精通 Android Studio

毕小朋 / 编著

Android开发者必备利器

资深Android开发工程师根据新版Android Studio 2.2精心打造

详解 Android Studio 实用功能与使用技巧
全面、**系统**、**专业**，示例丰富

清華大學出版社

精通 Android Studio

毕小朋 / 编著

清華大学出版社
北 京

内容简介

本书以通俗易懂的语言全面系统地介绍了 Android Studio 实用工具和操作技巧，场景明确，步骤清晰，图文结合。全书共 16 章。第 1 章对 Android Studio 做概要的介绍，从特性、界面、工具、功能到环境搭建和常用的偏好设置，让读者对 Android Studio 有一个全面的认识。第 2 章介绍了项目、文件和资源的创建。第 3 章主要介绍可视化布局编辑器的使用，让读者学会如何快速创建布局文件。第 4 章介绍项目、文件、IDE 和代码的管理技巧。第 5~7 章对代码编辑、视图、导航操作技巧进行了全面的介绍。第 8 章介绍了代码生成、活动模板、自动补全、代码格式化。第 9 章介绍了代码检查工具的使用。第 10 章介绍了如何快速重构。第 11 章介绍了如何使用 Gradle 进行配置、编译和构建应用程序。第 12 章主要介绍了如何配置、运行和调试应用程序。第 13 章介绍了集成进 Android Studio 的各种实用工具的使用。第 14 章以 Git 为例介绍了版本控制系统的使用。第 15 章介绍了窗口和标签的管理。第 16 章读者将学会如何配置一个个性化、高效、符合自己开发习惯的 IDE。

本书并不是一本循序渐进的学习书籍，而更像是一本 Cookbook，读者需要有目的地去阅读。当遇到问题或者想了解某个工具如何使用时，可直接定位到相关的章节进行阅读。

本书适用于 Android 开发、测试以及想学习 Android 开发的相关从业人员，适合放在床头案边时常翻阅。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

精通 Android Studio/毕小朋编著. —北京：清华大学出版社，2017
（移动开发丛书）
ISBN 978-7-302-45530-1

Ⅰ. ①精… Ⅱ. ①毕… 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（2016）第 272704 号

责任编辑：王金柱
封面设计：王　翔
责任校对：闫秀华
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**36.25　　**字　　数：**928 千字
版　　次：2017 年 1 月第 1 版　　**印　　次：**2017 年 1 月第1 次印刷
印　　数：1～3000
定　　价：89.00 元

产品编号：071059-01

前　言

人总是在不断地探索和尝试，总是发明出一些新的科技来改善我们的生活。为了风雨无阻，人类发明了汽车，为了自由翱翔，人类发明了飞机。人们总是朝着更快、更高、更远的目标奋进。科技的进步让我们的工作变得更加高效，于是我们有了更多的时间来思考，思考如何使用技术让这个世界变得更加美好。

现如今，移动互联网正值浪潮之颠，物联网、大数据、云计算、虚拟现实（VR）、增强现实（AR）等，新技术不断涌现又将会带来新一轮的裂变，而你我有幸处在这样一个充满机遇和创意的时代。作为程序员的我们，作为移动互联网产品的开发者，应该要有更多的时间学习和思考，需要把更多的时间和精力放在产品的设计和创新上。那些简单的、重复的、有规律的、易出错的编码和测试工作，都应该由工具来帮我们完成。

Android Studio正是这样一款Android开发者们梦寐以求的工具，它的诞生就是为了让Android开发变得更加简单和高效。

含着金汤勺出生的Android Studio，由于继承自Intellij IDEA这个号称当前最好最强最智能的Java IDE，天然地拥有了智能和效率上的优势。加上Google为其量身定做的Android开发工具，让Android Studio成为了Android项目开发和测试的必备神器。

从 2013 年 5 月 16 日Google在I/O大会上发布Android Studio 第 1 个预览版本开始，到现在 2.2 版本的发布。Google对它的增强和优化从未间断过，很多实用的功能被不断地集成进来，每一次重大的更新都会使开发者欢呼雀跃。我对Android Studio的学习和使用也从未间断过，感觉它就像一座需要不断探索和挖掘的宝藏，每次不经意地获得一项新的技能都会让我感慨它的奇妙。

正所谓“闻道有先后，术业有专攻”。我采访过很多资深的开发人员，他们平日里更多地会去研究和学习Android开发技术，不会有太多的时间去探究工具的使用。目前中国市场上关于Android开发的书籍有很多，但是至今还没有一本专门介绍Android Studio的书，机缘巧合，我做了第一个吃螃蟹的人，希望能够通过这本书把我所学习到的一些实用的工具和技能介绍给大家，让大家体会到Android Studio的强大和便捷之处。

于是 2013 年 8 月 3 日，中国第一本《Android Studio入门指南》在杭州市滨江区星光大道的星巴克诞生了，一时间霸占了我厂头条，然后迅速在网上传播开来。

2015 年 4 月 29 日，又是在同一家星巴克，我开始动笔编写《Android Studio实用指南》。

2015 年 5 月 5 日，《Android Studio实用指南》写满一万字在百度阅读上架，被读者@萌萌番长购买。

2016 年 5 月 30 日，与清华大学出版社签约，书名改为《精通Android Studio》。

经过3年的积累，15个月的努力，200多次的修改，1000多位读者的支持和监督，近70万字的《精通Android Studio》最终得以出版。激动和感激之情无以言表，感谢大家，感谢这个伟大的、一切皆有可能的互联网时代。

本书定位

本书并不是一本循序渐进的学习书籍，它更像是一本Cookbook，你需要有目的地去阅读本书。当你遇到问题或者想了解某个工具如何使用时，可以直接定位到相关的章节。

本书像是一本非常实用的指导手册，它几乎囊括Android Studio所有的实用功能和操作技巧，适合放在读者的电脑旁经常翻阅。

本书以通俗易懂的语言描述工具和使用技巧，并且每个操作都有实例演示，让读者感觉是在跟一个有经验的人聊天。

本书以近1500张图片详细描述Android Studio的使用，是真正的图文并茂。

本书以解决问题为目的，讲述如何使用工具解决实际问题。

本书专注于操作技巧的讲解，对于Android开发的基础知识略有提及，但不是本书的重点。

本书以macOS上的操作为例进行演示，不同操作系统上Android Studio的操作差异不大，对于快捷键会区分macOS/Windows/Linux。

本书的大部分操作技巧同样适用于IntelliJ IDEA。

目标读者

- 如果你初学Android开发；
- 如果你想从Eclipse转到Android Studio；
- 如果你从其他语言转到Android开发；
- 如果你想深入了解Android Studio；
- 如果你想深入了解IntelliJ IDEA；
- 如果你从事Android测试开发；
- 如果你英语不好；
- 如果你想节省搜索的时间；
- 如果你想提高工作效率；

那么本书就是为你量身定做的！

读者须知

关于开发环境

- 本书适用于macOS/Windows/Linux操作系统，主要以macOS来演示。
- 本书以JDK1.7为例介绍如何配置开发环境，如果你使用的是Android Studio 2.2及以上版本，需要配置JDK1.8。

- 由于Android Studio界面变化太快，因此本书中有些图片可能是老版本的界面，但是丝毫不会影响理解和阅读。

关于快捷键

- 本书所有操作实例均使用Android Studio默认快捷键。
- 书中所有快捷键都包括macOS、Windows、Linux。
- macOS快捷键（F1~F12）使用时都需要同时按下Fn键。

名词解释

本书中用到的名词、缩写、字符等统一为如下解释：

名词	解释
macOS	苹果 macOS 系统
Windows	Windows 系统，默认以 Windows 7 作为演示
Linux	Linux 系统，默认以 Ubuntu 作为演示
AS	Android Studio
IDE	集成开发工具
APK	Android 安装包
APP	Android 应用程序
设备	指真机或模拟器
真机	真实的 Android 手机设备
模拟器	Android 虚拟机
偏好设置	macOS 上的设置叫偏好设置，Windows/Linux 上对应的是设置
Preferences	macOS 上的 Preferences 对应 Windows/Linux 上的 Settings
小贴示	一些提示信息和注意事项

勘误与反馈

本书已经尽可能全面地适配了当前最新的 2.2 版本，但是由于IntelliJ IDEA在持续更新，Android Studio也在持续更新，再加上笔者能力有限，书中难免会有遗漏、理解错误或者表达不清晰的地方。如果你在阅读本书时发现了这些问题，请发邮件（wirelessqa@163.com）或者直接提交BUG到Github：https://github.com/bxiaopeng/AndroidStudio/issues。

本书的所有勘误及版本适配都会更新在Github上。

致谢

好多次，在出门前，儿子都会问“爸爸，你跟我们一起去吗？”或者问“爸爸，你去哪里？”妈妈会抢着说“爸爸加班，晚上陪你玩”。现在儿子已经快 3 岁了，有时候会问“爸爸怎么老是加班啊”，我总是说“因为爸爸忙啊”。

写书不是一件轻松的事情，这本书几乎用掉了我所有的休息时间。每个礼拜天我都会雷打不动地在星巴克写啊写，如果没有家人的理解和支持，没有老婆和儿子莫大的支持和鼓励，我是无法完成这本书的，感谢他们的付出。

感谢楼主、友哥、猴哥、炜哥、战老师对本书的校对，他们都是非常资深的Android开发工程师，感谢他们对本书提出的修改建议。

感谢已经买了电子书的 1000 多位读者，他们见证了这本书的的成长，没有他们的支持和鼓励，我也没有信心出版这本书。

感谢所有的读者，感谢你对本书的关注。

毕小朋

2016 年 9 月 22 日下午改于双城国际的星巴克

目 录

第 1 章 初识 Android Studio……1
1.1 什么是 Android Studio……1
1.1.1 Android Studio 简介……1
1.1.2 系统要求……2
1.1.3 下载地址……2
1.1.4 为什么要用 Android Studio……2
1.2 Android Studio 的特点……3
1.3 macOS 环境配置……7
1.3.1 配置 JDK……7
1.3.2 下载 Android Studio……8
1.3.3 配置 Android 的环境变量……8
1.4 Windows 环境配置……9
1.4.1 配置 JDK……9
1.4.2 配置 Android Studio……10
1.4.3 配置 Android 的环境变量……11
1.5 Linux 环境配置……12
1.5.1 配置 JDK……12
1.5.2 配置 Android Studio……12
1.5.3 配置 Android 的环境变量……13
1.6 认识欢迎界面……13
1.6.1 最近打开的项目……14
1.6.2 开始一个项目……14
1.7 认识配置界面……15
1.8 帮助和教程……19
1.9 更新 Android Studio 版本……22
1.10 配置 Android Studio 更新通道……23
1.10.1 在偏好设置中配置更新通道……23
1.10.2 在检测结果对话框中配置更新通道……23
1.10.3 单独下载最近更新的版本……23
1.10.4 四种版本的区别……24
1.11 在 Android Studio 中使用代理……25
1.12 认识偏好设置……26
1.12.1 基础配置……27
1.12.2 个性化配置……28

1.13 认识工作台 ······ 30
1.14 认识工具窗口 ······ 33

第 2 章 项目与模块 ······ 37

2.1 Android Studio 的项目结构 ······ 37
2.1.1 项目和模块 ······ 37
2.1.2 基本的项目结构 ······ 38
2.2 导入项目和模块 ······ 40
2.2.1 导入 Android Studio 项目 ······ 40
2.2.2 导入 Eclipse 项目 ······ 40
2.2.3 导入 Android 示例代码 ······ 44
2.2.4 导入模块 ······ 44
2.2.5 导入.JAR/.AAR ······ 46
2.2.6 从 VCS 检出项目 ······ 47
2.3 创建项目和模块 ······ 49
2.3.1 创建项目 ······ 49
2.3.2 创建应用程序模块 ······ 53
2.3.3 创建 Android 公共库模块 ······ 55
2.4 删除模块 ······ 57
2.5 添加 so 文件 ······ 58
2.6 创建类和文件 ······ 59
2.7 创建 Activity ······ 61
2.7.1 Activity 模板列表 ······ 61
2.7.2 Activity Gallery ······ 61
2.7.3 新建一个 Activity ······ 62
2.8 创建 Fragment 文件 ······ 64
2.9 创建 Service 文件 ······ 65
2.9.1 创建 Service 文件 ······ 66
2.9.2 创建 Intent Service 文件 ······ 66
2.10 创建自定义组件 ······ 67
2.11 创建 App Widget ······ 69
2.12 创建可编译的资源文件 ······ 70
2.12.1 可编译的资源文件 ······ 70
2.12.2 创建可编译的资源文件 ······ 72
2.12.3 资源限定符 ······ 74
2.13 创建不同分辨率的图标 ······ 77
2.13.1 启动图标 ······ 78
2.13.2 活动栏和选项卡图标 ······ 79
2.13.3 通知图标 ······ 80
2.14 创建矢量图 ······ 81
2.14.1 使用定义好的素材图标 ······ 81
2.14.2 使用本地的 SVG 文件 ······ 81
2.15 创建 AIDL 文件 ······ 82
2.16 创建 Android 文件夹 ······ 83
2.17 创建 Resource Bundle 文件 ······ 84

第 3 章 布局 ………… 86

3.1 认识布局 ………… 86
3.1.1 Android 中定义布局的方法 ………… 86
3.1.2 快速开始 ………… 87
3.2 设计布局 ………… 88
3.2.1 文本编辑器 ………… 88
3.2.2 可视化布局编辑器 ………… 89
3.3 组件列表 ………… 90
3.4 预览 ………… 94
3.4.1 设置控件属性 ………… 94
3.4.2 警告和错误提示 ………… 94
3.4.3 界面缩放 ………… 96
3.4.4 控件操作 ………… 96
3.5 结构树 ………… 96
3.5.1 快速转换布局属性 ………… 97
3.5.2 选择控件 ………… 97
3.5.3 跳到源码 ………… 97
3.6 属性 ………… 97
3.7 工具栏 ………… 99

第 4 章 管理 ………… 103

4.1 项目窗口 ………… 103
4.1.1 视图模式 ………… 103
4.1.2 常用设置和操作 ………… 105
4.2 项目管理 ………… 110
4.2.1 打开和关闭项目 ………… 110
4.2.2 管理最近打开的项目 ………… 111
4.3 文件管理 ………… 112
4.3.1 文件同步 ………… 112
4.3.2 导出到 HTML ………… 112
4.3.3 切换文件编码方式 ………… 113
4.3.4 切换行分隔符 ………… 113
4.3.5 使文件只读 ………… 114
4.3.6 使用省电模式 ………… 114
4.3.7 打开文件/文件夹所在磁盘目录 ………… 114
4.4 Android Studio 管理 ………… 115
4.4.1 隐藏/显示/退出 Android Studio ………… 115
4.4.2 清除缓存/重启 Android Studio ………… 116
4.5 收藏夹 ………… 116
4.5.1 添加到收藏夹 ………… 116
4.5.2 管理收藏夹 ………… 118
4.6 TODO ………… 119
4.6.1 添加 TODO 任务 ………… 119
4.6.2 查看 TODO 任务 ………… 120
4.6.3 TODO 工具窗口常用操作 ………… 120

4.6.4 设置 TODO……122
4.6.5 使用 TODO 分配代码 Review 任务……123

第 5 章 编辑……126

5.1 撤消/重做/剪切/复制/粘贴……126
5.2 复制技巧……127
5.2.1 复制为纯文本……127
5.2.2 复制引用……127
5.2.3 从复制历史中选择粘贴……127
5.2.4 设置粘贴历史记录的个数……128
5.2.5 复制行……128
5.3 合并两行内容……129
5.4 选择技巧……129
5.4.1 扩大选择范围……129
5.4.2 缩小选择范围……130
5.4.3 使用列选择模式……130
5.5 缩进设置……131
5.6 自动补全当前的语句……131
5.7 一键切换大小写字母……132
5.8 查找工具栏……132
5.8.1 打开查找工具栏……132
5.8.2 快速查找……133
5.8.3 查找范围设置……133
5.9 在查找结果中跳转……134
5.10 选择查找结果……134
5.11 指定查找路径……135
5.12 替换……136
5.13 指定替换路径……137
5.14 在结构中查找和替换……139
5.15 查找用法……141
5.16 设置查找用法的过程和范围……141
5.17 显示用法……144
5.18 查看在当前文件中的用法……144
5.19 在文件中高亮显示字符……145
5.20 最近查找……145
5.21 Macros（宏）……146

第 6 章 视图……148

6.1 工具窗口……148
6.1.1 显示/隐藏工具窗口……148
6.1.2 快速切换工具窗口……149
6.2 工作台管理……150
6.3 查看定义……151
6.4 查看同胞元素……152
6.5 查看文档……152

6.6 查看方法的参数信息 …… 154
6.7 查看表达式的类型 …… 155
6.8 查看上下文信息 …… 155
6.9 查看源码 …… 156
6.10 查看最近打开过的文件 …… 156
6.11 查看最近改动过的文件 …… 156
6.12 查看最近的改动 …… 156
6.13 对比任意文件 …… 158
6.14 将选中的文件和正在编辑的文件进行对比 …… 160
6.15 将选中的文件和剪切板上的内容进行对比 …… 160
6.16 切换编辑器配色方案 …… 161
6.16.1 快速切换编辑器配色方案 …… 161
6.16.2 切换编辑器配色方案 …… 161
6.17 切换代码风格 …… 161
6.18 切换键盘映射 …… 162
6.19 快速切换视图模式 …… 163
6.20 快速切换主题 …… 164
6.21 设置编辑器是否显示空格 …… 164
6.22 设置编辑器是否显示行号 …… 165
6.23 设置编辑器是否显示缩进向导 …… 166
6.24 设置编辑器是否使用自动换行 …… 167
6.25 设置编辑器是否弹出导入提示 …… 168
6.26 使用演示模式 …… 169
6.27 使用免打扰模式 …… 170
6.28 使用全屏模式 …… 171

第 7 章 导航 …… 172

7.1 搜索并打开类文件 …… 172
7.2 搜索并打开某个文件 …… 173
7.3 搜索并打开某个文件或方法 …… 174
7.4 使用自定义代码块 …… 174
7.5 快速跳转到某一行代码 …… 176
7.6 快速跳转到光标的历史位置 …… 177
7.7 快速跳转到编辑过的历史位置 …… 178
7.8 标记书签 …… 178
7.9 使用助记符标记书签 …… 179
7.10 管理书签 …… 179
7.10.1 在书签管理界面管理书签 …… 180
7.10.2 在收藏夹中管理书签 …… 181
7.11 快速跳转到导航栏 …… 181
7.12 快速跳转到声明 …… 181
7.13 快速跳转到实现 …… 182
7.14 快速跳转到类型声明 …… 182
7.15 快速跳转到父类 …… 183

7.16 类和测试类之间快速跳转 ……184
7.17 查看相关联的文件 ……186
7.18 查看文件结构 ……187
7.19 查看类的层次结构图 ……188
7.20 查看方法类型的层次结构 ……189
7.21 查看方法调用层次结构 ……190
7.22 快速跳转到错误代码的位置 ……190
7.23 在方法间前后跳转 ……191
7.24 使用翻页功能 ……191
7.25 选择当前文件在哪里显示 ……192
7.26 光标快速跳转到编辑器 ……192
7.27 光标快速跳转到页首/页尾 ……193

第 8 章 编码 ……194

8.1 覆写或实现方法 ……194
8.2 实现接口方法 ……196
8.3 实现代理方法 ……196
8.4 生成构造函数 ……199
8.5 生成 Getter 和 Setter 方法 ……200
8.6 覆写 equals 和 hashcode 方法 ……200
8.7 覆写 toString 方法 ……201
8.8 插入版权信息 ……203
8.8.1 插入版权信息 ……203
8.8.2 配置版权信息 ……204
8.8.3 共享版权信息配置 ……207
8.9 提取或删除代码 ……208
8.10 自动补全提示 ……209
8.11 代码补全 ……210
8.12 补全循环扩展词 ……212
8.13 展开或折叠代码 ……213
8.13.1 展开或折叠当前代码块 ……213
8.13.2 展开或折叠当前代码块中的所有子模块 ……213
8.13.3 展开和折叠全部代码块 ……214
8.13.4 展开和折叠当前文件中的所有注释 ……214
8.13.5 指定展开层级 ……214
8.13.6 展开和折叠选中区域 ……215
8.13.7 折叠代码片段 ……215
8.14 插入代码模板 ……215
8.14.1 类中常用的缩写 ……215
8.14.2 方法中常用的缩写 ……216
8.15 使用代码模板包裹代码 ……218
8.16 查看和编辑代码模板 ……219
8.17 使用常用代码模板包裹代码 ……220
8.18 注释代码 ……222

8.19 格式化代码 …… 222
8.20 自动缩进行 …… 224
8.21 优化导入 …… 224
8.22 重新排列代码 …… 225
8.23 移动一段代码 …… 225
8.24 移动一行代码 …… 226
8.25 操作意图提示 …… 227
8.26 正则表达式操作意图提示 …… 228
第 9 章 检查 …… 229
9.1 代码检查工具 …… 229
9.2 全面了解 Lint …… 230
9.2.1 Lint 是什么 …… 230
9.2.2 为什么要用 Lint …… 230
9.2.3 Lint 会检查哪些错误 …… 230
9.2.4 Lint 工作流程 …… 231
9.2.5 报告中的 Issue 和 Category …… 232
9.2.6 Lint 使用场景 …… 232
9.2.7 如何配置 Lint 检查 …… 233
9.2.8 Lint 命令行用法介绍 …… 233
9.2.9 Lint 命令行用法举例 …… 235
9.3 执行一次代码检查 …… 237
9.4 指定检查范围 …… 239
9.4.1 先执行检查再指定范围 …… 239
9.4.2 选定范围再执行检查 …… 239
9.5 代码检查结果及辅助工具 …… 240
9.6 详解代码检查结果辅助工具 …… 242
9.6.1 重新执行代码检查 …… 242
9.6.2 全部展开或折叠检查结果 …… 242
9.6.3 在检查结果中快速上下跳转 …… 243
9.6.4 自动定位到问题的源码 …… 243
9.6.5 导出代码检查结果 …… 243
9.6.6 按严重程度分组排查问题 …… 244
9.6.7 按目录分组排查问题 …… 244
9.6.8 过滤已解决的问题 …… 244
9.6.9 高亮显示不同和仅显示不同 …… 244
9.6.10 快速设置 …… 245
9.6.11 快速解决问题 …… 245
9.6.12 对检查出的问题进行操作 …… 246
9.7 禁用和启用某项检查 …… 247
9.7.1 在检查结果中禁用和启用某项检查 …… 247
9.7.2 在偏好设置中禁用和启用某项检查 …… 248
9.8 忽略检查 …… 249
9.9 在指定范围内执行某项检查 …… 252
9.10 解决检查出的问题 …… 252

9.11 管理代码检查配置文件……253
9.12 配置代码检查规则……255
9.13 Android 类目的所有检查项……259
9.14 Android Lint 类目的检查项……260
9.15 在 lint.xml 文件中配置 Lint 检查……264
9.16 在 Gradle 中配置 Lint 检查……264
9.17 使用 Gradle 执行 Lint 检查……267
9.17.1 命令行执行 Lint 检查……267
9.17.2 Gradle 工具窗口执行 Lint 检查……267
9.18 在 Java 和 XML 源码中配置 Lint 检查……267
9.18.1 在 Java 源码中配置 Lint 检查……267
9.18.2 在 XML 源码中配置 Lint 检查……268
9.19 代码清理……269
9.20 通过名字来指定代码检查项……271
9.21 配置当前文件自动检查的规则……272
9.22 导入并查看离线检查结果……274
9.23 自动添加是否可为空注解……275
9.24 分析依赖……276
9.25 分析反向依赖……277
9.26 分析模块依赖……278
9.27 分析循环依赖……279
9.28 分析数据流……279
9.29 分析堆栈信息……280

第 10 章 重构……282

10.1 重命名……282
10.1.1 重命名类……282
10.1.2 重命名变量……283
10.1.3 重命名文件……283
10.2 更改方法签名……284
10.3 迁移变量类型……284
10.4 转成静态方法……286
10.5 静态方法转为实例方法……287
10.6 移动类……288
10.7 移动静态方法……289
10.8 移动静态字段……290
10.9 复制……291
10.10 安全删除……291
10.11 提取变量……294
10.12 提取常量……295
10.13 提取字段……296
10.14 提取参数……298
10.15 提取函数式参数……300
10.16 提取参数对象……301

10.17 提取方法 …… 303
10.18 提取方法对象 …… 304
10.19 提取委托 …… 304
10.20 提取接口 …… 306
10.21 提取父类 …… 307
10.22 内联方法 …… 309
10.23 内联临时变量 …… 310
10.24 查找并替换重复代码 …… 311
10.25 反转布尔值 …… 312
10.26 把成员拉到父类 …… 313
10.27 把成员推到子类 …… 314
10.28 尽可能使用接口 …… 316
10.29 使用委托替换继承 …… 318
10.30 移除中间人 …… 321
10.31 包装方法返回值 …… 323
10.32 将匿名类转成内部类 …… 325
10.33 封装字段 …… 326
10.34 使用查询替换临时变量 …… 327
10.35 使用工厂方法替换构造方法 …… 329
10.36 使用构建器替换构造方法 …… 330
10.37 泛型化 …… 332
10.38 国际化 …… 333

第 11 章 构建 …… 335

11.1 认识 Gradle …… 335
11.1.1 Gradle 是什么 …… 335
11.1.2 Gradle 中依赖的仓库 …… 336
11.2 配置 Gradle 环境 …… 338
11.3 Gradle Wrapper …… 339
11.4 查看和执行 Gradle 任务 …… 340
11.4.1 查看当前项目支持的 Gradle 任务 …… 340
11.4.2 执行 Gradle 任务 …… 342
11.4.3 常用 Gradle 任务 …… 342
11.4.4 Gradle 工具窗口 …… 343
11.5 构建项目和模块 …… 345
11.5.1 编译项目 …… 345
11.5.2 编译模块 …… 346
11.5.3 设置自动编译项目 …… 347
11.5.4 重新构建项目 …… 347
11.5.5 Make Project 与 Rebuild Project 的区别 …… 348
11.5.6 清理项目 …… 348
11.6 Gradle Script …… 348
11.6.1 Gradlew 配置文件 gradle-wrapper.properties …… 349
11.6.2 项目全局配置文件 settings.gradle …… 350

11.6.3 本地属性配置文件 local.properties ······350
11.6.4 Gradle 配置文件 gradle.properties ······351
11.6.5 代码混淆规则配置文件 proguard-rules.pro ······351
11.6.6 项目构建配置文件 build.gradle ······354
11.6.7 模块构建配置文件 build.gradle ······355
11.7 在项目结构中配置模块构建 ······357
11.7.1 配置应用程序属性 ······357
11.7.2 配置应用程序签名 ······358
11.7.3 配置应用程序特性 ······359
11.7.4 配置应用程序构建类型 ······361
11.7.5 配置应用程序依赖 ······364
11.8 签名和打包 ······365
11.8.1 创建签名证书 ······365
11.8.2 生成签名的 APK ······366
11.8.3 自动打包和签名 ······370
11.8.4 混淆打包 ······371
11.8.5 多渠道打包 ······372
11.9 配置开发者服务 ······377
第 12 章 运行和调试 ······378
12.1 运行和调试配置 ······378
12.1.1 运行和调试配置 ······378
12.1.2 Android 应用程序配置 ······379
12.2 运行应用程序 ······383
12.3 调试应用程序 ······384
12.4 断点 ······388
12.4.1 行断点 ······388
12.4.2 方法断点 ······388
12.4.3 字段观察点 ······389
12.4.4 条件断点 ······390
12.4.5 临时断点 ······390
12.4.6 异常断点 ······391
12.4.7 日志断点 ······391
12.4.8 禁用断点 ······392
12.4.9 断点设置 ······392
12.5 帧调试窗口 ······393
12.6 变量调试窗口 ······394
12.7 监视窗口 ······397
12.7.1 添加变量或表达式到监视窗口 ······398
12.7.2 快捷工具 ······399
12.8 调试控制工具 ······399
12.9 步进调试工具 ······403
12.10 计算表达式 ······405
12.10.1 在堆栈帧中计算表达式或代码片段 ······405
12.10.2 计算任意表达式 ······407
12.10.3 快速计算表达式的值 ······407

12.10.4 选中表达式立即显示表达式的值 ……408
12.11 关联调试到 Android 进程 ……408
12.12 配置和运行单元测试 ……409
12.12.1 配置和运行本地单元测试 ……409
12.12.2 使用命令行运行单元测试 ……412
12.12.3 配置 Android 单元测试 ……412
第 13 章 工具 ……414
13.1 任务 ……414
13.1.1 任务介绍和配置 ……414
13.1.2 打开任务 ……417
13.1.3 创建新任务 ……418
13.1.4 任务变更列表 ……419
13.1.5 切换/关闭任务 ……420
13.1.6 管理上下文 ……420
13.2 JavaDoc ……421
13.2.1 配置 JavaDoc ……421
13.2.2 生成 JavaDoc ……422
13.3 将当前文件保存为模板 ……424
13.4 IDE Scripting Console ……425
13.5 管理 Android SDK ……425
13.5.1 管理 Android SDK 平台 ……426
13.5.2 管理 SDK 开发工具和更新站点 ……428
13.6 管理 Android 模拟器 ……429
13.6.1 认识模拟器 ……429
13.6.2 创建模拟器 ……430
13.6.3 启动模拟器 ……431
13.7 即时运行 ……433
13.8 Android 监视器 ……436
13.9 截图 ……440
13.10 录像 ……440
13.11 捕获系统信息 ……441
13.12 布局解析 ……443
13.13 Logcat 监视器 ……445
13.14 内存监视器 ……450
13.14.1 Dump Java Heap ……452
13.14.2 Allocation Tracking ……456
13.15 CPU 监视器 ……458
13.16 网络监视器 ……460
13.17 GPU 监视器 ……460
13.18 APK 分析器 ……461
13.19 主题编辑器 ……462

第 14 章 版本控制 ······ 466
14.1 版本控制系统 ······ 466
14.2 Git 偏好设置 ······ 469
14.3 配置 GitHub 账户信息 ······ 470
14.4 从 GitHub 克隆代码 ······ 470
14.5 将本地项目共享到 GitHub ······ 471
14.6 查看本地变更历史 ······ 472
14.7 Git 添加文件 ······ 474
14.8 Git 提交变更 ······ 475
14.9 Git 文件逐行追溯 ······ 476
14.10 显示当前修订版本 ······ 477
14.11 Git 文件比较 ······ 478
14.12 Git 撤销操作 ······ 479
14.13 Git 版本回退 ······ 480
14.14 Git 查看提交历史 ······ 481
14.15 Git 分支管理 ······ 483
14.16 Git 创建标签 ······ 486
14.17 Git 合并分支 ······ 487
14.18 解决 Git 合并中的冲突 ······ 488
14.19 Git 使用 Rebase 合并分支 ······ 489
14.20 Git 暂存/恢复暂存变更 ······ 494
14.21 Git 获取最新内容 ······ 496
14.22 Git 合并最新内容 ······ 497
14.23 Git 更新项目 ······ 497
14.24 刷新文件状态 ······ 499
14.25 Git 补丁 ······ 500
14.26 Git 搁置变更 ······ 501
14.27 查看 Git 项目的提交信息 ······ 502
第 15 章 窗口 ······ 504
15.1 最小化和最大化窗口 ······ 504
15.2 保存和恢复窗口布局 ······ 504
15.3 工具窗口的显示和隐藏 ······ 505
15.4 工具窗口的隐藏技巧 ······ 506
15.5 工具窗口调整技巧 ······ 507
15.6 移动工具窗口的位置 ······ 508
15.7 工具窗口的查看模式 ······ 509
15.8 编辑器标签设置 ······ 512
15.9 快速切换编辑器标签 ······ 513
15.10 关闭编辑器标签 ······ 514
15.11 管理编辑器标签 ······ 514
15.12 标签显示位置 ······ 515
15.13 拆分编辑器窗口 ······ 516

15.14 多个项目之间切换 ······ 517

第 16 章 偏好设置 ······ 518

16.1 外观与行为 ······ 518
16.1.1 设置工具提示的延迟时间 ······ 518
16.1.2 设置在状态栏显示内存状态 ······ 519
16.1.3 对菜单选项和工具栏的工具进行增/删改 ······ 519
16.2 系统设置 ······ 519
16.3 键盘映射 ······ 522
16.4 编辑器常规设置 ······ 523
16.4.1 设置单击编辑器光标定位在一行的结尾或定位在单击的位置 ······ 523
16.4.2 设置鼠标悬停在元素上会显示文档提示 ······ 523
16.4.3 设置是否自动换行 ······ 524
16.4.4 设置使用 command+鼠标控制代码的缩放 ······ 526
16.4.5 开启使用驼峰单词 ······ 526
16.4.6 关闭单词拼写检查 ······ 527
16.4.7 设置代码折叠规则 ······ 527
16.5 设置自动导入 ······ 528
16.5.1 设置粘贴时自动导入包 ······ 528
16.5.2 设置自动导入需要的包 ······ 528
16.5.3 设置是否弹出导入提示 ······ 529
16.6 编辑器外观 ······ 530
16.6.1 设置编辑器一直显示行号 ······ 530
16.6.2 设置编辑器显示方法分隔符 ······ 530
16.6.3 设置编辑器显示空格 ······ 530
16.6.4 设置编辑器显示缩进向导 ······ 531
16.7 代码补全 ······ 531
16.7.1 设置自动补全时是否区分大小写 ······ 531
16.7.2 加快自动弹出代码补全提示的速度 ······ 532
16.7.3 关闭自动弹出代码补全提示 ······ 533
16.7.4 设置查看方法参数信息的时候显示方法签名 ······ 533
16.8 文件标签 ······ 534
16.8.1 设置用星号标记修改过的文件标签 ······ 534
16.8.2 设置打开的文件标签可以多行显示 ······ 534
16.8.3 设置文件标签的显示位置 ······ 535
16.8.4 设置文件标签超过一定数量时的关闭规则 ······ 535
16.9 编辑器颜色 ······ 536
16.9.1 设置是否显示条标和条标的显示颜色 ······ 536
16.9.2 设置控制台的颜色 ······ 537
16.9.3 设置控制台的字体 ······ 538
16.9.4 自定义代码的颜色 ······ 539
16.10 代码风格 ······ 541
16.10.1 设置 Java 注释按缩进显示 ······ 541
16.10.2 设置语句不要都显示在一行 ······ 541
16.10.3 设置 Java 简单的类合并为一行 ······ 542
16.10.4 设置 Java 字段和变量列对齐 ······ 542

16.10.5 设置自动生成字段名称时添加前缀······542
16.11 文件和代码模板······543
16.11.1 设置新建文件的注释模板······543
16.11.2 模板中内置的变量······544
16.11.3 设置新建类文件模板······544
16.11.4 设置 IDE 和项目的编码······545
16.11.5 对动态模板进行增删改查······545
16.11.6 设置展开代码的按键······547
16.11.7 给一个文件类型添加匹配规则······547
16.11.8 添加一个自定义的文件类型······547
16.11.9 设置忽略某类文件或文件夹······548
16.12 插件······549
16.12.1 安装插件······550
16.12.2 禁用插件······553
16.12.3 卸载插件······554
16.12.4 常用插件······555
16.13 编译和构建······556
16.13.1 设置 Android Studio 的内存参数······556
16.13.2 设置自动编译项目······556
16.13.3 设置并行编译······557
16.13.4 调整编译内存大小······557
附录 Android Studio 重要版本发布时间线······559
参考资料······560

第 1 章　初识 Android Studio

在学习使用Android Studio之前我们需要了解Android Studio是什么，为什么要用Android Studio以及它的基础配置和操作。

本章从特性、界面、工具、功能到环境搭建和常用的设置，对Android Studio做一个笼统的介绍，让大家对Android Studio有一个全面的认识。

本章重要知识点 >>>>>>>>>>

- 了解 Android Studio 及其特点；
- 了解欢迎、配置、工作台、工具界面的组成及功能；
- 了解如何配置 Android Studio 开发环境；
- 了解如何更新 Android Studio 和 SDK；
- 了解如何进行一些常用配置。

1.1　什么是 Android Studio

1.1.1　Android Studio简介

Android Studio是一个基于IntelliJ IDEA社区版本的Android开发环境，与Eclipse ADT插件相似，Android Studio提供了集成的Android开发工具用于开发和调试。作为Google官方的IDE，Android Studio提供了开发和构建Android应用的所有工具，包括智能代码编辑器、布局编辑器、代码分析和调试工具、应用构建系统、模拟器以及性能分析工具等。

Android Studio是Android平台上构建高质量、高效应用的最快方法，包括手机、平板、Android Auto、Android Wear和Android TV。

Android Studio于 2013 年 5 月 16 日在谷歌I/O大会正式对外发布，目前已更新到 2.2 版本（见图 1-1）。

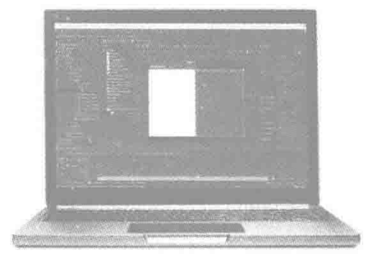


图 1-1

1.1.2 系统要求

macOS、Windows、Linux三大操作系统全部支持。

需要满足最基本的系统要求，如表 1-1 所示。

表 1-1

平台	macOS	Windows	Linux
操作系统	macOS® 10.8.5 或更高版本	Microsoft® Windows® 7/8/10（32 位或 64 位）或更高版本	GNOME 或 KDE 桌面，推荐 Ubuntu® 12.04
内存大小	最低 2GB RAM，推荐使用 8GB RAM		
硬盘空间	最低 2GB 可用磁盘空间，推荐 4GB（IDE 500 MB + Android SDK 和模拟器映像 1.5GB）		
JDK版本	Java开发工具包（JDK）7 或更高版本		
屏幕分辨率	最低屏幕分辨率为 1280×800		

提 示

IntelliJ IDEA 被认为是当前 Java 开发效率最快的 IDE 工具，它整合了开发过程中实用的众多功能，几乎不用鼠标就可以快速地完成你要做的任何事情，最大程度加快开发的速度。其简单而又强大的功能与其他一些烦冗而复杂的 IDE 工具有鲜明的对比。

1.1.3 下载地址

官方网站的下载地址为http://developer.android.com/sdk/index.html，这里提供了三大平台的安装包，选择对应的平台进行下载即可，如图 1-2 所示。

平台	Android Studio 软件包	大小	SHA-1 校验和
Windows	android-studio-bundle-145.3276617-windows.exe 包含 Android SDK（推荐）	1608 MB (1686392376 bytes)	04321c38b42d1aca901509d92174f8b42e37b1e9
	android-studio-ide-145.3276617-windows.exe 无 Android SDK	407 MB (426837480 bytes)	9d94f24be62e68c7fb004e4813155f5fc41b92f5
	android-studio-ide-145.3276617-windows.zip 无 Android SDK，无安装程序	428 MB (449589181 bytes)	fe47002865b292d5ed8e14acc64731dbc57251c0
Mac OS X	android-studio-ide-145.3276617-mac.dmg	423 MB (444453948 bytes)	e8230bed054719836caa2710c1036c19a0693b5f
Linux	android-studio-ide-145.3276617-linux.zip	428 MB (449256351 bytes)	4eec979ad4d218fd591ebe07123¢7c746cedb114

图 1-2

1.1.4 为什么要用Android Studio

原因一：Eclipse Android开发工具停止更新

谷歌至 2015 年底已停止对Eclipse Android开发工具的一切支持，包括ADT插件、Ant构建系统、DDMS、Traceview与其他性能和监控工具。

原因二：基于IntelliJ IDEA开发

众所周知，IntelliJ IDEA是世界上最好用的Java开发IDE，Android Studio是基于IntelliJ IDEA

社区版本开发的，所以Android Studio基本上继承了IntelliJ IDEA社区版本的所有功能。

原因三：谷歌出品

Android Studio是谷歌专门为Android开发量身定做的编辑器。

原因四：功能强大

Android Studio更加智能，集成了版本控制系统、代码分析工具、UI编辑器、Gradle构建工具、Android Monitor、模拟器、测试工具、各种模板和示例等，还有各种插件支持。

有了上面这些原因，还有什么理由不用Android Studio呢？

1.2 Android Studio 的特点

1. 智能代码编辑器

Android Studio最核心的功能就是智能代码编辑器（见图 1-3），能够帮助我们非常高效地完成代码补全、重构和代码分析。

图 1-3 代码编辑器界面

它还支持多种实用的视图模式，如演示模式、免打扰模式。它的快捷键、代码的显示方式、颜色、主题等都是可配的。

2. 代码模板和GitHub集成

Android Studio的新建项目向导让新建项目变得非常简单。在新建项目时我们可以选择想要的Activity模板（见图 1-4），也可以从GitHub上直接导入项目，还可以直接通过导入代码模板来快速开始项目。

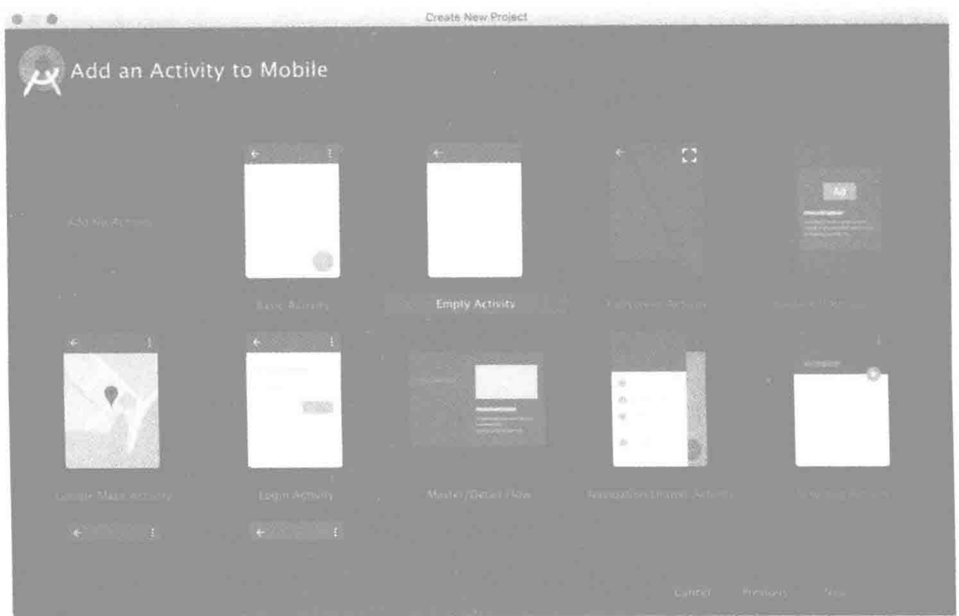


图 1-4　Activity 模板选择界面

3. 开发适用于多屏幕的应用

Android Studio支持构建适用于 Android 手机、平板电脑、Android Wear、Android TV、Android Auto 以及 Google Glass 的应用。全新的项目视图（见图 1-5）和模块支持让应用和资源管理变得更加轻松。

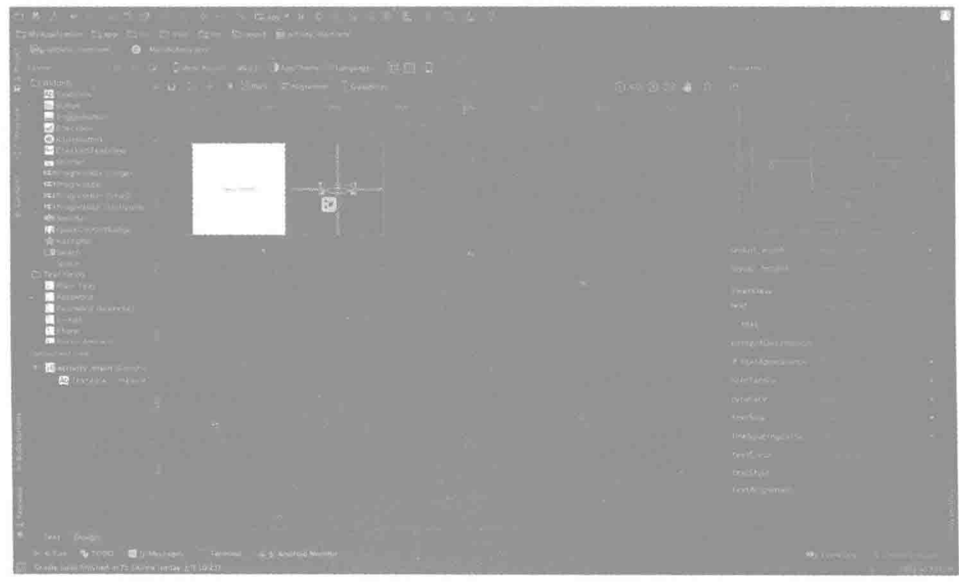


图 1-5　可视化布局界面

4. 支持所有形状和尺寸的模拟器

Android Studio预先配置了经过优化的模拟器映像。

经过更新和精简的虚拟设备管理器可以为常见 Android 设备提供预定义设备配置文件（见图 1-6）。自Android Studio 2.0 开始，模拟器的速度、性能、易用性都有了非常大的提升，你甚至可以放弃genymotion了。

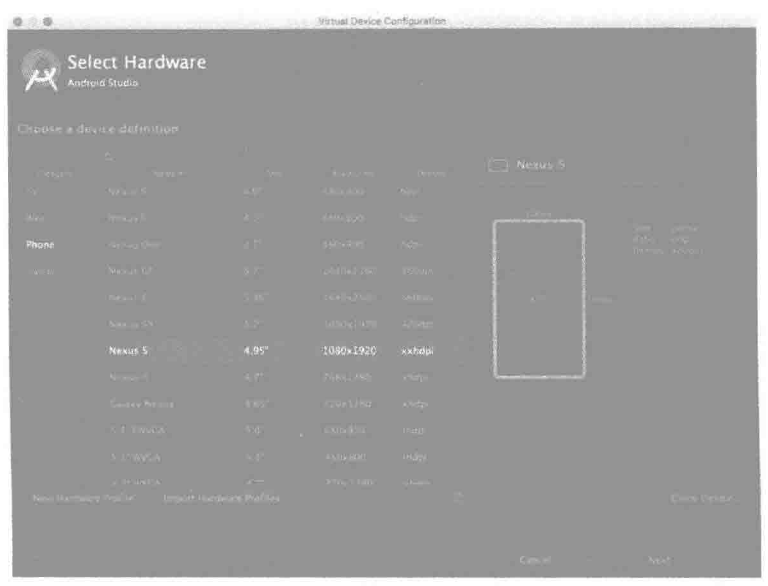


图 1-6 硬件设备选择界面

5. 基于Gradle的灵活构建系统

使用同一个项目的不同配置（见图 1-7）可以为我们的应用构建出多个具有不同功能的APK。

图 1-7 Gradle 配置界面

我们可以使用Android Studio或命令行来构建应用，非常灵活。

6. 强大到逆天的即时运行功能

即时运行（Instant Run，见图 1-8），顾名思义，就是你一边写代码，一边就会在模拟器或真机上立即看到修改后的运行效果，再也不用重新开始编译运行，大大提高了开发效率。

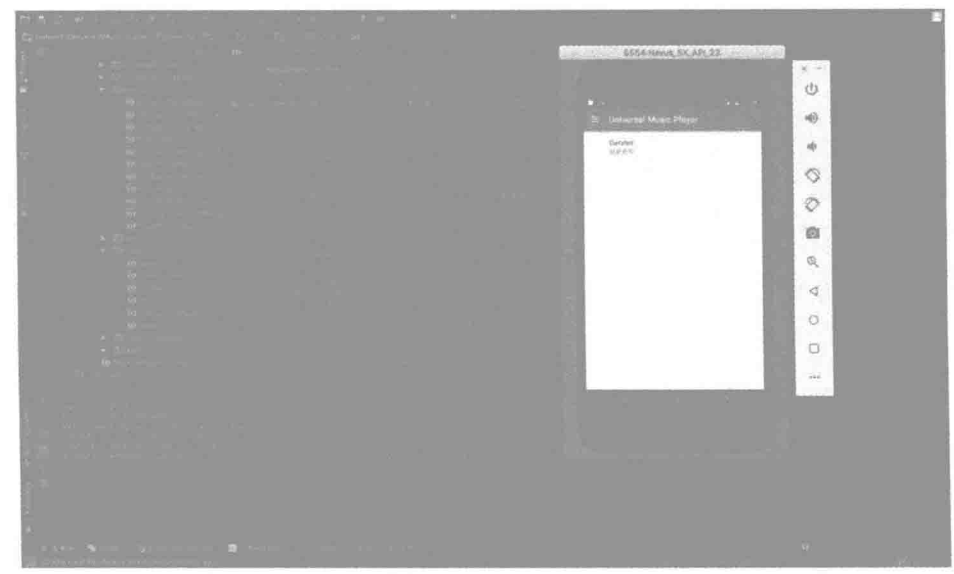


图 1-8　Instant Run

7. 速度更快的ADB

Android 6.0 Marshmallow或者更高的系统映像支持对称式多重处理，对模拟器和ADB进行了重大的改进，当使用ADB推送文件时速度可以比真实设备快五倍。如果应用比较大，这个功能将非常有帮助。

8. 更多特性

支持最新的Android版本；

- 支持基于 Gradle 的灵活的构建系统；
- 支持构建变种版本和生成多个 APK 文件；
- 支持通过代码模板来快速建立通用的 App 功能；
- 支持可视化的设计工具，可拖放主题编辑；
- 支持 Lint 提示工具，可以更好地对程序性能、可用性、版本兼容和其他问题进行控制捕捉；
- 支持代码混淆和应用签名；
- 支持 C/C++开发；
- 支持速度更快的即时运行；
- 内置的 Google 云平台支持，可轻松集成 Google Cloud Messaging 和应用引擎；
- 内置 ADB 工具，可以方便地查看 log；
- 内置性能检测工具，可以实时检测内存、CPU、流量、GPU。

1.3 macOS 环境配置

1.3.1 配置JDK

第 1 步：下载JDK。

下载地址为 http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html（JDK7）或http://www.oracle.com/technetwork/java/javase/downloads/jdk8- downloads-2133151.html（JDK8）。

这里以JDK7 为例进行介绍，但是Android Studio 2.2 及以后的版本需要使用JDK8。选择 Mac OS X x64 下载，如图 1-9 所示。

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	119.43 MB	jdk-7u75-linux-i586.rpm
Linux x86	136.77 MB	jdk-7u75-linux-i586.tar.gz
Linux x64	120.83 MB	jdk-7u75-linux-x64.rpm
Linux x64	135.66 MB	jdk-7u75-linux-x64.tar.gz
Mac OS X x64	185.86 MB	jdk-7u75-macosx-x64.dmg
Solaris x86 (SVR4 package)	139.55 MB	jdk-7u75-solaris-i586.tar.Z
Solaris x86	95.87 MB	jdk-7u75-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.66 MB	jdk-7u75-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u75-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	138.66 MB	jdk-7u75-solaris-sparc.tar.Z
Solaris SPARC	98.56 MB	jdk-7u75-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u75-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.37 MB	jdk-7u75-solaris-sparcv9.tar.gz
Windows x86	127.8 MB	jdk-7u75-windows-i586.exe
Windows x64	129.52 MB	jdk-7u75-windows-x64.exe

图 1-9

第 2 步：安装JDK。

下载完成后，双击安装包，然后按照提示进行安装，如图 1-10 所示。

图 1-10

安装完成后的路径：

```
/Library/Java/JavaVirtualMachines/
```

如果安装了多个JDK版本，这里会显示多个。

```
$ ls
jdk1.7.0_71.jdk jdk1.7.0_75.jdk jdk1.7.0_79.jdk
```

切换到Home目录下：

```
/Library/Java/JavaVirtualMachines/jdk1.7.0_75.jdk/Contents/Home
```

第 3 步：配置环境变量。

先查看原来的Java版本，如图 1-11 所示。

```
bixiaopeng@bixiaopeng ~$ java -version
java version "1.6.0_65"
Java(TM) SE Runtime Environment (build 1.6.0_65-b14-466.1-11M4716)
Java HotSpot(TM) 64-Bit Server VM (build 20.65-b04-466.1, mixed mode)
```

图 1-11

再利用vim .bash_profile配置JAVA_HOME，如图 1-12 所示。

如图 1-12

保存之后执行source .bash_profile命令，更新成功，如图 1-13 所示。

```
bixiaopeng@bixiaopeng ~$ source .bash_profile
bixiaopeng@bixiaopeng ~$ java -version
java version "1.7.0_75"
Java(TM) SE Runtime Environment (build 1.7.0_75-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.75-b04, mixed mode)
```

图 1-13

1.3.2 下载Android Studio

既可以下载可执行文件进行安装（http://developer.android.com/sdk/index.html），也可以下载beta版本（http://tools.android.com/recent）试用。beta版本不用安装，可以多个版本一起使用。这里下载beta版本（1.4.2 小节和 1.5.2 小节也是以beta版本为例）。

1.3.3 配置Android的环境变量

如果我们想直接使用Android提供的一些工具，就需要配置环境变量。

假设我们本地SDK的路径为/home/bixiaopeng/Android/Sdk，vim ~/.bash_profile 会把下面这些环境变量加进去：

```
export ANDROID_HOME=/home/bixiaopeng/Android/Sdk
export PATH=$ANDROID_HOME/platform-tools:$ANDROID_HOME/tools:$PATH
```

如果你的电脑上也需要配置，请把SDK路径换成自己的。

1.4 Windows 环境配置

1.4.1 配置JDK

第 1 步：下载JDK7。

选择Windows x64 下载（可参考 1.3.1 小节）。

第 2 步：安装JDK。

下载完成后双击安装，并按照提示一步一步直到完成安装。安装时请记住JDK的安装位置，后面在配置环境变量的时候要用到。

第 3 步：配置环境变量。

右击“计算机”→属性→高级系统设置→环境变量，如图 1-14 所示。

新建系统变量JAVA_HOME→“变量值”设为刚才安装的JDK路径，如图 1-15 所示。

图 1-14

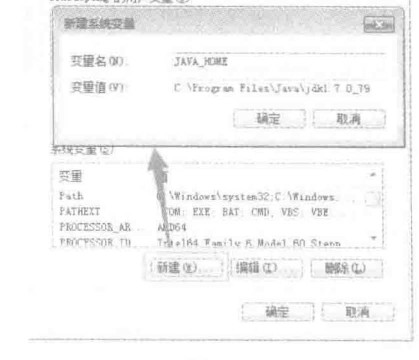


图 1-15

在系统变量path中添加 %JAVA_HOME%\bin，如图 1-16 所示。

新建系统变量CLASS_PATH→添加变量值“%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;”，如图 1-17 所示。

图 1-16

图 i-17

到此为止，环境变量配置完毕，下面来验证一下是否配置成功。

如图 1-18 所示，在终端输入javac命令，如果显示帮助信息就证明配置成功了。

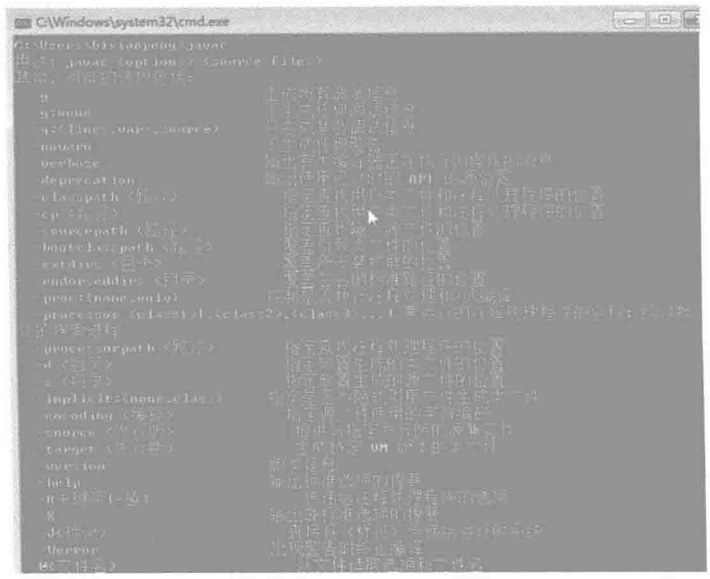


图 1-18

1.4.2 配置Android Studio

1. 下载Android Studio

在下载页面选择所要的安装包进行下载，如图 1-19 所示。

Windows: https://dl.google.com/dl/android/studio/ide-zips/2.1.0.1/android-studio-ide-143.2682553-windows.zip (293 MB)
Mac: https://dl.google.com/dl/android/studio/ide-zips/2.1.0.1/android-studio-ide-143.2682553-mac.zip (292 MB)
Linux: https://dl.google.com/dl/android/studio/ide-zips/2.1.0.1/android-studio-ide-143.2682553-linux.zip (291 MB)

图 1-19

2. 启动Android Studio

下载→解压→进入android/bin目录→双击studio64，启动Android Studio，如图 1-20 所示。

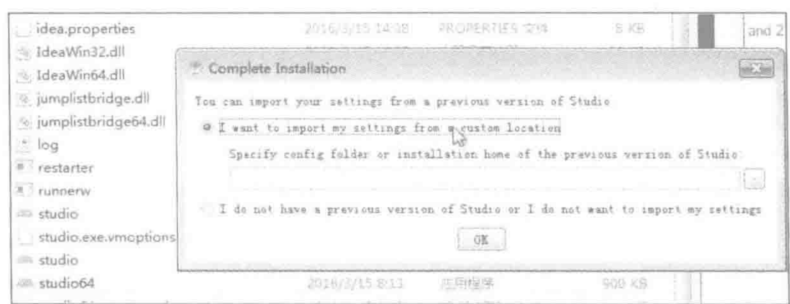


图 1-20

如果想导入原来的设置就选中上面的单选按钮，如果不想就选中下面的单选按钮。

接下来按照提示逐步进行安装。安装过程中会安装SDK和一些组件（见图 1-21），请注意一下SDK的安装位置。如果要在终端直接使用Android命令就需要配置Android的环境变量。

下载完成后就可以正常启动Android Studio了（见图 1-22）。

图 1-21

图 1-22 Android Studio 欢迎界面

1.4.3 配置Android的环境变量

如果想直接使用Android提供的一些工具，就需要配置环境变量。

打开下载好的SDK目录，看一下里面的目录结构，有很多工具可供我们使用，如图 1-23 所示。

接下来配置环境变量。右击“计算机”→属性→高级系统设置→环境变量→新建系统变量ANDROID_HOME→“变量值”设为刚才安装的SDK的路径，如图 1-24 所示。

图 1-23

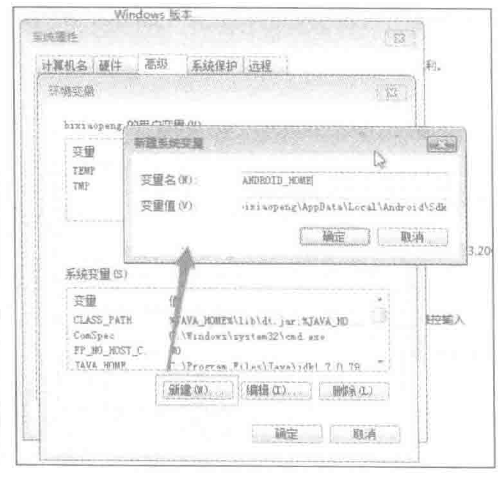


图 1-24

接下来在Path中添加Android SDK和Android工具的路径，如图 1-25 所示。

配置完成后验证一下配置是否正确，如图 1-26 所示。

%ANDROID_HOME%;%ANDROID_HOME%\tools;%ANDROID_HOME%\build-tools;%ANDROID_HOME%\platform-tools;

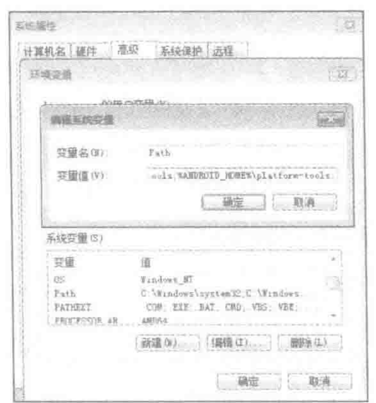


图 1-25

```
C:\Windows\system32\cmd.exe
C:\Users\bixiaopeng>adb devices
List of devices attached

C:\Users\bixiaopeng>
```

图 1-26

只要不报adb不是内部命令的错误，那就是配置成功了。

1.5　Linux 环境配置

1.5.1　配置JDK

第 1 步：下载JDK7。

选择Linux x64 下载（可参考 1.3.1 小节）。

第 2 步：解压JDK。

下载jdk-7u79-linux-x64.tar.gz后解压到当前目录。

```
tar -zxvf jdk-7u79-linux-x64.tar.gz
```

解压后的目录为jdk1.7.0_79，再把解压后的目录放到指定的目录中。

```
cp -r /home/bixiaopeng/下载/jdk1.7.0_79  /home/bixiaopeng/soft/jdk
```

第 3 步：配置环境变量。

vim ~/.bashrc 把下面这些环境变量加进去：

```
export JAVA_HOME=/home/bixiaopeng/soft/jdkexport
export PATH=$JAVA_HOME/bin：$PATH
export CLASSPATH=.：$JAVA_HOME/lib/dt.jar：$JAVA_HOME/lib/tools.jar
```

生效后验证一下配置是否正确，如图 1-27 所示。

```
bixiaopeng@bixiaopeng-virtual-machine:~$ java -version
java version "1.7.0_79"
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
```

图 1-27

1.5.2　配置Android Studio

1. 下载Android Studio

假设下载的压缩包为android-studio-ide-141.2456560-linux.zip，下载完成后需要解压：

```
unzip android-studio-ide-141.2456560-linux.zip
```

解压后的目录为android-studio。

2. 启动Android Studio

进入android/bin目录，如图 1-28 所示。

图 1-28

运行studio.sh就可以启动Android Studio了，如图 1-29 所示。

图 1-29

如果你想导入原来的设置就选中上面的单选按钮，如果不想就选中下面的单选按钮。

接下来按照提示逐步进行安装。

安装过程中会安装SDK和一些组件，请注意一下SDK的安装位置。如果你要在终端直接使用Android命令还需要配置Android的环境变量。

1.5.3 配置Android的环境变量

如果想直接使用Android提供的一些工具就需要配置环境变量。

利用vim ~/.bashrc把下面这些环境变量加进前面的SDK安装目录（/home/bixiaopeng/Android/Sdk）中：

```
export ANDROID_SDK=/home/bixiaopeng/Android/Sdk

export PATH=$ANDROID_SDK/platform-tools：$ANDROID_SDK/tools：$PATH
```

如果你的电脑上也需要配置，请把SDK路径换成自己的。

1.6 认识欢迎界面

安装完成后打开Android Studio首先会进入欢迎界面（见图 1-30）。欢迎界面提供了一个项目开始前所有可能的操作入口，比如查看版本信息、最近打开项目列表、新建项目、打开项目、导入项目、配置、帮助等。

图 1-30

1.6.1 最近打开的项目

在最近打开项目列表中可以方便地对最近打开过的项目进行管理，快速执行打开、删除、分组等操作，如图 1-31 所示。

图 1-31

1.6.2 开始一个项目

在欢迎界面可以快速开始一个新项目，包括新建、打开已经存在的项目、从版本控制系统中检出、导入项目和导入一个示例代码（见图 1-32）。

- Start a new Android Studio project：新建一个 Android Studio 项目。
- Open an existing Android Studio project：打开一个已存在的 Android Studio 项目。
- Check out project from Version Control：从版本控制系统中导出一个项目。
- Import project（Eclipse ADT，Gradle，etc.）：从 Eclipse 或 Gradle 中导入项目。
- Import an Android code sample：导入一个 Android 示例代码。

具体操作在第 2 章有详细的介绍。

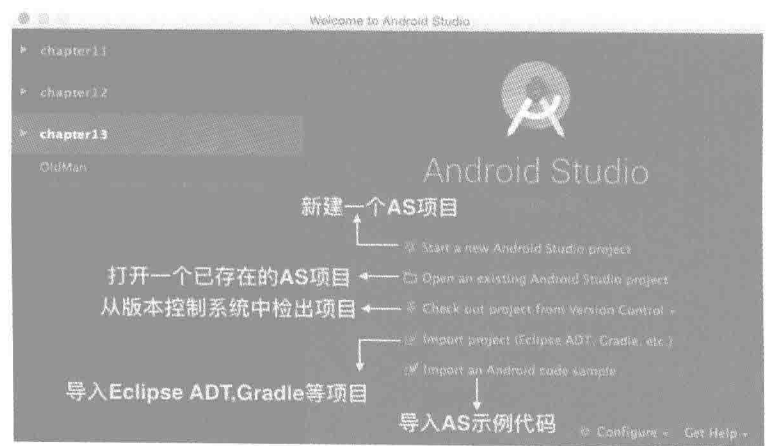


图 1-32

1.7 认识配置界面

Android Studio提供了快速配置功能，可以在不打开项目的情况下进行配置（在Android Studio欢迎界面单击【Configure】可进入配置界面）。

1. 配置功能概览

配置功能如图 1-33 所示。

- SDK Manager：管理 SDK。
- Preferences：偏好设置。
- Plugins：插件。
- Import Settings：导入设置。
- Export Settings：导出设置。
- Settings Repository：设置仓库。
- Check for Update：检查更新。
- Project Defaults：项目默认设置。

图 1-33 配置功能

2. 管理Android SDK

在Android Studio中需要通过SDK Manager来安装、更新和卸载Android SDK和开发工具。

（1）在欢迎界面直接启动 SDK Manager

操作步骤：欢迎界面→Configure→SDK Manager。

打开的SDK Manager界面如图 1-34 所示。

（2）其他启动方法

❶ 在工具栏直接启动，如图 1-35 所示。

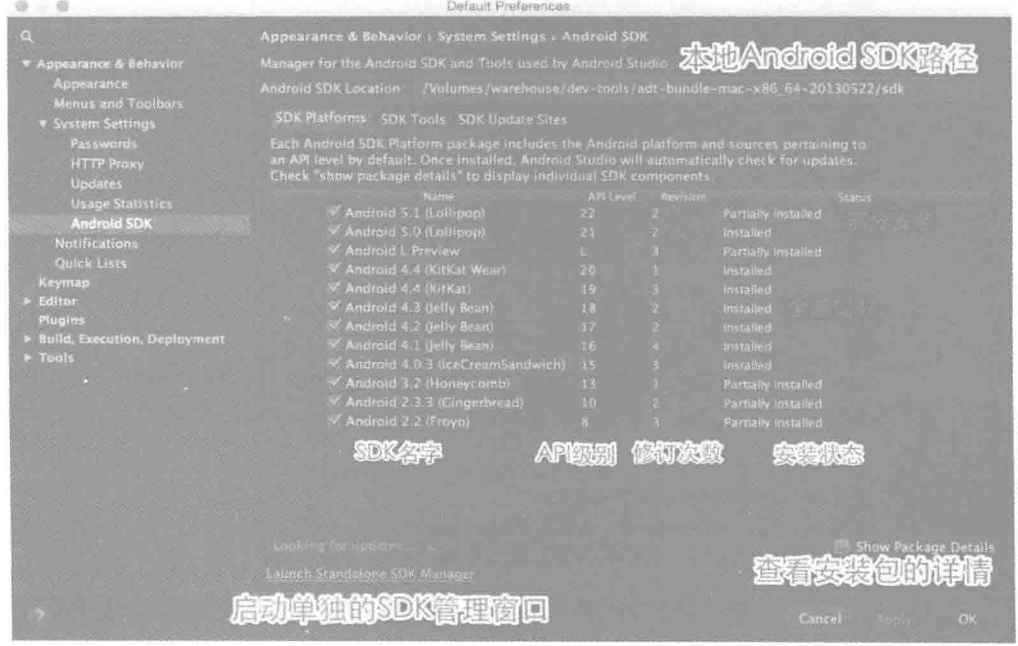


图 1-34 SDK Manager

图 1-35

❷ 在偏好设置中启动：Preferences→Appearance & Behavior→System Settings→Android SDK→SDK Platforms。

如果无法更新SDK就设置代理。

3. 偏好设置

操作步骤：欢迎界面→Configure→Preferences。

关于偏好设置更详细的内容请看第 16 章。

4. 插件

操作步骤：欢迎界面→Configure→Plugins。

打开的插件管理界面如图 1-36 所示。

Android Studio有着非常丰富且强大的插件支持，更多插件管理的内容请看第 16 章。

5. 导出设置

Android Studio支持将偏好设置文件以jar包的形式进行共享，因此我们可以将设置导出。

操作步骤：欢迎界面→Configure（或者菜单栏File）→Export Settings。

进入导出设置界面，然后选择你要导出的设置，默认是全选，如图 1-37 所示。

如果想把代码风格（约定好的代码规范）导出共享给合作团队，就只选择一个Code Style，然后保存为codestyle.jar，其他人导入这个jar包即可。

图 1-36

图 1-37

6. 导入设置

操作步骤：欢迎界面→Configure→Import Settings→选择settings.jar，或者是菜单栏→File→Import Settings。

然后会提示选择要导入的设置选项，如图 1-38 所示。

图 1-38

单击OK按钮后成功导入。

7. 设置仓库

Settings Repository（设置仓库）使用GitHub/Gitlab来存储设置，可以共享设置文件给所有基于JetBrains平台的产品（Android Studio、Intellij IDEA、Pycharm等）。

操作步骤：欢迎界面→Configure→Settings Repository 或者"菜单栏"→File→Settings Repository，然后弹出配置对话框，如图 1-39 所示。

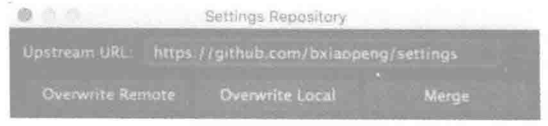


图 1-39

可以在Upstream URL中输入用于共享设置的Git仓库的地址。当需要同步设置的时候，可以选择本地的设置覆盖远程（Overwrite Remote），或远程的设置覆盖本地（Overwrite Local），也可以合并本地和远程的设置（Merge）。

同步设置可以选择手动或自动。

自动同步：

- 执行“Update Project”或“Push”。
- 关闭软件或关闭项目。

手动同步：

“菜单栏”→VCS→Sync Settings

同步有风险，操作需谨慎。更多请参考：https://github.com/JetBrains/intellij-community/tree/master/plugins/settings-repository。

8. 检查更新

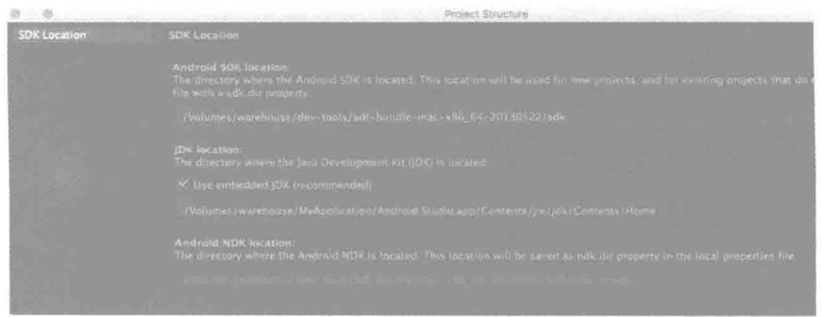
操作步骤：欢迎界面→Configure→Check for Update。

具体操作请参考第 1.9 节。

9. 项目默认设置

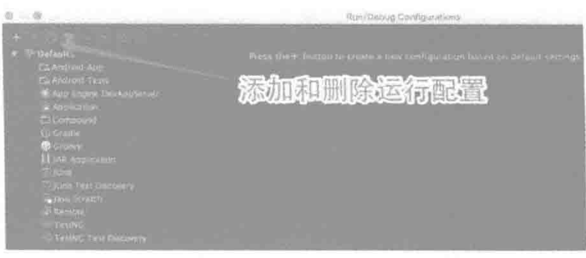
操作步骤：欢迎界面→Configure→Project Defaults。

进入项目默认配置界面，可以根据需要调整相关配置，如图 1-40 所示。



（a）项目结构

（b）偏好设置



（c）运行配置

图 1-40

1.8 帮助和教程

1. 查看帮助文档

Android Studio的文档非常齐全，如果你直接读英文没有什么障碍，强烈建议你多看看帮助文档。

操作步骤：欢迎界面→Get Help（菜单栏→Help）→Help→打开IntelliJ IDEA的帮助文档。

另外，还可以到IntelliJ IDEA的官方网站去查看帮助文档：

```
https://www.jetbrains.com/help/idea/2016.1/meet-intellij-idea.html
```

Android Studio的帮助文档：

```
https://developer.android.com/studio/intro/index.html
```

2. 每天一个技巧提示

Tips of The Day会提示一些实用的操作技巧，如图 1-41 所示。勾选【Show Tips on Startup】后会在每次启动Android Studio时都显示这个提示。

操作步骤：欢迎界面→Get Help（菜单栏→Help）→Tip of The Day。

图 1-41

3. 查看快捷键

操作步骤：欢迎界面→Get Help（菜单栏→Help）→Keymap Reference

这里会显示Android Studio对应平台的默认快捷键，如图 1-42 所示。这里使用的电脑系统是macOS，所以会显示Mac上的快捷键。

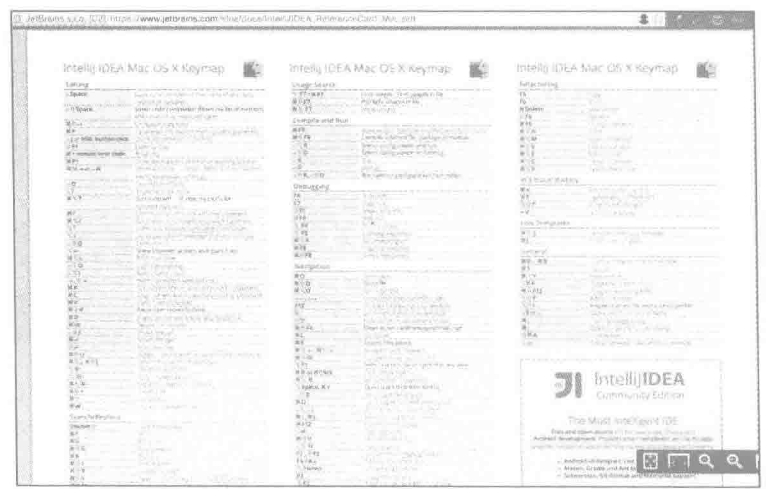


图 1-42

这些快捷键全部都可以自定义，后面介绍快捷键设置的时候会讲到。

4. 插件开发

没有人能开发出一个IDE能满足所有开发者的需求，因此一个优秀的IDE一定是支持插件开发的。如果想为Android Studio开发插件，可以参考下面的插件开发文档。

操作步骤： 欢迎界面→Get Help（菜单栏→Help）→Plugin Development。

5. 搜索菜单选项

当我们忘记某个菜单选项在哪里的时候，可以通过搜索来找到这个选项。例如，忘记了打开Captures的选项在哪里，就可以利用搜索来找到。

操作步骤： 菜单栏→Help→Search→输入capture→光标移动到菜单选项上→菜单会自动打开，且有蓝色箭头的指示指到Captures，如图 1-43 所示。

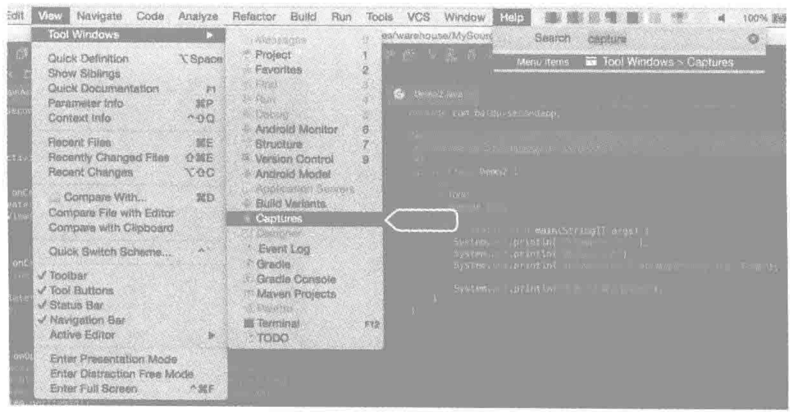


图 1-43

再举个例子：搜索commit选项在哪里，如图 1-44 所示。

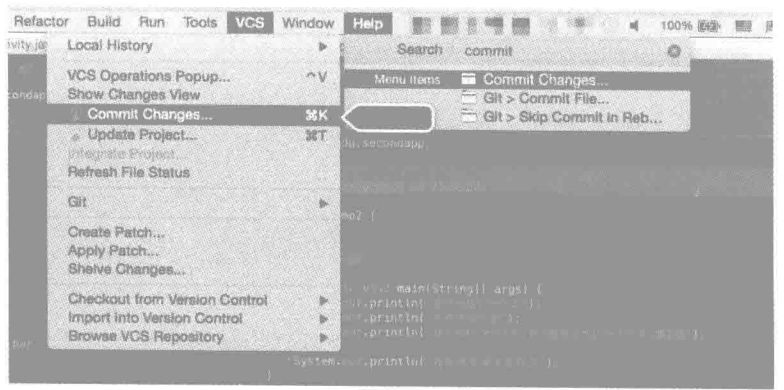


图 1-44

6. 全局查找操作

查找操作功能会搜索Android Studio中所有能匹配到的操作，包括菜单选项、设置、工具等。对于没有设置快捷键或者忘记快捷键的菜单或者操作，可以通过输入名字快速调用。例如，搜索Captures选项，操作步骤如下：

第 1 步：调出操作窗口。

菜单栏：Help→Find Action。

快捷键：shift + command + A（macOS）或Ctrl + Shift + A（Windows/Linux）。

第 2 步：输入关键字。

输入capture→光标移动到下拉列表上，选中后执行操作，如图 1-45 所示。

再举个例子：搜索 commit相关操作，如图 1-46 所示。

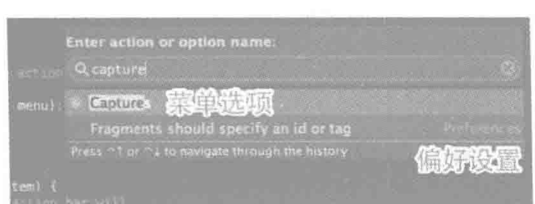


图 1-45

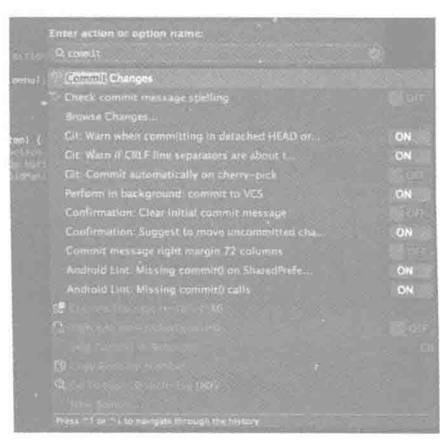


图 1-46

7. 效率指南

效率指南统计使用Android Studio工作的一些效率情况。

操作步骤：菜单栏→Productivity Guide→弹出效率统计窗口，如图 1-47 所示。

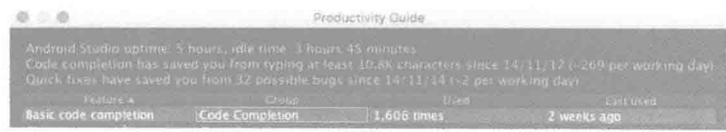


图 1-47

我们来看一下统计的结果：

Android Studio 运行：5 小时，空闲 3 小时 45 分钟。

从 2014 年 11 月 17 日以来，Android Studio 的代码补全功能已经至少帮你输入了 10.8KB 字符（平均每个工作日 269 个）。

从 2014 年 11 月 14 日以来 Android Studio 帮你快速解决了 32 个可能的 BUG（平均每个工作日 2 个）。

图 1-48 所示是一些功能的使用情况统计。

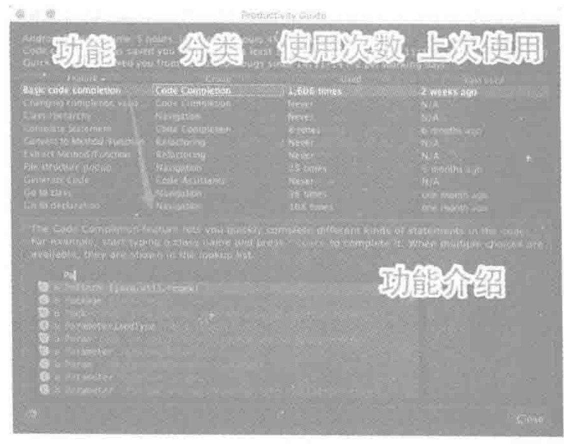


图 1-48

1.9 更新 Android Studio 版本

1. 关于Android Studio

当我们想查看当前Android Studio的版本和使用的JDK版本时，选择菜单栏中的Android Studio→About Android Studio（macOS）或者Help→About（Windows/Linux），然后会弹出一个对话框显示Android Studio的版本号、构建日期以及JDK版本，如图 1-49 所示。

图 1-49

2. 检测版本更新

在欢迎界面单击底部Check for updates now的Check，或者在菜单栏中选择Android Studio（macOS）或Help→Check for Updates（Windows/Linux），Android Studio就会开始检测是否有更新。

如果已经是最新版本，就会弹出如图 1-50 所示的这个提示。

如果不是最新版本，就会弹出更新提示（可能是Android Studio版本更新，也有可能是插件或SDK更新），如图 1-51 所示。

图 1-50

图 1-51

3. 版本更新过程

下面以更新到Dev Channel的最新版本为例进行介绍。当有新版本可以更新时（见图 1-52），单击【Update and Restart】按钮后会开始更新并重启（见图 1-53）。

图 1-52

图 1-53

更新完成后会提示你是否要导入之前版本的设置（见图 1-54），单击【OK】按钮就可以了！

图 1-54

1.10 配置 Android Studio 更新通道

Android Studio提供了不同的更新通道，我们可以在偏好设置中指定更新通道。

1.10.1 在偏好设置中配置更新通道

操作步骤：偏好设置→Appearance & Behavior→System Settings→Updates，然后显示更新通道配置界面，如图 1-55 所示。

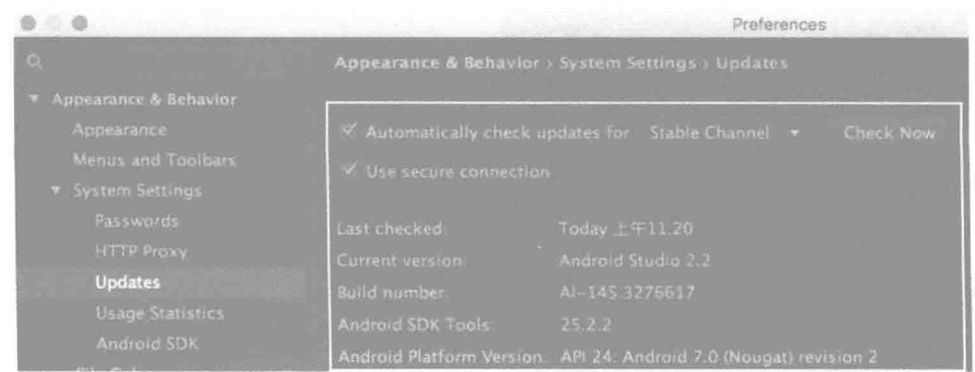


图 1-55

在Updates界面上显示了当前版本的信息和上次检查更新的时间，可以在这里配置Android Studio和Android SDK的更新通道，然后立即检查更新。

1.10.2 在检测结果对话框中配置更新通道

当我们执行检测更新以后，弹出的检测结果对话框中也是可以配置更新通道的。

操作步骤：菜单栏→Android Studio（macOS）或Help→Check for Updates（Windows/Linux），在弹出的对话框中选择Updates，进入更新通道配置界面，如图 1-56 所示。

1.10.3 单独下载最近更新的版本

有四种通道的版本可供我们下载，如图 1-57 所示。（下载地址为http://tools.android.com/download/studio。）

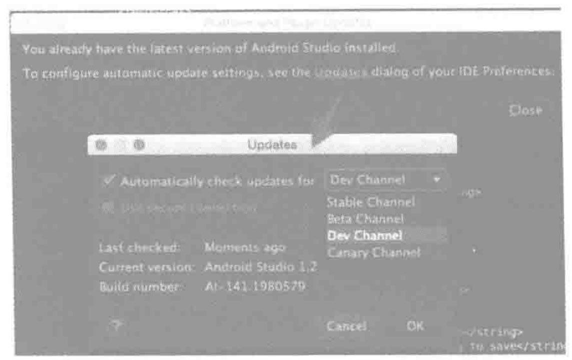


图 1-56

图 1-57

1.10.4 四种版本的区别

Android Studio提供了四种更新通道（见图 1-58），在检测更新时会分别检测不同的版本。

图 1-58

- Stable Channel：稳定版本。
- Beta Channel：测试版本。
- Dev Channel：开发版本。
- Canary Channel：金丝雀版本。

这四种版本更新频率从Canary往上逐渐递减：Canary大概 1 或者 2 周会更新一次，Beta则为相对稳定的发布版，Stable则是正式版。稳定性与其更新频率相反，稳定性从Canary往上逐渐递增。环境问题经常会困扰着我们，所以如果不是特别需要，最好使用稳定版本。如果想尝鲜，可以使用Canary Channel。

图 1-59 所示为Canary Channel最近的更新信息（地址为http://tools.android.com/download/studio/canary/latest）。

图 1-59

1.11 在 Android Studio 中使用代理

1. 为什么使用代理

Android Studio中默认选中【No proxy】，意思是不使用代理。但是作为开发人员，特别是Android开发人员，如果不用代理好多事情是做不了的，例如查看Android开发文档、查找英文资料等，也会出现好多组件无法下载或下载速度很慢以及SDK无法更新等问题，如图 1-60 所示。

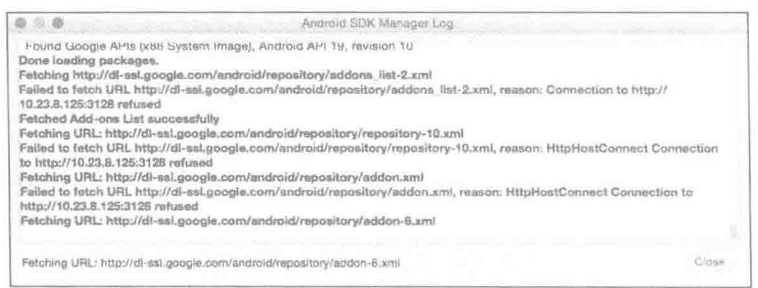


图 1-60

因此需要通过设置代理（自动检测代理设置和手动配置代理）来更加顺畅地使用Android Studio，更加方便地使用Google提供的各种服务。

2. 设置代理

操作步骤： 偏好设置→Appearance & Behavior→System Settings→HTTP Proxy，然后显示HTTP代理设置界面，如图 1-61 所示。

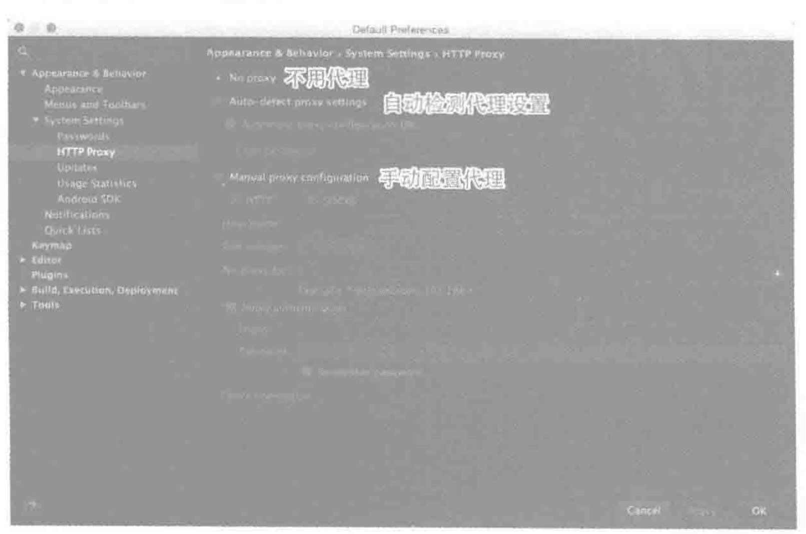


图 1-61

3. 配置手动代理

配置手动代理的界面如图 1-62 所示。根据需要配置好代理以后，单击【Check connection】来检查是否能正常连接。

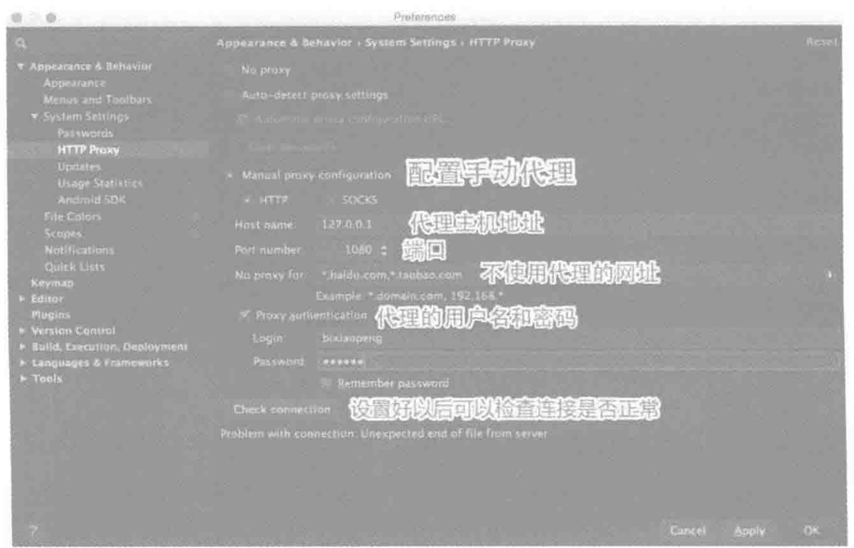


图 1-62

1.12　认识偏好设置

macOS上是Preferences（偏好设置），Windows/Linux上是Settings，因为这里是用macOS做演示，所以关于设置的内容在本书中统一称为偏好设置。

Android Studio的偏好设置默认被分为八大类，如图 1-63 所示。

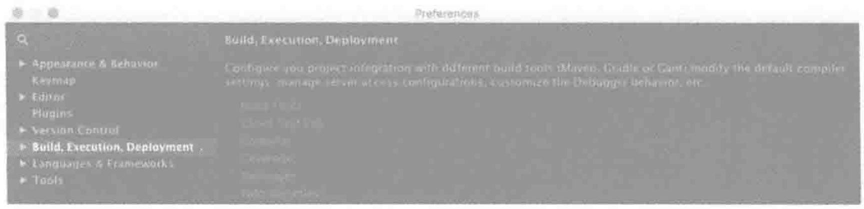


图 1-63

- Appearance & Behavior：外观与行为。
- Keycap：键盘映射。
- Editor：编辑器。
- Plugins：插件。
- Version Control：版本控制。
- Build，Execution，Deployment：构建，执行，部署。
- Languages & Frameworks：语言与框架。
- Tools：工具。

当我们需要进行偏好设置的时候，通过分类可以快速找到对应的配置项。除此之外，Android Studio还提供了非常便捷的配置搜索功能，可以通过模糊匹配来找到对应的配置项，如图 1-64 所示。

图 1-64

1.12.1 基础配置

在开始使用Android Studio开发项目之前，需要做一些基础的配置。

1. 设置代理

只有设置了代理以后，更新IDE、更新SDK、使用示例代码、使用Google提供的服务才能够畅通无阻，因此学会设置代理是一切的前提。

当然，不设置代理，除了Google的服务不能用之外，其他的问题都是有方法解决的。例如，IDE和SDK更新可以使用国内搬运工搬进来，示例代码能在GitHub上找到。

操作步骤：偏好设置→Appearance & Behavior→System Settings→HTTP Proxy，然后配置手动代理（参见图 1-62）。

2. 配置SDK/JDK路径

配置SDK/JDK路径如图 1-65 所示。

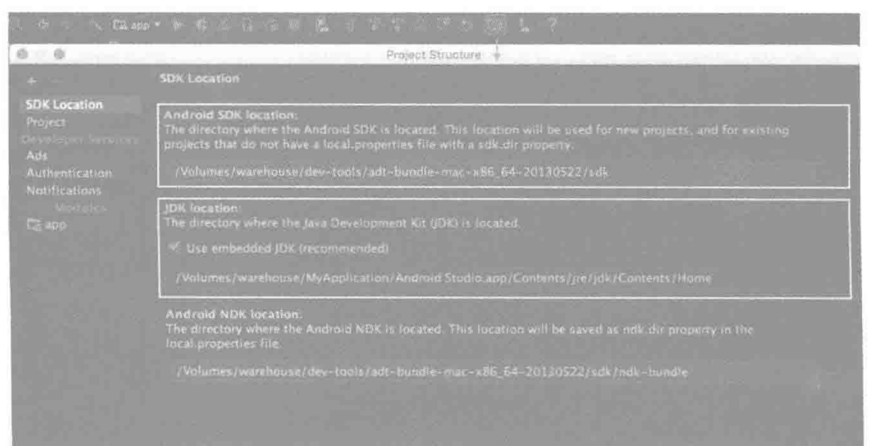


图 1-65

3. 下载SDK

找一个网络好的地方，下载所需要的SDK，如图 1-66 所示。

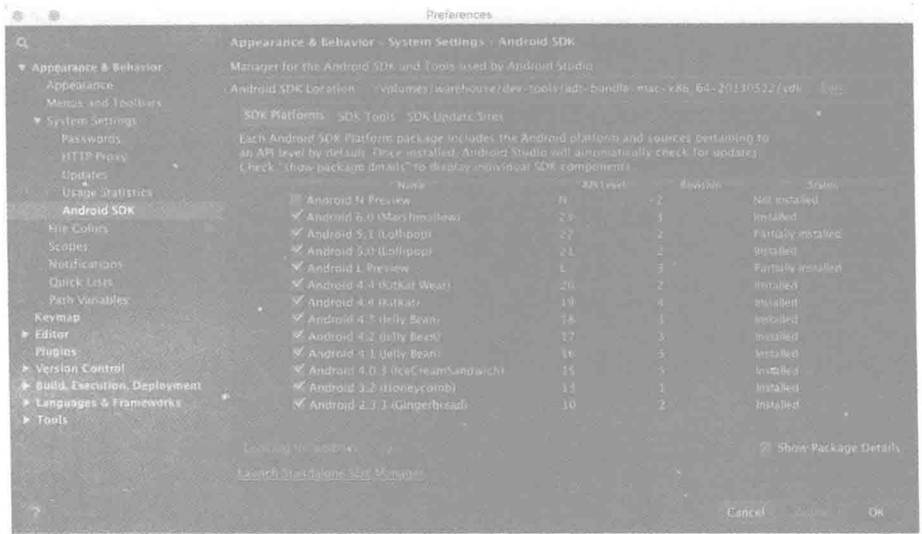


图 1-66

4. 编码方式

编码方式不匹配会导致乱码，我们需要知道在哪里设置编码方式（见图 1-67）。

操作步骤：偏好设置→Editor→File Encodings。

图 1-67

1.12.2 个性化配置

1. 主题

个性化的配置不能少了主题，Android Studio默认的主题如图 1-68 所示。

如果我们想把Android Studio设置为如图 1-69 所示的主题，应该怎么设置呢？

操作步骤：偏好设置→Appearance & Behavior→Appearance→Theme设为【Darcula】。

图 1-68

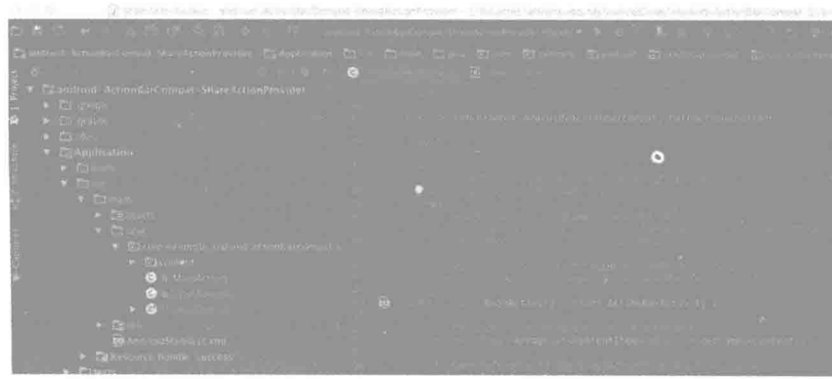


图 1-69

2. 字体和大小

操作步骤：偏好设置→Appearance & Behavior→Appearance→勾选【Override default fonts by（not recommended）】→选择所要的字体和字号，如图 1-70 所示。

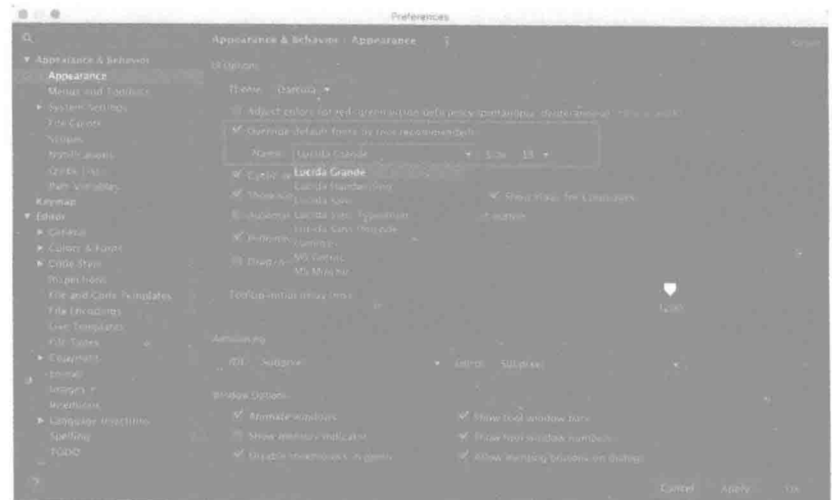


图 1-70

注意，不推荐自己去设置字体和字号。

3. 更新

想尝鲜就使用Canary Chanel，想稳定就使用Stable Chanel。

操作步骤：偏好设置→Appearance & Behavior→System Settings→Updates，如图 1-71 所示。

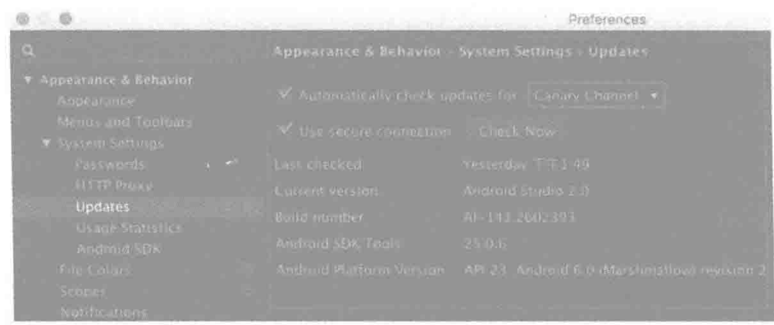


图 1-71

更多设置技巧请看第 16 章。

1.13 认识工作台

Android Studio的工作窗口称为工作台，工作台通常是由菜单栏、工具栏、导航条、编辑器、工具窗口、状态栏组成的。

1. 整体布局

打开一个Android项目进入编辑界面，能够看到Android Studio的整体布局，如图 1-72 所示。

图 1-72

2. 菜单栏

菜单栏提供了文件、编辑、视图、导航、代码、分析、重构、构建、运行、工具、版本控制系统、窗口、帮助等功能菜单，如图 1-73 所示。每个菜单都包含了很多非常实用的子功能，大部分功能都有快捷键。这使得我们可以快速地编写代码以及对项目进行管理。

Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

图 1-73

3. 工具栏

工具栏的工具都是从菜单栏中提取出来的一些常用的功能，为的是能够快速操作，如图 1-74 所示。

4. 导航条

导航条用来辅助查看打开的项目和文件。

通过菜单栏中的View→勾选Navigation Bar可以显示导航条，去掉勾选Navigation Bar即可关闭导航条。

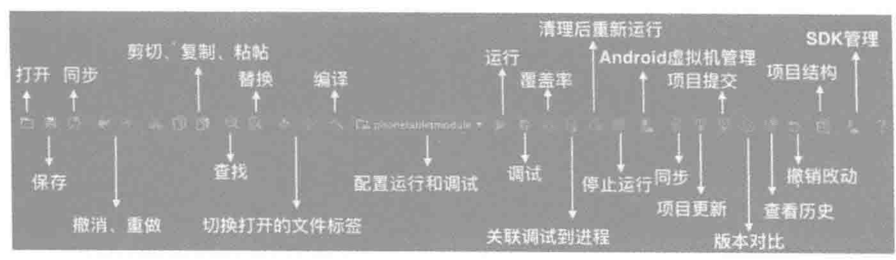


图 1-74

5. 编辑器

编辑器（见图 1-75）是基于标签的，在Android Studio中打开一个文件编辑时会打开一个新的文件标签。

图 1-75

（1）编辑区

在编辑区进行编码工作，编辑器提供了辅助编码的功能。

（2）左边栏

左边栏显示了代码的附加信息，并显示不同的图标来区别代码结构、书签、断点、范围指示符、变化标记和代码折叠线。

（3）右边栏

右边栏显示了代码的警告或错误信息，黄色为警告，红色为错误。将鼠标放到上面可以查看警告和错误数量，单击警告可以跳转到对应的代码。

（4）文件标签

打开一个文件就显示一个标签，我们可以通过标签在多个已打开的文件中快速切换。单击文件标签就相当于打开一个文件，文件的内容会显示出来且处于可编辑状态。

如果编辑器处于活跃状态（文件处于打开状态），当焦点不在编辑器时，按下 Esc 键可以让焦点从任何其他工具窗口返回活跃的编辑器。

6. 工具条

工具条（见图 1-76）是用来放置工具的，单击后可以展开工具窗口（详细介绍参见 1.14 节）。Android Studio 中的工具条分布在主界面的左右两边和底部（状态栏上面）。可以通过菜单栏→View→Tool Windows→查看Android Studio支持的所有工具列表。

图 1-76　工具条

7. 状态栏

状态栏通常在界面的最底部，主要显示 Android Studio当前的状态和执行任务，如图 1-77 所示。

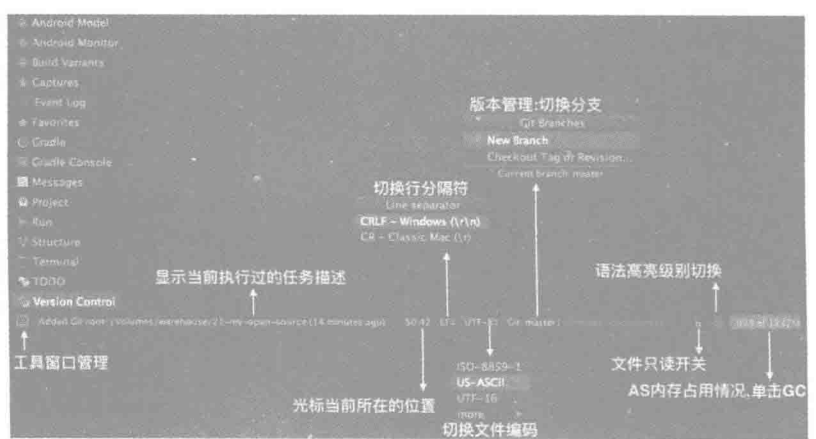


图 1-77

1.14 认识工具窗口

工具窗口寄居在工具条上，通过一定的规则显示和隐藏，主要提供某个工具的操作功能。下面介绍一下Android Studio中常用的工具窗口。

1. 项目工具窗口：Project

项目工具窗口提供了多种视图模式来查看项目结构，在项目工具窗口中可以对项目中的文件和目录进行各种操作，如图 1-78 所示。

图 1-78

2. 收藏夹工具窗口：Favorites

在使用Android Studio的日常编码中，如果某个文件或某段代码是我们经常需要查看或使用的，就可以将其添加到收藏夹中，以便快速查看，如图 1-79 所示。从中可以看到收藏夹中支持收藏项目中的文件、书签和断点。

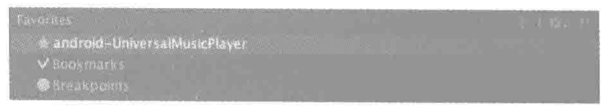


图 1-79

3. 结构工具窗口：Structure

结构工具窗口会以树状形式展现文件中元素的层次结构，如图 1-80 所示。单击元素可以跳转到编辑器中对应代码的位置。

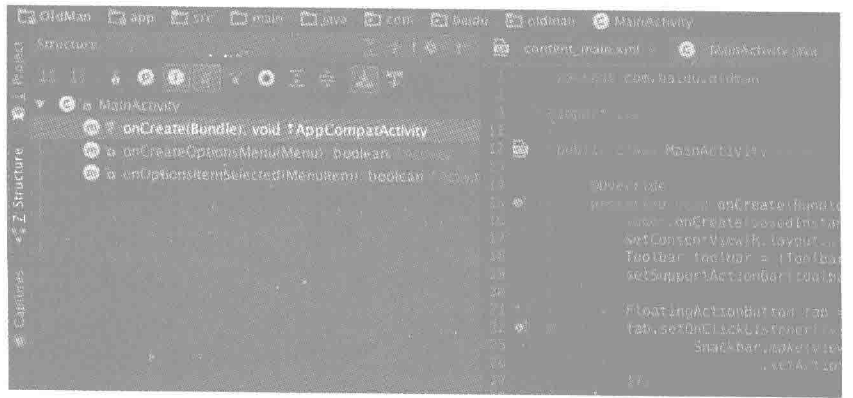


图 1-80

4. 待处理任务工具窗口：TODO

在Android Studio中待处理的任务可以在注释中使用TODO来标识。它表明了这个地方是需要注意的，有可能是未完成的功能，也有可能是需要解决的BUG或者需要优化的代码。在项目开发的过程中，团队成员们可以通过TODO来关注这些问题，以便更好地完成开发。

例如，在下面这个方法中添加TODO注释（见图 1-81）。打开待处理任务工具窗口TODO就会显示所有待处理的任务（见图 1-82）。

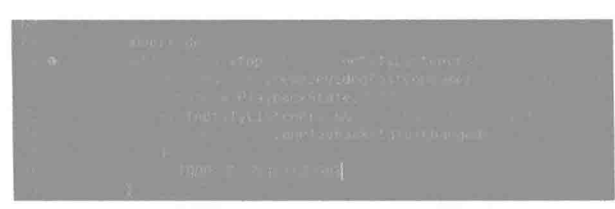


图 1-81

图 1-82

5. Android监控工具窗口：Android Monitor

Android监控工具窗口提供了非常丰富的工具来查看日志、截图、录屏、监控性能（CPU\Memory\Network\GPU），如图 1-83 所示。

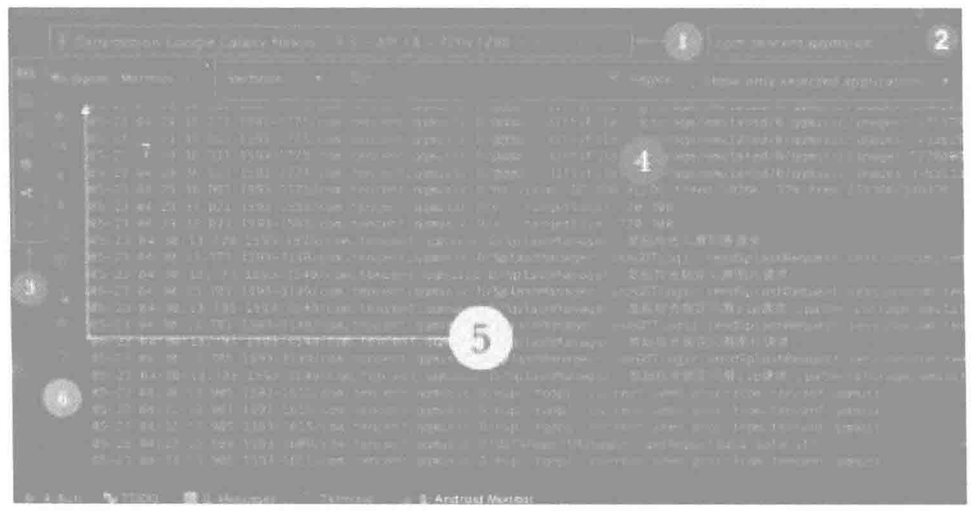


图 1-83

❶ 已连接到电脑上的设备列表，在这里选择我们需要使用的设备。

❷ 手机上运行的App的进程列表。

❸ 针对手机操作的工具栏，从上到下依次是截图、录像、系统信息、终止应用、布局解析、帮助工具。

❹ Logcat日志过滤工具栏，从左到右依次是日志级别、关键字、是否使用正则匹配、过滤配置。

❺ Logcat日志输出面板。
❻ Logcat日志工具栏。
❼ 内存、CPU、网络、GPU监控。

6. 快照工具窗口：Captures

快照工具窗口中存放Android Monitor中dump处的heap、allocation、系统信息、布局解析等文件。Android Studio为每一种文件都提供了相应的解析器，可以直接打开进行分析，如图 1-84 所示。

7. 运行工具窗口：Run

运行工具窗口主要显示Android Studio的运行过程，如图 1-85 所示。

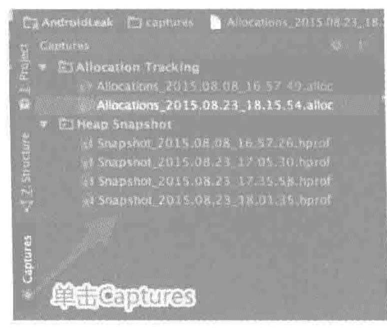


图 1-84

图 1-85

8. 终端工具窗口：Terminal

在终端工具窗口中可以直接执行终端命令，使用起来非常方便，如图 1-86 所示。

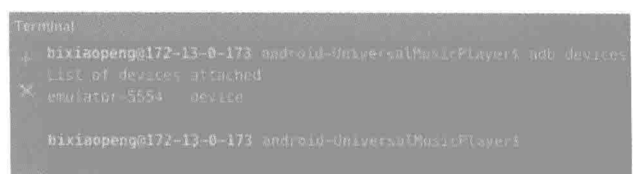


图 1-86

9. 构建变体工具窗口：Build Variants

构建变体工具窗口用来选择当前多渠道版本的构建，执行Build APK会构建出选中的渠道版本，如图 1-87 所示。

10. Gradle工具窗口：Gradle project

Gradle工具窗口列出了当前项目和模块中支持的所有Gradle任务和运行配置，以便快速操作，如图 1-88 所示。

图 1-87

图 1-88

11. 版本控制工具窗口：Version Control

版本控制工具窗口提供了版本控制的操作功能（Git/SVN），如图 1-89 所示。

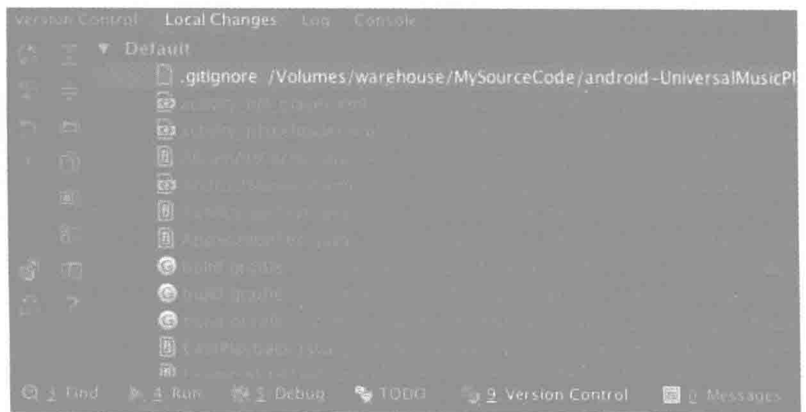


图 1-95

关于这些工具的具体用法，我们会在后续的章节中详细介绍。

第 2 章　项目与模块

Android Studio提供了非常方便的功能来新建和导入项目、模块、文件以及各种资源，集成了非常丰富的模板来帮助我们快速创建不同类型的文件。熟悉这些模板和工具的使用，会大大提高我们的开发效率。

本章将向大家介绍如何在Android Studio中新建和导入项目、模块、文件。

本章重要知识点 >>>>>>>>>>

- Android Studio 的项目结构；
- 如何新建、导入、删除项目和模板；
- 如何创建资源、图标、Activity、Fragment、Service 等文件。

2.1　Android Studio 的项目结构

2.1.1　项目和模块

Android Studio中有两个重要的概念，即项目（Project）和模块（Module）　。

1. 模块（Module）

Module是一个可以单独运行和调试的application或公共库，相当于Eclipse当中的Project，如图 2-1 所示。

我们通常把一些公共的代码放到Module当中，当你的APP需要添加一些依赖的库或者需要依赖一些公共的项目时可以导入Module。

每个Module中都有自己的build.gradle文件，用来配置模块的构建任务。

在build.gradle文件中我们可以配置SDK版本、构建工具版本、应用程序版本、打包参数、模块的依赖等。

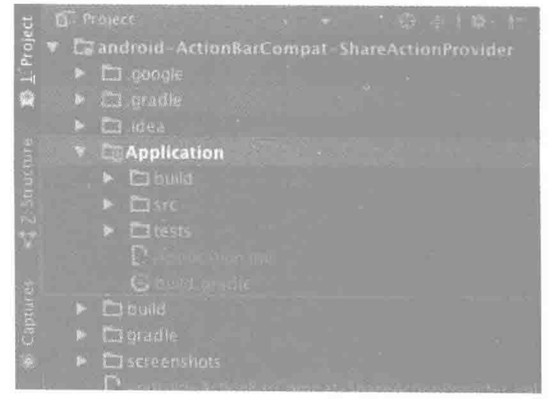


图 2-1

2. 项目（Project）

Project可以理解为一个完整的APP项目，由application module和一些依赖的module组成，相当于Eclipse当中的workspace，如图 2-2 所示。

一个Project中有多个Module。

Project中也有一个build.gradle文件，用来指定构建的项目和任务。导入或新建一个Module时，build.gradle将会自动更新。

图 2-2

3. 项目 VS 模块

- Module 相当于 Eclipse 当中的 Project。
- Project 相当于 Eclipse 当中的 workspace。
- 一个 Project 中可以包含多个 Module。
- Module 中的 build.grade 用于配置模块的构建任务。
- Project 中的 build.grade 用于指定构建的项目和任务。

4. Eclipse VS Android Studio

Eclipse和Android Studio项目中相关名字的对比如表 2-1 所示。

表 2-1

Eclipse	Android Studio	Eclipse	Android Studio
Workspace	Project	Classpath variable	Path variable
Project	Module	Project dependency	Module dependency
Project-specific JRE	Module JDK	Library	Module library
User library	Global library		

接下来我们认识一下Android Studio项目中基本的项目结构。

2.1.2 基本的项目结构

Android Studio中提供了多种视图的查看模式。Project模式会展示全部文件信息，文件的位置是真实的物理结构，因此在查看文件的时候建议切换到Project模式。

图 2-3 所示为基本的目录结构。

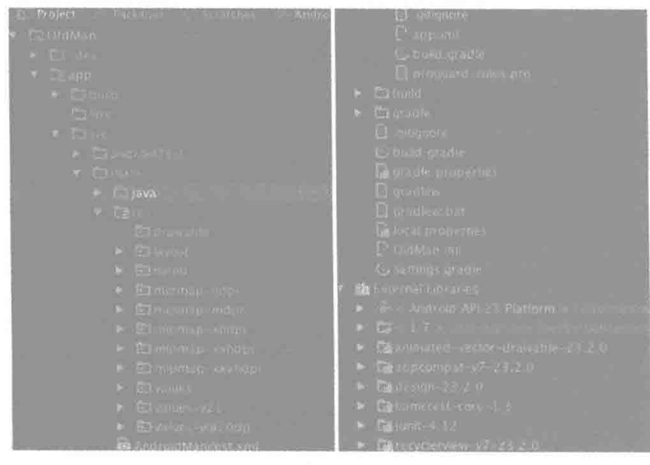


图 2-3

目录结构说明如表 2-2 所示。

表 2-2

文件/目录名	说明
OldMan	项目（Project）
OldMan/.idea	自动生成的用于存放Android Studio配置文件的目录，包括版权、检查配置、jar包信息、项目名、编译、编码、gradle、模块等
OldMan/app	项目中的模块（Module）
OldMan/app/build	模块编译后的文件存放目录
OldMan/app/libs	模块依赖的jar包存放目录
OldMan/app/src/androidTest	Android单元测试代码存放目录
OldMan/app/src/test	本地单元测试代码存放目录
OldMan/app/src/main	代码和资源存放目录
OldMan/app/src/main/java	代码目录
OldMan/app/src/main/res	资源目录
OldMan/app/src/main/res/drawable	存放能转换为绘制资源的位图文件或定义了绘制资源的XML文件
OldMan/app/src/main/res/layout	存放定义了用户界面布局的XML文件
OldMan/app/src/main/res/menu	存放定义了应用程序菜单资源的XML文件
OldMan/app/src/main/res/mipmap	存放启动图标目录
OldMan/app/src/main/res/values	存放定义了多种类型资源的XML文件
OldMan/app/src/main/jniLibs	so文件存放目录
OldMan/app/src/main/assets	附加的资源文件存放目录
OldMan/app/src/main/res/AndroidManifest.xml	应用程序配置文件
OldMan/app/.gitignore	模块中Git忽略配置文件
OldMan/app/app.iml	模块配置文件
OldMan/app/build.gradle	模块构建配置文件
OldMan/app/proguard-rules.pro	代码混淆配置文件
OldMan/build	项目编译目录
OldMan/gradle	gradle目录
OldMan/.gitignore	项目中Git的忽略配置文件
OldMan/build.gradle	项目构建配置文件
OldMan/gradle.properties	gradle配置文件
OldMan/gradlew	gradlew配置文件
OldMan/gradlew.bat	Windows上的gradlew配置文件

（续表）

文件/目录名	说明
OldMan/local.properties	属性配置文件
OldMan/settings.gradle	全局配置文件
External Libraries	项目中使用到的依赖库存放目录
External Libraries/< Android API 23 Platform >	Android SDK版本和存放路径
External Libraries/< 1.7 >	JDK版本和存放路径

2.2 导入项目和模块

2.2.1 导入Android Studio项目

操作步骤：菜单栏→File→New→Import Project或者欢迎界面→Import project，然后会弹出项目选择对话框，如图 2-4 所示。

Android Studio项目都会显示Android Studio图标，选择项目名字或者build.gradle就可以将项目导入。

2.2.2 导入Eclipse项目

第 1 步：选择要导入的Eclipse项目。

从菜单栏选择File→New→Import Project，或者从欢迎界面中选择Import project，然后会弹出项目选择对话框，如图 2-5 所示。

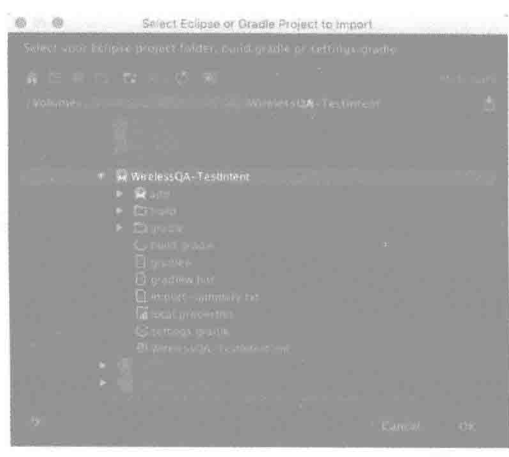


图 2-4

图 2-5

本例中只有一个工程，所以直接导入就可以了。如果项目中有多个依赖工程，需要导入主工程。

第 2 步：指定项目存放路径。

导入项目会复制原来的工程到指定的目录下，原来的Eclipse项目会被保留，不会有改动。

所以需要指定工程副本放到哪个目录下面，如图 2-6 所示。

指定目录→单击【Next】→弹出转换配置对话框，如图 2-7 所示。

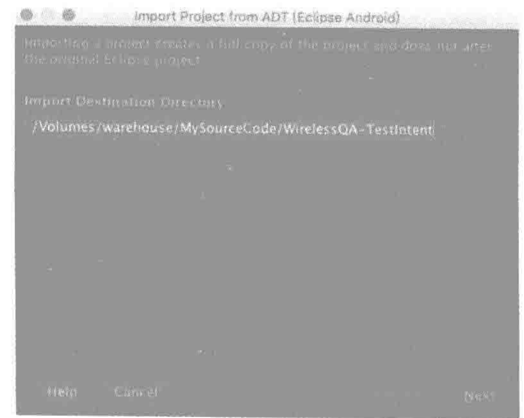


图 2-6

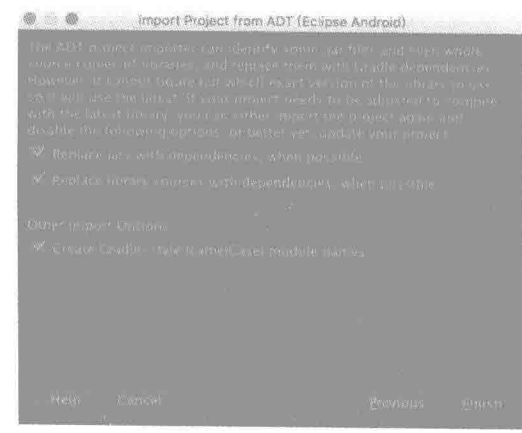


图 2-7

第 3 步：转换配置。

在导入ADT项目时，Android Studio能识别出jar文件和整个库的资源副本文件，并且可以把这些文件替换为Gradle依赖。但是Android Studio无法识别出确切的版本，因此会将这些文件替换为最新版本。所以有了下面的选项：

- Replace jars with dependencies，when possible：在可能的情况下，使用依赖替换 jars。
- Replace library sources with dependencies，when possible：在可能的情况下，使用依赖替换库资源。
- Create Gradle-style（camelCase）module names：创建 Gradle 风格（驼峰式）的模块名。

这些选项默认全部勾选，可以根据实际情况做出调整。

单击【Finish】后开始导入项目（请耐心等待，Android需要下载一些依赖）。

第 4 步：导入摘要。

项目导入成功后会自动打开import-summary.txt文件，如图 2-8 所示。

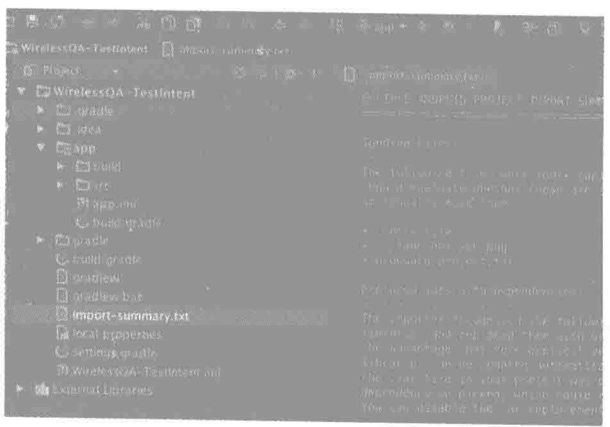


图 2-8

import-summary.txt文件中记录了项目导入的一些摘要，我们可以通过这份摘要看出Eclipse项目是如何被转换成Android Studio所需要的Gradle项目的。

```
ECLIPSE ANDROID PROJECT IMPORT SUMMARY
======================================
Ignored Files:
--------------
The following files were *not* copied into the new Gradle project; you
should evaluate whether these are still needed in your project and if
so manually move them:

* .checkstyle
* ic_launcher-web.png
* proguard-project.txt
Replaced Jars with Dependencies:
--------------------------------
The importer recognized the following .jar files as third party
libraries and replaced them with Gradle dependencies instead. This has
the advantage that more explicit version information is known, and the
libraries can be updated automatically. However, it is possible that
the .jar file in your project was of an older version than the
dependency we picked, which could render the project not compileable.
You can disable the jar replacement in the import wizard and try again:

android-support-v4.jar => com.android.support: support-v4: 18.0.0

Moved Files:
------------
Android Gradle projects use a different directory structure than ADT
Eclipse projects. Here's how the projects were restructured:

* AndroidManifest.xml => app/src/main/AndroidManifest.xml
* assets/ => app/src/main/assets
* res/ => app/src/main/res/
* src/ => app/src/main/java/

Next Steps:
-----------
You can now build the project. The Gradle project needs network
connectivity to download dependencies.
Bugs:
-----
If for some reason your project does not build, and you determine that
it is due to a bug or limitation of the Eclipse to Gradle importer,
please file a bug at http://b.android.com with category
Component-Tools.
(This import summary is for your information only, and can be deleted
after import once you are satisfied with the results.)
```

简单翻译为：

```
Eclipse Android 项目导入摘要
======================================
忽略文件：
---------------
下面的文件没有被复制到新的 Gradle 项目；如果你的项目中仍然需要用到它们，请手动复制过来：
* .checkstyle
* ic_launcher-web.png
* proguard-project.txt

使用依赖替换的 jar 文件：
--------------------------------
导入时识别出下面的这些 jar 文件是三方库，并且使用 Gradle 依赖替换。这样做的好处是可以知道库的确切版本信息，并且这些库能够自动更新。当然选择的 jar 包也有可能是旧版本，这可能会导致项目编译失败。所以你可以禁用 jar 导入向导，再试一次替换：
android-support-v4.jar => com.android.support: support-v4: 18.0.0

移动文件：
------------
Android Gradle 项目与 ADT Eclipse 项目使用不同的目录结构，项目是这样被重组的：
* AndroidManifest.xml => app/src/main/AndroidManifest.xml
* assets/ => app/src/main/assets
* res/ => app/src/main/res/
* src/ => app/src/main/java/
下一步：
-----------
你现在可以构建项目了，Gradle 项目需要联网来下载依赖.

Bug：
-----
如果你确认因为 Gradle 导入 Eclipse 项目导致你的项目无法构建
请提 BUG 到 http://b.android.com 的 Component-Tools.
import-summary.txt 文件仅是一些导入信息，你可以删除.
```

第 5 步：解决错误。

项目导入难免遇到错误，当遇到错误时请不要着急，仔细查看Android Studio报错信息，一般都能够解决。

【实例演示】

导入一个很旧的Eclipse项目后报某个Android SDK版本找不到，在Message工具窗口中会有相关的错误信息提示。

解决办法有两个，一是下载对应的SDK版本，二是修改build.grade中的SDK版本。修改完成后再构建项目，查看是否解决问题。

更多Gradle项目操作技巧请查看第 11 章。

2.2.3 导入Android示例代码

Android在GitHub上开源了很多示例代码，从Android Studio可以直接查看并导入这些代码。

01 进入示例代码选择界面。从欢迎界面选择，import an Android code sample 或者从菜单栏中选择 File→New→Import Sample，然后打开示例代码选择界面，如图 2-9 所示。

02 选择一个示例代码。查看说明和截图，确认是自己想要的，单击【Next】按钮。

03 设置应用程序名和项目存放位置（见图 2-10）。

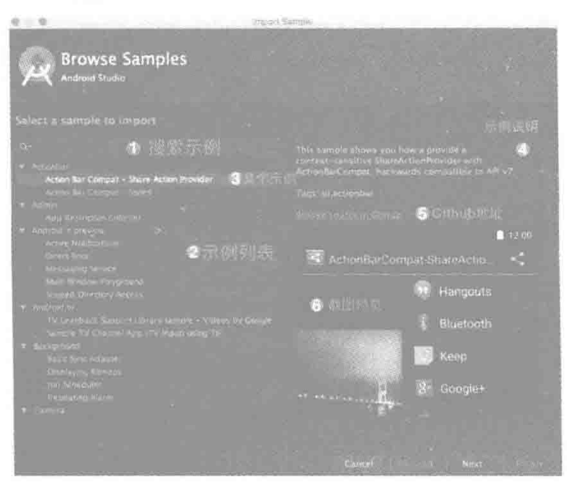


图 2-9

图 2-10

确定后开始进行下载、导入、构建等一系列操作，如图 2-11 所示。最后，示例代码导入成功。

04 运行应用程序并查看效果，如图 2-12 所示。

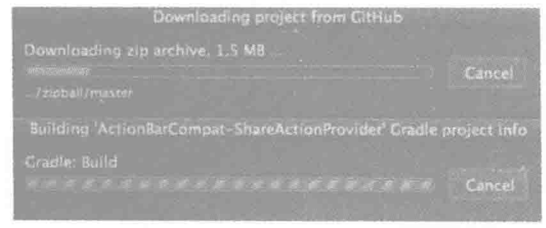


图 2-11

图 2-12

2.2.4 导入模块

从菜单栏中选择File→New→Import Module→弹出选择本地模块对话框→选择或输入本地模块的路径，如果有错误就会有相应的提示。

（1）模块已存在（见图 2-13）。

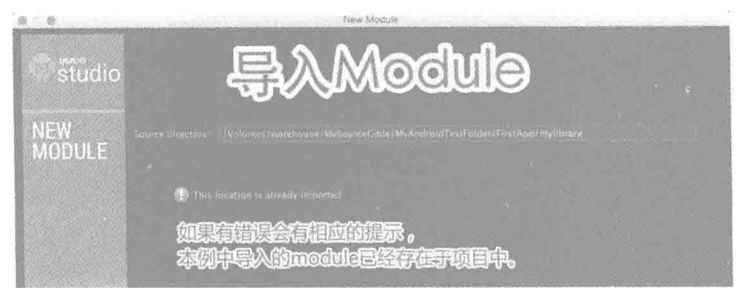


图 2-13

（2）路径不存在（见图 2-14）。

图 2-14

（3）导入的模块跟项目中的模块名冲突（见图 2-15）。

图 2-15

可以给冲突的模块重命名，如图 2-16 所示。

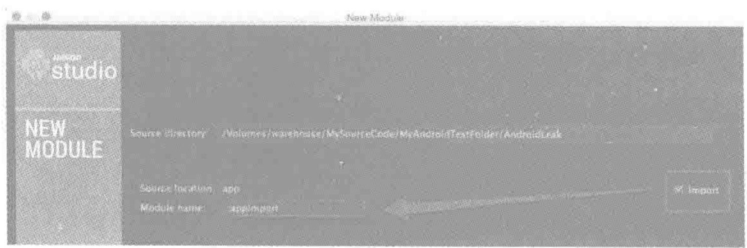


图 2-16

最后导入成功，如图 2-17 所示。

图 2-17

2.2.5 导入.JAR/.AAR

.aar格式的包是Android独有的第三方库（Android Library），包含可重用的Java文件和Android组件。我们可以通过新建Module创建自己的Android Library，然后打包成.aar格式的包与别人共享（使用方法跟jar包基本一样）。

从菜单栏中选择File→New→New Module…→Import .JAR/.AAR Package，如图 2-18 所示。

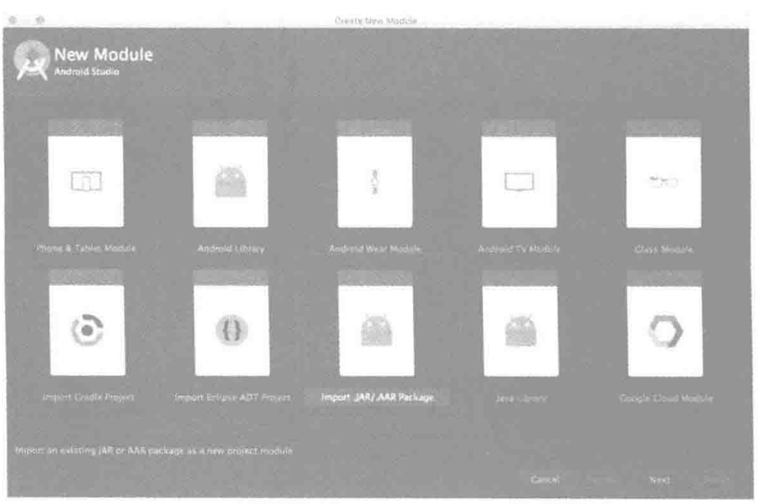


图 2-18

选择aar包的存放路径，并为子项目命名，如图 2-19 所示。

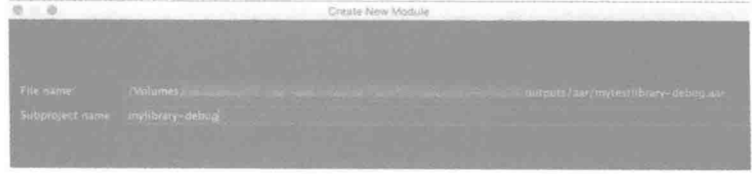


图 2-19

完成后可以看到作为模块导入的.aar文件，如图 2-20 所示。

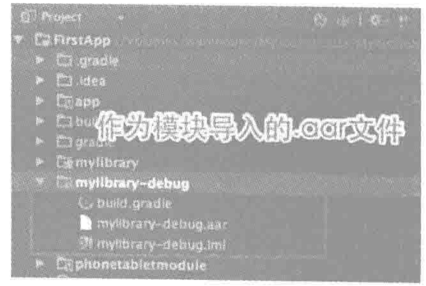


图 2-20

mylibrary-debug的构建文件build.gradle：

```
configurations.create ("default")
artifacts.add ("default", file
('mylibrary-debug.aar'))
```

Project 的 settings.gradle 文件中自动添加了 mylibrary-debug模块：

```
include ': app', ': phonetabletmodule', ': mylibrary', ': mylibrary-debug'
```

将新建的模块添加为应用程序模块的依赖，方法同普通依赖，如图 2-21 所示。

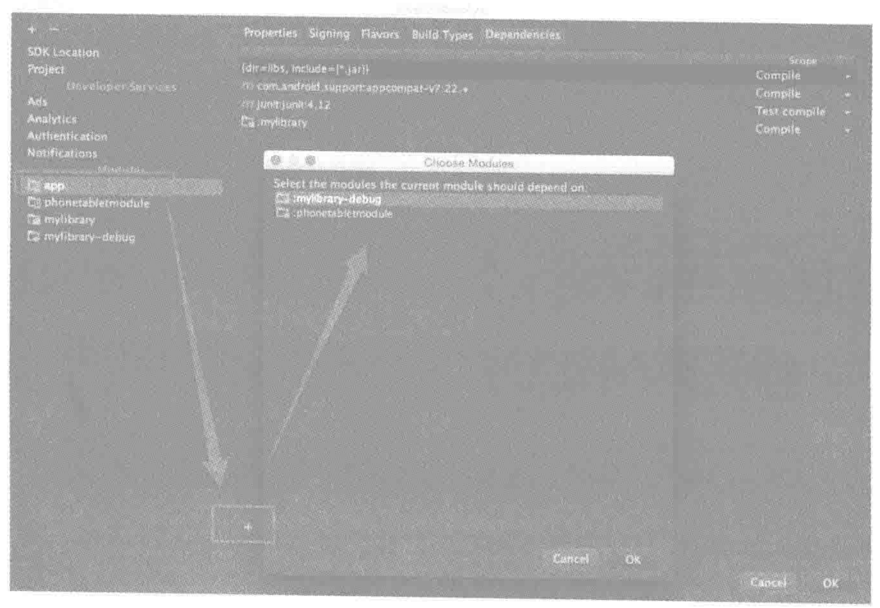


图 2-21

2.2.6 从VCS检出项目

第 1 步： 选择从GitHub中检出项目。

欢迎界面： Check out project from Version Control→GitHub 。

菜单栏： File→New→Project From Version Control→GitHub。

第 2 步： 输入Master Password。

如果是第一次使用GitHub，会要求我们输入 Master Password，如图 2-22 所示。

Master Password 是用来保护存储在 Android Studio数据库中的密码。如果忘记了密码是无法进行到下一步的，如图 2-23 所示。

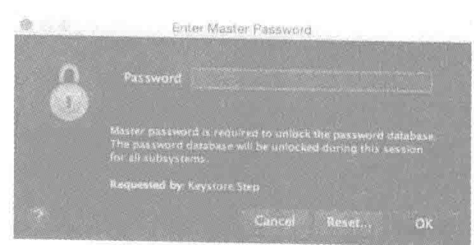


图 2-22

该怎么解决这个问题呢？单击【Reset...】→弹出重置Master密码窗口→输入新的密码→OK，如图 2-24 所示。

图 2-23

图 2-24

单击【Yes】确认重置操作，如图 2-25 所示。

如果记得密码，就直接输入。

第 3 步：输入GitHub账号和密码，单击【Login】，如图 2-26 所示。

图 2-25

图 2-26

第 4 步：克隆GitHub上的项目，如图 2-27 所示。

图 2-27

克隆前可以测试一下项目是否存在，如图 2-28 所示。

图 2-28

第 5 步：单击【Clone】后开始下载源码。

2.3　创建项目和模块

2.3.1　创建项目

第 1 步：打开新建项目向导。

在欢迎界面选择Start a new Android Studio project，或者在菜单栏中选择File→New→Create New Project，弹出创建新项目向导。

第 2 步：创建一个新项目，如图 2-29 所示。

单击【Next】按钮，进入选择目标设备界面。

图 2-29

第 3 步：选择目标设备。

在这里列出了不同外形的设备：手机和平板、手表、电视、车载、眼镜，我们需要确认自己的APP准备运行在哪些设备上，如图 2-30 所示。

图 2-30

- Minimum SDK：我们选择不同版本的SDK，设备覆盖率会有相应的变化，SDK 版本越低，设备覆盖率就越高。由于 Android 碎片化严重，因此如果想支持更多的设备，就不得不选择低版本的 SDK，如图 2-31 所示。

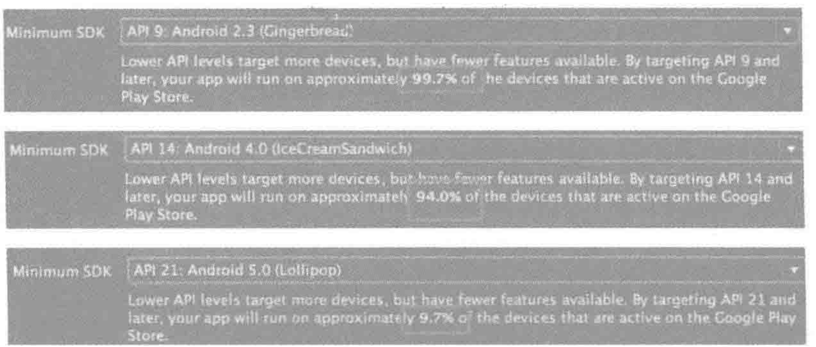


图 2-31

- help me choose：单击【help me choose】可以查看 Android 版本更多信息，以便帮助我们选择合适的 SDK 版本，如图 2-32 所示。

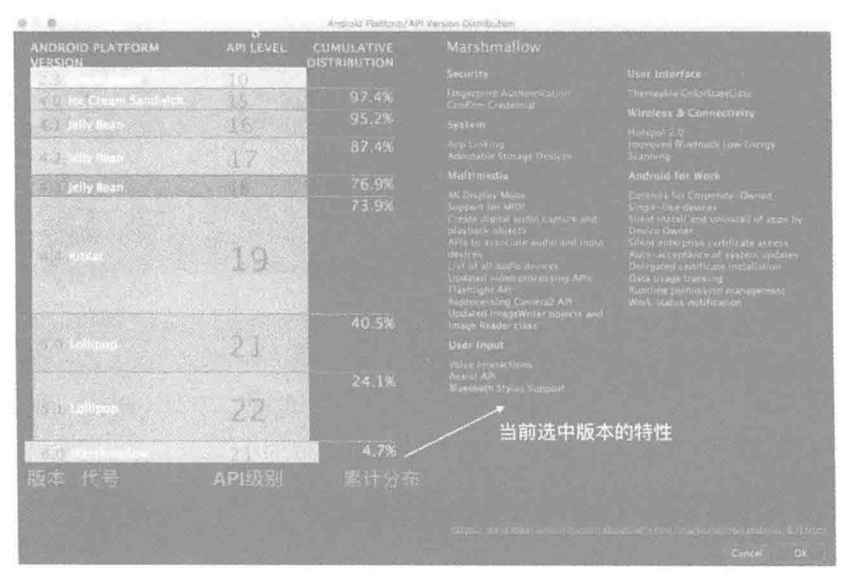


图 2-32

回到第 2 步的操作界面，继续单击【Next】按钮，进入模板选择界面。

第 4 步：选择一个模板。

Android Studio为我们提供了常用的Activity模板，可以使用模板来加快开发效率，如图 2-33 所示。

单击【Next】按钮，进入自定义Activity界面。

第 5 步：自定义Activity。

Activity文件名和资源都是自动生成的，如果不想使用默认的可以自己修改，如图 2-34 所示。

单击【Finish】按钮，开始创建项目。

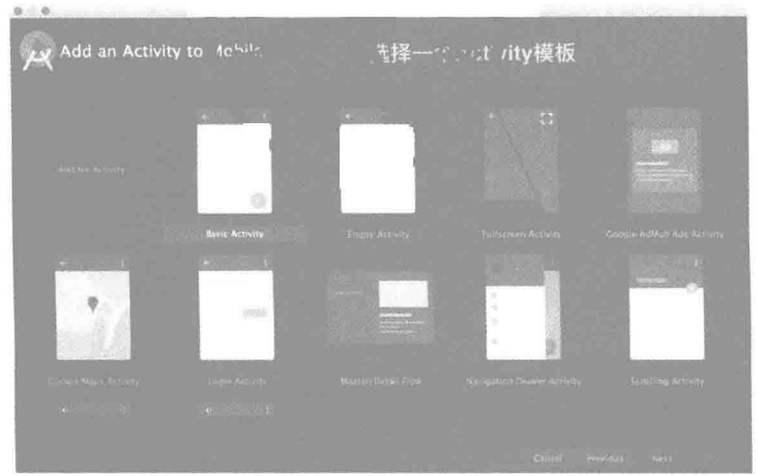


图 2-33

图 2-34

第 6 步：开始创建项目，如图 2-35 所示。

如果是第一次使用Android Studio创建项目，可能会比较慢，因为Android Studio第一次会先下载Gradle。

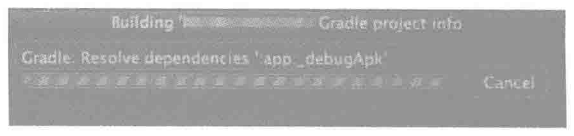


图 2-35

第 7 步：解决错误（如果有的话）。

耐心等待一会儿，项目创建成功后可能有错误提示，如图 2-36 所示。

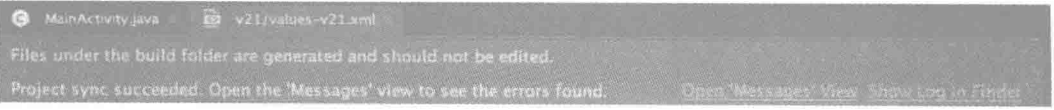


图 2-36

不要怕，Android Studio会给出相应的解决方案，单击【Open Messages View】可打开错误提示界面。

Messages Gradle Sync
/Volumes/warehouse/MySourceCode/MyAndroidTestFolder/FirstApp/app/build.gradle
Failed to resolve: com.android.support:appcompat-v7:22.+
Error:(23, 13) Install Repository and sync project
Show in File
Show in Project Structure dialog

