Android 入门到精通详解

目　录

第一篇 Android 系统结构和 SDK 使用

第 1 章　Android 的系统介绍

1.1 系统介绍

Android 是 Google 开发的基于 Linux 平台的、开源的、智能手机操作系统。Android 包括操作系统、中间件和应用程序，由于源代码开放，Android 可以被移植到不同的硬件平台上。

OHA（Open Handset Alliance，开放手机联盟)，为 Google 与 33 家公司联手为 Android 移动平台系统的发展而组建的一个组织。

HTC 和 Google 合作推出了几款手机：G1、G2、Hero 和 Nexus One，其他的手机厂商也推出了几款 Android 手机，如下图所示：

图 G1、G2、Hero 和 Nexus One 手机

围绕在 Google 的 Android 系统中，形成了移植开发和上层应用程序开发两个不同的开发方面。手机厂商从事移植开发工作，上层的应用程序开发可以由任何单位和个人完成，开发的过程可以基于真实的硬件系统，还可以基于仿真器环境。

Android 1.5 以前的仿真环境，Android 1.6 以后的仿真器环境如下所示：

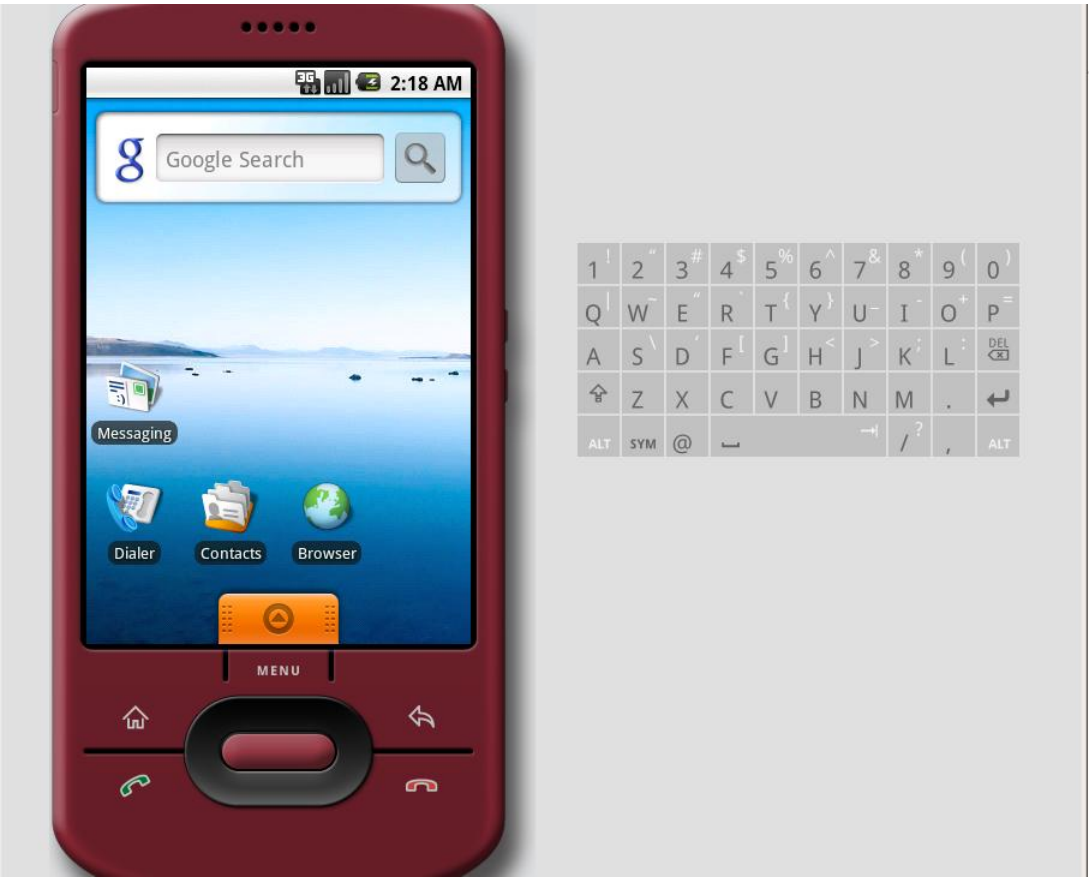


图 Android 1.5 以前的仿真器环境

图 Android 1.6 以后的仿真器环境

作为一个手机平台，Android 在技术上的优势主要有以下几点：

- 全开放智能手机平台
- 多硬件平台的支持
- 使用众多的标准化技术
- 核心技术完整，统一
- 完善的 SDK 和文档
- 完善的辅助开发工具

Android 的开发者可以在完备的开发环境中进行开发，Android 的官方网站也提供了丰富的文档、资料。这样有利于 Android 系统的开发和运行在一个良好的生态环境中。

1.2 软件结构和使用的工具

从宏观的角度来看，Android 是一个开放的软件系统，它包含了众多的源代码。从下至上，Android 系统分成 4 个层次：

- 第 1 层次：Linux 操作系统及驱动；
- 第 2 层次：本地代码（C/C++）框架；
- 第 3 层次：Java 框架；
- 第 4 层次：Java 应用程序。

Android 系统的架构如图所示：

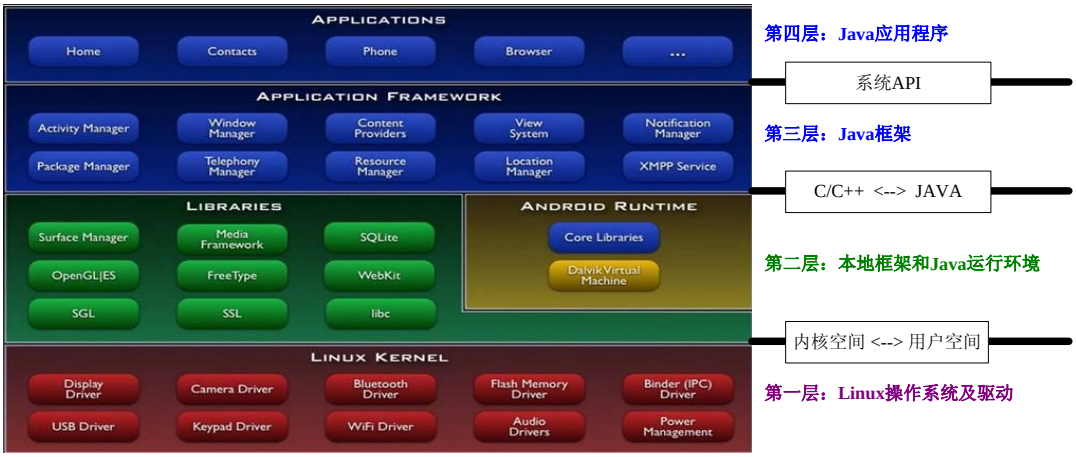


图 Android 系统的架构

Android 的第 1 层次由 C 语言实现，第 2 层次由 C 和/C++实现，第 3、4 层次主要由 Java 代码实现。

第 1 层次和第 2 层次之间，从 Linux 操作系统的角度来来看，是内核空间与用户空间的分界线，第 1 层次运行于内核空间，第 2、3、4 层次运行于用户空间。

第 2 层次和第 3 层次之间，是本地代码层和 Java 代码层的接口。

第 3 层次和第 4 层次之间，是 Android 的系统 API 的接口，对于 Android 应用程序的开发，第 3 层次以下的内容是不可见的，仅考虑系统 API 即可。

由于 Android 系统需要支持 Java 代码的运行，这部分内容是 Android 的运行环境（Runtime），由虚拟机和 Java 基本类组成。

对于 Android 应用程序的开发，主要关注第 3 层次和第 4 层次之间的接口。

除了软件本身的代码之外，Android 还提供了一系列工具来辅助系统开发，这些主要的工具包括：

- aapt（Android Asset Packaging Tool）：用于建立 zip 兼容的包（zip、jar、apk），也可用于将资源编译到二进制的 assets。
- adb（Android Debug Bridge，Android 调试桥）：使用 adb 工具可以在模拟器或设备上安装应用程序的.apk 文件，并从命令行访问模拟器或设备。也可以用它把 Android 模拟器或设备上的应用程序代码和一个标准的调试器连接在一起。
- android 工具：android 工具是一个脚本，用于创建和管理 Android Virtual Devices （AVDs）。
- AIDL 工具（Android Interface Description Language，Android 接口描述语言工具），AIDL 工具可以生成进程间接口的代码，诸如 Service 可能使用的接口。
- AVDs（Android Virtual Devices，Android 虚拟设备）
- 用于配置模拟器，模拟出类似的设备效果
- DDMS（Dalvik Debug Monitor Service，Dalvik 调试监视器服务）：这个工具集成了 Dalvik，能够在模拟器或者设备上管理进程并协助调试。可以使用它杀死进程，选择某个特定的进程来调试，产生跟踪数据，观察堆（heap）和线程信息，截取模拟器或设备的屏幕画面，还有更多的功能。
- dx：dx 工具用于将.class 字节码（bytecode）转换为 Android 字节码（保存在.dex 文件中）这个字节码文件是给 Android 的 Java 虚拟机运行用的。
- Draw 9-patch：Draw 9-patch 工具允许使用所见即所得（WYSIWYG）的编辑器轻松地创建 NinePatch 图形。
- Emulator（模拟器）：模拟器是一个运行于主机上的程序，可以使用模拟器来模拟一个实际的 Android 系统的运行，使用模拟器非常适合调试和测试应用程序。
- Hierarchy Viewer（层级观察器）：层级观察器工具允许调试和优化用户界面。它用可视的方法把视图（view）的布局层次展现出来，此外，还给当前界面提供了一个具有像素栅格（grid）的放大镜观察器。
- mksdcard：帮助创建磁盘映像（disk image），可以在模拟器环境下使用磁盘

映像来模拟外部存储卡（例如 SD 卡）。

- Monkey：Monkey 是在模拟器或设备上运行的一个小程序，它能够产生随机的用户事件流，例如：点击（click）、触摸（touch）、挥手（gestures），还包括一系列系统级事件。可以使用 Monkey 给正在开发的程序做随机的但可重复的压力测试。
- sqlite3：sqlite3 工具能够方便地访问 SQLite 数据文件，这是一个 sqlite 标准命令行工具。
- Traceview：这个工具可以将 Android 应用程序产生的跟踪日志（trace log）转换为图形化的分析视图。

第 2 章 Android SDK 的开发环境

2.1 Android SDK 的结构
2.2 Android SDK 的环境安装
2.3 Android 中运行仿真器环境
2.4 Android 中建立中程

Android 的 SDK 开发环境使用预编译的内核和文件系统，屏蔽了 Android 软件架构第三层及以下的内容，开发者可以基于 Android 的系统 API 配合进行应用程序层次的开发。在 SDK 的开发环境中，还可以使用 Eclipse 等作为 IDE 开发环境。

2.1 Android SDK 的结构

Android SDK 在 IDE 环境中使用的组织结构如图所示：

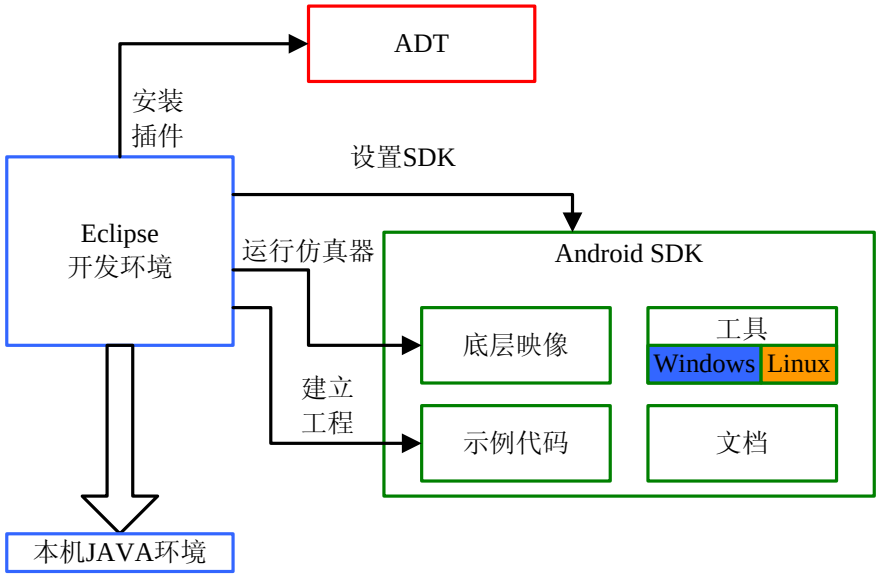


图 Android 系统的 IDE 开发环境

Android 提供的 SDK 有 Windows 和 Linux(其区别主要是 SDK 中工具不同)，在 Android 开发者的网站上可以直接下载各个版本的 SDK。

Android 的 SDK 命名规则为：

android-sdk-{主机系统}_{体系结构}_{版本}

例如，Android 提供 SDK 的几个文件包如下所示：

- android-sdk-windows-1.5_r2.zip
- android-sdk-linux_x86-1.5_r2.zip
- android-sdk-windows-1.6_r1.zip
- android-sdk-linux_x86-1.6_r1.zip

SDK 的目录结构如下所示：

- add-ons：附加的包；
- docs：HTML 格式的离线文档；
- platforms：SDK 核心内容；
- tools：工具。

在 platforms 中包含了的各个 Android SDK 版本的目录中，包含系统映像，工具、示例代码等内容。

- data/ ：包含默认的字体、资源等内容；
- images/ ：包含默认的 Android 磁盘映像，包括了系统映像（Android system image），默认的用户数据映像（userdata image）,默认的内存盘映像（ramdisk image）等等，这些映像是仿真器运行的时候需要使用的；
- samples/：包含一系列的应用程序，可以在 Android 的开发环境中，根据它们建立工程，编译并在仿真器上运行；
- skins/：包含了几个仿真器的皮肤，每个皮肤对应了一种屏幕尺寸；
- templates/：包含了几个用 SDK 开发工具的模板；
- tools/：特定平台的工具； Any development tools that are specific to the platform version.
- android.jar：Android 库文件的 JAVA 程序包，在编译本平台的 Android 应用程序的时候被使用。

不同保本的 API 对应着不同的 API 级别，Android 已经发布，并且属于正式支持的各个版本的 SDK 如下所示：

Android 的发布版本	API 级别
Android 1.1	2
Android 1.5	3
Android 1.6	4
Android 2.0	5
Android 2.0.1	6
Android 2.1	7

Android 的 SDK 需要配合 ADT 使用，ADT（Android Development Tools）是 Eclipse 集成环境的一个插件。通过扩展 Eclipse 集成环境功能，使得生成和调试 Android 应用程序既容易又快速。

2.2 Android SDK 环境安装

Android 的 SDK Windows 版本需要以下的内容：

- JDK 1.5 或者 JDK 1.6
- Eclipse 集成开发环境
- ADT（Android Development Tools）插件
- Android SDK

其中 ADT 和 Android SDK 可以到 Android 开发者的网站去下载，或者在线安装亦可，ADT 的功能如下所示：

- 可以从 Eclipse IDE 内部访问其他的 Android 开发工具。例如，ADT 可以让你直接从 Eclipse 访问 DDMS 工具的很多功能——屏幕截图、管理端口转发（port-forwarding）、设置断点，观察线程和进程信息。
- 提供了一个新的项目向导（New Project Wizard），帮助你快速生成和建立起新 Android 应用程序所需的最基本文件
- 使构建 Android 应用程序的过程变得自动化，以及简单易行。
- 提供了一个 Android 代码编辑器，可以帮助你为 Android manifest 和资源文件编写有效的 XML

在 Eclipse 环境中使用 Android SDK 的步骤如下所示：

2.2.1. 安装 JDK 基本 Java 环境。

Eclipse 的运行需要依赖 JDK，因此需要下载使用 JDK 的包，并进行安装。

JDK 1.6 版本其文件为 jdk-6u10-rc2-bin-b32-windows-i586-p-12_sep_2008.exe，点击直接进行安装即可。

2.2.2. 安装 Eclipse

Eclipse 集成开发环境是开放的软件，可以到 Eclipse 的网站上去下载：

http://www.eclipse.org/downloads/

Eclipse 包含了以下的几个版本

- Eclipse 3.3（Europa）
- Eclipse 3.4（Ganymede）

- Eclipse 3.5（Galileo）

在 Android 的开发中，推荐使用 Eclipse 3.4 和 Eclipse 3.5，Eclipse 3.3 虽然也可以使用，但是没有得到 Android 官方的验证。

如果使用 Eclipse 3.4，可以去下载 eclipse-SDK-3.4-win32.zip 包；如果使用 Eclipse 3.5，可以去下载 eclipse-SDK-3.5.1-win32.zip 包。这个包不需要安装，直接解压缩即可，解压缩后执行其中的 eclipse.exe 文件。

2.2.3. 获得 Android SDK

Android 的 SDK 是一个比较庞大的部分，包含了 Android 系统的二进制内容、工具和文档等。得到 Android SDK，可能使用到两种方式：

- 下载 Android SDK 的包（Archives）
- 通过软件升级的方式（Setup）

下载 Android SDK 的包：对于 Android SDK 1.6 之前的版本，包括 Android SDK 1.1，Android SDK 1.5，Android SDK 1.6 可以直接从 Android 开发者中下载得到，每个 SDK 包含 Linux、Windows 和 MAC 三个版本。在 Windows 环境中，使用 Windows 的版本，例如：android-sdk-windows-1.5_r2.zip，android-sdk-windows-1.6_r1.zip，这个包通常用几百 M 的大小。

