

Android Studio实战

快速、高效地构建Android应用

Learn Android Studio

Build Android Apps Quickly and Effectively



[美] Adam Gerber 著
Clifton Craig
靳晓辉 张文书 译



移动开发经典丛书

Android Studio 实战

快速、高效地构建

Android 应用

[美] Adam Gerber 著
Clifton Craig
靳晓辉 张文书 译

清华大学出版社

北 京

Adam Gerber, Clifton Craig

Learn Android Studio: Build Android Apps Quickly and Effectively

EISBN: 978-1-4302-6601-3

Original English language edition published by Apress Media. Copyright © 2015 by Apress Media.

Simplified Chinese-Language edition copyright © 2016 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。未经出版者书面许可,不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字: 01-2016-4648

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

Android Studio 实战 快速、高效地构建 Android 应用/(美) 亚当·格伯(Adam Gerber), (美) 克利夫顿·克雷格(Clifton Craig) 著; 靳晓辉, 张文书 译. —北京: 清华大学出版社, 2016
(移动开发经典丛书)

书名原文: Learn Android Studio: Build Android Apps Quickly and Effectively

ISBN 978-7-302-44153-3

I. ①A… II. ①亚…②克…③靳…④张… 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 148557 号

责任编辑: 王 军 李维杰

装帧设计: 牛静敏

责任校对: 成凤进

责任印制: 王静怡

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 装 者: 北京嘉实印刷有限公司

经 销: 全国新华书店

开 本: 185mm×260mm 印 张: 24.75 字 数: 602 千字

版 次: 2016 年 7 月第 1 版 印 次: 2016 年 7 月第 1 次印刷

印 数: 1~3000

定 价: 59.80 元

产品编号: 067858-01

译者序

在当今的移动互联网时代，绿色机器人 Android 可谓家喻户晓。Android 开发自然也随着搭载 Android 平台的智能手机、平板电脑和可穿戴设备的热销而变得异常火热。“工欲善其事，必先利其器”——Android Studio 作为官方推荐的 IDE，已经成为高效 Android 开发者的必备神器。Android Studio 是 Google 公司在 2013 Google I/O 大会上发布的全新 Android 开发 IDE。引用原书中的描述：Android Studio 是 JetBrains 和 Google 合作的产物——它基于 JetBrains 的 IntelliJ 构建，其功能是 IntelliJ 的超集。

互联网是一个比拼效率的行业，而移动互联网更加速了应用开发的节奏。与之相适应的是现代 IDE 的设计也更加注重用户体验，以便让开发者能够将更多的精力放在编写代码上。因此，除了基础的编辑功能之外，智能代码补全、可感知语法的重构、可视化的界面编辑器、调试和性能分析工具、代码管理和工程构建工具已经成为一款优秀 IDE 的标配。如果以此作为衡量标准，那么与其他 Android 开发工具相比，Android Studio 在所有这些方面无疑都更胜一筹。Android Studio 中集成了大量主流的辅助工具，能帮助开发者应对在项目各个阶段可能遇到的问题，可谓提供了 Android 开发的“一站式”服务。

本书结合若干个实际可用的示例，循序渐进地讲解了 Android Studio 的基本用法、Android App 的开发流程(包括设计用户界面、搭建布局、采用 MVC 模式编写代码等)、代码管理工具 Git 和构建工具 Gradle 的使用技巧。本书还简要介绍了 Android 可穿戴设备以及使用 Android Wear SDK 进行开发的方法。本书适合正在考虑从 Eclipse+ADT 向 Android Studio 迁移的 Android 开发者。

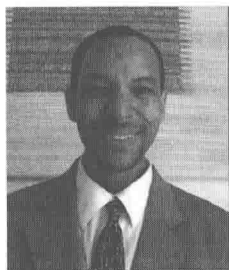
最后，我要在这里感谢清华大学出版社的李阳编辑给予我的信任，将该书的翻译工作交给我，并认真阅读译稿，提出诸多宝贵的修改意见，您的专业和认真给我留下了深刻的印象。本书的翻译和校对工作均是利用业余时间完成的，因此牺牲了不少本应该陪伴家人的时间，在此感谢他们的理解和支持。本书主要由靳晓辉翻译，此外，郭文豪、何孟宇、黄宇轩、李一夫、刘峰、龙伟、贾立克、穆玉伟、乌力吉、张国、张文书、张宇欣、周俊健、朱文等人也参与了本书部分章节的翻译，在此一并对他们表示感谢。由于译者水平有限，加之时间仓促，译文中可能会存在不准确的内容。如果读者在阅读过程中发现谬误和遗漏之处，烦请多多包涵并不吝批评指正。

靳晓辉

作者简介



Adam Gerber 是 Android Studio 的最早期使用者之一，他使用 Android Studio 开发专业的 Android 应用并将其用于自己在芝加哥大学开设的 Android 应用开发和技术创业等课程的教学。Adam 是“芝加哥创新交流”的成员并从事技术和创业领域的咨询工作。Adam 拥有伊利诺伊大学工业设计本科学位以及位于巴黎的法国国立工艺学院管理科学博士学位。Adam 的邮箱是 gerber@uchicago.edu。



Clifton Craig 从事软件工程师的工作已逾 16 载。他的经验涵盖了 J2ME/BlackBerry、Android 和 iOS，以及基于 JEE 的后端系统。他参与过一些备受瞩目的项目，包括 MapQuest 天然气价格门户网站、基于 J2ME 和 Android 的 MapQuest 移动版、基于 iOS 的 MapQuest 移动版以及 Skype 的 iOS 和 Android 版。他维护着一个技术博客 (cliftoncraig.com)，其中涵盖各种软件主题，从 Android 和 Linux 到 iOS 和 OS X。他有着从军经历而且是一位狂热的自行车迷、虔诚的基督徒以及两个天才小女孩的父亲。

技术审稿人简介

Jim Graham 于 1989 年在德克萨斯农工大学获得电子科学系电信专业的本科学位。他在国际通信协会的 1988 期的 ICA 公报上发表过论文(“Fast Packet Switching: An Overview of Theory and Performance”); 在伊利诺伊州芝加哥市 Amoco 公司的网络设计小组做过助理网络工程师; 在佛罗里达州沃尔顿堡海滩的 Tybrin 公司做过高级网络工程师; 在佛罗里达州赫尔伯特训练场的“第 16 特种作业翼情报处”和“美国空军总部特种作战指挥情报处”做过情报系统分析员。他于 2001 年收到来自“第 16 特种作业翼情报处”的正式表扬信。

致 谢

讲述像 Android 这样宏大的主题以及像 Android Studio 这样强大的工具需要投入、努力和多人协作。我们想要感谢编辑和技术审稿人——Corbin Collins、Mark Powers 和 Jim Graham。此外，我们还要向其他付出了直接和间接努力的人士致谢。

在大部分写作过程中，Android Studio 都处于 beta 版并且在不断更新。我们的实验和代码示例需要如此频繁地重做，以至于我已经记不清更新的具体次数了。非常感谢我的联合作者——Clifton Craig，他总是大刀阔斧地解决难题。我还想感谢我的家人和朋友，尤其是 Mia Park 一直支持着我度过这段充满挑战但收获颇丰的历程。我还想要感谢 Marilyn Meyers 对我一直以来的信任。非常感谢来自 Apress 的优秀专业团队，他们在编辑方面提供的支持非常重要。

——Adam Gerber

感谢 Onur Cinar 将我介绍给 Apress 的同事们，他们都是和蔼可亲的人。感谢我的联合作者 Adam Gerber，他在整个过程中都保持着积极的态度并且是一位优秀的推动者。也要感谢我最亲密的一些朋友——Juan Carlos Jimenez、Steve O'Sullivan、Nizam Gok 和 Yanxia Zhang，当遇到困难时，他们都不断地支持和鼓励我。在顶尖的科技公司里做全职工作需要不断地平衡工作与生活，而在这中间插入一部技术书籍则是一项艰巨的任务。在这个过程中，我不得不牺牲或忽略掉许多事情。我想对我的经理 Will Camp 和 Aravind Vijayakirthi 说声谢谢，感谢你们容忍我的疏漏并一直给予我指导。最后，我要致意并感谢我的妻子 Altaress，当我面对笔记本电脑陷入沉思时，她一直都在那儿陪着我和孩子们。

——Clifton Craig

前言

大约 5.3 亿年前，在那个地质学家称之为“寒武纪大爆发”的时代，包括现存所有门在内的大量物种在短短 100 万年的时间迅速出现——而这对于地质学时间来说仅仅是一眨眼工夫。科学家们对于此现象一直感到非常惊讶，达尔文自己都认为“寒武纪大爆发”发生如此之快，甚至让他的自然进化论产生了疑点。今天，我们正在经历着科技领域的“寒武纪大爆发”。美国劳工统计局预测现在的高中毕业生在其一生中将从事 11 份工作，而这种职业生涯短暂现象主要归因于科技的快速变化¹。

技术会衍生出更多的技术，而新的技术也正在加速发展。有些新技术几年之后依然会存在，但大多数将会消失。没有比投入时间和精力去掌握一门已经过时或者效用短暂的新技能更糟糕的事情了。我们编写本书是因为相信其中涵盖的工具和技术将会留存下来，而且值得你为之投入。

小即是美

摩尔定律是无情的，它指出 CPU 的处理能力大约每 18 个月会翻一倍。在过去几年里，笔记本电脑已经达到了与体积更大的桌面电脑相同的性能。笔记本电脑和平板电脑占据了 2014 年 PC 销售额的 81%²，而且销量还会继续增长，相反桌面电脑的销量将会下滑。没有哪个人或组织能够阻止或逆转这种趋势——这是经济规律的力量，是个体选择集聚的结果。笔记本电脑在未来大约 10 年将会是脑力工作者的首选工具。然而，一场悄无声息的革命正在发生，它将会很快推翻全能的笔记本电脑。大约到 2025 年，或者可能更早，我们的智能手机将会达到与笔记本电脑相同的性能——也就是说，更大体积的外形已经不再具备任何性能优势了。最终，我们的移动电脑(Mobile Computer, MC)将用于大多数计算应用，即使是那些你我现在认为只能在笔记本电脑上完成的应用。这场变革是可预见的，与推翻桌面电脑的变革一样确定。同时，你可以期望 MC(换句话说，智能手机或平板电脑)开始发挥出和笔记本电脑一样的功能，包括连接诸如键盘、显示器和鼠标等外围设备。

PC 时代即将结束，但 MC 时代其实更具个性化。不久，一系列新型可穿戴设计即将可用，例如手表、眼镜和鞋子。我们预见在不久后的某一天，我们将会在身上穿着自己的电脑，并在显示器、键盘和鼠标这些外围设备可用的地方连接它们。这将会是一个真正的

1 <http://online.wsj.com/news/articles/SB10001424052748704206804575468162805877990>。

2 来源：Forrester Research eReader Forecast, 2010 至 2015(美国)。

个人电脑时代，尽管我们不大可能再这样称呼它了。

Android 的优势

如果渴望成为一名 Android 开发者，那么你已经做出了极好的选择。在未来的 10 年里，这个不断发展的世界上的数十亿人都会来到线上。对于这些人中的大多数，他们的第一部电脑将会是智能手机，而这些智能手机中的大部分将会搭载 Android 系统³。我们的乐观是有充分原因的，因为已经有了大量可供推算的历史数据。Gartner Group 公司指出 2015 年将售出 12.5 亿部 Android 设备⁴。在撰写本书时，Android 独占了超过四分之三的中国市场份额⁵，而中国消费者在移动设备上的投入是惊人的，一些人会在新款移动设备上花费 70% 的月薪，因为互联是参与全球经济的先决条件⁶。在绝对数量上，中国是最大的市场，但我们能够在所有发展中国家观察到类似的趋势。更进一步，由于 Android 操作系统是开源免费的，因此它几乎总是电视游戏机、游戏系统、增强现实系统以及大量其他电子设备厂商的首选。

出于多种原因，Android 将继续巩固其在全球市场的主导地位。Android 的模块化架构允许进行各种各样的配置和定制。Android 设备标配的所有核心应用均可以被任意数量的第三方应用替代，这包括类似电话拨号器、电子邮件客户端、浏览器甚至操作系统导航器等应用。可用的 Android 设备有着各种各样令人惊讶的形状和功能，有 Android 增强现实眼镜、Android 游戏机(Ouya 是最著名的)、Android 手表、各种尺寸的 Android 平板电脑，当然还有 Android 智能手机。

Android 的核心技术毫不逊色于它的主要竞争者。Android 的包容性和开源许可证已经吸引了大量盟友，包括三星——全球最具创新性的公司之一。免费⁷和可定制的操作系统意味着 Android 设备厂商可以专注于向市场上推出优秀的产品，而高度竞争的 Android 设备市场也会持续生产出廉价、高质量且结构上开放的设备。

Android Studio 是革命性的

作为一名脑力劳动者，工具的选择极其重要。我们总是在寻求可以提高生产效率并能自动完成工作的工具。某些工具有着显而易见的好处，人们会立刻采纳它。Android Studio

3 <http://news.yahoo.com/android-projected-own-smartphone-market-next-fouryears-213256656.html>, <http://www.idc.com/getdoc.jsp?containerId=prUS24302813>。

4 www.bbc.co.uk/news/technology-25632430。

5 报告：Windows Phone 在意大利赶超 iOS，且在欧洲市场份额增加——The Next Web 网(时间未知)。取自 <http://thenextweb.com/insider/2013/11/04/report-windows-phone-overtakes-ios-in-italy-and-makes-progress-in-europe/#!pSdH1>。

6 报告：Windows Phone 在意大利赶超 iOS，且在欧洲市场份额增加——The Next Web 网(时间未知)。取自 <http://thenextweb.com/insider/2013/11/04/report-windows-phone-overtakes-ios-in-italy-and-makes-progress-in-europe/#!pSdH1>。

7 有一点很重要，需要注意，虽然 Google 放弃了 Android 的许可费，但是移动技术增值的整体趋势会提升 Google 的广告收入。

就是一款这样的工具。

2013年在Google I/O预发布Android Studio几天之后,我们就接触了它。在那以前,我们在专业领域和教学时均使用Android开发者工具(Android Developer Tool, ADT)。ADT是一个Android开发环境,内置于一个称为Eclipse的开源集成开发环境(IDE)中。虽然Android Studio仍处于早期预发布阶段,但我们已开始在专业领域内使用Android Studio。

Android Studio是JetBrains和Google合作的产物。Android Studio基于JetBrain的IntelliJ构建,因此其功能是IntelliJ的超集。能够使用IntelliJ做到的大多数事情,都可以在Android Studio中完成。Android Studio是革命性的,因为它流程化了Android开发过程并让Android开发比以前更加容易接近⁸。Android Studio目前是Android的官方IDE。

Android 工具生态系统

Android是一个有着自身工具生态系统支持的技术平台。紧随Android Studio之后,Git是Android生态系统中下一个最重要的工具。Git是一款分布式的源代码管理工具,它正在迅速成为标准,不仅是在移动开发领域,而是在整个软件工程领域。我们所有的移动开发项目均使用Git来做版本控制,没有例外。Git足可以用一本书来介绍,但幸运的是,你无须了解Git的所有功能就可以熟练地使用它。Android Studio包含优秀的、全功能的且已集成的Git工具,它有着令人印象深刻的图形用户界面。本书涵盖了成为高效Git用户所需要了解的特性,如果想要深入掌握关于这个不可或缺工具的知识,我们为你指出了进一步学习所需的资源。

Android生态系统中的另一个重要工具是Gradle。Gradle是一款类似于Ant和Maven的构建工具,它允许你管理库和库项目、运行仪器测试以及创建条件构建。Android Studio在库管理方面本身已经很不错了,但Gradle使得此项任务更加便捷。与Git一样,Gradle已经完全集成到Android Studio,而且拥有令人印象深刻的界面,使得用户能够图形化地检查Gradle文件并监测Gradle构建过程的输出。

Android 和 Java

如果在没有充分了解Java的情况下尝试在Android Studio中开发Android App,那么你将会遭遇挫折。出于诸多原因,Java是一门极其有用且流行的编程语言。或许Java流行的最重要原因是其内存托管机制。内存托管意味着程序员不必考虑释放堆内存,也不必担心内存泄漏。在内存托管环境中进行开发的程序员通常更高效,而且程序的运行时错误会更少。和Java类似,Android也是一种内存托管的编程环境。托管内存被证明是一个非常好

⁸ 开发Android App需要对Java有深入的了解。像Android这样强大的系统不会很容易掌握,但使用Android Studio会让开发Android App的任务变得轻松些。

的思路，使得微软和苹果公司均在各自的移动开发平台中采用了此模型⁹。

从 ADT/Eclipse 转型

如果你是一位有经验的 Android 开发者而且习惯使用 ADT 编程，那么你会收获惊喜。幸运的是，所有 SDK 工具(例如 DDMS 和 Hierarchy Viewer)都依然可用，你将发现在 Android Studio 中很容易就可以访问到它们。如果你是一位 ADT 用户，那么你可能需要不断地清理并重新构建项目，以便资源能够与源代码同步(可怕的 R.java 同步错误)。在使用 Android Studio 的这些日子里，我们一直没有被这个问题困扰过。如果你是一位有经验的 ADT 用户，那么为了提升使用 Android Studio 的效率，你将需要学习一些键盘快捷键、熟悉 Gradle 以及适应 Android Studio 的表现逻辑。总而言之，享受 Android Studio 带来的强大功能和乐趣只需付出很小的代价。

本书约定

Android Studio 在操作系统之间保持了高度的一致性。事实上，Windows 和 Linux 中的用户界面几乎就是一样的。不过，Mac OS 用户将会发现部分菜单的位置和键盘快捷键是不同的。当涉及需要系统导航的内容时，我们使用 Windows。不过，当给出键盘快捷键时，我们将同时包含 Windows-Linux 和 Mac 快捷键，中间使用竖线分隔(例如，Ctrl+K | Cmd+K)。我们会适时为 Mac 用户给出提示、链接和其他资源。

源代码

读者在学习本书中的示例时，可以手动输入所有的代码，也可以使用本书附带的源代码文件。本书使用的所有源代码都可以从 <http://www.apress.com> 下载，还可访问 www.tupwk.com.cn/downpage 来下载源代码。下载代码后，只需要用自己喜欢的解压缩软件进行解压缩即可。

⁹ Xcode——用于开发 iOS App 的 IDE，最近引入了一项称为“自动引用计数(Automatic Reference Counting)”的特性，使得编译器能够生成自动管理内存的代码。C#就是一种受到 Java 启发的内存托管的编程环境。

目 录

第 1 章	Android Studio 入门	1
1.1	在 Windows 上安装 Java 开发工具包	1
1.1.1	在 Windows 上下载 JDK	2
1.1.2	在 Windows 上执行 JDK 向导	3
1.1.3	配置 Windows 环境变量	4
1.2	在 Mac 上安装 Java 开发工具包	7
1.2.1	在 Mac 上下载 JDK	7
1.2.2	在 Mac 上执行 JDK 向导	8
1.2.3	在 Mac 上配置 JDK 版本	9
1.3	安装 Android Studio	10
1.4	创建第一个项目: HelloWorld	12
1.5	使用 Android 虚拟设备管理器	15
1.6	在 AVD 上运行 HelloWorld	16
1.7	在 Andriod 设备上运行 HelloWorld	17
1.8	小结	19
第 2 章	在 Android Studio 中遨游	21
2.1	编辑器	22
2.1.1	Editor 选项卡	22
2.1.2	折叠线	23
2.1.3	标记栏	23
2.1.4	工具按钮	23
2.1.5	默认布局	24
2.2	导航工具窗口	24
2.2.1	Project 工具窗口	25

2.2.2	Stucture 工具窗口	26
2.2.3	Favorites 工具窗口	26
2.2.4	TODO 工具窗口	27
2.2.5	Commander 工具窗口	27
2.3	主菜单栏	27
2.4	工具栏	28
2.5	导航栏	28
2.6	状态栏	28
2.7	常用操作	29
2.7.1	选择文本	29
2.7.2	使用 Undo 和 Redo	29
2.7.3	找到最近的文件	30
2.7.4	遍历最近的导航操作	30
2.7.5	剪切、复制和粘贴	30
2.8	上下文菜单	31
2.9	获取帮助	32
2.10	使用键盘导航	32
2.10.1	Select In 命令	32
2.10.2	Class 命令	33
2.10.3	File 命令	33
2.10.4	Line 命令	33
2.10.5	Related File 命令	33
2.10.6	Last Edit Location 命令	33
2.10.7	Type Hierarchy 命令	34
2.10.8	Declaration 命令	34
2.11	查找和替换文本	34
2.11.1	Find 命令	34
2.11.2	Find in Path 命令	34
2.11.3	Replace 命令	35
2.11.4	Replace in Path 命令	35
2.12	小结	35

第 3 章 在 Android Studio 中编程37

- 3.1 使用代码折叠37
- 3.2 执行代码补全39
- 3.3 注释代码42
- 3.4 使用代码生成42
 - 3.4.1 构造函数43
 - 3.4.2 getter/setter44
 - 3.4.3 重载方法44
 - 3.4.4 toString()方法45
 - 3.4.5 代理方法46
- 3.5 插入动态模板47
- 3.6 移动代码48
- 3.7 设计代码风格50
 - 3.7.1 Auto-Indent Lines 选项51
 - 3.7.2 Rearrange Code 选项51
 - 3.7.3 Reformat Code 选项52
 - 3.7.4 Surround With52
- 3.8 小结53

第 4 章 重构代码55

- 4.1 重命名56
- 4.2 修改签名57
- 4.3 类型迁移58
- 4.4 移动58
- 4.5 复制59
- 4.6 安全删除60
- 4.7 抽取60
 - 4.7.1 抽取变量61
 - 4.7.2 抽取常量61
 - 4.7.3 抽取字段62
 - 4.7.4 抽取参数62
 - 4.7.5 抽取方法63
- 4.8 高级重构65
 - 4.8.1 下推成员和上拉成员65
 - 4.8.2 使用代理代替继承66
 - 4.8.3 封装字段67
 - 4.8.4 封装方法返回值68
 - 4.8.5 使用工厂方法代替构造函数69
 - 4.8.6 将匿名类转换为内部类69

4.9 小结70

第 5 章 备忘录实验：第 1 部分71

- 5.1 启动新项目73
- 5.2 初始化 Git 仓库74
- 5.3 构建用户界面77
 - 5.3.1 使用可视化设计器78
 - 5.3.2 编辑布局的原始 XML78
 - 5.3.3 添加视觉增强效果83
 - 5.3.4 向 ListView 添加条目84
 - 5.3.5 设置操作栏溢出菜单86
- 5.4 持久化备忘录87
 - 5.4.1 数据模型87
 - 5.4.2 SQLite API89
- 5.5 小结95

第 6 章 备忘录实验：第 2 部分97

- 6.1 添加/删除备忘97
- 6.2 响应用户交互100
- 6.3 提供多选上下文菜单102
 - 6.3.1 兼容较早的 SDK104
 - 6.3.2 添加上下文操作模式105
- 6.4 实现添加、编辑和删除107
 - 6.4.1 设计自定义对话框108
 - 6.4.2 将设计转换为代码108
 - 6.4.3 创建自定义对话框110
 - 6.4.4 添加自定义图标113
- 6.5 小结115

第 7 章 Git 入门117

- 7.1 安装 Git117
- 7.2 忽略文件119
- 7.3 添加文件120
- 7.4 克隆参考 App: Reminders121
 - 7.4.1 分叉和克隆121
 - 7.4.2 使用 Git 日志124
 - 7.4.3 分支125
- 7.5 在分支上开发125
 - 7.5.1 Git 提交和分支131
 - 7.5.2 回退在哪里?132

7.5.3 合并.....	136	第 10 章 货币实验：第 2 部分.....	225
7.5.4 Git 重置修改历史.....	138	10.1 定义 MainActivity 的成员	225
7.5.5 Git 变基.....	142	10.2 从 bundle 中解压出货币	
7.5.6 分离头部.....	144	代码.....	226
7.5.7 相对引用.....	146	10.3 创建选项菜单.....	227
7.5.8 在变基时解决冲突.....	148	10.4 实现选项菜单行为.....	229
7.5.9 Git 远端.....	153	10.5 创建 spinner_closed 布局.....	230
7.6 小结.....	154	10.6 将 mCurrencies 绑定到选择	
第 8 章 设计布局	155	列表.....	231
8.1 Activity.....	155	10.7 将选择列表行为代理给	
8.2 View 和 ViewGroup.....	156	MainActivity	232
8.2.1 预览面板	157	10.8 创建偏好管理器.....	234
8.2.2 宽度和高度	159	10.9 根据给定代码查找位置.....	235
8.2.3 设计器模式	161	10.10 从货币字符串中抽取代码.....	236
8.2.4 帧布局.....	161	10.11 实现共同偏好	237
8.2.5 线性布局.....	164	10.12 按钮单击行为.....	239
8.2.6 相对布局.....	166	10.13 保存开发者密钥.....	240
8.2.7 嵌套布局.....	169	10.14 获取开发者密钥.....	241
8.2.8 列表视图.....	173	10.15 CurrencyConverterTask.....	242
8.3 布局设计指导原则.....	180	10.15.1 onPreExecute()	246
8.3.1 覆盖各种显示尺寸.....	180	10.15.2 doInBackground()	246
8.3.2 组合在一起	183	10.15.3 onPostExecute()	246
8.4 Fragment	190	10.16 按钮选择器.....	247
8.5 小结.....	200	10.17 启动图标.....	248
第 9 章 货币实验：第 1 部分.....	201	10.18 小结.....	249
9.1 Currencies 规范	201	第 11 章 测试和分析	251
9.2 初始化 Git 仓库.....	205	11.1 创建新的仪器测试.....	251
9.3 修改 MainActivity 的布局	207	11.1.1 定义 SetUp()和 TearDown()	
9.4 定义颜色.....	211	方法.....	252
9.5 为布局应用颜色.....	212	11.1.2 在 MainActivity 中定义	
9.6 创建并应用样式.....	213	回调.....	254
9.7 创建 JSONParser 类.....	216	11.1.3 定义一些测试方法	255
9.8 创建启动界面.....	217	11.1.4 运行仪器测试.....	257
9.9 获取 JSON 格式的活动货币		11.1.5 修改 Bug.....	258
代码.....	220	11.2 使用 Monkey.....	259
9.10 启动 MainActivity	223	11.3 使用分析工具.....	260
9.11 小结.....	224	11.3.1 检查代码.....	260
		11.3.2 分析依赖	261

11.3.3	分析栈轨迹	262	14.2.2	方法跟踪工具	324
11.4	小结	264	14.2.3	分配跟踪器	325
第 12 章	调试	265	14.2.4	屏幕抓取	325
12.1	日志	265	14.3	导航编辑器	327
12.1.1	使用 logcat	266	14.3.1	设计用户界面	328
12.1.2	写入 Android 日志	268	14.3.2	导航编辑器初步	328
12.2	捕捉 Bug!	268	14.3.3	连接 Activity	330
12.2.1	使用交互式调试器	272	14.3.4	编辑菜单	331
12.2.2	表达式求值	275	14.4	终端	333
12.2.3	使用栈轨迹	277	14.4.1	查询设备	333
12.2.4	探索交互式调试的工具		14.4.2	安装 APK	333
	窗口	280	14.4.3	下载文件	333
12.2.5	使用断点浏览器	281	14.4.4	上传文件	333
12.2.6	条件断点	283	14.4.5	端口转发	334
12.3	小结	285	14.5	Google 云工具	334
第 13 章	Gradle	287	14.5.1	创建 HelloCloud 前端	335
13.1	Gradle 语法	288	14.5.2	创建 Java 后台模块	337
13.2	IntelliJ 核心构建系统	289	14.5.3	组合在一起	339
13.3	Gradle 构建概念	290	14.5.4	部署到 App Engine	343
13.3.1	Gradle Android 结构	290	14.6	小结	346
13.3.2	项目依赖	291	第 15 章	Android 可穿戴设备实验	347
13.4	案例研究: 使用 Gradle 的		15.1	设置可穿戴设备环境	347
	天气预报项目	292	15.1.1	安装设备驱动程序	347
13.5	Android 库依赖	299	15.1.2	设置 SDK 工具	350
13.5.1	Java 库依赖	303	15.1.3	设置可穿戴虚拟设备	350
13.5.2	第三方库	311	15.1.4	设置 Android 可穿戴设备	
13.6	打开较旧的项目	313		硬件	353
13.7	小结	314	15.2	创建 MegaDroid 项目	353
第 14 章	更多 SDK 工具	315	15.2.1	针对屏幕的优化技术	355
14.1	Android 设备监视器	315	15.2.2	构建 watch-face 服务	356
14.1.1	线程监视器	316	15.2.3	初始化可绘制资源和	
14.1.2	堆监视器	317		样式	358
14.1.3	分配跟踪器	318	15.2.4	管理手表更新	359
14.1.4	网络统计	319	15.2.5	绘制界面	363
14.1.5	层次查看器	320	15.3	小结	367
14.2	Android 监视器	323	第 16 章	定制 Android Studio	369
14.2.1	内存监视器	323	16.1	代码风格	370
			16.2	外观、颜色和字体	372

16.3 键盘映射.....	374	16.7 插件.....	378
16.4 宏.....	375	16.8 小结.....	380
16.5 文件和代码模板.....	375		
16.6 菜单和工具栏.....	377		

Android Studio 入门

本章带领你安装并配置开发环境，以便你能够跟随本书中的示例和实验进行学习。首先，你将要安装一个重要的前置组件，称为 Java 开发工具包(Java Development Kit, JDK)。接着你将下载并安装 Android Studio，以及构建 Android App 所需的一套软件工具——Android 软件开发包(Software Development Kit, SDK)。我们将为你展示如何使用 New Project Wizard 创建一个名为 HelloWorld 的简单项目。最后，将向你展示如何建立与 Android 虚拟设备(Android Virtual Device, AVD)和 Android 物理设备的连接。到本章结束时，你应该会拥有开始在 Android Studio 中开发 App 所需的所有东西。

1.1 在 Windows 上安装 Java 开发工具包

本节主要面向 Windows 用户。如果你是 Mac 用户，那么跳到标题为“在 Mac 上安装 Java 开发包”一节。Android Studio 使用 Java 工具链构建，因此在开始使用 Android Studio 之前，你需要确保已经在自己的电脑上安装了 Java 开发工具包(JDK)。如果你是资深的 Android 或 Java 开发者，那么你的电脑上很有可能已经安装 JDK 了。如果电脑上已经安装了 JDK，而且正在运行着 JDK 1.6 或更高版本，那么可以跳过此节。不过，你可能还是需要下载、安装并配置最新的 JDK。可以从下面的 Oracle 站点下载 JDK：

www.oracle.com/technetwork/java/javase/downloads/index.html

当你到达这个页面时，单击 Java Download 按钮，如图 1-1 所示。

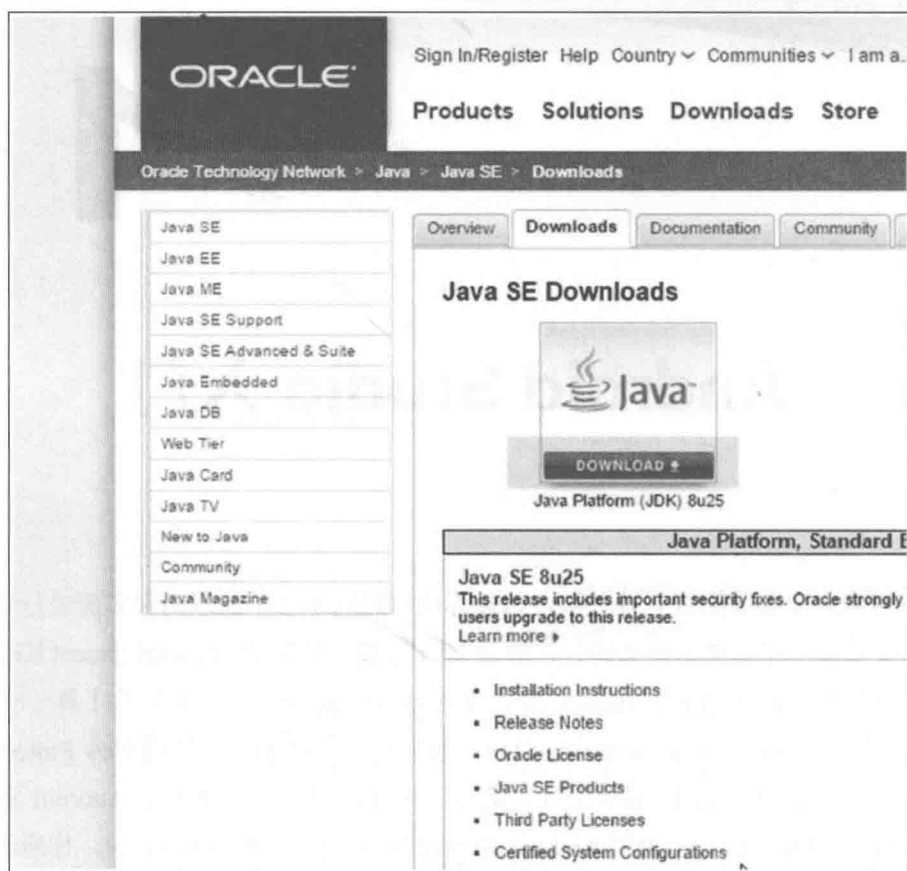


图 1-1 Java 下载页面中的 Java Download 按钮

1.1.1 在 Windows 上下载 JDK

如图 1-2 所示, 安装的下一步要求你选中 **Accept License Agreement** 单选按钮以接受许可协议。接着, 你需要选择适合于自己操作系统的 JDK。如果正在使用 Windows 7 或 Windows 8, 那么应该点击 **Windows x64** 标签右侧的文件链接, 仍如图 1-2 所示。Oracle 经常会发布 JDK 的更新版本。到本书出版的时候, JDK 应该已经有了更新的版本, 因此务必下载最新版。等待安装文件下载完成。这个文件的大小通常约为 125MB, 所以下载过程不会花费太长时间。

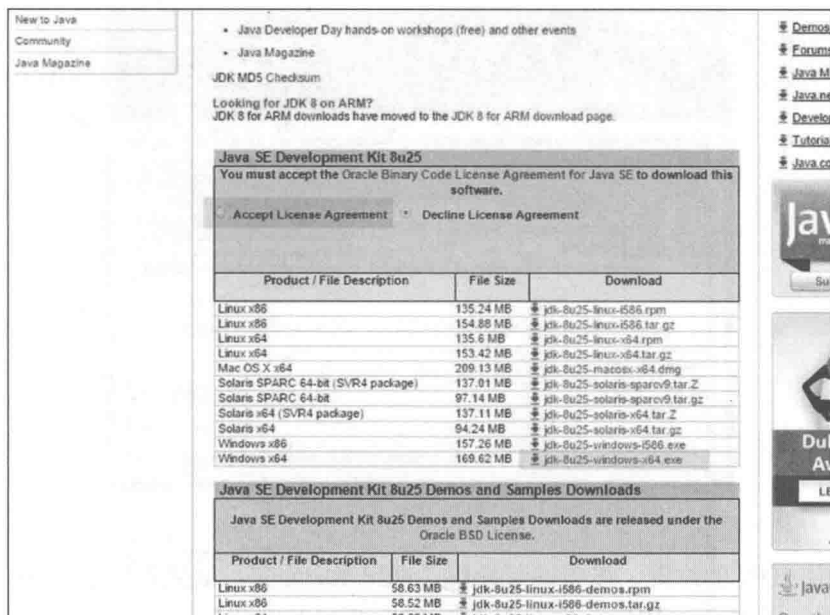


图 1-2 接受许可协议并点击适用于 Windows 的链接

1.1.2 在 Windows 上执行 JDK 向导

安装 JDK 之前，在 C 盘根目录下创建名为 Java 的目录。此目录可以采用任意名称，命名为 Java 是因为我们将要在这里安装的大多数工具都与 Java 相关，包括 JDK、Android Studio 和 Android SDK。将与 Android Studio 相关的工具统一安装在 C:\Java 目录中还可以让你的开发环境保持组织有序。

导航到浏览器下载安装文件的位置，并双击执行该文件。一旦安装开始，你将会看到 Installation Wizard，如图 1-3 所示。在 Windows 中，JDK 安装程序的默认路径为 C:\Program Files\Java\。要更改安装目录的位置，可单击 Change 按钮。我们建议将 JDK 安装在 C:\Java 目录，因为此路径中不包含空格且易于记忆，参见图 1-4。



图 1-3 Windows 中的 JDK 安装向导

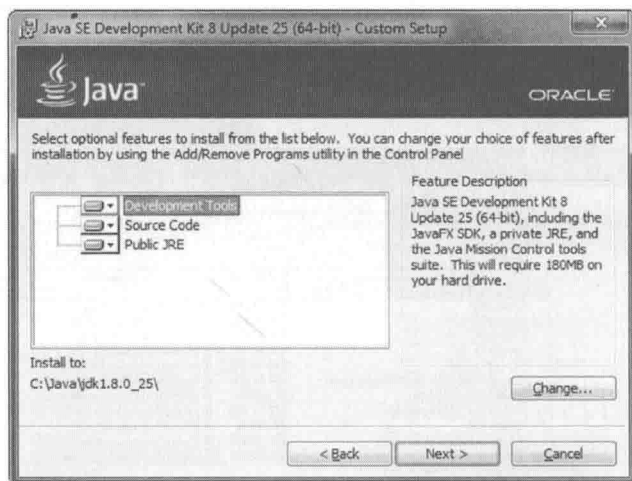


图 1-4 选择 JDK 的安装目录

记住安装 JDK 的位置。按照提示操作，直到完成安装。如果提示安装 Java 运行时版本(Java Runtime Edition, JRE)，那么选择与 JDK 相同的安装目录。

1.1.3 配置 Windows 环境变量

本节说明如何配置 Windows，以便 Android Studio 能够找到 JDK。在运行 Windows 的电脑上，按住 Windows 键并单击 Pause 键，打开 System 窗口。单击 Advanced system settings 选项，如图 1-5 所示。

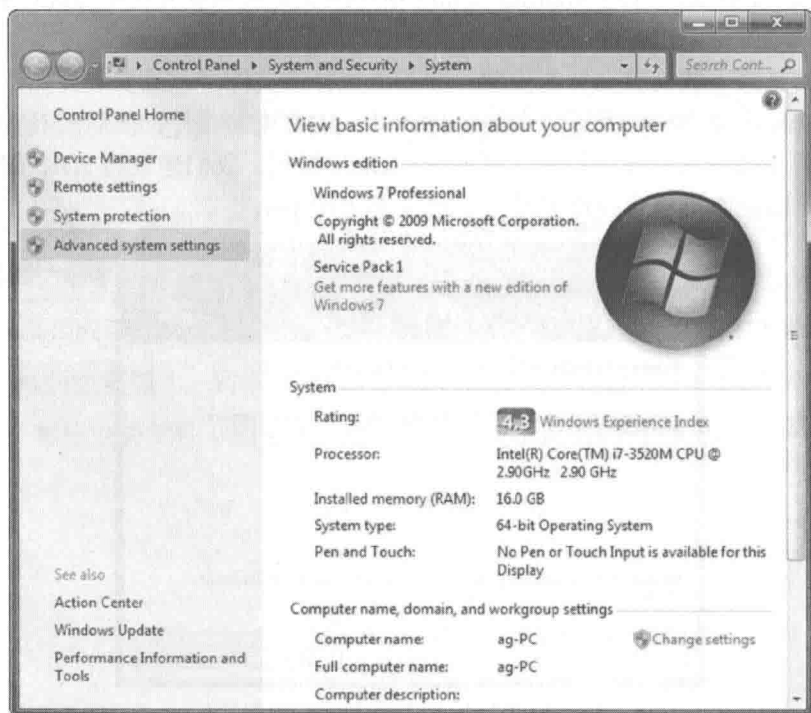


图 1-5 Windows 系统窗口

单击 Environmental Variables 按钮, 如图 1-6 所示。在底部列出的 System variables 列表中, 如图 1-7 所示, 导航至 JAVA_HOME 项。如果 JAVA_HOME 项不存在, 单击 New 按钮创建它。否则, 单击 Edit 按钮。



图 1-6 系统属性

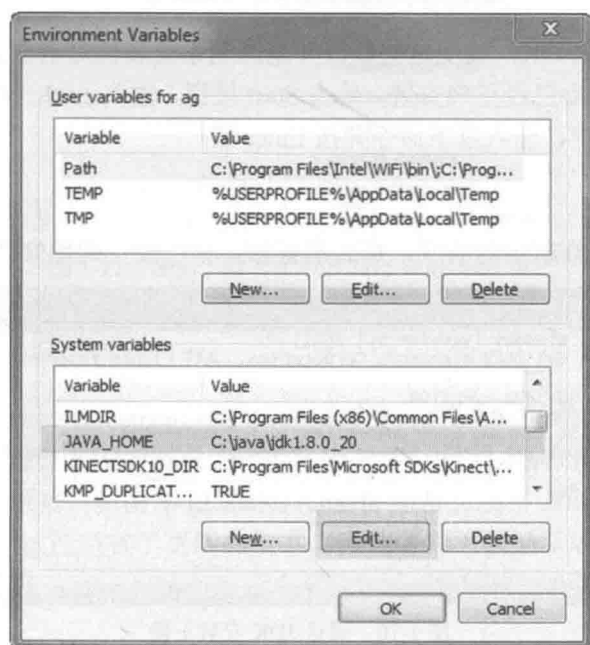


图 1-7 环境变量

单击 New 或 Edit 按钮会显示一个类似于图 1-8 所示的对话框。务必在 Variable name 文本框中输入 JAVA_HOME。在 Variable value 文本框中, 输入之前安装 JDK 的位置(没有任何尾随斜线), 如图 1-4 所示。现在请单击 OK。

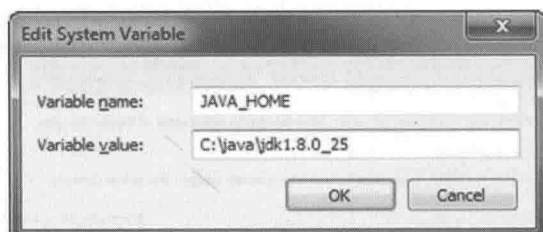


图 1-8 编辑 JAVA_HOME 环境变量

与之前处理 JAVA_HOME 环境变量的方式相同, 需要编辑 PATH 环境变量, 参见图 1-9。将光标置于 Variable value 文本框中内容的末尾并输入以下内容:

```
;%JAVA_HOME%\bin
```

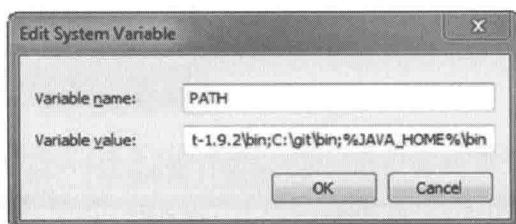


图 1-9 编辑 PATH 环境变量

现在连续三次单击 OK, 接受这些修改并返回到系统属性对话框。

为了测试新的 JDK 已经正确安装, 单击 Start 按钮, 输入 cmd, 接着按 Enter 键打开命令行。在命令行窗口中, 输入以下命令并按 Enter 键:

```
java -version
```

如果得到如图 1-10 所示的响应, 那么恭喜你, 你已经正确地安装了 JDK。

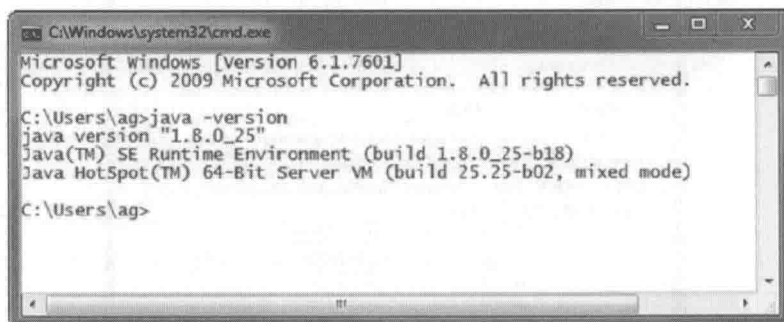


图 1-10 确认 JDK 安装正确

1.2 在 Mac 上安装 Java 开发工具包

在 Mac 上安装 JDK 的前两步与在 Windows 上相同。在浏览器中打开以下站点：

`www.oracle.com/technetwork/java/javase/downloads/index.html`

当到达这个页面时，单击 Java Download 按钮，如图 1-11 所示。

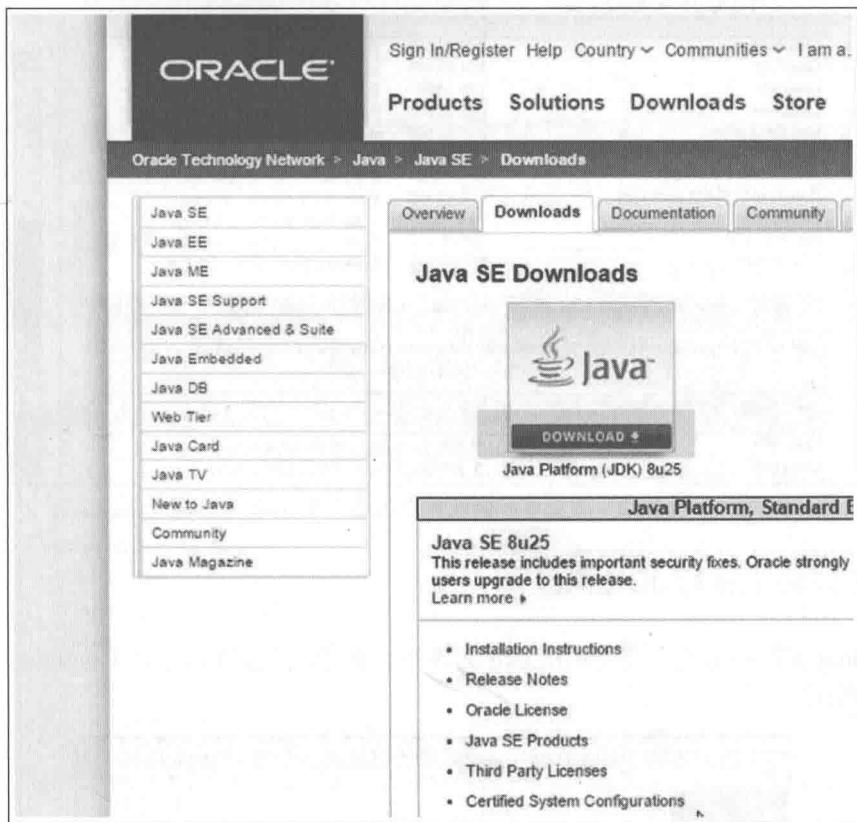


图 1-11 Java 下载页面中的 Java Download 按钮

1.2.1 在 Mac 上下载 JDK

选中 Accept License Agreement 单选按钮以接受许可协议，如图 1-12 所示。接下来，需要选择适合于自己操作系统的 JDK。如果正在使用 64 位版本的 OS X，应该点击 Mac OS X64 标签右侧的文件链接，如图 1-12 所示。Oracle 经常会发布 JDK 的更新版本。到本书出版的时候，JDK 应该已经有了更新的版本，因此务必下载最新版。等待安装文件下载完成。

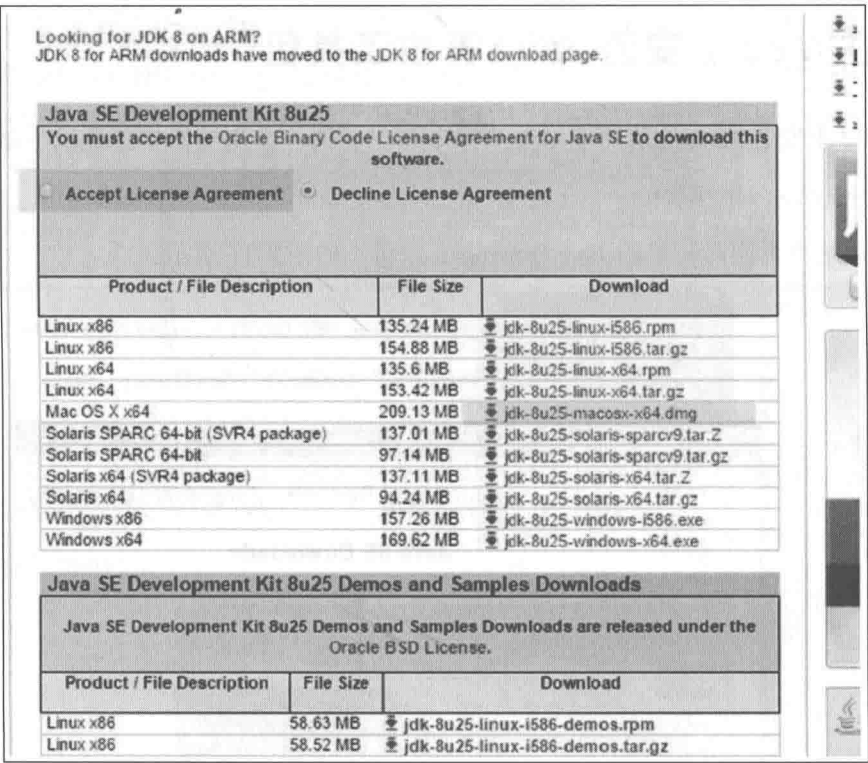


图 1-12 接受许可协议并点击适用于 Mac 的链接

1.2.2 在 Mac 上执行 JDK 向导

双击.dmg 文件执行它。现在单击.pkg 文件开启向导，并按要求单击 Continue，如图 1-13 至图 1-15 所示。

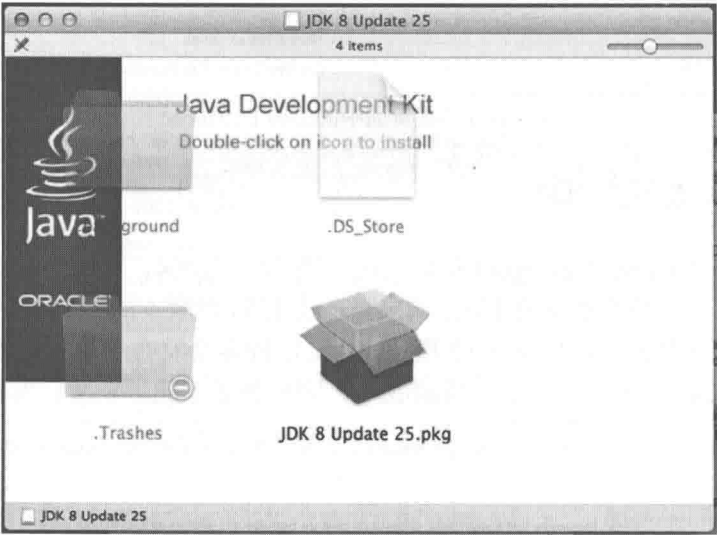


图 1-13 JDK 8 Update 25.pkg

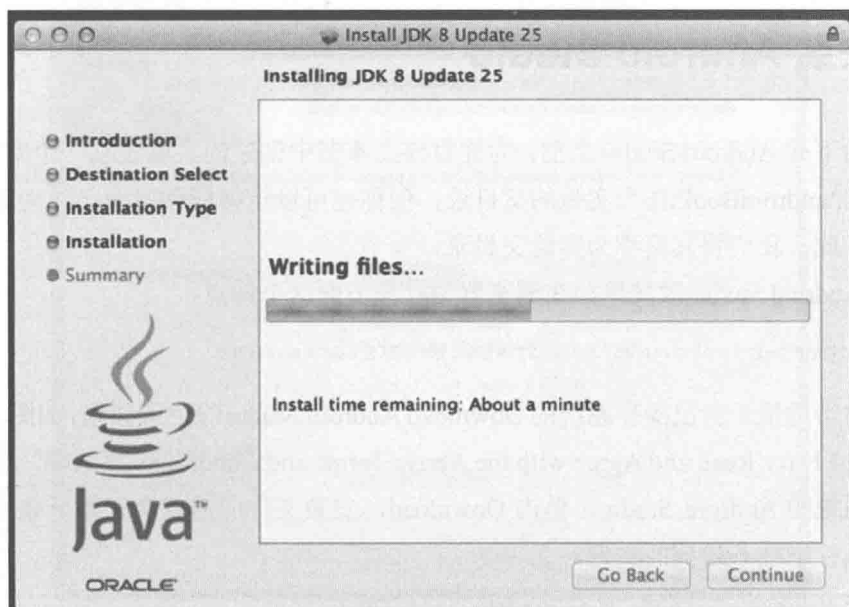


图 1-14 安装向导

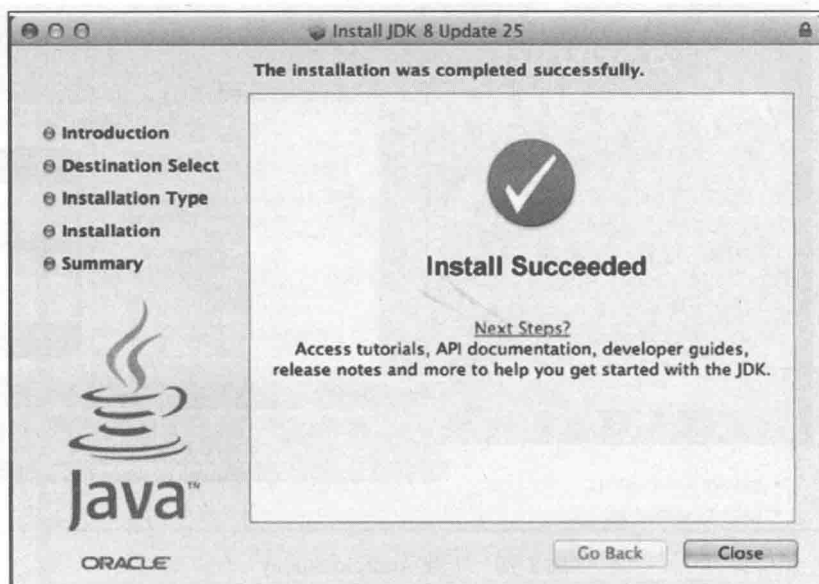


图 1-15 安装成功

1.2.3 在 Mac 上配置 JDK 版本

要对 Mac 进行配置，以便 Android Studio 能够找到正确的 JDK，可打开 Finder 窗口并选择 Applications | Utilities。在这里，打开 Java Preferences，并按提示将新版本拖至列表的顶部，以便将其识别为预设版本。

1.3 安装 Android Studio

在开始下载 Android Studio 之前, 为你将要在本书中创建的实验建立一个实验父目录。我们使用 C:\androidBook\ 作为实验的父目录, 但你也可以选择或创建自己认为适合的任意目录。鉴于此, 我们将其简称为实验父目录。

下载 Android Studio 很简单。在浏览器中打开下面这个站点:

developer.android.com/sdk/installing/studio.html

现在单击适用于自己操作系统的 Download Android Studio 绿色大按钮, 如图 1-16 所示。接着, 选中 I Have Read and Agree with the Above Terms and Conditions 复选框。选择适用于自己操作系统的 Android Studio, 单击 Download, 安装文件应该就会开始下载。一旦下载完毕, 就执行刚刚下载到的文件。



图 1-16 下载 Android Studio

在 Installation Wizard 启动后, 单击 Next 按钮向前推进界面, 一直到达 Choose Components 界面。在这里, 选中所有组件复选框, 如图 1-17 所示。接下来, 单击 Next。再次接受各种条款。当到达 Configuration Settings: Install Locations 界面时, 如图 1-18 所示, 选择 Android Studio 和 Android SDK 的位置。为保持一致性, 我们选择在 C:\Java\astudio\ 下安装 Android Studio, 而在 C:\Java\asdk\ 下安装 Android SDK。



图 1-17 选择组件

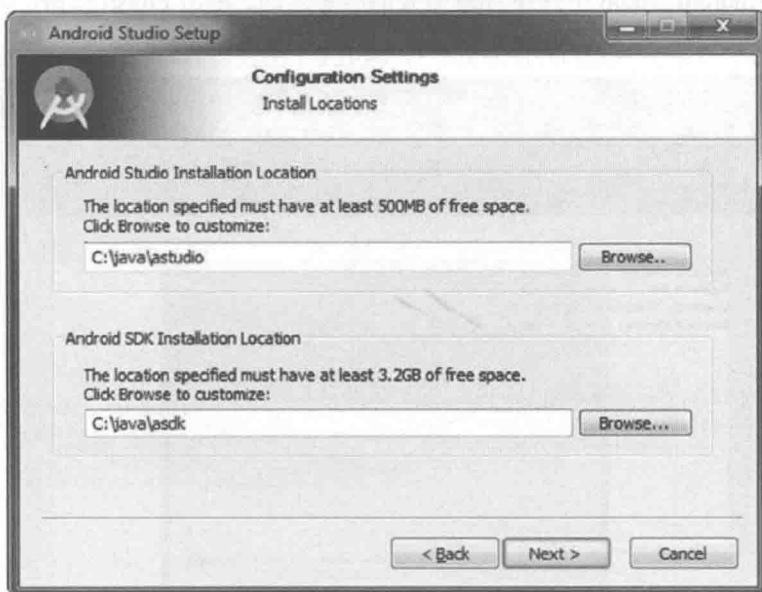


图 1-18 为 Android Studio 和 SDK 选择位置

连续单击几个 Next 按钮，安装 Android Studio 和 Android SDK。最后你应该会到达 Completing the Android Studio Setup 界面，如图 1-19 所示。Start Android Studio 复选框能够让 Android Studio 在单击 Finish 之后启动。确保选中了复选框，接着继续单击 Finish，Android Studio 将会启动。请注意从此之后，将需要通过桌面图标或 Start 菜单来启动 Android Studio。



图 1-19 完成 Android Studio 的安装

当 Android Studio 第一次启动时,如图 1-20 所示的 Installation Wizard 将会分析你的系统,查找已有 JDK(例如,之前安装的那个)以及 Android SDK 的位置。Installation Wizard 应该会下载在 Android Studio 中开发 App 需要的所有东西。单击 Finish 按钮,关闭 Installation Wizard。

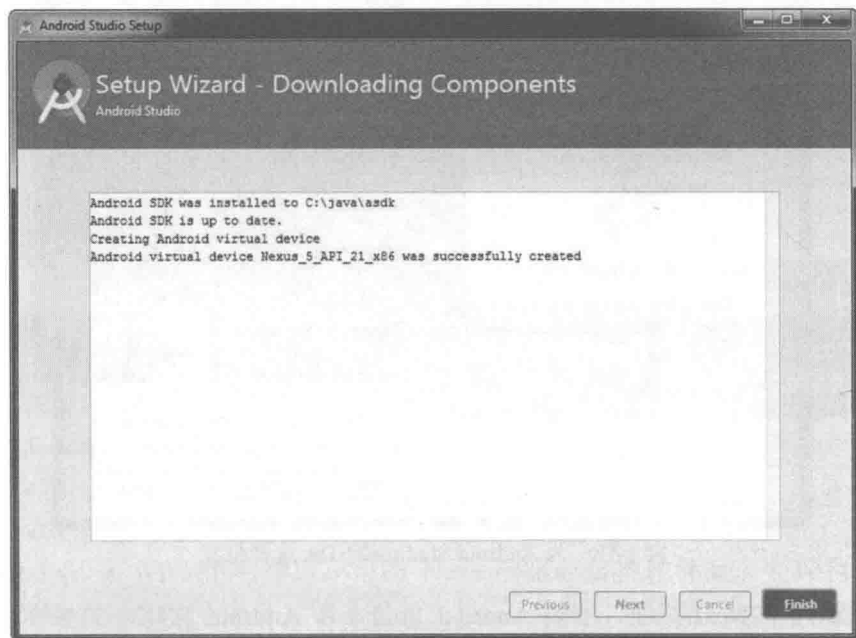


图 1-20 安装向导——下载组件

1.4 创建第一个项目: HelloWorld

一旦完成 Installation Wizard, Welcome to Android Studio 对话框就会出现,如图 1-21 所示,单击 Start a New Android Project 选项。

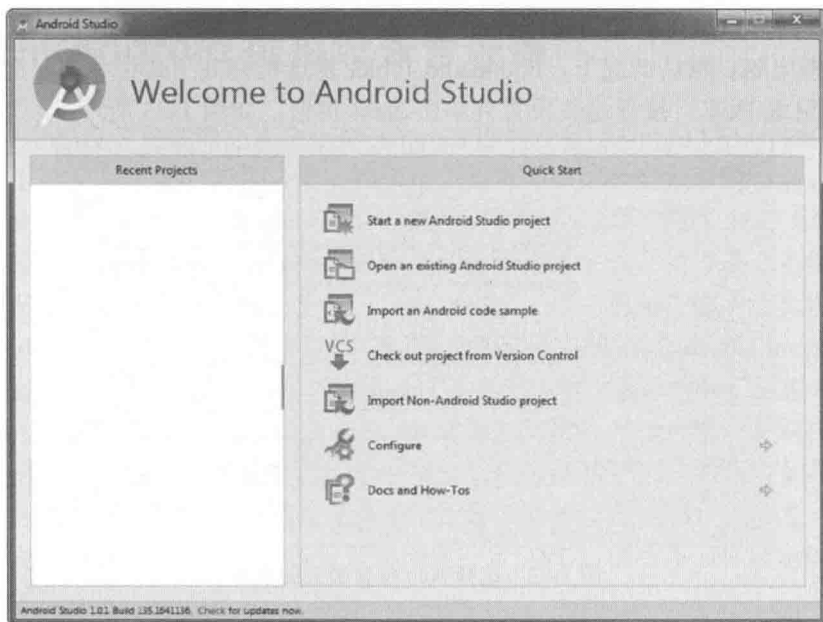


图 1-21 欢迎进入 Android Studio

在出现的 New Project 向导(参见图 1-22)中,在 Application name 文本框中输入 HelloWorld 并在 Company Domain 文本框中输入 gerber.apress.com。注意,包名是公司域名的倒序加上项目的名称。在实验父目录下安装 HelloWorld 项目。如前所述,如果正在运行 Windows,就使用 C:\androidBook\。如果正在使用 Mac 或 Linux,那么实验父目录名将不会以字母开头,而是以斜线开头。



图 1-22 配置新项目

Android 操作系统可以在许多平台上运行，包括游戏控制台、电视、手表、眼镜、智能手机和平板电脑。默认情况下，Phone and Tablet 复选框将处于选中状态，而且 API-8 将被选为最低 SDK 版本。接受这些设置并单击 Next 按钮，如图 1-23 所示。

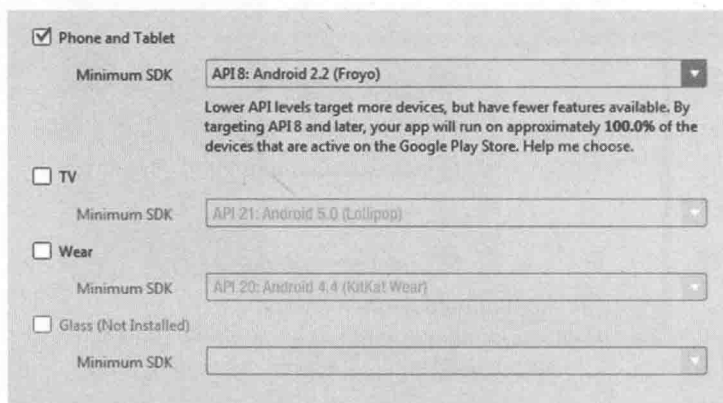


图 1-23 选择 App 将要面向的设备

New Project Wizard 中的后续界面会提示你选择布局。选择 Blank Activity 并单击 Next 按钮。接受默认的名称，如图 1-24 所示，其他设置如下所示：

- Activity 名称: MainActivity
- 布局名称: activity_main
- 标题: MainActivity
- 菜单资源名称: menu_main

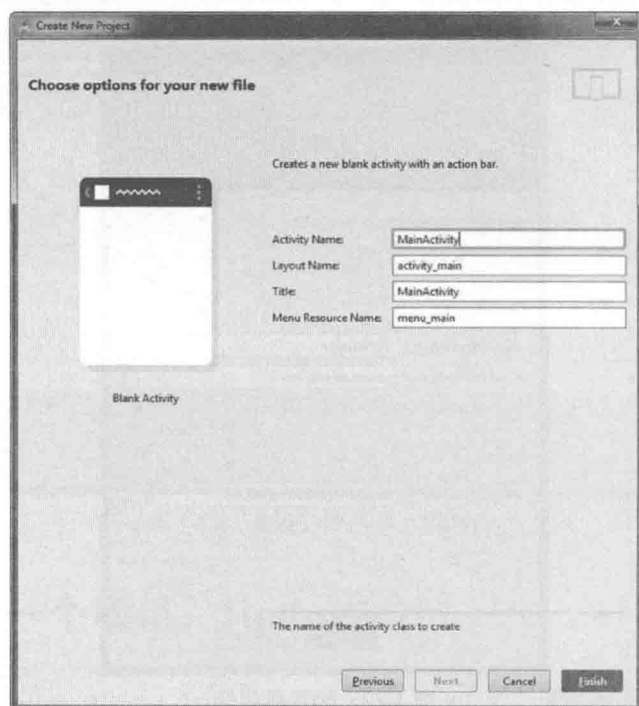


图 1-24 设置新文件的属性

1.5 使用 Android 虚拟设备管理器

Android 虚拟设备管理器允许你创建 Android 虚拟设备(Android Virtual Device, AVD), 进而在电脑上运行它来模拟设备。模拟和仿真之间有一个细微却重要的差别。模拟意味着虚拟设备仅是表象, 它模仿实际物理设备可能的行为, 但并不运行目标操作系统。iOS 开发环境使用的模拟方式, 对于该平台有限种类的可用设备来说, 或许是个不错的选择。

然而, 对于仿真来说, 电脑会另辟一块内存出来, 用于重新构造所仿真设备的环境。Android Studio 使用仿真, 这意味着 Android 虚拟设备管理器会启动 Linux 内核和整个 Android 栈的沙箱版, 以便仿真物理 Android 设备上的环境。尽管相对于模拟来说, 仿真为测试 App 提供了更真实的环境, 但启动 AVD 却需要花费数分钟时间, 且取决于电脑的运行速度。好消息是模拟器在内存中激活之后就能够保持快速响应。然而, 如果有 Android 手机或平板电脑的话, 我们还是建议使用物理设备, 而非使用 AVD 来测试 App。即便如此, 让我们先使用 Android 虚拟设备管理器创建一台 AVD, 稍后再向你展示如何连接物理设备(如果有的话)。

单击图 1-25 中圈出的 Android 虚拟设备管理器图标。在 Android 虚拟设备管理器向导的第一个界面中, 单击 Create Virtual Device 按钮。在接下来的界面中, 如图 1-26 所示, 选择 Galaxy Nexus 并单击 Next 按钮。下一个界面如图 1-27 所示, 允许你选择系统镜像。选择第一个选项 Lollipop(或者最新的 API), 将 ABI 选为 x86_64。单击 Next 按钮。在接下来的界面中, 单击 Finish 按钮来验证你的 AVD 设置。恭喜, 你刚刚创建了一台新的 AVD。



图 1-25 AVD 图标

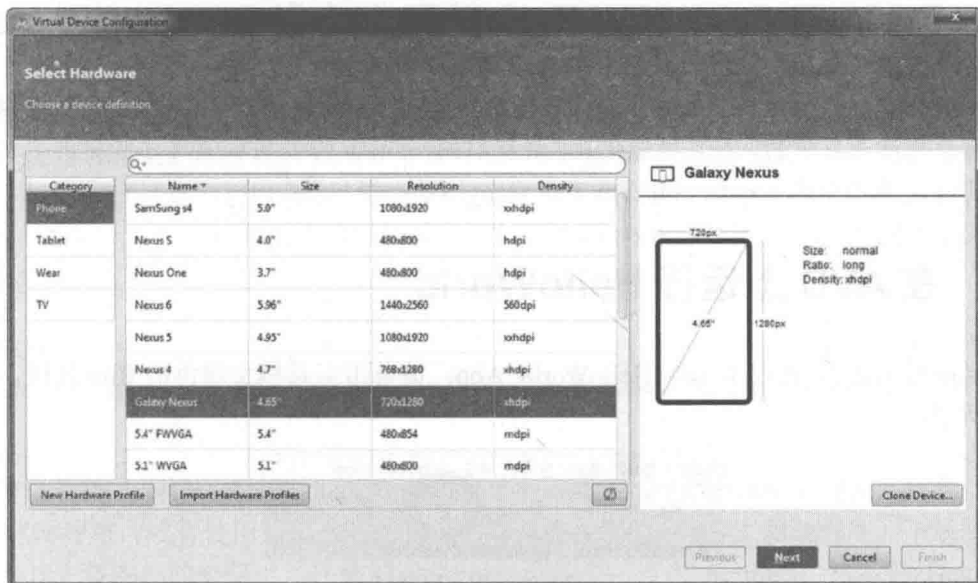


图 1-26 选择 Galaxy Nexus 硬件

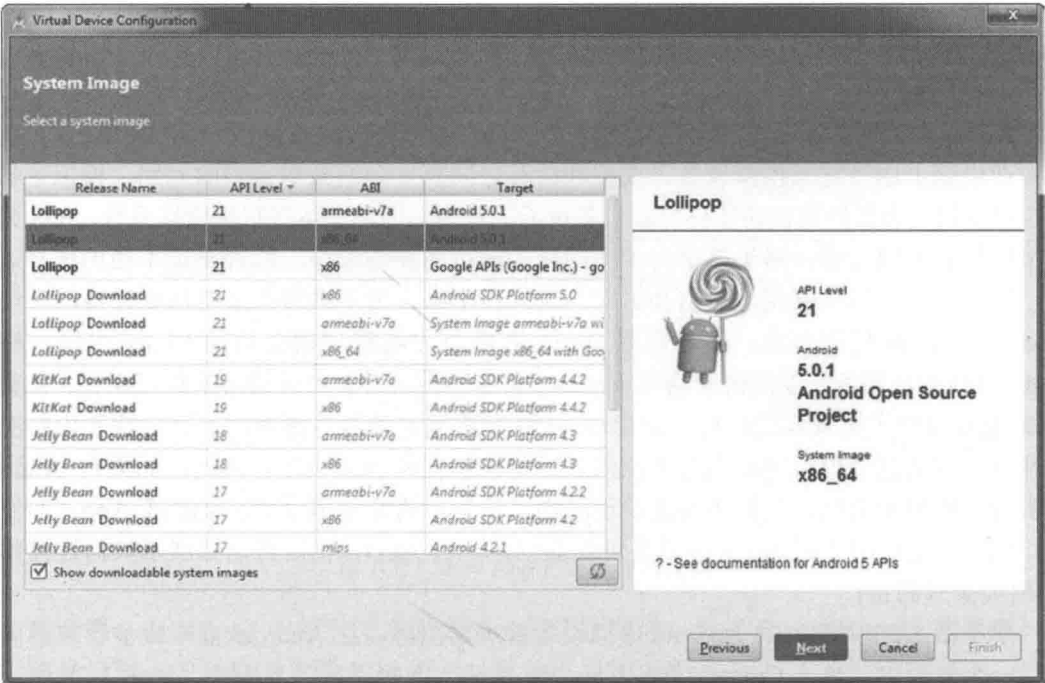


图 1-27 选择 x86_64 系统镜像

注意

x86_64 版本需要 Intel 硬件加速，它只能在有限数量的 Intel 芯片组上工作。如果无法安装 x86_64，就试着安装 armeabi-vxx 版。

提示

如果想要为 Android Studio 尚未包含其定义的设备创建 AVD 的话，我们推荐从网址 phonearena.com 查找你需要的机型。你将会在这里找到用于创建新设备定义的技术规格。创建新设备定义之后，就可以使用相同的步骤创建新的 AVD 了。

市场上有一款优秀的第三方 Android 模拟器，称为 Genymotion。Genymotion 模拟器对于非商业用途是免费的，而且性能优良。讲解 Genymotion 的安装和使用方法超出了本书的讨论范畴，不过可从 genymotion.com 下载 Genymotion 模拟器。

1.6 在 AVD 上运行 HelloWorld

要在新创建的 AVD 上运行 HelloWorld App，请单击工具栏上绿色的 Run 按钮，如图 1-28 所示。



图 1-28 Run 按钮

确保选中了 **Launch emulator** 单选按钮，接着在下方的下拉列表框中选择 **Galaxy Nexus API 21**。单击 **OK** 按钮，如图 1-29 所示。耐心点，因为启动 AVD 可能需要花费几分钟时间。你现在应该会看到 **HelloWorld App** 运行在自己电脑的窗口中，如图 1-30 所示。

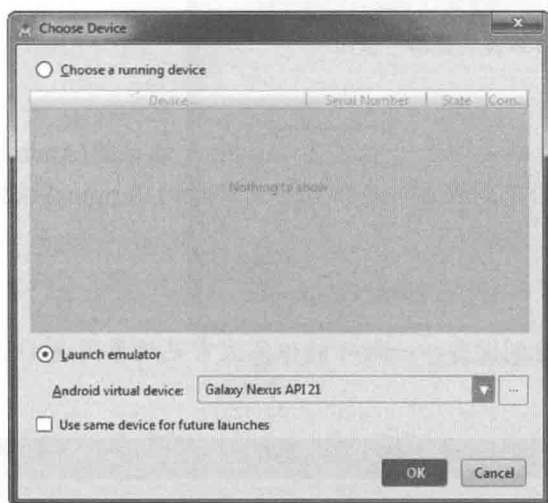


图 1-29 选择设备并启动模拟器

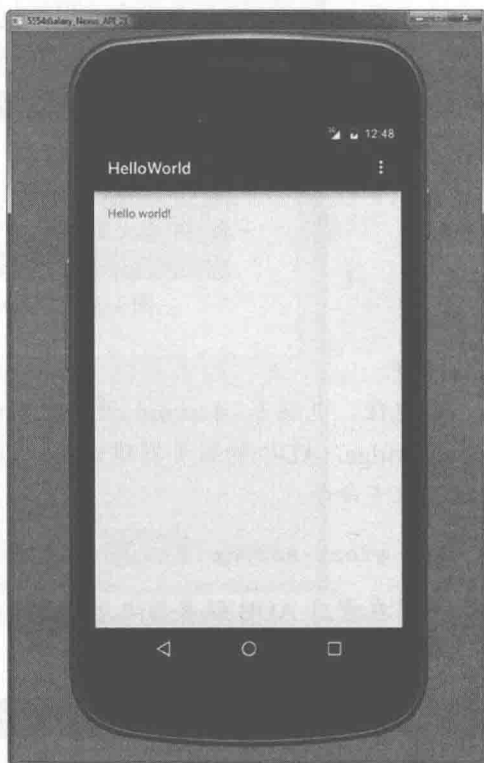


图 1-30 模拟器屏幕截图

1.7 在 Android 设备上运行 HelloWorld

如前所述，尽管 AVD 对于模拟特定设备(尤其是那些你手头上没有的设备)来说很有用，但是在物理 Android 设备上开发 App 更为可取。当把 Android 设备通过 USB 电缆连接到电脑时，如果电脑没有识别出设备，可能就需要安装 USB 驱动程序。如果电脑最初识别了 Android 设备，那么应该避免安装不同的或更新版本的 USB 驱动程序，因为这可能会导致 USB 连接失败。

注意

Mac 和 Linux 用户通常不必为在 Android 设备和电脑之间建立 USB 连接而下载驱动程序。

可以利用 developer.android.com/tools/extras/oem-usb.html#Drivers 上的表格找到合适的 USB 驱动程序，或者使用自己喜欢的搜索引擎来查找手机的 USB 驱动程序。下载驱动程序并将其安装在你的电脑上。在你的 Android 设备上，点击 **Settings**，然后是 **Developer Options**。确保选中了 **USB Debugging** 复选框。某些设备，例如三星设备，需要安全码来启

动 USB 调试，因此你可能需要使用自己喜欢的搜索引擎来弄清如何在设备上启用 USB 调试。如果“在特定设备上启用 USB 调试”这个过程并不显而易见的话，那么 YouTube 也可以提供一些优秀的参考视频。

大多数 Android 设备会附带一根数据线，一端为 USB 公插头，另一端为小口 USB 公插头。使用这根线缆将 Android 设备连接到你的电脑。单击图 1-31 中圈出的 Android Device Monitor 按钮。如果正确安装了驱动程序，你应该会看到设备在此列出且处于已连接状态，如图 1-32 所示。



图 1-31 “Android 设备监视器”按钮

注意

要记住，电脑和 Android 设备之间的连接是通过一台称为 Android 调试桥(Android Debug Bridge, ADB)的服务器建立的。如果看不到设备，单击 IDE 左下角的 Terminal 按钮并执行以下命令：

```
adb start-server
```

如果在重启 ADB 服务器之后仍然没有看到设备，一种可能但不太常见的原因是 USB 驱动程序需要在重启系统之后才能生效。

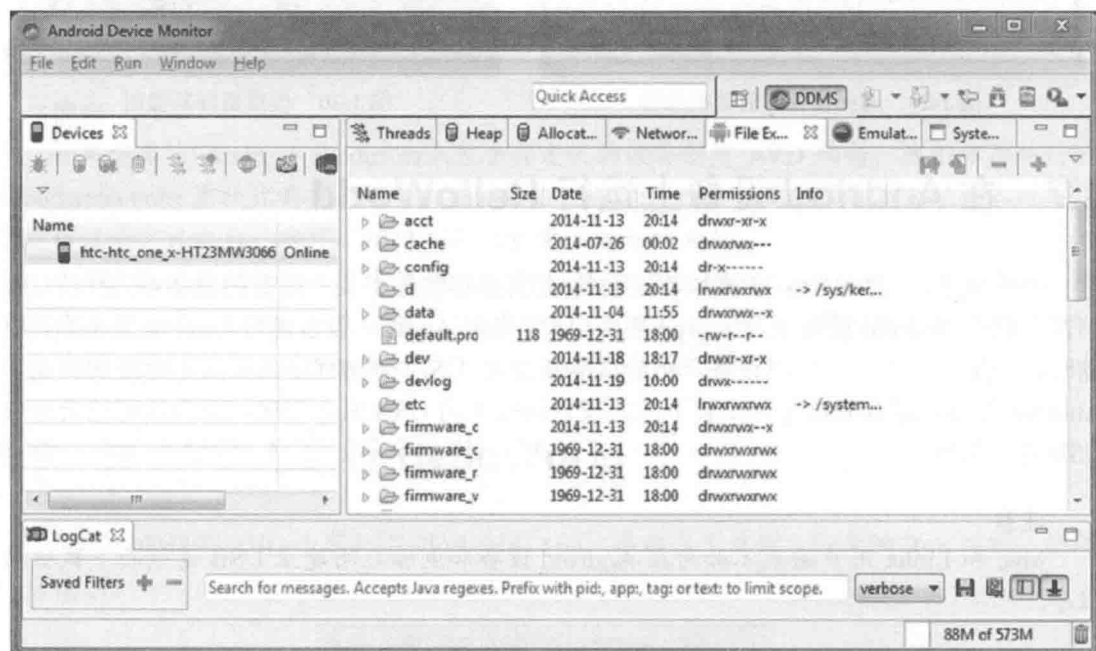


图 1-32 显示已连接物理设备的 Android 设备监视器界面

现在单击绿色的 Run 按钮，如前面的图 1-28 所示。选择已连接的 Android 设备。在图 1-33 中，已连接的设备是一款 HTC One X Android 智能手机。单击 OK 按钮，等待几秒钟，

你将看到 HelloWorld 运行在自己的 Android 设备上。

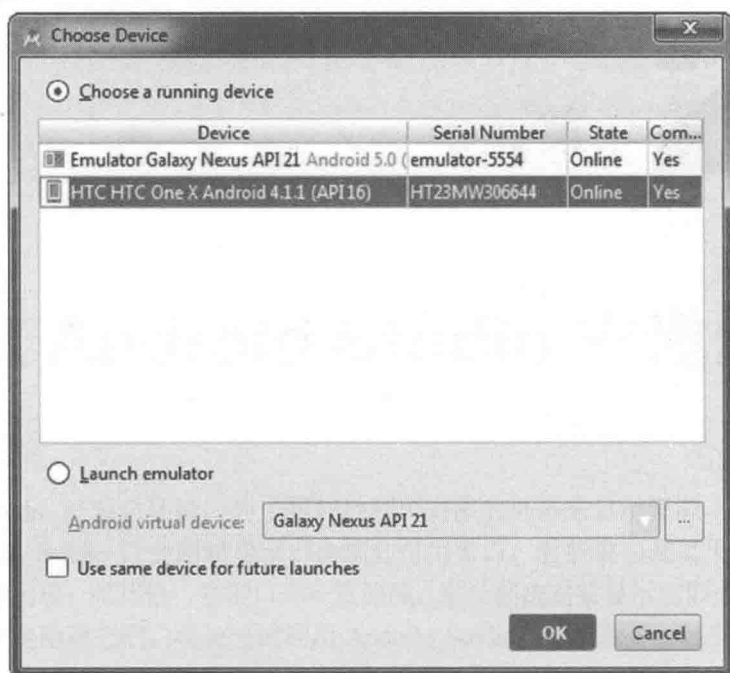


图 1-33 列出了物理 Android 设备的 Choose Device 界面

1.8 小结

在这一章中，你安装了 Java 开发工具包(Java Development Kit, JDK)，并安装了 Android Studio 和 Android SDK。你用 New Project Wizard 创建了一个名为 HelloWorld 的简单 App。接下来，你创建了 Android Virtual Device，即 AVD。我们演示了如何安装所需的 USB 驱动程序。最后向你展示了如何在 AVD 和物理 Android 设备上启动 HelloWorld。你现在应该具备了在 Android Studio 中开发 Android App 所需的全部软件。

第 2 章

在 Android Studio 中遨游

Android Studio 是窗口环境，为了最好地利用有限的屏幕来显示资源，并且保证你不会被干扰，Android Studio 只会同时显示一小部分可用窗口。有些窗口是上下文敏感的，仅在适当的上下文中出现；而其他一些窗口会一直隐藏，直至你决定要显示它们；或者反过来一直可见，直至你决定隐藏它们。要充分地利利用 Android Studio，你需要了解这些窗口的功能，以及打开它们的方法和时机。本章将向你展示如何管理 Android Studio 中的窗口。

所有集成开发环境(Integrated Development Environment, IDE)的核心功能之一就是导航。Android 项目通常由大量的包、目录和文件构成，而且即使是一个中等复杂度的 Android 项目也会包含数百个此类资源。使用 Android Studio 的生产效率将极大地取决于你在这些资源之间来回导航时的舒适程度。本章将向你展示如何在 Android Studio 中导航。

最后，我们将向你展示如何使用 Android Studio 的帮助系统。为充分理解这一章的内容，请打开我们在第 1 章中创建的 HelloWorld 项目。如果这个项目已经在 Android Studio 中打开了，那就可以开始了。在我们讨论以下导航特性的过程中，请参考图 2-1。

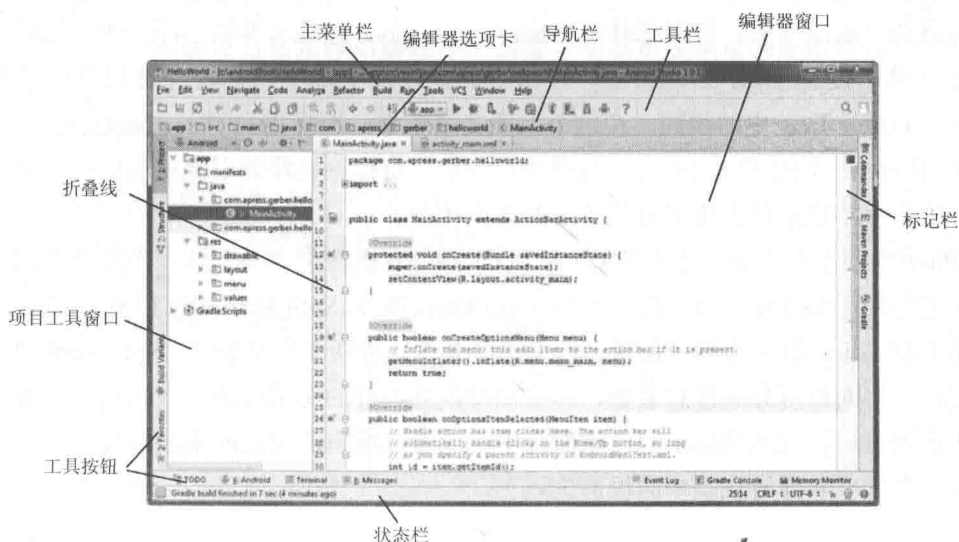


图 2-1 Android Studio 的集成开发环境

2.1 编辑器

任何 IDE 的主要目的都是编辑文件。正如我们期待的那样，Android Studio 中允许用户编辑文件的窗口位于 IDE 面板的中心。Editor 窗口是唯一一个时时可见并且总是位于中心面板中的窗口。事实上，Editor 窗口是 Android Studio 中极常用的特性，从现在起，我们将其简称为 Editor。Android Studio 中的所有其他窗口均被称为工具窗口，它们位于 Editor 周围(左侧、下方和右侧)的面板中。

Editor 是一个选项卡窗口，从这个角度讲，它构成了一款现代 Web 浏览器。当你通过某个工具窗口、键盘快捷键或上下文菜单打开一个文件时，该文件会显示为 Editor 的一个选项卡。正如你之前见到的那样，当构建第一个项目——HelloWorld 的时候，MainActivity.java 和 activity_main.xml 文件会自动加载为 Editor 的选项卡。Android Studio 会尝试预测你将要开始编辑的文件，并在 New Project Wizard 完成的时候在 Editor 中以选项卡形式自动打开它们。几乎所有文件都可以在 Editor 中打开，尽管原始的图像和声音文件(尚且)不能在 Android Studio 中进行编辑。也可以从工具窗口中将文件拖放到 Editor 中，这样会在 Editor 中以选项卡形式打开该文件。

Editor 的上方是 Editor 选项卡。Editor 的左侧边栏是折叠线，右侧边栏是标记栏。下面逐一研究每个部分。

2.1.1 Editor 选项卡

要在 Android Studio 的 Editor 选项卡之间进行导航，可使用 Alt+向右箭头 | Ctrl+向右箭头或 Alt+向左箭头 | Ctrl+向左箭头。当然，可以总是用鼠标来选择 Editor 选项卡。Editor 选项卡的选项位于主菜单栏的 Window | Editor Tabs 中。从此菜单中选择的任意操作均会应用于当前选中的选项卡。将鼠标移至 MainActivity.java 选项卡并右击(在 Mac 电脑上按住 Ctrl 键并单击)它。如图 2-2 所示，在出现的上下文菜单中，你将会看到有许多选项与 Window | Editor Tabs 中的相同。在这个上下文菜单中，选择 Tabs Placement 子菜单。菜单项 Top、Bottom、Left 和 Right 允许你移动标签栏。将标签栏移到右侧或左侧可以容纳更多可见选项卡，不过是以占用屏幕显示资源作为代价。

Editor 选项卡上下文菜单中的 Close 和 Close All 操作很容易理解。当想要关闭除活动选项卡之外的所有选项卡时，可以使用 Close Others 操作。Split Vertically 和 Split Horizontally 操作用于将 Editor 划分为多个面板。如果想要并排比较两个文件的话，Split Vertically 就极其有用。可将面板拆分成任意数量，尽管此种嵌套拆分的实用价值会很快消失。也可以把文件从其他窗口拖放到 Editor 的任意面板中，或者在编辑面板之间来回拖曳。关闭面板中的最后一个选项卡会导致整个面板消失。

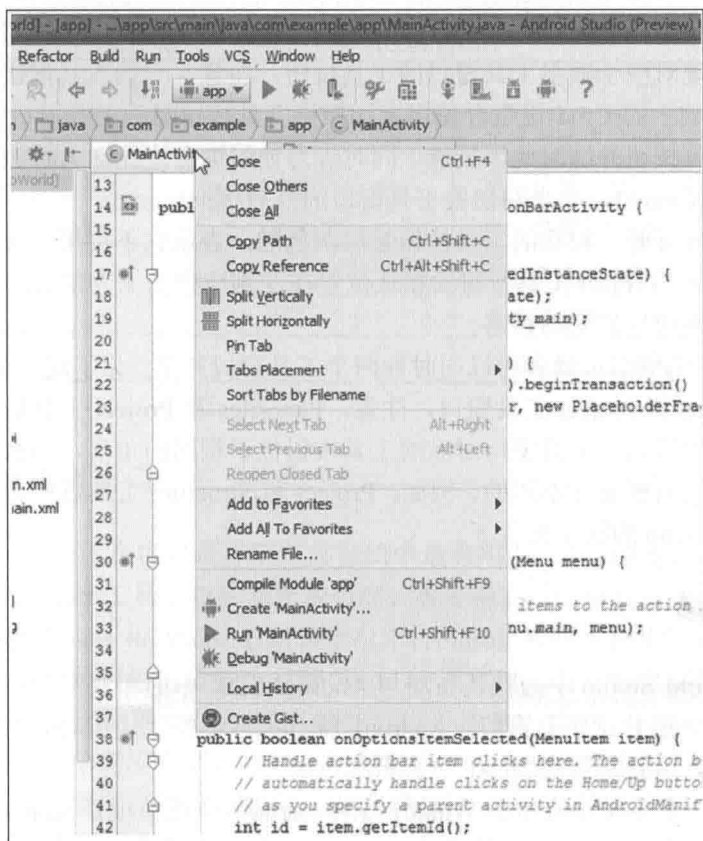


图 2-2 Editor 选项卡的上下文菜单

2.1.2 折叠线

折叠线(gutter)用于表达关于代码的信息。折叠线最显著的特性或许是那些带有颜色的小标签和图标，它们沿着对应的代码行显示，用于指示这些可视化资源。折叠线也用于设置断点、完成代码折叠以及显示作用域标识。后续章节会更详细地涵盖所有这些特性。

2.1.3 标记栏

Editor 的右侧是标记栏。标记栏用于标识源文件中重要行的位置。例如，标记栏会突出显示 Java 或 XML 文件中的警告和编译时错误。标记栏还会向你展示未提交的更改、查找结果和书签的位置。

标记栏不会像折叠线那样滚动；相反，标记栏上的颜色标记会出现在文件长度的相对位置。单击标记栏中的颜色标记会立即跳转到文件中的该位置。现在就单击标记栏中的一些颜色标记，练习使用它。

2.1.4 工具按钮

你已经看到了 Project 工具窗口，它默认显示在左侧面板中。要查看所有工具窗口的列

表, 可以选择主菜单栏中的 View | Tool Windows。现在仔细观察 IDE 的左侧、右侧和下方边栏。你将会发现对应于很多工具窗口的工具按钮。注意, 一些工具按钮还标有数字, 可以与 Alt(Mac 电脑上的 Cmd)键组合使用来切换该工具按钮对应工具窗口的开启/关闭。现在就尝试单击工具按钮并练习此项技能。同时练习使用键盘快捷键 Alt+1 | Cmd+1、Alt+2 | Cmd+2 和 Alt+3 | Cmd+3, 并来回切换工具窗口的开启/关闭。

当工具窗口打开时, 相应的工具按钮是深灰色的, 表示它不可用。注意工具按钮位于边栏的角落。例如, Project 工具按钮的默认位置在左侧边栏的上方角落, 而 Favorites 工具按钮默认位于左侧边栏的底部角落。

(左、下以及右)侧面板最多可以同时被两个工具窗口共享。为了观察如何共享侧面板, 同时打开 Favorites 和 Project 工具窗口。注意, Favorites 和 Project 工具按钮位于同一边栏中相对的两角。如果两个工具窗口对应的工具按钮位于相同的角落, 那么尝试在这两个工具窗口之间共享侧面板是行不通的。例如, Project 和 Structure 工具窗口无法同时显示——至少在 Android Studio 的默认配置下。

2.1.5 默认布局

不要把 Android Studio 中的默认布局与 Android SDK 中的布局混为一谈。默认布局是围绕着 Editor 的一套特定工具窗口。Android Studio 自带的默认布局会在左侧面板中展示 Project 工具窗口。这就是前面图 2-1 中所示的布局。

让我们研究一下主菜单栏中的 Window 菜单。前面两个菜单项是 Store Current Layout as Default 和 Restore Default Layout。Restore Default Layout 操作通常在 IDE 变得过于拥挤时使用, 或者你只是想要清理面板并返回到更熟悉的布局。可以定制自己的默认布局——打开和关闭任意工具窗口、重新设置大小/或者重新摆放它们, 然后通过选择 Store Current Layout as Default 将此新布局设置为默认布局。

重新摆放工具按钮

如前所述, Project 和 Structure 工具窗口无法同时显示, 因为它们对应的工具按钮位于同一角落。然而, 可以将任意工具按钮移动到你想要的任意角落。将 Structure 工具按钮拖放至左侧边栏的底角。现在, 通过使用快捷键 Alt+1 | Cmd+1 和 Alt+7 | Cmd+7, 或者通过单击它们的工具按钮, 开启 Project 和 Structure 工具窗口。由于我们将它们的工具按钮移到了相对的角落, 现在 Project 和 Structure 工具窗口就可以共享相同的侧面板并同时显示了。

2.2 导航工具窗口

本节讨论专门用于导航的工具窗口: Project、Structure、Favorites、TODO 和 Commander。表 2-1 列出了每个导航工具窗口的功能。后续章节会涵盖多数其他工具窗口。

表 2-1 导航工具窗口

工具窗口	PC 键	Mac 键	功 能
项目	Alt+1	Cmd+1	允许浏览项目中的文件和资源
收藏	Alt+2	Cmd+2	显示收藏、书签和断点
结构	Alt+7	Cmd+7	显示当前文件中对象或元素的树型结构
命令			类似于 Project 工具窗口，但可以更容易地管理文件
TODO			显示项目中所有有效 TODO 的列表

2.2.1 Project 工具窗口

我们发现 Project 工具窗口是导航工具窗口中最有用的一个，因为它用途广泛且相对容易使用。要领会 Project 工具窗口的功能和范畴，需要将窗口的模式设置为 Project。模式共有三种：Project、Packages 和 Android。默认情况下，Android Studio 会将模式设置为 Android。Android 和 Project 是最有用的模式，尽管 Android 模式可能会对你隐藏某些目录。模式设置复选框靠近 IDE 左上角的 Project 工具按钮且位于其 90° 方向。Project 工具窗口提供了文件和嵌套目录的简单树型界面，可以切换其显示。Project 工具窗口为你展示了项目中所有包、目录和文件的概况。如果在 Project 工具窗口中右击(在 Mac 电脑上按住 Ctrl 键并单击)某个文件，则会出现一个上下文菜单。在这个上下文菜单中有三个重要的菜单项：Copy Path、File Path 和 Show in Explorer。单击 Copy Path 会将此文件在操作系统中的绝对路径复制到剪贴板。单击 File Path 会以目录栈的形式显示路径，并以栈顶的文件结束，而单击任意这些目录都会在操作系统中打开它们。单击 Show in Explorer 会在一个新的操作系统窗口中显示该文件。参见图 2-3。

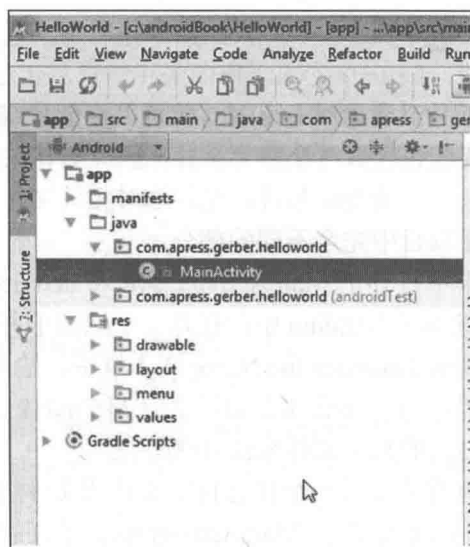


图 2-3 Project 工具窗口

2.2.2 Structure 工具窗口

Structure 工具窗口显示文件中元素的层次结构。当 Editor 显示 Java 源文件(例如 MainActivity.java)时, Structure 工具窗口显示包含字段、方法和内部类等元素的树。而当 Editor 显示 XML 文件(例如 activity_main.xml)时, Structure 工具窗口显示 XML 元素的树。单击 Structure 工具窗口中的任意元素会立刻将光标移至该元素在 Editor 中的位置。Structure 工具窗口对于在较大源文件的元素中导航尤其有用。练习此项技能——打开 Structure 工具窗口并在 MainActivity.java 和 activity_main.xml 的元素之间导航。参见图 2-4。

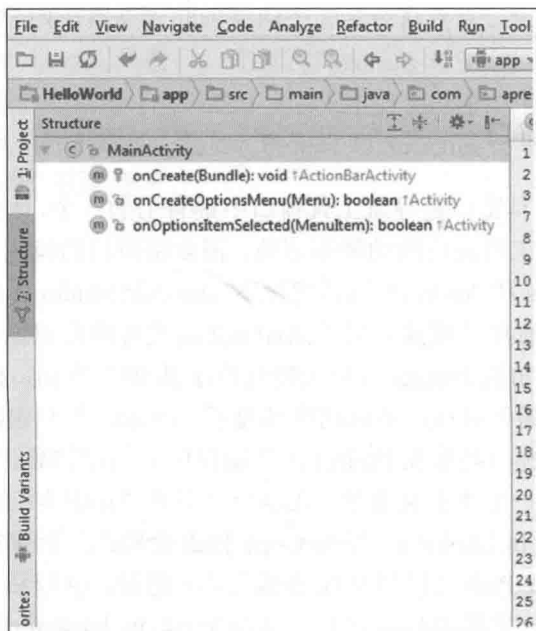


图 2-4 Structure 工具窗口

2.2.3 Favorites 工具窗口

当开发 Android 中的特性(或者调试 bug)时,你可能会创建或修改一些相关的文件。中等复杂度的 Android 项目可能包含数百个独立文件,因此对相关文件分组的功能的确很有用。Favorites 工具窗口中包含一些收藏条目,允许你对相关文件的引用进行逻辑分组,而这些文件可能在物理上位于项目中完全不同的部分。

确保在 Editor 的选项卡中打开了 MainActivity.java 和 activity_main.xml 文件。现在右击(在 Mac 电脑上按住 Ctrl 键并单击)Editor 中的任意一个选项卡并在上下文菜单中选择 Add All to Favorites。在 Input New Favorites list Name 文本框中,输入 main 并单击 OK 按钮。如果 Favorites 工具窗口没有打开,那么现在通过 Alt+2 | Cmd+2 来启动它。展开名为 main 的收藏条目,并双击其中列出的某个文件来打开/激活它。

就像 Favorites 窗口允许你立即导航至任意特定文件或文件组那样,书签允许你立即导航至文件中的任意特定行。将光标置于 MainActivity.java 的任意行中。现在按 F11 键(在 Mac 电脑上按 F3 键)。此操作会创建或删除任何源文件(包括 XML 文件)中的书签。注意,

折叠线中的对号和标记栏中的黑色标记均表示新的书签。要查看刚刚创建的书签，可以在 Favorites 工具窗口中打开书签。

注意

在 PC 机上，如果按 F11 键没有响应，那么检查并确保键盘上的 F-Lock 键已经激活。

断点供调试时使用。与可以在任意文件中设置的书签不同，需要在 Java 源文件中才能设置断点。打开 MainActivity.java 并单击靠近以下代码行处的折叠线：

```
setContentView(R.layout.activity_main);
```

你将会发现折叠线上出现了一个红圈，而且该行也会以红色突出显示。断点只能在可执行代码行处设置；在(例如)注释行上尝试设置断点是行不通的。要查看新创建的断点，可在 Favorites 工具窗口中打开 Breakpoints 树。还可以使用断点完成更多有趣的事情，我们将在专注于调试的第 12 章中详细讨论断点。

2.2.4 TODO 工具窗口

显然，TODO 表示代办事项。TODO 本质上就是注释，用于提醒程序员和他们的合作者还有尚未完成的工作。TODO 的写法类似于注释，以两个前向斜杠开始，单词 TODO 全部大写，然后是一个空格。例如：

```
//TODO inflate the layout here.
```

在 MainActivity.java 中创建一个 TODO，并打开 TODO 工具窗口来观察它。在 TODO 工具窗口中单击一个 TODO 会立刻跳转到该 TODO 在源代码中的位置。

2.2.5 Commander 工具窗口

Commander 工具窗口是一个导航辅助工具，有左右两个面板。这两个面板的功能非常类似于 Project 和 Structure 工具窗口所做的事情。Commander 工具窗口与显示嵌套目录树的其他窗口不同，它同时仅显示一个目录级别。如果更喜欢 Windows 风格的导航或者觉得 Project 工具窗口过于复杂，那么 Commander 工具窗口可能是很好的替代。

2.3 主菜单栏

主菜单栏位于 Android Studio 的最上方，通过使用它的菜单和子菜单，几乎可以执行任何操作。与 Android Studio 中其他的栏目不同，主菜单栏无法隐藏。不要被主菜单栏及其子菜单中的诸多操作吓倒。即使是最老练的 Android 开发者在日常工作中也仅会用到这些操作中的一小部分，而且这些操作中的大部分均有对应的键盘快捷键和/或上下文菜单项。我们会在后续章节中讨论主菜单栏中包含的多种操作。

2.4 工具栏

工具栏中包含频繁使用的文本操作按钮，例如 Cut、Copy、Paste、Undo 和 Redo。如你在第 1 章中所见，工具栏同时包含在 Android Studio 中进行各种管理的按钮，包括 SDK Manager 和 Android Virtual Device Manager。工具栏还包含用于 Settings 和 Help 的按钮，以及用于 Run 和 Debug 应用的按钮。工具栏中的所有按钮均有对应的菜单项和键盘快捷键。为节省屏幕空间，高级用户可以通过取消对 View | Toolbar 菜单项的选中来隐藏工具栏。

2.5 导航栏

导航栏显示一个横向的链式箭头框，表示从项目根目录(位于左侧)到 Editor 当前所选选项卡中文件(位于右侧)的路径。导航栏用于在不借助 Project 或 Commander 工具窗口的情况下，在项目的资源之间进行导航。

2.6 状态栏

如图 2-5(以及之前的图 2-1)所示，状态栏显示相关且上下文敏感的反馈，例如关于所有正在运行的进程或项目 Git 仓库状态的信息。现在让我们来深入探索状态栏。

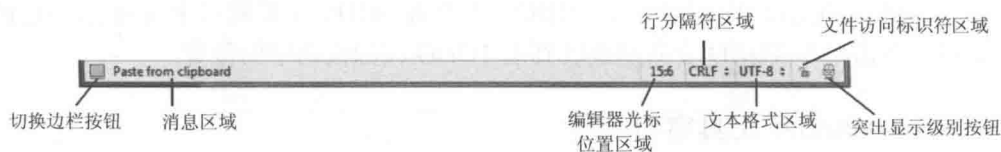


图 2-5 状态栏

状态栏的最左角是 Toggle Margins 按钮。单击这个按钮会切换边栏的隐藏和显示。此外，当把鼠标移至此按钮上时，会出现一个上下文菜单，允许你激活任何工具窗口。

消息区域用于提供反馈并显示关于当前正在运行进程的所有信息。当把鼠标移至菜单项或工具栏中的按钮等 UI 元素时，此区域还会显示提示。在这个区域上单击会打开 Event 日志。

Editor 光标位置以行:列的格式显示光标在 Editor 中的位置。单击这个区域会激活一个对话框，允许你直接导航至代码中的某个特定行。

行分隔符区域显示了文本文件中使用的回车换行格式。在 Windows 上，默认值是 CRLF，表示回车换行符。LF 是 Unix 和 Mac 机器上使用的标准格式，也是 Git 中所采用的。如果正在 Windows 电脑上进行开发，Git 通常会在向仓库提交代码时将 CRLF 转换为 LF。

文本格式区域描述了用于源文件的文本编码。默认值是 UTF-8，它是 ASCII 的超集且涵盖了大多数西方字符，包括你能够在标准 Java 或 XML 文件中找到的所有字符。

文件访问标识符区域允许你在读/写和只读之间进行切换。打开的锁的图标表示拥有 Editor 中当前文件的读/写权限。锁图标意味着编辑器中的当前文件是只读的。可以通过单击标识符的图标来切换这些设置。

Highlighting Level 按钮会激活一个带有滑块的对话框，允许你设置想要在代码中看到的突出显示级别。

默认设置是 Inspections，它对应一个皱着眉头的 Inspections Manager 图标。此设置意味着你要准备好接受一些严格的检查，因为 Inspections Manager 会严格识别代码中的语法错误和可能的问题(称为警告)。你会看到 Inspections Manager 在标记栏中以黄色标记的形式生成的一些警告。

滑块上的下一个设置是“语法”，它对应于 Inspections Manager 中的一个图标。在这种设置下，Inspections Manager 会忽略警告。Syntax 模式没有 Inspections 模式严格，但仍然会突出显示那些将会阻止代码编译的语法问题。

滑块上的最后一个突出显示模式是 None，它对应一个微笑着的 Inspections Manager 图标。这个图标会让人联想到 Inspections Manager 喝醉了，已经不再关心你的代码了。在此模式下，即使是最严重的语法错误也会被忽略，尽管在尝试构建的时候，编译器还是会被这些错误卡住。推荐将突出显示级别保留为 Inspections，并学着接受 Inspections Manager 的严格检查。

2.7 常用操作

本节讲述 Android Studio 中的各种常见操作。如果用过类似 Microsoft Word 的文本编辑器，那么你可能会很熟悉本节涵盖的特性。

2.7.1 选择文本

与所有优秀文本编辑器一样，双击源文件中的单词就可以选中它。此外，沿着字母或单词单击并拖曳光标会选中这些文本元素。将光标置于源文件中的任意位置并按 Shift+向下箭头或 Shift+向上箭头，则会从光标开始处选中文本行。三击文本行中的任意位置可以选中整行。按 Ctrl+A | Cmd+A 可以选中文件中的全部文本。

如果将光标置于任意单词中并按 Ctrl+W | Alt+向上箭头，那么整个词会被选中。如果继续按 Ctrl+W | Alt+向上箭头，选中区域则会扩展至包含任意数量的相邻文本。如果现在按 Ctrl+Shift+W | Alt+向下箭头，则选中区域会缩小。这个扩展/缩小选中区域的功能在 Android Studio 中称为结构化选择。

2.7.2 使用 Undo 和 Redo

Undo 和 Redo 命令用于向后和向前回滚有限数量的编辑操作。修改由特定 UI 事件来分隔，例如按下 Enter 键或者重新设置光标的位置。Undo 和 Redo 对应的键盘快捷键分别是 Ctrl+Z | Cmd+Z 和 Ctrl+Shift+Z | Cmd+Shift+Z。工具栏左侧有紫色的向右和向左箭头，

可以完成相同的功能。Android Studio 默认会记录自上一次保存起的所有步骤，最多记录 300 步。Undo 和 Redo 每次只能作用于一个文件，因此回退修改的最有效方法是使用 Git，我们会在第 7 章中讨论它。

2.7.3 找到最近的文件

Android Studio 最优秀的特性还包括它会记录你在近期操作过的所有文件。要触发此命令，选择 View | Recent Files 或者按 Ctrl+E | Cmd+E。出现的对话框允许你选择任意近期文件并在 Editor 中以选项卡的形式打开。默认记录数的上限是 50 个之前的文件。可以通过导航至 File | Settings | Limits | Editor | Recent Files Limit 修改这些限制。

2.7.4 遍历最近的导航操作

Android Studio 还会记住你最近的导航操作。导航操作包含光标移动、选项卡切换和文件打开。要遍历你的导航操作历史，请按 Ctrl+Alt+向左箭头 | Cmd+Alt+向左箭头或 Ctrl+Alt+向右箭头 | Cmd+Alt+向右箭头。要记住导航操作不同于编辑操作；如果想要遍历编辑操作，应该使用 Undo 和 Redo。

2.7.5 剪切、复制和粘贴

如果使用过任意一款文本编辑器或字处理程序，那么你一定很熟悉 Cut、Copy 和 Paste 命令。表 2-2 列出了这些基本命令，以及一些扩展的剪贴板命令。

表 2-2 剪切、复制和粘贴

命 令	PC 键	Mac 键
Cut	Ctrl+X	Cmd+X
Copy	Ctrl+C	Cmd+C
Paste	Ctrl+V	Cmd+V
Extended Paste	Ctrl+Shift+V	Cmd+Shift+V
Copy Path	Ctrl+Shift+C	Cmd+Shift+C
Copy Reference	Ctrl+Alt+Shift+C	Cmd+Alt+Shift+C

除了操作系统剪贴板提供的简单 Cut、Copy 和 Paste 功能以外，Android Studio 还拥有会记录最后 5 次 Cut 和 Copy 操作的扩展剪贴板。当从 Android Studio 剪切或复制文本，或者在 Android Studio 运行过程中从几乎任何其他应用剪切或复制文本时——Android Studio 都会将该文本置于栈中。要查看扩展剪贴板的栈，请按 Ctrl+Shift+V | Cmd+Shift+V。出现的对话框允许选择想要粘贴的任意条目。参见图 2-6。



图 2-6 扩展剪贴板

也可通过导航至 File | Settings | Limits | Editor | Maximum Number of Contents to Keep in Clipboard 来修改扩展剪贴板的栈大小。还可以通过右击选定文本并选择 Compare with Clipboard，将任意当前选中文本与扩展剪贴板中的最近元素进行比较。

Copy Path 命令 Ctrl+Shift+C|Cmd+Shift+C 可以复制 Project 或 Commander 工具窗口，以及任意 Editor 选项卡中选中文件或目录的操作系统全路径。Copy Path 对于终端会话中的操作尤其有用。

使用 Copy Reference Ctrl+Alt+Shift+C | Cmd+Alt+Shift+C，Android Studio 能够复制方法、变量或类的逻辑引用。当把此引用粘贴到另外一个源文件中时，Android Studio 会自动包含所有必需的包限定符和导入语句。除了诸如拖放等鼠标操作之外，还可以在 Project 和 Commander 工具窗口中对包、目录和文件使用通用的 Cut、Copy 和 Paste 操作，以便在项目中重新安排资源位置。

2.8 上下文菜单

通过在 IDE 中右击(在 Mac 电脑上按住 Ctrl 键并单击)可以调用大量的上下文菜单。在上一节中，你已经探索了 Editor 选项卡的上下文菜单。如果右击(在 Mac 电脑上按住 Ctrl 键并单击)的话，Android Studio 中的大多数面板、图标和工具条都会产生上下文菜单。Android Studio 最伟大的特性之一就是可以采用不止一种方法来执行操作。这种冗余意味着可以根据自己的偏好自由地培养技能和习惯。本人发现与 Android Studio 交互的最有效方

法是对最频繁的操作使用键盘快捷键，对不太频繁的操作使用菜单和上下文菜单操作。现在通过右击(在 Mac 电脑上按住 Ctrl 键并单击)IDE 中的工具条、选项卡、面板和文件来探索上下文菜单。

2.9 获取帮助

Android Studio 的 Help 菜单中包含几个有用的菜单项。Find Action (Ctrl+Shift+A | Cmd+Shift+A)是在 Android Studio 中获取帮助的最常用命令。此命令会打开一个对话框，允许你搜索 Android Studio 中的所有特性。按 Ctrl+Shift+A | Cmd+Shift+A 并在搜索框中输入 Show Line Numbers。现在使用方向键选择 Settings 并按 Enter 键。在 Settings 窗口中，选择 Editor | Appearance。你应该会看到 Show Line Numbers 复选框。

Help | Online Documentation 是所有 Android Studio 技术规范文档的来源，这是关于 Android Studio 最详细的文档。此外，Help | Default Keymap Reference 菜单项是一个很有用的参考。可考虑将此 PDF 打印出来并在学习使用 Android Studio 的过程中随身携带。

2.10 使用键盘导航

键盘或许是在 Android Studio 中导航的最有效方式了。选择主菜单栏中的 Navigate 菜单，观察其内容。本节讨论了最重要的菜单项(如表 2-3 所示)以及它们在 Navigate 菜单中的相应键盘快捷键。后续章节会讨论其他菜单项。

表 2-3 键盘导航

命 令	PC 键	Mac 键
Select In	Alt+F1	Alt+F1
Class	Ctrl+N	Cmd+O
File	Ctrl+Shift+N	Cmd+Shift+O
Line	Ctrl+G	Cmd+L
Related File	Ctrl+Alt+Home	Alt+Cmd+向上箭头
Last Edit Location	Ctrl+Shift+Backspace	Cmd+Shift+Backspace
Type Hierarchy	Ctrl+H	Ctrl+H
Declaration	Ctrl+B	Cmd+B

2.10.1 Select In 命令

Android Studio 最棒的特性之一即导航是双向的。你已经看到了利用各种工具以 Editor 选项卡的形式打开/激活文件的方法。现在你将要学习如何在 Editor 的各种工具窗口中导航。

按 **Alt+F1** 快捷键，这会激活 **Select In** 上下文菜单，它包含一些菜单项，包括 **Project View**、**Favorites** 和 **File Structure**。单击 **Project View** 选项。**Project** 工具窗口变为活动状态，对应于 **Editor** 活动选项卡的文件会突出显示，该文件的所有父目录也均处于打开状态。**Android** 项目通常会含有大量文件资源；因此，使用 **Select In** 是你需要掌握的最重要技能之一。

2.10.2 Class 命令

Class 操作允许你导航至特定的 **Java** 类。很重要的一点，是要注意此操作仅在 **Java** 源文件或 **Java** 源文件的内部类中进行查找。按 **Ctrl+N** | **Cmd+O** 并输入 **act**。**Android Studio** 已经对所有文件建立了索引，因此它会为你提供一个可能匹配的列表，并突出显示最有可能的匹配项。你需要做的所有事情就是按 **Enter** 键来打开 **MainActivity.java**。

2.10.3 File 命令

File 操作允许你导航至项目中的任意文件。如果正在查找项目中的 **XML** 文件，那么你需要采用这种操作方法。按 **Ctrl+Shift+N** | **Cmd+Shift+O** 并开始输入 **act**。我们使用相同的查找关键字 **act**，目的是要演示 **Navigate | File** 操作更广泛的作用范围。注意，查找结果包含了 **Java** 源文件 **MainActivity.java** 以及所有其他类型文件，例如 **activity_main.xml**。使用方向键选中 **activity_main.xml** 并按 **Enter** 键打开它。

2.10.4 Line 命令

Line 操作 **Ctrl+G** | **Cmd+L** 会激活一个对话框，允许你导航至源文件中某个特定的行和列。如果只在出现的 **Go to Line** 对话框中输入一个数字并单击 **OK** 按钮，那么 **Android Studio** 将会跳转至该行，而不会考虑列。

2.10.5 Related File 命令

Related File 操作 **Ctrl+Alt+Home** | **Alt+Cmd+向上箭头** 是 **Android Studio** 中最有用的命令之一。**Android** 项目通常有大量相关联的文件。例如，普通的 **Android Activity** 通常含有至少一个对应的 **XML** 布局文件，用于渲染 **Activity** 的布局；以及一个对应的 **XML** 菜单文件，用于渲染 **Activity** 的菜单。当使用 **Fragment** 时，这种复杂性还会增加。你已经看到了如何使用 **Favorites** 将相关的文件组合在一起。可以使用 **Navigate | Related File** 来请求 **Android Studio** 展示相关联的文件。激活 **MainActivity.java** 选项卡后，按 **Ctrl+Alt+Home** | **Alt+Cmd+向上箭头**。你应该会看到 **activity_main.xml** 列在了这里。使用方向键选中它并按 **Enter** 键。

2.10.6 Last Edit Location 命令

Last Edit Location 操作 **Ctrl+Shift+Backspace** | **Cmd+Shift+Backspace** 允许导航至最后一次编辑之处。如果继续使用此命令，光标将会移到上一次编辑的文件/位置，以此类推。

2.10.7 Type Hierarchy 命令

Android 使用 Java，这是一种面向对象的编程语言。面向对象语言的特征之一就是继承，它简化了代码重用和多态。在 Editor 中激活 MainActivity.java 文件，按 Ctrl+H 打开 Hierarchy 工具窗口。你将在这里看到一系列级联对象，所有这些对象均有自己的父对象，一直追溯至 Object，它是 Java 中所有对象的父对象。要记住，Navigate | Type Hierarchy 操作仅在 Editor 的活动选项卡中是 Java 源文件时起作用。

2.10.8 Declaration 命令

Declaration 操作允许你跳转至方法、变量和资源的原始声明处。激活此操作的另一种方法是按住 Ctrl|Cmd 键的同时在文件中的方法、变量或资源上滑动鼠标。如果元素变为带下划线的状态，那么在继续按住 Ctrl|Cmd 键的同时，单击该元素就可以导航至其声明处。在 MainActivity.java 的 setContentView(...)方法中的任意位置单击光标，并按 Ctrl+B |Cmd+B。你将会立刻被带到此方法的声明处，位于 MainActivity 的超类(名为 ActionBarActivity.java)中。

2.11 查找和替换文本

查找和替换文本是编程中很重要的一部分，Android Studio 含有强大的工具集来帮助你完成这些工作。本节涵盖了一些最重要的工具。表 2-4 列出了这些内容。

表 2-4 查找和替换

命 令	PC 键	Mac 键
Find	Ctrl+F	Cmd+F
Find in Path	Ctrl+Shift+F	Cmd+Shift+F
Replace	Ctrl+R	Cmd+R
Replace in Path	Ctrl+Shift+R	Cmd+Shift+R

2.11.1 Find 命令

Find 操作用于在单独文件中找出文本的出现位置。在 MainActivity.java 中，按 Ctrl+F |Cmd+F，打开一个出现在 Editor 上方的搜索栏。在搜索栏中输入 action。你将注意到文件中的 action 会立即以黄色高亮显示。你还将注意到标记栏中绿色的小标记，它们标识了已找到文本的位置。把鼠标滑动到查找栏的向右双箭头之上将显示高级查找选项。

2.11.2 Find in Path 命令

相对于之前描述的 Find 操作，Find in Path 允许在更广的范围内进行搜索。也可以使用

正则表达式，并使用文件掩码来限定结果范围。按 `Ctrl+Shift+F` | `Cmd+Shift+F` 并在 Editor 顶部的搜索栏中输入 `hello`。默认情况下，Find in Path 的搜索范围是 Whole Project，尽管可以将搜索范围限定为某个特定目录或模块。接受默认值 Whole Project 并单击 Find 按钮。结果会出现在 Find 工具窗口中。如果在 Find 工具窗口中单击条目，会立即在 Editor 中以新选项卡形式打开包含该条目的文件并跳转至出现处。

2.11.3 Replace 命令

Replace 操作 `Ctrl+R` | `Cmd+R` 用于在单个文件中替换某个文本实例，Replace 功能是 Find 的超集。替换文本的更安全方式是使用 Refactor | Rename 命令，我们将会在后面涉及。

2.11.4 Replace in Path 命令

Replace in Path 操作 `Ctrl+Shift+R` | `Cmd+Shift+R` 是 Find in Path 的超集。然而，使用 Refactor | Rename 总比使用 Replace in Path 好一些，因此使用此命令的时候要尤其小心，因为很可能会引入错误。

2.12 小结

本章讨论了 Editor 及其周围的工具窗口。我们已经讨论了如何使用工具按钮以及重新摆放它们。我们还讨论了那些用于导航的工具窗口和 IDE 中的主要 UI 元素，包括主菜单栏、工具栏、状态栏、折叠线和标记栏。我们也讨论了如何使用菜单和键盘快捷键查找和导航，以及使用查找和替换功能。最后，我们讨论了如何使用 Android Studio 中的帮助系统。最重要的是，我们建立起了一套关于 Android Studio 中 UI 元素的词汇体系，后续章节中会引用它们。



第 3 章

在 Android Studio 中编程

本章涵盖了如何在 Android Studio 中编写和生成代码。Android Studio 利用其面向对象编程优势生成高度相关且结构良好的代码。本章涵盖的特性包括重载方法、使用 Java 块包裹语句、使用模板插入代码、使用自动代码补全、注释代码和移动代码。如果阅读本书的目的是掌握 Android Studio，那么你要重点关注本章，因为这里介绍的工具和技巧能极大地帮助你提高编程效率。

让我们开始吧。如果还没有打开你在第 1 章中创建的 HelloWorld App，那么现在就打开它。

3.1 使用代码折叠

代码折叠是在 Editor 中节省屏幕显示资源的一种方法。代码折叠允许你隐藏特定代码块，以便能够更加专注于自己感兴趣的那些代码块。如果 MainActivity.java 没有打开，那么按 `Ctrl+N` | `Cmd+O` 并输入 Main。通过按 `Enter` 键来打开 MainActivity.java 类，如图 3-1 所示。

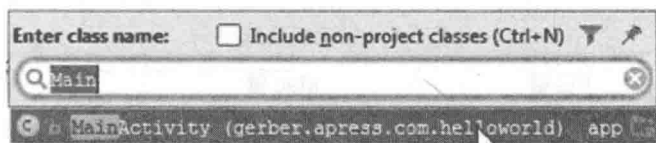


图 3-1 使用 Enter Class Name 对话框打开 MainActivity.java

如果没有默认显示行号，那么导航至 `Help | Find Action`。输入 `show line numbers` 并选择 `Show Line Numbers Active Editor` 选项，如图 3-2 所示。

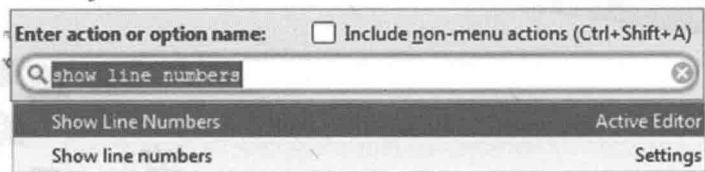


图 3-2 使用 Enter Action or Option Name 对话框显示行号

当观察 MainActivity.java 中的行号时，你将会发现一些奇怪的事情：行号不是连续的。在图 3-3 中，行号以 1、2、3 开头，然后跳到 7、8、9。注意看图 3-3 中的第 3 行。你会注意到 import 语句的左侧方框中有一个加号标志且后面有一个省略号。如果仔细地观察自己的代码，你还会发现省略号以浅绿色突出显示。所有这些视觉元素均在提示你 Android Studio 隐藏了一个代码块——它被折叠了。

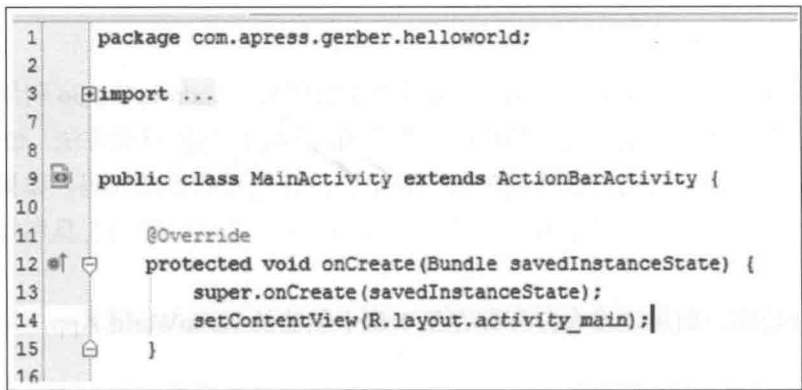


图 3-3 在 import 语句处折叠的代码块

一条称为折叠轮廓的细虚线位于左侧边栏中，位于灰色的折叠线和白色的 Editor 之间。折叠轮廓可能包含三种图标：方框中的加号图标(如图 3-3 中的第 3 行所示)以及内部含有一条水平线的向上和向下箭头(参见图 3-3 中的第 12 行和第 15 行)。向下箭头表示一个可折叠代码块的开始，而向上箭头表示可折叠代码块的结束。如前所述，加号框表示已折叠的代码块。单击这些图标中的任意一个会使相应代码块在已折叠或未折叠状态之间切换。表 3-1 中包含了关于所有代码折叠操作的描述和键盘快捷键。

表 3-1 代码折叠选项

选 项	PC 键	Mac 键	描 述
Expand	Ctrl+数字加号	Cmd+数字加号	展开光标所处位置的已折叠代码块
Collapse	Ctrl+数字减号	Cmd+数字减号	折叠光标处已展开的代码块
Expand All	Ctrl+Shift+数字加号	Cmd+Shift+数字加号	展开窗口中的全部代码
Collapse All	Ctrl+Shift+数字减号	Cmd+Shift+数字减号	折叠窗口中的全部代码
Toggle Fold	Ctrl+句号	Cmd+句号	折叠/展开光标所处位置的代码块

将光标置于 MainActivity.java 的 onCreate()方法中的任意位置。然后多次按 Ctrl+句号 | Cmd+句号, 切换此代码块的展开和收起状态。同样尝试使用 Expand 键盘快捷键 Ctrl+数字加号 | Cmd+数字加号和 Collapse 键盘快捷键 Ctrl+数字减号 | Cmd+数字减号。

最后, 使用鼠标单击折叠轮廓中的代码折叠图标, 切换代码的折叠和展开状态。记住折叠单个块、多个块甚至文件中的所有块只是为了从视野中移除它们, 以便节省屏幕显示资源。不过当构建的时候, 编译器依然会尝试编译它们。类似地, 折叠包含问题或错误的代码将不会删除标记栏中的任何警告或错误。可以通过选择菜单选项 Settings | Editor | Code Folding 来修改代码折叠选项。

3.2 执行代码补全

大多数现代 IDE 均提供某种形式的代码补全, 而 Android Studio 也不例外。Android Studio 时刻准备着提供帮助, 即使你没有主动地寻求帮助。在实践中, 这意味着 Android Studio 默认会在你输入的过程中为补全代码提供各种各样的建议选项。Android Studio 生成的建议列表并非总是完美的, 但这些建议是根据最佳实践排序的, 而且它们通常遵从适当的命名规范。Android Studio 非常了解 Android SDK 和 Java 编程语言; 事实上, 它对这些主题的了解甚至远远超过你。如果谦虚地接触此工具并迫切地学习, 那么无论之前的编程经验如何, 你最后都将会成为明星程序员。