图 2-37

图 2-37 所示的错误是com.android.support：appcompat-v7：22.+解析失败。

看一下这个jar包有什么用，单击【Show in Project Structure dialog】，打开项目结构配置界面，如图 2-38 所示。

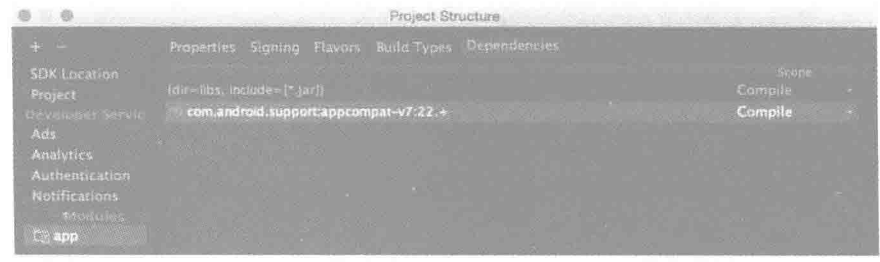


图 2-38

从图 2-38 中可以看出，这个jar包是编译的时候用到的。

OK，那我们安装这个组件，单击【Install Repository and sync project】，安装过程如图 2-39 所示。

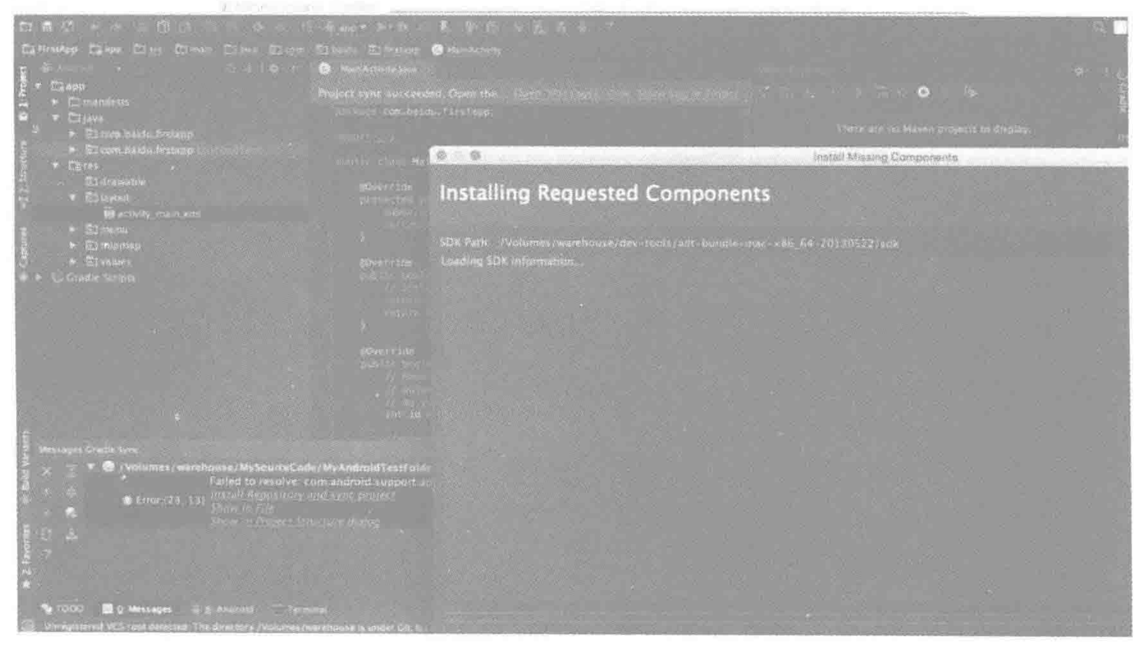


图 2-39

单击【Finish】按钮完成安装，如图 2-40 所示。

第 8 步：运行。

安装成功后项目中不再有报错，可成功运行，如图 2-41 所示。

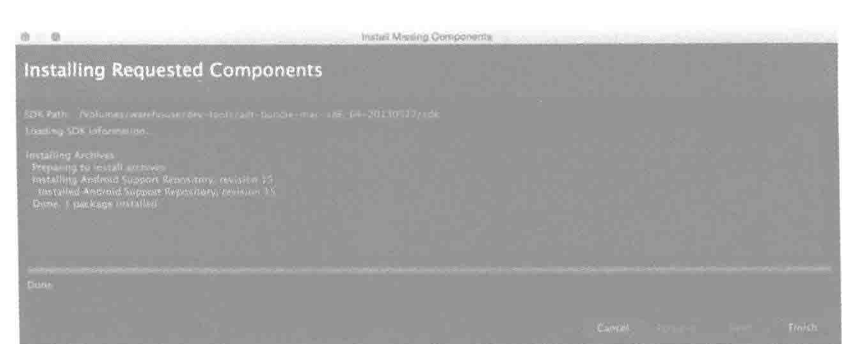


图 2-40

图 2-41

2.3.2 创建应用程序模块

第 1 步：在菜单栏中选择File→New→New Module，弹出创建模块界面，如图 2-42 所示。

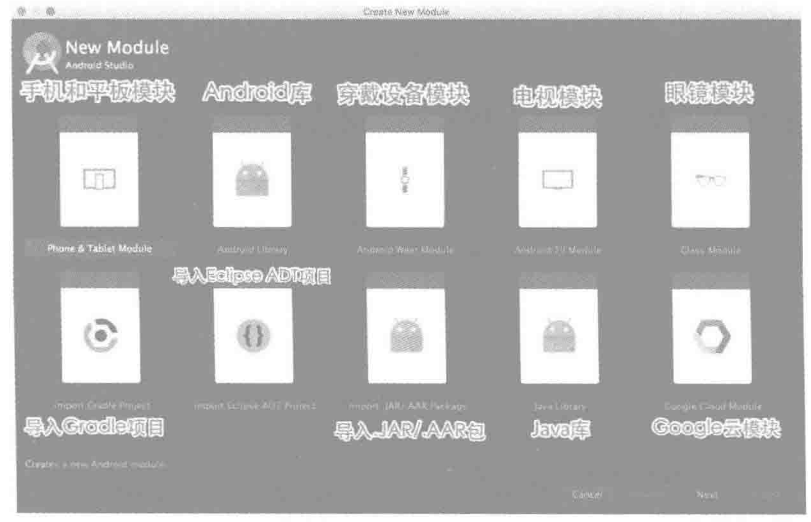


图 2-42

第 2 步：Android Studio中支持这么多种模块类型，我们要新建的模块类型为应用程序，因此要选择 【Phone & Table Module】。选择【Phone & Table Module】→Next→配置新的模块，如图 2-43 所示。

第 3 步：Next→选择一个Activity模板，如图 2-44 所示。（这里以Login Activity模板作为演示）。

第 4 步：Next→配置Activity。这里有对即将创建的Activity模板的说明：创建一个新的登录Activity，允许用户选择使用Google＋ 登录，或输入email地址和密码登录，或注册你的APP，如图 2-45 所示。我们这里使用默认配置。

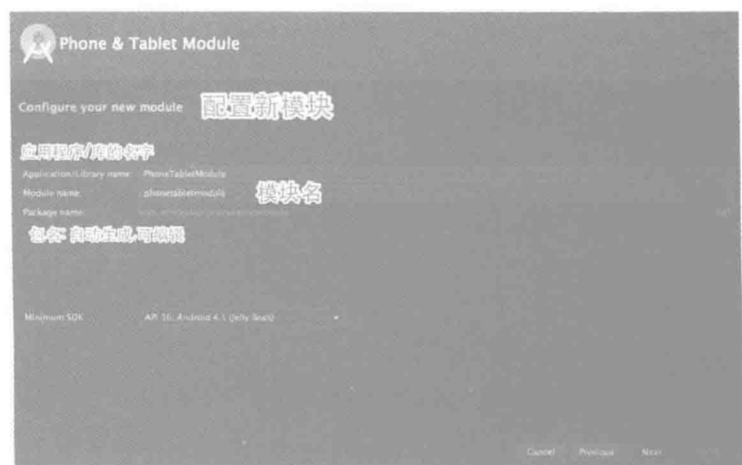


图 2-43

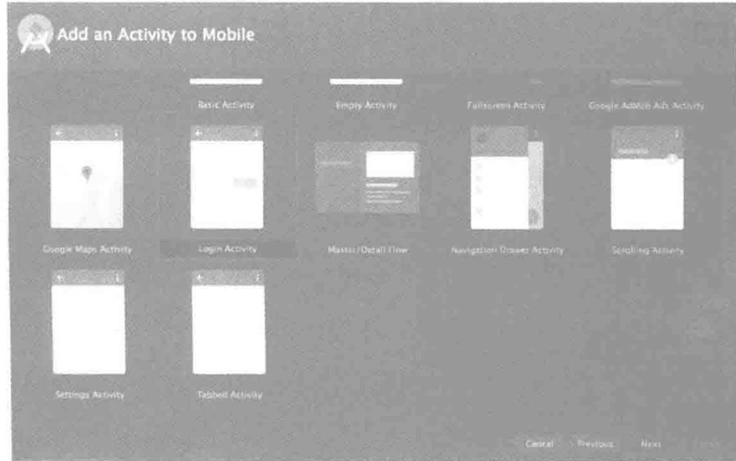


图 2-44

图 2-45

第 5 步：单击【Finish】→新建应用程序（Phone & Table Module）模块成功，如图 2-46 所示。

图 2-46

2.3.3 创建Android公共库模块

Java的公共库是直接将公用代码打成jar包，Android的公共库类似，目的都是为了代码的重用。使用Android公共库还可以使项目模块化，以便协同开发和更好地扩展。

1. 创建Android Library

在菜单栏中选择File→New→New Module，然后弹出New Module对话框，选择Android Library，如图 2-47 所示。

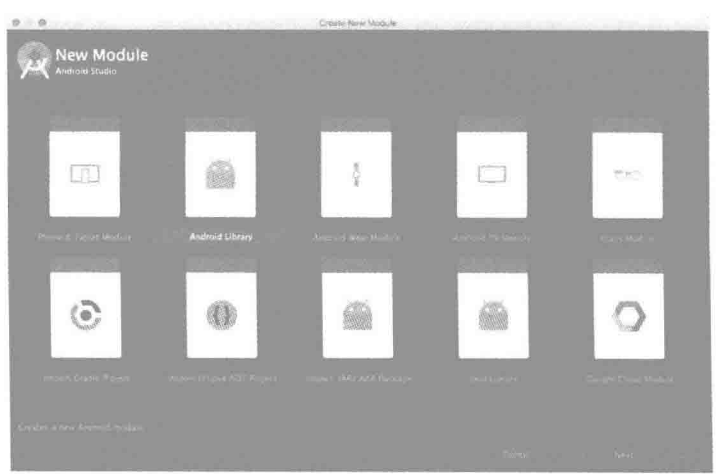


图 2-47

单击【Next】→输入Library name和Module name，如图 2-48 所示。

单击【Finish】按钮，结果如图 2-49 所示。

Android Library与Android Application的区别是有不同的标记和不同的插件。在build.gradle文件中可以看到不同的插件，Android Application的插件是apply plugin：'com.android.application'，Android Library的插件是apply plugin：'com.android.library'。

新建后settings.gradle自动添加include：

```
include ': app', ': phonetablemodule', ': mylibrary'
```

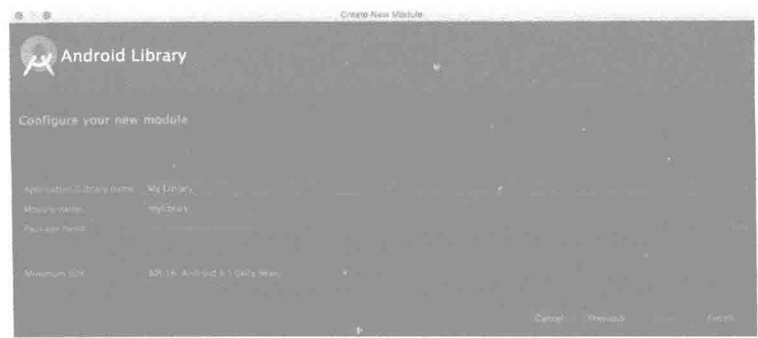


图 2-48

图 2-49

2. 为应用程序添加模块依赖

打开Project Structure→选择要操作的应用程序→添加已存在项目中的模块作为依赖，如图 2-50 所示。

图 2-50

依赖添加成功，在应用程序的build.gradle文件中会添加相应的依赖代码，如图 2-51 所示。

图 2-51

编译运行，如果有错（可能是资源冲突或配置冲突），就会有相应的提示。

3. 将Android Library打包成.aar

打开Gradle工具窗口，找到Android Library模块，在 build任务中双击assemble，如图 2-52 所示。任务执行成功以后，在mylibrary/build/outputs/aar目录下就会打出.aar格式的包。默认Debug和Release的AAR包都会打出来，当然也可以选择只打Debug的包，双击assembleDebug任务就可以了，只打Release的包与此相同。

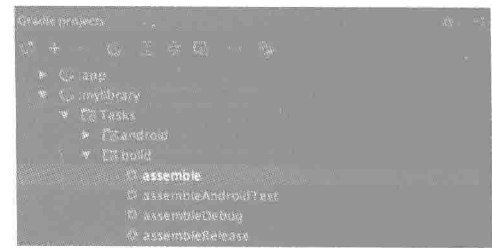


图 2-52

2.4 删除模块

文件和文件夹都是可以通过右击后在菜单列表中选择【Delete】删除的，但是右击模块却找不到删除这个选项。那么模块应该怎样删除呢？

打开Project Structure→选中要删除的module→单击左上角的“-”，如图 2-53 所示。

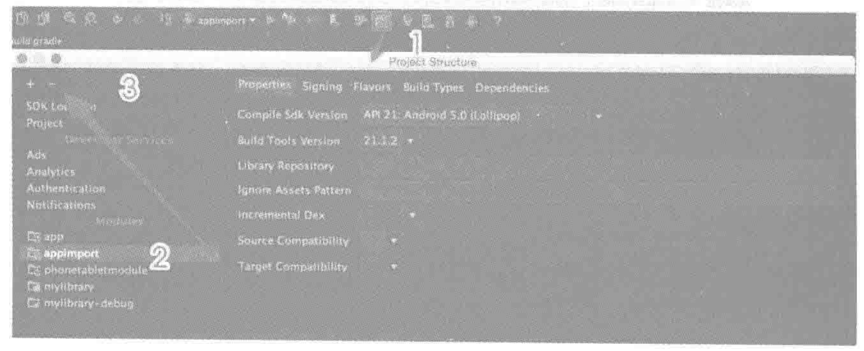


图 2-53

然后弹出确认删除对话框，如图 2-54 所示。

确定删除后，Gradle会进行同步，在项目的settings.gradle中会去掉删除的模块，本例中是appimport。此时模块并不是直接被删除了，而是需要我们再执行一次删除。在项目中找到appimport，右击后调出操作菜单，单击“Delete...”，如图 2-55 所示。

图 2-54

图 2-55

然后会弹出确认删除提示，确认删除后这个模块就真的从项目中删除了。

2.5 添加 so 文件

1. JNI是什么

Android系统的底层库是由C/C++编写，上层Android应用程序和应用程序框架通过JNI（Java Native Interface）调用底层接口。

Android使用JNI开发分两种情况：一是使用已经编译好的.so动态库；二是使用C/C++源代码开发。

一些第三方的库出于性能或代码安全的目的，会将核心代码用C/C++来实现，然后提供编译好的so文件或jar包给我们。

2. so文件是什么

Android中用到的so文件是一个C++的函数库，apk或jar包中调用so文件时，需要将对应so文件打包进apk。

3. 如何加载so文件

假设so文件为libBaiduMapSDK_v350_1.so，可以这样加载：

```
String libName = "BaiduMapSDK_v350_1";
        //库名 libName，没有前缀 lib 和后缀.so
System.loadLibrary ( libName ) ;
        //在使用前一定要先加载
```

4. so文件应该放在哪里

so文件在Eclipse中放在libs目录下，在Android Studio中则放在moduleName/src/main/jniLibs目录下（见图 2-56）。如果没有jniLibs目录，就新建一个。

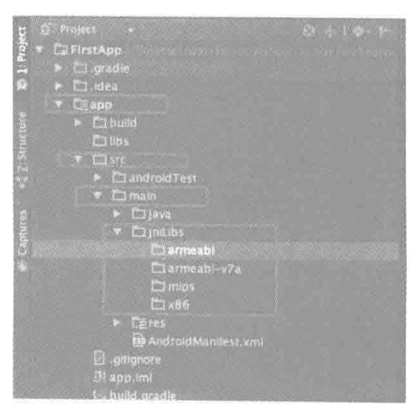


图 2-56

5. armeabi、armeabi-v7a、mips、x86 这四个文件夹是干什么的

它们表示四种不同的CPU类型，不同的CPU特性不一样，在使用so文件时要注意区分。可以直接复制so文件，粘贴到对应的文件夹下，如图 2-57 所示。单击【OK】按钮，so文件复制成功，如图 2-58 所示。

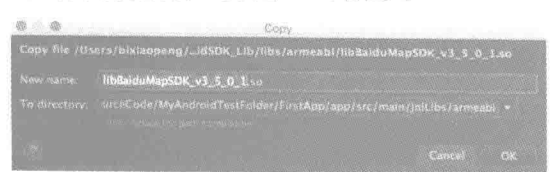


图 2-57

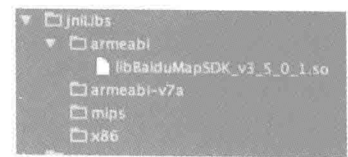


图 2-58

2.6 创建类和文件

1. 创建类文件

Android Studio创建类时可以选择多个常用的类模板。

（1）操作步骤

在菜单栏中选择File→New→Java Class→输入类名，选择种类，如图 2-59 所示。

（2）类的种类

普通类：

```
/**
 * 新建普通类
 * Created by bixiaopeng on 15/11/14.
 */
public class Demo {
}
```

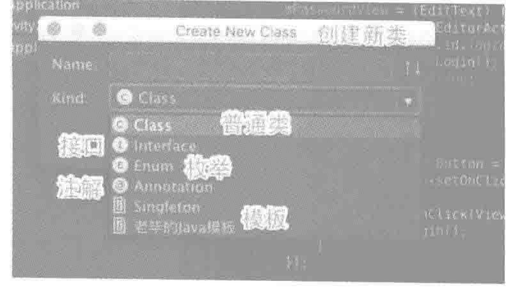


图 2-59

接口类：

```
/**
 * 新建接口类
 * Created by bixiaopeng on 15/11/14.
 */
public interface DemoInterface {
}
```

枚举类：

```
/**
 * 新建枚举类
 * Created by bixiaopeng on 15/11/14.
 */
public enum DemoEnum {
}
```

注解类：

```
/**
 * 新建注解类
 * Created by bixiaopeng on 15/11/14.
 */
public @interface DemoAnnotation {
}
```

（3）使用自定义模板文件

Android Studio中提供了保存文件模板的功能，可以把常用的文件保存为模板，在新建类时直接选择。

2. 新建其他格式的文件

可以直接新建各种格式的文件，有一些Android Studio支持的会直接新建成功。如果不支持，需要先指定默认打开文件的类型，只需要一次设置，以后就可以默认使用这个格式。

【实例演示】第一次新建markdown文件。

01 在菜单栏中选择 File→New→File→输入文件名和格式，如图 2-60 所示。

02 单击【OK】按钮。因为 Android Studio 无法识别 markdown 文件，需要指定一个默认的打开格式，我们在这里指定为 Text，如图 2-61 所示。

03 单击【OK】按钮设置成功，以后新建 markdown 文件都会默认使用 Text 格式打开。

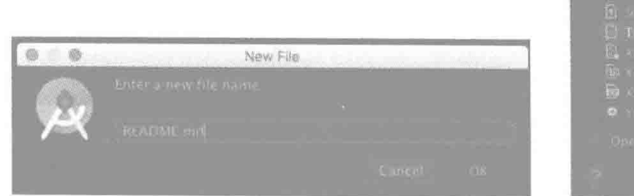


图 2-60

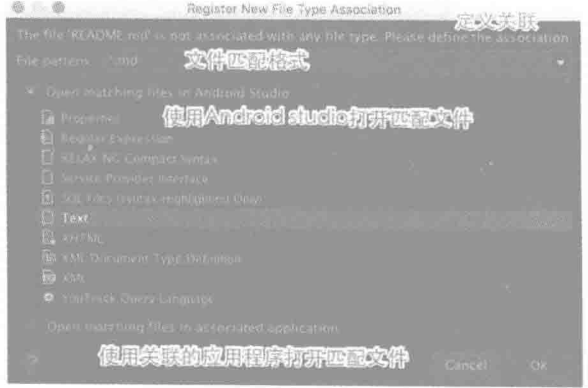


图 2-61

如果安装了markdown插件，那么默认会使用markdown插件打开。

2.7 创建 Activity

Android Studio提供了很多常用的Activity模板，这些模板能够自动创建Activity和布局文件，并可通过选项进行相应的配置，大大提高了开发效率。

2.7.1 Activity模板列表

如图 2-62 所示，列表中列出了所有Activity模板，选择后就会跳转到配置界面。

图 2-62

有一些模板没有办法使用，需要miniSdk大于等于 20 才可以用，现在修改一下miniSdk。在build.gradle中把miniSdk改为 20，如图 2-63 所示。再次查看列表，所有的模板就都可用了。

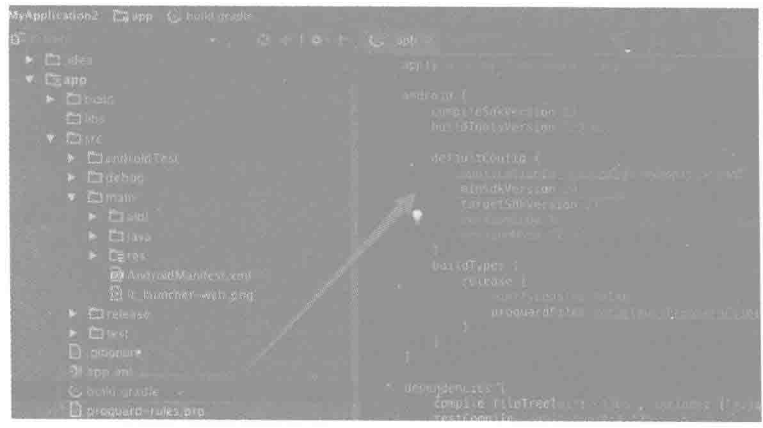


图 2-63

2.7.2 Activity Gallery

在Gallery中还可以查看每个Activity的效果图，如图 2-64 所示。

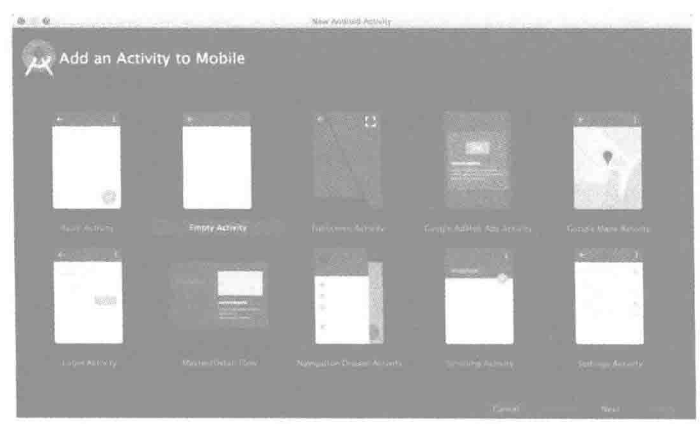


图 2-64

2.7.3 新建一个Activity

操作步骤：

第 1 步：选择一个模板。

在Add an Activity to Mobile界面中选择【Navigation Drawer Activity】→单击【Next】，弹出配置界面，如图 2-65 所示。

- Activity Name: Activity 名。
- Layout Name: Activity 对应的布局名。
- Title: Activity 的标题。
- Launcher Activity: 新建的 Activity 是否作为启动的 Activity，如果是就勾选，如果不是就不要勾选。

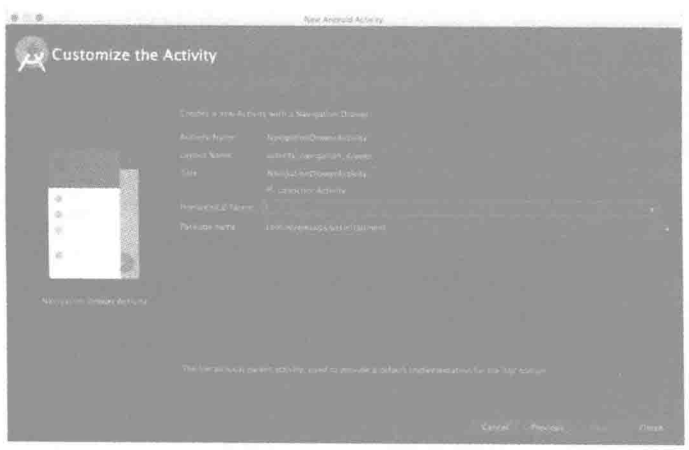


图 2-65

- Hierarchical Parent: 父层次。

我们可以选择已存在的Activity作为父层次，如图 2-66 所示。（当然也可以为空，本例就为空。）

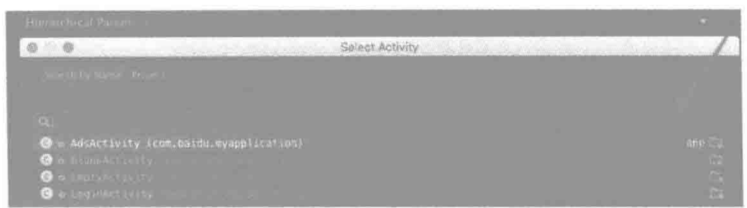


图 2-66

如果配置了父层次，在新建的Activity上按回退键会回到配置的父层次这里。

第 2 步：这里为了演示，勾选Launcher Activity，其他均使用默认配置，单击【Finish】按钮，NavigationDrawerActivity.java、activity_navigation_drawer.xml创建成功。

第 3 步：设置Launcher Activity并运行。

查看AndroidManifest.xml文件：

```
<activity
    android: name=".NavigationDrawerActivity"
    android: label="@string/title_activity_navigation_drawer">
    <intent-filter>
        <action android: name="android.intent.action.MAIN" />

        <category android: name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

因为我们在新建Activity时将NavigationDrawerActivity设置为Launcher Activity，所以在AndroidManifest.xml文件中添加了Launcher Activity，但是可能有多个Activity声明为Launcher Activity，需要指定一下。

第 4 步：打开运行和调试配置界面→app→General→Launch Options ，如图 2-67 所示。（Launch使用Default Activity作为Launcher Activity。）

第 5 步：选择Specified Activity，再选择Activity，如图 2-68 所示，单击【OK】按钮保存。

第 6 步：运行（见图 2-69）。

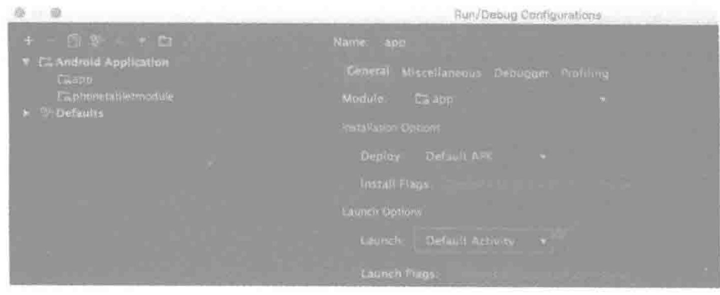


图 2-67

图 2-68

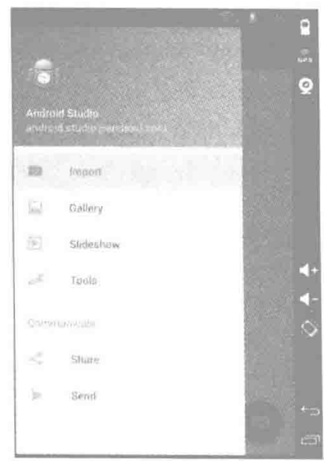


图 2-69　运行效果

2.8 创建 Fragment 文件

为了解决碎片化的问题，Android从Android 3.0 （API level 11）开始引入Fragment。我们可以把Fragment当成Activity界面的一部分，甚至是Activity的界面由完全不同的Fragment组成。Fragment具有自己的生命周期，不能独立存在，必须嵌入Activity中，而且Fragment的生命周期直接受所在的Activity的影响。

创建一个Fragment的一般步骤如下。

01 在 Layout 目录下创建 Fragment 的布局文件。

02 创建一个类文件，继承 Fragment 或者 Fragment 子类，并在 onCreateView 方法中加载布局文件。

03 在 Activity 的布局文件中声明 FragmentLayout。

使用模板基本上可以自动生成第 1 步和第 2 步。

如何使用Android Studio提供的模板快速创建Fragment及其所用到的布局文件呢？可以在菜单栏中选择File→New→Fragment，显示Fragment创建选项，如图 2-70 所示。

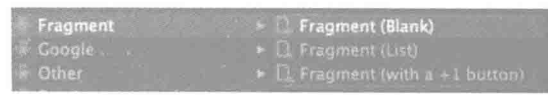


图 2-70

- Fragment（Blank）：创建一个空白的 Fragment。
- Fragment（List）：创建一个包含网格列表的 Fragment。
- Fragment（with a + 1 button）：创建一个带有 Google Plus +1 按钮的 Fragment。

1. 创建一个空白的Fragment

单击【Fragment（Blank）】→弹出【New Android Component】界面，如图 2-71 所示。

- FragmentName：自定义的 Fragment 类名，会继承 Fragment。本例中为 BlankFragment。
- Create layout XML?：如果勾选，就会同时创建 BlankFragment 类对应的布局文件，并在 BlankFragment 类中自动添加加载该布局文件的代码。
- Fragment Layout Name：BlankFragment 类对应的布局文件名，会根据类名自动生成，可自定义。
- Include fragment factory methods?：如果勾选，会在 BlankFragment 类中生成工厂方法。
- Include interface callback?：如果勾选，会在 BlankFragment 类中生成回调接口。

2. 创建一个包含网格列表的Fragment

单击【Fragment（List）】→弹出【New Android Component】界面，如图 2-72 所示。

- Package name：包名。
- Object Kind：对象类型。
- Fragment class name：Fragment 类的名字。
- Column Count：网格的列数。
- Object content layout file name：对象内容布局文件名。

- List layout file name：列表布局文件名。
- Adapter class name：Adapter 类名。

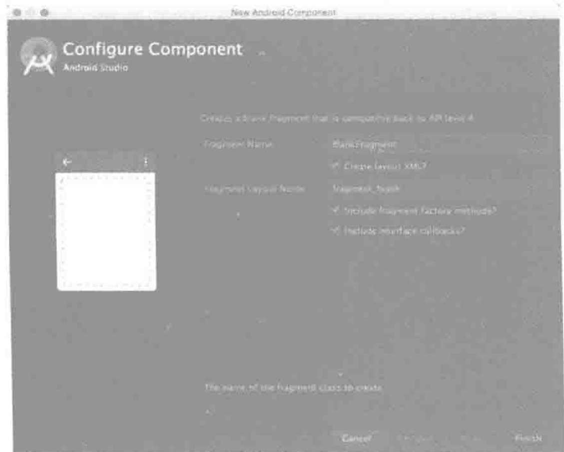


图 2-71

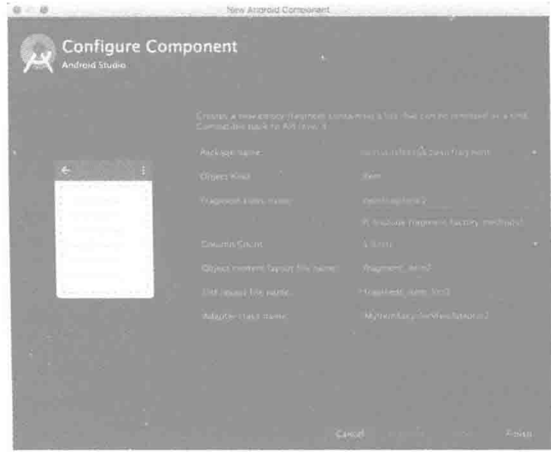


图 2-72

3. 创建一个带有Google Plus +1 按钮的Fragment

创建界面如图 2-73 所示。

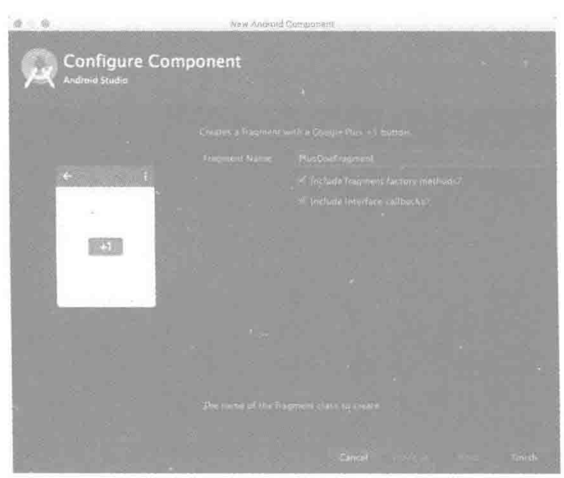


图 2-73

2.9 创建 Service 文件

Service是Android系统中的四大组件之一，只能运行在后台，不提供用户界面。

IntentService是继承于Service并处理异步请求的一个类。在IntentService内有一个工作线程来处理耗时操作。启动IntentService的方式和启动传统Service一样，当任务执行完毕后，IntentService会自动停止，而不需要手动控制。

Android Studio中提供了创建Service和Intent Service的模板，可以使用模板快速创建Service和Intent Service文件，并且在AndroidManifest.xml中自动配置。

2.9.1 创建Service文件

在菜单栏中选择File→New→Service，然后弹出【New Android Component】界面，如图2-74所示。

在这里可以创建一个Service组件，并添加到AndroidManifest.xml文件中。

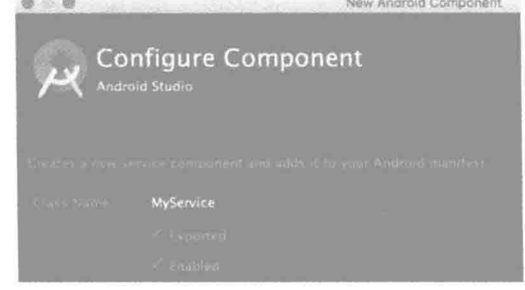


图 2-74

- Class Name：类名，会继承 Service。
- Exported：Service 的属性，表示是否支持其他应用调用当前组件。
- Enable：Service 的属性，表示该服务是否能够被实例化。

使用默认配置，然后创建成功。

类文件：

```
public class MyService extends Service {
    public MyService () {
    }

    @Override
    public IBinder onBind (Intent intent) {
        // TODO: Return the communication channel to the service.
        throw new UnsupportedOperationException ("Not yet implemented") ;
    }
}
```

AndroidManifest.xml：

```
<service
    android: name=".MyService"
    android: enabled="true"
    android: exported="true" />
```

2.9.2 创建Intent Service文件

操作步骤：在菜单栏中选择File→New→Service→Service(IntentService)，然后弹出【New Android Component】界面，如图2-75所示。

- Class Name：Intent Service 类名。
- Include helper start method：如果勾选，会生成一些帮助我们开始的方法。

创建一个新的Intent Service类，使用默认配置创建即可。

图 2-75

类文件如图 2-76 所示。

图 2-76

AndroidManifest.xml：

```
<service
    android: name=".MyIntentService"
    android: exported="false"></service>
```

2.10 创建自定义组件

有时Android提供的组件无法满足我们的需求，因此需要自定义组件。

创建自定义组件的一般步骤：

01 新建类文件，要继承 View 或 View 的子类。

02 覆写父类的一些方法。

03 使用自定义组件类。

Android Studio会通过模板自动生成文件并覆写方法，我们只需要根据自己的实际需求修修改即可。

在菜单栏中选择File→New→UI Component→Custom View，弹出【New Android Component】界面，如图 2-77 所示。使用默认配置，单击【Finish】按钮后创建成功。

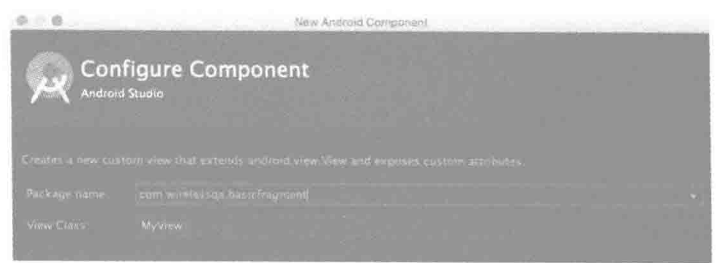


图 2-77

- Package name：包名。
- View Class：自定义的类名，按约定应该以 View 结尾。

MyView.java文件如图 2-78 所示。

模板帮助我们继承了View并覆写了方法，然后根据自己的需求修改代码即可。

另外，在layout目录下会自动生成一个sample_my_view.xml文件，这是一个示例，用来告诉你自定义组件怎么用，如图 2-79 所示。

图 2-78

图 2-79

咦，为什么自动生成的布局文件会报错？

原来这一行的写法在Eclipse中是可用的，但在Android Studio中被废弃了，所以需要转换一下。

如何转换呢？

单击报错这一行，弹出意图提示图标，单击图标，给出建议的解决方案，如图 2-80 所示。

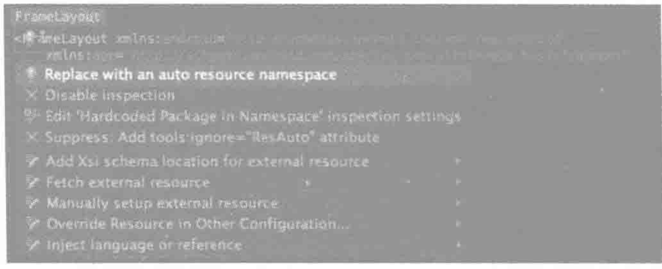


图 2-80

单击“Replace with an auto resource namespace”，报错这一行被替换为：

```
xmlns: app="http://schemas.android.com/apk/res-auto"
```

问题解决，最终代码是：

```
<FrameLayout xmlns: android="http://schemas.android.com/apk/res/android"
    xmlns: app="http://schemas.android.com/apk/res-auto"
    android: layout_width="match_parent"
    android: layout_height="match_parent">

    <com.wirelessqa.basicfragment.MyView
        android: layout_width="300dp"
        android: layout_height="300dp"
        android: background="#ccc"
        android: paddingBottom="40dp"
```

```
        android: paddingLeft="20dp"
        app: exampleColor="#33b5e5"
        app: exampleDimension="24sp"
        app: exampleDrawable="@android: drawable/ic_menu_add"
        app: exampleString="Hello, MyView" />

</FrameLayout>
```

2.11 创建 App Widget

App Widget是应用程序的窗口小部件，可以被嵌入其他应用程序中（如桌面）并接收周期性的更新。

创建Widget的一般步骤：

01 在 res/layout 目录下创建一个 Widget 布局文件。

02 创建一个类继承 AppWidgetProvider。

03 在 res/xml 目录下创建一个 XML 文件，用来定义 Widget 的特性。

04 在 AndroidManifest.xml 中声明 Widget。

使用Android Studio的模板功能可以自动完成上面这些步骤。

在菜单栏中选择File→New→Widget→ App Widget，弹出【New Android Component】界面，如图 2-81 所示。

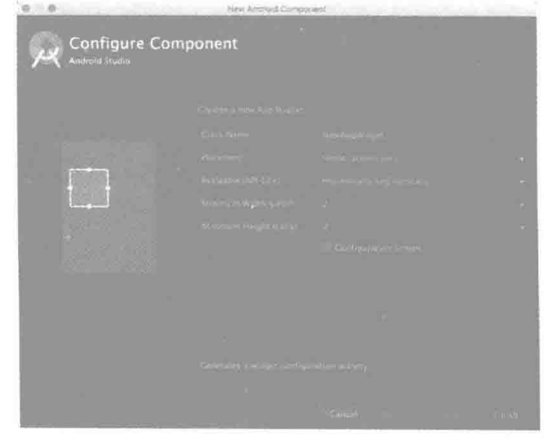


图 2-81

- Class Name: 类名，继承 AppWidgetProvider。
- Placement: Widget 放在哪儿。
  - ➢ Home-screen and Keyguard: 在主屏幕和锁键上。
  - ➢ Home-screen only: 仅在主屏幕上。
  - ➢ Keyboard only（API 17+）: 仅在锁键上（只支持Android 4.2 及以上版本）。
- Resizable（API 12+ ）: Widget 是否可调整大小，只支持 Android 3.1 及以上版本。
  - ➢ Horizontally and vertically: 水平和垂直显示时可调整。
  - ➢ Only horizontally: 仅水平时可调整。
  - ➢ Only vertically: 仅垂直时可调整。
  - ➢ Not resizable: 不可调整。
- Minimum Width: 最小宽度，参照左边预览窗口的单元格。
- Minimum Height: 最小高度，参照左边预览窗口的单元格。
- Configuration Screen: 勾选后会生成 widgets 配置 Activity。

使用默认配置，单击【Finish】按钮后创建成功。

2.12 创建可编译的资源文件

2.12.1 可编译的资源文件

Android项目中可编译的资源文件存放在res目录下，在R.java中会自动生成这些资源文件的ID，可以通过R.XXX.ID来访问。资源文件常用的默认目录如图 2-82 所示。

图 2-82

常用的默认目录和对应资源类型如表 2-3 所示。

表 2-3

默认目录	资源类型	说明
anim	存放定义了补间动画或逐帧动画的 XML 文件	补间动画是只定义起始、结束帧，中间的动作由系统进行计算获取动作过程
animator	存放定义了属性动画的 XML 文件	属性动画就是定义起始结束、动作、重复时间等参数齐全的动画
transition	存放定义了场景过渡动画的 XML 文件	
color	存放定义了颜色状态列表资源的 XML 文件	
layout	存放定义了用户界面布局的 XML 文件	LinearLayout、TableLayout、FrameLayout、RelativeLayout、GridLayout
menu	存放定义了应用程序菜单资源的 XML 文件	该文件下的 XML 文件可以应用于选项菜单、子菜单、上下文菜单等
values	存放定义了多种类型资源的 XML 文件，这些资源的类型可以是字符串、数据、颜色、尺寸、样式等	arrays.xml 存放数组资源，colors.xml 存放颜色资源，dimens.xml 存放尺寸值资源，strings.xml 存放字符串资源，styles.xml 存放样式资源，这些 XML 文件资源的根元素是 resource
xml	存放任意的 XML 文件	在运行时可以通过调用 Resources.getXML（）读取

图像文件目录和对应资源类型如表 2-4 所示。

表 2-4

默认目录	资源类型	说明
drawable	存放了能转换为绘制资源的位图文件或者定义了绘制资源的 XML 文件	后缀为.png、.9.png、.jpg、.gif 的图像文件
mipmap-mdpi	mdpi：medium dpi，中密度，适用于屏幕密度为 160 的手机设备	mipmap 用来存放启动图标
mipmap-hdpi	hdpi：high dpi，高密度，适用于屏幕密度为 240 的手机设备	
mipmap-xhdpi	extra high dpi：超高密度，适用于屏幕密度为 320 的手机设备	
mipmap-xxhdpi	extra extra high dpi：超超高密度，适用于屏幕密度为 480 的手机设备	
mipmap-xxxhdpi	extra extra extra high dpi：超超超高密度，适用于屏幕密度为 640 的手机设备	

扩展阅读

对于分辨率繁多的Android设备，为了方便原生应用的界面适配，Google按照dpi大小将设备分成 5 种模式（hdpi、mdpi、xhdpi、xxhdpi、xxxhdpi）。

dpi（dots per inch，每英寸点数）表示屏幕像素密度，密度越高，显示画面就越清晰。

以 1 倍的mdpi作为基准，所有的Android屏幕都可以找到自己的位置，并赋予相应的倍率：

```
mdpi [1 倍] hdpi [1.5 倍] xhdpi [2 倍] xxhdpi [3 倍] xxxhdpi [4 倍]
```

我们还可以在创建虚拟机的界面查看密度、分辨率和尺寸的对应关系，如图 2-83 所示。

图 2-83

2.12.2 创建可编译的资源文件

将光标放在不同的文件夹上，新建列表中显示的选项是不同的。

1. 光标放到module上

在菜单栏中选择File→右击module→New→Android resource file，弹出【New Resource File】对话框，如图 2-84 所示。

图 2-84

- File name：新建的资源文件名。
- Resource type：资源类型，下拉列表中会列出前面讲过的默认的资源文件类型，如图 2-85 所示。

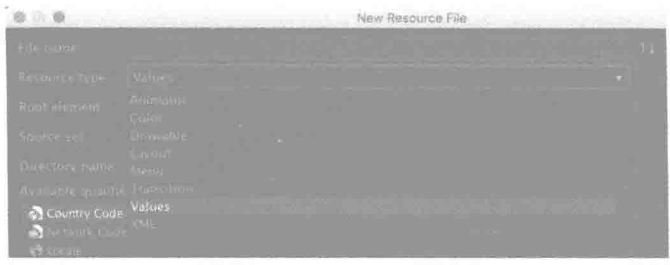


图 2-85

不同的资源类型会有不同的根元素和目录名，根元素和目录名会根据所选择的文件类型而变化。

例 1：资源类型为Drawable，根元素为selector，目录名为drawable，如图 2-86 所示。

图 2-86

例 2：资源类型为Layout，根元素为LinearLayout，目录名为layout，如图 2-87 所示。

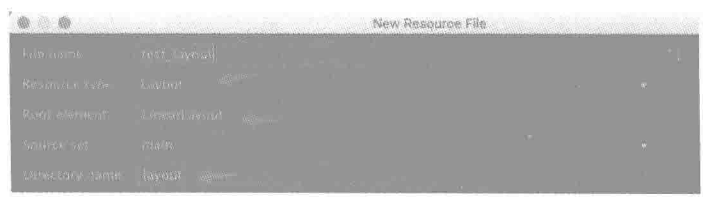


图 2-87

- Root element：XML 文件的根元素，不同的资源类型有不同的根元素，在新建文件以后会自动生成。

例 1：资源类型为Values的文件，根元素为resources。

```
<?xml version="1.0" encoding="utf-8"?>
<resources></resources>
```

例 2：资源类型为Transition的文件，根元素为transitionManager。

```
<?xml version="1.0" encoding="utf-8"?>
<transitionManager xmlns:
 android="http://schemas.android.com/apk/res/android">
</transitionManager>
```

- Source set：编译时引用的资源文件的来源设置，默认的是 main、debug、release，如图 2-88 所示。

资源文件会放在不同的文件夹中，在打不同的包时会引用不同的资源，如图 2-89 所示。

- Directory name：资源文件的目录名与资源类型一一对应，不可更改，新建的资源文件会自动放到这个目录下面，如果目录不存在会自动创建。
- Available qualifiers：可用的资源限定符。

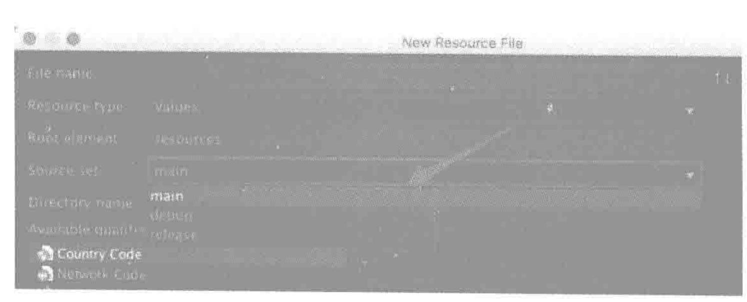


图 2-88

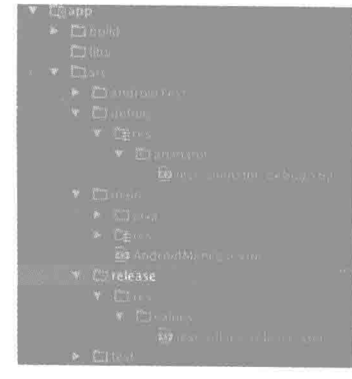


图 2-89

2. 光标放到资源文件夹上

光标放到资源文件夹上会显示对应的资源文件新建选项，如图 2-90 所示。单击后也会提示新建对应的资源文件，如图 2-91 所示，这样目标更明确。

图 2-90

图 2-91

3. 新建资源文件目录

新建资源文件目录（见图 2-92）只是新建目录，没有新建文件的地方。在配置项里，目录名可以自定义，如图 2-93 所示。

图 2-92

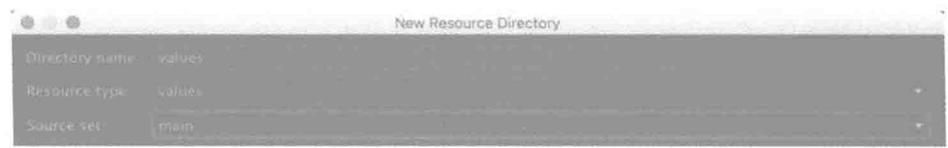


图 2-93

2.12.3 资源限定符

在新建资源文件时可以选择资源限定符。

在Available qualifiers列表中列出了适配时常用的一些限定符，如图 2-94 所示。

选择限定符，再进行相应的配置，就可以在res目录下生成相应的资源文件夹。

- Country Code：移动设备国家代码，如图 2-95 所示。唯一识别移动用户所属的国家，共 3 位，中国为 460。

图 2-94

图 2-95

- Network Code：移动设备网络代码，与 Country Code 相结合，用来表示唯一一个移动设备的网络运营商。
- Locale：国际化（多语言）。如果用户改变了系统中的语言设置，那么在应用程序的运行期间也能够改变为对应的语言，如图 2-96 所示。

图 2-96

- Layout Direction：布局方向，如图 2-97 所示。
  - ➢ LTR：从左到右。
  - ➢ RTL：从右到左。

图 2-97

- Smallest Screen Width：最小屏幕宽度。这个值是布局支持的最小宽度，而不管屏幕的当前方向。当应用程序提供了多个带有不同值的最小宽度限定符资源目录时，系统会使用最接近（不超出）设备最小宽度的那个资源。
- Screen Width：最小的可用屏幕宽度，如图 2-98 所示。当方向在横向和纵向之间改变时，这个配置值会跟当前的实际宽度相匹配。

图 2-98

- Screen Height：最小的可用屏幕高度。当方向在横向和纵向之间改变时，这个配置值会跟当前的实际高度相匹配。
- Size：屏幕尺寸，如图 2-99 所示。

图 2-99

- ➢ Small：小屏幕尺寸，类似低分辨率的QVGA屏幕，大约是 320 × 426dp。
- ➢ Normal：中等屏幕尺寸，类似中等分辨率的HVGA屏幕，大约是 320 × 470dp。
- ➢ Large：大屏幕尺寸，类似中等分辨率的VGA屏幕，大约是 480 × 640dp。
- ➢ X-large：超大屏幕尺寸，比HVGA屏幕大，大约是 720 × 960dp。

- Ratio：宽高比率，如图 2-100 所示。指的是实际的物理尺寸宽高比率，分为 Long（长屏幕）和 NotLong（非长屏幕）。
- Orientation：限制横竖屏切换，如图 2-101 所示。

图 2-100

Chosen qualifiers
Portrait
Screen orientation:
Portrait
Portrait
Landscape
Square

图 2-101

- ➢ Portrait：竖屏。
- ➢ Landscape：横屏。
- ➢ Square：正方形。

- UI Mode: UI 模式，如图 2-102 所示。

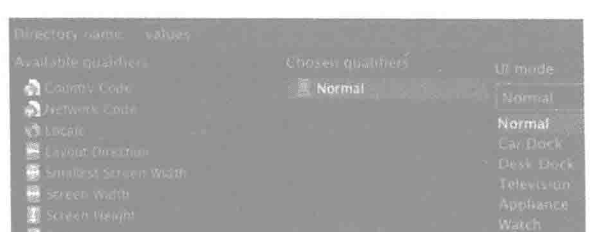


图 2-102

- ➢ Car Dock：车座。
- ➢ Desk Dock：桌座。
- ➢ Television：电视上。
- ➢ Appliance：装置。
- ➢ Watch：手表。

当用户将手机放在不同的Dock上或插入不同的地方，应用程序在运行期间就能够改变这个限定。

- Night Mode：夜间模式，如图 2-103 所示。
  - ➢ Not Night：白天。
  - ➢ Night：夜间。

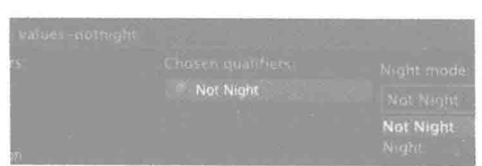


图 2-103

- Density：密度，如图 2-104 所示。
  指定不同密度的手机使用不同的资源，如果没有提供与当前设备配置匹配的可选资源，那么系统会使用最接近的资源。
- Touch Screen：触屏类型，如图 2-105 所示。
  - ➢ No Touch：非触屏。
  - ➢ Stylus：触控笔。
  - ➢ Finger：手指（触屏设备）。

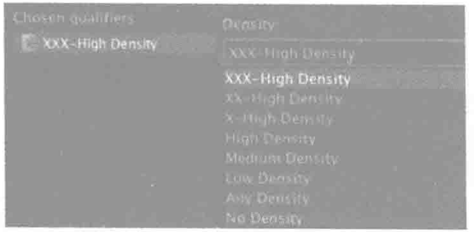


图 2-104

- Keyboard：键盘类型，如图 2-106 所示。
  - ➢ Exposed：设备有可用的键盘（硬或软）。
  - ➢ Hidden：设备有可用的硬键盘（被隐藏，且无可用的软键盘）。
  - ➢ Soft：设备有可用的软键盘（不管是否可见）。
- Text Input：文本输入法，如图 2-107 所示。

图 2-105

图 2-106

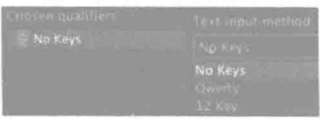


图 2-107

  - ➢ No Keys：设备没有用于文本输入的硬键盘。
  - ➢ Qwerty：设备有标准的硬键盘，不管用户是否可见。
  - ➢ 12 Key：设备有 12 个键的硬键盘，不管用户是否可见。
- Navigation State：导航键的状态，如图 2-108 所示。
  - ➢ Exposed：导航键可用。
  - ➢ Hidden：导航键不可用。

  如果用户能够看到导航键，那么在应用程序运行时就能够改变这个限定。
- Navigation Method：导航方法，如图 2-109 所示。

图 2-108

图 2-109

  - ➢ None：除了使用触屏以外，设备没有其他导航设施。
  - ➢ D-pad：设备有用于导航的定向板。
  - ➢ Trackball：设备有用于导航的轨迹球。
  - ➢ Wheel：设备有用于导航的定向滚轮。
- Dimension：自定义屏幕尺寸，如图 2-110 所示。
- Version：版本，设备支持的 API 级别，如图 2-111 所示。

图 2-110

图 2-111

2.13 创建不同分辨率的图标

我们可以快速方便地把现有的图片、剪贴画或文本生成不同分辨率的应用图标。

在菜单栏中选择File→右击Module→New→Image Asset→弹出Asset Studio对话框，如图2-112 所示。可以通过Icon Type切换不同的图标类型。

图 2-112

2.13.1 启动图标

在Icon Type中选择【Launcher Icons】，可以创建启动图标。

1. 使用本地图片

设置Assert Type为Image，然后选择一个本地的图片，自动生成不同分辨率的图标，如图2-113 所示。

设置图标的效果如图 2-114 所示。

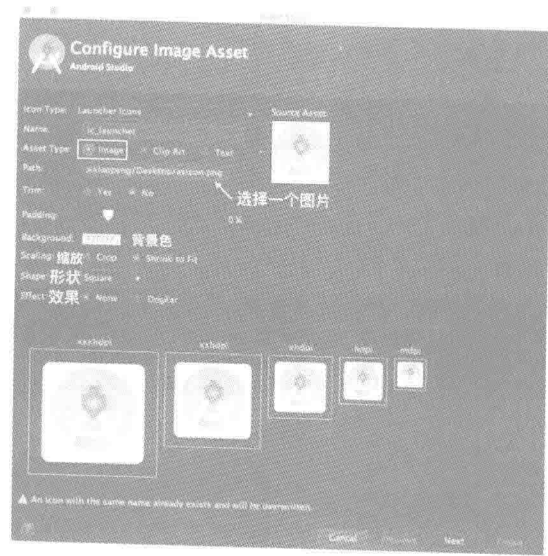


图 2-113

图 2-114

导出目录，如图 2-115 所示。

2. 使用剪贴画

设置Assert Type为Clip Art，可以选择一些剪贴画，如图 2-116 所示。

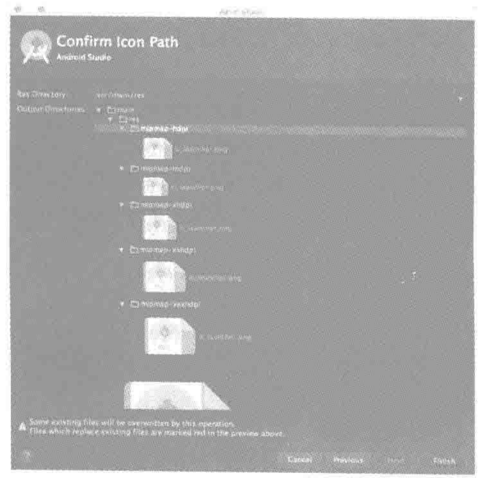


图 2-115

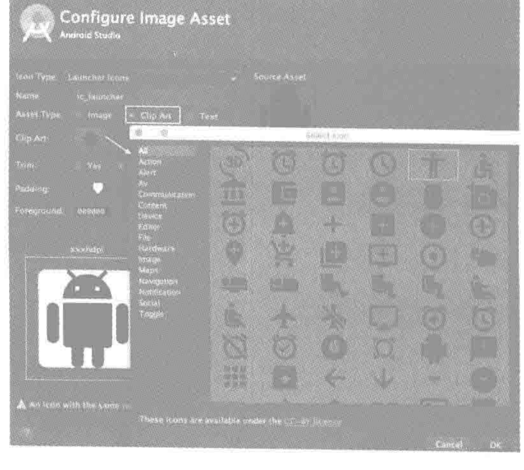


图 2-116

设置图标，如图 2-117 所示。

3. 使用文本

设置Assert Type为Text，如图 2-118 所示。

图 2-117

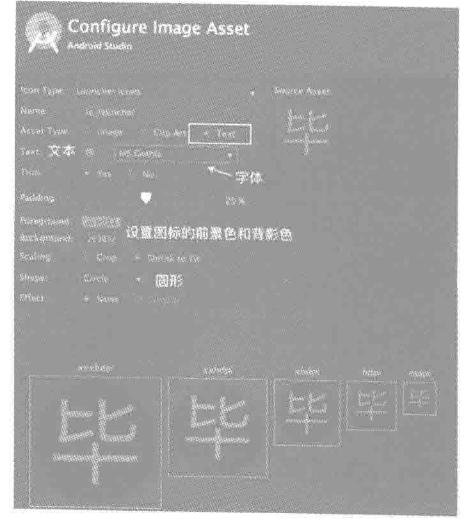


图 2-118

2.13.2　活动栏和选项卡图标

在Icon Type中选择【Action Bar and Tab Icons】，可以创建活动栏和选项卡图标。使用本地图片、剪贴画和文本的设置界面分别如图 2-199、图 2-120 和图 2-121 所示。

图 2-119

图 2-120

图 2-121

2.13.3 通知图标

在Icon Type中选择【Notification Icons】，可以创建通知图标。使用本地图片、剪贴画和文本的设置界面分别如图 2-122~图 2-124 所示。

图 2-122

图 2-123

图 2-124

2.14 创建矢量图

在Android 5.0（API Level 21）中，我们可以使用矢量图。矢量图的特点是不会因为图像的缩放而失真。也就意味着在Android开发中不需要为不同分辨率的设备定义不同大小的图片资源，只需一个vector drawable就够了。Android Studio从 1.4 版本开始支持vector drawable。

在Android Studio中，既可以选择定义好的Material Icon，也可以选择一个本地的SVG（可缩放矢量图形）文件。改变图片的尺寸和透明度后，Android Studio会生成一个 XML图像文件。

操作步骤：右击module→New→Vector Asset，打开Configure Vector Asset界面，如图 2-125 所示。

2.14.1 使用定义好的素材图标

选中【Material Icon】→单击Icon→选择定义好的素材图标→设置大小和透明度→查看预览效果，如图 2-126 所示。

图 2-125

图 2-126

单击【Next】→默认输出目录为res/drawable，如图 2-127 所示。

查看内容，如图 2-128 所示。

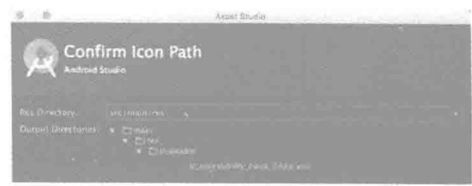


图 2-127

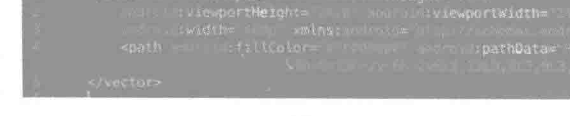


图 2-128

2.14.2 使用本地的SVG文件

选中【Local SVG file】→在Path中选择本地的svg文件，如图 2-129 所示。

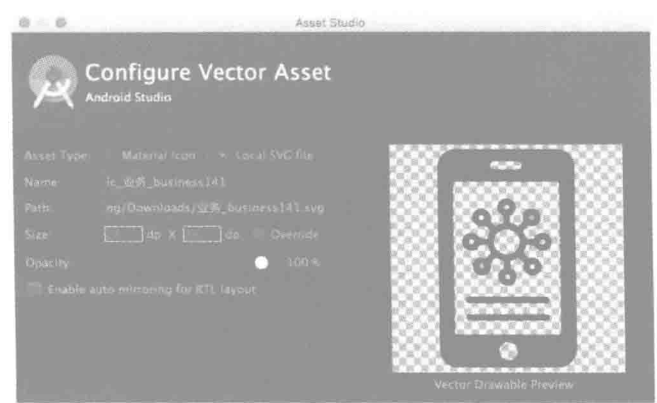


图 2-129

其他操作同样使用定义好的图标。

2.15 创建 AIDL 文件

Android接口定义语言（Android Interface Definition Language，AIDL）是一种接口描述语言。

AIDL的IPC机制是基于接口的，该接口描述了与一个远程对象进行通信的抽象协议。

编译器可以通过AIDL文件生成一段代码，通过预先定义的接口达到两个进程内部通信和跨进程对象访问的目的。

实现跨进程通信的一般步骤：

01 定义 AIDL 接口，包括创建 AIDL 文件、实现接口和向客户端公开接口。

02 通过 IPC 传递对象。

03 调用 IPC 方法。

本文只介绍在Android Studio中如何创建AIDL文件。前提条件是光标定位在模块上。

01 打开创建 AIDL 文件对话框。在菜单栏中选择 File→右击 Module→New→AIDL→AIDL File，弹出新建 AIDL 对话框，如图 2-130 所示。

图 2-130

02 配置并创建 AIDL 文件。输入接口名→选择目标来源（main/debug/release），默认是 main→Finish，结果如图 2-131 所示。

图 2-131

我们需要在这个接口里面定义对外提供的服务。

提 示

如果设置 Targe source set 为 debug，那么这个 AIDL 文件将会放在 debug 目录下。

03 定义对外提供的服务。在 AIDL 文件中定义需要对外提供的服务：

```
interface IMyTestAIDL {
   String getName ();
   int getAge (int name);
}
```

04 生成接口文件。在菜单栏中选择 Build→Rebuild Project，然后生成跟 AIDL 文件同名的接口文件，如图 2-132 所示。

这个接口文件里包含了一个名为Stub的静态抽象类以及在AIDL文件中定义的两个方法，如图 2-133 所示。然后就可以去实现这个接口了。

图 2-132

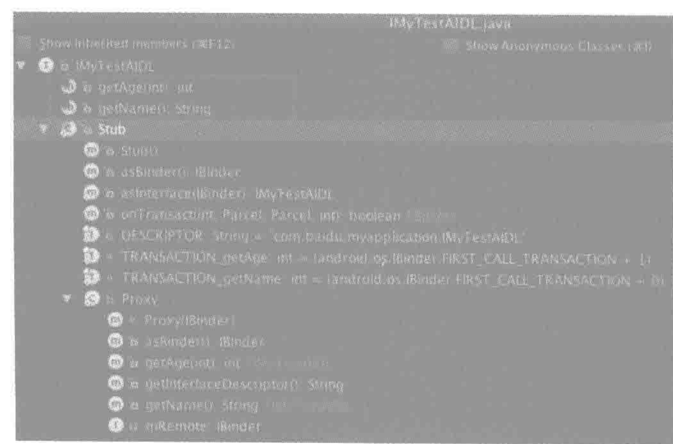


图 2-133

2.16 创建 Android 文件夹

在Android Studio中提供了创建文件夹的功能，在这里可以创建一些默认的文件夹，如图 2-134 所示。

图 2-134

如图 2-134 所示，这些文件夹都有对应的文件。新建某一类型的文件时，对应文件夹会自动被创建。例如，创建AIDL文件以后，AIDL文件夹就会自动被创建。这些文件夹默认的位置如表 2-5 所示。

表 2-5

文件夹	作用	默认位置
AIDL Folder	用于存放AIDL文件	src/main/aidl/
Assets Folder	用于存放原生资源文件	src/main/assets/
JNI Folder	用于存放JNI文件	src/main/jni/
Java Folder	用于存放Java文件	src/main/java/
Java Resources Folder	用于存放Java资源文件	src/main/resources/
RenerScript Folder	用于存放RenderScript文件	src/main/rs/
Res Folder	用于存放Android资源文件	src/main/res/

2.17 创建 Resource Bundle 文件

ResourceBundle（资源绑定）最大的好处是可以使程序国际化，在bundle文件名后加上国家、语言关键字，系统就会自动载入相应的bundle文件。

properties文件一般的命名规范是：自定义名_语言代码_国别代码.properties。例如，success.properties（默认）、success_zh.properties（中文）、success_ja.properties（日文）。

如果系统是日文，就会自动加载success_ja.properties。如果系统是中文，success.properties和success_zh.properties都存在，就会优先使用success_zh.properties文件。当success_zh.properties文件不存在时再使用success.properties。

新建Resource Bundle文件

01 右击 Module→New→Resource Bundle→输入文件名“success”→添加语言代码，如图 2-135 所示。

02 单击【OK】按钮，如图 2-136 所示。

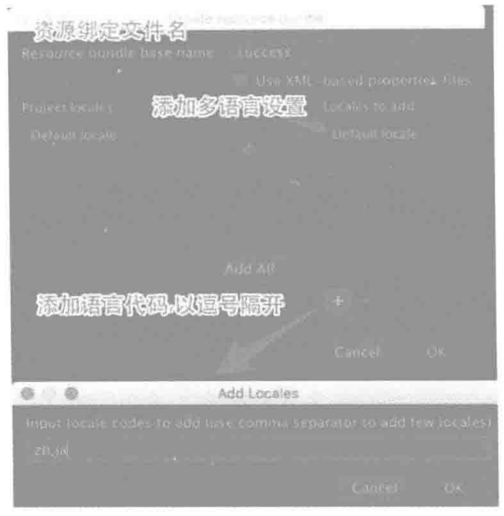


图 2-135

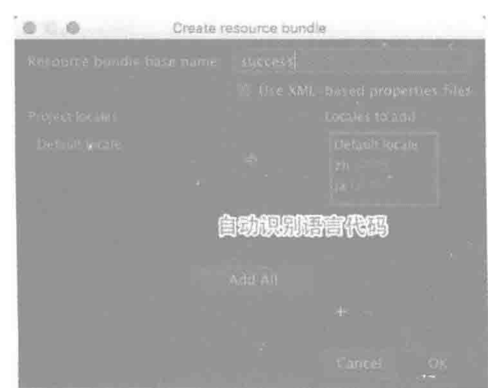


图 2-136

03 单击【OK】按钮，效果如图 2-137 所示。

04 双击 properties 文件→切换到 Resource Bundle→单击＋按钮→输入主键，如图 2-138 所示。

图 2-137

图 2-138

05 双击主键名称→编辑不同语言的值，如图 2-139 所示。

图 2-139

properties文件全部被编辑了，是不是很方便、高效呢？

第3章 布 局

Android Studio中创建布局文件有两种方式：在XML文本编辑器中直接编写代码和使用可视化布局编辑器通过拖曳控件自动生成。

可视化布局编辑器可以通过拖曳控件生成布局，可以实时预览布局的效果，可以在不安装APP的情况下随时查看布局在不同分辨率、不同版本、不同品牌手机上的显示效果，节省大量调试界面的时间。

本章主要向大家介绍Android Studio中可视化布局编辑器的使用。

本章重要知识点 >>>>>>>>>>

- 如何快速创建布局和组件；
- 如何设置布局和组件的属性；
- 如何预览布局以及预览工具的使用。

3.1 认识布局

布局（Layout）是一个Activity中视图界面的架构，定义了一个视图的结构。结构中包含的一个个组件构建出了漂亮的视图界面。

3.1.1 Android中定义布局的方法

Android中提供了两种方法来定义布局。

1. 在XML格式的布局文件中定义

XML格式的布局文件一般放在项目的res/layout目录下，通常以对应的Activity名称来命名。例如，MainActivity.java对应的布局文件名为activity_main.xml。

2. 在代码中使用相应的方法来实例化布局元素

可以在代码中创建View和ViewGroup对象并操作它们的属性。最常用的方法是在XML布局文件中定义布局层次结构和使用到的组件，在代码中控制和动态修改组件的状态。 这样做的好处是将应用程序的视图界面和操作行为的代码分离，可以更好地遵循MVC设计模式。

提 示

MVC 设计模式=Model + View + Controller，即业务逻辑、视图显示、数据处理分开。

3.1.2 快速开始

下面通过一个简单的布局实例来介绍如何新建、设计、运行一个布局文件。

01 新建布局文件。右击 res 目录下的 layout 目录→New→XML→Layout XML File，设置布局文件名和 XML 文件的根标签，如图 3-1 所示。文件名通常以对应的 Activity 名称来命名，根标签默认是线性布局，可以根据需求更改。

02 设计布局。布局文件创建后，就需要根据需求来设计布局了。这里设计一个最简单的布局文件：添加 TextView 和 Button 两个组件。既可以在 xml 文件中直接输入（见图 3-2），也可以使用可视化的布局编辑器直接拖曳生成（见图 3-3）。

图 3-1

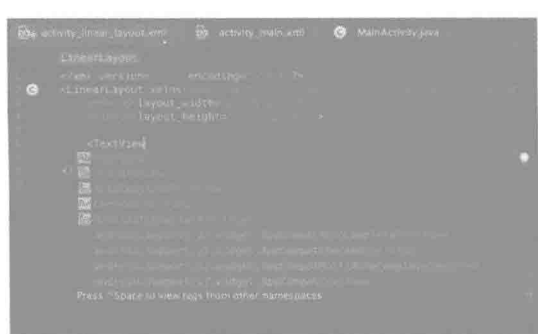


图 3-2

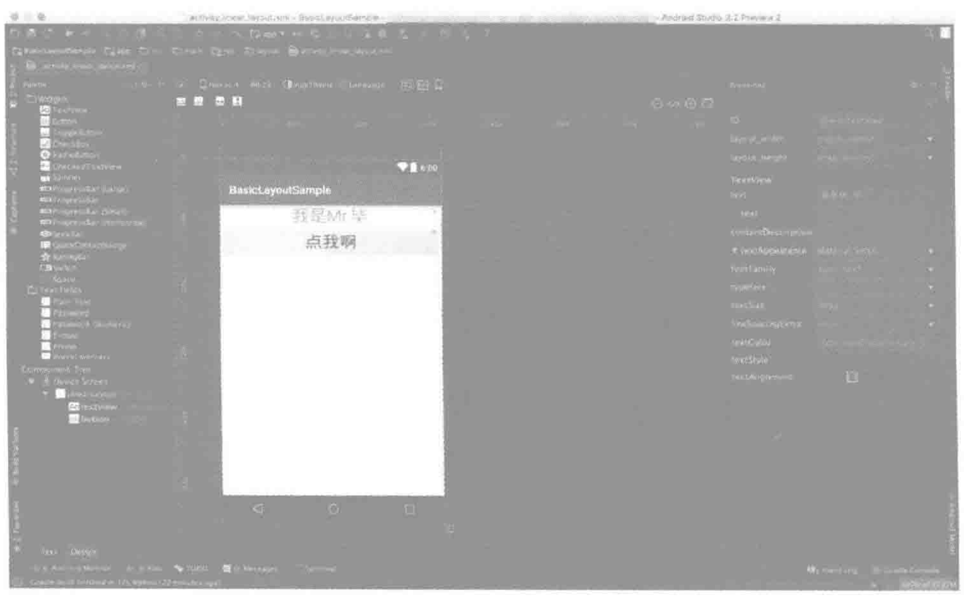


图 3-3

不会用布局编辑器也不要着急，接下来我们会详细介绍如何使用布局编辑器。

通过拖曳生成的布局文件源码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns: android="http://schemas.android.com/apk/res/android"
    android: layout_width="match_parent"
```

```
    android: layout_height="wrap_content"
    android: orientation="vertical">

    <TextView
        android: id="@+id/textView"
        android: textAlignment="center"
        android: textSize="30sp"
        android: layout_height="wrap_content"
        android: layout_width="match_parent"
        android: text="我是 Mr.毕" />

    <Button
        android: text="点我啊"
        android: layout_width="match_parent"
        android: layout_height="wrap_content"
        android: id="@+id/button"
        android: elevation="0dp"
        android: textSize="30sp" />
</LinearLayout>
```

03 关联 Activity。布局文件有了，接下来需要跟对应的 Activity 关联：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState) ;
        setContentView (R.layout.activity_linear_layout) ;
    }
}
```

运行程序，效果如图 3-4 所示。

图 3-4

3.2 设计布局

Android Studio中设计布局有两种方法：一种是在xml中使用文本编辑器进行设计，另一种是使用可视化的编辑器进行设计。

3.2.1 文本编辑器

在XML文本编辑器中设计布局时可以打开右边工具栏的Preview工具窗口实时预览效果，如图 3-5 所示。

布局效果预览窗口与可视化布局编辑器窗口中的效果预览是一样的，具体的使用方法后面

会详细介绍。文本和可视化布局编辑器可通过底部的Text、Design标签进行切换。

展开组件列表，效果如图 3-6 所示。即可以拖曳组件来设计布局，也能实时看到生成的代码，还可以通过代码对控件进行一些调整。

图 3-5

图 3-6

3.2.2 可视化布局编辑器

可视化布局编辑器由五大部分组成：组件列表、结构树、工具栏、预览和属性，如图 3-7 所示。

图 3-7

① 组件列表中提供了常用的UI组件，可以直接拖曳到预览窗口生成布局控件元素。

② 结构树显示了当前布局的层次结构，即可以管理控件，又可以调整控件的位置。

③ 工具栏提供了很多预览工具，可以查看布局在不同分辨率、不同API和不同主题上的显示效果。

❹ 预览窗口实时显示布局的效果，选中组件可以在属性中进行设置。
❺ 在属性面板中可以非常方便地设置选中控件的各项参数。

接下来我们主要介绍可视化布局编辑器的使用方法。

3.3 组件列表

可视化布局编辑器的组件列表中提供了非常丰富的UI组件，可以直接拖曳组件到预览窗口来生成布局文件。组件列表按照一定的规则分成了不同的类别，如表 3-1 所示。

表 3-1

序号	分类	说明
1	Widgets	最常用的组件
2	Text Fields	文本框组件
3	Layouts	布局组件
4	Containers	容器组件
5	Image & Media	图片和媒体组件
6	Date & Time	日期和时间组件
7	Transitions	转换组件
8	Advanced	高级组件
9	Custom - Google	定制的 Google 服务组件
10	Custom - Design	定制的 Material Design 组件
11	Custom - AppCompat	定制的 APP 兼容组件

定制的组件在使用时都会添加依赖的jar包，如图 3-8 所示。单击【OK】按钮会自动将依赖添加到app模块的build.gradle中。

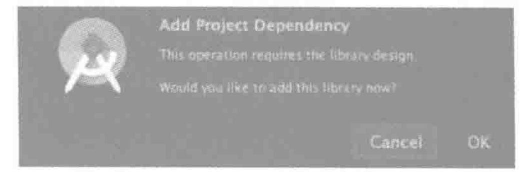


图 3-8

1. 组件列表显示模式

组件列表有两种显示模式（见图 3-9）：一是显示组件的图标和描述文本，二是显示组件的预览效果。两种模式的显示效果如图 3-10 所示。

图 3-9

Android Studio 2.2 以前，组件列表只显示组件的图标和描述文本。对于新手来说，经常要拖动到预览窗口查看效果后才能确定是否为自己想要的组件。现在好了，从 2.2 版本开始，组件列表可以直接切换到显示组件预览效果的模式，可以直接根据显示效果选择需要的组件，使用起来更加方便。

例如，Text Fields组件的预览效果如图 3-11 所示。

需要注意的是，所有定制组件必须添加对应的依赖以后才能显示预览效果。

图 3-10

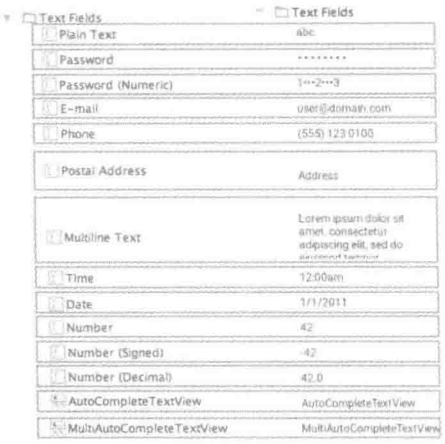


图 3-11

2. Widgets组件

Widgets中提供了一些最常用的组件，包括组件常用的样式。这些组件都是一些相对独立单一的功能，如表 3-2 所示。

表 3-2

组件	描述	组件	描述
TextView	文本	ProgressBar	进度条（标准）
Button	按钮	ProgressBar（Small）	进度条（小）
ToggleButton	切换按钮	ProgressBar（Horizontal）	进度条（水平）
CheckBox	复选框	SeekBar	拖动条
RadioButton	单选按钮	QuickContactBadge	快捷联系人标识
CheckedTextView	文本复选框	RatingBar	星级评分条
Spinner	下拉列表	Switch	开关
ProgressBar（Large）	进度条（大）	Space	空白

3. Text Fields组件

Text Fields中提供了常用的文本框输入组件（见表 3-3），系统会根据不同的输入类型弹出不同的键盘。例如，输入类型为Phone，就限定了输入框中只能输入数字，所以只会弹出数字键盘。

表 3-3

组件	描述	组件	描述
Plain Text	输入普通文本	Time	输入时间
Password	输入任意字符的密码	Date	输入日期
Password（Numeric）	输入数字密码	Number	输入数字
E-mail	输入邮箱地址	Number（Signed）	输入带正负号的数字

（续表）

组件	描述	组件	描述
Phone	输入电话	Number（Decimal）	输入带小数点的数字
Postal Address	输入邮政地址	AutoCompleteTextView	自动完成文本框（单选）
Multiline Text	输入多行文本	MultiAutoCompleteTextView	自动完成文本框（多选）

4. Layouts组件

Layouts中提供了布局相关的组件（见表 3-4）。

表 3-4

布局	描述	布局	描述
ConstraintLayout	约束布局	RelativeLayout	相对布局
GridLayout	网格布局	TableLayout	表格布局
FrameLayout	帧布局	TableRow	表格行（表格中的一行，与 TableLayout 搭配使用）
LinearLayout（Horizontal）	线性布局（水平方向）	<fragment>	布局文件中用 fragment 标签指定实例化的 Fragment 类
LinearLayout（Vertical）	线性布局（垂直方向）		

5. Containers组件

Containers中提供了容器相关的组件（见表 3-5）。

表 3-5

组件	描述	组件	描述
RadioGroup	单选框组	HorizontalScrollView	水平滚动视图
ListView	列表视图	TabHost	标签栏
GridView	网格视图	WebView	网页视图
ExpandableListView	可展开的列表视图	SearchView	搜索视图
ScrollView	滚动视图		

6. Image & Media组件

Image & Media中提供了图片和媒体相关的组件（见表 3-6）。

7. Date & Time组件

Date & Time中提供了日期和时间相关的组件（见表 3-7）。

表 3-6

组件	描述
ImageButton	图片按钮
ImageView	图片视图
VideoView	视频视图

表 3-7

组件	描述
TimePicker	时间选择器
DatePicker	日期选择器
CalendarView	日历视图

8. Transitions组件

Transitions中提供了转换相关的组件（见表3-8）。

9. Advanced组件

Advanced中提供了一些高级组件（见表3-9）。

表3-8

组件	描述
ImageSwitcher	图片切换器
AdapterViewFlipper	AdapterView翻转器
StackView	堆叠视图
TextSwitcher	文本切换器
ViewAnimator	视图动画
ViewFlipper	视图翻转器
ViewSwitcher	视图切换器

表3-9

组件	描述
<indude>	用于在一个布局文件中引用另一个布局
<requestFocus>	用于指定屏幕内的焦点 View
<view>	视图
ViewStub	占位视图
TextureView	结构视图
SurfaceView	表面视图
NumberPicker	数字选择器

10. Custom-Google组件

Custom-Google中提供了定制的Google服务组件（见表3-10）。

表3-10

组件	描述	扩展阅读
AdView	用于显示横幅广告的视图容器，会响应用户的触摸而展示小型 HTML5 广告	https://developers.google.com/mobile-ads-sdk/docs/admob/fundamentals?hl=zh-cn#header
MapFragment	地图容器，提供对 GoogleMap 对象的访问权	https://developers.google.com/maps/documentation/android-api/intro?hl=zh-cn
MapView	地图容器，通过 GoogleMap 对象公开核心地图功能	同上

11. Custom - Design

Custom - Design中提供了定制的Material Design组件（见表3-11）。

表3-11

组件	描述
CoordinatorLayout	协调布局，是增强型的 FrameLayout
AppBarLayout	应用程序栏布局，是垂直的 LinerLayout，支持滚动手势
NestedScrollView	嵌套 ScrollView，兼容新的控件
FloatingActionButton	浮动操作按钮
TextInputLayout	文本输入布局，主要用于包含 EditText，会默认生成一个浮动的 Label

更多扩展内容可参考https：//guides.codepath.com/android/Design-Support-Library#features。

12. Custom - AppCompat

Custom - AppCompat中提供了定制的App兼容组件（见表 3-12）。

表 3-12

组件	描述	组件	描述
CardView	卡片视图	RecyclerView	回收视图，用来取代 ListView
GridLayout	网格布局	Toolbar	导航栏，用来取代 Actionbar

3.4 预 览

预览窗口左边显示的是设计预览，右边显示蓝图模式。

- 设计预览用于实时显示布局设计的效果。
- 蓝图模式用于显示控件的间距和布局的结构，在蓝图模式可以非常直观地看到布局及其控件的结构和摆放细节。

3.4.1 设置控件属性

单击预览或蓝图中的控件，Component Tree（组件结构树）中的控件也被会选中，同时会显示选中控件的属性，如图 3-12 所示。

图 3-12

我们既可以在预览或蓝图界面通过拖动改变控件的属性（大小和位置），也可以在结构树中改变选中控件的位置，在属性面板中设置控件精确的属性。

3.4.2 警告和错误提示

Android Studio会使用Lint检测出布局文件中的一些问题，并在布局文件中进行提示。右上角会显示布局文件中检测出的问题数量，单击后显示详细信息，如图 3-13 所示。

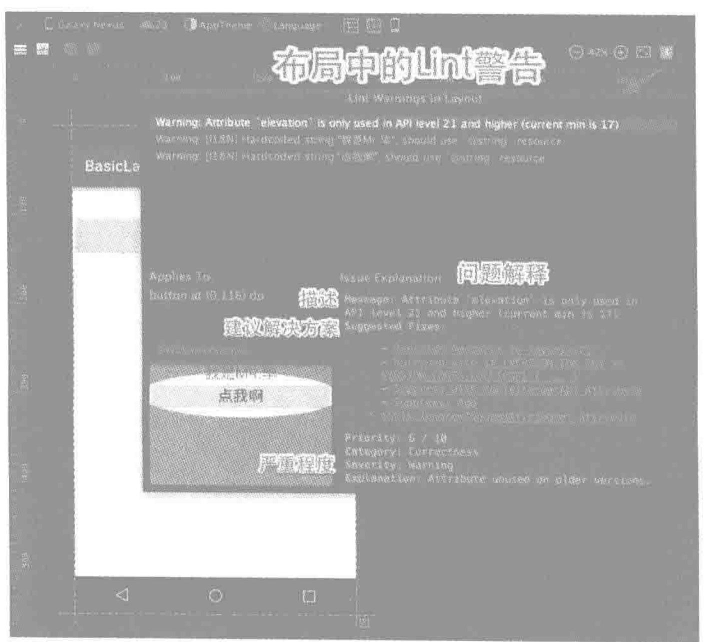


图 3-13

这就相当于执行了Lint检查一样，每一个问题都给出了描述信息以及解决方案，以便快速解决问题。

另外，在预览界面也会显示警告标识，将鼠标放在警告标识上，会显示问题描述，如图 3-14 所示。

如果布局在渲染时出现错误，就会提示错误原因及解决方案，如图 3-15 所示。

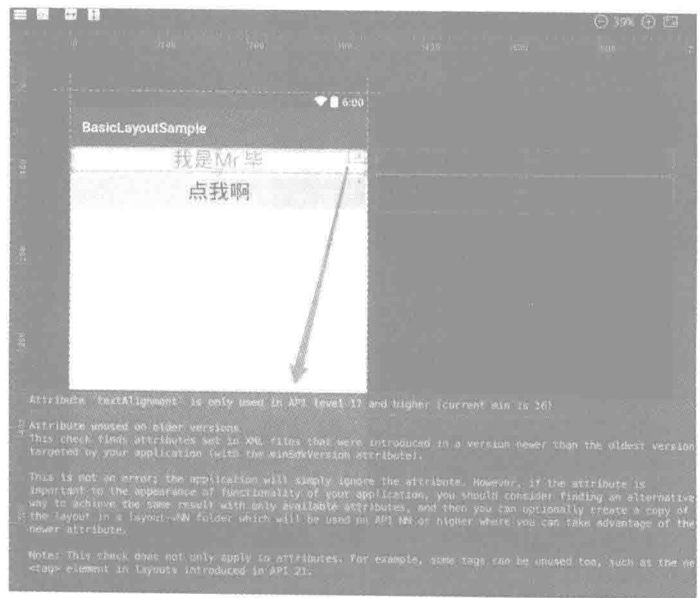


图 3-14

图 3-15

3.4.3 界面缩放

可以利用图 3-16 中的各按钮进行界面缩放。

图 3-16

3.4.4 控件操作

右击控件或布局会显示一些操作选项，如图 3-17 所示。这里的操作选项跟结构树中是一致的，可参看 3.5 节。

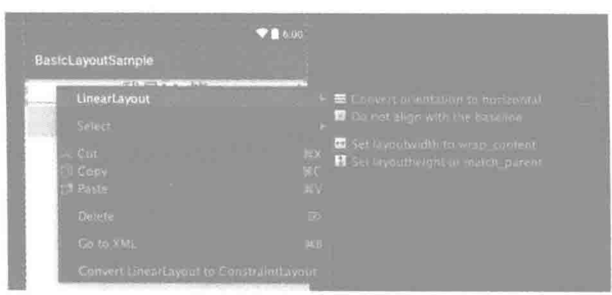


图 3-19

3.5 结构树

通过结构树可以清晰地查看组件的类型和布局的层次结构，还可以对组件进行复制、粘贴、剪切、删除等操作，如图 3-18 所示。

结构树和预览窗口中对控件的操作是一样的，一些基本的操作就不再讲了，主要介绍下面几个操作。

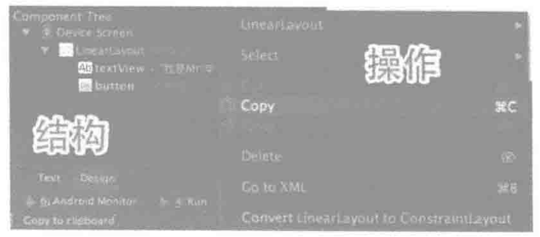


图 3-18

3.5.1 快速转换布局属性

操作步骤：右击控件→布局名→列出布局属性转换工具，如图 3-19 所示。从上到下依次为转换布局方向、不与基线对齐、设置 layout_width 为 wrap_content/match_parent 和设置 layout_height为wrap_content/match_parent，并与工具栏上的快捷工具一一对应。

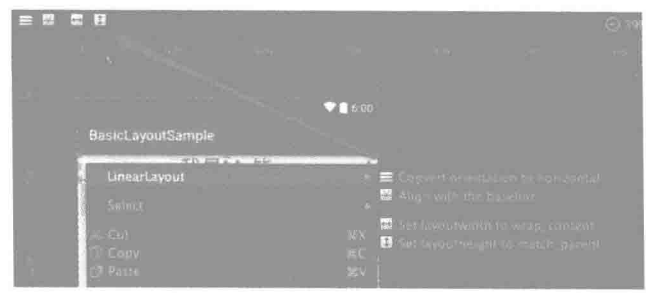
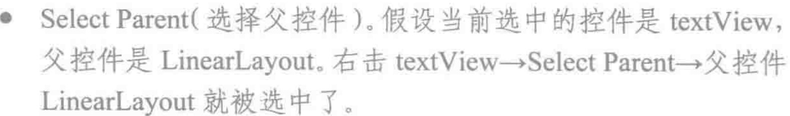


图 3-17

3.5.2 选择控件

操作步骤：右击控件→Select→列出选择控件的方法，如图 3-20 所示。

图 3-20



- Select Parent（选择父控件）。假设当前选中的控件是 textView，父控件是 LinearLayout。右击 textView→Select Parent→父控件 LinearLayout 就被选中了。
- Select Siblings（选择同级控件）。选择 Select Siblings，会把结构树中与此控件同级的控件全部选中。
- Select Same Type（选择相同类型的控件）。选择 Select Same Type，会把结构树中与此控件类型相同的控件全部选中。
- Select All（全选）。
- Deselect All（取消全选）。

3.5.3 跳到源码

当我们想查看某个控件的源码时，可以快速跳转到控件声明的位置。

方法一：右击控件→Go To Declaration。

方法二：快捷键Command + B（macOS）或Ctrl + B（Windows/Linux）。

注意：快捷键只有在预览窗口才能生效。

3.6 属 性

在预览窗口或结构树中选中一个控件时，相关的属性就会被显示出来，在属性面板中可以非常方便地设置选中控件的各项参数，如图 3-21 所示。

选中控件，默认只显示关键属性。

这个属性面板被重新设计过了，看起来都非常方便，可以达到快速编辑属性的目的。

（1）显示全部属性

如果想显示全部属性，可单击右上解的切换按钮，如图 3-22 所示。

图 3-21

图 3-22

在完整的属性界面中设置一个参数时会稍微麻烦一点。

（2）查看属性信息

如果不知道某个属性是什么意思，将光标放到属性上面，就会显示属性的介绍信息，如图 3-23 所示。

（3）搜索属性

属性面板中列出的属性非常多，要搜索某个属性时，可以将光标定位在属性面板上，直接输入要搜索的属性名（如theme），如图 3-24 所示。

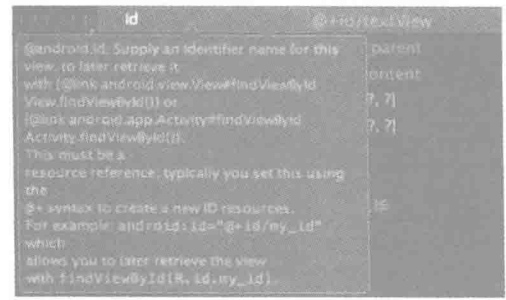


图 3-23

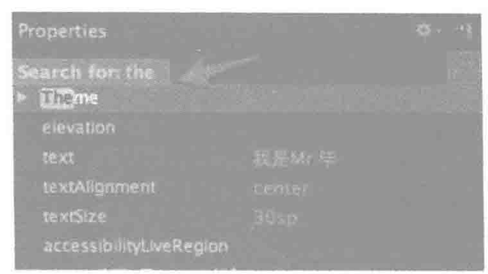


图 3-24

3.7 工具栏

工具栏提供了很多设置和预览工具，可以快速预览布局的效果，如图 3-25 所示。

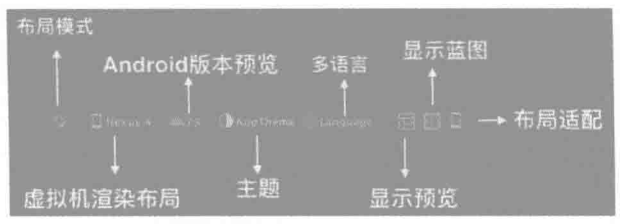


图 3-25

1. 布局模式

布局模式如图 3-26 所示。

图 3-26

这里主要是横竖屏模式切换、模拟UI模式和夜间模式。

- UI Mode：UI 模式，有 Normal（正常）、Car Dock（车座）、Desk Dock（桌座）、Television（电视上）、Appliance（装置）和 Watch（手表）6 种选择。当用户将手机放在不同的 Dock 上或插入不同的地方时，应用程序在运行期间就能够改变这个限定。
- Night Mode：夜间模式，包括 Not Night（白天）和 Night（夜间）两个选项。

2. 虚拟机渲染布局

使用虚拟机渲染布局的操作列表可分为 5 个部分，如图 3-27 所示。

❶ 已创建的虚拟机列表。

❷ 典型的手机列表。

❸ Android Wear列表。

❹ Android TV列表。

❺ 自定义尺寸（普通的手机和平板或添加自定义设备）。

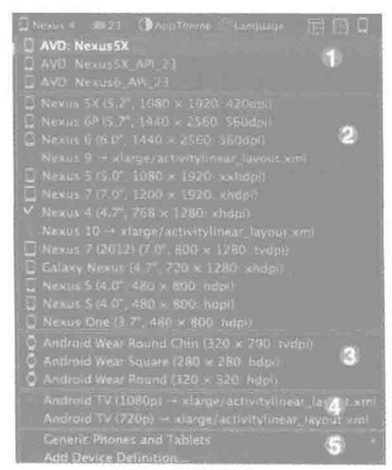


图 3-27

3. 预览Android版本

当我们需要预览当前布局在不同Android版本上的显示效果时，可以使用此功能，如图 3-28 所示。

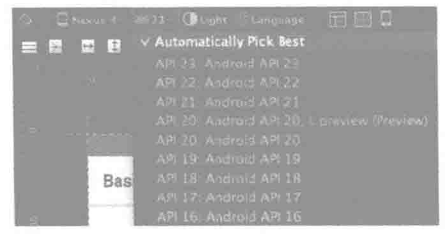


图 3-28

4. 主题预览

主题作用于某个Activity或整个应用程序，用于统一窗口的外观。当我们使用某个主题时，Activity或应用程序中相同的属性就会被替换为这个主题的样式。Android Studio可视化布局编辑器中提供了主题实时预览的功能。

在选择主题界面（见图 3-29），Android Studio将主题进行了归类（见表 3-13），可以很方便地找到想要的主题。

图 3-29

表 3-13

分类	说明
Recent	最近使用过的主题
Manifest Themes	AndroidManifest.xml 中配置的主题
Project Themes	项目中创建的主题
Material Dark	Material 暗色主题，只支持 Android 5.0 及以上版本
Material Light	Material 亮色主题，只支持 Android 5.0 及以上版本
Holo Dark	Holo 暗色主题，只支持 Android 4.0 及以上版本
Holo Light	Holo 亮色主题，只支持 Android 4.0 及以上版本
Device Default	设备上的默认主题
Dialogs	所有对话框主题
Light	所有亮色主题
All	所有可用主题

Holo Light主题与Material Light主题对比效果如图 3-30 所示。

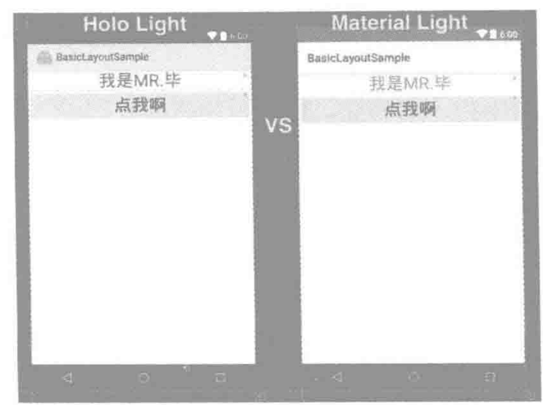


图 3-30

5. 预览多语言

APP在国际化时需要适配不同的语言。布局编辑器中提供了语言预览的功能，如图 3-31 所示。

单击【Edit Translations】，打开翻译编辑器，可以在这里添加多语言翻译，如图 3-32 所示。

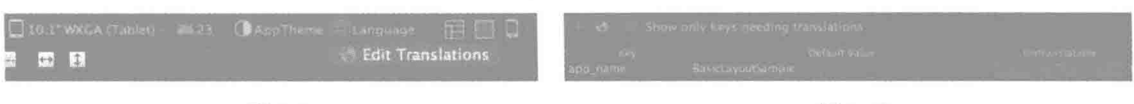


图 3-31　　　　图 3-32

- Default Value 为默认的语言，在这里默认语言是中文。
- Untranslatable 选项需要我们确认是否不可译，如果勾选，对应的语言就可以不翻译，否则不翻译就会报错。

【实例演示】为APP添加英语翻译 English（en），如图 3-33 所示。

图 3-33

res目录下会新建一个values-en/strings.xml文件：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Basic Layout Sample</string>
    <string name="name">I am Mr.B</string>
    <string name="click_me">CLICK ME</string>
</resources>
```

回到布局编辑器，查看效果，如图 3-34 所示。

6. 布局适配

这里主要列出了布局适配相关的功能，提供了创建横屏/竖屏和超大布局文件的功能，如图 3-35 所示。

图 3-34

图 3-35

（1）Create Landscape Variation（创建横屏布局文件）

当前的布局是竖屏的，单击【Create Landscape Variation】后，就创建了一个横屏的布局文件，如图 3-36 所示。

图 3-36

这个布局文件放在layout-land目录下。

创建横屏版本以后，这个工具列表就多了一个横竖屏切换的功能，如图 3-37 所示。这样就可以很方便地知道当前的布局有没有横屏文件，如果有就可以直接切换查看。

（2）Create layout-xlarge Variation（创建超大布局）

创建超大布局主要是针对平板之类的适配，布局文件会被放在layout-xlarge目录下。创建之后可切换的布局又多了一个，即Switch to layout-xlarge，如图 3-38 所示。

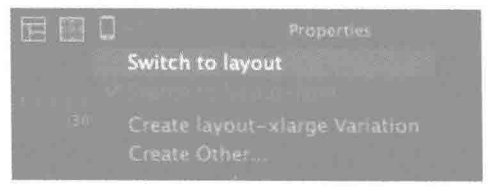


图 3-37

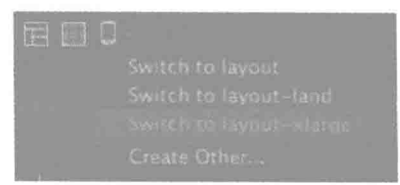


图 3-38

（3）Create Other（创建其他文件）

主要是创建一些适配的资源文件。

第4章　管　理

项目中的文件和代码管理是非常重要的，如果我们不了解Android Studio中的项目管理技巧，很多情况下对项目进行操作会不知所措，甚至会走很多弯路。学会了本章的内容，你就可以大胆地对项目和文件进行管理了。

本章向大家介绍Android Studio中项目、文件和代码的管理技巧。

本章重要知识点 >>>>>>>>>>

- 如何使用项目窗口的视图模式；
- 如何管理项目；
- 如何管理文件；
- 如何清理、重启 IDE；
- 如何使用收藏和 TODO。

4.1　项目窗口

4.1.1　视图模式

Android Studio中支持好多种视图模式，很多开发同学刚开始弄不清楚这些视图的区别，搞不清楚什么时候该用什么样的视图，本文会向大家介绍这些视图的特点以及作用。

Android Studio中提供的视图模式如图 4-1 所示。每一种视图模式的显示方式和特性都是不同的，常用的视图模式有以下几种。

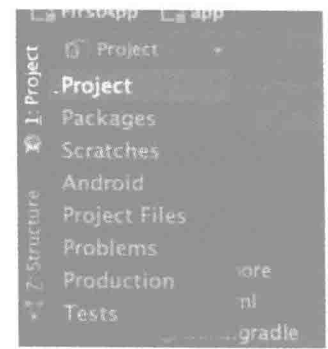


图 4-1

1. Project视图模式

如图 4-2 所示，Project视图模式展示全部文件信息，文件的位置是真实的物理结构，因此在查看文件的时候建议切换到Project模式。

2. Packages视图模式

如图 4-3 所示，Packages视图模式仅显示项目本身的代码和资源，其他的信息都被隐藏了，代码和资源都以层级文件的形式显示。

3. Scratches视图模式

如图 4-4 所示，Scratches视图模式只显示草稿文件。

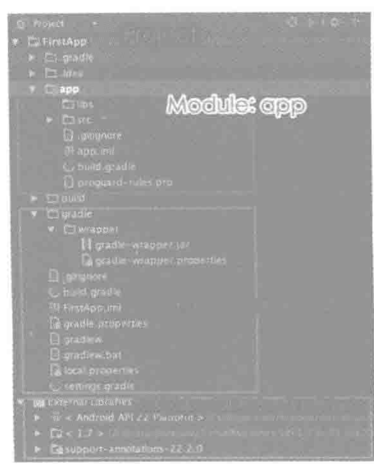


图 4-2

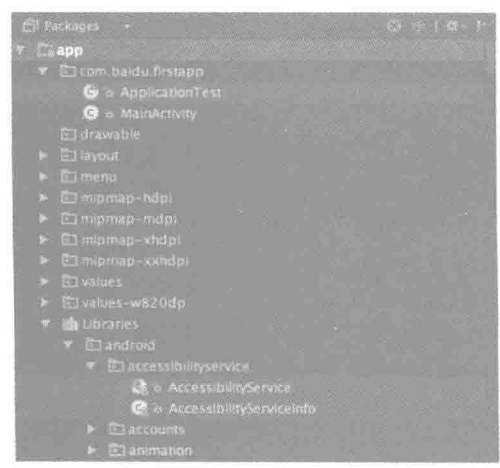


图 4-3

Scratch Files（草稿文件）是一个非常方便的功能，可以帮助我们快速实验和建立原型。使用Scratch Files可以在不修改项目和创建任何文件的情况下快速画出草图，Android Studio（Intellij idea）为Scratch Files提供了辅助编码功能。

图 4-4

想了解更多，请参考http://blog.jetbrains.com/idea/2014/09/intellij-idea-14-eap-138-2210-brings-scratch-files-and-better-mercurial-integration/。

4. Android视图模式

如图 4-5 所示，Android视图模式会把一些我们不关心的文件和目录隐藏起来，以一种扁平化的方式显示项目结构，文件和目录通过类型进行归类，可以非常方便地查看文件。

5. Project Files视图模式

如图 4-6 所示，Project Files视图模式是类似Eclipse的项目结构形式。

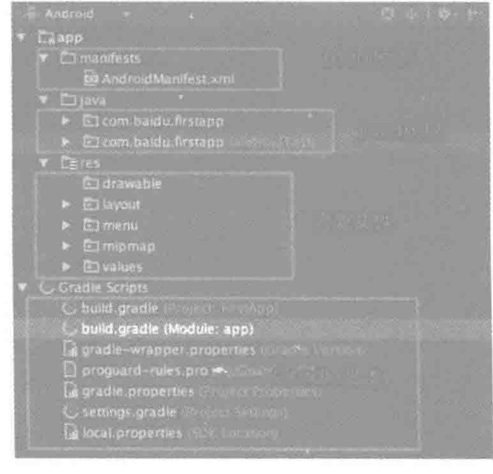


图 4-5

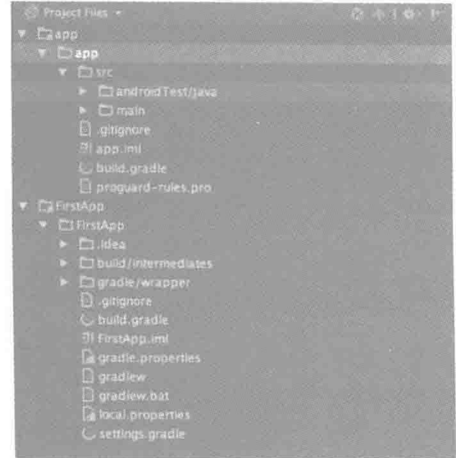


图 4-6

6. Problems视图模式

如图 4-7 所示，Problems模式仅显示报错的文件结构。

图 4-7

7. Production视图模式

如图 4-8 所示，Production模式仅显示生产文件结构。

8. Tests视图模式

如图 4-9 所示，Tests模式仅显示测试文件结构。

图 4-8

图 4-9

4.1.2 常用设置和操作

1. 展开和收缩项目窗口

（1）展开项目窗口

快捷键：shift +command + 向右箭头（macOS）或者 Shift + Ctrl + 向右箭头（Windows/Linux）

每操作一次窗口向右展开一下，最终项目窗口会被最大化，编辑窗口被隐藏，如图 4-10 所示。

图 4-10

（2）收缩项目窗口

图 4-11

快捷键：shift+command+向左箭头（macOS）或者Shift+Ctrl+向左箭头（Windows/Linux）

每操作一次窗口向左收缩一下，最终整个项目窗口被隐藏，如图 4-11 所示。

（3）最大化项目窗口

快捷键：shift +command + '（macOS）或者Shift + Ctrl + '（Windows/Linux）

2. 按顺序切换视图模式

在项目窗口我们既可以通过展开视图列表来切换视图模式，也可以使用快捷键按顺序切换视图模式。

图 4-12

右击项目窗口的工具栏，在Mac中，选择Select Next Tab 或 Select Previous Tab（见图 4-12）；在Windows/ Linux中，Select Next View或Select Previous View。

快捷键：shift +command +]或Shift +Command + [（macOS），Alt + 向右箭头或Alt + 向左箭头（Windows/Linux）

3. 定位文件在项目中的位置

我们想查看当前打开的文件在项目中的位置时，应该如何操作呢？

假设当前打开的文件是AndroidManifest.xml，如图 4-13 所示，此时我们想知道它在项目中的位置。

操作步骤：单击项目窗口工具栏上的Scroll from Source按钮（见图 4-14）就定位到了文件所在位置。

图 4-13

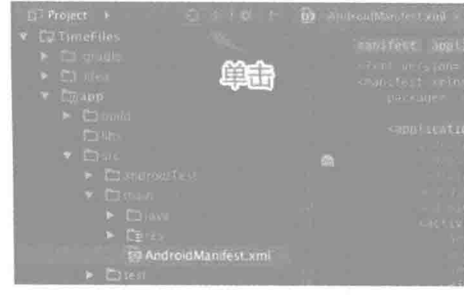


图 4-14

4. 收起所有项目列表

单击项目窗口工具栏上的Collapse All，如图 4-15 所示。

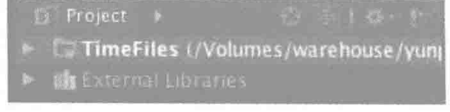


图 4-15

快捷键：command + "-"（macOS）或Ctrl + NumPad + "-"（Windows/Linux）

5. 扁平化显示包名

勾选Flatten Packages（扁平化显示包名），所有包名都会显示在同一级目录下，不会有层次结构，否则项目窗口中包名的显示是有层次结构的。

操作步骤：项目窗口→工具栏→设置按钮→Flatten Packages，如图 4-16 所示。

Flatten Packages选项勾选的效果对比如图 4-17 所示。

图 4-16

图 4-17

6. 隐藏空的中间包名

当我们勾选了Flatten Packages的时候会显示Hide Empty Middle Packages。勾选之后，项目窗口中的包名中间有空文件夹的包名将会被隐藏。

操作步骤：项目窗口→工具栏→设置按钮→Hide Empty Middle Packages。

Hide Empty Middle Packages选项勾选效果对比如图 4-18 所示。

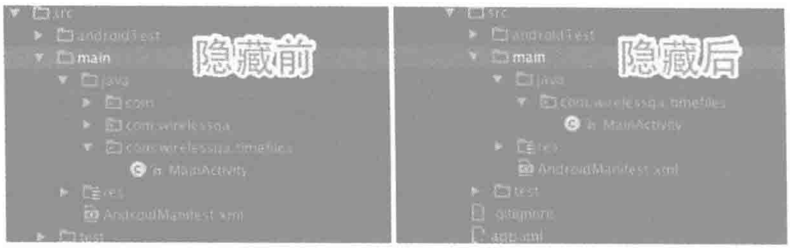


图 4-18

隐藏空的中间包名后项目列表会更加清晰，当我们想向空的文件夹中添加文件时可以不隐藏空的中间包名。

7. 缩写限定包名

如果想让项目窗口中显示缩写的包名，需要勾选此选项，否则项目窗口中会显示完整的包名。

操作步骤：项目窗口→工具栏→设置按钮→Abbreviate Qualified Package Names。

Abbreviate Qualified Package Names选项勾选效果对比如图 4-19 所示。

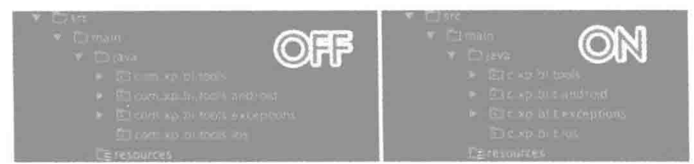


图 4-19

8. 显示成员

如果想让项目窗口中的文件显示其他成员（字段、方法），需要勾选此选项，否则只会显示类文件。

操作步骤：项目窗口→工具栏→设置按钮→Show Members。

Show Members选项勾选效果对比如图 4-20 所示。

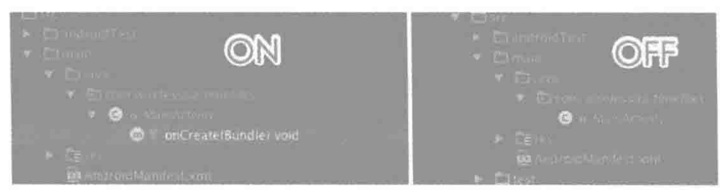


图 4-20

9. 按类型排序

如果想让文件按类型排序，需要勾选此选项，否则项目窗口中的文件是按字母顺序排序的。

操作步骤：项目窗口→工具栏→设置按钮→Sort By Type。

Sort By Type选项勾选效果对比如图 4-21 所示。

10. 文件夹显示在文件之前

如果想让所有文件夹始终显示在文件之前，需要勾选此选项，否则所有的文件和文件夹都会混在一起，按字母顺序排序。

操作步骤：项目窗口→工具栏→设置按钮→Folder Always on Top。

Folder Always on Top选项勾选效果对比如图 4-22 所示。

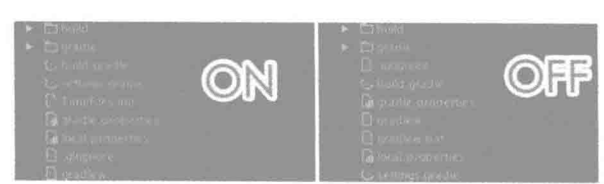


图 4-21

图 4-22

11. 单击文件打开源码

如果想在项目窗口中单击文件就能够在编辑器中自动打开（见图 4-23），需要勾选此选项。

操作步骤：项目窗口→工具栏→设置按钮→Autoscroll to Source。

图 4-23

12. 从源码定位到文件

如果想打开源码后就自动定位到该文件在项目窗口中的位置（见图 4-24），需要勾选此选项。

操作步骤：项目窗口→工具栏→设置按钮→Autoscroll from Source。

图 4-24

13. 视图标签分组

操作步骤：项目窗口→工具栏→设置按钮→Group Tabs。

Group Tabs选项勾选效果对比如图 4-25 所示。

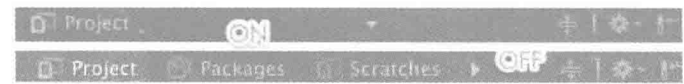


图 4-25

4.2 项目管理

4.2.1 打开和关闭项目

1. 打开一个Android项目

操作步骤：菜单栏→File→Open，或者欢迎界面→Open an existing Android Studio project（见图 4-26），然后在弹出的选择界面选择一个本地Android Studio项目，如图 4-27 所示。

图 4-26

图 4-27

2. 打开一个最近打开过的项目

默认Android Studio会记录 50 个你最近打开过的项目，当你想打开一个最近打开过的项目时，可以在最近打开项目列表中选择。

操作步骤：菜单栏→File→Open Recent→显示最近打开过的项目列表→选中一个项目打开，然后会弹出对话框，需要确认以什么样的方式打开项目，如图 4-28 所示。

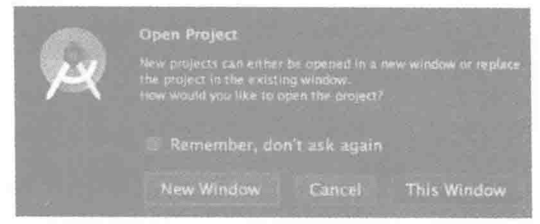


图 4-28

- 选择 New Window：会再打开一个 Android Studio 窗口。
- 选择 This Window：当前项目被关闭，打开所选中的项目。
- 选择 Cancel：取消本次操作。
- 勾选 Remember，don's ask again：会记住你的选择，以后就不用再选了。

3. 设置项目的打开方式

当我们打开一个新项目时默认会弹出一个确认提示，让我们确认是在新窗口打开或者在当前窗口打开项目。如果想改变默认设置，应该如何操作呢？

操作步骤：偏好设置→Appearance & Behavior→System Settings→设置Project Opening，如图 4-29 所示。

- Open project in new window：在一个新窗口打开项目。
- Open project in the same window：在同一个窗口打开项目（原来的项目会被关闭）。
- Confirm window to open project in：弹出确认对话框，可选择在新窗口或原窗口打开。

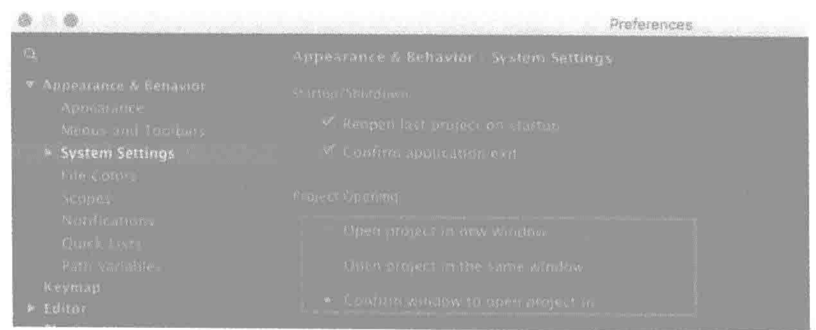


图 4-29

4. 关闭当前项目

操作步骤：菜单栏→File→Close Project→当前项目会被直接关闭。

4.2.2 管理最近打开的项目

Android Studio允许记录最近打开的项目，如果记录的项目比较多时，我们想找到需要的项目就会比较困难，因此Android Studio提供了管理最近打开的项目功能，允许我们在项目记录列表中搜索想要的项目，还支持对最近打开的项目进行分组、更换图标和移除等操作。

1. 搜索最近打开的项目

操作步骤：菜单栏→File→Open Recent→Manager Projects→在打开的界面搜索想要的项目名→按回车键即可打开，如图 4-30 所示。

注意，这里默认会在一个新窗口打开项目。

图 4-30

2. 为最近打开的项目分组

操作步骤：菜单栏→File→Open Recent→Manager Projects。

如果要新建一个分组，右击→New Project Group（见图 4-31）→输入分组名→OK。

如果要把项目添加到分组，右击→Move To Group→在列出的项目分组中选择目标分组。

图 4-31

3. 改变项目默认图标

操作步骤：菜单栏→File→Open Recent→Manager Projects…→Change Icon。

4. 从欢迎界面移除

操作步骤：菜单栏→File→Open Recent→Manager Projects…→Remove Selected from Welcome Screen。

5. 设置记录最近打开项目的个数

Android Studio默认记录 50 个最近打开过的项目，如果超出这个数字就会用最新的覆盖最旧的项目记录，如何调整这个数字呢？

操作步骤： 偏好设置→Editor→单击General→在Limits中调整Recent files limit个数，如图 4-32 所示。

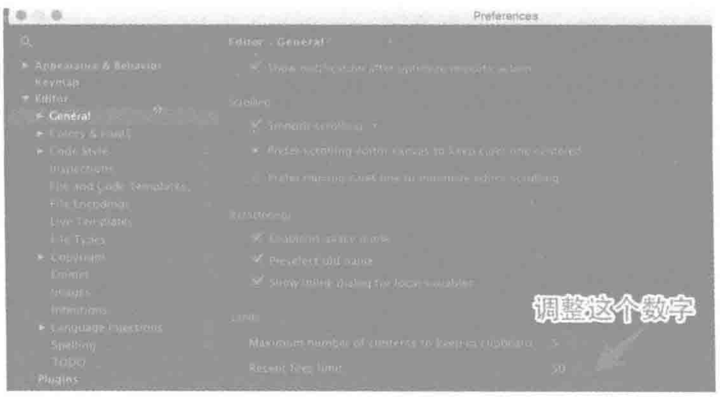


图 4-32

4.3 文件管理

4.3.1 文件同步

如果我们在Android Studio之外直接对文件做了更改，可能需要使用文件同步功能将修改同步过来。

操作步骤： 菜单栏→File→Synchronize（同步），或者利用快捷键Option +command + Y（macOS）或Ctrl + Alt + Y（Windows/Linux），然后将从文件系统中加载文件，如果有未保存的更改，就会提示你是否放弃更改。

4.3.2 导出到HTML

我们可以把代码以HTML的格式导出。

操作步骤： 菜单栏→File→Export to HTML（导出到HTML），如图 4-33 所示。

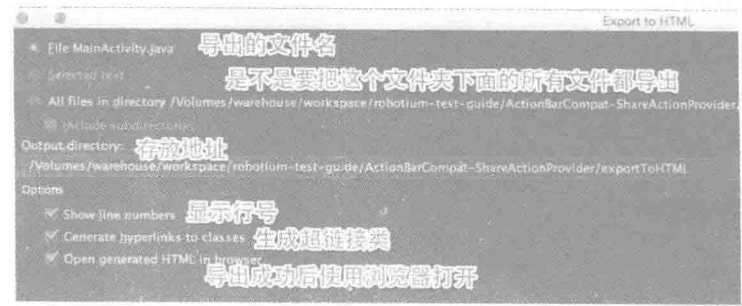


图 4-33

导出后的结果如图 4-34 所示。

图 4-34

4.3.3 切换文件编码方式

当我们只想切换当前正在查看文件的编码方式的时候，应该如何操作呢？

操作步骤：菜单栏→File→File Encoding（见图 4-35），或者使用更加快捷的方式，在状态栏进行切换（见图 4-36 所示）。

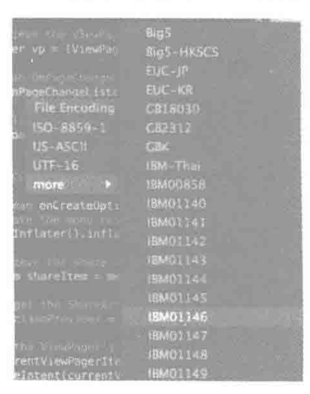


图 4-35

图 4-36

4.3.4 切换行分隔符

操作步骤：菜单栏→File→Line Separators （行分隔符，见图 4-37），或者使用更加快捷的切换行分隔符的方式，在状态栏进行切换（见图 4-38）。

图 4-37

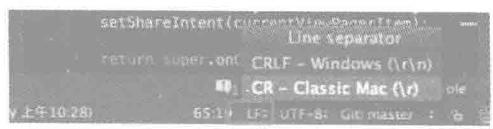


图 4-38

4.3.5 使文件只读

当我们想让某个文件只能看不能修改的时候需要将文件设置为只读。

操作步骤：菜单栏→File→Make File Read-only （使文件只读），如图 4-39 所示。

图 4-39

4.3.6 使用省电模式

使用省电模式的同学要注意了，如果勾选了此项，Android Studio就会去掉所有自动完成的功能，比如代码提示、代码检查等功能。那什么时候使用这个功能呢？当然是电池不够用又没带电源的时候了。

操作步骤：菜单栏→File→Power Save Mode （省电模式），如图 4-40 所示。

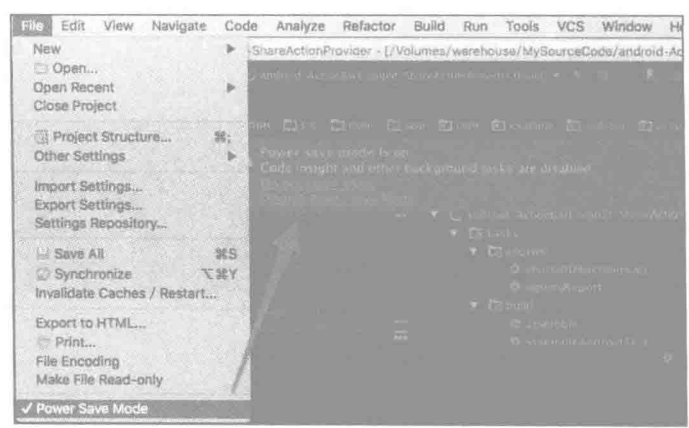


图 4-40

4.3.7 打开文件/文件夹所在磁盘目录

1. 在项目窗口中打开

操作步骤：右击文件/文件夹→Reveal in Finder（macOS，见图 4-41）或Show in Explorer（Windows/Linux），然后就会打开文件/文件夹所在磁盘目录（见图 4-42）。

2. 在编辑器中打开

操作步骤：快捷键command + 单击标签（macOS）或Ctrl + 单击标签（Windows/Linux），如图 4-43 所示。

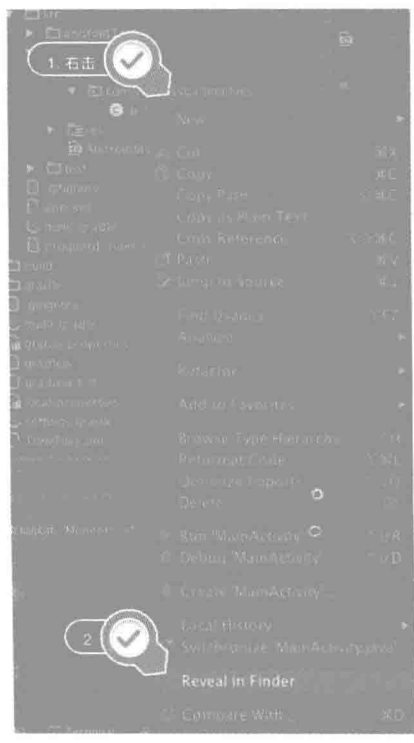


图 4-41

图 4-42

图 4-43

4.4 Android Studio 管理

4.4.1 隐藏/显示/退出Android Studio

1. 隐藏Android Studio

菜单栏（macOS）：Android Studio→Hide Android Studio
快捷键（macOS）：command + H

2. 隐藏所有打开的界面，只保留Android Studio

菜单栏（macOS）：Android Studio→Hide Others
快捷键（macOS）：option +command + H

3. 显示全部

隐藏了所有打开的界面，只保留了Android Studio，还可以显示全部。

菜单栏（macOS）：Android Studio→Show All

4. 退出Android Studio

菜单栏（macOS）： Android Studio→Quit Android Studio
快捷键（macOS）： command + Q

4.4.2 清除缓存/重启Android Studio

当我们直接修改了Android Studio的配置文件，但配置没有生效或遇到一些莫名其妙的问题时，可以尝试清除缓存再重启Android Studio来解决问题。

操作步骤： 菜单栏→File→Invalidate Caches / Restart （清除缓存/重启）→进入确认界面，如图 4-44 所示。

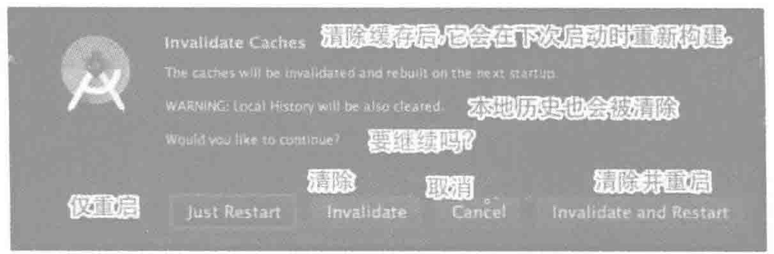


图 4-44

4.5 收藏夹

4.5.1 添加到收藏夹

在浏览器中，收藏夹用来收藏自己喜欢、常用的网址，我们把这些网址放到一个文件夹里，想用的时候可以通过收藏夹快速找到并打开某个网址。Android Studio中的收藏夹与此类似。

在使用Android Studio的日常编码中，如果某个文件或某段代码是我们经常需要查看或使用的，就可以把它添加到收藏夹中，以便快速查看。

打开收藏夹工具窗口（见图 4-45），从中可以看出收藏夹中支持收藏项目中的文件、书签和断点。

图 4-45

1. 打开收藏夹

快捷键： command + 2（macOS）或Alt + 2（Windows/Linux）
工具栏： 从左侧工具栏快速打开（见图 4-46）。

2. 添加到收藏夹

操作步骤： 菜单栏→File，或者右击文件→Add to Favorites。

图 4-46

【实例演示】

例 1：添加文件到收藏夹中（见图 4-47）。

01 在项目窗口的文件列表中选中文件。

02 右击文件→Add to Favorites。

03 选中已存在的收藏夹中的收藏列表或添加到一个新建的收藏列表。

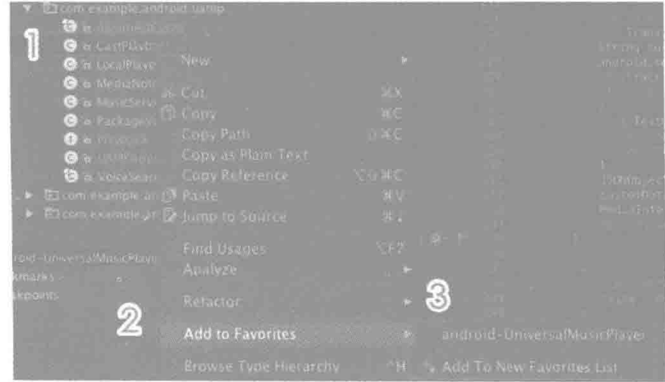


图 4-47

在这里我们选中已存在的收藏列表，结果如图 4-48 所示。双击文件，可在编辑器中打开这个文件。

图 4-48

例 2：添加一个方法到收藏夹。

01 将光标放在方法名上。

02 菜单栏→File→Add to Favorites→Add New Favorites List。

03 输入收藏列表的名字→OK（见图 4-49），结果如图 4-50 所示。

图 4-49

图 4-50

提 示

在编辑器中，如果光标没有选中方法，那么使用添加到收藏功能是将整个文件收藏；如果光标在某个方法上，那么使用添加到收藏功能是将选中的方法收藏。

例 3：添加断点到收藏夹。

添加断点很简单，只要在编辑窗口的左边栏单击，打上一个断点之后，这个断点就会自动被添加到收藏夹中，如图 4-51 所示。

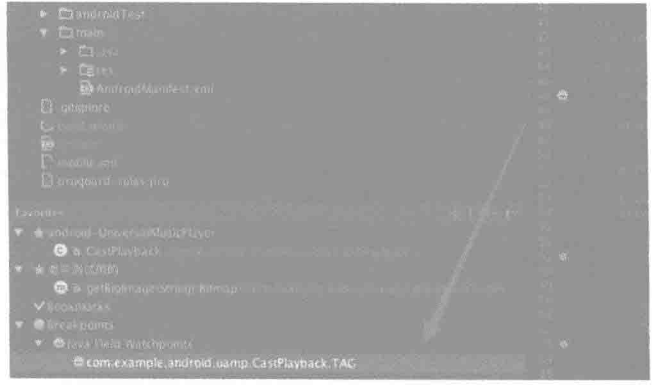


图 4-51

4.5.2 管理收藏夹

我们可以直接在收藏夹中对收藏的文件或方法进行操作。右击收藏的文件或方法即可看到操作列表，跟在项目列表中的操作是一样的，如图 4-52 所示。

我们还可以在收藏夹中对收藏列表或收藏的内容进行增、改、删（见图 4-53），以及移动已收藏的文件或代码（见图 4-54），也可以直接拖动。

图 4-52

图 4-53

图 4-54

正常情况下，在收藏夹中需要双击才能打开相应的源码，如果想单击一下就立即打开源码，需要在工具窗口中设置一下（单击右窗口右上角的设置按钮），选中Autoscroll to Source，如图 4-55 所示。

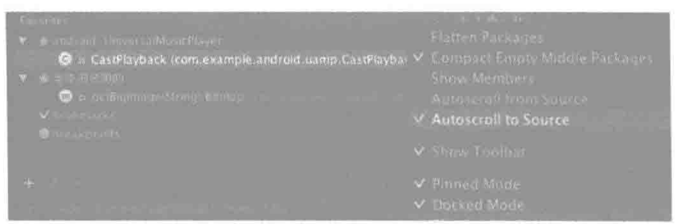


图 4-55

4.6 TODO

在Android Studio中待处理的任务可以在注释中使用TODO来标识，以表明这个地方是需要注意的，既有可能是未完成的功能，也有可能是需要解决的BUG或者是需要优化的代码。在项目开发的过程中，团队成员们可以通过TODO来关注这些问题，以便更好地完成开发。

4.6.1 添加TODO任务

添加TODO任务很简单，只需要在注释中输入TODO关键字就可以了。

操作步骤：添加注释→输入TODO→添加说明。

【实例演示】

java文件：

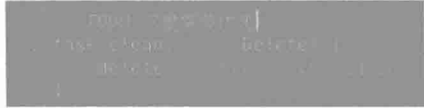
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.[illegible]);

    // TODO [illegible]
}
```

xml文件：

```
<application
    android:allowBackup=[illegible]
    android:icon=[illegible]
    android:label=[illegible]
    android:supportsRtl=[illegible]
    android:theme=[illegible]>
    <!-- TODO [illegible] -->
    <activity
```

gradle文件：



4.6.2 查看TODO任务

我们可以在TODO工具窗口中查看TODO任务。

操作步骤：菜单栏→View→TODO，打开TODO工具窗口，如图 4-56 所示。

从工具窗口中可以看出，当前项目中有 3 个TODO任务，分别在 3 个文件中。

图 4-56

4.6.3 TODO工具窗口常用操作

1. 指定范围

工具窗口中默认显示项目中所有的TODO任务，当然也可以指定范围。切换工具窗口的标签来指定查看范围，如图 4-57 所示。

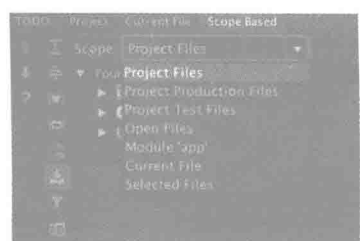


图 4-57

2. 展开和收起

展开和收起如图 4-58 所示。

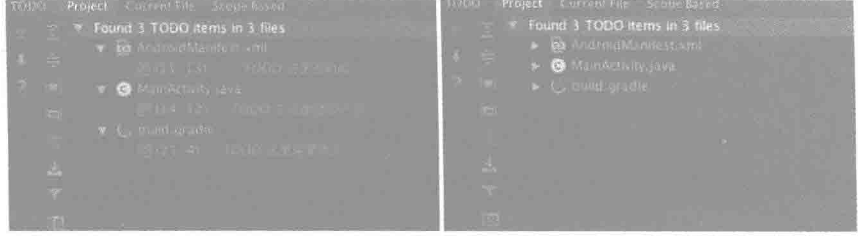


图 4-58

快捷键：command + "+" 和command + "-"（macOS），或者Ctrl + NumPad + "+" 和 Ctrl + NumPad + "-"（Windows/Linux）

3. 任务间上下跳转

光标定位在某个TODO任务上，可以使用快捷键快速上下跳转来查看。

快捷键：Alt +command + 向上箭头和Alt +command + 向下箭头（macOS），或者Ctrl + Alt + 向上箭头和Ctrl + Alt + 向下箭头（Windows/Linux）

4. 按模块分组

按模块分组如图 4-59 所示。

快捷键：command + M（macOS）或者Ctrl + M（Windows/Linux）

5. 按包名分组

按包名分组如图 4-60 所示。

图 4-59

图 4-60

快捷键：command + P（macOS）或者 Ctrl + P（Windows/Linux）

6. 扁平化显示包名

扁平化显示包名如图 4-61 所示。

快捷键：command + F（macOS）或者 Ctrl + F（Windows/Linux）

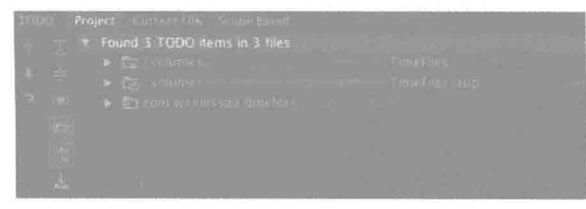


图 4-61

7. 自动打开源码

自动打开源码，如图 4-62 所示。

选中自动打开源码后，单击TODO任务时会自动打开源码，否则只能通过双击或右键菜单来打开源码，如图 4-63 所示。

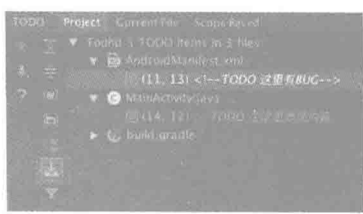


图 4-62

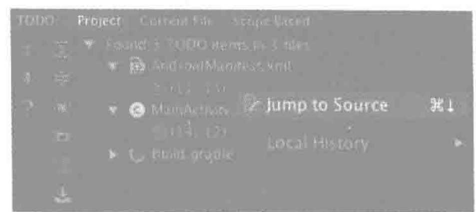


图 4-63

8. 自动预览源码

如图 4-64 所示，选中自动预览源码后，单击TODO任务时会自动打开源码预览窗口，达到快速预览的目的，否则只能打开源码在编辑器中查看，效率不高。

图 4-64

4.6.4 设置TODO

设置TODO界面既可以从偏好设置中打开，也可以从TODO工具窗口打开。虽然路径不同，但设置项是相同的。下面以偏好设置为例进行讲解。

1. 设置TODO关键字匹配区分大小写

默认TODO关键字匹配不区分大小写，因此不管是TODO还是todo都被识别为TODO任务，如图 4-65 所示。

如果想区分大小写，可以在偏好设置中选择Editor→TODO→在需要区分大小写的匹配前面勾选Case Sensitive，如图 4-66 所示。

图 4-65

图 4-66

默认的匹配都是小写的，也可以修改为大写的，如图 4-67 所示。

勾选Case Sensitive之后确定设置，todo就不会被识别为TODO任务了，如图 4-68 所示。

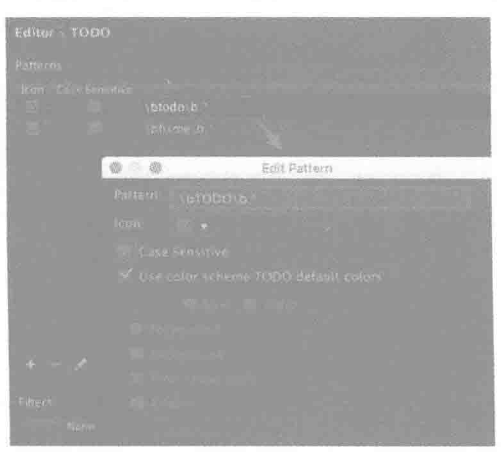


图 4-67

图 4-68

2. 自定义TODO关键字

当我们想通过TODO来分配任务的时候，使用自定义关键字会更好一点。比如：todo-me是分配给我自己的任务，todo-xiaoming是分配给小明的任务。

操作步骤：偏好设置→Editor→TODO→添加按钮→输入匹配表达式→选择icon→设置颜色，如图 4-69 所示。

确定后，使用效果如图 4-70 所示。

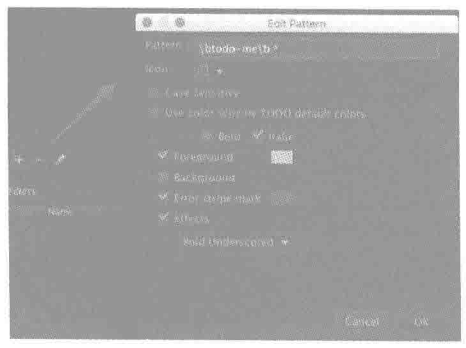


图 4-69

图 4-70

3. 设置过滤任务

前面我们讲了可以自定义TODO关键字来分配任务，但如果任务太多、关键字太多，想要在众多的任务中过滤出自己的任务时，可以在偏好设置中选择Editor→TODO→在Filters中单击添加按钮→输入Name→选择匹配，如图 4-71 所示。

确定后，就可以在TODO工具窗口中过滤任务了，如图 4-72 所示。

图 4-71

图 4-72

4.6.5 使用TODO分配代码Review任务

需求

（1）使用TODO来给团队成员分配代码Review任务。

（2）可以快速输入TODO关键字，并且插入分配日期。

（3）每个人可以根据名字过滤出分配给自己的任务。

操作步骤

第 1 步：自定义TODO关键字Review。

偏好设置→Editor→TODO→添加按钮→输入匹配表达式→设置显示颜色，如图 4-73 所示。

图 4-73

第 2 步：添加活动模板。

01 偏好设置→Editor→Live Templates→新建一个分组：me。

02 新建一个活动模板→输入缩写（Abbreviation）和描述→输入模板（见图 4-74）。

03 单击 Define→选择 Everywhere。

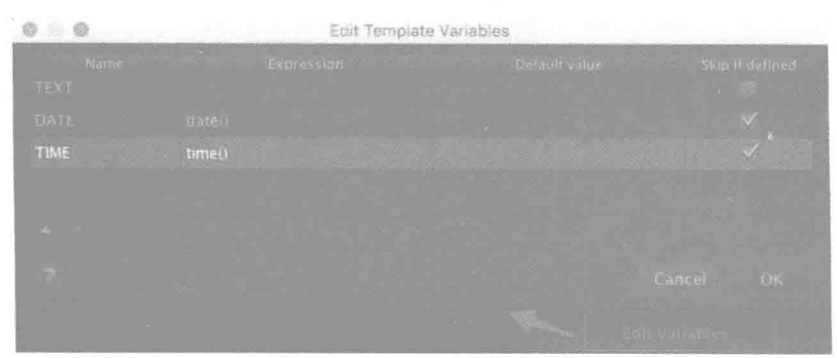


图 4-74

04 单击 Edit variables→定义日期和时间变量，如图 4-75 所示。

图 4-75

05 确定后自定义的动态模板就配置完成了。

如果需要添加其他的动态模板，可以按照添加活动模板这 5 个步骤自行添加，或直接复制后修改，如图 4-76 所示。

第 3 步：使用动态模板快速插入REVIEW任务，颜色可以自己定义。

01 在注释中输入 rvm/rvx/rvl，然后按下 Tab 键。

02 输入描述文本，结果如图 4-77 所示。颜色可以自己定义。

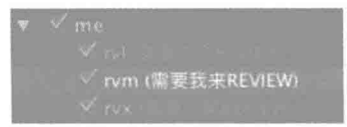


图 4-76

图 4-77

第 4 步：设置过滤任务关键字，只过滤出分配给我的REVIEW任务。

在TODO工具窗口单击Filter TODO Items→Edit Filters→在Patterns添加一个匹配表达式（\bREVIEW\b.*关键字.*）→在Filters中选择这个匹配，再起个名字，如图 4-78 所示。

图 4-78

然后就可以过滤出分配给自己的任务了，如图 4-79 所示。

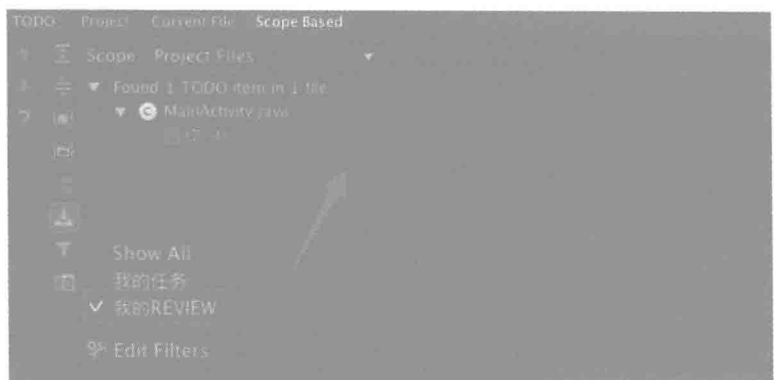


图 4-79

还有很多地方可以使用TODO来提醒我们完成必要的工作，请举一反三，自己发掘吧！

第 5 章　编　辑

复制、粘贴、选择、查找、替换应该是我们在编写代码时最常用的操作了，Android Studio可以让这些操作变得简单和高效。

本章向大家介绍在Android Studio中如何快速地使用这些操作技巧。

本章重要知识点 >>>>>>>>>>

- 如何使用复制和选择技巧；
- 如何使用查找和替换技巧；
- 宏的操作和管理。

5.1　撤消/重做/剪切/复制/粘贴

撤消、重做、剪切、复制、粘贴是最常用的编辑功能，既可以在工具栏上快速地进行操作，如图 5-1 所示，从左到右依次是撤消、重做、剪切、复制、粘贴，也可以在菜单栏中进行操作。

图 5-1

（1）撤消

菜单栏：Edit→Undo

快捷键：command + Z（macOS）或者Ctrl + Z（Windows/Linux）

（2）重做

菜单栏：Edit→Redo

快捷键：shift + command + Z（macOS）或者Ctrl + Shift + Z（Windows/Linux）

（3）剪切

菜单栏：Edit→Cut

快捷键：command + X（macOS）或者Ctrl + X（Windows/Linux）

（4）复制

菜单栏：Edit→Copy

快捷键：command + C（macOS）或者Ctrl + C（Windows/Linux）

（5）粘贴

菜单栏：Edit→Paste

快捷键：command + V（macOS）或者Ctrl + V（Windows/Linux）

最好记住快捷键，熟练使用快捷键是提升工作效率的关键。

5.2 复制技巧

5.2.1 复制为纯文本

普通的复制可能会带有一些格式，如果只想要纯文件，就使用Copy as Plain Text。

操作步骤：菜单栏→Edit→Copy as Plain Text。

【实例演示】

复制为纯文件跟普通复制的效果对比图如图 5-2 所示。

图 5-2

5.2.2 复制引用

复制引用是指复制一个类名或方法名的时候会把整个路径复制下来，效果如图 5-3 所示。

菜单栏：Edit→Copy Reference

快捷键：Alt + command + Shift + C（macOS）或者Ctrl + Alt + Shift + C（Windows/Linux）

图 5-3

5.2.3 从复制历史中选择粘贴

Android Studio记录了最近复制过的历史列表，可以从这个列表中选择内容进行粘贴。

菜单栏：Edit→Paste from History

快捷键：Shift + command + V（macOS）或者Ctrl + Shift + V（Windows/Linux）

然后弹出复制过的历史列表，以供选择，如图 5-4 所示。

光标定位在历史列表中，直接输入关键字就可以过滤出相关内容，如图 5-5 所示。

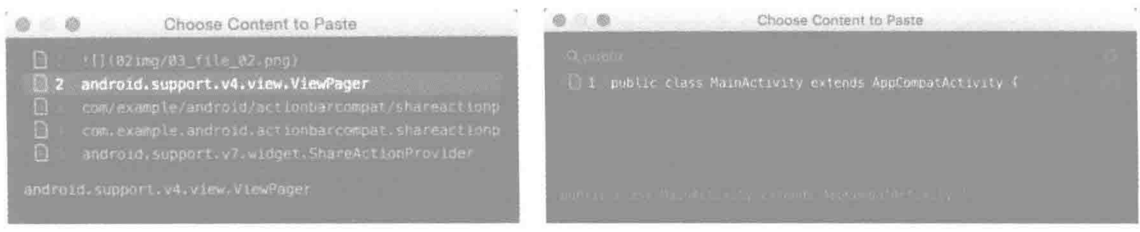


图 5-4　　　　图 5-5

光标定位在内容详情中，通过command + F （Windows/Linux：Ctrl + F ）可调出查找工具栏，如图 5-6 所示。

图 5-6

5.2.4 设置粘贴历史记录的个数

Android Studio默认记录 5 个粘贴历史，但这个值是可以调整的。如果要调整这个限制，可以选择Preferences→Editor→单击General→在Limits中调整Maximum number of contents to keep in clipboard的个数，如图 5-7 所示。

图 5-7

5.2.5 复制行

在Android Studio中可以快速复制一行内容。

菜单栏：Edit→Duplicate Line

快捷键：command + D（macOS）或者Ctrl + D（Windows/Linux）

【实例演示】

01 将光标放在要复制行的任意位置。

02 按快捷键 command + D（macOS），成功复制当前行。

5.3 合并两行内容

通常我们要合并两行内容都是狂按删除键来删除两行之间的空格，然后对代码的格式进行调整。Android Studio给我们提供了一个自动合并两行内容的功能，可以智能地合并字符串、注释、声明和赋值，一键完成上面的这些操作。

菜单栏：Edit→Join Lines

快捷键：control + Shift + J（macOS）或者Ctrl + Shift + J（Windows/Linux）

【实例演示】

01 合并两个语句，如图 5-8 所示。

02 合并两行字符串，如图 5-9 所示。

图 5-8

图 5-9

03 合并两行注释，如图 5-10 所示。

04 合并声明和赋值，如图 5-11 所示。

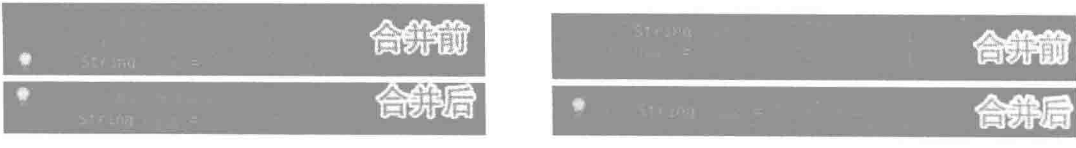


图 5-10

图 5-11

合并声明和赋值这个操作正好跟分离声明和赋值相反。

5.4 选择技巧

5.4.1 扩大选择范围

扩大选择范围是指每执行一次扩大选择，选择范围就会相应扩大。

菜单栏：Edit→Extend Selection

快捷键：option + ↑（macOS）或者Ctrl + W（Windows/Linux）

【实例演示】

执行 4 次扩大选择范围的效果如图 5-12 所示。

图 5-12

5.4.2 缩小选择范围

缩小选择范围与扩大选择范围正好相反，每执行一次缩小选择，选择范围就会相应缩小。

菜单栏： Edit→Shrink Selection

快捷键： option + ↓（macOS）或者Ctrl + Shift + W（Windows/Linux）

5.4.3 使用列选择模式

一般选择内容时都是横向选择，如果想竖向选择应该如何操作呢？

菜单栏： Edit→Column Selection Mode （列选择模式）

快捷键： Shift + command + 8（macOS）或者Alt + Shift + Insert（Windows/Linux）

【实例演示】

开启列选择模式后状态栏会显示Column状态，如图 5-13 所示。

图 5-13

开启【列选择模式】后我们可以对列进行选择，如图 5-14 所示。

图 5-14

被选中的列可同时进行增、删、改，非常方便。

如果想退出列选择模式，只需要再次按快捷键Shift + command + 8（macOS）即可。

5.5 缩进设置

1. 缩进

菜单栏：Edit→Indent Selection （必须选中行）

快捷键：Tab（macOS/Windows/Linux）

【实例演示】

01 选中要缩进的代码片段。

02 执行快捷键 Tab，效果如图 5-15 所示。

2. 取消缩进

菜单栏：Edit→UnIndent Line or Selection

快捷键：Shift + Tab（macOS/Windows/Linux）

【实例演示】

01 选中要取消缩的进代码片段。

02 执行快捷键 Shift +Tab，效果如图 5-16 所示。

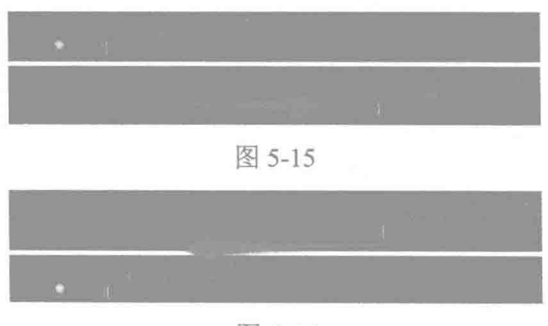


图 5-15

图 5-16

5.6 自动补全当前的语句

自动补全当前的语句可以帮助我们完成正在输入语句的剩余部分、自动增加漏掉的大括号小括号和必要的格式化处理。

菜单栏：Edit→Complete Current Statement

快捷键：Shift + command + 回车（macOS）或者Ctrl + Shift + 回车（Windows/Linux）

【实例演示】

例 1：自动补全方法声明。

01 输入一个方法时，光标在括号的里面或外面。

02 按快捷键 Shift + command + 回车（macOS）。

03 效果如图 5-17 所示，方法声明被补全了。

例 2：自动补全代码结构。

01 输入 if，将光标放在 if 右侧。

02 按快捷键 shift + command + 回车（macOS）。

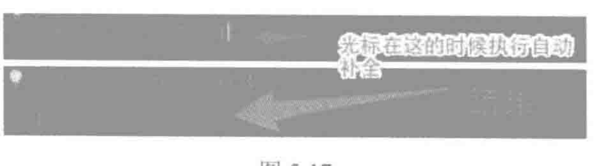


图 5-17

03 效果如图 5-18 所示，if 的代码结构被自动补全。

图 5-18

例 3：自动封装。

当在一个数据旁边输入新的方法调用时，Android Studio会自动封装方法的调用，如图 5-19 所示。

例 4：自动格式化（见图 5-20）。

图 5-19

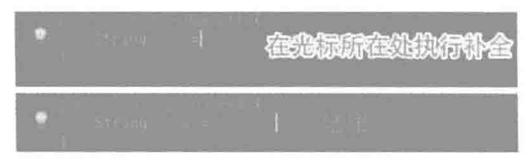


图 5-20

5.7 一键切换大小写字母

一键切换大小写字母的功能可以帮助我们一键切换选中字符的大小写。

菜单栏： Edit→Toggle Case

快捷键： Shift + command + U（macOS）或者Ctrl + Shift + U（Windows/Linux）

【实例演示】 将字符串“bixiaopeng”全部切换为大写。

