



《Android 实验手册》

Java 课程组

版本 1.0

文档提供：Java 课程组

修改记录

目录

第 1 章 Android 概述	1
1.1 实驗一 JDK 下載與安裝	1
1.2 實驗二 Android SDK 下載與安裝	12
1.3 實驗三 Android Studio 下載與安裝	20
1.4 實驗四 安裝 Android 運行、調試環境	27
1.5 實驗五 Android Studio 建立一個 Android 應用	53
1.6 實驗六 LogCat 和 DDMS 的使用	63
第 2 章 Android 應用的界面基礎	73
2.1 實驗一 註冊、登錄頁面佈局	73
2.2 實驗二 界面佈局	80
2.3 實驗三 ListView 的使用	88
2.4 實驗四 通知消息和 Toast	101
2.5 實驗五 對話框的使用	110
2.6 實驗六 菜單的使用	114
2.7 作業	119
第 3 章 Android 事件處理機制	121
3.1 實驗一 基於回調機制的事件處理	121
3.2 實驗二 基於監聽機制的事件處理	124
3.3 實驗三 Android 中的多線程 Handler 方式實現	128
3.4 作業	131
第 4 章 Android 中的 Activity 的使用	132
4.1 實驗一 Activity 建立	132
4.2 實驗二 Activity 生命週期	142
4.3 實驗三 Activity 之間的跳轉及傳遞各種形式的參數（序列化）	147
4.4 作業	152
第 5 章 Android 中的 Intent	153
5.1 實驗一 Intent 的使用	153
5.2 實驗二 IntentFilter 的使用	166

5.3 实验三 Intent 启动系统组件	172
5.4 作业	180
第 6 章 Android 中的资源处理	181
6.1 实验一 同一个应用程序的主题样式	181
6.2 实验二 应用程序的不同分辨率支持	185
6.3 实验三 不同语言支持	189
6.4 作业	192
第 7 章 Android 中图形图像处理	193
7.1 实验一 基本绘图	193
7.2 实验二 在图层上加上一层 addlayout	197
7.3 实验三 通过动画实现启动页播放动画自动跳转.....	201
7.4 作业	205
第 8 章 Android 的数据操作和特殊 IO.....	208
8.1 实验一 Android 中 Sqlite 使用	208
8.2 实验二 Android 中的 Sharedpreferences.....	220
8.3 实验三 特殊输入的使用	227
8.4 作业	235
第 9 章 Android 中的 ContentProvider 的实现	236
9.1 实验一 Android 中 ContentProvider 使用	238
9.2 实验二 Android 中自定义 ContentProvider.....	253
9.3 作业	267
第 10 章 Android 中服务（Service）的使用	268
10.1 实验一 Android 中的 Service 的创建	268
10.2 实验二 Android 中的 Service 的绑定	277
10.3 实验三 Android 中的 Service 的创建	283
10.4 实验四 系统常见 Service 调用	288
10.5 作业	298
第 11 章 Android 中的广播机制（Broadcast）	299
11.1 实验一 Android 中的 Broadcast 发送	299
11.2 实验二 Android 中的 Broadcast 接收	302

11.3 实验三 发送自定义广播.....	312
11.4 作业	315
第 12 章 Android 中的网络应用	315
12.1 实验一 使用 Socket 进行通信	315
12.2 实验二 网络图片下载.....	322
12.3 实验三 使用 HttpClient 访问网络.....	325
12.4 作业	333
第 13 章 Android 中的数据格式解析	334
13.1 实验一 Android 中 xml 数据解析.....	334
13.2 实验二 掌握 Android 中 json 数据解析.....	350
13.3 作业	362
第 14 章 Android 中的多媒体应用	363
14.1 实验一 使用 MediaPlayer 播放音频.....	363
14.2 实验二 使用 SoundPool 播放音效.....	370
14.3 实验三 使用 MediaRecorder 录制音频.....	375
14.4 实验四 实现摄像头拍照功能	379
14.5 作业	387
第 15 章 Android 中的传感器应用开发	388
15.1 实验一 常见 Android 传感器的应用.....	388
15.2 实验二 实现手机摇一摇功能	391
第 16 章 Android 中的 GPS 应用开发	394
16.1 实验一 使用 GPS 定位当前位置	394
16.2 作业	401
附章 AndroidGoogleMap 服务	402
16.3 实验一 AndroidGoogleMap 服务	402

第1章 Android 概述

1.1 实验一 JDK 下载与安装

1.1.1 实验目的

1. 安装 JDK。
2. 熟练掌握 Android 环境的搭建过程。

1.1.2 准备工作

准备一： JDK 下载。

如果之前安装过 JDK 请跳过本步，直接进行下一步。如果没有安装过 JDK 的请从以下两种方式中任选一种进行下载。

注意：为了教学方便，我们统一安装 FTP 中的 JDK，从官网获得 JDK 的过程只供补充参考。

➤ 从 FTP 下载

请到 FTP 上下载 JDK 安装文件，地址是：<ftp://ftp.xxxxx>

➤ 从官方网站下载 JDK 并安装

步骤一 百度搜索关键字 “JDK 下载” ，进入 Oracle 官网(Java 原属于 Sun 公司，Sun 公司于 2009 年 4 月 20 被 Oracle 甲骨文公司收购，所以现在 Java 属于 Oracle 公司了)。

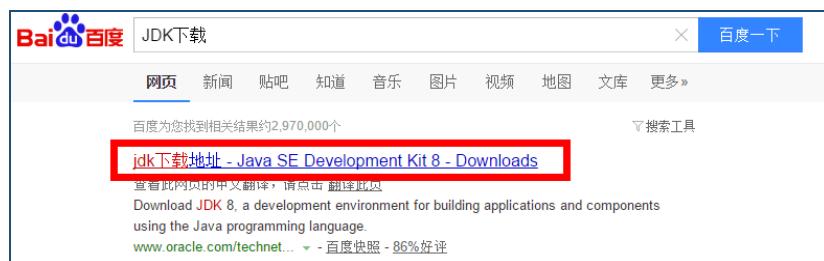


图 1.1.1

步骤二 进入官网后，下载最新版本的 JDK。如图 1.1.2。

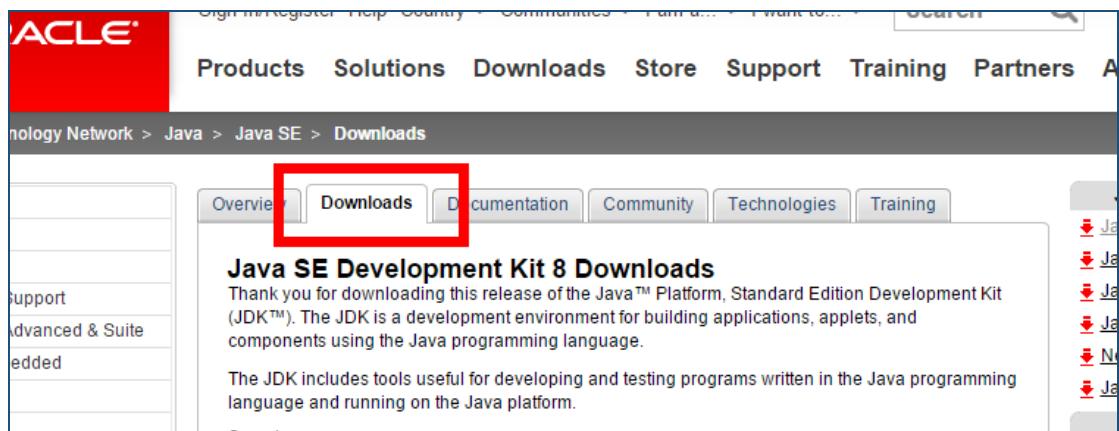


图 1.1.2

步骤三 进入下载选择界面，选择要下载的 JDK 版本，然后勾选 Accept License Agreement（接受服务条款），记得一定要把接受条款的单选按钮选中才能下载。然后根据自己计算机的操作系统，选择对应版本的 JDK，例如电脑是 Windows 64 位的，那么我就应该下载 Windows x64 版本的，如下图，下载到电脑中。

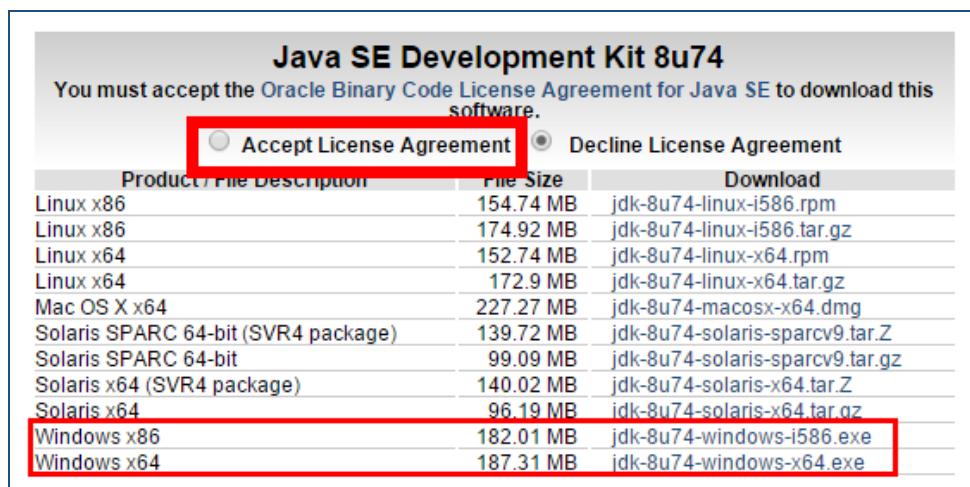


图 1.1.3

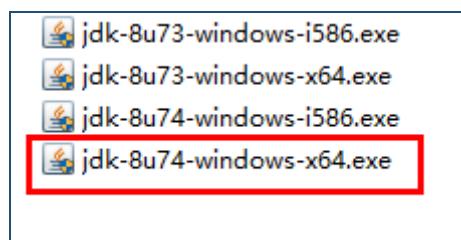


图 1.1.4

1.1.3 实验步骤

步骤一 下载完毕后双击该安装文件(图 1.1.5)，加载文件后出现以下安装页面，直接点击“下一步”。如图 1.1.6。

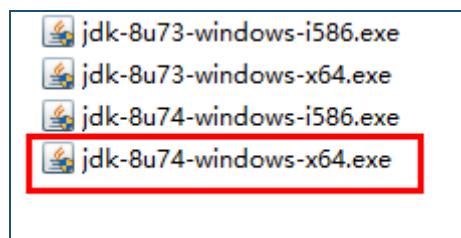


图 1.1.5



图 1.1.6

步骤二 点击“更改”选择安装目录。建议路径中不要出现中文，不要出现空格，然后点击“下一步”开始安装。

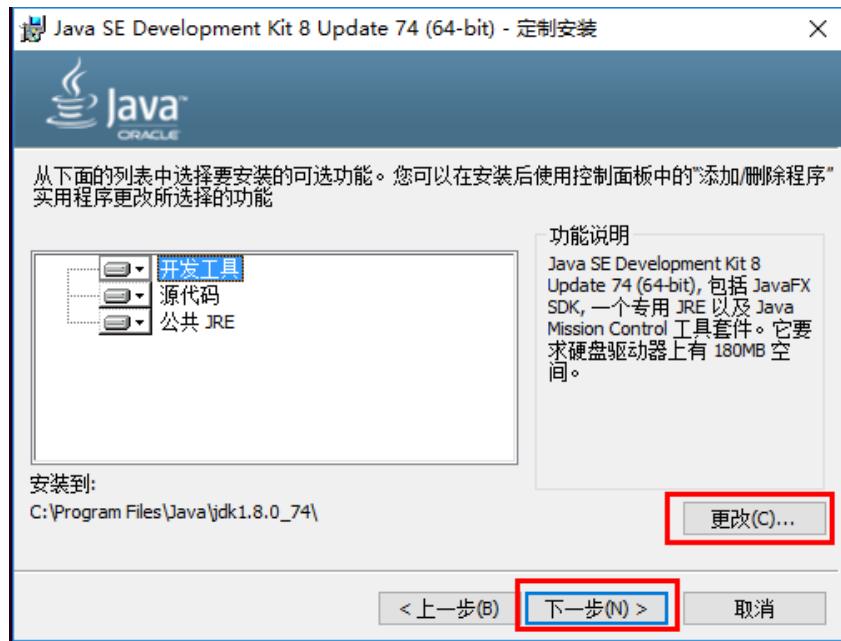


图 1.1.7

步骤三 点击“更改(A)”选择安装目录。建议路径中不要出现中文，不要出现空格，然后点击“下一步”开始解压安装。

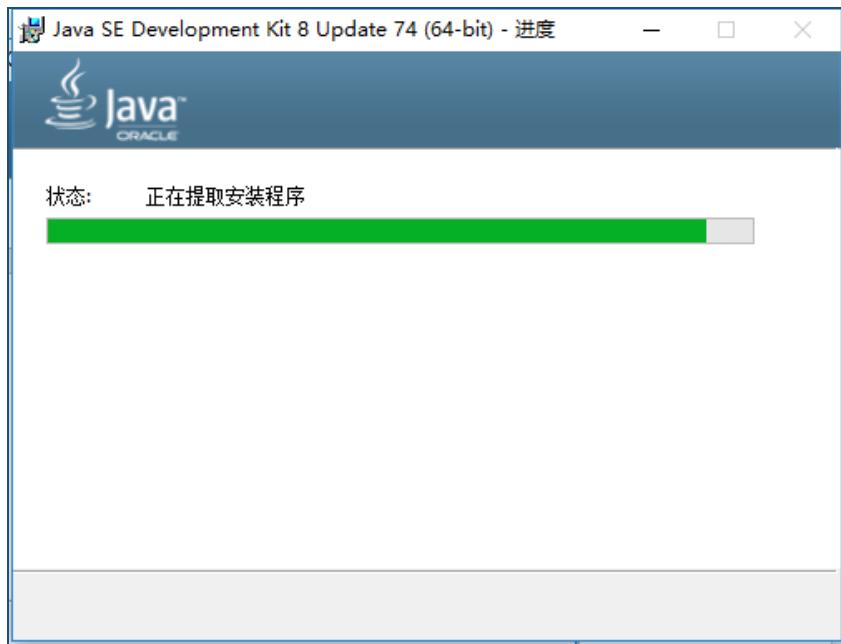


图 1.1.8

步骤四 最终完成安装会出现如图 1.1.9。



图 1.1.9

步骤五 然后我们到刚才设置的安装路径, 可以查看 JDK 的目录结构。如图 1.1.10。

电脑 > 本地磁盘 (C:) > Program Files > Java > jdk1.8.0_74		
名称	修改日期	类型
bin	2016/3/10 17:39	文件夹
db	2016/3/10 17:39	文件夹
include	2016/3/10 17:39	文件夹
jre	2016/3/10 17:39	文件夹
lib	2016/3/10 17:39	文件夹
COPYRIGHT	2016/1/29 18:11	文件
javafx-src.zip	2016/3/10 17:39	zip Archive
LICENSE	2016/3/10 17:39	文件
README.html	2016/3/10 17:39	Chrome HTML D...
release	2016/3/10 17:39	文件
src.zip	2016/1/29 18:11	zip Archive
THIRDPARTYLICENSEREADME.txt	2016/3/10 17:39	文本文档
THIRDPARTYLICENSEREADME-JAVAF...	2016/3/10 17:39	文本文档

图 1.1.10

步骤六 设置环境变量

需要设置三个环境变量: JAVA_HOME, PATH 和 CLASSPATH (不区分大小写)。

JAVA_HOME: 指向 JDK 的安装目录 (也就是我们解压后的目录)。

PATH: 指向 JDK 安装目录下的\bin 目录。

CLASSPATH: 指向 Java 程序的类路径。

步骤七 在桌面上“此电脑”处点击鼠标右键，选择“属性”，然后选择“高级系统设置”打开“系统属性”窗口，然后依次选择“高级”、“环境变量”。如图 1.1.12、图 1.1.13。

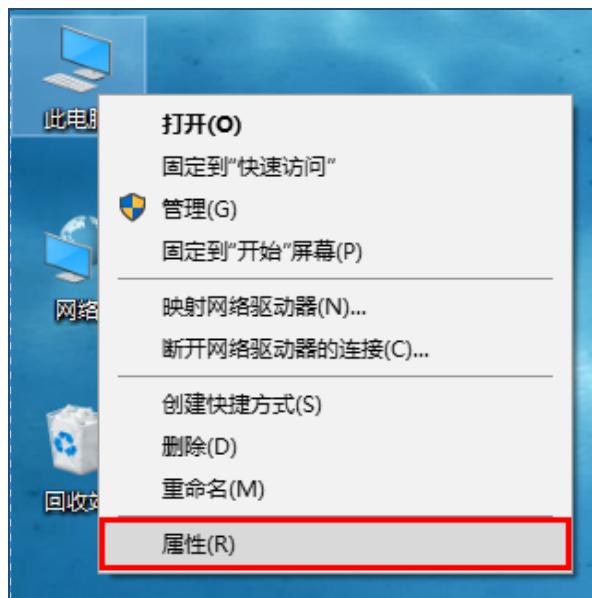


图 1.1.11



图 1.1.12

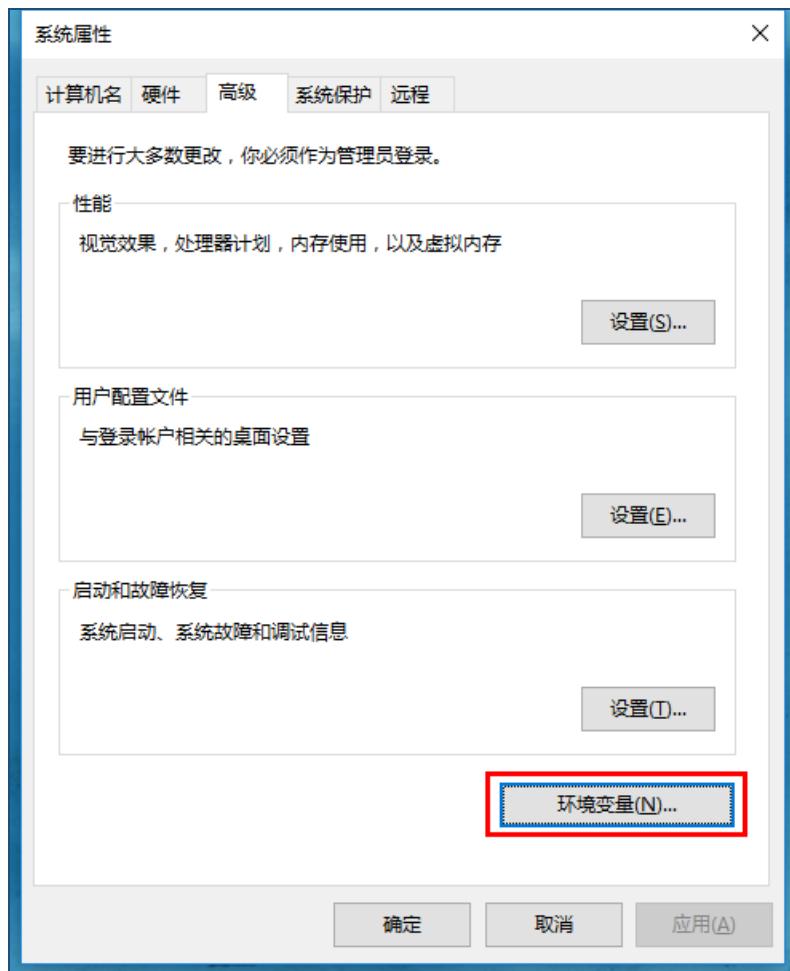


图 1.1.13

步骤八 进入“环境变量”设置窗口，可以点击下面的添加按钮，添加相关的环境变量，也可以对已经存在的环境变量进行删除和修改。在“系统变量”处点击“新建”按钮，弹出新建环境变量的对话框，添加 JAVA_HOME 环境变量，对应环境变量值为你刚才电脑安装 JDK 的路径，并点击“确定”。

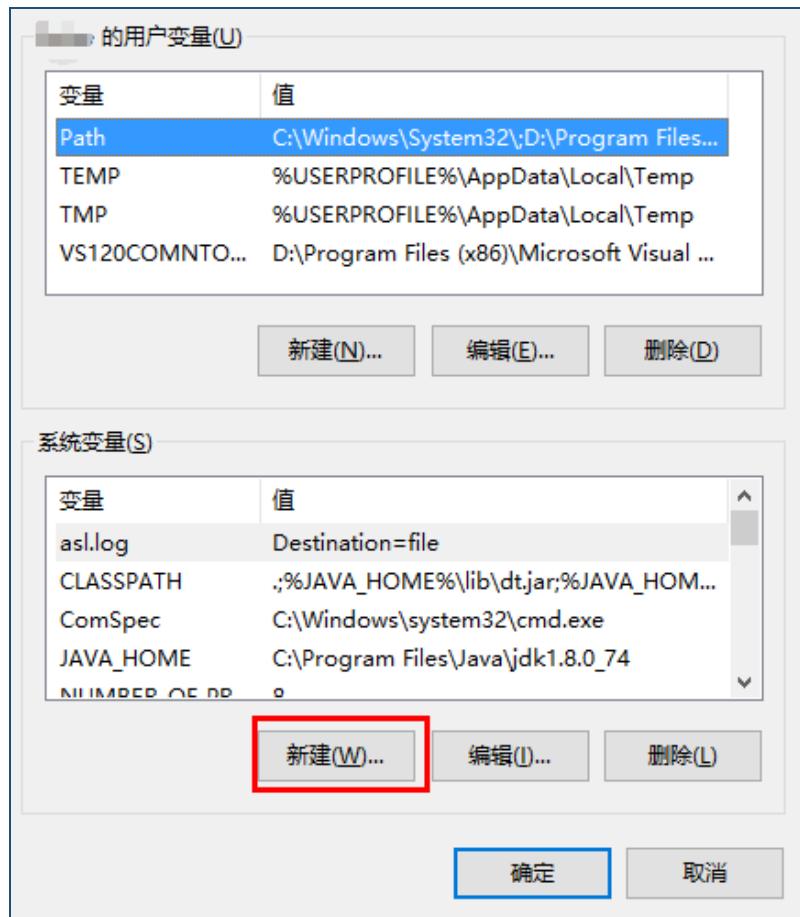


图 1.1.14

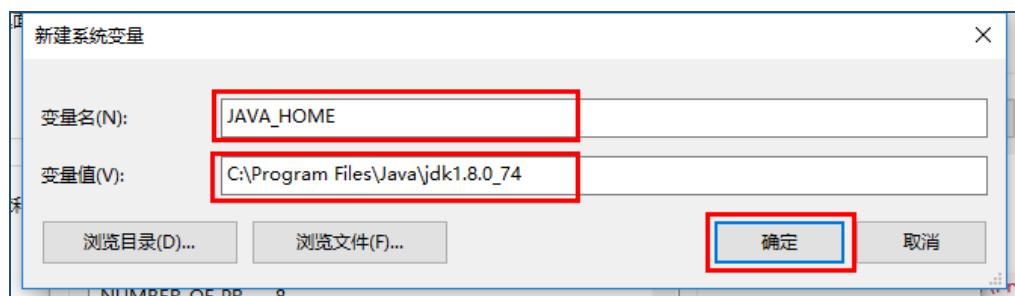


图 1.1.15

步骤九用同样的方法创建 CLASSPATH 环境变量，变量值输入如下值：`.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;`。输入完成后点击“保存”，使用分号将两个路径隔开。前面的“.”表示当前目录，意思就是在程序运行时，会先从当前目录寻找 Java 类，如果在当前路径下没有找到，则去 Java 安装目录下的 lib 文件夹里寻找，因为 lib 目录下放有 Java 的类库，你的程序中肯定会用到 Java 自带的类。

其中，%JAVA_HOME%则代表你之前配置的JAVA_HOME的环境变量值，这样，如果下一次JDK位置发生改变，只用修改JAVA_HOME的环境变量值即可。

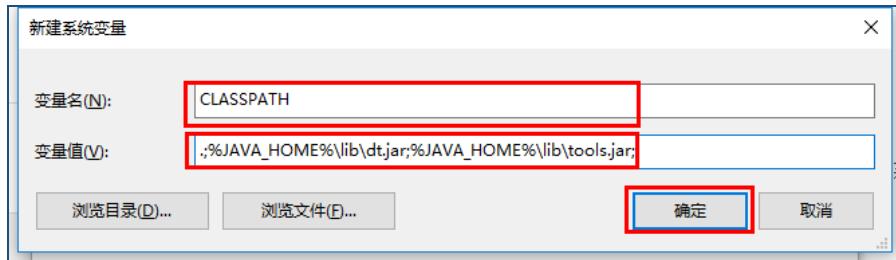


图 1.1.16

步骤十 我们要配的第三个环境变量是 PATH, 这个变量已经存在于系统中了，我们只需要在“系统变量”中找到它，然后点击“编辑”按钮，在后面加上 JDK 安装目录下的\bin 目录，在 Path 环境变量的末尾添加如下值: ;%JAVA_HOME%\bin (前面有个分号)。

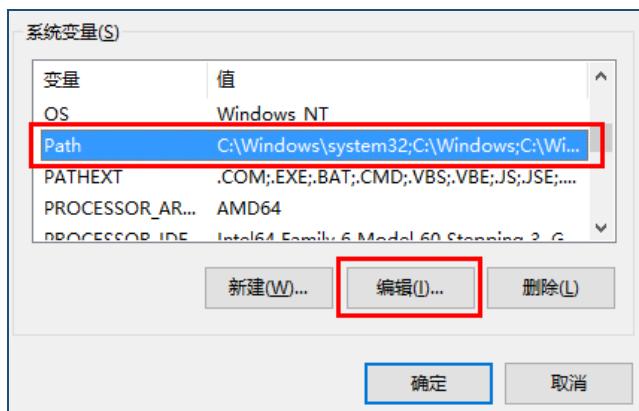


图 1.1.17

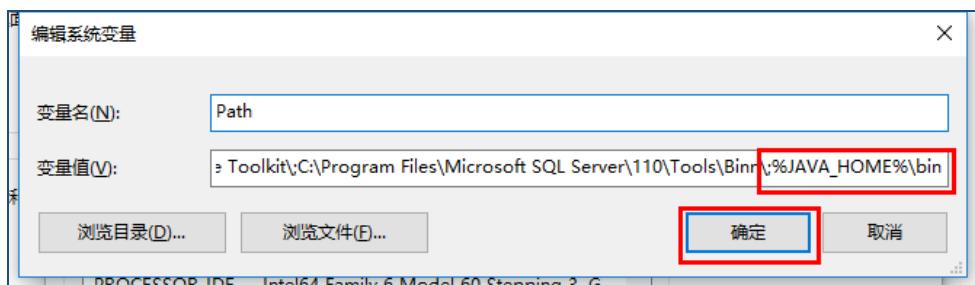


图 1.1.18

Windows10 用户点击“新建”按钮，然后输入 %JAVA_HOME%\bin 就可以了，然后点击“确定”按钮。

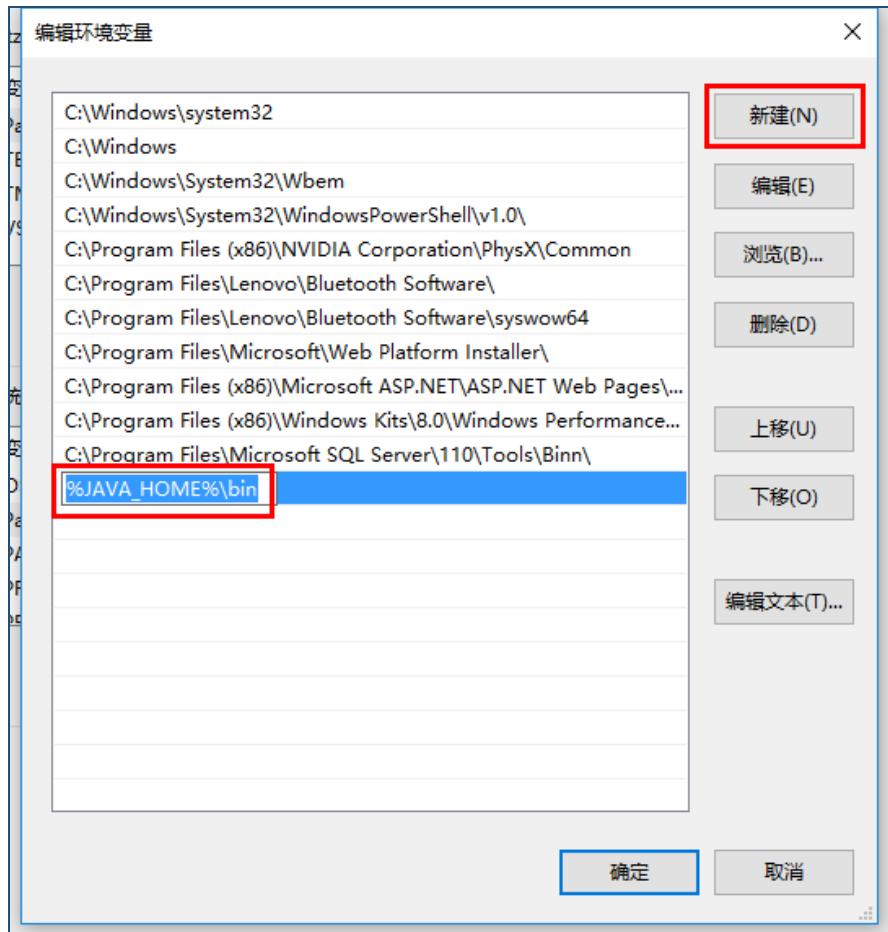


图 1.1.19

步骤十一 这样，系统的环境变量就设置好了。接下来测试 JDK 是否安装成功了。点击“开始”，然后选择“运行”，或者使用组合键“Windows”键+“r”键打开运行窗口，输入 cmd 打开命令行窗口，在命令行中输入“java -version”查看 JDK 的版本号，如出现 java version “1.8.0_74” 等版本信息，说明 JDK 已经正确的配置好了。

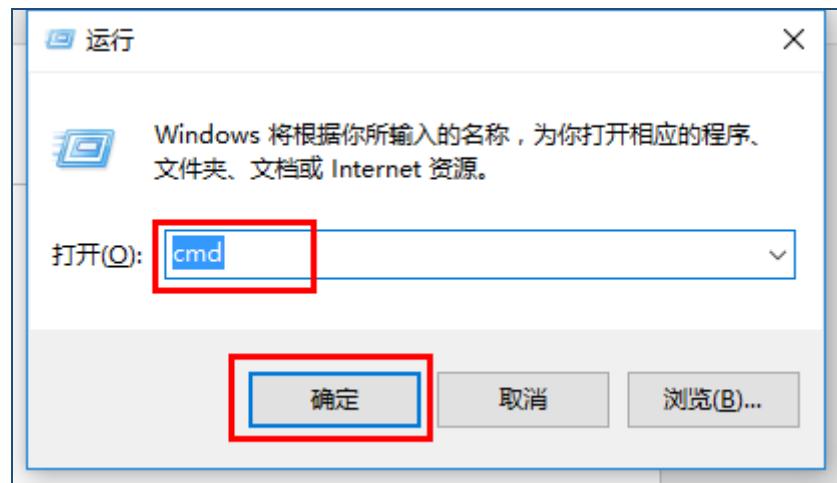


图 1.1.20

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.10586]
(c) 2016 Microsoft Corporation。保留所有权利。