以这种方式下载的 Android SDK，不需要安装，直接解压缩即可。

目前 Android 系统推荐使用的方式软件升级获得 Android 包：

- 第一步：获得 android-sdk_r04-windows.zip

从 Android 开发者上，获取 Android SDK 的相关包 android-sdk_r04-windows.zip，这个包比实际的 Android 的 SDK 要小得多，只有 20 多 M，其中包含了一个 Setup 可执行程序，获取完整的 SDK 是通过这个可执行程序获得的。解压缩这个包，获得 Android SDK 的基本目录结构，但是其中还没有实际的内容。

- 第二步：运行 SDK Setup.exe 程序，下载实际的 Android SDK

运行程序，Android SDK 的，出现 SDK 的下载界面：

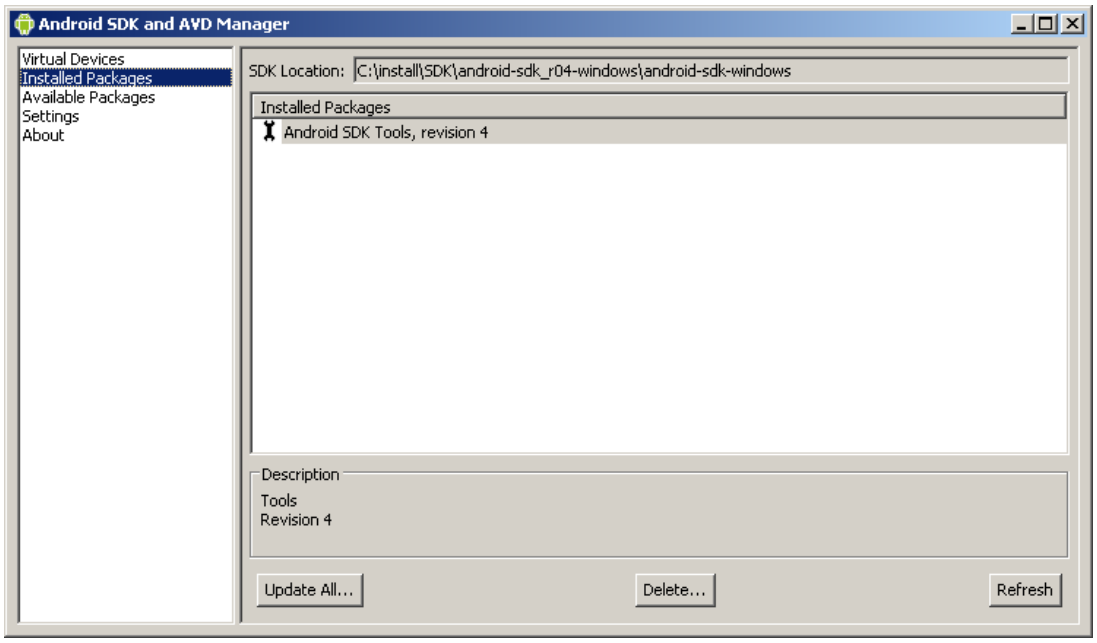


图 Android SDK 的安装界面

在 Settings 中进行设置，选中 Force ……项，并且选择保存（Save and Apply）。

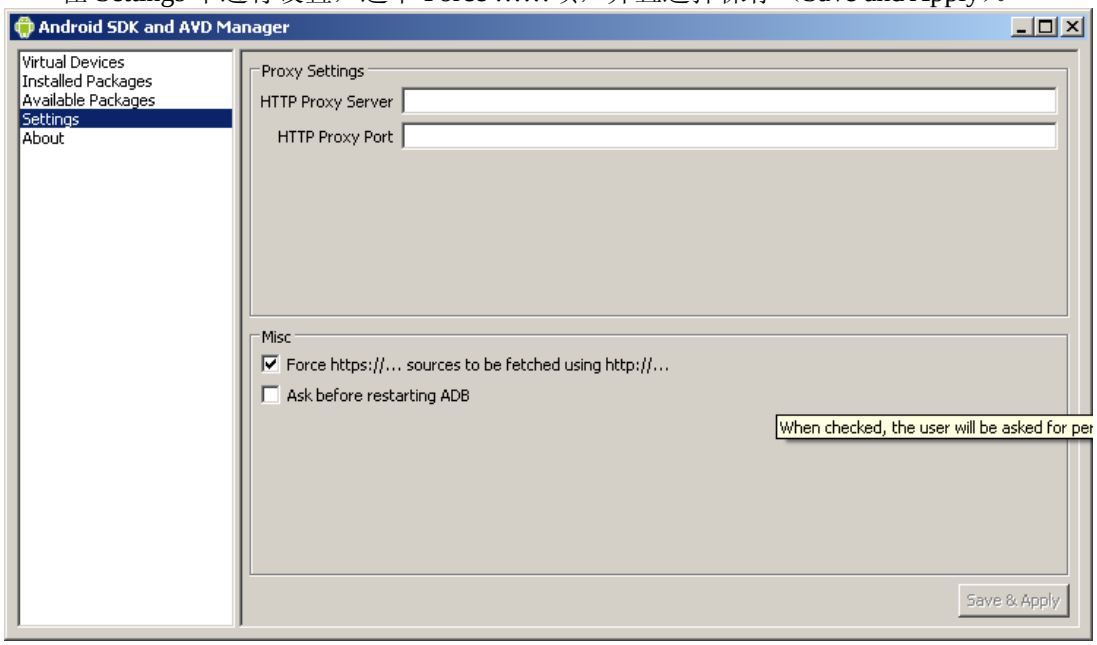


图 设置安装路径

回到 Installed Packages 中，进行安装，出现 Android 的各个版本的 SDK、工具、文档的安装界面，如下所示：

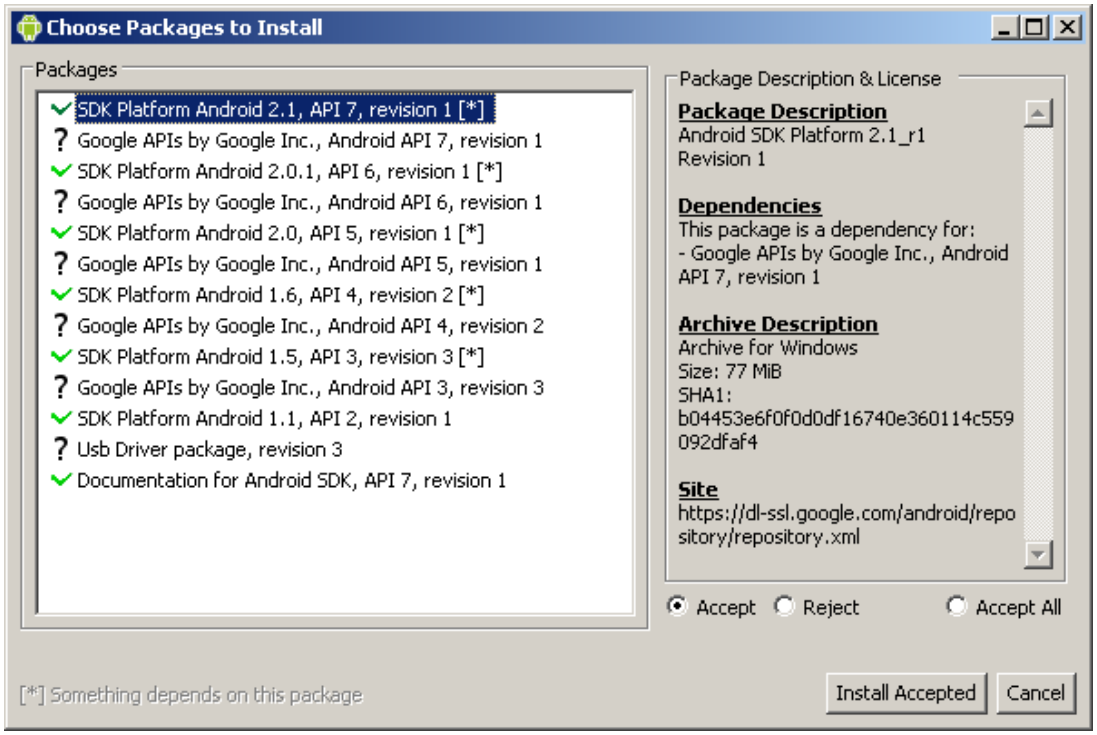


图 选择要安装的组件

每个组件可以选择，接受（Accept）表示安装，拒绝（Reject）表示不安装，接受全部（Accept All）表示安装所有的内容。文档一般安装成最新的版本。

选择后，安装程序将依次安装各个组件。

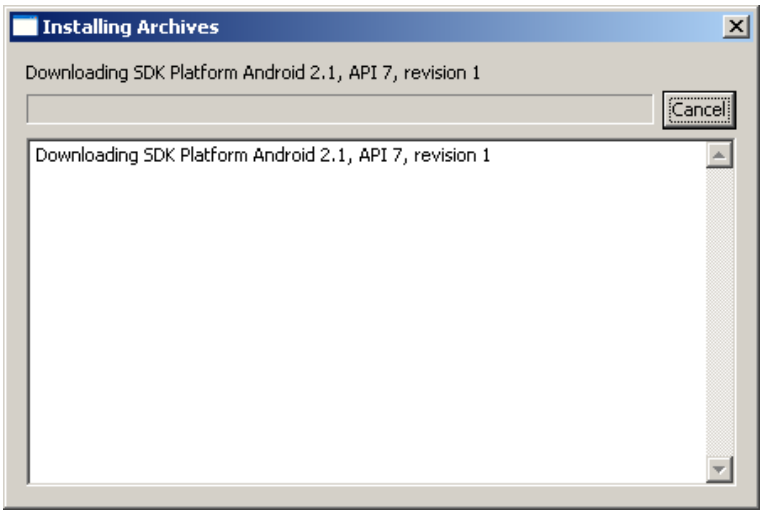


图 安装界面

下载过程中，每个组件将首先被放置到 temp 中，以一个 zip 包的形式存在。下载完成后，得到完整的 Android SDK。

2.2.4（1）．在 Eclipse 3.4（Ganymede）中安装 ADT

第一步：启动 Eclipse 选择“Help”>“Software Updates...”准备安装插件。

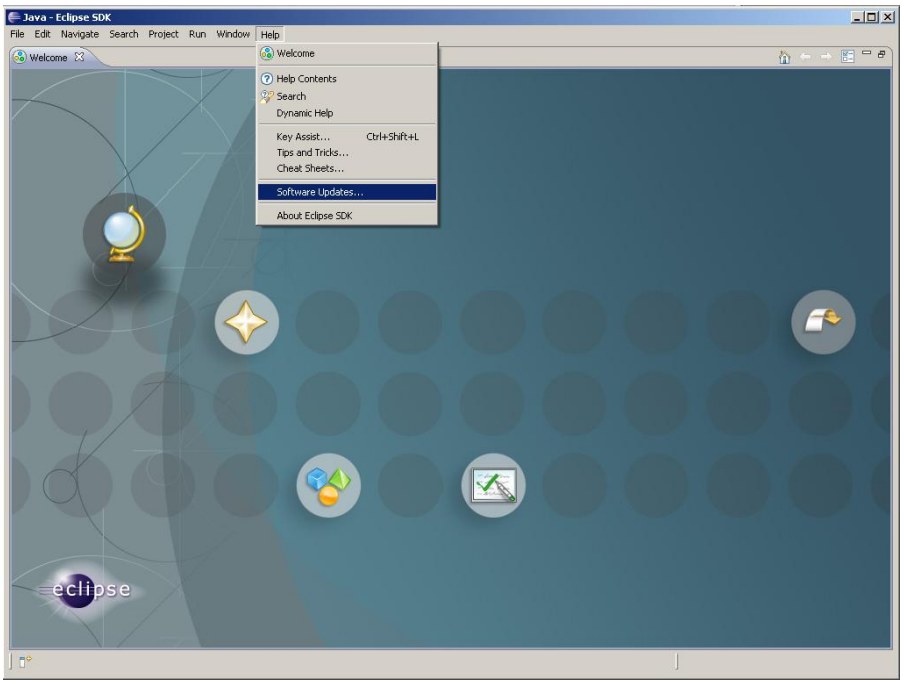


图 Eclipse 3.4 中选择软件升级

第二步：在打开的对话框中点击“Available Software”，出现 Eclipse 的现有软件对话框。

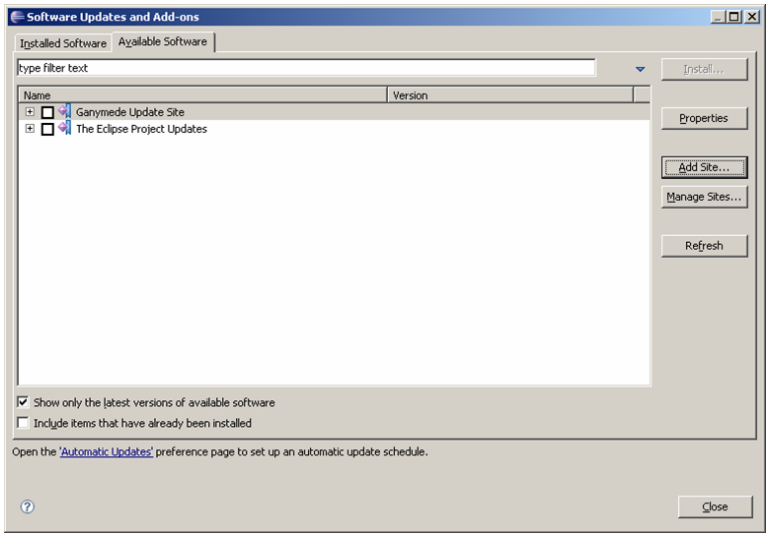


图 Eclipse 3.4 中选择要安装的插件

点击右侧自上而下的第 3 个按钮，“Add Site...”准备增加插件。

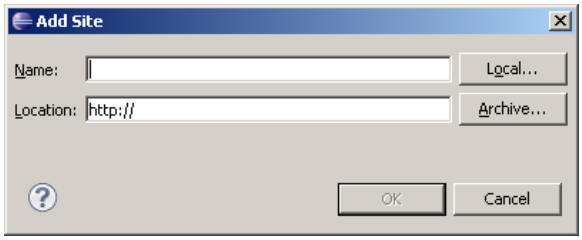


图 增加 ADT 的路径

在“Add Site”对话框中，输入 Android 插件的路径：

