第3章 Android开发环境

3.1 Android系统开发环境

3.1.1 Android系统与平台架构

1. Android系统特性

(1) 系统开源，Android底层使用Linux内核，使用的是GPL许可证，意味着相关的代码是必须开源的。

(2) 具有跨平台特性。

(3) 具有丰富的应用。

(4) Google强大的技术支持。

(5) 采用软件叠层方式构建。

2. Android平台架构

Android作为一个移动设备系统平台，其采用软件堆层的架构，共分为4层，自下而上分别是Linux内核(操作系统，即OS)、中间件层、应用程序框架和应用程序，如图3-1所示。

(1) Linux内核层以Linux核心为基础，由C语言开发，只提供基本功能，是硬件和其他软件堆层之间的一个抽象隔离层，提供安全机制、内存管理、进程管理、网络协议栈和电源管理等。

(2) 中间件层包括函数库(Library)和虚拟机(Virtual Machine)，由C++开发，由函数库和Android运行时构成，如图3-2所示。

① 函数库。主要提供一组基于C/C++的函数库，包括：

- Surface Manager，支持显示子系统的访问，提供应用程序与2D、3D图像层的平滑连接。
- Media Framework，实现音视频的播放和录制功能。
- SQLite，轻量级的关系数据库引擎。
- OpenGL ES，基于3D图像加速。

图 3-1　Android 系统结构

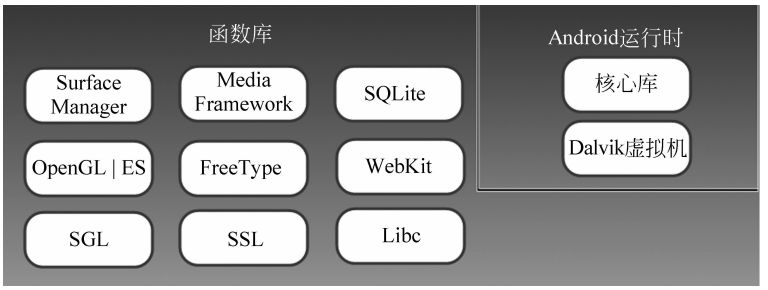


图 3-2　Android 系统中间件层

- FreeType，位图与矢量字体渲染。
- WebKit，Web 浏览器引擎。
- SGL，2D 图像引擎。
- SSL，数据加密与安全传输的函数库。
- Libc，标准 C 运行库，Linux 系统中底层应用程序开发接口。

② Android 运行时。

- 核心库，提供 Android 系统的特有函数功能和 Java 语言函数功能。
- Dalvik 虚拟机，实现基于 Linux 内核的线程管理和底层内存管理。

（3）应用程序框架层包含操作系统的各种管理程序，如图 3-3 所示，提供 Android 平台基本的管理功能和组件重用机制，包含：

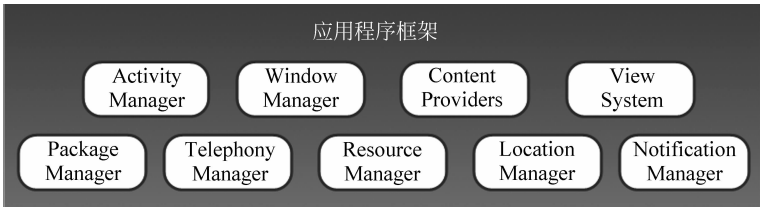


图 3-3 Android 系统应用程序框架层

- Activity Manager，管理应用程序的生命周期。
- Window Manager，启动应用程序的窗体。
- Content Providers，共享私有数据，实现跨进程的数据访问。
- Package Manager，管理安装在 Android 系统内的应用程序。
- Telephony Manager，管理与拨打和接听电话的相关功能。
- Resource Manager，允许应用程序使用非代码资源。
- Location Manager，管理与地图相关的服务功能。
- Notification Manager，允许应用程序在状态栏中显示提示信息。

(4) 应用程序层是最上层，有各种应用软件，如邮件客户端、浏览器、通讯录、日历等，应用软件则由各公司自行开发，用 Java 编写，如图 3-4 所示。

图 3-4 Android 系统应用程序层

3.1.2 Android 开发框架

开发框架方面包含基本的应用功能开发、数据存储、网络访问三大块。

1. 应用功能开发

一般而言，一个标准的 Android 程序由如下 4 部分组成，即 Activity、Broadcast Intent Receiver、Service、Content Provider。

1) Activity

Activity 是最频繁、最基本的模块。在 Android 中，一个 Activity 就是手机上一屏，相当于一个网页，不同的是，每个 Activity 运行结束了，有一个返回值，类似于一个函数。Android 系统会自动记录从首页到其他页面的所有跳转记录并且自动将以前的 Activity 压入系统堆栈，用户可以通过编程的方式删除历史堆栈中的 Activity Instance。

Activity 类主要是与界面资源文件关联起来(res/layout 目录下的 xml 资源，也可以不含任何界面资源)，内部包含控件的显示设计、界面交互设计、事件的响应设计以及数据处理设计、导航设计等 Application 设计的方方面面。

2) Broadcast Intent Receiver

Intent 提供了各种不同 Activity 进行跳转的机制，譬如，如果从 A activity 跳转到

B activity，则使用 Intent 实现如下：

```
Intent in =new Intent(A.this, B.class);
startActivity(in);
```

BroadcastReceiver 提供了各种不同的 Android 应用程序进行进程间通信的机制，如当电话呼叫来临时，可以通过 BroadcastReceiver 发布广播消息。对于用户而言，BroadcastReceiver 是不透明的，用户无法看到这个事件，BroadcastReceiver 通过 NotificationManager 通知用户这些事件发生了，它既可以在资源 AndroidManifest. xml 中注册，也可以在代码中通过 Context. registerReceiver()注册，只要注册了，当事件来临时，即使程序没有启动，系统在需要时也会自动启动此应用程序；另外，各应用程序可方便地通过 Context. sendBroadcast()将自己的事件广播给其他应用程序。

3) Service

这里讲的 Service 与 Windows 中的 Service 是一个概念，用户可以通过 startService (Intent service)启动一个 Service，也可以通过 Context. bindService 绑定一个 Service。

4) Content Provider

由于 Android 应用程序内部的数据都是私有的，Content Provider 提供了应用程序之间数据交换的机制，一个程序可以通过实现一个 ContentProvider 的抽象接口将自己的数据暴露出去，并且隐蔽了具体的数据存储实现，标准的 ContentProvider 提供了基本的 CRUD(Create，Read，Update，Delete)的接口，并且实现了权限机制，保护了数据交互的安全性。

