机。

为了方便操作 JSON 格式化数据，JavaScript 提供了JSON对象，主要使用两个方法:

(1) stringify() 把 JavaScript 对象转换成 JSON 格式的字符串。

(2) parse() 把 JSON 格式的字符串转换成 JavaScript 对象。

```
var zhangsan = {
    name: '张三',
    age: 18
};
JSON.stringify(xiaoming); //// {"name":"张三","age":18}

//输出 // Object {name: '张三', age: 18}
JSON.parse('{"name":"张三","age":18}');
```

3.4.4 Math

Math 对象用于执行数学任务。Math 对象不像Date对象那样需要new，而是直接把Math作为对象使用其中的方法，如:

```
var a=-10
Math.abs(a);//取绝对值, 10
```

Math 对象中的方法有很多，如表 3-2 所示。

表 3-2

方 法	描 述
abs(x)	返回数值的绝对值
acos(x)	返回数值的反余弦值
asin(x)	返回数值的反正弦值
atan(x)	以处于 -PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值
atan2(y,x)	返回从 X 轴到点 (x,y) 的角度（处于 --PI/2 与 PI/2 弧度之间）
ceil(x)	对数值进行上舍入
cos(x)	返回数值的余弦
exp(x)	返回 e 的指数
floor(x)	对数值进行下舍入
log(x)	返回数值的自然对数（底为 e）
max(x,y)	返回 x 和 y 中的最高值
min(x,y)	返回 x 和 y 中的最低值

续表

方　法	描　述
pow(x,y)	返回 x 的 y 次幂
random()	返回 0 ~ 1 的随机数
round(x)	把数值四舍五入为最接近的整数
sin(x)	返回数值的正弦
sqrt(x)	返回数值的平方根
tan(x)	返回角的正切
toSource()	返回该对象的源代码
valueOf()	返回 Math 对象的原始值

3.5 本章小结

本章主要介绍了 JavaScript 语法，包括数据类型、运算符、函数、标准对象等。JavaScript 语法应用范围非常广，除了微信小程序，还可以做网页等开发。微信小程序逻辑功能都是用 JavaScript 开发的，了解 JavaScript 可以更快速地上手微信小程序开发。

4 熟练掌握WXML和HTML

如果你做过网页开发，那么应该了解 HTML 和 CSS 语言。一般我们用 HTML 构建布局及 CSS 定义样式。微信小程序框架也提供了类似的语言——WXML 和 WXSS。

WXML(WeiXin Markup Language) 是框架设计的一套类似 HTML 的标签语言，结合基础组件、事件系统可以构建出页面的结构，即.wxml 文件。

WXSS(WeiXin Style Sheets) 是一套非常类似 CSS 的样式语言，用于描述 WXML 的组件样式。它将决定 WXML 的组件应该怎么显示，即.wxss 文件。

本章会详细介绍 WXML 和 WXSS。

4.1 WXML和HTML的差异

.wxml文件和.html文件的注释方式一样，都是通过下面的方式注释:

```
<!--注释-->
```

一般建议在.wxml文件开头注明页面名，写上<! --页面名.wxml-->，如index页面的.wxml文件 (index.wxml):

```
<!--index.wxml-->
<view class="container">
  <view  bindtap="bindViewTap" class="userinfo">
```

```
    <image class="userinfo-avatar" src="{{userInfo.avatarUrl}}" background-size="
        cover"></image>
    <text class="userinfo-nickname">{{userInfo.nickName}}</text>
  </view>
  <view class="usermotto">
    <text class="user-motto">{{motto}}</text>
  </view>
</view>
```

在上面的代码中，view、image、text都是 WXML 提供的标签。WXML 与 HTML 最大的差异就是标签。虽然目前 WXML 提供的标签相对于 HTML 来说很有限，但是 HTML 的大部分标签可以用 WXML 标签替代，从而满足各种 UI 需求。如表 4-1 为 WXML 于 HTML 标签的对比。

表 4-1 WXML 于 HTML 标签对比

WXML	HTML
view	article, aside, body, div, ul, li, caption, dd, dl, dt, footer, header, nav, section, table, thead, tbody, tr, td, th, ol, h1, h2, h3, h4, h5, h6, p, em
text	span
navigator	a
image	img
input	input[type='text']
checkbox	input[type='checkbox']
radio	input[type='radio']
button	button
form	form
icon	.ico 后缀的特殊图片格式以 favicon.ico 放在 web 根目录下或者 <link> 中使用

从表 4-1 可以看到：

(1) WXML 中的<view>标签替代了 HTML 里面的大部分标签。

(2) <text>标签是唯一一个可以长按选中文本的标签，而且 text 标签不允许于其他标签相互嵌套。

(3) <input>,checkbox,<radio>标签其实是拆分了 HTML 中的 input 标签。

WXML 除提供基础的标签外，还提供了一套标准化的组件，开发人员可以通过标准化的组件来实现一些 HTML 上比较复杂的功能，例如：可滚动视图区域、滑块视图容器、进度条等。这些在第 7 章会详细介绍。

4.2 WXML 语法

对比 HTML 语法，WXML 语法除了标签不一样，渲染界面的方式也存在差异。WXML 具有下面几种能力：

(1) 数据绑定

(2) 条件渲染

(3) 列表渲染

(4) 模板

(5) 事件绑定

(6) 引用

4.2.1 数据绑定

微信小程序是通过状态模式进行数据绑定的。状态模式定义一个对象，通过管理其状态的变化，从而让界面做出相应的变化。

简单地讲，只要对象状态发生变化，就通知页面更新视图元素。通过以下三个步骤可以实现：

- 识别哪个 UI 元素被绑定了相应的对象。
- 监视对象状态的变化。
- 将所有变化传播到绑定的视图上。

如图 4-1 所示，.wxml 文件中的动态数据均来自对应页面的.js文件中Page的data对象。一旦 data 对象中的数据发生变化，界面就会发生相应的变化。

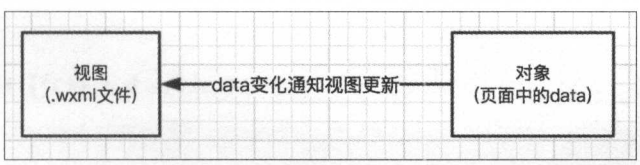

图 4-1 状态模式更新界面

数据绑定使用 Mustache 语法（双大括号）将变量包起来，例如：

```
//index.js
Page({
  data: {
    message: 'Hello 微信小程序'
  }
})
```

```
<!--index.wxml-->
<view> {{ message }} </view>
```

运行结果如图 4-2 所示。

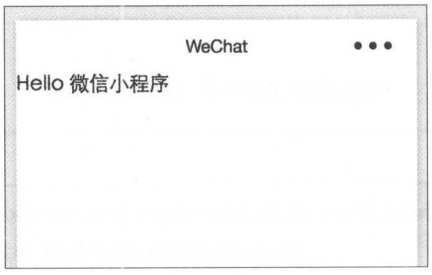

图 4-2 运行结果

简单绑定

绑定数据可以作用于三个位置——内容、组件属性和控制属性，而且还可以处理一些关键字。

内容 (见图 4-2):

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

组件属性 (需要在双引号之内):

```
<!--index.xml-->
<view id="item-{{id}}"> </view>
```

```
//index.js
Page({
  data: {
    id: 0
  }
})
```

如果data对象不再变化，那么效果相当于view组件定义了属性id=item-0。

控制属性 (需要在双引号之内):

```
<!--index.xml-->
<view wx:if="{{condition}}"> </view>
```

```
//index.js
Page({
  data: {
    condition: true
  }
})
```

wx:if是条件渲染用到的语法，后面会介绍到。

关键字: 在.wxml 文件中常用的关键字有true和false。

- true：boolean 类型的 true，代表真值。
- false：boolean 类型的 false，代表假值。

涉及true和false都需要用{{...}}包裹。

```
<checkbox checked="{{false}}"> </checkbox>
```

虽然checkbox组件还没有介绍到，但是上面的代码不难看懂。

值得注意的是:不要直接写 checked="false",其计算结果是一个字符串,转成 boolean 类型后代表真值。

运算

可以在{{...}}内进行简单的运算，支持下面几种方式。

三元运算:

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

算数运算:

```
<!--index.wxml-->
<view> {{a + b}} + {{c}} + d </view>
```

```
//index.js
Page({
  data: {
```

```
    a: 1,
    b: 2,
    c: 3
  }
})
```

view中的内容为 3+3+d。

逻辑判断:

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

字符串运算:

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

```
Page({
  data:{
    name: '微信小程序'
  }
})
```

运行后显示内容：hello 微信小程序。

数据路径运算:

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

```
Page({
  data: {
    object: {
      key: 'Hello '
    },
    array: ['小程序']
  }
})
```

运行后显示内容：Hello 小程序。

组合

也可以在 Mustache (双大括号如{{ }}) 内直接进行组合，构成新的对象或者数组。

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

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

wx:for 是微信小程序列表渲染常见的写法，后面介绍列表渲染时会涉及。在这个位置我们组合成了一个新的数组。最终组合成的数组为[0,1,2,3,4]：

```
<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}。

也可以用扩展运算符 ... 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，那么也可以间接地表达。

```
<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'}。

注意：上述方式可以随意组合，但是如果存在变量名相同的情况，则后边的变量名会覆盖前面，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}。

4.2.2 条件渲染

条件语句可用于.wxml 中的条件渲染，不同的条件可以进行不同的渲染。

wx:if

在框架中，我们用 wx:if="{{condition}}" 来判断是否需要渲染该组件，条件语句为true的时候，所在组件才会被渲染，写法如下：

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

也可以用 wx:elif 和 wx:else 来添加一个 else 块，elif相当于 js 语法中的else if：

```
<!--index.wxml-->
<view wx:if="{{length > 5}}"> 1 </view>
<view wx:elif="{{length > 2}}"> 2 </view>
<view wx:else> 3 </view>
```

```
//index.js
Page({
  data: {
    length:4
  }
})
```

当 length 为 4 的时候只有第二个view组件被渲染了，显示内容为 2。

block wx:if

因为 wx:if 是一个控制属性，需要将它添加到一个标签上。但是如果想一次性判断多个组件标签，则可以使用一个 <block/> 标签将多个组件包装起来，并在上面使用 wx:if 控制属性。

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

注意：<block/> 并不是一个组件，它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。

wx:if vs hidden

除了通过条件渲染控制是否显示和隐藏组件，还可以通过View的属性hidden控制，如下：

```
<view hidden="{{true}}"> {{message }} </view>
```

hidden属性默认值是true，下面的写法效果是一样的：

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

wx:if和hidden有什么区别呢？

因为 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，框架有一个局部渲染的过程，因为它会确保条件块在切换时销毁或重新渲染。

同时 wx:if 也是惰性的，如果初始渲染条件为 false，那么框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下，hidden 就简单多了，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，在需要频繁切换的情景下，用 hidden 更好，如果在运行时条件不大可能改变，则用 wx:if 较好。

4.2.3 列表渲染

列表是常见的样式（如果你接触过 Android 或者 iOS 开发），列表可以用 ListView 或者 TableView 实现。

相比之下，微信小程序实现的方式更加直观，更容易理解。.wxml 文件可以使用列表语句将列表中的各项数据进行重复渲染。语法包括wx:for和wx:key。

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。

默认数组的当前项的下标变量名默认为index，数组当前项的变量名默认为item:

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

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

运行结果如图 4-3 所示。

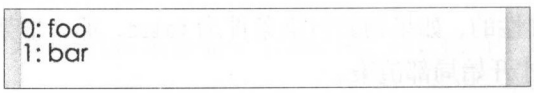

图 4-3 运行结果

如果不想使用默认的index和item，则可以使用 wx:for-item 指定数组当前元素的变量名，使用 wx:for-index 指定数组当前下标的变量名：

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

wx:for也可以嵌套，下面是一个九九乘法表：

```
<view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" 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>
```

block wx:for 类似block wx:if，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>
```

默认生成的 Quick Start 项目中的 log.wxml 就使用了列表渲染：

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

wx:key

wx:for有一个局限性：在每次数据改变的时候就需要重新渲染，并没有保存之前每个条目的状态。

如果列表中项目的位置会动态改变或者有新的项目被添加到列表中，并且希望列表中的项目保持自己的特征和状态（如<switch/>的选中状态），就需要使用 wx:key 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供：

(1) 字符串，代表在for循环的array中item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 for 循环中的条目本身，这种表示需要条目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，会校正带有key的组件，框架会确保它们被重新排序，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。

如果不提供 wx:key，则控制台会报警告，如果明确知道该列表是静态，或者不必关注其顺序，则可以选择忽略。

示例代码:

```
<!--index.wxml-->
<switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;"> {{item.id
    }}</switch>
<button bindtap="switch"> Switch </button>
```

这里为了方便演示，使用了switch组件，这是一个滑动开关的组件，style="display: block;" 表示会在控件后自动加换行。

后面还使用了button按钮组件，通过bindtap="switch"绑定了点击事件。当button按钮组件被点击了，就会调用index.js文件中的switch()函数。

```
//index.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'},
    ]
  },
  switch: function(e) {
    //获取数组的长度
    const length = this.data.objectArray.length
    //随机交换顺序
```

```
    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
    }
    //修改data数组 刷新界面
    this.setData({
      objectArray: this.data.objectArray
    })
  }
})
```

当点击按钮时就会随机排列switch组件的顺序，程序会自动保存switch组件的选中状态，运行结果如图 4-4 所示。

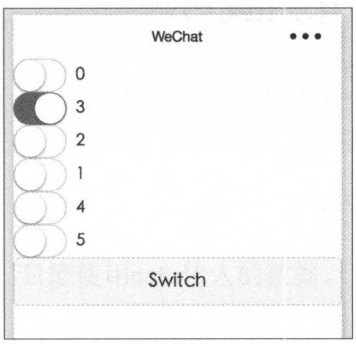

图 4-4 运行结果

4.2.4 模板

WXML 提供了模板（template），可以在模板中定义代码片段，然后在不同的地方调用。

定义模板

使用 name 属性作为模板的名字，然后在<template/>内定义代码片段，如：

```
<!--
  index: int
  msg: string
```

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

最好在模板上面添加注释，说明一下模板需要什么参数。

使用模板

使用is属性，声明需要使用的模板，然后将模板所需要的data传入，下面是完整代码：

```
<!--index.wxml-->
<template is="msgItem" data="{{...item}}"/>

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

```
//index.js
Page({
  data: {
    item: {
      index: 0,
      msg: '这是一个模板',
      time: '2017-06-01'
    }
  }
})
```

运行结果如图 4-5 所示。

图 4-5 运行结果

is 属性可以使用 Mustache 语法来动态决定具体需要渲染哪个模板：

```
<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>
```

模板的作用域

模板拥有自己的作用域，只能使用data传入的数据。

4.2.5 事件绑定

如图 4-6 所示，可以通过data把数据从逻辑层传入到视图层，视图层可以通过事件和逻辑层进行通信，从而改变data中的数据。

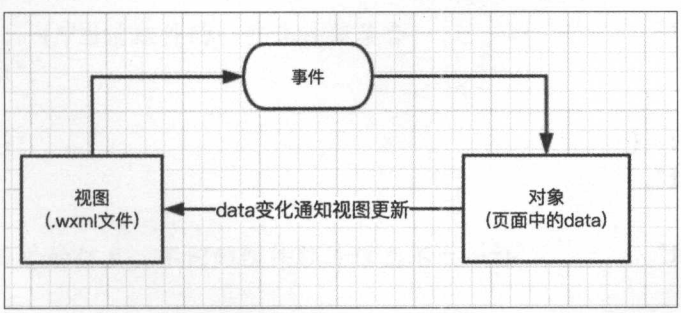

图 4-6 逻辑层和视图层通信

事件可以将用户的行为反馈到逻辑层进行处理。一般绑定在组件上，当达到触发事件时，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 和 touches。

事件的使用方式

首先需要在组件中绑定一个事件处理函数。常用的如bindtap，当用户点击该组件时，会在该页面对应的 Page 中找到相应的事件处理函数。

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

然后在相应的 Page 定义中写上相应的事件处理函数，参数是 event。

```
Page({
  tapName: function(event) {
    //输出日志
    console.log(event)
  }
})
```

事件对象

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。比如点击组件的时候可以看到日志信息大致如下：

```
{
"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
}]
}
```

下面就剖析一下事件对象，对上面的代码加上注释：

```
{
"type":"tap",  //事件类型
"timeStamp":895, //页面打开到触发事件所经过的毫秒数。
"target": {  //触发事件的源组件。
  "id": "tapTest", //事件源组件的id
  "dataset":  {  //事件源组件上由data-开头的自定义属性组成的集合
    "hi":"WeChat"
  }
},
"currentTarget":  {//当前组件的一些属性值集合
  "id": "tapTest",
  "dataset": {
    "hi":"WeChat"
  }
},
"detail": { //额外的信息，不同的组件额外信息不太一样
  "x":53,
  "y":14
},
```

```
"touches":[{ //触摸事件，当前停留在屏幕中的触摸点信息的数组
  "identifier":0, //触摸点的标识符
  "pageX":53,距离文档左上角的距离,文档的左上角为原点,横向为X轴
  "pageY":14, //纵向为Y轴
  "clientX":53,//距离页面可显示区域（屏幕除去导航条）左上角距离
  "clientY":14 //横向为X轴，纵向为Y轴
}],
//changedTouches 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）。
"changedTouches":[{
  "identifier":0,
  "pageX":53,
  "pageY":14,
  "clientX":53,
  "clientY":14
}]
}
```

事件分类

了解了事件的对象，下面再来介绍事件的分类和绑定。事件分为冒泡事件和非冒泡事件：

- 冒泡事件：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：当一个组件上的事件被触发后，该事件不会向父节点传递。

WXML 的冒泡事件如表 4-2 所示。

表 4-2

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
longtap	手指触摸后，超过 350ms 再离开

注意：除表 4-2 之外的其他组件自定义事件如无特殊申明都是非冒泡事件，例如<form/>的submit事件，<input/>的input事件，<scroll-view/>的scroll事件（详见本书第 7 章）。

事件绑定

事件绑定的写法同组件的属性，以 key、value 的形式：

- key 以bind或catch开头，然后加上事件的类型，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。不然当触发事件的时候会报错。

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。

例如在下面这个例子中，点击 inner view 会先后触发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），点击 middle view 会触发handleTap2，点击 outter view 会触发handleTap1。

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

4.2.6 引用

WXML 提供了两种文件引用方式：import和include。

import

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item的template：

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

import有作用域的概念，即只会import目标文件中定义的template，而不会import目标文件import的template。

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。

```
<!-- A.wxml -->
<template name="A">
  <text> A template </text>
</template>
```

```
<!-- B.wxml -->
<import src="a.wxml"/>
<template name="B">
  <text> B template </text>
</template>
```

```
<!-- C.wxml -->
<import src="b.wxml"/>
<template is="A"/>  <!--报错不能使用A.wml的模板 -->
<template is="B"/>
```

include

include可以引入目标文件中除了<template/>的整个代码，相当于是复制到include位置，如：

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

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

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

4.3 WXSS 语法

WXSS 是一套样式语言，用于描述 WXMl 的组件样式。它将决定 WXML 的组件应该怎么显示。

为了让广大的前端开发者适应，WXSS 具有 CSS 的大部分特性，同时为了更适合开发微信小程序，对 CSS 进行了扩充以及修改。

本节主要介绍 WXSS 和 CSS 相同的基本语法，WXSS 和 CSS 差异在 4.4 节中会介绍。

4.3.1 语法规则

如图 4-7 所示，WXSS 或者 CSS 语法规则由以下两个主要的部分构成：

(1) 选择器。

(2) 一条或多条声明。

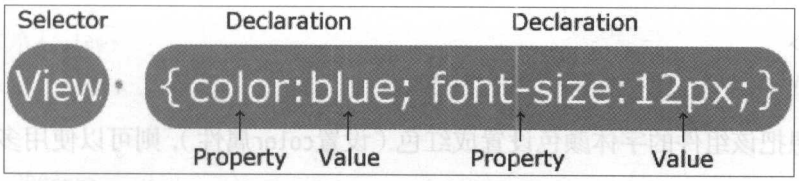

图 4-7 WXSS 语法规则

选择器通常是指向需要改变样式的 WXML 元素。每条声明由一个属性和一个值组成。属性（property）是你希望设置的样式属性（style attribute）。每个属性有一个值。属性和值被冒号分开。

4.3.2 注释

注释是用来解释代码的，并且可以随意编辑它，微信小程序会忽略它。CSS 注释以 /* 开始，以 */ 结束，这和 JavaScript 多行注释写法一致，如下所示：

```
/*这是个注释*/
View{
    color:blue;
}
```

4.3.3 选择器

选择器就是为了定位当前的样式控制.wxml 文件中的哪个元素。WXSS 并不支持全部 CSS 的选择器，但是常用的都支持。微信小程序官方推荐的选择器如表 4-3 所示。

表 4-3

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

选择器具体怎么用下面来演示一下。下面定义了一个组件，指定了 id 和 class 属性。

```
<!--wxml-->
<view id="v" class="red">红色字体</view>
```

如果想把该组件的字体颜色设置成红色（设置color属性），则可以使用多种选择器。

(1) 使用#id选择器：

```
#v{
    color: red;
}
```

(2) 使用.class选择器：

```
.red{
    color: red;
}
```

(3) 使用element选择器：

```
View{
    color: red;
}
```

以上三种写法任选一种就能实现把字体变成红色，如果三种写法对应的属性值不一致，则优先级为：#id>.class>element。

建议使用.class选择器，因为在一个.wxml 文件中，每个控件的id一般都是唯一存在的，局限性太大。而通过element选择器范围又太大。使用.class选择器恰到好处，而且每个组件的class属性可以定义多个值，如下:

```
<!--wxml-->
<view class="red whiteBg">红色字体</view>
```

这里就定义了 red和whiteBg两个值，<view>组件就具备这两个选择器的所有样式。

4.4 WXSS 基本属性

WXSS 和 CSS 一样，样式设置有很多，足够用一本书介绍的了。我们不可能在这一节中介绍完。其实好多属性也没必要死记，可以在需要的时候查阅《CSS 参考手册》(文档地址：http://www.phpstudy.net/css3/)。

下面先介绍几个基本常用的样式属性，参考下面代码:

```
<!--wxml-->
<view>小程序</view>

/*WXSS*/
View{
    width: 200rpx;
    height: 200rpx;
    background-color: blue;
    color: red;
    font-size: 50rpx;
}
```

上面用到的几个属性就是最基本的属性了：

(1) width 组件的宽度。

(2) height 组件的高度。

(3) background-color 组件的背景颜色，在上面的代码中定义为蓝色。

(4) color 内容的颜色，在上面的代码中定义为红色。

(5) font-size 字体的大小。

注意：代码中用到的尺寸单位为 rpx，可以参考 4.5 节。

代码运行结果如图 4-8 所示。

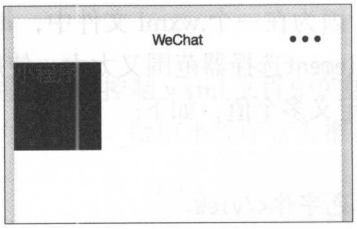

图 4-8 运行结果

4.5 CSS 和 WXSS 的区别

WXSS 具有 CSS 的大部分特性，同时，为了满足微信小程序开发的要求（主要体现在不同手机屏幕的适配问题），WXSS 对 CSS 进行了扩充修改，与 CSS 相比，WXSS 所具有的特性有：

(1) 尺寸单位。

(2) 样式导入。

4.5.1 尺寸单位

WXSS 新增的两种尺寸单位为 rpx 与 rem，和原有尺寸相比，这两种尺寸单位会根据手机屏幕的尺寸而改变，从而可以更好地适配不同尺寸的屏幕。

rpx（responsive pixel）：可以根据屏幕宽度进行自适应。规定屏幕宽为 750rpx。例如在 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，1rpx = 0.5px = 1 物理像素。在不同屏幕尺寸的手机上，rpx 和 px 转换算法如表 4-4 所示。

表 4-4

设备	rpx 换算 px (屏幕宽度/750)	px 换算 rpx (750/屏幕宽度)
iPhone 5	1rpx = 0.42px	1px = 2.34rpx
iPhone 6/7	1rpx = 0.5px	1px = 2rpx
iPhone 6/7 plus	1rpx = 0.552px	1px = 1.81rpx

rem（root em）：规定屏幕宽度为 20rem；1rem = (750/20)rpx。

建议在开发微信小程序时，设计师可以用 iPhone 6 作为视觉稿的标准，模拟器默认的设备也是 iPhone 6。

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。

4.5.2 样式导入

使用 @import 语句可以导入外联样式表，@import 后面会有需要导入的外联样式表的相对路径，用;表示语句结束。示例代码：

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

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

4.5.3 内联样式

小程序组件除了支持class、id等，还支持使用style属性来控制组件的样式。

style控制的样式优先级最高，和 css 不一样的是，style属性的值支持{{...}}动态设置。

一般静态的样式统一写到 class 中。style 接收动态的样式，在运行时会进行解析，所以不要将静态的样式写进 style 中，以免影响渲染速度。

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

4.5.4 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 page 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。

4.6 本章小结

本章主要介绍了 WXML 和 WXSS 语法，这一章主要为了后面学习小程序的核心知识做铺垫。

5 微信小程序开发基础

在第 2 章中已经阐述过微信小程序的基本框架和目录结构，本章会阐述微信小程序开发的配置、注册、样式展现等基础知识。这是学习微信小程序开发的前提，是必须掌握的部分。

5.1 全局配置

微信小程序的全局配置都要在 app.json 文件中进行。这些配置可以决定页面文件的路径、窗口表现、设置网络超时时间、设置多 tab 等。app.json 配置项列表如表 5-1 所示。

表 5-1

属性名	类型	必填	说明
pages	String Array	是	设置页面路径
window	Object	否	设置默认页面的窗口表现
tabBar	Object	否	设置底部 tab 的表现
networkTimeout	Object	否	设置网络超时时间
debug	Boolean	否	设置是否开启 debug 模式

pages 配置项

pages 接受一个数组值，每一项都是字符串，用来指定小程序由哪些页面组成。每一项代表对应页面的 [路径 + 文件名] 信息。

注意：

- 数组的第一项代表小程序的初始页面。
- 小程序中新增/减少页面，都需要对 pages 数组进行修改。
- 文件名不需要写文件后缀，因为框架会自动去寻找路径.json、.js、.wxml、.wxss 的四个文件进行整合。

代码示例如图 5-1 所示。

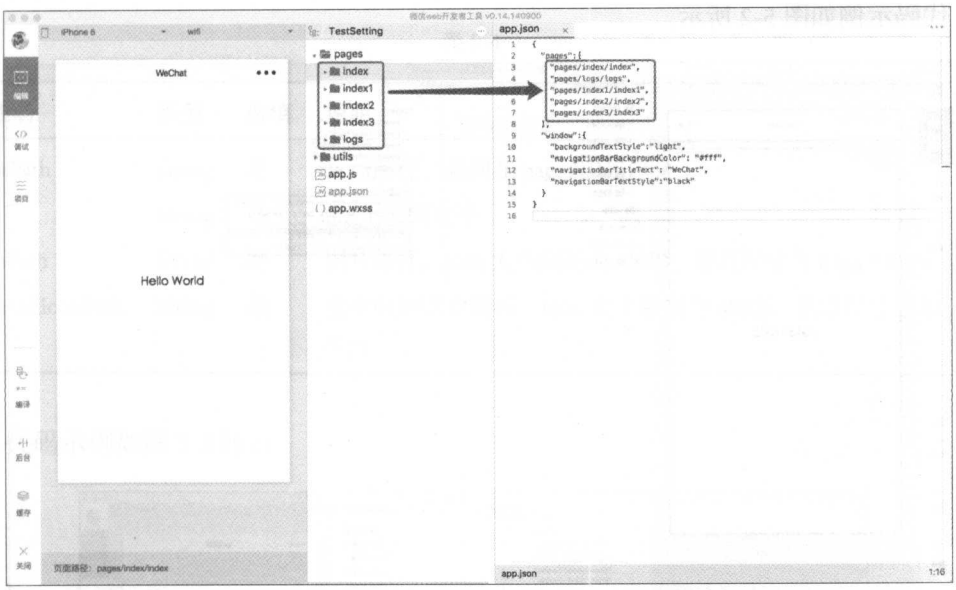

图 5-1 pages 配置项

window 配置项

window 接受对象值，用来设置小程序的状态栏、导航条、窗口页面等对象的基本样式属性。window 配置的对象都有默认值且为非必填，可配置的对象如表 5-2 所示。

表 5-2

属性名	类型	默认值	说明
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如"#000000"
navigationBarTextStyle	String	white	导航栏标题颜色，仅支持 black/white
navigationBarTitleText	String		导航栏标题文字内容
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	String	dark	下拉背景字体、loading 图的样式，仅支持 dark/light
enablePullDownRefresh	Boolean	false	设置是否开启下拉刷新

代码示例如图 5-2 所示。

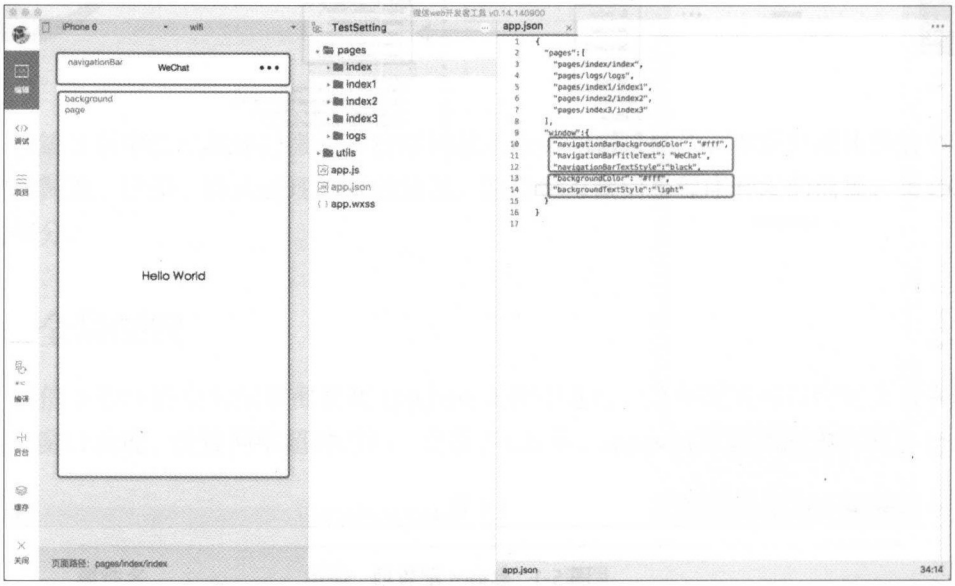

图 5-2　window 配置项

tabBar 配置项

tabBar 接受一个数组值，用来设置 tab 底部标签栏的样式表现，以及 tab 切换时显示的对应页面。tabBar 所包含的属性如表 5-3 所示。

表 5-3

属性名	类型	必填	说明
color	HexColor	是	tab 上的文字默认颜色
selectedColor	HexColor	是	tab 上的文字选中时的颜色
backgroundColor	HexColor	是	tab 的背景色
borderStyle	String	否	tabbar 上边框的颜色，仅支持 black/white，默认为 black
list	Array	是	tab 的列表，详见 list 属性说明，最少两个，最多 5 个 tab
position	String	否	可选值 bottom、top，默认为 bottom

其中 list 接受一个数组，数组中的每个项都是一个对象，其属性值如表 5-4 所示。

表 5-4

属性名	类型	必填	说明
pagePath	String	是	页面路径，必须在 pages 中先定义
text	String	是	tab 上按钮文字
iconPath	String	是	图片路径，icon 大小限制为 40KB，建议尺寸为 81px * 81px
selectedIconPath	String	是	选中时的图片路径，icon 大小限制为 40KB，建议尺寸为 81px * 81px

代码示例如图 5-3 所示。

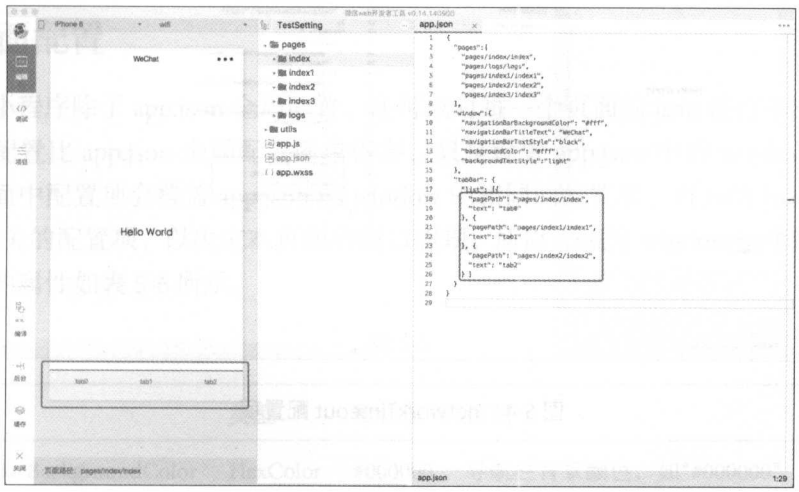

图 5-3 tabBar 配置项

注意：

- tabBar 只能配置最少两个、最多 5 个 tab，tab 按数组的顺序排序。
- 通过页面跳转（wx.navigateTo）或者页面重定向（wx.redirectTo）所到达的页面，即使它是定义在 tabBar 配置中的页面，也不会显示底部的 tab 栏。

networkTimeout 配置项

networkTimeout 接受对象值，用来设置各种网络请求对象超时时间。超时时间单位为毫秒，默认为 60000。networkTimeout 所包含的属性如表 5-5 所示。

表 5-5

属性名	类型	必填	说明
request	Number	否	wx.request 的超时时间，单位：毫秒，默认为 60000
connectSocket	Number	否	wx.connectSocket 的超时时间，单位：毫秒，默认为 60000
uploadFile	Number	否	wx.uploadFile 的超时时间，单位：毫秒，默认为 60000
downloadFile	Number	否	wx.downloadFile 的超时时间，单位：毫秒，默认为 60000

代码示例如图 5-4 所示。

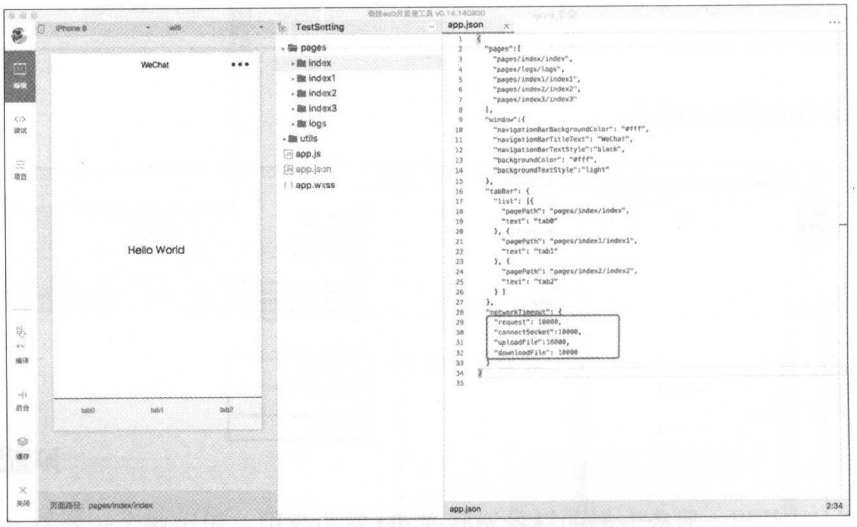

图 5-4 networkTimeout 配置项

debug 配置项

debug 接收一个 Boolean 值，用于设置开启开发者工具的调试模式。默认是 false，开启后，在开发者工具的控制台面板中，调试信息以 info 的形式给出，其信息有 Page 的注册、页面路由、数据更新、事件触发。这样可以帮助开发者快速定位一些常见的问题，但是，在正式发布时，应当关闭此配置项。

代码示例如图 5-5 所示。

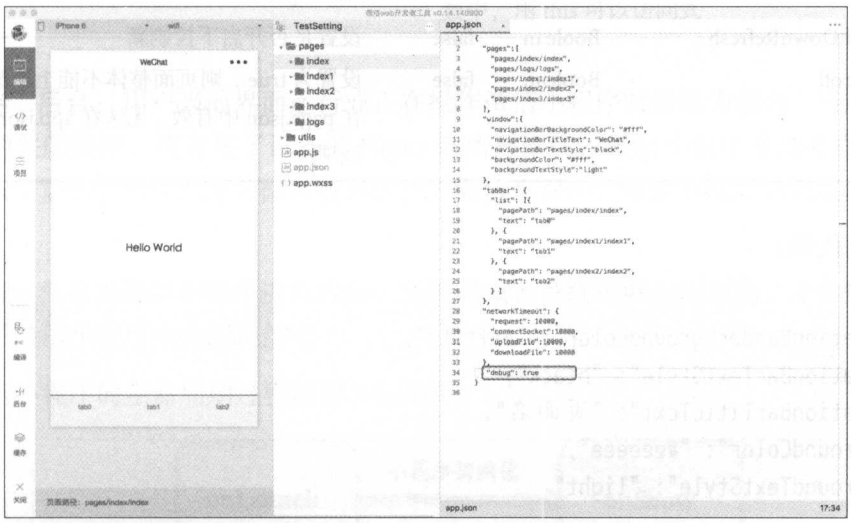

图 5-5 debug 配置项

5.2 页面配置

微信小程序除了 app.json 全局配置，还可以对每一个页面的.json 进行单独配置。每个页面的配置比 app.json 全局配置简单得多，只是设置 app.json 中的 window 配置项的内容，页面中配置项会覆盖 app.json 的 window 中相同的配置项。页面的.json 只能设置 window 相关的配置项，以决定本页面的窗口表现，所以无须写 window 这个键。页面配置所包含的属性如表 5-6 所示。

表 5-6

属性名	类型	默认值	说明
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如"#000000"

续表

属性名	类型	默认值	说明
navigationBarTextStyle	String	white	导航栏标题颜色，仅支持 black/white
navigationBarTitleText	String		导航栏标题文字内容
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	String	dark	下拉背景字体、loading 图的样式，仅支持 dark/light
enablePullDownRefresh	Boolean	false	设置是否开启下拉刷新
disableScroll	Boolean	false	设置为 true，则页面整体不能上下滚动；只在 page.json 中有效，无法在 app.json 中设置该项

示例代码：

```
{
  "navigationBarBackgroundColor": "#ffffff",
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "页面名",
  "backgroundColor": "#eeeeee",
  "backgroundTextStyle": "light"
}
```

5.3 注册程序

App() 函数

App() 函数是微信小程序注册程序函数。App() 接受一个 object 参数，其指定微信小程序的生命周期函数等。App() 方法有且仅有一个，存在于 app.js 中。object 参数说明如表 5-7 所示。

表 5-7

属性名	类型	说明	触发时机
onLaunch	Function	生命周期函数——监听小程序初始化	当微信小程序初始化完成时，会触发 onLaunch（全局只触发一次）
onShow	Function	生命周期函数——监听小程序显示	当微信小程序启动，或从后台进入前台显示，会触发 onShow

续表

属性名	类型	说明	触发时机
onHide	Function	生命周期函数——监听小程序隐藏	当微信小程序从前台进入后台，会触发 onHide
onError	Function	错误监听函数	当微信小程序发生脚本错误，或者 API 调用失败时，会触发 onError 并带上错误信息
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 可以访问式

前台、后台：用户当前界面运行或正在操作微信小程序时被称为前台。当用户点击左上角的关闭按钮，或者按了设备的 Home 键离开微信，微信小程序并没有直接销毁，而是进入了后台；当再次进入微信或再次打开微信小程序，微信小程序又会从后台进入前台。

销毁：只有当微信小程序进入后台一定时间或者系统资源占用过高，才会被真正销毁。此时微信小程序生命周期结束。

微信小程序的生命周期过程如图 5-6 所示。

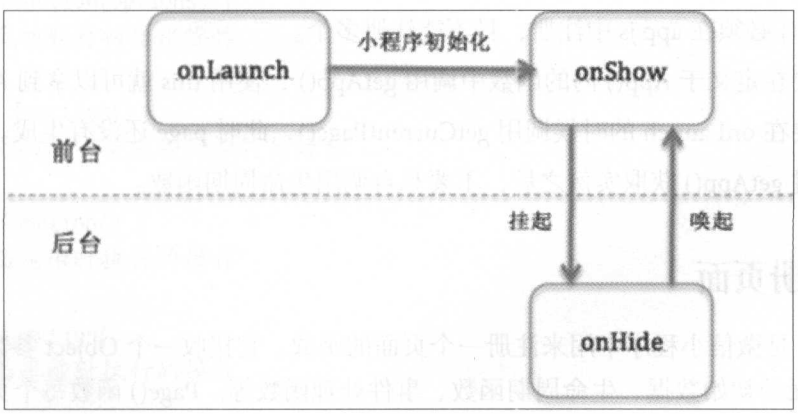

图 5-6 微信小程序的生命周期

示例代码：

```
App({
  onLaunch: function() {
    // 启动时执行的初始化方法
  },
```

```
  onShow: function() {
      // 小程序进入前台时执行的方法
  },
  onHide: function() {
      // 小程序进入后台时执行的方法
  },
  onError: function(msg) {
    console.log(msg)
  },
  globalData: 'I am global data'
})
```

getApp() 函数

微信小程序提供了全局的 getApp() 函数，可以用来获取小程序实例。示例代码：

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

注意：

- App() 必须在 app.js 中注册，且不能注册多个。
- 不要在定义于 App() 内的函数中调用 getApp()，使用 this 就可以拿到 App 实例。
- 不要在 onLaunch 的时候调用 getCurrentPage()，此时 page 还没有生成。
- 通过 getApp() 获取实例之后，不要私自调用生命周期函数。

5.4 注册页面

Page() 是微信小程序中用来注册一个页面的函数。它接收一个 Object 参数，可以用于指定页面的初始数据、生命周期函数、事件处理函数等。Page() 函数每个页面有且仅有一个，存在于.js 文件中。Object 参数说明如表 5-8 所示。

表 5-8

属性名	类型	说明
data	Object	页面的初始数据
onLoad	Function	生命周期函数——监听页面加载
onReady	Function	生命周期函数——监听页面初次渲染完成

续表

属性名	类型	说明
onShow	Function	生命周期函数——监听页面显示
onHide	Function	生命周期函数——监听页面隐藏
onUnload	Function	生命周期函数——监听页面卸载
onPullDownRefresh	Function	页面相关事件处理函数——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	用户点击右上角分享按钮
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

示例代码：

```
//index.js
Page({
  data: {
    text: "This is page data."
  },
  onLoad: function(options) {
    // 页面加载时初始化操作
  },
  onReady: function() {
    // 页面初次渲染完成时执行的操作
  },
  onShow: function() {
    // 页面显示时执行的操作
  },
  onHide: function() {
    // 页面隐藏时执行的操作
  },
  onUnload: function() {
    // 页面卸载或关闭时执行的操作
  },
  onPullDownRefresh: function() {
    // 用户在页面下拉时执行的操作
  },
  onReachBottom: function() {
    // 到达页面底部时的操作
```

```
  },
  onShareAppMessage: function () {
   // return custom share data when user share.
  },
  // 事件处理
  viewTap: function() {
    this.setData({
      text: 'Set some data for updating view.'
    })
  },
  customData: {
    hi: 'MINA'
  }
})
```

注意：

- bug：对于 iOS/Android 6.3.30，首次进入页面时，如果页面不满一屏时会触发 onReachBottom，应为只有用户主动上拉才触发。
- bug：对于 iOS/Android 6.3.30，手指上拉时，会触发多次 onReachBottom，应为一次上拉只触发一次。

5.4.1 页面的生命周期

每个微信小程序应用都有自己的生命周期，而微信小程序应用的每个页面也有自己的生命周期，在 5.3 节介绍注册程序时，已经阐述了应用的生命周期，本节主要阐述页面的生命周期。

页面的生命周期和应用的生命周期略有不同，页面的生命周期如图 5-7 所示。

(1) onLoad：页面加载。一个页面只会调用一次。接收页面参数可以获取 wx.navigateTo 和 wx.redirectTo 及 <navigator/> 中的 query。

(2) onShow：页面显示。每次打开页面都会调用一次。

(3) onReady：页面初次渲染完成。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。对界面的设置（如 wx.setNavigationBarTitle）可在 onReady 之后设置。

(4) onHide：页面隐藏。当 navigateTo 或底部 tab 切换时调用。

(5) onUnload：页面卸载。当进行 redirectTo 或 navigateBack 操作时调用。

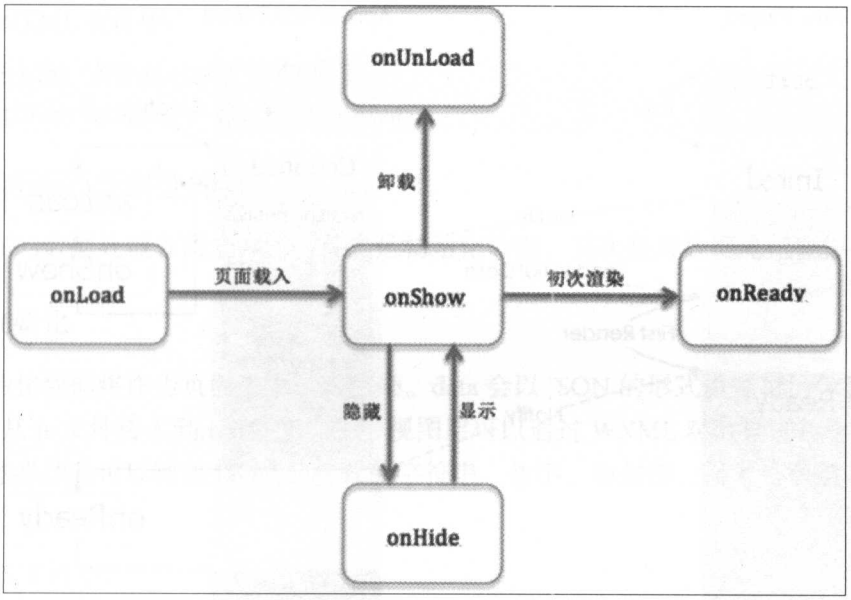

图 5-7 页面的生命周期

在本书前面阐述过实例，那么，页面与实例的整体交互流程和生命周期是怎样的呢？这里用图 5-8 加以说明。

从图 5-8 中可以看到，左侧是视图层（.wxml 与.wxss 文件），右侧是逻辑层（.js 文件）。页面初始化后，在整个生命周期中持续进行相应的业务数据准备、数据表现、响应事件、数据保存等，直到页面销毁。

5.4.2 页面的事件处理

除了初始化数据和生命周期函数，在页面 Page() 中还可以定义一些特殊的函数：事件处理函数。在 WXML 中，可以在组件里加入事件绑定，当达到触发事件时，就会执行 Page 中定义的事件处理函数。

示例代码：

在 JS 文件中：

```
Page({
//定义一个 viewTap 事件处理函数
  viewTap: function() {
    console.log('view tap')
  }
})
```

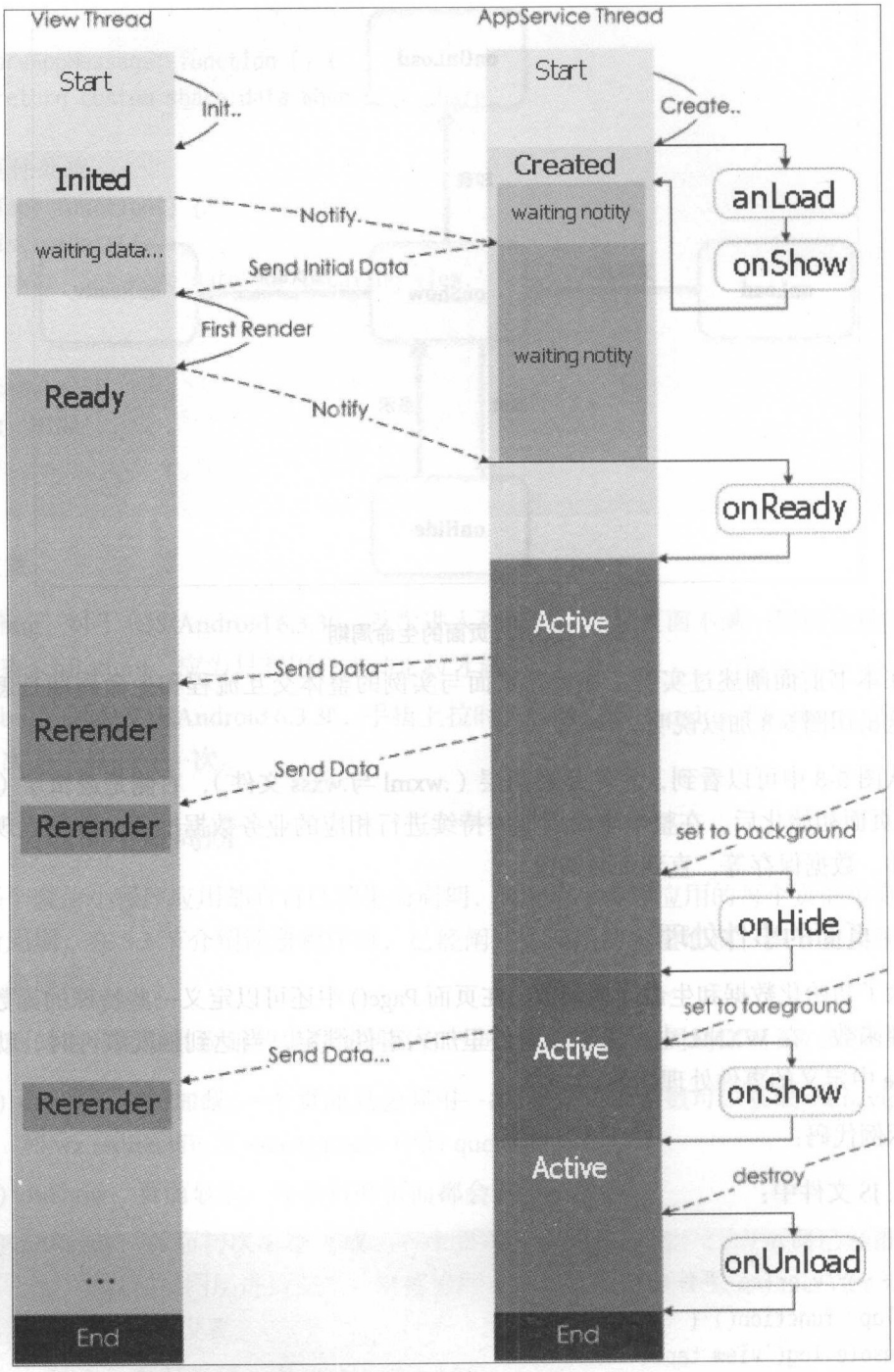

图 5-8　页面与实例的生命周期

在 WXML 文件中：

```
//绑定viewTap 事件到view组件中
<view bindtap="viewTap"> click me </view>
```