https://dl-ssl.google.com/android/eclipse/

另外的一种方式是点击 Archive…按钮，这样可以不使用网络，直接指定磁盘中的 ADT 包（目前最新的版本是 ADT-0.9.5.zip）。

第三步：回到安装对话框，可以看到 plugin 的 URL 下面有“Developer Tools”。选择到“Developer Tools”中，

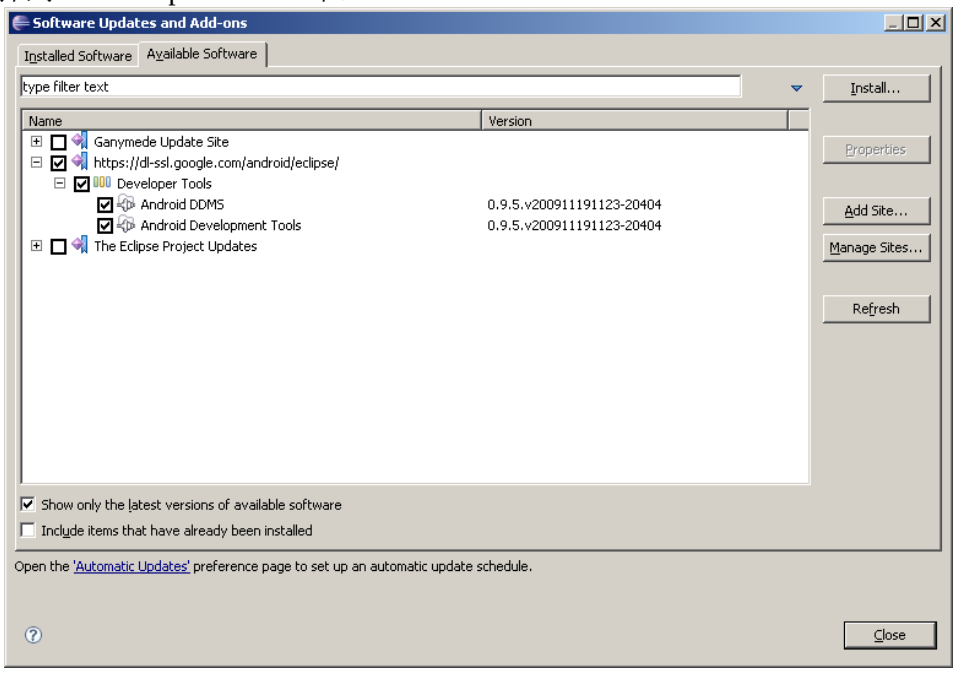


图 Eclipse 3.4 中选择安装 Android 的 DDMS 和 ADT

然后点击“Install...”按钮，继续运行，如图所示：

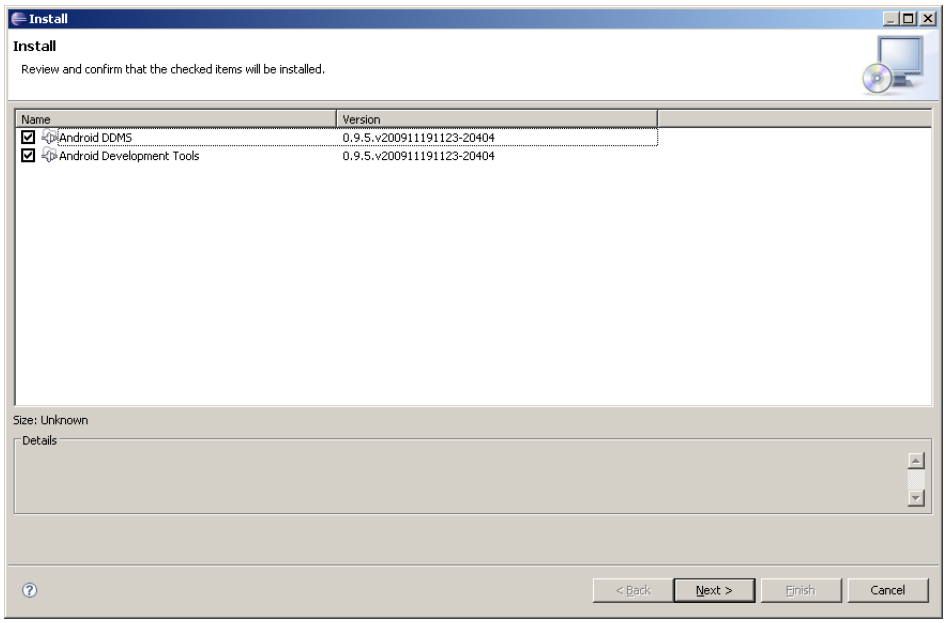


图 Eclipse 3.4 中进行安装 Android 的 DDMS 和 ADT

选择 Next 将出现如图的对话框：

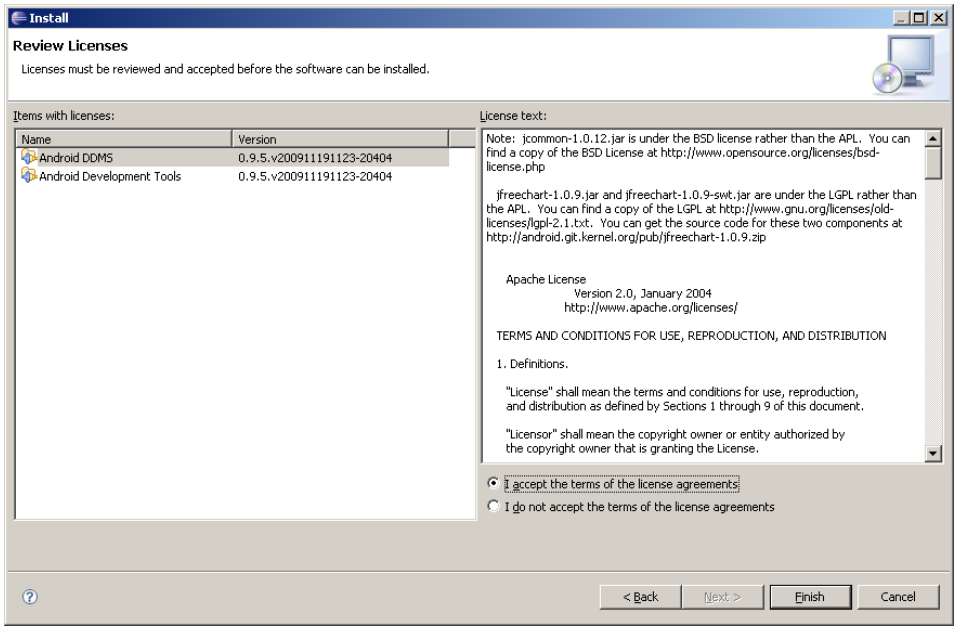


图 Eclipse 3.4 中选择同意 Android 的协议

选择接受（accept）并且选择 Finish 完成安装之前的配置，后面的将进入安装的 Android 组件的阶段。

安装的过程要经过寻找依赖和安装两个阶段，如图所示：

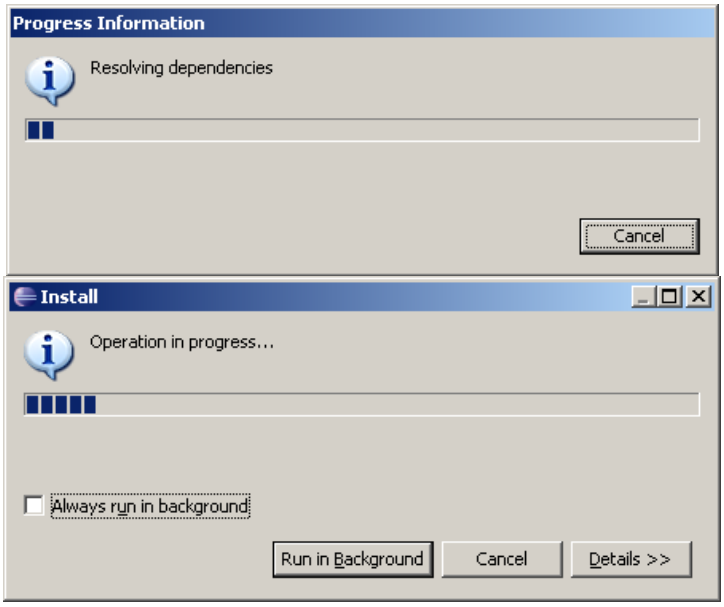


图 Eclipse 3.4 中解决依赖关系和安装

第四步：安装完成，关闭并重新启动 Eclipse。再次进入 Eclipse 3.4 后，将发现 ADT 已经被安装。

2.2.4（2）．在 Eclipse 3.5（Galileo）中安装 ADT

第一步：启动 Eclipse 选择 “Help”>“Install New Software...”准备安装插件。

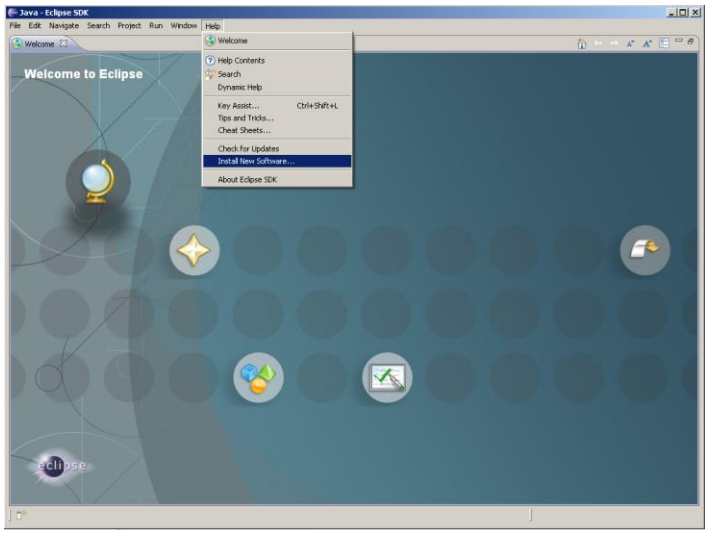


图 Eclipse 3.5 中选择安装新软件

第二步：出现软件升级的对话框

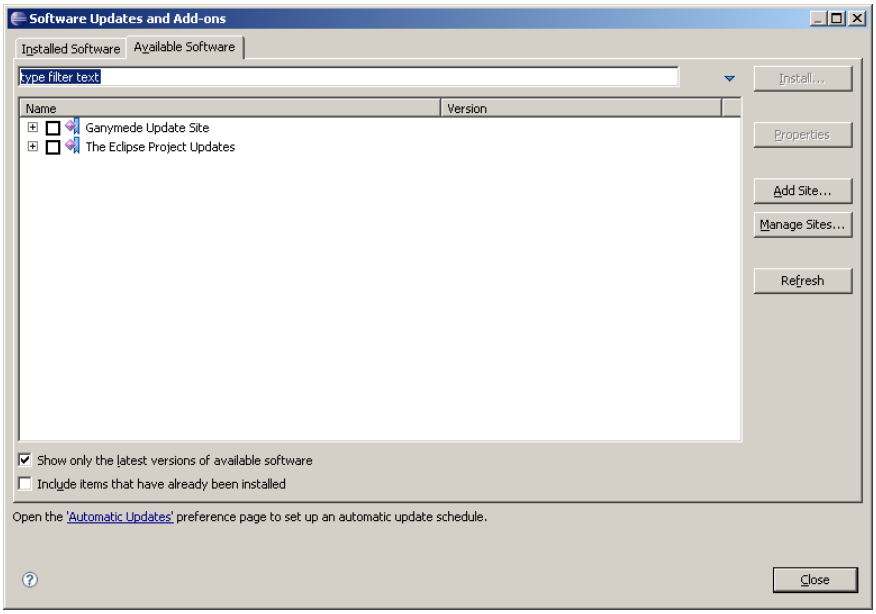


图 Eclipse 3.5 的软件升级的对话框

点击右侧自上而下的第 3 个按钮，“Add Site...”准备增加插件。

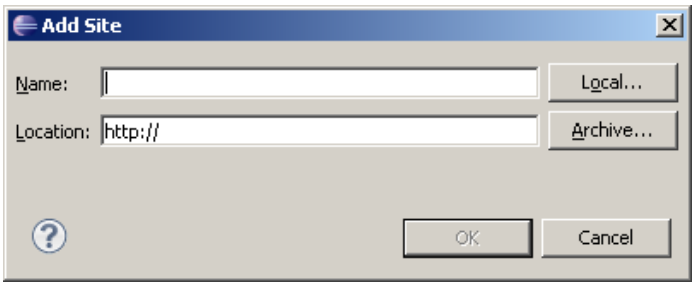


图 Eclipse 3.5 中增加 ADT 插件的路径

在“Add Site”对话框中，输入 Android 插件的路径：

https://dl-ssl.google.com/android/eclipse/

另外的一种方式是点击 Archive…按钮，这样可以不使用网络，直接指定磁盘中的 ADT 包（最新的版本是 ADT-0.9.5.zip）。

第四步：回到软件升级对话框，work with 的路径变为了 https://dl-ssl.google.com/android/eclipse/，后面的列表变为了“Developer Tools”，其中包含了两个项目：