```
String name = "bixiaopeng";
```

01 选中"bixiaopeng"。

02 按快捷键 Shift + command + U（macOS），bixiaopeng 变成了 BIXIAOPENG。

再执行一次第 2 步的快捷键，大写字母就会全变成小写字母。

5.8 查找工具栏

Android Studio中提供了非常方便的查找工具，既可以在编辑器的文件中使用，也可以在Android输出的日志中使用。

5.8.1 打开查找工具栏

快捷键： command + F（macOS）或者Ctrl + F（Windows/Linux）

无论是在编辑器还是在日志中，我们都可以使用快捷键快速调出查找工具栏（见图 5-21）。

图 5-21

5.8.2 快速查找

当我们想查找某一个单词的时候，只需要先复制这个单词，然后调出查找工具栏，这个单词就会被自动填入查找框，按回车键后就可以进行查找了。

01 复制单词：command + C（macOS）或者 Ctrl + C（Windows/Linux）

02 调出查找工具栏：command + F（macOS）或者 Ctrl + F（Windows/Linux）

03 开始查找：Enter（macOS）或者 Enter（Windows/Linux）

5.8.3 查找范围设置

在查找工具栏上可以设置查找范围，如图 5-22 所示。

图 5-22

- In Comments Only：仅在注释中搜索。
- In Literals Only：仅在文本中搜索。
- Except Comments：注释除外。
- Except Literals：文本除外。
- Except Comments and Literals：注释和文本除外。

【实例演示】

- 当勾选【In Comments Only】时仅匹配注释中的关键字（见图 5-23）。

图 5-23

- 当勾选【In Literals Only】时仅匹配文本中的关键字，注释会被排除在外（见图 5-24）。

图 5-24

5.9　在查找结果中跳转

当查找到多个结果时，如何在多个结果中跳转呢？

（1）利用工具栏（见图 5-25）。

图 5-25

（2）利用快捷键。

01 下一个匹配到的结果：command + G（macOS）或者 F3（Windows/Linux）。

02 上一个匹配到的结果：command + Shift + G（macOS）或者 Shift + F3（Windows/Linux）。

03 在匹配到的相同的结果间跳转：Enter。

5.10　选择查找结果

当我们查找到多个结果时，如何选择这些查找结果呢？

1. 同时选中所有查找到的结果

菜单栏：Edit→Find→Select All Occurrences

快捷键：control + command + G（macOS）或者Ctrl + Alt + Shift + J（Windows/Linux）

【实例演示】在当前文件中查找一个关键字“container”。

01 搜索“container”。

02 同时选中所有查找到的结果（按快捷键 control + command + G（macOS）），效果如图 5-26 所示。

图 5-26

2. 连续选择当前及下一个结果

假设当前选中了一个查找结果，还想继续选择下一个结果时可以按下述方法操作。

菜单栏：Edit→Find→Add Select for Next Occurrence

快捷键：control + G（macOS）或者Alt + J（Windows/Linux）

3. 逐个取消选择

菜单栏：Edit→Find→UnSelect Occurrence

快捷键：control + shift + G（macOS）或者Alt + Shift + J（Windows/Linux）

5.11 指定查找路径

除了在当前文件中查找以外，还可以在其他路径中查找。另外，还可以设置很多查找条件，以方便更加精确和快速地得到查找结果。

前提条件：打开查找路径设置窗口（见图 5-27）。

菜单栏：Edit→Find→Find in Path...

快捷键：shift + command + F（macOS）或者Ctrl + Shift + F（Windows/Linux）

切换到预览选项卡，如图 5-28 所示。

图 5-27

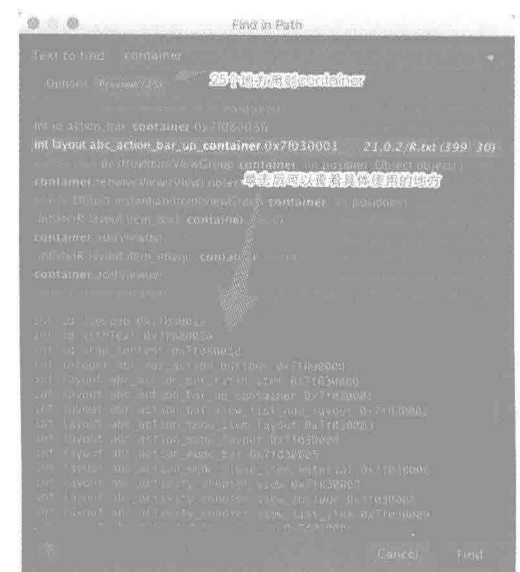


图 5-28

默认是查找整个项目，单击【Find】按钮，结果如图 5-29 所示。

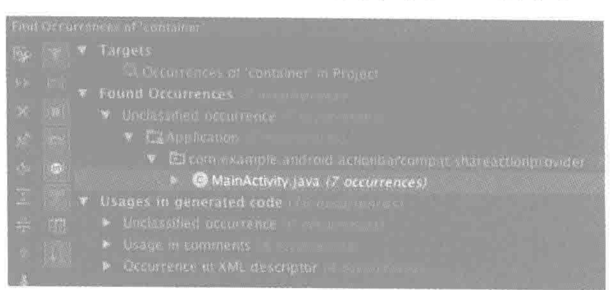


图 5-29

5.12 替 换

自古查找替换不分家，查找这么方便，替换当然也非常方便啦。

前提条件：调出查找替换工具栏（见图 5-30）。

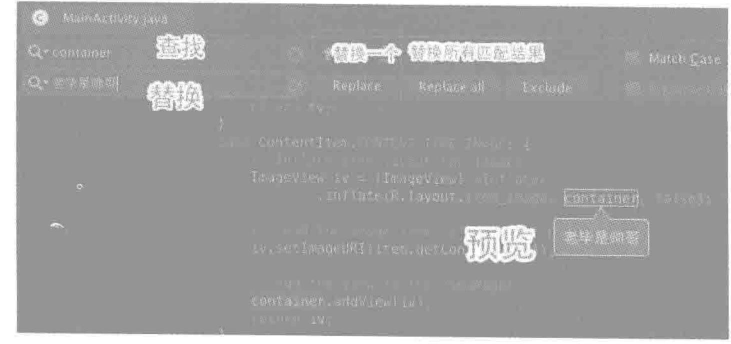


图 5-30

菜单栏：Edit→Find→Replace

快捷键：command + R（macOS）或者Ctrl + R（Windows/Linux）

如果选择排除（Exclude）某个匹配项，那么就算单击了【Replace all】按钮，被排除的地方也不会被替换，如图5-31所示。

图5-31

5.13 指定替换路径

除了在当前文件中查找并替换以外，还可以指定其他替换路径，并且可以设置很多查找替换的条件，以便更加精确和快速地得到替换结果。

前提条件：打开查找路径设置对话框。

菜单栏：Edit→Find→Replace

快捷键：command + shift + R（macOS）或者Ctrl + Shift + R（Windows/Linux）

打开指定替换路径对话框，如图5-32所示。

上面的例子是在整个项目中所有的地方查找“container”并替换为“老毕是帅哥”，在Preview（预览）中显示了有25个匹配结果。

（1）指定只在注释中查找，如图5-33所示。

（2）指定只查找xml文件，如图5-34所示。查看预览结果，如图5-35所示。

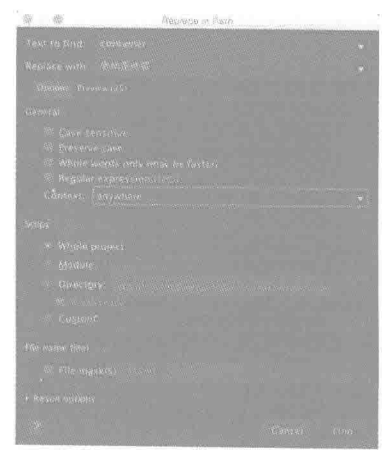


图5-32

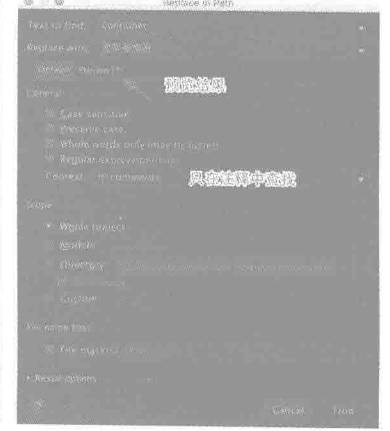


图5-33

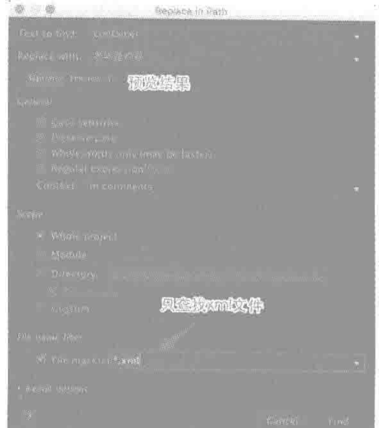


图5-34

这就非常精确地查找到了需要替换的地方。

（3）指定查找范围，如图 5-36 所示。

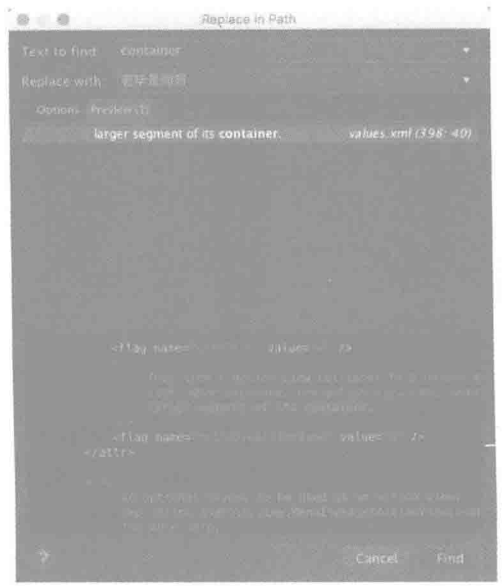


图 5-35

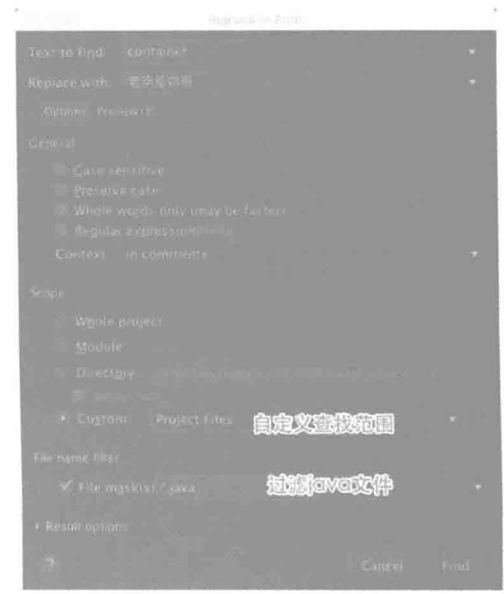


图 5-36

（4）指定替换步骤。单击Find→弹出确认对话框，如图 5-37 所示。

图 5-37

默认单击【Replace】是一个一个替换的（见图 5-38），也可以有其他选择。

图 5-38

- 单击【Skip】会跳过当前匹配结果。
- 单击【Replace All in This File】会替换当前文件中所有的匹配结果。
- 单击【Skip To Next File】会跳到下一个文件。
- 单击【All Files】就全部替换了。
- 单击【Review】可在工具栏显示查找结果，单击结果可以查看详情。

Replace Selected替换所选匹配项，Replace All全部替换。

（5）查找替换历史。

Android Studio会记录所有的查找替换历史和配置，当再一次进行查找的时候默认使用上一次的替换记录，如图 5-39 所示。

图 5-39

5.14 在结构中查找和替换

Android Studio提供了一个使用代码模板查找和替换的方式，可以方便地定义代码模板。

操作步骤：菜单栏→Edit→Find→Search Structurally/Replace Structurally→打开【Structural Search】，如图 5-40 所示。

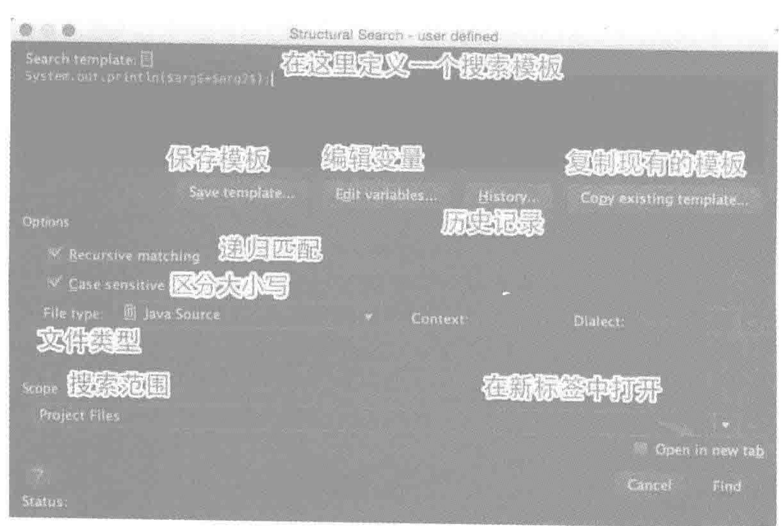


图 5-40

通过搜索模板System.out.println（arg+$arg2$）搜索后的结果如图 5-41 所示。

编辑变量，如图 5-42 所示。

历史记录，如图 5-43 所示。

图 5-41

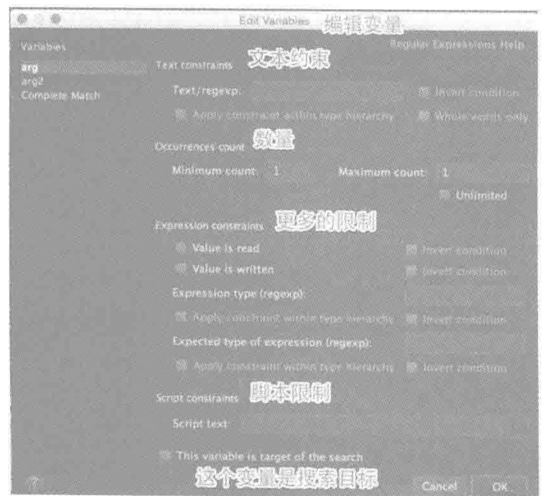


图 5-42

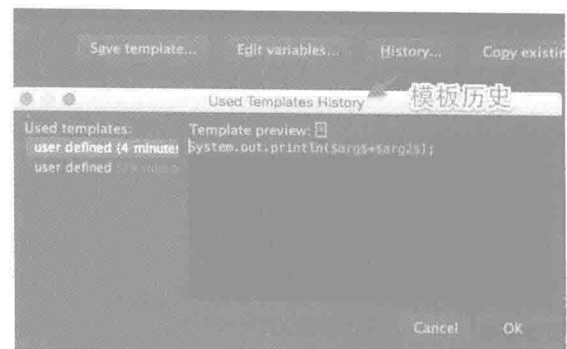


图 5-43

复制现有模板，如图 5-44 所示。

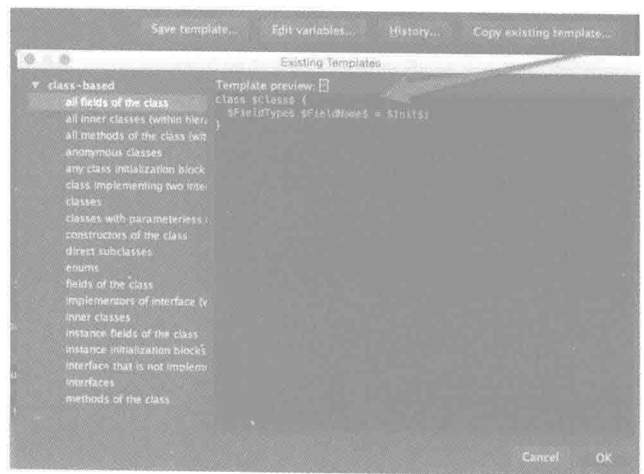


图 5-44

替换这里就不演示了，跟查找差不多。

5.15 查找用法

如果想查看某个方法或变量在哪些地方使用的时候，应该如何操作呢？

菜单栏：Edit→Find→Find Usages

右键菜单：Find Usages

快捷键：fn + option + F7（macOS）或者Alt + F7（Windows/Linux）

【实例演示】查找setShareIntent方法在哪些地方使用。

01 右击 setShareIntent 方法→在弹出的菜单中选择【Find Usages】。

02 查找结果显示在工具窗口中，如图 5-45 所示。

图 5-45

5.16 设置查找用法的过程和范围

如果我们想设置查找用法的过程和范围，应该如何操作呢？

菜单栏：Edit→Find→Find Usages Settings

快捷键：option + command + Shift + F7（macOS）或者Ctrl + Alt + Shift + F7（Windows/Linux）

查找不同的数据（字段、变量、参数、类、标记、属性等）会有不同的配置窗口。

1. 默认的设置项（见图 5-46）

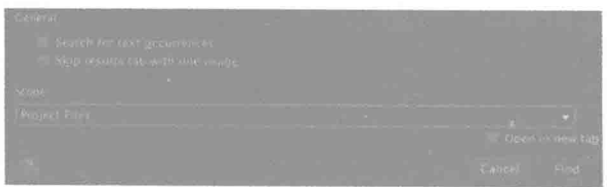


图 5-46

● Skip results tab with one usage：当搜索的数据只有一个地方使用时，不显示搜索结果标签。

例如，shareItem只在一个地方被使用，如果没有勾选Skip results tab with one usage，那么查找结果如图 5-47 所示；如果勾选了Skip results tab with one usage，查找结果则如图 5-48 所示。

图 5-47

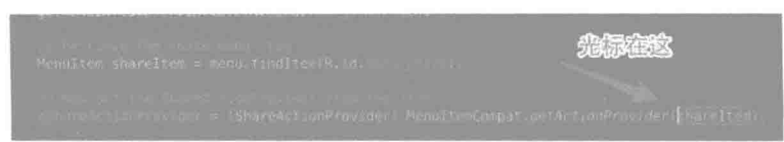


图 5-48

当只有一个地方使用时，光标会直接跳到使用的地方，不会打开一个标签显示结果。

● Scope：设置查找范围，使查找更加快速和精确，如图 5-49 所示。

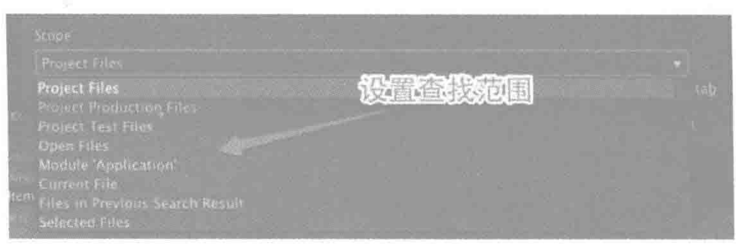


图 5-49

● Open in new tab：如果勾选了此项，每一次查找都会打开一个新的标签；如果不勾选此项，那么每次新的查找结果将会覆盖上一次的查找结果。

2. 查找类的使用设置（见图 5-50）

● Usages：勾选此项，会查找这个类所有的引用。

● Usages of methods：勾选此项，会查找被选定类方法的所有调用。

● Usages of fields：勾选此项，会查找选定类字段的用法。

● Derived classes：勾选此项，会查找选定类的所有扩展类。

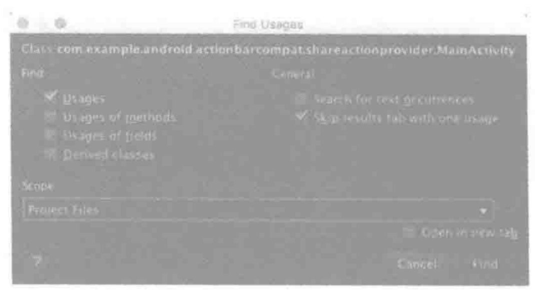


图 5-50

3. 查找构造函数的使用设置（见图 5-51）

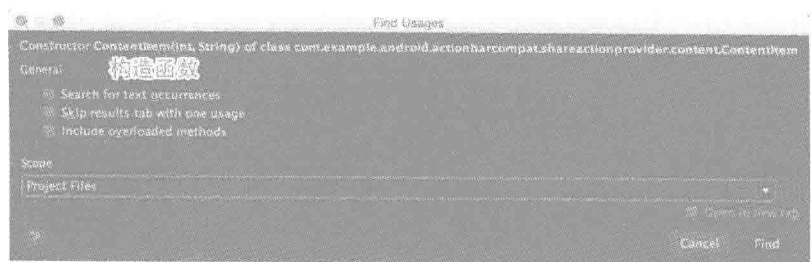


图 5-51

4. 查找方法的使用设置（见图 5-52）

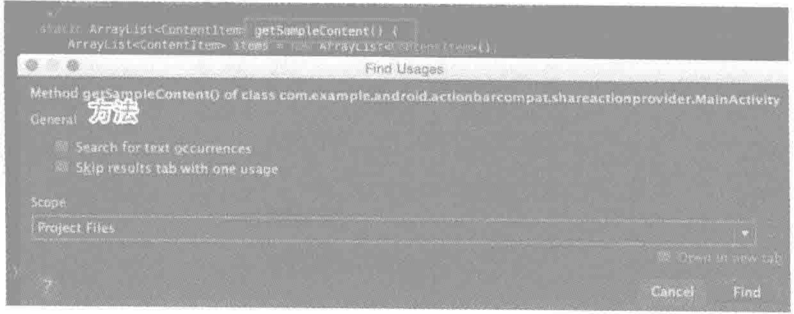


图 5-52

5. 查找变量的使用设置（见图 5-53）

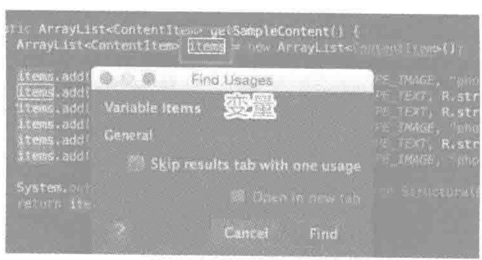


图 5-53

6. 查找包的使用设置（见图 5-54）

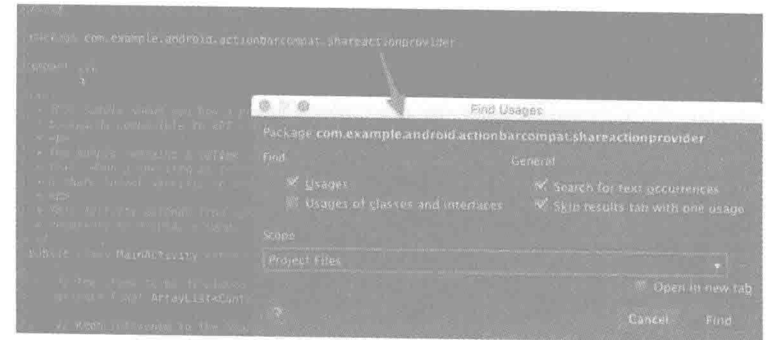


图 5-54

- Usages：通过名字查找选定包在源码中的所有引用。
- Usages of classes and interfaces：通过名字查找选定包在源码中所有类和接口的引用。

5.17 显示用法

显示用法功能不会另外打开一个工具窗口来显示查找结果，而是直接弹出提示框显示数据在何处被使用，查看起来更加方便。

菜单栏： Edit→Find→Show Usages

快捷键： fn + option + command + F7（macOS）或者Ctrl + Alt + F7（Windows/Linux）

效果如图 5-55 所示。

图 5-55

显示功能如图 5-56 所示。

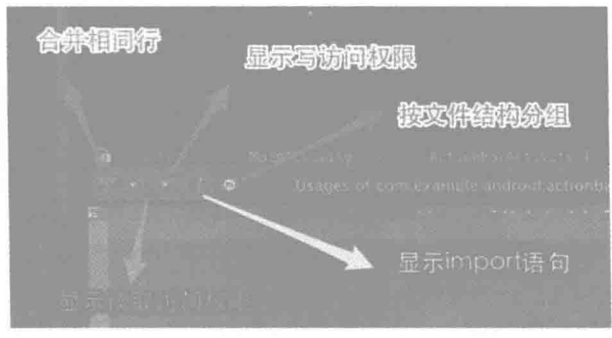


图 5-56

在提示框的最右边还有设置按钮，单击后进入查找使用设置界面。

5.18 查看在当前文件中的用法

如果想查看某个数据（字段、变量、参数、类、 标记、 属性等）在当前文件中的用法，应该如何操作呢？

菜单栏： Edit→Find→Find Usages in File

快捷键： command + F7（macOS）或者Ctrl + F7（Windows/Linux）

当前文件中所有结果都会被匹配到，通过Shift + Command + G能够在所有匹配结果中跳转。

5.19 在文件中高亮显示字符

查看文件时，如果想高亮显示某个字符在这个文件中的位置时，应该如何操作呢？

前提条件：光标放在这个字符上面。

菜单栏：Edit→Find→Highlight Usages in File

快捷键：fn + shift + command + F7（macOS）或者Ctrl + Shift + F7（Windows/Linux）

【实例演示】没有高亮显示的效果如图 5-57 所示。

图 5-57

高亮显示后的效果如图 5-58 所示。

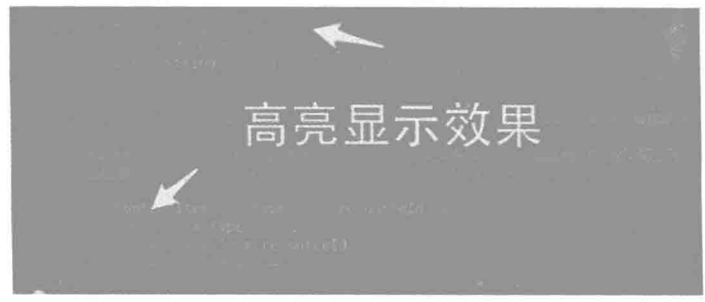


图 5-58

取消高亮显示，执行相同的操作（（macOS）fn + shift + command + F7）即可，或者按两下 Esc 键。

5.20 最近查找

如果想查看并使用最近的查找记录，应该怎么操作呢？

菜单栏：Edit→Find→Recent Find Usages

【实例演示】

如图 5-59 所示，查看最近的查找记录。

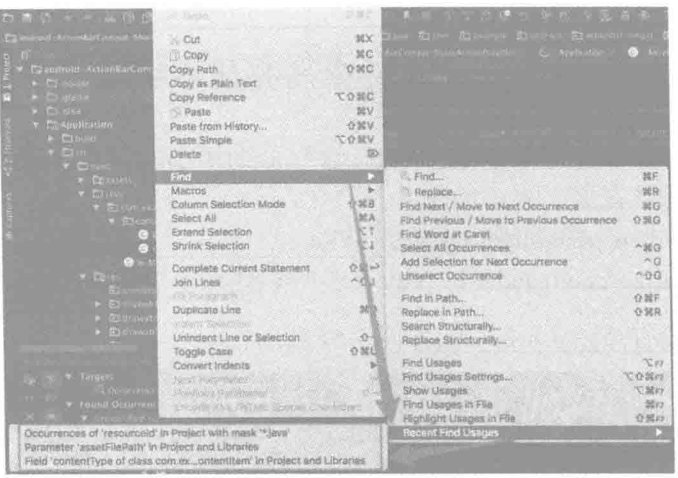


图 5-59

5.21 Macros （宏）

1. 什么是宏

宏是一种批处理的称谓，由一些命令组织在一起，作为一个单独的命令来完成一个特定任务。

Android Studio中支持录制宏的功能，如图 5-60 所示。

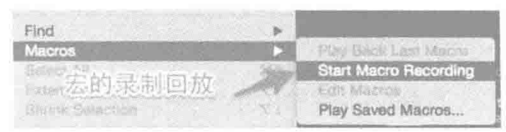


图 5-60

2. 录制回放宏

我们举个例子来看看如何录制回放宏，需求是自动保存代码，然后运行代码。

01 选择 Edit→Macros→Start Macros Recording→Android Studio 右下角显示开始录制提示。

02 按快捷键 command + s 保存文件。

03 按快捷键 control + s 运行代码（见图 5-61）。

04 选择 Edit→Macros→Stop Macros Recording→在输入框中输入已录制宏的名字，如图 5-62 所示。

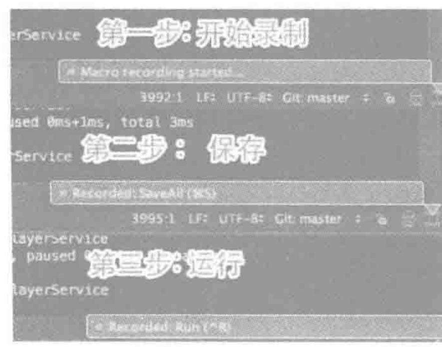


图 5-61

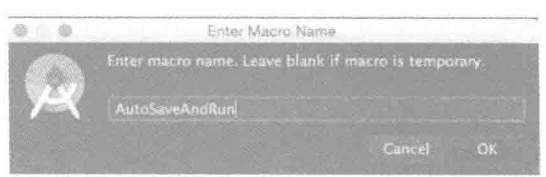


图 5-62

05 定义快捷键。选择 Android Studio→Keymap→Macros→找到刚才录制的宏【AutoSaveAndRun】，如图 5-63 所示。

选择添加键盘快捷键，然后进入编辑界面，如图 5-64 所示。

图 5-63

图 5-64

确定后就可以使用快捷键来执行宏了，显示效果如图 5-65 所示。也可以对比一下宏菜单列表中的显示，如图 5-66 所示。

图 5-65

3. 编辑宏

菜单栏：Edit→Macros→Edit Macros，如图 5-67 所示。

图 5-66

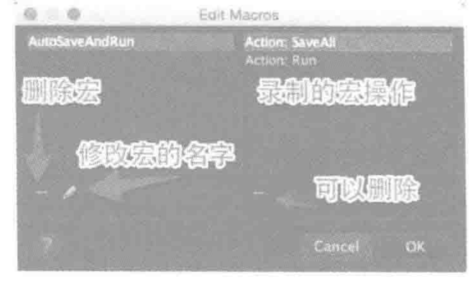


图 5-67

第 6 章　视　　图

Android Studio中的视图操作可以控制工作台和工具窗口的显示、查看文件和代码的文档信息、快速切换编辑器的显示方案。

本章将向大家介绍Android Studio中的视图操作。

本章重要知识点 >>>>>>>>>>

- 如何管理工具窗口和工作台;
- 如何查看参数和方法的文档信息;
- 如何查看最近打开和修改的文件;
- 如何对比文件和查看源码;
- 如何切换编辑器的显示方案和显示效果;
- 如何使用演示、免打扰和全屏模式。

6.1　工具窗口

6.1.1　显示/隐藏工具窗口

视图菜单中的Tool Windows提供了Android Studio中常用工具窗口的显示和隐藏功能。在这里我们可以查看到Android Studio中常用工具以及显示和隐藏它们的快捷键。

操作步骤: 菜单栏→View→Tool Windows，然后显示工具列表，如图 6-1 所示。

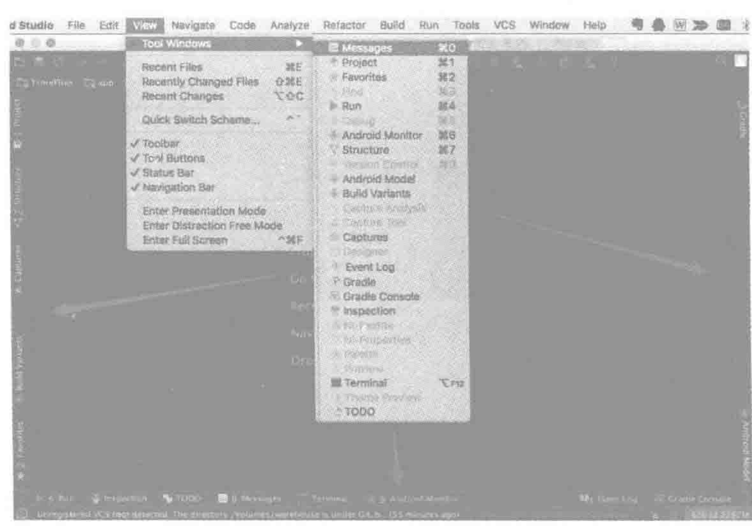


图 6-1

快捷键

- 打开/隐藏 项目工具窗口：command + 1（macOS）或者 Alt + 1（Windows/Linux）
- 打开/隐藏 收藏工具窗口：command + 2（macOS）或者 Alt + 2（Windows/Linux）
- 打开/隐藏 运行窗口：command + 4（macOS）或者 Alt + 4（Windows/Linux）
- 打开/隐藏 调试窗口：command + 5（macOS）或者 Alt + 5（Windows/Linux）
- 打开/隐藏 Android Monitor 工具窗口： command + 6（macOS）或者 Alt + 6（Windows/Linux）
- 打开/隐藏 项目结构窗口：command + 7（macOS）或者 Alt + 7（Windows/Linux）
- 打开/隐藏 版本控制工具窗口：command + 9（macOS）或者 Alt + 9（Windows/Linux）
- 打开/隐藏 终端工具窗口： fn + option + F12（macOS）或者 Alt + F12（Windows/Linux）

状态栏

将光标移动到工具窗口管理图标上就会弹出工具列表，如图 6-2 所示，我们可以单击工具来显示或隐藏工具窗口。

图 6-2

6.1.2 快速切换工具窗口

不使用鼠标，通过快捷键在工具窗口间快速切换：control + Tab（macOS）或者Ctrl + Tab（Windows/Linux）

【实例演示】

按下control + Tab键，弹出Switcher窗口，如图 6-3 所示。

按住control键不放，通过单击Tab切换窗口，松开后相关窗口打开。也可以按住control不放，再通过按下工具旁边的数字或字母打开相关窗口。

图 6-3

6.2 工作台管理

有些开发同学喜欢纯净的编辑器环境，不希望工作台周边有太多东西显示，而将工具栏、工具条、状态栏、导航栏全部隐藏，如图 6-4 所示。

```
layout/activity_relative_layout.xml -
layout/activity_relative_layout.xml
RelativeLayout  ImageView
<?xml version=     encoding=     ?>
<RelativeLayout xmlns:
    android:layout_width=
    android:layout_height=
    android:paddingLeft=
    android:paddingRight=     >

    <ImageView
        android:layout_width=
        android:layout_height=
        android:id=
        android:layout_alignParentTop=
        android:layout_centerHorizontal=
        android:layout_marginTop=
        android:background=
        android:contentDescription=     />

    <TextView
        android:layout_width=
Text  Design
```

图 6-4

问题来了，在Android Studio中如何管理工作台呢？又如何显示和隐藏工具栏、工具条、状态栏、导航栏？

1. 显示/隐藏工具栏（见图 6-5）

菜单栏：View→Toolbar

图 6-5 工具栏

2. 显示/隐藏工具条（见图 6-6）

菜单栏：View→Tool Buttons
状态栏：单击工具窗口管理按钮

3. 显示/隐藏状态栏（见图 6-7）

菜单栏：View→Status Bar

4. 显示/隐藏导航栏（见图 6-8）

菜单栏：View→Status Bar

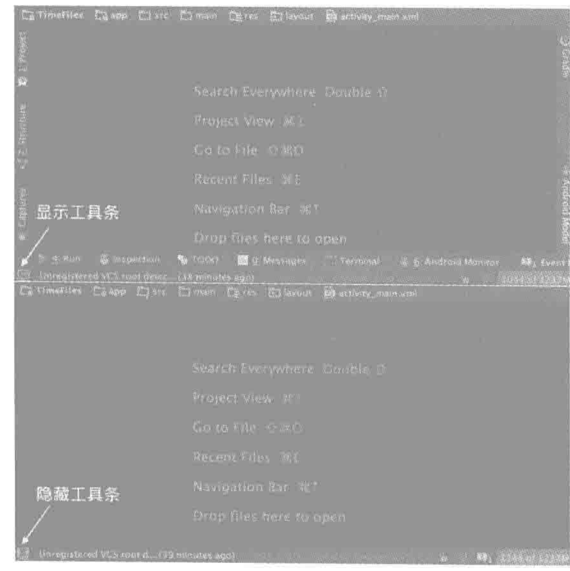


图 6-6 显示和隐藏工具条

图 6-7 状态栏

图 6-8 导航栏

6.3 查看定义

快速查看定义是指在不离开当前文件的情况下快速查看某个变量、类或方法的定义，如图 6-9 所示。

菜单栏：View→Quick Definition

快捷键：fn + option +空格（macOS）或者Ctrl + Shift + I（Windows/Linux）

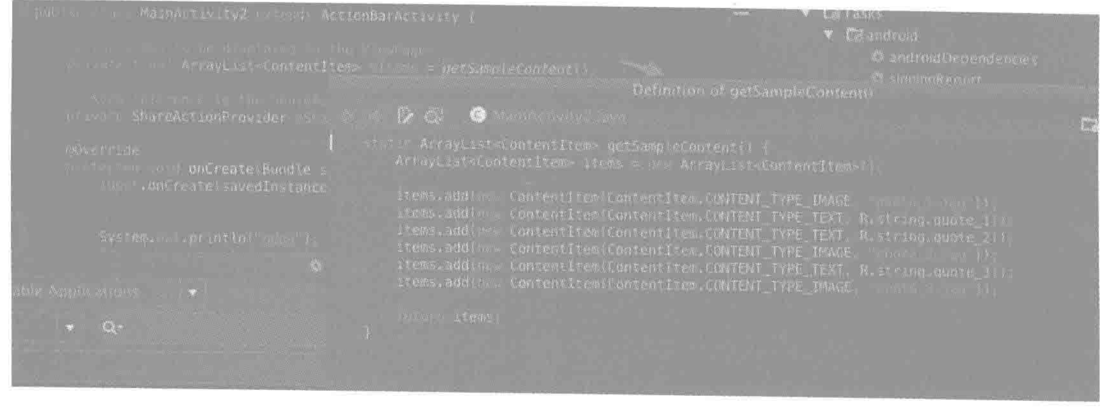


图 6-9

提 示

显示当前选中文件的源码：command + 回车
编辑当前选中文件的源码：command + ↓

如果要查看某个变量、类或方法定义的具体位置，需要按住command键，再单击变量、类或方法。

6.4 查看同胞元素

快速查看同胞元素是指查看跟当前类一样继承自同一个父类的所有元素（类），可以不用离开当前界面就完整查看。

操作步骤：菜单栏→View→Show Siblings。

【实例演示】 查看MainActivity2 的所有同胞元素，如图 6-10 所示。

图 6-10

01 光标放在 MainActivity2 类里面的方法上。

02 选择菜单栏中的 View→Show Siblings，如图 6-11 所示。

图 6-11

6.5 查看文档

如果我们想在不离开当前文件的情况下快速查看某个变量、类或方法的文档，应该如何操作呢？

1. 快速查看文档（见图 6-12）

图 6-12 快速查看文档

菜单栏：View→Quick Documentation

快捷键：fn + F1（macOS）或者Ctrl + Q（Windows/Linux）

如果嫌字体太小，还可以调整字体，如图 6-13 所示。只需要一次设置，以后查看文档就都会显示调整后的效果。

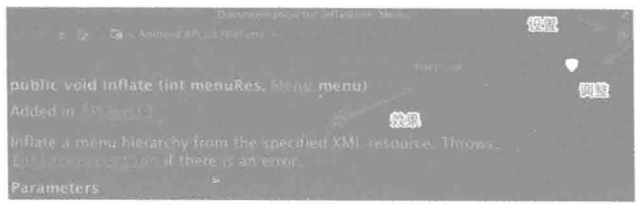


图 6-13

2. 设置鼠标悬停在元素上会显示文档提示

默认情况下，想查看某个变量、类或方法的文档时，需要使用fn+F1 快捷键（macOS）才能查看，其实还有一种更快的方法，即让鼠标悬停在某个变量、类或方法上，直接显示文档提示。

操作步骤：偏好设置→Editor→General→Other→勾选【Show quick documentation on mouse move】，如图 6-14 所示。

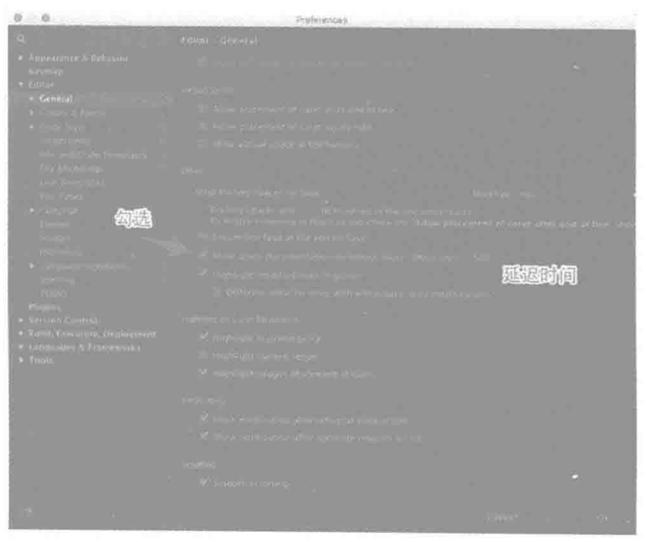


图 6-14

效果如图 6-15 所示。

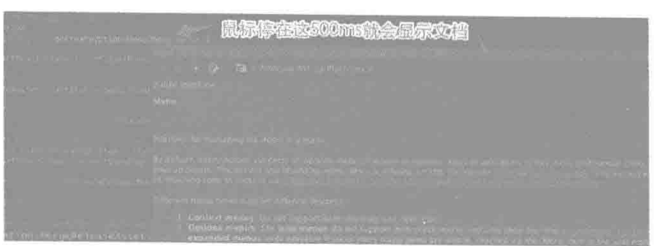


图 6-15

6.6 查看方法的参数信息

如果想快速查看某个方法的参数信息，应该如何操作呢?

1. 快速查看方法的参数信息

菜单栏：View→Parameter Info

快捷键：command + P（macOS）或者Ctrl + P（Windows/Linux）

【实例演示】查看mItems.get（position）方法的参数信息。

将光标放到get上→按快捷键command + P（macOS）→弹出参数信息，如图 6-16 所示。

```
private void setShareIntent(int position) {
    // BEGIN_INCLUDE(update_sap)
    if (mShareActionProvider != null) {
        // Get the currently selected item, and retrieve it's share intent
        ContentItem item = mItems.get(position);
        Intent shareIntent = item.getShareIntent(MainActivity.this);

        // Now update the ShareActionProvider with the new share intent
        mShareActionProvider.setShareIntent(shareIntent);
    }
    // END_INCLUDE(update_sap)
}
```

command + p 快速查看参数信息

int index

图 6-16

2. 设置查看参数信息时显示整个方法的签名信息

Android Studio默认只显示参数的提示信息，非常简单。如果想在查看参数信息时显示整个方法的签名信息（见图 6-17），可以在偏好设置中选择Editor→General→Code Completion→勾选【Show full signatures】，如图 6-18 所示。

图 6-17

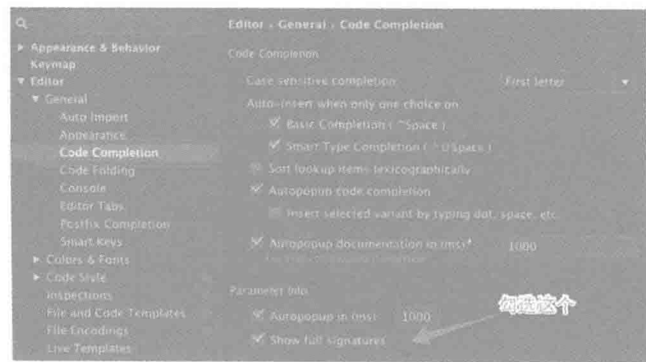


图 6-18

6.7 查看表达式的类型

如果想快速查看表达式的类型（见图 6-19），可以通过菜单栏或快捷键来完成。

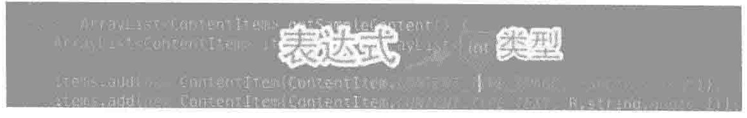


图 6-19

菜单栏： View→Expression Type

快捷键： control + shift + P（macOS）或者Ctrl + Shift + P（Windows/Linux）

【实例演示】

01 将光标放在表达式上→按快捷键 control + shift + p（macOS）→弹出表达式选择列表，如图 6-20 所示。

图 6-20

02 选择一个表达式→显示这个表达式的类型，如图 6-21 所示。

图 6-21

6.8 查看上下文信息

想查看当前代码所在的上下文信息（所在方法名或类名，见图 6-22），可通过菜单栏或快捷键来完成。

图 6-22

菜单栏：View→Context Info

快捷键：control + shift + Q（macOS）或者Alt + Q（Windows/Linux）

6.9 查看源码

菜单栏：View→Jump to Source

快捷键：command + ↓（macOS）或者F4（Windows/Linux）

鼠标：按住command键，再用鼠标单击对应的方法或类

很明显，最快的操作方法是使用快捷键。

6.10 查看最近打开过的文件

菜单栏：View→Recent Files

快捷键：command + E（macOS）或者Ctrl + E（Windows/Linux）

利用菜单栏或快捷键操作后会弹出最近打开过的文件列表，如图 6-23 所示。左边显示的是工具名，右边显示的是文件名。按回车键或单击文件名就可以打开对应的文件或工具。

6.11 查看最近改动过的文件

菜单栏：View→Recently Changed Files

快捷键：shift + command + E（macOS）或者Ctrl + Shift + E（Windows/Linux）

利用菜单栏或快捷键操作后会弹出最近改动文件列表，如图 6-24 所示。左边显示的是工具名，右边显示的是最近改动过的文件名。按回车键或单击文件名就可以打开对应的文件或工具。

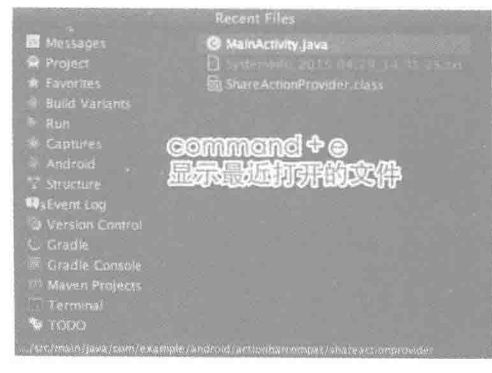


图 6-23

图 6-24

6.12 查看最近的改动

菜单栏：View→Recent Changes （最近的改动）

快捷键（可能会跟系统快捷键冲突）：control + shift + C（macOS）或者Alt + Shift + C（Windows/Linux）

【实例演示】

01 编辑器任意位置→菜单栏：View→Recent Changes→弹出改动列表，如图 6-25 所示。

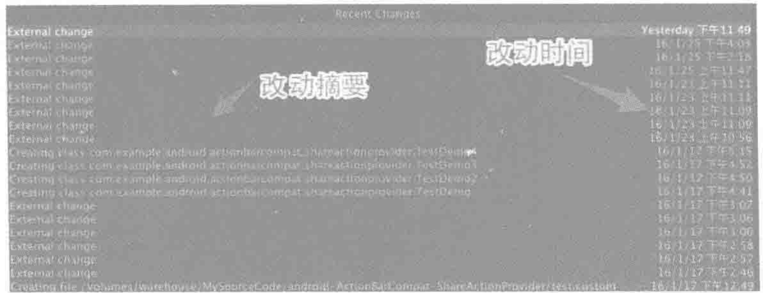


图 6-25

02 选择要查看的改动摘要→按回车键或单击→显示改动的文件，如图 6-26 所示。

图 6-26

03 选中文件→然后就可以还原代码，进行版本对比了，如图 6-27 所示。

图 6-27

还原代码也会被记录在最近改动列表里。

提　示

如果快捷键跟系统快捷键冲突了，怎么解决呢？（见图 6-28）

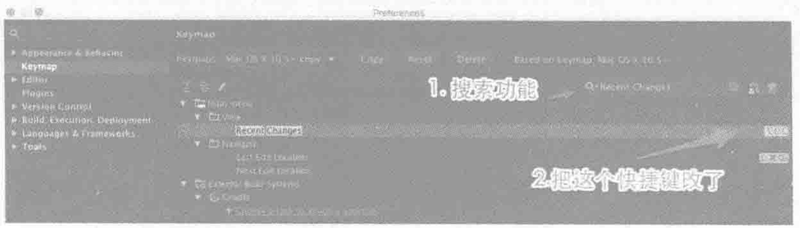


图 6-28

6.13 对比任意文件

菜单栏：View→Compare With

快捷键：command + D（macOS）或者Ctrl + D（Windows/Linux）

此功能可以对比任意的项目文件（包括图像文件）。

【实例演示】对比MainActivity和MainActivity2 两个文件存在的差异。

01 将光标放在 MainActivity 文件上→按快捷键 command + D（macOS）→选择要对比的文件 MainActivity2，如图 6-29 所示。

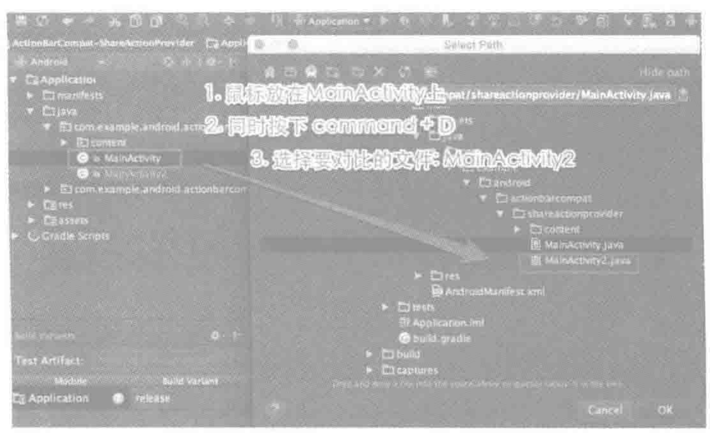


图 6-29

02 Android Studio 以默认的方式打开了两个文件，并高亮显示了存在差异的地方，如图 6-30 所示。

图 6-30

接着就可以对存在差异的地方进行检查了。

上面我们打开的是默认的查看方式（文件显示在两边），还可以切换对比显示方式（文件显示在同一边），如图 6-31 所示。

图 6-31

其他的一些功能如图 6-32 所示。

其他的一些设置如图 6-33 和图 6-34 所示。

图 6-32

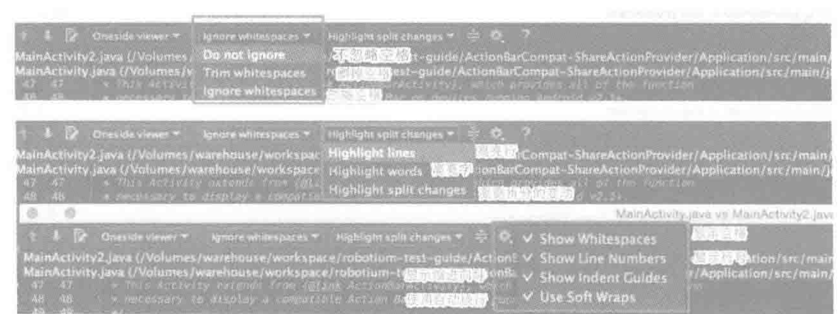


图 6-33

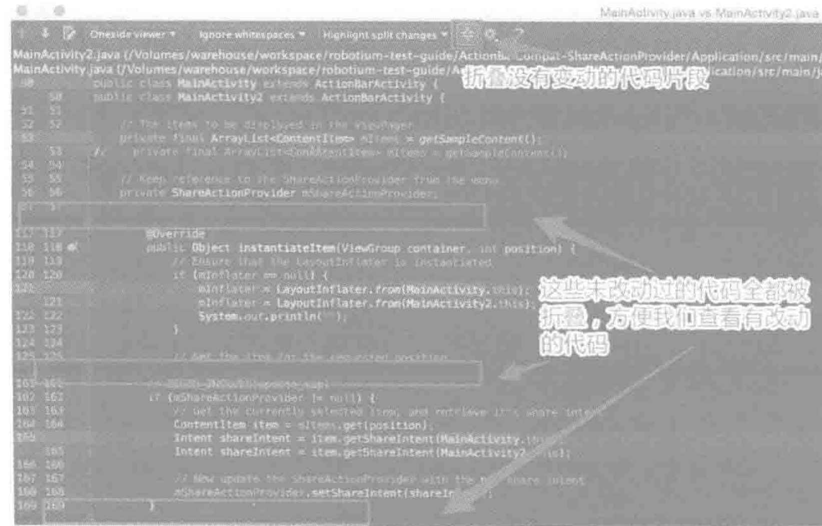


图 6-34

6.14 将选中的文件和正在编辑的文件进行对比

菜单栏：View→Compare File with Editor

【实例演示】

如图 6-35 所示，在编辑器中打开MainActivity2 文件→选中项目中的文件MainActivity→菜单栏：View→Compare File with Editor→弹出文件对比窗口，在这个窗口对比两个文件的差异。

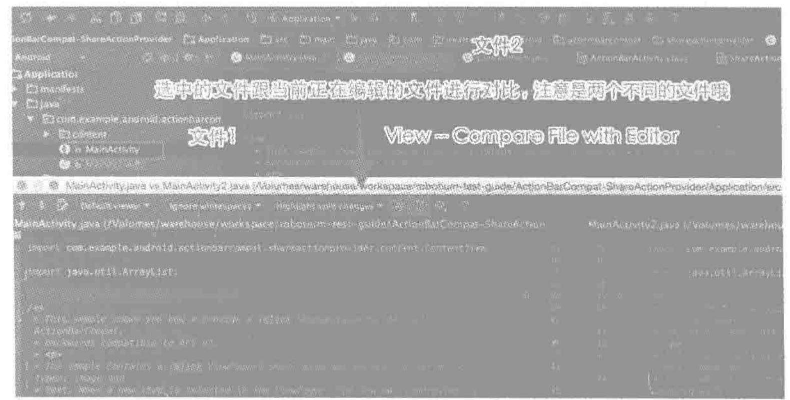


图 6-35

6.15 将选中的文件和剪切板上的内容进行对比

菜单栏：View→Compare with Clipboard （对比剪切板上的内容）

【实例演示】

01 我们剪切了下面的内容。

```
System.out.println ("我是用来测试 Compare with Clipboard 的");
System.out.println ("我是用来测试 Compare with Clipboard 的");
System.out.println ("我是用来测试 Compare with Clipboard 的");
System.out.println ("我是用来测试 Compare with Clipboard 的");
System.out.println ("我是用来测试 Compare with Clipboard 的");
System.out.println ("我是用来测试 Compare with Clipboard 的");
```

02 选择要对比的文件（MainActivity）→菜单栏：View→Compare with Clipboard，如图 6-36 所示。

图 6-36

6.16 切换编辑器配色方案

6.16.1 快速切换编辑器配色方案

通常我们如果想切换编辑器的配色方案是要到偏好设置（Colors & Fonts）去设置的，还有两种非常快捷的方式可以快速切换。

菜单栏：View→Quick Switch Scheme→选择 1.Color Scheme

快捷键：control + `→1（macOS）或者Ctrl + `→1（Windows/Linux）

【实例演示】切换编辑器颜色方案为Sublime Text 2。

01 按快捷键 control + `→1（macOS）→弹出全部的颜色方案，如图 6-37 所示。这里可以看出，我的 Android Studio 有 6 种颜色方案

02 按快捷键 5（macOS）或移动光标选择，显示结果如图 6-38 所示。

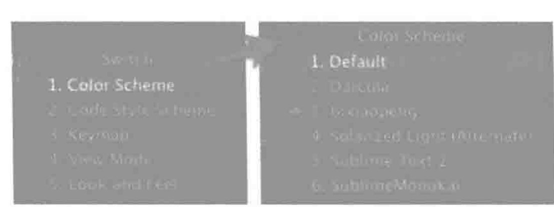


图 6-37

图 6-38

编辑器中文件的颜色显示方案立刻就切换成了Sublime Text 2。

6.16.2 切换编辑器配色方案

通常切换编辑器配色方案是到偏好设置中进行设置的。

操作步骤：偏好设置→Editor→Colors & Fonts→切换Scheme，如图 6-39 所示。

图 6-39

6.17 切换代码风格

1. 快速切换代码风格

通常情况下切换代码风格是在偏好设置（Code Style）中设置的，还有两种非常快捷的方式可以快速切换。

菜单栏：View→Quick Switch Scheme→选择 2.Code Style Scheme

快捷键：control + `→2（macOS）或者Ctrl + `→2（Windows/Linux）

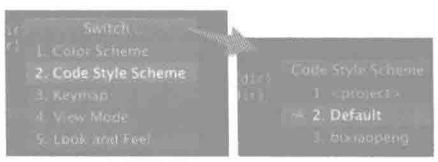


图 6-40

【实例演示】切换编辑器代码风格为bixiaopeng。

01 按快捷键 control + `→2（macOS）→弹出代码风格列表，如图 6-40 所示。

02 按快捷键 3（macOS）或移动光标选择

结果就会切换为bixiaopeng这个代码风格。

2. 在偏好设置中切换代码风格

操作步骤： 偏好设置→Editor→Code Style→切换Scheme，如图 6-41 所示。

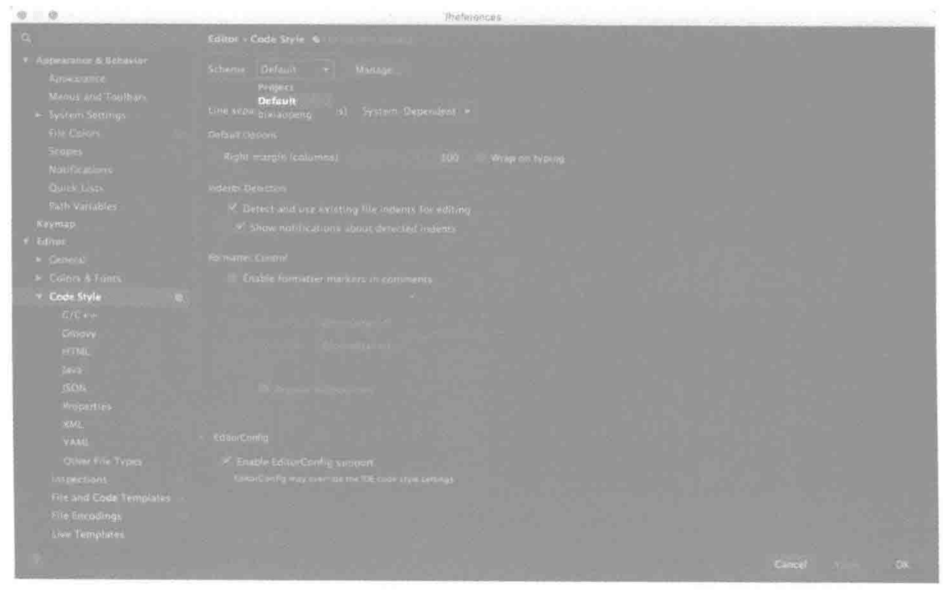


图 6-41

6.18 切换键盘映射

1. 快速切换键盘映射

通常情况下切换键盘映射（快捷键）是在偏好设置（Keymap）中设置的，还可以用非常快捷的方式来实现快速切换。

菜单栏： View→Quick Switch Scheme→选择 3.Keymap

快捷键： control + `→3（macOS）或者Ctrl + `→3（Windows/ Linux）

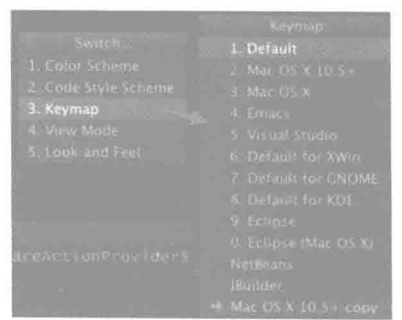


图 6-42

【实例演示】切换键盘快捷键为Eclipse。

01 按快捷键 control + `→3（macOS）→弹出键盘映射列表，如图 6-42 所示。

02 按快捷键 9（macOS）或移动光标选择 9. Eclipse。

2. 在偏好设置中切换键盘映射

操作步骤： 偏好设置→Keymap→切换Keymaps，如图 6-43 所示。

图 6-43

6.19 快速切换视图模式

通常情况下切换视图模式是通过View菜单来完成的，除了进入全屏模式有快捷键以外，进入演示模式和免打扰模式是没有快捷键的，不过还是有快捷方式的。

菜单栏： View→Quick Switch Scheme→选择 4. View Mode

快捷键： control + `→4（macOS）或者Ctrl + `→4（Windows/Linux）

【实例演示】 切换视图模式为演示模式。

01 按快捷键 control + `→4（macOS）→弹出视图模式列表，如图 6-44 所示。

02 按快捷键 1（macOS）或移动光标选择 1. Enter Presentation Mode。

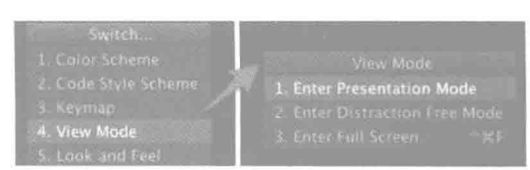


图 6-44

效果如图 6-45 所示，这是本例的全屏显示图，特点就是全屏，字体很大。

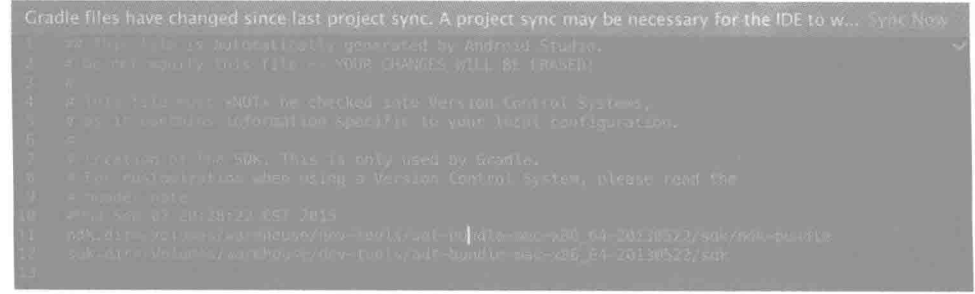


图 6-45

提 示

退出模式操作同进入模式。

6.20 快速切换主题

通常情况下切换主题是在偏好设置（Appearance）中设置的，但也有快捷方式来快速切换。

菜单栏：View→Quick Switch Scheme→选择 5. Look and Feel

快捷键：control + `→5（macOS）或者Ctrl + `→5（Windows/Linux）

【实例演示】切换Android Studio主题为Default。

01 按快捷键 control + `→5（macOS）→弹出主题列表，如图 6-46 所示。

02 按快捷键 1（macOS）或移动光标选择 1. Default。

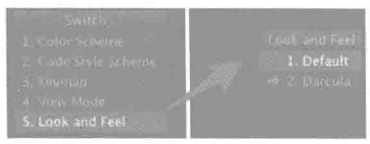


图 6-46

6.21 设置编辑器是否显示空格

默认编辑器界面是不会显示空格的，如果想知道代码中哪里敲入了空格，也可以设置编辑器显示空格。

1. 设置当前编辑器是否显示空格

只对当前打开的文件进行临时设置，可以通过**菜单栏**：View→Active Editor→Show Whitespaces完成，效果如图 6-47 所示。

2. 设置整个编辑器显示空格

操作步骤：偏好设置→Editor→General→Appearance→勾选【Show whitespaces】（见图 6-48）。默认会勾选Leading、Inner、Trailing，也就是显示一行当中的开头、中间和结尾的空格，只想显示某一种时可以自定义选择。

图 6-47

图 6-48

效果如图 6-49 所示，一个点表示一个空格。

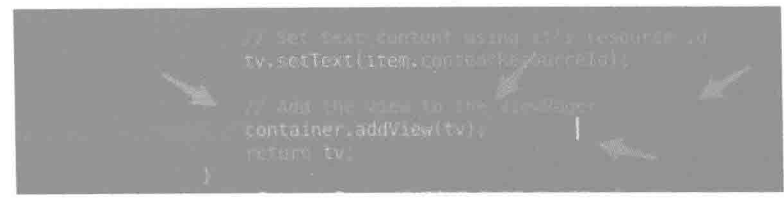


图 6-49

6.22　设置编辑器是否显示行号

默认编辑器界面是不显示行号的，如果习惯于让编辑器显示行号，就可以设置编辑器显示行号。

1. 设置当前编辑器显示行号

如果只想让当前打开的文件临时显示行号（见图 6-50），可以如下操作。

菜单栏： View→Active Editor→Show line number

快捷键： 右击编辑器窗口左边栏→Show Line Numbers（见图 6-51）。

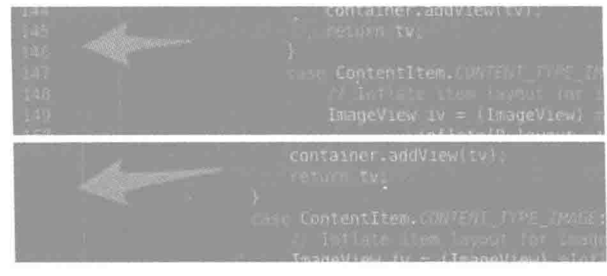


图 6-50

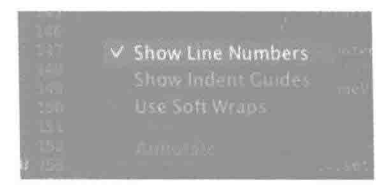


图 6-51

2. 设置整个编辑器显示行号

操作步骤： 偏好设置→Editor→General→Appearance→勾选【Show line numbers】，如图 6-52 所示。

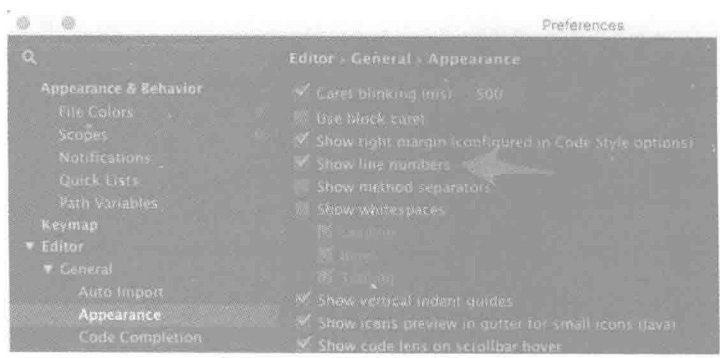


图 6-52

6.23 设置编辑器是否显示缩进向导

显示缩进向导（见图 6-53）以后能够很清晰地看到文件中的缩进关系，从而可以很好地控制每一行的缩进。缩进向导默认情况下是显示的，当然也可以快速进行设置。

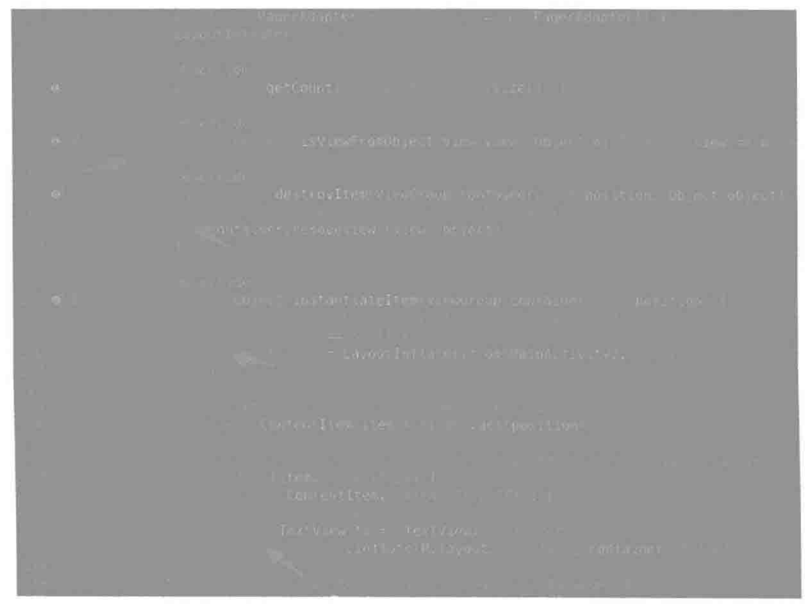


图 6-53

1. 设置当前编辑器显示缩进向导

菜单栏：View→Active Editor→Show Indent Guides

快捷键：右击编辑器窗口左边栏→Show Indent Guides（见图 6-54）。

图 6-54

2. 设置整个编辑器显示缩进向导

操作步骤：偏好设置→Editor→General→Appearance→【Show vertical indent guides】，如图 6-55 所示。

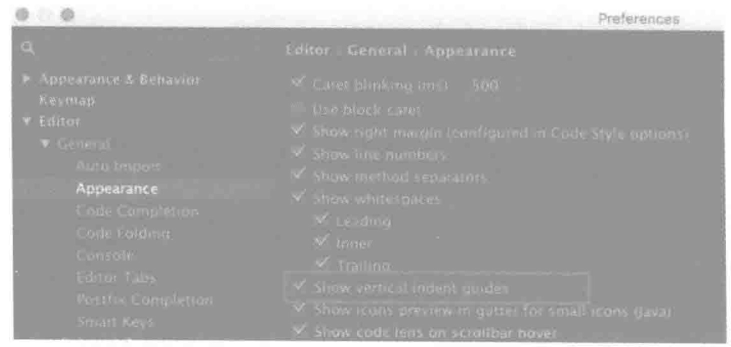


图 6-55

6.24 设置编辑器是否使用自动换行

当一行的内容长得超过编辑器的宽度时，想查看这一行最后边的内容非常麻烦（见图6-56），这时自动换行的好处就来了（见图6-57）。

图 6-56

图 6-57

1. 设置当前编辑器使用自动换行

菜单栏：View→Active Editor→Use Soft Wraps

快捷键：右击编辑器窗口左边栏→Use Soft Wraps（见图6-58）。

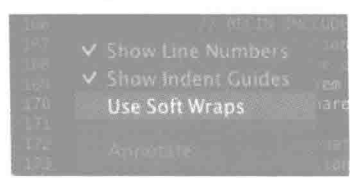


图 6-58

2. 设置整个编辑器使用自动换行

操作步骤：偏好设置→Editor→单击General→查看 Soft Wraps→勾选 【Use soft wraps in editor】，如图 6-59 所示。

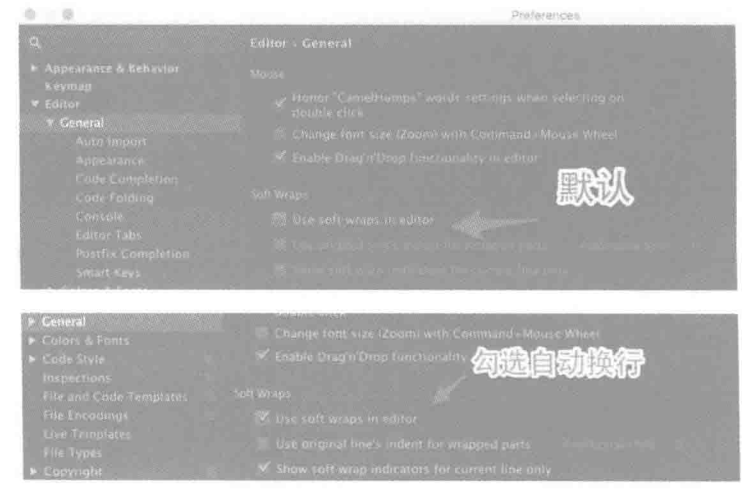


图 6-59

（1）让换行符一直显示

默认【show soft wrap indicators for current line only】会被勾选，意思是仅在你正在编辑的当前行显示换行符，如果离开了当前行，换行符就不显示了。如果想显示所有的换行符，就不要勾选【show soft wrap indicators for current line only】，效果如图 6-60 所示。

图 6-60

（2）设置换行后的缩进字符

默认换行后第二行的缩进是从第 0 个字符开始的（见图 6-61），如果想自定义换行后第二行的缩进，可以勾选【Use original line's indent for wrapped parts】→设置 Additional shift（这里的数字就是换行后第二行缩进的次数）。

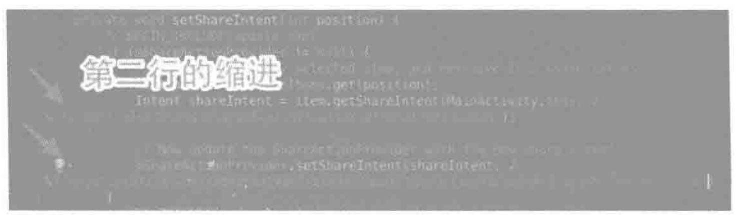


图 6-61

这里设置换行后第二行缩进从第 20 个字符开始（见图 6-62），效果如图 6-63 所示。

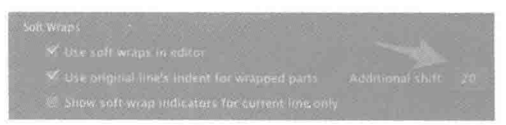


图 6-62

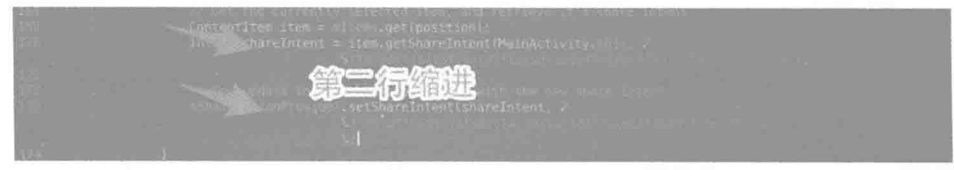


图 6-63

6.25 设置编辑器是否弹出导入提示

Android Studio默认是会显示导入提示的，如图 6-64 所示。

图 6-64

当我们输入的类的声明没有被导入时，会弹出一个导入提示，提示上面还显示了导入的快捷键，让我们可以快速导入。这是一个非常方便的功能，当然所有的功能都支持自定义，毕竟每个人的喜好和习惯都不尽相同。

1. 设置当前编辑器弹出导入提示

操作步骤：菜单栏→View→Active Editor→Show Import Popups

2. 设置整个编辑器弹出导入提示

操作步骤：偏好设置→Editor→General→Auto Import→Show import popup，如图 6-65 所示。

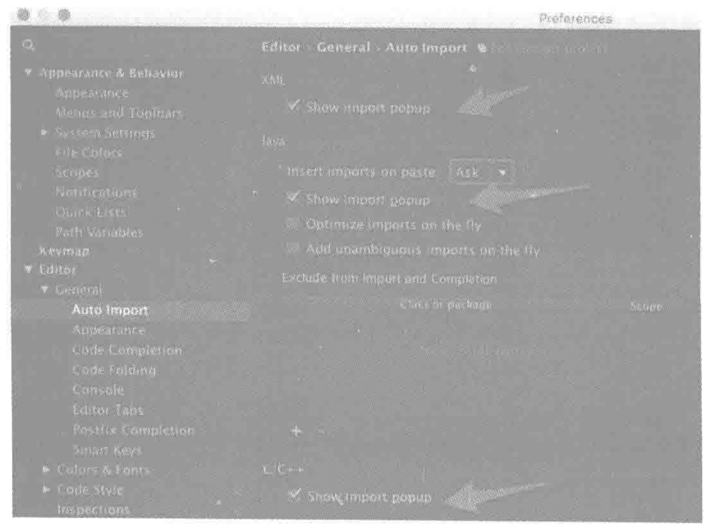


图 6-65

这里同时提供了XML、Java、C/C++是否弹出导入提示的开关。

6.26 使用演示模式

顾名思义，当你想给别人演示代码时就会用到演示模式。演示模式的特点是全屏，开启免打扰模式，工具栏、状态栏等全部被隐藏，同时代码字体会被放大，这样大家就会更专注于代码。

1. 进入演示模式

操作步骤：菜单栏→View→Enter Presentation Mode，效果如图 6-66 所示。

图 6-66

2. 设置演示模式的字体大小

操作步骤：偏好设置→Appearance & Behavior→Appearance→Presentation Mode，如图 6-67 所示。

3. 退出演示模式

操作步骤：菜单栏→View→Exit Presentation Mode。

图 6-67

6.27 使用免打扰模式

当我们只想安心地写代码，不想界面上有其他多余的东西干扰时，可以使用免打扰模式（见图 6-68）。

图 6-68

在进入免打扰模式以后，整个界面什么都没有了，既看不到菜单栏，也看不到工具栏。如果想使用一些功能，需要用快捷键或者右击鼠标。

1. 进入免打扰模式

操作步骤：菜单栏→View→Enter Distraction Free Mode。

2. 退出免打扰模式

操作步骤：菜单栏→View→Exit Distraction Free Mode。

6.28 使用全屏模式

演示模式、免打扰模式和全屏模式是可以组合使用的，当进入演示模式时默认会进入免打扰模式和全屏模式，如果你不想进入免打扰模式和全屏模式可以单独退出。所以如何组合使用这三种模式，就看你的习惯和使用场景了。

1. 进入全屏模式

菜单栏：View→Enter Full Screen
快捷键：control + command + F（macOS）

2. 退出全屏模式

菜单栏：View→Exit Full Screen

第 7 章 导 航

判断一个IDE是否好用，导航功能起着至关重要的作用。当我们阅读代码或搜索文件的时候一定会用到导航功能。Android Studio 的导航功能可以在目录、文件、方法和类之间快速切换，还可以查看文件、类、方法的层次结构。如果能熟练使用这些功能的快捷键，将会大大提高工作效率。

本章将向大家介绍如何使用Android Studio的导航功能。

本章重要知识点 >>>>>>>>>>>

- 如何搜索文件和方法;
- 如何在代码、方法、历史位置之间快速切换;
- 如何查看文件、类、方法的层次结构。

7.1 搜索并打开类文件

如果想在项目中搜索某个类文件，应该如何操作呢？

菜单栏： Navigate→Class

快捷键： command + O（macOS）或者Ctrl + N（Windows/Linux）

【实例演示】

01 打开输入窗口，如图 7-1 所示。

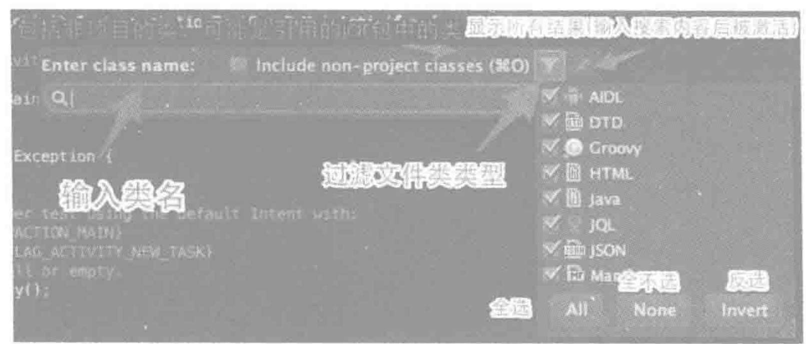


图 7-1

02 输入类文件的关键字→列出模糊匹配结果，如图 7-2 所示。

图 7-2

03 单击匹配结果会打开对应的类文件。

提 示

想跳转到类文件的某一行，应该在类文件后面加上行号。操作如图 7-3 所示。

图 7-3

此功能不支持查找其他文件类型，不支持查找方法。

7.2 搜索并打开某个文件

前面我们讲的只是搜索类文件，其他类型的文件并不能搜索。想搜索其他类型的文件时（不限类型），可以如下操作。

菜单栏：Navigate→File

快捷键：shift + command + O（macOS）或者Ctrl + Shift + N（Windows/Linux）

效果如图 7-4 所示。

图 7-4

此功能的操作跟搜索类文件一样。

7.3 搜索并打开某个文件或方法

前面两种操作仅是搜索文件的，并不支持搜索方法，既想搜索文件又想要搜索方法名时可以如下操作。

菜单栏： Navigate→Symbol

快捷键： option + command + O（macOS）或者Ctrl + Alt + Shift + N（Windows/Linux）

【实例演示】

01 按快捷键 option + command + O（macOS）打开搜索弹窗。

02 输入方法名。

03 按回车键后会打开方法所在文件，并定位到方法所在位置，如图 7-5 所示。

图 7-5

到此为止，我们讲了 3 种搜索文件的方法，按搜索范围排序应该是Symbol > File > Class，且是一种包含的关系（Symbol 包含File ，File包含Class）。

7.4 使用自定义代码块

1. 什么是代码块

简单来说，代码块就是写在{}中的代码，如图 7-6 所示。

我们可以通过快捷键【command +】和【command -】（macOS）来展开和折叠代码块。

2. 什么是自定义代码块

自定义代码块就是在代码片段的开始和结束分别加上标识，有了这个标识以后就可以跟普通的代码块一样展开和折叠了。

图 7-6

另外，自定义代码块还有一个更实用的功能，那就是添加代码片段的描述。添加描述以后，我们就可以很方便地搜索到这段代码，也可以快速地在不同的代码片段之间进行跳转。

```
//<editor-fold desc="这是测试用的">
MenuItem shareItem = menu.findItem(R.id.menu_share);
mShareActionProvider = (ShareActionProvider)
            MenuItemCompat.getActionProvider(shareItem);
//</editor-fold>
```

上面这段代码就被设置成了自定义代码块，是被标签包住的，非常直观。

3. 如何自定义代码块

01 选中要自定义的代码片段→菜单栏：Code→Surround With 或按快捷键 option + command + T（macOS）。

02 在弹窗中选择 B. <editor_folding>Comments，如图 7-7 所示。

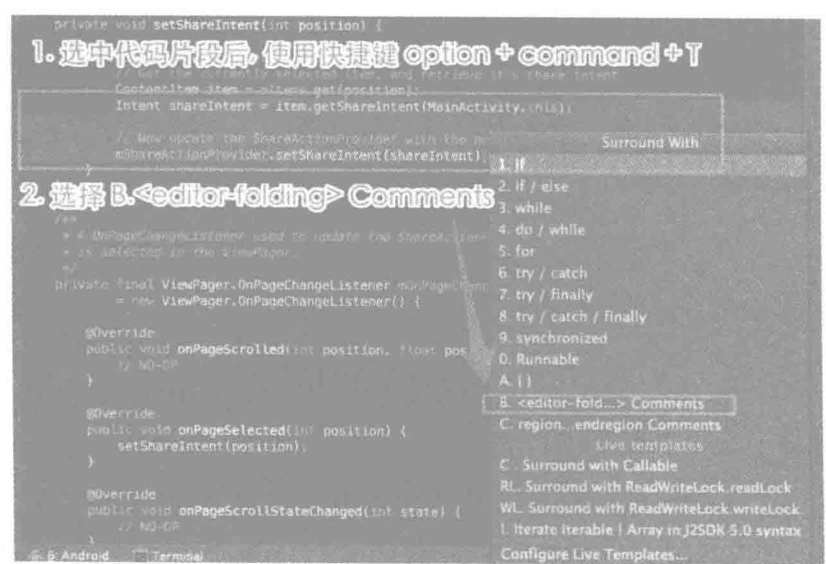


图 7-7

03 我们刚才选择的代码片段开始和结束被加上了标识。

04 修改代码块的描述，des="描述信息"。

05 可以跟普通的代码块一样对这段代码进行展开和折叠，如图 7-8 所示。

图 7-8

4. 在自定义代码块之间选择和跳转

菜单栏：Navigate→Custom Folding

快捷键：option + command + " . "（macOS）或者Ctrl + Alt +" . "（Windows/Linux）

【实例演示】

01 自定义两个代码块：测试自定义代码块 1 和测试自定义代码块 2。

02 按快捷键 option + command + " . "（macOS）→打开选择界面，单击就可以跳转到相应的自定义代码块，如图 7-9 所示。

图 7-9

7.5 快速跳转到某一行代码

菜单栏：Navigate→Line

快捷键：command + L（macOS）或者Ctrl + G（Windows/Linux）

【实例演示】

例 1：光标快速跳转到当前文件的第 123 行。

01 将光标放到当前文件编辑窗口的任意位置。

02 按快捷键 command + L（macOS）。

03 在弹出的 Go to Line 对话框中输入需要跳转的行 123。

04 单击【OK】。

结果：跳转到第 123 行。

例 2：快速跳转到当前文件的 123 行第 1 列。

如果想更精确一点，快速定位到某一行的某一列，可以如下操作。

01和02同例 1。

03 在弹出的 Go to Line 对话框中输入需要跳转的行和列 123:1，如图 7-10 所示。

04 单击【OK】按钮，跳转到当前文件的 123 行第 1 列。

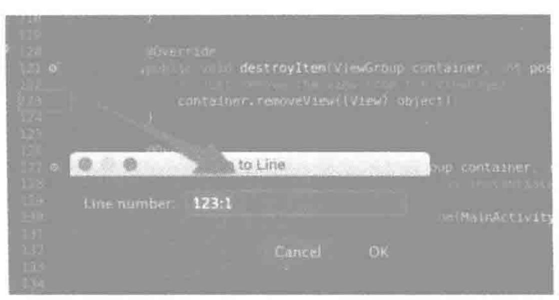


图 7-10

7.6 快速跳转到光标的历史位置

当我们在阅读或编写代码的时候会打开不同的文件，光标也会在不同的文件和不同的位置之间切换，如果想快速跳转到光标的历史位置，可以进行如下操作。

1. 快速跳转到光标上一个历史位置

菜单栏：Navigate→Back

快捷键：command + [（macOS）或者Ctrl + Alt + ←（Windows/Linux）

2. 快速跳转到光标下一个历史位置

菜单栏：Navigate→Forward

快捷键：command + [（macOS）或者Ctrl + Alt +→（Windows/Linux）

【实例演示】

光标的历史位置按时间顺序有位置 1、位置 2、位置 3，如图 7-11 所示。

图 7-11

例 1：光标现在所在的位置是位置 3，想回到位置 2 或位置 1 时，按快捷键command +[（macOS）。

例 2：光标现在所在的位置是位置 1，想回到位置 2 或位置 3 时，按快捷键command +]（macOS）。

7.7 快速跳转到编辑过的历史位置

在写代码的时候经常会在不同的文件之间切换，Android Studio会记录所编辑过的位置。如果想快速切换到编辑过的位置时，应该如何操作呢？例如，我们在A文件里写了一个方法，在B文件里进行调用。写到一半时突然想到A文件里的这个方法可能有BUG，想快速切换回去修复，这时就需要快速跳转到上一个编辑过的位置。

1. 快速跳转到上一个编辑过的位置

菜单栏：Navigate→Last Edit Location
快捷键：shift + command + delete（macOS）或者Ctrl + Shift + Backspace（Windows/Linux）

2. 快速跳转到下一个编辑过的位置

菜单栏：Navigate→Next Edit Location

7.8 标记书签

1. 书签是什么

生活中的书签是为了记录阅读进度而夹在书里的小纸片，在Android Studio中的书签是指对代码的标记，标记我们阅读的位置。给代码加上书签，可以快速地在不同的书签中切换。

2. Android Studio中提供的书签功能（见图 7-12）

图 7-12

- Toggle Bookmark：标记书签。
- Toggle Bookmark with Mnemonic：使用助记符标记书签。
- Show Bookmarks：显示书签。
- Next Bookmark：下一个书签。
- Previous Bookmark：上一个书签。

3. 标记书签操作

菜单栏：Navigate→Bookmarks→Toggle Bookmark
快捷键：fn + F3（macOS）或者F11（Windows/Linux）

【实例演示】

01 把光标定位在想加书签的那一行代码。

02 按快捷键 fn + F3（macOS），这行代码就会打上标签，如图 7-13 所示。

```
120    @Override
121    public void destroyItem(ViewGroup container, int position, Object object
126    @Override
127    public Object instantiateItem(ViewGroup container, int position) {
130            mInflater = LayoutInflater.from(MainActivity.this);
131            System.out.println( );
```

第1步：光标定位在想加书签的那一行
第2步：Navigation→Bookmarks→Toggle Bookmark

图 7-13

取消书签的步骤同标记书签。

7.9 使用助记符标记书签

顾名思义，助记符是帮助我们记忆的符号。Android Studio中支持使用助记符来标记书签。

菜单栏：Navigate→Bookmarks→Toggle Bookmark with Mnemonic

快捷键：fn + option + F3（macOS）或者Ctrl + F11（Windows/Linux）

【实例演示】

01 把光标定位在想加书签的那一行代码。

02 按快捷键 fn + Option + F3（macOS）。

03 在弹出的 Bookmark Mnemonic 弹窗中选择助记符。助记符为一个数字和字母。

04 在编辑器的左边栏就会显示助记符书签，如图 7-14 所示。

图 7-14

把光标放到助记符书签上，会提示我们这是一个书签。

7.10 管理书签

书签管理包括对书签进行搜索、排序、修改描述、删除、查看等操作。

7.10.1 在书签管理界面管理书签

1. 打开书签管理界面（见图 7-15）

图 7-15

菜单栏：Navigate→Bookmarks→Show Bookmarks
快捷键：fn + command + F3（macOS）或者Shift + F11（Windows/Linux）
在书签管理界面可以对项目中所有的书签进行管理。

2. 修改书签描述

为了更好地管理书签，可以给书签加上描述信息（见图 7-16 和图 7-17）。

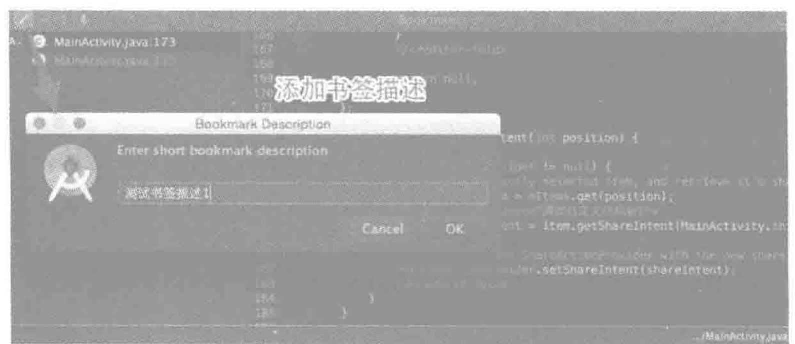


图 7-16

图 7-17

操作步骤：在书签管理界面单击左上角的编辑按钮，或者按快捷键command +回车（macOS）→在弹窗中输入描述。

7.10.2 在收藏夹中管理书签

在收藏夹也可以对书签进行管理。

操作步骤：工具栏→Favorites→Bookmarks，如图 7-18 所示。

图 7-18

7.11 快速跳转到导航栏

菜单栏：Navigate→Jump to Navigation Bar

快捷键：command + ↑（macOS）或者Alt + Home（Windows/Linux）

【实例演示】

如图 7-19 所示，跳转到导航栏以后就可以通过键盘或鼠标来进行文件切换了。

例如，光标定位在MainActivity上面时，在键盘上按向下箭头，就会把MainActivity同级的文件列出来，方便快速切换，如图 7-20 所示。

图 7-19

图 7-20

7.12 快速跳转到声明

在写代码或阅读代码时，想快速查看一个引用首次被声明的位置，可以按照如下方法操作。

菜单栏：Navigate→Declaration

快捷键：command + B（macOS）或者Ctrl + B（Windows/Linux）

鼠标（macOS）：按住command键→单击类、变量或方法

【实例演示】查看onCreate方法的声明（见图 7-21）。

01 将光标放在 onCreate 方法上。

02 按快捷键 command + B（macOS）。

图 7-21

7.13 快速跳转到实现

查看某个接口在哪里被实现可以作用如下方法。

菜单栏：Navigate→Implementation

快捷键：option + command + B（macOS）或者Ctrl + Alt + B（Windows/Linux）

【实例演示】查看接口TmpDelegateProvider的实现。

01 将光标放在接口名上。

02 按快捷键 option + command + B（macOS），跳转到接口 TmpDelegateProvider 的实现位置，如图 7-22 所示。

图 7-22

7.14 快速跳转到类型声明

快速查看某个引用的类型声明时可以使用如下方法。

菜单栏：Navigate→Type Declaration

快捷键： shift + command + B（macOS）或者Ctrl + Shift + B（Windows/Linux）
鼠标（macOS）： 按住command+Shift键→单击类、变量或方法

【实例演示】 查看引用类getMenuInflater的类型声明（见图 7-23）。

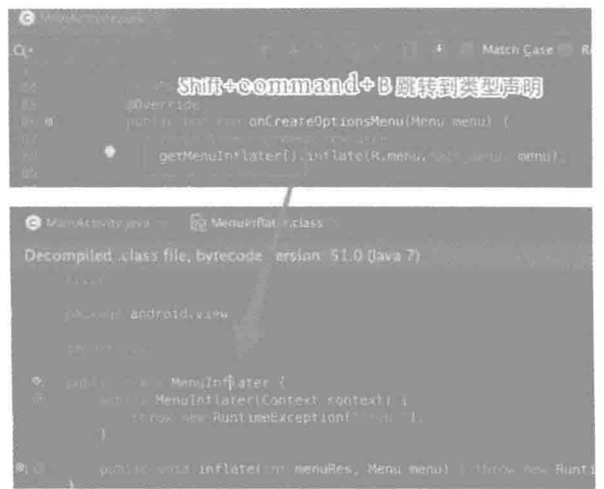


图 7-23

01 将光标放在 getMenuInflater 上。
02 按快捷键 shift + command + B（macOS）。
03 查看类型声明。

7.15 快速跳转到父类

如果某个子类中覆写了父类的一个方法，我们想查看这个方法在父里类的实现，应该如何操作呢？

前提条件： 光标定位在方法体内或类名上。
菜单栏： Navigate→Super Method
快捷键： command + U（macOS）或者Ctrl + U（Windows/Linux）

【实例演示】

例 1： 快速跳转到MainActivity的父类。

01 将光标放在 MainActivity 这一行的任意位置。
02 按快捷键 command + U(macOS)，然后光标跳转到 MainActivity 的父类 ActionBarActivity，如图 7-24 所示。

例 2： 快速跳转到onCreate方法父类中实现的地方。

01 将光标定位在 MainActivity 类中的 onCreate 方法体内。
02 按快捷键 command + U（macOS），光标跳转到 MainActivity 的父类 ActionBarActivity 中 onCreate 方法实现的地方，如图 7-25 所示。

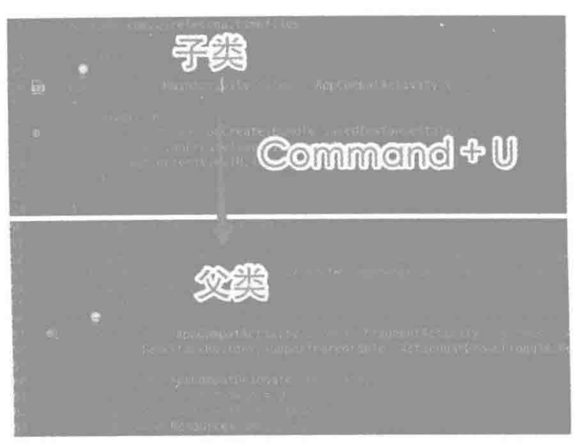


图 7-24

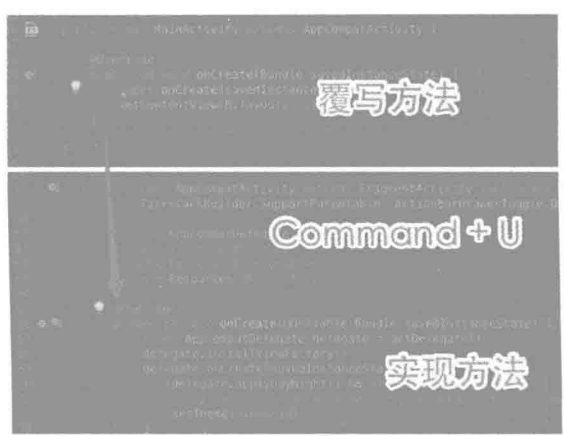


图 7-25

7.16 类和测试类之间快速跳转

Android Studio中类和测试类之间可以快速跳转，如果没有测试类，可以快速创建。现在假设类文件MainActivity.java没有测试类。

1. 快速跳转到测试类

前提条件：光标定位在类文件编辑界面。

菜单栏：Navigate→Test

快捷键：shift + command + T（macOS）或者Ctrl + Shift + T（Windows/Linux）

利用菜单栏或快捷键操作后会弹出【Choose Test for MainActivity】对话框，如图 7-26 所示。

图 7-26

如果有测试类，会让我们选择跳转到哪一个测试类；如果没有测试类，可以创建一个。

2. 创建测试类

01 单击【Create New Test...】，弹出创建界面，如图 7-27 所示。测试库默认是JUnit4，可以在下拉列表中选择其他库。

02 配置测试类。选中需要生成的测试方法，其他配置使用默认值。然后单击【OK】按钮，弹出【Choose Destination Directory】窗口，如图 7-28 所示。

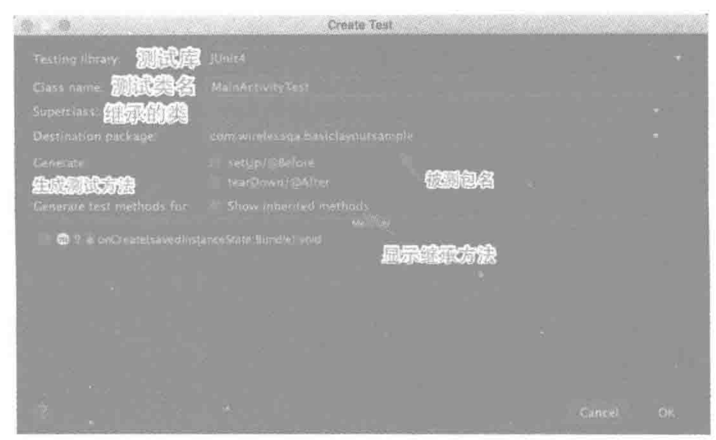


图 7-27

图 7-28

03 选择目标目录。如果我们要创建的是 Android 单元测试，选择的路径中包含 androidTest。如果我们要创建的是本地单元测试，选择的路径中包含 test。

本例我们创建的是Android单元测试，所以选择上面一个，单击【OK】按钮之后成功创建测试类。

3. 从测试类跳转到被测类

前提条件：打开刚创建好的测试类，将光标放在编辑界面。

操作步骤：在菜单栏选择Navigate→Test，或者按快捷键shift + command + T（macOS）或者Ctrl + Shift + T（Windows/Linux），被测类MainActivity就会立刻被打开。

4. 从类跳转到测试类

前提条件：光标定位在类的编辑界面。

操作步骤：在菜单栏中选择Navigate→Test，或者按快捷键shift + command + T（macOS）或者Ctrl + Shift + T（Windows/Linux），然后会弹出测试类选择界面，如图 7-29 所示。

单击测试类或按回车键就会跳转到测试类。

5. 直接运行选中的测试类

在测试类选择界面中选中某个测试类，使用快捷键control + shift + R（macOS），直接运行选中的测试类，如图 7-30 所示。

图 7-29

图 7-30

图 7-30 中显示了直接运行测试类的快捷键提示。

6. 搜索测试类

如果在测试类选择界面列出了很多测试类，我们还可以进行搜索，以便快速筛选出所需要的测试类。

操作步骤： 按快捷键shift + command + T（macOS）或者Ctrl + Shift + T（Windows/Linux），在弹出的测试类选择界面直接输入搜索关键字，比如上面的例子就可以直接输入M，筛选出所有以M开头的测试类，如图 7-31所示。

图 7-31

7.17 查看相关联的文件

菜单栏： Navigate→Related Symbol

快捷键： control + command + ↑（macOS）或者Ctrl + Alt + Home（Windows/Linux）

【实例演示】查看与MainActivity.java相关联的文件。

01 打开 MainActivity.java 文件。

02 按快捷键 control + command + ↑（macOS）。

03 在相关文件选择对话框中，会显示出 MainActivity 相关的布局文件和测试文件，如图 7-32 所示。

04 单击打开文件进行查看。

图 7-32

提 示

除了上面讲过的操作方式，还可以通过单击左边栏的标识快速查看和切换。

- 在 Activity 中查看相关联的布局文件，如图 7-33 所示。
- 在布局文件中查看类文件，如图 7-34 所示。

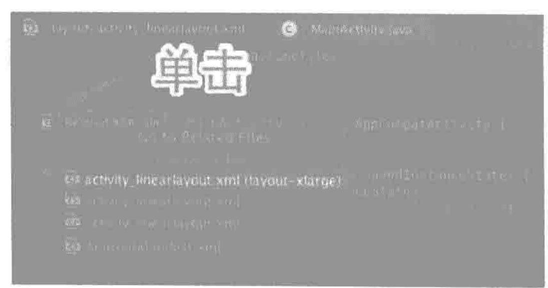


图 7-33

图 7-34

7.18 查看文件结构

我们可以快速调出当前文件的结构（见图 7-35），并通过模糊匹配快速跳转至指定的方法。

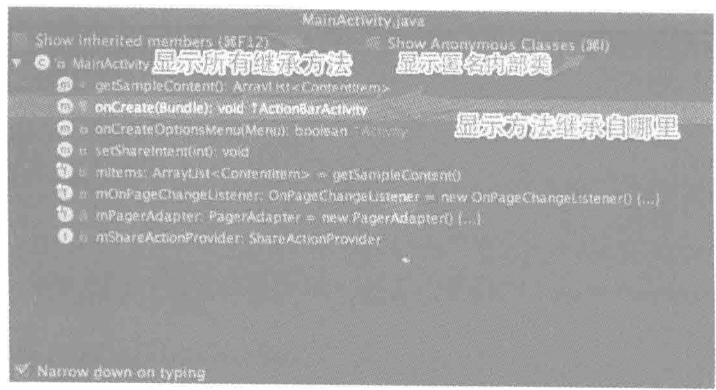


图 7-35 查看文件结构界面

菜单栏：Navigate→File Structure

快捷键：fn + command + F12（macOS）或者Ctrl + F12（Windows/Linux）

在弹出的文件结构界面输入要查找的方法，系统会进行模糊匹配，如图 7-36 所示。

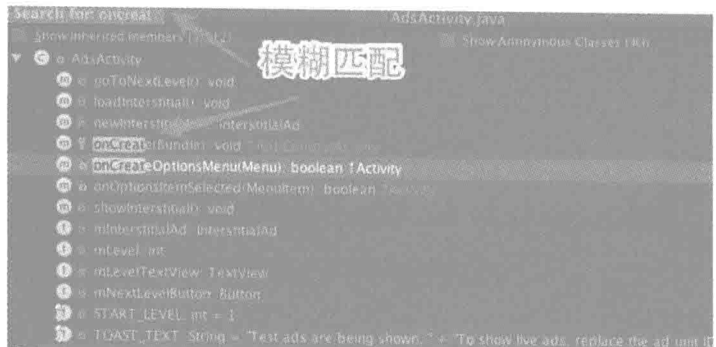


图 7-36

1. 显示匿名内部类

要在文件结构中显示匿名内部类，需要勾选【Show Anonymous Classes】，如图 7-37 所示。

2. 显示所有继承的方法

要在文件结构中显示匿名内部类，需要勾选【Show inherited members】，如图 7-38 所示。

图 7-37

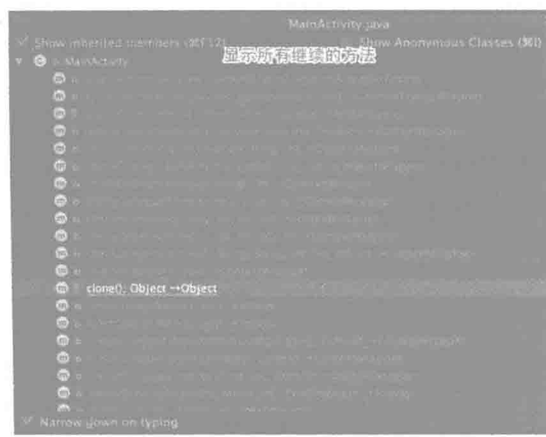


图 7-38

3. 输入时缩小

输入时缩小是指在输入搜索关键字时搜索结果会自动过滤，不匹配的会被过滤掉。

- 勾选【Narrow down on tying】，如图 7-39 所示。
- 不勾选【Narrow down on tying】，在输入搜索关键字时，不匹配的项是不会被过滤掉的，如图 7-40 所示。

图 7-39

图 7-40

7.19 查看类的层次结构图

一个子类继承了一个父类，这个父类还可以继承自它的父类，这样就形式了类的层次结构。如果想查看某个类的层次结构图，可以按照如下方法操作：

菜单栏：Navigate→Type Hierarchy

快捷键：control + H（macOS）或者Ctrl + H（Windows/Linux）

【实例演示】查看MainActivity的层次结构图，如图 7-41 所示。

另外，还可以切换类结构的显示方式、排序、筛选范围等，如图 7-42 所示。

图 7-41

图 7-42

7.20 查看方法类型的层次结构

如果我们想查看某个方法在哪个类中定义，在哪些类中实现，可以按照如下方法操作。

菜单栏：Navigate→Type Method

快捷键：shift + command + H（macOS）或者Ctrl + Shift + H（Windows/Linux）

【实例演示】查看所有定义和实现了onCreate（）方法的类的层次结构图。

01 将光标定位在 onCreate（）方法上。

02 按快捷键 shift + command + H（macOS）。

03 显示出类的层次结构图，如图 7-43 所示。

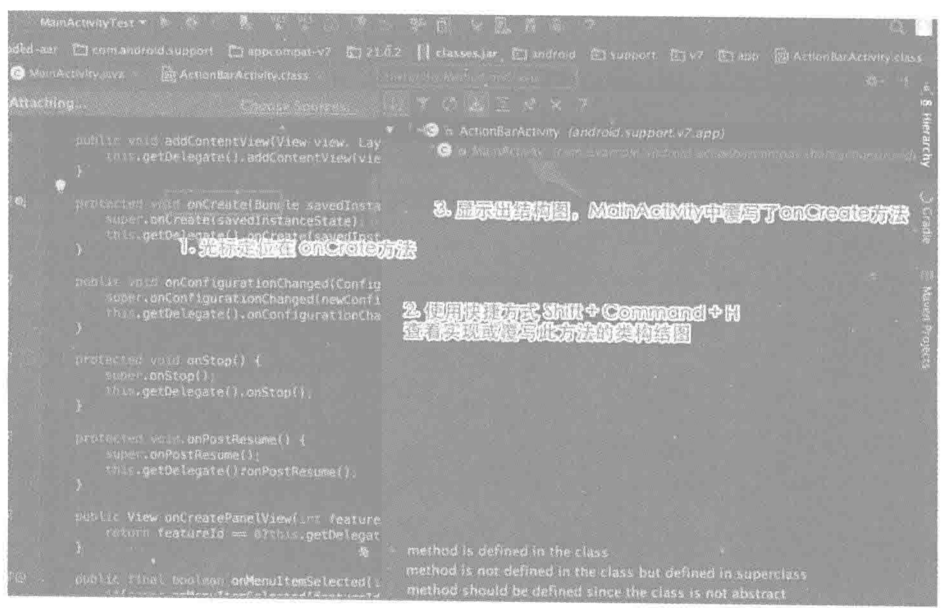


图 7-43

注意层次结构图中类前面的icon标识（见图 7-44）。

method is defined in the class
method is not defined in the class but defined in superclass
method should be defined since the class is not abstract

图 7-44

- 加号表示方法是在这个类中定义的。

- 减号表示方法不是在这个类中定义的，是在父类中定义的。
- 叹号表示不是抽象类，方法应该被定义。

7.21　查看方法调用层次结构

当我们想查看当前方法的用法时，可以使用快捷键fn + option + F7（macOS）。当我们想查看方法调用层次结构的时候，应该如何操作呢？

菜单栏： Navigate→Call Hierarchy

快捷键： control + option + H（macOS）或者Ctrl + Alt + H（Windows/Linux）

【实例演示】 查看onCreate方法的调用层次结构。

01 将光标放在方法上。

02 按快捷键 control + option + H（macOS）。

03 显示方法调用的层次结构图。

如图 7-45 所示，最上面的类.方法名是指方法在这里定义，下面的类.方法名表示方法在这里被调用。单击后光标会定义到方法定义或被调用的位置。

图 7-45

7.22　快速跳转到错误代码的位置

我们编写代码时如果出现错误，就会在编辑器右边栏高亮显示红色的条标，如果有多处错误就会显示多个条标。如果想快捷查看错误代码，可以通过单击右边栏的条标进行快速跳转。如果想挨个查看错误，应该如何操作呢？

1. 跳转到下一个错误位置

菜单栏： Navigate→Next Highlighted Error

快捷键： fn + F2（macOS）或者F2（Windows/Linux）

2. 跳转到上一个错误位置（见图 7-46）

菜单栏： Navigate→Previous Highlighted Error

快捷键： fn + Shift + F2（macOS）或者Shift + F2（Windows/Linux）

图 7-46 跳转到上一个错误位置

7.23 在方法间前后跳转

当我们查看一个类文件时，如果需要一个方法一个方法挨个查看，应该如何操作呢？

1. 跳转到下一个方法

菜单栏：Navigate→Next Method
快捷键：control + ↑（macOS）或者Alt + ↑（Windows/Linux）
（此快捷键可能与系统快捷键冲突，请自行修改）

2. 跳转到上一个方法

菜单栏：Navigate→Previous Method（上一个方法）
快捷键：control + ↓（macOS）或者Alt + ↓（Windows/Linux）
（此快捷键可能与系统快捷键冲突，请自行修改）

7.24 使用翻页功能

当正在阅读的代码很长时，除了用滚动鼠标的方式来查看，还可以使用翻页功能来查看。Android Studio中的一页是指当前编辑器窗口显示的大小。如图 7-47 所示，当前编辑器窗口显示的大小就是一页，这一页是从第 1 行到 72 行，下一页将从 73 行开始。

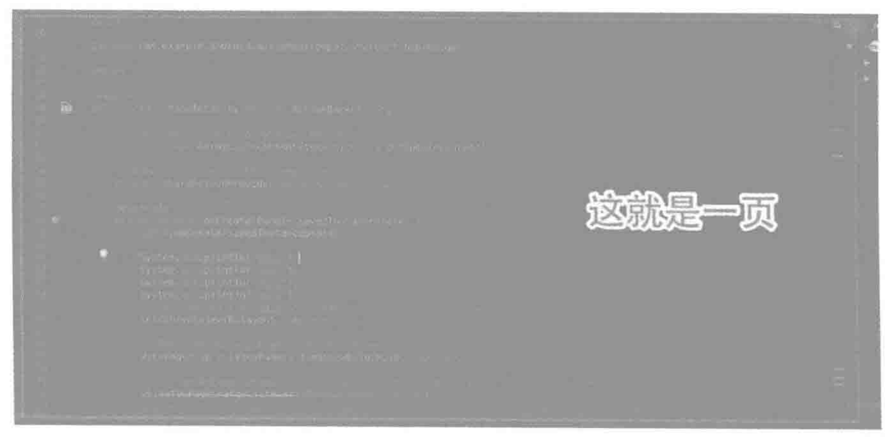


图 7-47

- 向上翻页：快捷键为 fn + ↑（macOS）或者 Page Up（Windows/Linux）。
- 向下翻页：快捷键为 fn + ↓（macOS）或者 Page Down（Windows/Linux）。

7.25 选择当前文件在哪里显示

当选中一个文件时，此功能列出所有可以显示该文件的地方以供选择，如图 7-48 所示。

菜单栏：Navigate→Select in

快捷键：fn + option + F1（macOS）或者Alt + F1（Windows/Linux）

图 7-48

7.26 光标快速跳转到编辑器

如果编辑器处于活跃状态（文件处于打开状态），当焦点不在编辑器时，按Esc键可以让焦点从任何其他工具窗口返回到活跃的编辑器。

【实例演示】

01 光标在工具窗口中，如图 7-49 所示。

图 7-49

02 按 Esc 键，光标就跳转到编辑器界面，如图 7-50 所示。

图 7-50

提 示

按 Esc 键仅光标跳转到编辑器，如果想关闭工具窗口后再跳转到编辑器，可以按 Shift + Esc 键。

7.27 光标快速跳转到页首/页尾

如果我们阅读时想快速跳转到页首或页尾，应该如何操作呢？

- 光标快速跳转到页首：快捷键为 fn + command + ←（macOS）或者 Ctrl + Home（Windows/ Linux）。
- 光标快速跳转到页尾：快捷键为 fn + command +→（macOS）或者 Ctrl + End（Windows/ Linux）。

第 8 章　编　码

Android Studio中提供的代码生成、活动模板、自动补全和格式化代码的功能能够帮助我们极大地提高编码效率。使用Android Studio可以快速覆写或实现方法、快速插入常用的代码模板、在输入代码时自动补全以及一键优化代码格式。

本章将向大家介绍如何使用Android Studio提高编码效率。

本章重要知识点 >>>>>>>>>>

- 如何快速生成和移除代码;
- 如何使用代码自动补全;
- 如何设置并使用代码模板;
- 如何设置并使用代码格式化。

8.1　覆写或实现方法

需要覆写父类的方法或实现接口方法的时候，可以按照如下方法操作。

菜单栏：Code→Override Method

快捷键： control + O（macOS）或者Ctrl + O（Windows/Linux）

【实例演示】

01 在子类中使用快捷键 control + O（macOS）→弹出方法选择对话框，如图 8-1 所示。

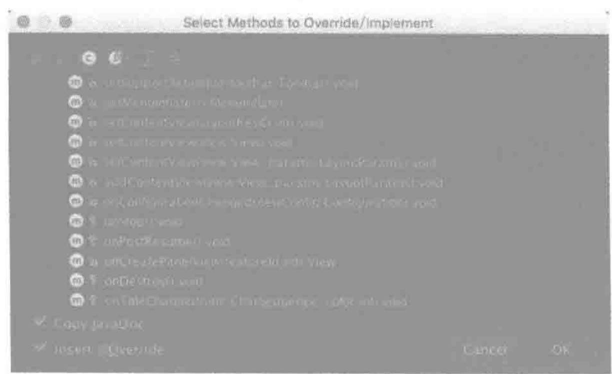


图 8-1

02 注释和注解。默认会勾选【Copy JavaDoc】和【Insert @Override】，意思就是覆写或实现选择的方法时，会同时复制父类中的 JavaDoc 并同时插入@Overrid 这样一个注解，如果不想要这些可以取掉勾选。

03 选择方法来实现。我们可以选择 1 个或多个方法来实现（见图 8-2），同时还可以搜索方法（见图 8-3）。

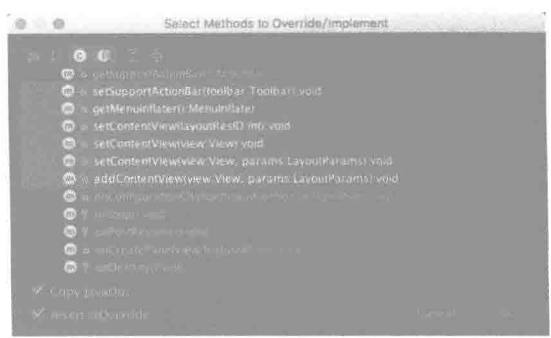


图 8-2

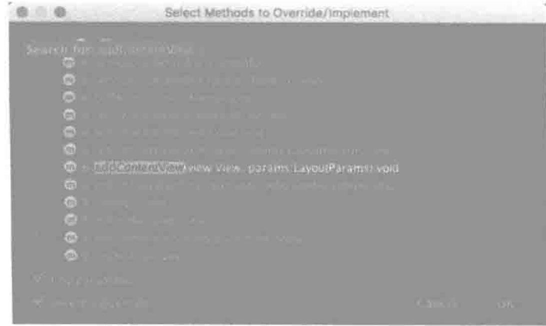


图 8-3

04 按回车键即可覆写方法，如图 8-4 所示。

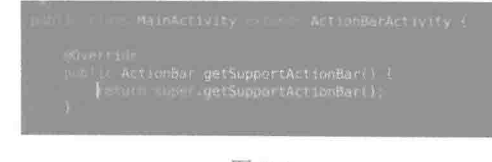


图 8-4

在方法选择界面中，排序、过滤和显示的几个按钮功能如下。

- 按重写方法的类的比重排序（见图 8-5）。
- 按字母顺序排序（见图 8-6）。

图 8-5

图 8-6

- 按类分组显示（见图 8-7）。
- 显示要实现的方法（见图 8-8）。如果不选择此项，只会显示覆写的方法。

图 8-7

图 8-8

- 全部展开和全部收起（见图 8-9）。

图 8-9

8.2　实现接口方法

Override Method可以实现接口方法，也可以覆写父类的方法，但Implement Methods只能实现接口方法。只需要实现接口方法时，可以按照如下方法操作。

菜单栏： Code→Implement Methods

快捷键： control + L（macOS）或者Ctrl + I（Windows/Linux）

【实例演示】 实现接口方法。

01 在实现类中执行快捷键 Control + L（macOS）→弹出选择方法窗口，如图 8-10 所示。

02 至 **04** 同第 8.1 节，选择实现的方法。

按回车键后显示结果，如图 8-11 所示。

图 8-10

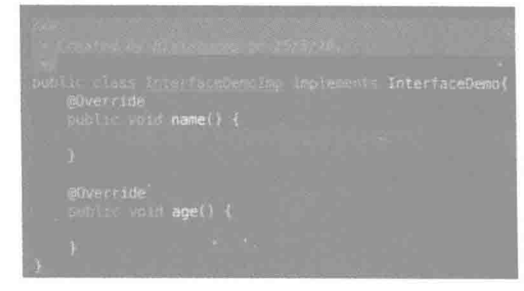


图 8-11

然后就可以在里面实现接口中定义的方法了。

8.3　实现代理方法

实现代理方法前我们需要先了解什么是代理模式。

为了控制其他对象对某个对象的访问而提供的一种代理叫代理模式。简单来说，代理模式就像是你想访问某个对象时不能直接访问，必须通过一个中介来间接访问一样。

例如，我们搞个活动需要找明星来站台，通常情况下我们不可能直接联系明星，而是通过演艺公司（代理）来联系，代理在这里就起到了一个中介的作用。

代理模式一般涉及的角色有以下 3 种。

- 抽象角色（明星要做的事，即唱歌、跳舞）：声明真实对象和代理对象的共同接口。
- 代理角色（演艺公司）。
  - 可以操作真实对象：代理对象角色内部含有对真实对象的引用，从而可以操作真实对象。
  - 可以代替真实对象：代理对象提供与真实对象相同的接口，以便在任何时刻都能代替真实对象。
  - 可以封装真实对象：代理对象可以在执行真实对象操作时附加其他的操作，相当于对真实对象进行封装。
- 真实角色（明星）：代理角色所代表的真实对象，是我们最终要引用的对象。

当我们需要快速生成代理方法的时候，代理角色的部分代码是可以自动生成的。

操作步骤：

菜单栏：Code→Delegate Methods

右键菜单：右击→Generate→Delegate Methods

快捷键：command + N（macOS）或者Alt + Insert（Windows/Linux）→Delegate Methods

【实例演示】

环境准备：新建 3 个类文件，Work.java（抽象角色：明星要做的事）、SuperStar.java （真实角色：明星）、Company.java （代理角色：演艺公司）。

代码如下：

Work.java

```
/**
 * 抽象角色：明星要做的
 * Created by bixiaopeng on 15/5/11.
 */
public abstract class Work {
    abstract public void sing();
    abstract public void dance();
}
```

SuperStar.java

```
/**
 * 真实角色：明星
 * Created by bixiaopeng on 15/5/11.
 */
public class SuperStar extends Work{
    @Override
    public void sing() {
        System.out.println("我要唱歌");
```

```
    }

    @Override
    public void dance() {
        System.out.println("我要跳舞");
    }
}
```

Company.java

```
/**
 * 代理角色：演艺公司
 * Created by bixiaopeng on 15/5/11.
 */
public class Company extends Work{
    private SuperStar superStar;

    @Override
    public void sing() {
        superStar.sing();
    }

    @Override
    public void dance() {
        superStar.dance();
    }
}
```

操作步骤：

01 将光标放在代理类（Company）中→菜单栏：Code → Delegate Methods。

02 选择需要为谁生成代理。

03 单击【OK】按钮，如图 8-12 所示。

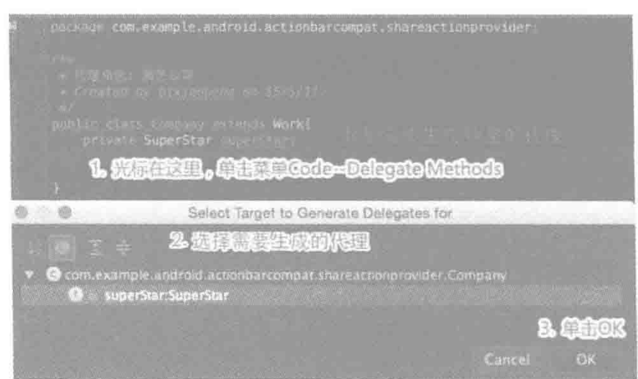


图 8-12

04 选择需要生成的代理方法。

05 单击【OK】按钮。

06 查看生成的代码方法，如图 8-13 所示。

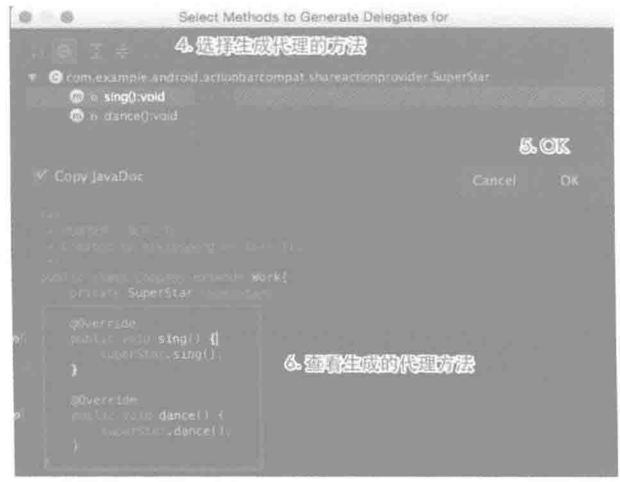


图 8-13

8.4 生成构造函数

菜单栏：Code→Generate→Constructor

快捷键：command + N（macOS）或者Alt + Insert→Constructor（Windows/Linux）

【实例演示】生成DataBean的构造函数。

01 将光标放在需要插入构造函数的地方。

02 按快捷键 command + N（macOS）。

03 选择 Constructor。

04 选择需要在构造函数中初始化的字段。

05 单击【OK】按钮，生成构造函数，如图 8-14 所示。

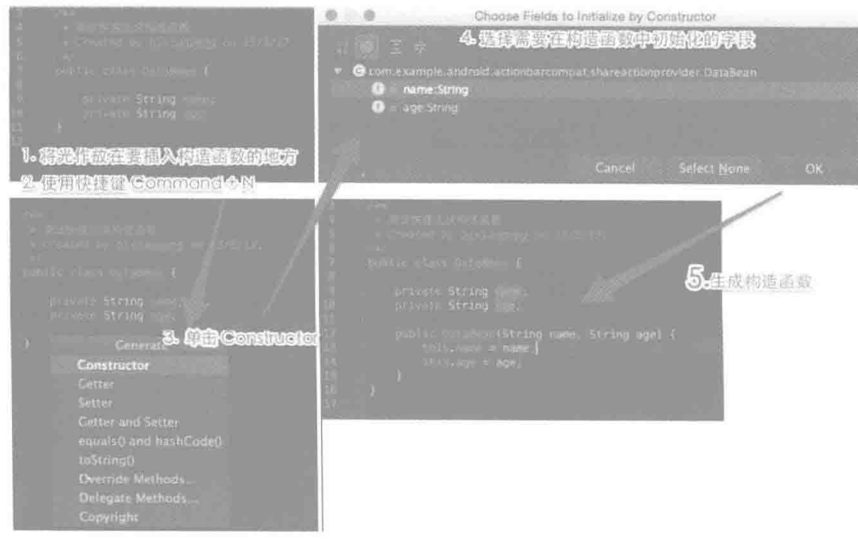


图 8-14

8.5 生成 Getter 和 Setter 方法

菜单栏：Code→Generate→Getter and Setter

快捷键：command + N（macOS）或者Alt + Insert（Windows/Linux）→Getter and Setter

【实例演示】快速生成Getter和Setter方法。

01 将光标放在要插入生成方法的地方。

02 按快捷键 command + N（macOS）。

03 选择 Getter and Setter。

04 选择需要生成 Getter 和 Setter 的字段，如图 8-15 所示。

05 查看结果，如图 8-16 所示。

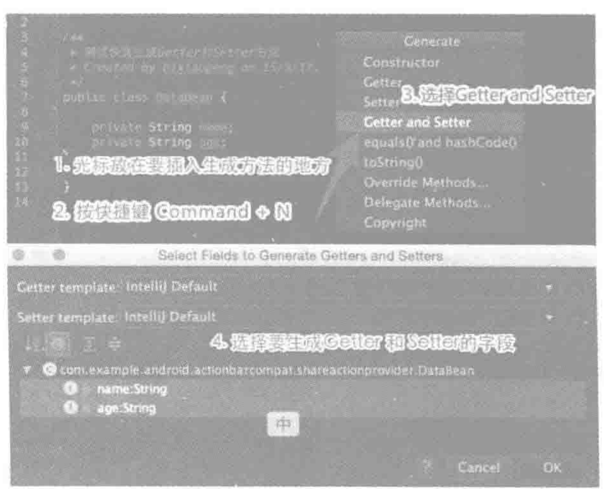


图 8-15

图 8-16

8.6 覆写 equals 和 hashcode 方法

equals方法和hashcode方法都是用来判断两个对象是否相等的。

（1）equals方法

① 判断两个对象的内容是否相等。例如，DataBean这个类有两个属性name和age，如果两个属性相等，就证明两个对象相等。

② 给用户调用的方法。如果要判断两个对象是否相等，就需要重写equals方法，然后在代码中调用。

（2）hashcode方法

① 判断两个对象的编码是否相等。和equals的不同在于它返回的是整型，比较起来不直观。一般要求在覆写equals的同时也要覆写hashcode，让它们的逻辑一致。

② 一般用户不会调用。例如，在hashmap中key是不可以重复的，它在判断key是否重复时就判断了hashcode这个方法，同时也用到了equals方法。

如果我们想快速覆写equals和hashcode方法，应该如何操作呢？

菜单栏： Code或者右键菜单→ Generate→equals() and hashCode()

快捷键： command + N（macOS）或者Alt + Insert（Windows/Linux）→equals() and hashCode()

【实例演示】

01 将光标定位在要插入方法的地方。

02 使用快捷键 command + N（macOS）。

03 单击 equals() and hashCode()，如图 8-17 所示。

04 选择默认的模板，如图 8-18 所示。

图 8-17

图 8-18

05 选择 equals 方法里包含的字段，如图 8-19 所示。

06 选择 hashCode 方法里包含的字段。

07 选择非空字段，结果如图 8-20 所示。

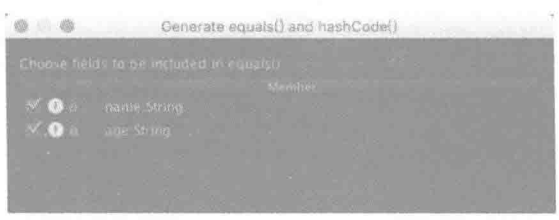


图 8-19

图 8-20

8.7 覆写 toString 方法

默认的toString方法打印出来的内容我们看不懂，为了能更好地除错，我们需要覆写toString方法。

菜单栏： Code或者右键菜单→Generate→toString()

快捷键： command + N（macOS）或者Alt + Insert（Windows/Linux）→toString()

【实例演示】

01 光标定位在要插入方法的地方。

02 使用快捷键 command + N（macOS）→单击 toString()。

03 选择模板（见图 8-21），结果如图 8-22 所示。

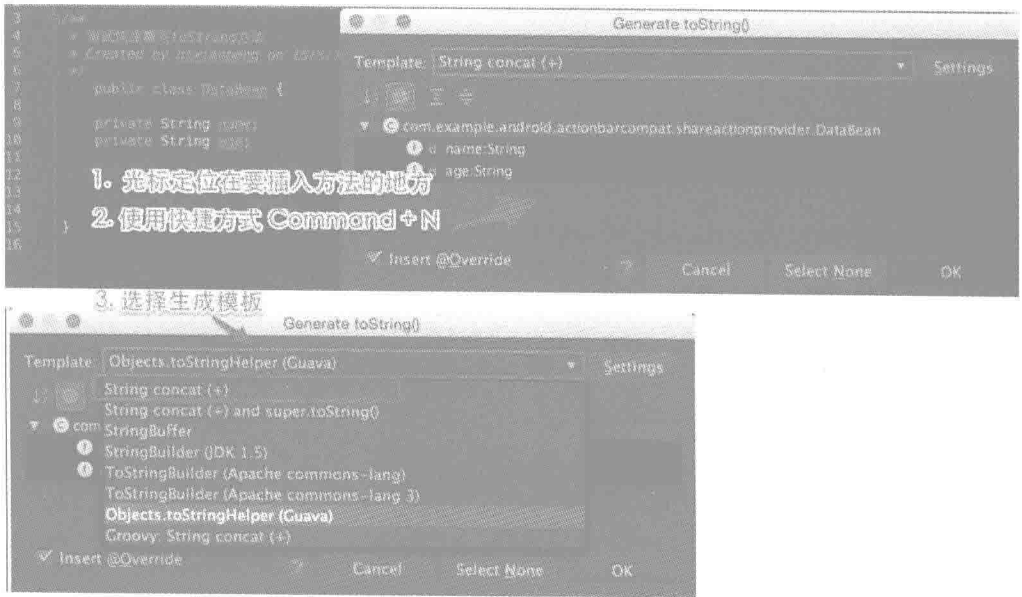


图 8-21

我们可以根据自己的实际需要定义toString方法的生成模板（见图 8-23），还可以对现有的模板进行增、删、改（见图 8-24）。

图 8-22

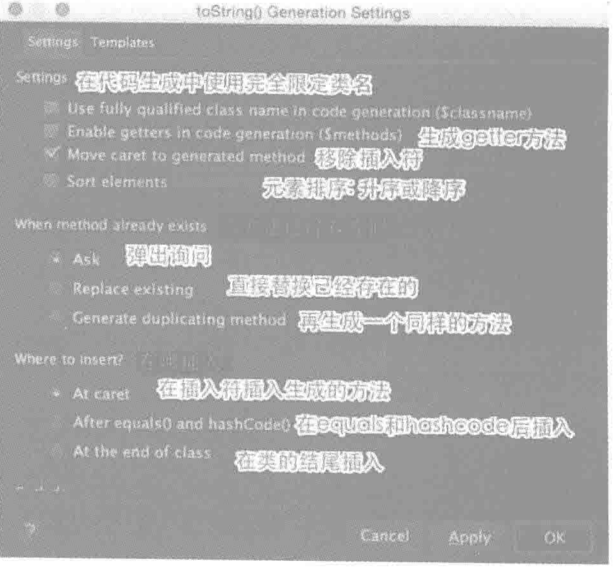


图 8-23

图 8-24

8.8 插入版权信息

版权信息（copyright）是指作者对其作品依法享有的某些特殊权利，亦称著作权。非经同意，他人不得出版或更改。如果一个文件包含许可证或版权信息，那么它应当被放在文件最前面：

```
/*
 * Copyright (C) 2013 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

8.8.1 插入版权信息

前提条件： 光标定位到要插入版权信息的地方。

菜单栏： Code或者右键菜单→ Generate→Copyright

快捷键： command + N（macOS）或者Alt + Insert（Windows/Linux）→Copyright

如果当前没有定义任何copyright信息，就会出现提示，如图 8-25 所示。

图 8-25

单击【OK】按钮，配置版权信息。

8.8.2 配置版权信息

菜单栏：Android Studio→Preferences→Copyright→Copyright Profiles

【实例演示】

01 新建一个版权信息文件，如图 8-26 所示。

图 8-26

02 配置版权信息模板，如图 8-27 所示。

```
/*
 * Copyright (C) $today.year 用于测试配置版权信息
 * 老男人有限公司 版权所有
 * OLD MAN CO., LTD. All Rights Reserved. */
```

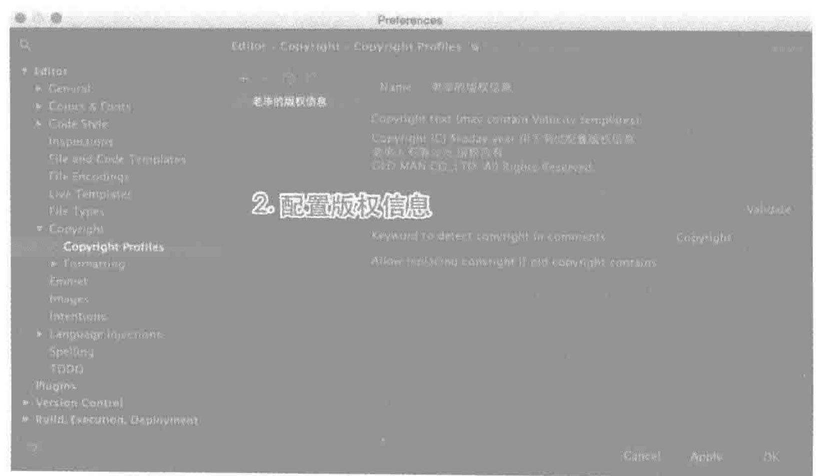


图 8-27

03 校验模板是否有效，如图 8-28 所示。

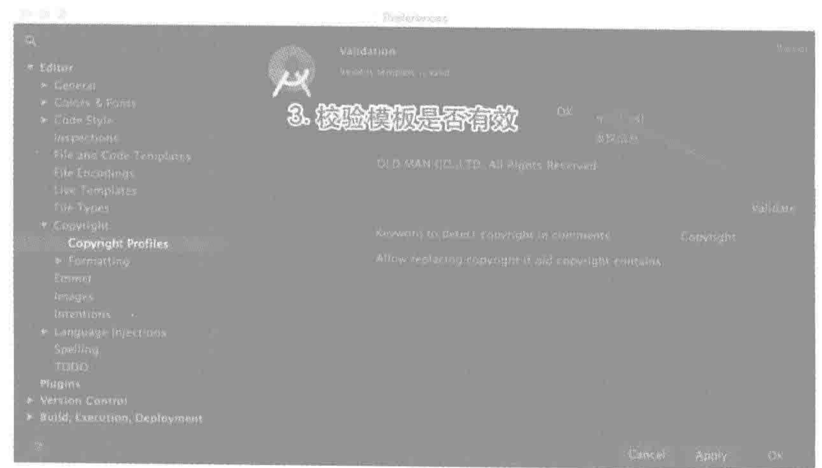


图 8-28

单击Copyright，可以选择默认的项目Copyright，如图 8-29 所示。

图 8-29

04 添加模板，根据自己的需求配置模板使用范围，如图 8-30 和图 8-31 所示。

图 8-30

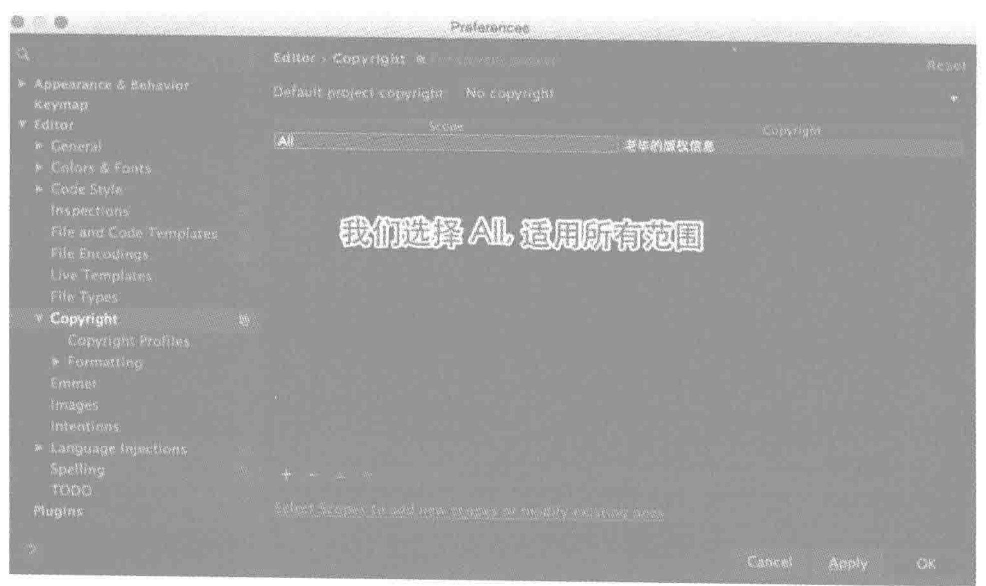


图 8-31

05 快速插入版权信息。使用快捷方式 command + N 或通过 Code → Generate → Copyright 来插入，如图 8-32 所示。

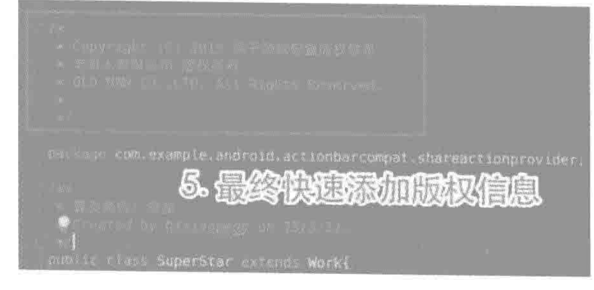


图 8-32

新建一个文件的时候，版权信息就会自动被加入，如图 8-33 所示。

如果在Copyright中添加了两个相同范围的Copyright，那么新的将会覆盖旧的，如图 8-34 所示。

图 8-33

图 8-34

8.8.3 共享版权信息配置

我们创建的copyright文件放在项目根目录/.idea/copyright下，直接把copyright这个目录复制给团队中的其他人即可实现共享。

```
#查看 copyright 下的文件
bixiaopeng@bixiaopengtekiMacBook-Pro copyright$ ls -l
total 16
-rw-r--r--  1 bixiaopeng  wheel  267 May 25 18: 16 profiles_settings.xml
#查看配置文件
bixiaopeng@bixiaopengtekiMacBook-Pro copyright$ cat profiles_settings.xml
<component name="CopyrightManager">
  <settings default="">
    <module2copyright>
      <element module="Tests" copyright="老毕的版权信息" />
      <element module="Project Files" copyright="版权信息 2" />
    </module2copyright>
  </settings>
</component>
```