C:\Users\[REDACTED]\>java -version
java version "1.8.0_74"
Java(TM) SE Runtime Environment (build 1.8.0_74-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.74-b02, mixed mode)
```

图 1.1.21

步骤十二 接下来我们就可以开发 Java 应用程序了。

1.1.4 实验结论

JDK 环境配置成功。

1.2 实验二 Android SDK 下载与安装

1.2.1 实验目的

1. 安装 Android SDK。
2. 熟练掌握 Android 环境的搭建过程。

1.2.2 准备工作

Android SDK 下载

Android SDK 请从以下两种方式中任选一种进行下载。

注意：为了教学方便，我们统一安装 FTP 中的 JDK，从官网获得 JDK 的过程只供补充参考。

➤ 从 FTP 下载

从 FTP 服务器拷贝，拷贝后解压即可，**注意：Android SDK 的解压路径不要带有中文以及空格。** 地址是：<ftp://ftp.xxxxx>

➤ 通过 Android SDK 官网下载 SDK

步骤一 打开 Android SDK 官方网站，网址为：

<http://developer.android.com/sdk/index.html>

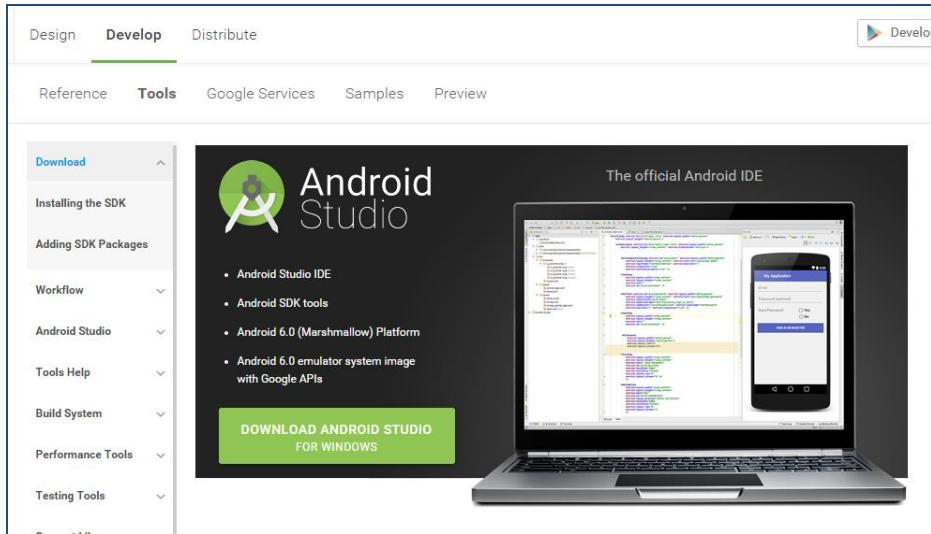


图 1.2.1

步骤二 将网页一直往下拉，找到 SDK Tools Only 的下载地址，如图 1.2.2。

Other Download Options

SDK Tools Only

If you prefer to use a different IDE or run the tools from the command line or with build scripts, you can instead download the stand-alone Android SDK Tools. These packages provide the basic SDK tools for app development, without an IDE. Also see the [SDK tools release notes](#).

Platform	Package	Size	SHA-1 Checksum
Windows	installer_r24.4.1-windows.exe (Recommended)	151659917 bytes	f9b59d72413649d31e633207e31f456443e7ea0b
	android-sdk_r24.4.1-windows.zip	199701062 bytes	66b6a6433053c152b22bf8cab19c0f3fef4eba49
Mac OS X	android-sdk_r24.4.1-macosx.zip	102781947 bytes	85a9cccb0b1f9e6f1f616335c5f07107553840cd
Linux	android-sdk_r24.4.1-linux.tgz	326412652 bytes	725bb360f0f7d04eaccff5a2d57abdd49061326d

图 1.2.2

步骤三 根据自己的操作系统选择相应的压缩包，点击相应的下载链接进行下载。

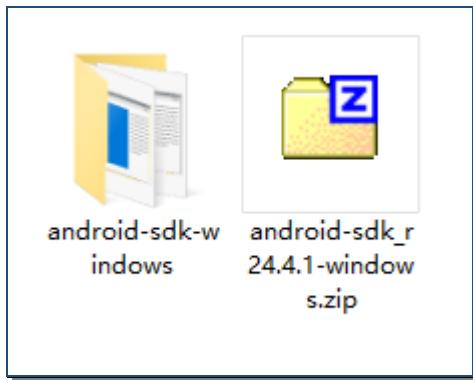


图 1.2.3

1.2.3 实验步骤

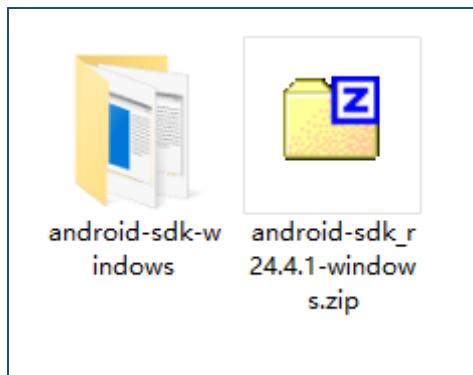


图 1.2.4

步骤一 将下载好的压缩文件进行解压缩，解压缩后文件夹里面文件目录如图 1.2.5。将解压缩后得到的文件夹拷贝到本地硬盘。**注意：路径中不要出现中文字符。**

- add-ons：该目录下存放第三方公司为 Android 平台开发的附加功能系统，刚解压缩时该目录为空。
- platforms：该目录下存放不同版本的 Android 系统。刚解压缩时该目录为空。
- tools：该目录下存放了大量的 Android 开发、调试的工具。
- AVD Manager. exe：该程序是 AVD（Android 虚拟设备）管理器。通过该工具可以管理 AVD。

- SDK Manager.exe: 该程序就是 Android SDK 管理器。通过该工具可以管理 Android SDK。

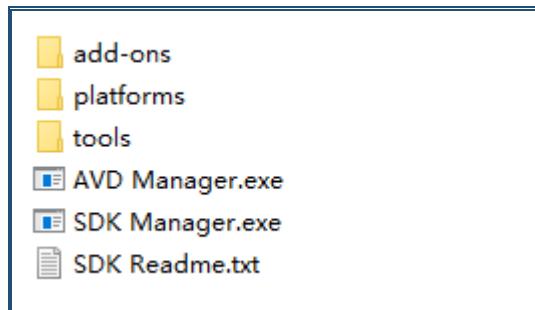


图 1.2.5

步骤二 Android SDK 下载与更新，双击启动图 1.2.5 中的 SDK Manager.exe，即可看到如图 1.2.6 所示的窗口。

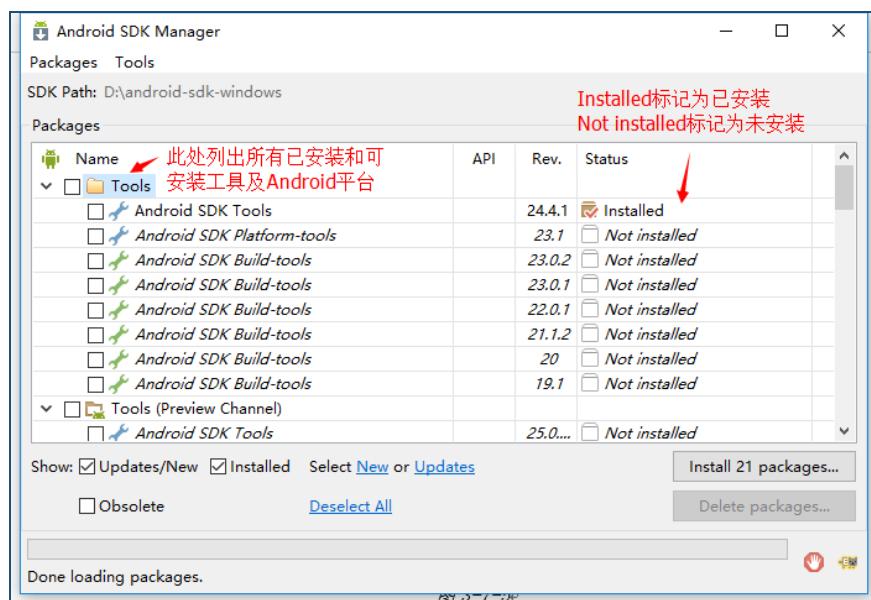


图 1.2.6

步骤三 在窗口的列表中勾选需要安装的平台和工具，比如 Android 6.0 的工具和平台，其中 Android 文档、SDK Platform 是必选的。如果想查看 Android 官方提供的示例程序，使用 Android SDK 的源代码，则可以勾选“Samples for SDK”和“Sources for Android SDK”两个列表项（最好将 Android 6.0 包含的所有工具都安装上，如果无需为 Android TV、可穿戴设备开发应用，则可暂时不勾选以 Android TV、Android Wear 开头的选项）。

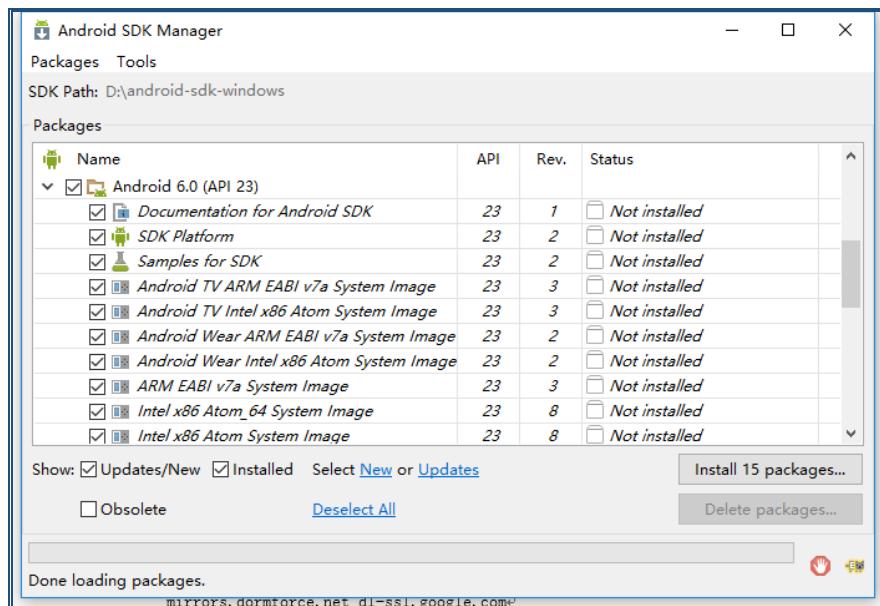


图 1.2.7

步骤四 选中需要安装的工具之后，单机“Install XX packages（XX代表用户勾选的列表项的数量）”按钮，将会出现如图 1.2.8 所示窗口。

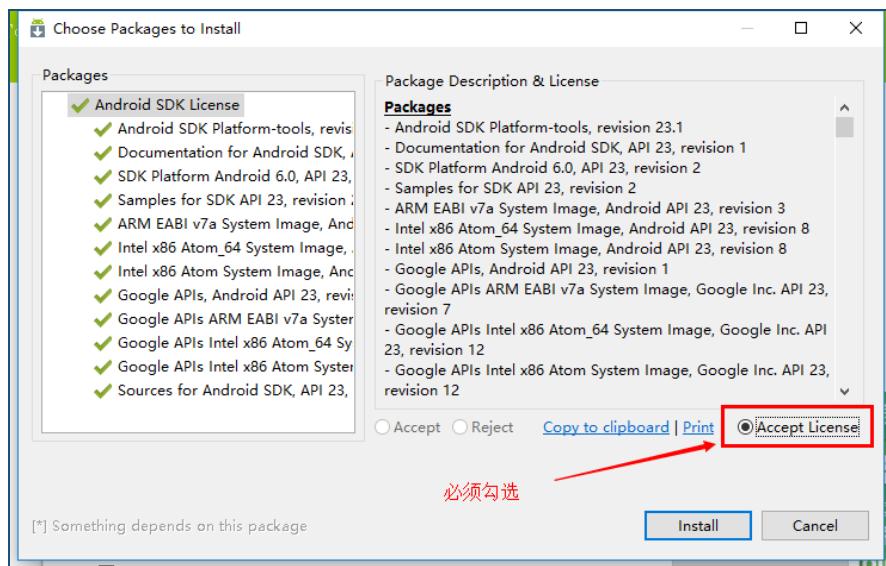


图 1.2.8

步骤五 勾选“Accept License”单选按钮，确认需要安装所有的工具包，然后单机“Install”按钮，系统开始在线安装 Android SDK 及相关工具。

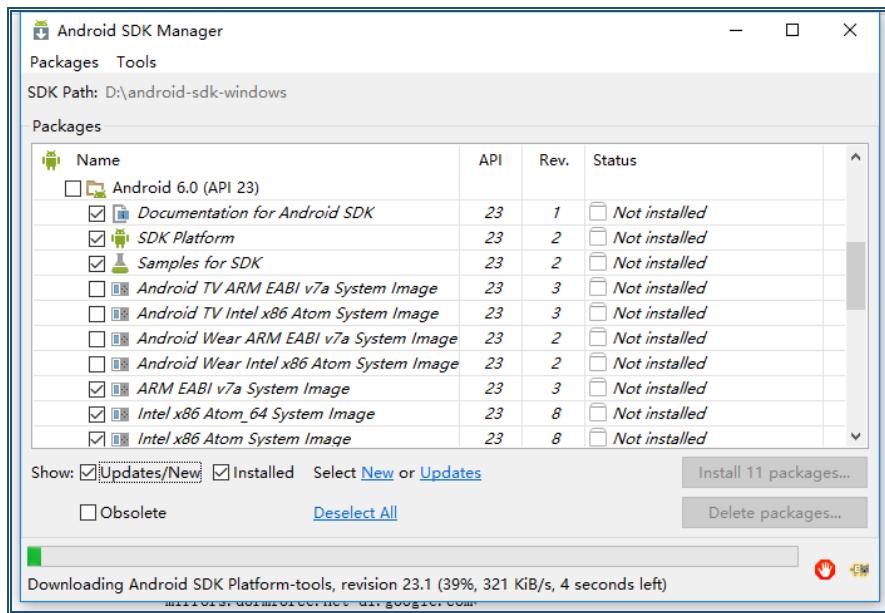


图 1.2.9

步骤六 由于默认连写的是境外谷歌的服务器进行下载，所以下载速度会非常慢，可以通过设置镜像服务器来加快下载速度。在如图 1.2.9 窗口中单击 Tools 菜单，选择 Options...，会弹出如图 1.2.10 窗口，在“HTTP Proxy Server”与“HTTP Proxy Port”中输入国内镜像站地址，下面列出了国内的 Android SDK 在线更新镜像服务器地址。

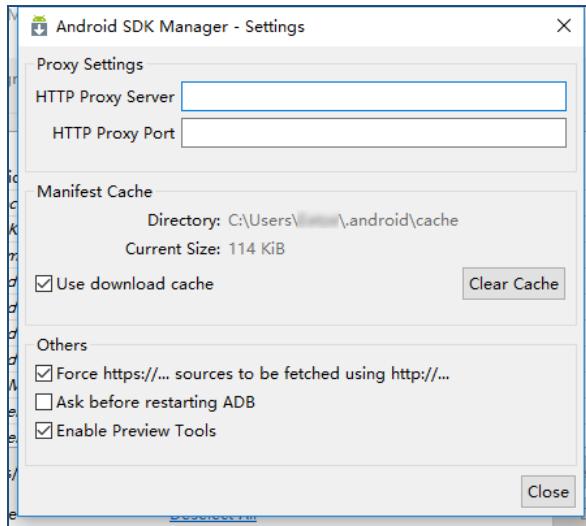


图 1.2.10

- 中国科学院开源协会镜像站地址：

IPV4/ IPV6: mirrors. opencas. cn 端口：80

IPv4/IPv6: mirrors.opencas.org 端口: 80

IPv4/IPv6: mirrors.opencas.ac.cn 端口: 80

- 上海 GDG 镜像服务器地址:

sdk.gdgshanghai.com 端口: 8000

- 北京化工大学镜像服务器地址:

IPv4: ubuntu.buct.edu.cn/ 端口: 80

IPv4: ubuntu.buct.cn/ 端口: 80

IPv6: ubuntu.buct6.edu.cn/ 端口: 80

- 大连东软信息学院镜像服务器地址:

mirrors.neusoft.edu.cn 端口: 80

- 腾讯 Bugly 镜像:

android-mirror.bugly.qq.com 端口: 8080

步骤七 输入要使用的镜像服务器地址及端口, 如图 1.2.11 所示, 然后点击“Close”关闭窗口, 此时下载时使用的就是国内镜像服务器了。

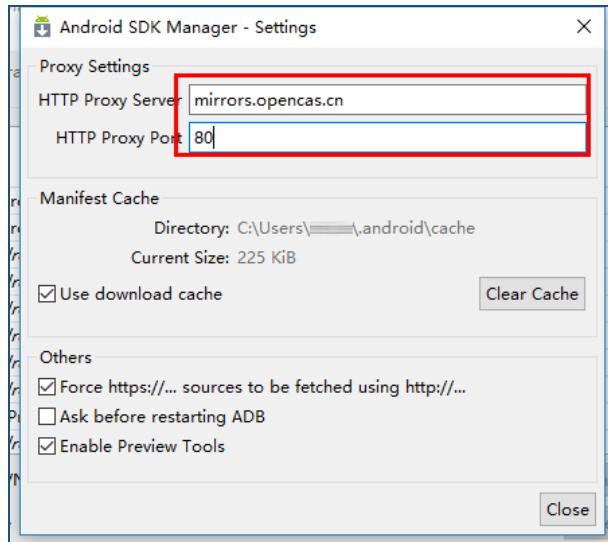


图 1.2.11

步骤八 当 Android SDK 下载完成之后，此时的 Android SDK 文件夹发生了变化，增加了如下几个文件夹，如图 1.2.12 所示。

- docs：该文件夹下存放了 Android SDK 开发文件和 API 文档等。
- extras：该文件夹下存放了 Google 提供的 USB 驱动、Intel 提供的硬件加速等附加工具包。
- platform-tools：该文件夹下存放了 Android 平台的相关工具。
- samples：该文件夹下存放了不同 Android 平台的示例程序。
- sources：该文件夹下存放了 Android 的源代码。
- system-images：该文件夹下存放了不同 Android 平台针对不同 CPU 架构提供的系统镜像。

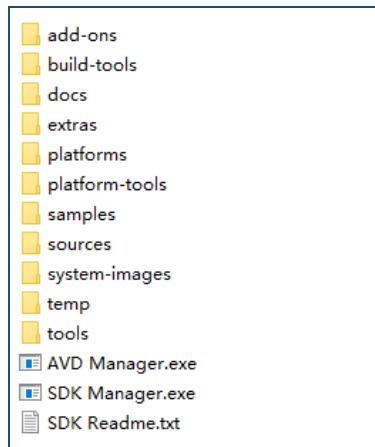


图 1.2.12

1.2.4 实验结论

Android SDK 安装成功。

1.3 实验三 Android Studio 下载与安装

Android Studio 是一项全新的基于 IntelliJ IDEA 的 Android 开发环境，类似于 Eclipse ADT 插件，Android Studio 提供了集成的 Android 开发工具用于开发和调试。当前 Android Studio 版本已经更新到 2.0，在安装 2.0 之前需要确保 JDK 已经安装并配置完成。

Android Studio 安装文件分为 Android Studio Bundle 版与 Android Studio IDE 版。Bundle 版中包含 IDE 与 SDK，IDE 版中只有 IDE，因为 Bundle 版中包含的 SDK 并不是最新的，通常我们在安装完 Bundle 版本的 Android Studio 后还要另外安装最新的 SDK，所以我们直接安装最新的 Android Studio IDE 版。

1.3.1 实验目的

1. 安装 Android Studio。
2. 熟练掌握 Android 环境的搭建过程。

1.3.2 准备工作

Android Studio 下载

Android Studio 请从以下两种方式中任选一种进行下载。

注意：为了教学方便，我们统一安装 FTP 中的 Android Studio 2.0。

➤ 从 FTP 下载

请到 FTP 上下载 Android Studio 压缩包：<ftp://ftp.xxxxx>。

➤ 从 Android Studio 官方网站进行下载

<http://developer.android.com/sdk/index.html>

➤ 从相关社区网站上获得

<http://www.androiddevtools.cn/>。

1.3.3 实验步骤

步骤一 下载 Android Studio 2.0 压缩文件到本地硬盘。

步骤二 将 Adnroid Studio 2.0 压缩文件进行解压，如图 1.3.1。

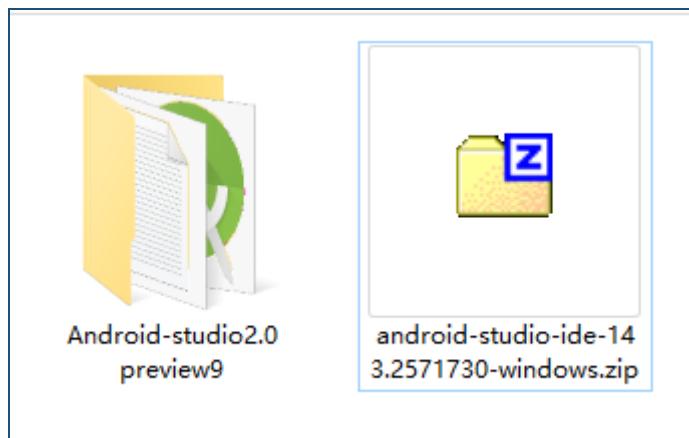


图 1.3.1

步骤三 下图为解压后的文件目录。

名称	修改日期	类型	大小
bin	2016/2/25 15:50	文件夹	
gradle	2016/2/25 15:50	文件夹	
lib	2016/2/25 15:50	文件夹	
license	2016/2/25 15:50	文件夹	
plugins	2016/2/25 15:50	文件夹	
build.txt	2016/1/29 20:59	文本文档	1 KB
LICENSE.txt	2016/1/29 21:00	文本文档	12 KB
NOTICE.txt	2016/1/29 21:00	文本文档	1 KB
uninstall.exe	2016/1/29 21:00	应用程序	2,298 KB

图 1.3.2

步骤四 将解压后的文件夹拷贝到要存放 Android Studio 2.0 的本地硬盘位置。

步骤五 双击 Android Studio2.0 文件夹里的 bin 文件夹双击 Studio.exe 或 Studio64.exe（分别对应 32 位系统与 64 位系统）启动 Android Studio。

名称	修改日期	类型	大小
appletviewer.policy	2016/1/29 21:00	POLICY 文件	1 KB
breakgen.dll	2016/1/29 21:00	应用程序扩展	33 KB
breakgen64.dll	2016/1/29 21:00	应用程序扩展	38 KB
focuskiller.dll	2016/1/29 21:00	应用程序扩展	37 KB
focuskiller64.dll	2016/1/29 21:00	应用程序扩展	43 KB
fsnotifier.exe	2016/1/29 21:00	应用程序	63 KB
fsnotifier64.exe	2016/1/29 21:00	应用程序	112 KB
idea.properties	2016/1/29 21:00	PROPERTIES 文件	2 KB
IdeaWin32.dll	2016/1/29 21:00	应用程序扩展	36 KB
IdeaWin64.dll	2016/1/29 21:00	应用程序扩展	42 KB
jumplistbridge.dll	2016/1/29 21:00	应用程序扩展	54 KB
jumplistbridge64.dll	2016/1/29 21:00	应用程序扩展	61 KB
log.xml	2016/1/29 21:00	XML 文档	3 KB
restarter.exe	2016/1/29 21:00	应用程序	52 KB
runnerw.exe	2016/1/29 21:00	应用程序	64 KB
studio.exe	2016/1/29 21:00	应用程序	867 KB
studio.exe.vmoptions	2016/1/29 21:00	VMOPTIONS 文件	1 KB
studio.ico	2016/1/29 21:00	图标	348 KB
studio64.exe	2016/1/29 21:00	应用程序	894 KB
studio64.exe.vmoptions	2016/1/29 21:00	VMOPTIONS 文件	1 KB
vistalauncher.exe	2016/1/29 21:00	应用程序	62 KB

图 1.3.3

步骤六 Android Studio 2.0 就安装完成了，运行界面如下。

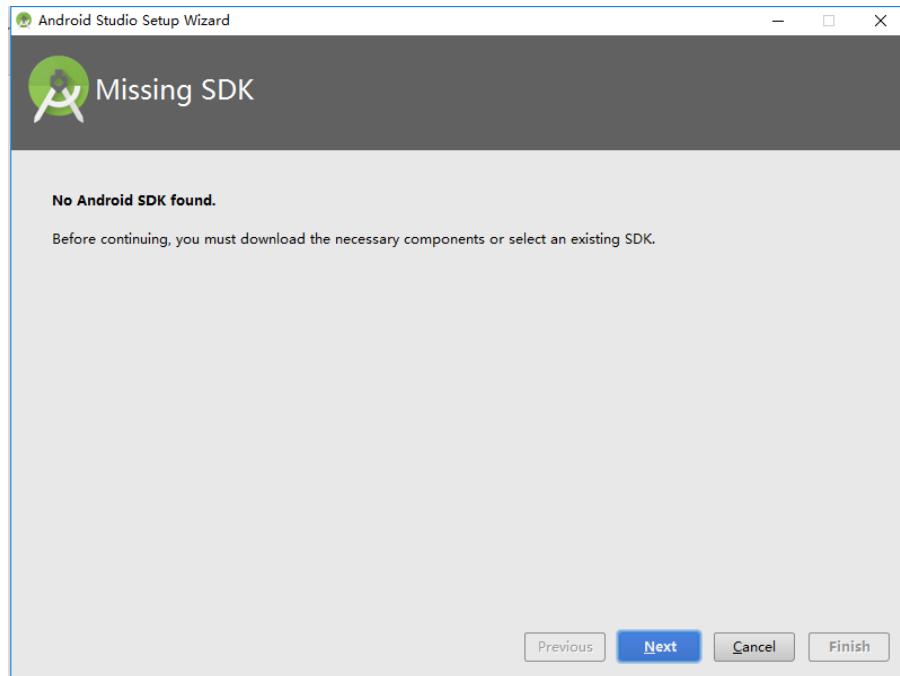


图 1.3.4

步骤七 Android Studio 中配置 SDK，因为安装的是 IDE 版本的 Android Studio，第一次运行 Android Studio 时需要配置 SDK 的本地路径，所以会出现如图 1.3.5 窗口。

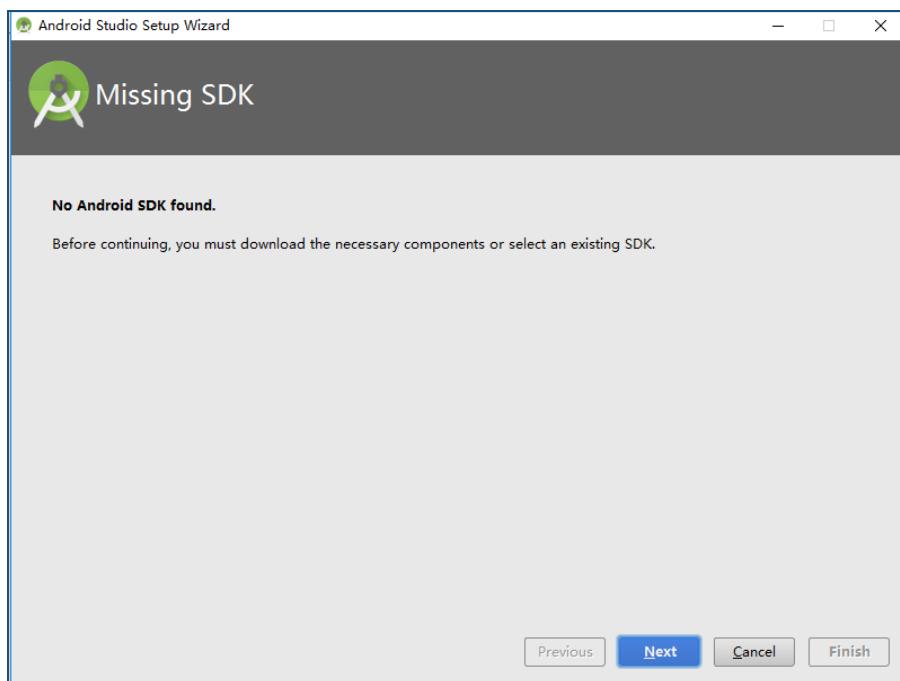


图 1.3.5

步骤八 点击“Next”按钮，然后再出现的窗口中点击“...”按钮来指定 SDK 存放的本地硬盘路径，修改好本地 SDK 路径之后点击“Next”按钮。

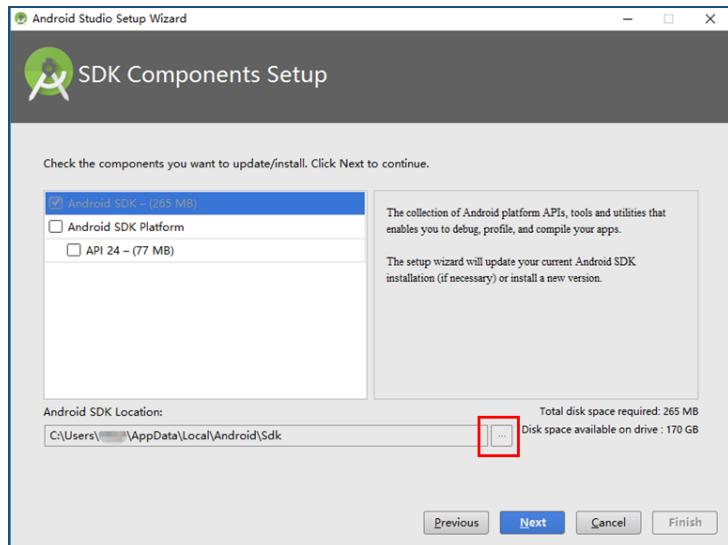


图 1.3.6

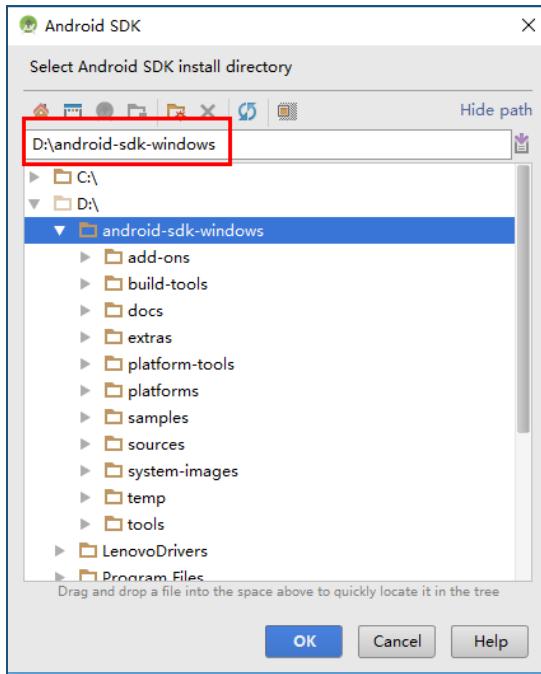


图 1.3.7

步骤九 在出现的窗口中会显示当前的一些配置信息，点击“Finish”按钮。

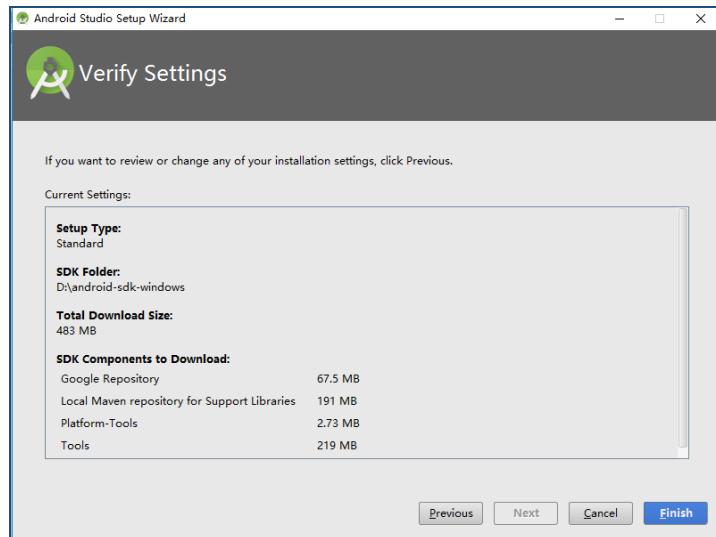


图 1.3.8

步骤十 然后 Android Studio 下载一些运行所必须的组件，耐心等待。

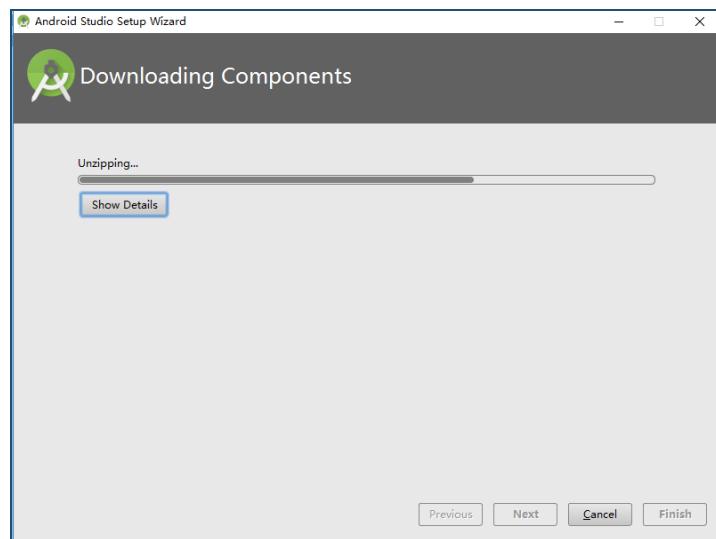


图 1.3.9

步骤十一 下载完成之后直接点击“Finish”按钮，出现如所图 1.3.11 示窗口，此时 Android Studio 中 SDK 已经配置完成。

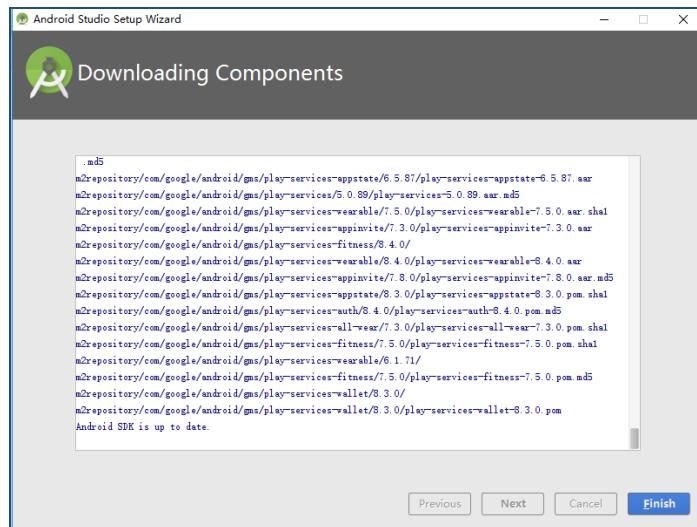


图 1.3.10

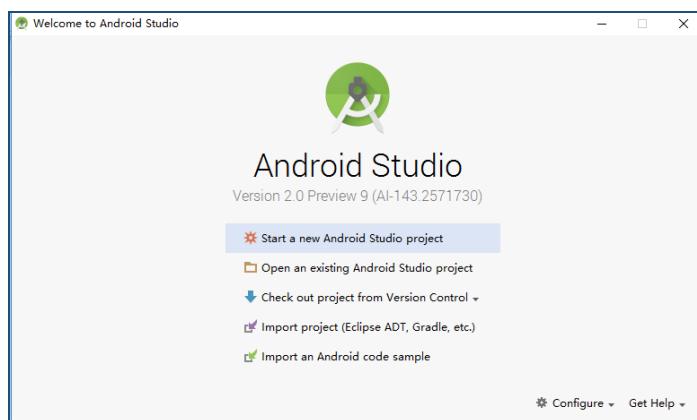


图 1.3.11

1.3.4 实验结论

Android Studio 安装成功。

1.4 实验四 安装 Android 运行、调试环境

Android 程序必须在 Android 手机上运行，因此 Android 开发时必须准备相关的运行、调试环境。准备 Android 程序的运行、调试环境有如下 3 种方式。

- Android 系统的真机。
- Android 虚拟设备（即 AVD）。
- 第三方提供的 Genymotion 模拟器。

1.4.1 实验目的

1. 熟练掌握 Android 运行、调试环境。
2. 熟练掌握 Android 环境的搭建过程。

1.4.2 准备工作

Genymotion 模拟器下载

Genymotion 模拟器请从以下两种方式中任选一种进行下载。

注意：为了教学方便，我们统一从 FTP 中下载 Genymotion 模拟器。

➤ 从 FTP 下载

请到 FTP 上下载 Genymotion 模拟器安装包：<ftp://ftp.xxxx>。

➤ 从 Genymotion 官方网站进行下载

步骤一 打开 Genymotion 官网 <https://www.genymotion.com/>，下载 Genymotion 需要登录，所以点击右上角的“Sign in”跳转到登录页面。

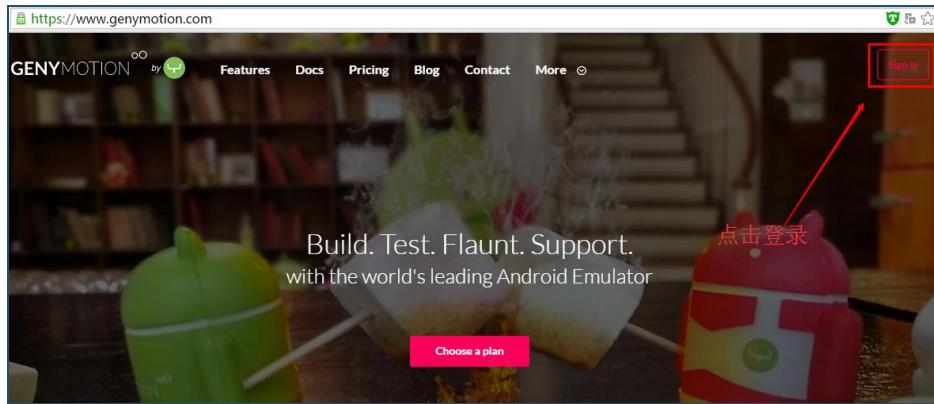


图 1.4.1

步骤二 如果有账号的话可以在登陆页面进行登录，如果没有账号点击“Create an account”进行注册，如图 1.4.3 所示。注册时需输入一个有效的邮箱用来激活账号，输入完个人信息后点击“Create an account”，然后登录注册时的邮箱完成激活，就可以使用账号进行登录了。

A screenshot of the Genymotion sign-in page. The page has a dark header with navigation links: Features, Docs, Pricing, Blog, Contact, More. The main area is titled "Sign in". It contains two input fields: "Username" and "Password". Below the password field is a link "Forgotten your password?". At the bottom, there are two buttons: "Sign in" (in a pink box) and "Create an account" (in a pink box). There is also a checkbox for "Remember me on this computer". A red arrow points from the text "注册" (Register) to the "Create an account" button.

图 1.4.2

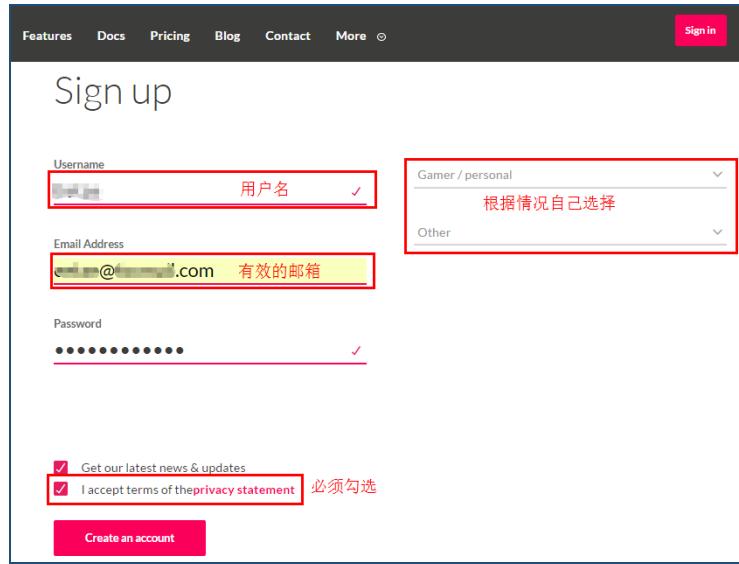


图 1.4.3

步骤三 登录之后返回官网的首页 <https://www.genymotion.com/>, 网页往下拉找到“Get Genymotion now”，如图 1.4.4 所示，Genymotion 提供企业版与个人版，选择个人免费基础版，点击“Get Started”跳转到下载页面。

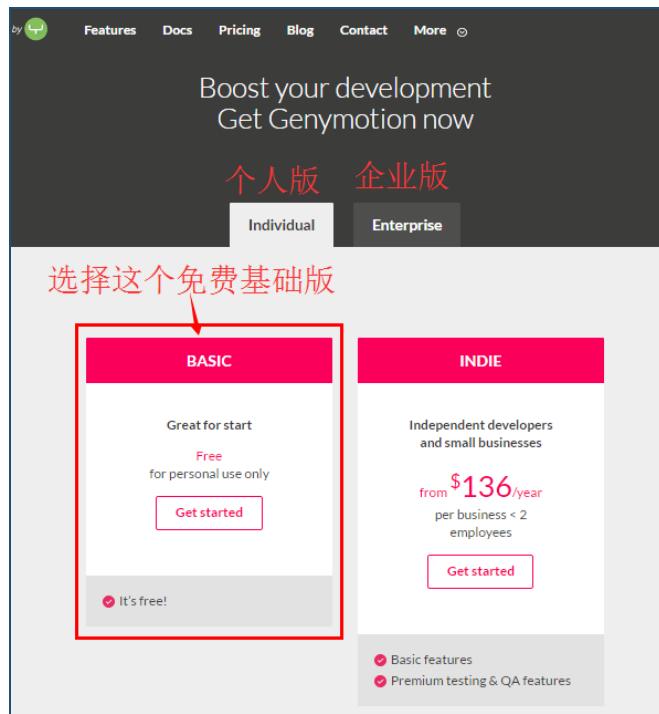


图 1.4.4

步骤四 在下载页面选择带 VirtualBox 的版本进行下载，然后等待下载完成。

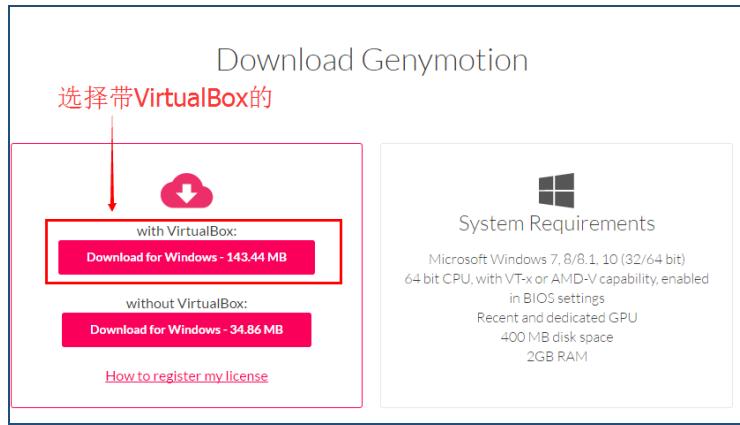


图 1.4.5

1.4.3 实验步骤

➤ 真机作为运行、调试环境

步骤一 用 USB 连接线将 Android 手机连接到电脑上。

步骤二 在电脑上为手机安装驱动，不同手机厂商的 Android 手机的驱动略有差异，请登录各手机厂商官网下载手机驱动。

步骤三 打开手机的调试模式。打开手机，依次单击“Dev Tools”、“开发者选项”进入设置界面，在该界面中“开启”开发者选项。

步骤四 勾选“Always stay awake”、“USB 调试”、“允许模拟定位”三个选项即可。如果还有其他需求，则可以勾选其他的开发者选项。

➤ AVD 作为运行、调试环境

Android SDK 为开发者提供了可以在电脑上运行的“虚拟手机”，Android 把它称为 Android Virtual Device (AVD)。如果开发者没有 Android 手机，则完全可以在 AVD 上运行编写的 Andorid 应用。

步骤一 创建、删除和浏览 AVD 之前，通常应该先为 Android SDK 设置一个环境变量：ANDROID.SDK.HOME，该环境变量的值为磁盘上一个已有的路径。如果

不设置该环境变量，开发者创建的虚拟设备默认保存在 C:\Documents and Settings\<user_name>\.android 目录下。如果设置了 ANDROID_SDK_HOME 环境变量，那么虚拟设备就会保存在%ANDROID_SDK_HOME%\ . android 路径下。

此处设置的 ANDROID_SDK_HOME 环境变量并不是 Android SDK 的安装目录，而是为了指定 AVD 的存放目录。

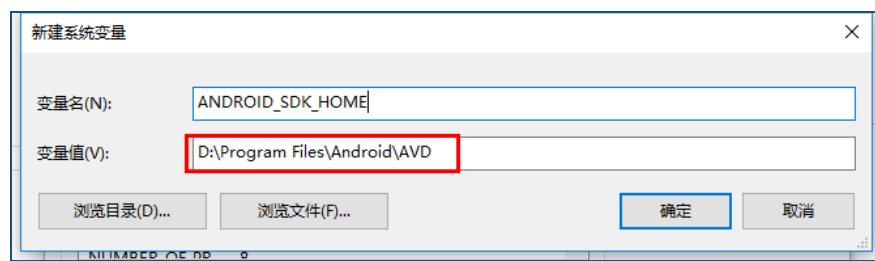


图 1.4.6

步骤二 Android SDK 安装目录下的 AVD Manager. exe 启动 AVD 管理器。

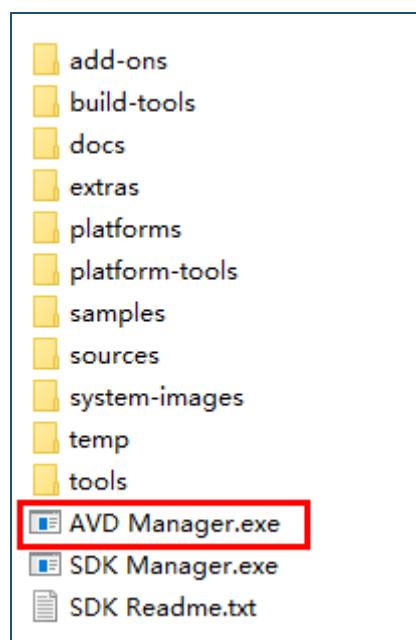


图 1.4.7

步骤三 如图 1.4.8 所示，单机管理器左边的“Android Virtual Devices”选项卡，管理器列出当前已有的 AVD 设备。

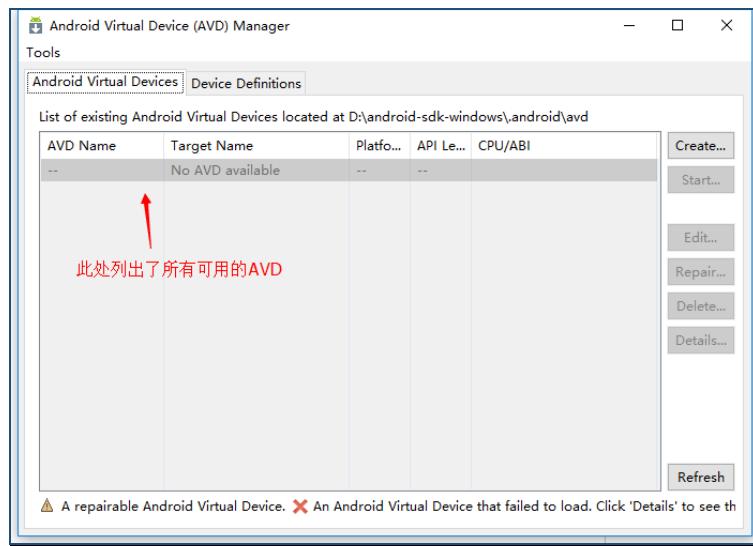


图 1.4.8

步骤四 单击右边的“Create...”按钮，AVD管理器弹出如图 1.4.9 所示的对话框。

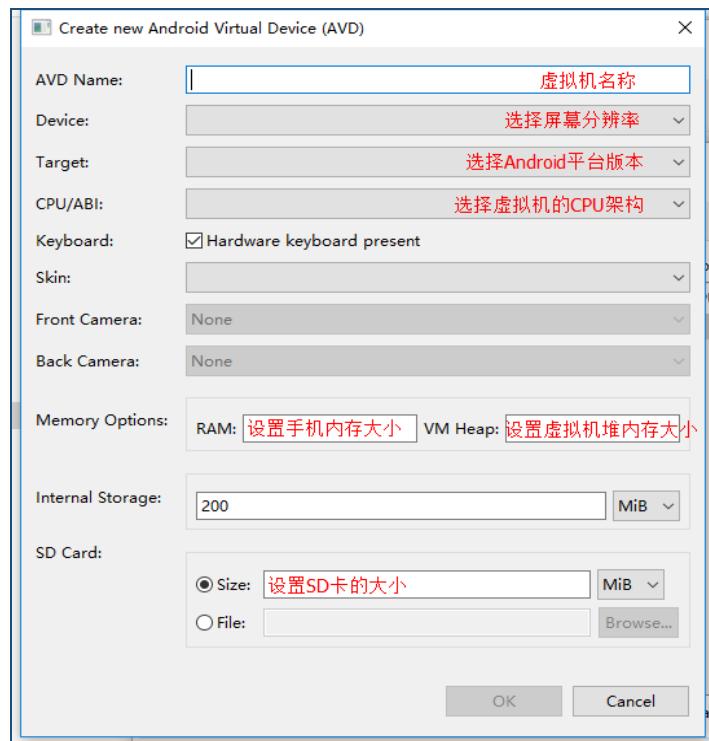


图 1.4.9

步骤五 填写 AVD 设备的名称、Android 平台的版本和虚拟 SD 卡的大小，然后单机“OK”按钮，管理器即将开始创建 AVD 设备，只需要稍作等待即可。

Skin 代表你新建的 AVD 的分辨率是多少，一般选择 HVGA。

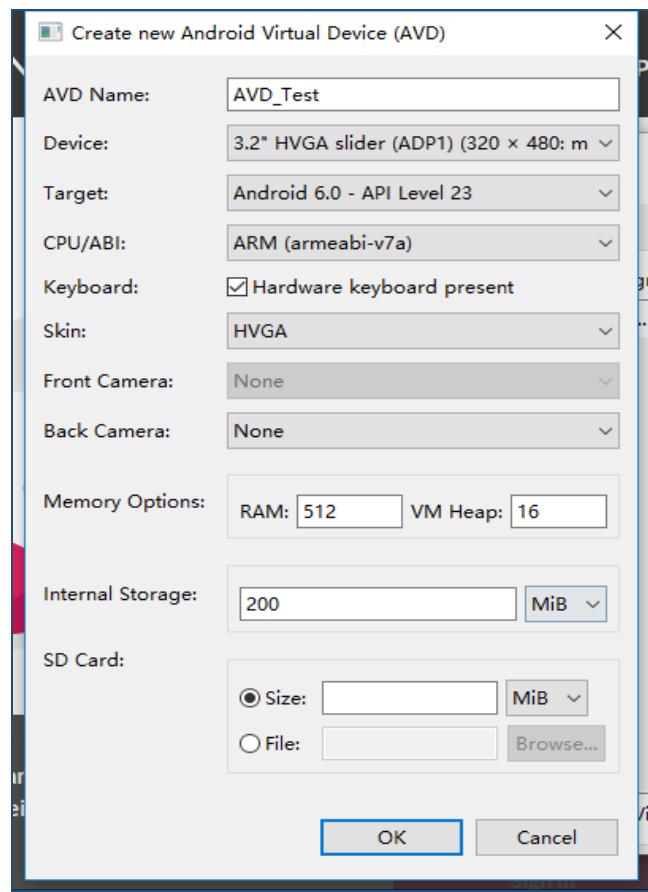


图 1.4.10

步骤六 在弹出的窗口中点击“OK”按钮，然后管理器返回如图 1.4.12 所示窗口，此时该管理器会列出当前所有可用的 AVD 设备。如果想删除某个 AVD 设备，只需选择指定的 AVD 设备，然后单击右边的“Delete...”按钮即可。

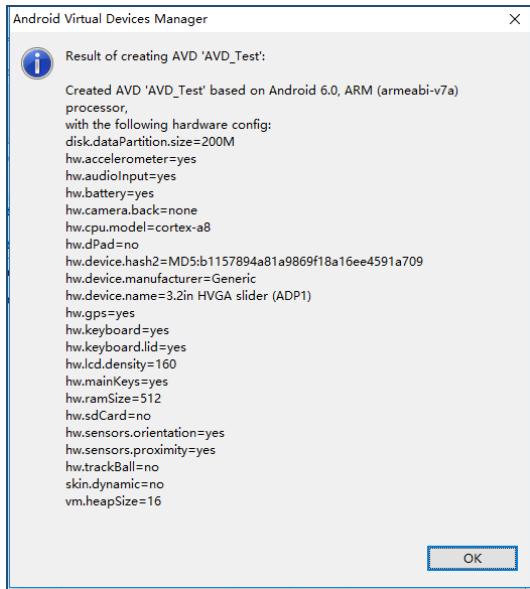


图 1.4.11

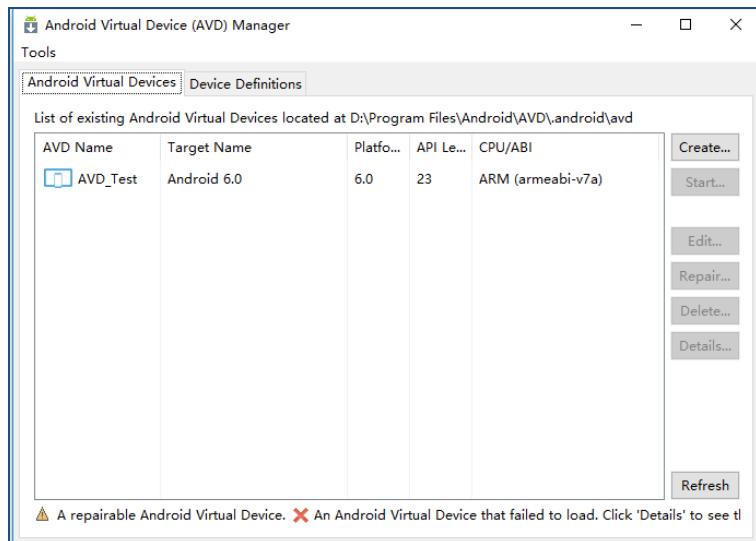


图 1.4.12

步骤七 AVD 创建成功后，接下来就可以使用模拟器来运行该 AVD 了。在 AVD Manager 中选择要运行的 AVD 设备，然后点击右侧的“Start...”按钮，然后在弹出的窗口中单击“Launch”按钮。

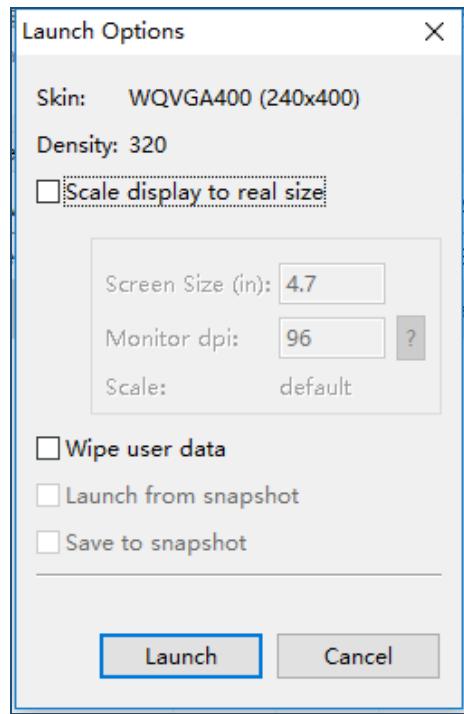


图 1.4.13

步骤八 启动 AVD 设备后需等待 AVD 设备系统启动，等待时间可能很长，需耐心等待。

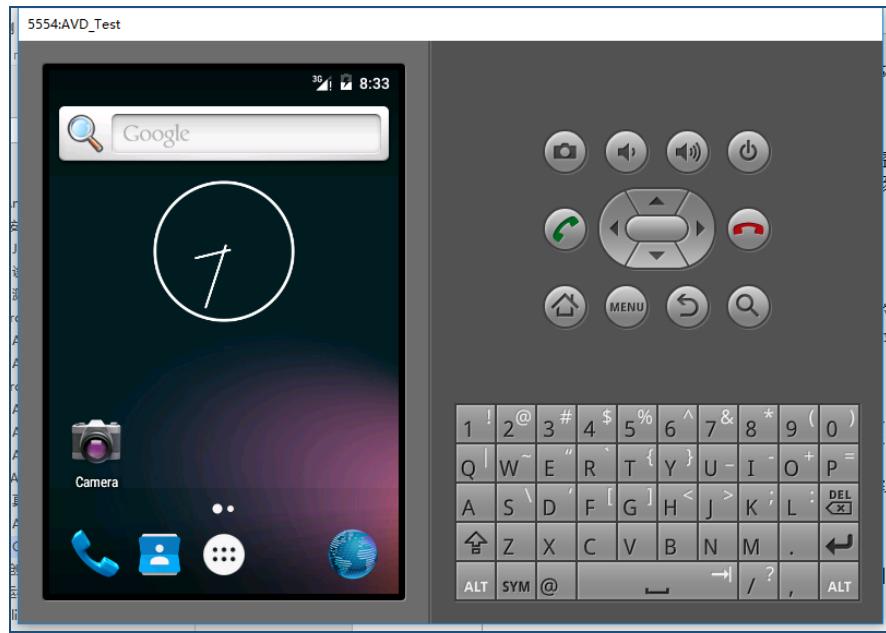


图 1.4.14

步骤九 此时 AVD 设备就是一个运行在电脑上的虚拟手机，用过 Android 手机的人肯定对这个界面不会太陌生，此时就可以通过该“虚拟手机”来模拟一些

手机操作了。

➤ Genymotion 模拟器作为运行、调试环境

使用 Android 自带的模拟器虽然简单、方便，但是最大的问题就是慢，慢到大部分的开发者难以忍受，这时我们可以选择使用第三方的模拟器：Genymotion，这个模拟器最大的特点就是速度快，使用速度可模拟出真机媲美的速度。

步骤一 双击下载好的 Genymotion 安装文件，打开 Genymotion 安装程序。



图 1.4.15

步骤二 Genymotion 安装非常简单，根据提示进行选择安装就可以，安装前需要指定本地安装位置、是否创建开始菜单、是否创建桌面图标选项，选择完一直惦记“Next”按钮，直到出现“Install”按钮，点击之后开始安装。

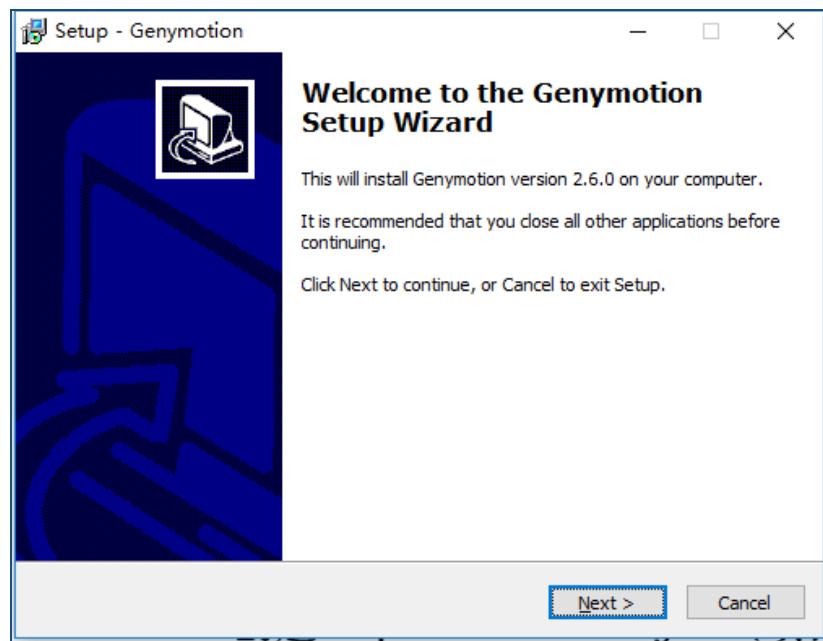


图 1.4.16

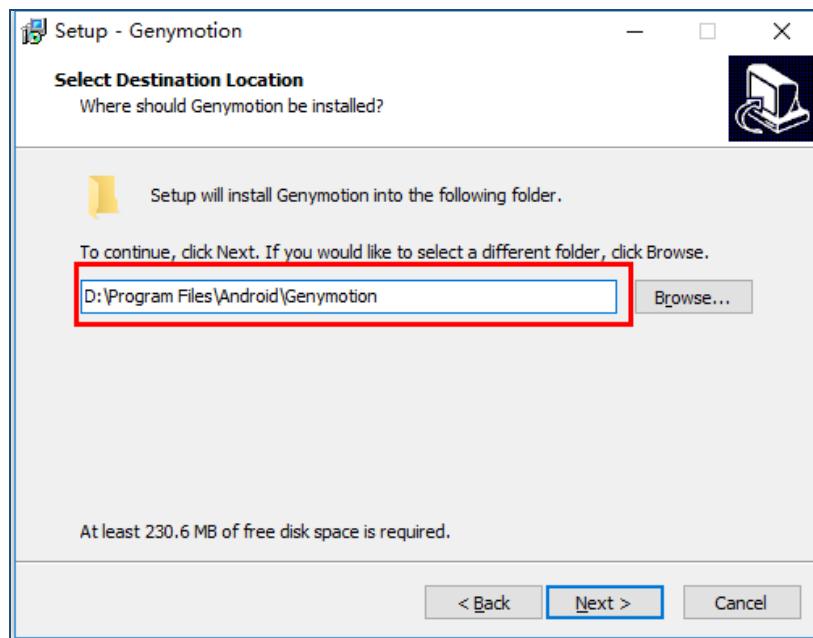


图 1.4.17

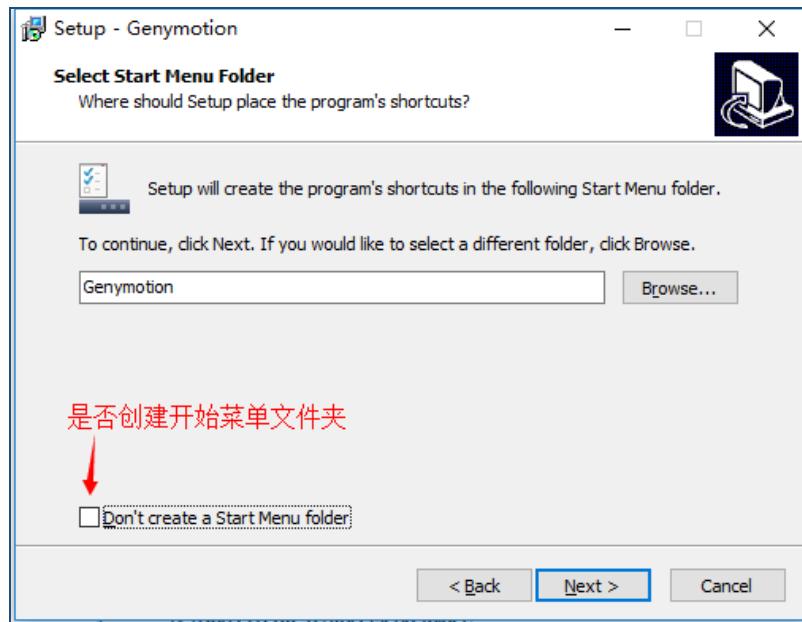


图 1.4.18

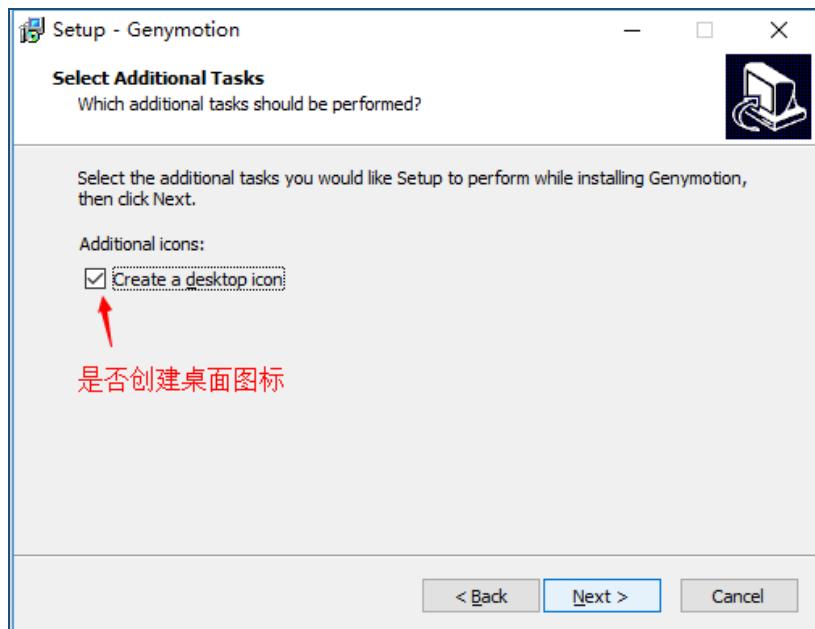


图 1.4.19

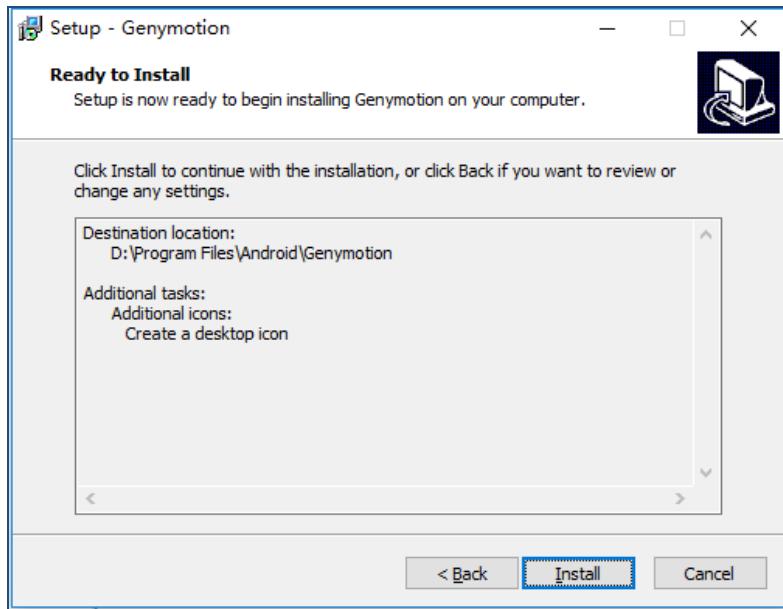


图 1.4.20

步骤三 因为下载的是包含 VirtualBox 的安装包，所以在 Genymotion 安装完成之后，还会出现 VirtualBox 的安装界面，直接点击“Next”按钮进入下一个安装界面。



图 1.4.21

步骤四 在如图 1.4.22 所示窗口中指定要安装的本地路径，**注意：VirtualBox 的安装路径中不要出现中文字符，使用纯英文字符路径**。指定好路径之后点击“Next”按钮。

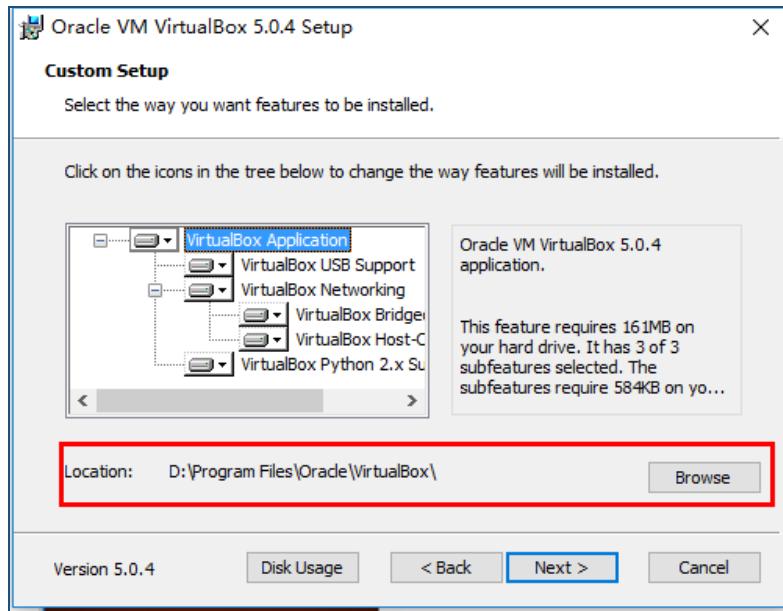


图 1.4.22

步骤五 在如图 1.4.23 所示窗口中，根据自己的需求对选项进行勾选，然后点击“Next”按钮。

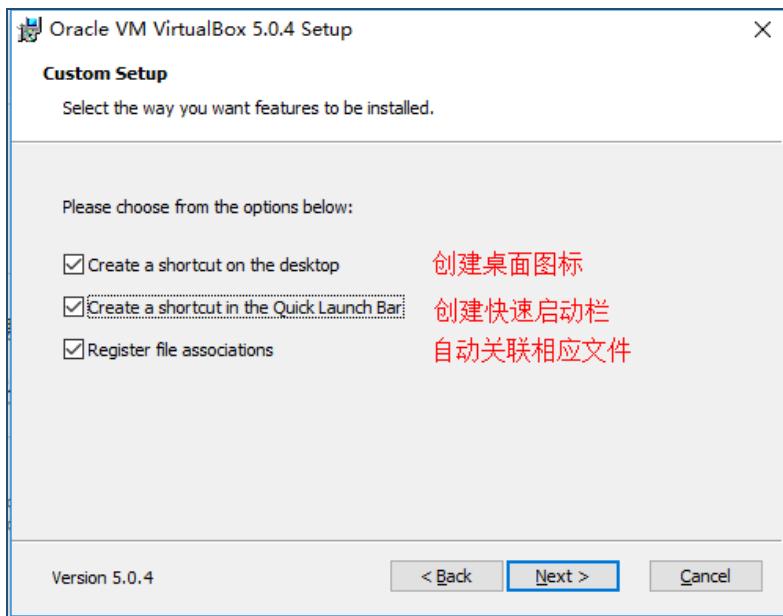


图 1.4.23

步骤六 接下来会弹出关于网卡的提示窗口，意思是当前安装 VirtualBox 时，其中涉及到网络方面的功能，需要创建一个虚拟网卡，会导致你的网络暂时中断，不过当然也会马上就恢复正常。所以，点击“Yes”即可。

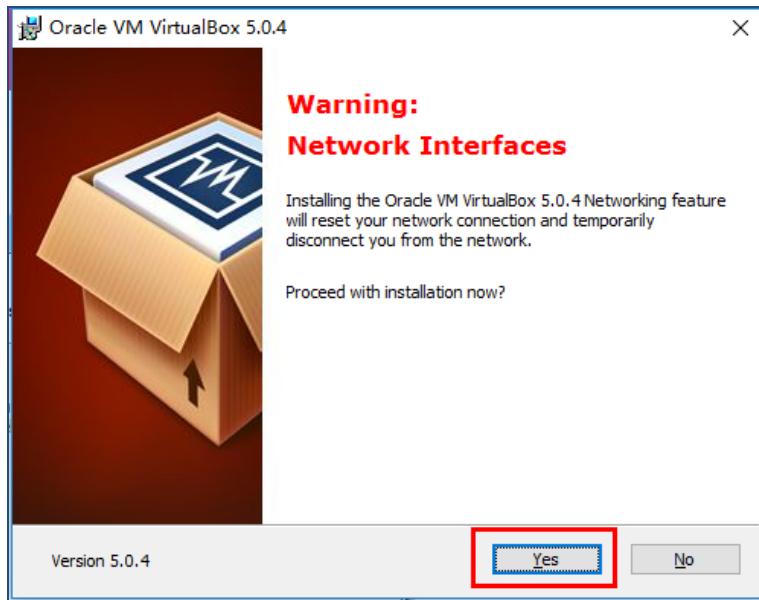


图 1.4.24

步骤七 然后在弹出的窗口中直接点击“Install”按钮进行安装。期间会弹出如图 1.4.26 所示窗口，询问是否安装这个设备软件，选择“安装”。

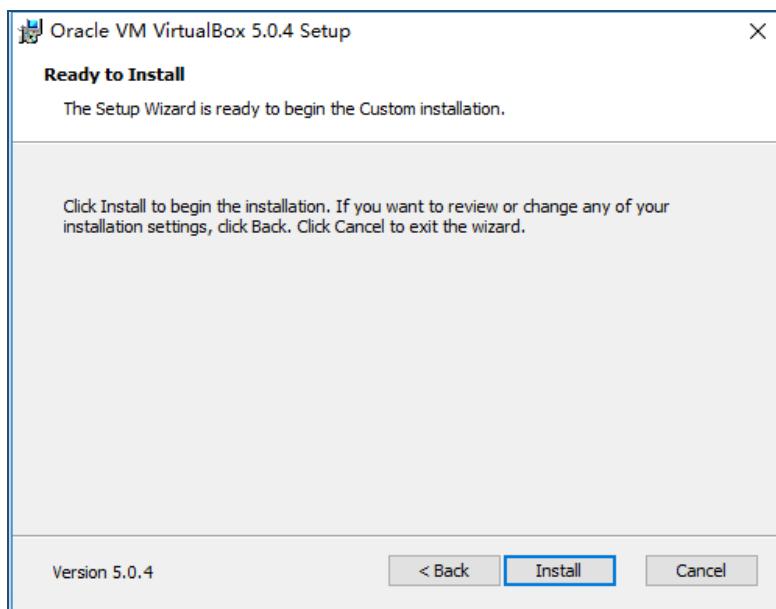


图 1.4.25



图 1.4.26

步骤八 当安装完成后会出现如图 1.4.27 所示窗口，关闭即可，此时 Genymotion 的安装窗口也会提示安装完成，此时 Genymotion 安装完成。



图 1.4.27

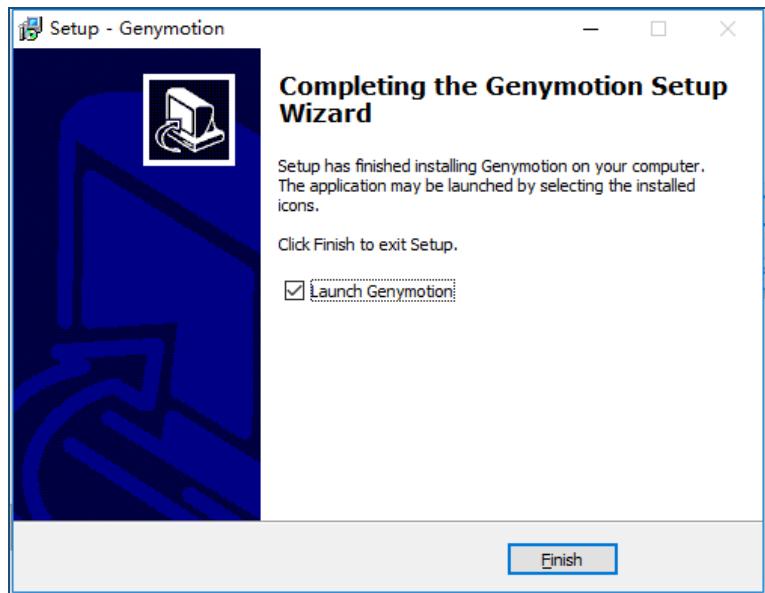


图 1.4.28

步骤九 Genymotion 模拟器的使用，安装完 Genymotion 之后，双击 Genymotion 模拟器的图标打开 Genymotion，即可看到如图 1.4.30 所示的窗口



图 1.4.29

步骤十 这个窗口显示的对于个人免费版的一些说明，禁止用于商业用途和盈利性开发，直接点击“Accept(接受)”按钮。



图 1.4.30

步骤十一 在当前窗口中会列出所有可用的模拟器，此时还没有任何 Genymotion 模拟器，可以通过点击“Add”按钮来添加模拟器，点击“Add”按钮。

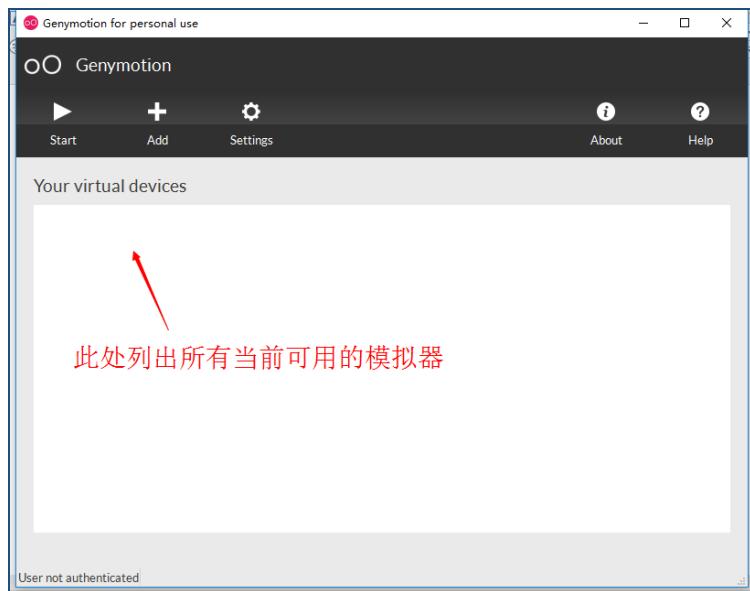


图 1.4.31

步骤十二 在窗口虚拟设备列表中是空的，无法进行新设备的添加，此时需要点击右下角的“Sign in”按钮，然后使用 Genymotion 账号进行登录，Genymotion 账号的注册参考前面的注册方法。

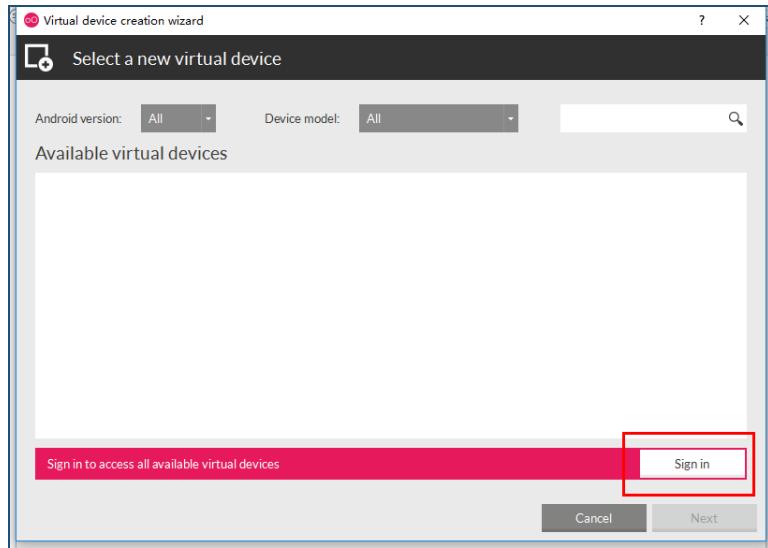


图 1.4.32

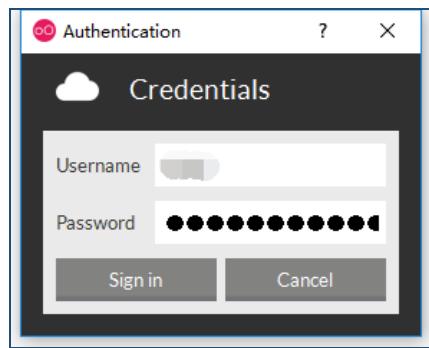


图 1.4.33

步骤十三 登录之后就可以看到允许添加的设备信息了，可以根据 Android 版本以及设备型号进行筛选，选择一个 Android 6.0 的虚拟设备，点击“Next”按钮。

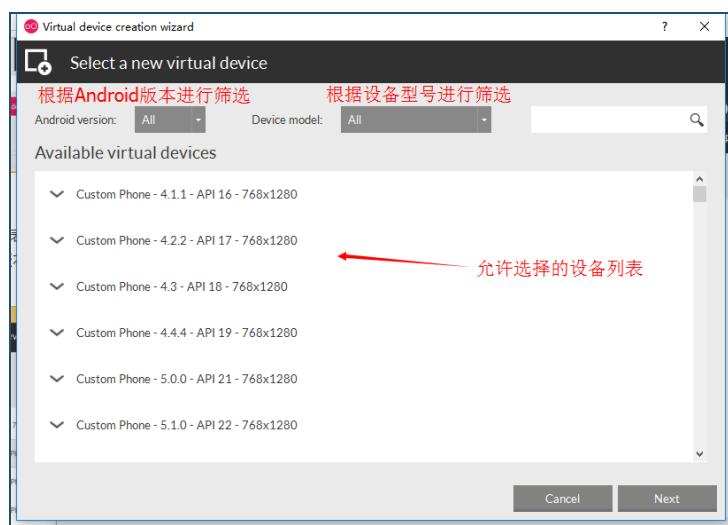


图 1.4.34

步骤十四 在新出现的窗口中输入虚拟设备的名称，然后点击“Next”按钮，此时会对选择的虚拟设备进行下载，耐心等待下载完成，下载完成之后点击“Finish”按钮。

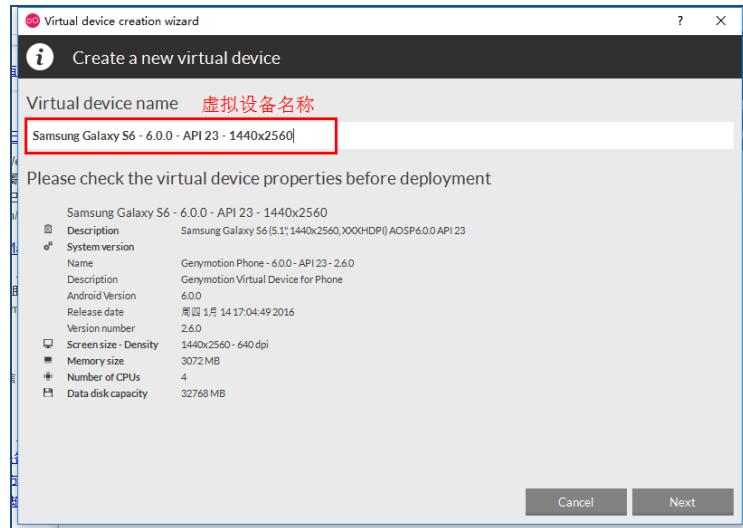


图 1.4.35

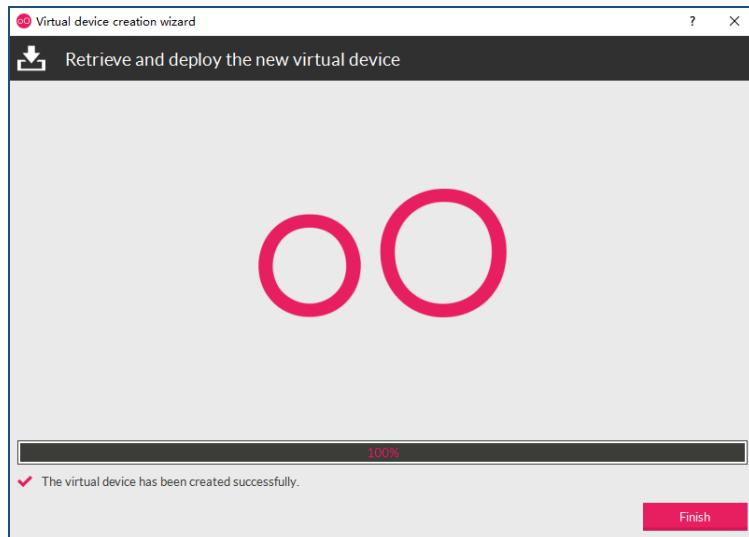


图 1.4.36

步骤十五 虚拟设备下载完成之后，在列表中会显示出当前本机下载的虚拟设备信息，如需对虚拟设备进行设置，点击设备列表右侧的扳手图标对虚拟设备进行设置。

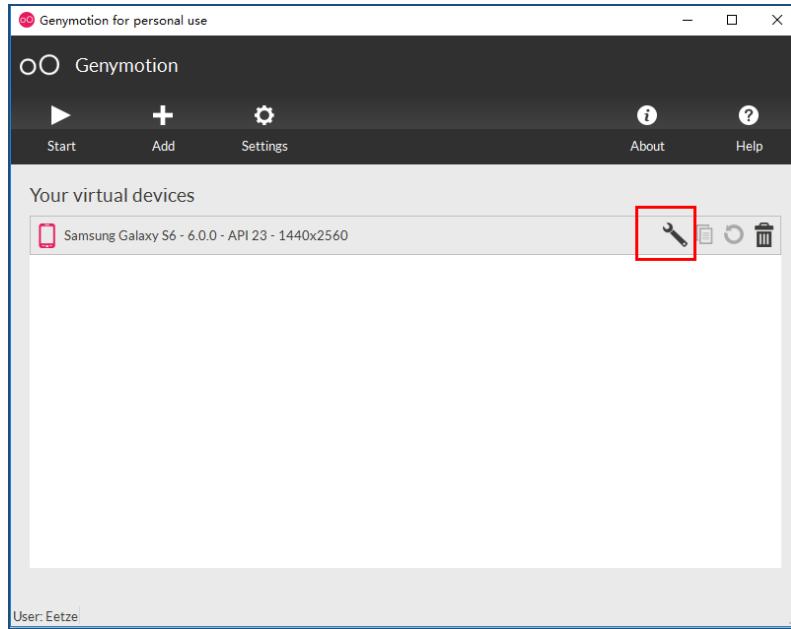


图 1.4.37

步骤十六 在弹出的虚拟设备设置窗口中可以对设备的 CPU 个数、内存大小以及 dpi 等参数进行设置。具体设置的数值根据电脑的配置，如果电脑配置较高可以对虚拟设备设置较高的参数。设置完成之后点击“OK”按钮。

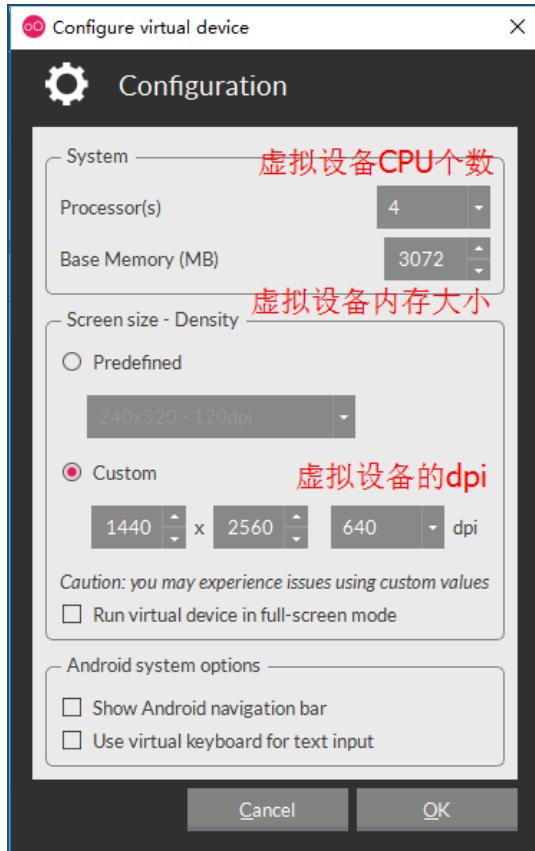


图 1.4.38

步骤十七 虚拟设备信息设置完成之后，还需要对 Genymotion 进行设置，在如图 1.4.39 窗口中点击“Settings”。

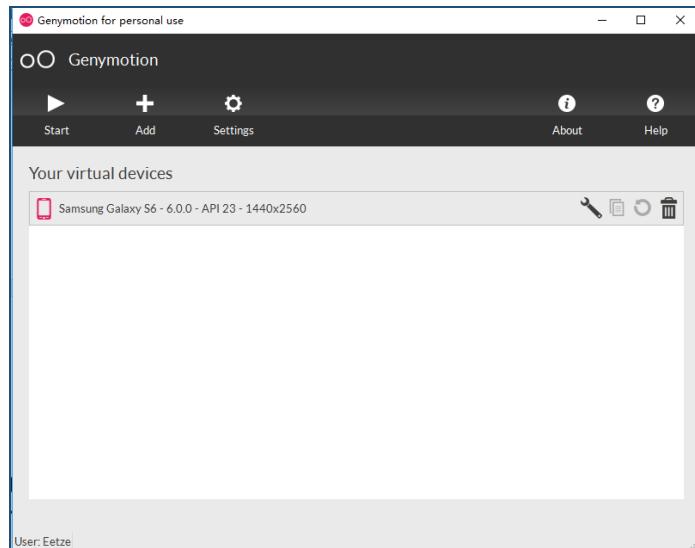


图 1.4.39

步骤十八 然后再弹出的设置窗口中选择“ADB”标签，设置“Use custom Android SDK tools”的Android SDK 路径为之前本地安装的路径。设置完成之后关闭窗口。

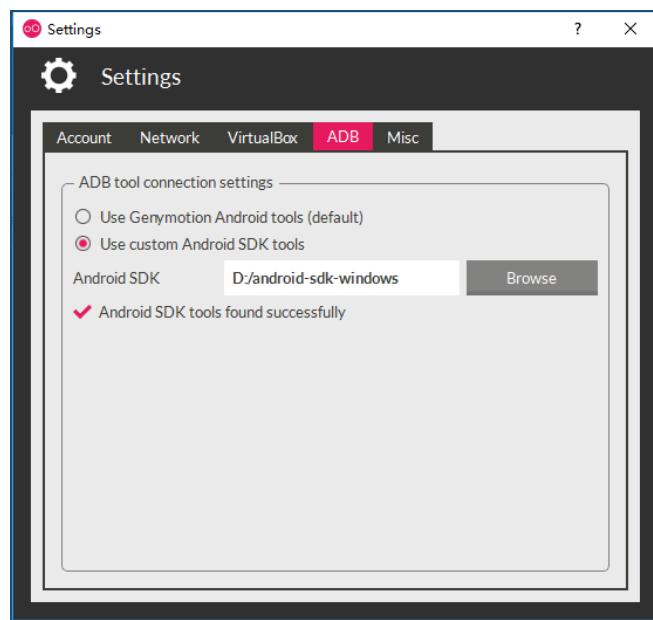


图 1.4.40

步骤十九 选中虚拟设备，然后点击“Start”按钮，启动虚拟设备。Genymotion 模拟器的启动速度比 Android 自带的模拟器的启动速度快，因此很快就能看到如所示的界面。

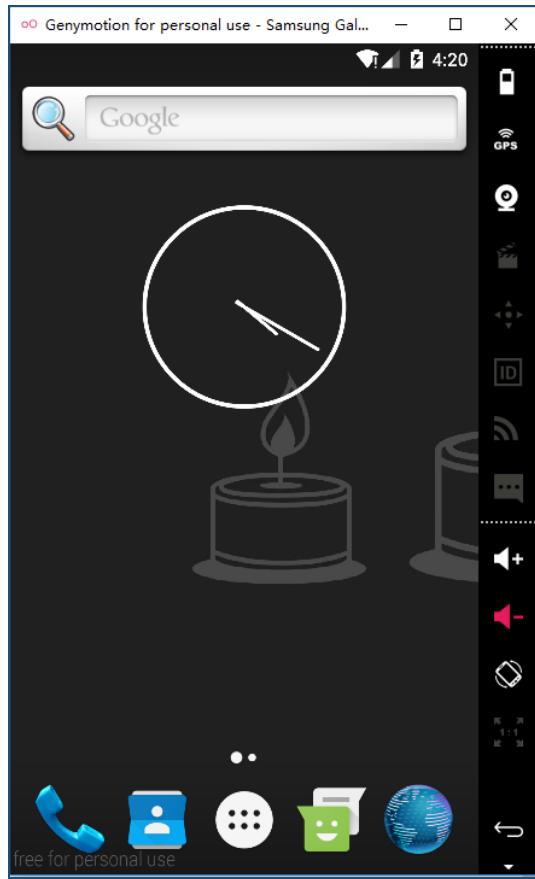


图 1.4.41

步骤二十 Android Studio 中配置 Genymotion 模拟器，在 Android Studio 窗口中依次选择“Configure”、“Settings”进入 Android Studio 设置窗口。

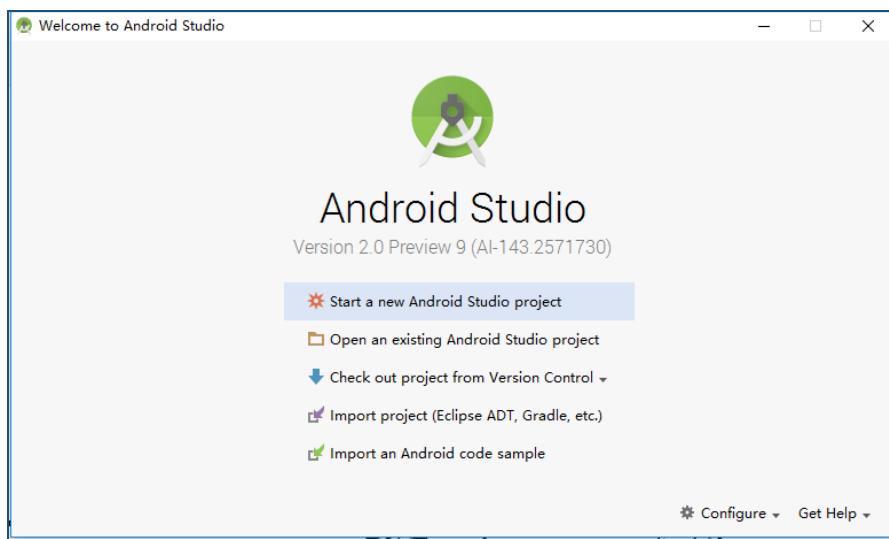


图 1.4.42

步骤二十一 然后在左侧列表中选择“Plugins”，然后在右侧下方点击

“Browse repositories...” 按钮。

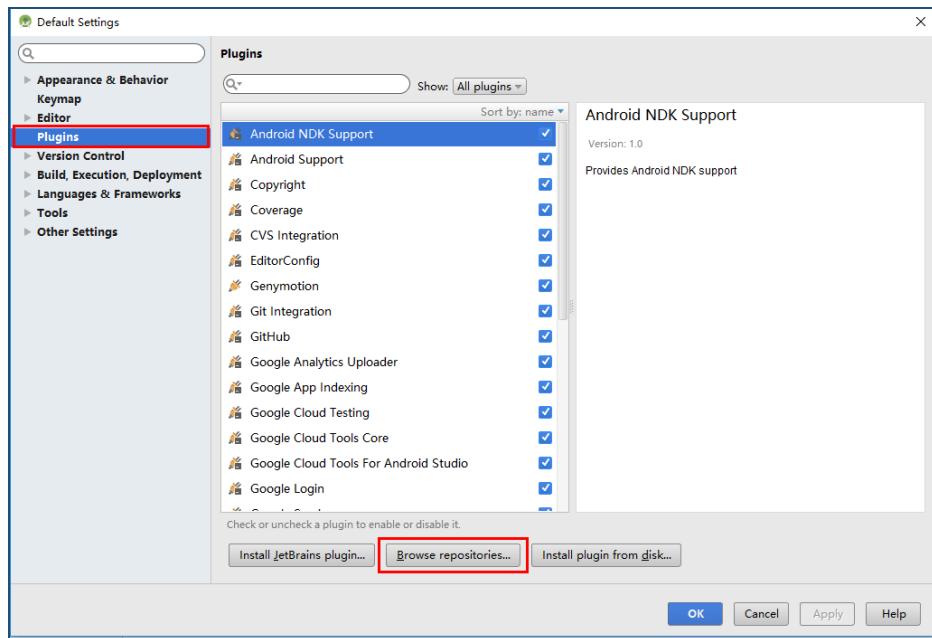


图 1.4.43

步骤二十二在出现窗口的搜索框中输入 genymotion，找到对应的 Genymotion 插件，然后点击“Install”按钮，对 Genymotion 插件进行安装。

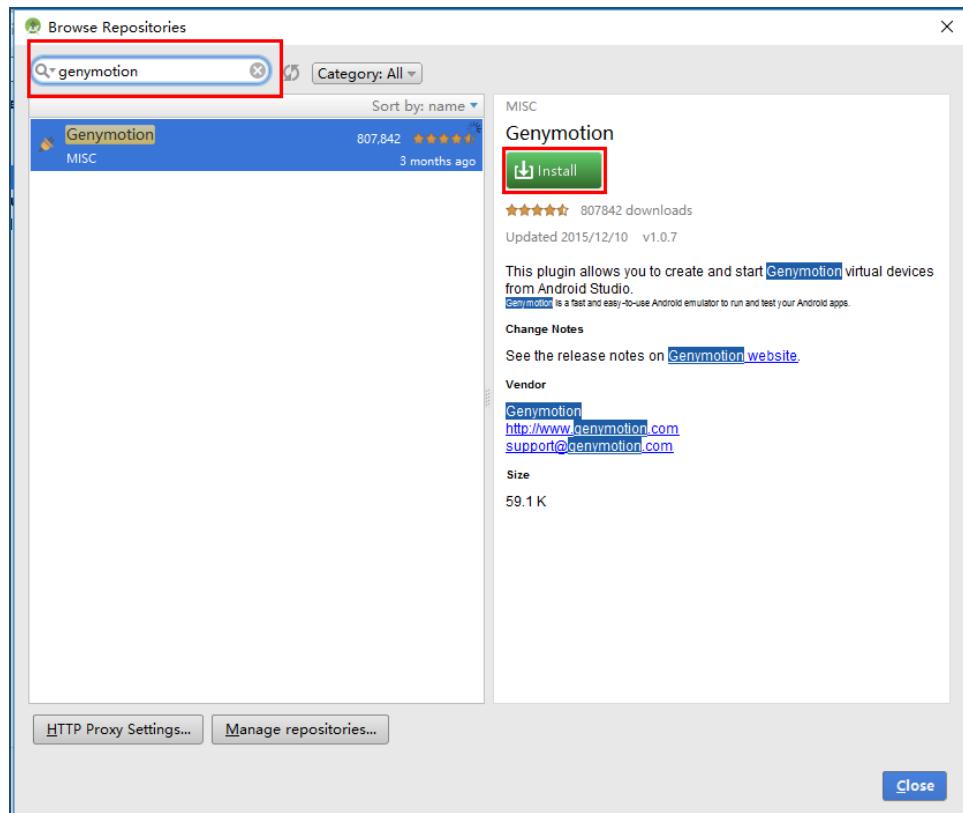


图 1.4.44

步骤二十三 安装完 Genymotion 插件之后，点击“Restart Android Studio”对 Android Studio 进行重启。

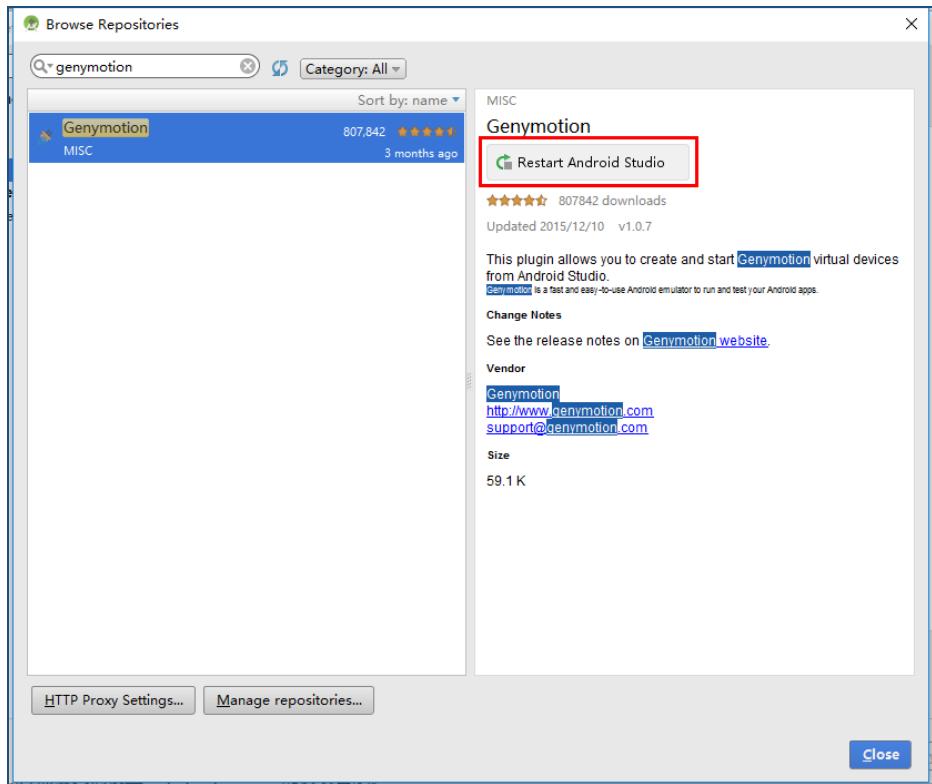


图 1.4.45

步骤二十四 重新启动 Android Studio 之后，在 Android Studio 窗口中依次选择“Configure”、“Settings”进入 Android Studio 设置窗口。然后在左侧树状列表中依次点击“Other Settings”、“Genymotion”。在窗口右侧设置之前本地安装 Genymotion 的路径，设置完成之后关闭设置窗口。

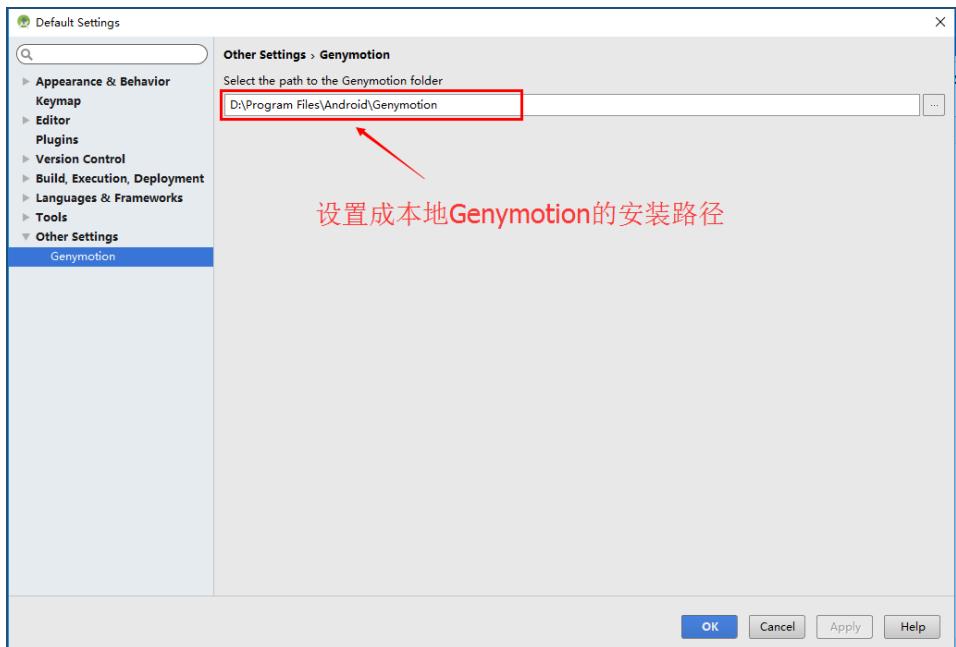


图 1.4.46

1.4.4 实验结论

Android 运行、调试环境安装成功。

1.5 实验五 Android Studio 创建一个 Android 应用

1.5.1 实验目的

1. 熟练掌握 Android Studio 开发环境。
2. 熟练掌握 Android 应用的开发步骤。

1.5.2 准备工作

1. 配 JDK。
2. 配置 Android SDK。
3. 配置 Android Studio。
4. 配置好一种 Android 运行、调试环境。

1.5.3 实验步骤

步骤一 创建 Android Studio 项目，打开 Android Studio，点击“Start a new Android Studio project”来创建一个 Android Studio 项目。

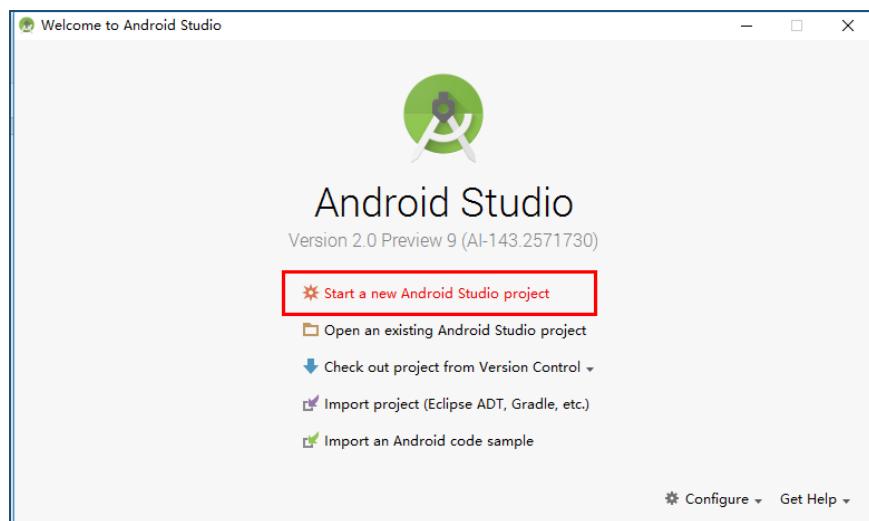


图 1.5.1

步骤二 在出现的窗口中输入项目名称、公司域名，以及指定项目存放的本地路径之后，点击“Next”按钮。

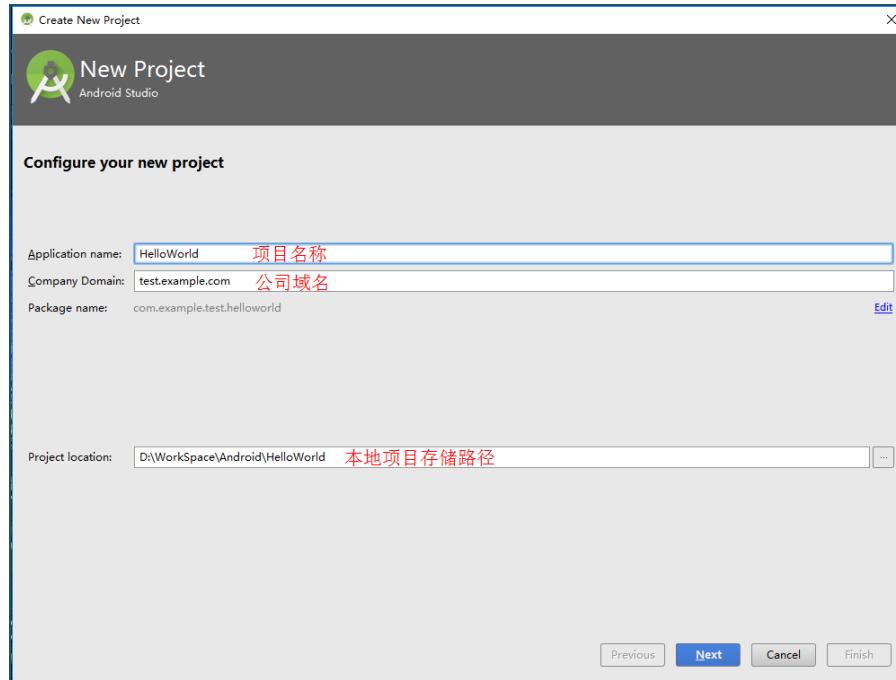


图 1.5.2

步骤三 选择“Phone and tablet”（手机及平板电脑），也就是项目应用所使用的硬件平台，然后再指定项目所支持的最低版本 SDK，点击“Next”按钮进入下一个窗口。

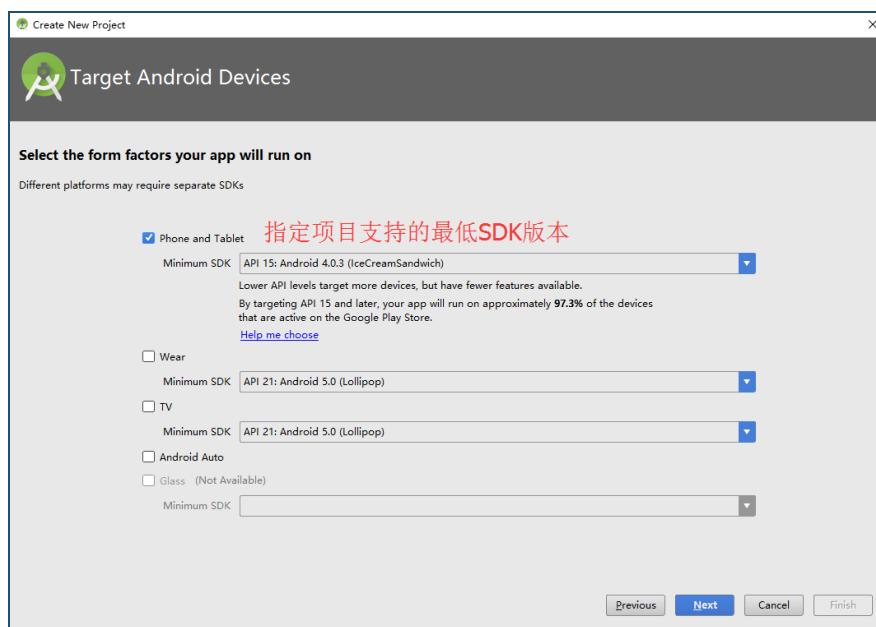


图 1.5.3

步骤四 在如所示窗口中选择“Empty Activity”，然后点击“Next”按钮。

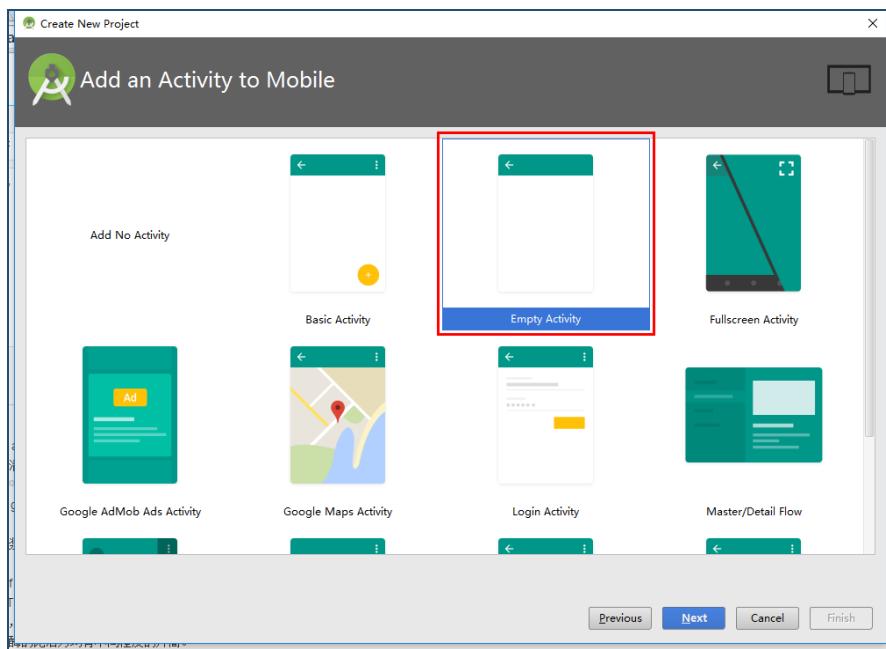


图 1.5.4

步骤五 接着在出现的窗口中不需要做修改，直接点击“Finish”按钮，完成项目的创建。

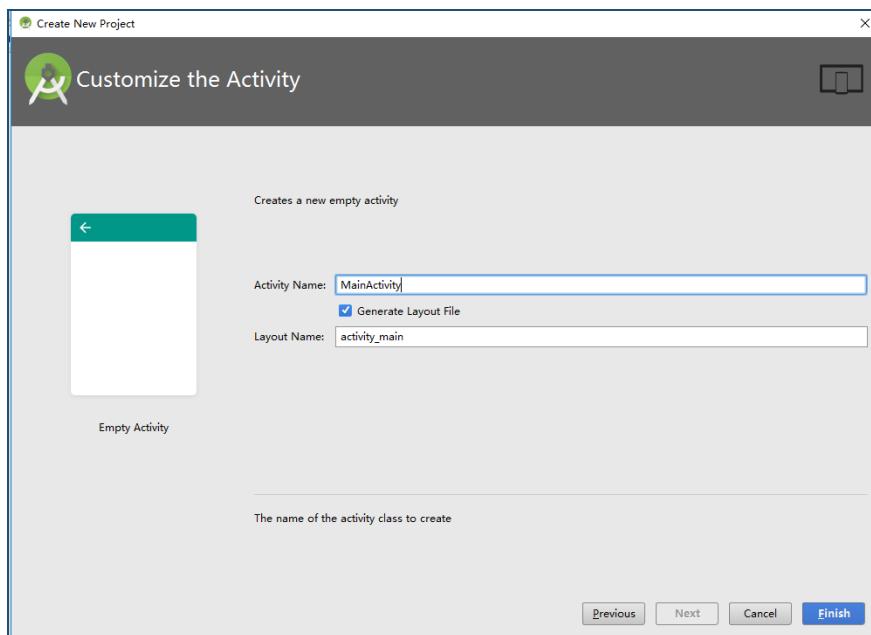


图 1.5.5

步骤六 运行 Android Studio 项目，当项目完成创建之后，在 Android Studio 下侧可能会出现如图 1.5.6 所示的错误信息。提示工具版本太旧，错误的原因是因为当前使用的 Android Studio 并非正式版，造成了 Android Studio 使用的

Gradle 版本与其使用 Gradle 插件版本不兼容造成的。**如果没有遇到此问题，可直接跳转到步骤十二。**

Gradle 是一种依赖管理工具，基于 Groovy 语言，面向 Java 应用为主，它抛弃了基于 XML 的各种繁琐配置，取而代之的是一种基于 Groovy 的内部领域特定（DSL）语言。Android Studio 使用了 Gradle 来构建项目。

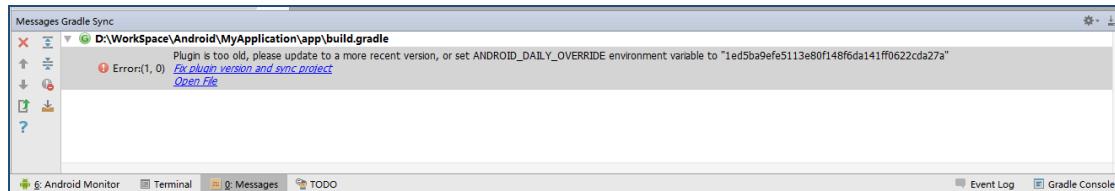


图 1.5.6

步骤七 点击“File”菜单，选择“Project Structure”，打开项目结构窗口。

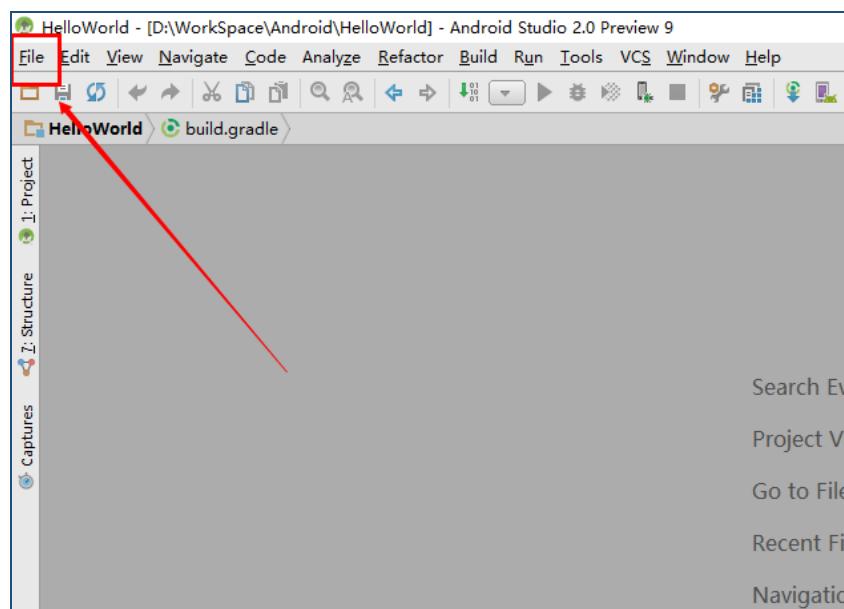


图 1.5.7

步骤八 在项目结构窗口左侧选择“Project”，然后在右侧可以看到当前 Gradle 版本与插件版本

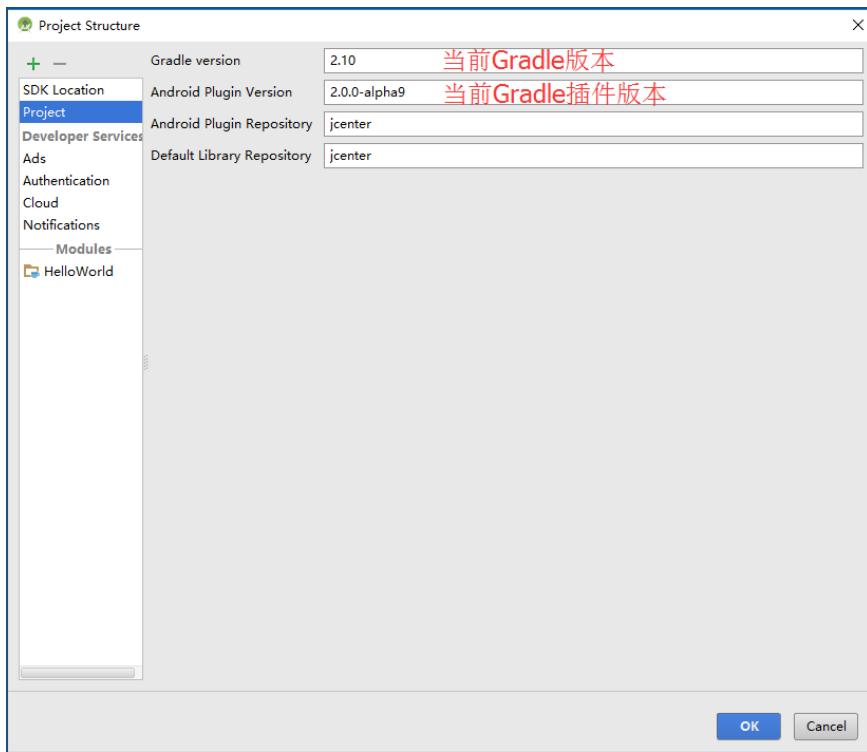


图 1.5.8

步骤九 打开如下 Gradle 插件地址，可以查看 Gradle 插件的所有版本信息，在其中选择一个较新的版本，记住版本名称，例如 2.1.0-alpha3 这个版本。

地址：<https://jcenter.bintray.com/com/android/tools/build/gradle/>

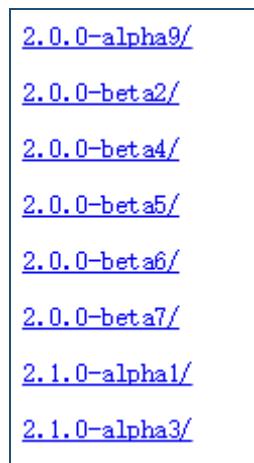


图 1.5.9

步骤十 然后在如图 1.5.8 窗口中，修改当前 Gradle 插件版本名称为 2.1.0-alpha3，然后点击“OK”按钮。当修改完 Gradle 插件版本之后，Android Studio 会自动下载指定版本的 Gradle 插件。(如果插件版本名称错误是无法完成下载的)。

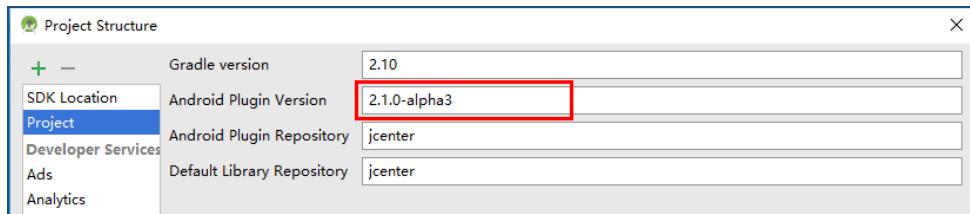


图 1.5.10

步骤十一 当 Android Studio 窗口最下方的状态栏显示“Gradle build finished in Xs XXXms”信息时，表示 Gradle 插件已经下载完毕。



图 1.5.11

步骤十二 在 Android Studio 中依次点击菜单栏的“File”、“Settings”，打开设置窗口，在设置窗口左侧找到“Build, Execution, Deployment”下的“Instant Run”，在窗口右侧将出现的三个单选框都取消勾选，然后点击“OK”按钮。

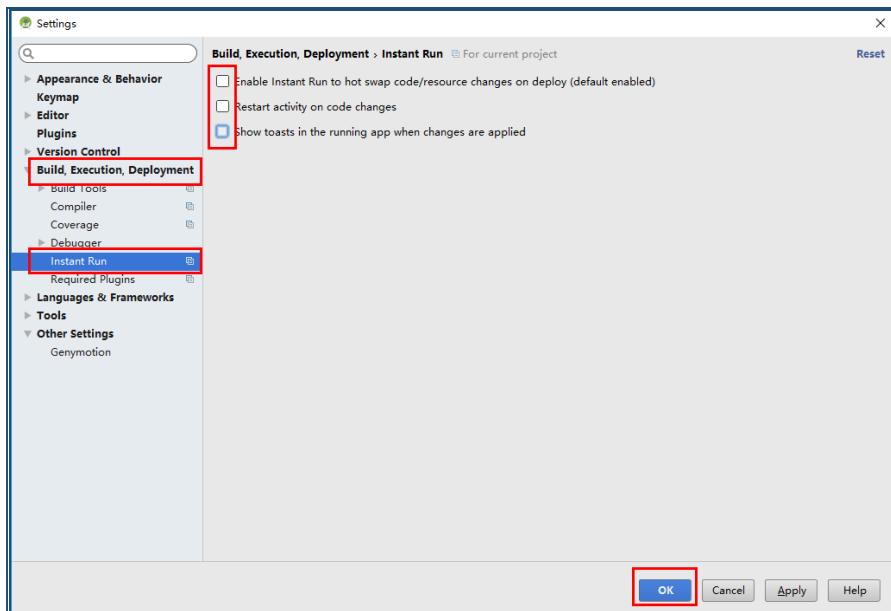


图 1.5.12

步骤十三 在 Android Studio 的工具栏中找到之前安装的 Genymotion 插件按钮，点击打开。

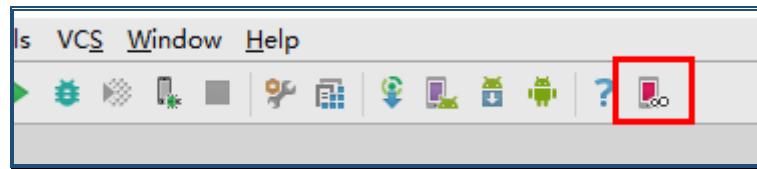


图 1.5.13

步骤十四 在 Genymotion 虚拟设备管理窗口中选择之前安装的虚拟设备，并点击“Start...”按钮，等待一会便会打开虚拟 Android 设备。

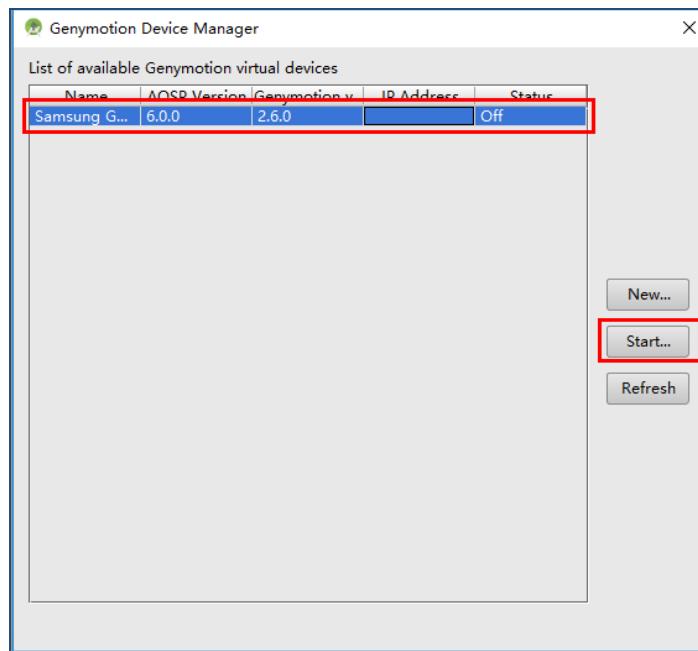


图 1.5.14

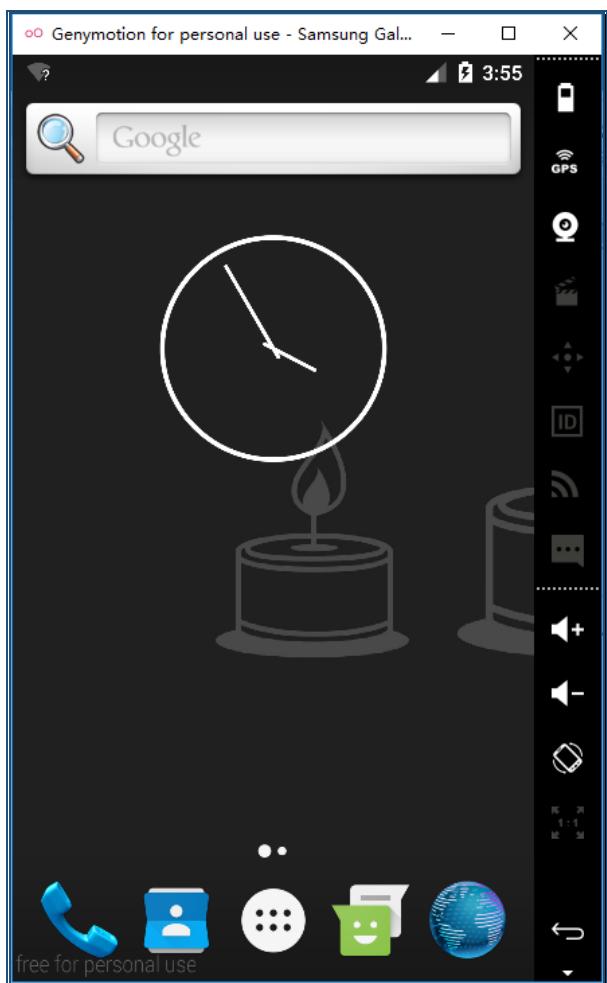


图 1.5.15

步骤十五 在 Android Studio 的工具栏点击运行按钮，并在弹出的选择部署设备窗口中选择刚刚启动的 Genymotion 虚拟设备，并点击“OK”按钮。

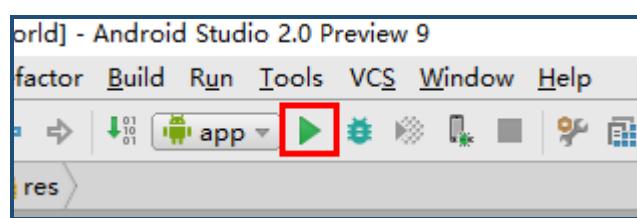


图 1.5.16

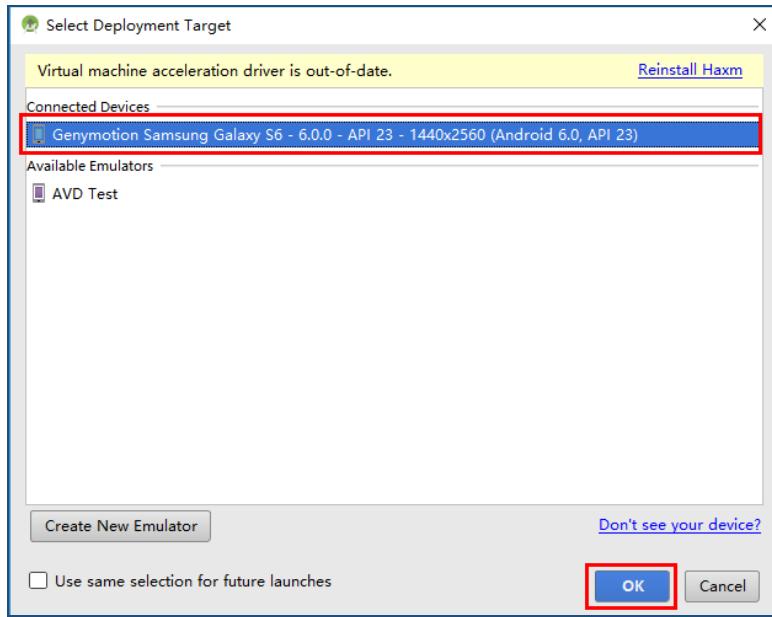


图 1.5.17

步骤十六 经过短暂的等待，在 Genymotion 虚拟 Android 设备中会运行项目应用，此时 Android Studio 项目应用程序就运行起来了。

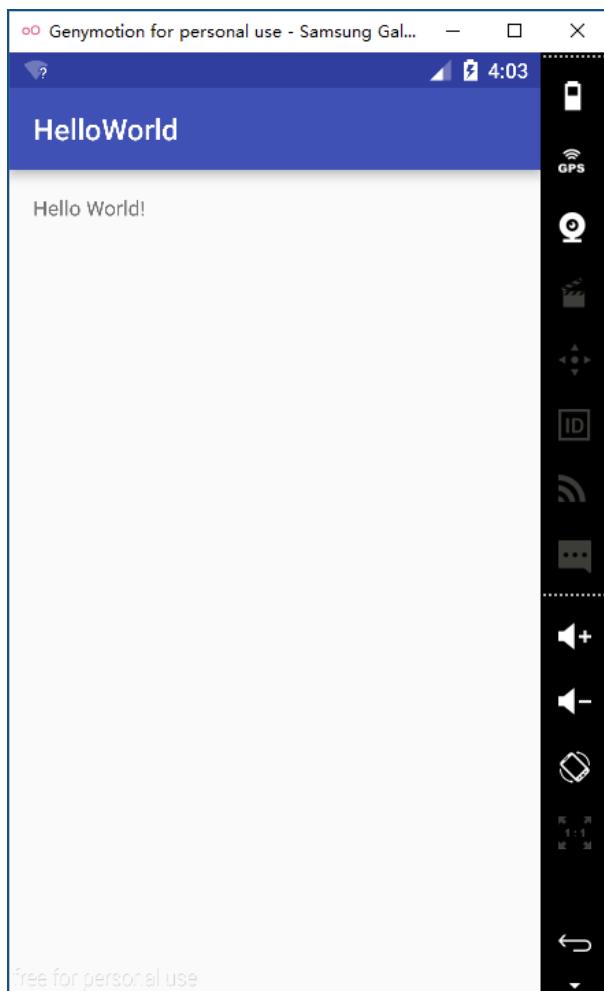


图 1.5.18

1.5.4 实验结论

通过以上实验，掌握使用 Android Studio 来搭建 Android 开发环境及 Android 程序的开发步骤，并且能够使用 Android Studio 开发第一个 Android 应用。

1.6 实验六 LogCat 和 DDMS 的使用

1.6.1 实验目的

1. 熟练掌握使用 LogCat 查看 Android 应用程序提示消息的方法。
2. 熟练掌握查看 LogCat 消息查找应用程序错误的方法。
3. 掌握 DDMS 的基本作用。

1.6.2 准备工作

1. Android 开发环境搭建完成。
2. Android 模拟器配置完成。
3. Android 应用程序创建和启动过程。

1.6.3 实验步骤

➤ 建立 Android 应用程序项目

步骤一 在 Android Studio 中创建一个 Android 应用程序 LogCatTest。创建 Empty Activity 及默认视图布局文件。完成后的应用程序项目目录如下。

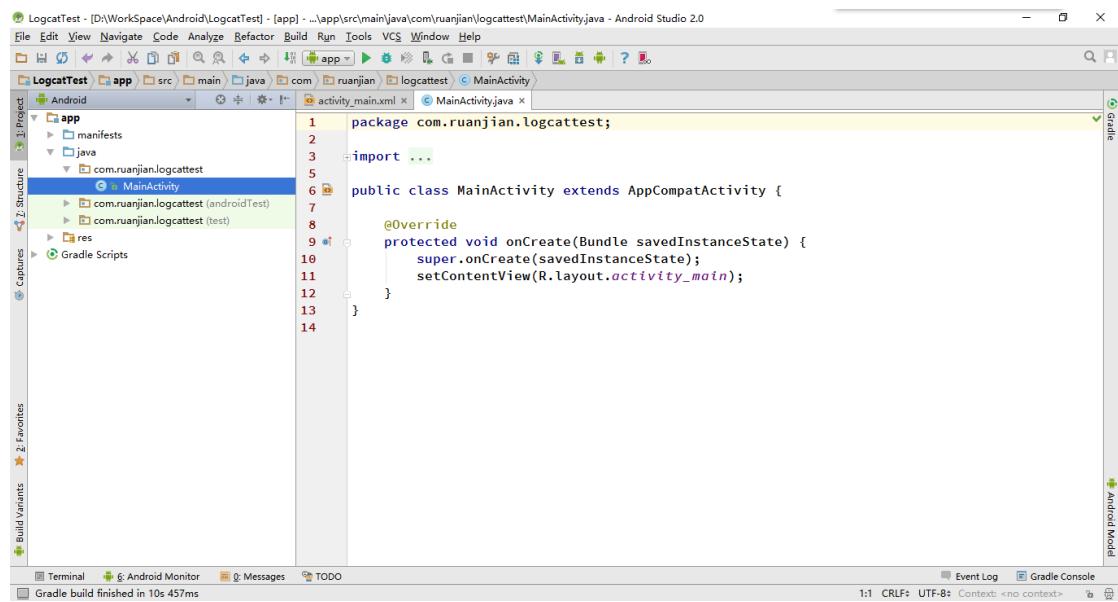


图 1.6.1

步骤二 启动 Android 应用程序，确保当前应用程序创建成功。

➤ **使用 Logcat 查看消息**

步骤三 日志在任何项目的开发过程中都会起到非常重要的作用，在 Android 项目中如果你想要查看日志则必须要使用 LogCat 工具。在 Android Studio 最下方的 Status Bar 中找到 6: Android Monitor，点击她。如下图。



图 1.6.2

步骤四 此时 LogCat 窗口出现了，如下图。

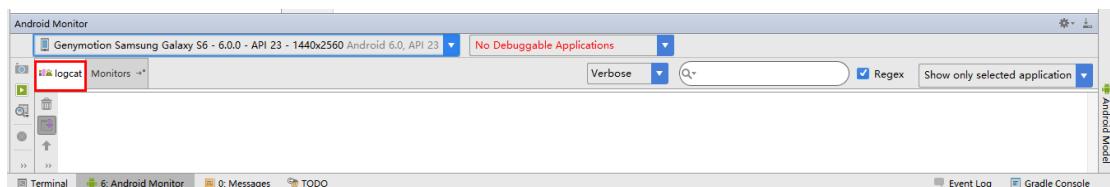


图 1.6.3

步骤五 修改应用程序项目目录中的 app\src\main\java 目录下的 MainActivity.java 文件，参考代码如下。