- Android DDMS
- Android Development Tools

选择继续进行安装：

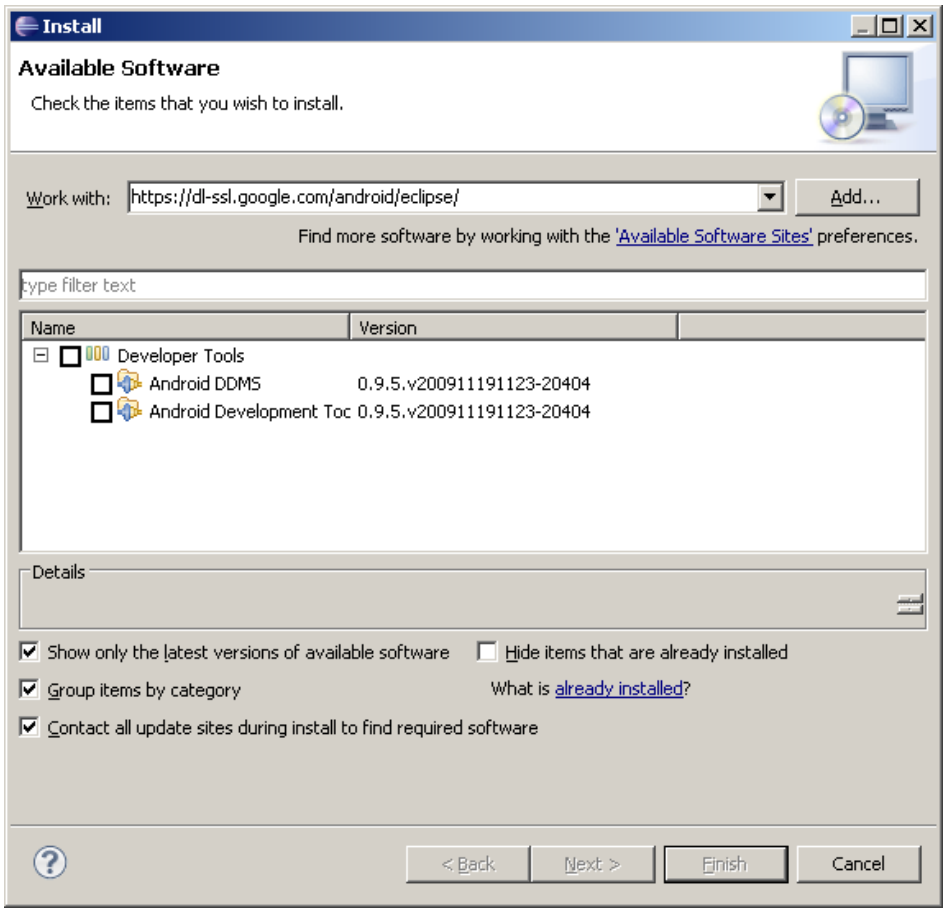


图 Eclipse 3.5 中选择安装 Android 的 DDMS 和 ADT

选中后，点击 Finish 将出现安装的详细信息的对话框，如图所示：

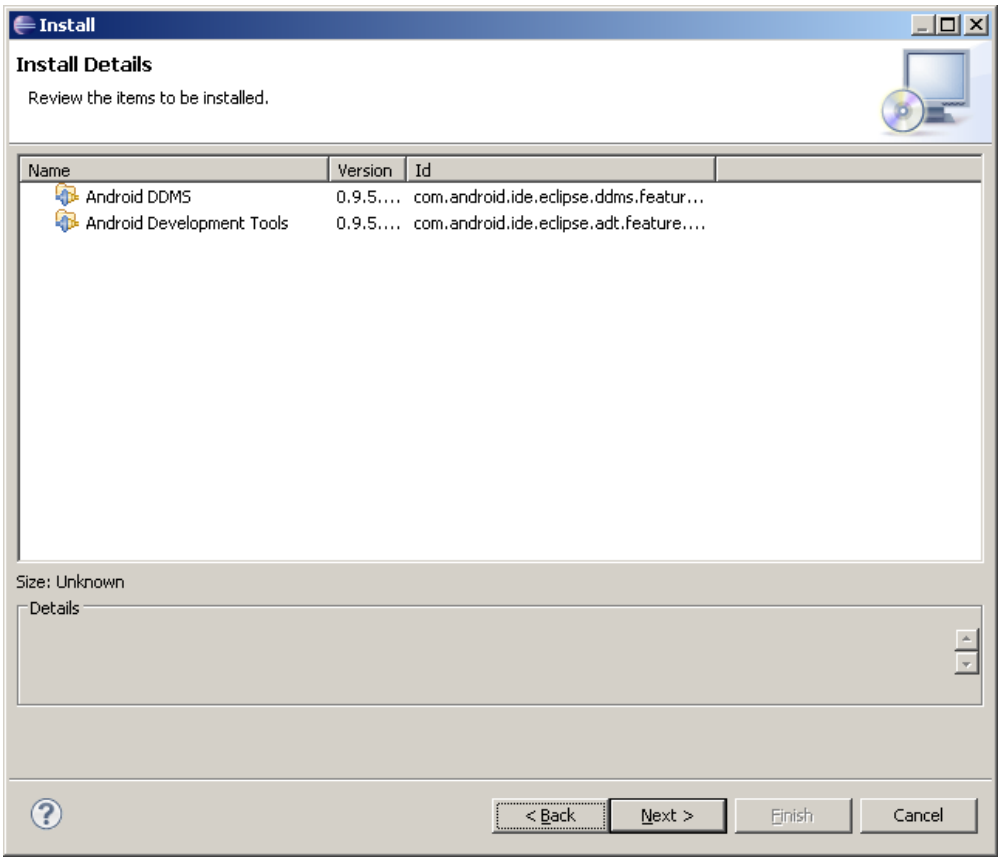


图 Eclipse 3.5 中选择安装 Android 的插件

选择 Next 进行下一步的安装。

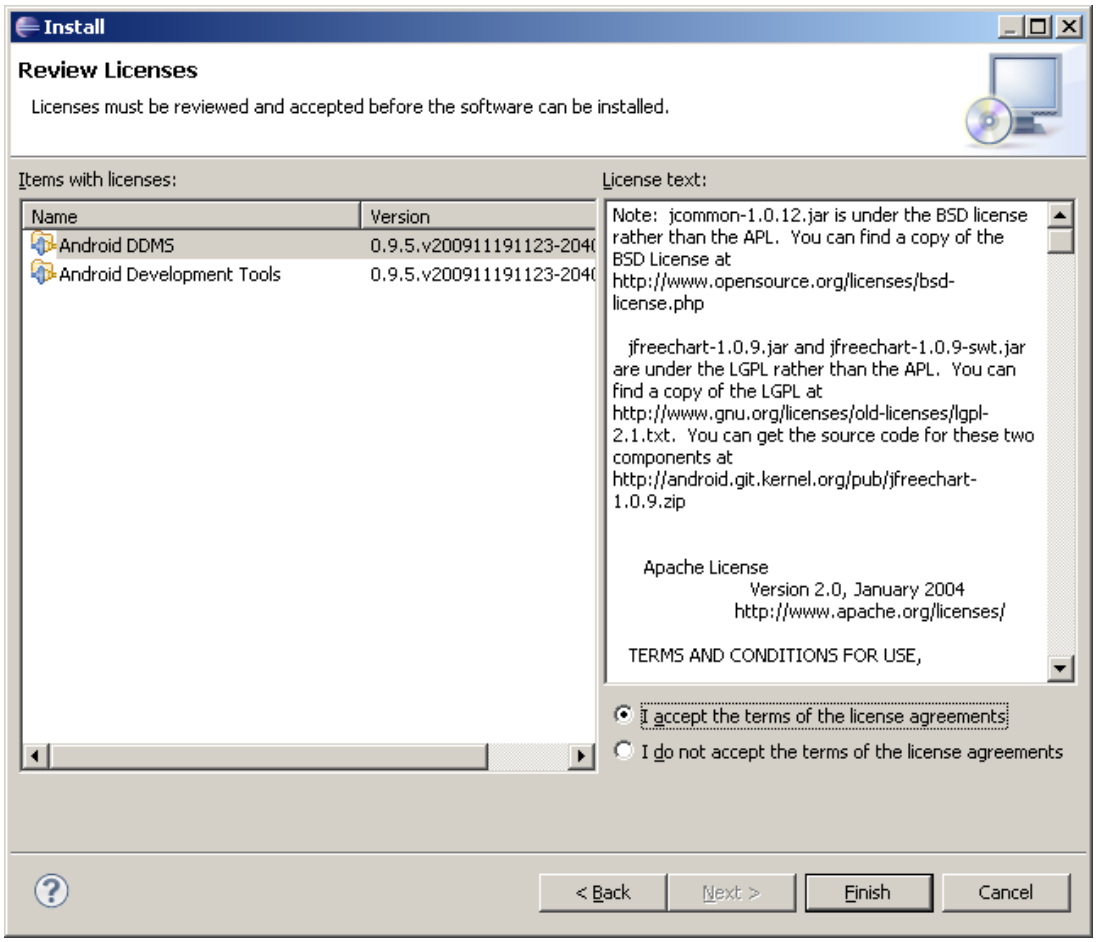


图 Eclipse 3.5 中选择同意 Android 的协议

选择接受（accept）并且选择 Finish 完成安装之前的配置，后面的将进入安装的 Android 组件的阶段。安装的过程如图所示：

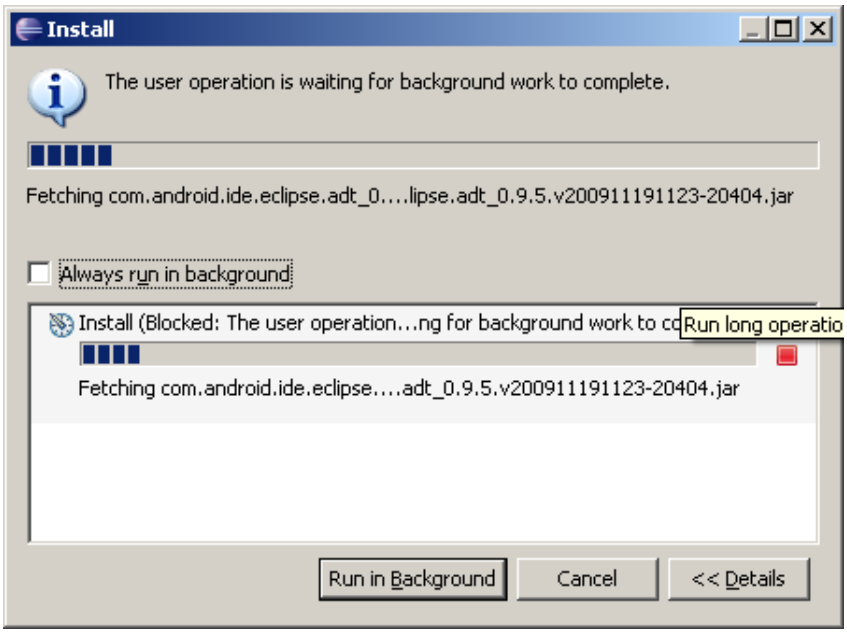


图 Eclipse 3.5 中选择进行 Android 的插件

第五步：安装完成，关闭并重新启动 Eclipse。再次进入 Eclipse 3.5 后，将发现 ADT 已经被安装。

2.2.5. 在 Eclipse 中配置 Android SDK

进入安装 ADT 的 Eclipse 环境后，选择“Window”>“Preference”,从左侧的列表中选择 Android 项：

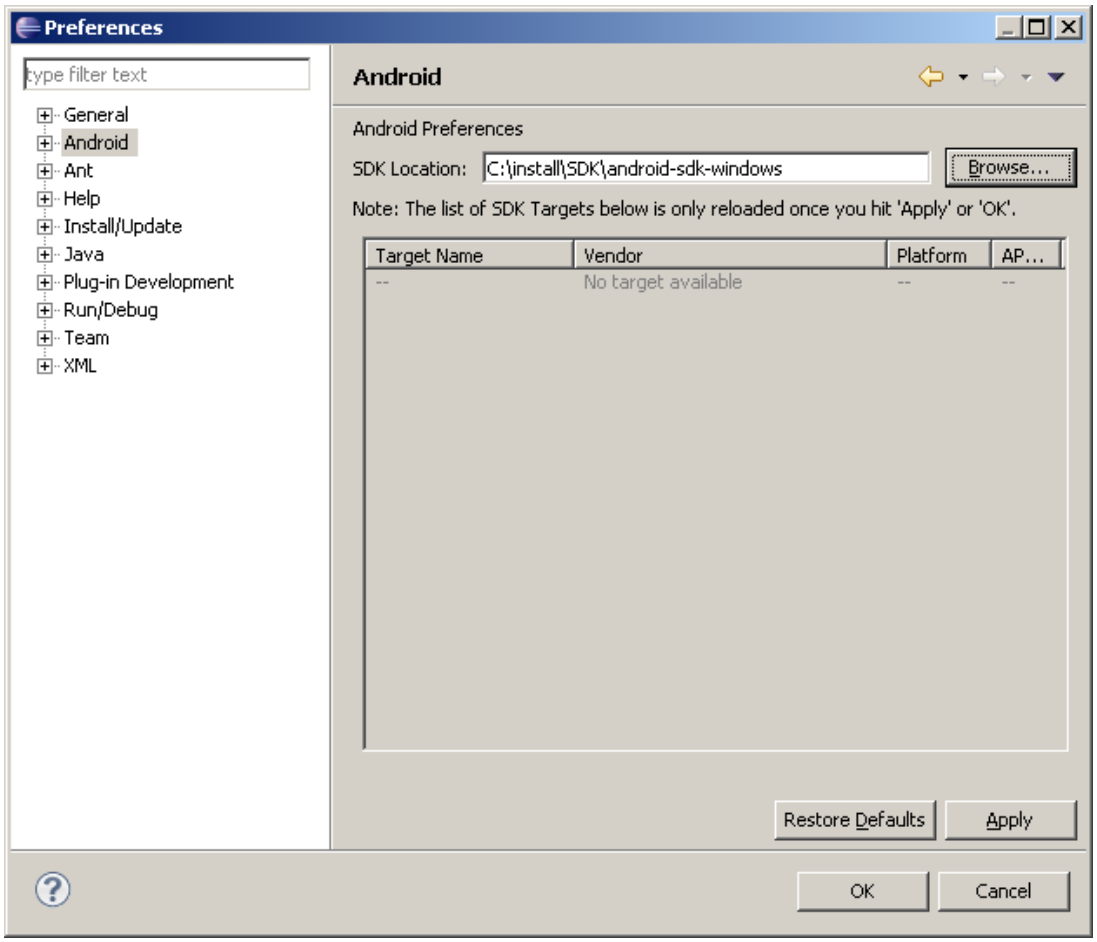


图 Eclipse 中选择 Android SDK 的路径

左侧的 Android 选项是由于安装了 Android 的 SDK 而出现的。

在 SDK 设置 SDK Location 中，点击“Browse”…按钮；选择 Android，SDK 的目录，点击“OK”按钮。

2.3 Android 中运行仿真器环境

2.3.1. 建立 Android 虚拟设备

为了运行一个 Android 仿真器的环境，首先需要建立 Android 虚拟设备（AVD）。在 Eclipse 的菜单中，选择“Window”>“Android AVD Manager”，出现“Android SDK and AVD Device Manager”窗口，界面如图所示：

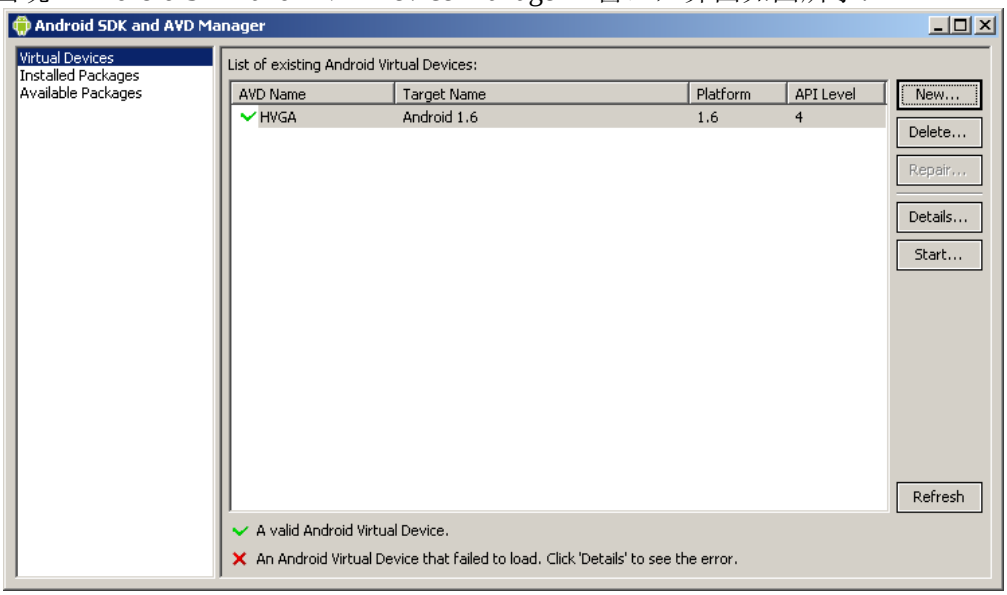


图 Android SDK 和 AVD 管理器

界面中间的列表表示了目前可以使用的 Android 虚拟设备，在没有虚拟设备的情况下点击右侧的 New 选择建立一个虚拟设备。

建立新的 Android 虚拟设备的窗口为 Create new AVD，如图所示：

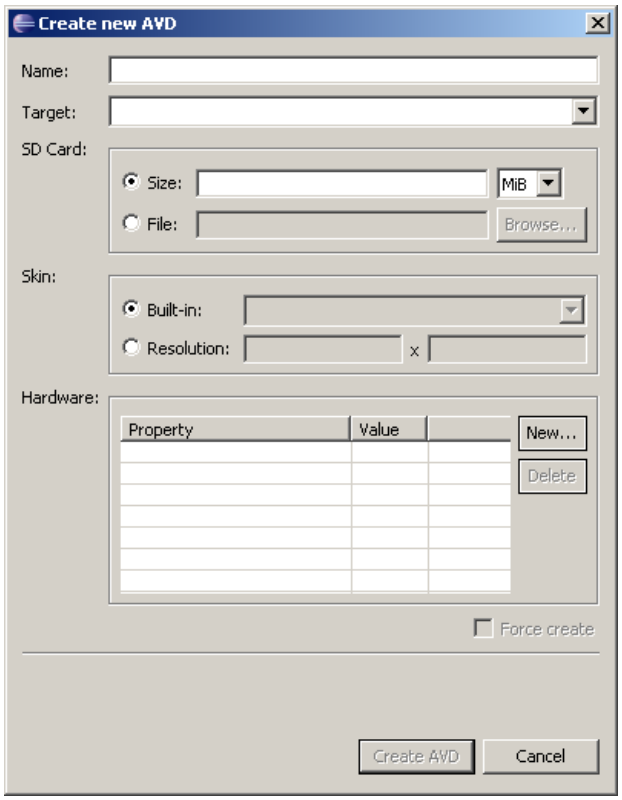


图 建立新的 AVD

Android 虚拟设备的建立包含了以下的一些选项：

- 名字（Name）：这个虚拟设备的名称，由用户自定义；
- 目标（Target）：选择不同的 SDK 版本（依赖一目前 SDK 的 platform 目中包含了哪些版本的 SDK）
- SD 卡：模拟 SD 卡，可以选择大小或者一个 SD 卡映像文件，SD 卡映像文件是使用 mksdcard 工具建立的。
- 皮肤（Skin）：这里皮肤的含义其实是仿真器运行尺寸的大小，默认的尺寸有 HVGA-P（320x480），HVGA-L（480x320）等，也可以通过直接指定尺寸的方式制定屏幕的大小。
- 属性：可以由用户指定仿真器运行的时候，Android 系统中一些属性

2.3.2. 运行虚拟设备

在“Android SDK and AVD Device Manager”窗口中，选择一个设备，点击右侧的 Start，将启动虚拟设备，运行一个 Android 系统，一个 HVGA-P（320x480）尺寸的运行结果如图所示：

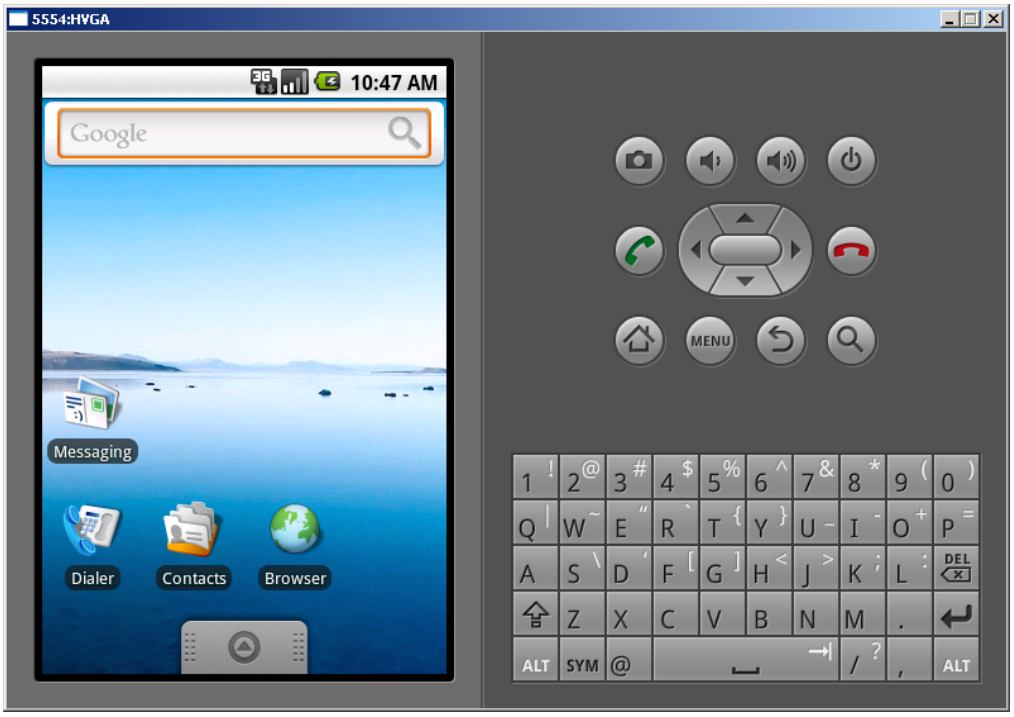


图 使用仿真器的运行 Android 系统

出现窗口的左侧是运行的仿真器的屏幕，右侧是模拟的键盘。设备启动后，可以使用右侧的键盘模拟真实设备的键盘操作，也可以用鼠标点击（或者拖拽和长按）屏幕，模拟触摸屏的操作。

除了使用右侧的模拟键盘之外，也可以使用 PC 机的键盘来进行模拟真实设备的键盘操作。尤其是当仿真器的大小不是标准值的时候，可能不会出现按键的面板，在这种情况下只能使用键盘的按键来控制仿真器的按键

按键之间的映射关系如下表所示：

仿真器的虚拟按键	键盘的按键
Home	HOME

Menu (左软按键)	F2 *or* Page-up button
Star (右软按键)	Shift-F2 *or* Page Down
Back	ESC
Call/dial button	F3
Hangup/end call button	F4
Search	F5
Power button	F7
Audio volume up button	KEYPAD_PLUS, Ctrl-5
Audio volume down button	KEYPAD_MINUS, Ctrl-F6
Camera button	Ctrl-KEYPAD_5, Ctrl-F3
切换到上一个布局方向(例如 portrait 和 landscape)	KEYPAD_7, Ctrl-F11
切换到下一个布局方向(例如 portrait 和 landscape)	KEYPAD_9, Ctrl-F12
切换 Cell 网络的开关 on/off	F8
切换 Code profiling	F9
切换全屏模式	Alt-Enter
切换跟踪球（trackball）模式	F6
临时进入跟踪球（trackball）模式（当长按按键的时候）	Delete
DPad left/up/right/down	KEYPAD_4/8/6/2
DPad center click	KEYPAD_5
Onion alpha 的增加和减少	KEYPAD_MULTIPLY(*) / KEYPAD_DIVIDE(/)

Android 仿真器启动虚拟设备之后，默认就可以使用主机的网络作为自己的网络、使用主机的音频设备作为自己的声音输出。

2.3.3. 使用 Android 中的工具

在仿真器环境中，可以使用集成的 Android 相关工具。使用的方法是 Window-> Show View -> Other 选项，可以开启 Android 的各个工具。调用的过程如下图所示：