8.9 提取或删除代码

如果我们想从for、foreach、if..elseif...else、try...catch...finally、while...do、do...while中快速提取或删除代码，可以按照如下方法操作。（注意，在提取或删除代码的时候，被绿色选中的是要提取的内容，被红色选中的是要删除的内容。）

菜单栏：Code→Unwrap/Remove

快捷键：command + shift + Delete（macOS）或者Ctrl + Shift + Delete（Windows/Linux）

【实例演示】

下面这段代码是我们要进行测试的：

```
try {
    System.out.println("测试快速提取或删除");
    System.out.println("测试快速提取或删除");
    System.out.println("测试快速提取或删除");
} catch (Exception e) {
    e.printStackTrace();
}
```

例 1：光标定位在System.out.println上，执行Unwrap/Remove，会把System.out.println()删除，把()中要打印的内容提取出来，如图 8-35 所示。

例 2：System.out.println外还嵌套有try或for等语句，执行Unwrap/Remove，会提示你选择到底要删除哪个，如图 8-35 所示。

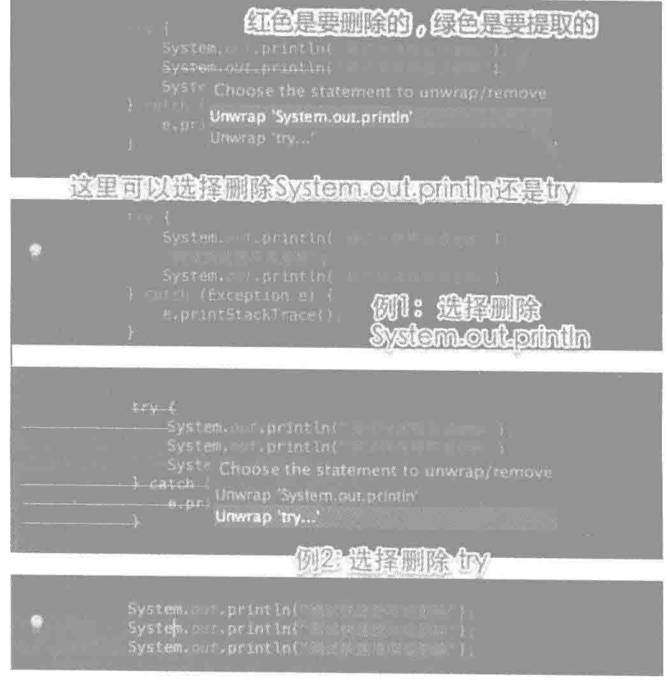


图 8-35

例 3：选中System.out.println，执行Unwrap/Remove会直接删除try语句，如图 8-36 所示。

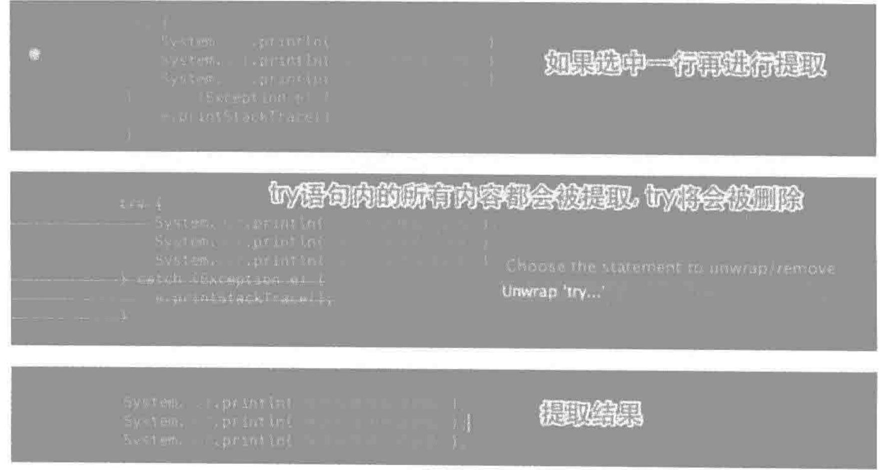


图 8-36

例 4：for、foreach、if..elseif...else等提取方法同上，如图 8-37 所示。

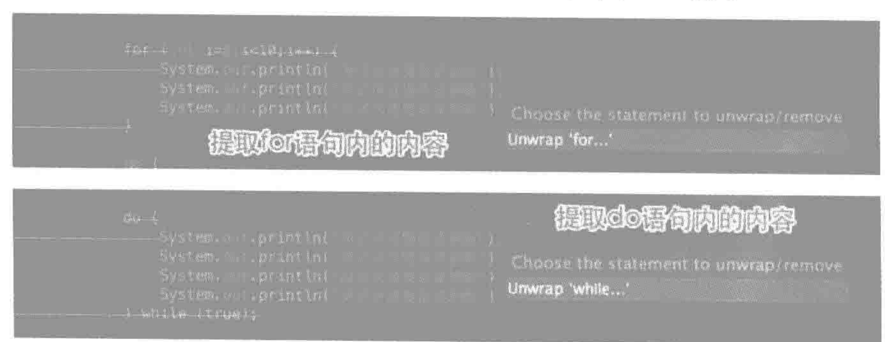


图 8-37

8.10 自动补全提示

1. 基本自动补全提示

当我们在输入代码的时候，Android Studio就会给我们自动补全提示，如图 8-38 所示。

图 8-38

在输入的时候没有自动触发，也可以触发自动补全提示。

菜单栏：Code→Completion→Basic（见图 8-39）

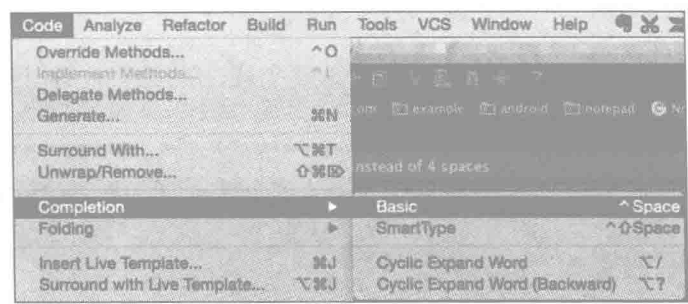


图 8-39

快捷键：control + 空格（macOS）或者Ctrl + 空格（Windows/Linux）

2. 智能自动补全提示

智能自动补全提示（见图 8-40）会把不适用的条目过滤掉，只显示可用的类、变量、属性或者方法，这个方法不但提升了性能，还规避掉了一些不必要的错误。

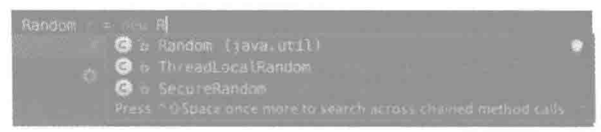


图 8-40

菜单栏：Code→Completion →SmartType

快捷键： control + Shift + 空格（macOS）或者Ctrl + Shift + 空格（Windows/Linux）

8.11 代码补全

当弹出代码补全提示时，我们可以用鼠标/Enter/Tab/!/./;进行补全，但这几种补全方法又是不同的。（需要确认代码补全的时候可以使用此功能。）

1. 补全后不删除后面的代码

操作步骤：弹出代码补全提示→选中补全代码→鼠标/Enter。

【实例演示】

01 在 name.之后使用快捷键 control + 空格（macOS）弹出代码补全提示。

02 选中补全代码，如图 8-41 所示。

03 鼠标/Enter。结果补全后没有删除后面的代码，如图 8-42 所示。

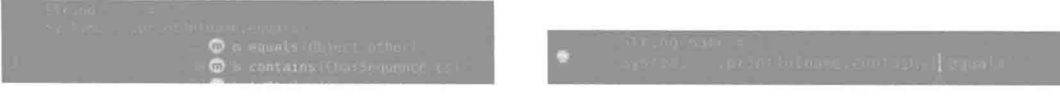


图 8-41　　图 8-42

2. 补全后删除后面的代码

操作步骤：弹出代码补全提示→选中补全代码→Tab。

【实例演示】

01 在 name.之后使用快捷键 control + 空格（macOS）弹出代码补全提示。

02 选中补全代码，如图 8-43 所示。

03 按 Tab 键，结果补全后删除了后面的代码，如图 8-44 所示。

图 8-43　　图 8-44

3. 布尔值取反补全

当我们需要补全一个布尔值再取反的时候可以使用此功能。

操作步骤：弹出布尔值代码补全提示→选中补全代码→ 叹号（!）。

【实例演示】

01 弹出布尔值代码补全提示→选中补全代码，如图 8-45 所示。

图 8-45

02 输入叹号（!）。结果补全后取反了，如图 8-46 所示。

图 8-46

4. 点和分号补全

在弹出代码提示的时候，如果使用点（.）和分号（;）补全会自动在补全后添加点（.）和分号（;）。

操作步骤：弹出代码补全提示→选中补全代码→点（.）或分号（;）。

【实例演示】

01 弹出代码补全提示→选中补全代码，如图 8-47 所示。

图 8-47

02 输入点（.）。结果补全了被选中的代码，同时在代码后面自动添加了点（.），如图 8-48 所示。

图 8-48

03 回到**02**，如果输入分号（;），就补全被选中的代码，同时在代码后面自动添加分号（;），如图 8-49 所示。

图 8-49

这里代码有误，请忽略，重点在于说明补全功能。

8.12 补全循环扩展词

如果我们想补全循环扩展的关键字，应该如何操作呢？

菜单栏：Code→Completion →cyclic expand word 或cyclic expand word （backward）

快捷键：option + / 或 option + ?（macOS）或者Alt + / 或Alt + Shift + /（Windows/Linux）

此功能会自动补全循环扩展的关键字，比如Random r = R（在这里触发循环扩展字，会自动补全为Random）。

如果快捷键跟电脑上其他的设置冲突了，可以更改**快捷键**：偏好设置→Keymap→Main Menu→Code→Completion，修改冲突的快捷键，如图 8-50 所示。

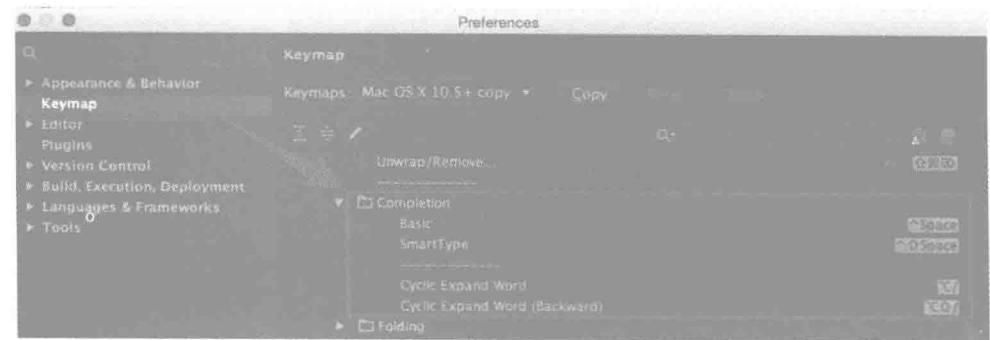


图 8-50

修改方法，如图 8-51 所示。

图 8-51

8.13 展开或折叠代码

在Android Studio中可以对结构代码（类、方法等）进行折叠和展开。这个功能的目的是为了让我们在不关心某些代码的时候可以将其折叠起来，想看的时候再展开。代码的展开和折叠状态如图 8-52 所示。

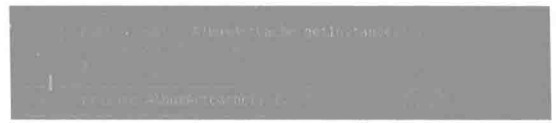


图 8-52

8.13.1 展开或折叠当前代码块

此功能仅会展开或折叠当前光标所在处的一段代码。

1. 展开当前代码块的代码

菜单栏：Code→Folding→Expand
快捷键：command + "+"（macOS）或者Ctrl + NumPad + "+"（Windows/Linux）

2. 折叠当前代码块的代码

菜单栏：Code→Folding→Collapse
快捷键：command + "-"（macOS）或者Ctrl + NumPad + "-"（Windows/Linux）

8.13.2 展开或折叠当前代码块中的所有子模块

1. 展开当前代码块中的所有子模块

菜单栏：Code→Folding→Expand Recursively
快捷键：option + command + "+"（macOS）或者Ctrl + Alt + NumPad + "+"（Windows/Linux）

2. 折叠当前代码块中的所有子模块

菜单栏：Code→Folding→Collapse Recursively

快捷键：option + command + "-"（macOS）或者Ctrl + Alt + NumPad + "-"（Windows/Linux）

8.13.3　展开和折叠全部代码块

1. 展开全部代码块

菜单栏：Code→Folding→Expand All
快捷键：shift + command + "+"（macOS）或者Ctrl + Shift + NumPad + "+"（Windows/Linux）

2. 折叠全部代码块

菜单栏：Code→Folding→Collapse All
快捷键：shift + command + "-"（macOS）或者Ctrl + Shift + NumPad + "-"（Windows/Linux）

8.13.4　展开和折叠当前文件中的所有注释

使用此功能会自动展开当前文件里的所有注释，不会展开代码（见图 8-53）。

图 8-53

展开代码注释：Code→Folding→Expand doc comments
折叠代码注释：Code→Folding→Collapse doc comments

8.13.5　指定展开层级

有时代码里嵌套了几层折叠代码，我们可以打开不同层级的代码（见图 8-54）。

图 8-54

指定展开代码层级：Code→Folding→Expand to level
指定全部展开代码的层级：Code→Folding→Expand all to level

8.13.6 展开和折叠选中区域

前提条件：选中要折叠代码片段。
菜单栏：Code→Folding→Fold Selection / Remove regio
快捷键：command + "."（macOS）或者Ctrl + "."（Windows/Linux）
利用菜单栏或快捷键操作后，选中的代码就会被折叠，再触发一次，代码就会被展开。

8.13.7 折叠代码片段

此功能用来折叠{}中的代码片段和自定义的代码片段。
菜单栏：Code→Folding→Fold code block
快捷键：shift + command +"."（macOS）或者Ctrl + Shift +"."（Windows/Linux）

8.14 插入代码模板

Live Template就是把常用的代码提取成一个模板，在编写代码的时候可以通过非常少的字母调出这个模板，达到快速输入、提高效率的目的。Android Studio中提供的Live Template定义了一些常用的缩写，我们可以通过输入缩写快速生成常用的代码模板。
菜单栏：Code→Insert Live Template
快捷键：command + J（macOS）或者Ctrl + J（Windows/Linux）
利用菜单栏或快捷键操作后在弹出的列表中选取缩写。当然最快的方法是直接输入LiveTemplate定义的缩写，然后按Tab键插入。

8.14.1 类中常用的缩写

类中常用的缩写（见图8-55）如下：

```
1.geti + tab：// 插入单例方法

public static 当前类名 getInstance() {
  return sInstance;
}

2.psf + tab：// 插入public static final

3.psfi + tab：// 插入public static final int

4.psfs + tab：// 插入public static final String

5.psvm + tab：// 插入main方法声明

6.St + tab：// 插入String
```

图 8-55

8.14.2　方法中常用的缩写

方法中常用的缩写如图 8-56 所示。

```
C       Surround with Callable
fori    Create iteration loop
I       Iterate Iterable | Array in J2SDK 5.0 syntax
ifn     Inserts ''if null'' statement
inn     Inserts ''if not null'' statement
inst    Checks object type with instanceof and down-casts it
itar    Iterate elements of array
itco    Iterate elements of java.util.Collection
iten    Iterate java.util.Enumeration
iter    Iterate Iterable | Array in J2SDK 5.0 syntax
itit    Iterate java.util.Iterator
itli    Iterate elements of java.util.List
ittok   Iterate tokens from String
itve    Iterate elements of java.util.Vector
lazy    Performs lazy initialization
lst     Fetches last element of an array
mn      Sets lesser value to a variable
mx      Sets greater value to a variable
```

图 8-56

常用代码模板：

```
1.ifn + tab：//null 判断
if (变量 == null) {
}

2.inn + tab：//非空判断
if (变量 != null) {
```

```
}

3.lazy + tab：//延迟初始化
if (对象 == null) {
   对象 = new String();
}

4.mn + tab：//得到 2 个变量中较小的
变量 = Math.min(变量 1， 变量 2);

5.mx + tab：//得到 2 个变量中较大的
变量 = Math.max(变量 1， 变量 2);

6.inst + tab：//判断变量是否是某个对象的实例
if (变量 instanceof Object) {
     Object  = (Object) 变量;
}

7.toar + tab：//把 collection 的对象存储到一个数组里
      .toArray(new Object[.size()])

8.thr + tab：// 抛出异常
throw new
```

循环遍历代码模板：

```
1.fori + tab：//for 循环
   for (int i = 0; i < ; i++) {
   }

2.itar + tab：//遍历数组
 for (int i = 0; i < array.length; i++) {
            = array[i];
}

3.itco + tab：//迭代器循环
   for (Iterator  = collection.iterator(); .hasNext(); ) {
           Object  =  .next();
   }

4.iten + tab：//遍历枚举内容
        while (enumeration.hasMoreElements()) {
           Object  =  enumeration.nextElement();
        }

5.iter + tab：//for each 循环
        for (Object  : ) {
        }

6.itit + tab：//遍历迭代器
        while (iterator.hasNext()) {
```

```
            Object  =  iterator.next();
        }
    7.itli + tab：//遍历 list
        for (int i = 0; i < list.size(); i++) {
            Object o =  list.get(i);
        }
    8.ritar + tab：//倒叙遍历数组
    for (int i = array.length - 1; i >= 0; i--) {
             = array[i];
        }
```

打印代码模板：

```
1.serr  + tab：System.err.println("");
2.souf  + tab：System.out.printf("");
3.sout  + tab：System.out.println()
4.soutm + tab：//打印当前类名和方法名
System.out.println("当前类名.方法名");
5.soutp + tab：//打印出当前方法的所有参数列表及其值
System.out.println("参数 1 = [" + 参数 1 的值 + "]，  参数 2 = [" + 参数 2 的值 + "]");
6.soutv + tab：//打印一个变量值
System.out.println("变量 = " + 变量值);
```

太多了，这里就不一一介绍了，如果大家想了解更多可以去看看所有的Live Template。

一般这些缩写都是模板关键词首字母的组合，所以很好记。

8.15 使用代码模板包裹代码

我们可以使用代码模板包裹一段代码，从而快速实现常用功能。

菜单栏：Code→Surround with Live Template

快捷键：option + command + J（macOS）或者Ctrl + Alt + J（Windows/Linux）

利用菜单栏或快捷键操作后选择代码模板，如图 8-57 所示。

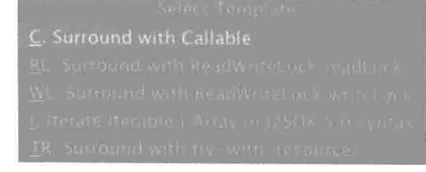


图 8-57

- C. Surround with Callable：使用 Callable 包围选中的代码。
- RL. Surround with ReadWriteLock.readLock：使用 ReadWriteLock 读锁包围选中的代码。
- WL. Surround with ReadWriteLock.writeLock：使用 ReadWriteLock 写锁包围选中的代码。
- I. Iterator Iterable | Array in J2SDK 5.0 syntax：使用遍历包围选中的代码。
- TR. Surround with try-with-resource：使用 try-with-resource 语句包围选中的代码。

在Select Template弹窗中单击模板或输入模板前面的大写字母都可以插入代码模板，如图 8-58 所示。

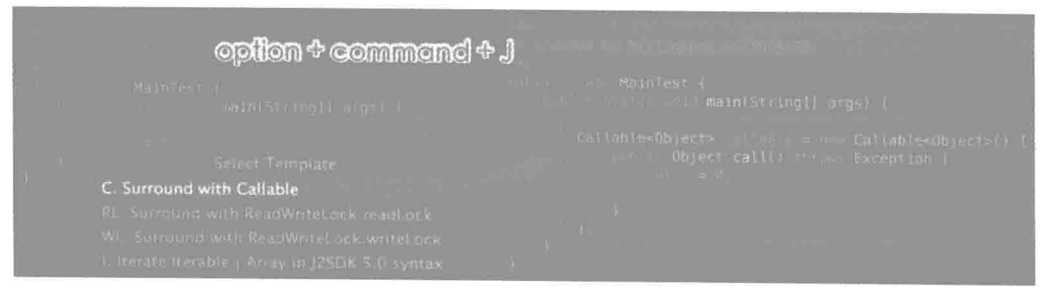


图 8-58

8.16 查看和编辑代码模板

1. 查看代码模板

菜单栏：Preferences→Editor→Live Templates

快捷键： option + command + T（macOS）或者Ctrl + Alt + T（Windows/Linux）

在弹出的对话框中选择【Configure Live Templates】，如图 8-59 所示。我们可以在这里对代码模板进行增、删、改，如图 8-60 所示。

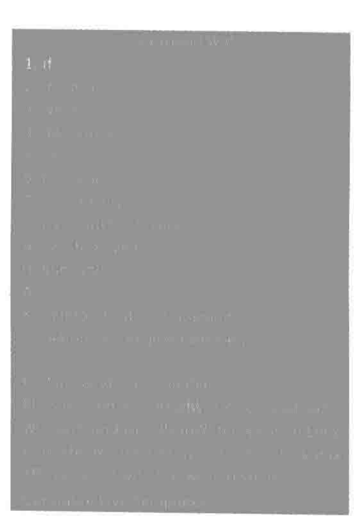


图 8-59

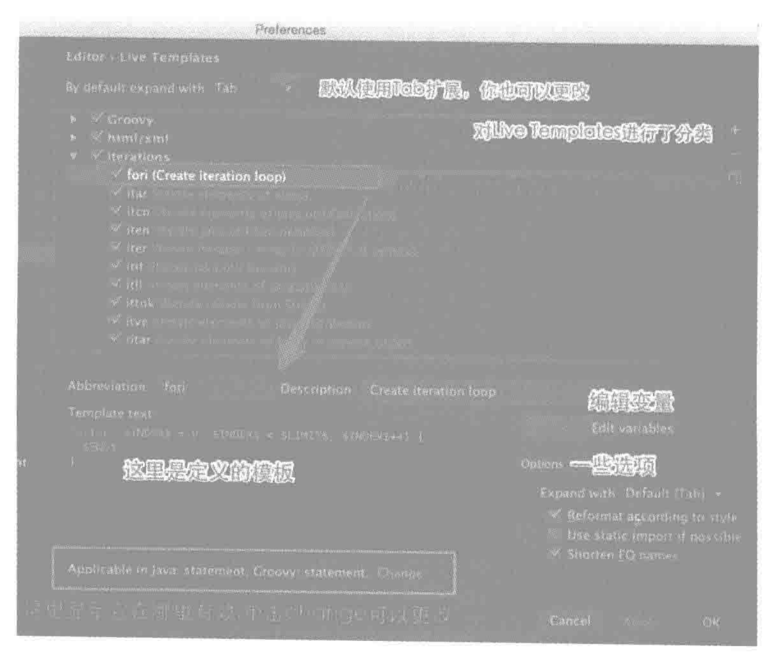


图 8-60

2. 编辑Live Templates

所有的缩写模板都是可以增、删、改的，根据使用习惯设置即可。例如，添加一个模板（见图 8-61，可以参考其他的模板是如何定义的，这个就不举例了）。

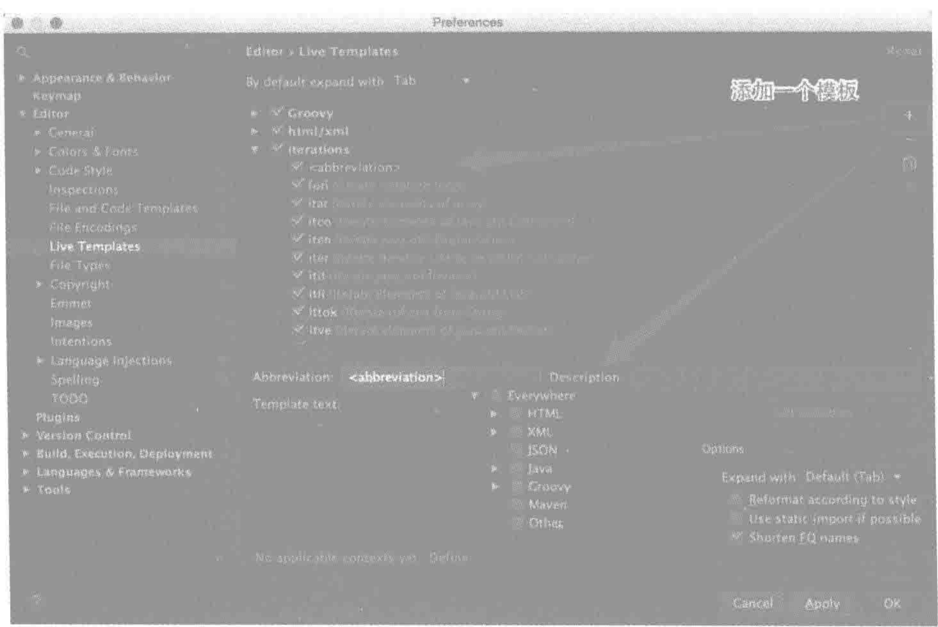


图 8-61

8.17 使用常用代码模板包裹代码

我们前面介绍过如何使用代码模板包裹代码，本节将介绍如何使用常用的代码模板来包裹代码。

菜单栏：Code→Surround with

快捷键：option + command + T（macOS）或者Ctrl + Alt + T（Windows/Linux）

在弹出的对话框中列出了常用的代码模板，如图 8-62 所示。

Surround With
1. if
2. if / else
3. while
4. do / while
5. for
6. try / catch
7. try / finally
8. try / catch / finally
9. synchronized
0. Runnable
A. { }
B. <editor-fold...> Comments
C. region...endregion Comments
Live templates
C. Surround with Callable
RL. Surround with ReadWriteLock readLock
WL. Surround with ReadWriteLock writeLock
I. Iterate Iterable | Array in J2SDK 5.0 syntax
TR. Surround with try-with-resources
Configure Live Templates...

图 8-62

【实例演示】

假设一个方法中有下面这条语句：

```
System.out.println("我是帅哥");
```

01 把光标放在这条语句上。

02 按快捷键 option + command +T（macOS）。

03 这些常用代码的插入效果如下。

选中if：

```
if () {
    System.out.println("我是帅哥");
}
```

选中if/else：

```
if () {
    System.out.println("我是帅哥");
} else {
}
```

选中while：

```
while (true) {
    System.out.println("我是帅哥");
}
```

选中do/while：

```
do {
    System.out.println("我是帅哥");
} while (true);
```

选中for：

```
for () {
    System.out.println("我是帅哥");
}
```

选中try/catch：

```
try {
    System.out.println("我是帅哥");
} catch (Exception e) {
    e.printStackTrace();
}
```

选中try/finally：

```
try {
    System.out.println("我是帅哥");
} finally {
}
```

选中try/catch/finally：

```
try {
    System.out.println("我是帅哥");
} catch (Exception e) {
    e.printStackTrace();
} finally {
}
```

选中synchronized：

```
synchronized () {
    System.out.println("我是帅哥");
}
```

选中Runnable：

```
Runnable runnable = new Runnable() {
    public void run() {
        System.out.println("我是帅哥");
    }
};
```

选中{}：

```
{
    System.out.println("我是帅哥");
}
```

选中Comments：

```
//<editor-fold desc="Description">
System.out.println("我是帅哥");
//</editor-fold>
```

选中region...endregion Comments：

```
//region Description
System.out.println("我是帅哥");
//endregion
```

8.18 注释代码

我们可以快速注释一行或一段代码。

1. 注释行

菜单栏：Code→Comment with Line Comment

快捷键：command + /（macOS）或者Ctrl + /（Windows/Linux）

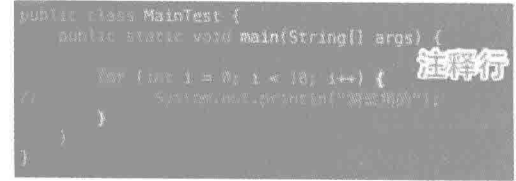
2. 注释代码块

菜单栏：Code→Comment with Block Comment

快捷键：option + command + /（macOS）或者Ctrl + Shift + /（Windows/Linux）

【实例演示】

注释一行和一段代码的效果如图 8-63 所示。



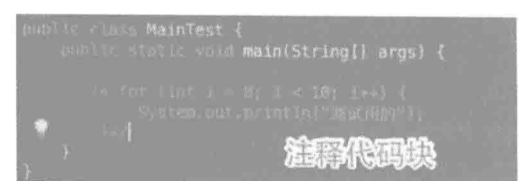


图 8-63

8.19 格式化代码

Android Studio预设了代码风格，我们可以快速格式化代码，统一代码风格。

菜单栏： Code或者右键菜单→Reformat Code

快捷键： option + command + L（macOS）或者Ctrl + Alt + L（Windows/Linux）

【实例演示】

代码格式化前后的对比如图 8-64 和图 8-65 所示。

图 8-64

图 8-65

默认的Reformat 不会优化导入，也不会重新排列代码，如果想在格式化的时候同时优化导入和重新排列代码，可以配置格式化选项：

通过**快捷键：** option + command + shift + L（macOS）打开配置对话框，如图 8-66 所示。

右击要优化的文件（可以多选，也可以选择包，还可以选择项目），在弹出的菜单中选择Reformat Code，就会弹出优化选项，可以格式化文件，如图 8-67 所示。

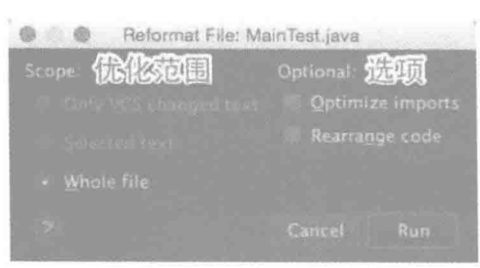


图 8-66

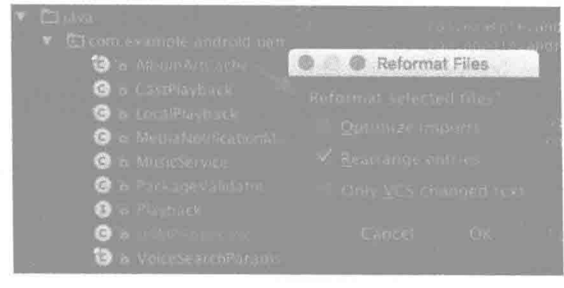


图 8-67

- Optimize imports：优化导入。
- Rearrange entries：重新排列代码。
- Only VCS changed text：仅优化版本控制系统中的文本。

8.20 自动缩进行

自动缩进行是优化代码风格的一个选项，当我们需要单独使用它的时候，可以利用如下方法。

菜单栏：Code→Auto-Indent Lines

快捷键：option + control + I（macOS）或者Ctrl + Alt + I（Windows/Linux）

【实例演示】

假设有一个缩进有问题的代码（见图 8-68），当光标放在 1 处的时候，执行自动缩进行，执行完成后光标会自动跳到下一行，这样我们就可以仅仅通过键盘完成界面上所有代码的缩进。（缩进以后的代码如图 8-69 所示。）

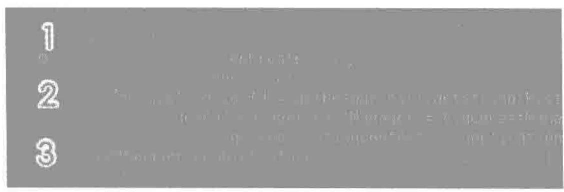


图 8-68

图 8-69

8.21 优化导入

我们可以优化导入，去除代码中无效的导入。

菜单栏：Code→Optimize Imports

快捷键：control + option + O（macOS）或者Ctrl + Alt + O（Windows/Linux）

右击文件菜单：Optimize Imports

【实例演示】

优化导入前和优化导入后的效果对比如图 8-70 所示。

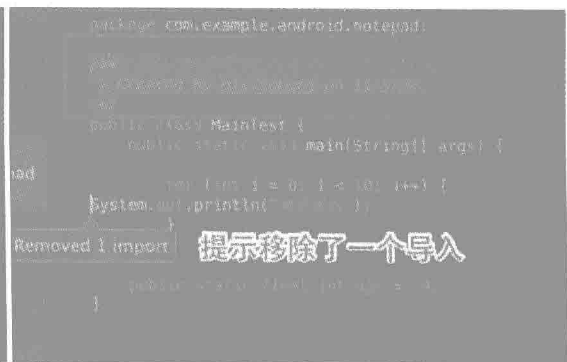


图 8-70

8.22 重新排列代码

我们可以使用预设的代码规则重新排列代码。
菜单栏：Code→Rearrange Code

【实例演示】

重新排列代码的效果如图 8-71 所示。

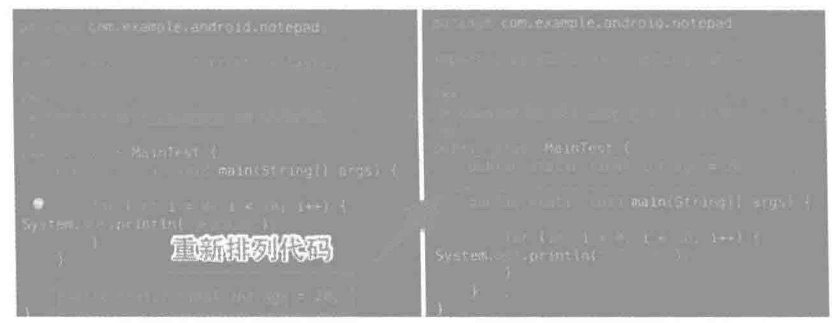


图 8-71

8.23 移动一段代码

我们可以在编辑器中移动一段代码或一个方法。

前提条件：如果要移动一段代码，需要先选中这段代码；如果要移动方法，光标需要定位在方法名上。

1. 向上移动代码

菜单栏：Code→Move Statement Up
快捷键：shift + command + ↑（macOS）或者Ctrl + Shift + ↑（Windows/Linux）

2. 向下移动代码

菜单栏：Code→Move Statement Down
快捷键：shift + command + ↓（macOS）或者Ctrl + Shift + ↓（Windows/Linux）

【实例演示】

例 1：移动一段代码。

01 选中这段代码。
02 通过快捷键 shift + command + ↓/↑（macOS）来移动代码，如图 8-72 所示。

例 2：移动一个方法。

01 将光标定位在方法名上。
02 通过快捷键 shift + command + ↓/↑（macOS）来移动方法，如图 8-73 所示。

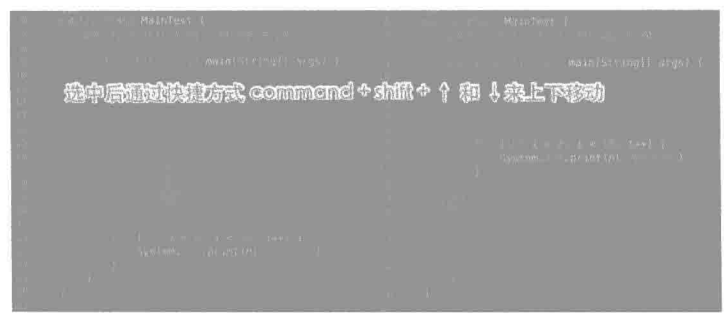


图 8-72

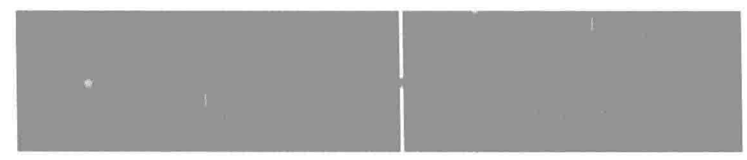


图 8-73

8.24 移动一行代码

我们可以不用鼠标快速移动一行代码。

前提条件：将光标定位在需要移动的代码行上。

1. 向上移动代码

菜单栏：Code→Move Line Up

快捷键：option + command + ↑（macOS）或者Alt + Shift + ↑（Windows/Linux）

2. 向下移动代码

菜单栏：Code→Move Line Down

快捷键：option + command + ↓（macOS）或者Alt + Shift + ↓（Windows/Linux）

【实例演示】

01 将光标定位在需要移动的代码行上。

02 使用快捷键 option + command + ↓/↑（macOS）来移动代码，如图 8-74 所示。

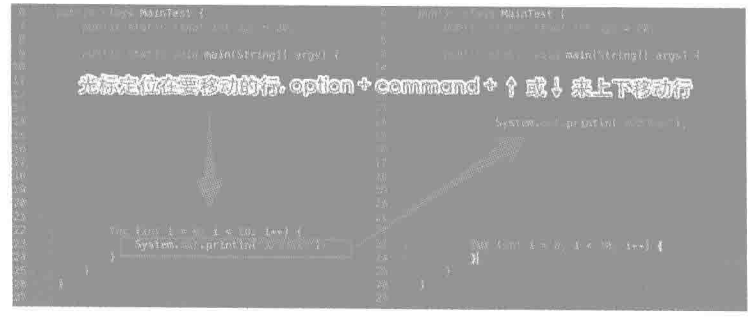


图 8-74

8.25 操作意图提示

Android Studio的操作意图提示可以帮助我们快速解决代码中可能存在的问题，还可以帮助我们优化和改进代码。会根据我们当时的操作场景列出很多可能的操作意图，当代码中有错误时，会显示红色的灯泡，没有错误时会显示黄色的灯泡，单击灯泡会弹出操作意图提示列表。

有错误时提示解决方案（见图8-75），没有错误时则提示优化方案（见图8-76）。

图 8-75

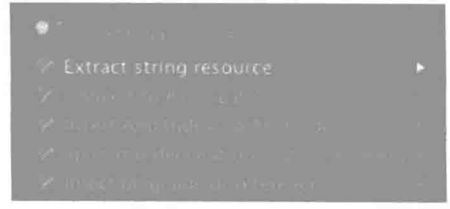


图 8-76

当我们想快速解决问题或优化和改进代码的时候可以使用此功能。

前提条件：光标定位在需要操作的代码上。

操作步骤：单击灯泡→按快捷键option + Enter（macOS）或者Alt + Enter（Windows/Linux），然后会弹出操作意图建议列表（见图8-77），再根据需要完成操作。

图 8-77

【实例演示】

例 1：没有导入用到的类。

01 将光标放在类名上→ 显示红色的灯泡。

02 单击灯泡 → 弹出快速修复建议列表，如图 8-78 所示。

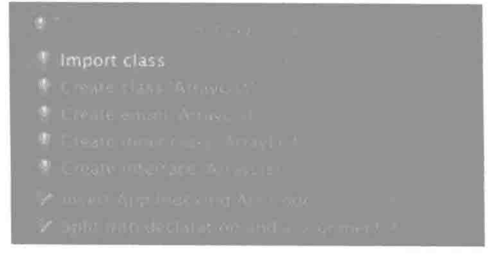


图 8-78

提 示

使用快捷键 option + Enter（macOS）可以快速导入类。

例 2：拆分声明和赋值。

01 将光标放在要拆分的语句上。

02 按快捷键 option + Enter（macOS）→弹出操作意图建议列表，如图 8-79 所示。

图 8-79

03 单击 Split into declaration and assignment。

04 执行操作，结果如图 8-80 所示。

提 示

Android Studio 上的操作意图是可以配置的，在偏好设置→ Editor → Intentions 中可以查看、开启或禁用某个或某类意图提示。

图 8-80

8.26 正则表达式操作意图提示

我们可以使用工具输入正则表达式并校验其正确性。

快捷键：option + Enter（macOS）或者Alt + Enter→Check RegExp（Windows/Linux）

【实例演示】输入并验证正则表达式的正确性。

01 将光标定位在""之间。

02 按快捷键 option + Enter（macOS），弹出选择列表，如图 8-81 所示。

03 单击 Check RegExp。

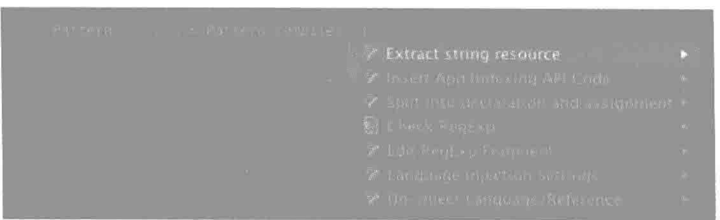


图 8-81

04 弹出对话框，输入正则表达式和匹配样本。我们可以在这里调试正则表达式。如果匹配成功，样本输入框会显示绿色，右下角提示【Matches!】（见图 8-82）；如果匹配失败，样本输入框会显示红色，右下角提示【no match】（见图 8-83）。之后按 Esc 键关闭弹窗。

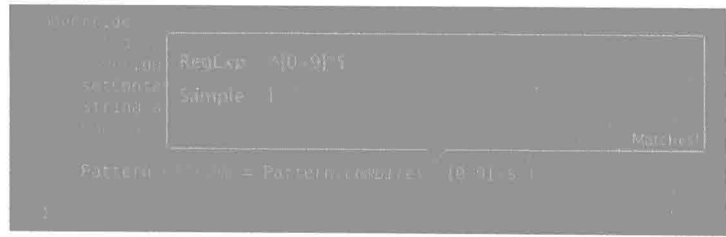


图 8-82

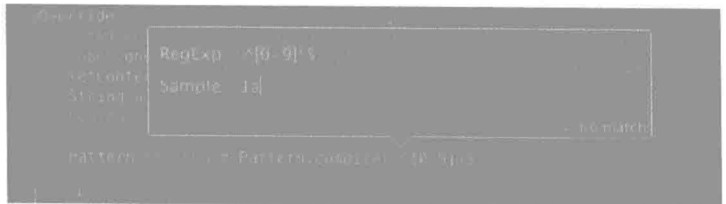


图 8-83

第 9 章 检 查

为了保证代码的质量，我们除了测试以外还需要注意代码规范，需要使用各种最佳实践来保证代码的性能和稳定性。Android Studio提供了代码检查工具，可以帮助我们对代码进行静态分析，找出代码中潜在的问题，以便提前修复。

本章将向大家介绍Android Studio中如何使用代码检查工具。

本章重要知识点 >>>>>>>>>>

- 代码检查工具 Lint 的使用方法；
- 代码检查工具的配置、管理；
- 代码检查规则的启用和禁用；
- 代码检查的运行方法；
- 分析代码依赖的方法。

9.1 代码检查工具

1. 代码检查工具介绍

Android Studio提供了功能强大、快速、灵活的代码检查工具，能够检测出编译器和运行时的错误，在编译之前建议修正和改进。

Android Studio代码检查不仅能检查出编译错误，还会检查出一些效率低下的代码，支持一些代码规范、编程指南、最佳实践，当存在无法访问的代码、未使用的代码、非本地化字符串、无法解析的方法、内存泄漏甚至拼写的问题时，Android Studio的代码分析工具都能够快速检查出来。

目前支持的语言有Android、Java、XML、HTML等。

Android Studio集成了代码扫描工具Lint，可以帮助你很轻松地识别和纠正Android代码的结构和质量问题，本书将重点介绍Lint工具。

2. 代码检查是可以灵活配置的

我们可以在偏好设置中配置代码检查的规则，启用或禁用每一个代码检查或更改它的严重性，创建自己的配置文件，在不同的范围进行不同的检查，禁止在某个特定的代码片段检查等。

3. 代码检查所涵盖的最常见的任务

- 发现可能的错误；
- 定位死代码；

- 检测性能问题;
- 改进可维护性和代码结构;
- 与编码准则和标准一致;
- 符合规范。

9.2 全面了解 Lint

9.2.1 Lint是什么

Lint是谷歌从Android 4.1 版本开始提供的代码分析工具，可以帮助开发找到项目工程中存在的问题以及一些冗余的不规范的文件代码，可以在不执行应用或测试用例的情况下检查出代码结构和质量问题，被检查出来的问题会以报告的形式展示出来，包括问题的分类、严重级别和描述。

Lint检查报告如图 9-1 所示。

Lint Report

Check performed at Sun Jun 07 18:14:24 CST 2015.
4 errors and 11 warnings found:

Lint
4 ! LintError: Lint Failure

Correctness
3 ⚠ OldTargetApi: Target SDK attribute is not targeting latest version
2 ⚠ MissingVersion: Missing application name/version

Security
2 ⚠ ExportedContentProvider: Content provider does not require permission
3 ⚠ AllowBackup: Missing allowBackup attribute

Usability:Icons
1 ⚠ MissingApplicationIcon: Missing application icon

Disabled Checks (195)

图 9-1

9.2.2 为什么要用Lint

- Lint 通过代码检查，可以发现潜在的问题，并能对 Android 程序进行优化处理。
- Lint 发现的每一个问题都有一个描述信息和严重级别，可以快速地定位问题，并根据严重级别来确定优先级。
- 习惯使用 Lint 可以提高 Android 应用的性能、可靠性和兼容性，让代码更容易阅读和维护。

9.2.3 Lint会检查哪些错误

Android Lint主要用于检查以下这些错误。

- 缺少转译（和未使用的转译）。
- 布局性能问题（所有的问题都是通过 layoutopt 工具找出来的）。
- 未使用的冗余资源。
- 在数组中定义多个配置时，数组大小不一致。
- 可访性问题和国际化的问题（硬编码，缺少 contentDescription）。
- 图标问题（失真、重复图标、错误尺寸等）。
- 可用性问题（例如，在输入框中不能指定输入方式）。
- 配置文件错误。

更多内容请参考http://tools.android.com/tips/lint-checks。

9.2.4 Lint工作流程

Lint 工作流程如图 9-2 所示。

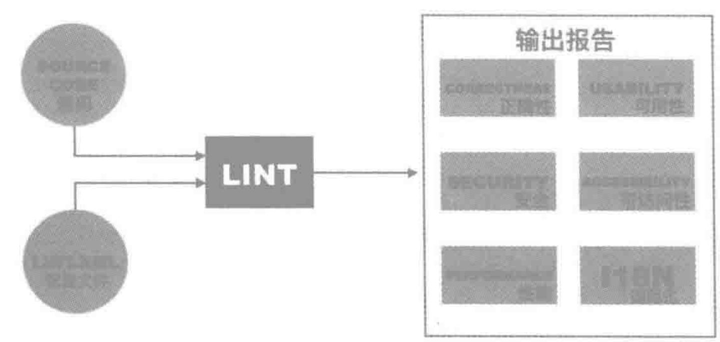


图 9-2

（1）源码：包括Java、XML、icon、ProGuard配置文件。

（2）配置文件：可以在lint.xml这个配置文件里配置严重程度和要排除的检查项。

（3）检查结果：可以在控制台或Android Studio的Event Log窗口中查看Lint检查结果，结果中会显示源码中每一个被检查出来的问题的位置和描述。

（4）输出报告：Lint工具的输出报告会从 6 个维度来分析代码存在的问题，并且会提供相应的修复建议。（可参考Android Studio偏好设置中对Lint的分类、问题描述和修复建议，见图 9-3）。

- 正确性（correctness）。
- 安全（security）。
- 性能（performance）。
- 可用性（usability）。
- 可访问性（accessibility）。
- 国际化（internationalization）。

图 9-3

9.2.5 报告中的Issue和Category

1. Issue

Lint检查出的所有问题都是以Issue（问题）来描述的。Issue以一个文本短语来作为id，对Issue的定制等操作都是基于id的。Issue以Severity（严重性）来标识危害程度：Fatal/Error/Warning/Information/Ignore。对Issue的忽略操作其实就是降低它的严重性为Ignore。

2. Category

Lint中的Issue被分为10个Category （类别，见图9-4）。

图9-4

- Accessibility（可访问性）。
- Correctness（正确性）。
- Correctness > Messages（正确性：信息）。
- Internationalization（国际化）。
- Internationalization > Bidirectional Text（双向文本）。
- Performance（性能）。
- Security（安全）。
- Usability（可用性）。
- Usability：Icons（可用性：图标）。
- Usability：Typography（可用性：字体）。

下文所有的Issue和Category都称为问题和类别。

9.2.6 Lint使用场景

Android Lint有两种使用场景，即IDE和命令行。Android Studio/Intellij idea/Eclipse/Jenkins都有Android Lint的插件支持。本书重点介绍Android Studio中Lint的使用，同时会顺便介绍Lint命令行的使用。

9.2.7 如何配置Lint检查

默认情况下执行Lint检查时会检查所有的问题，当然我们是可以通过配置来限定Lint检查问题类别和问题严重程度的。

也可以配置Lint的检查范围：

- 整个项目。
- 项目中的每个模块。
- 项目中的每个产品模块。
- 项目中的每个测试模块。
- 每个打开状态的文件。
- 每个类结构。
- 每个版本控制系统范围。

使用下面这些方法配置Lint：

- 在 Android Studio 偏好设置中配置（参见 9.12 配置代码检查规则）。
- 在 lint.xml 文件中配置（参见 9.15 在 lint.xml 文件中配置 Lint 检查）。
- 在 build.gradle 文件中配置（参见 9.16 在 Gradle 中配置 Lint 检查）。
- 在 Java 和 XML 源码中配置（参见 9.18 在 Java 和 XML 源码中配置 Lint 检查）。

9.2.8 Lint命令行用法介绍

命令用法：

```
lint [flags] <项目目录>
```

Flags：

```
--help                  显示帮助命令
--help <topic>          有关特定主题的帮助，例如"suppress"
--list                  列出可用的问题 id，然后退出
--version               输出版本信息，然后退出
--exitcode              如果发现错误，将退出代码设置为 1
--show                  提供问题列表以及完整的解释
--show <ids>            针对指定的问题 id 显示完整的解释
```

启用检查：

```
Enabled Checks:
--disable <list>    禁用类别列表或特定问题 id，列表中应该是以逗号分隔的问题 id 或类别
--enable <list>     启用特定问题列表，将检查所有默认问题和指定启用的问题，列表中应该是
以逗号分隔的问题 id 或类别
--check <list>      只检查特定列表的问题，会先禁用一切，然后重新启用给定的列表问题。列
表中应该是以逗号分隔的问题 id 或类别
-w, --nowarn            仅检查 errors (忽略 warnings)
-Wall                   检查所有的 warnings，包括那些默认关闭的
-Werror                 将所有警告视为错误
```

```
--config <filename>      使用给定的配置文件来确定问题是启用还是禁用，如果项目包含一个lint.xml 文件，那么这个配置文件作为备用
```

提 示

如果项目中已经有 lint.xml 文件，对于某条 issue 的检查来说，先用项目中 lint.xml 的规则，然后是 config 指定的规则，最后才是系统默认的规则。

输出选项：

```
Output Options:
--quiet                     不显示进展
--fullpath                  错误输出使用完整的路径
--showall                   不截断长消息，显示完整的信息
--nolines                   输出信息中不包括错误所在源文件中的行，默认情况下是显示的，但这个参数可以关闭
--html <filename>           创建一个 HTML 报告，如果文件名是一个目录(或一个新的没有扩展名的文件名)，lint 将会为每个被扫描的项目创建一个报告
--url filepath=url          添加链接到 HTML 报告中，使用 URL 前缀取代本地路径前缀。映射可以是一个逗号分隔的路径前缀对应 URL 前缀的列表。例如：C:temp\Proj1=http://buildserver/sources/temp/Proj1。如果要关闭链接到文件，使用参数 --url none
--simplehtml <filename>     创建一个简单的 HTML 报告
--xml <filename>            创建一个 XML 文件报告
```

项目选项：

```
--resources <dir>        添加一个指定的文件夹(或路径)作为项目的资源目录
--sources <dir>          添加一个指定的文件夹(或路径)作为项目的资源目录，只有在一个单独项目中运行 lint 时才有效
--classpath <dir>        添加文件夹(或 jar 文件或路径)作为项目的工程类目录，只有在一个单独项目中运行 lint 时才有效
--libraries <dir>        添加文件夹(或 jar 文件或路径)作为项目的类库，只有在一个单独项目中运行 lint 时才有效
```

退出状态：

```
0                    Success
1                    Lint 检测错误
2                    Lint 用法
3                    不影响现有的文件
4                    Lint 帮助
5                    无效的命令行参数
```

【实例演示】扫描当前项目。

```
$ lint .

Scanning debug: .
Scanning debug (Phase 2):
Scanning
1.0.3: ..........................................................................
Scanning 21.0.3 (Phase 2): ..................... . ...................
canning debug: .
Scanning debug (Phase 2):
```

```
    Scanning 21.0.3: .
    Scanning 21.0.3 (Phase 2):
    Scanning release: .
    Scanning release (Phase 2):
    ...
    ###################### 1.显示扫描出的问题 ######################
        <uses-sdk android: minSdkVersion="15" android: targetSdkVersion="21" />
                                            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ###################### 2.问题描述 ######################
    app/build/intermediates/manifests/full/debug/AndroidManifest.xml: 31:
Warning: Not targeting the latest versions of Android; compatibility modes apply.
Consider testing and updating this version. Consult the
android.os.Build.VERSION_CODES javadoc for details. [OldTargetApi] #问题标签
           android: targetSdkVersion="21" />
           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    app/build/intermediates/manifests/full/release/AndroidManifest.xml: 31:
Warning: Not targeting the latest versions of Android; compatibility modes apply.
Consider testing and updating this version. Consult the
android.os.Build.VERSION_CODES javadoc for details. [OldTargetApi]
           android: targetSdkVersion="21" />
           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
...
    app/build/intermediates/manifests/test/debug/AndroidManifest.xml: 7: Warning:
Should explicitly set android: icon, there is no default [MissingApplicationIcon]
        <application>
        ^
    4 errors, 11 warnings ### 项目扫描结果
```

9.2.9 Lint命令行用法举例

接下来我们通过下面几个例子来介绍Lint命令参数的一些用法。

1. 如何查看哪些id和类别是可用的

```
~$ lint --list
```

【实例演示】

```
$ lint -list

## 有效的问题类别:
Valid issue categories:
   Correctness(正确性)
   Correctness: Messages(正确性: 信息)
   Security(安全)
   Performance(性能)
   Usability: Typography(可用性: 字体)
   Usability: Icons(可用性: 图标)
   Usability(可用性)
   Accessibility(可访问性)
   Internationalization(国际化)
   Bi-directional Text(双向文字)
```

```
## 有效的问题 id：（太多了就不一一翻译了）
Valid issue id's:
"ContentDescription": Image without contentDescription
"AddJavascriptInterface": addJavascriptInterface Called
"AlwaysShowAction": Usage of showAsAction=always
"LocalSuppress": @SuppressLint on invalid element
.....
```

2. 如何禁用某项检查

```
~$ lint --disable MissingTranslation, UnusedIds, Usability: Icons 项目路径
```

3. 如何启用某项检查（某些检查默认是禁用的）

```
~$ lint --enable MissingTranslation, UnusedIds, Usability: Icons 项目路径
```

4. 如何对项目只做某项检查

```
~$ lint --check MissingPrefix 项目路径
```

5. 如何查看某个id的说明

```
~$ lint --show id
```

【实例演示】查看MissingPrefix的说明。

```
$ lint --show MissingPrefix
MissingPrefix
--------------
Summary: Missing Android XML namespace

Priority: 6 / 10
Severity: Error
Category: Correctness

Most Android views have attributes in the Android namespace. When referencing
these attributes you must include the namespace prefix,  or your attribute will
be interpreted by aapt as just a custom attribute.

Similarly,  in manifest files,  nearly all attributes should be in the android:
namespace.
```

6. 如何以html格式输出Lint检查结果

```
~$ lint 项目地址 --html test.html
```

7. 检查本地项目

```
~$ lint <本地项目地址> --html <输出的 html 文件名.html>
```

【实例演示】

```
$ lint . --html report.html

Scanning debug: .
```

```
    Scanning debug (Phase 2):
    Scanning
    1.0.3: ..........................................................
..........................................................................
..............................
    Scanning 21.0.3 (Phase
2): .....................................................................
...............................................................
    Scanning debug: .
    Scanning debug (Phase 2):
    Scanning 21.0.3: .
    Scanning 21.0.3 (Phase 2):
    Scanning release: .
    Scanning release (Phase 2):
    Wrote HTML report to file:
/Volumes/warehouse/ExampleTestProject_AndroidStudio_v5.3/report.html
    Lint found 4 errors and 11 warnings
```

HTML报告如图 9-5 所示。

图 9-5

8. 检查网络项目

```
    ~ $ lint --html /tmp/report.html --url
/src/MyProj=http://buildserver/src/MyProject
```

9.3 执行一次代码检查

操作步骤：菜单栏 | 右击文件夹/文件→Analyze→Inspect Code→指定检查范围→OK。

【实例演示】对当前项目进行代码检查。

01 菜单栏→Analyze→Inspect Code→弹出指定检查范围对话框（见图 9-6）。

02 选择 Whole project → 使用默认的配置文件 → OK → 开始执行代码检查（见图 9-7）。

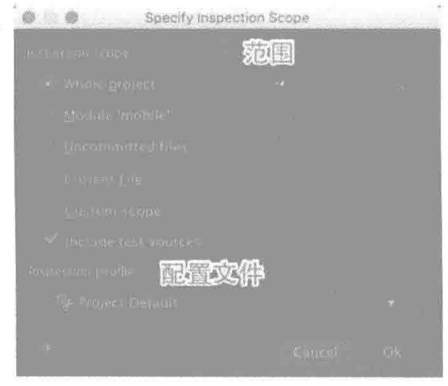


图 9-6

图 9-7

03 分析检查结果（见图 9-8）。

图 9-8

如图 9-6 所示，我们使用的代码检查规则是'Project Default'这个配置文件，所以检查结果会显示【Results for Inspection Profile 'Project Default'】。

结果中显示这个项目一共扫描出 772 个问题。分析工具把检查出来的问题进行了分类，其中问题最多的是拼写问题：Spelling（650 个）。单击问题分类可以查看问题的详细信息（见图 9-9）。

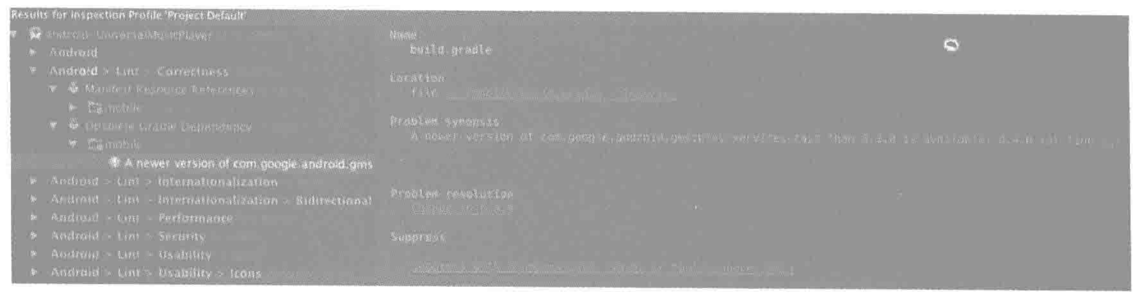


图 9-9

详细信息里非常明确地显示了有问题的文件名、位置、问题的描述、解决方案和忽略问题的方法。

9.4　指定检查范围

让我们再回到执行代码检查时的范围选择，详细介绍下指定检查范围。

指定检查范围有两种方法，即先执行检查再指定范围和选定范围再执行检查。两种方法的快捷程度差不多，大家可以按照自己的习惯来操作。

9.4.1　先执行检查再指定范围

操作步骤：菜单栏→Analyze→Inspect Code→弹出指定检查范围对话框，如图9-10所示。

图9-10

在这里可以选择执行代码检查的范围。

- 整个项目：Whole project。
- 当前的模块：Module 'mobile'。
- 未提交文件：Uncommitted files。
- 当前文件：Current File。
- 自定义范围：Custom scope。
- 包括测试资源：Include test sources。

默认是检查整个项目，还可以自定义检查范围（见图9-11）、指定检查的配置文件（见图9-12）。

图9-11

图9-12

检查的配置文件是可以增、删、改的，也支持导入导出，我们可以将团队制定的代码规范通过此功能共享给团队的所有成员。

9.4.2　选定范围再执行检查

先选中所要执行检查的文件夹/文件，再执行检查。

操作步骤：右击文件夹/文件→Analyze→Inspect Code→弹出指定检查范围对话框（见图9-13），默认选中文件夹/文件→单击OK即可进行检查（见图9-14）。

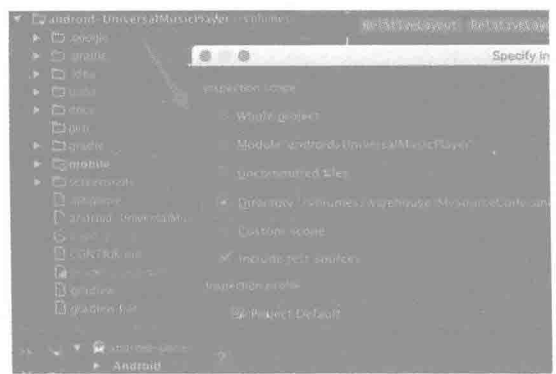


图 9-13

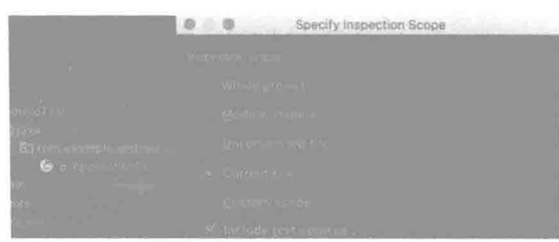


图 9-14

9.5 代码检查结果及辅助工具

我们先对整个项目执行一次代码检查，再来详细介绍怎么查看代码检查的结果。

1. 代码检查结果概览

如果有多次代码检查会显示多个结果标签。如图 9-15 所示，我们执行过两次检查，所以有两个标签。如果想关闭标签可以使用快捷键（Shift +鼠标单击），或者单击左边的辅助工具栏（见图 9-16）。

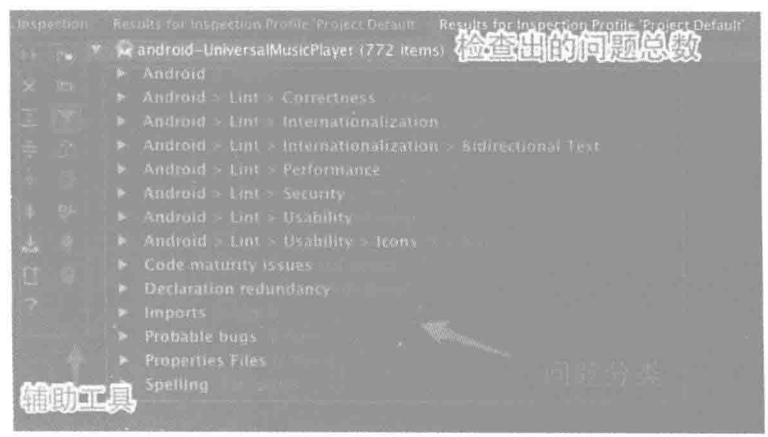


图 9-15

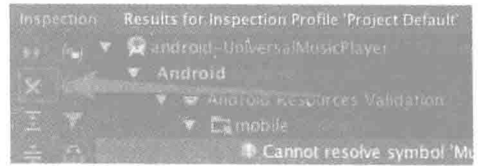


图 9-16

2. 查看具体检查项

展开检查结果的分类信息，会显示具体的检查项，如图 9-17 所示。

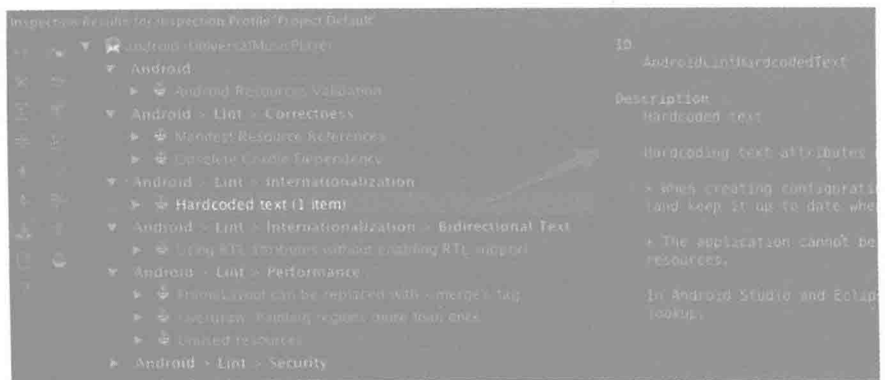


图 9-17

单击检查项会显示检查项的详情，请多注意查看检查项的描述信息。

3. 查看具体的问题

具体检查项下面会显示具体的问题，如图 9-18 所示。

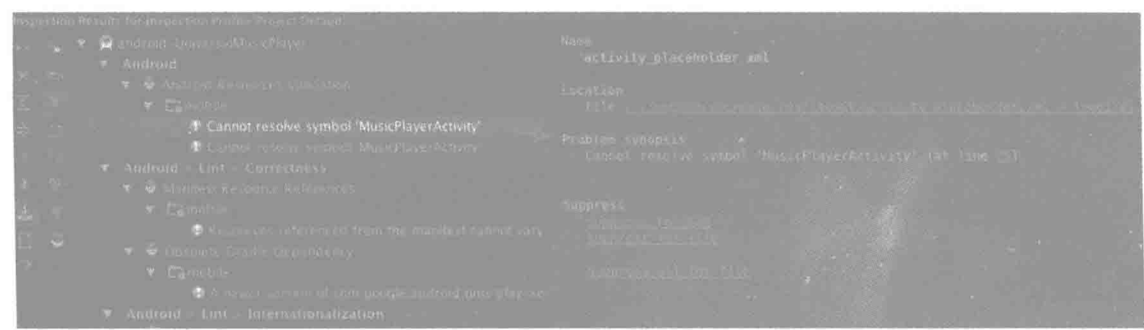
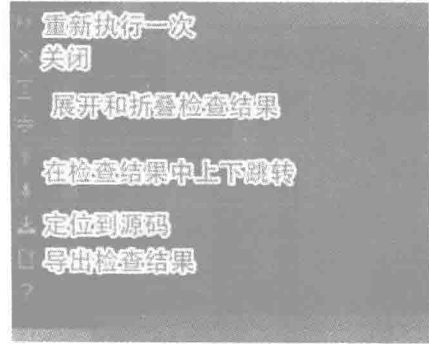


图 9-18

详细信息里非常明确地显示了有问题的文件名、位置、问题的描述、解决方案和忽略问题的方法。

4. 辅助工具概览

图 9-19 清晰地展示出有哪些辅助工具。



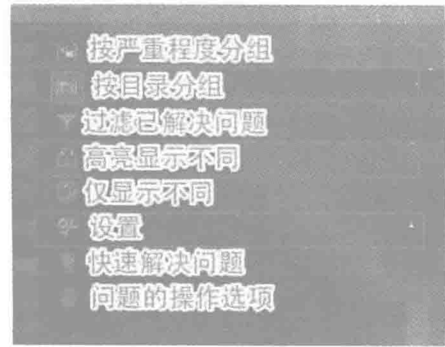


图 9-19

9.6 详解代码检查结果辅助工具

9.6.1 重新执行代码检查

在检查结果窗口重新执行代码检查，如图9-20所示。

注意，重新执行会覆盖上一次的执行结果。从菜单栏再执行一次，就会再打开一个执行结果标签。这两种执行是有区别的。

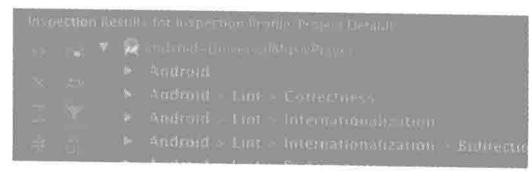


图 9-20

9.6.2 全部展开或折叠检查结果

1. 全部展开检查结果（见图 9-21）

图 9-21

快捷键：command + "+"（macOS）或者Ctrl + NumPad + "+"（Windows/Linux）

2. 全部折叠检查结果（见图 9-22）

图 9-22

快捷键：command + “-” （macOS）或者Ctrl + NumPad + “-” （Windows/Linux）

9.6.3 在检查结果中快速上下跳转

这个功能非常方便，在上下跳转的时候会同时定位到源码，这样就可以不用鼠标快速查看检查结果的所有问题，如图 9-23 所示。

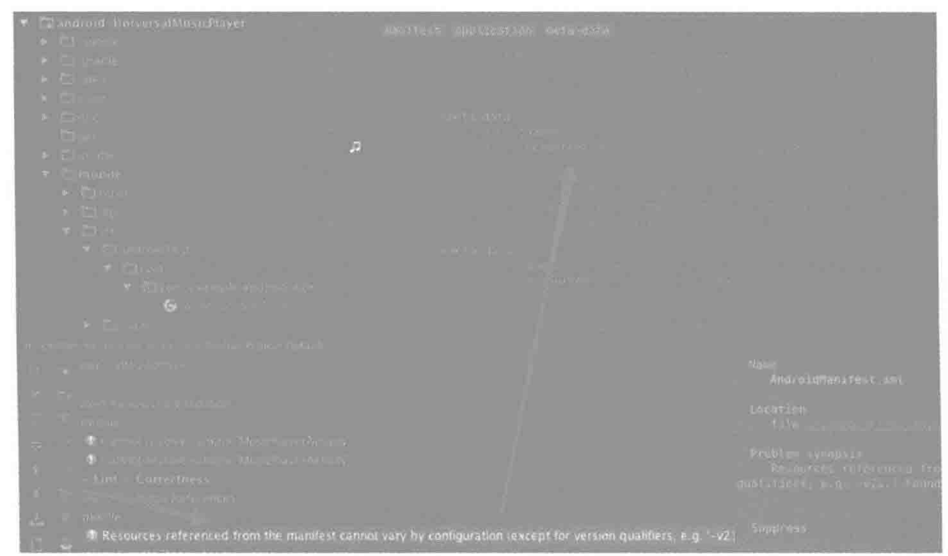


图 9-23

1. 跳转到上一个问题

快捷键：option + command + ↑（macOS）或者Ctrl + Alt + ↑（Windows/Linux）

2. 跳转到下一个问题

快捷键：option + command + ↓（macOS）或者Ctrl + Alt + ↓（Windows/Linux）

9.6.4 自动定位到问题的源码

默认情况下单击问题只会显示问题的详情，但是不会自动定位到源码，如果想单击的时候自动定位到源码，可以选中这个按钮，如图 9-24 所示。

此功能可以跟其他功能组合使用。

图 9-24

9.6.5 导出代码检查结果

如果我们需要把代码检查的结果作为报告发送出去，或者需要共享给项目组的同事，就要导出代码检查结果，如图 9-25 所示。

这里提供HTML和XML两种导出格式。

9.6.6 按严重程度分组排查问题

按严重程度分组会更有目的性，通常我们需要重点关注Error的问题，如图 9-26 所示。

图 9-25

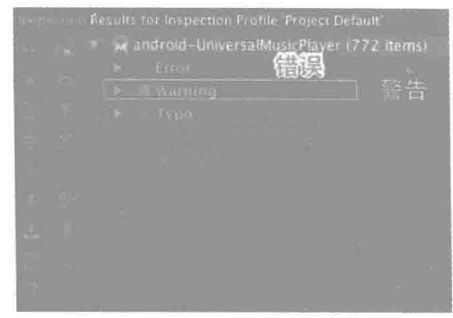


图 9-26

问题的严重程度是可以配置的，可以对关心的问题提高严重程度。

9.6.7 按目录分组排查问题

默认情况下检查结果会直接显示问题，在问题列表中不会显示问题所在的文件或目录。如果想更直观地通过路径来排查问题，可以使用按目录分组，效果如图 9-27 所示。

图 9-27

9.6.8 过滤已解决的问题

在检查结果中，当你已经解决了一些问题的时候，如果想让已解决的问题不再显示出来就可以过滤掉，如图 9-28 所示。

图 9-28

9.6.9 高亮显示不同和仅显示不同

与前一个版本对比，显示或过滤出新增的问题，如图 9-29 所示。

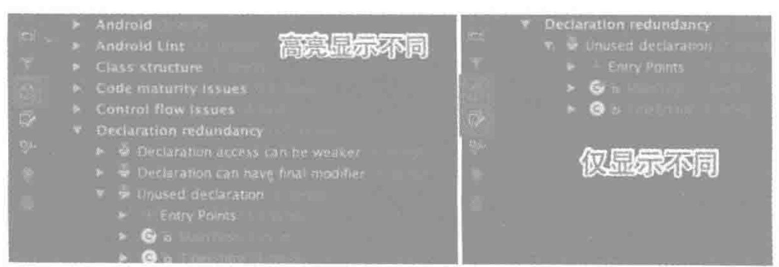


图 9-29

提 示

只有选中了高亮显示不同（有差异）文件，才能够使用仅显示（过滤）不同文件。

9.6.10 快速设置

这里的快速设置跟偏好设置中是一样的，只不过这里提供了一个快捷的入口，如图 9-30 所示。

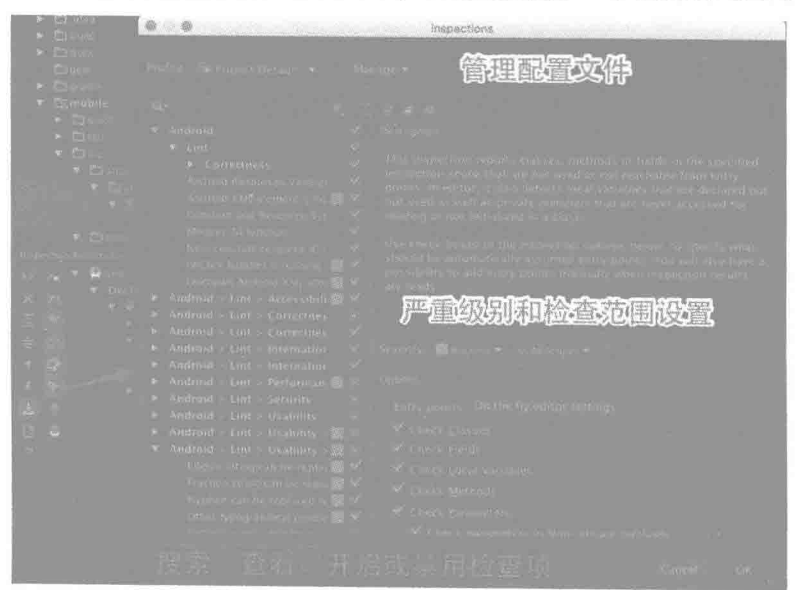


图 9-30

9.6.11 快速解决问题

快速解决问题（见图 9-31）只有在被检查出的问题有相应的解决方案时才能使用，否则是不可用的。单击快速解决问题按钮后，Android Studio会用相应的解决方案来解决这个问题，如图 9-32 所示。请注意最重要的一点，可以同时解决多个选中的问题。

快捷键：option + enter（macOS）或者 Alt + Enter（Windows/Linux）

图 9-31 快速解决问题按钮

图 9-32

9.6.12 对检查出的问题进行操作

代码检查项的操作选项一般包括禁用检查、指定检查范围和忽略检查。要对某个或某些检查项进行操作，首先要选中这个检查项或具体问题，我们举两个简单的例子。

01 检查项【Android Resources Validation】的操作选项如图 9-33 所示。

图 9-33

- Disable inspection：禁用【Android Resources Validation】检查。
- Run inspection on…：指定【Android Resources Validation】的检查范围。
- Suppress for tag：忽略【Android Resources Validation】对这个标签的检查。
- Suppress for file：忽略【Android Resources Validation】对这个文件的检查。
- Suppress all for file：忽略这个文件的所有检查项。

02 检查项【Unused declaration】的操作选项如图 9-34 所示。

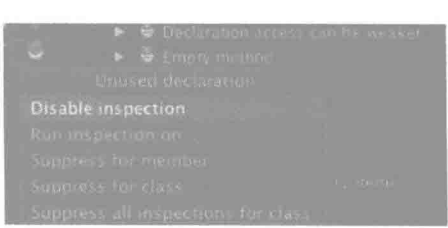


图 9-34

- Disable inspection：禁用【Unused declaration】检查。
- Run inspection on…：指定【Unused declaration】的检查范围。
- Suppress for member：忽略【Unused declaration】对这个成员（函数）的检查。
- Suppress for class：忽略【Unused declaration】对这个类的检查。
- Suppress all inspections for class：忽略这个类的所有检查项。

这里只做了简单的介绍，详细的使用请见下文。

9.7 禁用和启用某项检查

9.7.1 在检查结果中禁用和启用某项检查

当我们在检查结果中使用了Disable inspection（禁用检查）选项时，这个检查项将会被禁用，也就是说你再执行代码检查时，被禁用的检查项就不会被检查了。

1. 禁用某项检查

【实例演示】禁用【Android Resource Validation】这类问题的检查。

01 在检查结果中单击 Android→选中【Android Resource Validation】或具体问题。

02 右击或单击左边工具栏的操作选项→单击【Disable inspection】，右击 Android 的问题分类或具体问题，如图 9-35 所示。或选中 Android 的问题分类或具体问题后再单击左边工具栏的操作选项，如图 9-36 所示。

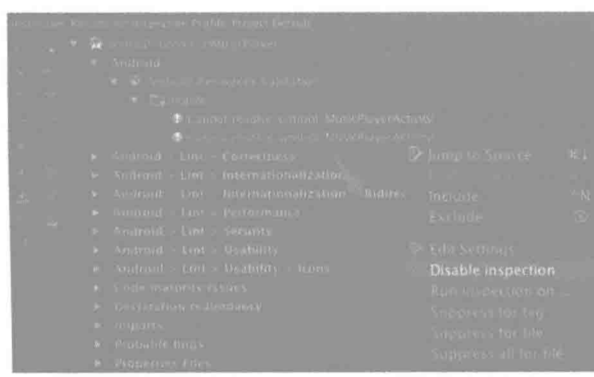


图 9-35

图 9-36

03 禁用【Android Resource Validation】检查的效果。单击【Disable inspection】后并没有任何显示上的效果，再一次执行代码检查后效果就出来了。

04 重新执行一次代码检查如图 9-37 所示。

05 代码检查结果，如图 9-38 所示。

图 9-37

图 9-38

刚才禁用的Android类下的Android Resource Validation的问题已经没有了（因为原来Android这个类目录下就有一个有问题的检查项，被我们禁用后Android类目录下有问题的检查项是 0，所以Android这个类目也不显示了）。

2. 启用某项检查

01 在检查结果中单击设置，如图 9-39 所示。

图 9-39

02 在设置中查看被禁用的检查项，如图 9-40 所示。被禁用的检查项会高亮显示，所以能够很容易地找到哪些检查项被我们禁用了。

图 9-40

03 勾选要启用的检查项。

04 重新执行代码检查。

9.7.2 在偏好设置中禁用和启用某项检查

操作步骤： 偏好设置→Editor→Inspections。

在这里被禁用的检查也同样会高亮显示，可以通过勾选和取消勾选来启用或禁用检查项，如图 9-41 所示。

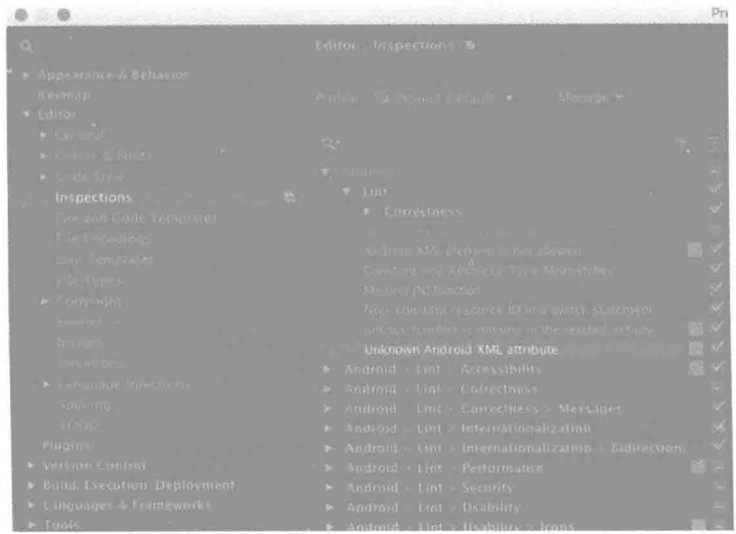


图 9-41

9.8 忽略检查

由于种种原因，我们可能不想对某一段代码或某一个文件进行检查，直接忽略检查这段代码或这个文件就可以了。下面列举两个例子。

【实例演示】

例 1：忽略Android资源验证：Android Resources Validation检查（见图 9-42）。

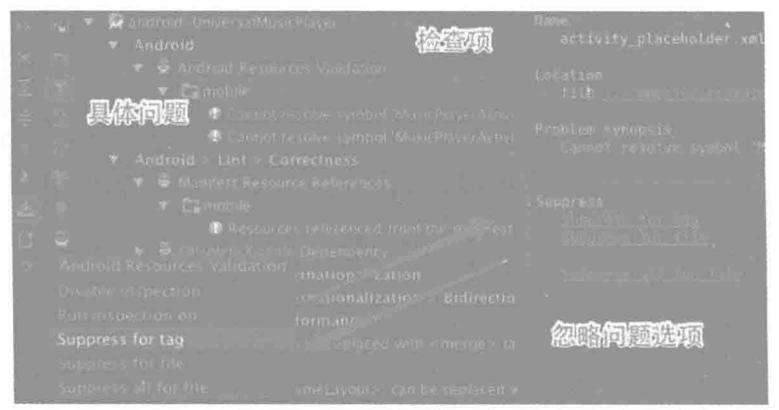


图 9-42

如图 9-42 所示，在【Android Resources Validation】检查项下面选中具体问题，在问题的详情中会显示忽略问题的选项，这个选项的内容和右击问题或单击左边栏的选项是相同的。

（1）单击【Suppress for tag】：忽略【Android Resources Validation】对这个标签的检查，会在要忽略的标签上添加：

```
<!--suppress AndroidDomInspection -->
```

具体情况如图 9-43 所示。

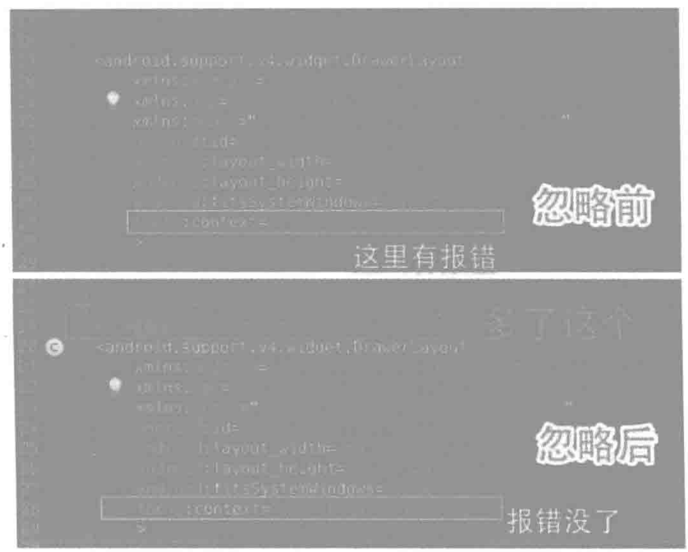


图 9-43

（2）单击【Suppress for file】：忽略【Android Resources Validation】对这个文件的检查，会在要忽略的文件上（<?xml version="1.0" encoding="utf-8"?>下面）添加：

```
<!--suppress AndroidDomInspection -->
```

（3）单击【Suppress all for file】：忽略这个文件的所有检查项，会在要忽略的文件上（<?xml version="1.0" encoding="utf-8"?>下面）添加：

```
<!--suppress ALL -->
```

例 2：忽略使用了过时的API：Deprecate API usage检查（见图 9-44）。

图 9-44

如图 9-44 所示，在【Deprecate API usage】检查项下面选中具体问题，在问题的详情中会显示忽略问题的选项。这个选项的内容和右击问题或单击左边栏的选项是相同的。

（1）单击【Suppress for statement】：忽略【Deprecate API usage】对这个语句的检查，会在要忽略的语句上添加：

```
//noinspection deprecation
```

具体情况如图 9-45 所示。

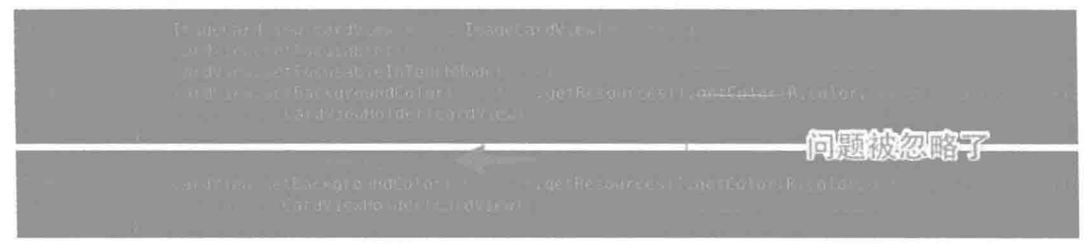


图 9-45

（2）单击【Suppress for member】：忽略【Deprecate API usage】对这个成员（函数）的检查，会在要忽略的成员函数上添加：

```
@SuppressWarnings("deprecation")
```

具体情况如图 9-46 所示。

图 9-46

（3）单击【Suppress for class】：忽略【Deprecate API usage】对这个类的检查，会在要忽略的类上添加：

```
@SuppressWarnings("deprecation")
```

具体情况如图 9-47 所示。

（4）单击【Suppress all inspections for class】：忽略这个类的所有检查项，会在要忽略的类上添加：

```
@SuppressWarnings("ALL")
```

具体情况如图 9-48 所示。

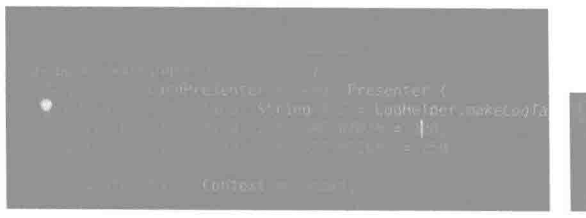


图 9-47

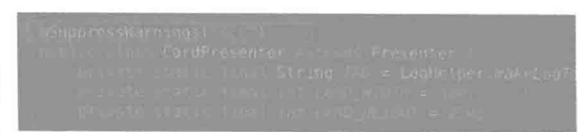


图 9-48

9.9 在指定范围内执行某项检查

如果想查看某一项检查在指定的范围内的检查情况，可以在检查结果中选中这个检查项，再指定检查范围进行检查。

操作步骤：在检查结果中选中一个检查项→右击，执行【Run Inspection on...】→选择检查范围→执行检查。

【实例演示】在“mobile”中执行【Android Resources Validation】检查。

01 选中【Android Resources Validation】。

02 执行【Run Inspection on...】，如图 9-49 和图 9-50 所示。

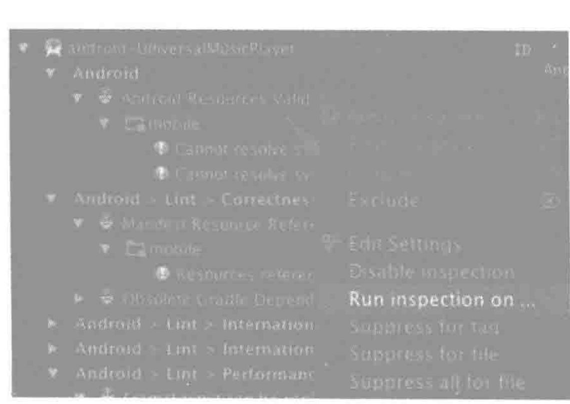


图 9-49

图 9-50

03 选择检查范围 → 执行检查，如图 9-51 所示。

04 查看检查结果，如图 9-52 所示。

图 9-51

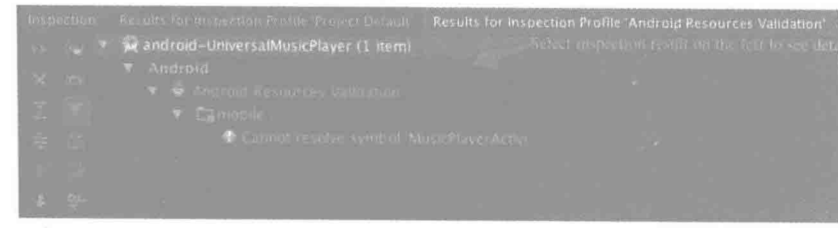


图 9-52

9.10 解决检查出的问题

我们通过一个简单的例子来介绍如何分析并解决检查出来的问题。

第 1 步：搜索无用声明这个检查项，如图 9-53 所示。

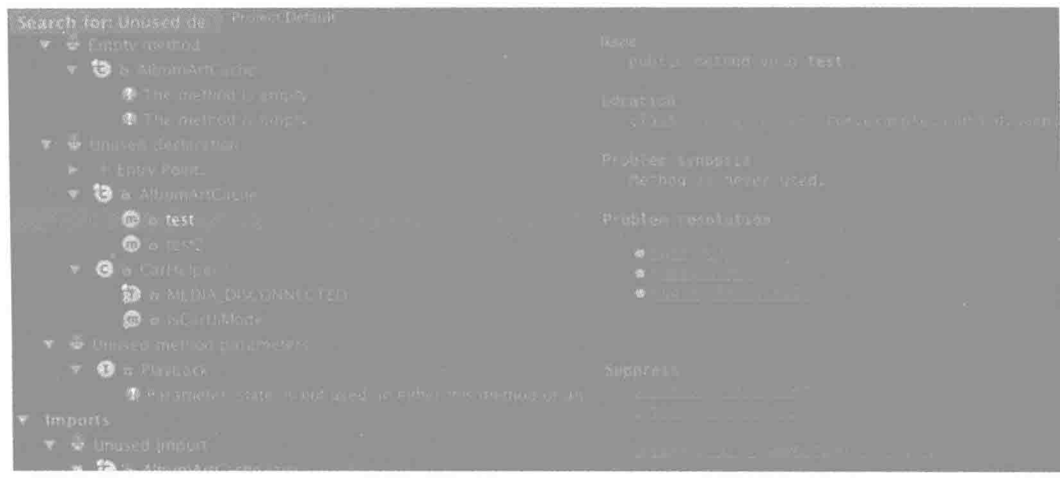


图 9-53

第 2 步：选中具体问题→查看问题详情。

- Name：显示了有问题的方法名是一个公共方法 test()。
- Location：这个方法的位置，单击可打开这个类。
- Problem synopsis：问题摘要，这个方法从未使用。
- Problem resolution：问题解决方案，单击可快速解决。
- Safe delete：安全删除。
- Comment out：注释掉。
- Add as Entry Point：添加为入口点。
- Suppress：忽略检查。
- Suppress for member：忽略成员。
- Suppress for class：忽略类。
- Suppress all inspections for class：忽略类的所有检查，

第 3 步：解决问题。
可以根据解决方案来快速解决问题，如果觉得不是问题也可以忽略对这个方法的检查。
其他问题的解决方法类似，这里就不一一列举了。

9.11 管理代码检查配置文件

我们可以在偏好设置中管理代码检查配置文件。
操作步骤：菜单栏→Android Studio→Preferences→Inspections→Manage，如图 9-54 所示。
配置文件的操作选项如图 9-55 所示。

1. 复制当前配置文件

操作步骤：Manage→Copy→重命名配置文件，如图 9-56 所示。
如果忘记了重命名或命名错了，还可以再次重命名。

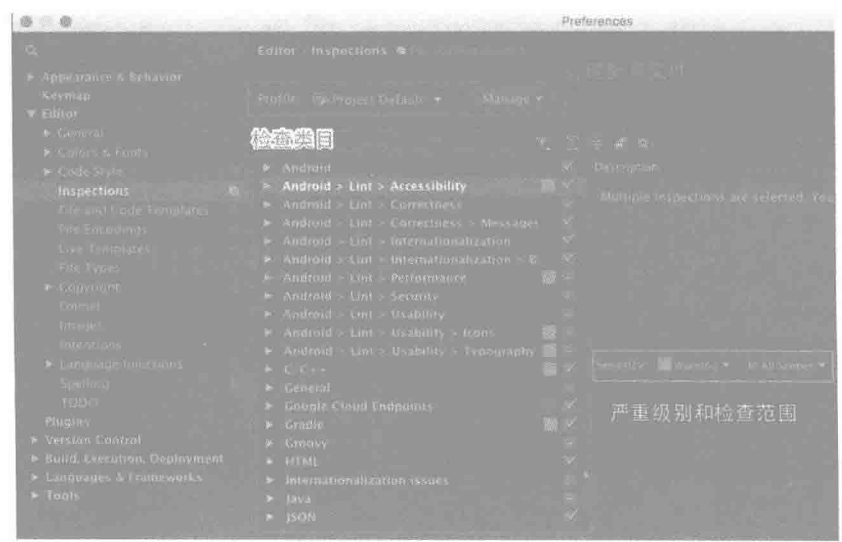


图 9-54

图 9-55

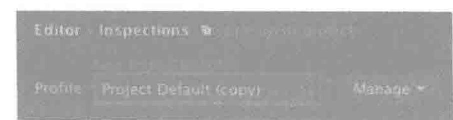


图 9-56

2. 重命名配置文件

操作步骤：Manage→Rename→ 重命名配置文件，如图 9-57 所示。

图 9-57

3. 给配置文件添加描述

如果嫌配置文件不够清楚，还可以给配置文件添加描述。

操作步骤：Manage→Add description→ 输入配置文件的描述，如图 9-58 所示。

图 9-58

这个描述也同样是可以编辑和删除的。

4. 导出配置文件

如果想把配置文件共享给团队成员，可以导出配置文件。

操作步骤：Manage→Export→选择导出路径→会导到一个xml格式的配置文件。

5. 导入配置文件

如果想导入别人共享的配置文件，可以导入配置文件。

操作步骤：Manage→Import→选择要导入的配置文件路径。

9.12 配置代码检查规则

我们可以在偏好设置中配置代码检查规则。

操作步骤：菜单栏→Android Studio→Preferences→Inspections，如图 9-59 所示。

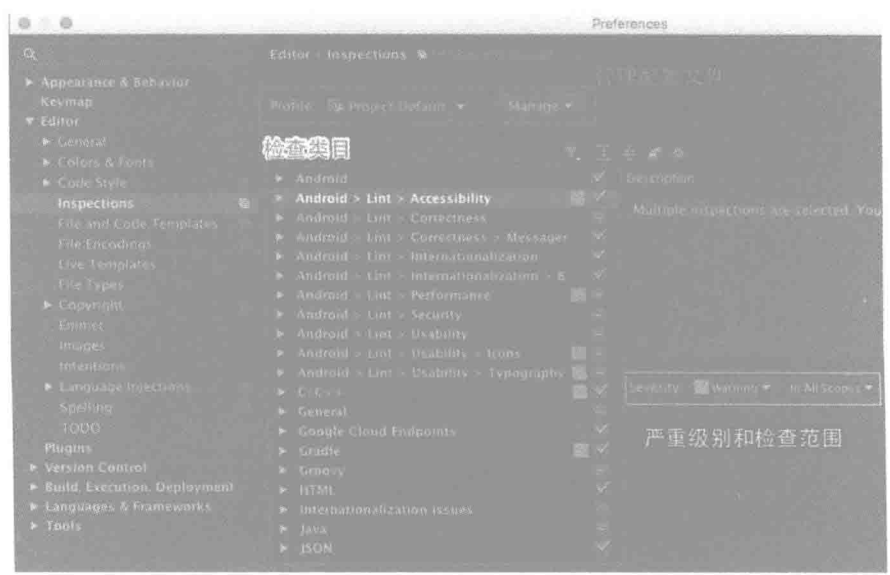


图 9-59

1. 快速搜索某个检查项

如果我们明确知道要对哪一个检查项进行配置，直接搜索就可以了。如果只是模糊地知道某个单词，没关系，这里可以进行模糊匹配，如图 9-60 所示。

2. 过滤某一类检查项

这个功能太好了，可以过滤出某一类检查项进行配置，不用一个一个去找了，如图 9-61 所示。

图 9-60

图 9-61

提 示

过滤规则可多选。

【实例演示】过滤出所有代码清理的检查项。

选中【Show Only Cleanup Inspections】，然后就会过滤出所有代码清理时用到的检查项，包括已勾选和未勾选的，如图 9-62 所示。

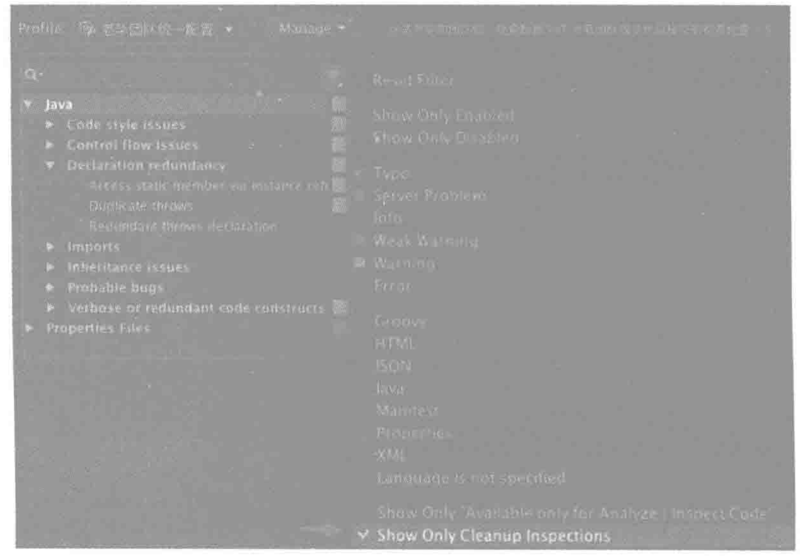


图 9-62

如果只想过滤出已勾选的检查项，就再把【Show Only Enabled】勾选了。如果想取消所有过滤规则，直接单击【Reset Filter】，如图 9-63 所示。

3. 清空所有选中的检查项

清空所有选中的检查项工具（见图 9-64），一下就可以清空所有的检查项（见图 9-65），请慎用。

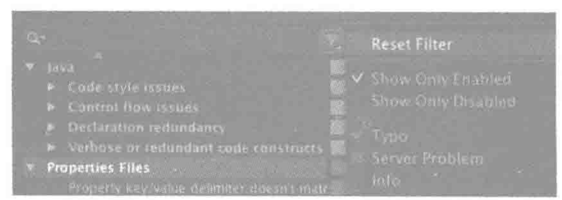


图 9-63

图 9-64　清空所有选中的检查项按钮

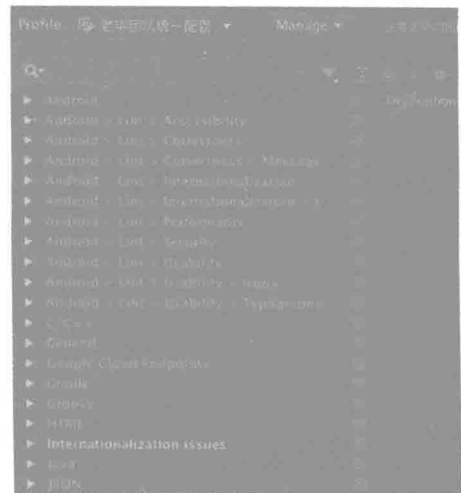


图 9-65　清空所有检查项的结果

4. 默认禁用新的检查和恢复默认设置（见图 9-66）

图 9-66 默认禁用新的检查和恢复默认设置

- 默认禁用新的检查（Disable new inspections by default）：当 Android Studio 更新后可能会有新的检查出现，如果怕出问题可以默认禁用新的检查。
- 恢复默认设置（Reset to Default Settings）：这是后悔药，当设置检查出错时，可以恢复默认设置。

5. 设置检查项的严重级别

我们可以对某个类目或具体的检查项进行设置。

第 1 步：选中某个检查项，如图 9-67 所示。

图 9-67

第 2 步：展开Severity的下拉列表，如图 9-68 所示。默认有 6 种错误级别，从拼写错误一直到严重错误。

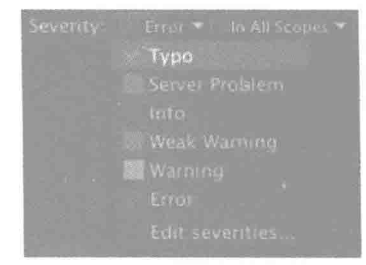


图 9-68

- Typo：拼写错误。
- Server Problem：服务端问题。
- Info：详情。
- Weak Warning：弱警告。
- Warning：警告。
- Error：严重错误。

第 3 步：选择/自定义严重级别。

除了选择默认的 6 种级别外，还可以自定义严重级别。

添加一个新的严重级别

01 单击【Edit severities】→在打开的 Severities Editor 对话框中单击+ →输入自定义严重级别的名字，如图 9-69 所示。

02 定义颜色、字体、特效等，如图 9-70 所示。

03 之后就可以使用自定义的严重级别了，如图 9-71 所示。

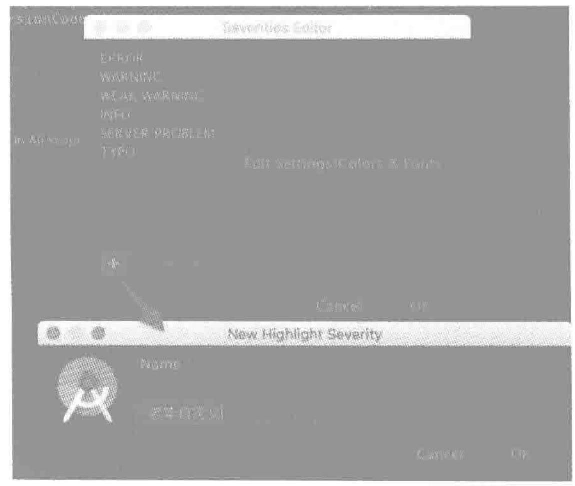


图 9-69

图 9-70

图 9-71

编辑默认的严重级别

除了添加新的严重级别，还可以编辑原来默认的严重级别。

01 单击【Edit severity】 → 在打开的 Severities Editor 对话框中单击【Edit Settings|Colors &Fonts】，如图 9-72 所示。

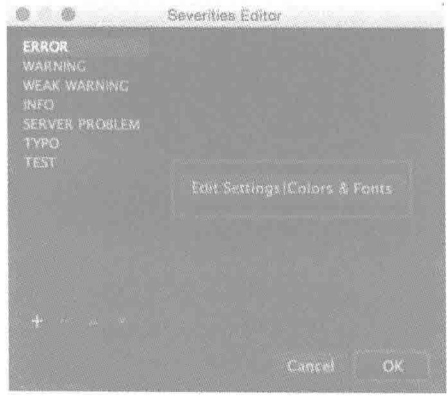


图 9-72

02 进入 Colors &Fonts> General 页面（见图 9-73），在这个页面配置即可。

图 9-73

6. 设置检查项的应用范围

我们还可以灵活配置每一个检查项的应用范围，如图 9-74 所示。

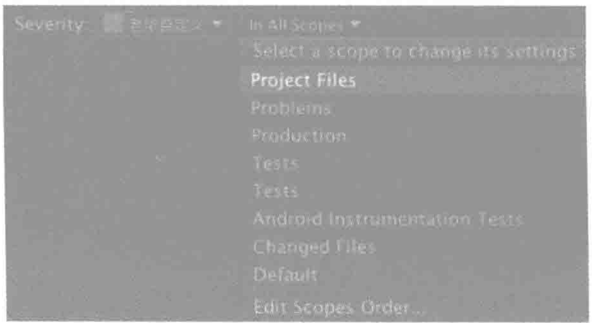


图 9-74

9.13 Android 类目的所有检查项

操作步骤：菜单栏→Android Studio→Preferences→Inspections→展开Android类目（见图 9-75），默认所有检查项全部被勾选。

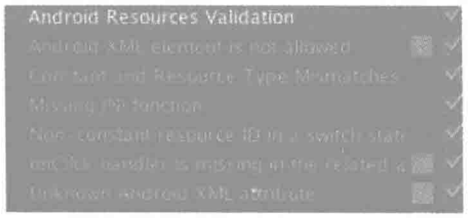


图 9-75

- Android Resources Validation：验证 Android XML 中的资源引用。
- Android XML element is not allowed：验证 Android Resource 文件和 AndroidManifest.xml 中不允许的标签。
- Constant and Resource Type Mismatches：常量和资源类型不匹配。
- Missing JNI function：报告项目中的native方法声明在项目中没有找到相应的JNI函数。
- Non-constant resource ID in a switch statement：检查在 Android library 模块中的 switch 语句是否使用了Resource IDs，因为从SDK Tools r14开始在library项目中Resource IDs就是非 final，这意味着 library 代码不能将这些 id 作为常量。
- onClick handler is missing in the related activity：检查相关的 activity 是否声明了 XML 的 onClick 属性中指定的方法。
- Unknow Android XML attribute：检查并高亮显示 Android Resource 文件和 AndroidManifest.xml 中未知的 XML 属性。

针对这里的每一项检查，我们都可以自定义严重性和范围，也都可以勾选或不勾选。

9.14 Android Lint 类目的检查项

Lint检查所有的Issue可分为 10 大类，Android Studio中的Lint分类，如图 9-76 所示。

图 9-76

每个类别的问题太多，我们有选择性地讲解一下。

1. Accessibility（可访问性）

（1）Image without contentDescription：图片没有 contentDescription 属性。

Image View和ImageButtons 这样的非文本控件应该使用contentDescription属性来描述控件。这样可访问工具（如uiautomatorview）就可以更好地理解控件的用途，从而更加充分地分析用户界面。

注意，contentDescription仅是应用程序屏幕上的装饰，并不会提供任何内容或使用户访问不具有可访问性的内容。

禁用这个警告可使用：

```
ignore="ContentDescription"
```

（2）Missing labelFor attribute：labelFor 属性缺失。

如果minSdkVersion≥17，文本字段应该标有labelFor属性。如果view已经在不同的layout中标记了该属性，需要从Lint中禁用这个警告。

2. Correctness（正确性）

（1）Accidental Octal：意外的八进制。

在Groovy中，以整数 0 开始的数字通常会被理解为一个八进制数，这个不经意的小意外可能会导致一个小BUG ，例如在APP中使用versionCode。

（2）Activity registered more than once：Activity 注册了不止一次。

一个Activity在manifest中只应该被注册一次， 如果不小心注册了多次，就可能会出现一些小错误。

因为两个元素的属性声明是不会合并的，所以可能会不小心删除以前的声明。

（3）AdapterViews cannot have children in XML：在 XML 中 AdapterViews 不能有子对象。

AdapterViews与ListViews一样，必须从Java code中配置数据，例如ListAdapter。

（4）Calling new methods on older versions：在老版本中调用新方法。

（5）Custom views in libraries should use res-auto-namespace：库中自定义的 view 应该使用 res-auto-namespace。

当在库项目中使用自定义属性来定义一个view时，layout必须使用特殊的命名空间 http://schemas.android.com/apk/res-auto替代URI，包括该库项目自己的包。当库资源合并到应用程序项目中时，将会自动调整属性的命名空间。

（6）Cycle in resource definitions：资源循环定义。

资源定义不应该循环，否则会导致运行时异常。

（7）Duplicate definitions of resources：资源重复定义。

可以在不同的资源文件夹中定义相同的资源文件，但是不同在同一资源文件夹中定义相同的资源。

（8）Duplicate ids within a single layout：在一个单独的 layout 中有重复的 id。

一个布局文件中id应该是唯一的，否则用findViewById()会得到一个意料之外的view。

（9）Fragment not instantiatable：Fragment 没有实例化。

每个Fragment都必须有一个空的构造函数，所以当它的activity恢复状态时才能够被实例化。由于fragment被重新实例化的时候带参数的构造函数不会被调用，因此强烈建议子类不要有带参数的构造函数。

（10）Invalid ID declaration：无效的 ID 声明。

一个id的定义格式必须是 @+id/yourname，如果想使用id表示不同的范围，可以在name前加上前缀，例如@+id/login_button1 和 @+id/login_button2。

（11）Manifest Resource References：Manifest 资源引用。

Manifest中的元素能够引用资源，但是这些资源的配置不能改变（除了特殊情况，版本和一些特定的封装属性，如应用程序的标题和图标）。

（12）Reference to an unknown id：引用了一个未知的 id。

（13）WebViews in wrap_content parents：WebView 的父视图是 wrap_content。

如果WebView父视图是wrap_content而不是match_parent，启用时会有一定的性能优化，很可能不能正常工作，这样可能会导致UI问题。

3. Correctness > Messages（正确性 > 信息）

（1）Extra translation：多余的翻译。

如果一个字符串出现在一个特定的语言翻译文件中，但在默认的local（本地化）中没有相应的字符串，那么这个字符串可能是未被使用的。

注意，如果字符串被检查出在任何本地化文件中都没有提供翻译，那么这些字符串很可能导致crash，所以清理它们非常重要。

（2）Incomplete translation：翻译不完全。

（3）Invalid format string：无效格式的字符串。

（4）Missing quantity translation：数量的翻译缺失。

不同的语言对数量进行描述的语法规则也不同。比如在英语里，数量 1 就是一个特殊情况，我们写成“1 book”，但其他任何数量都要写成“n books”。这种单复数之间的区别是很普遍的，不过其他语言会有更好的区分方式，Android支持的全集包括zero、one、 two、few、many和other。

Lint会检查每一个“复数”的翻译，并确保所有的数量字符串是由给定的语言所提供的翻译。例如，一个英语的翻译必须提供一个quantity="one"的字符串；同样，一个捷克语的翻译必须提供一个quantity="few"的字符串。

（5）String.format string doesn't match the XML format string：String.format 字符串不匹配 XML 格式的字符串。

4. Internationalization（国际化）

（1）Byte order mark inside files：字节顺序在文件中的标记。

（2）Encoding used in resource files is not UTF-8：资源文件不是 UTF-8 格式。

（3）Hardcode text：硬编码文本。

硬编码的字符串应该在资源里定义。

5. Internationalization > Bidirectional Text（国际化>双向文本）

Right-to-left text compatibility issues：Right-to-left 文件兼容性问题。

6. Performance（性能）

（1）Obsolete layout params：过时的布局参数。
应该避免使用过时的布局参数。

（2）Using FloatMath instead of Math：使用 FloatMath 替代 Math。

（3）Nested layout weights：嵌套布局 weights。
应该尽量避免嵌套weight，因为那会拖累执行效率。

（4）Unused resources：未被使用的资源
未被使用的资源使程序变大、编译速度降低，应该清除。

（5）Overdraw：过度绘制。

7. Security（安全）

（1）Hardcoded value of android：debuggable in the manifest。
最好不要在manifest中设置android：debuggable属性，如果设置了，在构建debug apk时工具会自动插入android：debuggable=true。
在执行release构建时，会自动设置为false。如果在manifest中指定一个特殊的值，那么工具将会一直使用它。这可能导致调试信息泄漏，造成安全问题。

（2）Packaged private key：打包私钥。
不应该在APP中打包私钥文件。

8. Usability（可用性）

（1）Missing menu title ：缺失菜单标题。

（2）URL not supported by app for Google App Indexing：URL 不支持 Google 的 APP 索引
要确保应用支持 URL，以便从 Google 搜索中获得安装和流量。

9. Usability > Icons（可用性 > 图标）

（1）Clashing PNG and 9-PNG files：PNG 和 9-PNG 文件冲突。

（2）Duplicated icons under different names：在不同的名字下有重复的图标。

（3）Missing application icon：应用程序图标缺失。

（4）Missing density folder：密度文件夹缺失。

10. Usability> Typography（可用性> 字体）

（1）Ellipsis string can be replaced with ellipsis character：省略号字符可以用省略号代替，“...”需要用“…”代替。

（2）Fraction string can be replaced with fraction character：分数字符串可以用分数字符代替，“½”需要用“½”；“¼”需要用“¼”。

（3）Hyphen can be replaced with dash：连字符可以用破折号代替，“–”需要用“–”、“—”需要用“—”。

由此来看，Android Studio中集成了Lint进行代码检查，真是大大方便了配置和使用。

9.15 在 lint.xml 文件中配置 Lint 检查

前面讲到过，我们可以在lint.xml这个配置文件里配置Lint检查项的严重程度和要排除的检查项。正常情况下，我们在Android Studio的偏好设置中修改了Lint的配置，lint.xml文件会被自动添加到Android项目中，如果没有，也可以手动创建一个。基本格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<lint>
        <!-这里添加检查项(issue)的配置 -->
</lint>
```

可以通过运行lint —list得到完整的issue id列表及对应的描述信息。

lint.xml文件配置示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<lint>
    <!-- 禁用 IconMissingDensityFolder 这项检查 -->
    <issue id="IconMissingDensityFolder" severity="ignore" />

    <!-- 在指定的文件中忽略 ObsoleteLayoutParam 检查 -->
    <issue id="ObsoleteLayoutParam">
        <ignore path="res/layout/activation.xml" />
        <ignore path="res/layout-xlarge/activation.xml" />
    </issue>

    <!-- 在指定的文件中忽略 UselessLeaf 检查 -->
    <issue id="UselessLeaf">
        <ignore path="res/layout/main.xml" />
    </issue>
    <!-- 将 HardcodedText 的严重级别设置为"error" -->
    <issue id="HardcodedText" severity="error" />
</lint>
```

9.16 在 Gradle 中配置 Lint 检查

每个Module中都有一个build.gradle文件，如图 9-77 所示。

图 9-77

在build.gradle文件中添加lintOptions的配置，基本格式如下：

```
android {
    lintOptions {
        // 出现 error 时是否停止进程
        abortOnError true
        // 关闭检查项，参数是 issue id
        disable 'TypographyFractions', 'TypographyQuotes'

        // 开启检查项，参数是 issue id
        enable 'RtlHardcoded', 'RtlCompat',  'RtlEnabled'
        // 仅检查指定的检查项，参数是 issue id
        check 'NewApi',  'InlinedApi'
        // 设置检查项的严重等级为可忽略，参数是 issue id
        ignore 'NewApi',  'InlinedApi'
    }
}
```

每次用gradle打包的时候都会默认执行lint检查。如果想在检查出error问题的时候停止打包，就设置abortOnError true；如果想在检查出error问题的时候不停止打包，就设置abortOnError false。

默认的lint会执行所有检查。如果想指定检查某一个或某些检查项，就设置check 'issue id', 'issue id'；如果想指定禁用某一个或某些检查项，就设置disable 'issue id'，'issue id'；如果想开启某一个或某些禁用的检查项，就设置 enable 'issue id'，'issue id'；如果想设置某一个某些检查项的严重级别，就设置：

```
error 'issue id', 'issue id'
fatal 'issue id', 'issue id'
ignore 'issue id', 'issue id'
warning 'issue id', 'issue id'
```

【实例演示】

将InconsistentLayout检查项的严重级别设置为error，如果检查到此问题就立刻停止打包。gradle.build中的设置如下：

```
lintOptions {
    abortOnError true
    error 'InconsistentLayout'
}
```

在项目根目录运行：

```
./gradlew lint
```

运行结果：

```
构建失败！
```

原因分析如图 9-78 所示。

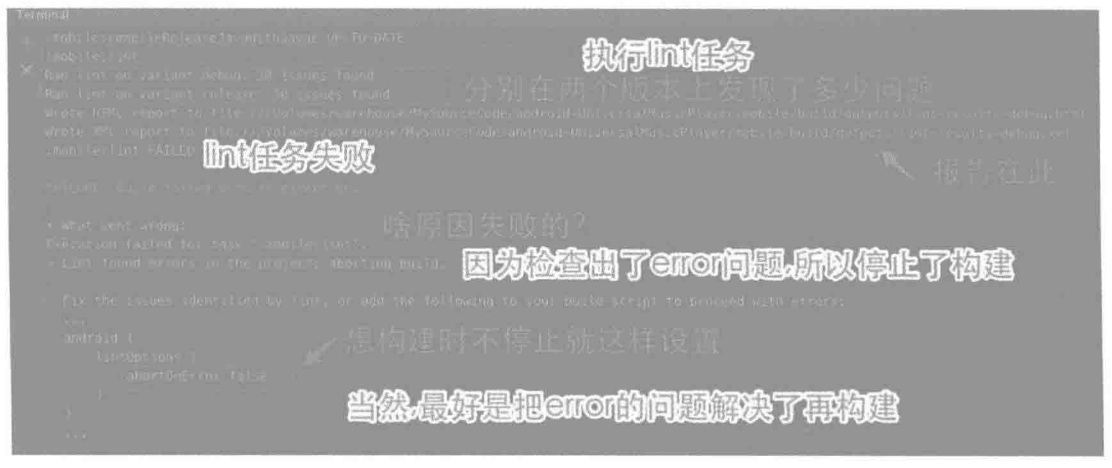


图 9-78

解决例子中构建失败的问题有 4 种方法：一是把error的问题解决了，二是把abortOnError设置为false，三是把检查项的严重级别降低，四是禁用这个检查项。

为了方便我们使用第二种方法：

```
lintOptions {
    abortOnError false
    error 'InconsistentLayout'
}
```

重新执行lint：

```
./gradlew lint
```

这次构建成功了！

9.17 使用 Gradle 执行 Lint 检查

在Android Studio中使用Gradle执行Lint检查有两种方法：一是通过命令行，二是通过Gradle工具窗口。

9.17.1 命令行执行Lint检查

在Android Studio中，可以针对不同的渠道版本来执行Lint检查。

对所有渠道版本执行Lint检查：

```
./gradlew lint
```

对Release版本执行Lint检查：

```
./gradlew lintRelease
```

对Debug版本执行Lint检查：

```
./gradlew lintDebug
```

如果有其他渠道版本，单独执行Lint检查的方法同上。

9.17.2 Gradle工具窗口执行Lint检查

如果要对某个模块进行lint检查，请打开Gradle工具窗口中指定模块的任务目录。

在verification下面找到对应的lint任务，如图 9-79所示。

- 双击 lint：对所有渠道版本执行 lint 检查
- 双击 lintDebug：对 Debug 版本执行 lint 检查
- 双击 lintRelease：对 Release 版本执行 lint 检查

图 9-79

如果有其他渠道版本，单独执行Lint检查的方法同上。

9.18 在 Java 和 XML 源码中配置 Lint 检查

Lint检查还可以在Java和XML源码中直接进行配置，但这里只能配置禁用某项或全部检查。

9.18.1 在Java源码中配置Lint检查

在Java源码中使用注解@SuppressLint（"issue id"） 来禁用检查。

禁用多项检查的参数为一个字符串数组，如@SuppressLint（{"issue id"}，{"issue id"}）。

禁用所有检查的参数为all，如@SuppressLint（"all"） 。

【实例演示】

例 1：在类中禁用NewApi检查。

```
@SuppressLint("NewApi")
public class CastPlayback implements Playback {
//  .......
}
```

例 2：在类中禁用NewApi和ParserError检查。

```
@SuppressLint({"NewApi", "ParserError"})
public class CastPlayback implements Playback {
//  .......
}
```

例 3：在方法中禁用NewApi检查。

```
    @SuppressLint("NewApi")
    @Override
    public void stop(boolean notifyListeners) {
       //.....
    }
```

例 4：在变量中禁用NewApi检查。

```
    @SuppressLint("NewApi")
    private static final String ITEM_ID = "itemId";
```

例 5：在类中禁用所有Lint检查。

```
@SuppressLint("all")
public class CastPlayback implements Playback {
  //.....
}
```

9.18.2 在XML源码中配置Lint检查

在XML源码中使用属性tools:ignore="issue id"来禁用检查。

禁用多项检查：tools:ignore="issue id，issue id"。

禁用所有检查：tools:ignore="all"。

提 示

如果想使用 tools: ignore 属性，就必须在 XML 的命名空间上添加 xmlns: tools="http://schemas.android.com/tools"，如图 9-80 所示。

图 9-80

【实例演示】

例 1：禁用RelativeLayout元素中的UnusedResources检查（见图 9-81）。

图 9-81

如果没有在命名空间上添加xmlns:tools="http://schemas.android.com/tools"，那么在使用tools:ignore属性的时候会提示添加。

提 示

禁用 RelativeLayout 元素中的 UnusedResources 检查时，子元素也会禁用这个检查。就相当于禁用了类中的某项检查，类中的方法、变量也自动禁用那项检查。

例 2：禁用ImageView中的UnusedResources检查，如图 9-82 所示。

图 9-82

例 3：禁用RelativeLayout中的所有Lint检查，如图 9-83 所示。

图 9-83

9.19 代码清理

虽然可以通过代码扫描来检查出很多问题，但是不得不逐个地手动去修复，如果有个工具可以把检查出来的常见问题自动修复那该多好。Android Studio恰好就提供了这么一个功能：Code Cleanup（代码清理）。我们可以自定义代码清理的规则，以此来帮助我们快速修复问题。

1. 清理代码

操作步骤：菜单栏→Analyze→Code cleanup→指定代码清理范围→执行清理。

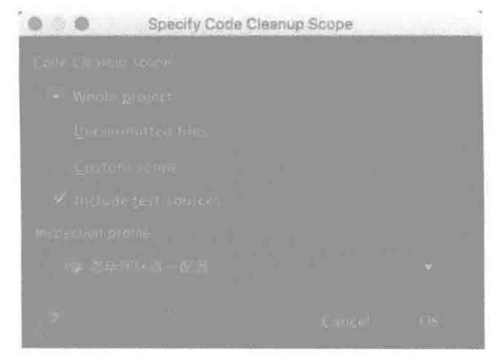


图 9-84

【实例演示】对整个项目执行自动清理。

01 菜单栏→Analyze→Code cleanup。

02 指定代码清理范围，如图 9-84 所示。

03 单击【OK】按钮。

代码清理前后的对比如图 9-85 所示。

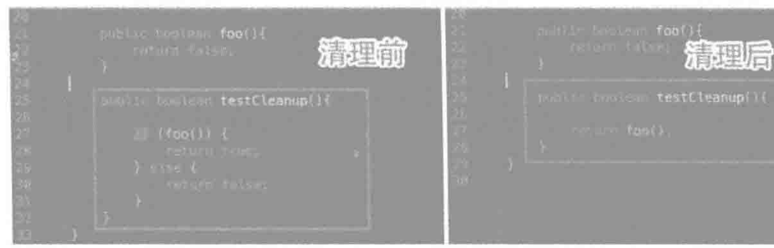


图 9-85

从例子中可以看出，代码清理帮我们把不规范的代码（或者可以有更好的写法）规范了。

2. 配置代码清理规则

操作步骤：菜单栏→Analyze→Code cleanup→单击Inspection profile右边的...可以打开代码清理检查配置界面，如图 9-86 所示。

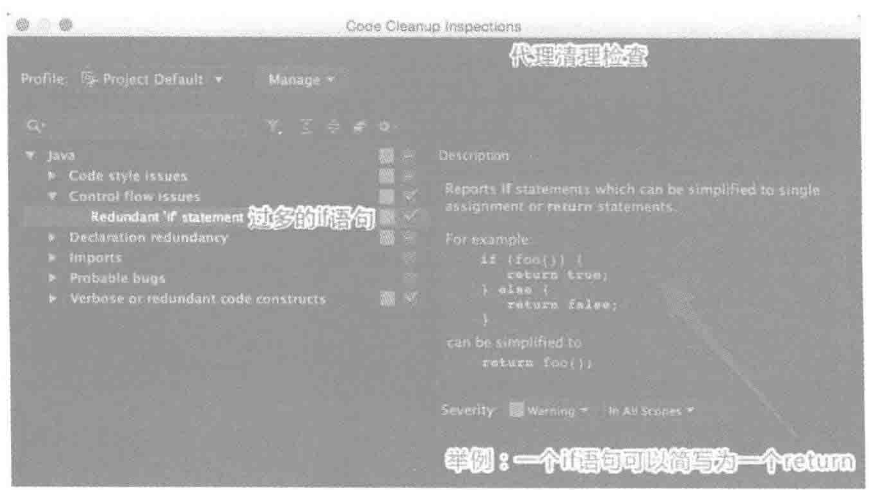


图 9-86

我们可以在这里禁用或启用一些规则、查看规则详情或导入导出规则……确定执行后Code cleanup就会自动检测出问题，并自动修正。Code cleanup还可以在版本控制的commit对话框中设置（后面讲到版本控制的时候会提到）。

9.20 通过名字来指定代码检查项

Android Studio可以通过输入具体检查项的名字来指定执行代码检查。

菜单栏或右击文件：Analyze→Run Inspection by Name

快捷键：option + command + shift + I（macOS）或者Ctrl + Alt + Shift + I（Windows/Linux）

通过快捷操作后弹出规则输入框，接着输入规则名（例如memory）→自动联想出相关的检查规则，如图 9-87 所示。选择检查项之后会弹出检查范围选择窗口，选择后就执行检查了。

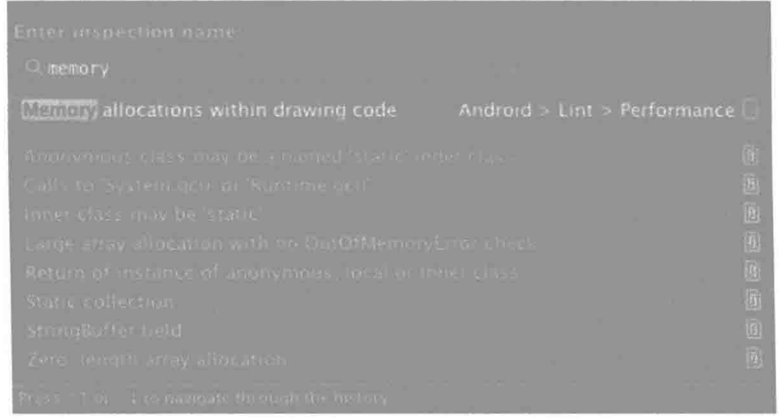


图 9-87

【**实例演示**】检查项目中无用的资源文件。

01 菜单栏→Analyze→Run Inspection by Name→弹出规则输入框。

02 输入规则名（unused resources）并选择，如图 9-88 所示。

03 选择检查整个项目，如图 9-89 所示。

图 9-88

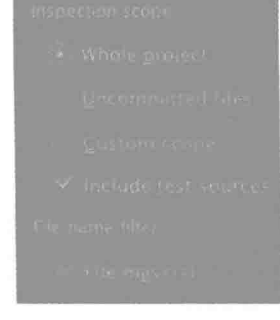


图 9-89

提 示

在指定范围的对话框中还可以通过 File mask 来过滤文件类型。

检查结果如图 9-90 所示，可以根据检查结果决定是否要清除没用过的资源。

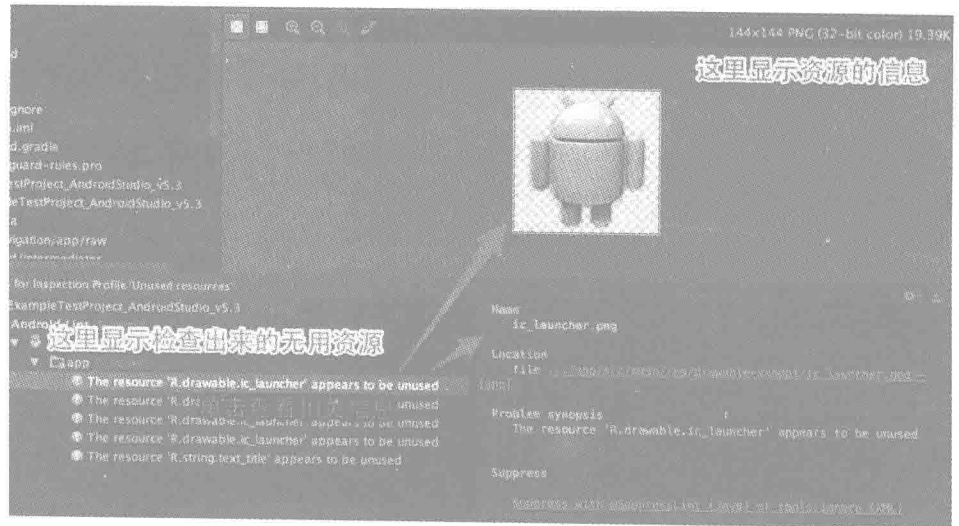


图 9-90

9.21 配置当前文件自动检查的规则

Android Studio会对当前文件自动进行代码检查，检查出问题后相关的代码会被高亮显示。当把鼠标放在高亮显示的地方时会有相关的问题提示，单击more...可以查看详情（见图 9-91）。

图 9-91

对文件自动检查的规则（级别）有三个，如图 9-92 所示。

图 9-92

- None：没有（没有任何检查）。
- Syntax：语法 （仅检查语法）。
- Inspections：代码检查（代码检查里所有选中的规则都会被自动检查）。

提 示

此功能仅是对当前文件进行配置，也只对当前文件有效。

可利用下面几种方式打开配置窗口（见图 9-93）。

菜单栏或右击文件：Analyze →Configure Current File Analysis...

快捷键：option + command + shift + H（macOS）或者Ctrl + Shift + Alt + H（Windows/Linux）

状态栏：单击状态栏的按钮

在配置窗口中还有两个可选项，即Power Save Mode（省电模式）和Import popup（弹出导入提示）。

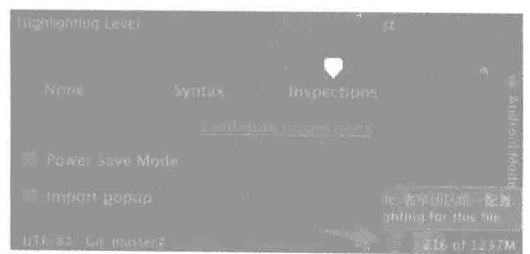


图 9-93

【实例演示】

例 1：当前文件检查的级别为默认的Inspections，如图 9-94 所示。

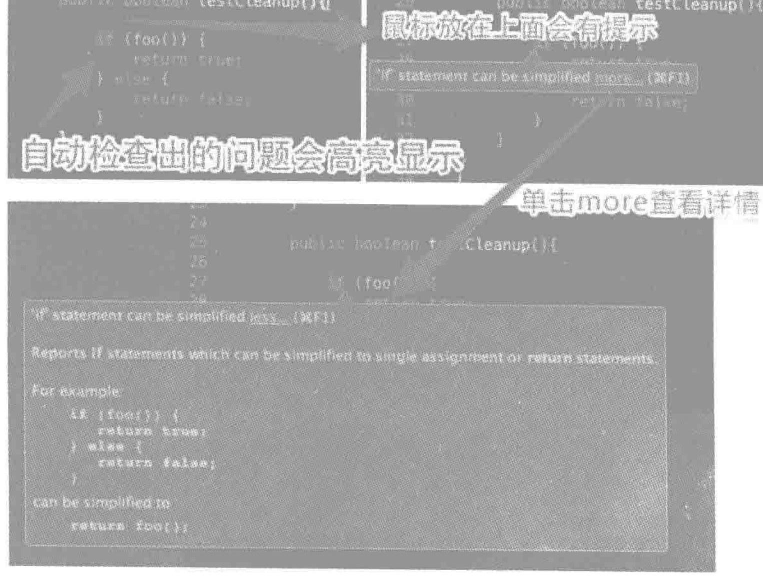


图 9-94

如果当前文件的检查level为None，就不会有这些提示。

例 2：勾选Power Save Mode（见图 9-95），Android Studio会去掉所有自动完成的功能，比如代码提示、代码检查等。

- 在代码编辑器中输入代码，不会有代码提示。
- 随便注释掉一段代码，编辑器也不会有报错提示。

例 3：勾选Import popup（见图 9-96），效果如图 9-97 所示。

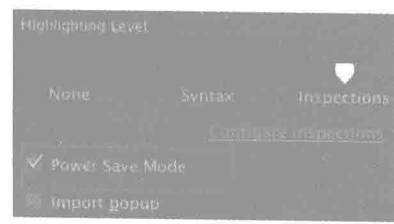


图 9-95

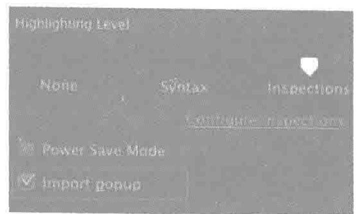


图 9-96

默认Import popup是被勾选的，如果取消掉勾选，导入的提示就不会显示了。

图 9-97

9.22 导入并查看离线检查结果

1. 导出代码检查结果

Android Studio中代码检查结果是可以导入导出的，前面提到了导出结果的功能，这里做个演示（见图 9-98）。

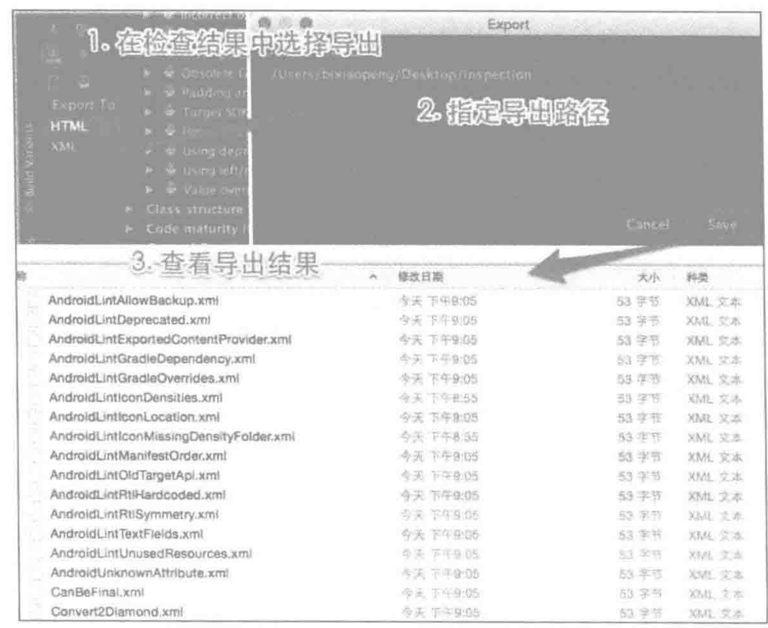


图 9-98

01 在代码检查结果窗口选择导出。

02 指定导出路径。

03 查看导出结果。

可以方便地将这些导出的文件打包共享给有需求的同学。

2. 导入检查结果

拿到这些文件的同学可以通过下述操作打开检查结果。

操作步骤：菜单栏→Analyze→View Offline Inspection Results，弹出选择对话框（见图 9-99），打开检查结果文件（见图 9-100），然后就可以分析并解决别人共享的问题了。

图 9-99

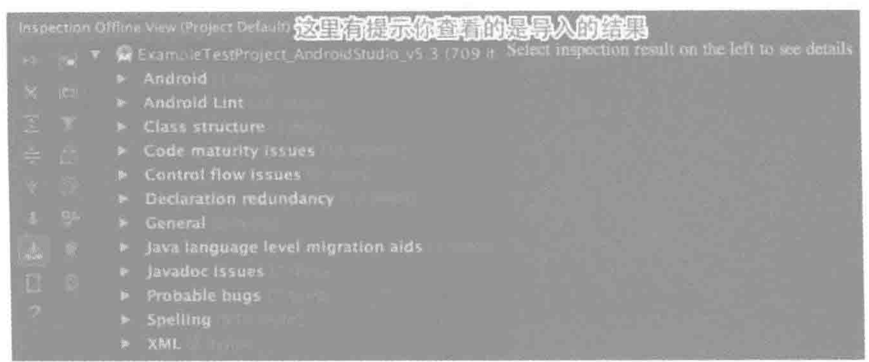


图 9-100

9.23 自动添加是否可为空注解

为什么要使用@Nullable和@NotNull?

在写程序的时候可以定义变量和参数是否可为空指针，可以通过使用@NotNull和@Nullable之类的annotation（注解）来声明一个方法是否是空指针安全的。这两个注解的作用是为了约束。

- @NotNull：变量或参数不能为 null。
- @Nullable：变量或参数可以为 null。

如果忘记判断空指针了，代码检查工具（findbugs）或IDE（Android Studio）可以通过读取此注解来添加忘记的空指针检查，或者提示不必要的空指针检查。当看到@NotNull和@Nullable时，我们可以自己决定是否做空指针检查。

Android Studio的Analyze（代码分析）菜单中提供了Infer Nullity，这个功能可以通过分析代码推断出需要判断空指针的变量和参数，并且可以自动加上@Nullable和@NotNull注解。如果想体会这个功能的好，就必须习惯使用注解。

操作步骤：菜单栏或右击→Analyze→Infer Nullity，弹出检查范围选择对话框，如图 9-101所示。

图 9-101

【实例演示】

01 菜单栏→Analyze→Infer Nullity。

02 弹出检查范围选择对话框→选择 Current File→OK。检查结果如图 9-102 所示。

在 Find 工具窗口中会显示检查的结果。

如果对结果不做任何操作，直接单击【Infer Nullity Annotation】按钮会自动为所有检查结果添加注解。当然我们最好自己查看下结果。如果觉得某些检查结果没必要添加注解，可以在结果中排除，如图 9-103 所示。排除以后再执行【Infer Nullity Annotation】就会忽略排除的检查结果。

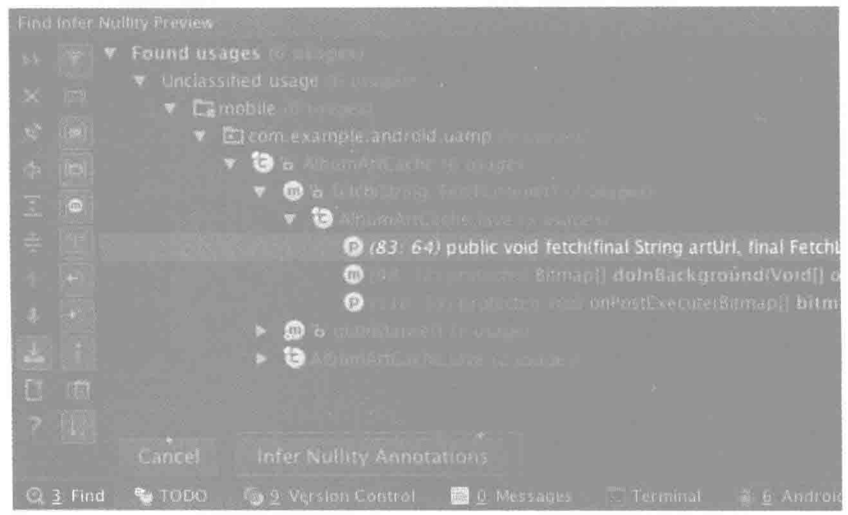


图 9-102

图 9-103

03 这里单击【Infer Nullity Annotation】为所有检查结果添加注解。加上注解的变量和参数如图 9-104 所示。此时再执行【Infer Nullity】就不会有结果显示了。

图 9-104

9.24 分析依赖

依赖是面向对象系统中类与类或组件的一种基本关系。例如，A类在编译时要用到B类，就称A在编译时依赖B。当我们想分析某个文件依赖哪些文件时，可以使用Android Studio提供的分析依赖工具。

菜单栏或右击文件：Analyze→Analyze Dependences

快捷键：Shift + Command + A（macOS）或者Ctrl + Shift + A→Analyze Dependences（Windows/Linux）

通过上述快捷操作后弹出指定分析范围对话框，如图 9-105 所示。

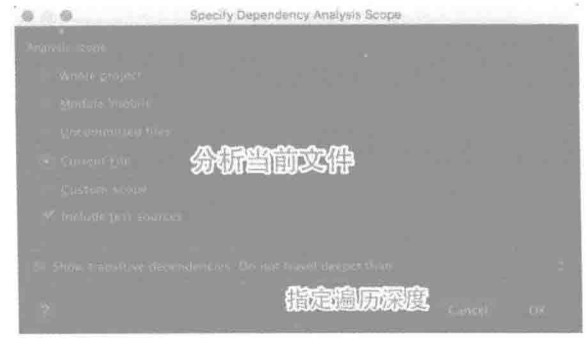


图 9-105

【实例演示】分析AlbumArtCache.java依赖的文件（见图 9-106）。

01 打开 AlbumArtCache.java 文件→菜单栏：Analyze → Analyze Dependences。
02 在弹出的指定分析范围对话框中选择 Current File。
03 OK → 在 Dependency Viewer 工具窗口显示分析结果。

图 9-106

结果分析：

- 选中被分析的类文件 → 在右边会显示这个类文件所依赖的文件，包括项目中的文件和外部文件。
- 单击右边依赖的文件会在下面显示这个文件在左边类文件中的用法。
- 查看依赖文件的用法，更好地理解文件结构。

9.25 分析反向依赖

顾名思义，反向依赖就是分析出当前文件被哪些文件所依赖。

操作步骤：菜单栏或右击文件→Analyze→Analyze Backward Dependences，然后弹出指定分析范围对话框，如图 9-107 所示。

【实例演示】分析依赖AlbumArtCache.java的文件。

01 打开 AlbumArtCache.java 文件→菜单栏：Analyze → Analyze Backward Dependences。

02 在弹出的指定分析范围对话框中选择Current File。

提示

如果是大型项目，需要考虑指定分析用法的范围（Scope to Analyze Usage in），因为默认分析整个项目的文件，那样会很慢。

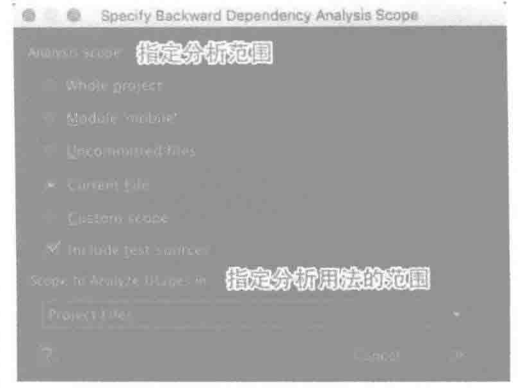


图 9-107

03 OK → 在 Dependency Viewer 工具窗口显示分析结果，如图 9-108 所示。

图 9-108

9.26 分析模块依赖

模块依赖分析（Module dependencies analysis）显示指定范围内存在的所有模块和它们之间的关系。

操作步骤：菜单栏或右击文件→Analyze→Analyze Module Dependences，然后弹出指定分析范围对话框（见图 9-109），再指定是分析整个项目还是某个模块。

分析效果如图 9-110 所示。（模块的依赖关系是：选中module C，显示出A依赖B，B依赖C。）

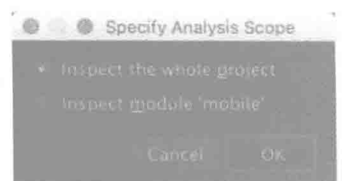


图 9-109

图 9-110

9.27 分析循环依赖

循环依赖就是循环引用，是两个或多个类相互持有对方，比如A引用B，B引用C，C引用A，则它们最终成为一个环。

操作步骤： 菜单栏或右击文件→Analyze→Analyze Cyclic Dependences，弹出指定分析范围对话框，如图 9-111 所示。

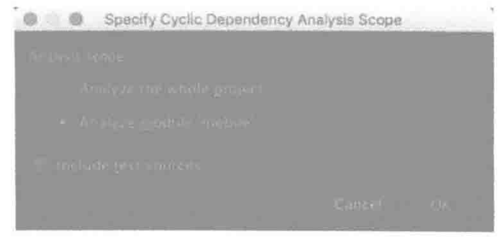


图 9-111

分析效果如图 9-112 所示。

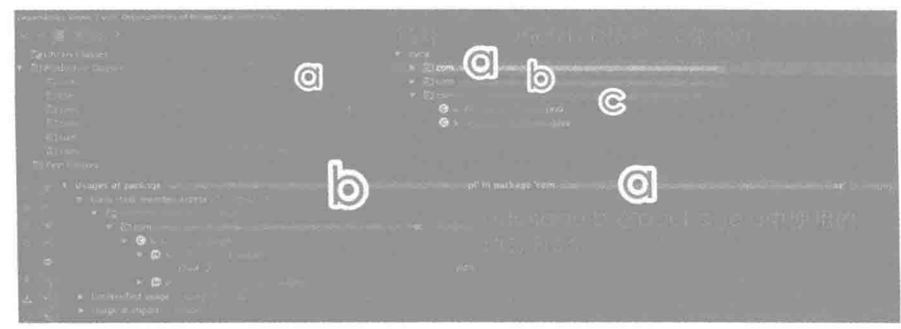


图 9-112

9.28 分析数据流

想更好地了解项目代码的继承时，分析数据流可以逐步查看复杂代码并找到代码瓶颈。

使用数据流分析功能可以查看变量被分配的值从哪里来的、找出变量所有可能的值、找出表达式/变量/方法参数可以流入哪些地方以及找出潜在的NullPointerException可能出现的地方。

前提条件：光标放在变量/字段/参数上。

分析数据流就像思考人生的两个问题：我从哪来，我要到哪去?