5.4.3 页面的数据处理

页面的数据处理分两个方面，首先是初始化数据，其次是对数据进行操作。

数据的初始化

初始化数据将作为页面的第一次渲染。data 会以 JSON 的形式由逻辑层传至视图层（也就是从.js 文件传入到.wxml 文件中，视图层可以通过 WXML 对数据进行绑定），所以其数据必须是可以转成 JSON 的格式的字符串、数字、布尔值、对象、数组。示例代码：

在 JS 文件中：

```
Page({
  data: {
    text: 'init data',    //初始化一个字符串
    array: [{msg: '1'}, {msg: '2'}]     //初始化一个数组
  }
})
```

在 WXML 文件中：

```
<view>{{text}}</view>    //绑定并显示字符串
<view>{{array[0].msg}}</view>  //绑定并显示数组元素
```

数据的设置及展现

在页面 Page() 中，不仅可以初始化数据，还可以通过 setData () 函数把数据从逻辑层发送到视图层。同时改变对应的 this.data 的值。

注意：

- 直接修改 this.data 无效，无法改变页面的状态，还会造成数据不一致。
- 单次设置的数据不能超过 1024KB，尽量避免一次设置过多的数据。

setData() 函数接受一个对象，以 keyvalue 的形式表示将 this.data 中的 key 对应的值改变成 value。其中 key 可以非常灵活，以数据路径的形式给出，如 array[2].message，a.b.c.d，并且不需要在 this.data 中预先定义。

示例代码：

在 JS 文件中：

```
Page({
  data: {
    text: 'init data',
    array: [{text: 'init data'}],
    object: {
      text: 'init data'
    }
  },
  changeText: function() {
    // this.data.text = 'changed data'  // bad, it can not work
    this.setData({
      text: 'changed data'
    })
  },
  changeItemInArray: function() {
    // you can use this way to modify a danamic data path
    this.setData({
      'array[0].text':'changed data'
    })
  },
  changeItemInObject: function(){
    this.setData({
      'object.text': 'changed data'
    });
  },
  addNewField: function() {
    this.setData({
      'newField.text': 'new data'
    })
  }
})
```

在 WXML 文件中：

```
<view>{{text}}</view>
<button bindtap="changeText"> Change normal data </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>
```

5.4.4 页面的栈

微信小程序提供的框架是以栈的形式维护所有页面。当发生路由切换的时候，页面栈的表现如表 5-9 所示。

表 5-9

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，新页面入栈
Tab 切换	页面全部出栈，只留下新的 Tab 页面

getCurrentPages() 函数

getCurrentPages() 函数用于获取当前页面栈的实例，以数组形式按栈的顺序给出，第一个元素为首页，最后一个元素为当前页面。

注意：

- 不要尝试修改页面栈，会导致路由以及页面状态错误。

5.4.5 页面的路由

在微信小程序中，所有页面的路由全部由框架进行管理。路由的触发方式以及页面生命周期函数，如表 5-10 所示。

表 5-10

路由方式	触发时机	路由后页面	路由前页面
初始化	微信小程序打开的第一个页面	onLoad，onShow	—

续表

路由方式	触发时机	路由后页面	路由前页面
打开新页面	调用 API wx.navigateTo 或使用组件 <navigator open-type="navigate"/>	onLoad，onShow	onHide
页面重定向	调用 API wx.redirectTo 或使用组件 <navigator open-type="redirect"/>	onLoad，onShow	onUnload
页面返回	调用 API wx.navigateBack 或用户按左上角返回按钮	onShow	onUnload（多层页面返回每个页面都会按顺序触发 onUnload）
Tab 切换	调用 API wx.switchTab 或使用组件 <navigator open-type="switchTab"/> 或用户切换 Tab	—	分多种情况

Tab 切换对应的生命周期（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），如表 5-11 所示。

表 5-11

当前页面	路由后页面	触发的生命周期（按顺序）
A	A	Nothing happend
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(), A.onShow()
D	B	D.onUnload(), C.onUnload(), B.onLoad(), B.onShow()
D（从分享进入）	A	D.onUnload(), A.onLoad(), A.onShow()
D（从分享进入）	B	D.onUnload(), B.onLoad(), B.onShow()

5.5 简单封装与调用

文件作用域

在.js 文件中声明的变量和函数只在该文件中有效；在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，则可以在 App() 中设置。示例代码：

```
// app.js
//定义全局变量
App({
  globalData: 1
})
```

```
// a.js
// localValue变量，只在 a.js中有效.
var localValue = 'a'
// 获取App实例
var app = getApp()
// 获取全局数值并修改
app.globalData++
```

```
// b.js
// 可以在b.js中重新定义localValue变量，不会影响a.js中的localValue变量，两个变
    量是分别独立于相应.js文件的。
var localValue = 'b'
// 如果 a.js 在 b.js 之前运行，则此时的 globalData 的数值为2.
console.log(getApp().globalData)
```

模块化

在日常编程过程中，一般都会把一些可以复用的代码提炼出来，放到一个公共文件中。在微信小程序中，可以将一些公共的代码抽离成为一个单独的 JS 文件，作为一个模块。模块只有通过 module.exports 或者 exports 才能对外暴露接口。

注意：

- exports 是 module.exports 的一个引用，因此在模块里随意更改 exports 的指向会造成未知的错误。所以推荐开发者采用 module.exports 来暴露模块接口。
- 微信小程序目前不支持直接引入 node_modules，开发者需要使用 node_modules 时建议复制出相关的代码到微信小程序的目录中。

示例代码：

定义公共模块。

```
// common.js
//定义接口
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}
//对外暴露接口
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。

```
var common = require('common.js')
Page({
  helloMINA: function() {
    common.sayHello('MINA')
  },
  goodbyeMINA: function() {
    common.sayGoodbye('MINA')
  }
})
```

5.6 本章小结

本章已经详细阐述了微信小程序开发的基础知识，包括配置、注册、封装和调用。这些知识是微信小程序开发的基础，熟练掌握这些基础知识，能够使开发者理解微信小程序的开发框架，提高开发效率。微信小程序不仅提供了这些基础框架，还提供了丰富的组件和 API 等，这些知识在后面的章节会进行详细阐述。

6 Flexbox 布局

微信小程序通过 Flexbox 模型布局对组件进行排列。相比普通的布局方式，Flexbox 更容易实现宽高适合屏幕的布局，使用起来更灵活，非常适合于微信小程序的布局要求。

6.1 基本要素

Flexbox 布局的主要思想为通过设定容器（flex container 即伸缩容器）与子元素（flex item 即伸缩项目）的规则，使所有 view 组件在主轴（main axis）与侧轴（cross axis）上合理地自动分配，其模型如图 6-1 所示。

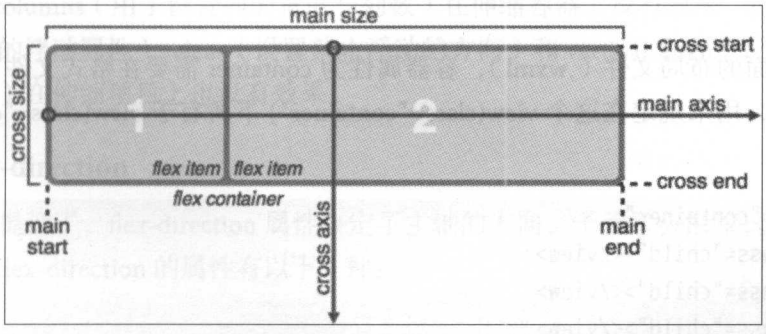

图 6-1　Flexbox 布局模型图

在大多数情况下，添加的 view 组件会沿着某一个轴（主轴或侧轴）从起点向终点排列（从 mainstart 向 mainend 或从 crossstart 向 crossend），各部分的定义为：

主轴（main axis）：容器伸缩的方向，子元素会沿着这个轴排列。注意，主轴并没有固定的方向（即不一定是像图 6-1 这样从左到右），它主要由 flex-direction 这个属性决定（详细见图 6-3）。

主轴起点（main start）/ 主轴终点（main end）：子元素从起点向终点排列。

主轴尺寸（main size）：伸缩项目在主轴方向上的尺寸，相当于项目的宽度或长度（取决于主轴是在宽度方向还是在长度方向上）。

侧轴（cross axis）：与主轴垂直的另一条轴。

侧轴起点（cross start）/侧轴终点（cross end）：子元素行上的放置由起点向终点方向。

侧轴尺寸（cross size）：与主轴尺寸相对应，取决于宽度和长度哪一个处在侧轴方向上。

6.2 容器属性

容器属性指的是包裹子元素的容器（flex container）属性，如图 6-2 所示。

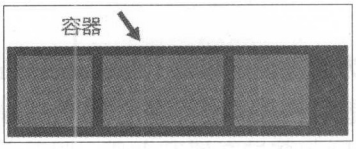

图 6-2 容器示意图

对于下面的布局文件（.wxml），容器属性为 container 需要在格式文件（.wxss）中设定的属性，用来规定在这个 view(class="container") 下所有子 view(class="child") 的排列方式。

```
<view class="container">
   <view class="child"></view>
   <view class="child"></view>
   <view class="child"></view>
</view>
```

6.2.1 display

使用 display:flex 或 display:inline-flex 来定义容器为伸缩容器。

```
.container{
    display: flex | inline-flex;
}
```

除这两个用于 Flexbox 布局的 display 属性外，还有三个经常使用的 display 属性。

```
.container{
    display: block | inline | inline-block;
}
```

block 表示块级元素，它的特点为：

- 每次添加子元素总在新的一行开始。
- 高度、行高以及顶和底边距都可控制。
- 如果未设定宽度，则宽度是其容器的 100%（即占满整个容器的宽度）。

inline 表示行内元素，它的特点为：

- 每次在同一行上添加新元素。
- 高、行高及顶和底边距不可改变。
- 宽度就是它的文字或图片的宽度，不可改变。

inline-block 则融合了两者的某些特点，即所有元素处于同一行上且可以更改其高度、宽度等属性。

对于 Flexbox 布局的两个 display 属性，flex 表示父组件为块级元素，inline-flex 表示父组件为行内元素。

注意：columns（用于设置列的宽度与列数）在伸缩容器上没有效果，此外 float（用于设置元素的浮动属性）、clear（设置禁止浮动的方向）和 vertical-align（设置元素的垂直对齐方式）在伸缩项目上也没有效果。

6.2.2 flex-direction

前面也提到了，flex-direction 属性决定了主轴的方向，子元素会沿着这个方向添加到容器中。flex-direction 的属性有以下 4 种：

```
.container{
    flex-direction: row | row-reverse | column | column-reverse;
}
```

每种对应的排列方式如图 6-3 所示。

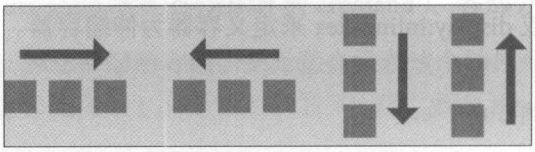

图 6-3 flex-direction 示意图

如果不设置，则默认值为 row，即从左向右排列。

6.2.3 flex-wrap

在默认情况下，添加进来的子元素会与其他元素处在同一行或同一列上，即便屏幕已经没有足够的位置也不换行。flex-wrap 属性决定了子元素是单行显示还是多行显示，如果为多行，则换行方向与侧轴方向一致。

```
.container{
    flex-wrap: nowrap | wrap | wrap-reverse ;
}
```

这三个属性分别代表不换行、换行以及沿侧轴反方向换行。

要实现子元素的自动排列，除设置此属性为 wrap 外，还有配合下面要介绍的 alignItem 来设置，且 alignItem 不能为“stretch”。

6.2.4 flex-flow

flex-flow 为 flex-direction 与 flex-wrap 的缩写版本，同时定义两个属性。

```
.container{
    flex-flow: row nowrap;
}
```

row nowrap 为默认值。

6.2.5 justify-content

justify-content 用来定义伸缩项目沿主轴的对齐方式，主要变现为在项目不能伸缩或已经达到伸缩最大值时，对多余的空白空间进行分配，分配的方式有：

```
.container{
    justify-content: flex-start | flex-end | center | space-between | space-around;
}
```

flex-start 为默认值。每种方式对应的示意图为 6-4 所示。

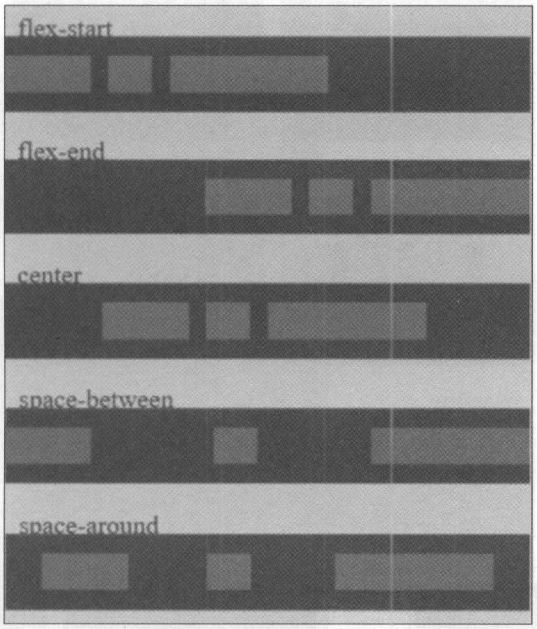

图 6-4 flex-start 示意图

其中 space-between 与 space-around 的区别为：space-between 将剩余空间平分然后添加在 item 之间；space-around 在平分剩余空间后，将分配空间的一半分别添加到 item 两端。

6.2.6 align-item

align-item 属性相当于侧轴上的 justify-context，决定了子元素在侧轴上的对齐方式。

```
.container{
    align-items: flex-start | flex-end | center | stretch | baseline;
}
```

每种情况对应的示意图如图 6-5 所示。

这里需要解释的是，baseline 指的是基线，一般在需要对齐文字的情况下使用，另外对 stretch 来说，子元素还是会遵循其 max/min-height/width，在此基础上最大限度地占满空间。

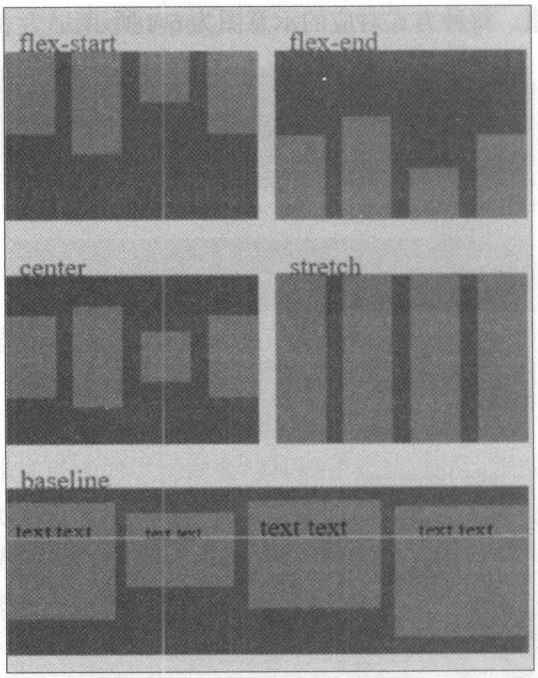

图 6-5 align-item 示意图

6.2.7 align-content

align-content 属性用来调整当子元素在容器中的对齐方式，只有在行数大于 1 的布局中有效。

```
.container{
    align-content: flex-start | flex-end | center | space-between | space-around;
}
```

用法与 justify-content 类似，效果如图 6-6 所示。其默认值为 stretch。

6.3 子元素属性

子元素属性定义了伸缩项目的属性，如图 6-7 所示。

对下面的布局文件 (.wxml) 来说，子元素属性为 child 需要在格式文件 (.wxss) 中设定的属性，用来规定这个 view(class="child") 所独有的属性值。

```
<view class="container">
   <view class="child"></view>
   <view class="child"></view>
```

```
    <view class="child"></view>
</view>
```

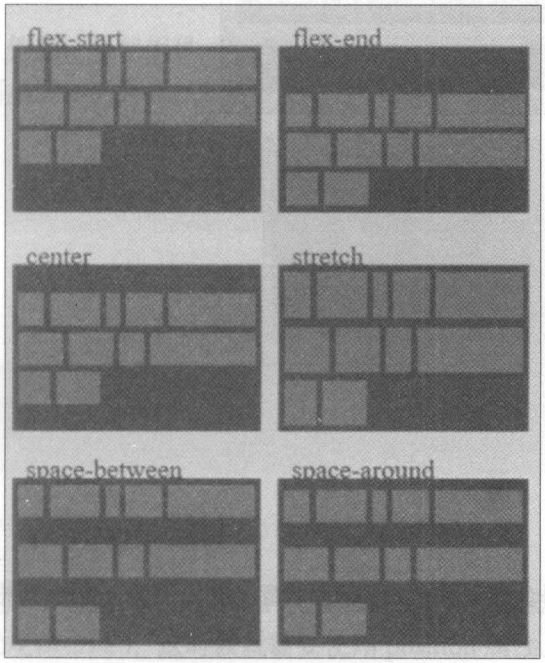

图 6-6 justify-content 示意图

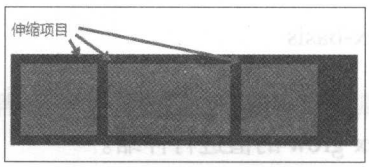

图 6-7 伸缩项目示意图

order

子元素默认按照编写的顺序排列，使用 order 属性可以更改这一方式，使所有子元素按 order 从小到大排列，这个值可以为负数。

```
.child{
    order: 1;
}
```

flex-grow

flex-grow 属性定义了子元素的扩展能力，数值越大在布局中占的比例越大，并且大小与这个数值成正比。负值无效。

```
.child{
    flex-grow: 2;
}
```

效果如图 6-8 所示。

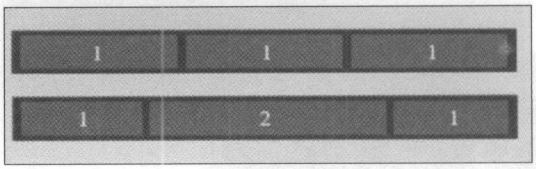

图 6-8　flex-grow 示意图

flex-shrink

flex-shrink 与 flex-grow 相对，定义了子元素收缩的能力。负值无效。

```
.child{
    flex-shrink: 2;
}
```

flex-basis

flex-basis 定于伸缩的基准值，项目会按这个比例进行伸缩。默认值为 auto，即根据 flex-grow 的值进行伸缩。

```
.child{
    flex-basis: 0 | auto ;
}
```

flex

flex 为前面三种属性的缩写，第二个值（flex-shrink）与第三个值（flex-basis）可以省略，默认值为“0 1 auto”，相当于同时设定了“flex-grow:0;flex-shrink:1;flex-basis:auto”。

```
.child{
    flex: 1;
}
```

这里建议使用 flex 而不是分别设置三个属性，使用 flex 会智能地设置其他的属性。

align-self

align-self 用来覆盖父组件样式中的 alignItems 的值，使子元素拥有自己想要的对齐方式。

```
.child{
    align-self: flex-end;
}
```

例如，在父组件 alignItems 为 flex-start 时，将子组件 align-self 设置为 flex-end 的示意图如图 6-9 所示。

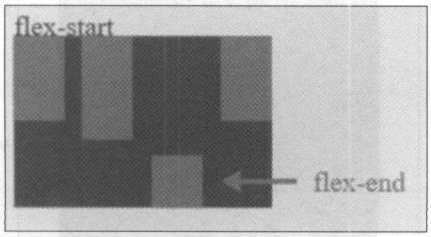

图 6-9 flex-start 示意图

6.4 position 属性

除了通过以上方式让系统自动安排布局，有些时候需要手动设置位置（例如布局属于无规律布局），这时候就需要使用 position 属性。

position 可能的值有：

- absolute：表示当前元素相对于第一个非 static 属性的父组件绝对定位，使用 top、bottom、left、right 四个键值来决定其到父组件的最上、最下、最左、最右的距离。
- fixed：表示当前元素相对于屏幕的绝对定位，同样使用 top、bottom、left、right 来设定位置。
- relative：表示当前元素相对于上一个同级组件的相对定位，使用 top、left 来设定到上一个同级组件最上沿、最左沿的距离，默认值为 0。
- static：默认值，没有定位，会忽略 top、bottom、left、right 或 z-index 的声明。
- inherit：表示当前元素继承其父元素的 position 属性。

这里比较常用的为 absolute 与 relative。

下面来看一个例子，首先在屏幕中间放置一个红色的 view，文字居中显示，如图 6-10 所示。

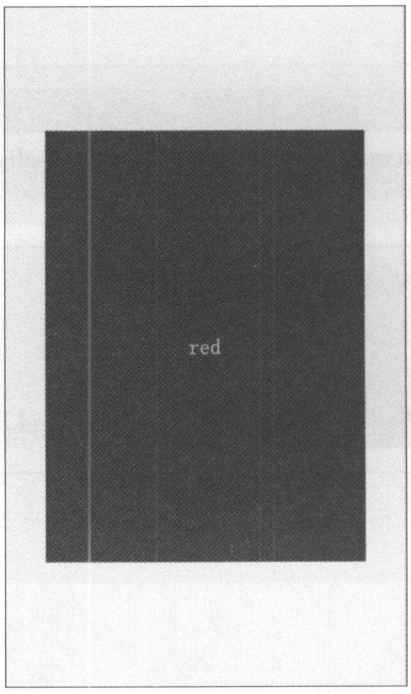

图 6-10　position 属性例图——初始布局

这时的 WXSS 设定为：

```
.center-view{
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100%
}

.red{
  display:flex;
  height: 800rpx;
  background-color: red;
  width: 600rpx;
  justify-content: center;
  align-items: center;
  color: #ffffff;
  font-size: 40rpx;
}
```

然后我们想在红色的 view 里添加一个绿色的 view，距离红色的顶端与左端各 100rpx，按照前面讲的，这种情况适合使用 position 属性，首先来看 relative 是否适用。将绿色的 view 添加到红色的 view 里：

```
<view class="center-view">
  <view class="red">red
    <view class="green"/>
  </view>
</view>
```

然后设定绿色的 view 的属性为：

```
.green{
  height: 200rpx;
  background-color: green;
  width: 200rpx;
  position: relative;
  top: 100rpx;
  left: 100rpx;
}
```

结果如图 6-11 所示。

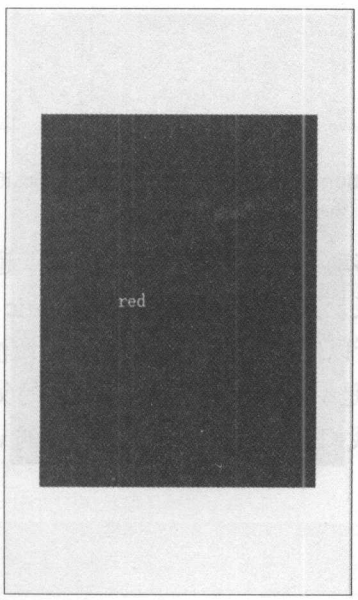

图 6-11 position 属性例图——relative 效果

可以看到，绿色方块加入到了红色布局里，但是位置不对，这是因为我们在红色 view 里设定了居中显示，所以根据 relative 的定义，这里的 100rpx 距离其实是文字的，而不是其父容器的。

接着把 relative 改成 absolute 试试看，结果如图 6-12 所示。

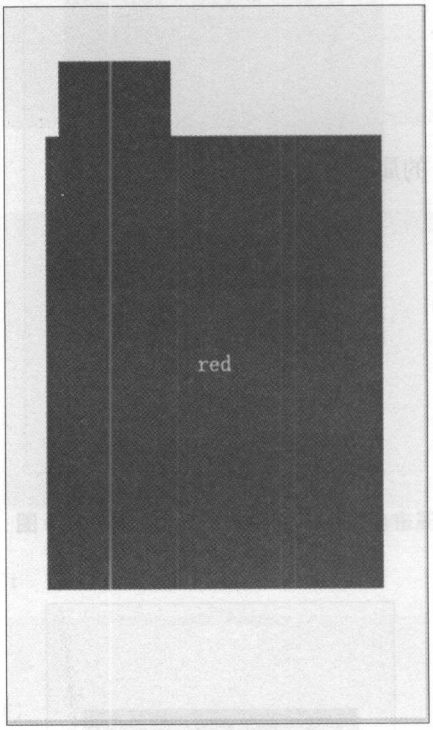

图 6-12 position 属性例图——父布局缺失 position 属性情况

我们发现，绿色方块被移动到了红色的 view 的上方，超出了红色的 view 的范围。这是因为，根据 absolute 的定义，其相对的是第一个非 static 的父组件，而红色的 view 因为没有设定 position 属性，所以使用默认值 static。在系统渲染布局时，对于 absolute 属性，不会根据红色的 view 而是一直向上寻找不为 static 的父组件，最后设定为了相对于整个布局的位置。为了达到我们要的效果，这里为红色的 view 添加 position：relative 的属性。

```
.red{
  display:flex;
  height: 800rpx;
  background-color: red;
```

```
  width: 600rpx;
  justify-content: center;
  align-items: center;
  color: #ffffff;
  font-size: 40rpx;
  position: relative;
}

.green{
  height: 200rpx;
  background-color: green;
  width: 200rpx;
  position: absolute;
  top: 100rpx;
  left: 100rpx;
}
```

这样就得到了我们所要的布局样式，如图 6-13 所示。

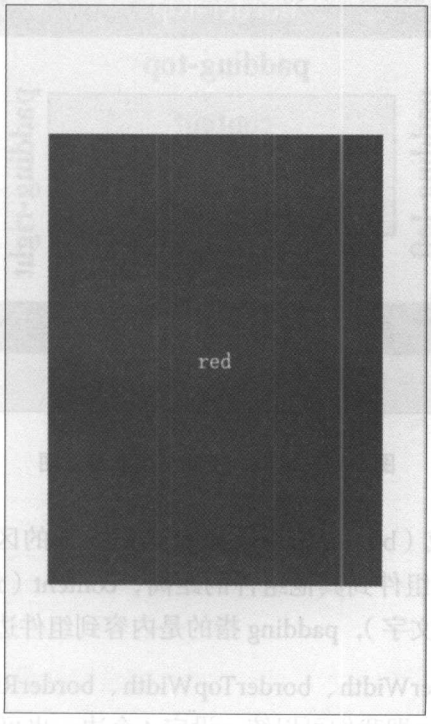

图 6-13 position 属性例图——正确结果

在设定组件的宽高时，可用的属性有：width、height、maxHeight、maxWidth、minHeight、minWidth。为了发挥 Flexbox 动态改变组件宽高的特性，尽量不指定组件的宽高，由系统根据组件的内容与设定的排列方式来动态地设定宽高。如果对宽高有要求，则尽量设定组件的最大值与最小值，以便其在一定范围内弹性改变（这也有利于配适不同尺寸的手机）。另外，如果必须要设定宽高，那么尽可能只设定其中一项，把另一项交给系统来设定。

6.5 边框、空隙与填充

对一个组件来说，还需要经常为其设定边框（border）、空隙（margin）与填充（padding）属性。

边框很好理解，空隙指的是组件之间的距离，而填充指的是组件的内容到其边框的距离，请参考图 6-14。

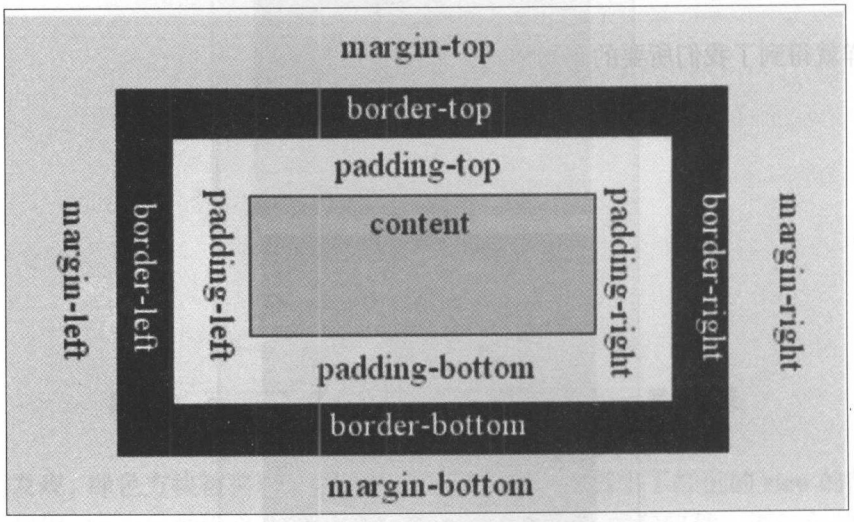

图 6-14 边框、空隙与填充示意图

在此图中，边框区域（border 部分）以及边框内部的区域为整个组件所占区域，margin的距离指的是这个组件到其他组件的距离，content（最里面的方块）表示这个组件的内容（例如按钮的文字），padding 指的是内容到组件边框的距离。

设定边框的值有 borderWidth、borderTopWidth、borderRightWidth、borderBottomWidth 与 borderLeftWidth，即我们可以统一设定 4 个边，也可以根据需求单独设定某一边。

设定填充的值有 padding（统一设定四个边）、paddingHorizontal（设定水平的两个边）、paddingVertical（设定垂直的两个边）、paddingBottom、paddingLeft、paddingRight 与 paddingTop。

设定空隙的值与填充类似，有 margin、marginHorizontal、marginVertical、marginBottom、marginLeft、marginRight 与 marginTop。

6.6 本章小结

本章介绍了编写微信小程序非常重要的布局写法，Flexbox 布局是微信小程序实现高效动态布局的关键（同样也是编写网页等前端技术中很实用的部分），希望通过本章的讲解，读者们可以理解 Flexbox 布局各个属性的作用。本章讲的内容会在后面的内容中反复出现，可以结合后面的内容，继续巩固这些知识。

7 组件的开发应用

微信小程序为开发者提供了一系列功能丰富的基础组件，开发者可以对这些组件进行组合，从而进行快速开发，创建出各式各样的微信小程序。组件是视图层的基本元素，是构建页面的基础。组件不仅自带一些功能，还具有微信风格的样式。一个组件通常包括开始标签、结束标签、属性、内容。属性用来修饰该组件，内容放在开始标签和结束标签之间，用页面来显示。

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

注意：

- 所有组件与属性都是小写，以连字符-连接。

共同属性类型

所有组件共有的属性如表 7-1 所示。

特殊属性

每个组件都有自定义的属性，可以对功能样式进行修改，但只支持以下七种数据类型，如表 7-2 所示。

表 7-1

属性名	类型	描述	注释
id	String	组件的唯一标示	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	组件上触发事件时，会发送给事件处理函数
bind*/catch*	EventHandler	组件的事件	—

表 7-2

类型	描述	注释
Boolean	布尔值	组件写上该属性，不管该属性等于什么，其值都为 true，只有组件上没有写该属性时，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	—

组件

微信小程序为开发者提供了七类基础组件和客服会话按钮，如表 7-3 所示。

表 7-3

类别	包含组件	说明
视图容器 (View Container)	view、scroll-view、swiper	视图、可滚动视图、滑块视图
基础内容 (Basic Content)	icon、text、progress	图标、文字、进度条
表单 (Form)	button、form、input、checkbox、radio、picker、picker-view、slider、switch、label	构建表单的基础组件

续表

类别	包含组件	说明
多媒体 (Media)	audio、image、video	音频、图片、视频
地图 (Map)	map	地图
导航 (Navigation)	navigator	页面导航
画布 (Canvas)	canvas	画布
客服会话按钮	contact-button	进入客服会话按钮

开发者通过这些组件，可以完美地构建出微信小程序。本章详细阐述这七类组件的开发及使用方法。

7.1 视图容器组件

微信小程序具有丰富的用户界面组件，借助这些组件，开发者可以很方便地搭建用户界面。一般情况下，小程序的界面都需要在.wxml 里面进行搭建。

7.1.1 view

微信小程序所有的组件都是建立在 view 基础上的，view 类似于 HTML 中的 元素。view 包含了一些属性，如表 7-4 所示。

表 7-4

属性名	类型	默认值	说明
hover	Boolean	false	是否启用点击态
hover-class	String	none	指定按下去的样式类。当hover-class="none"时，没有点击态效果
hover-start-time	Number	50	按住后多久出现点击态，单位：毫秒
hover-stay-time	Number	400	手指松开后点击态保留时间，单位：毫秒

示例代码：

在 WXML 中：

```
<view class="viewTitle">
  <text>View展示</text>
```

```
</view>

<!--样式一，横向排列-->
<view class="section">
  <view class="section__title">样式一，横向排列</view>
  <view class="flex-wrp">
    <view class="flex-item bc_green">111</view>
    <view class="flex-item bc_red">222</view>
    <view class="flex-item bc_blue">333</view>
  </view>
</view>

<!--样式二，竖向排列,注意在 .wxml的文件中也可以通过style参数进行样式设计-->
<view class="section">
  <view class="section__title">样式二，竖向排列</view>
  <view class="flex-wrp" style="height:300px">
    <view class="flex-item bc_green" style="margin-top: 0px">111</view>
    <view class="flex-item bc_red" style="margin-top: 100px">222</view>
    <view class="flex-item bc_blue" style="margin-top: 200px">333</view>
  </view>
</view>
```

在 WXSS 中，WXSS 可以直接写 CSS 代码设置一些样式。

```
.flex-wrp {
    height:200rpx;
    display:flex;
    background-color:#ffffff;
}
.flex-item {
    width:200rpx;
    height:200rpx;
    color:#ffffff;
    display:flex;
    justify-content:center;
    align-items:center;
}
.bc_green {
    background-color:#09BB07;
}
```

```
.bc_red {
    background-color:#F76260;
}
.bc_blue {
    background-color:#10AEFF;
}
```

运行效果如图 7-1 所示。

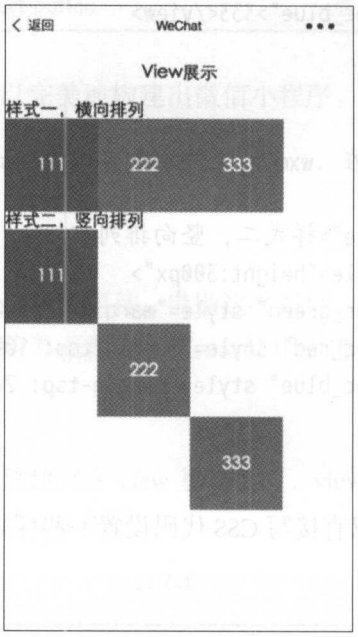

图 7-1 view 组件示例运行效果图

7.1.2 scroll-view

scroll-view 组件用来定义一个可滚动的视图区域，相当于 iOS 开发中的 UIScrollView。里面的内容如果超过这个区域的宽或高时，就会出现滚动条。scroll-view 包含的属性如表 7-5 所示。

表 7-5

属性名	类型	默认值	说明
scroll-x	Boolean	false	允许横向滚动
scroll-y	Boolean	false	允许纵向滚动

续表

属性名	类型	默认值	说明
upper-threshold	Number	50	距顶部/左边多远时（单位：px），触发 scrolltoupper 事件
lower-threshold	Number	50	距底部/右边多远时（单位：px），触发 scrolltolower 事件
scroll-top	Number	—	设置竖向滚动条位置
scroll-left	Number	—	设置横向滚动条位置
scroll-into-view	String	—	值为某子元素 id 时，则滚动到该元素，元素顶部对齐滚动区域顶部
bindscrolltoupper	EventHandle	—	滚动到顶部/左边，会触发 scrolltoupper 事件
bindscrolltolower	EventHandle	—	滚动到底部/右边，会触发 scrolltolower 事件
bindscroll	EventHandle	—	滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}

示例代码：

在 WXML 中：

```
<view class="viewTitle">
    <text class="titleName">ScrollView视图展示</text>
  </view>
  <!--样式一，竖向滑动-->
<view class="section">
    <view class="section__title">样式一，竖向滑动Vertical</view>
    <view class="flex-wrp">
    <!--bindscrolltoupper后面的参数可以不写，在.js文件中
    有对应的交互方法-->
      <scroll-view scroll-y="true" style="height: 200px;"
      bindscrolltoupper="upper" bindscrolltolower="lower"
      bindscroll="scroll" scroll-into-view="{{toView}}"
      scroll-top="{{scrollTop}}">
        <view id="blue" class="scroll-view-item bc_blue"></view>
        <view id="yellow"  class="scroll-view-item bc_yellow"></view>
        <view id="red" class="scroll-view-item bc_red"></view>
        <view id="green" class="scroll-view-item bc_green"></view>
      </scroll-view>
    </view>
```

```
</view>
<!--样式二，横向滑动-->
<view class="section">
    <view class="section__title">样式二，横向滑动Horizontal</view>
    <view class="flex-wrp">
    <scroll-view class="scroll-view_H" scroll-x="true" bindscroll="scroll" style="
        width: 100%">
        <view id="green" class="scroll-view-item_H bc_green"></view>
        <view id="red"  class="scroll-view-item_H bc_red"></view>
        <view id="yellow" class="scroll-view-item_H bc_yellow"></view>
        <view id="blue" class="scroll-view-item_H bc_blue"></view>
      </scroll-view>
    </view>
</view>
```

运行效果如图 7-2 所示。

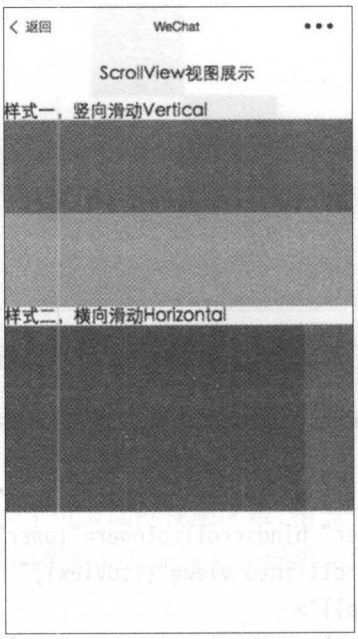

图 7-2　scroll-view 组件示例运行效果图

注意：

- 请勿在scroll-view中使用textarea、map、canvas、video组件。

- scroll-into-view的优先级高于scroll-top。
- 在滚动scroll-view时会阻止页面回弹，所以在scroll-view中滚动，是无法触发onPullDownRefresh 的。
- 若要使用下拉刷新功能，则应使用页面的滚动，而不是scroll-view，这样也能通过点击顶部状态栏回到页面顶部。

7.1.3 swiper

swiper 即滑块视图，比如首页进行滚动展示就需要用到这个组件。swiper 所包含的属性如表 7-6 所示。

表 7-6

属性名	类型	默认值	说明
indicator-dots	Boolean	false	是否显示面板指示点
indicator-color	Color	rgba(0, 0, 0, .3)	指示点颜色（这个属性目前暂未启用）
indicator-active-color	Color	#000000	当前选中的指示点颜色（这个属性目前暂未启用）
autoplay	Boolean	false	是否自动切换
current	Number	0	当前所在页面的 index
interval	Number	5000	自动切换时间间隔
duration	Number	500	滑动动画时长
circular	Boolean	false	是否采用衔接滑动
bindchange	EventHandle	—	current 改变时会触发 change 事件，event.detail = {current: current}

注意：

- 其中只可放置<swiper-item>组件，否则会导致未定义的行为。<swiper-item>组件仅可放置在<swiper>组件中，宽高自动设置为 100%。

示例代码：

在 WXML 中：

```
<view class="viewTitle">
  <text class="titleName">Swiper视图展示</text>
</view>
```

```
<view class="page__bd">
  <view class="section section_gap swiper">
    <swiper indicator-dots="{{indicatorDots}}" vertical="{{vertical}}"
      autoplay="{{autoplay}}" interval="{{interval}}"
      duration="{{duration}}">
      <block wx:for="{{background}}">
        <swiper-item>
          <view class="swiper-item bc_{{item}}"></view>
        </swiper-item>
      </block>
    </swiper>
  </view>
</view>
```

在 JS 中：

```
Page({
  data: {
    background: ['green', 'red', 'yellow'],
    indicatorDots: true,
    vertical: false,
    autoplay: false,
    interval: 3000,
    duration: 1200
  },
  changeIndicatorDots: function (e) {
    this.setData({
      indicatorDots: !this.data.indicatorDots
    })
  },
  changeVertical: function (e) {
    this.setData({
      vertical: !this.data.vertical
    })
  },
  changeAutoplay: function (e) {
    this.setData({
      autoplay: !this.data.autoplay
    })
  },
```

```
  intervalChange: function (e) {
    this.setData({
      interval: e.detail.value
    })
  },
  durationChange: function (e) {
    this.setData({
      duration: e.detail.value
    })
  }
})
```

运行结果如图 7-3 所示。

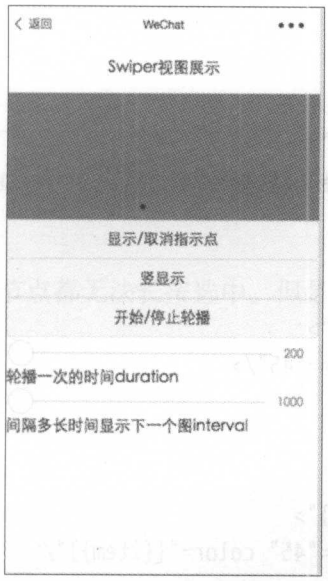

图 7-3 swiper 组件演示

7.2 基础内容组件

前面介绍了微信小程序的容器组件，所谓“万丈高楼平地起”，无论看上去多么美观的界面，都需要先创建容器组件来承载其他各式各样的组件。本节会介绍最基本的组件——基础内容组件，这些组件是每个微信小程序都需要用到的，包括 icon、text 和 progress，用于在界面中展示图标、文字、滚动条信息等。

7.2.1 icon

icon 即图标组件，微信小程序中自带的图标。icon 包括的属性如表 7-7 所示。

表 7-7

属性名	类型	默认值	说明
type	String	—	icon 的类型，有效值：success、success_no_circle、info、warn、waiting、cancel、download、search、clear
size	Number	23	icon 的大小，单位：px
color	Color	—	icon 的颜色，同 CSS 的 color

示例代码：

在 WXML 中：

```
<view class="group">
  <block wx:for="{{iconSize}}">
    <icon type="success" size="{{item}}"/>
  </block>
</view>
<view class="group">
  <block wx:for="{{iconType}}">
    <icon type="{{item}}" size="45"/>
  </block>
</view>
<view class="group">
  <block wx:for="{{iconColor}}">
    <icon type="success" size="45" color="{{item}}"/>
  </block>
</view>
```

运行效果如图 7-4 所示。

7.2.2 text

text 即文本组件，显示文字时使用的组件，类似于 HTML 中的标签，支持“\n”换行，<text>组件内只支持<text>嵌套。同时<text>是唯一可复制文本的标签。

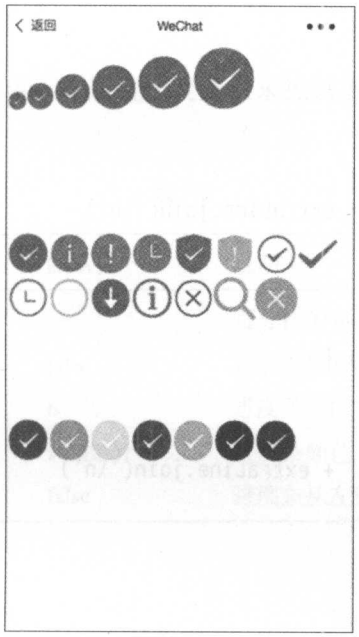

图 7-4 icon 组件示例运行效果图

注意：

- 除文本节点以外的其他节点都无法长按选中。但是，目前版本的 text 长按复制功能尚未实现。

示例代码：

在 WXML 中：

```
<view class="btn-area">
  <view class="body-view">
    <text>{{text}}</text>
    <button bindtap="add">add line</button>
    <button bindtap="remove">remove line</button>
  </view>
</view>
```

在 JS 中：

```
var initData = '我是第一行\n我是第二行\n我是第三行'
var extraLine = [];
Page({
  data: {
```

```
    text: initData
  },
  add: function(e) {
    extraLine.push('下一行')
    this.setData({
      text: initData + '\n' + extraLine.join('\n')
    })
  },
  remove: function(e) {
    if (extraLine.length > 0) {
      extraLine.pop()
      this.setData({
        text: initData + '\n' + extraLine.join('\n')
      })
    }
  }
})
```

运行效果如图 7-5 所示。

图 7-5　text 组件示例运行效果图

7.2.3 progress

progress 即进度条，开发者可以通过改变属性来控制进度条样式。progress 包含的属性如表 7-8 所示。

表 7-8

属性名	类型	默认值	说明
percent	Float	—	百分比为 0~100
show-info	Boolean	false	在进度条右侧显示百分比
stroke-width	Number	6	进度条线的宽度，单位为 px
color	Color	#09BB07	进度条颜色
active	Boolean	false	进度条从左往右的动画

示例代码：

在 WXML 中：

```
<view class="progress-view">
  <view class="progress-box">
    <progress percent="20" show-info />
  </view>
  <view class="progress-box">
    <progress percent="40" stroke-width="12" />
  </view>
  <view class="progress-box">
    <progress percent="60" color="pink" />
  </view>
  <view class="progress-box">
    <progress percent="100" active />
  </view>
</view>
```

运行效果如图 7-6 所示。

7.3 表单组件

表单组件用于构建与用户交互的表单。它包含 button、checkbox、radio、input、textarea、form、label、picker、picker-view、slider 和 switch。

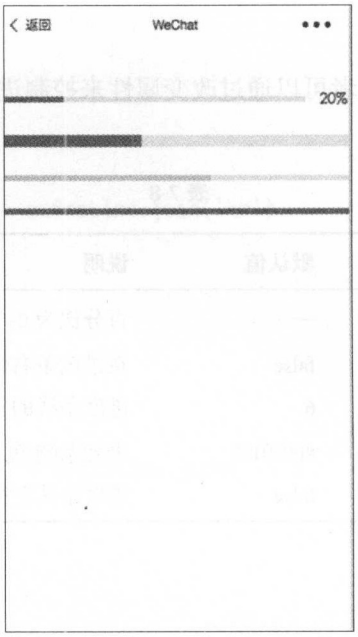

图 7-6 progress 组件示例运行效果图

7.3.1 button

button 即按钮，它可以供用户点击，从而触发一系列操作。它有许多属性来控制按钮样式。button 包含的属性如表 7-9 所示。

表 7-9

属性名	类型	默认值	说明
size	String	default	有效值 default、mini
type	String	default	按钮的样式类型，有效值 primary、default、warn
plain	Boolean	false	按钮是否镂空，背景色透明
disabled	Boolean	false	是否禁用
loading	Boolean	false	名称前是否带 loading 图标
form-type	String	—	有效值：submit、reset，用于<form/>组件，点击分别会触发 submit/reset 事件
hover-class	String	button-hover	指定按钮按下去的样式类。当 hover-class="none" 时，没有点击态效果
hover-start-time	Number	50	按住后多久出现点击态，单位：毫秒
hover-stay-time	Number	400	手指松开后点击态保留时间，单位：毫秒

示例代码：

在 WXML 中：