代码补全特性是上下文敏感的, 因此给出的建议将会随着光标作用域的不同而不同。如果在类作用域中输入代码, 代码补全建议将会不同于在方法作用域中输入时得到的建议。即使选择不接受代码补全建议, 但鉴于上述原因, 你也应该留意它们。

表 3-2 列出了 Android Studio 中的 4 种代码补全类型:

- Default 代码补全会在你开始输入的时候自动出现。
- Basic 代码补全类似于 Default 代码补全, 但还会在建议列表中当前选中条目的旁边显示一个 Javadoc 窗口。
- SmartType 代码补全也会显示 Javadoc, 但还会生成一个更具可选性和相关性的建议列表。
- Cyclic Expand Word 会循环遍历源文档中已经使用过的单词并允许你选择它们。

表 3-2 代码补全选项

选 项	PC 键	Mac 键	描 述
Default	无	无	Default 代码补全行为。Android Studio 会在紧挨着输入光标的位置显示一个建议列表。可以使用向上和向下箭头键在建议列表的条目之间导航, 并按 Enter 键选择某个条目

(续表)

选 项	PC 键	Mac 键	描 述
Basic	Ctrl+空格	Ctrl+空格	Basic 代码补全功能类似于 Default 代码补全, 但是还会在当前选中条目的旁边显示 Javadoc 窗口。单击 Javadoc 窗口中的向上箭头图标可以显示详细文档
SmartType	Ctrl+Shift+空格	Ctrl+Shift+空格	SmartType 代码补全功能类似于 Basic 代码补全, 但会生成更具可选性和相关性的建议列表
Cyclic Expand Word	Alt+/	Alt+/	提供已经在文档中使用过的单词。向上循环
Cyclic Expand Word (Backward)	Alt+Shift+/	Alt+Shift+?	提供已经在文档中使用过的单词。向下循环

下面开始编码, 看看代码补全是如何工作的。在 `com.apress.gerber.helloworld` 包上右击 (在 Mac 电脑上按住 `Ctrl` 键并单击), 选择 `New | Java Class`, 打开 `Create New Class` 对话框, 如图 3-4 所示。将这个类命名为 `Sandbox` 并单击 `OK` 按钮。

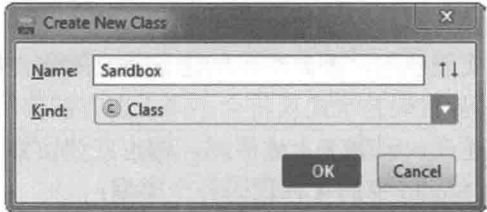


图 3-4 Create New Class 对话框

在 `Sandbox.java` 中 `Sandbox` 类的花括号内, 开始输入 `private Li` 来定义成员, 如图 3-5 所示。代码补全菜单会提供一个用于补全代码的可选列表。使用向上和向下方向键在代码补全菜单中导航。使用向下方向键选择 `List<E>` 选项, 并按 `Enter` 键。

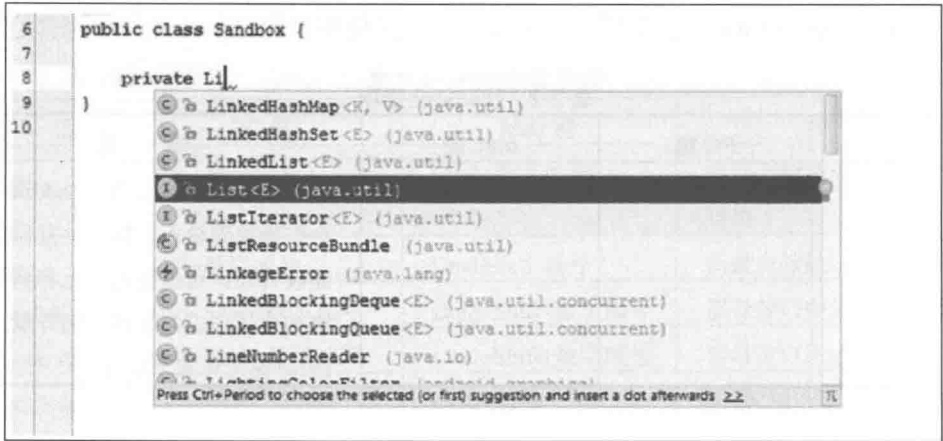


图 3-5 当开始输入时出现的代码补全菜单

Android Studio 中的默认行为是在开始输入的时候就显示代码补全建议列表。无需使用任何键盘快捷键来触发 Default 代码补全——它会自动执行。你现在应该有这样一行代码——`private List`，如图 3-6 所示。紧挨着单词 `List`，输入用于在 Java 中定义泛型的左尖括号(`<`)。注意，Android Studio 会使用向右的尖括号关闭尖括号子句并将光标置于括号中间。

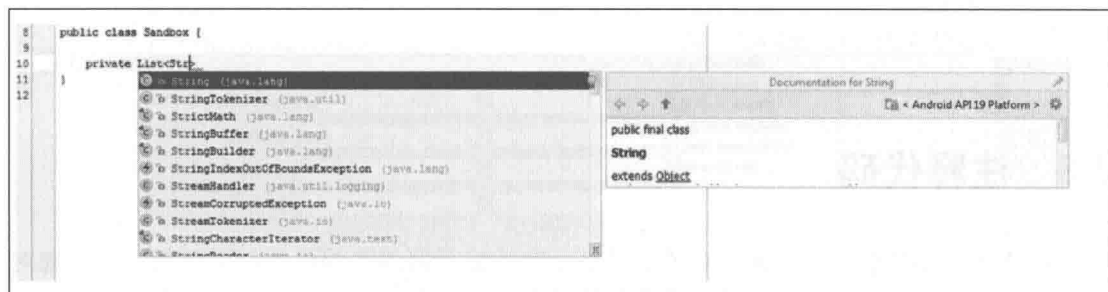


图 3-6 代码补全——将 `String` 作为列表的泛型类

在尖括号中输入 `Str` 并按 `Ctrl+空格键` 以调用 Basic 代码补全。你将会发现，在建议列表中当前选中条目(`String`)的旁边出现了关于 `String` 的 Javadoc 文档窗口。滚动 Javadoc 窗口，查看关于 `String` 的 Javadoc 文件。单击 Javadoc 窗口中的向上按钮可以在默认浏览器中显示关于 `String` 的详细 API 文档。回到 Android Studio，并输入 `Enter` 键将 `String` 选为定义 `List<String>` 所使用的泛型类。

Android Studio 的最优秀特性之一是它会为你推荐变量名称。在 `private List<String>` 之后输入一个空格，按 `Ctrl+空格键` 以激活 Basic 代码补全。Android Studio 会生成一个建议列表，但是这些变量名的描述性都不够，因此输入 `mGreetings`。小写的 `m` 代表成员(也称为字段)，为类成员名称添加 `m` 作为前缀是 Android 中的命名约定。类似的，静态类成员使用小写字母 `s` 作为前缀。你并不一定要遵从这个命名约定，但如果遵从的话，你的代码会更容易被其他人理解。要记住，本地(方法作用域内的)变量不采用 `m` 和 `s` 前缀命名约定。

将代码行修改为 `List<String> mGreetings = new`。按 `Ctrl+Shift+空格键` 以激活 SmartType 代码补全。选择 `ArrayList<>()` 来补全语句，包括结尾的分号，如图 3-7 所示。SmartType 代码补全类似于 Basic 代码补全，只有当生成建议列表中的条目时，它才比 Default 和 Basic 代码补全有着更广泛的变量作用域。例如，如果在赋值操作符的右侧使用 SmartType 代码补全，那么建议列表中通常将会包含相关的工厂方法。

注意

如果将 Android Studio 中的 JDK 设置为 7 或更高，代码补全生成的代码可能会使用菱形符号。例如，`ArrayList<String>` 可能会以 `ArrayList<>` 的形式出现在使用了泛型赋值语句的声明的右侧，就像图 3-7 中的情况。

Cyclic Expand Word 有着好听的名字，但它实际上非常简单。按住 `Alt` 键的同时按数次前向斜线即可激活“循环扩展词”。它为你提供的单词与那些已经在文档中出现的相同。在循环遍历单词的过程中，注意以黄色突出显示的部分。现在按住 `Alt` 和 `Shift` 键，同时按数次前向斜线键(Mac 电脑上的问号键)，激活“反向循环扩展词”。注意现在提供的/突出显示的单词是从光标处向下循环而非向上循环。

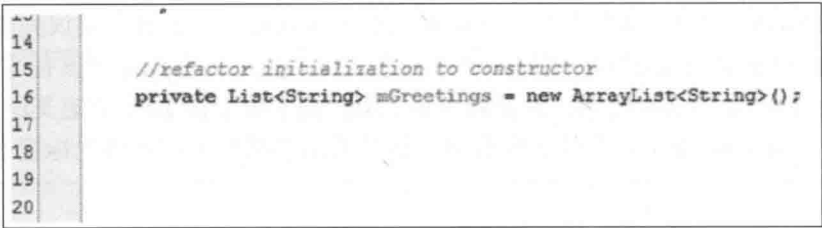


图 3-7 SmartType 代码补全

3.3 注释代码

如果从事过编程工作，就会知道注释是一些被编译器忽略的代码行，但其中包含着对于编码人员和他们的合作者来说非常重要的消息或元数据。注释可以是以两条前向斜线开始的行注释，或是以前向斜线和星号开始并以星号和前向斜线结束的注释块。在主菜单中，可以通过选择 Code | Comment 来启用注释。然而，激活注释的最好方法是使用表 3-3 中列出的键盘快捷键。

表 3-3 注释选项

选 项	PC 键	Mac 键	描 述
Toggle Comment Line	Ctrl+/ 	Cmd+/ 	使用 Java 行注释风格(例如, //...), 在注释行和非注释行之间切换。可以通过选择多行来对其执行此操作
Toggle Comment Block	Ctrl+Shift+/ 	Alt+Cmd+/ 	使用 Java 块注释风格(例如/*...*/), 在注释块和非注释块之间切换选中文本。对选中文本应用注释块将会包含注释块中的所有选中文本

在 mGreetings 声明的上方输入 refactor initialization to constructor。按 Ctrl+/ | Cmd+/将此文本转换为注释，如图 3-7 所示。使用键盘快捷键 Ctrl+/ | Cmd+/试着切换此注释的开启和关闭。

3.4 使用代码生成

使用得当的话，代码生成会是一个能够为你节省大量时间的特性。代码生成能够帮助你生成各种方法，包括构造函数、getter、setter、equals()、hashCode()、toString()等。

在使用代码生成之前，让我们检查是否已经正确进行了配置，使得 Android Studio 能够忽略掉成员名称的前缀 m 和 s。单击 File | Settings | Code Style | Java | Code Generation，弹出默认选中 Code Generation 选项卡的 Settings 对话框。如果 Static field 文本框中不包含相应的 m 和 s，那么现在输入它们并单击 Apply，接着单击 OK 按钮，如图 3-8 所示。

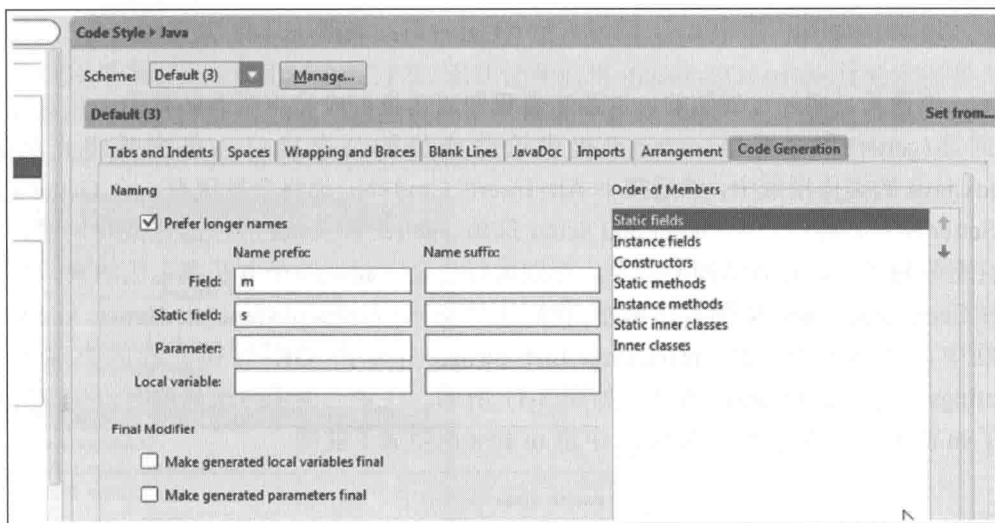


图 3-8 在 Code Generation 选项卡中为 Field 和 Static field 添加 m 和 s

3.4.1 构造函数

将光标置于 Sandbox.java 的类作用域中。要在 Android Studio 中生成构造函数，请按 Alt+Insert | Cmd+N 并选择 Constructor。如图 3-9 所示，Choose Fields to Initialize by Constructor 对话框允许你将类成员选为参数。我们想要一个无参构造函数，因此单击 Select None 按钮。在 Java 中，让构造函数接收多个不同类型参数的情况很常见。例如，可以再次激活此对话框，生成一个接收 List<String> 类型参数的构造函数并将此参数赋值为我们的成员 mGreetings:List<String>。

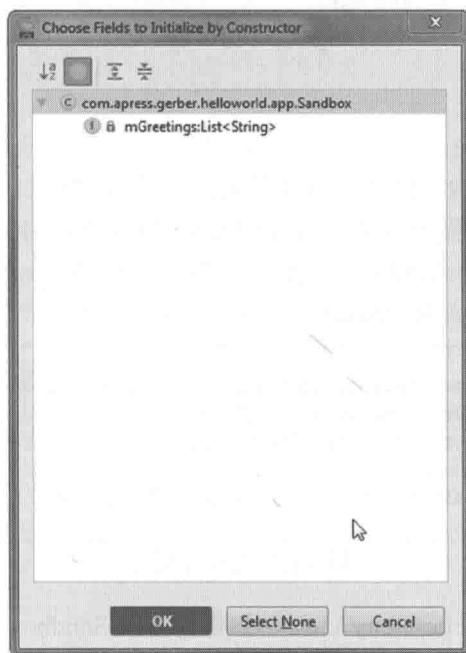


图 3-9 Choose Fields to Initialize by Constructor 对话框

3.4.2 getter/setter

Java 类通常是要经过封装的,这意味着我们通常将类成员声明为私有的,并通过公共的访问器(getter)和设置器(setter)来提供这些成员的公有接口。单击并将光标置于 Sandbox.java 的类作用域中,然后按下 Alt+Insert | Cmd+N。你将会发现有一个 Getter 选项、一个 Setter 选项,还有一个 Getter and Setter 选项。getter 和 setter 方法通常成对出现,因此除非有充分的理由需要省略掉某一个,否则最好还是一起把两个方法都生成出来。从列表中选择 Getter and Setter,如图 3-10 所示。在后续的 Select Fields to Generate Getters and Setters 对话框中,从列表中选择 mGreetings:List<String>并单击 OK 按钮。你的类现在有了 mGreetings 的 getter 和 setter 方法,如图 3-11 所示。注意当生成方法名称时,生成的代码忽略了 m 前缀,因为之前在 Settings 中将 m 和 s 声明成了前缀。

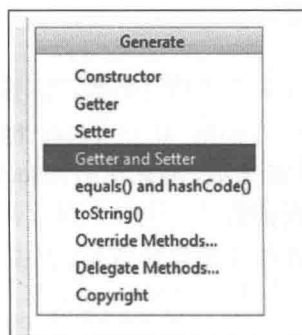


图 3-10 生成 getter 和 setter

```

10 public class Sandbox {
11
12
13     //refactor initialization to constructor
14     private List<String> mGreetings = new ArrayList<String>();
15
16
17     public Sandbox() {
18
19     }
20
21     public List<String> getGreetings() {
22         return mGreetings;
23     }
24
25     public void setGreetings(List<String> greetings) {
26         mGreetings = greetings;
27     }
28
29 }

```

图 3-11 已生成的 getter 和 setter 方法

3.4.3 重载方法

代码生成了解类的结构,因此可以在任何超类或实现接口中重载方法。Sandbox.java 是一个简单的普通 Java 对象(Plain Old Java Object, POJO)。现在修改 Sandbox 类,使其派生自 RectShape。当输入 extends RectShape 时,单词 RectShape 会以红色突出显示。如果这样的话,按 Alt+Enter 键导入 RectShape 类,如图 3-12 所示。

```

10 import android.graphics.drawable.shapes.RectShape;
11 import java.util.ArrayList;
12 import java.util.List;
13
14 public class Sandbox extends RectShape {
15

```

图 3-12 派生自超类

如果按 Ctrl+H 键激活 Hierarchy View,你将会看到 Sandbox 的类结构——RectShape、Shape 和 Object 是它的祖先,如图 3-13 所示。现在按 Alt+Insert | Cmd+N 并选择 Override

Methods。让我们重载 Shape 类的 hasAlpha()方法，如图 3-14 所示。从 Java 5 版本开始的约定是使用@Override 来注解重载方法，因此让我们将 Insert @Override 复选框保留为选中状态。@Override 注解告知编译器同时验证方法的名称和签名，以确保方法被重载。修改 hasAlpha()的返回语句，让其总是返回 true。

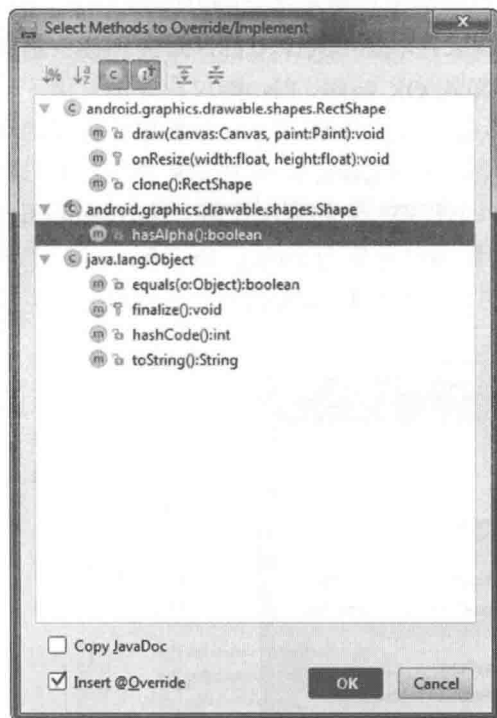


图 3-13 选择 RectShape 需要重载/实现的方法

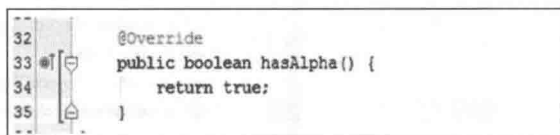


图 3-14 修改 hasAlpha()方法

3.4.4 toString()方法

Android Studio 可以为你生成 toString()方法。让我们为 Sandbox 创建一个 toString()方法并包含 mGreetings 成员。按 Alt+Insert | Cmd+N 键并选择 toString()。选择仅有的一个成员——mGreetings，并单击 OK 按钮。Android Studio 生成了一个返回字符串，例如 "Sandbox{" + "mGreetings=" + mGreetings + "}"，如图 3-15 所示。如果类中有多个成员并且将其选中，那么它们也会被追加到此方法的返回字符串中。当然，toString()生成的代码并非不可改变；需要的话可以修改这个方法，只要它返回 String 即可。

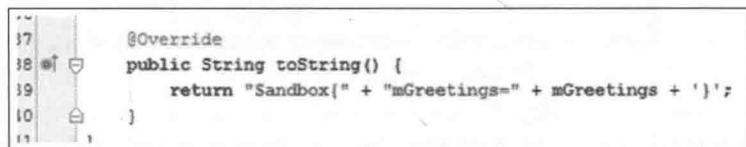


图 3-15 生成 toString()方法

3.4.5 代理方法

Android Studio 掌握类成员的信息，因而允许在类中定义代理方法并将行为代理到类成员的方法。听起来有些复杂，但实际上很简单。为了展示“代理方法”选项的工作原理，让我们直接跳到代码。

在 Sandbox.java 中，将光标置于类的作用域内。按 Alt+Insert | Cmd+N 键并接着选择 Delegate Methods。选择 mGreetings:List<String> 并单击 OK 按钮。List 接口有许多可用于代理行为的方法。简单起见，选择 add(object:E):boolean，如图 3-16 所示。如果想要代理多个方法，那么在选择这些方法时按住 Ctrl 键(Mac 电脑上的 Cmd 键)，单击 OK 按钮。

现在，Sandbox.java 中生成的 add() 方法就是一个代理，它将行为代理到 mGreetings 成员的 add() 方法，如图 3-17 所示。注意 add() 方法的参数被定义为 String，以匹配 mGreetings 的泛型定义——List<String>。代理方法不是重载方法，所以可以随意重命名代理方法，但是名称 add() 意义明确，因此我们将保留这个名称。



图 3-16 选择方法来生成代理

```

43      public boolean add(String object) {
44          return mGreetings.add(object);
45      }
46

```

图 3-17 生成的 add() 方法

3.5 插入动态模板

Android Studio 自带了大量模板,允许直接向源文件中插入预先定义好的代码。在许多 IDE 中,生成的代码只是从模板中粘贴过来,并未考虑作用域;然而,Android Studio 的模板是作用域敏感的而且还可以集成变量数据。

在开始使用 Android Studio 中的动态模板之前,让我们探索一下已有的动态模板并自己创建一个。导航至 File | Settings | Live Templates,选择 Plain 模板组。然后单击右上角的绿色加号按钮并选择 Live Templates。填写 Abbreviation、Description 和 Template 文本框,如图 3-18 所示。在应用这个模板之前,必须单击 Define 按钮,它位于窗口的底部,看上去像一个蓝色的超链接。现在选择 Java 并选择所有范围(语句、表达式、声明等)。单击 Apply 按钮。

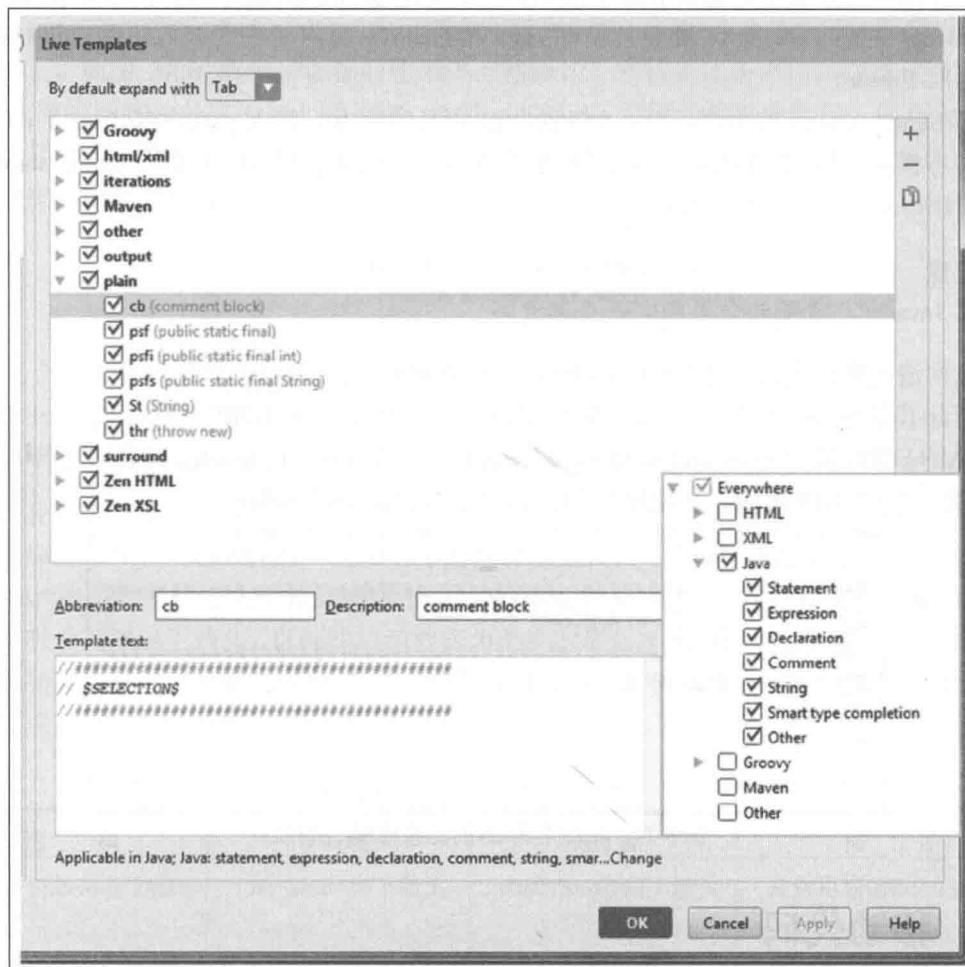


图 3-18 创建名为 cb 的动态模板(注释块)

你刚刚创建了一个名为 cb 的自定义动态模板,它将在编码任何 Java 源文件时以及在任何作用域内可用。图 3-18 中所示的红色单词 \$SELECTION\$ 是一个变量。稍后你将会看

到这个变量所起的作用。表 3-4 中描述了 Live Templates 选项。

表 3-4 Live Templates 选项

选 项	PC 键	Mac 键	描 述
Insert Live Templates	Ctrl+J	Cmd+J	激活作用域敏感的 Live Templates 列表，将会在你的文档中插入模板代码
Surround with Live Templates	Ctrl+Alt+J	Cmd+Alt+J	激活作用域敏感的 Surround with Live Templates 列表，将会使用一个作用域敏感的动态模板包裹选中内容

在离开动态模板的 Settings 页面之前，快速浏览一个现有的动态模板，它的缩写是 psfs，位于“普通”模板组中。单击 psfs 检查其内容，你将会注意到这个模板会生成 public static final String 类型的 String 常量并且仅在 Java 和 Groovy 的声明作用域内可用。单击 OK 按钮，返回 Editor。

在 Sandbox.java 的声明部分、mGreetings 定义的下方，输入 psfs 并按 Ctrl+J | Cmd+J 调用动态模板，然后按 Enter 键。给定名称和赋值以完成此语句，类似这样：public static final String HELLO = "Hello Sandbox";。

注意
在 Java 中，常量的命名约定是全部大写。

在构造函数的上面，输入单词 CONSTRUCTORS。现在将这个单词转换为一个注释块，以便引起其他程序员的注意。选中整个单词——CONSTRUCTORS，然后按 Ctrl+Alt+J | Cmd+Alt+J 键以调用 Surround with Live Templates。从 Live Templates 列表中选择 cb 并按 Enter 键，如图 3-19 所示。你刚刚应用了自己之前创建的动态模板。

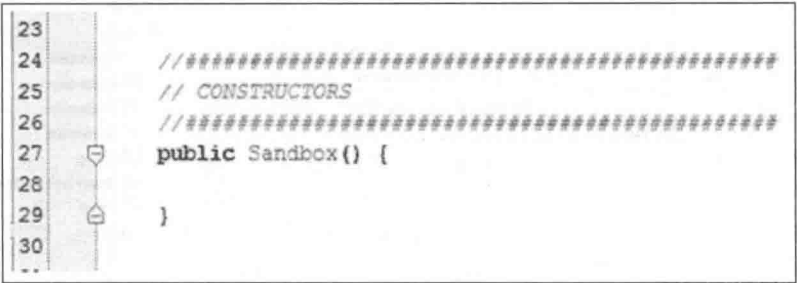


图 3-19 应用名为 cb 的动态模板(注释块)

3.6 移动代码

Android Studio 了解代码块的划分方式，因此移动行或代码块都很容易。Move Statement 和 Move Line 之间的差别在于：Move Statement 会同时考虑边界和作用域，而 Move Line 并不会考虑它们。如果选择使用 Move Statement 来移动，该语句将会保持在其包裹块作用

域的范围；如果使用 Move Line 来移动相同的语句，那么 Android Studio 会把该语句视为一行简单文本，而且会将其移至你想要的任何位置。

也可以移动整个代码块。有了 Move Statement，你需要做的全部就是将光标置于想要移动的块的起始行(带有开始花括号的那行)中的任意位置，并按 Ctrl+Shift+向下箭头 | Cmd+Shift+向下箭头或 Ctrl+Shift+向上箭头 | Cmd+Shift+向上箭头。整个块会一起移动，同时考虑其他块的边界并保持在其所属的作用域范围内。Move Line 并不会考虑作用域和边界，但是你仍然可以移动多行——首先选中它们，然后应用“向上移动行”或“向下移动行”操作，它们在 PC 和 Mac 上的快捷键一样，分别是 Alt+Shift+向上箭头和 Alt+Shift+向下箭头。

要理解 Android Studio 中的移动操作，最好就是实际操作一下。我们首先在 add()方法中创建声明。在内容为 return mGreetings.add(object);的行之后，按 Enter 键开启一个新行并输入 soutm。接着按 Ctrl+J | Cmd+J 调用 Live Templates，它会生成 System.out.println("Sandbox.add");。你可能会注意到新的代码行不会被执行，因为返回语句位于其上，如图 3-20 所示。让我们使用“上移语句”来移动这条语句。在按住 Ctrl|Cmd 和 Shift 键的同时，多次按向上箭头键。Android Studio 会重新放置此语句，但不会让你意外地将其移至一个没有任何意义的作用域内。使用 Move Line (Alt+Shift+向上箭头)再次尝试此操作并观察其行为。

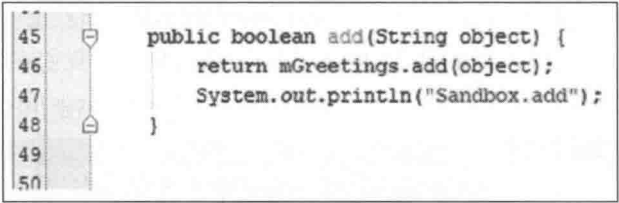


图 3-20 移动语句与移动行

让我们尝试通过另外一个例子来演示 Move Statement 的强大功能——把构造函数移至类的底部。确保在 Sandbox()声明和上方的注释块之间没有任何空行。现在，将光标置于 Sandbox()声明行中的任意位置并调用 Move Statement Down——在按住 Ctrl|Cmd 和 Shift 键的同时反复按向下箭头，直到构造函数成为类中的最后一个方法。注意整个块(包括注释)都会跳到类的底部，且不受中间其他方法的影响。表 3-5 中列出了 Move Code 操作及其键盘快捷键。

表 3-5 Move Code 选项

选 项	PC 键	Mac 键	描 述
Move Statement Down	Ctrl+Shift+向下箭头	Cmd+Shift+向下箭头	在作用域的范围，下移一条或多条语句。如果移动了块，那么整个块将会一起移动到下一个语法正确的位置

(续表)

选 项	PC 键	Mac 键	描 述
Move Statement Up	Ctrl+Shift+向上箭头	Cmd+Shift+向上箭头	与“下移语句”相同，只不过是向上移动
Move Line Down	Alt+Shift+向下箭头	Alt+Shift+向下箭头	将(多条)语句或(多个)行下移，并不考虑作用域范围或语法
Move Line Up	Alt+Shift+向上箭头	Alt+Shift+向上箭头	与下移行相同，不过是向上移动

3.7 设计代码风格

代码风格约定处于不断演进之中。有关在方法后面应该留多少个空格，或者开始花括号应该与方法签名出现在同一行还是在其下面一行，这些内容本来就没有定论。不同的组织倾向于定义其自身的代码风格，但每个程序员的代码风格也各不相同；况且你可能也有一种自己习惯的代码风格。幸运的是，Android Studio 让应用风格和组织代码变得很简单。在开始设计代码风格之前，让我们研究一下 Settings for Code Style。选择 File | Settings | Code Style，打开 Settings 对话框，如图 3-21 所示。Java 和 XML 是我们在 Android 开发中需要关注的语言。在左侧面板中打开 Code Style，选择 Java，并在 Settings 窗口中查看每个选项卡。

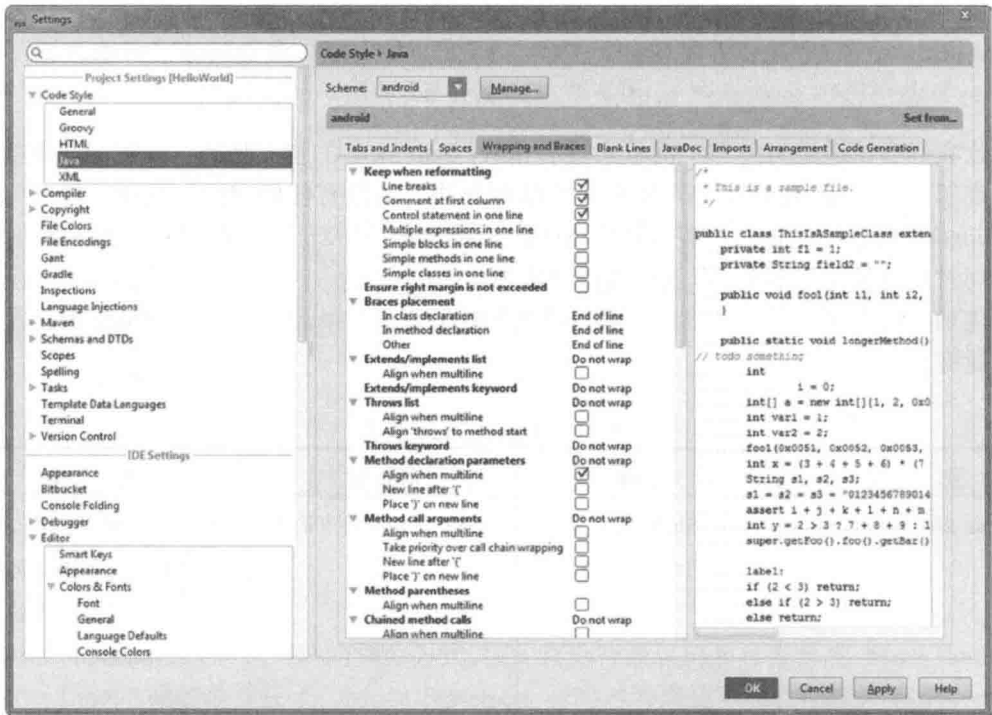


图 3-21 选中了 Code Style | Java 并显示 Wrapping and Braces 选项卡的 Settings 对话框

通过在各个选项卡的中间面板中选中和取消选中复选框来实验这些设置，并注意右侧面板中示例类针对相应风格做出的变化。单击上方的 **Manage** 按钮定义一个新方案。现在单击 **Save As** 并命名你的方案，例如 **android**，然后单击 **OK** 按钮。如果进一步修改了某个已保存的方案，那么单击 **Apply** 按钮应用它们。当使用 **Ctrl+Alt+L** | **Cmd+Alt+L** 格式化代码时，你在 **Code Style** 选项卡中选择的设置将会被应用。表 3-6 中描述了代码组织选项。

表 3-6 代码组织选项

选 项	PC 键	Mac 键	描 述
Auto-Indent Lines	Ctrl+Alt+I	Ctrl+Alt+I	根据方案设置，对当前选中的一行或多行应用缩进
Optimize Imports	Ctrl+Alt+O	Ctrl+Alt+O	从导入语句中删除所有未使用的导入。 Android Studio 在保持导入干净及相关方面做了很多努力，以至于此命令几乎是冗余的
Rearrange Code	None		根据在 Arrangement 设置中建立的规则，重新安排代码元素的顺序
Reformat Code	Ctrl+Alt+L	Cmd+Alt+L	应用特定方案的代码风格设置

3.7.1 Auto-Indent Lines 选项

Auto-Indent Lines 用于在编码过程中保持行的正确缩进。可以通过 **File | Settings | Code Style | Java | Tabs and Indents** 来访问 **Java** 中管理 **Tab** 键和缩进的规则。自动缩进行适用于当前行，如果选择了多行，则会应用于所有选中的行。

在 **Sandbox.java** 中，选择整个代码块并按 **Tab** 键。该块应该会向右移动一个 **Tab** 键的距离。现在将光标置于该块的第一行代码中，并按 **PC** 和 **Mac** 电脑上的 **Ctrl+Alt+I**。你将会发现自动缩进将该行重新置于适当的缩进位置，尽管方法块的其余部分仍未受到影响。现在通过按 **Ctrl+A** | **Cmd+A** 选中类中的所有代码，并再次按 **Ctrl+Alt+I**。这一次，整个文件都适当应用了缩进。

3.7.2 Rearrange Code 选项

Arrangement 管理着代码中元素的顺序。例如，大多数人喜欢将类成员声明置于类的顶部，接着是构造函数，然后是 **getter** 和 **setter**，等等。可以在 **Arrangement** 选项卡中编辑 **Arrangement** 设置，通过 **File | Settings | Code Style | Java | Arrangement** 可以访问 **Arrangement** 选项卡。

在上一节中，你将构造函数移到了类的底部。这通常不是它应该在的位置。从主菜单中选择 **Code | Rearrange Code**，你将会发现构造函数已经回到了它应该在的位置——声明部分的下面。**Rearrange Code** 根据 **Arrangement** 设置中的规则来执行重排操作。

3.7.3 Reformat Code 选项

Reformat Code 是最强大的 Code Style 操作，因为它为你提供了应用 Code Style 设置中定义的所有代码风格选项的能力。如你所见，可以通过主菜单中的 File | Settings | Code Style 来访问 Code Style 设置。此外，Reformat Code 允许你重新格式化当前选中的文件，或者相同类型和目录中的所有文件。更进一步，Reformat Code 允许你在命令中链接 Rearrange Entries(将会在 Java 文件中应用 Rearrange Code)以及 Optimize Imports，如图 3-22 所示。按 Ctrl+Alt+L | Cmd+Alt+L 快捷键尝试重新格式化 Sandbox.java。

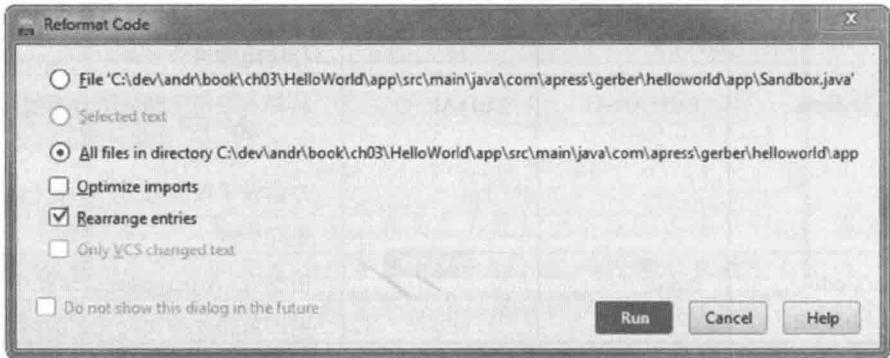


图 3-22 选中了“重排条目”的“重新格式化代码”对话框

3.7.4 Surround With

Surround With (Ctrl+Alt+T | Cmd+Alt+T)是 Surround with Live Template (Ctrl+Alt+J | Cmd+Alt+J)功能的超集。不过，Surround With 还包含使用 Java 块(例如 if/else、for、try/catch 等)来包裹选中的一条或多条语句的选项。尽管 Sandbox 类中的简单代码并不至于抛出任何未捕获的异常，但使用 try/catch 块包裹可能会抛出异常的语句确实是 Surround With 的最佳应用场景之一；这或许是键盘快捷键 Ctrl+Alt+T | Cmd+Alt+T 中包含 T 键的原因。表 3-7 中描述了 Surround With 操作。

表 3-7 Surround With 选项

选 项	PC 键	Mac 键	描 述
Surround With	Ctrl+Alt+T	Cmd+Alt+T	使用 if/else、for 和 try/catch 等 Java 代码块包裹选中的一条或多条语句
Unwrap/Remove	Ctrl+Shift+Delete	Cmd+Shift+Delete	取消选中一条或多条语句的包裹代码块

在 Sandbox.java 的 add()方法中，想要确保不会重复添加。下面使用 if/else 块包住 return mGreetings.add(object);，如图 3-23 所示。选择整行并按 Ctrl+Alt+T | Cmd+Alt+T 快捷键，调用 Surround With。现在从菜单中选择 if/else。在 if 语句的括号中，输入!mGreetings.contains(object)，并在 else 块中输入 return false;。

```

44 public boolean add(String object) {
45     System.out.println("Sandbox.add");
46     if (!mGreetings.contains(object)) {
47         return mGreetings.add(object);
48     } else {
49         return false;
50     }
51 }

```

图 3-23 利用 Surround With 包裹和取消包裹代码块

假定业务规则发生了变化，你不再关心 `mGreetings` 中的重复条目了。使用 `Unwrap/Remove` 来删除你刚刚创建的 `if/else` 块。将光标置于 `return mGreetings.add(object);` 语句中的任意位置，按 `Ctrl+Shift+Delete` | `Cmd+Shift+Delete` 快捷键，并选择 `unwrap if`。该方法现在看上去应该和你修改它之前一样。

`Surround With` 的另一个伟大应用是遍历集合。在上一节中，你自动生成了一个 `toString()` 方法。现在修改此方法，使得可以遍历 `mGreetings` 集合。删除 `toString()` 方法中的 `return` 语句，让 `toString()` 的方法体为空。然后输入 `mGreetings` 并按 `Ctrl+Alt+T` | `Cmd+Alt+T` 快捷键。从列表中选择 `Iterate Iterable`，或者按 “I” 键。再次按 `Enter` 键，将 `greeting` 作为单个元素的名称。所得到的代码是一个 `for-each` 循环。注意 `Android Studio` 知道 `mGreetings` 中包含 `String` 类型，而且它还生成了一个名为 `greeting` 的局部变量，该变量采用 `mGreetings` 的单数形式并且去掉了 `m`。进一步修改 `add()` 方法，如图 3-24 所示。

```

39 @Override
40 public String toString() {
41
42     StringBuilder stringBuilder = new StringBuilder();
43     for (String greeting : mGreetings) {
44         stringBuilder.append(greeting + " ");
45     }
46     return stringBuilder.toString().trim();
47 }

```

图 3-24 利用 Surround With 遍历 iterable

3.8 小结

本章涵盖了 `Android Studio` 中最重要的代码生成特性。我们鼓励你返回到 `File | Settings | Code Style | Java` 和 `File | Settings | Code Style` 中并多花几分钟去探索其中的各种设置。`Android Studio` 为编写代码提供了大量键盘快捷键，但你并不需要把它们全都记住。如果搞不清楚的话，可以将本书作为参考，或者导航至 `Code` 菜单并将其菜单项和子菜单作为参考。

重 构 代 码

在 Android Studio 中开发的解决方案并不总是遵循从设计到完成的直线路径。要成为高效的 Android 程序员，需要头脑灵活，并且能够在开发、调试和测试的过程中重构代码。在前面的章节中，你学习了如何用 Android Studio 生成代码；而在本章中，你将会看到如何使用 Android Studio 重构代码。重构代码的最大风险是可能会引入意外的错误。Android Studio 通过分析某些具有危险性的重构操作来降低这些风险，并接着激活 Find 工具窗口，其中标识出了所有的错误和冲突，使你可以在提交之前预览所做的修改。

本章所介绍的大多数重构操作也可以不借助 Android Studio 的重构工具来完成。然而，你应该避免使用蛮力来重构(例如，借助于全局的查找-替换选项)，因为 Android Studio 在这种情况下无法保证不会引入错误。相反，如果 Android Studio 检测到你正在尝试进行重构操作，那么它会尽量阻止你犯些愚蠢的错误。例如，在 Project 工具窗口中将某个 Java 源文件从一个包拖曳到另一个包中将会触发一次 Refactor | Move 操作，它会分析移动操作所产生的影响，让你预览修改，并接着优雅地将整个项目中所有针对该类的 import 语句修改为新包名的完整路径。

大多数重构操作都局限于一个方法或类中，因此不太可能向项目中引入错误。有风险的重构操作是指那些涉及两个或更多个资源的操作。如果重构操作引入了编译错误，Inspections Manager 将会在 Editor 中使用红色标签标识出受影响的资源。在这种情况下，可以尝试修改它们，或者通过按 Ctrl+Z | Cmd+Z 来简单地撤消整个重构操作。如果重构操作成功且没有编译错误，但却涉及大量资源，那么仍然需要进行测试，以验证没有引入任何运行时错误。第 11 章涵盖了有关测试的内容。

提示

应该将所有重要的重构修改单独做一次 Git 提交，以便之后可以轻松地回退。第 7 章涵盖了有关 Git 的内容。

本章主要关注一些最具实用性的重构操作。在开始探讨个别重构操作之前，我们需要指出 Android Studio 有一项极其便利的重构操作，称为 Refactor | Refactor This。选择此选项会显示一个上下文菜单，如图 4-1 所示，其中囊括了最有用的重构操作。此操作对应的

键盘快捷键是 `Ctrl+Alt+Shift+T` | `Ctrl+T`，对于 PC 来说，可以方便地利用首字母缩写(CAST)来记住它。



图 4-1 Refactor This 菜单中包含了最有用的重构操作

在开始使用本章中的示例之前，修改第 3 章中的 `Sandbox.java` 文件，使其不继承自任何类，且不包含任何方法或成员，类似于以下代码片段：

```
public class Sandbox {  
  
}
```

4.1 重命名

从 Project 工具窗口中选择 `Sandbox` 并接着导航到 `Refactor | Rename` 或者按 `Shift+F6` 快捷键。出现的对话框允许重命名类，以及该名称在注释、测试用例和继承类中所有出现的地方。将 `Sandbox` 重命名为 `Playpen` 并单击 `Refactor` 按钮，如图 4-2 所示。你应该看到 `Rename` 操作在项目中的效果。现在按 `Ctrl+Z` | `Cmd+Z` 快捷键来撤消 `Rename` 操作。

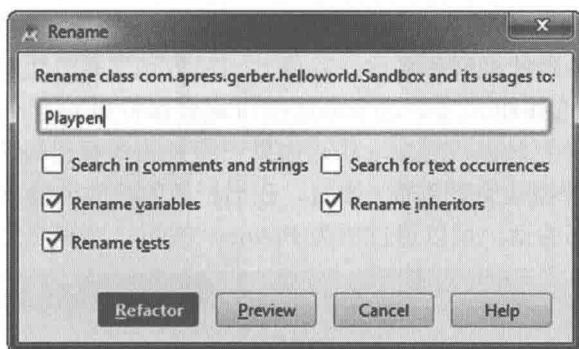


图 4-2 将 Sandbox 重命名为 Playpen

4.2 修改签名

Change Signature 操作允许修改方法的以下属性：可见性、名称、返回类型、参数和抛出的异常。在 Sandbox.java 中创建一个方法，如下面这个代码片段所示：

```
public String greetings(String message){
    return "Hello " + message;
}
```

将光标置于单词 **greetings**(以粗体突出显示)中的任意位置并按 Ctrl+F6 | Cmd+F6 快捷键，或者导航至 Refactor | Change Signature。出现的对话框允许修改方法的签名，如图 4-3 所示。

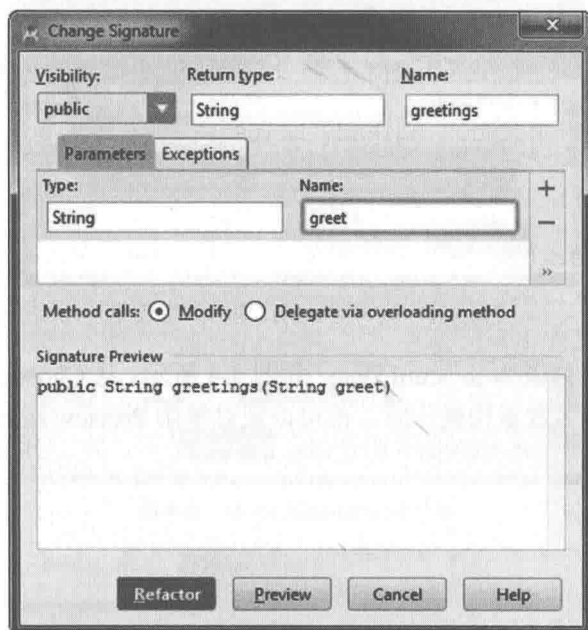


图 4-3 Change Signature 对话框

在 Parameters 选项卡中，单击 String 消息选项。把 String 参数的名称由 message 修改

为 `greet`，如图 4-3 所示。绿色的加号和红色的减号图标允许相应地增加或删除方法的参数；还可以在列表中编辑参数的类型和名称。除修改当前方法之外，也可以选中 `Delegate via Overloading method` 单选按钮。选中这个单选按钮将会保持原始方法不受影响，但会根据定义的新签名生成另一个方法。在 Java 中，如果一组方法有着相同的名称，而参数顺序和/或参数类型不同，就会被认为是重载。不过，我们所做的修改并没有让此方法满足重载的条件。如果做出该选择的话，可以通过单击 `Preview` 按钮，在提交之前预览修改。要完成操作并关闭对话框，单击 `Refactor` 按钮。

4.3 类型迁移

顾名思义，类型迁移允许把一种 Java 类型改为另一种。我们假定你创建了 `Person` 类。在后续的开发中，你发现 `Person` 过于泛化，因此创建了一个派生自 `Person` 的 `Manager` 类。如果想要将 `Person` 的所有实例修改为 `Manager`，那么使用类型迁移可以很容易地实现这一点。

将光标置于 `greetings` 方法的 `String` 声明(在以下代码片段中以粗体突出显示)中的任意位置，并按 `Ctrl+Shift+F6` | `Cmd+Shift+F6` 快捷键，或者选择 `Refactor | Type Migration`，出现的对话框如图 4-4 所示。

```
public String greetings(String greet){
    return "Hello " + greet;
}
```

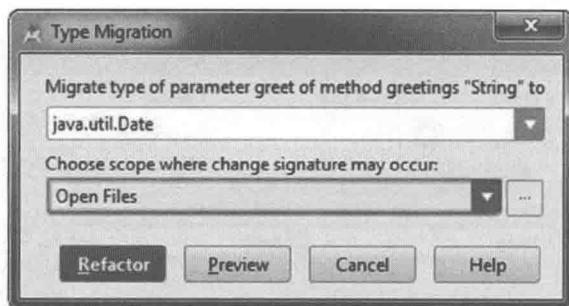


图 4-4 从字符串到日期的类型迁移

将 `java.lang.String` 修改为 `java.util.Date`，如图 4-4 所示。从 `Choose Scope` 下拉列表中选择 `Open Files`。在做大多数重构操作时，都可以通过单击 `Preview` 按钮来预览发生的修改。单击 `Refactor` 按钮。

4.4 移动

可以使用以下三种方法之一来移动源文件：

- 在 `Project` 窗口中把源文件从一个包拖曳到另一个包下
- 选择该源文件并从主菜单中导航至 `Refactor | Move`

- 在 Project 工具窗口中选择文件并按 F6 键

右击(在 Mac 电脑上按住 Ctrl 键并单击)com.apress.gerber.helloworld 包并选择 New | Package, 将包命名为 refactor。从 Project 工具窗口中, 将 Sandbox 类拖放至重构包中, 且在出现如图 4-5 所示的提示对话框时单击 OK 按钮。你在 Project 工具窗口中所做的任何拖放操作都会触发一次 Refactor | Move 操作, 这使得你能够安全地把类从一个包移到另一包中。

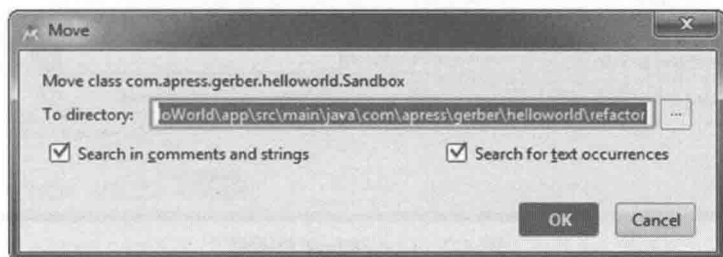


图 4-5 通过拖放操作打开的 Move 对话框

除了移动类之外, 还可能需要移动成员。在 Sandbox 类中, 定义如下所示的新成员:

```
public static final String HELLO = "Hello Android Studio";
```

将光标置于这行代码上并按 F6 键。出现的对话框允许你把成员从一个类移到另一个类中, 如图 4-6 所示。单击 Cancel 按钮取消此操作。

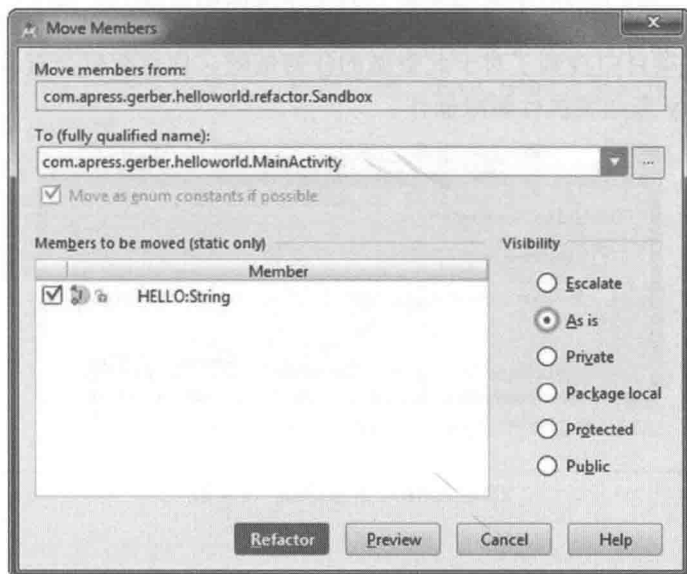


图 4-6 Move Members 对话框

4.5 复制

类似于 Move 操作, Copy 操作可以通过按键盘快捷键 F5 或者从主菜单中选择 Refactor

| Copy 来访问。在 Project 工具窗口中,选择重构包中的 Sandbox 类并按 F5 键。从 Destination Package 下拉菜单中选择 com.apress.gerber.helloworld 包并单击 OK 按钮,如图 4-7 所示。类似这样不问青红皂白地复制 Java 源文件并不是一个好主意,由于该操作模棱两可,因此会出现潜在的错误。

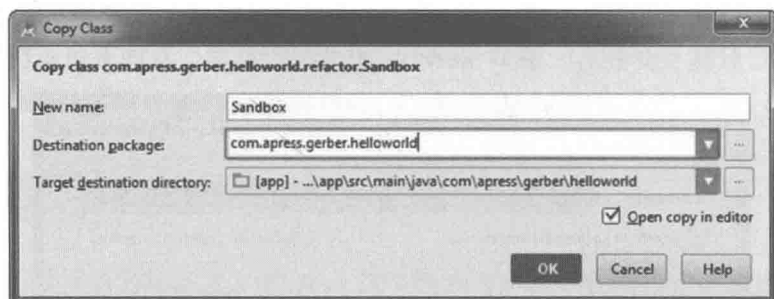


图 4-7 Copy Class 对话框

4.6 安全删除

让我们删除之前创建的副本类。在 Android Studio 中,总是可以通过在 Project 工具窗口中选择文件和资源并单击 Delete 键来删除它们。在重构包中单击 Sandbox 文件并按下 Delete 键,出现的对话框允许通过选中 Safe Delete 复选框来使用 Safe Delete 选项。使用 Safe Delete 的优势在于我们可以在执行删除之前查找该资源的依赖(它们可能会被破坏),如图 4-8 所示。如果在项目中找到了对于此资源的任何依赖,你将有机会查看它们,或者通过单击 Delete Anyway 来强制执行删除操作。

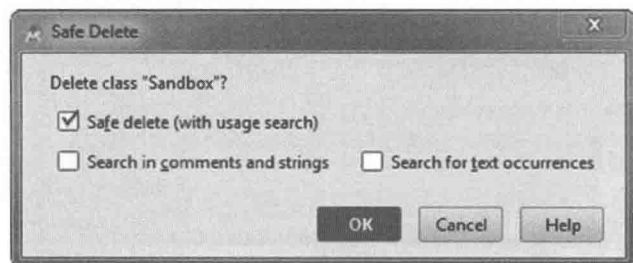


图 4-8 Safe Delete 对话框

4.7 抽取

Extract 不只是一个操作,而是一系列操作。本节涵盖了一些较为重要的抽取操作:抽取变量、抽取常量、抽取字段、抽取参数和抽取方法。删除 Sandbox 类中的所有成员和方法,让我们从头开始:

```
public class Sandbox {
    }
}
```

4.7.1 抽取变量

在 Sandbox.java 类中, 定义一个方法, 如下所示:

```
private String saySomething() {
    return "Something";
}
```

将光标置于硬编码的 **Something** 值(以粗体突出显示)的任意位置并选择 Refactor | Extract | Variable, 或者按 Ctrl+Alt+V | Cmd+Alt+V 快捷键并接着按 Enter 键, 同时取消选中 Declare final 复选框。Android Studio 抽取本地变量并根据硬编码的字符串来命名它。你应该会得到类似如下的代码:

```
private String saySomething() {
    String something = "Something";
    return something;
}
```

4.7.2 抽取常量

在开发 Android App 的过程中, 你将发现自己会使用大量的 String 作为键, 例如在 Map 和 Bundle 中。因此, 抽取常量将会为你节省大量时间。

定义一个类似于以下代码片段所示的方法。将光标置于 name_key 字符串中的任意位置并按 Ctrl+Alt+C | Cmd+Alt+C 快捷键。出现的对话框看上去应该类似于图 4-9。在这里, Android Studio 提供了一些命名建议。按照惯例, Java 中的常量均使用大写字母。选择 NAME_KEY 并按 Enter 键。

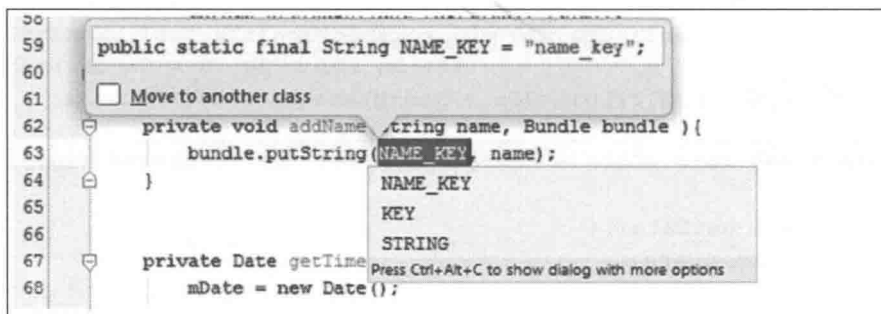


图 4-9 抽取常量 NAME_KEY

注意

你需要导入 android.os.Bundle, 以便在不出现编译时错误的情况下创建上述方法。

```
private void addName(String name, Bundle bundle) {
    bundle.putString("name_key", name);
}
```

你应该会得到一个名为 NAME_KEY 的常量, 它的定义如下所示:

```
public static final String NAME_KEY = "name_key";
```

4.7.3 抽取字段

Extract Field 会将一个本地变量转换为类中的一个字段(也称为成员)。

注意

你需要导入 `java.util.Date`, 以便能够在不产生编译时错误的情况下创建处理方法。

在 `Sandbox` 类中定义一个方法:

```
private Date getDate() {
    return new Date();
}
```

将光标置于 `Date`(以粗体突出显示)中的任意位置并按 `Ctrl+Alt+F` | `Cmd+Alt+F` 快捷键, 你将会看到一个如图 4-10 所示的对话框。在 Android 中, 命名约定是给字段(也称为成员)添加字母 `m` 作为前缀。你还会看到一个下拉菜单, 让你能够在当前方法、字段声明或构造函数中初始化该字段。选择 `Field Declaration` 并按 `Enter` 键。

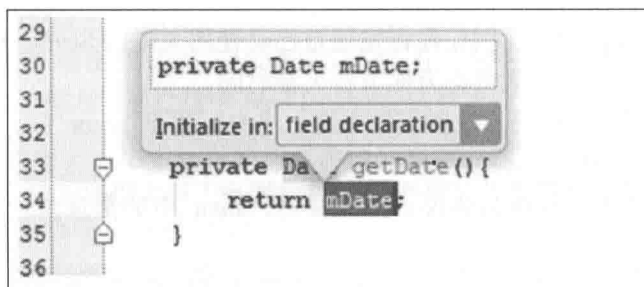


图 4-10 Extract Field 对话框

你应该会得到类似下面这样的代码:

```
private final Date mDate = new Date();
...
private Date getDate() {
    return mDate;
}
```

删除 `final` 关键字, 使得声明类似于以下代码片段所示:

```
private Date mDate = new Date();
```

4.7.4 抽取参数

Extract Parameter 允许你抽取变量并将其作为所在方法的参数。在 `Sandbox` 类中定义一个方法:

```
private void setDate() {
```

```
mDate = new Date();
}
```

将光标置于 **Date**(以粗体突出显示)中的任意位置, 按 **Ctrl+Alt+P** | **Cmd+Alt+P** 快捷键, 并按 **Enter** 键。所得到的方法看上去应该如以下代码片段所示:

```
private void setDate(Date date){
    mDate = date;
}
```

4.7.5 抽取方法

Extract Method 允许你选择一行或多行连续代码并将它们置于一个单独的方法中。想要这样做有两个原因。第一个原因是你的方法过于复杂。相对于一个有着 100 行代码的方法来说, 将算法划分为 10-20 行左右的多个独立块会让代码更容易阅读并且能够极大降低出现错误的可能性。

第二个原因是, 重复某个代码块永远不是一个好主意, 因此如果发现了重复的代码块, 那么最好抽取出一个方法, 并在重复该块的地方调用该方法。通过抽取方法并在之前使用重复代码块的地方调用它, 你可以在一处维护此方法, 而且如果需要修改它, 仅需要修改一次。在 **Sandbox** 类中重新创建以下两个方法, 如代码清单 4-1 所示。自由自在地复制和粘贴吧!

代码清单 4-1 抽取方法代码

```
private String methodHello (){

    String greet = "Hello";
    StringBuilder stringBuilder = new StringBuilder();
    for(int nC = 0; nC < 10; nC++){
        stringBuilder.append(greet + nC);
    }
    return stringBuilder.toString();

}

private String methodGoodbye (){

    String greet = "Goodbye";
    StringBuilder stringBuilder = new StringBuilder();
    for(int nC = 0; nC < 10; nC++){
        stringBuilder.append(greet + nC);
    }
    return stringBuilder.toString();

}
```

如前所述, 当发现重复的代码块或者正在复制粘贴某个代码块时, 应该考虑使用

Extract Method。选择代码清单 4-1 中以粗体突出显示的所有行。现在按 **Ctrl+Alt+M** | **Cmd+Alt+M** 快捷键来抽取方法，你将会看到一个显示方法签名的对话框。将此方法重命名为 `getGreet`，如图 4-11 所示，并单击 **OK** 按钮。

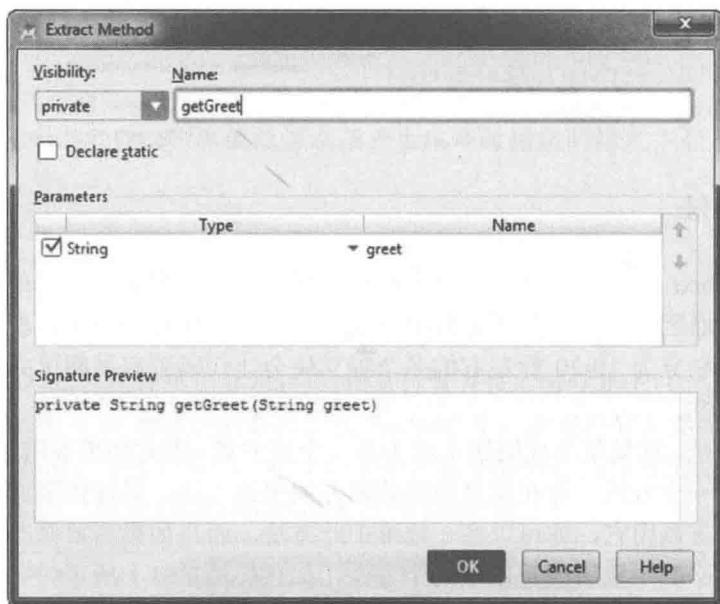


图 4-11 Extract Method 对话框

Android Studio 会扫描文件并发现还有一个相同的代码块实例。单击 **OK** 按钮接受 **Process Duplicates** 对话框中的建议，如图 4-12 所示。

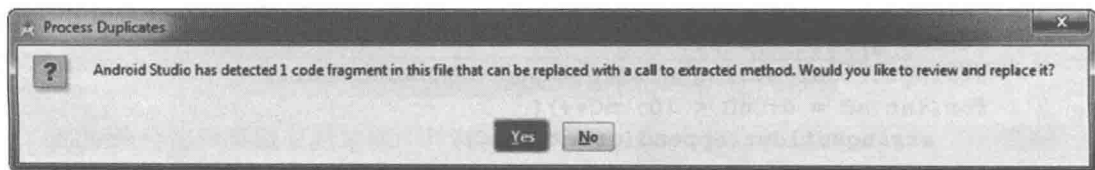


图 4-12 Process Duplicates 对话框

你应该会得到类似于清单 4-2 中所示的代码。所得到的方法仅存在于一处，现在它更容易维护了。

代码清单 4-2 Extract Method 操作所得到的代码

```
private String methodHello () {

    String greet = "Hello";
    return getGreet(greet);

}

private String methodGoodbye () {
```

```

String greet = "Goodbye";
return getGreet(greet);
}

private String getGreet (String greet){

    StringBuilder stringBuilder = new StringBuilder();
    for(int nC = 0; nC < 10; nC++){
        stringBuilder.append(greet + nC);
    }
    return stringBuilder.toString();
}

```

4.8 高级重构

本章剩余内容中介绍的均为高级重构操作。如果感兴趣的是快速上手 Android Studio, 那么你已经具备了高效使用重构操作的足够知识, 可以跳过此节。然而, 如果你很熟悉 Java 并且想要深入了解一些更加高级的重构操作, 那么请继续阅读。

删除 Sandbox.java 中的所有方法和成员, 以一个干净的类开始:

```

public class Sandbox {

}

```

右击(在 Mac 电脑上按住 Ctrl 键并单击)Project 工具窗口中的 com.apress.gerber.helloworld 包并选择 New | Java Class, 将类命名为 Minibox。修改 Minibox 的定义, 使其派生自 Sandbox 并且包含一个名为 mShovel 的成员, 如下所示:

```

public class Minibox extends Sandbox {
    private String mShovel;

}

```

4.8.1 下推成员和上拉成员

下推成员和上拉成员用于继承关系中。注意我们已经在 Minibox 类中定义了 mShovel 成员。假设我们后来决定 mShovel 对于派生自 Sandbox 的其他类来说也是有用的。要做到这一点, 打开 Minibox 类并选择 Refactor | Pull Members Up, 出现的对话框看上去类似于图 4-13。

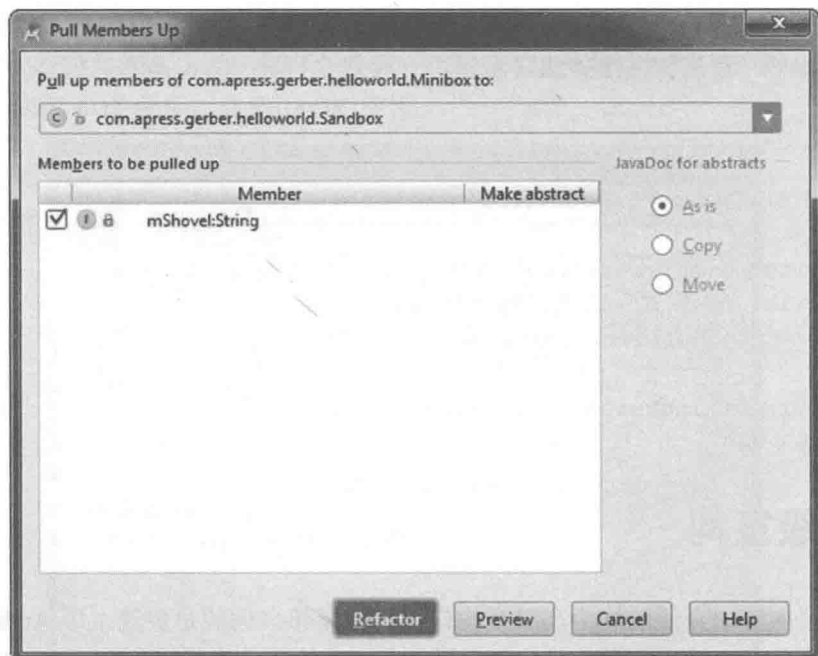


图 4-13 Pull Members Up 对话框

成员 `mShovel` 默认被选中，由于 `Sandbox` 是 `Minibox` 的超类，因此 `Pull Up Members` 复选框也默认被设置为 `com.apress.gerber.helloworld.Sandbox`。单击 `Refactor`。如果仔细检查 `Sandbox` 和 `Minibox`，你将会注意到成员 `mShovel` 属于 `Sandbox`，而且已经不在 `Minibox` 中了。作为通用的规则，如果认为某个成员对于其他派生类来说是有用的，那么你应该在继承层级中上拉该成员。要下推成员的层级，可以采用类似的步骤。

4.8.2 使用代理代替继承

右击(在 Mac 电脑上按住 `Ctrl` 键并单击)`com.apress.gerber.helloworld` 包并选择 `New | Java Class`。将类命名为 `Patio`，并让它派生自 `Sandbox`：

```
public class Patio extends Sandbox {
}

```

经过进一步的分析，我们确定了 `Patio` 不是 `Sandbox`，而是包含 `Sandbox`。要修改这个关系，导航至 `Refactor | Replace Inheritance with Delegation`。在出现的对话框中，选中 `Generate Getter for Delegated Component` 复选框，如图 4-14 所示。



图 4-14 Replace Inheritance With Delegation 对话框

Patio 类现在应该有了一个 Sandbox 成员，如以下代码片段所示：

```
public class Patio {

    private final Sandbox mSandbox = new Sandbox();

    public Sandbox getSandbox() {
        return mSandbox;
    }
}
```

4.8.3 封装字段

封装是一种面向对象策略，它通过将类成员的访问级别设为私有来隐藏它们，接着通过公共可见的 getter/setter 方法为这些成员提供公开接口。Refactor | Encapsulate Fields 类似于 Code | Generate | Getter and Setter，不过当选择 Refactor Encapsulate Fields 时你会面对更多的选项。打开 Sandbox 类并定义一个名为 mChildren 的新成员，如下面代码片段中以粗体突出显示的部分。在主菜单中，选择 Refactor | Encapsulate Fields。

```
public class Sandbox {
    private String mShovel;
    private int mChildren;
}
```

出现的对话框允许你选择字段的封装方式及其应有的访问级别。真正封装好的字段的可见性应该为私有，并且拥有公开可见的访问(getter)和设置(setter)方法。单击图 4-15 中所示的 Refactor 按钮，注意 Android Studio 已经在 Sandbox.java 类中为我们生成了 getter 和 setter 方法。

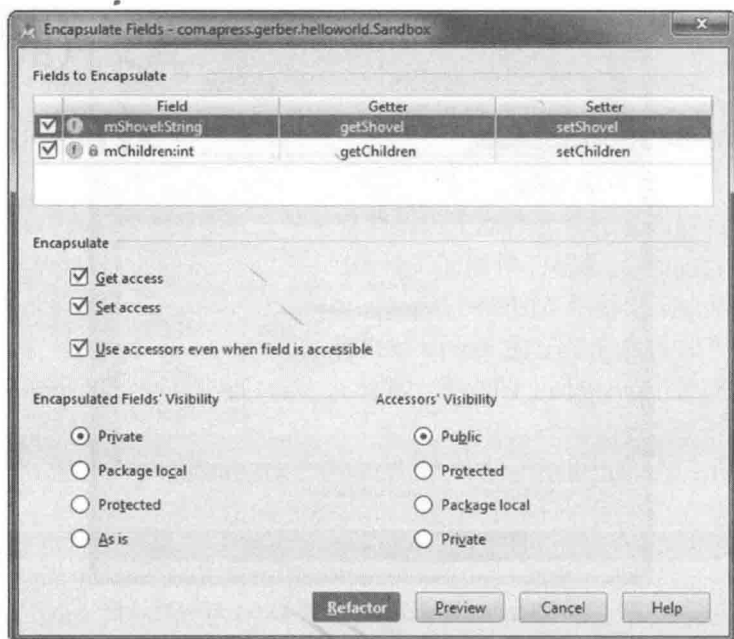


图 4-15 Encapsulate Fields 对话框

4.8.4 封装方法返回值

当需要返回一个对象而非原始类型时,封装返回值会很有用(尽管还可能会在其他场景中需要用到封装返回值)。将光标置于 `getChildren()` 方法中并导航至 **Refactor | Wrap Method Return Value**。选中 **Use Existing Class** 复选框并在 **Name** 文本框中输入 `java.lang.Integer`,在 **Wrapper Field** 中选择 `value`,如图 4-16 所示。现在单击 **Refactor** 并注意观察, `getChildren()` 方法已经返回 `Integer` 对象而非原始的 `int` 值了。



图 4-16 Wrap Return Value 对话框

4.8.5 使用工厂方法代替构造函数

将光标置于 `Sandbox` 类定义的闭合括号中。按 `Alt+Insert` | `Cmd+N` 快捷键并选择 `Constructor` 来生成一个新的构造函数。选择这两个成员，如图 4-17 所示，并单击 `OK` 按钮。

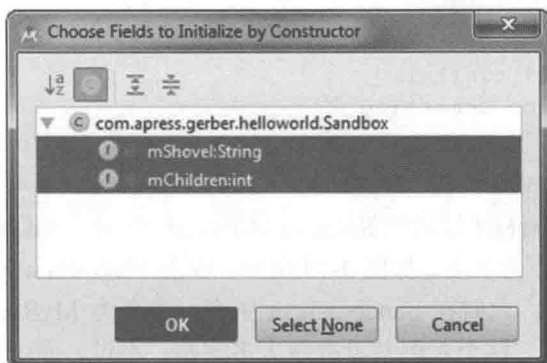


图 4-17 Choose Fields to Initialize by Constructor 对话框

将光标置于新定义的构造函数(如以下代码片段所示)中的任意位置，接着导航至 `Refactor` | `Replace Constructor with Factory Method`。出现的对话框类似于图 4-18 所示，单击 `Refactor` 按钮生成一个工厂方法。

```
public Sandbox(String shovel, int children) {
    mShovel = shovel;
    mChildren = children;
}
```



图 4-18 Replace Constructor with Factory Method 对话框

注意此时构造函数是私有的，而新的静态方法返回 `Sandbox` 类的实例，如以下代码片段所示。如果正要创建一个单例模式，那么这个操作尤其有用。

```
public static Sandbox createSandbox(String shovel, int children) {
    return new Sandbox(shovel, children);
}
```

4.8.6 将匿名类转换为内部类

在 `Sandbox` 类的构造函数中，添加下列行：

```
new Thread(new Runnable()).start();
```

将光标置于 `Runnable()` 上并按 `Alt+Enter` 快捷键激活代码补全操作。接着选择 **Implement Methods**。选择 `run` 方法并单击 **OK** 按钮。你的代码看上去应该类似于以下代码片段：

```
new Thread(new Runnable() {
    @Override
    public void run() {
        //do something
    }
}).start();
```

将光标置于 `Runnable()` 并导航至 **Refactor | Convert Anonymous to Inner**。Android Studio 建议你以 `MyRunnable` 作为类名，如图 4-19 所示。取消 **Make class static** 复选框的选中状态并单击 **OK** 按钮。注意，此时在 `Sandbox.java` 中有一个名为 `MyRunnable` 的私有内部类，它实现了 `Runnable` 接口。这个示例并没有做太多事情；不过，你可能会在代理视图行为的时候用到此类操作。



图 4-19 Convert Anonymous to Inner 对话框

4.9 小结

本章讨论了 Android Studio 中许多可用的重构操作。重构代码是所有编程项目的必要环节，而 Android Studio 中的重构工具也是出类拔萃。Android Studio 会分析操作的后果，且在提交操作之前，允许你在 **Find** 工具窗口中预览结果，以此来降低执行某些重构的风险。最重要的重构操作可通过 **Refactor | Refactor This** 对话框来完成，使用键盘快捷键 `Ctrl+Alt+Shift+T` | `Ctrl+T` 可以调用该对话框。

第 5 章

备忘录实验：第 1 部分

到目前为止，你已经熟悉了关于创建新项目、编程和重构的基础知识，是时候创建一款 Android 应用(也称为 App)了。本章介绍 4 个实验项目中的第 1 个。这些实验的目的是通过开发 App 让你熟悉 Android Studio 的用法。在这个项目中，你将要开发一款用于管理备忘事项列表的 App。核心功能是允许你创建和删除备忘并将某些备忘标记为重要。重要的事项会通过备忘文本左侧的橙色标签突出显示。这个 App 将会用到操作栏菜单、上下文菜单、用于持久化的本地数据库以及支持多选的设备上的多项选择。

图 5-1 展示了已完成的 APP 运行在模拟器上的样子。这个示例会向你展示 Android 基础操作，你还会学到如何使用内置的 SQLite 数据库来持久化数据。如果不熟悉某些主题，那么不必担心；后续章节会更加详细地涵盖这些主题。

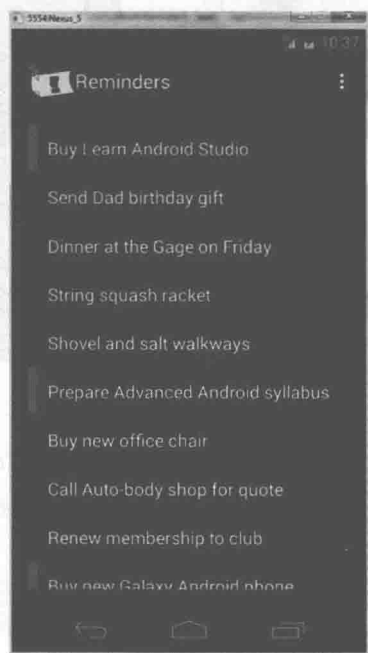


图 5-1 已完成的 App 界面

注意

我们邀请你使用 Git 克隆此项目,以便跟随学习进度,尽管你将从头开始重新创建此项目的 Git 仓库。如果你的电脑上还没有安装 Git,那么请阅读第 7 章。在 Windows 上打开 Git-bash 会话(或者 Mac 和 Linux 上的终端),导航至 C:\androidBook\reference(如果没有 reference 目录,就创建它。在 Mac 上导航至 /your-labs-parent-dir/reference/)并执行以下 git 命令: `git clone https://bitbucket.org/csgerber/reminders.git` Reminders。

要操作 Reminders App,可以使用 Action Bar 上的溢出菜单。单击溢出按钮(位于菜单栏的右侧,看上去像三个竖直排列的小点,如图 5-2 所示)会打开一个带有两个选项的菜单: New Reminder 和 Exit。单击 New Reminder 会打开如图 5-3 所示的对话框。在对话框中,可以为新备忘添加文本,然后单击 Commit 将其添加到列表中。单击 Exit 即可退出 App。

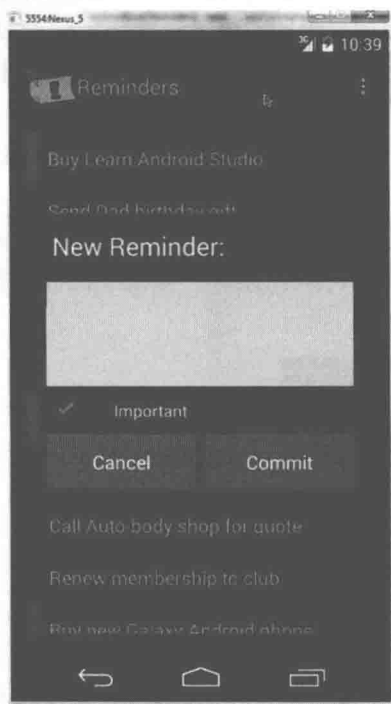


图 5-2 单击溢出菜单后的 App 界面

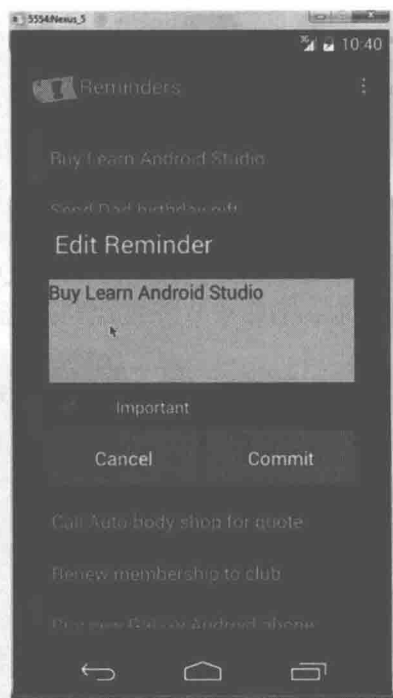


图 5-3 New Reminder 对话框

单击列表中的任意备忘会打开一个带有两个选项(Edit Reminder 和 Delete Reminder)的上下文菜单,如图 5-4 所示。在上下文菜单中点击 Edit Reminder 会打开图 5-5 中所示的 Edit Reminder 弹出对话框,可以在这里修改备忘的文本。点击上下文菜单中的 Delete Reminder 会从列表中删除备忘。

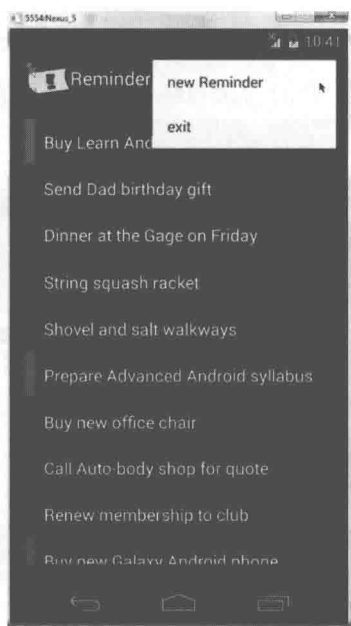


图 5-4 上下文菜单

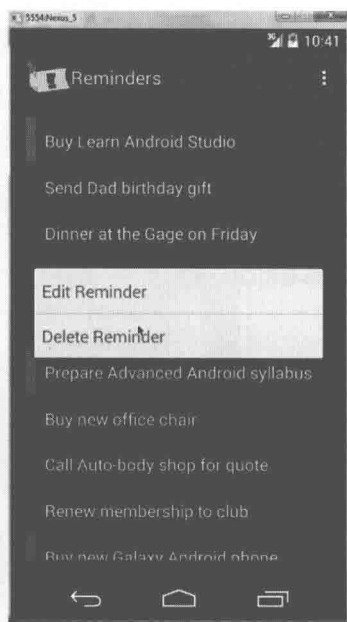


图 5-5 Edit Reminder 对话框

5.1 启动新项目

使用第1章中介绍的 New Project Wizard 在 Android Studio 中开启一个新的项目。输入 Reminders 作为应用名，将公司域名设置为 gerber.apress.com 并选择 Blank Activity 模板。将此项目保存在 C:\androidBook\Reminders 路径下。为了保持示例的一致性，将所有实验项目保存在公共文件夹(例如 C:\androidBook 或者 Mac/Linux 上的~/androidBook)中是个好主意。在向导的下一页中，选择 Phone and Tablet 并将最低 SDK 版本设置为 Android 2.2 (Froyo)。将最低 API 级别设置为 8 可以让 App 在超过 99% 的 Android 设备上可用。单击 Next 按钮，从可用模板中选择 Blank Activity，然后再次单击 Next 按钮。将 Activity 名称设置为 RemindersActivity，接着单击 Finish 按钮，如图 5-6 所示。

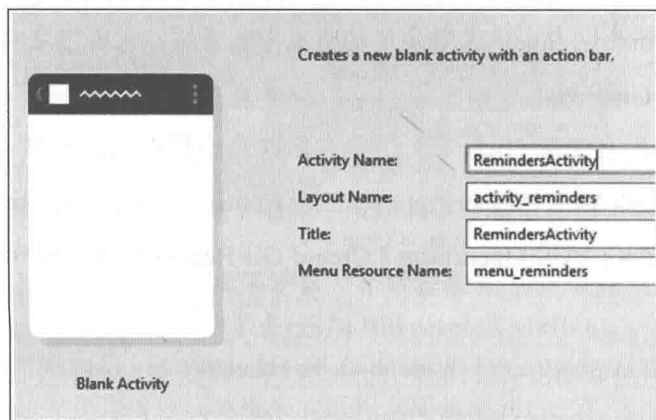


图 5-6 输入 Activity 名称

Android Studio 在 Design 模式下显示 activity_reminders.xml。activity_reminders.xml 是主 Activity 的布局，如图 5-7 所示。如第 1 章中所述，此时该项目应该能够运行在模拟器或设备上。放心地去连接设备或者启动模拟器并运行该项目吧！

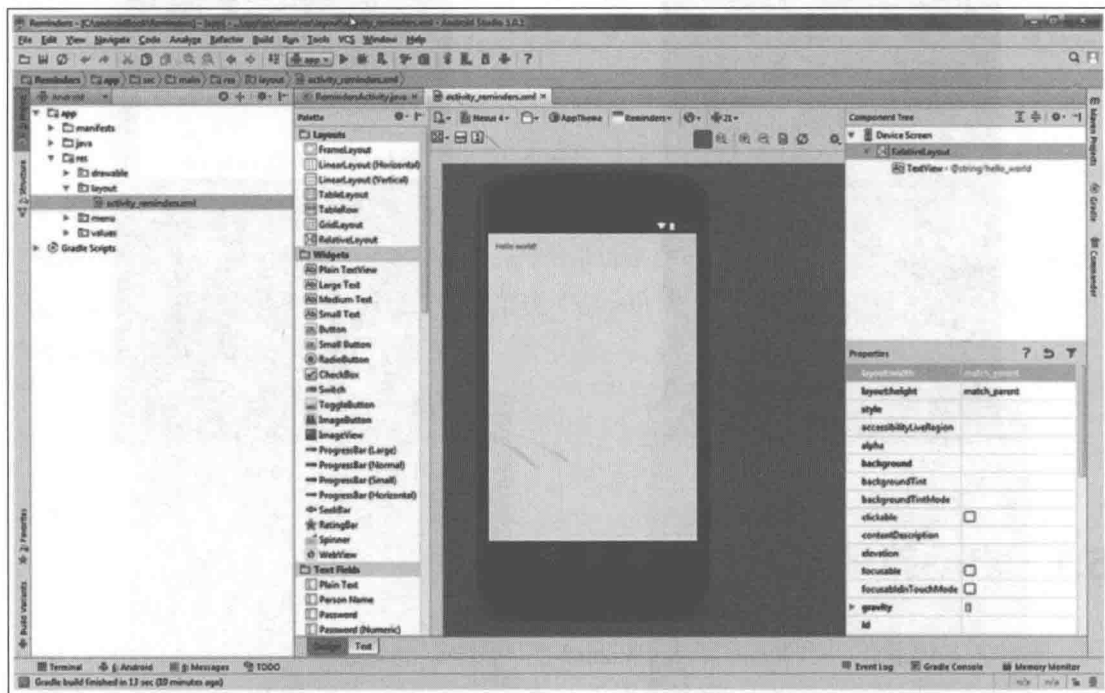


图 5-7 设计模式下的 activity_reminders

5.2 初始化 Git 仓库

创建新项目之后的第一步应该是使用版本控制来管理源代码。本书中的所有示例均使用 Git——一款与 Android Studio 无缝集成的版本控制系统，它可以从网上免费下载。第 7 章会更详细地探讨 Git 和版本控制。

如果电脑上还没有安装 Git，那么请参考第 7 章中标题为“安装 Git”的那一节(7.1 节)。从主菜单中选择 VCS | Import into Version Control | Create Git Repository(在苹果操作系统中，选择 VCS | VCS Operations | Create Git Repository)。图 5-8 和图 5-9 展示了此过程。

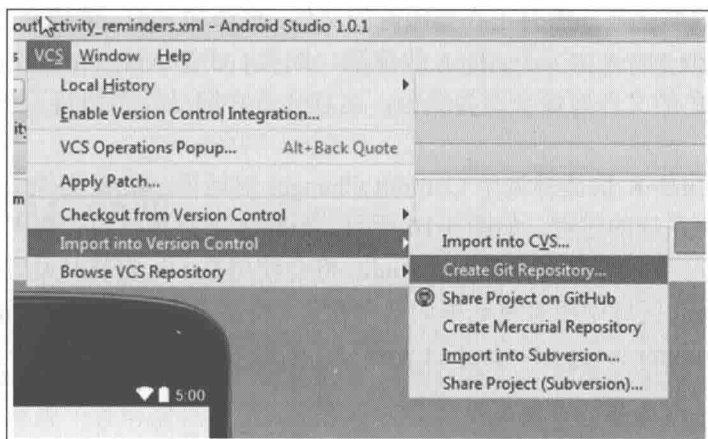


图 5-8 创建 Git 仓库

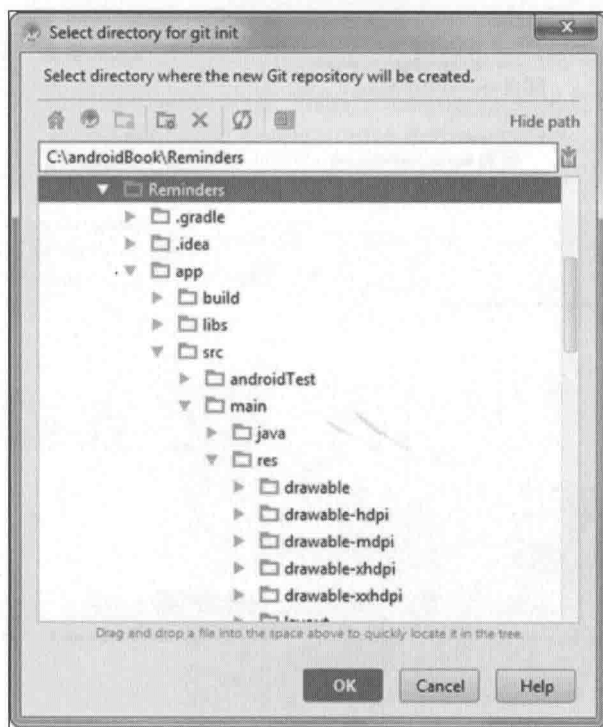


图 5-9 选择 Git 仓库的根目录

当提示选择用于 Git init 的目录时，确保在项目的根目录中初始化 Git 项目(同样，在本例中是 Reminders)。单击 OK 按钮。

你将会发现 Project 工具窗口中的大多数文件均变为了棕色，这意味着它们已经被 Git 跟踪，但尚未添加到 Git 仓库中。一旦项目处于 Git 的管控之下，Android Studio 就会使用一种配色方案来标识文件的创建、修改或删除状态。我们会在后面详细讲解这种配色方案，也可以通过以下网址来仔细研究这个主题：jetbrains.com/idea/help/file-status-highlights.html。

单击位于底部边栏的 Changes 工具按钮，打开 Changes 工具窗口并展开 Unversioned

Files 叶子标签。这里会显示所有正在被跟踪的文件。要添加它们，请选择 Unversioned Files 叶子节点并按 **Ctrl+Alt+A** | **Cmd+Alt+A** 快捷键，或者右击 Unversioned Files 叶子节点并选择 **Git | Add**。棕色的文件应该会变为绿色，这意味着它们已经添加到了 Git 中，而且现在可以提交了。

按 **Ctrl+K** | **Cmd+K** 快捷键调用 Commit Changes 对话框。提交文件是将项目的修改记录到 Git 版本控制系统的过程。如图 5-10 所示，Author 下拉菜单用于覆盖当前的默认提交者。可以把 Author 字段留空，Android Studio 将会使用在 Git 安装时设置的初始默认值。取消 Before Commit 区域中所有复选框的选中状态。将以下消息输入到 Commit Message 文本框中：Initial commit using new project wizard。单击 Commit 按钮并从下拉项中再次选择 Commit。

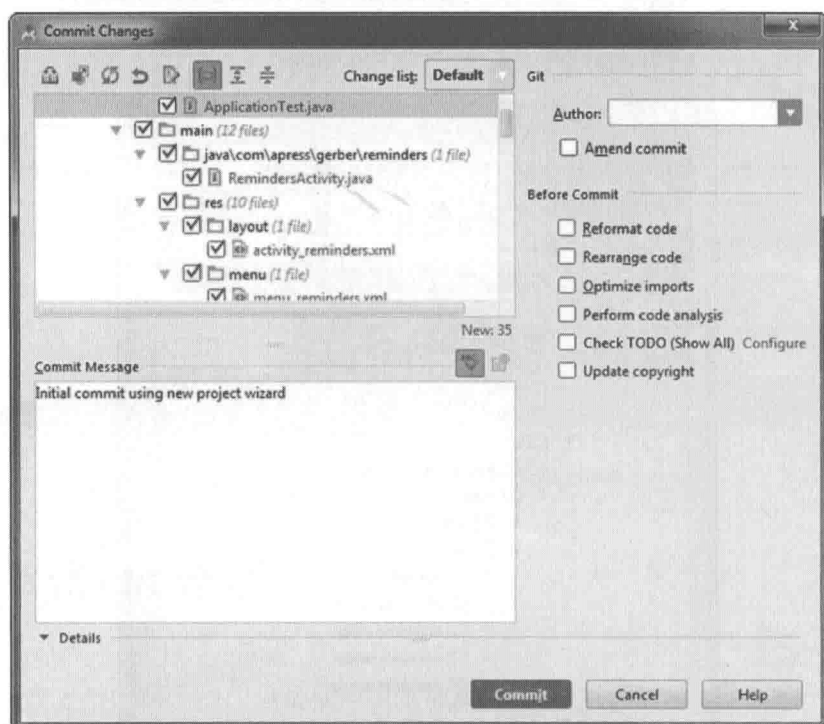


图 5-10 向 Git 提交修改

默认情况下，Project 工具窗口应该会打开。Project 工具窗口会以几种不同的方式组织项目，这取决于在窗口上方的模式下拉菜单中选择了哪种视图。默认情况下，下拉菜单被设置为 Android 视图，它会根据文件的用途进行组织，与这些文件在计算机操作系统中的组织方式无关。在探索 Project 工具窗口时，你将会注意到 App 下面有三个文件夹：manifests、java 和 res。Android manifest 文件位于 manifests 文件夹下。java 文件夹中存放着 Java 源文件。res 文件夹保存了所有 Android 资源文件。位于 res 目录下的资源可能是 XML 文件、图像、声音以及辅助定义 App 外观和 UI 体验的其他资源。在探索完 Android 视图之后，我们推荐切换到更为直观的 Project 视图，因为它直接对应于计算机上的文件结构。

注意

如果用过其他 IDE 或者较老的 beta 版 Android Studio, 你将会注意到自 Android Studio 发布时起就在 Project 工具窗口中引入了 Android 和 Package 视图。

5.3 构建用户界面

默认情况下, Android Studio 会在 Editor 的新标签页中打开与主 Activity 相关的 XML 布局文件并将其模式设置为 Design, 因此 Visual Designer 通常会是你在新项目中看到的第一个界面。Visual Designer 允许你编辑 App 的可视化布局。屏幕的中间是 Preview Pane。Preview Pane 会展现一款可视化的 Android 设备并渲染当前正在编辑的布局的效果。使用屏幕上方的预览布局控件可以控制其展现方式。这些控件能够调整预览, 而且可以用于选择不同(或多种)风格的 Android 设备——从智能手机到平板电脑或可穿戴设备。也可以修改与布局描述相关的主题。在屏幕的左侧, 你会发现 Control 调板。它包含可以拖曳到编辑区域(设备的可视化表示)的多种控件和组件。IDE 的右侧包含一个组件树, 它显示了布局中组件的层次结构。布局使用 XML。当在 Visual Designer 中做出修改时, XML 也会相应更新。可以通过单击 Design 和 Text 标签来切换可视化和文本编辑模式。图 5-11 标出了 Visual Designer 中的几个关键区域。

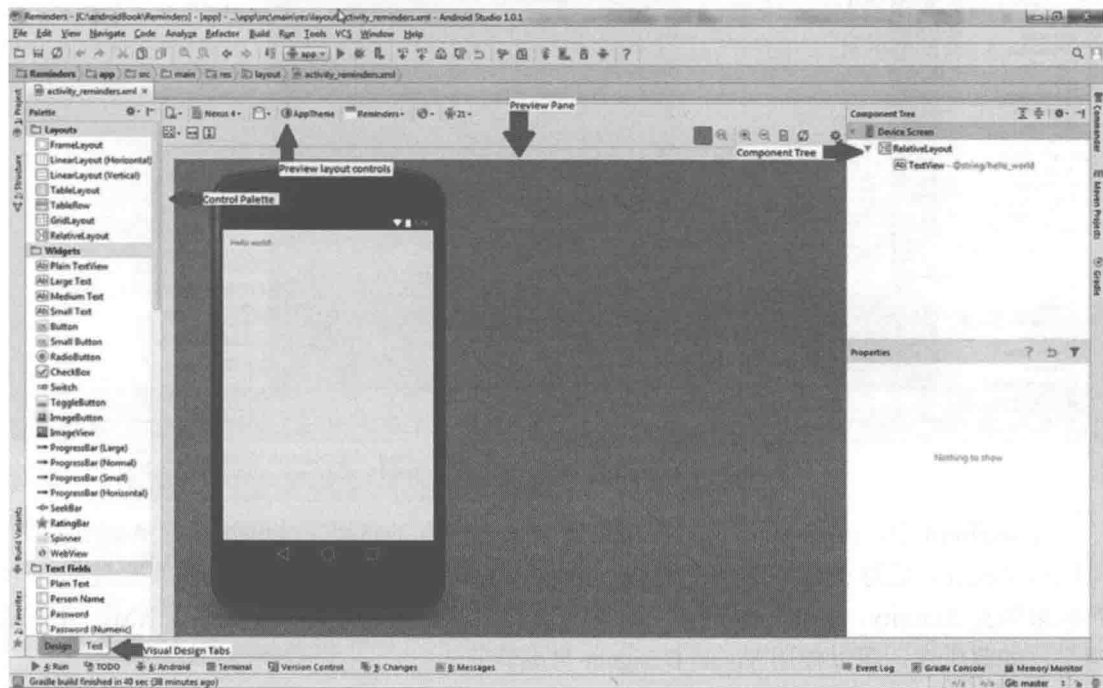


图 5-11 Visual Designer 的布局

5.3.1 使用可视化设计器

让我们从创建备忘录列表开始。单击编辑区域内的 Hello World TextView 控件，然后单击 Delete 删除它。找到调板中的 ListView 控件并将其拖曳到编辑区域。在拖曳的过程中，IDE 将会显示各种标线和对齐提示线来帮助你放置控件，当拖曳靠近边缘时，控件倾向于吸附在它们上面。放下 ListView，使得它与屏幕的顶部对齐。可以把它放在左上或顶部居中。放置之后，在 Editor 的右下方找到 Properties 视图。将 id 属性设置为 reminders_list_view。id 属性是你为控件赋予的名字，以便能够在 Java 代码中以编程的方式引用它们；在后面修改 Java 源代码时，这就是我们引用 ListView 的方法。修改 Properties 窗口中的 layout:width，将其设置为 match_parent。这将会拉伸该控件，使其占据父控件中尽可能多的空间。你将在第 8 章中学到更多关于设计布局的详细内容。现在，你的布局应该会变为图 5-12 的样子。

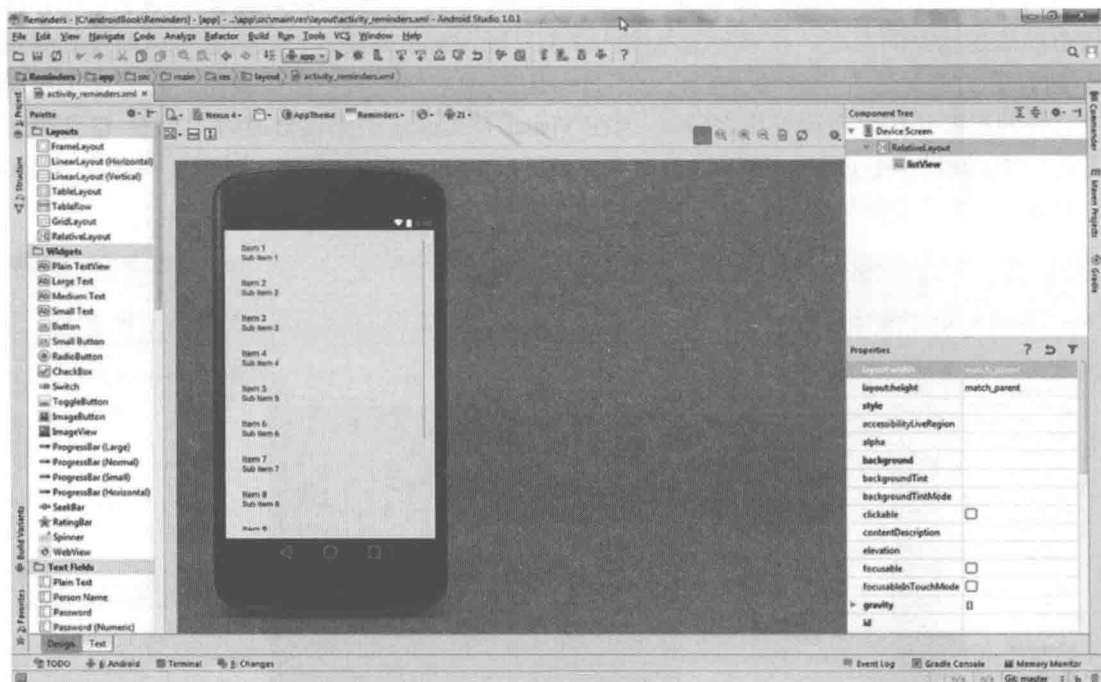


图 5-12 ListView 中的 activity_reminders 布局

在 Android 中，Activity 定义了控制用户与 App 交互的逻辑。当初次学习 Android 时，可以将 Activity 视为 App 中的一个界面，虽然 Activity 比这要复杂很多。通常会使用一个布局来填充 Activity，该布局中定义了组件在屏幕上的出现位置。布局文件以 XML 的形式定义，如前所述，可以使用 Visual Designer 来编辑它。

5.3.2 编辑布局的原始 XML

单击底部的 Text 标签，从可视化编辑切换为文本编辑。这会出现布局的原始 XML 视图，以及右侧的实时预览。你对 XML 所做的修改会立刻体现在预览面板中。在

`android:layout_height="match_parent"` 一行的下面插入 `android:background="#181818"`，将 `RelativeLayout` 的背景颜色修改为深灰色。颜色使用十六进制值来表示。关于十六进制颜色值的更多信息请参见第9章。注意，此时在已插入行(用于设置根 `ViewGroup` 的背景颜色)旁边的折叠线处出现了一个深灰色的标记。如果切换回 `Design` 模式，你将会看到现在整个布局都是深灰色的。

在 XML 布局文件中硬编码颜色值并不是最佳方案。更好的方法是在 `values` 资源文件夹下定义 `colors.xml` 文件并在那里定义自己的颜色。我们将值提取到 XML 文件(例如 `colors.xml`)中的原因是为了将这些资源保存在一处，这样便于编辑而且可以很容易地在整个项目中引用它们。

选择十六进制值 `#181818` 并通过使用 `Ctrl+X` | `Cmd+X` 快捷键或者选择 `Edit` | `Cut` 将其放入剪贴板。在这个地方输入 `@color/dark_grey`。这个值使用特定语法来引用名为 `dark_grey` 的 Android 颜色值。这个值应该定义在名为 `colors.xml` 的 Android 资源文件中，但由于项目中尚未存在此文件，因此 Android Studio 以红色突出显示了此错误。按 `Alt+Enter` 快捷键，你将会看到修改错误的选项提示。选择第二项——创建颜色值资源 `dark_grey`，然后在 `Resource value`(接下来出现的对话框中的文本框)中粘贴数值并单击 `OK` 按钮。

`New Color Value Resource` 对话框将会创建 Android 资源文件 `colors.xml` 并使用十六进制值填充它。单击 `OK` 按钮，接着在 `Add Files to Git` 对话框中单击 `OK` 按钮来将这个新文件添加到版本控制，务必选中 `Remember, Don't Ask Again` 复选框，以便不会再次被此消息打扰。图 5-13 展示了这个流程。

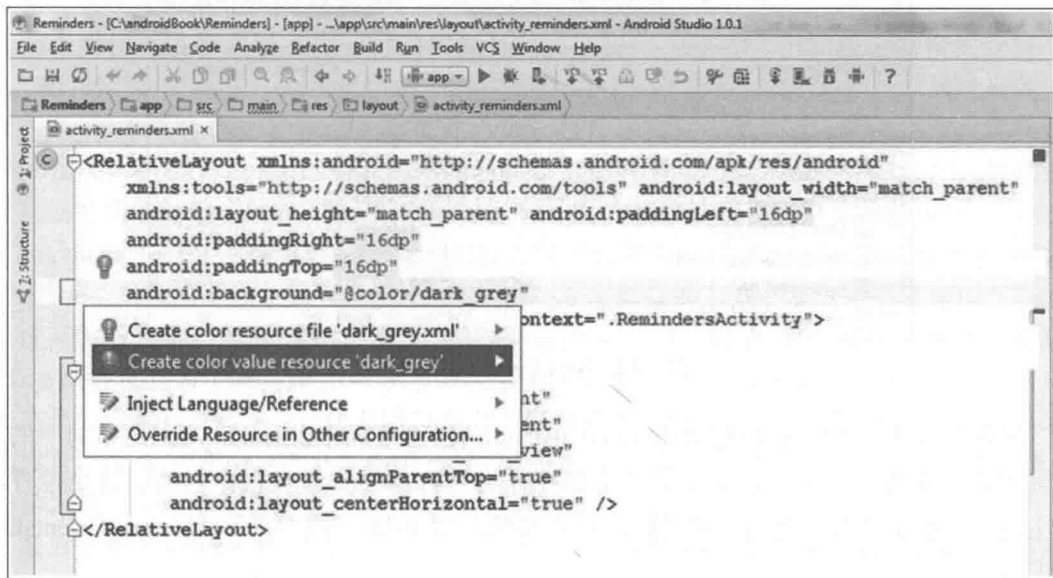


图 5-13 将硬编码的颜色值抽取为资源值

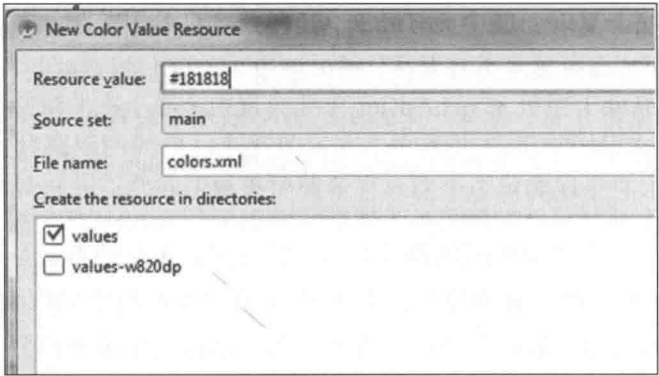


图 5-13 将硬编码的颜色值抽取为资源值(续)

在预览模式中，ListView 所包含的行布局并没有与我们选择的背景颜色产生足够的反差。要改变这些列表项的显示方式，需要在单独的布局文件中为每一行定义布局。右击 res 中的 layout 文件夹并选择 New | Layout Resource File。在 New Resource 对话框中输入 reminders_row。使用 LinearLayout 作为根 ViewGroup，其余保留图 5-14 中所示的默认值。

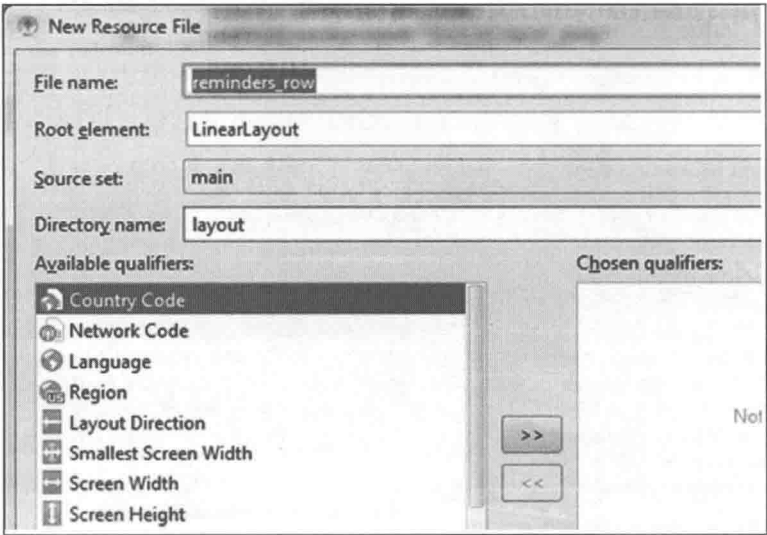


图 5-14 New Resource 对话框

现在将要为单个列表项行(individual list item)创建布局。LinearLayout 是布局中的最外层元素。使用预览面板顶部工具栏中的控件将其方向设置为竖直。当使用这个控件时要注意——水平线表示竖直方向排列，反之亦然。图 5-15 突出显示了 Change Orientation 按钮。

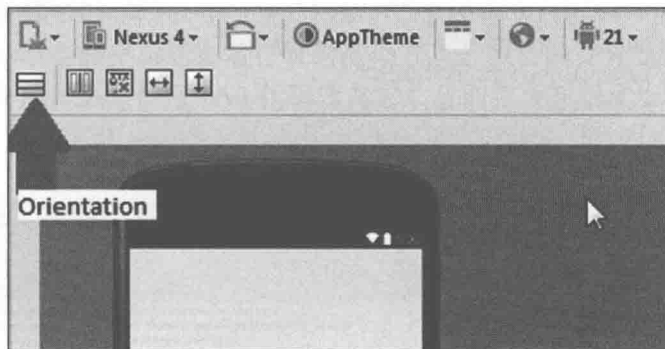


图 5-15 Change Orientation 按钮

在预览面板的右下方找到属性视图。找到 `layout:height` 属性并将其设置为 `50dp`。这个属性控制控件的高度，而 `dp` 后缀表示以密度无关的像素作为单位。这是 Android 使用的一种度量方法，使得无论在何种密度的屏幕上渲染，都能够保证布局正确缩放。可以在此视图中单击任意属性，开始输入并增量查找属性，接着单击向上或向下箭头继续查找。

将一个水平 `LinearLayout` 拖放至该竖直 `LinearLayout` 中。将一个 `CustomView` 控件拖放到水平 `LinearLayout` 内部，设置其类属性为 `android.view.View` 以创建一个通用的空视图，将它的 `id` 属性设置为 `row_tab`。在编写此书时，Android Studio 中仍有限制，不允许从调板中拖曳通用的 `View`。一旦单击 `CustomView`，将会看到一个带有不同选项的对话框，但其中并不包含通用的 `View` 类。从对话框中选择任意类型并将其放到你的布局中。使用属性面板右侧的属性窗口找到刚刚放置视图的类属性，将其修改为 `android.view.View` 来绕过这个限制。实现方法参见代码清单 5-1。

你将要使用这个通用 `View` 标签来把某条备忘标记为重要。仍然将编辑模式保持为 `Text`，把自定义 `View` 的 `layout:width` 属性修改为 `10dp`，并将其 `layout:height` 属性设置为 `match_parent`。使用 `match_parent` 值使此 `View` 控件与其父容器拥有一样的高度。切换到 `Design` 模式，在水平 `LinearLayout` 的 `Component Tree` 中拖放一个 `Large Text` 控件，并设置其宽高属性为 `match_parent`。验证你的 `Large Text` 组件被放置到了自定义视图控件的右侧。在 `Component Tree` 中，标签为 `textView` 的组件应该嵌套在(水平)`LinearLayout` 组件的内部，且位于 `view` 组件的下方。如果 `textView` 出现在 `view` 组件的上方，那么使用鼠标向下拖动它，使其附着到第二个(即最后一个)位置。将 `TextView` 控件的 `id` 值赋为 `row_text` 并将其 `textSize` 属性设置为 `18sp`。`sp` 后缀表示缩放无关像素度量，它与 `dp` 类似，但还会考虑用户的文本尺寸设置；举例来说，如果用户视力不好并希望在手机上显示大字体，那么 `sp` 会遵从此设置，而 `dp` 却不会。因此，使用 `sp` 作为 `textSize` 的单位总是一个好主意。你将在第 8 章中学习到关于屏幕度量的更多知识。

最后，将 `TextView` 控件的文本属性设置为 `Reminder Text`。切换到 `Text` 模式，并按照代码清单 5-1 所示进一步修改 XML。

代码清单 5-1 reminders_row 的 XML 布局代码

```
<?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="50dp"
    android:orientation="vertical">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="48dp">

        <view
            android:layout_width="10dp"
            android:layout_height="match_parent"
            class="android.view.View"
            android:id="@+id/row_tab" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:textAppearance="?
            android:attr/textAppearanceLarge"
            android:text="Reminder Text"
            android:id="@+id/row_text"
            android:textSize="18sp" />

    </LinearLayout>
</LinearLayout>

```

现在创建一些自定义颜色。切换到 Design 模式。在 Component Tree 中选择根 LinearLayout(竖直)。将它的 android:background 属性设置为重用我们之前定义的颜色 @color/dark_grey。选择 Component Tree 中的 row_tab 组件并将其 android:background 属性设置为 @color/green。选择 row_text 组件并将其 android:textColor 属性设置为 @color/white。与之前一样，这些颜色并没有在 colors.xml 文件中定义，你需要使用和之前一样的方法定义它们。切换到 Text 模式。反复按 F2 键，在这两个新出现的错误之间来回跳转并按 Alt+Enter 快捷键调用 IntelliSense 建议。针对这两个问题，均选择第二个建议并填充弹出对话框——将 #ffffff 用于白颜色，将 #003300 用于绿颜色。在使用建议对话框修复了这些错误之后，可以按住 Ctrl 键并用左键单击任意一个颜色，这会将你带到 colors.xml 文件中，该文件应该如代码清单 5-2 所示。

代码清单 5-2 colors.xml 文件

```

<resources>
    <color name="dark_grey">#181818</color>
    <color name="white">#ffffff</color>
    <color name="green">#003300</color>
</resources>

```

返回 activity_reminders.xml 布局文件。你现在要在此布局中将新的 reminders_row 布局连接到 ListView。切换到 Text 模式并向 ListView 元素添加以下属性：tools:listitem="@layout/

reminders_row", 如图 5-16 所示。

添加此属性并不会改变布局在运行时的渲染方式, 只会改变预览面板中的列表项视图。要使用新的布局, 必须使用 Java 代码填充它, 我们将会在后续步骤中为你展示实现方法。



图 5-16 预览面板现在渲染出了一个自定义的 ListView 布局

5.3.3 添加视觉增强效果

你刚刚已经完成了 ListView 行的自定义布局, 但现在还不能停下来。增加一些视觉增强效果会让你的 App 脱颖而出。看一下文本渲染在屏幕上的样子。如果仔细观察就会发现它稍微偏离中心并且靠近左侧的绿色标签。打开 reminders_row 布局来做一些细微调整。你想让文本向着行的竖直中心移动并且指定内边距, 使其与两侧边界有一些视觉上的分隔。使用清单 5-3 中的代码替换 TextView 元素。

代码清单 5-3 TextView 的额外属性

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:text="Reminder Text"
    android:id="@+id/row_text"
    android:textColor="@color/white"
    android:textSize="18sp"
    android:gravity="center_vertical"
    android:padding="10dp"
    android:ellipsize="end"
    android:maxLines="1"
/>
```

额外的 ellipsize 特性将会截断过长的文本, 并用省略号来填充行的结尾, 而 maxLines 特性将每个条目中的文本行数限定为 1。最后, 在内层 LinearLayout 之后、外层 LinearLayout 的关闭标签之前, 向代码清单 5-4 中再添加两个通用的视图对象, 在行的底部创建一条水平线。将外层 LinearLayout 设置为 50dp 高, 并将内层 LinearLayout 设置为 48dp 高。这两

个通用视图对象将会占据布局内部竖直方向剩余的 2dp，创建一条有斜面效果的边界，如代码清单 5-4 所示。

代码清单 5-4 用于产生有斜面效果边界的额外通用视图

```
</LinearLayout>
<view
    class="android.view.View"
    android:layout_width="fill_parent"
    android:layout_height="1dp"
    android:background="#000"/>
<view
    class="android.view.View"
    android:layout_width="fill_parent"
    android:layout_height="1dp"
    android:background="#333"/>
</LinearLayout>
```

5.3.4 向 ListView 添加条目

现在需要修改刚刚调整过的布局对应的 Activity。打开 Project 工具窗口，并在 java 源文件夹下找到 RemindersActivity 文件，它将会在 com.apress.gerber.reminders 包中。在这个文件中找到 onCreate() 方法，它应该是在类中定义的第一个方法。声明一个名为 mListView 的 ListView 成员，按照代码清单 5-5 所示的代码修改 onCreate() 方法。你需要导入 ListView 和 ArrayAdapter。

代码清单 5-5 向 ListView 添加列表项

```
public class RemindersActivity extends ActionBarActivity {

    private ListView mListView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reminders);
        mListView = (ListView) findViewById(R.id.reminders_list_view);
        //The arrayAdatper is the controller in our
        //model-view-controller relationship. (controller)
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
            //context
            this,
            //layout (view)
            R.layout.reminders_row,
            //row (view)
            R.id.row_text,
            //data (model) with bogus data to test our listview
            new String[]{"first record", "second record", "third record"});
```

```

        mListView.setAdapter(arrayAdapter);
    }
    //Remainder of the class listing omitted for brevity
}

```

这段代码使用之前定义的 `id` 属性查找 `ListView` 并删除默认的列表分隔符，以便我们之前创建的自定义斜面效果分隔符能够正确显示。该段代码还会创建一个带有一些示例列表项的适配器。`Adapter` 是 Android SDK 中定义的一个特殊 Java 类，其作用是 Model-View-Controller(模型-视图-控制器)关系中的控制器，其中 SQLite 数据库对应模型、`ListView` 对应视图，而 `Adapter` 对应控制器。`Adapter` 将模型绑定到视图并处理更新和刷新。`Adapter` 是 `ArrayAdapter` 的超类，它负责将数组的元素绑定到视图。在我们的例子中，这个视图是 `ListView`。`ArrayAdapter` 通过其构造函数接收三个参数。第一个参数是表示当前 Activity 的 `Context` 对象。`Adapter` 还需要知道使用哪个布局，以及使用布局中的哪个(些)字段来显示行数据。要满足此需求，需要同时传入布局以及其中 `TextView` 条目的 `id` 值。最后一个参数是对应于列表中每个条目的字符串数组。如果此时运行项目，你将会看到 `ArrayAdapter` 构造函数中的给定值显示在了如图 5-17 所示的列表视图中。



图 5-17 `ListView` 示例

按 `Ctrl+K` | `Cmd+K` 快捷键向 Git 提交你的修改，使用 `Adds ListView with custom colors` 作为提交消息。在项目开发的过程中，一种好的实践是向 Git 进行增量提交，同时使用提交信息来描述每次提交增加/删除/修改的特性。保持这个习惯有助于很容易地区分每次提交，而且可以为未来的合作者和用户提供后续的构建发布指南。

5.3.5 设置操作栏溢出菜单

Android 使用一个称为 Action Bar 的常用可视化元素。Action Bar 是许多 App 放置导航和其他选项的地方，它允许用户执行一些重要的任务。如果此时运行 App，你可能会注意到一个看上去像是三个竖排点的菜单图标。这些点就是所谓的溢出菜单。单击溢出菜单图标会打开一个只包含一项的菜单，名为 settings。这个菜单项由新建项目向导模板创建，本质上是一个并不执行任何操作的占位符。RemindersActivity 加载 menu_reminders.xml 文件，而它位于 res/menu 文件夹下。修改这个文件，向 Activity 中添加新菜单项，如代码清单 5-6 所示。

代码清单 5-6 新菜单项

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.apress.gerber.reminders.app.RemindersActivity">
    <item android:id="@+id/action_new"
        android:title="new Reminder"
        android:orderInCategory="100"
        app:showAsAction="never" />
    <item android:id="@+id/action_exit"
        android:title="exit"
        android:orderInCategory="200"
        app:showAsAction="never" />
</menu>
```

在前面的代码清单中，title 特性对应于菜单项中显示的文本。由于我们硬编码了这些特性，因此 Android Studio 将会把这些值标记为警告。按 F2 键在这些警告之间跳转，而按 Alt+Enter 快捷键会打开 IntelliSense 建议。你只需要按 Enter 键来接受第一项建议，为新的 String 资源输入一个名字，当出现弹出对话框时，再次按 Enter 键接受该命名资源。将 new_reminder 作为第一项的名称，将 exit 作为第二项的名称。

打开 RemindersActivity 并使用代码清单 5-7 中的文本替换 onOptionsItemSelected() 方法。你需要导入 Log 类。当在 App 中单击菜单项目时，运行时会调用此方法，传入被单击 MenuItem 的引用。switch 语句接收 MenuItem 的 itemId 作为参数，并执行 log 语句或者结束 Activity，而这取决于单击了哪一项。这个示例使用 Log.d() 方法，它会把文本写入 Android 调试日志。如果 App 包含多个 Activity，而且这些 Activity 先于当前 Activity 打开，那么调用 finish() 仅会把当前 Activity 推出回退栈并把控制权传递给接下来的 Activity。由于 RemindersActivity 是这个 App 中仅有的 Activity，因此 finish() 方法会将仅有的一个 Activity 推出回退栈，即导致 App 的结束。

代码清单 5-7 onOptionsItemSelected() 方法定义

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
```

```

case R.id.action_new:
    //create new Reminder
    Log.d(getLocalClassName(), "create new Reminder");
    return true;
case R.id.action_exit:
    finish();
    return true;
default:
    return false;
}
}

```

运行该 App 并测试这个新的菜单项。点击新的 Reminder 菜单选项并观察 Android 日志中出现的消息。当在模拟器或设备上运行 App 时，Android DDMS(Dalvik 调试监视器服务，Dalvik Debug Monitor Service)窗口将会打开，你需要在 Log Level 下面选择 Debug 选项来查看调试日志。运行 App 并与菜单项交互。在点击 New Reminder 菜单项的过程中，注意 Android DDMS 窗口中的日志。最后，按 Ctrl+K | Cmd+K 快捷键向 Git 提交你的修改，使用 Adds new reminder and exit menu options 作为提交消息。

5.4 持久化备忘录

由于 Reminders App 需要保存备忘列表，因此你需要某种持久化机制。Android SDK 和运行时提供了一个称为 SQLite 的嵌入式数据库引擎，其设计目标就是在内存受限的环境中执行操作，非常适合于移动设备。本节涵盖了 SQLite 数据库并探讨了如何保存备忘列表。我们的策略将会包含数据模型、数据库代理类和 CursorAdapter。模型将保存从数据库读取以及写入数据库的数据。代理将会是一个适配器类，它把来自 App 的简单调用转换为对 SQLite 数据库的 API 调用。最后，CursorAdapter 将继承以抽象方式处理数据访问的标准 Android 类。

5.4.1 数据模型

让我们从创建数据模型开始。右击 com.apress.gerber.reminders 包并选择 New | Java Class。将类命名为 Reminder 并按 Enter 键。使用代码清单 5-8 中的代码装饰你的类。这个类是一个简单的 POJO(Plain Old Java Object，普通 Java 对象)，定义了一些实例变量以及相应的 getter 和 setter 方法。Reminder 类包含一个整数 ID、一个字符串值和一个数值的重要程度值。ID 是一个唯一数值，用于标识每条备忘。String 的值保存了备忘的文本。重要值是一个数值标识符，用于标识某条备忘是否重要(1=重要，0=不重要)。我们在这里使用 int 而非 boolean，因为 SQLite 数据库中没有 boolean 数据类型。

代码清单 5-8 Reminder 类定义

```
public class Reminder {
```

```

private int mId;
private String mContent;
private int mImportant;

public Reminder(int id, String content, int important) {
    mId = id;
    mImportant = important;
    mContent = content;
}

public int getId() {
    return mId;
}

public void setId(int id) {
    mId = id;
}

public int getImportant() {
    return mImportant;
}

public void setImportant(int important) {
    mImportant = important;
}

public String getContent() {
    return mContent;
}

public void setContent(String content) {
    mContent = content;
}
}

```

现在你将要创建一个数据库的代理。同样，这个代理将会把简单的应用调用转换为底层的 SQLite API 调用。在 `com.apress.gerber.reminders` 包中创建一个名为 `RemindersDbAdapter` 的新类。直接把代码清单 5-9 中的代码放入新创建的 `RemindersDbAdapter` 类中。在处理导入时，你将会注意到 Android SDK 中没有 `DatabaseHelper`。我们将在后续步骤中定义 `DatabaseHelper` 类。这段代码定义了列名和索引值；用于日志的 TAG；两个数据库 API 对象；一些用于数据库名称、版本和主表名称的常量；上下文对象以及用于创建数据库的 SQL 语句。

代码清单 5-9 将要放入 `RemindersDbAdapter` 类中的代码

```

//these are the column names
public static final String COL_ID = "_id";
public static final String COL_CONTENT = "content";
public static final String COL_IMPORTANT = "important";

```

```

//these are the corresponding indices
public static final int INDEX_ID = 0;
public static final int INDEX_CONTENT = INDEX_ID + 1;
public static final int INDEX_IMPORTANT = INDEX_ID + 2;

//used for logging
private static final String TAG = "RemindersDbAdapter";

private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;

private static final String DATABASE_NAME = "dba_remdrs";
private static final String TABLE_NAME = "tbl_remdrs";
private static final int DATABASE_VERSION = 1;

private final Context mContext;