一个标准的 Android 应用程序的工程文件包含如下六大部分。

① Java 源代码部分(包含 Activity)，都在 src 目录中。

② R. java 文件，这个文件是 Eclipse 自动生成与维护的，开发者不需要修改，提供了 Android 对的资源全局索引。

③ Android Library，这是应用运行的 Android 库。

④ assets 目录，这个目录主要用于放置多媒体等一些文件。

⑤ res 目录，放置的是资源文件，与 VC 中的资源目录类似，其中的 drawable 包含的是图片文件，layout 里面包含的是布局文件，values 目录里面主要包含的是字符串(strings. xml)、颜色(colors. xml)以及数组(arrays. xml)资源。

⑥ AndroidManifest. xml，这个文件非常重要，是整个应用的配置文件。在这个文件中需要声明所有用到的 Activity、Service、Receiver 等。

2. 数据存储

在 Android 系统中可供选择的存储方式包括 SharedPreferences、文件存储、SQLite 数据库存储、内容提供器(Content Provider)以及网络存储 5 种，具体如下。

1) SharedPreferences

SharedPreferences 是 Android 提供的一种配置文件读写方式，默认存在应用的 data/＜package name＞/shared _ prefs 下，通过 getSharedPreferences(xx, 0)；获取 SharedPreferences 对象进行读写操作。

2) 文件存储

通过 openFileInput、openFileOutput 等系统提供的 API 进行数据的读写访问，特别需

要注意的是，在 Android 中应用程序的数据是私有的，这就是说，当前应用程序产生的文件其他应用程序无法访问。

3）SQLite 数据库存储

SQLite 数据库存储方式是通过继承 SQLiteOpenHelper 类，并且获取此类的应用程序级别的实例进行数据库操作的，该类中提供了默认的 CRUD 访问接口，方便了应用程序的数据存储操作。

4）内存提供器

内容提供器(Content Provider)方式，如在上面应用方面论述的一样，通过调用其他应用程序的数据接口实现数据的读写访问。

5）网络存储

网络存储主要是通过下面要提到的网络访问该网络提供的网络服务接口实现数据的读写服务(如 WebService 数据访问接口)。

3. 网络访问

网络访问主要是 HTTP 访问技术的封装，通过 Java.NET.*；以及 Android.net.*；下面提供的 HttpPost、DefaultHttpClient、HttpResponse 等类提供的访问接口实现具体的 Web 服务访问。

3.1.3 Android 开发环境的搭建

1. JDK 的安装与配置

1）下载 JDK

到 Oracle 公司官网(http://www.oracle.com/technetwork/java/javase/downloads/index.html)下载 JDK 安装包，进入下载页面，根据自己的机型选择相应的版本。JDK 下载界面如图 3-5 所示。操作系统选择界面如图 3-6 所示。

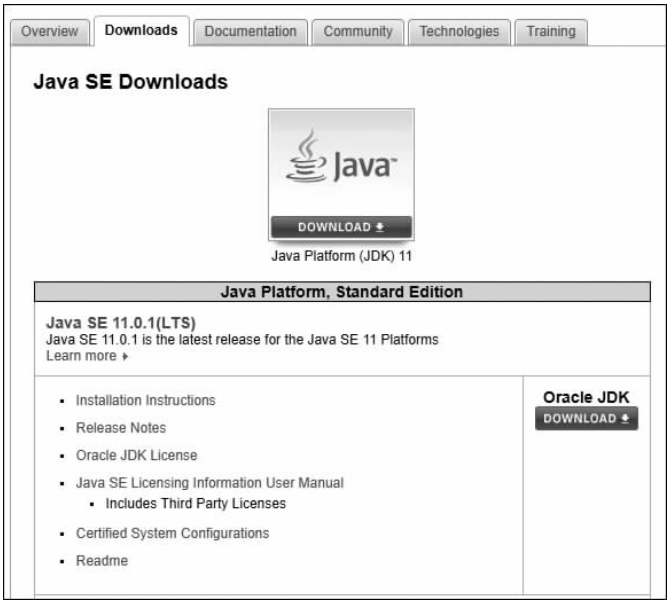
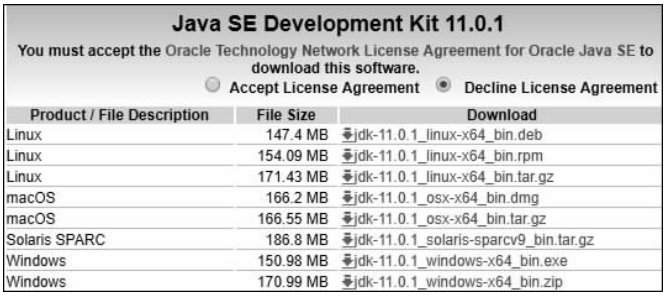


图 3-5 JDK 下载界面



Java SE Development Kit 11.0.1

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

Accept License Agreement　Decline License Agreement

Product / File Description	File Size	Download
Linux	147.4 MB	jdk-11.0.1_linux-x64_bin.deb
Linux	154.09 MB	jdk-11.0.1_linux-x64_bin.rpm
Linux	171.43 MB	jdk-11.0.1_linux-x64_bin.tar.gz
macOS	166.2 MB	jdk-11.0.1_osx-x64_bin.dmg
macOS	166.55 MB	jdk-11.0.1_osx-x64_bin.tar.gz
Solaris SPARC	186.8 MB	jdk-11.0.1_solaris-sparcv9_bin.tar.gz
Windows	150.98 MB	jdk-11.0.1_windows-x64_bin.exe
Windows	170.99 MB	jdk-11.0.1_windows-x64_bin.zip

图 3-6　操作系统选择界面

2）安装 JDK

安装 JDK 时，选择安装目录过程中会出现两次安装提示：第一次是安装 JDK；第二次是安装 JRE。无特殊要求时，安装位置按默认位置，一直单击“下一步”按钮，安装完毕后单击“关闭”按钮。

若要更改安装目录，建议 JDK 和 JRE 都安装在同一个 Java 文件夹中的不同子文件夹中（不能都安装在 Java 文件夹的根目录下，JDK 和 JRE 安装在同一文件夹中时会出错）。更改 JDK 安装路径如图 3-7 所示。

图 3-7　更改 JDK 安装路径

3）环境变量配置

安装完 JDK 后，然后配置环境变量。通过选择“计算机→系统属性→高级系统设置→高级→环境变量”命令，可打开环境变量配置界面，如图 3-8 所示。

创建系统变量 JAVA_HOME、CLASSPATH 并编辑修改系统变量 Path 的值，具体如图 3-9 所示。

(1) 创建 JAVA_HOME，值是刚刚安装 JDK 的目录，如 C:\Program Files\Java\jdk1.8.0_151。
(2) 创建 CLASSPATH，值是.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar（注意最前面有一点）。
(3) 编辑 Path，把值放到最前边%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;。