```
<view class="content">
  <view class="button_box">
    <button size="default">Content</button>
  </view>
  <view class="button_box">
    <button size="mini">Content</button>
  </view>
  <view class="button_box">
    <button type="default">Content</button>
  </view>
  <view class="button_box">
    <button type="primary">Content</button>
  </view>
  <view class="button_box">
    <button type="warn">Content</button>
  </view>
  <view class="button_box">
    <button type="primary" plain="true">Content</button>
  </view>
  <view class="button_box">
    <button type="primary" disabled="true">Content</button>
  </view>
  <view class="button_box">
    <button type="primary" loading="true">Content</button>
  </view>
  <view class="button_box">
    <button type="primary" fromTyoe="reset">Content</button>
  </view>
  <view class="button_box">
    <button type="primary" hover-class="none">Content</button>
  </view>
</view>
```

运行效果如图 7-7 所示。

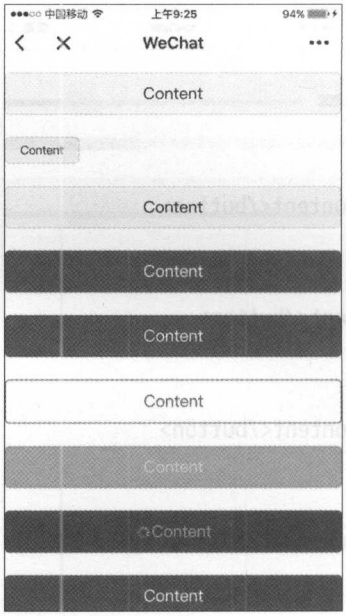

图 7-7 button 组件示例运行效果图

7.3.2 checkbox

checkbox 即复选框，为用户提供一系列选项，用户可以选择多个选项。checkbox-group 即多项选择器，内部由多个checkbox组成，如表 7-10 所示。

表 7-10

属性名	类型	默认值	说明
bindchange	EventHandle	—	在<checkbox-group/>中选中选项发生改变时触发 change 事件，detail = {value:[选中的 checkbox 的 value 的数组]}

checkbox 包含的属性如表 7-11 所示。

表 7-11

属性名	类型	默认值	说明
value	String	—	<checkbox/>标识，选中时触发<checkbox-group/>的 change 事件，并携带 <checkbox/>的 value
disabled	Boolean	false	是否禁用

续表

属性名	类型	默认值	说明
checked	Boolean	false	当前是否选中，可用来设置默认选中
color	Color	—	checkbox 的颜色，同 css 的 color

示例代码：

在 WXML 中：

```
<view class="container">
  <template is="head" data="{{title: 'checkbox'}}" />
  <view class="page-section-title">多选框默认样式
  </view>
  <label class="checkbox">
    <checkbox value="cb" checked="true" />项目一
  </label>
  <label class="checkbox">
    <checkbox value="cb" />项目二
  </label>
  <label class="checkbox">
    <checkbox value="cb" />项目三
  </label>
  <label class="checkbox">
    <checkbox value="cb" />项目四
  </label>
  <label class="checkbox">
    <checkbox value="cb" />项目五
  </label>
  <label class="checkbox">
    <checkbox value="cb" />项目六
  </label>
  <template is="foot" />
</view>
```

运行效果如图 7-8 所示。

7.3.3 radio

radio 即单选框，为用户提供一系列选项，用户可以选择其中一项，与多选框 checkbox 类似。radio-group 即单项选择器，内部由多个<radio/>组成，如表 7-12 所示。

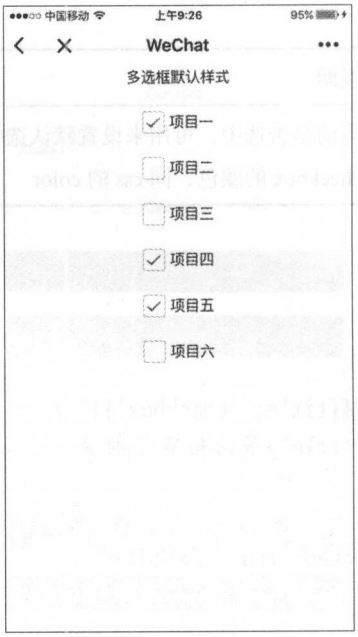

图 7-8 checkbox 组件示例运行效果图

表 7-12

属性名	类型	默认值	说明
bindchange	EventHandle	—	<radio-group/>中的选中项发生变化时触发 change 事件，event.detail = {value: 选中项 radio 的 value}

radio 包含的属性如表 7-13 所示。

表 7-13

属性名	类型	默认值	说明
value	String	—	<radio/>标识。当该<radio/>被选中时，<radio-group/>的 change 事件会携带<radio/>的 value
checked	Boolean	false	当前是否选中
disabled	Boolean	false	是否禁用
color	Color	—	radio 的颜色，同 CSS 的 color

示例代码：

在 WXML 中：

```
<radio-group class="radio-group" bindchange="radioChange">
  <label class="radio">
    <radio value="{{item.value}}" checked="true" />北京
  </label>
  <label class="radio">
    <radio value="{{item.value}}" checked="true" />天津
  </label>
  <label class="radio">
    <radio value="{{item.value}}" checked="true" />上海
  </label>
  <label class="radio">
    <radio value="{{item.value}}" checked="true" />重庆
  </label>
  <label class="radio">
    <radio value="{{item.value}}" checked="true" />深圳
  </label>
  <label class="radio">
    <radio value="{{item.value}}" checked="true" />广州
  </label>
</radio-group>
```

运行效果如图 7-9 所示。

图 7-9 radio 组件示例运行效果图

7.3.4 input

input 即输入框，类似于一个文本编辑器，用来和用户进行文本交互。input 所包含的属性如表 7-14 所示。

表 7-14

属性名	类型	默认值	说明
value	String	—	输入框的初始内容
type	String	text	input 的类型，有效值：text、number、idcard、digit
password	Boolean	false	是否是密码类型
placeholder	String	—	输入框为空时占位符
placeholder-style	String	—	指定 placeholder 的样式
placeholder-class	String	input-placeholder	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为 -1 的时候不限制最大长度
cursor-spacing	Number	0	指定光标与键盘的距离，单位：px。取 input 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离
auto-focus	Boolean	false	（即将废弃，请直接使用 focus）自动聚焦，拉起键盘
focus	Boolean	false	获取焦点
bindinput	EventHandle	—	当使用键盘输入时，触发 input 事件，event.detail = {value: value}，处理函数可以直接 return 一个字符串，将替换输入框的内容
bindfocus	EventHandle	—	输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle	—	输入框失去焦点时触发，event.detail = {value: value}
bindconfirm	EventHandle	—	点击完成按钮时触发，event.detail = {value: value}

示例代码：

在 WXML 中：

```
<view class="container">
  <template is="head" data="{{title: 'input'}}" />
  <view class="page-section">
    <view>可以自动聚焦的input</view>
    <view class="weui-cells">
      <input class="weui-input" auto-focus placeholder="将会获取焦点" />
    </view>
  </view>
  <view class="page-section">
    <view>控制最大输入长度的input</view>
    <view class="weui-cells">
      <input class="weui-input" maxlength="10" placeholder="最大输入长度为10" />
    </view>
  </view>
  <view class="page-section">
    <view>实时获取输入值: {{inputValue}}</view>
    <view class="weui-cells">
      <input class="weui-input" maxlength="10" bindinput="bindKeyInput" placeholder
          ="输入同步到view中" />
    </view>
  </view>
  <view class="page-section">
    <view>控制输入的input</view>
    <view class="weui-cells">
      <input class="weui-input" bindinput="bindReplaceInput" placeholder="连续的两
          个1会变成2" />
    </view>
  </view>
  <view class="page-section">
    <view>控制键盘的input</view>
    <view class="weui-cells">
      <input class="weui-input" bindinput="bindHideKeyboard" placeholder="输入123自
          动收起键盘" />
    </view>
  </view>
  <view class="page-section">
    <view>数字输入的input</view>
    <view class="weui-cells">
      <input class="weui-input" type="number" placeholder="这是一个数字输入框" />
```

```
    </view>
  </view>
  <view class="page-section">
    <view>密码输入的input</view>
    <view class="weui-cells">
      <input class="weui-input" password type="text" placeholder="这是一个密码输入
          框" />
    </view>
  </view>
  <view class="page-section">
    <view>带小数点的input</view>
    <view class="weui-cells">
      <input class="weui-input" type="digit" placeholder="带小数点的数字键盘" />
    </view>
  </view>
  <view class="page-section">
    <view>身份证输入的input</view>
    <view class="weui-cells">
      <input class="weui-input" type="idcard" placeholder="身份证输入键盘" />
    </view>
  </view>
  <view class="page-section">
    <view>控制占位符颜色的input</view>
    <view class="weui-cells">
      <input class="weui-input" placeholder-style="color:#F76260" placeholder="占位
          符字体是红色的" />
    </view>
  </view>
  <template is="foot" />
</view>
```

在JS中：

```
Page({
  data: {
    focus: false,
    inputValue: ''
  },
  bindKeyInput: function (e) {
    this.setData({
```

```
      inputValue: e.detail.value
    })
  },
  bindReplaceInput: function (e) {
    var value = e.detail.value
    var pos = e.detail.cursor
    var left
    if (pos !== -1) {
      // 光标在中间
      left = e.detail.value.slice(0, pos)
      // 计算光标的位置
      pos = left.replace(/11/g, '2').length
    }

    // 直接返回对象，可以对输入进行过滤处理，同时可以控制光标的位置
    return {
      value: value.replace(/11/g, '2'),
      cursor: pos
    }

    // 或者直接返回字符串,光标在最后边
    // return value.replace(/11/g,'2'),
  },
  bindHideKeyboard: function (e) {
    if (e.detail.value === '123') {
      // 收起键盘
      wx.hideKeyboard()
    }
  }
})
```

运行效果如图 7-10 所示。

注意：

- 在微信版本 6.3.30 中，focus 属性设置无效。
- 在微信版本 6.3.30 中，placeholder 在聚焦时出现重影问题。
- input 组件是一个 native 组件，字体是系统字体，所以无法设置 font-family。
- 在 input 聚焦期间，避免使用 CSS 动画。

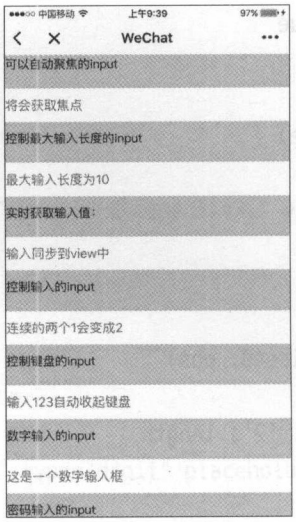

图 7-10　input 组件示例运行效果图

7.3.5　textarea

textarea 即多行输入框，与 input 类似，相当于文本编辑器，可以输入多行文字。

textarea 所包含的属性如表 7-15 所示。

表 7-15

属性名	类型	默认值	说明
value	String	—	输入框的内容
placeholder	String	—	输入框为空时占位符
placeholder-style	String	—	指定 placeholder 的样式
placeholder-class	String	textarea-placeholder	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为 -1 的时候不限制最大长度
auto-focus	Boolean	false	自动聚焦，拉起键盘
focus	Boolean	false	获取焦点
auto-height	Boolean	false	是否自动增高，设置为 auto-height 时，style.height 不生效
fixed	Boolean	false	如果 textarea 是在一个position:fixed的区域，则需要显示指定属性 fixed 为 true

续表

属性名	类型	默认值	说明
cursor-spacing	Number	0	指定光标与键盘的距离，单位为 px 。取 textarea 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离
bindfocus	EventHandle	—	输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle	—	输入框失去焦点时触发，event.detail = {value: value}
bindlinechange	EventHandle	—	输入框行数变化时调用，event.detail = {height: 0, heightRpx: 0, lineCount: 0}
bindinput	EventHandle	—	当键盘输入时，触发 input 事件，event.detail = {value: value}，bindinput 处理函数的返回值并不会反映到 textarea 上
bindconfirm	EventHandle	—	点击完成时，触发 confirm 事件，event.detail = {value: value}

示例代码：

在 WXML 中：

```
<view class="container">
  <template is="head" data="{{title: 'textarea'}}" />
  <view class="textarea-view">
    <view>输入区域高度自适应，不会出现滚动条</view>
    <view class="textarea-wrp">
      <textarea bindblur="bindTextAreaBlur" auto-height />
    </view>
  </view>
  <view class="textarea-view">
    <view>这是一个可以自动聚焦的textarea</view>
    <view class="textarea-wrp">
      <textarea auto-focus="true" style="height: 3em" />
    </view>
  </view>
  <template is="foot" />
</view>
```

在 JS 中：

```
Page({
  data: {
    focus: false
  },
  bindTextAreaBlur: function(e) {
    console.log(e.detail.value)
  }
})
```

运行效果如图 7-11 所示。

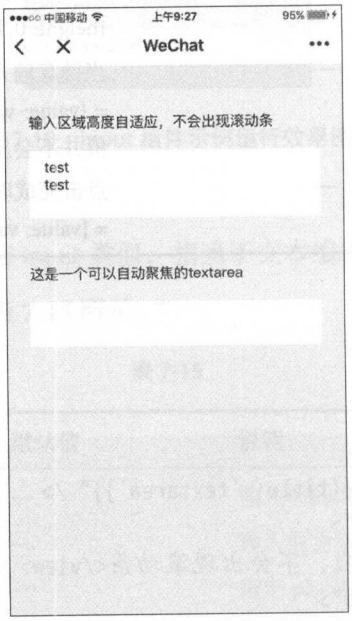

图 7-11　textarea 组件示例运行效果图

注意：

- 在微信版本 6.3.30 中，textarea 在列表渲染时，新增加的 textarea 在自动聚焦时的位置计算错误。
- textarea 的 blur 事件会晚于页面上的 tap 事件，如果需要在 button 的点击事件获取 textarea，则可以使用 form 的 bindsubmit。
- 不建议在多行文本上对用户的输入进行修改，所以 textarea 的 bindinput 处理函数并不会将返回值反映到 textarea 上。

- textarea 组件是由客户端创建的原生组件，它的层级是最高的。
- 不要在 scroll-view 中使用 textarea 组件。
- css 动画对 textarea 组件无效。

7.3.6 form

form 即表单，将组件内用户输入的<switch/> <input/> <checkbox/> <slider/> <radio/> <picker/> 标签进行提交。

当点击<form/>标签中 formType 为 submit 的<button/>组件时，会将表单组件中的 value 值进行提交，需要在表单组件中加上 name 来作为 key。

form 所包含的属性如表 7-16 所示。

表 7-16

属性名	类型	说明
report-submit	Boolean	是否返回 formId 用于发送模板消息
bindsubmit	EventHandle	携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'}, formId: ''}
bindreset	EventHandle	表单重置时会触发 reset 事件

示例代码：

在 WXML 中：

```
<view class="page-body">
    <form catchsubmit="formSubmit" catchreset="formReset">
      <view class="page-section page-section-gap">
        <view class="page-section-title">switch</view>
        <switch name="switch"/>
      </view>

      <view class="page-section page-section-gap">
        <view class="page-section-title">radio</view>
        <radio-group name="radio">
          <label><radio value="radio1"/>选项一</label>
          <label><radio value="radio2"/>选项二</label>
        </radio-group>
      </view>
```

```
    <view class="page-section page-section-gap">
      <view class="page-section-title">checkbox</view>
      <checkbox-group name="checkbox">
        <label><checkbox value="checkbox1"/>选项一</label>
        <label><checkbox value="checkbox2"/>选项二</label>
      </checkbox-group>
    </view>

    <view class="page-section page-section-gap">
      <view class="page-section-title">slider</view>
      <slider value="50" name="slider" show-value ></slider>
    </view>

    <view class="page-section">
      <view class="page-section-title">input</view>
      <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
          <view class="weui-cell__bd">
            <input class="weui-input" name="input" placeholder="这是一个输入框"
                />
          </view>
        </view>
      </view>
    </view>

    <view class="btn-area">
      <button type="primary" formType="submit">Submit</button>
      <button formType="reset">Reset</button>
    </view>
  </form>
</view>
```

在 JS 中：

```
Page({
  data: {
    pickerHidden: true,
    chosen: ''
  },
```

```
  pickerConfirm: function (e) {
    this.setData({
      pickerHidden: true
    })
    this.setData({
      chosen: e.detail.value
    })
  },
  pickerCancel: function (e) {
    this.setData({
      pickerHidden: true
    })
  },
  pickerShow: function (e) {
    this.setData({
      pickerHidden: false
    })
  },
  formSubmit: function (e) {
    console.log('form发生了submit事件，携带数据为：', e.detail.value)
  },
  formReset: function (e) {
    console.log('form发生了reset事件，携带数据为：', e.detail.value)
    this.setData({
      chosen: ''
    })
  }
})
```

运行效果如图 7-12 所示。

7.3.7 label

label 用于改进表单组件的可用性，使用for属性找到对应的id，或者将控件放在该标签下，当点击时，就会触发对应的控件。

for的优先级高于内部控件，内部有多个控件的时候默认触发第一个控件。

目前可以绑定的控件有：<button/>, <checkbox/>, <radio/>, <switch/>组件。

label 只包含一个for属性，如表 7-17 所示。

图 7-12　form 组件示例运行效果图

表 7-17

属性名	类型	说明
for	String	绑定控件的 id

示例代码：

在 WXML 中：

```
<view class="page-body">
  <view class="page-section page-section-gap">
    <view class="page-section-title">表单组件在label内</view>
    <checkbox-group class="group" bindchange="checkboxChange">
      <view class="label-1" wx:for="{{checkboxItems}}">
        <label>
          <checkbox value="{{item.name}}" checked="{{item.checked}}"></checkbox>
          <text class="label-1-text">{{item.value}}</text>
        </label>
      </view>
```

```
    </checkbox-group>
  </view>

  <view class="page-section page-section-gap">
    <view class="page-section-title">label用for标识表单组件</view>
    <radio-group class="group" bindchange="radioChange">
      <view class="label-2" wx:for="{{radioItems}}">
        <radio id="{{item.name}}" value="{{item.name}}" checked="{{item.checked}}
            "></radio>
        <label class="label-2-text" for="{{item.name}}"><text>{{item.name}}</text
            ></label>
      </view>
    </radio-group>
  </view>

  <view class="page-section page-section-gap">
    <view class="page-section-title">label内有多个时选中第一个</view>
    <label class="label-3">
      <checkbox class="checkbox-3">选项一</checkbox>
      <checkbox class="checkbox-3">选项二</checkbox>
    </label>
  </view>
</view>
```

在JS中：

```
Page({
  data: {
    checkboxItems: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'}
    ],
    radioItems: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'}
    ],
    hidden: false
  },
  checkboxChange: function (e) {
    var checked = e.detail.value
```

```
    var changed = {}
    for (var i = 0; i < this.data.checkboxItems.length; i++) {
      if (checked.indexOf(this.data.checkboxItems[i].name) !== -1) {
        changed['checkboxItems[' + i + '].checked'] = true
      } else {
        changed['checkboxItems[' + i + '].checked'] = false
      }
    }
    this.setData(changed)
  },
  radioChange: function (e) {
    var checked = e.detail.value
    var changed = {}
    for (var i = 0; i < this.data.radioItems.length; i ++) {
      if (checked.indexOf(this.data.radioItems[i].name) !== -1) {
        changed['radioItems[' + i + '].checked'] = true
      } else {
        changed['radioItems[' + i + '].checked'] = false
      }
    }
    this.setData(changed)
  },
  tapEvent: function (e) {
    console.log('按钮被点击')
  }
})
```

运行效果如图 7-13 所示。

7.3.8 picker

picker 即滚动选择器，从底部弹出，现支持三种样式并通过 mode 来区分，分别是普通选择器、时间选择器和日期选择器，默认是普通选择器（mode = selector）。

普通选择器（mode = selector）所包含的属性如表 7-18 所示。

表 7-18

属性名	类型	默认值	说明
range	Array / Object Array	[]	mode 为 selector 时，range 有效

续表

属性名	类型	默认值	说明
range-key	String	—	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	Number	0	value 的值表示选择了 range 中的第几个（下标从 0 开始）
bindchange	EventHandle	—	value 改变时触发 change 事件，event.detail = {value: value}
disabled	Boolean	false	是否禁用

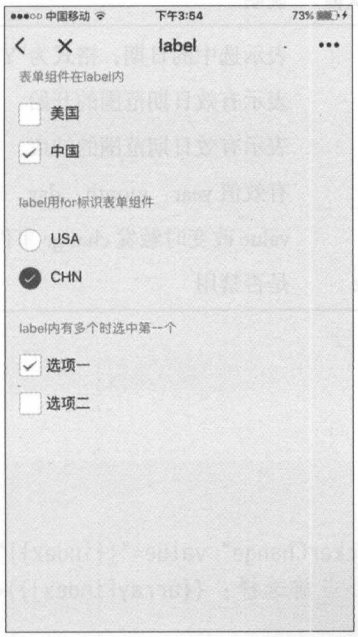

图 7-13 label 组件示例运行效果图

时间选择器（mode = time）所包含的属性如表 7-19 所示。

表 7-19

属性名	类型	默认值	说明
value	String	—	表示选中的时间，格式为"hh:mm"
start	String	—	表示有效时间范围的开始，字符串格式为"hh:mm"

续表

属性名	类型	默认值	说明
end	String	—	表示有效时间范围的结束，字符串格式为"hh:mm"
bindchange	EventHandle	—	value 改变时触发 change 事件，event.detail = {value: value}
disabled	Boolean	false	是否禁用

日期选择器（mode = date）所包含的属性如表 7-20 所示。

表 7-20

属性名	类型	默认值	说明
value	String	0	表示选中的日期，格式为"YYYY-MM-DD"
start	String	—	表示有效日期范围的开始，字符串格式为"YYYY-MM-DD"
end	String	—	表示有效日期范围的结束，字符串格式为"YYYY-MM-DD"
fields	String	day	有效值 year、month、day，表示选择器的粒度
bindchange	EventHandle	—	value 改变时触发 change 事件，event.detail = {value: value}
disabled	Boolean	false	是否禁用

示例代码：

在 WXML 中：

```
<view class="picker-view">
  <view>地区选择器</view>
  <picker bindchange="bindPickerChange" value="{{index}}" range="{{array}}">
    <view class="picker-wrp">当前选择：{{array[index]}}</view>
  </picker>
</view>
<view class="picker-view">
  <view>时间选择器</view>
  <picker mode="time" value="{{time}}" start="09:01" end="21:01" bindchange="
      bindTimeChange">
    <view class="picker-wrp">当前选择：{{time}}</view>
  </picker>
</view>

<view class="picker-view">
```

```
  <view>日期选择器</view>
  <picker mode="date" value="{{date}}" start="2015-09-01" end="2017-09-01"
      bindchange="bindDateChange">
    <view class="picker-wrp">当前选择: {{date}}</view>
  </picker>
</view>
```

在JS中：

```
Page({
  data: {
    array: ['美国', '中国', '巴西', '日本'],
    objectArray: [
      {
        id: 0,
        name: '美国'
      },
      {
        id: 1,
        name: '中国'
      },
      {
        id: 2,
        name: '巴西'
      },
      {
        id: 3,
        name: '日本'
      }
    ],
    index: 0,
    date: '2016-09-01',
    time: '12:01'
  },
  bindPickerChange: function(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      index: e.detail.value
    })
  },
```

```
  bindDateChange: function(e) {
    this.setData({
      date: e.detail.value
    })
  },
  bindTimeChange: function(e) {
    this.setData({
      time: e.detail.value
    })
  }
})
```

运行效果如图 7-14 所示。

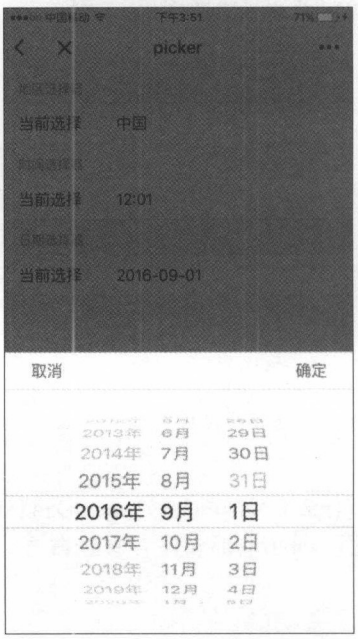

图 7-14 picker 组件示例运行效果图

7.3.9 picker-view

picker-view 即嵌入页面的滚动选择器，可以自定义显示内容和样式。

picker-view 所包含的属性如表 7-21 所示。

表 7-21

属性名	类型	说明
value	Number Array	数组中的数字依次表示 picker-view 内的 picker-view-colume 选择的第几项（下标从 0 开始），数字大于 picker-view-column 可选项长度时，选择最后一项
indicator-style	String	设置选择器中间选中框的样式
bindchange	EventHandle	当滚动选择，value 改变时触发 change 事件，event.detail = {value: value}；value 为数组，表示 picker-view 内的 picker-view-column 当前选择的是第几项（下标从 0 开始）

示例代码：

在 WXML 中：

```
<view>
  <view>{{year}}年{{month}}月{{day}}日</view>
  <picker-view indicator-style="height: 50px;" style="width: 100%; height: 300px;"
      value="{{value}}" bindchange="bindChange">
    <picker-view-column>
      <view wx:for="{{years}}" style="line-height: 50px">{{item}}年</view>
    </picker-view-column>
    <picker-view-column>
      <view wx:for="{{months}}" style="line-height: 50px">{{item}}月</view>
    </picker-view-column>
    <picker-view-column>
      <view wx:for="{{days}}" style="line-height: 50px">{{item}}日</view>
    </picker-view-column>
  </picker-view>
</view>
```

在 JS 中：

```
const date = new Date()
const years = []
const months = []
const days = []

for (let i = 1990; i <= date.getFullYear(); i++) {
  years.push(i)
```

```
}

for (let i = 1 ; i <= 12; i++) {
  months.push(i)
}

for (let i = 1 ; i <= 31; i++) {
  days.push(i)
}

Page({
  data: {
    years: years,
    year: date.getFullYear(),
    months: months,
    month: 2,
    days: days,
    day: 2,
    year: date.getFullYear(),
    value: [9999, 1, 1],
  },
  bindChange: function(e) {
    const val = e.detail.value
    this.setData({
      year: this.data.years[val[0]],
      month: this.data.months[val[1]],
      day: this.data.days[val[2]]
    })
  }
})
```

注意：

在 picker-view 中只可放置<picker-view-column/>组件，放置其他标签不会显示。picker-view-column 其孩子节点的高度会自动设置成与 picker-view 的选中框的高度一致。

7.3.10 slider

slider 即滑块选择器。slider 所包含的属性如表 7-22 所示。

表 7-22

属性名	类型	默认值	说明
min	Number	0	最小值
disabled	Boolean	false	是否禁用
value	Number	0	当前取值
color	Color	#e9e9e9	背景条的颜色
selected-color	Color	#1aad19	已选择的颜色
show-value	Boolean	false	是否显示当前 value
bindchange	EventHandle	—	完成一次拖动后触发的事件，event.detail = {value: value}

示例代码：

在 WXML 中：

```
<view class="slider">
  <text>设置left/right icon</text>
  <slider bindchange="slider1change" left-icon="cancel" right-icon="
      success_no_circle" />
</view>
<view class="slider">
  <text>设置step</text>
  <slider bindchange="slider2change" step="5" />
</view>
<view class="slider">
  <text>显示当前value</text>
  <slider bindchange="slider3change" show-value/>
</view>
<view class="slider">
  <text>设置最小/最大值</text>
  <slider bindchange="slider4change" min="50" max="200" show-value/>
</view>
```

在 JS 中：

```
var pageData = {}
for(var i = 1; i < 5; ++i) {
  (function (index) {
    pageData[`slider${index}change`] = function(e) {
      console.log(`slider${index}发生change事件，携带值为`, e.detail.value)
```

```
    }
  })(i);
}
Page(pageData)
```

运行效果如图 7-15 所示。

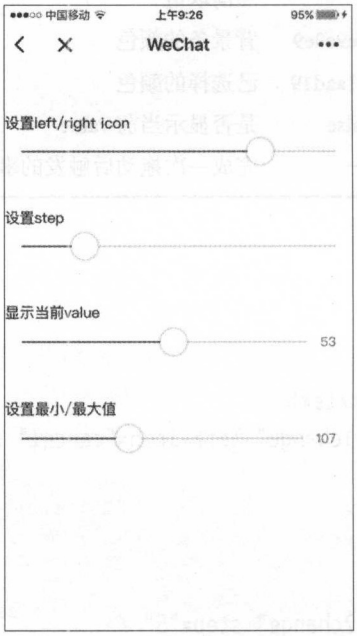

图 7-15 slider 组件示例运行效果图

7.3.11 switch

switch 即开关选择器。switch 所包含的属性如表 7-23 所示。

表 7-23

属性名	类型	默认值	说明
checked	Boolean	false	是否选中
type	String	switch	样式，有效值：switch, checkbox
bindchange	EventHandle		checked 改变时触发 change 事件，event.detail={ value:checked}
color	Color		switch 的颜色，同 CSS 的 color

示例代码：

在 WXML 中：

```
<view class="switch-view">
  <view>type="switch"</view>
  <view class="switch">
    <switch checked="{{switch1Checked}}" bindchange="switch1Change" />
  </view>
</view>
<view class="switch-view">
  <view>type="checkbox"</view>
  <view class="switch">
    <switch type="checkbox" checked="{{switch2Checked}}" bindchange="
        switch2Change" />
  </view>
</view>
```

在 JS 中：

```
var pageData = {
  data: {
    switch1Checked: true,
    switch2Checked: false,
    switch1Style: '',
    switch2Style: 'text-decoration: line-through'
  }
}
for(var i = 1; i <= 2; ++i) {
  (function(index) {
    pageData[`switch${index}Change`] = function(e) {
      console.log(`switch${index}发生change事件，携带值为`, e.detail.value)
      var obj = {}
      obj[`switch${index}Checked`] = e.detail.value
      this.setData(obj)
      obj = {}
      obj[`switch${index}Style`] = e.detail.value ? '' : 'text-decoration: line-
          through'
      this.setData(obj)
    }
  })(i)
```

```
}
Page(pageData)
```

运行效果如图 7-16 所示。

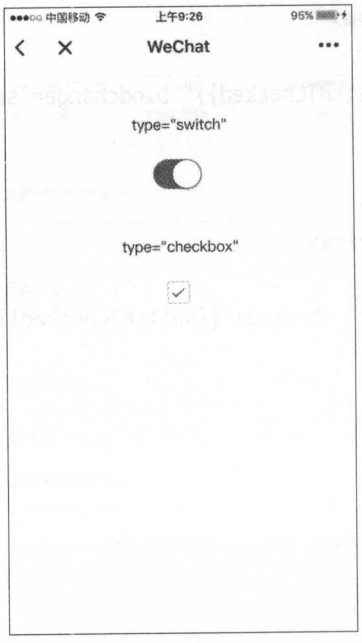

图 7-16　switch 组件示例运行效果图

7.4　多媒体组件

多媒体组件可以在页面中加载图片、音频、视频，并且能够控制显示方式和进程，使页面更加多元化，更具吸引力。多媒体组件包含 image、audio 和 video 3 种组件。

7.4.1　image

image 即图片，用于界面显示图片，是程序中比较常见的组件。它的默认宽度为 300px，高度为 225px。image 所包含的属性如表 7-24 所示。

表 7-24

属性名	类型	默认值	说明
src	String		图片资源地址

续表

属性名	类型	默认值	说明
mode	String	'scaleToFill'	图片裁剪、缩放的模式
binderror	HandleEvent	—	当错误发生时，发布到 AppService 的事件名，事件对象 event.detail = {errMsg: 'something wrong'}
bindload	HandleEvent	—	当图片载入完毕时，发布到 AppService 的事件名，事件对象 event.detail = {height:' 图片高度 px', width:' 图片宽度 px'}

其中，mode 有 13 种模式，其中 4 种是缩放模式，9 种是裁剪模式。

缩放模式如表 7-25 所示。

表 7-25

模式	说明
scaleToFill	不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
aspectFit	保持纵横比缩放图片，使图片的长边能完全显示出来。也就是说，可以完整地将图片显示出来
aspectFill	保持纵横比缩放图片，只保证图片的短边能完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取
widthFix	宽度不变，高度自动变化，保持原图宽高比不变

裁剪模式如表 7-26 所示。

表 7-26

模式	说明
top	不缩放图片，只显示图片的顶部区域
bottom	不缩放图片，只显示图片的底部区域
center	不缩放图片，只显示图片的中间区域
left	不缩放图片，只显示图片的左边区域
right	不缩放图片，只显示图片的右边区域
top left	不缩放图片，只显示图片的左上边区域
top right	不缩放图片，只显示图片的右上边区域

续表

模式	说明
bottom left	不缩放图片，只显示图片的左下边区域
bottom right	不缩放图片，只显示图片的右下边区域

示例代码：

在WXML中：

```
<view wx:for="{{array}}" wx:for-item="item">
  <view>{{item.text}}</view>
  <view class="image">
    <image style="width: 200px; height: 200px; background-color: #eeeeee;" mode="{{
        item.mode}}" src="{{src}}"></image>
  </view>
</view>
```

在JS中：

```
Page({
  data:{
    array: [{
      mode: 'aspectFit',
      text: 'aspectFit: 保持纵横比缩放图片，使图片完整的显示出来'
    }, {
      mode: 'scaleToFill',
      text: 'scaleToFill: 不保持纵横比缩放图片，使图片拉伸适应'
    }, {
      mode: 'aspectFill',
      text: 'aspectFill: 保持纵横比缩放图片，只保证图片的短边能完全显示出来'
    }, {
      mode: 'top',
      text: 'top: 不缩放图片，只显示图片的顶部区域'
    }, {
      mode: 'bottom',
      text: 'bottom: 不缩放图片，只显示图片的底部区域'
    }, {
      mode: 'center',
      text: 'center: 不缩放图片，只显示图片的中间区域'
    }, {
```

```
      mode: 'left',
      text: 'left: 不缩放图片，只显示图片的左边区域'
    }, {
      mode: 'right',
      text: 'right: 不缩放图片，只显示图片的右边区域'
    }, {
      mode: 'top left',
      text: 'top left: 不缩放图片，只显示图片的左上边区域'
    }, {
      mode: 'top right',
      text: 'top right: 不缩放图片，只显示图片的右上边区域'
    }, {
      mode: 'bottom left',
      text: 'bottom left: 不缩放图片，只显示图片的左下边区域'
    }, {
      mode: 'bottom right',
      text: 'bottom right: 不缩放图片，只显示图片的右下边区域'
    }],
    src: '../resources/testImage.png'
  },
  imageError: function(e) {
    console.log('image3发生error事件，携带值为', e.detail.errMsg)
  },

})
```

运行效果如图 7-17 所示。

7.4.2 audio

audio 即音频，用来加入一段音频，使页面更加丰富。

audio 所包含的属性如表 7-27 所示。

表 7-27

属性名	类型	默认值	说明
id	String	—	video 组件的唯一标识符
src	String	—	要播放音频的资源地址
loop	Boolean	false	是否循环播放

续表

属性名	类型	默认值	说明
controls	Boolean	true	是否显示默认控件
poster	String	—	默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false，则设置 poster 无效
name	String	未知音频	默认控件上的音频名字，如果 controls 属性值为 false，则设置 name 无效
author	String	未知作者	默认控件上的作者名字，如果 controls 属性值为 false，则设置 author 无效
binderror	EventHandle	—	当发生错误时触发 error 事件，detail = {errMsg: MediaError.code}
bindplay	EventHandle	—	当开始/继续播放时触发 play 事件
bindpause	EventHandle	—	当暂停播放时触发 pause 事件
bindtimeupdate	EventHandle	—	当播放进度改变时触发 timeupdate 事件，detail = {currentTime, duration}
bindended	EventHandle	—	当播放到末尾时触发 ended 事件

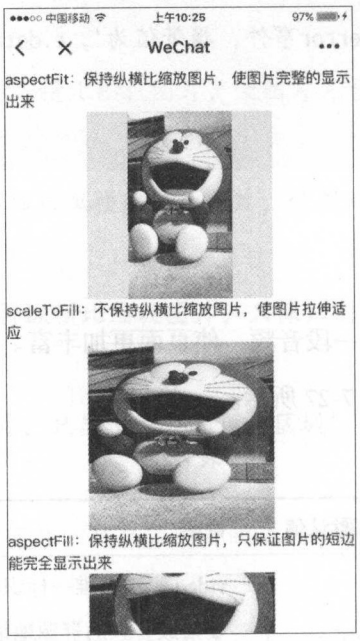

图 7-17　image 组件示例运行效果图

MediaError.code 的错误码如表 7-28 所示。

表 7-28

错误码	说明
MEDIA_ERR_ABORTED	获取资源被用户禁止
MEDIA_ERR_NETWORD	网络错误
MEDIA_ERR_DECODE	解码错误
MEDIA_ERR_SRC_NOT_SUPPOERTED	不合适资源

示例代码：

在 WXML 中：

```
<view class="audio-view">
  <audio poster="{{poster}}" name="{{name}}" author="{{author}}" src="{{src}}" id="
      myAudio" controls loop></audio>
</view>
<view class="audio-view">
  <button class="button" type="default" bindtap="audioPlay">播放</button>
  <button class="button" type="default" bindtap="audioPause">暂停</button>
  <button class="button" type="default" bindtap="audio14">设置当前播放时间为14秒<
      /button>
  <button class="button" type="default" bindtap="audioStart">回到开头</button>
</view>
```

在 JS 中：

```
Page({
  data:{
    poster: '音频封面地址',
    name: '音频名称',
    author: '音频作者',
    src: '音频资源地址',
  },
  audioPlay: function () {
    this.audioCtx.play()
  },
  audioPause: function () {
    this.audioCtx.pause()
```

```
  },
  audio14: function () {
    this.audioCtx.seek(14)
  },
  audioStart: function () {
    this.audioCtx.seek(0)
  }
})
```

运行效果如图 7-18 所示。

图 7-18 audio 组件示例运行效果图

7.4.3 video

video 即视频，用来加入视频播放功能，使页面更加丰富。video 标签默认宽度为 300px，高度为 225px，设置宽高需要通过 WXSS 设置 width 和 height。

video 所包含的属性如表 7-29 所示。

表 7-29

属性名	类型	默认值	说明
src	String	—	要播放视频的资源地址
controls	Boolean	true	是否显示默认播放控件（播放/暂停按钮、播放进度、时间）
danmu-list	Object Array	—	弹幕列表
danmu-btn	Boolean	false	是否显示弹幕按钮，只在初始化时有效，不能动态变更
enable-danmu	Boolean	false	是否展示弹幕，只在初始化时有效，不能动态变更
autoplay	Boolean	false	是否自动播放
bindplay	EventHandle	—	当开始/继续播放时触发 play 事件
bindpause	EventHandle	—	当暂停播放时触发 pause 事件
bindended	EventHandle	—	当播放到末尾时触发 ended 事件
bindtimeupdate	EventHandle	—	播放进度变化时触发，event.detail = {currentTime: ' 当前播放时间'} 。触发频率应该在 250ms 一次
objectFit	String	contain	当视频大小与 video 容器大小不一致时，视频的表现形式。contain：包含，fill：填充，cover：覆盖

示例代码：

在 WXML 中：

```
<view class="video-view">
  <video id="myVideo" src="视频资源地址"
  danmu-list="{{danmuList}}" enable-danmu danmu-btn controls></video>
</view>
<view class="view">
  <button bindtap="bindButtonTap">获取视频</button>
  <view class="view">
    <view class="weui-cell__hd">
      <view class="weui-label">弹幕内容</view>
    </view>
    <view class="weui-cell__bd">
      <input bindblur="bindInputBlur" class="input-view" type="text" placeholder="
          在此处输入弹幕内容" />
    </view>
  </view>
```

```
  <button class="view" type="primary" bindtap="bindSendDanmu">发送弹幕</button>
</view>
```

在 JS 中：

```
function getRandomColor() {
  let rgb = []
  for (let i = 0; i < 3; ++i) {
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length == 1 ? '0' + color : color
    rgb.push(color)
  }
  return '#' + rgb.join('')
}
Page({
  inputValue: '',
  data: {
    src: '',
    danmuList: [
      {
        text: '第 1s 出现的弹幕',
        color: '#ff0000',
        time: 1
      },
      {
        text: '第 3s 出现的弹幕',
        color: '#ff00ff',
        time: 3
      }]
  },
  bindButtonTap: function () {
    var that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: ['front', 'back'],
      success: function (res) {
        that.setData({
          src: res.tempFilePath
        })
```

```
      }
    })
  },
  bindSendDanmu: function () {
    this.videoContext.sendDanmu({
      text: this.inputValue,
      color: getRandomColor()
    })
  }
})
```

运行效果如图 7-19 所示。

图 7-19 video 组件示例运行效果图

注意：

- video组件是由客户端创建的原生组件，它的层级是最高的。
- 请勿在scroll-view中使用video组件。
- CSS 动画对video组件无效。

7.5 地图组件

本节介绍地图组件，微信小程序提供了地图导航功能，使用地图组件和地图位置 API 就可以方便、快捷地实现地图定位导航。微信小程序可以直接使用微信内置地图，这样大大节省了设备的内存空间，使用户在不降低体验的情况下减少内存消耗。

map

map 即地图，可以在界面中显示地图，最常用于 LBS 服务和位置导航。map 所包含的属性如表 7-30 所示。

表 7-30

属性名	类型	默认值	说明
longitude	Number	—	中心经度
latitude	Number	—	中心纬度
scale	Number	16	缩放级别，取值范围为 5~18
markers	Array	—	标记点
covers	Array	—	即将移除，请使用 markers
polyline	Array	—	路线
circles	Array	—	圆形
controls	Array	—	控件
include-points	Array	—	缩放视野以包含所有给定的坐标点
show-location	Boolean	—	显示带有方向的当前定位点
bindmarkertap	EventHandle	—	点击标记点时触发
bindcontroltap	EventHandle	—	点击控件时触发
bindregionchange	EventHandle	—	视野发生变化时触发
bindtap	EventHandle	—	点击地图时触发

markers 即标记点，用于在地图上显示标记的位置。covers 属性即将移除，请使用 markers 替代。

markers 所包含的属性如表 7-31 所示。

表 7-31

属性名	类型	必填	说明
id	Number	否	标记点 id，marker 点击事件回调会返回此 id
latitude	Number	是	纬度，浮点数，范围：-90 ~ 90
longitude	Number	是	纬度，浮点数，范围：-180 ~ 180
title	String	否	标注点名
iconPath	String	是	显示的图标，项目目录下的图片路径，支持相对路径写法，以'/' 开头则表示相对小程序根目录
rotate	Number	否	旋转角度，顺时针旋转的角度，范围：0 ~ 360，默认为 0
alpha	Number	否	标注的透明度，默认 1，无透明
width	Number	否	标注图标宽度，默认为图片实际宽度
height	Number	否	标注图标高度，默认为图片实际高度

polyline 即指定一系列坐标点，从数组第一项连线至最后一项。

polyline 所包含的属性如表 7-32 所示。

表 7-32

属性名	类型	必填	说明
points	Array	是	经纬度数组，[{latitude: 0, longitude: 0}]
color	String	否	线的颜色，8 位十六进制表示，后两位表示 alpha 值，如：#000000AA
width	Number	否	线的宽度
dottedLine	Boolean	否	是否虚线，默认 false

circles 即在地图上显示圆。

circles 所包含的属性如表 7-33 所示。

表 7-33

属性名	类型	必填	说明
latitude	Number	是	纬度，浮点数，范围：-90 ~ 90
longitude	Number	是	经度，浮点数，范围：-180 ~ 180

续表

属性名	类型	必填	说明
color	String	否	描边的颜色，用 8 位十六进制数表示，后两位表示 Alpha 值，如：#000000AA
fillColor	String	否	填充颜色，用 8 位十六进制数表示，后两位表示 Alpha 值，如：#000000AA
radius	Number	是	半径
strokeWidth	Number	否	描边的宽度

controls 即在地图上显示控件，控件不随着地图移动。

controls 所包含的属性如表 7-34 所示。

表 7-34

属性名	类型	必填	说明
id	Number	否	控件 id，在控件点击事件回调会返回此 id
position	Object	是	控件在地图的相对位置
iconPath	String	是	显示的图标，项目目录下的图片路径，支持相对路径写法，以'/' 开头则表示相对小程序根目录
clickable	Boolean	否	是否可点击，默认不可点击

position 即位置。

position 所包含的属性如表 7-35 所示。

表 7-35

属性名	类型	默认值	必填	说明
left	Number	0	否	距离地图的左边界多远
top	Number	0	否	距离地图的上边界多远
width	Number	图片宽度	否	控件宽度
height	Number	图片高度	否	控件高度

示例代码：

在 WXML 中：

```
<map id="map" longitude="117.20" latitude="39.06" scale="14" controls="{{controls}}
    " bindcontroltap="controltap" markers="{{markers}}" bindmarkertap="markertap"
    polyline="{{polyline}}" bindregionchange="regionchange" show-location style="
    width: 100%; height: 300px;"></map>
```

运行效果如图 7-20 所示。

图 7-20 map 组件示例运行效果图

注意：

- map组件是由客户端创建的原生组件，它的层级是最高的。
- 请勿在scroll-view中使用map组件。
- css 动画对map组件无效。

7.6 导航组件

本节介绍导航组件，它是微信小程序中用来控制所有页面顺序的组件。可以帮助我们实现页面的路由和跳转，使用起来相当便捷。

navigator

navigator即导航组件，类似于HTML中的<a>标签，但无target属性，默认跳转到新页面，redirect为在当前页面打开。navigator所包含的属性如表7-36所示。

表7-36

属性名	类型	默认值	说明
url	String	—	应用内的跳转链接
redirect	Boolean	false	打开方式为页面重定向，对应wx.redirectTo（将被废弃，推荐使用open-type）
open-type	String	navigate	可选值'navigate'、'redirect'、'switchTab'，对应于wx.navigateTo、wx.redirectTo、wx.switchTab的功能
hover-class	String	navigator-hover	指定点击时的样式类，当hover-class="none"时，没有点击态效果
hover-start-time	Number	50	按住后多久出现点击态，单位为毫秒
hover-stay-time	Number	600	手指松开后点击态保留时间，单位为毫秒

注意：navigator-hover默认为{background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}，<navigator/>的子节点背景色应为透明色。

示例代码：

```
<view class="viewName">
   <navigator url="../image/image" hover-class="navigator-hover">
     <text class="view-Name">image展示</text>
   </navigator>
 </view>
  <view class="viewName">
   <navigator url="../audio/audio" hover-class="navigator-hover">
     <text class="view-Name">audio展示</text>
   </navigator>
 </view>
 <view class="viewName">
   <navigator url="../video/video" hover-class="navigator-hover">
     <text class="view-Name">video展示</text>
   </navigator>
 </view>
```

7.7 画布组件

本节介绍画布组件，它可用于实现一些小程序控件没有提供的页面元素，开发者可以自由地使用画布画出想要的页面效果。

canvas

canvas 即画布组件，可用于画出任意形状，实现页面动画，canvas 标签默认宽度为 300px，高度为 225px。并且同一页面中的 canvas-id 不可重复，如果使用一个已经出现过的 canvas-id，则该 canvas 标签对应的画布将被隐藏并不再正常工作。canvas 所包含的属性如表 7-37 所示。

表 7-37

属性名	类型	默认值	说明
canvas-id	String	—	canvas 组件的唯一标识符
disable-scroll	Boolean	false	当在 canvas 中移动时，禁止屏幕滚动以及下拉刷新
bindtouchstart	EventHandle	—	手指触摸动作开始
bindtouchmove	EventHandle	—	手指触摸后移动
bindtouchend	EventHandle	—	手指触摸动作结束
bindtouchcancel	EventHandle	—	手指触摸动作被打断，如来电提醒，弹窗
bindlongtap	EventHandle	—	手指长按 500ms 之后触发，触发了长按事件后进行移动不会触发屏幕的滚动
binderror	EventHandle	—	当发生错误时触发 error 事件，detail = {errMsg: 'something wrong'}

示例代码：

在 WXML 中：

```
<view class="container">
  <view class="canvas_area">
    <!--注意：同一页面中的 canvas-id 不可重复，如果使用一个已经出现过的 canvas
        -id，该 canvas 标签对应的画布将被隐藏并不再正常工作-->
    <canvas canvas-id="myCanvas" class="myCanvas" disable-scroll="false"
        bindtouchstart="touchStart" bindtouchmove="touchMove" bindtouchend="
        touchEnd">
    </canvas>
```

```
  </view>
  <view class="canvas_tools">
    <view class="box box1" bindtap="penSelect" data-param="5">细体</view>
    <view class="box box2" bindtap="penSelect" data-param="15">粗体</view>
    <view class="box box3" bindtap="colorSelect" data-param="#FF3300"></view>
    <view class="box box4" bindtap="colorSelect" data-param="#FFFF00"></view>
    <view class="box box5" bindtap="clearCanvas">清除</view>
  </view>
</view>
```

在JS中：

```
Page({
  data: {
    pen: 3, //画笔粗细默认值
    color: '#33CC00' //画笔颜色默认值
  },
  startX: 0, //保存X坐标轴变量
  startY: 0, //保存X坐标轴变量
  isClear: false, //是否启用橡皮擦标记
  //手指触摸动作开始
  touchStart: function (e) {
    //得到触摸点的坐标
    this.startX = e.changedTouches[0].x
    this.startY = e.changedTouches[0].y
    this.context = wx.createContext()

    if (this.isClear) { //判断是否启用的橡皮擦功能：ture表示清除，false表示画画
      this.context.setStrokeStyle('#F8F8F8') //设置线条样式，此处设置为画布的背
          景颜色，清除按钮原理就是：利用擦过的地方被填充为与画布的背景颜色一
          致，从而达到清除的效果
      this.context.setLineCap('round') //设置线条端点的样式
      this.context.setLineJoin('round') //设置两线相交处的样式
      this.context.setLineWidth(20) //设置线条宽度
      this.context.save();  //保存当前坐标轴的缩放、旋转、平移信息
      this.context.beginPath() //开始一个路径
      this.context.arc(this.startX, this.startY, 5, 0, 2 * Math.PI, true);  //添加
          一个弧形路径到当前路径，顺时针绘制。这里总共画了360度，也就是一个
          圆形
      this.context.fill();  //对当前路径进行填充
```