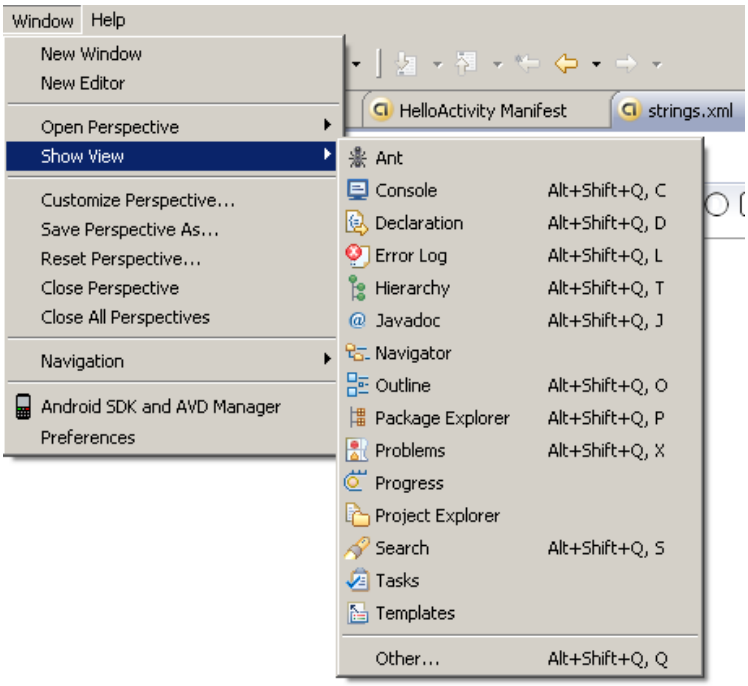


图 选择 Android 的各个工具

选择 Android 工具的对话框如图所示：

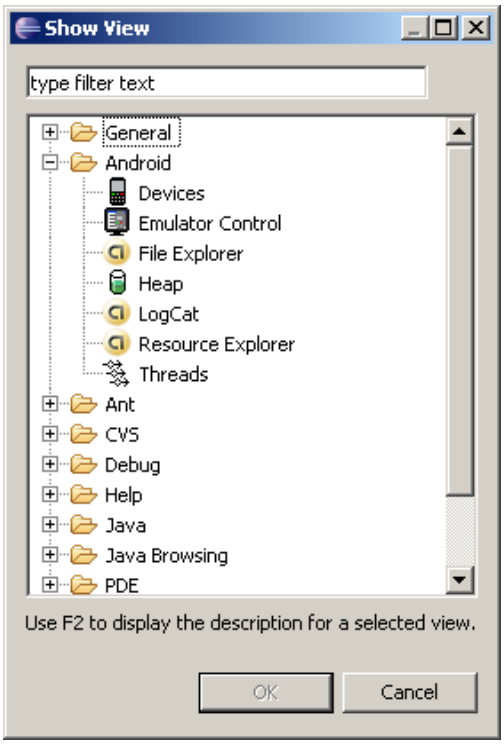


图 选择 Android 工具的对话框

这里可以选择的主要工具有 Device（设备控制）、Emulator Control（仿真器控制）、File Explore（文件浏览）、Heap（堆内存）、Logcat、Resource Explore（资源浏览）、Threads（线程等）。每个工具开启之后，将出现一个单独的选项卡。

2.3.4. 使用 logcat

Logcat 工具是查看系统 Log 信息的工具，可以获得 Android 系统运行的时候打印出来的信息。工具的界面如下所示：

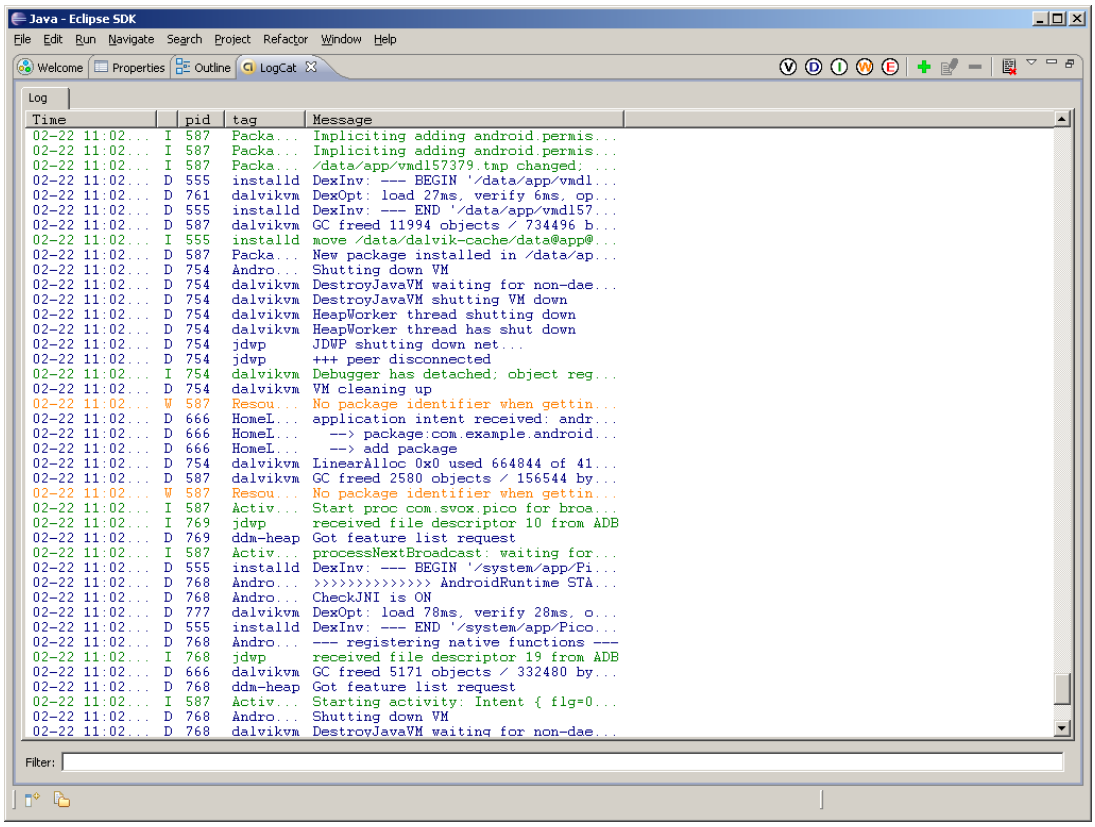


图 使用 Logcat 工具显示 LOG

Logcat 实际上是一个运行在目标系统的工具，也就是一个 Linux 的命令行程序，这是界面种是带有 GUI 的效果。Logcat 的窗口中记录的信息也就是实际的 Android 系统打印出来的。包含了时间（Time）、级别（Level）、进程 ID（Pid）、标签（tag）、Log 内容（Message）等项目。

Logcat 窗口可以设置 Log 的过滤器（Filter），这样可以仅仅获得自己需要的 Log 信息，屏蔽其他的信息。

命令行程序 logcat，位于目标文件系统中该工具位于 system/bin 目录中，Logcat 的使用方法如下所示：

```
# logcat [options] [filterspecs]
```

logcat 工具的选项如下所示：

-s 设置过滤器，例如指定 '*:s

-f <filename>　　输出到文件，在默认情况下是标准输出
-r [<kbytes>]　　循环 log 的字节数（默认为 16），需要 －f
-n <count>　设置循环 log 的最大数目，默认为 4
-v <format>　设置 log 的打印格式，<format> 是下面的一种：
brief process tag thread raw time threadtime long
-c　　清除所有 log 并退出
-d　得到所有 log 并退出（不阻塞）
-g　　得到环形缓冲区的大小并退出
-b <buffer>　请求不同的环形缓冲区（'main'（默认）、'radio'、'events'）
-B　将 log 输出到二进制文件中

2.3.5. 使用仿真器控制

选择 Emulator Control 选项可以开启仿真器的控制对话框，它的界面如下所示：

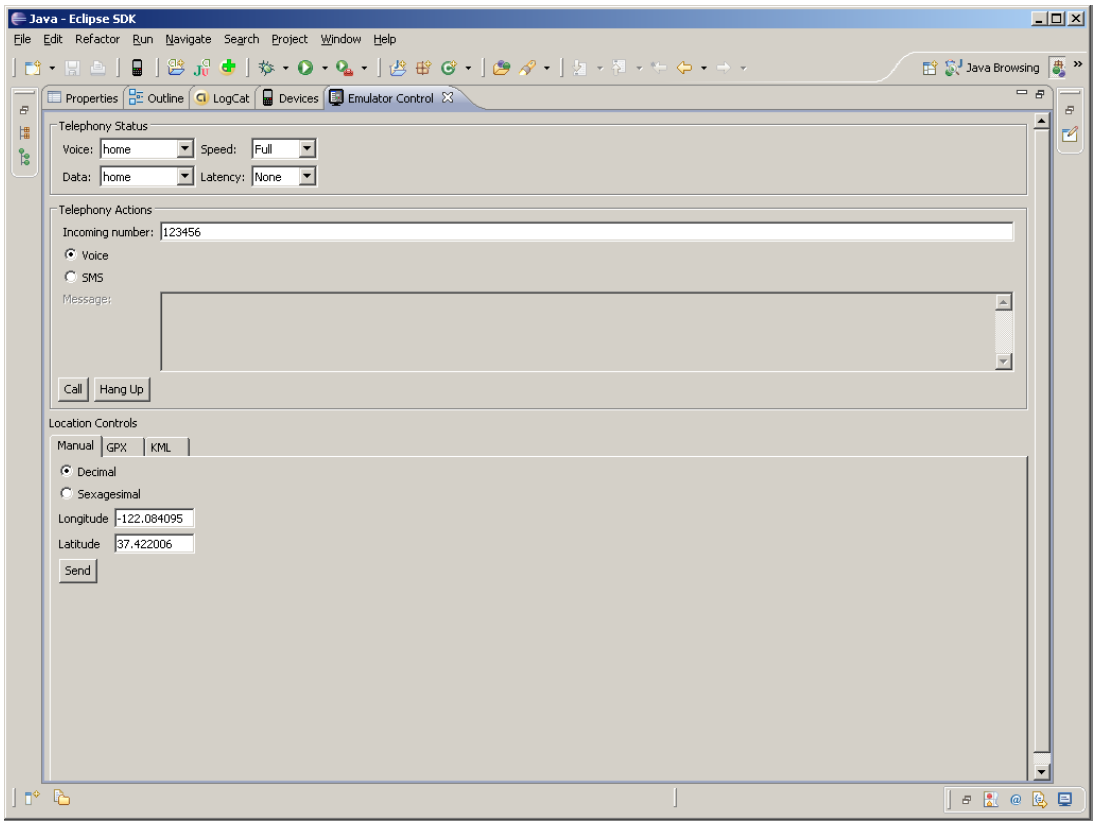


图 Android 仿真器控制界面

它甚至可以模拟打电话，发短信的过程。例如在 incoming number 中输入电话号码，然后点击 Call 按钮。这是仿真器的运行界面如图所示：

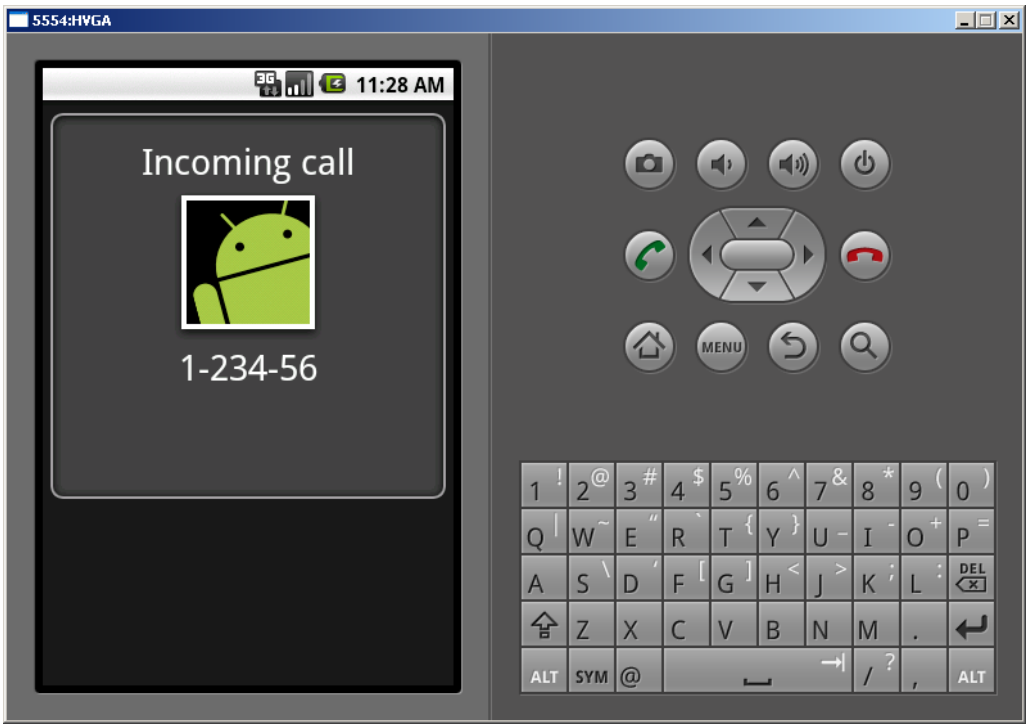


图 Android 仿真器接收来电

接受电话的程序已经被调用起来，这里显示的电话号码 1-234-56，也是在仿真器控制的窗口中设置的。

模拟发送短信的界面显示如下所示：

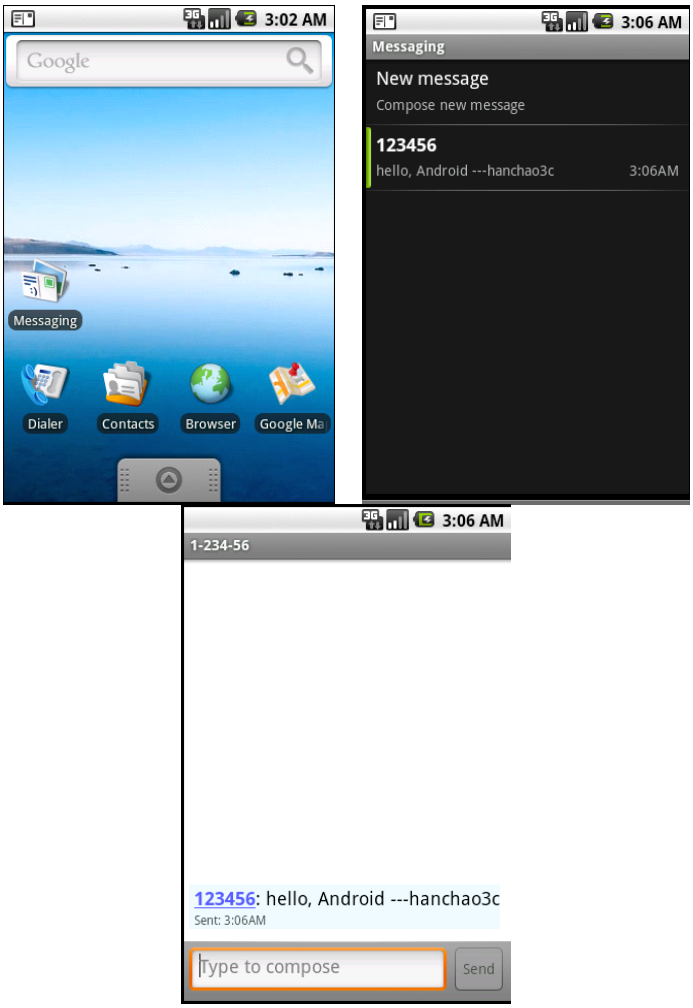


图 Android 仿真器接收短信

2.3.6. 命令行工具 adb、mksdcard 等

有一些 Android 的工具需要在命令行的环境中运行，只是可以选择 Windows 的开始->运行，键入 cmd 并确定，进入命令行的界面中运行。主要的命令行工具包括 adb 和 mksdcard 等。命令行的工具在 Android SDK 的 tools 目录中，使用命令行的窗口如图所示：

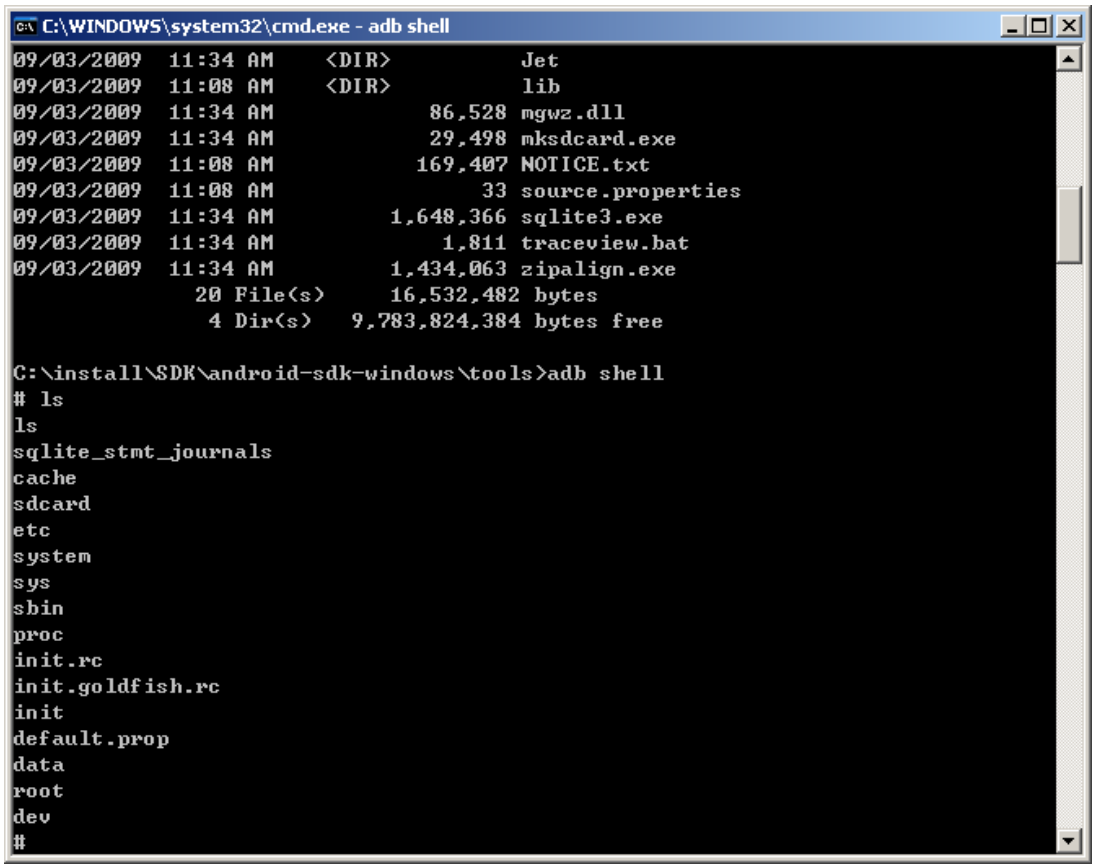


图 在命令行中使用 adb

adb（Android Debug Bridge，Android 调试桥）是 Android 的主要调试工具，它可以通过网络或者 USB 连接真实的设备，也可以连接仿真器。使用 adb 进行调试，通常在命令行的界面中。