（1）我从哪来

操作步骤： 菜单栏或右击变量→Analyze→Analyze Data Flow to here，然后会弹出分析范围选择对话框，确定范围后查看分析结果，如图 9-113 所示。

图 9-113

（2）我要到哪去

操作步骤：菜单栏或右击变量→Analyze→Analyze Data Flow From here，然后会弹出分析范围选择对话框，确定范围后查看分析结果，如图 9-114 所示。

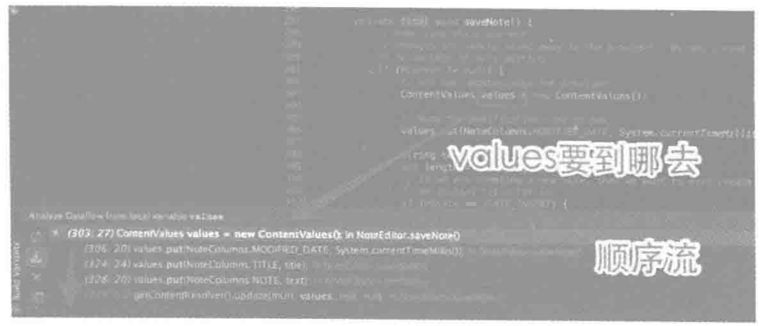


图 9-114

9.29 分析堆栈信息

当我们想更加方便和专注地分析堆栈信息的时候可以使用此功能。

前提条件：复制分析堆栈信息。

菜单栏：Analyze →Analyze StackTrace

快捷键：Shift + Command + A（macOS）或者Ctrl + Shift + A（Windows/Linux）→Analyze StackTrace

【实例演示】

假设我们的应用在测试的时候出现了CRASH，需要对方提供相关日志给我们来分析，拿到日志以后就可以使用此功能来快速地分析问题了。

操作步骤：复制log→菜单→Analyze→Analyze StackTrace→弹出Analyze Stacktrace对话框（见图 9-115）。

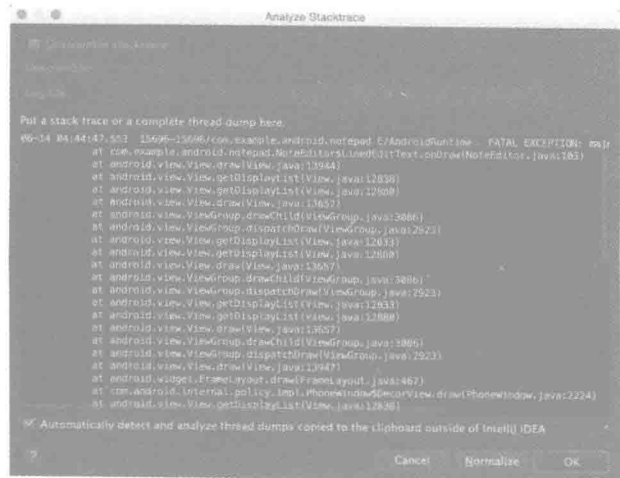


图 9-115

单击【OK】按钮确定后，跳转到分析堆栈的对话框。这里提供了很多方便分析的功能，如图9-116所示。

图9-116

第 10 章　重　构

重构是指在不改变外部行为的条件下，对现有的代码进行改善，以增加可读性，使设计和逻辑更加清晰。Android Studio的重构功能可以帮助我们安全、简单、快速地完成代码重构。

本章向大家介绍如何使用Android Studio快速重构代码。

本章重要知识点 >>>>>>>>>>

- 如何重命名类、变量、文件;
- 如何移动、删除、复制方法和类;
- 如何提炼常量、字段、参数、方法、委托、接口等。

10.1　重命名

重命名可能是我们重构时最常用的功能了，当我们发现Java元素的命名不清晰或原来的功能已发生改变时，就需要通过重命名重构来保持代码的可读性。

需要对类、方法、字段、变量、包或其他文件进行重命名时，可以按如下方法进行操作。

菜单栏：Refactor→Rename

快捷键：fn + shift + F6（macOS）或者Shift + F6（Windows/Linux）

10.1.1　重命名类

将光标定位在类名上→按快捷键fn + shift + F6（macOS）出现重命名选项（见图 10-1），选中后弹出重命名选择对话框（见图 10-2）。

```
public class NotePadTest extends ActivityInstrumentationTestCase2<NotesList> {
             notePadTest
    private  notepadtest
             NotePadTest
    public NPress ⇧F6 to show dialog with more options
        super(NotesList.class);
```

图 10-1

提　示

如果想要更多选项就再按一次 fn + shift + F6。

重命名对话框也可以快速通过按两次快捷键fn + shift + F6 获得。为了谨慎起见，这里不要直接单击【Refactor】，可以先预览一下。单击【Preview】，查看结果，如图 10-3 所示。

我们可以对查找结果进行修改，一些不需要重命名的地方可以排除。

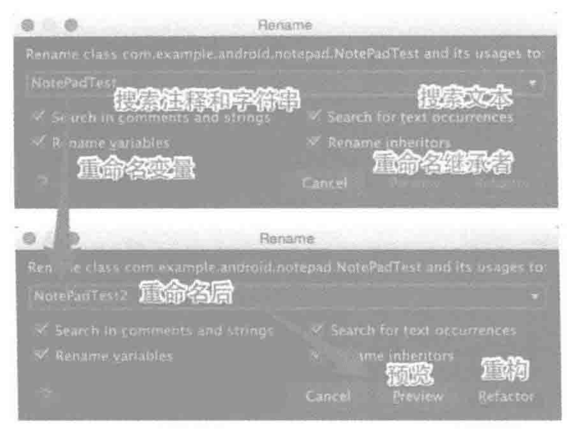


图 10-2

图 10-3

10.1.2 重命名变量

重命名变量（见图 10-4），方法同上。

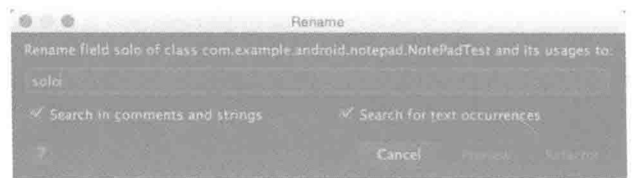


图 10-4 重命名变量

10.1.3 重命名文件

选中文件→按快捷键fn + shift + F6（macOS）→弹出重命名对话框→输入新的文件名，如图 10-5 所示。

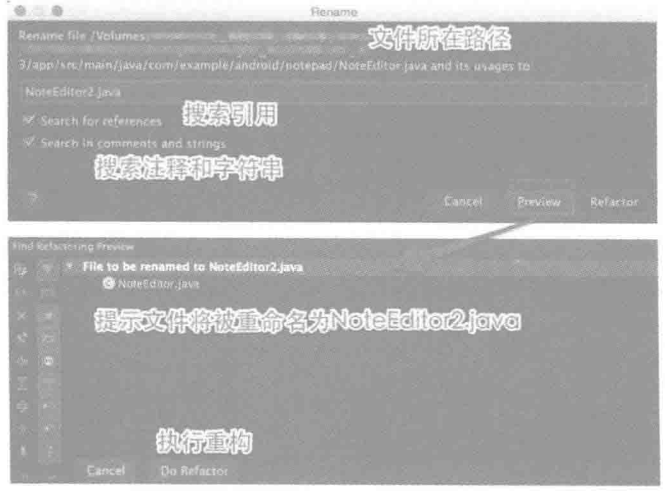


图 10-5

确定后执行重构，所有相关的文件名将会被更改。

10.2 更改方法签名

方法签名由方法名称和一个参数列表（方法的参数的顺序和类型）组成。更改方法签名可以改变方法的名称，改变函数的可访问性，对参数进行添加、删除、重命名和重新排序。更改方法签名的方法如下。

菜单栏：Refactor →Change Signature

快捷键：fn + command + F6（macOS）或者Ctrl + F6（Windows/Linux）

【实例演示】

01 将光标定位在方法名上→按快捷键fn+ command + F6（macOS）→弹出对话框，如图 10-6 所示。

02 可以改变方法的可访问性、对参数进行增、删、改。

03 在最下面可以对更改实时预览。

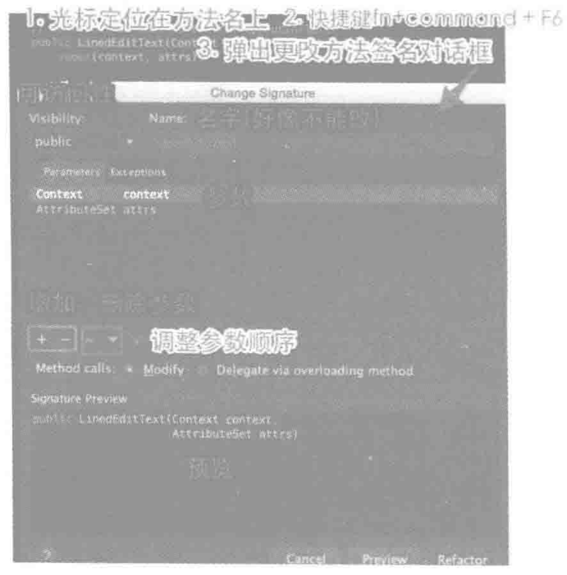


图 10-6

10.3 迁移变量类型

迁移类型（Type Migration）就是改变变量的类型，可如下操作。

菜单栏：Refactor →Type Migration

快捷键：fn+ shift + command + F6（macOS）或者Ctrl + Shift + F6（Windows/Linux）

【实例演示】将变量的类型由String变为int。

有两个文件Book.java和BookStore.java，如图 10-7 所示。

01 将光标放在变量类型上面。

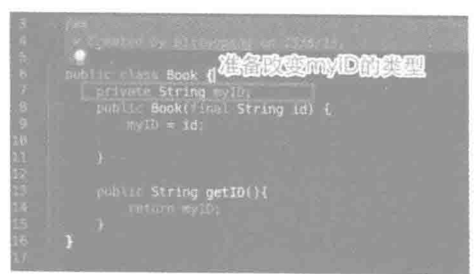
02 按快捷键 fn+ Shift + command + F6（macOS）。

03 弹出迁移类型对话框。

04 输入要改变的类型“int”，并选择变更范围（默认是整个项目文件）。

05 单击预览。

整个步骤如图 10-8 所示。




```
public class BookStore {

    private List<Book> myBooks = new ArrayList<>();
    public void addBook(String id){
        myBooks.add(new Book(id));

    }

    public Book findBook(String id){
        for (Book book: myBooks){
            if (book.getID().equals(id)){
                return book;
            }
        }

        return null;
    }
}
```

图 10-7

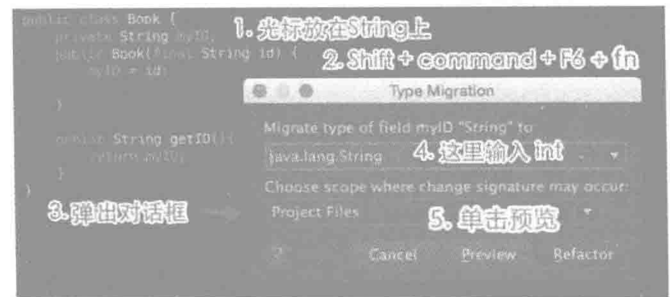


图 10-8

查看预览结果，如图 10-9 所示。

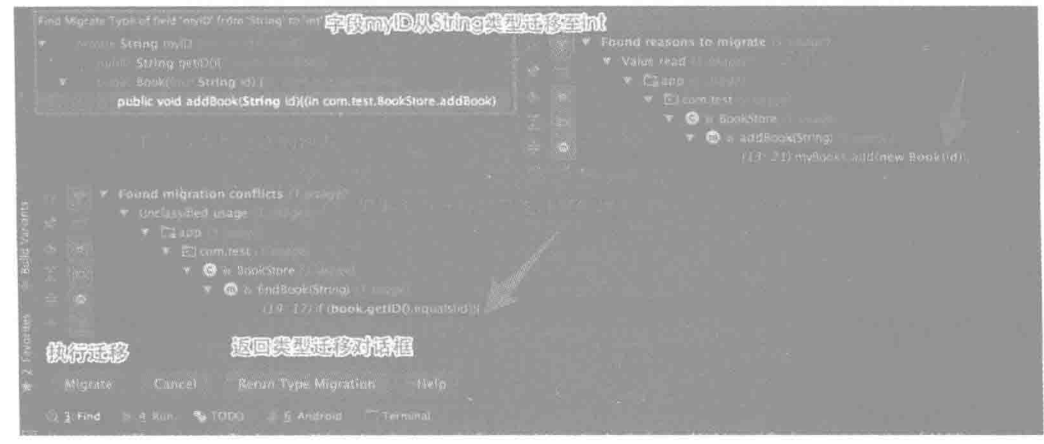


图 10-9

类型迁移前后效果对比如图 10-10 所示。

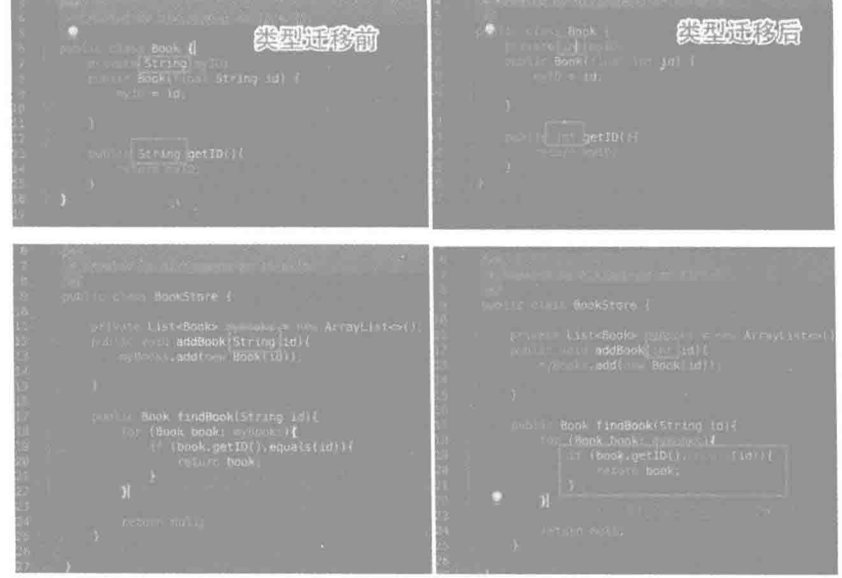


图 10-10　类型迁移前后对比

10.4 转成静态方法

如果想把实例方法转成静态方法，并且能够自动校正所有调用、实现、重写该方法的地方，可以如下操作。

操作步骤：菜单栏→Refactor→Make Static。

【实例演示】把方法getID重构为静态方法。

01 将光标放在你要转成静态的方法上面。

02 菜单栏→Refactor→Make Static→弹出对话框，如图 10-11 所示。

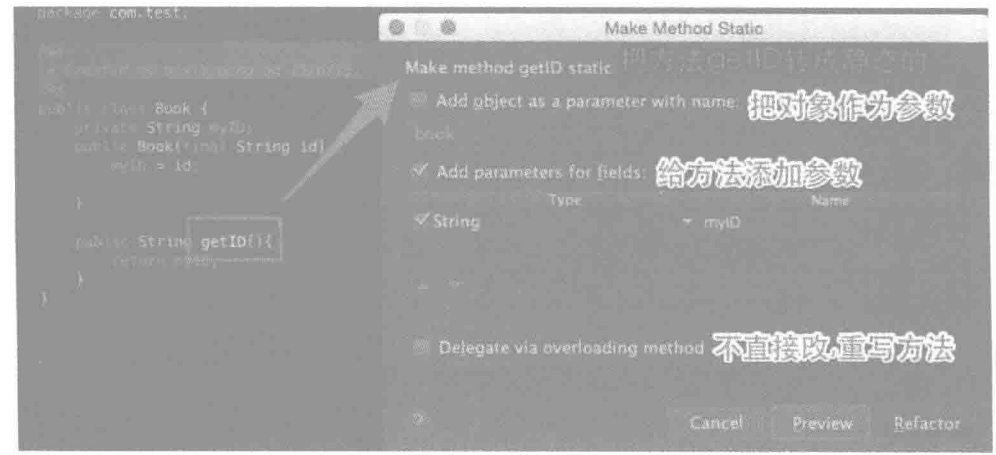


图 10-11

03 单击预览（Preview），如图 10-12 所示。

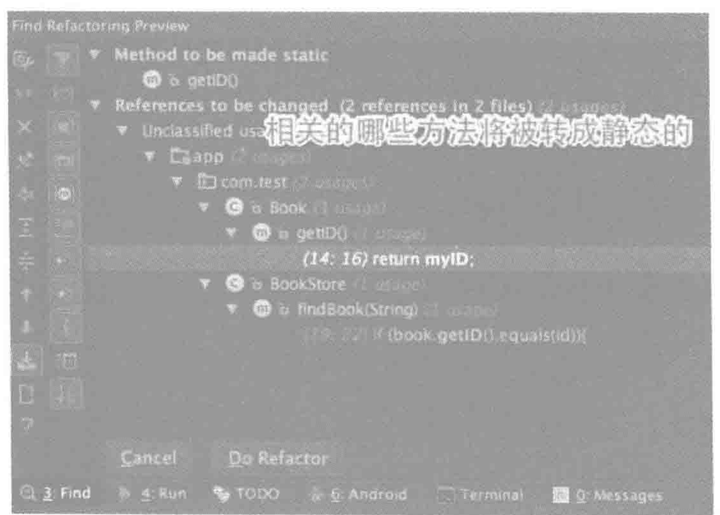


图 10-12

04 单击【Do Refactor】按钮，转成静态方法，前后对比如图 10-13 所示。

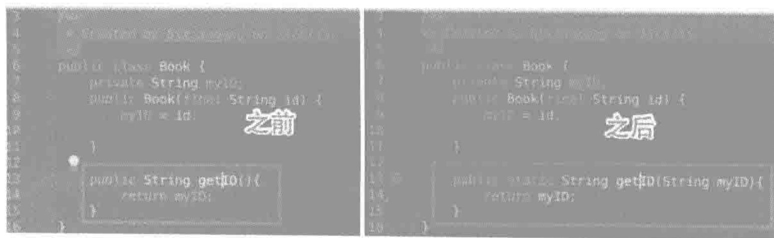


图 10-13 转成静态方法前后对比

如果不想直接修改方法，可以重写方法，然后转成静态的，方法如图 10-14 所示。

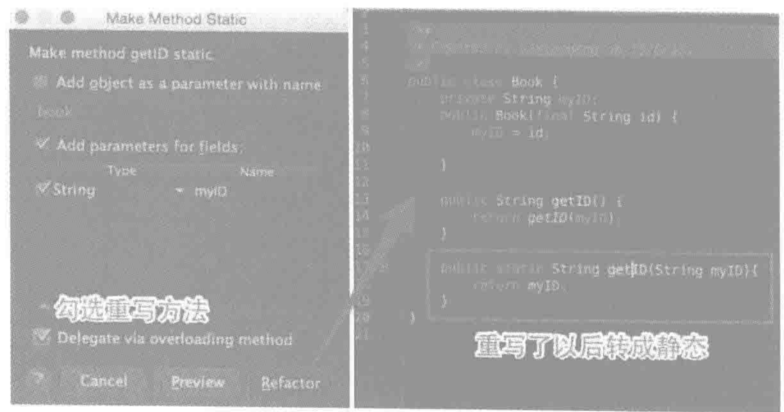


图 10-14

10.5 静态方法转为实例方法

如果想把静态方法转成实例方法，应该如何操作呢？

前提条件：该方法是静态的，它的参数类型是项目中的类。

操作步骤：菜单栏→Refactor →Convert to Instance Method。

【实例演示】将MainTest这个类中的initMethod静态方法转成实例方法。

01 将光标定位在静态方法 initMethod 上，如图 10-15 所示。

02 菜单栏→Refactor→Convert to Instance Method→弹出对话框，如图 10-16 所示。

```
public class MainTest {

    TestA testA = new TestA();
    TestB testB = new TestB();

    public static void initMethod(TestA testA, TestB testB){
        System.out.println("testA = " + testA);
        System.out.println("testB = " + testB);
    }

    public void test(){
        MainTest.initMethod(testA, testB);
    }
}
```

图 10-15

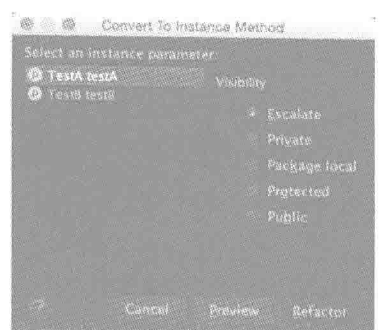


图 10-16

- Select an instance parameter: 选择一个实例参数，转换后的实例方法将会放到这个类里，所有用到这个类的方法都将会被替换。
- Visibility: 可见性。

03 单击预览，查看要改动的地方，如图 10-17 所示。

图 10-17

可以对要改动的地方进行修改。

04 执行重构，结果如图 10-18 所示。

图 10-18

从结果中可以看出initMethod方法用一个类的实例方法作为参数的类型。

10.6 移动类

移动类的重构主要是把一个类移动到不同的包下。要移动类时，最简单的方法是直接把类拖动到目标地址，当然还有我们下面要讲的方法。

前提条件： 光标放在类名上（可多选）或文件编辑界面的任意空白处。

菜单栏： Refactor→Move...

快捷键： fn + F6（macOS），有可能会跟系统快捷键冲突或者F6（Windows/Linux）

利用菜单栏或快捷键操作后会弹出配置对话框，如图 10-19 所示。

单击【Refactor】就会执行重构操作。

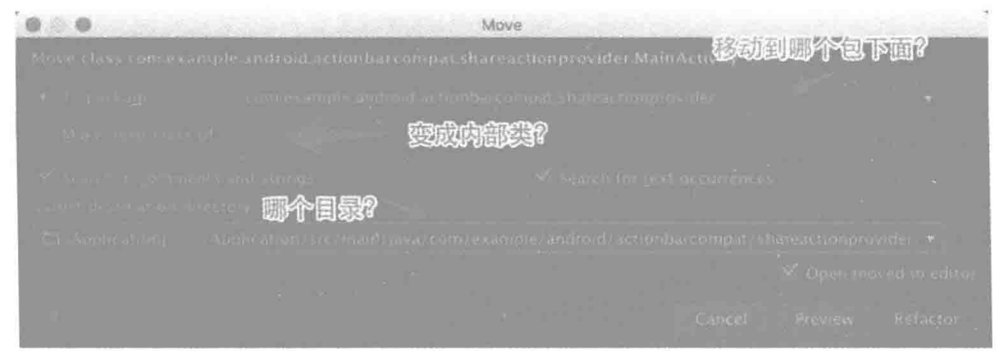


图 10-19

10.7 移动静态方法

前提条件：将光标放在静态方法名上。

操作步骤：

菜单栏：Refactor→Move...

快捷键：fn + F6（macOS）有可能会跟系统快捷键冲突或者F6（Windows/Linux）

利用菜单栏或快捷键操作后会弹出配置对话框，如图 10-20 所示。

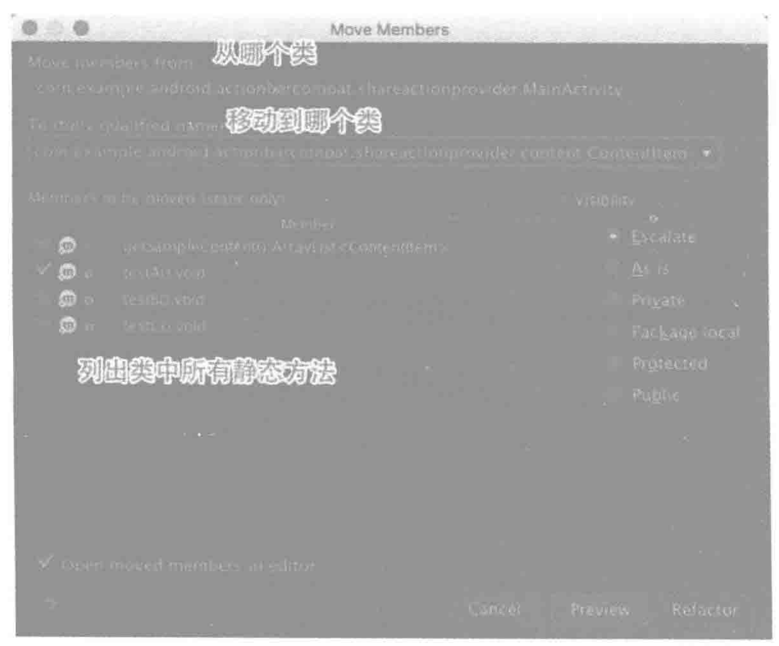


图 10-20

单击【Refactor】按钮进行重构，结果被选中的静态方法就被移动到指定的类中了，所有使用到方法的地方，类名也会变为新的类。

10.8 移动静态字段

前提条件：将光标放在静态字段名上。

菜单栏：Refactor→Move...

快捷键：fn + F6（macOS），有可能会跟系统快捷键冲突或者F6（Windows/Linux）

【实例演示】

01 将光标放在静态字段名上→按快捷键 fn + F6（macOS）→ 然后弹出 Move Members 对话框，如图 10-21 所示。

图 10-21

02 指定目标路径。单击【To（fully qualified name)】最右边的...，弹出目标地址选择对话框。我们可以通过输入类名关键字来搜索目标，如图 10-22 所示。还可以直接在项目中进行选择，如图 10-23 所示。

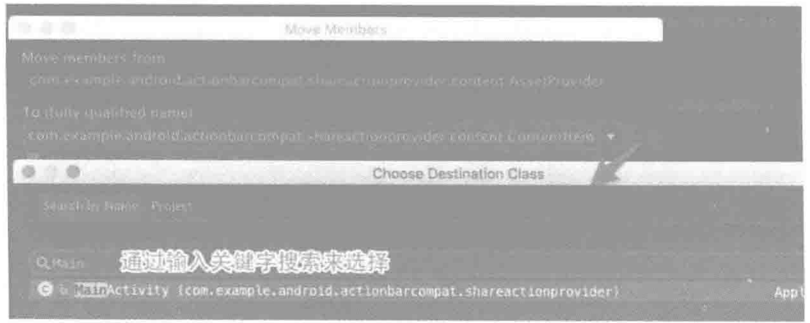


图 10-22

03 单击【OK】按钮后选中路径，再单击【Refactor】进行重构。

图 10-23

10.9 复制

复制重构是在不同的包下面创建类的副本，也可以创建文件、目录或包的副本，前提是副本跟原始文件不能在同一个目录下。如果放在同一个目录下，需要把副本重命名。

菜单栏：Refactor→Copy...

快捷键：fn + F5（macOS）或者F5（Windows/Linux）

【实例演示】

01 选中类名 →执行快捷键 fn + F5（macOS），如图 10-24 所示。默认名是被复制类的名字，如果要放在同一个目录下需要对副本进行重命名。

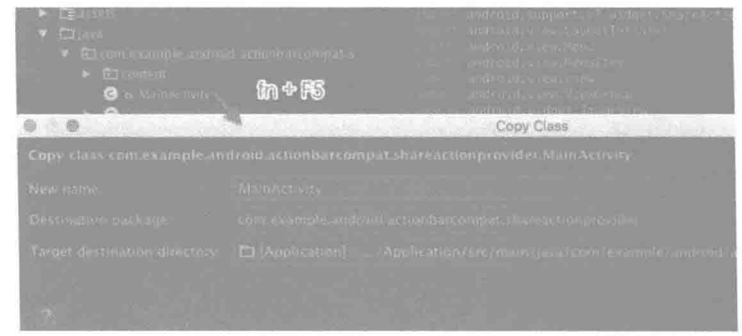


图 10-24

02 重命名文件 → 单击【OK】按钮 → 副本创建成功。

10.10 安全删除

安全删除是从源码中安全地删除文件或字符。为了保证能够安全地删除，Android Studio 会想办法找到被删除字符的用法，结果会在工具栏展示。我们可以在工具窗口浏览它们，并在删除之前对代码进行必要的修正。

前提条件：选中方法或参数。

操作步骤：菜单栏→Refactor→Safe Delete...。

【实例演示】安全删除一个类。

01 选中类文件 → 在菜单栏中执行安全删除操作，弹出安全删除对话框，如图 10-25 所示。

图 10-25

02 单击【OK】按钮 → 提示有些地方不能够安全删除，如图 10-26 所示。

Usages Detected

The following problems were found:
class com.example.android.actionbarcompat.shareactionprovider.MainActivit
has 1 usages that are not safe to delete.

图 10-26

03 查看用法或直接删除（建议先查看用法），如图 10-27 所示。

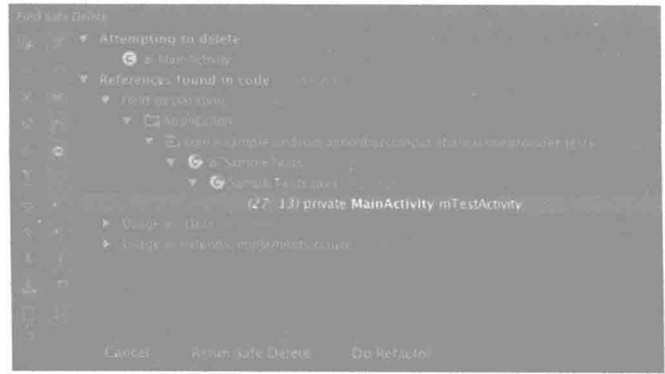


图 10-27

- Rerun Safe Delete：重新执行安全删除，会重新执行第 1 步操作。
- Do Refactor：执行重构。
- Cancel：取消此次重构。

如果发现有些地方的用法不需要删除，可以右击，并单击Exclude将此处排除在外。

（1）在调用层次结构中安全删除参数

安全删除参数前：

```
private void a(int i) {
    b(i);
}
```

```
    private void b(int i) {
        c(i);
    }

    private void c(int i) {
    }
```

01 选中方法 c 中的参数 int i，然后执行安全删除 → OK，如图 10-28 所示。

02 弹出删除对话框，可以选择方法传递的参数进行安全删除，如图 10-29 所示。

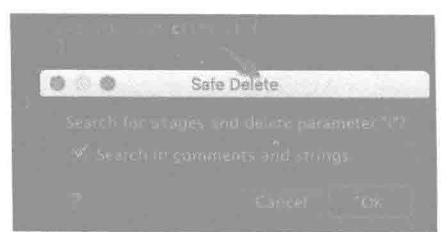


图 10-28　安全删除对话框

图 10-29

03 全选后单击【OK】按钮。安全删除参数重构之后：

```
    private void a() {
        b();
    }

    private void b() {
        c();
    }

    private void c() {
    }
```

（2）在调用层次结构中安全删除方法

安全删除方法前：

```
public class TestExtract {

    private void a(int i) {
        b(i);
    }

    private void b(int i) {
```

```
        c(i);
    }

    private void c(int i) {
    }
}
```

01 光标放在方法 a 上，执行安全删除（见图 10-30）。

图 10-30　安全删除

02 单击【OK】按钮后弹出删除对话框，我们可以选中方法进行级联删除，如图 10-31 所示。

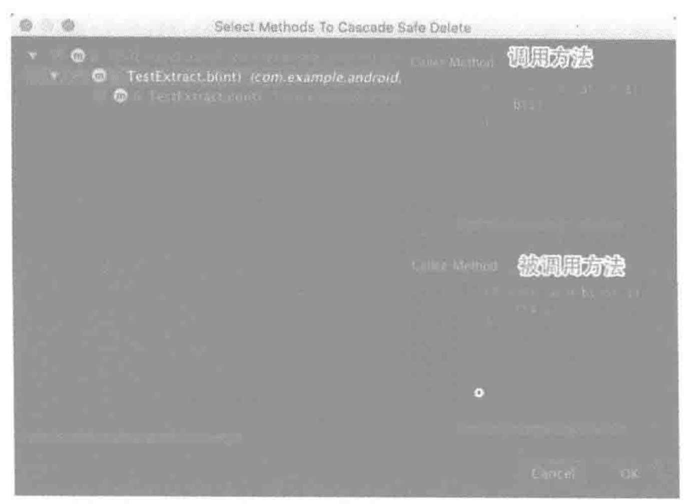


图 10-31

03 全选后单击【OK】按钮。安全删除方法后：

```
public class TestExtract {

}
```

10.11　提取变量

提取变量重构是将一个表达式提取为一个变量，并使用变量替换原来的表达式，这样能使代码更容易被理解。

要提取变量时，可以进行如下操作。

菜单栏： Refactor→Extract→Variable...

快捷键： option + command + V（macOS）或者Ctrl + Alt + V（Windows/Linux）

【实例演示】

提炼为变量前：

```
public void test() {
    String a = "My name is ";
    String b = a + new Info().getName();
    String c = b + new Info().getName();
}
```

01 将光标放到表达式 new Info().getName()上 → 按快捷键 option + command + V（macOS）→ 单击选择表达式，如图 10-32 所示。

02 选择替换范围，如图 10-33 所示。

图 10-32

图 10-33

03 选择变量名，如图 10-34 所示。

自定义变量名要再执行一次快捷键。如果想将变量声明为final，请选择Declare final。

提炼为变量后：

```
public void test() {
    String a = "My name is ";
    String name = new Info().
getName();
    String b = a + name;
    String c = b + name;
}
```

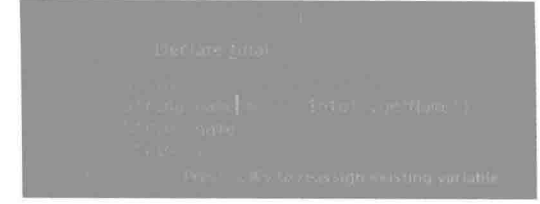


图 10-34

10.12 提取常量

提取常量是从临时变量快速提取出静态常量，这也是我们常用的重构手段。

菜单栏：Refactor→Extract→Constant...

快捷键：option + command + C（macOS）或者Ctrl + Alt + C（Windows/Linux）

【实例演示】

提取常量之前：

```
public void getInfo(){
    String name =  "老毕";
    searchByName(name);
    print(name);
}
```

01 按光标放到变量 NAME 上→按快捷键 option + command + C（macOS）。

02 弹出快速选择提示，如图 10-35 所示。我们可以快速选择常量名，如果想要更多的操作选项，就再执行一次快捷键，如图 10-36 所示。

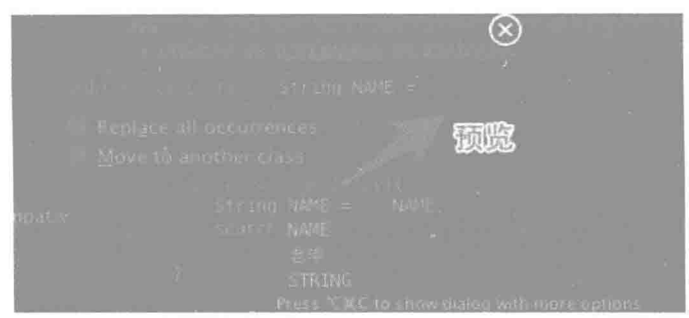


图 10-35

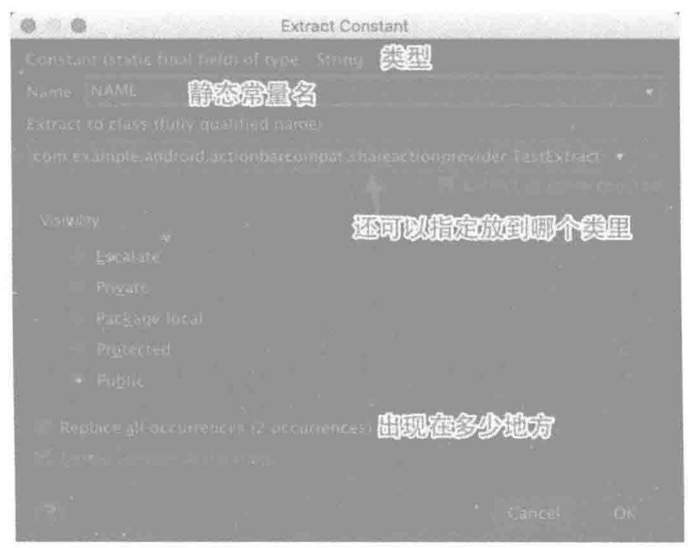


图 10-36

03 单击【OK】按钮后开始重构。提取常量之后：

```
public static final String NAME = "老毕";

    public void getInfo(){
        searchByName(NAME);
        print(NAME);
    }
```

10.13　提取字段

提取字段重构是将选中的表达式赋值给一个新声明的字段，原来的表达式使用这个新的字段来代替。

菜单栏： Refactor→Extract→Field...

快捷键： option + command + F（macOS）或者Ctrl + Alt + F（Windows/Linux）

【实例演示】

提取字段前：

```
public class TestExtract {
    Demo d = new Demo();

    public void getInfo() {
        String a = "我是";
        String b = a +  d.getName();
        String c = b +    d.getName();
    }

    public void getName(){
        System.out.println(d.getName());
    }

}

class Demo{
    public String getName(){
        return "帅哥";
    }
}
```

01 将光标放在 d.getName()上→按快捷键 option + command + F（macOS）→弹出表达式选择界面，如图 10-37 所示。

02 选择要提炼的表达式 → 弹出各种配置项，如图 10-38 所示。

图 10-37

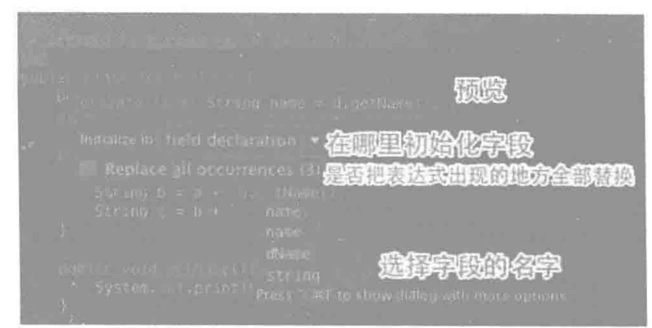


图 10-38

03 选择字段名为 name→在字段声明的地方初始化字段→替换全部表达式→执行重构。提取字段后：

```
public class TestExtract {
    Demo d = new Demo();
    private final String name = d.getName();

    public void getInfo() {
        String a = "我是";
        String b = a + name;
```

```
        String c = b + name;
    }

    public void getName(){
        System.out.println(name);
    }
}
```

在第 2 步选择字段在哪里初始化，这里有 3 个选项可以选择，如图 10-39 所示。

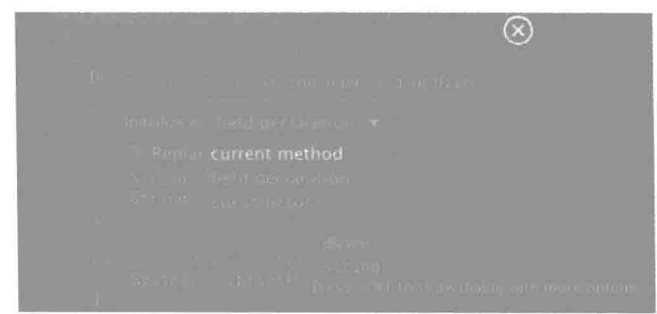


图 10-39

- current method：当前方法中。
- field declaration：字段声明的地方。
- constructor：构造器中。

默认是field declaration，其他两个我就不做演示了。

10.14 提取参数

提取参数重构是将方法A中的值提取为方法A的新参数，调用方法A的地方传入参数的值，这个值为之前提取的参数值。

菜单栏：Refactor→Extract→Parameter...

快捷键：option + command + P（macOS）或者Ctrl + Alt + P（Windows/Linux）

【实例演示】

提炼参数前：

```
public class TestExtract {

    public static void print() {
        System.out.println(getInfo());
    }

    private static String getInfo() {
        return "老毕".trim() + "是一个帅哥";
    }

}
```

01 将光标放在要提取的参数（"老毕"）上→按快捷键 option + command + P（macOS）→弹出参数选择窗口，如图 10-40 所示。

02 按回车键选择要提取的参数 → 弹出参数名选择窗口，如图 10-41 所示。

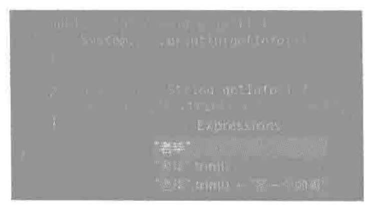


图 10-40

图 10-41

03 选择 s 作为参数名，其他默认。

提取参数后的代码：

```
public class TestExtract {

    public static void print() {
        System.out.println(getInfo("老毕"));
    }

    private static String getInfo(String s) {
        return s.trim() + "是一个帅哥";
    }
```

提 示

（1）回到第 3 步，选择覆写方法，结果如下：

```
public class TestExtract {
    public static void print() {
        System.out.println(getInfo());
    }

    private static String getInfo() {
        return getInfo("老毕");
    }

    private static String getInfo(String s) {
        return s.trim() + "是一个帅哥";
    }
}
```

（2）还是回到第 3 步，再执行一次快捷键就会有更多的选项，如图 10-42 所示。

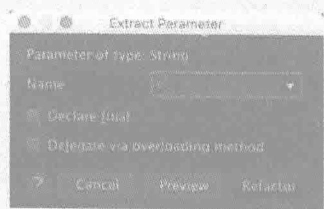


图 10-42

10.15　提取函数式参数

执行提取函数式参数重构：

（1）当我们要提取选中的代码片段到一个新的独立的方法中时，Android Studio会先分析选中的代码片段，找出方法签名。

（2）找出所有用到此方法签名的函数式接口（只包含一个抽象方法的接口，Java8 提供了@FunctionalInterface注解。），并建议你选择一个（如果接口被标记为@FunctionalInterface或使用一些知名的库，如Guava、Apache Collections等时才会被建议）。

（3）基于选择的接口，使用匿名类来包装代码片段，并使用匿名类作为参数。

菜单栏： Refactor→Extract→Functional Parameter...

快捷键： option + shift + command + P（macOS）或者Ctrl + Alt + Shift + P（Windows/Linux）

【实例演示】

提取参数前：

```
@FunctionalInterface
public interface Info {
    public void get (String name);
}

public class TestExtract {
    private void printInfo () {
        String name="老毕";
        System.out.println(name);
    }
    private void print() {
        printInfo();
    }
}
```

01 选中代码片段 System.out.println（name）；→按快捷键 option + Shift + command + P（macOS）。

02 Android Studio 会找出所有使用方法签名为（(String→void）的函数式接口，并建议你选择其中一个。（本例中选择接口 Info。）

03 选中的代码片段 System.out.println（name）；被包装成一个基于接口 Info 的匿名类，这个匿名类作为参数传递给 print()方法中的 printInfo()。

提取函数式参数后：

```
@FunctionalInterface
public interface Info {
    public void get (String name);
}
```

```
public class TestExtract {

    private void printInfo (Info info) {
        String name="老毕";
        info.get(name);
    }
    private void print() {
        printInfo(new Info() {
            @Override
            public void get(String name) {
                System.out.println(name);
            }
        });
    }
}
```

10.16 提取参数对象

某些参数总是同时出现，可能好几个方法都使用这样一组参数，为了避免参数列表过长，同时也为了避免重复代码，可以将这些参数提炼为参数对象，原来传入参数的地方使用这个参数对象代替。

操作步骤：菜单栏→Refactor→Extract→Parameter Object ...。

【实例演示】

提取参数对象前：

```
public class TestExtract {

    private void getAndroidInfo(String serialNumber, String packageName,
String versionName, String versionCode){
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.setSerialNumber(serialNumber);

        PackageInfo packageInfo = new PackageInfo();
        packageInfo.setPackageName(packageName);
    }
}
```

01 选中参数 getAndroidInfo 的所有参数 → 执行菜单栏：Refactor → Extract → Parameter Object ... → 弹出参数配置对话框，如图 10-43 所示。

02 本例我们选择创建内部类，名为 AndroidInfo，默认提炼所有参数 → Refactor。

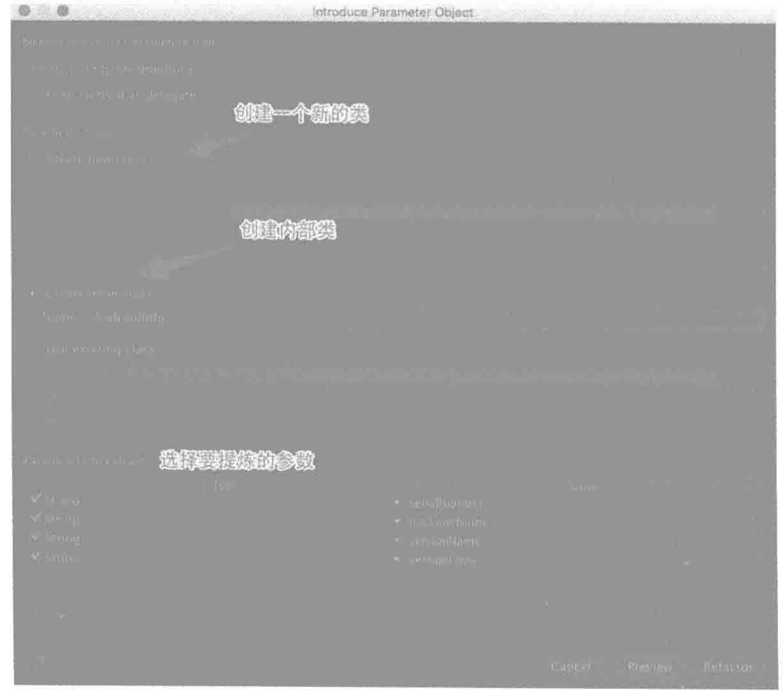


图 10-43

提取参数对象后：

```
public class TestExtract {

    private void getAndroidInfo(AndroidInfo androidInfo){
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.setSerialNumber(androidInfo.getSerialNumber());

        PackageInfo packageInfo = new PackageInfo();
        packageInfo.setPackageName(androidInfo.getPackageName());
    }

    private static class AndroidInfo {
        private final String serialNumber;
        private final String packageName;
        private final String versionName;
        private final String versionCode;

        private AndroidInfo(String serialNumber, String packageName, String
versionName, String versionCode) {
            this.serialNumber = serialNumber;
            this.packageName = packageName;
            this.versionName = versionName;
            this.versionCode = versionCode;
        }

        public String getSerialNumber() {
            return serialNumber;
        }
```

```
        public String getPackageName() {
            return packageName;
        }

        public String getVersionName() {
            return versionName;
        }

        public String getVersionCode() {
            return versionCode;
        }
    }
}
```

10.17　提取方法

在重构时我们经常想把复杂方法中的一段代码提取出来作为一个单独的方法使用，这种重构手法叫Extract Method。

菜单栏： Refactor→Extract→Method

快捷键： option + command + M（macOS）或者Ctrl + Alt + M（Windows/Linux）

【实例演示】

01 选中要提取的代码片段。

02 按快捷键 option + command + M（macOS）→弹出配置对话框，如图 10-44 所示。

03 选择方法可见性，如图 10-45 所示。

04 选择或自定义方法名，如图 10-46 所示。

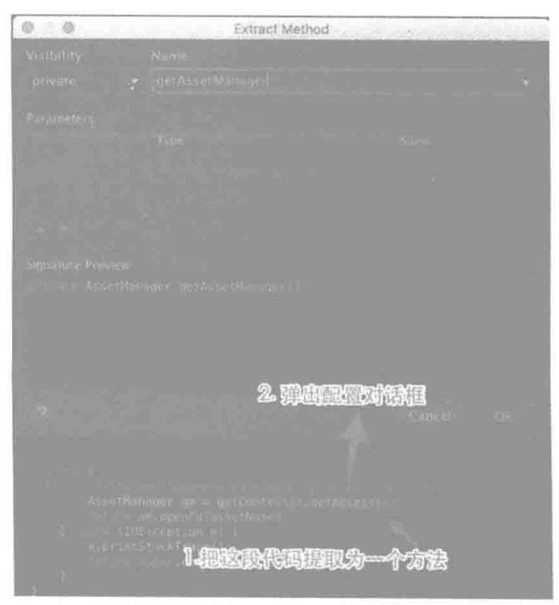


图 10-44

图 10-45

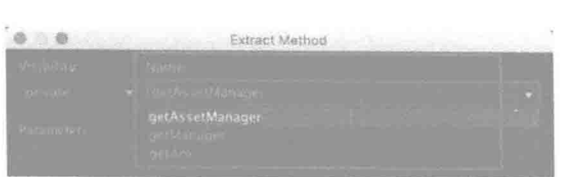


图 10-46

Android Studio 智能推荐一些方法名，我们直接选择就可以了，当然也允许我们自定义。注意方法名一定要解释这个方法的用途。

05 查看预览。对话框底部提供了预览功能，便于我们实时预览。

06 单击【OK】按钮。如图 10-47 所示，此时我们要提炼的代码被放进了一个独立的方法中。

图 10-47

10.18 提取方法对象

我们可以把一段代码提取为方法对象使用。

操作步骤：菜单栏→Refactor→Extract→Method Object...。

【实例演示】

01 选中要提取的代码片段。

02 菜单栏：Refactor → Extract → Method Object... → 弹出配置对话框，如图 10-48 所示。

图 10-48

10.19 提取委托

委托是有两个对象参与处理同一个请求，接受请求的对象将请求委托给另一个对象来处理。提取委托重构是在一个类中提取一些字段和方法到一个单独的、新创建的类中。

当一个类变得太大并且做了太多的事情时，我们就需要考虑重构，将这个类拆分为更小、更有凝聚力的类。

提取委托时，可以进行如下快捷操作。

操作步骤：菜单栏→Refactor→Extract→Delegate...。

【实例演示】

提取委托前：

```
public class TestExtract {

    public String getInfo() {
        String sex = "帅哥";
        return "我是" + sex;
    }

}

class Demo{
    TestExtract  t = new TestExtract();
    String info = t.getInfo();
}
```

01 把光标放在 return "我是" + sex; 上 → Refactor → Extract → Delegate... → 弹出配置对话框，如图 10-49 所示。

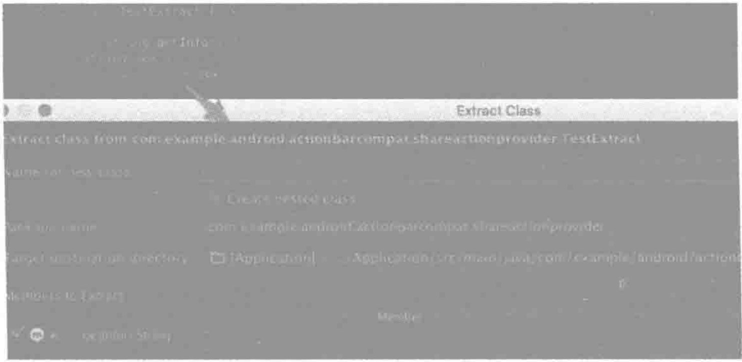


图 10-49

02 输入新的类名 → 不创建嵌套类 → 指定目标路径和包名（使用默认），如图 10-50 所示。

03 执行重构。重构后新类 Info 已被创建，重构后的代码：

```
public class TestExtract {

    private final Info info = new Info();

    public String getInfo() {
        return info.getInfo();
    }
}

class Demo{
    TestExtract  t = new TestExtract();
    String info = t.getInfo();
}
```

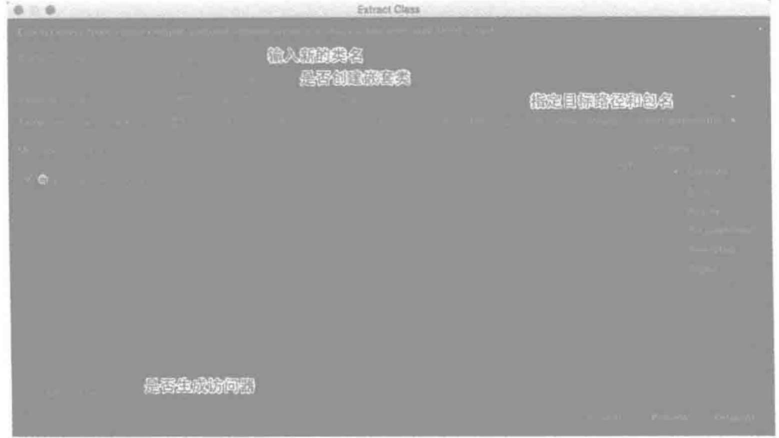


图 10-50

如果在第 2 步勾选了创建嵌套类，结果如下：

```
public class TestExtract {

    private final Info info = new Info();

    public String getInfo() {
        return info.getInfo();
    }

    public static class Info {
        public Info() {
        }

        public String getInfo() {
            String sex = "帅哥";
            return "我是" + sex;
        }
    }
}

class Demo{
    TestExtract  t = new TestExtract();
    String info = t.getInfo();
}
```

10.20　提取接口

提取接口重构是从一个已存在的类中提取接口，可以从某个类中选择方法，把选中的方法提取到一个单独的接口中。

操作步骤：菜单栏→Refactor→Extract→Interface ...。

【实例演示】

01 将光标放在某个类的任意位置 → 执行菜单栏：Refactor → Extract → Interface ... → 弹出配置对话框，如图 10-51 所示。

从哪个类中提炼接口
接口名
新的接口放在哪里
选择要提炼到接口的方法

图 10-51

02 输入接口名为 IContentItem，选中方法 getContentUri()和 getShareIntent（context：Context），将其提炼到接口中→ 执行重构。

提取的接口如下：

```
/**
 * Created by bixiaopeng on 16/1/3.
 */
public interface IContentItem {
    Uri getContentUri();
    Intent getShareIntent(Context context);
}
```

原来的类实现了这个接口：

```
public class ContentItem implements IContentItem {
  ...
}
```

10.21 提取父类

某个类做了应该由两个类做的事情，我们需要考虑对这个类进行拆分，从中提取出一个新类。如果需要父类化就提取父类，如果需要子类化就提取子类。

提取父类

操作步骤：菜单栏→Refactor→Extract→Superclass...。

【实例演示】

01 将光标放到类名上 →Refactor → Extract → Superclass... → 弹出提取父类配置对话框，如图 10-52 所示。

02 默认勾选 Extract superclass → 自定义父类名为 Content，如图 10-53 所示。

请注意，当你勾选某个方法时，如果下面有参数显示红色，说明这个方法中调用了这个方法或字段，需要一并勾选，不然是会出错的。

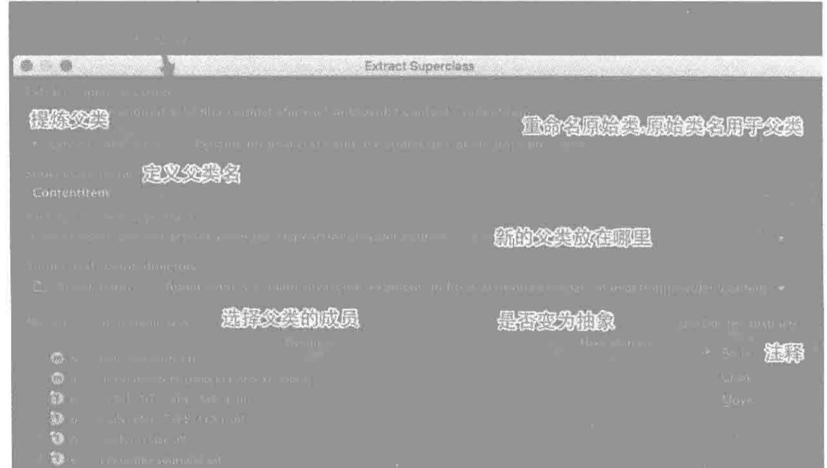


图 10-52

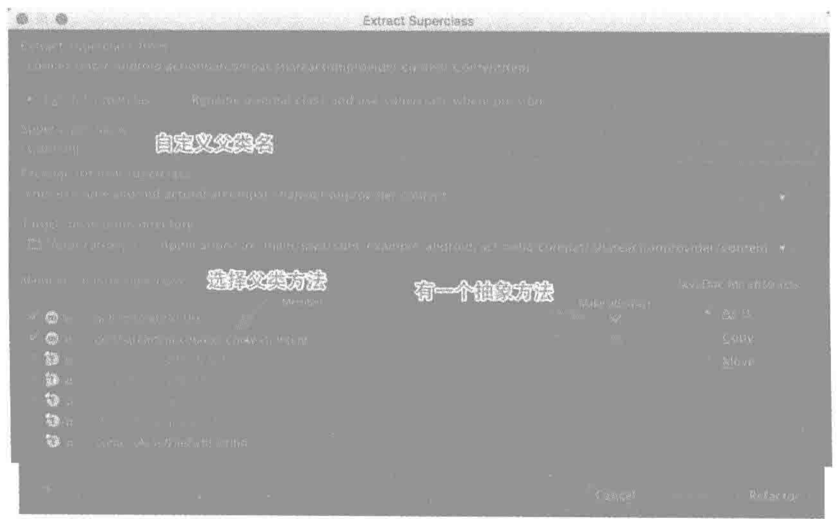


图 10-53

当然第 2 步还有另外一个选项：Rename original class and use superclass where possible（重命名原始类，父类使用原始类名），如图 10-54 所示。

图 10-54

03 单击 Refactor 进行重构，弹出确认对话框，如图 10-55 所示，提示父类创建成功，是否使用父类来替换子类。

04 单击 Yes → 工具窗口显示哪里使用了子类，需要再次确认是否执行重构，如图 10-56 所示。

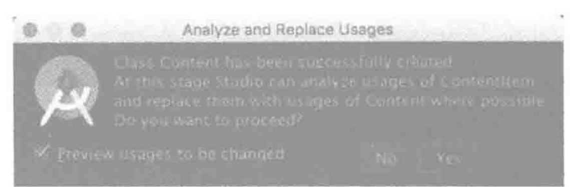


图 10-55

图 10-56

05 单击 Do Refactor → 全部重构。

10.22 内联方法

内联方法跟提取方法是反过来的，一个是将代码片段提取为方法，一个是将方法转为代码片段。所以内联方法的用途之一就是恢复被提取出来的方法。

什么时候使用内联方法？

当一个方法的本体与方法名同样清楚易懂时，就没有必要使用单独的方法了，直接把方法换成方法体即可，也就是使用内联方法进行重构。

菜单栏： Refactor →Inline

快捷键： option+command + N（macOS）或者Ctrl + Alt + N（Windows/Linux）

【实例演示】

```
    @Override
    public AssetFileDescriptor openAssetFile(Uri uri, String mode) throws
FileNotFoundException {
        ....
        try {
            AssetManager am = getAssetManager();//此处调用
            return am.openFd(assetName);
        } catch (IOException e) {
            e.printStackTrace();
            return super.openAssetFile(uri, mode);
        }
```

```
    }

    private AssetManager getAssetManager() {
        return getContext().getAssets();
    }
```

01 将光标放到方法名（getAssetManager）上。

02 执行内联操作(快捷键 option+command + N（macOS)）→ 弹出对话框，如图 10-57 所示。

- inline all invocations and remove the method：内联所有的调用并删除方法。

图 10-57

03 单击【Refactor】按钮。

```
  @Override
    public AssetFileDescriptor openAssetFile(Uri uri, String mode) throws 
FileNotFoundException {
        ....

        try {
            AssetManager am = getContext().getAssets();//此处被方法替换
            return am.openFd(assetName);
        } catch (IOException e) {
            e.printStackTrace();
            return super.openAssetFile(uri, mode);
        }
    }
```

10.23 内联临时变量

当一个临时变量只被简单地引用了一次，而且会影响其他重构时，就要使用内联变量进行重构。

菜单栏： Refactor →Inline

快捷键： option+command + N（macOS）或者Ctrl + Alt + N（Windows/Linux）

【实例演示】

```
    public String getInfo(){
        String name = "老毕";
        return name + "是个帅哥";    }
```

01 将鼠标放到 name 上。

02 按快捷键 option+command + N（macOS），会弹出内联变量对话框，如图 10-58 所示。如果有多个地方引用，可以单击【Preview】进行预览。

03 单击【Refactor】进行重构。

重构后，临时变量被删除，引用的地方直接替换成了原来的变量本身。

图 10-58

```
    public String getInfo(){
        return "老毕" + "是个帅哥";
    }
```

10.24 查找并替换重复代码

查找并替换重复代码重构是查找与选中方法或常量字段重复或类似的代码，并通过调用此方法或常量来替换它们。

操作步骤：菜单栏→Refactor→Find and Replace Code Duplicates...。

【实例演示】

查找并替换重复代码前：

```
public class TestExtract {

    public void test() {
        int a = 10;
        int b = 20;
        int c = a + b;
        int d = b + c;
    }

    private int getTotal(int a,  int b) {
        return a + b;
    }

}
```

01 将光标放在方法 getTotal 上，在菜单栏中选择 Refactor → Find and Replace Code Duplicates... → 弹出配置对话框，主要选择查找范围，如图 10-59 所示。

02 使用默认配置（当前文件）→OK→弹出替换对话框→选择 All（全部替换）。

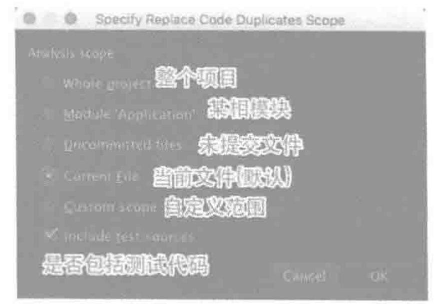


图 10-59

查找并替换重复代码后：

```
public class TestExtract {

    public void test() {
        int a = 10;
        int b = 20;
        int c = getTotal(a, b);
        int d = getTotal(b, c);
    }

    private int getTotal(int a, int b) {
        return a + b;
    }

}
```

10.25 反转布尔值

反转布尔值重构是把一个Boolean方法或变量改变为相反的意义，也就是原来是true的变为false，原来是false的变为true。

操作步骤：菜单栏→Refactor→Invert Boolean。

【实例演示】

反转布尔值前：

```
public class TestExtract {

    int a = 20;

    public boolean test() {
        if (a > 20) {
            return true;
        } else {
            return false;
        }

    }
}
```

01 将光标放在 test 方法上，选择菜单栏中的 Refactor → Invert Boolean →弹出反转布尔值方法确认对话框，如图 10-60 所示。

图 10-60

02 单击【Refactor】按钮。反转布尔值后：

```
public class TestExtract {

    int a = 20;

    public boolean test() {
        if (a > 20) {
            return false;
        } else {
            return true;
        }

    }
}
```

10.26 把成员拉到父类

把成员拉到父类重构是把当前类的方法和属性移动到它的父类中去。

操作步骤： 菜单栏→Refactor→Pull Members Up...。

【实例演示】

把成员拉到父类重构前：

```
/**
 * 子类
 */
public class TestExtract extends Extract{

    public void test() {
        System.out.println("just test");
    }

    public void test2() {
        System.out.println("just test");
    }
}

/**
 * 父类
 */
abstract class Extract {
    public abstract void test();
}
```

01 将光标放在方法 test2 上 → 选择菜单栏中的 Refactor → Pull Members Up... → 弹出配置对话框，如图 10-61 所示。

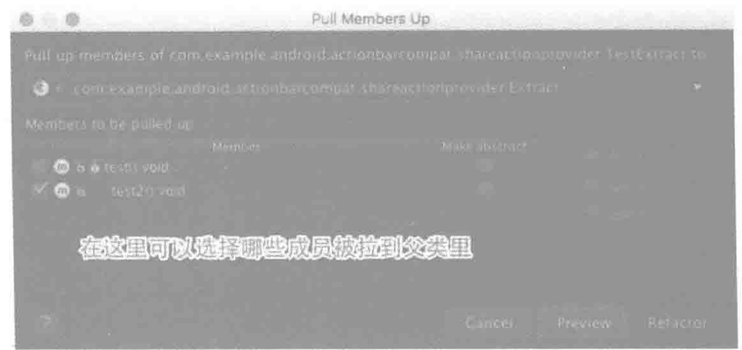


图 10-61

02 单击【Refactor】按钮。重构以后的代码：

```
/**
* 子类
 */
public class TestExtract extends Extract{

    public void test() {
        System.out.println("just test");
    }

}

/**
 * 父类
 */
abstract class Extract {
    public abstract void test();          //重构以后这个方法被拉到父类中来了
    public void test2() {
        System.out.println("just test");
    }
}
```

10.27 把成员推到子类

把成员推到子类重构是把父类的方法和属性移动到所有子类中，父类的方法可以选择性地保留抽象方法。

操作步骤：菜单栏→Refactor→Push Members Down...。

【实例演示】

把成员推到子类跟把成员拉到父类正好相反，因此我们复用上面的代码。

```
/**
 * 子类
 */
public class TestExtract extends Extract{
    public void test() {
```

```
        System.out.println("just test");
    }
}

/**
 * 父类
 */
abstract class Extract {
    public abstract void test();                //我们将把这个方法推到子类中去
    public void test2() {
        System.out.println("just test");
    }
}
```

01 把光标放在方法 test2 上→选择菜单栏中的 Refactor → Extract → Push Members Down… → 弹出配置对话框，如图 10-62 所示。

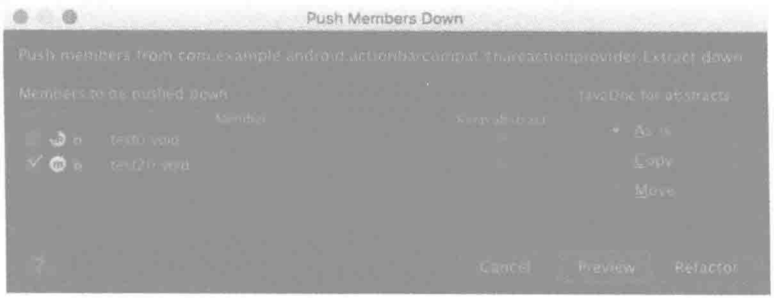


图 10-62

02 单击【Refactor】按钮。重构以后的代码：

```
/**
 * 子类
 */
public class TestExtract extends Extract{

    public void test() {
        System.out.println("just test");
    }

    //重构以后这个方法被推到子类中来了
    public void test2() {
        System.out.println("just test");
    }
}

/**
 * 父类
 */
abstract class Extract {
    public abstract void test();
}
```

10.28 尽可能使用接口

尽可能使用接口重构是委托执行一个指定的方法，这个方法是从基类或接口中派生出来的。

操作步骤：菜单栏→Refactor→Use Interface Where Possible...。

【实例演示】

重构之前的代码：

```
/**
 * 接口
 */
interface IExtract {
    void test();
}

/**
 * 实现
 */
class ExtractImpl implements IExtract{

    @Override
    public void test() {
        System.out.println("test1");
    }

    public void test2(){
        System.out.println("test2");
    }
}

/**
 * 使用
 */
public class TestExtract {

    ExtractImpl extract1;
    ExtractImpl extract2; //这个类可以使用接口替换

    public void testDemo(){
        extract1.test();
        extract1.test2();

        extract2.test();//这个方法是从接口中派生出来的

        if (extract2 instanceof ExtractImpl){
            System.out.println("此类可用接口");
        }
    }
}
```

01 将光标放在类 ExtractImpl 上→选择菜单栏中的 Refactor → Use Interface Where Possible… →弹出配置窗口，如图 10-63 所示。

图 10-63

02 单击【Refactor】按钮执行重构。重构以后：

```
/**
 * 接口
 */
interface IExtract {
    void test();
}

/**
 * 实现
 */
class ExtractImpl implements IExtract{

    @Override
    public void test() {
        System.out.println("test1");
    }

    public void test2(){
        System.out.println("test2");
    }
}

/**
 * 使用
 */
public class TestExtract {

    ExtractImpl extract1;
    IExtract extract2; //这个类可以使用接口替换

    public void testDemo(){
        extract1.test();
        extract1.test2();

        extract2.test();//这个方法是从接口中派生出来的

        if (extract2 instanceof IExtract){
```

```
            System.out.println("此类可用接口");
        }
    }
}
```

对比一下，如图 10-64 所示。

图 10-64

10.29 使用委托替换继承

使用委托替换继承重构是移除当前子类继承的父类，但同时又能够保留并使用父类中的一些功能。

什么时候用委托替换继承？

当某个子类只使用了父类的一部分功能时，或者根本就不需要继承来的数据时，就要考虑使用委托替换继承。

如何使用委托替换继承？

Android Studio会先在子类中创建一个私有的内部类，用来继承超类或接口。然后在子类中新建一个字段来实例化这个内部类，再委托子类中的方法来调用父类的方法。

操作步骤：菜单栏→Refactor→Replace Inheritance with Delegation...。

【实例演示】

重构之前的代码：

```
/**
 * 父类
 */
abstract class Info {

    public static final String NAME = "个人信息";

    public abstract void printName();

    public abstract void printAge();

    public abstract void printHeight();

    public void printWeigt() {
        System.out.println("80 公斤");
    }
}

/**
 * 子类
 */
public class TestExtract extends Info {


    @Override
    public void printName() {

    }

    @Override
    public void printAge() {

    }

    @Override
    public void printHeight() {

    }
```

子类TestExtract继承了父类Info，可我只想用到父类的printName方法， 这个时候就可以使用委托来代替继承。

01 将光标放在子类 TestExtract 上→选择菜单栏中的 Refactor → Replace Inheritance with Delegation... → 弹出配置对话框，如图 10-65 所示。

02 在配置对话框中选择要使用到的父类中的方法，比如 printName。

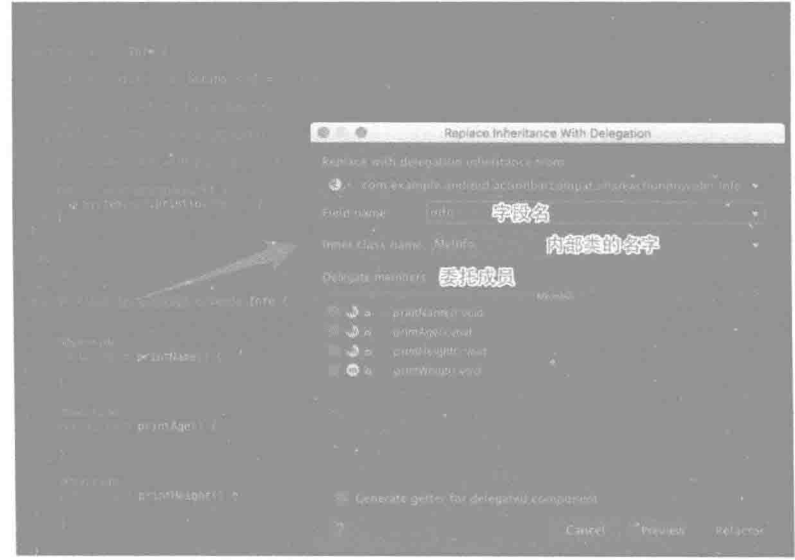


图 10-65

03 单击【Refactor】按钮重构。重构后的代码：

```
/**
 * 父类
 */
abstract class Info {

    public static final String NAME = "个人信息";

    public abstract void printName();

    public abstract void printAge();

    public abstract void printHeight();

    public void printWeigt() {
        System.out.println("80 公斤");
    }
}

/**
 * 子类
 */
public class TestExtract {

    //我们在对话框中指定的新字段名用来实例化新建的内部类
    private final MyInfo info = new MyInfo();

    public void printName() {
        info.printName();//委托调用父类方法
    }
```

```
    //新建的内部类，用来继承父类，并实现需要用到的父类方法
    private class MyInfo extends Info {
        @Override
        public void printName() {

        }

        @Override
        public void printAge() {

        }

        @Override
        public void printHeight() {

        }
    }
}
```

10.30 移除中间人

移除中间人重构是让你不使用委托方法而直接调用受委托的类。

什么时候使用移除中间人重构?

当某个类做了过多的简单委托时，服务类就完全变成了“中间人”，因为每当你要使用受委托类的新特性时都要在服务类添加一个简单的委托方法，这样会非常痛苦。此时你就要考虑使用移除中间人重构。

操作步骤：菜单栏→Refactor→Remove Middleman...。

【实例演示】

重构之前的代码：

```
class Message {
    Info info;

    public Message getMessage() {
        return info.getMessage();//委托方法
    }
}


/**
 * 受委托的类
 */
class Info {
    private Message message;

    public Info(Message message) {
```

```
        this.message = message;
    }

    public Message getMessage() {
        return message;
    }
}

/**
 * 客户端
 */
public class TestExtract {
    Message m;
    Message m2 = m.getMessage();

}
```

01 把光标放在委托方法上→选择菜单栏中的 Refactor → Remove Middleman…→弹出配置对话框，如图 10-66 所示。

图 10-66

02 单击【Refactor】按钮进行重构。重构后的代码：

```
class Message {
    Info info;

    public Info getInfo() {
        return info; //这里去掉了委托，直接调用受委托的类
    }
}
```

```
/**
 * 受委托的类
 */
class Info {
    private Message message;

    public Info(Message message) {
        this.message = message;
    }


    public Message getMessage() {
        return message;
    }
}

/**
 * 客户端
 */
public class TestExtract {
    Message m;
    Message m2 = m.getInfo().getMessage();

}
```

10.31 包装方法返回值

包装方法返回值重构是使用一个新建的包装类来替换一个方法的返回值，这个类也可以是已存在的类。

当你想让一个方法返回更多信息时，怎么办呢？

使用一个包装类来作为返回值，这样就可以返回更多的数据了。它通常用来包装一个简单的返回值，使所需的接口和实现保持独立。

操作步骤：菜单栏→Refactor→Wrap Return Value...。

【实例演示】

重构之前的代码：

```
class Info {
    String name;

    //我们来包装这个方法的返回值
    public String getName() {
        return name;
    }

}
```

01 将光标放到方法 getName 上→选择菜单栏中的 Refactor → Wrap Return Value… → 弹出配置对话框，如图 10-67 所示。

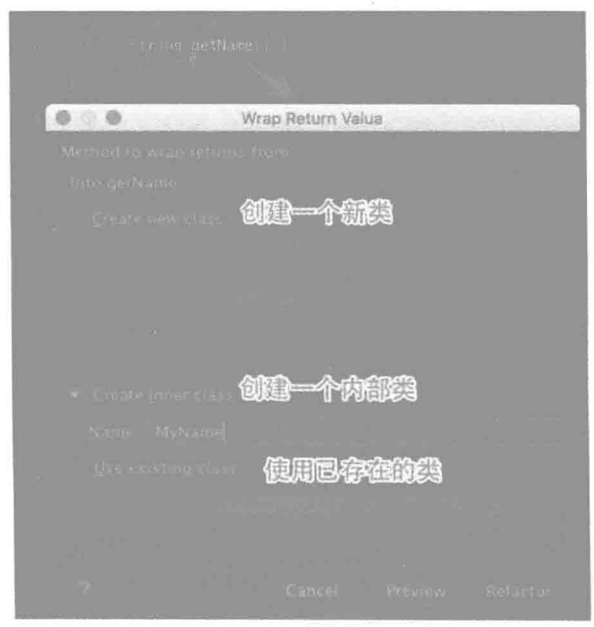


图 10-67

02 创建或选择一个包装类，我们选择创建一个内部类，名为 MyName。

03 单击【Refactor】按钮进行重构。

重构后的代码：

```
class Info {
    String name;

    //我们来包装这个方法的返回值
    public MyName getName() {
        return new MyName(name);//返回了一个包装类
    }

    //新创建的内部类
    public class MyName {
        private final String value;

        public MyName(String value) {
            this.value = value;
        }

        public String getValue() {
            return value;
        }
    }
}
```

10.32 将匿名类转成内部类

匿名类转成内部类重构可以把一个匿名类转成内部类，这样一来这个类的其他部分就可以共享此类。

操作步骤：菜单栏→Refactor→Convert Anonymous to Inner...

【实例演示】

重构前的代码：

```
interface IExtract {
    int test();
}

public class TestExtract {

    public IExtract method() {
        final int i = 0;
        return new IExtract() {//匿名类
            @Override
            public int test() {
                return i;
            }
        };
    }

}
```

01 把光标放在 new IExtract()上，选择菜单栏中的 Refactor → Convert Anonymous to Inner... → 弹出配置对话框，如图 10-68 所示。

图 10-68

02 使用默认配置，单击【OK】按钮。重构以后的代码：

```
interface IExtract {
    int test();
}

public class TestExtract {

    public IExtract method() {
        final int i = 0;
        return new MyIExtract(i);
    }

  //内部类
  private static class MyIExtract implements IExtract {
        private final int i;

        public MyIExtract(int i) {
            this.i = i;
        }

        @Override
        public int test() {
            return i;
        }
    }
}
```

10.33 封装字段

封装字段重构可以隐藏数据（改变可访问性），然后创建访问器（get/set方法）来访问。操作步骤：菜单栏→Refactor→Encapsulate Field ...。

【实例演示】

重构前的代码：

```
class Info {
    public String name; //封装这个字段

}


public class TestExtract {
    public Info info;
    public void print(){
        info.name = "老毕";
    }
}
```

01 将光标放在字段 name 上 → 菜单栏：Refactor → Encapsulate Field → 弹出配置对话框，如图 10-69 所示。

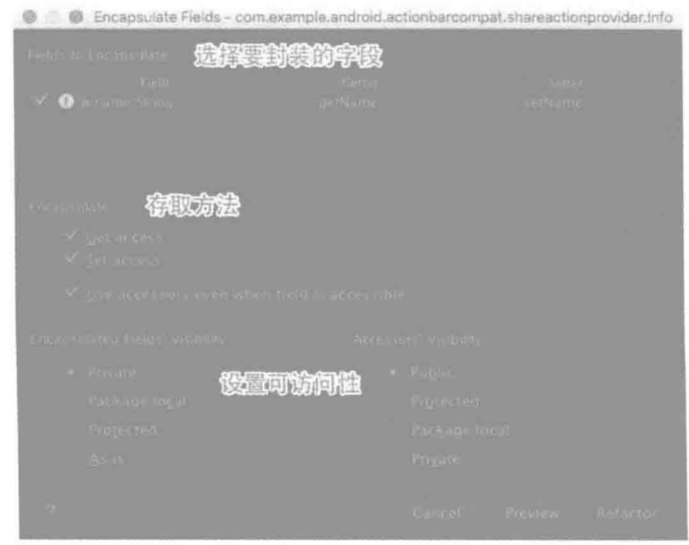


图 10-69

02 选择要封装的字段，设置存取方法和可访问性。

03 单击【Refactor】进行重构。重构后的代码：

```
class Info {
    private String name; //封装这个字段，注意变化

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}


public class TestExtract {
    public Info info;
    public void print(){
        info.setName("老毕");
    }
}
```

10.34 使用查询替换临时变量

如果一个临时变量保存了一个表达式，使用查询替换临时变量重构就是将这个表达式提炼到一个独立的方法中，并将这个临时变量的所有引用点替换为新的方法的调用。

操作步骤：菜单栏→Refactor→Replace Temp with Query...。

【实例演示】

重构前的代码：

```
public class TestExtract {

    public void info() {
        String name = "bixiaopeng";
         //替换这个临时变量
        String info = "name is: " + name.toUpperCase();
        System.out.println(info);
    }
}
```

01 将光标放在临时变量 info 上→菜单栏：Refactor → Replace Temp with Query... → 弹出配置对话框，如图 10-70 所示。

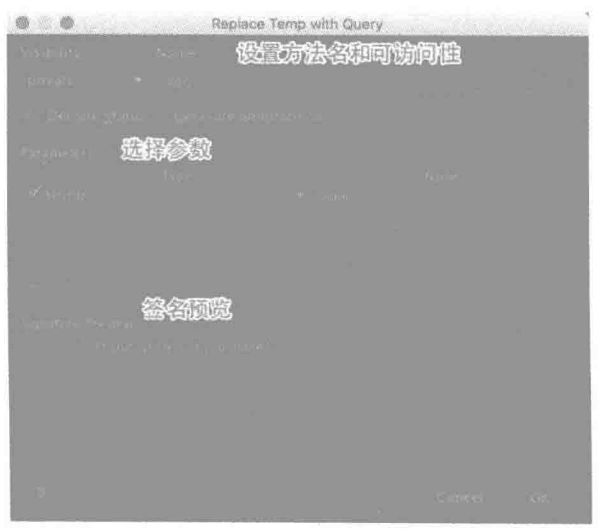


图 10-70

02 设置方法名和可访问性，并选择方法的参数。

03 单击【OK】按钮进行重构。重构后的代码：

```
public class TestExtract {

    public void info() {
        String name = "bixiaopeng";
        System.out.println(info(name));
    }
    //原来的表达式被提炼到这个新方法中
    private String info(String name) {
        return "name is: " + name.toUpperCase();
    }
}
```

10.35 使用工厂方法替换构造方法

使用工厂方法替换构造方法重构是使用一个静态方法返回一个实例化的类来替换原来的构造方法。

工厂方法是将值改为引用对象的基础，当构造方法不能满足需求时，就需要考虑使用工厂方法来替换它。根据参数的个数和类型，也可以让工厂方法返回不同的构建行为。

操作步骤：菜单栏→Refactor→Replace Constructor with Factory Method...。

【实例演示】

重构前的代码：

```
class Info {

    private String name;
    //构造方法
    public Info(String name){
        this.name = name;
    }

    public void print(){
        System.out.println(name);
    }
}


public class TestExtract {
    public static void main(String[] args) {
        Info info = new Info("old man");
    }
}
```

01 将光标放在构造方法上→菜单栏：Refactor →Replace Constructor with Factory Method... →弹出配置对话框，如图 10-71 所示。

图 10-71

02 输入工厂方法的名字 → 单击【Refactor】按钮进行重构。重构后的代码：

```
class Info {

    private String name;
    //构造方法
    private Info(String name){
        this.name = name;
    }

    //新建了一个工厂方法
    public static Info createInfo(String name) {
        return new Info(name);
    }

    public void print(){
        System.out.println(name);
    }
}


public class TestExtract {
    public static void main(String[] args) {
        Info info = Info.createInfo("old man");
    }
}
```

10.36 使用构建器替换构造方法

当创建一个对象需要传入多个参数的时候我们通常会根据参数的数量写多个不同的构造方法，但是这种方法不够灵活，可读性也不高。为了改善这种情况，我们需要使用构建器来替换构造方法。

操作步骤：菜单栏→Refactor→Replace Constructor with Builder...。

【实例演示】

重构前的代码：

```
class Info {

    private int a, b, c, d;

    //构造方法，要传入多个参数，写起来复杂且不够灵活
    public Info(int a,  int b , int c , int d){
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
    }
```

```
    public void print(){
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
    }
}


public class TestExtract {
    public static void main(String[] args) {
        Info info = new Info(1, 2, 3, 4);
    }
}
```

01 将光标放在构造方法 Info 上 → 菜单栏：Refactor → Replace Constructor with Builder… → 弹出配置对话框，如图 10-72 所示。

图 10-72

02 配置完成后单击【Refactor】按钮进行重构。

首先会新建一个构建器类InfoBuilder：

```
public class InfoBuilder {
    private int a;
    private int b;
    private int c;
```

```
    private int d;

    public InfoBuilder setA(int a) {
        this.a = a;
        return this;
    }

    public InfoBuilder setB(int b) {
        this.b = b;
        return this;
    }

    public InfoBuilder setC(int c) {
        this.c = c;
        return this;
    }

    public InfoBuilder setD(int d) {
        this.d = d;
        return this;
    }

    public Info createInfo() {
        return new Info(a, b, c, d);
    }
}
```

原来的调用处也会被替换为实例化新的构建类：

```
public class TestExtract {
    public static void main(String[] args) {
        Info info = new
InfoBuilder().setA(1).setB(2).setC(3).setD(4).createInfo();
    }
}
```

看起来是不是非常直观了？可读性大大提高。

10.37 泛型化

泛型化重构用来将那些没有使用泛型的代码转化为泛型可识别的代码。重构时Android Studio会分析现有代码的上下文，然后为每个原始类型创建安全且一致的参数类型。

操作步骤：菜单栏→Refactor→Gentrify...。

【实例演示】

重构前的代码：

```
public class TestExtract {
    public static void main(String[] args) {
```

```
        List list = new ArrayList();
        list.add(1);
    }
}
```

01 将光标放在 list 上→菜单栏：Refactor → Gentrify… → 弹出配置对话框，如图 10-73 所示。

图 10-73

- Drop obsolete casts：删除过时的转换。
- Leave Object-parameterized types raw：保留对象参数化类型。
- Preserve raw arrays：保留原始队列。
- Perform exhaustive search：执行穷举搜索。
- Generify Objects：泛型化对象。
- Produce wildcard types：产生通配符类型。

02 使用默认选项→单击【Refactor】按钮进行重构。重构后的代码：

```
public class TestExtract {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        list.add(1);
    }
}
```

这里我们看到Android Studio分析上下文得出的参数类型为Integer，如果执行list.add（"a"），参数类型就会是String。

10.38 国际化

国际化重构就是从源代码中提取字符串，并将它们作为属性写入属性文件中，再通过生成的表达式来读取Resource Bundle中的属性来获得多语言的字符串。

操作步骤：菜单栏→Refactor→Internationalize...

【实例演示】

重构前的代码：

```
public class TestExtract {
    public static void main(String[] args) {
```

```
        String name = "bixiaopeng";
    }
}
```

01 将光标放在 bixiaopeng 上→菜单栏：Refactor→Internationalize→弹出配置对话框，如图 10-74 所示。

02 选择一个属性文件来写入 key 和 value，使用默认的 Resource bundle 表达式→单击【OK】按钮。

```
public class TestExtract {
    public static void main(String[] args)
{
        //这里会报错
        String name =
resourceBundle.getString("bixiaopeng");
    }
}
```

图 10-74

生成的代码会报错，需要稍微改动一下：

```
public class TestExtract {
    public static void main(String[] args) {

        final String PROPERTIES_FILE_NAME = "success.properties";
        ResourceBundle resourceBundle = ResourceBundle
                .getBundle(PROPERTIES_FILE_NAME, Locale.ENGLISH);
        String name = resourceBundle.getString("bixiaopeng");
    }
}
```

在刚才选中要写入key/value的属性文件中写入了：

```
bixiaopeng=bixiaopeng
```

我们可以进一步进行编辑，以更好地对其进行国际化，如图 10-75 所示。

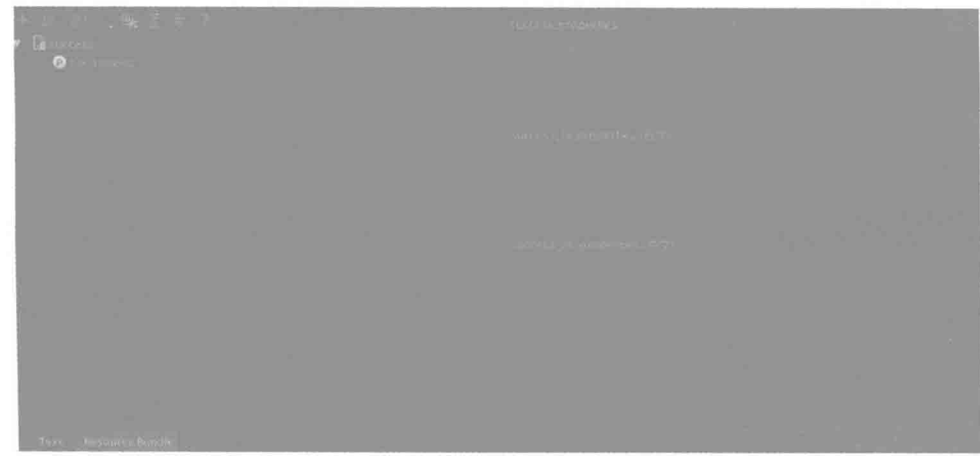


图 10-75

第 11 章　构　建

Android Studio默认使用Gradle来进行项目构建，Gradle是一个自动化构建工具，采用Groovy的Domain Specific Language（领域特定语言）来描述和控制构建逻辑。

本章将向大家介绍Android Studio中如何使用Gradle进行配置、编译和构建。

本章重要知识点 >>>>>>>>>>

- 如何配置 Gradle 文件;
- 如何执行 Gradle 任务;
- 如何编译和构建项目;
- 如何签名和打包。

11.1　认识 Gradle

11.1.1　Gradle是什么

Gradle是一个自动化构建工具，采用Groovy的Domain Specific Language（领域特定语言）来描述和控制构建逻辑。它的特点是语法简洁、可读性强、配置灵活等。基于Intellij IDEA社区版本开发的Android Studio天生支持Gradle构建程序。

Gradle使用指南请看https：//docs.gradle.org/current/userguide/userguide.html。

1. Groovy是什么

Groovy是一种基于JVM的敏捷开发语言，结合了Python、Ruby、Smalltalk的许多强大特性。Groovy 代码既能够与 Java 代码很好地结合，也能够用于扩展现有的代码。更多Groovy请参考http://www.groovy-lang.org/。

Gradle的特点如下：

- Gradle 支持多工程构建和局部构建。
- Gradle 支持远程或本地依赖管理：支持从远程 maven 仓库、nexus 私服、ivy 仓库以及本地仓库获取依赖。
- Gradle 与 Ant、Maven 兼容。
- Gradle 可轻松迁移：Gradle 适用于任何结构的工程，可以在同一个开发平台平行构建原工程和 Gradle 工程。
- Gradle 使用灵活：Gradle 的整体设计是以作为一种语言为导向的，而非成为一个严格死板的框架。
- Gradle 免费开源。

- Gradle 跟 IDE 集成得非常好。
- Gradle 可以更容易地集成到自动化构建系统。

2. Gradle中的Project（项目）、Task（任务）和Plugin（插件）

项目（project）是指构建产物（比如Jar包）或部署的产物（将应用程序部署到生产环境）。每个项目包含一个或多个任务。

任务（Task）是指不可分的最小工作单元，代表一个逻辑上较为独立的执行过程，比如编译、复制、打包。

Gradle只是提供了构建项目的一个框架，所有有用的特性都由Gradle**插件**提供，一个Gradle插件能够完成以下事项。

- 向项目中添加新任务。
- 为新加入的任务提供默认配置，这个默认配置会在项目中注入新的约定（如源文件位置）。
- 加入新的属性，可以覆盖插件的默认配置属性。
- 为项目加入新的依赖。

更多请参考：http://www.gradle.org/docs/current/userguide/standard_plugins.html项目、任务和插件的关系是什么？

项目代表要被构建的组件或整个项目，为任务提供了执行的上下文，而插件用来向项目中添加属性和任务。一个任务可以读取和设置项目的属性以完成特定的操作。

3. 如何配置Gradle构建

Gradle的构建肯定会包含下面这几个配置文件。

- Gradle 构建脚本（build.gradle）指定了一个项目及其任务。
- Gradle 属性文件（gradle.properties）用来配置构建属性。
- Gradle 设置文件（gradle.settings）对于只有一个项目的构建而言是可选的，如果我们的构建中包含多于一个项目，那么它就是必需的。因为它描述了哪一个项目参与构建。

每一个多项目构建都必须在项目结构的根目录中加入一个设置文件gradle.settings。

11.1.2 Gradle中依赖的仓库

1. 仓库是什么

顾名思义，仓库就是存放东西的。放什么东西呢？简单来说就是存放我们依赖的jar包。

2. Gradle支持的仓库

- Maven 仓库。
- Ivy 仓库。
- 平级目录仓库。

3. 如何在构建中加入这些仓库

使用Ivy仓库的人应该不多，这里就不多做介绍，重点放在maven仓库上。

在build.gradle中添加仓库的声明，从Maven仓库中获取依赖：

```
repositories {

    #1.从指定的远程 Maven 仓库中获取依赖
    maven {
        url "http://maven.helloword.net/repo"
    }

    #2.从指定的本地 Maven 仓库中获取依赖
    maven {
        url "file：///Users/bixiaopeng/mvn"
    }

    #3.从中央 Maven 仓库中获取依赖
    mavenCenter()

    #4.从新的中央远程仓库中获取依赖
    jcenter()

    #5.从本地仓库中获取依赖
    mavenLocal()

    #6.需要认证的库
    maven {
        credentials {
            username 'user'
            password 'password'
        }
        url "http://repo.helloword.com/maven2"
    }
}
```

4. Maven仓库的别名

为了更加方便地加入Maven仓库， Gradle为我们提供了 3 种别名。

- mavenCentral(): 表示从 Maven 中央仓库中获取依赖。地址为 http://repo1.maven.org/maven2。
- jcenter(): 一个新的远程中央仓库，兼容 Maven 中央仓库，而且性能更优。Gradle 默认使用 jcenter 作为仓库。jcenter 存放在 https://bintray.com/中。
- mavenLocal(): 表示从本地 Maven 仓库中获取依赖，本地地址为{user.home}/.m2/repository。

5. 从平级目录仓库中获取依赖

从本地目录中获取依赖，在build.gradle中添加：

```
repositories {
        //从当前项目的平级目录 lib 中获取依赖
```

```
        flatDir(dir: 'lib', name: 'libs directory')
        //从当前项目的平级目录 libA 和 libB 中获取依赖
        flatDir {
            dirs 'libA', 'libB'
            name = 'All dependency directories'
        }
    }
```

11.2 配置 Gradle 环境

本文以Mac上配置Gradle环境为例进行介绍。

1. 下载gradle

（1）下载地址：http://gradle.org/gradle- download/。

（2）下载当前最新版本gradle-3.1（写到本章时为3.1），如图 11-1 所示。

当然你也可以选择下载某一个历史版本，在页面右边Choose Vesion 中选择。下载到本地后解压。

DOWNLOAD GRADLE 3.1

The latest release of Gradle is version 3.1, released on *19th September 2016.*

MANUAL INSTALL

- Complete distribution (binaries, sources and offline documentation)
- Binary only distribution (no documentation or source code)
- Gradle source code (just the Gradle source code; not a usable Gradle installation)

图 11-1

2. 配置环境变量

（1）我的本地存放路径为/Volumes/warehouse/dev-tools/tools-jars/gradle-3.1。

（2）编辑.bash_profile：

```
vim ~/.bash_profile
```

（3）配置环境变量：

```
    export GRADLE_HOME=/Volumes/warehouse/dev-tools/tools-jars/gradle-3.1
export PATH=$PATH: $GRADLE_HOME/bin
```

（4）使配置立即生效：

```
source ~/.brash_profile
```

3. 查看配置是否成功

```
bixiaopeng@bixiaopengtekiMacBook-Pro ~$ gradle -v

------------------------------------------------------------
Gradle 3.1
------------------------------------------------------------

Build time:   2016-09-19 10:53:53 UTC
Revision:     13f38ba699afd86d7cdc4ed8fd7dd3960c0b1f97

Groovy:       2.4.7
Ant:          Apache Ant(TM) version 1.9.6 compiled on June 29 2015
JVM:          1.8.0_91 (Oracle Corporation 25.91-b14)
OS:           Mac OS X 10.11.2 x86_64
```

在电脑上不单独配置Gradle环境也没有关系，因为Android Studio中使用了Gradle Wrapper，它可以在我们没有安装Gradle的时候进行项目构建，下面我们会讲到。

提 示

Windows 和 Linux 上配置 Gradle 的开发环境方法差不多，这里就不一一介绍了。

提 示

macOS 和 Linux 上也可以使用 SDKMAN 来安装。

11.3 Gradle Wrapper

1. 认识Gradle Wrapper

（1）Gradle Wrapper 是什么

Gradle Wrapper可以理解为对Gradle的一层封装，使用它可以在没有安装Gradle的系统上使用Gradle来构建项目。

（2）如何做到

Gradle Wrapper通过两个脚本文件实现这一功能，一个是用于Windows的批处理文件gradlew.bat，一个是用于Linux和UNIX的Shell脚本文件gradlew。

使用Android Studio创建的项目默认为我们生成了Gradle Wrapper的文件结构，如图 11-2 所示。

图 11-2

在gradle/wrapper目录下有两个文件：gradle-wrapper.jar和gradle-wrapper.properties。gralde-wrapper.properties文件中声明了Gradle的版本和下载地址。

```
#Mon Aug 08 14:30:41 CST 2016
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\: //services.gradle.org/distributions/gradle-2.14.1-all.zip
```

在第一次使用gradlew进行项目构建的时候，Gradle Wrapper会自动下载gralde-wrapper.properties指定的Gradle版本。

提 示

通过gradlew执行Gradle构建跟直接使用Gradle是一样的，如果想直接使用Gradle构建需要先配置环境变量。

2. 初始化构建环境

在第一次使用gradlew进行项目构建的时候会对构建环境进行初始化，把Gradle的安装包、插件和相关依赖下载下来。在Terminal中输入命令：

```
bixiaopeng@bixiaopengtekiMacBook-Pro BaseBuildSample$  ./gradlew clean
Downloading https://services.gradle.org/distributions/gradle-2.14.1-all.zip
..........................................................................这是一个慢长的等待过程.

Unzipping
/Users/bixiaopeng/.gradle/wrapper/dists/gradle-2.14.1-all/8bnwg5hd3w55iofp
58khbp6yv/gradle-2.14.1-all.zip to
/Users/bixiaopeng/.gradle/wrapper/dists/gradle-2.14.1-all/8bnwg5hd3w55iofp
58khbp6yv
Set executable permissions for:
/Users/bixiaopeng/.gradle/wrapper/dists/gradle-2.14.1-all/8bnwg5hd3w55iofp
58khbp6yv/gradle-2.14.1/bin/gradle

…
BUILD SUCCESSFUL
```

首次执行命令行会先下载Gradle安装包，过程可能会有些慢。

11.4　查看和执行 Gradle 任务

11.4.1　查看当前项目支持的Gradle任务

使用./gradlew task来查看当前项目支持哪些Gradle任务：

```
bixiaopeng@bixiaopengtekiMacBook-Pro FirstApp$ ./gradlew task
: tasks

------------------------------------------------------------
All tasks runnable from root project(所有从项目根目录可运行的任务)
------------------------------------------------------------

Android tasks(Android 任务)
-------------
androidDependencies - 显示项目的 Android 依赖
signingReport - 显示每个变种版本的签名信息
sourceSets - 打印出所有在这个项目中定义的 source 集合

Build tasks(构建任务)
-----------
assemble - 编译并打出应用程序所有变种版本的包
assembleAndroidTest -编译并打出所有测试应用的包
assembleDebug - 编译并打出 Debug 版本的包
assembleRelease - 编译并打出 Release 版本的包
build - 执行所有检查并编译打包
buildDependents - 检查所有的依赖并编译打包
```

```
buildNeeded - 检查所有的依赖并编译打包
clean - 删除构建目录
compileDebugAndroidTestSources
compileDebugSources
compileDebugUnitTestSources
compileReleaseSources
compileReleaseUnitTestSources
mockableAndroidJar - 创建一个适用于单元测试的 android.jar 版本

Build Setup tasks (构建设置任务)
------------------
init - 初始化一个新的 Gradle 构建
wrapper - 生成 Gradle wrapper 文件

Help tasks(帮助任务)
----------
buildEnvironment - 显示项目根目录中声明的所有构建脚本的依赖
components - 显示项目根目录产生的组件
dependencies - 显示项目根目录中所有依赖的声明
dependencyInsight - 显示并洞察项目根目录中一个特殊的依赖关系
help - 显示帮助信息
model - 显示项目根目录的配置模型
projects - 显示项目根目录中的子项目
properties - 显示项目根目录的属性
tasks - 显示从项目根目录可以运行的任务(有些显示的任务可能属于子项目)

Install tasks(安装任务)
-------------
installDebug - 编译打包并安装 Debug 版本的包
installDebugAndroidTest - 编译打包并安装 Debug 版本的测试包到设备上
uninstallAll - 卸载所有版本的包
uninstallDebug - 卸载 Debug 版本的包
uninstallDebugAndroidTest - 从设备上卸载 Debug 版本的 Android 测试包
uninstallRelease - 卸载 Release 版本的包

Verification tasks(验证任务)
------------------
check - 运行所有检查
connectedAndroidTest - 在已连接的设备上安装所有 flavors(渠道包)并运行
instrumentation 测试
connectedCheck - 在当前已连接的设备上运行设备检测
connectedDebugAndroidTest - 在已连接的设备上安装并运行 Debug 版本的测试
deviceAndroidTest - 在所有提供的设备上安装并运行 instrumentation 测试
deviceCheck - 在所有提供的设备和测试服务器上运行设备检测
lint - 在所有变种版本上运行 lint 检测
lintDebug - 在 Debug 版本上运行 lint 检测
lintRelease - 在 Release 版本上运行 lint 检测
test - 在所有变种版本上运行单元测试
testDebugUnitTest - 在 Debug 版本上运行单元测试
testReleaseUnitTest - 在 Release 版本上运行单元测试
```

```
Other tasks (其他任务)
------------
jarDebugClasses
jarReleaseClasses
transformResourcesWithMergeJavaResForDebugUnitTest
transformResourcesWithMergeJavaResForReleaseUnitTest
```

想查看所有任务和更多详情，请运行：

```
gradlew tasks -all
```

想查看一个任务的更多详情，请运行：

```
gradlew help --task <task>

BUILD SUCCESSFUL

Total time: 6.717 secs
```

这个构建可以更快，请考虑使用Gradle守护：https：//docs.gradle.org/2.10/userguide/gradle_daemon.html

11.4.2 执行Gradle任务

执行命令："gradle + 任务名称"或者"./gradlew + 任务名称"。

Gradle 的 Android 插件提供了 4 个顶级任务：打包（assemble）、检测（check）、构建（build）、清理（clean）。当我们执行一个顶级任务的时候会同时执行与其依赖的任务。

例如，执行./gradlew assemble，就会把配置的所有构建类型（Build Types）全部打出来，默认的构建类型是Debug和Release，因此最起码会执行两个任务：gradlew assembleDebug和gradlew assembleRelease。如果有其他的构建类型，任务名应该是"gradlew assemble+构建类型名"。另外，执行构建（build）任务会执行检测（check）和打包（assemble）任务。

11.4.3 常用Gradle任务

（1）查看gradle版本：

```
$ ./gradlew -v
```

（2）编译并打出Debug版本的包：

```
./gradlew assembleDebug
```

（3）编译并打出Release版本的包：

```
./gradlew assembleRelease
```

（4）执行检查并编译打包：

```
./gradlew build
```

打出所有Release和Debug的包。

（5）删除build目录：

```
./gradlew clean
```

会把app下面的build目录删掉。

（6）编译打包并安装Debug版本的包：

```
./gradlew installDebug
```

（7）卸载Debug版本的包：

```
./gradlew uninstallDebug
```

（8）使用-info查看任务详情：

```
./gradlew uninstallDebug -info
```

11.4.4 Gradle工具窗口

如图 11-3 所示，Gradle工具窗口列出了当前项目和模块中支持的所有Gradle任务和运行配置，以方便我们快速操作。

1. Tasks

Tasks列表里的任务跟我们执行./gradlew task得到的任务列表是一样的。把光标放在某个任务上面会显示任务的描述信息，如图 11-4 所示。

图 11-3

图 11-4

双击任务即可执行。任务执行结果如图 11-5 所示。

单击Run工具栏左上角的切换按钮，可以在任务执行模式和文本式之间切换，如图 11-6 所示。

2. Run Configurations

如图 11-7 所示，Run Configurations列表中列出了项目中执行过的任务配置，这些配置都是执行任务时自动生成的。

图 11-5

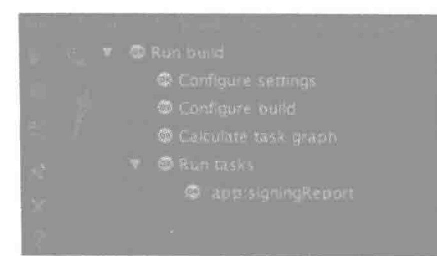


图 11-6

想再次执行的时候可以在配置列表中直接选择，如图 11-8 所示。

图 11-7

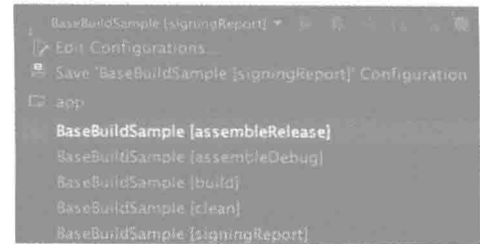


图 11-8

可以在Run/Debug Configurations中对这些配置进行修改，如图 11-9 所示。

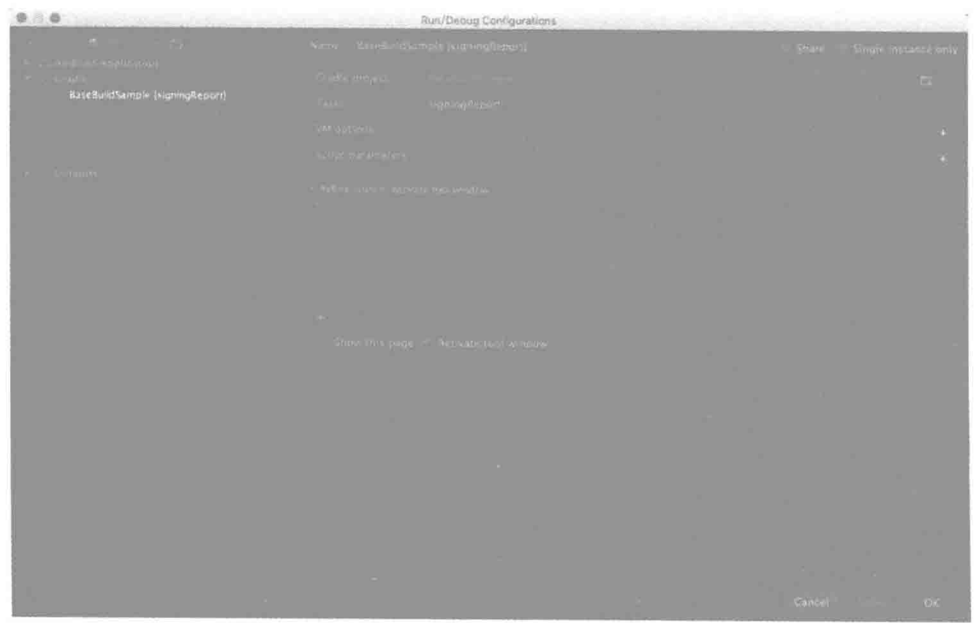


图 11-9

11.5 构建项目和模块

11.5.1 编译项目

当我们只想对修改过的文件进行编译时就会使用Make进行编译，可以指定对项目（Project）或模块（Module）进行编译。

编译项目会同时编译当前项目中所有的模块，如果代码较多，编译时间会比较长。如果想编译快一点需要指定编译某个模块。

菜单栏：Build→Make Project，如图 11-10 所示。

快捷键：command + F9（macOS）或者Ctrl + F9（Windows/Linux）

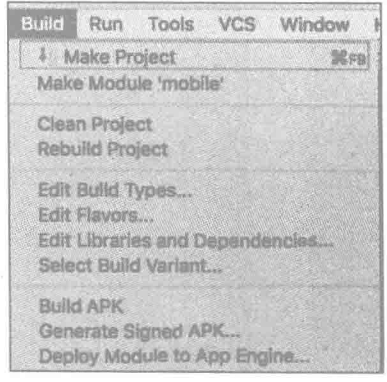


图 11-10

【实例演示】

项目中有两个模块：mobile和mylibrary。执行Make Project→在Gradle Console窗口中查看执行任务和结果：

```
Executing tasks: [: mobile: generateDebugSources, : mobile:
generateDebugAndroidTestSources, :mobile:prepareDebugUnitTestDependencies, :
mobile: mockableAndroidJar, : mobile: compileDebugSources, : mobile:
compileDebugAndroidTestSources, : mobile: compileDebugUnitTestSources, :
mylibrary: generateDebugSources, : mylibrary:
prepareDebugUnitTestDependencies, :mylibrary:mockableAndroidJar, :mylibrary:
generateDebugAndroidTestSources, :mylibrary:compileDebugSources, :mylibrary:
compileDebugUnitTestSources, : mylibrary: compileDebugAndroidTestSources]

.....此处省略 N 行

: mobile: preBuild UP-TO-DATE
: .....此处省略 N 行
: mobile: prepareComGoogleAndroidSupportWearable130Library UP-TO-DATE

: mylibrary: bundleDebug UP-TO-DATE
: ......此处省略 N 行
: mylibrary: compileDebugAndroidTestSources UP-TO-DATE

BUILD SUCCESSFUL

Total time: 4.542 secs
```

从上面的执行过程可以看出，Make Project同时编译了两个模块，即mobile和my library，用时 4.542 秒。

在启动 app 之前默认会先执行 Make 编译项目，如图 11-11 所示。

图 11-11

11.5.2 编译模块

前提条件：选中某个模块。

操作步骤：菜单栏→Build →Make Module '模块名'。

【实例演示】

项目中有两个模块：mobile和mylibrary。执行Make Module 'mobile'→在Gradle Console窗口中查看执行任务和结果：

```
Executing tasks: [: mobile: generateDebugSources, : mobile:
generateDebugAndroidTestSources, : mobile:
prepareDebugUnitTestDependencies, : mobile: mockableAndroidJar, : mobile:
compileDebugSources, : mobile: compileDebugAndroidTestSources, : mobile:
compileDebugUnitTestSources]

.....此处省略 N 行
: mobile: preBuild UP-TO-DATE
.....此处省略 N 行
: mobile: compileDebugUnitTestSources UP-TO-DATE

BUILD SUCCESSFUL

Total time: 1.543 secs
```

用时 1.543 秒。

11.5.3 设置自动编译项目

前面我们讲了编译项目是对新产生变化的文件进行一次编译，已经编译过的文件就不用重编译了，所以如果想加快编译速度，可以设置项目自动编译。

操作步骤：Android Studio→Preferences...→搜索Compiler→勾选Make project automatically，如图 11-12 所示。

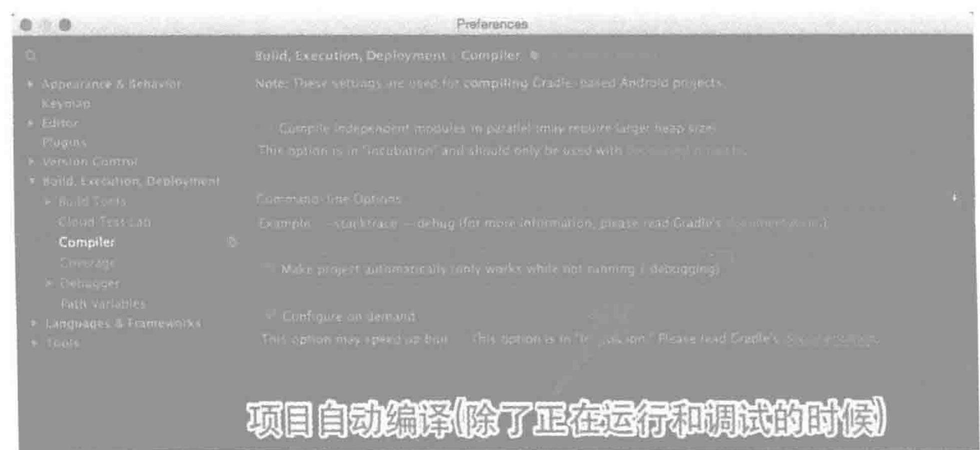


图 11-12

11.5.4 重新构建项目

操作步骤：菜单栏→Build→Rebuild Project（重新构建项目）。

【实例演示】

项目中有两个模块：mobile和mylibrary。执行Rebuild Project→在Gradle Console窗口中查看执行任务和结果：

```
Executing tasks: [clean, :mobile:generateDebugSources, :mobile:
generateDebugAndroidTestSources, :mobile:
prepareDebugUnitTestDependencies, :mobile:mockableAndroidJar, :mobile:
compileDebugSources, :mobile:compileDebugAndroidTestSources, :mobile:
compileDebugUnitTestSources, :mylibrary:generateDebugSources, :mylibrary:
prepareDebugUnitTestDependencies, :mylibrary:mockableAndroidJar, :mylibrary:
generateDebugAndroidTestSources, :mylibrary:compileDebugSources, :mylibrary:
compileDebugUnitTestSources, :mylibrary:compileDebugAndroidTestSources]

...
:mobile:clean
:mylibrary:clean
:mobile:preBuild UP-TO-DATE
....此处省略N行
:mobile:compileDebugUnitTestSources UP-TO-DATE
:mylibrary:preBuild UP-TO-DATE
....此处省略N行
```

```
:mylibrary:compileDebugAndroidTestSources

BUILD SUCCESSFUL

Total time: 16.053 secs
```

从上面的执行过程可以看出Rebuild Project（重新构建项目） 会先清空模块，然后重新对项目中的模块进行编译，用时 16.053 秒。

11.5.5 Make Project与Rebuild Project的区别

Make Project跟Rebuild Project都是执行相同的两个任务，最明显示的区别就是执行时间，从上面的例子中我们可以看出，Make Project用时 4.5 秒，而Rebuild Project用时则需要 16 秒。

为什么Rebuild Project会慢这么多？

因为Rebuild Project是对整个项目进行重新编译，不管之前有没有编译过，因此用时会比较长。而Make Project只是对新产生变化的文件进行一次编译，已经编译过的文件就不用重新编译了，所以用时比较少。

11.5.6 清理项目

清理项目会清空output目录下的文件，并重新编译项目。

操作步骤：菜单栏→Build→Clean Project（清理项目）。

【实例演示】

项目中有两个模块：mobile和mylibrary。执行Clean Project→在Gradle Console窗口中查看执行任务和结果：

```
Executing tasks: [clean, :mobile:generateDebugSources, :mobile:
prepareDebugUnitTestDependencies, :mobile:mockableAndroidJar, :mobile:
generateDebugAndroidTestSources, :mylibrary:generateDebugSources, :mylibrary:
prepareDebugUnitTestDependencies, :mylibrary:mockableAndroidJar, :mylibrary:
generateDebugAndroidTestSources]
...
BUILD SUCCESSFUL
Total time: 12.077 secs
```

从实例中可以看出，清理项目时先执行了clean任务，又执行了编译任务。

11.6 Gradle Script

我们把项目查看模式切换成Android，所有的文件会通过类型进行归类，这个并不是实际在电脑中的文件结构，如果想看实际的物理结构要切换到Project。

切换成Android可以查看所有的Gradle Script，如图 11-13 所示。

在Gradle Scripts列表中，每个文件后面都有一个灰色字体描述。

- build.gradle：项目构建配置文件。
- build.gradle：模块构建配置文件。
- gradle-wrapper.properties：gradle wrapper 配置文件。
- proguard-rules.pro：混淆规则配置文件。
- graddle.properties：gradle 配置文件。
- settings.gradle：项目全局配置文件。
- local.properties：SDK、NDK 配置文件。

切换到Project模式可以查看各个文件之间的关系，如图 11-14 所示。

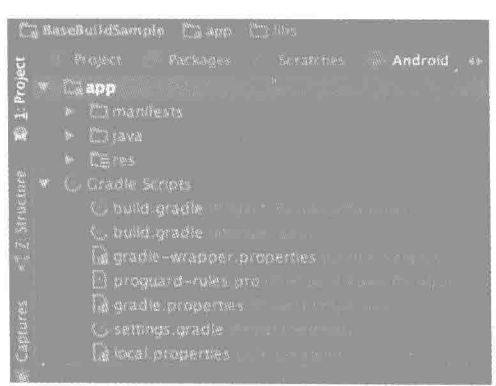


图 11-13

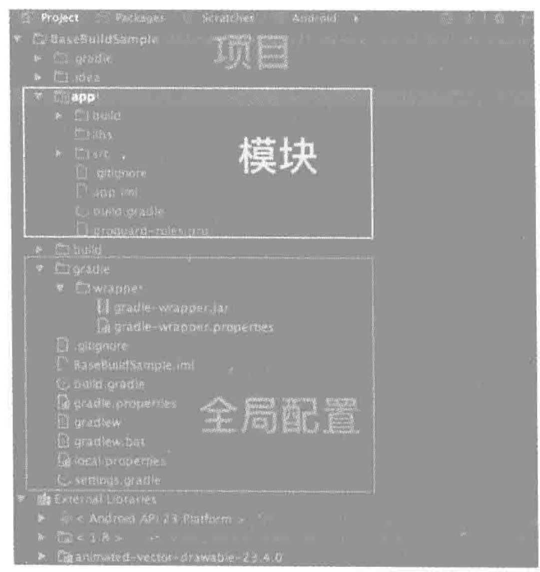


图 11-14

项目中有一个app模块，这个模块是一个Android应用程序，有自己的构建脚本和混淆配置文件。

项目根目录下的脚本文件是针对它所依赖模块的全局配置。下面我们详细介绍一下每个文件的作用。

11.6.1 Gradlew配置文件gradle-wrapper.properties

gradle-wrapper.properties 是gradle wrapper的配置文件。默认的gradle-wrapper.properties文件内容如下：

```
#Mon Dec 28 10：00：20 PST 2015
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\：
//services.gradle.org/distributions/gradle-2.10-all.zip
```

一般这个文件是不用动的，除非想手动指定Gradle的版本，可以修改distributionUrl。也可以在Project Structure→Project中设置Gradle version，如图 11-15 所示。

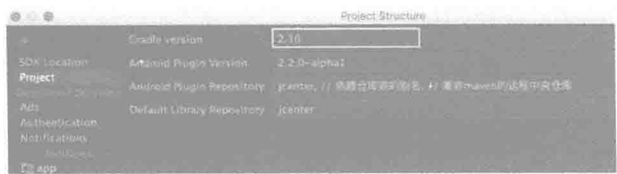


图 11-15

11.6.2 项目全局配置文件settings.gradle

默认的settings.gradle文件内容如下：

```
include ': app'
```

settings.gradle是项目的全局配置文件，主要声明一些需要加入构建的模块，本例中只有一个模块：app。

如果新增一个模块需要在这里添加配置，通过Android Studio添加依赖的module会自动在这里添加相应的配置。

包含多个模块的格式是这样的：

```
include ': app', ': other-module-name'
```

如果包含的是某个目录下的模块，格式是这样的：

```
include ': app',  ': dir-name: other-module-name'
```

11.6.3 本地属性配置文件local.properties

默认的local.properties文件内容如下：

```
# 这个文件是由 Android Studio 自动生成的，不要修改这个文件——即使改了也会被自动擦除。
# 这个文件不能被添加到版本控制系统中，因为它包含的信息只针对你本地系统的配置。
# 本地的 SDK 仅被用于 gradle 构建。
#Fri May 20 16: 18: 51 CST 2016
ndk.dir=/../sdk/ndk-bundle
sdk.dir=/../sdk
```

我们可以在Project Structure→SDK Location中设置SDK和NDK的路径，如图 11-16 所示。修改后会同步到local.properties文件中。

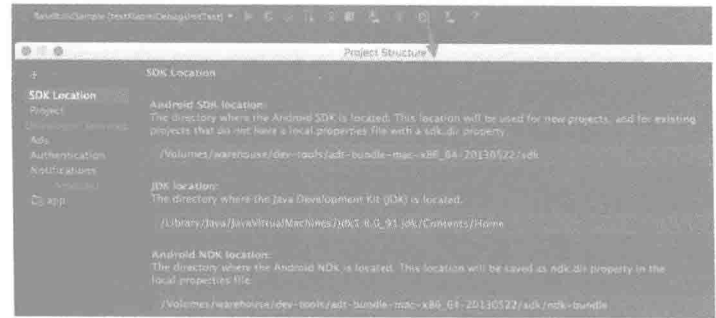


图 11-16

11.6.4 Gradle配置文件gradle.properties

gradle.properties是Gradle的配置文件，build.gradle通过读取这个文件配置的参数来进行相应的构建。

默认的gradle.properties文件内容如下：

```
# 项目的 Gradle 设置。


# 关于如何配置构建环境的更多详情，请访问：
# http://www.gradle.org/docs/current/userguide/build_environment.html

# 配置守护进程的 JVM 参数。
# 该设置用于调整内存。
# 默认值：-Xmx10248m -XX：MaxPermSize=256m
# org.gradle.jvmargs=-Xmx2048m -XX：MaxPermSize=512m -XX：
+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8

# 配置完毕后，Gradle 将在并行模式下运行（并行编译）
# 只应在解耦项目中使用此选项，更多详情请访问：
# http://www.gradle.org/docs/current/userguide/multi_project_
builds.html#sec：decoupled_projects
# 使用并行编译
# org.gradle.parallel=true
# 编译时使用守护进程
# org.gradle.daemon=true
```

可以在这里设置Gradle的代理。

在gradle.properties中添加：

```
systemProp.http.proxyHost=Proxy Server
systemProp.http.proxyPort=Proxy port

# 设置代理的用户名和密码，如果没有就不用设置
systemProp.http.proxyUser=User
systemProp.http.proxyPassword=password

# 设置不需要代理的地址，多个地址使用或'|'隔开
systemProp.http.nonProxyHosts=*.baidu.com|*.taobao.com
```

11.6.5 代码混淆规则配置文件proguard-rules.pro

如果想在打包的时候进行代码混淆，就需要在proguard-rules.pro中配置代码混淆规则，如图 11-17 所示。

图 11-17

常用的代码混淆规则：

```
# 在这里添加项目的代码混淆规则
# 混淆规则请参考：http://proguard.sourceforge.net/index.html#manual/usage.html

##################### 一般使用默认 #####################

# 不使用大小写混合类名，混淆后的类名为小写
-dontusemixedcaseclassnames
# 混淆第三方库
-dontskipnonpubliclibraryclasses
# 混淆时记录日志，有助于排查错误
-verbose
# 代码混淆使用的算法
-optimizations !code/simplification/arithmetic, !code/simplification/cast,
!field/*, !class/merging/*
# 代码混淆压缩比，值在 0~7 之间，默认为 5
-optimizationpasses 5
# 优化时允许访问并修改有修饰符的类和类的成员
-allowaccessmodification

##################### 不混淆 #####################

# 这些类不混淆
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgent
-keep public class * extends android.preference.Preference
-keep public class * extends android.support.v4.app.Fragment
-keep public class * extends android.support.v4.app.DialogFragment
-keep public class * extends com.actionbarsherlock.app.SherlockListFragment
-keep public class * extends com.actionbarsherlock.app.SherlockFragment
-keep public class * extends
com.actionbarsherlock.app.SherlockFragmentActivity
-keep public class * extends android.app.Fragment
-keep public class com.android.vending.licensing.ILicensingService
```

```
# Native 方法不混淆
-keepclasseswithmembernames class * {
 native <methods>;
}

# 自定义组件不混淆
-keep public class * extends android.view.View {
 public <init>(android.content.Context);
 public <init>(android.content.Context, android.util.AttributeSet);
 public <init>(android.content.Context, android.util.AttributeSet, int);
 public void set*(...);
}

# 自定义控件类和类的成员不混淆(所有指定的类和类成员是要存在的)
-keepclasseswithmembers class * {
 public <init>(android.content.Context, android.util.AttributeSet);
}

# 同上
-keepclasseswithmembers class * {
 public <init>(android.content.Context, android.util.AttributeSet, int);
}

# 自定义控件类不混淆
-keepclassmembers class * extends android.app.Activity {
 public void *(android.view.View);
}

# 枚举类不被混淆
-keepclassmembers enum * {
 public static **[] values();
 public static ** valueOf(java.lang.String);
}

# android.os.Parcelable 的子类不混淆
-keep class * implements android.os.Parcelable {
 public static final android.os.Parcelable$Creator *;
}

# 资源类不混淆
-keepclassmembers class **.R$* {
 public static <fields>;
}

#################### 第三方库不混淆 ####################

# 保留第三方库 android.support.v4 不被混淆
-keep class android.support.v4.app.** { *; }
-keep interface android.support.v4.app.** { *; }
```

```
# 打包时忽略警告
-dontwarn android.support.**

# 如果你的项目中使用了第三方库，需要参考官方文档的说明来进行混淆配置
# 例如，百度地图的配置，可参考：http://developer.baidu.com/map/sdkandev-question.htm
#-keep class com.baidu.** { *; }
#-keep class vi.com.gdi.bgl.android.**{*;}

# 例如，支付宝的混淆，可参考：https://doc.open.alipay.com/doc2/detail.htm?treeId=59&articleId=103683&docType=1
#-libraryjars libs/alipaySDK-20150602.jar
#
#-keep class com.alipay.android.app.IAlixPay{*;}
#-keep class com.alipay.android.app.IAlixPay$Stub{*;}
#-keep class com.alipay.android.app.IRemoteServiceCallback{*;}
#-keep class com.alipay.android.app.IRemoteServiceCallback$Stub{*;}
#-keep class com.alipay.sdk.app.PayTask{ public *;}
#-keep class com.alipay.sdk.app.AuthTask{ public *;}
```

混淆规则配置文件配置好以后，需要在build.gradle中开启混淆，如图 11-18 所示。

图 11-18

11.6.6 项目构建配置文件build.gradle

Project：build.gradle用来配置项目的构建任务。默认的build.gradle内容如下：

```
//项目构建文件，可以到各子项目/模块添加常用的配置选项

buildscript {
    //Android 插件从这个仓库中下载
    repositories {
        jcenter() // 依赖仓库源的别名，兼容 maven 的远程中央仓库
    }

    //依赖
    dependencies {
        // android gradle 插件
        classpath 'com.android.tools.build: gradle: 2.2.0-alpha1'

        // 提示：
```

```
        // 请不要在此处添加应用程序依赖;它们应该在单个Module(模块)build.gradle文件中
           添加
        // 这里添加的应该只是Project的依赖
    }
}

//此处配置Project中默认的仓库源，包括每个module的依赖
//这样每个module就不用单独配置仓库了
allprojects {
    repositories {
        jcenter()
    }
}

// 打包前执行clean任务
// 任务类型是 Delete
// clean任务就是删除项目根目录下的build目录(build为输出目录)
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

提 示

buildscript 中的 repositories 指定 Android 插件的仓库源。
allprojects 中的 repositories 指定整个 Project 中默认的仓库源。

在Project Structure中设置

我们可以在Project Structure→Project中设置Gradle和Android插件的版本，以及Android插件和默认第三方库的仓库源。Project Structure和Project build.gradle对应的关系如图 11-19 所示。

图 11-19

11.6.7 模块构建配置文件build.gradle

Module：build.gradle用来配置模块的构建任务。默认的build.gradle文件内容如下：

```
//插件:
//这个 module 是一个 android 程序,使用 com.android.application
//如果是 android 库,应该使用 com.android.library

apply plugin: 'com.android.application'

android {//android 程序构建需要配置的参数

    //编译使用的 SDK 版本
    compileSdkVersion 23
    //buildtool 版本
    buildToolsVersion "23.0.2"

    defaultConfig {//默认配置

         applicationId "com.wirelessqa.basebuildsample" //apk 包名

        //最小 SDK 版本
         minSdkVersion 16

        //目标 SDK 版本
         targetSdkVersion 23

        //version code
        versionCode 1

        //应用程序的版本
        versionName "1.0"
       //android 单元测试 test runner
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }

    //构建类型,此处配置 debug 和 release 版本的一些参数,比如混淆、签名配置
    buildTypes {

        //release 版本的配置
        release {

            //是否开启混淆
            minifyEnabled false

            //指定混淆文件及混淆规则配置文件的位置
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

//模块依赖
```

```
dependencies {

    //编译依赖 libs 目录下所有 jar 包
    compile fileTree(dir: 'libs', include: ['*.jar'])

    //编译依赖 appcompat 库
    compile 'com.android.support: appcompat-v7: 23.4.0'
}
```

模块构建配置文件build.gradle也可以在项目结构中配置，下面我们详细介绍一下。

11.7 在项目结构中配置模块构建

Project Structure用来配置项目和模块的各种构建参数和属性，前面我们已经介绍过了SDK Location和Project的配置，这一节主要介绍模块构建的一些配置。

菜单栏：File→Project Structure

快捷键：command + ; （macOS）或者Ctrl + Shift + Alt + S（Windows/Linux）

通过菜单栏或快捷键操作后打开Project Structure对话框，工具栏如图 11-20 所示。

Modules用来设置模块构建配置文件中的属性、签名、渠道特性、构建类型和依赖。

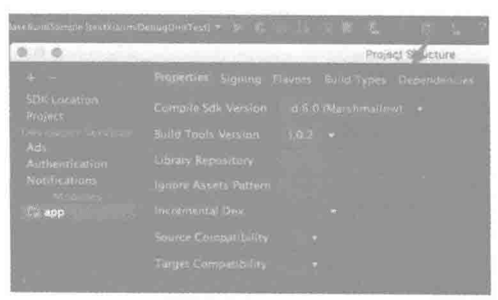


图 11-20

11.7.1 配置应用程序属性

- Compile Sdk Version: 指定 Android 的编译版本。

对应build.gradle文件中的参数是：

```
compileSdkVersion 23
```

- Build Tools Version: 指定构建工具的版本。

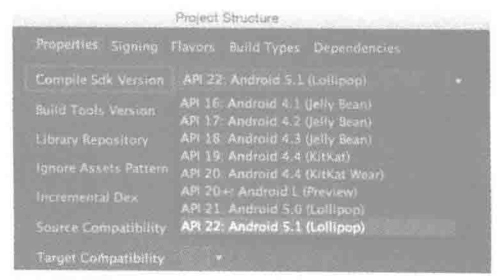
对应build.gradle文件中的参数是：

```
buildToolsVersion "23.0.2"
```

如图 11-21 所示，SDK编译版本和构建工具的版本都是我们已经下载到本地的，如果本地没有就不会出现在选择列表中。

- Library Repository: 指定依赖的仓库源。

默认Project的build.gradle中已经指定了全局的仓库源，默认都是jcenter，因此这里可以为空。如果重新指定，将会覆盖project build.gradle中的配置。如果此处指定从maven中央仓库下载依赖，相应的module build.gradle文件中会新增下面的参数：



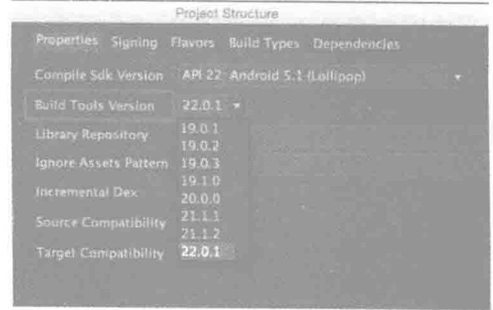


图 11-21

```
repositories {
    mavenCenter
}
```

如果从Project Structure中删除配置的参数，相应的build.gradle文件中也会删除。

- Ignore Assets Pattern：指定构建打包时要忽略的文件。

Assets是用于存放资源文件的。使用aapt工具打包时，如果配置了打包时要忽略的文件，appt在打包资源文件的时候就不会把这个文件打包进去。例如，输入要忽略的文件名为bixiaopeng，相应的build.gradle文件中会新增下面的参数：

```
aaptOptions {
    ignoreAssetsPattern 'bixiaopeng'
}
```

- Incremental Dex：增量 Dex 打包。

开启此功能可以提升编译打包的速度，因为是增量打包而不是全量。

补充：Dex将.class文件转为Dalvik VM能识别的.dex文件。

如果打开增量打包Dex为true，相应的build.gradle文件中会增加下面的参数：

```
dexOptions {
    incremental true
}
```

- Source Compatibility：指定资源版本。
- Target Compatibility：指定目标版本。

修改后在build.gradle文件中会增加下面的参数：

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_7
    targetCompatibility JavaVersion.VERSION_1_7
}
```

执行结果如图 11-22 所示。

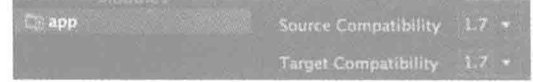


图 11-22　属性中对话的配置

11.7.2　配置应用程序签名

如图 11-23 所示，如果我们在这里配置了签名，相应地在module app的build.gradle文件中会自动添加下面的配置：

```
// 签名配置
    signingConfigs {
        MySigning {
            keyAlias 'myandroid'
            keyPassword '123456'
            storeFile file('/Users/bixiaopeng/myandroid.jks')
```

```
        storePassword '123456'
    }
}
```

图 11-23

对比一下使用maven打包，pom里的签名配置：

```
<!-- Sign apk -->
<!--签名文件路径-->
<sign.keystore>/Users/bixiaopeng/release.keystore</sign.keystore>
<!--签名文件别名-->
<sign.alias>laobi</sign.alias>
<!--签名文件别名密码-->
<sign.keypass>222222</sign.keypass>
<!--签名文件密码-->
<sign.storepass>222222</sign.storepass>
```

11.7.3 配置应用程序特性

多渠道打包要在Flavors中配置相应的属性，如图 11-24 所示。

图 11-24

- Name：Flavor 的名字，如我们常用的渠道 xiaomi、baidu。
- Min Sdk Version：向下兼容的最小 SDK 版本。
- Application Id：应用程序的包名。
- Proguard File：指定混淆文件路径，如果不指定会使用默认的。

- Signing Config：指定签名文件。签名的文件在 Signing 中设置。
- Target Sdk Version：目标 SDK 版本。
- Test Instrumentation Runner：指定 Test Runner。
- Test Application Id：测试应用的 ID。
- Version Code：应用程序的版本号，用于升级。
- Version Name：应用程序的版本名称。
- Version Name Suffix：应用程序版本名称的后缀。

默认配置的Flavor参数与build.gradle文件中的参数一一对应，如图 11-25 所示。

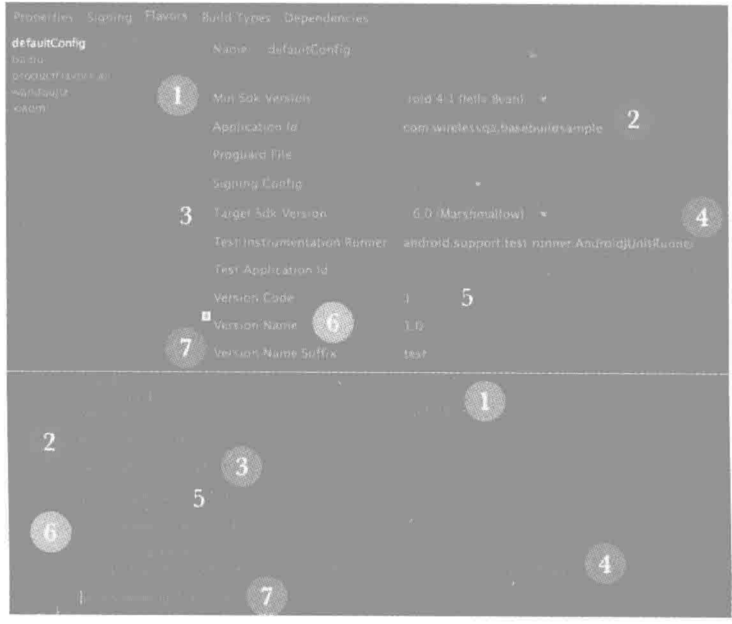


图 11-25

新增一个Flavor，所有的参数都允许重新设置，如果不设置就都使用defaultConfig的配置，如图 11-26 所示。

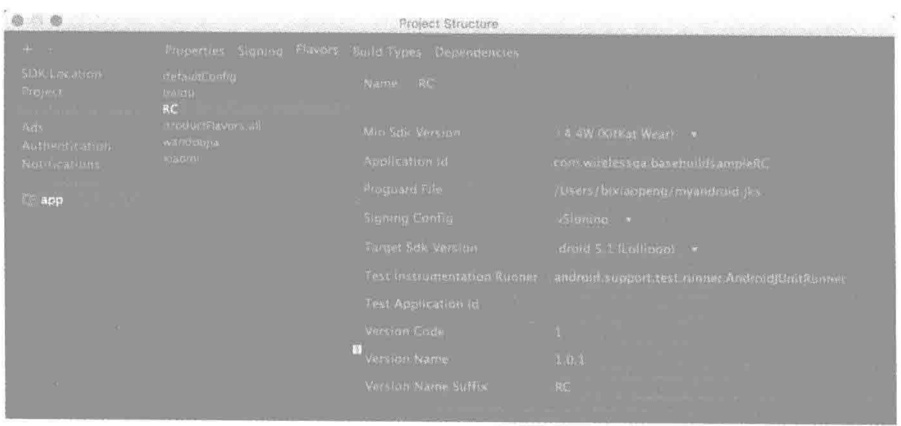


图 11-26

```
    // 产品特性
    productFlavors {
        xiaomi {} //渠道名 name 为 xiaomi
        baidu {
        }
        wandoujia {}
        // 自动替换 AndroidManifest.xml 中的渠道号
        productFlavors.all { flavor ->
            flavor.manifestPlaceholders = [CHANNEL_ID: name]
        }
        RC {
            minSdkVersion 20
            applicationId 'com.wirelessqa.basebuildsampleRC'
            proguardFile '/Users/bixiaopeng/myandroid.jks'
            signingConfig signingConfigs.MySigning
            targetSdkVersion 22
            testInstrumentationRunner 'android.support.test.runner.
AndroidJUnitRunner'
            versionCode 1
            versionName '1.0.1'
            versionNameSuffix 'RC'
        }
    }
```

针对不同的APP分发渠道，我们可以定义不同的productFlavors（产品特性）；也可以定义APP开发不同阶段的版本，比如内测版本、灰度版本、正式版本；还可以为不同的版本指定不同的ApplicationId（包名），这样在同一个设备上可以同时安装两个版本，方便测试。有了上面的配置，在执行打包命令的时候就会打出不同渠道的包。

11.7.4 配置应用程序构建类型

构建类型 Build Types非常重要，如图 11-27 所示。在这里可以配置构建版本的一些非常重要的参数，默认有两个构建版本：debug和release。

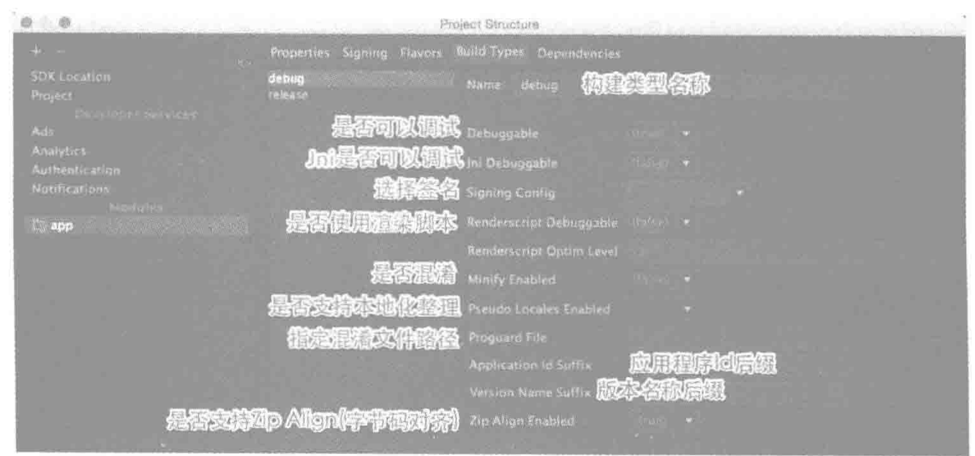


图 11-27

默认的debug和release的区别只有Debuggable（是否可调试）：debug默认可调试，release默认不可以调试。

- Name：构建类型名称。
- Debuggable：是否可以调试。
- Jni Debuggable：Jni 是否可以调试。
- Signing Config：指定签名，同样是在 signing 中配置的，若为空则不签名，打出来的包也是未签过名的。
- Renderscript Debuggale：是否使用渲染脚本（一种低级的高性能编程语言，用于 3D 渲染和处理密集型计算）。
- Renderscript Optim Level：Rendescript 版本。
- Minify Enabled：是否混淆。
- Pseudo Locales Enabled：是否支持本地化整理。
- Proguard File：指定混淆文件路径。
- Application Id Suffix：应用程序 Id 后缀。
- Version Name Suffix：版本名称后缀。
- Zip Align Enabled：是否支持 Zip Align（字节码对齐）。