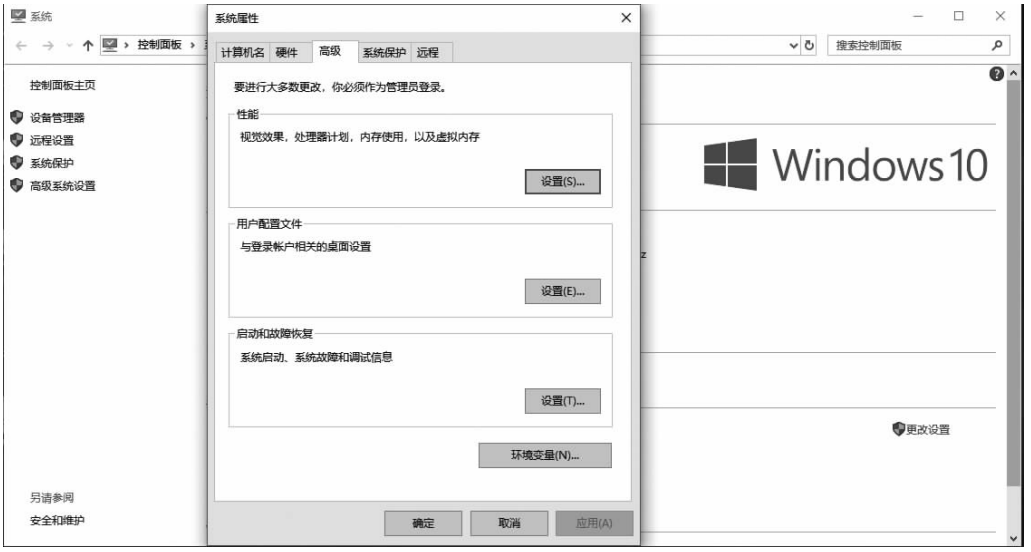


图 3-8　环境变量配置界面

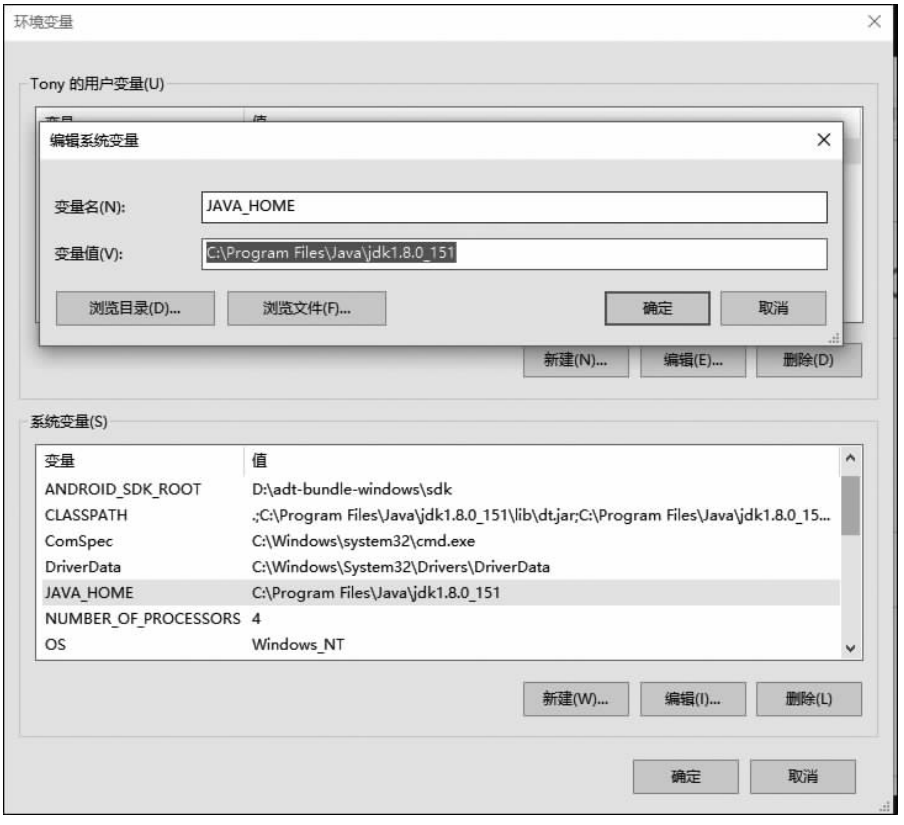


图 3-9　环境变量配置

4）检验安装及配置情况

运行 CMD(Win＋R 或右下角单击“开始”菜单的输入处)，在展开的命令行窗口中输入

两条命令校验！若展示如下，则说明配置成功；若没有展示，请检查前边的配置，具体如图 3-10 所示（图为 1.8 版本，请根据自己安装的版本检查）。

```
C:\>java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)

C:\>javac -version
javac 1.8.0_151
```

图 3-10　检验安装及配置

java -version 命令检查 Java 安装版本。

javac -version 命令检查 Javac 安装版本。

2. Android Studio 的安装与配置

1）下载 Android Studio

前往 Android Studio 中文社区（官网）https://developer.android.google.cn/studio/index.html，具体如图 3-11 所示。

图 3-11　Android Studio 下载网站

2）安装 Android Studio

打开步骤 1）下载完成的安装包。Android Studio 开始安装，如图 3-12 所示。Android Studio 安装界面如图 3-13 所示。

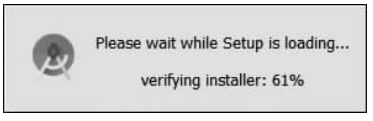


图 3-12　Android Studio 开始安装

选择需要安装的组件，如图 3-14 所示。Android Studio 主程序默认已勾选，Android Virtual Device（安卓虚拟设备）复选框，就是在计算机上虚拟出安卓手机的环境，让用户可以直接在计算机上运行开发出的 App，也将该复选框勾选上，然后单击 Next 按钮。