将出现 shell 提示符，这就是 Android 所运行的 Linux 系统中的 shell 终端，可以在这个 shell 提示符后执行 Android 系统提供的 Linux 命令。

使用 ls 命令查看 Android 系统根目录：

```
# ls -l
drwxrwxrwt root     root              2009-06-15 02:17 sqlite_stmt_journals
drwxrwx--- system   cache             2009-06-15 02:18 cache
d--------- system   system            2009-06-15 02:17 sdcard
lrwxrwxrwx root     root              2009-06-15 02:17 etc -> /system/etc
drwxr-xr-x root     root              2009-05-28 02:16 system
drwxr-xr-x root     root              1970-01-01 00:00 sys
```

```
drwxr-x--- root     root              1970-01-01 00:00 sbin
dr-xr-xr-x root     root              1970-01-01 00:00 proc
-rwxr-x--- root     root         9075 1970-01-01 00:00 init.rc
-rwxr-x--- root     root         1677 1970-01-01 00:00 init.goldfish.rc
-rwxr-x--- root     root       106568 1970-01-01 00:00 init
-rw-r--r-- root     root          118 1970-01-01 00:00 default.prop
drwxrwx--x system   system            2009-05-28 02:49 data
drwx------ root     root              1970-01-01 00:00 root
drwxr-xr-x root     root              2009-06-15 02:18 dev
```

Android 根目录中的主要文件夹与目标系统的 out/target/product/generic/root 内容相对应，此外 etc、proc 等目录是在 Android 启动后自动建立的，system 映像被挂接到根文件系统的 system 目录中，data 映像被挂接到根文件系统的 data 目录中。

使用 ps 命令可以查看 Android 系统的进程：

```
# ps
USER     PID   PPID  VSIZE RSS     WCHAN    PC         NAME
root     1     0     280   188     c008de04 0000c74c S /init
root     2     0     0     0       c004b334 00000000 S kthreadd
root     3     2     0     0       c003cf68 00000000 S ksoftirqd/0
root     4     2     0     0       c00486b8 00000000 S events/0
root     5     2     0     0       c00486b8 00000000 S khelper
root     10    2     0     0       c00486b8 00000000 S suspend
root     42    2     0     0       c00486b8 00000000 S kblockd/0
root     45    2     0     0       c00486b8 00000000 S cqueue
root     47    2     0     0       c016f13c 00000000 S kseriod
root     51    2     0     0       c00486b8 00000000 S kmmcd
root     96    2     0     0       c0065c7c 00000000 S pdflush
root     97    2     0     0       c0065c7c 00000000 S pdflush
root     98    2     0     0       c006990c 00000000 S kswapd0
root     100   2     0     0       c00486b8 00000000 S aio/0
root     269   2     0     0       c016c884 00000000 S mtdblockd
root     304   2     0     0       c00486b8 00000000 S rpciod/0
root     540   1     740   328     c003aa1c afe0d08c S /system/bin/sh
system   541   1     808   264     c01654b4 afe0c45c S /system/bin/servicemanager
root     542   1     836   364     c008e3f4 afe0c584 S /system/bin/vold
root     543   1     668   264     c0192c20 afe0cdec S /system/bin/debuggerd
radio    544   1     5392  684     ffffffff afe0cacc S /system/bin/rild
root     545   1     72256 20876   c008e3f4 afe0c584 S zygote
media    546   1     17404 3496    ffffffff afe0c45c S /system/bin/mediaserver
bluetooth 547  1     1168  568     c008de04 afe0d25c S /system/bin/dbus-daemon
root     548   1     800   300     c01f3b04 afe0c1bc S /system/bin/installd
root     551   1     840   356     c00ae7b0 afe0d1dc S /system/bin/qemud
root     554   1     1268  116     ffffffff 0000e8f4 S /sbin/adbd
system   570   545   175652 23972  ffffffff afe0c45c S system_server
radio    609   545   105704 17584  ffffffff afe0d3e4 S com.android.phone
app_4    611   545   113380 19492  ffffffff afe0d3e4 S android.process.acore
app_12   632   545   95392 13228   ffffffff afe0d3e4 S com.android.mms
app_4 645 545  97192 12964 ffffffff afe0d3e4 S com.android.inputmethod.latin
```

```
app_5    655   545   95164 13376 ffffffff afe0d3e4 S android.process.media
app_7    668   545   97700 14264 ffffffff afe0d3e4 S com.android.calendar
app_11   684   545   94132 12624 ffffffff afe0d3e4 S com.android.alarmclock
root     702   540   888   340   00000000 afe0c1bc R ps
```

从系统的进程中可以看到，系统 1 号和 2 号进程以 0 号进程为父进程。init 是系统运行的第 1 个进程，即 Android 根目下的 init 可执行程序，这是一个用户空间的进程。kthreadd 是系统的 2 号进程，这是一个内核进程，其他内核进程都直接或间接以它为父进程。

Zygote、/system/bin/sh、/system/bin/mediaserver 等进程是被 init 运行起来的，因此它们以 init 为父进程。其中 android.process.acore（Home）、com.android.mms 等进程代表的是应用程序进程，它们的父进程都是 zygote。

使用 adb 连接目标系统终端的方式如下所示：

```
> adb shell
```

使用 adb 安装应用程序的方法为：

```
> adb install XXX.apk
```

使用 adb 在主机和目标机之间传送文件的方法为：

```
> adb push {host_path} {target_path}
> adb pull {target_path} {host_path}
```

mksdcard 是用来建立 SD 卡映像的工具，用来建立一个 Fat32 格式的磁盘映像，其使用方法如下所示：

```
mksdcard [-l label] <size> <file>
```

mksdcard 的参数-l 用于指定磁盘映像的标签，size 用于指定磁盘映像的大小，其后面可以跟 K、M、G 等参数，file 是磁盘映像的文件名称，这个文件也就是在仿真器运行过程中指定的文件。

mksdcard 的一个使用的示例如下所示：

```
> mksdcard 128M sdcard.img
```

这表示建立了一个大小为 128M，名称为 sdcard.img 的 Fat32 磁盘映像文件。

2.3.7. 使用设备控制

Device 工具可以用于进一步控制仿真器的运行状况，在其中可以查看 Heap（堆内存）、Threads（线程）的信息，还具有停止某个进程的运行，截取屏幕等功能。Device 工具的窗口如图所示：

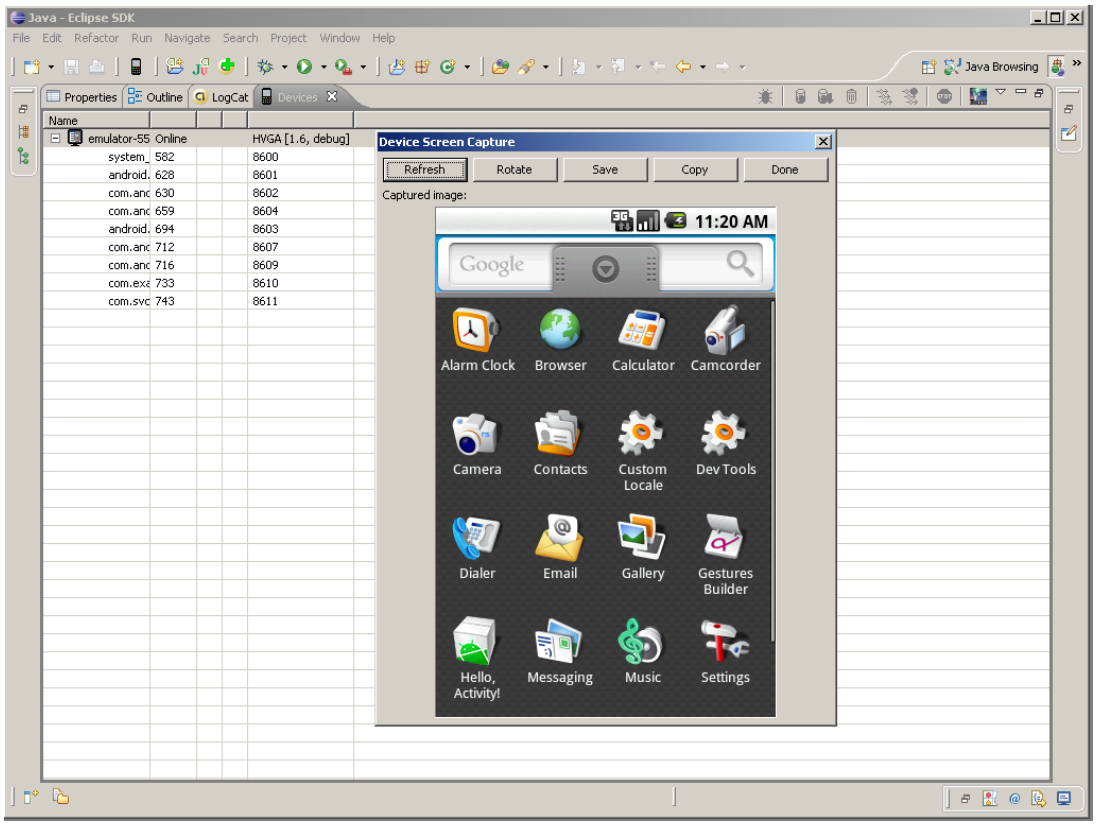


图 Android 的设备工具

点击 Device 窗口工具栏最右侧的 Screen Capture 按钮，可以打开截取屏幕的窗口，如上图所示。

2.4 Android 中建立工程

2.4.1. 建立工程

Android 的 SDK 环境安装完成后，就可以在 SDK 中建立工程并进行调试了。建立 Android 工程步骤如下：

选择“File”>“New”>“Project”

选择“Android”>“Android Project”，点击“Next”按钮：

选择 the contents for the project。

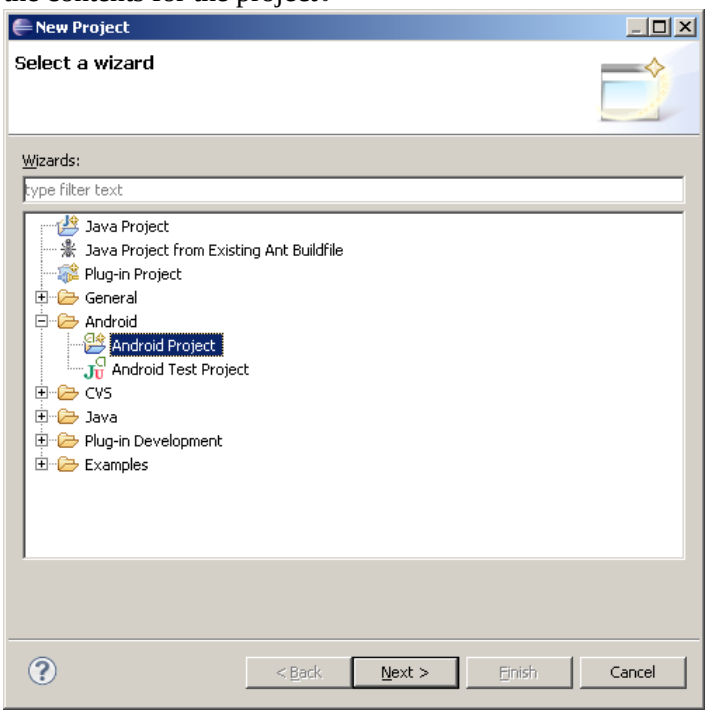


图 建立新的 Android 工程

可以选择新建工程或从源代码建立工程，如果从源代码建立工程，那么所指定的目录中需要具有 AndroidManifest.xml 文件。

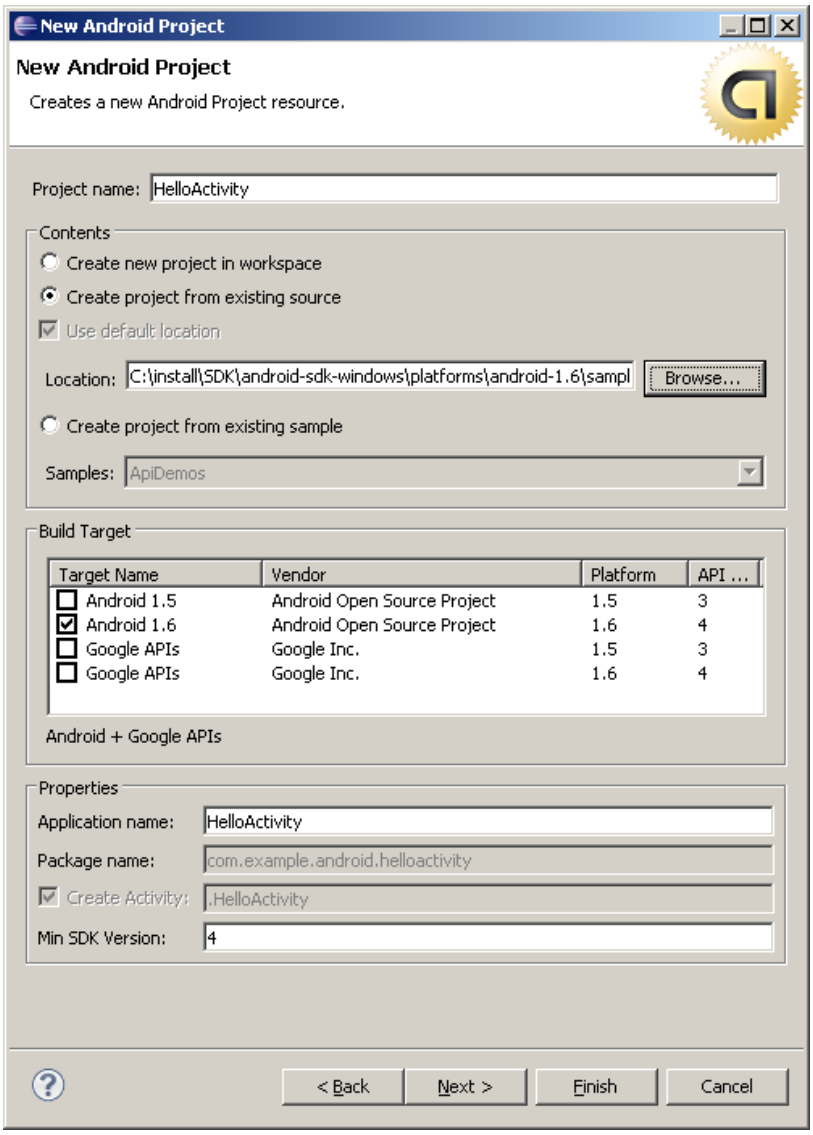


图 使用已有的示例建立新工程

可以使用 SDK 的 platforms/android-XXX/samples 中的各个子目录建立工程，这是 SDK 自带的示例程序，例如，使用 HelloActivity 示例程序。

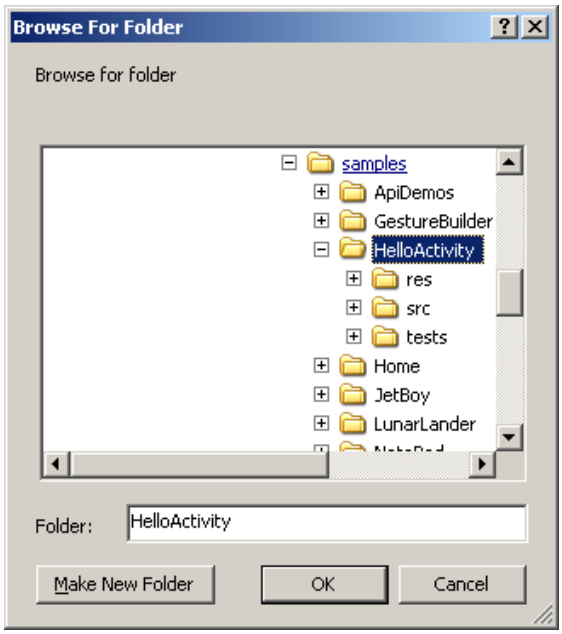


图 选择工程示例

点击“Finish”按钮，工程将被建立。

2.4.2. 查看和编辑各个文件

建立工程后，可以通过 IDE 环境查看和编辑 Android 应用程序中的各个文件。不同的文件将使用不同的工具查看。

查看 AndroidManifest.xml 文件的情况，如图所示：

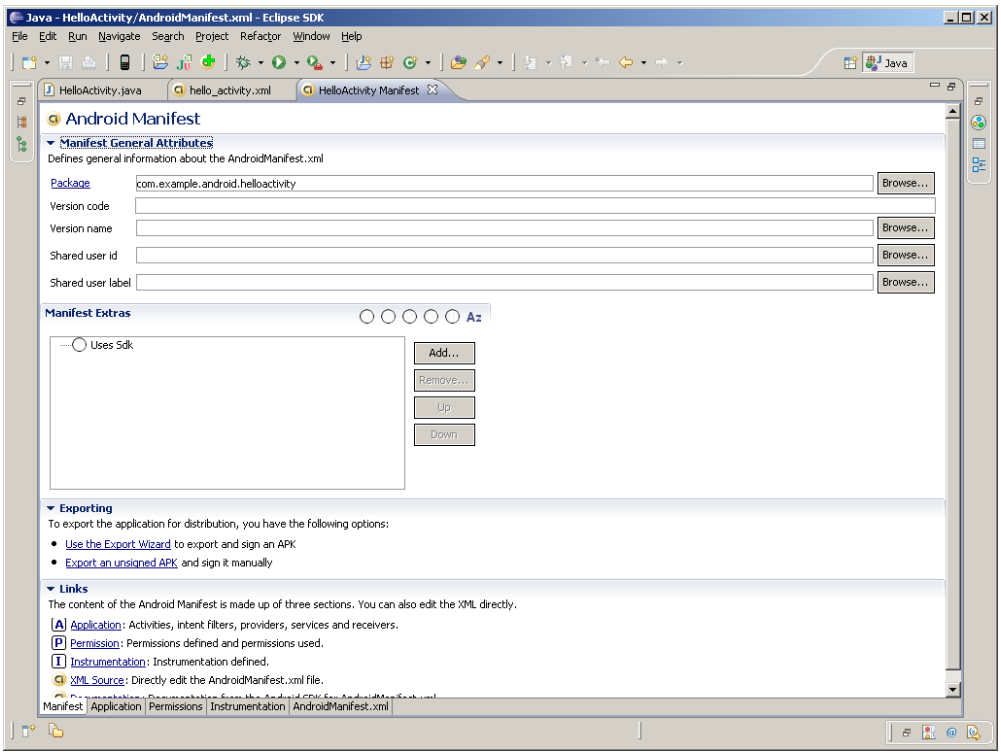


图 查看和编辑 AndroidManifest.xml 文件

显示的内容是以窗口的方式查看和更改 AndroidManifest.xml 中的内容，点击下面的 AndroidManifest.xml 标签将切换到文本模式，使用文本的形式查看和编辑 AndroidManifest.xml 中的内容。