```

1 package com.ruanjian.logcattest;
2
3 import android.os.Bundle;
4 import android.support.v7.app.AppCompatActivity;
5 import android.util.Log;
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13
14        // 显示LogCat消息
15        String Tag = "MainActivity";
16        Log.d(Tag, "这是LogCat消息");
17        Log.e(Tag, "这是error级别的LogCat消息");
18    }
19 }
20

```

图 1.6.4

步骤六 运行应用程序，并打开 LogCat 窗口，LogCat 窗口打开后的示意图如下，会在其中发现如下信息。使用真机的同学，可能看到的还是空荡荡的 LogCat。需要做这么一件事，菜单栏 Tools → Android → Enable ADB Integration 把这个勾上，然后再次运行。



图 1.6.5

步骤七 在 LogCat 窗口中可以选择连接的设备，也可以选择显示的日志级别，包括：Verbose、Debug、Info、Warn、Error、Assert。以上级别依次升高。



图 1.6.6

步骤八 Android 中的日志工具类是 Log(android.util.Log)，提供了以下几个方法供我们打印日志。

- ✓ Log.v() 对应 Verbose
- ✓ Log.d() 对应 Debug
- ✓ Log.i() 对应 Info

- ✓ Log.w() 对应 Warn
- ✓ Log.e() 对应 Error
- ✓ Log.wtf() 在输出日志的同时，会把此处代码此时的执行路径(调用栈)打印出来。

步骤九 在 LogCat 窗口右侧为过滤器，可以允许用户创建过滤器以方便查找 LogCat 消息。点击过滤器下拉框，选择 Edit Filter Configuration，在弹出的对话框中点击左上角的加号新创建一个 Filter。如下图即打开创建过滤器对话框。

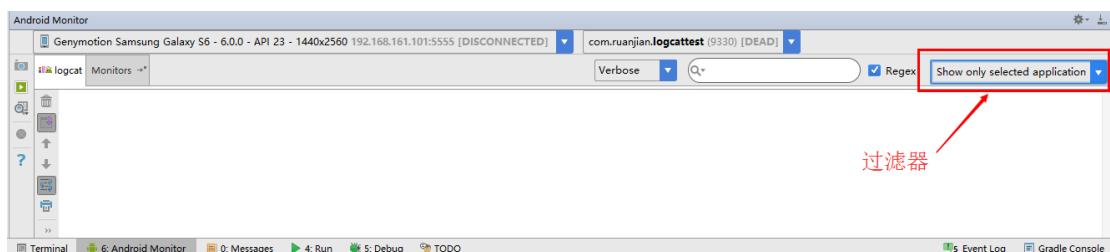


图 1.6.7

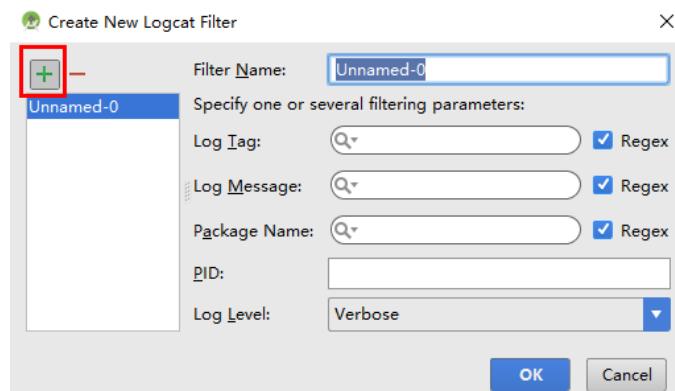


图 1.6.8

- ✓ Name: Filter 名称
- ✓ Log Tag: 通过日志的 tag 过滤
- ✓ Log Message: 通过日志的 msg 内容过滤
- ✓ Package Name: 通过包名过滤
- ✓ PID: 通过 PID 过滤
- ✓ Log Level: 通过日志等级过滤
- ✓ regex: 表示可以使用正则表达式进行匹配
- ✓

注意：以上筛选类型可以填写任意部分，也可全部填写，当作组合筛选处理。

步骤十 添加一个筛选器，筛选不同的 LogCat 消息。在 Filter Name 中输入过滤器名称，然后在 Log Tag 中输入 MainActivitu，点击 OK。

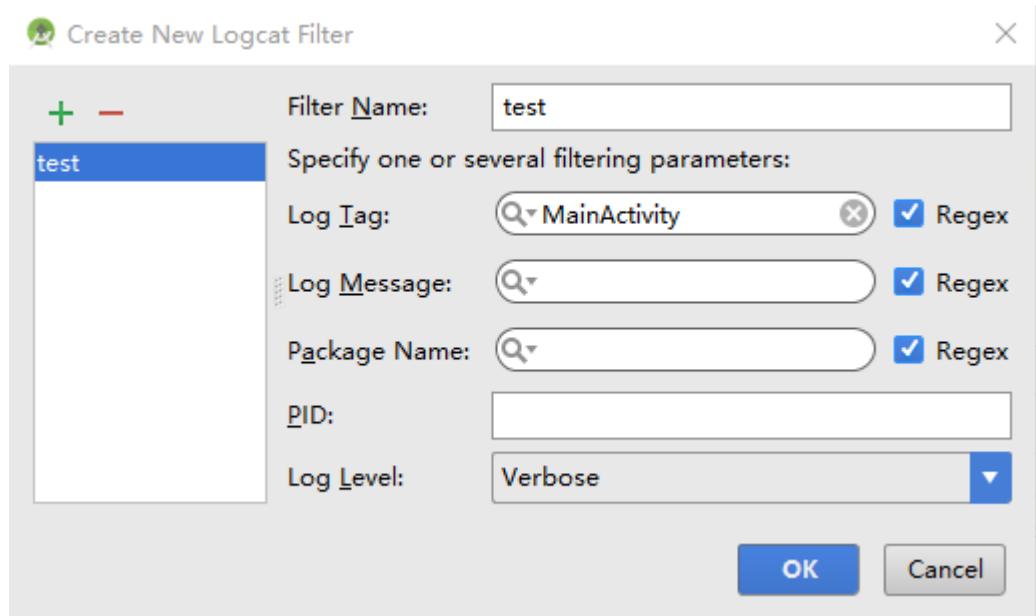


图 1.6.9

步骤十一 此时在过滤器下拉列表中选择刚刚创建的 test 过滤器，LogCat 中就会只显示 Tag 为"MainActivity"的日志信息。



图 1.6.10

➤ 使用 LogCat 查找错误

步骤十二 修改 MainActivity.java 文件，使应用程序中出现一段错误代码。此段代码插入到 MainActivity 的 onCreate()方法中。

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // 显示LogCat消息  
        String Tag = "MainActivity";  
        Log.d(Tag, "这是LogCat消息");  
        Log.e(Tag, "这是error级别的LogCat消息");  
  
        TextView tv = null;  
        tv.setText("test");  
    }  
}
```

图 1.6.11

步骤十三 重新启动应用程序，将出现错误提示。

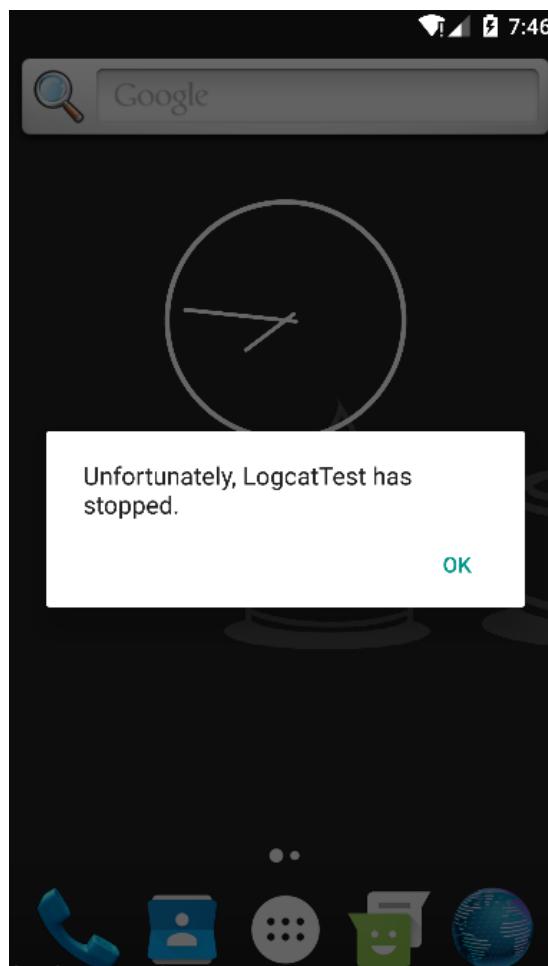


图 1.6.12

步骤十四 打开 LogCat 窗口，查看 Error 级别的消息，示意图如图所示。

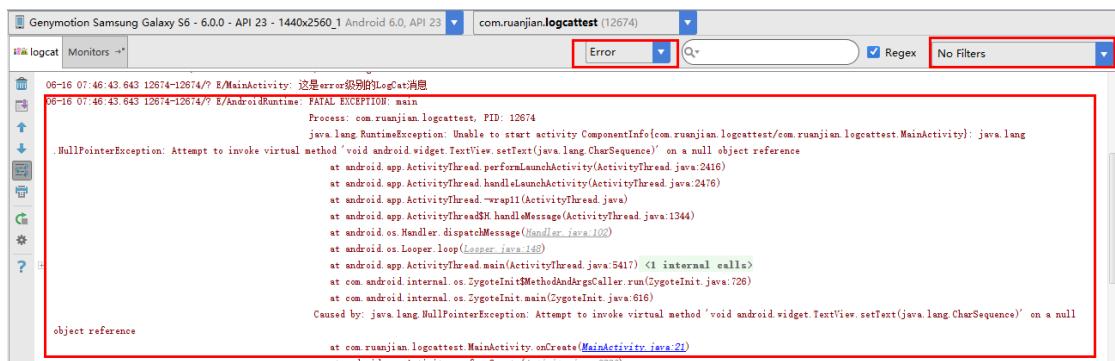


图 1.6.13

步骤十五 当程序出错时，查找错误的基本方法是：在 LogCat 消息中，查看 Error 级别的错误，查找消息中“Text”值为“FATAL EXCEPTION: main”的哪一行，即为错误的产生原因。

步骤十六 查找到“FATAL EXCEPTION: main”行，然后向下查找，找到你自己所编写的 java 文件，即为错误行位置（本例中为 MainActivity.java 文件的第 21 行出现空指针异常）；鼠标点击蓝色字部分即可跳转到问题处，改正该行错误即可。

```

at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:126)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
Caused by: java.lang.NullPointerException: Attempt to invoke virtual method 'void andr

at com.ruanjian.logcattest.MainActivity.onCreate(MainActivity.java:21)
at android.app.Activity.performCreate(Activity.java:6205)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1107)
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2369)

```

➤ 使用 DDMS

步骤十七 DDMS (Dalvik Debug Monitor Service)，是 Android 开发环境中的 Dalvik 虚拟机调试监控服务。可以进行的操作有：为测试设备截屏，查看特定行程中正在运行的线程以及堆信息、Logcat、广播状态信息、模拟电话呼叫、接收 SMS、虚拟地理坐标等，功能非常强大，对于安卓开发者来说是一个非常好的工具。

打开方式为：在 Android Studio 菜单上选择“Tools”→“Android”→“Android Device Monitor”，将会 DDMS 窗口。也可以直接在 Android Studio 快捷工具栏中点击“Android Device Monitor”图标，直接打开 DDMS 窗口。

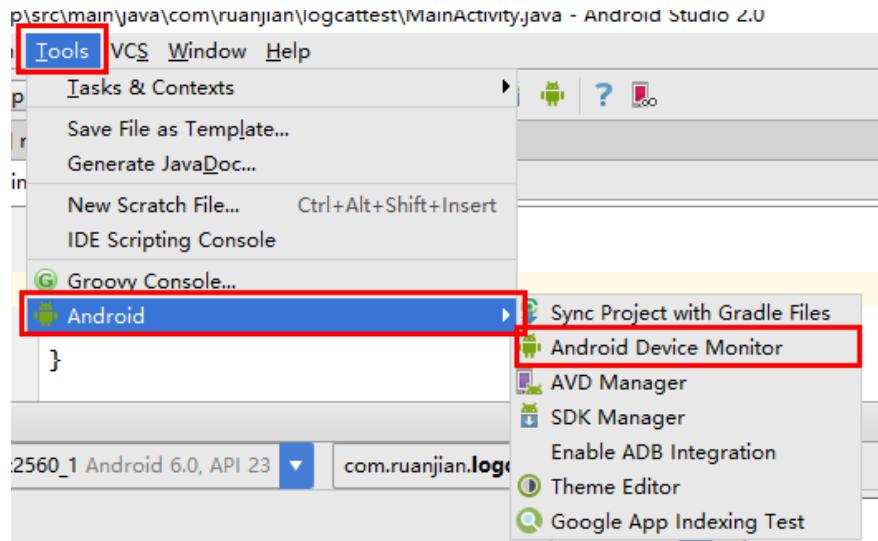


图 1.6.14



图 1.6.15

步骤十八 打开后的 DDMS 界面效果。

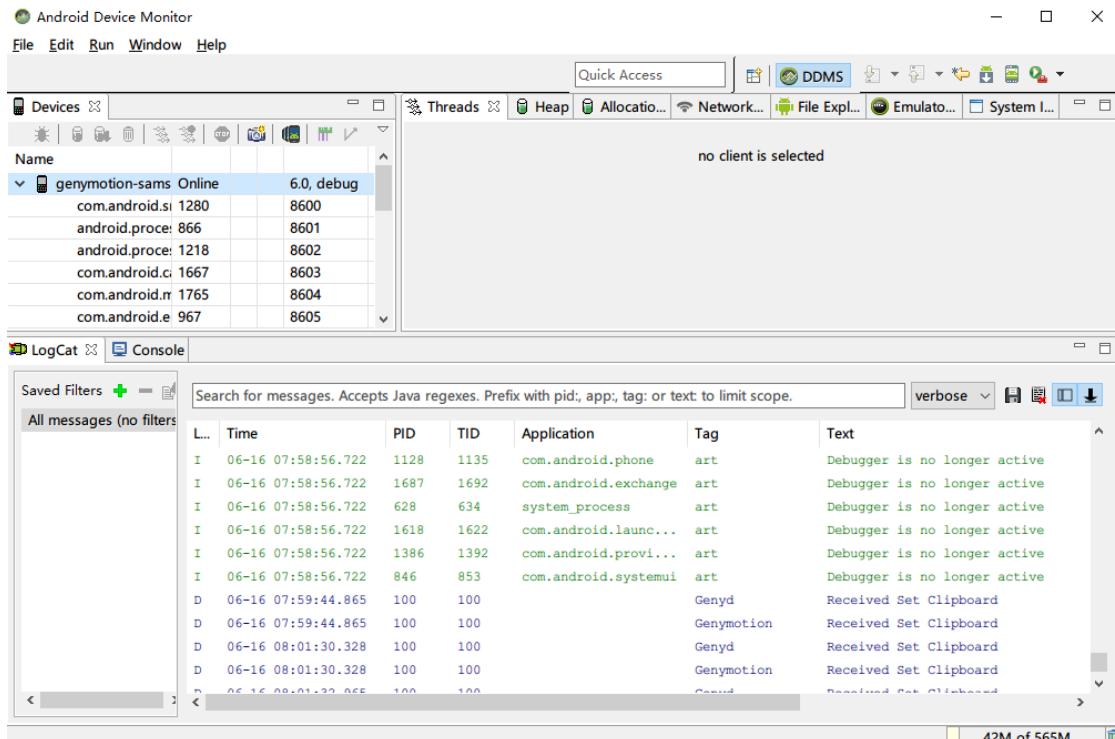


图 1.6.16

- ✓ Devices: 查看到所有与 DDMS 连接的模拟器详细信息，以及每个模拟器正在运行的 APP 进程，每个进程最右边相对应的是与调试器链接的端口。

- ✓ Emulator Control: 实现对模拟器的控制, 如: 接听电话, 根据选项模拟各种不同网络情况, 模拟短信发送及虚拟地址坐标用于测试 GPS 功能等。

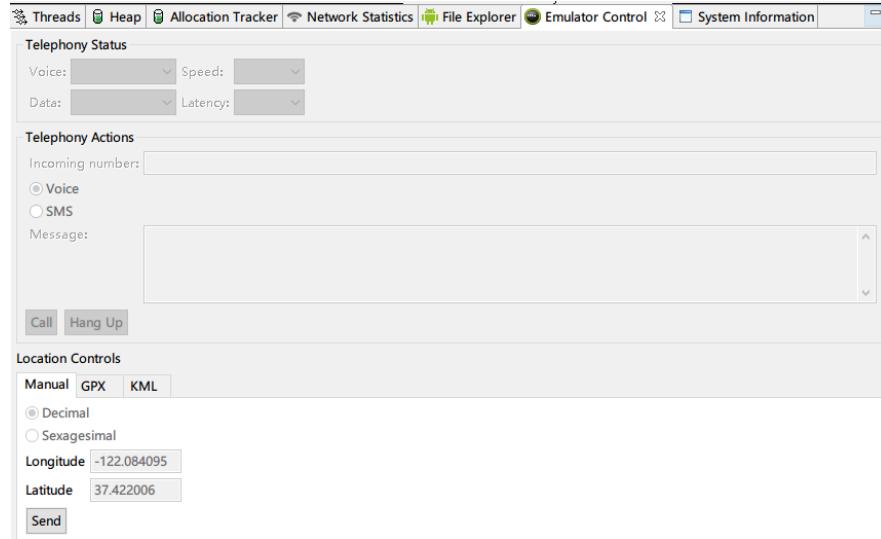


图 1.6.17

- ✓ LogCat : 查看日志输入信息, 可以对日志输入进行 Filter 过滤一些调试的信息筛选查看等。
- ✓ File Explorler:File Explorler 文件浏览器, 查看 Android 模拟器中的文件, 可以很方便的导入/出文件。 打开该界面后示意图如图所示。

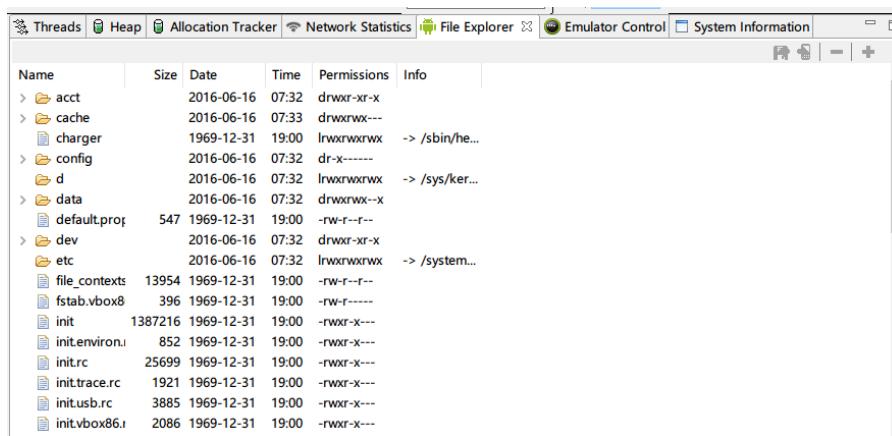


图 1.6.18

- ✓ Heap: 查看应用中内存使用情况。
- ✓ Dump HPROF file: 点击 DDMS 工具条上面的 Dump HPROF 文件按钮, 选择文件存储位置, 然后在运行 hprof-conv。可以用 MAT 分析 heap dumps 启动 MAT 然后加载刚才我们生成的 HPROF 文件。MAT 是一个强大的工具, 讲述它所有的特性超出了本文的范围, 所以我只想演示一种你可以用来检测泄露的方法: 直方图 (Histogram) 视图。它显示了一个可以排序的类实例的列表, 内

容包括: shallow heap (所有实例的内存使用总和), 或者 retained heap (所有类实例被分配的内存总和, 里面也包括他们所有引用的对象) 等。

- ✓ Screen captrue: 截屏操作
- ✓ Thread: 查看进程中线程情况。
- ✓ 其它工具。

1.6.4 实验结论

使用 LogCat 查看 Android 应用程序提示消息, 并能够查找应用程序错误, 掌握 DDMS 的基本作用。

第2章 Android 应用的界面基础

本章的主要内容是 Android 中界面与视图组件的介绍，Android 中基本界面组件的介绍，对 Android 中布局管理器的介绍，Android 中高级界面组件的介绍，以及 Android 中的对话框、提示信息、菜单的使用。

通过本章实验应该达到的目的：能够使用所学知识完成简单的注册、登陆界面，能够使用简单的 ListView，并能自定义 ListView，能够使用简单的 TabHost，并能自定义 TabHost，熟悉并掌握对话框的功能及创建方法，能够熟练应用 Toast 及菜单。

2.1 实验一 注册、登录页面布局

2.1.1 实验目的

1. 掌握 Android 中线性布局的基本用法
2. 掌握 Android 中界面布局的两种方式

2.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常

2.1.3 实验步骤

通过课堂的学习我们知道了布局有两种形式一种是 Java 代码布局，一种是使用 xml 文件布局，下面我们分别使用两种方法实现一个简单地注册页面。

➤ Java 代码布局

步骤一 在 Android Studio 中创建一个 Android 应用程序 LoginDemo1。创建

“Empty Activity” 及默认视图布局文件。完成后的应用程序项目目录如下。

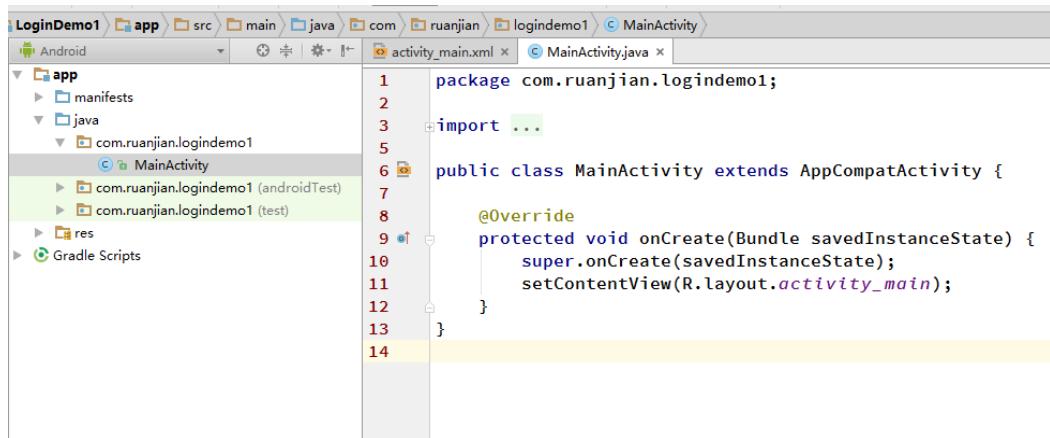


图 2.1.1

步骤二 打开应用程序项目目录中的 app\src\main\java 目录下的 MainActivity.java 文件，修改 MainActivity 类的 onCreate 方法，参考代码如下。

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        LinearLayout.LayoutParams lp = new LinearLayout.
            LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT);

        TextView title = new TextView(this);
        title.setGravity(Gravity.CENTER);
        title.setText("注册");
        layout.addView(title, lp);

        TextView textView1 = new TextView(this);
        textView1.setText("请输入邮箱:");
        layout.addView(textView1, lp);
    }
}
```

```
EditText editText1=new EditText(this);
layout.addView(editText1, lp);

TextView textView2 = new TextView(this);
textView2.setText("请输入密码:");
layout.addView(textView2, lp);

EditText editText2=new EditText(this);
layout.addView(editText2, lp);

Button btn=new Button(this);
btn.setText("提交");
layout.addView(btn, lp);

setContentView(layout);
}

}
```

步骤三 运行后效果如图所示。



图 2.1.2

➤ xml 文件布局

步骤四 在 Android Studio 中创建一个 Android 应用程序 LoginDemo2。创建“Empty Activity”及默认视图布局文件。完成后的应用程序项目目录如下。

The screenshot shows the Android Studio project structure for "LoginDemo2". The project tree includes "Android", "app", "src", "main", "java", "com", "ruanjian", "logindemo2", and "MainActivity". The "activity_main.xml" file is selected in the list. The code editor shows the following Java code for MainActivity:

```
1 package com.ruanjian.logindemo2;
2
3 import ...
4
5 public class MainActivity extends AppCompatActivity {
6     @Override
7     protected void onCreate(Bundle savedInstanceState) {
8         super.onCreate(savedInstanceState);
9         setContentView(R.layout.activity_main);
10    }
11
12 }
13
14 |
```

图 2.1.3

步骤五 打开应用程序项目目录中的 app\src\main\res\layout 目录下的 activity_main.xml 文件，修改其中代码，参考代码如下。

```
<LinearLayout 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:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="请输入账号"/>

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName" >
    </EditText>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="请输入密码"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="none" >
    </EditText>
```

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
    android:gravity="center">  
  
    <Button  
        android:layout_width="160dp"  
        android:layout_height="wrap_content"  
        android:text="登录"/>  
  
    <Button  
        android:layout_width="160dp"  
        android:layout_height="wrap_content"  
        android:text="取消"/>  
  
    </LinearLayout>  
</LinearLayout>
```

步骤六 运行后效果如图所示。



图 2.1.4

2.1.4 实验结论

通过本实验我们能够使用所学知识完成简单的注册、登录界面，掌握 Android 界面布局的两种方式，掌握 LinearLayout 的使用。

2.2 实验二 界面布局

2.2.1 实验目的

通过本实验我们可以完成包含多个简单组件的界面，并可以熟练掌握 Android 的几种布局。

2.2.2 准备工作

1. 搭建好的 Android 开发环境

2.2.3 实验步骤

在上个试验中我们完成了简单的注册登录界面，是不是感觉界面太过简单呢？下面我们就一起完成一个相对比较复杂的界面，公交车查询界面，最终要达到的界面效果如图：

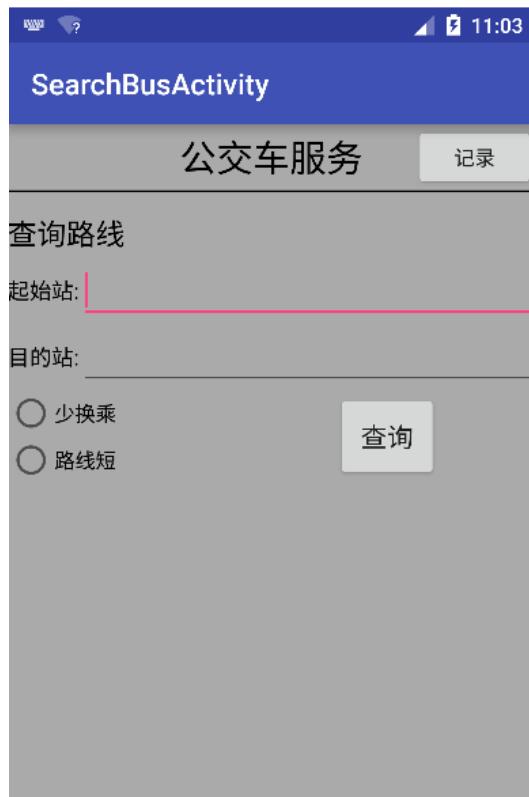


图 2.2.1

步骤一 首先让我们一起分析下它的界面布局，做成简单的示意图大致如下图所示：

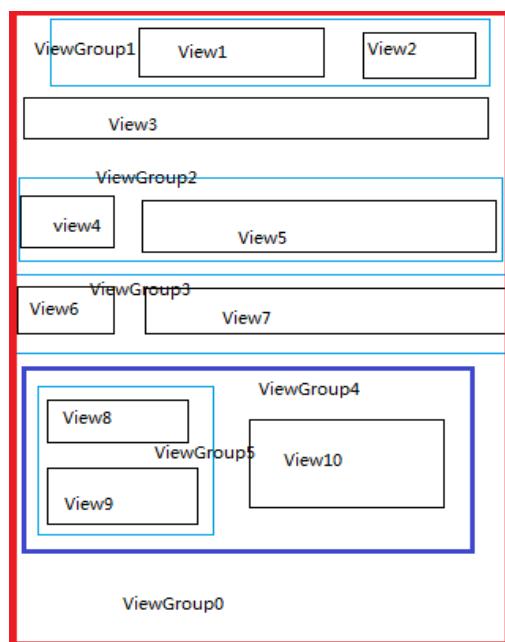


图 2.2.2

步骤二 新建工程 SearchBusActivity, 其中会有 activity_main.xml 代码如

下：

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:color/darker_gray"
    android:gravity="top|center">

    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="45dp">

        <TextView
            android:id="@+id/bustitle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@color/black"
            android:textSize="25dp"
            android:layout_marginLeft="120dp"
            android:layout_centerInParent="true"
            android:text="@string/busheading" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_toRightOf="@+id/bustitle"
            android:layout_marginLeft="35dp"
            android:text="@string/record" />
    
```

</RelativeLayout>

```
<TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="1dp"
        android:background="@color/black"/>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="15dp"/>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="30dp"
        android:layout_gravity="center_horizontal"
        android:textSize="20dp"
        android:textColor="@color/black"
        android:text="@string/bussearch" />

    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/busstart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:textColor="@color/black"
            android:textSize="15dp"
            android:text="@string/searchstart" />

        <EditText
            android:inputType="none"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@+id/busstart"
```

```
        android:layout_marginLeft="0dp"
        android:textColor="@color/black"/></RelativeLayout>

<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/busend"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:textColor="@color/black"
        android:textSize="15dp"
        android:text="@string/searchend" />
    <EditText
        android:inputType="none"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/busend"
        android:layout_marginLeft="0dp"
        android:textColor="@color/black"/></RelativeLayout>

<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <RadioGroup android:id="@+id/RadioGroup01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <RadioButton android:id="@+id/RadioButton01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
```

```
        android:textColor="@color/black"
        android:text="@string/searchset1" >
    </RadioButton>
    <RadioButton android:id="@+id/RadioButton02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/black"
        android:text="@string/searchset2" >
    </RadioButton></RadioGroup>
<Button
    android:id="@+id/searchbtn"
    android:layout_width="70dp"
    android:layout_height="60dp"
    android:layout_toRightOf="@+id/RadioGroup01"
    android:layout_marginLeft="150dp"
    android:layout_centerVertical="true"
    android:textSize="18dp"
    android:text="@string/search"/>
</RelativeLayout>
</LinearLayout>
```

步骤三 资源文件 string.xml 为：

```
<resources>
    <string name="app_name">SearchBusActivity</string>
    <string name="busheading">公交车服务</string>
    <string name="record">记录</string>
    <string name="bussearch">查询路线</string>
```

```
<string name="searchstart">起始站:</string>
<string name="searchend">目的站:</string>
<string name="searchset1">少换乘</string>
<string name="searchset2">路线短</string>
<string name="search">查询</string>
</resources>
```

步骤四 资源文件 colors.xml 为：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="black">#000000</color>
</resources>
```

步骤五 运行工程效果如图，大家还可以在此基础上进行美化：

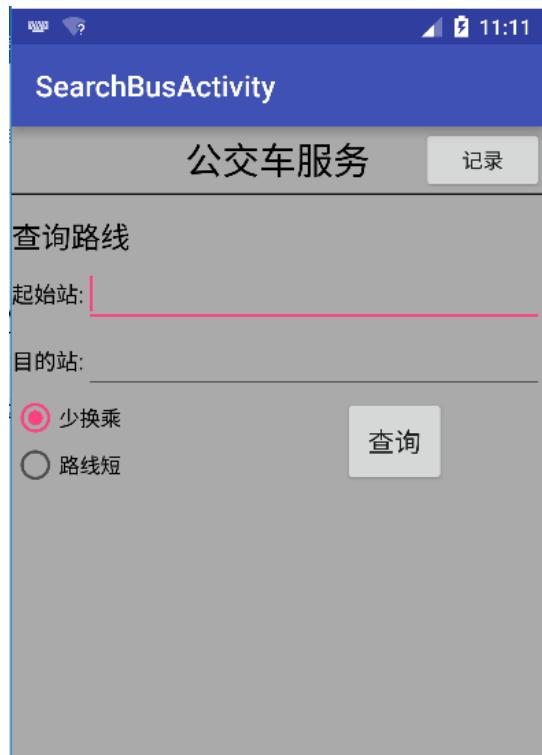


图 2.2.3

2.2.4 实验结论

通过本实验我们可以熟练运用几种布局，完成较复杂的界面。

2.3 实验三 ListView 的使用

2.3.1 实验目的

通过本实验我们可以使用 Adapter 定义简单的 ListView，并可以自定义我们自己形式的 ListView。

2.3.2 准备工作

准备一： 搭建好的 Android 开发环境

2.3.3 实验步骤

ListView 是我们 Android 开发中用的很多的一种组件，这次我们一起来学习 ListView 的用法，ListView 不仅用的频率比较高而且其样式也多种多样，如下图：



图 2.3.1

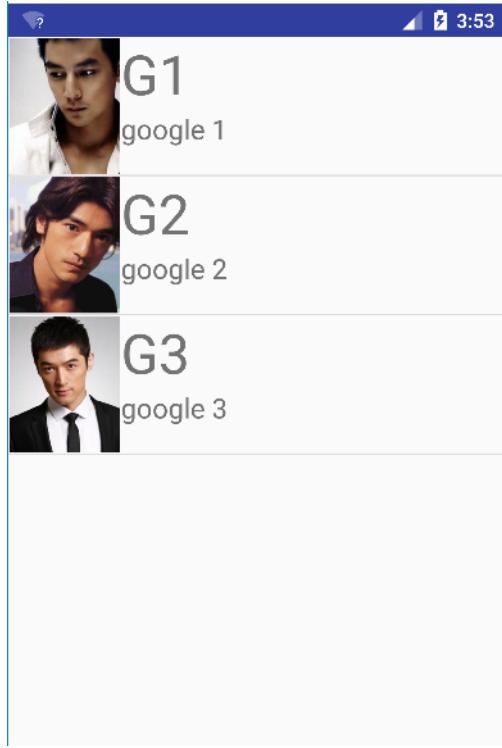


图 2.3.2

步骤一 下面我们一起来实现上图中 ListView 的效果，新建工程 CustomListView，修改 MainActivity.java 此文件的代码如下：此文件主要内容就是得到两个按钮，分别跳转到两个 ListView 的 Activity。

```
package com.ruanjian.customlistview;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
    private Button myListviewBtn;
    private Button simpleAdapterBtn;
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    findViews();
    setListeners();
}

private void findViews() {
    myListviewBtn = (Button) findViewById(R.id.mylistview_btn);
    simpleAdapterBtn = (Button) findViewById(R.id.simpleAdapter_btn);
}

private void setListeners() {
    myListviewBtn.setOnClickListener(myListviewBtnListener);
    simpleAdapterBtn.setOnClickListener(simpleAdapterBtnListener);
}

private View.OnClickListener simpleAdapterBtnListener = new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.v("MyListView2", "simpleAdapter");
        Intent intent = new Intent(MainActivity.this, myListview2.class);
        startActivity(intent);
    }
};

private View.OnClickListener myListviewBtnListener = new

```

```
View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Log.v("MyListView", "myListViewBtnListener");  
        ComponentName comp = new ComponentName(MainActivity.this,  
                myListview1.class);  
        Intent intent = new Intent();  
        intent.setComponent(comp);  
        startActivity(intent);  
    }  
};  
}  
}
```

步骤二 根据 Activity 的知识可知，此 Activity 采用 xml 布局，所以大家可以在 activity_main.xml 中修改布局代码：如下：

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" >  
    <TextView  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/hello"/>  
    <Button  
        android:id="@+id/mylistview_btn"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/s_mylistview_btn"
```

```
    android:allowUndo="false" />

    <Button
        android:id="@+id/simpleAdapter_btn"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/s_simpleAdapter_btn"/>
</LinearLayout>
```

步骤三 新创建一个类 myListview1, 其中 myListview1. java 中的代码如下:

```
package com.ruanjian.customlistview;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.List;

public class myListview1 extends Activity {
    private ListView listView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.simple_adapter);
        listView = (ListView) findViewById(R.id.listView);
        String[] values = {"Hello", "Android", "Custom", "List", "View", "Tutorial"};
        ArrayAdapter adapter = new ArrayAdapter(this, R.layout.simple_adapter_item, values);
        listView.setAdapter(adapter);
    }
}
```

```
listView = new ListView(this);

listView.setAdapter(new ArrayAdapter<String>(this,
    android.R.layout.simple_expandable_list_item_1,
    getData()));

listView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> arg0,
        View arg1,
        int arg2,
        long arg3) {
        Log.e("listViewItem", arg2+"");
    }
});

setContentView(listView);
}

private List<String> getData() {
    List<String> data = new ArrayList<String>();
    data.add("测试数据 1");
    data.add("测试数据 2");
    data.add("测试数据 3");
    data.add("测试数据 4");

    return data;
}
}
```

步骤四 此文件没有对应的 xml 文件, 通过 `setContentView(listView);` ; 设置 `listview` 为此 activity 的中心视图。此效果就是我们上面提到的第一种效果



图 2.3.3

步骤五 新创建一个类 `myListview2`, 其中 `myListview2.java` 中的代码如下:

```
package com.ruanjian.customlistview;

import android.app.ListActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ListView;
import android.widget.SimpleAdapter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```

public class myListview2 extends ListActivity {

    @Override

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        SimpleAdapter adapter = new SimpleAdapter(this,
            getData(),
            R.layout.activity_mylistview2,
            new String[]{"title", "info", "img"},
            new int[]{R.id.title, R.id.info, R.id.img});

        setListAdapter(adapter);
    }

    @Override

    protected void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);

        Log.e("MyListView2", "MyListView2");

    }

    private List<Map<String, Object>> getData() {
        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();

        Map<String, Object> map = new HashMap<String, Object>();
        map.put("title", "G1");
        map.put("info", "google 1");
        map.put("img", R.drawable.i1);
        list.add(map);

        map = new HashMap<String, Object>();
        map.put("title", "G2");
        map.put("info", "google 2");
    }
}

```

```

        map.put("img", R.drawable.i2);

        list.add(map);

        map = new HashMap<String, Object>();

        map.put("title", "G3");

        map.put("info", "google 3");

        map.put("img", R.drawable.i3);

        list.add(map);

        return list;
    }
}

```

步骤六 此 java 文件为一个 ListActivity 的子类，此类和 Activity 类用法基本相同，只是此类不需要 xml 布局，其中心为一个固定 ListView。在其中通过一个 xml 布局文件来定义 ListView 的每个条目的样式，xml 内容如下：

```

SimpleAdapter adapter = new SimpleAdapter(this,
    getData(),
    R.layout.activity_mylistview2,
    new String[]{"title", "info", "img"},
    new int[]{R.id.title, R.id.info, R.id.img});

```

步骤七 对应 activity_mylistview2.xml 代码如下：

```

<?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="100dp">

```

```
<ImageView android:id="@+id/img"
    android:layout_width="80dp"
    android:layout_height="match_parent"
    android:layout_margin="3px"
    android:scaleType="centerCrop" />

<LinearLayout android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="40sp" />

    <TextView
        android:id="@+id/info"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp" />

</LinearLayout>
</LinearLayout>
```

步骤八 准备三张图片 i1.jpg、i2.jpg、i3.jpg 放在 res\drawable 下，运行后效果如图：

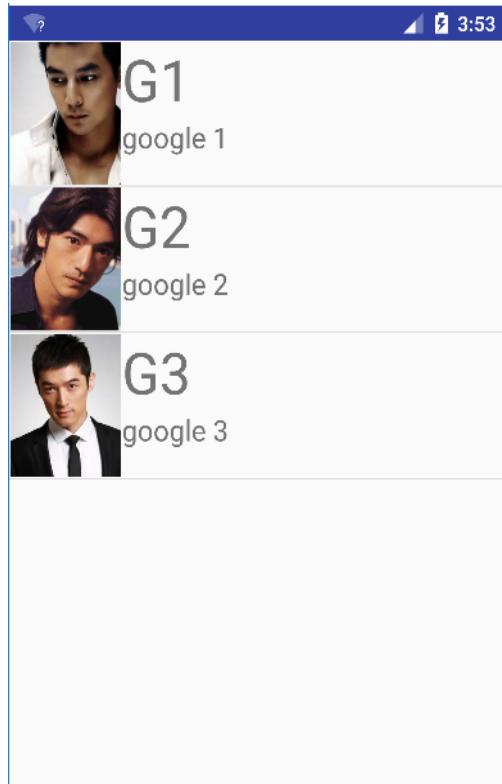


图 2.3.4

步骤九 res\values\strings.xml 中修改如下：

```
<resources>
    <string name="app_name">CustomListView</string>
    <string name="hello">ListView Tutorial</string>
    <string name="s_mylistview_btn">简单的 ListView</string>
    <string name="s_simpleAdapter_btn">SimpleAdapter</string>
</resources>
```

步骤十 AndroidManifest.xml 中修改如下：

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

```
    android:label="@string/app_name"  
    android:supportsRtl="true"  
    android:theme="@style/AppTheme">  
        <activity android:name=".MainActivity">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
        <activity android:name=".myListview1"/>  
        <activity android:name=".myListview2"/>  
    </application>  
  
</manifest>
```

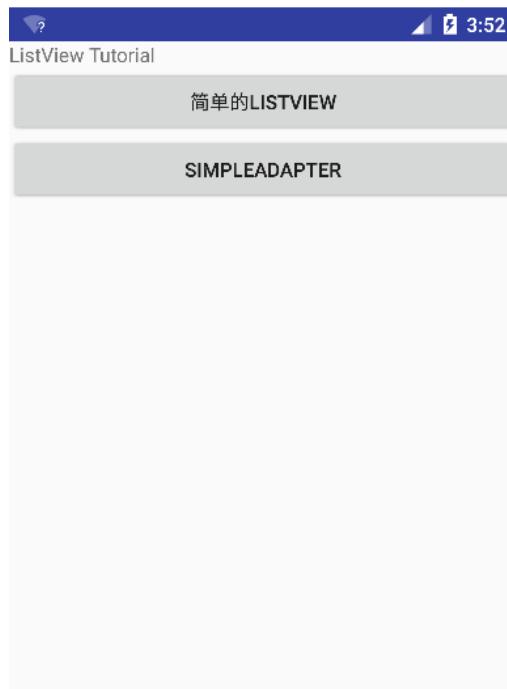


图 2.3.5



图 2.3.6

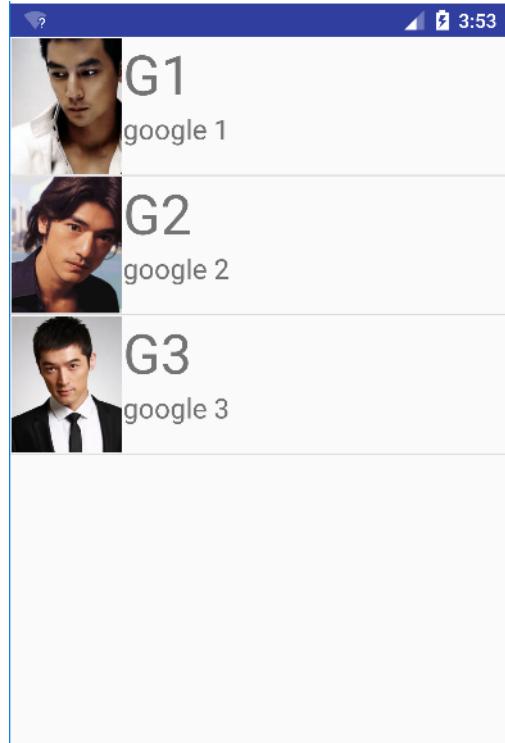


图 2.3.7

2.3.4 实验结论

通过本实验我们学会了定义简单的 ListView，并可以根据自己的需要自定义 ListView。

2.4 实验四 通知消息和 Toast

2.4.1 实验目的

1. 掌握 Android 中的 Toast 用法。

2.4.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

2.4.3 实验步骤

Toast 是 Android 中一个用的非常频繁的组件，在 Android 应用中有很多信息需要告知用户，而有些信息对不同用户而言又或许不是太重要，我们没必要每次提示消息都显示一个提示框出来，这时候我们就需要用到 Toast 了

步骤一 新建工程 ToastDemo，在 MainActivity.java 中创建 Toast。其代码如下：

```
package com.ruanjian.toastdemo;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.LinearLayout;
```

```
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity implements View.OnClickListener {

    Handler handler = new Handler();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.btnSimpleToast).setOnClickListener(this);

        findViewById(R.id.btnSimpleToastWithCustomPosition).setOnClickListener(this);
        findViewById(R.id.btnSimpleToastWithImage).setOnClickListener(this);
        findViewById(R.id.btnCustomToast).setOnClickListener(this);
        findViewById(R.id.btnRunToastFromOtherThread).setOnClickListener(this);
    }

    public void showToast() {
        handler.post(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(getApplicationContext(), R.string.other_threads,
                        Toast.LENGTH_SHORT).show();
            }
        });
    }

    @Override
```

```
public void onClick(View v) {  
  
    Toast toast = null;  
  
    switch (v.getId()) {  
  
        case R.id.btnSimpleToast:  
  
            Toast.makeText(getApplicationContext(),  
                R.string.simple,  
                Toast.LENGTH_SHORT).show();  
  
            break;  
  
        case R.id.btnSimpleToastWithCustomPosition:  
  
            toast = Toast.makeText(getApplicationContext(),  
                R.string.simple_position,  
                Toast.LENGTH_LONG);  
  
            toast.setGravity(Gravity.CENTER, 0, 0);  
  
            toast.show();  
  
            break;  
  
        case R.id.btnSimpleToastWithImage:  
  
            toast = Toast.makeText(getApplicationContext(),  
                R.string.simple_image,  
                Toast.LENGTH_LONG);  
  
            toast.setGravity(Gravity.CENTER, 0, 0);  
  
            LinearLayout toastView = (LinearLayout) toast.getView();  
  
            ImageView imageCodeProject = new  
            ImageView(getApplicationContext());  
  
            imageCodeProject.setImageResource(R.drawable.ic_launcher);  
  
            toastView.addView(imageCodeProject, 0);  
  
            toast.show();  
  
            break;  
    }  
}
```

```
case R.id.btnCustomToast:  
    LayoutInflater inflater = getLayoutInflater();  
    View layout = inflater.inflate(R.layout.custom,  
        (ViewGroup) findViewById(R.id.llToast));  
    ImageView image = (ImageView) layout  
        .findViewById(R.id.tvImageToast);  
    image.setImageResource(R.drawable.ic_launcher);  
    TextView title = (TextView)  
        layout.findViewById(R.id.tvTitleToast);  
    title.setText(R.string.simple_custom_title);  
    TextView text = (TextView) layout.findViewById(R.id.tvTextToast);  
    text.setText(R.string.simple_custom);  
    toast = new Toast(getApplicationContext());  
    toast.setGravity(Gravity.RIGHT | Gravity.TOP, 12, 40);  
    toast.setDuration(Toast.LENGTH_LONG);  
    toast.setView(layout);  
    toast.show();  
    break;  
  
case R.id.btnRunToastFromOtherThread:  
    showToast();  
    break;  
}  
}  
}
```

步骤二 编辑布局文件 activity_main.xml，采用流式布局，主要代码如下：

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:padding="5dip"
    android:gravity="center">

    <Button android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:id="@+id	btnSimpleToast"
        android:text="@string/btn_simple">

    </Button>

    <Button android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="@string/btn_simple_position"
        android:id="@+id	btnSimpleToastWithCustomPosition">

    </Button>

    <Button android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:id="@+id	btnSimpleToastWithImage"
        android:text="@string/btn_simple_image">

    </Button>

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

```
        android:text="@string/btn_simple_custom"
        android:id="@+id(btnCustomToast">
    </Button>

    <Button android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:text="@string/btn_other_threads"
            android:id="@+id	btnRunToastFromOtherThread">
    </Button>
</LinearLayout>
```

步骤三 在此示例中自定义的 Toast 的样式对应的 xml 文件是 custom.xml，同样使用流式布局（LinearLayout）主要代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="#ffffffff"
    android:orientation="vertical"
    android:id="@+id/llToast" >

    <TextView
        android:layout_height="wrap_content"
        android:layout_margin="1dip"
        android:textColor="#ffffffff"
        android:layout_width="fill_parent"
        android:gravity="center"
```

```
    android:background="#bb000000"

    android:id="@+id/tvTitleToast" />

<LinearLayout

    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/l1ToastContent"
    android:layout_marginLeft="1dip"
    android:layout_marginRight="1dip"
    android:layout_marginBottom="1dip"
    android:layout_width="wrap_content"
    android:padding="15dip"
    android:background="#44000000" >

    <ImageView

        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:id="@+id/tvImageToast" />

    <TextView

        android:layout_height="wrap_content"
        android:paddingRight="10dip"
        android:paddingLeft="10dip"
        android:layout_width="wrap_content"
        android:gravity="center"
        android:textColor="#ff000000"
        android:id="@+id/tvTextToast" />

</LinearLayout>
```

```
</LinearLayout>
```

步骤四 编辑 string.xml 文件，在其中添加如下字符串资源。

```
<resources>

    <string name="app_name">ToastDemo</string>

    <string name="simple">默认 Toast 样式</string>

    <string name="simple_position">自定义显示位置</string>

    <string name="simple_image">带图片的 Toast</string>

    <string name="simple_custom">完全自定义 Toast</string>

    <string name="simple_custom_title">Attention</string>

    <string name="other_threads">我来自其他线程</string>

    <string name="btn_simple">默认</string>

    <string name="btn_simple_position">自定义显示位置</string>

    <string name="btn_simple_image">带图片</string>

    <string name="btn_simple_custom">完全自定义</string>

    <string name="btn_other_threads">其他线程</string>

</resources>
```

步骤五 准备一张图片命名为 ic_launcher.jpg，并放在 app\src\main\res\drawable 目录下。运行该项目，界面效果如图所示。



图 2.4.1



图 2.4.2



图 2.4.3



图 2.4.4

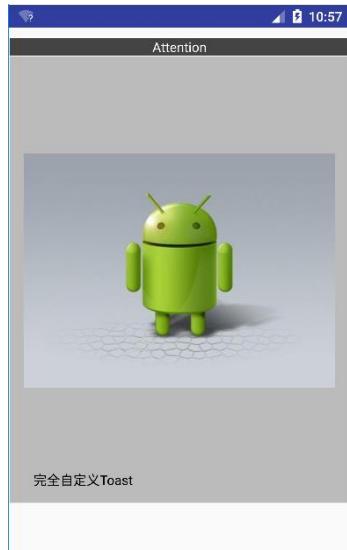


图 2.4.5



图 2.4.6

2.4.4 实验结论

通过本实验我们可以熟练掌握和使用 Toast，并可以根据需要自定义 Toast。

2.5 实验五 对话框的使用

2.5.1 实验目的

1. 通过本实验我们可以熟练掌握和使用 AlertDialog。

2.5.2 准备工作

1. 搭建好的 Android 开发环境

2.5.3 实验步骤

在我们 Android 应用中有很多程序反馈回来的信息，这些信息都是需要用户来处理的，这时候就需要用到 Android 中的对话框了。

步骤一 新建工程 AlertDialogDemo，其中会有 MainActivity.java，此文件的代码如下：主要在此 activity 中创建 AlertDialog。

```
package com.ruanjian.alertdialogdemo;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
    private Button alertBtn;
    @Override
```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    alertBtn = (Button) findViewById(R.id.alertBtn);  
    alertBtn.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            AlertDialog.Builder builder  
                    = new AlertDialog.Builder(MainActivity.this);  
            builder.setIcon(R.drawable.ic_launcher);  
            builder.setTitle("你确定要离开吗？");  
            builder.setPositiveButton("确定",  
                    new DialogInterface.OnClickListener() {  
                        public void onClick(DialogInterface dialog, int  
                                whichButton) {  
                            //这里添加点击确定后的逻辑  
                            Toast.makeText(MainActivity.this,  
                                    "你选择了确定",  
                                    Toast.LENGTH_SHORT)  
                                    .show();  
                        }  
                });  
  
            builder.setNegativeButton("取消",  
                    new DialogInterface.OnClickListener() {  
                        public void onClick(DialogInterface dialog,  
                                int whichButton) {  
                            Toast.makeText(MainActivity.this,
```

```
        "你选择了取消",
        Toast.LENGTH_SHORT)
        .show();
    }

    builder.create().show();
}

})
}
}
```

步骤二 对应的 activity_main.xml 文件，代码如下：

```
<?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:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello AlertDialog" />

    <Button
        android:id="@+id/alertBtn"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="普通 AlertDialog"/>

```

```
</LinearLayout>
```

步骤三 准备一张图片命名为 ic_launcher.jpg，并放在 app\src\main\res\drawable 目录下。运行该项目，界面效果如图所示。



图 2.5.1

2.5.4 实验结论

通过实验了解了 AlertDialog 一般用途，并能在程序中使用 AlertDialog。

2.6 实验六 菜单的使用

2.6.1 实验目的

- 通过本实验我们可以熟练掌握和使用 Android 中菜单。

2.6.2 准备工作

- 搭建好的 Android 开发环境

2.6.3 实验步骤

菜单 Menu 是 Android 中又一个用的比较多的组件，对手机屏幕而言，分辨率是手机开发无法逾越的障碍，并不是所有的应用都适合在界面上设置一些按钮的，比如画图、地图搜索，这种情况下使用 Menu 设置一些常用的选项再合适不过了。

步骤一 新建工程 MenuDemo，里面有 MainActivity.java 代码如下：

```
package com.ruanjian.menudemo;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.ImageView;

public class MainActivity extends Activity {
    private ImageView img;
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

img=(ImageView)findViewById(R.id.img);

}

public boolean onCreateOptionsMenu(Menu menu) {

    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {

    switch(item.getItemId()) {

        case R.id.menu_1:
            img.setImageResource(R.drawable.grey);
            break;

        case R.id.menu_2:
            img.setImageResource(R.drawable.red);
            break;

        case R.id.menu_3:
            img.setImageResource(R.drawable.black);
            break;

        case R.id.menu_4:
            this.finish(); break;
    }
    return true;
}
}

```

步骤二 其中修改 activity_main.xml 文件的代码如下：

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

<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"
    tools:context="com.ruanjian.menudemo.MainActivity">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/img"
        android:scaleType="fitXY"/>

</RelativeLayout>
```

步骤三 在 app\src\main\res 目录下新建 menu 目录，并在其中添加 menu.xml 文件， menu.xml 文件的代码如下：

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<group  
    android:enabled="true"  
    android:id="@+id/main_group" >  
  
<item  
    android:id="@+id/menu_1"  
    android:title="灰色" />  
  
<item  
    android:id="@+id/menu_2"  
    android:title="红色" />  
  
<item  
    android:id="@+id/menu_3"  
    android:title="黑色" />  
  
<item  
    android:id="@+id/menu_4"  
    android:title="退出" />  
  
</group>  
</menu>
```

步骤四 在 app\src\main\res\drawable 目录下添加 black.jpg、grey.jpg、red.jpg 三张纯色图片，运行工程后点开菜单效果如下：

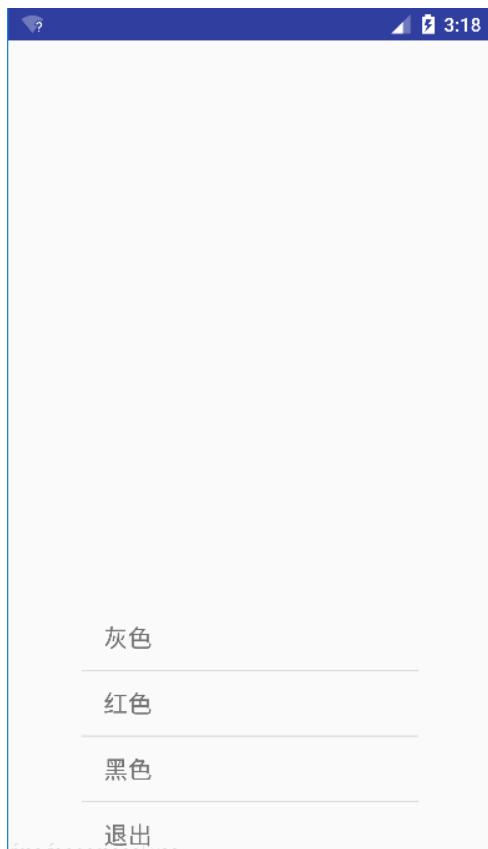


图 2.6.1

步骤五 点击 menu 菜单中的红色，屏幕背景会变成红色。

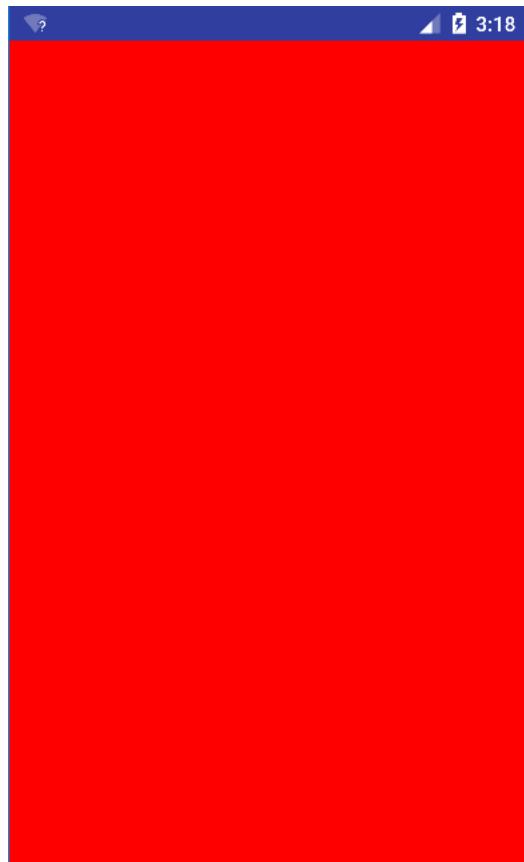


图 2.6.2

2.6.4 实验结论

通过本实验我们掌握了如何构建和使用 menu。

2.7 作业

2.7.1 完成学生信息注册的界面

要求在注册页面，注册人员可以输入姓名，性别，班级，联系方式，下面有提交信息的按钮。

2.7.2 完成学生信息登录的界面

要求登录界面登录人员可以输入用户名及密码，下面要有登录按钮，退出按钮。

2.7.3 完成学生信息的列表页面

要求学生信息的列表页面，每个列表要包含学生的姓名、班级。

2.7.4 将如上三个页面放入一个 TabHost 中

要求 Tabhost 包含三个选项卡，点击第一个选项卡进入注册页面，点击第二个进入登录页面，点击第三个进入列表页面。

2.7.5 退出系统时显示对话框确认是否退出

要求退出系统时显示对话框确认是否退出，点击确定退出页面，点击取消返回页面。

2.7.6 输入信息的校验使用 Toast 提示错误信息

要求当输入格式错误时会出现 Toast 提示错误，例如：你的输入有误，手机号码只能为 11 位数字。

第3章 Android 事件处理机制

本章主要内容有对 Android 中事件处理机制的介绍，对 Android 中基于监听的事件处理的介绍，对 Android 中基于回调的事件处理的介绍，对 Android 中相应的系统设置的事件的介绍，对 Android 中的 Handler 消息传递机制（多线程）的介绍。

通过本章实验我们能够理解 Android 中事件监听的机制，理解 Android 中回调函数的机制，掌握 Android 中多线程之间的消息传递机制。

3.1 实验一 基于回调机制的事件处理

3.1.1 实验目的

1. 掌握 Android 中事件监听机制的基本原理。
2. 掌握回调机制事件处理的方法。

3.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

3.1.3 实验步骤

➤ 任务：练习自定义回调事件。

步骤一：

新建工程 CallbackDemo，自动创建 Activity，名称为 CallbackDemo。

步骤二：

Activity 对应的布局文件不做修改。添加 2 个 Button 按钮，设置自定义的

点击事件。

布局文件（部分）如下所示：

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="第一个按钮"  
    android:id="@+id/button"  
    android:onClick="onClick"  
    android:layout_below="@+id/textView" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="第二个按钮"  
    android:id="@+id/button2"  
    android:onClick="onClick"  
    android:layout_below="@+id/button" />
```