```
      this.context.restore();  //恢复之前保存过的坐标轴的缩放、旋转、平移信息
    } else {
      this.context.setStrokeStyle(this.data.color)
      this.context.setLineWidth(this.data.pen)
      this.context.setLineCap('round') // 让线条圆润
      this.context.beginPath()
    }
  },
  //手指触摸后移动
  touchMove: function (e) {
    var startX1 = e.changedTouches[0].x
    var startY1 = e.changedTouches[0].y
    if (this.isClear) { //判断是否启用的橡皮擦功能：ture表示清除，false表示画画
      this.context.save();  //保存当前坐标轴的缩放、旋转、平移信息
      this.context.moveTo(this.startX, this.startY);  //把路径移动到画布中的指定
          点，但不创建线条
      this.context.lineTo(startX1, startY1);  //添加一个新点，然后在画布中创建从
          该点到最后指定点的线条
      this.context.stroke();  //对当前路径进行描边
      this.context.restore()  //恢复之前保存过的坐标轴的缩放、旋转、平移信息
      this.startX = startX1;
      this.startY = startY1;
    } else {
      this.context.moveTo(this.startX, this.startY)
      this.context.lineTo(startX1, startY1)
      this.context.stroke()
      this.startX = startX1;
      this.startY = startY1;
    }
    //只是一个记录方法调用的容器，用于生成记录绘制行为的actions数组。context
        与<canvas/>不存在对应关系，一个context生成画布的绘制动作数组可以应用
        于多个<canvas/>
    wx.drawCanvas({
      canvasId: 'myCanvas',
      reserve: true,
      actions: this.context.getActions() // 获取绘图动作数组
    })
  },
  //手指触摸动作结束
```

```
  touchEnd: function () {

  },
  //启动清除按钮方法
  clearCanvas: function () {
    if (this.isClear) {
      this.isClear = false;
    } else {
      this.isClear = true;
    }
  },
  //更改画笔大小的方法
  penSelect: function (e) {
    console.log(e.currentTarget);
    this.setData({ pen: parseInt(e.currentTarget.dataset.param) });
    this.isClear = false;
  },
  //更改画笔颜色的方法
  colorSelect: function (e) {
    console.log(e.currentTarget);
    this.setData({ color: e.currentTarget.dataset.param });
    this.isClear = false;
  }
})
```

注意：

- canvas组件是由客户端创建的原生组件，它的层级是最高的。
- 请勿在scroll-view中使用canvas组件。
- CSS 动画对canvas组件无效。

7.8 客服会话按钮

contact-button 即客服会话按钮，用于在页面上显示一个客服会话按钮，用户点击该按钮后会进入客服会话，在微信小程序内联系客服。

contact-button 所包含的属性如表 7-38 所示。

表 7-38

属性名	类型	默认值	说明
size	Number	18	会话按钮大小，有效值：18~27，单位：px
type	String	default-dark	会话按钮的样式类型，有效值：default-dark, default-light
session-from	String	navigate	用户从该按钮进入会话时，开发者将收到带上本参数的事件推送。本参数可用于区分用户进入客服会话的来源

示例代码：

```
<contact-button
  type="default-light"
  size="20"
  session-from="weapp"
>
</contact-button>
```

7.9 本章小结

本章主要介绍了七大类组件的功能样式和使用方法。这些组件是开发小程序的基础，所有的 UI 界面搭建都是靠各个组件来完成的。所以学习好本章内容，是成为小程序开发者必不可少的步骤。当然，一个成功的小程序，只有漂亮的 UI 界面是不够的，还要有相应的逻辑处理功能，这就需要掌握丰富的微信小程序 API。API 的相关知识在第 8 章中会详细阐述。

8 API 接口

微信小程序为开发者提供了丰富的微信原生 API，让开发者可以很方便地使用这些 API 实现各种复杂的功能，如获取用户信息、播放多媒体、本地存储、支付功能等。微信小程序目前提供的 API 可分为八大类：网络 API、多媒体 API、文件 API、数据存储 API、位置 API、设备 API、界面 API 以及开放接口。这些 API 可以实现各种各样复杂的原生功能。

说明：

- 以 wx.on 开头的 API 是监听某个事件发生的 API 接口，接受一个 CALLBACK 函数作为参数。当该事件触发时，会调用 CALLBACK 函数。
- 如果未特殊约定，则其他 API 接口都接受一个 OBJECT 作为参数。
- 在 OBJECT 中可以指定`success`, `fail`, `complete`来接收接口调用结果，如表 8-1 所示。

表 8-1

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）

8.1 网络相关

本节主要阐述微信小程序网络相关的 API。网络 API 主要用于本地和服务器交互。每个微信小程序需要事先设置一个通信域名，微信小程序可以跟指定的域名与进行网络通信，包括普通 HTTPS 请求（wx.request）、WebSocket 通信（wx.connectSocket）、上传文件（wx.uploadFile）和下载文件（wx.downloadFile)。

注意：网络请求的 referer 是不可以设置的，格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 AppId，{version} 为小程序的版本号，版本号为 0 表示为开发版。

8.1.1 发送请求

wx.request

wx.request(OBJECT) 发起的是 HTTPS 请求。

OBJECT 参数说明如表 8-2 所示。

表 8-2

参数名	类型	必填	说明
url	String	是	需要访问的服务器接口地址
data	Object、String	否	请求的参数
header	Object	否	设置请求的 header，header 中不能设置 Referer
method	String	否	默认为 GET，有效值：OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT
dataType	String	否	默认为 JSON。如果设置了 dataType 为 json，则会尝试对响应的数据做一次 JSON.parse
success	Function	否	收到开发者服务成功返回的回调函数，res = {data: ' 开发者服务器返回的内容 '}
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）

data 数据说明

最终发送给服务器的数据是 String 类型，如果传入的 data 不是 String 类型，则会被转换成 String 。转换规则如下：

- 对于 header['content-type'] 为 'application/json' 的数据，会对数据进行 JSON 序列化。
- 对于 header['content-type'] 为 'application/x-www-form-urlencoded' 的数据，会将数据转换成 query string（encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...）。

示例代码：

```
wx.request({
  url: 'https://xxxx.xxxx.com/request', //仅为示例，并非真实的接口地址
  data: {
     x: '' ,
     y: ''
  },
  header: {
      'content-type': 'application/json'
  },
  success: function(res) {
    console.log(res.data)
  }
})
```

注意：

- content-type 默认为 'application/json'。
- 在开发者工具 0.10.102800 版本中，header 的 content-type 设置异常。
- 客户端的 HTTPS TLS 版本为 1.2，但 Android 的部分机型还未支持 TLS 1.2，所以请确保 HTTPS 服务器的 TLS 版本支持 1.2 及以下版本。
- 要注意 method 的 value 必须为大写（例如：GET）。
- url 中不能有端口。
- request 的默认超时时间和最大超时时间都是 60s。
- request 的最大并发数是 5。
- 网络请求的 referer 是不可以设置的，格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 AppId，{version} 为微信小程序的版本号，版本号为 0 表示为开发版。

8.1.2 上传和下载

在微信小程序开发过程中，经常会用到文件的上传和下载功能。这就会用到网络交互上传和下载 API。

上传

wx.uploadFile

wx.uploadFile(OBJECT) 为上传 API，将本地资源上传到开发者服务器。如页面通过 wx.chooseImage 等接口获取到一个本地资源的临时文件路径后，可以通过此接口将本地资源上传到指定服务器。客户端发起一个 HTTPS POST 请求，其中 content-type 为 multipart/form-data。OBJECT 参数说明如表 8-3 所示。

表 8-3

参数名	类型	必填	说明
url	String	是	开发者服务器 URL
filePath	String	是	要上传文件资源的路径
name	String	是	文件对应的 key，开发者在服务器端通过这个 key 可以获取到文件二进制内容
header	Object	否	HTTP 请求 Header，header 中不能设置 Referer
formData	Object	否	HTTP 请求中其他额外的 form data
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）

success 返回参数说明如表 8-4 所示。

表 8-4

参数名	类型	说明
data	String	开发者服务器返回的数据
statusCode	Number	HTTP 状态码

示例代码：

```
wx.chooseImage({
  success: function(res) {
    var tempFilePaths = res.tempFilePaths
    wx.uploadFile({
      url: 'https://xxxx.xxxx.com/upload', // 上传路径url，仅为示例，非真实的接
          口地址
      filePath: tempFilePaths[0],
      name: 'file',
      formData:{
        'user': 'test'
      },
      success: function(res){
        var data = res.data
        //do something
      }
    })
  }
})
```

注意：

- 最大并发限制是 10 个。
- 默认超时时间和最大超时时间都是 60s。

下载

wx.downloadFile

wx.downloadFile(OBJECT) 为下载 API，下载文件资源到本地。客户端直接发起一个 HTTP GET 请求，返回文件的本地临时路径。OBJECT 参数说明如表 8-5 所示。

表 8-5

参数名	类型	必填	说明
url	String	是	下载资源的 URL
header	Object	否	HTTP 请求 Header
success	Function	否	下载成功后以 tempFilePath 的形式传给页面，res = {tempFilePath: ' 文件的临时路径'}
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）

示例代码：

```
wx.downloadFile({
  url: 'http://xxxx.xxxx.com/download/123', //下载资源URL，仅为示例，并非真实的
      资源
  success: function(res) {
    wx.playVoice({
      filePath: res.tempFilePath
    })
  }
})
```

注意：

- 最大并发限制是 10 个。
- 默认超时时间和最大超时时间都是 60s。
- 文件的临时路径在微信小程序本次启动期间可以正常使用，如果需要持久保存，则需要在主动调用 wx.saveFile，在微信小程序下次启动时才能访问得到。
- 网络请求的 referer 是不可以设置的，格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html，其中 {appid} 为微信小程序的 appid，{version} 为微信小程序的版本号，版本号为 0 表示为开版。
- 6.5.3 以及之前版本的 iOS 微信客户端 header 设置无效。

8.1.3 WebSocket

wx.connectSocket

wx.connectSocket(OBJECT) 即创建一个 WebSocket 连接；一个微信小程序同时只能有一个 WebSocket 连接，如果当前已存在一个 WebSocket 连接，则会自动关闭该连接，并重新创建一个 WebSocket 连接。OBJECT 参数说明如表 8-6 所示。

表 8-6

参数名	类型	必填	说明
url	String	是	开发者服务器接口地址，必须是 WSS 协议，且域名必须是后台配置的合法域名
data	Object	否	请求的数据
header	Object	否	HTTP Header，Header 中不能设置 Referer

续表

参数名	类型	必填	说明
method	String	否	默认是 GET，有效值为：OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT
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）

示例代码：

```
wx.connectSocket({
  url: 'test.php',
  data:{
    x: '',
    y: ''
  },
  header:{
    'content-type': 'application/json'
  },
  method:"GET"
})
```

wx.onSocketOpen

wx.onSocketOpen(CALLBACK) 即监听 WebSocket 连接打开事件。

示例代码：

```
wx.connectSocket({
  url: 'test.php'
})
wx.onSocketOpen(function(res) {
  console.log('WebSocket连接已打开！')
})
```

wx.onSocketError

wx.onSocketError(CALLBACK) 即监听 WebSocket 错误。

示例代码：

```
wx.connectSocket({
  url: 'test.php'
})
wx.onSocketOpen(function(res){
  console.log('WebSocket连接已打开！')
})
wx.onSocketError(function(res){
  console.log('WebSocket连接打开失败，请检查！')
})
```

wx.sendSocketMessage

wx.sendSocketMessage(OBJECT)即通过WebSocket连接发送数据，需要先wx.connectSocket，并在wx.onSocketOpen回调之后才能发送。

OBJECT参数说明如表8-7所示。

表8-7

参数名	类型	必填	说明
data	String/ArrayBuffer	是	需要发送的内容
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）

示例代码：

```
var socketOpen = false
var socketMsgQueue = []
wx.connectSocket({
  url: 'test.php'
})

wx.onSocketOpen(function(res) {
  socketOpen = true
  for (var i = 0; i < socketMsgQueue.length; i++){
     sendSocketMessage(socketMsgQueue[i])
  }
  socketMsgQueue = []
```

```
})

function sendSocketMessage(msg) {
  if (socketOpen) {
    wx.sendSocketMessage({
      data:msg
    })
  } else {
     socketMsgQueue.push(msg)
  }
}
```

wx.onSocketMessage

wx.onSocketMessage(CALLBACK) 即监听 WebSocket 接受到服务器的消息事件。CALLBACK 返回参数说明如表 8-8 所示。

表 8-8

参数名	类型	说明
data	String/ArrayBuffer	服务器返回的消息

示例代码：

```
wx.connectSocket({
  url: 'test.php'
})

wx.onSocketMessage(function(res) {
  console.log('收到服务器内容: ' + res.data)
})
```

wx.closeSocket

wx.closeSocket() 即关闭 WebSocket 连接。

wx.onSocketClose

wx.onSocketClose(CALLBACK) 即监听 WebSocket 关闭。

示例代码：

```
wx.connectSocket({
  url: 'test.php'
})

//注意这里有时序问题，
//如果 wx.connectSocket 还没回调 wx.onSocketOpen，而先调用 wx.closeSocket，那么
   就做不到关闭 WebSocket 的目的。
//必须在 WebSocket 打开期间调用 wx.closeSocket 才能关闭。
wx.onSocketOpen(function() {
  wx.closeSocket()
})

wx.onSocketClose(function(res) {
  console.log('WebSocket 已关闭！')
})
```

8.2 多媒体

在 7.4 节中已经阐述了多媒体组件的使用，如果只是单纯地使用组件，则往往达不到一些复杂项目所要求的功能效果。这时候就需要结合多媒体 API 来进行复杂的逻辑处理，来完成更加灵活复杂的操作，本节会详细阐述多媒体 API 的功能及使用方法。

8.2.1 图片

在项目开发过程中，经常需要实现上传图片的功能。本节会阐述在微信小程序开发中如何选择并上传图片。

wx.chooseImage

wx.chooseImage(OBJECT) 即选择图片。选择图片有两种方式：从本地相册选择图片或使用相机拍照。OBJECT 参数说明如表 8-9 所示。

表 8-9

参数名	类型	必填	说明
count	Number	否	最多可以选择的图片张数，默认为 9
sizeType	StringArray	否	original 为原图，compressed 为压缩图，默认二者都有

续表

参数名	类型	必填	说明
sourceType	StringArray	否	album 为从相册选图，camera 为使用相机，默认二者都有
success	Function	是	成功则返回图片的本地文件路径列表 tempFilePaths
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）

wx.previewImage

wx.previewImage(OBJECT) 即预览图片。OBJECT 参数说明如表 8-10 所示。

表 8-10

参数名	类型	必填	说明
current	String	否	当前显示图片的链接，不填则默认为 urls 的第一张
urls	StringArray	是	需要预览的图片链接列表
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）

wx.getImageInfo

wx.getImageInfo(OBJECT) 即获取图片信息。OBJECT 参数说明如表 8-11 所示。

表 8-11

参数名	类型	必填	说明
src	String	是	图片的路径，可以是相对路径，临时文件路径，存储文件路径，网络图片路径
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）

success 返回参数说明如表 8-12 所示。

表 8-12

参数名	类型	说明
width	Number	图片宽度，单位：px
height	Number	图片高度，单位：px
path	String	返回图片的本地路径

示例代码：

在 WXML 中：

```
<button style="margin:30rpx;" bindtap="chooseimage">获取图片</button>
<image src="{{tempFilePaths }}" mode="aspecFill" style="width: 100%; height: 700rpx
    "/>
```

在 JS 中：

```
Page({
  data: {
    tempFilePaths: ''
  },
  chooseimage: function () {
    var _this = this;
    wx.chooseImage({
      count: 1, // 默认9
      sizeType: ['original', 'compressed'], // 可以指定是原图还是压缩图，默认二
          者都有
      sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机，默认二者
          都有
      success: function (res) {
        // 返回选定照片的本地文件路径列表，tempFilePath可以作为img标签的src属
            性显示图片
        _this.setData({
          tempFilePaths: res.tempFilePaths
        })
      }
    })
    wx.getImageInfo({
      src: 'images/a.jpg',
      success: function (res) {
```

```
        console.log(res.width)
        console.log(res.height)
      }
    })
  }
})
```

运行效果如图 8-1 所示。

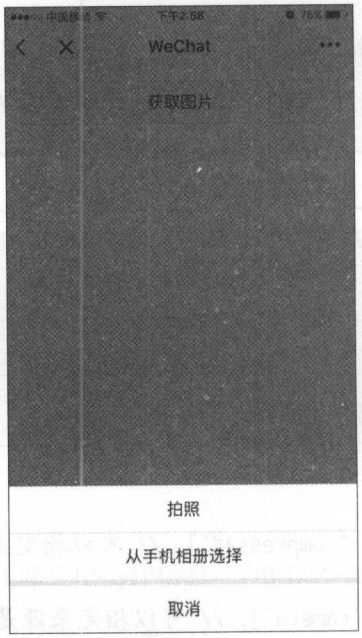

图 8-1　图片 API 示例运行效果图

注意：

文件的临时路径在微信小程序本次启动期间可以正常使用，如果需要持久保存，则需要在主动调用 wx.saveFile，在微信小程序下次启动时才能访问得到。

8.2.2　录音

语音录制功能也是开发者经常会使用的功能，本节会阐述录音功能相关的 API。

wx.startRecord

wx.startRecord(OBJECT) 即开始录音。当主动调用 wx.stopRecord 或者录音超过 1 分钟时，自动结束录音，返回录音文件的临时文件路径。当用户离开小程序时，则此接口无法调用。OBJECT 参数说明如表 8-13 所示。

表 8-13

参数名	类型	必填	说明
success	Function	否	录音成功后调用，返回录音文件的临时文件路径，res = {tempFilePath: ' 录音文件的临时路径'}
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）

wx.stopRecord

wx.stopRecord() 即主动调用停止录音。

示例代码：

在 WXML 中：

```
<scroll-view>
  <view wx:if="{{voices}}" class="common-list" style="margin-bottom:120rpx;">
    <block wx:for="{{voices}}">
      <view class="board">
        <view class="cell">
          <view class="cell-bd" data-key="{{item.filePath}}" bindtap="gotoPlay">
            <view class="date">存储路径:{{item.filePath}}</view>
            <view class="date">存储时间:{{item.createTime}}</view>
            <view class="date">音频大小:{{item.size}}KB</view>
          </view>
        </view>
      </view>
    </block>
  </view>
</scroll-view>
<view wx:if="{{isSpeaking}}" class="speak-style">
  <image class="sound-style" src="../../images/record_icon_1.png"></image>
  <image wx:if="{{j==2}}" class="sound-style" src="../../images/record_icon_2.png">
```

```
      </image>
  <image wx:if="{{j==3}}" class="sound-style" src="../../images/record_icon_3.png">
      </image>
  <image wx:if="{{j==4}}" class="sound-style" src="../../images/record_icon_4.png">
      </image>
  <image wx:if="{{j==5}}" class="sound-style" src="../../images/record_icon_5.png">
      </image>
</view>
<view class="record-style">
  <button class="btn-style" bindtouchstart="touchdown" bindtouchend="touchup">按住
      录音</button>
</view>
```

在 JS 中：

```
//获取应用实例
var app = getApp()
Page({
  data: {
    j: 1,//帧动画初始图片
    isSpeaking: false,//是否正在说话
    voices: [],//音频数组
  },
  //手指按下
  touchdown: function () {
    console.log("手指按下了...")
    console.log("new date : " + new Date)
    var _this = this;
    speaking.call(this);
    this.setData({
      isSpeaking: true
    })
    //开始录音
    wx.startRecord({
      success: function (res) {
        //临时路径，下次进入微信小程序时无法正常使用
        var tempFilePath = res.tempFilePath
        console.log("tempFilePath: " + tempFilePath)
        //持久保存
        wx.saveFile({
```

```
        tempFilePath: tempFilePath,
        success: function (res) {
          //持久路径
          //本地文件存储的大小限制为100MB
          var savedFilePath = res.savedFilePath
          console.log("savedFilePath: " + savedFilePath)
        }
      })
      wx.showToast({
        title: '恭喜!录音成功',
        icon: 'success',
        duration: 1000
      })
      //获取录音音频列表
      wx.getSavedFileList({
        success: function (res) {
          var voices = [];
          for (var i = 0; i < res.fileList.length; i++) {
            //格式化时间
            var createTime = new Date(res.fileList[i].createTime)
            //将音频大小B转为KB
            var size = (res.fileList[i].size / 1024).toFixed(2);
            var voice = { filePath: res.fileList[i].filePath, createTime:
                createTime, size: size };
            console.log("文件路径: " + res.fileList[i].filePath)
            console.log("文件时间: " + createTime)
            console.log("文件大小: " + size)
            voices = voices.concat(voice);
          }
          _this.setData({
            voices: voices
          })
        }
      })
    },
    fail: function (res) {
      //录音失败
      wx.showModal({
        title: '提示',
```

```
        content: '录音的姿势不对!',
        showCancel: false,
        success: function (res) {
          if (res.confirm) {
            console.log('用户点击确定')
            return
          }
        }
      })
    }
  })
},
//手指抬起
touchup: function () {
  console.log("手指抬起了...")
  this.setData({
    isSpeaking: false,
  })
  clearInterval(this.timer)
  wx.stopRecord()
},
//点击播放录音
gotoPlay: function (e) {
  var filePath = e.currentTarget.dataset.key;
  //点击开始播放
  wx.showToast({
    title: '开始播放',
    icon: 'success',
    duration: 1000
  })
  wx.playVoice({
    filePath: filePath,
    success: function () {
      wx.showToast({
        title: '播放结束',
        icon: 'success',
        duration: 1000
      })
    }
```

```
    })
  }
})

//麦克风帧动画
function speaking() {
  var _this = this;
  //话筒帧动画
  var i = 1;
  this.timer = setInterval(function () {
    i++;
    i = i % 5;
    _this.setData({
      j: i
    })
  }, 200);
}
```

运行效果如图 8-2 所示。

图 8-2 录音 API 示例运行效果图

注意：

- 文件的临时路径，在微信小程序本次启动期间可以正常使用，如果需要持久保存，则需要在主动调用 wx.saveFile，在微信小程序下次启动时才能访问得到。
- wx.startRecord 接口需要用户授权，请兼容用户拒绝授权的场景。

8.2.3 音频

本节会阐述音频播放控制功能，例如要播放自己录制的语音，就可以使用本节介绍的 API 来实现。

wx.playVoice

wx.playVoice(OBJECT) 即开始播放语音。同时只允许一个语音文件正在播放，如果前一个语音文件还没播放完，则将中断前一个语音播放。OBJECT 参数说明如表 8-14 所示。

表 8-14

参数名	类型	必填	说明
filePath	String	是	需要播放的语音文件的文件路径
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）

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath: tempFilePath,
      complete: function(){
      }
    })
  }
})
```

wx.pauseVoice

wx.pauseVoice() 即暂停正在播放的语音。再次调用 wx.playVoice 播放同一个文件时，会从暂停处开始播放。如果想从头开始播放，则需要先调用 wx.stopVoice。

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
      wx.playVoice({
      filePath: tempFilePath
    })

    setTimeout(function() {
       //暂停播放
      wx.pauseVoice()
    }, 5000)
  }
})
```

wx.stopVoice

wx.stopVoice() 即结束播放语音。

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath:tempFilePath
    })

    setTimeout(function(){
      wx.stopVoice()
    }, 5000)
  }
})
```

8.2.4 背景音乐

本节主要阐述微信小程序中对背景音乐的控制。例如，一些游戏场景须要播放背景音乐，都需要通过本节介绍的 API 来实现。

wx.getBackgroundAudioPlayerState

wx.getBackgroundAudioPlayerState(OBJECT) 即获取后台音乐播放状态。OBJECT 参数说明如表 8-15 所示。

表 8-15

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）

success 返回参数说明如表 8-16 所示。

表 8-16

参数名	说明
duration	选定音频的长度（单位：s），只有在当前有音乐播放时返回
currentPosition	选定音频的播放位置（单位：s），只有在当前有音乐播放时返回
status	播放状态（2：没有音乐在播放，1：播放中，0：暂停中）
downloadPercent	音频的下载进度（整数，80 代表 80%），只有在当前有音乐播放时返回
dataUrl	歌曲数据链接，只有在当前有音乐播放时返回

wx.playBackgroundAudio

wx.playBackgroundAudio(OBJECT) 即使用后台播放器播放音，对微信客户端来说，只能同时有一个后台音乐在播放。当用户离开微信小程序后，则音乐将暂停播放；当用户点击“显示在聊天顶部”时，音乐不会暂停播放；当用户在其他微信小程序占用了音乐播放器，原有微信小程序内的音乐将停止播放。OBJECT 参数说明如表 8-17 所示。

表 8-17

参数名	类型	必填	说明
dataUrl	String	是	音乐链接
title	String	否	音乐标题
coverImgUrl	String	否	封面 URL
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）

wx.pauseBackgroundAudio

wx.pauseBackgroundAudio() 即暂停播放音乐。

wx.seekBackgroundAudio

wx.seekBackgroundAudio(OBJECT) 即控制音乐播放进度。OBJECT 参数说明如表 8-18 所示。

表 8-18

参数名	类型	必填	说明
position	Number	是	音乐位置，单位：秒
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）

wx.stopBackgroundAudio

wx.stopBackgroundAudio() 即停止播放音乐。

wx.onBackgroundAudioPlay

wx.onBackgroundAudioPlay(CALLBACK) 即监听音乐播放。

wx.onBackgroundAudioPause

wx.onBackgroundAudioPause(CALLBACK) 即监听音乐暂停。

wx.onBackgroundAudioStop

wx.onBackgroundAudioStop(CALLBACK) 即监听音乐停止。

示例代码：

在 WXML 中：

```
<view class="audio-view">
  <button class="button" type="primary" bindtap="listenerButtonPlay">播放</button>
  <button class="button" type="primary" bindtap="listenerButtonPause">暂停</button>
  <button class="button" type="primary" bindtap="listenerButtonSeek">设置播放进度<
      /button>
  <button class="button" type="primary" bindtap="listenerButtonStop">停止播放</
      button>
  <button class="button" type="primary" bindtap="listenerButtonGetPlayState">获取播
      放状态</button>
</view>
```

在 JS 中：

```
Page({
  data: {},
  listenerButtonPlay: function () {
    wx.playBackgroundAudio({
      //播放地址
      dataUrl: 'https://xxxx.xxxxx.com/xxx.mp3',
      //title 音乐名字
      title: '名字',
      //图片地址
      coverImgUrl: 'https://xxxx.xxxxx.com/xxxxxx'
    })
  },
  /**
   * 播放状态
   */
  listenerButtonGetPlayState: function () {
    wx.getBackgroundAudioPlayerState({
```

```
      success: function (res) {
        console.log(res)
        //duration 总时长
        //currentPosition 当前播放位置
        //status 播放状态
        //downloadPercent 下载状况 100 即为100%
        //dataUrl 当前播放音乐地址
      }
    })
  },
  /**
   * 监听button暂停按钮
   */
  listenerButtonPause: function () {
    wx.pauseBackgroundAudio();
  },
  /**
   * 设置进度
   */
  listenerButtonSeek: function () {
    wx.seekBackgroundAudio({
      position: 30
    })
  },
  /**
   *停止播放
   */
  listenerButtonStop: function () {
    wx.stopBackgroundAudio()
  }
})
```

运行效果如图 8-3 所示。

注意：

在 iOS 6.3.30 中，wx.seekBackgroundAudio 会有短暂延迟。

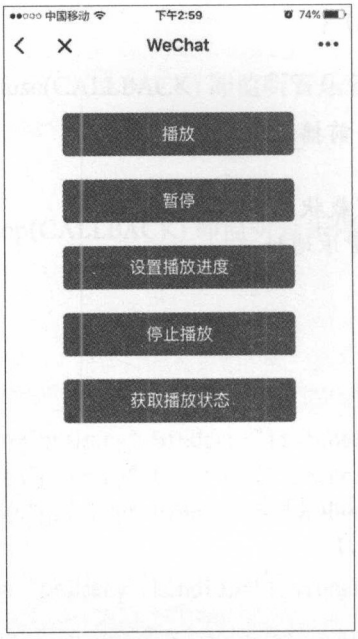

图 8-3 背景音乐 API 示例运行效果图

8.2.5 音频组件控制

wx.createAudioContext

wx.createAudioContext(audioId) 即创建并返回 audio 上下文 audioContext 对象。audioContext 通过 audioId 与一个<audio/>组件绑定，通过它可以操作对应的<audio/>组件。

audioContext 对象的方法如表 8-19 所示。

表 8-19

方法	参数	说明
setSrc	src	音频的地址
play	无	播放
pause	无	暂停
seek	position	跳转到指定位置，单位：s

示例代码：

在 WXML 中：

```
<audio  src="{{src}}" id="myAudio" ></audio>
<button type="primary" bindtap="audioPlay">播放</button>
<button type="primary" bindtap="audioPause">暂停</button>
<button type="primary" bindtap="audio14">设置当前播放时间为14秒</button>
<button type="primary" bindtap="audioStart">回到开头</button>
```

在 JS 中：

```
Page({
  onReady: function (e) {
    // 使用 wx.createAudioContext 获取 audio 上下文 context
    this.audioCtx = wx.createAudioContext('myAudio')
    this.audioCtx.setSrc('https://xxxx.xxxx.com/xxxxx')
    this.audioCtx.play()
  },
  data: {
    src: ''
  },
  audioPlay: function () {
    this.audioCtx.play()
  },
  audioPause: function () {
    this.audioCtx.pause()
  },
  audio14: function () {
    this.audioCtx.seek(14)
  },
  audioStart: function () {
    this.audioCtx.seek(0)
  }
})
```

8.2.6 视频

视频功能在平时的项目开发中也是时常会用到的，本节会阐述视频播放、停止等视频控制 API。

wx.chooseVideo

wx.chooseVideo(OBJECT) 即拍摄视频或从手机相册中选视频，返回视频的临时文件路径。OBJECT 参数说明如表 8-20 所示。

表 8-20

参数名	类型	必填	说明
sourceType	StringArray	否	album：从相册选视频，camera：使用相机拍摄，默认为：['album', 'camera']
maxDuration	Number	否	拍摄视频最长拍摄时间，单位：秒。最长支持 60 秒
camera	String	否	默认调起的为前置还是后置摄像头。front：前置，back：后置，默认为 back
success	Function	否	接口调用成功，返回视频文件的临时文件路径，详见返回参数说明
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）

返回参数说明如表 8-21 所示。

表 8-21

参数名	说明
tempFilePath	选定视频的临时文件路径
duration	选定视频的时间长度
size	选定视频的数据量大小
height	返回选定视频的长
width	返回选定视频的宽

示例代码：

在 WXML 中：

```
<View>1.播放网络视频</View>
<view>
  <video style="width: 100%;height=400px;margin:1px;" src="https://xxxx.xxxx.com/
      xxxx"
  binderror="videoErrorCallback"></video>
</view>
<View>2.播放本地视频</View>
<view style="display: flex;flex-direction: column;">
  <video style="width: 100%;height=400px;margin:1px;" src="{{src}}"></video>
```

```
  <view class="btn-area">
    <button bindtap="bindButtonTap">打开本地视频</button>
  </view>
</view>
```

在JS中：

```
Page({
  data: {
    src: ''
  },
  // 打开本地视频
  bindButtonTap: function () {
    var that = this
    //拍摄视频或从手机相册中选视频
    wx.chooseVideo({
      //album 从相册选视频，camera 使用相机拍摄，默认为：['album', 'camera']
      sourceType: ['album', 'camera'],
      //拍摄视频最长拍摄时间，单位：秒。最长支持60秒
      maxDuration: 60,
      //前置或者后置摄像头，默认为前后都有，即：['front', 'back']
      camera: ['front', 'back'],
      //接口调用成功，返回视频文件的临时文件路径，详见返回参数说明
      success: function (res) {
        console.log(res.tempFilePaths[0])
        that.setData({
          src: res.tempFilePaths[0]
        })
      }
    })
  },
  /**
   * 当发生错误时触发error事件，event.detail = {errMsg: 'something wrong'}
   */
  videoErrorCallback: function (e) {
    console.log('视频错误信息:')
    console.log(e.detail.errMsg)
  }
})
```

运行效果如图8-4所示。

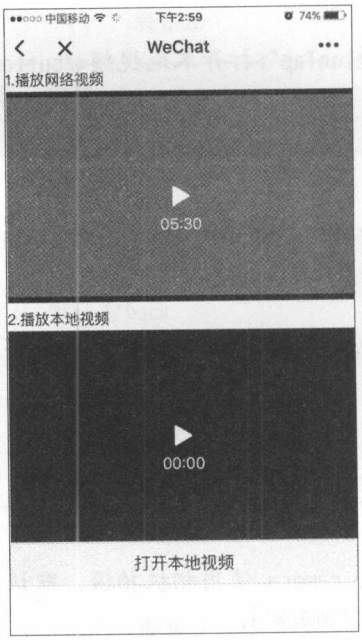

图 8-4 视频 API 示例运行效果图

注意：

文件的临时路径，在微信小程序本次启动期间可以正常使用，如果需要持久保存，则需要在主动调用 wx.saveFile，在微信小程序下次启动时才能访问得到。

8.2.7 视频组件控制

wx.createVideoContext

wx.createVideoContext(videoId) 用于创建并返回 video 上下文 videoContext 对象。videoContext 通过 videoId 和一个 video 组件绑定，通过它可以操作一个 video 组件。videoContext 对象的方法如表 8-22 所示。

表 8-22

方法	参数	说明
play	无	播放
pause	无	暂停
seek	position	跳转到指定位置，单位：s
sendDanmu	danmu	发送弹幕，danmu 包含两个属性 text, color

示例代码：

在 WXML 中：

```
<view class="section tc">
  <video id="myVideo" src="https://xxxx.xxxx.com/xxxx"   enable-danmu danmu-btn
      controls></video>
  <view class="btn-area">
    <input bindblur="bindInputBlur"/>
    <button bindtap="bindSendDanmu">发送弹幕</button>
  </view>
</view>
```

在 JS 中：

```
function getRandomColor () {
  let rgb = []
  for (let i = 0 ; i < 3; ++i){
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length == 1 ? '0' + color : color
    rgb.push(color)
  }
  return '#' + rgb.join('')
}

Page({
  onReady: function (res) {
    this.videoContext = wx.createVideoContext('myVideo')
  },
  inputValue: '',
  bindInputBlur: function(e) {
    this.inputValue = e.detail.value
  },
  bindSendDanmu: function () {
    this.videoContext.sendDanmu({
      text: this.inputValue,
      color: getRandomColor()
    })
  }
})
```

8.3 文件

使用 API 对小程序中出现的各类文件进行操作。

wx.saveFile

wx.saveFile(OBJECT) 即保存文件到本地。

OBJECT 参数说明如表 8-23 所示。

表 8-23

参数	类型	必填	说明
tempFilePath	String	是	需要保存的文件的临时路径
success	Function	否	返回文件的保存路径，res = {savedFilePath: ' 文件的保存路径 '}
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）

示例代码：

```
wx.saveFile({
  success: function(res) {
    var tempFilePath = res.tempFilePath    //获取文件保存的路径
    wx.saveFile({
      tempFilePath: tempFilePath,
      success: function(res) {
        var savedFilePath = res.savedFilePath
      }
    })
  }
})
```

本地文件存储的大小限制为 10MB。

wx.getSavedFileList

wx.getSavedFileList(OBJECT) 即获取本地已保存的文件列表。

OBJECT 参数说明如表 8-24 所示。

表 8-24

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，返回结果见 success 返回参数说明
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）

success 返回参数说明如表 8-25 所示。

表 8-25

参数	类型	说明
errMsg	String	接口调用结果
fileList	Object Array	文件列表

fileList 中的项目说明如表 8-26 所示。

表 8-26

键	类型	说明
filePath	String	文件的本地路径
createTime	Number	文件的保存时的时间戳，从 1970/01/01 08:00:00 到当前时间的秒数
size	Number	文件大小，单位：B

示例代码：

```
wx.getSavedFileList({
  success: function(res) {
    console.log(res.fileList)                    //打印全部列表
    console.log(res.fileList[0].filePath)                //打印第一项的文件路径
  }
})
```

wx.getSavedFileInfo

wx.getSavedFileInfo(OBJECT) 即获取本地文件的文件信息

OBJECT 参数说明如表 8-27 所示。

表 8-27

参数	类型	必填	说明
filePath	String	是	文件路径
success	Function	否	接口调用成功的回调函数，返回结果见 success 返回参数说明
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）

success 返回参数说明如表 8-28 所示。

表 8-28

参数	类型	说明
errMsg	String	接口调用结果
size	Number	文件大小，单位：B
createTime	Number	文件的保存是的时间戳，从 1970/01/01 08:00:00 到当前时间的秒数

示例代码：

```
wx.getSavedFileInfo({
  filePath: 'wxfile://somefile', //仅做示例用，非真正的文件路径
  success: function(res) {
    console.log(res.size)
    console.log(res.createTime)
  }
})
```

wx.removeSavedFile

wx.removeSavedFile(OBJECT) 即删除本地存储的文件。

OBJECT 参数说明如表 8-29 所示。

表 8-29

参数	类型	必填	说明
filePath	String	是	需要删除的文件路径
success	Function	否	接口调用成功的回调函数

续表

参数	类型	必填	说明
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）

示例代码：

```
wx.getSavedFileList({
  success: function(res) {
    if (res.fileList.length > 0){
      wx.removeSavedFile({
        filePath: res.fileList[0].filePath,        //此路径为文件列表第一项的路径
        complete: function(res) {
          console.log(res)
        }
      })
    }
  }
})
```

wx.openDocument

wx.openDocument(OBJECT) 即新开页面打开文档，支持格式：.doc、.xls、.ppt、.pdf、.docx、.xlsx、.pptx。

OBJECT 参数说明如表 8-30 所示。

表 8-30

参数	类型	必填	说明
filePath	String	是	文件路径，可通过 downFile 获得
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）

示例代码：

```
wx.downloadFile({
  url: 'http://example.com/somefile.pdf',     //演示地址
  success: function (res) {
    var filePath = res.tempFilePath       //获取下载文件地址
    wx.openDocument({
      filePath: filePath,
      success: function (res) {
        console.log('打开文档成功')
      }
    })
  }
})
```

8.4 数据缓存

每个微信小程序都可以有自己的本地缓存，可以通过 wx.setStorage（wx.setStorageSync）、wx.getStorage（wx.getStorageSync）、wx.clearStorage（wx.clearStorageSync）对本地缓存进行设置、获取和清理。本地缓存最大为 10MB。

localStorage 是永久的本地存储，但是为了防止用户在更换设备后丢失信息，所以对于重要信息还是建议存储在服务器端，通过请求获取。

数据的存储与读取主要通过“key”与“value”的形式实现，例如存储密码信息采用“password”-“123456”这样的形式，“password”为这组值的 key，“123456”为实际存储的值。由于 key 的值是唯一的，所以在需要读取数据时即可依靠这唯一的值准确获取存储的信息。

wx.setStorage

wx.setStorage(OBJECT) 即将数据存储在本地缓存中指定的 key 中，会覆盖原来该 key 对应的内容，这是一个异步接口。

OBJECT 参数说明如表 8-31 所示。

表 8-31

参数	类型	必填	说明
key	String	是	本地缓存中的指定的 key
data	Object/String	是	需要存储的内容

续表

参数	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）

示例代码：

```
wx.setStorage({
  key:"key",
  data:"value"
})
```

wx.setStorageSync

wx.setStorageSync(KEY,DATA) 即将 data 存储在本地缓存中指定的 key 中，会覆盖原来该 key 对应的内容，这是一个同步接口。

参数说明如表 8-32 所示。

表 8-32

参数	类型	必填	说明
key	String	是	本地缓存中的指定的 key
data	Object/String	是	需要存储的内容

示例代码：

```
try {
    wx.setStorageSync('key', 'value')
} catch (e) {
}
```

wx.getStorage(OBJECT)

wx.getStorage(OBJECT) 为从本地缓存中异步获取指定 key 对应的内容。

OBJECT 参数说明如表 8-33 所示。

表 8-33

参数	类型	必填	说明
key	String	是	本地缓存中的指定的 key
success	Function	是	接口调用的回调函数，res = {data: key 对应的内容}
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）

示例代码：

```
wx.getStorage({
  key: 'key',
  success: function(res) {
      console.log(res.data)
  }
})
```

wx.getStorageSync

wx.getStorageSync(KEY) 即从本地缓存中同步获取指定 key 对应的内容。

参数说明如表 8-34 所示。

表 8-34

参数	类型	必填	说明
key	String	是	本地缓存中的指定的 key

示例代码：

```
try {
  var value = wx.getStorageSync('key')
  if (value) {
      // Do something with return value
  }
} catch (e) {
  // Do something when catch error
}
```

wx.getStorageInfo(OBJECT)

wx.getStorageInfo(OBJECT) 即异步获取当前 storage 的相关信息

OBJECT 参数说明如表 8-35 所示。

表 8-35

参数	类型	必填	说明
success	Function	是	接口调用的回调函数，详见返回参数说明
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）

success 返回参数说明如表 8-36 所示。

表 8-36

参数	类型	说明
keys	String Array	当前 storage 中所有的 key
currentSize	Number	当前占用的空间大小，单位：KB
limitSize	Number	限制的空间大小，单位：KB

示例代码：

```
wx.getStorageInfo({
  success: function(res) {
    console.log(res.keys)
    console.log(res.currentSize)
    console.log(res.limitSize)
  }
})
```

wx.getStorageInfoSync

wx.getStorageInfoSync 即同步获取当前 storage 的相关信息。

示例代码：

```
try {
  var res = wx.getStorageInfoSync()
```

```
  console.log(res.keys)
  console.log(res.currentSize)
  console.log(res.limitSize)
} catch (e) {
  // Do something when catch error
}
```

wx.removeStorage

wx.removeStorage(OBJECT) 即从本地缓存中异步移除指定 key。

OBJECT 参数说明如表 8-37 所示。

表 8-37

参数	类型	必填	说明
key	String	是	本地缓存中的指定的 key
success	Function	是	接口调用的回调函数
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）

示例代码：

```
wx.removeStorage({
  key: 'key',
  success: function(res) {
    console.log(res.data)
  }
})
```

wx.removeStorageSync

wx.removeStorageSync(KEY) 即从本地缓存中同步移除指定 key。

参数说明如表 8-38 所示。

表 8-38

参数	类型	必填	说明
key	String	是	本地缓存中的指定的 key

示例代码：

```
try {
  wx.removeStorageSync('key')
} catch (e) {
  // Do something when catch error
}
```

wx.clearStorage

wx.clearStorage() 即清理本地数据缓存。

示例代码：

```
wx.clearStorage()
```

wx.clearStorageSync

wx.clearStorageSync() 即同步清理本地数据缓存。

示例代码：

```
try {
    wx.clearStorageSync()
} catch(e) {
  // Do something when catch error
}
```

同步与异步的区别在于，使用同步的请求必须要在请求完成后才能执行下一步请求，而异步请求不需要等待返回结果就可以执行下一个请求。使用异步请求效率更高，在多数情况下，我们会优先使用异步请求。只有在返回数据会影响请求者状态的情况下，才需要使用同步请求。例如在本地存储了数据“password”(key)为“123456”(value)，然后使用 setStorage 更改这个数值，接着使用 getStorage 获取密码，因为是异步操作，所以在发出更改请求后会立即发出获取请求，但数据的更改需要花费时间，这就导致了我们获取的密码可能还是更改前的“123456”。所以在这种情况下就需要使用同步操作，在确保上一个请求完成后再进行下一个。注意，在使用同步请求时需要使用“try-catch”语句包裹请求代码。

8.5 位置

在某些微信小程序中我们需要使用位置信息（例如寻找附近的餐饮时就需要先获取用户当前位置），本节会介绍位置相关 API，以及在使用地图组件时用到的一些 API。

8.5.1 获取与查看位置

wx.getLocation

wx.getLocation(OBJECT) 即获取当前的地理位置、速度。当用户离开微信小程序后，此接口无法调用；当用户点击“显示在聊天顶部”选项时，此接口可继续调用。

OBJECT 参数说明如表 8-39 所示。

表 8-39

参数	类型	必填	说明
type	String	否	默认为 wgs84 返回 GPS 坐标，gcj02 返回可用于 wx.openLocation 的坐标
success	Function	是	接口调用成功的回调函数，返回内容详见返回参数说明
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）

success 返回参数说明如表 8-40 所示。

表 8-40

参数	说明
latitude	纬度，浮点数，范围为-90~90，负数表示南纬
longitude	经度，浮点数，范围为-180~180，负数表示西经
speed	速度，浮点数，单位 GPSm/s
accuracy	位置的精确度
name	位置名称
address	详细地址
latitude	纬度，浮点数，范围为-90~90，负数表示南纬
longitude	经度，浮点数，范围为-180~180，负数表示西经

示例代码：

```
wx.getLocation({
  type: 'wgs84',
  success: function(res) {
    console.log(res.latitude)
    console.log(res.longitude)
    console.log(res.speed)
    console.log(res.accuracy)
  }
})
```

模拟器运行结果为：

```
39.084158
117.200983
undefined        //这里undefined的原因是模拟器没有获取速度的传感器
undefined
```

wx.chooseLocation

wx.chooseLocation(OBJECT) 即打开地图选择位置。

OBJECT 参数说明如表 8-41 所示。

表 8-41

参数	类型	必填	说明
success	Function	是	接口调用成功的回调函数，返回内容详见返回参数说明
cancel	Function	否	用户取消时调用
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）

success 返回参数说明如表 8-42 所示。

表 8-42

参数	说明
name	位置名称
address	详细地址
latitude	纬度，浮点数，范围为-90~90，负数表示南纬

续表

参数	说明
longitude	经度，浮点数，范围为-180~180，负数表示西经

示例代码：

```
wx.chooseLocation({
      success: function(res){
        console.log(res.latitude)
        console.log(res.longitude)
      },
      fail: function() {
        // fail
      },
      complete: function() {
        // complete
      }
    })
```

调用此 API 会打开地图，在选择地点后点击右上角的“发送”按钮，即可获取所选地点的经纬度、地址、名称的信息。

wx.openLocation

wx.openLocation(OBJECT) 即使用微信内置地图查看位置。

OBEJCT 参数说明如表 8-43 所示。

表 8-43

参数	类型	必填	说明
latitude	Float	是	纬度，范围为-90~90，负数表示南纬
longitude	Float	是	经度，范围为-180~180，负数表示西经
scale	INT	否	缩放比例，范围为 5~18，默认为 18
name	String	否	位置名
address	String	否	地址的详细说明
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

续表

参数	类型	必填	说明
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.getLocation({
  type: 'gcj02', //返回可以用于wx.openLocation的经纬度
  success: function(res) {
    var latitude = res.latitude
    var longitude = res.longitude
    wx.openLocation({
      latitude: latitude,
      longitude: longitude,
      scale: 28
    })
  }
})
```

wx.getLocation 与 wx.chooseLocation 接口需要用户授权（询问用户是否允许微信小程序获取当前位置），需要对用户拒绝授权的场景做出处理（拒绝后使用默认地点或弹出提示框要求用户开启授权后重试）。

8.5.2 地图组件控制

下面介绍配合地图组件使用的 API。

wx.createMapContext

wx.createMapContext(mapId) 即创建并返回 map 上下文 mapContext 对象。mapContext 通过 mapId 跟一个地图组件绑定，通过它可以操作对应的地图组件。

mapContext 对象的方法列表如表 8-44 所示。

表 8-44

方法	参数	说明
getCenterLocation	OBJECT	获取当前地图中心的经纬度，返回的是 GCJ02 坐标系，可以用于 wx.openLocation

续表

方法	参数	说明
moveToLocation	无	将地图中心移动到当前定位点，需要配合 map 组件的 show-location 使用

getCenterLocation 的 OBJECT 参数列表如表 8-45 所示。

表 8-45

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，res = { longitude: " 经度", latitude: " 纬度"}
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）

示例代码：

```
<map id="myMap" show-location />

<button type="primary" bindtap="getCenterLocation">获取位置</button>
<button type="primary" bindtap="moveToLocation">移动位置</button>
```

```
onReady: function (e) {
    // 使用 wx.createMapContext 获取 map 上下文
    this.mapCtx = wx.createMapContext('myMap')
  },
  getCenterLocation: function () {
    this.mapCtx.getCenterLocation({
      success: function (res) {
        console.log(res.longitude)
        console.log(res.latitude)
      }
    })
  },
  moveToLocation: function () {
    this.mapCtx.moveToLocation()
  },
```

运行结果如图 8-5 所示。

图 8-5 地图组件控制 API 例图

8.6 设备

有些时候，我们需要获取手机的信息或使用手机的部分功能（例如打电话、扫描二维码），微信小程序中提供了调用这些功能的 API。

8.6.1 系统信息

wx.getSystemInfo

wx.getSystemInfo(OBJECT) 即获取系统信息。

OBJECT 参数说明如表 8-46 所示。

表 8-46

参数	类型	必填	说明
success	Function	是	接口调用成功的回调
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）

success 回调参数说明如表 8-47 所示。

表 8-47

属性	说明
model	手机型号
pixelRatio	设备像素比

续表

属性	说明
windowWidth	窗口宽度
windowHeight	窗口高度
language	微信设置的语言
version	微信版本号
system	操作系统版本
platform	客户端平台

示例代码：

```
wx.getSystemInfo({
  success: function(res) {
    console.log(res.model)
    console.log(res.pixelRatio)
    console.log(res.windowWidth)
    console.log(res.windowHeight)
    console.log(res.language)
    console.log(res.version)
    console.log(res.platform)
  }
})
```

wx.getSystemInfoSync

wx.getSystemInfoSync() 即获取系统信息同步接口。

示例代码：

```
try {
  var res = wx.getSystemInfoSync()
  console.log(res.model)
  console.log(res.pixelRatio)
  console.log(res.windowWidth)
  console.log(res.windowHeight)
  console.log(res.language)
  console.log(res.version)
  console.log(res.platform)
} catch (e) {
```