新增一个构建类型，如图 11-28 所示。

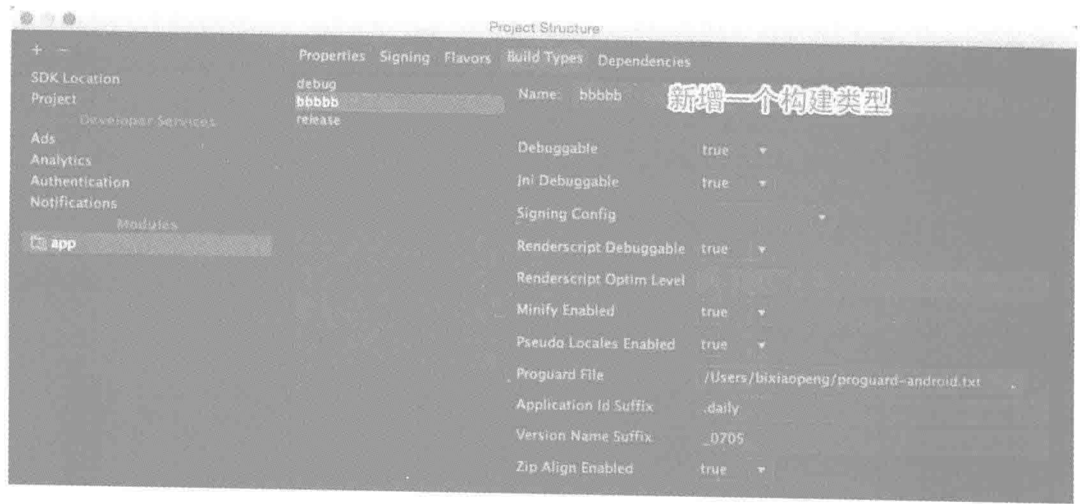


图 11-28

修改后build.gradle文件中同步修改为：

```
buildTypes {
     release {
         minifyEnabled false
         proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
     }
     bbbbb {
         debuggable true
```

```
            jniDebuggable true
            renderscriptDebuggable true
            minifyEnabled true
            pseudoLocalesEnabled true
            proguardFile '/Users/bixiaopeng/proguard-android.txt'
            applicationIdSuffix '.daily'
            versionNameSuffix '_0705'
            zipAlignEnabled true
        }
    }
```

可以指定为某个构建类型打包。

- 打出 Debug 的包：./gradlew assembleDebug。
- 打出 Release 的包：./gradlew assembleRelease。
- 打出 Bbbbb 的包：./gradlew assembleBbbbb。

试着打出Bbbbb这种类型的包：

```
$ ./gradlew assembleBbbbb
: app: preBuild UP-TO-DATE
: app: preBaiduBbbbbBuild UP-TO-DATE
: app: checkBaiduBbbbbManifest

...
: app: packageXiaomiBbbbb
: app: assembleXiaomiBbbbb
: app: assembleBbbbb

BUILD SUCCESSFUL

Total time: 25.301 secs
```

在app/build/output/apk目录下就会打出相应的apk包。

解析apk，查看包名和版本号跟我们配置的一致。

```
~$ aapt d badging
/Volumes/FirstApp/app/build/outputs/apk/app-baidu-bbbbb-unsigned.apk
package: name='com.baidu.firstapp.daily' versionCode='1'
versionName='1.0_0705'
sdkVersion: '8'
targetSdkVersion: '22'
....
```

提 示

Project Structure 中没有的配置但 buildTypes 中支持的属性还有很多，例如：

```
// 显示 Log
buildConfigField "boolean", "LOG_DEBUG", "true"
// 移除无用的 resource 文件，minifyEnabled 必须为 true
shrinkResources true
```

11.7.5 配置应用程序依赖

我们即可以在这里添加模块中依赖的jar包、文件和模块，还可以配置它们的作用范围。默认配置中的依赖，如图 11-29 所示。

build.gradle中配置的依赖在上面已经介绍过，这里简单提一下，这两个依赖配置的意思是指定编译时需要依赖libs目录下所有的jar文件和一个android组件。

图 11-29

添加依赖，如图 11-30 所示。

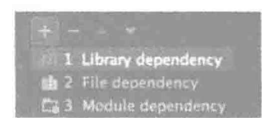


图 11-30

添加一个jar包，如图 11-31 所示。

可以输入关键字或者完整的坐标来搜索jar包。例如，输入junit，单击搜索按钮后结果如图 11-32 所示。

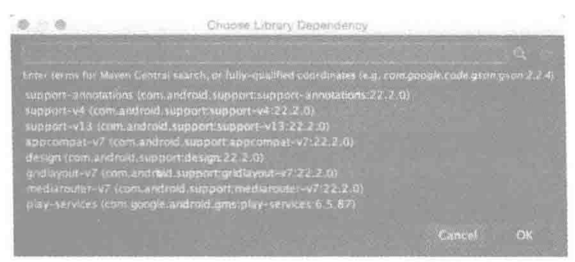


图 11-31

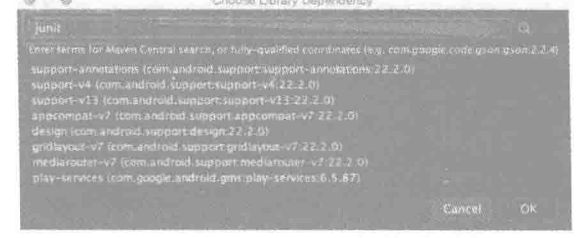


图 11-32

确认后，此依赖被添加，默认的作用域是compile（编译），可以修改为Test compile，如图 11-33 所示。

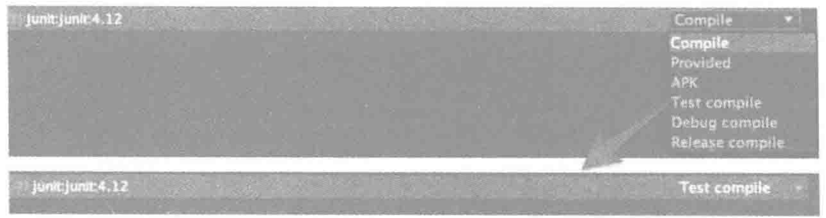


图 11-33

在build.gradle中会相应新增配置：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support: appcompat-v7: 22.+'
    //此为新增依赖
    androidTestCompile 'junit: junit: 4.12'
}
```

11.8 签名和打包

Android系统要求应用程序必须经过签名才能够安装到系统中，签名就是在应用程序特定的字段中写入标识信息，Android系统通过签名判断你是否是这个应用程序的开发者。

签名相同的应用程序：

- 能够正常地覆盖安装旧版本
- 能够允许代码和数据共享

默认签名证书

Android SDK工具会自动生成一个调试用的签名证书debug.keystore。默认存放位置如下：

- macOS/Linux: /Users/你的用户名/.android/debug.keystore
- Windows: C:/Documents and Settings/你的用户名/.android/debug.keystore

当我们打debug包时，默认使用debug.keystore进行签名。debug包可以在真机和模拟器上运行和调试，但不能发布到应用市场。

不同电脑上生成的调试签名证书可能都不一样，签名不一样会导致应用程序无法覆盖安装，就相当于应用程序无法升级。如果想避免这样的事情发生，对外发布时要使用统一的签名。

创建签名证书的一般步骤：

01 创建签名文件（如果已经有了，就不用创建了）。

02 Build Types 中指定签名文件。

03 打包。

11.8.1 创建签名证书

操作步骤：

01 进入创建签名文件窗口：Build → Generate Signed Apk → 选择 Module → Next → Create new... →弹出创建签名文件窗口，如图 11-34 所示。

图 11-34

- 存储（Store）。
  - Key store path：签名证书存储路径。
  - Password：密码。
  - Confirm：再次确认密码。
- 密钥（Key）。
  - Alias：别名。
  - Password：密码。
  - Confirm：再次确认密码。
  - Validity（years）：证书的有效期，默认是 25 年。
- 证书（Certificate）。
  - First and Last Name：你的姓名。
  - Organizational Unit：你的组织单位。
  - Organization：你的组织。
  - City or Locality：你所在的城市或地区。
  - State or Province：你所在的州或省。
  - Country Code （XX）：国家代码。

02 输入签名证书需要的信息，如图 11-35 所示。

03 单击【OK】按钮后签名证书 myandroid.jks 就创建成功了。

接下来会回到签名证书配置窗口，如图 11-36 所示。

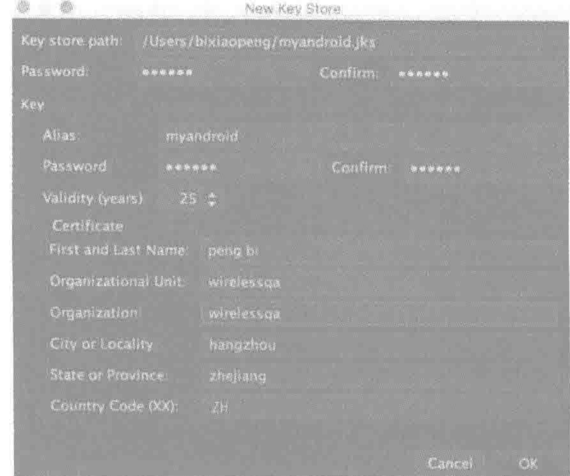


图 11-35

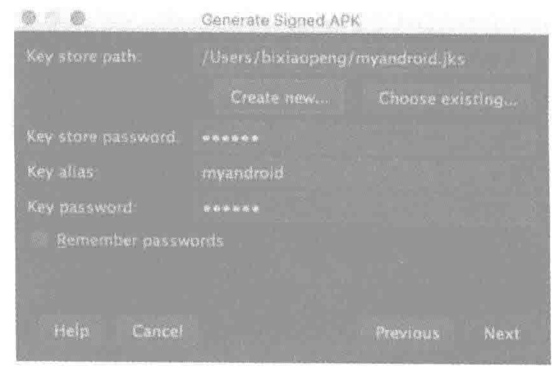


图 11-36

这里自动选择了我们刚才创建好的签名证书，并解析出了相关的参数，如果想生成签名的APK就单击【Next】按钮继续。

11.8.2 生成签名的APK

假设已经有了签名证书，然后再生成签名的APK，应该是怎样的操作步骤呢？

01 签名证书配置窗口：Build → Generate Signed Apk → 选择 Module → Next，然后弹出签名证书配置窗口，如图 11-37 所示。既可以单击 【Choose existing】来选择一个已存在的签名证书，也可以使用上次使用过的签名证书。

02 输入选择证书的存储密码和密钥密码，如果不想每次都输入密码，就勾选【Remember passwords】，如图 11-38 所示。

03 单击【Next】按钮，要求输入 Master Password，如图 11-39 所示。

图 11-37

图 11-38

Master Password是用来保护我们存储在Android Studio数据库中的密码的。如果忘记了密码是无法进行到下一步的，如图 11-40 所示。

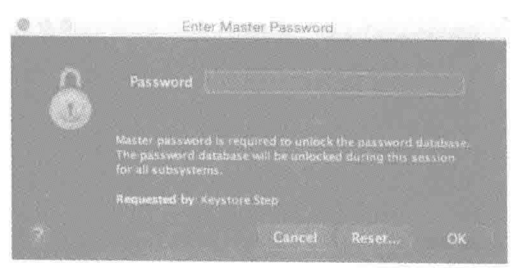


图 11-39

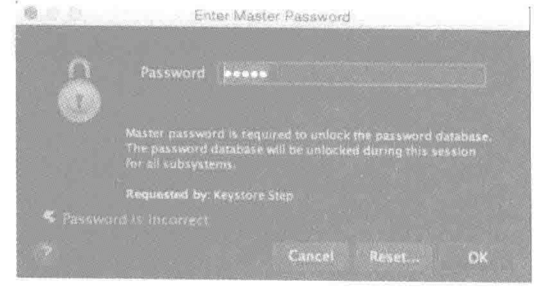


图 11-40

怎么解决这个问题呢？

方法一：在当前窗口中重置密码。

单击【Reset...】→ 弹出重置Master密码窗口→ 输入新的密码→OK，如图 11-41 所示。单击【Yes】按钮确认重置操作，如图 11-42 所示。

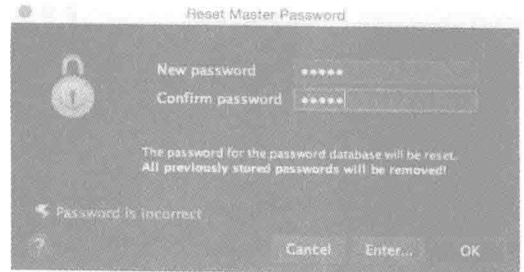


图 11-41

图 11-42

方法二：到偏好设置中重置密码，如图 11-43 所示。

如果记得密码，直接输入密码。Master password只会在我们第一次使用时要求输入。

图 11-43

04 配置应用程序构建类型和存储位置，如图 11-44 所示。

- APK Destination Folder: 指定应用程序存储位置（默认放在 app 根目录下）。
- Build Type: 构建类型。
- Flavors: 渠道（没有就使用默认配置）。

05 单击【Finish】按钮之后开始打包任务。

图 11-44

在Gradle Console工具窗口可以查看任务执行的过程：

```
Executing tasks: [: app: assembleRelease]

Configuration on demand is an incubating feature.
Incremental java compilation is an incubating feature.
: app: preBuild UP-TO-DATE
: app: preReleaseBuild UP-TO-DATE
: app: checkReleaseManifest
: app: preDebugBuild UP-TO-DATE
: app: prepareComAndroidSupportAnimatedVectorDrawable2340Library UP-TO-DATE
: app: prepareComAndroidSupportAppcompatV72340Library UP-TO-DATE
: app: prepareComAndroidSupportConstraintConstraintLayout100Alpha1Library
UP-TO-DATE
: app: prepareComAndroidSupportDesign2340Library UP-TO-DATE
: app: prepareComAndroidSupportRecyclerviewV72340Library UP-TO-DATE
: app: prepareComAndroidSupportSupportV42340Library UP-TO-DATE
: app: prepareComAndroidSupportSupportVectorDrawable2340Library UP-TO-DATE
```

```
: app: prepareReleaseDependencies
: app: compileReleaseAidl UP-TO-DATE
: app: compileReleaseRenderscript UP-TO-DATE
: app: generateReleaseBuildConfig UP-TO-DATE
: app: mergeReleaseShaders UP-TO-DATE
: app: compileReleaseShaders UP-TO-DATE
: app: generateReleaseAssets UP-TO-DATE
: app: mergeReleaseAssets UP-TO-DATE
: app: generateReleaseResValues UP-TO-DATE
: app: generateReleaseResources UP-TO-DATE
: app: mergeReleaseResources UP-TO-DATE
: app: processReleaseManifest UP-TO-DATE
: app: processReleaseResources UP-TO-DATE
: app: generateReleaseSources UP-TO-DATE
: app: incrementalReleaseJavaCompilationSafeguard UP-TO-DATE
: app: compileReleaseJavaWithJavac UP-TO-DATE
: app: compileReleaseNdk UP-TO-DATE
: app: compileReleaseSources UP-TO-DATE
: app: lintVitalRelease
: app: prePackageMarkerForRelease
: app: processReleaseJavaRes UP-TO-DATE
: app: transformResourcesWithMergeJavaResForRelease UP-TO-DATE
: app: transformClassesAndResourcesWithProguardForRelease UP-TO-DATE
: app: transformClassesWithDexForRelease UP-TO-DATE
: app: mergeReleaseJniLibFolders UP-TO-DATE
: app: transformNative_libsWithMergeJniLibsForRelease UP-TO-DATE
: app: validateExternalOverrideSigning
: app: packageRelease UP-TO-DATE
: app: assembleRelease

BUILD SUCCESSFUL

Total time: 4.684 secs
```

打包成功后在IDE的右上角弹出打包成功提示，如图 11-45 所示。

06 查看生成的 APK。生成的 APK 存储在第 4 步指定的位置，默认在 app/build/outputs/apk 目录下，如图 11-46 所示。然后就可以把包发布到应用市场了。

图 11-45

图 11-46

11.8.3 自动打包和签名

前面讲的是通过Android Studio的集成工具来生成签名的apk包，如果每次打一个release包都这样一步一步地操作未免太麻烦，也没有办法实现持续集成。那有没有办法自动打包并使用指定的签名文件进行签名呢？ 当然有了。

第 1 步：配置签名证书。

前提是我们已经创建了自己的签名证书，然后打开Project Structure窗口。

01 选择 app 模块。
02 单击 Signing 标签。
03 新增一个签名证书的配置。
04 选择证书，填入相关信息，如图 11-47 所示。

图 11-47

第 2 步：配置构建类型。

01 单击 Build Types 标签。
02 选择 release。
03 Signing Config 这一项选择刚才我们配置的签名，如图 11-48 所示。

图 11-48

单击【OK】按钮后在build.gradle中就生成了相应的签名配置，如图 11-49 所示。

图 11-49

第 3 步：自动打包和签名。

在终端工具窗口中执行：

```
./gradlew assembleRelease
```

或在Gradle工具窗口单击，如图 11-50 所示。

完成APK打包和签名。

图 11-50

11.8.4 混淆打包

为什么要代码混淆？

出于安全考虑，为了让别人不能反编译我们的应用程序，或者说增加反编译的成本。如果我们的应用程序很容易被反编译，核心代码全部都能被别人看到，那不相当于开源了么？

Android代码混淆

Android使用ProGuard进行代码混淆，通过删除从未用过的代码和使用晦涩名字来重命名类、字段和方法，从而对代码进行压缩、优化和混淆。

Android Studio中代码混淆的操作步骤如下。

01 在混淆配置文件 proguard-rules.pro 中配置混淆规则。混淆规则请参考：http://proguard.sourceforge.net/index.html#manual/usage.html

02 在 build.gradle 中打开混淆开关，minifyEnabled 设置为 true。

```
//    构建类型
    buildTypes {
        // release 包的配置
        release {
            //开启混淆
            minifyEnabled true
            // 指定混淆文件
```

```
        proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
```

11.8.5 多渠道打包

国内Android应用程序有 360、小米、豌豆荚、百度等非常多的下载渠道，如果想统计每个渠道的下载量和活跃度，就需要使用统计平台。

我们以友盟统计为例，介绍如何配置渠道信息并执行自动化打包。

1. 在AndroidMainfest.xml配置可动态替换的渠道参数

友盟集成文档中有说明，使用友盟统计需要在AndroidMainfest.xml配置相应的渠道号：

```
<meta-data
    android: name="UMENG_CHANNEL"
    android: value="xiaomi" /><!--渠道号为：小米-->
```

如果想动态替换渠道号怎么办呢？

```
<meta-data
    android: name="UMENG_CHANNEL"
    android: value="${CHANNEL_ID}" /><!--动态替换渠道号-->
```

2. 在build.gradle中配置渠道信息和自动替换脚本

```
// 多渠道打包
  productFlavors {
      xiaomi {} //渠道名 name 为 xiaomi
      baidu {}
      wandoujia {}

      // 自动替换 AndroidManifest.xml 中的渠道号
      productFlavors.all { flavor ->
          flavor.manifestPlaceholders = [CHANNEL_ID: name]
      }
  }
```

配置好以后在Build Variants窗口中可以选择不同渠道的变种版本，如图 11-51 所示。

Gradle工具栏也会生成相应的任务，如图 11-52 所示。

图 11-51

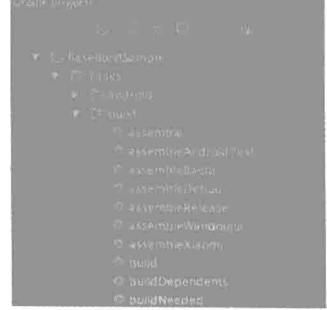


图 11-52

3. 默认配置

```
    // 默认配置
    defaultConfig {
        applicationId "com.wirelessqa.basebuildsample" //apk包名
        minSdkVersion 16
        targetSdkVersion 23
        versionCode 1
        versionName "1.0" //版本号
        //android单元测试test runner
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }
```

所有渠道默认使用这一配置，如果渠道有特殊需求，可以在productFlavors对应的渠道号中单独配置。

4. 打包后自动修改apk的名字

```
    // 打包后自动修改 apk 的名字
    // release 包的命名格式为：产品名_版本号_渠道号.apk
    // debug 包的命名格式为：产品名_版本号_渠道号_Debug_打包时间.apk
    applicationVariants.all { variant ->
        variant.outputs.each { output ->
            def outputFile = output.outputFile
            if (null != outputFile && outputFile.name.endsWith('.apk')) {
                File outputDir = new File(outputFile.parent);
                def baseName = PRODUCT_NAME + "${defaultConfig.versionName}" +
"_" + variant.productFlavors[0].name
                def newApkName

                if (variant.buildType.name.equals('release')) {
                    newApkName = baseName + '.apk'
                } else if (variant.buildType.name.equals('debug')) {
                    newApkName = baseName + "_Debug_${packageTime()}.apk"
                }

                output.outputFile = new File(outputDir, newApkName)
            }
        }

    }
```

5. 自动化打包

如何一次打出所有渠道的Release包呢？

方法一：命令行。

```
$ ./gradlew assembleRelease
```

方法二：Gradle工具窗口，如图 11-53 所示。

方法三：菜单栏→Build→Generate Signed APK→一步步下去→在Flavors中全选→ Finish，如图 11-54 所示。

这样所有渠道的Release包都被打出来了，如图 11-55 所示。

图 11-53

图 11-54

图 11-55

6. 查看渠道号是否被正确替换

单击apk之后，Android Studio会自动解析apk，这样就可以在Android Studio中直接查看apk的信息了，如图 11-56 所示。

图 11-56

❶单击baidu这个渠道→❷单击AndroidManifest.xml→❸查看渠道号为baidu。

由此证明我们的打包脚本是OK的。

提 示

> 如果要打 Debug 包，执行 assembleDebug 任务就可以了。
> 如果只想打某一个渠道的包，执行对应的打包任务就可以了。

本例中全部的build.gradle脚本如下：

```
//插件：
//这个 module 是一个 android 程序，使用 com.android.application
//如果是 android 库，应该使用 com.android.library
apply plugin: 'com.android.application'
```

```
//产品名
def PRODUCT_NAME = "wirelessqa"
//打包时间
def packageTime() {
    return new Date().format("MMddhhmmss", TimeZone.getTimeZone("GMT+8"))
}

android {

    // 签名配置
    signingConfigs {
        MySigning {
            keyAlias 'myandroid'
            keyPassword '123456'
            storeFile file('/Users/bixiaopeng/myandroid.jks')
            storePassword '123456'
        }
    }
    // 编译 sdk 版本
    compileSdkVersion 23
    // 构建工具版本
    buildToolsVersion "23.0.2"

    // 默认配置
    defaultConfig {
        applicationId "com.wirelessqa.basebuildsample" //apk 包名
        minSdkVersion 16
        targetSdkVersion 23
        versionCode 1
        versionName "1.0" //版本号
        //android 单元测试 test runner
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
        versionNameSuffix 'test'
    }

    // 产品特性
    productFlavors {
        xiaomi {} //渠道名 name 为 xiaomi
        baidu {
        }
        wandoujia {}
        // 自动替换 AndroidManifest.xml 中的渠道号
        productFlavors.all { flavor ->
            flavor.manifestPlaceholders = [CHANNEL_ID: name]
        }
    }

    // 构建类型，此处配置 debug 和 release 版本的一些参数，像混淆、签名配置。
    buildTypes {
```

```
        // release 包的配置
        release {
            //开启混淆
            minifyEnabled true
            // 指定混淆文件
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
            // 指定签名配置
            signingConfig signingConfigs.MySigning
            zipAlignEnabled true
            //移除无用的资源文件
            shrinkResources true
        }
    }

    // 打包后自动修改 apk 的名字
    // release 包的命名格式为：产品名_版本号_渠道号.apk
    // debug 包的命名格式为：产品名_版本号_渠道号_Debug_打包时间.apk
    applicationVariants.all { variant ->
        variant.outputs.each { output ->
            def outputFile = output.outputFile
            if (null != outputFile && outputFile.name.endsWith('.apk')) {
                File outputDir = new File(outputFile.parent);
                def baseName = PRODUCT_NAME + "${defaultConfig.versionName}" +
"_" + variant.productFlavors[0].name
                def newApkName

                if (variant.buildType.name.equals('release')) {
                    newApkName = baseName + '.apk'
                } else if (variant.buildType.name.equals('debug')) {
                    newApkName = baseName + "_Debug_${packageTime()}.apk"
                }

                output.outputFile = new File(outputDir, newApkName)
            }
        }

    }
}

// 依赖的第三方库
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'com.android.support: appcompat-v7: 23.4.0'
    compile 'com.android.support.constraint: constraint-layout: 1.0.0-alpha1'
    compile 'com.android.support: design: 23.4.0'
    testCompile 'junit: junit: 4.12'
    androidTestCompile 'com.android.support.test.espresso: espresso-core:
2.2.2'
    androidTestCompile 'com.android.support.test: runner: 0.5'
```

```
    androidTestCompile 'com.android.support: support-annotations: 23.4.0'
}
```

11.9 配置开发者服务

开发者服务中都是Google提供的一些非常实用的服务，包括广告（见图 11-57）、身份验证（见图 11-58）和通知（见图 11-59）。

图 11-57

图 11-58

图 11-59

第 12 章　运行和调试

调试应该是每个程序员必备的技能，代码中总会出现问题，为了解决问题并验证程序的正确性，我们总会用到调试功能。Android Studio中强大的调试功能可以帮助我们快速定位问题。

本章将向大家介绍在Android Studio中如何配置、运行和调试应用程序。

本章重要知识点 >>>>>>>>>>

- 如何配置和运行应用程序；
- 如何使用调试工具；
- 如何配置和运行单元测试。

12.1　运行和调试配置

12.1.1　运行和调试配置

打开运行和调试配置的方法如下。

菜单栏：Run→Edit Configurations...

快捷键：control + alt + R（macOS）或者Alt + Shift + F9→0（Windows/Linux）

工具栏：单击Edit Configurations...，如图 12-1 所示。

图 12-1

利用上述快捷方法后弹出配置界面，如图 12-2 所示。

图 12-2

Defaults列出了所有默认的配置，单击 + 按钮可以新建一个新的Android启动/调试配置。

12.1.2 Android应用程序配置

假设我们已经新建了一个名为app的Android应用配置，界面如图 12-3 所示。

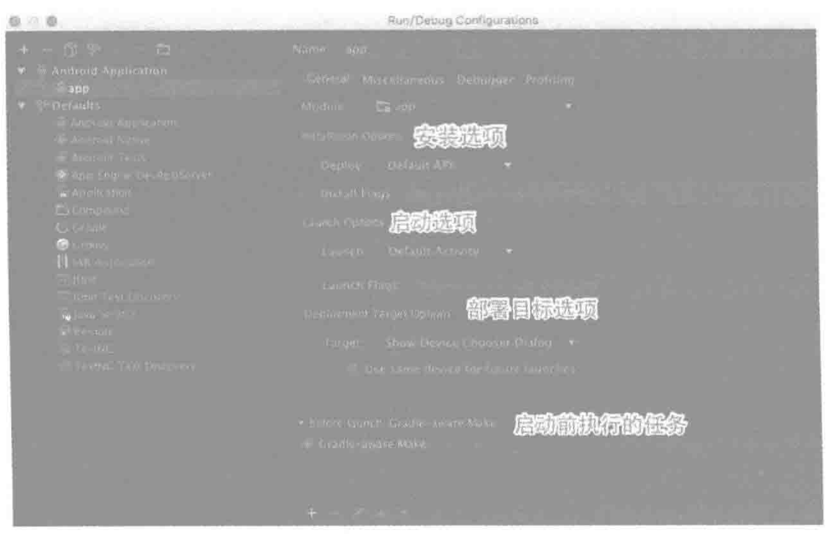


图 12-3

1. Name

在这里输入配置的名字，这个名字会在工具栏运行应用程序配置的下拉列表中看到，如图 12-4 所示。

2. General

在这里配置安装、启动、部署应用程序选项。

（1）Module：列表中列出了当前项目中的所有模块，可以指定相应的模块来运行。

（2）Installation Options：安装选项。

① Deploy：下拉列表中列出了应用程序运行时的部署模式，如图 12-5 所示。

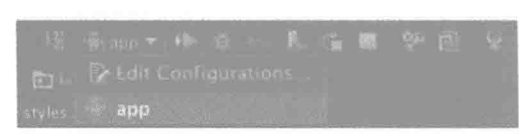


图 12-4

图 12-5

这里有以下 3 个选项。

- Default APK：部署默认的 APK，运行时会先打包安装，再启动 APK。
- Custom Artifact：部署自定义的 APK，会根据所选择的模块来选择对应的配置。
- Nothing：不做任何部署，运行时会直接启动应用，如果应用已经安装了会直接启动，没有安装就会报错。

② Install Flags：给adb shell pm install 添加运行参数，参数参加在pm install后面。Install Flags为空时，运行应用程序时执行的命令是这样的：

```
# 1.把打好的包放到手机中的/data/local/tmp/目录下
$ adb push /Volumes/MyApplication/app/build/outputs/apk/app-debug.apk
/data/local/tmp/com.wirelessqa.myapplication
# 2.重新安装应用程序
$ adb shell pm install -r "/data/local/tmp/com.wirelessqa.myapplication"
    pkg: /data/local/tmp/com.wirelessqa.myapplication
Success
```

添加一个参数-f，界面设置如图 12-6 所示。

```
# 2.重新安装应用程序，在 install 后就多了个-f 参数
$ adb shell pm install -f -r "/data/local/tmp/com.wirelessqa.myapplication"
pkg: /data/local/tmp/com.wirelessqa.myapplication
Success
```

（3）Launch Options：启动选项

① Launch提供了 4 个选项，如图 12-7 所示。

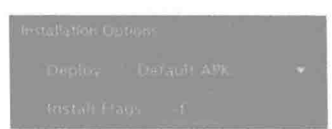


图 12-6

图 12-7

- Default Activity：启动默认 Activity，运行时会启动默认的 MainActivity，如果没有就会报错。
- Specified Activity：指定启动的 Activity。

在输入框中输入Activity的名字，输入时会有智能联想，如图 12-8 所示。

如果记不住名字，还可以搜索（见图 12-9）。或在项目结构中查找（见图 12-10）。

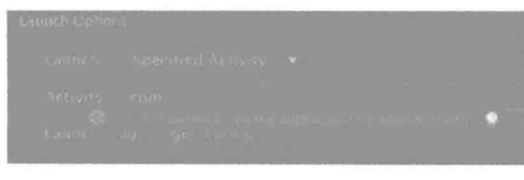


图 12-8

图 12-9

图 12-10

定义好启动Activity后，运行应用时这个Activity就会被启动。

- Nothing：运行时不会启动任何 Activity。
- URL：在这里可以指定启动的 scheme。

② Launch Flags：给adb shell am 添加运行参数，参数添加在命令的最后面。

（4）Deployment Target Options：部署目标选项，如图 12-11 所示。

① Target

- Show Device Chooser Dialog：选择此选项，每次运行时都会弹出选择设备对话框。
- USB Device：使用 USB 连接的设备。
- Emulator：使用模拟器。

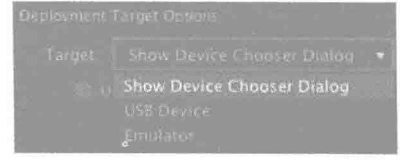


图 12-11

图 12-12　选择设备对话框

② Use same selection for future launches：如果勾选此项，以后运行时都使用同样的选择，不需要再次选择了。

3. Miscellaneous

在这里配置日志和安装选项，如图 12-13 所示。

（1）Logcat。

- Show logcat automatically：运行时自动显示 logcat 日志。
- Clear log before launch：启动前清空日志。

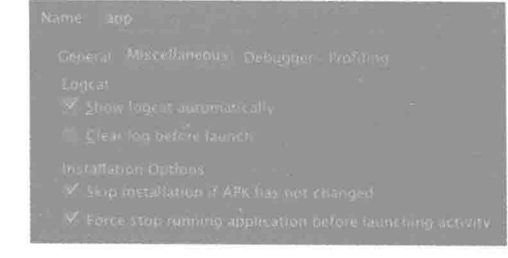


图 12-13

（2）Installation Options。

- Skip installation if APK has not changed：如果代码没有变更，运行时跳过安装。
- Force stop running application before launching activity：启动 Activity 前强制关闭运行的应用程序。

4. Debugger

在这里配置调试类型，如图 12-14 所示。Debug类型包括Java、Native、Hybrid。

图 12-14

5. Profiling

在这里配置图形跟踪选项，如图 12-15 所示。

Disable precompiled shaders and programs：禁用预编译着色器和程序。

6. Before launch

在这里可以配置运行之前需要执行的任务，默认会执行Make，如图 12-16 所示。

添加任务

单击+添加一个新的任务，如图 12-17 所示。（各选项说明如表 12-1 所示。）

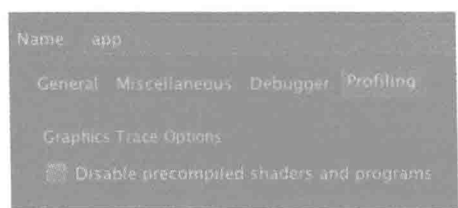


图 12-15

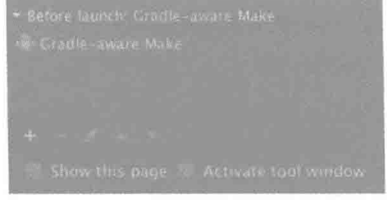


图 12-16

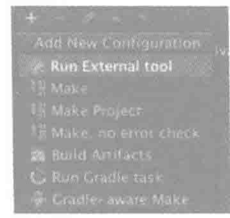


图 12-17

表 12-1

任务	说明
Run External tool	运行外部工具，如果有的话可以直接选择，如果没有可以去新建
Make	编译选择的模块
Make Project	编译项目
Make，no error check	编译选择的模块，但不进行错误校验
Build Artifacts	构建构件，如果有的话可以选择对应的构件
Run Gradle task	运行Gradle任务
Gradle-aware Make	执行Gradlew任务

【实例演示】在运行应用之前先执行gradlew的clean任务。

01 单击+，选择【Gradle-aware Make】。

02 在弹出的 Gradle 任务选择框中输入 clean，然后会弹出智能联想，选择 app：clean，如图 12-18 所示。

03 单击【OK】按钮确定，将 gradle：app：clean 移到最上面，如图 12-19 所示。其中，app 是模块名，clean 是 gradle 任务，因此确定后在运行时会先对 app 这个模块执行 clean 任务。

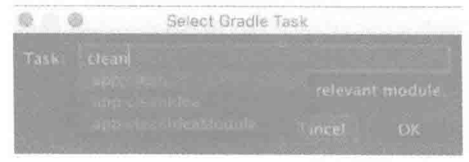


图 12-18

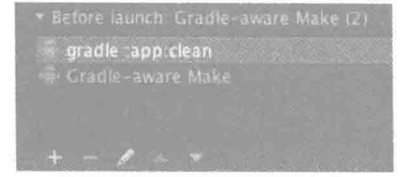


图 12-19

如果勾选Show this page选项，那么在每次运行时都会显示运行和调试配置界面，如图 12-20 所示。

图 12-20

如果我们需要每次运行前都更改配置，就勾选此项。

- Activate tool window：激活工具窗口。
- Before launch 执行顺序：Before launch 任务的执行顺序显示配置界面→执行 Gradle 任务（按顺序）→激活工具窗口，如图 12-21 所示。

图 12-21

12.2 运行应用程序

1. 运行指定应用程序

菜单栏： Run→Run '工具栏选中的配置名'

快捷键： control + R（macOS）或者Shift + F10（Windows/Linux）工具栏如图 12-22 所示。

图 12-22

2. 运行应用程序或编辑配置

菜单栏： Run→Run...

快捷键： control + alt + R（macOS）或者Alt + Shift + F10（Windows/Linux）

利用菜单栏或快捷键操作后弹出选择界面，如图 12-23 所示。如果项目中有多个配置，默认光标定位在工具栏选中的配置上面，如图 12-24 所示。

图 12-23

图 12-24

我们可以通过选项前面的数字快速运行应用程序或打开编辑配置对话框。如果按住Shift键，Run就变成了Debug。

3. 运行应用程序时的执行顺序

假设我们的配置是这样的，如图 12-25 所示。Run 'app'后的执行顺序如下：

（1）显示配置界面。
（2）弹出设备选择对话框。
（3）执行Gradle任务。
（4）安装应用。
（5）启动Activity。

图 12-25

4. 清理后重新运行应用程序

这是 2.1 版本新增的功能，应用程序运行后此功能被激活，单击后会执行彻底清理→构建→安装→运行一系列操作。

菜单栏：Run→Clean and Rerun'配置名'（见图 12-26）
快捷键：alt + command+ R（macOS）或者Ctrl + F5（Windows/Linux）

5. 停止应用程序运行

菜单栏：Run→Stop '配置名'
快捷键：fn + command+ F2（macOS）或者Ctrl + F2（Windows/Linux）
工具栏如图 12-27 所示。

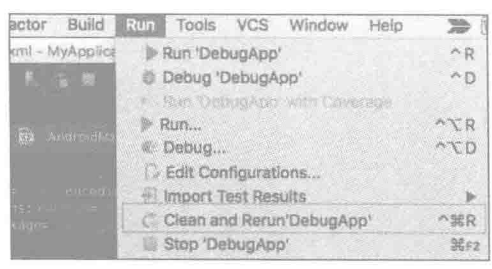


图 12-26

图 12-27

12.3 调试应用程序

在项目开发过程中，我们的大部分时间可能是用来解决问题的，不管是测试报过来的问题还是我们在自测的时候发现的问题，都需要通过调试来排查和定位问题的原因，然后解决问题。

Android Studio允许我们在模拟器和真机上调试应用程序， 我们可以根据需求设置不同类型的断点，可以查看内存中数据的变化，在运行时可以添加日志或计算表达式。

调试应用程序的一般步骤如下：

01 添加断点。
02 运行调试。
03 执行到断点。
04 显示调试器窗口。
05 查看调试信息。
06 使用步进调试工具分析代码。
07 使用控制调试工具管理断点和程序运行。

下面通过一个例子来完整演示调试应用程序的步骤。

第 1 步：添加断点。
添加断点很简单，只需要在左边栏单击一下即可，如图 12-28 所示。

第 2 步：运行调试。
设置好断点后，我们就可以运行调试了。
菜单栏：Run→Debug
快捷键：control + alt + D（macOS）或者Alt + Shift + F9（Windows/Linux）

工具栏如图 12-29 所示。

图 12-28

图 12-29

第 3 步：执行到断点。
运行调试后，线程执行到该断点时会暂时挂起，同时激活调试器窗口，如图 12-30 所示。

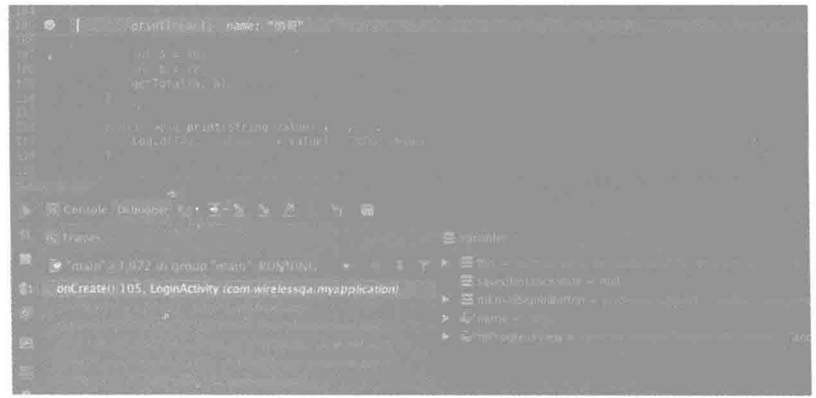


图 12-30

调试器会自动选择当前断点上的线程，并显示该线程的堆栈帧，如图 12-31 所示。

图 12-31 的线程图标表示当前断点上的线程。

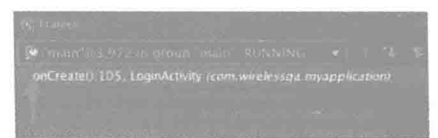


图 12-31

第 4 步：显示调试器窗口。

现在整体认识一下调试器窗口的布局，如图 12-32 所示。

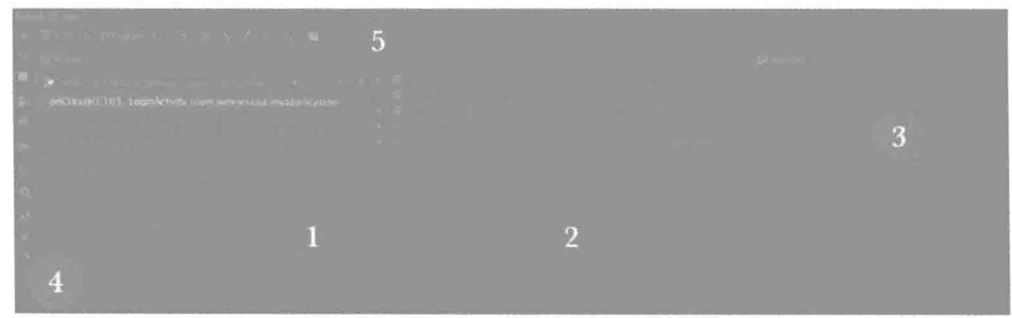


图 12-32

❶ 帧调试窗口。

❷ 变量调试窗口。

❸ 监视窗口。

❹ 调试控制工具栏。

❺ 步进调试工具栏。

关于窗口的作用和工具栏的使用，下面我们会详细介绍。

第 5 步：查看调试信息。

如图 12-33 所示，在调试器窗口中可以看出：

❶ 当前挂起的线程为主线程。

❷ 当前触发的断点在com.wirelessqa.myapplication包下面的LoginActivity类中，具体位置在该类的onCreate方法中，行号为第 105 行。

❸ 变量name和mProgressView的值。

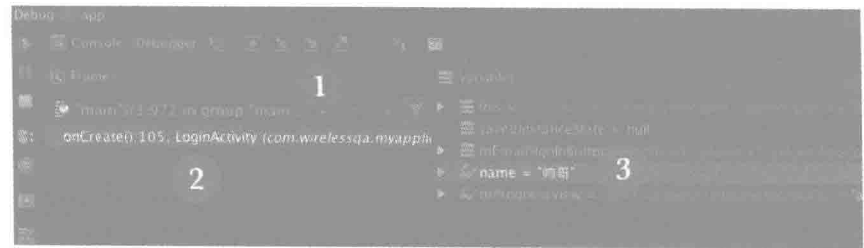


图 12-33

第 6 步：使用步进调试工具。

01 执行单步进入，如图 12-34 所示。

图 12-34

会进入方法体内，如图 12-35 所示。

图 12-35

02 执行单步跳过，会执行下一行，如图 12-36 所示。

图 12-36

第 7 步：使用调试控制工具。

01 执行到下一个断点，如图 12-37 所示。

02 结束程序运行，如图 12-38 所示。

图 12-37

图 12-38

调试应用程序的一般步骤就介绍到这里，关于断点和调试工具的详细用法，接下来会详细介绍。

12.4 断　点

断点会暂停应用程序的运行，线程被挂起，然后可以通过调试器有逐行查看代码。

断点的状态分为已启用断点和已禁用断点，如图 12-39 所示。

- 已启用的断点在调试状态下，应用程序每次执行到该断点时都会暂停。
- 已禁用的断点在调试状态下，不会对应用程序的执行造成任何影响。

图 12-39　启动和禁用断点的状态

Android Studio中的断点分为很多类型， 每一种断点都有它适用的场合和特殊的作用。了解这些断点，才有可能更好地使用断点。

12.4.1　行断点

行断点是我们最常用到的断点，被用于对代码中特定的行进行调试。

菜单栏： Run→Toggle Line Breakpoint

快捷键： command + F8（macOS）或者Ctrl + F8（Windows/Linux）

单击某行（非方法名所在行）左边栏即可设置断点，如图 12-40 所示。

取消行断点跟添加行断点的方法相同。

属性设置

右击行断点，弹出属性设置对话框，如图 12-41 所示。

图 12-40

图 12-41

- Enabled：断点启用和禁用。
- Suspend：勾选 All，执行到断点时所有线程都会被挂起。勾选 Thread，执行到断点时只有当前断点所在的线程会被挂起。切换线程挂起策略的时候，会显示 Make Default，单击后就会把当前选中的策略变为默认的。
- Condition：设置断点暂停条件。

12.4.2　方法断点

方法断点主要用来检查方法的输入和输出（参数和返回值）。

操作步骤： 菜单栏→Run→Toggle Method Breakpoint→单击方法名所在行的左边栏，如图 12-42 所示。

属性设置： 右击方法断点的图标，弹出属性设置对话框，如图 12-43 所示。

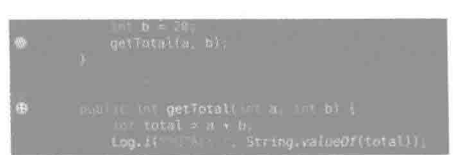


图 12-42

图 12-43

跟行断点相比，方法断点的属性中多了一个Watch，用来设置是否监视方法的进入和退出。Method entry 和 Method exit 默认都被选中，也就是说调用此方法开始的时候和结束的时候断点都会被触发，如果不选中就不会触发。调试效果如图 12-44 所示。

图 12-44

12.4.3 字段观察点

当我们对程序运行的过程不太关心，只关心某个变量的变化时可以使用字段观察断点。

操作步骤：菜单栏→Run→Toggle Field Watchpoint→单击字段所在行的左边栏，如图 12-45 所示。

属性设置：右击变量的断点图标，显示属性设置，如图 12-46 所示。

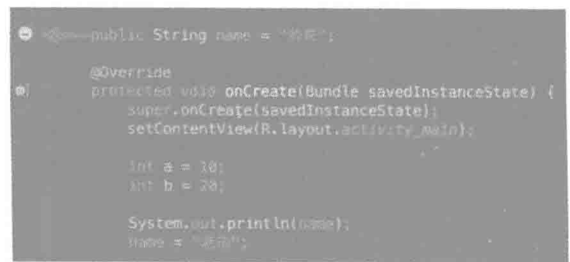


图 12-45

图 12-46

这里有两个选择是Java Field Watchpoints特有的。

- Field access：当字段被访问的时候触发断点。
- Field modification：当字段被修改的时候触发断点。

在执行界面中的效果如图 12-47 所示。

图 12-47

12.4.4 条件断点

条件断点用来设置断点被触发的条件，如果条件不满足断点是不会被触发的。

操作步骤：右击左边栏的断点图标→勾选Condition→设置暂停条件，如图 12-48 所示。

图 12-48 中设置的是当变量a不等于 10 的时候触发断点，线程挂起。

图 12-48

12.4.5 临时断点

当我们想某个断点只被触发一次后就自动删除的时候，可以使用临时断点。

菜单栏：Run→Toggle Temporary Line Breakpoint

快捷键：fn + command + option + shift + F8（macOS）或者Ctrl + Alt + Shift + F8（Windows/Linux）

按住Alt键，单击左边栏，如图 12-49 所示。

如果想把临时断点变为普通断点，可以在属性中取消勾选【Remove once hit】(见图 12-50)。

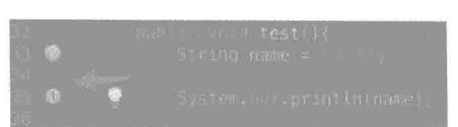


图 12-49

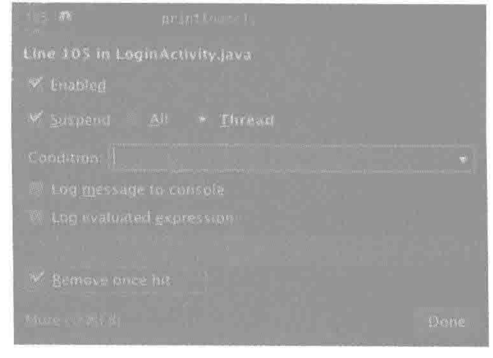


图 12-50

12.4.6 异常断点

异常断点会在某个异常发生时触发断点，这样我们就可以第一时间得到异常信息，方便排查问题。

菜单栏： Run→View Breakpoints

快捷键： fn + shift + command + F8（macOS）或者Ctrl + Alt + F8（Windows/Linux）

利用菜单栏或快捷键操作后，在打开的断点窗口单击左上角的加号（+）按钮，选择Java Exception Breakpoints。然后在弹出的【Enter Exception Class】窗口中输入要调试的异常，例如NullPointerException，如图 12-51 所示。

图 12-51

确定后当应用程序抛出空指针异常时，就会在异常处触发断点。

12.4.7 日志断点

在调试的时候，我们想临时多加一些日志但又不想重新构建应用程序的时候，就可以使用日志断点。

操作步骤： 右击断点→取消勾选【Suspend】→在展开的选项中勾选【Log evaluated expression】→输入日志信息表达式，如图 12-52 所示。

图 12-52

因为取消了Suspend，所以执行到断点处不会暂停，而是会打印日志。日志会打印在Console窗口中，如图 12-53 所示。

图 12-53

12.4.8 禁用断点

当某个断点暂时不需要但又不想删除时可以先禁用，如图 12-54 所示。

菜单栏：Run→Toggle Breakpoint Enable

快捷键：option +单击断点（macOS）或者Alt +单击断点（Windows/Linux）

禁用断点之后，执行到该断点时就不会暂停了。如果想恢复断点，使用禁用相同的操作即可。

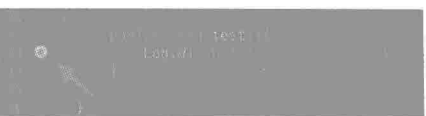


图 12-54 禁用断点状态

12.4.9 断点设置

在断点的设置对话框中既可以查看和管理项目中的断点，也可以设置断点的属性。

打开属性配置窗口

菜单栏：Run→View Breakpoints

快捷键：fn + shift + command + F8（macOS）或者Ctrl + Alt + F8（Windows/Linux）

调试工具窗口：单击左边工具栏的View Breakpoints

右击断点的图标→单击More（见图 12-55），然后会打开断点设置对话框（见图 12-56）。

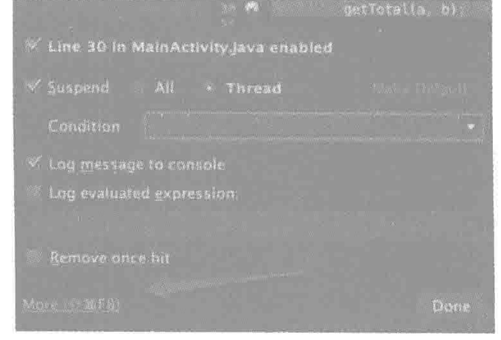


图 12-55

❶ 断点管理：断点按类型进行分组，在这里可以新建和删除断点。

❷ 常用设置：在编辑器窗口中右击断点图标就能够设置这些属性。

❸ 日志断点设置：在这里可以设置日志断点的属性。

❹ 其他设置：设置临时断点，以及选中的断点被执行后是禁用还是丢弃当前断点。

❺ 过滤：设置断点的一些限制，如作用于实例的ID、指定类以及有效次数。

❻ 预览窗口：在这里查看断点所在的代码位置。

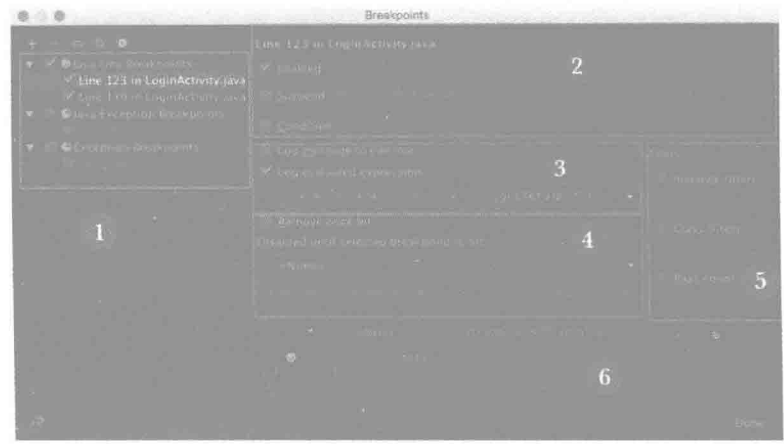


图 12-56

12.5 帧调试窗口

帧调试窗口显示了当前断点所在的线程以及执行到该断点所调用过的方法。在介绍帧调试窗口前我们先要了解一下什么叫堆栈帧。

1. 堆栈帧

堆栈帧是用来存储数据和部分过程结果的数据结构，同时也用来处理动态链接、方法返回值和异常分派。每一次调用方法在堆栈中都会占用一部分内存，单位是帧。堆栈帧随着方法调用而创建，随着方法结束而销毁。每个堆栈帧中都包括传入的参数、返回地址、方法存储在堆栈上的局部变量以及对程序调试提供支持的信息。一个线程包括多个堆栈帧。

2. 当前堆栈帧

一个线程在执行过程中，执行到断点处暂停时，如果只有当前正在执行的那个方法的堆栈帧是活动的，这个堆栈帧就叫当前堆栈帧。

帧调试窗口如图 12-57 所示。

图 12-57

❶ 显示当前断点所在的线程。

❷ 显示执行过的方法（自下而上），也可以说是执行到当前堆栈帧这个过程中所创建过的堆栈帧。从当前堆栈帧可以看出，当前触发的断点在 com.wirelessqa.myapplication 包下面的 LoginActivity类中，具体位置在该类的getTotal方法中，行号为第 119 行。

❸ 查看工具：查看执行过的帧和过滤当前应用的帧，如图 12-58 所示。

内容菜单：右击堆栈帧，显示可操作菜单，如图 12-59 所示。

图 12-58

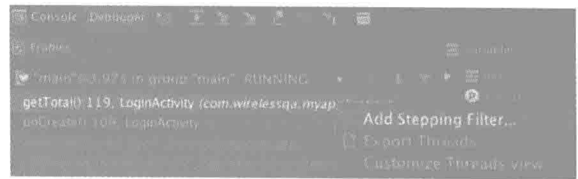


图 12-59

❶ 添加步进过滤器（Add Stepping Filter），使用此命令在打开的对话框中添加一个步进过滤器。

❷ 导出线程（Export Threads），如图 12-60 所示。

❸ 自定义线程显示（Customize Threads View），如图 12-61 所示。

图 12-60

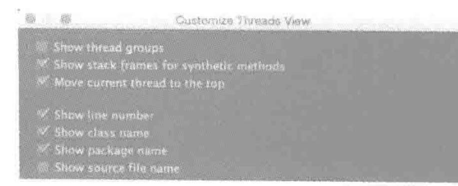


图 12-61

12.6 变量调试窗口

我们可以在变量调试窗口中检查应用程序中对象值的存储。当选择堆栈帧的时候，变量调试窗口就会显示范围内的所有数据（方法的参数、本地变量和实例变量），如图 12-62 所示。

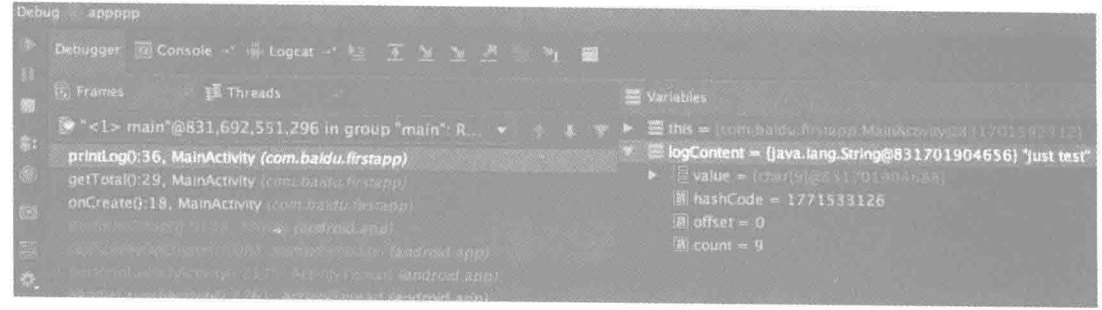


图 12-62

在这里我们可以设置对象的标签、检查对象、计算表达式、添加变量到监视窗口等，如图 12-63 所示。

接下来我们一一介绍变量窗口中提供的这些功能。

- Inspect：检查功能用来检查字段、局部变量和表达式的引用，既可以打开一个非模式检查窗口（可以托离主窗口，所以称为非模式），也可以按照需求打开多个检查窗口。

操作步骤：右击要检查的字段、局部变量或表达式→选择Inspect→弹出一个非模式检查窗口（见图 12-64）。

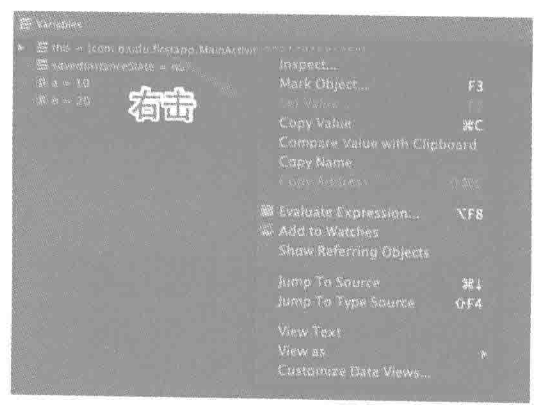


图 12-63

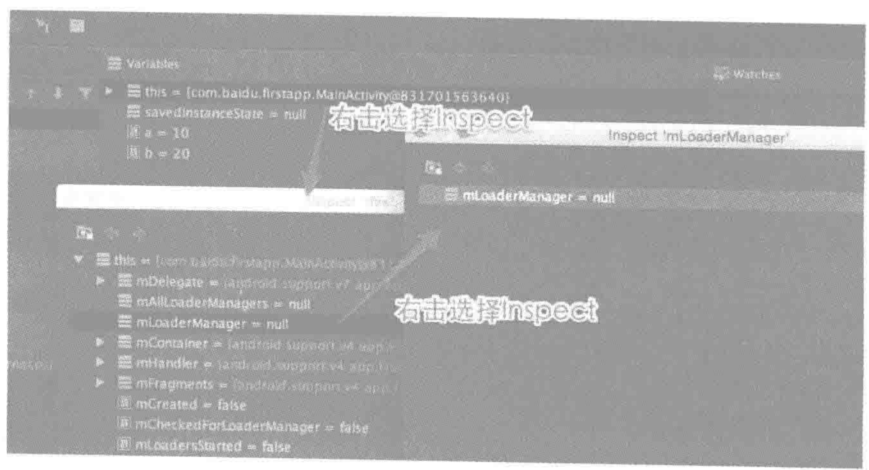


图 12-64

- Mark Object：标记对象功能可以给对象添加标签，看起来更加直观。

操作步骤：右击要标记的对象→选择Mark Object或者按快捷键fn+F3（macOS）或者F11（Windows/Linux），然后在弹出的窗口中输入对象的标签，如图 12-65 所示。

单击Preview右边的■按钮可以选择标签的颜色，如图 12-66 所示。

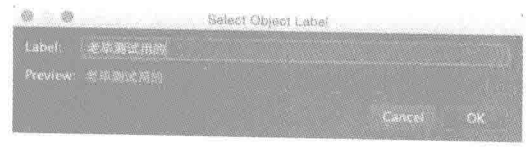


图 12-65

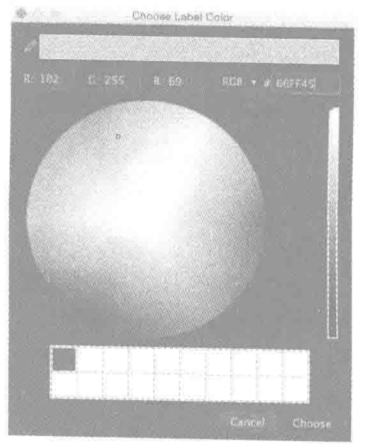


图 12-66

例如，选择绿色，如图 12-67 所示。

确定后的显示情况如图 12-68 所示。

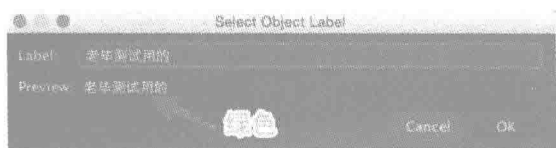


图 12-67

图 12-68

- Set Value：用来更改运行时字段或变量的值。

我们来看一个例子，变量a的值是 10，b的值是 20，a和b的和total是 30，如图 12-69 所示。

调试开始后改变变量a的值。右击变量a，选择Set Value或者使用快捷键fn + f2，此时a的值可以被修改，如图 12-70 所示。在下拉列表中有操作过的历史记录可供选择。

图 12-69

图 12-70

修改后按回车，可以看到变量调度窗口和编辑器中a的值随即变为我们设定的值，如图 12-71 所示。接着再运行，total的值也发生了变化，如图 12-72 所示。这样调试是不是非常方便？

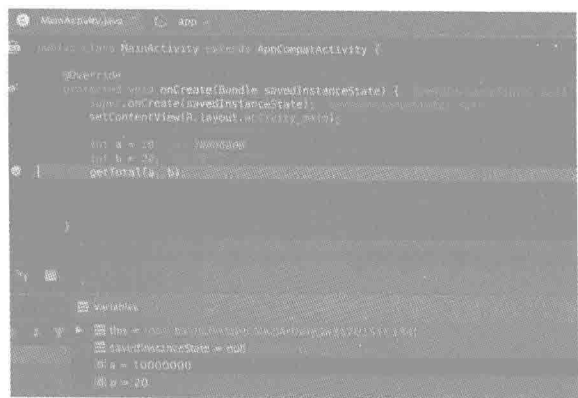


图 12-71

图 12-72

- Copy Value：用来复制变量或结构。当选中一个变量来复制时可以复制变量的值，当选择多项时会复制整个结构，如图 12-73 所示。
- Compare Value with Clipboard：跟剪切板中的值比较，这个功能会把当前变量或结构的值跟剪切板中的进行比较，如图 12-74 所示。

图 12-73

图 12-74

- Copy Name：复制变量的名字。
- Evaluate Expression：计算表达式。
- Add to Watches：添加到监视窗口会把选择的变量或表达式添加到监视窗口，在我们调试应用程序的时候监视窗口会显示对应的表达式或变量的值。
- Show Referring Objects：显示选中变量引用的对象列表。
- Jump to Source：跳转到所选变量或字段的源码。
- Jump to Type Source：打开源代码中所选变量或字段的类型。
- View Text：用来显示所选变量的文本，如图 12-75 所示。

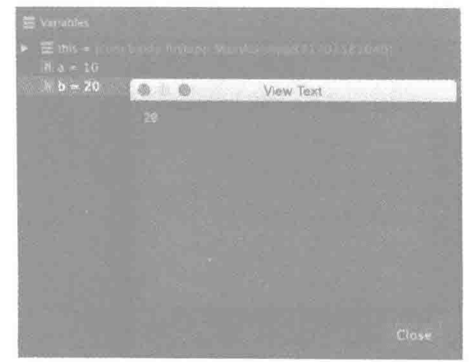


图 12-75

- View as：选择查看类型，如图 12-76 所示。
- Customize Data Views：自定义数据显示，如图 12-77 所示。

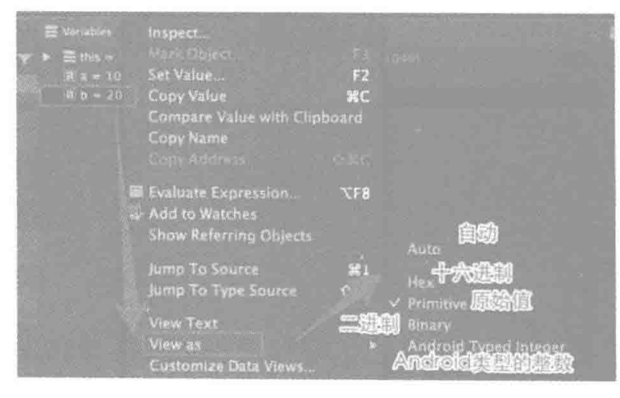


图 12-76

图 12-77

12.7　监视窗口

监视窗口用来计算当前堆栈帧范围内的变量或表达式，在调试的过程中通过监视窗口来监视变量或表达式的值。

12.7.1 添加变量或表达式到监视窗口

1. 监视变量

方法一：在Variables窗口中右击变量→Add to Watches（见图 12-78），然后在右边的监视窗口将会出现变量a（见图 12-79）。

方法二：在Watches窗口中单击左下角的+按钮→输入变量名，会弹出智能联想，选中后添加成功，如图 12-80 所示。每当程序执行到包含a变量的时候，监视窗口的值会随着a变量值的变化而变化，达到监视的目的。

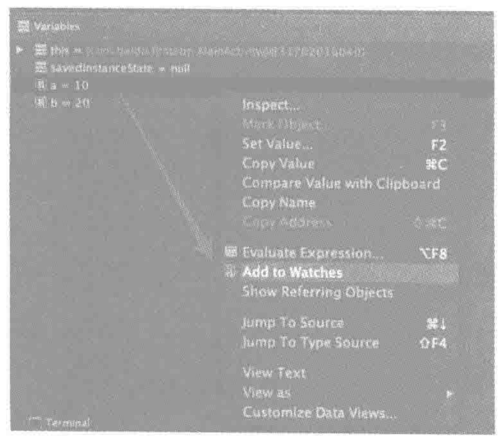


图 12-78

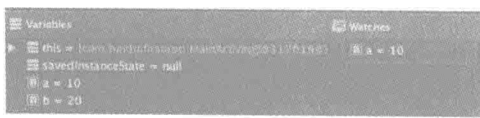


图 12-79

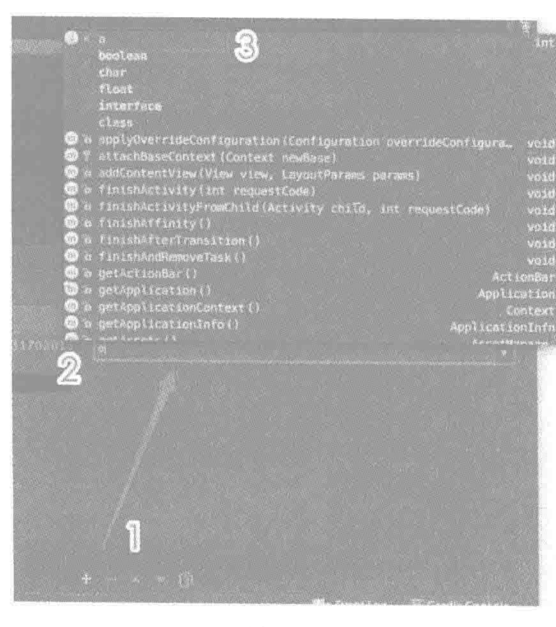


图 12-80

2. 监视表达式

方法一：按快捷键fn + option + F8（macOS）或者Alt + F8（Windows/Linux），进入计算表达式窗口，输入所要监视的表达式，如图 12-81 所示。

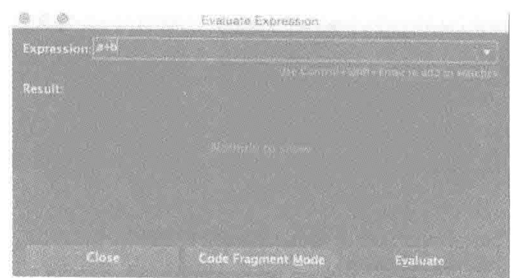


图 12-81

再使用Expression下面提示的快捷键，Mac上的快捷键是control + shift + enter添加到监视窗口（见图 12-82）。

方法二：在Watches窗口中单击左下角的+按钮→输入表达式，如图 12-83 所示。

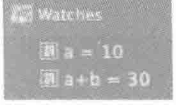


图 12-82 监视窗口

图 12-83

12.7.2 快捷工具

利用监视窗口底部的工具栏可以快速方便地进行新建、删除、移动、复制操作，如图 12-84 所示。单击鼠标右键会弹出更多监视窗口的操作功能，如图 12-85 所示。大部分功能跟变量窗口提供的差不多，前 5 个命令是监视窗口独有的，主要就是增、删、改、查。

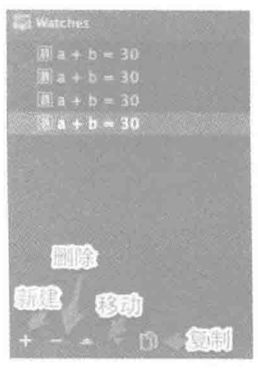


图 12-84

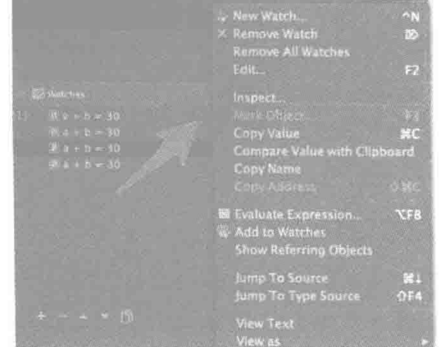


图 12-85

12.8 调试控制工具

调试控制工具用来管理调试当中的程序运行，提供了如下常用功能。

- 暂停、恢复程序运行。
- 终止进程。
- 查看、禁用断点。
- 获取线程堆栈。

1. 恢复程序运行

当程序在断点处暂停的时候，可以使用此功能来恢复程序运行。如果有下一个断点， 就会跳转到下一个断点处。如果没有断点，程序就继续运行。

快捷键：option + command + R（macOS）或者F9（Windows/Linux）

调试工具栏：Resume Program，如图 12-86 所示。

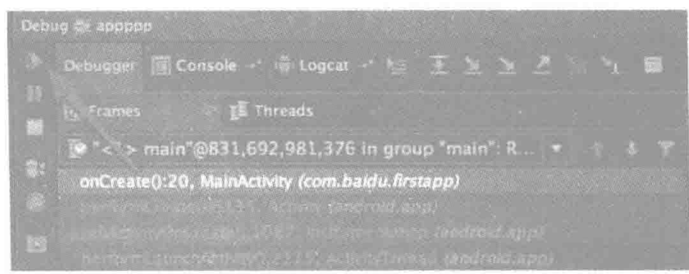


图 12-86

2. 暂停程序运行

程序运行时单击按钮来暂停程序运行。

操作步骤：调试工具栏→Pause Program，如图 12-87 所示。

图 12-87

3. 终止进程

当我们想终止调试的时候可以直接终止进程。

快捷键：fn + command + F2（macOS）或者Ctrl + F2（Windows/Linux）

调试工具栏：Stop '配置名'，如图 12-88 所示。

图 12-88

4. 查看断点

快捷键：fn + shift + command + F8（macOS）或者Ctrl + Shift + F8（Windows/Linux）

调试工具栏：View Breakpoints，如图 12-89 所示。

单击按钮可打开断点对话框，配置断点属性，如图 12-90 所示。

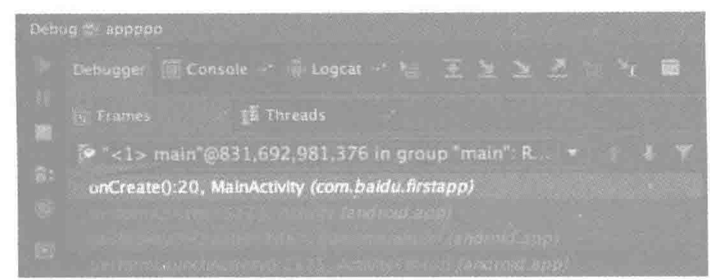


图 12-89

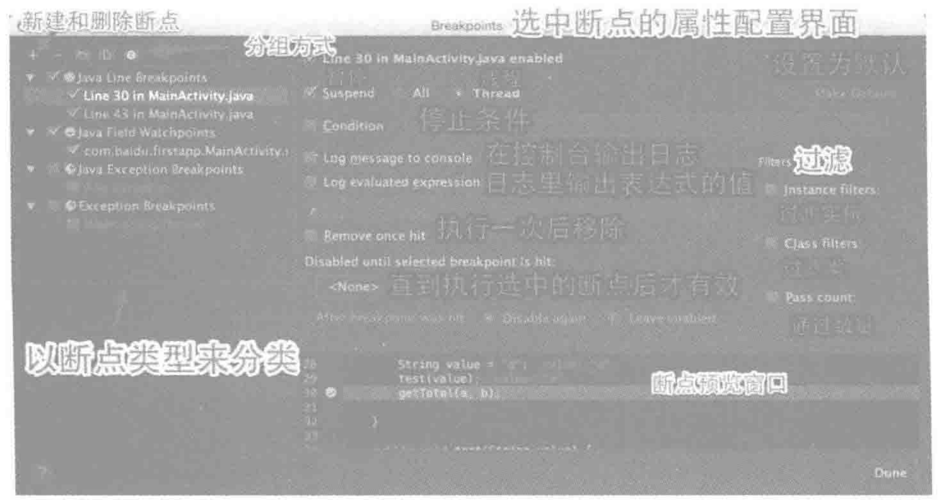


图 12-90

我们可以在断点的属性配置里对断点的属性进行增、删、改，非常方便。

5. 禁用断点

禁用断点功能可以切换断点状态（启用/禁用）。我们可以暂时禁用项目中的断点去执行程序，这样就不会在断点处停止了。

操作步骤： 调试工具栏→Mute Breakpoints，如图 12-91 所示。

使用方法： 将光标放到断点所在行，单击按钮。被禁用的断点如图 12-92 所示。

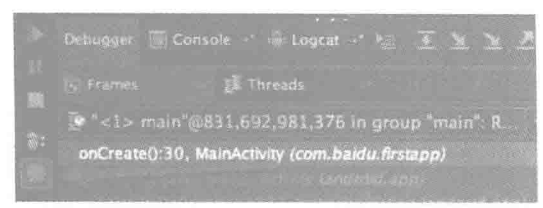


图 12-91

图 12-92

6. 获取线程堆栈

操作步骤： 调试工具栏→Get thread dump，如图 12-93 所示。

单击【Get thread dump】后显示线程堆栈面板，如图 12-94 所示。

图 12-93

图 12-94

7. 恢复布局

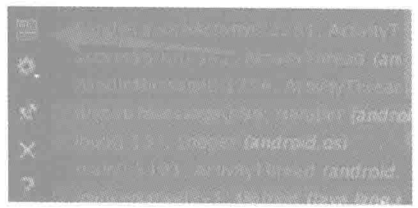


图 12-95

恢复布局功能可以恢复到原始的布局，当前所有的布局变更都会被放弃。

操作步骤：调试工具栏→Restore Layout，如图 12-95 所示。

8. 设置

操作步骤：调试工具栏→Settings，如图 12-96 所示。

- Show Values Inline：选中后启动内联调试功能，允许在编辑器中观察执行过的变量的值，如图 12-97 所示。

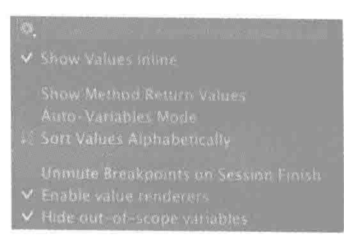


图 12-96

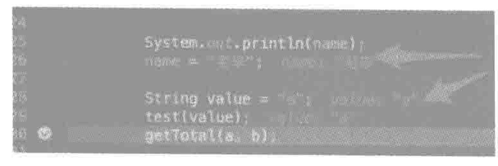


图 12-97

- Show Method Return Values：选中后会显示上次执行方法的返回值。
- Auto-Variables Mode：选中后调试器可以自动评估某些变量（在断点前几行和后几行的变量）。
- Sort Values Alphabetically：选中后窗口中的变量值按字母的顺序排列。

- Unmute Breakpoints on Session Finish：选中后当一个调试会话完成就会重新启用所有禁用的断点。

12.9 步进调试工具

使用步进调试工具可以一行一行地查看代码的执行，就像程序执行的慢镜头回放一样，我们可以使用步进调试工具查看代码执行的过程。

1. 显示执行点

当我们需要查看当前的执行点时，光标会立刻定位到当前执行到的断点。

快捷键： fn + option + F10（macOS）或者Alt + F10（Windows/Linux）

调试工具栏： Show Execution Point（见图 12-98）。

然后编辑器中会高亮显示当前的执行点，并在帧面板中显示对应的堆栈帧。

图 12-98

【实例演示】

光标定位和选中的堆栈帧如图 12-99 所示。单击显示执行点（Show Execution Point）按钮，效果如图 12-100 所示。

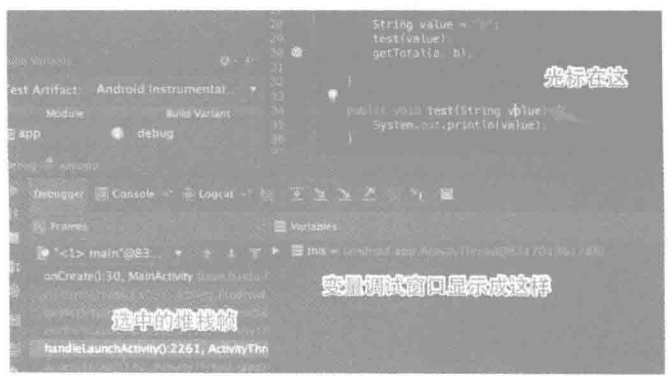


图 12-99

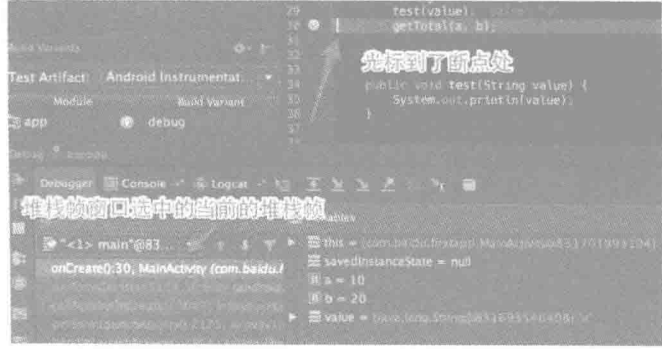


图 12-100

2. 单步跳过

执行单步跳过，就会执行下一行。如果下一行是一个方法，就不会进入方法体，而是会执行完此方法，然后跳到下一行。

快捷键： fn + F8（macOS）或者F8（Windows/Linux）

调试工具栏： Step Over（见图 12-101）。

3. 单步进入

执行单步进入，会执行下一行。如果下一行是一个方法，且该方法如果是自定义的方法，则会进入方法内。如果不是自定义的（官方类库的方法）则不会进入。

快捷键： fn + F7（macOS）或者F7（Windows/Linux）

调试工具栏： Step Into（见图 12-102）。

4. 强制进入

执行强制进入，会进入任何方法，不管该方法是自定义的还是官方类库的。

快捷键： fn + option + shift + F7（macOS）或者Alt + Shift + F7（Windows/Linux）

调试工具栏： Force Step Into（见图 12-103）。

图 12-101

5. 单步跳出

执行单步跳出，会跳出当前进入的方法，返回该方法被调用处的一下行。（注意：跳出意味着该方法被执行完毕。）

快捷键： fn + shift + F8（macOS）或者Shift + F8（Windows/Linux）

调试工具栏： Force Step Out（见图 12-104）。

图 12-102

图 12-103

6. 丢弃帧

如果你已经进入了某个方法内，执行丢弃帧，当前方法会被中断，并返回当前方法被调用的地方。另外，变量的值也会回到最初。

当我们想反复调试某个方法，观察此方法一遍遍地执行时，可以使用此功能。

操作步骤： 调试工具栏→Drop frame，如图 12-105 所示。

图 12-104

图 12-105

7. 运行到光标处

在调试时，光标可以放在任意一行。然后执行Run to Cursor，程序会运行到光标所在行暂停。其实这里的光标相当于一个临时断点。

快捷键： fn + option + F9（macOS）或者Alt + F9（Windows/Linux）

调试工具栏： 单击Run to Cursor，如图 12-106 所示。

8. 计算表达式

执行Evaluate Expression会打开计算表达式窗口。

快捷键： fn + option + F8（macOS）或者Alt + F8（Windows/Linux）

调试工具栏： Evaluate Expression（见图 12-107）。

图 12-106

图 12-107

关于计算表达式的用法下面会详细介绍。

12.10 计算表达式

当我们想临时修改某个变量的值或查看其内部方法返回值的时候，可以使用计算表达式功能。

Android Studio中提供了一个计算表达式和代码片段的功能，使用起来非常方便。它除了支持正则表达式计算以外，还支持操作表达式、匿名表达式和内部类的计算。

有以下两种计算模式。

- Expression Mode：计算单行表达式。
- Code Fragment Mode：计算代码片段，可以对声明、赋值、循环和 if/else 进行计算。

使用表达式计算功能时需要注意以下两点。

（1）只有在调试并且断点被触发的时候，才可以使用计算表达式功能，也就是说如果不是在调试的状态，计算表达式功能是不可用的。

（2）如果计算表达式内调用一个方法，方法内又恰好有断点，那么该断点会被忽略，直接计算出表达式的值。

12.10.1 在堆栈帧中计算表达式或代码片段

第 1 步： 在帧调试窗口中，选择想要计算表达式的堆栈帧。

第 2 步： 调用计算表达式功能，有以下 3 种方法。

菜单栏： Run →Evaluate Expression

快捷键： fn + option + F8（macOS）或者Alt + F8（Windows/Linux）

在编辑器中光标定位处右击，选择Evaluate Expression，如图 12-108 所示。

图 12-108

第 3 步：选择计算模式。

可以通过中间的按钮【Code Fragment Mode】和【Expression Mode】来切换单行表达式和代码片段这两种计算模式，如图 12-109 和 12-110 所示。

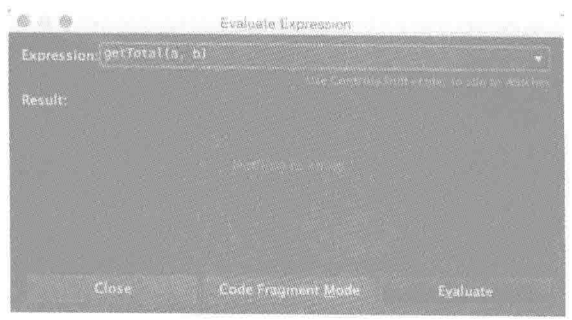


图 12-109

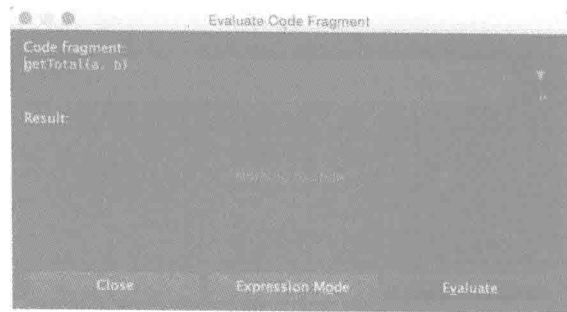


图 12-110

第 4 步：输入表达式或语句。

基于选择的模式输入表达式或语句，输入的时候会有补全提示，如图 12-111 所示。

如果给对象设置了标签，那么在输入表达式的时候可以通过标签来找到对应的表达式，Android Studio也会给出补全推荐，如图 12-112 所示。

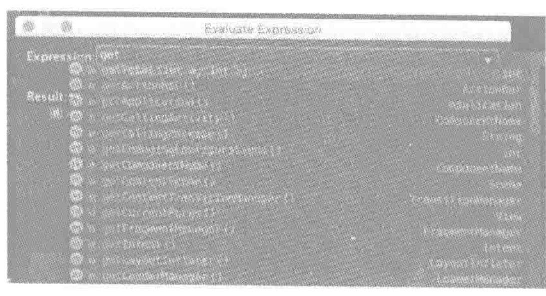


图 12-111

图 12-112

第 5 步：执行计算。

单击【Evaluate】来进行计算，如图 12-113 所示。

如果指定的表达式不能计算或是有错误，单击【Evaluate】后会给出可能的错误原因，如图 12-114 所示。

图 12-113

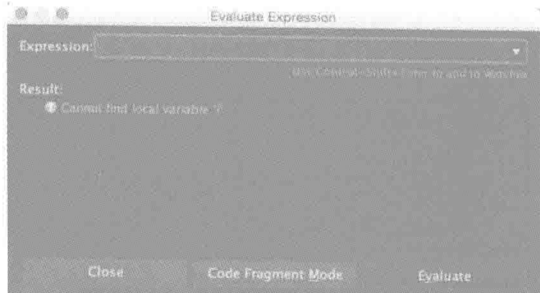


图 12-114

12.10.2 计算任意表达式

在变量调试窗口中右击变量→Evaluate Expression，如图 12-115 所示。

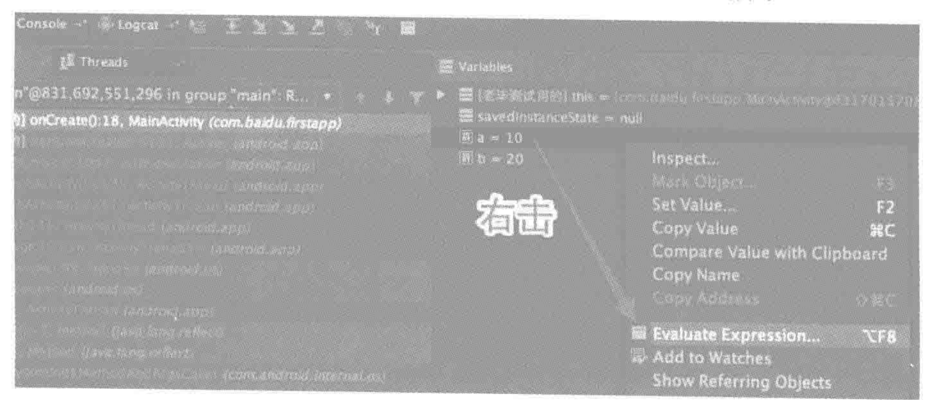


图 12-115

然后会进入表达式计算窗口，如图 12-116 所示。之前选中的变量会显示在表达式输入框中。在这里我们可以切换计算模式、查看计算过的历史、输入表达式，如图 12-117 所示。

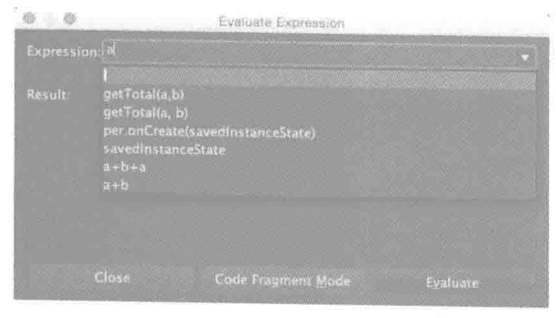


图 12-116

图 12-117

12.10.3 快速计算表达式的值

前提条件：将光标放在表达式上或选中表达式。

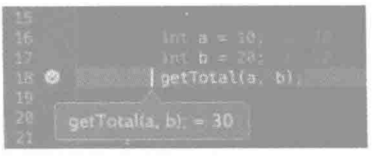


图 12-118

菜单栏：Run→Quick Evaluate Expression

快捷键：fn + option + command + F8（macOS）或者Ctrl + Alt + F8（Windows/Linux）

显示计算结果，如图 12-118 所示。

12.10.4 选中表达式立即显示表达式的值

在调试工具栏上单击设置按钮，勾选【show value on selection change】，如图 12-119 所示。

然后选中表达式，会在表达式的上面显示计算出来的结果，如图 12-120 所示。此功能默认是不开启的。

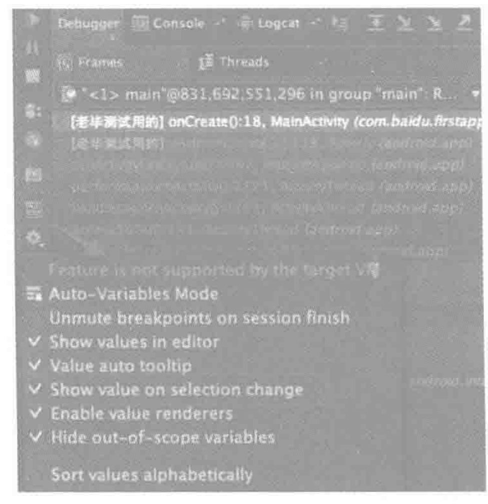


图 12-119

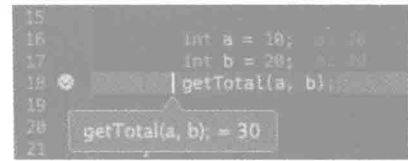


图 12-120

12.11 关联调试到 Android 进程

通常我们调试应用程序时，需要先添加断点，再运行调试（Debug）。这样做会比较慢，因为需要重新部署（打包、安装、运行）应用程序。

Android Studio提供了一种方法可以随时调试应用程序，不管当前应用程序是否以调试模式运行。

当我们想快速调试一个正在运行的应用程序时，可以使用此功能。

前提条件：应用程序已经在设备上运行，已添加断点。

操作步骤：菜单栏→Run→Attach Debugger to Android Process或者在工具栏中单击按钮（见图 12-121）。

图 12-121

然后弹出进程选择对话框，选择需要调试的应用程序的进程（见图 12-122）。确定后，调试器窗口被激活（见图 12-123），接下来正常调试应用程序。

图 12-122

图 12-123

12.12 配置和运行单元测试

12.12.1 配置和运行本地单元测试

本地单元测试

本地单元测试用来执行那些对Android没有依赖或Android依赖容易mock的单元测试。

本地单元测试运行在自己电脑上，测试用例在本地虚拟机上编译运行，执行速度快。

本地单元测试写在app/src/test/java目录下。

本地单元测试使用JUnit或TestNG测试框架。

使用的演示代码

这里使用Google官方开源的示例进行演示，地址为https://github.com/googlesamples/android-testing/tree/master/unit/BasicSample，也可以使用加了中文注释的例子：https://github.com/bxiaopeng/AndroidStudio/tree/master/chapter12/BasicSample。

使用的测试框架

使用JUnit测试框架举例。

使用IDE运行本地单元测试

打开BasicSample项目，在EmailValidatorTest编辑界面或项目窗口中的类名上右击→Run 'EmailValidatorTest'，如图 12-124 所示。

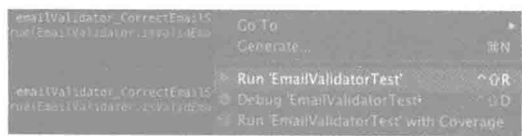


图 12-124

这样EmailValidatorTest这个测试类里面的所有测试用例都会被执行。执行结果如图 12-125 所示。

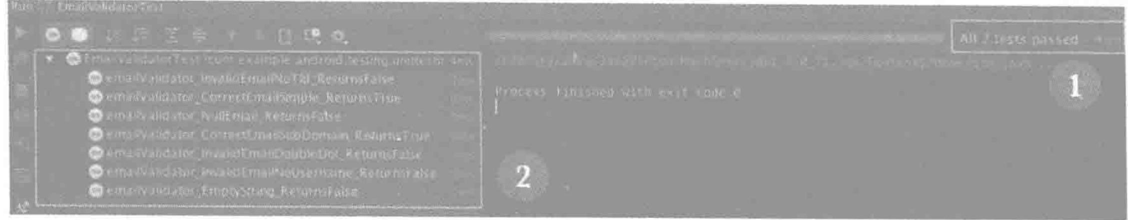


图 12-125

（1）显示测试结果：7 条测试用例通过，共耗时 4ms。

（2）显示测试列表：显示测试类及其测试用例的执行结果和执行时间。

提 示

如果只想执行某一个测试用例，在测试用例名上右击，选择‘Run 用例名’来执行。如果想让某个目录下所有的用例都被执行，在目录名上右击，选择‘Run Test in 目录名’来执行。

保存临时配置

通过上面的方法运行完测试之后，在工具栏的运行配置列表中显示了刚才运行的测试类。这个配置是自动生成的，如果需要保存，单击【Save '测试类名' Configuration】，如图 12-126 所示。

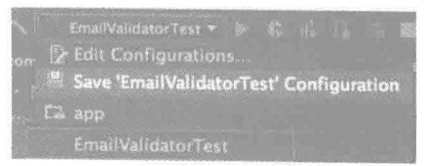


图 12-126

配置本地单元测试

需要自定义本地单元测试配置时，可以打开Run/Debug Configurations对话框→单击左上角的+号→JUnit，然后新增一个JUnit配置窗口，如图 12-127 所示。

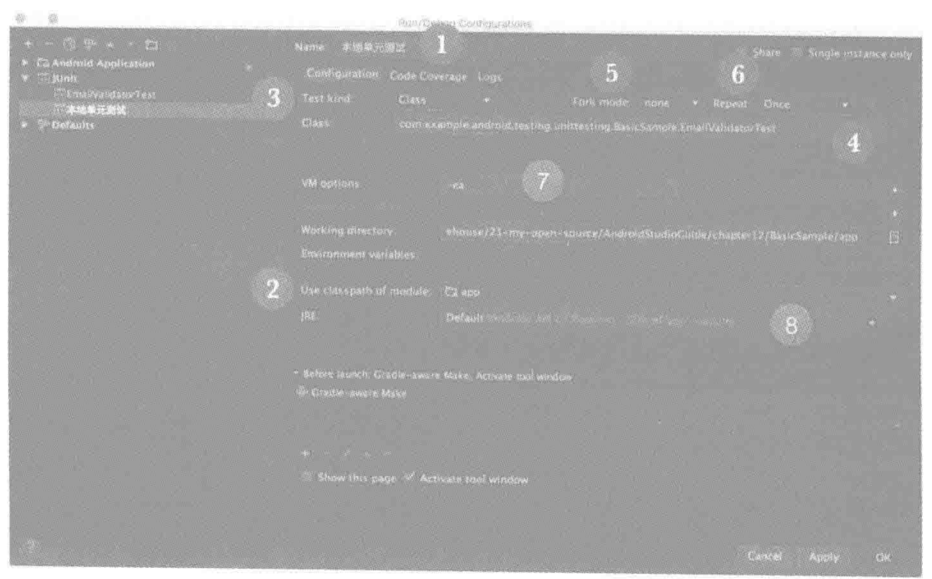


图 12-127

❶ Name：输入配置名。

❷ Use classpath of module：选择测试模块，下拉列表中会列出项目所有的模块，选择我们需要测试的。

❸ Test kind：选择一种测试范围，本例中选择的是Class。

❹ 测试用例配置：Class，选择测试类。

测试用例配置项跟我们选择的测试范围是相关联的。

- 如果 Test kind 选择 Class，显示测试类的配置项，如图 12-128 所示。

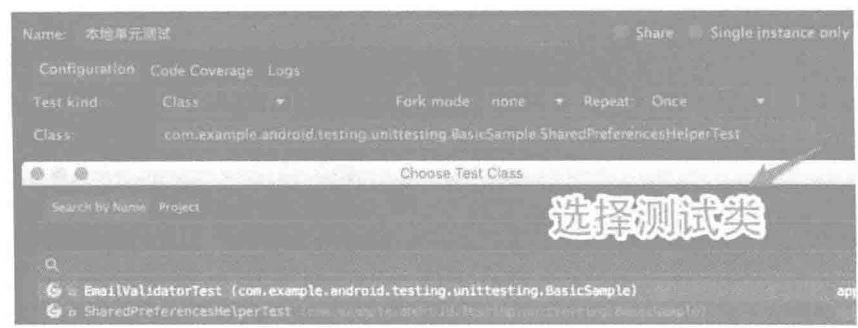


图 12-128

- 如果 Test kind 选择 Method，显示测试类和方法的配置项，如图 12-129 所示。

图 12-129

其他Test kind与此类似，这里就不一一列举了。

❺ Fork mode：用来指定是为每个测试用例都创建一个进程还是所有测试用例在一个进程中执行。有以下两个选项。

- none：在一个进程中执行所有测试，如图 12-130 所示。
  执行结果：耗时 8ms。

图 12-130

- method：每个测试用例都会创建一个测试进程，这种方式会比较慢，如图 12-131 所示。
  执行结果：耗时 19ms。

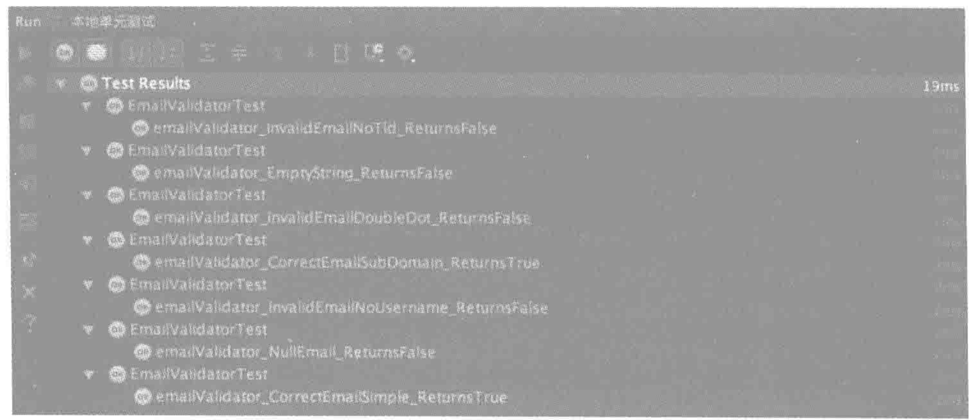


图 12-131

❻ Repeat：有以下 4 个选项。

- Once：所有用例只执行一次。
- N Times：自定义用例执行次数，比如设置为 5 次，每一条用例都会被执行 5 次。
- Until Failure：不断重复执行，直到用例失败。
- Until Stopped：不断重复执行，直到手动停止。

❼ VM options：默认值为-ea，用来设置jvm是否启动断言机制。
❽ JER：使用默认的设置即可。

12.12.2 使用命令行运行单元测试

```
./gradlew test
```

执行Gradle测试任务后所有的测试用例都会被运行，然后会产生测试报告。测试报告存放在app/build/reports/debug/index.html里，如图 12-132 所示。

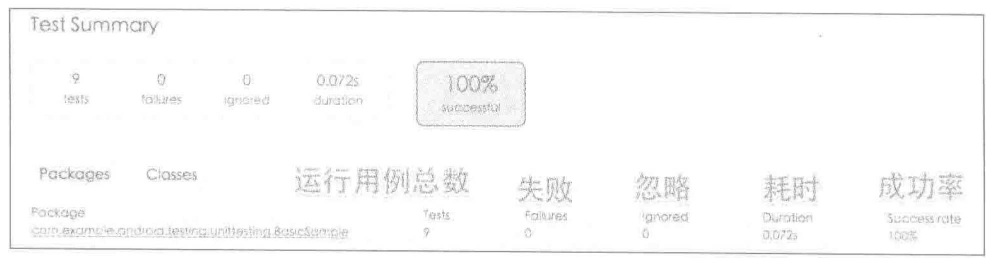


图 12-132

通过Gradle工具栏执行的效果同命令行。

12.12.3 配置Android单元测试

Android单元测试

Android单元测试用来执行那些有Android依赖或Android依赖不好mock的单元测试。Android单元测试运行在真机或模拟器上。

Android单元测试写在app/src/androidTest/java目录下。

Android单元测试使用Android Tests测试框架。

使用的演示代码

这里使用Google官方开源的示例进行演示，地址为https://github.com/googlesamples/android-testing/tree/master/unit/BasicUnitAndroidTest，也可以使用加了中文注释的例子：https://github.com/bxiaopeng/AndroidStudio/tree/master/chapter12/BasicUnitAndroidTest。

使用的测试框架

使用Android Tests测试框架。

配置Android单元测试

需要自定义本地单元测试配置，可以打开Run/Debug Configurations对话框→单击左上角的+号→Android Tests，新增一个Android Tests配置窗口（见图 12-133）。

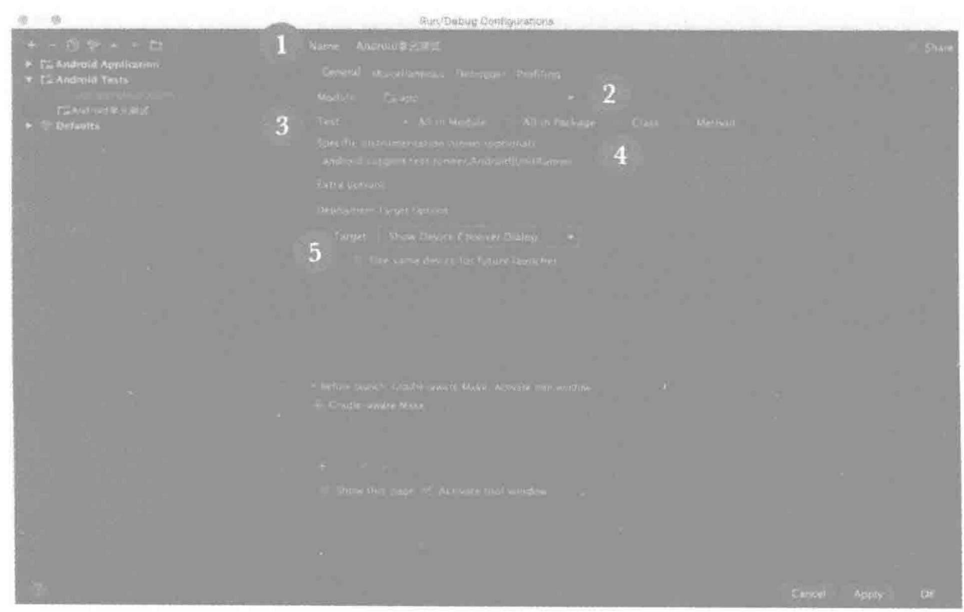


图 12-133

❶ Name：输入配置名。
❷ Module：选择需要测试的模块。
❸ Test：指定测试范围。
❹ 指定instrumentation runner。
❺ 指定目标设备。

提 示

运行 Android 单元测试方法同运行本地单元测试。

第 13 章　工　具

Android Studio中集成了很多非常好用的工具，除了IntelliJ IDEA自带任务管理、生成Java Doc等工具，还集成了以前放在SDK目录中可以独立运行的Android开发工具，如Android Monitor（以前叫DDMS）、Android SDK Manager、模拟器等，现在它们已经被全部集成到了Android Studio当中，而且变得更好用。

本章将向大家介绍Android Studio中集成的非常好用的工具。

本章重要知识点 >>>>>>>>>>

- 如何使用任务管理；
- 如何管理和使用模拟器；
- 如何管理 Android SDK；
- 如何使用即时运行；
- 如何使用监视器工具（Logcat、CPU、内存、GPU、流量）。

13.1　任　务

13.1.1　任务介绍和配置

1. 任务是什么

任务（Task）就是我们要完成的一项工作，比如解决一个分配给自己的BUG或完成一个分配给自己的需求。这些任务是通过任务跟踪管理系统来进行管理的。

Android Studio支持与很多任务跟踪管理系统进行集成，只要配置一个任务跟踪管理系统并与我们的账号绑定，就可以方便地创建、跟踪、处理分配给自己的任务。

Android Studio支持的任务跟踪管理系统如图 13-1 所示。

一个任务可以对应多个changelist，changelist就是我们修改过的文件。我们可以给每一个changelist起名字，当完成一项任务后，可以选择提交相应的change list。

下面我们来介绍下如何使用任务的配置、修改、提交、删除等功能。

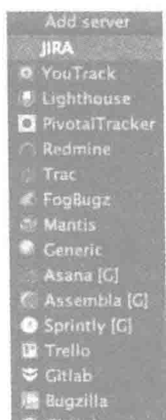


图 13-1

2. 配置任务

（1）进入配置界面

操作步骤：

- 偏好设置→Tools→Tasks→Servers。
- Tools→Tasks & Contents→Configure Servers。
- open task 右上角的 setting（见图 13-2）。

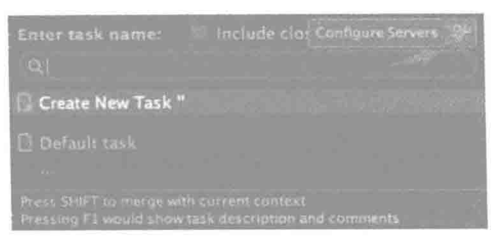


图 13-2

（2）配置 JIRA

JIRA是集项目计划、任务分配、需求管理、错误跟踪于一体的商业软件，在很多公司中使用比较广泛。

下面我们来介绍在Android Studio如何配置JIRA服务。

01 进入服务器配置界面：偏好设置→Tools → Tasks → Servers。

02 添加 JIRA 服务器：单击【+】按钮，选择 JIRA。

03 输入 URL 和账号：输入 JIRA 的 URL 地址，再输入用户名和密码，如图 13-3 所示。

图 13-3

测试配置是否正确：单击【Test】，提示连接成功，如图 13-4 所示。

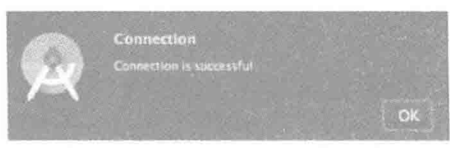


图 13-4

（3）配置 GitHub

01 进入服务器配置界面：偏好设置→Tools → Tasks → Servers。

02 添加 JIRA 服务器：单击【+】按钮，选择 GitHub。

03 输入 URL 和账号：输入相关信息，如图 13-5 所示。

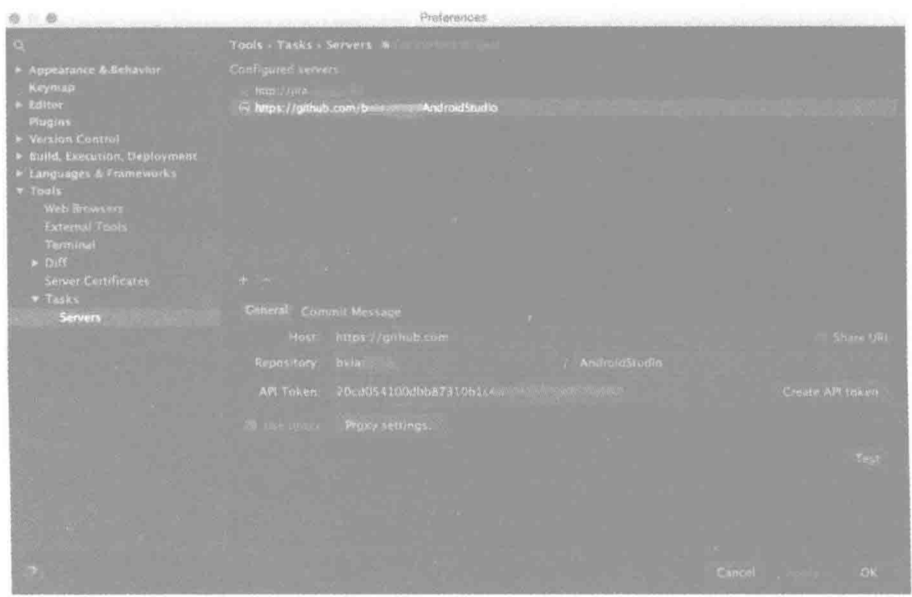


图 13-5

如果不记得API Token，可以单击Create API token来创建一个。可以单击【Test】，测试配置是否正确。

（4）配置 Gitlab

01 进入服务器配置界面：偏好设置 →Tools → Tasks → Servers。

02 添加 JIRA 服务器：单击【+】按钮，选择 Gitlab。

03 输入 URL 和 Token，并选择对应的项目。

Token在Gitlab里可以找到，如图 13-6 所示。

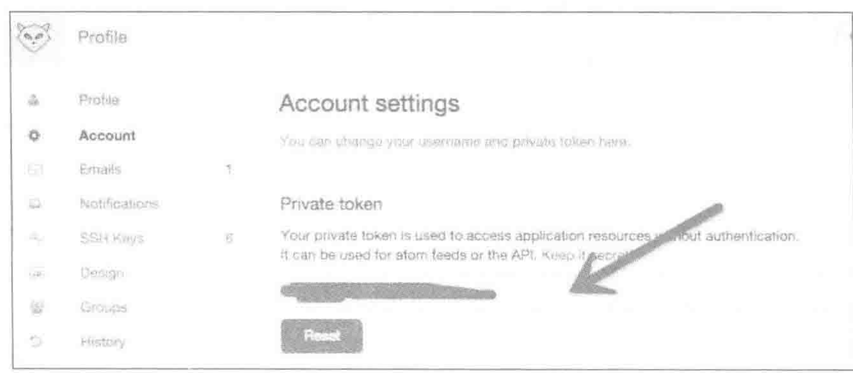


图 13-6

配置结果如图 13-7 所示。

同样，单击【Test】来测试配置是否正确。另外，如果所使用的服务器需要设置代理，可以使用Proxy setting来设置。

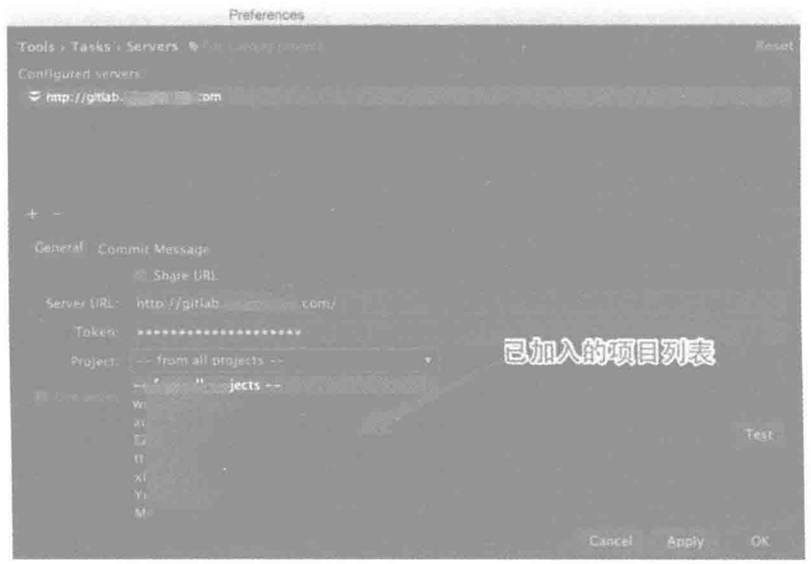


图 13-7

13.1.2 打开任务

1. 直接打开任务

菜单栏：Tools→Tasks & Contents→Open Tasks

快捷键：option + shift + N（macOS）或者Alt + Shift + N（Windows/Linux）

如果在配置的任务跟踪管理系统中有任务分配给你，那么这里就会列出相关的任务，如图 13-8 所示。

2. 搜索任务

如果任务太多需要搜索，通过输入任务名会进行相关的匹配，如图 13-9 所示。

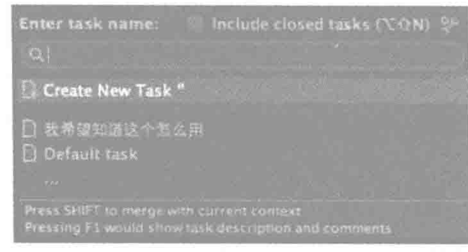


图 13-8

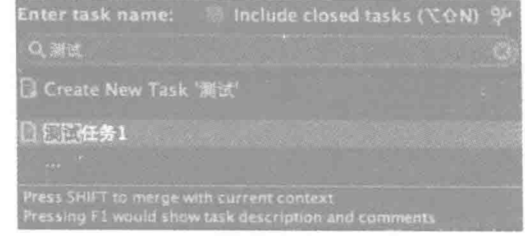


图 13-9

3. 包括已关闭的任务

默认显示的任务都是未关闭状态的，如果想查看已关闭的任务，需要勾选【Include closed tasks】，如图 13-10 所示。

4. 查看任务的描述和注释信息

使用fn+F1 键可以查看任务的描述和注释信息，如图 13-11 所示。

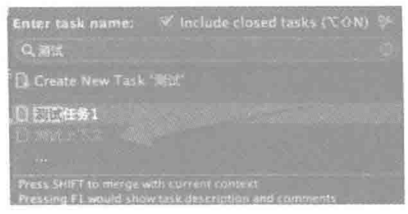


图 13-10

图 13-11

5. 切换任务并合并上下文

在Open Task界面，选中一个任务，按Shift + 回车键会切换到选中的任务，上下文（context）会合并。

正常情况下，如果切换任务，上下文也是切换的，之前的上下文并不会保留，除非使用上面的方法。

13.1.3 创建新任务

如图 13-12 所示，当我们想要创建一个新任务时，可以在Open Task界面输入新任务的名字，按回车键就会开始创建。

按回车键后打开新任务的配置界面，如图 13-13 所示。

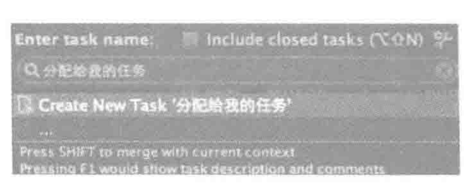


图 13-12

图 13-13

新建一个任务的时候会新建一个分支，分支的名字默认为任务名，可以随意修改，它会跟任务相对应。同理，changelist的名字也可以随意修改，一个任务可以对应多个changelist，如果对应了多个changelist，在提交修改的时候需要每一个都提交一次。

默认情况下，新建一个任务时会清空前一个任务的上下文。如果不想清空，需要取消勾选【Clear current context】。单击【OK】按钮后，任务被创建（对应的分支和changelist也同时被创建），如图 13-14 所示。

图 13-14

13.1.4 任务变更列表

1. 新建changelist

前面我们讲了新建任务时会创建一个changelist，如果想再创建一个changelist，可以按照图 13-15 所示进行操作。

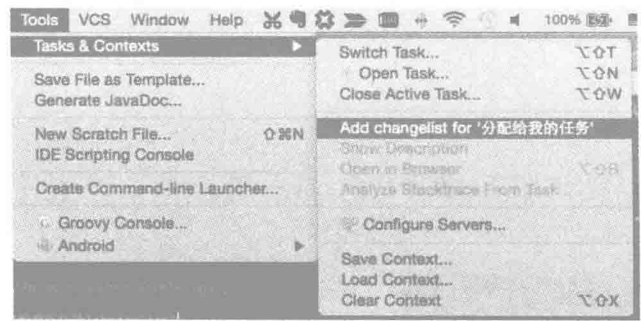


图 13-15

输入新的changelist的名字，如图 13-16 所示。

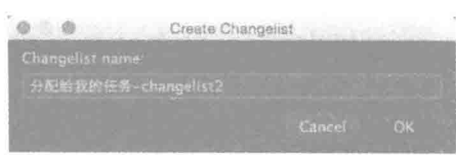


图 13-16

记住，在新建一个changelist之前，所有的文件改动都属于上一个changelist，新建一个changelist之后的所有变更都属于新建的changelist。

2. 提交change list（见图 13-17）

菜单栏：VCS→Commit Changes

快捷键：command + K（macOS）

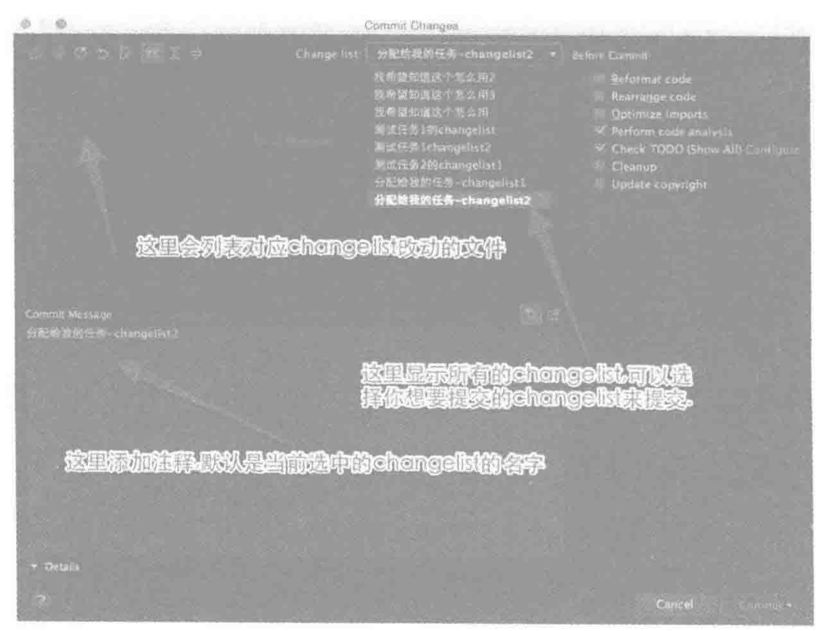


图 13-17　提交 change list

13.1.5 切换/关闭任务

1. 切换任务

操作步骤：

工具栏→打开任务列表→双击切换（也可以单击switch to），如图 13-18 所示。

切换任务以后，之前任务的上下文会被关闭，当前任务的上下文将被打开，分支也会切换到当前任务所对应的分支。

2. 关闭任务

当一个任务完成以后，可以通过**菜单栏**：Tools→ Tasks & Contexts→ Close Active Task（见图 13-19）或者快捷键option + shift + W来关闭任务。

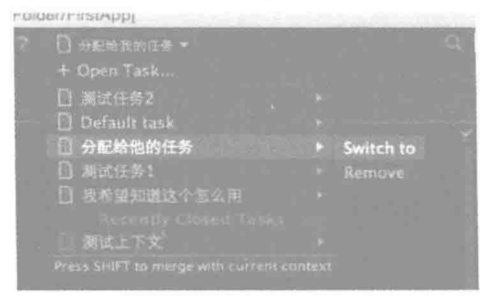


图 13-18

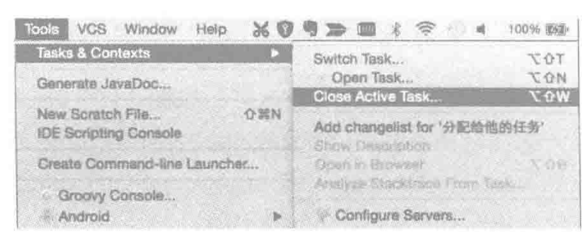


图 13-19

之后会弹出确认窗口让我们确认删除任务时是否要提交变更和合并分支（分支合并到master上，如图 13-20 所示）。确认后之前任务的分支将被删除，当前分支切换到maser上，如图 13-21 所示。然后在Recently Closed Tasks可以查看最近关闭的任务。

如果要删除任务，可以执行Remove操作，如图 13-22 所示。

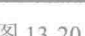


图 13-20

图 13-21

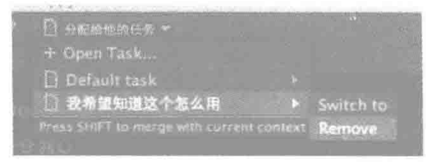


图 13-22

13.1.6 管理上下文

1. 上下文是什么

上下文（Context）是指与一个任务（Task）相关的在编辑器中被打开的文件，包括变更列表、编辑器、项目查看状态、运行配置和断点。

上下文与任务是密切相关的，新建一个任务，上一个任务的上下文是要被清空的，表现就是工作区已打开的文件全部被关闭，然后新的上下文就开始被建立。

我们可以方便地对这些任务的上下文进行管理，Android Studio提供了保存、载入、清除上下文的功能，所以上下文也可以独立于任务 ，可以载入之前保存过的上下文。操作菜单如图 13-23 所示。

2. 保存上下文

可以选择菜单栏：Tools→Tasks & Contexts→Save Context→在弹出的对话框中为要保存的上下文起名，确定后即可被保存。

3. 载入上下文

可以选择菜单栏：Tasks & Contexts→Load Context→弹出之前保存过的context list（见图 13-24）。在这个列表里加载、合并、移除上下文。如果选择了load上下文，当前选中的上下文将被载入。

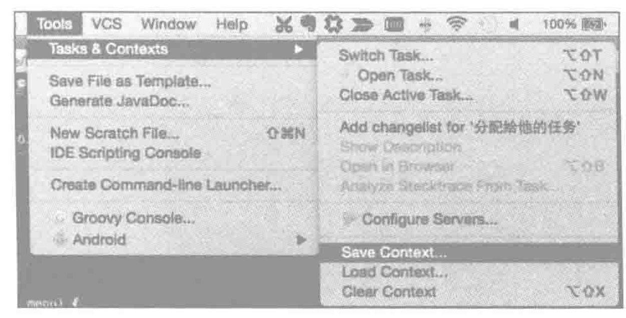


图 13-23

图 13-24

4. 清除上下文

清除上下文功能会把当前所有上下文的信息全部清除，可以通过菜单栏：Tools→Tasks & Contexts→Clear Context或快捷键option + shift + x（macOS）来完成。

13.2 JavaDoc

JavaDoc是Sun公司提供的一个技术，从程序源代码中抽取类、方法、成员等注释形成一个和源代码配套的API帮助文档。

也就是说，只要在编写程序时以一套特定的标签做注释，在程序编写完成后，通过Javadoc就可以同时形成程序的开发文档了。

13.2.1 配置JavaDoc

在Android Studio中配置JavaDoc可以通过偏好设置→ Editor→Code Style→Java→ JavaDoc来完成，如图 13-25 所示。JavaDoc格式配置如图 13-26 所示。

图 13-25

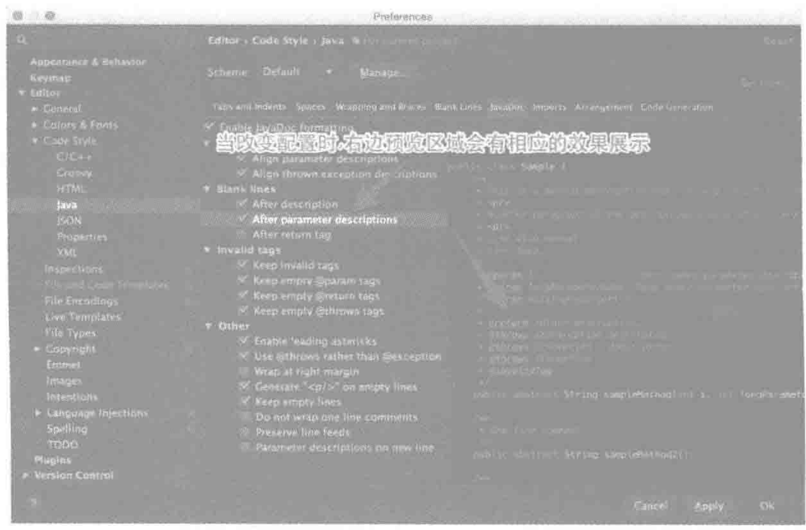


图 13-26

当我们添加注释时，会使用配置好的JavaDoc格式，如图 13-27 所示。

```
37        /**
38         * |
39         * @param a
40         * @param b
41         * @return
42         */
43        public int getTotal(int a, int b) {
44            int total = a + b;
45            Log.i("TOTAL: ", String.valueOf(total));
```

图 13-27

13.2.2　生成JavaDoc

通过菜单栏：Tools→Generate JavaDoc来指定生成JavaDoc的范围和参数。配置窗口如图 13-28 所示。在这里可以指定生成JavaDoc的范围、输出目录、输出内容和参数等。

特别注意Other command line arguments的配置，如果项目中使用的是UTF8 有编码格式，在这里要指定参数：-encoding utf-8 -charset utf-8，否则生成的JavaDoc中文会出现乱码。

配置好以后单击【OK】按钮，会开始执行JavaDoc任务，如图 13-29 所示。

图 13-28

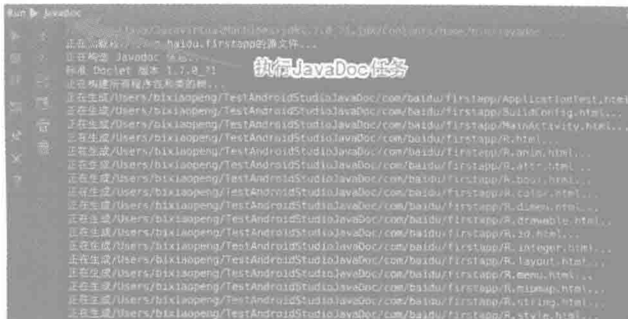


图 13-29

生成JavaDoc时会先执行Gradle构建任务，如图 13-30 所示。生成结果如图 13-31 所示。

图 13-30

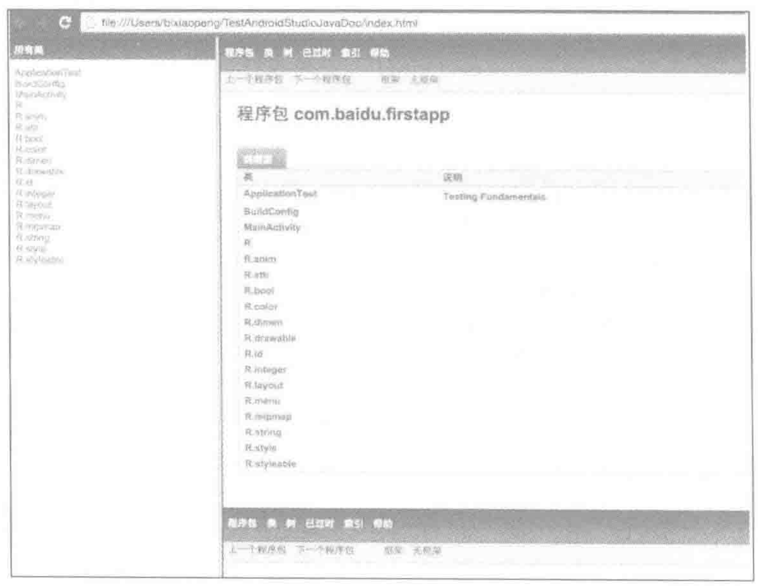


图 13-31

输出目录，如图 13-32 所示。

提 示

如果生成 JavaDoc 时报错说 android.jar 找不到，就需要在 Dependencies 中添加 android.jar。

图 13-32

13.3　将当前文件保存为模板

前提条件：光标定位在编辑界面。

操作步骤：菜单栏→Tools→Save File as Template（见图 13-33）→弹出模板配置窗口（见图 13-34）→单击【OK】按钮后模板被创建。

图 13-33

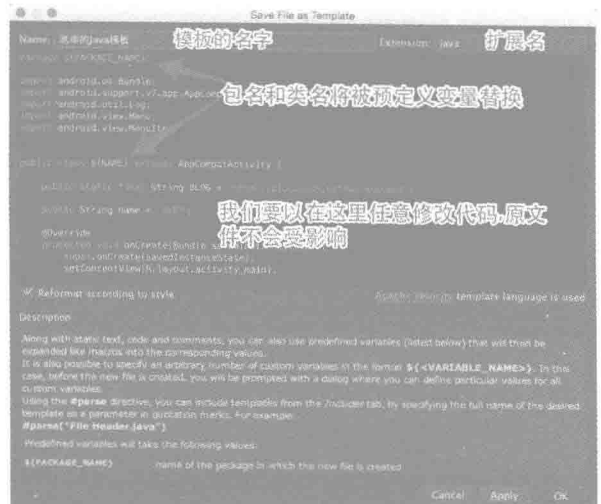


图 13-34

当新建一个文件时，可以看到刚才创建的模板选项，如图 13-35 所示。

如果想删除或重新编辑模板（见图 13-36），就单击删除按钮或在菜单栏中单击File→New→【Edit File Templates】。

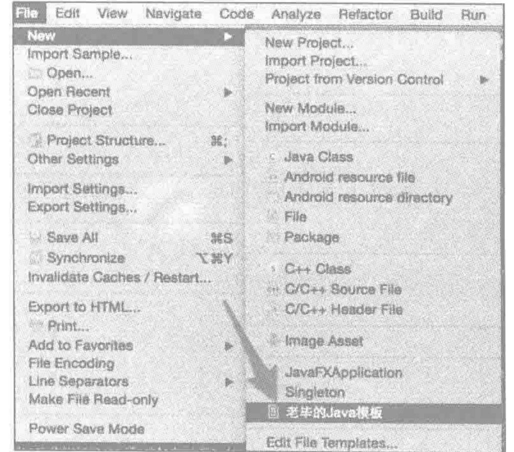


图 13-35

图 13-36

13.4 IDE Scripting Console

使用IDE Scripting Console可以测试Groovy的代码片段。

操作步骤：菜单栏→Tools→IDE Scripting Console。

当项目中有两个及两个以上的模块时会让你选择模块使用的类路径，选择后控制台显示如图 13-37 所示，展示控制台对Groovy代码的自动补全功能。运行和关闭功能如图 13-38 所示。

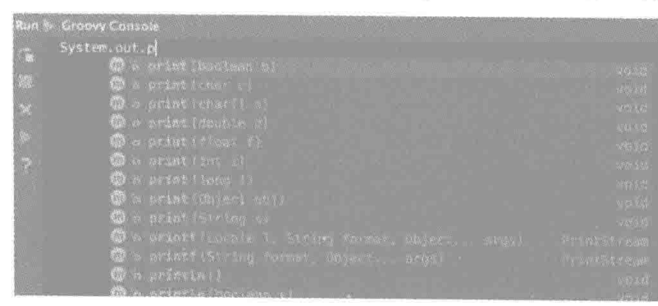


图 13-37

图 13-38

更多Groovy的语法，请利用Google搜索。

13.5 管理 Android SDK

Android SDK管理Android Studio中使用的SDK和工具，我们可以在这里查看、安装、启动、删除SDK。可以通过以下 3 种方法进入Android SDK。

- 菜单栏：Tools→Android→SDK Manager

- 偏好设置：Appearance & Behavior→System Settings→Android SDK
- 工具栏（见图 13-39）。

图 13-39

13.5.1 管理Android SDK平台

每个Android SDK平台的安装包都默认包含了与API级别对应的Android平台和相关的源码（见图 13-40），安装后，Android Studio会自动检测更新。

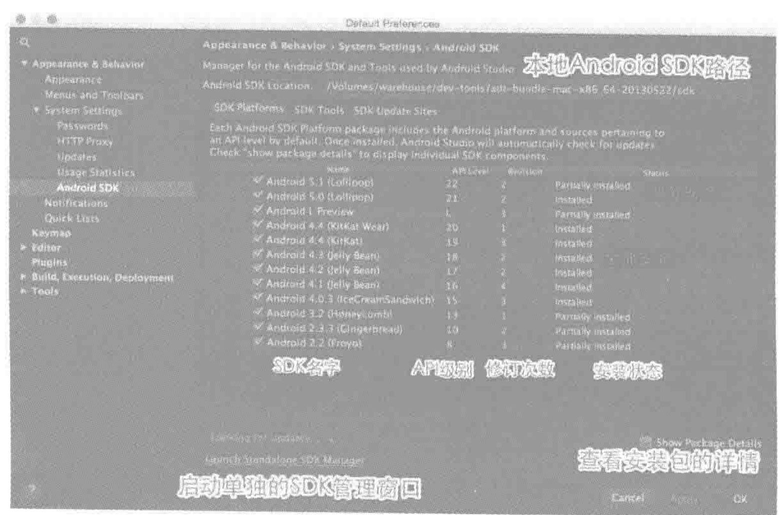


图 13-40

（1）查看安装包的详情：勾选右下角的【Show Package Details】可以显示安装包的详情。
（2）删除已安装的包，如图 13-41 所示。

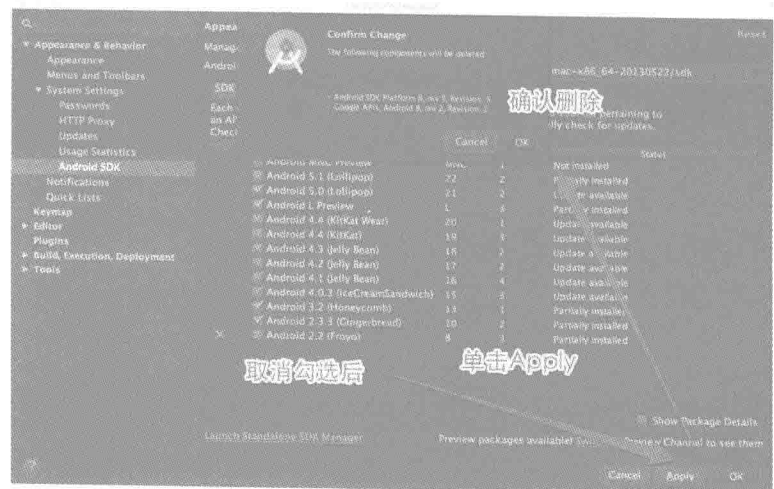


图 13-41

01 取消勾选【Show Package Details】。

02 单击【Apply】，弹出确认删除对话框。

03 单击【OK】按钮。

（3）安装组件，如图 13-42 所示。

01 勾选【Show Package Details】，显示每个包的安装组件。

02 选中某个组件，单击【Apply】按钮，弹出确认安装对话框，如图 13-42 所示。

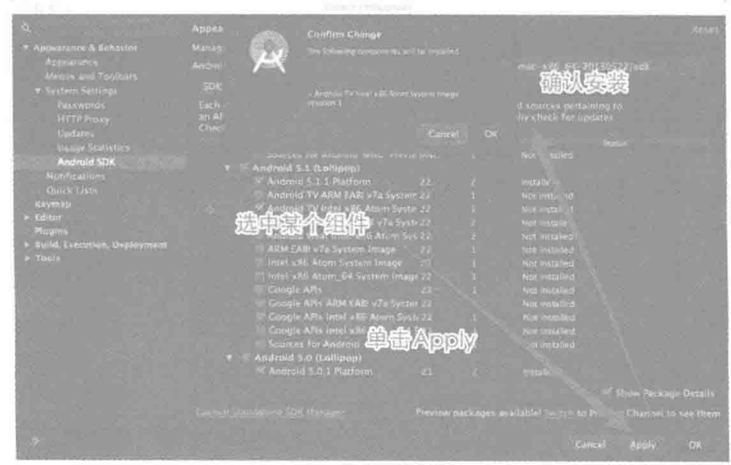


图 13-42

03 单击【OK】按钮开始安装组件，如图 13-43 所示。

图 13-43

（4）启动单独的SDK管理窗口，如图 13-44 所示。

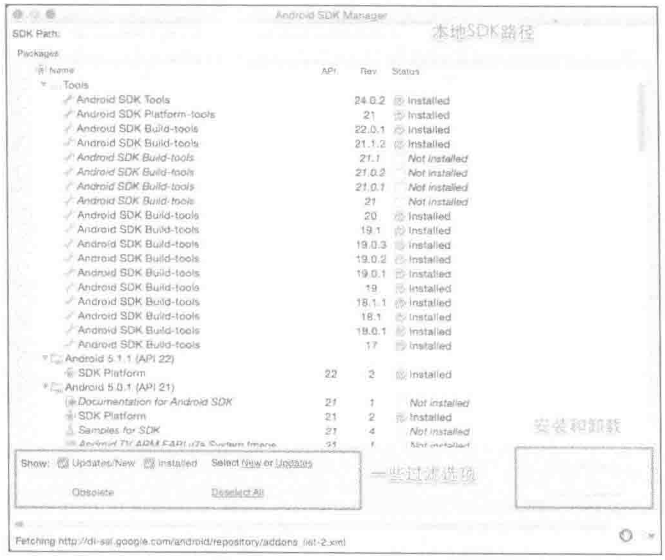


图 13-44

（5）在单独的SDK管理窗口中删除已安装的包，如图 13-45、图 13-46、图 13-47 所示。

图 13-45

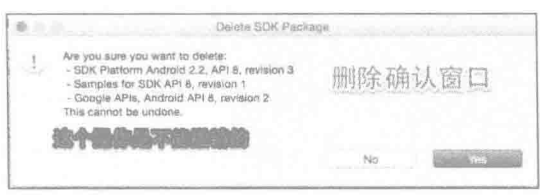


图 13-46

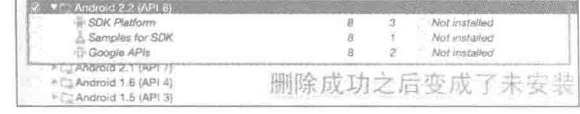


图 13-47

13.5.2 管理SDK开发工具和更新站点

SDK Tools选项卡中显示了Android SDK的开发工具，安装后可自动检测更新，勾选【Show Package Details】会显示所有可用的SDK版本，如图 13-48 所示。

图 13-48

SDK Update Sites选项卡用来管理Android SDK工具更新的站点。如果站点没有被勾选，Android Studio SDK Manager就不会去检测更新，如图 13-49 所示。利用左下角的按钮可以添加自定义的更新站点。

图 13-49

13.6 管理 Android 模拟器

Android Virtual Device（Android虚拟设备）通常被称为Android模拟器，是Google官方提供的工具，用于在电脑上模拟出Android手机的运行环境。

Android模拟器是我们开发、调试和适配中不可缺少的工具，但由于以前官方模拟器性能很差，很多人都已经放弃了使用，转而使用性能强大的Genymotion。自从Android Studio 2.0版本开始，Google对Android模拟器进行了一系列的优化，其速度、性能、易用性都得到了非常大的提升，也许以后你会喜欢上官方的模拟器。我们可以在AVD Manager （Android Virtual Device Manager）中管理Android模拟器。

13.6.1 认识模拟器

从工具栏（见图 13-50）可以快速打开Android模拟器管理界面，其中列出了已创建的模拟器（见图 13-51）。

图 13-50

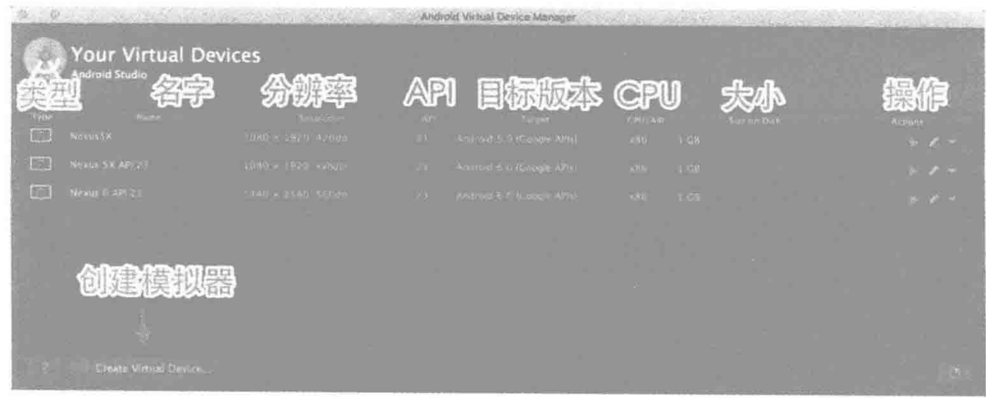


图 13-51

我们可以对模拟器进行运行、编辑、复制、擦除数据、在磁盘中查看、查看详情、删除、停止等操作，如图 13-52 所示。

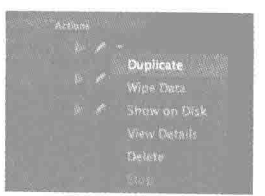


图 13-52

13.6.2 创建模拟器

在模拟器管理界面中单击左下角的【Create Virtual Device】，进入模拟器配置界面，如图 13-53 所示。

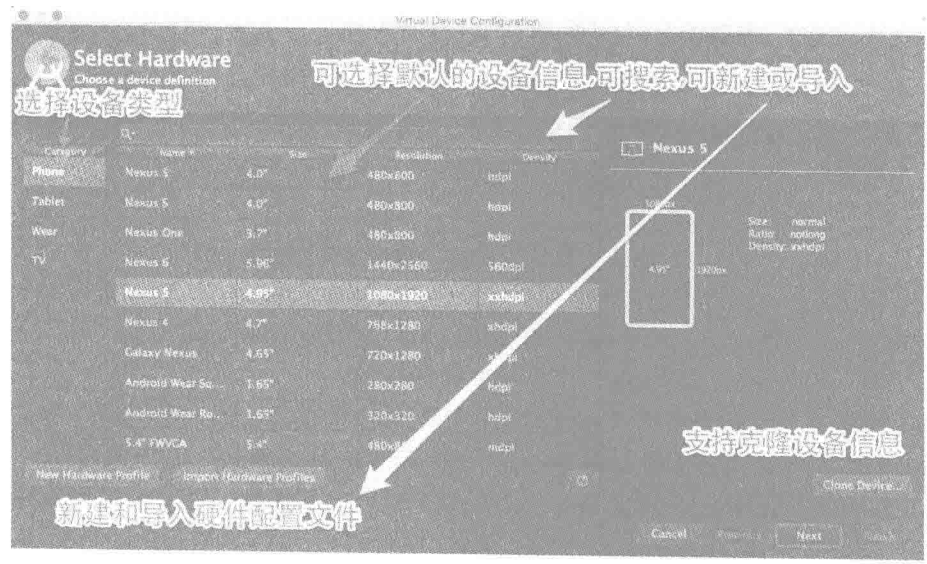


图 13-53

这里我们使用默认的设备，单击【Next】按钮，选择一个系统映像，如图 13-54 所示。

图 13-54

选择后单击【Next】按钮，最后核对一遍配置，如图 13-55 所示。

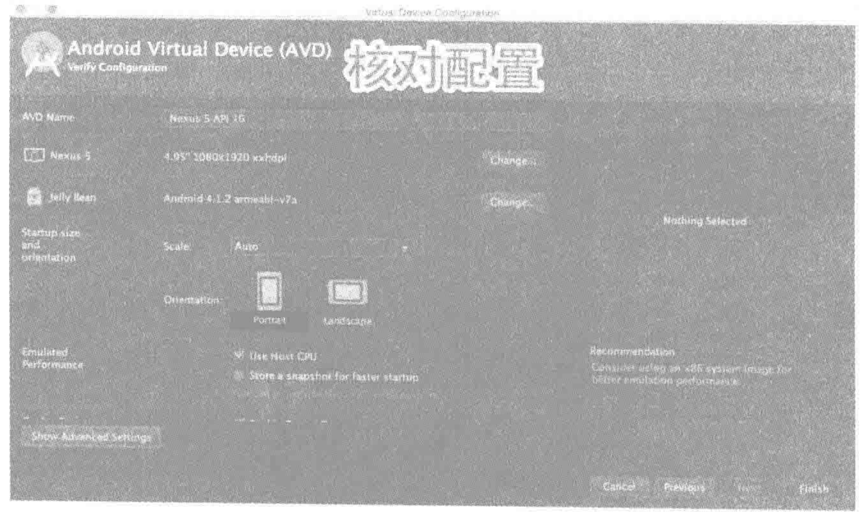


图 13-55

单击【Finish】按钮，创建成功。

13.6.3 启动模拟器

默认情况下，Android Studio在x86 模拟器系统映像上使用 CPU加速，支持Android 6.0 系统映像中新的对称多重处理器（SMP），执行速度媲美Genymotion。

启动模拟器，如图 13-56 所示。新版的模拟器左边工具栏提供了常用的操作工具，而且每个操作都提供了相应的快捷键（见表 13-1），使用起来非常的方便。

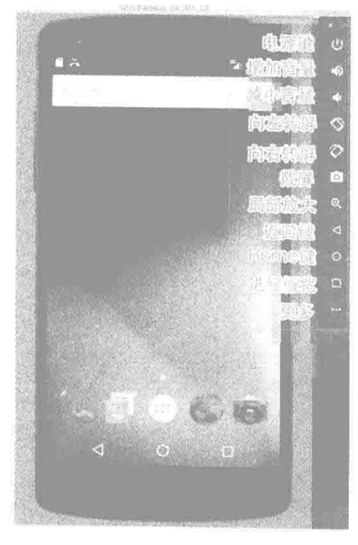


图 13-56

表 13-1

功能	快捷键（macOS）	快捷键（Windows）
电源键	command + P	Ctrl + P
增加音量	command + +	Ctrl + +
减小音量	command + -	Ctrl + -
向左转屏	command + ←	Ctrl + ←
向右转屏	command + →	Ctrl + →
截屏	command + S	Ctrl + S
局部放大	command + Z	Ctrl + Z
返回键	command + delete	Ctrl + delete
Home 键	command + shift + H	Ctrl + Shift + H
进程概览	command + O	Ctrl + O

大部分功能一眼看过去就知道怎么用了，这里不多介绍，下面挑几个特殊的来介绍一下。

（1）电源键

点一下是锁屏，长按是关机。

（2）窗口缩放和局部放大

既可以缩放模拟器的窗口大小，还可以使用局部放大工具查看某一个局部的细节，如图13-57和图13-58所示。

图 13-57

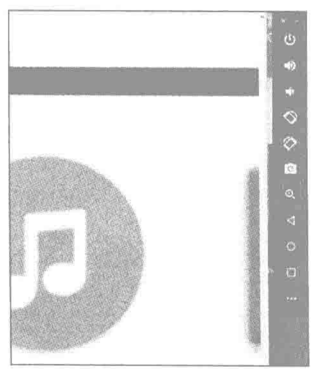


图 13-58

（3）快速安装/查看/卸载应用

可以通过拖放快速安装应用，并可通过在应用列表中长按应用来查看应用信息和卸载应用。

（4）SD 卡设置

如果没有设置过SD卡，在通知栏会有SD卡设置提示（见图 13-59），可以单击【SET UP】按钮进行设置。

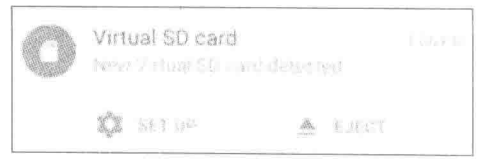


图 13-59

这里提供了两种SD卡的存储模式（见图13-60），当你选择了一种存储模式想切换为另一种时，可利用切换存储模式的功能（见图 13-61），但会把SD卡格式化。

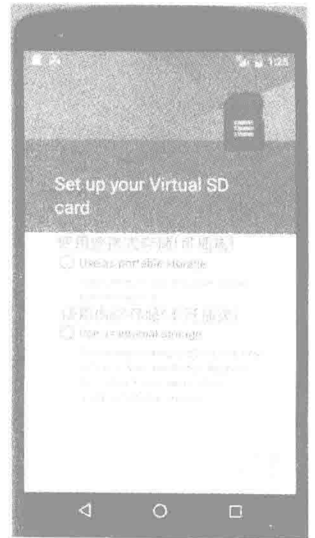


图 13-60

图 13-61

13.7 即时运行

如今的移动开发最注重的就是速度和敏捷，但是我们在构建APK时却感到笨重和缓慢。Instant Run会让我们的开发过程变得快速流畅。

1. 即时运行是什么

即时运行（Instant Run）是Android Studio 2.0 推出的一个革命性的功能，使用即时运行功能可以做到一边写代码一边在设备（真机或模拟器）上运行，可以即时查看运行效果。

单击【Instant Run】按钮时，会先分析我们修改的内容，然后决定以最快的方式部署新的代码。

即时运行支持“热交换”“暖交换”和“冷交换”。

- 热交换：能够在 APP 运行时处理变更方法的具体实现。
- 暖交换：能够在 APP 运行时把资源文件注入应用中。
- 冷交换：能够在 APP 运行时处理一些结构性的变更，如类的层次结构变更、方法签名的变更等。

2. 启用即时运行

操作步骤：偏好设置→Instant Run→勾选【Enable Instant Run to hot swap code/resource changes on deploy】（启用即时运行），如图 13-62 所示。

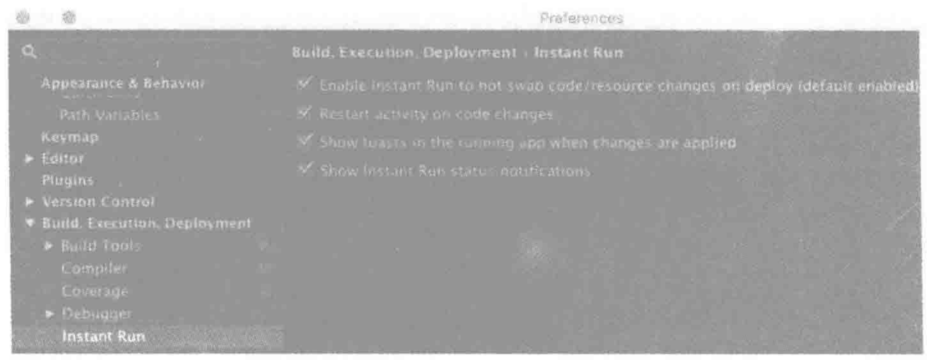


图 13-62

- Restart activity on code changes：代码变更时重启 activity。
- Show toasts in the running app when changes are applied：每次变更被应用时都显示 toast 提示。
- Show Instant Run status notifications：显示即时运行状态通知。

如果Instant Run显示如图 13-63 所示的界面，说明你的Android 插件还没有更新，请先单击Update Project来更新插件。

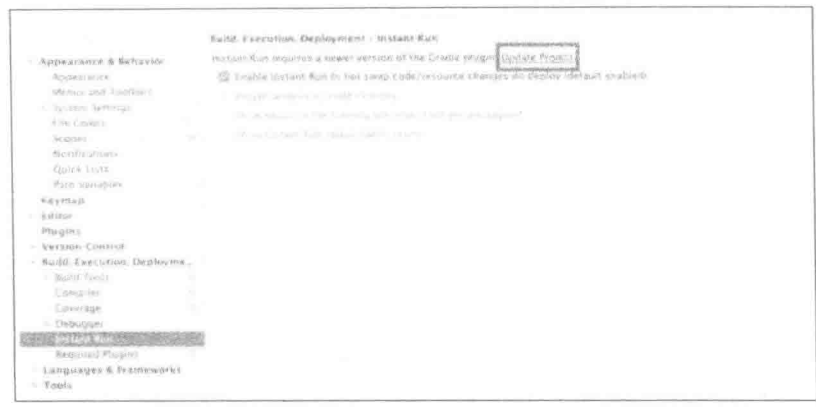


图 13-63

3. 使用即时运行之前需要知道的事情

（1）即时运行只支持 Android 4.0 （API level 15）及以上的版本。

（2）即时运行只支持 Debug版本，所以运行前请先确保构建的APK是Debug（可以通过View→Tool Windows→Build Variants查看，见图 13-64）。

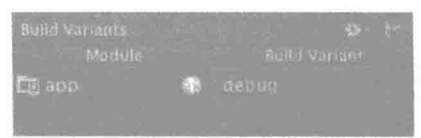


图 13-64

4. 如何判断是否已开启即时运行

通过一张图就能够判断出来，如图 13-65 所示。

图 13-65

注意，重点是那个闪电标识，有了这个标识就说明已经开启即时运行。

5. 即时运行是如何工作的

即时运行被设计用来加快应用的构建和部署，通过执行“热交换”“暖交换”和“冷交换”把变更的代码和资源推送到设备上（模拟器或真机）。即时运行能够自动判断被推送来的变更类型，然后执行相应的交换类型（见表 13-2）。

表 13-2

代码变更	即时运行行为
改变现有方法的实现代码	支持“热交换”：APP 保持正常运行，待下次方法被调用时会使用新的实现
添加/删除或改变现有的资源	支持“暖交换”：变更的资源会被推送到设备上，然后 Activity 会自动重启

（续表）

代码变更	即时运行行为
结构代码变更，如： 1. 添加、删除或改变 - 注解 - 实例字段 - 静态字段 - 静态方法签名 - 实例方法签名 2. 改变当前类继承的父类 3. 改变实现的接口列表 4. 改变类的静态初始值设定 5. 利用动态的资源 ID 重新布局元素	支持“冷交换”（API≥21）：结构代码变更被推送到目标设备，然后整个应用重启
1.改变应用的 manifest 2.应用的 manifest 中改变了资源的引用	不支持。即时运行不支持改变 Android manifest 或 manifest 中资源的引用，这是因为 APK 中包含的某些信息（如名字、icon 资源和 intent filters）是从 Android manifest 中确定的。 当 APK 已经被安装到设备上时，如果改变了 manifest 就必须重装 APK。由于热、暖、冷交互是为了避免重新安装 APK ，为了加快速度，如果改变了 Android manifest，Android Studio 会自动部署一个新的构建。

【实例演示】

第 1 步：新建一个项目，Activity 选择 Basic Activity。

第 2 步：确认即时运行已经被启用。

第 3 步：确认项目的 build.gradle 版本。