步骤三：

编辑 CallbackDemo.java 文件，获取布局文件的按钮，并实现 onClick() 回调方法，主要代码如下：

```
public class CallbackDemo extends AppCompatActivity {
    private Button btnFirst;
    private Button btnSecond;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    // 注意 这里没有 @Override 标签
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                Toast.makeText(this, "第一个按钮点击",
Toast.LENGTH_SHORT).show();
                break;
            case R.id.button2:
                Toast.makeText(this, "第二个按钮点击",
Toast.LENGTH_SHORT).show();
                break;
            default:
                break;
        }
    }
}
```

步骤四：

运行工程，效果如下图所示。

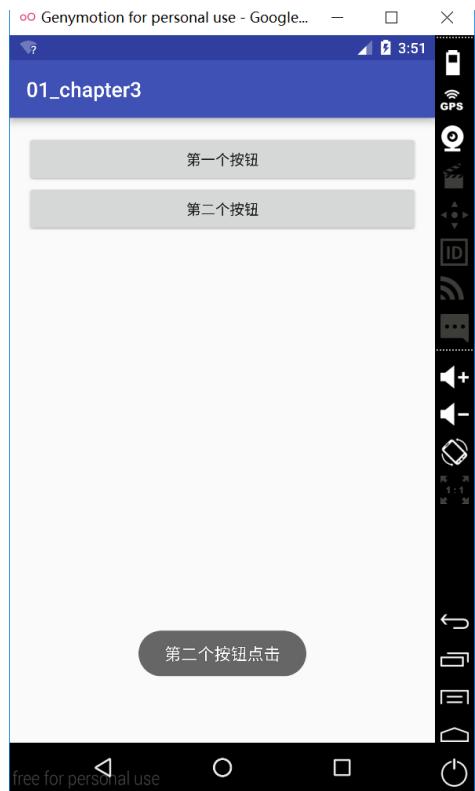


图 3.1.1

3.1.4 实验结论

3.2 实验二 基于监听机制的事件处理

3.2.1 实验目的

1. 掌握 Android 中事件监听机制的基本原理。

3.2.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

3.2.3 实验步骤

➤ 任务：练习 Button 的点击事件。

步骤一：

不管是桌面应用还是手机应用程序，面对最多的就是用户，经常需要处理的就是用户动作——也就是需要为用户动作提供响应，这种为用户动作提供响应的机制就是事件处理。Android 提供了强大的事件处理机制，包括两套事件处理机制：基于监听的事件处理和基于回调的事件处理。

新建工程 ButtonDemo，自动创建 Activity。

步骤二：

拷贝 down.png 图片到工程目录 mipmap-hdpi 目录下。

编辑对应的布局文件，采用相对布局，主要代码如下。

```
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="net.onest.lww.MainActivity" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:id="@+id/textView" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_below="@+id/textView" />
    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageButton"
        android:layout_below="@+id/button" />
</RelativeLayout>
```

步骤三：

在 onCreate() 方法中获取各组件。

```
Button button = (Button) findViewById(R.id.button);
ImageButton imageView=(ImageButton) findViewById(R.id.imageButton);
button.setText("Button 按钮");
imageView.setImageResource(R.mipmap.down);
final TextView textView = (TextView) findViewById(R.id.textView);
```

给按钮注册事件监听器。

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        textView.setText("Button 按钮");
    }
});
imageButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        textView.setText("ImageButton 按钮");
    }
});
```

步骤四：

运行工程监听不同的 Button 的点击事件，Text 会显示不同的文本，效果如下图所示。

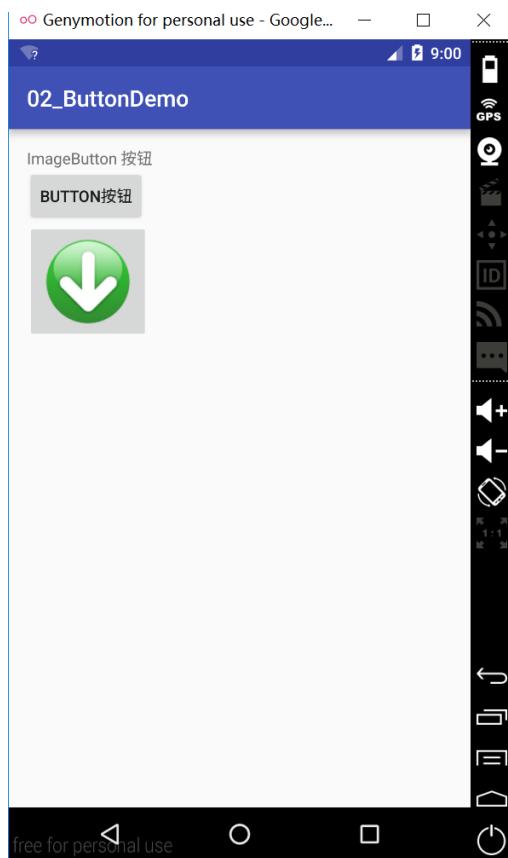


图 3.2.1

3.2.4 实验结论

3.3 实验三 Android 中的多线程 Handler 方式实现

3.3.1 实验目的

1. 了解 Android 中多线程实现的基本原理。
2. 掌握 Handle 机制实现多线程的方法。

3.3.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

3.3.3 实验步骤

➤ 任务：采用 Handle 机制实现多线程修改主线程的界面。

在 Android 中有时当程序执行到某一步骤的时我们需要修改 UI，即 Android 中的 MainThread，对于开发人员而言并不总是可以直接修改 UI 的，比如程序的 UI 已加载完毕，这时候就用到多线程了，让程序中的子线程向主线程发送请求，然后由主线程更新 UI。

本次任务我们一起来做一个多线程例子，本次实验的原理是子线程发送请求给主线程，主线程更新界面。

步骤一：

新建工程 ThreadDemo，并自动创建 Activity。

步骤二：

修改 res/values/strings.xml，添加一个字符串常量 hello。

```
<resources>
    <string name="app_name">03_ThreadDemo</string>
    <string name="hello">Hello World</string>
</resources>
```

步骤三：

修改布局文件中的文本框的属性，主要代码如下。

```
<TextView
    android:id="@+id/tv"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

步骤四：

编辑 MainActivity.java 文件，首先声明一个 TextView 组件，在 onCreate() 方法中获取布局文件中的 TextView 组件，代码如下：

```
private TextView textView;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    textView=(TextView) findViewById(R.id.tv);
}
```

步骤四：

在 Activity 中实例化 Handle，并实例化新线程。

```
//实例化Handler
Handler mHandler = new Handler() {
    //接收子线程发来的消息，同时更新UI
    public void handleMessage(Message msg) {
        textView.setText("歌词下载完成。");
    }
};

//实例化新的线程
private Thread checkUpdate = new Thread() {
    @Override
    public void run() {
        Looper.prepare();
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //向主线程发送消息
        mHandler.sendMessage(mHandler.obtainMessage());
    }
};
```

步骤五：

在 onCreate() 方法中启动线程。

```
checkUpdate.start();
textView.setText("准备下载...");
```

步骤六：

运行工程，效果如下图所示。

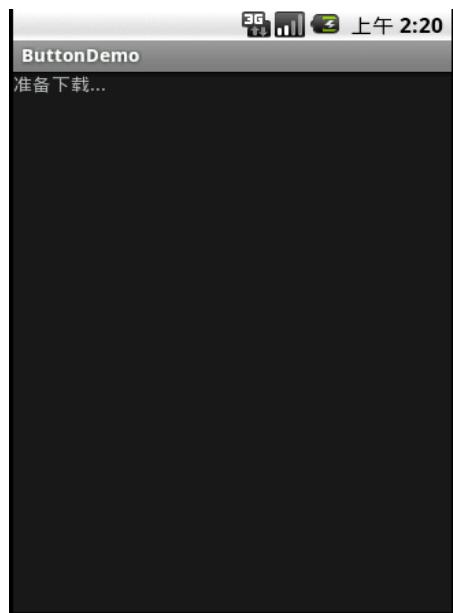


图 3.3.1



图 3.3.2

注：本任务中，并没有把启动线程的操作放在按钮的点击事件下，也没有真正的下载网络资源，只是简单实现了多线程机制的基本操作方法。

3.3.4 实验结论

通过实验，我们熟悉了 Android 中多线程之间的消息传递机制，并能实现简单的线程间的通信。

3.4 作业

3.4.1 两种事件功能

优化第二章作业 2.13.2，使用回调事件和监听器事件两种事件处理方式实现登陆功能。

3.4.2 多线程任务

修改 3.4.1 中的登陆功能，要求点击登陆按钮后，使用多线程请求 www.baidu.com，并使得返回的数据在编辑框或文本框中显示。

第4章 Android 中的 Activity 的使用

本章主要内容有 Android 中 Activity 机制，Android 中 Activity 的生命周期，Android 中 Activity 之间的跳转及数据传递。

通过本章实验我们能够理解 Android 中 Activity 机制，理解 Android 中 Activity 的生命周期，掌握 Android 中 Activity 之间的跳转及数据传递。

4.1 实验一 Activity 创建

4.1.1 实验目的

1. 掌握 Android 中创建 Activity 的方法。

4.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

4.1.3 实验步骤

➤ 方法一：

打开 AndroidStudio，新建一个 Android 项目。如图 4.1.1 所示。

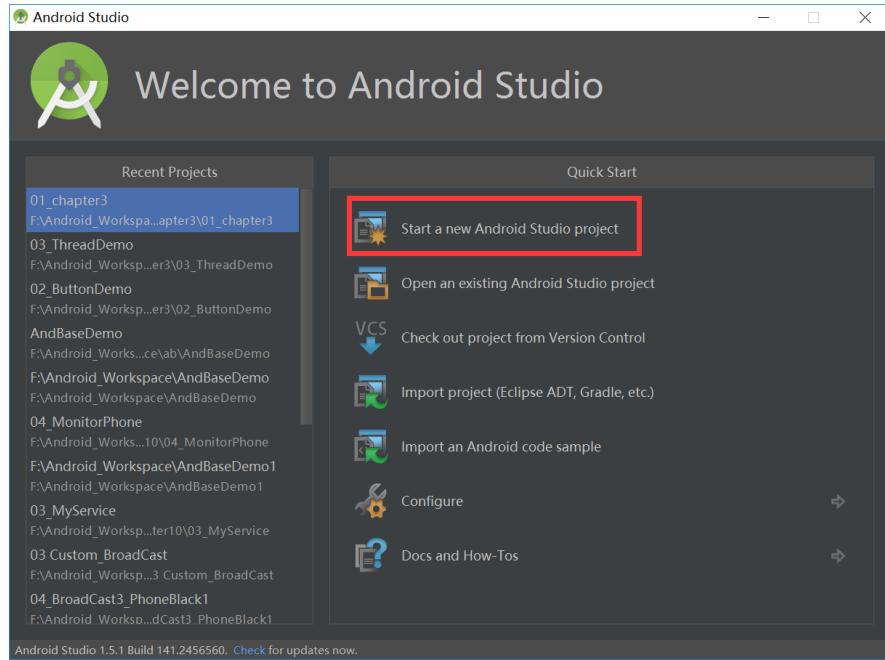


图 4.1.1

输入项目的名称，如 `AndroidActivityDemo`，点击 `Next`。如图 4.1.2 所示。

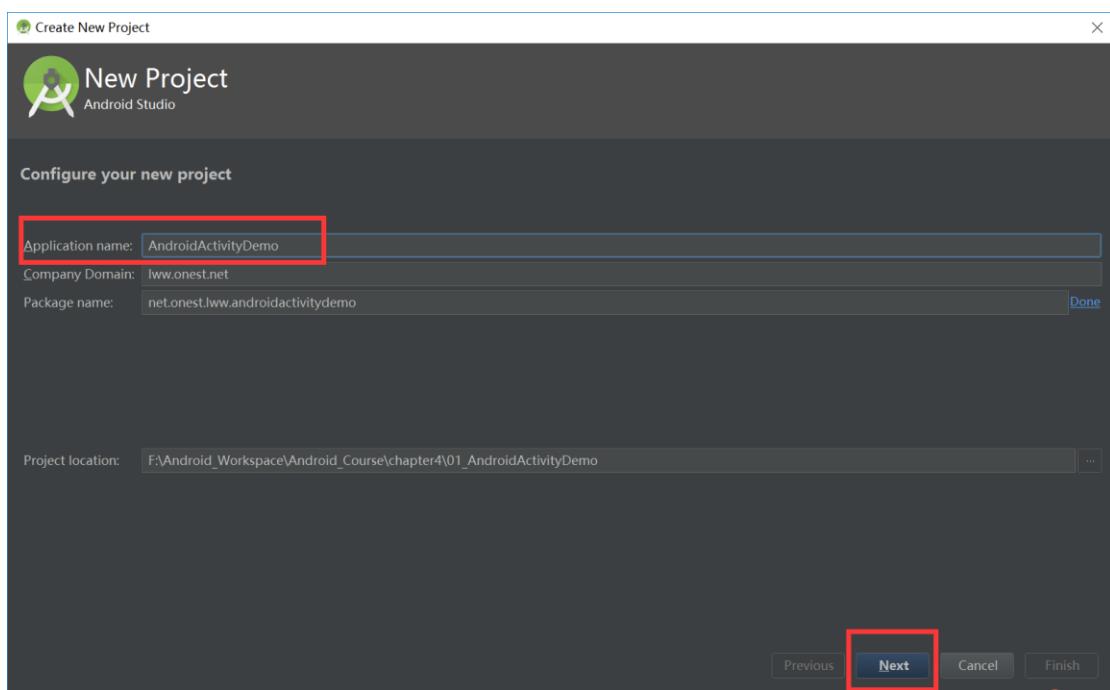


图 4.1.2

选择创建的 Activity 的类型，如图 4.1.3 所示。

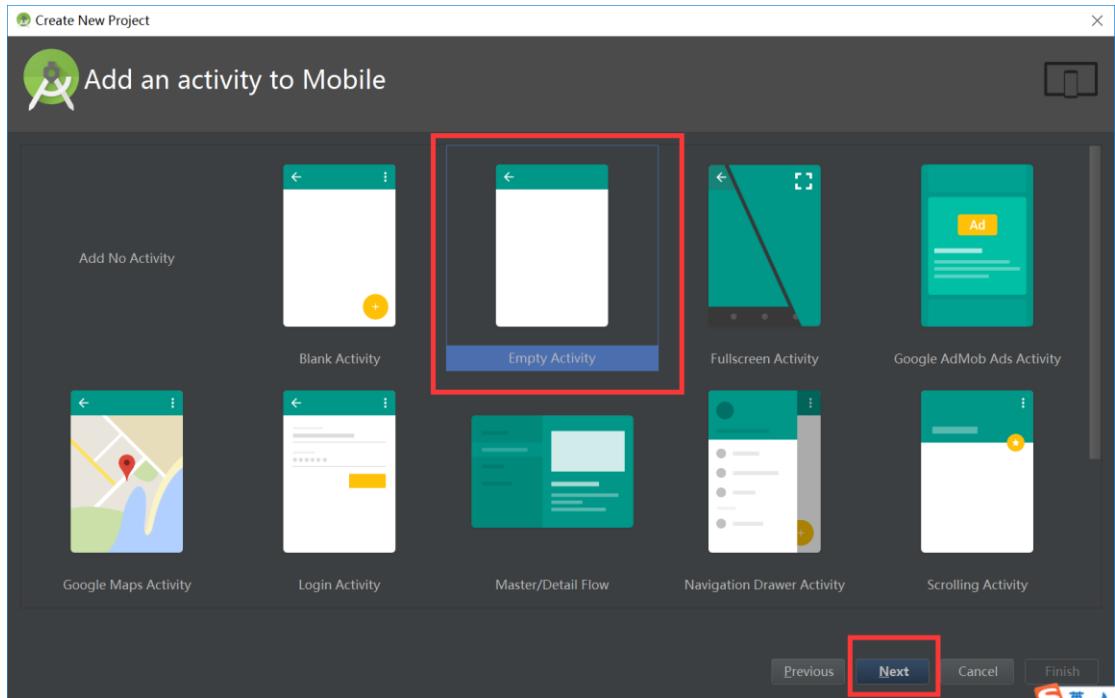


图 4.1.3

Activity 和布局文件的名称可以自定义。勾选“Generate Layout File”，将为我们自动创建一个布局文件，点击“Next”，如图 4.1.4 所示。

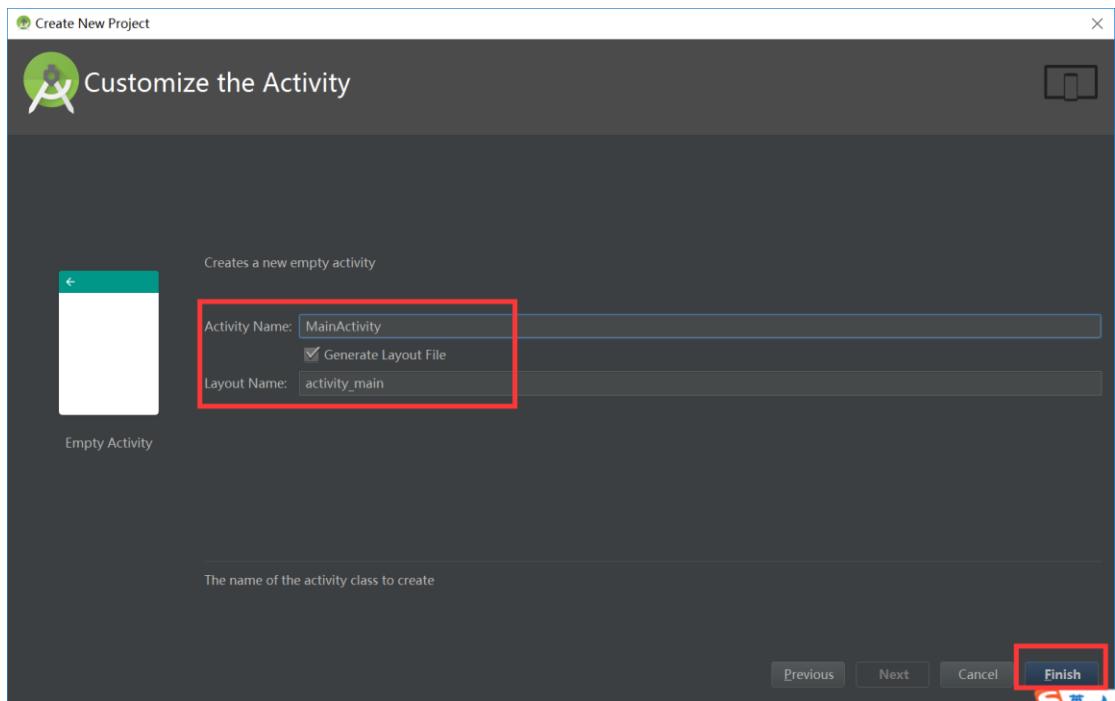


图 4.1.4

注：Activity 对应的布局文件 Layout 的名字通常是 Activity 名称各单词倒置，并间隔下划线的形式。

➤ 方法二：

方法一是通过创建 Android 项目的同时，由 AndroidStudio 自动创建 Activity。如果需要在已有的项目中再次创建 Activity 时，仍然可以通过 AndroidStudio 来创建一个 Activity。

在方法一中的项目中使用 AndroidStudio 创建一个 Activity。

首先切换至 Project 视图模式。

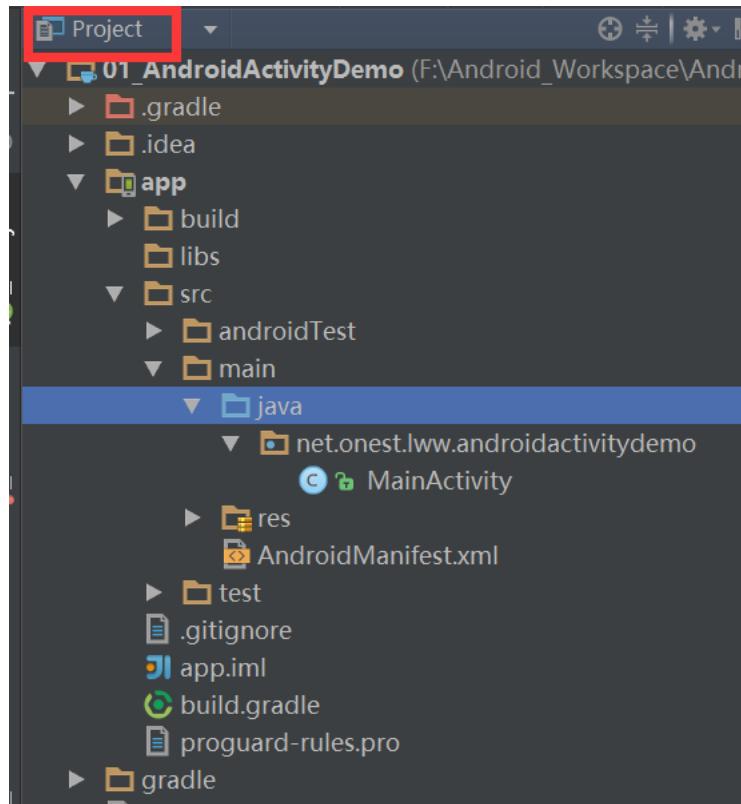


图 4.1.5

然后在项目的包上点击鼠标右键 --> New, 如图 4.1.6 所示。

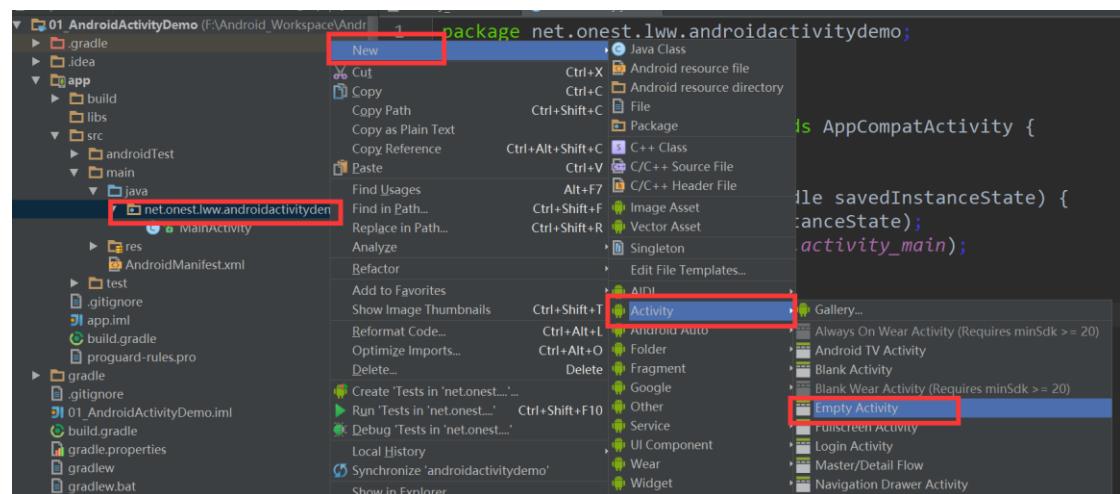


图 4.1.6

点击两次“Next”，出现如图 7 所示对话框，输入新的 Activity 的名称。

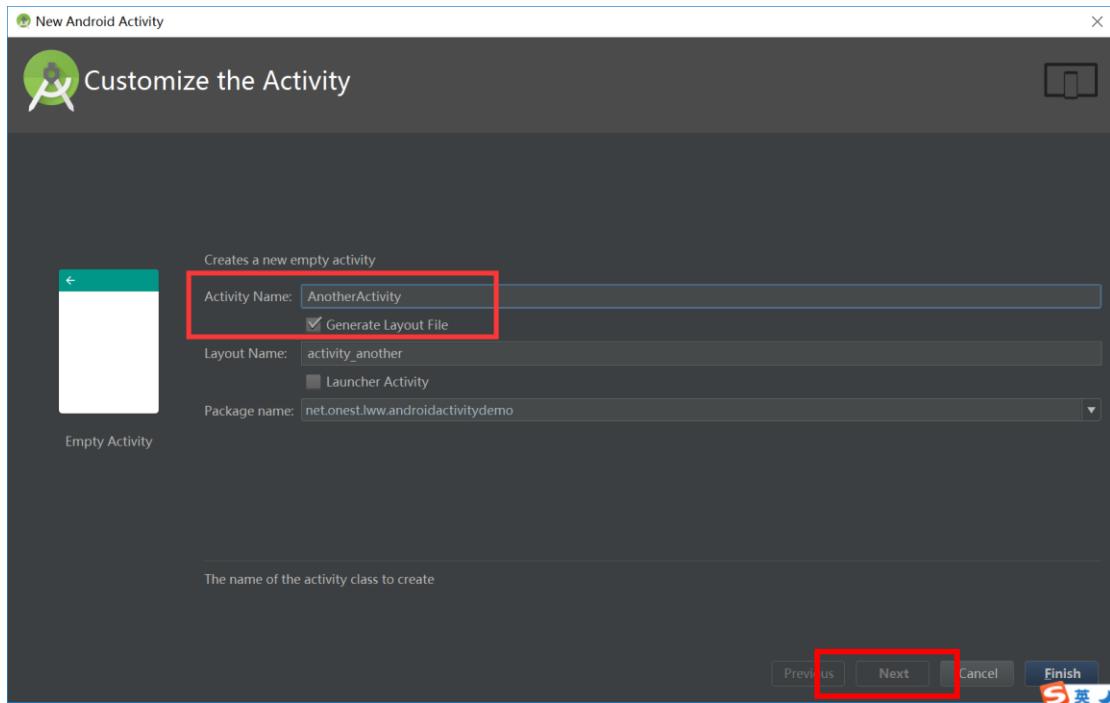


图 4.1.7

注：通过方法一和方法二所创建的 Activity 都自动创建了所对应的布局文件 Layout，并且自动注册该 Activity。对于初学者，还需要了解 Activity 与 Layout 的对应关系，以及在一个 Android 项目中，这两者的组织形式（参考方法三）。

➤ 方法三：

除方法一和方法二能够创建 Activity 以外，我们还可以使用创建一个 java 类文件的方式，来创建一个 Activity。

- 1) 在项目的包上右击鼠标 → New → class。如图 4.1.8 所示。

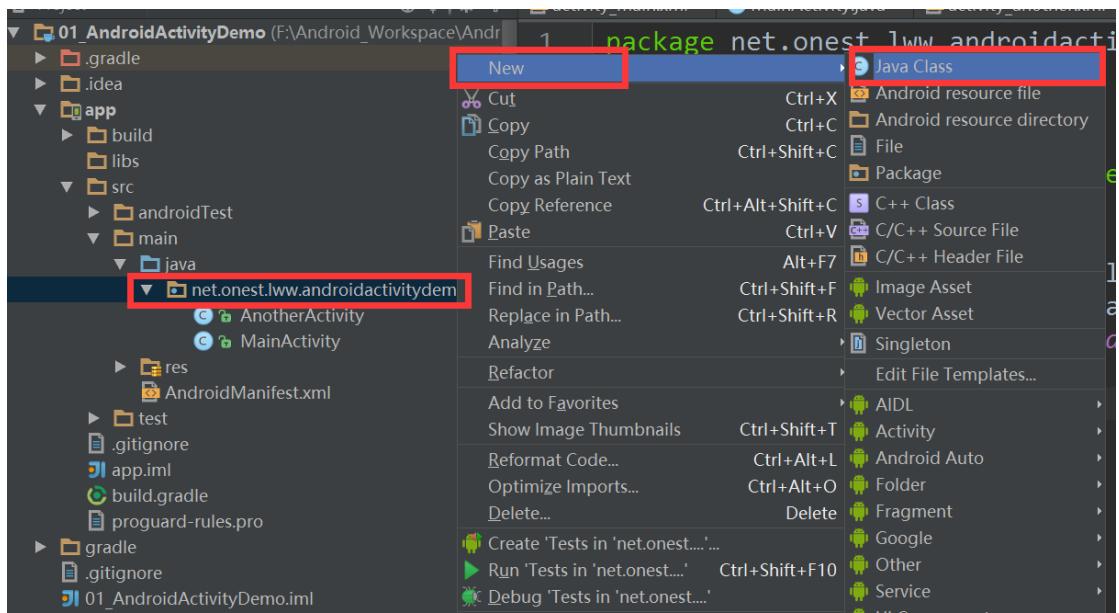


图 4.1.8

在弹出的对话框中输入 Activity 的名称，并继承 Activity 类（继承关系也可以在生成的 java 文件中手动编写代码实现）。如图 4.1.9 所示。

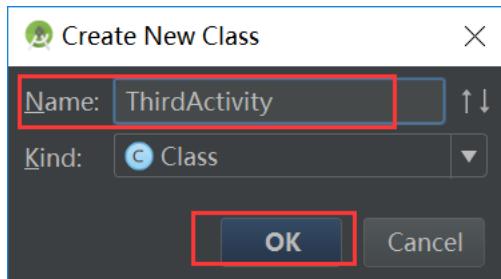


图 4.1.9

查看该 Activity 的 java 类文件，使该类继承 AppCompatActivity

```
public class ThirdActivity extends AppCompatActivity {
```

图 4.1.10

重写 onCreate 方法。在类内空白处点击鼠标右键，如图 4.1.11-4.1.13 所示。

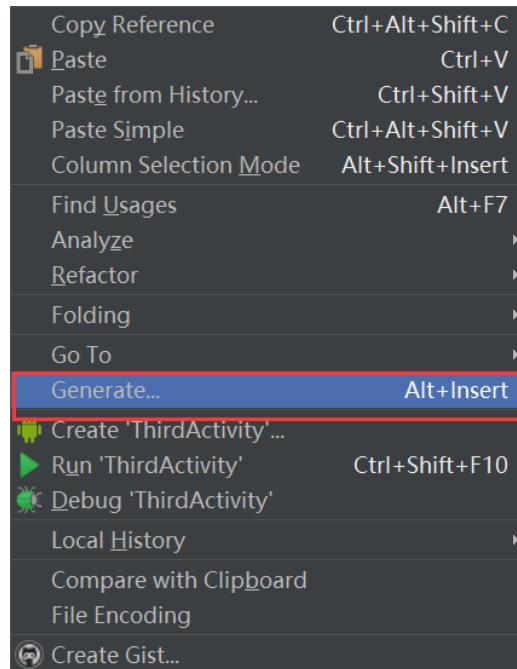


图 4.1.11

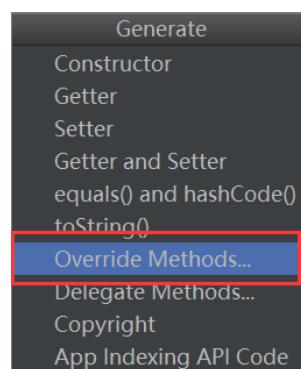


图 4.1.12

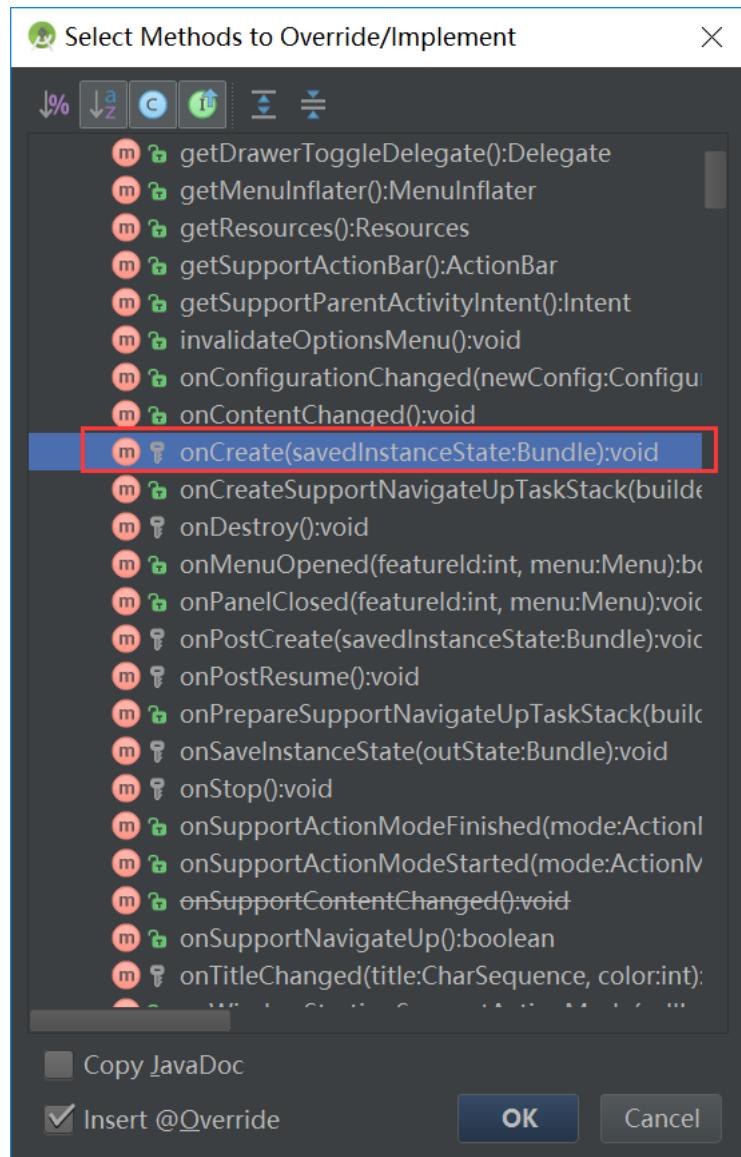


图 4.1.13

主要代码如下：

```
public class ThirdActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

2) 编写该 Activity 所对应的布局文件 activity_third.xml。

在项目上或 res 节点下的 layout 节点上右击鼠标 -> New -> XML -> Layout XML File。

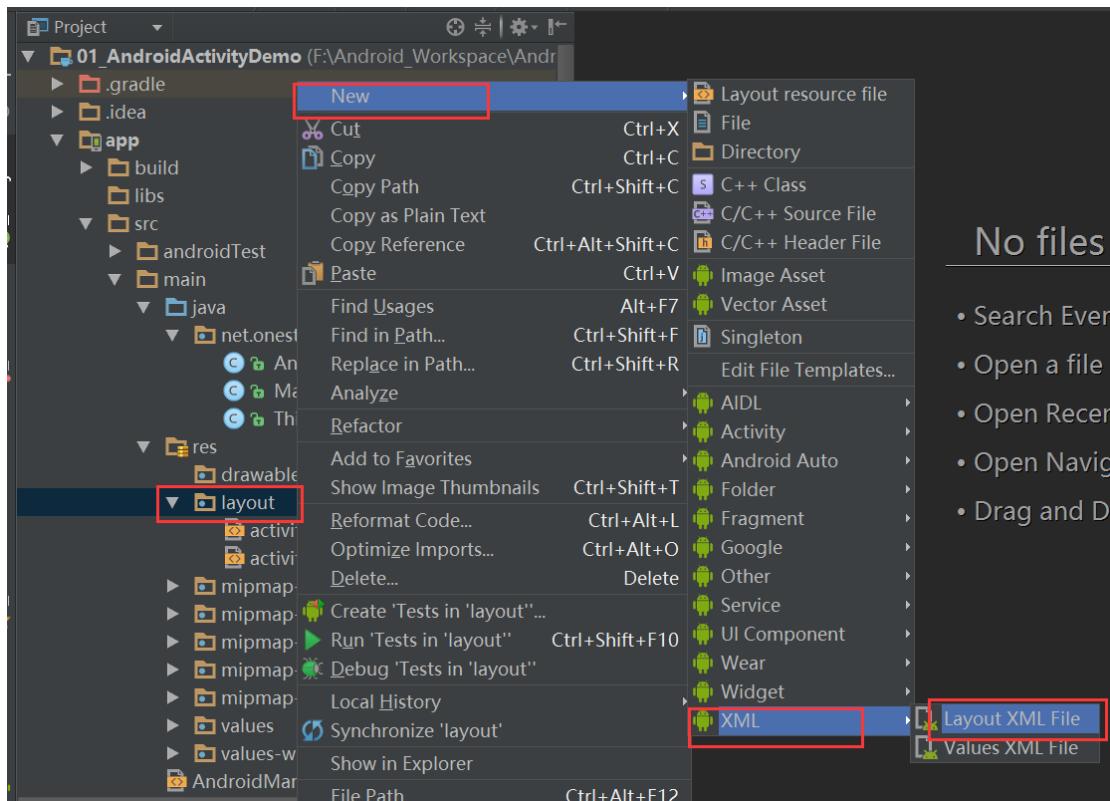


图 4.1.14

输入布局文件的名字，如图 4.1.15 所示。

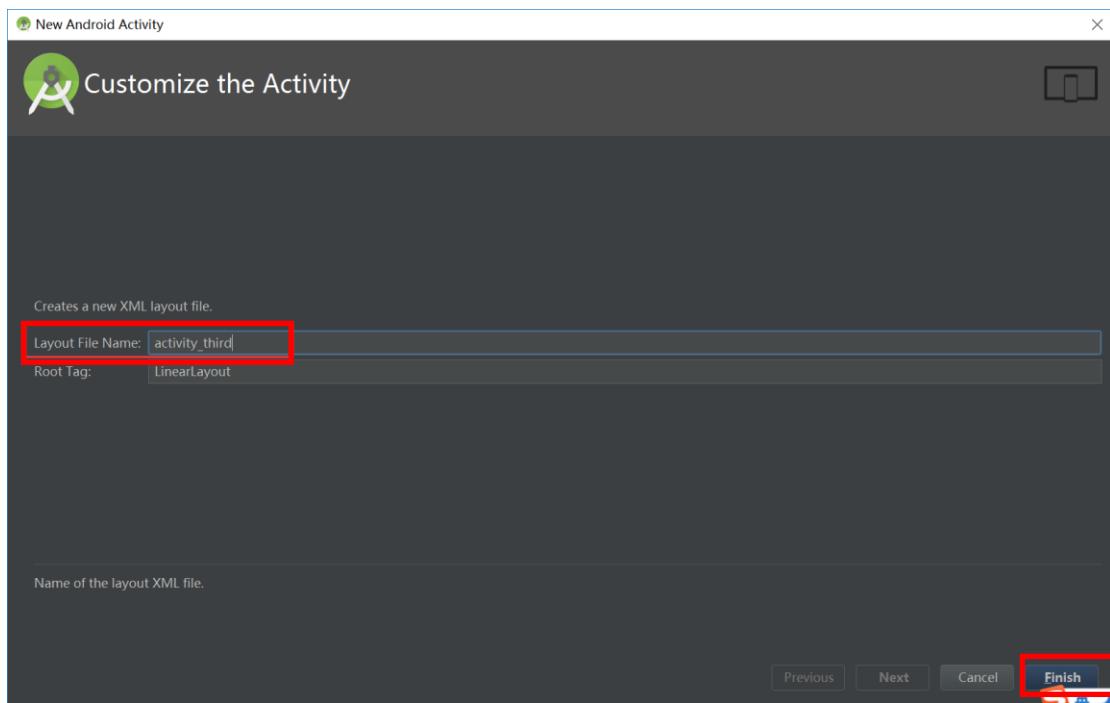


图 4.1.15

注：点击“Finish”就完成了布局文件的创建，可以在此布局文件中添加需要的组件。如图 4.1.16 所示，

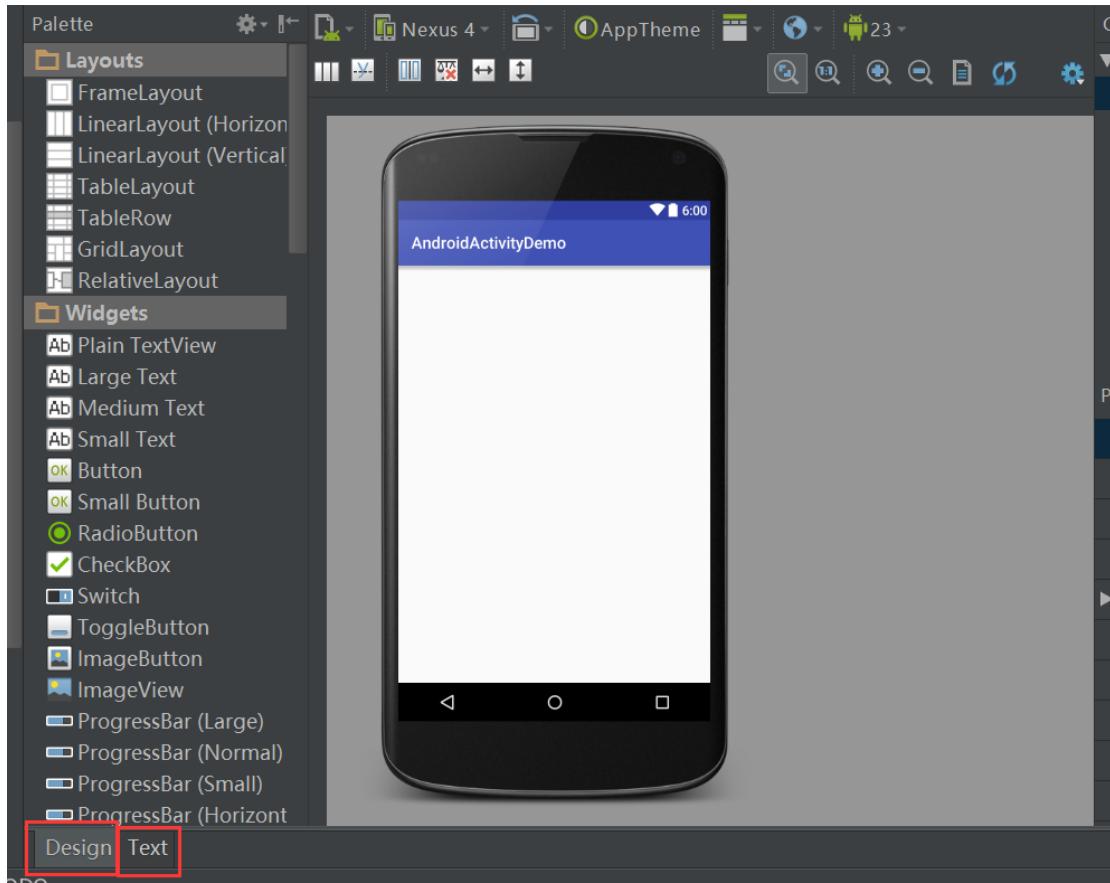


图 4.1.16

注：上图中 Graphical 表示图形视图选项卡，在该选项卡页面可以将左边的组件列表中的组件拖动到右边的布局视图中，即完成了组件的添加。对于已经添加的组件，还可以拖动更改组件的摆放位置，或设置其大小。

Activity_third.xml 选项卡表示代码视图选项卡，在该选项卡页面需要编写代码，来实现组件的添加。

3) 将刚创建的布局文件绑定给对应的 Activity, activity_third.xml 布局文件对应的 Activity 是 ThirdActivity。需要在 ThirdActivity 的 onCreate() 方法中编写代码。

```
public class ThirdActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_third);
    }
}
```

注：`setContentView()`方法是用于给当前的 Activity 设置布局文件的，其参数是一个 int 类型的值，表示布局文件的 id。

4) 在 `AndroidManifest.xml` 文件中注册 `ThirdActivity`。在 `application` 节点下添加注册 Activity 的代码。

```
<activity android:name="net.onest.lww.androidactivitydemo.ThirdActivity" />
```

注：以上 3 中创建 Activity 的方法是比较常用的几种方法，另外还有其他的方法，有兴趣的学生可以参阅其他参考资料，自行学习。

4.1.4 实验结论

4.2 实验二 Activity 生命周期

4.2.1 实验目的

1. 熟练掌握 Android 中 Activity 的生命周期。

4.2.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

4.2.3 实验步骤

学习并掌握 Activity 的生命周期，对 Android 的学习是至关重要的。Activity 在运行时会受到一些突然事件的影响，例如：你正使用一个 Activity，突然来了一个电话，这时你的应用就要具备处理这些突发事件的能力，要处理这些突发事件，就需要用到 Activity 的生命周期。

步骤一：

创建一个Android工程，重写Activity类中onCreate、onStart、onRestart、onResume、onPause、onStop、onDestroy方法，在这几个方法中加入相应的标记。

```
public class ActivityLife extends AppCompatActivity {
    private static final String TAG = "ActivityLife";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_life);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.e(TAG, "onDestroy");
    }
}
```

```
@Override
protected void onPause() {
    super.onPause();
    Log.e(TAG, "onPause");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.e(TAG, "onRestart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.e(TAG, "onResume");
}

@Override
protected void onStart() {
    super.onStart();
    Log.e(TAG, "onStart");
}

@Override
protected void onStop() {
    super.onStop();
    Log.e(TAG, "onStop");
}
}
```

注：重写某方法的操作有两种方法可以实现：

- 1) 手动编写代码
- 2) AndroidStudio生成。在代码空白处点击鼠标右键 --> Generate --> Override Methods...。如图4.2.1、图4.2.2所示。在图2红色框中勾选对应的方法，点击“确定”按钮即可。

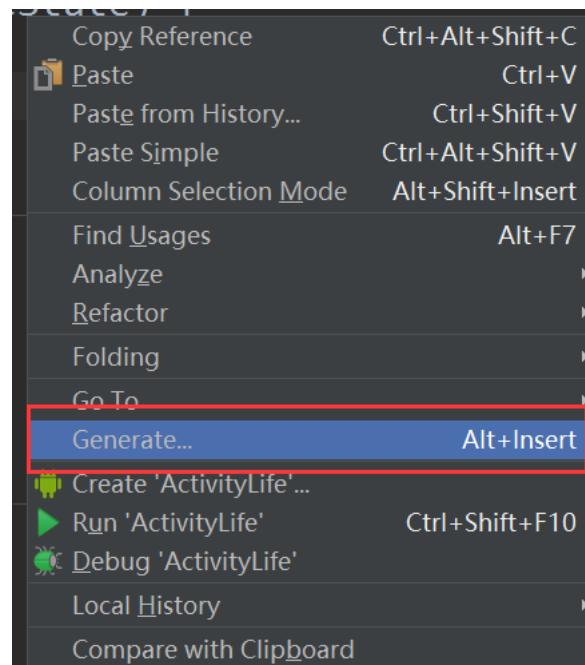


图 4.2.1

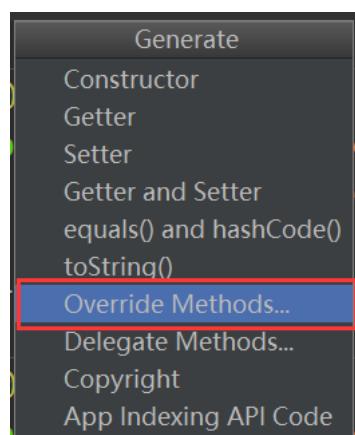


图 4.2.2

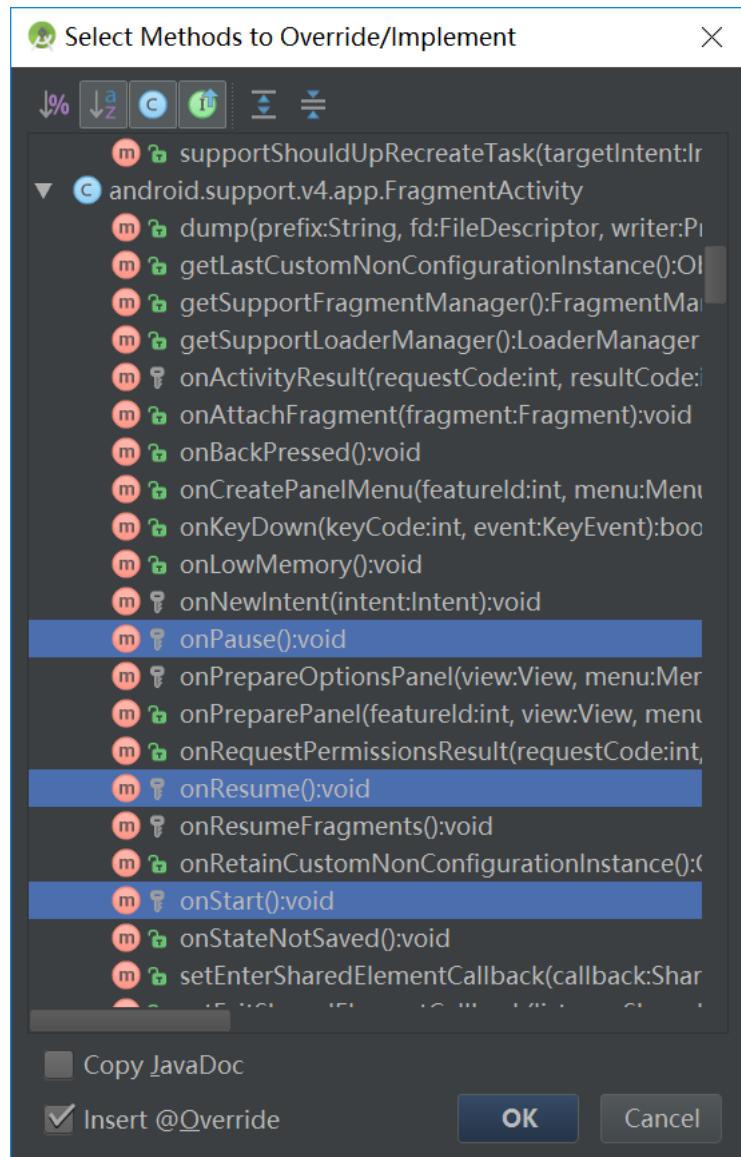


图 4.2.3

步骤二：

运行该项目，执行如下操作：观察 LogCat，查看当你做如下操作的时候 Activity 的声明周期如何？

- 1) 启动 Activity 时，生命周期如何？
- 2) 结束 Activity 时，生命周期如何？
- 3) 启动 Activity，然后点击 home 键，生命周期如何？
- 4) 启动 Activity 时，有电话呼入，然后电话结束，Activity 生命周期如何？
- 5) 启动 Activity，然后点击按钮跳转到另一个 Activity，这是生命周期如何？

根据你的理解，总结 Activity 的生命周期。

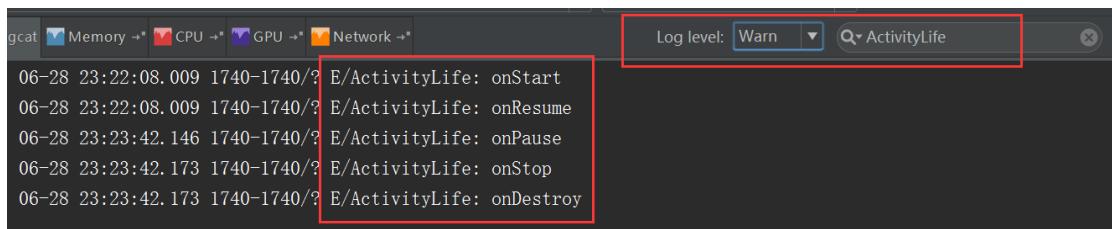


图 4.2.4

4.2.4 实验结论

4.3 实验三 Activity 之间的跳转及传递各种形式的参数（序列化）

4.3.1 实验目的

本次试验我们将一起学习 Activity 之间的跳转及传递各种形式的参数。

4.3.2 准备工作

搭建好的 Android 开发环境

4.3.3 实验步骤

大部分的 Android 应用都包含有不止一个 Activity，这时候就涉及到了 Activity 的跳转及 Activity 之间参数的传递。下面我们一起通过一个小例子来实现 Activity 间的跳转及参数的传递。新建工程 SerializeObjectDemo，主界面的布局文件中有一个按钮，其中 MainActiviy.java 的代码如下：

```
public class MainActivity extends AppCompatActivity {
```

```

private Button myButton;

@Override

public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    myButton = (Button) findViewById(R.id.myButton);

    myButton.setOnClickListener(new OnClickListener() {

        @Override

        public void onClick(View arg0) {

            BenParcelable benParcelable = new BenParcelable("BenZeph",
23, "Java/Android Engineer");

            Intent intent = new Intent();

            intent.setClass(getApplicationContext(),
                GetParcelableActivity.class);

            Bundle bundle = new Bundle();

            bundle.putParcelable("Ben", benParcelable);

            intent.putExtras(bundle);

            startActivity(intent);

        }

    });

}

}

```

其中 GetParcelableActivity.java 代码为:

```

public class GetParcelableActivity extends AppCompatActivity {

    private TextView textview;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.getParcelable);

```

```
    textview=(TextView)findViewById(R. id. text) ;  
  
    BenParcelable parcelable = getIntent().getParcelableExtra("Ben");  
  
    textview.setText("Name:" +parcelable.getName() +"\n"+ "Age:" +parcelable.getAge() +  
"\n"+ "Profession:" +parcelable.getprofession());  
}  
}
```

用于序列化的 BenParcelable 类即 BenParcelable.java 代码为：

```
public class BenParcelable implements Parcelable {  
  
    public String name;  
  
    public int age;  
  
    public String profession;  
  
    public BenParcelable(String name, int age, String profession) {  
  
        this.name = name;  
  
        this.age = age;  
  
        this.profession = profession;  
    }  
  
    public String getName() {  
  
        return name;  
    }  
  
    public void setName(String name) {  
  
        this.name = name;  
    }  
  
    public int getAge() {  
  
        return age;  
    }  
  
    public void setAge(int age) {  
  
        this.age = age;  
    }  
}
```

```

}

public String getprofession() {
    return profession;
}

public void setprofession(String profession) {
    this.profession = profession;
}

@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(Parcel parcel, int flag) {
    parcel.writeString(name);
    parcel.writeInt(age);
    parcel.writeString(profession);
}

public static final Creator<BenParcelable> CREATOR = new Creator<BenParcelable>()
{
    public BenParcelable createFromParcel(Parcel in) {
        return new BenParcelable(in);
    }

    public BenParcelable[] newArray(int size) {
        return new BenParcelable[size];
    }
};

private BenParcelable(Parcel in) {
    name = in.readString();
    age = in.readInt();
}

```

```
    profession = in.readString();  
}  
}  
}
```

需要注意的是我们需要在 `AndroidManifest.xml` 中标识 `GetParcelableActivity`:

```
<application  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme">  
    <activity  
        android:name=".MainActivity"  
        android:label="@string/title_activity_main">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent-filter>  
    </activity>  
    <activity android:name=".GetParcelableActivity" />  
    </application>
```

运行工程，效果如下：

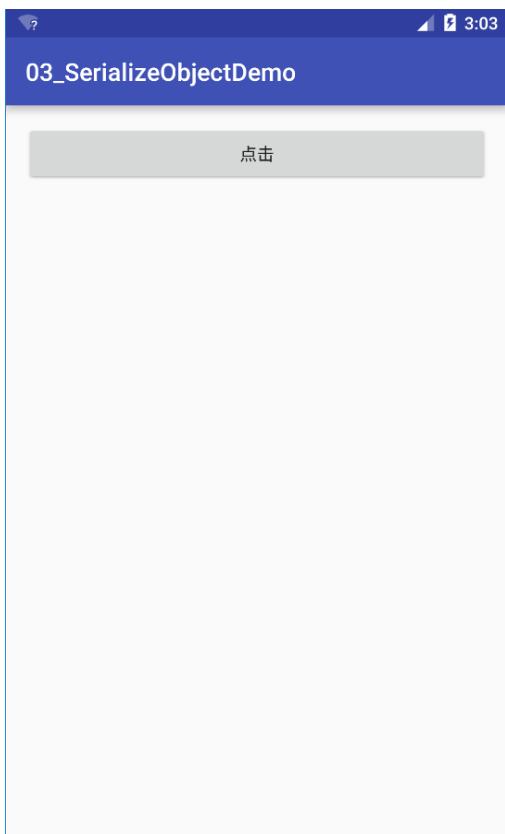


图 4.3.1

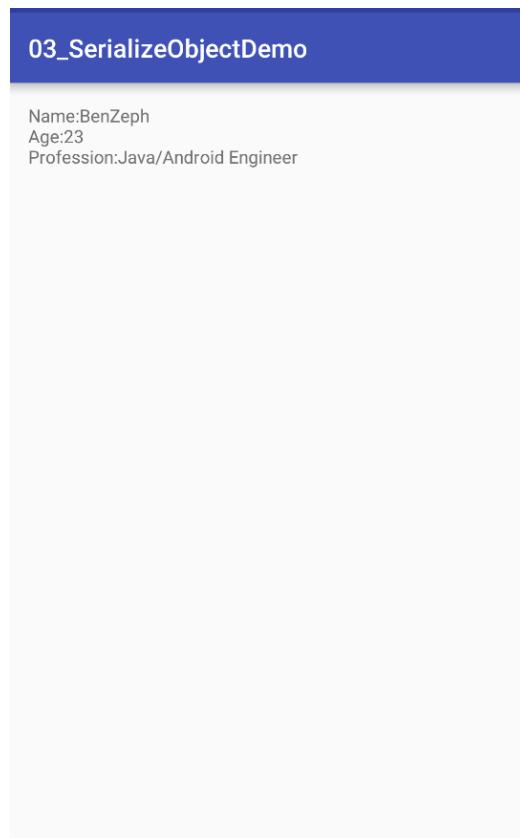


图 4.3.2

4.3.4 实验结论

通过本次试验，我们熟练掌握了 Activity 之间的跳转及传递参数的方法。

4.4 作业

4.4.1 完善第二章的作业，使用 Intent 串联各个页面

要求运行程序后首先显示的是注册界面，注册后跳转到登录界面，登录后跳转到学生列表界面。

第5章 Android 中的 Intent

本章的主要内容有对 Android 中 Intent 对象的简介, 对 Android 中的 Intent 的属性及 Intent-Filter 的介绍。

通过本章实验我们将能理解 Intent 对象的使用方式, 学会如何使用 Intent 的基本属性, 我们将学会如何使用 Intentfilter, 我们将学会如何使用 Intent 启动一些常用的系统组件。

5.1 实验一 Intent 的使用

5.1.1 实验目的

1. 掌握 Android 中 Intent 的用法。
2. 掌握 Activity 之间数据的传递方法。

5.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

5.1.3 实验步骤

➤ 任务: 从一个用户界面跳转到另一个用户界面, 掌握界面跳转基本用法。

Intent 是 Android 里面发起请求的一种常见的方式, 在 Android 中最常见的使用 Intent 就是 Activity 间的跳转了。本任务主要是学习使用 Intent 的基本属性。

步骤一 新建工程 IntentDemo，并且自动创建 Activity，名称采用默认的名称。

步骤二 编辑布局文件 activity_main.xml，采用线性布局，添加 1 个按钮，用于跳转到另一个 Activity (TwoActivity)，主要代码如下。

```
<?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: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="com.ruanjian.intentdemo.MainActivity"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="这里是 MainActivity! " />

    <Button
        android:id="@+id	btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击跳转至 TwoActivity! " />

</LinearLayout>
```

步骤三 在该 Android 工程中新建一个 Activity，名称为 TwoActivity，对应的布局文件名称为 activity_two.xml。

如果创建 Activity 的时候，创建的是类，手动继承了 Activity。采用这种方式创建的 Activity 需要在 AndroidManifest.xml 文件中注册该 Activity。

```
<activity android:name=".TwoActivity"/>
```

步骤四 编辑 activity_two.xml 布局文件，采用线性布局，添加一个按钮，用以返回到 MainActivity。

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.ruanjian.intentdemo.TwoActivity"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="这里是 TwoActivity! " />

    <Button
        android:id="@+id/btn"
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"  
    android:text="点击返回 MainActivity! " />  
  
</LinearLayout>
```

步骤五 编辑 MainActivity.java 文件，首先获取布局文件中的 Button 组件，然后在 onCreate() 方法中获取该组件对象。之后采用匿名内部类的方式，给按钮注册事件监听器，采用 Intent 方式实现 Activity 的跳转。

```
package com.ruanjian.intentdemo;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.view.View;  
import android.widget.Button;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        //获取按钮组件对象  
        Button btn = (Button) findViewById(R.id.btn);  
  
        //采用匿名内部类的方式，给按钮组件对象注册事件监听器  
        btn.setOnClickListener(new View.OnClickListener() {  
            @Override
```

```
public void onClick(View arg0) {  
    Intent i = new Intent();  
    i.setClass(MainActivity.this, TwoActivity.class);  
    startActivity(i);  
    MainActivity.this.finish();  
}  
});  
}  
}
```

步骤六 编辑 TwoActivity.java 文件，声明按钮组件对象，在 onCreate() 方法中获取布局文件中的按钮组件对象，并给该按钮注册事件监听器。

```
package com.ruanjian.intentdemo;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.view.View;  
import android.widget.Button;  
  
public class TwoActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_two);  
  
        //获取按钮组件对象  
        Button btn = (Button) findViewById(R.id.btn);  
    }  
}
```

```
//注册事件监听器  
btn.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View arg0) {  
        Intent i = new Intent();  
        i.setClass(TwoActivity.this, MainActivity.class);  
        startActivity(i);  
        TwoActivity.this.finish();  
    }  
});  
}  
}
```

步骤七 启动模拟器，运行该项目，点击相应按钮会完成 Activity 之间的跳转，效果如图所示。



图 5.1.1

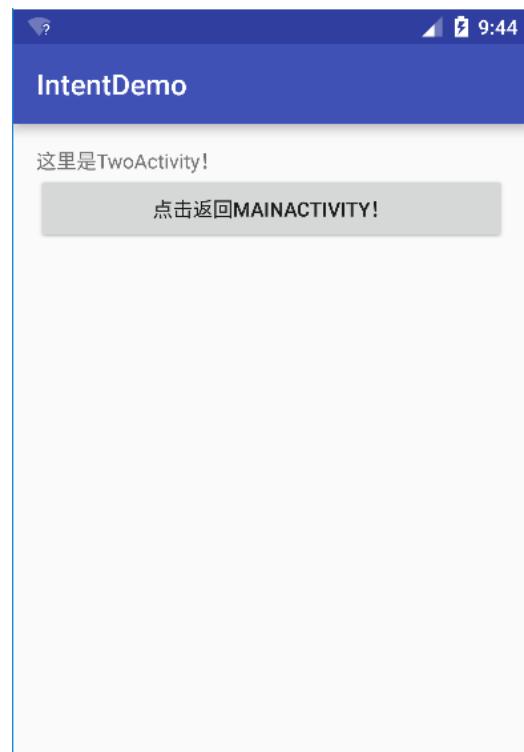


图 5.1.2

- 任务：实现在 Activity 之间跳转的同时传递各种形式的参数。

步骤一 新建工程 SerializeObjectDemo，并且自动创建 Activity，名称采用默认的名称 MainActivity.java，对应布局文件的名称为 activity_main.xml。

新建一个 Activity，名称为 GetParcelableActivity.java，对应布局文件的名称为 activity_get_parcelable.xml。

步骤二 编辑 activity_main.xml 布局文件，采用线性布局(LinearLayout)，添加一个按钮，用于点击跳转至另一个 Activity。

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.ruanjian.serializeobjectdemo.MainActivity"
    android:orientation="vertical">

    <Button
        android:id="@+id/myButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击"/>

</LinearLayout>
```

步骤三 修改 activity_get_parcelable.xml 布局文件，改用采用线性布局(LinearLayout)，并修改布局文件中的 TextView 组件的属性。

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="com.ruanjian.serializeobjectdemo.GetParcelableActivity">

    <TextView
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="" />

</LinearLayout>
```

步骤四 创建一个用于序列化的 BenParcelable 类，用于设定 Activity 之间传递数据的自定义格式，该类能够实现数据的序列化。

```
package com.ruanjian.serializeobjectdemo;

import android.os.Parcel;
import android.os.Parcelable;

public class BenParcelable implements Parcelable {
    public String name;
```

```
public int age;

public String profession;

public BenParcelable(String name, int age, String profession) {

    this.name = name;
    this.age = age;
    this.profession = profession;
}

public String getName() {

    return name;
}

public void setName(String name) {

    this.name = name;
}

public int getAge() {

    return age;
}

public void setAge(int age) {

    this.age = age;
}

public String getprofession() {

    return profession;
}
```

```
public void setprofession(String profession) {  
    this.profession = profession;  
}  
  
@Override  
public int describeContents() {  
    return 0;  
}  
  
@Override  
public void writeToParcel(Parcel parcel, int flag) {  
    parcel.writeString(name);  
    parcel.writeInt(age);  
    parcel.writeString(profession);  
}  
  
public static final Creator<BenParcelable> CREATOR = new  
Creator<BenParcelable>() {  
    public BenParcelable createFromParcel(Parcel in) {  
        return new BenParcelable(in);  
    }  
    public BenParcelable[] newArray(int size) {  
        return new BenParcelable[size];  
    }  
};  
  
private BenParcelable(Parcel in) {  
    name = in.readString();  
    age = in.readInt();  
}
```

```
    profession = in.readString();

}

}
```

步骤五 编辑主 Activity，即 MainActivity.java 文件，在 onCreate() 方法中获取对应布局 activity_main.xml 文件中的 Button 按钮对象，并给其注册事件监听器。

```
package com.ruanjian.serializeobjectdemo;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

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

        // 获取组件对象
        Button myButton = (Button) findViewById(R.id.myButton);
        myButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View arg0) {
                // 实例化序列化类
                BenParcelable benParcelable = new BenParcelable("BenZeph",

```

```
    23, "Java/Android Engineer");  
  
    // Activity 跳转，并传递数据  
    Intent intent = new Intent();  
    intent.setClass(getApplicationContext(),  
        GetParcelableActivity.class);  
    Bundle bundle = new Bundle();  
    bundle.putParcelable("Ben", benParcelable);  
    intent.putExtras(bundle);  
    startActivity(intent);  
}  
});  
}  
}
```

步骤六 编辑 GetParcelableActivity.java 文件，在 onCreate() 方法中获取对应布局 activity_get_parcelable.xml 文件中的文本框组件对象，接收传递的数据并显示在该文本框组件中。

```
package com.ruanjian.serializeobjectdemo;  
  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.widget.TextView;  
  
public class GetParcelableActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_get_parcelable);  
        // 获取组件对象
```

```
TextView textview = (TextView) findViewById(R.id.textview);

// 实例化序列化类，接收通过 Intent 跳转，传递过来的数据

BenParcelable parcelable = getIntent().getParcelableExtra("Ben");

// 将 Activity 传递过来的数据显示在 TextView 组件中

textview.setText("Name:" + parcelable.getName() + "\n"
    + "Age:" + parcelable.getAge() + "\n" + "Profession:"
    + parcelable.getprofession());

}

}
```

步骤七 启动模拟器，运行该项目，如图 3、图 4 所示。



图 5.1.3

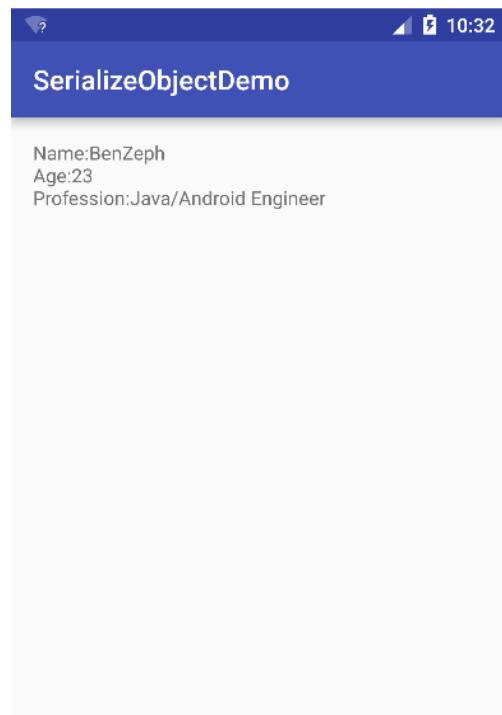


图 5.1.4

5.1.4 实验结论

通过本次实验我们学会了如何使用 Intent 的基本属性。

5.2 实验二 IntentFilter 的使用

5.2.1 实验目的

1. 掌握 IntentFilter 的用法。

5.2.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

5.2.3 实验步骤

Android 中的 Intent 通过 Action, Category 和 data 等属性进行了相应的描述，我们想做某些事情（达成某些目标），就需要填写这些参数的部分或全部，这样 Android 才会帮助我们自动的去进行某些操作。IntentFilter 是配合 Intent 而生的，你有目标行动或者结果，那么那些行动和结果就会有他完成的特定要求，这些要求就是 IntentFilter，可以理解为 Intent 和 IntentFilter 是相对应的。

步骤一 新建工程 IntentfilterDemo, 其中在 MainActivity.java 中获取布局文件中的按钮并给按钮注册事件监听器，具体代码如下：

```
package com.ruanjian.intentfilterdemo;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
```

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    Button CallBtn = (Button) findViewById(R.id.CallBtnID);  
    // 设置监听器  
    CallBtn.setOnClickListener(new Button.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            // 创建 Intent  
            Intent in = new Intent();  
            in.setAction("Hello");  
            startActivity(in);  
        }  
    });  
}  
}
```

步骤二 编辑布局文件 activity_main.xml，采用线性布局，添加 1 个按钮，用于跳转到另一个 Activity (SecondActivity)，主要代码如下

```
<?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:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="com.ruanjian.intentfilterdemo.MainActivity">

    <Button
        android:id="@+id/CallBtnID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击"/>

</LinearLayout>
```

步骤三 在项目中新建一个 Activity，命名为 SecondActivity，对应布局文件为 activity_second.xml，SecondActivity.java 具体代码如下。

```
package com.ruanjian.intentfilterdemo;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.EditText;

public class SecondActivity extends AppCompatActivity {

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

        // 获取该Activity对应的Intent的Action属性
    }
}
```

```
        String action = getIntent().getAction();

        EditText show = (EditText) findViewById(R.id.show);

        show.setText("Action 为:" + action);

    }

}
```

步骤四 activity_second.xml 文件中添加一个 EditText 用于显示当前 Activity 对应的 Intent 的 Action 属性，具体代码如下。

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.ruanjian.intentfilterdemo.SecondActivity"
    android:orientation="vertical">

    <EditText
        android:id="@+id/show"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

步骤五 在 AndroidManifest.xml 文件中添加 SecondActivity 所能响应的 Action 属性，代码如下。

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".SecondActivity">
            <intent-filter>
                <action android:name="Hello" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

步骤六 运行工程后效果如下。



图 5.2.1

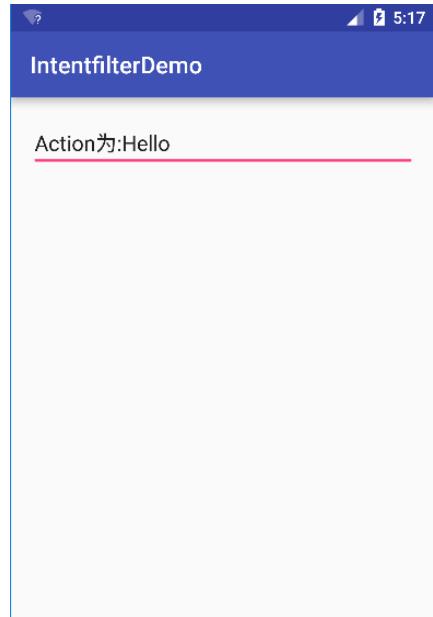


图 5.2.2

5.2.4 实验结论

通过本次试验我们学会了 Intentfilter 的用法。

5.3 实验三 Intent 启动系统组件

5.3.1 实验目的

1. 掌握 Intent 启动系统应用的方法。

5.3.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

5.3.3 实验步骤

本任务我们一起更深入的熟悉 Intent 的使用，在 Android 中常见的使用 Intent 发起的功能有电话，短信，打开网站等，下面的例子我们就做一个关于 Intent 的例子：在 Activity 中获取相应的 Button，然后在其触发的事件监听器里编写点击启动相应 intent 的逻辑代码。

步骤一 新建工程 IntentSysDemo，并且自动创建 Activity，名称采用默认的名称 MainActivity.java，对应布局文件的名称为 activity_main.xml。

编辑 activity_main.xml 布局文件，采用线性布局（LinearLayout），添加几个个按钮，分别用于点击启动系统内置的 Intent 应用。

```
<?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="match_parent"  
    android:orientation="vertical">  
  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="拨打电话" />  
  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="发送短信" />  
  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="打开浏览器" />  
/>
```

```
    android:id="@+id/web"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="web"/>

<Button
    android:id="@+id/call"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="call"/>

<Button
    android:id="@+id/sms"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="sms"/>

<Button
    android:id="@+id/image"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="image"/>

<Button
    android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="map"/>
```

```
</LinearLayout>
```

步骤二 在 MainActivity.java 中分别获得了相应的 Button 然后设置相应的点击启动 Intent 的功能，其中 MainActivity.java 代码为：