```
  //当获取信息发生错误时，在这个调用错误时需要执行的代码
}
```

在虚拟机上运行打印的结果为：

```
iPhone 6
3
414
696
zh_CN
6.3.9
devtools
```

8.6.2 网络状态

wx.getNetworkType

wx.getNetworkType(OBJECT) 即获取网络类型。

OBJECT 参数说明如表 8-48 所示。

表 8-48

参数	类型	必填	说明
success	Function	是	接口调用成功，返回网络类型 networkType
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）

示例代码：

```
wx.getNetworkType({
  success: function(res) {
    // 返回网络类型，有效值：
    // wifi/2g/3g/4g/unknown(Android下不常见的网络类型)/none(无网络)
    var networkType = res.networkType
    console.log(networkType)
  }
})
```

8.6.3 重力感应

wx.onAccelerometerChange

wx.onAccelerometerChange(CALLBACK) 即监听重力感应数据，频率为 5 次/秒。

CALLBACK 返回参数如表 8-49 所示。

表 8-49

参数	类型	说明
x	Number	X 轴
y	Number	Y 轴
z	Number	Z 轴

示例代码：

```
wx.onAccelerometerChange(function(res) {
  console.log(res.x)
  console.log(res.y)
  console.log(res.z)
})
```

8.6.4 罗盘

wx.onCompassChange

wx.onCompassChange(CALLBACK) 即监听罗盘数据，频率为 5 次/秒。

CALLBACK 返回参数如表 8-50 所示。

表 8-50

参数	类型	说明
direction	Number	面对的方向度数

示例代码：

```
wx.onCompassChange(function (res) {
  console.log(res.direction)
})
```

8.6.5 拨打电话

wx.makePhoneCall

wx.makePhoneCall(OBJECT) 即调用拨打电话的操作。OBJECT 参数说明如表 8-51 所示。

表 8-51

参数	类型	必填	说明
phoneNumber	String	是	需要拨打的电话号码
success	Function	否	接口调用成功的回调
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）

示例代码：

```
wx.makePhoneCall({
  phoneNumber: '1340000' //仅为示例，并非真实的电话号码
})
```

调用这行代码会弹出“是否拨打电话：1340000”的选择框，点击“确定”按钮后会拨打电话。

8.6.6 扫描二维码

wx.scanCode

wx.scanCode(OBJECT) 即调起客户端扫描二维码界面，扫描二维码成功后返回对应的结果。

Object 参数说明如表 8-52 所示。

表 8-52

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，返回内容详见返回参数说明
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）

success 返回参数说明如表 8-53 所示。

表 8-53

参数	说明
result	所扫描二维码的内容
scanType	所扫描二维码的类型
charSet	所扫描二维码的字符集
path	当所扫描的二维码为当前微信小程序的合法二维码时，会返回此字段，内容为二维码携带的 path

示例代码：

```
wx.scanCode({
  success: (res) => {   //相当于function(res)
    console.log(res)
  }
})
```

8.7 界面

在创建微信小程序界面时，不仅需要使用微信小程序的组件，还要通过调用其相关的 API 来实现组件的使用或完成组件之间的逻辑关系。

8.7.1 交互反馈

在第 6 章中介绍了用于用户交互的表单组件（例如按钮），在大多数情况下，用户都希望在点击这些组件后，系统可以反馈一些信息。微信小程序提供了负责交互反馈的 API 来帮助我们实现这一目标。

wx.showToast

wx.showToast(OBJECT) 即显示消息提示框。

OBJECT 参数说明如表 8-54 所示。

表 8-54

参数	类型	必填	说明
title	String	是	提示的内容
icon	String	否	图标，只支持"success"、"loading"
duration	Number	否	提示的延迟时间，单位：毫秒，默认值为 1500，最大值为 10000
mask	Boolean	否	是否显示透明蒙层，防止触摸穿透，默认值为 false
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）

示例代码：

```
<button type="default" bindtap="toastTap">点击弹出toast</button>
```

```
toastTap: function () {
    wx.showToast({
      title: "确定",
      // icon: "success",
      // duration: 2000,
      // mask: true,
    })
 },
```

运行结果如图 8-6 所示。

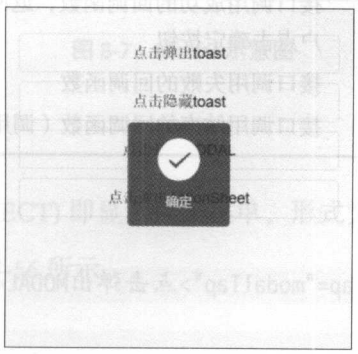

图 8-6 Toast 示意图

wx.hideToast

wx.hideToast() 即隐藏消息提示框。

示例代码：

```
<button type="default" bindtap="hideToast">点击隐藏toast</button>
```

```
hideToast: function () {
    wx.hideToast()
  }
```

wx.showModal

wx.showModal(OBJECT) 为显示模拟弹窗，即带交互按钮的提示框。

OBJECT 参数说明如表 8-55 所示。

表 8-55

参数	类型	必填	说明
title	String	是	提示的标题
content	String	是	提示的内容
showCancel	Boolean	否	是否显示取消按钮，默认为 true
cancelText	String	否	取消按钮的文字，默认为“取消”，最多 4 个字符
cancelColor	HexColor	否	取消按钮的文字颜色，默认为"#000000"
confirmText	String	否	确定按钮的文字，默认为“确定”，最多 4 个字符
confirmColor	HexColor	否	确定按钮的文字颜色，默认为"#3CC51F"
success	Function	否	接口调用成功的回调函数，返回 res.confirm 为 true 时，表示用户点击确定按钮
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）

示例代码：

```
<button type="default" bindtap="modalTap">点击弹出MODAL</button>
```

```
modalTap: function () {
    wx.showModal({
      //title: "标题",
```

```
    content: "弹窗内容，告知当前状态、信息和解决方法，描述文字尽量控制在三
        行内",
    //showCancel: false,
    confirmText: "确定",
    cancelText: "取消",
    //confirmColor: "#66ccff",
    //cancelColor: "ffcc66"
  })
},
```

运行结果如图 8-7 所示。

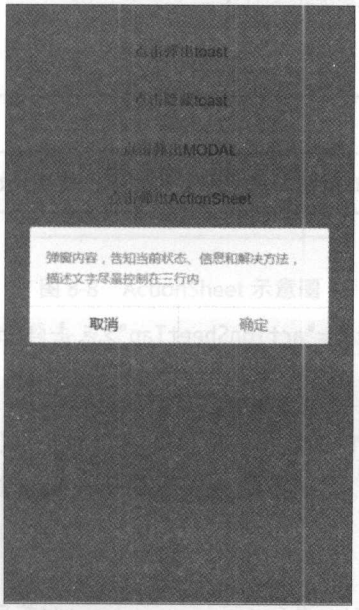

图 8-7 Modal 示意图

wx.showActionSheet

wx.showActionSheet(OBJECT) 即显示操作菜单，形式为底部弹出选择按钮。

OBJECT 参数说明如表 8-56 所示。

表 8-56

参数	类型	必填	说明
itemList	String Array	是	按钮的文字数组，数组长度最大为 6 个
itemColor	HexColor	否	按钮的文字颜色，默认为"#000000"
success	Function	否	接口调用成功的回调函数，详见返回参数说明
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）

success 返回参数说明如表 8-57 所示。

表 8-57

参数	类型	说明
tapIndex	Number	用户点击的按钮，从上到下的顺序，从 0 开始

示例代码：

```
<button type="default" bindtap="actionSheetTap">点击弹出ActionSheet</button>
```

```
actionSheetTap: function () {
    wx.showActionSheet({
      itemList: ['item1', 'item2', 'item3', 'item4'],
      //itemColor:"#223344",
      success: function (e) {
        var item=""
        if(e.tapIndex!=undefined){                    //获取点击的是第几项
          item="点击了第"+String(e.tapIndex+1)+"项"
        }else{                                        //如果点击了取消按钮或菜单以外
            区域，则返回“取消选择”
          item="取消选择"
        }
        wx.showToast({                    //弹出提示框
          title: item
        })
      }
    })
  },
```

运行结果如图 8-8 所示。

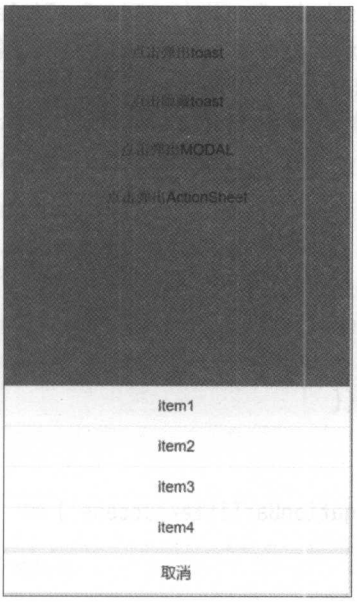

图 8-8 ActionSheet 示意图

8.7.2 设置导航条

前面已经介绍过通过 JSON 文件设置页面标题栏文字的方式，但有些时候我们需要动态地设置页面的标题，例如标题为网络请求返回的字段，可以通过微信小程序的 API 来实现。

wx.setNavigationBarTitle

wx.setNavigationBarTitle(OBJECT) 即动态设置当前页面的标题。

OBJECT 参数说明如表 8-58 所示。

表 8-58

参数	类型	必填	说明
title	String	是	页面标题
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）

示例代码：

```
<form class="page-body" bindsubmit="setNaivgationBarTitle">
    <input class="btn-area" type="text" placeholder="请输入页面标题并点击设置即
        可" name="title"></input>
    <view class="btn-area">
        <button type="primary" formType="submit">设置</button>
    </view>
</form>
```

```
setNaivgationBarTitle: function (e) {
    var title = e.detail.value.title
    console.log(title)
    wx.setNavigationBarTitle({
      title: title,
      success: function() {
        console.log('setNavigationBarTitle success')
      },
      fail: function(err) {
        console.log('setNavigationBarTitle fail, err is', err)
      }
    })
}
```

运行结果如图 8-9 所示。

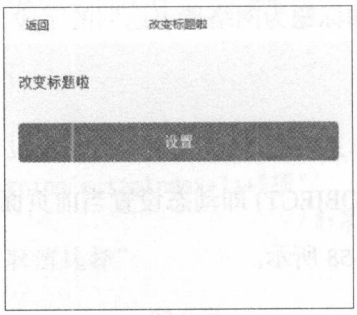

图 8-9 setNavigationBarTitle 示意图

我们还可以利用页面的导航条来表示页面信息的加载状态，需要调用的 API 为：

wx.showNavigationBarLoading：即在当前页面显示导航条加载动画。

wx.hideNavigationBarLoading：隐藏导航条加载动画。

示例代码：

```
<view class="btn-area">
    <button type="primary" bindtap="showNavigationBarLoading">显示加载动画</
        button>
    <button bindtap="hideNavigationBarLoading">隐藏加载动画</button>
  </view>
```

```
showNavigationBarLoading: function () {
    wx.showNavigationBarLoading()
  },
hideNavigationBarLoading: function () {
    wx.hideNavigationBarLoading()
}
```

效果如图 8-10 所示。

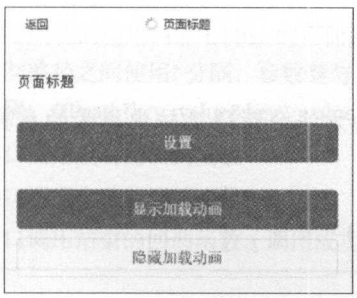

图 8-10 显示/隐藏加载动画示意图

例如对于一个搜索框，我们可以在按钮点击事件中先调用显示的 API，再调用网络请求，然后在 success 的回调函数中调用隐藏加载动画的 API。

8.7.3 导航

对一个微信小程序来说，多数情况下我们要使用多个页面来达到想要的效果，所以合理地在这些页面之间进行切换十分重要。

wx.navigateTo

wx.navigateTo(OBJECT) 即保留当前页面，跳转到应用内的某个页面，使用 wx.navigateBack 可以返回到原页面。

OBJECT 参数说明如表 8-59 所示。

表 8-59

参数	类型	必填	说明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径，路径后可以带参数。参数与路径之间使用? 分隔，参数键与参数值用 = 相连，不同参数用 & 分隔；如 'path?key=value&key2=value2'
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）

示例代码：

```
//准备开始跳转的页面
wx.navigateTo({
  url: 'test?id=1'
})
```

```
//跳转到的页面，通过下面的方法会获得跳转中携带的参数“id=1”
Page({
  onLoad: function(option){
    console.log(option.query)
  }
})
```

注意：为了不让用户在使用微信小程序时造成困扰，我们规定页面路径只能是五层，请尽量避免多层级的交互方式。

从原理上讲，微信小程序的页面是由栈管理的。栈就像一个盒子，如图 8-11 所示。

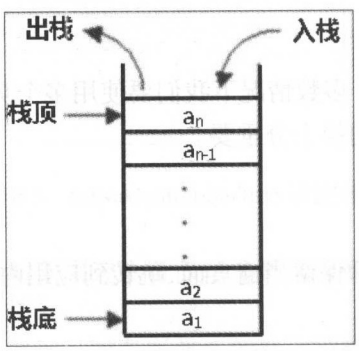

图 8-11 栈模型示意图

这里面的 $a_1...a_n$ 就是我们的页面，a_1 为首页，每次使用 navigateTo，就会有新的页面（依次为 a_2，a_3...）加入到这个盒子中。对微信小程序而言，当向盒子里加入了 a_5 之后（即连续调用了 4 次 navigateTo），再调用 navigateTo 的话就会失败。一般情况下，微信小程序不会出现这种问题，只有非常复杂的微信小程序需要注意这一点，解决这个问题的方法是使用下面要介绍的 redirectTo 代替 navigateTo 来实现页面跳转。

wx.redirectTo

wx.redirectTo(OBJECT) 即关闭当前页面，跳转到应用内的某个页面。

OBJECT 参数说明如表 8-60。

表 8-60

参数	类型	必填	说明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径，路径后可以带参数。参数与路径之间使用? 分隔，参数键与参数值用 = 相连，不同参数用 & 分隔；如'path?key=value&key2=value2'
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）

示例代码：

```
wx.redirectTo({
  url: 'test?id=1'
})
```

redirectTo 的用法与 navigateTo 的用法一样，这两者的区别在于 navigateTo 在调用时将新页面入栈，而使用 redirectTo 会先将当前页面出栈（即关闭当前页面），再将新的页面入栈并打开。所以使用 redirectTo 不会导致微信小程序页面的层数增长，也就不会导致页面层数超过 5 层。

使用 redirectTo 与 navigateTo 的区别还在于，用户在页面跳转之后点击“返回”按钮，例如从首页（index）使用 navigateTo 跳转到页面 A，然后再调用 navigateTo 到页面 B，这时点击“返回”按钮，页面 B 出栈，当前显示的页面返回到跳转前的页面 A。而如果我们在页面 A 使用 redirectTo 跳转到页面 B，这时点击返回，因为页面 A 已经被出栈了，所以页面会返回到首页上，这在一定程度上会让用户混乱。因此建议在进行页面

跳转的时候，尽可能使用 navigateTo，只有在特定情况或页面层数超过 5 层再考虑使用 redirectTo。

wx.switchTab

wx.switchTab(OBJECT) 即跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面。

OBJECT 参数说明如表 8-61 所示。

表 8-61

参数	类型	必填	说明
url	String	是	需要跳转的 tabBar 页面的路径（需要在 app.json 的 tabBar 字段定义的页面），路径后不能带参数
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）

示例代码：

```
//定义在app.json里设置页面底部的选项卡
{
  "tabBar": {
    "list": [{
      "pagePath": "index",
      "text": "首页"
    },{
      "pagePath": "other",
      "text": "其他"
    }]
  }
}
```

```
//调用API切换选项卡
wx.switchTab({
  url: '/index'
})
```

注意：wx.navigateTo 和 wx.redirectTo 不允许跳转到 tabbar 页面，只能用 wx.switchTab 跳转到 tabbar 页面。

wx.navigateBack

wx.navigateBack(OBJECT) 即关闭当前页面，返回上一页面或多级页面。可通过 getCurrentPages() 获取当前的页面栈，决定需要返回几层。

OBJECT 参数说明如表 8-62 所示。

表 8-62

参数	类型	默认值	说明
delta	Number	1	返回的页面数，如果 delta 大于现有页面数，则返回到首页。

示例代码：

```
// 注意：调用 navigateTo 跳转时，调用该方法的页面会被加入堆栈，而 redirectTo 方法则不会。见下方示例代码

// 此处是A页面
wx.navigateTo({
  url: 'B?id=1'
})

// 此处是B页面
wx.navigateTo({
  url: 'C?id=1'
})

// 在C页面内 navigateBack，将返回A页面
wx.navigateBack({
  delta: 2
})
```

使用 navigateBack 可以使用户一次后退多个页面，参数 delta 的值表示要后退页面的个数（也就是页面从栈里出栈的个数）。

这个方法会使我们的页面跳转更灵活，尤其是像返回首页这种功能，使用 navigateBack 很容易实现。如果当前页面的层数不容易计算或层数为动态值，则可以先用 getCurrentPages() 来获取当前栈里有多少个页面，然后再调用 navigateBack。

8.7.4 动画

wx.createAnimation

wx.createAnimation(OBJECT) 即创建一个动画实例 animation。调用实例的方法来描述动画，最后通过动画实例的 export 方法导出动画数据传递给组件的 animation 属性。

OBJECT 参数说明如表 8-63 所示。

表 8-63

参数	类型	必填	说明
duration	Integer	否	动画持续时间，单位：ms，默认值为 400
timingFunction	String	否	定义动画的效果，默认值为"linear"，有效值为"linear","ease","ease-in","ease-in-out","ease-out","step-start","step-end"
delay	Integer	否	动画延迟时间，单位：ms，默认值为 0
transformOrigin	String	否	设置 transform-origin，默认值为"50% 50% 0"

示例代码：

```
var animation = wx.createAnimation({
  transformOrigin: "50% 50%",
  duration: 1000,
  timingFunction: "ease",
  delay: 0
})
```

这四个属性里比较简单的是 duration 与 delay，timingFunction 定义了动画的播放方式，其定义如下：

- linear：默认为 linear 动画一直较为均匀。
- ease：开始时缓慢，中间加速，到快结束时减速。
- ease-in：开始的时候缓慢。
- ease-in-out：开始和结束时减速。
- ease-out：结束时减速。
- step-start：动画一开始就跳到 100% 直到动画持续时间结束一闪而过。
- step-end：保持 0% 的样式直到动画持续时间结束一闪而过。

在示例代码里定义了 timingFunction 为 ease，如果动画效果为从左向右平移，则设置 ease 会导致一开始动画移动速度较慢，最后逐渐加速，在路程中间速度达到最大值，然后再减速直到终点时停止。不设置此属性的话，则采用默认值则会已匀速平移。

在解释 transformOrigin 之前我们先明确微信小程序上的坐标轴（见图 8-12）：

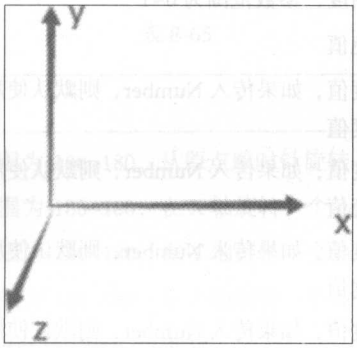

图 8-12 微信小程序坐标示意图

X 轴和 Y 轴分别代表了手机上的水平方向与竖直方向，Z 轴则是垂直于手机屏幕的轴。

transformOrigin 里的三个参数就是在这三个轴上的坐标，通过这三点来获取空间上的一个点，动画效果则是以这个点为中心来实现的。默认值为“50% 50% 0”，也就是组件的中心位置，改变这个属性会影响动画的效果，在后面会举例说明。

animation

动画实例可以调用以下方法来描述动画，调用结束后会返回自身，支持链式调用的写法，具体的步骤如下。

(1) 创建一个动画实例 animation。用调用实例的方法来描述动画，最后通过动画实例的 export 方法导出动画数据传递给组件的 animation 属性。

(2) 调用动画操作方法后要调用 step() 来表示一组动画完成，可以在一组动画中调用任意多个动画方法，一组动画中的所有动画会同时开始，一组动画完成后才会进行下一组动画。step 可以传入一个与 wx.createAnimation() 一样的配置参数用于指定当前组动画的属性。

注意：export 方法每次调用后会清之前的动画操作。

样式（见表 8-64）

表 8-64

方法	参数	说明
opacity	value	透明度，参数范围为 0~1
backgroundColor	color	颜色值
width	length	长度值，如果传入 Number，则默认使用 px，可传入其他自定义单位的长度值
height	length	长度值，如果传入 Number，则默认使用 px，可传入其他自定义单位的长度值
top	length	长度值，如果传入 Number，则默认使用 px，可传入其他自定义单位的长度值
left	length	长度值，如果传入 Number，则默认使用 px，可传入其他自定义单位的长度值
bottom	length	长度值，如果传入 Number，则默认使用 px，可传入其他自定义单位的长度值
right	length	长度值，如果传入 Number，则默认使用 px，可传入其他自定义单位的长度值

这些属性读者肯定不陌生，这与我们在编写 WXSS 文件时使用的属性值一样，即我们可以通过调用动画 API 来直接更改组件的属性以达到动画的效果。

```
Page({
  data: {                                    //设置用于存储animation的data
    animationData: {}
  },
  onShow: function(){
    var animation = wx.createAnimation({      //创建animation实例
      duration: 1000,
        timingFunction: 'ease',
    })
    this.animation = animation                //获取实例
    animation.width(400).height(400).step()   //将组件的宽高设置为400
    this.setData({
      animationData:animation.export()   //使用export()将写好的动画效果加载进去
    })
})
```

在布局文件里我们为想要实现动画的组件添加 animation 的属性：

```
<view class="animation-element" animation="{{animation}}"></view>
```

如果这个组件的宽高为 200px，则我们的动画效果会使它变大一倍。

旋转（见表 8-65）

表 8-65

方法	参数	说明
rotate	deg	deg 的范围为-180~180，从原点顺时针旋转一个 deg 角度
rotateX	deg	deg 的范围为-180~180，在 X 轴旋转一个 deg 角度
rotateY	deg	deg 的范围为-180~180，在 Y 轴旋转一个 deg 角度
rotateZ	deg	deg 的范围为-180~180，在 Z 轴旋转一个 deg 角度
rotate3d	(x,y,z,deg)	同 transform-function rotate3d

缩放（见表 8-66）

表 8-66

方法	参数	说明
scale	sx,[sy]	一个参数时，表示在 X 轴、Y 轴同时缩放 sx 倍数；两个参数时表示在 X 轴缩放 sx 倍数，在 Y 轴缩放 sy 倍数
scaleX	sx	在 X 轴缩放 sx 倍数
scaleY	sy	在 Y 轴缩放 sy 倍数
scaleZ	sz	在 Z 轴缩放 sy 倍数
scale3d	(sx,sy,sz)	在 X 轴缩放 sx 倍数，在 Y 轴缩放 sy 倍数，在 Z 轴缩放 sz 倍数

偏移（见表 8-67）

表 8-67

方法	参数	说明
translate	tx,[ty]	一个参数时，表示在 X 轴偏移 tx，单位：px；两个参数时，表示在 X 轴偏移 tx，在 Y 轴偏移 ty，单位：px
translateX	tx	在 X 轴偏移 tx，单位：px

续表

方法	参数	说明
translateY	ty	在 Y 轴偏移 tx，单位：px
translateZ	tz	在 Z 轴偏移 tx，单位：px
translate3d	(tx,ty,tz)	在 X 轴偏移 tx，在 Y 轴偏移 ty，在 Z 轴偏移 tz，单位：px

倾斜（见表 8-68）

表 8-68

方法	参数	说明
skew	ax,[ay]	参数范围为-180~180；一个参数时，Y 轴坐标不变，X 轴坐标延顺时针倾斜 ax 度；两个参数时，分别在 X 轴倾斜 ax 度，在 Y 轴倾斜 ay 度
skewX	ax	参数范围为-180~180；Y 轴坐标不变，X 轴坐标延顺时针倾斜 ax 度
skewY	ay	参数范围为-180~180；X 轴坐标不变，Y 轴坐标延顺时针倾斜 ay 度

矩阵变形（见表 8-69）

表 8-69

方法	参数	说明
matrix	(a,b,c,d,tx,ty)	同 transform-function matrix
matrix3d		同 transform-function matrix3d

示例代码：

```
<view class="animation-element-wrapper">
    <view class="animation-element" animation="{{animation}}"></view>
</view>
<view class="btn-area">
    <button class="animation-button" bindtap="rotate">旋转</button>
    <button class="animation-button" bindtap="scale">缩放</button>
    <button class="animation-button" bindtap="translate">移动</button>
    <button class="animation-button" bindtap="skew">倾斜</button>
</view>
```

```
Page({
  onReady: function () {
    this.animation = wx.createAnimation()
  },
  rotate: function () {
    this.animation.rotate(Math.random() * 720 - 360).step()
    this.setData({animation: this.animation.export()})
  },
  scale: function () {
    this.animation.scale(Math.random() * 2).step()
    this.setData({animation: this.animation.export()})
  },
  translate: function () {
    this.animation.translate(Math.random() * 100 - 50, Math.random() * 100 - 50).
        step()
    this.setData({animation: this.animation.export()})
  },
  skew: function () {
    this.animation.skew(Math.random() * 90, Math.random() * 90).step()
    this.setData({animation: this.animation.export()})
  },
})
```

这里面使用了 Math.random() 来随机生成一个 0~1 的数，以此来实现每次点击旋转随机角度等效果。

运行结果如下：

原图如图 8-13 所示。

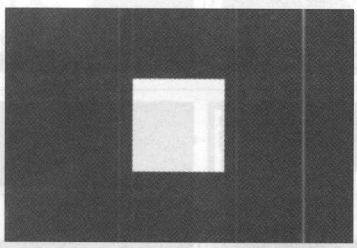

图 8-13 动画 API 示例——原图

动画后如图 8-14 所示。

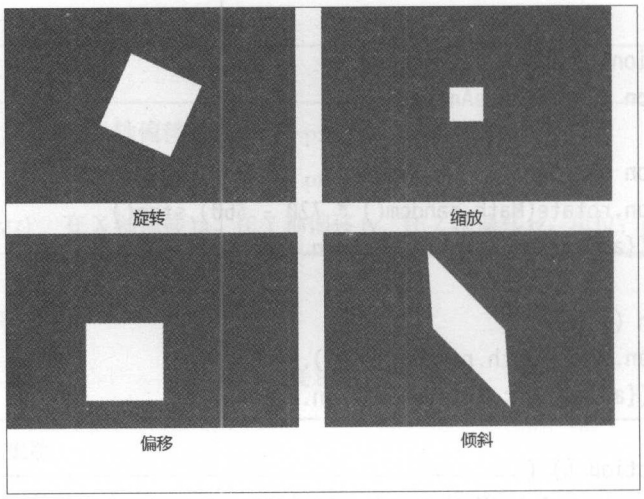

图 8-14 动画 API 示例——效果图

回到一开始所说的问题，现在我们在 createAnimation() 里添加 transformOrigin 属性。

```
onReady: function () {
    this.animation = wx.createAnimation({
      transformOrigin:"0% 0%"
    })
  },
```

结果如图 8-15 所示。

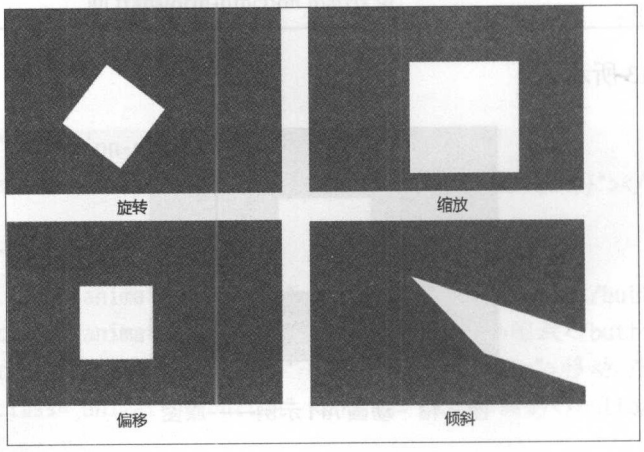

图 8-15 动画 API 示例-改变动画基准点后效果图

可以看到除偏移以外，其他结果都有了明显变化，这是因为动画的中心点发生了变化。以旋转来说，默认（50% 50%）是以白色方块的中心旋转，所以在旋转后白色方块还处于整个屏幕中间，而改成“0% 0%”之后，旋转动画就是以白色方块的左上角坐标来旋转了，所以产生了不同的效果。

除这样一次执行一个动画以外，我们还可以让多个动画效果同时发生：

```
rotateAndScale: function () {
    this.animation.rotate(Math.random() * 720 - 360)
        .scale(Math.random() * 2)
        .step()
    this.setData({animation: this.animation.export()})
  },
```

也可以让动画依次按顺序发生：

```
rotateThenScale: function () {
    this.animation.rotate(Math.random() * 720 - 360).step()
        .scale(Math.random() * 2).step()
    this.setData({animation: this.animation.export()})
  },
```

可以看到代码通过 step() 来将动画分为一段段，每个 step 前的动画会同时发生，多个 step 依次发生。

另外，在参数表里未详细说明的 rotate3d 表示组件在三维空间旋转，4 个参数（x，y，z，deg）分别代表是否沿 X 轴、Y 轴、Z 轴旋转以及旋转的角度，“1”表示是，“0”表示否。例如（1，0，0，45deg）表示沿 X 轴旋转 45 度。

矩阵部分：matrix 利用传入的 6 个参数构建矩阵，再用这个矩阵与现有坐标相乘，从而使坐标改变，达到动画效果。实际上本节所有的动画 API 都是用矩阵完成的。图 8-16 说明了这种坐标是如何改变的。

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + cy + e \\ bx + dy + f \\ 0 + 0 + 1 \end{bmatrix}$$

图 8-16 matrix 原理图

现有坐标 (x,y) 调用 matrix(a,b,c,d,e,f) 后，坐标变为 (ax+cy+e,bx+dy+f)。

matrix3d 则更复杂，需要的参数也更多，但本质上与 matrix 相同。

8.7.5 绘画

在第 7 章简单介绍了画布组件，并且在例子中演示了其使用方法。本节会更加仔细介绍微信小程序为帮助我们使用画布组件而提供的 API。

Canvas 的坐标系：画布组件（canvas）处于二维的网格中，使用坐标（x，y）来表示位置。规定左上角的坐标为（0，0），越向右，x 的值越大，越向下，y 的值越大。

获取 context 对象

绘图的第一步：创建一个 Canvas 绘图的上下文，也就是绘图的场景。具体方法有：

(1) wx.createCanvasContext(canvasId)：创建 canvas 绘图上下文（指定 canvasId）。

(2) wx.createContext (不推荐使用)：创建并返回绘图上下文。

(3) drawCanvas (不推荐使用)：用所提供的 actions 在所给的 canvas-id 对应的 canvas 上进行绘图。

(4) canvasToTempFilePath(OBJECT)：把当前画布的内容导出生成图片，并返回文件路径。

颜色、样式和阴影

(1) setFillStyle：设置填充色。

(2) setStrokeStyle：设置边框颜色。

(3) setShadow：设置阴影样式。

渐变

(1) createLinearGradient：创建一个线性的渐变颜色。

(2) createCircularGradient：创建一个圆形的渐变颜色。

(3) addColorStop：创建一个颜色的渐变点。

矩形

(1) rect：创建一个矩形，需要用 fill() 或者 stroke() 方法将矩形真正画到 canvas 中。

(2) fillRect：填充一个矩形。

(3) strokeRect：画一个矩形 (非填充)。

(4) clearRect：清除画布上在该矩形区域内的内容。

线条样式

(1) setLineWidth：设置线条的宽度。

(2) setLineCap：设置线条的端点样式。

(3) setLineJoin：设置线条的交点样式。

(4) setMiterLimit：设置最大斜接长度，斜接长度指的是在两条线交汇处内角和外角之间的距离。

路径

(1) fill：对当前路径中的内容进行填充。

(2) stroke：画出当前路径的边框。

(3) beginPath：开始创建一个路径，需要调用 fill 或者 stroke 才会使用路径进行填充或描边。

(4) closePath：关闭一个路径，关闭路径会连接起点和终点。

(5) moveTo：把路径移动到画布中的指定点，不创建线条。

(6) lineTo：增加一个新点，然后创建一条从上次指定点到目标点的线。用 stroke() 方法来画线条。

(7) arc：画一条弧线。

(8) quadraticCurveTo：创建二次贝塞尔曲线路径。

(9) bezierCurveTo：创建三次方贝塞尔曲线路径。

变形

(1) scale：在调用 scale 方法后，之后创建的路径其横纵坐标会被缩放。

(2) rotate：以原点（默认为（0，0））为中心，顺时针旋转当前坐标轴。

(3) translate：对当前坐标系的原点 (0, 0) 进行变换，默认的坐标系原点为页面左上角。

文字与图片

(1) setFontSize：设置字体的字号。数字越大，字体越大。

(2) fillText：在画布指定位置绘制被填充的文本。

(3) drawImage：绘制图像，图像源可以是本地图片或网络图片。

其他

(1) setGlobalAlpha：设置全局画笔透明度。

(2) save：保存当前的绘图上下文。

(3) restore：恢复之前保存的绘图上下文。

(4) draw：将之前在绘图上下文中的描述（路径、变形、样式）画到 canvas 中。绘图上下文需要由 wx.createCanvasContext(canvasId) 来创建。

(5) getActions (不推荐使用)：返回绘图上下文的绘图动作。

(6) clearActions (不推荐使用)：清空绘图上下文的绘图动作。

在大多数的情况下，我们只需要使用微信小程序为我们提供的组件即可达成需求，真正需要我们自己去绘图的情况比较少，所以这里只罗列了绘图 API 的所有方法，具体参数以及使用方法请查阅官方文档。

8.7.6 下拉刷新

onPullDownRefresh

onPullDownRefresh 为在 Page 中定义 onPullDownRefresh 处理函数，监听该页面用户下拉刷新事件。下拉刷新是最常用的功能，基本上所有微信小程序，只要需要访问网络获取数据都需要下拉刷新的功能。

需要在 config 的 window 选项中开启 enablePullDownRefresh。

在需要下拉刷新的页面，向.json 文件里添加：

```
{
    "navigationBarTitleText": "下拉刷新",
    "enablePullDownRefresh": true
}
```

当处理完数据刷新后，wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。也就是说，在多数情况下我们需要在 request 的回调函数里调用这个 API。

wx.stopPullDownRefresh

wx.stopPullDownRefresh() 即停止当前页面下拉刷新。

示例代码：

```
Page({
  onPullDownRefresh: function () {
    wx.showToast({                                    //开始刷新后弹出“载入中”的消息
        提示框
      title: 'loading...',
      icon: 'loading'
    })
  },
  stopPullDownRefresh: function () {
    wx.stopPullDownRefresh({
      complete: function (res) {                      //停止刷新后，隐藏提示框
        wx.hideToast()
      }
    })
  }
})
```

8.8 开放接口

除上述的接口外，微信官方还提供了一些开放接口，如果你接触过微信公众号的开发，就会发现这些接口大部分实现起来差不多，具体包括以下几种。

(1) 登录

(2) 获取用户信息

(3) 使用微信支付

(4) 模板消息

(5) 客服消息

(6) 分享

(7) 获取二维码

8.8.1 登录

wx.login

wx.login(OBJECT) 为调用接口获取登录凭证（code）进而换取用户登录态信息，包括用户的唯一标识（openid）及本次登录的会话密钥（session_key）。用户数据的加解密通信需要依赖会话密钥完成。

OBJECT 参数说明如表 8-70 所示。

表 8-70

参数	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）

success 返回参数说明表如表 8-71 所示。

表 8-71

参数	类型	说明
errMsg	String	调用结果
code	String	用户允许登录后，回调内容会带上 code（有效期五分钟），开发者需要将 code 发送到开发者服务器后台，使用 code 换取 session_key api，将 code 换成 openid 和 session_key

示例代码：

```
//app.js
App({
  onLaunch: function() {
    wx.login({
      success: function(res) {
        if (res.code) {
          //发起网络请求
          wx.request({
            url: 'https://test.com/onLogin',
            data: {
              code: res.code
            }
          })
        } else {
          console.log('获取用户登录态失败！' + res.errMsg)
        }
      }
```

```
    });
  }
})
```

code 换取 session_key

这是一个 HTTPS 接口，开发者服务器使用登录凭证 code 获取 session_key 和 openid。其中 session_key 是对用户数据进行加密签名的密钥。为了自身应用安全，session_key 不应该在网络上传输。

接口地址：

```
https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=
    JSCODE&grant_type=authorization_code
```

请求参数如表 8-72 所示。

表 8-72

参数	必填	说明
appid	是	小程序唯一标识
secret	是	小程序的 app secret
js_code	是	登录时获取的 code
grant_type	是	填写为 authorization_code

返回参数如表 8-73 所示。

表 8-73

参数	说明
openid	用户唯一标识
session_key	会话密钥

返回说明：

```
//正常返回的JSON数据包
{
    "openid": "OPENID",
    "session_key": "SESSIONKEY"
```

```
}
//错误时返回JSON数据包(示例为Code无效)
{
    "errcode": 40029,
    "errmsg": "invalid code"
}
```

登录态维护

通过 wx.login 获取到用户登录态之后，需要维护登录态。开发者要注意不应该直接把 session_key、openid 等字段作为用户的标识或者 session 的标识，而应该自己派发一个 session 登录态（请参考登录时序图）。对于开发者自己生成的 session，应该保证其安全性且不应该设置较长的过期时间。session 派发到微信小程序客户端之后，可将其存储在 storage，用于后续通信使用。

登录时序图

登录需要和自己的程序服务器后台同时完成，如图 8-17 所示。

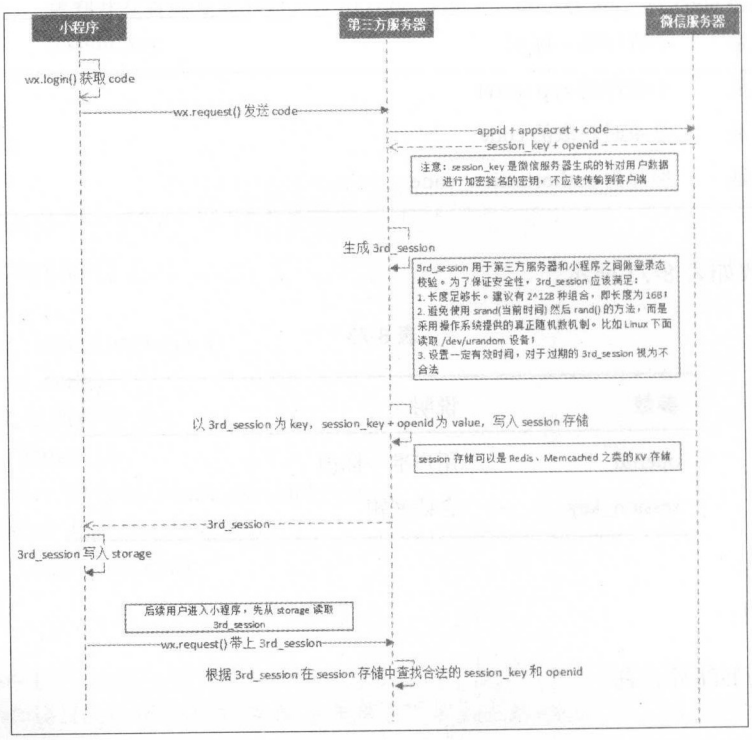

图 8-17　登录时序图

wx.checkSession

wx.checkSession(OBJECT) 即检查登录态是否过期，如表 8-74 所示。

表 8-74

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，登录态未过期
fail	Function	否	接口调用失败的回调函数，登录态已过期
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.checkSession({
  success: function(){
    //登录态未过期
  },
  fail: function(){
    //登录态过期
    wx.login()
  }
})
```

注意：在 iOS/Android 6.3.30 中，在 App.onLaunch 调用 wx.login 会出现异常。

数据签名校验

数据签名校验介绍

为了确保开放接口返回用户数据的安全性，微信会对明文数据进行签名。开发者可以根据业务需要对数据包进行签名校验，确保数据的完整性。

(1) 签名校验算法涉及用户的 session_key，通过 wx.login 登录流程获取用户 session_key，并自行维护与应用自身登录态的对应关系。

(2) 通过调用接口（如 wx.getUserInfo）获取数据时，接口会同时返回 rawData、signature，其中 signature = sha1(rawData + session_key)。

(3) 开发者将 signature、rawData 发送到开发者服务器进行校验。服务器利用用户对应的 session_key 使用相同的算法计算出签名 signature2，比对 signature 与 signature2 即可校验数据的完整性。

如 wx.getUserInfo 的数据校验：

接口返回的 rawData：

```
{
  "nickName": "Band",
  "gender": 1,
  "language": "zh_CN",
  "city": "Guangzhou",
  "province": "Guangdong",
  "country": "CN",
  "avatarUrl": "http://wx.qlogo.cn/mmopen/vi_32/1vZvI39NWFQ9XM4LtQpFrQJ1xlgZxx3w7b
QxKARol6503Iuswjjn6nIGBiaycAjAtpujxyzYsrztuuICqIM5ibXQ/0"
}
```

用户的 session-key：

```
HyVFkGl5F5OQWJZZaNzBBg==
```

所以，用于签名的字符串为：

```
{"nickName":"Band","gender":1,"language":"zh_CN","city":"Guangzhou","province":
"Guangdong","country":"CN","avatarUrl":"http://wx.qlogo.cn/mmopen/vi_32/1vZvI39NWF
Q9XM4LtQpFrQJ1xlgZxx3w7bQxKARol6503Iuswjjn6nIGBiaycAjAtpujxyzYsrztuuICqIM5ibXQ/0"}
HyVFkGl5F5OQWJZZaNzBBg==
```

使用 sha1 得到的结果为：

```
75e81ceda165f4ffa64f4068af58c64b8f54b88c
```

加密数据解密算法

接口如果涉及敏感数据（如 wx.getUserInfo 当中的 openId 和 unionId ），接口的明文内容将不包含这些敏感数据。开发者如果需要获取敏感数据，则需要对接口返回的加密数据 (encryptedData) 进行对称解密。解密算法如下：

(1) 对称解密使用的算法为 AES-128-CBC，数据采用 PKCS#7 填充。

(2) 对称解密的目标密文为 Base64_Decode(encryptedData)。

(3) 对称解密秘钥 aeskey = Base64_Decode(session_key)，aeskey 是 16 字节。

(4) 对称解密算法初始向量 iv 会在数据接口中返回。

微信官方提供了多种编程语言的示例代码（https://mp.weixin.qq.com/debug/wxadoc/dev/demo/aes-sample.zip）。每种语言类型的接口名字均一致。调用方式可以参照示例。

另外，为了应用能校验数据的有效性，我们会在敏感数据加上数据水印 (watermark)。

watermark 参数说明如表 8-75 所示。

表 8-75

参数	类型	说明
watermark	OBJECT	数据水印
appid	String	敏感数据归属 AppID，开发者可校验此参数与自身 AppID 是否一致
timestamp	DateInt	敏感数据获取的时间戳，开发者可以用于数据时效性校验

如接口 wx.getUserInfo 敏感数据当中的 watermark：

```
{
    "openId": "OPENID",
    "nickName": "NICKNAME",
    "gender": GENDER,
    "city": "CITY",
    "province": "PROVINCE",
    "country": "COUNTRY",
    "avatarUrl": "AVATARURL",
    "unionId": "UNIONID",
    "watermark":
    {
        "appid":"APPID",
        "timestamp":TIMESTAMP
    }
}
```

注意：

此前提供的加密数据（encryptData）以及对应的加密算法将被弃用，请开发者不要再依赖旧逻辑。

8.8.2 用户信息

wx.getUserInfo

wx.getUserInfo(OBJECT) 为获取用户信息，需要先调用 wx.login 接口。

OBJECT 参数说明如表 8-76 所示。

表 8-76

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）

success 返回参数说明如表 8-77 所示。

表 8-77

参数	类型	说明
userInfo	OBJECT	用户信息对象，不包含 openid 等敏感信息
rawData	String	不包括敏感信息的原始数据字符串，用于计算签名。
signature	String	使用 sha1(rawData + sessionkey) 得到字符串，用于校验用户信息。
encryptedData	String	包括敏感数据在内的完整用户信息的加密数据，详细见登录中的数据解密算法
iv	String	加密算法的初始向量，详细见登录中的数据解密算法

示例代码：

```
wx.getUserInfo({
  success: function(res) {
    var userInfo = res.userInfo
    var nickName = userInfo.nickName
    var avatarUrl = userInfo.avatarUrl
    var gender = userInfo.gender //性别 0：未知、1：男、2：女
    var province = userInfo.province
    var city = userInfo.city
    var country = userInfo.country
  }
})
```

encryptedData 解密后为以下 JSON 结构，详见登录中的数据解密算法。

```
{
    "openId": "OPENID",
    "nickName": "NICKNAME",
```

```
    "gender": GENDER,
    "city": "CITY",
    "province": "PROVINCE",
    "country": "COUNTRY",
    "avatarUrl": "AVATARURL",
    "unionId": "UNIONID",
    "watermark":
    {
        "appid":"APPID",
    "timestamp":TIMESTAMP
    }
}
```

UnionID 机制说明：如果开发者拥有多个移动应用、网站应用和公众账号（包括微信小程序），可通过 unionid 来区分用户的唯一性，因为只要是同一个微信开放平台账号下的移动应用、网站应用和公众账号（包括微信小程序），用户的 unionid 是唯一的。换句话说，同一用户，对同一个微信开放平台下的不同应用，unionid 是相同的。

微信开放平台绑定微信小程序流程

前提：微信开放平台账号必须已完成开发者资质认证。

开发者资质认证流程：

登录微信开放平台 (open.weixin.qq.com) -账号中心-开发者资质认证页面，如图 8-18 所示。

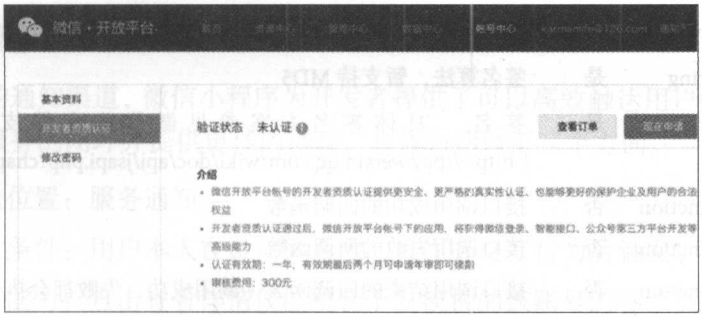

图 8-18 认证开发资质

绑定流程：登录微信开放平台（open.weixin.qq.com）—管理中心—公众号—绑定公众账号页面，如图 8-19 所示。

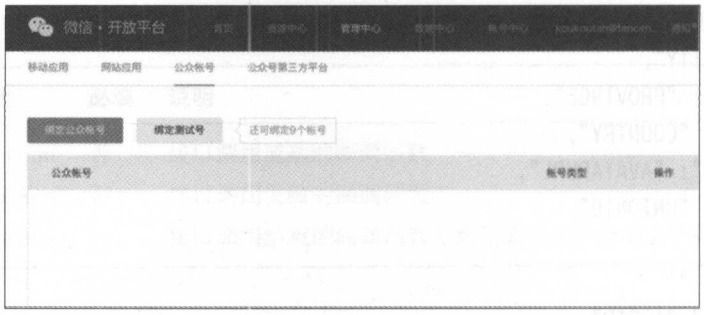

图 8-19 绑定公众账号

注意：wx.getUserInfo 接口需要用户授权，请兼容用户拒绝授权的场景。

8.8.3 微信支付

wx.requestPayment

wx.requestPayment(OBJECT) 为发起微信支付。

Object 参数说明如表 8-78 所示。

表 8-78

参数	类型	必填	说明
timeStamp	String	是	时间戳从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前的时间
nonceStr	String	是	随机字符串，长度为 32 个字符以下
package	String	是	统一下单接口返回的 prepay_id 参数值，提交格式如：prepay_id=*
signType	String	是	签名算法，暂支持 MD5
paySign	String	是	签名，具体签名方案参见微信公众号支付帮助文档（https://pay.weixin.qq.com/wiki/doc/api/jsapi.php?chapter=4_3）
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）

如果要了解更多信息，请查看微信支付接口文档（https://pay.weixin.qq.com/wiki/doc/api/wxa/wxa_api.php?chapter=7_3&index=1）。

回调结果如表 8-79 所示。

表 8-79

回调类型	errMsg	说明
success	requestPayment:ok	调用支付成功
fail	requestPayment:fail cancel	用户取消支付
fail	requestPayment:fail (detail message)	调用支付失败，其中 detail message 为后台返回的详细失败原因

示例代码：

```
wx.requestPayment({
   'timeStamp': '',
   'nonceStr': '',
   'package': '',
   'signType': 'MD5',
   'paySign': '',
   'success':function(res){
   },
   'fail':function(res){
   }
})
```

注意：在微信 6.5.2 及之前版本中，用户取消支付不会触发 fail 回调，只会触发 complete 回调，回调 errMsg 为 'requestPayment:cancel'。

8.8.4 模板消息

基于微信的通知渠道，微信小程序为开发者提供了可以高效触达用户的模板消息能力，以便实现服务的闭环并提供更佳的体验，具体包括以下三个方面。

- 模板推送位置：服务通知。
- 模板下发条件：用户本人在微信体系内与页面有交互行为后触发。
- 模板跳转能力：点击查看详情仅能跳转下发模板的该账号的各个页面。

使用说明

(1) 获取模板 id。登录 https://mp.weixin.qq.com 获取模板，如果没有合适的模板，则可以申请添加新模板，审核通过后可使用，如图 8-20 所示。