//SQL statement used to create the database
private static final String DATABASE_CREATE =
    "CREATE TABLE if not exists " + TABLE_NAME + " ( " +
        COL_ID + " INTEGER PRIMARY KEY autoincrement, " +
        COL_CONTENT + " TEXT, " +
        COL_IMPORTANT + " INTEGER );";

```

5.4.2 SQLite API

DatabaseHelper 是一个用于打开和关闭数据库的 SQLite API 类。它使用 Context，这是一个 Android 抽象类，用于提供对 Android 操作系统的访问。DatabaseHelper 是一个自定义类，需要由你来定义。使用代码清单 5-10 中的代码，将 DatabaseHelper 实现为 RemindersDbAdapter 的内部类。将上述代码置于 RemindersDbAdapter 的结尾部分，但仍然要在 RemindersDbAdapter 的结束花括号内。

代码清单 5-10 ReminderDbAdapter

```

private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.w(TAG, DATABASE_CREATE);
        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "

```

```

        + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}

```

DatabaseHelper 派生自 **SQLiteOpenHelper**，使用特定的回调方法来帮助维护数据库。回调方法是指在应用的整个生命周期中运行时环境将会调用的方法，它们使用参数中提供的 **SQLiteDatabase** 变量来执行 SQL 命令。构造函数是数据库初始化的地方。构造函数将数据库名和版本号传递给它的超类；接着由超类完成建立数据库的繁重工作。当需要创建数据库时，**onCreate()**方法会被运行时自动调用。此操作仅在 App 首次启动且数据库尚未创建时运行一次。每当数据库需要更新时，**onUpgrade()**方法就会被调用，例如当开发者修改了模式时。如果确实修改了数据库模式，那么务必将 **DATABASE_VERSION** 的值增加 1，这样 **onUpgrade()**就可以处理其余的事情了。如果忘记增加 **DATABASE_VERSION**，那么即使在调试模式下，你的 App 也会崩溃。在上述代码中，我们使用 SQL 命令在运行 **onCreate()**方法重新建表之前删除了数据库中仅有的一个表。

代码清单 5-11 中的代码演示了如何使用 **DatabaseHelper** 打开和关闭数据库。构造函数保存了一个 **Context** 实例，并将它传给 **DatabaseHelper**。**open()**方法初始化助手类并使用它来获取数据库的实例，而 **close()**方法使用助手类来关闭数据库。将此代码加入到 **RemindersDbAdapter** 类里面，置于所有成员变量定义之后、**DatabaseHelper** 内部类定义之前。在处理导入时，使用 **android.database.SQLException** 类。

代码清单 5-11 数据库的 Open 和 Close 方法

```

public RemindersDbAdapter(Context ctx) {
    this.mCtx = ctx;
}

//open
public void open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();
}

//close
public void close() {
    if (mDbHelper != null) {
        mDbHelper.close();
    }
}

```

代码清单 5-12 包含了在 **tbl_remdrs** 表中处理 **Reminder** 对象创建、读取、更新和删除操作的所有逻辑。这些通常被称为 **CRUD** 操作；**CRUD** 表示创建、读取、更新和删除。在 **RemindersDbAdapter** 类的内部添加上述代码，置于 **close()**方法的后面。

代码清单 5-12 数据库 CRUD 操作

```

//CREATE
//note that the id will be created for you automatically
public void createReminder(String name, boolean important) {
    ContentValues values = new ContentValues();
    values.put(COL_CONTENT, name);
    values.put(COL_IMPORTANT, important ? 1 : 0);
    mDb.insert(TABLE_NAME, null, values);
}

//overloaded to take a reminder
public long createReminder(Reminder reminder) {
    ContentValues values = new ContentValues();
    values.put(COL_CONTENT, reminder.getContent()); // Contact Name
    values.put(COL_IMPORTANT, reminder.getImportant()); // Contact Phone Number

    // Inserting Row
    return mDb.insert(TABLE_NAME, null, values);
}

//READ
public Reminder fetchReminderById(int id) {
    Cursor cursor = mDb.query(TABLE_NAME, new String[]{COL_ID,
        COL_CONTENT, COL_IMPORTANT}, COL_ID + "=?",
        new String[]{String.valueOf(id)}, null, null, null, null
    );
    if (cursor != null)
        cursor.moveToFirst();

    return new Reminder(
        cursor.getInt(INDEX_ID),
        cursor.getString(INDEX_CONTENT),
        cursor.getInt(INDEX_IMPORTANT)
    );
}

public Cursor fetchAllReminders() {
    Cursor mCursor = mDb.query(TABLE_NAME, new String[]{COL_ID,
        COL_CONTENT, COL_IMPORTANT},
        null, null, null, null, null, null
    );

    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}

//UPDATE

```

```

public void updateReminder(Reminder reminder) {
    ContentValues values = new ContentValues();
    values.put(COL_CONTENT, reminder.getContent());
    values.put(COL_IMPORTANT, reminder.getImportant());
    mDb.update(TABLE_NAME, values,
        COL_ID + "=?", new String[] {String.valueOf(reminder.getId())});
}

//DELETE
public void deleteReminderById(int nId) {
    mDb.delete(TABLE_NAME, COL_ID + "=?", new String[] {String.valueOf(nId)});
}

public void deleteAllReminders() {
    mDb.delete(TABLE_NAME, null, null);
}

```

这些方法均使用 `SQLiteDatabase mDb` 变量生成并执行 SQL 语句。如果熟悉 SQL 的话，你可能会猜到这些 SQL 语句的形式将会是 `INSERT`、`SELECT`、`UPDATE` 或 `DELETE`。

两个创建方法均使用一个特殊的 `ContentValues` 对象，它是一个数据梭，用于将数据值传递给数据库对象的 `insert` 方法。数据库进而会将这些对象转换为 SQL `insert` 语句并执行。有两个读取方法，一个用于获取单独一条备忘，而另一个用于获取遍历所有备忘的游标。稍后你会在一个特殊的 `Adapter` 类中使用 `Cursor`。

`update` 方法类似于第二个创建方法。不过，此方法会调用较底层数据库对象的更新方法，这将会生成并执行 `update` SQL 语句而非 `insert`。

最后，有两个 `delete` 方法。第一个接收一个 `id` 参数并使用数据库对象来生成和执行对于某条特定备忘的 `delete` 语句。第二个方法要求数据库生成并执行删除表中所有备忘的 `delete` 语句。

此时，你需要一种从数据库获取备忘并加入到 `ListView` 中的方法。代码清单 5-13 演示了通过继承之前所见的特殊 `Android Adapter` 类来将数据库值绑定到单个行对象所需的逻辑。在 `com.apress.gerber.reminders` 包中创建一个名为 `RemindersSimpleCursorAdapter` 的新类，并使用以上代码装饰它。在处理导入时，使用 `android.support.v4.widget.SimpleCursorAdapter` 类。

代码清单 5-13 `ReminderSimpleCursorAdapter` 代码

```

public class RemindersSimpleCursorAdapter extends SimpleCursorAdapter {

    public RemindersSimpleCursorAdapter(Context context, int layout,
        Cursor c, String[] from, int[] to, int flags) {
        super(context, layout, c, from, to, flags);
    }

    //to use a viewholder, you must override the following two methods and
    //define a ViewHolder class
    @Override

```

```

public View newView(Context context, Cursor cursor, ViewGroup parent) {
    return super.newView(context, cursor, parent);
}

@Override
public void bindView(View view, Context context, Cursor cursor) {
    super.bindView(view, context, cursor);

    ViewHolder holder = (ViewHolder) view.getTag();
    if (holder == null) {
        holder = new ViewHolder();
        holder.colImp = cursor.
            getColumnIndexOrThrow(RemindersDbAdapter.COL_IMPORTANT);
        holder.listTab = view.findViewById(R.id.row_tab);
        view.setTag(holder);
    }
    if (cursor.getInt(holder.colImp) > 0) {
        holder.listTab.setBackgroundColor(context.getResources().
            getColor(R.color.orange));
    } else {
        holder.listTab.setBackgroundColor(context.getResources().
            getColor(R.color.green));
    }
}

static class ViewHolder {
    //store the column index
    int colImp;
    //store the view
    View listTab;
}
}

```

我们向 `ListView` 注册 `Adapter` 来显示备忘。在运行时，随着用户加载并滚动列表，`ListView` 将会利用屏幕上的单个 `View` 对象反复调用 `Adapter` 中的 `bindView()` 方法。`Adapter` 的职责就是使用列表项来填充这些视图。在这个代码示例中，我们使用 `Adapter` 的一个子类，名为 `SimpleCursorAdapter`。这个类使用 `Cursor` 对象，它可以跟踪表中的行。

你看到了一个 `ViewHolder` 模式的示例。这是一种常见的 `Android` 模式，将一个小 `ViewHolder` 对象作为标签绑定到每个视图。这个对象通过使用数据源(在本例中是 `Cursor`)为列表中的 `View` 对象添加装饰。`ViewHolder` 被定义为一个静态内部类，含有两个实例变量——一个用于 `Important` 表列的索引，另一个用于在布局中定义的 `row_tab` 视图。

`bindView()` 方法首先调用超类方法，将通过游标获取到的值映射到 `View` 中的元素。接下来，它会检查 `holder` 是否已经绑定到了标签并在需要时创建新的 `holder`。然后，`bindView()` 方法通过使用之前定义的 `Important` 列索引和 `row_tab` 配置 `holder` 的实例变量。在找到或配置了 `holder` 之后，它使用当前备忘 `COL_IMPORTANT` 常量对应的值来决定 `row_tab` 应该使用哪种颜色。该例使用了一种新的橙色——`<color name="orange">#ffff381a</color>`，你需要将其加入到 `colors.xml` 文件中。

你之前使用 `ArrayAdapter` 来管理模型和视图之间的关系。`SimpleCursorAdapter` 遵从相同的模式，只不过它的模型是 `SQLite` 数据库。将代码清单 5-14 中的代码修改为使用新的 `RemindersDbAdapter` 和 `RemindersSimpleCursorAdapter`。

代码清单 5-14 `ReminderActivity` 代码

```
public class RemindersActivity extends ActionBarActivity {

    private ListView mListView;
    private RemindersDbAdapter mDbAdapter;
    private RemindersSimpleCursorAdapter mCursorAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reminders);
        mListView = (ListView) findViewById(R.id.reminders_list_view);
        mListView.setDivider(null);
        mDbAdapter = new RemindersDbAdapter(this);
        mDbAdapter.open();

        Cursor cursor = mDbAdapter.fetchAllReminders();

        //from columns defined in the db
        String[] from = new String[]{
            RemindersDbAdapter.COL_CONTENT
        };

        //to the ids of views in the layout
        int[] to = new int[]{
            R.id.row_text
        };

        mCursorAdapter = new RemindersSimpleCursorAdapter(
            //context
            RemindersActivity.this,
            //the layout of the row
            R.layout.reminders_row,
            //cursor
            cursor,
            //from columns defined in the db
            from,
            //to the ids of views in the layout
            to,
            //flag - not used
            0);

        // the cursorAdapter (controller) is now updating the listView (view)
        //with data from the db (model)
```

```
mListView.setAdapter(mCursorAdapter);  
}  
//Abbreviated for brevity  
}
```

如果此时运行该 App，将不会在列表中看到任何内容；屏幕完全是空的，因为最后的修改插入的是 SQLite 功能而非示例数据。按 **Ctrl+K** | **Cmd+K** 快捷键并提交你的修改，附带消息为“为备忘录添加 SQLite 数据库持久化支持以及用于重要备忘的新颜色”。作为挑战，可以尝试使用新的 **RemindersDbAdapter** 将示例条目添加回来。下一章会涵盖这些内容，因此你可以继续阅读并检查自己的工作。

5.5 小结

此时，你有了一款趋于成熟的 Android App。在本章中，你首次学习了如何建立 Android 项目并使用 Git 控制源代码。你还探索了如何在 Design 和 Text 模式中编辑 Android 布局。你已经看到了在 Action Bar 中创建溢出菜单的示例。本章最后探讨了 ListView 和 Adapter，并将数据绑定到了内置的 SQLite 数据库。在接下来的一章中，你将添加创建和编辑备忘的功能并完成这个 App。

第 6 章

备忘录实验：第 2 部分

本章涵盖了通过使用自定义对话框获取用户输入的方法。我们还会继续演示适配器和 SQLite 数据库的使用方法。在这一章中，我们会完成在 5 章中开始的实验。

6.1 添加/删除备忘

第 5 章中的示例展示了一个没有任何备忘的空界面。要查看带有备忘列表的 App 布局，最好的方法是在 App 启动时添加一些示例备忘。如果尝试解决前一章中提出的挑战问题，那么请将你的代码与代码清单 6-1 中的修改做比较。代码清单 6-1 中的代码会检查是否有任何已保存的实例；如果没有，就先创建一些示例数据。为此，代码调用了 `DatabaseAdapter` 中的两个方法；一个用于清除所有备忘，而另一个用于插入一些备忘。

代码清单 6-1 添加一些示例备忘

```
public class RemindersActivity extends ActionBarActivity {

    private ListView mListView;
    private RemindersDbAdapter mDbAdapter;
    private RemindersSimpleCursorAdapter mCursorAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_reminders);
        mListView = (ListView) findViewById(R.id.reminders_list_view);
        mListView.setDivider(null);
        mDbAdapter = new RemindersDbAdapter(this);
        mDbAdapter.open();
    }
}
```

```

        if (savedInstanceState == null) {
            //Clear all data
            mDbAdapter.deleteAllReminders();
            //Add some data
            mDbAdapter.createReminder("Buy Learn Android Studio", true);
            mDbAdapter.createReminder("Send Dad birthday gift", false);
            mDbAdapter.createReminder("Dinner at the Gage on Friday", false);
            mDbAdapter.createReminder("String squash racket", false);
            mDbAdapter.createReminder("Shovel and salt walkways", false);
            mDbAdapter.createReminder("Prepare Advanced Android syllabus",
                true);
            mDbAdapter.createReminder("Buy new office chair", false);
            mDbAdapter.createReminder("Call Auto-body shop for quote", false);
            mDbAdapter.createReminder("Renew membership to club", false);
            mDbAdapter.createReminder("Buy new Galaxy Android phone", true);
            mDbAdapter.createReminder("Sell old Android phone - auction",
                false);
            mDbAdapter.createReminder("Buy new paddles for kayaks", false);
            mDbAdapter.createReminder("Call accountant about tax returns",
                false);
            mDbAdapter.createReminder("Buy 300,000 shares of Google", false);
            mDbAdapter.createReminder("Call the Dalai Lama back", true);
        }
        //Removed remaining method code for brevity...
    }

    //Removed remaining method code for brevity...
}

```

这里多次调用了 `createReminder()` 方法，参数均是一个包含备忘文本的 `String` 值和一个表示该备忘是否重要的布尔值。我们将一些值设置为 `true` 以产生更好的显示效果。单击并拖曳选取所有的 `createReminder()` 调用，接着按 `Ctrl+Alt+M` | `Cmd+Alt+M` 快捷键，打开 `Extract Method` 对话框，如图 6-1 所示。这是诸多可用的重构操作之一，可以使用 `Refactor` 菜单和快捷键组合来调用。输入 `insertSomeReminders` 作为新方法的名字并按 `Enter` 键。`RemindersActivity` 中的代码将会被一个新的方法调用代替，该方法是在 `Extract Method` 对话框中指定的，而之前的代码将会被移至这个方法体内。

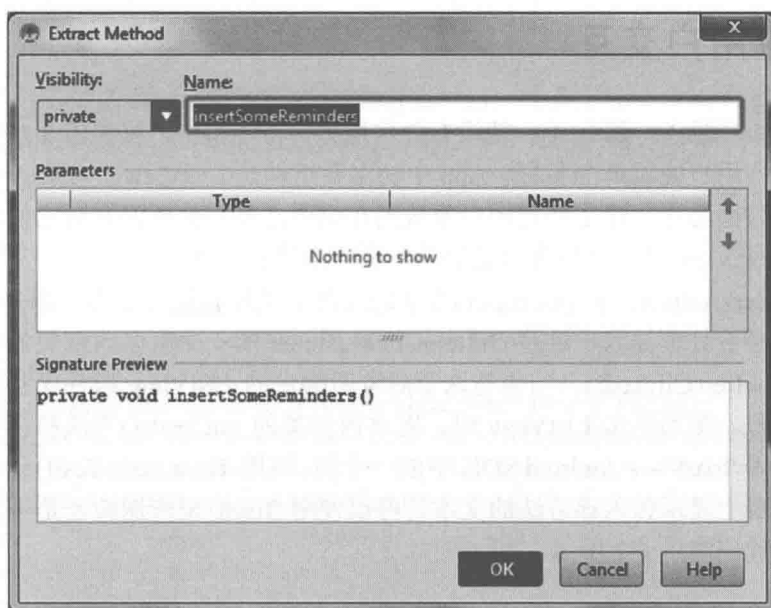


图 6-1 Extract Method 对话框，创建 insertSomeReminders()方法

运行 App，观察有了示例备忘之后的界面表现和行为。你的 App 看上去应该类似于图 6-2 中的屏幕截图。部分备忘将会显示绿色行标签，而那些被标记为重要的将会显示橙色标签。提交你的修改，附带消息为 Adds Example reminders。

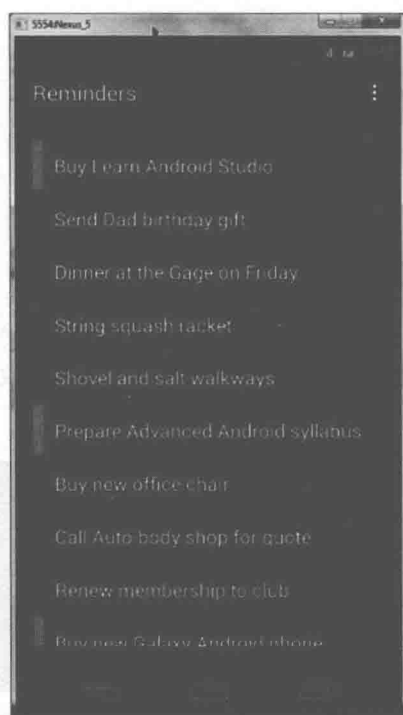


图 6-2 插入了示例备忘之后的运行时状态

6.2 响应用户交互

如果不能响应输入，那么 App 就没有任何用处。在本节中，将要添加响应触摸事件的逻辑，进而允许用户编辑单个备忘。App 中的主要组件是 `ListView`，它是 `Android View` 对象的子类。到目前为止，除了将视图对象放到布局中之外，你还没有对它们做过更多的操作。`android.view.View` 对象是所有可绘制到屏幕上的组件的超类。

在 `RemindersActivity` 中 `onCreate()` 方法的底部、关闭花括号之前，添加代码清单 6-2 中的代码，并接着处理导入。这是 `OnItemClickListener` 的一个匿名内部类的实现，它只有一个方法——`onItemClicked()`。当你与这个对象所绑定的 `ListView` 组件交互时，运行时将会使用这个对象。每当单击 `ListView` 时，匿名内部类的 `onCreate()` 方法都会被调用。我们定义的方法使用 `Toast`——Android SDK 中的一个类。调用 `Toast.makeText()` 会产生一个小弹框，它会在屏幕上显示传入该方法的文本。可以使用 `Toast` 来快速验证是否正确地调用了某个方法，如代码清单 6-2 所示。

注意

`Toast` 消息在某些设备上可能会被隐藏。另外一种替代方案是使用 `Android logger` 输入日志消息，这会在第 12 章中详细介绍。

代码清单 6-2 使用 `Toast` 设置 `OnItemClickListener`

```
//when we click an individual item in the listview
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
        long id) {
        Toast.makeText(RemindersActivity.this, "clicked " + position,
            Toast.LENGTH_SHORT).show();
    }
});
```

单击列表中的第一个条目会调用 `onItemClick()` 方法，传入的位置值为 0，因为列表中的元素从零开始编号。接下来的逻辑会弹出一个包含被单击文本和位置的 `Toast` 提示，如图 6-3 所示。

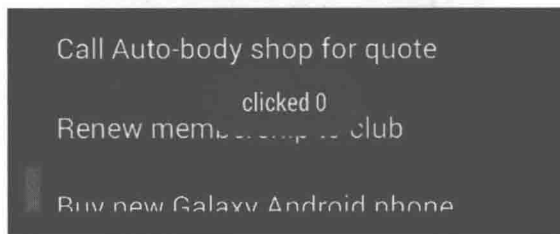


图 6-3 单击第一条备忘后显示的 `Toast` 消息

使用对话框

熟悉了单击事件之后，现在可以扩展单击监听器，令其显示一个对话框。使用代码清单 6-3 中的代码替换整个 `onItemClick()` 方法。在处理导入的时候，请使用 `android.support.v7.app.AlertDialog` 类。

代码清单 6-3 修改 `onItemClick()`，允许编辑/删除

```
public void onItemClick(AdapterView<?> parent, View view, final int
masterListPosition, long id) {
    AlertDialog.Builder builder = new
        AlertDialog.Builder(RemindersActivity.this);
    ListView modeListView = new ListView(RemindersActivity.this);
    String[] modes = new String[] { "Edit Reminder", "Delete Reminder" };
    ArrayAdapter<String> modeAdapter = new
        ArrayAdapter<>(RemindersActivity.this,
            android.R.layout.simple_list_item_1, android.R.id.text1, modes);
    modeListView.setAdapter(modeAdapter);
    builder.setView(modeListView);
    final Dialog dialog = builder.create();
    dialog.show();
    modeListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int
            position, long id) {
            //edit reminder
            if (position == 0) {
                Toast.makeText(RemindersActivity.this, "edit "
                    + masterListPosition, Toast.LENGTH_SHORT).show();
            }
            //delete reminder
            else {
                Toast.makeText(RemindersActivity.this, "delete "
                    + masterListPosition, Toast.LENGTH_SHORT).show();
            }
            dialog.dismiss();
        }
    });
}
```

你在上面的代码中看到了另外一个 Android 类——`AlertDialog.Builder` 的作用。`Builder` 是 `AlertDialog` 类中的内嵌静态类，用于构建 `AlertDialog`。

到目前为止，此实验中的代码创建了一个 `ListView` 以及用于向 `ListView` 填充列表项的 `ArrayAdapter`。可以回忆一下第 5 章中的这种模式。适配器的创建需要一个数组，其中包含两个核心选项——`Edit Reminder` 和 `Delete Reminder`；在将其传递给 `ListView` 之前，首先需要传递给 `AlertDialog.Builder`。构建器接着创建并显示带有选项列表的对话框。

注意代码清单 6-3 中的最后一段代码。它类似于之前添加的 `OnItemClickListener()` 代码；不过，我们是在当前 `OnItemClickListener` 的内部为新创建的 `modeListView` 绑定单击监听器。

你看到的情形是在 `ListView` 的 `OnItemClickListener` 内部创建了另一个 `modeListView` 和内嵌 `OnItemClickListener`, 而该内嵌 `OnItemClickListener` 用于响应 `modeListView` 的单击事件。

内嵌的单击监听器弹出一条标识编辑或删除条目被单击的 `Toast` 消息。它还将外层 `OnItemClickListener` 的位置参数重命名成了 `masterListPosition`, 以便与内嵌 `OnItemClickListener` 的位置参数区分开来。这个主位置用于在 `Toast` 中标识将要编辑或删除的是哪条备忘。最后, 从单击监听器调用 `dialog.dismiss()` 方法, 它会将对话框彻底删除。

通过在设备或模拟器上运行它来测试图 6-4 中所示的新特性。单击一条备忘, 接着从新的弹出对话框中单击 `Edit Reminder` 或 `Delete Reminder`。如果 `Toast` 中报告的备忘位置与你单击的备忘不匹配, 那么再次检查追加到 `Toast` 文本中的是 `masterListPosition` 值而非使用了 `position`。按 `Ctrl+K` | `Cmd+K` 快捷键并提交你的修改, 附带消息为 `Adds a ListView dialog for individual list items`。

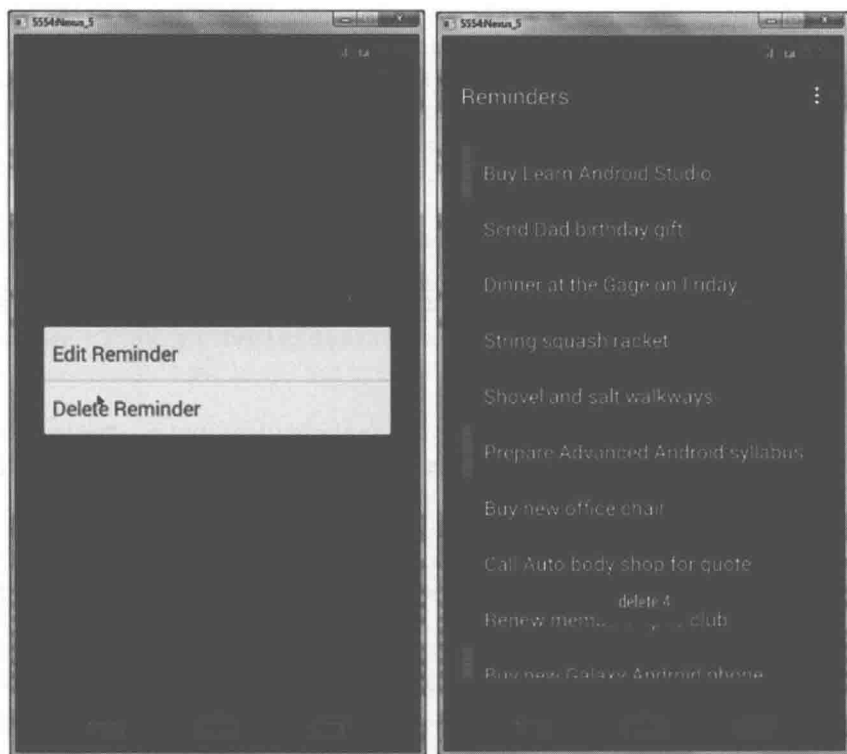


图 6-4 模拟删除一条备忘

6.3 提供多选上下文菜单

随着 App 逐渐成形, 你现在将要添加一个特性, 允许在一次操作中编辑多条备忘。这个特性仅在运行着 API 11 或更高版本的设备上可用。你将要使用资源加载约定让 App 中的这个特性条件可用。这个过程会在本章的后面进行解释, 而第 8 章中有详细介绍。你还需要引入运行时的检查以确定是否启用该特性。

首先，为备忘行条目创建另一个布局。打开 Project 工具窗口并右击 res 文件夹，打开上下文菜单。从菜单中选择 New Android Resource File 并在对话框中输入 reminders_row 作为文件名，如图 6-5 所示。

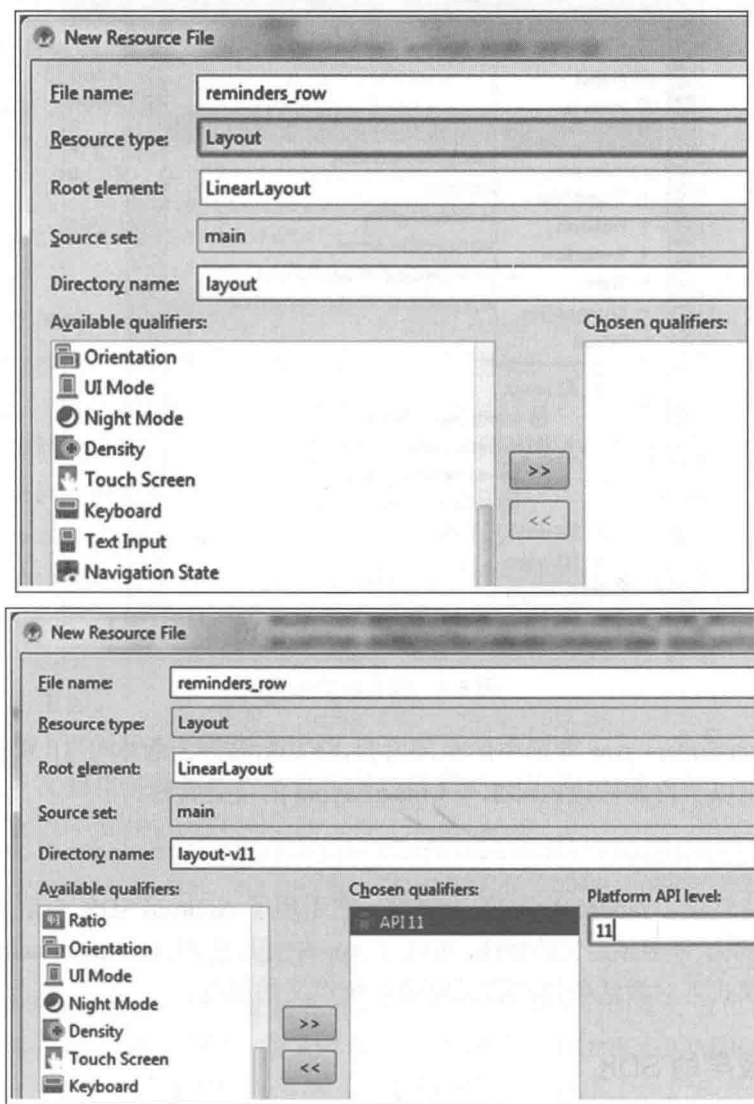


图 6-5 reminders_row 的新资源文件

选择 Layout 作为 Resource Type，该操作会自动将目录名修改为 layout。在 Available qualifiers 下方选择 Version，接着单击双箭头按钮(>>)，将 Version 添加到已选限定符列表中。在 Platform API level 中输入 11，注意目录名已经更新，体现出了已选限定符。这些均称为资源限定符，它们会在运行时用到，允许为特定设备和平台版本自定义用户界面。按下 Enter 键(或者单击 OK 按钮)，接受这个带有资源限定符的新目录并继续。如果打开 Project 工具窗口，并将其视图设置为 Android，如图 6-6 所示，那么你将会看到两个 reminders_row 布局文件都位于 layout 文件夹中。再次说明，项目窗口的 Android 视图会将相关联的文件组织在一起，以便你能够高效地管理它们。

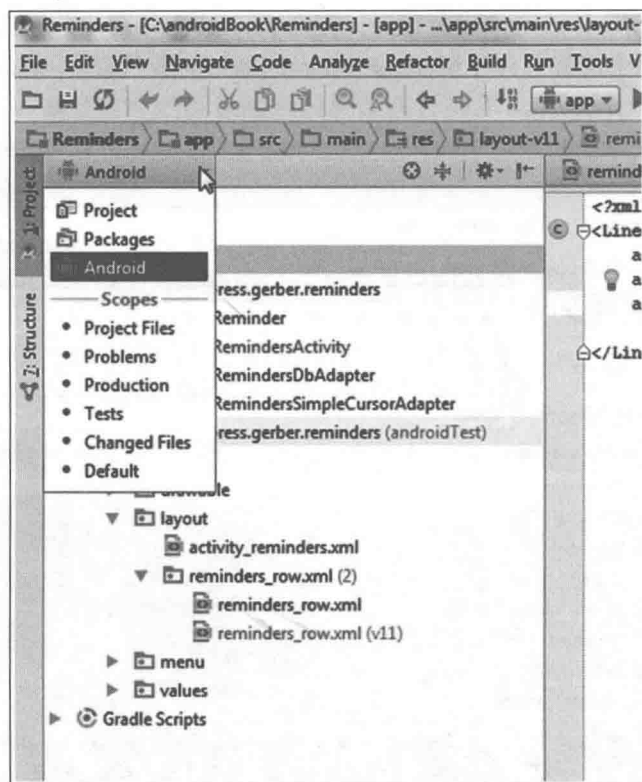


图 6-6 组合后的布局

复制之前 `reminders_row` 布局中的全部内容, 将其粘贴到为版本号 11 新创建的布局中。现在, 通过使用以下内容修改内部水平 `LinearLayout` 的背景特性:

```
android:background="?android:attr/activatedBackgroundIndicator"
```

这个背景特性值以 `?android:attr/` 作为前缀, 它引用了 Android SDK 中定义的一个样式。Android SDK 提供了许多预定义的特性, 可以在 App 中使用它们。`activatedBackgroundIndicator` 特性会在多选模式下对激活条目的背景应用系统定义的颜色。

6.3.1 兼容较早的 SDK

现在你将要学习如何引入有平台依赖的特性。打开 Project 工具窗口并打开 Gradle Scripts(它是第二项)下面 app 模块的 `build.gradle` 文件。Gradle 文件保存了用于编译和打包 App 的构建逻辑。关于 App 支持哪些平台的所有配置均位于这些特殊文件中(第 13 章会深入探讨 Gradle 构建系统)。注意, 将 `minSdkVersion` 设置为 8 可以让你的 App 能够在超过 99% 的 Android 设备上运行。我们将要创建的这个特性需要的最低 SDK(也称为 API)版本为 11。我们在本节中讲解的代码和特性将允许运行 SDK 版本 11 或更高版本的用户体验一种名为上下文操作模式的特性。此外, 那些运行 SDK 版本低于 11 的用户将不会看到此特性, 但更重要的是, 他们的 App 并不会崩溃。

6.3.2 添加上下文操作模式

接下来的这个特性在多选模式下引入了上下文操作菜单，它是一个操作列表，可以应用于所有选中条目的上下文。通过右击 `res/menu` 目录添加新的菜单资源，选择 `New | Menu resource` 并将其命名为 `cam_menu`。使用以下代码来装饰它：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_delete_reminder"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="delete" />
</menu>
```

这个资源文件为上下文菜单定义了一个单独的 `delete` 动作。你在这里使用略微不同的特性值。这些特殊值类似于之前在背景特性中所使用的那些(让你能够访问内置的 Android 默认值)。不过，`?android:attr/`前缀只能在引用样式特性时使用。在这里，这些特性所采用的语法遵从一种稍有不同的形式。使用 `at` 符号(`@`)可以触发对资源值名称空间的查找。可以使用此方法访问各种名称空间。所有内置的 Android 值均位于 `android` 名称空间中。各种资源，例如 `drawable`、`string` 和 `layout`，均位于此名称空间中。当使用特殊的 `@+id` 前缀时，它会在项目的 `R.java` 文件中创建一个新的 ID；而当使用 `@id` 前缀时，它会在 Android SDK 的 `R.java` 文件中查找已有 ID。这个示例定义了一个新的 ID 名称——`menu_item_delete_reminder`，它与菜单选项相关联。它还引用了 `android:drawable` 名称空间中的一个图标，并将其作为自己的图标。

有了新的上下文菜单和针对运行 API 11 或更高版本设备的替代布局之后，可以添加复选框来配合上下文操作菜单有条件地开启多选模式。打开 `RemindersActivity` 并在 `onCreate` 方法的末尾添加以下 if 块：

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
}
```

`Build` 类位于 `android.os` 包中并且能够访问一系列常量值，而这些值可以用于匹配特定 API 级别的设备。在本例中，你希望 API 级别等于或高于 `HONEYCOMB`(对应整数值 11)。在你刚刚定义的 if 块里面插入代码清单 6-4 中的代码。if 块用于保护那些运行的 OS 低于 `Honeycomb` 的设备，如果没有它的话，App 将会崩溃。

代码清单 6-4 MultiChoiceModeListener 示例

```
mListView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
mListView.setMultiChoiceModeListener(new
AbsListView.MultiChoiceModeListener() {
    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position,
long id, Boolean checked) { }

    @Override
```

```

public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.cam_menu, menu);
    return true;
}

@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    return false;
}

@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    switch(item.getItemId()) {
        case R.id.menu_item_delete_reminder:
            for (int nC = mCursorAdapter.getCount() - 1; nC >= 0; nC--) {
                if (mListView.isItemChecked(nC)) {
                    mDbAdapter.deleteReminderById(getIdFromPosition(nC));
                }
            }
            mode.finish();
            mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
            return true;
        }
        return false;
    }

@Override
public void onDestroyActionMode(ActionMode mode) { }
});

```

导入所有必需的类。你将会注意到 `getIdFromPositon()` 没有定义而且被标记为红色。将光标置于方法上，按 **Alt+Enter** 快捷键调用 **IntelliSense** 并选择 **Create Method**。选择 **RemindersActivity** 作为目标类。选择 **int** 作为返回值。按代码清单 6-5 所示装饰该方法。

代码清单 6-5 `getIdFromPosition()` 方法

```

private int getIdFromPosition(int nC) {
    return (int)mCursorAdapter.getItemId(nC);
}

```

上述逻辑定义了 `MultiChoiceModeListener` 并将其绑定到 `ListView`。当长按 `ListView` 中的条目时，运行时调用 `MultiChoiceModeListener` 的 `onCreateActionMode()` 方法。如果该方法返回布尔值 `true`，则会进入多选模式。当处于此模式时，重载方法中的逻辑会填充一个显示在操作栏中的上下文菜单。使用多选操作模式的好处是可以选择多行。单击一次会选中项，再次单击则会取消选中。当单击上下文菜单中的各个条目时，运行时将会调用被单击菜单项的 `onActionItemClicked()` 方法。

在此方法中，通过比较 `itemId` 和添加到菜单项中的删除元素的 `id` 可以验证是否单击了

删除选项(关于删除元素的 id 的描述请参见本节开始处的 XML 代码清单)。如果选中了该项,那么循环遍历每个列表项并请求 `mdbAdapter` 删除它们。在删除了选中项之后,逻辑会调用 `ActionMode` 对象的 `finish()` 方法,这将会禁用多选操作模式并返回到 `ListView` 的正常状态。接下来,调用 `fetchAllReminders()` 从数据库中重新加载所有备忘,并将该调用返回的游标传递给 `mCursorAdapter` 对象的 `changeCursor` 方法。最后,方法返回 `true`,表示已经妥善地处理了该操作。对于逻辑没有处理的所有其他分支,方法返回 `false`,表示其他事件监听器可以处理该单击事件。

`Android Studio` 会将一些语句突出显示为错误,因为正在使用比 `Honeycomb` 更老的平台不可用的 API。这个错误是由 `Lint` 生成的,它是 `Android SDK` 中内置的静态分析工具,而且已经完全集成到了 `Android Studio` 中。你需要在 `@Override` 注解的上面或下面向 `RemindersActivity.onCreate()` 方法添加以下注解并导入 `TargetApi`:

```
@TargetApi (Build.VERSION_CODES.
HONEYCOMB)
```

这个特殊的注解告诉 `Lint` 方法调用兼容这里提供的 API 级别即可,无需考虑构建配置中指定的值。将你的修改提交至 `Git`,附带信息为 `Adds Contextual Action Mode with context action menu`。图 6-7 展示了当你构建并运行 App 来测试新特性时可能会看到的情形。

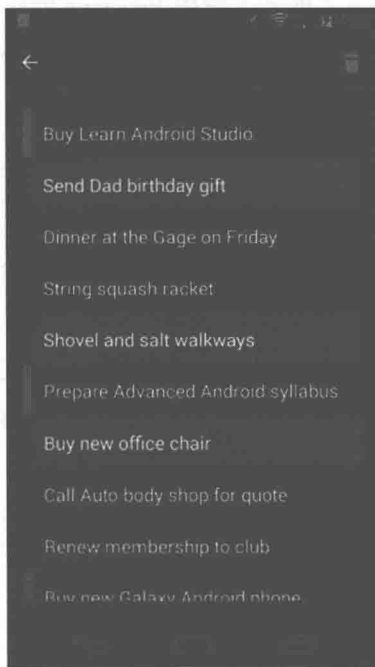


图 6-7 已启用的多选模式

6.4 实现添加、编辑和删除

到目前为止,你已经添加了从列表中删除备忘的逻辑。此逻辑仅在上下文操作模式中可用。目前,你还没有任何办法来插入新备忘或修改已有备忘。不过,你现在将要创建一个用来添加备忘的自定义对话框,以及一个编辑已有备忘的对话框。后面你将要把这些对话框绑定到 `RemindersDbAdapter`。

在继续之前,你需要定义一些额外的颜色。向 `colors.xml` 文件添加如下颜色定义:

```
<color name="light_grey">#bababa</color>
<color name="black">#000000</color>
<color name="blue">#ff1118ff</color>
```

注意

通常,App 应该有一个整体的颜色主题,以确保所有界面和对话框之间的一致性。然而,颜色主题超出了这个简单实验的范畴。

6.4.1 设计自定义对话框

一种好的开发习惯是在实现之前使用简单工具描绘出你的 UI。这样做能够让你在编写代码之前看到元素在屏幕上的样子。可以使用编辑器，例如 Inkscape(它是跨平台的)，或者使用一些更简单的东西——纸和铅笔。在移动开发中，这些素描图被称为线框图。

图 6-8 展示了使用 Inkscape 制作的自定义对话框。线框图本来就是非正式的，它主要强调组件的摆放，而非具体的界面外观。

注意

本书中的部分自定义插图和线框图是使用 Inkscape 创建的，它是一款支持多平台的矢量图形编辑器。可以从 www.inkscape.org 免费下载它。

有了线框图，就可以规划如何在屏幕上摆放组件了。由于大多数组件均是从上到下排列，因此使用竖直 `LinearLayout` 作为最外层容器是一个直观的选择。不过，底部的两个按钮是左右相邻的。对于这种情况，可以使用水平 `LinearLayout` 并将其嵌入到外层的竖直 `LinearLayout` 中。图 6-9 为草图添加了注释并突出显示了这个嵌套组件。



图 6-8 自定义对话框的线框图

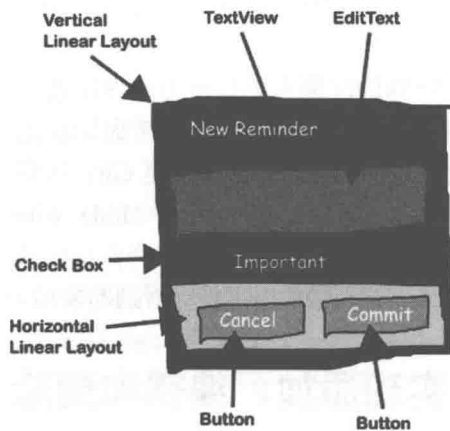


图 6-9 带有组件标签的线框图

6.4.2 将设计转换为代码

有了这些线框图之后，尝试使用“可视化设计器”来设计布局。首先，在 `Project` 工具窗口中右击 `res` 目录并选择 `New Android Resource File` 选项，将资源文件命名为 `dialog_custom`，接着将 `Layout` 选为 `Resource` 类型。使用 `LinearLayout` 作为根元素，完成该对话框。为了重建我们的线框图，从调板中拖曳并将各种 `View` 放入编辑区域。代码清单 6-6 中包含了已完成的布局 XML 定义，包括将要在 Java 代码中使用的 ID 值。

代码清单 6-6 已完成的 dialog_custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/custom_root_layout"
    android:layout_width="300dp"
    android:layout_height="fill_parent"
    android:background="@color/green"
    android:orientation="vertical"
    >

    <TextView
        android:id="@+id/custom_title"
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        android:gravity="center_vertical"
        android:padding="10dp"
        android:text="New Reminder:"
        android:textColor="@color/white"
        android:textSize="24sp" />

    <EditText
        android:id="@+id/custom_edit_reminder"
        android:layout_width="fill_parent"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:background="@color/light_grey"
        android:gravity="start"
        android:textColor="@color/black">
        <requestFocus />
    </EditText>

    <CheckBox
        android:id="@+id/custom_check_box"
        android:layout_width="fill_parent"
        android:layout_height="30dp"
        android:layout_margin="4dp"
        android:background="@color/black"
        android:paddingLeft="32dp"
        android:text="Important"
        android:textColor="@color/white" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal">
```

```

<Button
    android:id="@+id/custom_button_cancel"
    android:layout_width="0dp"
    android:layout_height="60dp"
    android:layout_weight="50"
    android:text="Cancel"
    android:textColor="@color/white"
/>

<Button
    android:id="@+id/custom_button_commit"
    android:layout_width="0dp"
    android:layout_height="60dp"
    android:layout_weight="50"
    android:text="Commit"
    android:textColor="@color/white"
/>

</LinearLayout>

</LinearLayout>

```

6.4.3 创建自定义对话框

现在要在 `RemindersActivity` 中使用这个已经完成的对话框。代码清单 6-7 是新方法 `fireCustomDialog()` 的实现。将此代码加入到 `RemindersActivity.java` 文件中，置于 `onCreateOptionsMenu()` 方法之上，并处理导入。

代码清单 6-7 `fireCustomDialog()` 方法

```

private void fireCustomDialog(final Reminder reminder){
    // custom dialog
    final Dialog dialog = new Dialog(this);
    dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
    dialog setContentView(R.layout.dialog_custom);

    TextView titleView = (TextView) dialog.findViewById(R.id.custom_title);
    final EditText editCustom = (EditText) dialog.
        findViewById(R.id.custom_edit_reminder);
    Button commitButton = (Button) dialog.findViewById(R.id.custom_button_commit);
    final CheckBox checkBox = (CheckBox) dialog.findViewById(R.id.custom_check_box);
    LinearLayout rootLayout = (LinearLayout) dialog.
        findViewById(R.id.custom_root_layout);
    final boolean isEditOperation = (reminder != null);

```

```

//this is for an edit
if (isEditOperation){
    titleView.setText("Edit Reminder");
    checkBox.setChecked(reminder.getImportant() == 1);
    editCustom.setText(reminder.getContent());
    rootLayout.setBackgroundColor(getResources().getColor(R.color.blue));
}

commitButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String reminderText = editCustom.getText().toString();
        if (isEditOperation) {
            Reminder reminderEdited = new Reminder(reminder.getId(),
                reminderText, checkBox.isChecked() ? 1 : 0);
            mDbAdapter.updateReminder(reminderEdited);
            //this is for new reminder
        } else {
            mDbAdapter.createReminder(reminderText, checkBox.isChecked());
        }
        mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
        dialog.dismiss();
    }
});

Button buttonCancel = (Button) dialog.findViewById(R.id.custom_button_cancel);
buttonCancel.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        dialog.dismiss();
    }
});

dialog.show();
}

```

`fireCustomDialog()`方法将用于插入和编辑，因为这两个操作之间几乎没有差别。此方法的前三行创建一个没有标题的 Android 对话框，并使用代码清单 6-6 中的布局进行填充。`fireCustomDialog()`方法接着找出此布局中的所有重要元素并将它们保存在局部变量中。接着，该方法通过检查 `reminder` 参数是否为空来设置布尔变量 `isEditOperation` 的值。如果有备忘传入(或者说如果值不为空)，那么该方法认为这是编辑操作并将该变量设置为 `true`；否则它被设置为 `false`。如果对 `fireCustomDialog()`的调用是编辑操作，那么标题会被设置为 `Editor Reminder`，同时还会根据备忘参数的值设置 `CheckBox` 和 `EditText` 中的内容。这个方法还会把最外层容器的布局背景设置为蓝色，目的是可视化地区分编辑对话框和插入对话框。

接下来的几行构成了一个代码块，设置并定义了用于 `Commit` 按钮的 `OnClickListener`。此监听器会响应 `Commit` 按钮的单击事件并更新数据库。同样，`isEditOperation()`会被检查，

如果正处于编辑模式，则使用备忘参数的 ID、EditText 中的值和复选框的界面选中状态创建一条新的备忘。我们通过使用 `updateReminder()` 方法将备忘录传入 `mDbAdapter`。

如果没有处于编辑状态，那么逻辑会要求 `mDbAdapter` 使用 EditText 中的值和复选框的界面选中状态在数据库中创建一条新的备忘。在调用更新或创建方法之后，都会使用 `mCursorAdapter.changeCursor()` 方法重新加载备忘。这个逻辑类似于你之前在代码清单 6-5 中添加的那个。重新加载备忘之后，单击监听器会关闭对话框。

配置了 Commit 按钮的单击行为之后，本例为 Cancel 按钮设置了另一个单击监听器。这个监听器只负责让对话框消失。在指定这两个按钮的行为之后，本例终于可以显示自定义对话框了。

现在可以在 `OnItemClickListener` 中对 `modeListView` (在 `onCreate()` 方法中) 使用这个新方法。找到这个监听器的 `onItemClick()` 方法并使用以下代码替换整个方法：

```
public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
    //edit reminder
    if (position == 0) {
        int nId = getIdFromPosition(masterListPosition);
        Reminder reminder = mDbAdapter.fetchReminderById(nId);
        fireCustomDialog(reminder);
        //delete reminder
    } else {
        mDbAdapter.deleteReminderById(getIdFromPosition(masterListPosition));
        mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
    }
    dialog.dismiss();
}
```

要编辑备忘，需要将 `Toast.makeText()` 替换为根据 `ListView` 的位置查找备忘。接着，将这条备忘传递给 `fireCustomDialog()` 方法并触发编辑行为。要删除备忘，需要在多选模式下使用代码清单 6-5 中的那些逻辑。同样，`mDbAdapter.deleteReminderById()` 用于删除备忘，而 `changeCursor()` 用于处理从 `mDbAdapter.fetchAllReminders()` 调用返回的游标。

在 `RemindersActivity.java` 文件的最底部找到 `onOptionsItemSelected()` 方法，按照代码清单 6-8 所示的代码修改它。

代码清单 6-8 `onOptionsItemSelected` 的定义

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.action_new:
            //create new Reminder
            fireCustomDialog(null);
            return true;
        case R.id.action_exit:
            finish();
            return true;
        default:
            return false;
    }
}
```

```

    }
}

```

当所选的菜单项是 `action_new` 时，你只需要添加一个对 `fireCustomDialog()` 的调用。向该方法传递 `null`，之前所述的逻辑会检测到空值并将 `isEditOperation` 设置为 `false`，从而调用 `New Reminder` 对话框。运行 App 并测试这个新特性，你应该能够看到新的自定义对话框。当创建新备忘时，你将会看到绿色的对话框；而当编辑备忘时，则是蓝色的对话框，分别如图 6-10 和图 6-11 所示。测试菜单项以确保创建和删除操作能够按照预期的方式完成。向 Git 提交修改，附带提交消息为 `Adds database Create, Read, Update, and Delete support with custom dialogs`。

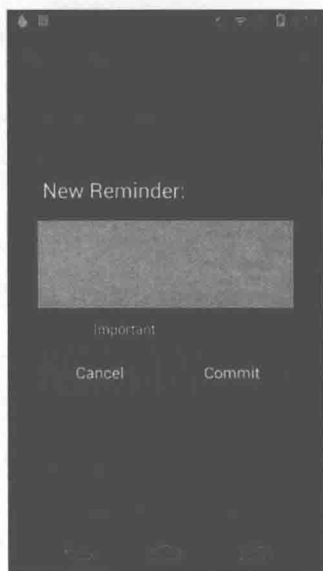


图 6-10 New Reminder 对话框

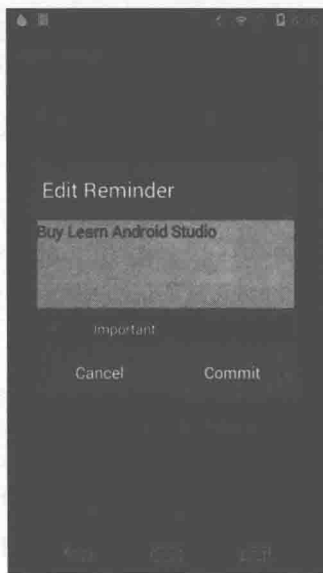


图 6-11 Editor Reminder 对话框

6.4.4 添加自定义图标

有了所有这些特性，可以添加一个自定义图标作为结尾。可以使用任意图像编辑器创建一个图标，或者如果你对作图不感兴趣，就从网上找些免费的剪贴画。我们的示例使用在 `Inkscape` 中创建的自定义图片替换 `ic_launcher` 图标。打开 `Project` 工具窗口并右击 `res/mipmap` 目录，选择 `New | Image Asset`。你将会看到类似于图 6-12 所示的对话框。单击位于 `Image file` 文本框最右侧的省略号按钮并导航至你已经准备好的图像资源的位置。其余的设置保留图 6-13 中所示的值。现在单击 `Next` 按钮，并在后续对话框中单击 `Finish` 按钮。

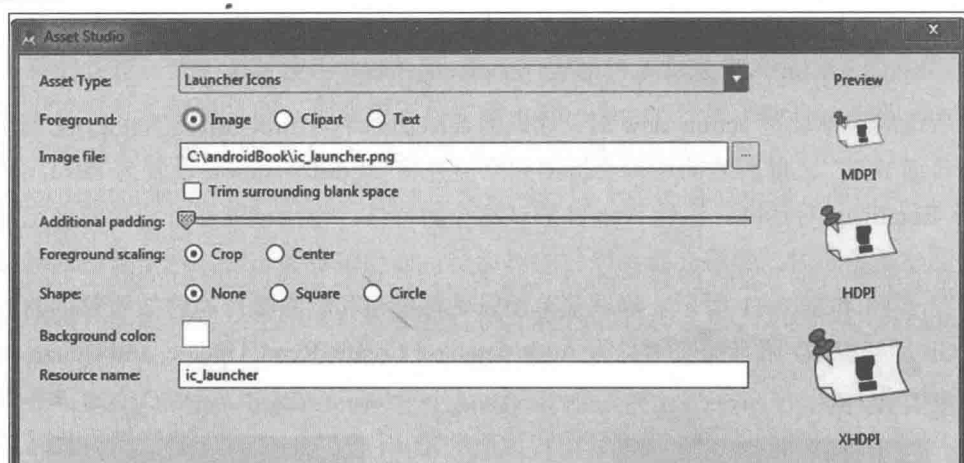


图 6-12 New Image Asset 对话框

有大量名为 `mipmap` 的文件夹。这些文件夹均以对应的屏幕大小限定符作为前缀。Android 运行时将会在特定文件夹下提取资源，而这取决于运行 App 设备的屏幕分辨率。第 8 章会更具体地介绍资源文件及其前缀。

向 `RemindersActivity` 的 `onCreate()` 方法中插入以下代码行，并将其置于填充布局的代码行 `setContentView(R.layout.activity_reminders);` 之后。此代码在你的 Action Bar 中显示一个自定义图标：

```
ActionBar actionBar = getSupportActionBar();
actionBar.setHomeButtonEnabled(true);
actionBar.setDisplayHomeAsUpEnabled(true);
actionBar.setIcon(R.mipmap.ic_launcher);
```

当运行代码时，会看到 Action Bar 中的自定义图标。图 6-13 展示了一个带有自定义图标的 App 运行示例。

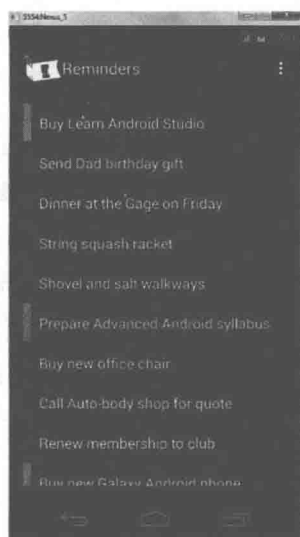


图 6-13 操作栏中的自定义图标

按 Ctrl+K | Cmd+K 快捷键并提交你的修改，附带消息为 Adds a customer icon。

6.5 小结

祝贺你！你已经使用 Android Studio 实现了第一个 Android App。在此过程中，你学会了如何使用 Visual Designer 来编辑 XML 布局。你还学会了如何使用 Text 模式编辑原始 XML。本章向你展示了如何在支持“上下文操作模式”的平台上有条件地实现该特性。最后，你看到了如何为各种屏幕分辨率添加自定义图标。

第 7 章

Git 入门

Git 版本控制系统(VCS)正在迅速地成为事实标准，不仅在 Android 应用开发中，甚至对于软件编程整体来说都是如此。与需要使用中心服务器的早期版本控制系统不同，Git 是分布式的，这意味着仓库的每一个副本均包含项目的完整历史，而且所有贡献者都没有特权。Git 由大名鼎鼎的 Linus Torvalds 开发，目的是管理 Linux 操作系统的开发。与开源运动自身类似，Git 从系统角度来讲没有层级而且鼓励协作。

虽然 Git 提供了丰富的命令行特性，但本章主要关注在 Android Studio 中使用 Git。基于 IntelliJ 平台的 Android Studio 对于近年来的多种 VCS 系统提供了良好的支持，其中也包含 Git。我们会采用深入浅出的方式为新手和专家介绍这些不同系统之间的共性。然而，理解在 Android Studio 中使用 Git 和在命令行中使用 Git 的差别很重要。本章详细介绍入手 Git 所需的全部内容。你将复用之前章节中创建的 Reminders App 来学习关于提交、分支、推送、获取以及其他一些重要命令的基础知识。你将要同时使用本地和远程 Git 仓库并查看如何在协同环境中使用 Git 和 Android Studio。

打开你在第 1 章中创建的 HelloWorld 项目。如果跳过了那一章，那么需要创建一个名为 HelloWorld 的新项目。在通过向导的过程中全部使用默认设置。你将利用这个项目简要地了解 Git 安装的基础知识。

7.1 安装 Git

在可以使用 Git 之前，你需要先安装它。在浏览器中打开 <http://git-scm.com/downloads>。单击适合自己操作系统的 Download 按钮，如图 7-1 所示。

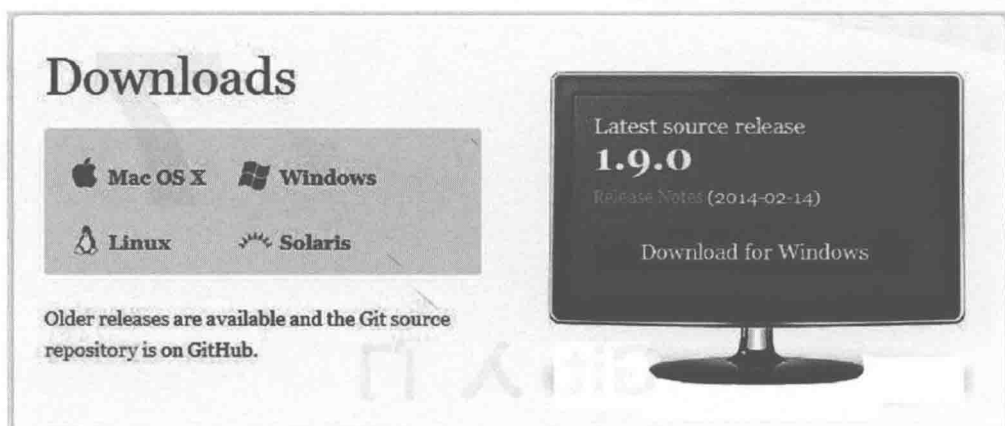


图 7-1 Git 下载界面

我们建议将 Git 安装在 Windows 上的 C:\java\ 目录、Mac 或 Linux 上的 ~/java 目录中。无论决定安装在哪里，确保整条路径下有足够的空间。例如，不要将 Git 安装在 C:\Program Files 目录中，因为 Program 和 Files 之间有一个空格。类似 Git 这种面向命令行的工具在处理名称中含有空格的目录时通常会遇到问题。安装完以后，需要确保 PATH 环境变量中包含 C:\java\git\bin\ 目录。关于如何将路径添加到 PATH 环境变量的详细指南请参见第 1 章。

单击 Git Bash 图标，启动 Git Bash 终端。如果正在运行 Mac 或 Linux，那就打开终端。你需要使用自己的名字和电子邮箱配置 Git 以便你的提交有相应的作者。在 Git Bash 中，输入以下命令并将 John Doe 的名字和电子邮件地址替换成你自己的。图 7-2 展示了一个例子。

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

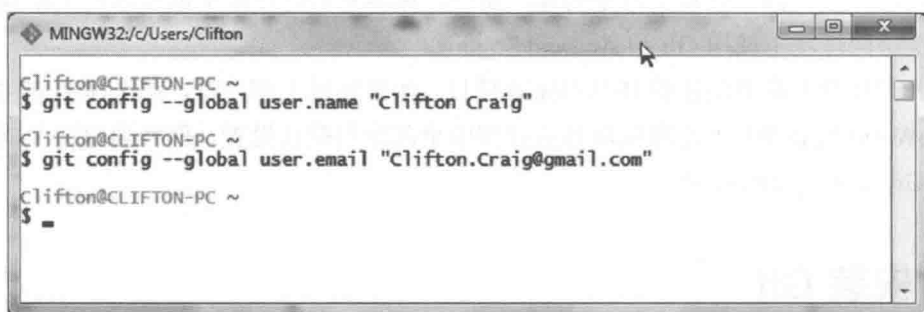


图 7-2 向 Git 添加你的名字和电子邮件

返回 Android Studio，继续设置 Git 与 Android Studio 的集成。导航至 File | Settings，接着在左侧面板中 Version Control 区域的下方找到 Git。单击省略号按钮并浏览到刚刚安装的 Git 二进制文件。单击 Test 按钮，确保 Git 环境是可以正常使用的。你应该会看到表示 Git 执行成功的弹出框，以及所安装 Git 的版本。

导航至 VCS | Import into Version Control | Create Git Repository。当对话框提示选择用

于创建新 Git 仓库的目录时，务必选择项目根目录 HelloWorld。也可以在目录选择对话框中单击 Android Studio 小图标。此图标将导航至项目的根目录，如图 7-3 所示。单击 OK 按钮，即创建了你的本地 Git 仓库。

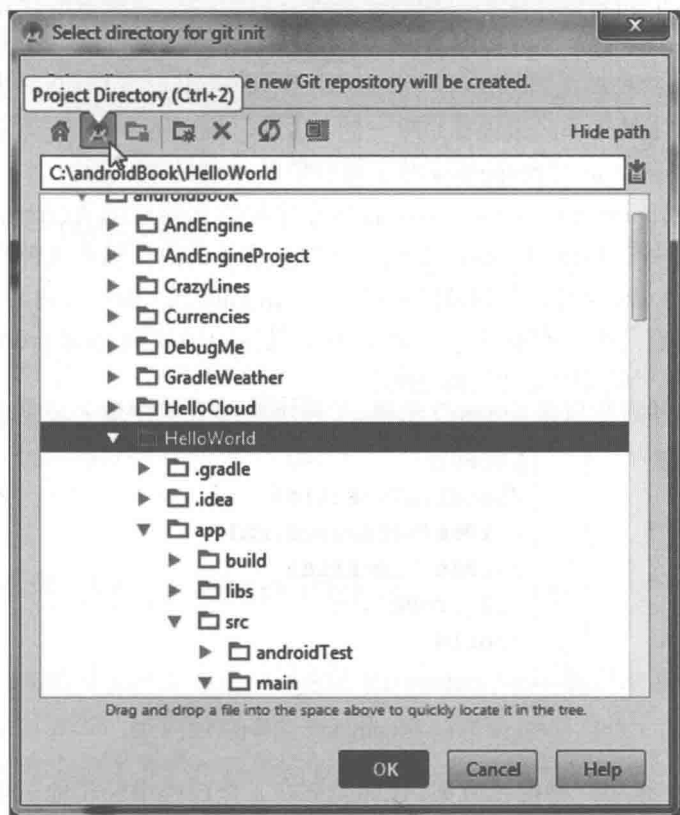


图 7-3 为 Git 仓库选择目录

你将会注意到 Project 工具窗口中的大多数文件名都变成了棕色。这意味着 Git 在本地识别了这些文件，但是尚未跟踪，而且并没有计划添加。Git 采用一种两阶段的方式来管理提交(这有别于其他 VCS 工具，例如 Subversion 和 Perforce 的实现方式)。编辑区域是 Git 组织提交前修改的地方。正在进行的修改、编辑区域的修改和已提交的修改之间的差别很重要，可能会让新用户产生混淆。其结果是 Android Studio 并没有暴露这些不同。相反，你只会看到一个简单的修改界面，使得你能够轻松地管理已修改文件并提交它们。

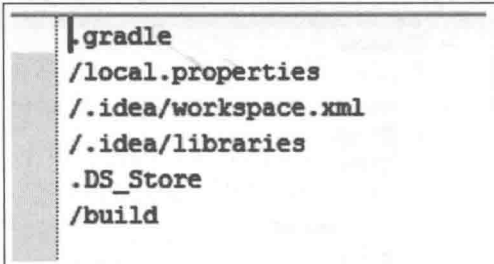
7.2 忽略文件

当创建本地仓库时，Android Studio 会生成一个特殊的.gitignore 文件，用于禁止跟踪某些路径。除非特别指定，否则 Git 将继续跟踪此目录及其子目录中的所有文件。不过，.gitignore 文件可以让 Git 忽略某些文件或整个目录。

通常，根目录中会有.gitignore 文件，而每个项目中也有.gitignore 文件。在 HelloWorld

中，一个.gitignore 文件位于 HelloWorld 的根目录，还有一个.gitignore 文件位于 App 文件夹的根目录。打开位于 HelloWorld 根目录中的.gitignore 文件并观察其内容。图 7-4 展示了在项目根目录下生成的.gitignore 文件。默认情况下，Android Studio 会将某些文件排除在 Git 仓库之外。这个列表包含由项目构建生成的文件或者特定于本地计算机的控制设置。例如，/.idea/workspace.xml 文件控制 Android Studio 本地配置的设置。尽管可以在 Git 中跟踪它，但它并不是待构建项目的必要组成部分，而且在实际中可能会产生问题，因为这个文件对于每台电脑上的工作区来说都是唯一的。注意，.gitignore 中有一项是/local.properties。类似于 workspace.xml，local.properties 对于每台电脑来说是唯一的。

注意列表中的/build 条目。Gradle——第 13 章中将深入讨论的 Android Studio 构建系统，会在编译和运行项目的时候将它的所有输出产生在这里。由于这个文件夹将会包含各种东西，从.class 文件到.dex 文件，再到最终的可安装 Android 包，而且其内容还是不断变化的，因此使用 Git 跟踪它几乎没有意义。在 Project 工具窗口中找到 local.properties 文件，你会注意到它是黑色的，而其他文件是棕色的。



```

| gradle
| /local.properties
| /.idea/workspace.xml
| /.idea/libraries
| .DS_Store
| /build

```

图 7-4 根.gitignore 文件的内容

Android Studio 采用一种配色方案，让你能够在工作过程中轻松地识别出当前的版本控制状态。正如我们已经谈到的，棕色表示文件已经被 Git 本地识别，但是没有被 Git 跟踪，而且没有计划添加。蓝色表示文件正被 Git 跟踪而且已经被修改。绿色表示被 Git 跟踪的全新文件。黑色表示没有改动或者没有被跟踪的文件。Android Studio 会持续跟踪已加入到项目中的文件并提示你尽可能地保持这些文件与 Git 同步。

7.3 添加文件

打开位于屏幕底部的 Changes 视图。它包含两个区域：Default 和 Unversioned Files。Default 区域最初是空的，表示活动修改列表。当你修改和创建文件时，它们将出现在这个区域中，因为其中保存了已经准备好提交到 VCS 的文件。Unversioned Files 区域包含了没有被 VCS 跟踪的所有内容。

因为所有的项目文件都还没有被跟踪，所以它们都位于 Unversioned Files 区域。你需要将它们添加到仓库中。Changes 视图的左侧是两列图标。在右侧的列中，单击从上面数的第三个图标(一个文件夹图标)；参见图 7-5 中所示的圆形标记。这是一个开关，允许你使用文件夹分组文件，以便更好地了解它们在项目中的相对位置。右击 Unversioned Files

区域的顶部并单击上下文菜单中的 Add to VCS，将这些文件添加到 Git 索引。除此以外，可以单击并将整个区域拖曳至以粗体显示的 Default 区域中。

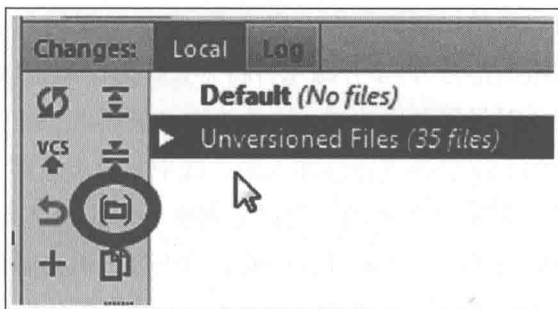


图 7-5 使用文件夹分组文件

在添加所有文件之后，单击带有绿色向上箭头的 VCS 图标。这会打开你所熟悉的 Commit 对话框，你在第 5 章中就开始使用它了。单击 Commit 来记录你的修改，同时 Default 部分将会被清空。你也可以按 Ctrl+K | Cmd+K 来执行相同的操作。从此刻起，在 Android Studio 中修改的每个文件都会被 Git 跟踪。

7.4 克隆参考 App: Reminders

这一节扩展了在第 5 章和第 6 章中创建的 Reminders App。我们邀请你使用 Git 克隆此项目，以便跟随学习进度，尽管需要为此项目重新创建新的 Git 仓库，该仓库从第 5 章和第 6 章中所使用的仓库分叉而来。如果你的电脑上还没有安装 Git，那么请阅读第 7 章。在 Windows 上打开 Git-bash 会话(或者 Mac 和 Linux 上的终端)并导航至 C:\androidBook\reference(如果没有 reference 目录，就创建它。在 Mac 上，导航至 ~/your-labs-parent-dir/reference/)，输入以下 git 命令：`git clone https://bitbucket.org/csgerber/reminders-git.git RemindersGit`。你将要使用 Git 的特性来修改项目，就如同在团队中工作一样。

在这个过程中，你将会学到如何分叉并克隆项目，以及在开发功能的同时建立并维护分支。在练习开始之前，请将你在第 6 章中完成的 Reminders 项目重命名为 RemindersChapter6，因为稍后你需要重新创建此文件夹。在 Windows 上，可以右击 Explorer 中的文件夹并选择重命名。在 Linux 或 Mac 上运行以下命令：`mv ~/androidBook/Reminders ~/androidBook/RemindersChapter6`。

7.4.1 分叉和克隆

分叉远程仓库是指在某个网络服务中从一个远程账号/分区克隆到另一个远程账号/分区。Fork 并不是 Git 命令；它是网络服务(例如 Bitbucket 或 GitHub)的一种操作。据我们所知，两个较知名的网络服务——Bitbucket 和 GitHub 并不允许在它们的服务器之间进行分叉操作。分叉项目是一个将项目从其原来的远程仓库复制到自己的远程 Git 仓库的过程，目的是要修改它或者做一些衍生工作。

历史上，分叉有一定的贬义，因为它通常是项目成员之间最终目标不一致或者存在分歧的结果。这些差异通常会导致来自多个小组的看似相同软件的多种版本，而用户社区却没有清晰的官方版本可以依赖。然而现在，多亏了 Git，分叉得到了极大的鼓励。分叉已经成为协作的一个自然组成部分。许多开源项目将分叉作为提高整体代码层次的一种方法。成员们鼓励彼此做分叉并对代码做出改进。这些改进可以通过 pull 请求的方式迁入到原始项目中，或者将单个人的 bug 修改或特性加入到主线中。由于使用 Git 进行合并和分支非常灵活，因此可以将任何东西(从单独的一次提交到整个分支)迁入自己的仓库。

本章并不会涵盖拉取请求和开放源码合作的完整内容，但会涵盖使用此项强大协作功能的方法。登录你的 Bitbucket 账号并查看 Bitbucket 上的案例分析。如果你没有 Bitbucket 账号，那么使用浏览器访问 bitbucket.org 并注册。注册需要花费大约 30 秒。一旦登录 Bitbucket，就可以通过使用 Bitbucket 网页界面右上角的搜索框找到 Reminders 仓库。在搜索框中，输入 `csgerber/reminders`。此外，不要对所得结果中的 `reminders-git` 仓库感到困惑，你之前已经将其克隆出来作为参考了。要分叉此项目，单击左侧边缘的 Fork 按钮，如图 7-6 所示。在后续的提示窗口中，接受默认值并单击 Fork repository 按钮，如图 7-7 所示。

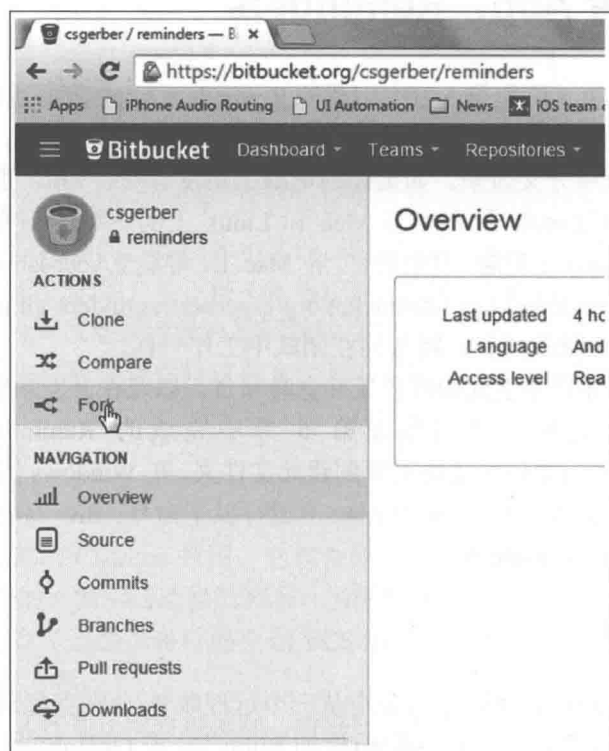


图 7-6 在 Reminders 仓库左侧边栏的控件中单击 Fork

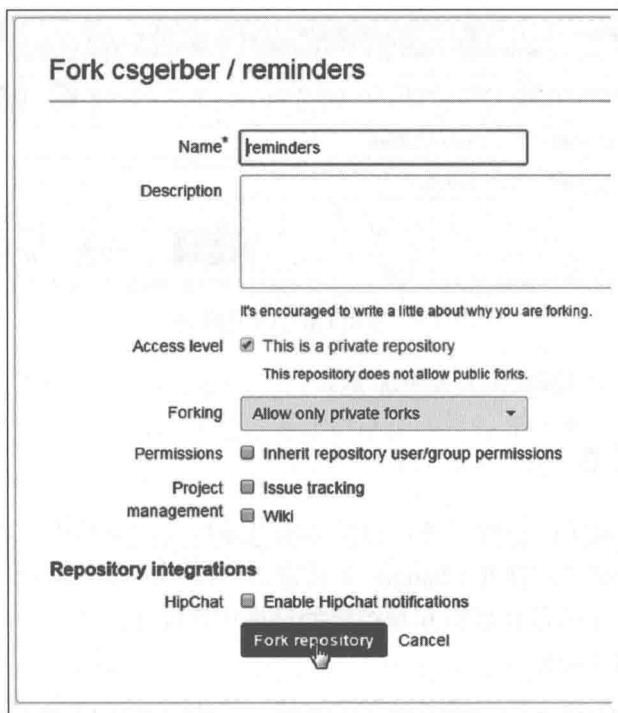


图 7-7 单击 Fork repository 按钮

现在，我们将克隆你刚刚分叉过的仓库。在 Git 中，克隆是从另一个位置复制整个 Git 项目的过程(通常是从远程到本地)。找到你的项目分叉并复制 URL。可以通过在 Bitbucket 网页的搜索框中输入你的 bitbucket 用户名/reminders 来实现。在搜索框的正下方，沿着 Bitbucket 网页的右上方，你将会找到克隆框，里面会有看上去类似于 `git@bitbucket.org:csgerber/reminders.git` 或 `https://your-bitbucket-username@bitbucket.org/your-bitbucketusername/reminders.git` 的 URL。如果没有 http URL，那么单击 URL 旁边的按钮，它的标签应该是 SSH，如图 7-8 所示。它将显示一个允许你选择 http URL 的下拉菜单。从 Version Control | Git 导航至 VCS | Checkout。图 7-9 中所示的对话框将会打开，提示你输入 VCS Repository URL、Parent Directory 和 Directory Name。VCSRepository URL 是之前克隆框中的 URL，而 Parent Directory 和 Directory Name 的组合是你想要复制到本地计算机上的位置。默认情况下，Directory Name 中的项目名是小写的。我们建议使用大写字母命名项目，因此请按照图 7-9 来做些修改。

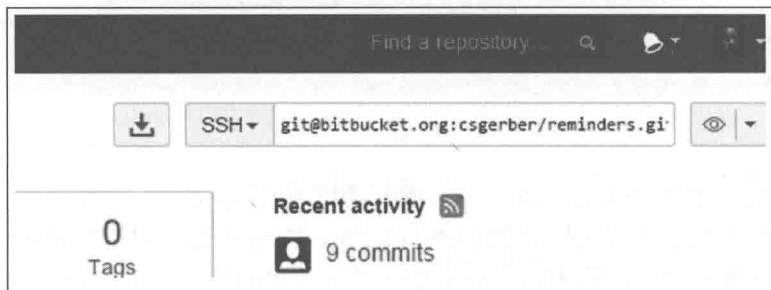


图 7-8 Bitbucket 分享 URL

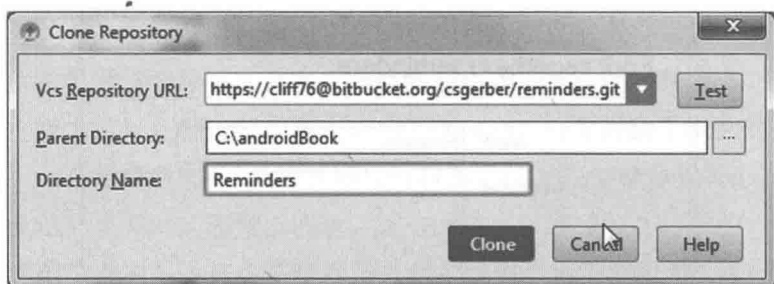


图 7-9 使用 Git GUI 克隆仓库

单击 Clone，即可将源代码复制到本地。

7.4.2 使用 Git 日志

Git 日志是一个强大的特性，使得你能够浏览项目的提交历史。单击对应的工具按钮或者按下 Alt+9 | Cmd+9，打开 Changes 工具窗口，接着选择 Log 选项卡展现出日志。图 7-10 展示了 Reminders 项目直至第 6 章结尾处最后一次提交的历史。这个视图展示了与仓库中单个分支相关联的时间线。

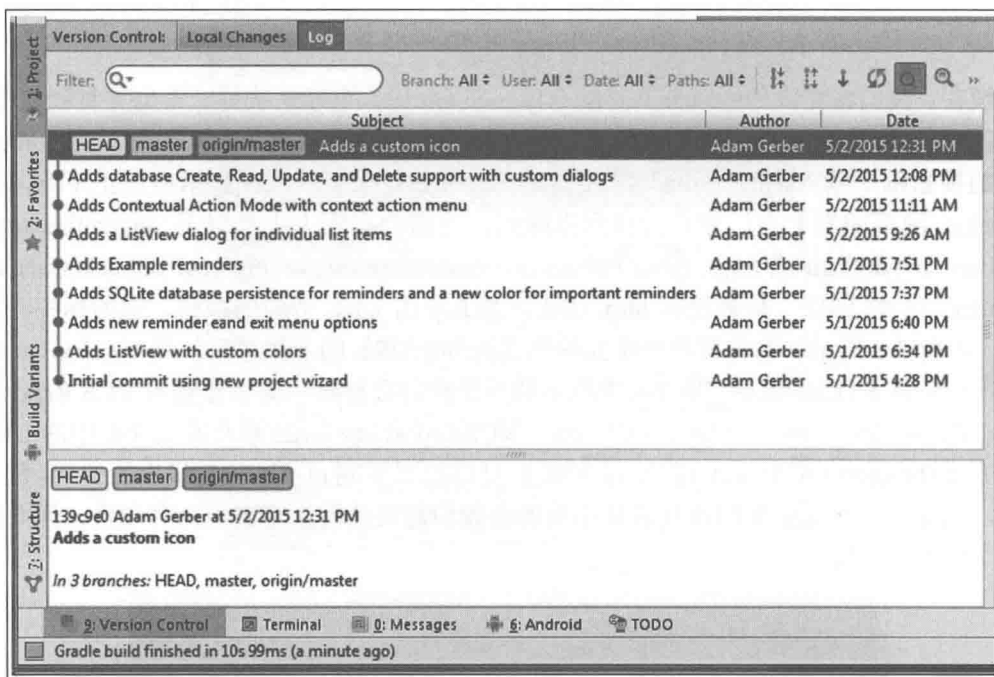


图 7-10 探索 Git 日志

单击时间线中的单个条目会在右侧的修改列表中显示文件；这些是提交时修改的文件。单击任意特定提交中的文件并按 Ctrl+D | Cmd+D(或者仅是双击它们)会得到可视化文本 diff，它是一种并排的比较工具，突出显示了文件中的修改。可以使用修改列表上方的工具栏按钮来编辑源代码、打开文件的仓库版本或者回退选中的修改。也可以使用日志下

方的窗口查看提交作者、日期、时间和哈希代码 ID。这些哈希代码是唯一 ID，当使用 Git 的某些更高级特性时，可以用于标识单次提交。

7.4.3 分支

到现在为止，你的所有提交均是在一个称为 **master** 的单独分支中进行的，这是默认的分支名。然而，你并不需要一直在 **master** 分支上工作。Git 允许你创建任意数量的分支，而分支在 Git 中可以实现多种用途。以下是一种可能的场景。假定你正在和一支开发团队一起工作，并且在开发周期中接到了某些特殊任务。这些任务涉及增加新特性和一些 bug 修改。完成这些工作的一种合理方法是为每项任务建立一个分支。开发者都会认同这一点——当任务完成并通过测试时，开发者需要把任务分支合并到名为 **dev** 的分支中，然后删除任务分支。开发周期结束时，由 QA 团队测试 **dev** 分支，他们可能会拒绝修改并将项目退回开发团队，或者结束周期并将 **dev** 合并到 **master**。这个过程被称为“Git 工作流”，它是使用 Git 进行团队软件开发的推荐方式。关于“Git 工作流”的更多内容请参见如下网址：

<https://guides.github.com/introduction/flow/index.html>

Git 工作流对于大型团队来说工作良好，但是如果你单独开发或者仅与一两个开发者合作的话，那么你可能需要其他工作流。不管采用何种工作流，Git 中的分支功能都很灵活而且允许你将工作流与 Git 相互适配。在本节中，我们假定你正在参与一个团队项目并且得到了在 **Reminders App** 中添加特性的任务，该任务是允许用户为 **Reminders** 指定一天内的某个特定时间。

7.5 在分支上开发

通过选择 **File | Import Project** 打开你之前克隆的 **Reminders-Git** 项目。右击项目视图中的 **Reminders-Git** 根文件夹并选择 **Git | Repository | Branches**，打开 **Branches** 提示窗口。这个提示窗口允许你浏览所有的可用分支。单击提示窗口中的 **New Branch**，将分支命名为 **ScheduledReminders**，如图 7-11 所示。

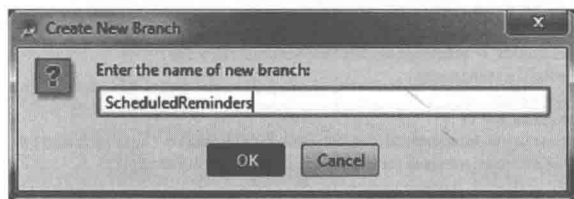


图 7-11 使用 Git 创建新的分支

新的分支将被创建并迁出以供你使用。打开 **Changes** 视图并单击绿色的加号按钮可以创建新的修改列表。和新分支一样，将其命名为 **ScheduledReminders**，因为下一轮修改将会引入设定备忘时间的特性。确保选中了 **Make this changelist active** 复选框，如图 7-12 所示。

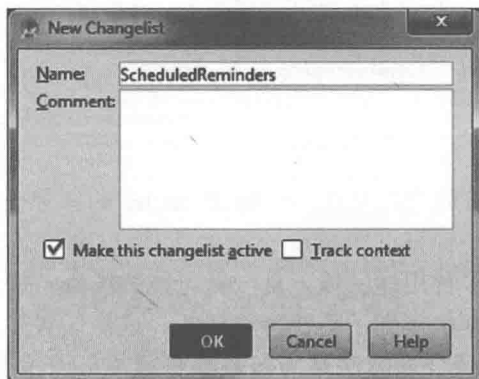


图 7-12 为分支创建新的修改列表

为了开始开发新特性，需要向单击备忘时出现的对话框添加一个新选项。打开 `RemindersActivity.java` 并转到第一个 `OnItemClickListener` 内嵌类(与 `mListView` 相互绑定)的 `onItemClick()` 方法。在用于构建可单击选项的 `String` 数组中将 `Schedule Reminder` 添加为第三个条目，如图 7-13 中的第 92 行所示。接下来，需要允许用户在单击新选项时为备忘设置时间。找到第二个 `OnItemClickListener` 内嵌类(与 `modeListView` 相互绑定)，当单击某条备忘时，它会创建对话框。此对话框会在 `dialog.show()` 方法调用之后显示。观察 `onItemClick()` 方法，并按照图 7-13 所示进行修改。你需要导入 `Date` 类。

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    public void onItemClick(AdapterView<?> parent, View view, final int masterListPosition, long id) {
        AlertDialog.Builder builder = new AlertDialog.Builder(RemindersActivity.this);
        ListView modeListView = new ListView(RemindersActivity.this);
        String[] modes = new String[] { "Edit Reminder", "Delete Reminder", "Schedule Reminder" };
        ArrayAdapter<String> modeAdapter = new ArrayAdapter<>(RemindersActivity.this,
            android.R.layout.simple_list_item_1, android.R.id.text1, modes);
        modeListView.setAdapter(modeAdapter);
        builder.setView(modeListView);
        final Dialog dialog = builder.create();
        dialog.show();
        modeListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                //edit reminder
                if (position == 0) {
                    int nId = getIdFromPosition(masterListPosition);
                    Reminder reminder = mDbAdapter.fetchReminderById(nId);
                    fireCustomDialog(reminder);
                    //delete reminder
                } else if (position == 1) {
                    mDbAdapter.deleteReminderById(getIdFromPosition(masterListPosition));
                    mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
                } else {
                    Date today = new Date();
                    new TimePickerDialog(RemindersActivity.this, null, today.getHours(), today.getMinutes(), false).show();
                }
                dialog.dismiss();
            }
        });
    }
});
```

图 7-13 为了设设备忘时间而进行的修改

这里，你将删除备忘的 `else` 块修改为 `else if` 块，它判断位置值是否为 1。添加一个 `else` 块，当单击了第三个新选项时会运行这个块。这个代码块创建了一个表示今天的新 `Date` 对象并用它来构建 `TimePickerDialog`。现在运行该 App 并测试这个新选项。图 7-14 展示了新特性运行时的样子。

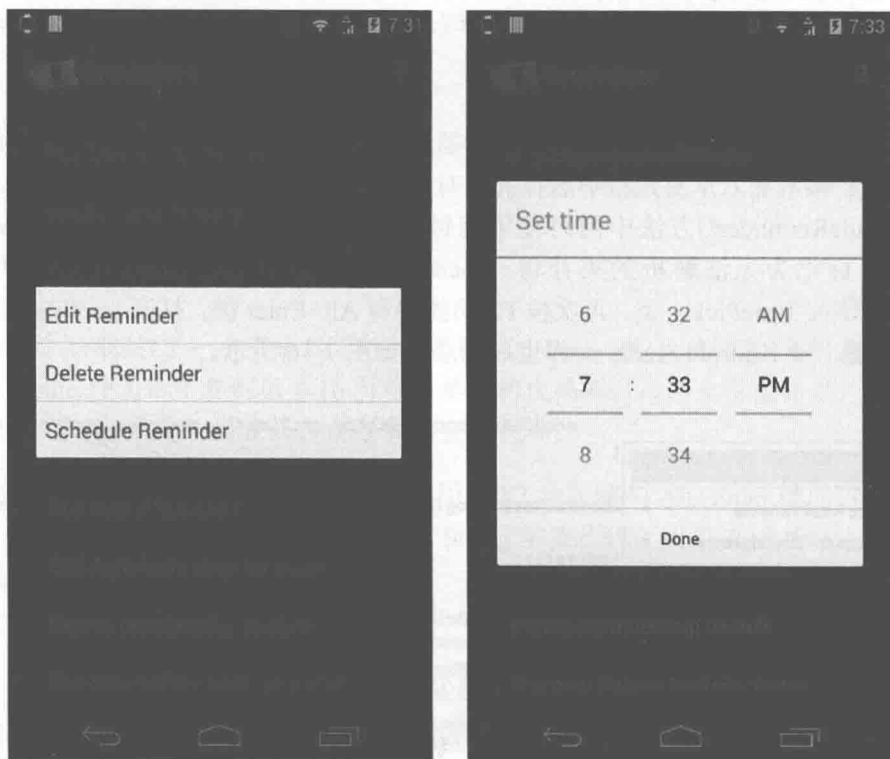


图 7-14 尝试 Schedule Reminder 选项

既然新特性已经能够部分运行了，现在就按 `Ctrl+K` | `Cmd+K` 并提交，附带消息为 `Adds new scheduled time picker option`。回到 IDE 中并将查找备忘的两行代码移至 `position==0` 条件之外。将 `reminder` 变量标记为 `final`。参见图 7-15 中的示例。

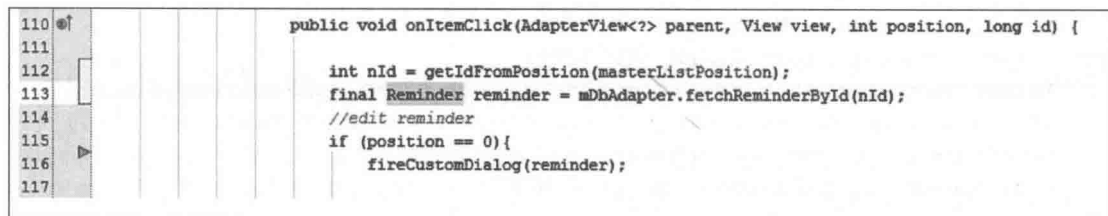


图 7-15 将 `reminder` 变量移到 `if` 块之外

接着转到刚刚添加的 `else` 块，你在这里构造并显示时间选取器对话框。在显示对话框的代码行(对应图 7-13 中的第 113 行)之前添加如下代码：

```
final Date today = new Date();
```

```

    TimePickerDialog.OnTimeSetListener listener = new
    TimePickerDialog.OnTimeSetListener() {
        @Override
        public void onTimeSet(TimePicker timePicker, int hour, int minute) {
            Date alarm = new Date(today.getYear(), today.getMonth(),
            today.getDate(), hour, minute);
            scheduleReminder(alarm.getTime(), reminder.getContent());
        }
    };

```

这为时间选取器对话框创建了一个监听器。在此监听器中，使用今天的日期作为闹钟的基准时间。接着加入从对话框中选择的小时和分钟并为备忘创建闹钟日期变量。可以在新的 `scheduleReminder()` 方法中同时使用闹钟时间和备忘内容。Android Studio 将会把 `TimePicker` 标记为无法解析的类并将 `scheduleReminder()` 标记为无法解析的方法。按 `Alt+Enter`，导入 `TimePicker` 类。再次按 `F2` 功能键和 `Alt+Enter` 键，打开 `IntelliSense` 对话框并按 `Enter` 键，让 Android Studio 为你生成方法，如图 7-16 所示。

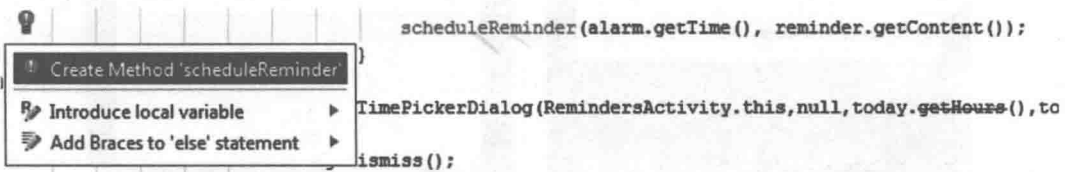


图 7-16 使用 `IntelliSense` 生成方法

选择 `RemindersActivity` 类，如图 7-17 所示。

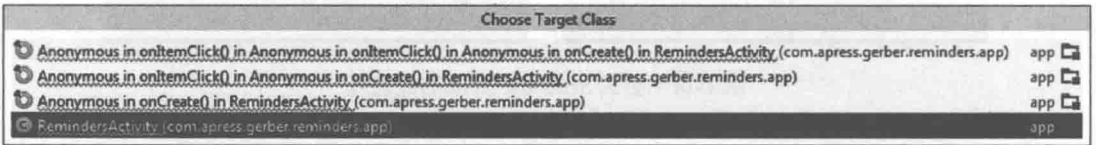


图 7-17 将 `RemindersActivity` 选为目标类

在新的方法体内添加以下代码：

```

    AlarmManager alarmManager = (AlarmManager)
    getSystemService(Context.ALARM_SERVICE);
    Intent alarmIntent = new Intent(this, ReminderAlarmReceiver.class);
    alarmIntent.putExtra(ReminderAlarmReceiver.REMINDER_TEXT, content);
    PendingIntent broadcast = PendingIntent.getBroadcast(this, 0, alarmIntent, 0);
    alarmManager.set(AlarmManager.RTC_WAKEUP, time, broadcast);

```

同样，Android Studio 将会标识出一系列错误，因为代码中缺少导入语句。按 `F2` 功能键和 `Alt+Enter` 键打开快速修改提示框并修改每个错误。快速修改选项会提示你 `ReminderAlarmReceiver` 不存在。按 `Alt+Enter` 键并选择第一项来生成类。在第一个弹出对话框中按 `Enter` 键，使用建议的包，接着在第二个弹出对话框中再次按 `Enter` 键，将这个新的类文件添加到 `Git`。确保该类派生自 `BroadcastReceiver` 并实现了 `onReceive()` 方法。你的

ReminderReceiver.java 文件看上去应该类似下面这样：

```
package com.apress.gerber.reminders;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class ReminderAlarmReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

    }
}
```

提示

在按 F2 功能键(下一个突出显示的错误)和 Alt+Enter 键(快速修复)之间反复切换，让 Android Studio 修复由于复制图 7-16 所示清单中的代码而引起的大多数错误。它将会导入缺失的类，同时为未定义的方法、常量和类生成代码。

返回到 RemindersActivity.java 文件。使用按 F2 功能键和 Alt+Enter 键的方法找到并修复最后一个错误——选择第二个建议：生成 String 常量代码，如图 7-18 所示。将此文本的值设置为 REMINDER_TEXT。

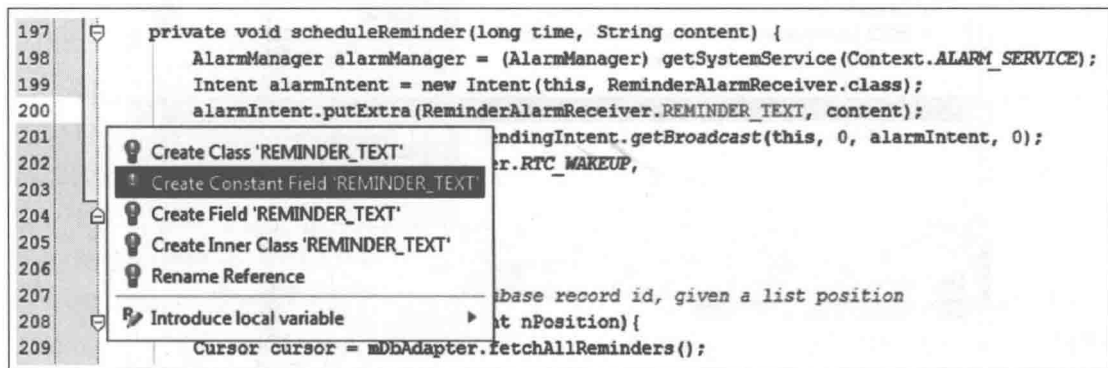


图 7-18 生成常量字段

最后，打开 AndroidManifest.xml 文件并添加接收器标签来定义新的 BroadcastReceiver，如图 7-19 所示。

运行 App 进行测试。你应该能够单击一条备忘、选择 Schedule Reminder 并为其设置触发的时间。选择时间操作并没有产生任何效果，因为我们还没有详细涉及 BroadcastReceiver。现在按 Ctrl+K | Cmd+K，打开 Commit Changes 对话框。在 Commit Changes 对话框中花一些时间确认已经做出的修改。注意，对话框中保留了上一次提交时的消息，你应该更新它，如图 7-20 所示。


```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.apress.gerber.reminders" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Reminders"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".RemindersActivity"
            android:label="Reminders" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".ReminderAlarmReceiver"/>
    </application>
</manifest>

```

图 7-19 manifest 中的 BroadcastReceiver

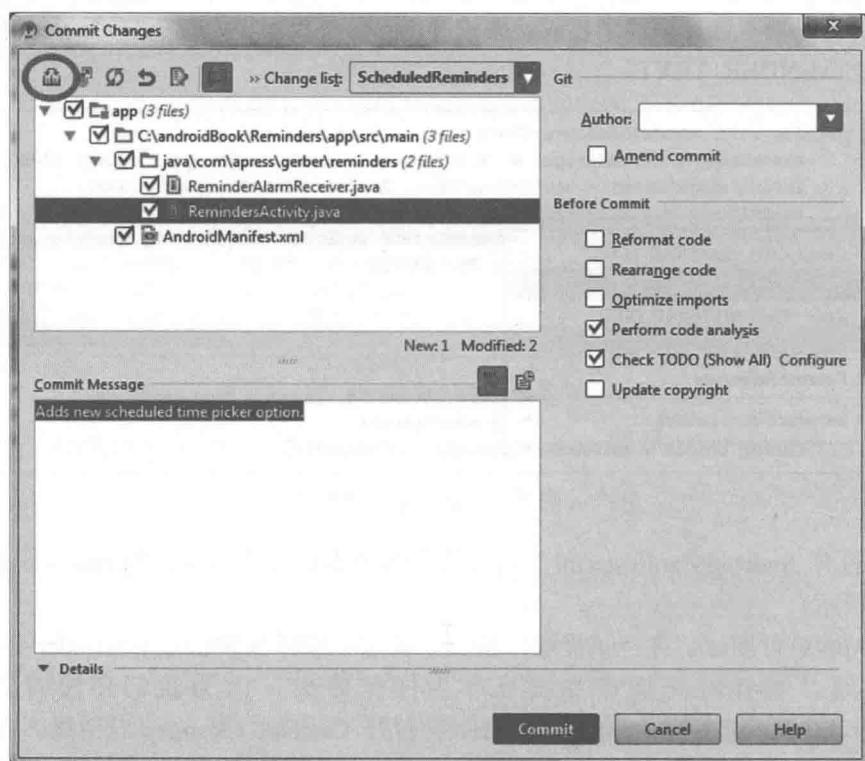


图 7-20 Git 的 Commit Changes 对话框

选择了 RemindersActivity 之后，单击 Show Diff 按钮(如图 7-20 所示)打开一个并排比较所有修改的窗口。按左上角的向上和向下箭头或者按 F7 键在文件中较老和较新的差异

之间移动。图 7-21 展示了这三个控件。使用向下箭头按钮移到 onItemClickListener 中感兴趣的修改处。

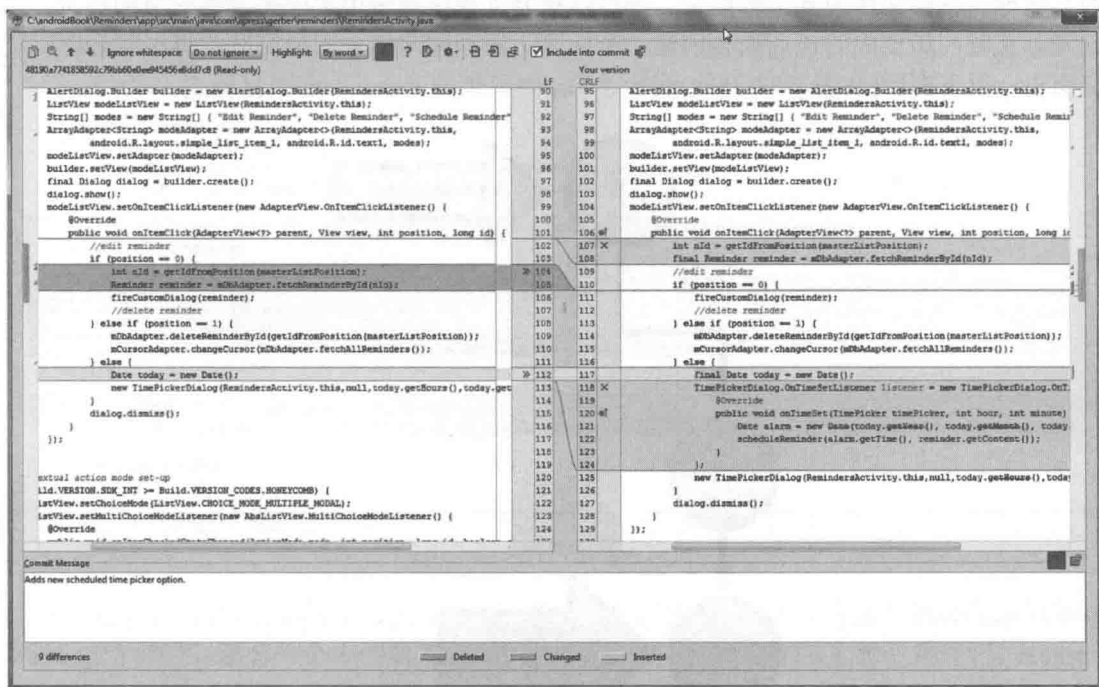


图 7-21 可视化的文本差异视图

到目前为止，你已经添加了 OnTimeSetListener，但是还没有用它(浅灰色的 listener 变量表示没有在代码中使用它)。在此视图下遍历代码的过程中，你不仅可以回忆起已经完成的修改，而且还会发现那些被遗漏的修改，这再一次为你提供了在提交之前修改问题的机会。差异视图还是一个具有某些语法感知特性的编辑器，因此你可以选择使用它做一些微小的调整，也可以利用诸如自动代码补全等特性。

按 Escape 键关闭差异视图，在提交修改之前更新提交信息。单击 Commit 并允许 Android Studio 执行代码分析。你将会看到另一个对话框，它告知你有些文件中存在问题。对话框将会提示代码中含有警告。此时，可以单击 Review 按钮来取消提交并生成所有潜在问题的列表。尽管忽略警告并不是好的实践，但你现在可以将其保留并继续下面的步骤。

7.5.1 Git 提交和分支

如果有着使用类似 Perforce 或 Subversion 等传统 VCS 工具的背景，那么你会觉得 Git 风格的分支提交虽然很类似，但又与之前所熟悉的工具有一定不同。你将会希望了解 Git 在提交和分支管理方式上的细微差别。这些差别会让新手感到困惑，但它们是让 Git 具备强大功能和灵活特性的核心。

Git 中的提交被视作项目历史中的头等实体，可以通过一个特殊的提交哈希代码来标识它。虽然你不需要了解 Git 实现单次提交和版本控制的细节，但很重要的一点是将提交

视作存在于历史时间轴中代表仓库整体状态的对象或实体。提交是发生在 Git 历史中某个时间点上具有原子性的工作单元，带有一段用于描述工作的提交消息。每次提交均有之前的一次或多次提交作为其父级提交。你可以将分支视作指向历史中某一次提交的标签。当创建分支时，历史中的该点处会创建一个标签；而当你向该分支提交时，该标签会跟踪提交历史。以下框图(从图 7-22 开始)演示了 Git 中记录的 Reminders 项目历史。

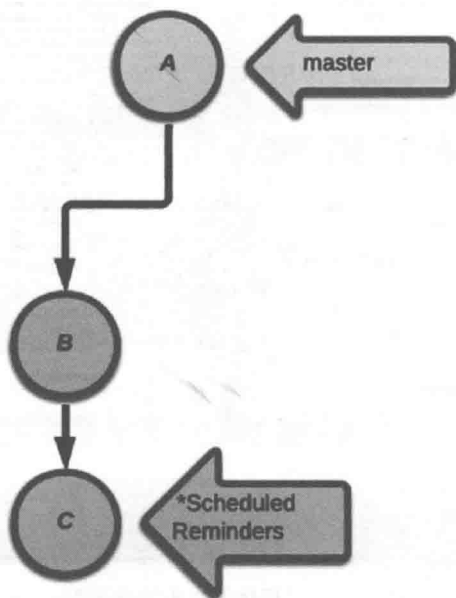


图 7-22 使用 Git 历史显示 ScheduledReminders 分支

注意

Android Studio 从下到上展示提交日志，而我们的框图从上到下进行展示。

主分支由灰色箭头表示，它指向已克隆项目的最后一次提交 A(与 Git 日志视图做比较，你将会发现 A 之前还有其他提交，但为了简洁，这里将它们略去)。ScheduledReminders 分支是绿色箭头，指向系列提交 B 和 C(实现了新特性)中最新的一次。为了简单，我们使用单个字母作为标签，但 Git 使用提交哈希代码，它有着更长的十六进制名称，例如 c04ee425eb5068e95c1e5758f6b36c6bb96f6938。你可以使用哈希代码的前几个字符来引用某次特定提交，只要它们是唯一的或者与其他哈希代码的前几个字母不同即可。

7.5.2 回退在哪里？

很多人初次使用 Git 时遇到的一个较大困难是适应 Git 的回退操作，因为它们与其他 VCS 客户端的工作方式不同。Git revert 是一次解除之前提交的提交(工作单元)。理解它的最好方式是实际观察一下。让我们做些修改，解决掉 RemindersActivity.java 中的已废弃警告。引入 Calendar 对象并删除 Date 对象，如图 7-23 所示。

```

@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

    int nId = getIdFromPosition(masterListPosition);
    final Reminder reminder = mDbAdapter.fetchReminderById(nId);
    //edit reminder
    if (position == 0){
        fireCustomDialog(reminder);

        //delete reminder
    } else if (position == 1) {
        mDbAdapter.deleteReminderById(getIdFromPosition(masterListPosition));
        mCursorAdapter.changeCursor(mDbAdapter.fetchAllReminders());
    } else {
        TimePickerDialog.OnTimeSetListener listener = new TimePickerDialog.OnTimeSetListener() {
            @Override
            public void onTimeSet(TimePicker timePicker, int hour, int minute) {
                final Calendar alarmTime = Calendar.getInstance();
                alarmTime.set(Calendar.HOUR, hour);
                alarmTime.set(Calendar.MINUTE, minute);
                scheduleReminder(alarmTime.getTimeInMillis(), reminder.getContent());
            }
        };
        final Calendar today = Calendar.getInstance();
        new TimePickerDialog(RemindersActivity.this, null, today.get(Calendar.HOUR), today.get(Calendar.MINUTE), false).show();
    }
    dialog.dismiss();
}

```

图 7-23 解决已废弃警告

构建并运行代码，验证修改起了作用，接着提交这些修改，附带消息为 Fixes deprecation warnings。注意，还有一个未使用变量的警告，我们将在后面的 7.5.8 节中处理它。Android Studio 中的回退命令与 Git 回退命令大相径庭。这里，你将要在命令行中使用 `git revert` 命令来体会其中的差异。在 Changes 工具窗口的 Git 历史中找到“Fixes deprecation warnings”这次提交，右击它并选择复制哈希代码，将提交的哈希代码复制到系统剪贴板中。现在单击底部边栏处的终端窗口按钮，打开终端并输入 `git revert`，然后将提交的哈希代码粘贴在命令的最后。你的命令看上去应该类似于图 7-24。按 Enter 键，Git 将在终端中打开提交消息编辑会话，如图 7-25 所示。输入“:q”，退出编辑会话，此操作会保存默认提交消息并执行提交。

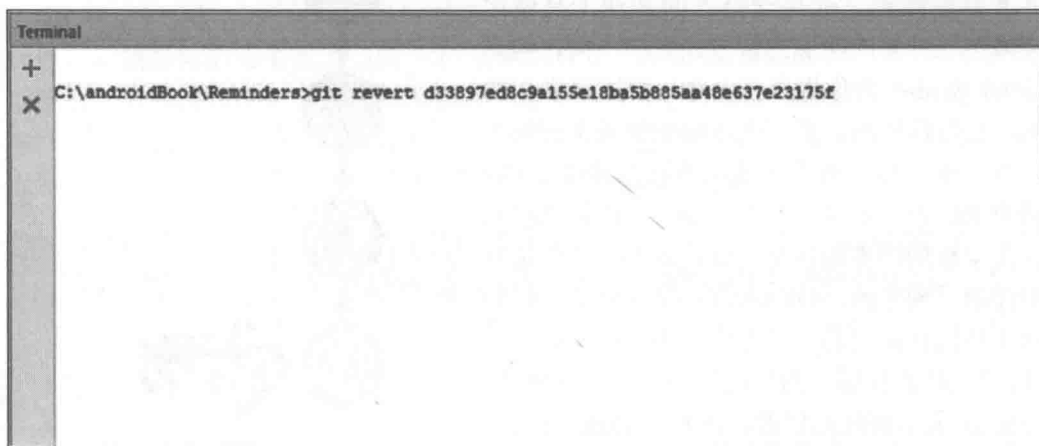


图 7-24 通过终端发出 git revert 命令

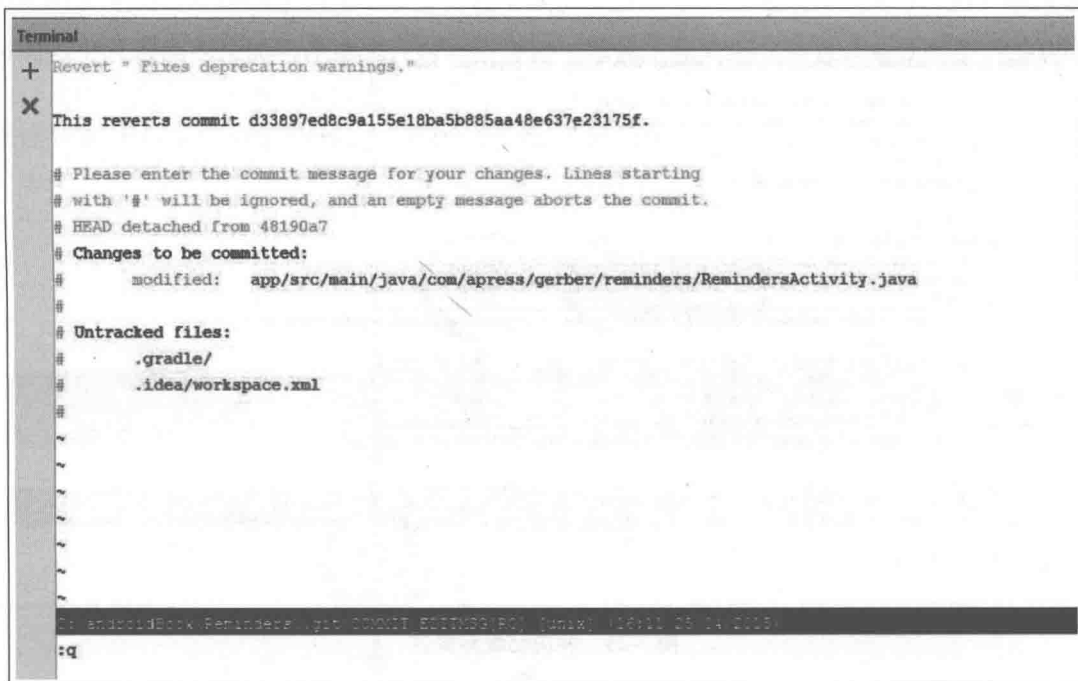


图 7-25 提交消息编辑, 输入:q 可以退出

Git 回退将产生一次新的提交, 它会解除之前的提交。切换回 Android Studio 并观察发生的变化。修改已经解除, 所有的已废弃警告都回来了。你的 Git 历史将体现出此次提交记录。Git 会应用之前提交的反向修改并随即使用这些修改执行一次提交, 该提交使用与最后一次提交相同的消息并加上 Revert 前缀。将这一点与跟踪本地文件修改并允许你撤消之前提交修改的其他工具做比较。虽然这种修改回退的新方式有些不同, 但是 Android Studio 为你提供了一个进行回退操作的界面, 它与传统的、更为人所熟知的版本控制工具一致。在编写本书的时候, 还没有等效于 Git 回退的 IDE 命令或菜单操作。不过, 内置选项允许你在本地应用本地历史、Git 历史, 甚至是补丁文件的反向修改。Git 回退会自动完成应用反向修改和执行提交的两步操作。图 7-26 展示了 Git 历史, 提交 D 引入的修改解决了已废弃问题, 而提交 -D 表示解除此修改, 恢复对 Date 对象的已过期调用。

解除已提交修改的另外一种方法是使用

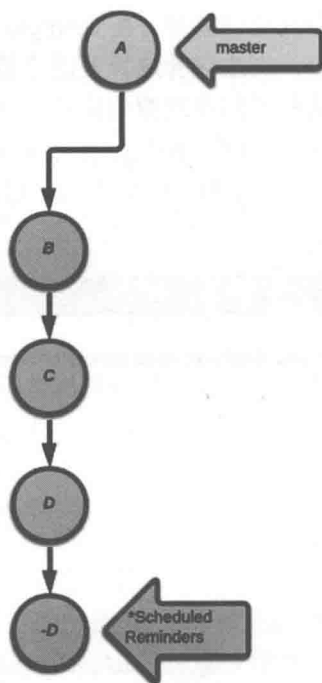


图 7-26 回退之后的 Git 历史

reset 命令，它类似于 revert，但有着微妙的差别。将图 7-23 中所示的修改放回到源代码中并再次提交它们。你的 Git 历史将在-D 之后又多出一次提交 E，如图 7-27 所示。这次选择 VCS | Git | Reset Head。在弹出的对话框中输入 HEAD~1，如图 7-28 所示，单击 Reset 按钮。

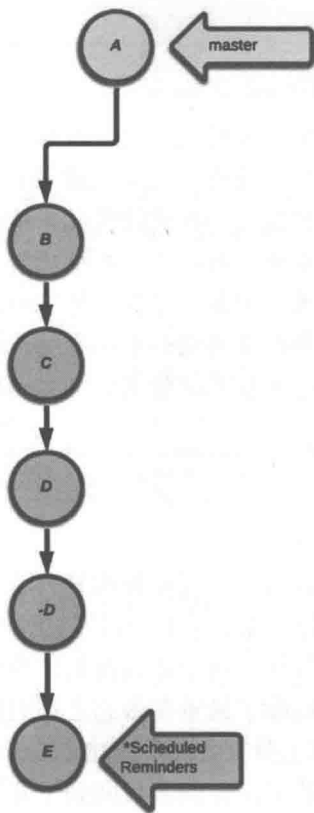


图 7-27 重新应用“解决已废弃警告”之后的 Git 历史

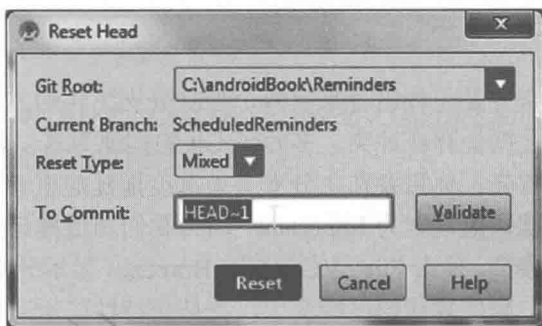


图 7-28 Git Reset Head 对话框

Git 将会把你的仓库同步到最后提交之前的一次提交，这等效于对该次提交执行了撤销操作——使得你的历史变为如图 7-26 所示的样子。Android Studio 加强了 Git 重置操作，可以使用当前修改列表重新应用修改。万一意外地执行了重置操作，这使得你有第二次机会撤回提交。在大多数情况下，你希望彻底丢弃重置之后的修改。单击修改工具窗口中的回退修改可以彻底丢弃修改。图 7-29 中圈出了回退修改按钮。

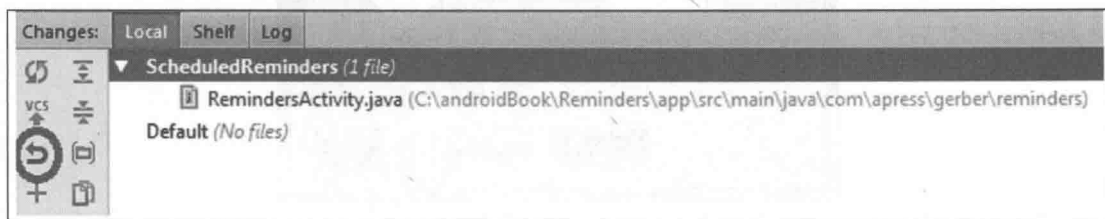


图 7-29 单击回退修改按钮

让我们进一步删除关于已废弃方法调用的所有工作痕迹。选择 VCS | Git | Reset Head。接着在弹出的对话框中输入 HEAD~2，如图 7-28 所示，单击 Reset 按钮。记住之后单击回退修改按钮。这将会成为你在 Android Studio 中使用 Git Reset 时的一个习惯。你的历史将会体现出这些内容，如图 7-22 所示。

回退与重置

回退和重置之间的差别比较微妙，却很重要。回退会增加一次提交，它将最后提交的修改反转，而重置会将提交删除。重置本质上就是将你的分支标签返回到某次指定编号的提交。如果错误提交了一些内容，你通常需要撤消或删除某次提交。在这种情况下使用重置是合理的，因为它是最简单的选择而且不会添加到你的历史中。然而在某些情况下，你可能想要历史中体现出解除提交的工作——例如，如果你从项目中抽出了某个特性而且想要为用户社区编写说明删除该特性的文档。回退的另一个重要用途与远程仓库有关，我们将在本章的后面讨论。由于你只能向远程仓库添加提交，因此删除或解除某次提交的仅有方法就是使用回退，它会将反向修改追加为一次提交。

7.5.3 合并

合并是一种组合来自两个独立分支工作的方法。历史上，合并需要额外的努力，因为分支之间会存在冲突。多亏了 Git 的实现方式，合并已经变得没那么令人痛苦了。

首先，你将在主分支上为重度拖延症患者添加一个新特性。这个新特性将把所有备忘的默认值设置为 Important，因为我们知道拖延症患者会忽略除了最重要备忘之外的所有其他事情。单击 File | VCS | Git | Branches 显示分支列表。选择主分支并接着单击 Checkout。注意，底层源代码已经变了，支持新特性的所有修改都被删除了，你的项目回到了开始添加设置备忘时间功能之前的状态。创建一个新的命名修改列表并将其设置为活动状态。当出现提示时，删除空的 ScheduledReminders 修改列表。图 7-30 和图 7-31 演示了这个流程。

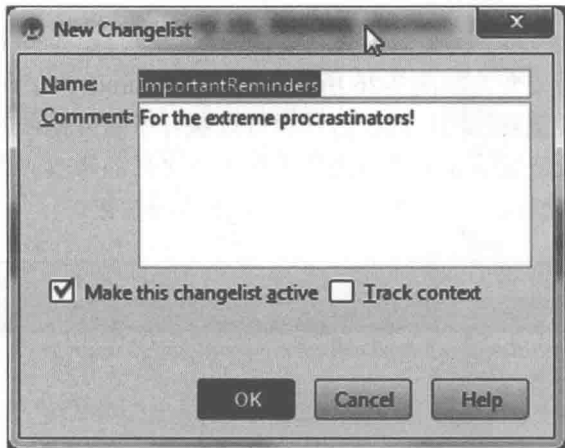


图 7-30 新建修改列表对话框

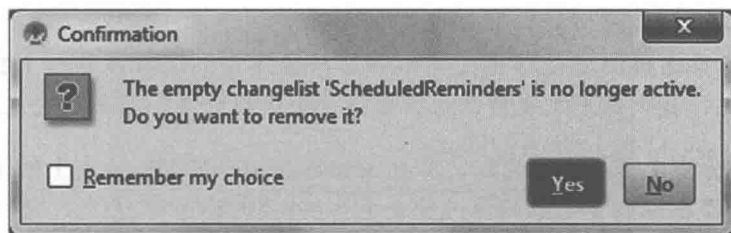


图 7-31 当删除旧的修改列表时会出现一个确认对话框

查看 `fireCustomDialog()` 方法并找到从对话框布局中获取复选框的那行代码。添加一行新代码，调用 `checkBox.setChecked(true)`，这将会设置新的默认值，如图 7-32 中的第 200 行所示。

```

196 TextView textView = (TextView) dialog.findViewById(R.id.custom_title);
197 final EditText editCustom = (EditText) dialog.findViewById(R.id.custom_edit_reminder);
198 Button buttonCustom = (Button) dialog.findViewById(R.id.custom_button_commit);
199 final CheckBox checkBox = (CheckBox) dialog.findViewById(R.id.custom_check_box);
200 checkBox.setChecked(true); //All reminders are important!
201 LinearLayout linearLayout = (LinearLayout) dialog.findViewById(R.id.custom_root_layout);

```

图 7-32 将复选框设置为默认选中

构建并运行 App，测试新特性，接着按 `Ctrl+K` | `Cmd+K` 进行提交。Git 历史将会如图 7-33 中所示，它表示最后一次提交位于紧随初始克隆之后的分支中。

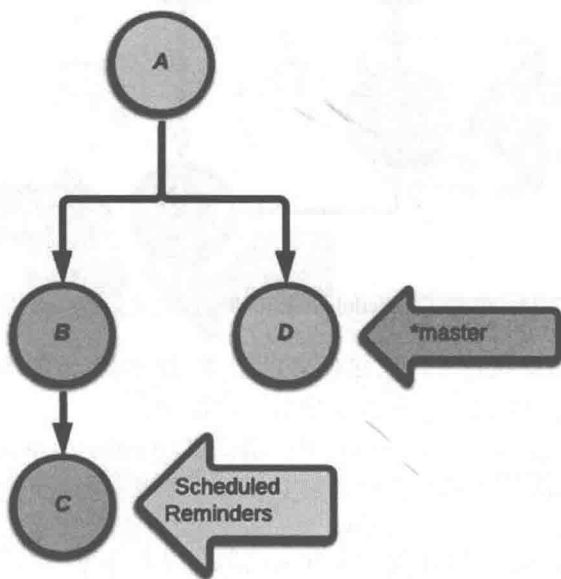


图 7-33 向主分支添加特性之后的提交历史

这里，你将 `HEAD` 切换至主分支并完成提交 `D`。最后一次提交与针对 `ScheduledReminders` 特性的提交有着不同的历史路径，因为这次提交没有在相同的分支上进行。

注意

如果在 Git 日志视图中追踪历史，那么你将会注意到还有另外一个指向 A 提交的 origin/master 分支，我们这里没有展示它。这是一个远程分支，我们将会在后面讨论。

你已经在主分支上做了一些工作，并在 ScheduledReminders 分支上做了一些提交来增加新特性，因此现在你将要把这些修改集合到主线(或者说主分支)上，使得其他人可以看到它们。再次单击 File | VCS | Git | Branches 显示分支列表。选择 ScheduledReminders 分支并单击 Merge。该分支上所有的修改和历史将会融入你的主分支。构建并运行 App，测试这两个特性。单击选项菜单中的 New Reminder 将会打开 New Reminder 对话框，其中的 Important 复选框处于选中状态；而单击列表中的任意备忘都会出现为其设置时间的选项。图 7-34 展示了 Git 是如何管理修改的。

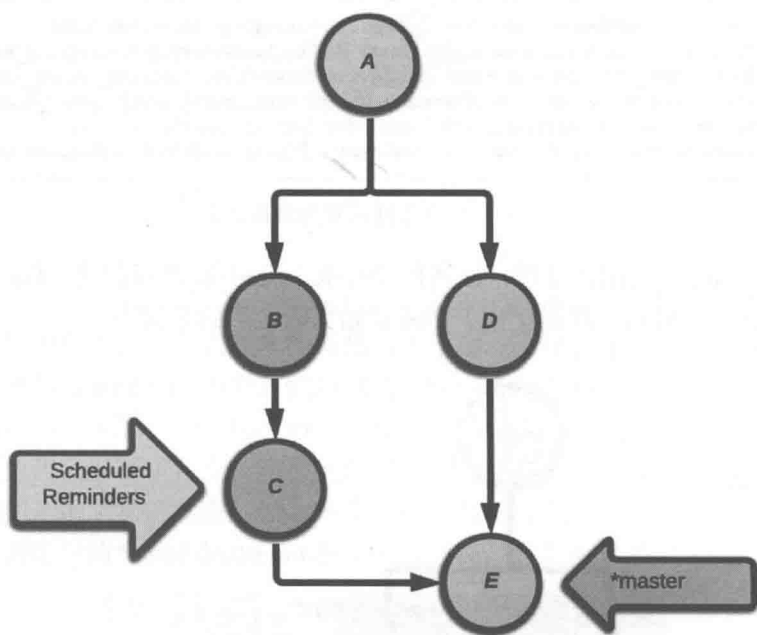


图 7-34 合并了 ScheduledReminders 特性之后的提交历史

一次新的提交 E 自动执行了，它包含来自 C 和 D 的修改(E 的父级提交)。还要注意 HEAD 现在指向主分支的头部，它包含了最后一次提交。

7.5.4 Git 重置修改历史

如果想要将重要的备忘特性作为一个分支该怎么办？你根本没有为此特性创建分支。相反，你就是在主分支顶部进行开发。你需要强制主分支向后退并指向提交 D，那么现在就让我们来实现吧！单击 File | VCS | Git 并单击 Reset Head。To Commit 文本框中的内容将被设置为 HEAD。如图 7-35 所示，将其设置为 HEAD~1 并单击 Reset 按钮再次重置主分支(更像是一个标签)。记得回退 Git 重置所保存的修改。它将会指向之前的提交。现在的 Git 仓库如之前的图 7-33 所示。



图 7-35 Git Reset Head 对话框

由于最后一次提交包含合并的修改，因此重置操作使得合并根本没有发生而且让你处于最新的提交状态(引入了 ImportantReminders 特性)。这使得你可以随意修改历史并让这个新特性看上去好像是在分支上开发的。单击 File | VCS | Git，接着单击 Branches，打开分支对话框。单击 New Branches，将这个分支命名为 ImportantReminders 并单击 OK 按钮来创建它。现在，你的历史应该如图 7-36 所示。