图 3-13　Android Studio 安装界面

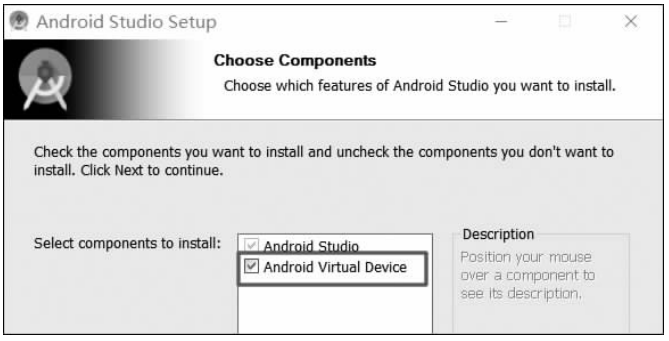


图 3-14　Android Studio 安装组件的选择

Android Studio 安装目录如图 3-15 所示。

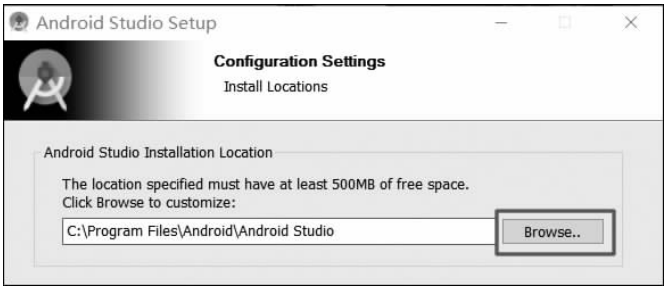


图 3-15　Android Studio 安装目录

直接单击 Install 进行安装(这里没有勾选 Do not create shortcuts 复选框,即不创建桌面快捷方式,如图 3-16 所示)。

Android Studio 安装等待如图 3-17 所示。

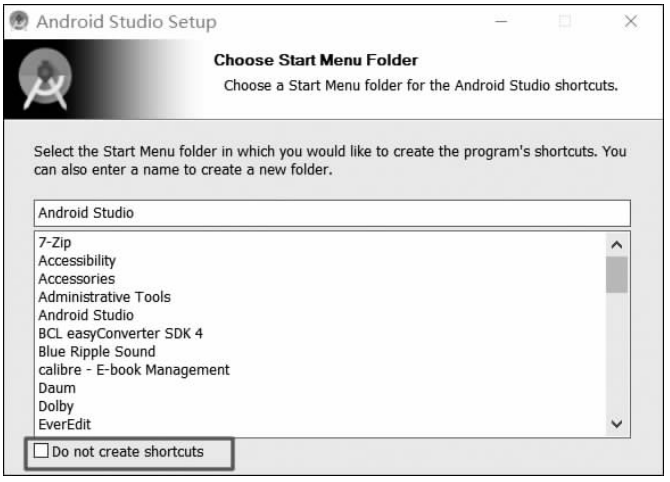


图 3-16　Android Studio 桌面快捷方式的选择

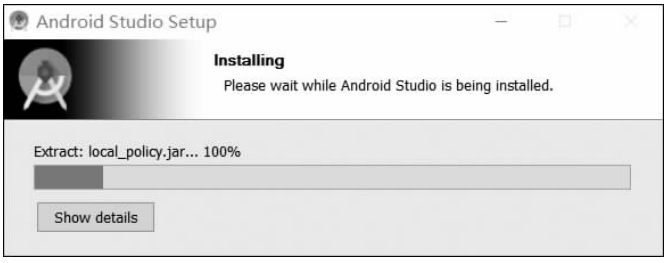


图 3-17　Android Studio 安装等待

Android Studio 安装完成如图 3-18 所示。

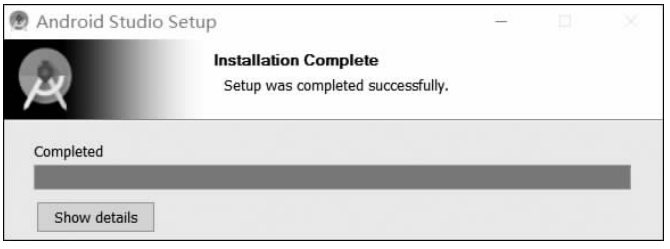


图 3-18　Android Studio 安装完成

之后单击 next 按钮安装完成。勾选 Start Android Studio，单击 Finish 按钮，就直接打开 Android Studio 了，具体操作如图 3-19 所示。