浏览布局文件，如图所示：

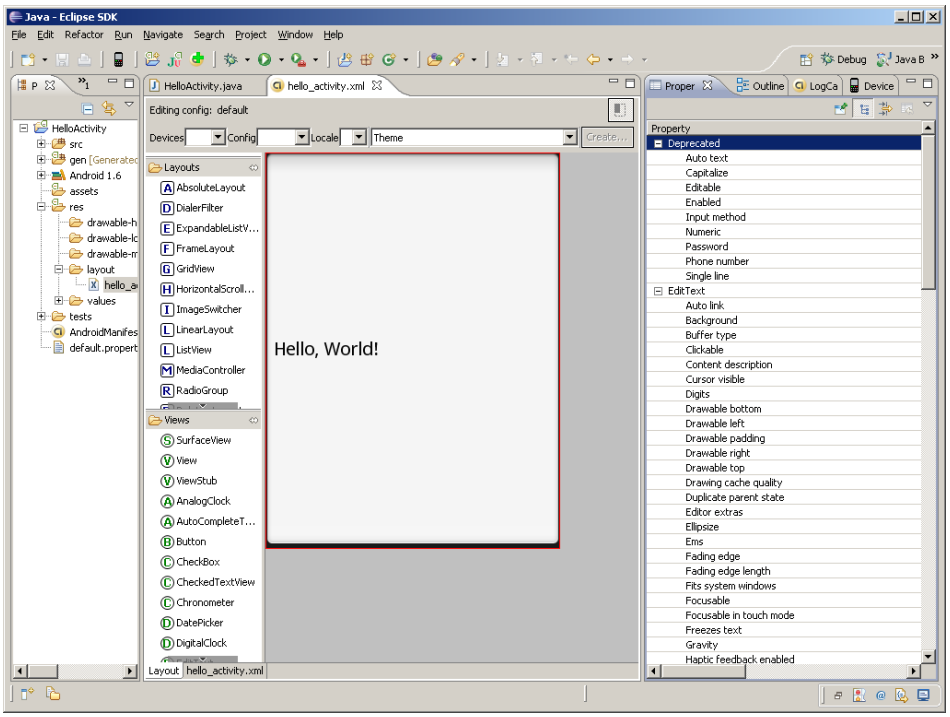


图 查看和编辑布局文件

浏览布局文件是一个更有用的功能，可以直观地查看程序的 UI 布局，点击标签（布局文件的名称）可以切换到文本模式。利用 IDE 的布局查看器，可以在程序没有运行的情况下直接查看和组织目标 UI 界面。

查看各个 value 文件和建立数值，如图所示：

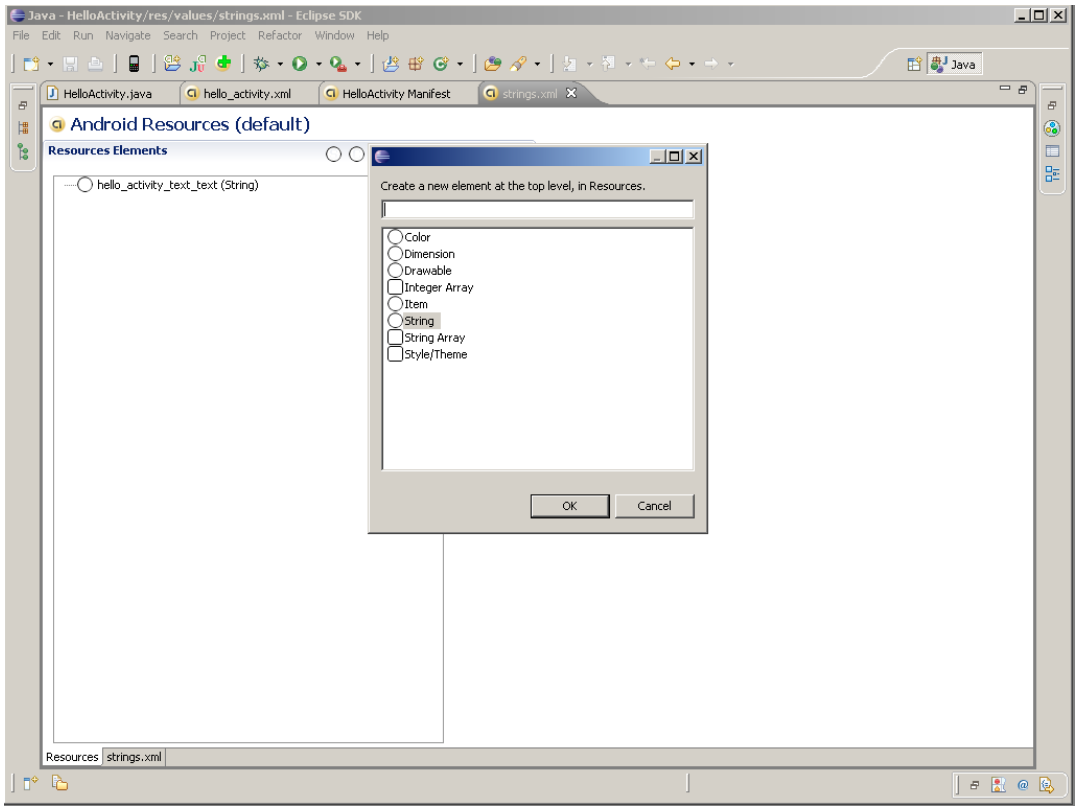


图 查看各个 value 文件和建立数值

查看各个 Java 源代码文件，如图所示：

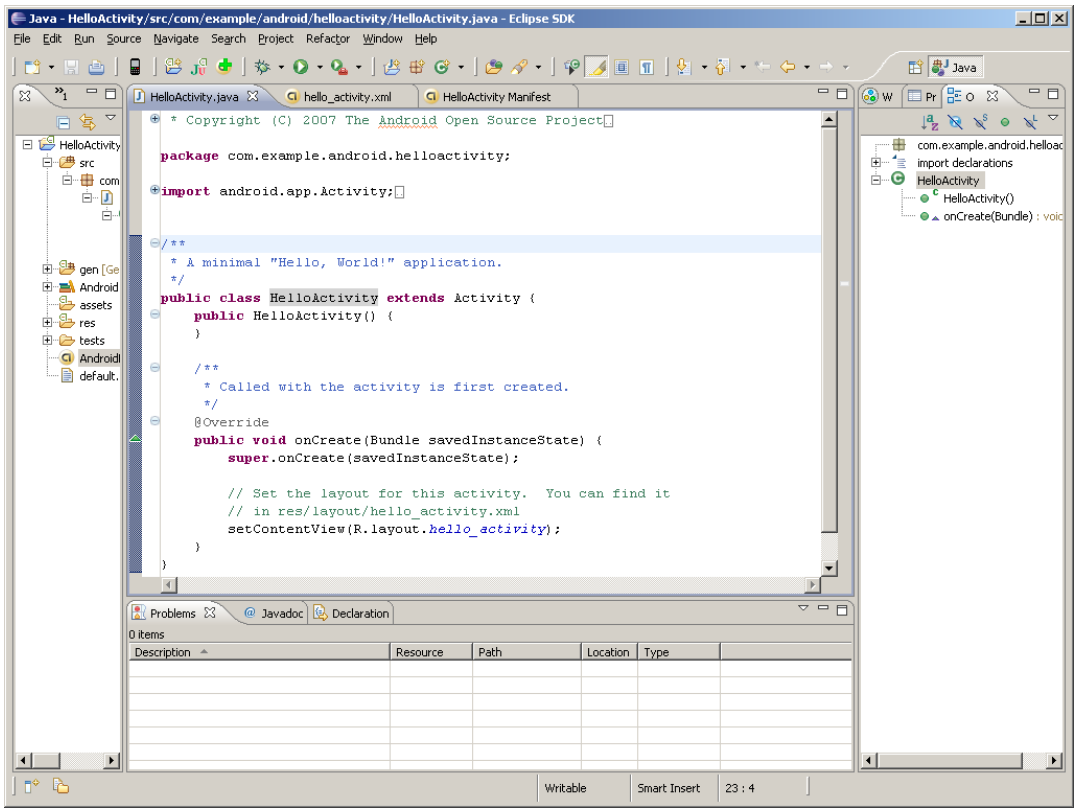


图 Java 源代码文件的编辑界面

Java 源代码采用文本的方式，但是在右边也列出了 Java 源代码中类的层次结构。在 IDE 的源代码环境开发 JAVA 程序，还具有自动修正、自动增加依赖包、类方法属性查找等功能。

2.4.3. 运行工程

在 Android 中运行一个工程，可以使用，右键单击工程名称，“选择 Run As”或者“Debug As”来运行和调试工程：

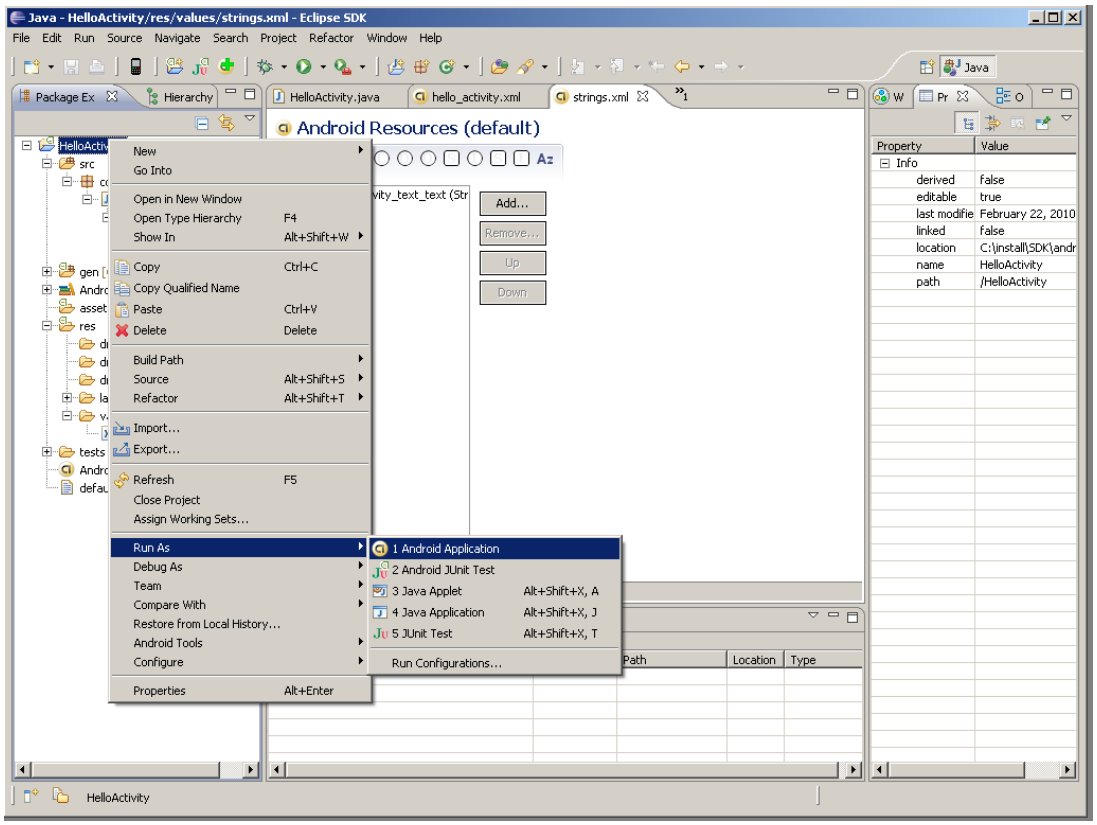


图 运行 Android 工程

开始运行的时候，如果现在已经有连接到真实的设备或者仿真器设备上，将直接使用这个设备，否则将启动一个新的仿真设备。

开始运行后，在 IDE 下层的控制台（console）标签中，将出现目标运行的 log 信息，可以获取目标运行的信息。出现类似的 Log 信息：

```
    [HelloActivity]Android Launch!
    [HelloActivity]adb is running normally.
    [HelloActivity]Performing  com.example.android.helloactivity.HelloActivity
activity launch
    [HelloActivity]Automatic Target Mode: using existing emulator 'emulator-5554'
running compatible AVD 'HVGA'
    [HelloActivity]WARNING: Application does not specify an API level requirement!
    [HelloActivity]Device API version is 4 (Android 1.6)
    [HelloActivity]Uploading HelloActivity.apk onto device 'emulator-5554'
    [HelloActivity]Installing HelloActivity.apk...
    [HelloActivity]Success!
    [HelloActivity]Starting                                          activity
```

```
com.example.android.helloactivity.HelloActivity on device
    [HelloActivity]ActivityManager:              Starting:              Intent
{ cmp=com.example.android.helloactivity/.HelloActivity }
```

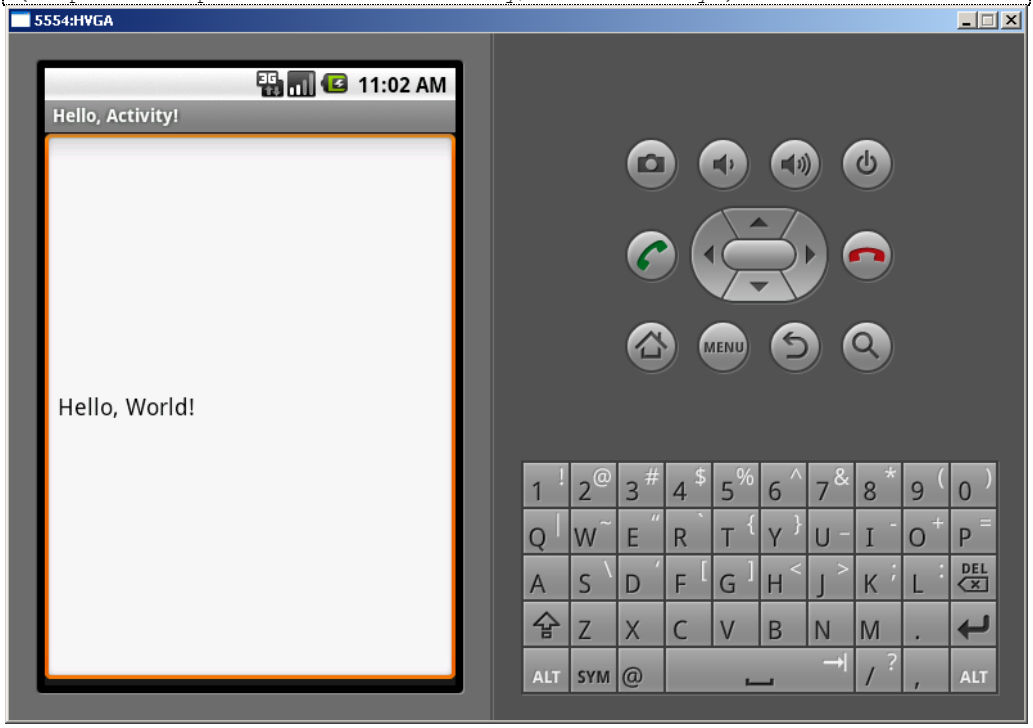


图 运行 HelloActivity 程序

在运行的一个仿真设备的时候，可以进一步通过选择“Run As”中的“Run Configurations”进行进一步的配置。 启动后的界面如图所示：

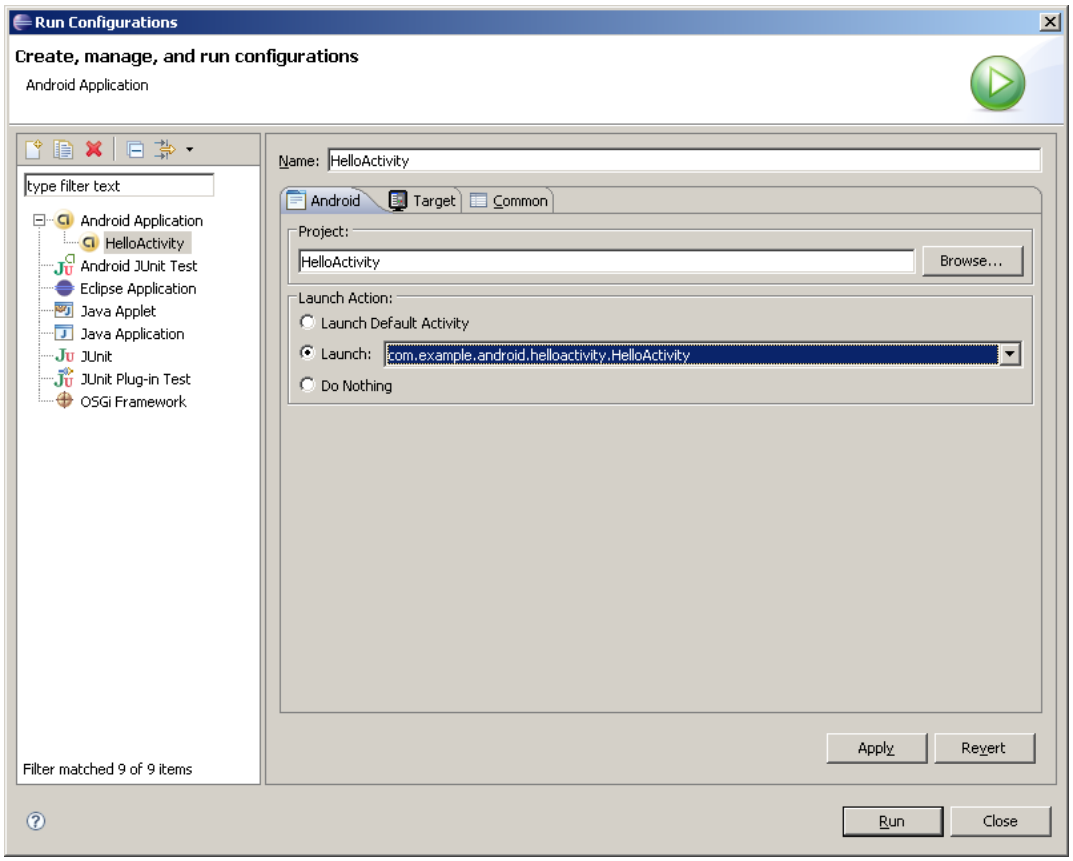


图 选择工程中运行的动作

其中，在 Android 的标签中可以选择启动的工程，启动活动（Launch Action）选项中可以选择启动的哪一个活动（Android 的一个工程中可以包含多个活动）。在 Target 标签中可以选择启动的时候使用的设备。