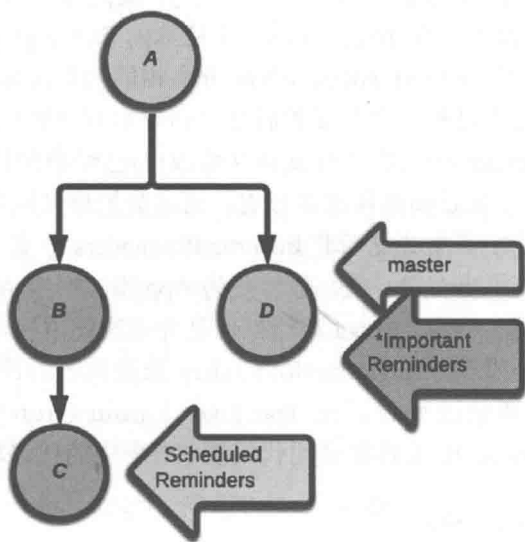


图 7-36 显示出新分支的 Git 历史

主分支和 ImportantReminders 分支均指向相同的提交。使用 Branches 对话框迁出主分支，该对话框可以通过单击状态栏右侧的分支区域或者选择 File | VCS | Git | Branches 来打开。再次重置此分支，让其指向刚刚将项目从 Bitbucket 克隆出来时的版本，接着迁出 ImportantReminders 分支。现在可以从历史中看出两个实验性质的特性分支均处于开发中，而(在 IDE 中看到的)工作副本中的项目处于刚刚克隆出来时的状态。可以在图 7-37 中看到这一点。

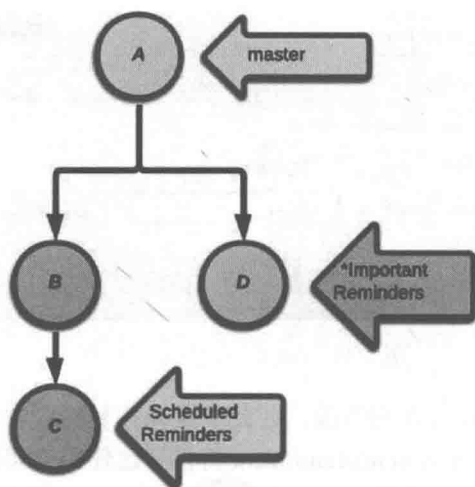


图 7-37 将主分支重置到初始版本之后的 Git 历史

现在，你想要进一步修改历史并重新排列特性提交顺序，使得它们看上去像是按顺序开发的而且开发过程中没有使用分支。在这样做之前，迁出主分支并将其与 `ImportantReminders` 分支合并。合并将会导致一个特殊的 `Fast Forward` 操作：Git 只是在历史中将主分支向前推进到与 `ImportantReminders` 分支相同的那次提交。这与之前的合并分支示例不同，因为一个分支是另一个分支的后裔。如果观察足够仔细，你将会发现创建新的提交(从 `ImportantReminders` 分支向主分支合并修改)与已经指向相同分支的提交 D 相同。因此，Git 优化了该操作，只是向前移动主分支，而这会把你带回到类似于图 7-36 所示的历史中。不同之处是你迁出了主分支而非 `ImportantReminders` 分支。

现在，我们让历史变得更有意思一些。你将要为 App 添加一个 `About` 对话框，以便用户能够了解有关开发者的更多信息。`About` 对话框也是介绍所使用技术和美工的一个好地方。你的会相对简单。如果还没删除 `ImportantReminders` 修改列表的话，那么删除它，创建一个名为 `AboutScreen` 的新修改列表。在 `app | src | main | res | layout` 下面创建名为 `dialog_about.xml` 的新资源 XML 文件并使用代码清单 7-1 中的代码填充它。

代码清单 7-1 `dialog_about.xml`

```

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

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Reminders!"
        android:id="@+id/textView2"
        android:layout_alignParentTop="true"
    
```

```

        android:layout_centerHorizontal="true" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Version 1.0\nAll rights reserved\nDeveloped by yours
truly!"
    android:id="@+id/textView3"
    android:layout_marginTop="34dp"
    android:layout_below="@+id/imageView"
    android:layout_centerHorizontal="true"
    android:gravity="center" />

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true"
    android:src="@drawable/ic_launcher" />
</RelativeLayout>

```

这个布局定义了 About 对话框，它包含用于标题和内容的文本视图，并将 Reminders 启动图标放置在它们中间。你需要一个新的菜单项来调用这个对话框。打开 menu_reminders.xml 并在第一项和第二项标签之间添加以下 XML 片段：

```

<item android:id="@+id/action_about"
    android:title="About"
    android:orderInCategory="200"
    app:showAsAction="never" />

```

将 Exit 菜单项的 orderInCategory 由 200 修改为 300，使其排列在新的 About 菜单项的后面。

现在打开 RemindersActivity.java 并为新菜单项添加一个 case 分支(调用新的 fireAboutDialog 方法)，如图 7-38 所示。

fireAboutDialog()方法使用新布局构建对话框并显示它。构建并运行 App，测试新特性。最后，按 Ctrl+K | Cmd+K 并提交，附带消息为 Adds an About screen。现在，Git 历史在重要备忘特性之后又有了一次提交并且有一个分支指向它。图 7-39 中的最新提交 E 包含了 About 对话框特性。

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()){
        case R.id.action_new:
        case android.R.id.home:
            //create new Reminder
            fireCustomDialog(null);
            return true;

        case R.id.action_about:
            fireAboutDialog();
            return true;

        case R.id.action_exit:
            finish();
            return true;

        default:
            return false;
    }
}

private void fireAboutDialog() {
    // about dialog
    final Dialog dialog = new Dialog(this);
    dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
    dialog setContentView(R.layout.dialog_about);
    dialog.show();
}

```

图 7-38 添加 About 界面

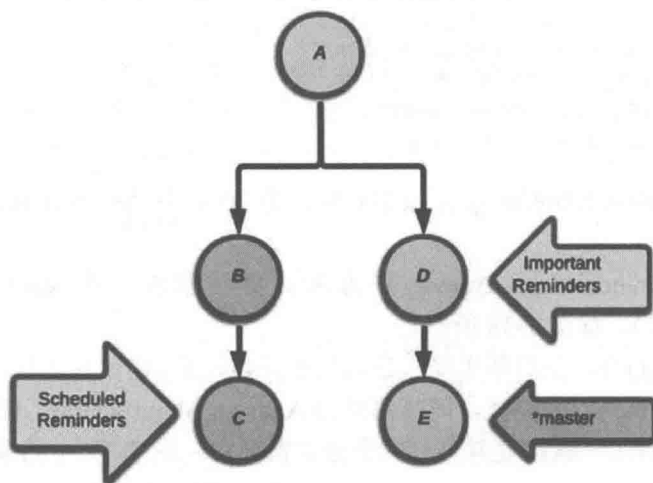


图 7-39 添加了 About 界面之后的 Git 历史

7.5.5 Git 变基

变基(rebasing)是基于另一个分支或一系列提交来建立分支的一种方法。它类似于组合

不同分支上修改的合并操作，但采用的方式是创建没有多个父级提交的提交历史。最好是用当前历史来做例子。单击 File | VCS | Git | Rebase，打开 RebaseBranches 对话框。通过从 Onto 下拉菜单中选择 ScheduledReminders 分支告诉这个对话框你想要将主分支变基到该分支，如图 7-40 所示。将 Interactive 选项保留为选中状态，以便更好地控制将要合并的内容。

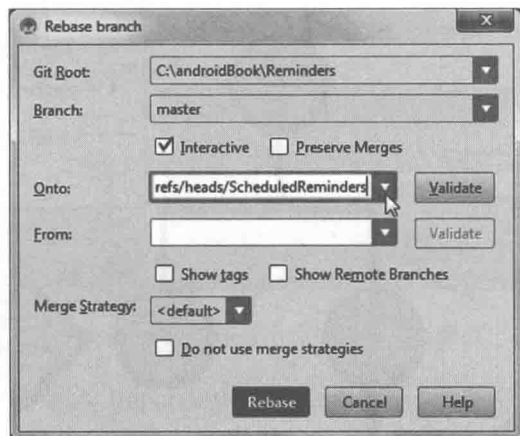


图 7-40 Git 变基分支对话框

你将会进入交互式变基模式，如图 7-41 中的对话框所示。交互式变基是 Git 的强大特性之一。在这里，你可以删除并修改提交历史中的单个提交。Rebasing Commits 对话框列出所选分支历史中发生的所有提交，一直追溯到与 Onto 分支的第一个公共祖先。首先需要注意的是每次提交对应的 Action 列下面的选项。对话框中给出了选项——挑选、编辑、跳过或挤压。不过，Android Studio 会将每次提交默认置为挑选。

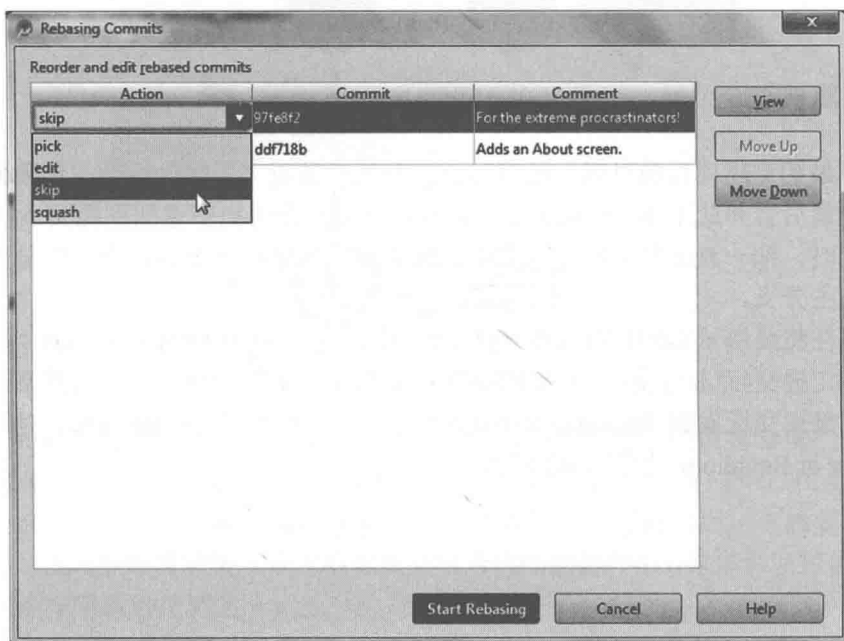


图 7-41 Git 变基提交对话框

让我们假定你不再需要此分支中的 ImportantReminders 特性，但仍然对 About 界面很感兴趣。选择 Skip 操作从列表中删除这次提交，完成变基且合并分支之后，这些修改将不会显示。单击 Start Rebasing 选项完成该操作。你的 Git 历史现在看上去类似于图 7-42 所示。

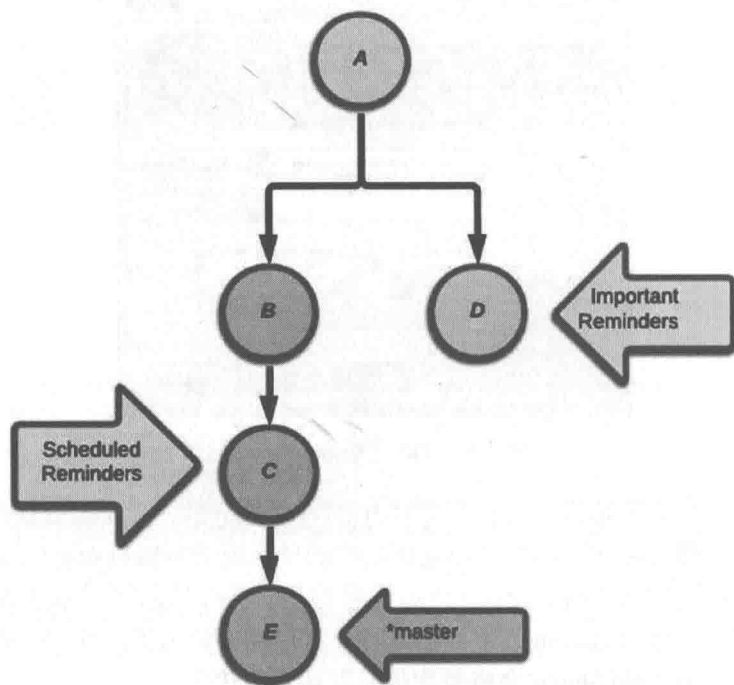


图 7-42 变基和跳过 ImportantReminders 分支之后的情形

7.5.6 分离头部

假定在最初克隆项目的时候，还有另一位开发者负责“闹钟”特性。让我们进一步假定你想要在最后合并此工作。要模仿这种情况，你需要在历史中退回到提交 A 并开启新特性。直到此时，你一直在某个特定分支中工作和提交，包括一个自己命名的分支和最初导入时创建的主分支。

我们现在将要演示 Git 中的另外一种操作方法，它被称为 Detached HEAD 模式。如果迁出某次特定提交(而非分支)，那么 HEAD 就会从你所工作的分支下分离并暴露出来。首先，需要将父提交迁出到 ImportantReminders 分支。为此，打开 Branches 对话框并单击 Checkout Tag or Revision，如图 7-43 所示。

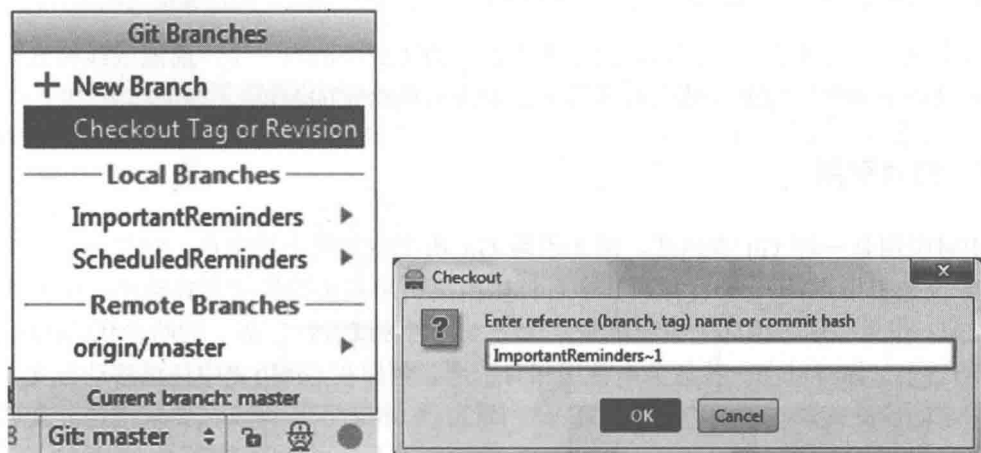


图 7-43 在 ImportantReminders 分支中迁出最后修改之前的修改

在 Checkout 提示框中输入 ImportantReminders~1。你现在将会处于分离模式，HEAD 分支以及项目状态将会体现出项目最初克隆时的最后一次提交，如图 7-44 所示。

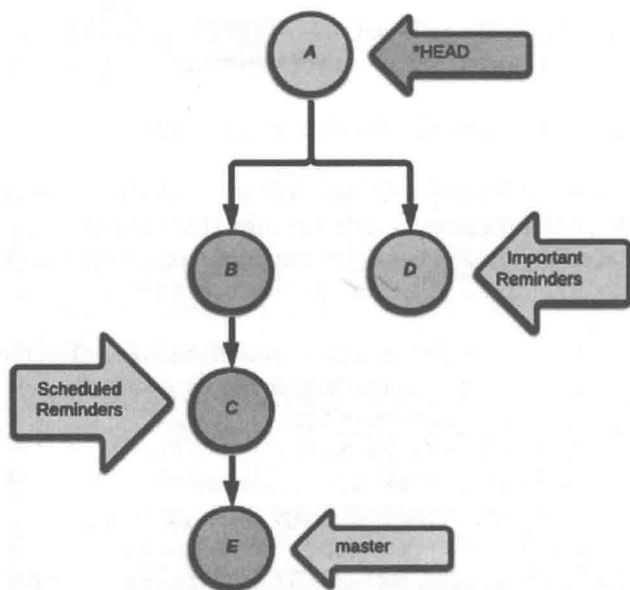


图 7-44 git_diagram8

注意 Git 现在暴露了一个新的 HEAD，它从开发过程中创建的所有分支中分离出来。HEAD 之前会跟随你迁出的任意分支。在做出提交之后，迁出分支将会随 HEAD 一起移至最后一次提交。你输入的 ImportantReminders~1 文本是一个相对引用，你想要从这里迁出并开始工作。在大多数要求输入分支或者提交哈希代码的操作中，你都可以提供相对引用。相对引用使用以下两种格式之一：

```
BranchOrCommitHash^
```

```
BranchOrCommitHash~NumberOfStepsBack
```

单个脱字符格式会从左侧指定的分支或提交的历史中回退一步，而波浪线格式会在历史中回退多个步骤，回退步骤的数量等于波浪线右侧给定的数值。

7.5.7 相对引用

相对引用是一种 Git 表达式，用于引用 Git 历史中的某个特定点。它们使用一个起始点(或者说是引用点)和一个目标，其中目标以距离引用点步数值的形式给出。由于通常使用 HEAD 作为引用，因此它也可能是某个分支的名称或者特定提交的哈希代码(或是缩略的哈希代码)。你可以在一些任务中使用相对引用，例如在 Git 历史中任意移动分支、选择某次特定的提交或者在历史中将 HEAD 移到特定点。相对引用可以在要求指定分支名称或者提交哈希代码的地方作为参数。我们已经在 IDE 中看到了使用它们的大量示例，它们最好与 Git 命令行配合使用。

创建用于开发下一个特性的新分支，并将它命名为 SetAlarm。创建一个对应于新分支的修改列表，同时删除所有旧的空修改列表。在 `com.apress.gerber.reminders` 包中添加一个名为 `RemindersAlarmReceiver` 的新类，并使用以下代码填充它：

```
public class ReminderAlarmReceiver extends BroadcastReceiver{
    public static final String REMINDER_TEXT = "REMINDER TEXT";

    @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
    @Override
    public void onReceive(Context context, Intent intent) {
        String reminderText = intent.getStringExtra(REMINDER_TEXT);
        Intent intentAction = new Intent(context, RemindersActivity.class);
        PendingIntent pi = PendingIntent.getActivity(context, 0,
            intentAction, 0);
        Notification notification = new Notification.Builder(context)
            .setSmallIcon(R.drawable.ic_launcher)
            .setTicker("Reminder!")
            .setWhen(new Date().getTime())
            .setContentText(reminderText)
            .setContentIntent(pi)
            .build();
        NotificationManager notificationManager = (NotificationManager)
            context.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(1, notification);
    }
}
```

在这里，我们有一个接收包含 Intent 附加值 REMINDER_TEXT 的 BroadcastReceiver。它使用文本并创建动作 Intent，该 Intent 用于构建发布在通知托盘中的通知。接着在 AndroidManifest.xml 中 activity 标签的后面、application 关闭标签之前添加以下内容，定义 BroadcastReceiver：


```
<receiver android:name="com.apress.gerber.reminders.  
ReminderAlarmReceiver"/>
```

按 Ctrl+K | Cmd+K 并提交 SetAlarm 修改列表，附带消息为 Adds BroadcastReceiver for alarms。你的 Git 历史记录将会如图 7-45 所示，起始点 A 的下面有了第三个提交。

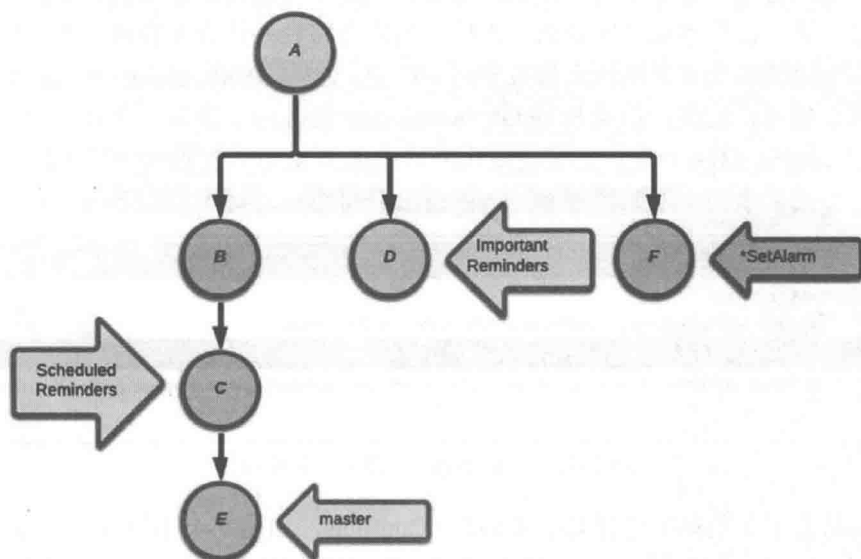


图 7-45 提交 SetAlarm 分支之后的 Git 历史

此特性自身不会做太多事情。它需要与 ScheduledReminders 特性合并，而该特性位于自己的分支中。为了完成自己的工作，你需要合并这两个特性并将它们推送到远程 Bitbucket 主机，但你想要以这种方式来完成——它看上去像是由一个人或一支团队在主分支上完成的，而且要清除所有其他分支。之前，你看到了 Git 合并的方式，它会创建一次拥有两个父级提交的提交，而这两个父级提交分别来自合并所涉及的分支。你还学到 Git 变基会以线性的方式合并两个分支，它有着单独的父级提交，而这正是你想要的。打开 Branches 对话框并迁出主分支。单击 File | VCS | Git | Rebase，选择 SetAlarm 为将要变到的分支，并取消选中 Interactive 复选框，因为你现在想要包含来自主干的所有修改。单击 Start Rebasing，你应该会看到如图 7-46 中所示的错误弹出框。

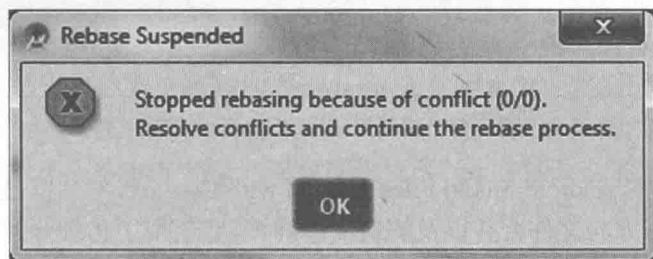


图 7-46 变基冲突弹出框

7.5.8 在变基时解决冲突

弹出框应该不会令你感到惊讶，因为它指出 Git 发现了一些冲突。Git 会将无法自动合并的文件标记为冲突状态。在变基可以继续之前，你需要解决这些冲突。传统上，解决冲突会导致大量的协同工作。当遇到错误或冲突时感到不舒服是很正常的，尤其是在合并操作时。不过，让自己熟悉这种不那么令人愉快的协作路径并让合并和解决冲突成为习惯，能够增长你在团队和个人之间协调修改方面的能力。此外，Android Studio 也让解决此类冲突变得不那么痛苦。记住，作为 ScheduledReminders 特性的一部分，你在主分支中开启了 BroadcastReceiver。这两个包含着类似或相同修改的分支中的代码成为产生冲突的原因。查看以红色突出显示的文件并找到 Changes 视图中的冲突，如图 7-47 所示。

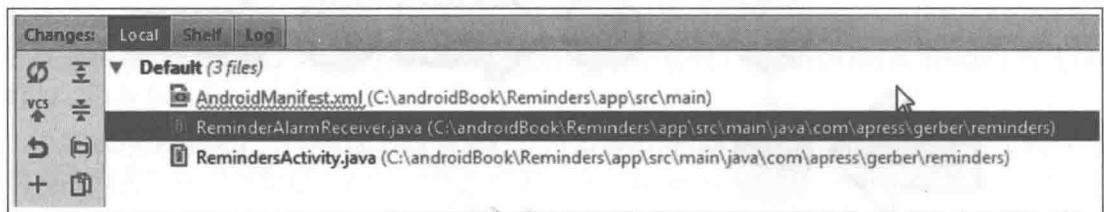


图 7-47 在 Changes 视图中合并冲突

右击并从上下文菜单中选择 Git | Resolve Conflicts，如图 7-48 所示。这将会打开 Files Merged with Conflicts 对话框。解决冲突通常涉及源代码的两个来源：你的本地修改(或者说你的)以及他们的外来修改(或者说他们的)。

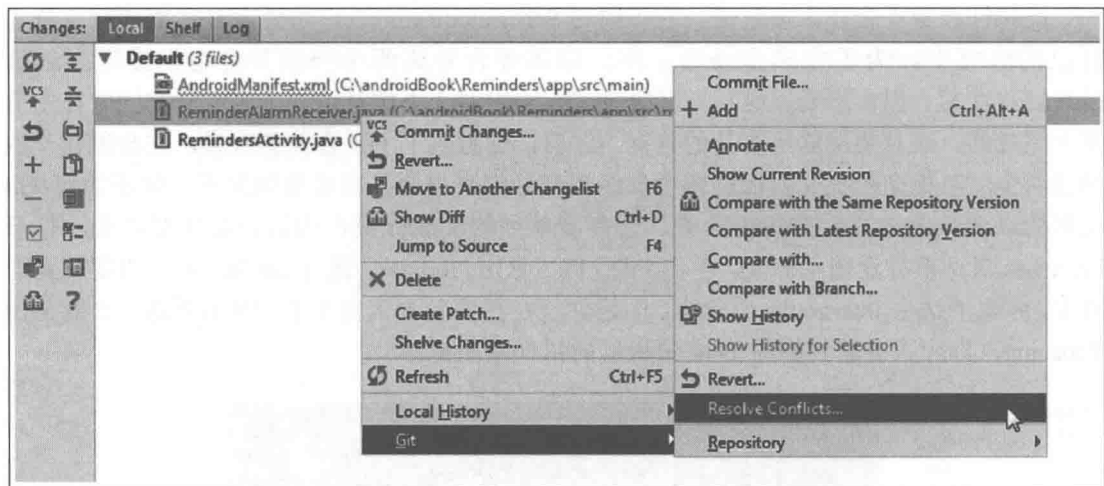


图 7-48 选择 Resolve Conflicts 选项

图 7-49 中所示的 Android Studio Files Merged with Conflicts 对话框是一款强大的合并工具，可以用于完成三方向文件合并以及解决文本冲突。它借用了传统合并场景中的术语(“你的”和“他们的”)并引导你完成合并。合并工具将你要变基到的 SetAlarm 分支视作“他们的”，或者说是外来服务器修改。你要变基的主分支认为是“你的”，或者说是本地工作副本。Files Merged with Conflicts 对话框允许你选择 Accept Yours、Accept Theirs 或 Merge。

Accept Yours 选项忽略要变基到的分支中的外来服务器文件修改，采用本地工作副本分支中的修改，并将文件标记为已解决。Accept Theirs 选项使用要变基到的分支中的外来服务器文件修改替换当前分支中的本地工作副本，同时将文件标记为已解决。Merge 选项会带你带入三方合并编辑器，你可以在这里将外来服务器和工作副本中的单个行修改拉取到基础合并副本中，同时自定义合并需要的内容。基础合并副本是合并的输出，或者说是结果。

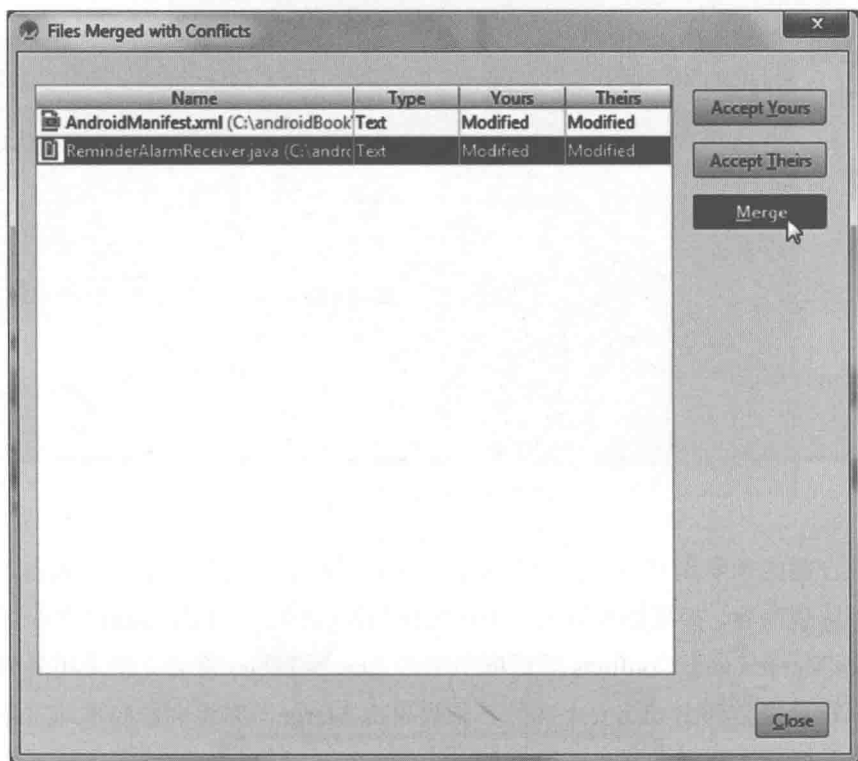


图 7-49 合并 ReminderAlarmReceiver

单击 Merge 按钮并观察其工作方式，图 7-50 中所示的 Merge 编辑器会打开。

Merge 编辑器将工作副本和外来副本组织在 Merge Result 的两侧，而 Merge Result 是屏幕中的可编辑部分。它能够感知语法和导入，这意味着你可以在编辑本地工作副本的时候使用自动完成、快速修复和其他键盘快捷键。这为你提供了外部 VCS 合并工具所不具备的某些高级特性。编辑器同时展示本地工作副本和外来更新，后者被标记为 Changes from Server。这些是来自你要变基到的 SetAlarm 分支的修改。在侧边栏中，你将会在已修改的行的旁边看到小小的双箭头和 X 标记。单击任意一侧的双箭头即可将其中的特定修改包含到合并结果中。单击 X 将会忽略对应的修改。这些修改也采用了颜色编码，红色表示冲突、绿色表示额外的行而蓝色表示已修改的行。在本例中，文件的大部分都处于冲突状态。

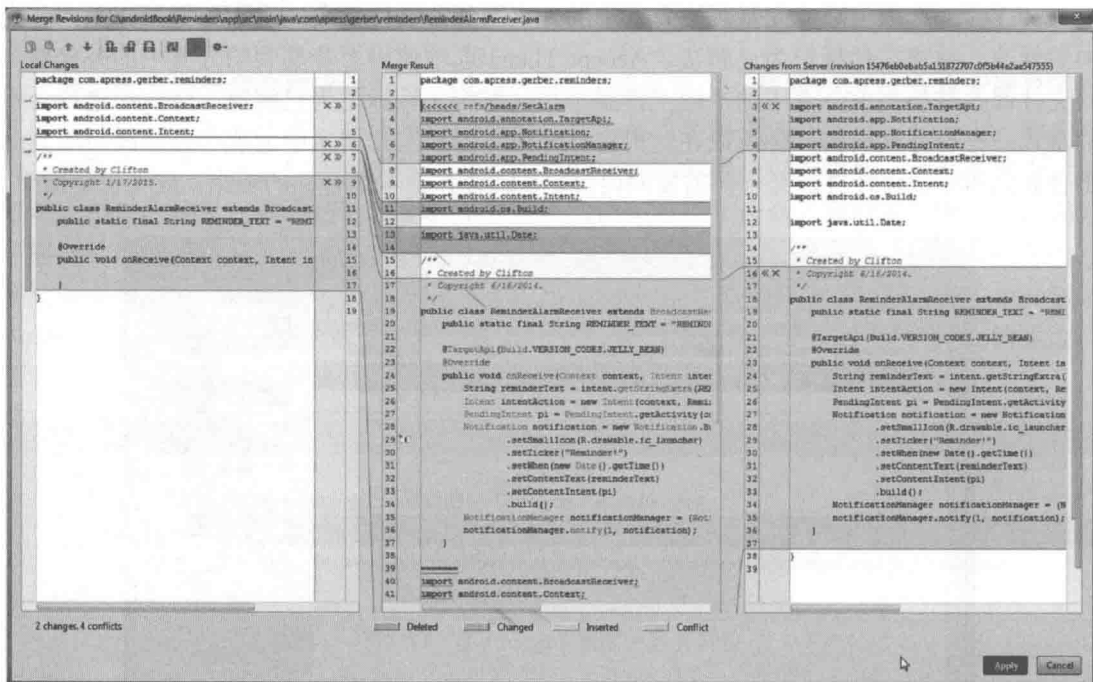


图 7-50 Merge 编辑器

由于在左侧的本地副本中仅有一个类的框架，因此合理的做法是接受来自右侧外来修改中的全部实现内容。如果想要退出且不应用任何修改的话，单击 **Cancel** 并在提示中回答 **Yes**。在 **Files Merged with Conflicts** 对话框中单击 **Accept Theirs** 将会完全采用外来服务器修改。对话框接着将会列出 **manifest** 文件。如果单击 **Merge**，你将会看到本地工作副本包含了与外来服务器副本相同的修改，因此可以选择 **Yours** 或 **Theirs**。单击本地工作副本中的双箭头以接受修改，并单击外来副本面板中的 **X** 来拒绝。在弹出的提示框中单击 **Save** 和 **Finish** 即完成了合并操作。两个文件在 **Git** 中均会被标记为冲突已解决。如果观察 **Changes** 工具窗口，你将会在 **Default** 修改列表中看到已合并的文件。**Git** 会在重放转到 **ScheduleAlarm** 分支的系列修改中间暂停并等待你继续。

访问主菜单并找到 **VCS | Git | Continue Rebasing** 选项，如图 7-51 所示。注意也可以选择取消变基或者在变基时跳过此次提交。如果正处在一次复杂合并的过程中而且意识到出了一些灾难性的错误，那么可以单击 **Abort Rebasing** 并让所有文件恢复到开始变基之前的状态。如果意外地包含了存在一些冲突的提交，那么也可以选择跳过。单击 **Continue Rebasing** 来完成变基操作。

变基将会完成，同时会执行一次新的提交。**Git** 历史将会在时间线中体现出主分支上 **SetAlarm** 提交之后的所有修改副本。这体现在了图 7-52 中。

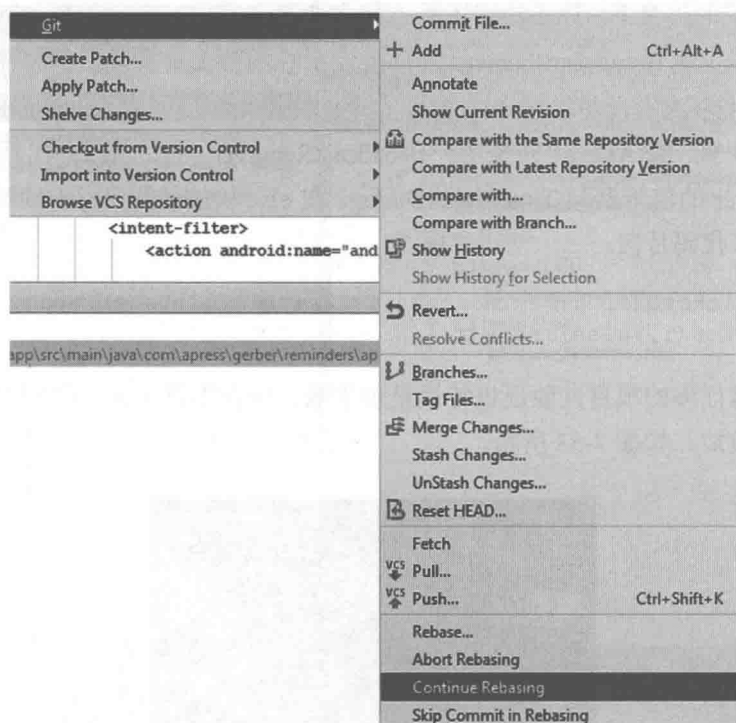


图 7-51 单击 Continue Rebasing 菜单项

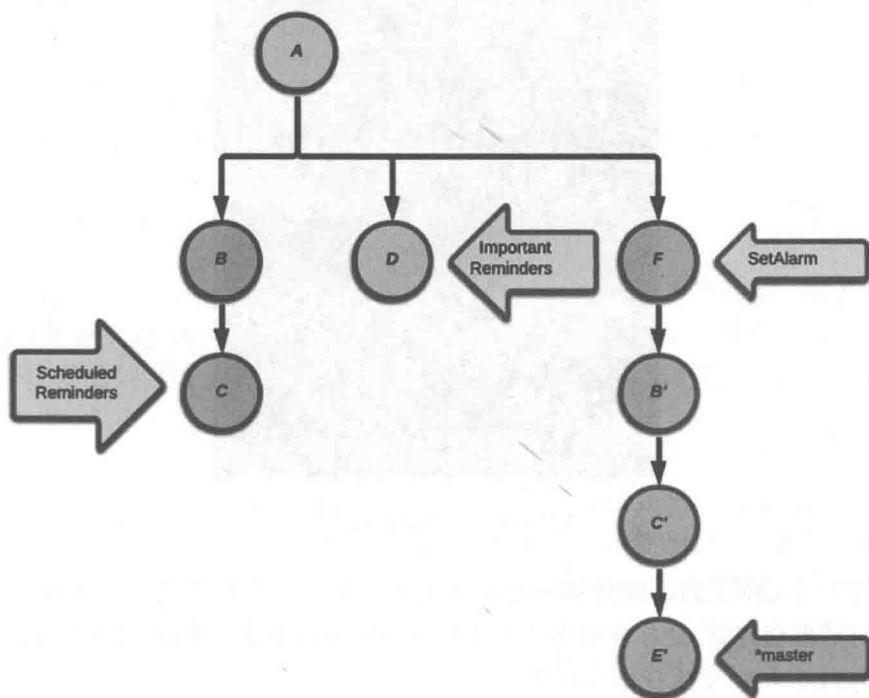


图 7-52 变基和解决冲突之后的 Git 历史

主分支包含提交 B 和 C(支持 ScheduledReminders)、提交 E(添加了 About 界面)以及来

自 SetAlarm 分支的提交 F。你还决定了不再包含变基之前的 ImportantReminders 特性。

设置闹钟和实现 `BroadcastReceiver` 的任务在单独的分支中实现，但它在你的时间线中看起来像是个标签(或者说是里程碑)。为了完成此特性，你需要将 `ScheduleReminders` 分支中的工作加入到 `SetAlarm` 分支的 `BroadcastReceiver` 中。修改代码来连接调用 `BroadcastReceiver` 的监听器与 `TimePickerDialog`。在 `else` 块的结尾、用于编辑提醒的对话框之前，插入以下代码片段：

```
new TimePickerDialog(RemindersActivity.this, listener, today.getHours(),
today.getMinutes(), false).show();
```

在设备上运行你的项目并验证该特性是否生效。现在当设置备忘时间的时候，你应该会获取到设备通知，如图 7-53 所示。

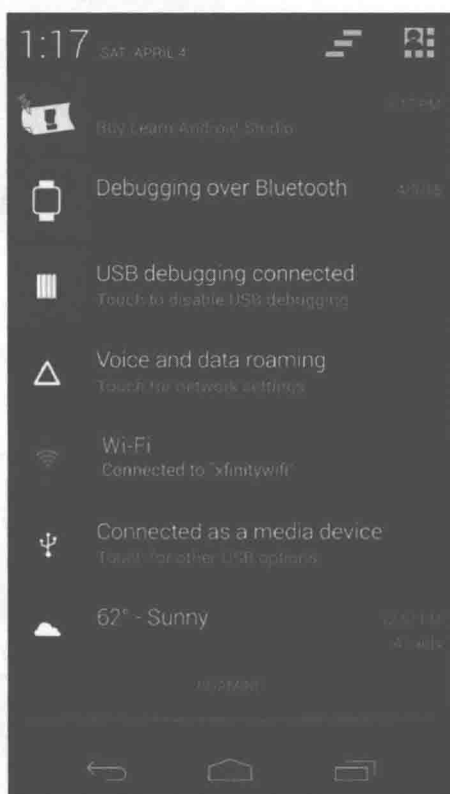


图 7-53 来自备忘的通知

可以将主分支推送到远程的 Bitbucket 主机。从 `File` 菜单中选择 `VCS | Git | Push`，图 7-54 中的对话框会打开，让你能够将本地主分支中的修改推送到 Bitbucket 仓库的远程主分支中。单击 `Push` 按钮来执行推送操作。

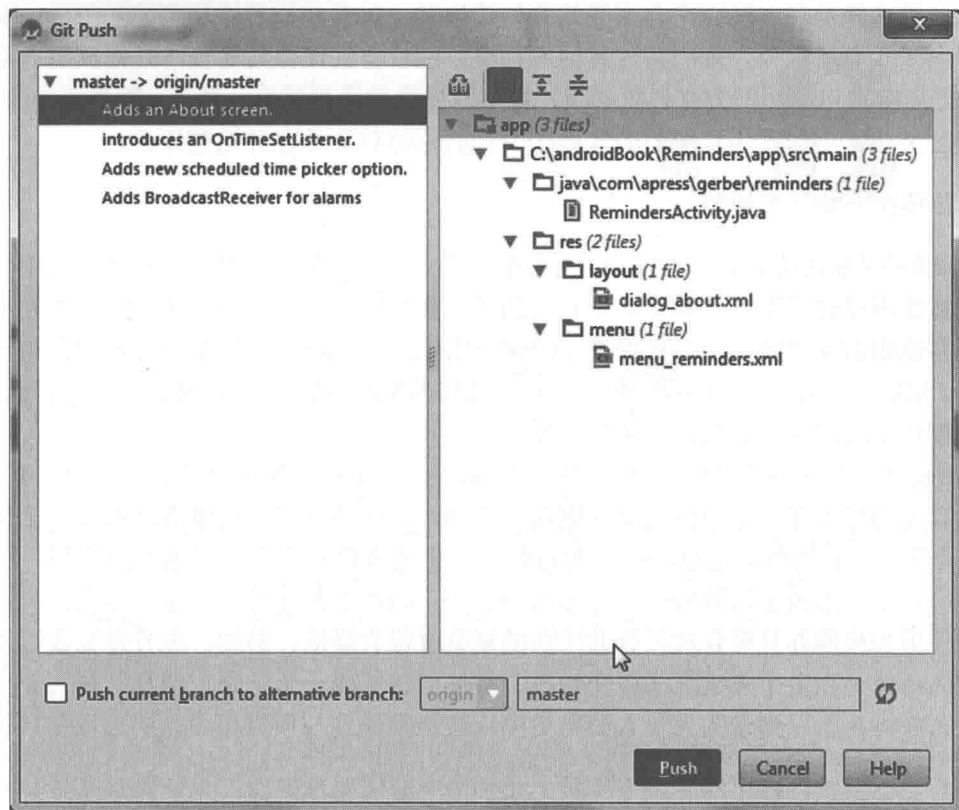


图 7-54 将你的修改推送到 Bitbucket

由于已经用完了 `ScheduledReminders` 和 `ImportantReminders` 分支，现在可以把它们删除了。打开 `Branches` 对话框并依次选中这两个分支；单击 `Delete` 将它们删除。

7.5.9 Git 远端

Git 远端就是 Git 仓库在远程服务器上的副本，它可以通过网络访问。由于可以像传统客户端/服务器模型的 VCS(类似 `Subversion`)那样使用它们，因此可以将它们认为是工作的公共可访问副本。在 Git 工作流中，你不向共享的中心服务器提交；相反，你通过拉取请求来共享工作。

拉取请求是单个开发者的请求，以其身份从公共仓库拉取修改。其他人可以根据需要自由地引入单次提交或者你的全部工作。你通常会找到一个主分支，由一个或多个带头的开发者负责保持该分支处于最新状态，其中包含了最新和最有价值的特性和提交。带头人通过使用 Git 的全部特性集拉取各个贡献者的修改，Git 的特性集合允许对单次提交进行选择、删除、重新排序、合并、挤压和修正。

然而，拉取请求是供高级 Git 用户使用的。人们开始使用 Git 的最常见方式是从 Git 服务器克隆项目——将 Git 仓库的整个副本下载到本地使用。可以继续在本地图出修改并提交它们，最后将这些修改推送回远程仓库。也可以获取并合并由其他人推送到远端的修改。

另一种选择是在本地以空仓库开始并构建项目。接着将项目推送到诸如 Bitbucket 或 GitHub 等 Git 主机服务，推广并与其他人共享，或者将其保留为私有，并根据需要邀请其他人。之后继续以常用的方式进行开发，将本地提交推送到远端。最后，随着工作的展开，贡献者分叉并将项目添加到他们的远程副本，而你将获取并合并这些修改。

“拉模型”与“推模型”

传统的 VCS 系统依赖于推模型，其中的特性由多个开发者完成，最后推送到中心服务器。虽然此模型已经工作了多年，但它受到诸多限制，当贡献者尝试通过使用差异和补丁文件合并他们的修改时，主分支的单个副本会出现问题。补丁文件是修改源文件动作的文本表示；例如，表示添加一些行、删除一些行或修改某些行。大多数 VCS 系统采用此模型，将修改视作在时间轴上应用的一系列差异。

Git 采用分布式的拉模型，将项目视作共享的实体。由于 Git 允许主分支的分布式副本，因此所有人均可以在任意时刻提交并更新本地副本，这减少了在贡献者之间合并工作所引入的复杂性。Git 还提高了单次提交的重要性，将其视作仓库在时间轴上的快照。这使得此工具可以更好地用于管理修改。它还增强了管理向单个源文件进行多次提交的灵活性。合并操作更加精确并且可管控，合并工作的复杂度极大降低。例如，项目带头人可以拉取你实现的特性、在多个分支之间提交修改、将他们挤压成一个、修正消息、在主分支的其他提交之前将其组织到带头人个人的历史中，最终推送到与项目相关联的远端并宣传它。

7.6 小结

这一章涵盖了在 Android Studio 中使用 Git 的基础知识。在本章，你看到了如何安装 Git 并使用它跟踪修改。我们演示了将源文件添加到 Git 以及使用 Git 日志特性查看提交历史汇总的方法。你已经看到了一些深入的示例，它们演示了分支如何像指向单次提交的标签一样工作。通过使用相对引用可以让这些分支在提交之间移动，甚至还可以将它们完全删除。我们已经演示了如何修改 Git 历史，改变同时提交的修改并将它们线性排列。我们还演示了一些协作场景，包括在多个分支上同时展开工作。

第 8 章

设计布局

发挥出 App 的最大效用通常意味着让其具有视觉吸引力，以取悦你的目标用户。虽然 Android 让起步变得很容易而且提供了多种模板项目，但有时你可能还需要进一步控制自己应用的界面外观。你可能想要把单选按钮的位置调整到另一个控件的旁边，或者可能需要创建自定义控件。本章涵盖了设计布局和组织控件的基础知识，让它们能够正确地展现在各式各样的 Android 设备上。

Android 布局基于三个核心的 Android 类：View、ViewGroup 和 Activity。当绘制界面时，这些是基础构建块。而用户界面中所含的大量类多数都派生自这些核心类且使用这些核心类，或者是它们的组件。另一个重要组件 Fragment 是在 Android 3.0 Honeycomb (API 11) 中引入的。Fragment 解决了设计模块化用户界面的核心需求，允许在多种外形尺寸的设备(尤其是平板电脑)上重用用户界面。本章首先介绍核心用户界面类，接着在后面的小节中讲解 Fragment。

8.1 Activity

Android Activity 表示用户可以与之交互的界面。Activity 类自身并不绘制任何东西；它只是一个根容器，负责组织将要被绘制的每一个组件。所有绘制到屏幕上的组件均在 Activity 的边界之内。Activity 类也用于响应用户输入。随着用户在不同界面之间浏览，Activity 也会进行切换。Activity 有着很容易理解的生命周期，表 8-1 中做了详细说明。本章后面还会引用 Activity 的生命周期。

表 8-1 Activity 生命周期方法

方 法	描 述	调用后是否结束	下一个
onCreate()	在 Activity 最初创建时回调。它负责创建视图，将数据绑定到控件以及管理 bundle 或者从中恢复状态	否	onStart()

(续表)

方 法	描 述	调用后是否结束	下一个
onRestart()	在 Activity 被停止之后、马上要再次启动之前会调用此方法。这会在一些情况下发生，例如电话通话结束或让 App 返回前台时	否	onStart()
onStart()	Activity 即将显示到屏幕上之前会调用此方法。如果 Activity 回到前台，则随后调用 onResume(); 如果 Activity 隐藏，则随后调用 onStop()	否	onResume() 或 onStop()
onResume()	当 Activity 创建、开始以及准备好接收用户输入时会调用 onResume()方法。Activity 会在这个方法完成之后开始运行	否	onPause()
onPause()	当系统准备好恢复 Activity 时调用此方法。当前 Activity 还在执行，但系统准备切换到另一个 Activity 时会调用此方法；或者在当前 Activity 被打断并被置于后台时调用	是	onStop() 或 onResume()
onStop()	当 Activity 不再可见时调用此方法	是	onRestart() 或 onDestroy()
onDestroy()	Activity 在即将销毁之前会接收到此调用。这通常是在 Activity 中直接调用 finish()的结果，还有一种情形就是 WatchDog 需要杀掉 Activity 来回收内存或者 Activity 失去了响应。这是 Activity 将会收到的最后一个调用	是	无

8.2 View 和 ViewGroup

尽管 Activity 是根元素，但它通常包含多个 View 和 ViewGroup 对象。View 是屏幕上所有的可见组件(包括视图组合)的超类，这些组件的例子有按钮、文本框、文本输入控件和复选框等元素。视图中通常包含一个或多个视图组合。视图组合表示一个或更多视图对象的集合。视图组合可以通过多级嵌套来创建复杂的布局。视图组合的主要职责是控制一个或多个内嵌 View 或 ViewGroup 对象的布局。多种不同类型的视图组合会控制其子组件的位置。

有多种布局容器对象。每个布局对象有着不同的行为并使用特有的位置属性。LinearLayout、RelativeLayout、FrameLayout、TableLayout 和 GridLayout 是核心的布局容器。

为更好地理解每个布局的工作原理,让我们看几个例子。使用 New Project Wizard 启动一个名为 SimpleLayouts 的新项目。选择 Phone and Tablet 外形参数,设置最低 API 为 14(IceCreamSandwich),并使用 Blank Activity 模板。保留默认名称为 MainActivity,Layout Name 框中的内容为 activity_main.xml,接着继续创建项目。你应该会进入主 Activity 布局的编辑模式,如图 8-1 所示。

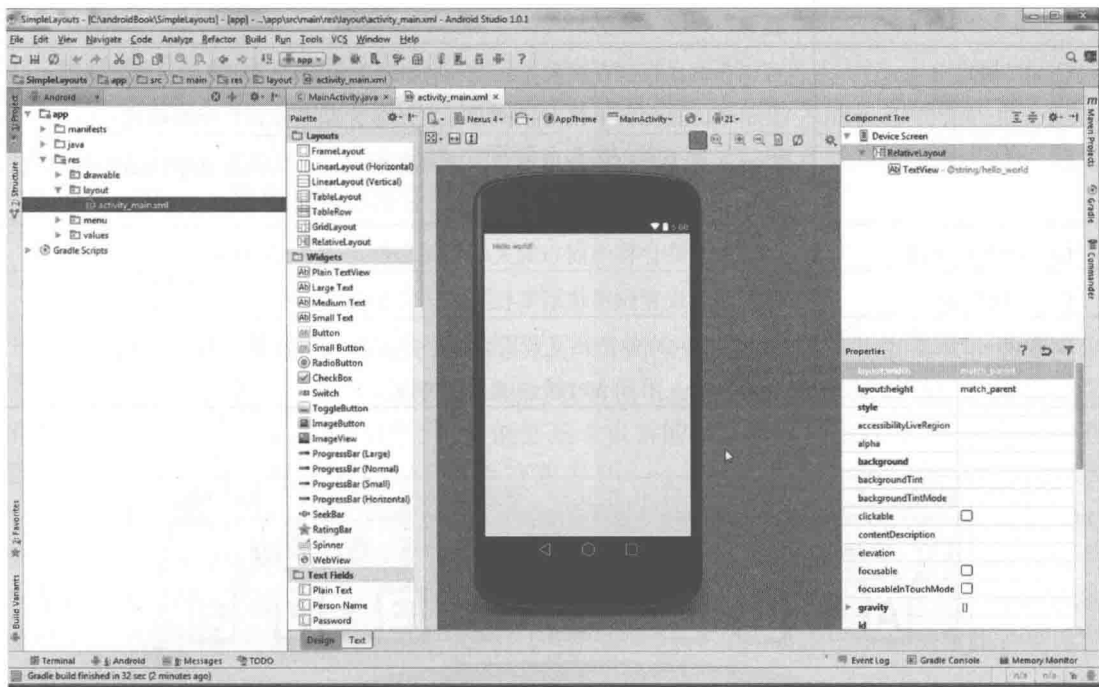


图 8-1 从主 Activity 的布局开始

8.2.1 预览面板

在新项目中,可以从文本编辑模式开始设计主 Activity 的布局 XML。如果项目未处于这个模式,按 `Ctrl+Shift+N` | `Cmd+Shift+O` 打开 File Search 对话框,输入名称 `activity_main` 来查找主布局。设计布局时,Android Studio 同时支持文本模式和设计模式,你应该熟练掌握它们。使用 Editor 窗口左下方的选项卡可在这些模式之间切换。默认的文本模式允许你像处理所有其他源文件那样直接编辑 XML 文件。

预览面板位于 Editor 的右侧,让你能够在进行修改的过程中实时地看到布局的样子。也可以通过在 Configuration Render 菜单中选择 Preview All Screen Sizes 选项,来预览布局在多种设备上的外观。Virtual Device 下拉菜单中有着相同的可用选项。这两个菜单都位于预览面板的左上角。可以切换预览选项的开和关并观察其工作方式。

预览面板的顶部有一些控件,允许你修改预览的渲染方式。可以在定义了 AVD 的任意设备上渲染预览效果。可以同时预览多个设备。也可以修改用于渲染预览界面的 API 级别和主题。表 8-2 描述了预览面板中的标记区域,它们突出显示在图 8-2 中。

表 8-2 预览面板的描述

区 域	描 述
A: 预览开关	这是预览开关，它拥有选择特定 Android 版本或者选择所有屏幕尺寸的选项，可以用来基于当前布局为特定屏幕尺寸快速创建布局
B: AVD 渲染	此菜单允许你在特定设备上预览布局。它还可以用作切换所有屏幕尺寸的菜单
C: UI 模式	可以在这里找到在横向、纵向和各种 UI 模式之间切换预览器的选项，以及车辆、桌面和电视对接模式。它还包含家电模式和夜间模式
D: 主题控制	Theme 开关允许你使用特定主题预览布局。它默认为 AppTheme，但可以从 SDK 或项目中选择各种主题
E: Activity 关联	此菜单允许你将当前布局关联到某个特定的 Activity
F: 区域控制	这个菜单设置预览使用某种特定语言的译文
G: Android 版本	API 菜单允许你把预览设置为某个特定的 API 级别。可以使用这个特性来观察布局在不同 API 级别的显示情况

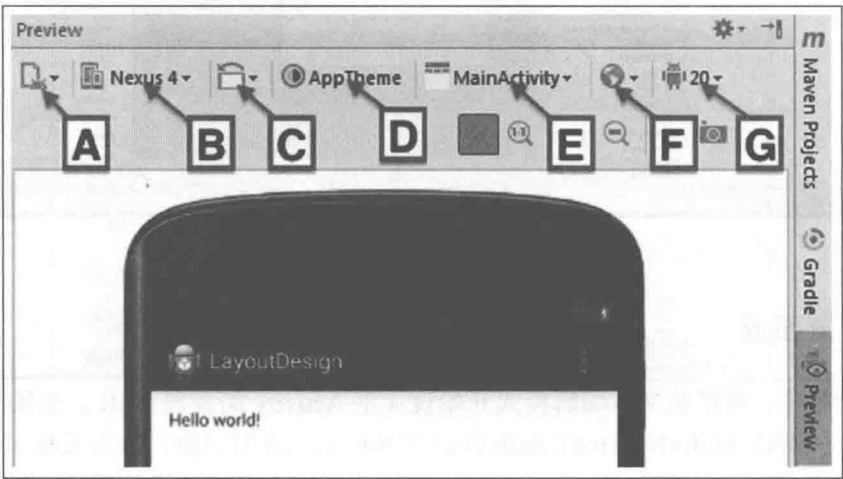


图 8-2 详细的预览面板

在文本模式中，选择 RelativeLayout 标签并将其起始和结束标签设置为 FrameLayout。注意，预览面板中没有发生任何改变，因为只是改变了根布局标签，但尚未触及里面的任何内容。稍后你将会学到关于这些布局之间差异的更多知识。

在内嵌的 TextView 中选择“Hello World”文本，它将会自动扩展为“@string/hello_world”，这是一个指向外部 strings.xml 文件中文本的引用。Android Studio 的代码折叠特性会默认隐藏外部字符串引用。按 Ctrl + - | Cmd + - 可以折叠(或者说收起)属性的呈现形式；按 Ctrl + = | Cmd + = 展开它，即可看到实际属性值。在 Android 布局中硬编码字符串值被认为是坏的实践，使用字符串引用是更好的处理方法。在诸如这里创建的简单示例中，硬编码字符串并不会有什么问题，但商业 App 可能需要支持多种语言，而外部字符串可以简

化这个过程。因此，养成使用外部字符串的习惯是一个好主意。

引用是资源文件中编码的特殊属性值，它指向在其他地方定义的实际值。在本例中，特殊字符串“@string/hello_world”引用在 strings.xml 资源文件中定义的一个值。按住 Ctrl(或 Cmd)键并单击文本，导航到“Hello World”字符串定义，它看上去应该类似于以下这样：

```
<string name="hello_world">Hello world!</string>
```

将值修改为“Hello Android Studio!”。按下 Ctrl + Alt + 向左箭头 | Cmd + Alt + 向左箭头导航回到布局，观察预览面板中已更新的值。现在将文本修改为随机的硬编码值，例如“Goodbye, Las Vegas!”，预览将会再次更新；但这种情况下，你直接重写了字符串。随着你修改 TextView，预览面板也将会更新。

8.2.2 宽度和高度

文本视图是可以添加到布局的多种视图之一。每个视图均有用于控制其大小的宽度和高度属性。可以设置绝对像素值(例如 250px)或者使用一种相对值(例如 250dp)。最好使用带有 dp 后缀的相对值，因为这使得组件能够基于设备的像素密度调整尺寸。相对尺寸会在后面的 8.3.1 小节中介绍。将 TextView 标签改为 Button 标签，并将 android:layout_width 属性改为 match_parent。文本视图将会变成一个按钮并延伸至整个屏幕的长度。将 android:layout_height 属性改为 match_parent。这个按钮将会占据整个屏幕。将 android:layout_width 属性改为 wrap_content，按钮宽度将变窄，但仍占据整个屏幕高度。match_parent 是一个特殊的相对值，它基于视图的父容器来确定其尺寸。图 8-3 展示了将 match_parent 用于组件宽度和/或高度的可能变化。wrap_content 是另一个广泛使用的相对值，它会调整视图的大小，使其紧紧地包裹住内容。将 Button 标签改为 TextView 标签，将其宽度和高度设置为 match_parent 并向布局中添加一些其他组件，如 Button 和 CheckBox，参见代码清单 8-1 中的定义。

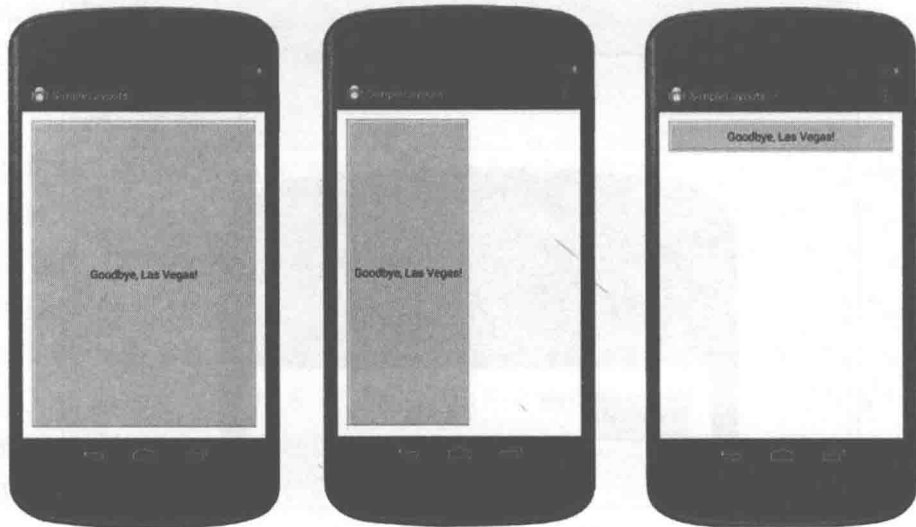


图 8-3 match_parent 大小值的差异

代码清单 8-1 向布局中添加更多组件

```

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:text="Goodbye, Las Vegas!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <Button
        android:text="Push Me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <CheckBox
        android:text="Click Me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</FrameLayout>

```

注意这些组件全部重叠在了一起。图 8-4 展示了这个问题。FrameLayout 的行为是按照组件定义的顺序来堆叠它们。现在删除额外的堆叠组件，以便可以探索设计器模式并理解如何可视化地摆放组件。

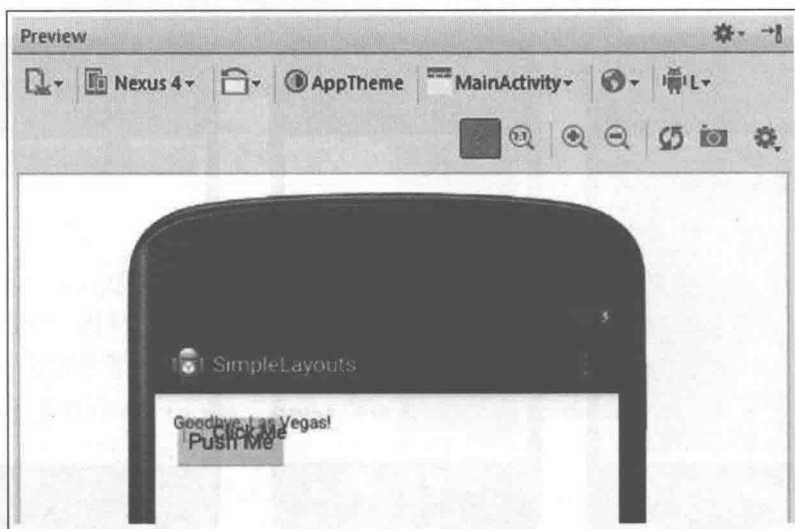


图 8-4 组件被堆叠在一起

让我们研究一下 `FrameLayout` 容器标签。这个标签定义了两个属性——`android:layout_width` 和 `android:layout_height`，均指定为 `match_parent`。这意味着帧的宽度和高度将与包含它的父组件的宽度和高度一致。由于 `FrameLayout` 在最外层(或者说是根元素)，因此它是所有其他组件的父组件。因此，它的宽度和高度将覆盖设备屏幕的整个可视化区域。

8.2.3 设计器模式

单击 Editor 左下方的 Design 选项卡(如图 8-5 所示)开启设计模式。在这一节中，你要探索如何使用 Visual Designer 来放置控件。

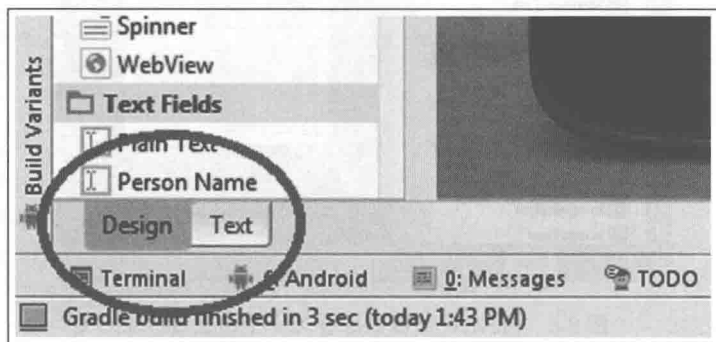


图 8-5 设计器和文本视图选项卡

与文本模式相同，设计模式有着相同的实时预览面板，但增加了一个组件调板。当以可视化方式设计布局时，可将组件从调板拖放到预览面板中。可视化设计器会自动生成 XML，同时让你专注于布局的界面外观。设计模式还配有右上角的组件树面板以及下方的属性面板。组件树给出了当前布局中所有视图和视图组合组件的层次视图。顶部为根组件，在本例中为 `FrameLayout`。

8.2.4 帧布局

如你所见，`FrameLayout` 按照组件定义的顺序来堆叠它们。不过，它还将屏幕分为 9 个特殊区域。在组件树中单击 `TextView` 并按 `Delete` 键删除它。以同样的方法删除 `Checkbox` 和 `Button` 组件，彻底清空界面。在左侧调板中找到 `Button` 组件并单击它。在预览面板上移动鼠标并注意鼠标移过时突出显示的部分。屏幕被划分为多个区域，由特定的 `FrameLayout` 区域来标识(参见图 8-6)。在左上区域单击，放下按钮。双击按钮并将其文本改为 `Top Left`，用以标识其位置。继续在其他 8 个区域拖放组件并相应地标记它们。当拖放每个按钮时，在文本模式和设计模式之间来回切换，查看生成的 XML。当完成时，布局应该如图 8-7 所示。创建此布局的代码请参见代码清单 8-2。

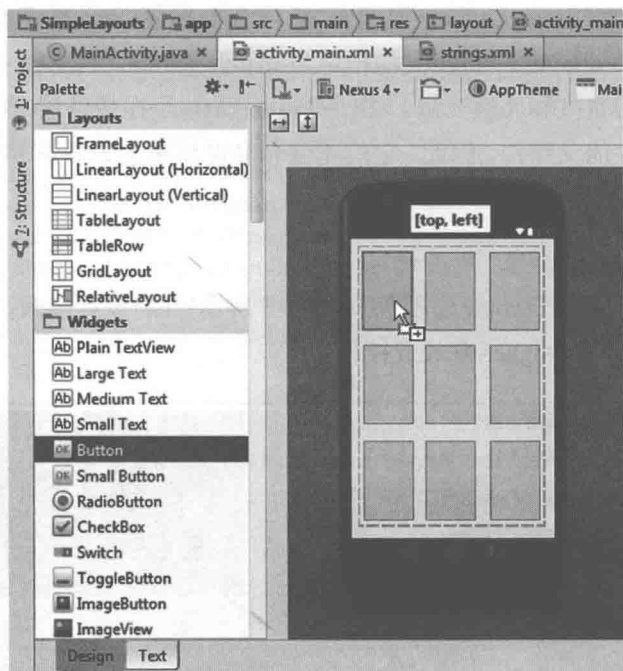


图 8-6 Preview 面板被划分为 9 个可放置区域

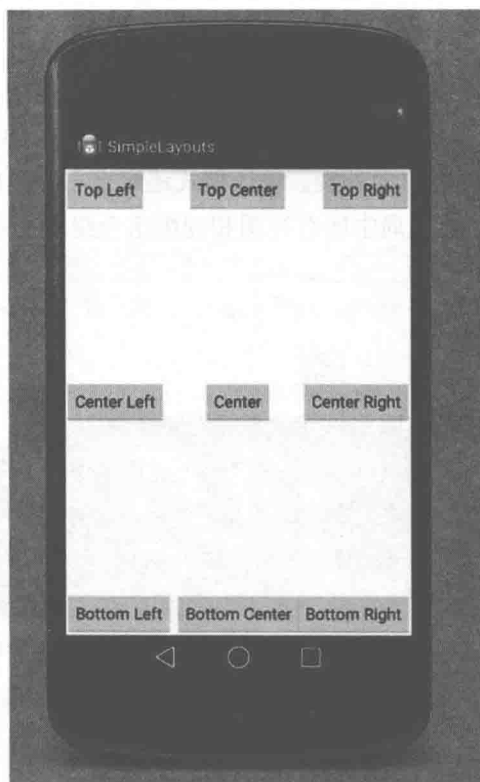


图 8-7 FrameLayout 布局演示

代码清单 8-2 创建出图 8-7 所示布局的代码

```

<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Center"
        android:id="@+id/button"
        android:layout_gravity="center" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:id="@+id/button2"
        android:layout_gravity="left|top" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Center"
        android:id="@+id/button3"
        android:layout_gravity="center_horizontal|top" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:id="@+id/button4"
        android:layout_gravity="right|top" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Center Left"
        android:id="@+id/button5"
        android:layout_gravity="center|left" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Center Right"

```

```

        android:id="@+id/button6"
        android:layout_gravity="center|right" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom Left"
    android:id="@+id/button7"
    android:layout_gravity="bottom|left" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom Center"
    android:id="@+id/button8"
    android:layout_gravity="bottom|center" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom Right"
    android:id="@+id/button9"
    android:layout_gravity="bottom|right" />
</FrameLayout>

```

设计器生成此 XML，它以 `FrameLayout` 标签开头。它的宽度和高度被设置为占满屏幕的整个可见区域。每个内嵌按钮均指定了 `layout_gravity`，它用于确定按钮落入屏幕的哪个区域。

8.2.5 线性布局

`LinearLayout` 在水平或竖直方向彼此相邻地组织其子组件。打开左侧的项目面板，找到 `res` 下的 `layout` 目录并右击打开上下文菜单。单击 `New | XML | XML Layout File`，创建一个新的布局资源文件，并将其命名为 `three_button`。在预览面板中单击并放入三个按钮，每一个均在上一按钮的下方。你的布局看上去应该类似于图 8-8 的左侧。在预览面板的左上方，单击 `Convert Orientation` 按钮(在第二行按钮中)。屏幕上的按钮将会从纵向排列切换为横向排列，如图 8-8 中的右侧图片所示。

下面这段 XML(如代码清单 8-3 所示)指定方向属性的 `LinearLayout` 根标签作为开始。方向可以设置为竖直或水平。`LinearLayout` 中的内嵌 `Button` 标签从上到下或者从左到右排列，而这取决于方向。

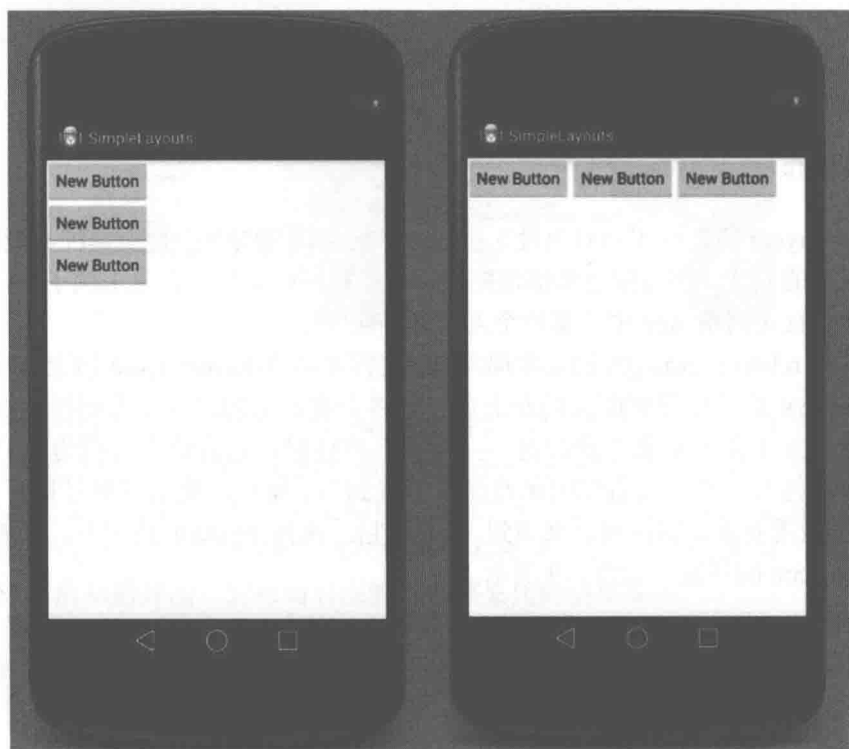


图 8-8 竖直 LinearLayout 与水平 LinearLayout

代码清单 8-3 三按钮的 LinearLayout 示例

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```

        android:text="New Button"
        android:id="@+id/button3" />
    </LinearLayout>

```

8.2.6 相对布局

`RelativeLayout` 通过使用相对属性来组织其子元素的位置。当使用这种类型的布局时，可创建更复杂的设计，因为能更精细地控制每个子视图的位置。在这个例子中，假定你要创建类似于社交网络 App 中的那种个人信息界面。

创建名为 `relative_example` 的新布局 XML 文件并将 `RelativeLayout` 指定为根元素。将一个 `ImageView` 拖放至预览面板的左上角。你将会在拖动的过程中看到提示线，而且它应该会吸附在左上角。不要感到惊讶——当放下控件时，它会消失，因为我们还没有为其指定维度和内容。在屏幕右侧的属性面板中找到 `src` 属性，单击省略号打开 `Resources` 对话框(你可能需要滚动属性列表来找到 `src` 属性)。选择 `System` 选项卡，接着选择名为 `sym_def_app_icon` 的资源，如图 8-9 所示。

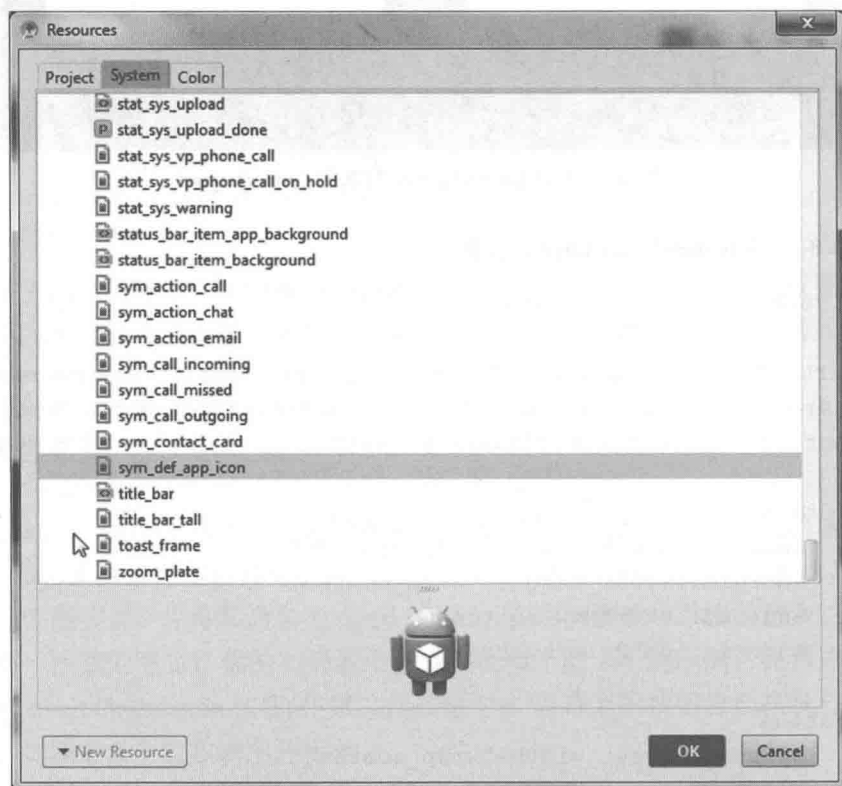


图 8-9 选择 `sym_def_app_icon`

该图标将显示在这个已添加到预览面板的 `ImageView` 中。从调板中单击 `PlainTextView`，接着单击 `ImageView` 的右上方，将 `PlainTextView` 放在此组件的右侧并与其父组件的顶部对齐。当你在图像右侧边缘附近移动鼠标时，将会有有一个指示当前放置位置的工具提示出现。微调位置直至工具提示显示 `toRightOf=imageView` 和 `alignParentTop`，如图 8-10 所示。

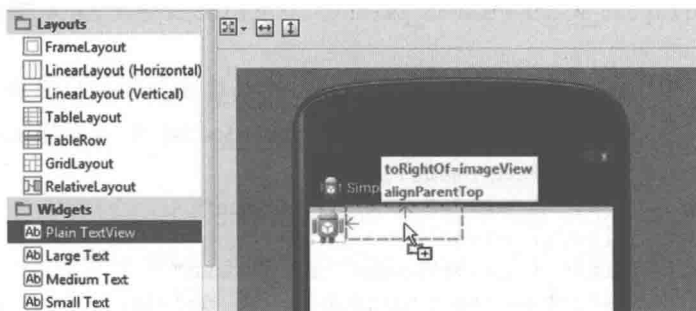


图 8-10 在视图周围移动时显示的工具提示

再向预览面板中拖曳两个 PlainTextView 组件,将每一个置于前一个的下方、ImageView 的右侧。使用提示线来帮助你。双击顶部的 TextView 并将其文本修改为一个姓名。将中间的 TextView 的文本修改为一座著名的城市。最后,修改底部的 TextView 的文本,令其包含一个网址。在设计视图的过程中,来回切换文本视图,查看生成的 XML。你应该会看到类似于图 8-11 所示的界面。此布局对应的代码参见代码清单 8-4。



图 8-11 用于个人信息的相对布局

代码清单 8-4 图 8-11 中布局对应的代码

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

```

        android:layout_width="match_parent" android:layout_height="match_parent">

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imageView"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:src="@android:drawable/sym_def_app_icon" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Clifton Craig"
            android:id="@+id/textView1"
            android:layout_alignParentTop="true"
            android:layout_toRightOf="@+id/imageView" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="California"
            android:id="@+id/textView2"
            android:layout_below="@+id/textView1"
            android:layout_toRightOf="@+id/imageView" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="http://codeforfun.wordpress.com"
            android:id="@+id/textView3"
            android:layout_below="@+id/textView2"
            android:layout_toRightOf="@+id/imageView" />

    </RelativeLayout>

```

生成的 XML 将 `RelativeLayout` 作为其根元素。这个布局包含一个 `ImageView`，它有两个特性：`layout_alignParentTop` 和 `layout_alignParentLeft`。这些特性会让 `ImageView` 保持在布局的左上角。`layout_alignParentStart` 特性用于支持自右向左的语言，以防止发生歧义。`ImageView` 也采用与之前相同的高度和宽度特性。最后，它定义 `src` 特性指向 `sym_def_app_icon` 资源，这是由 Android 运行时预定义的内置资源。

每个组件均包含一个 `android:id` 特性，其值以 `@+id/` 开头。这些 ID 特性提供了一种在运行时定位单个组件的方法。它们对于 `RelativeLayout` 来说很重要，因为需要用它们来指定组件的相对位置。注意其余 `TextView` 组件在 `layout_below` 和 `layout_toRightOf` 特性中使用这些值的方法。它们都指定了 `layout_toRightOf=@+id/imageView`，这会直接将它们置于图像视图的右侧边缘。最后两个 `TextView` 组件指定 `layout_below` 特性，它指向前面那个 `TextView`。

8.2.7 嵌套布局

通过布局的嵌套可以创造出复杂的设计。如果想要美化之前的个人信息界面，可以采用将 `LinearLayout` 嵌入到 `RelativeLayout` 中的方法。这个布局包含一个在线状态标签和一个描述字段。

单击调板中的竖直 `LinearLayout`，然后在预览面板中 `ImageView` 的下方单击并放置它。确保工具提示显示 `alignParentLeft` 和 `below=imageView`。单击调板中的 `Plain TextView`，接着在新添加的 `LinearLayout` 内部单击并放置此组件。这将是你的在线状态标识符。接着找到 `Large Text` 组件；在调板中单击它，这次在右侧组件树中找到另一个新添加的 `TextView`，在其下方单击并放置组件。当把鼠标移到 `LinearLayout` 中 `TextView` 的下方时，将出现一个较粗的放置目标标识符，如图 8-12 所示。

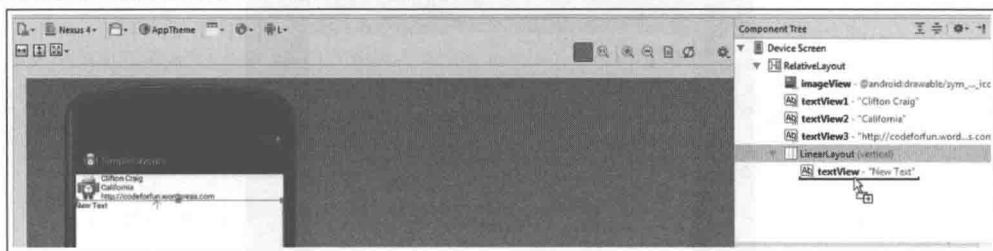


图 8-12 将鼠标移至 `TextView` 下方时看到放置目标标识符，单击并添加组件

使用属性面板，将第一个 `TextView` 的文本属性修改为 `online` 并为下方 `TextView` 的文本属性添加描述。接下来在预览面板的任意位置单击并按下 `Ctrl + A` | `Cmd + A` 以选中所有组件。找到 `layout:margin` 属性，展开它，将所有值设置为 `5dp`，让每个组件有 5 像素的外边距，如图 8-13 所示。

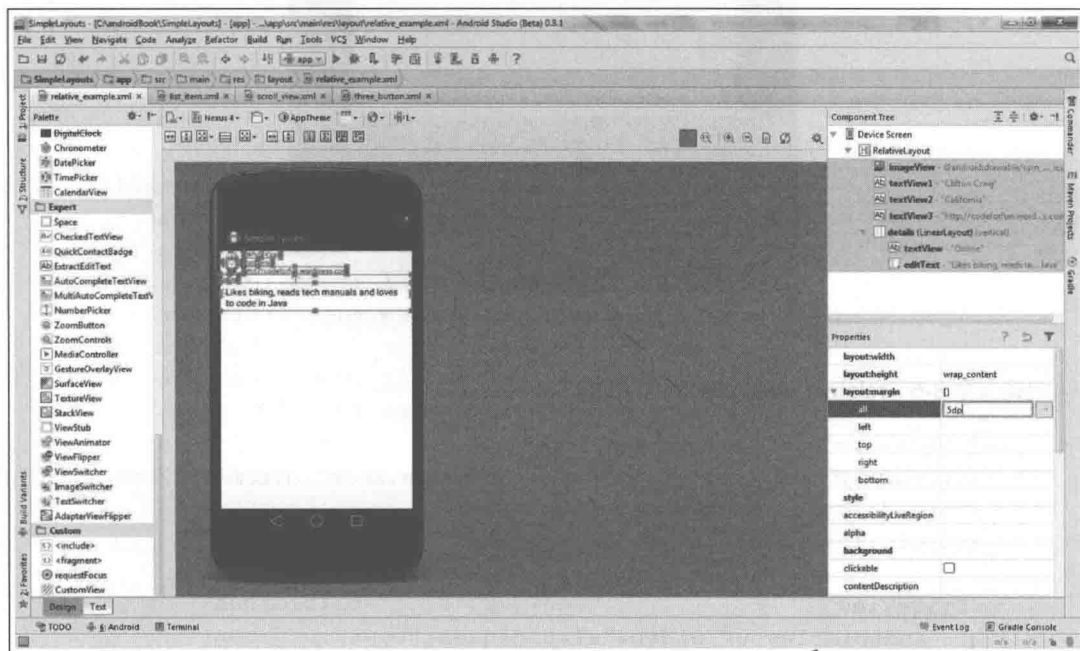


图 8-13 为所有组件指定 5 像素外边距

外边距控制组件的边与任意相邻组件之间的空白大小。为每个组件设置外边框是一种避免出现界面混乱的好方法。虽然我们为所有组件设置了相同的外边距，但也可以在某些边上实验并设置不同的外边距。

`layout:margin` 组包含用于 4 条边界的设置：左边距、上边距、右边距和下边距。再次选择所有组件，展开 `layout:margin` 设置并选择 All 选项。删除 5dp 这个值，转而将左侧外边距设置为 5dp。这些组件将会紧密地聚合在一起，但是左侧外边距刚好留出了足够的空间。选择文本内容为 online 的 `TextView`，将其顶部外边距设置为 5dp，让它和上方图像之间有更大的空间。图 8-14 展示了此时的界面。代码清单 8-5 呈现了此布局对应的代码。

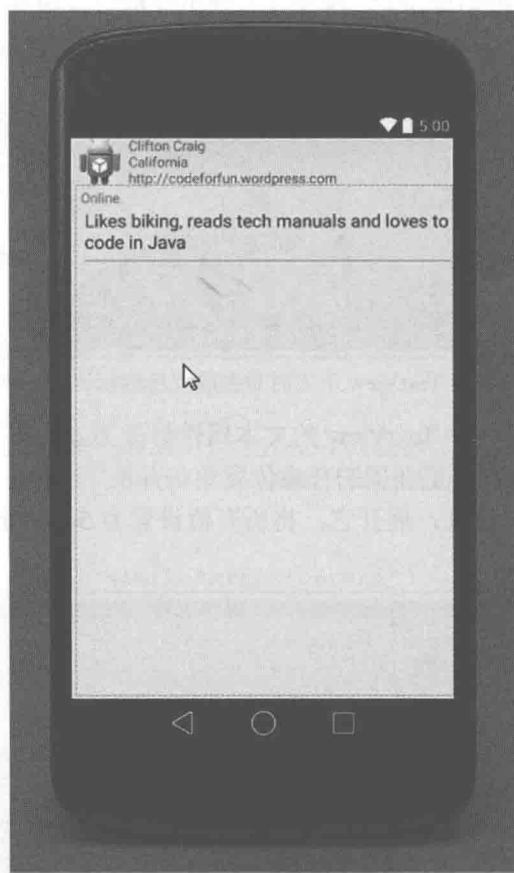


图 8-14 添加左侧和顶部外边距后的效果

代码清单 8-5 `relative_example.xml` 的代码

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

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```



```

    android:id="@+id/imageView"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:src="@android:drawable/sym_def_app_icon"
    android:layout_marginLeft="5dp" />

```

```
<TextView
```

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Clifton Craig"
    android:id="@+id/textView1"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/imageView"
    android:layout_marginLeft="5dp" />

```

```
<TextView
```

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="California"
    android:id="@+id/textView2"
    android:layout_below="@+id/textView1"
    android:layout_toRightOf="@+id/imageView"
    android:layout_marginLeft="5dp" />

```

```
<TextView
```

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="http://codeforfun.wordpress.com"
    android:id="@+id/textView3"
    android:layout_below="@+id/textView2"
    android:layout_toRightOf="@+id/imageView"
    android:layout_marginLeft="5dp" />

```

```
<LinearLayout
```

```

    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/imageView"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="5dp">

```

```
<TextView
```

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Online"
    android:id="@+id/textView"
    android:layout_marginLeft="5dp"
    android:layout_marginTop="5dp" />

```

```

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:text="Likes biking, reads tech manuals and loves to code
in Java"
    android:layout_marginLeft="5dp" />

</LinearLayout>

</RelativeLayout>

```

另一种嵌套布局的方法是使用 `include` 间接引用它们。找到 `LinearLayout` 并修改其特性，加入一个值为 `details` 的 `id` 特性，并确保将其高度设置为 `wrap_content`。同时修改并设置 `layout_below` 特性，使其下移到 `textView3` 的底部。这体现在以下代码中：

```

<LinearLayout
    android:id="@+id/details"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView3"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="5dp">

```

接下来在最后一个 `TextView` 标签之后，在 `LinearLayout` 结束标签之前添加以下内容：

```

<include layout="@layout/three_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/details"/>

```

这个特定的 `include` 标签可将任意预先定义好的布局添加到当前布局中。在前一个例子中，我们在当前布局中引入了之前创建的三个按钮示例。将宽度声明为 `match_parent` (扩展了布局的整体宽度)，将高度设置为 `wrap_content`。再将按钮布局设置在 `details` 组件下方 (`details` 是相对布局的名称)。

`Ctrl+单击` | `Cmd+单击` 布局特性值 `@layout/three_button`，导航到它的定义。在定义内部，修改每个按钮的文本，使其对应于社交网络 App 中可用的典型操作。按照顺序将每个按钮的文本特性修改为 `Add Friend`、`Follow` 和 `Message`。在文本或设计模式中都可以完成这件事。图 8-15 展示了它在设计模式中的样子。

完成后，回到 `relative_example.xml` 并查看已经集成好的按钮。图 8-16 展示了完成后的样子。

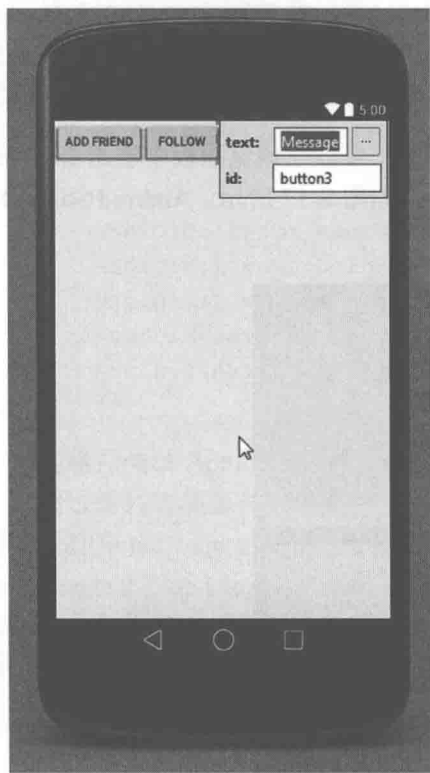


图 8-15 为按钮添加标签

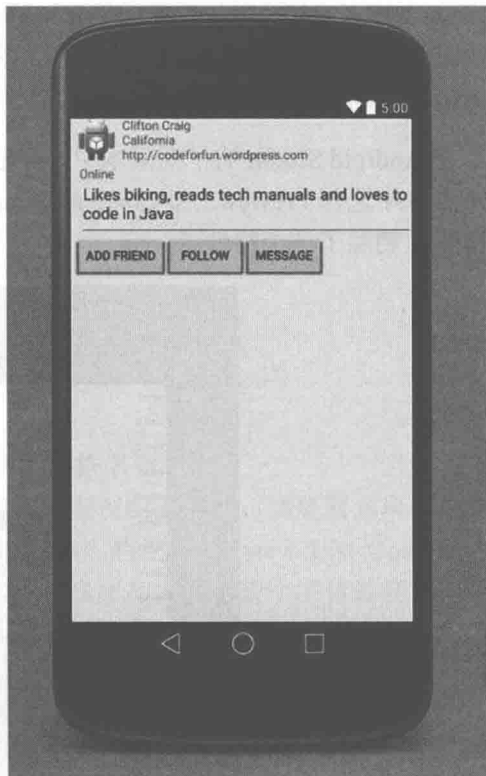


图 8-16 包含已集成按钮的 relative_example.xml

8.2.8 列表视图

ListView 组件是一个容器控件，它显示一个条目列表，其中每一项都是可操作的。这些列表项均被组织到一个可滚动的视图布局中。每个列表项的内容均由适配器以编程方式提供，它从数据源提取内容。适配器将数据映射为布局中的单个视图。在这个示例中，你将探索 ListView 组件的简单使用方法。

在 res | layout 文件夹下创建一个名为 list_view 的新布局，将 FrameLayout 指定为根元素，将 ListView 添加到 FrameLayout 的中央。预览面板将会使用名为“Simple 2-Line List Item”的默认布局来显示 ListView。切换至文本编辑模式并在根元素标签中添加一个 xmlns:tools 特性，将其值设置为 <http://schemas.android.com/tools>。这使得 tools:前缀特性变为可用，你将用它来改变预览的渲染方式。为 ListView 标签添加 tools:listitem 特性，并将其值设置为“@android:layout/simple_list_item_1”，如以下代码片段所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
>
<ListView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@+id/listView"
```

```

        android:layout_gravity="center"
        tools:listitem="@android:layout/simple_list_item_1"
    />
</FrameLayout>

```

在 Android Studio 的早期版本中,当处于设计模式时,可以在预览面板中右击 ListView 并从菜单中选择 Preview List Content | Simple List Item,如图 8-17 所示。Android Studio 1.0 发行版中删除了这个特性。

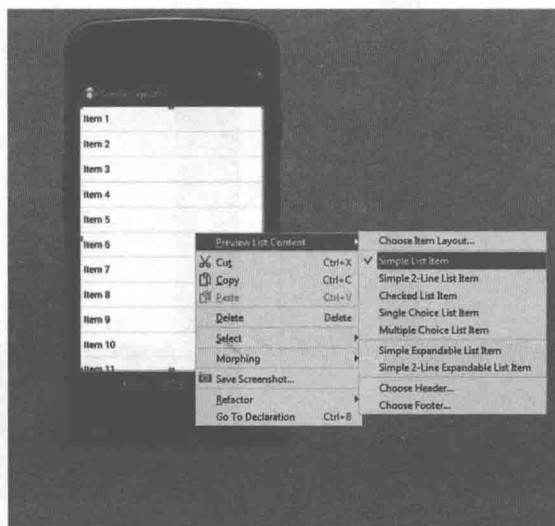


图 8-17 Android Studio 0.8 beta 版中的 List Preview Layout 特性

打开 MainActivity 类,将其修改为派生自 ListActivity,接着在 onCreate()方法中输入以下内容:

```

public class MainActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list_view);
        String[] listItems = new String[]{"Mary", "Joseph", "Leah", "Mark"};
        setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1,
            listItems));
    }

    //...
}

```

ListActivity 是一个特殊基类,用于提供操作 ListView 的常用功能。在示例中,我们使用它提供的 setListAdapter 方法把适配器与列表视图相关联。我们创建 ArrayAdapter 并向它传递上下文(当前正在执行的 Activity)、列表项布局以及用于填充 ListView 条目的数组。现

在构建并运行 App，它将会崩溃！这是一种对 `ListActivity` 的常见误用。这个特定的 `Activity` 查找 `id` 值为 `@android:id/list` 的 `ListView`。这是系统定义的特殊 Android `id`，这个特殊的 `id` 能让 `ListActivity` 找到它的 `ListView` 并自动将其绑定到给定的 `ListAdapter`。在 `list_view` 布局中按如下方法修改 `ListView` 标签：

```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@android:id/list"
    android:layout_gravity="center"
    tools:listitem="@android:layout/simple_list_item_1"
/>
```

构建并测试 App，你应该会看到如图 8-18 所示的姓名列表。

通过为列表项提供自定义布局，可进一步定制列表视图的外观。想要看到所得结果的样子，打开 `list_view.xml`。在预览面板中右击 `ListView`，将它的 `Preview List Content` 恢复为“Simple 2-Line List Item”。此布局使用一个大字体视图加上一个较小字体的视图来显示多个值。切换至文本视图并查看已生成的 XML，如代码清单 8-6 所示。

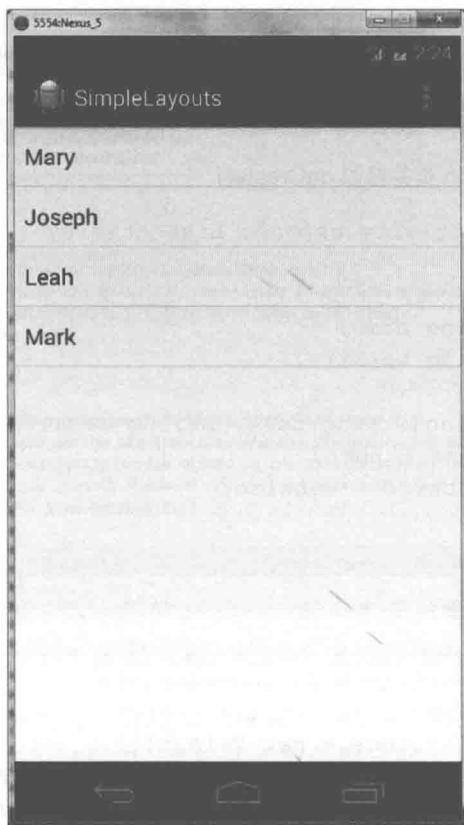


图 8-18 一个简单 `ListView` 的屏幕截图

代码清单 8-6 为列表项自定义布局

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

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:id="@android:id/list"
        android:layout_gravity="center"
        tools:listitem="@android:layout/simple_list_item_2" />
</FrameLayout>
```

ListView 元素中已经添加了一个特殊的 `tools:listitem` 特性，它可以在预览面板中控制布局。此特性定义在 XML 命名空间 `tools` 中，它被添加到 `FrameLayout` 根元素中。按住 `Ctrl` | `Cmd` 键并单击 `listitem` 特性的值，跳转至其定义。这个布局包含两个子视图，`id` 值分别为 `@android:id/text1` 和 `@android:id/text2`。我们在较早的示例中引入了数组适配器，它可将值添加到 `simple_list_item_1` 布局中。有了这个新布局，需要使用自定义逻辑为这两个子视图设置值。返回 `MainActivity` 类。在最顶部定义一个内部 `Person` 类，为列表中的每个人保存一个额外的网址，并按代码清单 8-7 所示修改 `onCreate()` 方法。

代码清单 8-7 创建 `Person` 类并修改 `onCreate()`

```
public class MainActivity extends ListActivity {

    class Person {
        public String name;
        public String website;

        public Person(String name, String website) {
            this.name = name;
            this.website = website;
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list_view);
        Person[] listItems = new Person[]{
            new Person("Mary", "www.allmybuddies.com/mary"),
            new Person("Joseph", "www.allmybuddies.com/joeseph"),
            new Person("Leah", "www.allmybuddies.com/leah"),
            new Person("Mark", "www.allmybuddies.com/mark")
        };
        setListAdapter(new PersonAdapter(this,
```

```

        android.R.layout.simple_expandable_list_item_2,
        listItems)

    };

}

//...
}

```

在这些修改中，创建了一个 `Person` 对象数组，该对象将名字和网址字符串作为构造函数的参数。这些值缓存在公共变量中(尽管在日常实践中，我们强烈提倡使用 `getter` 和 `setter`，而非公共变量，但为了简便起见，我们在这个示例中使用了后者)。接着将列表以及相同的 `simple_expandable_list_item_2` 布局传递给自定义的 `PersonAdapter`(我们还没有定义它)。按 `Alt + Enter` 键调用 `IntelliSense`，这让你能够在 `PersonAdapter` 中创建一个内部类的框架，参见图 8-19。

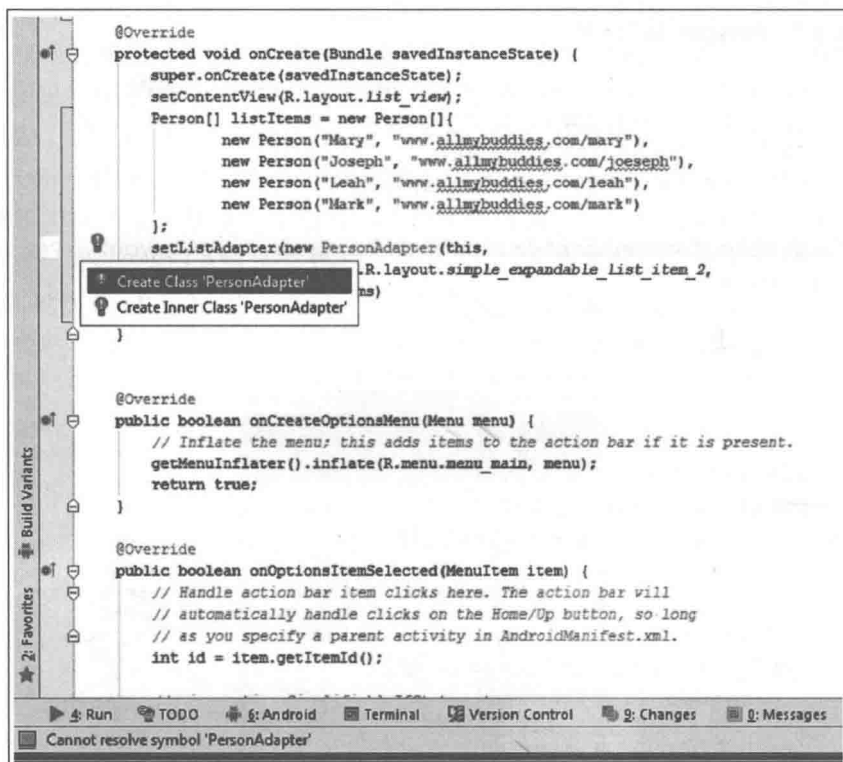


图 8-19 在 `onCreate()` 方法中添加 `PersonAdapter`

选择 `Create Inner Class` 选项，系统就会为你在当前类中生成一个类的框架。使用 `Tab` 键遍历构造函数参数。在遍历过程中，将构造函数参数分别改为 `Context context`、`int layout` 和 `Person[] listItems`。让这个类派生自 `BaseAdapter`，而非实现 `Listitem`，接着使用代码清单 8-8 中的代码完成其定义。由于我们在 `PersonAdapter` 中使用了 `Person` 类，因此需要将它移到 `MainActivity` 的外面。将光标置于 `Person` 类的定义中并按 `F6` 键将其移至更高级。你将看到图 8-20 所示的对话框。单击 `Refactor` 移动这个类。

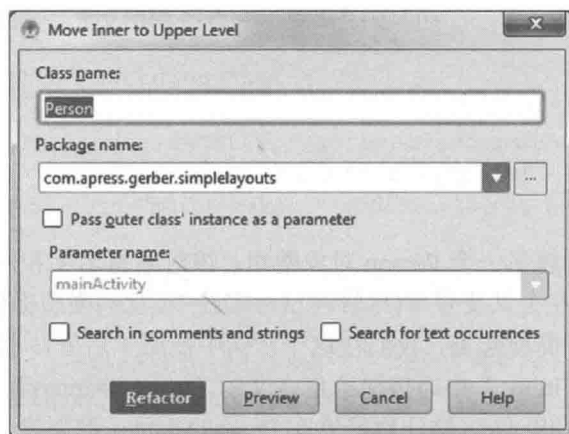


图 8-20 在 onCreate()方法中添加 PesonAdapter

代码清单 8-8 PersonAdapter 类

```

public class PersonAdapter extends BaseAdapter {
    private final Context context;
    private final int layout;
    private Person[] listItems;

    public PersonAdapter(Context context, int layout, Person[]
listItems) {
        this.context = context;
        this.layout = layout;
        this.listItems = listItems;
    }

    @Override
    public int getCount() {
        return listItems.length;
    }

    @Override
    public Object getItem(int i) {
        return listItems[i];
    }

    @Override
    public long getItemId(int i) {
        return i;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view = convertView;
        if(view==null) {
            LayoutInflater inflater = (LayoutInflater) context

```



```

        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        view = inflater.inflate(layout, parent, false);
    }
    TextView text1 = (TextView) view.findViewById(android.R.id.text1);
    TextView text2 = (TextView) view.findViewById(android.R.id.text2);
    text1.setText(listItems[position].name);
    text2.setText(listItems[position].website);
    return view;
}
}

```

这个基础示例展示了适配器创建单个可显示列表项的方式。定义首先将上下文、布局资源 ID 和列表项保存为成员变量，后续会使用它们来创建各个列表项视图。继承 **BaseAdapter** 为你提供了适配器接口中某些方法的默认实现，否则需要显式地定义这些方法。然而，你仍然需要提供抽象方法 **getCount()**、**getItem()**、**getItemId()** 和 **getView()** 的实现。**getCount()** 方法会被运行时调用，以便它知道需要渲染多少个视图。对于那些需要根据给定位置获取对应项的调用者来说，**getItem()** 是必需的。**getItemId()** 为给定位置的条目返回一个唯一数值。在我们的示例中，可以仅返回作为参数传入的位置值，因为它是唯一的。最后，**getView()** 包含了实现每个列表项视图的所有逻辑。它会反复被调用，参数为位置、可能为空或非空的 **convertView**，以及包含该视图的父级 **ViewGroup**。如果 **convertView** 为空，那么必须绘制新的视图来保存列表项细节——使用构造函数中保存的布局 ID 并将父视图组合作为其目标。可以使用 **LAYOUT_INFLATER_SERVICE** 系统服务来完成这些填充。绘制视图后，分别找到 **text1** 和 **text2** 子视图，并使用相应位置 **Person** 的名字和网址值填充它们。运行示例并观察 **Person** 对象映射到新版布局的样子。图 8-21 展示了界面的样子。

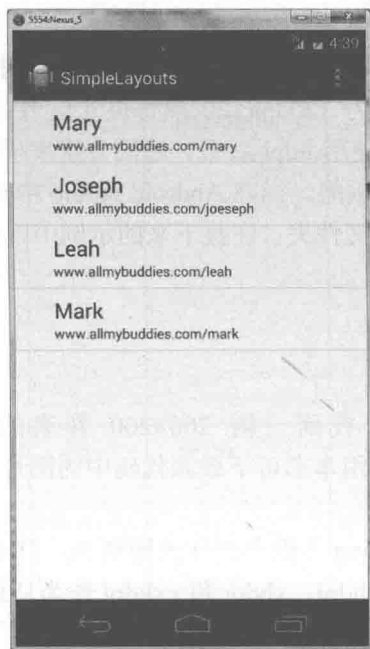


图 8-21 显示新的列表项布局并使用了 **PersonAdapter**

8.3 布局设计指导原则

由于市场上有大量可用的 Android 设备，而且每款都有不同的屏幕尺寸和密度，因此布局设计很具有挑战性。当设计布局时，需要知晓一些要点。但还是有一些可以遵从的规则，让你能够跟上快速变化的形式。大体来说，需要注意屏幕分辨率和像素密度。

屏幕分辨率是屏幕在水平和竖直方向可以容纳的总像素数，以二维数字的形式给出。通常使用标准 VGA 单位来描述分辨率。VGA 表示视频图形阵列，台式机和笔记本电脑的标准值为 640×480。这意味着宽 640 像素、高 480 像素。目前，你会发现一些移动设备上的变化形式，例如半 VGA(HVGA)，480×320；四分之一 VGA(QVGA)，320×240；宽屏 VGA(WVGA)，800×480；扩展图形阵列(XGA)；宽屏 XGA(WXGA)；等等。还会有其他一些可能的分辨率。

像素密度表示单位长度可以容纳的总像素数。这个尺寸与屏幕大小无关，尽管它会受屏幕大小的影响。例如，设想将分辨率为 1024×768 像素的 20 英寸显示器与分辨率为 1024×768 像素的 5 英寸显示器做比较。两者有着相同的像素数，但是后者将这些像素压缩到更小的区域内，而这增加了它们的密度。像素密度的单位是点每英寸(dpi)，表示每英寸区域内可以容纳的点(或者说像素)的数量。在 Android 屏幕上，密度通常使用一种称为密度无关像素(dp)的单位来度量。它是一种基线度量，1dp 等于 160dpi 屏幕上的 1 像素。使用 dp 作为测量单位使得布局能够在不同密度的设备上正确地缩放。

Android 包含应对不同屏幕密度的另一种方法：资源限定符。在之前的示例中，我们将图片复制到 drawable 文件夹中，这里是提取所有可绘制资源的默认位置。可绘制资源通常是图片，但也包括定义形状、轮廓和边框的资源 XML 文件。为定位可绘制资源，Android 运行时首先获取当前设备的屏幕规格。如果在主要分类列表中，那么运行时会在带有资源限定符后缀的 drawable 文件夹下进行查找。有多种后缀，例如 ldpi、mdpi、hdpi 和 xhdpi。ldpi 后缀表示低密度屏幕，约为 120dpi(120 点每英寸)。中密度屏幕，160dpi，使用 mdpi 后缀。高密度屏幕，320dpi，使用 hdpi 后缀。超高分辨率屏幕使用 xhdpi 后缀。这不是一个完备列表，但包括了常见的后缀。当在 Android Studio 中开启项目时，res 文件夹下会创建一系列对应特定分辨率的子文件夹。在接下来的示例中，你将体会到如何以一种有效的方式来使用这些文件夹。





8.3.1 覆盖各种显示尺寸

在这个练习中，你将要找到一幅 200×200 像素的头像并替换到之前构建的 RelativeLayout 示例中。可以选用本书可下载源代码中的图片。这将是你在最高分辨率显示器上需要使用的图片。

将图片命名为 my_profile.png 并保存到你的硬盘上。打开项目窗口并展开 res 文件夹。你的项目中应该包含以 mdpi、hdpi、xhdpi 和 xxhdpi 作为后缀的 drawable 文件夹。需要为不同屏幕尺寸创建原始图像的缩小版。调整大小时要保持 3:4:6:8 的长宽比。可以使用 Microsoft 画图或者任何其他工具来调整大小(一个名为 Image Resizer 的 Windows 开源项目，

网址为 imageresizer.codeplex.com, 可以轻松地完成此任务并且与 Windows 浏览器完美地集成)。参考表 8-3, 根据我们的指导比例, 在对应文件夹中创建各种缩放尺寸。根据表格的说明, 在文件夹中保存图片的各个版本并全部使用相同的名称——my_profile.png。

表 8-3 各种图片资源尺寸和描述

文件夹	原始大小	比率	缩放后的大小	图片
drawable-xxhdpi	200×200	N/A	200×200	
drawable-xhdpi	200×200	3:4	150×150	
drawable-hdpi	150×150	4:6	100×100	
drawable-mdpi	100×100	6:8	75×75	

添加这些图片之后,在设计模式中打开 `relative_example.xml` 并找到图片视图。单击这个组件的 `src` 特性旁边的省略号,在 Resources 对话框中找到图片 `my_profile`,如图 8-22 所示。

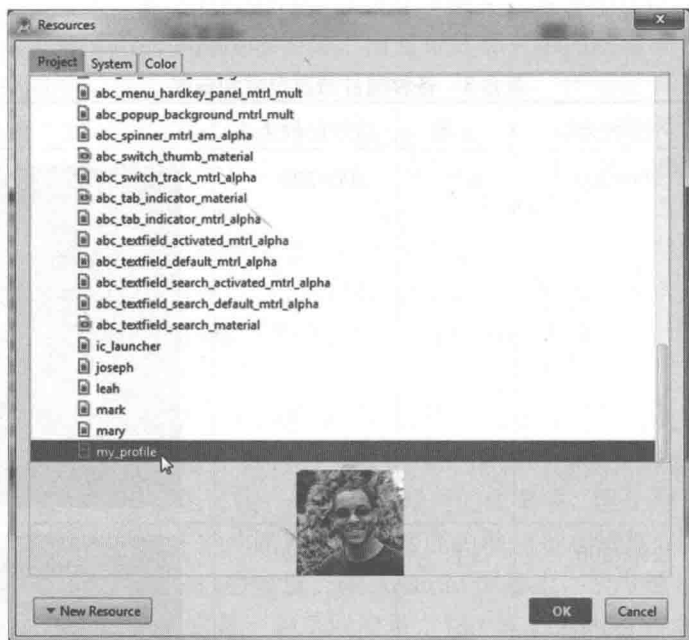


图 8-22 含有 Clifton Craig 照片的资源对话框

更新图片后,单击预览窗口中的 Android Virtual Device 按钮,使用各种屏幕渲染选项进行实验,如图 8-23 所示。选择 Preview All Screen,即可在多种设备上查看模拟头像的渲染效果,如图 8-24 所示。

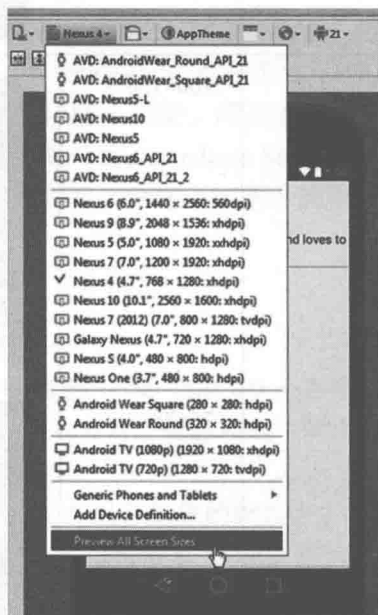


图 8-23 在 Visual Designer 的 Design 模式中预览所有屏幕尺寸

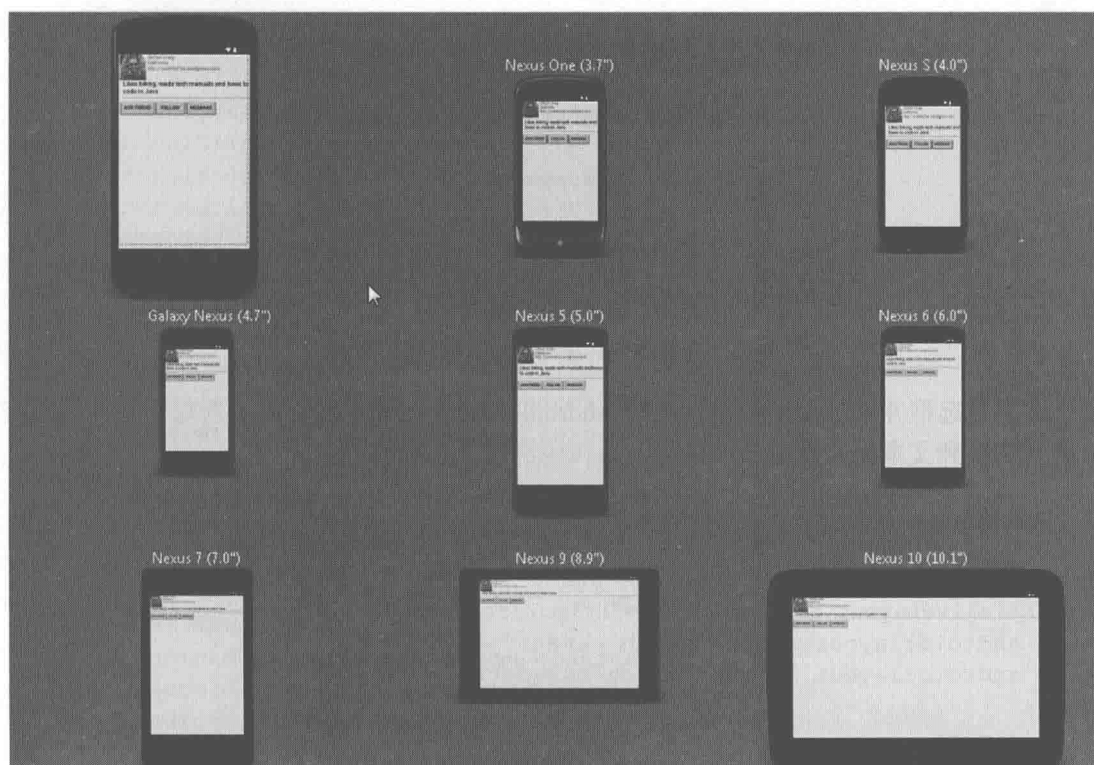


图 8-24 各种设备上的布局预览

8.3.2 组合在一起

现在，将使用 Java 加载布局并探索如何在运行时微调布局。在开始之前，需要为将要用到的组件添加具有描述性质的 ID。在设计模式中打开 `relative_example.xml` 布局并为内嵌在 `LinearLayout` 中的以下组件添加 ID：

- `imageView`: `profile_image`
- `textView1`: `name`
- `textView2`: `location`
- `textView3`: `website`
- `textView4`: `online_status`
- `editText`: `description`

单击每个组件并在右侧面板的属性编辑器中修改其 `id` 属性。在修改时，你将会看到一个弹出式对话框，询问是否同时更新引用，参见图 8-25。

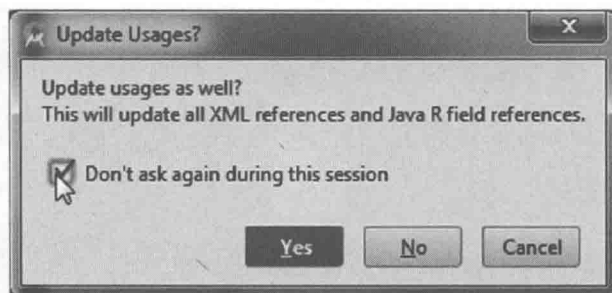


图 8-25 在修改时, Android Studio 将更新引用

选中复选框并单击 OK 按钮, 允许 Android Studio 在工作过程中更新每个组件的所有引用。切换到文本模式查看最终结果, 如代码清单 8-9 所示。

代码清单 8-9 放入组件之后的新布局

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

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/profile_image"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:src="@drawable/my_profile"
        android:layout_marginLeft="5dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clifton Craig"
        android:id="@+id/name"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/profile_image"
        android:layout_marginLeft="5dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="California"
        android:id="@+id/location"
        android:layout_below="@+id/name"
        android:layout_toRightOf="@+id/profile_image"
        android:layout_marginLeft="5dp" />
```

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="http://codeforfun.wordpress.com"
    android:id="@+id/website"
    android:layout_below="@+id/location"
    android:layout_toRightOf="@+id/profile_image"
    android:layout_marginLeft="5dp" />

<LinearLayout
    android:id="@+id/details"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/profile_image"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="5dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Online"
        android:id="@+id/online_status"
        android:layout_marginLeft="5dp"
        android:layout_marginTop="5dp" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/description"
        android:text="Likes biking, reads tech manuals and loves to code in Java"
        android:layout_marginLeft="5dp" />

</LinearLayout>
<include
    android:id="@+id/buttons"
    layout="@layout/three_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/details"/>

</RelativeLayout>

```

注意，Android Studio 不仅会更新 id 的定义，还会更新每一处用到此 id 的地方，以保持组件和之前一样彼此相邻。创建一个名为 `ProfileActivity` 的新类，并按照代码清单 8-10 所示对其进行修改。

代码清单 8-10 ProfileActivity 类

```

public class ProfileActivity extends Activity {

    private TextView name;
    private TextView location;
    private TextView website;
    private TextView onlineStatus;
    private EditText description;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.relative_example);
        name = (TextView) findViewById(R.id.name);
        location = (TextView) findViewById(R.id.location);
        website = (TextView) findViewById(R.id.website);
        onlineStatus = (TextView) findViewById(R.id.online_status);
        description = (EditText) findViewById(R.id.description);

        View parent = (View) name.getParent();
        parent.setBackgroundColor(getResources().getColor(android.R.color.
            holo_blue_light));
        name.setTextAppearance(this, android.R.style.TextAppearance_
            DeviceDefault_Large);
        location.setTextAppearance(this,
            android.R.style.TextAppearance_DeviceDefault_Medium);
        location.setTextAppearance(this,
            android.R.style.TextAppearance_DeviceDefault_Inverse);
        website.setTextAppearance(this,
            android.R.style.TextAppearance_DeviceDefault_Inverse);
        onlineStatus.setTextAppearance(this,
            android.R.style.TextAppearance_DeviceDefault_
                Inverse);
        description.setEnabled(false);
        description.setBackgroundColor(getResources().getColor(android.R.
            color.white));
        description.setTextColor(getResources().getColor(android.R.color.
            black));
    }
}

```

在这里为每个 `TextView` 和 `EditText` 组件添加成员字段。`onCreate()`方法首先找到每个视图组件并将它们分别保存在成员变量中。接下来,找到 `name` 标签的父元素并将其背景颜色改为浅蓝色。Android Studio 有个独特的功能,可以使用一个方框来装饰左侧的折叠线,显示出这一行中引用的颜色。这些方块还会出现在引用了颜色资源的其他行中。接着修改各个 `TextView` 的外观,让名字以大字体突出显示。你正在使用 `android.R` 类中预先定义的风格,包括 Android SDK 中所有可用资源的引用。其他 `TextView` 也已更新为使用中等字体

或反色显示。最后，禁用 EditText 的 description，以防止对其内容进行修改。你还将其背景设置为白色，同时将文本颜色改为黑色。

为了测试 ProfileActivity 和布局，需要在 AndroidManifest.xml 中定义它并将其连接到 MainActivity。打开 manifest 并在 MainActivity 定义的下面为 ProfileActivity 添加标签：

```
<activity
    android:name=".ProfileActivity"
    android:label="@string/app_name" />
```

接着返回 MainActivity 并使用以下代码重载 onListItemClick() 方法——创建指向 ProfileActivity 类的 Intent，并启动该 Activity。运行这个示例，尝试单击任意列表项以显示相应的个人信息，参见图 8-26。

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Intent intent = new Intent(this, ProfileActivity.class);
    startActivity(intent);
}
```

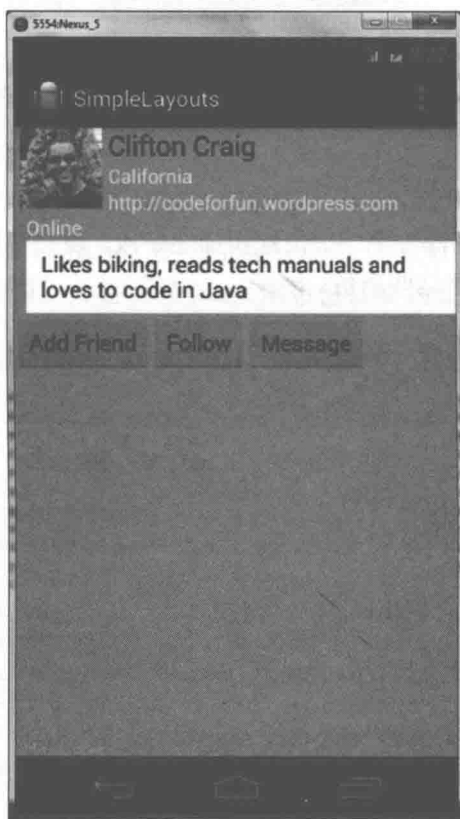


图 8-26 使用了按钮和 EditText 的新布局

现在将学习如何把值从列表视图带到下一个 Activity。使用代码清单 8-11 中的代码修

改 MainActivity 类中的 onCreate() 方法。

代码清单 8-11 对 MainActivity 所做的修改

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.list_view);
    Person[] listItems = new Person[]{
        new Person(R.drawable.mary, "Mary", "New York",
            "www.allmybuddies.com/mary",
            "Avid cook, writes poetry."),
        new Person(R.drawable.joseph, "Joseph", "Virginia",
            "www.allmybuddies.com/joeseph",
            "Author of several novels."),
        new Person(R.drawable.leah, "Leah", "North Carolina",
            "www.allmybuddies.com/leah",
            "Basketball superstar. Rock climber."),
        new Person(R.drawable.mark, "Mark", "Denver",
            "www.allmybuddies.com/mark",
            "Established chemical scientist with several patents.")
    };

    setListAdapter(new PersonAdapter(this,
        android.R.layout.simple_expandable_list_item_2,
        listItems)
    );
}
```

你正在向构造函数调用添加名字、位置和描述字段。现在使用代码清单 8-12 中的代码修改 Person 类，接收并保存这些新值。

代码清单 8-12 对 Person 类所做的修改

```
class Person {
    public int image;
    public String name;
    public String location;
    public String website;
    public String descr;

    Person(int image, String name, String location, String website, String
    descr) {
        this.image = image;
        this.name = name;
        this.location = location;
        this.website = website;
        this.descr = descr;
    }
}
```

接下来按如下所示修改 onItemClick():

```

@Override
protected void onItemClick(ListView l, View v, int position, long id) {
    super.onItemClick(l, v, position, id);
    Person person = (Person) l.getItemAtPosition(position);
    Intent intent = new Intent(this, ProfileActivity.class);
    intent.putExtra(ProfileActivity.IMAGE, person.image);
    intent.putExtra(ProfileActivity.NAME, person.name);
    intent.putExtra(ProfileActivity.LOCATION, person.location);
    intent.putExtra(ProfileActivity.WEBSITE, person.website);
    intent.putExtra(ProfileActivity.DESCRPTION, person.descr);
    startActivity(intent);
}

```

在这里，获取被单击的 **Person** 对象并将它的每个成员变量以 **Intent** 附带值的形式传递给下一个 **Activity**。这些附带值对应于 **ProfileActivity** 中的常量，我们在 **ProfileActivity** 类的顶部定义它们：

```

public class ProfileActivity extends Activity {

    public static final String IMAGE = "IMAGE";
    public static final String NAME = "NAME";
    public static final String LOCATION = "LOCATION";
    public static final String WEBSITE = "WEBSITE";
    public static final String DESCRIPTION = "DESCRIPTION";

    //...
}

```

现在，按照代码清单 8-13 所示，在 **ProfileActivity** 中做以下修改——定义 **ImageView** 类型的 **profileImage** 成员变量，并将所有 **Intent** 附带值读取到缓存的视图组件中。

代码清单 8-13 对 **PersonActivity** 类所做的修改

```

private ImageView profileImage;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.relative_example);
    name = (TextView) findViewById(R.id.name);
    location = (TextView) findViewById(R.id.location);
    website = (TextView) findViewById(R.id.website);
    onlineStatus = (TextView) findViewById(R.id.online_status);
    description = (EditText) findViewById(R.id.description);
    profileImage = (ImageView) findViewById(R.id.profile_image);

    int profileImageId = getIntent().getIntExtra(IMAGE, -1);
    profileImage.setImageDrawable(getResources().getDrawable(profileImageId));
    name.setText(getIntent().getStringExtra(NAME));
    location.setText(getIntent().getStringExtra(LOCATION));
}

```

```
website.setText(getIntent().getStringExtra(WEBSITE));
description.setText(getIntent().getStringExtra(DESCRIPTION));
```