3）配置 SDK 路径

启动 Android Studio，单击 Configure 按钮，如图 3-20 所示。

单击进入 Project Defaults，具体如图 3-21 所示。

单击进入 Project Structure，具体如图 3-22 所示。

单击进入 Project Structure，设置已经安装好的 SDK，具体如图 3-23 所示。

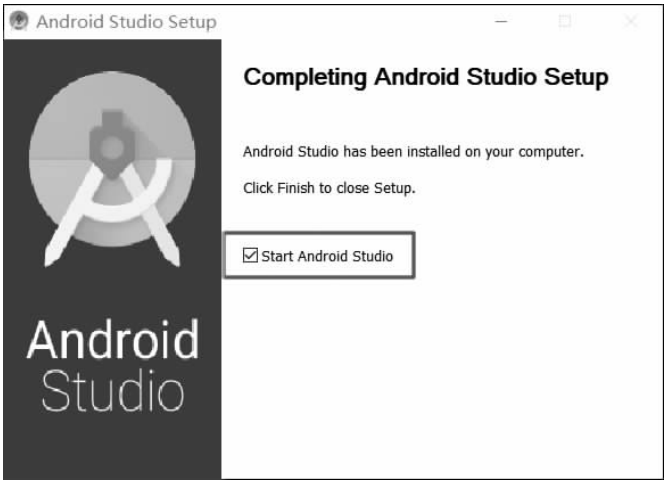


图 3-19　开始载入 Android Studio 主程序

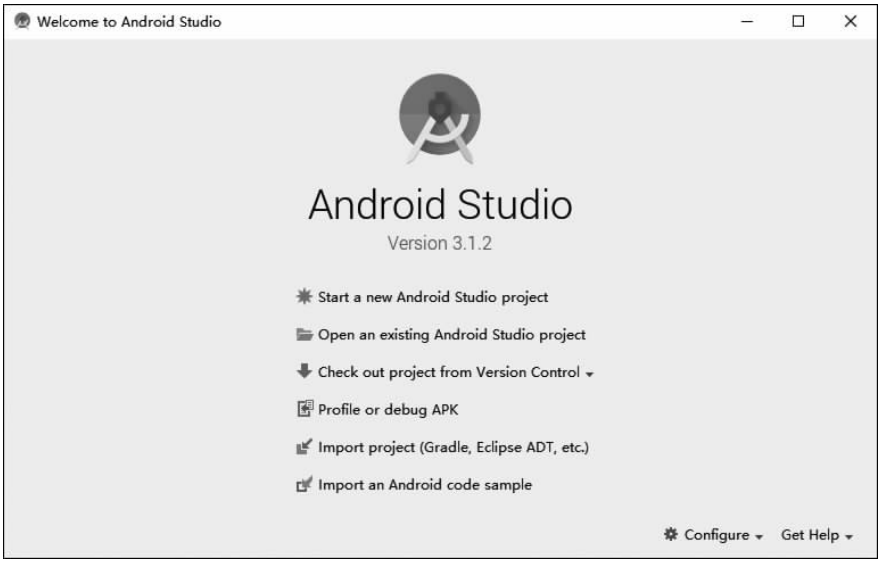


图 3-20　配置 SDK 路径

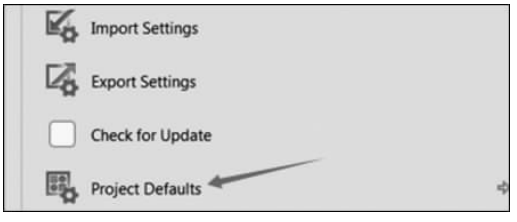


图 3-21　工程默认选项

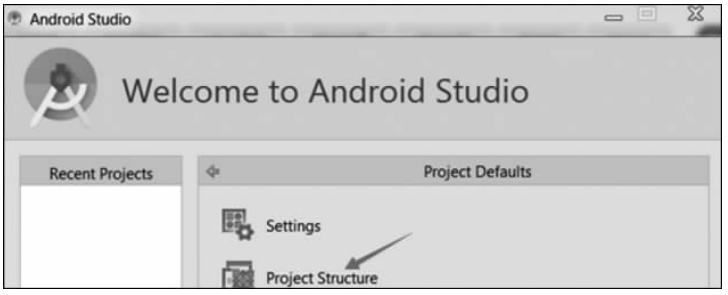


图 3-22　Android Studio 软件界面

图 3-23　本地 SDK 路径

3.2　Android 工程的创建与调试

3.2.1　Android 工程框架

1. Android Studio 工程项目目录

对于初学者来说，理解整个 Android 项目目录结构很重要，各自的作用，分别在什么时候用，哪个资源、哪个文件、哪个配置放在什么地方读者需要很明白，还要明白如何增加、删除、更新。图 3-24 显示了一个简单的目录结构。

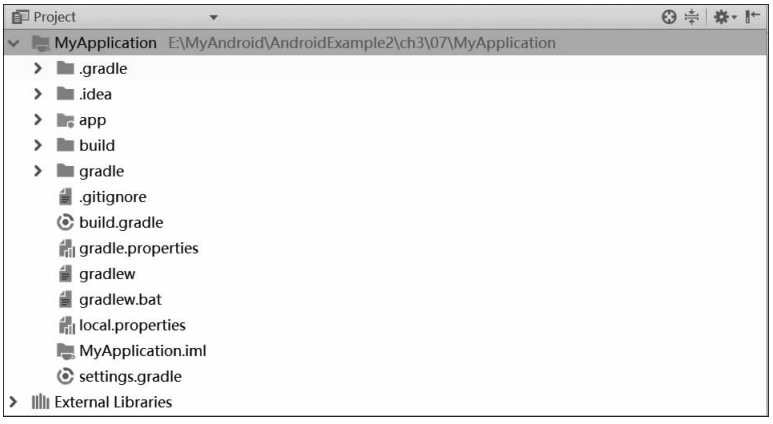


图 3-24　Android Studio 工程项目目录

1）.gradle 和.idea

.gradle 和.idea 两个目录下放置的都是 Android Studio 自动生成的一些文件，用户无须关心，也不要手动编辑。

2）app

项目中的代码、资源等内容几乎都放置在这个目录下，后面的开发工作也基本都是在这个目录下进行的，后面还会对这个目录单独展开讲解。

3）build

不必过多关心这个目录，它主要包含一些在编译时自动生成的文件。

4）gradle

gradle 目录下包含了 gradle wrapper 的配置文件，使用 gradle wrapper 的方式不需要提前将 gradle 下载好，而是会自动根据本地的缓存情况决定是否需要联网下载 gradle。Android Studio 默认没有启动 gradle wrapper 的方式，如果需要打开，可以单击 Android Studio 导航栏 → File → Settings → Build，Execution，Deployment → Gradle，进行配置更改。

5）.gitignore

.gitignore 文件用来将指定的目录或文件排除在版本控制外。

6）build.gradle

build.gradle 是项目全局的 gradle 构建脚本，通常这个文件中的内容是不需要修改的。后面会详细分析 gradle 构建脚本中的具体内容。

7）gradle.properties

gradle.properties 文件是全局的 gradle 配置文件，在这里配置的属性将会影响项目中所有的 gradle 编译脚本。

8）gradlew 和 gradlew.bat

gradlew 和 gradlew.bat 两个文件是用来在命令行界面中执行 gradle 命令的，其中 gradlew 在 Linux 或 Mac 系统中使用，gradlew.bat 在 Windows 系统中使用。

9）MyApplication.iml

iml 文件是所有 IntelliJ IDEA 项目都会自动生成的一个文件（Android Studio 是基于 IntelliJ IDEA 开发的），用于标识这是一个 IntelliJ IDEA 项目，用户不需要修改这个文件中的任何内容。

10）local.properties

local.properties 文件用于指定本机中的 Android SDK 路径，通常内容都是自动生成的，用户并不需要修改。除非你本机中的 Android SDK 位置发生了变化，那么就将这个文件中的路径改成新的位置。

11）settings.gradle

settings.gradle 文件用于指定项目中引入的所有模块。由于 MyApplication 项目中只有一个 App 模块，因此该文件中也就只引入了 App 这一个模块。通常情况下，模块的引入都是自动完成的，需要手动修改这个文件的场景可能比较少。

2. Android 应用解析

1）应用启动方式

（1）冷启动：当启动应用时，后台没有该应用的进程，这时系统会重新创建一个新的进程分配给该应用，这个启动方式就是冷启动。冷启动因为系统会重新创建一个新的进程分配给它，所以会先创建和初始化 Application 类，再创建和初始化 MainActivity 类（包括一系列的测量、布局、绘制），最后显示在界面上。

（2）热启动：当启动应用时，后台已有该应用的进程（例如，按 back 键、home 键，应用虽然会退出，但是该应用的进程依然会保留在后台，可进入任务列表查看），所以在已有进程的情况下，这种启动会从已有的进程中启动应用，这个方式叫热启动。热启动因为会从已有的进程中启动，所以热启动就不会走 Application 这一步了，而是直接走 MainActivity（包括一系列的测量、布局、绘制）。所以，热启动的过程只需要创建和初始化一个 MainActivity，不必创建和初始化 Application，因为一个应用从新进程的创建到进程的销毁，Application 只会初始化一次。

2）应用启动流程

Android Application 与其他移动平台有两个重大不同点。

每个 Android App 都在一个独立空间里，意味着其运行在一个单独的进程中，拥有自己的 VM，被系统分配一个唯一的 user ID。

Android App 由很多不同组件组成，这些组件还可以启动其他 App 的组件。因此，Android App 并没有一个类似程序入口的 main()方法。

Android Application 组件包括

（1）Activities：前台界面，直接面向 User，提供 UI 和操作。

（2）Services：后台任务。

（3）Broadcast Receivers：广播接收者。

（4）Context Providers：数据提供者。

Android 进程与 Linux 进程一样，默认情况下，每个 APK 都运行在自己的 Linux 进程中。另外，默认一个进程里面只有一个线程——主线程。这个主线程中有一个 Looper 实例，通过调用 Looper. loop()从 Message 队列里取出 Message 进行相应的处理。