```
package com.ruanjian.intentsysdemo;

import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private Button btn_web;

    private Button btn_call;

    private Button btn_sms;

    private Button btn_image;

    private Button btn_map;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

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

```
btn_map = (Button) findViewById(R. id. map) ;  
  
btn_web = (Button) findViewById(R. id. web) ;  
  
btn_call = (Button) findViewById(R. id. call) ;  
  
btn_sms = (Button) findViewById(R. id. sms) ;  
  
btn_image = (Button) findViewById(R. id. image) ;  
  
  
  
  
btn_web.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View paramView) {  
  
        Uri uri = Uri.parse("http://www.baidu.com");  
  
        Intent intent = new Intent(Intent.ACTION_VIEW,uri);  
  
        startActivity(intent);  
  
    }  
  
});  
  
  
  
  
btn_map.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View paramView) {  

```

```
Intent intent = new Intent(Intent.ACTION_VIEW);

Uri uri = Uri.parse("geo:38.003451,114.529366");

intent.setData(uri);

startActivity(intent);

}

});

btn_call.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View paramView) {

        Intent intent = new Intent(Intent.ACTION_DIAL);

        Uri uri = Uri.parse("tel:10086");

        intent.setData(uri);

        startActivity(intent);

    }

});

btn_sms.setOnClickListener(new View.OnClickListener() {

    @Override
```

```

public void onClick(View paramView) {

    Uri uri = Uri.parse("smsto:15200006059");

    Intent intent = new Intent(Intent.ACTION_SENDTO,uri);

    intent.putExtra("sms_body", "testtest");

    startActivity(intent);

}

}) ;

btn_image.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

    Intent intent = new Intent(Intent.ACTION_VIEW);

    intent.setDataAndType(Uri.parse("file:///sdcard/Download/a.jpg"),

    "image/*");

    startActivity(intent);

}

});

}

```

步骤三 需要注意的是我们要在 AndroidManifest.xml 加上如下权限：

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.ruanjian.intentsysdemo">

    <uses-permission android:name="android.permission.CALL_PHONE"/>

    <uses-permission android:name="android.permission.SEND_SMS"/>

    <application

        android:allowBackup="true"

        android:icon="@mipmap/ic_launcher"

        android:label="@string/app_name"

        android:supportsRtl="true"

        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">

            <intent-filter>

                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />

            </intent-filter>

        </activity>

    </application>

</manifest>
```

步骤四 运行后效果如下。

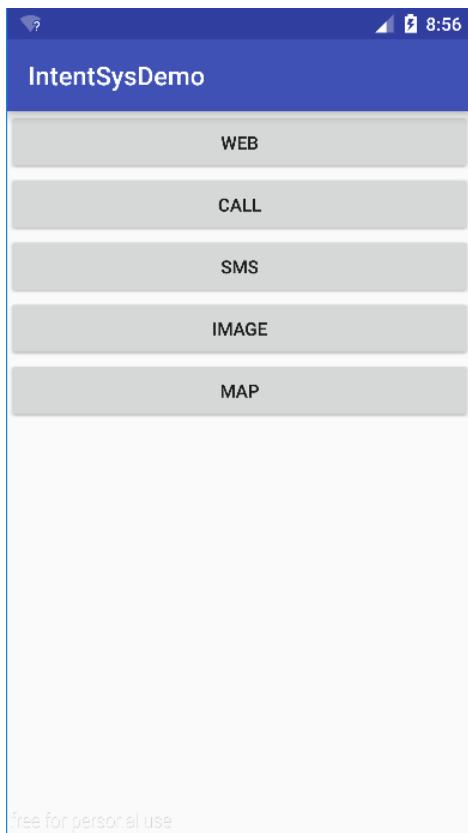


图 5.3.1



图 5.3.2

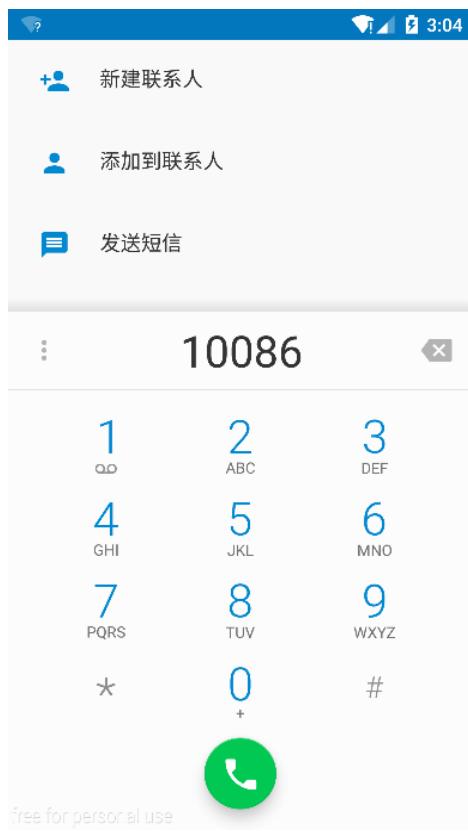


图 5.3.3

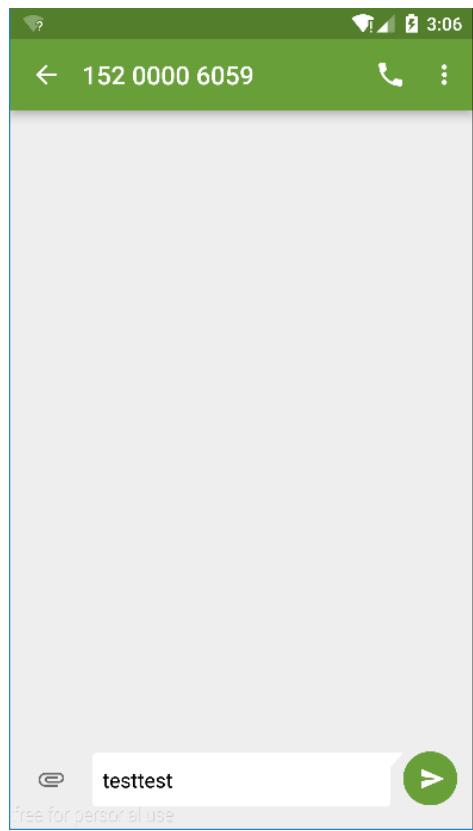


图 5.3.4

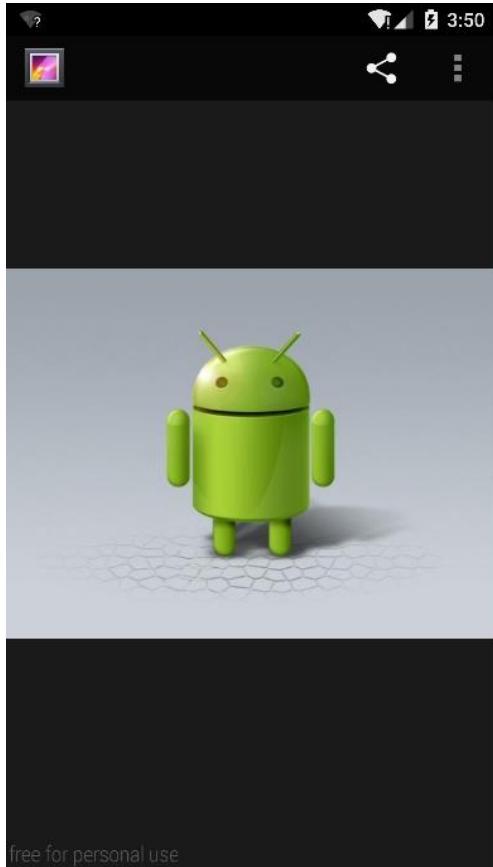


图 5.33.5



图 5.33.6

5.3.4 实验结论

通过实验我们学会了使用 Intent 启动一些常用的系统组件。

5.4 作业

5.4.1 完善学生管理系统

要求使用 Intent 加入例如打电话，发邮件等功能。

第6章 Android 中的资源处理

本章的主要内容有对资源的基本介绍，对 Android 中的字符串资源的使用，对 Android 中的字符串，演示，尺寸资源的介绍，对 Android 中的图片资源的使用的介绍，对 Android 中的 xml 资源的使用的介绍，对 Android 中布局资源的介绍，对 Android 中的菜单资源的介绍，对 Android 中的样式和主题的介绍，对 Android 中的其他资源的使用的介绍。

通过本章实验我们能掌握 Android 字符串，图片资源的使用，掌握 Android 中 xml 资源和文件资源的使用。

6.1 实验一 同一个应用程序的主题样式

6.1.1 实验目的

1. 本次试验我们将一起学习如何为一个应用程序更换主题。

6.1.2 准备工作

1. Android Studio 开发环境

6.1.3 实验步骤

一款 Android 应用如果有一个漂亮的主题，无疑能够令人耳目一新，如果主题能切换那就比较人性化了。新建工程 ThemeDemo 里面的 MainActivity.java 代码如下：

```
package com.onest.wyl;  
  
import android.app.Activity;  
  
import android.content.Context;  
  
import android.content.SharedPreferences;
```

```

import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.Button;

public class MainSkinActivity extends Activity {
    private static final String SKIN_ID = "skin_id";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        Button btnChangeSkin = (Button) findViewById(R.id.btn_change_skin);
        btnChangeSkin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if ("bg0".equals(getSkinResourceName())) {
                    setSkinResourceName("bg1");
                } else if ("bg1".equals(getSkinResourceName())) {
                    setSkinResourceName("bg2");
                } else if ("bg2".equals(getSkinResourceName())) {
                    setSkinResourceName("bg0");
                }
                refreshSkin();
            }
        });
    }
}

```

```
    @Override

    protected void onResume() {
        super.onResume();
        refreshSkin();
    }

    /**
     * 更换皮肤
     * 1. 点击更换皮肤快捷菜单后
     * 2. 程序运行起来后
     */

    private void refreshSkin() {
        int skinId = getSkinResourceId();
        findViewById(R.id.layout).setBackgroundResource(skinId);
    }

    private int getSkinResourceId() {
        int skinId = R.drawable.back_1;
        String skinName = getSkinResourceName();
        if (skinName.equals("bg1")) {
            skinId = R.drawable.back_2;
        } else if (skinName.equals("bg2")) {
            skinId = R.drawable.back_1;
        }
        return skinId;
    }

    private String getSkinResourceName() {
        try {
            SharedPreferences preferences = getSharedPreferences("skinxml",
                    Context.MODE_PRIVATE);
            return preferences.getString(SKIN_ID, "bg0");
        }
    }
}
```

```
    } catch (Exception e) {
        e.printStackTrace();
        return "bg0";
    }
}

private void setSkinResourceName(String skinName) {
    SharedPreferences preferences = getSharedPreferences("skinxml",
    Context.MODE_PRIVATE);
    Editor editor = preferences.edit();
    editor.putString(SKIN_ID, skinName);
    editor.commit();
}
}
```

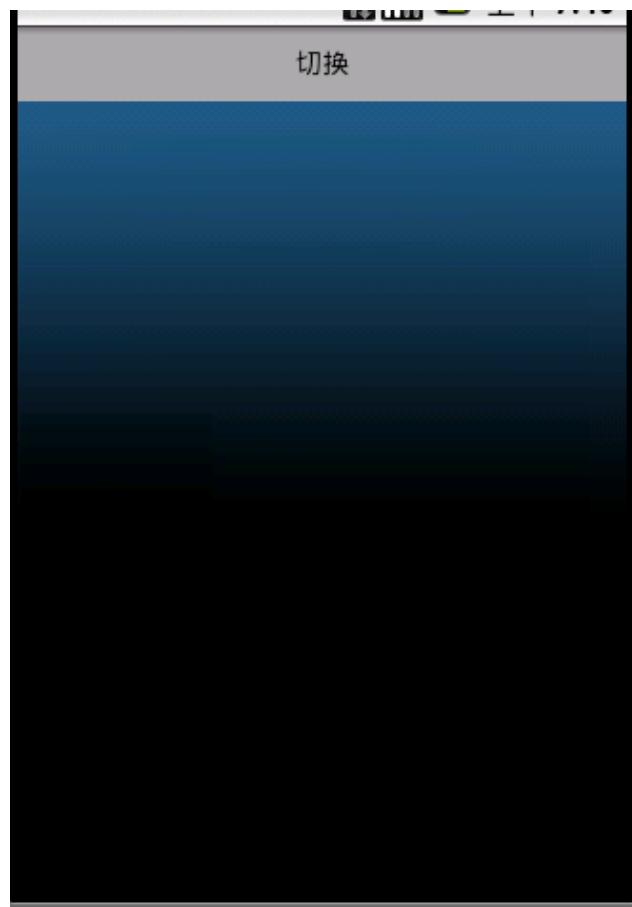


图 6.1.1

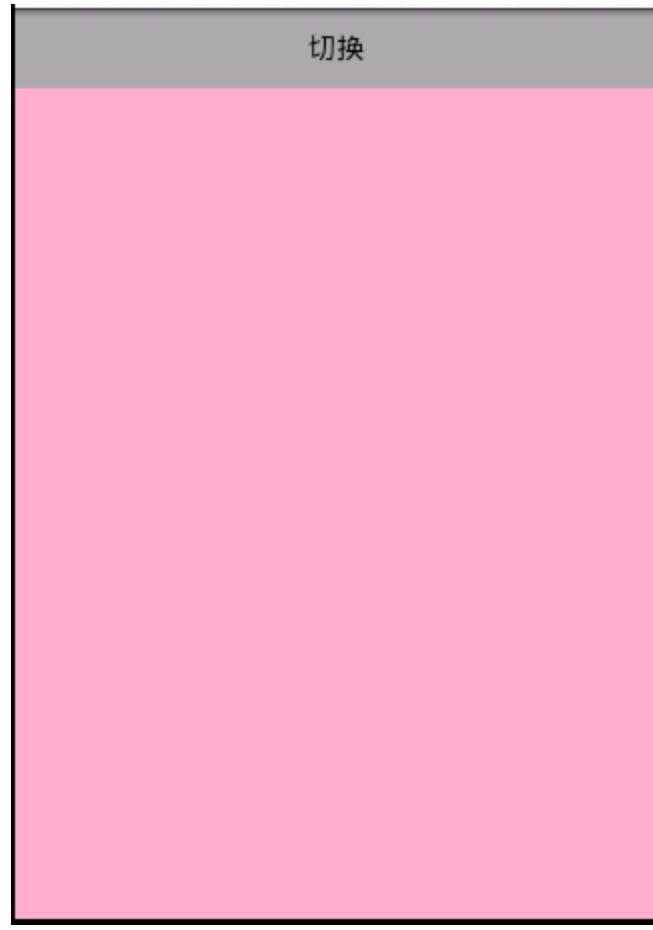


图 6.1.2

6.1.4 实验结论

通过本次试验我们学会了如何为一个应用程序更换主题。

6.2 实验二 应用程序的不同分辨率支持

6.2.1 实验目的

1. 本次试验我们将一起学习如何使同一个应用程序支持不同的分辨率。

6.2.2 准备工作

1. Android 开发环境

6.2.3 实验步骤

在 Android 开发的过程中，同一个布局对应不同的手机会显示出不同的效果。导致这个现象产生的原因是不同手机的分辨率不同，下面我们一起学习如何使同一个应用程序支持不同的分辨率，新建工程 ResolutionDemo 其中 MainActivity.java 代码为：

```
public class MainActivity extends Activity {  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.activity_main);  
  
    }  
  
}
```

其中 activity_main.xml 代码为：

```
<?xml version="1.0" encoding="utf-8"?>  
  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
  
    android:orientation="vertical"  
  
    android:layout_width="fill_parent"  
  
    android:layout_height="fill_parent">  
  
    <TextView  
  
        android:id="@+id/text"  
  
        android:layout_width="wrap_content"  
  
        android:layout_height="wrap_content"  
  
        android:text="@string/introduction"  
  
        android:drawableBottom="@drawable/qiushan"/>  
  
    <TextView  
  
        android:layout_width="fill_parent"  
  
        android:layout_height="wrap_content"  
  
        android:text="@string/size_1"/>
```

```
</LinearLayout>
```

新建文件夹 layout-320x240，新建 activity.xml 即
layout-320x240/activity.java 代码为：

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
    <TextView  
        android:id="@+id/text"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/introduction"  
        android:drawableBottom="@drawable/qiushan"/>  
    <TextView  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/size_2"/>  
</LinearLayout>
```

需要注意的是要在 AndroidManifest.xml 注册支持多分辨率：

```
<supports-screens  
    android:smallScreens="true"  
    android:normalScreens="true"  
    android:largeScreens="true"  
    android:xlargeScreens="true"  
    android:resizeable="true"/>
```

运行工程效果如下：



图 6.2.1



图 6.2.2

6.2.4 实验结论

通过本次试验我们学会了如何使同一个应用程序支持不同的分辨率。

6.3 实验三 不同语言支持

6.3.1 实验目的

1. 本次试验我们将一起学习如何使同一个应用程序支持不同的语言。

6.3.2 准备工作

1. Android 开发环境

6.3.3 实验步骤

上面实验二我们学习了同一个应用程序对不同的分辨率的支持，下面我们一起来学习同一个应用程序对不同语言的支持。新建工程 LanguageSupportDemo 其中的 MainActivity.java 代码如下：

```
import android.os.Bundle;
import android.app.Activity;
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

res\values\strings.xml 代码为：

```
<resources>
    <string name="app_name">LanguageSupportDemo</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">LanguageSupportDemo</string>
```

```
<string name="username">Please enter your name:</string>
<string name="password">Please enter your password:</string>
<string name="login">Log In</string>
<string name="cancel">Cancel</string>
</resources>
```

在 res 目录下新建文件夹 values-zh-rCN , 在此文件夹下新建 string.xml , 即 res\values-zh-rCN\strings.xml , 其代码为:

```
<resources>
    <string name="app_name">LanguageSupportDemo</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">LanguageSupportDemo</string>
    <string name="username">用户名: </string>
    <string name="password">密码: </string>
    <string name="login">登录</string>
    <string name="cancel">取消</string>
</resources>
```

运行工程，效果如下：



图 6.3.1



图 6.3.2

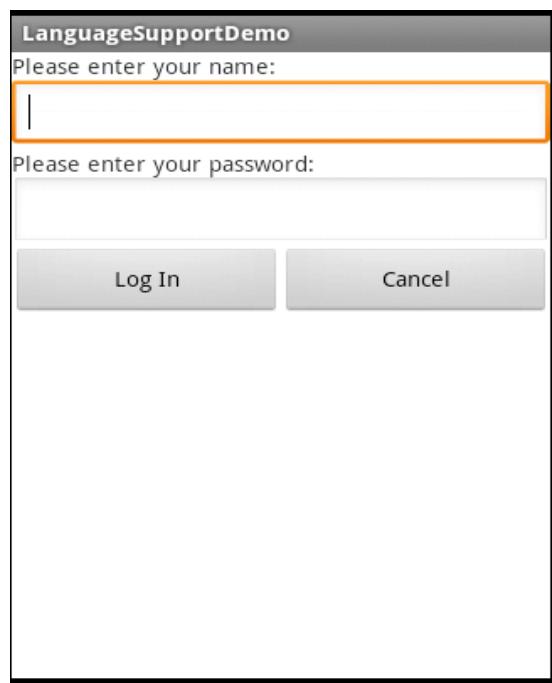


图 6.3.3

6.3.4 实验结论

通过本次试验我们学会了如何使同一个应用程序支持不同的语言。

6.4 作业

6.4.1 实现学生管理系统修改主题功能

要求完善学生管理系统可以修改主题。

6.4.2 实现学生管理系统支持不同分辨率功能

要求完善学生管理系统可以支持不同分辨率。

6.4.3 实现学生管理系统支持不同语言功能

要求完善学生管理系统可以支持不同语言。

第7章 Android 中图形图像处理

本章的主要内容为对 Android 中基本的图片使用的介绍，对 Android 中的绘图的介绍，对 Android 中的特效处理的介绍，对 Android 中四种动画的类型的介绍，对 Android 中两种动画的模式的介绍，对 Android 中使用 xml 实现各种动画的介绍，对 Android 中使用代码实现各种动画的介绍。

通过本章实验我们能够掌握 Android 中基本的绘图方式，掌握 Android 中使用 xml 实现各种动画，掌握 Android 中使用代码实现各种动画。

7.1 实验一 基本绘图

7.1.1 实验目的

1. 本次实验我们将一起学习 Android 中基本的绘图方式。

7.1.2 准备工作

1. Android 开发环境

7.1.3 实验步骤

Android 中经常需要在运动时动态生成图片，这些图片是使用 Canvas 类来实现的，Canvas 可以理解为画布，本次试验我们就一起使用 Canvas 类来实现简单的画图功能，新建工程 DrawingDemo 其中 MainActivity.java 的代码如下：

```
package com.onest.wyl;  
  
import android.app.Activity;  
  
import android.content.Context;  
  
import android.graphics.Bitmap;  
  
import android.graphics.Canvas;
```

```

import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;

public class MainActivity extends Activity {

    private Paint mPaint;

    /** Called when the activity is first created. */
    @Override

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(new MyView(this));

        mPaint = new Paint();
        mPaint.setStrokeWidth(4);
        mPaint.setStyle(Paint.Style.STROKE);
    }

    public class MyView extends View {

        private Bitmap mBitmap;
        private Canvas mCanvas;
        private Path mPath;
        private Paint mBitmapPaint;

        public MyView (Context context){
            super(context);

            mBitmap = Bitmap.createBitmap(320, 480, Bitmap.Config.ARGB_8888);
            mCanvas = new Canvas(mBitmap);
            mPath = new Path();
        }

        public void onDraw(Canvas canvas)
    }
}

```

```

{
    canvas.drawColor(Color.WHITE);

    canvas.drawBitmap(mBitmap, 0, 0, mBitmapPaint);

    canvas.drawPath(mPath, mPaint);
}

private float mX, mY;

private static final float TOUCH_TOLERANCE = 4;

private void touch_start(float x, float y) {
    mPath.reset();

    mPath.moveTo(x, y);

    mX = x;

    mY = y;
}

private void touch_move(float x, float y) {
    float dx = Math.abs(x - mX);

    float dy = Math.abs(y - mY);

    if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {
        mPath.quadTo(mX, mY, (x + mX)/2, (y + mY)/2);

        mX = x;

        mY = y;
    }
}

private void touch_up() {
    mPath.lineTo(mX, mY);

    mCanvas.drawPath(mPath, mPaint);

    mPath.reset();
}

@Override

public boolean onTouchEvent(MotionEvent event) {

```

```
float x = event.getX();
float y = event.getY();

switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        touch_start(x, y);
        invalidate();
        break;
    case MotionEvent.ACTION_MOVE:
        touch_move(x, y);
        invalidate();
        break;
    case MotionEvent.ACTION_UP:
        touch_up();
        invalidate();
        break;
}
return true;
}
```

运行工程效果如图:

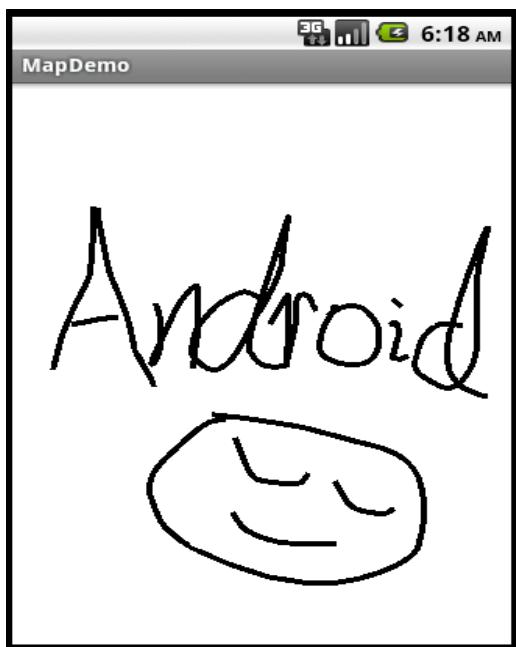


图 7.1.1

7.1.4 实验结论

通过本次实验我们掌握了 Android 中基本的绘图方式。

7.2 实验二 在图层上加上一层 addlayout

7.2.1 实验目的

1. 本次实验我们将一起学习在图层上加上一层 addlayout。

7.2.2 准备工作

1. Android 开发环境

7.2.3 实验步骤

有时候在空白的画布上画图，未免太过单调，如果有一个好的背景就好的，下面我们就一起学习如何在背景图片上画图，即在图层上加上一层 addlayout。

新建工程 AddlayoutDemo，其中 MainActivity.java 代码如下：

```
public class MainActivity extends Activity {  
    private Paint mPaint;  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(new MyView(this));  
        mPaint = new Paint();  
        mPaint.setColor(Color.GREEN);  
        //设置画笔风格为空心  
        mPaint.setStyle(Paint.Style.STROKE);  
    }  
  
    public class MyView extends View {  
        private Bitmap mBitmap;  
        private Canvas mCanvas;  
        private Path mPath;  
        private Paint mBitmapPaint;  
        public MyView(Context context){  
            super(context);  
            mBitmap = Bitmap.createBitmap(320, 480, Bitmap.Config.ARGB_8888);  
            mCanvas = new Canvas(mBitmap);  
            mPath = new Path();  
        }  
        public void onDraw(Canvas canvas)  
        {  
            // 获取资源文件的引用res  
            Resources res = getResources();  
            // 获取图形资源文件
```

```
Bitmap bmp = BitmapFactory.decodeResource(res, R.drawable.back);

// 设置canvas画布背景为白色
canvas.drawColor(Color.WHITE);

canvas.drawBitmap(bmp, 0, 0, null);

canvas.drawBitmap(mBitmap, 0, 0, mBitmapPaint);

canvas.drawPath(mPath, mPaint);

}

private float mX, mY;

private static final float TOUCH_TOLERANCE = 4;

private void touch_start(float x, float y) {

    mPath.reset();

    mPath.moveTo(x, y);

    mX = x;

    mY = y;

}

private void touch_move(float x, float y) {

    float dx = Math.abs(x - mX);

    float dy = Math.abs(y - mY);

    if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {

        mPath.quadTo(mX, mY, (x + mX)/2, (y + mY)/2);

        mX = x;

        mY = y;

    }

}

private void touch_up() {

    mPath.lineTo(mX, mY);

    mCanvas.drawPath(mPath, mPaint);

    mPath.reset();

}
```

```
}

@Override

public boolean onTouchEvent(MotionEvent event) {

    float x = event.getX();

    float y = event.getY();

    switch (event.getAction()) {

        case MotionEvent.ACTION_DOWN:

            touch_start(x, y);

            invalidate();

            break;

        case MotionEvent.ACTION_MOVE:

            touch_move(x, y);

            invalidate();

            break;

        case MotionEvent.ACTION_UP:

            touch_up();

            invalidate();

            break;

    }

    return true;
}
}
```

运行工程效果如下：

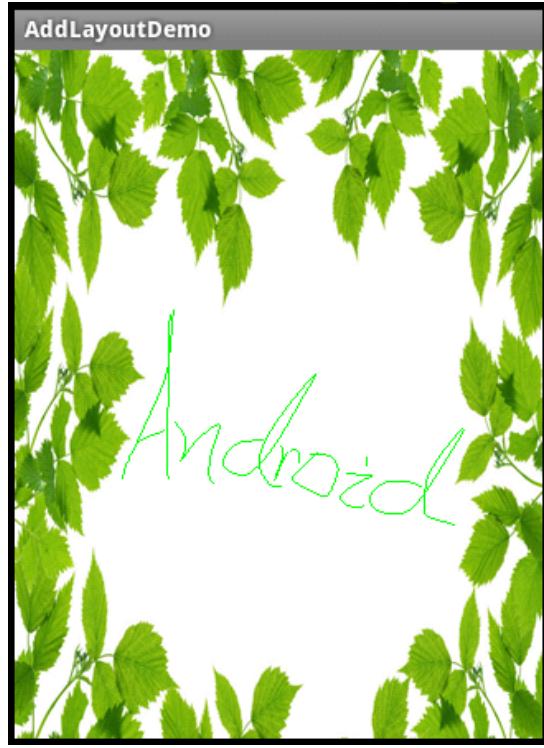


图 7.2.1

7.2.4 实验结论

通过本次实验我们学会了在图层上加上一层 addlayout。

7.3 实验三 通过动画实现启动页播放动画自动跳转

7.3.1 实验目的

1. 本次实验我们将一起学习如何通过动画实现启动页播放动画自动跳转。

7.3.2 准备工作

1. Android 开发环境

7.3.3 实验步骤

动画是我们在工程中常用的一种效果，它能给人很好的视觉体验，本次试验

我们在手机 QQ 首次启动的欢迎界面跳转的基础上加上动画效果，新建工程 SkipAnimationDemo 其中的 MainActivity.java 代码如下：

```
package com.onest.wyl;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.animation.Animation;

import android.view.animation.Animation.AnimationListener;

import android.view.animation.AnimationUtils;

import android.widget.ImageView;

public class MainActivity extends Activity implements AnimationListener {

    private ImageView imageView = null;

    private Animation alphaAnimation = null;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        imageView = (ImageView) findViewById(R.id.img);

        alphaAnimation = AnimationUtils.loadAnimation(this, R.anim.welcome);

        alphaAnimation.setFillEnabled(true); //启动Fill保持

        alphaAnimation.setFillAfter(true); //设置动画的最后一帧是保持在View上面

        imageView.setAnimation(alphaAnimation);

        alphaAnimation.setAnimationListener(this); //为动画设置监听

    }

    @Override

    public void onAnimationStart(Animation animation) {

    }

    @Override

    public void onAnimationEnd(Animation animation) {
```

```
//动画结束时结束欢迎界面并转到软件的主界面

Intent intent = new Intent(this, NextActivity.class);
startActivity(intent);
this.finish();
}

@Override

public void onAnimationRepeat(Animation animation) {
}

}
```

动画效果的实现需要在 res 目录下新建文件夹 anim, anim 文件夹下里的 welcomo.xml 代码如下：

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

<set xmlns:android="http://schemas.android.com/apk/res/android"
      android:interpolator="@android:anim/accelerate_interpolator">

    <alpha
          android:fromAlpha="0.0"
          android:toAlpha="1.0"
          android:duration="2000" />

    <alpha
          android:fromAlpha="1.0"
          android:toAlpha="0.0"
          android:startOffset="3000"
          android:duration="3000" />
</set>
```

运行工程效果如图：



图 7.3.1



图 7.3.2



图 7.3.3

7.3.4 实验结论

通过本次实验我们掌握了如何通过动画实现启动页播放动画自动跳转。

7.4 作业

7.4.1 在图上做标记

如图：



图 7.4.1



图 7.4.2

要求将上图中显示地图的图片设为背景，将用于标记的图片，缩放合适的倍数，

标记在地图中“河北师范大学新校区”附近，最终要实现的效果如图：



图 7.4.3

7.4.2 实现学生管理系统启动页面功能

要求完善学生管理系统在开始启动程序的时候加入动画启动页面。

第8章 Android 的数据操作和特殊 IO

本章的主要内容有对 Android 中的 SharedPreferences 和 Editor 的使用的介绍，对 Android 中的 Files 的使用的介绍，对 Android 中的 SQLite 的使用的介绍，对 Android 中的特殊输入的介绍。

本章实验主要内容为 SharedPreferences 的使用，SQLite 的使用，特殊输入的使用。

8.1 实验一 Android 中 Sqlite 使用

8.1.1 实验目的

1. 掌握小型嵌入式数据库 Sqlite 的使用。

8.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

8.1.3 实验步骤

➤ 任务：练习使用 Sqlite 嵌入式数据库

步骤一：

创建“Android Application Project”，项目名称为 SqliteDemo。并勾选自动创建 Activity。

步骤二：

编辑 SharedPreferencesActivity 对应的布局文件，要求如图 8.1.1 所示效果。

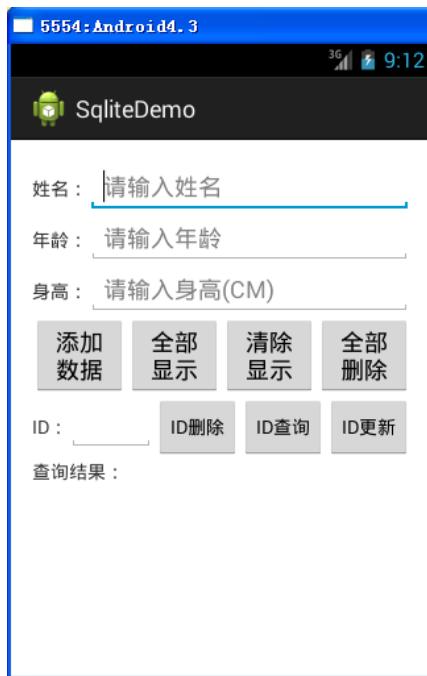


图 8.1.1

其中“显示结果”下两行有 2 个 TextView 组件，用于显示操作的结果状态。
关键代码略。各组件属性如下表所示。

表 8.1.1 组件列表

组件类型	组件 ID	显示文字	组件类型	组件 ID	显示文字
TextView		姓名:	EditText	edt_name	
TextView		年龄:	EditText	edt_age	
TextView		身高:	EditText	edt_height	
TextView		ID:	EditText	edt_id	
TextView		查询结果:	Button	btn_delete	ID 删除
TextView	tv_status		Button	btn_search	ID 查询
TextView	tv_result		Button	btn_update	ID 更新
Button	btn_add	添加数据	Button	btn_clear	清除显示
Button	btn_showAll	全部显示	Button	btn_deleteAll	全部删除

步骤三：

编写 Person 类，该类描述了要保存的数据的类型。该类封装了某人的姓名、年龄、身高属性，重写 `toString()` 方法。关键代码如下。

```

public class People {
    public int id = -1;
    public String name;
    public int age;
    public float height;
    @Override
    public String toString() {
        String result = "";
        result += "ID: " + this.id + ", 姓名: " + this.name + ", 年龄: "
            + this.age + ", 身高: " + this.height + ", ";
        return result;
    }
}

```

注：该类中所封装的属性，就是要存在数据库中的每条记录的类型。

步骤四：

编写数据库操作的工具类 DBAdapter。

- 1) 定义数据库操作工具类的属性并编写构造方法。

```

//定义各常量
private static final String DB_NAME = "people.db";
private static final String TABLE_NAME = "peopleinfo";
private static final int DB_VERSION = 1;
private static final String ID = "_id";
private static final String NAME = "_name";
private static final String AGE = "_age";
private static final String HEIGHT = "_height";

//声明数据库对象
private SQLiteDatabase db;
//声明上下文Context对象
private Context context;

//声明DBOpenHelper类的对象
private DBOpenHelper dbOpenHelper;

public DBAdapter(Context context){
    this.context = context;
}

```

注：DBOpenHelper 类是一个继承自 SQLiteOpenHelper 类的静态内部类，主要负责建立、更新和打开数据库的操作。

2) 在该类中编写静态内部类 DBOpenHelper，继承 SQLiteOpenHelper 类，需要实现 onCreate() 和 onUpgrade() 方法，该类主要用于建立、更新和打开数据库。主要代码如下。

```
private static class DBOpenHelper extends SQLiteOpenHelper{
    //定义创建数据表的字符串常量
    private static final String TABLE_CREATE = "create table IF NOT EXISTS "
        + TABLE_NAME + "(" + ID
        + " integer primary key autoincrement," + NAME + " text not null,"
        + AGE + " integer," + HEIGHT + " float);";
    public DBOpenHelper(Context context, String name, CursorFactory factory
        , int version) {
        super(context, name, factory, version);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(TABLE_CREATE);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("drop table if exists" + TABLE_NAME);
        onCreate(db);
    }
}
```

3) 在数据库操作工具类 DBAdapter 类中编写数据库的打开和关闭操作。

```
/*
 * @desc close the database.
 */
public void close(){
    if(db != null){
        db.close();
        db = null;
    }
}

/*
 * @desc open the database
 */
public void openDB() throws SQLiteException{
    dbOpenHelper = new DBOpenHelper(context, DB_NAME, null, DB_VERSION);
    try {
        db = dbOpenHelper.getWritableDatabase();
    } catch (SQLiteException ex) {
        db = dbOpenHelper.getReadableDatabase();
    }
}
```

- 4) 在 DBAdapter 类中编写增、删、改操作。

```

/**
 * @desc 向数据库表中插入数据
 * @param people 要插入的数据对象
 * @return 返回插入的新数据的行ID号, 若插入失败则返回-1
 */
public long insert(People people){
    ContentValues newValue = new ContentValues();
    newValue.put(NAME, people.name);
    newValue.put(AGE, people.age);
    newValue.put(HEIGHT, people.height);
    return db.insert(TABLE_NAME, null, newValue);
}

/**
 * @desc 删除所有记录
 * @return 删除成功则返回影响的行数, 否则返回0
 */
public long deleteAllData(){
    return db.delete(TABLE_NAME, null, null);
}

/**
 * @desc 删除某条记录
 * @param id 将被删除的记录id
 * @return 返回删除的行数,否则返回0
 */
public long deleteData(long id){
    return db.delete(TABLE_NAME, ID + "=" + id , null);
}

/**
 * @desc 更新数据库表
 * @param id 待更新的记录id
 * @param people 更新的数据
 * @return 返回更新的行数
 */
public long updateData(long id, People people){
    ContentValues values = new ContentValues();
    values.put(NAME, people.name);
    values.put(AGE, people.age);
    values.put(HEIGHT, people.height);
    return db.update(TABLE_NAME, values, ID + "=" + id, null);
}

```

5) 在 DBAdapter 类中编写查询数据的方法

```

/**
 * @desc 查询所有记录
 * @return 返回所有记录对象数组
 */
public People[] queryAllData(){
    Cursor results = db.query(TABLE_NAME, new String[]{ID, NAME, AGE
        , HEIGHT}, null, null, null, null);
    return convertToPeople(results);
}

public People[] queryData(long id) {
    Cursor results = db.query(TABLE_NAME, new String[] { ID, NAME, AGE
        , HEIGHT},
        ID + "=" + id, null, null, null);
    return convertToPeople(results);
}

/**
 * @desc 读取记录集中的数据，并保存到对象数组中。
 * @param cursor 待读取的记录集对象
 * @return 保存着记录集内容的对象数组
 */
private People[] convertToPeople(Cursor cursor){
    int resultCounts = cursor.getCount();
    if(resultCounts == 0 || !cursor.moveToFirst()){
        return null;
    }
    People[] peoples = new People[resultCounts];
    for(int i = 0; i < resultCounts; i++){
        peoples[i] = new People();
        peoples[i].id = cursor.getInt(0);
        peoples[i].name = cursor.getString(cursor.getColumnIndex(NAME));
        peoples[i].age = cursor.getInt(cursor.getColumnIndex(AGE));
        peoples[i].height = cursor.getFloat(cursor.getColumnIndex(HEIGHT));
        cursor.moveToNext();
    }
    return peoples;
}

```

步骤五：

完善 MainActivity。

- 1) 声明各组件对象

```
//声明组件  
private EditText edtName = null;  
private EditText edtAge = null;  
private EditText edtHeight = null;  
private Button btnAdd = null;  
private Button btnShowAll = null;  
private Button btnClear = null;  
private Button btnDeleteAll = null;  
  
private EditText edtId = null;  
private Button btnDelete = null;  
private Button btnSearch = null;  
private Button btnUpdate = null;  
  
private TextView tvStatus = null;  
private TextView tvResult = null;  
  
//声明数据库工具类  
private DBAdapter dbAdapter = null;
```

2) 在 onCreate() 方法中，获取布局文件中的组件对象

```
//获取组件对象  
edtName = (EditText) findViewById(R.id.edt_name);  
edtAge = (EditText) findViewById(R.id.edt_age);  
edtHeight = (EditText) findViewById(R.id.edt_height);  
  
btnAdd = (Button) findViewById(R.id.btn_add);  
btnShowAll = (Button) findViewById(R.id.btn_showAll);  
btnClear = (Button) findViewById(R.id.btn_clear);  
btnDeleteAll = (Button) findViewById(R.id.btn_deleteAll);  
  
edtId = (EditText) findViewById(R.id.edt_id);  
  
btnDelete = (Button) findViewById(R.id.btn_delete);  
btnSearch = (Button) findViewById(R.id.btn_search);  
btnUpdate = (Button) findViewById(R.id.btn_update);  
  
tvStatus = (TextView) findViewById(R.id.tv_status);  
tvResult = (TextView) findViewById(R.id.tv_result);
```

3) 给按钮添加事件监听器

```

btnAdd.setOnClickListener(new MyButtonOnClickListener());
btnShowAll.setOnClickListener(new MyButtonOnClickListener());
btnClear.setOnClickListener(new MyButtonOnClickListener());
btnDeleteAll.setOnClickListener(new MyButtonOnClickListener());

btnDelete.setOnClickListener(new MyButtonOnClickListener());
btnSearch.setOnClickListener(new MyButtonOnClickListener());
btnUpdate.setOnClickListener(new MyButtonOnClickListener());

```

4) 编写监听器类, 实现 View.OnClickListener 接口。实现 onClick() 方法, 使用 switch...case 结构, 通过判定被点击按钮的 id 来确定具体执行什么操作。

```

class MyButtonOnClickListener implements View.OnClickListener{
    public void onClick(View v) {
        String msg = "";
        People people = null;
        People[] peoples = null;
        // 判定被点击的按钮
        switch (v.getId()) {
            case R.id.btn_add: // 添加数据按钮
                people = new People();
                people.name = edtName.getText().toString();
                people.age = Integer.parseInt(edtAge.getText().toString());
                people.height = Float.parseFloat(edtHeight.getText().toString());

                long column = dbAdapter.insert(people);
                if(column == -1){
                    tvStatus.setText("插入数据错误！");
                } else {
                    tvStatus.setText("成功插入数据, ID: " + column);
                }
                break;
            case R.id.btn_showAll: // 全部显示按钮
                peoples = dbAdapter.queryAllData();
                if (peoples == null){
                    tvStatus.setText("数据库中没有数据");
                    return;
                }
                tvStatus.setText("数据库: ");
                msg = "";
        }
    }
}

```

```

        for (int i = 0 ; i<peoples.length; i++){
            msg += peoples[i].toString()+"\n";
        }
        tvResult.setText(msg);
        break;
    case R.id.btn_clear: // 清除显示按钮
        tvStatus.setText("");
        tvResult.setText("");
        break;
    case R.id.btn_deleteAll: // 全部删除按钮
        dbAdapter.deleteAllDate();
        msg = "数据全部删除";
        tvResult.setText(msg);
        break;
    case R.id.btn_delete: // ID删除按钮
        long id = Integer.parseInt(edtId.getText().toString());
        long result = dbAdapter.deleteData(id);
        msg = "删除ID为"+edtId.getText().toString()+"的数据" + (result>0
                ?"成功":"失败");
        tvStatus.setText(msg);
        break;
    case R.id.btn_search: // ID查询按钮
        int deleteId = Integer.parseInt(edtId.getText().toString());
        peoples = dbAdapter.queryData(deleteId);
        if (peoples == null){
            tvStatus.setText("数据库中没有ID为"+String.valueOf(deleteId)
                    +"的数据");
            return;
        }
        tvStatus.setText("数据库: ");
        tvResult.setText(peoples[0].toString());
        break;
    case R.id.btn_update: // ID更新按钮
        people = new People();
        people.name = edtName.getText().toString();
        people.age = Integer.parseInt(edtAge.getText().toString());
        people.height = Float.parseFloat(edtHeight.getText().toString());
        int updateId = Integer.parseInt(edtId.getText().toString());
        long count = dbAdapter.updateDate(updateId, people);

```

```
    if (count == -1 ){
        tvStatus.setText("更新错误！ ");
    } else {
        tvStatus.setText("更新成功， 更新数据"+String.valueOf(count)
                +"条");
    }
    break;
}
}
```

步骤六：

运行该程序。在项目上点右键 → Run ‘app’ 按钮。启动模拟器，如图 8.1.2 所示。



图 8.1.2

在编辑框中输入姓名、年龄、身高，如图 8.1.3 所示。

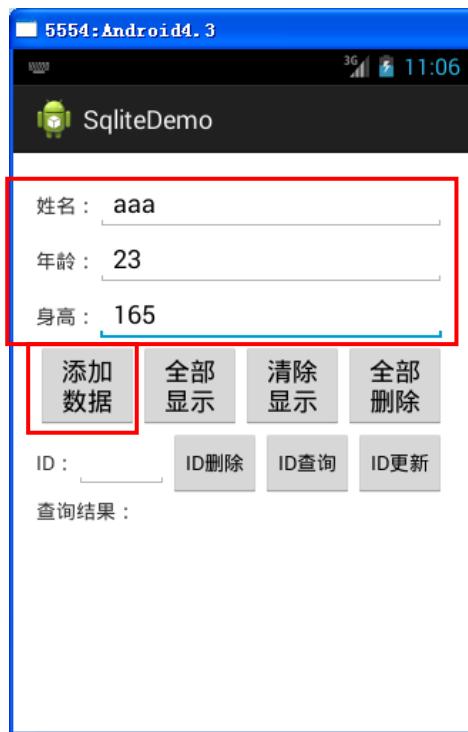


图 8.1.3

点击“添加数据”按钮，如图 8.1.4 所示。

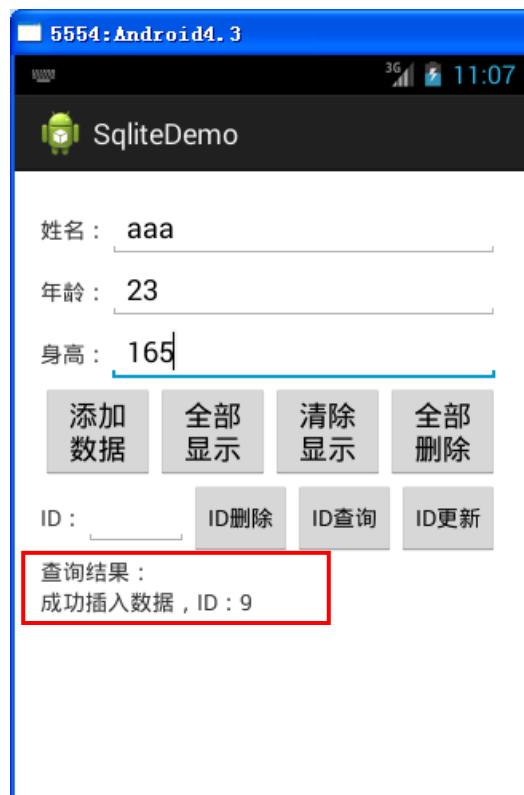


图 8.1.4

点击“清除显示”按钮，在 ID 编辑框输入刚插入的行 ID “9”，点击“ID

查询”按钮，查看查询结果。

修改姓名、年龄、身高信息，点击“ID 更新”按钮，点击“全部显示”按钮，查看结果。

测试其他按钮功能，并观察操作结果。

8.1.4 实验结论

通过本次试验我们学会了 SQLite 的用法

8.2 实验二 Android 中的 Sharedpreferences

8.2.1 实验目的

1. 掌握 SharedPreferences 的使用。

8.2.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

8.2.3 实验步骤

➤ 任务：练习使用 SharedPreferences 和 Editor 接口

实现 SharedPreferences 存储的步骤如下：

- a) 根据 Context 获取 SharedPreferences 对象
- b) 利用 edit() 方法获取 Editor 对象。
- c) 通过 Editor 对象存储 key-value 键值对数据。
- d) 通过 commit() 方法提交数据。

步骤一：

创建“Android Application Project”，项目名称为 SharedPreferences。
创建 Activity 名称为 SharedPreferencesActivity。

步骤二：

编辑 SharedPreferencesActivity 对应的布局文件，要求如图所示效果。



图 8.2.1

关键代码如下所示：

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="25dp"  
    android:text="" />  
<LinearLayout  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:paddingTop="10dp"  
    android:gravity="top/center_horizontal" >  
    <Button  
        android:id="@+id	btn_write"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="写入数据" />  
    <Button  
        android:id="@+id	btn_read"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="读取数据" />  
</LinearLayout>
```

步骤三：

在 SharedPreferencesActivity 中声明对应的 XML 布局文件中的组件及用于

存储信息的 SharedPreferences、Editor 引用对象。

```
EditText etData = null;
Button btnRead = null;
Button btnWrite = null;
//声明SharedPreferences及Editor的引用对象
SharedPreferences pre = null;
SharedPreferences.Editor edt = null;
```

在 onCreate() 方法中得到组件的实例。

```
//得到布局中的组件
etData = (EditText) findViewById(R.id.et_data);
btnRead = (Button) findViewById(R.id.btn_read);
btnWrite = (Button) findViewById(R.id.btn_write);
//获取只能在本程序访问的SharedPreferences对象及Editor对象
pre = getSharedPreferences("MyData", Context.MODE_PRIVATE);
edt = pre.edit();
```

步骤四：

给两个按钮注册点击事件监听器，采用匿名内部类的方式。

```
// 给两个按钮注册点击事件监听器
btnRead.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // 读取字符串数据
        String time = pre.getString("time", null);
        // 读取int类型的数据
        int Num = pre.getInt("num", 0);
        String result = time == null ? "您暂时还未写入数据" : "写入时间为: "
            + time + "\n你上次输入的数为: : " + Num;
        // 使用Toast提示信息
        Toast.makeText(SharedPreferencesActivity.this, result, 5000).show();
    }
});
btnWrite.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        String text = etData.getText().toString();
        int num = Integer.parseInt(text);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 "
            + "hh:mm:ss");
        // 存入当前时间
        edt.putString("time", sdf.format(new Date()));
        // 存入一个数
        edt.putInt("num", num);
        // 提交所有存入的数据
        edt.commit();
    }
});
});
```

步骤五：

运行该程序。在项目上点右键 → Run ‘APP’。启动模拟器，如图 8.2.2 所示。

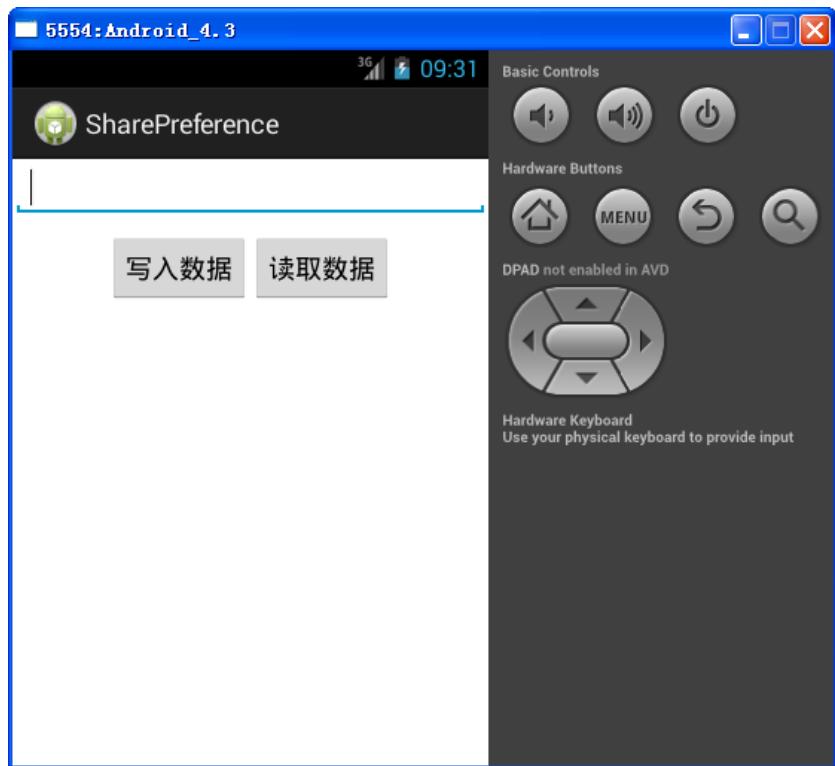


图 8.2.2

在编辑框中输入内容，点击“写入数据”按钮，如图 8.2.3 所示。



图 8.2.3

点击“读取数据”按钮，如图 8.2.4 所示。



图 8.2.4

步骤六：

在 DDMS 视图中查看运行程序之后生成的文件。如图所示。

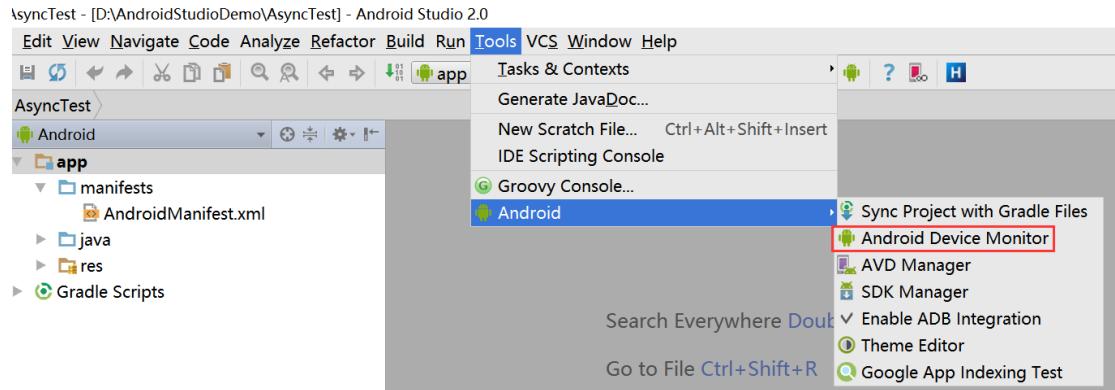


图 8.2.5

点击“File Explorer”选项卡，如图 8.2.6 所示。如果没有该选项卡，可以点击 Eclipse 菜单“Window”→ Show View → File Explore。如图 11 所示。

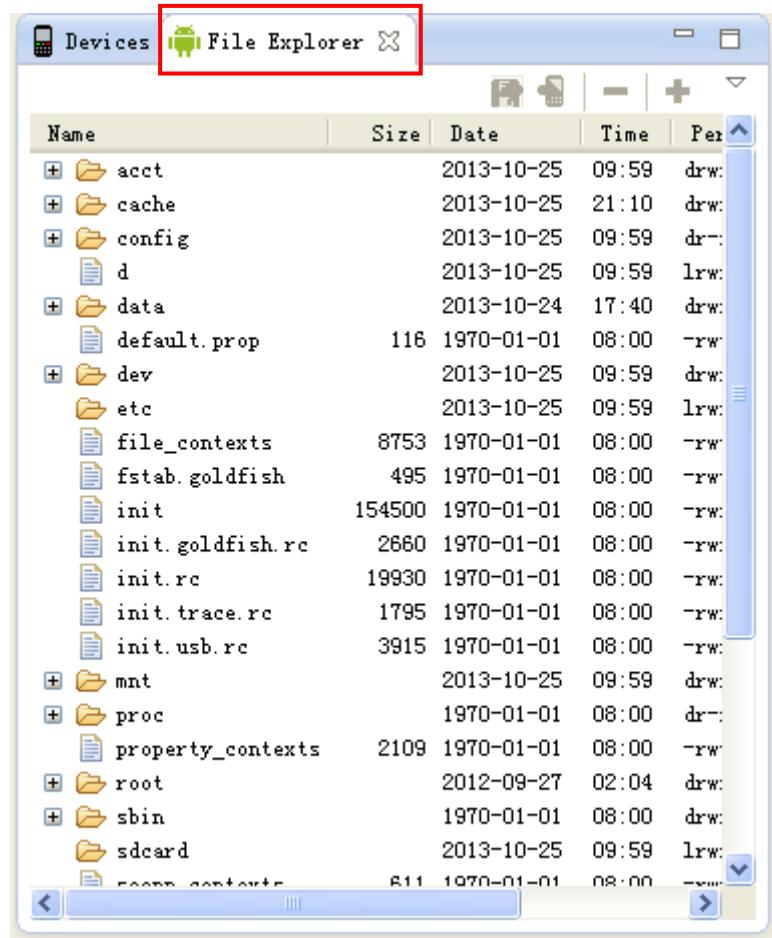


图 8.2.6

在 File Explorer 选项卡下，在 /data/data/net.onest.sharepreference 目录下出现

MyDate.xml 文件。如图 8.2.7 所示。

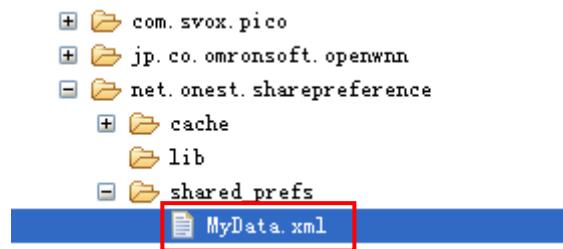


图 8.2.7

选中该 xml 文件，点击选项卡右边的从设备导出图标，如图 8.2.8 所示。查看文件存储的内容。如图 8.2.9 所示。

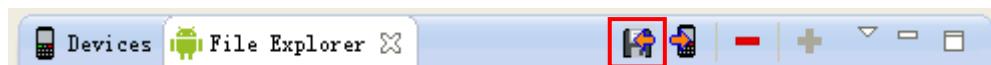


图 8.2.8

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <map>
  <string name="time">2013年10月25日 10:04:25</string>
  <int name="num" value="345" />
</map>
```

图 8.2.9

8.2.4 实验结论

通过本实验要求学生理解 Sharedpreferences 的用法。

8.3 实验三 特殊输入的使用

8.3.1 实验目的

1. 本次试验我们将一起学习 Android 中一些特殊的输入的使用。

8.3.2 准备工作

1. Android 开发环境

8.3.3 实验步骤

Android 中除了简单的点击还有一些其他的比较特殊的输入方式，比如说人性化的交互方式，手势输入，自动朗读等。下面我们通过一个小例子一起来学习下 Android 中的特殊输入的使用，新建工程 SlideTab 其中 MainActivity.java 的代码如下：

```
package com.onest.wyl;

import android.os.Bundle;

import android.app.TabActivity;

import android.content.Intent;

import android.view.GestureDetector;
```

```

import android.view.MotionEvent;

import android.view.View;

import android.view.GestureDetector.SimpleOnGestureListener;

import android.widget.TabHost;

public class MainActivity extends TabActivity {

    private GestureDetector gestureDetector;

    View.OnTouchListener gestureListener;

    private int currentView = 0;

    private static int maxTabIndex = 2;

    /** Called when the activity is first created. */

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        final TabHost tabHost=getTabHost();

        tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("Tab1")

                .setContent(new Intent(this,OneActivity.class)));

        tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator("Tab2")

                .setContent(new Intent(this,TwoActivity.class)));

        tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator("Tab3")

                .setContent(new Intent(this,ThreeActivity.class)));

        gestureDetector = new GestureDetector(new MyGestureDetector());

        gestureListener = new View.OnTouchListener() {

            public boolean onTouch(View v, MotionEvent event) {

                if (gestureDetector.onTouchEvent(event)) {

                    return true;

                }

                return false;

            }

        }

    }

}

```

```

    } ;

}

class MyGestureDetector extends SimpleOnGestureListener {

    @Override

    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
    float velocityY) {

        TabHost tabHost = getTabHost();

        if (e1.getX() - e2.getX() > 120) {

            if (currentView == maxTabIndex) {

                currentView = 2;

            }

            else {

                currentView++;

            }

            tabHost.setCurrentTab(currentView);

        }

        else if (e2.getX() - e1.getX() > 120) {

            if (currentView == 0) {

                currentView = 0;

            }

            else {

                currentView--;

            }

            tabHost.setCurrentTab(currentView);

        }

        return false;
    }
}

```

```
public boolean dispatchTouchEvent(MotionEvent event) {  
    if(gestureDetector.onTouchEvent(event)) {  
        event.setAction(MotionEvent.ACTION_CANCEL);  
    }  
    return super.dispatchTouchEvent(event);  
}  
}
```

需要注意的是要在 `AndroidManifest.xml` 注册跳转的三个 Activity:

```
<application  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme">  
    <activity  
        android:name=".MainActivity"  
        android:label="@string/title_activity_main">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent-filter>  
    </activity>  
    <activity android:name=".OneActivity"></activity>  
    <activity android:name=".TwoActivity"></activity>  
    <activity android:name=".ThreeActivity"></activity>  
</application>
```

运行工程效果如下：



图 8.3.1

下面再让我们做另一种比较常见的特殊输入的例子，新建工程 TouchSlideDemo 其中 MainActivity.java 代码如下：

```
public class MainActivity extends Activity implements OnGestureListener {  
    private GestureDetector detector;  
    private ViewFlipper flipper;  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        detector = new GestureDetector(this);  
        flipper = (ViewFlipper) this.findViewById(R.id.ViewFlipper01);  
        flipper.addView(addView(R.layout.one));  
        flipper.addView(addView(R.layout.two));  
        flipper.addView(addView(R.layout.three));  
        flipper.addView(addView(R.layout.four));  
        flipper.addView(addView(R.layout.five));  
        flipper.addView(addView(R.layout.six));  
    }  
}
```

```
private View addView(int layout) {  
    LayoutInflater inflater = (LayoutInflater)  
        getSystemService(Context.LAYOUT_INFLATER_SERVICE);  
    View view = inflater.inflate(layout, null);  
    return view;  
}  
  
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    Log.i("Fling", "Activity onTouchEvent!");  
    return this.detector.onTouchEvent(event);  
}  
  
@Override  
public boolean onDown(MotionEvent e) {  
    // TODO Auto-generated method stub  
    return false;  
}  
  
/**  
 * 监听滑动  
 */  
  
@Override  
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,  
    float velocityY) {  
    // TODO Auto-generated method stub  
    Log.i("Fling", "Fling Happened!");  
    if (e1.getX() - e2.getX() > 120) {  
        this.flipper.setInAnimation(AnimationUtils.loadAnimation(this, R.anim.push_left_in));  
        this.flipper.setOutAnimation(AnimationUtils.loadAnimation(this, R.anim.push_left_out));  
    }  
}
```

```
    this.flipper.showNext();

    return true;

} else if (e1.getX() - e2.getX() < -120) {

    this.flipper.setInAnimation(AnimationUtils.loadAnimation(this, R.anim.push_right_in));

    this.flipper.setOutAnimation(AnimationUtils.loadAnimation(this, R.anim.push_right_out));

    this.flipper.showPrevious();

    return true;

}

return false;
}

@Override

public void onLongPress(MotionEvent e) {

// TODO Auto-generated method stub

}

@Override

public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
float distanceY) {

// TODO Auto-generated method stub

return false;

}

@Override

public void onShowPress(MotionEvent e) {

// TODO Auto-generated method stub

}

@Override

public boolean onSingleTapUp(MotionEvent e) {

// TODO Auto-generated method stub
```

```
    return false;  
}  
}  
}
```

运行工程，效果如图：

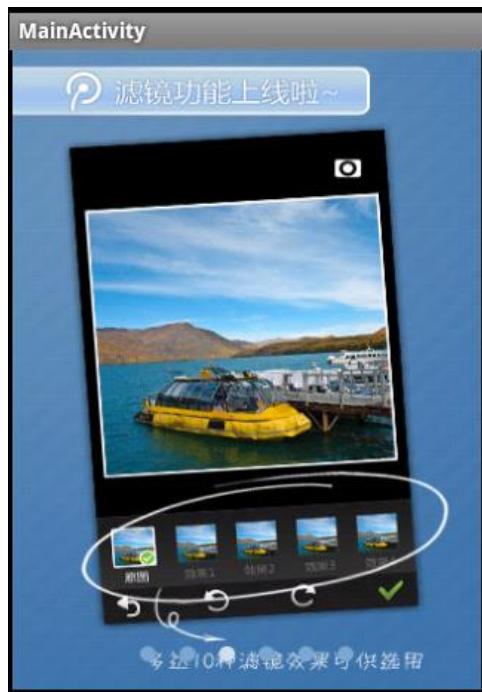


图 8.3.2



图 8.3.3



图 8.3.4

8.3.4 实验结论

通过本次试验我们学会了使用 Android 中一些特殊的输入。

8.4 作业

8.4.1 完善学生管理系统，在登录页面加上 SharedPreferences

要求完善学生管理系统，在登录页面加上 SharedPreferences，可以记住用户的登录名和密码。

8.4.2 完善学生管理系统，数据使用 SQLite 存储

要求完善学生管理系统，使用 SQLite 存储学生的姓名、性别、班级、联系方式等相关数据，当打开学生列表时会从数据库读取数据。

第9章 Android 中的 ContentProvider 的实现

由于 Android 中的数据大多是私有的，不利于不同应用之间的数据共享。ContentProvider 主要用于实现不同应用程序之间数据的传递和共享。

本实验的主要内容为对 Android 中数据共享的标准的介绍，对 Android 中操作系统的 ContentProvider 的介绍，对实现 ContentProvider 的介绍，对监听 ContentProvider 的数据的介绍。

通过本章实验我们能够理解 Android 中 ContentProvider 的使用场景及在 Android 中 ContentProvider 的操作，我们能够调用系统 URI 并可以制作自己的 URI。

➤ ContentProvider 特点简介

1. ContentProvider 为存储和获取数据提供了统一的接口。
ContentProvide 对数据进行封装，不用关心数据存储的细节。使用表的形式来组织数据。
2. 使用 ContentProvider 可以在不同的应用程序之间共享数据。
3. Android 为常见的一些数据提供了默认的 ContentProvider（包括音频、视频、图片和通讯录等）。

➤ ContentProvider 所提供的函数

query(), insert(), update(), delete(), getType(), onCreate() 等。

➤ ContentProvider 的数据模型如下

表 9.0.1 ContentProvider 数据模型

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland	Bully	TYPE_WORK

			office	Pulpit	
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201. 555. 4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

➤ ContentResolver

Content Providers 会实现一些共同的接口，包括数据的查询、添加、更改、删除。应用通过唯一的 ContentResolver 接口来使用某个 Content Provider。ContentResolver 对象通过 `getContentResolver()` 方法来取得，继而通过调用 ContentResolver 提供的方法操作 ContentProvider。

```
ContentResolver cr = getContentResolver();
```

用户不需要同 ContentProvider 对象直接打交道，对于每一种类型的 ContentProvider 只有一个实例，它可以与在不同程序或进程中的多个 ContentResolver 对象进行通信。

➤ URI（统一资源标识符）的使用方法

为系统的每一个资源给其一个名字，比方说通话记录。

- a) 每一个 ContentProvider 都拥有一个公共的 URI，这个 URI 用于表示这个 ContentProvider 所提供的数据。
- b) Android 所提供的 ContentProvider 都存放在 `android.provider` 包中。将其分为 A, B, C, D 4 个部分：(如下图所示)

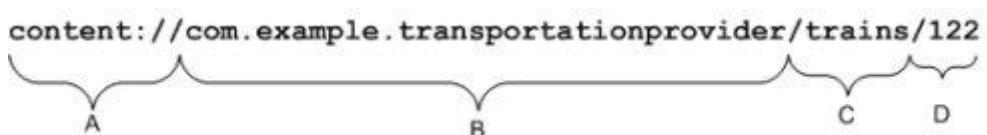


图 9.0.1 Android 系统中的 URI 格式

- A. “`content://`”，标准前缀，用来说明一个 Content Provider 控制这些数据，固定不变的；
- B. 数据的路径（URI 的标识），它定义了是哪个 Content Provider 提供这

些数据。对于第三方应用程序，为了保证 URI 标识的唯一性，它必须是一个完整的、小写的类名。这个标识在 `元素的 authorities 属性中说明：一般是定义该 ContentProvider 的包 . 类 的 名称；"content://hx.android.text.myprovider"`

C. 标识 ID（可选），通俗的讲就是要操作的数据库中表的名字（也可以自己定义，但在使用的时候要保持一致）；
"content://hx.android.text.myprovider/tablename"

D. 如果 URI 中包含表示需要获取的记录的 ID，则就返回该 id 对应的数据，如果没 ID，就表示返回全部；
"content://hx.android.text.myprovider/tablename/#" #表示数据 id

例如以下是系统的一些 URI：

content://media/internal/images (该 URI 返回设备上存储的所有图片)
content://contacts/people/5 (联系人信息中 ID 为 5 的联系人记录)
content://contacts/people (该 URI 返回设备上的所有联系人信息)

➤ 扩展

android.provider 包下提供了一系列的辅助类，其中包含了很多以变量形式给出的查询字符串。上面第 2 个 URI 可以改写成以下方式：

```
Uri person = ContentUris.withAppendedId(People.CONTENT_URI, 5)
```

通过实验学习类和对象的定义方式，体会面向对象的主要特征（封装、继承、多态），学习抽象类和接口的定义方式，学习内部类和包装器类的使用方式。

9.1 实验一 Android 中 ContentProvider 使用

9.1.1 实验目的

1. 掌握 Android 中 ContentProvider 的使用。
2. 使用 ContentResolver 操作系统 ContentProvider 所暴露的数据。

9.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

9.1.3 实验步骤

➤ 任务：调用系统提供的联系人的 ContentProvider 的 uri，查看联系人
通过 ContentResolver 访问系统 CP 的步骤：

- a) 调用 Activity 的 ContentResolver 获取 CR 对象
- b) 调用 CR 中的 insert, delete, update, query 方法获得系统 CP 提供的数据

`insert(Uri uri, ContentValues values)`：向 Uri 对应的 ContentProvider 中插入 values 对应的数据；

`delete(Uri uri, String where, String[] selectionArgs)`：删除 Uri 对应的 ContentProvider 中 where 提交匹配的数据；

`update(Uri uri, ContentValues values, String where, String[] selectionArgs)`：更新 Uri 对应的 Contentprovider 中 where 提交匹配的数据；

`query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`：查询 Uri 对应的 ContentProvider 中 where 提交匹配的数据。

注：操作系统的 ContentResolver，需要先了解 ContentProvider 的 Uri。

一般来说，ContentProvider 是单实例模式的，当多个应用程序通过 ContentResolver 来操作 ContentProvider 提供的数据时，ContentResolver 调用的数据操作将会委托给同一个 ContentProvider 处理。

步骤一：

新建工程 ContentProviderTest，Activity 名字 MainActivity。目录结构如图 9.1.1 所示。

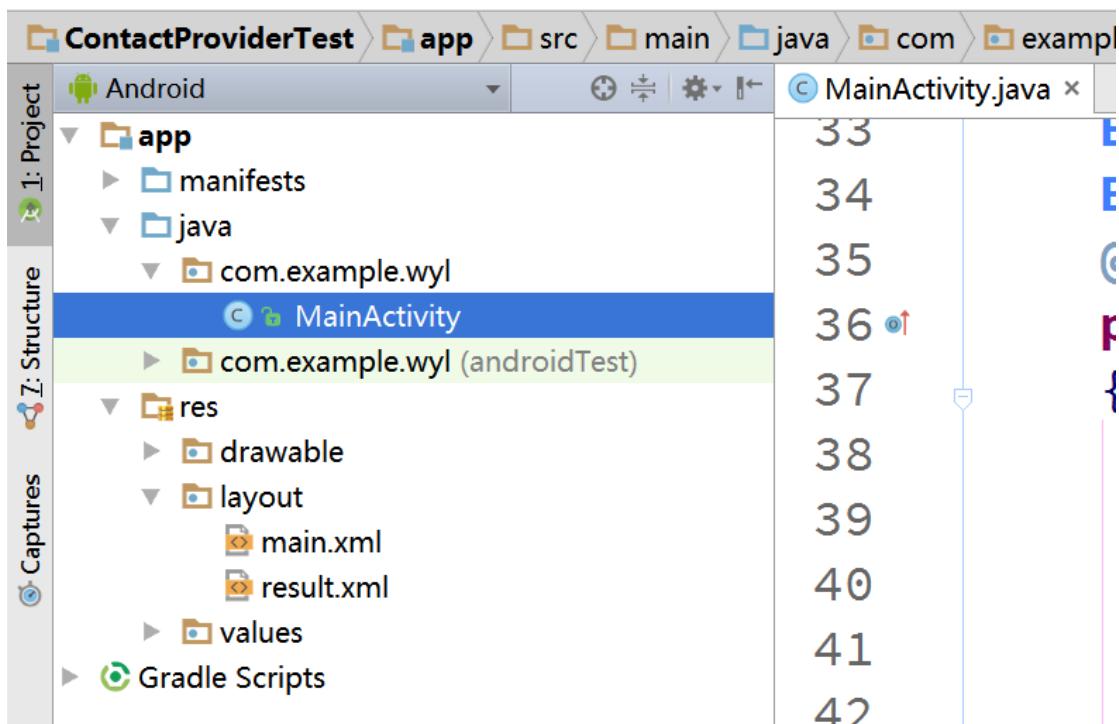


图 9.1.1

步骤二：

编辑 activity_main.xml 页面。如图 9.1.2 所示。

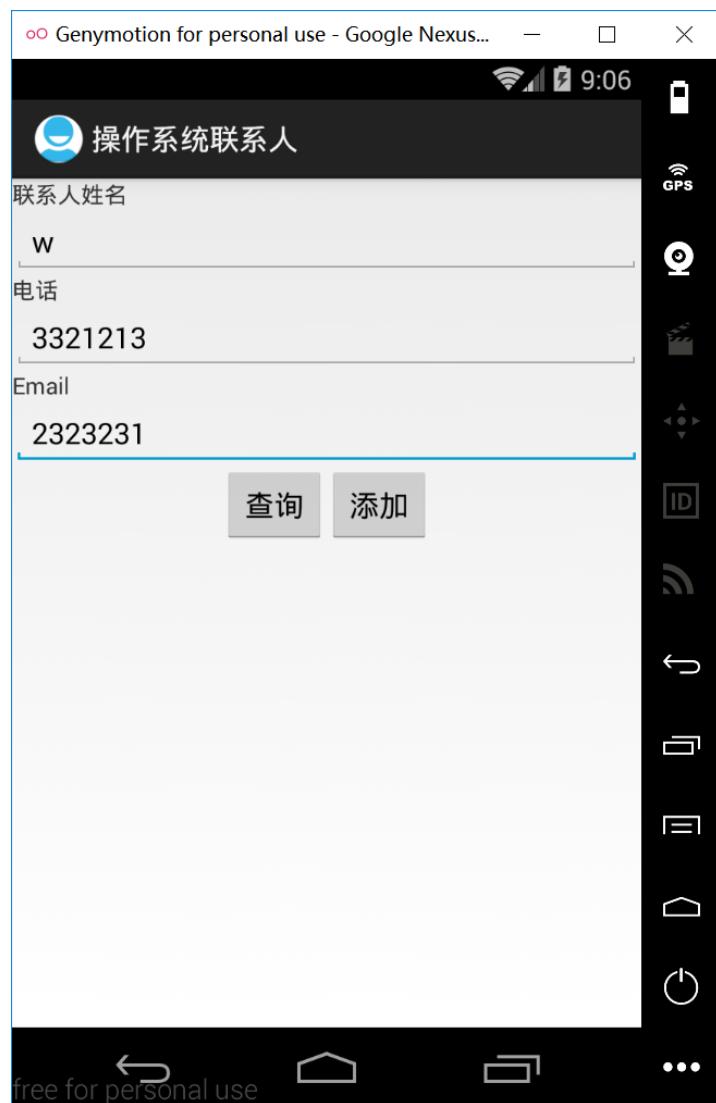


图 9.1.2

其中各控件的属性如表 1 所示。

表 9.1.1 布局文件控件属性表

属性 控件类型	<code>id</code>	<code>text</code>	<code>hint</code>
TextView		姓名	
TextView		电话	
TextView		E-mail	
EditText	<code>edtName</code>		请填写您的姓名
EditText	<code>edtTel</code>		请填写您的电话号码