运行 App 并进行实验，单击列表视图中的条目以显示对应的个人信息。可以按 Back 键返回到列表视图并选择不同的项，参见图 8-27。

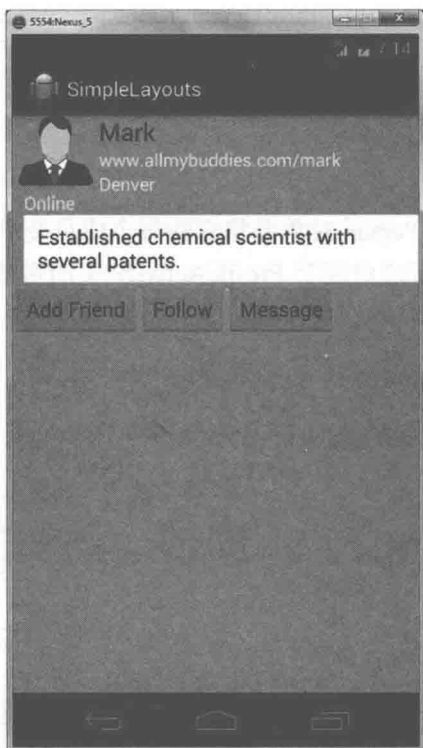


图 8-27 含有 ImageView 的布局

8.4 Fragment

Fragment 介于之前所述的 Activity 和可引入文件之间。Fragment 是可重用的 XML 片段，与布局引入类似。不过，类似于 Activity，它们有包含业务逻辑的额外优势。Fragment 用于将用户界面适配到不同尺寸的屏幕上。试想之前为智能手机开发的示例在 10 英寸平板电脑上的样子。较大显示器提供的额外显示资源会使屏幕上的简单列表视图看上去非常笨拙。利用 Fragment，可以智能地组合界面，使得在较小屏幕上的显示效果与现在相同，而在较大屏幕上同时包含列表和详细视图。为此，需要从 Activity 中提取所有 UI 更新逻辑并加入到新的 Fragment 类中。从 MainActivity 中的 ListView 逻辑开始，需要将内嵌类抽取为外部顶层类。单击 MainActivity 顶部的 Person 类并按下 F6 键。弹出的 Move Refactor 对话框询问想要将类移动到哪个包和目录中。可以接受这里的默认值。对底部的 PersonAdapter 类做同样的事情。

创建名为 BuddyListFragment 的新类，它派生自 ListFragment 并包含之前在 MainActivity

中进行的 ListView 初始化,如代码清单 8-14 所示。

代码清单 8-14 派生自 ListFragment 的 BuddyListFragment 类

```
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;

public class BuddyListFragment extends ListFragment {

    private OnListItemSelectedListener onListItemSelectedListener;

    public interface OnListItemSelectedListener {
        void onListItemSelected(Person selectedPerson);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Person[] listItems = new Person[]{
            new Person(R.drawable.mary, "Mary",
                "www.allmybuddies.com/mary",
                "New York","Avid cook, writes poetry."),
            new Person(R.drawable.joseph, "Joseph",
                "www.allmybuddies.com/joeseph",
                "Virginia","Author of several novels"),
            new Person(R.drawable.leah, "Leah",
                "www.allmybuddies.com/leah",
                "North Carolina",
                "Basketball superstar. Rock climber."),
            new Person(R.drawable.mark,"Mark",
                "www.allmybuddies.com/mark",
                "Denver",
                "Established chemical scientist with several patents.")
        };
        setListAdapter(new PersonAdapter(getActivity(),
            android.R.layout.simple_expandable_list_item_2,
            listItems)
        );
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        if(!(activity instanceof OnListItemSelectedListener)) {
            throw new ClassCastException(
```

```

        "Activity should implement OnListItemSelectedListener");
    }
    //Save the attached activity as an onListItemSelectedListener
    this.onListItemSelectedListener = (OnListItemSelectedListener)
activity;
}

@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    Person selectedPerson = (Person) l.getItemAtPosition(position);
    this.onListItemSelectedListener.onListItemSelected(selectedPerson);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    return inflater.inflate(R.layout.list_view, container, false);
}
}

```

Fragment 镜像了 MainActivity 中的 onCreate()方法，但添加了两个额外的生命周期方法。onAttach()方法获取绑定的 Activity，该 Activity 必须实现在类的顶部声明的 OnListItemSelectedListener()。ListFragment 超类中定义了在这里重载的 onListItemClick()回调方法。在我们的自定义版本中，引用缓存的 onListItemSelectedListener()并将选中的 Person 对象传递给它。最后，重载 onCreateView()生命周期方法，它填充 list_view 布局并将其返回给运行时。

创建一个派生自 Fragment 的 BuddyDetailFragment 类，并使用代码清单 8-15 中的代码填充它。

代码清单 8-15 BuddyDetailFragment 类

```

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

public class BuddyDetailFragment extends Fragment {
    public static final String IMAGE = "IMAGE";
    public static final String NAME = "NAME";
    public static final String LOCATION = "LOCATION";
    public static final String WEBSITE = "WEBSITE";
    public static final String DESCRIPTION = "DESCRIPTION";
    private Person person;
}

```

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle bundle) {
    updatePersonDetail(bundle);
    return inflater.inflate(R.layout.relative_example, container, false);
}

@Override
public void onStart() {
    super.onStart();
    updatePersonDetail(getArguments());
}

private void updatePersonDetail(Bundle bundle) {
    //if bundle arguments were passed, we use them
    if(bundle != null) {
        this.person = new Person(
            bundle.getInt(IMAGE),
            bundle.getString(NAME),
            bundle.getString(LOCATION),
            bundle.getString(WEBSITE),
            bundle.getString(DESCRIPTION)
        );
    }
    //if we have a valid person from the bundle
    //or from restored state then update the screen
    if(this.person != null){
        updateDetailView(this.person);
    }
}

public void updateDetailView(Person person) {
    FragmentActivity activity = getActivity();
    ImageView profileImage = (ImageView)
activity.findViewById(R.id.profile_image);
    TextView name = (TextView) activity.findViewById(R.id.name);
    TextView location = (TextView)
activity.findViewById(R.id.location);
    TextView website = (TextView) activity.findViewById(R.id.website);
    EditText description = (EditText)
activity.findViewById(R.id.description);

    profileImage.setImageDrawable(getResources().getDrawable(person.
image));
    name.setText(person.name);
    location.setText(person.location);
    website.setText(person.website);
    description.setText(person.descr);
}
}

```

这个思路类似于之前创建的 ProfileActivity。不过，你现在有了一个内部的 Person 成员变量，用于保存 bundle 值。这样做的原因是现在需要从两个地方——onCreate() 和 onStart() 中读取值。你还定义了一个公共方法，允许外部调用者使用给定 Person 对象更新 Fragment。还有一件需要注意的事情是你重载了 onCreateView() 生命周期方法并要求填充器使用资源 ID 填充适当的视图。

主界面被修改为一个单独的 Fragment，它包含与之前相同的列表视图。将 activity_main 布局简化为只包含单个 FrameLayout：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/empty_fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    >
</FrameLayout>
```

这个空布局有一个恰如其分的 ID 值，值为 empty_fragment_container。稍后将会向这个布局中动态添加 Fragment。重新访问 res 目录，使用特殊资源限定符 large 创建另一个布局文件。通过右击 res 文件夹添加新资源文件并选择 New | Android resource file。将名称设置为 activity_main——与之前使用的名称相同。将 Resource Type 设置为 Layout。从可用的限定符列表中选择尺寸；从 Screen Size 下拉菜单中选择 Large。参见图 8-28 中的设置。这类似于之前为各种屏幕密度添加图片的示例。此布局文件将会在 layout-large 目录中。被分类为大屏幕的设备(例如 7 英寸或更大尺寸的平板电脑)将会选用这个文件夹下的布局。

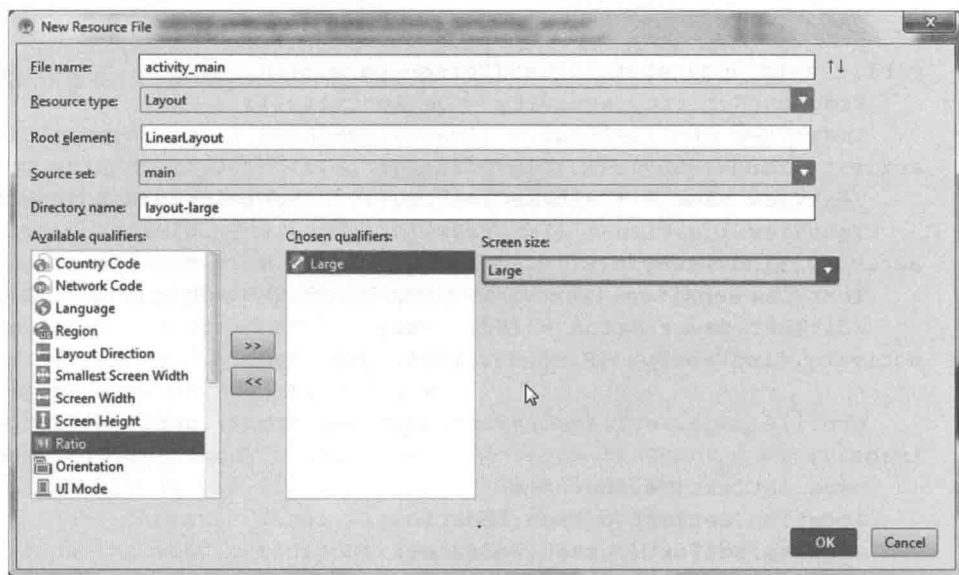


图 8-28 在 New Resource Directory 中选择 layout-large

打开新创建的 activity_main 布局，切换到文本模式并输入以下 XML：

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyListFragment"
        android:id="@+id/list_fragment"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        />

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyDetailFragment"
        android:id="@+id/detail_fragment"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

注意，它在平板 AVD 上渲染布局。预览面板将会提示没有用于渲染布局的足够设计信息。它将建议可供预览的一系列布局，同时给出从项目中选择布局的选项。使用 Pick Layout 超链接为项目选择布局。为第一个 Fragment 选择 list_view 布局，并为第二个 Fragment 选择 relative_example，如图 8-29 所示。

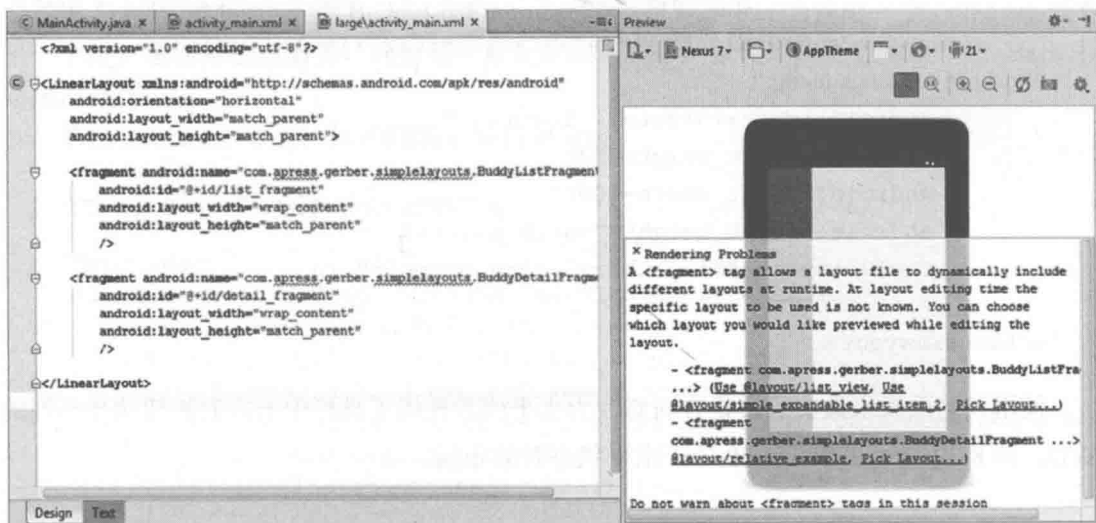


图 8-29 包含 Fragment 的线性布局

此时，列表视图将会占据整个屏幕。需要对宽度和权重稍作调整，腾出能够同时查看两个 Fragment 的空间。诀窍是将宽度设置为 0dp 并使用权重(weight)属性来恰当地设置组件大小。将两个 Fragment 的宽度都设置为 0dp。将 BuddyListFragment 的权重设置为 1，并将 BuddyDetailFragment 的权重设置为 2。weight 属性可以根据可用空间的比例来设置组件的大小。系统会汇总布局中所有组件的权重，并根据总和来分配可用空间。每个组件占据等同于其权重的一部分空间。在我们的示例中，详情 Fragment 将占据三分之二的屏幕，而列表将占据三分之一的屏幕。修改之后的代码如清单 8-16 所示。

代码清单 8-16 包含修改过的 Fragment 的线性布局

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.apress.gerber.simplelayouts.BuddyListFragment"
        android:id="@+id/list_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        tools:layout="@layout/list_view" />

    <fragment android:name="com.apress.gerber.simplelayouts.
BuddyDetailFragment"
        android:id="@+id/detail_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        tools:layout="@layout/relative_example" />

</LinearLayout>
```

使用预览面板来做实验。将方向修改为水平或者使用工具栏中的控制按钮选择不同的 AVD。图 8-30 展示了 Nexus 10 处于水平状态时的布局。

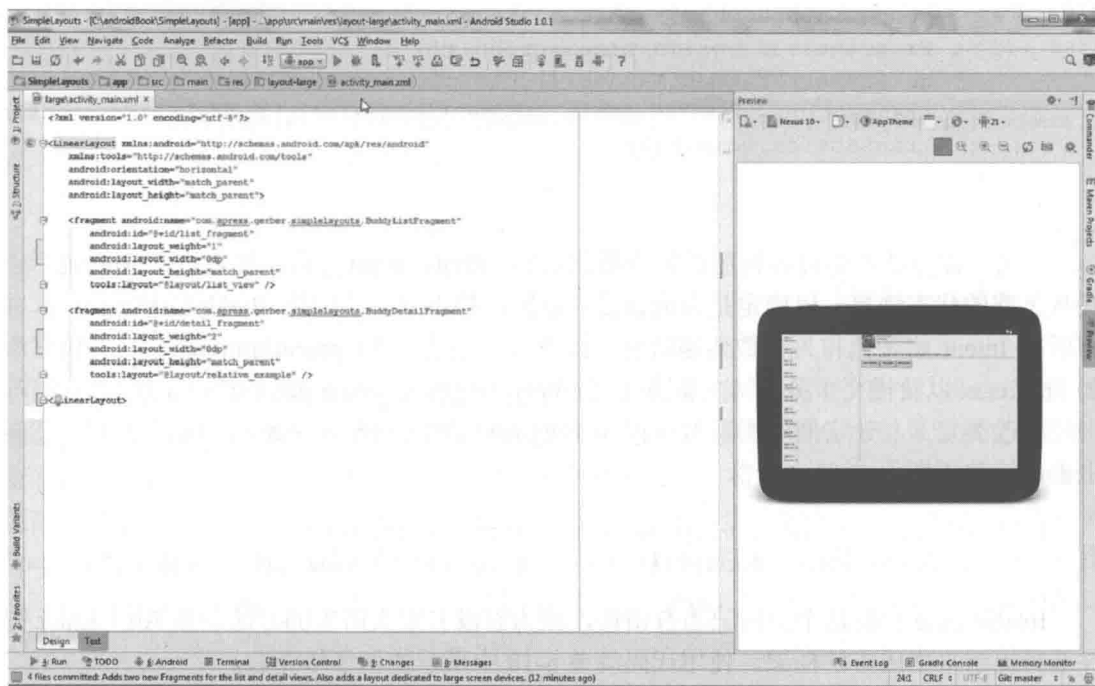


图 8-30 在 Nexus 10 上横向查看 Fragment

有了这两个 Fragment，就可以打开 MainActivity 并简化它了。让其派生自 FragmentActivity。FragmentActivity 是一个特殊类，允许你在视图层次中找到 Fragment 并通过 FragmentManager 类执行 Fragment 事务。你将会在示例中使用事务来添加和替换屏幕上的 Fragment。在屏幕较小的设备上，运行时将选择 empty_fragment_container 布局。你将使用 FragmentManager 把 BuddyListFragment 添加到屏幕中。当使用一个 Fragment 替换另一个时，也需要创建事务，同时将它添加到回退栈中，以便用户可以通过单击 Back 按钮撤消操作。

简化 onCreate() 方法，如代码清单 8-17 所示。

代码清单 8-17 简化后的 onCreate() 方法

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if (findViewById(R.id.empty_fragment_container) != null) {
        // We should return if we're being restored from a previous state
        // to avoid overlapping fragments.
        if (savedInstanceState != null) {
            return;
        }
        BuddyListFragment buddyListFragment = new BuddyListFragment();

        // Pass any Intent extras to the fragment as arguments
```

```

        buddyListFragment.setArguments(getIntent().getExtras());
        FragmentTransaction transaction =
            getSupportFragmentManager().beginTransaction();
        transaction.add(R.id.empty_fragment_container, buddyListFragment);
        transaction.commit();
    }
}

```

首先，在方法中将内容视图设置为简化后的 `activity_main` 布局。接着检查 App 是否正在从之前的状态恢复，以确定是否应该提前返回。接下来，实例化 `BuddyListFragment` 并将所有 `Intent` 附带值作为参数传递给它。接下来，创建了 `FragmentTransaction`，并向它添加 `Fragment` 以及提交事务。仅在能够找到 `empty_fragment_container` 的情况下执行此操作。

修改类定义，让它同时实现 `BuddyListFragment.OnListItemSelectedListener` 接口。它看上去应该如下所示：

```

public class MainActivity extends FragmentActivity
    implements BuddyListFragment.OnListItemSelectedListener {

```

IntelliSense 会将这个类标记为有错误，因为它没有定义所需的方法。按 `Alt + Enter` 键并选择提示来生成方法框架。使用代码清单 8-18 中所示的例子填充它。

代码清单 8-18 展示 `FragmentManager` 事务的代码

```

@Override
public void onListItemSelected(Person selectedPerson) {
    BuddyDetailFragment buddyDetailFragment = (BuddyDetailFragment)
        getSupportFragmentManager().findFragmentById(R.id.detail_fragment);
    if (buddyDetailFragment != null) {
        buddyDetailFragment.updateDetailView(selectedPerson);
    } else {
        buddyDetailFragment = new BuddyDetailFragment();
        Bundle args = new Bundle();
        args.putInt(BuddyDetailFragment.IMAGE, selectedPerson.image);
        args.putString(BuddyDetailFragment.NAME, selectedPerson.name);
        args.putString(BuddyDetailFragment.LOCATION,
            selectedPerson.location);
        args.putString(BuddyDetailFragment.WEBSITE,
            selectedPerson.website);
        args.putString(BuddyDetailFragment.DESCRPTION,
            selectedPerson.descr);
        buddyDetailFragment.setArguments(args);
        //Start a fragment transaction to record changes in the fragments.
        FragmentTransaction transaction =
            getSupportFragmentManager().beginTransaction();

```

```

// Replace whatever is in the fragment_container view with this
// fragment, and add the transaction to the back stack so the user
// can navigate back
transaction.replace(R.id.empty_fragment_container, buddyDetailFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
}
}

```

删除 `onListItemSelected()` 方法，使用这段代码替换它。需要在这里检查 `buddyDetailFragment` 是否已经在视图层次结构中。如果是这样，那么找到并更新它。否则，重新创建它并使用你在 `BuddyDetailFragment` 中定义的键，将选中的 `Person` 对象以 `bundle` 值的形式传入。最后，创建并提交 `Fragment` 事务——使用详细信息 `Fragment` 替换列表 `Fragment` 并将该事务添加到回退栈。同时在平板电脑和智能手机(分别参见图 8-31 和图 8-32)上运行代码，观察不同的行为。可以创建一个 Nexus 10 平板 AVD 以供大屏幕测试之用。

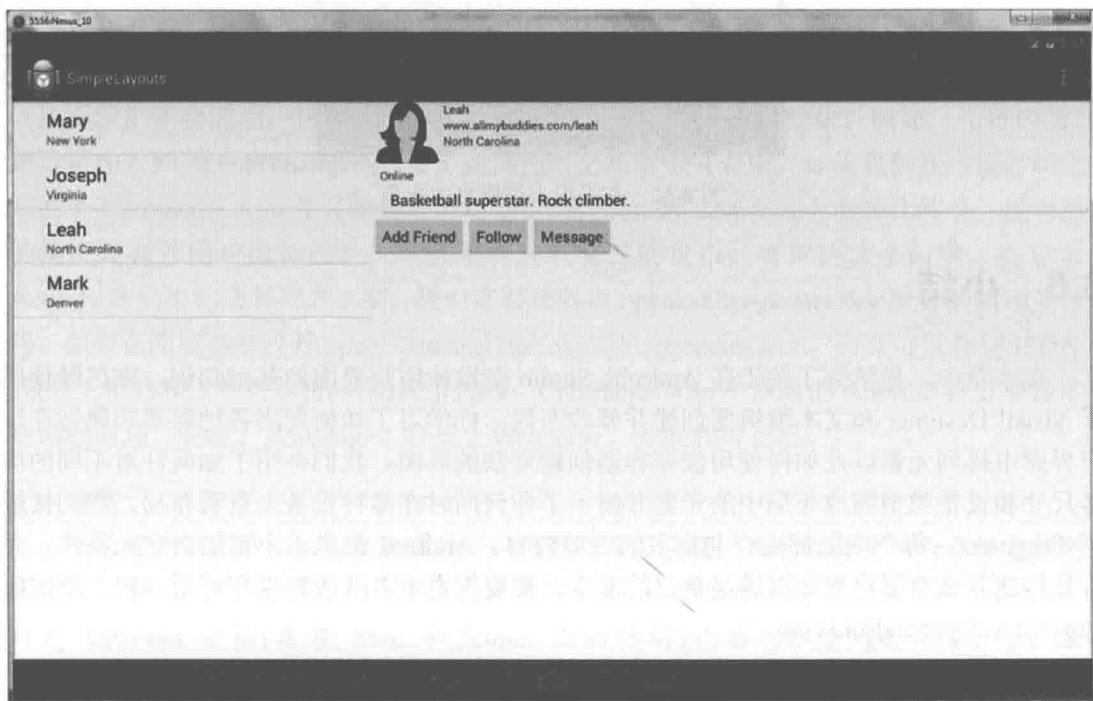


图 8-31 在平板电脑上渲染出的布局

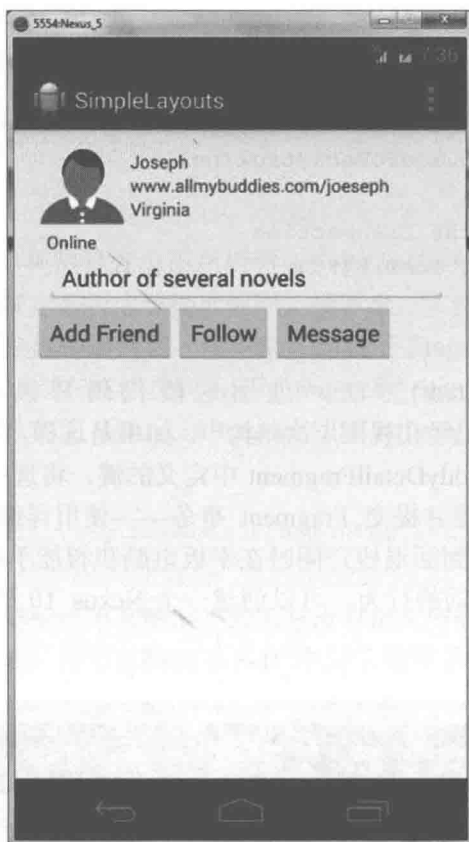


图 8-32 在手机上渲染出的布局

8.5 小结

在本章中，你学到了关于在 Android Studio 中设计用户界面的基础知识。你已经使用了 Visual Designer 和文本编辑器创建并修改布局。你学习了如何使用各种容器和属性在用户界面中排列元素以及如何使用嵌套容器创建复杂的界面。我们介绍了如何针对不同的屏幕尺寸和设备类型缩放布局中的元素并演示了如何同时在多种设备上查看布局。我们接触了 Fragment。每个主题都还有非常多的细节内容。Android 提供了丰富的自定义特性，允许你构建并调整用户界面以满足自己的需求。想要探索更多高级特性和可用 API，请访问 <https://developer.android.com>。

第 9 章

货币实验：第 1 部分

本章以及下一章解释如何使用 Android Studio 构建一款名为 Currencies 的 App。Currencies 的目的是提供一种换算外国货币和本国货币的便利方法。典型的应用场景是用户在国外旅行时需要兑换现金或者使用外国货币购买商品。货币汇率一直处于波动状态，每分钟都有可能发生变化，因此让用户访问到最实时的数据很重要。Currencies App 从 openexchangerates.org 域名下的一个 Web 服务获取最新的汇率。

不仅汇率会波动，可交易的活动货币代码列表也可能发生变化。例如，比特币(BTC)最近被加入到 openexchangerates.org 提供的可交易货币列表中。如果我们在一段时间之前开发了 Currencies App 并且硬编码了活动货币代码，那么现在就会漏掉比特币，或者更糟的情况是允许用户选择已经失效且不再进行交易的货币。要解决这个问题，在显示主 Activity 布局中的选择列表之前，我们需要获取由 openexchangerates.org 发布的活动货币代码。如果在浏览器中打开 openexchangerates.org/api/currencies.json，那么可以看到 JSON 格式(对于机器和人都可读)的活动货币代码。Currencies App 中涵盖的 Android 特性和技术包括高级布局、资源、共同偏好、样式、Web 服务、并发和对话框。

注意

我们邀请你使用 Git 克隆此项目，以便跟随学习进度，尽管你将要从头开始重新创建此项目的 Git 仓库。如果你的电脑上还没有安装 Git，那么请阅读第 7 章。在 Windows 上打开 Git-bash 会话(或者 Mac 和 Linux 上的终端)并输入以下 git 命令：`git clone https://bitbucket.org/csgerber/currencies.git` Currencies。

9.1 Currencies 规范

为了解决之前所述的活动货币代码问题，我们将使用一个典型的 Android 术语——启

动画面。当 App 首次启动时(如图 9-1 所示),用户会看到一个仅包含各种货币图片的 Activity。当这个启动画面可见时,后台线程获取活动货币代码。后台线程成功结束之后,启动画面 Activity 调用主 Activity 并将活动货币代码传递给它。主 Activity 接着使用活动货币代码显示其选择列表。即使网络质量不好,启动画面 Activity 也只会显示约一秒钟。



图 9-1 货币图片启动画面

如果用户之前选择了本国货币和外国货币,我们会从共同偏好中获取这些值并应用到选择列表中(参见图 9-2)。例如,如果最后的货币组合是将 HKD 作为外国货币,将 USD 作为本国货币,那么下次用户启动 App 时,这些值将被应用到选择列表中。一种临界情况是共同偏好中保存的本国货币和/或外国货币已经不再交易了。在这个场景中,受影响的选择列表将会显示数据中的第一个货币代码。

当主 Activity 可见时,焦点被设置在位于主 Activity 顶部的 EditText 控件上。这个 EditText 控件只接收数值数据,表示待转换的外国货币金额。从选择列表中选取外国货币和本国货币,并输入想要转换的金额之后,用户可以单击 Calculate 按钮,它会启动一个后台线程获取当前汇率。当后台线程处于活动状态时,用户会看到一个显示“稍等片刻”的对话框(参见图 9-3);这个对话框允许用户通过单击 Cancel 按钮来终止操作。一旦后台线

程成功结束，openexchangerates.org 会返回一个 JSON 对象，其中包含所有活动货币代码相对于美元的汇率。接着抽取所需的值并计算结果。结果值被格式化为 5 位小数并显示在主 Activity 底部的 TextView 控件中，如图 9-4 所示。

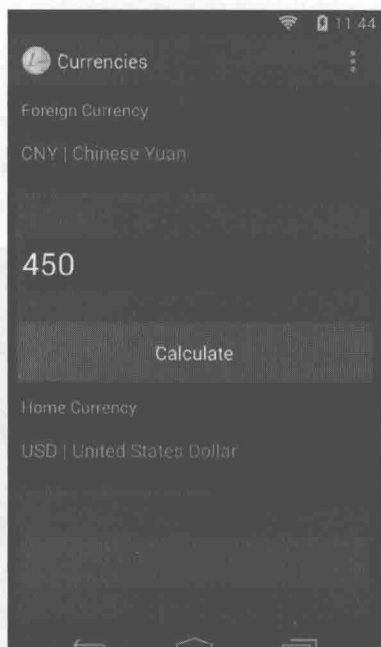


图 9-2 输入货币金额



图 9-3 计算结果

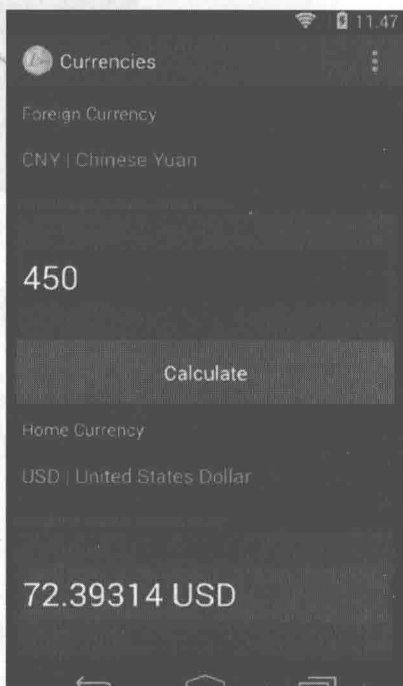


图 9-4 返回结果

Currencies App 的操作栏中有个菜单，其中有三个选项：view active codes、invert codes 和 exit(参见图 9-5)。view active codes 选项会启动浏览器并打开 openexchangerates.org/api/currencies.json。invert codes 选项会将选择列表中显示的本国货币值和外国货币值互换。例如，如果外国货币是 CNY，本国货币是 USD，单击了 invert codes 菜单选项之后，外国货币将会是 USD，而本国货币将会是 CNY。exit 选项可以退出 App。即使我们使用相同的输入值 450，图 9-4 和 9-5 中显示的结果(72.39314 USD 和 72.44116 USD)也会稍有不同。不同的原因是从 openexchangerates.org 获取到的汇率每分钟都在变化，而这两个截屏中的结果是相隔几分钟得到的。

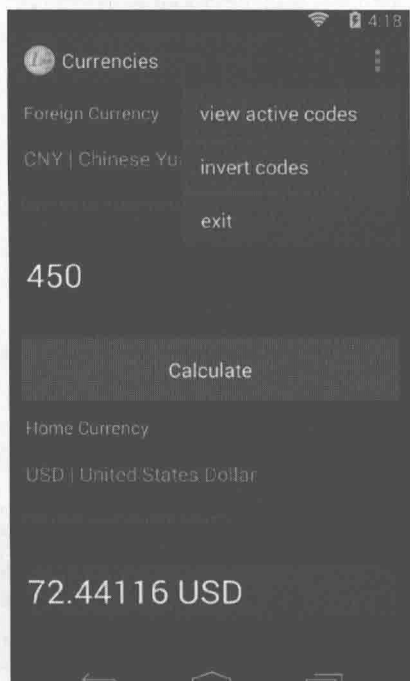


图 9-5 计算结果反映汇率在变化

既然已经了解了对 Currencies App 期望的功能，那就让我们通过选择 File | New Project 来创建一个新项目(第 1 章中涵盖了关于 New Project Wizard 及其界面的内容)。将你的 App 命名为 Currencies。选择使用 gerber.apress.com 作为域名，但也可以输入自己喜欢的任何域名。Android(和 Java)中的约定是将倒序的域名作为包名。你将会注意到包名是倒序域名加上全部小写的项目名。与本书中的其他实验和练习相同，可以将此实验保存在 C:\androidBook\Currencies 目录中。如果正在运行 Mac，请将 Currencies App 置于你的实验父目录中。单击 Next 按钮。

向导中的下一步是选择目标 API 级别。这里存在折中——让 App 兼容尽可能多的设备(将目标 API 设置为较低的值)，或是增加对于开发者可用的特性数量(将目标 API 设置为较

高的值)。不过,这种折中更倾向于将目标 API 设置为较低的值,因为 Google 给出了优秀的兼容库,提供了包含在后续 API 中的大多数功能。开发商用 Android App 的最佳实践是选择让 App 能够在接近 100%的设备上运行的最高目标 API 级别。目前,该目标 API 级别为 API 8。请注意,Android Studio 将会自动导入适当的兼容库(也称为支持库)。API 8: Android 2.2 (Froyo)应该被默认选中。如果尚未选择,那么选择 API 8: Android 2.2 (Froyo)并单击 Next 按钮。

向导的下一步是选择将要生成的 Activity 类型。选择 Blank Activity 并单击 Next 按钮。如果默认值与图 9-6 中所示的不同,那么按图 9-6 所示设置它们。单击 Finish, Android Studio 将会为你生成一个新的项目。Gradle(与 Android Studio 绑定的构建工具,将在第 13 章中介绍)将会开始下载所有的依赖,例如兼容库。留意状态栏,观察这些过程的状态。一旦完成这些处理,你应该会得到一个没有任何错误的新项目。

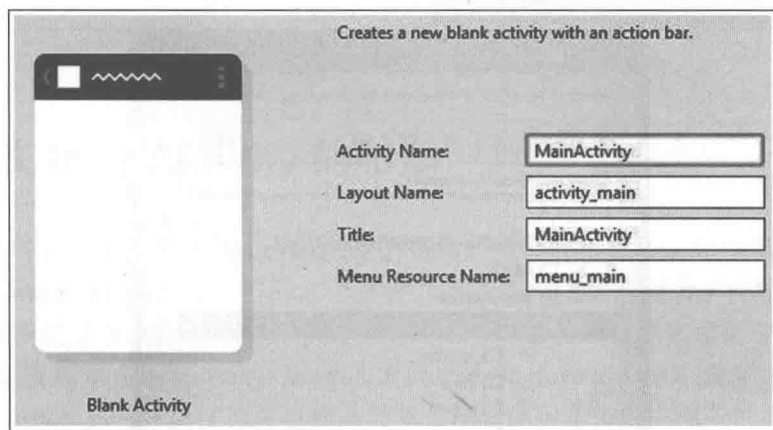


图 9-6 Create New Project 对话框

9.2 初始化 Git 仓库

Git 已经成为 Android App 开发中不可或缺的工具,这里向你展示如何初始化 Android 项目的 Git 仓库。关于如何使用 Git 的更详细教程,请参阅第 7 章。选择 VCS | Import into Version Control | Create Git Repository, 如图 9-7 所示。当提示选择在哪个目录中创建新的 Git 仓库时,确保在项目的根目录中初始化 Git 项目,本例中项目的根目录为 Currencies,位于 C:\androidBook\Currencies,如图 9-8 所示。如果使用的是 Mac 电脑,那么在实验父目录中选择 Currencies 目录,单击 OK 按钮。

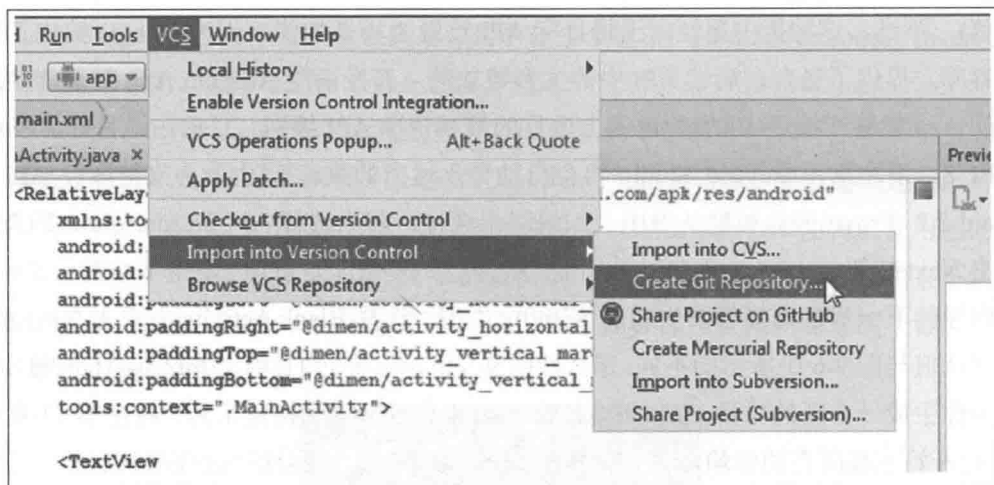


图 9-7 初始化 Git 仓库

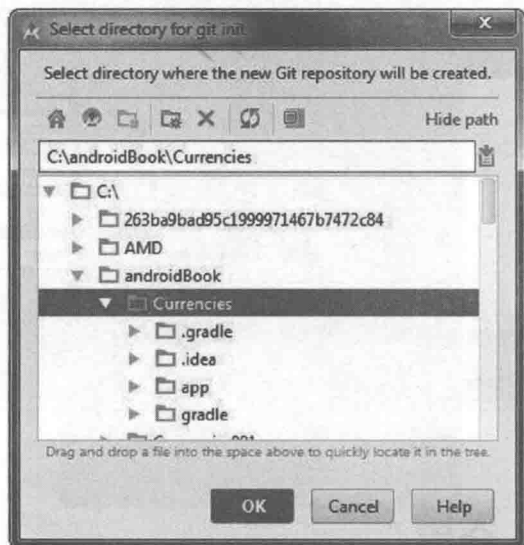


图 9-8 选择用于 Git 初始化的目录

务必将 Project 工具窗口切换至 Project 视图。视图组合框位于 Project 工具窗口的顶部，默认被设置为 Android。如果查看 Project 工具窗口中的文件，你将会发现这些文件中的大多数都变成了棕色，这意味着 Git 跟踪了它们，但尚未添加到仓库中。要添加它们，请选择 Project 工具窗口中的 Currencies 目录并按 **Ctrl + Alt + A** | **Cmd + Alt + A**，或者单击 **Git | Add**。棕色的文件应该会变为绿色，这意味着它们已经被添加到了 Git 中而且现在可以提交了。虽然这个添加并编辑资源的过程很烦琐，但是这些操作只需要做一次；此后，Android Studio 将会自动管理文件的添加和编辑。

按 **Ctrl + K** | **Cmd + K** 打开 Commit Changes 对话框，如图 9-9 所示。Author 组合框用于覆盖当前的默认提交者。如果将 Author 组合框留空，那么 Android Studio 将会使用在 Git 安装时设置的初始默认值。在 Before Commit 区域，取消选中所有的复选框。在 Commit

Message 字段中输入以下消息：Initial commit using new project wizard。两次单击 Commit 按钮。通过按 Alt + 9 | Cmd + 9 调用 Changes 工具窗口可以查看你的提交。

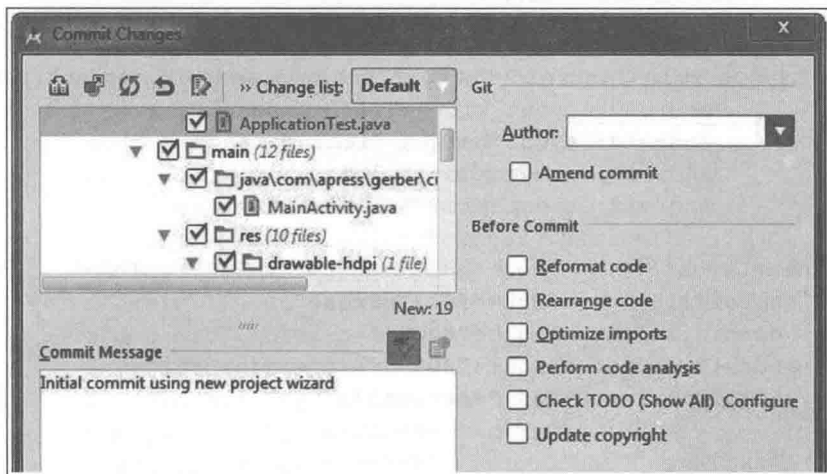


图 9-9 使用 Commit Changes 对话框提交初始修改

9.3 修改 MainActivity 的布局

在这一节中，我们将修改 MainActivity 的布局。New Project Wizard 为我们创建了一个名为 activity_main.xml 的文件。打开这个文件并参考图 9-2(如前面所示)和代码清单 9-1。图 9-2 中的视图是竖直排列的，因此竖直方向的 LinearLayout 看起来是根 ViewGroup 的首选。视图的宽度需要充满父 ViewGroup，因此将所有 layout_width 设置为 fill_parent。fill_parent 和 match_parent 属性值可以相互替换使用。在指定布局中视图的高度时，我们想要尽量避免使用硬编码的 dp(密度无关像素)值。相反，此处将要使用名为 layout_weight 的属性，要求 Android Studio 根据视图占父 ViewGroup 的百分比来渲染其高度。

layout_weight 属性用于计算某个子视图占父 ViewGroup 中所有子视图 layout_weight 值之和的比例。例如，假定在竖直方向的 LinearLayout 中嵌入了一个 TextView 和一个 Button。如果 TextView 的 layout_weight 为 30，而 Button 的 layout_weight 为 70，那么 TextView 将占据其父布局高度的 30%，而 Button 将占据其父布局高度的 70%。为了简化我们的任务，假定 layout_weight 的总和为 100，以便以百分数的形式表示所有 layout_weight 值。这个技巧的唯一问题在于 layout_height 是 Android 视图的必需参数，因此我们必须将 layout_height 设置为 0 dp。将 layout_height 设置为 0 dp 之后，Android 会忽略 layout_height 并转而使用 layout_weight。

在查看这个布局中的视图时，注意某些视图有 ID，而另外一些则没有。仅当 Java 代码中将要引用某个视图时，为其赋予 ID 值才是有用的。如果视图在用户使用过程中一直是静态的，那就无须为其指定 ID。当重新创建代码清单 9-1 中的布局时，请注意 id 的使用，以及 layout_weight 和 layout_height 的用法。选中 activity_main.xml 标签页之后，你将会在底部看到两个额外的选项卡——Design 和 Text。单击 Text 选项卡，输入代码清单 9-1 中的

代码。务必完全替换 activity_main.xml 中的全部已有 XML。

代码清单 9-1 activity_main.xml 的代码

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#000"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="20"
        android:orientation="vertical">

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="30"
            android:gravity="bottom"
            android:text="Foreign Currency"
            android:textColor="#ff22e9ff"/>

        <Spinner
            android:id="@+id/spn_for"
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="55"
            android:gravity="top"/>

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="15"
            android:gravity="bottom"
            android:text="Enter foreign currency amount here:"
            android:textColor="#666"
            android:textSize="12sp"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="fill_parent"
```

```

    android:layout_height="0dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_weight="20"
    android:background="#222">

    <EditText
        android:id="@+id/edt_amount"
        android:layout_width="fill_parent"
        android:layout_height="50dp"
        android:layout_gravity="center_vertical"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp"
        android:background="#111"
        android:digits="0123456789."
        android:gravity="center_vertical"
        android:inputType="numberDecimal"
        android:textColor="#FFF"
        android:textSize="30sp">

        <requestFocus/>
    </EditText>
</LinearLayout>

<Button
    android:id="@+id/btn_calc"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_weight="10"
    android:text="Calculate"
    android:textColor="#AAA"/>

<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_weight="20"
    android:orientation="vertical">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_weight="30"
        android:gravity="bottom"
        android:text="Home Currency"
        android:textColor="#ff22e9ff"/>

```

```

<Spinner
    android:id="@+id/spn_hom"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_weight="55"
    android:gravity="top"/>

<TextView
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_weight="15"
    android:gravity="bottom"
    android:text="Calculated result in home currency:"
    android:textColor="#666"
    android:textSize="12sp"/>
</LinearLayout>

<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_weight="20"
    android:background="#222">

    <TextView
        android:id="@+id/txt_converted"
        android:layout_width="fill_parent"
        android:layout_height="50dp"
        android:layout_gravity="center_vertical"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp"
        android:background="#333"
        android:gravity="center_vertical"
        android:textSize="30sp"
        android:typeface="normal"/>
    </LinearLayout>

</LinearLayout>

```

创建完这个布局后，按 **Ctrl + K** | **Cmd + K** 并提交，附带消息为“修改 activity_main 布局”。

9.4 定义颜色

在查看代码清单 9-1 中的 XML 源代码时，注意我们硬编码了一些属性，例如 `textColor` 和 `background`。将颜色值提取到外部资源文件中是一个好主意，尤其是当颜色出现重复时。一旦将颜色提取出来，只需要在资源文件中修改一个值，就可以改变整个应用中的颜色。在第 5 章中，我们向你展示了如何使用 `IntelliSense` 创建颜色定义。在此将以颜色定义作为开始并替换硬编码值。使用对你来说最简单的方法。右击(在 Mac 上按住 `Ctrl` 键并单击)`res/values` 目录并选择 `New | Values`。将文件命名为 `colors` 并单击 `OK` 按钮。如果提示你向 `Git` 添加文件，那么选中 `Remember, Don't Ask Again` 复选框并单击 `OK` 按钮。按照代码清单 9-2 中的代码修改 `colors.xml` 文件。

代码清单 9-2 在 `colors.xml` 中定义一些颜色

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

    <color name="white">#FFF</color>
    <color name="black">#000</color>

    <color name="grey_very_dark">#111</color>
    <color name="grey_dark">#222</color>
    <color name="grey_med_dark">#333</color>
    <color name="grey_med">#666</color>
    <color name="grey_light">#AAA</color>

    <color name="turquoise">#ff22e9ff</color>
    <color name="flat_blue">#ff1a51f4</color>

</resources>
```

在 `Android` 中，颜色以十六进制数值表示。十六进制数会用到以下字母数字值：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E 和 F。十进制和十六进制数从 0 到 9 都是一样的，但为了在十六进制中表示 10、11、12、13、14、15，人们分别使用了 A、B、C、D、E 和 F。十六进制数字不区分大小写，因此 F 与 f 是一样的。

在 `Android` 中，可以采用 4 种方式表示颜色：`#ARGB`、`#RGB`、`#AARRGGBB` 或 `#RRGGBB`；每个字母都是一个十六进制数。`#ARGB` 格式代表 Alpha、红、绿和蓝通道，其中 Alpha 是透明度通道。这种颜色方案中可能的颜色数是 16 种可能的透明度值 $\times 16 \times 16 \times 16$ 种可能的颜色组合。`#RGB` 格式代表红、绿、蓝通道，而 Alpha 通道被自动设置为完全不透明。`#AARRGGBB` 和 `#RRGGBB` 格式使用 8 比特通道，而非 `#ARGB` 和 `#RGB` 格式中使用的 4 比特通道。`#AARRGGBB` 格式中可能的颜色组合数是 256 种可能的透明度级别 $\times 256 \times 256 \times 256$ 种可能的颜色组合。`#RRGGBB` 类似于前面的格式，只是透明度级别被自动设置为完全不透明。

`colors.xml` 文件中的 `<color name="grey_med">#666</color>` 一行使用了 `#RGB` 格式。显

然，红、绿和蓝三色值相等的颜色是灰色。colors.xml 文件中的<color name="turquoise">#ff22e9ff</color>一行使用了#AARRGGBB 格式。我们可以看到青绿色的定义中包含大量蓝和绿，以及非常少量的红。如果单击 XML 文件折叠线中的任意颜色标记，我们将会看到一个能够定义任意颜色的“选择颜色”对话框，该对话框返回的字符串总是采用最精确的格式——#AARRGGBB，参见图 9-10。定义完颜色之后，按 Ctrl + K | Cmd + K 并提交，附带消息为“定义一些颜色”。

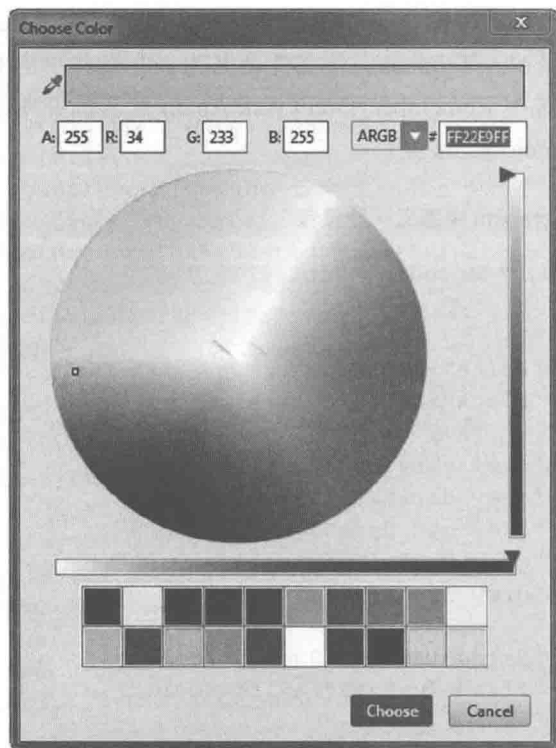


图 9-10 Choose Color 对话框

9.5 为布局应用颜色

既然已经在 colors.xml 文件中定义了颜色，现在就可以在布局中应用它们了。一种实现方法是使用 Android Studio 中的 Find/Replace 功能。创建颜色值的另外一种方法参见第 5 章。在 Editor 中以标签页的形式打开 activity_main.xml 和 colors.xml。右击(在 Mac 上按住 Ctrl 键并单击)colors.xml 标签页并选择 Move Right，以便能够并排地看到两个文件。将光标置于 activity_main.xml 选项卡中并按 Ctrl + R | Cmd + R 键。在 Find 框中输入#FFF，并在 Replace 框中输入@color/white。选中 Words 复选框，接着单击 Replace all 按钮。对所有已定义的颜色重复此步骤，除了 flat_blue 以外，我们将会在后面用到它。可以看到图 9-11 中展示的此过程。应用了颜色之后，按 Ctrl + K | Cmd + K 并提交，附带消息为“为布局应用颜色”。关闭 colors.xml 选项卡。

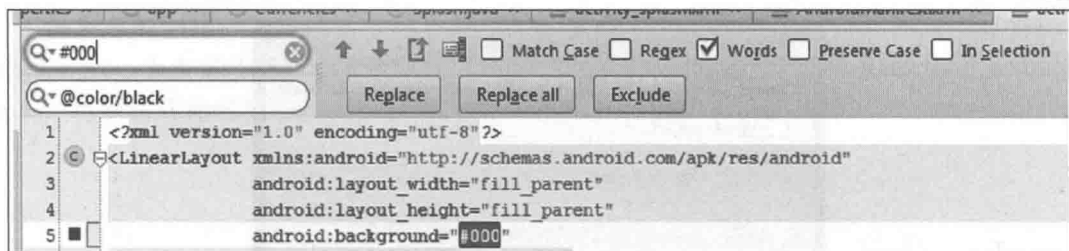


图 9-11 在 colors.xml 文件中使用命名引用替换硬编码的颜色值

9.6 创建并应用样式

样式可以极大地提升效率。从长期来看，创建样式的短期投入将会为你节省大量时间，同时提供极大的灵活性。在本节中，我们将要为 activity_main.xml 布局中的一些视图创建样式并向你展示如何应用它们。

我们采用的布局本身就适合使用样式，因为视图中有大量重复属性。例如，这两个青绿色的 TextView 控件除了文本以外的所有属性都相同。我们可以将这些重复的属性抽取为样式，然后为适当的 TextView 元素应用该样式。如果后期想要修改样式，我们只需要修改一次，使用了该样式的所有视图就都会相应改变。样式是有用的，但也没有必要过度使用并为所有视图应用样式。例如，为 Calculate 按钮创建样式就没有太大意义，因为它只有一个。

我们的第一个任务是为 activity_main.xml 布局中使用的标签(TextView)创建样式。将光标置于第一个 TextView(文本属性为 Foreign Currency)定义中的任意位置，从主菜单中选择 Refactor | Extract | Style。

在 Extract Android Style 对话框中，选中如图 9-12 所示的复选框。在 Style Name 框中输入 label。确保选中了 Launch 复选框并单击 OK 按钮。如图 9-13 所示，在后续的 Use Style Where Possible 对话框中，选中 File 单选按钮，然后单击 OK 按钮。现在单击 Find 工具窗口中的 Do Refactor(位于 IDE 的底部)，将此样式应用到其他三个共享这些属性的视图中。

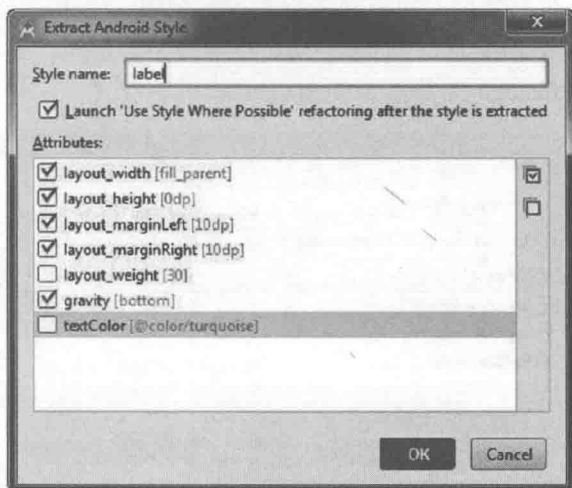


图 9-12 抽取名为 label 的样式

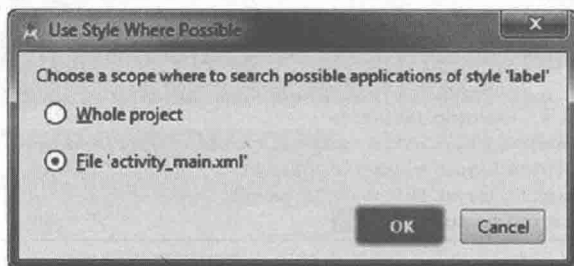


图 9-13 Use Style Where Possible 对话框

样式最优秀的特色之一是它们可以继承由你或 Android SDK 定义的父样式。将光标保持在相同 TextView 控件的定义中，再次选择 Refactor | Extract | Style。

你将会注意到为你提供的样式名称以 label 开头。label 后面的点意味着这个新的样式将继承名为 label 的父样式。将样式命名为 label.curr，如图 9-14 所示，单击 OK 按钮。再次单击 Do Refactor。

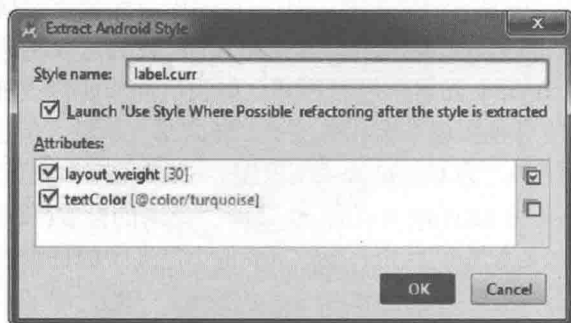


图 9-14 抽取名为 label.curr 的样式

在 activity_main.xml 文件中，导航至标签为 Enter foreign currency amount here: 的 TextView。将光标置于此视图定义括号内的任意位置，并从主菜单中选择 Refactor | Extract | Style。Android Studio 足够智能，知道文本不大可能相同，因而从 Extract Android Style 对话框中删掉了它。将此样式重命名为 label.desc 并单击 OK 按钮，如图 9-15 所示。同样，单击 IDE 底部的 Do Refactor，为第二个 TextView 应用该样式。

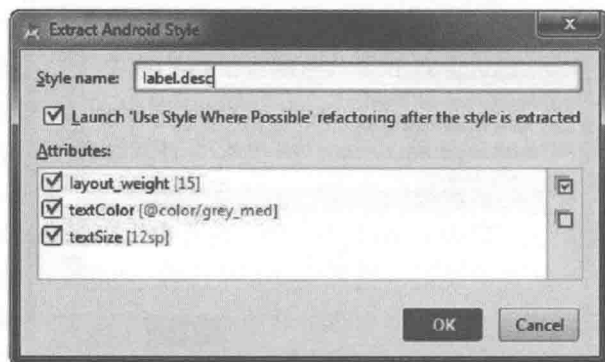


图 9-15 抽取名为 label.desc 的样式

再为布局创建一个样式，让输入框和输出框都具有灰色背景。将光标置于背景色为 `@color/grey_dark` 的 `LinearLayout` 定义中的任意位置。从主菜单中选择 `Refactor | Extract | Style`，将新样式命名为 `layout_back`，如图 9-16 所示，并单击 `OK` 按钮。

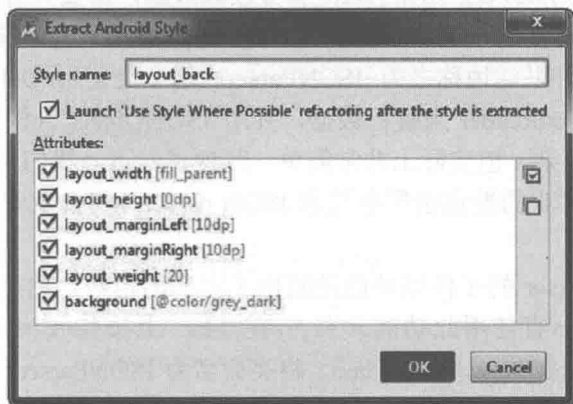


图 9-16 抽取名为 `layout_back` 的样式

在 `Use Style Where Possible` 对话框中选中 `File` 单选框并单击 `OK` 按钮。现在单击 `Do Refactor`，将样式应用到布局的第二次出现之处。

按 `Ctrl + Shift + N` | `Cmd + Shift + O`，输入 `styles`，选择 `res/values/styles.xml` 文件并在 `Editor` 中以标签页的形式打开它。你应该会看到类似于图 9-17 中所示的样子。按 `Ctrl + K` | `Cmd + K` 并提交，附带消息为“创建并应用样式”。

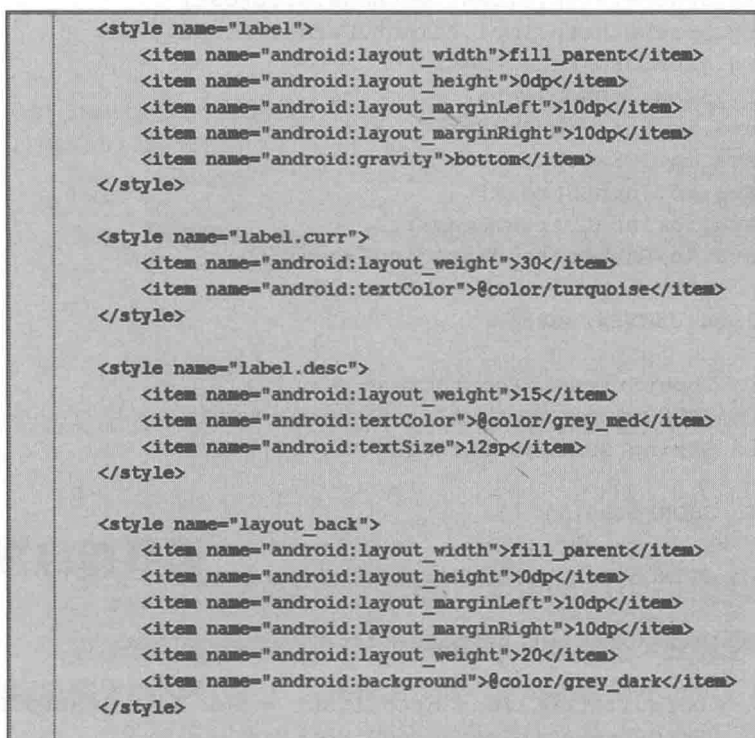


图 9-17 `styles.xml` 文件中自动创建了样式

9.7 创建 JSONParser 类

要从 openexchangerates.org 这个 Web 服务读取数据，需要一种解析 JSON 的方法。JSON，即 JavaScript 对象表示法，已经成为 Web 服务事实上的标准格式。我们创建了自己的 JSON 解析器，并贴切地称之为 JSONParser。这个类使用 DefaultHttpClient 来填充 InputStream、用 BufferedReader 来解析数据，并用 JSONObject 来构造和返回 JSONObject 对象。虽然听起来很复杂，但实际上非常简单。顺便说一下，我们不是唯一使用 JSON 解析器的人；如果你喜欢的搜索引擎中搜索 JSON parser，将会发现这个基础模式的大量实现。

详细介绍 JSONParser 的工作原理已经超出了本书的范畴。然而，请在项目中添加这个类，因为我们将一直使用此功能。右击(在 Mac 上按住 Ctrl 键并单击)com.apress.gerber.currencies 包并选择 New | Java Class。将类命名为 JSONParser。在这个类中输入代码清单 9-3 中的代码。

代码清单 9-3 JSONParser.java 代码

```
package com.apress.gerber.currencies;
import android.util.Log;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;

public class JSONParser {

    static InputStream sInputStream = null;
    static JSONObject sReturnJsonObject = null;
    static String sRawJsonString = "";

    public JSONParser() {}

    public JSONObject getJSONFromUrl(String url) {

        //attempt to get response from server
        try {
            DefaultHttpClient httpClient = new DefaultHttpClient();
            HttpPost httpPost = new HttpPost(url);
```

```

        HttpResponse httpResponse = httpClient.execute(httpPost);
        HttpEntity httpEntity = httpResponse.getEntity();
        sInputStream = httpEntity.getContent();

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    //read stream into string-builder
    try {
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            sInputStream, "iso-8859-1"), 8);
        StringBuilder stringBuilder = new StringBuilder();
        String line = null;
        while((line = reader.readLine()) != null) {
            stringBuilder.append(line + "\n");
        }
        sInputStream.close();
        sRawJsonString = stringBuilder.toString();
    } catch (Exception e) {
        Log.e("Error reading from Buffer: " + e.toString(),
            this.getClass().getSimpleName());
    }

    try {
        sReturnJsonObject = new JSONObject(sRawJsonString);
    } catch (JSONException e) {
        Log.e("Parser", "Error when parsing data " + e.toString());
    }

    //return json object
    return sReturnJsonObject;
}
}

```

在输入或粘贴前面的代码之后，按 **Ctrl + K** | **Cmd + K** 并提交修改，附带消息为“创建 **JSONParser** 类”。

9.8 创建启动界面

在本节中，我们将要创建启动界面。这个 **Activity** 的功能是为我们争取大约一秒钟的时间，以便获取活动货币代码。在后台线程工作时，我们将要显示一张货币的照片。如果这是一款商用 App，我们可能会显示带有公司标志的图片以及 App 的名字。

右击(在 Mac 上按住 **Ctrl** 键并单击)com.apress.gerber.currencies 包并选择 **New | Activity |**

Blank Activity。将你的 Activity 命名为 SplashActivity，并选中 Launcher Activity 复选框，如图 9-18 所示。

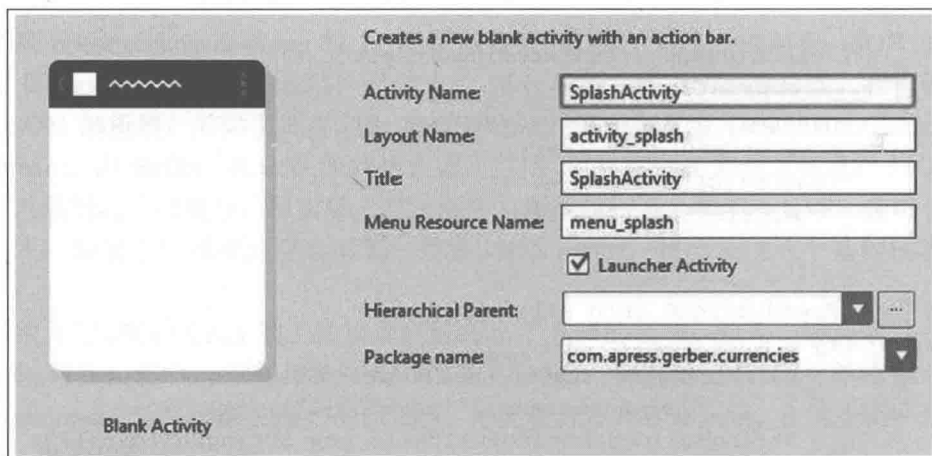


图 9-18 通过 New | Activity | Blank Activity 来创建 SplashActivity

在新创建的 SplashActivity.java 文件中修改类定义，使 SplashActivity 派生自 Activity 而非 ActionBarActivity。同样，在 onCreate() 方法中加入 this.requestWindowFeature(Window.FEATURE_NO_TITLE);，如代码清单 9-4 所示，并处理导入。

代码清单 9-4 将 SplashActivity 修改为派生自 Activity 并删除标题栏

```
public class SplashActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_splash);
    }
    ...
}
```

按 Ctrl + Shift + N | Cmd + Shift + O 并输入 And。选择并打开 app/src/main/AndroidManifest.xml 文件，将该文件修改为代码清单 9-5 所示的样子。

代码清单 9-5 修改 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.apress.gerber.currencies" >

    <uses-permission android:name="android.permission.INTERNET">
    </uses-permission>

    <application
        android:allowBackup="true"
```



```

    android:icon="@android:color/transparent"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:icon="@mipmap/ic_launcher"
        android:name=".MainActivity"
        android:label="@string/app_name" >

    </activity>
    <activity
        android:name=".SplashActivity"
        android:label="@string/title_activity_splash">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

```

我们在 `AndroidManifest.xml` 文件中添加的 `uses-permission` 代码行允许 App 获取对 Internet 的访问权限。此外，我们将 App 自身的 `icon` 属性设置为透明以确保 `SplashActivity` 优先显示。注意现在 `SplashActivity`(而非 `MainActivity`)包含 Intent 过滤器 `main/launcher`。Intent 过滤器 `main/launcher` 告知 Android 操作系统最先启动哪个 Activity。

我们需要在启动画面中显示一些免费图片。在浏览器中打开 `google.com/advanced_image_search`。在 All These Words 框中输入 `currencies`。在 Usage Rights 框中选择 Free to Use, Share or Modify, Even Commercially。单击 Advanced Search，找一张自己喜欢的图片并下载。将 `image` 命名为 `world_currencies.jpg`(或 `world_currencies.png`，如果文件是 PNG 格式的话)。将 `world_currencies.jpg` 复制粘贴到 Project 工具窗口的 `res/drawable` 目录中。修改 `activity_splash.xml` 文件，使其类似于代码清单 9-6 所示。

代码清单 9-6 修改 `activity_splash.xml` 文件，将 `world_currencies` 显示为背景

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/world_currencies"
    android:orientation="vertical">

</LinearLayout>

```

按 `Ctrl + K` | `Cmd + K` 并提交，附带消息为 `Creates splash activity and makes it the launched activity`。

9.9 获取 JSON 格式的活动货币代码

在前面的小节中，你让 `SplashActivity` 成为第一个启动的 `Activity` 并将其布局修改为显示 `world_currencies` 图片。在这一节中，你将要修改 `SplashActivity`，使其启动用于从 `openexchangerates.org/api/currencies.json` 获取活动货币代码的后台线程。

按 `Ctrl + N` | `Cmd + O`，输入 `Spl`，并选择 `SplashActivity.java` 文件。`SplashActivity` 中没有菜单，因此我们可以删除那些与菜单相关的方法。删除 `onOptionsItemSelected()` 和 `onOptionsItemSelected()` 这两个方法。

我们需要创建 `AsyncTask`，将其命名为 `FetchCodesTask`，并作为 `SplashActivity.java` 的内部类。`AsyncTask` 是一个主要用于在 Android 中快速实现并行(线程)操作的类。将在第 10 章中讨论 `AsyncTask` 的架构，因此现在只要认为 `AsyncTask` 可以胜任工作即可。

首先，在 `onCreate()` 方法的下面，将 `FetchCodesTask` 定义为 `SplashActivity.java` 的内部类，类似下面这样：

```
private class FetchCodesTask extends AsyncTask<String, Void, JSONObject> {
}
```

导入所有必需的类：将光标置于红色文本上，然后按 `Alt + Enter` 并选择 `Import Class`，如图 9-19 所示。

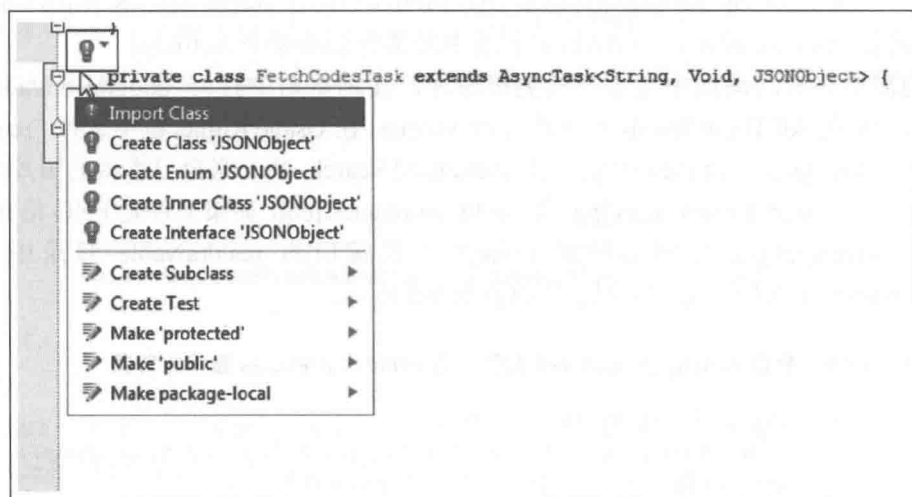


图 9-19 导入 `JSONObject` 和 `AsyncTask`

即使在处理这些导入之后，类定义仍然会有红色的下划线，表示存在编译时错误。将光标置于这个新的内部类定义中，按 `Alt + Insert` | `Cmd + N` 并选择 `Override Methods`。在出现的对话框中，按住 `Ctrl` 键 (Mac 上的 `Cmd` 键) 的同时选中 `doInBackground()` 和 `onPostExecute()` 这两个方法并单击 `OK` 按钮，如图 9-20 所示。

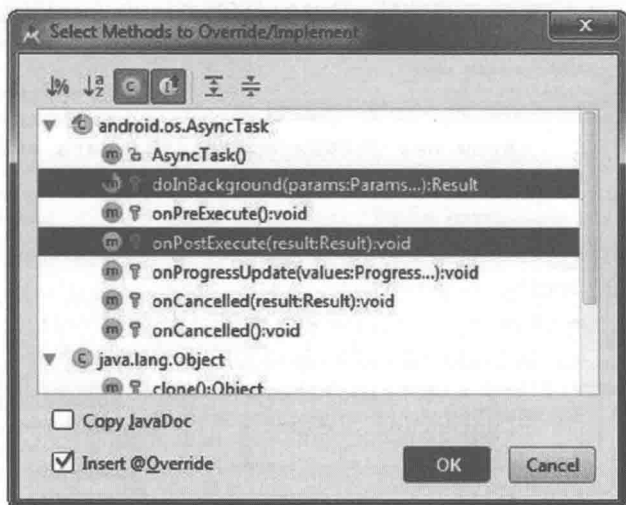


图 9-20 选中 doInBackground()和 onPostExecute()方法

注意，方法参数会根据你在内部类定义中引入的泛型来定义。将 SplashActivity.java 类修改为代码清单 9-7 的样子，并处理导入。

代码清单 9-7 修改 SplashActivity.java 文件

```
public class SplashActivity extends Activity {
    //url to currency codes used in this application
    public static final String URL_CODES =
"http://openexchangerates.org/api/currencies.json";
    //ArrayList of currencies that will be fetched and passed into MainActivity
    private ArrayList<String> mCurrencies;
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_splash);
        new FetchCodesTask().execute(URL_CODES);
    }

    private class FetchCodesTask extends AsyncTask<String, Void,
JSONObject> {

        @Override
        protected JSONObject doInBackground(String... params) {
            return new JSONObject().getJSONObject(params[0]);
        }

        @Override
```

```

protected void onPostExecute(JSONObject jsonObject) {

    try {
        if(jsonObject == null) {
            throw new JSONException("no data available.");
        }

        Iterator iterator = jsonObject.keys();
        String key = "";
        mCurrencies = new ArrayList<String>();
        while(iterator.hasNext()) {
            key = (String)iterator.next();
            mCurrencies.add(key + " | " + jsonObject.getString(key));
        }
        finish();

    } catch (JSONException e) {

        Toast.makeText(
            SplashActivity.this,
            "There's been a JSON exception: " + e.getMessage(),
            Toast.LENGTH_LONG

        ).show();

        e.printStackTrace();
        finish();

    }

}
}

```

单击代码行 `mCurrencies.add(key + " | " + jsonObject.getString(key));` 旁边的折叠线, 在该行设置断点。单击工具栏中的 **Debug** 按钮(看上去像只虫子的那个按钮)。等待项目构建并加载 **Debugger**。当到达断点时, 单击几次 **Resume**(**Debug** 窗口中的绿色向右箭头)按钮。如果在 **Debug** 窗口中打开了 `mCurrencies`, 你将会发现获取到的值没有特定顺序, 参见图 9-21。在 **Debugger** 窗口中, 单击看上去像个红色方框的停止按钮。既然对于获取到的数据已经很满意了, 就按 `Ctrl + K` | `Cmd + K` 并提交, 附带消息为 `Fetches codes as json from openexchangerates.org`。

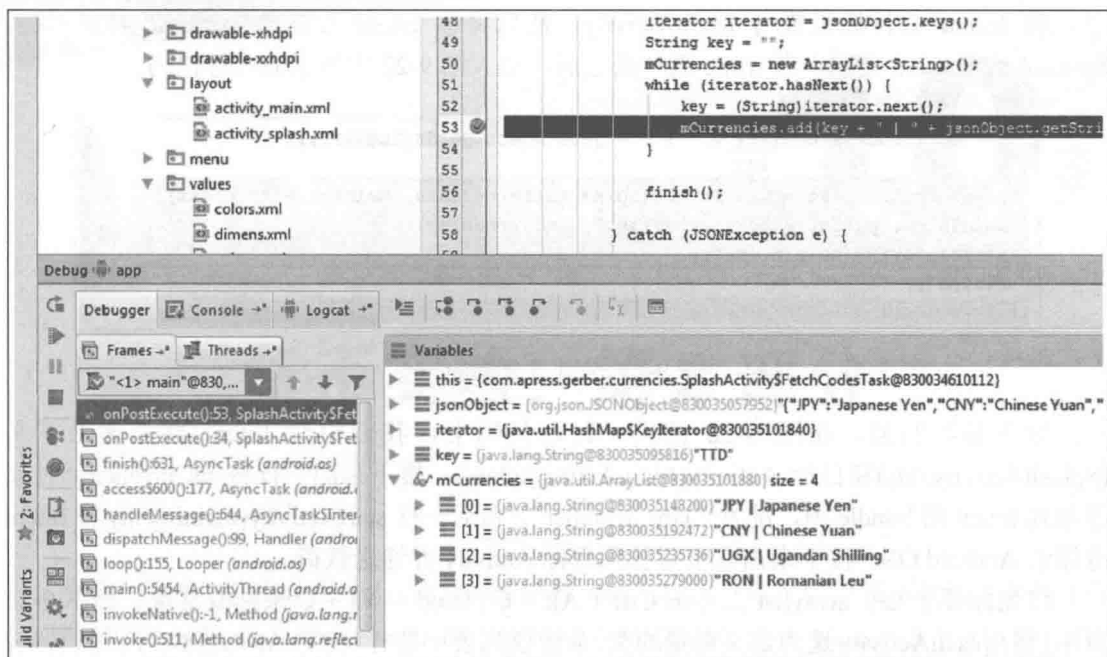


图 9-21 在 Debug 窗口中逐帧查看 mCurrencies

9.10 启动 MainActivity

在上一节中，你成功地使用 `AsyncTask` 获取到了活动货币代码。现在需要启动 `MainActivity` 并将活动货币代码传给它。

Android 的软件架构具有极度开放和模块化的特点。模块化是一个非常好的特性，因为我们可以将任意数量的第三方 App 集成进来。不过，模块化也存在问题，因为不同的 App 并不共享相同的内存空间，因此我们不能简单地来回传递对象引用。Android 通过为每个 Activity 构建一道“长城”来增强这种模块化特性，对象引用无法通过它。和 App 之间的通信一样，App 内部通信也采用仅以值传递的规则。即使 `SplashActivity` 和 `MainActivity` 位于相同 App 的同一个包中，我们也仍然需要序列化这两个组件之间的所有通信内容，就好像两个 Activity 位于不同的服务器上一样；这是我们在开放且模块化软件架构中进行开发所需付出的代价。

在 Android 中，使用值进行数据传递是由一个称为 `Intent` 的特殊类实现的。`Intent` 是向 Android 操作系统派发的消息。无法直接在 Activity 之间发送 `Intent`；Android OS 总会中转 Activity 之间的通信，而这也是必须在 `AndroidManifest.xml` 文件中列出所有 Activity 的原因。`Intent` 还可以包含称为 `bundle` 的净荷。`bundle` 是键/值对的映射，其中键是字符串，而值是 Java 基本数据类型或可序列化的对象。在 `Intent` 的 `bundle` 中装载了数据之后，Android OS 会分派 `Intent`——将 `Intent` 及其载荷递送到目标 Activity。

我们想要从 `SplashActivity` 传向 `MainActivity` 的数据只是一个字符串列表。幸运的是，`ArrayList` 已经实现了 `Serializable` 接口，因此我们将 `mCurrencies` 对象放入 `Intent` 的 `bundle`

中。将 Intent 的目标设置为 MainActivity，然后将该 Intent 分派给 Android OS。打开 SplashActivity.java 文件。在 while 循环块之后，加入图 9-22 中所示的三行代码。

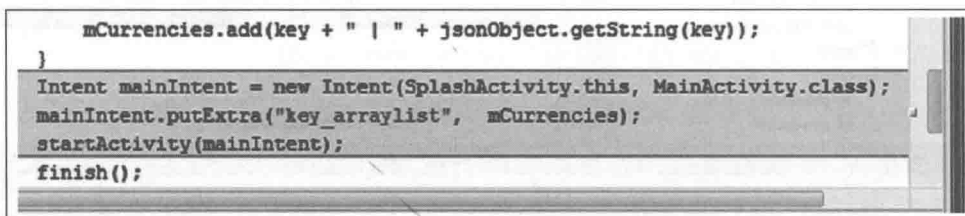


图 9-22 创建并分派 Intent

导入必需的类。在图 9-22 的第一行新代码中，我们构造 Intent 并传递上下文 (SplashActivity.this) 和目标 Activity (MainActivity.class)。接下来的一行将 mCurrencies 对象添加到 Intent 的 bundle 中，键为 “key_arraylist”。最后一行 startActivity(mainIntent); 将 Intent 分派给 Android OS，接下来由它负责找到目标 Activity 并递送载荷。

将光标置于 key_arraylist 上并按 Ctrl + Alt + C | Cmd + Alt + C 来抽取常量。如图 9-23 所示，将 SplashActivity 选为定义常量的类，从建议列表中选择 KEY_ARRAYLIST 并按 Enter 键在这个类中创建常量。

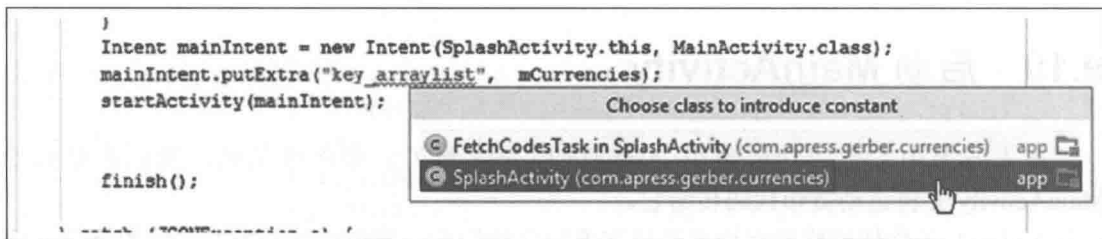


图 9-23 将 SplashActivity 选为用于定义常量的类

按 Ctrl + K | Cmd + K 并提交，附带消息为 Fires-up MainActivity with Intent and passes ArrayList into Bundle。

9.11 小结

在本章中，我们讨论了 Currencies App 的规范并进一步实现了它的部分特性。我们定义布局、抽取颜色、创建并应用样式。我们还涵盖了 JSON 并创建了用于获取活动货币代码(显示主 Activity 中的选择列表需要这些数据)的启动画面。我们介绍了 AsyncTask 并从 Web Service 获取 JSON 数据。我们还使用 Intent 在 Activity 之间进行通信。在下一章中，我们将完成 Currencies App。

第 10 章

货币实验：第 2 部分

在前一章中，我们使用 `SplashActivity` 中的 `AsyncTask` 获取了活动货币代码。你将货币代码装载到 `bundle` 中并将该 `bundle` 绑定到目标为 `MainActivity` 的 `Intent`。最后，你将 `Intent` 派发给 `Android` 操作系统。

在本章中，你将要：继续开发 `Currencies App`，重点关注 `MainActivity` 的功能并完成该 `App`；使用 `ArrayAdapter` 把字符串数组绑定到选择列表；使用 `Android Studio` 把视图行为的处理代理到包含它们的 `Activity`；学习如何使用共同偏好以及资源；学习 `Android` 中的并发，尤其是如何使用 `AsyncTask`；修改布局并使用 `Android Studio` 生成可绘制资源。

10.1 定义 `MainActivity` 的成员

让我们首先在 `MainActivity` 类中定义 `activity_main.xml` 布局文件中对应视图的引用，然后将对象赋值给它们。打开 `MainActivity.java` 和 `activity_main.xml` 这两个文件，以便可以同时引用它们。右击(在 `Mac` 上按住 `Ctrl` 键并单击) `activity_main.xml` 标签页并选择 `Move Right`，将 `activity_main.xml` 的模式改为 `Text`。将 `MainActivity.java` 文件修改为代码清单 10-1 所示的样子，并按 `Alt + Enter` 键导入所需的类。

注意，我们仅在 `MainActivity` 里面为 `activity_main.xml` 中那些之前已经分配了 `ID` 的视图定义引用。`setContentView(R.layout.activity_main);` 语句会使用 `activity_main.xml` 中包含的视图进行填充。在 `Android` 中，单词 `inflate` 表示当 `Android` 遍历 `activity_main.xml` 布局中定义的视图时，它会将每个视图初始化为堆上的 `Java` 对象。如果 `View` 对象有 `ID`，那么 `Android` 将会把对象的内存位置与其 `ID` 关联起来。这种联系可以在自动生成的名为 `R.java` 的文件中找到，它是资源和 `Java` 源文件的桥梁。

```

public class MainActivity extends ActionBarActivity {

    //define members that correspond to Views in our layout
    private Button mCalcButton;
    private TextView mConvertedTextView;
    private EditText mAmountEditText;
    private Spinner mForSpinner, mHomSpinner;

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

        //assign references to our Views
        mConvertedTextView = (TextView) findViewById(R.id.txt_converted);
        mAmountEditText = (EditText) findViewById(R.id.edt_amount);
        mCalcButton = (Button) findViewById(R.id.btn_calc);
        mForSpinner = (Spinner) findViewById(R.id.spn_for);
        mHomSpinner = (Spinner) findViewById(R.id.spn_hom);
    }
}

```

图 10-1 定义成员并将引用赋值给这些成员

在把布局及其所有视图填充到内存空间中以后，通过调用 `findViewById()` 方法并传入 ID 值可以把这些对象赋值给我们之前定义的引用。`findViewById()` 方法返回一个 `View` 对象，它是 Android 中所有 `View` 和 `ViewGroup` 的祖先；而这也是我们需要将 `findViewById()` 的返回值转换成适当 `View` 子类的原因。按 `Ctrl + K` | `Cmd + K` 并提交，附带消息为 `Gets references to views defined in layout.`

10.2 从 bundle 中解压出货币代码

在前一章中，我们将包含 `String` 类型的 `ArrayList` 传入用于启动 `MainActivity` 的 `Intent` 的 `bundle` 中。尽管 Android 操作系统已经成功地递送了它的载荷，但我们仍然需要解压它。我们在 `SplashActivity` 中使用的数据结构是一个向量 (`ArrayList<String>`)，这意味着它可以根据需要扩大或缩小。而我们将要用于在 `MainActivity` 中保存活动货币代码的数据结构是一个简单的字符串数组，它的长度是固定的。修改数据结构的原因是我们将要使用 `ArrayAdapter` 作为选择列表的控制器，`ArrayAdapter` 使用数组而非 `ArrayList`。按照图 10-2 所示修改 `MainActivity` 类并导入所有必需的类。


```

private EditText mAmountEditText;
private Spinner mForSpinner, mHomSpinner;
private String[] mCurrencies;

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

    //unpack ArrayList from the bundle and convert to array
    ArrayList<String> arrayList = ((ArrayList<String>)
        getIntent().getSerializableExtra(SplashActivity.KEY_ARRAYLIST));
    Collections.sort(arrayList);
    mCurrencies = arrayList.toArray(new String[arrayList.size()]);

    //assign references to our Views

```

图 10-2 从 ArrayList 中解压货币代码

语句 `ArrayList<String> arrayList = ((ArrayList<String>)getIntent().getSerializableExtra(SplashActivity.KEY_ARRAYLIST));` 从用于启动此 Activity 的 Intent 的 bundle 中解压出 `ArrayList<String>`。注意，我们在 MainActivity 中使用公共常量 (`SplashActivity.KEY_ARRAYLIST`) 作为键来解压 `ArrayList<String>`，而这个键值与之前在 SplashActivity 中用于压缩 `ArrayList<String>` 的相同。还要注意，我们使用 Collection 接口排序数据，然后将 `ArrayList<String>` 转换为字符串数组。按 `Ctrl + K` | `Cmd + K` 并提交，附带消息为 `Unpack currency codes from Bundle`。

10.3 创建选项菜单

New Project Wizard 为我们创建了一个名为 `menu_main.xml` 的菜单。按 `Ctrl + Shift + N` | `Cmd + Shift + O`，输入 `main`，选择 `res/menu/menu_main.xml` 并打开它。将 `menu_main.xml` 修改为如代码清单 10-1 所示。

代码清单 10-1 修改 menu_main.xml 文件

```

<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">
    <item
        android:id="@+id/mnu_codes"
        android:orderInCategory="100"
        app:showAsAction="never"
        android:title="search active codes"/>
    <item

```

```

        android:id="@+id/mnu_invert"
        android:orderInCategory="200"
        app:showAsAction="never"
        android:title="invert codes"/>

<item
    android:id="@+id/mnu_exit"
    android:orderInCategory="300"
    app:showAsAction="never"
    android:title="exit"/>

</menu>

```

`app:showAsAction` 属性决定了菜单项的位置。将此属性设置为 `never`，表示此菜单项永远不会出现在操作栏上，而是总出现在溢出菜单中。溢出菜单由操作栏右侧三个竖直的点来表示。

`android:orderInCategory` 用于设置菜单项的次序。Android 中的约定是使用 100 的倍数，因此举例来说，我们可以使用 250 在 200 和 300 之间插入一个新的菜单项，而使用 225 可以在 200 和 250 之间插入一个新的菜单项。`orderInCategory` 属性必须是整数，因此如果开始时使用诸如 2 和 3 这样的连续值，那么将会没有插入中间值的空间，而我们也不得不重新排序整个集合。

注意，我们为每个菜单项分配了 ID 值，因此后面可以在 Java 代码中引用这些对象。打开 `MainActivity.xml` 并按照代码清单 10-2 所示修改 `onOptionsItemSelected()` 方法。

代码清单 10-2 修改 `onOptionsItemSelected()` 方法

```

public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch(id) {

        case R.id.mnu_invert:
            //TODO define behavior here
            break;

        case R.id.mnu_codes:
            //TODO define behavior here
            break;

        case R.id.mnu_exit:
            finish();
            break;

    }

    return true;
}

```

注意，除了 Exit 菜单项以外，我们在实现代码的位置均放置了 TODO。在接下来的步

骤中，我们将会实现其余的菜单项功能。按 Ctrl + K | Cmd + K 并提交，附带消息为 Creates options menu。

10.4 实现选项菜单行为

在本节中，我们将要编写一些获取权限的代码。如果你是一位 Android 用户，那么应该会对安装 App 之前需要同意一大堆权限许可的情形感到很熟悉。有些 App 需要获取比其他 App 更多的权限，但大多数 App 至少需要一个权限。在之前的步骤中，我们向用户要求获取访问 Internet 的权限。在这个步骤中，我们将要求用户授权并获得对设备网络状态的访问权限。很容易就会忽略掉权限，尤其是菜鸟 Android 程序员。幸运的是，如果你忘记加入适当的权限，那么与此问题相关的异常会很明显。

打开 AndroidManifest.xml 文件——按 Ctrl + Shift + N | Cmd + Shift + O，输入 And，然后按 Enter 键并选择 AndroidManifest.xml。修改 AndroidManifest.xml，插入图 10-3 中突出显示的行。

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" ></uses-permission>
```

图 10-3 在 AndroidManifest.xml 文件中添加访问网络状态的权限

打开 MainActivity.java 类。在代码清单 10-3 中定义三个方法。isOnline()方法检查用户是否有 Internet 连接。这个方法使用 Android ConnectivityManager，这就是我们需要在 AndroidManifest.xml 文件中添加 android.permission.ACCESS_NETWORK_STATE 的原因。launchBrowser()方法接收一个保存着统一资源标识符(URI)的字符串。URI 是统一资源定位符(URL)的超集，因此任意合法的 HTTP 或 HTTPS 地址字符串都可以作为参数。launchBrowser()方法启动设备上的默认浏览器并打开我们传给它的 URI。invertCurrencies()方法只是反转选择列表中的本国与外国货币。当然，如果 TextView 中包含之前填入数据的计算结果，那么还需要清空它，以免让人感到困惑。将你的新方法放在 onCreate()方法的下面。

代码清单 10-3 在 MainActivity.java 中创建三个方法

```
public boolean isOnline() {
    ConnectivityManager cm =
        (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = cm.getActiveNetworkInfo();
    if(networkInfo != null && networkInfo.isConnectedOrConnecting()) {
        return true;
    }
    return false;
}
```

```

    }
    private void launchBrowser(String strUri) {

        if(isOnline()) {
            Uri uri = Uri.parse(strUri);
            //call an implicit intent
            Intent intent = new Intent(Intent.ACTION_VIEW, uri);
            startActivity(intent);
        }
    }
    private void invertCurrencies() {
        int nFor = mForSpinner.getSelectedPosition();
        int nHom = mHomSpinner.getSelectedPosition();

        mForSpinner.setSelection(nHom);
        mHomSpinner.setSelection(nFor);

        mConvertedTextView.setText("");
    }
}

```

将 MainActivity.java 文件的 onOptionsItemSelected() 方法中的 TODO 替换为对代码清单 10-4 中对应方法的调用。按 Ctrl + K | Cmd + K 并提交, 附带消息为 Implements options menu behavior and modifies manifest file。

代码清单 10-4 将 onOptionsItemSelected() 方法中的 TODO 替换为对刚刚定义的方法的调用

```

case R.id.mnu_invert:
    invertCurrencies();
    break;

case R.id.mnu_codes:
    launchBrowser(SplashActivity.URL_CODES);
    break;

```

10.5 创建 spinner_closed 布局

为处于关闭状态的选择列表创建布局。右击(在 Mac 上按住 Ctrl 键并单击)res/layout 目录并选择 New | Layout Resource File。将文件命名为 spinner_closed 并单击 OK 按钮, 如图 10-4 所示。

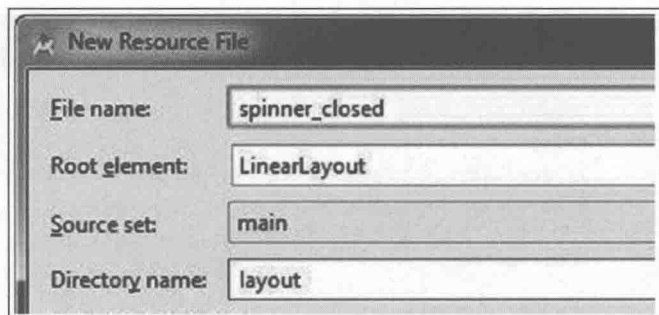


图 10-4 定义 spinner_closed 布局资源文件

修改 spinner_closed.xml，如代码清单 10-5 所示。

代码清单 10-5 spinner_close.xml 的定义

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    android:background="@color/grey_very_dark"
    android:textColor="@color/grey_light"
    android:singleLine="true"
    android:textSize="18sp"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:gravity="center_vertical"
    android:ellipsize="marquee"
/>
```

10.6 将 mCurrencies 绑定到选择列表

本国货币选择列表和外国货币选择列表将显示相同的条目。我们需要将 mCurrencies 绑定到这两个选择列表。要实现此功能，我们将使用一个名为 ArrayAdapter 的类。在 MainActivity.java 的 onCreate() 方法中，添加如图 10-5 中所示的代码并处理导入。

ArrayAdapter 构造函数接收三个参数：上下文、布局和一个数组。ArrayAdapter 作为模型-视图-控制器设计模式中的控制器，用于协调模型和视图之间的关系。我们这个示例中的模型是一个名为 mCurrencies 的字符串数组。mCurrencies 中的每个元素均包含一个货币代码、一条用于视觉分隔的管道线和一段货币描述。选择列表有两个视图：当选择列表打开时显示一个视图，而当选择列表关闭时显示另一个视图。最后两条语句将新构建的 ArrayAdapter 对象赋值给选择列表。按 Ctrl + K | Cmd + K 并提交，附带消息为 Binds data to spinners。按 Shift + F10 | Ctrl + R 运行 App，测试两个选择列表，确保它们能够正常工作。

```

//controller: mediates model and view
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(

    //context
    this,
    //view: layout you see when the spinner is closed
    R.layout.spinner_closed,
    //model: the array of Strings
    mCurrencies

);

//view: layout you see when the spinner is open
arrayAdapter.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);

//assign adapters to spinners
mHomSpinner.setAdapter(arrayAdapter);
mForSpinner.setAdapter(arrayAdapter);

```

图 10-5 将 mCurrencies 绑定到选择列表

10.7 将选择列表行为代理给 MainActivity

Java 事件模型极其灵活。我们可以将事件的处理代理到实现了相应监听器接口的任意对象。如果视图是唯一的，例如 Calculate 按钮，那么将其行为处理代理到一个匿名内部类是有意义的。然而，如果布局中包含相同类型的多个视图，比如 Currencies App 中包含两个或更多个选择列表这种情况，那么将这些视图的处理代理到包含它们的类通常是更为简便的方法。在 MainActivity.java 中 onCreate() 方法的结尾处添加两行代码，如图 10-6 所示。单词 this 下面将会出现红色下划线，表示有编译错误。

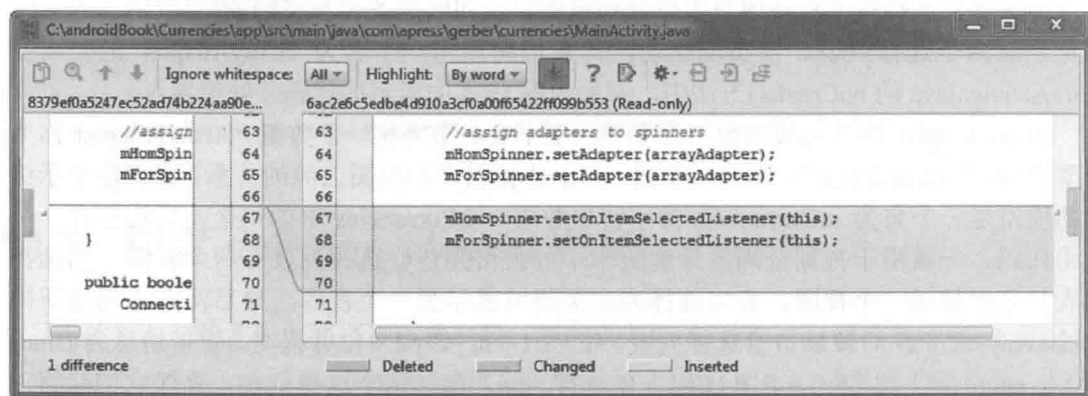


图 10-6 将选择列表的行为代理给 MainActivity

将光标置于任意一个单词 this 上，按 Alt + Enter 调用 IntelliJSense 代码补全。选择图 10-7 中所示的第二个选项(Make 'MainActivity' implement 'android.widget.AdapterView.

OnItemSelectedListener')。在 Select Methods to Implement 对话框中选中这两个方法,如图 10-8 所示,并单击 OK 按钮。如果滚动到类的顶部,将会发现 MainActivity 现在实现了 AdapterView.OnItemSelectedListener。

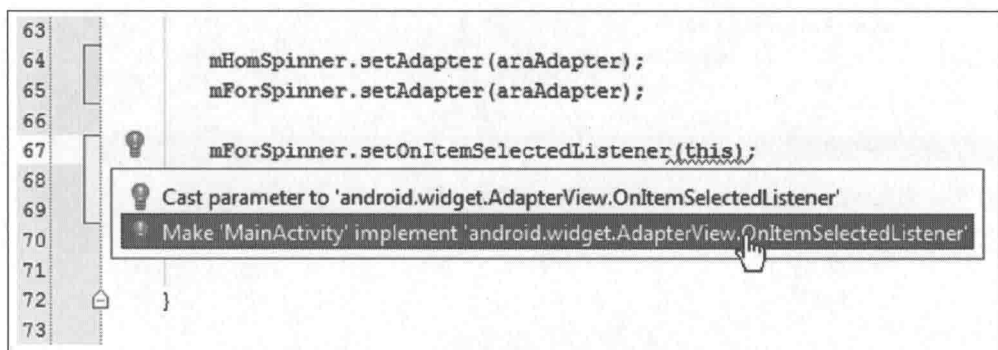


图 10-7 让 MainActivity 实现 OnItemSelectedListener

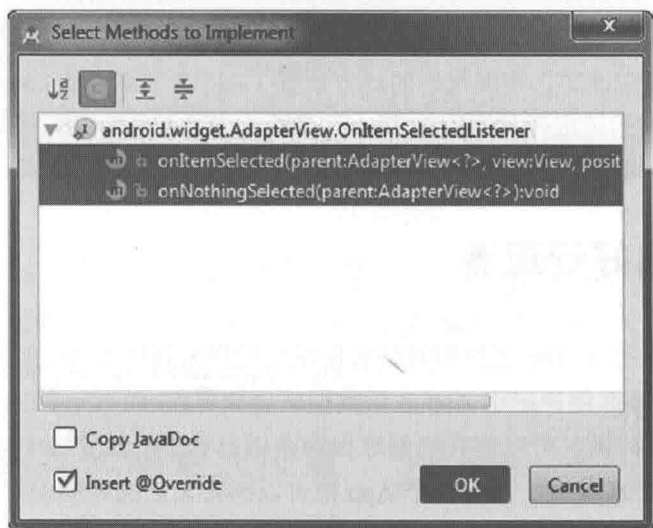


图 10-8 Select Methods to Implement 对话框

OnItemSelectedListener 接口有两个所有实现类必须重载的抽象方法: onItemSelected() 和 onNothingSelected()。在 onNothingSelected()方法的内部,我们将不提供任何实现代码。尽管 onNothingSelected()什么都没做,但它必须出现在 MainActivity 中以满足接口约定。

在 onItemSelected()方法中,我们需要通过判断 parent.getId()来确定选中了哪个选择列表,然后添加一些条件逻辑来处理所选选择列表的行为。修改 onItemSelected()方法,如图 10-9 所示。


```

@Override
public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

    switch (parent.getId()) {

        case R.id.spn_for:
            //define behavior here
            break;

        case R.id.spn_hom:
            //define behavior here
            break;

        default:
            break;
    }
}

```

图 10-9 修改 onItemSelected()方法

注意，我们在本应该加入实现代码的地方放置了占位符注释(*//define behavior here*)。我们将在后续步骤中实现选择列表的行为。按 **Ctrl + K** | **Cmd + K** 并提交，附带消息为 *Delegates handling of spinners' behavior to MainActivity*。

10.8 创建偏好管理器

共同偏好提供一种在 App 退出期间持久化保存用户偏好的方式。如果我们试图在内存中保存用户的偏好，那么用户退出 App 之后数据就会被覆盖，而 App 的内存也会被 Android OS 回收。要解决此问题，可以将共同偏好保存在用户设备上的文件中。这种文件是序列化的哈希表，保存着键/值对，而且每个 App 都可以拥有自己的共同偏好。

共同偏好中可以保存的值类型被限定为 Java 基本数据类型、字符串、序列化对象和序列化对象数组。与从 SQLite 数据库读写数据相比，共同偏好会慢一些。因此，不应该考虑将共同偏好用作记录管理；应该总是将 SQLite 数据库用作记录管理，如同你在备忘录实验中看到的那样。即便如此，共同偏好仍是持久化保存用户偏好的一种好方法。

我们想要持久化保存显示在本国货币和外国货币选择列表中的货币代码。下面是一种典型场景。假定一位美国用户正在伊斯坦布尔度假，他在集市上使用 **Currencies App** 换算一些珍贵拜占庭古物的价格。用户退出 App 并返回了酒店。第二天早上，他在当地餐厅吃早餐并打开 **Currencies App** 核对账单。如果我们的用户在完成另一次计算之前还需要在选择列表中重新选择 TRY 和 USD，那么体验将会非常差。相反，选择列表中应该自动填充用户之前选择的本国货币和外国货币代码。

我们将要创建一个能够访问共同偏好的实用工具类。此实用工具类将含有公共的静态

方法，允许我们获取和设置用户所选本国和外国货币的货币代码。右击(在 Mac 上按住 Ctrl 键并单击)com.apress.gerber.currencies 包并选择 New Java Class。将类命名为 PrefsMgr 并插入如图 10-10 所示的代码。

```
public class PrefsMgr {

    private static SharedPreferences sSharedPreferences;

    public static void setString(Context context, String locale, String code ){
        sSharedPreferences =
            PreferenceManager.getDefaultSharedPreferences(context);
        SharedPreferences.Editor editor = sSharedPreferences.edit();
        editor.putString(locale, code);
        editor.commit();
    }

    public static String getString(Context context, String locale){
        sSharedPreferences =
            PreferenceManager.getDefaultSharedPreferences(context);
        return sSharedPreferences.getString(locale, null);
    }

}
```

图 10-10 创建 PrefsMgr 类

setString()方法设置特定地区(本国或外国)的货币代码。getString()方法返回对应特定地区的货币代码值。如果代码没有找到，则返回默认的 null。按 Ctrl + K | Cmd + K 并提交，附带消息为 Creates our own preferences manager。

10.9 根据给定代码查找位置

选择列表使用从零开始的整数来表示其当前位置的值。为了将选择列表设置为某个特定代码，我们需要找到元素的对应位置或索引。由于 mCurrencies 是选择列表的模型，因此我们可以简单地将货币代码与 mCurrencies 中保存的聚合字符串的前三个字符进行比较。如果找到了相匹配的记录，则返回索引位置。如果没有找到相匹配的记录，我们就返回零，而这对应于选择列表的第一个位置。ISO 4217 货币代码标准指定货币代码的长度固定为三个字母。让我们编写一个方法，从包含货币代码、管道线和货币描述的聚合字符串中抽取三个字母的货币代码。我们知道聚合字符串的前三个字母就是货币代码，因此可以使用 String 类的 substring()方法来抽取它。打开 MainActivity.java 并在 invertCurrencies()方法的下面定义 findPositionGivenCode()方法，如图 10-11 所示。按 Ctrl + K | Cmd + K 并提交，附带消息为 Creates find position given code method。

```
private int findPositionGivenCode(String code, String[] currencies) {
    for (int i = 0; i < currencies.length; i++) {
        if ((currencies[i]).substring(0, 3).equalsIgnoreCase(code)) {
            return i;
        }
    }
    //default
    return 0;
}
```

图 10-11 创建 findPositionGivenCode()方法

10.10 从货币字符串中抽取代码

从 mCurrencies 的元素中保存的聚合字符串里面抽取三个字母的货币代码并不仅限于在 findPositionGivenCode()方法中使用。为了避免到处重复这段代码，抽取一个方法并在需要时调用它是个好主意。在 MainActivity.java 中，选取图 10-12 中所示的代码，按 Ctrl + Alt + M | Cmd + Alt + M 来抽取方法并选择第一个选项。

```
76 private int findPositionGivenCode(String code, String[] currencies) {
77
78     for (int i = 0; i < currencies.length; i++) {
79         if ((currencies[i]).substring(0, 3).equalsIgnoreCase(code)) {
80             return i;
81         }
82     }
83 }
```

图 10-12 选择将要被抽取为一个方法的代码

如图 10-13 所示，在 Extract Method 对话框中将方法名修改为 extractCodeFromCurrency，然后单击 OK 按钮。你应该会看到类似于图 10-14 中所示的界面。按 Ctrl + K | Cmd + K 并提交，附带消息为 Extracts method called extractCodeFromCurrency。

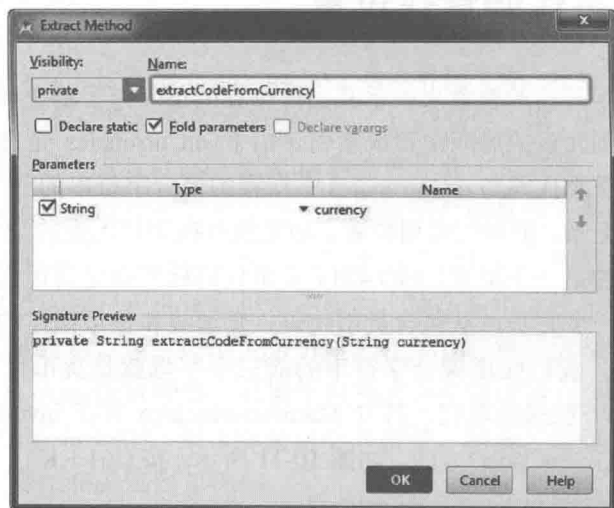


图 10-13 在 Extract Method 对话框中创建 extractCodeFromCurrency()

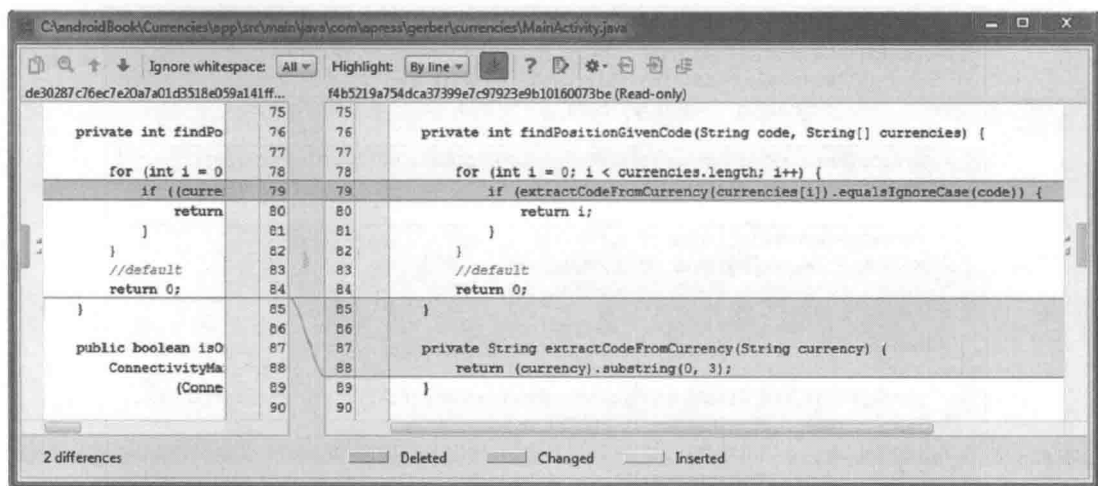


图 10-14 抽取方法操作所得到的代码

10.11 实现共同偏好

共同偏好中的数据保存在哈希表中，其中键总是字符串，因此很适合将键定义为 String 常量。打开 MainActivity.java 并定义图 10-15 中所示的两个 String 常量。

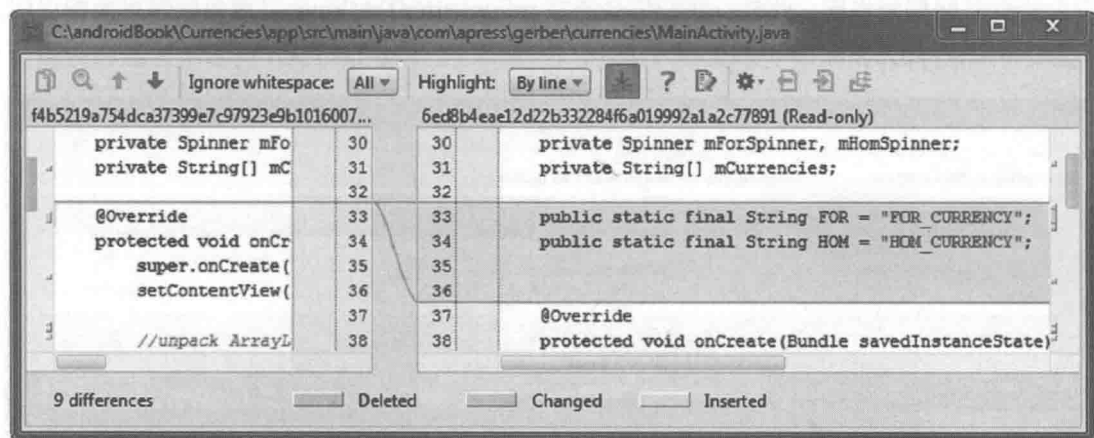


图 10-15 定义两个将要作为键的常量

在 MainActivity 类的 onCreate() 方法的结尾处插入图 10-16 中所示的 if/else 块。在上一步中，我们编写了 PrefsMgr 类，它在没有找到键的情况下返回 null。if 块判断本国货币和外国货币键均不存在的情况。这种情况只会发生一次——当在用户的设备上第一次使用 App 时——此时将外国货币和本国货币选择列表分别设置为 CNY 和 USD。如果不满足这个独特的条件，选择列表将被设置为用户共同偏好中保存的值。

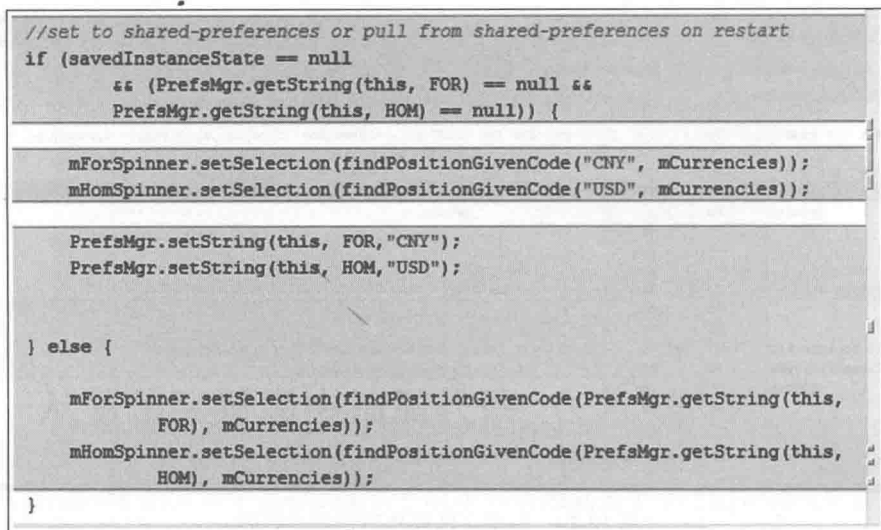


图 10-16 创建 if/else 块

使用共同偏好会带来轻微的性能损耗，我们想要尽量避免。我们在 if 语句的括号中加入了 `savedInstanceState == null &&`，如果 MainActivity 只是从中断或配置修改中恢复，程序则会进入这个块。

导航至我们之前定义的 `onItemSelected()` 方法。修改这个方法，使得每次选择列表项时都会进行共同偏好设置。此外，我们将会清除 `mConvertedTextView` 以避免任何可能的误解。修改 MainActivity.java，如图 10-17 所示。

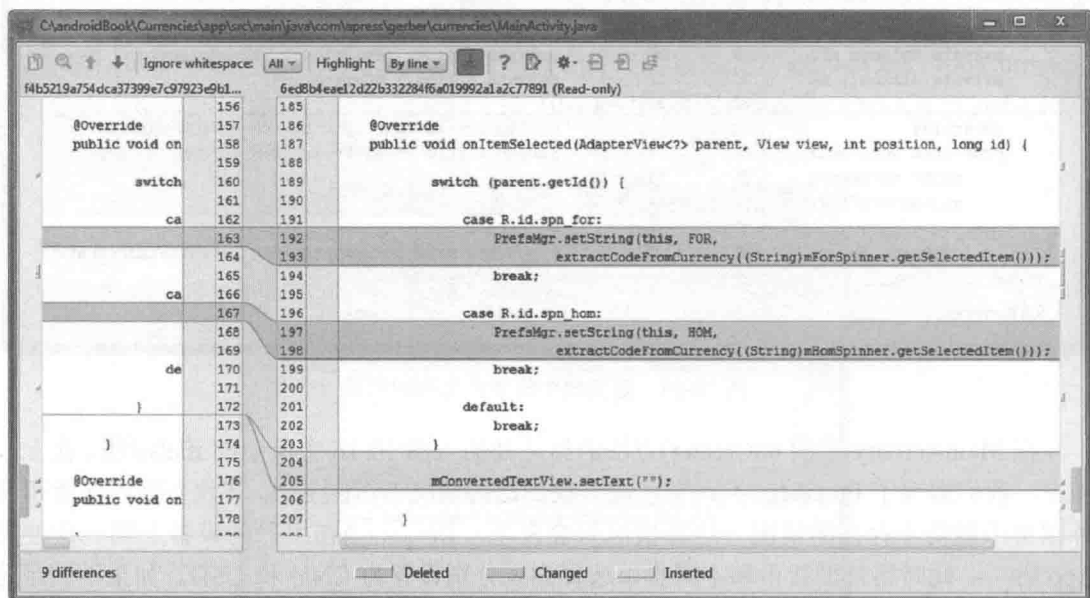


图 10-17 为 onItemSelected 方法应用共同偏好

最后，我们需要确保当用户在选项菜单中选择 Invert Currencies 菜单项时能够正确地设

置共同偏好。将图 10-8 中所示的两行代码添加到 invertCurrencies()方法的末尾。按 Ctrl + K | Cmd + K 并提交，附带消息为 Implements shared preferences。

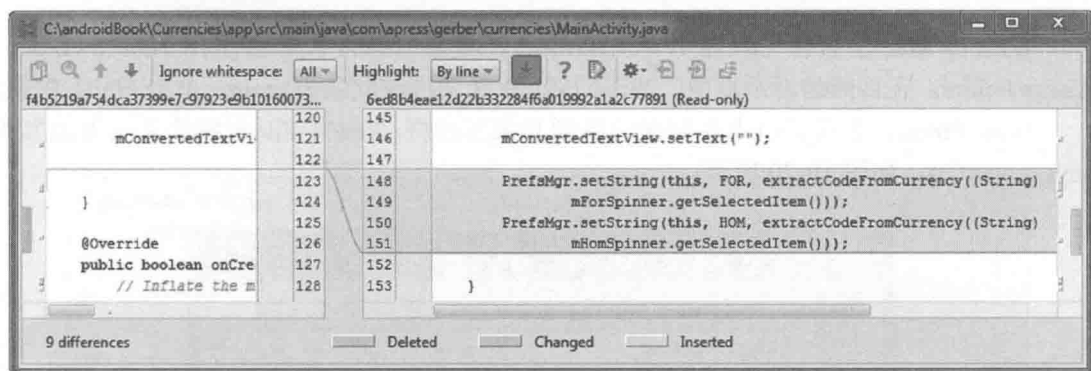


图 10-18 为 invertCurrencies 方法应用共同偏好

10.12 按钮单击行为

我们的 App 中只有一个按钮。因此，可以将按钮行为的处理代理到一个匿名内部类而非包含它的 Activity 中(我们之前对两个选择列表的处理方法)。

在 onCreate()方法的末尾、结束括号以内，输入 mCalcButton.setOnClickListener();。现在，将光标置于这个方法的括号内并输入 new On。使用向下箭头(如果需要的话)在代码补全提供的建议中选择 onClickListener{...}选项，然后按 Enter 键。在 onClick()方法中添加以下占位符，例如//define behavior here，如图 10-19 中所示。按 Ctrl + K | Cmd + K 并提交，附带消息为 Creates anon inner class to handle button behavior。

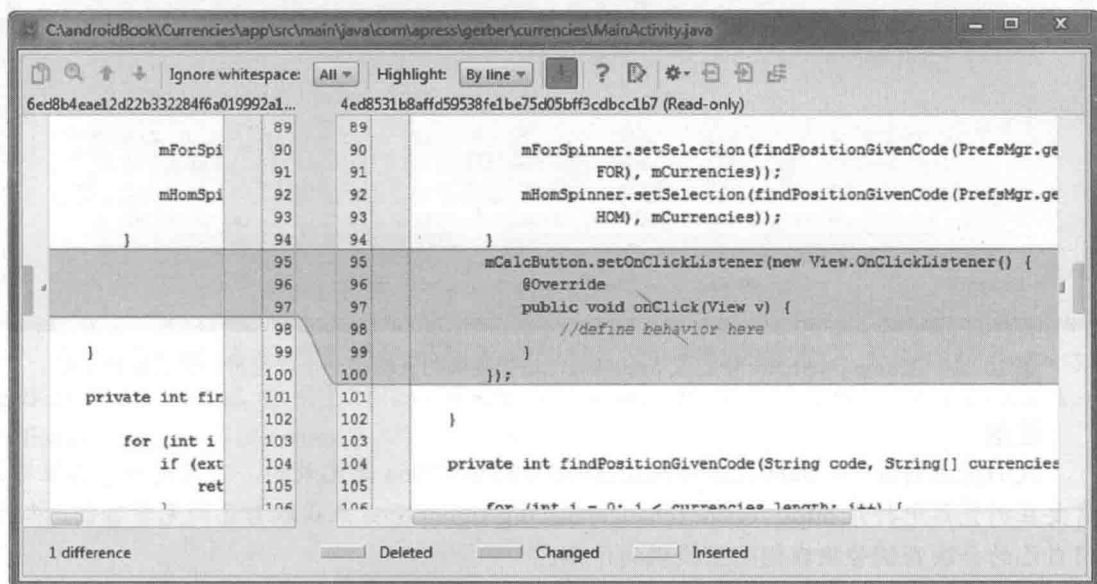


图 10-19 创建匿名内部类来处理按钮单击行为

10.13 保存开发者密钥

右击(在 Mac 上按住 Ctrl 键并单击)Project 工具窗口中的 App 并选择 New | Folder | Assets Folder。在后续的对话框中, Target Source 选项应该默认为 main。单击 Finish 按钮。

右击 Project 工具窗口中新创建的资源目录并选择 New | File, 将此新文件命名为 keys.properties, 如图 10-20 所示。



图 10-20 创建 keys.properties 文件

向 keys.properties 文件添加以下行:

```
open_key=9a894f5f4f5742e2897d20bdcac7706a
```

需要在浏览器中打开 <https://openexchangerates.org/signup/free> 来获取自己的免费密钥。这个过程很简单, 需要大概花费 30 秒。用自己的合法密钥替换我们这里提供的示例, 参见图 10-21。按 Ctrl + K | Cmd + K 并提交, 附带消息为 Defines openexchangerates.org key。

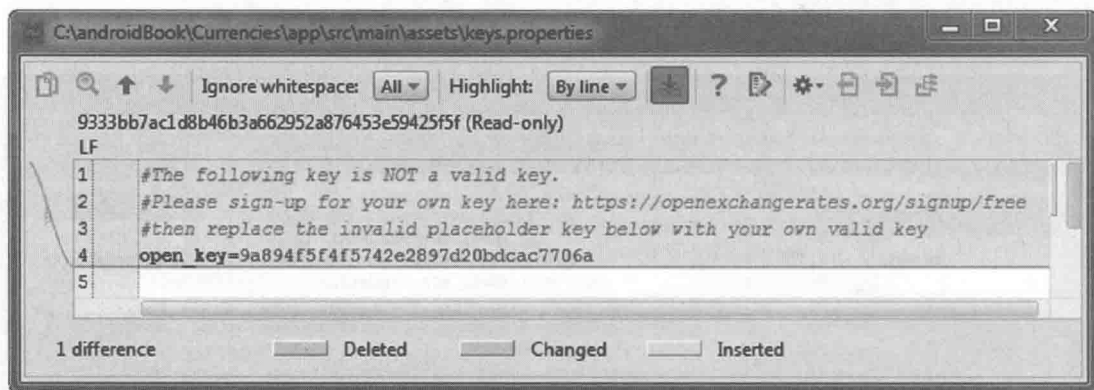


图 10-21 在 keys.properties 中定义 open_key, 这里提供的密钥是一个占位符, 无法正常工作

注意

我们提供的键——9a894f5f4f5742e2897d20bdcac7706a 是无效的, 它只是一个占位符。需要在浏览器中打开 <https://openexchangerates.org/signup/free> 来获取自己的免费密钥, 然后用自己的合法密钥替换我们这里提供的示例。

10.14 获取开发者密钥

在 MainActivity.java 中 extractCodeFromCurrency() 方法的下面定义名为 getKey() 的方法, 获取 keys.properties 中保存的密钥。注意, 我们正在使用 AssetManager 从 keys.properties 中读取密钥。需要导入必需的类, 参见图 10-22。

```
private String getKey(String keyName){
    AssetManager assetManager = this.getResources().getAssets();
    Properties properties = new Properties();
    try {
        InputStream inputStream = assetManager.open("keys.properties");
        properties.load(inputStream);

    } catch (IOException e) {
        e.printStackTrace();
    }
    return properties.getProperty(keyName);
}
```

图 10-22 定义 getKey() 方法

文件 I/O 是一个耗时的操作。我们在上一步中定义的 getKey() 方法包含这样的操作, 因此需要尽可能少地调用 getKey() 方法。我们将在 onCreate() 中调用一次 getKey() 方法并在 MainActivity 的成员 mKey 中保存这个值, 而非每次想要从 openexchangerates.org 获取汇率时都调用 getKey() 方法。定义 MainActivity 类的成员, 如图 10-23 所示。

```
//this will contain my developers key
private String mKey;
//used to fetch the 'rates' json object from openexchangerates.org
public static final String RATES = "rates";
public static final String URL_BASE =
    "http://openexchangerates.org/api/latest.json?app_id=";
//used to format data from openexchangerates.org
private static final DecimalFormat DECIMAL_FORMAT = new
    DecimalFormat("#,##0.00000");
```

图 10-23 定义成员以简化密钥的获取和结果的格式化

在 onCreate() 方法的末尾、结束括号以内, 为 mKey 赋值, 类似这样: mKey=getKey("open_key");, 参见图 10-24。按 Ctrl + K | Cmd + K 并提交, 附带消息为 Fetches key, defines members and constants。

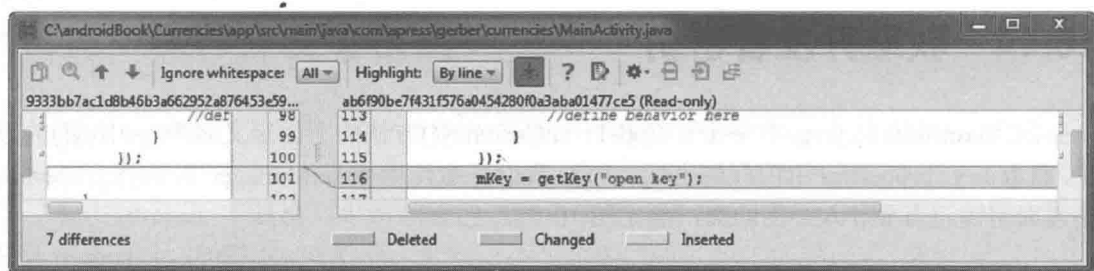


图 10-24 在 onCreate() 方法的最后一句中为密钥赋值

10.15 CurrencyConverterTask

线程是轻量级的进程，它在相同的应用中与其他线程并行执行。Android 并发的第一条规则是不要阻塞 UI 线程，也就是主线程。UI 线程在 App 启动时默认开启，用于操作用户界面。如果 UI 线程被阻塞超过 5000 毫秒，Android OS 将会显示“应用无响应” (Application Not Responding, ANR) 错误，你的应用也将会崩溃。阻塞 UI 线程不仅会导致 ANR 错误，还会让用户界面完全失去响应。因此，如果某项操作可能需要花费超过若干毫秒的时间，那么它就可能阻塞 UI 线程，应该在后台线程中完成它。例如，尝试从远处服务器获取数据可能会持续超过若干毫秒，应该在后台线程中完成此操作。当在 Android 上下文中使用时，后台线程这个术语表示除 UI 线程之外的所有线程。

注意

有时将 UI 线程称为主线程。

Android 并发的第二条规则是 UI 线程是唯一一个拥有和用户界面交互权限的线程。如果尝试从后台线程中更新任意视图，那么你的 App 将会立刻崩溃！违法任意一条 Android 并发规则都会导致极差的用户体验。

可以在 Android App 中开启老式的 Java 线程，但是有一个名为 AsyncTask 的类专门用于解决本节中描述的问题，因此它是 Android 并发的推荐实现方式。如果正确实现了 AsyncTask，你便已经遵守了 Android 并发的这两条规则。

在本节中，我们将要创建名为 CurrencyConverterTask 的内部类，用于从 openexchangerates.org 获取货币汇率。CurrencyConverterTask 是 AsyncTask 抽象类的一个具体实现。AsyncTask 含有一个名为 doInBackground() 的抽象方法，所有具体类都需要重载它。此外，可以重载其他一些方法，包括 onPreExecute()、onProgressUpdate() 和 onPostExecute()。AsyncTask 的魔力在于 doInBackground() 方法在后台线程中执行，而 AsyncTask 的其他方法均在 UI 线程中执行。只要在 doInBackground() 方法中不操作任何视图，就可以非常安全地使用 AsyncTask。

在 MainActivity.java 的末尾、MainActivity 关闭括号的内部，将 CurrencyConverterTask

定义为私有内部类。除继承 `AsyncTask` 之外，还必须定义三个泛型对象参数，如图 10-25 所示。导入所有必需的类。即使在处理这些导入之后，类定义仍然会有红色的下划线，表示存在编译时错误，现在先忽略此错误。