具体的启动过程分为以下 3 步。

（1）创建进程。

ActivityManagerService 调用 startProcessLocked()方法创建新的进程，该方法会通过 socket 通道传递参数给 Zygote 进程。Zygote 孵化自身，调用 ZygoteInit. main()方法实例化 ActivityThread 对象并最终返回新进程的 pid。ActivityThread 随后依次调用 Looper. prepareLoop()和 Looper. loop()开启消息循环。

（2）绑定 Application。

接下来要做的就是将进程和指定的 Application 绑定起来。这是通过上一步的 ActivityThread 对象中调用 bindApplication()方法完成的。该方法发送一个 BIND_APPLICATION 的消息到消息队列中，最终通过 handleBindApplication()方法处理该消息。然后调用 makeApplication()方法加载 App 的 classes 到内存中。

(3) 启动 Activity。

经过前两个步骤后,系统已经拥有了该 Application 的进程。后面的调用顺序就是普通的从一个已经存在的进程中启动一个新进程的 Activity 了。

实际调用方法是 realStartActivity( ),它会调用 Application 线程对象中的 sheduleLaunchActivity( ) 发送一个 LAUNCH_ACTIVITY 消息到消息队列中,通过 handleLaunchActivity()处理该消息。

3) App 的启动优化

基于上面的启动流程,尽量做到如下 3 点。

- 在 Application 的创建过程中尽量少进行耗时操作。
- 如果用到 SharePreference,尽量在异步线程中操作。
- 减少布局的层次,并且在生命周期回调的方法中尽量减少耗时操作。