EditText	edtEmail	请填写您的电子邮箱
Button	btnContacts	查询联系人
Button	btnAdd	添加联系人

添加一个布局文件 result.xml，用于显示查询的结果。该页面包含一个 ExpandableListView 组件。关键代码如下：

```
<ExpandableListView
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true" >
</ExpandableListView>
```

注：该页面使用的是相对布局。

步骤三：

在 MainActivity 中声明两个按钮和三个文本框。（位置：“public class ContentProviderTest extends Activity {”之后，“@Override”之前）

```
private Button btnContacts;
private Button btnAdd;
private EditText edtName;
private EditText edtTel;
private EditText edtEmail;
```

在覆盖的 onCreate 方法中获取布局文件中的两个按钮组件对象

```
btnContacts = (Button) findViewById(R.id.btnContacts);
btnAdd = (Button) findViewById(R.id.btnAdd);
```

步骤四：

给查询联系人按钮绑定事件，采用匿名内部类的方式。

1) 首先把所有的联系人都查询出来。需要定义 2 个数组，一个用于保存联系人的姓名的字符串数组，另一个是用于保存对应联系人的详细信息的字符串数组的数组。

```

btnContacts.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //定义两个List来封装系统的联系人信息，指定联系人的电话号码，E-mail等详情
        final ArrayList<String> names = new ArrayList<String>();
        final ArrayList<ArrayList<String>> details = new ArrayList<ArrayList<String>>();
        //使用ContentResolver查找联系人数据
        Cursor cursor = getContentResolver()
            .query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
    }
}

```

2) 使用循环便利查询结果，得到每个联系人的 ID 和姓名。

```

//遍历查询结果，获取系统中所有联系人
while(cursor.moveToNext()){
    //获取联系人ID
    String contactId = cursor.getString(cursor
        .getColumnIndex(ContactsContract.Contacts._ID));
    //获取联系人的名字
    String name = cursor.getString(cursor
        .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
    names.add(name);
}

```

3) 获取联系人的电话号码，由于每个联系人的电话号码有可能不止一个，所以还需要对电话号码循环遍历，得到某个联系人的所有电话号码。

```

//使用ContentResolver查找联系人的电话号码
Cursor phones = getContentResolver().query(ContactsContract
    .CommonDataKinds.Phone.CONTENT_URI
    ,null,ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "="
    + contactId,null,null);
ArrayList<String> detail = new ArrayList<String>();
//遍历查询结果，获取该联系人的多个电话号码
while (phones.moveToNext()) {
    //获取查询结果中电话号码列中的数据
    String phoneNumber = phones.getString(phones
        .getColumnIndex(ContactsContract.CommonDataKinds
        .Phone.NUMBER));
    detail.add("电话号码：" + phoneNumber);
}
phones.close();

```

4) 获取联系人的电子邮件地址，邮件地址和电话号码类似，一个联系人可能不止一个邮箱，所以同样需要使用循环遍历邮箱列表。最后，关闭查询游标。

```
//使用ContentResolver查找联系人的E-mail地址
Cursor emails = getContentResolver().query(ContactsContract
    .CommonDataKinds.Email.CONTENT_URI,null
    ,ContactsContract.CommonDataKinds.Email.CONTACT_ID
    + "=" + contactId, null, null);
//遍历查询结果，获取该联系人的多个E-mail地址
while(emails.moveToNext()){
    //获取查询结果中Email地址列中的数据
    String email = emails.getString(emails
        .getColumnIndex(ContactsContract.CommonDataKinds
        .Email.DATA));
    detail.add("邮件地址：" + email);
}
emails.close();
details.add(detail);
}
cursor.close();
```

5) 加载 result.xml 布局文件并获取布局文件中的组件。

```
//加载result.xml界面布局代表的视图
View resultDialog = getLayoutInflater().inflate(R.layout.result, null);
//获取resultDialog中ID为list的ExpandableListview
ExpandableListView list = (ExpandableListView) resultDialog
    .findViewById(R.id.list);
```

6) 创建 ExpandableListAdapter 对象，定义其具体样式。需要实现几个方法。

```
//创建一个ExpandableListAdapter对象
ExpandableListAdapter adapter = new BaseExpandableListAdapter(){
    //获取指定组位置，指定子列表项处的子列表项数据
    @Override
    public Object getChild(int groupPosition, int childPosition) {
        return details.get(groupPosition).get(childPosition);
    }
    @Override
    public long getChildId(int groupPosition, int childPosition) {
        return childPosition;
    }
}
```

```
@Override
public int getChildrenCount(int groupPosition) {
    return details.get(groupPosition).size();
}

private TextView getTextView(){
    AbsListView.LayoutParams lp = new AbsListView
        .LayoutParams(ViewGroup.LayoutParams.FILL_PARENT,64);
    TextView textView = new TextView(MainActivity.this);
    textView.setLayoutParams(lp);
    textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
    textView.setPadding(36, 0, 0, 0);
    textView.setTextSize(20);
    return textView;
}

//该方法决定每个子选项的外观
@Override
public View getChildView(int groupPosition, int childPosition,
    boolean isLastChild, View convertView, ViewGroup parent) {
    TextView textView = getTextView();
    textView.setText(getChild(groupPosition, childPosition).toString());
    return textView;
}

//获取指定组位置处的组数据
@Override
public Object getGroup(int groupPosition) {
    return names.get(groupPosition);
}

@Override
public int getGroupCount() {
    return names.size();
}

@Override
public long getGroupId(int groupPosition) {
    return groupPosition;
}

//该方法决定每个组选项的外观
@Override
public View getGroupView(int groupPosition, boolean isExpanded,
    View convertView, ViewGroup parent) {
    TextView textView = getTextView();
    textView.setText(getGroup(groupPosition).toString());
    return textView;
}
```

```

@Override
public boolean isChildSelectable(int groupPosition, int childPosition) {
    return true;
}
@Override
public boolean hasStableIds() {
    return true;
}
};

```

7) 给 result 布局文件中的 ExpandableListView 组件设置 Adapter 对象，并用对话框将联系人信息显示出来。

```

//为ExpandableListView设置Adapter对象
list.setAdapter(adapter);
//使用对话框来显示查询结果
new AlertDialog.Builder(MainActivity.this).setView(resultDialog)
    .setPositiveButton("确定", null).show();
}
});

```

注：在单击事件中主要实现了如下几个步骤：

1) 获取 ContentResolver 对象；

```
ContentResolver contentResolver = getContentResolver();
```

2) 通过 ContentResolver 对象的 query () 方法获取所有联系人，

```
Cursor cursor = contentResolver.query(ContactsContract
```

```
    .Contacts.CONTENT_URI, null, null, null, null);
```

3) 循环遍历所有联系人（分别遍历某联系人的所有电话号码和电子邮件）

使用 ContentResolver 向 ContactsContract.Contact.CONTENT_URI 查询数据，可查询出系统中所有的联系人信息；

使用 ContentResolver 向 ContactsContract.CommonDataKinds.Phone.CONTENT_URI 查询数据，可查询出指定联系人的所有电话信息；

使用 ContentResolver 向 ContactsContract.CommonDataKinds.Email.CONTENT_URI 查询数据，可查询出指定联系人的所有电子邮箱信息；

步骤五：

在 AndroidManifest.xml 文件的<manifest>节点中添加读联系人的权限。

```
<!-- 授予读联系人ContentProvider的权限 -->  
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

步骤六：

运行该项目，如图 9.1.3 所示。

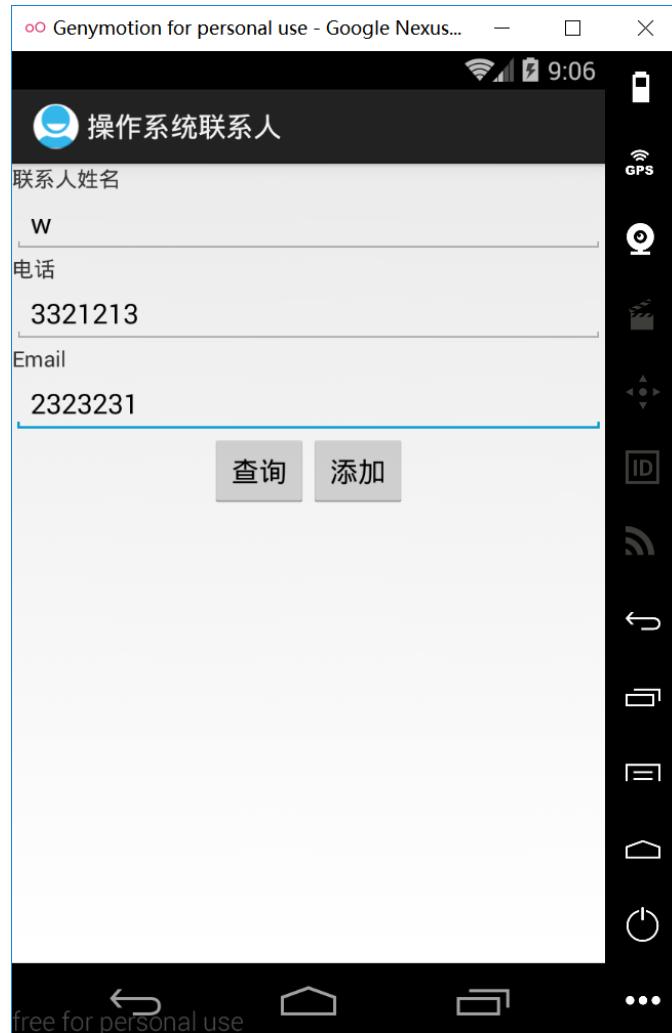


图 9.1.3

点击“查询联系人”按钮，将显示系统内的联系人列表。点击张三联系人左边的箭头，显示张三的详细信息。如图所示。



图 9.1.4

- 任务：调用系统 URI 添加联系人。本例子使用 ContentResolver 操作 ContentProvider，添加联系人。
在任务一的基础上增加添加联系人的功能。

步骤一：

给添加联系人按钮绑定事件监听器（采用内部类的方法）。

```
btnAdd.setOnClickListener(new AddClickListener());
```

步骤二：

编写 AddClickListener() 事件监听器类，需要覆写 onClick() 方法

```
class AddClickListener implements OnClickListener{  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
    }  
}
```

步骤三：

在覆盖 onClick() 方法中获取程序界面中的三个文本框数据

```
edtName = (EditText) findViewById(R.id.edtName);  
edtTel = (EditText) findViewById(R.id.edtTel);  
edtEmail = (EditText) findViewById(R.id.edtEmail);  
String name = edtName.getText().toString();  
String phone = edtTel.getText().toString();  
String email = edtEmail.getText().toString();
```

步骤四：

插入数据之前需要创建一个 ContentValues 对象，用于存放将要插入的数据

```
//创建一个空的ContentValues  
ContentValues values = new ContentValues();  
//向RawContacts.CONTENT_URI执行一个空值插入  
//目的是获取系统返回的RawContactId  
Uri rawContactUri = getContentResolver().insert(RawContacts.CONTENT_URI, values);  
long rawContactId = ContentUris.parseId(rawContactUri);  
values.clear();  
values.put(Data.RAW_CONTACT_ID, rawContactId);
```

步骤五：

将联系人名字存放在 ContentValues 对象的实例 values 中，并将其插入到联系人 Uri

```
//设置内容类型  
values.put(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE);  
//设置联系人名字  
values.put(StructuredName.GIVEN_NAME, name);  
//向联系人Uri添加联系人名字  
getContentResolver().insert(android.provider.ContactsContract.Data.CONTENT_URI, values);
```

步骤六：

清除 values 中数据（为存放联系人的电话号码做准备），将电话号码插入到联系人电话 Uri

```
values.clear();
values.put(Data.RAW_CONTACT_ID, rawContactId);
values.put(Data.MIMETYPE, Phone.CONTENT_ITEM_TYPE);
//设置联系人的电话号码
values.put(Phone.NUMBER, phone);
//设置电话类型
values.put(Phone.TYPE, Phone.TYPE_MOBILE);
//向联系人电话号码Uri添加电话号码
getContentResolver().insert(android.provider.ContactsContract.Data.CONTENT_URI, values);
```

步骤七：

清除 values 中数据（为存放联系人的电子邮箱做准备），将 E-mail 插入到联系人邮件 Uri

```
values.clear();
values.put(Data.RAW_CONTACT_ID, rawContactId);
values.put(Data.MIMETYPE, Email.CONTENT_ITEM_TYPE);
//设置联系人的Email地址
values.put(Email.DATA, email);
//设置该电子邮件的类型
values.put(Email.TYPE, Email.TYPE_WORK);
//向联系人E-mail Uri添加E-mail数据
getContentResolver().insert(android.provider.ContactsContract.Data.CONTENT_URI, values);
//使用Toast提示用户成功插入数据
Toast.makeText(ContentProviderTest.this, "联系人数据添加成功", 8000).show();
```

注：至此，添加联系人的监听器类的业务逻辑就编写完成了。

步骤八：

在 AndroidManifest.xml 文件中授予写联系人 ContentProvider 的权限：

```
<!-- 授予写联系人ContentProvider的权限 -->
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
```

步骤九：

运行该程序，输入新的联系人的信息，包括名称、电话、邮件地址。点击“添加联系人”按钮。如图 9.1.5 所示。



图 9.1.5

再次点击“查询联系人”按钮，查看联系人列表。

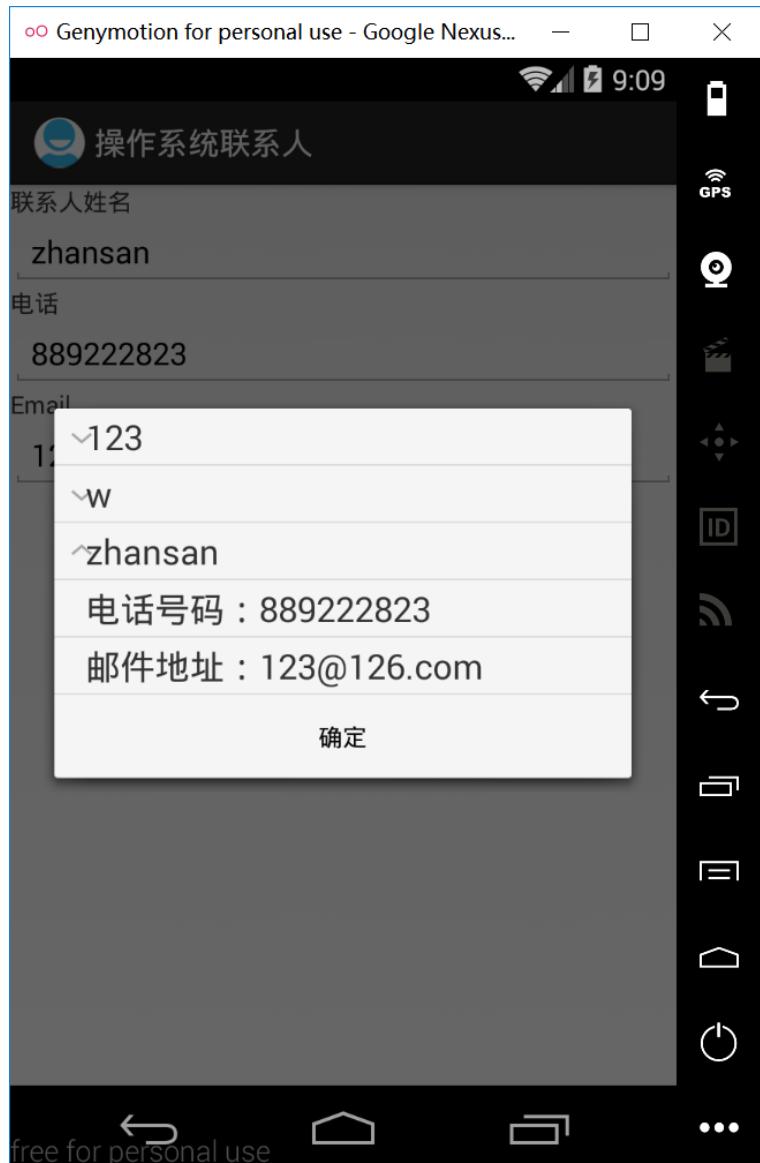


图 9.1.6

9.1.4 实验结论

希望各位同学能够通过本实验能够深入理解 ContentProvider 的建立及访问。

9.2 实验二 Android 中自定义 ContentProvider

9.2.1 实验目的

1. 掌握 Android 中 ContentProvider 的使用。
2. 掌握自定义 ContentResolver 的基本步骤。

9.2.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

9.2.3 实验步骤

自定义 ContentProvider，实现一个单词本的功能

基本步骤：

- 1) 开发单词本应用，设计数据库。
- 2) 添加 ContentProvider 子类，开放数据库数据。
- 3) 注册自定义 ContentProvider，编写新的工程调用 ContentProvider 提供的数据。

步骤一：

新建工程 DictProvider，Activity 名字 DictActivityTest。

注：ContentProvider 是用于给其他应用提供数据的，本任务将实现单词本的功能，测试单词本功能的应用编写在同一个应用中。学生可以尝试，在不同的应用中访问单词本所提供的数据。

步骤二：

设计单词本的工具类 Words.java。该类中通常会把 ContentProvider 的 Uri、数据列等数据信息以常量的形式开放，方便访问。我们在该类中定义一个 public static 的常量，一个静态的内部类。在静态内部类中定义数据列常量以及要用

到的两个 Uri。这个工具类的主要作用是提供了其他应用程序可以访问 ContentProvider 的常用入口。

```
// 定义ContentProvider的Authority静态常量  
public static final String AUTHORITY = "com.example.providers.dictprovider";
```

```
// 定义一个静态内部类  
public static final class Word implements BaseColumns {  
    // 定义Content所允许操作的三个数据列  
    public final static String _ID = "_ID";  
    public final static String WORD = "word";  
    public final static String DETAIL = "detail";  
    // 定义该Content提供服务的两个Uri  
    public final static Uri DICT_CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/words");  
    public final Uri WORD_CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/word");  
}
```

注：Words 工具类不是必须定义的，可在 ContentProvider 中直接使用字符串来定义提供服务的 Uri，但是需要用文档的形式，告诉其他应用程序访问该 ContentProvider 的入口。但是这种方式的缺点是可维护性不好。

步骤三：

创建 SQLiteOpenHelper 的子类 MyDatabaseHelper 来管理数据库连接，继承自 SQLiteOpenHelper。

```
//定义建表字符串：  
final String CREATE_TABLE_SQL = "create table dict(_id integer primary key autoincrement, word , detail)";  
//编写该类的构造方法：用父类的方法初始化子类的对象  
public MyDatabaseHelper(Context context, String name, int version) {  
    super(context, name, null, version);  
}  
@Override  
public void onCreate(SQLiteDatabase db) {  
    //第一个使用数据库时自动建表  
    db.execSQL(CREATE_TABLE_SQL);  
}  
@Override  
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    System.out.println("----onUpdate Called----" + oldVersion + "--->" + newVersion);  
}
```

注：该类的主要作用是管理数据库的初始化，允许应用程序通过该类获取 SQLiteDatabase 对象，并利用获取的数据库对象操作数据库。

步骤四：

实现自己的 ContentProvider (DictProvider)，继承 ContentProvider。需要重写其增、删、改、查等方法。

- 8) 首先在 DictProvider 中定义 UriMatcher 并为 UriMatcher 注册两个 Uri

```
private static UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
private static final int WORDS = 1;
private static final int WORD = 2;
private MyDatabaseHelper dbOpenHelper;
static {
    // 为UriMatcher注册两个Uri
    matcher.addURI(Words.AUTHORITY, "words", WORDS);
    matcher.addURI(Words.AUTHORITY, "word/#", WORD);
}
```

- 9) 覆写 onCreate、insert、update、delete、query 和 getType 方法。第一次调用该 DictProvider 时，系统先创建 DictProvider 对象，回调 onCreate() 方法。因此在该方法中编写创建数据库表的逻辑代码。

```
@Override
public boolean onCreate() {
    dbOpenHelper = new MyDatabaseHelper(this.getContext(), "myDict.db3", 1);
    return true;
}
```

重写插入数据的 insert() 方法。

```

// 插入数据
public Uri insert(Uri uri, ContentValues values) {
    SQLiteDatabase db = dbOpenHelper.getReadableDatabase(); // 获得数据库实例
    long rowId = db.insert("dict", Words.Word._ID, values); // 插入数据，返回行ID
    if (rowId > 0) { // 如果插入成功，返回uri
        // 在已有的Uri后面追加ID数据
        Uri wordUri = ContentUris.withAppendedId(uri, rowId);
        // 通知数据已经改变
        getContext().getContentResolver().notifyChange(wordUri, null);
        return wordUri;
    }
    return null;
}

```

重写删除数据的 delete() 方法。

```

// 删除数据的方法
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
    // 记录所删除的记录数
    int num = 0;
    // 对于uri进行匹配。
    switch (matcher.match(uri)) {
        case WORDS:
            num = db.delete("dict", selection, selectionArgs);
            break;
        case WORD:
            // 解析出所需要删除的记录ID
            long id = ContentUris.parseId(uri);
            String where = Words.Word._ID + "=" + id;
            // 如果原来的where子句存在，拼接where子句
            if (selection != null && !selection.equals("")) {
                where = where + " and " + selection;
            }
            num = db.delete("dict", where, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }
    // 通知数据已经改变
    getContext().getContentResolver().notifyChange(uri, null);
    return num;
}

```

重写更新数据的 update() 方法。

```
// 修改数据的方法
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
    SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
    // 记录所修改的记录数
    int num = 0;
    switch (matcher.match(uri)) {
        case WORDS:
            num = db.update("dict", values, selection, selectionArgs);
            break;
        case WORD:
            // 解析出想修改的记录ID
            long id = ContentUris.parseId(uri);
            String where = Words.Word._ID + "=" + id;
            // 如果原来的where子句存在，拼接where子句
            if (selection != null && !selection.equals("")) {
                where = where + " and " + selection;
            }
            num = db.update("dict", values, where, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }
    // 通知数据已经改变
    getContext().getContentResolver().notifyChange(uri, null);
    return num;
}
```

重写返回 MIMI 类型的 getType() 方法。

```
// 返回指定uri参数对应的数据的MIME类型
public String getType(Uri uri) {
    switch (matcher.match(uri)) {
        // 如果操作的数据是多项记录
        case WORDS:
            return "vnd.android.cursor.dir/com.example.dict";
        // 如果操作的数据是单项记录
        case WORD:
            return "vnd.android.cursor.item/com.example.dict";
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }
}
```

重写更新数据的 query() 方法。

```
// 查询数据的方法
@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String sortOrder) {
    SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
    switch (matcher.match(uri)) {
        case WORDS:
            // 执行查询
            return db.query("dict", projection, selection, selectionArgs, null,
                null, sortOrder);
        case WORD:
            // 解析出想查询的记录ID
            long id = ContentUris.parseId(uri);
            String where = Words.Word._ID + "=" + id;
            // 如果原来的where子句存在，拼接where子句
            if (selection != null && !"".equals(selection)) {
                where = where + " and " + selection;
            }
            return db.query("dict", projection, where, selectionArgs, null,
                null, sortOrder);
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }
}
```

步骤五：

在 AndroidManifest.xml 文件中注册该 ContentProvider。

```
<!-- 注册一个ContentProvider -->
<provider
    android:name=".DictProvider" android:authorities="com.example.providers.dictprovider" />
```

注：注册语句添加在 application 标签里

至此，ContentProvider 开发完成，为了测试该 ContentProvider 需要开发另一个应用程序，通过 ContentProvider 来操作生词本中的数据。接下来就直接在本应用中测试以上开发的 ContentProvider，学生也可以另外创建一个应用来测试。

步骤六：

编辑 DictActivityTest 所对应的布局文件，各组件的主要属性如表 9.2.1

所示。布局文件的效果如图 1 所示。

表 9.2.1 布局文件各组件属性表

组件类型	ID	TEXT/HINT	组件类型	ID	TEXT
TextView		单词本	Button	btnAdd	添加单词
TextView		单 词:	Button	btnSearc	查询单词
TextView		释 义:	Button	btnDelete	删除单词
EditText	edtWord	请输入单词	Button	btnUpdate	修改单词
EditText	edtExpl	请输入单词的释义			
EditText	edtSearc	请输入要查找/删除 h /修改的单词			



图 9.2.1 布局文件界面

步骤七：

编辑 DictActivityTest 文件。

- 1) 声明各组件及 ContentResolver 对象

```

//声明ContentResolver对象
ContentResolver contentResolver;
//声明组件
private Button btnAdd = null; //插入单词按钮
private Button btnSearch = null; //查询单词按钮
private Button btnDelete = null; //删除单词按钮
private Button btnUpdate = null; //更新单词按钮
private Button btnUpdateOk = null; //确认更新单词按钮

private EditText edtWord = null;
private EditText edtExpl = null;
private EditText edtSearch = null;

```

2) 在 onCreate ()方法中获取各组件及 ContentResolver 对象。

```

/**
 * 创建MyDatabaseHelper对象，指定数据库版本为1，此处使用相对路径即可
 * 数据库文件会自动保存在程序的数据文件夹的Databases目录下
 * dbHelper = new MyDatabaseHelper(this, "myDic.db3", 1);
 */
// 获取系统的ContentResolver对象
contentResolver = getContentResolver();
//获取程序界面中的三个文本框
edtWord = (EditText) findViewById(R.id.edtWord);
edtExpl = (EditText) findViewById(R.id.edtExpl);
edtSearch = (EditText) findViewById(R.id.edtSearch);
//获取添加单词、查询单词、删除单词、更新单词、确认更新单词按钮
btnAdd = (Button) findViewById(R.id.btnAdd);
btnSearch = (Button) findViewById(R.id.btnSearch);
btnDelete = (Button) findViewById(R.id.btnDelete);
btnUpdate = (Button) findViewById(R.id.btnUpdate);
btnUpdateOk = (Button) findViewById(R.id.btnUpdatOk);

```

3) 给各按钮注册事件监听器

```

//给添加生词按钮绑定事件监听器
btnAdd.setOnClickListener(new AddWord());
//给查找单词按钮绑定事件监听器
btnSearch.setOnClickListener(new SearchWord());
//给删除单词按钮绑定事件监听器
btnDelete.setOnClickListener(new DeleteWord());
//给更新单词按钮绑定事件监听器
btnUpdate.setOnClickListener(new UpdateWord());

```

4) 实现各个事件监听器类

```
//添加生词事件监听器
private class AddWord implements View.OnClickListener{
    @Override
    public void onClick(View arg0) {
        //获取用户输入
        String word = edtWord.getText().toString();
        String detail = edtExpl.getText().toString();
        //清空两个输入框
        edtWord.setText("");
        edtExpl.setText("");
        //插入生词记录
        ContentValues values = new ContentValues();
        values.put(Words.Word.WORD , word);
        values.put(Words.Word.DETAIL , detail);
        contentResolver.insert(Words.Word.DICT_CONTENT_URI, values);
        Toast.makeText(DictActivityTest.this, "添加生词" + word + "成功！"
                , 8000).show();           //显示提示信息
    }
}

//添加更新单词事件监听器
private class UpdateWord implements OnClickListener{
    @Override
    public void onClick(View v) {
        String key = edtSearch.getText().toString(); //获取用户输入
        //查询要更新的单词
        Cursor cursor = contentResolver.query(Words.Word.DICT_CONTENT_URI, null
                , "word = ?", new String[]{key}, null);
        cursor.moveToFirst();
        String word = cursor.getString(1);
        String detail = cursor.getString(2);
        //显示数据
        edtWord.setText(word);
        edtExpl.setText(detail);
        edtSearch.setText(""); //清空输入框
        btnUpdateOk.setVisibility(View.VISIBLE); //设置确认更新按钮可见
        //给确认更新按钮添加事件监听器
        btnUpdateOk.setOnClickListener(new UpdateWordOk());
    }
}
```

```
//确认更新事件监听器
private class UpdateWordOk implements OnClickListener {
    @Override
    public void onClick(View v) {
        //获取用户输入
        String word = edtWord.getText().toString();
        String detail = edtExpl.getText().toString();
        //更新单词记录
        ContentValues values = new ContentValues();
        values.put(Words.Word.WORD, word);
        values.put(Words.Word.DETAIL, detail);
        contentResolver.update(Words.Word.DICT_CONTENT_URI, values, "word = ?"
                , new String[]{word});
        //设置确认更新单词按钮不可见
        btnUpdateOk.setVisibility(View.INVISIBLE);
        //清空输入框
        edtWord.setText("");
        edtExpl.setText("");
        //显示提示信息
        Toast.makeText(DictActivityTest.this, "更新单词" + word + "成功!", 8000).show();
    }
}

//删除单词事件监听器
private class DeleteWord implements OnClickListener{
    @Override
    public void onClick(View v) {
        //获取用户输入
        String delete_word = edtSearch.getText().toString();
        //清空输入框
        edtSearch.setText("");
        //删除单词记录
        contentResolver.delete(Words.Word.DICT_CONTENT_URI, "word = ?"
                , new String[] {delete_word});
        //显示提示信息
        Toast.makeText(DictActivityTest.this, "删除单词" + delete_word + "成功!"
                , 8000).show();
    }
}
```

```

//查询单词事件监听器
private class SearchWord implements OnClickListener{
    @Override
    public void onClick(View v) {
        //获取用户输入
        String key = edtSearch.getText().toString();
        //清空输入框
        edtSearch.setText("");
        //执行查询
        Cursor cursor = contentResolver.query(Words.Word.DICT_CONTENT_URI, null
                , "word like ? or detail like ?", new String[]{"%" + key + "%", "%" + key
                + "%"}, null);
        //创建一个Bundle对象
        Bundle data = new Bundle();
        data.putSerializable("data", converCursorToList(cursor));
        //创建一个Intent
        Intent intent = new Intent(DictActivityTest.this, ResultActivity.class);
        intent.putExtras(data);
        //启动Activity
        startActivity(intent);
    }
}

//把记录集里的数据转化成ArrayList的形式
private ArrayList<Map<String, String>> converCursorToList(Cursor cursor){
    ArrayList<Map<String, String>> result = new ArrayList<Map<String, String>>();
    // 遍历Cursor结果集
    while (cursor.moveToNext()){
        // 将结果集中的数据存入ArrayList中
        Map<String, String> map = new HashMap<String, String>();
        // 取出查询记录中第2列、第3列的值
        map.put("word", cursor.getString(1));
        map.put("detail", cursor.getString(2));
        result.add(map);
    }
    return result;
}

```

步骤八：

添加一个名字为 ResultActivity 的 Activity，用于处理结果数据。对应的布局文件为 activity_result.xml。

1) 布局文件 activity_result.xml 中组件的主要属性如下表所示。

表 9.2.2 组件列表

组件类型	ID	TEXT	组件类型	ID
TextView	textView1	要查找的单词	ListView	show

其主要代码如下：

```
<TextView  
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginBottom="5dp"  
    android:layout_marginLeft="10dp"  
    android:layout_marginTop="10dp"  
    android:text="查找到的单词"  
    android:textSize="15sp" />  
  
<ListView  
    android:id="@+id/show"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" >  
</ListView>
```

2) 添加一个布局文件，名称为 dict_result.xml，该布局文件用于显示每个词条的详细信息。

该布局文件中组件的主要属性如下表所示。

表 9.2.3 组件列表

组件类型	ID	组件类型	ID
TextView	dic_Word	TextView	dict_Detail

其主要代码如下所示。

```

<!-- ListView的每一项都有两个TextView， 分别显示word和detail -->
<TextView
    android:id="@+id/dic_Word"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:textSize="10pt" />
<TextView
    android:id="@+id/dict_Detail"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:textSize="10pt" />

```

3) 编辑 ResultActivity.java 文件，在重写的 onCreate() 方法中使用 SimpleAdapter 的对象将数据与布局文件中的 ListView 关联，使数据显示在 ListView 中。

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_result);
    ListView listView = (ListView) findViewById(R.id.show);
    Intent intent = getIntent();
    // 获取该intent所携带的数据
    Bundle data = intent.getExtras();
    // 从Bundle数据包中取出数据
    @SuppressWarnings("unchecked")
    List<Map<String, String>> list = (List<Map<String, String>>) data
        .getSerializable("data");
    // 将List封装成SimpleAdapter
    SimpleAdapter adapter = new SimpleAdapter(ResultActivity.this, list,
        R.layout.dict_result, new String[] { "word", "detail" },
        new int[] { R.id.dic_Word, R.id.dict_Detail });
    // 填充ListView
    listView.setAdapter(adapter);
}

```

步骤九：

运行程序，首先向单词本中添加若干个单词。如图 9.2.2 所示。



图 9.2.2 添加单词界面

可以对单词进行查询、删除和修改操作。在“请输入要查找/删除/修改的单词”文本框中输入想要操作的单词。当点击修改单词的时候，该单词及其释义会显示在上方的两个编辑框中，并且会出现“确认修改单词”按钮，当输入修改的单词释义后，点击该按钮，完成对该单词的修改。

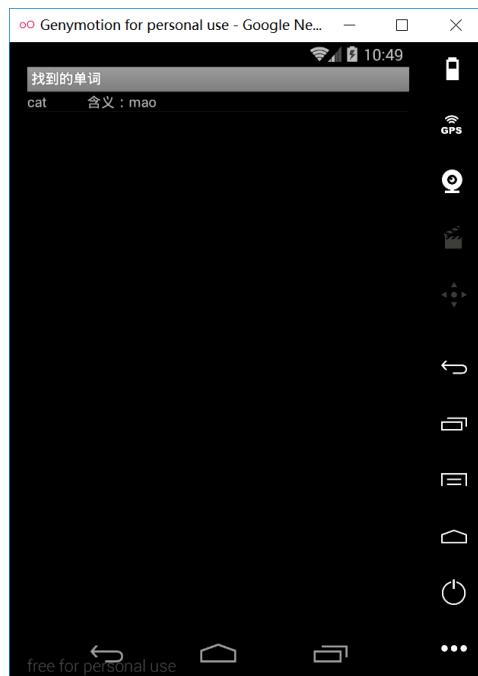


图 9.2.3 查询单个单词结果界面



图 9.2.4 删除单个单词结果界面



图 9.2.5 修改某个单词界面



图 9.2.6 确认修改单词后的结果界面

9.2.4 实验结论:

本次试验我们需要注意的是，对于自定义的 ContentProvider 一定要在 `AndroidManifest.xml` 文件中注册。

测试 ContentProvider 暴漏的数据所使用的应用程序不一定和该 ContentProvider 同在一个应用程序中，大多情况下两者处于不同的应用程序之中。

9.3 作业

9.3.1 完善实验一中的例子，增加删除和更新联系人功能

9.3.2 自定义 ContentProvider，封装学生信息系统中的 SQLite 成为 URI

(学生信息包括：学号、姓名、性别)

第10章 Android 中服务（Service）的使用

本章的主要内容有对服务的简介，Android 中 Service 的创建，Android 中 Service 的调用，Android 中 Service 的生命周期，Android 中常见的系统服务。

通过本章实验我们能够掌握 Android 中 Service 的用法，能够自定义 Service，能够调用系统常见的 Service。

10.1 实验一 Android 中的 Service 的创建

10.1.1 实验目的

1. 掌握 Service 生命周期各回调方法的调用过程。
2. 掌握 Service 的基本用法。

10.1.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

10.1.3 实验步骤

➤ 任务：使用 Service 生成随机数。

为了帮助熟练掌握 Service 的生命周期，本任务实现通过界面上的两个按钮来启动和停止 Service。该 Service 的作用是产生随机数。

Service 的启动有两种方式：

`context.startService()`、`context.bindService()`。

Service 的生命周期如图 1 所示。

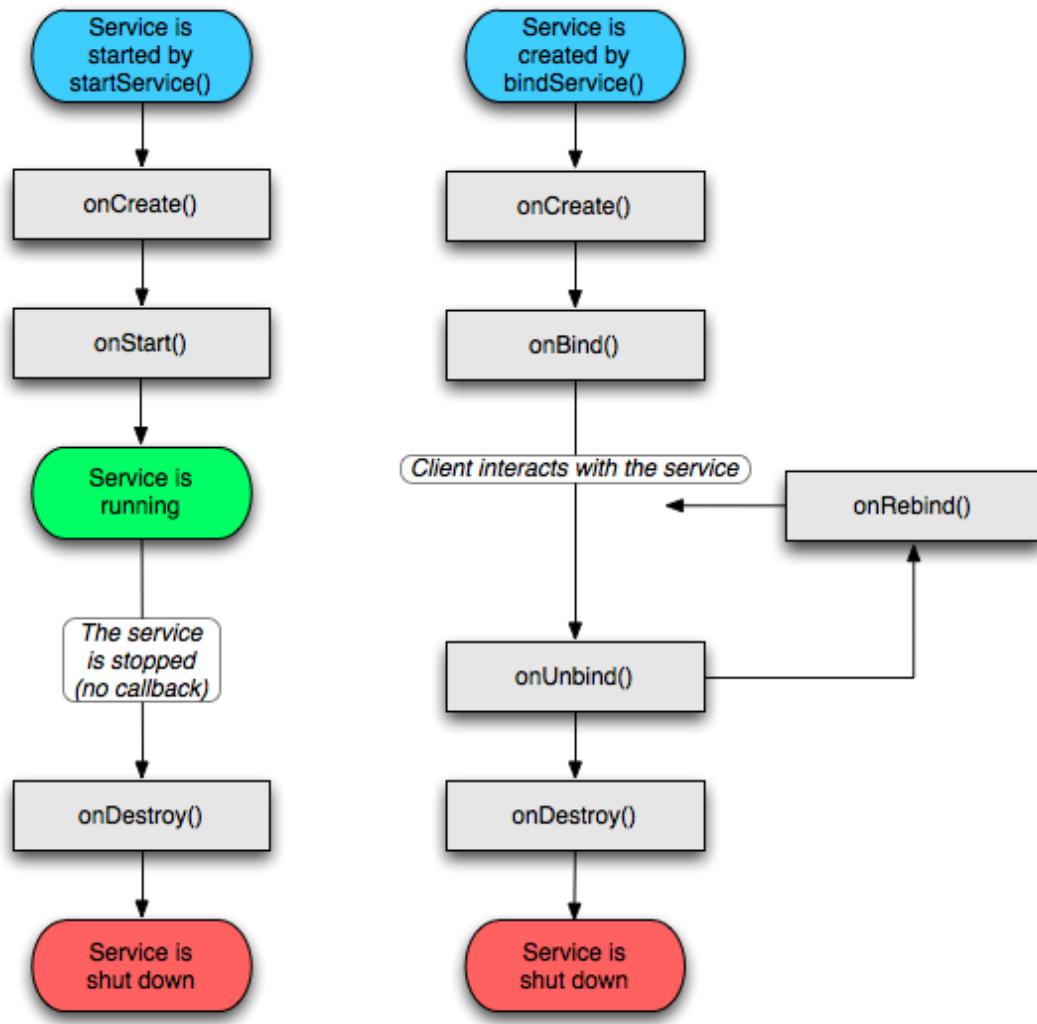


图 10.1.1

着重掌握 Service 和 Activity 生命周期的区别：

Activity: 某程序组件需要在运行时向用户呈现某种界面；

某程序需要在运行时与用户进行交互；

Service: 不需要用户界面或不需要与用户进行交互

步骤一：

新建一个 Android 应用程序 ServiceTest。

步骤二：

新建类，继承 Service，定义 Service 组件

步骤三：

在布局文件放置两个按钮，分别用于启动和停止 Service。该应用程序的目录结构和布局文件界面如图 10.1.2、图 10.1.3 所示。

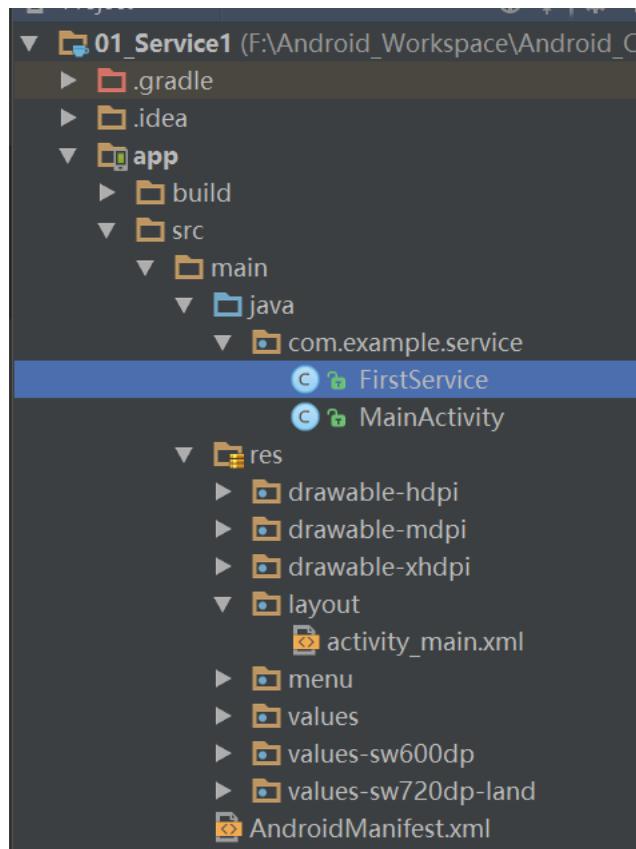


图 10.1.2



图 10.1.3

布局文件两个按钮的属性，如表 10.1.1 所示：

表 10.1.1 布局文件中的组件的属性表

控件类型 属性	id	text
Button	btn_start_service	启动 Service
Button	btn_stop_service	停止 Service

步骤四：

在类文件 FirstService.java 中复写 Service 组件的回调方法。

- 1) 复写 onCreate()

```
//Service被创建时回调该方法
@Override
public void onCreate() {
    super.onCreate();
    Toast.makeText(this, "(1) 调用 onCreate()", Toast.LENGTH_LONG).show();
    System.out.println("Service is Created");
}
```

- 2) 复写 onStartCommand()

```
//Service被启动时回调该方法
@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
    Toast.makeText(this, "(2) 调用 onStartCommand()", Toast.LENGTH_SHORT).show();
    double randomDouble = Math.random();
    String msg = "随机数: " + String.valueOf(randomDouble);
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
    System.out.println("Service is Started");
    return START_STICKY;
}
```

- 3) 复写 onDestroy() 方法

```
//Service被关闭之前回调
@Override
public void onDestroy() {
    super.onDestroy();
    System.out.println("Service is Destroyed");
    Toast.makeText(this, "(3) 调用 onDestroy()", Toast.LENGTH_SHORT).show();
}
```

注：所定义的 Service 组件只是重写了几个回调方法，并没有做具体的操作，这只是 Service 组件的一个框架。如果需要该 Service 组件做某些事情，只需在相应的方法中定义相关业务代码即可。

步骤五：

在 AndroidManifest.xml 文件中注册上述 Service 组件。

```
<!-- 配置一个Service组件 -->
<service android:name=".FirstService" />
```

步骤六：

编辑布局文件（activity_main.xml）所对应的 Activity。

- 1) 在 onCreate() 方法中调用封装的两个方法，作用分别是获取用户界面的两个按钮，给按钮注册事件监听器。

```
private Button start;
private Button stop;
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    findView();
    setListener();
}
```

- 2) 实现两个方法。

```
//获取程序界面中的两个按钮
private void findView() {
    start = (Button) findViewById(R.id.btn_start_service);
    stop = (Button) findViewById(R.id.btn_stop_service);
}
private void setListener() {
    // 分别给两个按钮绑定监听事件
    start.setOnClickListener(myListener);
    stop.setOnClickListener(myListener);
}
private OnClickListener myListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 创建启动Service的Intent
        final Intent intent = new Intent(getApplicationContext(),
FirstService.class);
        switch (v.getId()) {
            case R.id.btn_start_service:
                MainActivity.this.startService(intent);
                break;
            case R.id.btn_stop_service:
                MainActivity.this.stopService(intent);
                break;
            default:
                break;
        }
    }
};
```

注:

final 放在类前面:如果某个类在定义时, 前面有修饰词final, 则该类不能被继承;

final 放在属性前面:属性声明时, 如果前面有修饰词final关键字, 则该属性值不能被更改, 即此时该属性为常量;

final放在方法前面:如果某个方法在定义时, 前面有修饰词final, 该方法可以被调用, 不能重写;

步骤七:

启动模拟器, 运行该应用程序。

1) 当首次启动 Service 时, 控制台输出结果如图 10.1.4 所示, 程序运行界

面如图 10.1.5 所示。

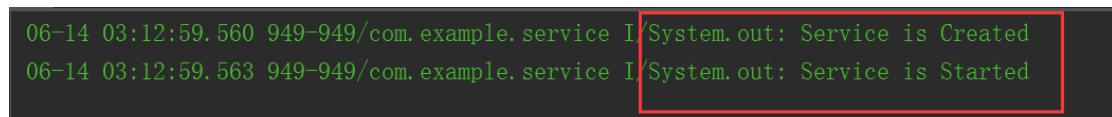


图 10.1.4

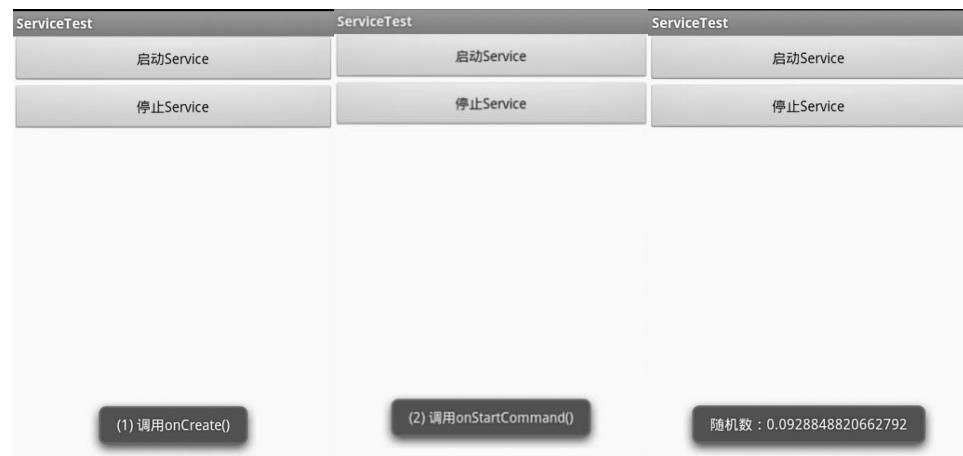


图 10.1.5

点击“Add to contacts”会启动通讯录界面，如图 10.1.4、图 10.1.5 所示。

2) 当再次启动 Service 时，控制台输出和运行界面分别如图 10.1.6、图 10.1.7 所示。

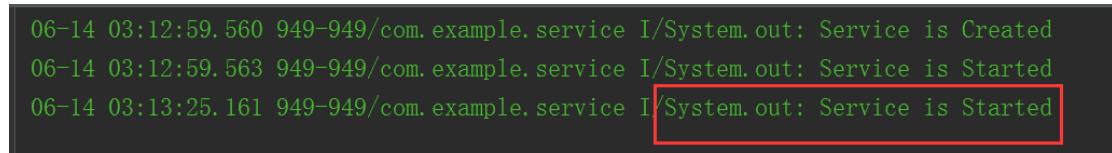


图 10.1.6



图 10.1.7

3) 当停止 Service 时, 控制台输出和界面如图 10.1.8、图 10.1.9 所示。

```
06-14 03:15:30.227 21433-21433/com.example.service I/System.out: Service is Destroyed
```

图 10.1.8

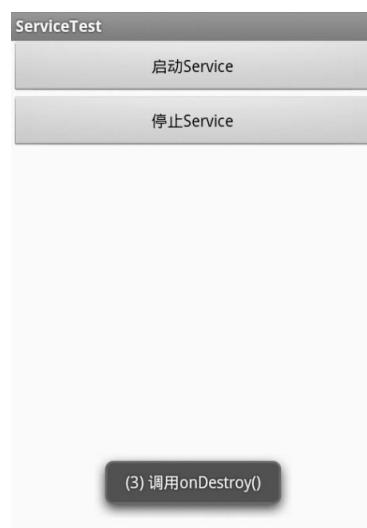


图 10.1.9

4) 再次启动 Service 时, 控制台输出和界面如图 10.1.10、图 10.1.11 所示。

```
06-14 03:16:12.474 21433-21433/com.example.service I/System.out: Service is Created  
06-14 03:16:12.475 21433-21433/com.example.service I/System.out: Service is Started
```

图 10.1.10

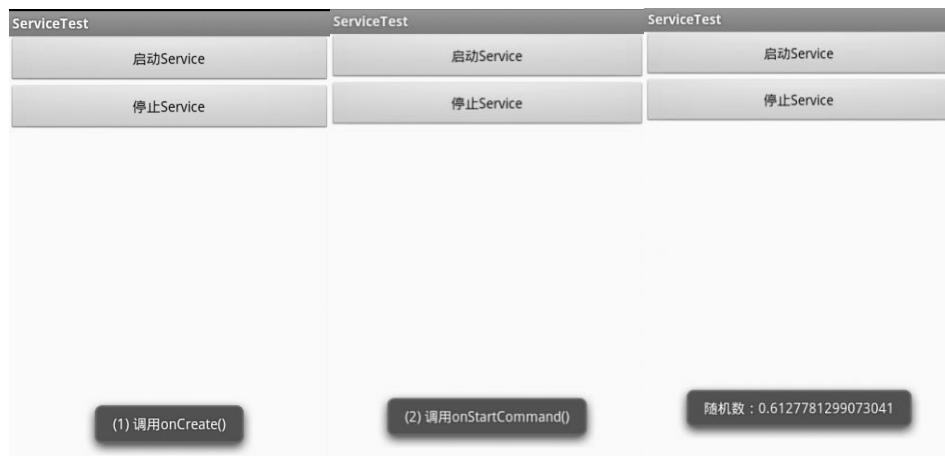


图 10.1.11

5) 实验总结

通过本次实验我们发现启动一个 Service 之后，会执行 onCreate() --> onStartCommand() 方法。一个 Service 可以被多次启动，多次执行 onStartCommand() 方法，该 Service 会被多次被启动，但是 onCreate() 方法只执行一次，即该 Service 只创建一次。当该 Service 被停止后，再次启动时，会再执行 onCreate() —> onStartCommand() 方法。

context.startService() 启动流程：

context.startService() —> onCreate() —> onStart() —> Service running —> context.stopService() —> onDestroy() —> Service stop

context.bindService() 启动流程：

context.bindService() —> onCreate() —> onBind() —> Service running —> onUnbind() —> onDestroy() —> Service stop

10.1.4 实验结论

10.2 实验二 Android 中的 Service 的绑定

10.2.1 实验目的

1. 掌握 Service 生命周期各回调方法的调用过程。
2. 掌握 Service 的基本用法。

10.2.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

10.2.3 实验步骤

➤ 任务：在实验一的基础上完成绑定本地 Service 并与之通信

步骤一：

在布局文件中新增三个按钮，分别用于绑定 Service、解除绑定、获取 Service 状态。该应用程序的布局文件界面如图 10.2.1 所示。



图 10.2.1

布局文件两个按钮的属性，如表 10.2.1 所示：

表 10.2.1 布局文件中的组件的属性表

控件类型	属性	id	text

Button	bind	绑定 Service
Button	unbind	解除绑定 Service
Button	getServiceStatus	获取 Service 状态

步骤二：

新建类文件 BindService. java, 继承 Service, 实现使用线程修改属性的值。
复写其抽象方法。

```
public class BindService extends Service{
    private int count;
    private boolean quit;
    // 定义 onBind 方法所返回的对象
    private MyBinder binder = new MyBinder();
    // 通过继承 Binder 来实现 IBinder 类
    public class MyBinder extends Binder {
        public int getCount() {
            // 获取 Service 的运行状态: count
            return count;
        }
    }
    // 必须实现的方法，绑定该 Service 时回调该方法
    @Override
    public IBinder onBind(Intent intent) {
        System.out.println("Service is Binded");
        // 返回 IBinder 对象
        return binder;
    }
    // Service 被创建时回调该方法
    @Override
    public void onCreate() {
        super.onCreate();
        System.out.println("Service is Created");
        // 启动一条线程，动态地修改 count 状态值
        new Thread() {
            @Override
            public void run() {
                while (!quit) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                    }
                    count++;
                }
            }
        }.start();
    }
}
```

```
// Service 被断开连接时回调该方法
@Override
public boolean onUnbind(Intent intent) {
    System.out.println("Service is Unbinded");
    return true;
}
// Service 被关闭之前回调该方法
@Override
public void onDestroy() {
    super.onDestroy();
    this.quit = true;
    System.out.println("Service is Destroyed");
}
}
```

步骤三：

在 AndroidManifest.xml 文件中注册上述 Service 组件。

```
<!-- 配置一个Service组件 -->
<service android:name=".BindService" />
```

步骤四：

编辑布局文件（activity_main.xml）所对应的 Activity。

- 1) 声明三个按钮对象及 Service 的 IBinder 对象，并定义 ServiceConnection 对象。

```

private Button bind;
private Button unbind;
private Button getServiceStatus;
// 保持所启动的Service的IBinder对象
BindService.MyBinder binder;
// 定义一个ServiceConnection对象
private ServiceConnection conn = new ServiceConnection() {
    // 当该Activity与Service连接成功时回调该方法
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        System.out.println("--Service Connected--");
        // 获取Service的onBind方法所返回的MyBinder对象
        binder = (BindService.MyBinder) service;
    }
    // 当该Activity与Service断开连接时回调该方法
    @Override
    public void onServiceDisconnected(ComponentName name) {
        System.out.println("--Service Disconnected--");
    }
};

```

2) 在 findView 方法中获取新增的按钮控件。

```

private void findView() {
    start = (Button) findViewById(R.id.btn_start_service);
    stop = (Button) findViewById(R.id.btn_stop_service);
    bind = (Button) findViewById(R.id.bind);
    unbind = (Button) findViewById(R.id.unbind);
    getServiceStatus = (Button) findViewById(R.id.getServiceStatus);
}

```

3) 给新增按钮注册事件监听器。

```

private void setListener() {
    start.setOnClickListener(myListener);
    stop.setOnClickListener(myListener);
    bind.setOnClickListener(myListener);
    unbind.setOnClickListener(myListener);
    getServiceStatus.setOnClickListener(myListener);
}

```

4) 实现按钮的监听器事件

```
private OnClickListener myListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 创建启动Service的Intent
        final Intent intent = new Intent(getApplicationContext()
                , FirstService.class);
        final Intent bindIntent = new Intent(
                getApplicationContext(), BindService.class);
        switch (v.getId()) {
            case R.id.btn_start_service:
                MainActivity.this.startService(intent);
                break;
            case R.id.btn_stop_service:
                MainActivity.this.stopService(intent);
                break;
            case R.id.bind:
                // 绑定指定Service
                bindService(bindIntent, conn
                        , Service.BIND_AUTO_CREATE);
                break;
            case R.id.unbind:
                // 解除绑定Service
                unbindService(conn);
                break;
            case R.id.getServiceStatus:
                // 获取、并显示Service的count值
                int count = binder.getCount();
                System.out.println("count = " + count);
                Toast.makeText(MainActivity.this,
                        "Service的count值为: " + count,
                        Toast.LENGTH_SHORT).show();
                break;
            default:
                break;
        }
    }
};
```

步骤五：

启动模拟器，运行该应用程序。点击绑定 Service、获取 Service 状态、解除绑定 Service，控制台输入结果如图 10.2.2 所示。

```
06-14 05:07:04.989 13631-13631/com.example.service I/System.out: Service is Created  
06-14 05:07:04.990 13631-13631/com.example.service I/System.out: Service is Binded  
06-14 05:07:05.008 13631-13631/com.example.service I/System.out: --Service Connected--  
06-14 05:07:07.359 13631-13631/com.example.service I/System.out: count = 2  
06-14 05:07:09.961 13631-13631/com.example.service I/System.out: count = 4  
06-14 05:07:11.797 13631-13631/com.example.service I/System.out: count = 6  
06-14 05:07:16.402 13631-13631/com.example.service I/System.out: count = 11  
06-14 05:07:19.034 13631-13631/com.example.service I/System.out: Service is Unbinded  
06-14 05:07:19.037 13631-13631/com.example.service I/System.out: Service is Destroyed
```

图 10.2.2

10.3 实验三 Android 中的 Service 的创建

10.3.1 实验目的

1. 掌握自定义 Service 和调用系统 Service 的方法。
2. 了解 Service 应用的场景。
3. 了解系统常用服务类。

10.3.2 准备工作

1. Android 环境搭建成功
2. 保证模拟器能够运行正常。

10.3.3 实验步骤

➤ 任务：自定义 Service 播放音乐。

步骤一：

新建一个 Android 工程 MyService，自动创建 Activity，名字默认。

步骤二：

编辑对应布局文件，添加 2 个 Button 按钮，分别用于启动和停止 Service，如图 10.3.1 所示。

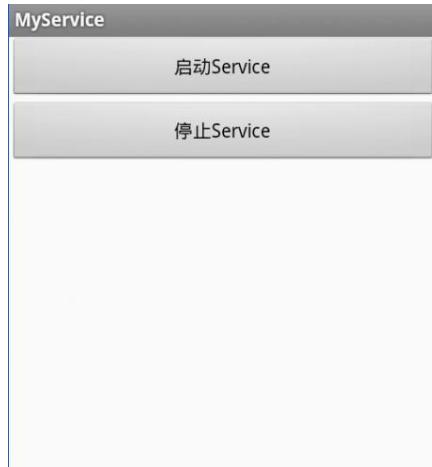


图 10.3.1

布局文件两个按钮的属性，如表 10.3.1 所示：

表 10.3.1 布局文件中的组件的属性表

控件类型 属性	id	text
Button	btn_start	启动 Service
Button	btn_stop	停止 Service

步骤三：

编辑主 Activity 文件 MainActivity.java。

- 1) 继承 Activity 的同时，实现 OnClickListener 接口。声明 2 个 Button 按钮。

```
public class MainActivity extends AppCompatActivity {
    private Button btnStart;
    private Button btnStop;
    ...
}
```

- 2) 在 onCreate() 方法中获取 2 个按钮组件，并且注册事件监听器。

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    btnStart=(Button)findViewById(R.id.btn_start);
    btnStop=(Button)findViewById(R.id. btn_stop);
    //注册事件监听器
    //注册事件监听器
    MyListener listener = new MyListener();
    btnStart.setOnClickListener(listener);
    btnStop.setOnClickListener(listener);
}
class MyListener implements View.OnClickListener{
    @Override
    public void onClick(View v) {
        switch (v.getId()){
            case R.id.btn_start://启动服务
                startService(new Intent(MainActivity.this, MyService.class));
                break;
            case R.id.btn_stop://停止服务
                stopService(new Intent(MainActivity.this, MyService.class));
                break;
        }
    }
}
}

```

步骤四：

在工程的 res 目录下新建目录：右键→ New → Directory

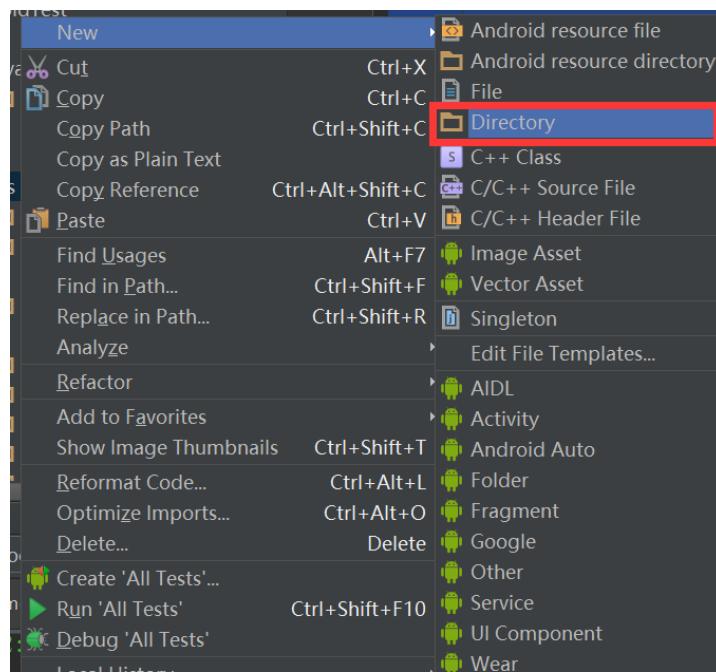


图 10.3.2

输入目录的名称 raw:

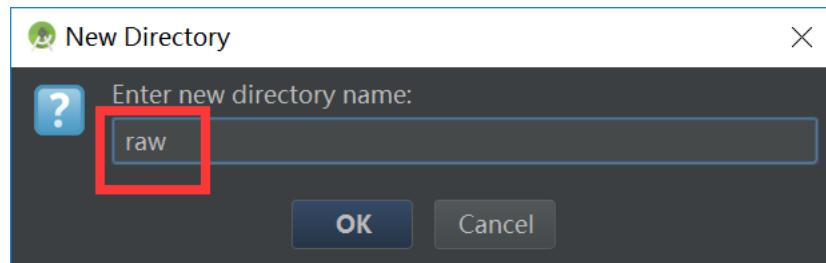


图 10.3.3

在该目录下拷贝一个 MP3 文件

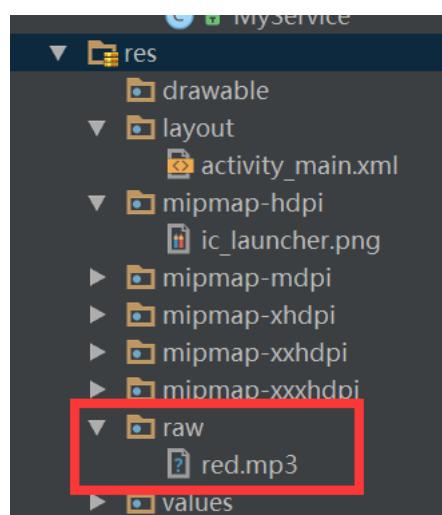


图 10.3.4

步骤五：

自定义服务。新建一个类 MyService.java，继承 Service，并实现 onBind()、
onCreate()、onDestroy()、onStart() 方法。

```
public class MyService extends Service {
    MediaPlayer player;
    public MyService() {
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        player      =      MediaPlayer.create(getApplicationContext(),
R.raw.red);
        player.setLooping(false);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        player.start();
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public void onDestroy() {
        player.stop();
    }
}
```

步骤六：

在 AndroidManifest 中注册自定义的 Service。

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

步骤七：

启动模拟器，运行该应用程序。

10.3.4 实验结论

10.4 实验四 系统常见 Service 调用

10.4.1 实验目的

本次试验我们一起学习调用系统常见的 Service。本实验我们将调用系统电话管理器 (TelephonyManager) 服务类，实现监听手机来电功能。通过本次实验，需要掌握调用系统服务类的方法和步骤，了解系统常用的服务类。

10.4.2 准备工作

1. TelephonyManager 服务类主要用于管理手机通话状态、电话网络信息，并且提供了大量的 getXxx() 方法来获取电话网络的相关信息。

2. 获取系统 TelephonyManager:

```
TelephonyManager tManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE)
```

10.4.3 实验步骤

步骤一：

新建工程 TelephonyManagerTest，其目录结构如下图所示：

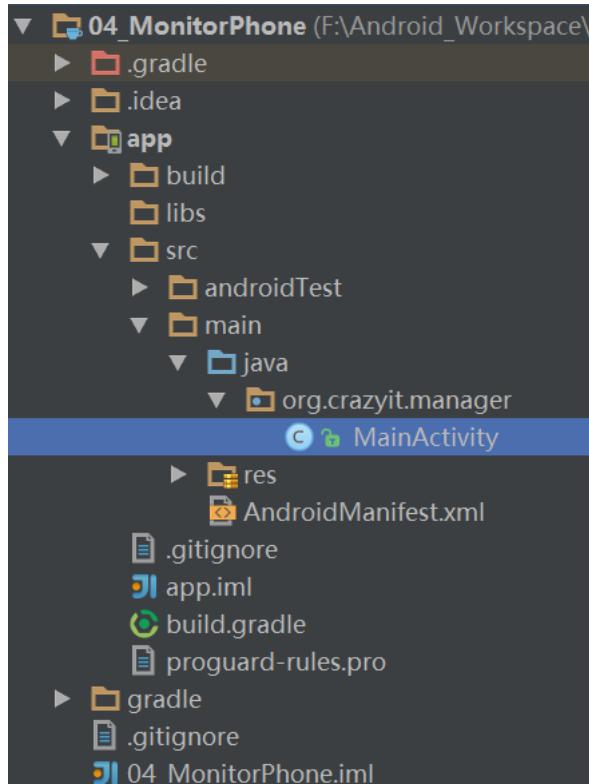


图 10.4.1 目录结构图

步骤二：

在 `MainActivity.java` 文件中调用 `TelephonyManager` 服务类

- 1) 在 `onCreate()` 方法之前定义 `TelephonyManager` 对象

```
private TelephonyManager tManager;
```

- 2) 在 `onCreate()` 方法中取得 `TelephonyManager` 对象

```
tManager = (TelephonyManager) getSystemService  
        (Context. TELEPHONY_SERVICE);
```

- 3) 创建通话状态监听器

```
PhoneStateListener listener = new PhoneStateListener()
{
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        switch(state)
        {
            case TelephonyManager.CALL_STATE_IDLE:
                Toast.makeText(getApplicationContext(),"挂断"
                    ,Toast.LENGTH_SHORT).show();
                break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
                Toast.makeText(getApplicationContext(),"接听"
                    ,Toast.LENGTH_SHORT).show();
                break;
            case TelephonyManager.CALL_STATE_RINGING:
                Toast.makeText(getApplicationContext(),"来电号码为: "
                    +incomingNumber,Toast.LENGTH_SHORT).show();
                break;
        }
        super.onCallStateChanged(state, incomingNumber);
    }
};
```

4) 监听电话通话状态的改变

```
tManager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
```

此时运行该程序会出错

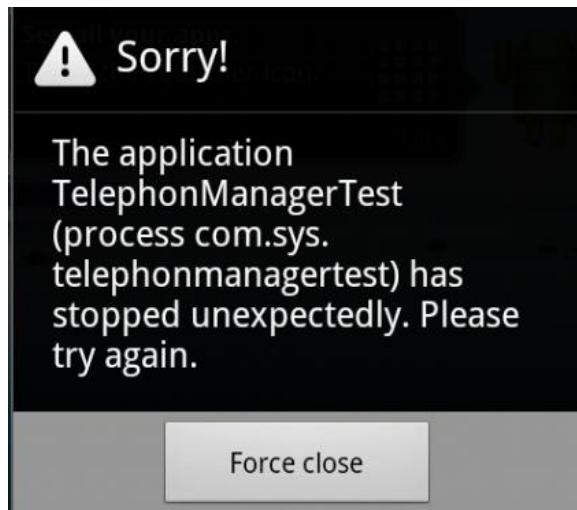


图 10.4.2 出错提示

步骤三：

在 AndroidManifest.xml 文件中添加权限 (<application>标签之前添加授权)

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>  
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

步骤四：

测试（两种方法）

首先运行应用程序

1. 启动 Eclipse 里面的 DDMS, 找到 Emulator control 选项卡, 找到 Telephone Actions, 在 incoming number 里面输入号码。然后选择 voice 单选按钮 (SMS 是模拟发彩信)。点击 Call 就可以成功了, 等几秒钟, 模拟器就会出现来电画面。

注：模拟器不检查号码是否正确，只要是数字的都可以。

或者在 Eclipse 的菜单 Window → Show View → Emulator control

2. 可以通过 DDMS 外壳调用 gsm call 命令直接拨打。我们首先需要启动 AndroidEmulator, 然后在 cmd 环境下执行 telnet localhost 5554, 下面就可以向 Android 模拟器拨号, 参数为 gsm call < phoneNum> , 比如给 10086 打电话 为 gsm call +10086

步骤五：

运行工程效果如下图所示：



图 10.4.3 来电界面



图 10.4.4 挂断界面

10.4.4 实验结论

通过本次试验我们了解了 Service 是四大基本组件之一，没有用户界面，只能在后台运行，并且可以和其他组件进行交互。

系统提供了很多常用的 Service，我们可以调用系统提供的 Service，也可以自定义 Service。如果调用系统的 Service 时必须要在 `AndroidManifest.xml` 文件中授权；如果是自定义的 Service，必须在 `AndroidManifest.xml` 文件中注册该 Service。

10.4.5 扩展知识——Android 的系统服务一览

- System_Server 进程

运行在 `system server` 进程中的服务比较多，这是整个 android 框架的基础

- Native 服务

`SurfaceFlinger`

这是 framebuffer 合成的服务，将各个应用程序及应用程序中的逻辑窗口图像数

据(surface)合成到一个物理窗口中显示(framebuffer)的服务程序

- Java 服务：

这部分的服务大部分都有一个供应用进程使用的 manager 类，这就是一个 RPC 调用，用户通过调用 xxxManager 的方法，实际上被 Binder 给迁移到 system_server 进程中对应的 xxxManagerService 中对应的方法，并将结果再通过 binder 带回。

1. EntropyService

熵服务，周期性的加载和保存随机信息。主要是 linux 开机后，/dev/random 的状态可能是可预知的，这样一些需要随机信息的应用程序就可能会有问题。这个无需提供应用程序接口。

2. PowerManagerService -> PowerManager

Android 的电源管理也是很重要的一部分。比如在待机的时候关掉不用的设备，待机时屏幕和键盘背光的关闭，用户操作的时候该打开多少设备等等。

3. ActivityManagerService->ActivityManager

这个是整个 Android framework 框架中最为核心的一个服务，管理整个框架中任务、进程管理，Intent 解析等的核心实现。虽然名为 Activity 的 Manager Service，但它管辖的范围，不只是 Activity，还有其他三大组件，和它们所在的进程。也就是说用户应用程序的生命管理，都是由他负责的。

4. TelephonyRegistry->TelephonyManager

电话注册、管理服务模块，可以获取电话的链接状态、信号强度等等。<
可以删掉，但要看的大概明白>

5. PackageManagerService -> PackageManager

包括对软件包的解包，验证，安装以及升级等等，对于我们来说不能安装.so 文

件的问题，应该先从这块着手分析原因。

6. AccountManagerService -> AccountManager

A system service that provides account, password, and auth token management for all accounts on the device.

7. ContentService -> ContentResolver

内容服务，主要是数据库等提供解决方法的服务。

8. BatteryService

监控电池充电及状态的服务，当状态改变时，会广播 Intent

9. HardwareService

一般是 ring 和 vibrate 的服务程序

10. SensorService -> SensorManager

管理 Sensor 设备的服务，负责注册 client 设备及当 client 需要使用 sensor 时激活 Sensor

11. WindowManagerService -> WindowManager -> PhoneWindowManager

和 ActivityManagerService 高度粘合

窗口管理，这里最核心的就是输入事件的分发和管理。

12. AlarmManagerService -> AlarmManager

闹钟服务程序

13. BluetoothService -> BluetoothDevice

蓝牙的后台管理和服务程序

14. StatusBarService -> StatusBarManager

负责 statusBar 上图标的更新、动画等等的服务，服务不大。

15. ClipboardService -> ClipboardManager

和其他系统的 clipBoard 服务类似，提供复制黏贴功过。

16. InputMethodManagerService -> InputMethodManager

输入法的管理服务程序，包括何时使能输入法，切换输入法等等。

17. NetStatService

手机网络服务

18. ConnectivityService -> ConnectivityManager

网络连接状态服务，可供其他应用查询，当网络状态变化时，也可广播改变。

19. AccessibilityManagerService-> AccessibilityManager

这块可能要仔细看一下，主要是一些 View 获得点击、焦点、文字改变等事件的分发管理，对整个系统的调试、问题定位等，也需要最这个服务仔细过目一下。

20. NotificationManagerService -> NotificationManager

负责管理和通知后台事件的发生等，这个和 statusbar 胶黏在一起，一般会在 statusbar 上添加响应图标。用户可以通过这知道系统后台发生了什么事情。

21. MountService

磁盘加载服务程序，一般要和一个 linux daemon 程序如 vold/mountd 等合作起作用，主要负责监听并广播 device 的 mount/unmount/bad removal 等等事件。

22. DeviceStorageMonitorService

监控磁盘空间的服务，当磁盘空间不足 10%的时候会给用户警告

23. LocationManagerService -> LocationManager

要加入 GPS 服务等，这部分要细看，现在应用中的 navigation 没响应，可以从此处着手看一下

24. SearchManagerService -> SearchManager

The search manager service handles the search UI, and maintains a registry of searchable activities.

25. Checkin Service(FallbackCheckinService)

貌似 checkin service 是 google 提供的包，没有源代码，源码只有 fallbackCheckinService

26. WallpaperManagerService -> WallpaperManager

管理桌面背景的服务，深度定制化桌面系统，需要看懂并扩展同时要兼容>这部分

27. AudioService -> AudioManager

AudioFlinger 的上层管理封装，主要是音量、音效、声道及铃声等的管理

28. HeadsetObserver

耳机插拔事件的监控小循环

29. DockObserver

如果系统有个座子，当手机装上或拔出这个座子的话，就得靠他来管理了

30. BackupManagerService -> BackupManager

备份服务

31. AppWidgetService -> AppWidgetManager

Android 可以让用户写的程序以 widget 的方式放在桌面上，这就是这套管理和服务的接口

32. StatusBarPolicy

管理哪个图标该在 status bar 上显示的策略。

● mediaServer 服务进程

MediaServer 服务基本上都是 native 的 services，mediaServer 进程也是在 init.rc 中启动的，它不是一个 daemon 进程，这点容易搞混。他也是和 systemserver 进程类似的系统服务进程，提供应用进程的 RPC 调用的真正服务代码所运行的位置。其服务都是和媒体录播放有关，主要有三个服务：

AudioFlinger

声音的录播放服务，包括混音等

MediaPlayerService

提供媒体播放服务，opencore 是这块的核心模块，对 java 端的接口在 mediaplayer.java

CameraService

提供 camera 的录制、preview 等功能的服务

AudioPolicyService

主要功能有检查输入输出设备的连接状态及系统的音频策略的切换等。

10.5 作业

10.5.1 自定义 Service

10.5.2 访问自定义 Service

10.5.3 调用系统短信管理器（SmsManager）服务，实现发送短信功能

第11章 Android 中的广播机制 (Broadcast)

本章的主要内容有对 Android 中广播机制的介绍，学习 Android 中如何使用 Broadcastreceiver 接收广播，学习在 Android 中如何发送广播及 Android 中广播机制的应用。

通过本章实验我们将能够掌握 Android 中的广播机制及 Android 中的 Broadcast 的发送和接受。

11.1 实验一 Android 中的 Broadcast 发送

11.1.1 实验目的

1. 理解 Broadcast 的原理。
2. 了解 Broadcast 的用途。
3. 掌握自定义广播的基本用法。

11.1.2 准备工作

1. Android 环境搭建成功。
2. 保证模拟器运行正常。

11.1.3 实验步骤

➤ 任务：自定义广播接收器，向系统发送一条广播并接收。

发送广播的实现方法比较简单，只需要调用 Context 的 sendBroadcast(Intent intent) 方法即可，这条广播将启动 intent 参数对应的 BroadcastReceiver。

步骤一：

新建工程 SendBroadCastTest，其中 Activity 的名字为 MainActivity. java，对应的布局文件名字为 activity_main. xml。

步骤二：

编辑对应的布局文件，采用线性布局，添加一个按钮组件，用于发送广播。该项目布局如图 11.1.1 所示。



图 11.1.1

布局文件 activity_main. xml 中的按钮属性如下表所示：

表 11.1.1 按钮组件属性

控件	id	text
Button	btn_send	发送广播

步骤三：

自定义自己的广播接收器，新建一个类，名字为 MyReceiver. java，继承 BroadcastReceiver，并实现 onReceive() 方法。该类使用 Toast 显示接收到的广播消息。

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String msgs = intent.getStringExtra("msgs");
        Toast.makeText(context, "接收到的 Intent 的 Action 为: " +
                intent.getAction()
                + "\n消息内容是: " + msgs, 5000).show();
    }
}
```