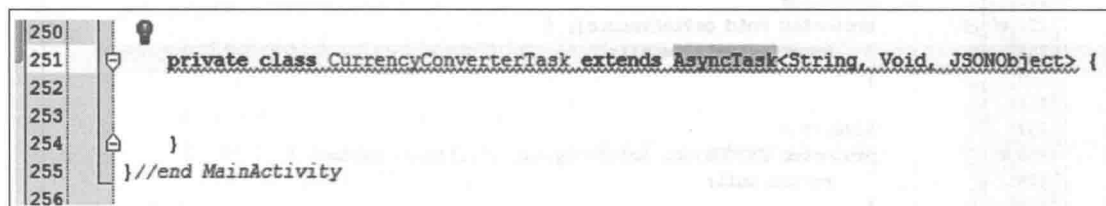


图 10-25 定义 `CurrencyConverterTask`

将光标置于 `CurrencyConverterTask` 类定义花括号的内部，按 `Alt + Insert | Cmd + N` 并选择 `Override Methods`。选择 `doInBackground()`、`onPreExecute()` 和 `onPostExecute()` 方法并单击 `OK` 按钮，如图 10-26 所示。注意，返回值以及 `doInBackground()` 和 `onPostExecute()` 的参数均是根据泛型参数 `<String, Void, JSONObject>` 定义的。第一个参数 `String` 用于 `doInBackground()` 方法的输入，第二个参数 `Void` 用于向 `onProgressUpdate()` 方法发送进度更新，而第三个参数 `JSONObject` 是 `doInBackground()` 的返回值以及 `onPostExecute()` 方法的输入参数。整个获取操作应该会花费大约一秒钟，因此用户几乎感觉不到进度更新；而这也是我们省略了 `onProgressUpdate()` 方法并使用 `Void` 作为第二个参数的原因。

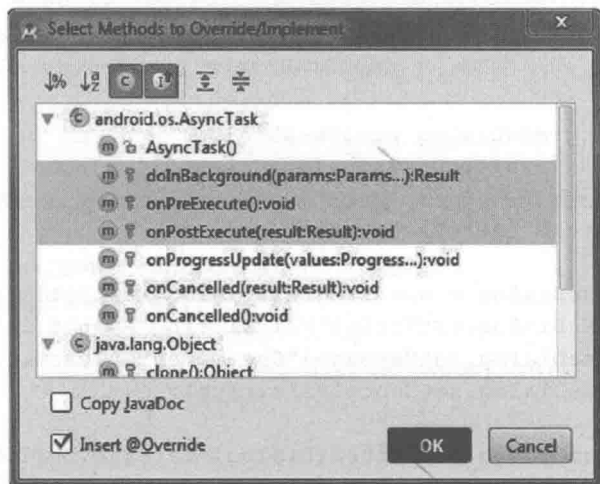


图 10-26 选择需要重载/实现的方法

让我们重新安排这些方法，使得它们按照被调用的顺序排列。选择整个 `onPreExecute()` 块，包括 `@Override` 注解，按 `Ctrl + Shift + Up | Cmd + Shift + Up` 将 `onPreExecute()` 方法移到 `doInBackground()` 方法之上。`CurrencyConverterTask` 现在应该类似于图 10-27 所示。

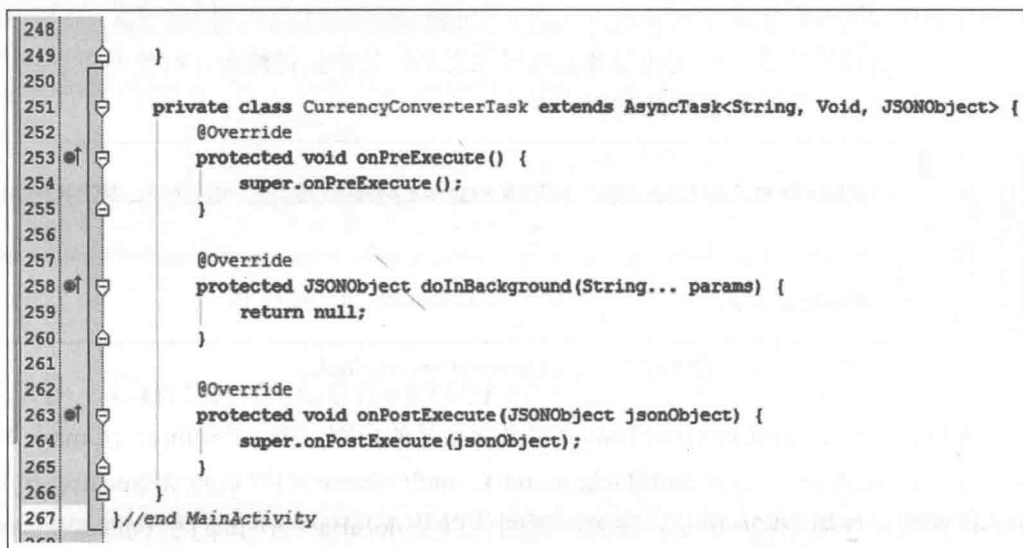


图 10-27 在 CurrencyConverterTask 中重载方法并上移 onPreExecute() 之后的界面

再次修改 CurrencyConverterTask，使其看上去如代码清单 10-6 所示并导入所有必需的类。让我们依次讨论 CurrencyConverterTask 中的三个重载方法。

代码清单 10-6 修改 CurrencyConverterTask

```

private class CurrencyConverterTask extends AsyncTask<String, Void,
JSONObject> {

    private ProgressDialog progressDialog;

    @Override
    protected void onPreExecute() {

        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setTitle("Calculating Result...");
        progressDialog.setMessage("One moment please...");
        progressDialog.setCancelable(true);

        progressDialog.setButton(DialogInterface.BUTTON_NEGATIVE,
            "Cancel", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    CurrencyConverterTask.this.cancel(true);
                    progressDialog.dismiss();
                }
            });
        progressDialog.show();
    }

    @Override
    protected JSONObject doInBackground(String... params) {

```

```

        return new JSONParser().getJSONFromUrl(params[0]);
    }
    @Override
    protected void onPostExecute(JSONObject jsonObject) {

        double dCalculated = 0.0;
        String strForCode = extractCodeFromCurrency(mCurrencies[mForSpinner.
getSelectedItemPosition()]);
        String strHomCode = extractCodeFromCurrency(mCurrencies[mHomSpinner.
getSelectedItemPosition()]);
        String strAmount = mAmountEditText.getText().toString();

        try {

            if(jsonObject == null){
                throw new JSONException("no data available.");
            }
            JSONObject jsonRates = jsonObject.getJSONObject(RATES);
            if(strHomCode.equalsIgnoreCase("USD")){
                dCalculated = Double.parseDouble(strAmount) /
jsonRates.getDouble(strForCode);
            } else if (strForCode.equalsIgnoreCase("USD")) {
                dCalculated = Double.parseDouble(strAmount) *
jsonRates.getDouble(strHomCode) ;
            }
            else {
                dCalculated = Double.parseDouble(strAmount) *
jsonRates.getDouble(strHomCode) / jsonRates.getDouble(strForCode);
            }
        } catch (JSONException e) {
            Toast.makeText(
                MainActivity.this,
                "There's been a JSON exception: " + e.getMessage(),
                Toast.LENGTH_LONG

            ).show();
            mConvertedTextView.setText("");
            e.printStackTrace();
        }
        mConvertedTextView.setText(DECIMAL_FORMAT.format(dCalculated) + " "
+ strHomCode);
        progressDialog.dismiss();
    }
}

```

10.15.1 onPreExecute()

`onPreExecute()`方法在 UI 线程中执行，它发生在启动 `doInBackground()`方法之前。由于我们无法在后台线程中操作任何 UI 视图，因此 `onPreExecute()`方法提供了在 `doInBackground()`启动之前修改 UI 的机会。当 `onPreExecute()`被调用时会显示 `ProgressDialog`，它带有 `Cancel` 按钮，用户可以单击并终止操作。

10.15.2 doInBackground()

`doInBackground()`方法是 `AsyncTask` 中 `execute()`方法的代理。例如，调用 `CurrencyConverterTask`的最简单方法是实例化一个新的引用(匿名对象)并调用它的 `execute()`方法，如下所示：

```
new CurrencyConverterTask().execute("url_to_web_service");
```

传入 `execute()`的参数将会被传入 `doInBackground()`，不过是在执行 `onPreExecute()`之后。`doInBackground()`的完整签名是 `protected JSONObject doInBackground(String... params)`。`doInBackground()`的参数被定义为可变参数，因此我们可以向 `execute()`传入任意数量、类型为 `String` 的逗号分隔参数，尽管我们在这个简单 App 中仅传入了一个表示 URL 的字符串。一旦进入 `doInBackground()`方法，`params`就会被视作一个字符串数组。我们使用 `params[0]`来引用第一个(也是唯一一个)元素。

在 `doInBackground()`的方法体中，调用 `return new JSONParser().getJSONFromUrl(params[0])`。`getJSONFromUrl()`方法从 Web Service 获取 `JSONObject`。由于这个操作需要在用户设备和远程服务器之间通信，因此可能花费超过若干毫秒，我们将 `getJSONFromUrl()`放在了 `doInBackground()`方法的内部。`getJSONFromUrl()`方法返回 `JSONObject`，而这是 `doInBackground()`定义的返回值。如前所述，`doInBackground()`是 `AsyncTask` 中唯一一个在后台线程中运行的方法，所有其他方法均在 UI 线程中运行。注意，我们在 `doInBackground()`方法中不会操作任何视图。

10.15.3 onPostExecute()

类似于 `onPreExecute()`，`onPostExecute()`方法在 UI 线程中运行。`doInBackground()`的返回值被定义为 `JSONObject`。相同的对象将被作为参数传入 `onPostExecute()`方法，它的完整签名被定义为 `protected void onPostExecute(JSONObject jsonObject)`。当进入 `onPostExecute()`方法时，`doInBackground()`方法的后台线程已经结束，现在可以使用从 `doInBackground()`获取到的 `JSONObject` 数据安全地更新 UI 了。最后，做些计算并将格式化好的结果赋值给 `mConvertedTextView`。

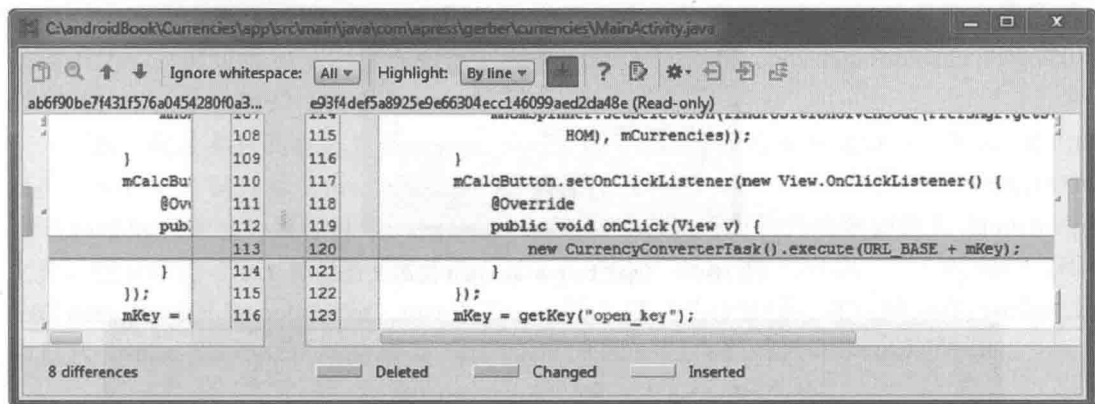


图 10-28 在 mCalcButton 的 onClick 方法中调用新创建的 CurrencyConverterTask

在可以运行 App 之前，需要对代码做最后一处修改，以便运行 CurrencyConverterTask。修改图 10-28 中所示 mCalcButton 的 onClick() 方法。

按 Ctrl + K | Cmd + K 并提交，附带消息为 Implements CurrencyConverterTask。按 Shift + F10 | Ctrl + R 来运行你的 App。在 Enter Foreign Currency Amount Here 框中输入一个数额并单击 Calculate 按钮。你应该能够从服务器取回结果，这个结果将会显示在 Calculated Result in Home Currency 框中。如果 App 没能返回结果，那么验证从 openexchangerates.org 获取的开发者密钥是否合法。

10.16 按钮选择器

当运行 Currencies App 时，你可能会注意到 mConvertedTextView 中显示的文本是黑色的，没有足够的对比度。打开 activity_main.xml 文件并修改 txt_convertedTextView 的定义，插入图 10-29 中突出显示的代码行。

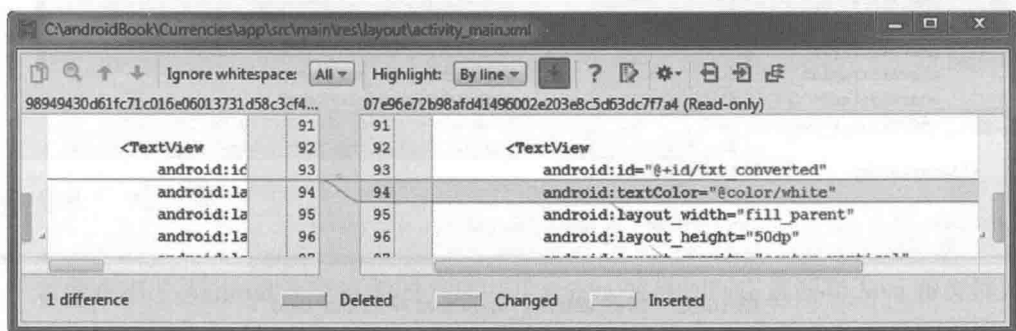


图 10-29 在 activity_main.xml 中插入 txt_converted 的 textColor 特性并将其设置为 @color/white

右击(在 Mac 上按住 Ctrl 键并单击)drawable 目录并选择 New | Drawable Resource File。将资源命名为 button_selector，如图 10-30 所示。将 XML 修改为类似于图 10-31 所示的样子。在 activity_main.xml 中修改 btn_calc 的定义，如图 10-32 所示。



图 10-30 创建 button_selector 资源文件

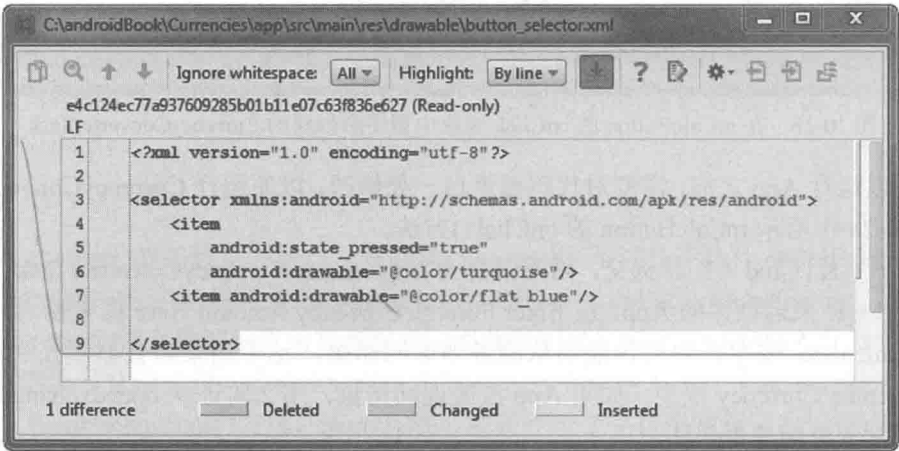


图 10-31 修改 button_selector 资源文件

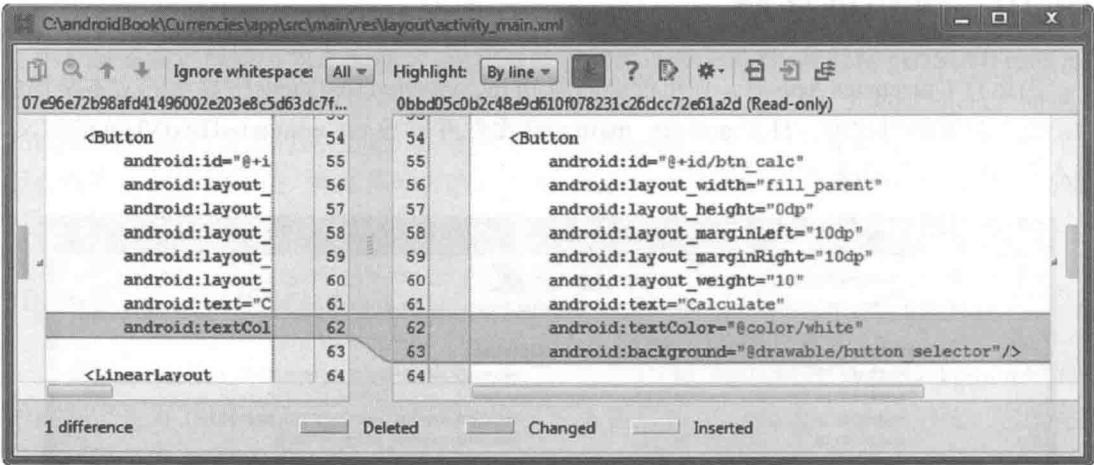


图 10-32 修改 activity_main.xml 中的 btn_calc

按 Ctrl + K | Cmd + K 并提交，附带消息为 Creates button selector。

10.17 启动图标

为了不再使用通用的 Android 图标，我们将要定义自己的启动图标。笔者使用 Google

高级图片搜索功能找到了一副免费图片，它是一个一欧元的硬币，这是现行流通的最优秀硬币之一。可以在如下网址找到这张图片：http://pixabay.com/static/uploads/photo/2013/07/13/01/21/coin-155597_640.png。

下载此图片并将其命名为 `coin.png`。将 Project 工具窗口切换为 Android 视图。右击(在 Mac 上按住 `Ctrl` 键并单击) `res/mipmap` 目录并选择 `New | Image Asset`。在后续的对话框中，将 `drawable` 目录选为目标目录。使用图 10-33 中的设置为每种分辨率创建 `ic_launcher.png` 文件，然后单击 `Next` 按钮。在 `MainActivity` 的 `onCreate()` 方法内部、填充布局的代码行 `setContentView(R.layout.activity_main);` 之后，插入以下几行代码，此代码会在 `Action Bar` 中显示一个自定义图标：

```
ActionBar actionBar = getSupportActionBar();
actionBar.setHomeButtonEnabled(true);
actionBar.setDisplayHomeAsUpEnabled(true);
actionBar.setIcon(R.mipmap.ic_launcher);
```

此 App 的图标现在将会是一个一欧元的硬币，而不再是标准的 Android 图标。按 `Ctrl + K` | `Cmd + K` 并提交，附带消息为 `Creates launcher icon`。

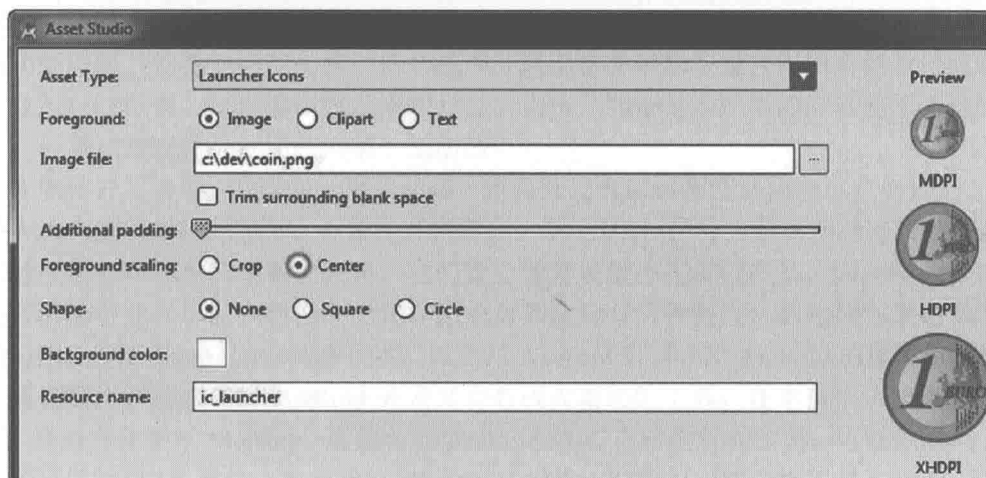


图 10-33 创建 `ic_launcher` 图标

10.18 小结

本章介绍了 Android 填充视图的方法以及 `R.java` 文件如何在资源和 Java 源文件之间起到桥梁作用。你学会了：从 `bundle` 中解压值以及如何实现菜单并对其行为进行编码；使用 `ArrayAdapter` 把字符串数组绑定到选择列表；使用 Android Studio 将视图事件的处理代理到包含它们的 `Activity`；使用共同偏好和资源的方法；以及 Android 中有关并发的知识——尤其是使用 `AsyncTask` 的方法。你还创建了自己的 `CurrencyConverterTask`，它从 `openexchangerates.org` 提供的 Web 服务获取货币汇率。最后，你使用 Android Studio 生成图片资源并创建了按钮选择器。

我们已经完成了在上一章中开始的 Currencies App。按 Shift + F10 | Ctrl + R 运行 App, 并确保它能够完成预期的功能。如果你是一位有经验的 Android 开发者或是一位有好奇心的 UI 测试人员, 你可能已经发现有一种临界情况会导致 App 崩溃。我们将保留这个 bug 并在专门讲解分析和测试的第 11 章中修复。

第 11 章

测试和分析

测试在任何软件开发生命周期中都是一个关键的阶段。在某些公司里，质量保证团队负责编写和维护测试用例，而在其他一些公司里，开发团队不得不完成此项任务。在这两种情形中，随着应用变得越来越复杂，测试也变得愈发重要。测试能够让团队成员发现应用的功能问题，使得他们有足够的信心继续开展工作并确保在源代码中所做的任何修改不会导致运行时错误、错误输出以及未知行为。当然，即使是最严格的测试也无法消除所有错误，但测试是软件开发团队的第一道防线。

在软件开发中，测试是一道争论题。所有开发者应该都会认同测试是必需的。然而，有些人认为测试是如此重要，应该在开发阶段之前启动(一种称为测试驱动开发的方法论)；而其他一些公司，尤其是初创公司，它们致力于开发最小化可行产品，因此将测试视作一种潜在的浪费并尽可能少地投入力量。无论对测试持有何种观点，都建议你掌握本章中介绍的技术，包括 `android.test` 库中的类，以及与 Android Studio 和 Android SDK 捆绑的工具。

我们选择讲解那些对 Android 开发者有着极大帮助的工具。在本章中，我们引入仪器测试；接着向你展示 Monkey，这是一款 Android SDK 自带的优秀工具，可以生成用于 App 压力测试的随机 UI 事件；最后我们展示 Android Studio 中的一些分析工具。

提示

Roboelectric 是一款优秀的第三方测试框架。虽然 Roboelectric 与我们这里讨论的 Android SDK 测试框架相比并没有明显的优势，但它在 Android 开发者中依然很流行。关于 Roboelectric 的更多信息可以在这里找到：roboelectric.org。

11.1 创建新的仪器测试

仪器测试允许你在设备上执行操作，就像人工操作它一样。在本节中，你将要通过继承 `android.test.ActivityInstrumentationTestCase2` 类来创建仪器测试。

打开第 10 章中创建的 Currencies 项目并将 Project 工具窗口切换至 Android 视图。

在 Project 工具窗口中，右击(在 Mac 上按住 Ctrl 键并单击)com.apress.gerber.currencies (androidTest)包并选择 New | Java Class。将你的类命名为 MainActivityTest，它派生自 ActivityInstrumentationTestCase2<MainActivity>。定义构造函数，如图 11-1 所示。你将会注意到 ActivityInstrumentationTestCase2<>的泛型参数是 MainActivity，这就是待测试的 Activity。

```
18
19 public class MainActivityTest extends ActivityInstrumentationTestCase2<MainActivity> {
20
21     public MainActivityTest() {
22         super(MainActivity.class);
23     }
24 }
```

图 11-1 定义一个名为 MainActivityTest 的类，它派生自 ActivityInstrumentationTestCase2

11.1.1 定义 SetUp()和 TearDown()方法

将光标置于 MainActivityTest 类的作用域内并再次按 Alt + Insert | Cmd + N 激活 Generate 上下文菜单，如图 11-2 所示。选择 SetUp()方法并按 Enter 键，为 TearDown()方法重复该过程。框架代码看上去应该类似于图 11-3。setUp()和 tearDown()方法是这个仪器测试的生命周期方法。setUp()方法让你能够连接所需的任意资源，通过 bundle 传入任意数据或者在运行测试之前赋值引用。tearDown()方法可以用于关闭所有连接以及在测试方法运行完之后清理所有资源。

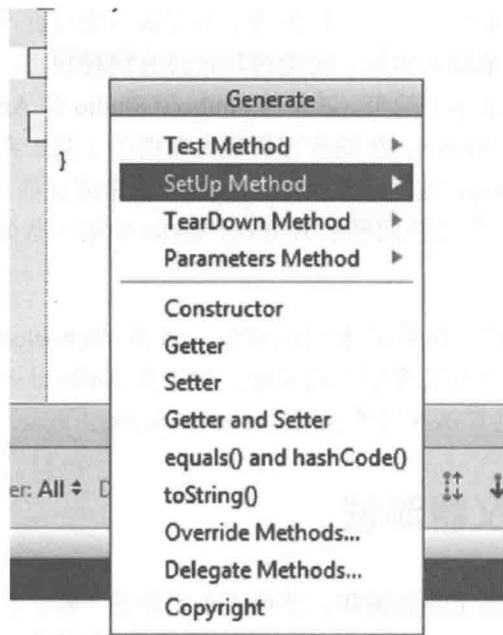


图 11-2 生成 SetUp()和 TearDown()方法

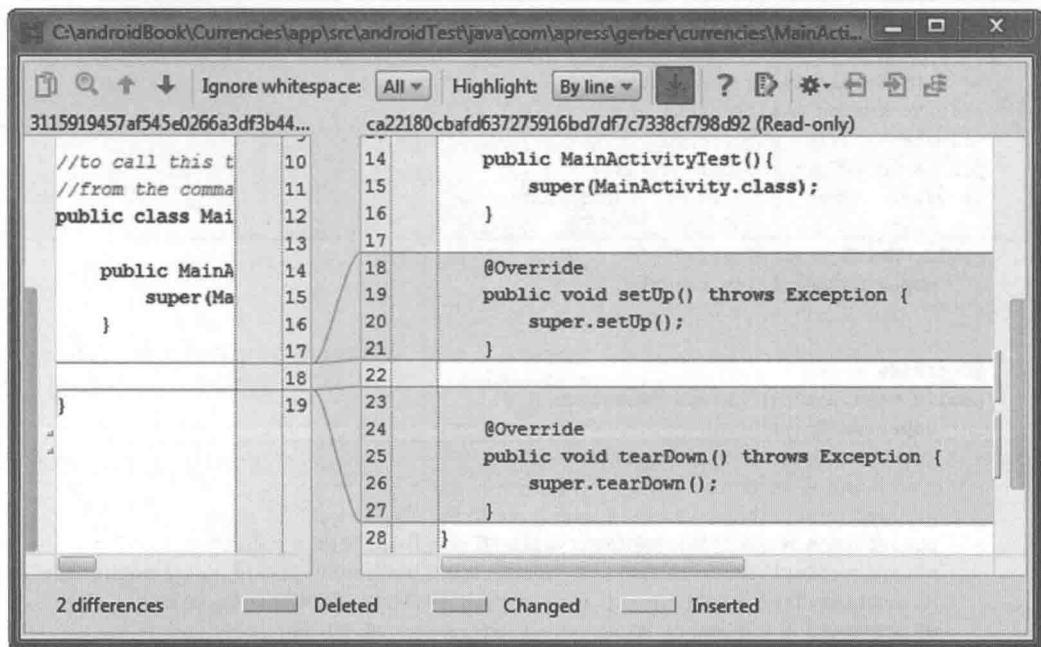


图 11-3 SetUp()和 TearDown()方法的框架代码

打开 MainActivity.java 文件(这是我们即将测试的 Activity), 并查看 onCreate()方法。每个 Activity(MainActivity 也不例外)的 onCreate()生命周期方法都是获取已填充视图引用的地方。例如, 在 MainActivity 中, `mCalcButton = (Button) findViewById(R.id.btn_calc);`一行找到已经在堆上初始化且 ID 为 `R.id.bnt_calc` 的视图, 还将其类型转换为 `Button` 并把该引用赋值给 `mCalcButton`。

在 MainActivityTest 中, 将以几乎完全相同的方式获取 MainActivity 中视图的引用。不过, 由于 `findViewById()`是 Activity(而非 `ActivityInstrumentationTestCase2`)的方法, 因此我们需要一个对 MainActivity 的引用, 以便能够完成这项任务。在 MainActivityTest 中定义引用 `MainActivitymActivity`;以及其他一些引用, 如图 11-4 所示。`ActivityInstrumentationTestCase2<MainActivity>`类有一个名为 `getActivity()`的方法, 它返回一个对 MainActivity 的引用。当 MainActivity 引用被传入 MainActivityTest 的构造函数时, MainActivity 中的视图已经填充完毕。一旦有了这个引用, 就可以调用 `mActivity.findViewById()`来获取视图的引用了, 如图 11-4 所示。

按 `Ctrl + K` | `Cmd + K` 并提交, 附带消息为 `Gets references to inflated views in MainActivity`。要知道在正常情况下, MainActivity 是由 SplashActivity 启动的, 而 SplashActivity 获取活动货币代码并把代码保存在 `ArrayList<String>`里面, 接着把 `ArrayList<String>`压缩到 bundle 中, 然后通过 Intent 将该 bundle 传递给 MainActivity。不用借助于 SplashActivity 就可以模拟所有这些情况。重新创建代码, 如图 11-5 所示。在 `setActivityResult(intent)`一行中, 为 MainActivity 提供测试数据——与正常情况下由 SplashActivity 调用 MainActivity 时传入的数据类型相同。

```

public class MainActivityTest extends ActivityInstrumentationTestCase2<MainActivity>

    private MainActivity mActivity;
    private Button mCalcButton;
    private TextView mConvertedTextView;
    private EditText mAmountEditText;
    private Spinner mForSpinner, mHomSpinner;

    public MainActivityTest(){
        super(MainActivity.class);
    }

    @Override
    public void setUp() throws Exception {
        super.setUp();

        //get the activity under test
        mActivity = getActivity();
        //assign references to our views
        mCalcButton = (Button) mActivity.findViewById(R.id.btn_calc);
        mConvertedTextView = (TextView) mActivity.findViewById(R.id.txt_converted);
        mAmountEditText = (EditText) mActivity.findViewById(R.id.edt_amount);
        mForSpinner = (Spinner) mActivity.findViewById(R.id.spn_for);
        mHomSpinner = (Spinner) mActivity.findViewById(R.id.spn_hom);
    }

```

图 11-4 定义成员和 setUp()方法的内容

```

public void setUp() throws Exception {
    super.setUp();

    //pass bogus currencies
    ArrayList<String> bogusCurrencies = new ArrayList<String>();
    bogusCurrencies.add("USD|United States Dollar");
    bogusCurrencies.add("EUR|Euro");
    Intent intent = new Intent();
    intent.putExtra(SplashActivity.KEY_ARRAYLIST, bogusCurrencies);
    setActivityIntent(intent);

    //get the activity under test
    mActivity = getActivity();
}

```

图 11-5 通过将已加载数据的 Intent 传递给 MainActivity 来模拟 SplashActivity 的工作

11.1.2 在 MainActivity 中定义回调

大多数情况下，仪器测试将在 UI 线程上进行，因此测试时不必修改 Activity。不过在本例中，我们想要测试 CurrencyConverterTask 完成其后台线程工作之后的应用状态。要实现此功能，需要在 MainActivity 中定义一个回调函数。

打开 MainActivity.java 并定义实例、接口和 setter 方法，如图 11-5 所示。同时，在 CurrencyConverterTask 的 onPostExecute()方法的末尾处，添加图 11-7 中的代码。按 Ctrl + K | Cmd + K 并提交，附带消息为 Define callback in MainActivity。

```
//used to format data from openexchangerates.org
private static final DecimalFormat DECIMAL_FORMAT = new
    DecimalFormat("#,##0.00000");

//create this interface for instrumentation testing with threads
private CurrencyTaskCallback mCurrencyTaskCallback;

public static interface CurrencyTaskCallback {
    void executionDone();
}

public void setCurrencyTaskCallback(CurrencyTaskCallback currencyTaskCallback) {
    this.mCurrencyTaskCallback = currencyTaskCallback;
}
}
```

图 11-6 在 MainActivity.java 类中定义一个接口

```
progressDialog.dismiss();

//for testing
if (mCurrencyTaskCallback != null) {
    mCurrencyTaskCallback.executionDone();
}

}
```

图 11-7 在 CurrencyConverterTask 的结尾处添加 if 代码块

11.1.3 定义一些测试方法

返回到 MainActivityTest.java。将光标置于类的作用域内，重新创建名为 proxyCurrencyConverterTask()和 convertToDouble()的方法，如代码清单 11-1 中所示。需要处理一些导入。proxyCurrencyConverterTask()向选择列表填充数据，模拟单击 Calculate 按钮，等待服务器的响应，然后测试服务器返回的数据是否准确。

代码清单 11-1 创建方法来模拟 CurrencyConverterTask 并等待其完成

```
public void proxyCurrencyConverterTask(final String str) throws Throwable {

    final CountDownLatch latch = new CountDownLatch(1);

    mActivity.setCurrencyTaskCallback(new
    MainActivity.CurrencyTaskCallback() {

        @Override
        public void executionDone() {
            latch.countDown();
        }
    });
}
```

```

        assertEquals(convertToDouble(mConvertedTextView.getText().
            toString().substring(0, 5)), convertToDouble(str));
    }
});

runOnUiThread(new Runnable() {

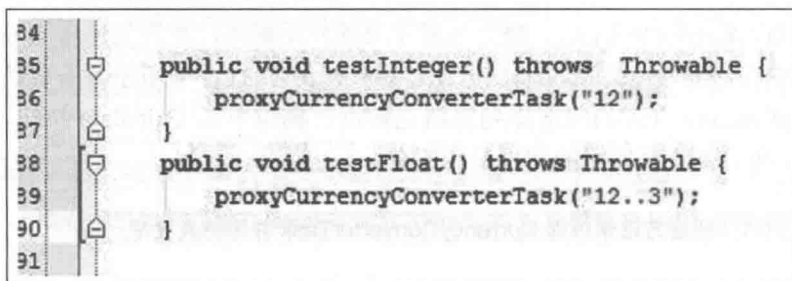
    @Override
    public void run() {
        mAmountEditText.setText(str);
        mForSpinner.setSelection(0);
        mHomSpinner.setSelection(0);
        mCalcButton.performClick();
    }
});

latch.await(30, TimeUnit.SECONDS);
}

private double convertToDouble(String str) throws NumberFormatException{
    double dReturn = 0;
    try {
        dReturn = Double.parseDouble(str);
    } catch (NumberFormatException e) {
        throw e;
    }
    return dReturn;
}
}

```

再次将光标置于类作用域内 `proxyCurrencyConverterTask()` 方法的下面，并按 `Alt + Insert` | `Cmd + N` 激活 `Generate` 上下文菜单。选择 `Test Method` 并按 `Enter` 键。将方法命名为 `testInteger()` 并重新创建如图 11-8 所示的方法，包括使用 `Throwable` 替换 `Exception`。对名为 `testFloat()` 的测试方法重复此步骤。



```

34
35 public void testInteger() throws Throwable {
36     proxyCurrencyConverterTask("12");
37 }
38 public void testFloat() throws Throwable {
39     proxyCurrencyConverterTask("12..3");
40 }
41

```

图 11-8 创建测试方法，向 `proxyCurrencyConverterTask()` 传递非数字值，例如 “12..3” 或 “12,,3”

在这两个测试方法中，把大部分行为代理给 `proxyCurrencyConverterTask()` 方法。要记住，为了让 `ActivityInstrumentationTestCase2` 能够识别测试方法，方法必须以小写的 `test` 开头。

在 `testInteger()` 中, 我们使用整数 12 的字符串表示来填充 `mAmountEditText`, 使用对应 EUR|Euro 的货币数组索引值设置 `mForSpinner` 和 `mHomSpinner`。接下来, 我们通过调用 `performClick()` 方法来模拟单击 `mCalculateButton`。我们将要使用一种叫做 `CountDownLatch` 的机制, 当从服务器获取货币汇率时, 它可以暂停当前线程。 `MainActivity` 中 `CurrencyConverterTask` 的线程一旦结束, `CurrencyConverterTask` 就会调用 `executionDone()`, 而它会释放 `CountDownLatch`, 允许 `ActivityInstrumentationTestCase2` 继续执行并调用 `assertEquals()`。本国货币和外国货币均被设置为 EUR, 因此输出应该等于输入。我们这里创建的仪器测试使用了 JUnit 框架; 因此, 只要 `assertEquals()` 方法返回 `true`, 我们的测试就会通过。

在 `testFloat()` 方法中, 我们模拟与之前所述相同的过程, 只是使用非数值数据(12..3)填充 `mAmountEditText`。虽然我们通过设置 `mAmountEditText` 的软键盘限制用户只能输入数字, 但用户仍有可能在一行中输入两个小数点, 而这也是我们此处要测试的场景。按 `Ctrl + K` | `Cmd + K` 并提交, 附带消息为 `Create proxy methods`。

注意

在某些语言中, 逗号用于代替表示小数点的句号。如果设备的默认语言被设置为这样一种语言, 软键盘将会显示逗号而非句号。需要测试(12,,3)而非(12..3)。

11.1.4 运行仪器测试

右击(在 Mac 上按住 `Ctrl` 键并单击) `Project` 工具窗口中的 `MainActivityTest` 并选择上下文菜单中的 `Run`。也可以从位于工具栏 `Run` 按钮左侧的组合框中选择 `MainActivityTest`, 然后单击 `Run` 按钮。 `Android Studio` 将会显示 `Run` 工具窗口, 而控制台将会显示进度。 `testFloat()` 方法应该会失败, 你将看到一个红色的进度条, 如图 11-9 所示。注意, 抛出的异常名为 `java.lang.NumberFormatException`。在 `testFloat()` 方法的内部, 将 `proxyCurrencyConvertTask()` 方法的值由 12..3 修改为 12.3(或者, 如果你的语言使用逗号代替表示小数点的句号, 将 12,,3 修改为 12,3)。现在测试应该成功了, 而且你会看到一个绿色的进度条, 如图 11-10 所示。按 `Ctrl + K` | `Cmd + K` 并提交, 附带消息为 `Creates instrumentation test`。



图 11-9 失败的 `testFloat()` 方法

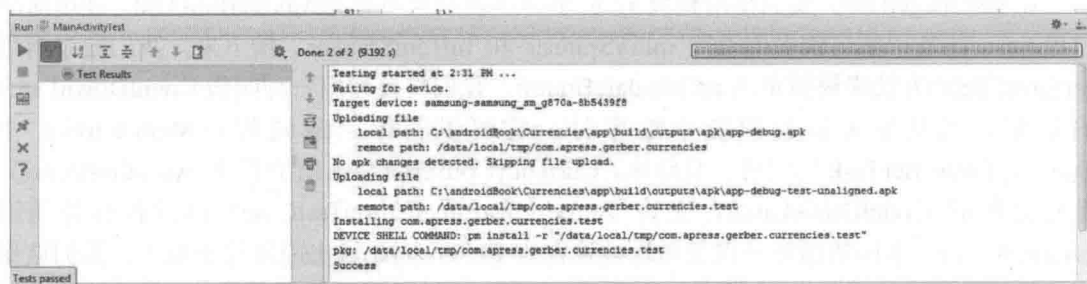


图 11-10 所有测试都成功

11.1.5 修改 Bug

刚刚运行失败的测试会突显出代码中的问题。虽然已经将键盘设置为只接收数字值，但还是可能多次输入小数点，当 Android 尝试将字符串(例如“12.3”)转换为 double 类型时，就会产生 `NumberFormatException`。需要在调用 `CurrencyConverterTask` 之前，验证用户输入的数据是否为数字值。在 `MainActivity.java` 中，创建名为 `isNumeric()` 的方法，如代码清单 11-2 所示。

代码清单 11-2 用于验证用户输入的 `isNumeric()` 方法

```
public static boolean isNumeric(String str)
{
    try{
        double dub = Double.parseDouble(str);
    }
    catch(NumberFormatException nfe) {
        return false;
    }
    return true;
}
```

修改 `mCalcButton` 的 `onClick()` 方法，在执行 `CurrencyConverterTask` 之前验证输入数据是否为数字值，如图 11-11 所示。

```
mCalcButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (isNumeric(String.valueOf(mAmountEditText.getText()))){
            new CurrencyConverterTask().execute(URL_BASE + mKey);
        } else {
            Toast.makeText(MainActivity.this, "Not a numeric value, try again.", Toast.LENGTH_LONG).show();
        }
    }
});
```

图 11-11 修改 `onClick()` 方法，使用 `isNumeric()` 验证 `mAmountEditText` 的输入值

祝贺你刚刚创建了仪器测试，使用它发现 bug 并在源代码中修改了这个 bug。按 Ctrl + K | Cmd + K 并提交，附带消息为“通过验证输入是否为数字来修改 Bug”。

11.2 使用 Monkey

Android SDK 自带一款名为 Monkey 的优秀工具，也称为 UI/应用测试器 Monkey。这款工具能够针对 App 生成随机的 UI 事件，就像一只猴子在使用 App 一样。Monkey 在做 App 压力测试时很有用。Monkey 的文档位于 developer.android.com/tools/help/monkey.html。

注意

除了 Monkey 之外，一款名为 MonkeyRunner 的工具也可以创建并运行 Python 脚本来进行自动化应用测试。MonkeyRunner 与 Monkey 无关。此外，掌握 MonkeyRunner 还需要了解如何使用 Python 脚本，而这超出了本书的讨论范畴。如果对学习关于 MonkeyRunner 的更多知识感兴趣，那么请参见 developer.android.com/tools/help/monkeyrunner_concepts.html 提供的文档。

首先，单击位于 IDE 底部边栏的 Terminal 窗口按钮，在 Android Studio 中打开终端会话。在工具栏组合框中选择 App 并单击绿色的 Run 按钮，启动 Currencies App。App 运行起来之后，在终端输入以下命令，然后按 Enter 键，如图 11-12 所示。

```
adb shell monkey -p com.apress.gerber.currencies -v 2000
```

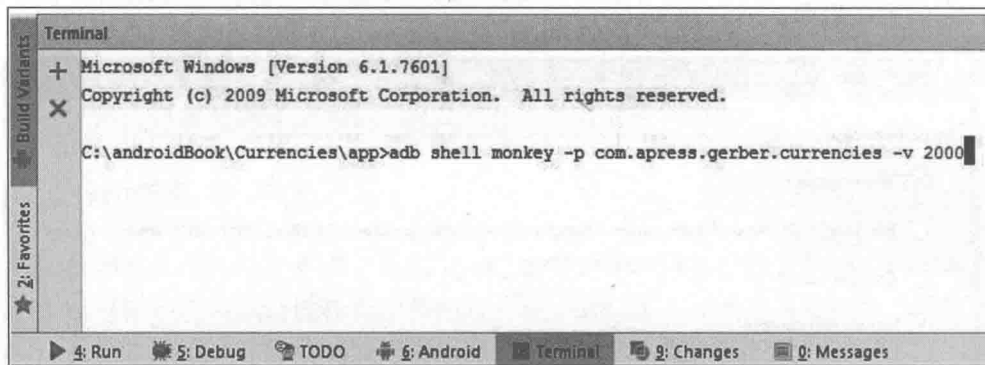


图 11-12 打开终端会话，输入 monkey 命令并接着按 Enter 键

首先，可以从这条命令看出 Monkey 使用 adb(或者说是 Android 调试桥)，它允许你与正在运行设备的操作系统 shell 交互。如果执行此命令之前忘记了启动 App，那么 Monkey 将无法工作。-p 开关告诉 Monkey 将随机 UI 事件限定在 com.apress.gerber.currencies 包中。-v 开关告诉 Monkey 尽可能详细地汇报事件和异常；如果 Monkey 确实抛出了异常，那么跟踪异常会更容易(如果报告详细的话)。最后一个参数(2000)是事件的数量。两千个随机 UI 事件应该能够暴露出 UI 中的所有问题，而且你随时都可以再次运行此命令。

警告

当运行 Monkey 时，虽然已经将 Monkey 的 UI 事件限定为特定包，但你还是会面临设备默认设置被修改的风险。例如，Monkey 关闭了 Wi-Fi 或者修改了默认语言也并不罕见。

11.3 使用分析工具

与 Android SDK 捆绑的分析工具称为 Lint。不久之前，开发者还只能通过命令行来调用这个工具。幸运的是，Lint 目前已经完全集成到了 Android Studio 中。Lint 将会分析你的源代码、XML 文件以及其他资源，查找潜在的 bug、未使用资源、低效布局、硬编码文本和其他 Android 相关的潜在问题。此外，Android Studio 拥有自己的分析工具，同时为 Java 和 Android 语法执行类似的操作，它甚至比 Lint 还要强大。总体来说，这些完全集成的工具组件将会保证你的代码整洁并尽量减少 bug。可以通过位于主菜单栏中的 Analyze 菜单来访问 Android Studio 的分析工具。

11.3.1 检查代码

检查代码是最有用且最复杂的分析操作。导航至 Analyze | Inspect Code 来运行此操作。在出现的对话框中选中 Whole Project 单选按钮并单击 OK 按钮，如图 11-13 所示。等待几秒钟，Android Studio 会分析整个项目并在 Inspect 工具窗口中显示结果，如图 11-14 所示。你将会发现 Android Lint 检查的目录被列在第一位，接着是 Android Studio 自带检查的一些目录，它们列在后面。

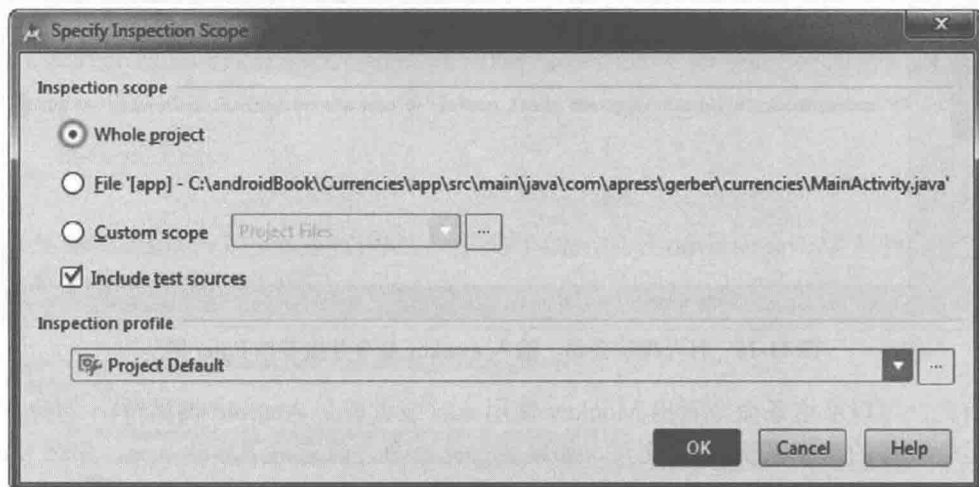


图 11-13 在 Specify Inspection Scope 对话框中选中 Whole Project 单选按钮

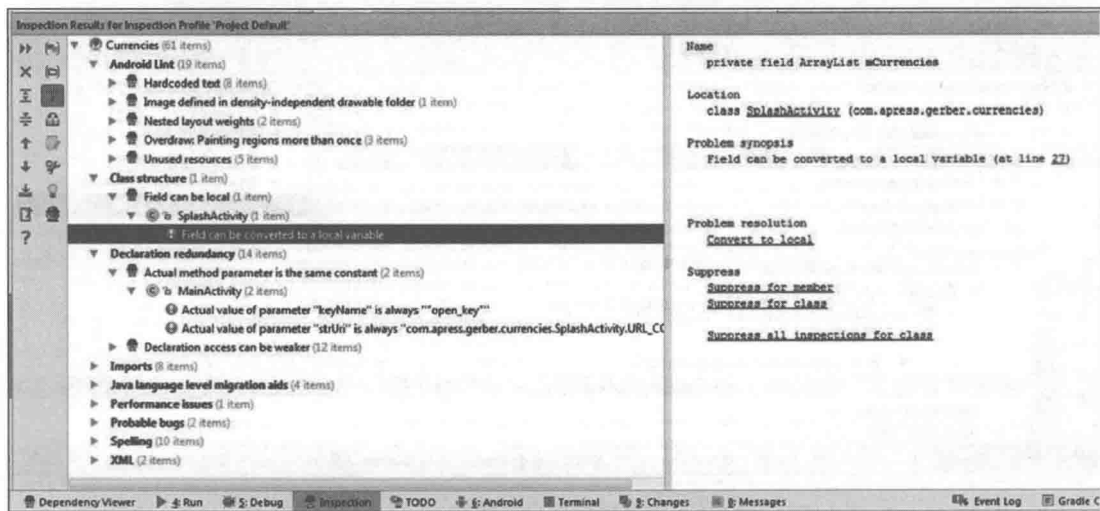


图 11-14 显示检查代码操作结果的 Inspections 工具窗口

要记住，检查代码操作识别出的问题可能并不是严重问题。因此，不要认为必须修改每一个问题。甚至在少数实例中，建议的解决方案实际上会破坏你的代码，或者与你的初衷背道而驰。因此，你应该只把 Lint 和 Android Studio 分析工具识别出的问题视作建议。

展开 Inspections 工具窗口中的目录，直到能够看到每行中的内容。当查看这些行中的内容时，注意 Inspections 工具窗口右侧面板中对每个可能问题的总结；详细内容包括 Name、Location、Problem Synopsis、Problem Resolution 和 Suppress，如图 11-14 所示。修复某个潜在问题很简单，只需直接单击 Problem Resolution 标题下面的蓝色超链接；Android Studio 将会完成其余的工作。不要试图修改检查代码操作识别出的每一个问题。如果修复了其中某个问题，那么要细心地测试 App，确保没有引入新的错误。

11.3.2 分析依赖

分析依赖操作同样位于主菜单栏的 Analyze 菜单中。分析依赖操作将会检查源代码并自动识别所有依赖。可以通过检查项目中每一个 Java 源文件的导入语句来人工完成此操作，但这会非常烦琐。分析依赖操作可以将你解脱出来，同时识别出每个依赖的位置。

Android 中的依赖可能是各种来源，包括 Java JDK、Android SDK、第三方 JAR 库(例如 Apache 公共库)和库项目(例如 Facebook)。如果合作开发者无法编译和运行项目，那么主要原因就是缺少依赖，而可以使用分析依赖操作来确定具体缺少哪些依赖。在 Gradle 之前，管理依赖是一个大问题。随着 Gradle 的使用，大多数依赖都可以自动下载，Gradle 让管理依赖变得更加简单便捷。

从主菜单栏中选择 Analyze | Analyze Dependencies。等待 Android Studio 执行操作并在 Dependency Viewer 工具窗口中查看结果，如图 11-15 所示。在左侧和右侧面板中遍历每行中的内容，注意底部面板突出显示了每个依赖在 Java 源文件中的位置。

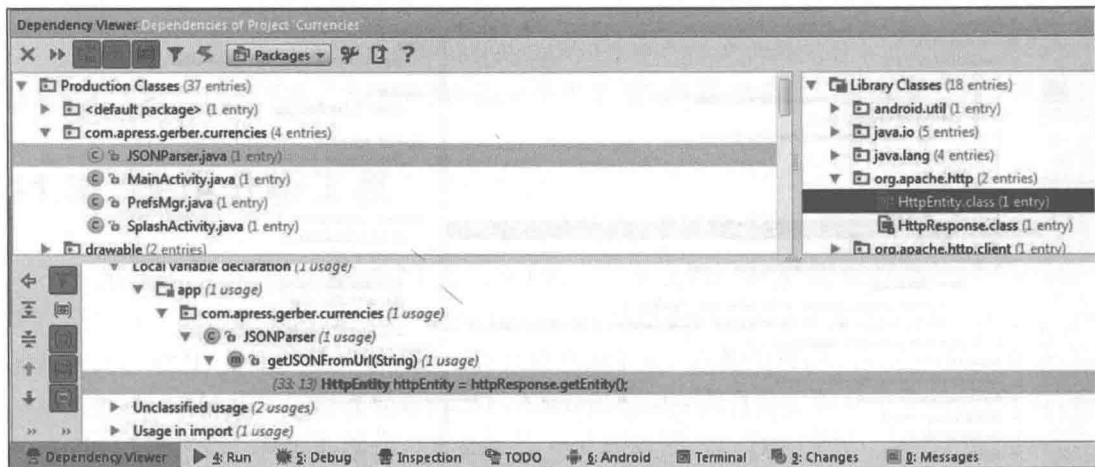


图 11-15 Analyze Dependencies 工具窗口中显示依赖于 org.apache.http.HttpEntity.class

11.3.3 分析栈轨迹

假定未处于调试模式且此时抛出了异常，那么跟踪它的最好方法是查看 logcat，它是 Android 的日志工具。logcat 中的内容非常详细，以至于很容易让你眼花缭乱，而这也是要使用分析栈轨迹操作的原因。撤消我们之前所做的 bug 修改。如果熟悉 Git，那么可以回退最后一次提交。否则，注释掉解决此 Bug 的代码，如代码清单 11-3 所示。

代码清单 11-3 将解决 Bug 的代码注释掉

```
mCalcButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        // if(isNumeric(String.valueOf(mAmountEditText.getText()))){
        new CurrencyConverterTask().execute(URL_BASE + mKey);
        // } else {
        // Toast.makeText(MainActivity.this, "Not a numeric value, try again.",
        // Toast.LENGTH_LONG).show();
        // }
    }
});
```

通过单击主工具栏中的绿色 Run 按钮，运行 Currencies App。Currencies App 启动完成之后，在 mAmountEditText 中输入 12.3(如果你的语言使用逗号代替句号，那么输入 12,,3)，然后单击 Calculate 按钮。此 App 将会崩溃，因为 12.3 不是一个数值。

按 Alt + 6 | Cmd + 6 激活 Android DDMS 工具窗口。单击 logcat 标签页，即 Android DDMS 工具窗口中最左侧的标签页。按 Ctrl + A | Cmd + A 选中 logcat 窗口中的所有文本，然后按 Ctrl + C | Cmd + C 复制这些文本，如图 11-16 所示。

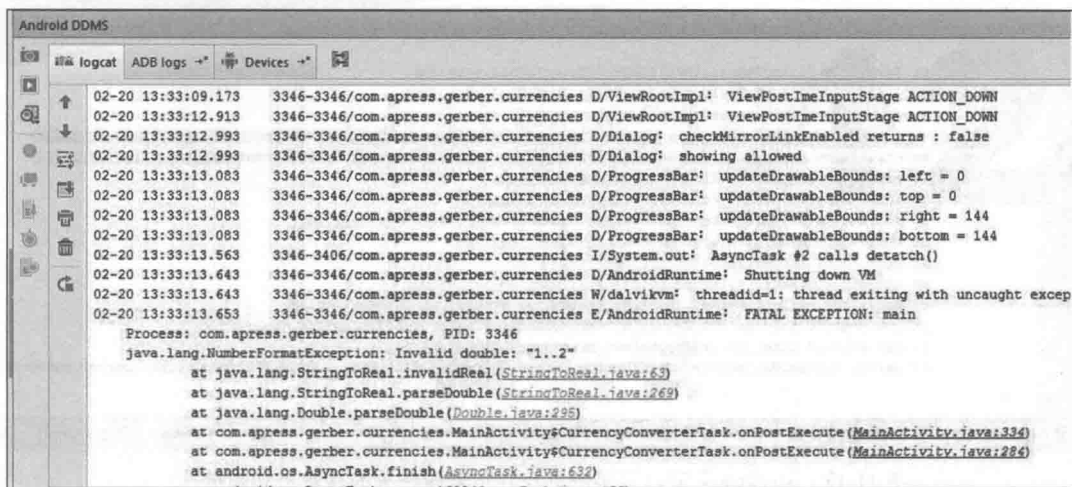


图 11-16 包含 verbose 级日志和栈轨迹的 logcat 窗口

选择 Analyze | Analyze Stacktrace, 调用 Analyze Stacktrace 操作。所有赋值给剪贴板的文本现在都会出现在 Analyze Stacktrace 对话框中。单击 Normalize 按钮并接着单击 OK 按钮, 如图 11-17 所示。Run 工具窗口将会打开, 栈轨迹也将可见(去除了多余的日志), 其中包括显示异常来源的超链接文本, 如图 11-18 所示。Analyze Stacktrace 在解析和显示栈轨迹相关内容方面表现良好, 现在就可以很轻松地分析这些内容了。

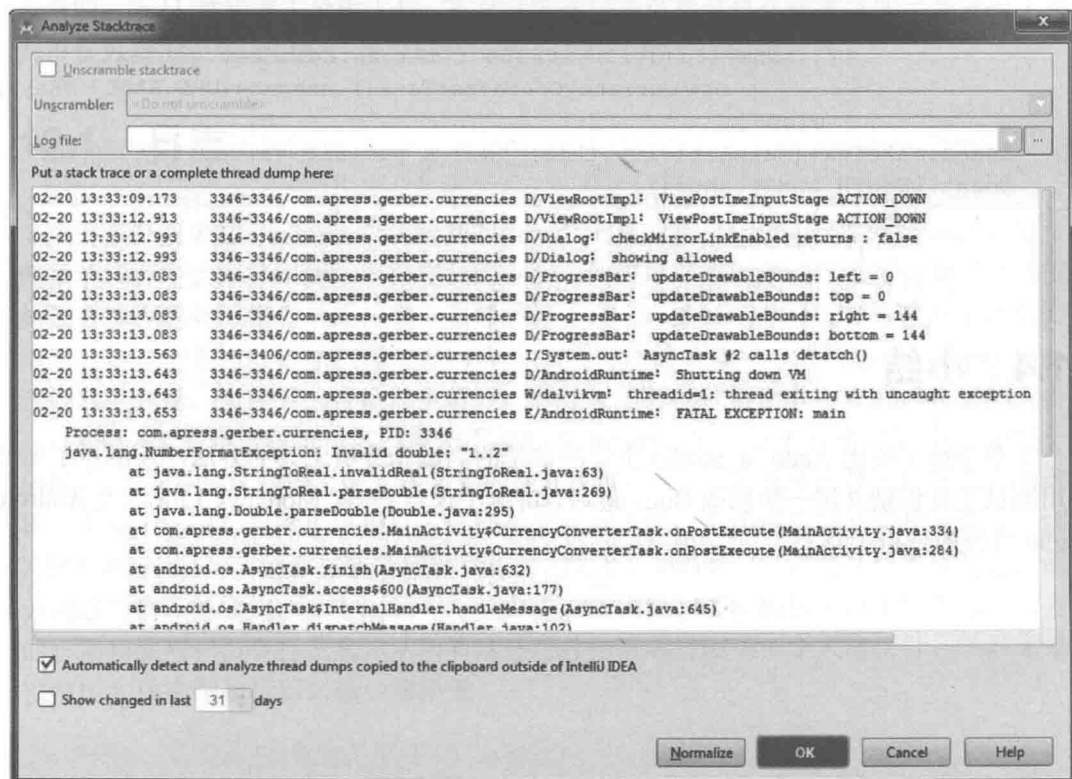


图 11-17 包含整个剪贴板内容的 Analyze Stacktrace 对话框

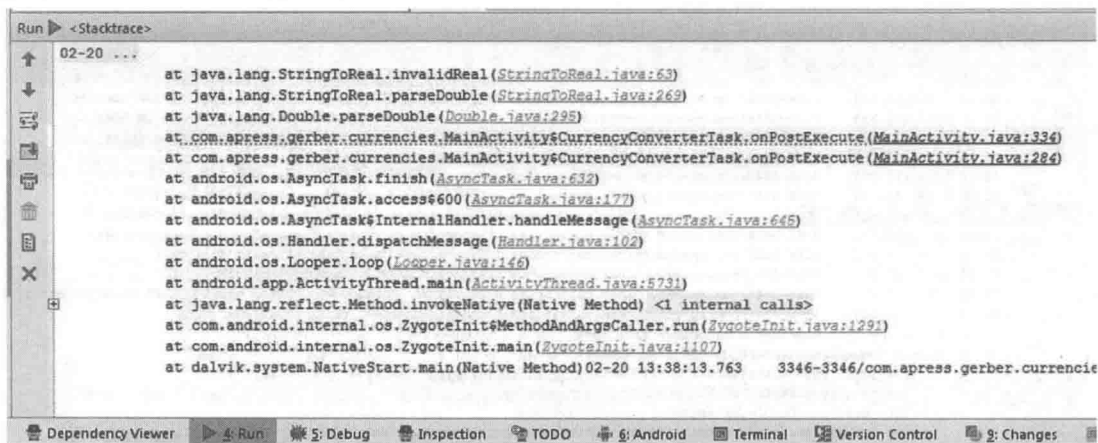


图 11-18 Stacktrace 窗口只显示了与异常来源相关的栈轨迹和超链接

可以使用 Git 回退最后一次提交，或者取消解决 Bug 代码的注释，如代码清单 11-4 所示。

代码清单 11-4 取消解决 Bug 代码的注释

```
mCalcButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if (isNumeric(String.valueOf(mAmountEditText.getText()))) {
            new CurrencyConverterTask().execute(URL_BASE + mKey);
        } else {
            Toast.makeText(MainActivity.this, "Not a numeric value, try again.",
                Toast.LENGTH_LONG).show();
        }
    }
});
```

11.4 小结

本章介绍了使用 Android Studio 中一些自带的测试和分析工具的方法，还演示了如何使用测试工具识别并进一步修改 Bug。最后，讲解了仪器测试、Monkey、Lint 以及 Android Studio 自带的分析工具。

第12章

调 试

App 越复杂，出现错误的可能性就越大。App 崩溃、在某些情况下无法运行或者执行了与期望不符的任务，这些都会让用户感到很沮丧。初级的开发方法就是假定代码总会沿着定义的路径执行。有时这也称为幸福路径。

了解代码会在何处偏离幸福路径是成为一名优秀软件开发者的关键。由于无法在开发过程中预测所有潜在的非幸福路径，因而了解 Android 开发中的多种诊断工具和技巧会很有帮助。第 11 章涵盖了分析工具；本章详细探讨调试器并再次探索其他一些分析工具，它们不仅可以用于修改错误，还可以深入观察工作中的潜在隐患。

12.1 日志

许多开发者在 Android 中接触到的第一个工具就是 Android 日志系统。日志是一种将变量值或程序状态打印到系统控制台的方法，使得可以在程序运行过程中阅读这些信息。如果有编程背景，那么对此技术会很熟悉。不过，Android 中的日志与其他平台上的稍有不同。第一个差别就是你在普通 Java 平台上所习惯的函数或方法调用。Android App 在一台机器上开发，但在另一台机器上运行，其结果是打印的输出并不在代码运行的设备上。

Android 平台上负责日志消息的框架称为 logger。它获取来自各种事件(并不限于你的应用)的输出并将这些输出保存在一系列循环缓冲区中。循环缓冲区是一种列表类的数据结构，类似于链表，但除了以串行方式连接其元素之外，它还将首尾元素相互连接。这些缓冲区包括 radio——包含与音频和电话相关的消息；events——包含系统事件消息，例如服务创建和销毁的通知；以及 main——包含主日志输出。SDK 提供了用于检查这些日志消息的一系列编程和命令行工具。从所有这些事件中查看日志类似于大海捞针。其结果是，可以使用各种操作和标识来减少输出量。

12.1.1 使用 logcat

可以在命令行中使用 `logcat`，它与绑定设备相互连接并将这些循环缓存区中的内容中转到开发控制台。它接收多种选项，表 12-1 中给出了调用它所需的语法。

```
adb logcat [option] ... [filter] ...
```

表 12-1 logcat 选项和过滤器

日志选项和过滤器	描 述
-c	清除或冲刷日志
-d	将日志打印到控制台
-f<filename>	将日志写入<filename>
-g	显示给定日志缓冲区的大小
-n<count>	设置滚动日志的数量。默认值为 4。此选项需要-r 选项
-r<kbytes>	每当到达给定千字节数时轮转日志文件。默认值是 16，而且此选项需要-f 选项
-s	将默认过滤器设置为静默
-v<format>	将输出设置为以下格式中的一种： brief 显示优先级、标签以及产生消息进程的 PID process 只显示 PID tag 只显示优先级和标签 raw 显示原始日志信息，不带任何其他字段 time 显示日期、调用时间、优先级、标签以及产生消息进程的 PID threadtime 显示日期、调用时间、优先级、标签以及每条消息所在线程的 PID 和线程 ID(TID) long 显示所有字段并使用空行分隔消息
-b<buffer>	显示指定缓冲区中的日志输出。缓存区可以是下列几种之一： radio 包含与音频/电话相关的消息 events 包含与事件相关的消息 main 主日志缓冲区(默认)

日志中的每条消息都有一个标签。标签是一个短字符串，通常用于标识发出消息的组件。组件将会是 `View`、`CustomErrorDialog` 或应用中定义的任意组件。每条消息还包含一个相关的优先级，用于确定该消息的重要程度。优先级如下：

- V: Verbose(最低优先级)
- D: Debug
- I: Info
- W: Warning

- E: Error
- F: Fatal
- S: Silent(最高优先级, 所有事情均在日志中忽略)

可以使用过滤器表达式来控制 logcat 的输出。使用正确的标记组合有助于关注与研究对象相关的输出。过滤器表达式采用 `tag:priority` 的形式。例如, `MyBroadcastReceiver:D` 将只包含来自 `MyBroadcastReceiver` 组件且标记为 `Debug` 优先级的日志消息。

Android Studio 包含内置的“设备 logcat 查看器”, 它使用图形控件处理命令行细节。插入设备或者启动模拟器, 接着单击 IDE 底部的 6 号选项卡打开 DDMS 查看器。如果尚未选择, 那么选择 `Devices | logcat` 选项卡。你的屏幕看上去应该类似于图 12-1。

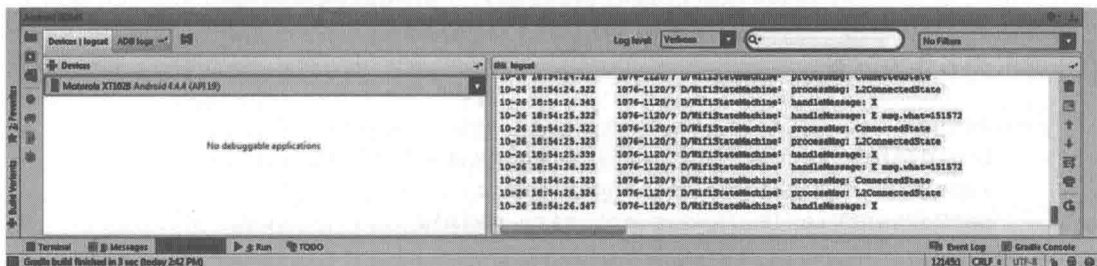


图 12-1 Android DDMS 工具窗口

在此视图的右上角, 你将会看到三个重要的过滤器控件。Log level 下拉列表根据优先级来控制过滤操作。在图 12-1 中, 此选项被设置为 `Verbose`, 它会打印所有信息。将 Log level 设置为 `Debug` 将会包含 `Debug` 或更高优先级的所有信息。与这个下拉列表相邻的是一个文本输入控件, 它将消息限定为包含这里所输入文本的那些内容。清空输入即可清除过滤器。接下来的下拉列表包含一系列预置的过滤器以及编辑或修改这些预置的选项。单击 `Edit Filter Configuration`, 打开 `Create New Logcat Filter` 对话框。如图 12-2 所示, 这个对话框包含用于修改全部预置过滤器的控件。

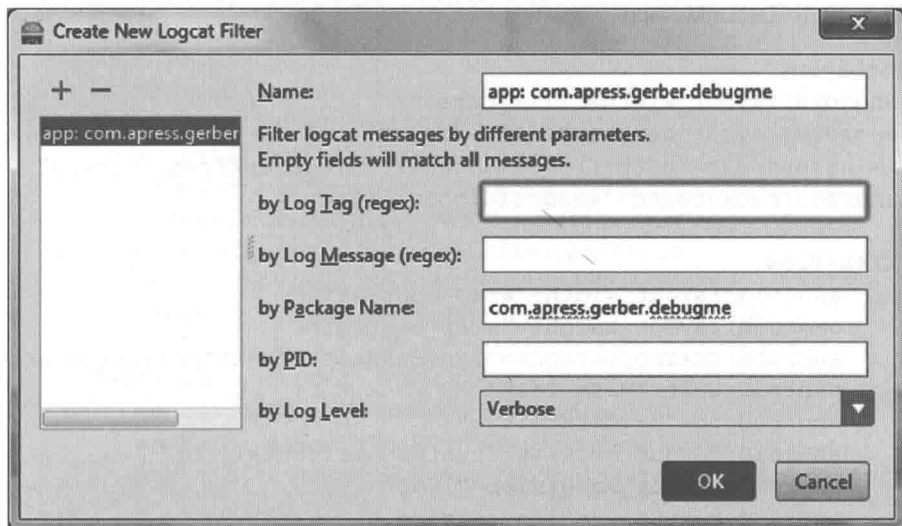


图 12-2 Create New Logcat Filter 对话框

也可以添加、修改或删除任意自定义过滤器。这些预置可以通过标签、包名、进程 ID(PID)和/或日志级别来过滤。

12.1.2 写入 Android 日志

当 App 运行时，你可能想要知道某个方法是否实际在执行，执行是否经过方法中的某个点或者某些变量的值。SDK 在名为 `android.util.Log` 的类中定义了一些静态方法，可以使用它们来写入日志。方法使用短名称——`v`、`d`、`I`、`w`、`e` 和 `f`——对应优先级 `Verbose`、`Debug`、`Info`、`Warn`、`Error` 和 `Fatal`。每个方法接收一个标签、一个消息字符串和一个可选的 `throwable` 作为参数。所选的方法决定了与你提供的消息相关联的优先级。例如，以下代码片段是你可能会在某个 `Activity` 中找到的一条日志。它将以 `Debug` 优先级将文本 `onCreate()` 输出为日志并使用类名作为标签：

```
protected void onCreate(Bundle savedInstanceState) {
    Log.d(this.getClass().getSimpleName(), "onCreate()");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

12.2 捕捉 Bug!

大多数开发者主要关注于编写能够正常工作的软件。本节会为你介绍一个无法正常工作的 App！我们故意让它存在问题，以供调试练习之用。这个简单的数学测试 App 包含几个用于输入任意数值的文本输入框。运算符下拉框允许你从加、减、乘、除中进行挑选。在底部的文本输入框中，可以尝试回答自己构建的数学问题。Check 按钮允许你检查答案。阅读代码清单 12-1 中所示的代码，了解它是如何工作的。

代码清单 12-1 DebugMe App

```
<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@android:color/black">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Math Test"
        android:id="@+id/txtTitle"
        android:layout_gravity="center_horizontal|top"
        android:layout_marginTop="10dp"
        android:textColor="@android:color/white" />
```

```

<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="center">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editItem1"
        android:text="25"
        android:layout_above="@+id/editItem2"
        android:layout_centerHorizontal="true"
        android:layout_alignStart="@+id/editItem2"
        android:textColor="@android:color/white" />

    <Spinner
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/spinOperator"
        android:layout_centerVertical="true"
        android:layout_toLeftOf="@+id/editItem2"
        android:layout_alignBottom="@+id/editItem2"
        android:spinnerMode="dropdown" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editItem2"
        android:text="50"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:layout_margin="25dp"
        android:textColor="@android:color/white" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="\???"
        android:id="@+id/editAnswer"
        android:layout_below="@+id/editItem2"
        android:layout_centerHorizontal="true"
        android:layout_marginLeft="25dp"
        android:textColor="@android:color/white" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textView"

```

```

        android:layout_below="@+id/editItem2"
        android:layout_toLeftOf="@+id/editAnswer"
        android:textColor="@android:color/white" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="25dp"
    android:text="Check"
    android:onClick="checkAnswer"
    android:layout_toRightOf="@id/editAnswer"
    android:layout_alignBottom="@id/editAnswer"
    android:textColor="@android:color/white" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="The answer is:\nXXX"
    android:id="@+id/txtAnswer"
    android:layout_below="@+id/editAnswer"
    android:layout_centerHorizontal="true"
    android:textColor="@android:color/holo_red_light"
    />
</RelativeLayout>
</FrameLayout>

public class MainActivity extends Activity {

    private static final int SECONDS = 1000;//millis
    private Spinner operators;
    private TextView answerMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log.d(this.getClass().getSimpleName(), "onCreate()");
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        answerMessage = (TextView) findViewById(R.id.txtAnswer);
        answerMessage.setVisibility(View.INVISIBLE);
        operators = (Spinner) findViewById(R.id.spinOperator);
        final ArrayAdapter<CharSequence> adapter =
        ArrayAdapter.createFromResource(this,R.array.operators_array,
        android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_
        dropdown_item);
        operators.setAdapter(adapter);
    }

    public void checkanswer(View sender) {

```

```

        InputMethodManager imm =
        (InputMethodManager) getSystemService (Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow (findViewById(R.id.editAnswer).
        getWindowToken(), 0);
        checkAnswer (sender);
    }

    public void checkAnswer (View sender) {
        String givenAnswer = ((EditText) findViewById(R.id.editAnswer)).
        getText().toString();
        int answer = calculateAnswer ((EditText) findViewById(R.id.editItem1),
        (EditText) findViewById(R.id.editItem2));
        final String message = "The answer is:\n" + answer;
        if (Integer.parseInt(givenAnswer) == answer) {
            showAnswer (true, message);
        } else {
            showAnswer (false, message);
        }
        eventuallyHideAnswer();
    }

    private int calculateAnswer (EditText item1, EditText item2) {
        int number1 = Integer.parseInt (item1.getText().toString());
        int number2 = Integer.parseInt (item2.getText().toString());
        int answer = 0;
        switch (((Spinner) findViewById(R.id.spinOperator)).
        getSelectedItemPosition()) {
            case 0:
                answer = number1 + number2;
                break;
            case 1:
                answer = number1 - number2;
                break;
            case 2:
                answer = number1 * number2;
                break;
            case 3:
                answer = number1 / number2;
                break;
        }
        return answer;
    }

    private void showAnswer (final boolean isCorrect, final String message) {
        if (isCorrect) {
            answerMessage.setText ("Correct! " + message);
            answerMessage.setTextColor (getResources().getColor (android.R
            .color.holo_green_light));
        } else {
            answerMessage.setText ("Incorrect! " + message);
        }
    }

```

```

        answerMessage.setTextColor(getResources().getColor(android.R
            .color.holo_red_light));
    }
    answerMessage.setVisibility(View.VISIBLE);
}

private void eventuallyHideAnswer() {
    final Runnable hideAnswer = new Runnable() {
        @Override
        public void run() {
            answerMessage.setVisibility(View.INVISIBLE);
        }
    };
    answerMessage.postDelayed(hideAnswer, 10 * SECONDS);
}
}

```

我们的 Activity 允许用户尝试解决简单的数学问题。onCreate()方法在实例变量中保存了所有视图组件并将基础运算符(加、减、乘、除)插入到 ArrayAdapter 中。checkanswer()方法隐藏键盘,然后调用重载的 checkAnswer()方法,完成检查答案的实际工作。这个重载的 checkAnswer()方法调用 calculateAnswer()方法计算出实际答案。checkAnswer()方法接下来比较答案和给定答案并构建答案消息。如果答案与给定答案匹配,那么调用 showAnswer()并传入 true 值,表示成功;否则调用 showAnswer()并传入 false。最后,checkAnswer()方法通过调用 eventuallyHideAnswer()方法隐藏答案消息,后者令一个 Runnable 代码块在 10 秒钟以后执行。

刚开始使用 App 时,你可能不会注意到 Bug,但是它们很快就会被发现。如果阅读了示例代码并在运行它之前自己输入一遍,那么可能已经感觉到其中存在的明显缺陷。采用默认答案或者尝试做出回答,然后单击 Check 按钮。App 将会立即崩溃。尝试再次运行它。此时,针对数学问题输入一个错误答案并单击 Check 按钮。结果并没有告知你答案是否正确的可视化反馈!你可能觉得自己已经知道了崩溃的来源位置,但与这种猜测假定问题的方法不同,我们将尝试使用调试器从容地找出 Bug。

12.2.1 使用交互式调试器

Android Studio 包含一个允许设置断点的交互式调试器。在待检查行的位置,单击 Editor 左侧边栏的折叠线,即可设置断点。要记住,断点一定要设置在包含可执行语句的行上;例如,无法在包含注释的行上设置断点。当你设置断点时,Android Studio 会在折叠线中添加一个粉色的圆形图标并将整行以粉色突出显示。当以调试模式运行应用且程序执行到断点时,折叠线中的圆点会变为红色,当前行会突出显示,而且执行会暂停并进入交互式调试模式。当处于交互式调试模式时,应用的大多数状态(包括变量和线程)均显示在 Debug 工具窗口中。可以在这里仔细地检查程序的状态,甚至改变它们。

要开始调试,可以单击顶部工具栏中的 Bug 图标或者 Bug 图标右侧相邻的图标以调试模式启动程序。这将会把调试器附着到正在运行的程序上(参见图 12-3)。运行的方式取决

于尝试要捕捉的问题。你的 Bug 可能会在某些实际情况下显现，因此需要将设备带到某个特定位置或者以某种特殊方式使用它。这种情况下，将设备与电脑相连将会很不方便。对于这种情况，可以让设备进入能够令 Bug 显现的状态，然后将设备连接到电脑并启动调试器。然而，如果 Bug 发生在 App 最初启动时，那么可以调试模式启动，这样执行就会在 App 启动时立即暂停。还有第三种方式，可以在 Android 设备的设置中将 App 设置为可调试，让其等待调试器附着。当试图捕捉在 App 启动时发生的问题，但又不想上传和替换已安装在设备上的实际 App 时，这种方法会很有用。

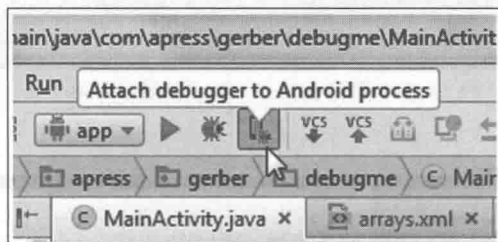


图 12-3 在运行过程中附着调试器

首先在 MainActivity 中每个方法的第一行添加断点。当不太确定问题的位置而且没有太多办法时，这种方式可以很好地工作。然而，随着 App 复杂度的增大，此方式将不再适用。在每个方法的第一行，单击左侧边栏的折叠线，你应该会看到类似于图 12-4 所示的情形。

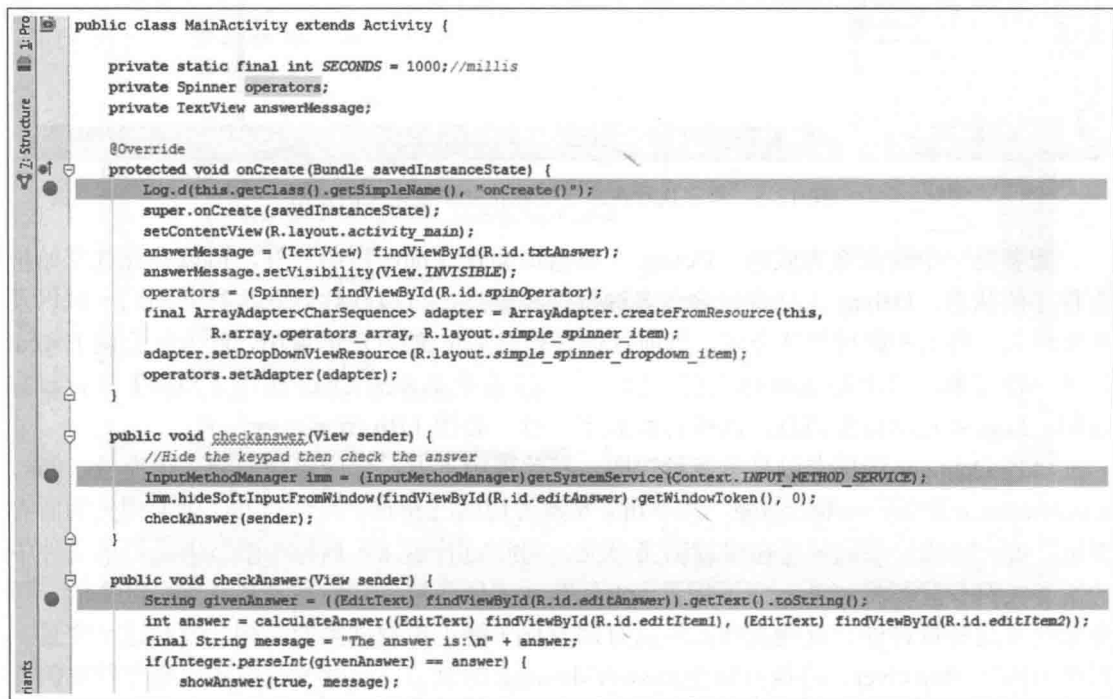


图 12-4 向 MainActivity 类中的每个方法添加断点

单击 Run | Debug App，等待 Android Studio 构建并在设备上启动 App。当 App 启动时，

在 IDE 建立连接之前，* 你将会看到一个简单的对话框，提示 adb(Android 调试桥)正在等待调试器附着。接着，Android Studio 将会(以蓝色)突出显示 onCreate()方法中第一个断点所在的行，如图 12-5 所示。Debug 工具窗口将会打开，如果你在等待断点时运行了其他程序，那么 IDE 甚至会获取焦点并跳到前台显示。这个特性很便利，但如果正在使用社交网络或者聊天应用，则可能会被打扰，因为键盘输入可能会进入到 Editor 中并破坏你的代码，因此要注意！

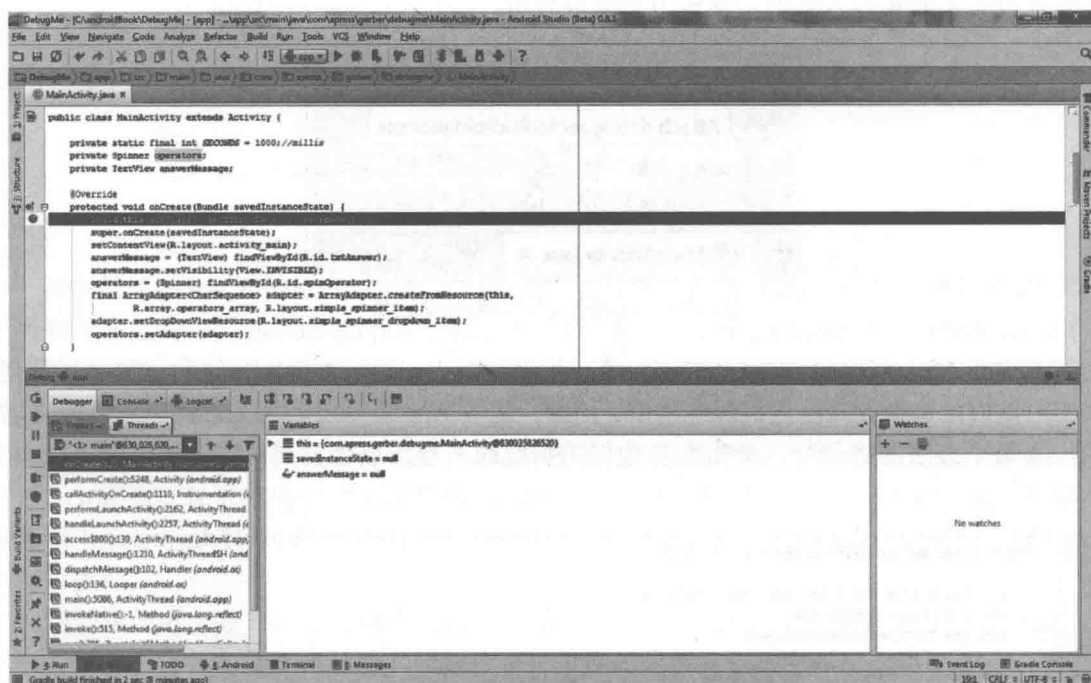


图 12-5 程序执行在断点处停止并用蓝色突出显示

随着第一个断点变为蓝色，Debug 工具窗口会在底部面板中打开，可以在此处开始检查程序的状态。Debug 工具窗口含有多特性控件，可以让你进入执行的不同区域以及单步进入、跳出和跳过某些方法。当前行恰好是对 Log.d()方法的调用，此方法会向 Logcat 发送一行文本。单击 Logcat 标签页显示日志，接着单击 Step Over 按钮【图标】执行日志语句。Logcat 显示日志消息，而执行移到下一行，如图 12-6 所示。

单击 Debugger 选项卡以显示变量视图。在此视图下方，你应该会看到三个变量：this、savedInstanceState 和 answerMessage。单击 this 变量旁边的三角形，展开与 this 对象相关的所有变量。this 对象一直表示正在执行的当前类，因此随着你深入到每个细节中，当前文件中的所有实例变量都将可见。你还将看到大量其他实例变量，它们均是从父类中派生的。筛查如此多的变量可能会有些乏味，但这有助于你了解正在调试类的结构。收起这个变量，再单击两次 Step Over，将执行移至 answerMessage 的赋值。注意变量视图中操作符实例变量的突然出现。随着执行指向实例变量的赋值，它们便开始出现在变量视图中。

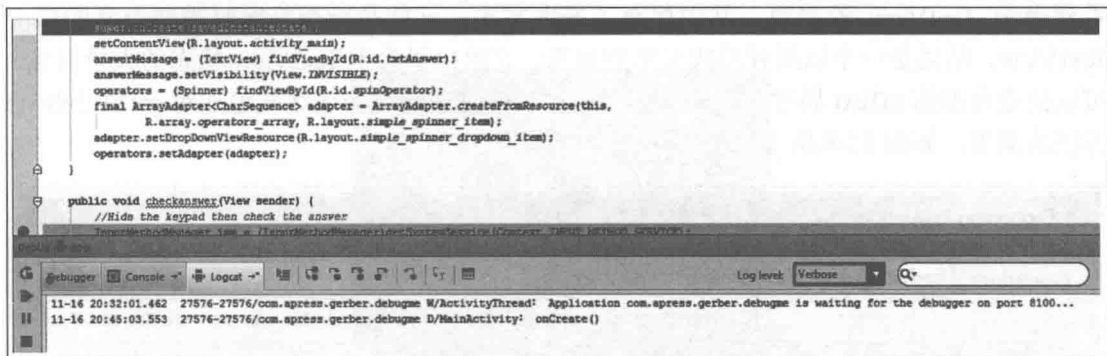


图 12-6 在单步跳过之后, logcat 视图中显示了日志消息

12.2.2 表达式求值

在运行设置 answerMessage 变量的赋值语句之前,可以截取代码行并查看表达式的值。单击并拖动鼠标,选中 findViewById(R.id.txtAnswer)表达式,然后按 Alt + F8 键,你将会看到一个类似于图 12-7 的对话框。

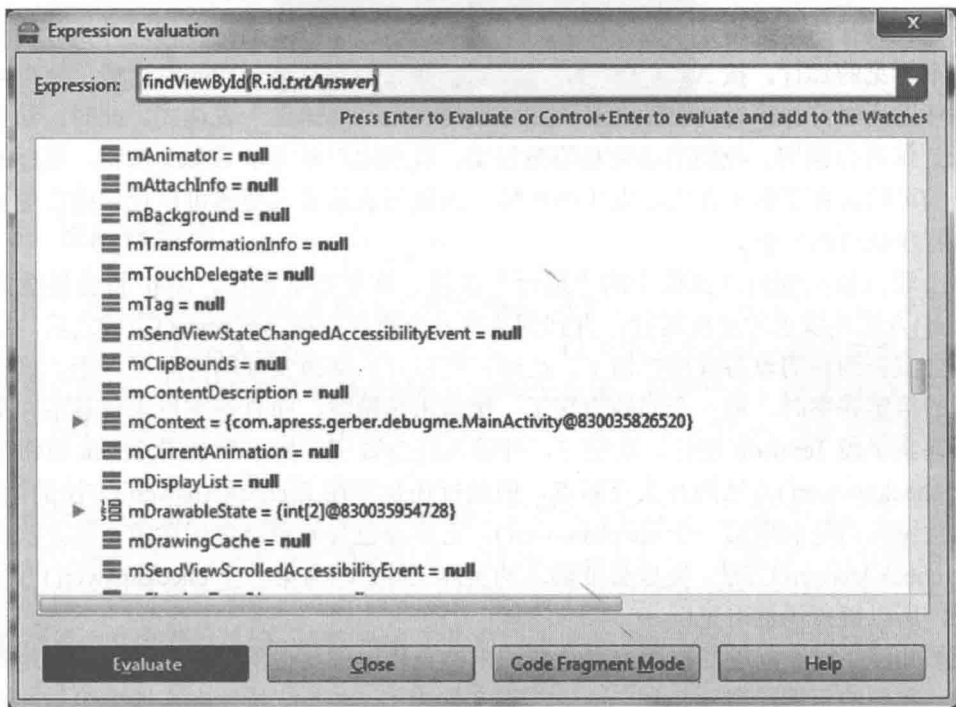


图 12-7 使用 Evaluate Expression 对话框

该表达式将被复制到 Evaluate Expression 对话框中,而且可以独立于代码行的其余部分执行。这个对话框接收任意 Java 代码片段并显示求值结果。单击 Evaluate(或者按 Enter 键,因为 Evaluate 已默认选中)计算并执行表达式。这个对话框将会显示表达式的值,可以

看到表示 `TextView` 的对象，其中保存了答案文本。这就是检查答案时需要查看的那个 `TextView`。结果是一个以展开形式显示的对象，它给出了关于 `TextView` 状态的大量信息。可以检查内部的 `mText` 属性、文本颜色、布局参数以及更多。为表达式追加对 `getVisibility()` 方法的调用，如图 12-8 所示。

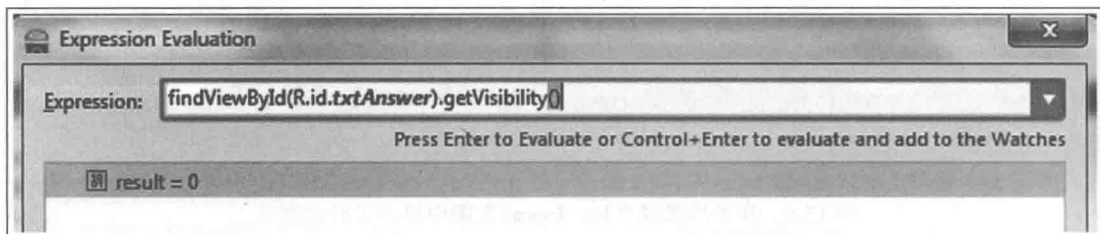


图 12-8 在 Evaluate Expression 对话框中检查答案 EditText 的可见性

表达式 `findViewById(R.id.txtAnswer).getVisibility()` 的结果是 0，它等于常量 `View.VISIBLE`。记往常量的值很难，但可以在表达式求值器中使用任意表达式。这意味着使用类似于下面的表达式，实际上就是在询问 Android Studio “我的视图可见吗？”：

```
findViewById(R.id.txtAnswer) == View.Visible
```

上述代码行的结果将是 `true`；不过，尝试单步执行接下来的两行代码并执行将该视图设置为不可见的那行。按 `Alt + F8` 键，再次打开 Evaluate Expression 对话框，使用向下箭头键循环遍历之前计算过的表达式并找到“我的视图可见吗？”表达式。此时，结果应该是 `false`，这符合预期。我们的思路是隐藏答案，直到用户单击了 Check 按钮。逐行地单步执行语句可以让你了解正在实际发生的事情，而使用表达式求值器可以让你确定变量或表达式在程序执行时的值。

单击调试器左侧控件面板中的“运行”按钮，恢复正常执行。App 将会继续执行完 `onCreate()` 方法并以正常速度运行，直到到达另一个断点。在 `onCreate()` 完成之后，用户界面应该会渲染到你的设备或模拟器上。此时，可以开始解决实际问题了。当尝试检查数学问题的给定答案时，第一个问题出现了。键盘未曾隐藏，而且答案也未曾显示。单击问号激活答案字段 `TextEdit` 控件，清空它，并输入任意数字。接下来单击 Check 按钮。虽然第一个 `checkanswer()` 方法的开头有断点，但执行还是停在了 `checkAnswer()` 方法的第一行。此处的意图应该是调用第一个 `checkanswer()`，它里面包含隐藏键盘的逻辑。此方法会调用第二个 `checkAnswer()` 方法，完成验证输入的实际工作。因为第一个 `checkanswer()` 方法没有被调用，所以键盘仍然可见！

既然知道了这个问题的原因，就让我们来检查代码的其他部分，查找没有调用此方法的原因。我们的示例使用 `activity_main` 布局文件中的 `onClick` 特性来连接按钮与方法。打开 `activity_main` 布局文件，你将会发现根本原因。Check 按钮的 `onClick` 特性被设置为 `checkAnswer`（使用混合大小写的版本），而你真正想要 `onClick` 特性调用的是 `checkanswer`（全部小写）。忽略使用仅有大小写区别的两个方法名这种明显有缺陷的模式，修改 `android:onClick` 特性中的调用，将其设置为 `checkanswer`。现在单击左侧控件面板中的调试器“停止”按钮。这将会脱离调试器并让程序恢复正常执行。再次构建并运行 App，查看

结果。你应该会看到类似于图 12-9 所示的界面。

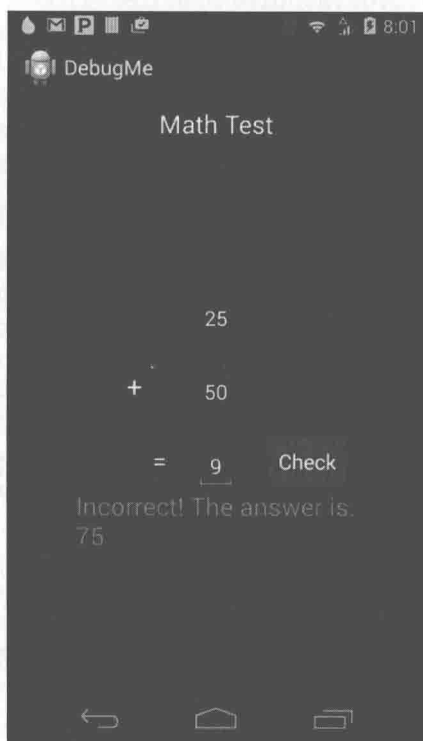


图 12-9 键盘消失了，而且答案 TextView 也可见了

12.2.3 使用栈轨迹

你使用交互式调试器发现并解决了两个 Bug。不过，还有其他问题存在。如果再次启动 App 并立刻单击 Check 按钮，App 将会崩溃。可以使用交互式调试器找出根本原因，也可以查看栈轨迹。栈轨迹会显示崩溃时栈中的每个方法，包括行号。栈中列出了一系列方法，每个方法均被其前面一个方法调用。Java 使用 Exception 或 Throwable 对象来表示程序错误。

这些特殊的对象中存有关于错误原因的元数据以及错误发生时的程序状态。异常会根据程序栈回溯到调用方法及其父级调用者，直到被捕获并处理。如果没有进行捕获或处理，它们会直接被抛给操作系统并让你的 App 崩溃。为了获得清晰的概念，最好来看一个例子。触发崩溃后，立刻查看 Android DDMS 工具窗口下方的 logcat 窗口并找到栈轨迹。

代码清单 12-2 单击复选框时产生的栈轨迹

```
03-08 20:10:56.660 9602-9602/com.apress.gerber.debugme E/AndroidRuntime :
FATAL EXCEPTION: main
    Process: com.apress.gerber.debugme, PID: 9602
    java.lang.IllegalStateException: Could not execute method of the activity
        at android.view.View$1.onClick(View.java:3841)
        at android.view.View.performClick(View.java:4456)
```

```

at android.view.View$PerformClick.run(View.java:18465)
at android.os.Handler.handleCallback(Handler.java:733)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:136)
at android.app.ActivityThread.main(ActivityThread.java:5086)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
at dalvik.system.NativeStart.main(Native Method)
    Caused by: java.lang.reflect.InvocationTargetException
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at android.view.View$1.onClick(View.java:3836)
at android.view.View.performClick(View.java:4456)
at android.view.View$PerformClick.run(View.java:18465)
at android.os.Handler.handleCallback(Handler.java:733)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:136)
at android.app.ActivityThread.main(ActivityThread.java:5086)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
at dalvik.system.NativeStart.main(Native Method)
    Caused by: java.lang.NumberFormatException: Invalid int: "???"
at java.lang.Integer.parseInt(Integer.java:137)
at java.lang.Integer.parse(Integer.java:374)
at java.lang.Integer.parseInt(Integer.java:365)
at java.lang.Integer.parseInt(Integer.java:331)
at com.apress.gerber.debugme.MainActivity.checkAnswer(MainActivity.java:46)
at com.apress.gerber.debugme.MainActivity.checkanswer(MainActivity.java:39)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at android.view.View$1.onClick(View.java:3836)
at android.view.View.performClick(View.java:4456)
at android.view.View$PerformClick.run(View.java:18465)
at android.os.Handler.handleCallback(Handler.java:733)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:136)
at android.app.ActivityThread.main(ActivityThread.java:5086)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)

```

```
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
at dalvik.system.NativeStart.main(Native Method)
```

栈轨迹可能会非常冗长，这取决于 App 的复杂程度，但学会如何分析它们可以令你增加一项宝贵的技能。在前面的栈轨迹中，你将看到列出了行号的各种方法名。列出的第一个方法 `View$1.onClick` 被认为是栈顶，它是最后被调用的方法。方法名的旁边是行号，它指向实际的源代码行，此处发生了异常。由于这个类不是我们在示例中编写的代码，因此需要查看栈的更深层。在沿着栈向下查看的过程中，你将会看到以“**Caused By**”开头的条目。可以通过这样的方式阅读它：你遇到了某个异常，它是由另一个异常导致的，而后者又是第 3 个异常导致的，以此类推。如果读到最后一个原因，你将会发现真正的问题——非法的 `int`：“???”。系统指出你向 `Integer.java` 中的 `InvalidInt` 方法传入了一个非法的整数值（一系列问号）。这是 Android 运行时的部分，已经超出了你的控制。然而，如果继续阅读，你将会看到 `invalidInt` 是由其他一些 Java 运行时方法调用的，而实际的调用者是 `checkAnswer`，它在 `MainActivity.java` 中。可以在 Logcat 视图中单击行号，它将会直接跳到以下代码片段：

```
if(Integer.parseInt(givenAnswer) == answer) {
    showAnswer(true, message);
} else {
    showAnswer(false, message);
}
eventuallyHideAnswer();
```

在此处单击了 Check 之后，将 `givenAnswer` 变量传递给 `Integer.parseInt` 方法。在同一个方法的前面几行中，你将会看到用于初始化 `givenAnswer` 变量的如下代码：

```
String givenAnswer = ((EditText) findViewById(R.id.editAnswer)).getText().
toString();
```

来自 `EditText` 控件的文本值保存在 `givenAnswer` 字符串变量中。在把这个值转换为数字之前，应该检查它是否为真正的数字，以防系统崩溃。将调用 `Integer.parseInt` 的 `if` 块修改为使用以下 `if/else if` 块：

```
if(! isNumeric(givenAnswer)) {
    showAnswer(false, "Please enter only numbers!");
} else if(Integer.parseInt(givenAnswer) == answer) {
    showAnswer(true, message);
} else {
    showAnswer(false, message);
}
```

接下来，按照如下所示定义 `isNumeric` 方法：

```
private boolean isNumeric(String givenAnswer) {
    String numbers = "1234567890";
    for(int i =0; i < givenAnswer.length(); i++){
        if(!numbers.contains(givenAnswer.substring(i,i+1))){
            return false;
        }
    }
    return true;
}
```

```

    }
    return true;
}

```

isNumeric()方法测试每个字符是否在全数字的列表中。如果方法返回 false, 那么修改后的 if 块将会调用 showAnswer(), 向用户发出一个只能输入数字的错误提示。有了它以后, 再次尝试运行 App。仍将默认答案保留为问号不做修改, 单击 Check 按钮。崩溃行为需要引起足够的重视。代码中还有一个故意放置的错误, 它会导致崩溃。我们稍后做出解释。使用 App 解决一些数学问题, 使用加法以外的其他运算符, 尝试将崩溃暴露出来。花些时间, 使用已经学到的方法, 看看是否能够找到这个问题。

本节介绍了调试的基础知识。现在你将要深入探索交互式调试器并使用它的更多特性。第 11 章讨论了“分析栈轨迹”工具, 它可以帮助你解析冗长的栈轨迹。


12.2.4 探索交互式调试的工具窗口


调试器工具窗口包含在跟踪执行时单步跳过和进入代码行的控件。默认情况下, 焦点所在的 Frames 选项卡显示了调用栈。调用栈是到达当前断点所调用方法的栈。在栈中, 最后被调用的方法位于顶部, 而调用它的方法出现在下面。属于 Android 运行时的方法以黄色显示, 与项目中定义的方法(以白色显示)相区别。


图 12-10 展示了调用栈并聚焦于两个项目方法。在本例中, checkAnswer()方法调用 calculateAnswer()方法, 因此 calculateAnswer()方法在栈的顶部。





图 12-10 使用帧视图检查调用栈

Step Over 按钮  会跳过当前行, 到达下一行。当前行中的所有指令(包括所有方法调用)均会立即执行。当到达下面一行时, App 将会暂停。

Step Into 按钮  执行当前行的所有指令, 直到该行调用的第一个方法。程序执行会在第一个方法调用中的第一行暂停。如果该行中有不止一个方法调用, 那么 Java 中定义的正常执行顺序为: 从左到右执行, 内嵌方法先执行。项目外部类中定义的方法(例如第三方 JAR 文件、Java 和 Android API 内置方法)不会被考虑。执行会跳过这些方法。

Force Step Into 按钮  的行为类似于 Step Into 按钮, 但它还会进入外部定义的方法(例如 Android SDK 中定义的那些方法)。

Step Out 按钮  完成当前方法中所有指令的执行并跳出该方法, 回到调用栈中之前的调用方法。执行会暂停在调用方法的下一行代码。

Show Execution Point  按钮会把你带到当前执行暂停的点。有时, 你可能已经远离断点并随着调试过程深入到了代码中。你可能使用导航中涵盖的一些高级特性进入各种方法调用或者正在探索类的调用者。这样的探索可能会让你丢掉最初跟踪方法的上下文。这个操作让你能够快速重新定位并返回开始处。

12.2.5 使用断点浏览器

单击 Run | Breakpoints, 打开 Breakpoints 对话框, 如图 12-11 所示。这个对话框向你展示了 App 中创建的所有断点的总体视图。如果双击列表中的任意断点, IDE 将会跳转到该行源代码。选中任何断点都会在右侧视图中显示其详细信息。详细视图让你能够禁用断点并控制当执行到达断点时 App 暂停的方式和时机。这个视图中充满了各种功能强大的选项, 允许你微调断点的行为。你能够运行任意程序语句, 在感兴趣的地方有条件地暂停 App, 甚至控制其他断点的执行。

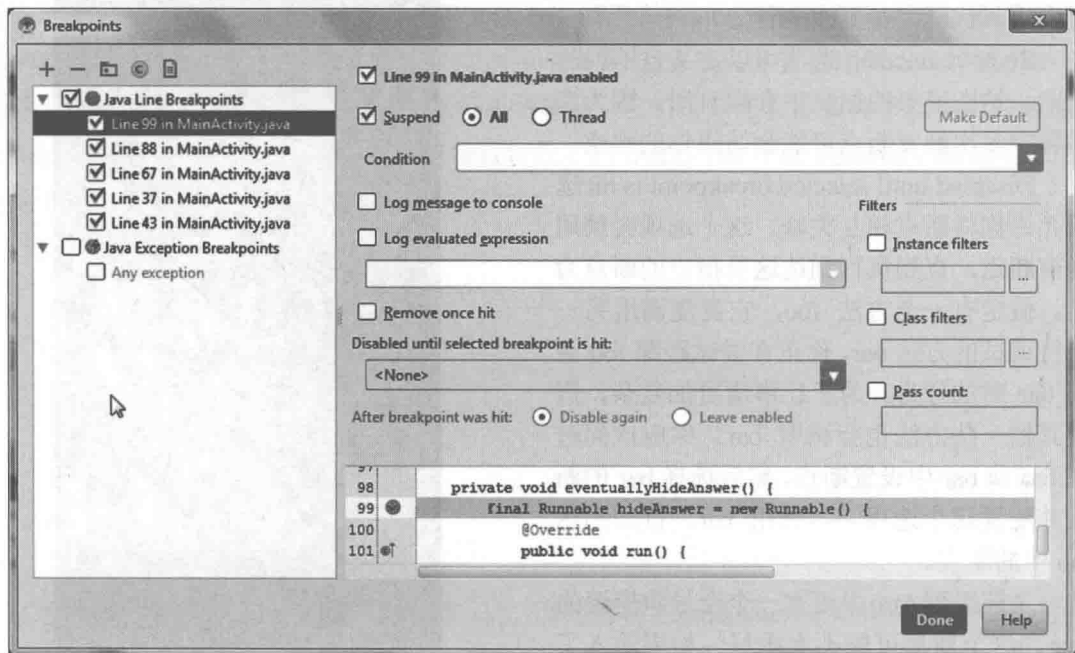


图 12-11 使用 Breakpoint 对话框设置断点属性

视图中的第一个复选框可以启用或禁用断点。Suspend 复选框控制当到达断点时的执行行为。当没有选中这个复选框时, 断点将被禁用并在 App 运行时不起任何作用。这个特性在与其他一些选项配合时会很有用, 例如 Log evaluated expression 选项。Suspend 按钮旁边的单选框可以相应地让断点暂停整个 App 或当前线程。这是一种用于辅助多线程 App 调试的高级特性, 这类 App 通常具有难以跟踪的行为。

Condition 选项允许指定激活断点的条件。下拉框接收任意合法的 Java Android 代码表达式, 表达式的结果值要求是布尔类型。表达式中的代码会在定义断点方法的上下文中执行。其结果是, 代码可以访问相应方法中可见的所有变量。它遵从 Java 作用域语法规则, 可以参考该规则以获取关于变量可见性的更多详细内容。当条件为假时, 断点会被忽略。当条件为真时, 执行将会在到达断点时暂停。

Log message to console 选项会在每次到达断点时向调试控制台输入一条通用日志消息。这条通用消息包括方法的完整签名和一个指向行号的可单击引用。要观察它的实际效果, 在 Breakpoints 对话框中遍历目前已经设置的每个断点。取消选中 Suspend 复选框并为

每一项选中 Log message to console 复选框。在 App 运行起来之后，单击 Check 按钮，触发对 checkanswer() 的调用。在调试器工具窗口中激活 Console 并找到调试器输出的日志信息。

Log evaluated expression 选项包含一个用于接收合法 Java 代码语句的文本输入框。每当到达断点时，下拉框中的代码会执行并将代码求值的结果写入调试控制台。非常类似于 Condition 选项，这段代码会在其定义方法的上下文中运行。该代码遵从与 Condition 选项相同的变量可见性规则。通常，你应该指定结果值为字符串类型的 Java 表达式，但要知道任何类型的 Java 语句都可以求值，即使是 Java 赋值语句。这使得你能够在 App 运行过程中插入代码，甚至改变行为！

Remove once hit 选项可以定义自销毁的断点。这在密集的循环中会很有用，因为在循环中多次触发断点可能会妨碍你的观察。

Disabled until selected breakpoint is hit 选项允许你将断点相互关联。这个选项会禁用当前断点，直到执行到达这里指定的断点为止。假定有一个方法 foo，它反复调用另一个待调试的方法 bar。你正在尝试跟踪 foo 调用 bar 时的行为。为了让事情更加复杂，假定其他一些方法也会调用 bar。你应该同时在 foo 和 bar 中设置断点，然后选择 bar 的断点并配置这个选项——禁用 bar，直到到达 foo 中的断点。

之前提到 App 中还有一个会导致崩溃的 Bug。这个崩溃可能不太明显。如果输入了类似于图 12-12 中所示的表达式，那么将会触发这个 Bug。可以使用本章中已探讨的所有特性来调试崩溃。查看栈轨迹可以让你直达问题的源头。

switch/case 块中的算术表达式需要做些保护，以防除数为 0。使用以下代码片段来解决崩溃：

```
switch(((Spinner) findViewById(R.id.spinOperator)).
getSelectedItemPosition()) {
    case 0:
        answer = number1 + number2;
        break;
    case 1:
        answer = number1 - number2;
        break;
    case 2:
        answer = number1 * number2;
        break;
```



图 12-12 尝试除法问题以找出崩溃！


```

case 3:
    if(number2 != 0) {
        answer = number1 / number2;
    }
    break;
}

```

12.2.6 条件断点

在调试中，更为烦琐的事情是在重复的方法调用和循环之间跟踪错误行为。取决于逻辑的复杂度，你可能需要花费宝贵的时间来单步执行大量代码并查找逻辑出现错误的特定条件。为了节省时间，Android Studio 支持条件断点。它们是只在给定条件下起作用的断点。作为演示，假定你想让 Math Test 支持指数特性。向 arrays.xml 中的 operators_array 添加指数运算符，如下所示：

```

<resources>
    <string-array name="operators_array">
        <item>+</item>
        <item>-</item>
        <item>x</item>
        <item>/</item>
        <item>exp</item>
    </string-array>
</resources>

```

由于在数组索引为 4 的位置添加了 exp，因此需要在 calculateAnswer()方法中添加另一个 case 块，如下所示：

```

case 4:
    if(number2!=0) {
        answer = 1;
        for(int i=0; i <=number2; i++) {
            answer = answer * number1;
        }
    }
    break;

```

已添加的代码是一个普通的循环，它使用第二个数作为计数器，让第一个数循环乘以自身。此时，这些特意设定的 Bug 对于你来说可能并不显而易见。构建并运行 App，尝试计算 2 的 8 次方这个数学问题。图 12-13 展示了修改之后的界面。

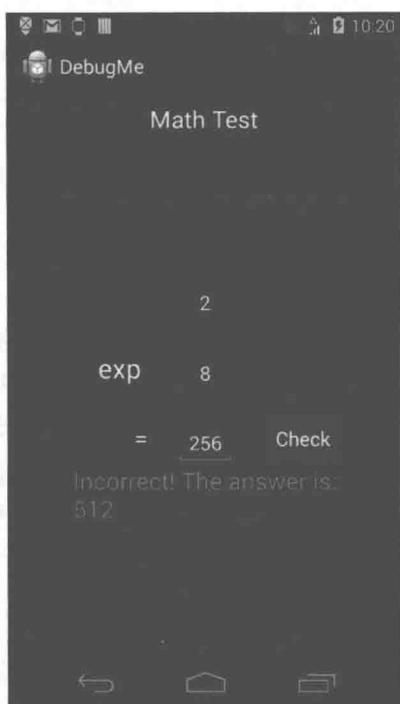


图 12-13 指数答案正确，但 App 提示错误

App 计算出了 512 这个错误的结果。需要使用交互式调试器找出问题。首先，清除所有断点，避免任何不必要的暂停。单击“附着调试器”图标，进入交互调试模式并附着调试器。现在，你应该在刚刚添加的 for 循环内部放置一个断点，单步执行 8 次循环并查看得到错误结果的原因。此外，可以使用条件断点查看最后一次循环时发生了什么。单击折叠线，在此行中添加断点：

```
answer = answer * number1;
```

接着右击断点并在条件框中输入表达式 `i==8`，如图 12-14 所示。

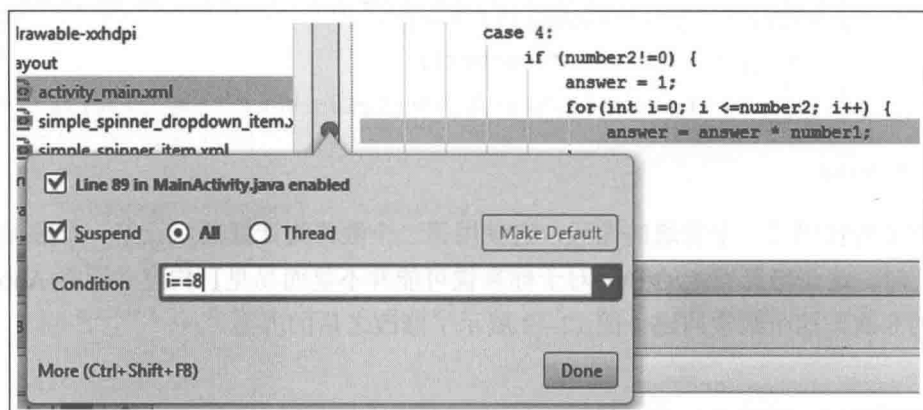


图 12-14 为断点设置条件

单击 **Done** 按钮关闭弹出窗口，接着单击设备或模拟器上的 **Check** 按钮。执行将会在断点处停止，但仅当计数器 *i* 的值增加到 8 以后。在 **Debug** 工具窗口的 **Variables** 视图中观察所有变量的状态。变量 **number1** 被赋值为 2，变量 **number2** 被赋值为 8，结果是 256。然而，此时单击 **Step Over** 图标会导致一次额外的乘法，这会改变结果值。**for** 循环的预期行为是在第 8 次循环之后结束，但它却没有。如果仔细查看 **for** 循环的条件，你将会看到 *i* 被初始化为 0，而判断条件是 $i \leq \text{number2}$ 。需要判断 $i < \text{number2}$ ，因为 *i* 是从 0 开始的。做出修改并接着正常编译和运行 App 来测试它。图 12-15 展示了做出修改之后的 App 运行情况。

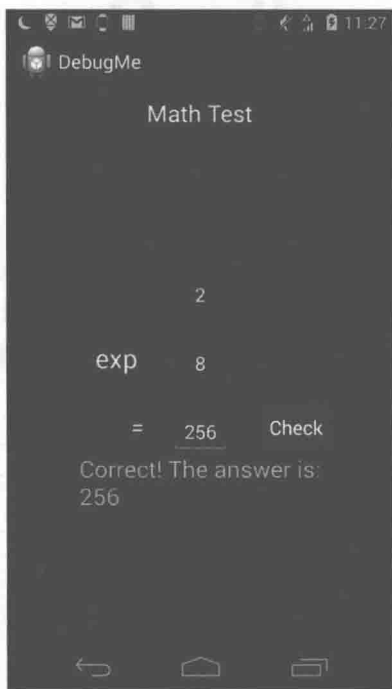


图 12-15 修改后的运行结果

12.3 小结

本章介绍了如何使用 **Android Studio** 中的各种工具和特性进行调试。你掌握了使用各种日志级别的方法以及如何直接在 **IDE** 中查看 **Android logcat**。除了探索交互式调试器并学习了它的先进特性外，你还研究了一个有缺陷的 **App** 的代码而且使用调试工具找到并修复了崩溃。伴随着代码示例，你熟练掌握了在栈轨迹中导航以及设置常规断点和条件断点的方法。本章仅仅涵盖了关于 **Android Studio** 调试的基础知识，可以根据自己的经验在调试器中创造性地组合多种特性，你也可以在调试会话中组合使用 **Android logcat** 以获得对 **App** 更为深入的了解。

第13章

Gradle

在 Android 最初发布时, Google 构建了一套基于 Apache Ant 的构建系统并将其作为 SDK 的一部分。经过多年改进, Ant 已经是一项成熟的技术, 有着庞大的贡献者社区。这些年来, 出现了其他一些构建系统, 有些已经拥有繁荣的社区。在这些构建系统中, Gradle 成为 Java 开发的下一个发展阶段。本章探讨 Gradle 并结合示例讲解如何将其用于开发和维护 Android App。

在阐述 Gradle 之前, 本章介绍什么是构建系统以及现有构建系统需要提升的原因。一直以来, 创建 App 或任何软件的过程都是使用特定编程语言编写代码, 然后将代码编译成可执行格式。

注意

本章后面有一个实验介绍了 Gradle 在多模块项目中的使用方法。我们建议使用 Git 克隆此实验对应的项目, 以便跟随学习进度, 尽管你将从头开始创建此项目的 Git 仓库。如果你的电脑上还没有安装 Git, 那么请阅读第 7 章。在 Windows 上打开 Git-bash 会话(或者 Mac 和 Linux 上的终端)并导航至 C:\androidBook\reference\ (如果没有 reference 目录, 那就创建它。在 Mac 上, 导航至 /your-labs-parent-dir/reference/, 并输入以下 Git 命令: `git clone https://bitbucket.org/csgerber/gradleweather.git` GradleWeather。

现代软件开发不仅包括连接和编译, 还包括测试、打包甚至最终产品的发布。构建系统通过提供完成这些任务的必要工具来满足用户的迫切需求。试想大量开发者如今面临的迫切需求: 支持最终产品的变体(调试版、发布版、付费版和免费版)、管理项目中包含的第三方软件库和组件, 以及基于外部参数向整体过程添加条件。

Android 构建系统最初是用 Ant 编写的。它基于一种扁平的项目结构, 对于一些特性支持不足, 例如编译变量、依赖管理以及将项目输出发布到中心仓库。Ant 采用基于 XML 标签的编程模型, 该模型简单且可扩展, 但许多开发者认为它很麻烦。此外, Ant 使用声明式模型。尽管 Ant 遵从一些函数式编程的原则, 但大量开发者更喜欢大多数现代编程语言经常采用的命令模式。简言之, Ant 并不直接支持类似于循环构造、条件分支和可重新

赋值属性(等同于 Ant 中的变量)这样的特性。

Gradle 构建使用 Groovy 程序语言编写, 构建在 Java 核心运行时和 API 之上。Groovy 大体采用 Java 的语法, 同时融入了自己的语法, 降低了学习难度。这增加了 Groovy 的吸引力, 由于它如此地接近 Java 语言, 以至于你只要做极少的修改就可以将大多数 Java 代码迁移到 Groovy。这也增强了 Gradle 的功能, 因为可以在 Gradle 构建的任何地方添加 Groovy 代码。鉴于 Groovy 语法与 Java 如此接近, 可以高效地在 Gradle 构建脚本中加入 Java 语法, 以便实现想要的效果。Groovy 还将闭包加入 Java 语法中。闭包是一个由花括号包围的代码块, 可以将其赋值给变量或者传递给方法。闭包是 Gradle 构建系统的核心部分, 稍后你将会学到更多相关内容。

13.1 Gradle 语法

Gradle 构建脚本实际上是遵从某些约定的 Groovy 脚本文件。因此, 可以在构建中包含任意合法的 Groovy 语句。然而, 大多数代码都由遵从简单块语法的语句构成。Gradle 构建脚本的基础结构包括配置块和任务块。任务块定义构建过程中在各个节点上执行的代码。配置块是特殊的 Groovy 闭包, 它们在运行时向底层对象添加属性和方法。可以在 Gradle 构建脚本中包含其他类型的块, 但这超出了本书的范畴。大多数情况下, 你都会和配置块打交道, 因为 Android 构建所需的 Gradle 任务已经定义好了。配置块采用以下格式:

```
label {
    //Configuration code...
}
```

其中 label 是一个特殊对象的名称, 而花括号定义对象的配置块。配置块中的代码采用以下形式:

```
{
    stringProperty "value"
    numericProperty 123.456
    buildTimeProperty anObject.someMethod()
    objectProperty {
        //nested configuration block
    }
}
```

块可以访问对象的单个属性并为它们赋值。这些属性可以是字符串、数字或对象本身。字符串属性可以接收 Groovy 方法调用返回的字符串值或数值。字符串值采用类似于 Java 的规则。不过, 字符串可以用双引号、单引号或 Groovy 中用于表示字符串的其他方式来标识。对象属性使用内嵌块来设置它们的单个属性。

Gradle 构建脚本遵从特定的标准。根据这个标准, 构建脚本的顶部用于声明 Gradle 插件。这些是用 Groovy/Gradle 编写的组件, 用于增加或扩展 Gradle 特性。插件声明采用这样的格式——`apply plugin: 'plugin.id'`, 其中 `plugin.id` 是想要使用的 Gradle 插件标识符。

Gradle 任务和配置块在插件定义之后且没有先后顺序。声明 Android 插件是个惯例,

它是构建脚本中的一个可用对象,可以通过 `android` 属性访问。项目的依赖通常遵从 Android 配置的形式。`dependencies` 列出了所有库,包括所有外部 API、已声明的插件以及项目中使用的组件。以下是 Gradle 构建脚本的一个示例。稍后你将学到更多细节。

代码清单 13-1 Gradle 构建脚本示例

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 20
    buildToolsVersion '20.0.0'

    defaultConfig {
        applicationId "com.company.package.name"
        minSdkVersion 14
        targetSdkVersion 20
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
}
```

13.2 IntelliJ 核心构建系统

Android Studio 在 IntelliJ IDEA 平台之上构建并继承了 IntelliJ 的大多数核心功能,它以插件的形式向核心中进一步添加了 Android 特有的功能。

插件是一种软件组件,可以从 IntelliJ 插件仓库中下载并以可插拔的方式安装或删除,非常类似于乐高积木。这些插件用于增强 IntelliJ 的功能,使用 Settings 窗口可以启用或禁用单个插件。IntelliJ Gradle 插件将 IntelliJ 的核心构建系统融入 Gradle 构建系统。这些操作通常会触发应用构建(而非 Gradle 调用),而输出通过 IntelliJ 核心反馈出来并格式化为 IntelliJ 所熟悉的格式。

13.3 Gradle 构建概念

Gradle 构建系统是一种从源文件集合中构建软件包的通用工具，它定义了一些用于构建软件的高层概念，这些概念适用于大多数项目。最常见的概念包括项目、源文件集合、构建模块、依赖模块和仓库。**project** 是硬盘上的一个位置，其中包含所有项目的源代码。Gradle 构建使用 **source** 表示源文件集合，其中包含可选的依赖列表。这些依赖是一些软件模块，包括 JAR 或 ZIP 归档文件、文本文件以及预编译二进制文件。模块可以从仓库中获取。仓库是一个以特定方式组织的模块集合，使得构建系统能够找到给定模块。它可以是硬盘上的一个位置，也可以是以标准约定来组织模块的特定 Web 站点。每个模块均包含可选的依赖集合，在构建过程中会引入它们。构建过程会将源文件集合与依赖模块组合并生成构建模块。构建过程还可以将这些模块发布到仓库，以便其他开发者或团队使用它们。

13.3.1 Gradle Android 结构

Gradle Android 项目具有层次结构，可以在项目根目录下的每个文件夹中嵌入子项目或模块。这类似于 Android Studio 底层基于的 IntelliJ 管理项目的传统方式。在 Gradle 和 IntelliJ 环境中，一个简单项目可以包含单独一个名为 **app** 的模块以及一些其他的文件夹和文件，或者包含不同名称的多个模块。相似性到此为止，Gradle 允许无限的模块嵌套。换句话说，项目还可以包含嵌套了其他模块的模块。其结果是，Android Studio 构建系统将会在后台运行 Gradle。以下列表给出了一个典型 Gradle Android 项目中所含各个文件和文件夹的简要描述。这个列表主要关注需要修改的那些文件：

- **.gradle**: 在这个文件夹下保存临时 Gradle 输出、缓存和其他支持性的元数据。
- **app**: 根据名称，在根目录的一个文件夹下嵌入单个模块。每个模块文件夹都包含一个 Gradle 项目文件，它们生成被主项目所使用的输出。最简单的 Android 项目将包含可以生成 APK 模块的单个 Gradle 项目。
- **gradle**: 在这个文件夹下包含 Gradle 包装器。Gradle 包装器是一个 JAR 文件，包含与当前项目兼容的一个 Gradle 运行时版本。
- **build.gradle**: 整体项目构建逻辑在这个文件中。它负责引入所需的全部子项目并触发对每一个的构建。
- **gradle.properties**: Gradle 和 JVM 属性保存在这个文件中。可以用它来配置 Gradle 守候进程并管理构建过程中启动 JVM 进程的方式。当使用网络代理时，可以使用这个文件辅助 Gradle 通信。
- **gradlew/gradlew.bat**: 这些文件是操作系统专有文件，用于通过包装器执行 Gradle。如果系统上没有安装 Gradle，或者没有与构建兼容的版本，那么建议使用这些文件之一来调用 Gradle。
- **local.properties**: 这个文件用于定义与本地机器相关的属性，例如 Android SDK 或 NDK 的位置。

- **settings.gradle**: 此文件在多项目构建或定义了子项目的项目中是必需的。它定义了整体构建中包含的子项目。
- **Project.iml**、**.idea**、**.gitignore**: 你可能注意到了, 在 Android Studio 中创建新项目之后, 所有这些文件就出现在了根目录中。虽然这些文件(第 7 章中已经讨论了.gitignore)不是 Gradle 构建系统的组成部分, 但它们会随着 Gradle 文件的修改而不断更新。它们会影响 Android Studio “看待”项目的方式。
- **build**: 所有 Gradle 构建输出均在此文件夹下, 其中包括生成的代码。Gradle 特意将所有输出组织到单独一个文件夹中。这简化了项目——不需要包含在版本控制中的文件列表一目了然, 而且在清理时也只需删除单独一个文件夹。

13.3.2 项目依赖

Gradle 简化了依赖管理, 它使得在多个项目之间重用代码变得很简单, 而且与平台无关。当项目的复杂度增加时, 通常需要将其分为多个部分, 这在 Android 中被称为“库”。这些库可以在不同的 Gradle 项目中独立开发或者在 Android Studio 的多模块项目中集中开发。由于 Android Studio 将模块视作 Gradle 项目, 因此这种界限变得比较模糊, 而这增强了代码共享特性。调用世界上任意一支团队开发的代码中的对象与调用本地某个模块中的对象几乎是一样的! 当项目中的代码需要调用另一个 Gradle 项目或 Android Studio 模块中的代码时, 只需要在主项目中声明依赖即可将代码绑定在一起。最终结果是将彼此分离的各个部分无缝拼接为一个完整的应用。

考虑一种简单的情况, 你的应用需要调用外部类 Foo 中的方法 bar。如果使用传统的 Android 工具, 需要找到定义了类 Foo 的项目。这需从 Web 下载, 而如果并不确定项目位置或主页, 甚至还需要奋力在网络上搜索。接下来, 你不得不做以下事情:

- 将已下载的项目保存在你的开发机上
- 可能需要通过源代码构建它
- 找到它输出的 JAR 文件, 将其复制或移动到项目的 libs 文件夹下
- 可能需要将其提交到源代码版本控制
- 将其添加到库编译路径——假定 IDE 或工具集没有自动为你完成
- 编写代码来调用方法

所有这些步骤都很容易出错, 而且如果该项目使用了其他项目的 JAR 或代码, 那么还要重复很多操作。此外, 项目的不同版本可能位于不同的位置, 或者与已经包含到 App 中的其他项目不兼容。如果项目由公司中的其他团队维护, 你可能会遇到缺少预编译 JAR 的问题, 这意味着需要将其他团队的构建文件整合到自己的构建文件中, 而这会极大地增加构建 App 所需的时间和复杂度。

有了 Android Studio 和 Gradle, 可以省去所有这些麻烦。只需要在构建文件中将项目声明为依赖, 接着即可编写代码调用方法。要了解依赖的声明方法, 回忆本章前面介绍的

示例 Gradle 文件，其中包含了下列块：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
}
```

第一行的 `compile` 告知 Gradle 在编译过程中获取 `libs` 文件夹下的所有 JAR 文件，这类类似于传统 Ant 构建脚本中使用依赖的方法，包含它的主要目的是与较老的项目兼容。第二行的 `compile` 告知 Gradle 找到版本 20 或更高的 `support-v4` 库（在仓库中位于 `com.android.support` 中）并让它在项目中可用。记住仓库是一个包含预编译模块集合的抽象位置。Gradle 将会根据需要从 Internet 下载依赖模块，将其加入到编译器的 `classpath` 中，并将它们打包到最终的 App 中。

13.4 案例研究：使用 Gradle 的天气预报项目

在本节，你将要研究一个项目——Gradle Weather，它会增量地展示各种 Gradle 构建类型。这个项目会显示天气预报。虽然某些实现使用了部分高级特性，但我们主要关注将 App 拼接起来的构建文件并截取了大量源代码清单。每一个学习步骤均有对应的 Git 分支。此项目的 Git 仓库中的每个分支均标出了对应于示例中的步骤。可以采用逐个迁出每个步骤或者在 Git 日志中查看它们修改列表的方式在本章参考这些步骤。尽情地深入研究这些代码吧！

我们首先以极简洁的方式实现 Gradle Weather，它显示假的天气预报。打开 Git 日志并找到名为 `Step1` 的分支。右击这个分支并从上下文菜单中选择 `New Branch`，创建如图 13-1 所示的新分支。将这个分支命名为 `mylocal`。你在学习的过程中将会向这个分支进行提交。Gradle Weather 基于 `FullScreen Activity` 模板构建，使用与这个模板同时生成的 `SystemUiHider` 逻辑。它显示一个持续 5 秒钟的启动画面并通过绘制来自一个硬编码普通 Java 对象（名为 `TemperatureData`）的数据来模拟加载天气预报。这个 `TemperatureData` 对象被传递给 `Adapter` 类，用于显示填充了预报信息的列表视图（第 8 章深入讨论了 `ListView` 组件）。`TemperatureData` 使用描述了指定一天天气预报的 `TemperatureItem` 类。用于此项目的构建脚本代码采用与之前定义的标准 Gradle Android 项目相同的结构。首先，你将要测试根目录中用于 Gradle 构建的文件。图 13-1 和代码清单 13-2 至 13-5 详细展现了用于控制构建的核心文件中的代码。

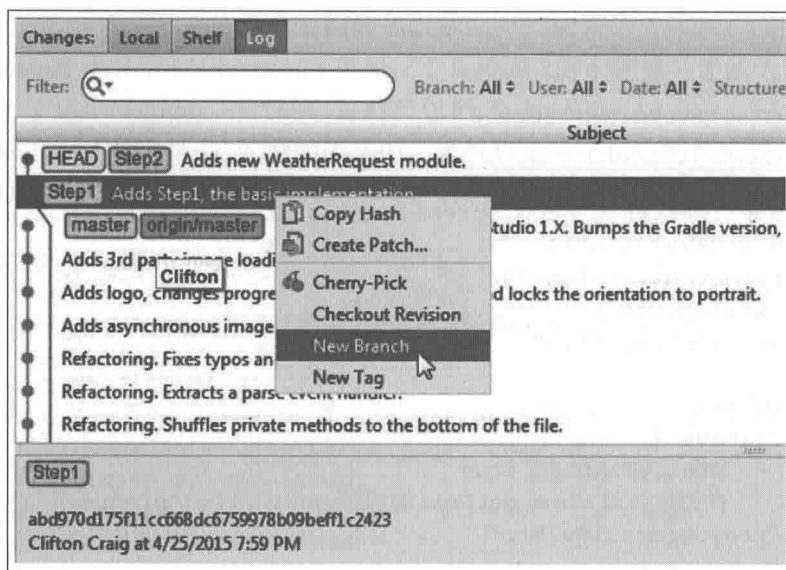


图 13-1 从 Step1 分支创建新分支

代码清单 13-2 Settings.gradle

```
include ':app'
```

代码清单 13-3 根 build.gradle

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:0.12.+'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

代码清单 13-4 local.properties

```
sdk.dir=C:\\Android\\android-studio\\sdk
```

代码清单 13-5 app\\build.gradle

```
apply plugin: 'com.android.application'
```

```

android {
    compileSdkVersion 20
    buildToolsVersion '20.0.0'

    defaultConfig {
        applicationId "com.apress.gerber.gradleweather"
        minSdkVersion 14
        targetSdkVersion 20
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
}

```

`settings.gradle` 仅定义了 App 子项目的路径。接下来是 `build.gradle`，它包含一个 `buildscript { ... }` 块。此 `buildscript { ... }` 块配置当前的编译文件，它包含这个应用中的唯一子项目——`app`。接下来，`build.gradle` 文件定义应用于所有子项目的全局构建设置，它定义了一个包含 JCenter 仓库的构建脚本块。这个可以通过 Internet 访问的 Maven 仓库包含大量 Android 依赖和开源代码项目。这个文件接着设置对 Gradle 0.12 或更高版本的依赖。最后，它将所有子项目设置为使用相同的 JCenter 仓库。

`local.properties` 文件仅包含关于 Android SDK 位置的配置。最后，我们研究一下 `app/build.gradle`，其中包含了用于 App 的所有构建配置和逻辑。第一行引入了用于当前构建的 Android Gradle 插件。接下来，它在 `android { ... }` 块中应用 Android 特有的配置。在这个块中，我们设置 SDK 版本和构建工具版本。SDK 表示编译时采用的 Android SDK API 版本，而构建工具版本表示用于诸如 Dalvik 可执行转换(dx 步骤)、ZIP 对齐等工具的版本。`defaultConfig { ... }` 块定义应用 ID(在提交到 Google Play 商店时使用)、App 需要兼容的最低 SDK 版本、目标 SDK、App 版本以及版本名称。

`buildTypes { ... }` 块控制构建的输出，它能够覆盖控制构建输出的其他配置。使用这个块，可以定义用于发布到 Google Play 商店的特殊配置。

`dependencies { ... }` 块定义了 App 的所有依赖。第一行依赖项是使用了 `fileTree` 方法调用的本地依赖，包含在 `libs` 子文件夹下的所有 JAR 文件中。第二行声明了一个外部依赖，它将会从远程仓库获取。使用给定字符串声明外部依赖时采用了一种特殊的语法。这个字符串会被冒号分隔成几段。第一段是组织 ID，用于标识创建该模块的公司或组织。第二段

是模块名称。最后一段是需要依赖模块的具体版本。

Gradle Weather 定义了 MainActivity 类和三个其他类，用于建模并显示天气数据。代码清单 13-6 展示了这个 Activity 的代码。这些类包括 TemperatureAdapter、TemperatureData 和 TemperatureItem。在此 App 的最初版本中，天气只是一个伪造的数据集，它硬编码在 TemperatureData 类中。

代码清单 13-6 MainActivity.java

```
public class MainActivity extends ListActivity implements Runnable{

    private Handler handler;
    private TemperatureAdapter temperatureAdapter;
    private TemperatureData temperatureData;
    private Dialog splashDialog;
    String [] weekdays = { "Sunday", "Monday", "Tuesday",
        "Wednesday", "Thursday", "Friday", "Saturday" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        temperatureAdapter = new TemperatureAdapter(this);
        setListAdapter(temperatureAdapter);
        showSplashScreen();
        handler = new Handler();
        AsyncTask.execute(this);
    }

    private void showSplashScreen() {
        splashDialog = new Dialog(this, R.style.splash_screen);
        splashDialog.setContentView(R.layout.activity_splash);
        splashDialog.setCancelable(false);
        splashDialog.show();
    }

    private void onDataLoaded() {
        ((TextView) findViewById(R.id.currentDayOfWeek)).setText(
            weekdays[Calendar.getInstance().get(Calendar.DAY_OF_WEEK)-1]);
        ((TextView) findViewById(R.id.currentTemperature)).setText(
            temperatureData.getCurrentConditions().get(TemperatureData.
                CURRENT));
        ((TextView) findViewById(R.id.currentDewPoint)).setText(
            temperatureData.getCurrentConditions().get(TemperatureData.
                DEW_POINT));
        ((TextView) findViewById(R.id.currentHigh)).setText(
            temperatureData.getCurrentConditions().get(TemperatureData.
                HIGH));
        ((TextView) findViewById(R.id.currentLow)).setText(
            temperatureData.getCurrentConditions().get(TemperatureData.
                LOW));
    }
}
```

```

        if(splashDialog!=null) {
            splashDialog.dismiss();
            splashDialog = null;
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public void run() {
        temperatureData = new TemperatureData(this);
        temperatureAdapter.setTemperatureData(temperatureData);
        // Set Runnable to remove splash screen just in case
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                onDataLoaded();
            }
        }, 5000);
    }
}

```

MainActivity.java 在模拟加载天气数据时显示临时启动画面(这是出于项目的后续版本考虑, 它将引入实际的数据加载)。接下来, 它使用 **TemperatureData** 类为界面上的每个视图加载数据。**TemperatureData** 类包含虚拟的预报数据集合, 如以下代码片段所示:

```

protected List<TemperatureItem> getTemperatureItems() {
    List<TemperatureItem> items = new ArrayList<TemperatureItem>();
    items.add(new TemperatureItem(drawable(R.drawable.early_sunny),
        "Today", "Sunny", "Sunny, with a high near 81. North northwest wind
        3 to 8 mph."));
    items.add(new TemperatureItem(drawable(R.drawable.night_clear),
        "Tonight", "Clear", "Clear, with a low around 59. North wind 5 to 10
        mph becoming light northeast in the evening."));
    items.add(new TemperatureItem(drawable(R.drawable.sunny_icon),
        "Wednesday", "Sunny", "Sunny, with a high near 82. North wind 3 to
        8 mph."));
    //example truncated for brevity...
    return items;
}

public Map<String, String> getCurrentConditions() {
    Map<String, String> currentConditions = new HashMap<String, String>();
    currentConditions.put(CURRENT, "63");
    currentConditions.put(LOW, "59");
}

```

```

currentConditions.put(HIGH, "81");
currentConditions.put(DEW_POINT, "56");
return currentConditions;
}

```

主 Activity 的布局包含一个 ListView，代码清单 13-7 中的 TemperatureAdapter 类将会为其填充数据。这个类接收一个 TemperatureData 对象，用于获取 TemperatureItems 列表。它会使用图 13-2 中所示的 temperature_summary 布局为每个 TemperatureItem 创建一个视图。每个 TemperatureItem(详见代码清单 13-8)都是一个数据储存器对象，其中包含获取重要数据字段的 getter 方法。这些都包含在 Activity 的主布局中，可以在图 13-3 中看到。



图 13-2 temperature_summary 布局

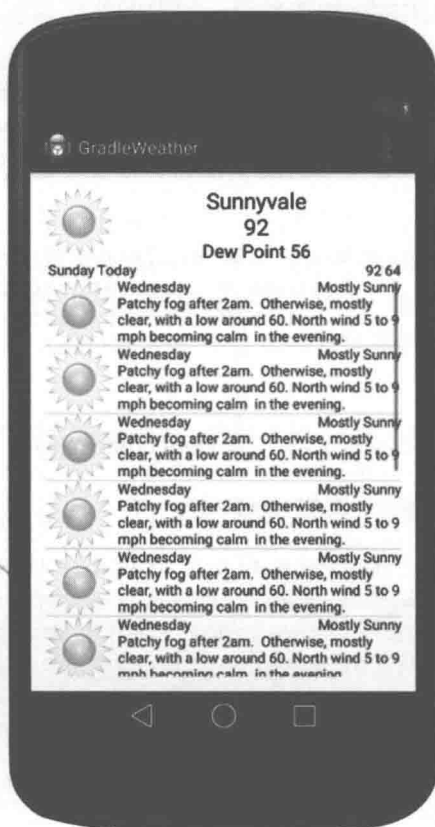


图 13-3 activity_main 布局

代码清单 13-7 TemperatureAdapter.java

```

public class TemperatureAdapter extends BaseAdapter {
    private final Context context;
    List<TemperatureItem>items;

    //This example is truncated for brevity...