3.2.2 Android 工程创建

通过前面的学习,我们对 Android Studio 开发环境已经非常熟悉了。下面创建一个 Android 工程,巩固前面的学习成果。

(1) 选择 File->New Project 命令,出现如图 3-25 所示的对话框。

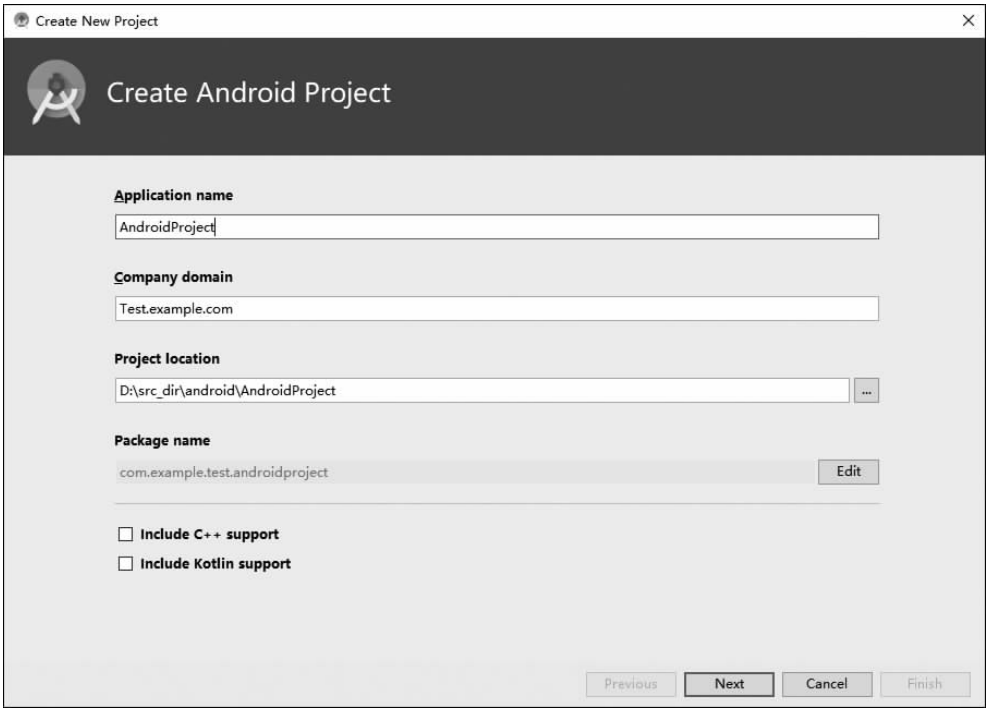


图 3-25 创建 Android 新工程

(2) 选择 Login Activity 例子,如图 3-26 所示。

(3) 完成创建工程。

单击 Next 按钮后,在弹出的向导页中单击 Finish 按钮,完成工程的创建,如图 3-27 所示。

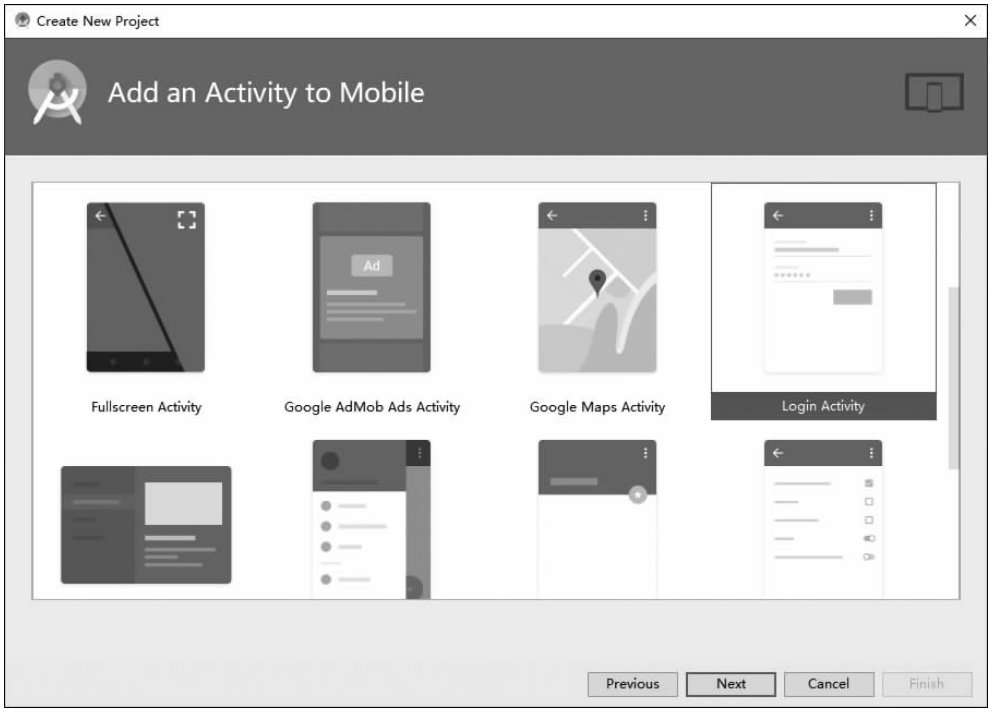


图 3-26　选择 Login Activity 例子

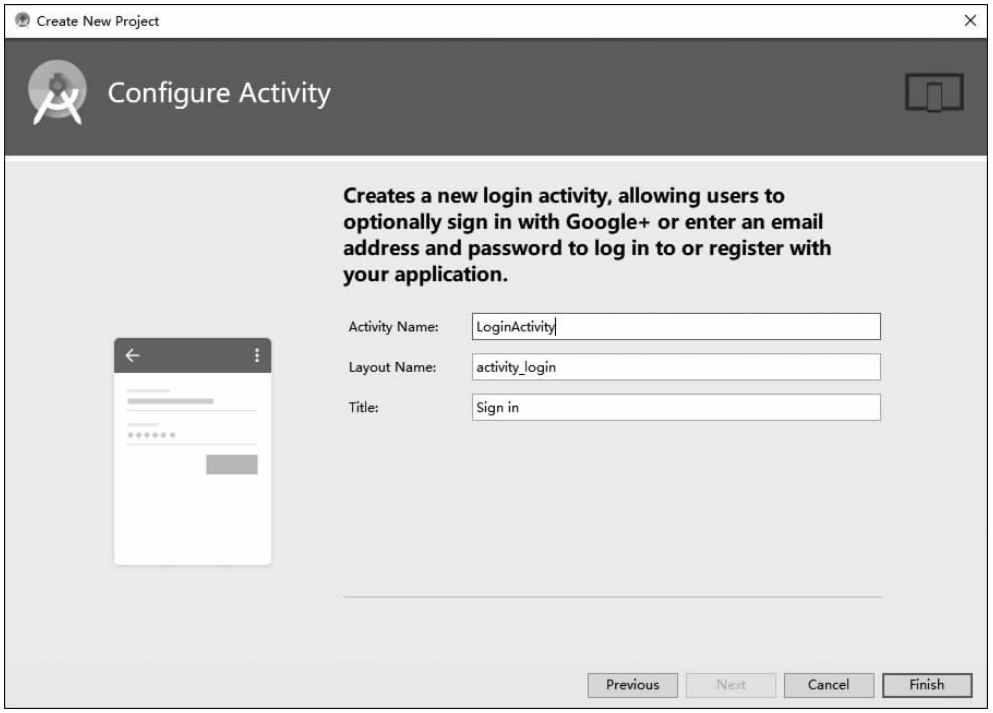


图 3-27　完成工程的创建

3.2.3 Android 工程调试

写代码不可避免有缺陷(Bug),通常情况下除了日志最直接的调试手段就是 debug;当程序出现缺陷时,调试可以快速找到缺陷。进入调试状态,我们可以清楚地了解程序的整个执行过程,可以对内存的数据进行监视。下面简单总结一下调试的基本使用和一些调试的技巧。

1. 进入调试

1) 插入断点

选定要设置断点的代码行,在行号的区域后单击鼠标左键即可,如图 3-28 所示。

```
65      @Override
66      protected void onCreate(Bundle savedInstanceState) {
67          super.onCreate(savedInstanceState);
68          setContentView(R.layout.activity_login);
69          // Set up the login form.
70          mEmailView = (AutoCompleteTextView) findViewById(R.id.email);
71          populateAutoComplete();
72
73          mPasswordView = (EditText) findViewById(R.id.password);
74          mPasswordView.setOnEditorActionListener(new TextView.OnEditorActionListener() {
75              @Override
76              public boolean onEditorAction(TextView textView, int id, KeyEvent keyEvent) {
77                  if (id == EditorInfo.IME_ACTION_DONE || id == EditorInfo.IME_NULL) {
78                      attemptLogin();
79                      return true;
80                  }
81                  return false;
82              }
83          });
```

图 3-28 断点设置

2) 进入调试状态

设置好断点后,单击工具栏中的小臭虫(Debug)进入调试状态,如图 3-29 所示。

图 3-29 进入调试状态

当一个应用进入调试状态后,Android Studio IDE 会在软件界面下方显示 Debug 视图,即调试者状态。在这里可以对程序进行监视和调试。调试状态界面如图 3-30 所示。

IDE 下方出现 Debug 视图,左边区域中显示了程序执行到断点处调用过的所用方法,越下面的方法被调用的越早;在右边区域可以给指定的变量赋值(鼠标左键选择变量,在单击右键弹出的菜单中选择 setValue...)。这个功能可以更加快速地检测条件语句和循环语句。在右边区域中可以对某一个特定的变量进行监视。

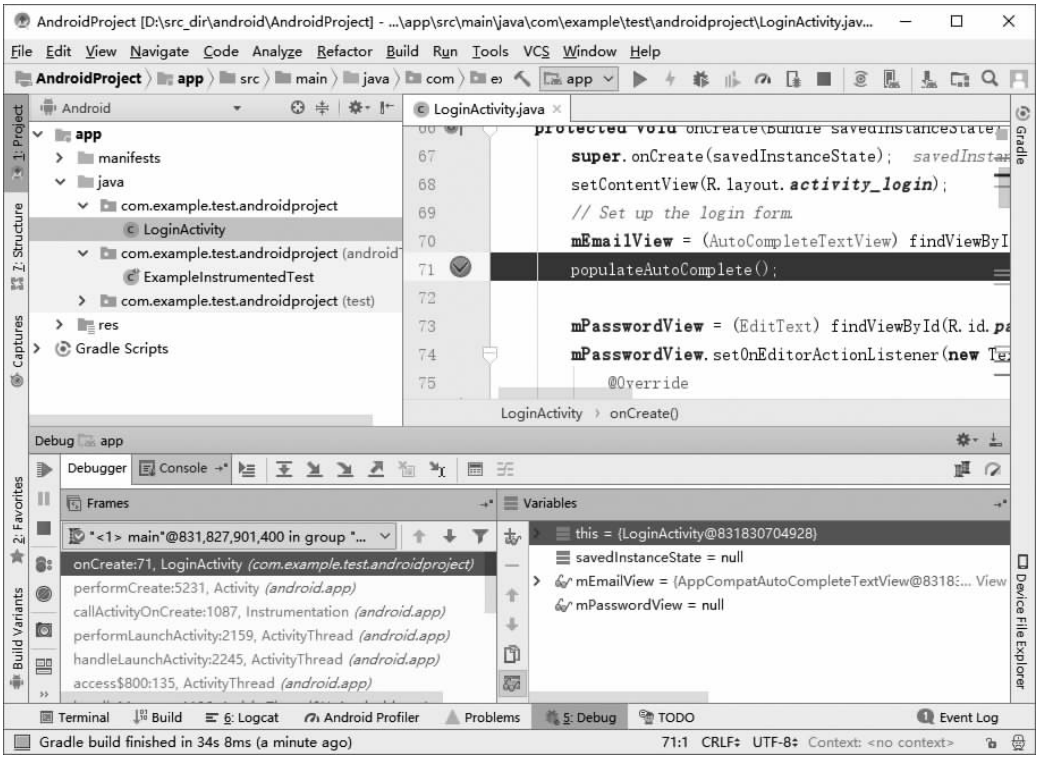


图 3-30　调试状态界面

2. 常用的调试方式和快捷键

1）常用的调试功能及快捷键

- Step Into（F7）：进入子函数。
- Step Over（F8）：越过子函数，但子函数会执行。
- Step Out（Shift ＋ F8）：跳出子函数。
- Run to Cursor（Alt ＋ F9）：运行到光标所在的位置。
- Show Execution Point（Alt ＋ F10）：快速定位当前调试的位置，并将该行高亮地显示出来。