图 8-20 模板消息

页面的<form/>组件的属性 report-submit 为 true 时，可以声明为需要发送模板消息，此时点击提交表单按钮可以获取 formId，用于发送模板消息。或者当用户完成支付行为，可以获取 prepay_id 用于发送模板消息。

(2) 调用接口下发模板消息。

接口说明

获取 access_token

access_token 是全局唯一接口调用凭据，开发者调用各接口时都需要使用 access_token，请妥善保存。access_token 的存储至少要保留 512 个字符的空间。access_token 的有效期目前为两个小时，需要定时刷新，重复获取将导致上次获取的 access_token 失效。

公众平台的 API 调用所需要的 access_token 的使用及生成方式说明如下。

(1) 为了保密 appsecrect，第三方需要一个 access_token 获取和刷新的中控服务器。而其他业务逻辑服务器所使用的 access_token 均来自于该中控服务器，不应该各自去刷新，否则会造成 access_token 覆盖而影响业务。

(2) 目前 access_token 的有效期通过返回的 expires_in 来传达，目前为 7200 秒之内的值。中控服务器需要根据这个有效时间提前去刷新 access_token。在刷新过程中，

中控服务器对外输出的依然是旧的 access_token，此时公众平台后台会保证在刷新短时间内，新旧 access_token 都可用，这样保证了第三方业务的平滑过渡。

(3) access_token 的有效时间可能会在未来有调整，所以中控服务器不仅需要内部定时主动刷新，还需要提供被动刷新 access_token 的接口，这样便于业务服务器在 API 调用获知 access_token 已超时的情况下，可以触发 access_token 的刷新流程。

开发者可以使用 AppID 和 AppSecret 调用本接口来获取 access_token。AppID 和 AppSecret 可以在微信公众平台官网—设置—开发设置中获得（需要已经绑定成为开发者，且账号没有异常状态）。AppSecret 生成后请自行保存，因为在公众平台每次生成查看都会导致 AppSecret 被重置。注意：调用所有微信接口时均需要使用 https 协议。如果第三方不使用中控服务器，而是选择各个业务逻辑点各自去刷新 access_token，那么就可能会产生冲突，导致服务器不稳定。

接口地址：https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET。

HTTP 请求方式：GET。

参数说明如表 8-80 所示。

表 8-80

参数	必填	说明
grant_type	是	获取 access_token 填写 client_credential
appid	是	第三方用户唯一凭证
secret	是	第三方用户唯一凭证密钥，即 appsecret

返回参数说明：在正常情况下，微信会返回下述 JSON 数据包给开发者。

```
{"access_token": "ACCESS_TOKEN", "expires_in": 7200}
```

具体参数说明参考表 8-81。

表 8-81

参数	说明
access_token	获取到的凭证
expires_in	凭证有效时间，单位：秒

错误时微信会返回错误码等信息，JSON 数据包示例如下（该示例为 AppID 无效错误）：

```
{"errcode": 40013, "errmsg": "invalid appid"}
```

发送模板消息

接口地址：https://api.weixin.qq.com/cgi-bin/message/wxopen/template/send?access_token=ACCESS_TOKEN(ACCESS_TOKEN 需要换成上文获取到的 access_token)。

HTTP 请求方式：POST。

POST 参数说明 (见表 8-82)：

表 8-82

参数	必填	说明
touser	是	接收者（用户）的 openid
template_id	是	所需下发的模板消息的 id
page	否	点击模板卡片后的跳转页面，仅限本微信小程序内的页面。支持带参数（示例 index?foo=bar）。该字段不填则模板无跳转
form_id	是	表单提交场景下，为 submit 事件带上的 formId；支付场景下，为本次支付的 prepay_id
value	是	模板内容，不填则下发空模板
color	否	模板内容字体的颜色，不填默认黑色
emphasis_keyword	否	模板需要放大的关键词，不填则默认无放大

示例：

```
{
  "touser": "OPENID",
  "template_id": "TEMPLATE_ID",
  "page": "index",
  "form_id": "FORMID",
  "data": {
      "keyword1": {
          "value": "339208499",
          "color": "#173177"
      },
      "keyword2": {
```

```
            "value": "2015年01月05日 12:30",
            "color": "#173177"
        },
        "keyword3": {
            "value": "粤海喜来登酒店",
            "color": "#173177"
        } ,
        "keyword4": {
            "value": "广州市天河区天河路208号",
            "color": "#173177"
        }
    },
    "emphasis_keyword": "keyword1.DATA"
}
```

返回码说明：

在调用模板消息接口后，会返回 JSON 数据包。

正常时返回 JSON 数据包的示例：

```
{
  "errcode": 0,
  "errmsg": "ok",
}
```

错误时会返回错误码信息，返回码说明如表 8-83 所示。

表 8-83

返回码	说明
40037	template_id 不正确
41028	form_id 不正确，或者过期
41029	form_id 已被使用
41030	page 不正确
45009	接口调用超过限额（目前默认每个帐号日调用限额为 100 万）

运行效果如图 8-21 所示。

注意：

微信 6.5.2 及以上版本支持模板功能。低于该版本将无法收到模板消息。

图 8-21 运行结果

8.8.5 客服消息

接收消息和事件

在页面中使用 <contact-button/> 可以显示进入客服会话页面按钮。

当用户在客服会话页面发送消息（或进行某些特定的用户操作引发的事件推送时），微信服务器会将消息（或事件）的数据包（JSON 或者 XML 格式）POST 请求开发者填写的 URL。开发者收到请求后可以使用发送客服消息接口进行异步回复。

微信服务器在将用户的消息发给微信小程序的开发者服务器地址（开发设置处配置）后，如果在五秒内收不到响应就会断掉连接，并且重新发起请求，总共重试三次。如果在调试中，发现用户无法收到响应的消息，则需要检查是否消息处理超时。

服务器收到请求必须做出下述回复，这样微信服务器才不会对此做任何处理，并且不会发起重试，否则，将出现严重的错误提示。

(1) 直接回复 success（推荐方式）。

(2) 直接回复空串（指字节长度为 0 的空字符串，而不是结构体中 content 字段的内容为空）。

一旦遇到以下情况，微信就会在小程序会话中，向用户下发系统提示“该小程序客服暂时无法提供服务，请稍后再试”：

(1) 开发者在5秒内未回复任何内容。

(2) 开发者回复了异常数据。

如果开发者希望增强安全性，则可以在开发者中心处开启消息加密，这样，用户发给微信小程序的消息以及微信小程序被动回复用户的消息都会继续加密。

发送客服消息

当用户和微信小程序客服产生特定动作的交互时，微信会把消息数据推送给开发者，开发者可以在一段时间内（目前修改为48小时）调用客服接口，通过POST的一个JSON数据包来发送消息给普通用户。此接口主要用于客服等有人工消息处理环节的功能，方便开发者为用户提供更加优质的服务。

目前允许的动作列表如表8-84所示，不同动作触发后，允许的客服接口下发消息条数和下发时限的不同可参考表8-84。当下发条数达到上限后，会收到错误。

表8-84

用户动作	允许下发条数限制	下发时限
用户通过客服消息按钮进入会话	1条	1分钟
用户发送信息	3条	48小时

客服接口——发消息

接口调用请求说明如下。

http请求方式：POST。

```
https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=ACCESS_TOKEN
```

接入概述

接入微信小程序消息服务，开发者需要按照如下步骤完成：

(1) 填写服务器配置。

(2) 验证服务器地址的有效性。

(3) 依据接口文档实现业务逻辑。

下面详细介绍这三个步骤。

第一步：填写服务器配置

登录微信小程序官网后，管理员在微信小程序官网的“设置-消息服务器”页面扫描二维码启用消息服务，填写服务器地址（URL）、Token 和 EncodingAESKey。

URL 是开发者用来接收微信消息和事件的接口 URL。Token 可由开发者任意填写，用作生成签名（该 Token 会和接口 URL 中包含的 Token 进行比对，从而验证安全性）。EncodingAESKey 由开发者手动填写或随机生成，将用作消息体的加解密密钥。

同时，开发者可选择消息加解密方式：明文模式、兼容模式和安全模式。还可以选择消息数据格式：XML 格式或 JSON 格式。加密方式的默认状态是明文格式，而数据格式的默认状态是 XML 格式。

模式的选择与服务器配置在提交后都会立即生效，请开发者谨慎填写及选择。切换加密方式和数据格式需要提前配置好相关代码。

第二步：验证消息的确来自微信服务器

开发者提交信息后，微信服务器将发送 GET 请求到填写的服务器地址 URL 上，GET 请求携带参数，如表 8-85 所示。

表 8-85

参数	描述
signature	微信加密签名，signature 结合了开发者填写的 token 参数和请求中的 timestamp 参数、nonce 参数
timestamp	时间戳
nonce	随机数
echostr	随机字符串

开发者通过检验 signature 对请求进行校验。若确认此次 GET 请求来自微信服务器，原样返回 echostr 参数内容，则接入生效，成功成为开发者，否则接入失败。加密/校验流程如下：

(1) 将token、timestamp、nonce三个参数进行字典序排序。

(2) 将三个参数字符串拼接成一个字符串进行 sha1 加密。

(3) 开发者获得加密后的字符串可与 signature 对比，标识该请求来源于微信。

PHP 示例代码下载地址：https://wximg.gtimg.com/shake_tv/mpwiki/cryptoDemo.zip。

第三步：依据接口文档实现业务逻辑。

验证 URL 有效性成功后即接入生效，成为开发者。至此用户向小程序客服发送消息或者进入会话页面等时，开发者填写的服务器配置 URL 将得到微信服务器推送过来的消息和事件，开发者可以依据自身业务逻辑进行响应。

另外要注意，开发者所填写的 URL 必须以 http:// 或 https:// 开头，分别支持 80 端口和 443 端口。

8.8.6 分享

可以在 Page 中定义 onShareAppMessage 函数，设置该页面的分享信息。

- 只有定义了此事件处理函数，在右上角的菜单中才会显示“分享”按钮。
- 用户点击“分享”按钮的时候会调用。
- 此事件需要return一个 Object，用于自定义分享内容。

自定义分享字段如表 8-86 所示。

表 8-86

字段	说明	默认值
title	分享标题	当前微信小程序名称
path	分享路径	当前页面 path，必须是以 / 开头的完整路径

示例代码：

```
Page({
  onShareAppMessage: function () {
    return {
      title: '自定义分享标题',
      path: '/page/user?id=123'
    }
  }
})
```

注意： 分享图片不能自定义；会取当前页面，从顶部开始，高度为 80% 屏幕宽度的图像作为分享图片。

8.8.7 获取二维码

通过后台接口可以获取微信小程序任意页面的二维码，扫描该二维码可以直接进入微信小程序对应的页面。

接口地址：

```
https://api.weixin.qq.com/cgi-bin/wxaapp/createwxaqrcode?access_token=ACCESS_TOKEN
```

可以通过微信公众号官方平台获取 TOKEN 信息。

POST 参数说明如表 8-87 所示。

表 8-87 POST 参数说明

参数	默认值	说明
path	—	不能为空，最大长度 128 字节
width	430	二维码的宽度

示例：

```
{"path": "pages/index?query=1", "width": 430}
```

注意：

- pages/index 需要在 app.json 的 pages 中定义。
- 通过该接口，仅能生成已发布的微信小程序的二维码。
- 可以在开发者工具预览时生成开发版的带参二维码。
- 带参数的二维码只有 100000 个，请谨慎调用。
- POST 参数需要转成 JSON 字符串，不支持 form 表单提交。

8.9 本章小结

本章已经详细阐述了微信小程序提供给开发者的八大类 API，用于实现各种具有原生功能体验的微信小程序处理能力。这些 API 是开发过程中的重点。如果想更加高效地完成项目，那么开发者需要熟练掌握本章内容。微信小程序开放的这些 API 数量和功能，也会随着小程序框架的升级完善发生细微的变化，但是会向上兼容。

9 组件进阶

在第7章介绍了微信小程序的组件，虽然微信小程序为开发者提供了一系列功能丰富的基础组件，但是还远远不够，我们还需要对基础组件进行合理组合，从而得到更加方便我们使用的组合组件。

微信小程序目前并不支持组件化，不能自定义组件，只能通过把封装的组件和样式放到单独的文件中，方便开发者移植使用。

本章介绍几种常见的组合组件。先创建新的工程，在app.wxss中定义一些公共的样式：

```
page {
  background-color: #f8f8f8;
  font-size: 32rpx;
}
.page__hd {
    padding: 80rpx;
}
.page__bd {
    padding-bottom: 80rpx;
}
.page__bd_spacing {
    padding-left: 30rpx;
```

```
    padding-right: 30rpx;
}

.page__ft{
    padding-bottom: 20rpx;
    text-align: center;
}

.page__title {
    text-align: left;
    font-size: 40rpx;
    font-weight: 400;
}

.page__desc {
    margin-top: 10rpx;
    color: #888888;
    text-align: left;
    font-size: 28rpx;
}
```

9.1 九宫格

九宫格是手机端比较常用的样式，如图 9-1 所示。

图 9-1　九宫格样式

如果你接触过 iOS 或者 Android 手机开发，那么可以通过tableView或者gridView轻松实现九宫格效果。

微信小程序里并没有相应的组件，需要通过基础控件组合成我们需要的效果。

分别创建grid.js、grid.wxml、grid.wxss，首先修改grid.wxml：

```
<view class="page">
    <view class="page__hd">
        <view class="page__title">Grid</view>
        <view class="page__desc">九宫格</view>
    </view>
    <!--以下是gird关键代码-->
    <view class="page__bd">
        <view class="weui-grids">
            <block wx:for-items="{{grids}}" wx:key="{{index}}">
                <navigator url="" class="weui-grid" hover-class="weui-grid_active">
                    <image class="weui-grid__icon" src="../images/icon_tabbar.png"
                        />
                    <view class="weui-grid__label">Grid</view>
                </navigator>
            </block>
        </view>
    </view>
</view>
```

grid.js代码比较简单，它只需要设置布局用到的data：

```
Page({
  data: {
    grids: [0, 1, 2, 3, 4, 5, 6, 7, 8]
  }
})
```

上面的代码通过grid.js传递给grid.xml grids 数组，然后再在布局文件中定义 <block wx:for-items="{{grids}}" wx:key="{{index}}"> 实现循环添加九宫格每个条目。

再来看看grid.wxss代码：

```
.weui-grids {
  border-top: 1rpx solid #D9D9D9;
  border-left: 1rpx solid #D9D9D9;
}
```

```
.weui-grid {
  position: relative;
  float: left;
  padding: 40rpx 20rpx;
  width: 33.33333333%;
  box-sizing: border-box;
  border-right: 1rpx solid #D9D9D9;
  border-bottom: 1rpx solid #D9D9D9;
}
.weui-grid_active {
  background-color: #ECECEC;
}
.weui-grid__icon {
  display: block;
  width: 56rpx;
  height: 56rpx;
  margin: 0 auto;
}
.weui-grid__label {
  margin-top: 10rpx;
  display: block;
  text-align: center;
  color: #000000;
  font-size: 28rpx;
  white-space: nowrap;
  text-overflow: ellipsis;
  overflow: hidden;
}
```

可以看到 gird 每个条目的宽度是 33.3333333%。每个条目右边和下边定义了一条分割线，整体左边和上边定义了分割线。

9.2 页脚

有时候 UI 设计的页面底部会有一个页脚，微信提倡的页脚风格如图 9-2 所示。

图 9-2 展示了几种不同的页脚风格，下面分别来实现一下。分别创建footer.js、footer.wxml和footer.wxss。

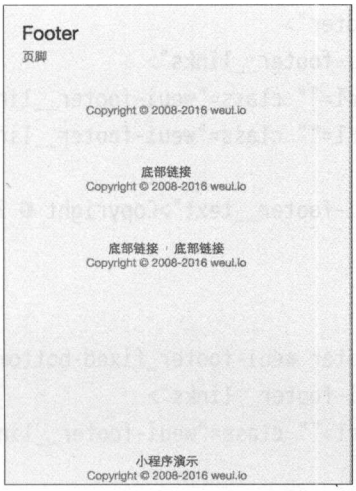

图 9-2 页脚

footer.js我们用一个空实现即可。

```
Page({})
```

主要看一下布局文件footer.wxml。footer.wxml代码如下:

```
<!--pages/footer/footer.wxml-->
<view class="page">
    <view class="page__hd">
        <view class="page__title">Footer</view>
        <view class="page__desc">页脚</view>
    </view>
    <!--页脚代码如下-->
    <view class="page__bd page__bd_spacing">
        <view class="weui-footer">
            <view class="weui-footer__text">Copyright © 2008-2016 weui.io</view>
        </view>

        <view class="weui-footer">
            <view class="weui-footer__links">
                <navigator url="" class="weui-footer__link">底部链接</navigator>
            </view>
            <view class="weui-footer__text">Copyright © 2008-2016 weui.io</view>
        </view>
```

```
        <view class="weui-footer">
            <view class="weui-footer__links">
                <navigator url="" class="weui-footer__link">底部链接</navigator>
                <navigator url="" class="weui-footer__link">底部链接</navigator>
            </view>
            <view class="weui-footer__text">Copyright © 2008-2016 weui.io</view>
        </view>


        <view class="weui-footer weui-footer_fixed-bottom">
            <view class="weui-footer__links">
                <navigator url="" class="weui-footer__link">小程序演示</navigator>
            </view>
            <view class="weui-footer__text">Copyright © 2008-2016 weui.io</view>
        </view>
    </view>
</view>
```

可以看到，我们只用了view和navigator组件就可以组合成页脚。我们重点关注一下样式文件footer.wxss:

```
.weui-footer{
    margin-bottom: 100rpx;
}
.weui-footer_fixed-bottom{
    margin-bottom: 0;
}

.weui-footer {
  color: #999999;
  font-size: 28rpx;
  text-align: center;
}
.weui-footer_fixed-bottom {
  position: fixed;
  bottom: .52em;
  left: 0;
  right: 0;
}
.weui-footer__links {
```

```
  font-size: 0;
}
.weui-footer__link {
  display: inline-block;
  vertical-align: top;
  margin: 0 .62em;
  position: relative;
  font-size: 28rpx;
  color: #586C94;
}
.weui-footer__link:before {
  content: " ";
  position: absolute;
  left: 0;
  width: 2rpx;
  border-left: 1rpx solid #C7C7C7;
  color: #C7C7C7;
  left: -0.65em;
  top: .36em;
  bottom: .36em;
}
.weui-footer__link:first-child:before {
  display: none;
}
.weui-footer__text {
  padding: 0 .34em;
  font-size: 24rpx;
}
```

这些样式我们都不陌生了。这样写完代码就实现图 9-2 显示的样式。

9.3 加载更多

我们经常看到手机 APP 列表界面中带上拉加载功能。如果微信小程序中也需要上拉加载功能，则需要在列表底部放一个加载中的控件和加载完成的控件，效果如图 9-3 所示。

下面介绍如何实现图 9-3 所示的效果。分别创建loadmore.js、loadmore.wxml、loadmore.wxss。loadmore.js不需要处理，空实现即可。

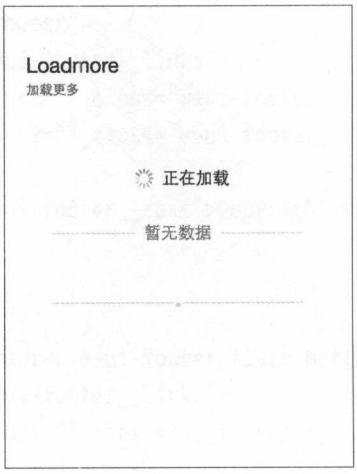

图 9-3 加载更多

先来看看布局文件loadmore.wxml的代码：

```
<view class="page">
    <view class="page__hd">
        <view class="page__title">Loadmore</view>
        <view class="page__desc">加载更多</view>
    </view>
    <view class="page__bd">
        <!--加载更多-->
        <view class="weui-loadmore">
            <view class="weui-loading"></view>
            <view class="weui-loadmore__tips">正在加载</view>
        </view>
        <!--暂无数据-->
        <view class="weui-loadmore weui-loadmore_line">
            <view class="weui-loadmore__tips weui-loadmore__tips_in-line">暂无数据<
                /view>
        </view>
        <!--中间显示一个点-->
        <view class="weui-loadmore weui-loadmore_line weui-loadmore_dot">
            <view class="weui-loadmore__tips weui-loadmore__tips_in-line weui-
                loadmore__tips_in-dot"></view>
        </view>
    </view>
</view>
```

可以看到，除了view这里没有用任何其他组件。主要看样式文件loadmore.wxss的代码：

```
page{
    background-color: #FFFFFF;
}

.weui-loading {
  margin: 0 5px;
  width: 20px;
  height: 20px;
  display: inline-block;
  vertical-align: middle;
  /**使用动画**/
  -webkit-animation: weuiLoading 1s steps(12, end) infinite;
          animation: weuiLoading 1s steps(12, end) infinite;
   /***通过svg创建加载中的矢量图*/
  background: transparent url(...省略...) no-repeat;
  background-size: 100%;
}
/**css3转圈动画**/
@-webkit-keyframes weuiLoading {
  0% {
    -webkit-transform: rotate3d(0, 0, 1, 0deg);
            transform: rotate3d(0, 0, 1, 0deg);
  }
  100% {
    -webkit-transform: rotate3d(0, 0, 1, 360deg);
            transform: rotate3d(0, 0, 1, 360deg);
  }
}
@keyframes weuiLoading {
  0% {
    -webkit-transform: rotate3d(0, 0, 1, 0deg);
            transform: rotate3d(0, 0, 1, 0deg);
  }
  100% {
    -webkit-transform: rotate3d(0, 0, 1, 360deg);
            transform: rotate3d(0, 0, 1, 360deg);
  }
```

```
}
.weui-loadmore {
  width: 65%;
  margin: 1.5em auto;
  line-height: 1.6em;
  font-size: 14px;
  text-align: center;
}
.weui-loadmore__tips {
  display: inline-block;
  vertical-align: middle;
}
.weui-loadmore_line {
  border-top: 1px solid #E5E5E5;
  margin-top: 2.4em;
}
.weui-loadmore__tips_in-line {
  position: relative;
  top: -0.9em;
  padding: 0 .55em;
  background-color: #FFFFFF;
  color: #999999;
}
.weui-loadmore__tips_in-dot {
  position: relative;
  padding: 0 .16em;
  width: 4px;
  height: 1.6em;
}
.weui-loadmore__tips_in-dot:before {
  content: " ";
  position: absolute;
  top: 50%;
  left: 50%;
  margin-top: -1px;
  margin-left: -2px;
  width: 4px;
  height: 4px;
  /**定义50%可以实现圆点效果**/
```

```
  border-radius: 50%;
  background-color: #E5E5E5;
}
```

在.weui-loading中，我们定义了 background 属性里面通过 svg 实现了一张矢量图（书中省略部分代码），svg 只需要了解即可，一般公司的 UI 设计师都可以通过一些软件自动生成。这里还使用了 CSS3 的动画实现了不停转圈的效果。最下面有一个圆形的点，通过定义相等的宽高加上border-radius: 50%;即可轻松实现。

9.4 导航条

导航条是比较常见的样式，如图 9-4 所示。

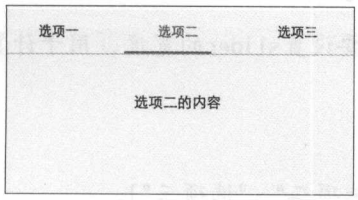

图 9-4 导航条

下面来实现它，分别创建navbar.js、navbar.wxml、navbar.wxss。

navbar.wxml代码如下：

```
<view class="page">
  <view class="page__bd">
    <view class="weui-tab">
      <view class="weui-navbar">
        <block wx:for-items="{{tabs}}" wx:key="{{index}}">
          <view id="{{index}}" class="weui-navbar__item {{activeIndex == index ? '
              weui-bar__item_on' : ''}}" bindtap="tabClick">
            <view class="weui-navbar__title">{{item}}</view>
          </view>
        </block>
        <view class="weui-navbar__slider" style="left: {{sliderLeft}}px; transform:
            translateX({{sliderOffset}}px)"></view>
      </view>
      <view class="weui-tab__panel">
```

```
        <view class="weui-tab__content" hidden="{{activeIndex != 0}}">选项一的内容
            </view>
        <view class="weui-tab__content" hidden="{{activeIndex != 1}}">选项二的内容
            </view>
        <view class="weui-tab__content" hidden="{{activeIndex != 2}}">选项三的内容
            </view>
    </view>
  </view>
 </view>
</view>
```

在布局文件中需要动态设置哪个选项卡高亮以及对应条目的内容是否隐藏。我们还需要动态设置选项卡底部滑条的位置，这些都需要通过逻辑层设置。下面具体看一下navbar.js代码是如何实现的:

```
var sliderWidth = 96; // 需要设置slider的宽度，用于计算中间位置

Page({
    data: {
        tabs: ["选项一", "选项二", "选项三"],
        activeIndex: "0",
        sliderOffset: 0,
        sliderLeft: 0
    },
    onLoad: function () {
        var that = this;
        wx.getSystemInfo({ //根据手机的宽度计算滑条的位置
            success: function(res) {
                that.setData({
                    sliderLeft: (res.windowWidth / that.data.tabs.length -
                        sliderWidth) / 2
                });
            }
        });
    },
    //选项卡的点击事件
    tabClick: function (e) {
        this.setData({
            sliderOffset: e.currentTarget.offsetLeft,
            activeIndex: e.currentTarget.id
```

```
        });
    }
});
```

这里通过data设置了三个选项卡，我们调用了微信小程序接口获取手机的宽度，根据手机宽度设置滑条左面的距离，通过这种方式可以适配不同的手机。

最后再来看一下样式文件navbar.wxss的代码:

```
page,
.page,
.page__bd{
    height: 100%;
}
.page__bd{
    padding-bottom: 0;
}
.weui-tab__content{
    padding-top: 120rpx;
    text-align: center;
}
.weui-navbar {
  display: flex;
  position: absolute;
  z-index: 500;
  top: 0;
  width: 100%;
  border-bottom: 1rpx solid #CCCCCC;
}
.weui-navbar__item {
  position: relative;
  display: block;
  -webkit-box-flex: 1;
  -webkit-flex: 1;
          flex: 1;
  padding: 26rpx 0;
  text-align: center;
  font-size: 0;
}
.weui-navbar__item.weui-bar__item_on {
  color: #1AAD19;
```

```
}
.weui-navbar__slider {
  position: absolute;
  content: " ";
  left: 0;
  bottom: 0;
  width: 6em;
  height: 6rpx;
  background-color: #1AAD19;
  transition: transform .3s;
}
.weui-navbar__title {
  display: inline-block;
  font-size: 30rpx;
  max-width: 8em;
  width: auto;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
  word-wrap: normal;
}
.weui-tab {
  position: relative;
  height: 100%;
}
.weui-tab__panel {
  /**border和padding计算入width之内*/
  box-sizing: border-box;
  height: 100%;
  padding-top: 100rpx;
  overflow: auto;
  -webkit-overflow-scrolling: touch;
}
```

9.5 搜索条

许多程序都有搜索功能，如图 9-5 所示。

点击搜索按钮展现样式如图 9-6 所示。

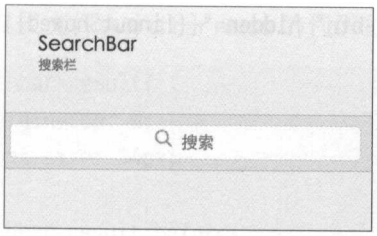

图 9-5 搜索条默认状态

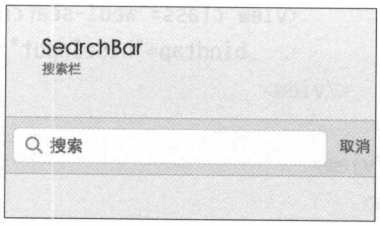

图 9-6 搜索条输入文字状态

下面就来实现图 9-5 和图 9-6 所示的效果，分别创建searchbar.js,searchbar.wxml,searchbar.wxss。searchbar.wxml代码如下：

```
<view class="page">
    <view class="page__hd">
        <view class="page__title">SearchBar</view>
        <view class="page__desc">搜索栏</view>
    </view>
    <!--下面是主要代码-->
    <view class="page__bd">
        <view class="weui-search-bar">
            <view class="weui-search-bar__form">
                <view class="weui-search-bar__box">
                    <icon class="weui-icon-search_in-box" type="search" size="14"><
                        /icon>
                    <input type="text" class="weui-search-bar__input" placeholder="
                        搜索" value="{{inputVal}}" focus="{{inputShowed}}"
                        bindinput="inputTyping" bindblur="hideInput" />
                    <view class="weui-icon-clear" wx:if="{{inputVal.length > 0}}"
                        bindtap="clearInput">
                        <icon type="clear" size="14"></icon>
                    </view>
                </view>
                <!--未输入时候显示-->
                <label class="weui-search-bar__label" hidden="{{inputShowed}}"
                    bindtap="showInput">
                    <icon class="weui-icon-search" type="search" size="14"></icon>
                    <view class="weui-search-bar__text">搜索</view>
                </label>
            </view>
            <!--取消按钮-->
```

```
            <view class="weui-search-bar__cancel-btn" hidden="{{!inputShowed}}"
                bindtap="hideInput">取消</view>
        </view>

    </view>
</view>
```

搜索图标可以用 icon 实现，逻辑层需要处理点击事件和监听用户输入的内容，来看一下searchbar.js：

```
Page({
    data: {
        inputShowed: false,
        inputVal: ""
    },
    //开始状态的点击事件
    showInput: function () {
        this.setData({
            inputShowed: true
        });
    },
    //取消按钮
    hideInput: function () {
        this.setData({
            inputVal: "",
            inputShowed: false
        });
    },
    //清空按钮点击事件
    clearInput: function () {
        this.setData({
            inputVal: ""
        });
    },
    //监听输入内容
    inputTyping: function (e) {
        this.setData({
            inputVal: e.detail.value
        });
    }
});
```

样式文件searchbar.wxss 的代码如下:

```
.searchbar-result{
    margin-top: 0;
    font-size: 14px;
}
.searchbar-result:before{
    display: none;
}
.weui-cell{
    padding: 12px 15px 12px 35px;
}

.weui-search-bar {
  position: relative;
  padding: 8px 10px;
  display: flex;
  box-sizing: border-box;
  background-color: #EFEFF4;
  border-top: 1rpx solid #D7D6DC;
  border-bottom: 1rpx solid #D7D6DC;
}
.weui-icon-search {
  margin-right: 8px;
}
.weui-icon-search_in-box {
  position: absolute;
  left: 10px;
  top: 7px;
}
.weui-search-bar__text {
  display: inline-block;
  font-size: 14px;
  vertical-align: middle;
}
.weui-search-bar__form {
  position: relative;
  flex: auto;
  border-radius: 5px;
  background: #FFFFFF;
```

```
  border: 1rpx solid #E6E6EA;
}
.weui-search-bar__box {
  position: relative;
  padding-left: 30px;
  padding-right: 30px;
  width: 100%;
  box-sizing: border-box;
  z-index: 1;
}
.weui-search-bar__input {
  height: 28px;
  line-height: 28px;
  font-size: 14px;
}
.weui-icon-clear {
  position: absolute;
  top: 0;
  right: 0;
  padding: 7px 8px;
  font-size: 0;
}
.weui-search-bar__label {
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 2;
  border-radius: 3px;
  text-align: center;
  color: #9B9B9B;
  background: #FFFFFF;
  line-height: 28px;
}
.weui-search-bar__cancel-btn {
  margin-left: 10px;
  line-height: 28px;
  color: #09BB07;
```

```
  white-space: nowrap;
}
```

读者应该对大部分属性已经不陌生了，其中界面通过z-index控制显示的优先级。

9.6 字母列表导航条

我们经常在手机 APP 中看到右侧带字母定位的列表，这种列表一般出现在手机联系人界面，效果如图 9-7 所示。

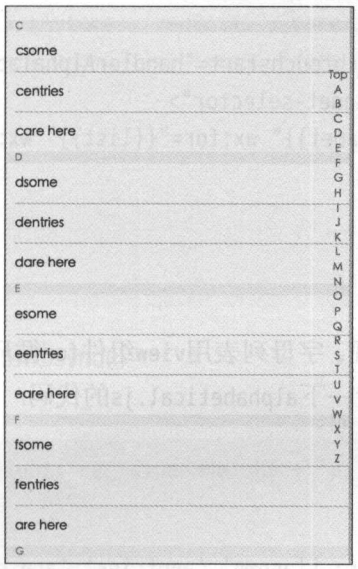

图 9-7 字母列表导航条

可以在微信小程序中实现图 9-7 所示的效果。创建alphabetical.js、alphabetical.wxml、alphabetical.wxss。先来看看alphabetical.wxml相关的代码：

```
<scroll-view scroll-y style="height: {{windowHeight}}" scroll-into-view="{{alpha}}"
    >
  <view class="alphabet">
    <view class="alphabet-list">
      <view wx:for="{{list}}" wx:key="unique" id="{{item.alphabet}}" class="section
          -item" wx:if="{{index!=0}}">
        <view class="section-item-header">
          {{item.alphabet}}
        </view>
```

```
        <view wx:for="{{item.datas}}" wx:key="unique" wx:for-item="cell" wx:for-
            index="cellIndex" class="section-item-cells">
          <view class="section-item-cell {{cellIndex != (item.datas.length-1) ? '
              border-bottom':''}}">
            <text>{{cell}}</text>
          </view>
        </view>
      </view>
    </view>
  </view>
</scroll-view>
<view data-id="selector" catchtouchstart="handlerAlphaTap" catchtouchmove="
    handlerMove" class="alphanet-selector">
  <view data-ap="{{item.alphabet}}" wx:for="{{list}}" wx:key="unique" class="
      selector-one">
    {{item.alphabet}}
  </view>
</view>
```

列表使用scroll-view实现，字母列表用view组件for循环实现。里面用到一些数据需要alphabetical.js提供。来看一下alphabetical.js的代码:

```
Page({
  data:{
    list: [
      {alphabet: 'Top', datas: ['asome','aentries','are here']},
      {alphabet: 'A', datas: ['asome','aentries','are here']},
      {alphabet: 'B', datas: ['bbsome','bebntries','bare here']},
      {alphabet: 'C', datas:  ['csome','centries','care here']},
      {alphabet: 'D', datas:  ['dsome','dentries','dare here']},
      {alphabet: 'E', datas:  ['esome','eentries','eare here']},
      {alphabet: 'F', datas:  ['fsome','fentries','are here']},
      {alphabet: 'G', datas:  ['gsome','gentries','gare here']},
      {alphabet: 'H', datas:  ['hsome','hentries','hare here']},
      {alphabet: 'I', datas:  ['isome','ientries','iare here']},
      {alphabet: 'J', datas:  ['jsome','jentries','jare here']},
      {alphabet: 'K', datas:  ['ksome','kentries','kare here']},
      {alphabet: 'L', datas:  ['lsome','lentries','lare here']},
      {alphabet: 'M', datas:  ['msome','mentries','mare here']},
      {alphabet: 'N', datas:  ['nsome','nentries','nare here']},
```

```
    {alphabet: 'O', datas:  ['osome','oentries','oare here']},
    {alphabet: 'P', datas:  ['psome','pentries','pare here']},
    {alphabet: 'Q', datas:  ['qsome','qentries','qare here']},
    {alphabet: 'R', datas:  ['rsome','rentries','rare here']},
    {alphabet: 'S', datas:  ['some','sentries','sare here']},
    {alphabet: 'T', datas:  ['tsome','tentries','tare here']},
    {alphabet: 'U', datas:  ['usome','uentries','uare here']},
    {alphabet: 'V', datas:  ['vsome','ventries','vare here']},
    {alphabet: 'W', datas:  ['wsome','wentries','ware here']},
    {alphabet: 'X', datas:  ['xsome','xentries','xare here']},
    {alphabet: 'Y', datas:  ['ysome','yentries','yare here']},
    {alphabet: 'Z', datas:  ['zsome','zentries','zare here']},
  ],
  alpha: '',
  windowHeight: ''
},
onLoad(options){
  try {
    //获取系统信息
    var res = wx.getSystemInfoSync()
    this.apHeight = 16;
    this.offsetTop = 80;
    this.setData({windowHeight: res.windowHeight + 'px'})
  } catch (e) {

  }
},
handlerAlphaTap(e) {
  //因为返回的是一个对象,我们定义一个对象接收
  let {ap} = e.target.dataset;
  //{ap}是对象,ap就是对象中ap属性的值
  this.setData({alpha: ap});
},
handlerMove(e) {
  let {list} = this.data;
  let moveY = e.touches[0].clientY;
  let rY = moveY - this.offsetTop;
  if(rY >= 0) {
    let index = Math.ceil((rY - this.apHeight)/ this.apHeight);
```

```
      if(0 <= index < list.length) {
        let nonwAp = list[index];
        nonwAp && this.setData({alpha: nonwAp.alphabet});
      }
    }
  }
})
```

逻辑层主要通过设置data中的数据传入到了布局中，实现了布局中绑定的事件。最后还需要看一下样式文件alphabetical.wxss

```
.alphabet-list .section-item .section-item-header {
  display: flex;
  align-items: center;
  font-size: 22rpx;
  color: #a8a8a8;
  background-color: #f2f4f5;
  padding: 4rpx 20rpx;
}

.alphabet-list .section-item .section-item-cells {
  padding-left: 20rpx;
}

.alphabet-list .section-item .section-item-cells .section-item-cell {
  font-size: 30rpx;
  line-height: 1.0;
  color: #333;
  padding: 29rpx 0;
}

.border-bottom {
  border-bottom: 1rpx solid #dbdbdb;
}

.alphanet-selector {
  position: absolute;
  top: 160rpx;
  right: 0;
  height: 864rpx;
```

```
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  z-index: 100;
  box-sizing: border-box;
}

.alphanet-selector .selector-one {
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 24rpx;
  color: #43c1f4;
  padding: 6rpx 6rpx;
  height: 24rpx;
}
```

9.7 日历

微信小程序本身没有提供日历组件，如果需要用到日历可以自己写一个，如图 9-8 所示。

图 9-8 日历

创建calendar.js，calendar.wxml，calendar.wxss。布局文件calendar.wxml代码如下：

```
<!--pages/record/record.wxml-->
<view class="selected_date">
  <view bindtap="preMonth">
    <image class="select_icon" src="/image/left.png"></image>
```

```
  </view>
  <view class="text_center">{{curYear+'-'+curMonth}}</view>
  <view style="text-align:right;" bindtap="nextMonth">
    <image class="select_icon" src="/image/right.png"></image>
  </view>
</view>
<view class="calendar_panel">
  <view class="calendar_box">
    <view class="weekday_label">日</view>
    <view class="weekday_label">一</view>
    <view class="weekday_label">二</view>
    <view class="weekday_label">三</view>
    <view class="weekday_label">四</view>
    <view class="weekday_label">五</view>
    <view class="weekday_label">六</view>
  </view>
  <view class="calendar_box" wx:for="{{dateList}}" wx:for-item="week" style="{{
      index==0?'justify-content: flex-end;':''}}">
    <view class="weekday_label" wx:for="{{week}}">
      <text class="{{item.value==selectedDate?'active_date':''}}" bindtap="
          selectDate" data-date="{{item}}">{{item.date}}</text>
    </view>
  </view>
</view>
<view class="show_box">选中日期：{{selectedDate}} {{selectedWeek}}</view>
```

可以看到布局只通过<view>组件就能实现，需要calendar.js提供数据来源。代码如下：

```
// pages/record/record.js
Page({
  data: {
    selectedDate: '',//选中的几月几号
    selectedWeek: '',//选中的星期几
    curYear: 2017,//当前年份
    curMonth: 0,//当前月份
    daysCountArr: [// 保存各个月份的长度，平年
      31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
    ],
    weekArr: ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期
```

```
      六'],
  dateList: []
},
onLoad: function (options) {
  // 页面初始化 options为页面跳转所带来的参数
},
onReady: function () {
  // 页面渲染完成
},

onShow: function () {
  var today = new Date();//当前时间
  var y = today.getFullYear();//年
  var mon = today.getMonth() + 1;//月
  var d = today.getDate();//日
  var i = today.getDay();//星期
  this.setData({
    curYear: y,
    curMonth: mon,
    selectedDate: y + '-' + mon + '-' + d,
    selectedWeek: this.data.weekArr[i]
  });

  this.getDateList(y, mon - 1);
},
getDateList: function (y, mon) {
  var vm = this;
  //如果是否闰年，则2月是29日
  var daysCountArr = this.data.daysCountArr;
  if (y % 4 == 0 && y % 100 != 0) {
    this.data.daysCountArr[1] = 29;
    this.setData({
      daysCountArr: daysCountArr
    });
  }
  //第几个月；下标从0开始实际月份还要再+1
  var dateList = [];
  // console.log('本月', vm.data.daysCountArr[mon], '天');
  dateList[0] = [];
```

```
    var weekIndex = 0;//第几个星期
    for (var i = 0; i < vm.data.daysCountArr[mon]; i++) {
      var week = new Date(Date.UTC(year, month - 1, i + 1)).getDay();
      dateList[weekIndex].push({
        value: y + '-' + (mon + 1) + '-' + (i + 1),
        date: i + 1,
        week: week
      });
      if (week == 6) {
        weekIndex++;
        dateList[weekIndex] = [];
      }
    }
    // console.log('本月日期', dateList);
    vm.setData({
      dateList: dateList
    });
  },
  selectDate: function (e) {
    var vm = this;
    // console.log('选中', e.currentTarget.dataset.date.value);
    vm.setData({
      selectedDate: e.currentTarget.dataset.date.value,
      selectedWeek: vm.data.weekArr[e.currentTarget.dataset.date.week]
    });
  },
  preMonth: function () {
    // 上个月
    var vm = this;
    var curYear = vm.data.curYear;
    var curMonth = vm.data.curMonth;
    curYear = curMonth - 1 ? curYear : curYear - 1;
    curMonth = curMonth - 1 ? curMonth - 1 : 12;
    // console.log('上个月', curYear, curMonth);
    vm.setData({
      curYear: curYear,
      curMonth: curMonth
    });
```

```
    vm.getDateList(curYear, curMonth - 1);
  },
  nextMonth: function () {
    // 下个月
    var vm = this;
    var curYear = vm.data.curYear;
    var curMonth = vm.data.curMonth;
    curYear = curMonth + 1 == 13 ? curYear + 1 : curYear;
    curMonth = curMonth + 1 == 13 ? 1 : curMonth + 1;
    // console.log('下个月', curYear, curMonth);
    vm.setData({
      curYear: curYear,
      curMonth: curMonth
    });

    vm.getDateList(curYear, curMonth - 1);
  }
})
```

逻辑层设置了上一个月，下一个月和每一天的点击事件。最后看一下样式文件的calendar.wxss代码：

```
/* pages/record/record.wxss */
.selected_date {
  padding: 20rpx;
  overflow: hidden;
}
/**表示具有class=".selected_date"属性组件内的所有view组件*/
.selected_date>view {
  width: 33.3%;
  float: left;
}

.calendar_panel {
  width: 100%;
  height: 600rpx;
}

.calendar_box {
  width: 100%;
```

```
  background: #fff;
  overflow: hidden;
  border-bottom: 1rpx solid #ececec;
  display: flex;
}

.weekday_label {
  width: 14.28%;
  float: left;
  text-align: center;
  font-size: 26rpx;
  padding: 10rpx 0;
}

.weekday_label>text {
  width:50rpx;
  height: 50rpx;
  line-height: 50rpx;
  display: inline-block;
}

.select_icon {
  width: 30rpx;
  height: 30rpx;
}

.active_date {
  background: #148447;
  color: #fff;
  border-radius: 50%;
}

.show_box {
  padding: 20rpx;
  font-size: 26rpx;
}
```

实际在微信小程序中也支持 .selected_date>view 这种选择器，其指向具有class=".selected_date"属性组件内的所有 view 组件。

9.8 本章小结

本章介绍了几种常用的组合组件，方便开发者在实际开发中使用。本书写到这里已经接近尾声了，第 10 章介绍了一个完整的案例，方便读者对之前的知识进行总结和应用。

10 综合案例——音乐播放小程序

为了更好地掌握前面所学的内容，本章介绍如何编写一个音乐播类放的微信小程序。本章的案例会从零开始，一步步地介绍如何开发一个功能完整的微信小程序。在案例中会尽可能多地涵盖前面学的内容，以便读者在学习完整程序开发流程的同时，复习前面的内容。

为了保护版权，本章所用的网络数据为开源数据（数据版权归数据提供方所有），使用无 APPID 的模式在模拟器中开发，并且所使用的地址为 http 格式，所以只能保证在模拟器上运行正确。这个案例只为帮助读者更好地学习微信小程序，并不具有实际应用价值。在实际开发，尤其是开发准备发布的微信小程序时，请务必按照微信小程序开发的规定，使用 https 作为网络请求的协议，申请 AppID 并对需要访问的地址授权。

更多有关发布的规定请参照第 11 章的内容。

10.1 项目需求

在开始编码音乐播放类的微信小程序之前，需要先确定微信小程序的基本功能。类似我们经常使用的音乐播放 APP，我们希望微信小程序具有以下功能：

(1) 首页为推荐页，主要以电台信息和歌曲推荐信息为主。

(2) 排行榜功能，包括多类不同的排行榜，例如中文、欧美、日韩这种分类。

(3) 搜索功能。

(4) 音乐播放页面。

根据这些需求，需要构建的页面有：

(1) 包涵标签栏的首页，通过标签栏在推荐页、排行榜页以及搜索页之间切换。

(2) 点击排行榜分类跳转的列表页。

(3) 点击专辑跳转的列表页。

(4) 歌曲播放页。

明确这些需求后，下面开始构建项目吧。

10.2 项目结构

按照需求，项目结构大致构建成如图 10-1 所示。

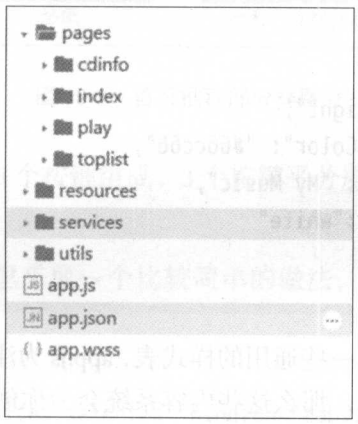

图 10-1 项目结构图

我们在新目录下创建小程序，系统会建立一个 quickstart 的 demo 文件，它包含了 app.js、app.json 和 app.wxss 的文件，可以直接在这个基础上更改成我们需要的形式。在这里要注意，直接生成的文件里会包涵 log 这个页面，我们在练习时可以忽略这个文件夹。如果一定要删除这个文件夹，那么记得要同时删除 app.json 里 pages 下的“pages/logs/logs”这行代码。

pages 文件夹下是我们需要的 5 个页面，resources 文件夹用来存放一些图片资源，services 文件夹下是我们编写网络请求所需的文件，utils 文件夹用来存放一些工具类文件。

当然，在所有页面文件夹下还需要创建对应的 4 个文件，例如首页 index 文件夹下需要创建 index.js，index.json，index.wxml 以及 index.wxss。

全部创建完成后就可以开始编码了。

10.3 配置项目文件

在开始写页面之前，先要在 app.json 里配置页面信息，以及更改程序顶部标题栏样式。

```
{
  "pages":[
    "pages/index/index",
    "pages/toplist/toplist",
    "pages/cdinfo/cdinfo",
    "pages/play/play",
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#66cc66",
    "navigationBarTitleText": "My Music",
    "navigationBarTextStyle":"white"
  }
}
```

在 app.wxss 里，可以设定一些通用的样式表，app.js 为注册 App 所需要的代码 App()。如果使用 quickstart 创建项目，那么这些内容系统会为你创建好，一开始不需要对这两个文件进行修改，所以保留原有样式就可以了。

如果需要编写功能相对复杂的微信小程序，那么推荐在开始编写页面之前，根据实际需要将频繁使用或大量页面通用的页面样式写在 app.wxss 里，这样既可以节省时间，也便于保证多个页面样式的一致性。

在 app.js 下，除必不可少的 APP 注册信息外，另一个功能是保存整个 APP 都会用到的全局变量，例如 quickstart 里 globalData 这个变量。

10.4 首页 index

开始构建小程序首页。

第一步，需要创建顶部的导航栏，效果应该类似图 10-2。

图 10-2 首页推荐部分样图

可以看到这个导航栏由 3 个按键组成，3 个按键平分屏幕宽度，文字居中显示，在选中后下方有绿色边框。

为了实现这个效果，这里采取一个比较简单的做法，为每个标签的每个状态（选中/未选中）创建一个 view。

```
<view class="tab">
  <view class="tab-item tab-item-selected" bindtap="tabItemTap" data-view="1" wx:if
      ="{{currentView==1}}">推荐</view>
  <view class="tab-item" data-view="1" bindtap="tabItemTap" wx:if="{{currentView
      !=1}}">推荐</view>
  <view class="tab-item tab-item-selected" bindtap="tabItemTap" data-view="2" wx:if
      ="{{currentView==2}}">排行</view>
  <view class="tab-item" data-view="2" bindtap="tabItemTap" wx:if="{{currentView
      !=2}}">排行</view>
  <view class="tab-item tab-item-selected" bindtap="tabItemTap" data-view="3" wx:if
      ="{{currentView==3}}">检索</view>
  <view class="tab-item" data-view="3" bindtap="tabItemTap" wx:if="{{currentView
      !=3}}">检索</view>
</view>
```

```
tab-item {
    float:left;
    width:33.333333%;
    height:86rpx;
    font-size:32rpx;
    text-align:center;
}
.tab-item-selected {
    color:#31c27c;
    border-bottom:4rpx solid #31c27c;
}
```