```
buildscript {
    repositories {
        jcenter ()
    }
    dependencies {
        //android 构建工具的版本要大于等于 2.0.0
        classpath 'com.android.tools.build: gradle: 2.0.0-beta6'
    }
}
```

第 4 步：运行，如图 13-66 所示。这时运行按钮上显示闪电标识。

第 5 步：在 MainActivity 中添加一段代码，如图 13-67 所示。

第 6 步：单击运行。非常快就构建好了，单击按钮 Toast 生效，如图 13-68 所示。如果想看构建详情，可以打开运行工具窗口。

图 13-66　　图 13-67　　图 13-68

如图 13-69 所示，执行了“热交换”变更。

第 7 步：不做任何改动，再单击一次运行，如图 13-70 所示，结果没有任何变更，不用部署 APK。

图 13-69　　图 13-70

13.8 Android 监视器

Android Studio集成了非常丰富的工具来对APP进行监控和调试，我们可以在Android 监视器查看日志、截图、录像、解析布局、实时查看性能。

使用Android监视器的前提是：真机或模拟器连接到电脑，并且指定APP的进程（必须先启用ADB集成）。

1. 启用ADB集成

Android Studio默认是没有启用ADB集成的，所以我们看不到APP的进程，也就无法对应用进行监控和调试，如图 13-71 所示。

图 13-71

想要使用监控工具，首先要启用ADB集成。

操作步骤：菜单栏→Tools→Android→勾选Enable ADB Integration。

启用后就可以看到指定手机上运行的APP进程了，如图 13-72 所示

提 示

如果开启后设备和进程都看不到了，重启 Android Studio 即可。

图 13-72

2. 打开/隐藏 Android Monitor工具窗口

快捷键：command + 6（macOS）或者Alt + 6（Windows/Linux）。Android Monitor工具窗口如图 13-73 和图 13-74 所示。

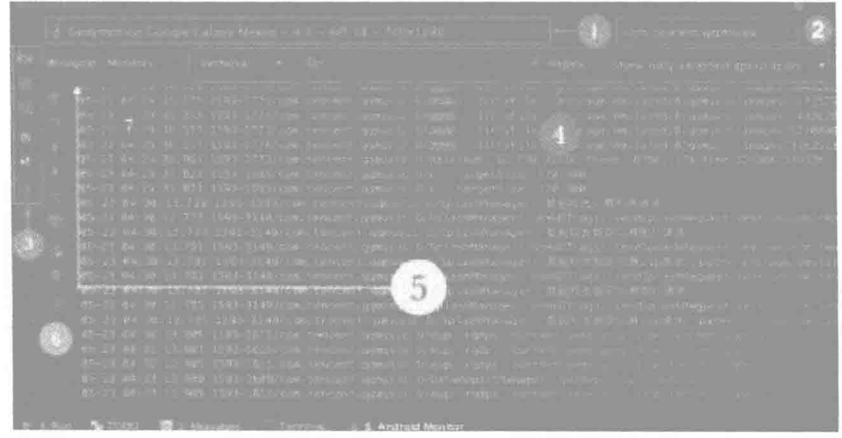


图 13-73

图 13-74

❶ 已连接到电脑上的设备列表，在这里选择需要使用的设备。

❷ 手机上运行的APP进程列表。

❸ 针对手机操作的工具栏，从上到下依次是截图、录像、系统信息、终止应用、布局解析、帮助工具。

❹ Logcat日志过滤工具栏，从左到右依次是日志级别、关键字、是否使用正则匹配、过滤配置。

❺ Logcat日志输出面板。

❻ Logcat日志工具栏。

❼ 内存、CPU、网络、GPU监控。

3. 监控环境准备

使用这些工具前，还需要做一些环境准备。

（1）使用 Logcat 监控

① 手机或模拟器上开启USB调试。

操作步骤：手机→设置 →应用程序→开发者选项→勾选【USB 调试】，如图 13-75 所示。

图 13-75

② 在 manifest 或 build.gradle 文件中设置 debuggable 为 true。（Debug默认为true，Release默认为false。）

操作步骤：Project Structure→app→Build Types→选择构建类型（渠道）→Debuggable设置为true，如图 13-76 所示。

（2）使用 GPU 监控

操作步骤：手机→设置→应用程序→开发者选项→GPU呈现模式分析→勾选【在adb shell dumpsys gfxinfo中】，如图 13-77 所示。

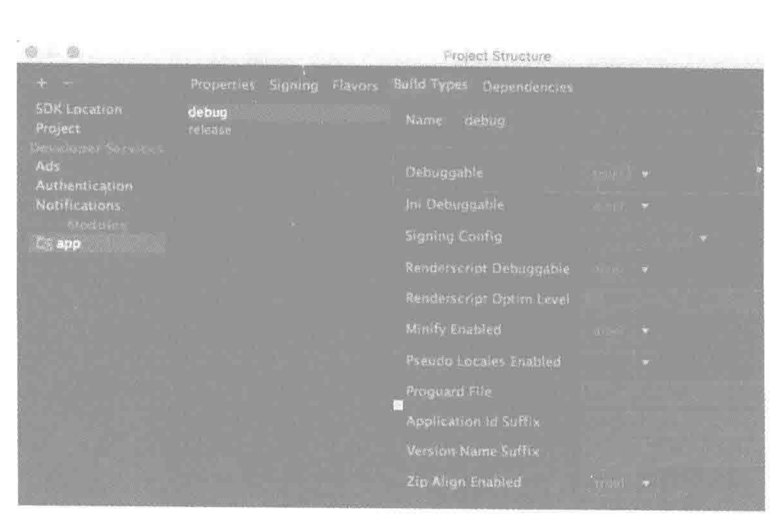


图 13-76

图 13-77

（3）使用网络监控和录像

只能用真机，不支持模拟器。

4. 选择设备和进程

（1）选择设备

设备选择列表中列出了当前已连接到电脑上的设备，并显示设备的相关信息，如系统版本和分辨率，如图 13-78 所示。

图 13-78

设备选择列表还会记录使用过的设备或模拟器信息，并显示设备的状态，如图 13-79 所示。

图 13-79

设备状态提示信息有如下 3 种。

- DISCONNECTED：断开连接，原因是模拟器被关闭或设备的数据线从电脑拔出。
- UNAUTHORIZED：未授权，原因是设备没有授权电脑的连接请求。当手机设备连接到电脑时，手机上会弹出是否允许调试对话框，单击 OK 才能够正常连接。
- OFFLINE：Android Monitor 无法和设备通信。

（2）选择进程

进程选择列表中列出了当前选择的设备上正在运行的APP进程，我们可以在这里选择一个进程进行监控，如图 13-80 所示。

如果进程死掉（进程被终止或APP被卸载），就会显示DEAD状态信息，如图 13-81 所示。

图 13-80

图 13-81

5. 终止进程

如图 13-82 所示，单击终止进程按钮，已选中设备上的那个进程就被终止了。

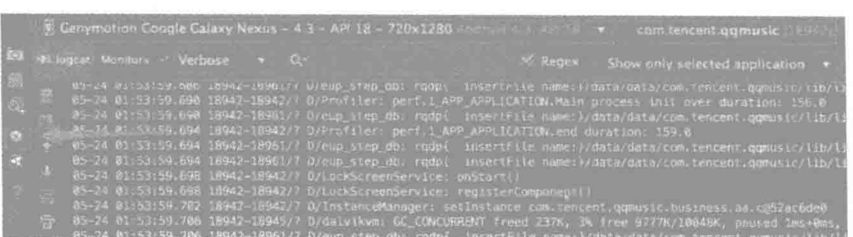


图 13-82

13.9 截 图

截图应该是我们使用最频繁的功能了。关于截图你应该知道：

- Android Studio 支持对真机和模拟器截图。
- 图片格式为 PNG。
- Android Studio 支持对截图做一些修饰。
- 截图是针对真机或模拟器的。

操作步骤：

01 真机连到电脑上，如果要对 APP 截图，需要安装 APP。

02 打开 Android Monitor，单击截图按钮，截图预览窗口。

03 保存截图时单击【Save】按钮，选择存储路径，确定后图片在 Android Studio 中自动打开。

如果想对截图做一些修改或装饰，可以回到截图预览窗口，在这里对截图进行一系列的操作，如重新截图、旋转、给截图加个手机框、给截图加上光效或去掉阴影、显示网格、显示棋格、缩放等，如图 13-83 所示。右击图片会显示图片支持的一些操作，包括使用外部编辑器打开。

图 13-83

13.10 录 像

关于Android Monitor的录像功能你需要知道：

- 视频格式为 MP4。
- 最长可以录制三分钟。
- 仅支持真机，模拟器不能录像。
- 录像是针对手机的。

我们可以通过录像来记录出现BUG时的操作过程和现象，也可以通过录像来制作APP的宣传内容。

操作步骤：

01 将真机连到电脑上，如果要录制 APP 操作过程，就需要安装 APP。

02 打开 Android Monitor，单击录像按钮。

03 弹出录像设置窗口，我们可以在这里设置录像的比特率和分辨率。

- Bit Rate：比特率，默认是 4Mbps。
- Resolution：分辨率，可设置像素的宽度和高度值，该值必须是 16 的倍数，默认是该设备的分辨率。

04 开始录制，如图 13-84 所示。

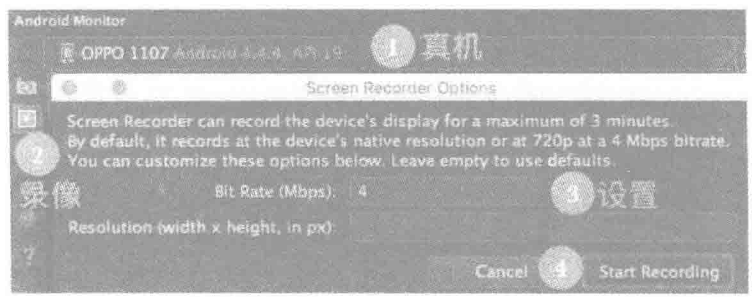


图 13-84

05 录制时会显示时间，如图 13-85 所示。如果操作完毕，单击 Stop Recording 停止录像。

图 13-85

06 保存录像。

13.11 捕获系统信息

通常我们想查看系统信息都是使用adb shell dumpsys命令，如查看某个进程的内存信息：

```
~$ adb -s 设备序列号 shell dumpsys meminfo 进程名
```

例如，查看QQ音乐主进程占用的内存信息：

```
~$ adb -s 192.168.57.101: 5555 shell dumpsys meminfo com.tencent.qqmusic
Applications Memory Usage (kB):
Uptime: 31613080 Realtime: 31613080

** MEMINFO in pid 18942 [com.tencent.qqmusic] **
                    Shared  Private    Heap     Heap     Heap
             Pss     Dirty    Dirty    Size    Alloc     Free
          ------   ------   ------   ------   ------   ------
   Native   3751     1180     3660    14936     9473      166
   Dalvik  22881    11260    21972    26908    26726      182
    Stack    248       12      248
   Cursor      0        0        0
   Ashmem      0        0        0
```

```
  Other dev       13       28        0
   .so mmap     5356     4988     3648
  .jar mmap        0        0        0
  .apk mmap      529        0        0
  .ttf mmap      510        0        0
  .dex mmap     4665        0        8
 Other mmap       67        8        4
    Unknown     1680        0     1680
      TOTAL    39700    17476    31220    41844    36199      348

Objects
             Views:      486        ViewRootImpl:        2
       AppContexts:        6          Activities:        2
            Assets:        2       AssetManagers:        2
     Local Binders:       45       Proxy Binders:       32
  Death Recipients:        9
   OpenSSL Sockets:        0

SQL
        MEMORY_USED:       684
 PAGECACHE_OVERFLOW:       123        MALLOC_SIZE:       62

DATABASES
     pgsz     dbsz   Lookaside(b)          cache  Dbname
        4       24             17         0/16/1
/data/data/com.tencent.qqmusic/databases/recogniz.db
        4       24             22         1/16/2
/data/data/com.tencent.qqmusic/databases/recogniz.db (2)
        4       92             17         0/16/1
/data/data/com.tencent.qqmusic/databases/QQMusic
        4       92            500       30/33/19
/data/data/com.tencent.qqmusic/databases/QQMusic (2)
        4       28            135       51/77/25
/data/data/com.tencent.qqmusic/databases/localalbum/local_dir.db
```

Android Monitor提供了捕获系统信息的工具，可以捕获到dumpsys的输入信息。比如我们使用Android Monitor捕获QQ音乐主进程占用的内存信息，如图 13-86 所示。

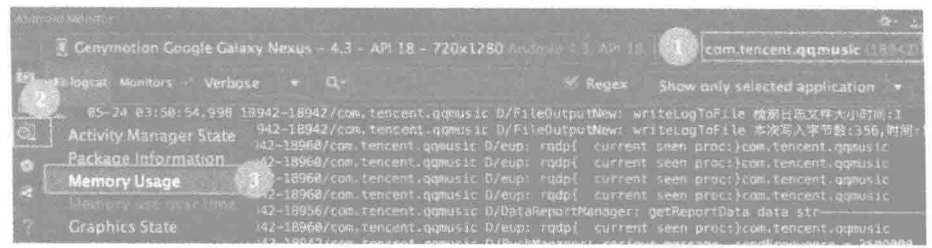


图 13-86

操作步骤：

01 选中 QQ 音乐的主进程（如果不指定进程，将会捕获所有当前正在运行进程的内存信息）。

02 单击【System Information】。

03 单击【Memory Usage】。

然后捕获的内存信息在编辑器中会被自动打开，如图 13-87 所示。

文件以进程名+捕获时间.txt命名，保存在Captures/System Information目录下，如图 13-88 所示。

```
com.tencent.qqmusic_2016.05.24_16.12.txt
Applications Memory Usage (kB):
Uptime: 31501963 Realtime: 31501963

** MEMINFO in pid 18942 [com.tencent.qqmusic] **
                  Shared  Private    Heap    Heap    Heap
            Pss    Dirty    Dirty    Size   Alloc    Free
         ------   ------   ------  ------  ------  ------
   Native  3747     1188     3656   14936    9464     175
   Dalvik 22874    11268    21964   26908   26600     308
    Stack   248       12      248
   Cursor     0        0        0
   Ashmem     0        0        0
Other dev    13       28        0
 .so mmap  5355     4988     3648
.jar mmap     0        0        0
.apk mmap   529        0        0
.ttf mmap   510        0        0
.dex mmap  4656        0        8
Other mmap   67        8        4
  Unknown  1680        0     1680
    TOTAL 39679    17484    31208   41844   36064     483

 Objects
```

图 13-87

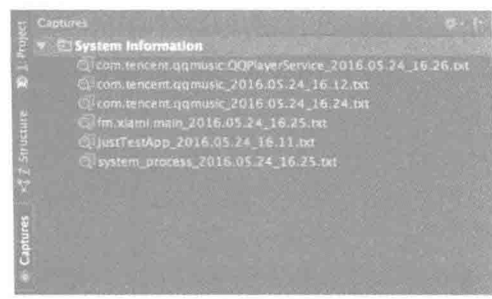


图 13-88

Android Monitor可以捕获的系统信息如表 13-3 所示。

表 13-3

选项	说明	对应的命令
Activity Manager State	获取 Activity 状态信息	dumpsys activity
Package Information	获取安装包的信息	dumpsys package
Memory Usage	获取内存使用信息	dumpsys meminfo
Memory Use Over Time	获取一段时间内的内存使用信息	dumpsys procstats
Graphics State	获取 GPU 信息	dumpsys gfxinfo

13.12 布局解析

Android Studio 2.2 版本在Android Monitor中新增了布局解析工具，可以分析出当前界面的布局结构以及每个组件的属性。这个功能有点像uiautomator，不过比uiautomator解析的属性更详细。

操作步骤：

01 选中手机和 APP 的进程。

02 在手在机上打开要分析的 APP 的界面。

03 单击【Layout Inspector】（见图 13-89）。

图 13-89

然后解析出来的布局信息被保存为.li格式的文件，保存在Captures/Layout Inspector Snapshot目录下（见图 13-90）。该文件会在编辑器中自动打开，如图 13-91 所示。

图 13-90

图 13-91

❶ 界面布局的次层结构，单击某个控件，在预览界面会被相应地圈出来。

❷ 预览界面，也可以在这里可视化地选择控件。

❸ 选中控件的属性列表。

❹ 选中控件属性对应的值。

有了这个布局解析工具，我们可以很方便地查看某个界面的布局结构以及控件的属性和值，可以非常方便地进行布局调试、性能优化、竞品分析、自动化测试等。

13.13 Logcat 监视器

Logcat监视器主要用来对APP进行监控和调试，能够实时显示APP和系统输出的日志信息。当异常发生时会显示相关的堆栈信息和代码链接，我们可以单击链接跟踪到代码。

默认Logcat监控会输出所有日志信息，如果只想看到自己感兴趣的可以使用过滤工具和搜索工具。Android中支持 6 种日志类型。

- Verbose：详细，用于打印所有不重要的、一般的日志信息。
- Debug：调试，用于打印调试用的日志信息。
- Info：信息，用于打印正常使用时需要关注的日志信息。
- Warn：警告，用于打印可能会有问题但还没发生错误的日志信息。
- Error：错误，用于打印运行时出现的严重错误的日志信息。
- Assert：断言，用于打印运行时出现的致命错误的日志信息。

对应代码中的使用：

```
Log.v (TAG, "详细，用于打印所有的不重要的、一般的日志信息.") ;
Log.d (TAG, "调试，用于打印调试用的日志信息.") ;
Log.i (TAG, "信息，用于打印正常使用时需要关注的日志信息.") ;
Log.w (TAG, "警告，用于打印可能会有问题，但还没发生错误的日志信息.") ;
Log.e (TAG, "错误，用于打印运行时出现的严重错误的日志信息.") ;
Log.wtf (TAG, "断言，用于打印运行时出现的致命错误的日志信息.") ;
```

提 示

在准备发布的版本中不能够出现 Verbose 和 Debug 类型的日志，其他类型的信息打印时需要注意敏感信息。

Logcat监视器中显示的结果如图 13-92 所示。

```
05-24 23:59:11.118 4154-4154/com.wirelessqa.basetoolsample V/MainActivity: 详细,用于打印所有的不重要的、一般的日志信息.
05-24 23:59:11.118 4154-4154/com.wirelessqa.basetoolsample D/MainActivity: 调试,用于打印调试用的日志信息.
05-24 23:59:11.118 4154-4154/com.wirelessqa.basetoolsample I/MainActivity: 信息,用于打印正常使用时需要关注的日志信息.
05-24 23:59:11.122 4154-4154/com.wirelessqa.basetoolsample W/MainActivity: 警告,用于打印可能会有问题,但还没发生错误的日志信息.
05-24 23:59:11.122 4154-4154/com.wirelessqa.basetoolsample E/MainActivity: 错误,用于打印运行时出现的严重错误的日志信息.
05-24 23:59:11.122 4154-4154/com.wirelessqa.basetoolsample A/MainActivity: 断言,用于打印运行时出现的致命错误的日志信息.
```

图 13-92

（1）日志没有显示

在工具栏单击【Restart】重新启动日志，如图 13-93 所示。

（2）设置日志头信息的显示

在工具栏单击【Logcat Header】设置日志头信息的显示内容，如图 13-94 所示。

图 13-93

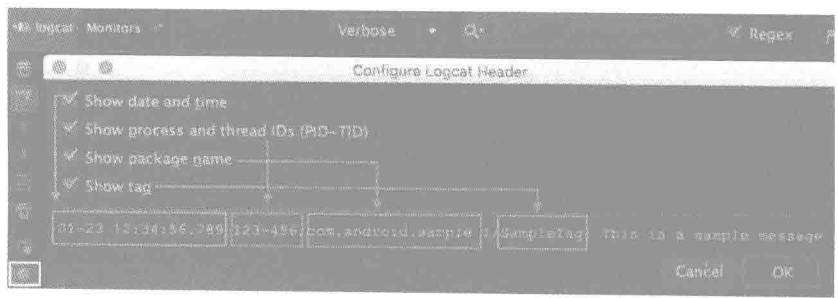


图 13-94

（3）过滤日志（见图 13-95）

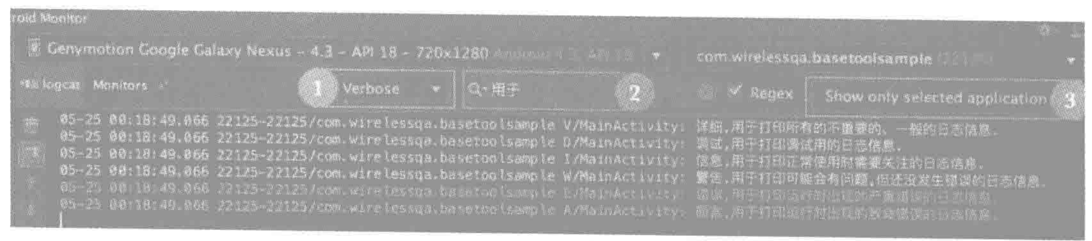


图 13-95

❶ 过滤日志类型，输出级别为Verbose > Debug > Info> Warn > Error > Assert。默认显示Verbose级别的日志，也就是说会打印所有输出的日志信息，包括Verbose、Debug、Info、Warn、Error、Assert。

如果选择Debug级别，Verbose类型的信息就不会被显示，只会显示Debug、Info、Warn、Error、Assert，依次类推。

❷ 过滤输出的日志信息，支持正则匹配。

❸ 过滤器，如图 13-96 所示。

- Show only selected application：仅显示当前选中的 APP 的日志信息。
- Firebase：仅显示 Firebase 的日志信息。
- No Filters：显示设备上所有的日志信息。
- Edit Filter Configuration：编辑过滤器配置。

创建一个新的Logcat过滤器（见图 13-97）。

- Filter Name：过滤器的名字。
- Log Tag：设置过滤 Tag 参数，支持正则匹配。
- Log Message：设置过滤日志信息参数，支持正则匹配。
- Package Name：设置过滤包名参数，支持正则匹配。
- PID：设置过滤的进程。
- Log Level：设置过滤的日志类型。

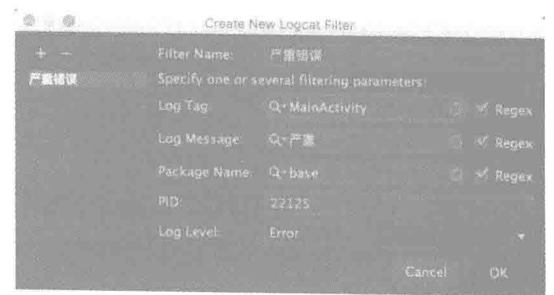


图 13-97

过滤参数可以设置一个或多个，确定后过滤器设置成功。设置好过滤器后就可以在过滤器列表中进行选择了，如图 13-98 所示。

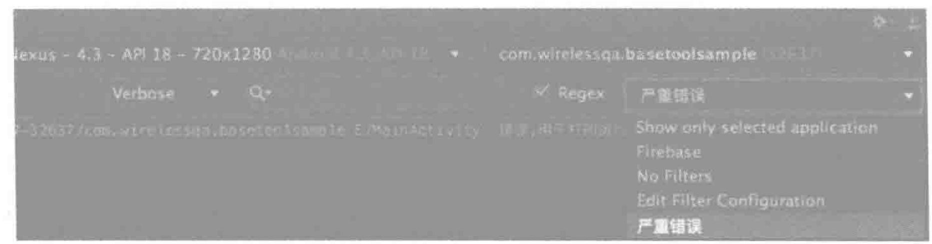


图 13-98

（4）搜索日志

按快捷键command + F（macOS）或者Ctrl + F（Windows/Linux）调出搜索工具栏，如图 13-99 所示。

图 13-99

区分大小写需要勾选【Match Case】，匹配完整的词需要勾选【Words】，如图 13-100 所示。

图 13-100

上面的例子是我们明确知道要搜索的关键字是ActivityManager，因此可以使用精确匹配。如果不知道ActivityManager的大小写，并想匹配结果就要取消勾选【Match case】，否则如图 13-101 所示。

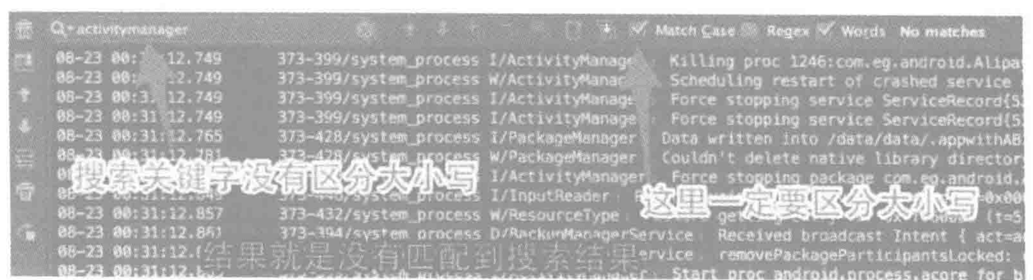


图 13-101

提示

Regex（正则匹配）跟 Words 不能同时使用。
例如，搜索一个模糊的关键词，不区分大小写，如图 13-102 所示。

图 13-102

匹配多行的效果如图 13-103 所示。

图 13-103

在搜索结果中跳转和选择的操作按钮如图 13-104 所示。

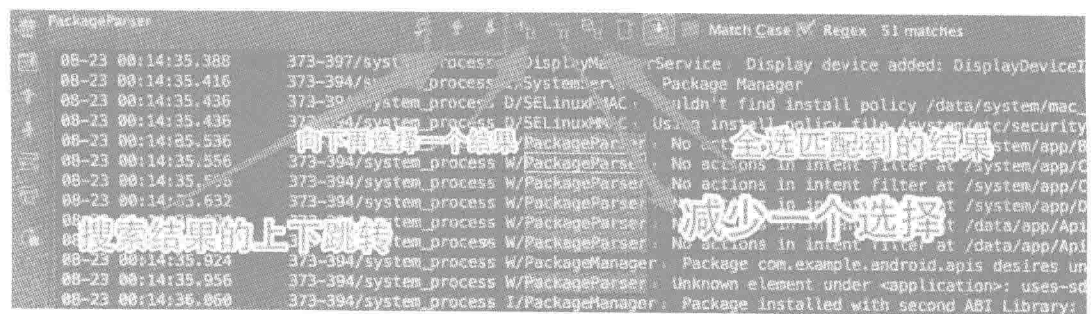


图 13-104

搜索工具会记录搜索历史，可以通过选择复用这些历史记录，如图 13-105 所示。

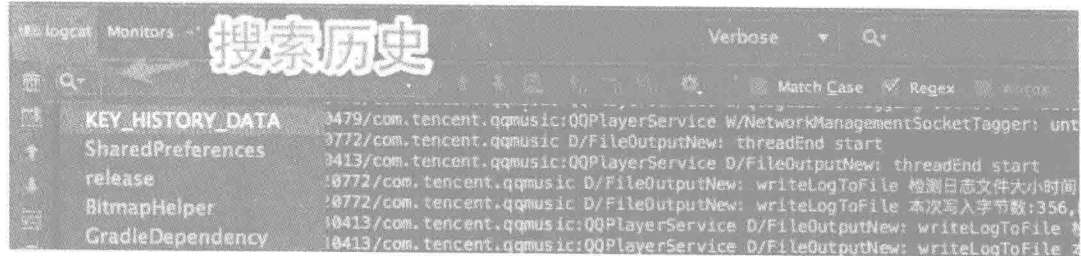


图 13-105

（5）自定义Logcat输出颜色

默认Logcat输出的颜色非常单调，如图 13-106 所示。

```
05-24 23:59:11.118 4154-4154/com.wirelessqa.basetoolsample V/MainActivity: 详细,用于打印所有的不重要的、一般的日志信息.
05-24 23:59:11.118 4154-4154/com.wirelessqa.basetoolsample D/MainActivity: 调试,用于打印调试用的日志信息.
05-24 23:59:11.118 4154-4154/com.wirelessqa.basetoolsample I/MainActivity: 信息,用于打印正常使用时需要关注的日志信息.
05-24 23:59:11.122 4154-4154/com.wirelessqa.basetoolsample W/MainActivity: 警告,用于打印可能会有问题,但还没发生错误的日志信息.
05-24 23:59:11.122 4154-4154/com.wirelessqa.basetoolsample E/MainActivity: 错误,用于打印运行时出现的严重错误的日志信息.
05-24 23:59:11.122 4154-4154/com.wirelessqa.basetoolsample A/MainActivity: 断言,用于打印运行时出现的致命错误的日志信息.
```

图 13-106

不同级别的日志应该用不同的颜色来区分才好，我们可以自定义Logcat的颜色。

第 1 步： 偏好设置→Editor→Colors & Fonts→Android Logcat，如图 13-107 所示。

图 13-107

第 2 步： 设置颜色，如图 13-108 所示。

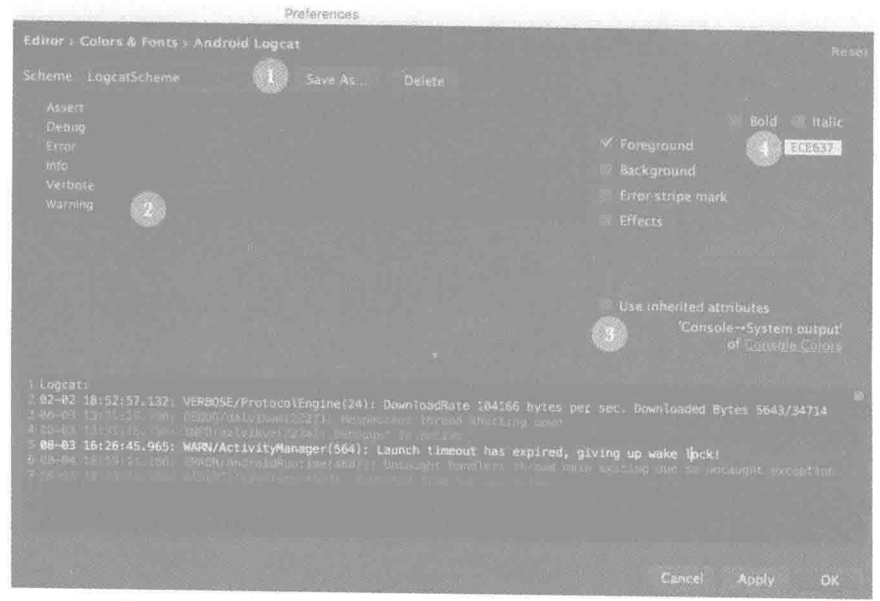


图 13-108

01 新建一个 Scheme。
02 选择要设置的日志类型。
03 取消勾选【Use inherited attributes】。
04 设置颜色。

效果如图 13-109 所示。

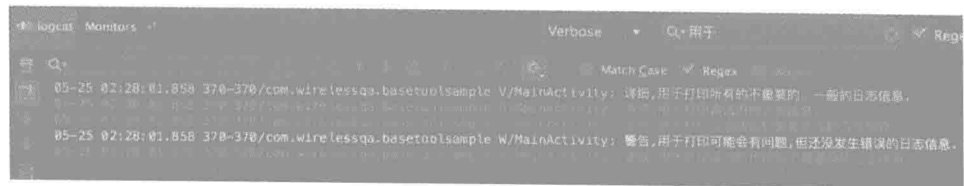


图 13-109

13.14 内存监视器

使用Android Monitor提供的内存监视器可以实时观察应用程序的内存使用情况，它显示了应用程序可用的和已分配的内存信息。内存监视器的界面如图 13-110 所示。

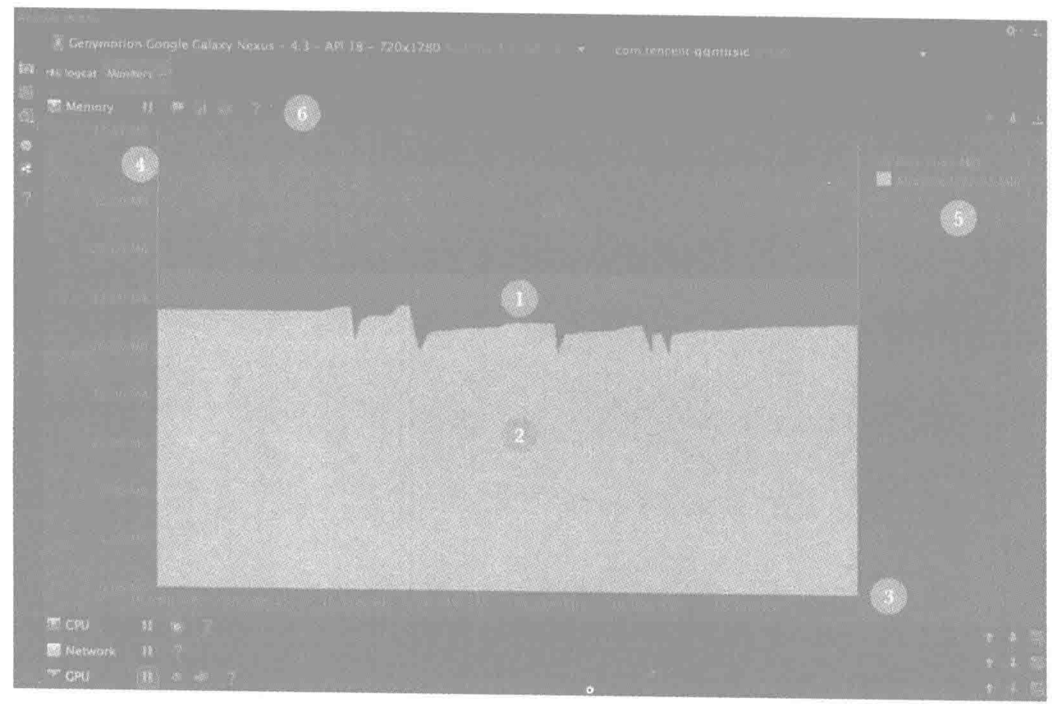


图 13-110

❶ 空闲内存。
❷ 已分配内存。
❸ 时间轴，随着时间的推移，我们可以观察到一段时间内应用程序内存的使用情况。

❹ 内存值。
❺ 当前内存的使用情况（空闲和已分配内存）。
❻ 工具栏提供的工具，如图 13-111 所示。

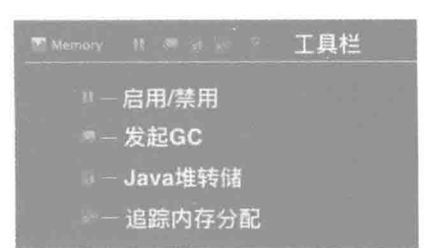


图 13-111

要想查看应用程序的内存，首先要选择设备和应用程序的进程，其次要确定内存监视器是启用状态。

禁用内存监视器时如图 13-112 所示，正常启用内存监视器后才能够看到内存的变化（见图 13-113）。

图 13-112

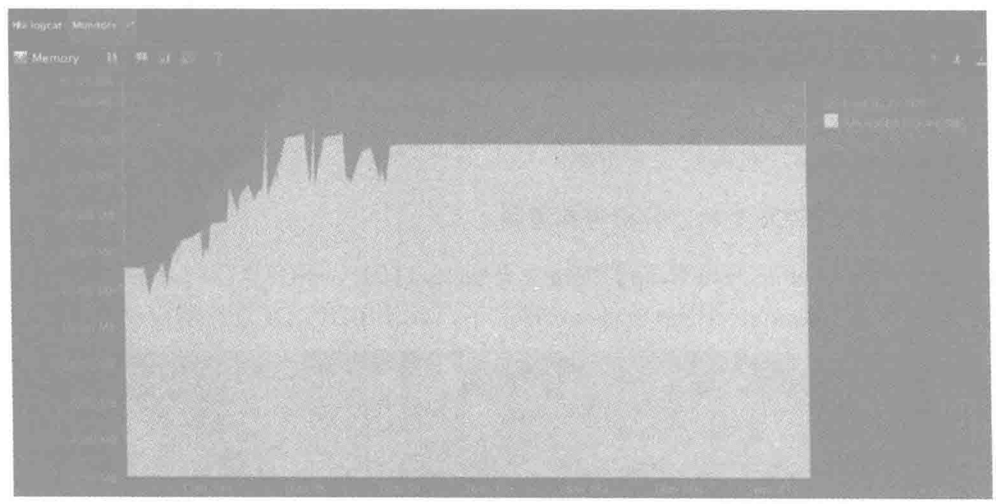


图 13-113

我们可以一边操作APP一边观察内存的变化，如果发现操作APP的过程中突然出现一个意料之外内存峰值或内存GC后还在不断增长，那就有可能会发生OOM或内存泄漏，如图 13-114 所示。

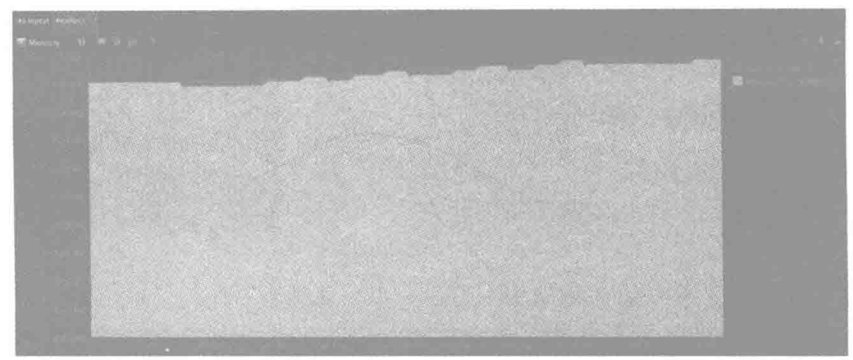


图 13-114

我们可以主动发起GC（垃圾回收），清除无用或不再被其他对象引用的那些对象所占用的内存空间，如图 13-115 所示。

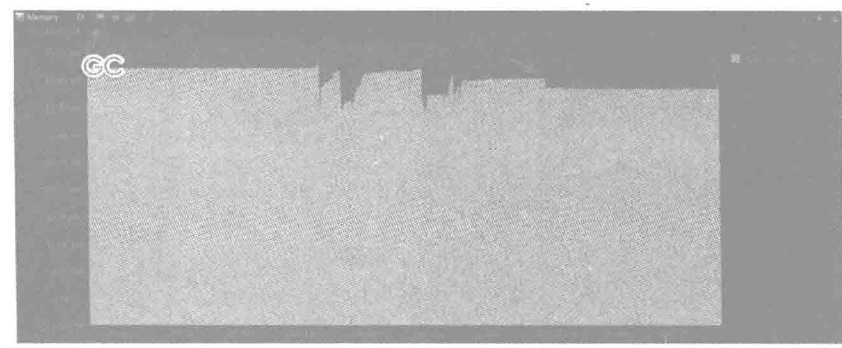


图 13-115

GC后内存应该稳定在一个值附近才对，如果还在不断增长，那一定是内存泄漏了。

13.14.1 Dump Java Heap

如果发现内存泄漏，应该怎么分析呢？

方法一：使用HPROF查看器分析内存泄漏

第 1 步：单击【Dump Java Heap】按钮（见图 13-116），得到当前内存的快照（HPROF）文件。文件被存放在Captures/Heap Snapshot目录下，并使用HPROF查看器自动打开。

图 13-116

第 2 步：使用HPROF查看器分析内存泄漏。

HPROF查看器显示了APP的内存分配情况，包括APP为哪些类分配了内存空间、类的对象个数以及占用的内存大小（见图 13-117）。我们可以通过这些信息确认是否发生了内存泄漏。

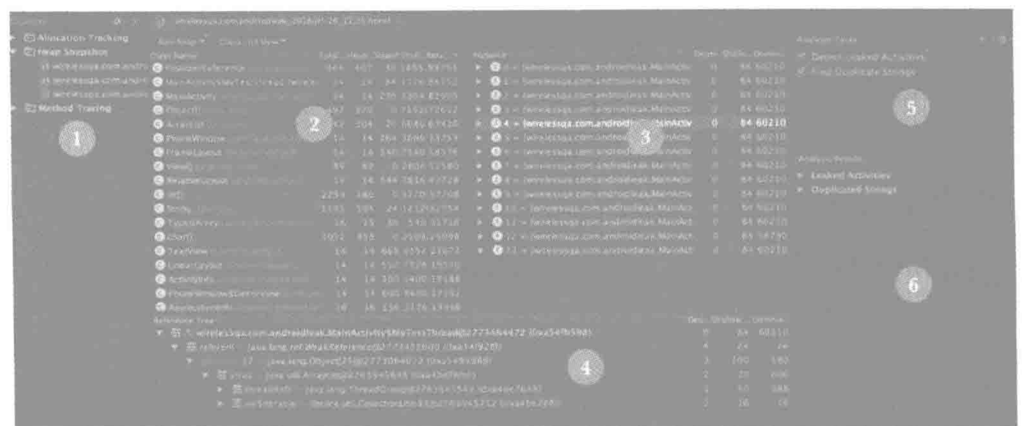


图 13-117

❶ Captures/Heap Snapshot：用于存放HPROF文件。

❷ 显示APP占用内存空间的类和类的属性（见图 13-118），详细说明如表 13-4 所示。

wirelessqa.com.androidleak_2016.05.26_12.35.hprof

App heap ▾ Class List View ▾

❶ Class Name	❷ Total Count ▾	❸ Heap Count	❹ Sizeof	❺ Shallow Size	❻ Retained Size
PhoneWindow$PanelFeatureState[]	14	14	0	56	1624
PhoneWindow$PanelFeatureState	14	14	112	1568	1568
PhoneWindow$DialogMenuCallback	14	14	20	280	280
PhoneWindow$DecorView	14	14	600	8400	17192
PhoneWindow$1	14	14	12	168	168
PhoneWindow	14	14	264	3696	137536
MainActivity$MyTestThread (wirelessqa.com.android	14	14	84	1176	841520
MainActivity	14	14	236	3304	839056
long[]	14	4	0	352	352

13-118

表 13-4

序号	属性	说明
1	Class Name	内存中的类名
2	Total Count	内存中该类的对象个数
3	Heap Count	选中的堆内存中该类的对象个数
4	Sizeof	该对象的大小
5	Shallow Size	选中的堆内存中所有对象浅堆大小总和（浅堆是指一个对象直接占用的内存大小，不包含对其他对象的引用），Shallow Size = Sizeof * Heap Count
6	Retained Size	选中的堆内存中所有对象保留堆大小总和（保留堆是指一个对象自己占用的浅堆加上从该对象能直接或间接访问到对象的浅堆之和），Retained Size 等于所有 Dominating Size 相加

右击类名，单击弹出菜单中的【Jump to Source】，可以跳转到该类的源码，如图 13-119 所示。

图 13-119

另外，我们还可以切换查看模式，Package Tree View以包名来分类，看起来更加清晰，如图 13-120 所示。

图 13-120

❸ 显示选中类的所有实例和属性（见图 13-121），详细说明如表 13-5 所示。

图 13-121

表 13-5

序号	属性	说明
1	Instance	类的具体对象
2	Depth	从 GC root 到选中对象最短的跳跃数
3	Shallow Size	该对象的浅堆大小
4	Dominating Size	该对象的保留堆大小

❹ Reference Tree：指向所选对象的引用，以及引用的引用，其他属性同上，如图 13-122 所示。

Reference Tree	Depth	Shallow Size	Dominating Size
wirelessqa.com.androidleak.MainActivity$MyTestThread@2773134288 (0xa54aabd0)	0	84	60210
referent java.lang.ref.WeakReference@2773134504 (0xa54aaca8)	4	24	24
5 java.lang.Object[25]@2773064072 (0xa5499988)	3	100	580
array java.util.ArrayList@2763945848 (0xa4be76b0)	2	20	600
threadRefs java.lang.ThreadGroup@2763945544 (0xa4be7648)	1	40	688
group java.lang.Thread@2773066824 (0xa549a448)	0	80	172
group com.android.tools.fd.runtime.Server$SocketServerThread@2773067624 (0xa549a768)	0	84	160
group wirelessqa.com.androidleak.MainActivity$MyTestThread@2773287264 (0xa54d0160)	0	84	60210
group wirelessqa.com.androidleak.MainActivity$MyTestThread@2774099616 (0xa55966a0)	0	84	60210
group wirelessqa.com.androidleak.MainActivity$MyTestThread@2773316464 (0xa54d7370)	0	84	60210
mMain java.lang.ThreadGroup	0	172	220
group wirelessqa.com.androidleak.MainActivity$MyTestThread@2773909432 (0xa5567fb8)	0	84	60210

图 13-122

❺ Analyzer Tasks：分析任务，如图 13-123 所示。使用这个功能可能分析出发生内存泄漏的Activity和重复的字符串。

- Detect Leaked Activities：检测内存泄漏的Activity。
- Find Duplicate String：找出重复的字符串。

单击右上角的开始按钮，就开始执行分析任务了，分析结果显示在Analysis Result中。

❻ Analysis Results：分析结果，如图 13-124 所示。

图 13-123

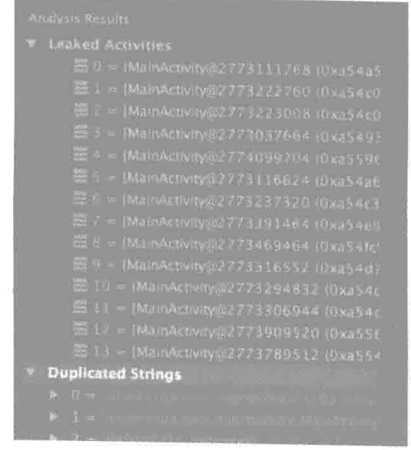


图 13-124

自动分析内存泄漏的功能非常方便。

方法二：使用MAT分析内存泄漏

如果还是习惯于使用MAT分析，也可以导出标准的hprof文件。

操作步骤：

01 Captures/Heap Snapshot → 右击.hprof 文件 → Export to standard .hprof，如图 13-125 所示。

02 选择文件的存储路径，如图 13-126 所示。

图 13-125

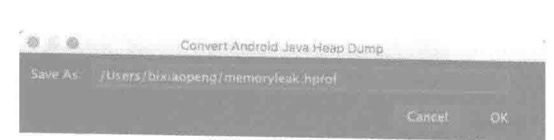


图 13-126

03 单击【OK】按钮后文件保存成功。

接下来就可以直接使用MAT打开这个文件进行分析了。

这里Android Studio帮我们将android hprof文件转成了标准的java hporf文件，所以才可以直接使用MAT打开来分析。如果是用DDMS导出的hprof文件，就需要自己使用命令转换。

13.14.2 Allocation Tracking

Allocation Tracking（内存分配追踪器）用来监视内存分配更详细的信息。我们可以监视在执行某些操作一段时间内的内存分配情况。知道这些分配，可以更加有针对性地优化相关方法。

使用Allocation Tracking：

- 可以查看何时、何处分配了对象，以及对象的大小、分配的线程和堆栈跟踪信息。
- 可以通过分配/释放模式复现内存抖动的情况。
- 可以配合 HPROF 查看器一起使用，追踪内存泄漏的问题。例如，当我们使用 HPROF 查看器发现一个 bitmap 驻留在堆中的时候，可以使用 Allocation Tracking 找到它的分配位置，如图 13-127 所示。

图 13-127

单击【Start Allocation Tracking】开始追踪内存分配，这时我们对APP进行一些操作，然后单击【Stop Allocation Tracking】，如图 13-128 所示。

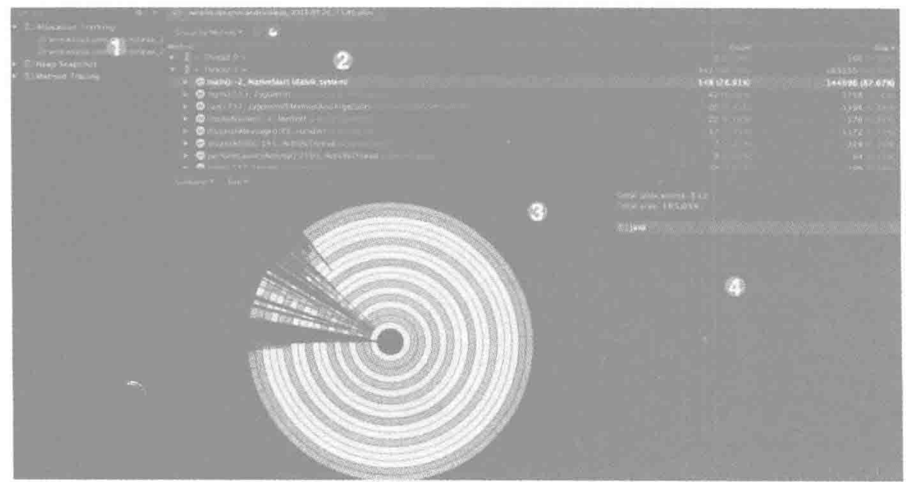


图 13-128

❶ 停止追踪后.alloc文件被保存在Capture/Allocation Tracting目录下，并且被自动打开。

❷ 显示内存分配信息。默认以方法来分类内存分配信息，如图 13-129 所示。还可以切换成以分配器来分类内存分配信息（见图 13-130 所示），详细说明如表 13-6 所示。

图 13-129

图 13-130

表 13-6

属性	说明
Method	负责内存分配的方法
Count	分配对象的个数
Size	分配的内存总大小（Byte）

单击这些属性可以进行相应的排序，我们可以通过内存分配的次数和大小来找出那些占用过多的内存、引发过多的GC且又非常耗时的方法。

❸ 内存分配的图表。通过这个漂亮的图表，我们可以看出哪些方法或类拥有最多的对象。图表的数据可以选择Count和Size（见图 13-131 所示），类型可以选择Sunburst和Layout（见图 13-132）。

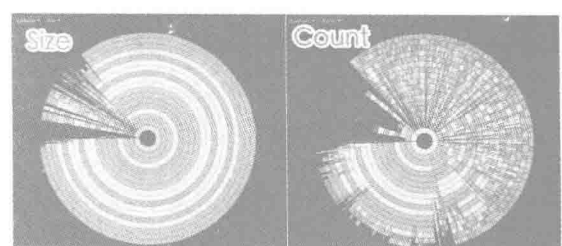


图 13-131

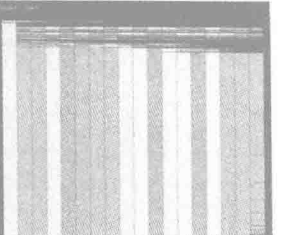


图 13-132

❹ 显示图标指定的线程或方法的分配次数和内存大小，如图 13-133 所示。

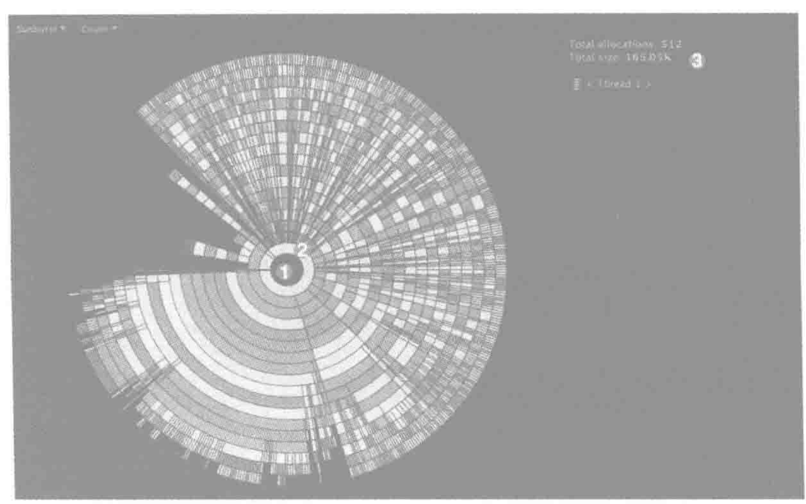


图 13-133

① 将光标放在中心点会显示图表中占比最大的线程信息，如线程名、分配次数和内存大小。
② 这个圆圈是所有线程的占比，一些占比较小的线程双击后可展开。
③ 显示线程名/方法名、分配次数和内存大小。

13.15 CPU 监视器

CPU监视器（见图 13-134）实时显示用户和内核模式下CPU使用率的总和，可以让我们很轻松地查看操作APP时的CPU使用率。

图 13-134

1. 模式

在用户（User）模式下，代码必须通过系统API访问硬件或内存，并且只能访问虚拟内存地址，崩溃通常是可回收的。在内核（Kernel）模式下，代码可以直接访问硬件，包括物理内存地址，崩溃会导致设备停止。

2. 方法追踪

Method Tracking（方法追踪）用来追踪一段时间内系统的运行情况，可以查看APP中的调用堆栈和时间信息。

操作步骤：

01 单击左上角的 Start Method Tracking。

02 操作 APP 一段时间后再单击一次。

03 trace 文件被保存在 Captures/Method Tracing 目录下，并且自动打开，如图 13-135 所示。

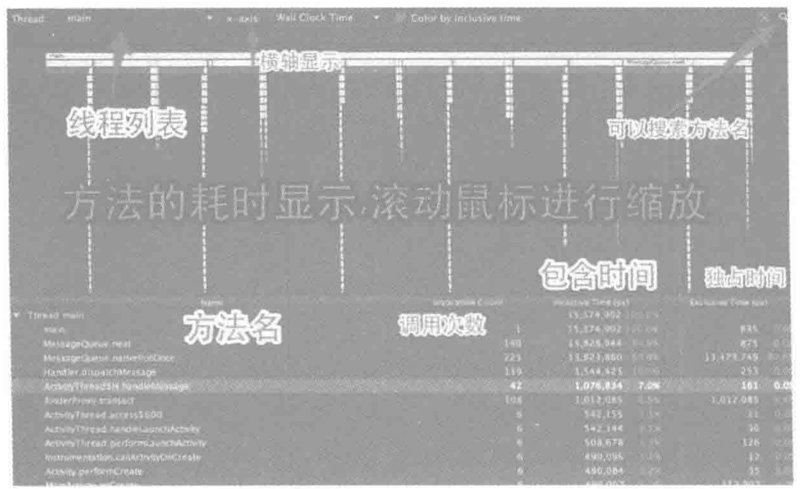


图 13-135

- Inclusive Time：包含时间（ms），方法占用的 CPU 时间，包含内部调用其他方法占用的 CPU 时间。
- Exclusive Time：独占时间（ms），方法占用的 CPU 时间，但不含内部调用其他方法所占用的 CPU 时间。

常用的判断方法：

（1）方法调用次数不多，但每次调用却需要花费很长时间，可能会有问题。

（2）自身占用时间不长，但调用却非常频繁的方法也可能会有问题。

搜索方法和显示方法详情的界面如图 13-136 所示。

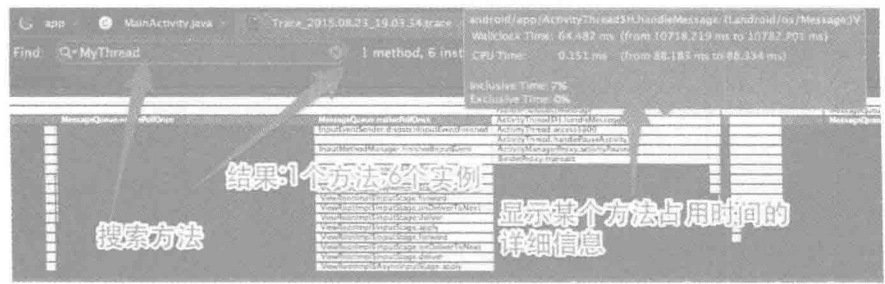


图 13-136

这个工具跟traceview非常像，如果你对traceview比较熟悉，对这个工具应该也不会感到陌生。

13.16 网络监视器

网络监视器用来查看APP的网络请求情况，可以实时监控到APP的数据传输，使用它可以帮助我们找到一些不必要的或不规范的网络请求。

网络监视器界面如图 13-137 所示。

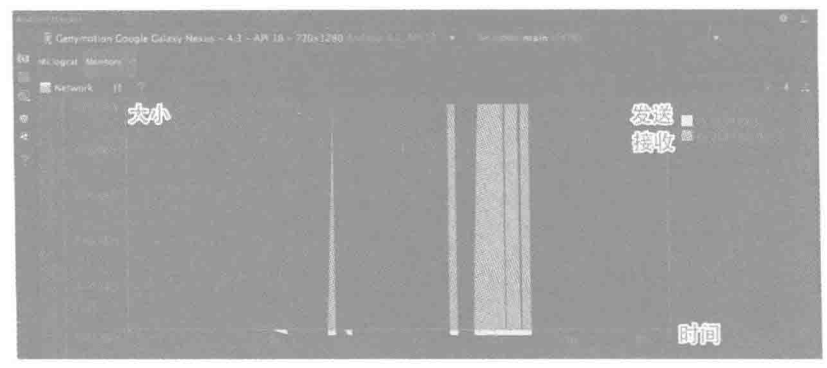


图 13-137

流量测试是专项测试中非常重要的一个测试项，我们需要保证APP不会有不必要的数据加载（偷跑流量）、不会有过大的数据加载（在运营商网络下，用户会非常关心）等。使用网络监视器做流量测试会很方便。

13.17 GPU 监视器

GPU监视器用来查看UI界面渲染帧需要花多少时间，可以实时查看每一帧渲染所花费的时间。

GPU监视器能帮我们快速查看UI界面的渲染执行、确认渲染路径是否花了较长的处理时间、寻找帧渲染的时间峰值与用户或程序操作相关联。要想使用CPU监视器，首先要打开GPU呈现模式分析，操作步骤为：手机→设置→应用程序→开发者选项→GPU呈现模式分析→勾选【在adb shell dumpsys gfxinfo中】→选中需要测试的APP进程，开启GPU监视器。

Android 5.0 之前，显示渲染帧的过程有Execute、Process、Draw，如图 13-138 所示。

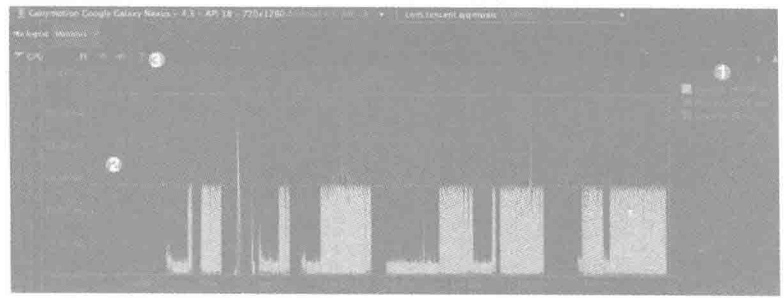


图 13-138

Android 5.0 之后新增了一个Prepare，如图 13-139 所示。

图 13-139

帧渲染的过程分成了四个不同的阶段。

- 蓝色：Draw（绘制），表示创建/刷新 DisplayList（包含需要 GPU 绘制到屏幕上的数据信息）所花费的时间。
- 紫色：Prepare（准备），表示 UI 线程将所有资源传递给渲染线程所花费的时间。
- 橙色：Process（过程），表示执行 DisplayList 并创建 OpenGL 命令所花费的时间。
- 黄色：Execute（执行），表示 CPU 发送 OpenGL 命令给 GPU，等待 GPU 渲染完成的时间。

Android 6.0 之后又更加细化了整个渲染过程，如图 13-140 所示。

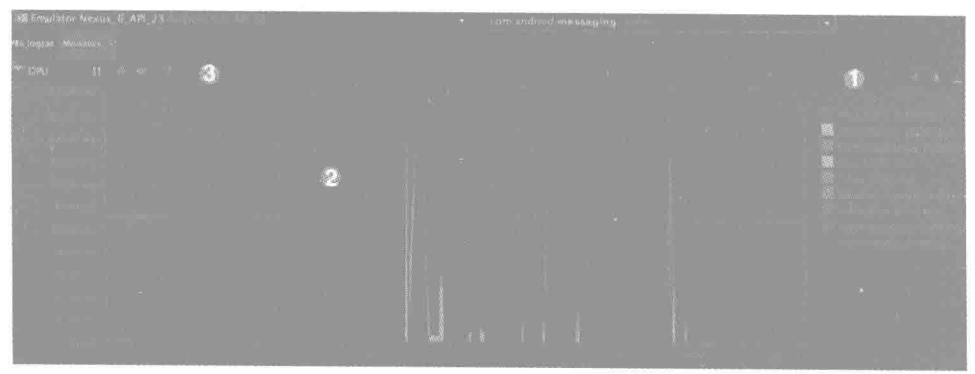


图 13-140

GPU监视器的组成包括渲染帧的不同阶段、显示渲染帧耗时曲线图。绿色的线表示 60FPS，即每一帧渲染时间大概是 16ms。红色的线表示 30FPS，即每一帧渲染时间大概是 33ms。通常在测试界面是否卡顿的时候需要关注帧率是否小于 60FPS，GPU监视器这条绿线正好给我们设置了参考标准，看起来非常直观。

13.18 APK 分析器

Android Studio 2.2 版本新增的APK分析器能反编译APK，可以帮助我们非常方便地分析APK当中的内容。我们可以用它来分析APK当中的混淆文件，一方面可以参考别人家的APK

是如何配置混淆文件的，另一方面也可以排查自己家的APK是否有相关的配置问题。我们还可以用它来查看已编译的资源文件，找出一些较大的文件进行具体分析，确认是否需要保留、减小包大小。右上角的[Compare with…]功能可以对比两个APK文件，找出差异性，方便分析不同版本APK之间的区别。

使用APK分析器来分析APK

操作步骤：菜单栏→Build→Analyze APK→选择要分析的APK→OK，或者APK本身就在项目当中，双击APK就可以打开分析结果，如图 13-141 所示。

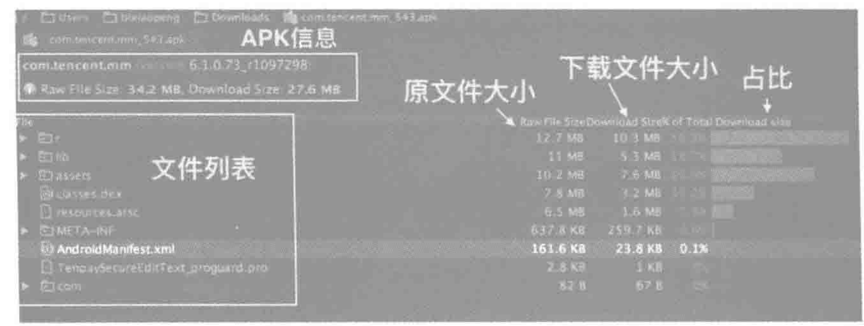


图 13-141

图 13-141 是微信的分析结果，下面就以微信APK为例来进行介绍。

- com.tencent.mm：微信的包名。
- version 6.1.0.73_r1097298：版本号。
- Raw File Size：34.2 MB ：分析出来的 APK 文件总大小。
- Download Size：27.6 MB：下载下来的 APK 的总大小。
- File：分析出来的文件列表。
- Raw File Size：分析出来的文件对应的原始大小（未压缩过的）。
- Download Size：分析出来的文件对应的压缩过的大小。
- % of Total Download size：分析出来的文件对应的压缩过的大小/下载下来的 APK 的总大小。

在APK分析结果界面，按文件大小从高到低排序，可以很清晰地看出哪些是大文件，然后查看大文件的内容。如果想减少包大小，需要先从大文件入手。另外，我们可以查看合并后的AndroidManifest.xml是否正确，还可以查看混淆文件，看看微信是如何进行混淆配置的。

13.19 主题编辑器

主题编辑器支持实时预览主题效果，支持横竖屏、UI模式、虚拟机、Android版本和多语言预览。

打开主题编辑器（见图 13-142）的操作步骤是菜单栏→Tools→Android→Theme Editor或者打开res/values/styles.xml→单击右上角Open editor。

图 13-142

主题属性对应的区域如图 13-143 所示。

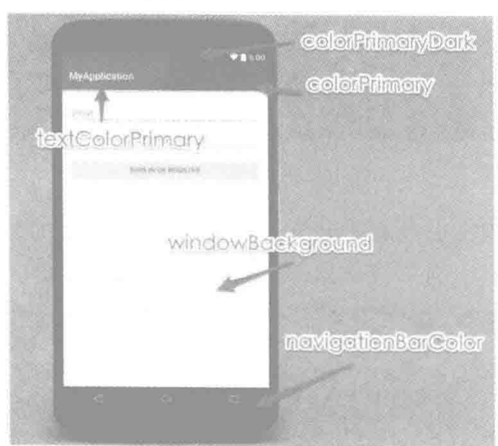


图 13-143

查看style.xml：

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

在主题编辑器中编辑后，这里的值会被同步修改。

修改主题

例如修改colorPrimaryDark的颜色，即可以直接使用colors.xml中定义的颜色（见图 13-144），也可以自定义颜色（见图 13-145）。

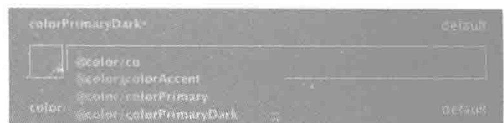


图 13-144

图 13-145

预览窗口对应区域的颜色也会随之改变，如果觉得不合适还可以再切换颜色，如图 13-146 所示。切换到布局编辑器中，预览改变后的主题，如图 13-147 所示。

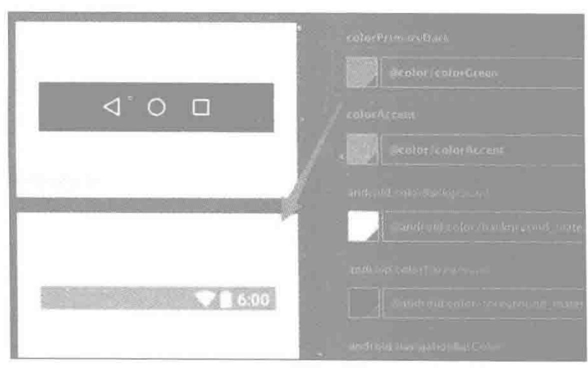


图 13-146

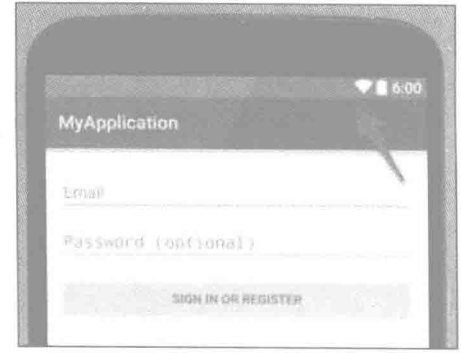


图 13-147

预览主题的兼容性

工具栏上提供了多个兼容性预览工具，这里仅以Android API版本举例演示。我们即可以在主题编辑器中只切换某个属性的API版本查看效果（见图 13-148），也可以在工具上切换API版本查看所有属性的显示效果（见图 13-149）。

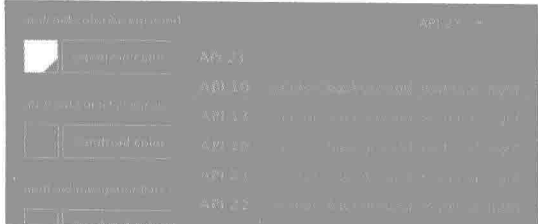


图 13-148

图 13-149

例如，切换到API 14，显示效果如图 13-150 所示。如果有兼容性问题，会有警告图标提示。

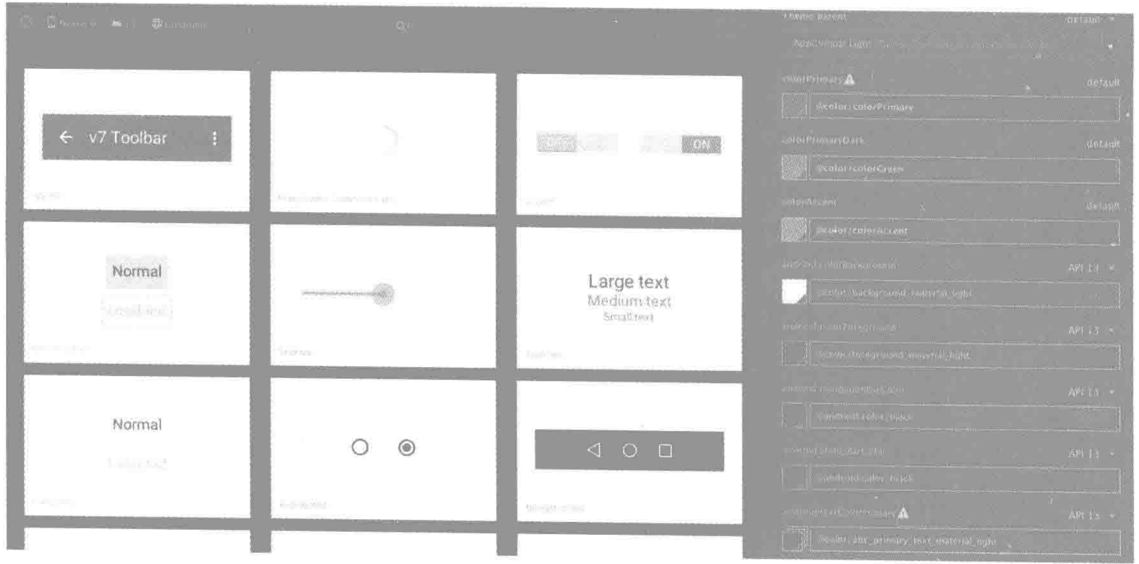


图 13-150

第 14 章　版本控制

版本控制是项目开发过程中必不可少的部分，不管是个人还是团队，灵活地使用版本控制将会使项目开发变得更加轻松。Android Studio天生集成了版本控制系统，目前支持CVS、SVN、Git、Mercurial等主流的版本控制系统。在Android Studio中使用版本控制系统一定要先确保电脑上已经配置好了相应的工作环境。

本章我们将会重点介绍目前最流行的版本控制系统Git的配置和使用。

本章重要知识点 >>>>>>>>>>

- 如何配置 Git 和 GitHub;
- 如何使用 Git 进行版本控制。

14.1　版本控制系统

假设我们打开的是一个没有使用版本控制的项目。首先启用Git作为项目的版本控制系统：菜单栏→VCS→Enable Version Control Integration→选择Git，如图 14-1 所示。然后项目根目录下就多了一个.git，项目中的文件都是红色的，表示未加入Git版本跟踪。

图 14-1

开启版本控制集成后，我们就可以看到Android Studio中的版本控制系统都有哪些操作入口了。

1. 菜单栏→VCS

查看菜单栏上的VCS，可以看到版本控制系统的所有功能，如图 14-2 所示。

2. 右键→菜单

右击要操作的文件或文件夹，在弹出的菜单中可以看到版本控制系统的选项，如图 14-3 所示。本例是Git，如果使用的是svn，这里会显示Subversion。

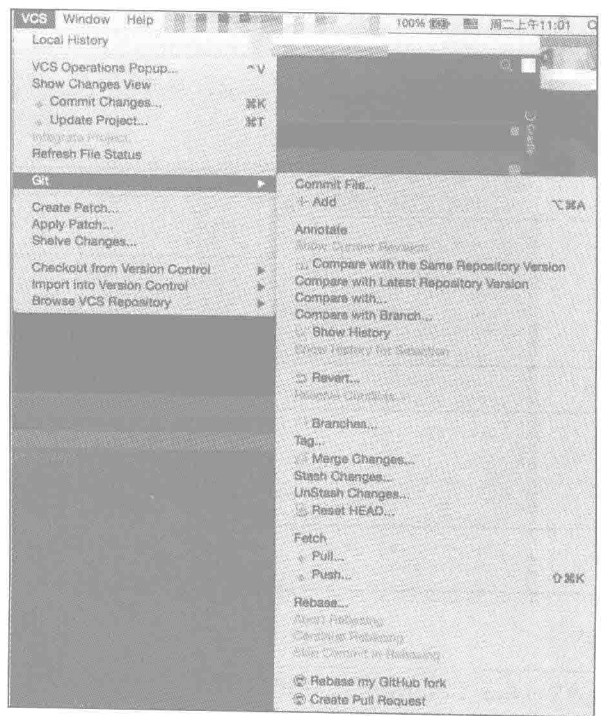


图 14-2

图 14-3

右键菜单中提供的功能只包含对应的版本控制系统特有的功能，没有包含通用的功能，这是菜单栏的区别。另外，从图 14-3 中可以看出，这里提供的Git操作区分了本地操作和远程操作，操作分类更加明确。

3. 底部工具栏→Version Control

底部工具栏会显示Version Control，如果没有显示，请单击View→Tool Windows来显示，如图 14-4 所示。

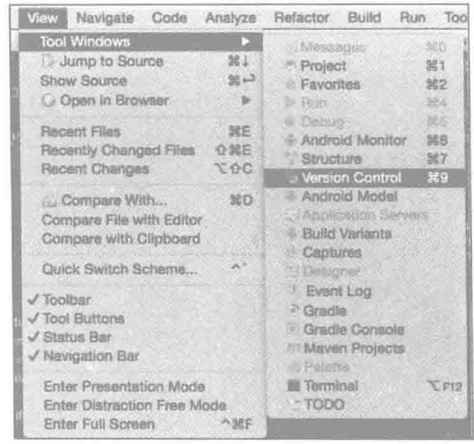


图 14-4

打开底部工具栏上的Version Control会打开操作面板，如图 14-5 所示。

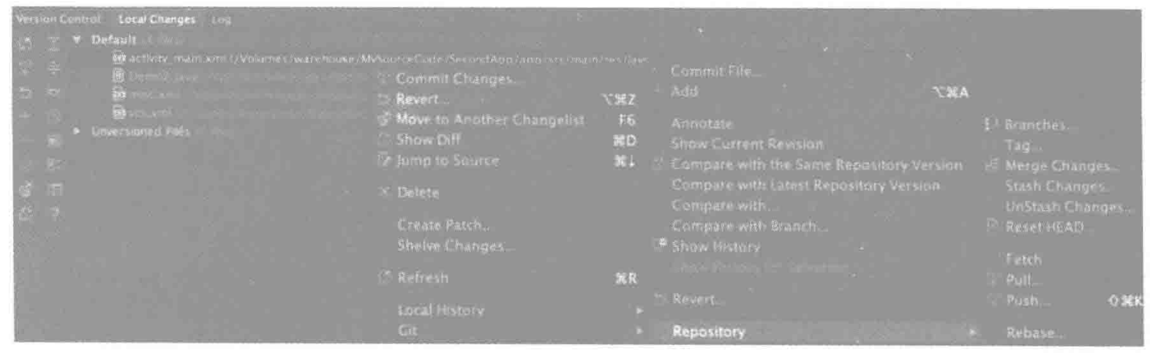


图 14-5

这里很多功能跟菜单栏VCS中的相同，但是这里的操作更加有针对性，还可以批量操作，具体用法将在后面介绍。

4. 顶部工具栏→常用功能

顶部工具栏显示了版本控制当中最常用的几个功能，如图 14-6 所示。

图 14-6

从左到右依次是：更新项目、提交变更、跟远程仓库中的文件进行对比、显示历史、撤销操作。

5. 状态栏→分支操作

状态栏提供了Git的分支操作，在这里我们可以对本地或远程的分支进行各种操作，比如检出、创建新的分支、对比、合并、删除等，如图 14-7 所示。

图 14-7

14.2 Git 偏好设置

当我们把Git的环境配置好后，在Android Studio的偏好设置中只需要使用默认的配置就可以了。一般不需要特殊配置，但也不排除特殊需求。下面我们介绍一下Git的偏好设置。

设置步骤：偏好设置→Version Control→Git，如图 14-8 所示。

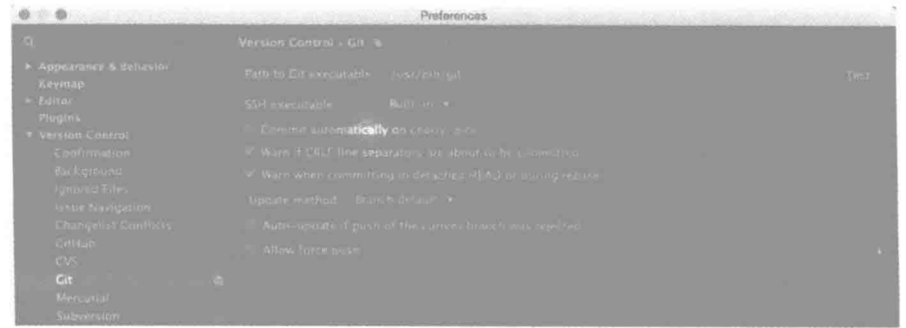


图 14-8

- Path to Git executable：Git 执行路径，这里使用的是默认路径，如果自定义了 Git 路径，记得要改一下，不然会报错。
- SSH executable：执行 SSH，默认是使用 Android Studio 内建的，还可以选择使用 Native（本地）的。
- commit automatically on cherry-pick：在 cherry-pick 时是否自动提交。

扩 展

cherry-pick 可以选择某一个分支中的一个或几个 commit 来进行操作。

- Warn if Crlf Line separators are about to be committed：如果回车换行符被提交是否警告。

- Warn when committing in detached HEAD or during rebase：提交 detached HEAD 或 rebase 时是否警告。

detached HEAD 用来让 HEAD 随便指向某个 commit id，rebase 用来合并代码。

- Update method：选择代码更新方式，包括 Branch defaut、merge、rebase。
- Auto-update if push of the current branch was rejected: push 当前分析被拒绝是否自动更新。
- Allow force push：是否允许强制 push。

14.3 配置 GitHub 账户信息

为了后面方便地使用GitHub来管理代码，我们需要配置好GitHub的账户信息。

第 1 步：进入偏好设置。

偏好设置→Version Control→GitHub。

第 2 步：输入账户信息。

01 输入 Host：https://github.com。
02 在 Login 输入用户名，在 Password 输入密码 （如果没有请先去注册）。
03 Test 连接是否成功，如图 14-9 所示。

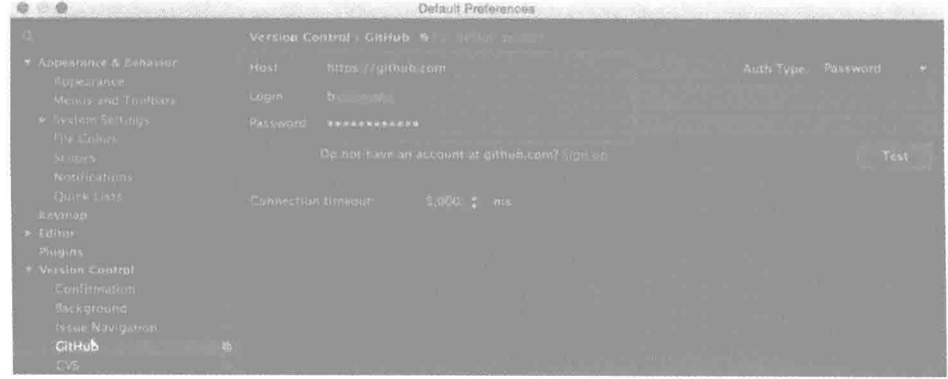


图 14-9

14.4 从 GitHub 克隆代码

如果想使用GitHub上的开源项目，可以使用Android Studio直接下载项目代码。

第 1 步：进入GitHub克隆界面。

欢迎界面：Check out project from Version Control→GitHub

菜单栏：File→New→Project from Version Control→GitHub

菜单栏：VCS→Checkout from Version Control→GitHub（见图 14-10）。

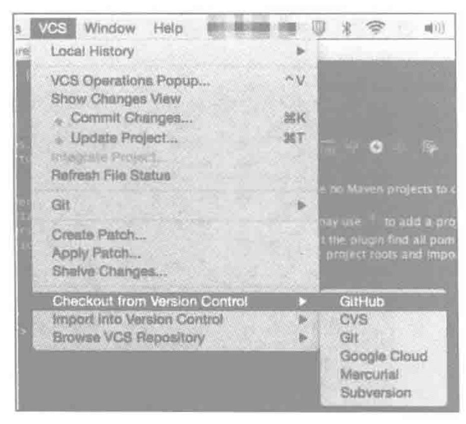


图 14-10

第 2 步：配置克隆项目地址。配置GitHub Repository，如图 14-11 所示。单击【Test】按钮可测试能否连接成功。

第 3 步：下载并打开项目。单击【Clone】按钮开始从给定的仓库地址克隆项目到本地，完成后会弹出一个【Import Project from Gradle】确认对话框，如图 14-12 所示。单击【OK】按钮打开项目。

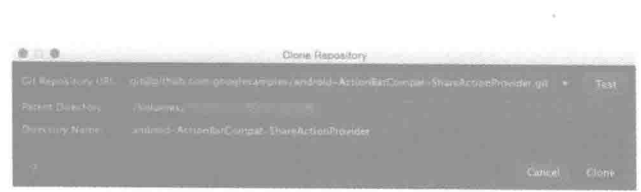


图 14-11

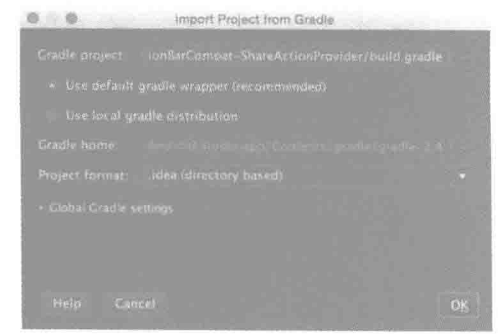


图 14-12

14.5 将本地项目共享到 GitHub

如果本地的项目想放到GitHub上开源，可以直接将本地项目共享到GitHub。

第 1 步：单击【Share Project on GitHub】。打开本地项目→菜单栏→Import into Version Control→Share Project on GitHub。

第 2 步：确定是本地项目。如果此项目已经在GitHub上，在Android Studio右上角会有相应的提示信息，单击GitHub可以跳转到项目主页。如果此项目没有在GitHub上，就会弹出一个新建仓库的对话框，如图 14-13 所示。

图 14-13

第 3 步：指定首次提交的文件和提交的备注，如图 14-14 所示。单击【OK】按钮以后在状态栏会显示进度。

第 4 步：确认分享成功。分享成功后在Android Studio的状态栏和右上角会显示成功提示，如图 14-15 所示。如果想查看GitHub上到底有没有分享成功，可以单击【SecondApp】到项目主页查看，如图 14-16 所示。

图 14-14

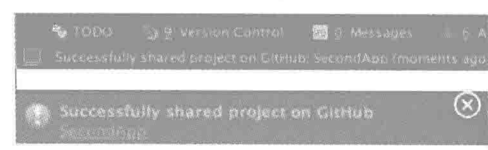


图 14-15

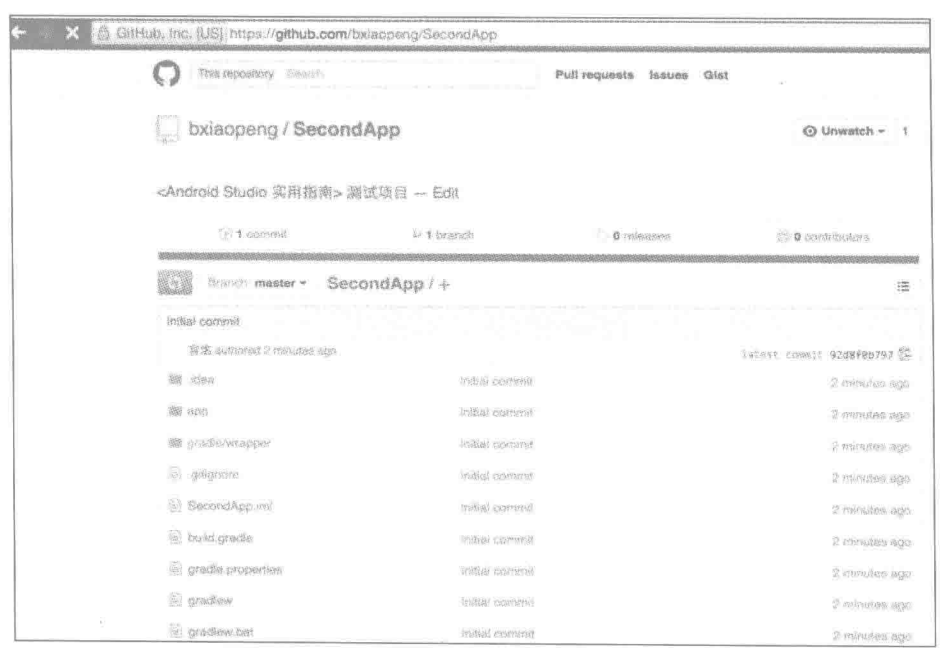


图 14-16

14.6 查看本地变更历史

1. 查看某个文件的本地变更历史

前提条件：在文件列表中选中此文件或打开此文件，光标在文件中。

操作步骤：右击→在弹出的操作选项中单击Local History→Show History（见图 14-17），或者通过菜单栏（见图 14-18）。

图 14-17

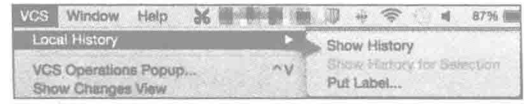


图 14-18

显示变更历史

如果两个文件不同就会提示不同的个数，窗口底部列出了不同颜色表示的变更操作，如图 14-19 所示。

图 14-19

2. 查看某段代码的本地变更历史

如果想查看某段代码本地的变更，需要先选中这段代码，然后在Local History列出的选项中选择【Show History for Selection】，如图 14-20 所示。

3. 查看某个文件夹的本地变更历史

如果想要查看某个文件夹的本地变更，只需要选中这个文件夹，然后右击选择Local History→Show History。我们还可以为每次变更贴上一个标签（Put Label），只要在Local History中选择Put Label，然后输入标签内容就可以了。

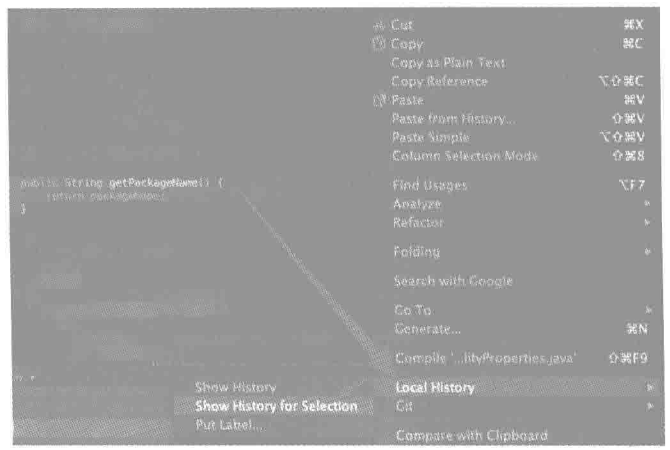


图 14-20

14.7 Git 添加文件

在讲添加文件之前先要了解一下Git中文件的状态和提交流程。我们的工具目录下的文件只有两种状态，即未跟踪和已跟踪。

- 未跟踪是指没有被纳入版本控制的文件，Git 就不会跟踪文件的变更。
- 已跟踪是指已被纳入版本控制的文件，在上一次的快照中有它们的记录，在工作一段时间后，这些文件的状态可能是未修改、已修改或已放入暂存区。

本地文件提交到远程仓库一般的流程为：添加文件跟踪→暂存已修改的文件→提交到本地仓库→推送到远程仓库。本节我们主要讲添加文件跟踪和暂存已修改的文件。

1. 添加文件跟踪

如果我们新建了一个文件，默认是没有被跟踪的，Git就不会跟踪文件的变更，那如何跟踪文件呢？例如，在Android Studio中新建一个文件Demo.java，然后会弹出一个对话框，询问你是否要将此文件添加到Git，如图 14-21 所示。单击【Yes】按钮，此文件就被跟踪了。

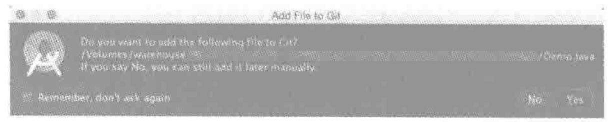


图 14-21

上面的这个方法很方便，在新建一个文件的时候会提示你进行文件跟踪。如果用别的方式新建文件，该怎么添加跟踪呢？例如，通过命令新建一个文件Demo.java。注意，文件不同的状态都用不同的颜色来表示。未跟踪的文件显示为红色，如图 14-22 所示。那我们如何将Demo添加到跟踪呢？

操作步骤：

01 右击 Demo 文件→Git→Add。

图 14-22

菜单栏：VCS→ Git→Add

快捷键：option + command + A（macOS）或者Ctrl + Alt + A（Windows/Linux）

02 添加完成后，Demo 文件颜色变为了绿色，即已跟踪，如图 14-23 所示。

03 提交到本地仓库/远程仓库的文件颜色显示为白色，如图 14-24 所示。

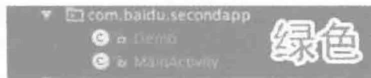


图 14-23

图 14-24

04 已修改的文件显示为蓝色。

2. 暂存已修改的文件

暂存已修改的文件方法同添加文件跟踪。

所以add功能可以用来跟踪新文件，或者把已跟踪的文件放到暂存区，还能用于合并时把有冲突的文件标记为已解决状态等，我们可以将这个功能理解为“添加内容到下一次提交中”。

不过如果想提交起来更简单，可以直接使用commit命令，在提交配置列表里选择已修改的文件和新添加的文件，在commit之前会自动添加选中的文件。

14.8 Git 提交变更

当本地文件变更以后，我们需要提交变更。

菜单栏：VCS→Git→Commit File

快捷键：command + K（macOS）或者Ctrl + K（Windows/Linux）

利用菜单栏或快捷键操作后弹出提交变更窗口（分支合并过后也会弹出提交变更窗口），如图 14-25 所示。

图 14-25

1. 配置提交信息

在提交变更窗口中可以选择Change list，也可以选择要提交的变更文件，默认是全选的。在Author中选择或者输入作者名字。选择Amend commit（修订提交）会在Commit Message中添加上一次的提交信息。

在提交之前，我们还可以选择做一些代码优化的工作，比如Reformat code（格式化代码）、Rearrange code（重新排列代码）、Optimize imports（优化导入）、Perform code analysis（执行代码分析）、Check TODO（检查待办事项）、Cleanup（清理）、Update copyright（更新版权声明）。

单击某个变更的文件，在Details中会显示本地和远程的对比。

2. 提交变更

当配置好提交信息以后，将鼠标放到Commit上面，会弹出提交操作列表。

- Commit and Push：将本地变更的文件提交到本地仓库，然后推送到远程仓库。
- Create Patch：将本地变更的文件作为补丁创建。
- Commit：将本地变更的文件提交到本地仓库。

这里选择Commit and Push。如果选择了提交之前进行一些代码检查或优化，会先执行优化。单击Review可以查看代码分析出来的问题，查看以后觉得没问题，重复上面的操作（Android Studio会有记录）。

如果单击Commit会直接提交。Commit成功后会提示推送到远程仓库，如图 14-26 所示。

默认提交到当前分支，也可以选择其他分支。单击【Push】按钮将本地变更推送到选择的远程分支。

图 14-26

14.9 Git 文件逐行追溯

如果想查看某个文件的某一行是谁修改的，就需要使用文件逐行追溯功能。

执行文件逐行追溯操作，会显示某个文件每一行的详细改动信息，也可以说是每一行的注释，包括修订版本号、提交者、提交日期以及提交次数。

如何进行文件逐行追溯？

其实就是执行git annotate命令，同git blame。

在Android Studio中执行annotate的操作路径有下面几个。

- 菜单栏：单击 VCS→选择 Annotate

- 右击文件编辑区域→单击 Git→选择 Annotate
- 右击文件左边栏→选择 Annotate（见图 14-27）。

图 14-27

通过上述操作后会显示文件每一行的注释，如图 14-28 所示。如果我们想追溯某一行的改动，单击这一行的注释，就会弹出那次提交的改动文件列表以及提交信息，如图 14-29 所示。

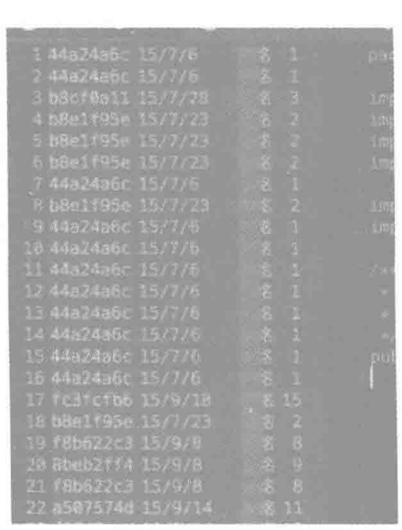


图 14-28

图 14-29

如果我们想在进行逐行追溯的时候省点力气，或者让review代码的人轻松一点，就要注意每次提交不要改动太多文件。提交信息（commit message）一定要写清楚改动的目的以及影响点，这些可以写到团队协作规范里。

14.10 显示当前修订版本

修订版本号（revision）是一组SHA-1 值，可以通过SHA-1 值来获取对应的那一次提交。SHA-1 是散列函数加密算法，输出的散列值为 40 位十六进制数字串，可用于验证信息的一致性，防止被篡改。

如何显示当前的修订版本呢？

操作步骤：菜单栏→VCS，或者右击某一个文件或文件编辑区→Git→Show Current Revision，如图 14-30 所示。

图 14-30

14.11 Git 文件比较

Android Studio集成的Git提供了非常好用的文件比较功能，可以将本地文件与远程仓库中的、某次提交的或其他分支的文件进行比较。

使用Git文件比较方法：菜单栏→右击某一个文件或文件的编辑区→VCS→Git，或者底部工具栏→Version Control→右击有变更的文件→Git。

比较功能有下面几个。

- Compare with the Same Repository Version：比校本地文件与远程仓库的文件。
- Compare with Latest Repository Version：本地文件与最近的一次提交比较。
- Compare with：本地文件与某一次提交比较。
- Compare with Branch：本地的文件或文件夹与某个分支上的进行比较。

1. 比校本地文件与远程仓库的文件

如果本地某一个文件被修改了，我们想查看它与远程仓库中的文件有什么不同，就需要比较一下。

比较方法：右击某一个文件或文件的编辑区→Git→Compare with the Same Repository Version，如图 14-31 所示。

图 14-31

2. 本地文件与提交的版本进行比较

这种比较包括Compare with Latest Repository Version和Compare with。单击Compare with Latest Repository Version选项会将本地文件与最近的一次提交进行比较。单击Compare with选项会显示文件的提交列表，可选择某一次提交进行比较，如图 14-32 所示。

3. 比较不同分支的文件或文件夹

前面讲的几个比较功能都是同一个分支上的文件进行比较，还可以跟其他分支上的文件或文件夹进行比较，利用Compare with Branch选项即可完成，如图 14-33 所示。

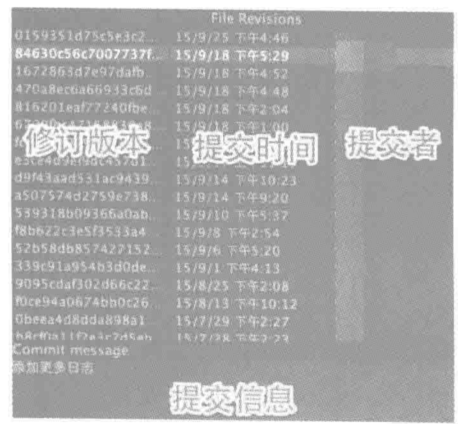


图 14-32

图 14-33

14.12 Git 撤销操作

如果我们对某个或某几个文件进行了修改，现在想撤销这些修改，可以使用Git提供的撤销操作功能。Android Studio中提供了多个快捷操作方式，可以方便地进行撤销操作，如图 14-34 所示。

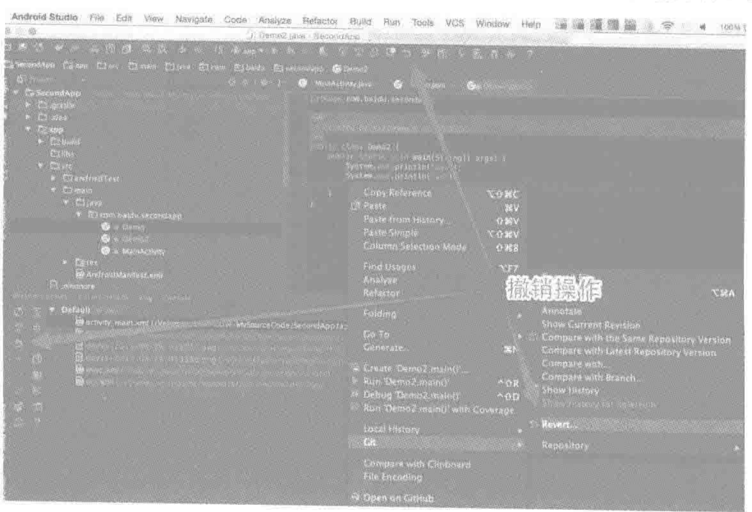


图 14-34

实际操作如图 14-35 所示。单击【Revert】按钮确认后，被选中的文件就恢复到了改动之前的状态。

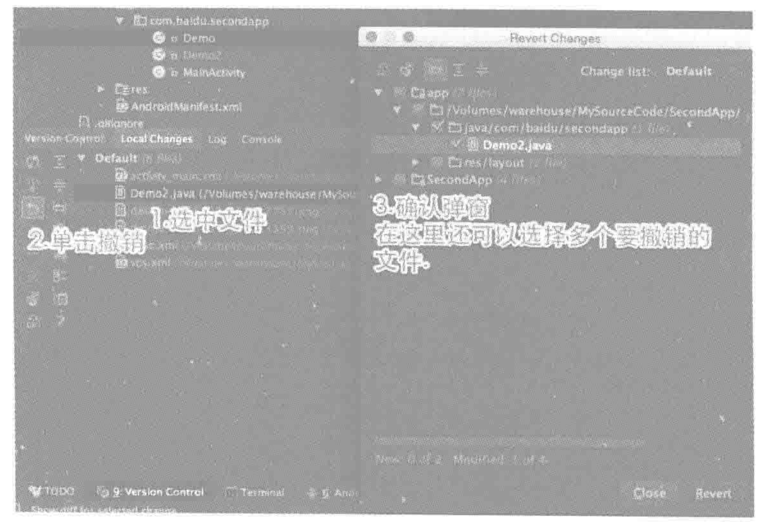


图 14-35

14.13 Git 版本回退

Git版本回退的意思就是将本地代码回退到某一个指定的版本，此版本之前的所有内容都会被重置。

操作方法：菜单栏→VCS→Git→Reset HEAD，然后弹出Reset Head对话框，如图 14-36 所示。

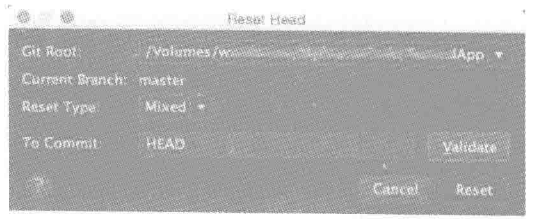


图 14-36

- Reset Type：回退类型。
  - ➢ Mixed：回退到某个版本，本地源码不会回退，会回退commit和index信息。
  - ➢ Soft：回退到某个版本，本地源码和index信息不会回退，只回退了commit的信息，如果还要提交，直接commit即可。
  - ➢ Hard：彻底回退到某个版本，本地的源码也会变为某个版本的内容。
- To Commit：回退版本。在 To Commit 中配置要回退到哪个版本，默认是 HEAD。
  - ➢ HEAD：回退到最近一个提交版本。
  - ➢ HEAD^：回退到上一个提交版本。
  - ➢ HEAD^^：回退到上上一个提交版本。

  ……

  以此类推。

  还可以使用另外一种方式回退版本：
  - ➢ HEAD~0：回退到最近一个提交版本。

➢ HEAD~1：回退到最近一个提交版本。

➢ HEAD~2：回退到最近一个提交版本。

……

以此类推。

- Validate：版本验证。在指定回退到某个版本之前也可以先验证一下是否是想回退的版本，如图 14-37 所示。确认后单击【Reset】按钮就可以回退到指定的版本了。

图 14-37

14.14 Git 查看提交历史

我们在 14.6 节介绍过了【查看本地变更历史】，在这一节我们介绍Git查看提交历史记录功能。

跟本地变更历史提供的功能相同，Git查看提交历史记录也都有查看文件、文件夹和代码段的历史记录功能，不同点在于【查看本地变更历史】仅能查看本地的变更，远程的改动是无法看到的，【Git查看历史记录】可以查看所有commit以后的历史。

因此当我们想查看某个文件或文件夹提交的历史记录的时候，可以使用此功能。

1. 查看某个文件的提交历史

前提条件：在文件列表中选中此文件或打开此文件，光标定位在文件中。

操作步骤：右击→在弹出的操作选项中单击 Git→Show History→Show History（见图 14-38），然后会在 Version Control 中显示 History，如图 14-39 所示。

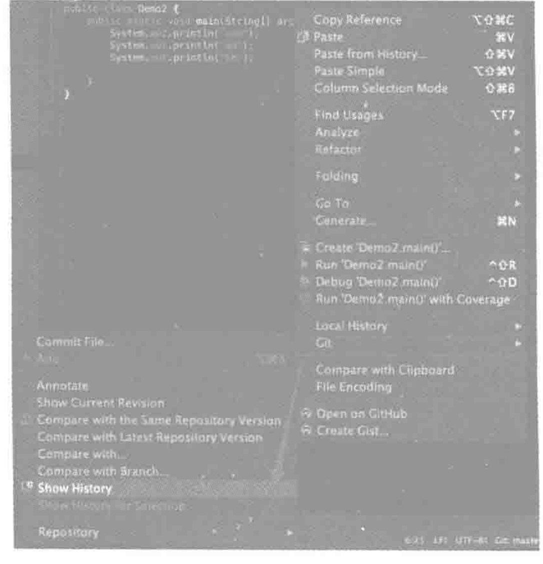


图 14-38

图 14-39

2. 查看某段代码的提交历史

如果我们想查看某段代码提交历史记录，需要先选中这段代码，然后在Git操作列表中选择【Show History for Selection】，如图 14-40 所示。还可比较上一次提交与上上一次提交结果，如图 14-41 所示。

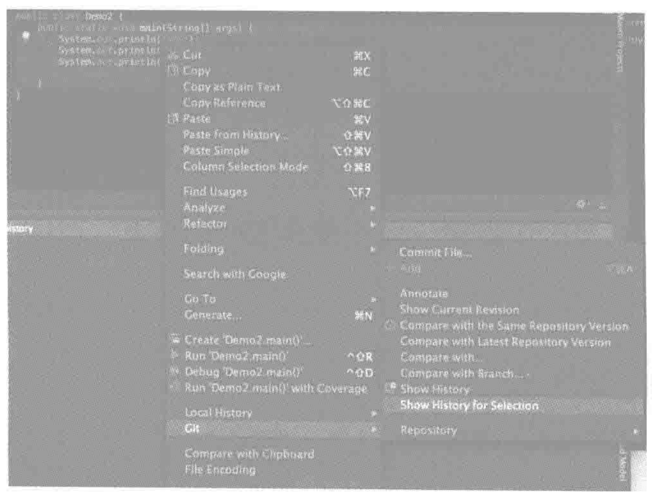


图 14-40

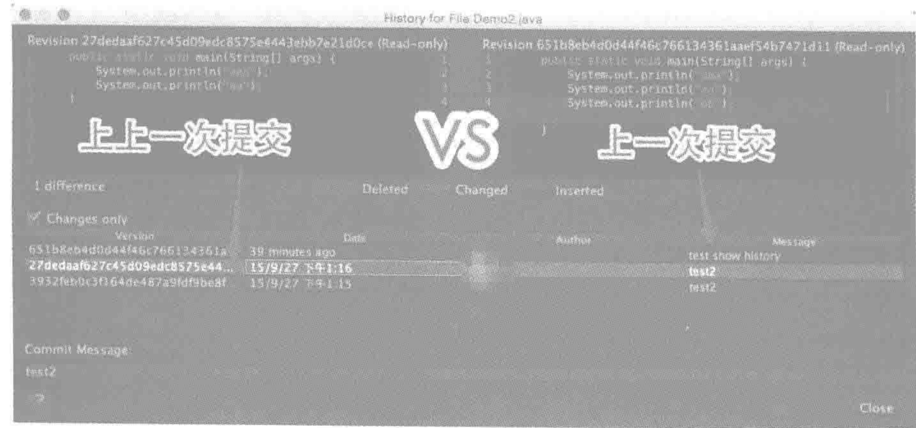


图 14-41

14.15 Git 分支管理

1. 什么是分支

当我们在进行软件开发时，同一个软件多个人协同开发，因此要有不同的分工。如果想让彼此的代码不受影响，就需要在不同的分支上进行开发，开发完成后再进行合并。

分支可以理解为一个主干衍生出来的枝干，可以在这些枝干上修改代码，且彼此不受影响，这样做的好处就是在同一个数据库里可以同时进行多个修改，最终会合并到一起。

Android Studio中Git的分支管理特别方便，我们可以通过Git操作列表中的branches或状态栏的分支管理来对本地或远程的分支进行各种操作，比如检出、创建新的分支、对比、合并、删除等。

Local Branches列出了本地所有的分支，这些分支不一定全部都推送到了远程仓库，Remote Branches列出了所有的远程仓库中的分支，如图 14-42 所示。

2. 新建分支

单击【New Branch】→输入分支的名字→OK，如图 14-43 所示。

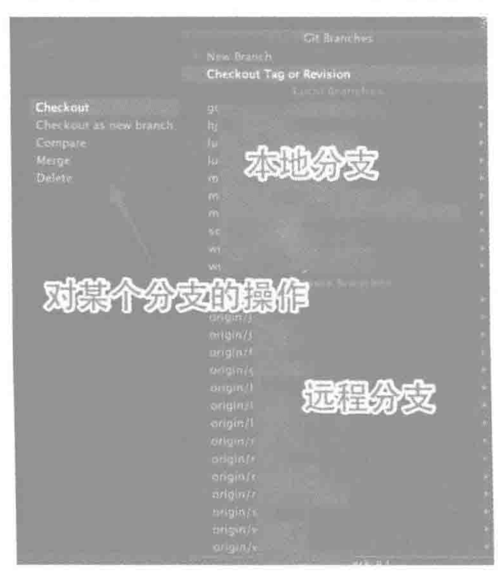


图 14-42

图 14-43

3. 检出分支

检出分支作为新的本地分支，意思就是你可以把一个远程分支checkout下来，可以起不同的名字，可以是多个本地分支，但是提交时都是向同一个远程分支进行提交。

操作步骤：选择一个分支（远程或本地分支）→单击【Checkout as new local branch】，如图 14-44 所示。

如果是远程分支就会默认使用远程分支名作为本地分支名。但是如果本地已经有了分支名，就会提示你重命名，如图 14-45 所示。

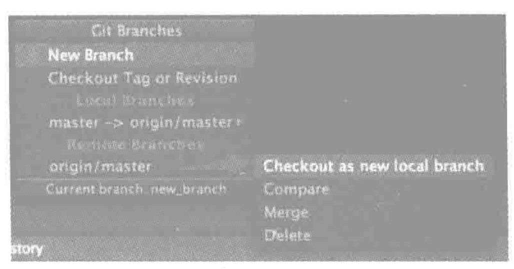


图 14-44

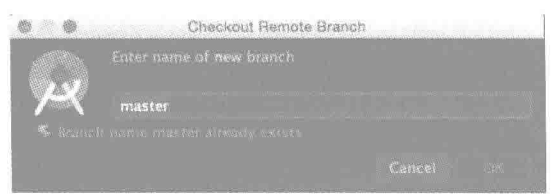


图 14-45

输入新的分支名→OK→成功检出，如图 14-46 所示。

如果是对本地分支进行【Checkout as new local branch】操作，就会直接弹出一个【Checkout New Branch】输入框，不会有默认的分支名，如图 14-47 所示。

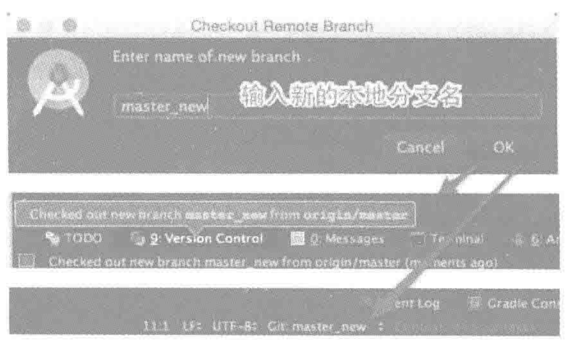


图 14-46

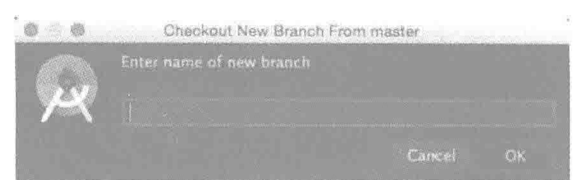


图 14-47

4. 检出分支/标签/修订版本

如果想检出某个分支/标签/修订版本，只需要知道对应的名字就可以。在Git Branches操作列表中选择【Checkout Tag or Revision】→在弹出的输入框中输入分支名→OK后会检出对应的分支，如图 14-48 所示。

检出tag也只需要输入tag名就可以了。如果检出当前分支的某个修订版本，就需要输入对应的修订版本号。

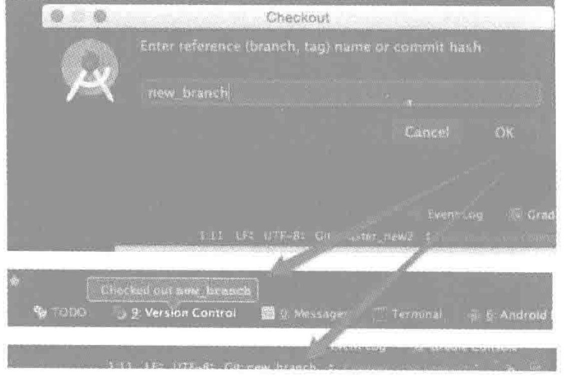


图 14-48

5. 切换分支

切换分支需要选择要切换的分支，单击【Checkout】，如图 14-49 所示。如果原来的分支没有变更，checkout后会切换到新的分支；如果有变更且有冲突，就会先解决冲突再切换。

6. 分支对比

分支对比是拿当前分支与另外一个分支（本地或远程）进行对比，如图 14-50 所示。

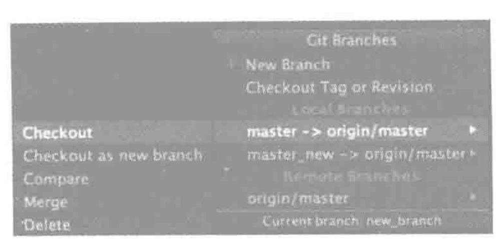


图 14-49

图 14-50

7. 对比两个分支的提交日志（见图 14-51）

图 14-51

8. 对比两个分支的不同（见图 14-52）

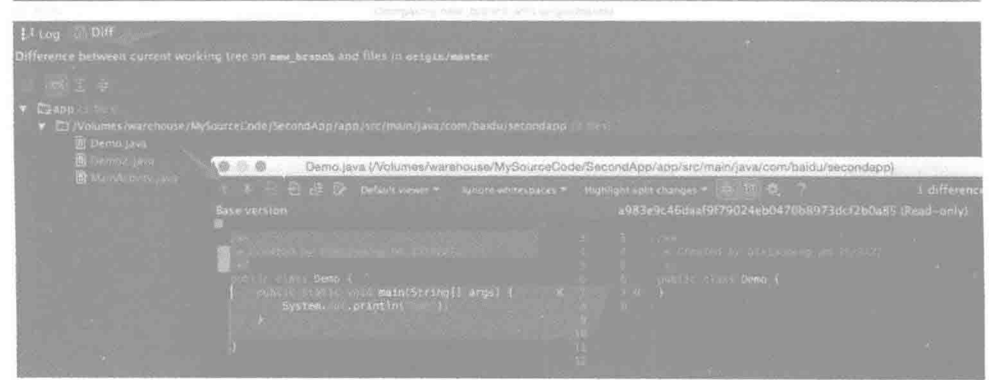


图 14-52

9. 合并分支

合并分支只将选中的分支与当前的分支进行合并，如图 14-53 所示。如果有冲突就会弹出冲突列表。解决冲突后可合并成功，合并冲突将会在后面介绍。

图 14-53

10. 删除分支

- 删除本地分支：本地分支会被直接删除。
- 删除远程分支：删除远程分支会有确认是否删除，确认删除后状态栏会有正在删除的信息显示，删除成功后 Version Control 上会有删除成功提示，Remote Branches 列表中此分支也会消失。

11. 提交分支

提交分支就是将本地推送到远程仓库。

14.16 Git 创建标签

Git可以给历史中的某一个修订版本打上标签，通常我们会使用标签来表示一个版本的发布。

操作步骤：菜单栏→VCS→Git →Tag，然后弹出创建标签配置窗口，如图 14-54 所示。

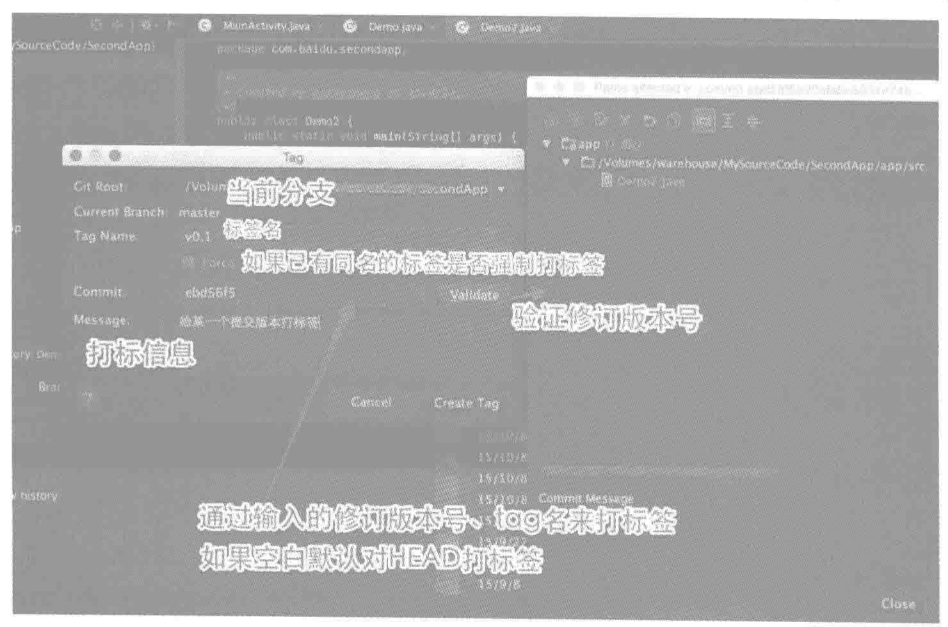


图 14-54

- Git Root：项目地址。
- Current Branch：显示当前的分支。
- Tag Name：标签名。
- Force：如果相同的 Tag 名已经存在，是否强制创建。
- Commit：通过输入的修订版本号、Tag 名来打标签。如果空白，默认对 HEAD 打标签。
- Create Tag：创建标签。

配置好以后单击【Create Tag】按钮，如图 14-55 所示。

另外，我们还可以在Version Control中右击某个提交信息，然后创建标签，如图 14-56 所示。

图 14-55

图 14-56

本地创建标签成功以后，推送到远程仓库就可以了，如图 14-57 所示。

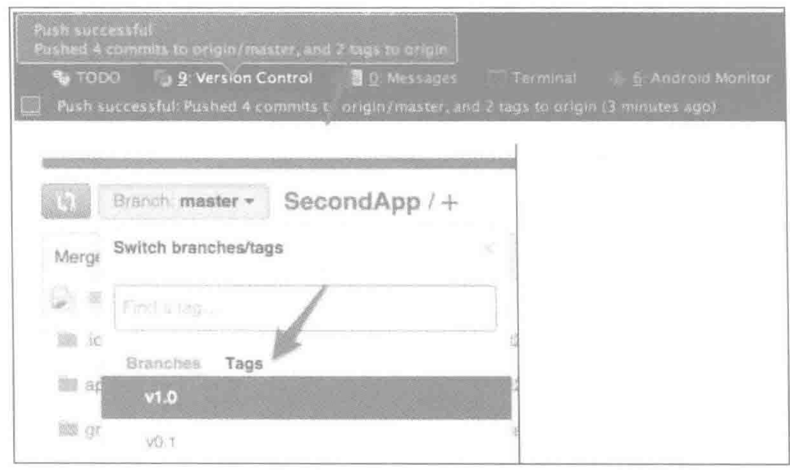


图 14-57

14.17 Git 合并分支

在Android Studio中合并分支操作非常简单，假设我们需要将master上的代码合并到当前分支。

第 1 步：选择合并变更。

右击项目→在弹出的操作列表中选择 Git→Repository →Merge Changes（合并变更）→弹出合并分支对话框。

第 2 步：配置合并信息（见图 14-58）。

- Current Branch：显示了当前的分支。

- Branches to merge: 显示了仓库中所有的分支，选择一个或多个要合并过来的分支。
- Strategy: 提供了可选择的合并策略，请按需选择。
- Commit Message: 添加本次合并的注释。

第 3 步： 合并分支。

配置好合并信息以后单击【Merge】按钮开始合并代码。如果合并分支的过程中有冲突，需要先解决冲突。如果合并分支的过程中没有冲突，就意味着合并成功了，然后可以直接提交合并完成后的代码。

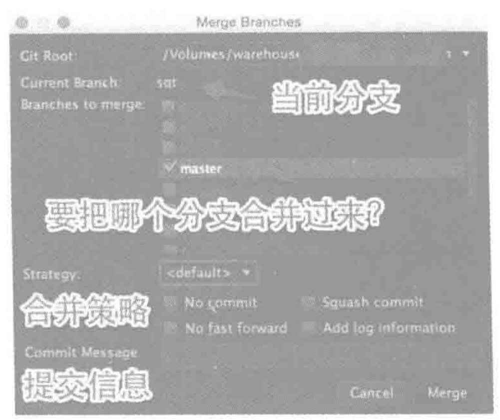


图 14-58

14.18 解决 Git 合并中的冲突

在解决冲突之前需要了解文件冲突的两个问题。

问题 1：合并分支为什么会有冲突?

因为在项目开发的过程中，可能大家会同时改动某个文件，当同一个文件在远程分支和本地分支按不同的方式被修改时，在合并分支的时候就会出现冲突。

问题 2：文件冲突会导致什么样的后果?

合并过程中如果文件冲突，合并就会失败，Git会在索引和工作树里设置一个特殊的状态，提示你如何解决冲突。我们必须解决完冲突并且更新索引后才能够将文件提交。在Android Studio中集成了非常直观方便的工具，可以帮助我们轻松解决合并时的文件冲突。

1. 打开文件冲突列表

接上一节，合并过程中如果有冲突就会弹出文件冲突列表，如图 14-59 所示。如果冲突列表被不小心关掉了，没关系，通过Git→Resolve Conflicts再打开就可以了。

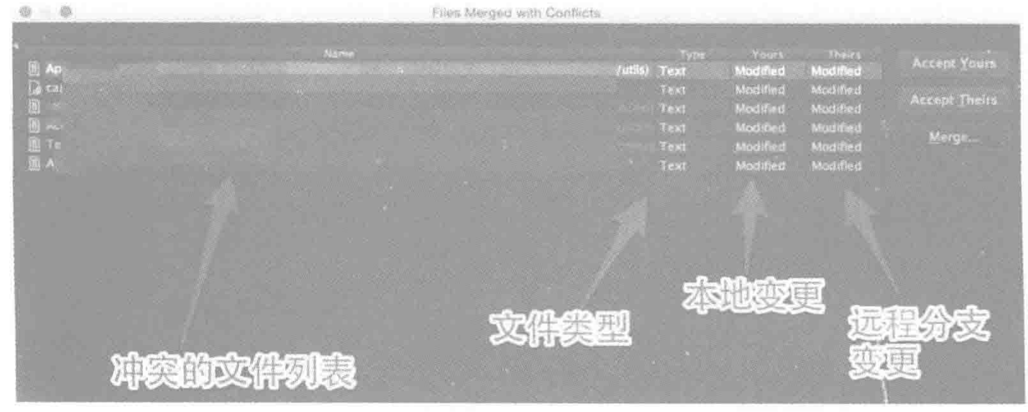


图 14-59

2. 解决冲突的方法

- 如果想放弃本地的改动，而完全使用远程分支的改动，可选择【Accept Theirs】，此文件冲突解决。
- 如果想放弃远程的改动，而完全使用本地的改动，可选择【Accept Yours】，此文件冲突解决。
- 如果想详细地解决冲突，单击 【Merge】→会打开文件比较窗口，如图 14-60 所示。左边是本地变更，右边是远程分支上的变更，中间是合并结果。如果把右边远程分支的变更用作合并结果，单击<<远程的变更就会到合并结果中了。同理，也可将本地变更合并到结果中。

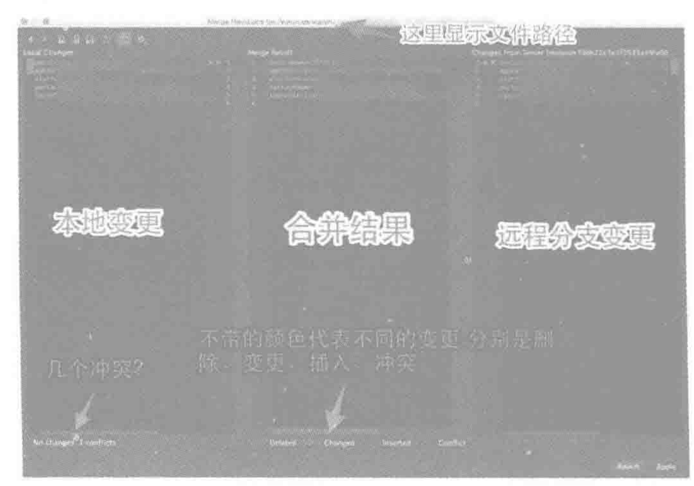


图 14-60

如果觉得改错了要恢复，可单击【Revert】。如果确定了合并结果，就单击【Apply】，此文件冲突解决。待所有冲突合并完成之后提交合并结果。

14.19 Git 使用 Rebase 合并分支

Git Rebase是什么?

Rebase顾名思义是‘变基’的意思，即合并时重新定义分支的根基或节点，让提交记录按时间线性排列。

那么它们的区别在哪?

Git Rebase跟Merge的区别是什么？

Rebase跟Merge一样，都是用来合并分支的，但是Merge操作会生成一个新的节点，与之前的提交分开显示；Rebase操作不会生成新的节点，而是将两个分支融合成一个线性的提交。

我们可以从下面看出区别：

```
# 两个分支 dev 和 master
```

```
      D---E (dev)
     /
A---B---C---F  (master)

# git merge 合并两个分支

      D--------E
     /          \
A---B---C---F----G  (dev, master)
# G是merge后产生的新节点

# git rebase 合并两个分支

A---B---D---E---C'---F'  (dev, master)
#Rebase 操作没有生成新的节点，而是将两个分支融合成一个线性的提交。
```

什么时候使用Git Rebase?

使用Rebase合并分支会使提交记录看起来非常清晰，因为它是以时间线线性地显示记录，所以当你想线性地查看分支的提交记录时，可以使用Git Rebase来合并分支。

在Android Studio中如何使用Git Rebase？我们可以通过例子来一起看一下。

【实例演示 1】

第 1 步：在dev分支中新增两个提交（commit）。

在Version Control中查看dev新增了两个提交日志，如图 14-61 所示。

第 2 步：在master分支中新增两个提交。

在Version Control中查看有两个提交日志显示，如图 14-62 所示。

图 14-61

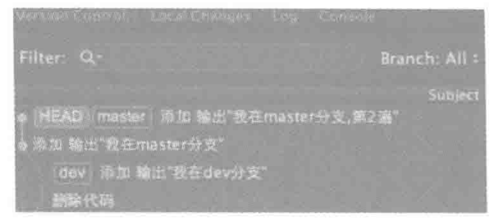


图 14-62

这个时候能看出master和dev这两个分支的提交信息是非线性显示的。

继续举例，现在需要在master分支上使用rebase把dev分支合并过来。

第 3 步：配置Rebase分支对话框。

在菜单栏中选择VCS→Git→Rebase→弹出配置对话框，如图 14-63 所示。

- Branch：显示当前分支。

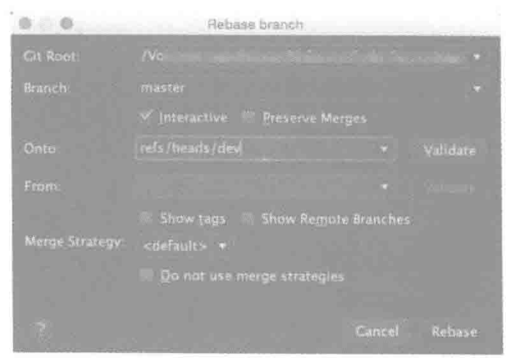


图 14-63

- Onto：要把哪个分支或修改版本合并到当前分支，既可以选择下拉列表中列出的分支，也可以输入修订版本号。
- Interactive：默认选中，即进行交互式的 rebase，意味着在 rebase 的过程中可以对一部分提交进行修改。

验证分支：本例中我们选择把dev分支合并过来，单击【Validate】验证dev有没有新的提交。如图 14-64 所示，显示有提交。如果同一个文件有多次提交，这里显示最新的一次。

第 4 步：配置交互式Rebase提交。

在Rebase Branch中单击Rebase→弹出Rebase Commits对话框（见图 14-65）。

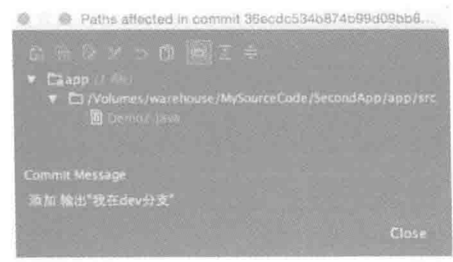


图 14-64

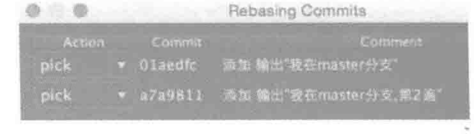


图 14-65

这里列出了master上的两个提交，可以对每个提交定义rebase行为。默认为pick，意思是以同样的提交信息保存提交。

行为列表，如图 14-66 所示。

- pick：以同样的提交信息保存提交。
- edit：在对下一提交进行操作之前，会让你对提交进行修正。
- skip：遇到冲突暂停时，跳过这个提交。
- squash：把这个提交和前一个提交合并成为一个新的提交。
- reword：提交时要编辑提交信息。
- fixup：和 squash 类似，但是要放弃提交信息。

图 14-66

调整顺序和查看详情，如图 14-67 所示。

图 14-67

第 5 步：开始Rebase。单击【Start Rebasing】，开始rebase，状态栏会显示正在rebasing，如图 14-68 所示。

图 14-68

第 6 步：解决冲突。右上角显示rebase进度，如果有冲突会有提示，这里我们遇到了冲突，如图 14-69 所示。

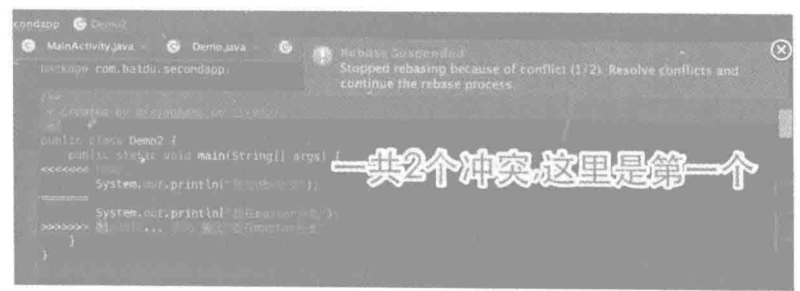


图 14-69

想办法解决冲突：Git→Resolve Conflicts→显示冲突→解决冲突。

第 7 步：继续Rebase。

解决完这个冲突以后继续Rebase，单击Git→Continue Rebasing ，如图 14-70 所示。

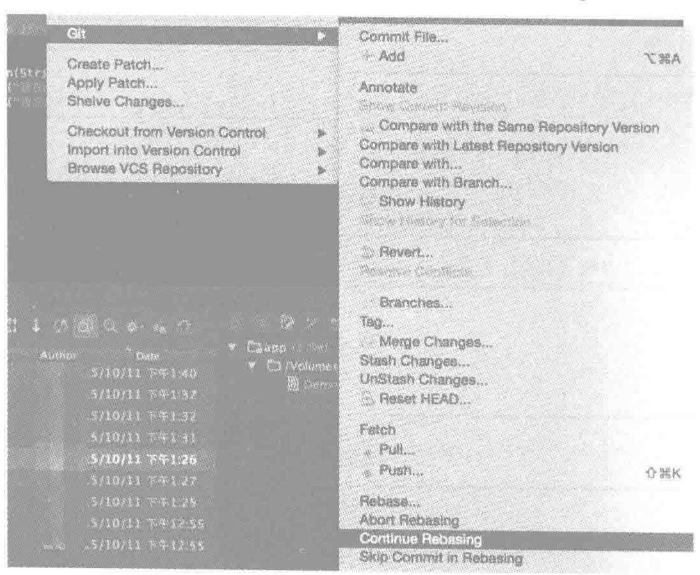


图 14-70

提 示

我们还可以选择 Abort Rebasing（终止 Rebase）、Skip Commit in Rebasing（跳过这个提交）。然后继续 Rebase 合并，一个提交合并完成前会弹出【Additional Rebase Input】，要求输入 commit 信息。

第 8 步：提交信息。

还记得我们在配置Rebase提交行为的时候选择的pick吧？对，因为我们选择了pick，所以这里默认以同样的提交信息保存提交，如图 14-71 所示。

当然你可以修改提交信息。为了举例，这里写了Rebase something，如图 14-72 所示。

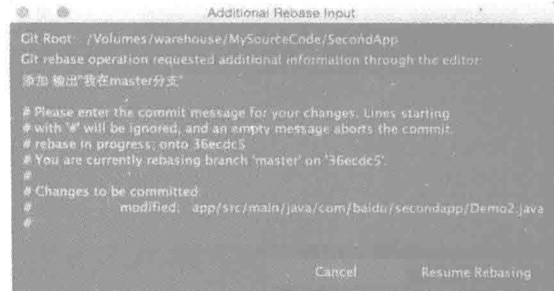


图 14-71

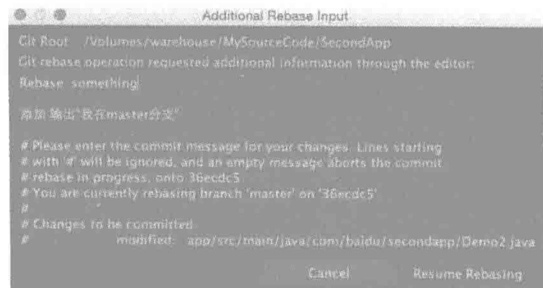


图 14-72

提 示

最好使用默认的提交信息，因为这里的改动会覆盖提交日志列表中的，所以如果你改了别人的提交信息就不好了。当然，如果你要是想重新编写提交信息，在这里修改就对了。

单击【Resume Rebasing】按钮继续。在Rebase第二个提交的时候又有冲突，解决办法跟第一个一样，先解决冲突，然后继续Rebasing，如图 14-73 所示。

```
Rebase Suspended
Stopped rebasing because of conflict (2/2). Resolve conflicts and
continue the rebase process.

package com.baidu.secondapp;

public class Demo2 {
    public static void main(String[] args) {
        System.out.println("我在master分支");
<<<<<<<
        System.out.println(
=======
        System.out.println(
>>>>>>>
    }
}
```

图 14-73

最后添加提交日志，如图 14-74 所示。

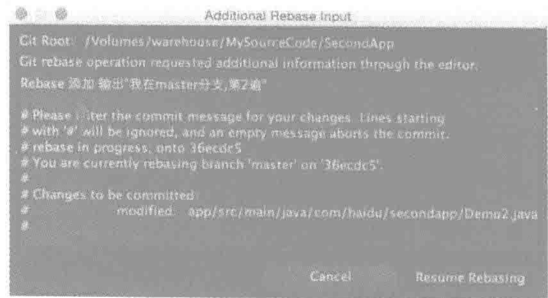


图 14-74

第 9 步：使用Rebase完成。

继续Rebase，这时提交日志的记录已经变成线性的了，并且是按照提交时间来进行排序的，看起来清晰多了，如图 14-75 所示。

图 14-75

【实例演示 2】

上面我们讲的是使用Rebase把dev合并到master，合并完成后如果想把master再合并到dev上，可以切换到dev分支，使用Rebase把master合并到dev，如图 14-76 所示。

图 14-76

弹出提示，如图 14-77 所示。前面的提交日志都已经是线性的了，并且当前分支（dev）是直接在base分支下面的，所以没有新的提交进行Rebase。单击【OK】按钮，当前分支就变成base分支了，如图 14-78 所示。

图 14-77

图 14-78

14.20 Git 暂存/恢复暂存变更

暂存是什么意思?

Git暂存就是从最近的一次提交中读取相关内容，让工作区保证和上次提交的内容一致，相当于清空本地的所有改动，只不过这些改动进行了备份，我们可以选择性地进行恢复。

什么时候使用暂存?

当我们正在编写一个功能的时候，突然有个紧急的BUG需要修复，我们需要切换到另外一个分支修复BUG，但是当前分支的功能还没有完成，又不想提交，就可以使用暂存了。

将写到一半的代码暂存，然后切换到其他分支修复BUG，BUG修复完成后再切回当前分支，然后恢复之前暂存的代码继续编写。

暂存和恢复暂存怎么用？

下面通过一个例子来介绍暂存和恢复暂存怎么用。假设当前分支为dev，工作区是最近的一次成功提交的代码，如图 14-79 所示。我们添加一段代码，如图 14-80 所示。

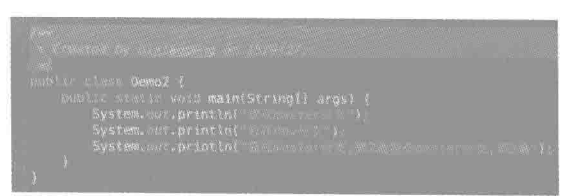


图 14-79

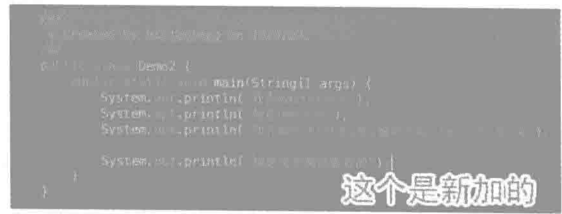


图 14-80

这时master分支有一个紧急BUG需要修复，就可以先暂存当前工作区改动的代码，如图 14-81 所示。并给这个暂存起个名字，如图 14-82 所示。单击【Create Stash】创建暂存。然后我们会发现刚才的改动不见了，工作区的代码恢复到了最初的代码，如图 14-83 所示。

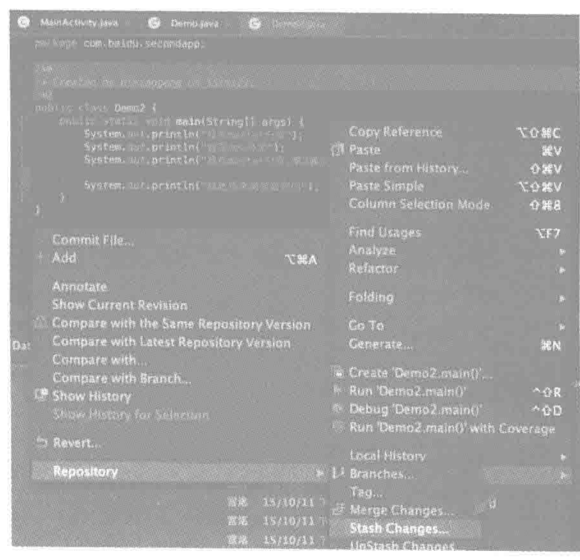


图 14-81

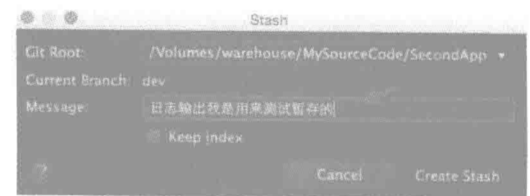


图 14-82

图 14-83

这时我们可以切换到master分支去修复BUG。过去了很长时间，OK，BUG修改好了，再切回dev。需要恢复之前暂存的代码，可以按图 14-84 进行操作。

此时会弹出一个暂存列表，如图 14-85 所示。

选择之前暂存的代码，单击【Apply Stash】按钮进行恢复，如图 14-86 所示。

之前暂存的代码回来了，暂存列表如图 14-87 所示。如果不删除暂存记录，那么记录会一直存在。很明显，你可以对暂存的代码进行View（查看）、Drop（放弃）、Clear（清空）等操作。

图 14-84

图 14-85

图 14-86

图 14-87

14.21 Git 获取最新内容

从远程仓库获取项目中最新的内容，包括远程仓库中分支的状态，可以通过菜单栏→VCS→Git→Fetch完成。Git会去拉取远程仓库最新的内容，并清除远程服务器上删掉的分支。

【实例演示】

执行Fetch以后，在Version Control上的Console中查看执行的命令：

```
git -c core.quotepath=false fetch origin --progress --prune
```

什么意思？

```
fetch origin：拉取远程分支的内容

--prune：清除远程服务器上已删掉的分支

--progress：显示进程

core.quotepath=false：改变中文路径的默认转义
```

所以更新项目就是使用Fetch来拉取最新的内容到本地。

提示

使用 Fetch 来拉取最新内容到本地并不会自动合并代码，这是 Fetch 和 Pull 的区别。

因为Fetch不会自动合并代码，所以当我们想把远程仓库中的内容和本地内容合并的时候需要先Fetch到本地，再进行Merge操作。

14.22 Git 合并最新内容

要从远程仓库获取最新的内容，然后和本地的内容自动进行合并，可以通过菜单栏→VCS→Git→Pull完成，如图 14-88 所示。

默认的，本地分支会跟远程分支合并，如果我们不选择远程分支，pull操作就无法执行。然后Git会先执行Fetch去拉取远程仓库最新的内容，再执行Merge把远程仓库中的代码和本地代码合并。执行Pull以后在Version Control中会显示更新的详情，如图 14-89 所示。

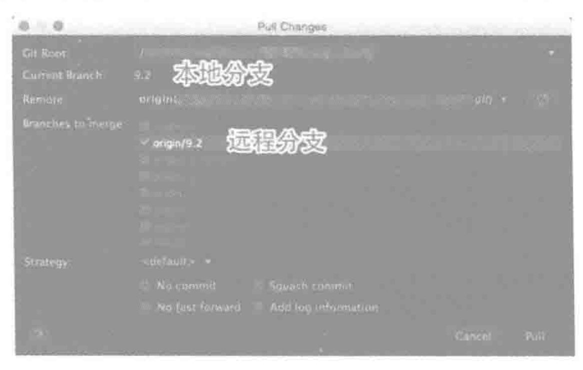


图 14-88

图 14-89

14.23 Git 更新项目

更新项目可能是我们最常用的功能了，Android Studio提供了很多快捷的方式。

- 菜单栏：VCS→Update Project
- 快捷键：Command + T（macOS）或者 Ctrl + T（Windows/Linux）
- 工具栏（见图 14-90）。

图 14-90

进行上述快捷操作后弹出Update Project对话框，如图 14-91 所示。在Update Project对话框中选择一个同步策略来将本地仓库和远程存储进行同步，还要选择一个存储变更的方式，因为更新之前要清理工程，在清理工程之前还要存储变更。

- Update Type（更新类型）：提供了 3 种类型来进行同步。
  - Merge：更新后合并，相当于先执行git fetch命令，再执行git merge或git pull——no-rebase。

> ➢ Rebase：更新后变基，相当于先执行git fetch命令，再执行git rebase或git pull——rebase。
> ➢ Branch Default：更新后使用分支默认命令，这个默认的命令在.git/config中配置。

- Clean working tree before update（更新之前先清理工作目录）：可选择在清空工作目录之前以什么样的方式来保存你的更改，当更新完成后，可以恢复之前的更改。这里提供了两种方式。
  ➢ Using Stash：使用暂存。
  ➢ Using Shelve：使用搁置。

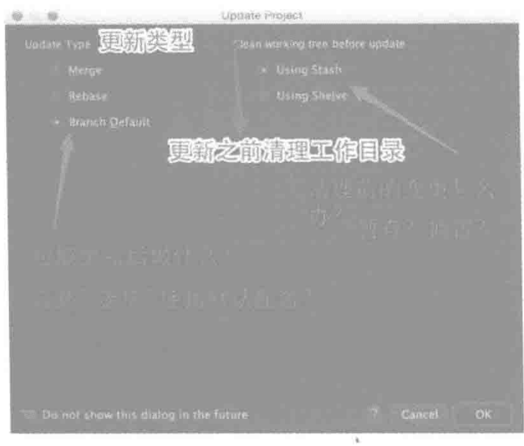


图 14-91

【实例演示】

下面是对三种更新类型的实例演示，选择不同的更新类型后，执行Update Project，在Version Control的Console会输出执行的命令。

Merge：

```
# 先获取最新的变更
10：52：40.501：git -c core.quotepath=false fetch origin --progress --prune

10：52：45.261：cd /Volumes/warehouse/MySourceCode/SecondApp
# 再合并
10：52：45.262：git -c core.quotepath=false merge --no-stat -v origin/master
Rebase：
# 先获取最新的变更
11：07：38.941：git -c core.quotepath=false fetch origin --progress --prune

...
# 再变基
11：07：46.940：git -c core.quotepath=false rebase origin/master

First, rewinding head to replay your work on top of it...