2）调试功能解释

- Step Into：单步执行，遇到子函数就进入并且继续单步执行。例如，当执行到 System. out. println("XXXX")时，使用这个功能就会进入 System. out. println 方法所在类的 println 方法下（当然，这样做是没有必要的，如果进入后想跳出，执行 step out 命令就可以了）。
- Step Over：单步执行时，在函数内遇到子函数时不会进入子函数内单步执行，而是将子函数整个执行完再停止，也就是把子函数整个作为一步。例如，上面的例子中，System. out. println("XXXX")执行完后是跳到下一个语句中，而不会跳进去，这个功能也是比较常用的，一直按 F8 键就可以了。

- Step Out：单步执行到子函数内时，用 Step Out 就可以执行完子函数的余下部分，并返回到上一层函数。
- Run to Cursor：运行到光标所在的位置，执行该功能后，不论执行到哪里，程序都可以执行到光标所在行下。
- Show Execution Point：当不知道程序当前已经执行到哪里时，可以使用这个功能，Android Studio 会跳到执行所在的界面，并将该行高亮地显示出来。

3.2.4 Android 生命周期

1. 程序的生命周期

程序的生命周期是指在 Android 系统中进程从启动到终止的所有阶段，也就是 Android 程序启动到停止的全过程。程序的生命周期由 Android 系统进行调度和控制。

Android 系统中的进程分为：前台进程、可见进程、服务进程、后台进程、空进程。

Android 系统中的进程优先级由高到低，如图 3-31 所示。

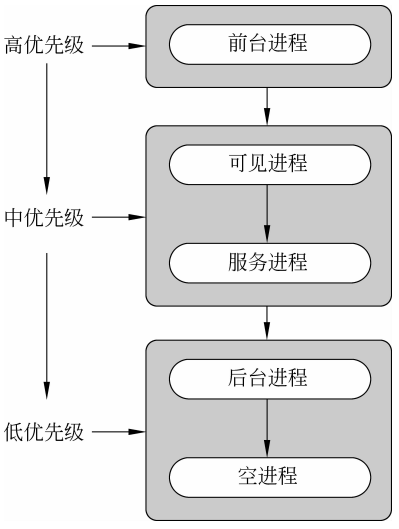


图 3-31 Android 系统的进程及优先级

1）前台进程

前台进程是 Android 系统中最重要的进程，是指与用户正在交互的进程，包含以下 4 种情况。

- 进程中的 Activity 正在与用户进行交互。
- 进程服务被 Activity 调用，而且这个 Activity 正在与用户进行交互。
- 进程服务正在执行声明周期中的回调方法，如 onCreate()、onStart()或 onDestroy()。
- 进程的 BroadcastReceiver 正在执行 onReceive()方法。

Android 系统在多个前台进程同时运行时，可能会出现资源不足的情况，此时会清除部分前台进程，保证主要的用户界面能够及时响应。

2）可见进程

可见进程指部分程序界面能够被用户看见，但不在前台与用户交互，不响应界面事件的进程。如果一个进程包含服务，且这个服务正在被用户可见的 Activity 调用，此进程同样被视为可见进程。

Android 系统一般存在少量的可见进程，只有在特殊的情况下，Android 系统才会为保证前台进程的资源而清除可见进程。

3）服务进程

服务进程是指包含已启动服务的进程，通常有如下特点：

- 没有用户界面。

- 在后台长期运行。

Android 系统在不能保证前台进程或可视进程运行所必要的资源情况下，才会强行清除服务进程。

4）后台进程

后台进程是指不包含任何已经启动的服务，而且没有任何用户可见的 Activity 的进程。

Android 系统中一般存在数量较多的后台进程，在系统资源紧张时，系统将优先清除用户较长时间没有见到的后台进程。

5）空进程

空进程是指不包含任何活跃组件的进程。空进程在系统资源紧张时会被首先清除，但为了提高 Android 系统应用程序的启动速度，Android 系统会将空进程保存在系统内存，用户重新启动该程序时，空进程会被重新使用。

除以上优先级外，以下两方面也决定它们的优先级。

- 进程的优先级取决于所有组件中的优先级最高的部分。
- 进程的优先级会根据与其他进程的依赖关系而变化。

2. Activity 生命周期

1）Activity 状态

在 Activity 生命周期中，其表现状态有 4 种，分别是：Active（活动状态）、Pause（暂停状态）、Stop（停止状态）和 Unactive（非活动状态）。

- Active（活动状态）：是指 Activity 通过 onCreate 被创建，Activity 在用户界面中处于最上层，完全能让用户看到，能够与用户进行交互。
- Pause（暂停状态）：是指当一个 Activity 失去焦点，该 Activity 将进入 pause 状态，Activity 在界面上被部分遮挡，该 Activity 不再处于用户界面的最上层，且不能与用户进行交互，系统在内存不足时会将其终止。
- Stop（停止状态）：是指当一个 Activity 被另一个 Activity 覆盖，该 Activity 将进入 Stop 状态，Activity 在界面上完全不能被用户看到。也就是说，这个 Activity 被其他 Activity 全部遮挡，系统在需要内存的时候会将其终止。
- Unactive（非活动状态）：不在以上 3 种状态中的 Activity 则处于非活动状态。

当 Activity 处于 Pause 或者 Stop 状态时，都可能被系统终止并回收。因此，有必要在 onPause 和 onStop 方法中将应用程序运行过程中的一些状态（如用户输入等），保存到持久存储中。如果程序中启动了其他后台线程，也需要注意在这些方法中进行一些处理，例如，在线程中打开了一个进度条对话框，如果不在 Pause 或 Stop 中取消掉线程，则当线程运行完取消掉对话框时，就会抛出异常，如图 3-32 所示。

在 Activity 生命周期中，其事件的回调方法有 7 个，Activity 状态保存/恢复的事件回调方法有两个，如下所示。

Activity 生命周期的事件回调方法见表 3-1。