步骤四：

编辑主 Activity 文件 MessageActivity. java。

首先声明布局文件中的 Button 组件。

```
private Button send;
```

在复写的 onCreate() 方法中获取按钮组件对象并注册事件监听器。

```
//获取程序界面中的按钮
send = (Button) findViewById(R.id.btn_send);
send.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 创建Intent对象
        Intent intent = new Intent();
        //设置Intent的Action属性
        intent.setAction("com.example.sendbroadcasttest.SendBroadcast");
        intent.putExtra("msgs", "简单的消息");
        //发送广播
        sendBroadcast(intent);
    }
});
```

步骤五：

在 manifest 文件中注册自定义的广播接收器。

```
<receiver android:name=".MyReceiver">
<intent-filter>
    <!-- 指定该BroadcastReceiver所响应的Intent的Action -->
    <action
        android:name="com.example.sendbroadcasttest.SendBroadcast"
    />
</intent-filter>
```

步骤六：

运行该 Android 工程，点击按钮，运行结果如图 11.1.2 所示。



图 11.1.2

11.1.4 实验结论

11.2 实验二 Android 中的 Broadcast 接收

11.2.1 实验目的

1. 理解 Broadcast 的原理。
2. 了解 Broadcast 的用途。
3. 掌握 Broadcast 的基本用法。

11.2.2 准备工作

1. Android 环境搭建成功。
2. 保证模拟器运行正常。

11.2.3 准备工作

BroadcastReceive 是 Android 系统四大组件之一, 其本质是一种全局的监听器, 用于监听系统全局的广播消息, 可以方便的实现系统中不同组件之间的通信。

1. BroadcastReceive 启动步骤:

- 1) 创建需要启动 BroadcastReceiver 的 Intent
- 2) 调用 Context 的 sendBroadcast() 或 sendOrderedBroadcast() 方法来启动指定的 BroadcastReceiver。

2. Activity 和 Service 均具有完整的生命周期, BroadcastReceiver 与之不同, 它只是一个系统级的监听器, 专门监听各程序发出的 Broadcast。

3. BroadcastReceiver 与 OnXxxListener 的区别

OnXxxListener 是程序级别的监听器, 运行在指定程序所在的进程之中, 当程序退出时, OnXxxListener 监听器也随之关闭。

BroadcastReceiver 是系统级别的监听器, 拥有自己的进程, 只要存在与之匹配的 Intent 被广播出来, BroadcastReceiver 就会被激发。

4. BroadcastReceiver 的实现方法

重写 onReceive (Context context, Intent intent) 方法

5. 指定 BroadcastReceiver 所能匹配的 Intent 方法 (两种)

➤ 使用代码指定, 调用 BroadcastReceiver 的 Context 的 RegisterReceiver (BroadcastReceiver receiver, IntentFilter filter) 方法。示例代码如下:

```
IntentFilter filter = new IntentFilter
    ( "android.provider.Telephony.SMS_RECEIVER" );
IncomingSMSReceiver receiver = new IncomingSMSReceiver();
registerReceiver(receiver, filter);
```

➤ 在 AndroidManifest.xml 文件中配置。示例代码如下:

```
<receiver android:name=". IncomingSMSReceiver" >
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVER" />
```

```
</intent-filter>  
</receiver>
```

6. BroadcastReceiver 实例的生命周期

当系统 Broadcast 事件发生后，系统就会创建对应的 BroadcastReceiver 的实例，并自动触发它的 `onReceive()` 方法，`onReceive()` 方法执行完后，BroadcastReceiver 的实例就会被销毁。

注意事项：

1. 如果 BroadcastReceiver 的 `onReceive()` 方法不能在 10 秒内执行完成，Android 会认为该程序无响应。所以不要在 `onReceive()` 方法中执行耗时的操作，否则会弹出 ANR (Application no Response) 的对话框。
2. 如果确实需要根据 Broadcast 来完成一项比较耗时的操作，可以考虑通过 Intent 启动一个 Service 来完成该操作。不应考虑使用新线程来完成耗时的操作，因为 Broadcast Receiver 的生命周期很短，可能出现的情况是子线程可能还没结束，BroadcastReceiver 就已经退出了。
3. 如果 BroadcastReceiver 所在的进程结束，虽然该进程中还有用户启动的新线程，但由于该进程中不含任何活动组件，系统可能会在内存紧张时优先结束该进程，这将可能导致 BroadcastReceiver 启动的子线程不能执行完成。

11.2.4 实验步骤

➤ 任务：练习使用广播机制接收系统短信消息

`BroadcastReceive` 是 Android 系统四大组件之一，其本质是一种全局的监听器，用于监听系统全局的广播消息，可以方便的实现系统中不同组件之间的通信。

在 Android 里面有各式各样的广播，比如电话的接收和短信的接收，电池的使用状态都会产生一个广播，Android 开发人员可以监听这些广播并做出程序逻辑的处理。

步骤一：

新建 Android 工程 `SMSRceiverDemo`，自动创建 `Activity`，修改该 `Activity` 的名称为 `MessageActivity.java`。

步骤二：

编辑对应的布局文件，采用线性布局，修改文本框组件的属性，该组件的作用是提示监听器的开启或关闭状态。

添加一个按钮组件，用于开启或关闭广播接收器。该项目的布局如图 11.2.1 所示。



图 11.2.1

该布局文件主要代码如下。

```
<TextView  
    android:id="@+id/tag"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/hello" />  
  
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/turn"  
    android:text="监听器"  
/>
```

步骤三：

自定义广播接收类，用于接收系统接收到的短信消息。新建一个类，继承 BroadcastReceiver，实现 onReceive() 方法。

```
public class SMSReceiver extends BroadcastReceiver {
    public static final String TAG = "ImiChatSMSReceiver";
    private Context context;
    public static final String SMS_RECEIVED_ACTION =
        "android.provider.Telephony.SMS_RECEIVED";
    SMSReceiver(Context context) {
        this.context = context;
    }
    //onCreate() 方法中实现将获取到的短信消息输出并使用Toast的方式提示信息内
    容
    @Override
    public void onReceive(Context context, Intent intent){
        if (intent.getAction().equals(SMS_RECEIVED_ACTION)){
            SmsMessage[] messages = getMessagesFromIntent(intent);
            for (SmsMessage message : messages) {
                Log.i(TAG, message.getOriginatingAddress() + " : " +
                    message.getDisplayOriginatingAddress() + " : " +
                    message.getDisplayMessageBody() + " : " +
                    message.getTimestampMillis());
                Toast.makeText(context,
                    message.getDisplayMessageBody()
                        , Toast.LENGTH_SHORT).show();
            }
        }
    }
    //获取短信消息内容数组
    public final SmsMessage[] getMessagesFromIntent(Intent intent)
    {
        Object[] messages = (Object[])
        intent.getSerializableExtra("pdus");
        byte[][] pduObjs = new byte[messages.length][];
        SmsMessage[] msgs = new SmsMessage[messages.length];
        for (int i = 0; i < messages.length; i++)
        {
            pduObjs[i] = (byte[]) messages[i];
            msgs[i] = SmsMessage.createFromPdu(pduObjs[i]);
        }
    }
}
```

步骤四：

编辑主 Activity 文件 MessageActivity.java。

首先声明布局文件中的组件及接收器类。

```
private SMSReceiver myBroadCastReceiver;
private IntentFilter filter;
private Button turn;
private Boolean flag = false;
private TextView tag;
```

在复写的 onCreate() 方法中设置广播接收器，并获取布局文件中的组件，给按钮注册事件监听器。

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    init(); // 初始化，设置广播接收器
    turn = (Button) findViewById(R.id.turn);
    tag = (TextView) findViewById(R.id.tag);
    turn.setOnClickListener(new OnClickListener() { // 注册事件监听器
        @Override
        public void onClick(View v) {
            if (false == flag) {
                registerReceiver(myBroadCastReceiver, filter);
                tag.setText("监听器开启");
                flag = true;
            } else{
                unregisterReceiver(myBroadCastReceiver);
                tag.setText("监听器关闭");
                flag = false;
            }
        }
    });
}
private void init() {
    myBroadCastReceiver = new SMSReceiver(this);
    // 设置接收的action
    filter = new IntentFilter();
    filter.addAction("android.provider.Telephony.SMS_RECEIVED");
}
```

步骤五：

此工程使用到了短信接收的功能，所以要在 manifest 文件中加上短信接收的权限。

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

步骤六：

运行该 Android 工程，点击监听器按钮，开启或关闭广播接收器。如图 11.2.2、图 11.2.3 所示。



图 11.2.2



图 11.2.3

当收到短信时，使用 Toast 显示短信消息的内容，如图 11.2.4 所示。

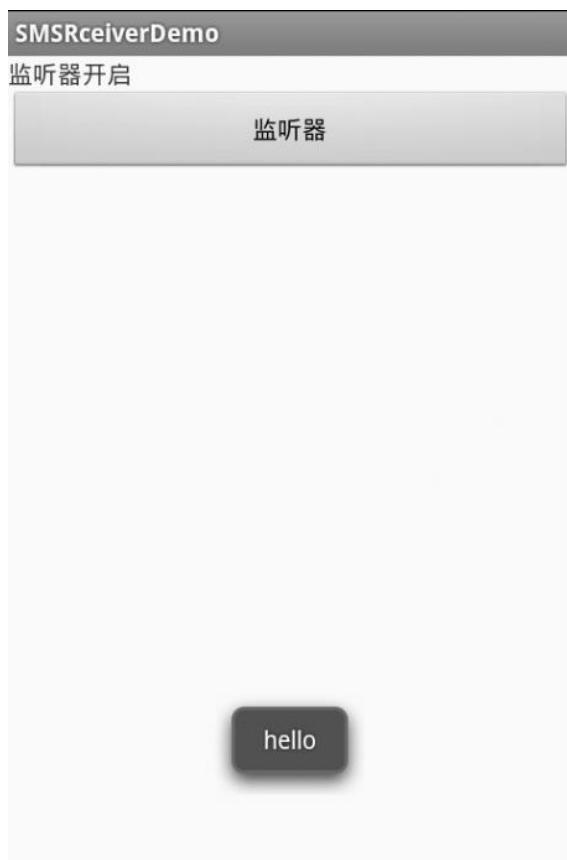


图 11.2.4

注：如果使用模拟器进行测试时，需要在 DDMS 页面，给模拟器发送一条短信，模拟收到短信的状态。

实验总结

1、Activity 和 Service 均具有完整的生命周期，BroadcastReceiver 与之不同，它只是一个系统级的监听器，专门监听各程序发出的 Broadcast。

2、BroadcastReceiver 与 OnXxxListener 的区别

OnXxxListener 是程序级别的监听器，运行在指定程序所在的进程之中，当程序退出时，OnXxxListener 监听器也随之关闭。

BroadcastReceiver 是系统级别的监听器，拥有自己的进程，只要存在与之匹配的 Intent 被广播出来，BroadcastReceiver 就会被激发。

3、指定 BroadcastReceiver 所能匹配的 Intent 方法（两种）

1) 使用代码指定，调用 BroadcastReceiver 的 Context 的 RegisterReceiver (BroadcastReceiver receiver, IntentFilter filter) 方法。示例代码如下：

```
IntentFilter filter = new IntentFilter
        ( “android.provider.Telephony.SMS_RECEIVER” );
IncomingSMSReceiver receiver = new IncomingSMSReceiver();
registerReceiver(receiver, filter);
```

2) 在 AndroidManifest.xml 文件中配置。示例代码如下:

```
<receiver android:name= “. IncomingSMSReceiver” >
    <intent-filter>
        <action android:name= “android:provider.Telephony.SMS_RECEIVER” />
    </intent-filter>
</receiver>
```

4、BroadcastReceiver 实例的生命周期

当系统 Broadcast 事件发生后，系统就会创建对应的 BroadcastReceiver 的实例，并自动触发它的 onReceive() 方法，onReceive() 方法执行完后，BroadcastReceiver 的实例就会被销毁。

注意事项：

1) 如果 BroadcastReceiver 的 onReceive() 方法不能在 10 秒内执行完成，Android 会认为任务该程序无响应。所以不要在 onReceive() 方法中执行耗时的操作，否则会弹出 ANR (Application no Response) 的对话框。

2) 如果确实需要根据 Broadcast 来完成一项比较耗时的操作，可以考虑通过 Intent 启动一个 Service 来完成该操作。不应考虑使用新线程来完成耗时的操作，因为 Broadcast Receiver 的生命周期很短，可能出现的情况是子线程可能还没结束，BroadcastReceiver 就已经退出了。

3) 如果 BroadcastReceiver 所在的进程结束，虽然该进程内还有用户启动的新线程，但由于该进程内不含任何活动组件，系统可能会在内存紧张时优先结束该进程，这将可能导致 BroadcastReceiver 启动的子线程不能执行完成。

5、Android 常见的广播 Action 常量（具体请参考 Android API 文档中关于 Intent 的说明）：

ACTION_TIME_CHANGED: 系统时间被改变

ACTION_DATE_CHANGED: 系统日期被改变

ACTION_TIMEZONE_CHANGED: 系统时区被改变

ACTION_BOOT_COMPLETED: 系统启动完成

ACTION_PACKAGE_ADDED: 系统添加包
ACTION_PACKAGE_CHANGED: 系统的包改变
ACTION_PACKAGE_REMOVED: 系统的包被删除
ACTION_PACKAGE_RESTARTED: 系统的包被重启
ACTION_PACKAGE_DATE_CLEARED: 系统的包数据被清空
ACTION_BATTERY_CHANGED: 电池电量改变
ACTION_BATTERY_LOW: 电池电量低
ACTION_POWER_CONNECTED: 系统连接电源
ACTION_POWER_DISCONNECTED: 系统与电源断开
ACTION_SHUTDOWN: 系统被关闭

11.2.5 实验结论

通过试验我们学会了如何在 Android 中接受系统广播。

知识扩展：

Android 常见的广播 Action 常量（具体请参考 Android API 文档中关于 Intent 的说明）：

ACTION_TIME_CHANGED: 系统时间被改变
ACTION_DATE_CHANGED: 系统日期被改变
ACTION_TIMEZONE_CHANGED: 系统时区被改变
ACTION_BOOT_COMPLETED: 系统启动完成
ACTION_PACKAGE_ADDED: 系统添加包
ACTION_PACKAGE_CHANGED: 系统的包改变
ACTION_PACKAGE_REMOVED: 系统的包被删除
ACTION_PACKAGE_RESTARTED: 系统的包被重启
ACTION_PACKAGE_DATE_CLEARED: 系统的包数据被清空
ACTION_BATTERY_CHANGED: 电池电量改变
ACTION_BATTERY_LOW: 电池电量低
ACTION_POWER_CONNECTED: 系统连接电源
ACTION_POWER_DISCONNECTED: 系统与电源断开

ACTION_SHUTDOWN: 系统被关闭

11.3 实验三 发送自定义广播

11.3.1 实验目的

本次实验，我们将学会在 Android 中发送广播和接收广播的基本方法。

11.3.2 准备工作

发送广播的实现方法比较简单，只需要调用 Context 的 sendBroadcast(Intent intent) 方法即可，这条广播将启动 intent 参数对应的 BroadcastReceiver。

11.3.3 实验步骤

上面我们刚学习了如何接收系统的广播，下面我们一起来学习如何自定义广播，新建工程 SendBroadCastTest，其中 Activity 的名字为 MainActivity. java，对应的布局文件名字为 activity_main. xml，工程文件中还包括一个接收广播消息的类：MyReceiver. java。其布局文件界面如下图所示。



图 11.3.1 布局文件界面图

布局文件 activity_main.xml 中的按钮属性如下表所示：

表 11.3.1 控件属性

控件	id	text
Button	btn_send	发送广播

MainActivity.java 中的主要代码：

```
public class MainActivity extends AppCompatActivity{
    Button send;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //获取程序界面中的按钮
        send = (Button) findViewById(R.id.btn_send);
        send.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建Intent对象
                Intent intent = new Intent();
                //设置Intent的Action属性
                intent.setAction("com.example.sendbroadcasttest
                    .SendBroadcast");
                intent.putExtra("msgs", "简单的消息");
                //发送广播
                sendBroadcast(intent);
            }
        });
    }
}
```

定义自己的广播接收器，用于接收广播的类 MyReceiver.java 中的代码：（该类使用 Toast 显示接收到的广播消息）

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String msgs = intent.getStringExtra("msgs");
        Toast.makeText(context, "接收到的 Intent 的 Action 为：" + intent.getAction()
            + "\n 消息内容是：" + msgs, 5000).show();
    }
}
```

在 manifest 中注册此广播，需要在 manifest 文件中添加的代码如下：

```
<receiver android:name=".MyReceiver">
    <intent-filter>
        <!-- 指定该BroadcastReceiver所响应的Intent的Action -->
        <action
            android:name="com.example.sendbroadcasttest.SendBroadcast"
        />
    </intent-filter>
</receiver>
```

运行工程，单击按钮后的界面效果如下图所示：



图 11.3.2 运行结果图

11.3.4 实验结论

通过以上实验，我们掌握了在 Android 中发送广播和接收广播的基本方法。

11.4 作业

11.4.1 电话黑名单

要求可以把指定的手机号码加入黑名单，屏蔽其来电。

第12章 Android 中的网络应用

本章的主要内容是对 Android 中多媒体应用的介绍，学习 Android 中如何使用 MediaPlayer 来播放音频，学习使用 SoundPool 播放音效，学习使用 MediaRecorder 录制音频，并且学习摄像头的应用。

通过本章实验我们将能够掌握 Android 中的常用多媒体应用。

12.1 实验一 使用 Socket 进行通信

12.1.1 实验目的

本次试验我们将在 Android 中实现使用 MediaPlayer 播放音频。

12.1.2 准备工作

1. TCP 协议控制两个通信实体互相通信的示意图：

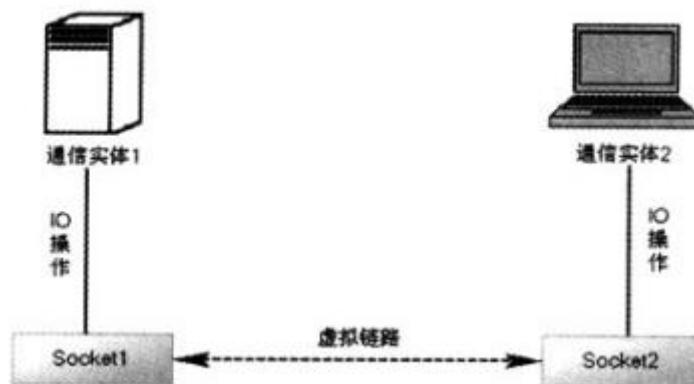


图 12.1.1 TCP 协议的通信示意图

2. Socket 通信的两个通信实体之间需要有服务器端 (ServerSocket) 和客户端 (Socket)。ServerSocket 是一个能够接收其他通信实体连接请求的类，该对象用于监听来自客户端的 Socket 连接，如果没有连接，将一直处于等待状态。

3. ServerSocket 提供了一个用于监听来自客户端连接请求的方法：

Socket accept ()：如果接收到客户端 Socket 的连接请求，该方法返回一个与连接客户端 Socket 对应的 Socket。否则该方法一直处于等待状态，线程也处于阻塞状态。

4. ServerSocket 提供了如下 3 个构造器，用于创建 ServerSocket。

- ServerSocket (int port)：用指定的端口 port 来创建一个 ServerSocket，端口范围：0~65535；
 - ServerSocket (int port, int backlog)：用指定的端口 port 来创建一个 ServerSocket，增加一个用来改变连接队列长度的参数 backlog；
 - ServerSocket (int port, int backlog, InetAddress localAddr)：在机器存在多个 IP 地址的情况下，允许通过 localAddr 这个参数来指定 ServerSocket 绑定到指定的 IP 地址；
5. ServerSocket 使用完毕后，使用 ServerSocket 的 close () 方法来关闭该 ServerSocket。

注：通常不止接收客户端的一个请求，而是不断的接收来自客户端的所有请求，所以 ServerSocket 的 accept () 方法应该被循环不断的调用。

```
//创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
ServerSocket ss = new ServerSocket(30000);

//采用循环不断接收来自客户端请求的方式
while(true)
{
    //每当接收到客户端 Socket 的请求，服务器端就对应产生一个 Socket
    Socket s = ss.accept();
    //接下来就可以使用 Socket 进行通信了
}
```

6. 客户端通过使用 Socket 的如下构造器连接到指定服务器：

- `Socket(InetAddress/String remoteAddress, int port)`: 创建连接到指定远程主机、远程端口的 Socket，该构造器没有指定本地地址、本地端口，使用本地主机的默认 IP 地址（系统动态指定的 IP 地址）；
- `Socket(InetAddress/String remoteAddress, int port, InetAddress localAddr, int localPort)`: 创建连接到指定远程主机、远程端口的 Socket，并指定本地 IP 地址、本地端口号，**适用于本地主机有多个 IP 地址的情形**；

7. Socket 提供的获取输入、输出流的方法

- `InputStream getInputStream()`: 返回该 Socket 对象对应的输入流，程序可通过该输入流从 Socket 中取出数据。
- `OutputStream getOutputStream()`: 返回该 Socket 对象对应的输出流，程序可通过该输出流向 Socket 中输出数据。

下图所示 Socket 通信模型与 Socket 输入/输出流图。

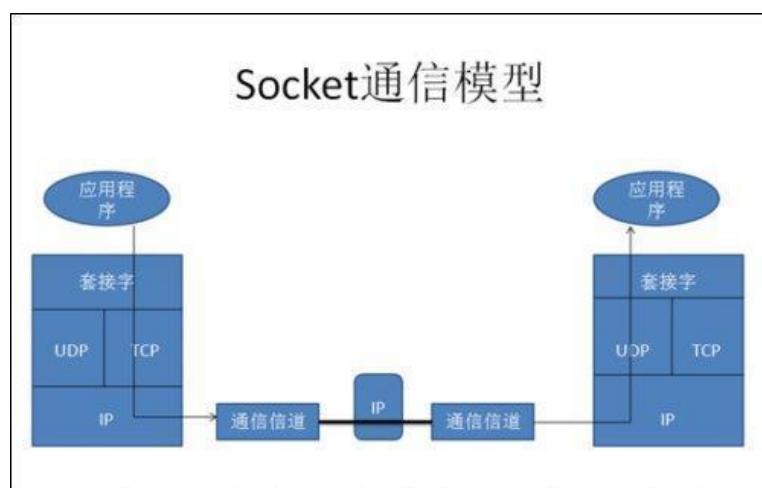


图 12.1.2

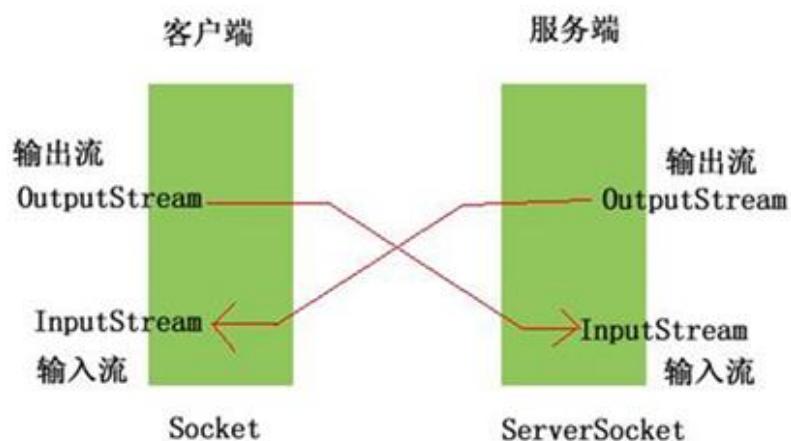


图 12.1.3

12.1.3 实验步骤

步骤一：

建立一个**运行在 PC 端上的 Java 程序**，该类创建 ServerSocket 监听，并使用 Socket 获取输入输出流。

```
public class SimpleServer{  
    public static void main(String[] args) throws IOException{  
        //创建一个ServerSocket，用于监听客户端Socket的连接请求  
        //该ServerSocket在30000端口监听，直到收到客户端程序的连接请求  
        ServerSocket ss = new ServerSocket(30000);  
        //采用循环不断接收来自客户端的请求  
        while(true){  
            //每当接收到客户端Socket的请求，服务器端就对应产生一个Socket  
            Socket s = ss.accept();  
            //接下来就可以使用Socket进行通信了  
            OutputStream os = s.getOutputStream();  
            os.write("服务器的返回信息！\n".getBytes("utf-8"));  
            //关闭输出流，关闭Socket  
            os.close();  
            s.close();  
        }  
    }  
}
```

注：上述程序并未把 OutputStream 流包装成 PrintStream，然后使用 PrintStream 直接输出整个字符串，因为该服务器端程序运行于 Windows 主机上，当直接使用 PrintStream 输出字符串时默认使用系统平台的字符串（即 GBK）进行编码；但该程序的客户端是 Android 应用，运行于 Linux 平台（Android 是 Linux 内核的），因此当客户端读取网络数据时默认使用 UTF-8 字符集进行解码，势必引起乱码。为保证客户端能正常解析到数据，此处手动控制字符串的编码，强行

指定使用 UTF-8 字符集进行编码，可避免乱码问题。

步骤二：

建立客户端程序，使用 Socket 建立与指定 IP、指定端口号的连接，并使用 Socket 获取输入流读取数据。

新建工程 SimpleClientSocketTest，其中 Activity 的名字为 SimpleClientActivity.java，对应的布局文件名字为 activity_simple_client.xml，布局文件中包括 1 个文本框，用于显示从服务器端读取的字符串数据。该工程目录结构及布局文件界面如下图所示

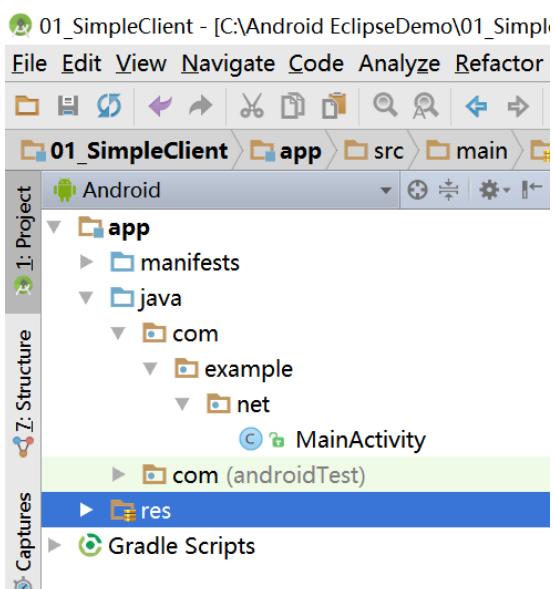


图 12.1.4 工程目录结构图

布局文件中文本框的 id 属性为 edt_show。

SimpleClientActivity.java 代码：

```
public class SimpleClientActivity extends Activity {

    private EditText edt_show;

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

```
edt_show = (EditText) findViewById(R.id.edt_show);

try {
    //构造Socket，与服务器建立连接
    Socket socket = new Socket("10.7.10.44", 30000);
    //将Socket对应的输入流包装成BufferedReader
    BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    //普通IO操作
    String line = br.readLine();
    edt_show.setText("来自服务器的数据: " + line);
    //关闭输入流，关闭Socket
    br.close();
    socket.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
```

步骤三：

该程序需要访问互联网，需要授予访问互联网的权限

```
<!-- 授予访问互联网权限 -->
<uses-permission android:name="android.permission.INTERNET"/>
```

步骤四：

先运行SimpleServerSocket，服务器将一直处于等待状态。

再运行客户端的程序，运行结果如下图所示：

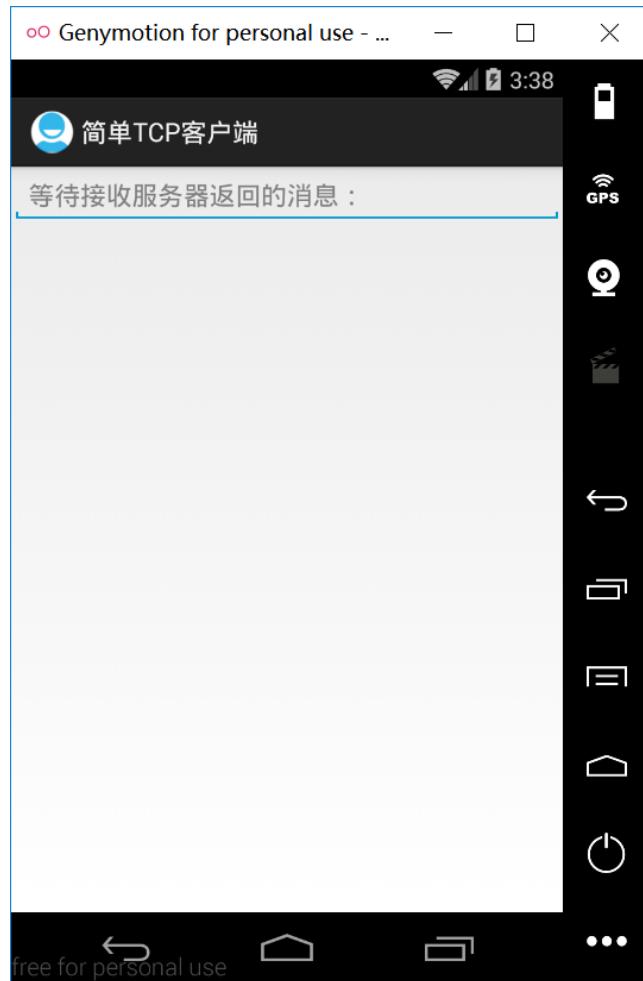


图12.1.5 运行结果图

12.1.4 实验结论

1. 通过试验我们学会了如何在 Android 中使用 Socket 进行简单通讯。
2. 试验中，并没有进行相应的异常处理，也没用使用 finally 关闭资源，在实际应用中，这是不可缺少的。
3. 在实际应用中同样不希望网络连接和数据读写一直处于阻塞状态，而是当操作超过合理时间后，系统自动任务该操作超时失败，这就需要设置超时时长。Socket 对象提供了 setSoTimeout(int timeout) 来设置超时时长。

```
Socket s = new Socket("127.0.0.1", 30000);  
//设置 10 秒后超时  
s.setSoTimeout(10000);
```

4. Socket 的连接没有指定超时时长，Socket 构造器没有指定超时时长的参

数，如果需要设置 Socket 连接超时，通常创建无连接的 Socket，再调用 Socket 的 connect() 方法来连接远程服务器，该方法可接受一个超时时长的参数。

```
//创建一个无连接的 Socket  
Socket s = new Socket();  
//该 Socket 连接远程服务器超过秒未能成功连接时，任务连接超时  
s.connect(new InetSocketAddress(host, port), 10000);
```

12.2 实验二 网络图片下载

12.2.1 实验目的

本实验，我们将学习使用 URL 访问网络资源，掌握文件下载的步骤和流程。

12.2.2 准备工作

1. URL (Uniform Resource Locator) 对象代表同意资源定位器，是指向互联网“资源”的指针，可以使文件或目录，也可以使复杂对象的引用。

URL 可以由协议名、主机、端口、资源组成，其格式如下：

```
protocol://host:port/resourceName  
URL eg: http://www.crazyit.org/index.htm
```

注（了解）：JDK 中提供了一个 URI (Uniform Resource Identifiers) 类，代表同意资源标识符，Java 的 URIIf 不能用于定位任何资源，其唯一作用是解析。与此对应，URL 则包含可打开到达该资源的输入流，因此，可以讲 URL 理解成 URI 的特例。

2. URL 类提供了多个构造器用于创建 URL 对象，获得了 URL 对象后即可调用如下常用方法来访问该 URL 对应的资源。

- String getFile(): 获取此 URL 的资源名;
- String getHost(): 获取此 URL 的主机名;
- String getPath(): 获取此 URL 的路径部分;
- int getPort(): 获取此 URL 的端口号;

- String getProtocol(): 获取此 URL 的协议名称;
- String getQuery(): 获取此 URL 的查询字符串部分;
- URLConnection openConnection(): 返回 URLConnection 对象，表示到 URL 所引用的远程对象的连接;
- InputStream openStream(): 打开与此 URL 的连接，并返回用于读取该 URL 资源的 InputStream。

12.2.3 实验步骤

新建工程 SoundPoolTest，其中 Activity 的名字为 MainActivity.java，对应的布局文件名字为 activity_main.xml，界面中包括 3 个按钮，分别用于播放 3 个不同的声音。在 res 目录下新建 raw 文件，并将 3 个音效资源文件导入。工程的目录结构和布局文件界面如下图所示。

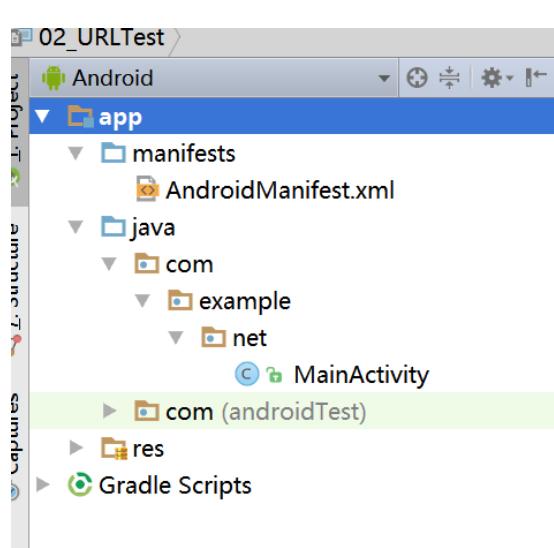


图 12.2.1 工程目录结构图

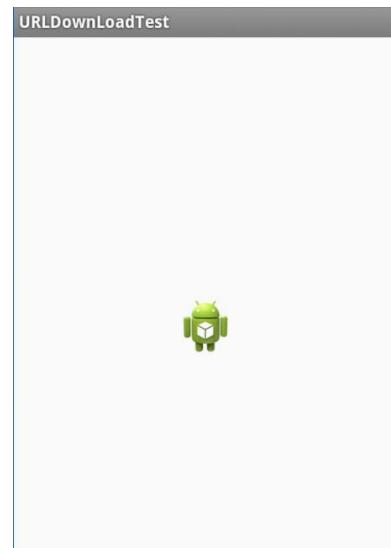


图 12.2.2 布局文件界面图

布局文件 activity_main.xml 中的按钮属性如下表所示：

表 12.2.1 控件属性

控件	id	src
ImageView	iv_show	@drawable/ic_launcher

MainActivity.java 中的主要代码：

```
public class MainActivity extends Activity {  
    private ImageView iv_show;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        iv_show = (ImageView) findViewById(R.id.iv_show);  
        //定义一个URL对象  
        try {  
            URL url = new  
URL("http://pica.nipic.com/2008-07-01/200871134114809_2.jpg");  
            //打开该URL对应资源的输入流  
            InputStream is = url.openStream();  
            //从InputStream中解析出图片  
            Bitmap bitmap = BitmapFactory.decodeStream(is);  
            //使用ImageView显示该图片  
            iv_show.setImageBitmap(bitmap);  
            //关闭输入流  
            is.close();  
            //再次打开URL对应的资源的输入流  
            is = url.openStream();  
            //打开手机文件对应的输出流  
            OutputStream os = openFileOutput("face.jpg", MODE_WORLD_READABLE);  
            byte[] buff = new byte[1024];  
            int hasRead = 0 ;  
            //将URL对应的资源下载到本地  
            while((hasRead = is.read(buff)) > 0) {  
                os.write(buff, 0, hasRead);  
            }  
            os.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
        os.write(buff, 0, hasRead);  
    }  
  
    //关闭输入/输出流  
  
    is.close();  
  
    os.close();  
  
} catch (Exception e) {  
  
    // TODO Auto-generated catch block  
  
    e.printStackTrace();  
  
}  
  
}
```

在 AndroidManifest.xml 文件中授予程序访问网络的权限

```
<!-- 授予访问互联网权限 -->  
<uses-permission android:name="android.permission.INTERNET" />
```

12.2.4 实验结论

希望学生能够通过本实验掌握 url 类的具体使用，并能够合理设计网络访问应用，并能够熟练实现网络下载文件的相关功能。

12.3 实验三 使用 HttpClient 访问网络

12.3.1 实验目的

本次实验，我们将学习使用 HttpClient 访问被保护页面的基本方法。

首先了解 HttpClient 是 Apache 开源组织提供的项目，它是一个简单的 HTTP 客户端（并非浏览器），可用于发送 HTTP 请求，接收 HTTP 响应。但不会缓冲服务器的响应，不执行 HTML 页面中嵌入的 JavaScript 代码，不对页面内容进行解析和处理。

简单来说，HttpClient 是一个增强版 HttpURLConnection，但是 HttpClient

只是关注于如何发送请求、接收响应，以及管理 HTTP 连接。

12.3.2 准备工作

Android 中集成了 HttpClient，我们可以直接在 Android 应用中使用 HttpClient 来访问提交请求、接收响应。

使用 HttpClient 发送请求、接收响应的步骤如下：

1. 创建 HttpClient 对象；
2. 如果需要发送 GET 请求，创建 HttpGet 对象；
如果需要发送 POST 请求，创建 HttpPost 对象。
3. 如果需要发送请求参数，可调用 HttpGet、HttpPost 共同的 setParams (HttpParams params) 方法来添加请求参数；

对于 HttpPost 对象而言，也可以调用 setEntity (HttpEntity entity) 方法来设置请求参数。

4. 调用 HttpClient 对象的 execute (HttpUriRequest request) 发送请求，执行该方法返回一个 HttpResponse。
5. 调用 HttpResponse 的 getAllHeaders ()、getHeaders (String name) 等方法可获取服务器的响应头；

调用 HttpResponse 的 getEntity () 方法可获取 HttpEntity 对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。

12.3.3 实验步骤

新建工程 HttpClientTest，其中 Activity 的名字为 MainActivity.java，对应的布局文件名字为 activity_main.xml。其目录结构和布局文件界面如下图所示。

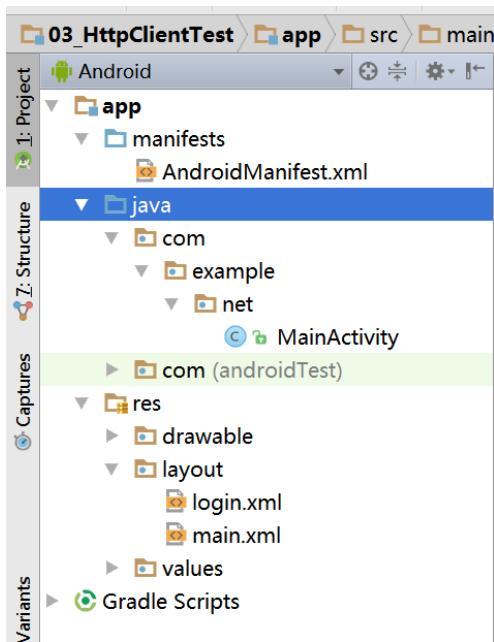


图 12.3.1 工程目录结构图



图 12.3.2 布局文件界面图

布局文件 activity_main.xml 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="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="horizontal">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="accessSecret"
            android:text="@string/get" />
    
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="showLogin"  
    android:text="@string/login" />  
  
</LinearLayout>  
  
<TextView  
    android:id="@+id/response"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:background="#0000"  
    android:gravity="top"  
    android:textColor="@android:color/holo_red_dark"  
    android:textSize="16dp" />  
  
</LinearLayout>
```

MainActivity.java 中的主要代码:

```
public class MainActivity extends Activity {  
  
    TextView response;  
  
    HttpClient httpClient;  
  
    Handler handler = new Handler() {  
  
        public void handleMessage(Message msg) {  
            if (msg.what == 0x123) {  
                // 使用 response 文本框显示服务器响应  
                response.append(msg.obj.toString() + "\n");  
            }  
        }  
    }  
}
```

```
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 创建 DefaultHttpClient 对象
    httpClient = new DefaultHttpClient();
    response = (TextView) findViewById(R.id.response);
}

public void accessSecret(View v) {
    response.setText("");
    new Thread() {
        @Override
        public void run() {
            // 创建一个 HttpGet 对象
            HttpGet get = new HttpGet(
                    "http://www.baidu.com"); // ①
            try {
                // 发送 GET 请求
                HttpResponse httpResponse = httpClient.execute(get); // ②
                HttpEntity entity = httpResponse.getEntity();
                if (entity != null) {
                    // 读取服务器响应
                    BufferedReader br = new BufferedReader(
                            new InputStreamReader(entity.getContent()));
                    String line = null;

```

```
        while ((line = br.readLine()) != null) {
            Message msg = new Message();
            msg.what = 0x123;
            msg.obj = line;
            handler.sendMessage(msg);
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
}.start();
}

public void showLogin(View v) {
    // 加载登录界面
    final View loginDialog = getLayoutInflater().inflate(
        R.layout.login, null);
    // 使用对话框供用户登录系统
    new AlertDialog.Builder(MainActivity.this)
        .setTitle("登录系统")
        .setView(loginDialog)
        .setPositiveButton("登录",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog,
                    int which) {
                    // 获取用户输入的用户名、密码
                    final String name = ((EditText) loginDialog
```

```
        . findViewById(R.id.name)).getText()
        .toString();

    final String pass = ((EditText) loginDialog
        .findViewById(R.id.pass)).getText()
        .toString();

    new Thread() {
        @Override
        public void run() {
            try {
                HttpPost post = new
HttpPost("http://www.baidu.com");//③
                // 如果传递参数个数比较多，可以对传
递的参数进行封装
                List<NameValuePair> params = new
ArrayList<>();
                params.add(new BasicNameValuePair
("name", name));
                params.add(new BasicNameValuePair
("pass", pass));
                // 设置请求参数
                post.setEntity(new
UrlEncodedFormEntity(
                    params, HTTP.UTF_8));
                // 发送 POST 请求
                HttpResponse response = httpClient
                    .execute(post); //④
                // 如果服务器成功地返回响应
                if (response.getStatusLine()
                    .getStatusCode() == 200) {
```

```
String msg = EntityUtils
        .toString(response.get
Entity());
        Looper.prepare();
        // 提示登录成功

Toast.makeText(MainActivity.this,
        msg,
        Toast.LENGTH_SHORT).show();
        Looper.loop();
    }
} catch (Exception e) {
    e.printStackTrace();
}
}.start();
}
}).setNegativeButton("取消", null).show();
}
}
```

在AndroidManifest.xml文件中注册应用网络访问的权限:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

运行效果图:

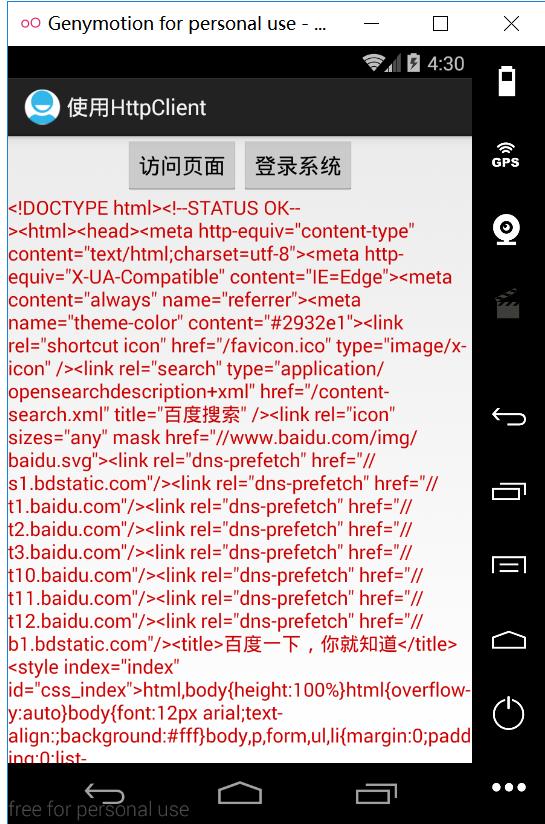


图 12.3.3 运行结果图

12.4 作业

12.4.1 使用 HttpClient 方式访问网络

要求使用 post 和 get 两种方法访问网络数据资源。

第13章 Android 中的数据格式解析

本章的主要内容是对 Android 中多媒体应用的介绍，学习 Android 中如何使用 MediaPlayer 来播放音频，学习使用 SoundPool 播放音效，学习使用 MediaRecorder 录制音频，并且学习摄像头的应用。

通过本章实验我们将能够掌握 Android 中的常用多媒体应用。

13.1 实验一 Android 中 xml 数据解析

13.1.1 实验目的

本次实验需要掌握 Android 中 xml 数据解析

13.1.2 准备工作

1. XML 可扩展标志语言 (Extensible Markup Language)，XML 提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。

Android 中 XML 数据文档的解析有三种方式：DOM，SAX，PULL 。

2. DOM 解析器：

DOM 方式解析 xml 是先把 xml 文档都读到内存中，然后再用 DOM API 来访问树形结构，并获取数据的。

DOM 是基于树形结构的的节点或信息片段的集合，允许开发人员使用 DOM API 遍历 XML 树、检索所需数据。分析该结构通常需要加载整个文档和构造树形结构，然后才可以检索和更新节点信息。

由于 DOM 在内存中以树形结构存放，因此检索和更新效率会更高。但是对于特别大的文档，解析和加载整个文档将会很耗资源。

解析的步骤：

- 1) 利用 DocumentBuilderFactory 创建一个 DocumentBuilderFactory 实例
- 2) 利用 DocumentBuilderFactory 创建 DocumentBuilder

- 3) 加载 XML 文档 (Document)
- 4) 获取文档的根结点 (Element)
- 5) 获取根结点中所有子节点的列表 (NodeList)
- 6) 使用再获取子节点列表中的需要读取的结点。

读取 xml 文档对象，并添加进 List 中的方法：

```
InputStream=this.context.getResources().getAssets().open(fileName)
```

3. SAX 解析器

SAX 不需要整个读入文档就可以对解析出的内容进行处理，是一种逐步解析的方法。程序也可以随时终止解析。这样，一个大的文档就可以逐步的、一点一点的展现出来，所以 SAX 适合于大规模的解析。

SAX (Simple API for XML) 解析器是一种基于事件的解析器，它的核心是事件处理模式，主要是围绕着事件源以及事件处理器来工作的。

SAX 解析 XML 文件采用的是事件驱动，也就是说，它并不需要解析完整个文档，在按内容顺序解析文档的过程中，SAX 会判断当前读到的字符是否合法 XML 语法规中的某部分，如果符合就会触发事件。所谓事件，其实就是一些回调 (callback) 方法，这些方法(事件)定义在 ContentHandler 接口。

当事件源产生事件后，调用事件处理器相应的处理方法，一个事件就可以得到处理。在事件源调用事件处理器中特定方法的时候，还要传递给事件处理器相应事件的状态信息，这样事件处理器才能够根据提供的事件信息来决定自己的行为。

SAX 解析器的优点是解析速度快，占用内存少。

只要为 SAX 提供实现 ContentHandler 接口的类，该类就可以得到通知事件（实际上就是 SAX 调用了该类中的回调方法）。因为 ContentHandler 是一个接口，在使用的时候可能会有些不方便，因此，SAX 还为其制定了一个 Helper 类：DefaultHandler，它实现了这个接口，但是其所有的方法体都为空，在实现的时候，你只需要继承这个类，然后重载相应的方法即可。

下面是一些 ContentHandler 接口常用的方法：

- startDocument() 当遇到文档的开头的时候，调用这个方法，可以在其中做一些预处理的工作。

- `endDocument()` 和上面的方法相对应，当文档结束的时候，调用这个方法，可以在其中做一些善后的工作。
- `startElement(String namespaceURI, String localName, String qName, Attributes attrs)` 当读到一个开始标签的时候，会触发这个方法。
namespaceURI 就是命名空间，localName 是不带命名空间前缀的标签名，qName 是带命名空间前缀的标签名。通过 attrs 可以得到所有的属性名和相应的值。要注意的是 SAX 中一个重要的特点就是它的流式处理，当遇到一个标签的时候，它并不会纪录下以前所碰到的标签，也就是说，在 `startElement()` 方法中，所有你所知道的信息，就是标签的名字和属性，至于标签的嵌套结构，上层标签的名字，是否有子元属等等其它与结构相关的信息，都是不得而知的，都需要你的程序来完成。这使得 SAX 在编程处理上没有 DOM 来得那么方便。
- `endElement(String uri, String localName, String name)` 这个方法和上面的方法相对应，在遇到结束标签的时候，调用这个方法。
- `characters(char[] ch, int start, int length)` 这个方法用来处理在 XML 文件中读到的内容，第一个参数用于存放文件的内容，后面两个参数是读到的字符串在这个数组中的起始位置和长度，使用 `new String(ch, start, length)` 就可以获取内容。

4. PULL 解析器：

PULL 解析器的运行方式和 SAX 类似，都是基于事件的模式。不同的是，在 PULL 解析过程中，我们需要自己获取产生的事件然后做相应的操作，而不像 SAX 那样由处理器触发一种事件的方法，执行我们的代码。PULL 解析器小巧轻便，解析速度快，简单易用，非常适合在 Android 移动设备中使用，Android 系统内部在解析各种 XML 时也是用 PULL 解析器。

Pull 解析器提供了类似的事件，如：开始元素和结束元素事件，使用 `parser.next()` 可以进入下一个元素并触发相应事件。事件将作为数值代码被发送，因此可以使用一个 `switch` 对感兴趣的事件进行处理。当元素开始解析时，调用 `parser.nextText()` 方法可以获取下一个 Text 类型元素的值。

注：参考 <http://www.2cto.com/kf/201202/121173.html>

13.1.3 实验步骤

步骤一：

新建工程 DataParser1_XML，其中 Activity 的名字为 MainActivity.java，对应的布局文件名字为 activity_main.xml，布局文件中包括 1 个文本框，3 个按钮，分别用 3 种方式来解析 XML 文档。该工程目录结构及布局文件界面如下图所示

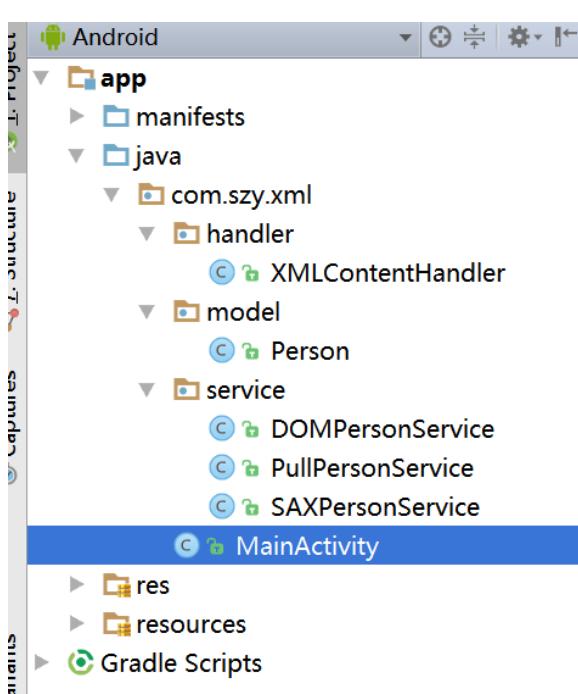


图 13.1.1 工程目录结构图



图 13.1.2 布局文件页面

布局文件中组件属性

表 13.1.1 组件属性表

组件	id	text
TextView	textview	XML文件解析！
Button	btnDom	DOM方式解析xml
Button	btnSax	SAX方式解析xml
Button	btnPull	Pull方式解析xml

待解析 XML 文档 person.xml 代码：

```
<?xml version="1.0" encoding="UTF-8"?>  
<persons xmlns:pre="http://www.baidu.com">
```

```
<pre:person id="001">  
    <name>coolszy</name>  
    <age>22</age>  
</pre:person>  
  
<pre:person id="002">  
    <name>kuka</name>  
    <age>22</age>  
</pre:person>  
</persons>
```

步骤二：

类文件代码：

MainActivity.java 代码：

```
public class MainActivity extends Activity {  
  
    private TextView textView;  
  
    private Button btnDom;  
    private Button btnSax;  
    private Button btnPull;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState)  
    {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        findViewById();  
        setListener();  
    }  
  
    private void setListener() {  
        // TODO Auto-generated method stub
```

```

        btnDom.setOnClickListener(myListener);
        btnSax.setOnClickListener(myListener);
        btnPull.setOnClickListener(myListener);

    }

private void findView() {
    // TODO Auto-generated method stub

    textView = (TextView) findViewById(R. id. textview);
    btnDom = (Button) findViewById(R. id. btnDom);
    btnSax = (Button) findViewById(R. id. btnSax);
    btnPull = (Button) findViewById(R. id. btnPull);
}

private OnClickListener myListener = new OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        InputStream
inStream=MainActivity. class. getClassLoader(). getResourceAsStream("person.xml");
        List<Person> persons = null;
        textView.setText("");
        switch (v. getId()) {
            case R. id. btnDom:
                textView.setText("DOM:");
                try {
                    persons = DOMPersonService. readXml(inStream);
                } catch (Exception e) {

```

```

        // TODO Auto-generated catch block
        e.printStackTrace();
    }

break;

case R.id.btnSax:
    textView.setText("SAX:");
    try {
        persons = SAXPersonService.readXml(inStream);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    break;

case R.id.btnPull:
    textView.setText("PULL:");
    try {
        persons = PullPersonService.readXml(inStream);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    break;

default:
    break;
}

for (Person person : persons)
{
    textView.setText(textView.getText()+"\n"+person.toString());
}

```

```
    }  
};  
}
```

1) DOM方式解析， DOMPersonService.xml代码：

```
public class DOMPersonService  
{  
    public static List<Person> readXml(InputStream inStream) throws Exception  
    {  
        List<Person> persons=new ArrayList<Person>();  
        //实例化一个文档构建器工厂  
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
        //通过文档构建器工厂获取一个文档构建器  
        DocumentBuilder builder = factory.newDocumentBuilder();  
        //通过文档构建器构建一个文档实例  
        Document document = builder.parse(inStream);  
        Element root = document.getDocumentElement();  
        NodeList nodes = root.getElementsByTagName("pre:person");  
        for(int i = 0 ;i < nodes.getLength();i++)  
        {  
            Element personElement = (Element)nodes.item(i);  
            Person person =new Person();  
            //得到xml的属性  
            person.setId(Integer.valueOf(personElement.getAttribute("id")));  
  
            //得到xml的子节点  
            NodeList childNodes = personElement.getChildNodes();  
            for(int j = 0;j<childNodes.getLength();j++)  
            {  
                Node childNode = (Node)childNodes.item(j);
```

```

    if (childNode.getNodeType() == Node.ELEMENT_NODE)

    {

        Element childElement = (Element) childNode;

        if ("name".equals(childElement.getNodeName()))

        {

            person.setName(childElement.getFirstChild().getNodeValue());

        }

        else if (("age".equals(childElement.getNodeName())))

        {

            person.setAge(new

Short(childElement.getFirstChild().getNodeValue()));

        }

    }

    persons.add(person);

}

return persons;

}

}

```

2) SAX方式解析XML文档, SAXPersonService.java代码:

```

public class SAXPersonService{

    public static List<Person> readXml(InputStream inStream) throws Exception

    {

        SAXParserFactory spf=SAXParserFactory.newInstance();

        SAXParser saxParser=spf.newSAXParser(); //创建解析器

        XMLContentHandler handler=new XMLContentHandler();

        saxParser.parse(inStream, handler);

        inStream.close();
    }
}

```

```
        return handler.getPersons();  
    }  
}
```

SAX方式解析XML文档，需要XMLContentHandler类，其代码如下：

```
public class XMLContentHandler extends DefaultHandler  
{  
  
    private static final String TAG = "XMLContentHandler";  
  
    private List<Person> persons;  
  
    private Person person;  
  
    private String preTag;  
  
  
    public List<Person> getPersons()  
    {  
        return persons;  
    }  
  
  
    @Override  
    public void startDocument() throws SAXException  
    {  
        persons = new ArrayList<Person>();  
        Log.i(TAG, "开始解析...");  
    }  
  
  
    /**  
     * uri: 命名空间  
     * localName: 不带命名空间前缀的标签名  
     * qName: 带命名空间前缀的标签名  
     * attributes: 属性集合  
     */
```

```

@Override
public void startElement(String uri, String localName, String qName, Attributes
attributes) throws SAXException
{
    if ("person".equals(localName))
    {
        person = new Person();
        person.setId(new Integer(attributes.getValue("id")));
    }
    preTag = localName;
    Log.i(TAG, "解析元素: "+localName);
}

/**
 * ch[]: 内容
 * start: 其实位置
 * length: 长度
 */
@Override
public void characters(char[] ch, int start, int length) throws SAXException
{
    if (person!=null)
    {
        String data = new String(ch, start, length);
        if ("name".equals(preTag))
        {
            person.setName(data);
        }
        else if ("age".equals(preTag))
    }
}

```

```

    {
        person.setAge(new Short(data));
    }
}

Log.i(TAG, "解析内容: "+new String(ch, start, length));
}

@Override
public void endElement(String uri, String localName, String qName) throws
SAXException
{
    if ("person".equals(localName)&&person!=null)
    {
        persons.add(person);
        person = null;
    }
    preTag = null;
    Log.i(TAG, localName + "解析完毕");
}

@Override
public void endDocument() throws SAXException
{
    super.endDocument();
    Log.i(TAG, "文档解析完毕");
}
}

```

3) 返回结果是一个person类的链表， person的代码如下：

```
public class Person

{
    private Integer id;
    private String name;
    private Short age;

    public Person()

    {
        super();
    }

    public Person(Integer id, String name, Short age)
    {
        super();
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public Integer getId()
    {
        return id;
    }

    public void setId(Integer id)
    {
        this.id = id;
    }
}
```

```
public String getName()
{
    return name;
}

public void setName(String name)
{
    this.name = name;
}

public Short getAge()
{
    return age;
}

public void setAge(Short age)
{
    this.age = age;
}

@Override
public String toString()
{
    return "id=" + getId() + ",name=" + getName() + ",age=" + getAge();
}
```

PULL方式解析XML文件， PullPersonService.xml代码：

```
public class PullPersonService
{
```

```
public static List<Person> readXml(InputStream inStream) throws Exception
{
    List<Person> persons=null;
    XmlPullParser parser=Xml. newPullParser();
    parser.setInput(inStream, "UTF-8");
    int eventCode=parser.getEventType();
    Person person=null;
    while(eventCode!=XmlPullParser.END_DOCUMENT)
    {
        switch (eventCode)
        {
            case XmlPullParser.START_DOCUMENT: //0 文档开始事件
                persons=new ArrayList<Person>();
                break;
            case XmlPullParser.START_TAG: //2 开始元素
                if ("person".equals(parser.getName()))
                {
                    person=new Person();
                    person.setId(new Integer(parser.getAttributeValue(0)));
                }
            else if(null!=person)
            {
                if ("name".equals(parser.getName()))
                {
                    person.setName(parser.nextText());
                }
            else if("age".equals(parser.getName()))
            {
                person.setAge(new Short(parser.nextText()));
            }
        }
    }
}
```

```

        }

    }

    break;

case XmlPullParser.END_TAG: //结束元素

    if ("person".equals(parser.getName())&&person!=null)

    {

        persons.add(person);

        person=null;

    }

    break;

}

eventCode=parser.next();

}

return persons;

}

}

```

步骤三：

运行程序，运行结果如下图所示：



图13.1.3 DOM方式解析结果

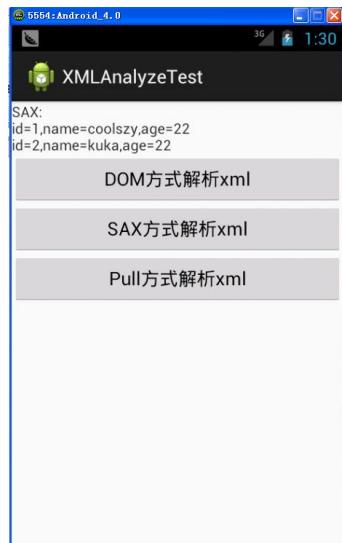


图13.1.4 SAX方式解析结果

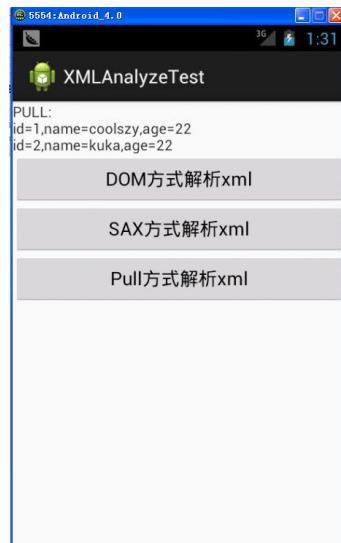


图13.1.5 PULL方式解析结果

13.1.4 实验结论

1. 通过试验我们学会了如何在 Android 中解析 XML 文档。
2. 实验中，待解析的 XML 文档 person.xml 存放于 src 目录下，否则会找不到该 XML 文档。

13.2 实验二 掌握 Android 中 json 数据解析

13.2.1 实验目的

本实验，我们将学习并掌握 Android 中 json 数据解析。

13.2.2 准备工作

JSON 的定义：

一种轻量级的数据交换格式，具有良好的可读和便于快速编写的特性。业内主流技术为其提供了完整的解决方案（有点类似于正则表达式，获得了当今大部分语言的支持），从而可以在不同平台间进行数据交换。JSON 采用兼容性很高的文本格式，同时也具备类似于 C 语言体系的行为。 - Json.org

JSON Vs XML

1. JSON 和 XML 的数据可读性基本相同
2. JSON 和 XML 同样拥有丰富的解析手段
3. JSON 相对于 XML 来讲，数据的体积小
4. JSON 与 JavaScript 的交互更加方便
5. JSON 对数据的描述性比 XML 较差
6. JSON 的速度要远远快于 XML

android2.3 提供的 json 解析类

android 的 json 解析部分都在包 org.json 下，主要有以下几个类：

JSONObject：可以看作是一个 json 对象，这是系统中有关 JSON 定义的基本

单元，其包含一对儿 (Key/Value) 数值。它对外部 (External: 应用 `toString()` 方法输出的数值) 调用的响应体现为一个标准的字符串 (例如: `{"JSON": "Hello, World"}`，最外被大括号包裹，其中的 Key 和 Value 被冒号 ":" 分隔)。其对于内部 (Internal) 行为的操作格式略微，例如：初始化一个 `JSONObject` 实例，引用内部的 `put()` 方法添加数值: `new JSONObject().put("JSON", "Hello, World!")`，在 Key 和 Value 之间是以逗号 "," 分隔。Value 的类型包括: `Boolean`、`JSONArray`、`JSONObject`、`Number`、`String` 或者默认值 `JSONObject.NULL object`。

JSONStringer: json 文本构建类，根据官方的解释，这个类可以帮助快速和便捷的创建 JSON text。其最大的优点在于可以减少由于 格式的错误导致程序异常，引用这个类可以自动严格按照 JSON 语法规则 (syntax rules) 创建 JSON text。每个 `JSONStringer` 实体只能对应创建一个 JSON text。。其最大的优点在于可以减少由于格式的错误导致程序异常，引用这个类可以自动严格按照 JSON 语法规则 (syntax rules) 创建 JSON text。每个 `JSONStringer` 实体只能对应创建一个 JSON text。

JSONArray: 它代表一组有序的数值。将其转换为 String 输出 (`toString()` 所表现的形式是用方括号包裹，数值以逗号 "," 分隔 (例如: `[value1, value2, value3]`)，大家可以亲自利用简短的代码更加直观的了解其格式)。这个类的内部同样具有查询行为，`get()` 和 `opt()` 两种方法都可以通过 `index` 索引返回指定的数值，`put()` 方法用来添加或者替换数值。同样这个类的 `value` 类型可以包括: `Boolean`、`JSONArray`、`JSONObject`、`Number`、`String` 或者默认值 `JSONObject.NULL object`。

JSONTokener: json 解析类

JSONException: json 中用到的异常

13.2.3 实验步骤

新建工程 DataParser2.Json，其中 Activity 的名字为 `MainActivity.java`，对应的布局文件名字为 `activity_main.xml`，界面中包括 5 个按钮，分别进行了不同的 5 种不同的 Json 格式的解析。工程的目录结构和布局文件界面如下图所示。

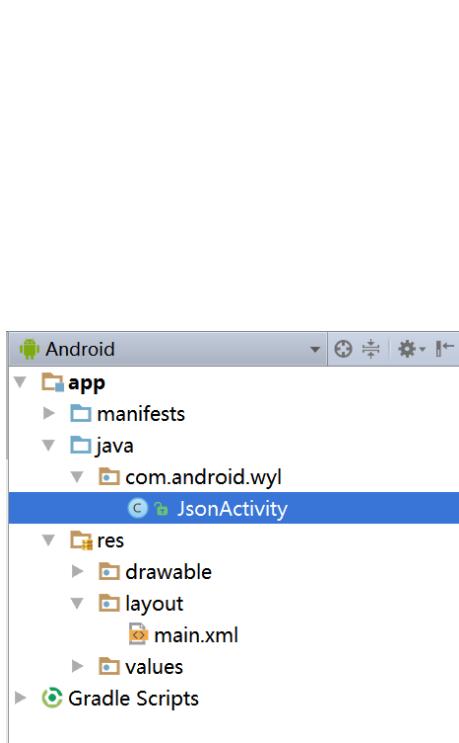


图 13.2.1 工程目录结构图



图 13.2.2 布局文件界面图

布局文件 activity_main.xml 中的按钮属性如下表所示：

表 13.2.1 组件属性

控件	id	text
TextView	tv	Json 数据解析！
Button	btn1	解析一个属性的对象
Button	btn2	解析多个属性的对象
Button	btn3	解析一个普通数组
Button	btn4	解析一个嵌套数组
Button	btn5	解析一个对象数组

MainActivity.java 中的主要代码：

```
public class MainActivity extends Activity {

    private String json1 = "{\"url\":\"http://www.baidu.com\"}";

    private String json2 = "{\"name\":\"android\", \"version\":\"2.3.1\"}";

    private String jsonArray1 = "{\"number\":[1,2,3]}";

    private String jsonArray2 =
```



```

private void findView() {

    // TODO Auto-generated method stub

    btn1 = (Button) findViewById(R. id. btn1) ;

    btn2 = (Button) findViewById(R. id. btn2) ;

    btn3 = (Button) findViewById(R. id. btn3) ;

    btn4 = (Button) findViewById(R. id. btn4) ;

    btn5 = (Button) findViewById(R. id. btn5) ;




    tv = (TextView) findViewById(R. id. tv) ;

}

private OnClickListener myListener = new OnClickListener() {

    @Override

    public void onClick(View v) {

        // TODO Auto-generated method stub

        switch (v. getId()) {

            case R. id. btn1:

                JSONObject demoJson;

                String url;

                try {

                    demoJson = new JSONObject(json1) ;

                    url = demoJson. getString("url") ;

                    tv. setText("url:" + url) ;

                } catch (JSONException e) {

                    // TODO Auto-generated catch block

                    e. printStackTrace() ;

                }

    }
}

```

```

        break;

    case R.id.btn2:
        JSONObject demoJson2;
        try {
            demoJson2 = new JSONObject(json2);
            String name = demoJson2.getString("name");
            String version = demoJson2.getString("version");
            tv.setText("name:" + name + ", version:" + version);
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        break;

    case R.id.btn3:
        JSONObject demoJson3;
        try {
            demoJson3 = new JSONObject(jsonArray1);
            JSONArray numberList = demoJson3.getJSONArray("number");
            tv.setText("");
            for(int i=0; i<numberList.length(); i++) {
                //因为数组中的类型为int, 所以为getInt, //其他getString,
                getLong同用
                tv.setText(tv.getText() + "\n" + numberList.getInt(i));
            }
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

```

```
    }

    break;

case R.id.btn4:
    //嵌套数组遍历
    JSONObject demoJson4;
    try {
        demoJson4 = new JSONObject(jsonArray2);
        JSONArray numberList = demoJson4.getJSONArray("number");
        tv.setText("");
        for(int i=0; i<numberList.length(); i++) {
            //获取数组中的数组
            tv.setText(tv.getText()+"\n"+numberList.getJSONObject(i).getString(0)+";"+numberList.getJSONObject(i).getInt(1));
        }
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    break;

case R.id.btn5:
    JSONObject demoJson5;
    try {
        demoJson5 = new JSONObject(jsonObj);
        JSONArray numberList = demoJson5.getJSONArray("mobile");
        tv.setText("");
        for(int i=0; i<numberList.length(); i++) {
            tv.setText(tv.getText()+"\n"+numberList.getJSONObject(i).getString("name"));
        }
    }
```

```
        }

    } catch (JSONException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

    }

    break;

default:

    break;

}

};

}
```

运行程序，结果如图所示：

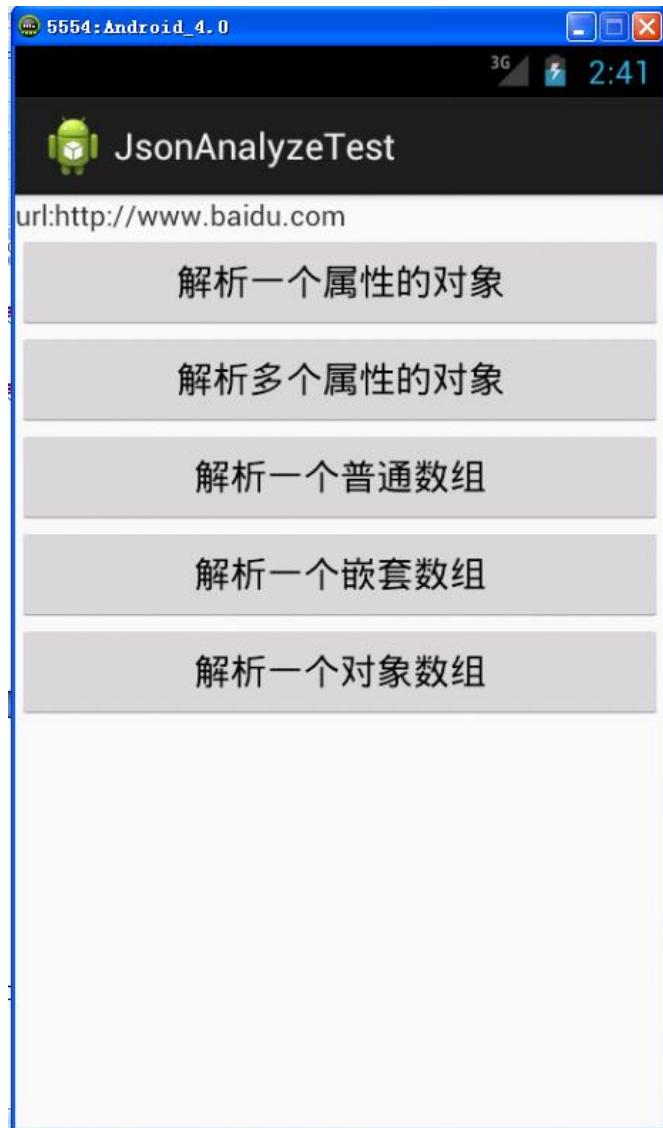


图 13.2.3 解析一个属性对象结果图



图 13.2.4 解析多个属性对象结果图



图 13.2.5 解析一个普通数组结果图

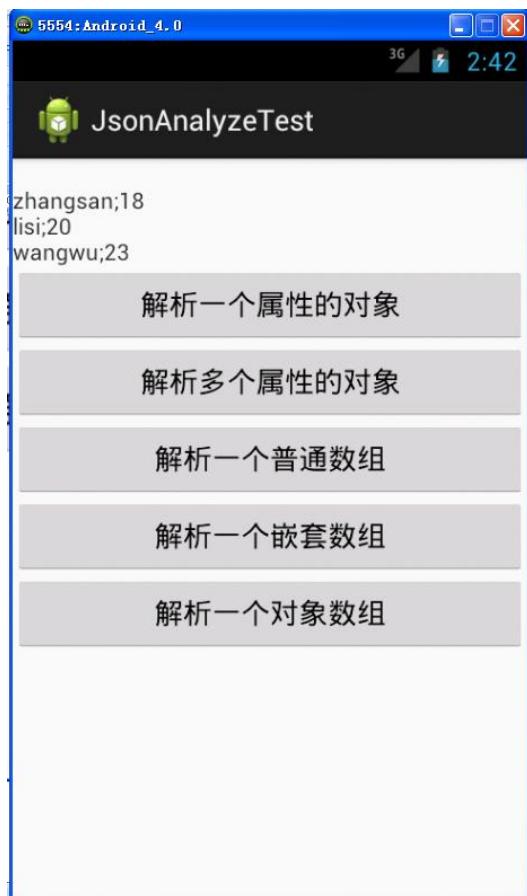


图 13.2.6 解析一个嵌套数组结果图



图 13.2.7 解析一个对象数组结果图

13.3 作业

13.3.1 分析 3 种 XML 文档解析方法的区别与优劣

要求：

总结 3 种解析 XML 数据的区别与优劣。

代码作业：选择一种方式解析 XML 文档。

13.3.2 编写一个 Json 解析的应用

要求：

总结解析 Json 数据的方式。

代码作业：结合 Android 网络应用章节的知识，从网络获取 Json 数据，

并进行 Json 数据格式的解析。

第14章 Android 中的多媒体应用

本章的主要内容是对 Android 中多媒体应用的介绍，学习 Android 中如何使用 MediaPlayer 来播放音频，学习使用 SoundPool 播放音效，学习使用 MediaRecorder 录制音频，并且学习摄像头的应用。

通过本章实验我们将能够掌握 Android 中的常用多媒体应用。

14.1 实验一 使用 MediaPlayer 播放音频

14.1.1 实验目的

本次试验我们将在 Android 中实现使用 MediaPlayer 播放音频。

14.1.2 准备工作

1. MediaPlayer 提供了两个静态方法用于装载指定的音频文件：

- static MediaPlayer creat(Context context, Uri uri)：从指定 uri 来装载将要播放的音频文件，并返回新创建的 MediaPlayer 对象。
- static MediaPlayer creat(Context context, int resid)：从指定的 resid 资源 ID 对应的资源文件中装载音频文件，并返回新创建的 MediaPlayer 对象。

2. 使用 MediaPlayer 播放音频的操作需要调用 MediaPlayer 的三个方法进行播放控制：

- start()：开始或恢复播放
- stop()：停止播放
- pause()：暂停播放

3. 使用 MediaPlayer 的两个静态方法装载音频文件会返回新创建的

MediaPlayer 对象，不适合于循环播放多个音频文件的操作。此时可以通过 MediaPlayer 的 setDataSource() 方法来装载指定的音频文件。

- void setDataSource(String path): 指定装载 path 路径所代表的文件。
- void setDataSource(FileDescriptor fd): 指定装载 fd 所代表的文件。
- void setDataSource(FileDescriptor fd, long offset, long length): 指定装载 fd 所代表的文件中从 offset 开始, 长度为 length 的文件内容。
- void setDataSource(Context context, Uri uri): 指定装载 uri 所代表的文件。

使用已有 MediaPlayer 对象装载“下一首”歌曲的方法:

```
try{  
    mPlayer.reset();  
    //装载下一首歌曲  
    mPlayer.setDataSource(“/mnt/sdcard/next.mp3”);  
    //准备声音  
    mPlayer.prepare();  
    //播放  
    mPlayer.start();  
} catch(IOException e) {  
    e.printStackTrace();  
}
```

注:

1) 使用 void setDataSource(FileDescriptor fd) 方法存在的问题:

不管程序调用 openFd(String name) 方法时指定打开哪个原始资源, MediaPlayer 将总是播放第一个原始的音频资源。

2) 执行上面所示的 setDataSource() 后, MediaPlayer 并未真正装载那些音频文件, 还需要调用 MediaPlayer 的 prepare() 方法去准备音频, 即真正装载执行音频文件。

4. MediaPlayer 提供的绑定事件监听器的方法

- setOnCompletionListener(MediaPlayer.OnCompletionListener

listener)：为 MediaPlayer 的播放完成事件绑定事件监听器。

- setOnErrorListener(MediaPlayer.OnErrorListener listener)：为 MediaPlayer 的播放错误事件绑定事件监听器。
- SetOnPreparedListener(MediaPlayer.OnPreparedListener listener)：当 MediaPlayer 调用 prepare() 方法时触发该监听器。
- setOnSeekCompleteListener(MediaPlayer.OnSeekCompleteListener listener)：当 MediaPlayer 调用 seek() 方法时触发该监听器。

5. 播放资源文件的步骤

- 1) 调用 Context 的 getAssets() 方法获取应用的 AssetManager。
- 2) 调用 AssetManager 对象的 openFd(String name) 方法，打开指定的原生资源，该方法返回一个 AssetFileDescriptor 对象。
- 3) 调用 AssetFileDescriptor 的 getFileDescriptor()、getStartOffset() 和 getLength() 方法来获取音频文件的 FileDescriptor、开始位置、长度等。
- 4) 创建 MediaPlayer 对象，并调用 MediaPlayer 对象的 setDataSource(FileDescriptor fd, long offset, long length) 方法来装载音频资源。
- 5) 调用 MediaPlayer 对象的 prepare() 方法准备音频。
- 6) 调用 MediaPlayer 对象的 start()、pause()、stop() 等方法控制播放。

注：

- ◆ 音频资源文件一般放在 Android 应用的/res/raw 目录下。
- ◆ 实际使用 setDataSource(FileDescriptor fd) 方法存在问题。

14.1.3 实验步骤

新建工程 MediaPlayerTest，布局文件中主要包括 3 个按钮，分别是播放、暂停、停止。准备素材，在 res 下建一个 raw 文件夹，将 love.mp3 拷贝到 raw 文件夹下，。该工程目录结构及布局文件界面如下图所示

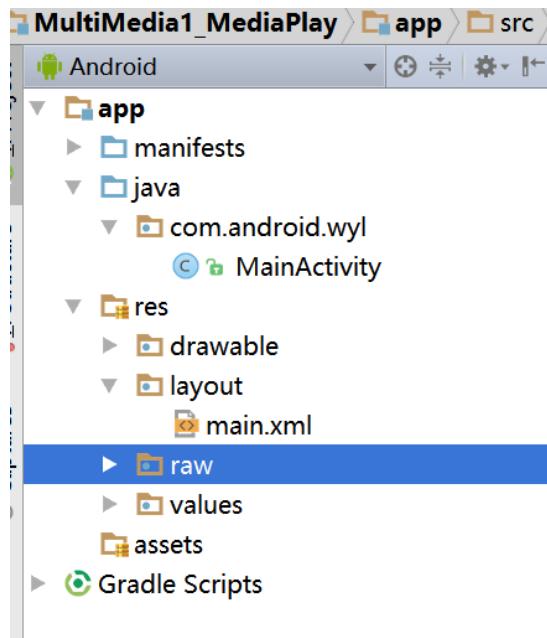


图 14.1.1 工程目录结构图



图 14.1.2 布局文件界面图

布局文件中各控件属性

表 14.1.1 组件属性表

控件	id	text
Button	btn_play	开始
Button	btn_pause	暂停
Button	btn_stop	停止

MainActivity.java代码：

```
public class MainActivity extends Activity {

    private Button BtnStart;
    private Button BtnPause;
    private Button BtnStop;
    private MediaPlayer player;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
player = MediaPlayer.create(getApplicationContext(), R.raw.love);

        findViewById();
        setListener();
    }

private void setListener() {
    // TODO Auto-generated method stub
    BtnStart.setOnClickListener(mylistener);
    BtnPause.setOnClickListener(mylistener);
    BtnStop.setOnClickListener(mylistener);
}

private void findViewById() {
    // TODO Auto-generated method stub
    BtnStart = (Button) findViewById(R.id.BtnStart);
    BtnPause = (Button) findViewById(R.id.BtnPause);
    BtnStop = (Button) findViewById(R.id.BtnStop);
}

OnClickListener mylistener = new OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        switch (v.getId()) {
            case R.id.BtnStart:
                try {
                    player.prepare();

```

```
        } catch (IllegalStateException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
    player.start();  
    break;  
  
    case R.id.BtnPause:  
        player.pause();  
        break;  
  
    case R.id.BtnStop:  
        player.seekTo(0);  
        player.stop();  
        try {  
            player.prepare();  
        } catch (IllegalStateException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        break;  
  
    default:  
        break;  
    }  
}  
};  
}
```

14.1.4 实验结论

通过试验我们学会了如何在 Android 中使用 MediaPlayer 播放音频。所播放的音频文件可以是本地的，或者外部存储器上的，也可以是来自网络的。步骤参考 2.5 只是扩展。

通常需要给 MediaPlayer 设置播放完成事件，在合适的事件调用 release() 方法释放资源。

MediaPlayer 存在缺点：

- ◆ 资源占用量较高、延迟时间较长。
- ◆ 不支持多个音频同时播放。

14.1.5 知识扩展

如果我们想播放手机卡里的音乐，或者 URL 下载流媒体来播放，示意程序如下：

```
MediaPlayer mp = new MediaPlayer();  
  
mp.setDataSource(String URL/FILE_PATH);  
  
mp.prepare();  
  
mp.start();
```

以上程序主要是通过 MediaPlayer.setDataSource() 的方法，将 URL 或文件路径以字符串的方式传入。使用 setDataSource () 方法时，要注意以下三点：

1. 构建完成的 MediaPlayer 必须实现 Null 对像的检查。
2. 必须实现接收 IllegalArgumentException 与 IOException 等异常，在很多情况下，你所用的文件当下并不存在。
3. 若使用 URL 来播放在线媒体文件，该文件应该要能支持 pragressive 下载。

播放外部存储器上音频文件按如下步骤执行。

① 创建 MediaPlayer 对象，并调用 MediaPlayer 对象的 setDataSource(String path)方法装载指定的音频文件。

② 调用 MediaPlayer 对象的 prepare()方法准备音频。

③ 调用 MediaPlayer 的 start()、 pause()、 stop()等方法控制播放即可。

例如如下代码：

```
MediaPlayer mPlayer = new MediaPlayer();
//使用 MediaPlayer 加载指定的声音文件
mPlayer.setDataSource("/mnt/sdcard/mysong.mp3");
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

播放来自网络的音频文件有两种方式：① 直接使用 MediaPlayer 的静态 create(Context context, Uri uri)方法；② 调用 MediaPlayer 的 setDataSource(Context context, Uri uri)方法装载指定 Uri 对应的音频文件。

以第二种方式播放来自网络的音频文件的步骤如下。

① 根据网络上的音频文件所在的位置创建 Uri 对象。

② 创建 MediaPlayer 对象，并调用 MediaPlayer 对象的 setDataSource(Context context, Uri uri)方法装载 Uri 对应的音频文件。

③ 调用 MediaPlayer 对象的 prepare()方法准备音频。

④ 调用 MediaPlayer 的 start()、 pause()、 stop()等方法控制播放即可。

例如如下代码片段：

```
Uri uri = Uri.parse("http://www.crazyit.org/abc.mp3");
MediaPlayer mPlayer = new MediaPlayer();
//使用 MediaPlayer 根据 Uri 来加载指定的声音文件
mPlayer.setDataSource(this, uri);
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

14.2 实验二 使用 SoundPool 播放音效

14.2.1 实验目的

SoundPool 比较适用于播放密集、短促的音效，这些音效的播放不适合采用 MediaPlayer。SoundPool 可以使用音效池来管理多个短促的音效，与 MediaPlayer 相比具有独特的优势特点：CPU 资源占用量低、反应延迟小，并且支持自行设置声音的品质、音量、播放比率等参数。

本次实验，我们将学会在 Android 中使用 SoundPool 播放音效的基本方法。

14.2.2 准备工作

1. SoundPool 构造器

`SoundPool(int maxStreams, int streamType, int SRCQuality)`: 该构造器的第 1 个参数设置可支持多少个声音, 第 2 个参数设置声音的类型, 第 3 个参数设置声音品质。

这将得到一个 SoundPool 对象, 接下来通过调用不同的重载方法 `load()` 来加载声音。

2. SoundPool 提供了 4 个重载的 `load()` 方法:

- ◆ `int load(Context context, int resId, int priority)`: 从 resId 所对应的资源加载声音。
- ◆ `int load(FileDescriptor fd, long offset, long length, int priority)`: 加载 fd 对应的文件从 offset 开始, 长度为 length 的声音。
- ◆ `int load(AssetFileDescriptor afd, int priority)`: 从 afd 所对应的文件中加载声音。
- ◆ `int load(String path, int priority)`: 从 path 对应的文件中加载声音。

注: `priority` 参数目前没有任何作用, 建议固定设置为 1.