Fast-forwarded master to origin/master.
```

Branch Default：

```
# 先获取最新的变更
11：12：05.002：git -c core.quotepath=false fetch origin --progress --prune

11：12：08.650：cd /Volumes/warehouse/MySourceCode/SecondApp
# 再合并
11：12：08.650：git -c core.quotepath=false merge --no-stat -v origin/master
```

难道默认的配置中就是merge？我们来看一下。

```
bixiaopeng@172-13-5-81 SecondApp$ cat .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/bxiaopeng/SecondApp.git
        fetch = +refs/heads/*: refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
[branch "master_new"]
        remote = origin
        merge = refs/heads/master
[branch "dev"]
        remote = origin
        merge = refs/heads/master
```

14.24 刷新文件状态

如果我们想刷新本地文件的状态，可以使用此功能。使用此功能时，Android Studio会刷新每个文件的状态，不管该文件是否已经被Android Studio或其他应用更改。

操作步骤：菜单栏→VCS→Refresh File Status，或者Version Control工具窗口→Local Changes→Refresh，如图14-92所示。

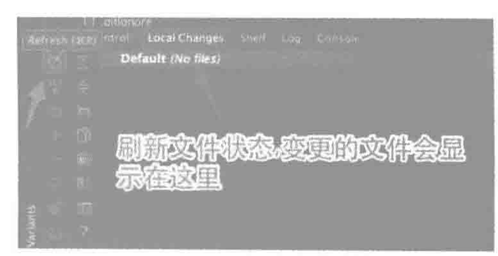


图 14-92

当然，你也可以设置定时刷新变更，如图14-93所示。

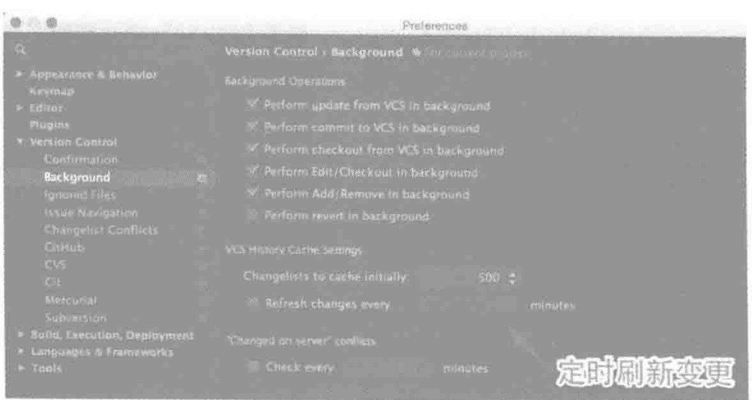


图 14-93

14.25 Git 补丁

什么是补丁

Git当中的补丁是一个文本文件，包含了对源代码的修改。你可以创建一个项目中的补丁，然后把它发给需要的人，拿到补丁文件的人可以为自己的项目打上这个补丁。

创建补丁

假设你修改了项目中的一个BUG，然后需要创建成一个补丁。

操作步骤：菜单栏→VCS→Create Patch...→ 弹出Create Patch配置对话框，如图 14-94 所示。

单击【Create Patch...】按钮，如图 14-95 所示进行操作。补丁创建成功之后就可以发送给需要的人了。

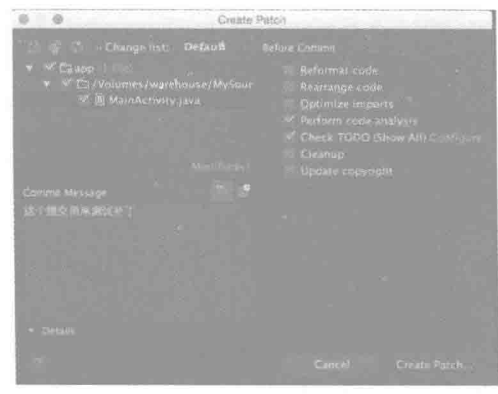


图 14-94

可以选择存储位置

创建成功

图 14-95

应用补丁

当我们拿到补丁后，可以通过菜单栏→VCS→Apply Patch...→选择补丁文件（见图 14-96）。→单击【OK】（见图 14-97）来应用补丁。

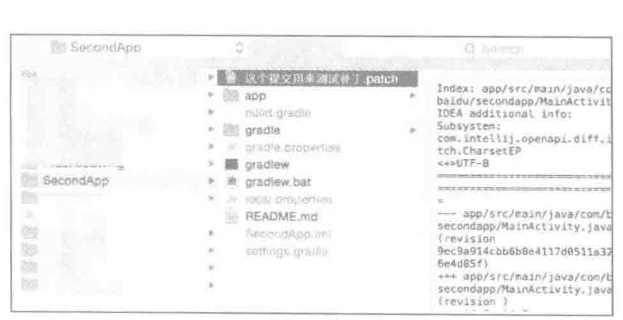


图 14-96

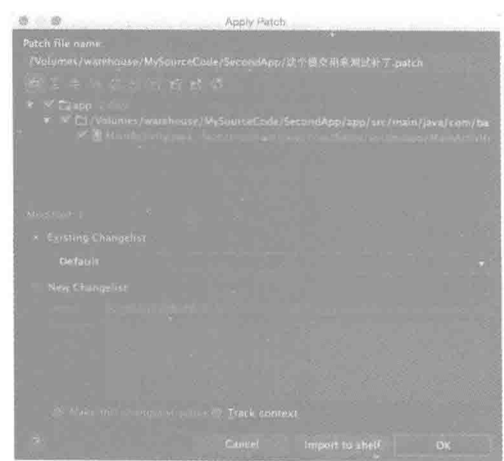


图 14-97

14.26 Git 搁置变更

Android Studio中提供了shelve的功能，意思是“将……搁在一边”，在项目中使用就是把还没有写完的代码先搁在一边，类似Git的暂存（Stash）命令，等想恢复的时候再来恢复搁置。与暂存不同的是，搁置功能相当于是本地打了个补丁，把补丁文件保存在本地，等想要恢复的时候给项目打上个补丁就可以，在后面的例子中我们会看到。

搁置变更

假设我们写了一半的代码需要暂时搁置到一边，然后需要立刻去解决另一个紧急的BUG，可以选择菜单栏中的VCS→Shelve Changes→在弹出的对话框中选择要暂时搁置的变更，如图14-98所示。

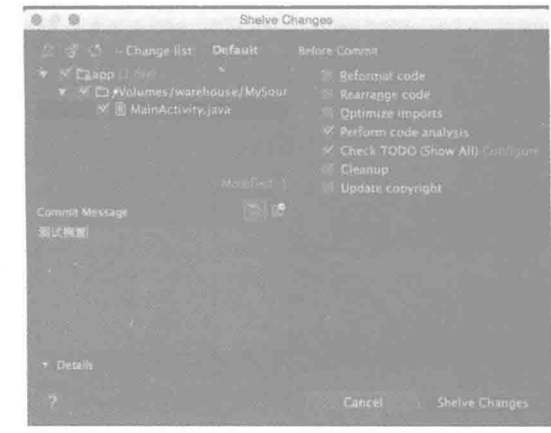


图 14-98

然后单击【Shelve Changes】，Version Control中会显示Shelf标签，Shelf中会显示暂时搁置的文件，以提交信息命名，如图14-99所示。

此时Local Changes界面如图14-100所示。搁置的代码在本地创建了一个补丁文件，后面恢复的时候就是给项目打上这个补丁。

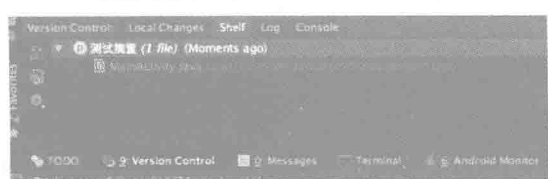


图 14-99

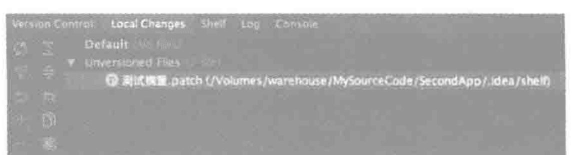


图 14-100

恢复搁置

当我们修改完BUG，回到之前的分支准备继续写代码时，需要恢复之前搁置的变更，可以使用两个选项。

- Unshelve Changes: 默认恢复搁置到新建的变更列表。

操作步骤： Version Control工具窗口→Shelf→右击之前搁置的文件→Unshelve Changes（见图14-101），默认新建一个Changelist，输入Name后单击【OK】按钮，之前搁置的文件被恢复，如图14-102所示。

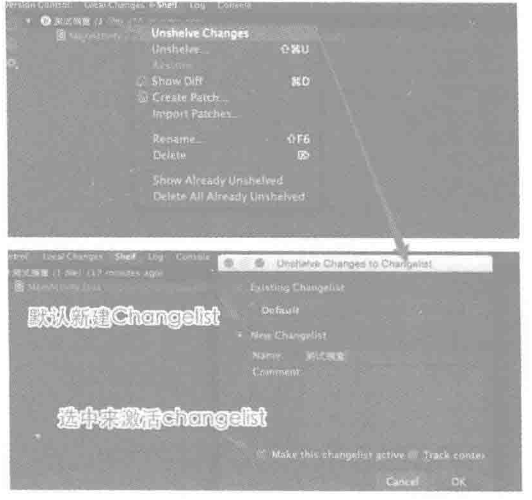


图 14-101

- Unshelve：默认恢复搁置到默认的变更列表。

操作步骤： Version Control 工具窗口→Shelf→右击之前搁置的文件→Unshelve→弹出恢复搁置的对话框→单击【OK】按钮后搁置恢复，如图 14-103 所示。

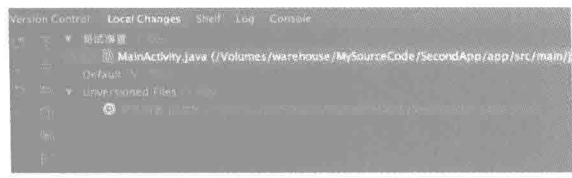


图 14-102

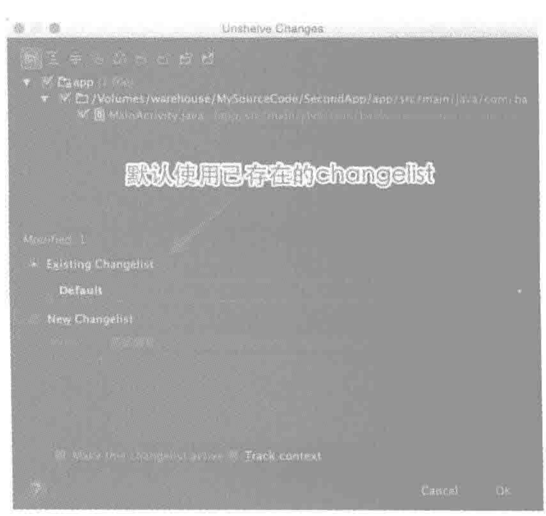


图 14-103

14.27 查看 Git 项目的提交信息

1. 查看本地Git项目提交信息

操作步骤： 菜单栏→VCS→Browse VCS Repository→Show Git Repository Log→弹出选择对话框，选择一个要浏览日志的项目。

显示结果：显示我们选择项目的提交日志，如图 14-104 所示。

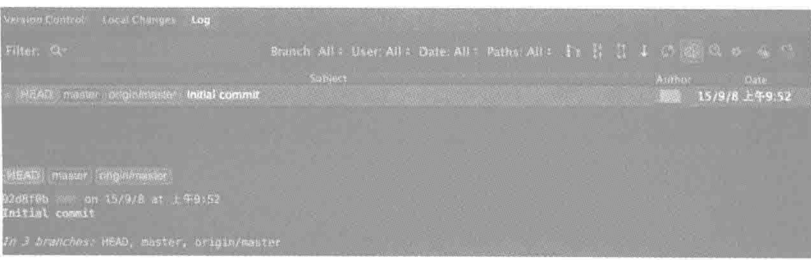


图 14-104

2. 查看当前项目的提交信息

在Version Control工具窗口的Log中查看当前项目的分支信息及提交信息，如图 14-105 所示。

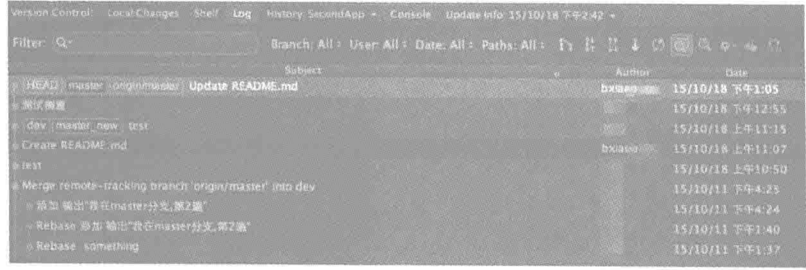


图 14-105

3. 过滤信息

- 过滤提交信息，如图 14-106 所示。

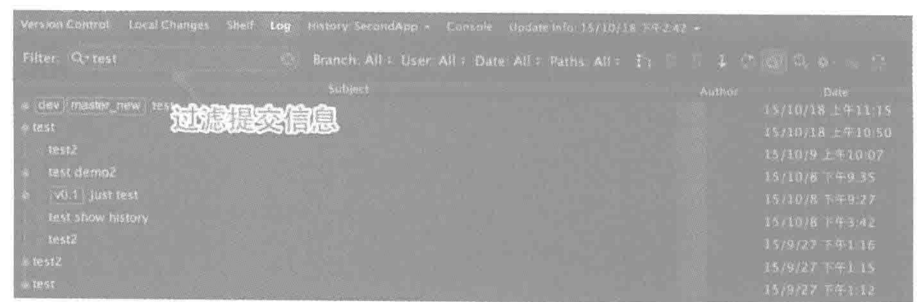


图 14-106

- 过滤分支/用户/日期/路径，如图 14-107 所示。

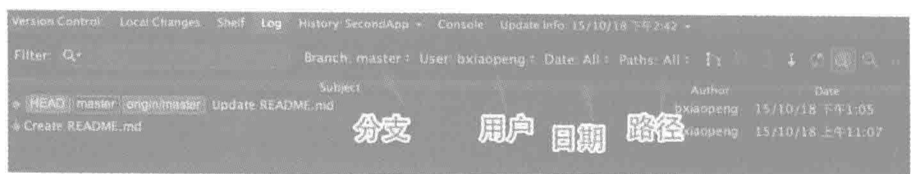


图 14-107

其他还有很多实用的功能，大家自己体验吧。

第 15 章　窗　口

Android Studio作为一个可视化的开发环境，界面布局由很多窗口组成。 为了有效利用屏幕的空间，这些窗口按照一定的规则或显示或隐藏。如果想玩转Android Studio，必须要了解这些窗口的显示规则以及操作方法。

本章将向大家介绍如何在Android Studio中管理窗口。

本章重要知识点 >>>>>>>>>>>

- 如何移动、调整、切换、显示和隐藏窗口；
- 如何操作和管理文件标签。

15.1　最小化和最大化窗口

1. 最小化窗口

菜单栏： Window→Minimize

快捷键： command + M（macOS）

2. 最大化窗口

菜单栏： Window→Zoom

快捷键： option + command + "="（macOS）

快捷键可能跟MAC系统快捷键冲突，如果常用这个功能，可以去偏好设置中将Zoom的快捷键（keymap）改掉，比如control + command + "="。

15.2　保存和恢复窗口布局

1. 保存当前布局为默认布局

当我们打开项目时，可以自己设定默认布局（见图 15-1）。

操作步骤： 菜单栏→Window→Store Current Layout as Default。

2. 恢复默认的布局

菜单栏： Window→ReStore Default Layout

快捷键： fn + Shift + F12（macOS，如果快捷键有冲突，自行修改）

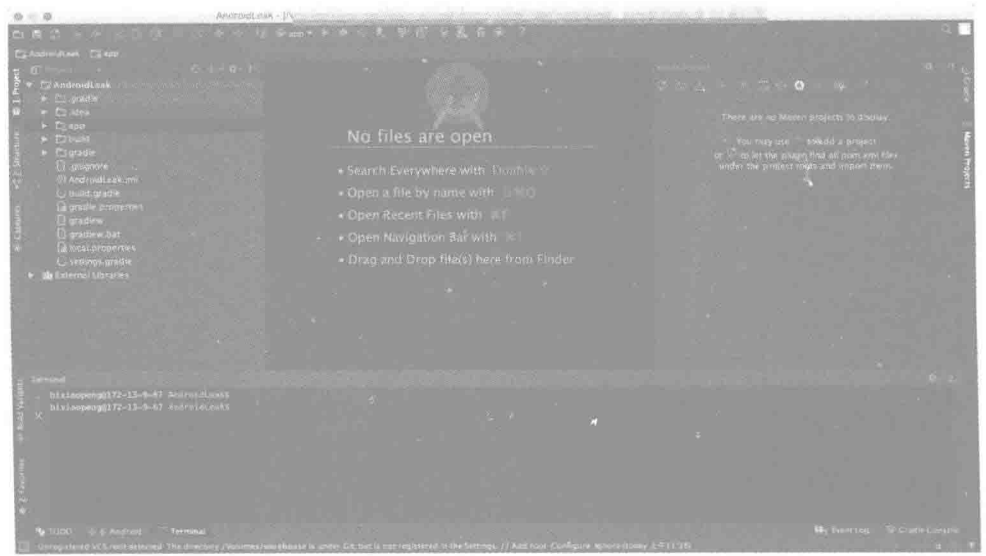


图 15-1

15.3　工具窗口的显示和隐藏

Android Studio中提供了很多工具，这些工具提供了不同的功能。有些工具窗口总是可见的，有些则需要被激活后才可见。Android Studio的工具条分布在主界面的左右两边和底部，如图 15-2 所示。

工具条上的工具可以通过拖动来移动位置，如图 15-3 所示。

图 15-2

图 15-3

每个工具条上只允许同时显示一个工具窗口，也就是说同一个工具条上的某个工具窗口被打开，前一个会自动隐藏。

通过Tool Windows显示/隐藏工具窗口

菜单栏：View→Tool Windows→显示Android Studio支持的所有工具列表，如图 15-4 所示。在这里我们可以选择显示或隐藏哪一个工具窗口。

通过工具条显示/隐藏工具窗口

每个工具窗口都可以通过工具条上的Tab来显示和隐藏，显示时颜色变深，隐藏时变浅，如图 15-5 所示。

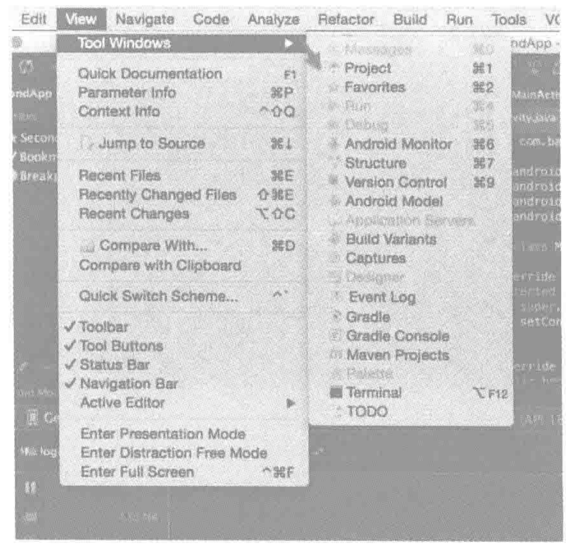


图 15-4

图 15-5

15.4 工具窗口的隐藏技巧

隐藏活动的工具窗口

当有多个窗口被打开的时候，只隐藏活动的那一个工具窗口。

菜单栏：Window→Active Tool Window→Hide Active Tool Window | Close Active Tab

如图 15-6 所示，Project和Android Monitor都被打开了，单击【Hide Active Tool Window】只会隐藏Project，因为当前活动的是Project。如果没有活动的工具窗口，此选项为不可点状态。

挨个隐藏工具窗口

当有多个窗口被打开的时候，使用此功能可以按打开的顺序逆向隐藏工具窗口。

菜单栏：Window→Active Tool Window→Hide side tool windows

隐藏所有工具窗口

当编辑器和工具窗口都打开的时候，界面会显得比较乱。如果我们只想专注地编写代码，就可以一键把所有的工具窗口隐藏。

菜单栏：Window→Active Tool Window→Hide All Windows

快捷键：fn + shift + command + F12（macOS）或者Ctrl + Shift + F12（Windows/Linux）

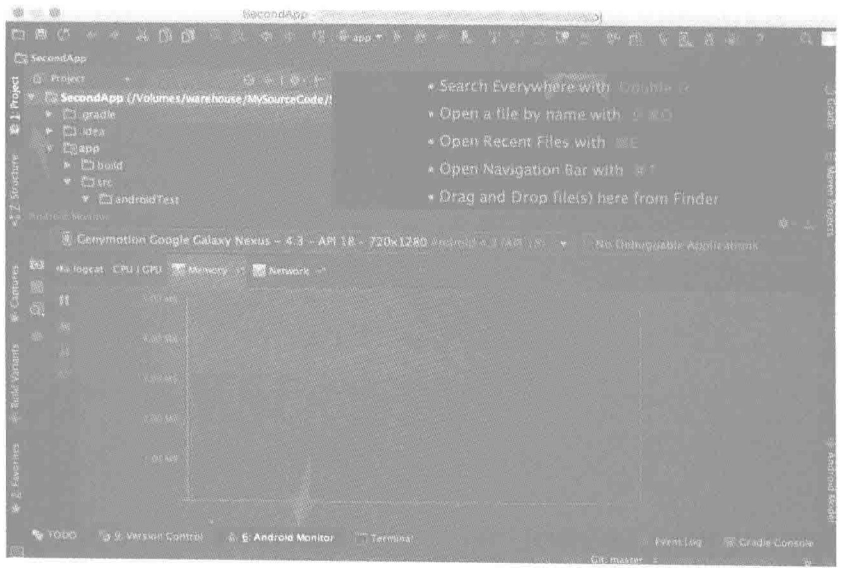


图 15-6

恢复隐藏的所有工具窗口

菜单栏：Window→Active Tool Window→Restore Windows
快捷键：fn + shift + command + F12（macOS）或者Ctrl + Shift + F12（Windows/Linux）

15.5 工具窗口调整技巧

跳转到上一个工具窗口

我们经常会在工具窗口和编辑器之间跳转，可以让光标快速跳转到上一个工具窗口。
菜单栏：Window→Active Tool Window→Jump to Last Tool Window
快捷键：fn + F12（macOS）或者F12（Windows/Linux）

最大化工具窗口（见图 15-7）

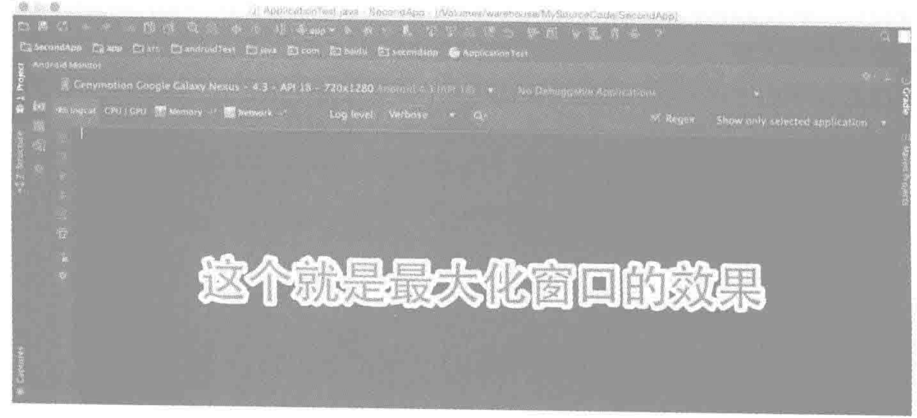


图 15-7

菜单栏：Window→Active Tool Window→Maximize tool window

快捷键：shift + command + " ' "

恢复最大化的工具窗口

最大化工具窗口后原来的【Maximize tool window】变成了【Restore tool window size】，单击后工具窗口可恢复到原来的大小。

逐渐调整工具窗口大小

最快的调整方式是使用鼠标长按拖动，如图 15-8 所示。

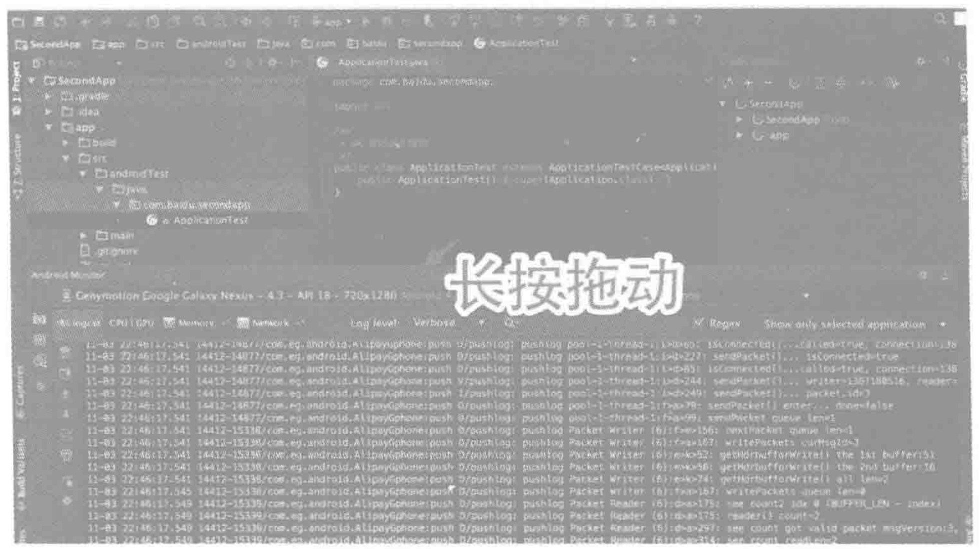


图 15-8

使用逐渐拉伸的方式来调整窗口时，方法如下：

- 工具窗口向上拉伸：激活工具窗口→菜单栏→Window→Active Tool Window→Resize→Stretch to Top，快捷键是 shift + command + ↑。
- 工具窗口向下拉伸：激活工具窗口→菜单栏→Window→Active Tool Window→Resize→Stretch to Bottom，快捷键是 shift + command + ↓。
- 工具窗口向右拉伸：激活工具窗口→菜单栏→Window→Active Tool Window→Resize→Stretch to Right，快捷键是 shift + command + →。
- 工具窗口向左拉伸：激活工具窗口→菜单栏→Window→Active Tool Window→Resize→Stretch to Left，快捷键是 shift + command + ←。

15.6 移动工具窗口的位置

我们可以把工具窗口上下左右移动，放到习惯的位置上。

操作步骤：右击工具窗口→Move to→选择要移动到的位置（见图 15-9），工具窗口移到上面，效果如图 15-10 所示。

图 15-9

图 15-10

15.7 工具窗口的查看模式

工具窗口提供了不同的查看模式，用于控制工具窗口的显示形式和表现方式，帮助使用者快速定位到工具窗口或者最大化编辑区域。

1. 停驻和浮动模式

停驻模式：当Floating Mode没有被选中时，工具窗口就处于停驻模式。停驻模式让工具窗口停驻在工具栏上，如图 15-11 所示。

当窗口是固定模式时，若同时还是脱开模式（Docked Mode未被选中），则窗口会占满所附着的工具窗口条的长度或高度（取决于工具窗口条是水平的还是垂直的）。效果如图 15-12 所示。

浮动模式：工具窗口处于Floating Mode时，工具窗口处于浮动模式。浮动模式可以让工具窗口停放在屏幕上的任意位置，如图 15-13 所示。

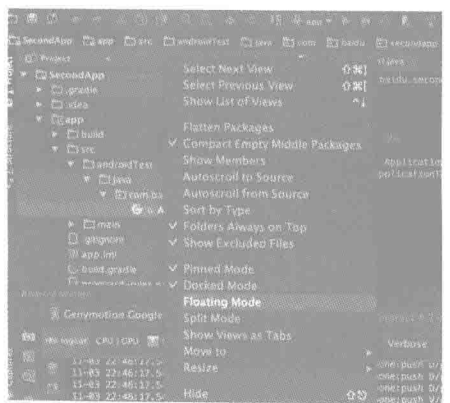


图 15-11

图 15-12

图 15-13

2. 停靠和脱开模式

工具窗口处于停靠模式时，这个窗口相邻的界面元素将环绕着该窗口。调整该窗口的大小会自适应调整相邻窗口的大小，如图 15-14 所示。

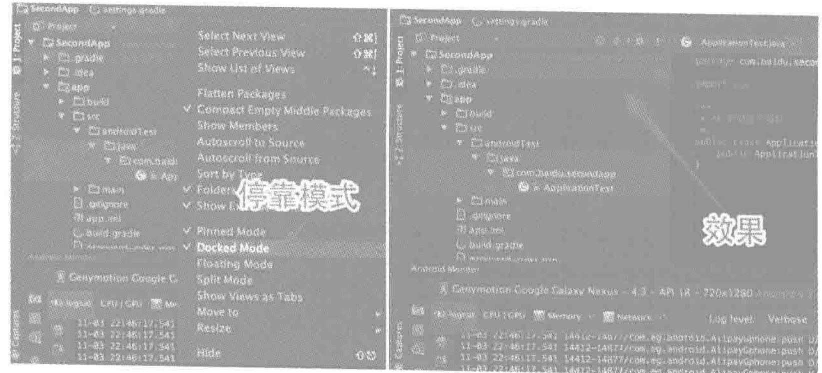


图 15-14

工具窗口处于脱开模式（未选中Docked Mode）时，该窗口会变成最上层的界面元素，盖住其他的元素与其相交的部分，重设脱开模式的窗口大小不会自适应地调整其他元素的大小，如图 15-15 所示。脱开模式的工具窗口变为不活动状态时，它会自动隐藏。

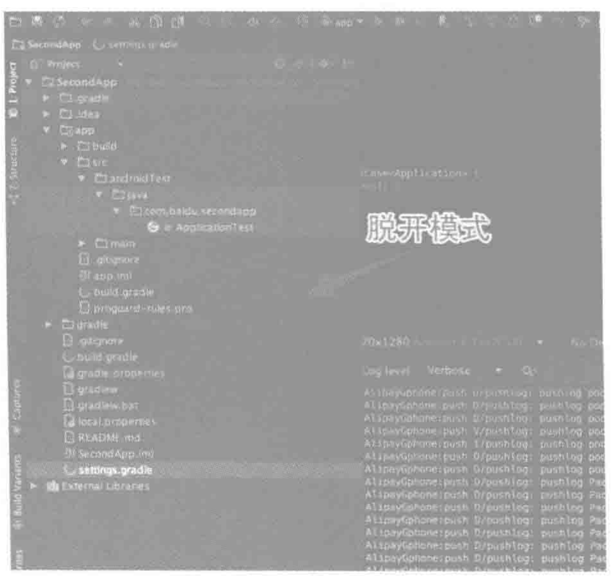


图 15-15

3. 固定与非固定模式

当选中Pinned Mode（固定模式）时，工具窗口变成不活跃状态时是可见的，若没有选中，工具窗口就会自动隐藏。

当然根据其他模式设定会有一些例外情况：选中Docked Mode的窗口不活跃时，总是被隐藏；选中Floating Mode的窗口不活跃时，将变成半透明。

4. 分离模式

当选中分离模式时，在同一个工具条上最多可以同时显示两个工具窗口，如图 15-16 所示。

图 15-16

5. 作为标签显示

选中【Show View as Tabs】模式时，以标签的形式显示多个内容层，如图 15-17 所示。未选中【Show View as Tabs】模式时，以下拉列表的形式显示多个内容层，如图 15-18 所示。

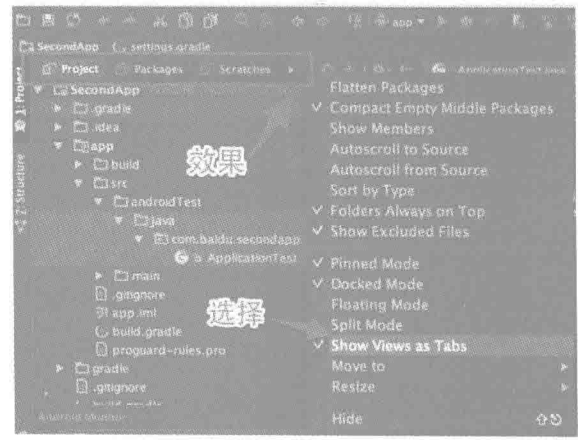


图 15-17

图 15-18

15.8 编辑器标签设置

任一时刻打开一个文件都会以标签页的形式在编辑窗口中出现。

该标签页出现在当前活动的标签页的下个位置，并成为新的活动标签页，当编辑窗口的标签页达到上限时，就会根据标签页关闭的优先策略来关闭一些标签页。当关闭活动的标签页时，会根据活动标签页选取策略来选择下一个活动标签页。

设置编辑器标签：偏好设置→Editor→Editor Tabs，如图 15-19 所示。

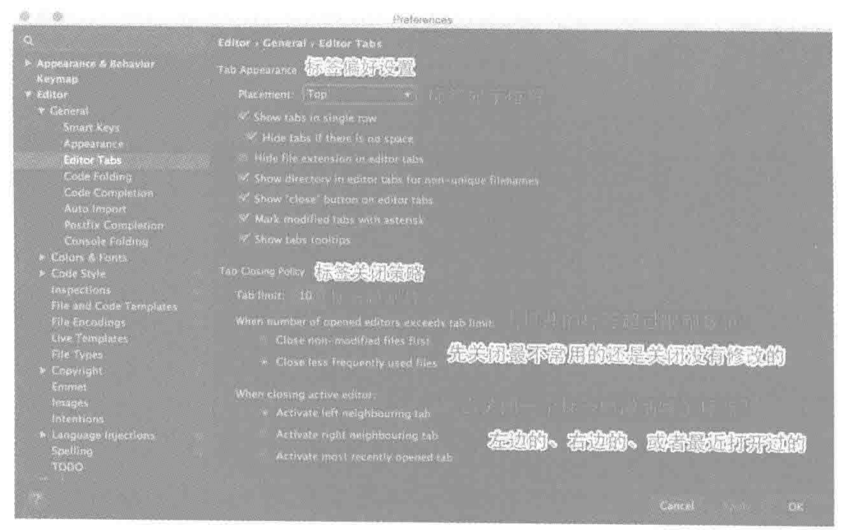


图 15-19

通常情况下我们使用默认的设置就好了。

15.9　快速切换编辑器标签

1. 上一个/下一个切换标签

当在编辑器中打开多个标签时，可以使用上一个/下一个来切换标签。

前提条件： 光标在编辑器中。

菜单栏： Window→Editor Tabs→Select Next Tab 或 Select Previous Tab

快捷键： shift + command + [和shift + command +]（macOS）

通常这种操作使用快捷键更方便。

2. 快速切换已打开的文件

当我们在编辑窗口打开多个文件（见图 15-20）以后，想快速在这些文件中切换时，可以使用快捷键control + tab调出switcher（切换对话框，见图 15-21）。

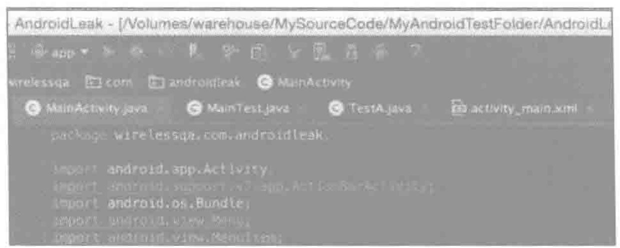


图 15-20

如图 15-21 所示，右边会显示当前编辑器窗口已打开的文件，此时按住control键，通过按tab键来进行切换。当切换到最后一个文件时，焦点会跳转到左边的工具列表，选中后会打开相应的工具窗口。

3. 快速切换工具窗口

使用快捷键control + tab调出switcher（切换对话框），默认选中的是文件，如果想默认选中的是工具，则使用快捷键control + shift +tab。长按control + shift ，通过按tab来切换工具，如图 15-22 所示。

图 15-21

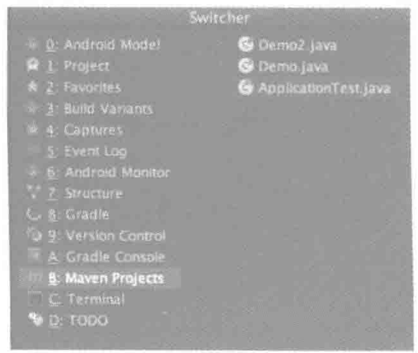


图 15-22

15.10　关闭编辑器标签

关闭当前打开的标签

快捷键： command + w

鼠标： 长按shift + 鼠标单击标签的任意位置，或右击文件标签→Close（见图 15-23），或直接用鼠标单击标签右上角的关闭按钮。

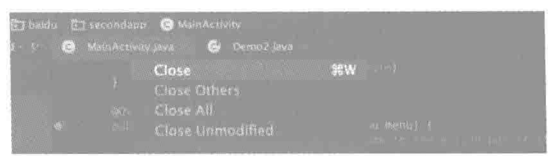


图 15-23

关闭除了当前文件以外的所有文件

方法一： 右击当前打开的标签，选择【Close Others】，如图 15-24 所示。

方法二： 按住Alt，然后单击当前文件标签右上角的关闭按钮，如图 15-25 所示。

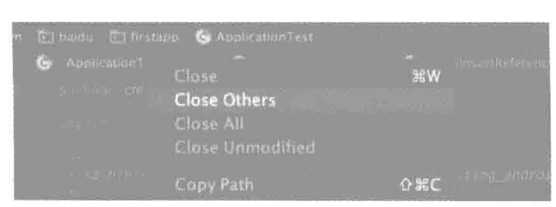


图 15-24

图 15-25

关闭所有标签

操作步骤： 右击文件标签→Close All。

关闭未修改的标签

操作步骤： 右击文件标签→Close Unmodified。

关闭除了固定标签外的所有标签

如果你固定了某个标签，那么在操作列表里会激活【Close All but Pinned】功能，单击此功能会关闭除了固定标签外的所有标签，如图 15-26 所示。

被固定标签页不会被编辑窗口自动关闭。

图 15-26

15.11　管理编辑器标签

重新打开关闭的标签

重新打开关闭的标签会按照关闭顺序的逆序来打开标签，也就是说最后关闭的标签会被最先打开。

操作步骤：右击编辑器标签→Reopen Closed Tab→最后被关闭的标签会被打开。

在最后打开新标签

操作步骤：右击编辑器标签→选中Open New Tabs At The End，再打开新的文件标签时就会在已打开的标签最后打开。

通过文件名排序标签

操作步骤：右击编辑器标签→Sort Tabs By Filename。

15.12　标签显示位置

默认标签是显示在编辑器窗口最上面的，我们也可以自定义显示的位置。

操作步骤：右击标签→Tabs Placement→选择显示位置（见图 15-27 所示）。各显示位置效果如图 15-28~图 15-31 所示。

图 15-27

图 15-28

图 15-29

图 15-30

图 15-31

15.13 拆分编辑器窗口

Android Studio支持编辑窗口分离，满足同时查看一个文件不同部分的要求。

拆分一个标签窗口

方法一：右击标签→Split Vertically（垂直拆分） 或 Split Horizontal（水平拆分），如图15-32 所示。

方法二：右击标签→Move Right 或 Move Down。相当于水平拆分和垂直拆分，不同点是Move会把当前的标签页移到拆分的一边，而Split不会。

改变拆分窗口的摆放方式

操作步骤：右键标签→Change Splitter Orientation（见图 15-33），改变拆分窗口的摆放方式，也就是水平拆分的变成垂直拆分，垂直拆分的变成水平拆分。

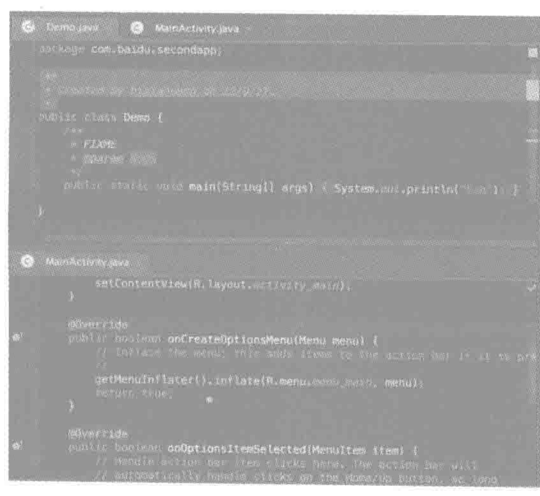


图 15-32

图 15-33

标签组

Android Studio可以通过将标签页分组的方式支持多文件同时编辑，组的数量没有限制。

创建一个标签页组只需要简单地将一个标签页分离，就可以创建一个新的标签页组了。要将一个标签页移动到另外一个分组，需要选择移动的标签页，右击打开上下文菜单，选择 Move to Opposite Group。如果想把当前文件在另外一个标签组打开可以选择Open In Opposite Group。

当然我们还可以通过拖动的方式来分组，如图 15-34 所示。

图 15-34

撤销拆分

操作步骤：右击标签页→Unsplit（撤销当前活动的标签页的拆分窗口） 或Unsplit All（撤销全部标签页的拆分窗口）。

15.14 多个项目之间切换

当我们同时打开多个Android Studio项目时，想要快速切换来查看不同的项目，就一定会用到这个功能。

打开后面一个项目：菜单栏→Window→Next Project Window或使用快捷键command + `。

打开前面一个项目：菜单栏→Window→Next Project Window或使用快捷键shift + command + `。

如果当前我们只打开了一个项目，那么上面两个功能是不可用的。

选择某个已打开项目：已经打开的Android Studio会在Window中显示出来，可以通过菜单→Window子菜单来完成，如图 15-35 所示。

图 15-35

第 16 章　偏好设置

Android Studio几乎所有的功能都是可以配置的，这给了我们很大的自由定制空间。如果我们想与众不同，想向高手学习一些好的开发习惯，就需要了解Android Studio的偏好设置，需要了解如何配置一个个性化、高效、符合自己开发习惯的IDE。

在前面的文章当中我们已经介绍了很多配置的技巧，本章只会挑一些常用的介绍给大家。

本章重要知识点 >>>>>>>>>>

- 如何设置 IDE、编辑器的外观和行为；
- 如何设置系统的一些常用属性和快捷键；
- 如果设置自动补全和代码导入；
- 如何设置文件和代码模板；
- 如何管理插件，常用的插件有哪些；
- 如何优化编译和构建性能。

16.1　外观与行为

16.1.1　设置工具提示的延迟时间

当我们把鼠标悬停在工具栏上的工具或编辑器中的项目时会出现提示，如图 16-1 所示。

图 16-1

鼠标悬停跟显示提示之间的时间称为工具提示的延迟时间，这是可以在偏好设置中设置的。

操作步骤：偏好设置→Appearance & Behavior→Appearance→滑动设置【Tooltip initial delay】，如图 16-2 所示。

延迟时间越短，工具提示显示得越快。但是这里会有一个问题，如果时间太短，当你调试的时候，当鼠标滑过时会到处显示提示，影响调试效率。所以这个时间最好使用默认值。

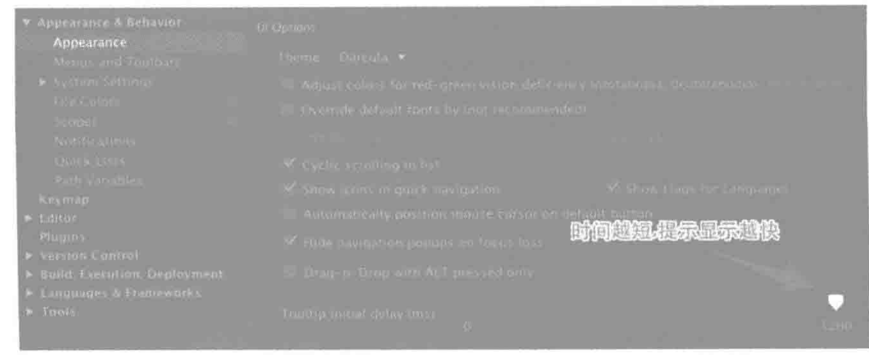


图 16-2

16.1.2 设置在状态栏显示内存状态

默认状态栏是不显示内存使用状态的，如果想实时显示内存的使用情况，可以设置在状态栏显示内存状态。

操作步骤：偏好设置→Appearance & Behavior→Appearance→勾选【Show memory indicator】，效果如图 16-3 所示。

图 16-3

16.1.3 对菜单选项和工具栏的工具进行增/删改

操作步骤：偏好设置→Appearance & Behavior→Menus and Toolbars→选择需要操作的选项，如展开Main Toolbar→对工具栏上的工具进行增删改，如图 16-4 所示。

图 16-4

16.2 系统设置

1. 设置启动Android Studio时是否自动打开项目

当我们打开Android Studio时默认会打开上次最后关闭的项目，如果不想自动打开上次关闭的项目，可以在偏好设置中选择Appearance & Behavior→System Settings→不勾选【Reopen last project on startup】。

2. 设置退出Android Studio时是否弹出确认提示

当我们关闭Android Studio时默认会弹出确认提示，如果不想每次都确认，可以在偏好设置中选择Appearance & Behavior→System Settings→不勾选【Confirm application exit】。

3. 设置打开一个项目时的打开方式

当我们打开一个新项目时默认会弹出一个确认提示，让我们确认是在新窗口打开还是在当前窗口打开项目，如果想改变默认设置，可以在偏好设置中选择Appearance & Behavior→System Settings→设置Project Opening，如图 16-5 所示。

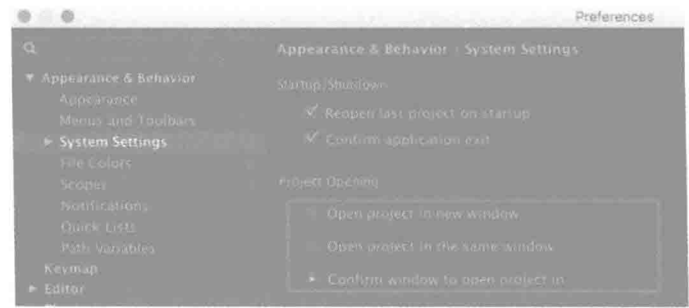


图 16-5

- Open project in new window：在一个新窗口打开项目。
- Open project in the same window：在同一个窗口打开项目（原来的项目会被关闭）。
- Confirm window to open project in：弹出确认对话框，可选择在新窗口或原窗口打开。

4. 设置文件定期自动保存

如果我们怕文件丢失，想每隔一段时间自动保存，可以在偏好设置中选择Appearance & Behavior→System Settings→勾选【Save files automatically if application is idle for [] sec.】。

如图 16-6 所示，默认自动保存的频率是 15 秒，当然时间可以自定义。

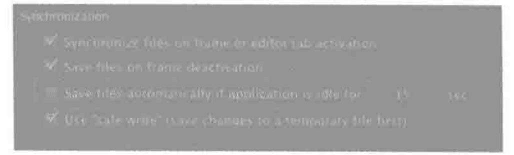


图 16-6

5. 设置编辑窗口失活时文件自动保存

编辑窗口失活的意思可理解为从编辑窗口切换到其他窗口。要让文件在编辑失活时自动保存，可以在偏好设置中选择Appearance & Behavior→System Settings→勾选【Save files on frame deactivation】（见图 16-6）。

6. 设置密码存储策略

Android Studio中的存储策略是指当我们使用密码保护的资源时（如版本控制时需要输入用户名和密码）是否记住或保存我们输入的密码，以便下次使用时不必再次输入。如果要保存，密码会被保存在Android Studio的数据库中。出于安全考虑，需要设置master password。

Android Studio中有以下 3 种密码存储策略。

- Do not remember passwords：不记住密码，每次都要输入密码。
- Remember passwords until the application is closed：记住密码，直到应用程序被关闭。
- Save on disk with master password protection：密码保存到磁盘，使用 master password 来保护。

操作步骤：偏好设置→Appearance & Behavior→System Settings→Passwords→设置 Password storage policy，如图 16-7 所示。

记住所设置的Master Password，如果忘记了也没关系，可以重置：单击Master Password，在弹出的Change Master Password窗口中单击Reset...，如图 16-8 所示。

图 16-7

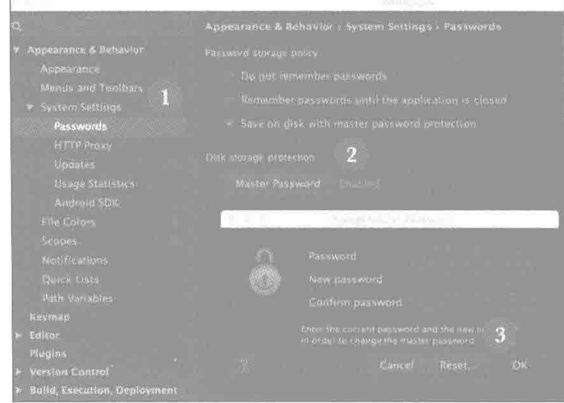


图 16-9

然后根据提示一步一步重设密码就可以了。

7. 设置是否让Google统计使用信息

操作步骤：偏好设置→Appearance & Behavior→System Settings→Usage Statistics→Send usage statistics to Google，如图 16-9 所示。

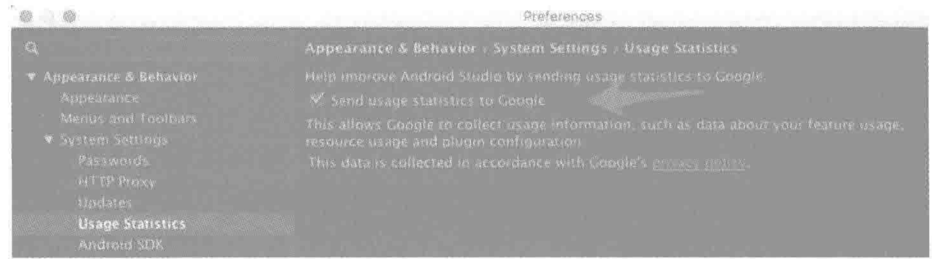


图 16-9

此功能统计Android Studio的使用情况，然后发送给Google，帮助Google来改善Android Studio。勾选Send usage statistics to Google将会允许Google收集使用信息，包括功能、资源的使用情况和插件的配置信息等。这些数据会按照Google隐私策略来收集，如果不想让Google收集使用数据，就不勾选Send usage statistics to Google。

16.3 键盘映射

1. 在Android Studio中如何使用Eclipse的快捷键

很多同学从Eclipse切换到Android Studio会很不习惯，因为常用的快捷键没有了。但是Android Studio提供了一个可以将Eclipse的快捷键迁移过来的功能，操作如图 16-10 所示。

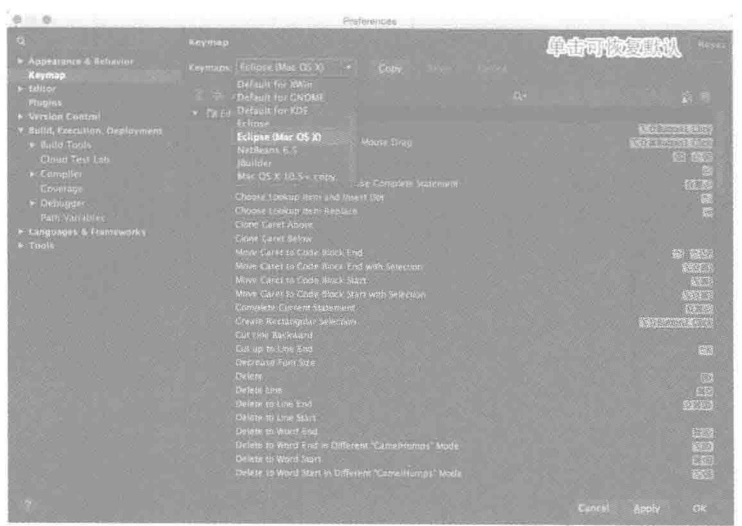


图 16-10

提 示

当你改变当前的 Keymaps 时，右上角会出现一个 Reset 按钮，单击后可以恢复修改之前的设置。

2. 如何自定义一份属于自己的keymap

每个人都有自己的习惯，对快捷键也是一样。如果想自定义一份属于自己的keymap，最好不要直接在原来的keymap上修改，建议先复制一份，在复制的keymap中修改，如图 16-11 所示。如果不想要了，单击【Delete】按钮删除就可以了。

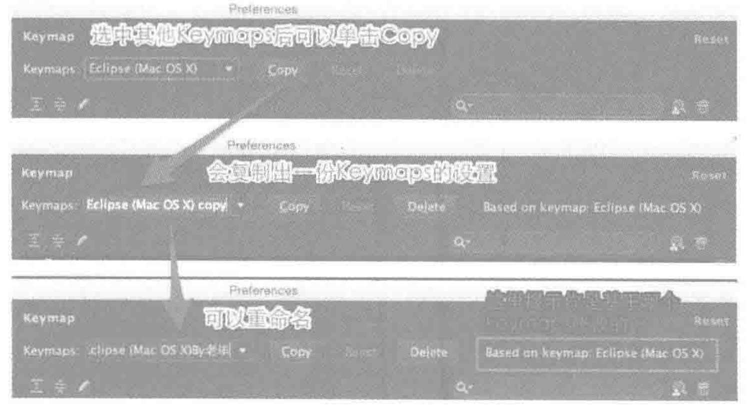


图 16-11

16.4 编辑器常规设置

16.4.1 设置单击编辑器光标定位在一行的结尾或定位在单击的位置

通常在一行文字的空白位置单击鼠标时，光标会定位到这一行的最后一个字符的结尾，但是使用Android Studio光标会定位在你单击的位置，如果不习惯，可以设置一下。

操作步骤: 偏好设置→Editor→General→Virtual Space→勾选【Allow placement of caret after end of line】，如图 16-12 所示。

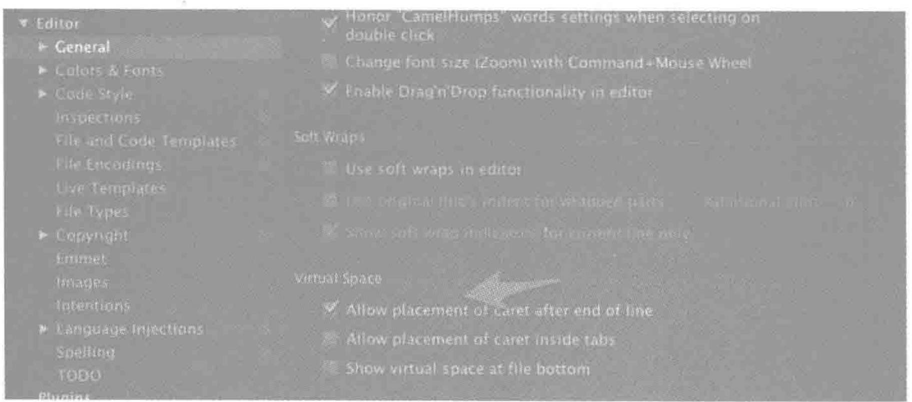


图 16-12

勾选【Allow placement of caret after end of line】，在空白位置单击，光标会定位在一行的结尾；不勾选【Allow placement of caret after end of line】，在空白位置单击，光标会定位在单击的地方，如图 16-13 所示。

图 16-13

16.4.2 设置鼠标悬停在元素上会显示文档提示

默认情况下想查看某个变量、类或方法的文档时，需要使用fn+F1 快捷键（macOS）才能查看。但是如果想让鼠标悬停在某个变量、类或方法上就直接显示它的文档，可以在偏好设置中选择Editor→General→Other→勾选【Show quick documentation on mouse move】，如图 16-14 所示。效果如图 16-15 所示。

图 16-14

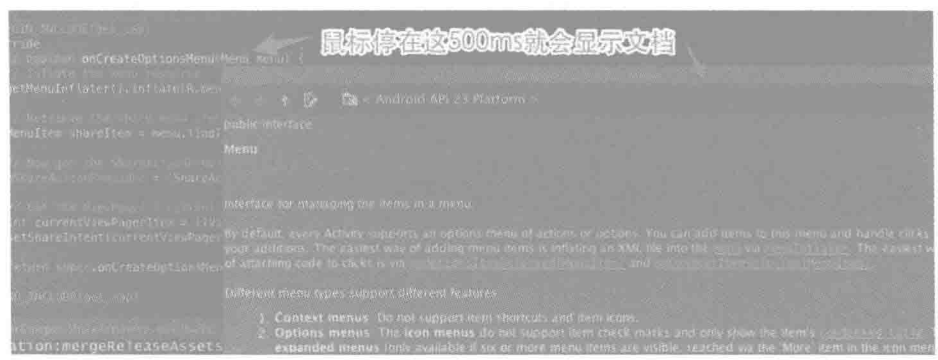


图 16-15

16.4.3 设置是否自动换行

当一行内容超过编辑器的宽度时，想查看这一行最后边的内容将非常麻烦，如图 16-16 所示。这时自动换行的好处就来了，当一行的内容超过编辑器的宽度时，它会自动换行，如图 16-17 所示。

图 16-16

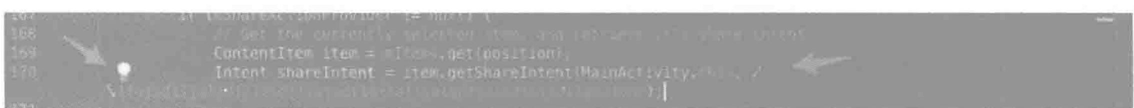


图 16-17

操作步骤：偏好设置→Editor→单击General→查看 Soft Wraps→勾选 【Use soft wraps in editor】，如图 16-18 所示。

图 16-18

让换行符一直显示

默认【show soft wrap indicators for current line only】会被勾选，意思是仅在你正在编辑的当前行显示换行符，如果离开了当前行，换行符就不显示了。

如果想显示所有的换行符，就不要勾选【show soft wrap indicators for current line only】。有换行符的效果如图 16-19 所示。

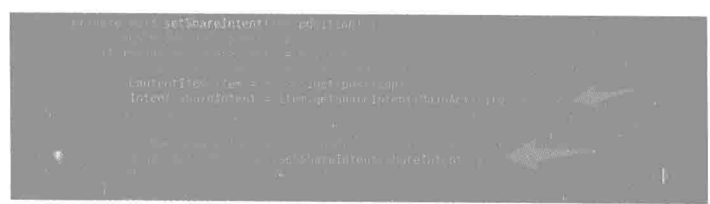


图 16-19

设置换行后的缩进字符

默认换行后第二行的缩进是从第 0 个字符开始的，如图 16-20 所示。

图 16-20

如果想自定义换行后第二行的缩进，可以勾选【Use original line’s indent for wrapped parts】→设置 Additional shift（这里的数字就是换行后第二行缩进的次数）。这里设置换行后第二行缩进从第 20 个字符开始，如图 16-21 所示。效果如图 16-22 所示。

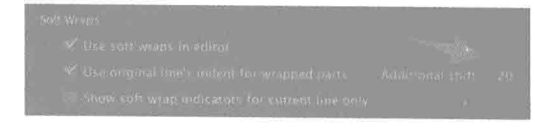


图 16-21

图 16-22

16.4.4 设置使用command+鼠标控制代码的缩放

默认情况下我们是无法使用鼠标控制编辑器中代码缩放的，不过在Android Studio中可以设置使用command（macOS，Windows是Ctrl）+鼠标来控制代码的缩放。

操作步骤：偏好设置→Editor→General→勾选【Change font size（Zoom）with Command + Mouse Wheel】，如图 16-23 所示。

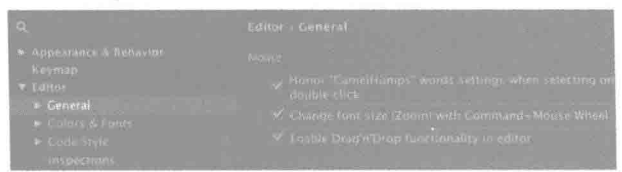


图 16-23

提 示

缩放只针对编辑器中当前打开的文件有效，如果关闭后重新打开，就会还原大小。

16.4.5 开启使用驼峰单词

默认情况下，使用驼峰选择单词功能是没有开启的。当我们使用快捷键option + ←/→（macOS）或者Ctrl + ←/→（Windows/Linux）移动光标时，或使用快捷键option + shift + ←/→（macOS）或者Ctrl + Shift + ←/→（Windows/Linux）选择单词时，光标是一个单词一个单词选中的。如图 16-24 所示，框里就会是被选中的一个个单词，即使有驼峰命名的单词也不会被区分。

开启使用驼峰选择单词功能后，选择的效果如图 16-25 所示。

图 16-24　　图 16-25

移动光标或选择单词时识别了驼峰，有这个操作习惯的朋友，可以开启这个功能。

操作步骤：偏好设置→Editor→General→Smart Keys→勾选【Use "CamelHumps" words】，如图 16-26 所示。

这样做的后果是双击驼峰命名的单词时，这个单词也不会全部被选中，可以通过偏好设置→Editor→General→取消勾选【Honor "CamelHumps" words settings when selecting on double click】（见图 16-27）来规避这个问题。这样一来，双击选中单词又跟原来一样了，不受驼峰单词的影响。

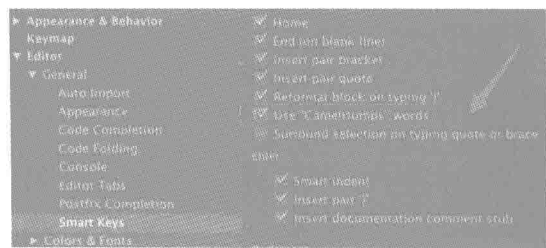


图 16-26

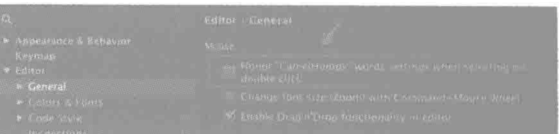


图 16-27

16.4.6　关闭单词拼写检查

在Android Studio的编辑器中，我们最常见的警告就是单词拼写检查错误提示，如图 16-28 所示。关掉这个提示的操作步骤是偏好设置→Editor→Inspections→取消勾选【Spelling】，如图 16-29 所示。

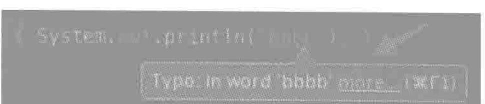


图 16-28

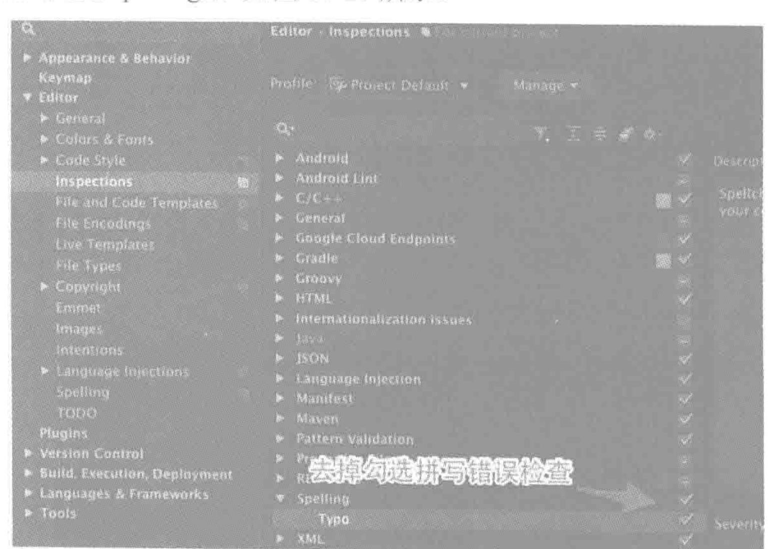


图 16-29

16.4.7　设置代码折叠规则

Android Studio默认有很多代码折叠的规则，如图 16-30 所示。

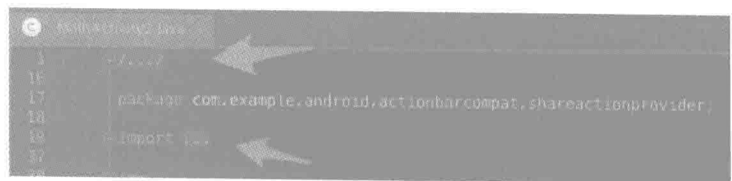


图 16-30

想自定义代码折叠的规则时，可以通过偏好设置→Editor→General→Code Folding来完成。如图 16-31 所示，已勾选的规则就是Android Studio中默认的折叠规则。

图 16-31

16.5 设置自动导入

16.5.1 设置粘贴时自动导入包

从其他地方复制一段代码粘贴到Android Studio文件中时，默认Android Studio会询问你是否导入引用的包，如果想在粘贴时自动导入包，可以通过偏好设置→Editor→General→Auto Import→Insert imports on paste来完成，如图 16-32 所示。

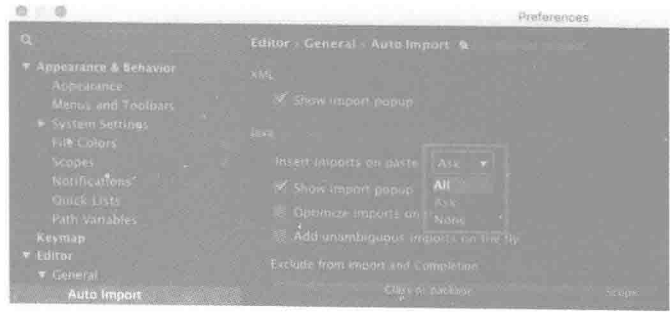


图 16-32

- All：全部自动导入。
- Ask：询问是否自动导入。
- None：不自动导入。

同时还需要勾选【Add unambiguous imports on the fly】（动态添加明确的导入）。

16.5.2 设置自动导入需要的包

Android Studio引用一些类的时候不会自动导入包，如果想自动导入需要的包可以进行如下操作。

操作步骤：偏好设置→Editor→General→Auto Import，勾选【Optimize imports on the fly】和【Add unambiguous imports on the fly】，如图 16-33 所示。

图 16-33

16.5.3 设置是否弹出导入提示

Android Studio默认是会显示导入提示的，如图 16-34 所示。

当你输入的类的声明没有被导入时，会弹出一个导入提示，显示导入的快捷键，让你可以快速导入。这是一个非常方便的功能，当然所有的功能都支持自己配置，毕竟每个人的喜好和习惯都不尽相同。

图 16-34

操作步骤：偏好设置→Editor→General→Auto Import→Show import popup（见图 16-35）。这里提供了XML、Java、C/C++是否弹出导入提示的开关。

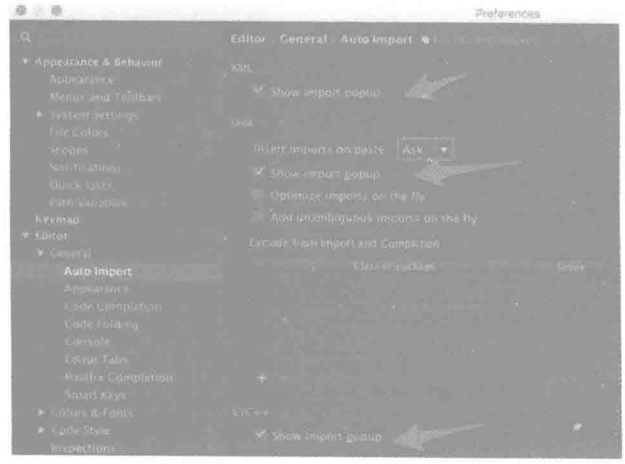


图 16-35

16.6 编辑器外观

16.6.1 设置编辑器一直显示行号

默认编辑器界面是不显示行号的，但可以设置编辑器显示行号（见图 16-36）。

操作步骤：偏好设置→Editor→General→ Appearance→勾选【Show line number】（见图 16-37）。

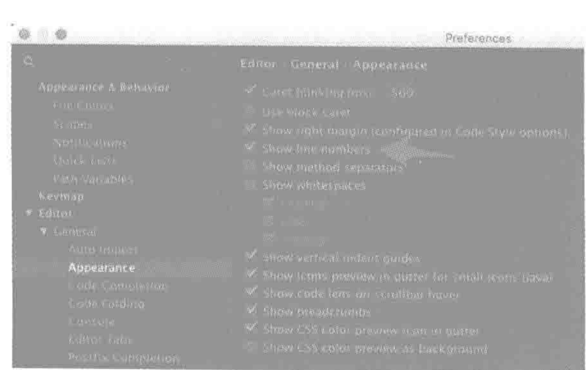


图 16-37

图 16-36

16.6.2 设置编辑器显示方法分隔符

默认编辑器界面是不会显示方法分隔符的，如果你想让每个方法区分得更明显，可以显示方法分隔符。

操作步骤：偏好设置→Editor→General→Appearance→勾选【Show method separators】。效果如图 16-38 所示。

图 16-38

16.6.3 设置编辑器显示空格

默认编辑器界面是不会显示空格的，如果你想知道代码中哪里有空格，就可以设置编辑器显示空格。

操作步骤：偏好设置→Editor→General→ Appearance→勾选【Show whitespace】（见图 16-39）。默认会勾选Leading、Inner、Trailing，也就是显示一行当中的开头、中间和结尾的空格，如果只想显示某一种可以自定义选择。效果如图 16-40 所示。（一个点表示一个空格。）

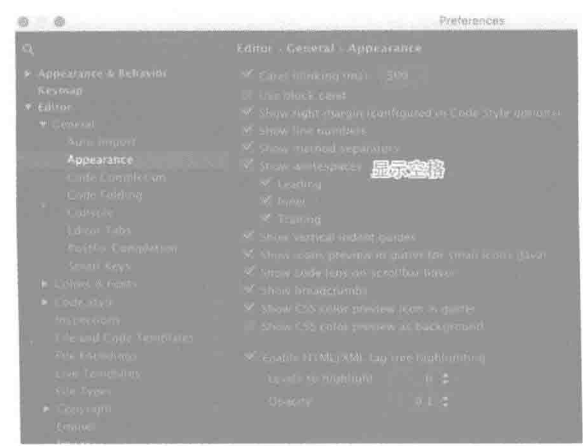


图 16-39

图 16-40

16.6.4 设置编辑器显示缩进向导

缩进向导可用图 16-41 来说明。显示了缩进向导以后，我们能很清晰地看到文件中的缩进关系，从而可以更好地控制每一行的缩进。

设置缩进导向是否显示

操作步骤： 偏好设置→Editor→ General→Appearance→勾选【Show vertical indent guides】（见图 16-42）。

图 16-41

图 16-42

16.7 代码补全

16.7.1 设置自动补全时是否区分大小写

默认自动补全时是区分首字母大小写的，如果不想区分大小写或区分全部大小写可以通过偏好设置→Editor→General→Code Completion（见图 16-43）来完成。

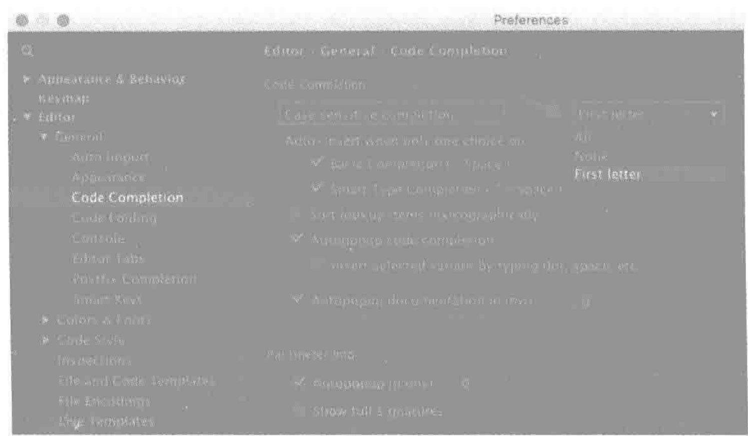


图 16-43

Case sensitive completion中有如下 3 个选项。

- First letter：区分首字母大小写。
- All：区分全部大小写。
- None：不区分大小写。

效果有什么不同?

假设我们想通过自动补全输入ArrayList。如果是区分首字母大小写或区分全部大小写，那么当我们输入array时是不会有任何补全提示的，如图 16-44 所示。如果不区分大小写，当我们输入array时就会有补全提示，如图 16-45 所示。

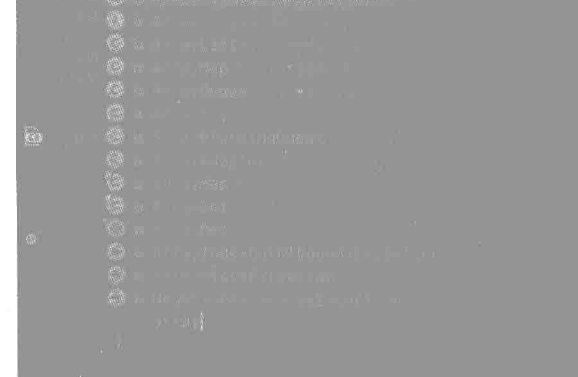


图 16-44

图 16-45

16.7.2 加快自动弹出代码补全提示的速度

Android Studio中弹出代码补全提示的速度默认是 1000ms，如果嫌慢，可以自定义提示的速度。操作步骤如图 16-46 所示。

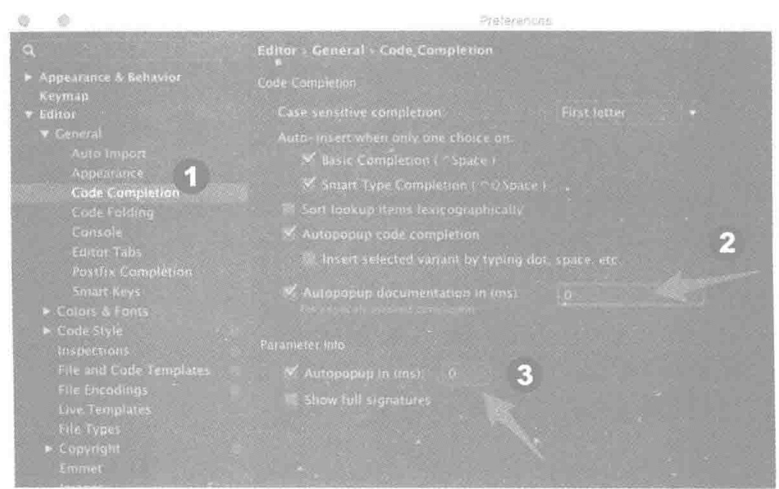


图 16-46

01 偏好设置→Editor →General → Code Completion。

02 在 Code Completion 中将【Autopopup documentation in（ms)】设置为 0（值越小弹出提示的速度越快)。

03 在 Parameter Info 中将【Autopopup in(ms)】设置为 0（值越小弹出提示的速度越快)。

16.7.3 关闭自动弹出代码补全提示

如果代码补全的提示完全没有必要，也可以将其关闭。

操作步骤：

01 偏好设置→Editor→General→Code Completion。

02 在 Code Completion 中不勾选【Autopopup documentation in（ms)】。

03 在 Parameter Info 中不勾选【Autopopup in（ms)】。

16.7.4 设置查看方法参数信息的时候显示方法签名

Android Studio默认只显示参数的提示信息，可以通过偏好设置→Editor→General→Code Completion→Show full signatures（见图 16-47）查看参数信息的时候显示整个方法的签名信息（见图 16-48）。

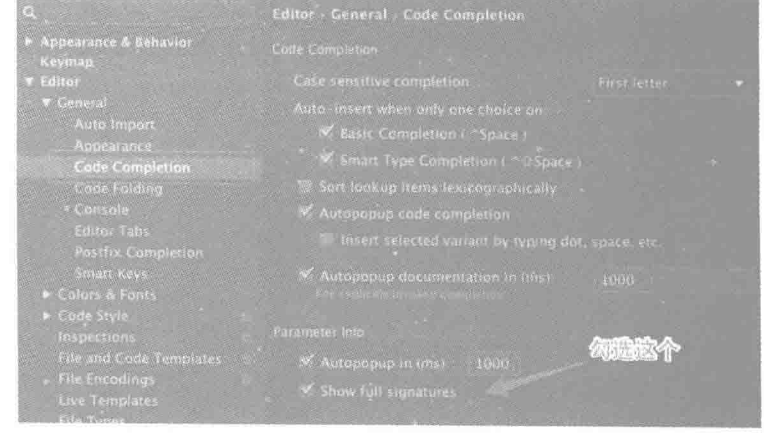


图 16-47

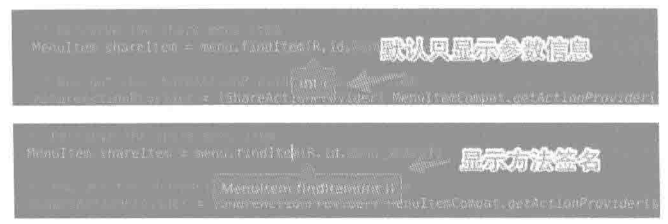


图 16-48

16.8 文件标签

16.8.1 设置用星号标记修改过的文件标签

操作步骤：偏好设置→Editor→General→Editor Tabs→勾选【Mark modified tabs with asterisk】，如图 16-49 所示。效果如图 16-50 所示。

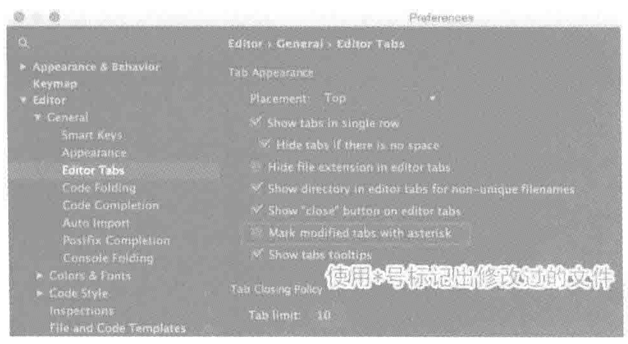


图 16-49

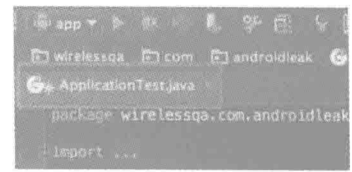


图 16-50

16.8.2 设置打开的文件标签可以多行显示

默认情况下新打开一个文件编辑器窗口就会新打开一个文件标签。标签是单行显示的，如果一行显示不下了就会被隐藏，如图 16-51 所示。

图 16-51

在偏好设置中选择 Editor→General→Editor Tabs→取消勾选【Show tabs in single row】（见图 16-52），即可让标签不隐藏，并且全部显示出来，修改后的效果如图 16-53 所示。（标签没有被隐藏，而是多行显示。）

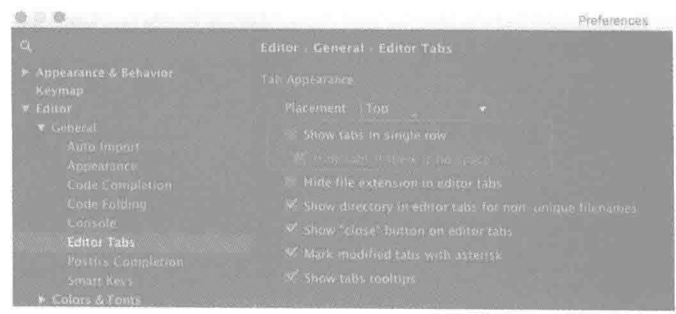


图 16-52

图 16-53

16.8.3 设置文件标签的显示位置

默认文件标签显示在编辑器的上方，如果想换个位置显示也是可以的。

操作步骤： 偏好设置→Editor→General→Editor Tabs→Placement→选择显示位置，如图 16-54 所示。

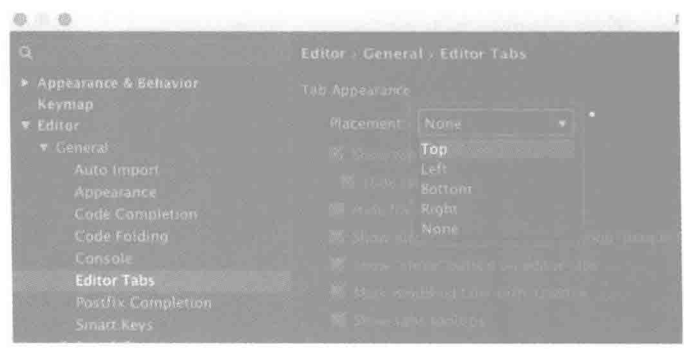


图 16-54

16.8.4 设置文件标签超过一定数量时的关闭规则

默认设置如图 16-55 所示。

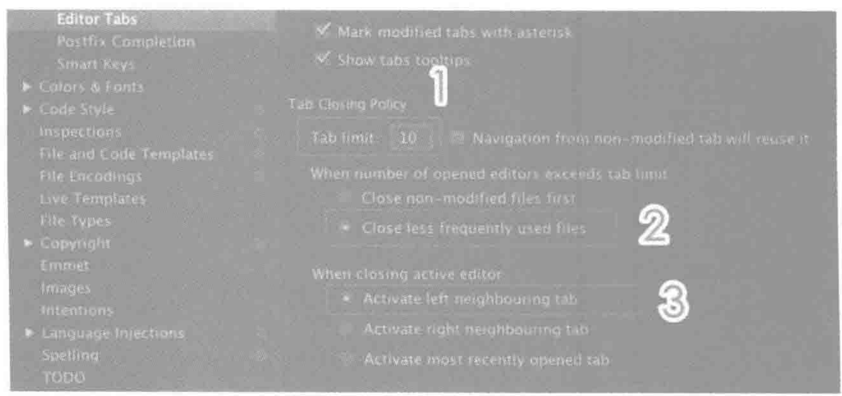


图 16-55

❶ 打开文件标签限制数量是 10。

❷ 当打开的文件标签超过 10 个时关闭使用频率较少的。

❸ 当关闭标签时，激活左边相邻的标签。

关于打开的标签数量限制、关闭规则、激活规则都可以自定义。

操作步骤： 偏好设置→Editor→General→Editor Tabs →Tab Closing Policy→配置关闭规则。

16.9 编辑器颜色

16.9.1 设置是否显示条标和条标的显示颜色

条标是什么？

当编辑器中有报错的时候，在右边框上会显示一个红色的条，即条标，如图 16-56 所示。

图 16-56

这个条标有一套默认的颜色配置，也允许自定义。

操作步骤：偏好设置→Editor→Colors & Fonts→General（见图 16-57）。

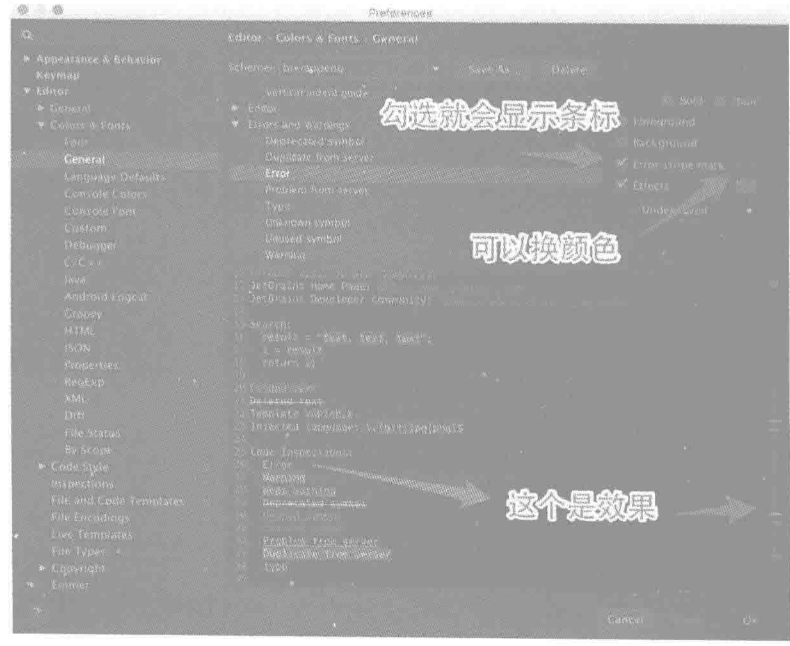


图 16-57

16.9.2 设置控制台的颜色

如果控制台输入的信息颜色不易区分或者更加个性化一点，可以自定义控制台颜色。

操作步骤：偏好设置→Editor→Colors & Fonts→Console Colors（见图 16-58）。

图 16-58

选择配色方案

我们可以选择一个现成的配色方案（见图 16-59）。例如，选择Sublime Text2，预览效果如图 16-60 所示。当然，这些配色都是可以修改的。

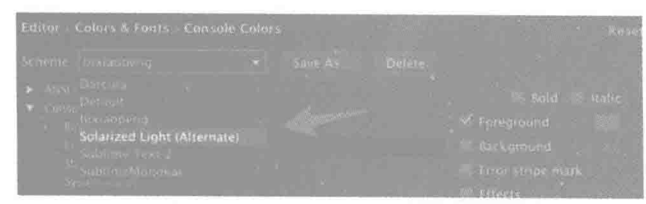


图 16-59

图 16-60

自定义配色方案

我们可以单击【Save As...】来复制一个配色方案，然后进行修改。如果我们想改变某一个输出类型的颜色或效果，可以参照图 16-61 进行操作。

如果不知道某一个颜色的色值，可以参考ANSI Colors，ANSI Colors中定义的颜色，单击后可以查看颜色的色值。例如，图 16-62 中的参数设置。效果如图 16-63 所示。

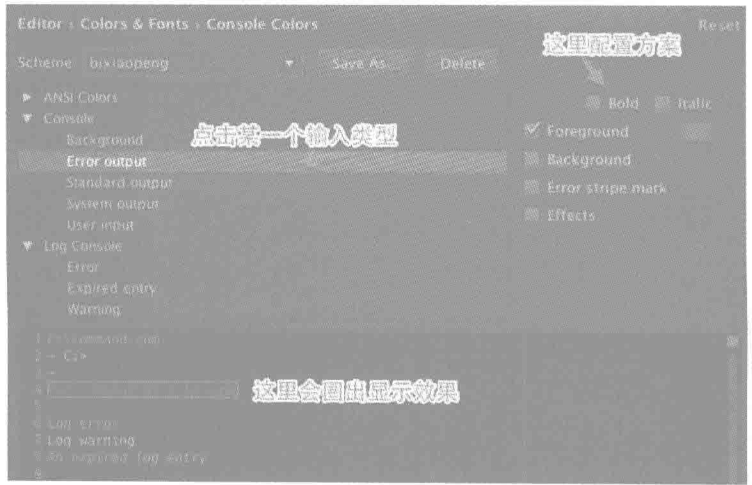


图 16-61

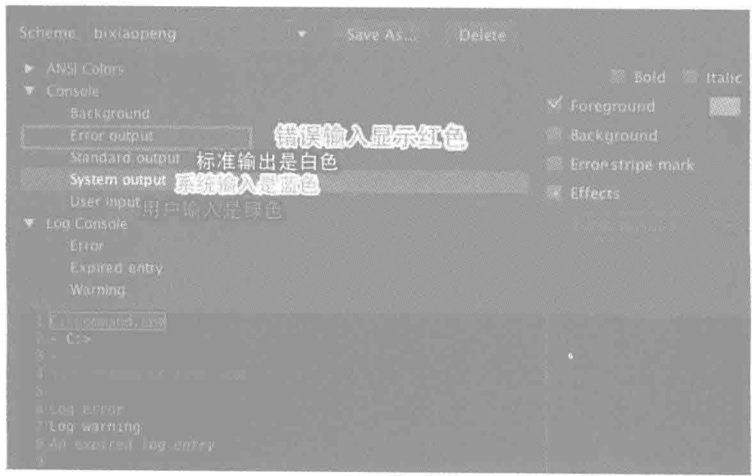


图 16-62

```
上午11:11:28: Executing external task 'build'...
Configuration on demand is an incubating feature.
:Application:preBuild UP-TO-DATE
:Application:preDebugBuild UP-TO-DATE
```

```
AAPT: /Volumes/warehouse/MySourceCode/android-ActionBarCompat-Shar
AAPT: /Volumes/warehouse/MySourceCode/android-ActionBarCompat-Shar
AAPT: /Volumes/warehouse/MySourceCode/android-ActionBarCompat-Shar
:Application:processReleaseManifest
```

图 16-63

16.9.3 设置控制台的字体

操作步骤：偏好设置→Editor→Colors & Fonts→Console Font（见图 16-64），效果如图 16-65 所示。

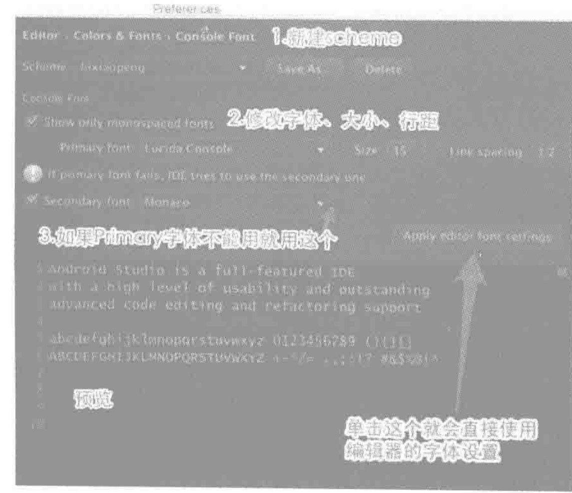


图 16-64

图 16-65

16.9.4 自定义代码的颜色

Android Studio代码的配色方案已经足够好，但是不满足的朋友若想更个性一点，也可以自定义代码的颜色。这里以Java为例进行介绍，其他代码自定义颜色方法与此相同。

操作步骤： 偏好设置→Editor→Colors & Fonts→Java→新建一个Scheme（例如JavaColors）。

Java代码按照一定的属性规则进行分类，如图 16-66所示。

图 16-66

单击具体的属性会显示具体的配色方案，如图 16-67所示。

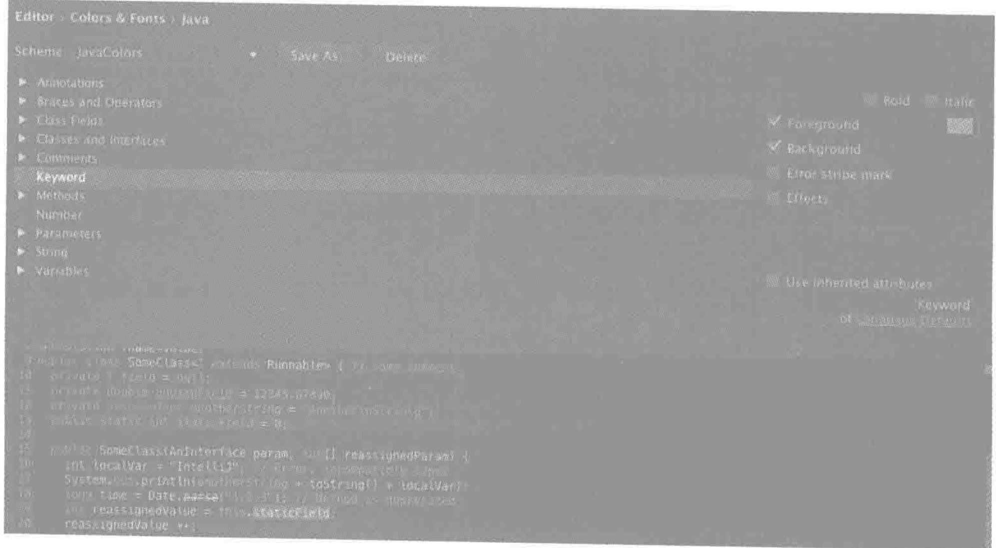


图 16-67

【实例演示】

Interface使用的是默认颜色，如图 16-68 所示。这里将把Interface的颜色变成绿色（见图 16-69）。

图 16-68

图 16-69

01 选中 Interface。

02 勾选 Foreground。

03 选择颜色（绿色）。

04 预览效果。

经过操作后，预览窗口中所有Interface都变成了绿色，如果不需要还可以再换个颜色。

16.10 代码风格

16.10.1 设置Java注释按缩进显示

Android Studio中的默认设置是代码格式化后注释会从第一列开始显示，如图 16-70 所示。这样看起来会有点别扭（见图 16-71），可以让注释按缩进显示。

图 16-70

图 16-71

操作步骤：偏好设置→Editor→Code Style→Java→Wrapping and Braces→Keep when reformatting→取消勾选【Comment at first column】，如图 16-72 所示。

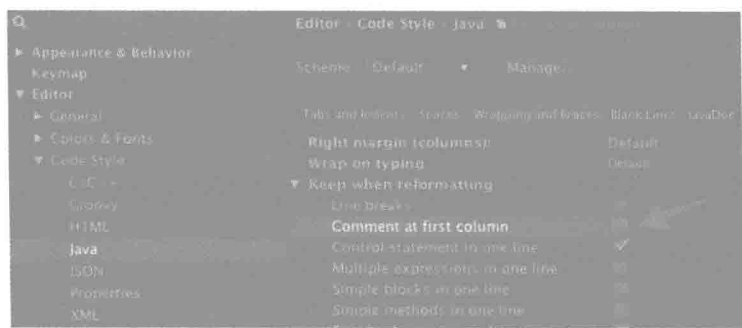


图 16-72

16.10.2 设置语句不要都显示在一行

Android Studio中的默认设置是代码格式化后较短的语句会合并成一行，如图 16-73 所示。这样可能会降低代码的可读性，可以去掉这个设置。

操作步骤：偏好设置→Editor→Code Style→Java→Wrapping and Braces→Keep when reformatting→取消勾选【Control statement in one line】，如图 16-74 所示。

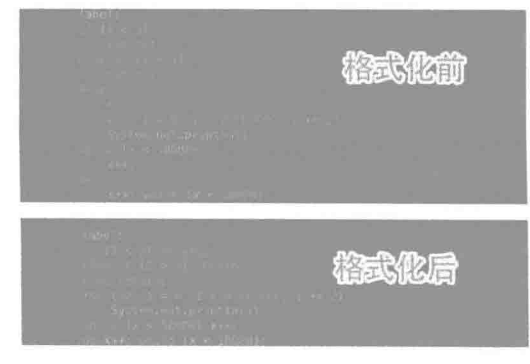


图 16-73

图 16-74

16.10.3 设置Java简单的类合并为一行

Android Studio中的默认设置是代码格式化后不会把简单的类合并为一行，如图 16-75 所示。不过我们可以把简单的类合并为一行（见图 16-76）。

图 16-75

图 16-76

操作步骤：偏好设置→Editor→Code Style→Java→Wrapping and Braces →Keep when reformatting→勾选 【Simple classes in one line】。

16.10.4 设置Java字段和变量列对齐

Android Studio中默认是没有列对齐的，效果如图 16-77 所示。如果设置了列对齐，效果如图 16-78 所示。

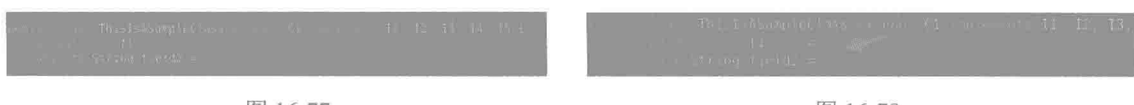


图 16-77　　图 16-78

操作步骤：偏好设置→Editor→Code Style→Java→Wrapping and Braces→Group declarations→勾选【Align fields in columns】和【Align variables in columns】，如图 16-79 所示。所有设置在格式化代码的时候就会生效。

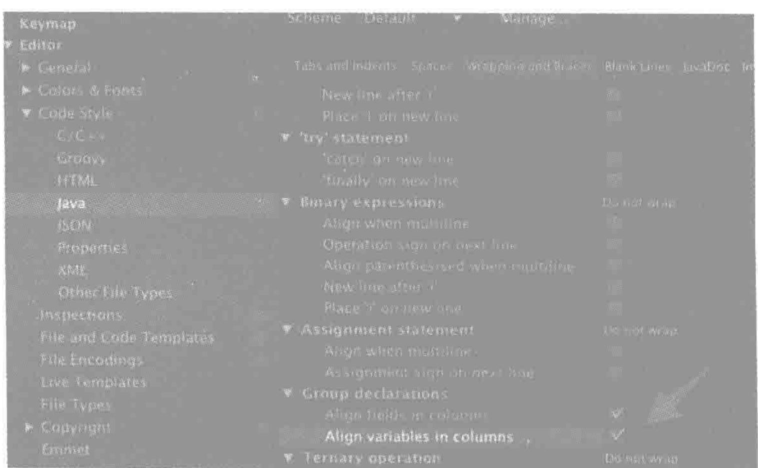


图 16-79

16.10.5 设置自动生成字段名称时添加前缀

默认自动生成字段名称是不会有任何前缀的，如图 16-80 所示。不过在Android Studio中

提供了自动添加自定义前缀或后缀的功能，以便统一编码规范。比如按照Android编码规范的指导，静态成员变量以“s”开头，非静态、私有成员变量以“m”开头。效果如图 16-81 所示。

图 16-80

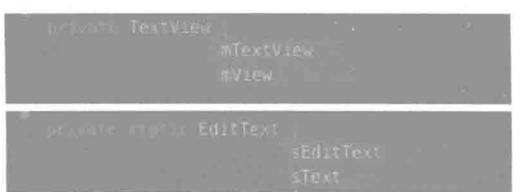


图 16-81

操作步骤：偏好设置→Editor→Code Style→Java →Code Generation→输入需要添加的前缀或后缀，如图 16-82 所示。

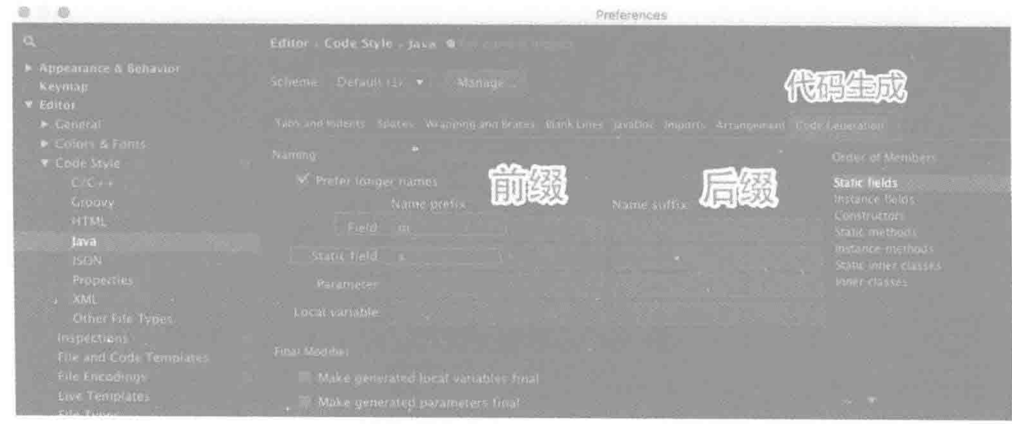


图 16-82

16.11 文件和代码模板

16.11.1 设置新建文件的注释模板

Android Studio中默认的文件注释如图 16-83 所示。

当我们新建一个文件时会自动插入这个注释，如图 16-84 所示。

图 16-83

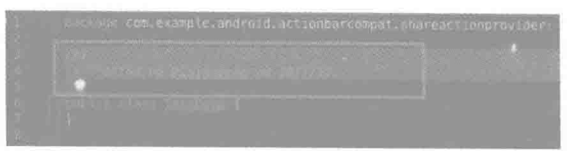


图 16-84

我们可以通过下面的操作修改注释，并新建为模板。

操作步骤：偏好设置→Editor→File and Code Templates→Includes→File Header→输入自定义的注释模板，如图 16-85 所示。新建一个文件，并保存为模板，如图 16-86 所示。

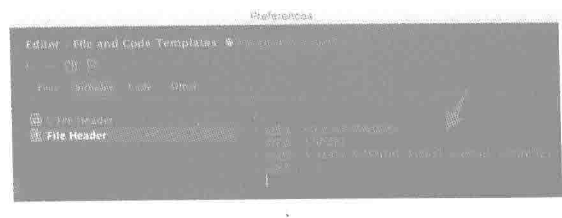


图 16-85

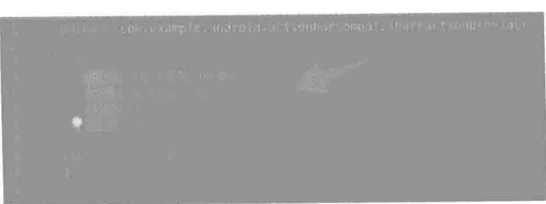


图 16-86

16.11.2 模板中内置的变量

- ${PACKAGE_NAME}：包名。
- ${USER}：系统的登录名。
- ${DATE}：系统日期。
- ${TIME}：系统时间。
- ${YEAR}：年份。
- ${MONTH}：月份。
- ${MONTH_NAME_SHORT}：月份的前三个字符。
- ${MONTH_NAME_FULL}：月份的全名。
- ${DAY}：天。
- ${HOUR}：时。
- ${MINUTE}：分钟。
- ${PROJECT_NAME}：项目名。

16.11.3 设置新建类文件模板

在类文件中可以引用头文件的注释模板，如图 16-87 所示。

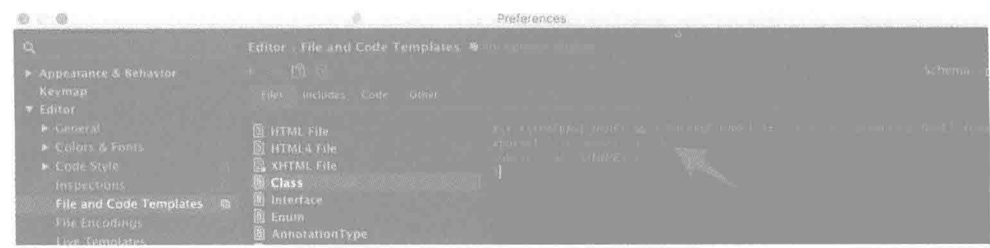


图 16-87

如果我们想在新建类文件时自动添加某段代码，也可以在这里设置，如图 16-88 所示。

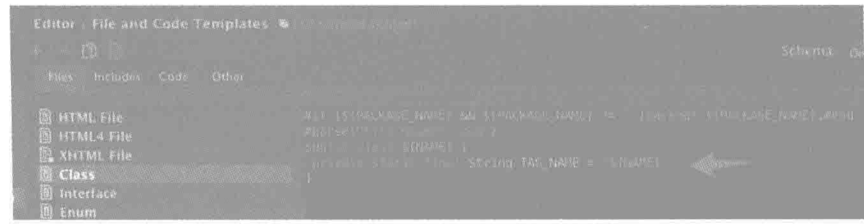


图 16-88

加了一段代码以后我们再新建一个类文件，如图 16-89 所示。

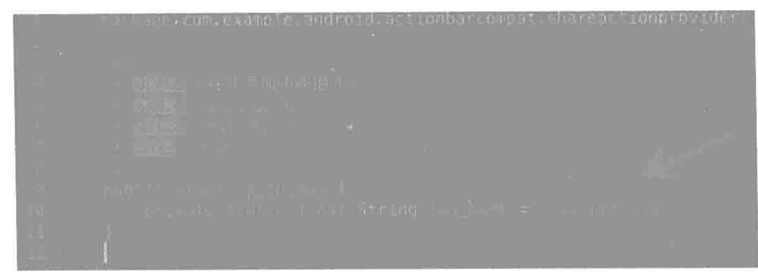


图 16-89

每次新建类文件时，这段代码模板都会被应用，效率会提高很多。

16.11.4 设置IDE和项目的编码

切换文件编码是比较常见的操作。在Android Studio中，我们可以设置默认的编码方式，如果某个项目需要切换就可以进行切换。

操作步骤：偏好设置→Editor→File Encodings。如图 16-90 所示，properties文件默认的编码方式是ISO 8859-1，如果属性文件中我们使用了中文，就有可能导致中文乱码。因此我们一般都会勾选Transparent native-to-ascii conversion用于属性文件的中文转码。

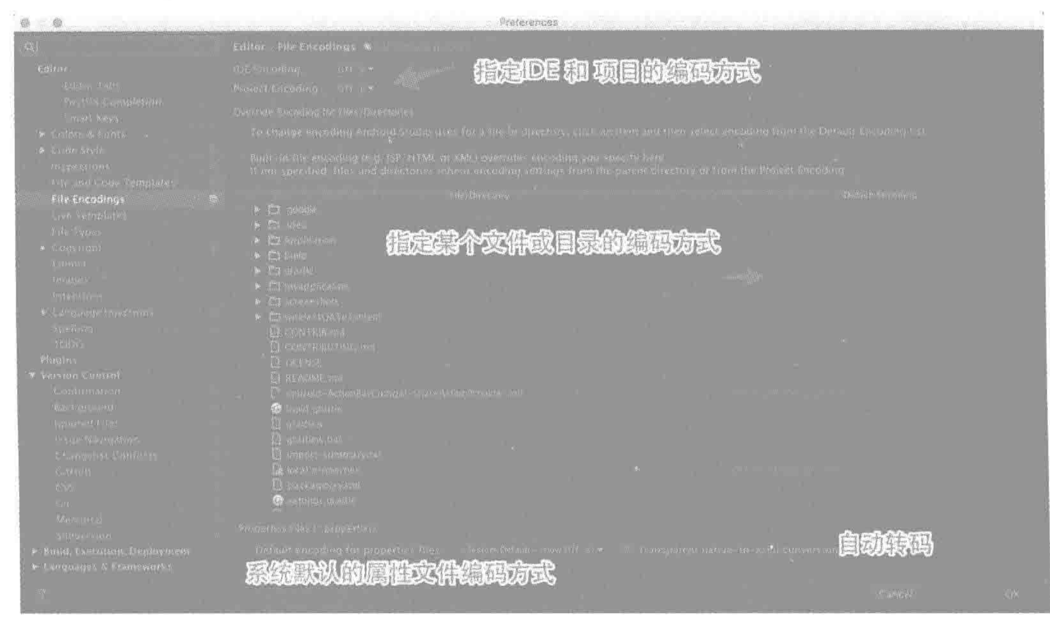


图 16-90

16.11.5 对动态模板进行增删改查

Android Studio中提供了快速插入常用代码的功能Live Templates（动态模板），我们可以对其进行增删改查。

操作步骤：偏好设置→Editor→Live Templates→对动态模板进行增删改查。

- 查看动态模板分组，效果如图 16-91 所示。

图 16-91

- 新增/删除/复制一个模板，如图 16-92 所示。新增模板的规则参考现有的模板就可以了。

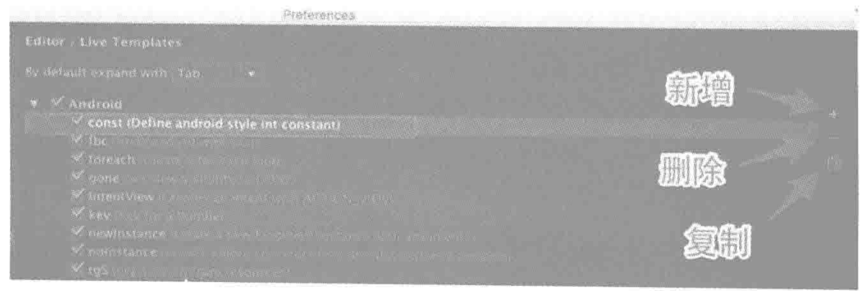


图 16-92

- 查看具体模板，如图 16-93 所示。

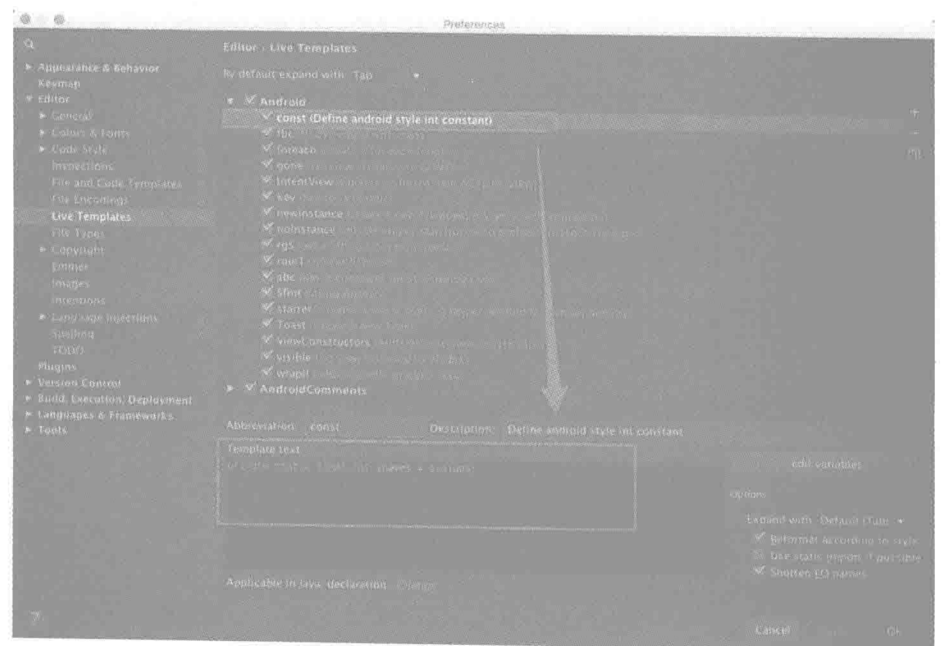


图 16-93

16.11.6 设置展开代码的按键

当我们要插入动态模板的时候，输完代码片段的缩写以后默认按Tab键就可以展开代码。当然也可以自定义展开按键（比如把空格、Tab、回车键设置成展开代码的按键），如图 16-94 所示。

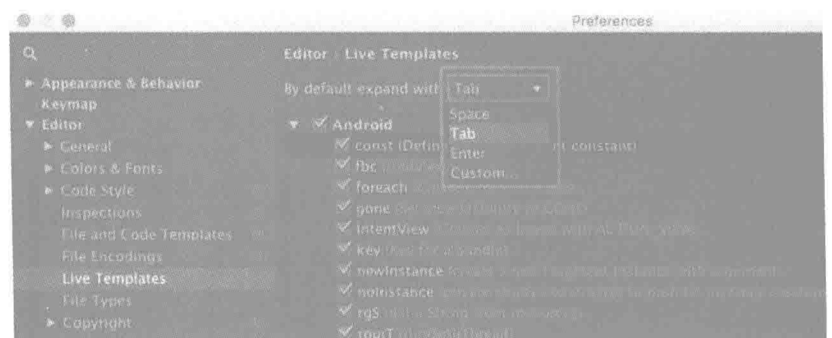


图 16-94

16.11.7 给一个文件类型添加匹配规则

操作步骤：偏好设置→Editor→File Types→在Recognized File Types列表中选中一个文件类型→在Registered Patterns中单击+，添加一个文件类型的匹配规则，如图 16-95 所示。

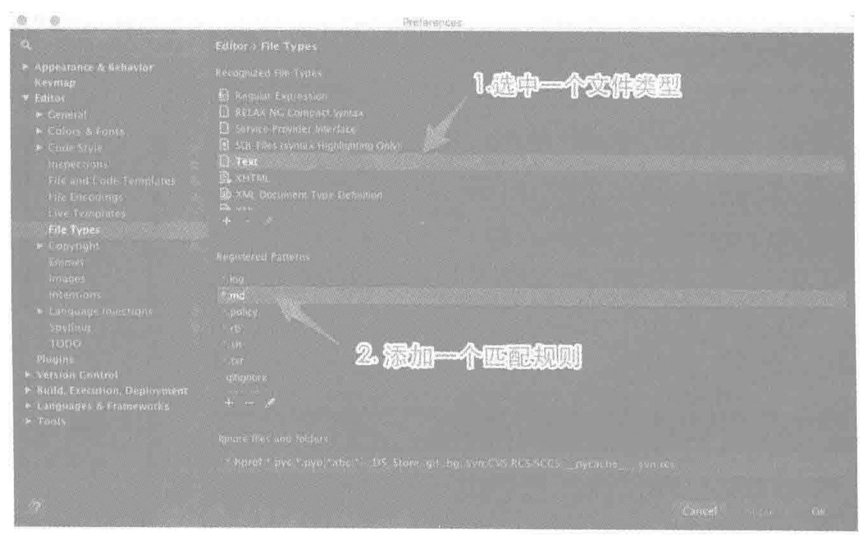


图 16-95

16.11.8 添加一个自定义的文件类型

如果Android Studio可识别的文件类型不能满足需求，可以自定义一个。Android Studio中支持自定义文件类型，可以对这个文件类型设置一些特性，如关键字、注释、高亮显示等，还可以设置一些模板，提高操作的效率。

操作步骤：偏好设置→Editor→File Types →在Recognized File Types中单击+→设置新的文件类型。

【实例演示】添加一个名为custom的文件类型。

01 进入新建文件类型界面 → 输入文件名和描述 → 设置高亮显示规则 → 输入关键字 → OK，如图 16-96 所示。

02 为自定义的文件类型添加匹配规则，如图 16-97 所示。

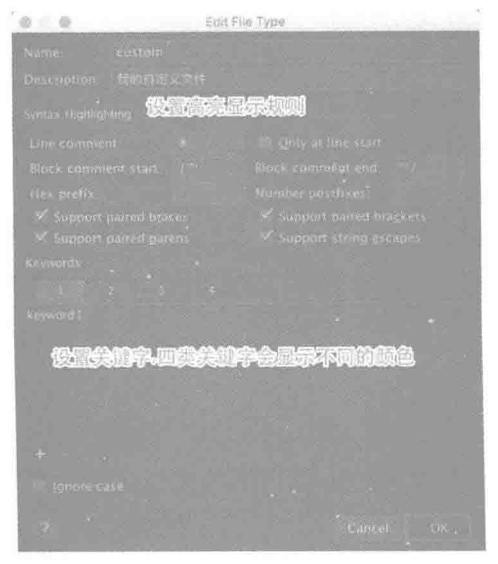


图 16-96

图 16-97

03 新建一个.custom 文件，输入关键字和注释，校验是否高亮显示了，如图 16-98 所示。

在图 16-98 中，我们自定义的语法高亮显示规则为：

- 注释显示为灰色（默认）。
- 字符串显示为绿色（默认）。
- 第一类关键字显示为橙色，通常用于条件判断和语法关键字，如 for、while、if 或 public 、int 。
- 第二类关键字显示为紫色，通常用于全局变量和保留变量。
- 第三类关键字显示为蓝色，通常用于系统级别的函数。
- 第四类关键字显示为红色，通常用于警示标志，如错误、异常等。

图 16-98

16.11.9 设置忽略某类文件或文件夹

如果我们不想让某个文件或某类型的文件在编辑器中显示，就可以将其忽略。

操作步骤：偏好设置→Editor→File Types→在Ignore files and folders中添加文件类型。

【实例演示】忽略项目中txt类型的文件。

操作步骤：Editor→File Types→在Ignore files and folders中添加*.txt;→OK，如图 16-99 所示。

设置成功后，项目中的txt文件在Android Studio项目中消失了，其实这个txt文件还是存在的，只不过被Android Studio忽略而不显示了。

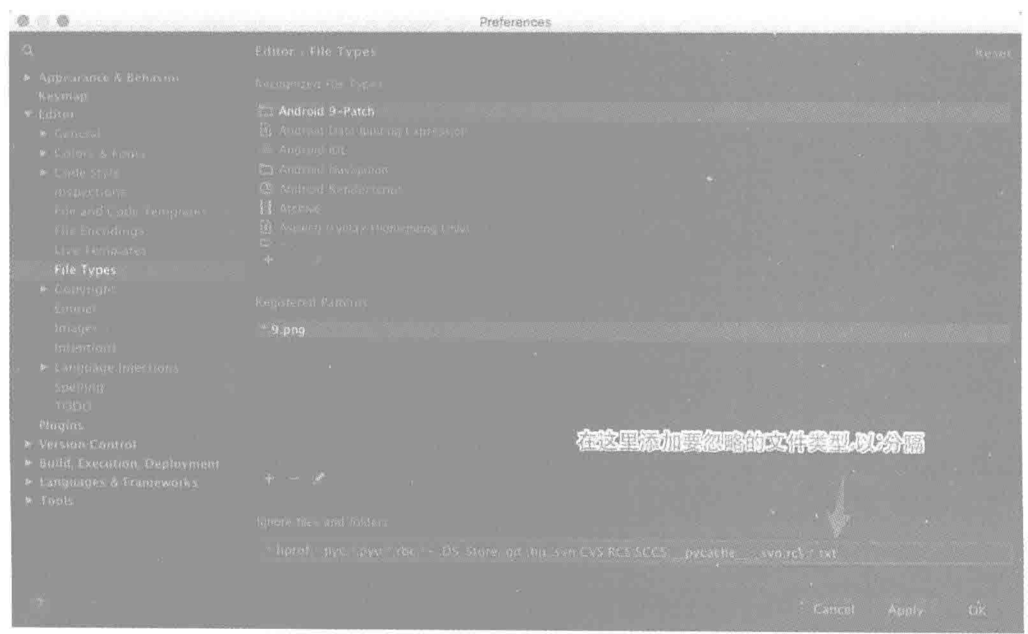


图 16-99

16.12 插　件

Android Studio是基于IntelliJ IDEA开发的，自然继承了IntelliJ IDEA优秀的插件系统。我们可以通过安装插件来扩展Android Studio的功能，这会极大地方便我们日常的开发工作。

进入到偏好设置中来管理插件：偏好设置→ Plugins，如图 16-100 所示。

图 16-100

过滤已安装插件，如图 16-101 所示。

- All plugins：所有插件。
- Enabled：所有已启用的插件。
- Disabled：所有已禁用的插件。
- Bundled：所有 IDEA 自带的插件。
- Custom：所有我们自己安装的插件。

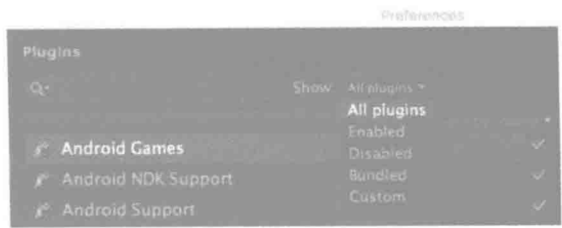


图 16-101

16.12.1 安装插件

Android Studio中有以下 3 种安装插件的途径。

1. 安装JetBrains插件（Install JetBrains plugin）

如果我们想安装JetBrains官方开发的插件，可以单击【Install JetBrains plugin】，然后弹出插件列表，如图 16-102 所示。

图 16-102

为了能够快速地找到我们想要的插件，既可以对插件列表进行排序，也可以过滤插件的分类。

（1）插件排序

插件列表默认按name排序，同时允许添加排序条件（多选），如Status（状态，是否已安装）、Downloads（下载量）、Rating（评价）、Last Updated（最后更新时间），如图 16-103 所示。

例如，我们想查看评价最高且最近有更新的插件，就需要勾选【Rating】和【Last Updated】，如图 16-104 所示。

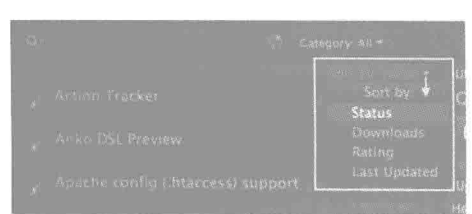


图 16-103

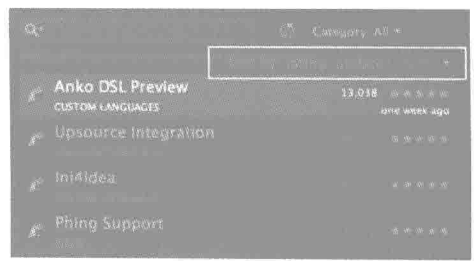


图 16-104

（2）插件分类（见图 16-105）

图 16-105

（3）安装插件

01 选中插件 → 在插件简介中单击【Install】按钮，如图 16-106 所示。如果没有显示 Install 按钮，说明这个插件已经被安装了。

图 16-106

02 下载插件，如图 16-107 所示。

图 16-107

03 下载完成后重启 Android Studio，插件安装成功，如图 16-108 所示。

图 16-108

（4）设置代理

如果插件无法安装，可以尝试设置代理，学会科学上网，如图 16-109 所示。

图 16-109

2. 浏览插件仓库（Browse repositories）

如果我们想查看插件仓库中所有的插件，可以单击【Browse repositories】，然后弹出插件列表。这里的操作跟安装JetBrains插件是一样的，不同的是多了一个【Manage repositories…】（见图 16-110），用于管理仓库地址。

图 16-110

3. 从本地磁盘安装插件（Install plugin from disk）

如果你不懂科学上网，或者嫌直接下载安装插件太慢，也可以从本地磁盘安装插件。

第 1 步：下载插件到本地。到Android Studio插件官网（https://plugins.jetbrains.com/?androidstudio）找到自己想要的插件，下载到本地。例如，插件.ignore，下载地址为https://plugins.jetbrains.com/plugin/7495?pr=androidstudio。

第 2 步：单击【Install plugin from disk】→ 选择下载的插件，如图 16-111 所示。

图 16-111

第 3 步：重启Android Studio后，插件安装成功，如图 16-112 所示。

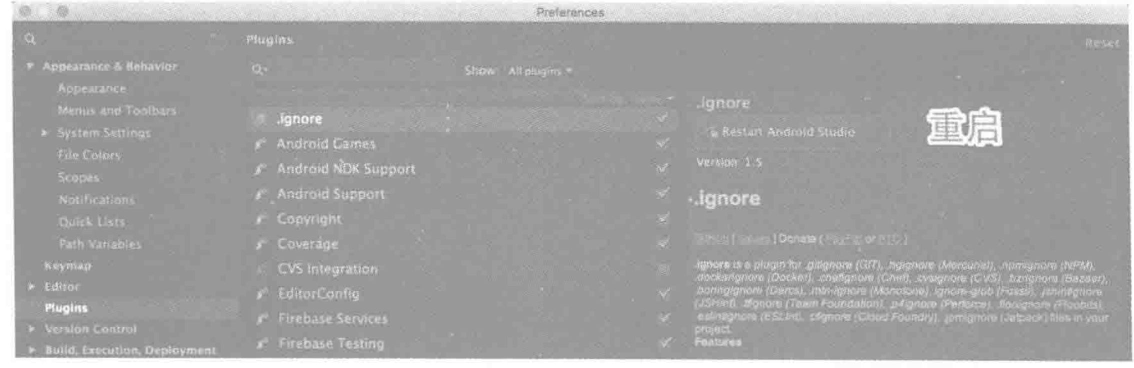


图 16-112

16.12.2 禁用插件

Android Studio中默认安装了一些根本用不到的插件，可以禁用这些插件。

操作步骤：偏好设置→Plugins→取消勾选想禁用的插件→重启→生效。

【实例演示】禁用CVS插件。

禁用前后效果对比如图 16-113、图 16-114 所示。

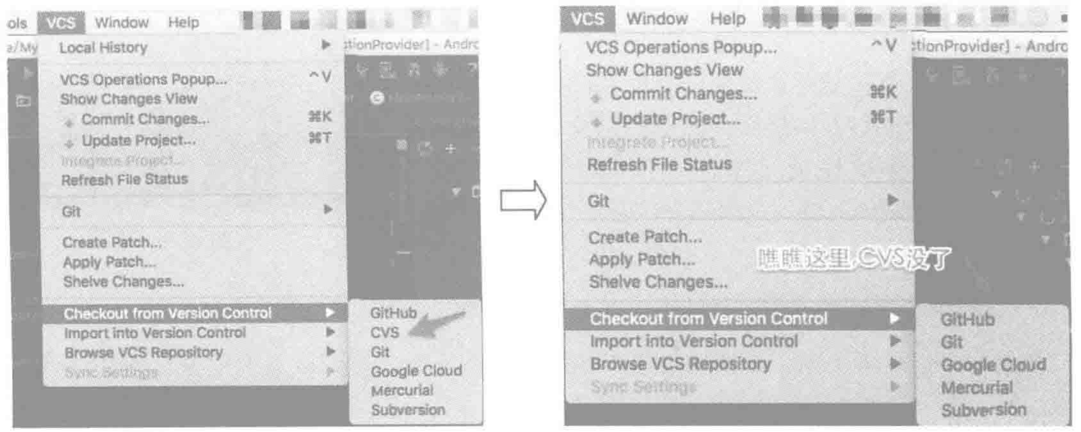


图 16-113　　　　图 16-114

设置禁用CVS插件的界面如图 16-115 所示。确定后重启即可看到图 16-114 所示的效果。

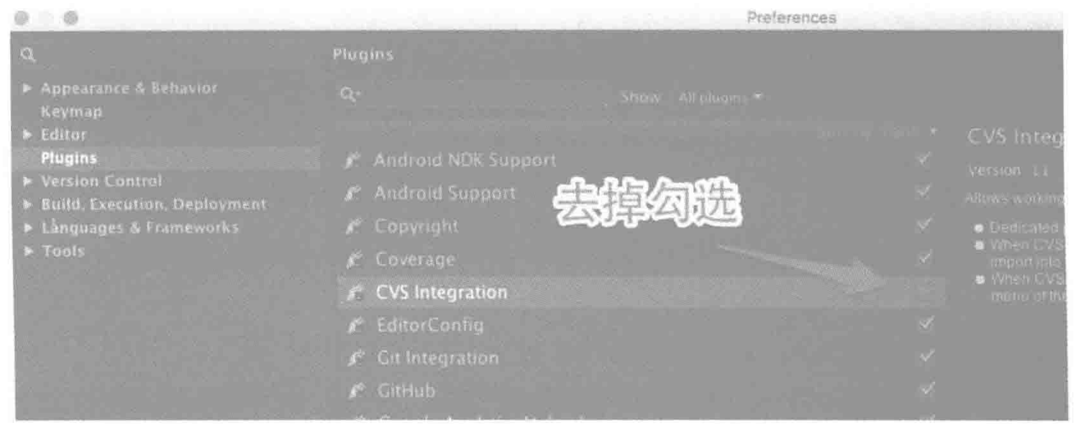


图 16-115

16.12.3　卸载插件

一些预装的插件是不能卸载的，但可以禁用。若是我们自己安装的插件则是可以卸载的。操作步骤：偏好设置→Plugins→选中插件→Uninstall（见图 16-116）→重启→生效。

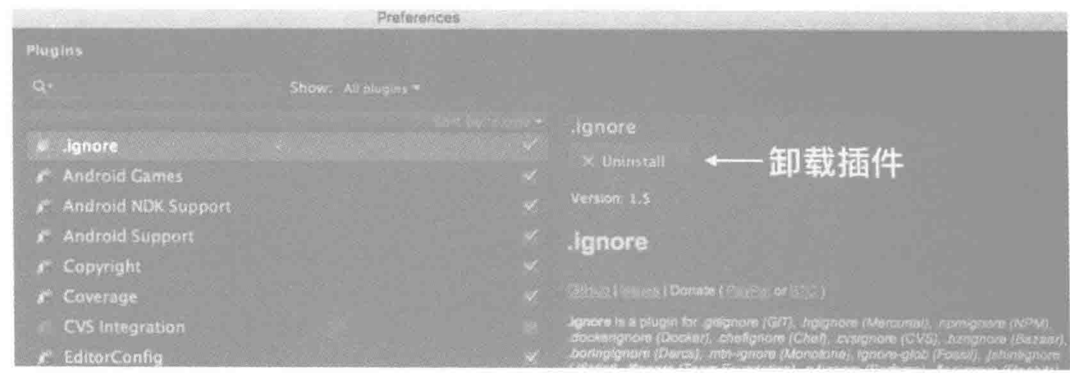


图 16-116　卸载插件

16.12.4 常用插件

IntelliJ IDEA和Android Studio的插件非常多，很多好用的插件需要我们自己去发掘，在这里只推荐几个常用而且被广泛好评的插件。

编辑器插件如表 16-1 所示。

表 16-1

插件名	描述
Key promoter	用于生成快捷键提示
CodeGlance	用于生成代码缩略图，方便快速定位
Eclipse Code Formatter	用于使用Eclipse 的代码格式化风格
BashSupport	用于支持Bash语言
IdeaVim	用于支持VIM
Grep Console	用于自定义控制台输出颜色
Maven Helper	用于查看Maven依赖树

代码工具插件如表 16-2 所示。

表 16-2

插件名	描述
String Manipulation	用于快速切换驼峰式命名和下划线命名
GsonFormat	用于快速格式化json数据，自动生成实体类参数
Android ButterKnife Zelezny	用于快速添加Butterknife注解
Android Drawable Importer	用于快速导入缩放图标到Android项目中
Android Parcelable code generator	用于快速实现Parcelable接口
Android Postfix completion	用于根据后缀快速完成代码
AndroidAccessors	用于快速实现get和set方法的插件
Android Postfix Completion	用于快速编写Toast，Log，isEmpty，FindView等代码
JavaDoc	用于快速生成Java注释，可自定义模板

检查工具插件如表 16-3 所示。

表 16-3

插件名	描述
CheckStyle-IDEA	用于检查Java源代码是否符合编码规则
FindBugs-IDEA	用于检查Java代码中潜在的问题
MetricsReloaded	用于检查代码复杂度

其他常用插件如表 16-4 所示。

表 16-4

插件名	描述
Android WiFi Adb	用于通过WIFI连接ADB，不需要root
ADB Idea	用于在Android Studio中可视化使用ADB命令
RESTClient	用于测试 RESTful web services
JSONOnlineViewer	用于直接在Android Studio中调试接口数据
jimu Mirror	用于在真机上快速测试布局
Codota	用于搜索Android代码实例
Genymotion	用于使用Genymotion运行和调试应用

16.13 编译和构建

很多朋友都会抱怨使用Android Studio编译和构建大型项目的时候特别慢，其实可以优化编译和构建的速度。

16.13.1 设置Android Studio的内存参数

如果Android Studio默认的内存参数不能满足我们的性能需求，可以重新设置这些参数。

操作步骤：菜单栏→Help→Edit Custom VM Options。

如果没有创建studio.vmoptions就会提示我们先创建，然后打开这个文件。

在文件中设置内存参数：

```
# custom Android Studio VM options, see
http://tools.android.com/tech-docs/configuration
-Xms2g
-Xmx2g
-XX：ReservedCodeCacheSize=1024m
-XX：+UseCompressedOops
```

如果想了解更多设置技巧，请看http://tools.android.com/tech-docs/configuration。设置成功后需要重启Android Studio，然后查看右下角状态栏上的内存状态，确认已经生效，如图 16-117 所示。如果状态栏没有显示内存状态，请参考 16.1.2 小节的内容。

图 16-117

16.13.2 设置自动编译项目

编译项目是对新产生变化的文件进行一次编译，已经编译过的文件就不用重编译了。所以如果我们想加快编译速度，可以设置项目自动编译。

操作步骤：偏好设置→Build，Execution，Deployment→Build Tools→Compiler→勾选【Make project automatically（only words while not running | debugging）】，如图 16-118 所示。

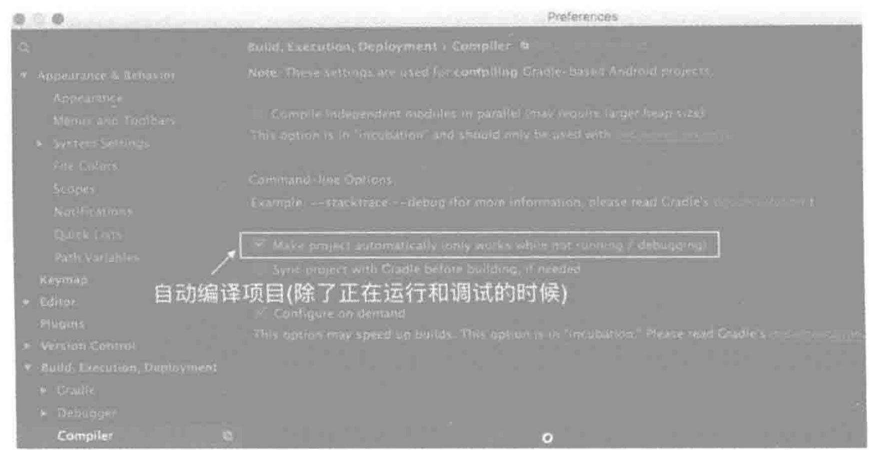


图 16-118

16.13.3　设置并行编译

想提升编译速度，可开启并行编译，这样会比单纯线依次编译快很多。

操作步骤：偏好设置→Build，Execution，Deployment→Build Tools→Compiler→勾选【Compile independent modules in parallel（may require larger heap size）】（请注意括号中的提示，因此使用并行编译需要大的堆内存，所以我们需要同时调整编译内存的大小），如图 16-119 所示。

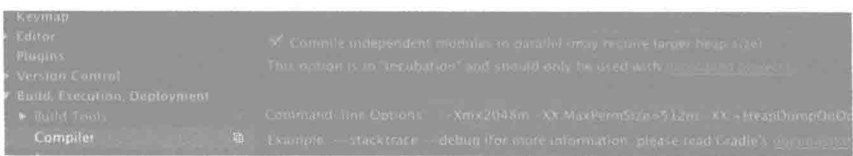


图 16-119

16.13.4　调整编译内存大小

编译内存太小有可能会导致OOM，可以按照如下操作调整编译内存。

操作步骤：偏好设置→Build，Execution，Deployment→Build Tools→Compiler→在【Command-line Options】中输入调整的内存参数，如图 16-120 所示。参数可参考gradle.properties中的配置提示。

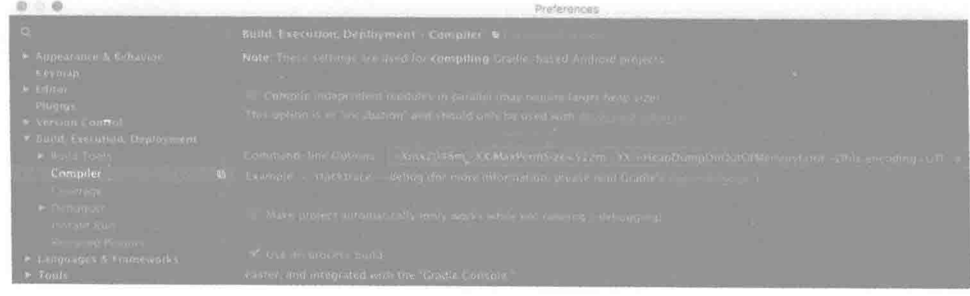


图 16-120

默认情况下，每次打开或导入项目时，Android Studio总会联网检查依赖的文件是否已经下载、是否有新版本，这样会导致速度很慢。如果我们很清楚依赖的文件已经下载且不想更新到最新的版本，可以使用Gradle离线工作模式。

操作步骤：偏好设置→Build，Execution，Deployment→Build Tools→Gradle →勾选【Offline work】，如图 16-121 所示。

图 16-121

提 示

通常情况下我们是不会知道是否所有的 jar 包或插件都已经下载，如果使用了离线工作模式很有可能导致项目报错。这时可以单击同步项目：Sync Project with Gradle Files。

附录　Android Studio 重要版本发布时间线

2013 年 5 月 16 日，Google I/O大会上发布了Android Studio 0.1 预览版本，全世界为此欢呼雀跃，Android的开发者终于有了自己的IDE工具。

2014 年 12 月 8 日，Android Studio 1.0 稳定版本发布，这一版本新增了很多特性，提供了安装向导、代码示例、项目创建向导、统一的构建系统（Gradle）、国际化字符串编码、可视化布局编辑器、性能分析工具、集成Google云服务等，这是真正意义的里程碑式的版本。

2015 年 12 月 18 日，Android Studio 1.5 稳定版本发布，这一版本专注于Android Studio自身的错误修复和稳定性，内存分析器中新增了检测常规内存泄漏的功能，Lint检查也增加了一些新的规则。

2016 年 4 月 8 日，Android Studio 2.0 稳定版本发布，这一版本专注于提升构建的效率，新增即时运行（Instant Run）、重新设计了模拟器。

2016 年 5 月 19 日，Android Studio 2.2 预览版Preview1 发布，这又是一次重大而全面的更新。设计方面：推出全新的可视化布局编辑器、新增一个全新的Android布局（约束布局）、集成布局检查器。开发方面：新增全新的开发者服务套件Firebase、增强代码检查功能、新增代码示例浏览器、改进C++支持等。构建方面：改进Jack 编辑器、合并清单文件查看器等。测试方面：新增基于Espresso的测试代码录制工具、APK分析器。

2016 年 9 月 19 日，时隔整整 4 个月后，Android Studio 2.2 稳定版本发布。

参考资料

[1] https://developer.android.com/studio/index.html

[2] https://www.jetbrains.com/idea/

[3] http://chinagdg.org/

[4] http://tools.android.com/recent

耗时两年，倾力打造
百度阅读畅销电子书

本书需要你有目的地去阅读，当你遇到问题或者想了解某个工具如何使用时，可以直接定位到相关的章节查看使用说明。

本书是一本非常实用的指导手册，它几乎包含了Android Studio所有的实用功能和操作技巧，适合放在你的电脑旁经常翻阅。

本书以通俗易懂的语言描述工具的使用技巧，并且每个操作都有实例演示，让读者感觉是在跟一个有经验的人聊天。

本书以近1500张图片详细描述Android Studio的使用，是真正的图文并茂。

本书以解决问题为目的，讲述如何使用工具来解决实际问题。

本书专注于操作技巧的讲解，对于Android开发的基础知识略有提及，但不是本书的重点。

本书以macOS上的操作为例进行演示，不同操作系统上Android Studio的操作差异不大，对于快捷键会区分macOS / Windows / Linux。

本书的大部分操作技巧同样适用于IntelliJ IDEA。

- 如果你是Android开发初学者；
- 如果你想从Eclipse转到Android Studio；
- 如果你是从其他语言转到Android开发；
- 如果你想深入了解Android Studio；
- 如果你想深入了解IntelliJ IDEA；
- 如果你想从事Android测试开发；
- 如果你想节省搜索时间；
- 如果你想提高工作效率；

那么，这本书正是为你量身打造的！

毕小朋，CSDN博客专家，8年移动互联网从业经验，熟悉Android开发与测试，所编写的《Android Studio实用指南》电子书持续畅销，备受读者推崇，获得百度阅读的鼎力推荐。

清华大学出版社数字出版网站

www.wqbook.com

ISBN 978-7-302-45530-1

定价：89.00元

[General Information]
书名=精通Android Studio
作者=毕小明
页数=560
SS号=14169118
DX号=
出版日期=2017.01
出版社=北京清华大学出版社