所有 6 个 view 都享有 tab-item 这个 class 的属性，在这里定义了组件的宽度为 1/3，字体居中显示以及字号。3 个布局拥有 tab-item-selected 属性，这个属性为这个 view 添加了底部的绿色边框。currentView 为控制这一组件的值，当 currentView=1 时，根据 wx:if 属性，只有带下边框的“推荐”view 与不带下边框的“排行”，“检索”会被渲染，也就实现了我们想要的结果。

在 index.js 里编写 view 的点击事件 tabItemTap，这个名字跟我们在写 wxml 时 bindtap 一致。

```
//获取应用实例
var app = getApp()
Page({
    data: {
        currentView: 1,
    },
    onLoad: function () {
        var that = this;
    },
    tabItemTap: function (e) {
        var _dataSet = e.currentTarget.dataset;
        this.setData({
            currentView: _dataSet.view
        });
    },
})
```

每次点击后，获取点击 view 的 data-view 值，然后将这个值赋值给 currentView，从而更新界面。

10.4.1 推荐页

完成标题栏后开始编写推荐页，即 mainView=1 时所要显示的页面。

根据图 10-2，推荐页由上方的轮播组件（banner）以及下方的电台列表由两部分构成。为了完成这个页面，下面先来看看网络请求返回的数据格式。

这里使用开源数据：

http://c.y.qq.com/musichall/fcgi-bin/fcg_yqqhomepagerecommend.fcg

参照第 8 章中的内容，在 services 文件夹下创建 music.js 文件，在里面开始编写网络请求代码：

```
// 获取首页的音乐数据
function getRecommendMusic(callback){
    //请求所需数据
    var data = {
            g_tk: 5381,
            uin: 0,
            format: 'json',
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            _: Date.now()
        };
        wx.request({
            //地址
            url: 'http://c.y.qq.com/musichall/fcgi-bin/fcg_yqqhomepagerecommend.fcg
                ',
            //数据
            data: data,
            //表示返回类型为JSON
            header: {
                'Content-Type': 'application/json'
            },
            success: function (res) {
                if (res.statusCode == 200) {
                    callback(res.data)
                } else {
```

```
            }

        }
    });
}

module.exports = {
  getRecommendMusic:getRecommendMusic
}
```

通过这个请求，返回 JSON 格式的数据样式为：

```
{
    "code": 0,
    "data": {
        "slider": [
            {
                "linkUrl": "http://share.y.qq.com/l?g=2766&id=1842365&g_f=
                    shoujijiaodian",
                "picUrl": "http://y.gtimg.cn/music/photo_new/
                    T003R720x288M000002QUG1D0iCyQM.jpg",
                "id": 8642
            },
            {
                "linkUrl": "http://y.qq.com/live/zhibo/0214liwen.html",
                "picUrl": "http://y.gtimg.cn/music/photo_new/
                    T003R720x288M000003KFpsf1mPzlY.jpg",
                "id": 8645
            },
            {
                "linkUrl": "http://v.qq.com/live/p/topic/22876/preview.html",
                "picUrl": "http://y.gtimg.cn/music/photo_new/
                    T003R720x288M000000ZZAWw1KsyoJ.jpg",
                "id": 8653
            },
            {
                "linkUrl": "http://y.qq.com/m/act/singer/index.html?ADTAG=
                    shoujijiao",
                "picUrl": "http://y.gtimg.cn/music/photo_new/
```

```
                T003R720x288M000001MG8W3200tuD.jpg",
            "id": 8609
        },
        {
            "linkUrl": "http://y.qq.com/w/album.html?albummid=0035hOHV0uUWA9",
            "picUrl": "http://y.gtimg.cn/music/photo_new/
                T003R720x288M000000cfVE83KCkmz.jpg",
            "id": 8607
        }
    ],
    "radioList": [
        {
            "picUrl": "http://y.gtimg.cn/music/photo/radio/track_radio_199_13_1
                .jpg",
            "Ftitle": "热歌",
            "radioid": 199
        },
        {
            "picUrl": "http://y.gtimg.cn/music/photo/radio/track_radio_307_13_1
                .jpg",
            "Ftitle": "一人一首招牌歌",
            "radioid": 307
        }
    ],
    "songList": []
  }
}
```

这里 code 为我们请求是否成功的标示，当它等于 0 时，表示请求成功。data 里就是我们需要的数据，里面包含 3 个部分，我们需要使用的为前两个，即 slider 部分——为我们的轮播组件提供数据，以及 radioList 部分——为电台列表提供数据。这两部分会分别以数组格式保存，通过名称来获取相应数据。

有了数据之后，下面开始编写显示数据的组件：

```
<view hidden="{{currentView != 1}}">
  <swiper indicator-dots="{{indicatorDots}}" autoplay="{{autoplay}}" interval="{{
      interval}}" duration="{{duration}}">
    <block wx:for="{{slider}}" wx:key="unique">
      <swiper-item data-id="{{item.id}}">
```

```
        <image src="{{item.picUrl}}" style="height:100%" class="slide-image" />
      </swiper-item>
    </block>
  </swiper>
  <view class="channel">
    <text class="channel-title">电台</text>
    <view class="radio-list">
      <block wx:for="{{radioList}}" wx:key="unique">
        <view class="radio-item" data-id="{{item.radioid}}" data-ftitle="{{item.
            Ftitle}}" bindtap="radioTap">
          <image class="radio-img" mode="aspectFit" style="height:167.5px;" src="{{
              item.picUrl}}" />
          <text class="radio-text">{{item.Ftitle}}</text>
        </view>
      </block>
    </view>
  </view>
</view>
```

最外层使用 view 组件包裹，当 currentView!=1 时隐藏。

轮播组件使用 swiper 组件来设置，包括设置是否显示指示点，是否自动播放，切换间隔以及滑动时间 4 个属性。每个 swiper-item 为图片，使用 item.picUrl 从 slider 里获取数据。

电台部分使用列表格式，数据保存在 radioList 内。每个 item 包涵两个部分：图片和标题，以 item.picUrl 和 item.Ftitle 保存，此外还要保存每个 item 的 ID（item.radioid) 用于页面跳转。点击的响应事件定义为 radioTap。

至此我们需要的数据有：indicatorDots、autoplay、interval、duration、slider、radioList。下面把这些加入到 index.js 中的 data 里。

```
//引用网络请求文件
var MusicService = require('../../services/music');

//获取应用实例
var app = getApp()
Page({
    data: {
        indicatorDots: true,
        autoplay: true,
```

```
        interval: 5000,
        duration: 1000,
        radioList: [],
        slider: [],
        mainView: 1,
    },
    onLoad: function () {
        var that = this;
        MusicService.getRecommendMusic(that.initPageData);
    },
})
```

data写好后，在onLoad里调用写好的网络请求函数，传入的参数（that.initPageData），即当请求成功后需要执行的函数，在这个函数里我们为数组radioList和slider赋值。

```
initPageData: function (data) {
        var self = this;
        //请求成功再赋值，需要判断code是否为0
        if (data.code == 0) {
            self.setData({
                slider: data.data.slider,
                radioList: data.data.radioList,
            })
        }
    },
```

到此为止，我们的页面应该可以显示数据了，最后一步我们要给写好的view添加点击事件radioTap，让用户点击后跳转到play（音乐播放）页面。

```
radioTap: function (e) {
        var dataSet = e.currentTarget.dataset;
        ...
    },
```

在跳转的时候，需要通过已经获得的radioid向网络请求数据，返回歌曲列表，并且在播放页面加载这个列表，这一部分就留到音乐播放页再完成吧。

目前用到的布局文件（index.wxss）为：

```
.tab {
    height:88rpx;
    line-height:84rpx;
```

```
    background:#fff;
}
.tab-item {
    float:left;
    width:33.333333%;
    height:86rpx;
    font-size:32rpx;
    text-align:center;
}
.tab-item-selected {
    color:#31c27c;
    border-bottom:4rpx solid #31c27c;
}
.slide-image {
    width:100%;
    height:100%;
}
.channel {
    margin-top:20rpx;
}
.channel-title {
    font-size:44rpx;
    padding:0 20rpx;
    box-sizing:border-box;
}
.radio-list {
    display:inline-block;
    width:100%;
    height:100%;
}
.radio-item {
    width:50%;
    height:100%;
    padding:20rpx;
    float:left;
    text-align:center;
    box-sizing:border-box;
}
.radio-img-box {
```

```
    width:100%;
    height:50%;
}
.radio-img {
    width:100%;
    height:100%;
    vertical-align:middle;
}
.radio-text {
    margin-top:20rpx;
    font-size:36rpx;
    overflow:hidden;
    text-overflow:ellipsis;
    white-space:nowrap;
}
```

10.4.2 排行页

获取排行榜的网络请求地址：

http://c.y.qq.com/v8/fcg-bin/fcg_myqq_toplist.fcg

同 10.4.1 节所做的一样，先在 services/music.js 里添加网络请求函数：

```
function getTopMusic(callback){
    var data = {
            format: 'json',
            g_tk: 5381,
            uin: 0,
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            _: Date.now()
        };
        wx.request({
            url: 'http://c.y.qq.com/v8/fcg-bin/fcg_myqq_toplist.fcg',
            data: data,
            header: {
                'Content-Type': 'application/json'
```

```
            },
            success: function (res) {
                if (res.statusCode == 200) {
                    callback(res.data)
                } else {

                }
            }
        });
}

module.exports = {
  getRecommendMusic:getRecommendMusic
  getTopMusic:getTopMusic
}
```

这里返回的JSON格式数据为：

```
{
    "code": 0,
    "subcode": 0,
    "message": "",
    "default": 0,
    "data": {
        "topList": [
            {
                "id": 4,
                "listenCount": 20000000,
                "picUrl": "http://y.gtimg.cn/music/common/upload/
                    iphone_order_channel/toplist_4_300_200669704.jpg",
                "songList": [
                    {
                        "singername": "赵雷",
                        "songname": "理想 (Live)"
                    },
                    {
                        "singername": "薛之谦/欧阳娜娜",
                        "songname": "小幸运 (Live)"
                    },
                    {
```

```
                    "singername": "迪玛希Dimash",
                    "songname": "秋意浓 (Live)"
                }
            ],
            "topTitle": "巅峰榜 · 流行指数",
            "type": 0
        },
        {
            "id": 26,
            "listenCount": 19900000,
            "picUrl": "http://y.gtimg.cn/music/common/upload/
                iphone_order_channel/toplist_26_300_109191643.jpg",
            "songList": [
                {
                    "singername": "李玉刚",
                    "songname": "刚好遇见你"
                },
                {
                    "singername": "周杰伦",
                    "songname": "告白气球"
                },
                {
                    "singername": "张杰",
                    "songname": "三生三世"
                }
            ],
            "topTitle": "巅峰榜 · 热歌",
            "type": 0
        },
        ...
    ]
}
}
```

可以看到这里显示了两类排行榜：“巅峰榜 · 流行指数”与“巅峰榜 · 热歌”。因篇幅原因，这里省去了其他 12 类，所以实际返回的排行榜类别为 14 类，每一类包涵标题（"topTitle"），该类的图标图片地址（"picUrl"），以及前三位的歌曲列表（"songList"）。因此，最后要完成的页面应该为图 10-3 所示。

图 10-3 首页排行榜样图

与 10.4.1 节内容同理，这里新增 topList 数组，调用网络请求，使用回调函数为 topList 赋值。

```
//引用网络请求文件
var MusicService = require('../../services/music');

//获取应用实例
var app = getApp()
Page({
    data: {
        indicatorDots: true,
        autoplay: true,
        interval: 5000,
        duration: 1000,
        radioList: [],
        slider: [],
        mainView: 1,
        topList:[]
    },
    onLoad: function () {
```

```
        var that = this;
        MusicService.getRecommendMusic(that.initPageData);
        MusicService.getTopMusic(that.initTopList);
    },

    ...

    initTopList: function (data) {
        var self = this;
        if (data.code == 0) {
            self.setData({
                topList: data.data.topList
            })
        }
    },

    ...

})
```

排行页主要由列表组成，所以使用 wx:for 为 topList 每一项创建 view，绑定每一项的 id 和点击事件 topListTap。

```
<!-- 排行页 -->
<view class="top-view" hidden="{{currentView != 2}}">
  <view class="top-item" wx:for="{{topList}}" wx:key="unique" data-id="{{item.id}}"
      bindtap="topListTap">
    ...
  </view>
</view>
```

列表的每一项由图片（数据源为 picUrl）以及前 3 名歌曲列表（数据源为 songList）组成。我们把这一部分加入到省略号位置。

```
<!-- 排行页 -->
<view class="top-view" hidden="{{currentView != 2}}">
  <view class="top-item" wx:for="{{topList}}" wx:key="unique" data-id="{{item.id}}"
      bindtap="topListTap">
    <image class="top-item-img" mode="aspectFit" src="{{item.picUrl}}" />
    <view class="top-item-info">
      ...
```

```
    </view>
  </view>
</view>
```

图片定义了属性 aspectFit，即在保持纵横比的前提下缩放图片，使图片在容器内都显示出来。

最后添加歌曲列表，每一项由文字（歌曲名-歌手）以及表示排名的图片组成。这个图片为本地图片，保存在 resources/images 下，名称为 1.png，2.png，3.png。所以这部分最终的代码为：

```
<!-- 排行页 -->
<view class="top-view" hidden="{{currentView != 2}}">
  <view class="top-item" wx:for="{{topList}}" wx:key="unique" data-id="{{item.id}}"
      bindtap="topListTap">
    <image class="top-item-img" mode="aspectFit" src="{{item.picUrl}}" />
    <view class="top-item-info">
      <view class="top-item-list" wx:for="{{item.songList}}" wx:key="unique">
        <image class="top-icon" src="../../resources/images/{{index+1}}.png" />
        <text class="top-item-text">{{item.songname}}-{{item.singername}}</text>
      </view>
    </view>
  </view>
</view>
```

需要的格式文件代码为：

```
.top-view {
    background:#f7f7f7;
    padding:20rpx;
}
.top-item {
    display:-webkit-box;
    height:200rpx;
    margin-bottom:20rpx;
    background:#fff;
    overflow: hidden;
}
.top-item-img {
    display:block;
    position:relative;
```

```
    width:200rpx;
    height:200rpx;
}
.top-item-info {
    margin:0 10rpx 0 20rpx;
    flex-direction:column;
}
.top-item-list {
    white-space:nowrap;
}
.top-icon {
    margin-top:16rpx;
    width:40rpx;
    height:40rpx;
}
.top-item-text {
    margin-bottom: 10rpx;
    font-size:40rpx;
}
```

页面完成后，在 index.js 里添加点击事件的代码：

```
topListTap: function (e) {
        wx.navigateTo({
            url: '../toplist/toplist'
        })
    },
```

这样在点击之后就进入了列表页面，但这样还不能完成我们的要求，因为这样列表页完全不知道我们点击了哪一类。为了让列表页获取这个信息，需要把类的 id 传过去，这就需要在 app.js 里添加一个全局变量。

```
//app.js
App({
  onLaunch: function () {
    //调用API从本地缓存中获取数据
    var logs = wx.getStorageSync('logs') || []
    logs.unshift(Date.now())
    wx.setStorageSync('logs', logs)
  },
  getUserInfo:function(cb){
```

```
    var that = this
    if(this.globalData.userInfo){
      typeof cb == "function" && cb(this.globalData.userInfo)
    }else{
      //调用登录接口
      wx.login({
        success: function () {
          wx.getUserInfo({
            success: function (res) {
              that.globalData.userInfo = res.userInfo
              typeof cb == "function" && cb(that.globalData.userInfo)
            }
          })
        }
      })
    }
  },
  //这里是我们添加的代码！！！
  setGlobalData: function (obj) {
    for(var n in obj) {
      this.globalData[n] = obj[n];
    }
  },
  globalData:{
    userInfo:null
  }
})
```

这里定义了一个方法，让我们可以向 globalData 里添加数据，之后只需要在点击事件里调用这个方法就可以了：

```
topListTap: function (e) {
        var _dataSet = e.currentTarget.dataset;   //获取点击的view的数据
        app.setGlobalData({                       //将数据里的id传给globaldata
            topListId: _dataSet.id
        });
        wx.navigateTo({
            url: '../toplist/toplist'
        })
    },
```

10.4.3 检索页（上）

首页的最后一部分为检索页，也就是让用户通过关键字搜索歌曲或专辑。说到搜索页，必不可少的就是关键字的输入框，如图 10-4 所示。

图 10-4 搜索框样图

参照如图 10-4 所示的样式，布局文件应该为：

```
<view class="search-bar">
    <view class="search-input-warp">
      <form bindsubmit="searchSubmit">
        <label class="search-input-icon"></label>
        <input type="text" class="search-input" bindfocus="bindFocus" name="search"
             value="{{searchKey}}" bindinput="bindKeyInput" placeholder="搜索歌
            曲、歌单、专辑" placeholder-class="search-input-placeholder"/>
        <view class="search-cancel" bindtap="searchOk">确定</view>
      </form>
    </view>
  </view>
```

格式文件为：

```
.search-bar {
    background:#f7f7f7;
    padding:20rpx;
}
.search-input-warp {
    position:relative;
    padding:16rpx 24rpx 16rpx 70rpx;
    background:#fff;
}
.search-input-icon {
    content:" ";
    position:absolute;
    top:18rpx;
    left:20rpx;
    display:inline-block;
    vertical-align:middle;
```

```
    width:40rpx;
    height:40rpx;
    background-image:url("../../resources/images/search.png");
    background-repeat:no-repeat;
    background-size:40rpx;
}
.search-input {
    font-size:28rpx;
    line-height:40rpx;
}
.search-input-placeholder {
    color:#ddd;
    font-size:28rpx;
    line-height:40rpx;
}
.search-cancel {
    position:absolute;
    right:0;
    top:0;
    display:inline-block;
    font-size:32rpx;
    height:90rpx;
    width:140rpx;
    text-align:center;
    line-height: 90rpx;
    z-index:50;
    background:#f7f7f7;
}
```

因为要提交关键字，所以使用表单组件，内部由表示搜索的放大镜图标、代表输入完成的“确定”按钮以及输入框组成。

除输入框外，此页还需要显示的内容还有搜索结果列表、搜索历史记录以及热门关键字。这些内容应该显示在输入框下方，并按一定逻辑切换。它们之间的逻辑关系为：

(1) 用户进入检索页，显示的内容应该是输入框与热门关键字。

(2) 开始输入后（即输入框获得焦点），应该显示历史记录。

(3) 点击“确定”按钮后，显示结果列表。

(4) 清空输入框内容后，返回关键字显示。

可以看出，对输入框的操作决定了当前页面显示的内容，所以需要监听输入框的获取焦点事件（bindfocus），确定点击事件（bindtap）以及输入事件（bindinput），还需要记录输入的关键字 searchKey。

输入框完成后下面先写热门关键字，这些关键字也是来源于网络，所以还需要请求网络。在 services/music 里添加方法：

```
function getHotSearchKey(callback){
    var data = {
            g_tk: 5381,
            uin: 0,
            format: 'json',
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            _: Date.now()
        };
        wx.request({
            url: 'http://c.y.qq.com/splcloud/fcgi-bin/gethotkey.fcg',
            data: data,
            header: {
                'Content-Type': 'application/json'
            },
            success: function (res) {
                if (res.statusCode == 200) {
                    callback(res.data)
                } else {

                }
            }
        });
}
...
module.exports = {
  ...
  getHotSearchKey:getHotSearchKey
}
```

返回 JSON 格式结果为：

```
{
    "code": 0,
    "data": {
        "hotkey": [
            {
                "k": "三生三世十里桃花 ",
                "n": 90558
            },
            {
                "k": "DJ ",
                "n": 67590
            },
            {
                "k": "薛之谦 ",
                "n": 60425
            },
            {
                "k": "凉凉 ",
                "n": 37056
            },
            {
                "k": "那片星空那片海 ",
                "n": 29170
            },
            {
                "k": "理想 ",
                "n": 28695
            },
            ...
        ],
        "special_key": "歌手",
        "special_url": "http://y.qq.com/m/act/singer/index.html?ADTAG=search"
    },
    "subcode": 0
}
```

这里返回的 hotkey 就是我们需要的关键字，除此之外，我们还看到下面有“special_key”这一部分，这相当于广告部分，为服务器推荐的搜索关键字。所以这里以标签

的形式显示关键字，对于广告那部分关键字，这里以红色字体显示，并且放在最前方，最后完成的样式应该如图 10-5 所示。

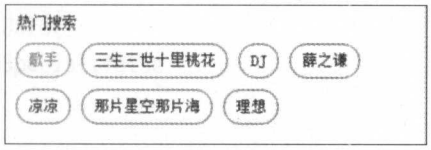

图 10-5 热门搜索标签样图

实现这一布局的代码为：

```
<view class="quick-search" wx:if="{{showSearchPanel == 1}}">
    <view class="quick-search-title">
      <text>热门搜索</text>
    </view>
    <view class="quick-search-bd">
      <text class="quick-search-item-red" data-url="{{special.url}}" data-key="{{
          special.key}}" bindtap="hotKeysTap" wx:if="{{showSpecial}}">{{special.key
          }}</text>
      <text class="quick-search-item" wx:for="{{hotkeys}}" wx:key="unique" data-n="
          {{item.n}}" bindtap="hotKeysTap" data-key="{{item.k}}">{{item.k}}</text>
    </view>
  </view>
```

这里面 showSearchPanel 的值为我们控制这个页面切换效果的依据，当其为 1 的时候，显示热门关键字。

绑定的数据有两个：special 和 hotkey。对于 special 部分，为了防止服务器缺失“special_key”这一部分从而导致没有数据显示，这里定义一个数据 showSpecial 来决定是否显示这一部分。

将这 3 个数据以及输入框用到的 searchKey 添加到 data 里。

修改 index.js 文件：

```
//引用网络请求文件
var MusicService = require('../../services/music');

//获取应用实例
var app = getApp()
Page({
```

```
data: {
    indicatorDots: true,
    autoplay: true,
    interval: 5000,
    duration: 1000,
    radioList: [],
    slider: [],
    mainView: 1,
    topList:[],
    hotkeys: [],
    showSpecial: false,
    special: { key: '', url: '' },
    searchKey: '',
},
onLoad: function () {
    var that = this;
    MusicService.getRecommendMusic(that.initPageData);
    MusicService.getTopMusic(that.initTopList);
    MusicService.getHotSearchKeys(that.initSearchHotKeys);
},

...
initSearchHotKeys: function (data) {
    var self = this;
    if (data.code == 0) {
        var special = { key: data.data.special_key, url: data.data.special_url
            };
        var hotkeys = [];
        if (data.data.hotkey && data.data.hotkey.length) {  //当返回数据不为空
            且长度不为0
            for (var i = 0; (i < data.data.hotkey.length && i < 6); i++) {   //
                如果长度大于6只保留前6个
                var item = data.data.hotkey[i];
                hotkeys.push(item);
            }
        }
        if (special != undefined) {     //当返回项有special部分时，
            showSpecial为true
            self.setData({
```

```
                showSpecial: true
            })
        } else {                                    //没有special部分，showSpecial为false
            self.setData({
                showSpecial:false
            })
        }
        self.setData({
            special: special,
            hotkeys: hotkeys
        })
    }
  },
  ...
})
```

数据加载完成后，下面为每个热门关键字 view 添加点击事件。

```
hotKeysTap: function (e) {
      //TODO
  },
```

在这个点击事件里，我们需要做的事情有：(1) 将点击的关键词加入历史记录。(2) 将页面切换到搜索结果列表。在完成搜索结果列表页后再来完善这一部分吧。

最后附上这一部分的格式文件。

```
.quick-search {
    padding:20rpx;
    box-sizing:border-box;
}
.quick-search-title {
    font-size:28rpx;
}
.quick-search-bd {
    position:relative;
    margin-top:20rpx;
}
.quick-search-item {
    font-size:28rpx;
    float:left;
    margin:0 20rpx 20rpx 0;
```

```
    line-height:40rpx;
    padding:10rpx 20rpx;
    border-radius:30rpx;
    color:#000;
    border:2rpx solid rgba(0,0,0,.6);
}
.quick-search-item-red {
    font-size:28rpx;
    float:left;
    margin:0 20rpx 20rpx 0;
    line-height:40rpx;
    padding:10rpx 20rpx;
    border-radius:30rpx;
    color:#fc4524;
    border:2rpx solid #fc4524;
}
```

10.4.4 检索页（中）

接下来完成历史记录部分，完成后应该类似图 10-6。

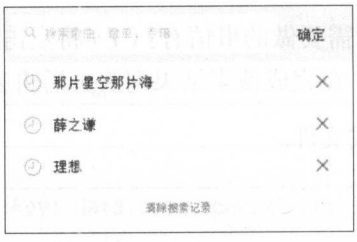

图 10-6 历史记录样图

这部分也是列表样式，每一项由前面的时钟图标、中间的文字以及最后表示删除的“X”组成，最下方为清空整个列表的选项。

```
<view class="search-hi" wx:if="{{showSearchPanel == 2}}">
    <block wx:if="{{historySearchs.length > 0}}">
      <view class="search-hi-item" wx:for="{{historySearchs}}" wx:key="unique">
        <view class="hi-icon"></view>
        <text class="hi-text" data-key="{{item}}" bindtap="historysearchTap">{{item
            }}</text>
        <view class="hi-close" data-index="{{index}}" bindtap="delHistoryItem"></
            view>
```

```
      </view>
      <view class="clear-serach" bindtap="clearHistorySearchs">清除搜索记录</view>
    </block>
  </view>
```

这一部分只有当 showSearchPanel=2 才显示。

首先需要判断是否有历史数据。使用 historySearchs 存储所有历史记录，只有当其长度大于 0 时，才渲染这一部分。

block 里的内容就如我们所要的一致，然后为文字，结尾的“X”标志和最后的“清除搜索记录”添加点击事件。

格式文件为：

```
.search-hi-item {
    height:88rpx;
    line-height:88rpx;
    border-bottom:2rpx solid #ededed;
}
.hi-icon {
    position:absolute;
    left:20rpx;
    display:inline-block;
    width:88rpx;
    height:88rpx;
    background:url('../../resources/images/clock_ic.png') no-repeat center center;
    background-size:40rpx;
}
.hi-text {
    position:absolute;
    left:108rpx;
    right:108rpx;
    display:block;
    white-space:nowrap;
    text-overflow:ellipsis;
    overflow:hidden;
    font-size:32rpx;
}
.hi-close {
    width:88rpx;
```

```
    height:88rpx;
    position:absolute;
    right:20rpx;
    background:url('../../resources/images/cancel.png') no-repeat center center;
}
.clear-serach {
    font-size:24rpx;
    text-align:center;
    color:#47c88a;
    line-height:88rpx;
}
```

background里分别加载了两个本地图片并为其设置了不平铺与居中的属性。

很显然，在我们每次点击“确定”按钮后，输入框内的值除了发送到服务器请求搜索结果，还要添加到historySearchs里。

所以下面先写搜索结果列表，然后一起处理“确定”按钮的点击事件。

首先是网络请求的函数：

```
function getSearchMusic(word,callback){        //word为传入的关键字，callback为回
    调函数
    var data = {
            g_tk: 5381,
            uin: 0,
            format: 'json',
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            w: word,
            zhidaqu: 1,
            catZhida: 1,
            t: 0,
            flag: 1,
            ie: 'utf-8',
            sem: 1,
            aggr: 0,
            perpage: 20,
            n: 20,
```

```
            p: 1,
            remoteplace: 'txt.mqq.all',
            _: Date.now()
        };
        wx.request({
            url: 'http://c.y.qq.com/soso/fcgi-bin/search_for_qq_cp',
            data: data,
            header: {
                'Content-Type': 'application/json'
            },
            success: function (res) {
                if (res.statusCode == 200) {
                    callback(res.data);
                } else {

                }

            }
        });
}
...
module.exports = {
  ...
  getSearchMusic:getSearchMusic
}
```

返回的 JSON 数据为：

```
{
    "code": 0,
    "data": {
        "keyword": "那片星空那片海 ",
        "priority": 0,
        "qc": [],
        "semantic": {
            "curnum": 0,
            "curpage": 1,
            "list": [],
            "totalnum": 0
        },
```

```
    "song": {
        "curnum": 6,
        "curpage": 1,
        "list": [
            {
                "albumid": 1829165,
                "albummid": "000cvc411fKPIC",
                "albumname": "那片星空那片海 电视原声带",
                "albumname_hilight": "<span class=\"c_tx_highlight\">那片星空
                    那片海</span> 电视原声带",
                "alertid": 100007,
                "chinesesinger": 0,
                "docid": "8313377529987609872",
                "format": "qqhq;common;mp3common;wmacommon",
                "grp": [],
                "interval": 227,
                "isonly": 1,
                "lyric": "《那片星空那片海》电视剧片头曲",
                "lyric_hilight": "《<span class=\"c_tx_highlight\">那片星空那
                    片海</span>》电视剧<span class=\"c_tx_highlight\">片</span
                    >头曲",
                "msgid": 13,
                "nt": 3247935513,
                "pay": {
                    "payalbum": 1,
                    "payalbumprice": 500,
                    "paydownload": 1,
                    "payinfo": 1,
                    "payplay": 1,
                    "paytrackmouth": 0,
                    "paytrackprice": 0
                },
                "preview": {
                    "trybegin": 0,
                    "tryend": 0,
                    "trysize": 0
                },
                "pubtime": 1486483200,
                "pure": 0,
```

```
            "singer": [
                {
                    "id": 12111,
                    "mid": "004YXxql1sSr2o",
                    "name": "金志文",
                    "name_hilight": "金志文"
                }
            ],
            "size128": 3649005,
            "size320": 9122181,
            "sizeape": 23599481,
            "sizeflac": 23676261,
            "sizeogg": 4916985,
            "songid": 200552006,
            "songmid": "001lggCh2Nw7us",
            "songname": "那片星空那片海",
            "songname_hilight": "<span class=\"c_tx_highlight\">那片星空那
                片海</span>",
            "songurl": "http://y.qq.com/#type=song&id=200552006",
            "stream": 5,
            "switch": 594177,
            "t": 0,
            "tag": 10,
            "type": 0,
            "ver": 0,
            "vid": ""
        },
        ...     //略去了部分相似内容
    ],
    "totalnum": 6
},
"totaltime": 0,
"zhida": {
    "albumid": 1829165,
    "albummid": "000cvc411fKPIC",
    "albumname": "那片星空那片海 电视原声带",
    "singername": "华语群星",
    "type": 3
}
```

```
    },
    "message": "",
    "notice": "",
    "subcode": 0,
    "time": 1487063618,
    "tips": ""
}
```

有了数据之后，我们需要在列表里显示的内容为歌曲名称（songname）与歌手名称（singer.name）。完成后的列表样式如图 10-7 所示。

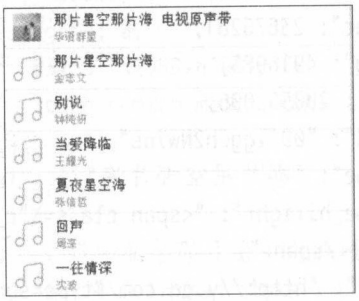

图 10-7　搜索结果样图

这个列表分为两个部分，首先是最上方的带图片的专辑信息，它的数据源为返回数据里“zhida”这个部分，然后下面就是歌曲信息了，其前面的音乐图标为本地文件。

```
<view class="ssong-list" wx:if="{{showSearchPanel == 3}}">
    <block wx:if="{{zhida.type == 3}}">
      <view class="ssong-item" data-data="{{zhida}}" data-id="{{zhida.albummid}}"
          bindtap="zhidaTap">
        <image class="ss-icon" mode="aspectFit" src="http://y.gtimg.cn/music/
            photo_new/T002R68x68M000{{zhida.albummid}}.jpg?max_age=2592000"></image
            >
        <view class="ss-text">
          <text class="ss-title">{{zhida.albumname}}</text>
          <text class="ss-sub-title">{{zhida.singername}}</text>
        </view>
      </view>
    </block>
    <view class="ssong-item" wx:for="{{searchSongs}}" wx:key="unique" data-index="
        {{index}}" bindtap="musuicPlay">
```

```
      <image class="ss-icon" mode="aspectFit" src="../../resources/images/song_icon
          .png"></image>
      <view class="ss-text">
        <text class="ss-title">{{item.songname}}</text>
        <text class="ss-sub-title">
          <block wx:for="{{item.singer}}">{{item.name}}</block>
        </text>
      </view>
    </view>
  </view>
```

首先这部分代码对应的内容只有在 showSearchPanel=3 的时候才显示，然后我们用 block 包裹了“zhida”这部分 view，在判断返回数据中，zhida.type 等于 3 后（表示搜索结果里包涵专辑结果，如果没有，那么这个值为 0），才渲染这部分代码。包裹的内容与下方列表内容一致，都是由图片、主标题（歌曲/专辑名称）与副标题（歌手名称）组成，只不过专辑里的图片为网络图片，通过拼接字符串的方式（即使用我们获得的专辑 id 代替 url 里 {{zhida.albummid}} 这部分组成完整地址）获取图片，而歌曲中我们使用的是本地图片。

这一部分的格式文件为：

```
.ssong-list {
    background:#fff;
}
.ssong-item {
    position:relative;
    height:88rpx;
    border-bottom:2rpx solid #ededed;
}
.ss-icon {
    position:absolute;
    left:20rpx;
    top:14rpx;
    width:60rpx;
    height:60rpx;
}
.ss-text {
    position:absolute;
    left:108rpx;
    right:20rpx;
```

```
    padding:10rpx 0;
}
.ss-title {
    font-size:32rpx;
    white-space:nowrap;
    text-overflow:ellipsis;
    overflow:hidden;
}
.ss-sub-title {
    display: block;
    font-size:24rpx;
    color:#999;
    padding-top:4rpx;
    white-space:nowrap;
    text-overflow:ellipsis;
    overflow:hidden;
}
```

10.4.5 检索页（下）

到目前为止，我们已经完成了检索页所有的布局和格式文件，也完成了所有的数据请求函数，现在需要将这些串联起来，实现这三部分之间的切换。

总结一下我们需要完成的逻辑有：

(1) 点击热门关键字，页面内容变成搜索结果页（需要请求网络，用我们写好的 getSearchMusic 方法），输入框内显示点击的关键字，同时这个关键字加入历史搜索结果。

(2) 在页面内容为热门关键字的时候，点击输入框使其获得焦点时，页面内容变成历史记录。

(3) 点击历史记录文字，内容变成搜索结果（请求网络），输入框显示这个记录。

(4) 点击历史记录每条末尾的“X”按钮，删除这一条记录，当所有记录全部删除或点击了“清除历史记录”选项，内容变为热门关键字。

(5) 输入框内的内容被全部删除后，也返回热门关键字。

(6) 点击“确认”按钮，内容变为搜索结果页，同时加入历史记录。

(7) 点击搜索结果的 item，页面转到专辑页或音乐播放页。

实现这些逻辑的相关事件我们已经在页面里注册好了，在具体实现这些事件之前，我们先写一个函数——将字符串加入到历史记录。

这个函数很简单，我们在写历史记录页面时，已经用到了 historySearchs 这个数组，所以添加时只要获取这个数组，然后将要添加的词 push 到数组里，用 setData 更新页面就可以了。

```
addHistorySearchs: function (key) {
        var historySearchs = this.data.historySearchs;
            historySearchs.push(key);
            this.setData({
                historySearchs: historySearchs
            })
    },
```

但是这样做面临的问题是：当用户多次搜索相同内容时，数组内就会多次加入同样的词，导致历史记录列表里出现重复内容，这显然是不合理的，所以在每次 push 前，需要判断数组内是否已经含有这个词。

```
findHistorySearchs: function (key) {
        var historySearchs = this.data.historySearchs;
        for (var i = 0; i < historySearchs.length; i++) {
            if (historySearchs[i] == key) { return false; }
        }
        return true;
    },
```

创建新的函数，这个函数会遍历 historySearchs 数组，如果存在相同项，则返回 false，否则返回 true。

然后更改 addHistorySearchs 方法：

```
addHistorySearchs: function (key) {
        var historySearchs = this.data.historySearchs;
        if (this.findHistorySearchs(key)) {
            historySearchs.push(key);
            this.setData({
                historySearchs: historySearchs
            })
        }
    },
```

有个这个方法后，下面开始逐条完成我们的事件代码。

将所有更新页面有关变量添加到data里：

```
data: {
        slider: [],
        indicatorDots: true,
        autoplay: true,
        interval: 5000,
        duration: 1000,
        radioList: [],
        currentView: 1,
        topList: [],
        hotkeys: [],
        showSpecial: false,
        special: { key: '', url: '' },
        searchKey: '',
        searchSongs: [],
        zhida: {},
        showSearchPanel: 1,
        historySearchs: [],
    },
```

热门关键词的点击事件：

```
hotKeysTap: function (e) {
        var dataSet = e.currentTarget.dataset;
        var key = dataSet.key;                                    //获取点击的关键词
        var self = this;
        if (key != '') {                                          //判断是否为空
            self.addHistorySearchs(key);       //调用我们写好的方法，加入历史记录
            self.setData({
                searchKey: key,                                   //为输入框内添加文字
                showSearchPanel: 3,                               //显示内容切换为搜索结果
            });
            MusicService.getSearchMusic(key, function (data) {    //请求网络数据
                if (data.code == 0) {
                    var songData = data.data;
                    self.setData({                  //将获得的数据添加到相应数组里
                        searchSongs: songData.song.list,
                        zhida: songData.zhida
                    });
                }
```

```
            });
        }
    },
```

输入框获取焦点事件：

```
bindFocus: function (e) {
        var self = this;
        if (this.data.showSearchPanel == 1) {      //判断内容是否为热门关键词
            self.setData({
                showSearchPanel: 2                  //切换到历史记录
            })
        }
    },
```

历史记录文字的点击事件：

```
historysearchTap: function (e) {
        var dataSet = e.currentTarget.dataset;
        var key = dataSet.key;                      //获取点击的历史记录文字
        var self = this;
        self.setData({
            searchKey: key,                         //输入框添加文字
            showSearchPanel: 3                      //显示搜索结果
        });
        MusicService.getSearchMusic(key, function (data) {     //请求网络，获取搜
            索结果
            if (data.code == 0) {
                var songData = data.data;
                self.setData({
                    searchSongs: songData.song.list,
                    zhida: songData.zhida
                });
            }
        });
    },
```

历史记录结尾的“X”与“清除历史记录”的点击事件：

```
    delHistoryItem: function (e) {
        var historySearchs = this.data.historySearchs;
        var dataSet = e.currentTarget.dataset;          //获取点击的条目
```

```
        if (dataSet.index != 'undefined') {
            var _index = parseInt(dataSet.index);    //获取点击条目为数组的第几项
            historySearchs.splice(_index, 1);          //从数组里删除对应的条目
            this.setData({                                  //更新页面
                historySearchs: historySearchs
            });
            if(historySearchs.length==0){             //如果历史记录里没有数据了
                this.setData({
                    showSearchPanel: 1                      //切换到热门关键字
                })
            }
        }
    },
    clearHistorySearchs: function () {
        this.setData({
            historySearchs: [],                             //清空历史记录数组
            showSearchPanel: 1                              //切换到热门关键字
        })
    },
```

在输入框输入事件：

```
bindKeyInput: function (e) {
        var self = this;
        self.setData({                                      //更新searchKey的值
                searchKey: e.detail.value
            });
        if (e.detail.value == "") {          //如果值为空且当前未显示热门关键字
            if (this.data.showSearchPanel != 1) {
                self.setData({
                    showSearchPanel: 1                      //切换为热门关键字
                })
            }
        }
    },
```

“确认”按钮的点击事件：

```
searchOk: function (e) {
        var self = this;
        var searchKey = this.data.searchKey;                //获取searchKey的值
```

```
    if (searchKey != "") {
        self.setData({
            showSearchPanel: 3                                    //显示搜索结果
        });
        self.addHistorySearchs(searchKey);                        //添加到历史记录
        MusicService.getSearchMusic(searchKey, function (data) {
            if (data.code == 0) {
                var songData = data.data;
                self.setData({
                    searchSongs: songData.song.list,
                    zhida: songData.zhida
                });
            }
        });
    }
},
```

搜索结果 item 的点击事件，分为专辑与歌曲两种：

```
zhidaTap: function (e) {                                  //专辑的跳转事件
    var dataSet = e.currentTarget.dataset;
    var mid = dataSet.id;

    app.setGlobalData({ 'zhidaAlbummid': mid });     //将专辑id保存为全局变量
    wx.navigateTo({                                       //页面跳转
        url: '../cdinfo/cdinfo'
    })

},
musuicPlay: function (e) {                          //歌曲的跳转事件
    var dataSet = e.currentTarget.dataset;
    //TODO
    }
},
```

歌曲的跳转事件相对复杂（不只是要跳转，而且要加入播放列表），这里将其留到播放页再更改这部分吧。

至此，首页内容全部完成（准确地说还缺少推荐页与搜索结果页向音乐播放页跳转的事件）。

10.5 列表页

列表页要制作的列表有两个：点击排行里的某一种排行跳转的歌曲列表页与搜索歌曲中专辑点击的跳转页。由于这两部分比较相似，所以放在一节里一起完成。

10.5.1 获取列表页数据

下面先来制作点击排行里的项目所跳转的排行列表。

首先还是先编写一下网络请求的函数。

```
function getTopListInfo(id,callback){
    var data = {
            g_tk: 5381,
            uin: 0,
            format: 'json',
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            tpl: 3,
            page: 'detail',
            type: 'top',
            topid: id,
            _: Date.now()
        };
        wx.request({
            url: 'http://c.y.qq.com/v8/fcg-bin/fcg_v8_toplist_cp.fcg',
            data: data,
            header: {
                'Content-Type': 'application/json'
            },
            success: function (res) {
                if (res.statusCode == 200) {
                    callback(res.data);
                } else {

                }
```

```
            }
        });
}

module.exports = {
  ...
  getTopListInfo:getTopListInfo
}
```

返回的 JSON 格式数据为：

```
{
    "color": 14729248,
    "comment_num": 1859,
    "cur_song_num": 100,
    "date": "2017_6",
    "song_begin": 0,
    "songlist": [
        {
            "Franking_value": "1",
            "cur_count": "1",
            "data": {
                "albumdesc": "",
                "albumid": 1829744,
                "albummid": "003GpN3b2UiNx0",
                "albumname": "Run Up",
                "alertid": 100002,
                "belongCD": 1,
                "cdIdx": 0,
                "interval": 203,
                "isonly": 1,
                "label": "0",
                "msgid": 14,
                "pay": {
                    "payalbum": 0,
                    "payalbumprice": 0,
                    "paydownload": 1,
                    "payinfo": 1,
                    "payplay": 0,
                    "paytrackmouth": 1,
```

```
        "paytrackprice": 200,
        "timefree": 0
    },
    "preview": {
        "trybegin": 0,
        "tryend": 0,
        "trysize": 0
    },
    "rate": 7,
    "singer": [
        {
            "id": 38150,
            "mid": "001J99OK2689F4",
            "name": "Major Lazer"
        },
        {
            "id": 177284,
            "mid": "002R2h8l05rVWb",
            "name": "PARTYNEXTDOOR"
        },
        {
            "id": 20710,
            "mid": "003kJN1r16tgOt",
            "name": "Nicki Minaj"
        }
    ],
    "size128": 3251944,
    "size320": 8129528,
    "size5_1": 0,
    "sizeape": 0,
    "sizeflac": 0,
    "sizeogg": 4972861,
    "songid": 200556634,
    "songmid": "003kPDEH3fSvh8",
    "songname": "Run Up",
    "songorig": "Run Up",
    "songtype": 0,
    "strMediaMid": "00191A2Z3qTVEr",
    "stream": 0,
```

```
            "switch": 636675,
            "type": 0,
            "vid": ""
        },
        "in_count": "2",
        "mb": "",
        "old_count": "1",
        "singer2": {
            "Fgenre": "0",
            "Fsinger_id": "177284",
            "Fsinger_mid": "002R2h8l05rVWb",
            "Fsinger_name": "PARTYNEXTDOOR",
            "Ftype": "0"
        },
        "singer3": {
            "Fgenre": "0",
            "Fsinger_id": "20710",
            "Fsinger_mid": "003kJN1r16tgOt",
            "Fsinger_name": "Nicki Minaj",
            "Ftype": "1"
        },
        "vid": {
            "Fmv_id": "",
            "Fstatus": "",
            "Fvid": ""
        }
    },
    ...
    "topinfo": {
    "ListName": "巅峰榜·欧美",
    "MacDetailPicUrl": "http://y.gtimg.cn/music/common//upload/
        iphone_order_channel/20140519104205.jpg",
    "MacListPicUrl": "http://y.gtimg.cn/music/common//upload/
        iphone_order_channel/20140519104155.jpg",
    "UpdateType": "1",
    "albuminfo": "QQ音乐巅峰榜·欧美专辑根据用户收听行为自动生成，集结当
        下最流行的欧美新碟！:更新时间：每周五|统计周期：一周（上周五至本
        周四）|统计对象：一年内发行的欧美专辑|排名数量：100张|统计算法：
        根据专辑内所有歌曲在一周内的有效播放次数总和，由高到低取前100名|
```

```
            有效播放次数：登录用户完整播放一首歌曲，记为一次有效播放；同一用
            户收听同一首歌曲，每天最多记录20次有效播放",
        "headPic_v12": "http://y.gtimg.cn/music/common//upload/iphone_order_channel
            /20160224140905.jpg",
        "info": "QQ音乐巅峰榜·欧美根据用户收听行为自动生成，集结当下最流行的
            欧美新歌！<br><br>更新时间：每周四22点<br>统计周期：一周（上周四
            至本周三）<br>统计对象：三个月内发行的欧美歌曲<br>统计数量：100首
            <br>统计算法：根据歌曲在一周内的有效播放次数，由高到低取前100名
            （同一歌手最多允许5首歌曲同时上榜）<br>有效播放次数：登录用户完整
            播放一首歌曲，记为一次有效播放；同一用户收听同一首歌曲，每天记录
            为1次有效播放。",
        "listennum": 9800000,
        "pic": "http://y.gtimg.cn/music/common//upload/iphone_order_channel
            /20141110194323.jpg",
        "picDetail": "http://y.gtimg.cn/music/common//upload/iphone_order_channel
            /20140519104128.jpg",
        "pic_album": "http://imgcache.qq.com/music/photo_new/
            T002R300x300M000003GpN3b2UiNx0.jpg",
        "pic_h5": "http://y.gtimg.cn/music/common//upload/iphone_order_channel
            /20140519104015.jpg",
        "pic_v11": "http://y.gtimg.cn/music/common//upload/iphone_order_channel
            /20141110194323.jpg",
        "pic_v12": "http://y.gtimg.cn/music/common/upload/iphone_order_channel/
            toplist_3_300_200556634.jpg",
        "topID": "3",
        "type": "0"
    },
    "total_song_num": 100,
    "update_time": "2017-02-09"
}
```

请求网络时用到的参数 ID 是我们在排行页面点击的 item 的 id，在 10.4 节了已经通过点击事件将它保存在全局变量里了。

在编写布局文件之前先来看一下最后完成后的样子，如图 10-8 所示。

可以看到这个页面主要由两部分组成，上半部分为榜单的图片以及名字等信息，下半部分是歌曲列表。

图 10-8 分类列表样图

先从歌曲列表开始，列表的每一项由序号、歌曲名称、演唱者（多名演唱者以“|”分隔）以及专辑名称组成，所以我们的布局文件为：

```
<view class="song-list" >
    <view class="song-item" wx:for="{{songList}}" wx:key="unique" data-data="{{item
        .data}}" data-index="{{index}}" bindtap="musicItemTap">
        <text class="song-index">{{index+1}}</text>
        <text class="song-item-title">{{item.name}}</text>
        <view class="song-item-text">
            <block wx:for="{{item.singer}}"  wx:key="unique">
                <block wx:if="{{index!=0}}">|</block>
                {{item.name}}
            </block>
            · {{item.album.name}}
        </view>
    </view>
</view>
```

格式文件为：

```
.song-list {
    padding-left: 100rpx;
    background: #000000;
}
```

```
.song-item {
    position: relative;
    padding: 20rpx 20rpx 20rpx 0;
    border-bottom: 2rpx solid rgba(255,255,255, 0.5);
}

.song-index {
    position: absolute;
    left: -80rpx;
    width: 80rpx;
    color: #fff;
    font-size: 40rpx;
    text-align: center;
    line-height: 100rpx;
}
.song-item-title {
    height: 60rpx;
    line-height: 60rpx;
    font-size: 32rpx;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    color: #fff;
}
.song-item-text {
    font-size: 28rpx;
    color: rgba(255,255,255, .75);       //表示颜色
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}
```

这部分我们应该比较熟悉了，需要注意的就是多个歌手的书写形式，除第一个歌手外，其他都在前面加上“|”就是我们要的形式了。

接下来我们完成上半部分，这部分的布局代码很简单：

```
<view class="list-top">
    <view class="top-info">
        <view class="top-info-inner">
            <view class="top-info-text">
```

```
                <view class="top-info-title">{{topinfo.ListName}}</view>
                <view class="top-info-time">
                    <text>{{update_time}}  更新</text>
                </view>
            </view>
            <view class="top-play" bindtap="mainTopTap"></view>
        </view>
    </view>
    <image class="top-img"  mode="aspectFit" src="{{topinfo.pic_album}}"></image>
    <view class="top-back"></view>
</view>
```