以上 4 个方法得到声音的 ID, 可通过该 ID 来播放声音。

3. SoundPool 提供的播放声音的方法:

`int play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)`: 该方法第 1 个参数指定将要播放哪个声音; 第 2、3 个参数指定左、右音量; 第 4 个参数指定播放声音的优先级, 数值越大, 优先级越高; 第 5 个参数指定是否循环, **0: 不循环, -1: 循环**; 第 6 个参数指定播放的比率, 数值从 0.5 到 2, 1 为正常比率。

注: 为更好的管理 SoundPool 所加载的每个声音 ID, 通常会使用 `HashMap<Integer integer>` 来管理声音。

14.2.3 实验步骤

SoundPool 播放声音的步骤：

1. 调用 SoundPool 的构造器，创建 SoundPool 对象；
2. 调用 SoundPool 对象的 load() 方法，从指定资源文件中加载声音；（推荐使用 HashMap<Integer integer> 来管理所加载的声音）
3. 调用 SoundPool 的 play() 方法播放声音。

上面我们刚学习了如何使用 MediaPlayer 播放音频，下面我们一起来学习如何使用 SoundPool 播放音效。新建工程 SoundPoolTest，其中 Activity 的名字为 MainActivity.java，对应的布局文件名字为 activity_main.xml，界面中包括 2 个按钮，分别用于播放 2 个不同的声音。在 res 目录下新建 raw 文件，并将 3 个音效资源文件导入。工程的目录结构和布局文件界面如下图所示。

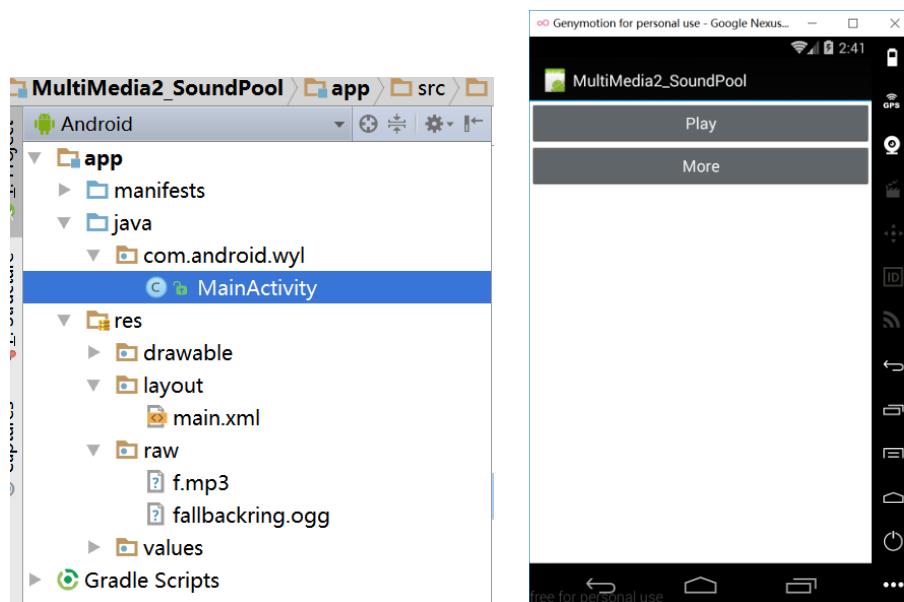


图 14.2.1 工程目录结构图

图 14.2.2 布局文件界面图

布局文件 activity_main.xml 中的按钮属性如下表所示：

表 14.2.1 组件属性

控件	id	text
Button	BtnPlay	Play
Button	BtnMore	More

MainActivity.java 中的主要代码：

```
public class MainActivity extends Activity implements View.OnClickListener,
    SoundPool.OnLoadCompleteListener {

    static String TAG = "TestSoundPoolActivity";
    SoundPool sndPool;

    private static final int SOUND_LOAD_OK = 1;
    private final Handler mHandler = new MyHandler();

    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        setView();

        sndPool = new SoundPool(16, AudioManager.STREAM_MUSIC, 0);
        sndPool.setOnLoadCompleteListener(this);
    }

    private void setView() {
        // TODO Auto-generated method stub
        Button b1 = (Button) findViewById(R.id.BtnPlay);
        b1.setOnClickListener(this);

        b1 = (Button) findViewById(R.id.BtnMore);
    }
}
```

```

        b1.setOnClickListener(this);

    }

    public void onDestroy() {
        sndPool.release();
        super.onDestroy();
    }

    private class MyHandler extends Handler {
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case SOUND_LOAD_OK:
                    sndPool.play(msg.arg1, (float) 0.8, (float) 0.8, 16, 10,
                    (float) 1.0);
                    break;
            }
        }
    }

    public void onLoadComplete(SoundPool soundPool, int sampleId, int status) {
        Message msg = mHandler.obtainMessage(SOUND_LOAD_OK);
        ;
        msg.arg1 = sampleId;
        mHandler.sendMessage(msg);
    }

    public void onClick(View v) {
        int id = v.getId();
        switch (id) {

```

```
        case R.id.BtnPlay:  
            Log.v(TAG, "play main");  
            if (sndPool != null)  
                sndPool.load(this, R.raw.f, 1);  
            break;  
  
        case R.id.BtnMore:  
            sndPool.load(this, R.raw.f, 1);  
            Log.v(TAG, "more");  
            break;  
    }  
}  
}
```

14.2.4 实验结论

通过以上实验，我们掌握了在 Android 中使用 SoundPool 播放音效的基本方法。在试验中，我们发现 2 个音效可以同时播放，这也是 MediaPlayer 所做不到。

在实际中使用 SoundPool 需要注意的问题：

1. SoundPool 可以一次性加载多个音乐，但是由于内存限制，应避免使用 SoundPool 来播放歌曲或者做游戏背景音乐，只有那些短促、密集的声音才考虑使用 SoundPool 进行播放。
2. SoundPool 的效率比 MediaPlayer 的效率好，但是仍然存在延迟的问题，尤其是性能不太好的手机。

14.3 实验三 使用 MediaRecorder 录制音频

14.3.1 实验目的

手机提供了麦克风的功能，Android 系统便可以利用系统提供的麦克风硬件设备录制音频。

本次实验，我们将学习使用 MediaRecorder 录制音频的基本方法。

14.3.2 准备工作

Android 中录制音频需要使用系统提供的 MediaRecorder 类，使用该类实现录制音频的步骤如下：

6. 创建 MediaRecorder 对象；
7. 调用 MediaRecorder 对象的 setAudioSource() 方法设置声音来源，一般传入 MediaRecorder.AudioSource.MIC 参数指定录制来自麦克风的声音。
8. 调用 MediaRecorder 对象的 setOutputFormat() 设置录制的音频文件的格式。
9. 调用 MediaRecorder 对象的 setAudioEncoder()、setAudioEncodingBitRate(int bitRate)、setAudioSamplingRate(int samplingRate) 设置所录制声音的编码格式、编码位率、采样率等。
10. 调用 MediaRecorder 对象的 setOutputFile(String path) 方法设置录制音频的存储路径。
11. 调用 MediaRecorder 对象的 prepare() 方法准备录制。
12. 调用 MediaRecorder 对象的 start() 方法开始录制。
13. 录制完成，调用 MediaRecorder 对象的 stop() 方法停止录制，并调用 release() 方法释放资源。

注： 上面所述步骤中，切记步骤 3 和步骤 4 不能颠倒，否则会抛出 IllegalStateException 异常。

14.3.3 实验步骤

上面我们刚学习了如何使用 MediaPlayer 播放声音，使用 SoundPool 播放音效，下面我们一起来学习如何使用 MediaRecorder 录制音频。新建工程 MediaRecorderTest，其中 Activity 的名字为 MainActivity.java，对应的布局文件名字为 activity_main.xml，界面中包括 2 个按钮，分别用于控制录音开始、录音停止。其目录结构和布局文件界面如下图所示。

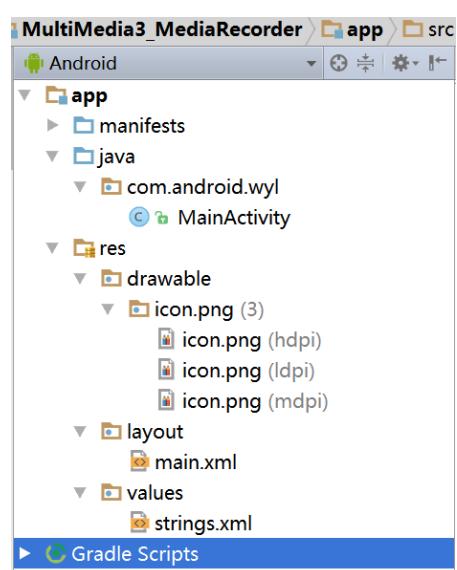


图 14.3.1 工程目录结构图



图 14.3.2 布局文件界面图

布局文件 activity_main.xml 中的按钮属性如下表所示：

表 14.3.1 组件属性

控件	id	text
Button	btn_start	开始录制
Button	btn_stop	停止录制

MainActivity.java 中的主要代码：

```

public class MainActivity extends Activity {

    Button start, stop;

    MediaRecorder mr;

    /** Called when the activity is first created. */

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        start=(Button)findViewById(R.id.start);

        stop=(Button)findViewById(R.id.stop);

        start.setOnClickListener(new OnClickListener() {
            @Override

```

```
public void onClick(View v) {  
    // TODO Auto-generated method stub  
  
    mr=new MediaRecorder();  
  
    mr.setAudioSource(AudioSource.MIC);  
  
    //设置音源,这里是来自麦克风  
  
    mr.setOutputFormat(OutputFormat.RAW_AMR);  
  
    //输出格式  
  
    mr.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);  
  
    //编码  
  
    mr.setOutputFile("/sdcard/sound.amr");  
  
    //输出文件路径  
  
    try{  
        mr.prepare();  
  
        //做些准备工作  
  
        mr.start();  
  
        //开始  
    }catch(Exception e){  
        e.printStackTrace();  
    }  
}  
  
stop.setOnClickListener(new OnClickListener(){  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
  
        mr.stop();  
  
        //停止  
  
        mr.release();  
  
        //释放  
    }  
});
```

```
    }});
}

}
```

在AndroidManifest.xml文件中授予录制声音的权限：

```
<!-- 授予录制声音的权限 -->

<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

14.3.4 实验结论

1. 由于本实验使用了麦克风来录制音频，因此需要在 AndroidManifest.xml 文件中授予录制声音的权限：

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

2. 录音完成后生成的 sound.amr 文件存储在/mnt/sdcard/目录下。
3. 完成音频的录制后要释放资源。

14.4 实验四 实现摄像头拍照功能

14.4.1 实验目的

现在的手机一般都提供相机功能，Android 应用可以控制手机上的相机拍照或者录制视频，本实验将学习使用摄像头的拍照功能。

14.4.2 准备工作

程序开发前提：

1. Android 环境搭建成功。
2. 保证模拟器运行正常。
3. 建议使用真机测试。

使用 Camera 拍照的步骤：

1. 调用 Camera 的 open() 方法打开相机;
2. 调用 Camera 的 getParameters() 方法获取拍照参数, 该方法返回一个 Camera.Parameters 对象;
3. 调用 Camera.Parameters 对象方法设置相机参数;
4. 调用 Camera 的 setParameters(), 并将 Camera.Parameters 对象作为参数传入, 即可控制相机的拍照参数;
5. 调用 Camera 的 startPreview() 方法预览取景, 在预览取景前调用 Camera 的 setPreviewDisplay(surfaceHolder holder) 方法设置使用哪个 SurfaceView 来显示取景图片;
6. 调用 Camera 的 takePicture() 方法进行拍照;
7. 结束程序时, 调用 Camera 的 stopPreview() 方法结束取景预览;
8. 调用 release() 方法释放资源。

注:

在 Android SDK 中有两种 Camera classes:

- android.hardware.Camera: 这是用来操控相机功能的类;
- android.graphics.Camera: 可以实现将 2D 物件在 3D 空间中移动, 并将在其移动后的结果画在屏幕上的类别。

14.4.3 实验步骤

本实验我们一起来学习如何使用 Camera 拍照。新建工程 MultiMedia4_Camera, 其中 Activity 的名字为 MainActivity.java, 对应的布局文件名字为 activity_main.xml。其目录结构和布局文件界面如下图所示。

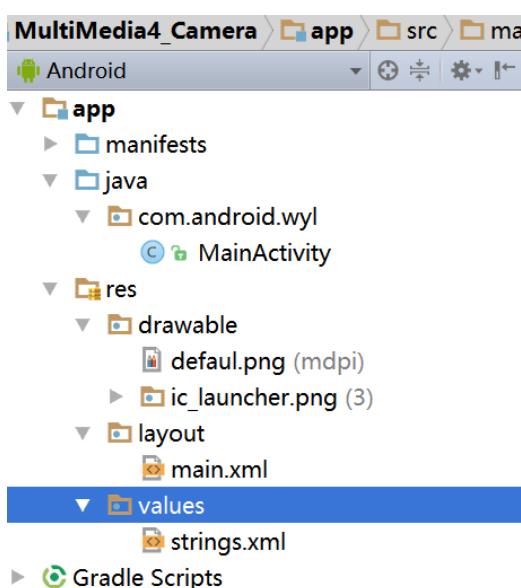


图 14.4.1 工程目录结构图



图 14.4.2 工程演示效果图

布局文件 activity_main.xml 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="@android:color/white"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/img"
        android:layout_width="200px"
        android:layout_height="200px"
        android:layout_gravity="center"
        android:scaleType="centerInside"
        android:src="@drawable/default" />

```

```
<Button  
    android:id="@+id	btn"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="选择头像" />  
</LinearLayout>
```

MainActivity.java 中的主要代码：

```
public class MainActivity extends Activity {  
  
    private Bitmap bitMap;  
  
    private boolean hasImage;  
  
    private ImageView imageView;  
  
    private Button button;  
  
    private String pathString;  
  
    /**  
     * Called when the activity is first created.  
     */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        pathString = Environment.getExternalStorageDirectory() + "\\picture";  
        File file = new File(pathString);  
        if (!file.exists()) {  
            file.mkdir();  
        }  
        button = (Button) findViewById(R.id.btn);  
        imageView = (ImageView) findViewById(R.id.img);
```

```
button.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        new AlertDialog.Builder(MainActivity.this).
            setTitle("上传照片").
            setItems(new String[]{"拍照上传", "本地上传"}, new
OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int
which) {
        // TODO Auto-generated method stub
        switch (which) {
            case 0:
                Intent cameraIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(cameraIntent,
1001);
                break;
            case 1:
                Intent localIntent = new Intent();
                localIntent.setType("image/*");
                localIntent.setAction("android.intent.action.GET_CONTENT");
                Intent localIntent2 =
Intent.createChooser(localIntent, "选择");
        }
    }
}
```

```
        startActivityForResult(localIntent2,
1002);

                break;

            default:

                break;
        }

    }

}).show();

}

});

}

}

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data)

{

    // TODO Auto-generated method stub

    switch (requestCode) {

        case 1002:

            if (bitMap != null && !bitMap.isRecycled()) {

                bitMap.recycle();

            }

            Uri selectedImageUri = data.getData();

            if (selectedImageUri != null) {

                try {


```

```

        bitMap =
BitmapFactory.decodeStream(getContentResolver().openInputStream(selectedImageU
ri));

        File file1 = new File(pathString,
System.currentTimeMillis() + ".jpg");

        FileOutputStream fos = new FileOutputStream(file1);

        byte[] buffer = new byte[1024];

        bitMap.compress(CompressFormat.JPEG, 100, fos);

        fos.write(buffer);

        fos.flush();

        fos.close();

        imageView.setImageBitmap(bitMap);

        hasImage = true;

    } catch (FileNotFoundException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

    } catch (IOException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

    }

} else {

    return;

}

break;

case 1001:

    Bundle bundle = data.getExtras();

    bitMap = (Bitmap) bundle.get("data");

    if (bitMap != null)

```

```

        bitMap.recycle();

        bitMap = (Bitmap) data.getExtras().get("data");

        File file2 = new File(pathString, System.currentTimeMillis() +
".jpg");

        try {

            FileOutputStream fos = new FileOutputStream(file2);

            byte[] buffer = new byte[1024];

            bitMap.compress(CompressFormat.JPEG, 100, fos);

            fos.write(buffer);

            fos.flush();

            fos.close();

        } catch (FileNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        } catch (IOException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        imageView.setImageBitmap(bitMap);

        hasImage = true;

        break;

    }

    default:

        break;

    }

// super.onActivityResult(requestCode, resultCode, data);

}

}

```

14.5 作业

14.5.1 自制音乐播放器

要求可以顺序播放多条音乐，或者循环播放单条音乐。

14.5.2 实现用户头像的拍照上传功能

在学生管理系统中显示学生的头像，并能够进行管理。

第15章 Android 中的传感器应用开发

本章的主要内容是对 Android 中传感器相关应用的介绍，学习 Android 中如何使用 SensorManager 来进行传感器的相关应用的开发。

通过本章实验我们将能够掌握 Android 中的常用传感器应用。

15.1 实验一 常见 Android 传感器的应用

15.1.1 实验目的

1. 了解手机自带的传感器及其应用场景。
2. 掌握常用传感器的基本使用方法

15.1.2 准备工作

1. Android 环境搭建成功。
2. 使用真机测试。

15.1.3 实验步骤

创建 Android 工程 Sensor1，建立 MainActivity.java 文件，主要核心代码如下：

```
public class MainActivity extends Activity {  
    private TextView tv;  
    private SensorManager sm;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);

setContentView(R.layout.main);

sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

        findViewById();

}

/* (non-Javadoc)
 * @see android.app.Activity#onResume()
 */
@Override

protected void onResume() {

    // TODO Auto-generated method stub

    super.onResume();

    sm.registerListener(sel,
        sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
        SensorManager.SENSOR_DELAY_UI);
}

@Override

protected void onStop() {

    // TODO Auto-generated method stub

    sm.unregisterListener(sel);

    super.onStop();
}

SensorEventListener sel = new SensorEventListener() {
```

```
//当传感器的值发生改变时触发该方法

@Override

public void onSensorChanged(SensorEvent event) {

    // TODO Auto-generated method stub

    String str;

    str = "X 方向的加速度是: "+event.values[0];

    str += "\nY 方向的加速度是: "+event.values[1];

    str += "\nZ 方向的加速度是: "+event.values[2];

    TextView.setText(str);

}

//当传感器经度发生改变时触发该方法

@Override

public void onAccuracyChanged(Sensor sensor, int accuracy) {

    // TODO Auto-generated method stub

}

};

private void findView() {

    // TODO Auto-generated method stub

    TextView = (TextView) findViewById(R.id.Tv);

}

}
```

本实验没有界面的呈现效果，Android 中的各种传感器都是通过监听器的方式获得传感器的相关数据，各位同学可以根据实际的需求进行相关应用的设计开发。

15.1.4 实验结论

通过本章实验我们将能够掌握 Android 中的常用传感器应用。

15.2 实验二 实现手机摇一摇功能

15.2.1 实验目的

1. 了解手机自带的传感器及其应用场景。
2. 掌握常用传感器的基本使用方法

15.2.2 准备工作

1. Android 环境搭建成功。
2. 使用真机测试。

15.2.3 实验步骤

步骤一：新建工程 AccelerateDemo，主 Activity 文件及布局文件的名字默认即可，其中布局文件包含 1 个文本框组件。

步骤二：编辑布局文件 activity_main.xml，采用线性布局。主要代码如下所示。

```
<TextView  
    android:id="@+id/Tv"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```

步骤三：编辑主 Activity 文件 MainActivity.java。除了继承 Activity 之外，还需实现 SensorEventListener 接口。在该文件中首先声明文本框组件并定义 sensor 管理器和震动。

```
public class MainActivity extends Activity implements SensorEventListener {  
    TextView tvStat;  
    // 定义 sensor 管理器  
    private SensorManager mSensorManager;  
    // 震动  
    private Vibrator vibrator;
```

在重写的 onCreate() 方法中获取布局文件中的文本框组件对象，并且获取传感器管理服务以及震动。

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    tvStat = (TextView) findViewById(R.id.tv_statue);  
    // 获取传感器管理服务  
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);  
    // 震动  
    vibrator = (Vibrator) getSystemService(Service.VIBRATOR_SERVICE);  
}
```

步骤四：实现 SensorEventListener 接口必须要实现 onAccuracyChanged() 和 onSensorChanged() 方法。

```
SensorEventListener sel = new SensorEventListener() {
    //当传感器的值发生改变时触发该方法
    @Override
    public void onSensorChanged(SensorEvent event) {
        // TODO Auto-generated method stub
        int sensorType = event.sensor.getType();
        float[] values = event.values;
        if(sensorType == Sensor.TYPE_ACCELEROMETER) {
            if (Math.abs(values[0])>14 ||
                Math.abs(values[1])>14 ||
                Math.abs(values[2])>14) {
                Tv.setText("您摇到一位好友");
            }
        }
    }
    //当传感器经度发生改变时触发该方法
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // TODO Auto-generated method stub
    }
};
```

步骤五：运行该 Android 工程， 使用真机测试该程序。

15.2.4 实验结论

希望各位同学能够熟练掌握 Android 中的传感器相关应用的开发，例如：微信中摇一摇功能的设计及开发。

第16章 Android 中的 GPS 应用开发

本章的主要内容是学习 Android 的 GPS 应用开发，学习 Android 中如何使用 GPS 来定位当前位置。

通过本章实验我们将能够掌握 Android 中的 GPS 定位的基本方法。

16.1 实验一 使用 GPS 定位当前位置

16.1.1 实验目的

本次试验我们将在 Android 中实现定位当前位置。

16.1.2 准备工作

1. GPS (Global Positioning System) 表示全球定位系统，是 20 世纪 70 年代由美国陆海空三军联合研制的新一代空间卫星导航定位系统，为全球的物体提供定位功能。

2. 支持 GPS 的核心 API：

- LocationManager 类，提供所有 GPS 定位相关的服务、对象；
- LocationProvider：定位提供者，是 GPS 定位支持的另一个重要的 API。该对象是 GPS 定位组件的抽象表示，可以获取该定位组件的相关信息。
- Location，它是一个代表位置信息的抽象类，提供的方法可获取具体的定位信息。

3. LocationManager 对象的获取方法：通过 Context 的 getSystemService() 方法获取。

```
LocationManager locManager = getSystemService(Context
```

. LOCATION_SERVICE)

4. 通过调用 LocationManager 的方法来获取 GPS 定位的相关服务和对象：

- Boolean addGpsStatusListener(GpsStatus.Listener listener)：添加一个 GPS 状态的监听器。通过该监听器可以实时更新 GPS 的定位信息；
- GpsStatus getGpsStatus(GpsStatus status)：获取 GPS 状态；
- Location getLastKnownLocation(String Provider) : 根据 LocationProvider 获取最近一次已知的 Location；
- boolean isEnabled(String provider)：判断指定名称的 LocationProvider 是否可用；
- void requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)：通过指定的 LocationProvider 周期性地获取定位信息，并通过 intent 启动相应的组件。
- void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)：通过指定的 LocationProvider 周期性地获取定位信息，并触发 listener 所对应的触发器。

5. 常用的 GPS 另一个重要 API：Location，它是一个代表位置信息的抽象类，提供了如下方法来获取定位信息：

- float getAccuracy()：获取定位信息的精度；
- double getAltitude()：获取定位信息的高度；
- float getBearing()：获取定位信息的方向；
- double getLatitude()：获取定位信息的纬度；
- double getLongitude()：获取定位信息的经度；
- String getProvider()：获取提供该定位信息的 LocationProvider；
- float getSpeed()：获取定位信息的速度；

16.1.3 实验步骤

步骤一：使用 Android GPS 的三个核心 API（）获取 GPS 定位信息的通用步骤：

- 1) 获取系统 LocationManager 对象；

2) 使用 LocationManager，通过指定 LocationProvider 来获取定位信息，定位信息由 Location 对象表示；

3) 从 Location 对象中获取定位信息；

步骤二： 新建工程 GPS1_Location，其中 Activity 的名字为 MainActivity.java，对应的布局文件名字为 main.xml，布局文件中包括 1 个文本框，其属性 id 为 tv，用于显示获取的定位信息。该工程目录结构如下图所示：

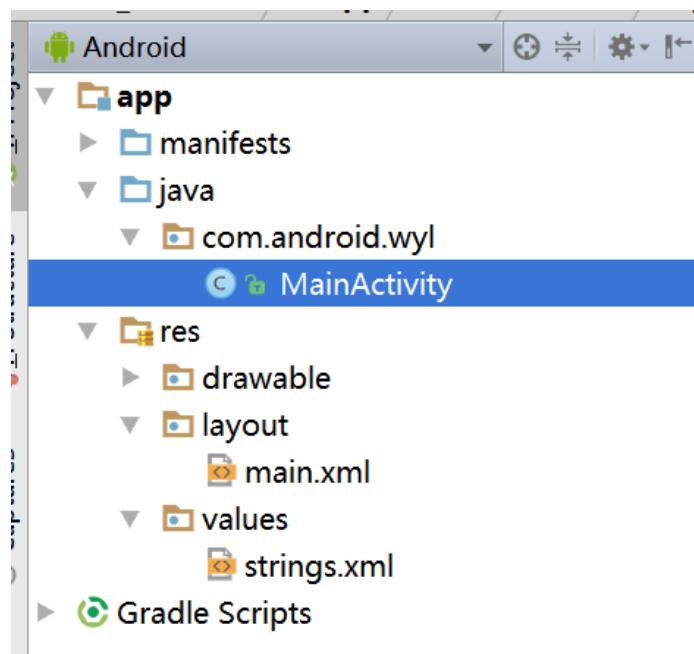


图 16.1.1 工程目录结构图

布局文件中文本框的 id 属性为 tv。

MainActivity.java 代码：

```
public class MainActivity extends Activity {

    private LocationManager lm;
    private Location location;
    private LocationListener locationListener = new LocationListener() {

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
            // TODO Auto-generated method stub
        }
    };
}
```

```
    }

    @Override
    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
    }

}

@Override
public void onProviderDisabled(String provider) {
    // TODO Auto-generated method stub
    updateToNewLocation(null);
}

@Override
public void onLocationChanged(Location location) {
    Log.e("11111", "22222");
    // TODO Auto-generated method stub
    updateToNewLocation(location);
}

};

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //1、获得所有的loactionprovider
    lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

```
List<String> providers = lm.getAllProviders();

Log.e("providers", providers.toString());

//2、根据locationprovider的名字获得loactionprovider

LocationProvider lp = lm.getProvider(LocationManager.GPS_PROVIDER);

//3、根据要求，获得最匹配的provider

Criteria myCri=new Criteria();

myCri.setAccuracy(Criteria.ACCURACY_FINE);//精确度

myCri.setAltitudeRequired(false);//海拔不需要

myCri.setBearingRequired(false);//方向

myCri.setCostAllowed(true);//允许产生现金消费

myCri.setPowerRequirement(Criteria.POWER_LOW);//耗电

String str = lm.getBestProvider(myCri, true);

Log.e("bestprovider", str);

lp = lm.getProvider(str);

location = lm.getLastKnownLocation(str);

// 设置监听器，自动更新的最长时间为间隔N秒(1秒为1*1000, 这样写主要为了方便)或最小位移变化超过N米

lm.requestLocationUpdates(str, 1* 1000, 0,
                           locationListener);

}

private void updateToNewLocation(Location location) {

    TextView tv1;

    tv1 = (TextView) this.findViewById(R.id.tv);

    if (location != null) {

        StringBuilder sb = new StringBuilder();

        sb.append("实时的位置信息: \n");

        sb.append("经度: " + location.getLongitude());

        sb.append("\n纬度: " + location.getLatitude());
    }
}
```

```

        sb.append("\n高度" + location.getAltitude());
        sb.append("\n速度: " + location.getSpeed());
        sb.append("\n方向:" + location.getBearing());
        tv1.setText(sb.toString());
    } else {
        tv1.setText("无法获取地理信息");
    }
}

//更新EditText中显示的内容

public void updateView(Location location) {
    if(location != null) {
        StringBuilder sb = new StringBuilder();
        sb.append("实时的位置信息: \n");
        sb.append("经度: " + location.getLongitude());
        sb.append("\n纬度: " + location.getLatitude());
        sb.append("\n高度" + location.getAltitude());
        sb.append("\n速度: " + location.getSpeed());
        sb.append("\n方向:" + location.getBearing());
        edt_show.setText(sb.toString());
    }
    else{
        //传入的Location对象为空，则清空EditText
        edt_show.setText("");
    }
}
}

```

步骤三：该程序需要访问定位模块信息，需要授予定位模块的权限

<!-- 授予程序获取定位信息的权限 -->

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

步骤四：运行程序

1) 如果使用真机测试该程序，需要先开启相应的位置服务：

设置 → 位置服务 → GPS卫星定位

2) 使用模拟器测试该程序的方法：

Android模拟器本身不能作为GPS接收机，需要借助DDMS工具给模拟器发送模拟的GPS定位信息。

首先，启动Genymotion Android模拟器；其次，打开Genymotion的GPS设置模块，设置GPS信息；最后，模拟器具有模拟定位信息，如下图所示：



图16.1.2 向模拟器发送定位信息图

步骤五：运行结果如下图所示：

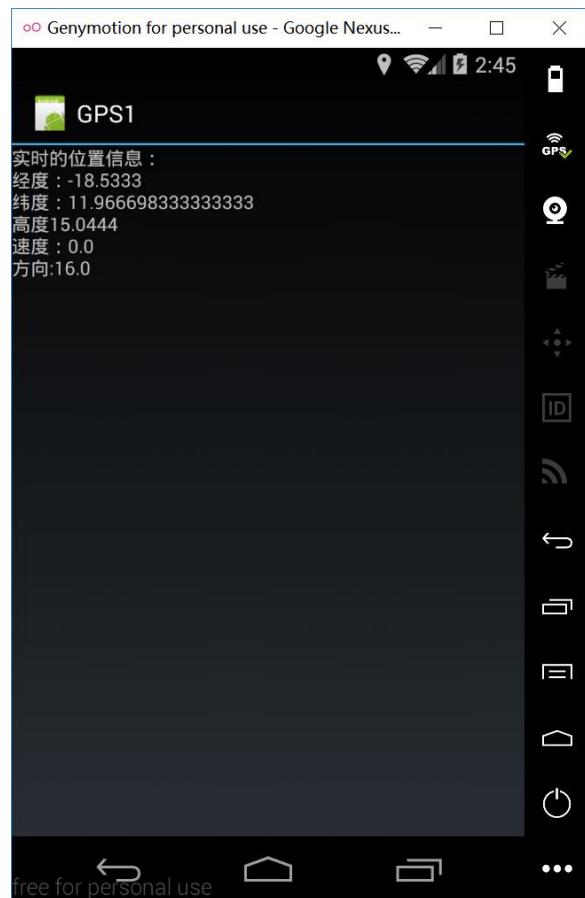


图16.1.3 运行结果图

16.1.4 实验结论

通过Android系统自带定位API的应用，可以通过三种方式：网络、基站和GPS，来获取到系统用户的经纬度等位置信息，根据这些信息我们可以方便的进行关于用户位置相关应用的开发。

16.2 作业

16.2.1 定位当前位置

请实现 demo，实时更新用户的位置。并可参考如下地址：

<http://blog.csdn.net/lyq8479/article/details/6387860>，实现通过系统 API 得到位置，然后调用百度 API 接口，地图显示当前位置。

附章 AndroidGoogleMap 服务

16.3 实验一 AndroidGoogleMap 服务

16.3.1 实验目的

1. 掌握 Android 中的 GoogleMap 的基本使用方法。

16.3.2 准备工作

1. Android 环境搭建成功。
2. 保证模拟器运行正常。

16.3.3 实验步骤

➤ 任务：使用 GoogleMap 定位当前位置，并能随位置改变而不断改变地图中的位置信息。

步骤一：

GoogleMap 环境准备。

Android 系统默认并不支持调用 Google Map，为正常调用 Google Map 服务，需要先进行如下准备工作。

- 1) 获取 Map API Key

单击 Eclipse 主菜单：Window → Preferences → 单击左侧“Android” → Build → 弹出如图 1 所示。

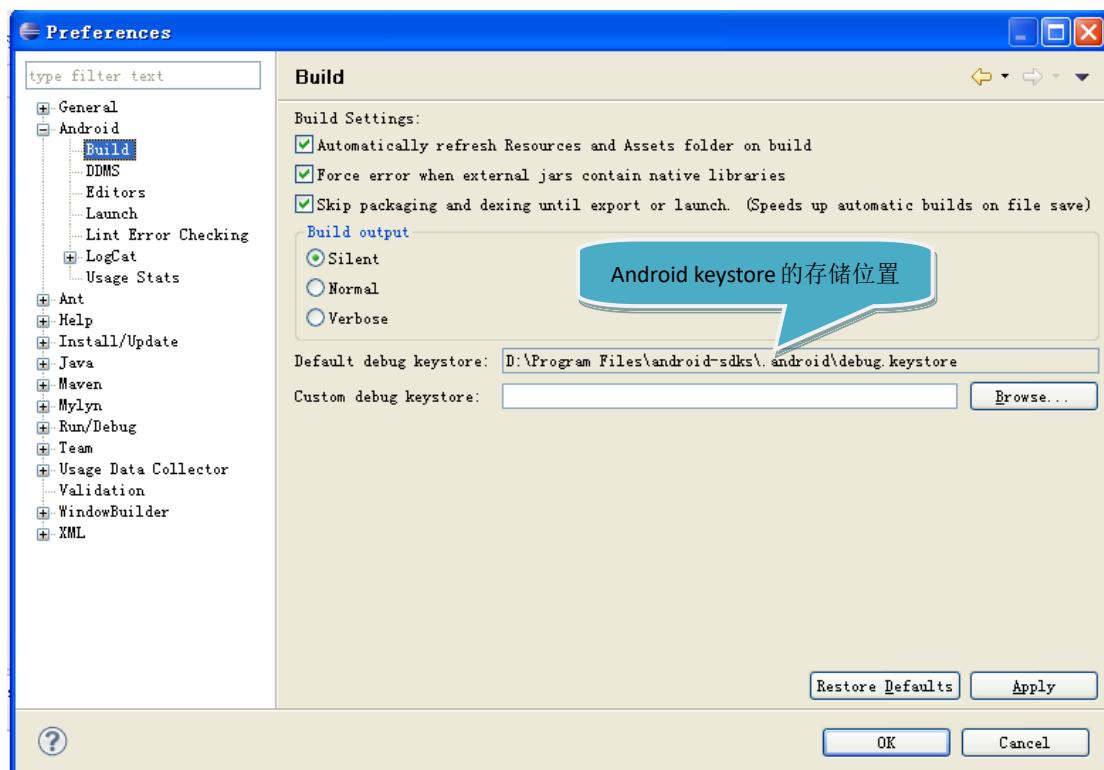


图 1 查看 Android 模拟器的 Keystore

- 2) 上图中显示的便是模拟器的 keystore 的存储位置, 接下来应用程序需要根据该 keystore 来生成 Google API 的 Key。
- 3) 用 JDK 提供的 keytool 工具为 Android keystore 生成认证指纹, 启动命令行窗口输入如下命令:

```
keytool -list -keystore <Android keystore 的存储位置>
```

其中<Android keystore 的存储位置>要替换成图 1 中 Android keystore 的存储位置, 如: Keytool -list -keystore "D:\Program Files\android-sdks\.android\debug.keystore"。

```
D:\>keytool -list -keystore "D:\Program Files\android-sdks\.android\debug.keystore"
输入密钥库口令:
密钥库类型: JKS
密钥库提供方: SUN

您的密钥库包含 1 个条目

androididdebugkey, 2013-1-7, PrivateKeyEntry,
证书指纹 <SHA1>: [REDACTED]
```

图 2 指纹

就会得到 MD5 的指纹, 记录下此记录: 二十段用冒号割开的数字段, 每段是

两个十六进制的数(图 2 中红色删除线下的部分)。

4) 在 Google APIs Console 上创建项目，并且注册 Maps API。

①首先，去这个网址：<https://code.google.com/apis/console/>

用 Gmail 的账户登录，如果是第一次的话，需要创建项目，默认情况会创建一个叫做 API Project 的项目。

登陆之后出现页面（如图 3 所示）：

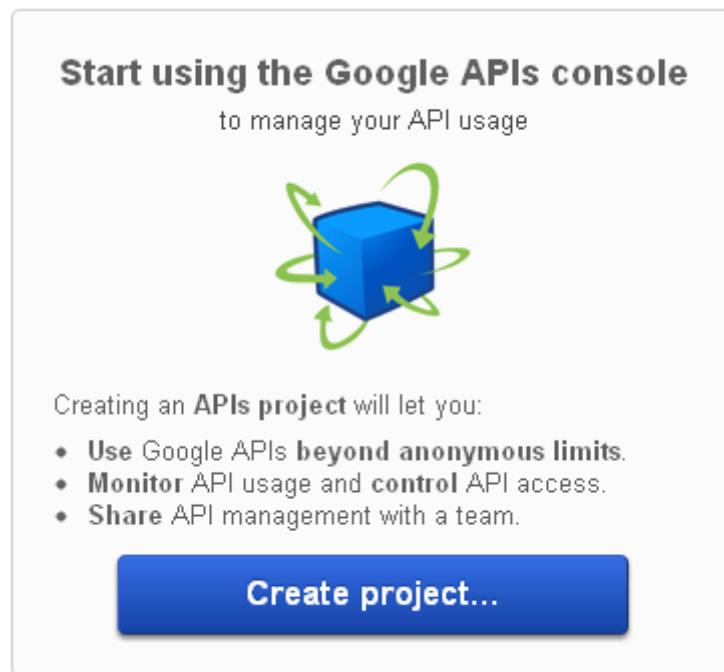


图 3 创建 API Project 工程

②单击“Create project...”后到达页面(如图 4 所示)：

The screenshot shows the 'All services' page of the Google APIs Console. On the left, a sidebar has tabs for 'API Project' (selected), 'Overview', 'Services', 'Team', and 'API Access'. The main area shows a table titled 'All services' with columns for 'Service', 'Status', and 'Notes'. Services listed include Ad Exchange Buyer API, Ad Exchange Seller API, AdSense Host API, AdSense Management API, Analytics API, Audit API, BigQuery API, Blogger API v3, Books API, Calendar API, and Custom Search API. Most services have their status set to 'OFF'. Some services like AdSense Host API and Books API have a 'Request access...' link instead of 'OFF'. Notes for each service provide courtesy limits and links to pricing information.

图 4 服务页面

③点击左边的 Services，会在中间看到很多的 APIs 和 Services，找到 Google Maps Android API v2，然后把它设置成 on，需要接受一些服务条款，如图 5 所示。



图 5 开启的服务页面

之后跳转到页面，如图 6 所示：

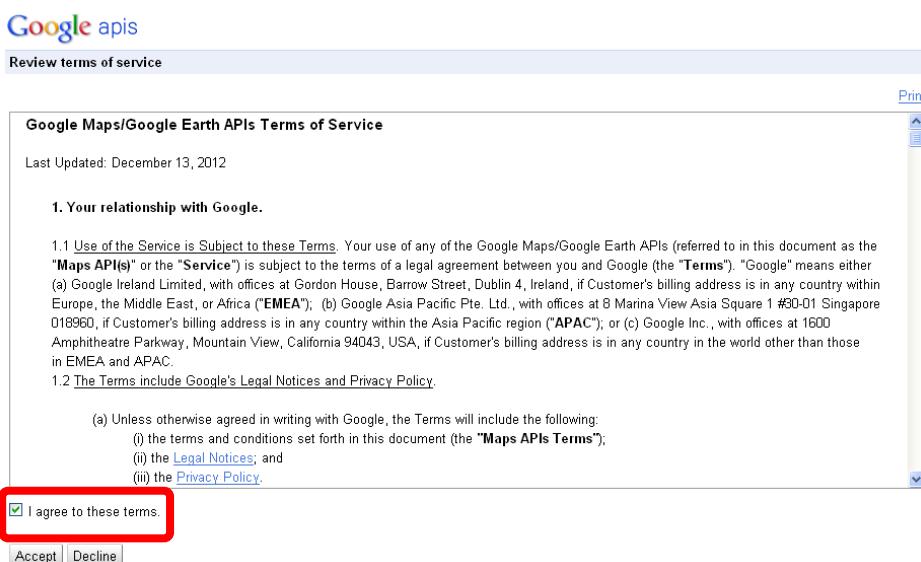


图 6 同意条款

④勾选同意条款，单击接受按钮。

5) 获得 API Key

在左边的导航条中选择 API Access（如图 7）。



图 7 导航栏

在出来的页面中选择 Create New Android Key... 就可以生成 key 了(如图 8 所示)。

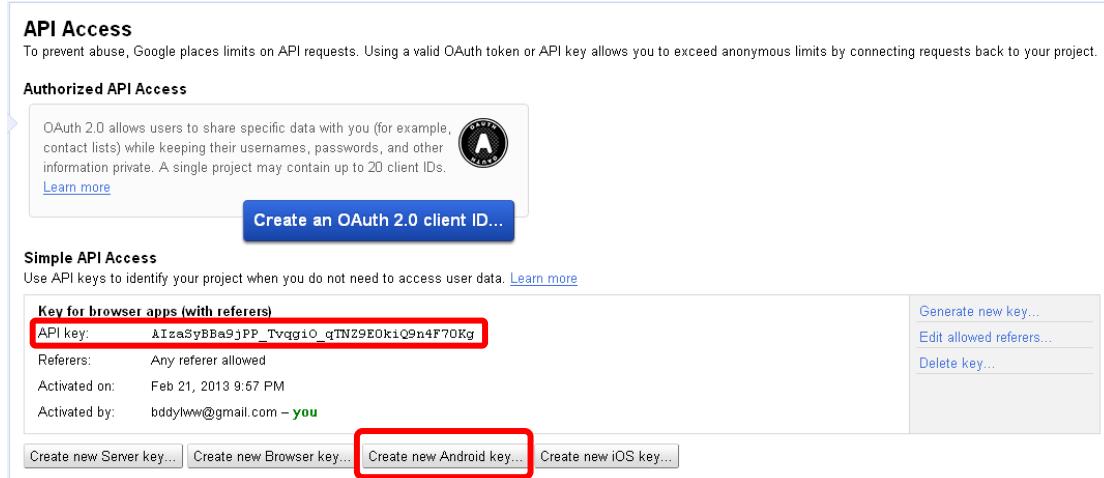


图 8 生成 Key

然后在对话框中填入: SHA-1 指纹, 分号隔开, 然后是应用的 package name.
然后就会生成一个 Key, 如图 9 所示。



图 9 生成自己的 Key 操作图

最后生成的 API Key 如图 10 所示:

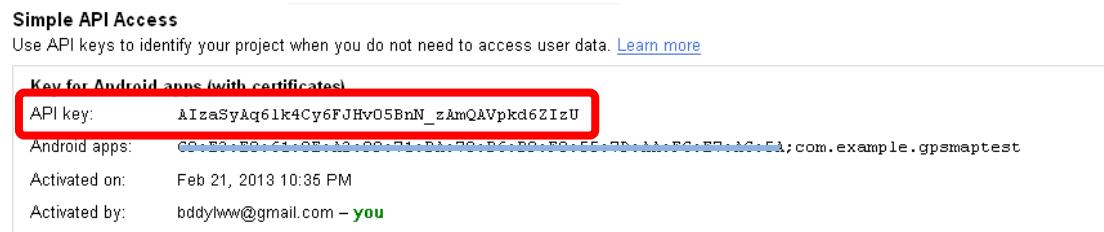


图 10 生成的 Key

步骤二:

把 API Key 加入应用程序。

- 1) 首先, 新建 Android 应用程序。创建应用程序时, 注意包名应该和申请 key 时候的包名一致。

2) 之后修改 AndroidManifest.xml 文件，在<application>元素中加入子标签。

```
<meta-data  
    android:name="com.google.android.maps.v2.API_KEY"  
    android:value="your_api_key" />
```

注：其中 your_api_key 置换成自己申请的 API Key。

在该文件中加入一些许可信息，允许必要的权限。

```
<permission  
    android:name="com.example.mapdemo.permission.MAPS_RECEIVE"  
    android:protectionLevel="signature"/>  
<uses-permission android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>
```

注：其中 com.example.mapdemo 换成自己的包名。

步骤三：

AndroidManifest.xml 文件中的其他的选项设置。

1) 许可设置

```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>  
<uses-permission  
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

2) OpenGL ES V2 特性支持(作为<manifest> 的子元素)

```
<uses-feature  
    android:glEsVersion="0x00020000"  
    android:required="true"/>
```

步骤四：

在布局文件中加上地图。

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/map"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    class="com.google.android.gms.maps.MapFragment"/>
```

遇到的问题和解决的方法。

程序编译错误，显示找不到一些类，如图 11 所示。

L...	Time	PID	TID	Application	Tag	Text
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at android.app.Fragment.instantiate(Fragment.java:564)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at android.app.Fragment.instantiate(Fragment.java:552)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at android.app.Activity.onCreateView(Activity.java:4656)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at android.view.LayoutInflater.createViewFromTag(LayoutInflater.java:680)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	... 21 more
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	Caused by: java.lang.ClassNotFoundException: com.google.android.gms.maps.MapF
						ragment
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:61)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at java.lang.ClassLoader.loadClass(ClassLoader.java:501)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at java.lang.ClassLoader.loadClass(ClassLoader.java:461)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	at android.app.Fragment.instantiate(Fragment.java:574)
E	12-26 12:50:05.754	708	708	com.example.maptest	AndroidRuntime	... 24 more

图 11 错误提示信息

解决这个问题，首先需要把 Google Play services 的类库加载进来：

在 Eclipse 里面选择：File > Import > Android > Existing Android Code Into Workspace 然后点击 Next.

之后 Browse...，找到路径下的<android-sdk-folder>/extras/google/google_play_services/libproject/google-play-services_lib，然后选择 Finish。

添加对这个库的引用：在自己的项目上右键，选 Properties，左边选 Android，然后在下面的 Library 里面 Add 刚才的 google-play-services_lib，如图 12 所示。

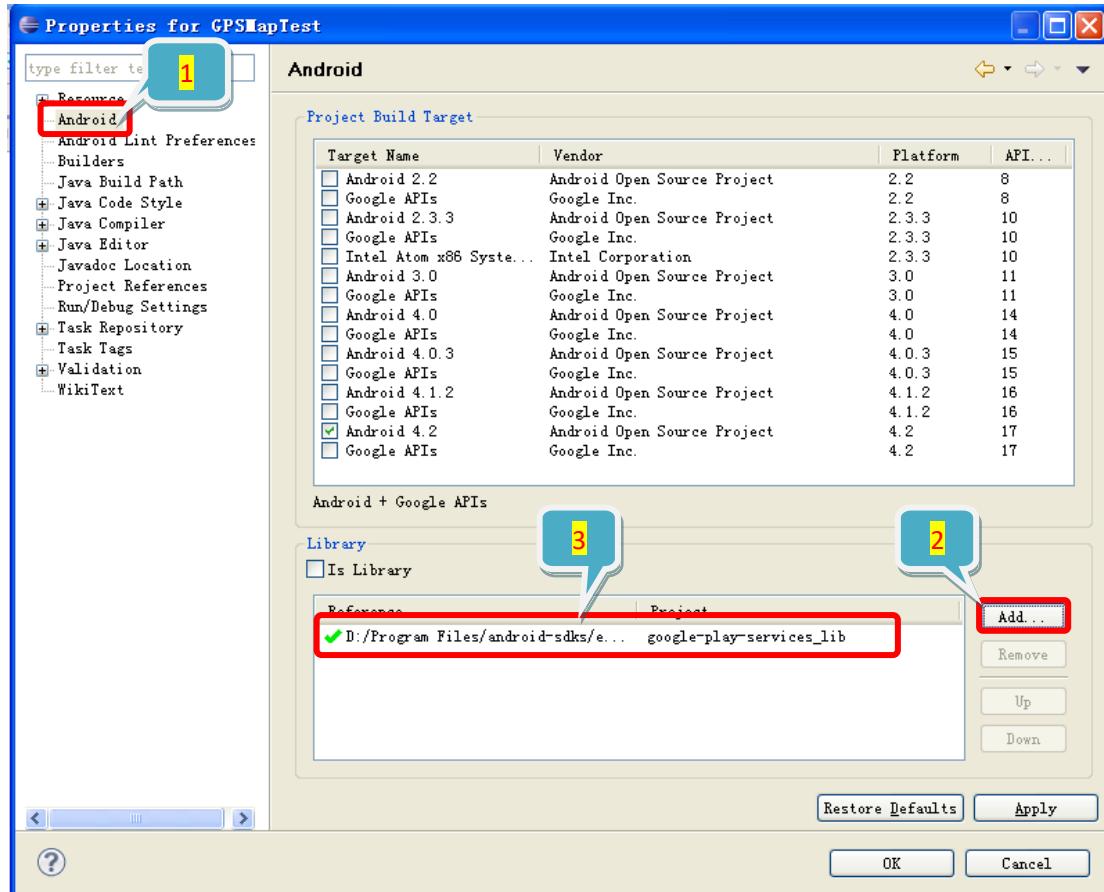


图 12 引用类库步骤图

之后运行程序就应该出来地图了。运行过程中，有可能会碰到下面的问题：程序运行成功，但是显示 This app won't run unless you update Google Play services，如图 13 所示。需要点击 Update，按照提示操作。

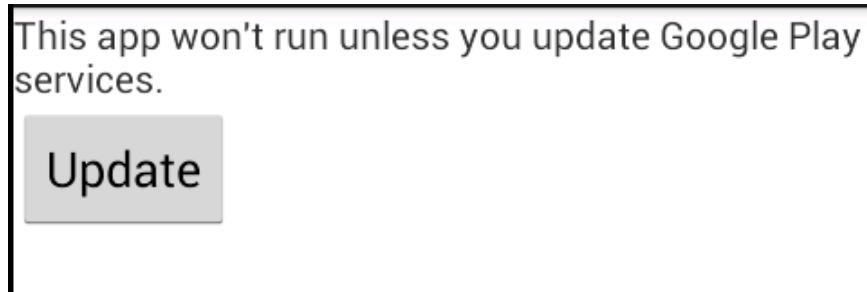


图 13 提示界面图

之后运行程序，就出地图了，如图 14 所示。：

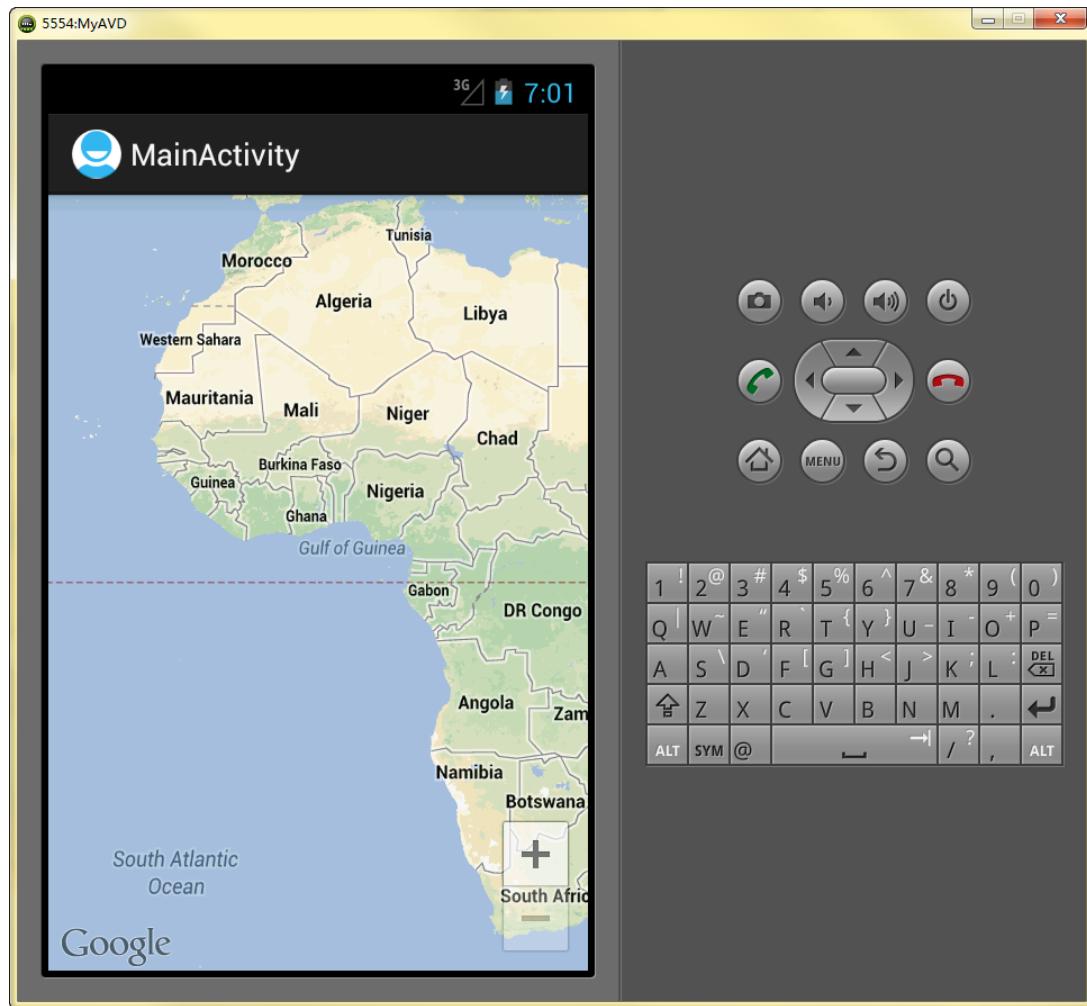


图 14 运行结果图

注：

因为 MapFragment 只在 API 12 及之后的版本才有，所以对于之前的版本需要使用 Support Library 来进行辅助。

如果 `minSdkVersion` 设置为 12 以前的，就需要使用 **Support Library**。

需要更改的地方是：布局文件中，把 MapFragment 改为 SupportMapFragment。

MainActivity 继承自 FragmentActivity 而不是 Activity。（需要 import `android.support.v4.app.FragmentActivity;`）

步骤五：

新建工程 GoogleMapTest，包名为 `com.example.googlemaptest`，将一张名为 `pos.png` 的图片导入到 `drawable` 目录下，用于标注当前的位置。项目中 Activity 的名字为 `MainActivity.java`，对应的布局文件名字为 `activity_main.xml`，布局文件中包括 2 个输入框，用于接收用户输入的经纬度

信息，有 1 个按钮，用于根据用户输入的经纬度定位具体位置；有两个单选按钮，分别显示普通地图和卫星地图；有 1 个地图组件。该工程目录结构及布局文件界面如图 15 所示：

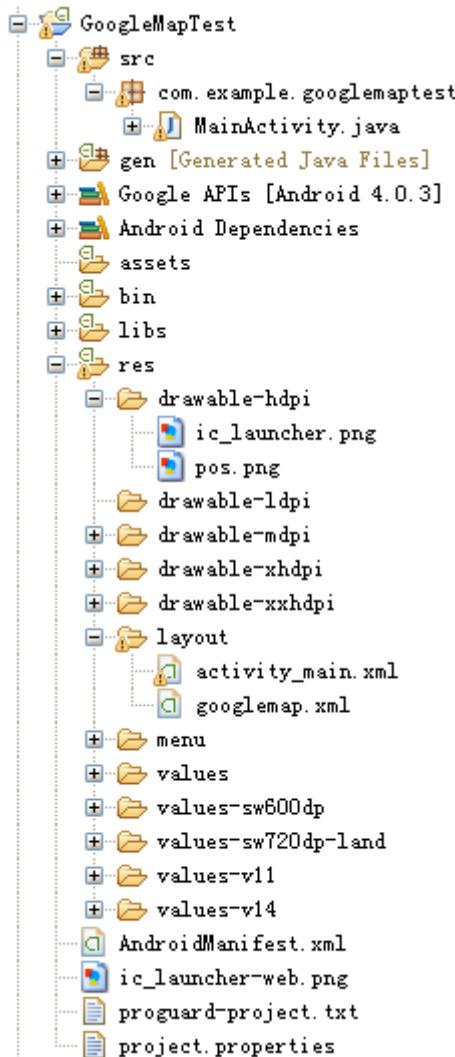


图 15 工程目录结构图

步骤六：

编辑地图组件对应的布局文件 googlemap.xml，主要代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:id="@+id/mapView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.MapFragment"
    map:cameraBearing="45"
    map:cameraTargetLat="25.033611"
    map:cameraTargetLng="121.565000"
    map:cameraTilt="0"
    map:cameraZoom="13"
    map:uiCompass="true"
    map:mapType="normal"
    map:uiRotateGestures="true"
    map:uiScrollGestures="true"
    map:uiTiltGestures="true"
    map:uiZoomControls="false"
    map:uiZoomGestures="true" />
```

注：此文件中，class 属性以下的几行代码，是用于设置地图的一些属性，这部分内容不是必须的，可以根据自己的需要进行设置，也可以都不设置，而在 Activity 中使用代码进行设置。

当在XML文件中加入这些map属性的设置时，必须要添加命名空间：

“`xmlns:map="http://schemas.android.com/apk/res-auto"`”，并且该XML文件中不能再加入其它的组件，例如编辑框，甚至不能为其添加容器，例如 LinearLayout。如果设置了这些属性，并且还想添加其它组件，就需要将该XML文件作为主XML布局文件的一部分，通过`<include>`标签包含在主布局文件中。具体方法稍后具体说明。

如果不在 XML 文件中设置这些 map 属性的话，该 XML 文件可以不添加命名空间，并且可以在此文件中添加其它的组件。

步骤七：

编写主布局文件：`activity_main.xml`，采用线性布局并将上面的地图组件对应的 XML 文件（`googlemap.xml`）包含进来。

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="定位"
        android:textSize="15sp" />
    <LinearLayout
        android:id="@+id/location"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="horizontal" >
        <!-- 定义输入经度值的文本框 -->
        <EditText
            android:id="@+id/edt_lng"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="6"
            android:hint="经度"
            android:textSize="20sp" />
        <!-- 定义输入纬度值的文本框 -->
        <EditText
            android:id="@+id/edt_lat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="5dp"
            android:ems="6"
            android:hint="纬度"
            android:textSize="20sp" />
    <Button
        android:id="@+id/btn_loc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="5dp"
        android:text="定位"
        android:textSize="15sp" />
</LinearLayout>
```

```

<LinearLayout
    android:id="@+id/mapsChoice"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <!-- 定义选择地图类型的单选按钮 -->
    <RadioGroup
        android:id="@+id/rg_mapType"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="horizontal">
        <RadioButton
            android:id="@+id/rb_nomal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:checked="true"
            android:text="普通视图"/>
        <RadioButton
            android:id="@+id/rb_satellite"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="卫星视图"/>
    </RadioGroup>
</LinearLayout>
</LinearLayout>
<include android:id="@+id/googleMap"
    layout="@layout/googlemap" />

```

步骤八：

编写 Activity 文件， MainActivity. java。

声明各组件及必要的类。

```

//定义界面上的可视化组件
private Button btn_loc;
private EditText edt_lng, edt_lat;
private RadioGroup rg_mapType;
GoogleMap mMap;
private CameraPosition cameraPosition;
private MarkerOptions markerOpt;
//定义LocationManager对象
private LocationManager locManager;
private Location location;
private String bestProvider;

```

在 onCreate() 方法中获取各组件，并给按钮注册事件监听器，实现地理位置的实时定位。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //获取用户界面的组件
    findViews();
    //创建LocationManager对象,并获取Provider
    initProvider();
    //取得地图组件
    mMap = ((MapFragment)getFragmentManager()
        .findFragmentById(R.id.mapView)).getMap();
    mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    //更新位置信息
    updateToNewLocation(location);
    //给按钮添加监听器
    btn_loc.setOnClickListener(new MapClickedListener());
    //为RadioGroup的选中状态改变添加监听器
    rg_mapType.setOnCheckedChangeListener(new ChangeMapTypeListener());
    // 设置监听器，自动更新的最小时间为间隔N秒(1秒为1*1000，这样写主要为了方便)或最小位移变化超过N
    locManager.requestLocationUpdates(bestProvider, 3 * 1000, 8
        , new LocationListener() {
        //当Provider的状态改变时
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
    }}
```

```

@Override
public void onProviderEnabled(String provider) {
    // 当GPS LocationProvider可用时，更新位置
    location = locManager.getLastKnownLocation(provider);
}

@Override
public void onProviderDisabled(String provider) {
    updateToNewLocation(null);
}

@Override
public void onLocationChanged(Location location) {
    // 当GPS定位信息发生改变时，更新位置
    updateToNewLocation(location);
}

private void initProvider() {
    //创建LocationManager对象
    locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    // List all providers:
    List<String> providers = locManager.getAllProviders();
    Criteria criteria = new Criteria();
    bestProvider = locManager.getBestProvider(criteria, false);
    location = locManager.getLastKnownLocation(bestProvider);
    System.out.println("经度：" + location.getLatitude() + "纬度：" + location.getLongitude());
}

//获取用户界面组件
private void findViews() {
    //获取界面上的两个按钮
    btn_loc = (Button) findViewById(R.id.btn_loc);
    //获取界面上的两个文本框
    edt_lng = (EditText) findViewById(R.id.edt_lng);
    edt_lat = (EditText) findViewById(R.id.edt_lat);
    //获得RadioGroup
    rg_mapType = (RadioGroup) findViewById(R.id.rg_mapType);
}

```

实现各个事件监听器类。

```

//定位按钮的点击事件监听器
private class MapClickedListener implements OnClickListener{
    //根据用户输入经纬度定位
    @Override
    public void onClick(View v) {
        //获取用户输入的经纬度
        String lng = edt_lng.getText().toString().trim();
        String lat = edt_lat.getEditableText().toString().trim();
        if(lng.equals("") || lat.equals("")){
            Toast.makeText(getApplicationContext(), "请输入有效的经纬度信息!
                ", Toast.LENGTH_LONG).show();
            location = locationManager.getLastKnownLocation(bestProvider);
            updateToNewLocation(location);
        }else{
            location.setLongitude(Double.parseDouble(lng));
            location.setLatitude(Double.parseDouble(lat));
            //调用方法更新地图定位信息
            updateToNewLocation(location);
        }
    }
}

private class ChangeMapTypeListener implements OnCheckedChangeListener{
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        switch(checkedId){
            case R.id.rb_normal://如果勾选的是"正常视图"的单选按钮
                mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
                break;
            case R.id.rb_satellite://如果勾选的是"卫星视图"的单选按钮
                mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
                break;
        }
    }
}

```

实现更新位置信息的方法。

```
private void updateToNewLocation(Location location){
    mMap.clear();
    //Marker1
    markerOpt = new MarkerOptions();
    double dLong = 114.51500;
    double dLat = 38.042000;
    if(location != null){
        //获取经度
        dLong = location.getLongitude();
        //获取纬度
        dLat = location.getLatitude();
    }
    markerOpt.position(new LatLng(dLat, dLong));
    markerOpt.draggable(false);
    markerOpt.visible(true);
    markerOpt.anchor(0.5f, 0.5f);//设为图片中心
    markerOpt.icon(BitmapDescriptorFactory.fromResource(R.drawable.pos));
    mMap.addMarker(markerOpt);
    //将摄影机移动到指定的地理位置
    cameraPosition = new CameraPosition.Builder()
        .target(new LatLng(dLat, dLong))
        .zoom(15) // 缩放比例
        .bearing(0) // Sets the orientation of the camera to east
        .tilt(30) // Sets the tilt of the camera to 30 degrees
        .build();// Creates a CameraPosition from the builder
    mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
}
```

步骤九：

编辑 AndroidManifest.xml，需要添加必要的权限及所申请的 API Key。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <uses-sdk
        ... />
    <!—下面的权限中， com.example.googlemaptest 要换成你的工程文件的包名-->
    <permission android:name="com.example.googlemaptest.permission.MAPS_RECEIVE"
        android:protectionLevel="signature"/>
    <uses-permission
        android:name="com.example.googlemaptest.permission.MAPS_RECEIVE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
        android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <!—下面的两条权限在API V2中是不要求的，但是在开发过程中建议添加 -->
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
    <application
        ...
        <activity
            android:name="com.example.googlemaptest.MainActivity"
            ...
            </activity>
            <meta-data android:name="com.google.android.maps.v2.API_KEY"
                <!—下面的属性值要求添加你申请的API Key值-->
                android:value="你申请的API Key值 " />
            <uses-library android:required="true"
                android:name="com.google.android.maps" />
        </application>
```

步骤十：

运行改项目，测试定位功能。

注，由于本项目提取历史定位数据，所以首次使用，在没有历史数据的情况下会闪退，需要首先给定一个历史定位数据，才能进行功能定位的测试。

16.3.4 实验结论