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

```

```

        View view = convertView != null ? convertView : createView(parent);
        TemperatureItem temperatureItem = items.get(position);
        ((ImageView) view.findViewById(R.id.imageIcon)).
            setImageDrawable(temperatureItem.getImageDrawable());
        ((TextView) view.findViewById(R.id.dayTextView)).setText(
            temperatureItem.getDay());
        ((TextView) view.findViewById(R.id.briefForecast)).setText(
            temperatureItem.getForecast());
        ((TextView) view.findViewById(R.id.description)).setText(
            temperatureItem.getDescription());
        return view;
    }

    private View createView(ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        return inflater.inflate(R.layout.temperature_summary, parent, false);
    }

    public void setTemperatureData(TemperatureData temperatureData) {
        items = temperatureData.getTemperatureItems();
        notifyDataSetChanged();
    }
}

```

代码清单 13-8 TemperatureItem.java

```

class TemperatureItem {

    private final Drawable image;
    private final String day;
    private final String forecast;
    private final String description;

    public TemperatureItem(Drawable image, String day, String forecast,
        String description) {
        this.image = image;
        this.day = day;
        this.forecast = forecast;
        this.description = description;
    }

    public String getDay() {
        return day;
    }

    public String getForecast() {
        return forecast;
    }

    public String getDescription() {

```



```

        return description;
    }

    public Drawable getImageDrawable() {
        return image;
    }
}

```

13.5 Android 库依赖

虽然简单的 Android App 可能只包含单独一支团队开发的代码，但随着时间的推移，App 终将包含一些由其他开发者或团队实现的特性。这可以通过外部 Android 库的方式实现。Android 库是一种特殊类型的 Android 项目，可以在其中开发为 App 提供某些行为的一个或一系列软件组件——这些行为可以简单到计算两个数相乘，也可以复杂到一个列出朋友和活动的社交网络模块。Android 库让你无须花费太多力气就可以即插即用地使用外部特性。Gradle 健壮的仓库系统让你能够轻松地找到并使用来自其他公司的代码、开发源码库或自己组织中其他人开发的库。在本节中，我们将使用 Android 库依赖改进自己的 App，让其通过网络请求获取天气数据。这个修改还不足以创建一个里程碑版本，因为它尚无法有效地展现网络数据。不过，它足以演示如何在已有 Android App 中使用来自库项目的代码。我们将进一步开发能够显示数据的版本。

添加 Android 库可采用类似于从头开始创建 Android App 的流程。选择 File | New Module，打开如图 13-4 所示的 New Module Wizard。在第一个对话框中选择 Android 库。在第二个对话框中，输入 WeatherRequest 作为模块名称并选择与 App 设置一致的最低 SDK 版本，如图 13-5 所示。

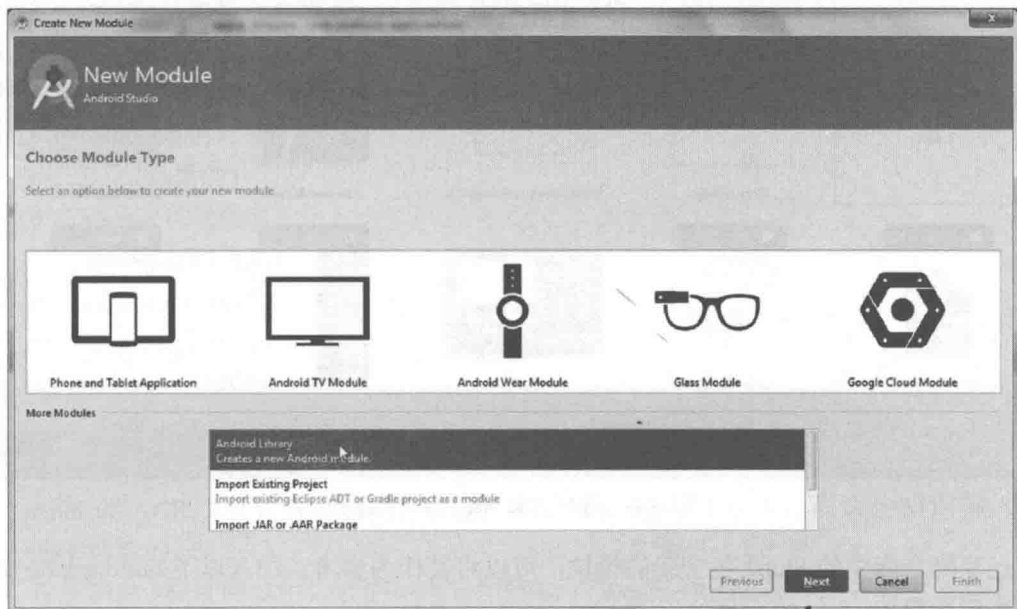


图 13-4 添加库模块

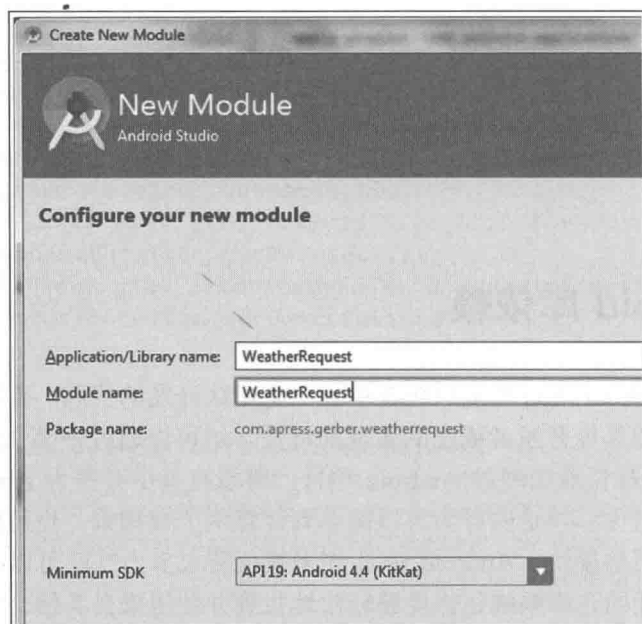


图 13-5 设置库模块的名称以及 SDK 级别

在向导的下一页选择 Add No Activity，如图 13-6 所示。单击 Finish 按钮，将库模块添加到项目中。

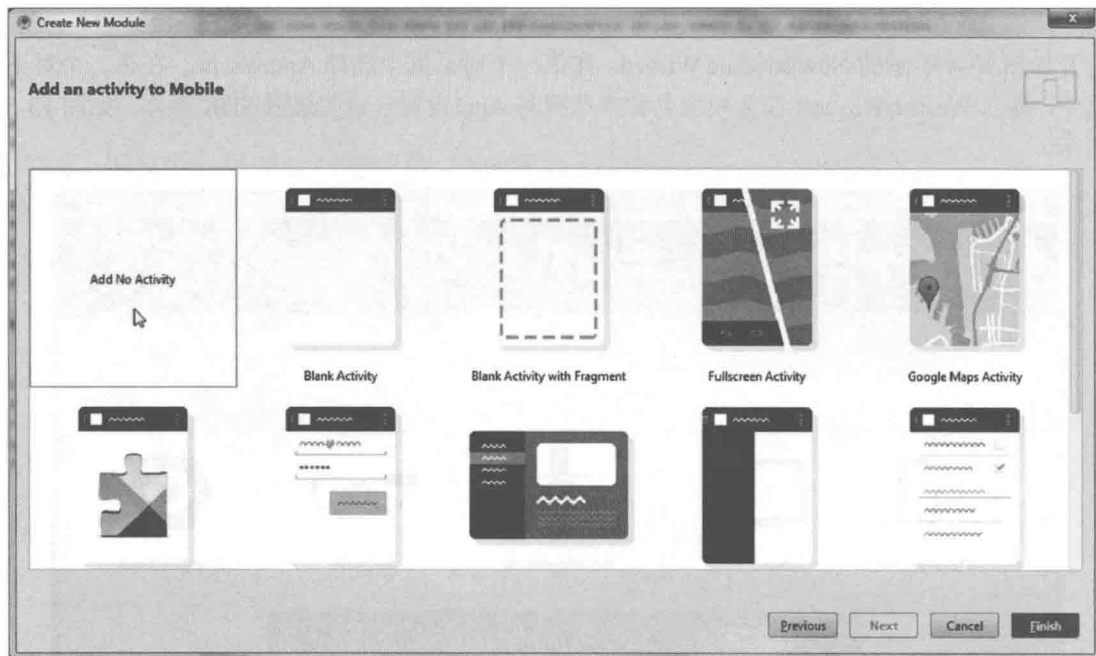


图 13-6 选择 Add No Activity 选项

已克隆仓库中的 Step2 包含新的模块，可以将其作为参考。加入以下 build.gradle 文件即可完成新模块：

```

apply plugin: 'com.android.library'

android {
    compileSdkVersion 20
    buildToolsVersion "20.0.0"

    defaultConfig {
        minSdkVersion 14
        targetSdkVersion 14
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}

```

这个构建与 App 主构建之间的主要不同是使用了 Android 库插件。此插件根据模块源文件生成一种特殊的 Android 归档文件格式——AAR。AAR 格式是 Android 中引入的新特性，它能够以库的形式在项目之间共享代码。可以使用新的 Gradle 构建系统将这些库发布到模块仓库。也可以声明依赖已作为 AAR 模块发布的任意项目并在你的项目中使用它。典型的 AAR 文件就是一个带有 .aar 扩展名的 ZIP 文件。它的结构如下：

- /AndroidManifest.xml(必需的)
- /classes.jar(必需的)
- /res/(必需的)
- /R.txt(必需的)
- /assets/(可选的)
- /libs/*.jar(可选的)
- /jni/*.so(可选的)
- /proguard.txt(可选的)
- /lint.jar(可选的)

AndroidManifest.xml 描述了归档的内容，而 classes.jar 中包含已编译的 Java 代码。资源位于 res 目录下。R.txt 文件包含了 aapt 工具的文本输出。

Android AAR 文件让你能够归档资源、本地库和/或 JAR 依赖，而这在 SDK 的早期版本中是不可能的。

在示例的 Step3 分支中，我们向项目中添加了 WeatherRequest 模块并将主 App 模块修改为包含对这个模块的依赖。这个新模块包含单独一个类——NationalWeatherRequest，它

辅助主 App 建立与国家天气服务部门的网络连接。这是一个返回任意指定位置天气信息的服务。位置以经纬度的形式给出，而响应是 XML 格式的。仔细研究并更好地理解代码清单 13-9 中所示的代码。

代码清单 13-9 NationalWeatherRequest.java

```
public class NationalWeatherRequest {

    public static final String NATIONAL_WEATHER_SERVICE =
        "http://forecast.weather.gov/MapClick.php?lat=%f&lon=
%f&FcstType=dwml";

    public NationalWeatherRequest(Location location) {
        URL url;
        try {
            url = new URL(String.format(NATIONAL_WEATHER_SERVICE,
                location.getLatitude(), location.getLongitude()));
        } catch (MalformedURLException e) {
            throw new IllegalArgumentException(
                "Invalid URL for National Weather Service: " +
                NATIONAL_WEATHER_SERVICE);
        }
        InputStream inputStream;
        try {
            inputStream = url.openStream();
        } catch (IOException e) {
            log("Exception opening Nat'l weather URL " + e);
            e.printStackTrace();
            return;
        }
        log("Dumping weather data...");
        BufferedReader weatherReader = new BufferedReader(
            new InputStreamReader(inputStream));
        try {
            for(String eachLine = weatherReader.readLine(); eachLine!=null;
                eachLine = weatherReader.readLine()) {
                log(eachLine);
            }
        } catch (IOException e) {
            log("Exception reading data from Nat'l weather site " + e);
            e.printStackTrace();
        }
    }

    private int log(String eachLine) {
        return Log.d(getClass().getName(), eachLine);
    }
}
```

作为使用 Android 库的基础示例，这个新类获取天气数据并将其输出到 Android 日志。

要在项目中包含新模块，必须编辑 App 模块中的 build.gradle 文件。找到依赖块并修改它，如下所示：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:20.+'
    compile project(':WeatherRequest')
}
```

compile project() 一行引入了项目依赖。项目位置是一条相对路径，以 project() 方法参数的形式出现，并且使用冒号作为路径分隔符。上面的示例定位了一个在名为 WeatherRequest 的文件夹中的项目，它位于主项目文件夹 GradleWeather 中。Gradle 将项目依赖视为主构建中的额外工作。在构建 App 模块之前，Gradle 将运行 WeatherRequest 依赖项目，然后查找这个项目并在 build/outputs 文件夹下找到它的输出。WeatherRequest 项目将 AAR 文件作为它的主要输出，而 App 模块中的构建将会使用它。AAR ZIP 文件会出现在 App 模块的 build/intermediates 文件夹下，它的内容会包含在编译输出中。你通常不必了解这里包含了哪些项目文件。仅在主模块的 dependencies 块中引用另一个模块是告知 Gradle 将其引入到 App 中的一种高级方法。对你的本地分支做相同的修改并提交它们。

13.5.1 Java 库依赖

项目的下个版本(在 Step4 中)包含一个纯粹的 Java 依赖，这体现了 Android 和 Gradle 构建系统的灵活性，因为它为引入大量已存在的代码敞开了大门。选择 File | New Module，打开如图 13-7 所示的 New Module Wizard。接下来在第一个对话框中选择 Java 库。在第二个对话框中输入 WeatherParse 作为库名并单击 Finish 按钮，如图 13-8 所示。

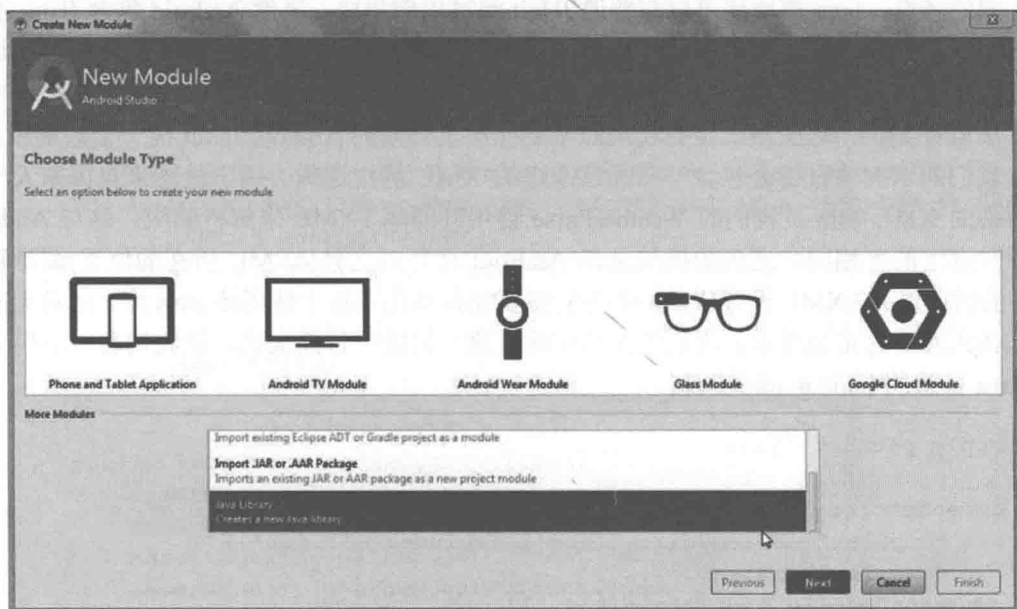


图 13-7 添加一个新的 JAR 库

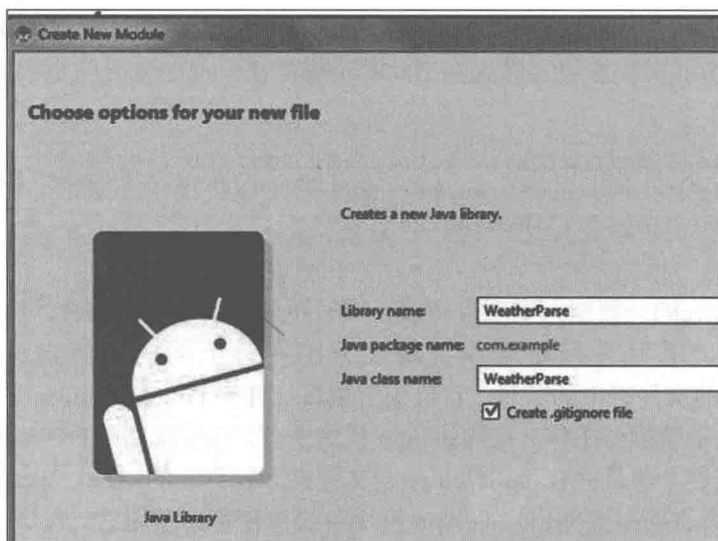


图 13-8 命名新的 JAR 库

如你所见，添加 Java 库模块类似于添加 Android 模块。主要的不同显然在于第二个对话框，它里面的选项更少一些。这是因为 Java 模块通常只包含已编译的 Java 类文件，而且它的输出是 jar 文件。将它与输出 aar 文件的 Android 库模块做对比，aar 文件中可以包含布局、本地 C/C++ 代码、资源、布局文件以及更多。

这便产生了一个问题，为什么会有人想要使用 Java 模块而非 Android 库？首先，Android 库的优势不是很明显，而如果使用 Java 模块，便可以在 Android 平台之外重用 Java 代码。这会让你在很多时候受益。考虑一种定义了用于匹配相似面孔的复杂图像处理算法的 Web 服务端解决方案，这种算法可以被单独定义为 Gradle 项目并在 Android App 中直接用于增加相同的特性。Java 模块还可以与普通 JUnit 测试用例集成。虽然 Android 包含 JUnit 框架的衍生版，但这些测试用例必须在设备或模拟器上部署和执行，而在几次迭代之后，这个过程会迅速变得十分烦琐。使用纯粹的 JUnit 测试 Java 代码，只需单击一下按钮即可在 IDE 中直接运行测试。这些测试运行起来通常要比与之等效的 Android JUnit 快一个数量级。

我们的示例项目将会包含一些经过改进的 XML 解析逻辑，用于处理来自国家天气服务网站的 XML 响应。我们的 WeatherParse 使用开源库 kXML 来解析响应。这与 Android 运行时绑定的库相同。遇到的挑战是在 Android 运行时之外(kXML 所在的位置)编译解析器。虽然可以为 kXML 设置依赖，但我们想要发布并在设备上使用该 Java 库，而且无需包含 kXML API 的冗余副本。我们将在后面解决这个问题。目前来说，让我们看一下用于添加 Java 依赖的 build.gradle 文件：

```
apply plugin: 'java'

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'kxml:kxml:2.2.2'
    testCompile 'junit:junit:4.11'
}
```

```

processTestResources << {
    ant.copy(todir:sourceSets['test'].output.classesDir) {
        fileset(dir:sourceSets['test'].output.resourcesDir)
    }
}

```

这与 Java 插件的声明没有什么不同。Java 插件配置 Gradle，将 JAR 文件作为输出，同时设置编译、测试和打包类文件所需的构建步骤。`dependencies { ... }`块定义了对 kXML 解析器和 JUnit 的编译时依赖。Gradle 将会生成一个 Java JAR 文件，其中仅包含项目中已编译的类。项目还包含两个 Java 类文件(一个用于调用解析器，另一个用于处理解析器事件)以及一个单元测试 Java 类。测试会将一个从服务获取到的典型天气 XML 响应的副本传递给解析器并验证解析器是否可以抽取天气信息。响应的副本保存在资源文件夹下。参见代码清单 13-10 中简化的单元测试代码片段。

代码清单 13-10 WeatherParseTest.java

```

public class WeatherParseTest extends TestCase {

    private WeatherParser weather;

    private String asString(InputStream inputStream) throws IOException {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(inputStream));
        StringBuilder builder = new StringBuilder();
        for(String eachLine = reader.readLine(); eachLine != null;
            eachLine = reader.readLine()) {
            builder.append(eachLine);
        }
        return builder.toString();
    }

    public void setUp() throws IOException, XmlPullParserException {
        URL weatherXml = getClass().getResource("/weather.xml");
        assertNotNull("Test requires weather.xml as a resource at the CP
            root.", weatherXml);
        String givenXml = asString(weatherXml.openStream());
        this.weather = new WeatherParser();
        weather.parse(new StringReader(givenXml.replaceAll("<br>",
            "<br/>")));
    }

    public void testCanSeeCurrentTemp() {
        assertEquals(weather.getCurrent("apparent"), "63");
        assertEquals(weather.getCurrent("minimum"), "59");
        assertEquals(weather.getCurrent("maximum"), "81");
        assertEquals(weather.getCurrent("dew point"), "56");
    }
}

```



```

public void testCanSeeCurrentLocation() {
    assertEquals("Should see the location in XML", weather.getLocation(),
        "Sunnyvale, CA");
}
}

```

右击测试方法名并单击上下文菜单中的 **Run** 选项即可运行任意单元测试。反馈会立即给出，因为测试是直接 **在 IDE 中运行的**，没有开启或选择设备、上传 APK 文件以及 App 启动这些开销。当在 **Android Studio** 中运行来自 **Java** 库的单元测试时，**Gradle** 会在后台运行并将资源从资源文件夹复制到测试指定的输出文件夹。测试用例中的 **setUp** 方法会使用已复制的 **weather.xml** 文件并使用自定义的 **asString** 方法将其读取为字符串(有一点需要注意，XML 中包含 HTML 标签，需要使用 **Java String** 类的 **replaceAll()** 方法妥善地处理它们，以防出现 XML 解析异常)。setUp 方法接下来创建 **WeatherParser** 对象，同时要求它解析 XML。包含以上代码的两个测试方法演示了如何将天气解析器用于在响应中查找当前温度和当前位置。

有了可用的天气解析 **Java** 库以后，可以随意修改并使用我们的 **Weather Request Android** 库。要实现该目的，需要做两件事情。首先，确保将 **Java** 库包含在 **settings.gradle** 文件中，该文件位于 **GradleWeather** 项目的根目录下。接着，在 **WeatherRequest** 的 **gradle** 构建中设置依赖，获取 **WeatherParse** 项目的输出。同样，**WeatherParse** 项目是一个输出单个 **JAR** 文件的 **Java** 库，但有一个需要注意的小细节。我们的 **Java** 库包含对 **kXML** 的依赖，它被认为是可传递的。我们可以按照如下所示在 **WeatherRequest** 模块中声明依赖：

```

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile project(':WeatherParse')
}

```

不过，这将会导致以下编译错误：

Output:

```

UNEXPECTED TOP-LEVEL EXCEPTION:
com.android.dex.DexException:
    Multiple dex files define Lorg/xmlpull/v1/XmlPullParser;

```

TOP-LEVEL EXCEPTION 是一个困扰许多开发者的常见问题，表示 **APK** 中多次引用了相同的文件。在本例中，异常的原因是 **Android** 已经包含了在 **SDK** 的 **kXML API** 中定义的 **XmlPullParser**。**Android SDK** 会在所有 **Android** 应用或库项目的编译过程中确保这些 **API** 可用。在构建 **WeatherParse** 模块时没有出现错误的原因是我们将它定义成了 **Java** 库。**Java** 库项目使用 **Java SDK** 编译，编译时不会包含 **Android API**。为了修正该错误，需要在 **WeatherRequest** 模块的依赖列表中删除这个可传递的依赖。我们向用于 **WeatherRequest** 模块的 **Gradle** 构建文件中添加如图 13-9 中所示的代码以解决该错误。

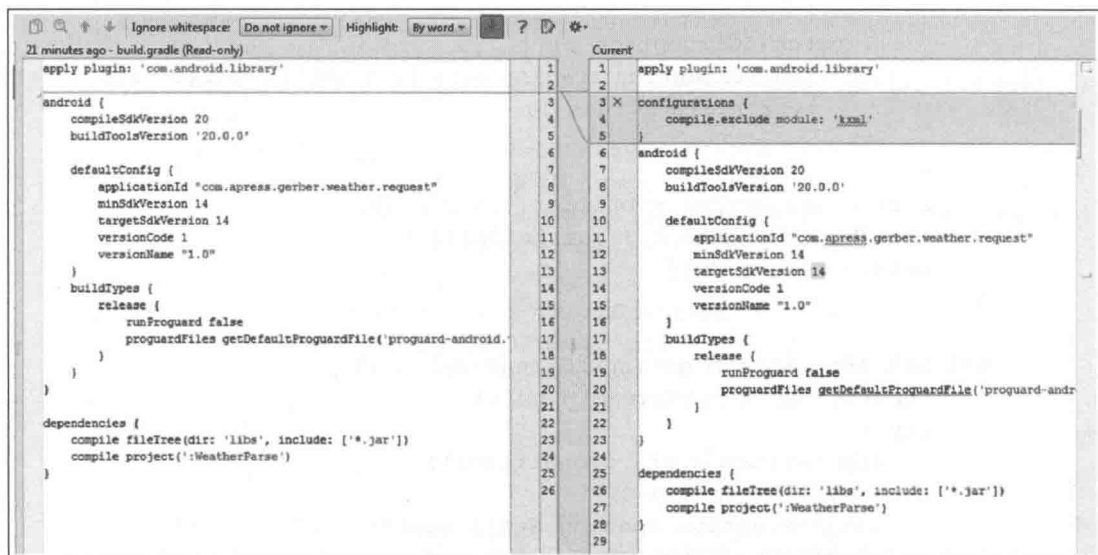


图 13-9 去除 kXML 依赖

项目现在已经能够解析 XML 天气响应并使用 XML 中的链接下载图片了。NationalWeatherRequest 对象将 URL 对象缓存为成员变量并添加 getWeatherXml 方法以使用该 URL，如代码清单 13-11 所示。

代码清单 13-11 NationalWeatherRequest.java

```
public class NationalWeatherRequest {

    public static final String NATIONAL_WEATHER_SERVICE =
        "http://forecast.weather.gov/MapClick.php?lat=%f&lon=
        %f&FcstType=dwml";
    private final URL url;

    //...

    public String getWeatherXml() {
        InputStream inputStream = getInputStream(url);
        return readWeatherXml(inputStream);
    }

    private String readWeatherXml(InputStream inputStream) {
        StringBuilder builder = new StringBuilder();
        if(inputStream!=null) {
            BufferedReader weatherReader = new BufferedReader(
                new InputStreamReader(inputStream));
            try {
                for(String eachLine = weatherReader.readLine();
                    eachLine!=null;
                    eachLine = weatherReader.readLine()) {
                    builder.append(eachLine);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```

        }
    } catch (IOException e) {
        log("Exception reading data from Nat'l weather site " + e);
        e.printStackTrace();
    }
}

String weatherXml = builder.toString();
log("Weather data " + weatherXml);
return weatherXml;
}

private InputStream getInputStream(URL url) {
    InputStream inputStream = null;
    try {
        inputStream = url.openStream();
    } catch (IOException e) {
        log("Exception opening Nat'l weather URL " + e);
        e.printStackTrace();
    }
    return inputStream;
}

```

代码清单 13-12 展示了 `NationalWeatherRequestData` 对象的细节，它使用新的 `getWeatherXML` 方法并将结果传递给新的 Java 组件——`WeatherParse`。

代码清单 13-12 `NationalWeatherRequestData.java`

```

public NationalWeatherRequestData(Context context) {
    this.context = context;
    Location location = getLocation(context);
    weatherParser = new WeatherParser();
    String weatherXml = new NationalWeatherRequest(location).getWeatherXml();
    //National weather service returns XML data with embedded HTML <br> tags
    //These will choke the XML parser as they don't have closing syntax.
    String validXml = asValidXml(weatherXml);
    try {
        weatherParser.parse(new StringReader(validXml));
    } catch (XmlPullParserException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String asValidXml(String weatherXml) {
    return weatherXml.replaceAll("<br>", "<br/>");
}

@Override
public List<TemperatureItem> getTemperatureItems() {

```

```

ArrayList<TemperatureItem> temperatureItems =
    new ArrayList<TemperatureItem>();
List<Map<String, String>> forecast = weatherParser.getLastForecast();
if(forecast!=null) {
    for(Map<String,String> eachEntry : forecast) {
        temperatureItems.add(new TemperatureItem(
            context.getResources().getDrawable(R.drawable.progress),
            eachEntry.get("iconLink"),
            eachEntry.get("day"),
            eachEntry.get("shortDescription"),
            eachEntry.get("description")
        ));
    }
}
return temperatureItems;
}

```

TemperatureAdapter 类经历了较大调整，变得非常复杂。它使用来自 WeatherRequest 的图片链接下载背景中的图片。参见代码清单 13-13 中的定义。

代码清单 13-13 TemperatureAdapter.java

```

public class TemperatureAdapter extends BaseAdapter {
    private final Context context;
    List<TemperatureItem>items;

    public TemperatureAdapter(Context context) {
        this.context = context;
        this.items = new ArrayList<TemperatureItem>();
    }

    @Override
    public int getCount() {
        return items.size();
    }

    @Override
    public Object getItem(int position) {
        return items.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view = convertView != null ? convertView : createView(parent);
        TemperatureItem temperatureItem = items.get(position);
    }
}

```

```

        ImageView imageView = (ImageView)
            view.findViewById(R.id.imageIcon);
        imageView.setImageDrawable(temperatureItem.getImageDrawable());
        if(temperatureItem.getIconLink()!=null){
            Animation animation = AnimationUtils.loadAnimation(
                context, R.anim.progress_animation);
            animation.setInterpolator(new LinearInterpolator());
            imageView.startAnimation(animation);
            ((ViewHolder) view.getTag()).setIconLink(temperatureItem.
                getIconLink());
        }
        ((TextView) view.findViewById(R.id.dayTextView)).setText(
            temperatureItem.getDay());
        ((TextView) view.findViewById(R.id.briefForecast)).setText(
            temperatureItem.getForecast());
        ((TextView) view.findViewById(R.id.description)).setText(
            temperatureItem.getDescription());
        return view;
    }

    class ViewHolder {
        private final View view;
        private String iconLink;
        private AsyncTask<String, Integer, Bitmap> asyncTask;

        public ViewHolder(View view) {
            this.view = view;
        }

        public void setIconLink(String iconLink) {
            if(this.iconLink != null && this.iconLink.equals(iconLink)) return;
            else this.iconLink = iconLink;

            if(asyncTask != null) {
                asyncTask.cancel(true);
            }
            asyncTask = new AsyncTask<String,Integer,Bitmap>() {
                @Override
                protected Bitmap doInBackground(String... url) {
                    InputStream imageStream;
                    try {
                        imageStream = new URL(url[0]).openStream();
                    } catch(IOException e) {
                        e.printStackTrace();
                        return null;
                    }
                    return BitmapFactory.decodeStream(imageStream);
                }
            };
        }

        @Override

```

```

protected void onPostExecute(final Bitmap bitmap) {
    if(bitmap == null) {
        return;
    }
    new Handler(context.getMainLooper()).post(new Runnable() {
        @Override
        public void run() {
            ImageView imageView = (ImageView)
                view.findViewById(R.id.imageIcon);
            imageView.clearAnimation();
            imageView.setImageBitmap(bitmap);
        }
    });
    asyncTask = null;
}

};
asyncTask.execute(iconLink);
}

}

private View createView(ViewGroup parent) {
    LayoutInflater inflater = (LayoutInflater) context
        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    View inflatedView = inflater.inflate(R.layout.temperature_summary,
        parent, false);
    inflatedView.setTag(new ViewHolder(inflatedView));
    return inflatedView;
}

public void setTemperatureData(TemperatureData temperatureData) {
    items = temperatureData.getTemperatureItems();
    notifyDataSetChanged();
}
}

```

每个 `ImageView` 均与一个 `ViewHolder` 相关联并使用转动图标和旋转动画来初始化,用于模拟不定进度的滚动条。主要的工作在 `ViewHolder` 的 `setIconLink` 方法中完成。这个方法会在后台下载天气图标。当下载完成时, `ImageView` 便更新为已下载的图片,同时取消滚动动画。另外,这个类文件中大量的复杂代码就是为了处理图片的加载。简化些不是更好吗?

13.5.2 第三方库

有时你不具备实现某个复杂逻辑的能力或经验。第三方库通常用于解决 Android 开发中的此类棘手问题。如前所述,调用世界上其他开发者或团队开发的代码与调用项目中其他模块的代码几乎是一样的。我们继续研究 Step5 分支,使用它演示如何在 Gradle Weather 项目中使用开源组件。我们的 App 下载了一系列图标,分别代表某天的天气状况。我们首

先在 App 模块下向 Gradle 构建添加了一行代码，如图 13-10 所示。

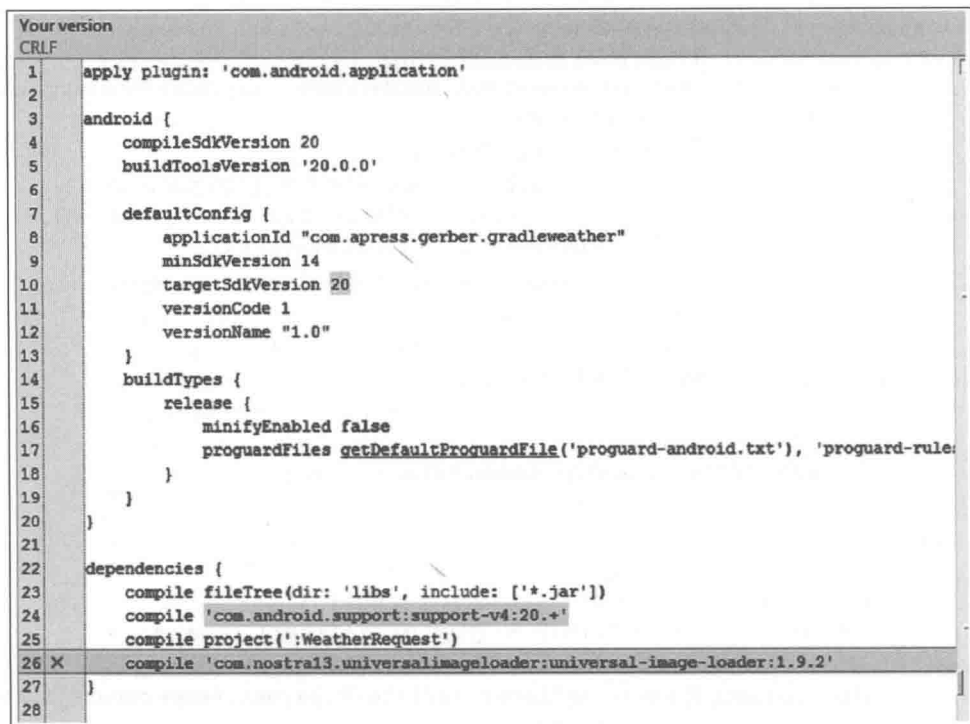


图 13-10 添加全局的图片加载器

此时，你将会立刻看到一个黄色的提示(表示 Gradle 文件已经修改)以及一个允许项目开始同步的超链接文本按钮。单击图 13-11 中所示的链接，允许 Android Studio 根据依赖同步底层的 IntelliJ 项目文件。Gradle 将在后台下载它们。

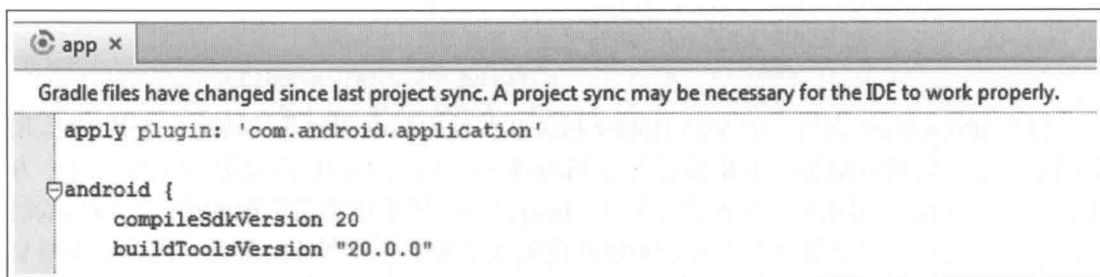


图 13-11 需要同步的 Gradle 文件

在项目同步和下载完成之后，代码可以修改为调用 API。重新访问之前的 TemperatureAdapter，我们不得不感慨在后台下载天气图标变得多么容易：

```

private final ImageLoader imageLoader;
List<TemperatureItem>items;

public TemperatureAdapter(Context context, ImageLoader imageLoader) {
    this.context = context;
}
  
```

```

this.imageLoader = imageLoader;
this.items = new ArrayList<TemperatureItem>();
}

public void setIconLink(String iconLink) {
    final ImageView imageView = (ImageView) view.findViewById(
        R.id.imageIcon);
    imageLoader.displayImage(iconLink, imageView,
        new SimpleImageLoadingListener() {
            @Override
            public void onLoadingComplete(String imageUri, View view,
                Bitmap loadedImage) {
                imageView.clearAnimation();
                super.onLoadingComplete(imageUri, view, loadedImage);
            }
        });
}
}

```

将构造函数更新为接收 `ImageLoader` 对象，并将其保存在一个实例变量中。`setIconLink` 方法只是向 `ImageLoader` 提供 `iconLink`，而 `ImageLoader` 完成了其余的烦琐工作。

13.6 打开较旧的项目

Android Studio 现在包含了强大的导入工具，用于将较旧的项目迁移到更新的 Gradle 构建系统中。这种支持几乎是透明的，而且会在打开较旧的项目时自动进行。在 Android Studio 早期的 beta 版中，很多人会因为打开这些较旧的项目而恼火。产生这些问题的部分原因是 Gradle 的快速更新周期，它可能导致较旧的构建有时无法运行。当使用较新版本的 Gradle 完成较旧的构建时会遇到这种问题。当导入较旧的项目时，使用 Gradle 包装器可以在一定程度上减轻这种烦恼，但有时这不够灵活和高效。当在更新版本的 Android Studio(例如，从 0.8x 版迁移到 1.x 版)中打开一个较旧的项目时，你可能会看到如图 13-12 所示的错误——不支持的 Android Gradle 插件。

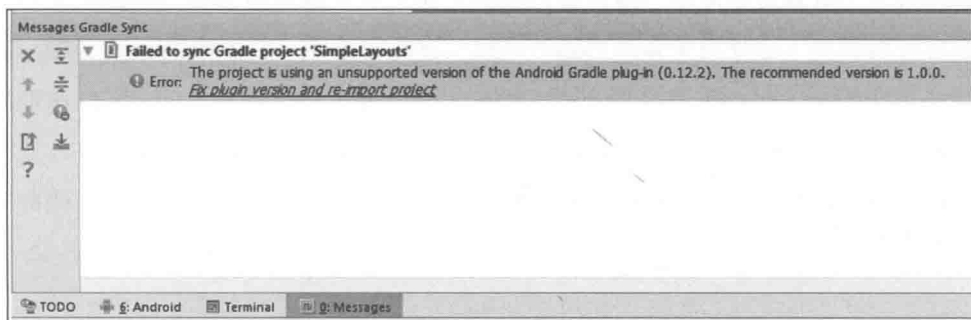


图 13-12 不支持版本错误

可以单击 `Fix Plug-in Version and Re-import Project` 链接，但又会看到图 13-13 所示的错误，提示缺少一个 DSL 方法 `runProGuard()`。在掌握了关于 Gradle 的新知识以后，可以推

测出什么是 DSL 方法，而且现在你知道应该打开 App 的 build.gradle 文件来查找这个报错的方法调用。1.x 版中废弃了这个调用，使用 minifyEnabled 取而代之。

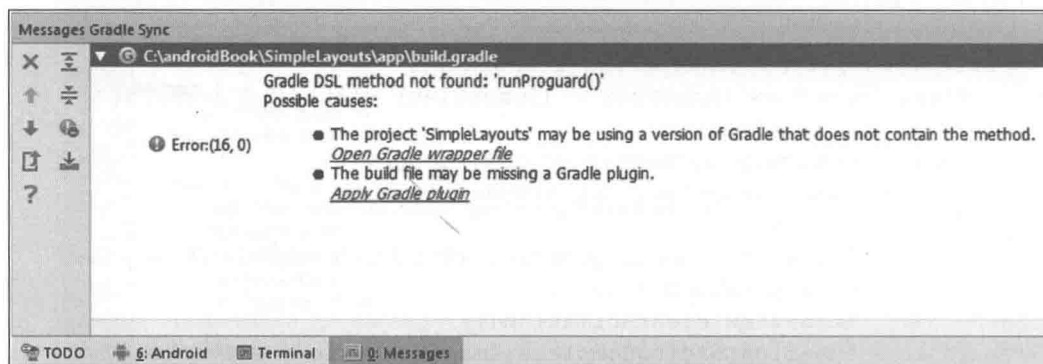


图 13-13 未找到 DSL 方法错误

13.7 小结

你已经学习了 Gradle 构建系统的基础知识。我们演示了一个带有不同类型项目依赖的多模块 Android 项目。你还看到了如何在 Android 项目中将常规 Java 代码用于 JUnit 测试。最后，你学习了如何使用内置在 Android Studio 中的导入功能打开较旧的项目。你体验了如何解决导入较旧项目时会遇到的常见问题。Gradle 还引入了健壮的依赖管理系统，让你能够轻松地在项目之间重用代码。本章只介绍了在 Android Studio 使用 Gradle 的皮毛。可以尽情地探索并改进示例项目。

第14章

更多 SDK 工具

Android Studio 是 IntelliJ IDEA 的一个特殊版本,集成了 Android 开发所需的多种工具。本章探讨可供你自由支配的多种工具,它们中的大多数都已融入各种工具窗口,有些只需一键即可调用。

14.1 Android 设备监视器

Android 设备监控器(Android Device Monitor, ADM)是 SDK 中最强大的工具之一。它让你能够从多个角度监控设备并检查诸如内存、CPU、网络使用等多种指标。要开始使用 ADM,请从 Android Studio 菜单中选择 Tools | Android | Android Device Monitor,在所打开窗口的左侧有一个 Devices 视图。

在这个界面中,你能够看到与开发机相连的所有设备,以及每台设备上正在运行的进程列表。如果 App 尚未运行,那么启动它,然后在进程列表中找到它。名称应该会遵从常见的包命名约定。如果读取进程名称时遇到困难,那么可以在 Devices 视图中重新设置每列的宽度。单击你的 App 并选中它,它将会变成 ADM 中各种工具的焦点。在这些示例中,你将要分析 Gradle Weather App。图 14-1 展示了选中 Gradle Weather App 时的 ADM 窗口。

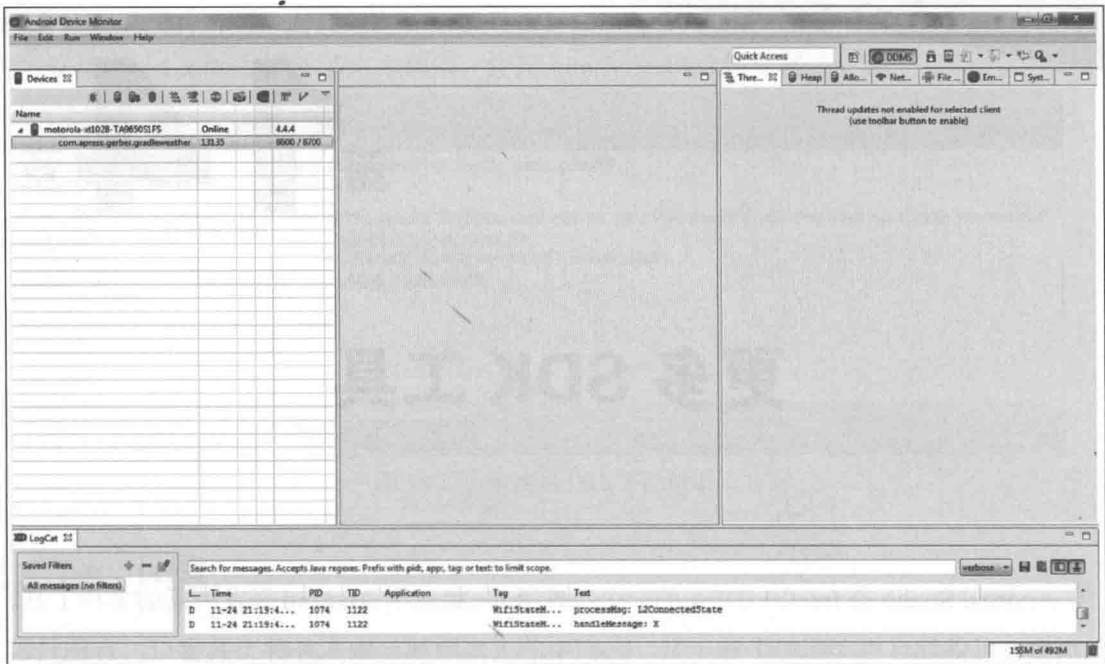


图 14-1 Android 设备监视器的主界面

14.1.1 线程监视器

选取 App 之后，可以通过单击启用 ADM 中的功能来探索其执行中的各种特性。线程活动是较容易监控的项之一。单击 Update Threads 按钮，右侧视图将会显示一个活动线程列表，内容包括 ID、状态和名称。当执行更新时，单击右侧视图中的任意线程将会显示关于其活动的详细信息。额外的详细信息将会在 Thread 选项卡下方的面板中以栈轨迹的形式展现。例如，单击名为 main 的线程，你可能会看到类似于下面的栈轨迹信息：

```
at android.os.MessageQueue.nativePollOnce(Native Method)
at android.os.MessageQueue.next(MessageQueue.java:138)
at android.os.Looper.loop(Looper.java:123)
at android.app.ActivityThread.main(ActivityThread.java:5086)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:515)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
at dalvik.system.NativeStart.main(Native Method)
```

主线程通常会遍历 android.os.MessageQueue，查找用户交互。当在屏幕上执行手势、敲击键盘或发生其他交互时，系统会将这些动作记录为消息并填充到 MessageQueue 中。在将这些消息作为事件递送给 App 之前，系统调用 nativePollOnce() 以获取它们。NativePollOnce() 是由 MessageQueue.next() 调用的，MessageQueue.next() 是由主循环调用的，而主循环是由 ActivityThread.main() 调用的。进一步查看栈，你会发现主线程开始于

Zygote.Init(), 它是设备开机时最先启动的进程之一。可以单击栈轨迹上方的 Refresh 按钮来更新它。

探索 App 中其他线程的栈轨迹, 了解它们都在做什么。在图 14-2 中, 我们研究 Gradle Weather 项目中大量 Universal Image Loader 线程中的一个并更新了栈轨迹。栈轨迹显示了从网络流读取图片并将其解码为位图的过程。

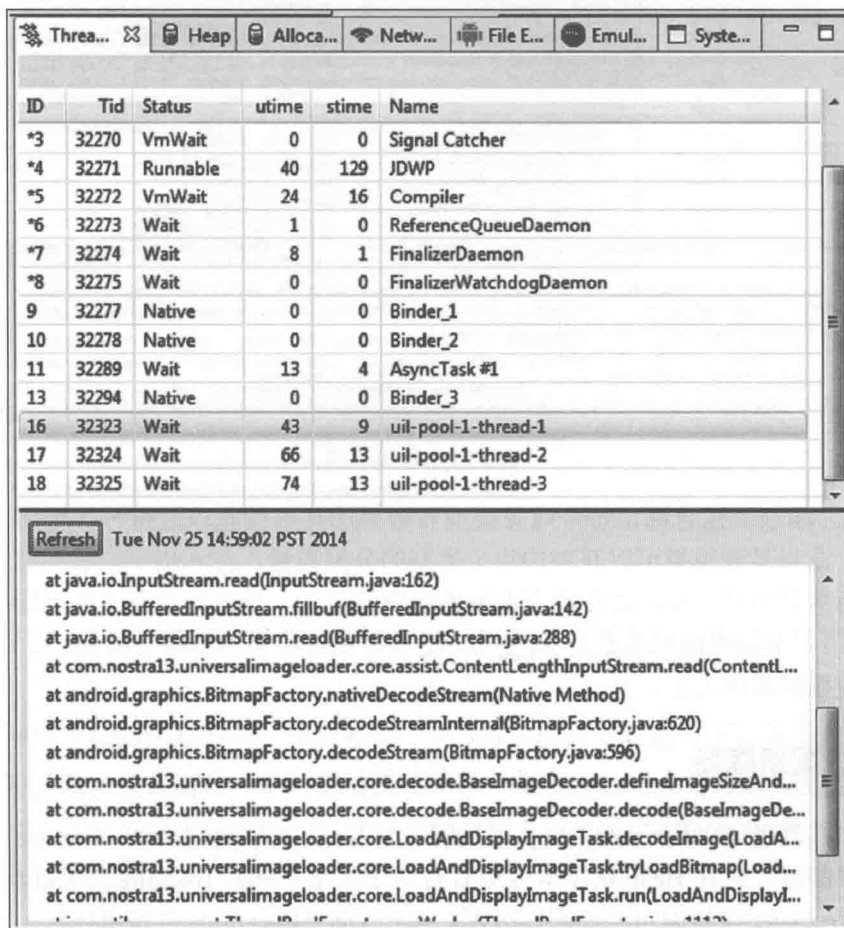


图 14-2 线程监视器

14.1.2 堆监视器

堆监视器允许你检查 App 运行过程中在堆上分配的对象。单击 ADM 窗口右侧 Threads 选项卡旁边的 Heap 选项卡, 将堆监视器置于前台。保持 App 在 Devices 面板中的选中状态, 单击 Update Heap 按钮启用堆更新, 如图 14-3 所示。每当垃圾收集器在设备上运行时都会发生堆更新; 每次执行时, 描述堆信息的最新数据都会被发送到 ADM 用户界面。与 App 随意地交互就可能会触发垃圾收集器的执行。也可以在任意时刻单击 Garbage Collect 图标(看上去像个垃圾桶)来强制执行该操作。

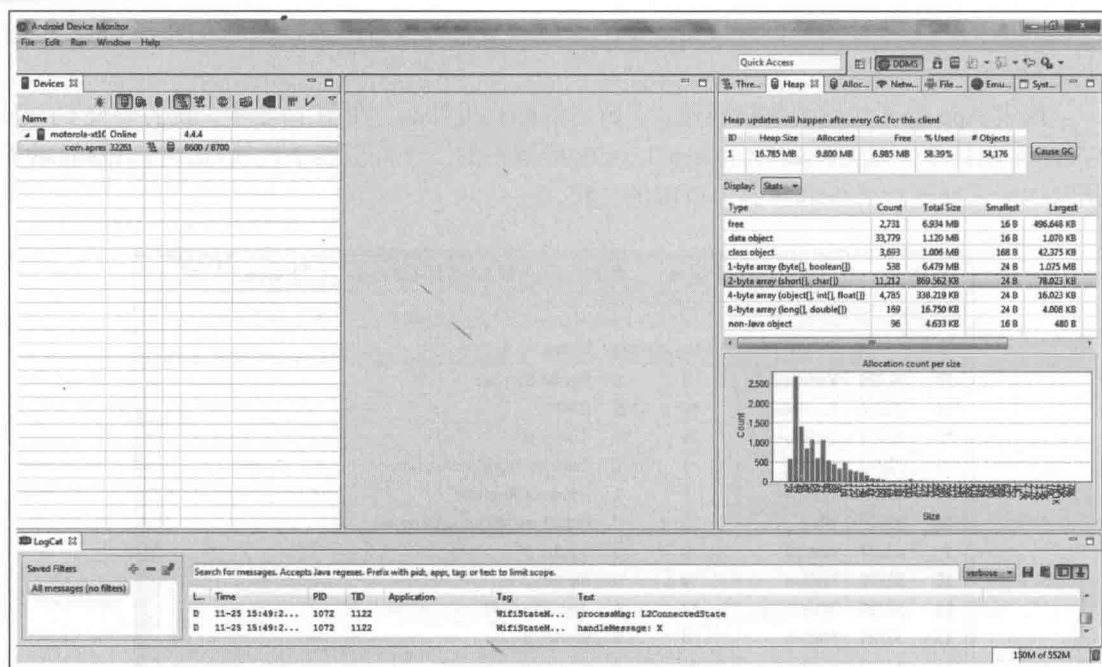


图 14-3 堆监视器

Heap 选项卡中包含标识每个对象类型和数量的详细信息以及每种类型所占的最小和最大空间。选择某种类型可以获取该特定类型的分配数量。在示例中，我们选取了数量为 11212 的两字节数组对象，它在堆上占据的总空间最大。堆详细信息下面的图表显示了超过 2500 个两字节数组的长度都是 32 字节。这些数组可能是为图标分配的，因为 32 字节是管理图像数据的最优大小。

14.1.3 分配跟踪器

分配跟踪器也可以用于跟踪 App 正在使用的内存。可以通过 Allocation Tracker 选项卡访问分配跟踪器，它在 Heap 选项卡旁边且有两个按钮：Start Tracking 和 Get Allocations。单击 Start Tracking 按钮即可开始跟踪分配。单击 Get Allocations 按钮可以在用户分配视图中加载获取到的数据。在跟踪器运行时，开始按钮会变成停止按钮。可以在任意时刻单击 Stop Tracking 来结束跟踪器。

获取之后，视图将会显示顺序、大小、类、线程 ID 以及每个分配对应的类和方法。列表最初是按降序排列的，但可以单击列头来修改排列顺序。重复单击列头可以在升序和降序之间切换。单击视图中的任意条目将会加载分配发生处的栈轨迹。这个示例再次使用 Gradle Weather App，可以上下滚动列表。App 将会加载用于不同天气的图标，而我們在此时跟踪分配。图 14-4 展示了结果。

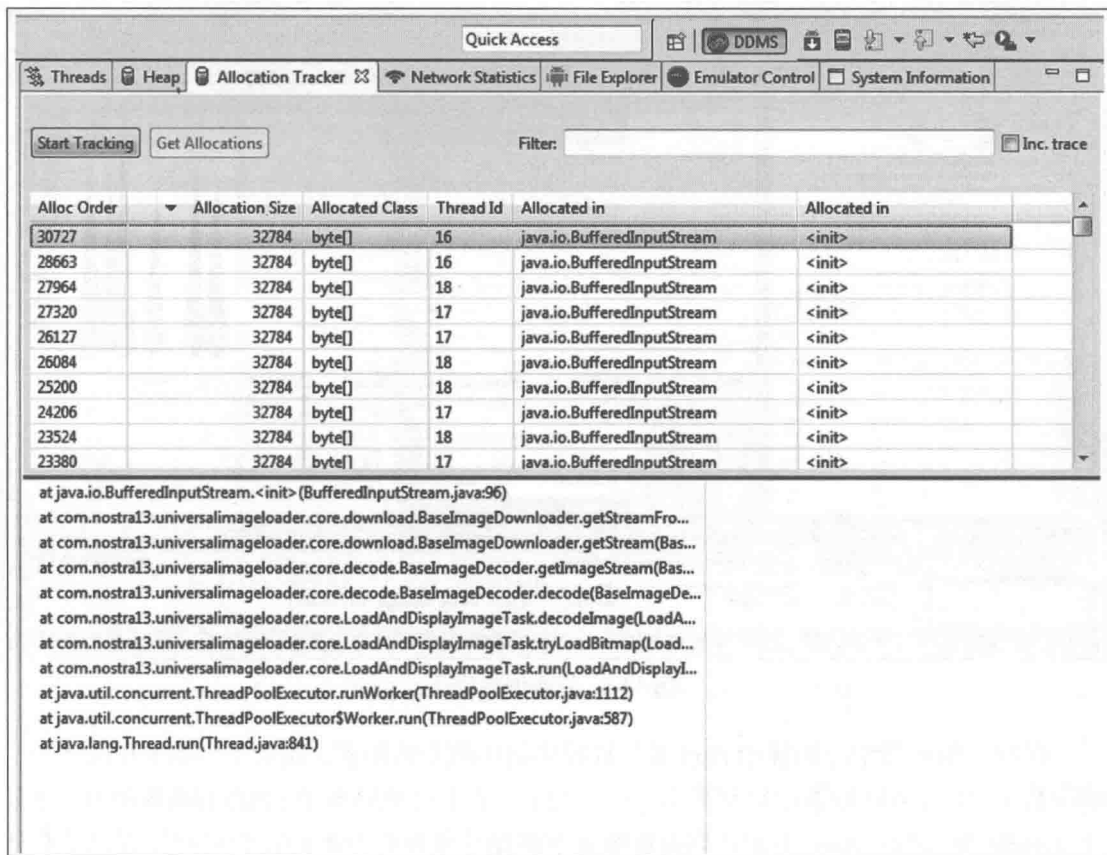


图 14-4 在 Gradle Weather 中跟踪分配的结果

当从网络下载图标数据时，你会看到一些 32KB byte 数组的分配。如果你的 App 遇到内存不足的问题，那么这是一个可能的优化目标。很重要的一点是要知道除非遇到了内存不足的问题，否则不要优化代码。过早地优化代码会导致不必要的复杂性，而且可能会与优化的目标背道而驰。

14.1.4 网络统计

Network Statistics 选项卡拥有监控网络流量的能力。这个工具和其他的工具一样易用。图 14-5 展示了该选项卡在开始网络统计抓取之前的样子。单击 Network Statistics 选项卡上的 Start 按钮即可开始抓取网络流量。Start 按钮变为 Stop 按钮，单击它可以结束抓取。

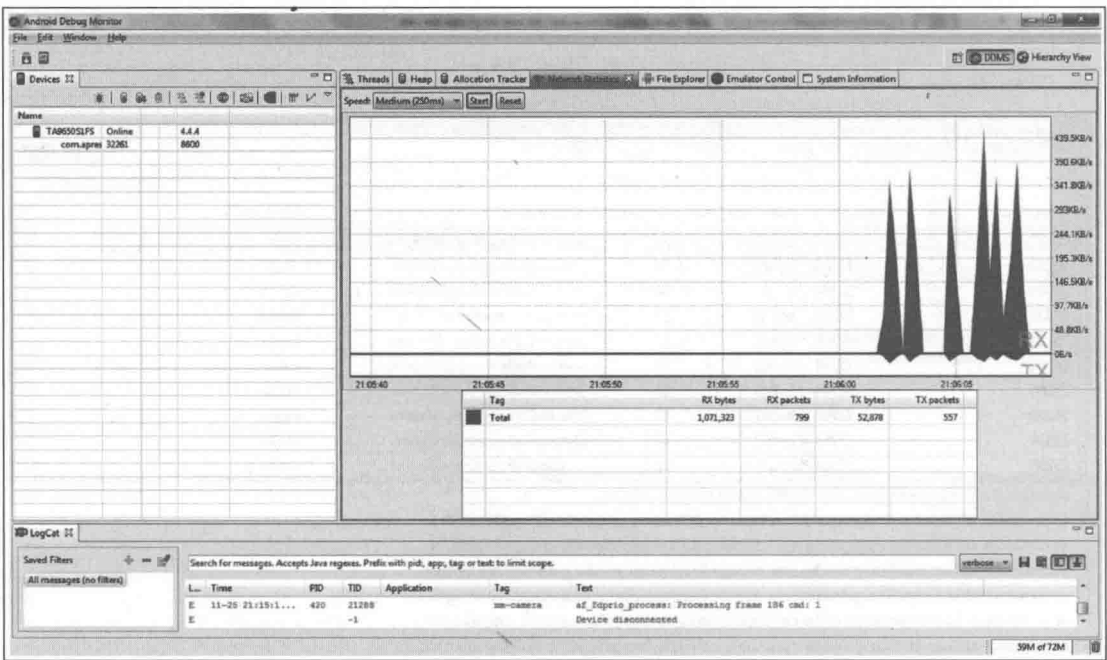


图 14-5 跟踪网络统计

视图中将会显示一幅描绘 App 运行过程中进出流量的图谱。图表上方的 RX 区域表示响应数据，而 TX 区域表示转移数据。在示例中，我们已经抓取了 1MB 的响应数据，这是在 Gradle Weather App 中滚动列表视图并下载图片数据时发生的。设备已经发出了总计 52KB 的请求数据。

14.1.5 层次查看器

你经常会遇到布局渲染不正确的情况。你的 Activity 中可能有基于用户交互摆放视图或设置可见性的逻辑。当情况变得复杂时，在 ADM 中展示视图层次会很有帮助。视图层次展示是一种可交互的屏幕快照。单击屏幕快照中的元素会在屏幕快照右侧的面板中出现一个分解视图。分解视图以树型结构给出，其中的节点表示 ViewGroup 对象。图 14-6 展示了 Gradle Weather App 的层次结构。可以单击这些节点并查看它们的布局属性。也可以展开任意节点并查看它的子对象。

单击右侧面板中的某个节点将会定位屏幕快照中对应的视图对象，同时它在它周围显示一个红色方框。所有选中节点的属性均会展现在树状视图下方的面板中。这些属性标识了视图的可见、拥有焦点、可单击、已选中以及更多状态。也可以检查视图边界、资源 ID 和内容描述。如果遇到本应可见的视图没有显示出来的情况，可以选中包含它的 ViewGroup 布局并在其中查找视图。

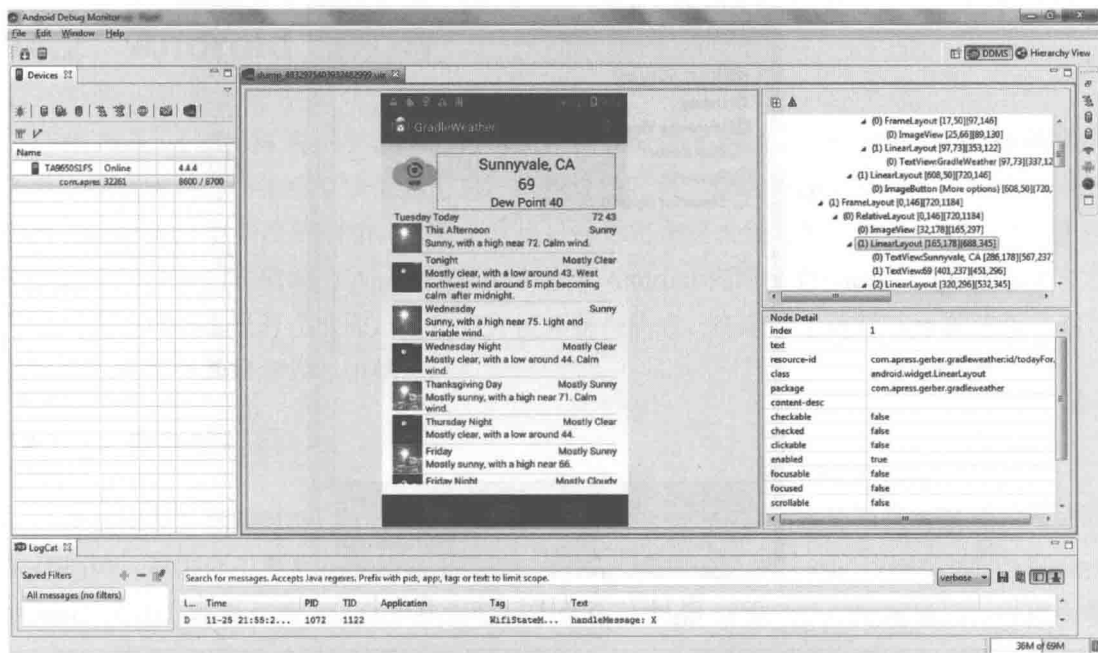


图 14-6 使用层次查看器研究 Gradle Weather 的用户界面

对于视图的一种常见误解是常量属性 `View.INVISIBLE` 和 `View.GONE` 之间的差别。标记为 `View.GONE` 的视图将不会出现在层次结构中。标记为 `View.INVISIBLE` 的视图将会在层次结构中出现但不会被绘制到屏幕上。要知道即使视图不可见，使用 `wrap_content` 属性的 `ViewGroup` 布局或容器也会为它们保留空间。如果将视图标记为 `View.GONE`，那么容器将不再保留空间并缩小至能够容纳其余所有内容的大小。

注意

Android 设备监视器基于 Eclipse 工具，让你能够通过切换视角来调整用户界面。如果不熟悉 Eclipse，那么你要知道视角代表一种特殊的工作流，其中的选项卡和视图均以最适合该工作流的方式摆放。Eclipse 工具通常有多个预先配置的视角，同时允许你自行创建。虽然 ADM 中的大量工具已经嵌入到 Android Studio IDE 中，但本节只涵盖 ADM 专有工具的一个子集。

单击 **Window | Open Perspective**，查看监视器中可用的工作流，如图 14-7 所示。

单击 **Hierarchy View** 选项，打开 **Hierarchy View** 视角。**Hierarchy Viewer** 与层次展现工具不同，后者只能在模拟器或已经 root 的设备上工作。要使用 **Hierarchy Viewer**，请启动模拟器并在模拟器中打开你的 App。单击 **Refresh** 按钮，然后在 **Windows** 选项卡的设备列表中找到你的模拟器。你的屏幕应该会像图 14-8 所示的样子。在设备列表中找到 App 对应的进程，单击 **Load** 按钮将 App 当前屏幕上的视图层次加载进来。层次结构视图提供了一个关于当前屏幕上已渲染布局的庞大且深层次的树型视图。

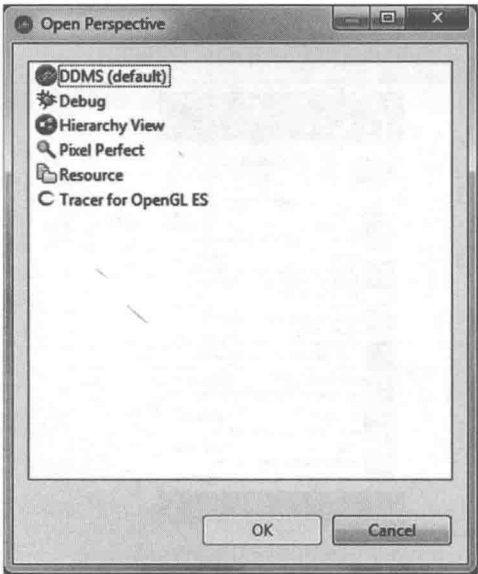


图 14-7 在 ADM 中切换视角

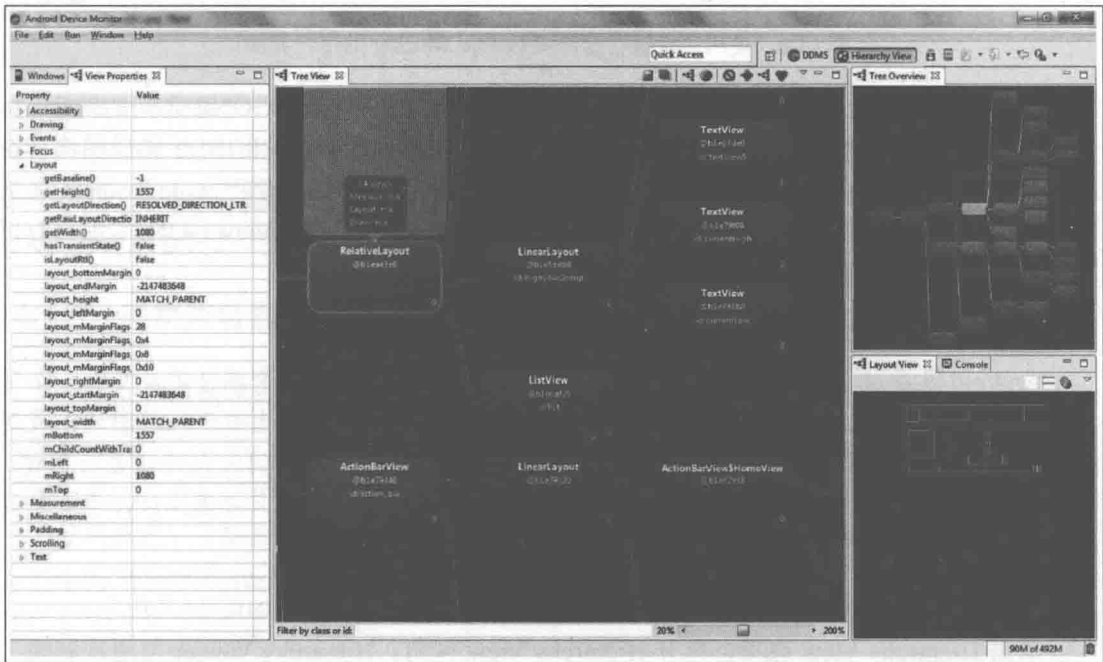


图 14-8 使用层次查看器研究 Gradle Weather UI

ADM 窗口左侧的 View Properties 选项卡中包含详尽的属性列表,而中间面板显示放大的了层次视图。可以在窗口的右下方找到 Layout View 选项卡,它以类似线框图的形式展现当前屏幕。单击任意一个选项卡中的元素会在其他选项卡中选中对等元素,因为它们都是保持同步的。

14.2 Android 监视器

在 Android DDMS 视图中, Android Studio 在 IDE 的底部加入了一些来自 ADM 的常用工具。这些工具能够生成系统信息汇总、执行垃圾收集、结束 App、分析堆以及执行方法跟踪。随着 App 复杂度的增加, 这些工具可以给你带来极大的帮助。在 Android DDMS 视图中, 选择进程列表里面的 App。进程列表位于 Android 视图中 Devices | Logcat 选项卡之下。如果此选项卡没有在前面, 单击它并使其获得焦点。在选择了运行 App 的进程之后, 其余的工具按钮都将变为可用状态。

14.2.1 内存监视器

内存监视器显示当前被调试 App 所消耗内存的图谱, 可以用于简单地查看内存使用的大体趋势。单击位于屏幕右下角且位于 Event Log 和 Gradle Console 按钮旁边的 Memory Monitor 按钮, 将会打开 Monitor 工具窗口。拿你的 App 做实验并在监视器运行过程中观察图像。在图 14-9 中, 我们运行 Gradle Weather App, 同时滚动预报列表并查看对内存的影响。还可以在任意时间点使用 Initiate GC 按钮来触发垃圾收集并查看内存回收的数量。如果图中显示的内存占用没有在启动垃圾收集器之后返回到一个合理的级别, 那么你的 App 可能存在内存泄漏。

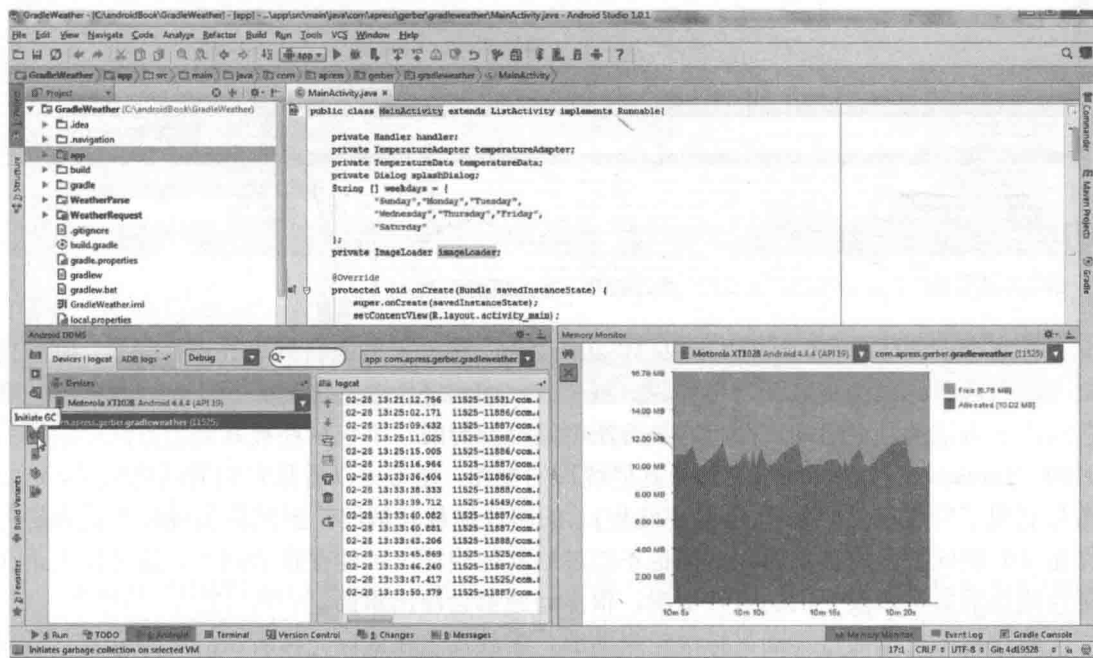


图 14-9 滚动列表时, Gradle Weather 的内存占用情况

14.2.2 方法跟踪工具

Method Trace 工具可以帮助你找出那些执行时会消耗大量 CPU 时钟的方法。CPU 时钟是一种宝贵的资源，应该珍惜它们。当一个或更多方法过度占用 CPU 时，应用就会变慢。如果 App 遇到了变慢的问题，或者只是想要更好地了解典型用例中的 CPU 使用方法，那么可以使用 Method Trace 工具来记录任意特定场景下使用 App 时的活动情况。

Method Trace 工具很容易使用。运行你的 App 或者让其进入想要研究的状态。从进程列表中选择了你的 App 之后，单击 Start Method Trace 图标即可开始跟踪。使用你感兴趣的方法做练习，接着再次单击按钮即可完成方法跟踪。在图 14-10 中，我们抓取了当滚动列表时 Gradle Weather 的活动情况。

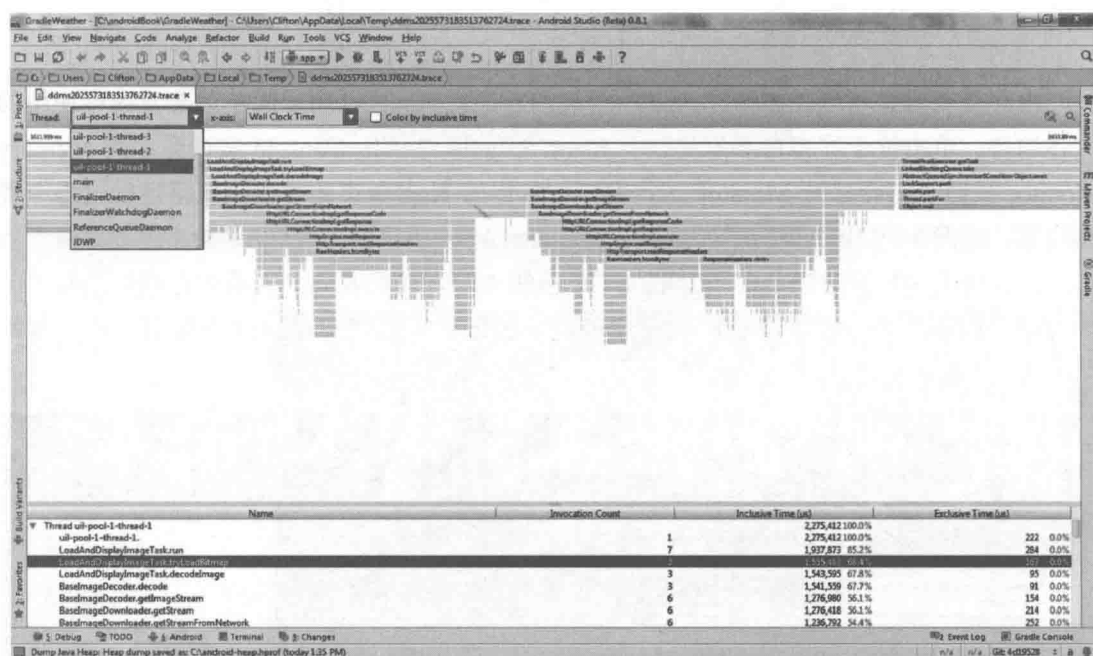


图 14-10 方法跟踪工具

在这个示例中，我们运行 Gradle Weather App、滑动天气条目列表并进行记录。刚刚完成方法跟踪时，视图将默认为主线程。每个方法调用采用条形可视化表示。这些条形的颜色取决于方法的独占时间，即仅在该方法中消耗的时间，不包括在被其调用方法中消耗的时间。Threads 下拉框可以用于切换其他线程的视图，使得可以看见它们的活动情况。这幅图片展现了发生在后台线程(不是主线程)上的图片加载和解码。虽然在主线程上做大量工作是 UI 响应缓慢的常见原因，但也不能忽略其他线程上可能存在的问题。通过检查正在运行线程的数量以及它们的执行情况，很多问题都会浮出水面。

跟踪视图可以通过滚动鼠标滚轮来缩放。缩放发生在鼠标光标所在的屏幕位置。适应这个跟踪视图可能需要一点时间，因为你可能习惯于典型的左/右滚动行为，而这在此查看器中是没有的。要查看某个位图加载方法调用的细节，可以在查看器中找到它并使用鼠标指向它。接着向下滚动鼠标滚轮将其放大，根据需要获取尽可能多的可视化细节。随着放

大的进行，查看器中会包含更多细节，同时显示并标记出栈中更底层的方法调用。随后，要查看之前发生的方法调用，可以向上滚动鼠标滚轮来缩小视图，以便查看更多的跟踪信息。接下来，可以指向更早的方法调用并重复此过程。

可视化查看器下方的表格展现了所有方法调用的详细信息。这些详细信息包括名称、调用次数以及非独占和独占时间。所有这些时间均相对于记录跟踪所消耗的时间。如果记录跟踪花费了 4 秒钟，那么读数 50% 就表示两秒钟。可以将鼠标滑动到查看器中的任意方法上，等待两秒钟，在出现的工具提示中会给出以毫秒为单位的精确时间。

14.2.3 分配跟踪器

分配跟踪器现已内置到了 Android Studio 中，它的工作方式类似于 ADM 的分配跟踪器。单击位于工具栏左侧、Android DDMS 工具窗口下方的内存跟踪即可开始跟踪分配。运行 App 并与之交互，接着再次单击按钮停止跟踪分配。编辑器中打开的新选项卡显示了跟踪的结果，如图 14-11 所示。

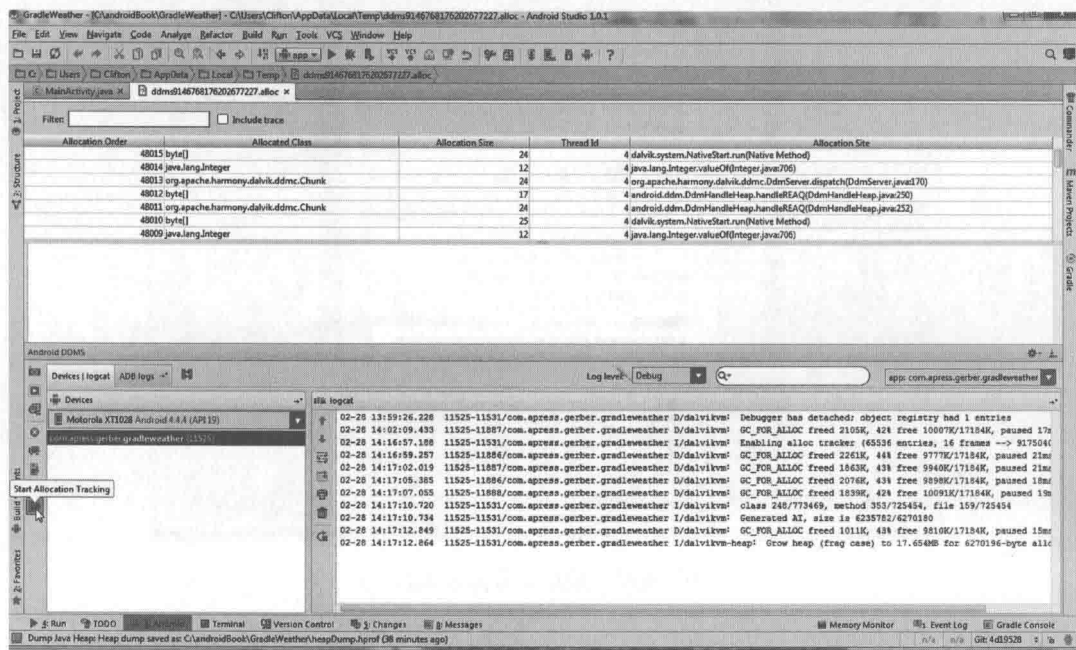


图 14-11 内置分配跟踪器

14.2.4 屏幕抓取

Android DDMS 窗口中包含多种选项，允许你在使用过程中抓取屏幕。Screen Capture 按钮会立即抓取设备的当前屏幕并将图像加载到预览对话框中，可以在此处选择将其保存到硬盘上，图 14-12 展示了此对话框。屏幕快照对话框还允许你使用手机或平板电脑上的设计为图片加上边框。缩放控件可以用于放大或缩小屏幕。可以在保存之前应用阴影、屏幕眩光，甚至旋转图片。单击 Reload 按钮，刷新这个显示着当前屏幕内容图片

的对话框。

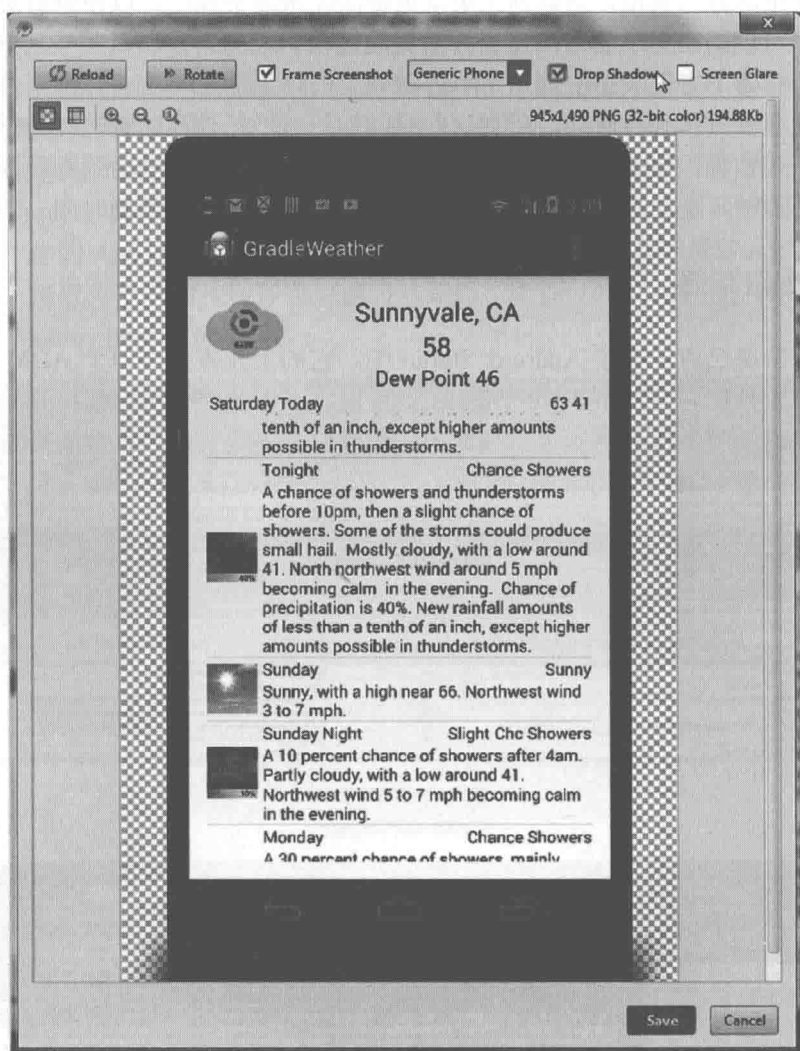


图 14-12 使用屏幕抓取工具

Screen Record 按钮允许你在与 App 交互的过程中将屏幕显示录制成视频。当单击这个按钮时，你会看到图 14-13 中所示的对话框，提示选择录制的比特率和分辨率。单击 Start Recording 按钮即可开始录制，接着使用你的 App。当完成时，单击 Stop Recording，生成已录制交互操作的视频文件。另一个对话框会提示你保存记录。使用任意文件名并将其保存到系统中容易找到的位置。Windows 用户可能需要安装相应的编解码器或软件，因为文件的保存格式为 MP4。图 14-14 使用 Windows 上流行的 VLC 播放器展示了与 Gradle Weather App 交互的回放。

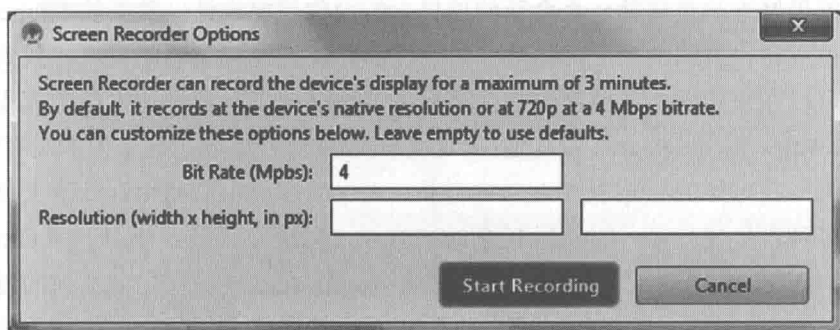


图 14-13 启动 Screen Recorder 工具

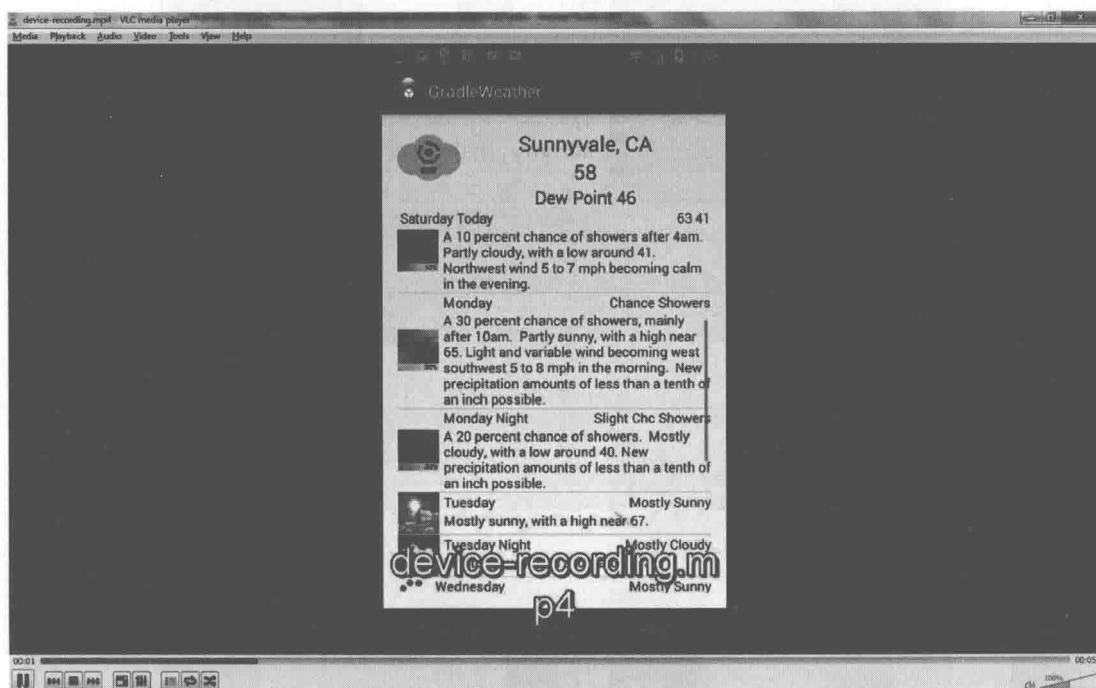


图 14-14 屏幕录像的回放

14.3 导航编辑器

导航编辑器(Navigation Editor)是 Android Studio 的一项全新特性。虽然在编写本书时已经可用,但它仍处于研发阶段。这个编辑器允许你快速设计 App 的高层工作流原型,同时在编辑模式中导航特定的 Activity 和 Fragment。如果有了 App 的大体思路并且想要预览用户在不同界面之间的切换情况,那么 Navigation Editor 是一款理想的工具。它还可以发现已有 App 界面之间的流转和连接。随着时间的推移,看到这款工具成熟起来会是一件令人激动的事情。

熟悉它的最好方法是借助一个全新的项目。假设你想要设计一款新的 Shopping App，它允许用户使用自己喜欢的社交网络账号快速注册并随意地浏览商品列表。找到某款商品之后，用户可以在决定购买之前单击它并获取更多详细信息。要设计这样的流程，可以使用 napkin sketches、whiteboards 或其他工具(仅与 IDE 部分集成，甚至根本没有集成)。将大体思路变为一款可用的 App 是个艰难的过程，工作的过程中需要用到外部工具和额外的技巧来管理多种设计器程序。在使用 IDE 的同时，人们通常还需要使用诸如 OmniGraffle、Lucidchart 等线框图或图表工具。为了实现可用 App 而在这些程序之间切换的过程并不是很容易。Navigation Editor 为你提供了一种在 IDE 中轻松建立原型和工作流的方法。在本节中，将使用这个工具来探索我们的 Shopping App。

14.3.1 设计用户界面

使用 New Project Wizard 和 Blank Activity 模板，创建一个名为 Navigate 的项目。在项目加载之后，应该首先在设计模式中编辑 activity_main.xml 布局。删除“Hello World”标签并拖曳出 Large Text Label，在它下面放置三个按钮。将标签上的文本修改为“Mini-Shopper”，并将按钮上的文本修改为三个示例社交网络服务的名称。图 14-15 中的示例使用 FaceBox、Twiggler 和 G++，你还可以尽情地创造。

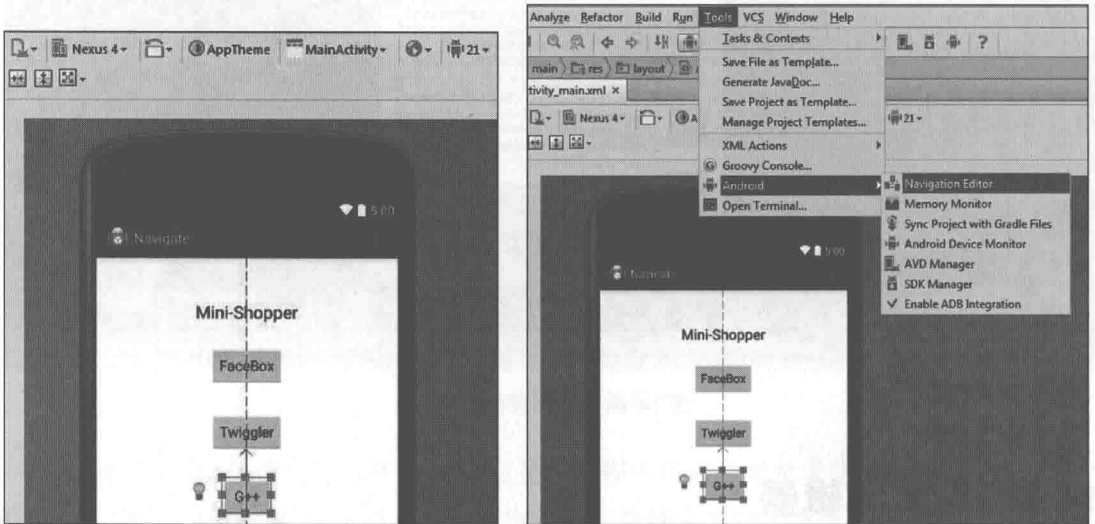


图 14-15 设计 FaceBox 用户界面

14.3.2 导航编辑器初步

接下来，单击主菜单上的 Tools Android | Navigation Editor，屏幕将如图 14-16 所示。

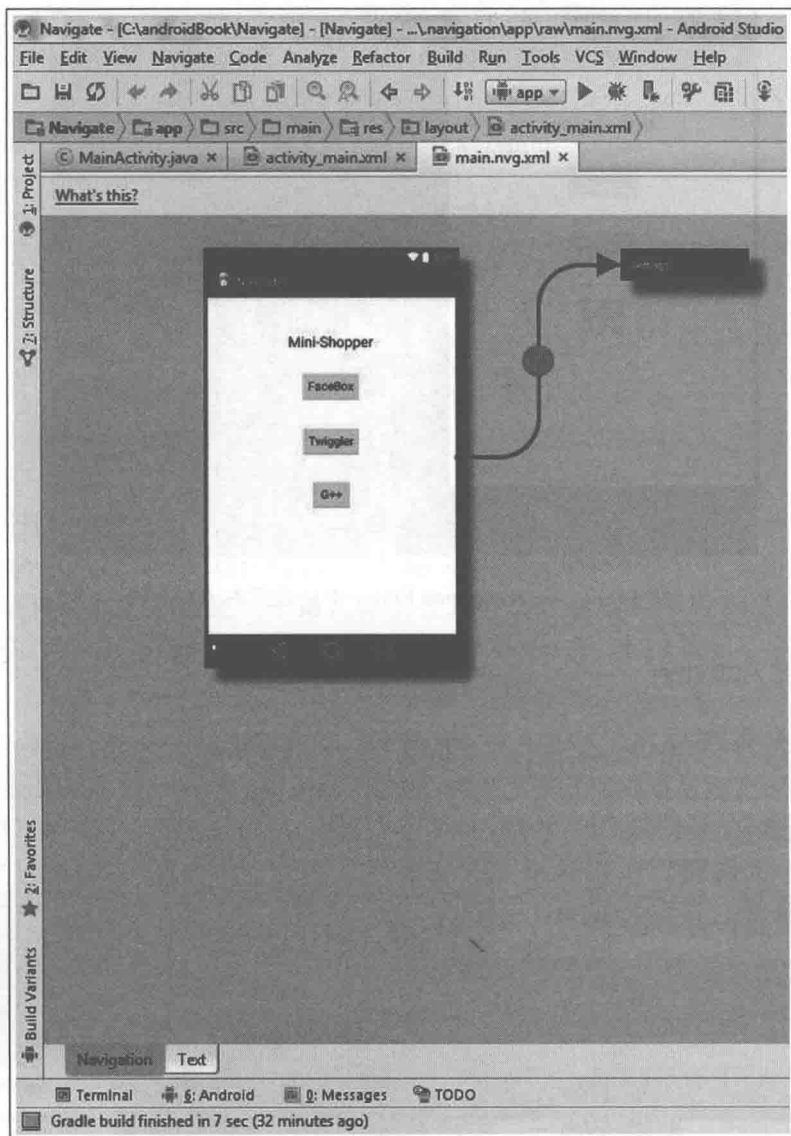


图 14-16 打开 Navigation Editor

Android Studio 将会创建 `main.nvg.xml` 文件并在 Navigation 编辑器中显示它。它将以可视化的方式展现 Activity 以及与之相关联的 Android 上下文菜单(空白 Activity 模板自动创建此上下文菜单)。这个编辑器允许你快速创建新的 Activity 并通过将它们与已有 Activity 上的控件关联来创建跳转。它也允许你建立与 Android 系统上下文菜单项的连接。可以在编辑器中单击并来回拖曳条目，例如上下文菜单。

在编辑器中的任意位置右击，打开编辑器上下文菜单，其中有一个 **New Activity** 选项，如图 14-17 所示。单击这个选项，打开 **New Activity Wizard**。选择 **Blank Activity** 模板并将这个新 Activity 命名为 `FaceBoxLoginActivity`。你将返回导航视图，它现在同时显示两个 Activity。

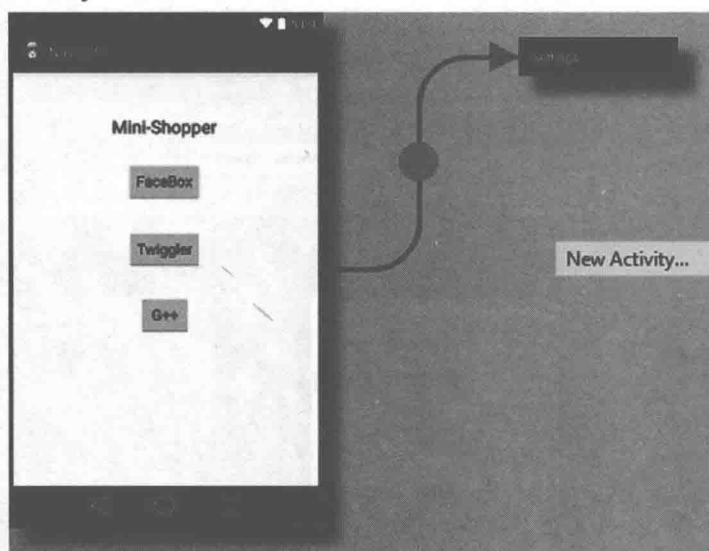


图 14-17 在 Navigation Editor 中创建新的 Activity

14.3.3 连接 Activity

重新放置新的 Activity，使其与原来的相邻。需要在它们之间建立连接。在编辑器中工作时，可以随意地重新放置上下文菜单。按住 Shift 键的同时单击 FaceBox 按钮并将其拖到新的 FaceBoxLoginActivity 之上。编辑器将会在它们之间绘制一条连接线，粉色的点表示这条线中的跳转。单击这个点来查看跳转的定义。这个跳转通过单击手势将 MainActivity(源端)连接到 FaceBoxLoginActivity(目的端)，如图 14-18 所示。

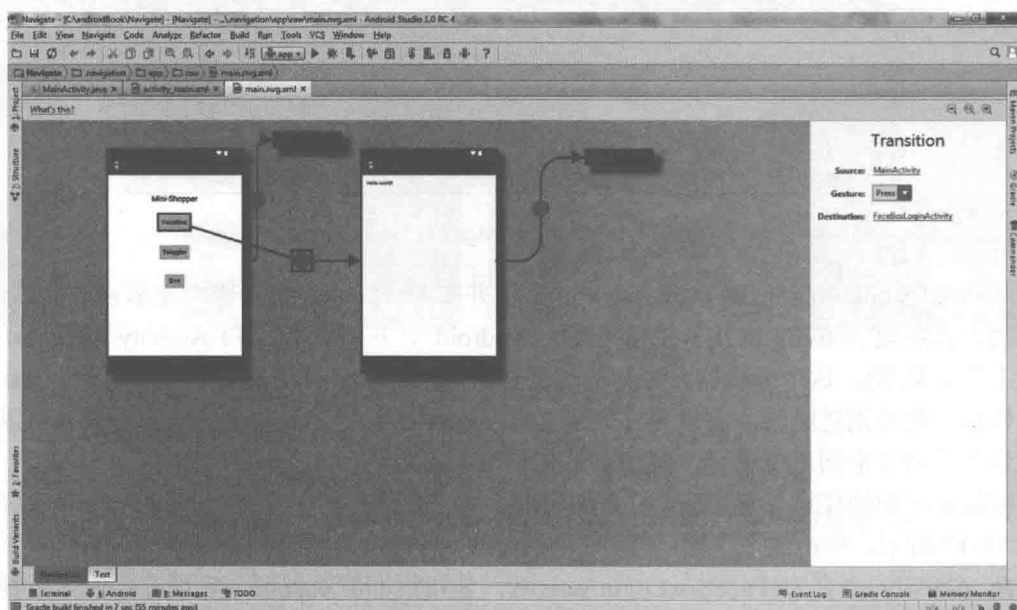


图 14-18 使用 Navigation Editor 连接 Activity

现在打开 MainActivity.java 源文件, 你将会看到一个绑定到按钮的单击监听器, 它启动了 FaceBoxLoginActivity:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            MainActivity.this.startActivity(new Intent(MainActivity.this,
                FaceBoxLoginActivity.class));
        }
    });
}
```

这段代码是由编辑器中简单的单击和拖曳操作生成的。返回 Navigation Editor 并双击 FaceBoxLoginActivity, 你将会看到这个 Activity 的图形化编辑视图, 可以在这里拖放更多的控件和选项来装饰它。创建一个最简单的登录界面, 包括用于用户名和密码的两个 TextView 标签、两个 EditText 输入框以及一个 Login 按钮。图 14-19 展示了虚拟的 FaceBox 登录界面。

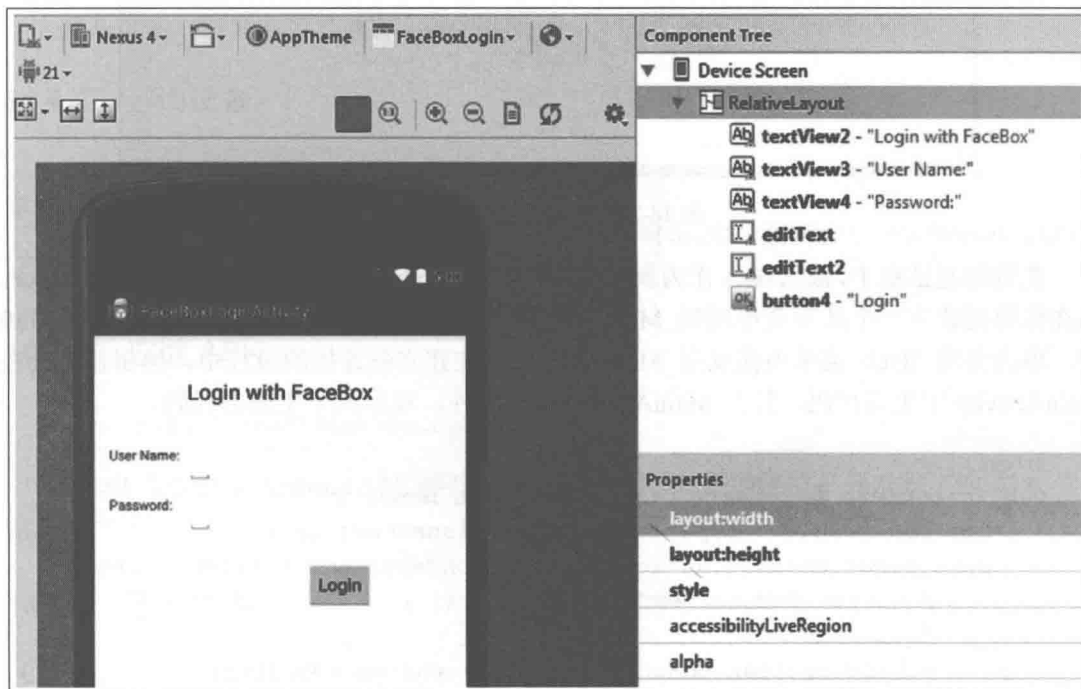


图 14-19 设计 FaceBox 登录界面

14.3.4 编辑菜单

返回到 Navigation Editor 中, 它现在将会体现出 FaceBoxLogin 布局中所做的修改: 可

以运行 App 来测试跳转和新 FaceBoxLogin 布局的变化。在 Navigation Editor 中，双击与登录 Activity 相关联的上下文菜单。menu_facebox_login.xml 文件将会打开，实时预览窗口位于其右侧。修改唯一的菜单项，将其 ID 设置为 @+id/action_back，将标题设置为 @string/action_back。按 Alt + Enter 键调用 Intention 对话框，建议操作为创建新字符串值资源，如图 14-20 所示。按 Enter 键完成此操作。

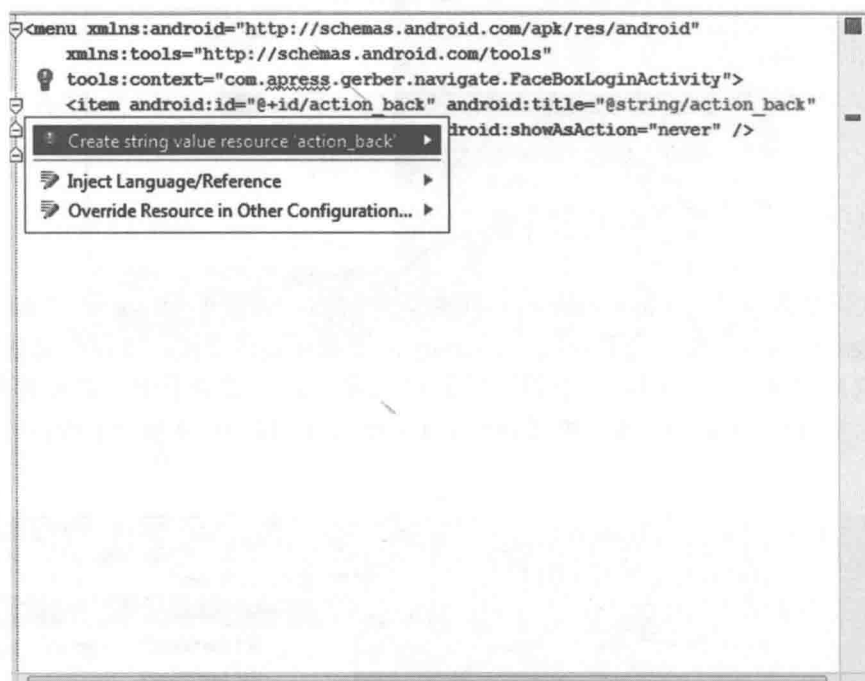


图 14-20 编辑 FaceBox 菜单

在资源对话框中，输入 back 作为新字符串的值，按 Enter 键继续。返回 Navigation Editor。现在你将要建立一个从新菜单项到 MainActivity 的连接。与之前的方法相同，按住 Shift 键，单击并将 Back 菜单项拖曳至 MainActivity。在建立新连接的过程中，编辑器将会在 MainActivity 中生成代码。打开 MainActivity.java 文件，观察以下生成的代码：

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    boolean result = super.onPrepareOptionsMenu(menu);
    menu.findItem(R.id.action_back).setOnMenuItemClickListener(new
        MenuItem.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem menuItem) {
                FaceBoxLoginActivity.this.startActivity(new
                    Intent(FaceBoxLoginActivity.this,
                        MainActivity.class));
                return true;
            }
        });
}
```

```

    return result;
}

```

建立完这些连接之后，构建并运行 App，测试跳转的工作情况。此时，你应该能够从 MainActivity 跳转到 FaceBoxLoginActivity，接着使用新的上下文菜单项返回 MainActivity。

既然已经熟悉了 Navigation Editor 的基本使用方法，现在就尝试再为 App 创建两个 Activity：一个用于展现商品列表；另一个用于查看商品详情。

14.4 终端

工具箱中最实用的插件或许就是 Terminal。单击 IDE 底部的终端选项卡，打开终端窗口，可以在这里输入操作系统命令。可以单击绿色的加号按钮，在单独的选项卡中开启新会话。当无法找到或记住 IDE 操作时，命令窗口可以帮助你完成任务。在终端，需要了解的最重要工具应该就是 ADB——Android 调试桥(Android Debug Bridge)。此工具能够让你直接控制已连接的设备或模拟器。命令采用的形式为 `adb {device-options} sub-command {sub-command-options}`。设备选项如下：`-d` 表示仅面向已绑定的设备，`-e` 表示仅面向已绑定的模拟器，而 `-s deviceID` 表示给定 ID 的特定设备。

打开终端，研究本节其余部分介绍的命令。

14.4.1 查询设备

```
adb devices
```

`devices` 子命令会列出每台已绑定设备的名称和设备 ID。它将会以 `emulator-<端口号>` 的格式列出带有设备 ID 的模拟器。

14.4.2 安装 APK

```
adb install /path/to/app.apk
```

`install` 命令会将 Android APK 推送到设备并安装，只需提供位于开发机上的 APK 文件路径。

14.4.3 下载文件

```
adb pull /path/to/device/file.ext /path/to/local/destination/
```

`pull` 命令可以将任意文件从设备上下载到你的开发机。

14.4.4 上传文件

```
adb push /path/to/local/file.ext /path/to/device/destination/
```

push 命令可以将任意文件从开发机上传到设备。

14.4.5 端口转发

```
adb forward local-port remote-port
```

forward 命令会把开发机的网络连接重定向至设备。这种技术会在一些高级场景中使用，例如在 Chrome 网页浏览器中调试代码或者连接到运行在设备上的网络服务器。

14.5 Google 云工具

前面研究了一款 Android App，它使用一个通过网络提供天气预报的服务。在本节中，将探索如何使用 Google 云工具开发和部署自己的后台。首先，你将要设计前端，它使用一个 Java Bean 类来构造通信内容。然后，你将构建后台并在本地运行它。最后，要把它发布到 Google 云服务并端到端地测试项目。下面使用 Google 账户登录，如图 14-21 所示。

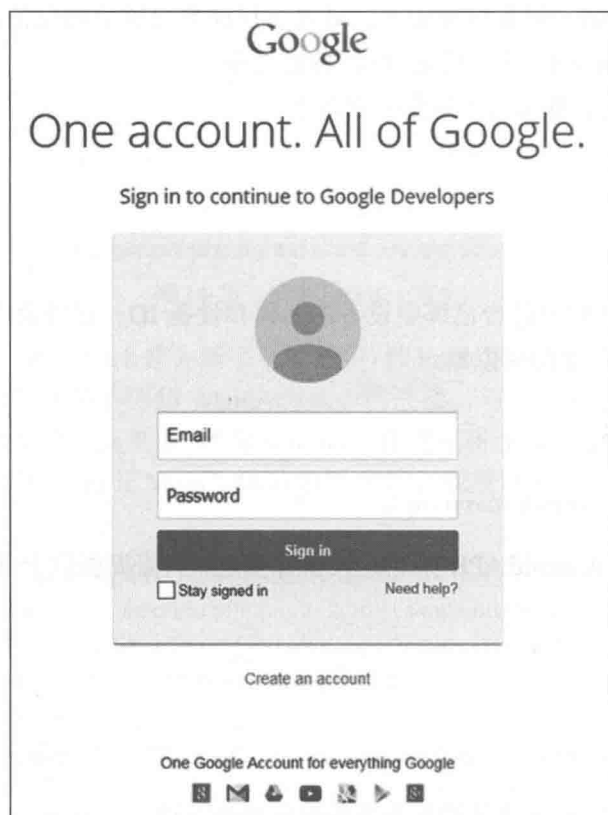


图 14-21 登录 Google

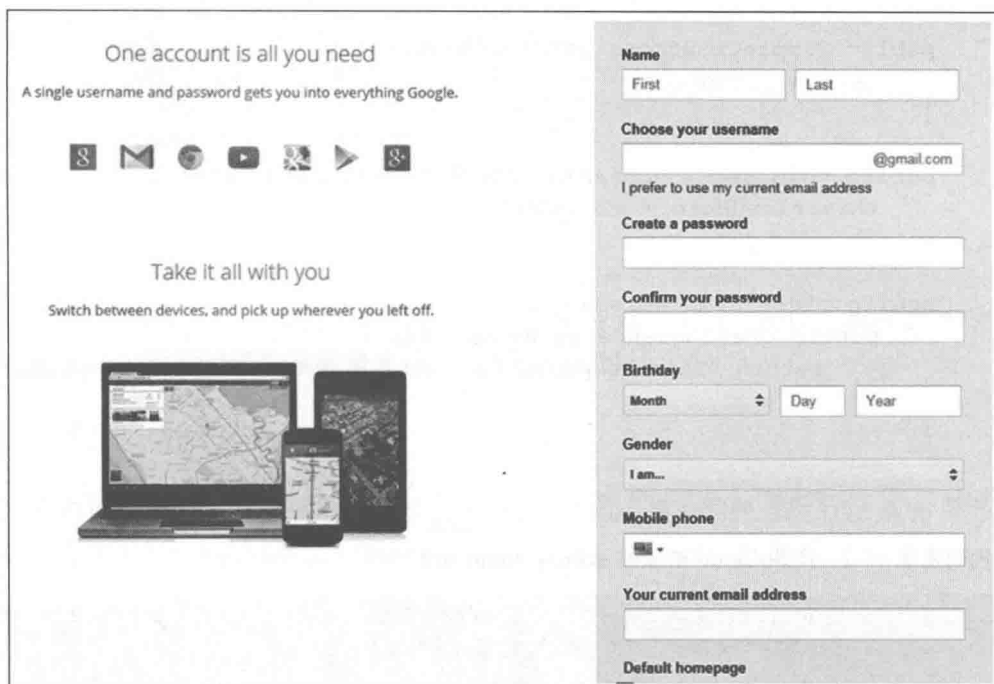


图 14-21 登录 Google(续)

14.5.1 创建 HelloCloud 前端

使用 Blank Activity 模板创建新的 Android 项目并将其命名为 HelloCloud。将这个空白 Activity 命名为 MainActivity 并单击 Finish 按钮来开启你的项目。将代码清单 14-1 中的代码用于 MainActivity，将代码清单 14-2 中的 XML 用于 activity_main.xml 布局。

代码清单 14-1 HelloCloud 前端的 MainActivity

```
public class MainActivity extends Activity {

    private SimpleCloudBean cloudBean;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setCloudBean(new SimpleCloudBean());
        setContentView(R.layout.activity_main);
    }

    public void onClick(View sender) {
        final TextView txtResponse = (TextView)
            findViewById(R.id.txtResponse);
        txtResponse.setText(getCloudBean().getResponse());
        txtResponse.setVisibility(View.VISIBLE);
    }
}
```

```

    public SimpleCloudBean getCloudBean() {
        return cloudBean;
    }

    public void setCloudBean(SimpleCloudBean cloudBean) {
        this.cloudBean = cloudBean;
    }

    public class SimpleCloudBean {
        public CharSequence getResponse() {
            return "This response is from " + getClass().getSimpleName();
        }
    }
}

```

代码清单 14-2 HelloCloud 前端的 activity_main.xml

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:text="@string/greeting_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txtGreeting" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="go!"
        android:id="@+id/button"
        android:layout_below="@+id/txtGreeting"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="42dp"
        android:layout_marginTop="72dp"
        android:onClick="onGoClick" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"

```