前面在 tl-top-info-inner 制作完成了文字部分与右边的播放按钮，又在这部分之后加载了图片，这样页面大体的布局就完成了。但是这样做会产生一个问题：我们设置文字为白色，如果加载的图片也为白色背景，那么文字就会很难看清了。

10.5.2 页面渲染

为了解决这个问题，下面给图片添加一个渐变的遮罩，就像图 10-8 所示的那样，这样到达文字部分时，背景就变成了黑色，不会影响文字的显示，而且达到了由图片到底下列表颜色渐变的效果，非常美观。

这个效果主要靠格式文件实现，下面先写我们熟悉的部分。

```
.list-top {
    position: relative;
    height: 100%;
}
.list-top::after {
    content: " ";
    display: block;
    padding-top: 100%;
}
.top-info {
    position: absolute;
    bottom: 0;
    width: 100%;
    z-index: 3;
}
.top-img {
    width: 100%;
```

```
    height: 100%;
    position: absolute;
}

.top-info-inner {
    display: -webkit-box;
    -webkit-box-align: center;
    margin: 0 30rpx 50rpx;
    color: #fff;
}

.top-info-text {
    -webkit-box-flex: 1;
    margin-right: 20rpx;
}
.top-info-title {
    font-size: 48rpx;
    line-height: 72rpx;
    white-space: nowrap;
    overflow: hidden;
}
.top-info-base {
    font-size: 28rpx;
    line-height: 40rpx;
}
```

“::after”表示在“.list-top”后边添加，为了在不修改布局文件的情况下，添加视图以达到美化的效果。

我们需要添加的遮罩为布局里“top—back”这部分，格式文件为：

```
.tl-top-b {
    position: absolute;
    bottom: 0;
    width: 100%;
    background-image: -webkit-linear-gradient(top,transparent,currentColor 80%);
}
.tl-top-b::after {
    content: " ";
    display: block;
    padding-top: 60%;
}
```

-webkit-linear-gradient(top,transparent,currentColor 80%) 这行代码建立了线性渐变的效果，这样图片底部就会出现渐变为黑色的效果了。

下面设置“播放”按钮的样式，这里因为用到了渐变的遮罩和背景图，为了达到最好的效果，这里选择的图片应该为透明背景的图片，如果你熟悉 CSS，那么也可以使用代码创建这个图标。

```
.tl-top-play {
    position: relative;
    display: block;
    width: 84rpx;
    height: 84rpx;
    margin-left: 20rpx;
    background: url("../../resources/images/play_btn_play.png");
    background-size: cover;
}
```

视图建立完毕，开始为视图填充数据。

```
//加载网络请求函数
var MusicService = require('../../services/music');
//获取应用实例
var app = getApp();

Page({
    data: {
        // text:"这是一个页面"
        songList: [],
        imgUrl: '',
        id: 0,
        topinfo: {},
        update_time: '',
    },
    onLoad: function (options) {
        // 页面初始化 options为页面跳转所带来的参数
        var self = this;
        var id = app.globalData.topListId;
        this.setData({
            id: id
        });
```

```
        MusicService.getTopListInfo(id, this.getTopListCallback)
    },
})
```

这里获取了保存于全局变量里的 topListId（即我们点击的排行分类的 ID），然后使用这个 ID 请求网络。

```
getTopListCallback: function (data) {
        var imgUrl = data.topinfo.pic_album;
        this.setData({
            topinfo: data.topinfo,
            update_time: data.update_time
        });
        this.setSongList(data.songlist);
    },
```

使用回调函数为 data 赋值之后，这里调用了 setSongList 这个方法，通过这个方法把返回数据中我们需要的内容保存到 songList 中。

```
setSongList: function (songs) {
        var list = [];
        for (var i = 0; i < songs.length; i++) {
            var item = songs[i];
            var song = {};
            var album = {};

            album.mid = item.data.albummid
            album.id = item.data.albumid
            album.name = item.data.albumname;
            album.desc = item.data.albumdesc

            song.id = item.data.songid;
            song.mid = item.data.songmid;
            song.name = item.data.songname;
            song.title = item.data.songorig;
            song.subTitle = '';
            song.singer = item.data.singer;
            song.album = album;
            song.time_public = item.time_public;
            song.img = 'http://y.gtimg.cn/music/photo_new/T002R150x150M000' + album
                .mid + '.jpg?max_age=2592000'
```

```
            list.push(song);
        }
        this.setData({
            songList: list
        })
    }
```

下面完成此页面里的点击事件：

```
mainTopTap: function (e) {
        var list = this.data.songList;
        app.setGlobalData({        //使用全局变量playList来保存我们当前的list
            playList: list,
            playIndex: 0                    //表示从第一首歌曲开始播放
        });
        wx.navigateTo({
            url: '../play/play'             //跳转到播放页
        });
    },
    musicItemTap: function (e) {
        var dataSet = e.currentTarget.dataset;
        var index = dataSet.index;                  //获取点击的item的序号
        var list = this.data.songList;
        app.setGlobalData({
            playList: list,
            playIndex: index                        //从点击歌曲开始播放
        });
        wx.navigateTo({
            url: '../play/play'
        });
    },
```

10.5.3 完成相似页面

专辑的页面与我们刚刚完成的排行列表很相似，除部分文字的变化外就是多了简介这一部分。

网络请求的函数为：

```
function getAlbumInfo(id,callback){
    var data = {
            albummid: id,
```

```
            g_tk: 5381,
            uin: 0,
            format: 'json',
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            _: Date.now()
        };
        wx.request({
            url: 'http://c.y.qq.com/v8/fcg-bin/fcg_v8_album_info_cp.fcg',
            data: data,
            header: {
                'Content-Type': 'application/json'
            },
            success: function (res) {
                console.log(res);
                if (res.statusCode == 200) {
                    callback(res.data);
                } else {

                }
            }
        });
}

module.exports = {
  ...
  getAlbumInfo:getAlbumInfo
}
```

页面的布局代码为：

```
<view class="list-top">
  <view class="top-info">
    <view class="top-info-inner">
      <view class="top-info-text">
        <view class="top-info-title">{{albumInfo.name}}</view>
        <view class="top-info-base">
```

```
          <text>{{albumInfo.singername}}</text>
          <text style="margin-left: 5px;">{{albumInfo.aDate}}</text>
          <text style="margin-left:10px;">{{albumInfo.genre}}</text>
        </view>
      </view>
      <view class="top-play"></view>
    </view>
  </view>
  <image class="top-img" mode="aspectFit" src="{{coverImg}}"></image>
  <view class="top-back"></view>
</view>
<view class="song-list" style="background:{{listBgColor}}">
  <view class="song-item" wx:for="{{albumInfo.list}}" data-data="{{item.data}}"
      data-mid="{{item.songmid}}">
    <text class="song-index">{{index+1}}</text>
    <view class="song-item-title">{{item.songname}}</view>
    <view class="song-item-text">
      <block wx:for="{{item.singer}}">
        <block wx:if="{{index!=0}}">|</block>
        {{item.name}}
      </block>
    </view>
  </view>
</view>
<view class="desc" style="background:{{listBgColor}}">
  <view class="desc-title">简介</view>
  <text>{{albumInfo.desc}}</text>
</view>
```

简介部分的格式文件：

```
.desc {
  box-sizing: border-box;
  font-size: 28rpx;
  padding: 80rpx 20rpx;
  color: #fff;
  line-height: 40rpx;
}

.desc-title {
```

```
  text-align: center;
  width: 100%;
  font-size: 32rpx;
  margin-bottom: 40rpx;
}
```

加载数据的代码为：

```
var MusicService = require('../../services/music');
var app = getApp()

Page({
    data: {
        albumInfo: {},
        coverImg: '',
    },
    onLoad: function (options) {
        // 页面初始化 options为页面跳转所带来的参数
        var mid = app.globalData.zhidaAlbummid;
        MusicService.getAlbumInfo(mid, this.setPageData)
    },
    setPageData: function (data) {
        if (data.code == 0) {
            var albummid = data.data.mid;
            var img = 'http://y.gtimg.cn/music/photo/mid_album_500/' + albummid.
                slice(-2, -1) + '/' + albummid.slice(-1) + '/' + albummid + '.jpg'
            this.setData({albumInfo: data.data, coverImg: img});
        }
    },
})
```

这里的点击事件与前文相同，就不再重复介绍了。

另外，前面在首页里未完成的两个点击事件，现在也可以完成了。先看电台的点击事件，这个事件与我们刚刚完成的一样，具体代码为：

```
radioTap: function (e) {
        var dataSet = e.currentTarget.dataset;
        MusicService.getRadioMusicList(dataSet.id, function (data) {
            wx.navigateTo({
                url: '../play/play'
            });
```

```
        if (data.code == 0) {
            var list = [];
            var dataList = data.data;
            for (var i = 0; i < dataList.length; i++) {
                var song = {};
                var item = dataList[i];
                song.id = item.id;
                song.mid = item.mid;
                song.name = item.name;
                song.title = item.title;
                song.subTitle = item.subtitle;
                song.singer = item.singer;
                song.album = item.album
                song.img = 'http://y.gtimg.cn/music/photo_new/T002R150x150M000'
                    + item.album.mid + '.jpg?max_age=2592000'
                list.push(song);
            }
            app.setGlobalData({
                playList: list,
                playIndex: 0
            });
        }
    });
},
```

这里面 getRadioMusicList 为网络请求，具体代码为：

```
function getRadioMusicList(id,callback){
    var data = {
            labelid: id,
            g_tk: 5381,
            uin: 0,
            format: 'json',
            inCharset: 'utf-8',
            outCharset: 'utf-8',
            notice: 0,
            platform: 'h5',
            needNewCode: 1,
            _: Date.now(),
        }
```

```
        wx.request({
            url: 'https://c.y.qq.com/v8/fcg-bin/fcg_v8_radiosonglist.fcg',
            data: data,
            header: {
                'Content-Type': 'application/json'
            },
            success: function (res) {
                if (res.statusCode == 200) {
                    callback(res.data);
                } else {

                }

            }
        });
}

module.exports = {
  ...
  getRadioMusicList:getRadioMusicList
}
```

另一部分为搜索结果里歌曲的点击事件:

```
musuicPlay: function (e) {
        var dataSet = e.currentTarget.dataset;
        var playingSongs = app.globalData.playList;
        if (typeof dataSet.index !== 'undefined') {
            var index = dataSet.index;
            var item = this.data.searchSongs[index];
            var song = {};
            var album = {};
            album.mid = item.albummid
            album.id = item.albumid
            album.name = item.albumname;
            album.desc = item.albumdesc

            song.id = item.songid;
            song.mid = item.songmid;
            song.name = item.songname;
```

```
            song.title = item.songorig;
            song.subTitle = '';
            song.singer = item.singer;
            song.album = album;
            song.time_public = item.time_public;
            song.img = 'http://y.gtimg.cn/music/photo_new/T002R150x150M000' + album
                .mid + '.jpg?max_age=2592000'
            this.addPlayingSongs(song);
        }
    },
```

前面的内容与前面介绍的一样，最后我们没有直接更新全局变量而是调用了一个新方法，因为前文所有的点击事件都更新了整个播放列表，而我们点击某一首歌曲时，是希望添加这首歌到已有的列表中，而不是先清空它。

```
addPlayingSongs: function (song) {
        var playingSongs = app.globalData.playList;          //获取当前的播放列表
        var index = -1;
        if (typeof playingSongs === 'undefined') {           //判断列表是否为空
            playingSongs = [];
            playingSongs.push(song);
            app.setGlobalData({                    //如果是空，则直接更新全局变量
                playList: playingSongs,
                playIndex: 0
            });
        } else {                 //不为空的话我们先判断当前列表是否包含选定歌曲
            for (var i = 0; i < playingSongs.length; i++) {    //遍历整个列表
                var item = playingSongs[i];
                if (item.mid == song.mid) { //如果发现有mid相同的（即同一首歌）
                    index = i;                        //获取这首歌在列表里的序号
                    break;
                }
            }
            if (index != -1) {                      //歌曲已存在
                app.setGlobalData({
                    playIndex: index          //用我们获取的序号更新当前播放序号
                });
            } else {                                        //不存在的情况
                playingSongs.push(song);
                index = playingSongs.length - 1;    //将歌曲加入播放列表，播放序
```

```
                号改为列表最后一项
            app.setGlobalData({
                playList: playingSongs,
                playIndex: index
            });
        }
    }
    wx.navigateTo({
        url: '../play/play'
    });
},
```

10.6 音乐播放页

最后制作完成音乐播放页面，在前文的基础上这部分的逻辑代码很好完成，下面将重点放在页面布局上，先看一下我们希望达到的效果（见图 10-9）。

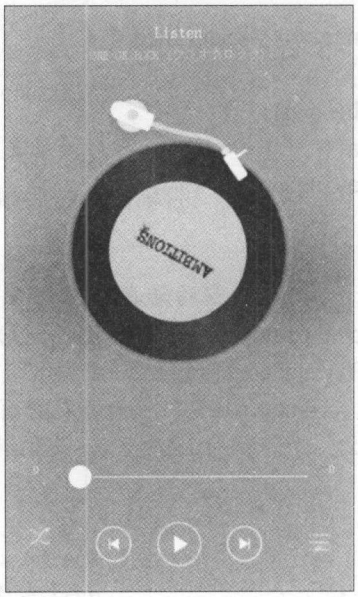

图 10-9 播放页样图

这个页面分为 3 个部分：顶部信息栏（歌曲名/专辑名）、中间的唱片图，以及底下的控制按钮。

首先是信息栏：

```
<view class="song-info">
      <text class="song-title">{{playingMusic.name}}</text>
      <text class="song-subtitle">
        <block wx:for="{{playingMusic.singer}}" wx:key="unique">
          <block wx-if="{{index!=0}}">*</block>{{item.name}}</block>
      </text>
    </view>
```

这部分很简单，与我们前面写过的类似，多个歌手之间用“*”分隔。格式文件为：

```
.song-info {
    width:100%;
    padding:30rpx;
    box-sizing:border-box;
    text-align:center;
}
.song-title {
    display:block;
    width:100%;
    color:#fff;
    font-size:36rpx;
    line-height:60rpx;
    overflow:hidden;
    white-space:nowrap;
    text-overflow:ellipsis;
}
.song-subtitle {
    display:block;
    width:100%;
    font-size:28rpx;
    line-height:40rpx;
    color:rgba(255,255,255,.6);
    overflow:hidden;
    white-space:nowrap;
    text-overflow:ellipsis;
}
```

然后是中间的图片，这部分以动画形式表现，让唱片一直旋转，先看布局文件：

```
<view class="cd-info">
      <view class="cd-gan"></view>
```

```
      <view class="cd-inner cd-animation">
        <image class="cd-img" src="{{playingMusic.img}}"></image>
      </view>
    </view>
```

“cd-gan”是唱片部分里白色的播放杆部分，“cd-inner”是唱片部分，在这里为它添加表示唱片的黑色背景，然后在里门用小一圈的“cd-img”来加载我们获取的网络图片。最后为整个唱片添加动画“cd-animation”。这些工作全部由格式文件完成。

```
.cd-info {
  position: relative;
  width: 100%;
  text-align: center;
  padding-top: 120rpx;
}

.cd-gan {
  position: absolute;
  left: 50%;
  margin-top: -80rpx;
  margin-left: -150rpx;
  display: block;
  width: 300rpx;
  height: 200rpx;
  background: url('../../resources/images/cd_g.png');
  background-size: cover;
  z-index: 10;
}

.cd-inner {
  position: relative;
  display: inline-block;
  z-index: 4;
  height: 500rpx;
  width: 500rpx;
  background: url('../../resources/images/cd_b.png');
  background-size: cover;
  text-align: center;
  padding-top: 100rpx;
  box-sizing: border-box;
```

```
}

.cd-animation {
  -webkit-animation: circle 10s infinite linear;
  animation: circle 10s infinite linear;
}

.cd-img {
  display: inline-block;
  width: 300rpx;
  height: 300rpx;
  border-radius: 50%;
}

@keyframes circle {
  0% {
    transform: rotate(0deg);
  }

  100% {
    transform: rotate(360deg);
  }
}
```

这里面大多数代码相信读者已经很熟悉了，重点看一下“cd-animation”这一部分，这里加载了动画“circle”并设置了动画时长与无限循环，从而实现了在音乐播放时，唱片一直旋转的效果。“circle”动画使用关键帧 keyframes 来实现。

完成这两部分后我以一个 view 来包裹它们，确定它在页面的位置。

```
<view class="main-box">
    <view class="song-info">
      <text class="song-title">{{playingMusic.name}}</text>
      <text class="song-subtitle">
        <block wx:for="{{playingMusic.singer}}" wx:key="unique">
          <block wx-if="{{index!=0}}">*</block>{{item.name}}</block>
      </text>
    </view>
    <view class="cd-info">
      <view class="cd-gan"></view>
```

```
      <view class="cd-inner cd-animation">
        <image class="cd-img" src="{{playingMusic.img}}"></image>
      </view>
    </view>
  </view>
```

```
.main-box {
  position: absolute;
  top: 0;
  bottom: 308rpx;
  z-index: 3;
  width: 100%;
  background: rgba(0, 0, 0, 0.2);
}
```

接着我们完成底下的操作部分。

```
<view class="ctre-box">
    <view class="slider-box">
      <text class="slider-text st-l">{{currTimeStr}}</text>
      <slider class="slider-inner"></slider>
      <text class="slider-text st-r">{{musicTimeStr}}</text>
    </view>
    <view class="music-ctr">
      <block wx-if="{{playType==0}}">
        <view class="music-sort ms-loop" data-type="{{playType}}" bindtap="
            changePlayType"></view>
      </block>
      <block wx-if="{{playType==1}}">
        <view class="music-sort ms-shuffle" data-type="{{playType}}" bindtap="
            changePlayType"></view>
      </block>
      <block wx-if="{{playType==2}}">
        <view class="music-sort ms-one" data-type="{{playType}}" bindtap="
            changePlayType"></view>
      </block>
      <view class="mc-inner">
        <view class="mci-icon mci-prev"></view>
        <view class="mci-icon mci-play"></view>
        <view class="mci-icon mci-next"></view>
      </view>
```

```
      <view class="music-list-btn" bindtap="showPlayList"></view>
    </view>
  </view>
```

操作控制部分由最上边的进度条部分“slider-box”和底下的操作按钮“mucis-ctr”组成。进度条使用 slider 组件，两段用两个 text 组件来显示当前播放时间与总时长。对于操作按钮部分：首先是播放模式的按钮，根据 playType 的值，改变顺序/随机/单曲的播放模式并对应加载不同的图片；然后是 3 个按钮，分别表示前一首、播放、下一首；最后是显示播放列表的按钮。

格式文件为：

```
.slider-box {
  box-sizing: border-box;
  padding: 20rpx 130rpx;
}

.slider-text {
  position: absolute;
  display: block;
  width: 100rpx;
  height: 40rpx;
  line-height: 40rpx;
  font-size: 24rpx;
  color: #fff;
}

.st-l {
  left: 60rpx;
}

.st-r {
  top: 20rpx;
  right: 40rpx;
  text-align: right;
}

.slider-inner {
  width: 100%;
}
```

```
.ctre-box {
  height: 308rpx;
  position: absolute;
  bottom: 0;
  z-index: 3;
  width: 100%;
  background: rgba(0, 0, 0, 0.2);
}

.music-ctr {
  position: relative;
  padding: 20rpx 120rpx;
}

.music-sort {
  position: absolute;
  left: 20rpx;
  width: 108rpx;
  height: 108rpx;
}

.ms-loop {
  background: url("../../resources/images/play_icn_loop.png");
  background-size: cover;
}

.ms-one {
  background: url("../../resources/images/play_icn_one.png");
  background-size: cover;
}

.ms-shuffle {
  background: url("../../resources/images/play_icn_shuffle.png");
  background-size: cover;
}

.music-list-btn {
  position: absolute;
  top: 36rpx;
```

```
  right: 20rpx;
  width: 108rpx;
  height: 108rpx;
  background: url("../../resources/images/play_icn_src.png");
  background-size: cover;
}

.mc-inner {
  text-align: center;
}

.mci-icon {
  display: inline-block;
  width: 142rpx;
  height: 142rpx;
}

.mci-prev {
  background: url("../../resources/images/play_btn_prev.png");
  background-size: cover;
}

.mci-play {
  background: url("../../resources/images/play_btn_play.png");
  background-size: cover;
}

.mci-pause {
  background: url("../../resources/images/play_btn_pause.png");
  background-size: cover;
}

.mci-next {
  background: url("../../resources/images/play_btn_next.png");
  background-size: cover;
}
```

最后写一下播放列表的布局：

```
<view class="play-list" hidden="{{showPlayList}}">
```

```
    <view class="play-list-header">
      <text>播放列表（185）</text>
      <text class="play-list-clear">清空</text>
    </view>
    <view class="play-list-inner">
      <block wx:for="{{playList}}" wx:key="unique">
        <view class="play-item">
          {{item.name}}
        </view>
      </block>
    </view>
    <view class="play-list-bottom" bindtap="closePlayList">关闭</view>
  </view>
```

格式文件为：

```
.play-list {
  position: absolute;
  top: 20%;
  bottom: 0;
  left: 0;
  width: 100%;
  z-index: 99;
  background: rgba(255, 255, 255, 0.95);
}

.play-list-header {
  line-height: 88rpx;
  font-size: 32rpx;
  text-align: center;
  border-bottom: 2rpx solid rgba(0, 0, 0, 0.1);
}

.play-list-clear {
  position: absolute;
  right: 30rpx;
  font-size: 28rpx;
  color: rgba(0, 0, 0, 0.6);
}
```

```
.play-list-bottom {
  position: absolute;
  width: 100%;
  bottom: 0;
  height: 100rpx;
  line-height: 100rpx;
  text-align: center;
  font-size: 32rpx;
  border-top: 2rpx solid rgba(0, 0, 0, 0.1);
}

.play-list-inner {
  position: absolute;
  top: 90rpx;
  bottom: 102rpx;
  width: 100%;
  overflow-x: hidden;
  overflow-y: auto;
  padding-left: 20rpx;
}

.play-item {
  position: relative;
  widows: 100%;
  box-sizing: border-box;
  padding-right: 90rpx;
  height: 88rpx;
  line-height: 88rpx;
  font-size: 30rpx;
  border-bottom: 2rpx solid rgba(0, 0, 0, 0.1);
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

这里使用“z-index: 99;background: rgba(255, 255, 255, 0.95);”使播放列表覆盖部分音乐播放页面且背景半透明。

最后使用一个 view 来为整个页面加载背景，这个背景为我们获取的图片加上模糊效果。最后用一个 view 包裹所有布局。

```
<view class="play-view">
   ...
   <view class="bg blur" style="background-image:url('{{playingMusic.img}}');"></
       view>
</view>
```

使用格式文件添加模糊效果：

```
.paly-view {
  display: block;
  position: relative;
  width: 100%;
  height: 100%;
  overflow: hidden;
}

.blur {
  filter: blur(80rpx);
}

.bg {
  position: absolute;
  left: 0;
  right: 0;
  top: 0;
  bottom: 0;
  background-size: cover;
  background-position: bottom center;
}
```

最后加载数据：

```
var app = getApp()

Page({
    data: {
        playList: [],
        playIndex: 0,
        showPlayList: true,
        playingMusic: {},
        musicTime: 0,
```

```
        currTime: 0,
        musicTimeStr: 0,
        currTimeStr: 0,
        isPlay: false,
        playInv: 0,
        playPro: '',
        playType: 1
    },
    onLoad: function (options) {
        // 页面初始化 options为页面跳转所带来的参数
        var self = this;
        var list = app.globalData.playList;
        var playingMusic = null;
        if (list.length) {
            var index = app.globalData.playIndex;
            index = (list.length - 1 < index) ? list.length - 1 : index;
            playingMusic = list[index];
            this.setData({
                playList: list,
                playIndex: index,
                playingMusic: playingMusic
            });
        }
        wx.playBackgroundAudio({
            dataUrl: list[index].url,
            title: list[index].title,
            coverImgUrl: list[index].img,
            success: function () {
            },
            fail: function () {
                console.log('播放失败!');
            }
        });
    },
    changePlayType: function (e) {
        var dataSet = e.currentTarget.dataset;
        if (dataSet.type == 1) {
            this.setData({
                playType: 2
```

```
            });
        }
        if (dataSet.type == 2) {
            this.setData({
                playType: 0
            });
        }
        if (dataSet.type == 0) {
            this.setData({
                playType: 1
            });
        }
    },
    closePlayList: function (e) {
        this.setData({
            showPlayList: true
        })
    },
    showPlayList: function (e) {
        this.setData({
            showPlayList: false
        })
    },
    //三个按钮的点击事件
    pauseMusic: function () {
    },
    playNextMusic: function () {
    },
    playPreMusic:function(){
    },
})
```

10.7 本章小结

至此，音乐播放小程序就基本制作完成了。

在最后的音乐播放页面内并没有加载歌曲，但是这肯定难不倒聪明的读者们，通过已经获取的歌曲 id 即可获取音乐的播放地址。另外播放页许多点击事件和前文一致，这里就不重复讲解了。

本案例只为给读者展现微信小程序的编写过程并回顾学习的知识，许多地方做得并不完美。但是相信读到这里的读者们都已经掌握了微信小程序的开发技巧了，继续完善这个项目或者开发自己的程序肯定不在话下了。俗话说，熟能生巧，快去根据自己的想法实现个性化的小程序吧。

11 发布微信小程序

在前面的10章，分别由浅入深地介绍了微信小程序的简史、工具的使用、JavaScript语法、微信小程序布局以及所有API接口，相信大家通过阅读本书，一定可以高效地开发出自己的微信小程序APP了。本章介绍发布微信小程序时的一些设置以及问题。

11.1 设置服务器域名

微信小程序在进行网络通信前，需要先到公众平台后台设置域名，不然会报错，微信小程序的网络通信只能使用指定的域名进行，而微信小程序的网络通信类型基本上分为4种，如表11-1所示。

表11-1 服务器域名配置表

服务器配置	说明	操作
request域名	一个月内可申请5次修改	普通HTTPS请求
socket域名	同上	WebSocket通信
uploadFile域名	同上	上传文件
downloadFile域名	同上	下载文件

设置步骤如下：可以通过微信公众平台入口 https://mp.weixin.qq.com 登录进入到微信小程序后台页面，在左侧的菜单中可以看到“设置”选项，如图 11-1 所示。

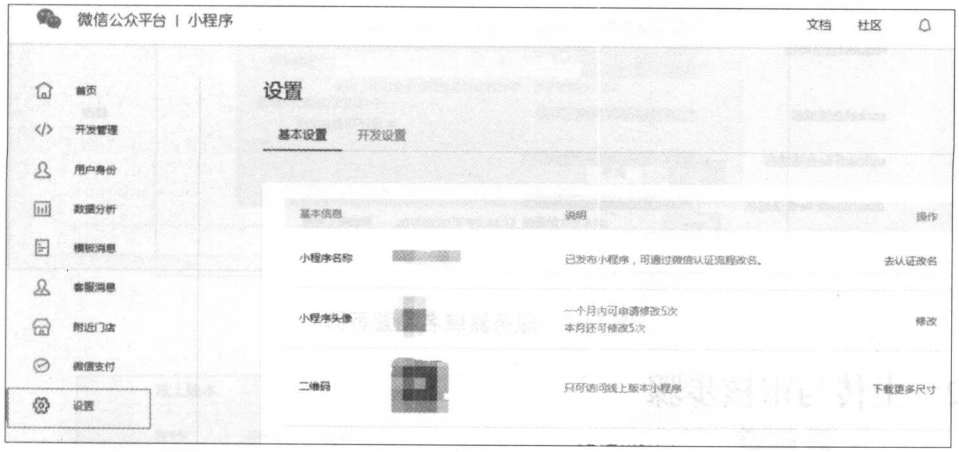

图 11-1 后台设置页面

点击“设置”选项，右边切换到设置页面，选择上面的开发设置，如图 11-2 所示。

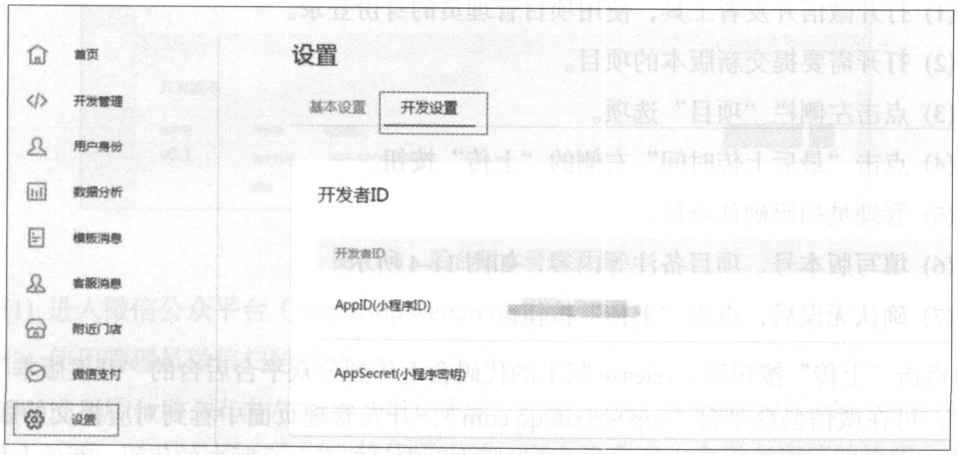

图 11-2 开发设置页面

可以看到服务器域名修改（注：修改域名的时候如果以 / 结尾，那么在程序中也必须写上，否则无法调用），如图 11-3 所示。

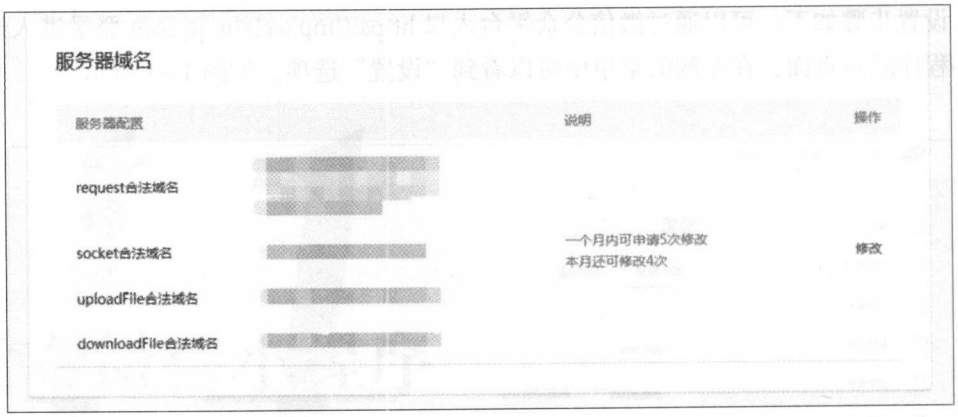

图 11-3 服务器域名设置页面

11.2 上传与审核步骤

上传代码

当微信小程序开发完成之后，开发者需要做的是将 release 版本代码上传至小程序在公众平台的后台（注意：只有管理员有权限上传，开发者没有权限上传。），操作步骤如下。

(1) 打开微信开发者工具，使用项目管理员的身份登录。

(2) 打开需要提交新版本的项目。

(3) 点击左侧栏“项目”选项。

(4) 点击“最后上传时间”右侧的“上传”按钮。

(5) 管理员扫码确认身份。

(6) 填写版本号、项目备注等内容，如图 11-4 所示。

(7) 确认无误后，点击“上传”按钮。

点击“上传”按钮后，release 版本的代码会上传到公众平台后台的“开发版本”一项中，可在微信公众平台（mp.weixin.qq.com），开发管理页面中看到对应提交的版本，如图 11-5 所示。需要注意的是，在“开发版本”中只能有一个版本的代码，新的上传操作会替代之前已存储在“开发版本”中的代码。

项目审核

当项目代码上传成功后，我们就可以把产品提交审核了，操作步骤：

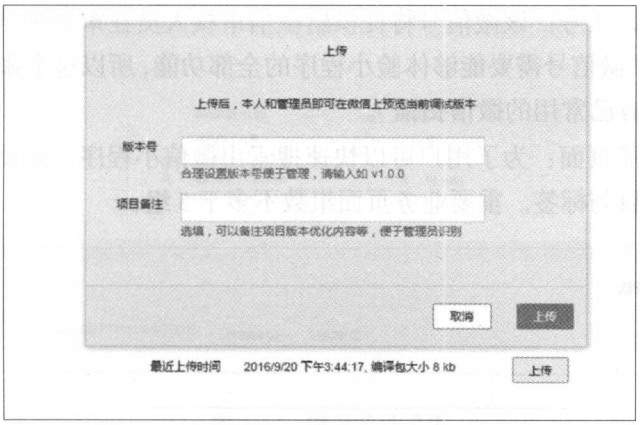

图 11-4 上传功能页面

图 11-5 后台版本管理页面

(1) 进入微信公众平台（http://mp.weixin.qq.com）登录。

(2) 使用管理员微信扫码验证。

(3) 进入后台后，点击左侧的“开发管理”按钮。在这个页面中有一个“开发版本”一项，也就是刚才我们上传代码来的。在右边会有一个提交审核的按钮，点击提交审核按钮，使用管理员账号扫码验证后进入到审核配置页面，在这里需要我们填写一些信息，如图 11-6 所示。全部填写完成后，直接点击“提交审核”按钮即可，在开发管理页中审核版本模块展示审核进度。

- 绑定测试账号：该微信号将提供给微信审核人员在审核微信小程序时登录使用，测试微信号需要能够体验小程序的全部功能，所以这个账号是选填的（注：不要用自己常用的微信扫描）。
- 配置功能页面：为了用户可以快速搜索出微信小程序，需要填写重要业务页面的类目与标签。重要业务页面组数不多于 5 组。

图 11-6 填写审核资料页面

11.3 微信小程序数据分析

微信小程序数据分析是面向小程序开发者、运营者的数据分析工具，这里提供了关键指标统计、实时访问监控、自定义分析等，便于小程序产品的迭代优化和运营。其主要功能如下。

(1) 概况：提供小程序关键指标趋势以及 top 页面访问数据，让我们快速了解微信小程序发展概况，如图 11-7 所示。

(2) 访问分析：提供微信小程序用户访问来源、规模、频次、时长、深度以及页面详情等数据，具体分析用户新增和活跃情况，如图 11-8 所示。

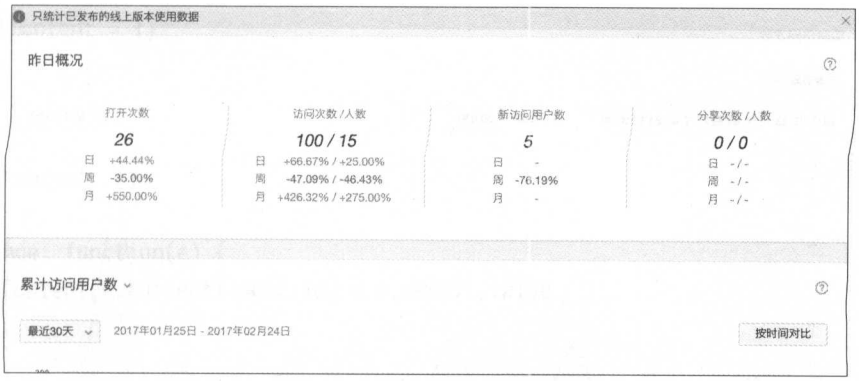

图 11-7 概况统计页面

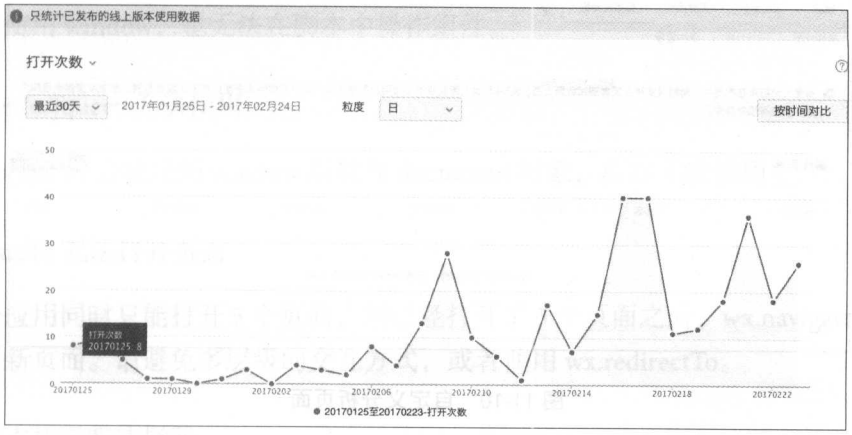

图 11-8 访问分析页面

(3) 实时统计：提供微信小程序实时访问数据，满足实时监控需求，如图 11-9 所示。

(4) 自定义分析：配置自定义上报，精细跟踪用户在微信小程序内的行为，结合用户属性、系统属性、事件属性进行灵活多维的事件分析和漏斗分析，满足微信小程序的个性化分析需求，如图 11-10 所示。

(5) 留存分析：提供微信小程序新增用户和活跃用户的留存数据，分析用户留存与流失（功能正在开发中）。

(6) 用户画像：提供微信小程序的用户画像数据，包括用户地域、性别、平台类型、设备、网络类型等（功能正在开发中）。

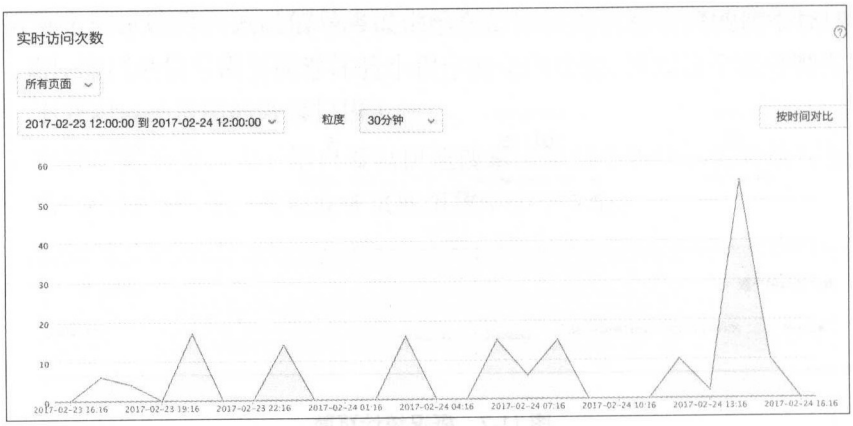

图 11-9 实时统计页面

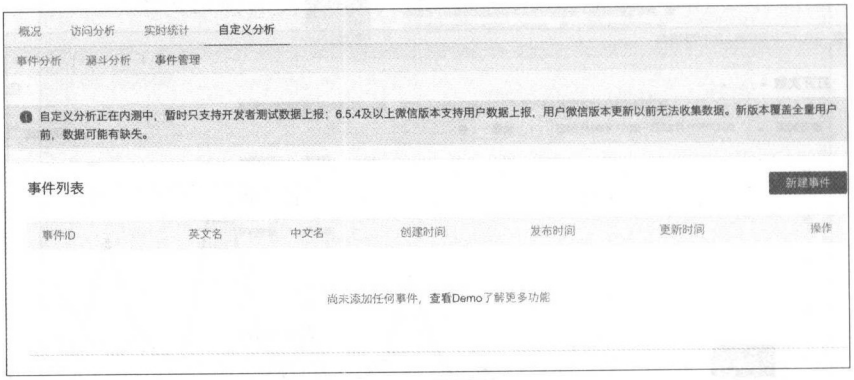

图 11-10 自定义分析页面

11.4 常见问题及注意事项

不能直接操作 Page.data

为避免直接对 Page.data 进行赋值修改，使用 Page.setData 进行操作才能将数据同步到页面中进行渲染。

怎么获取用户输入

要想获取用户输入的组件，需要使用组件的属性 bindchange 将用户的输入内容同步到 AppService。

```
<input id="myInput" bindchange="bindChange" />
<checkbox id="myCheckbox" bindchange="bindChange" />
```

```
var inputContent = {}

Page({
  data: {
    inputContent: {}
  },
  bindChange: function(e) {
    inputContent[e.currentTarget.id] = e.detail.value
  }
})
```

为什么脚本内不能使用 window 等对象

页面的脚本逻辑是在 JsCore 中运行，JsCore 是一个没有窗口对象的环境，所以不能在脚本中使用 window，也无法在脚本中操作组件。

为什么 zepto/jquery 无法使用

zepto/jquery 会使用到 window 对象和 document 对象，所以无法使用。

wx.navigateTo 无法打开页面

一个应用同时只能打开 5 个页面，当已经打开了 5 个页面之后，wx.navigateTo 不能正常打开新页面。请避免多层级的交互方式，或者使用 wx.redirectTo。

样式表不支持级联选择器

WXSS 支持以“.”开始的类选择器。如：

```
.normal_view {
  color: #000000;
  padding: 10px;
}
```

可以使用标签选择器，控制同一类组件的样式。如：使用 input 标签选择器控制 <input /> 的默认样式。

```
input {
  width: 100px;
}
```

本地资源无法通过 css 获取

background-image：可以使用网络图片、base64，或者使用 <image /> 标签。

如何修改窗口的背景色

使用 page 标签选择器，可以修改顶层节点的样式。

```
page {
  display: block;
  min-height: 100%;
  background-color: red;
}
```

为什么代码上传不成功

为了提升体验流畅度，编译后的代码包需要小于 1MB，大于 1MB 的代码包将上传失败。

为什么 HTTPS 请求不成功

tls 仅支持 HTTPS 1.2 及以上版本。

网络请求的 referer 为什么不可以设置

网络请求的 referer 是不可以设置的，格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html，其中 {appid} 为微信小程序的 Appid，{version} 为小程序的版本号，版本号为 0 表示为开发版。

为什么 map 组件总是在最上层

map、canvas、video、textarea 是由客户端创建的原生组件，原生组件的层级是最高的，所以页面中的其他组件无论设置 z-index 为多少，都无法盖在原生组件上。原生组件暂时还无法放在 scroll-view 上，也无法对原生组件设置 css 动画。

11.5 微信小程序审核不通过原因整理汇总

提交微信小程序之后，微信团队会对微信小程序进行审核。如果审核不通过，则开发者需要根据微信团队反馈回来的信息更改自己的微信小程序，然后再次提交审核。为

了缩短这个过程，下面总结了自微信小程序发布以来，开发者们在网上分享的审核不通过的原因：

(1) 微信小程序简介没有介绍小程序功能。

(2) 类目与页面提供的内容不一致，例如选择了富文本-小说，但是添加了新闻类的资讯。

(3) 微信小程序提供的服务和内容必须是正式的，不能以测试内容提交。

(4) 含有声音视频内容，请补充相关对应类目。

(5) 首页图片与文字有互相重叠，建议优化。

(6) 搜索框在极少数情况下才能选中，页面评论点击无响应，页面图片分辨率尺寸失真。

(7) 部分图片显示被压缩，用户体验不好。

(8) 有账号体系的微信小程序，除自有登录方式，必须支持微信授权登录。

(9) 必须登录才能使用的服务，请提供测试账号。

(10) 存在虚拟物品在线交易，iOS 系统需要走 IAP。

(11) 微信小程序服务类目所对应的页面中的核心内容必须与该类目一致。

(12) 必须保证用户在该页面中能使用该服务类目，不得隐藏，不得进行多次跳转。

(13) 不得展示和推荐第三方小程序。例如：不能做小程序导航，不能做小程序链接互推，小程序排行榜等。

(14) 在微信小程序的页面内容中，不得存在诱导类行为，包括但不限于诱导分享、诱导添加、诱导关注公众号、诱导下载等，要求用户分享、添加、关注或下载后才可操作的微信小程序，含有明示或暗示用户分享的文案、图片、按钮、浮层、弹窗等的小程序，通过利益诱惑诱导用户分享、传播的微信小程序，用夸张言语来胁迫、引诱用户分享的微信小程序，强制或诱导用户添加微信小程序的，都将会被拒绝。

(15) 禁止视频、音乐、语音等多媒体的自动播放。

(16) 如果微信小程序有账户系统，必须提供能正常使用且易于发现的“退出”账户选项。

(17) 微信小程序的内容不能包含赌博、竞猜和抽奖等。

(18) 微信小程序的页面内容中，不能存在测试类内容；示例：算命、抽签、星座运势等。

11.6 本章小结

介绍到这里，本书所有的内容你都学完了，现在你已经成功毕业，并且成为了一名合格的微信小程序开发者。但是如果想要成为一名出色的微信小程序开发者，光靠本书中这些理论知识以及少量的实践还是不够的，你需要真正在工作岗位中，通过更多的项目实战来不断地历练和提升自己。

附录 A

从微信的最初设想“应用号”到今天的“小程序”，微信官方都做了哪些动作呢？下面让我们来看看小程序的时间线：

- 2016 年 1 月 9 日，微信团队在一次内部会议上首次提出“应用号”这一设想。
- 2016 年 1 月 11 日，在 2016 年微信公开课 PRO 版上，张小龙意外现身并发表演讲，表示微信的本意是做一个提供服务的平台，所以微信团队专门拆出了服务号，但服务号以提供服务为主，基于一个诉求，效果并不理想，因此“我们将开发一个新的形态，叫作应用号。”
- 2016 年 1 月 20 日，微信发布自己的网页设计样式库 WeUI，开发者可以使用它快速开发出符合微信 UI 界面标准的网页。最初版本的 WeUI 包含了移动应用中所能使用到的所有元素，包括按钮、表单、Toast 提示、对话框、进度条等多种预置样式。WeUI 控件样式库奠定了小程序设计规范的基础。
- 2016 年 4 月 19 日，微信更新“微信 Web 开发者工具”至 0.5.0 版本，该版本的开发者工具支持远程调试。同时，微信宣布 Android 版微信内置浏览器全面升级为 X5 Blink 内核。微信官方宣称，这种内核可以让微信内浏览器具有更好的 HTML5/CSS3 支持、强大的渲染能力，同时提供了硬件状态检测功能。
- 2016 年 9 月 22 日凌晨，微信悄然将“应用号”更名为“小程序”，微信公众平台开始陆续对外发送小程序内测邀请。在首批开放的 API 接口中，小程序具有基本的视觉组件、页面导航、画布渲染、网络访问、文件处理、数据缓存、位置数据、设备信息、界面控制、用户体系等能力。同时张小龙在朋友圈披露其对小程序的定义——小程序是一个不需要下载安装就可使用的应用，它实现了应用触手可及的梦想，用户扫一扫或者搜一下即可打开应用。也体现了用完即走的理念，用户不用关心是否安装太多应用的问题。应用将无处不在，随时可用，但又无需安装卸载。”

- 2016 年 10 月 27 日，微信小程序官方 IDE“微信开发者工具”更新，宣布为小程序新增 19 种 API 接口。新增的 19 中接口中，包括增强的交互反馈功能、数据与文件管理、地理位置选择、获取图片信息、绘图、音频媒体、页面栈处理等逻辑接口，以及包括<textarea />在内的多种视觉元素组件等。
- 2016 年 11 月 3 日，微信小程序宣布进入公测状态。在公测期间，所有企业、政府、媒体和其他机构都可以登记注册小程序。在公测期间，开发者可以将小程序提交审核，但审核通过的小程序暂时不能公开使用。
- 2016 年 11 月 18 日，张小龙发布朋友圈照片，照片中有一台手机。这台手机的屏幕显示着一个摆满应用图标的桌面。从屏幕上的应用名称，到图片附文“程序猿的一小步，程序的一大步”来看，微信小程序在 Android 设备上极有可能以原生应用的形态存在。
- 2016 年 11 月 22 日，微信小程序官方 IDE“微信开发者工具”更新，增强了网络调试的体验。在新版本的开发者工具中，后端开发者可以在开发期间，专注于开发工作本身，无须考虑环境搭建问题，进一步增强了网络调试的体验。同时，开发者也可以在工具中查看到白名单域名，提高开发效率。
- 2016 年 12 月 28 日，在广州亚运城的 2017 年微信公开课上，张小龙再次现身演讲，解答了外界关于小程序的几大疑惑，包括没有商店、没有微信入口、无法推送消息等，张小龙同时透露，希望小程序“在 2017 年 1 月 9 日能被用户使用”。
- 2016 年 12 月 30 日，微信公众平台发布公告称，为便于商户线下推广，上线小程序二维码，最多可生成带参数二维码 10000 个。
- 2017 年 1 月 9 日凌晨，微信小程序正式上线。
- 2017 年 3 月 27 日晚，微信公众平台发布消息称，为增强小程序能力，扩大小程序使用场景，现新增个人开发者申请小程序、自定义菜单打开小程序、模板消息打开小程序、公众号关联小程序给粉丝下发通知等 6 大能力。

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

本书内容

第一部分
认识微信小程序
- 微信小程序介绍
- 体验微信小程序

第二部分
读懂每一行代码
- JavaScript语法
- 熟练掌握WXML和HTML

第三部分
精通微信小程序开发
- 微信小程序开发基础
- Flexbox布局
- 组件的开发应用
- API接口
- 组件进阶

第四部分
自己动手开发微信小程序
- 综合案例——音乐播放小程序
- 发布微信小程序

本书读者

√ 移动平台开发人员

√ JavaScript开发人员

√ 有编程经验想转行做微信小程序的开发人员

√ 对微信小程序开发有兴趣的人员

代码下载

本书配套源代码的下载地址：https://pan.baidu.com/s/1nvyjsQt，密码：wqtr；

或者http://www.broadview.com.cn/ 31314

博文视点Broadview

新浪微博 weibo.com
@博文视点Broadview

责任编辑：王　静
封面设计：王　乐

欢迎投稿：wangj@phei.com.cn

上架建议：程序设计

ISBN 978-7-121-31314-1

定价：69.00元