```

    android:text="Response Shows Here"
    android:id="@+id/txtResponse"
    android:layout_below="@+id/txtGreeting"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="34dp"
    android:visibility="invisible" />

```

```
</RelativeLayout>
```

这段代码调用一个简单的本地 Bean，它把响应返回给 Activity。响应会在隐藏的 TextView 组件中更新，然后将该组件设置为 View.Visible。

14.5.2 创建 Java 后台模块

现在可以向项目中添加一个新的后台模块。这个后台模块将包含运行在 Web 服务器上的代码。单击 File | New Module | Google Cloud Module，如图 14-22 所示。

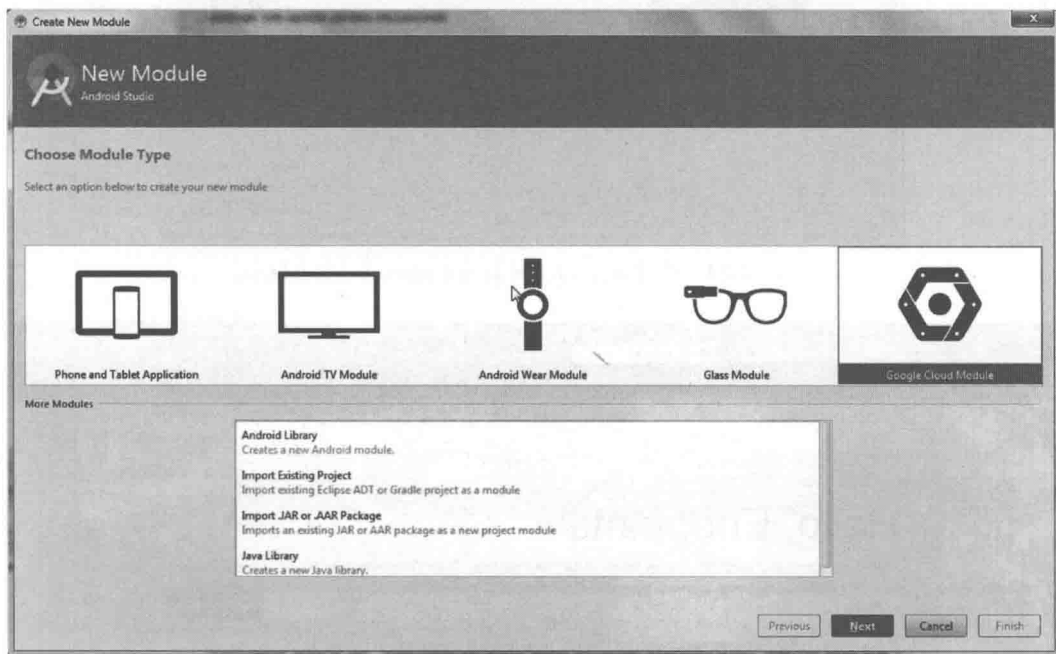


图 14-22 创建 App Engine 模块

将模块命名为 backend 并保留其他选项的默认设置，如图 14-23 所示。单击 Finish 按钮，Android Studio 将会生成一个基础的 Java servlet 项目，其中包含可用的 Google 云端点。Gradle 将会开始把这个新模块同步到你的项目中。

一旦同步完成，右击项目窗口中的后台模块并选择 Make Module back-end 选项。接下来，在运行配置列表中找到后台选项并单击 Run 按钮启动它。Android Studio 将会把 servlet 代码包装到 Jetty Web servlet 引擎的一个实例中，为了方便使用，该引擎运行在本地。控制台给出了关于如何使用 Web 浏览器与端点交互的提示。启动浏览器并打开 <http://localhost>:

8080/, 观察端点的运行情况, 你将看到 14-24 所示的页面。

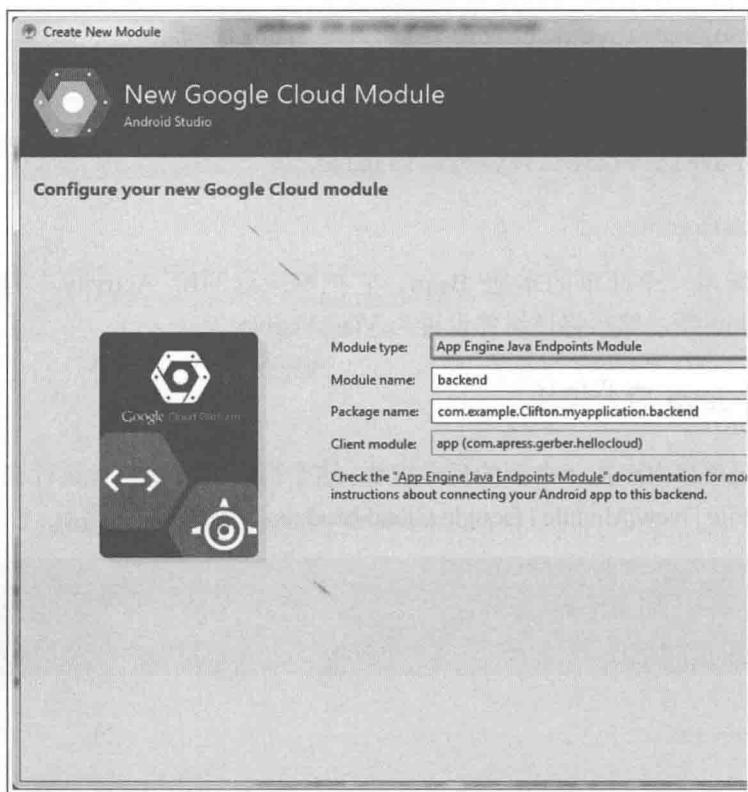


图 14-23 选择 App Engine Java Endpoints Module

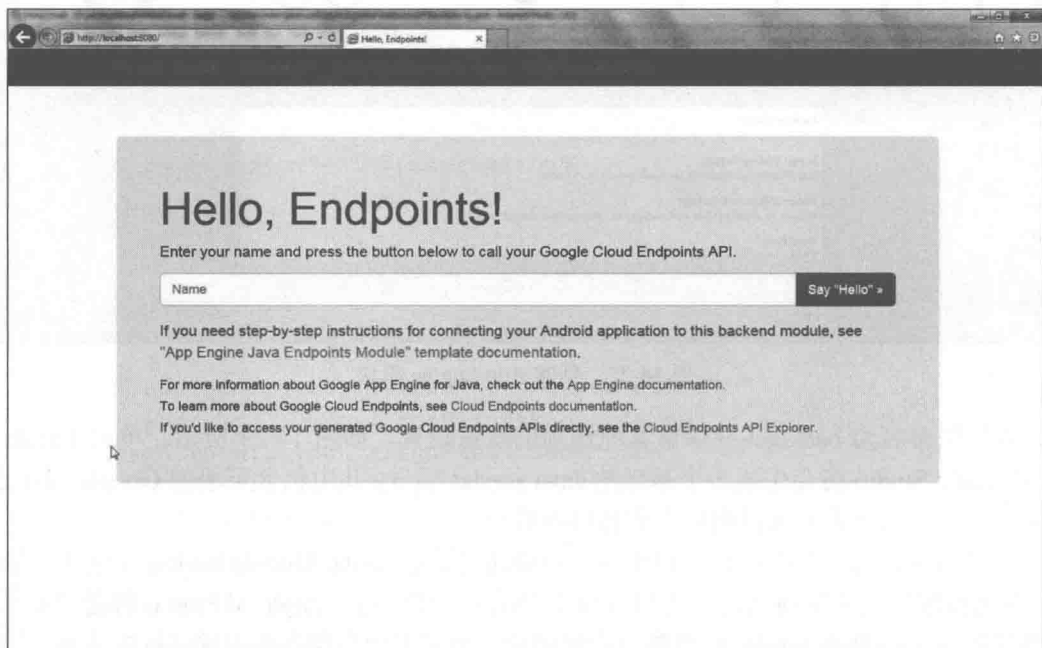


图 14-24 运行 Google 云端点

14.5.3 组合在一起

在验证端点正在运行之后,可以让 Android Studio 生成并安装能够在 Android App 中使用的客户端库。在右侧的 Gradle 构建工具窗口中找到用于后台模块的构建并运行 `appengineEndpointsInstallClientLibs` 任务,这体现在图 14-25 中。

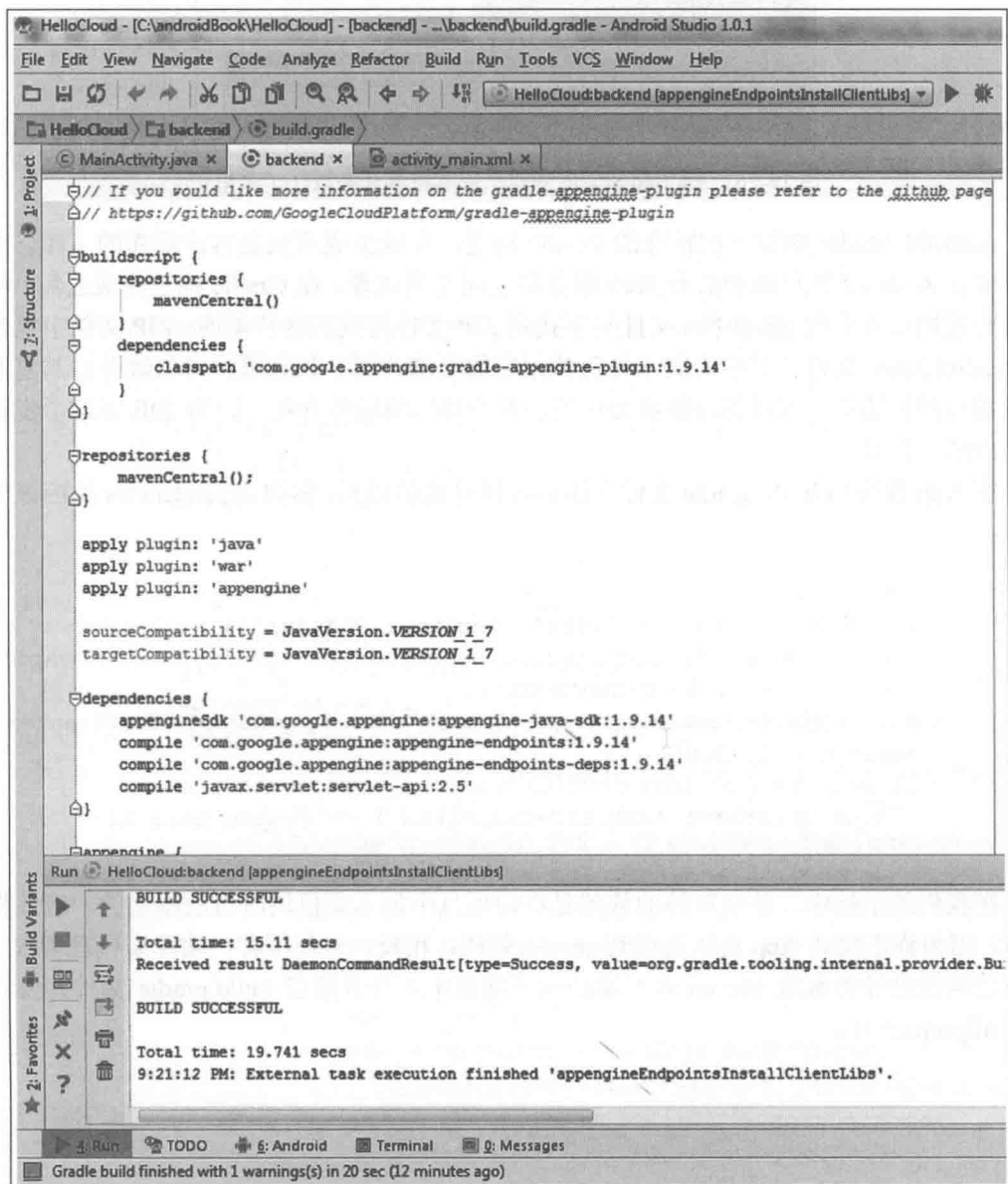


图 14-25 为端点安装客户端库

较早版本的 Android Studio 在菜单中有一个选项,从 1.0.1 版本起去掉了该选项。在 0.8.x 版本中,可以单击 `Tools | Google Cloud Tools | Install Client Libraries`。图 14-26 展示了之前的菜单。

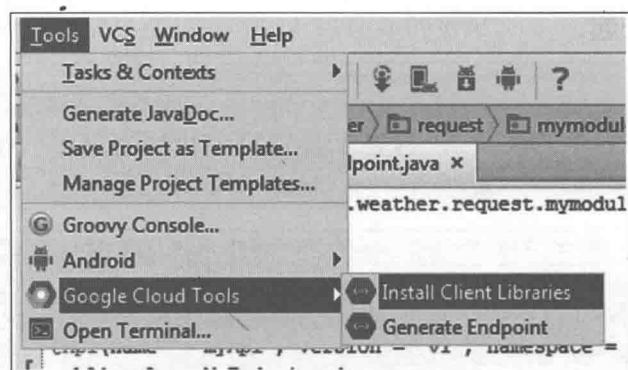


图 14-26 Android Studio 的较早版本将该任务放在了菜单中

Android Studio 触发一次特殊的 Gradle 构建，完成生成可安装客户端库的工作，而客户端库在 Android 客户端和后台 Web 服务器之间充当代理。在 Gradle 构建完成之后，可以在后台模块文件夹的 `client-libs` 文件夹下找到 ZIP 文件格式的客户端库。ZIP 文件中包含一个 `readme.html` 文件，其中有关于如何使用它的全部介绍。找到编译时依赖并将其复制到使用端点的模块中。可以忽略解释如何安装客户端库的额外介绍，因为 IDE 将这个步骤作为生成的一部分。

在 App 模块的 `build.gradle` 文件中添加编译时依赖以后，你的 `dependencies` 块应该如下所示：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile([group: 'com.apress.gerber.cloud.backend', name: 'myApi',
        version: 'v1-1.19.0-SNAPSHOT'])
    compile([group: 'com.google.api-client', name: 'google-api-client-android',
        version: '1.19.0'])
    // compile project(path: ':backend', <- remove this line
    // configuration: 'android-endpoints') <- remove this line
}
```

在我们的示例中，被注释掉的依赖是在向项目中加入新模块时自动添加的。应该将其删除，因为你不想让 App 直接连接到 `servlet` 代码；相反，它使用客户端库来代理请求。你还必须确保已经将本地 Maven 仓库添加到了项目中。打开顶层 `build.gradle` 文件并将它添加到 `allprojects` 中：

```
allprojects {
    repositories {
        jcenter()
        mavenLocal()
    }
}
```

在添加依赖和 `mavenLocal` 库之后，你应该利用 Gradle 构建同步项目，以便让 API 可用。在 App 模块中添加一个新的类并使用它，将此类命名为 `RemoteCloudBeanAsyncTask`，并让其派生自 `AsyncTask`。定义一个 `MyApi` 类型的静态变量。你会遇到导入类的提示，而

它应该在 `classpath` 中。如果没有导入它的选项，那么再次检查依赖并重新构建模块以确保正确引入了生成的客户端库。代码清单 14-3 定义了这个新类。

代码清单 14-3 RemoteCloudBeanAsyncTask 类定义

```
class RemoteCloudBeanAsyncTask extends AsyncTask<String, Void, String> {
    public static final String RESULT = "result";
    private static MyApi apiService = null;
    private final Handler handler;

    public RemoteCloudBeanAsyncTask(Handler handler) {
        this.handler = handler;
    }

    @Override
    protected String doInBackground(String... params) {
        String name = params[0];
        try {
            return getMyApi().sayHi(name).execute().getData();
        } catch (IOException e) {
            return e.getMessage();
        }
    }

    private MyApi getMyApi() {
        // Lazily initialize the API service
        if (apiService == null) {
            MyApi.Builder builder = new
                MyApi.Builder(AndroidHttp.newCompatibleTransport(),
                    new AndroidJsonFactory(), null)
                // The special 10.0.2.2 IP points to the local machine's
                // IP address in the emulator
                .setRootUrl("http://10.0.2.2:8080/_ah/api/")
                .setGoogleClientRequestInitializer(new
                    GoogleClientRequestInitializer() {
                        @Override
                        public void initialize(AbstractGoogleClientRequest<?>
                            abstractGoogleClientRequest) throws IOException {
                            abstractGoogleClientRequest.
                                setDisableGZipContent(true);
                        }
                    });

            apiService = builder.build();
        }
        return apiService;
    }
}
```

```

@Override
protected void onPostExecute(String result) {
    final Message message = new Message();
    final Bundle data = new Bundle();
    data.putString(RESULT, result);
    message.setData(data);
    handler.sendMessage(message);
}
}

```

记得使用 `AsyncTask` 很重要，因为初始化服务和完成网络调用可能会花费一些时间。此对象使用一个 `Android` 处理程序来初始化，在后续的逻辑中会用到它。我们在 `doInBackground` 方法中获取 API 的引用。返回引用的方法会延迟创建并初始化它。在获取到 API 引用之后，发出对 Web 端点的调用，随后返回调用结果。接着在 `onPostExecute` 方法中，将一条消息发送到处理程序。

通过修改 `onGoClick` 方法将 `AsyncTask` 插入 `MainActivity` 中：

```

public void onGoClick(View sender) {
    final RemoteCloudBeanAsyncTask remoteCloudBeanAsyncTask =
        new RemoteCloudBeanAsyncTask(new Handler() {
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);
                final String result =
                    msg.getData().getString(RemoteCloudBeanAsyncTask.RESULT);
                final TextView txtResponse = (TextView)
                    findViewById(R.id.txtResponse);
                txtResponse.setText(result);
                txtResponse.setVisibility(View.VISIBLE);
            }
        });
    remoteCloudBeanAsyncTask.execute("Developers");
}

```

我们这里创建了 `RemoteCloudBeanAsyncTask` 并向其提供一个处理程序，该处理将消息传递给隐藏的文本视图并让其可见。让后台服务器仍然在开发机上运行，然后在模拟器上构建并运行这个示例。单击 Go 按钮，你应该就会看到从 Google 云端点返回的消息，如图 14-27 所示。如果得到了提示超时的消息，那么再次检查服务器是否仍在运行以及是否可以通过 Web 浏览器访问。确保已经在 `manifest` 中声明了 Internet 权限。可能还需要修改或禁用任何已启用的攻击性防火墙设置。

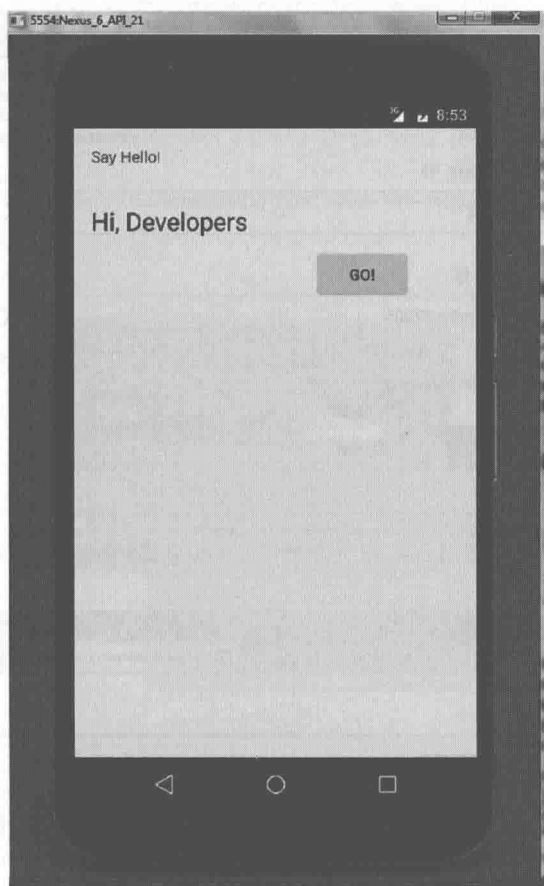


图 14-27 在模拟器上运行 App 来访问端点

14.5.4 部署到 App Engine

既然服务已经能够在本地运行并产生结果，现在就可以部署到 Google 的云服务器了。部署到云端很简单。如果后台仍在运行，那么停掉它。使用你的 Google 账号登录开发者控制台，网址为 <https://console.developers.google.com/project>。单击 Create Project 按钮，在 Google 的云服务中创建一个新的端点。为项目起一个名字，例如 MyBackend，复制生成的 Project ID 并保存到可访问的地方。参见图 14-28 中的示例。单击 Create，你将会看到图 14-29 中所示的进度条。稍等片刻，让 Google 服务完成处理工作。返回 Android Studio，找到 appengine-web.xml 文件并将你保存的 Project ID 复制到 application 标签中。此文件在 `src | main | webapp | WEB-INF` 下。

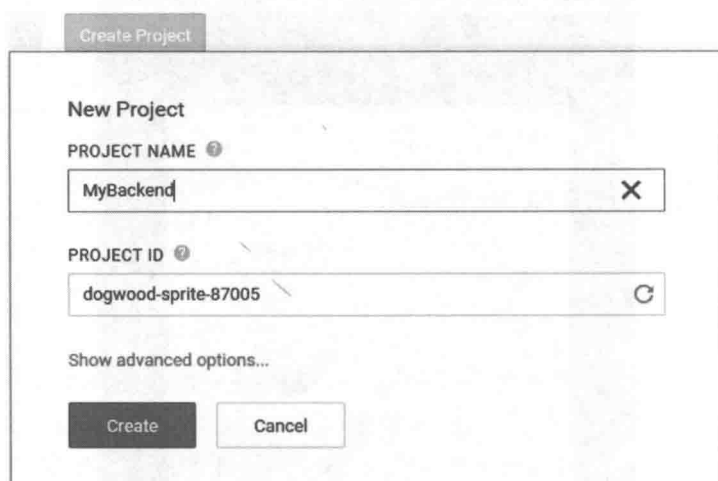


图 14-28 使用 Google 开发者控制台创建新的 Java 端点项目

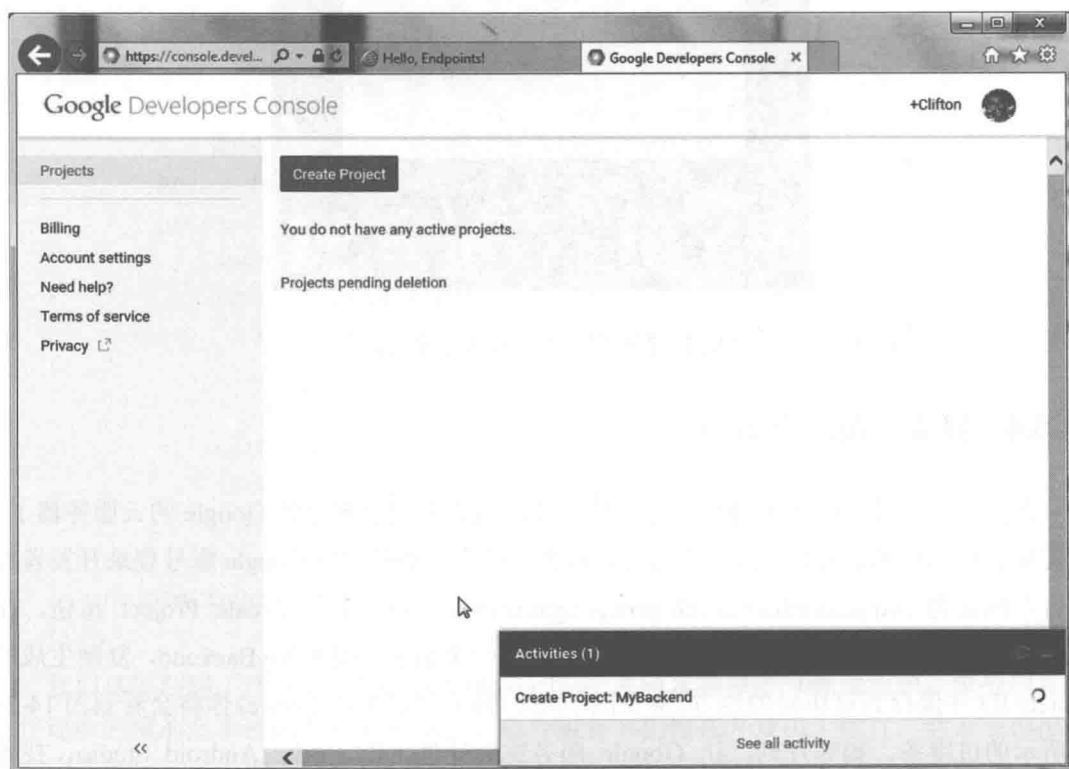


图 14-29 Google 开发者控制台在工作过程中将会给出提示

在顶部菜单中，选择 Build | Deploy Module to App Engine。单击 Deploy To 下拉框并选择你的项目 ID。如果是第一次部署，那么在做出这个选择时，你将会遇到登录到 Google 的提示。图 14-30 展示了单击 Deploy To 下拉框之后出现的登录界面。

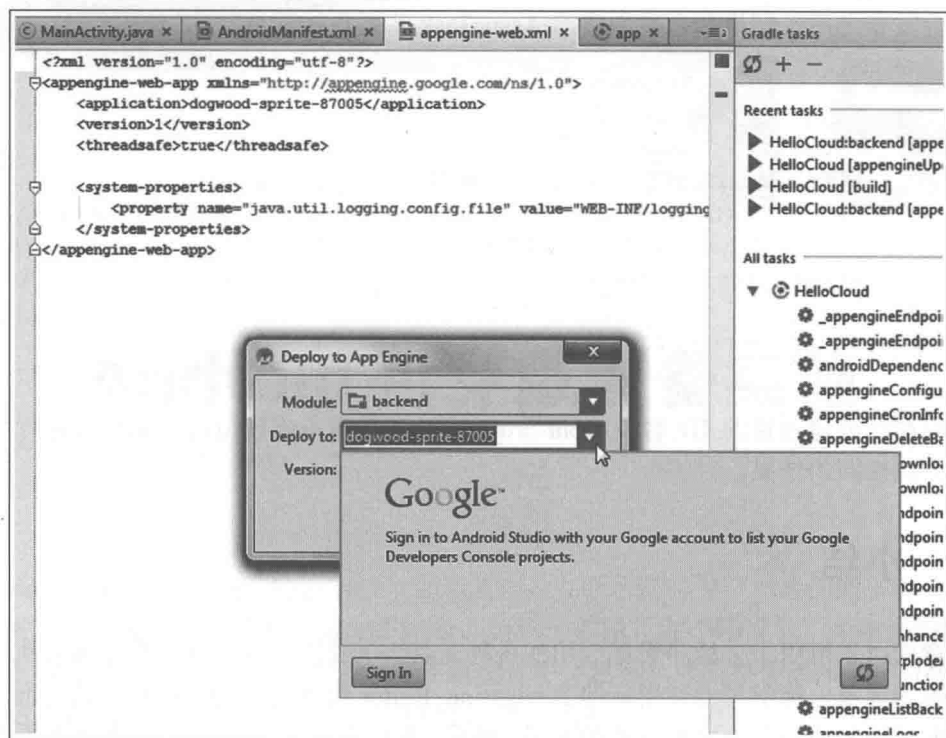


图 14-30 登录 Google 开发者控制台

登录提示打开一个浏览器窗口，如图 14-31 所示，单击 Accept 为其赋予必要的权限。

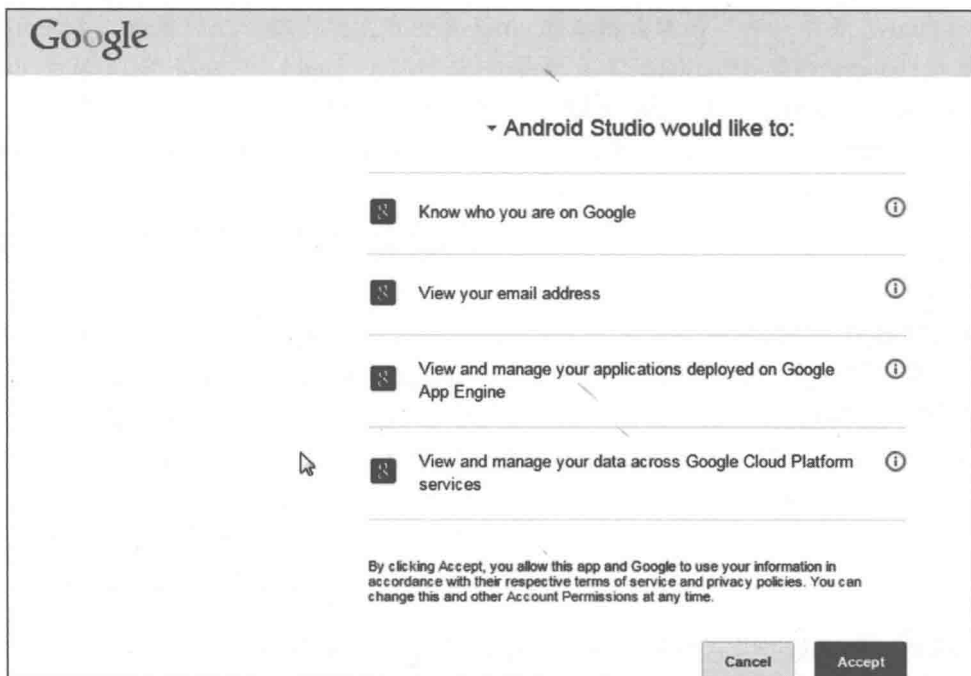


图 14-31 Google 开发者控制台的许可提示

后台发布之后，切换到之前创建的 `AsyncTask` 并更新加载 API 的方法：

```
private MyApi getMyApiRemote() {  
    //Lazily initialize the API service  
    if(apiService == null) {  
        MyApi.Builder builder = new MyApi.Builder(  
            AndroidHttp.newCompatibleTransport(), new AndroidJsonFactory(), null)  
                .setRootUrl("https://{your-project-id}.appspot.com/_ah/api/");  
        apiService = builder.build();  
    }  
    return apiService;  
}
```

使用在线创建项目的 ID 替换 `{your-project-id}`。在设备或模拟器上构建并运行 App，应该会得到相同的结果。

14.6 小结

本章探讨了可用于分析和设计应用的各种工具，介绍了从不同方面研究 App 执行情况的诸多可用选项。你学习并使用新的 Navigation Editor 快速绘制构思的原型图，而这些构思在后期可以转换为具有完整功能的应用。最后，你深入研究了 Google 云服务并看到了如何使用 Google 强大的计算引擎来构建、测试和部署客户端/服务器应用。每款工具都为你提供了强大的控制力和洞察力，它们可以用于构建健壮的应用。

第15章

Android 可穿戴设备实验

Android Wear 是 Google 最近的技术创新之一，为更亲密的用户体验创造了机会。在编写本书时，只有少量设备支持 Android Wear，但这个数量正在不断增长。Android Wear 目前还只支持手表，但随着技术的成熟，可穿戴设备将会包括从项链甚至到服装的任何东西。在所有这些设备中，手表的前三名制造商是三星、摩托罗拉和索尼。本章将介绍如何构建可穿戴设备 App 以及使用 Android Studio 以有线和无线的方式部署并运行它。

注意

我们邀请你使用 Git 克隆此项目，以便跟随学习进度，尽管你将从头开始创建此项目的 Git 仓库。如果你的电脑上还没有安装 Git，那么请阅读第 7 章。打开 Windows 上的 Git-bash 会话(或者 Mac 和 Linux 上的终端)并导航至 C:\androidBook\reference\ (如果没有 reference 目录，那就创建它。在 Mac 上导航至 /your-labs-parent-dir/reference/)，输入以下 git 命令：`git clone https://bitbucket.org/csgerber/megadroid MegaDroid`。

15.1 设置可穿戴设备环境

在开始开发可穿戴设备 App 之前，需要完成几个步骤来准备自己的工作环境。虽然仅使用模拟器开发是可行的，但最好还是拥有一款实际的 Wear 设备。确保你的设备正运行着最新版本的操作系统，如果在 Windows PC 上工作，那么下载并安装所有必需的驱动程序。连接可穿戴设备并在 Android DDMS 工具窗口的设备列表中查找它。如果设备出现在列表中，那么可以跳过下一节。

15.1.1 安装设备驱动程序

在 Windows 上，如果想要通过 USB 部署 App，可能需要为某些设备安装驱动程序。注意，仅当设备连接后无法识别时安装驱动程序。如果打算使用蓝牙部署 App，那么可以跳过这一节。第一次连接设备时，Windows 将会尝试自动安装驱动程序，但是会失败。打

开 Windows Device Manager 并在列表中 Other Devices 节点的下面找到你的设备,如图 15-1 所示。

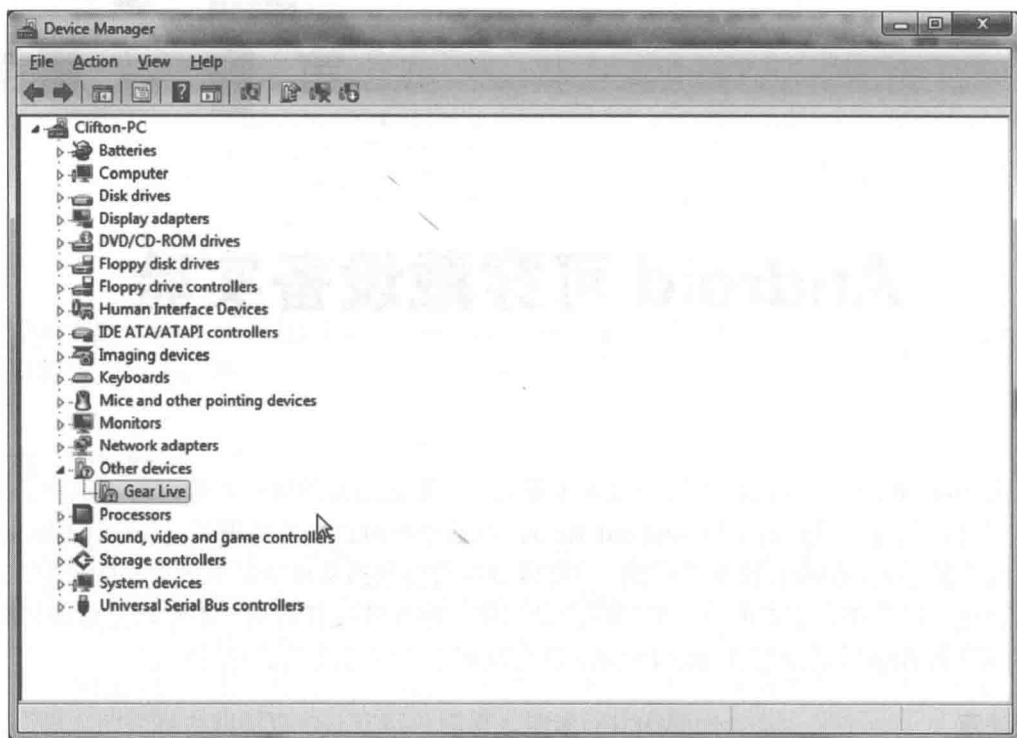


图 15-1 Device Manager 中列出了没有驱动程序的三星 Gear Live

右击未安装设备并单击上下文菜单中的 Update Driver Software, 从弹出框中选择 Browse my computer for driver software, 如图 15-2 所示。

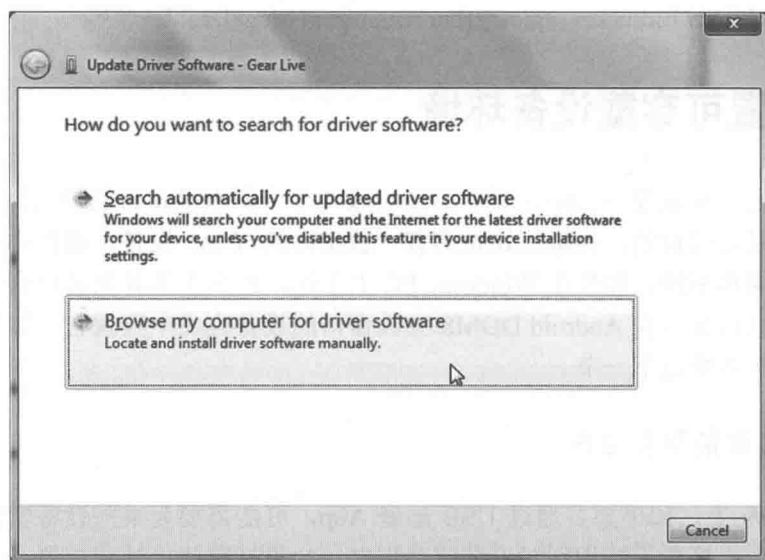


图 15-2 在你的电脑中查找驱动程序

单击 Let me pick from a list of device drivers on my computer, 如图 15-3 所示。

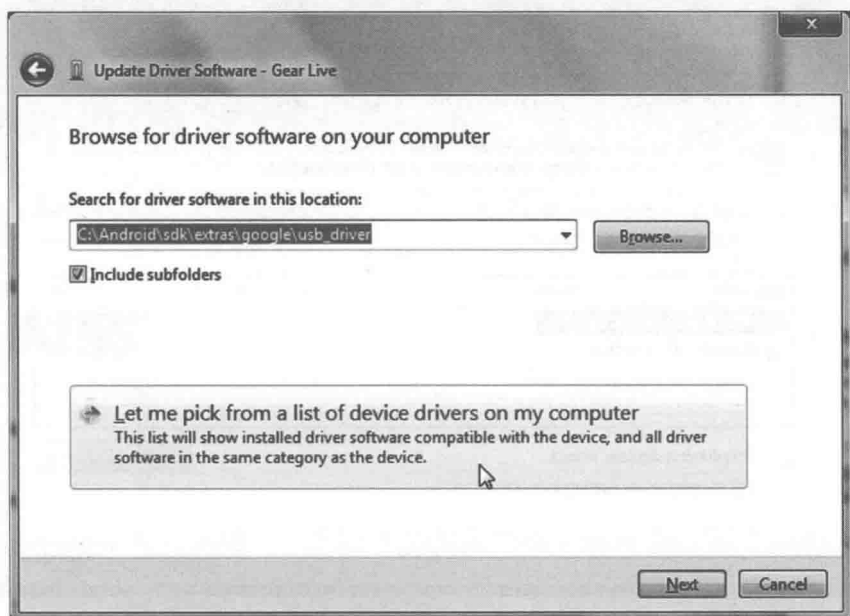


图 15-3 单击 Let me pick from a list of device drivers on my computer 选项

单击 Android Device, 如图 15-4 所示。

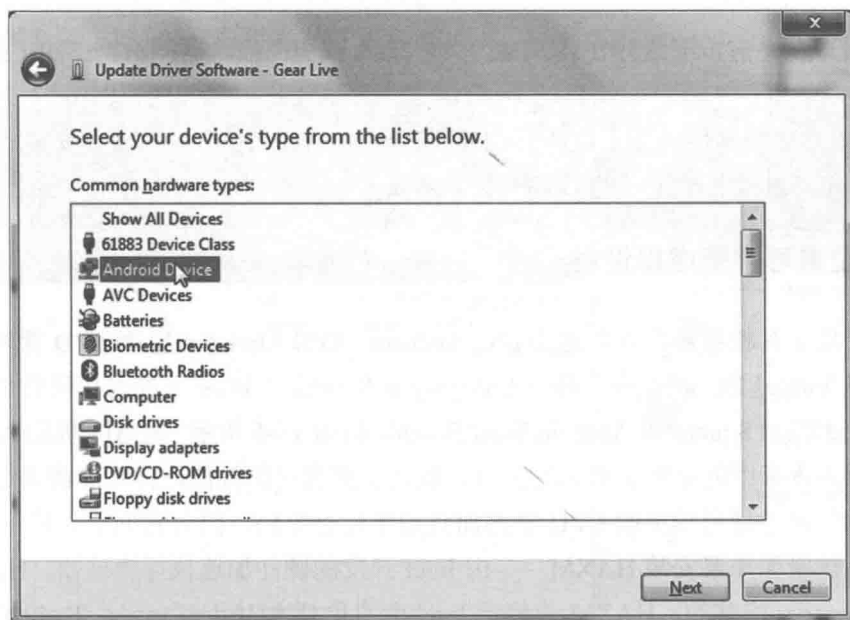


图 15-4 单击 Android Device

在下一个弹出框中选择复合驱动程序并单击 Next 按钮。驱动程序厂商千差万别, 可能与图 15-5 中所示的不同。可以安全地使用面向所有厂商的复合驱动程序。驱动程序将被安装, 而你也已经一切就绪。

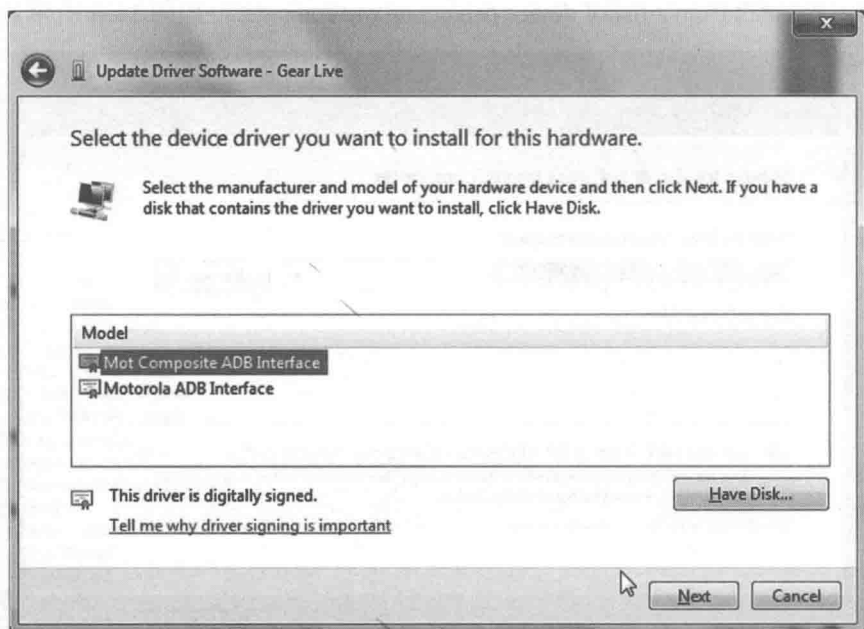


图 15-5 选择面向所有厂商的 Mot Composite ADB Interface

15.1.2 设置 SDK 工具

在开始开发之前，下载并安装 SDK 平台 5.0.1 或更高版本，同时将 SDK 工具更新到 24.0.2 或更高版本。平台版本 4.4W.2 和 SDK 工具版本 23.0 开始提供对 Android Wear 的支持。不过，本章中的示例使用了更新版 SDK 平台中包含的特性。你还需要安装更新版 SDK 平台的 Samples 以及 Extras 下的 Google Repository。

15.1.3 设置可穿戴虚拟设备

通过使用工具栏按钮或者单击 Tools | Android | AVD Manager 启动 AVD 管理器，接着单击 Create Virtual Device。在左侧的 Category 面板中选择 Wear 并在可用硬件类型列表中选择 Android Wear Square 或 Android Wear Round，如图 15-6 所示。将 API 级别选择为 5.0.1 或更高，因为本章中的示例需要 API 5.0.1。取决于开发机的性能，你可能需要选择 x86 系统镜像。这些镜像使用较少的 CPU 资源而且通常运行更快，因为它们不必模拟 CPU。不过，这些系统镜像需要安装 HAXM——由 Intel 开发的硬件加速执行管理器。HAXM 是通过 SDK Manager 安装的。HAXM 依赖于 Intel 虚拟化技术(Virtualization Technology, VT)的支持，某些机器可能不具备此特性。单击 Next 按钮并保留向导最后一页中的默认值。确保选择了 Use Host GPU 以提升速度。

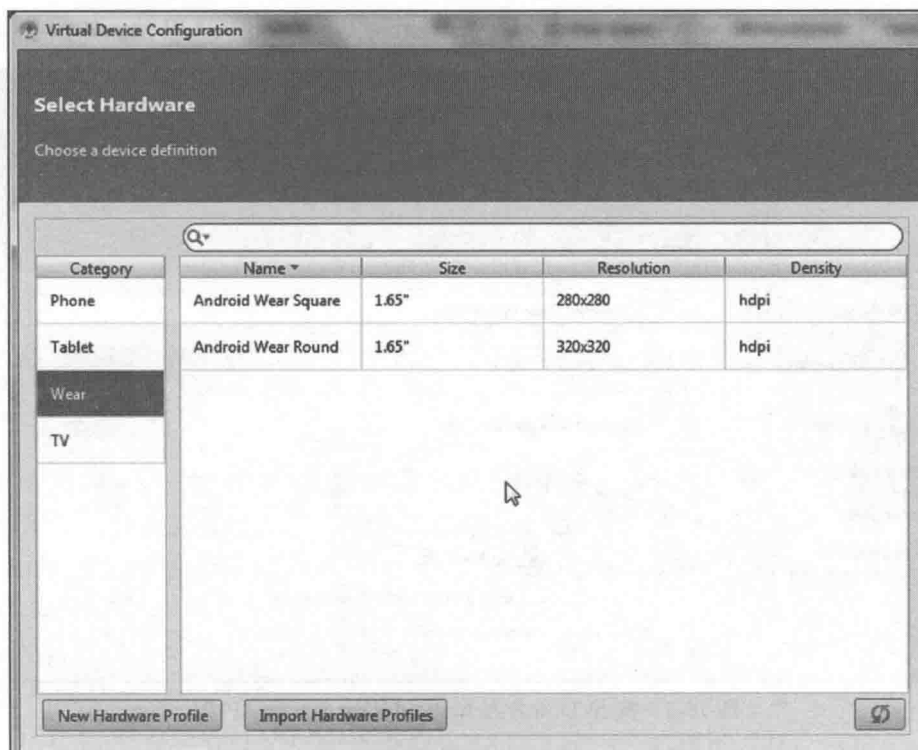


图 15-6 选择可穿戴设备类型

选择最新的 API 级别(编写本书时是 Lollipop)。接着单击 Next 按钮，如图 15-7 所示。

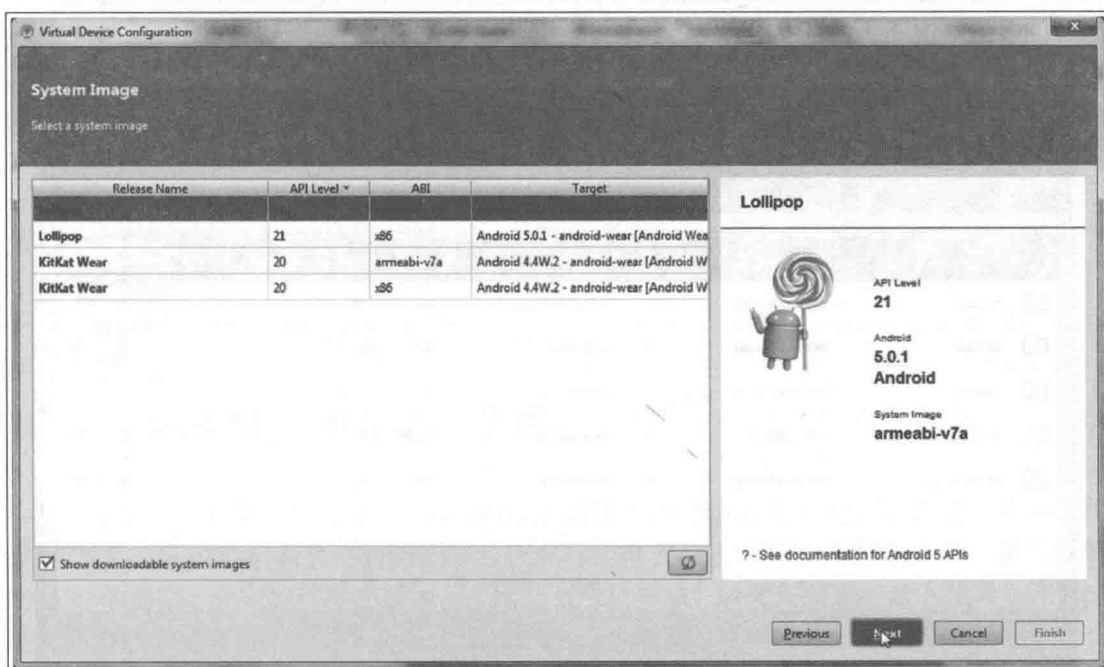


图 15-7 选择 Lollipop 作为系统镜像

在下一个界面中，将 AVD 的名称设置为 Android Wear Square API 21，如图 15-8 所示。

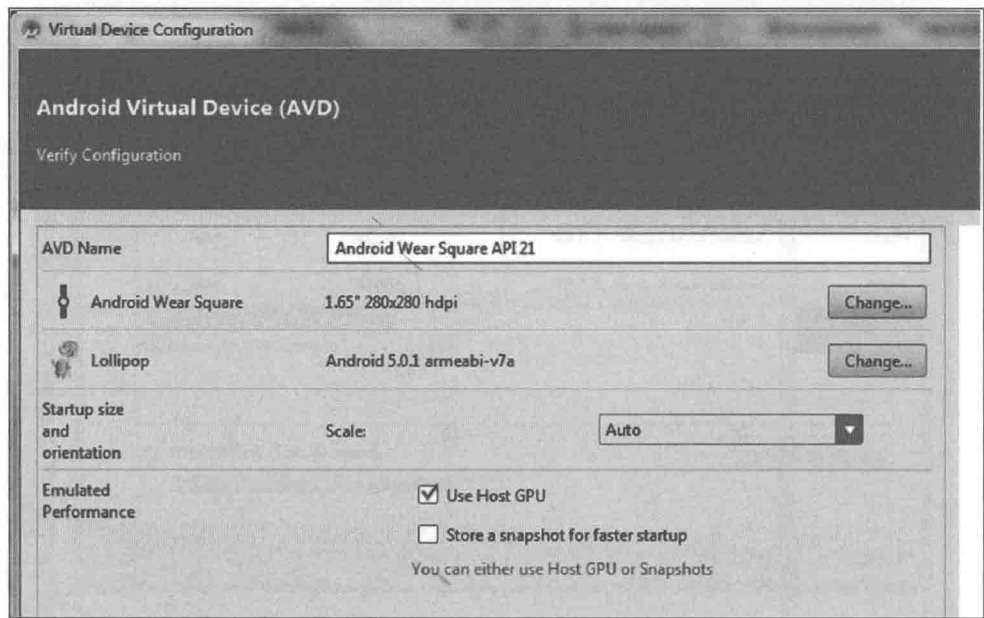


图 15-8 将 AVD 命名为 Android Wear Square API 21

单击 Finish 按钮来构建 AVD。构建完之后，可以在列表中单击 Wear AVD 旁边的下拉箭头并复制它，如图 15-9 所示。如果已经创建了 Square AVD，那么将其修改为使用圆形屏幕；否则，将其修改为使用方形屏幕。需要同时使用两种形状的屏幕来测试 App 能够在尽可能多的情况下正常工作。

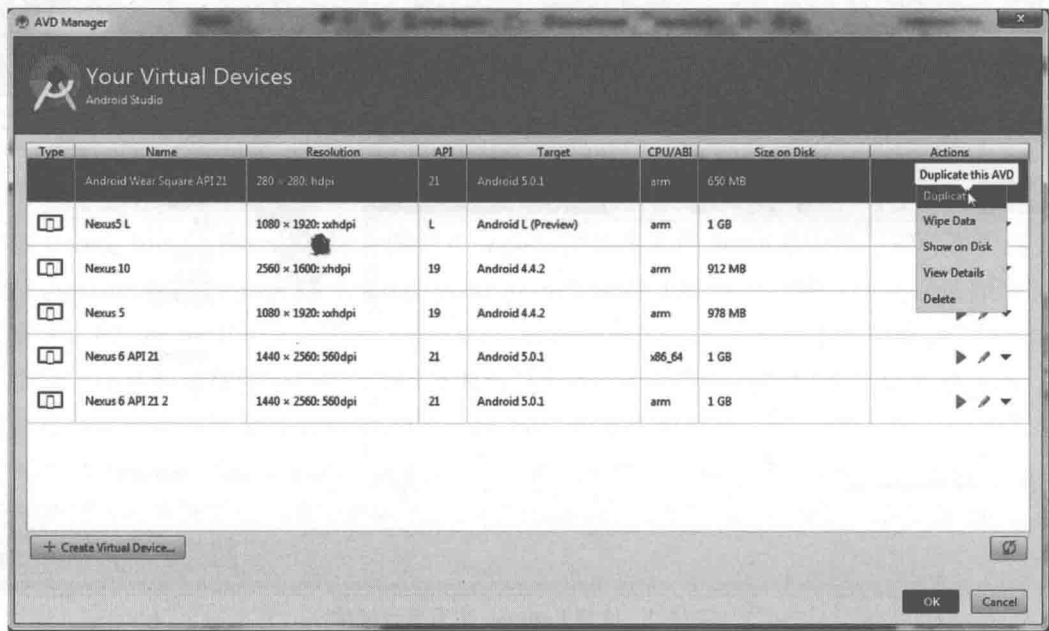


图 15-9 复制 AVD 以创建 Android Wear Round API 21

15.1.4 设置 Android 可穿戴设备硬件

如果拥有可穿戴设备,那么需要将其设置为允许进行开发。可穿戴设备 App 通过 Android 智能手机或平板电脑来部署和管理,因此为了在可穿戴设备上开发,你还需要有一款这样的设备。在 Android 智能手机上,安装 Google Play 上的 Android Wear App。启动该 App 并用它将你的智能手机和可穿戴设备配对。

有两种方法可以将 App 部署到可穿戴设备:有线或蓝牙。有线是最简单的方式,但如果 USB 端口不足或者不想与设备驱动程序打交道,那么蓝牙是一种不错的替代方案。当需要在一台仅有几个端口的电脑上支持智能手机、平板电脑以及 Wear 时就会遇到这种情况。

1. 启用开发者模式

如果没有启用过开发者模式或者你的设备是全新的,那么可以采用以下步骤来设置一些选项,例如一直开启模式、蓝牙调试、调试布局等:

- (1) 单击并按住侧面的按钮两秒钟,打开可穿戴设备上的 Settings App。
- (2) 滚动到底部并单击 About 选项。
- (3) 当打开 About 界面时,点击构建号 7 次。此后,你将会在 Settings 列表的 About 选项下面找到 Developer 选项。
- (4) 打开开发者选项并启用 ADB 调试。

2. 使用蓝牙调试

如果希望采用无线方式工作,那么在 Developer Options 中启用 Bluetooth。接下来,打开命令终端并运行以下两条 ADB 命令:

```
adb forward tcp:4444 localabstract:/adb-hub
adb connect localhost:4444
```

观察 Android 可穿戴 App 在移动设备上的运行状态,应该如下所示:

```
Host: connected
Target: connected
```

此时,你的可穿戴设备就可以安装 App 了。

15.2 创建 MegaDroid 项目

本节演示如何基于一个名为 MegaDroid 的虚拟电子游戏人物创建自定义手表界面项目。可以将 MegaDroid 视作 20 世纪 80 年代流行的电子游戏中两个人物(这里将不提及他们的名字)的混搭。手表界面将会展现一个太空战士的形象,他使用双剑与敌人战斗。该 App 是实际游戏的延伸。图 15-10 展示了此练习的最终结果。可以将这个示例作为参考,把自己的品牌展现在目标用户的手腕上。支持自定义手表界面是 Lollipop 中引入的新特性。这种特性允许你的 App 作为设备的实际界面来运行并为新型用户体验提供了诸多机会。你的

App 可以显示来自各种数据源的信息,包括但不限于 Internet、GPS、配对移动设备上的日程表或联系人列表以及更多。由于手表界面是一个持续运行的完整 Android App,因此它可以作为 App 的整体扩展。这个示例只涵盖了绘制用户界面以及从运行时获取更新来改变时间的基础知识。



图 15-10 MegaDroid 手表界面的最终结果

使用第 1 章中介绍的 New Project Wizard 开启新的 Android Wear 项目。打开向导的第二页,选中 Wear 复选框并选择 SDK 5.0 或更高版本,如图 15-11 所示。保留界面中其余部分的默认值,选择用于移动 App 的空白 Activity 和用于 Wear 组件的空白 Wear Activity。在最后一页完成向导并等待项目构建。单击 Run 按钮,在可穿戴设备或 AVD 上测试你的项目。

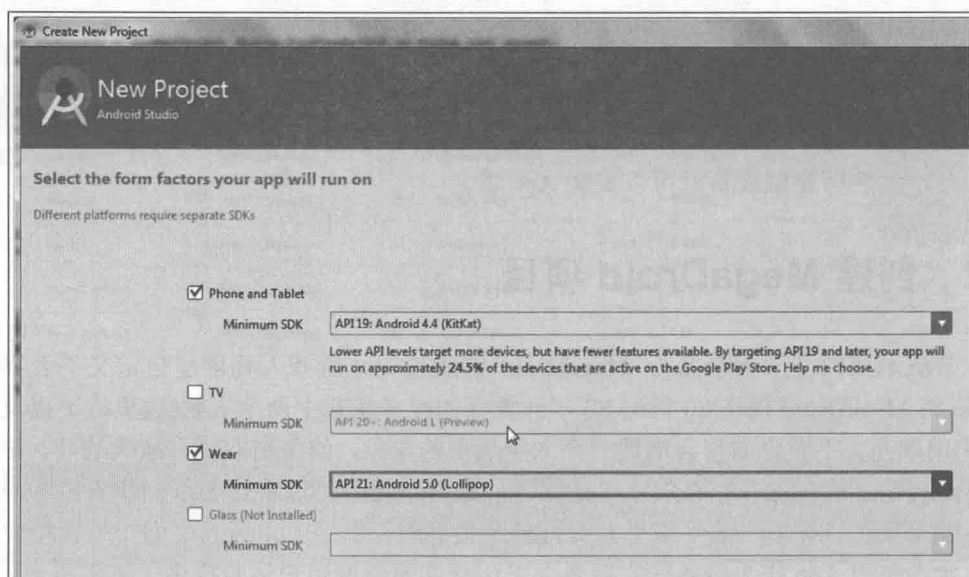


图 15-11 在 New Project Wizard 中选择 Wear

在为可穿戴设备设计 App 时,掌握 Android 的布局和设计很重要。对于一款优秀的产品来说,最初的开发时间大多数会花费在图形编辑器中,主要关注于设计方法、尺寸、颜色以及类似的事情。每个手表界面都是唯一的,采用的方法也会有差别,而这取决于要实现的目标。设计一款简单数字时钟界面所采用的方法与设计模拟时钟的不同,而且需要付出的努力更少一些。Android 网站上的在线开发者文档可能会吓到那些没有太多设计经验的人。大体来说,网站建议你应该在设计时考虑方形和圆形屏幕、确定是否要整合额外数据、允许系统 UI 元素保持可见并支持不同的显示模式。这些显示模式会在后续章节中介绍。你可能会考虑提供一个配置界面。但本例试图简化此过程并有意地忽略某些步骤。

15.2.1 针对屏幕的优化技术

Wear 运行时将会以两种显示模式执行 App: 环境模式和交互模式。当查看或使用 App 时,手表将会在这些模式之间切换。系统会自动启用环境模式来延长电池寿命。其结果是, Wear App 会自动检测到这种模式并相应地将其显示输出改变为使用暗淡的颜色。在这种模式中,更新每分钟发送一次,因此同步降低屏幕绘制次数也是有意义的。本例将会在这种模式下删除秒针并将绘制频率从每秒调整为每分钟绘制一次屏幕。

某些设备支持低比特环境模式。在此模式下,设备屏幕将会使用受限的颜色调色板。这有助于降低耗电量并防止屏幕烧坏。可以检测此模式并使用仅有的颜色(黑、白、蓝、红、品红、绿、青和黄)调整自己的图像。也可以使用轮廓来代替整幅图片。在低比特模式下,背景应该主要是黑色的。非黑色像素不应该超过总像素数的 10%,而白色像素占据屏幕的大小不应该超过 5%。这适用于支持此种特殊绘图模式的设备。在此种模式下绘图时,你还应该禁用抗锯齿。抗锯齿是一种在绘图时模糊边缘(令其看上去不那么像素化)的技术,如图 15-12 和图 15-13 所示。



图 15-12 没有抗锯齿的图像

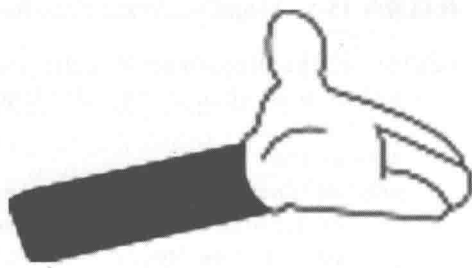


图 15-13 应用了抗锯齿的图像

对于本例来说,为了简便,我们将在环境模式中使用图像的灰度版本,如图 15-14 所示。从参考克隆版本中将所有图片复制到当前项目中。在 Windows 上,导航至 C:\androidBook\reference\MegaDroid\wear\src\main 文件夹,接着右击并复制 res 目录。导航至 C:\androidBook\MegaDroid\wear\src\main\res 文件夹,接着右击并粘贴已复制的文件夹,覆盖现有 res 文件夹。在 Mac 或 Linux 终端运行以下命令:

```
cp -R ~/androidBook/reference/MegaDroid/wear/src/main/res
```

~/androidBook/MegaDroid/wear/src/main/

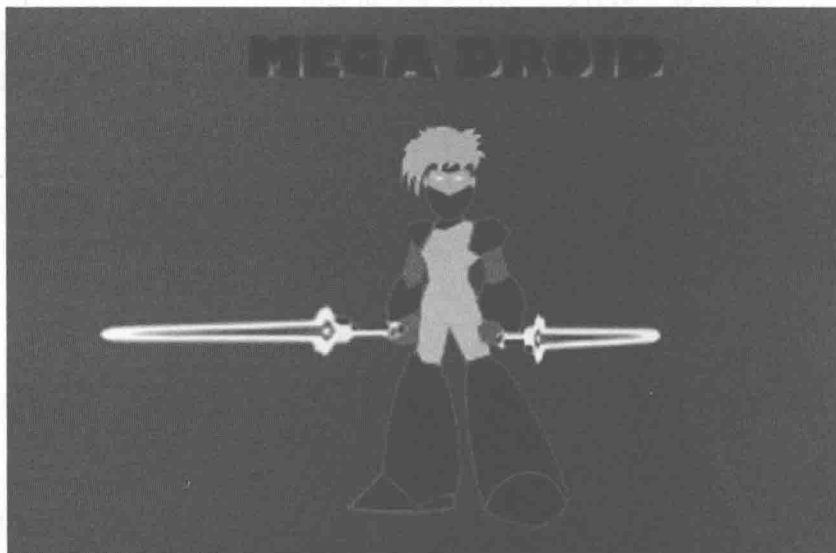


图 15-14 灰度图像

15.2.2 构建 watch-face 服务

watch-face 服务负责创建 `WatchFaceService.Engine`，它是手表界面的核心。`WatchFaceService.Engine` 响应系统回调并负责更新时间和绘制界面。创建一个新的 `MegaDroidWatchFaceService` 类，它派生自 `CanvasWatchFaceService` 类。使用代码清单 15-1 中的代码填充它。

代码清单 15-1 `MegaDroidWatchFaceService` 类

```
public class MegaDroidWatchFaceService extends CanvasWatchFaceService {
    private static final String TAG = "MegaDroidWatchSvc";

    @Override
    public Engine onCreateEngine() {
        // create and return the watch face engine
        return new MegaDroidEngine(this);
    }

    /* implement service callback methods */
    private class MegaDroidEngine extends CanvasWatchFaceService.Engine {

        private final Service service;

        public MegaDroidEngine(Service service) {
            this.service = service;
        }
    }
}
```

```

/**
 * initialize your watch face
 */
@Override
public void onCreate(SurfaceHolder holder) {
    super.onCreate(holder);
}

/**
 * called when system properties are changed
 * use this to capture low-bit ambient.
 */
@Override
public void onPropertiesChanged(Bundle properties) {
    super.onPropertiesChanged(properties);
}

/**
 * This is called by the runtime on every minute tick
 */
@Override
public void onTimeTick() {
    super.onTimeTick();
}

/**
 * Called when there's a switched in/out of ambient mode
 */
@Override
public void onAmbientModeChanged(boolean inAmbientMode) {
    super.onAmbientModeChanged(inAmbientMode);
}

@Override
public void onDraw(Canvas canvas, Rect bounds) {
    //Draw the watch face here
}

/**
 * Called when the watch face becomes visible or invisible
 */
@Override
public void onVisibilityChanged(boolean visible) {
    super.onVisibilityChanged(visible);
}
}
}

```

注册服务

在 Android manifest 中 `application` 关闭标签的前面添加以下内容:

```
<service
    android:name=".MegaDroidWatchFaceService"
    android:label="@string/mega_droid_service_name"
    android:allowEmbedded="true"
    android:taskAffinity=""
    android:permission="android.permission.BIND_WALLPAPER" >
    <meta-data
        android:name="android.service.wallpaper"
        android:resource="@xml/watch_face" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview"
        android:resource="@drawable/preview_analog" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview_circular"
        android:resource="@drawable/preview_analog_circular" />
    <intent-filter>
        <action android:name="android.service.wallpaper.WallpaperService" />
        <category
            android:name=
                "com.google.android.wearable.watchface.category.WATCH_FACE" />
    </intent-filter>
</service>
```

按 F2 功能键遍历这些错误。按 `Alt + Enter` 键显示建议并逐个修改错误。第一个建议是要求你为新的服务生成一个名称。这将在设备上显示的名称(位于手表界面的图片和名称列表中)。第二个建议是为 `wallpaper` 元数据标签生成 XML 描述符。创建 `xml/watch_face.xml` 并使用以下代码进行填充:

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

接下来的两个错误不应该采用 IntelliJ 的建议来修复。它们是指向可绘制资源的引用, 这些可绘制资源将成为界面选取器中手表界面的预览。需要使用图形编辑器来创建它们。此外, 你还可以截取 App 的屏幕快照, 但这似乎有些本末倒置。如果不熟悉图像编辑器或图形设计程序, 那么可以在这里临时添加一些占位图片, 让 App 运行起来。在 App 基本完成之后, 你再回过头来截取手表运行中的屏幕快照并使用它们。

15.2.3 初始化可绘制资源和样式

在 `onCreate()` 方法中, 添加以下逻辑, 设置样式并初始化可绘制资源:

```
public void onCreate(SurfaceHolder holder) {
    super.onCreate(holder);
    setWatchFaceStyle(new WatchFaceStyle.Builder(service)
```

```

        .setCardPeekMode(WatchFaceStyle.PEEK_MODE_SHORT)
        .setStatusBarGravity(Gravity.RIGHT | Gravity.TOP)
        .setHotwordIndicatorGravity(Gravity.LEFT | Gravity.TOP)
        .setBackgroundVisibility(WatchFaceStyle.BACKGROUND_VISIBILITY_INTERRUPTIVE)
        .setShowSystemUiTime(false)
        .build());

Resources resources = service.getResources();
Drawable backgroundDrawable = resources.getDrawable(R.drawable.bg);
this.backgroundBitmap = ((BitmapDrawable) backgroundDrawable).getBitmap();
this.character = ((BitmapDrawable) resources.getDrawable(
    R.drawable.character_standing)).getBitmap();
this.logo = ((BitmapDrawable) resources.getDrawable(
    R.drawable.megadroid_logo)).getBitmap();
this.minuteHand = ((BitmapDrawable) resources.getDrawable(
    R.drawable.minute_hand)).getBitmap();
this.hourHand = ((BitmapDrawable) resources.getDrawable(
    R.drawable.hour_hand)).getBitmap();

this.secondPaint = new Paint();
secondPaint.setARGB(255, 255, 0, 0);
secondPaint.setStrokeWidth(2.f);
secondPaint.setAntiAlias(true);
secondPaint.setStrokeCap(Paint.Cap.ROUND);

this.time = new Time();
}

```

15.2.4 管理手表更新

在 MegaDroid 内部添加如下两个静态字段：

```

private static final long INTERACTIVE_UPDATE_RATE_MS =
    TimeUnit.SECONDS.toMillis(1);
private static final int MSG_UPDATE_TIME = 0;

```

现在定义一个更新处理程序，它将基于之前定义的常量 `INTERACTIVE_UPDATE_RATE_MS` 来触发手表更新：

```

/** Handler to update the time once a second in interactive mode. */
final Handler mUpdateTimeHandler = new Handler() {
    @Override
    public void handleMessage(Message message) {
        switch(message.what) {
            case MSG_UPDATE_TIME:
                if(Log.isLoggable(TAG, Log.VERBOSE)) {
                    Log.v(TAG, "updating time");
                }
                invalidate();
            }
        }
    }
}

```

```

        if(shouldTimerBeRunning()) {
            long timeMs = System.currentTimeMillis();
            long delayMs = INTERACTIVE_UPDATE_RATE_MS
                - (timeMs % INTERACTIVE_UPDATE_RATE_MS);
            mUpdateTimeHandler.sendEmptyMessageDelayed(
                MSG_UPDATE_TIME, delayMs);
        }
        break;
    }
}

};

private boolean shouldTimerBeRunning() {
    return isVisible() && !isInAmbientMode();
}

```

在最初被调用之后，这个处理程序将会基于更新间隔来持续调度更新。它只是令显示失效，而这会间接地触发 `onDraw` 调用。在接着重新调度未来更新之前，它会检查手表界面是否可见以及是否处于环境模式。按照如下方式实现 `onDestroy` 方法——当服务运行时作为垃圾回收清除更新处理程序：

```

@Override
public void onDestroy() {
    mUpdateTimeHandler.removeMessages(MSG_UPDATE_TIME);
    super.onDestroy();
}

```

实现 `onPropertiesChanged` 方法来跟踪 `lowBitAmbient` 模式。设置一个 `Boolean` 成员变量来检查环境模式是否正在运行。需要据此来决定何时降低绘制频率。在实现中使用如下代码：

```

public void onPropertiesChanged(Bundle properties) {
    super.onPropertiesChanged(properties);
    this.lowBitAmbient = properties.getBoolean(
        PROPERTY_LOW_BIT_AMBIENT, false);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onPropertiesChanged: low-bit ambient = " + lowBitAmbient);
    }
}

```

还需要定义 `lowBitAmbient` 成员字段：

```

private boolean lowBitAmbient;

```

在 `onTimeTick` 方法中添加对 `invalidate` 的调用。你在这里调用 `invalidate`，它会触发屏幕的重新绘制：

```

@Override
public void onTimeTick() {
    super.onTimeTick();
    if (Log.isLoggable(TAG, Log.DEBUG)) {

```

```

        Log.d(TAG, "onTimeTick: ambient = " + isInAmbientMode());
    }
    invalidate();
}

```

现在实现 `onAmbientModeChanged` 回调。当处于环境模式时，使用黑白图像。你还将通过关闭秒针的抗锯齿绘制来进行优化，后面会介绍。

```

public void onAmbientModeChanged(boolean inAmbientMode) {
    super.onAmbientModeChanged(inAmbientMode);
    if(Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onAmbientModeChanged: " + inAmbientMode);
    }
    if(inAmbientMode) {
        character = ((BitmapDrawable) service.getResources().getDrawable(
            R.drawable.character_standing_greyscale)).getBitmap();
        logo = ((BitmapDrawable) service.getResources().getDrawable(
            R.drawable.megadroid_logo_bw)).getBitmap();
        hourHand = ((BitmapDrawable) service.getResources().getDrawable(
            R.drawable.hour_hand_bw)).getBitmap();
        minuteHand = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.minute_hand_bw)).getBitmap();
    } else {
        character = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.character_standing)).getBitmap();
        logo = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.megadroid_logo)).getBitmap();
        hourHand = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.hour_hand)).getBitmap();
        minuteHand = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.minute_hand)).getBitmap();
    }
    if(!lowBitAmbient) {
        boolean antiAlias = !inAmbientMode;
        secondPaint.setAntiAlias(antiAlias);
    }
    invalidate();

    // Whether the timer should be running depends on whether
    // we're in ambient mode (as well
    // as whether we're visible), so we may need to start
    // or stop the timer.
    updateTimer();
}

```

这里调用了 `updateTimer` 方法，接下来你要定义它。`updateTimer` 方法将会把空的更新消息发送至 `mUpdateTimeHandler`。你希望仅当手表界面可见且没有处于环境模式时发送更新。将以下代码片段作为你的实现：

```

private void updateTimer() {

```

```

        if (Log.isLoggable(TAG, Log.DEBUG)) {
            Log.d(TAG, "updateTimer");
        }
        mUpdateTimeHandler.removeMessages(MSG_UPDATE_TIME);
        if (shouldTimerBeRunning()) {
            mUpdateTimeHandler.sendEmptyMessage(MSG_UPDATE_TIME);
        }
    }
}

```

使用代码清单 15-2 定义广播接收器来响应时区的变化。注册和反注册方法将用于启用监听时区变化事件的接收器。

代码清单 15-2 时区 BroadcastReceiver

```

final BroadcastReceiver mTimeZoneReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        time.clear(intent.getStringExtra("time-zone"));
        time.setToNow();
    }
};

boolean mRegisteredTimeZoneReceiver = false;

private void registerReceiver() {
    if (mRegisteredTimeZoneReceiver) {
        return;
    }
    mRegisteredTimeZoneReceiver = true;
    IntentFilter filter = new IntentFilter(Intent.ACTION_TIMEZONE_CHANGED);
    service.registerReceiver(mTimeZoneReceiver, filter);
}

private void unregisterReceiver() {
    if (!mRegisteredTimeZoneReceiver) {
        return;
    }
    mRegisteredTimeZoneReceiver = false;
    service.unregisterReceiver(mTimeZoneReceiver);
}

```

此接收器会在接收到消息时更新时间。它注册了与 ACTION_TIMEZONE_CHANGED 动作绑定的 IntentFilter。使用 IntentFilter 是一种将 Activity 或 BroadcastReceiver 绑定到特定类型 Intent 动作的编程方法。

现在定义 onVisibilityChanged 回调，注册接收器并启动更新处理程序：

```

@Override
public void onVisibilityChanged(boolean visible) {
    super.onVisibilityChanged(visible);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onVisibilityChanged: " + visible);
    }
}

```



```

    }

    if(visible) {
        registerReceiver();

        // Update time zone in case it changed while we weren't visible.
        time.clear(TimeZone.getDefault().getID());
        time.setToNow();
    } else {
        unregisterReceiver();
    }

    // Whether the timer should be running depends on whether
    // we're visible (as well as
    // whether we're in ambient mode), so we may need to start
    // or stop the timer.
    updateTimer();
}

```

15.2.5 绘制界面

绘制手表界面需要大量逻辑。对 `invalidate` 的调用最终会导致对 `onDraw` 方法的调用。在此方法中，你将要综合利用各种图形资源来绘制界面。每次更新都会根据已逝去的时、分和秒来转动和绘制表针。重载 `onDraw` 方法并使用代码清单 15-3 中的代码。

代码清单 15-3 完整的 `onDraw` 方法实现

```

public void onDraw(Canvas canvas, Rect bounds) {
    time.setToNow();

    int width = bounds.width();
    int height = bounds.height();
    // Draw the background, scaled to fit.
    if(backgroundScaledBitmap == null
        || backgroundScaledBitmap.getWidth() != width
        || backgroundScaledBitmap.getHeight() != height) {
        backgroundScaledBitmap = Bitmap.createScaledBitmap(backgroundBitmap,
            width, height, true /* filter */);
    }
    canvas.drawBitmap(backgroundScaledBitmap, 0, 0, null);

    canvas.drawBitmap(character, (width- character.getWidth())/2,
        ((height- character.getHeight())/2)+ 20, null);
    canvas.drawBitmap(logo, (width- logo.getWidth())/2,
        (logo.getHeight()*2), null);

    float secRot = time.second / 30f * (float) Math.PI;
    int minutes = time.minute;
    float minRot = minutes / 30f * (float) Math.PI;
}

```

```

float hrRot = ((time.hour + (minutes / 60f)) / 6f) * (float) Math.PI;

// Find the center. Ignore the window insets so that, on round
//watches with a "chin", the watch face is centered on the
//entire screen, not just the usable portion.
float centerX = width / 2f;
float centerY = height / 2f;

Matrix matrix = new Matrix();
int minuteHandX = ((width - minuteHand.getWidth()) / 2)
    - (minuteHand.getWidth() / 2);
int minuteHandY = (height - minuteHand.getHeight()) / 2;
matrix.setTranslate(minuteHandX-20, minuteHandY);
float degrees = minRot * (float) (180.0 / Math.PI);
matrix.postRotate(degrees+90, centerX, centerY);
canvas.drawBitmap(minuteHand, matrix, null);

matrix = new Matrix();
int rightArmX = ((width - hourHand.getWidth()) / 2)
    + (hourHand.getWidth() / 2);
int rightArmY = (height - hourHand.getHeight()) / 2;
matrix.setTranslate(rightArmX + 20, rightArmY);
degrees = hrRot * (float) (180.0 / Math.PI);
matrix.postRotate(degrees-90, centerX, centerY);
canvas.drawBitmap(hourHand, matrix, null);

float secLength = centerX - 20;
if(!isInAmbientMode()) {
    float secX = (float) Math.sin(secRot) * secLength;
    float secY = (float) -Math.cos(secRot) * secLength;
    canvas.drawLine(centerX, centerY, centerX + secX,
        centerY + secY, secondPaint);
}
}

```

分段研究此代码将有助于你理解整体流程。首先，获取传入方法的 **bounds** 对象的宽度和高度：

```

int width = bounds.width();
int height = bounds.height();

```

接下来检查背景的缩放版本并将其绘制到屏幕上。需要将背景缩放至给定的宽度和高度。你只需要缩放一次，因为每次重绘时 **bounds** 是一样的。

```

// Draw the background, scaled to fit.
if(backgroundScaledBitmap == null
    || backgroundScaledBitmap.getWidth() != width

```

```

    || backgroundScaledBitmap.getHeight() != height) {
        backgroundScaledBitmap = Bitmap
            .createScaledBitmap(backgroundBitmap,
                width, height, true /* filter */);
    }
    canvas.drawBitmap(backgroundScaledBitmap, 0, 0, null);

```

现在绘制标志下面的人物。这些数学计算用于将人物水平居中，但让其相对竖直中心点有 20 像素的偏移。为了居中，可以计算边界宽度和人物宽度之差，然后除以 2。对高度的处理方法相同，但在偏移量上增加了 20 像素。

```

canvas.drawBitmap(character, (width- character.getWidth())/2,
    ((height- character.getHeight())/2)+ 20, null);
canvas.drawBitmap(logo, (width- logo.getWidth())/2,
    (logo.getHeight()*2), null);

```

接下来一段代码使用一些几何知识计算出分针、时针和秒针的旋转角度。它使用公式，将已逝去的秒数和分钟数除以 30，然后乘以 p 。时针则更加复杂一些。使用小时数加上已逝去分钟数的微小偏移，然后除以 6。接着将得到的结果乘以 p 。用已逝去的分钟数除以 60 来计算分针偏移，因为每分钟是一小时的 $1/60$ 。偏移量是可选的。如果想让时针直接指向当前小时而不用缓缓移动，那么可以忽略这部分计算：

```

float secRot = time.second / 30f * (float) Math.PI;
int minutes = time.minute;
float minRot = minutes / 30f * (float) Math.PI;
float hrRot = ((time.hour + (minutes / 60f)) / 6f ) * (float) Math.PI;

```

接着找到屏幕的中心点来固定时针、分针和秒针：

```

float centerX = width / 2f;
float centerY = height / 2f;

```

使用中心点，在绘制分针之前围绕屏幕的中心执行偏移和旋转。可使用 minRot 乘以 $180/p$ 来计算角度。这个示例中的图形直接指向了 9 点钟方向，因此需要在旋转角度上增加 90° ：

```

Matrix matrix = new Matrix();
int minuteHandX = ((width - minuteHand.getWidth()) / 2)
    - (minuteHand.getWidth() / 2);
int minuteHandY = (height - minuteHand.getHeight()) / 2;
matrix.setTranslate(minuteHandX-20, minuteHandY);
float degrees = minRot * (float) (180.0 / Math.PI);

matrix.postRotate(degrees+90, centerX, centerY);
canvas.drawBitmap(minuteHand, matrix, null);

```

时针采用基本相同的操作，但由于图形指向与分针相反的方向，因此需要从旋转角度减去 90° ：

```

matrix = new Matrix();
int rightArmX = ((width - hourHand.getWidth()) / 2) + (hourHand.getWidth() / 2);
int rightArmY = (height - hourHand.getHeight()) / 2;
matrix.setTranslate(rightArmX + 20, rightArmY);
degrees = hrRot * (float) (180.0 / Math.PI);
matrix.postRotate(degrees-90, centerX, centerY);
canvas.drawBitmap(hourHand, matrix, null);

```

最后绘制秒针，它就是从屏幕中心延伸出的一条红色线段。通过从中心点减去 20 像素来计算秒针的长度。将绘制逻辑置于一个条件块中，当设备处于环境模式时不会进入这个块。x 坐标终点由旋转角度的正弦值乘以秒针的长度来确定。y 坐标由旋转角度余弦值的相反数乘以秒针的长度来确定。有了这些坐标之后，调用 canvas 的 drawLine 方法，向其传入中心点的 X 和 Y 坐标，并将计算所得的 secX 和 secY 与中心点相加以确定线段的终点。我们使用在 onCreate 方法中创建的画笔对象将它们全部绘制出来：

```

float secLength = centerX - 20;

if(!isInAmbientMode()) {
    float secX = (float) Math.sin(secRot) * secLength;
    float secY = (float) -Math.cos(secRot) * secLength;
    canvas.drawLine(centerX, centerY, centerX + secX,
        centerY + secY, secondPaint);
}

```

现在构建并运行你的手表服务并将它部署到设备上。图 15-15 展示了该例在 Galaxy Gear Live 手表上的运行效果。

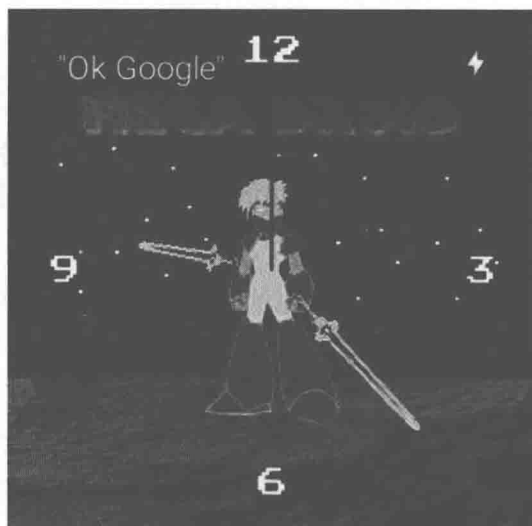


图 15-15 运行在设备上的 MegaDroid 手表界面

15.3 小结

在这个练习中，你学会了设计自定义手表界面的方法。你学到了如何通过 USB 和蓝牙来部署可穿戴设备 App。你学习了如何响应环境模式以节省电量。你研究了在设计手表界面过程中涉及的各种组件，包括服务和引擎，以及控制重绘频率的自定义定时器。本章只是初步介绍，可穿戴设备 App 的世界中还存在着大量机遇。手表界面拥有系统服务的全部访问权限，可以获取用于自定义显示的日程表条目、通讯录联系人、电池寿命信息以及更多。

第 16 章

定制 Android Studio

Android Studio 基于 IntelliJ IDEA，后者已经发展了很多年。改进之处包括随着每次软件发布引入的大量可定制特性。诸多可定制特性，加之数以百计的第三方插件，使得 IntelliJ 以及基于 IntelliJ 扩展而来的 Android Studio，成为市场上最具可定制性和灵活性的 IDE 之一。事实上，几乎所有你能够想到需要在 IDE 中定制的特性，在 Android Studio 中基本都可以定制。Android Studio 中可定制的特性是如此之多，以至于我们无法有效地将它们全部涵盖。在本书中，我们已经讨论了 Android Studio 中一些最重要的可定制特性，包括工具按钮和默认布局(见第 2 章)，以及动态模板、代码生成和代码风格(见第 3 章)。

本章展示其余一些我们认为对于 Android 开发者最有帮助的可定制特性。为了充分利用 Android Studio 中的可定制特性，应该熟悉 Settings 的键盘快捷键命令(Ctrl + Alt + S | Cmd + 逗号)。此操作位于主菜单中的 File | Settings(或者如果在 Mac 上，就是 File | Preferences)。键盘快捷键和菜单操作都可以打开 Settings 对话框。

Settings 对话框中包含了 Android Studio 的大部分可定制特性，我们将在本章中介绍如何使用它的诸多选项卡和子面板。在讨论 Settings 对话框中的特性时，请参考图 16-1。左侧面板包含一系列可定制的特性。这个列表被划分为两个区域：Project Settings 和 IDE Settings。你在前者对任何项目所做的修改都可以应用于当前项目或所有项目，而在后者中的任何修改都将应用于当前和未来的所有项目。Settings 对话框的左上角还包含一个过滤器栏。当在过滤器栏中输入文本时，它下方的列表将只会显示匹配该文本的条目。

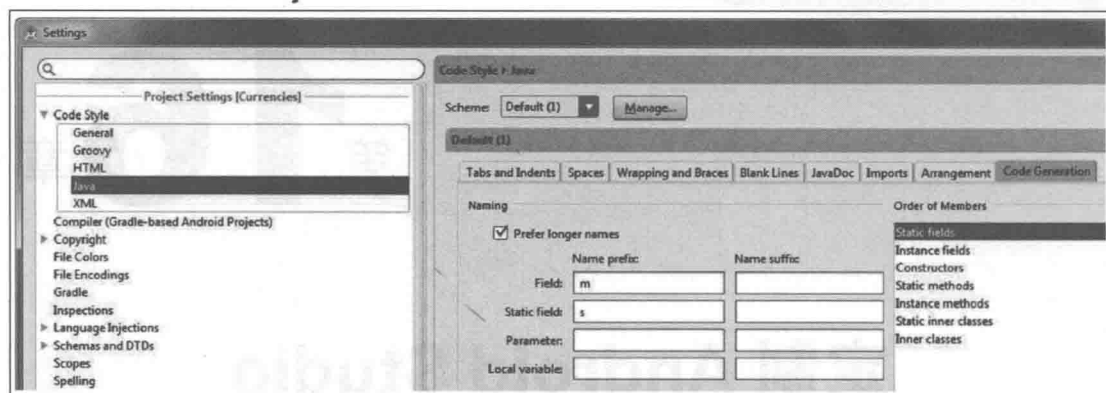


图 16-1 显示“Java | Code Generation”的 Settings 对话框

16.1 代码风格

尽管第 3 章涉及代码风格，但我们还是在这里更详细地讨论这个重要主题。通过按 **Ctrl + Alt + S** | **Cmd + 逗号** 来激活 Settings 对话框。在左侧面板中打开 Code Style 目录并选择 Java。选择右侧面板中的 Code Generation 选项卡。

如果完成了第 3 章中的练习，那么你应该会在 Field 和 Static Field 文本区域中看到小写的 m 和 s。如果没有看到这些字符，就在相应的框栏中输入它们。假定你遵从 Android 中的命名约定，即用 m(表示成员，例如 mCurrencies)或 s(表示静态成员，例如 sName)作为类成员名称的前缀，当自动生成 getter、setter、构造函数和其他代码时，这些前缀让 Android Studio 能够生成有意义的方法名。我们强烈建议你遵从这个命名约定，因而这个特殊设置是可以做的最重要设置之一。

选择 Arrangement 选项卡，它位于 Code Generation 选项卡的左侧，如图 16-3 所示。Arrangement 选项卡的目的是组织源文件中代码元素的顺序。Java 软件开发者的习惯是首先声明成员，然后是构造函数、方法、内部类等。Arrangement 选项卡允许你设置代码元素的顺序。作为一名软件工程师，你应该维护一个整洁有序的代码库。不过，不用考虑按照正确顺序来插入代码元素，因为 Android Studio 将会自动为你重新排列它们。要应用 Arrangement 设置，按 **Ctrl + Alt + L** | **Cmd + Alt + L** 并确保选中了 Rearrange entries 复选框；然后单击 Run，如图 16-2 所示。所得源文件中的元素将会根据你在 Arrangement 选项卡中选择的顺序重新排列。也可以通过在主菜单中选择 **Code | Rearrange Code** 来完成相同的功能。

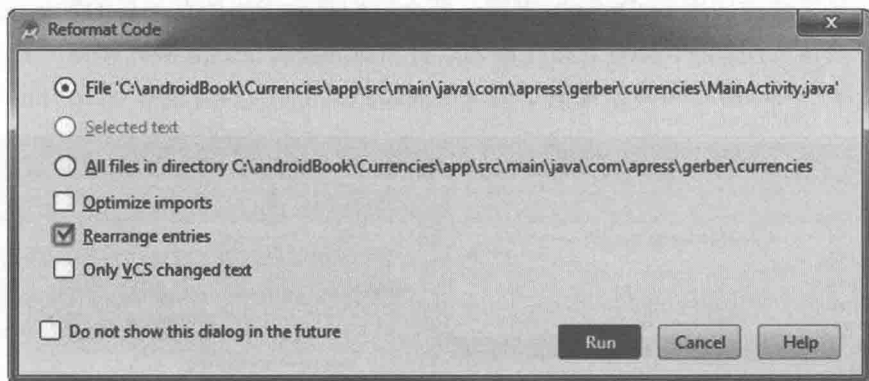


图 16-2 显示 Java | Arrangement 的 Settings 对话框

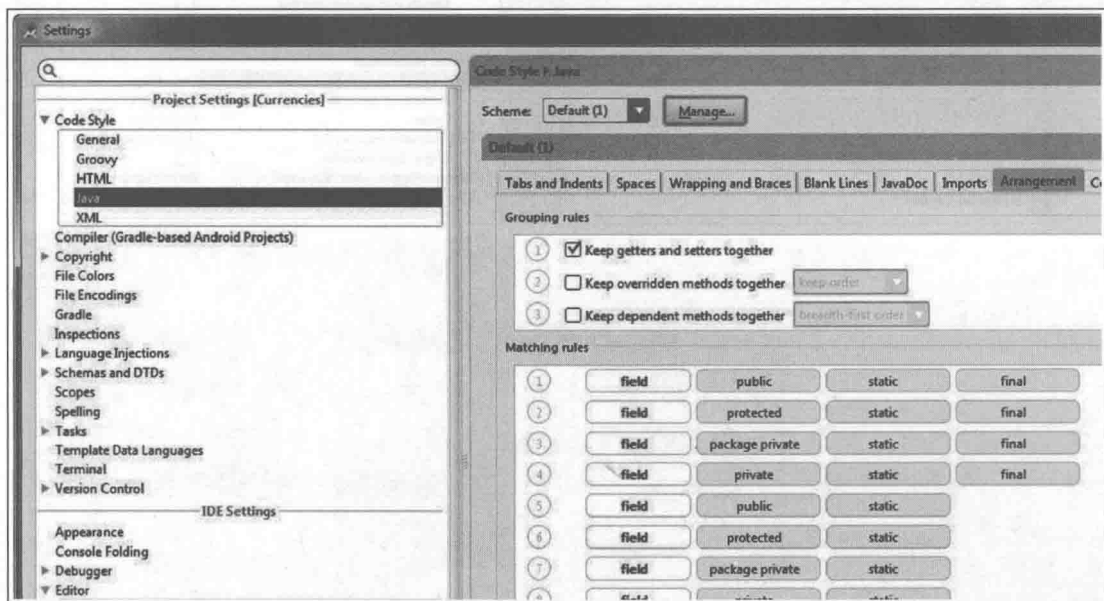


图 16-3 选中了 Rearrange entries 的 Reformat Code 对话框

选择 Wrapping and Braces 选项卡, 如图 16-4 所示。此选项卡的目的是让你设置包装代码的方式和时机。关于包装 Java 代码并没有硬性的规定。我们更喜欢将起始花括号置于代码块第一条语句的末尾, 而其他一些人则更喜欢起始和关闭花括号竖直对齐带来的对称性。如果更喜欢对齐的花括号, 那么可以在图 16-4 中突出显示的 Braces Placement 区域修改此设置(以及其他多个设置), 只需将设置从 End of Line 修改为 Next Line。当修改设置时, 注意右侧子面板中示例代码针对这些设置而发生的变化。

仔细探索 Code Style | Java 面板中的其他选项卡并根据自己的偏好定制代码风格。你不仅可以对 Java 应用代码风格, 还可以对 HTML、Groovy 和 XML 的代码风格应用类似的修改。XML 设置对于 Android 布局尤其有用。如果对 Code Style 设置满意, 你就可以通过单击位于 Code Style 面板顶部的 Manage 按钮来保存它们。如图 16-5 所示, 在出现名为 Code Style Schemas 的对话框中, 单击 Save As 按钮并为自己的代码风格起个名字, 例如 android1。

如果将代码风格修改应用到 **Default** 方案，那么它们将会成为所有项目的默认设置；如果将代码风格修改应用到 **Project** 方案，那么只有当前项目会受到影响。如你所见，Android Studio 在配置代码风格方面为你提供了极大的灵活性。

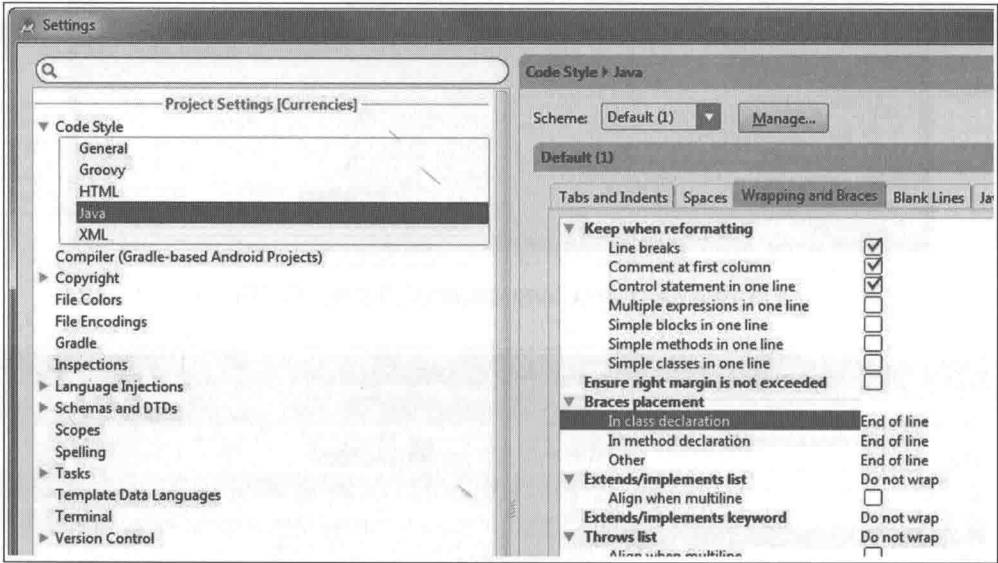


图 16-4 显示 Java | Wrapping and Braces 的 Settings 对话框

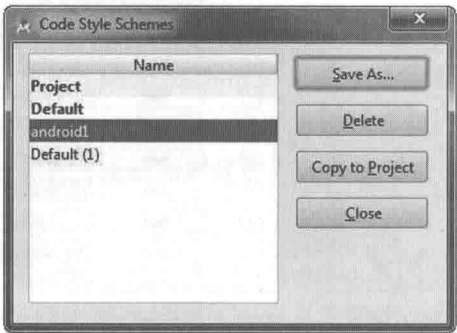


图 16-5 “代码风格方案” (Code Style Schemes)对话框允许你保存自己的代码风格

16.2 外观、颜色和字体

许多编程人员喜欢反转 IDE 的颜色主题，使背景成为黑色。有证据表明反色(黑色背景)主题可以缓解视疲劳，但适应反色主题需要花些时间，尤其是在你已经在白色背景下做了一段时间编码的情况下。Android Studio 自带三个预设主题：IntelliJ、Darcula 和 Windows。IntelliJ 是默认的光背景主题，Darcula 是暗背景主题，而 Windows 是复古 Windows 主题。你还可以从 ideacolorthemes.org 下载到更多主题。

通过按 **Ctrl + Alt + S** | **Cmd + 逗号** 打开 Settings 对话框。在过滤器栏中输入 **appearance**

并在列表中选择 Appearance 的第一个实例。将 Theme 字段修改为 Darcula。接着单击 Apply 和 OK 按钮，如图 16-6 所示。Android Studio 将要求自身重启，而你应该接受这个请求。一旦 Android Studio 重启完成，你的 IDE 应该会如图 16-7 所示。

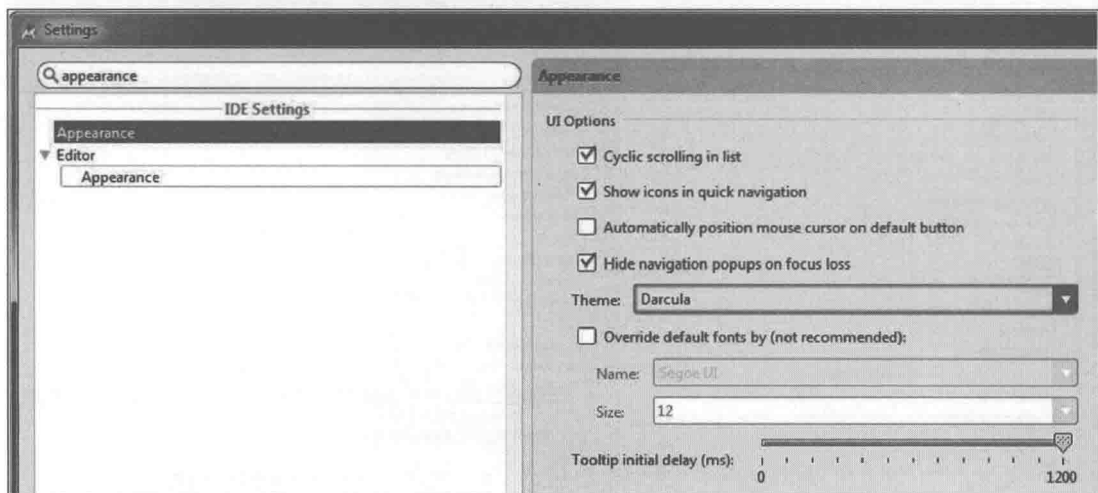


图 16-6 将 Appearance Theme 设置为 Darcula 的 Settings 对话框

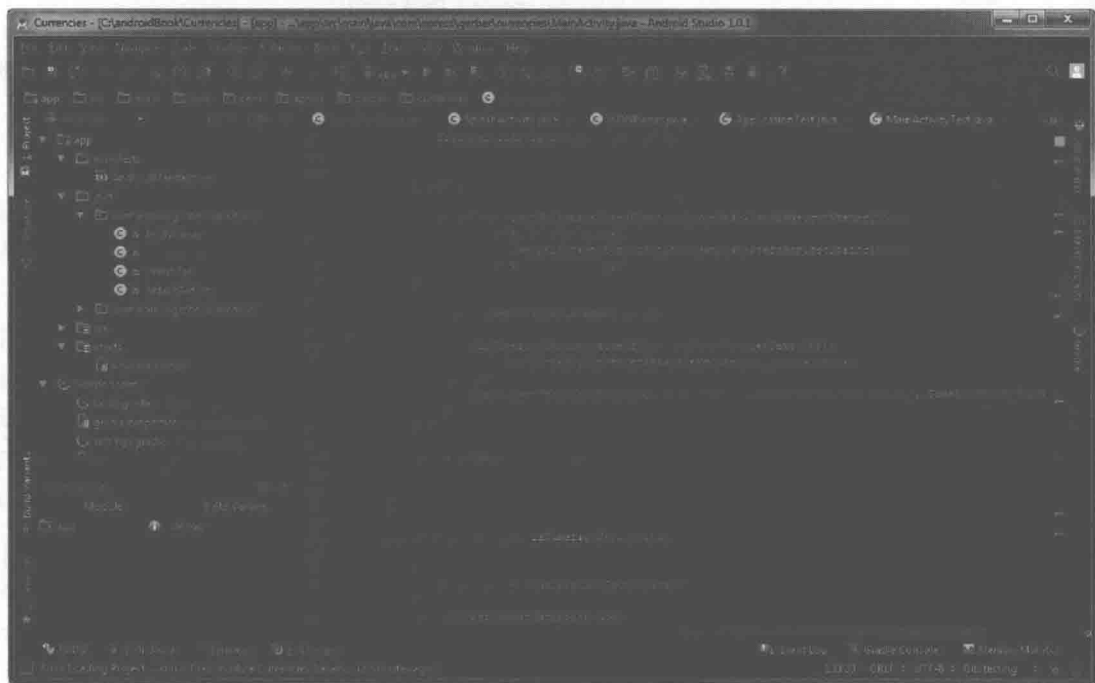


图 16-7 应用了 Darcula 主题的 Android Studio IDE

使用 Android Studio 的自带主题是修改颜色主题的最简单方法，但如果倾向于进一步自定义颜色和字体，可以通过调整 Editor | Colors and Fonts 下面的设置来实现，如图 16-8 所示。如果想要保存新的配色方案，那么可以通过单击 Save As 按钮并为配色方案设置名

称来完成。

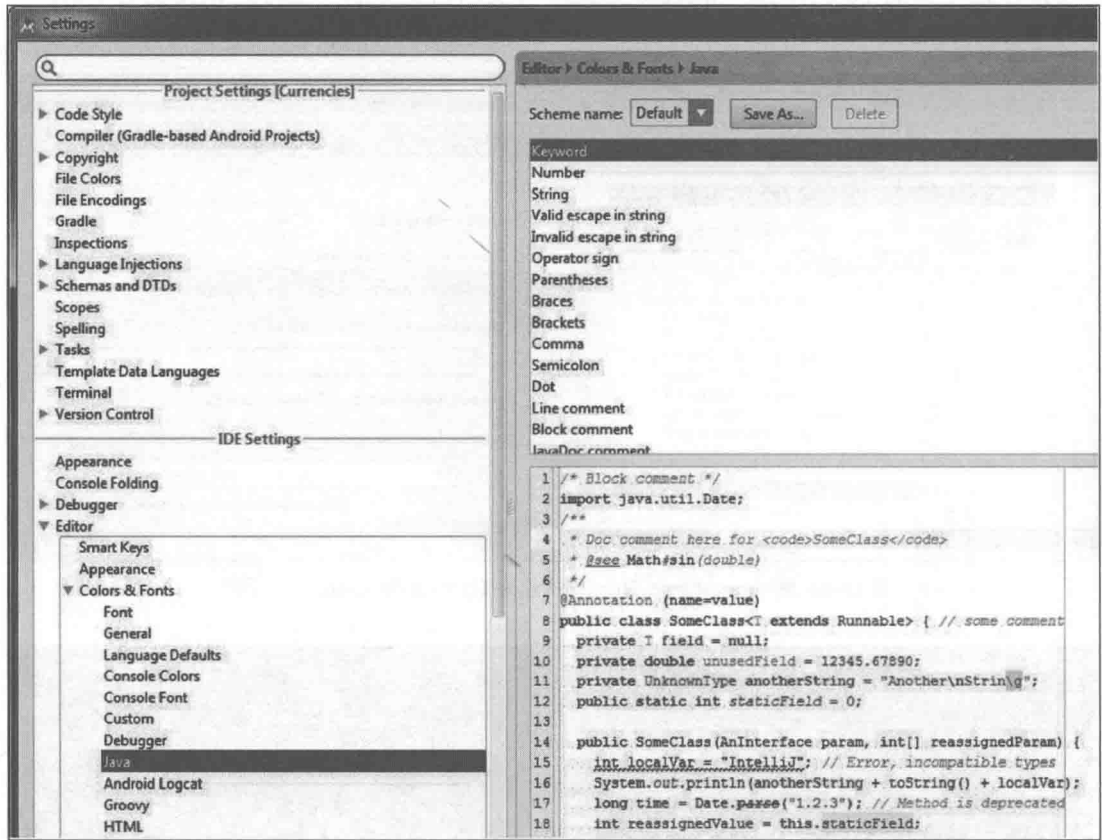


图 16-8 Settings | Colors and Fonts | Java

16.3 键盘映射

如果不确定某个键盘快捷键，那么可以通过导航至“帮助 | 默认键盘映射参考”来查看快速参考手册。这个命令会在浏览器中打开一个 PDF 文件，它位于 JetBrains 服务器上，网址为 jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard.pdf。

如果想要修改默认的键盘映射，可以这样做——激活 Settings 对话框(Ctrl + Alt + S | Cmd + 逗号)，然后在过滤器栏中输入 keymap。键盘映射条目根据菜单来组织，如图 16-9 所示。在开始自定义键盘映射之前，单击子面板顶部的 Copy 按钮，将 Default 中的所有键盘映射设置复制到一个新的键盘映射方案中，现在可以随意命名它。双击任意条目即可修改或添加你想要的任何键盘或鼠标快捷键，只要这些新的命令与已有的不冲突即可。如果经常使用某些没有快捷键的命令，那么可以为它们创建键盘或鼠标快捷键。

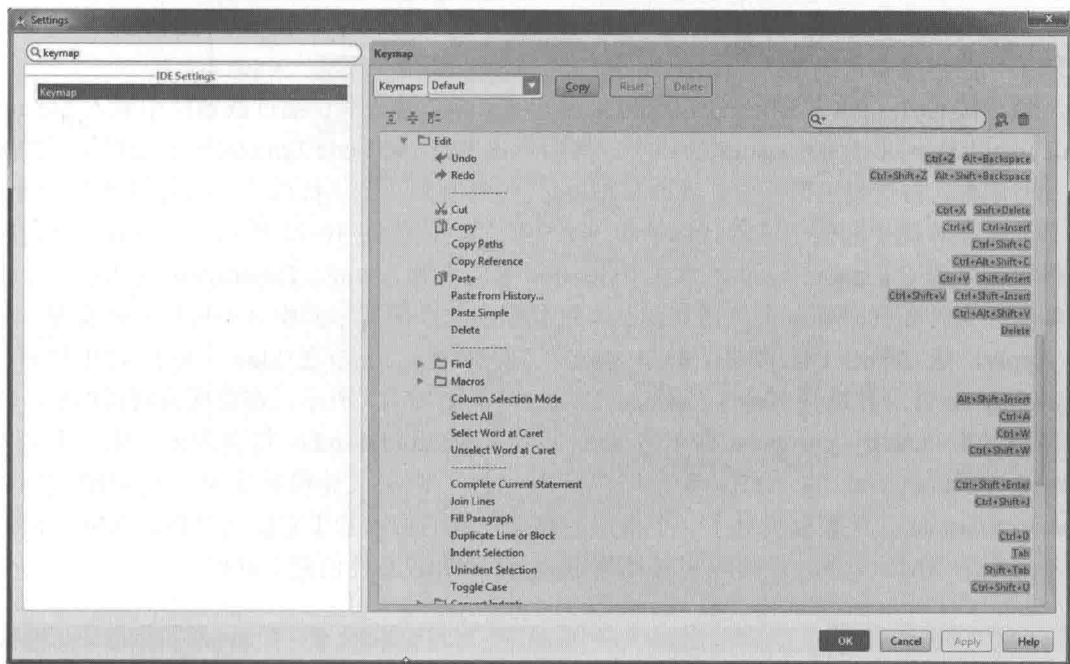


图 16-9 在 Settings | Keymap 中双击条目来修改或创建键盘快捷键

16.4 宏

在前面的 16.3 节中，当考虑已有的可用命令时，你可能在想除了用于激活命令的键盘或鼠标快捷键以外，是否存在一种定制命令自身的方法。有的——宏。宏让你能够记录 Android Studio 中任意一个或一系列事件并随意地回放它们。

让我们创建一个会发出 Monkey 命令的宏。单击 IDE 下边栏中的 Terminal 工具按钮，打开一个终端会话。导航到 Edit | Macros | Start Macro Recording。在终端会话中输入 `adb shell monkey -v 2000`。在状态栏的右下方，你将看到一条绿色消息，内容为“宏记录已启动”，如图 16-10 所示。单击左侧的红色停止按钮，在出现的对话框中输入 `monkey` 并单击 OK 按钮，如图 16-11 所示。现在可以通过导航至 Edit | Macros | Monkey 来选择这个宏。也可以采用前一节中介绍的指令为这个宏分配键盘或鼠标快捷键。

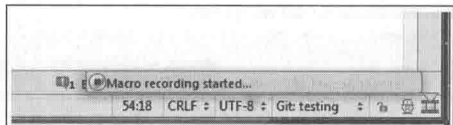


图 16-10 状态栏中的宏记录状态

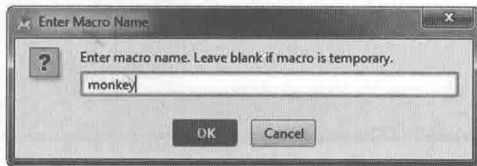


图 16-11 输入 monkey 作为宏的名称

16.5 文件和代码模板

文件和代码模板在创建频繁使用的文件或代码块时很有用。每次在 Android Studio 中

创建新的 Activity 时，文件模板都会用于生成其中的代码。在本节，你将基于第 9 章中已完成的代码创建自己的名为 CurrencyLayout 的代码模板。

通过按 Ctrl + Alt + S | Cmd + 逗号来激活 Settings 对话框。在过滤器栏中输入 file and code templates。单击 Templates 选项卡，然后单击 File and Code Templates 子面板顶部的绿色加号图标。将代码清单 9-1(在第 9 章)中的代码复制到右侧子面板中，或者如果正在阅读本书，就在右侧子面板中输入代码清单 9-1 中的代码，如图 16-12 所示。在 Name 字段中将模板命名为 CurrencyLayout，并在 Extension 字段中输入 xml。Description 文本框中介绍了如何在文件和代码模板中使用变量，不过我们在这个简单示例中并不使用任何变量。单击 Apply，然后单击 OK 按钮。在 Project 工具窗口中，右击(在 Mac 上按住 Ctrl 键并单击)res/layout 目录并选择 New | CurrencyLayout，如图 16-13 所示。在出现的对话框中，将文件命名为 activity_currency 并单击 OK 按钮。Android Studio 将会为你生成一个名为 activity_currency.xml 的 XML 新布局，其中包含了你在文件模板定义中使用的代码。CurrencyLayout 文件模板提供了一个很好的框架，我们可以基于它创建全新的 XML 布局。并不局限于 XML 文件，还可以轻松地 Java 或 HTML 文件创建文件模板。

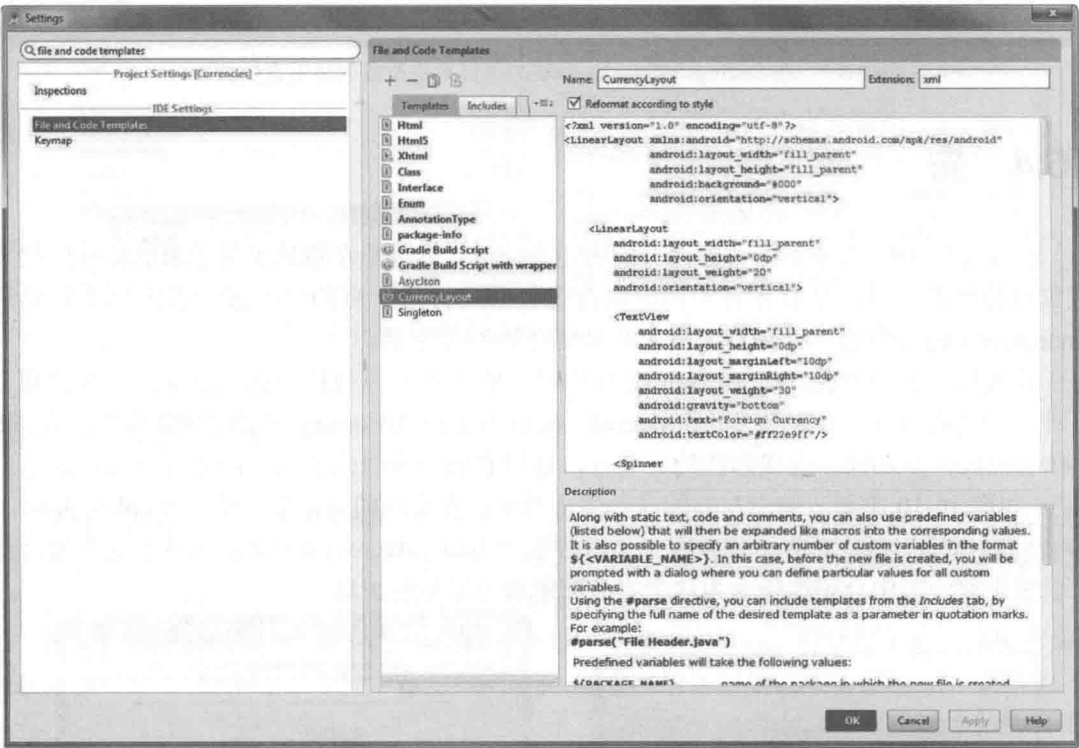


图 16-12 在 Settings | File and Code Templates 中创建 CurrencyLayout 文件布局

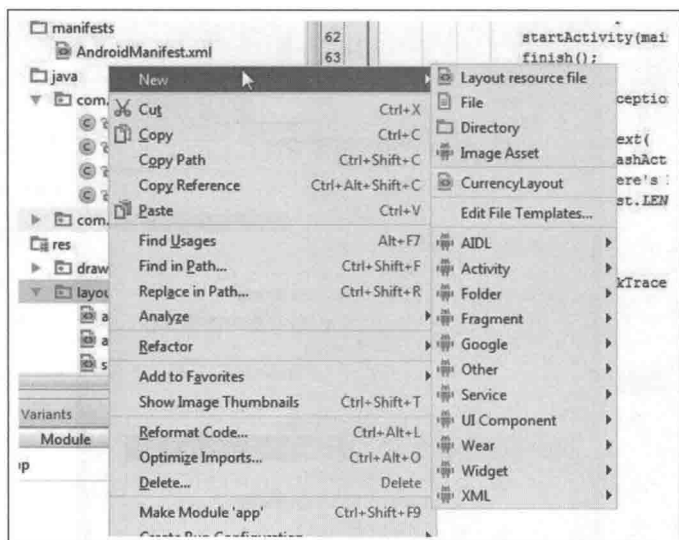


图 16-13 右击 res/layout 目录并选择 New | CurrencyLayout

16.6 菜单和工具栏

Android Studio 中很少有不能自定义的地方。如果不喜欢菜单和工具栏，也可以修改它们。在本节，你将向 Analyze 菜单中添加 Monkey 命令。

通过按 **Ctrl + Alt + S** | **Cmd + 逗号** 来激活 Settings 对话框。在过滤器栏中输入 **menus and toolbars**，并在列表中选择 **Menus and Toolbars**。从主菜单中选择 **Analyze | Analyze Actions**，展开相应的目录。选择 **Analyze Module Dependencies** 并单击 **Add After** 按钮，如图 16-14 所示。在出现的对话框中，导航至 **All Actions | Main menu | Edit | Macros | monkey**。如图 16-15 所示，单击 **OK** 按钮关闭该对话框并再次单击 **OK** 按钮以保存所做的修改。要查看并激活你刚刚所做的修改，请打开终端会话并导航至 **Analyze | Monkey**。

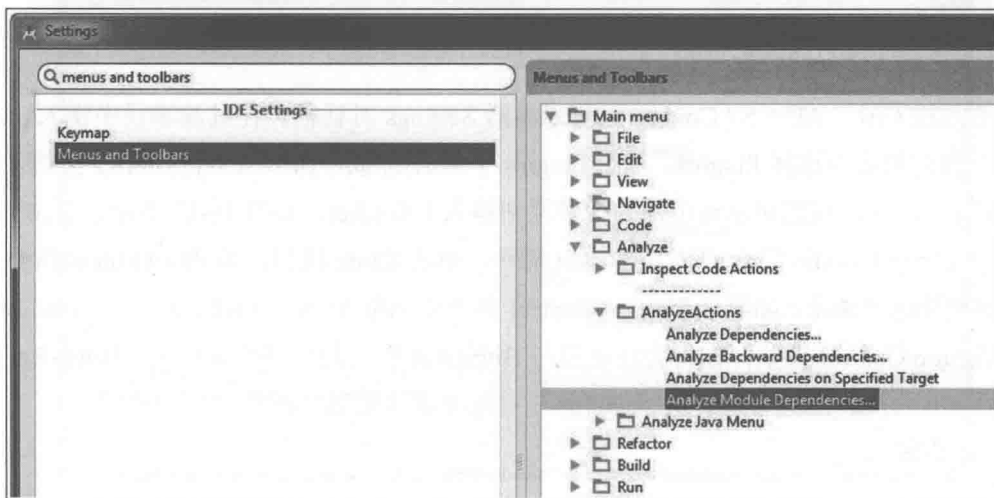


图 16-14 使用 Settings | Menus and Toolbars，在 Analyze 菜单中创建 Monkey 动作项

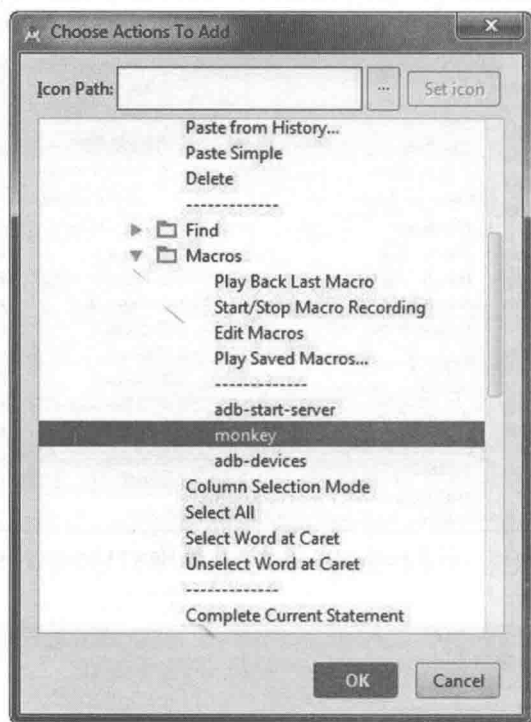


图 16-15 从 Macros 选项中选择 monkey

16.7 插件

有许多第三方插件可以与 Android Studio 配合使用。在本节，你将要安装 Bitbucket 插件，它是 Android Studio 的数百个插件之一。要查看插件的详细列表，请在浏览器中打开如下网址：

`plugins.jetbrains.com/?androidstudio`

通过按 `Ctrl + Alt + S` | `Cmd + 逗号` 来激活 Settings 对话框。在过滤器栏中输入 `plugins` 并在下面的列表中选择 `Plugins`。单击 `Plugins` 子面板底部的 `Browse repositories` 按钮，如图 16-16 所示。在出现的对话框顶部的搜索栏中输入 `bitbucket`，如图 16-17 所示。单击 `Install` 按钮，Android Studio 将会安装 Bitbucket 插件。单击 `Close` 按钮，Android Studio 将要求重启，请允许这个请求。如果从 `Version Control` 菜单中浏览 `VCS | Checkout` 以及 `VCS | Import into Version Control` 菜单，你将会注意到与 Bitbucket 相关的一些新菜单项。Bitbucket 插件方便了 Git 仓库的远程管理。关于 Git 的详细论述请参见第 7 章。

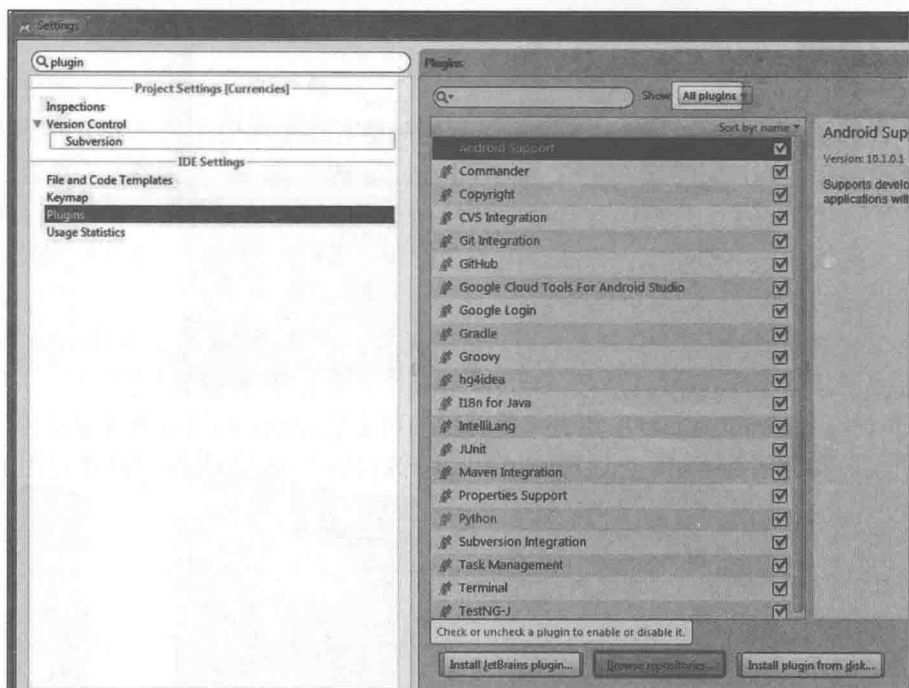


图 16-16 在 Settings | Plugins 中选择 Browse repositories

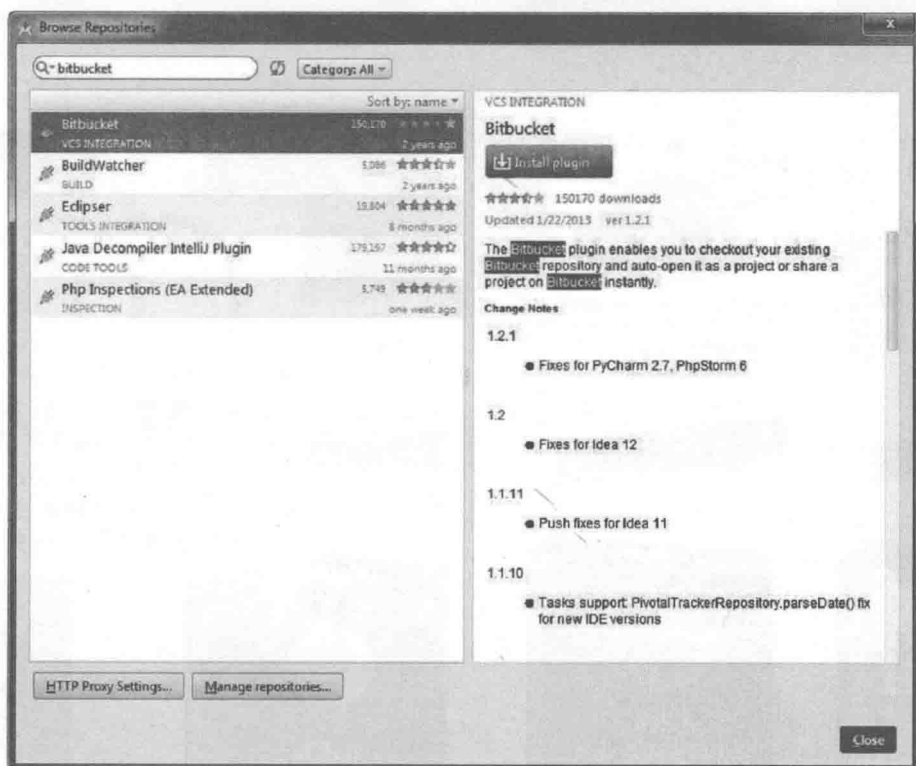


图 16-17 查找 Bitbucket 插件并安装它

16.8 小结

本章讲解了 Settings 对话框，还复习了代码风格，这是我们在第 3 章介绍的内容。你学习了如何保存代码风格方案和改变 Android Studio 的外观，以及调整已有主题并保存该主题的方法，还学习了如何修改键盘映射并保存自己的键盘映射方案。你创建了一个宏，并接着将该宏插入到菜单栏中。你还使用了文件和代码模板、修改了菜单和工具栏并在最后学习了插件的管理。

不要忘记我们已经介绍了定制工具按钮(第 2 章)、默认布局(第 2 章)和动态模板(第 3 章)的方法。尤其是动态模板，它是 Android Studio 中最重要的可定制特性之一。除了本书中涵盖的以外，Android Studio 中还有大量可用的可定制特性，而且许多可定制特性是可以自发现的。再次访问 Settings 对话框(Ctrl + Alt + S | Cmd + 逗号)并探索诸多可用的自定义特性。

《Android Studio实战 快速、高效地构建Android应用》全面涵盖关于Android Studio及其庞大工具生态系统的内容，包括Git和Gradle：除了介绍Android Studio与Git(用于源代码管理)和Gradle(一款构建及测试工具)的无缝工作方式外，还演示了如何使用诸如GitHub和Bitbucket的远程Git Web服务进行开发/协作。本书配有4个完整的Android项目，它们均可从公共的Git仓库下载。

通过学习本书，读者将能够掌握Android工具生态系统中最新、最实用的工具，以及Android App开发中的最佳实践。可以将实验代码作为模板或框架并在自己的类似App中重用和定制。

Android Studio是一款简单直观、功能丰富且极具包容性的集成开发环境，在开发Android App方面比Eclipse更加高效易用。有了这本书，你将快速掌握Android Studio并最大化Android开发时间。远程Web服务上的源代码均面向最新的Android Studio发行版——1.2版。

主要内容

- 如何开始使用Android Studio IDE
- 如何导航及使用Android Studio
- 如何使用Git进行版本控制
- 如何使用Gradle
- 如何使用崭新的Android Wear框架
- 如何使用Android Studio调试代码
- 如何管理应用项目
- 如何测试应用
- 如何分析并重构代码
- 如何定制Android Studio

清华大学出版社数字出版网站

WQBook  书文局泉
www.wqbook.com

Apress®
www.apress.com



源代码下载

ISBN 978-7-302-44153-3



9 787302 441533 >

定价：59.80元