第二篇 Android 应用程序的概述和框架

第 3 章 Android 应用层程序的开发方式

3.1 应用程序开发的结构

Android 应用程序开发是 Android 开发中最上面的一个层次，它们构建在 Android 系统提供的 API 之上。Android 应用程序的基础是 Android 提供的各个 Java 类，这些类组成了 Android 系统级的 API。

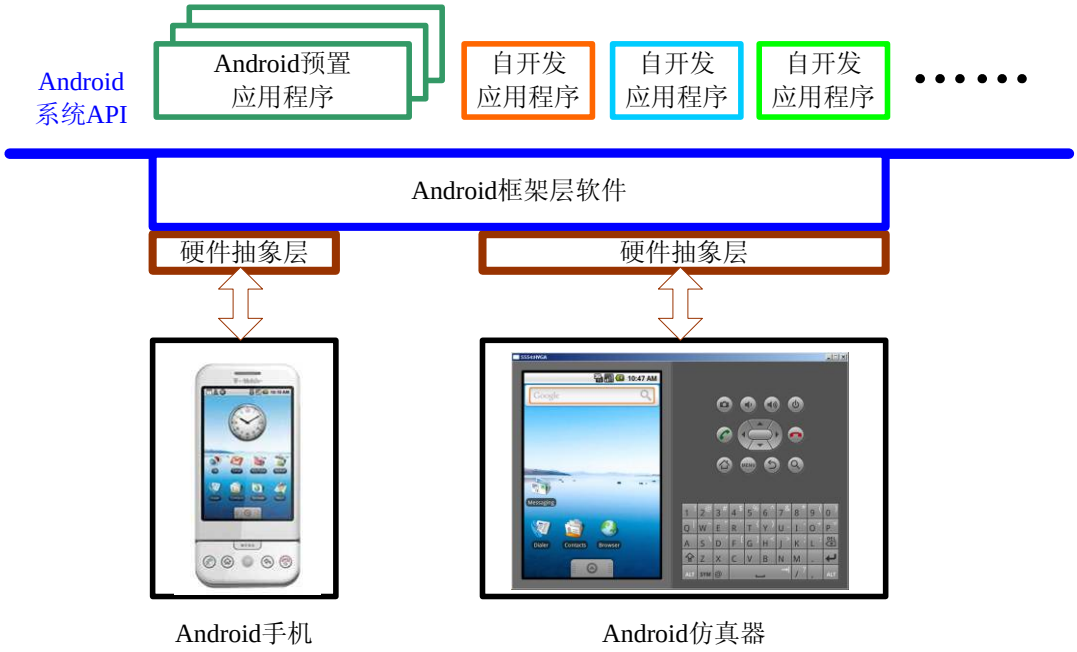


图　Android 应用的开发结构

Android 应用程序可以基于两种环境来开发：Android SDK 和 Android 源代码。Android 系统本身内置了一部分标准应用（也包括内容提供者），在仿真器（包括 SDK 环境和源代码环境）中已经包含这些内置的程序。

用户自行开发的应用程序和 Android 内置的应用层程序包位于同一个层次，都是基于 Android 框架层的 API 来构建的，它们的区别仅仅在于他们是否被包含在默认的 Android 系统中。

3.2 API 参考文档的使用

在开发 Android 应用程序时可以参考 SDK 中提供的参考文档，其内容包含在 Reference 标签中。

参考文档分成两种索引方式：

- Package Index（包索引）；
- Class Index（类索引）。

包索引根据字母顺序列出 Android 的各个包，每个包中包含若干个类、接口等内容；类索引按照字母顺序列出了所有的类（也包括接口等内容）。在查找一个类的帮助信息时，如果不知道其属于哪个包，则可以先根据类索引进行查找，打开类的帮助后，可以反向得知它属于哪个包。

根据包索引，每一个包中包含的主要内容大致如下所示：

- Interfaces（接口类）；
- Classes（类）；
- Enums（枚举值）；
- Exceptions（异常）。

每个包中包含的内容，基本上是 Java 语言中标准的内容。

Android 的参考文档中的类是 Android 系统 API 的主要组成部分，主要参考的内容包括了以下的内容。

根据类索引，每一个类中包含的主要内容大致如下所示：

- 扩展和实现的内容；
- 按包名的继承（扩展）关系（可用于反向查找这个类所在的包）；
- Overview（概览）；
- XML Attributes（XML 属性）；
- Constants（常量）；
- Constructors（构造方法）；
- Methods（方法）。

在这些帮助内容中，大部分是 Java 语言的基本语法内容，只有 XML Attributes（XML 属性）一项是 Android 专用的。某些重要的类还包含对于类的详细介绍的图表。

例如，Activity 类的帮助文档的前面的信息如下所示：

```
public class
```

```
Activity
extends ContextThemeWrapper
implements ComponentCallbacks KeyEvent.Callback LayoutInflater.Factory
        View.On CreateContextMenuListener Window.Callback
java.lang.Object
   ↳android.content.Context
     ↳ android.content.ContextWrapper
       ↳ android.view.ContextThemeWrapper
         ↳android.app.Activity
Known Direct Subclasses
ActivityGroup, AliasActivity, ExpandableListActivity, ListActivity

Known Indirect Subclasses
LauncherActivity, PreferenceActivity, TabActivity
```

从 Activity 类中可以看出，类的帮助文档主要包含以下一些内容：

- public class：表示只是一个公开的类；
- extends [……]：标明了这个类继承的父类（Java 不支持多继承，因此每个类只有一个唯一的父类），后面的内容表示这个类从祖先开始继承的关系。这里的类使用的是包含了其所在包名称的全名，因此在这里可以得知类及其祖先类属于哪个包；
- Implements [……]：标明了这个类实现的接口（可以有多个）；
- Known Direct Subclasses [……]：这个类的直接继承者；
- Known Indirect Subclasses [……]：这个类的间接继承者。

从中，可以看出 Activity 类在 android.app 包中，直接继承了 android.view.ContextThemeWrapper，并且被 ActivityGroup, ListActivity 等几个类直接继承。被 LauncherActivity 等几个类间接继承。

类的介绍的主要内容在后面，主要部分是各个类的方法的说明，这些方法也是在类的使用过程中需要主要关注的内容。

```
Class Overview
   （类的介绍）
Summary
   Constants
   （常量的列表）
   Inherited Constants
   （继承的常量的列表，按照继承类的顺序）
   Public Constructors
   （公共的构造函数）
   Public Methods
   （公共方法的列表）
   Protected Methods
   （保护方法的列表）
   Inherited Methods
   （继承方法的列表，按照继承类的顺序）
```

（详细的介绍）

类的帮助中一般只列出了自己的常量、方法、XML 属性等，对于继承得到的内容（包括方法和常量），按照继承的顺序列出。由于 JAVA 类是单向继承，因此在这个部分，首先是父类、然后是祖父类，以此类推。

某些与 UI 内容相关的类的帮助文档有一些特殊，主要区别是包含了 XML attributes（XML 属性）一类。XML Attributes（XML 属性），是出现在 AndroidManifest.xml 或者布局文件中（*.xml）的属性。

例如 Button 类的参考文档的主要内容如下所示：

```
public class
Button
extends TextView

java.lang.Object
   ↳ android.view.View
     ↳ android.widget.TextView
       ↳ android.widget.Button
Known Direct Subclasses
CompoundButton
CompoundButton A button with two states, checked and unchecked.


Known Indirect Subclasses
CheckBox, RadioButton, ToggleButton

XML attributes
See Button Attributes, TextView Attributes, View Attributes
Summary
    Inherited XML Attributes
        From class android.widget.TextView
        From class android.view.View
    Inherited Constants
    Public Constructors
    Inherited Methods
```

Button 类的头部信息和普通的类基本相同，但是包含了 XML attributes 一个项目，在这里包含了 Button Attributes，TextView Attributes，View Attributes，根据类的继承关系可以得知，这个启示是自己的属性、父类的属性（Button 类的父类是 android.widget.TextView）、祖父类的属性（Button 类的祖父类是 android.view.View）。

Button 类刚好没有自己的 XML 属性，但是其父类和祖父类有，展开 Inherited XML Attributes 项目的 From class android.widget.TextView 和 From class

android.view.View 可以得到这些属性的列表。每个属性包含了 Attribute Name（属性名称） Related Method（相关方法） Description（描述）几个项目。

例如，TextView 的几个属性如下所示：

```
    android:text         setText(CharSequence)          Text to display.
    android:textColor    setTextColor(ColorStateList)   Text color.
```

android:text 等表示了属性在 XML 文件中的名称，setText()等表示了在 JAVA 源文件中使用的方法，最右侧的内容是这个属性的描述。

点击 XML attributes 中的连接可以进入其详细的内容中查看，这些 XML 属性的帮助以及相关的值可以在 android.R.styleable 类中查找，这个类也可以直接被调出，方法为：

Package Index → android → android.R.styleable

android.R.styleable 中列出了一些类的 XML 属性，例如 TextView 的 capitalize 属性的相关内容如下所示：

```
    public static final int TextView_capitalize

    If set, specifies that this TextView has a textual input method and should
automatically capitalize what the user types. The default is "none".

    Must be one of the following constant values.

    Constant Value Description
        none         0 Don't automatically capitalize anything.
        sentences    1 Capitalize the first word of each sentence.
        words        2 Capitalize the first letter of every word.
        characters   3 Capitalize every character.

    This corresponds to the global attribute resource symbol capitalize.

    Constant Value: 44 (0x0000002c)
```

这里列出了属性的值（Value），这些值的本质是整数常量，但是在 XML 中使用的还是名称。整数值是 Android 内部运作使用的。

XML 属性有些是在布局文件中使用的，也有在 AndroidManifest.xml 中使用的，或者在其他的 XML 文件中使用。

这在 android.R.styleable 的帮助信息中，以 AndroidManifest 为开头的内容是在 AndroidManifest.xml 中使用的属性。

例如，AndroidManifestAction 项目是 AndroidManifest.xml 中的 Action 标签中使用的内容，如下所示：

```
    public static final int[] AndroidManifestAction
```

```
    Attributes that can be supplied in an AndroidManifest.xml action tag, a child
of the intent-filter tag. See addAction(String) for more information.

    Includes the following attributes:

    Attribute Summary
    android:name The name of an action that is handled, using the Java-style naming
convention.

    See Also
    AndroidManifestAction_name
```

Attribute Summary 中的 android:name 引用的内容是 AndroidManifest.xml 中的 Action 标签可以使用的 android:name 属性。

相比各种类的帮助信息，接口（Interface）的帮助信息更加简单一些。一般的接口是需要被实现才能够使用的。

例如，View.OnClickListener 的帮助信息前面的内容如下所示：

```
    public static interface
    View.OnClickListener
    android.view.View.OnClickListener

    Known Indirect Subclasses
    CharacterPickerDialog, KeyboardView
```

这里的，android.view.View.OnClickListener 表示了 View.OnClickListener 这个接口在 android.view 这个包中。对于一个接口，Indirect Subclasses 的含义为实现（implements）这个接口。

View.OnClickListener 的帮助信息后面的内容同样列出这个接口中包含的成员方法，如下所示：

```
    Summary
       Public Methods
          abstract void  onClick(View v) Called when a view has been clicked.
```

这些方法是要求接口的实现者来实现的，如果一个类实现了 View.OnClickListene 这个接口，其中就必须要有这个接口的 onClick()函数。

第 4 章 Android 应用程序示例

在软件开发的最初阶段，通常使用一个 Hello World 程序作为最简单的示例，本部分介绍一个 Android 中最简单应用程序，通过这部分内容可以了解到 Android 程序的文件结构和编译后的结构。

4.1 HelloActivity 程序的运行

HelloActivity 是一个简单的 Android 应用程序，其工程文件名称为 HelloActivity，在 Android 的源代码和 SDK 中，都包含了这个包。

HelloActivity 的图标和运行情况如图所示。

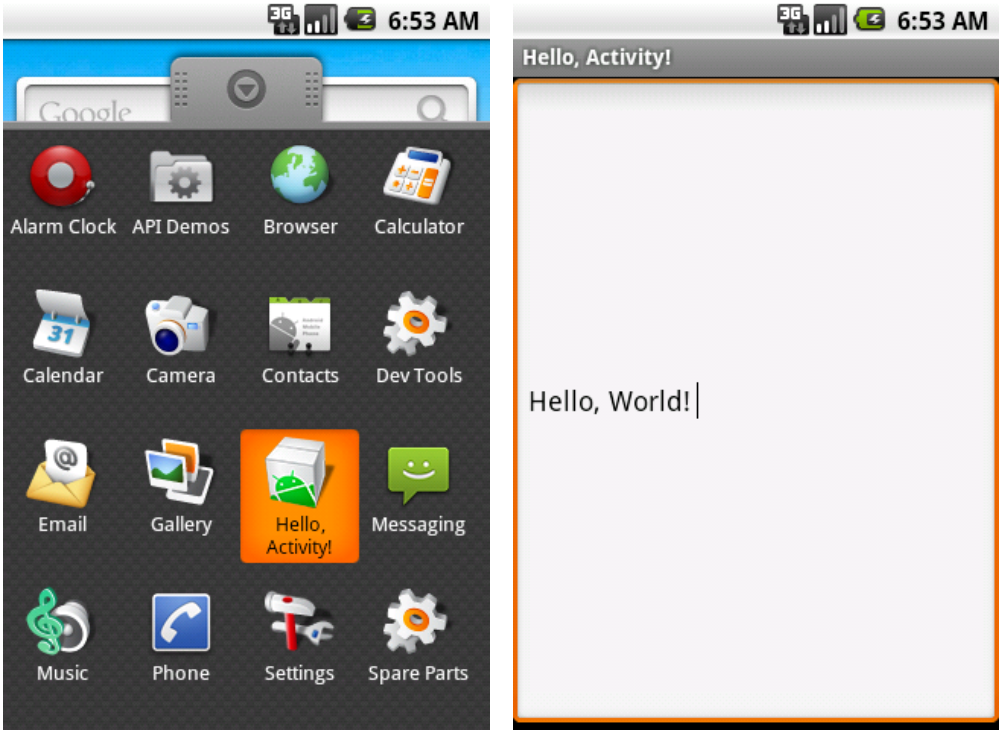


图　HelloActivity 的图标和运行情况

这个程序有一个简单的活动（Activity），用于启动一个新的界面，并在界面上显示 “Hello，World!”字符串。

4.2 HelloActivity 的源文件结构

HelloActivity 工程的源文件的结构按照目录树的方式如下所示：

```
HelloActivity/
|-- Android.mk                                  （工程管理文件）
|-- AndroidManifest.xml                         （工程描述文件）
|-- res                                         （资源文件）
|    |-- layout
|    |    `-- hello_activity.xml                （布局文件）
|    `-- values
|         `-- strings.xml                       （字串资源文件）
|-- src                                         （Java 源代码文件）
    `-- com
        `-- example
            `-- android
                `-- helloactivity
                    `-- HelloActivity.java
```

HelloActivity 工程中另有一个 tests 目录，其中也具有自己的 Android.mk 和 AndroidManifest.xml 文件，这是另一个工程，是 HelloActivity 工程的测试程序。

4.2.1. Android.mk 文件

Android.mk 文件是 Android 的工程管理文件，这个文件只在源代码开发的时候使用，在 SDK 的开发中不需要使用，它包含在工程的根目录中，其内容如下所示：

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE_TAGS := samples

# Only compile source java files in this apk.
LOCAL_SRC_FILES := $(call all-java-files-under, src)

LOCAL_PACKAGE_NAME := HelloActivity
```

```
LOCAL_SDK_VERSION := current

include $(BUILD_PACKAGE)

# Use the following include to make our test apk.
include $(call all-makefiles-under,$(LOCAL_PATH))
```

Android.mk 文件是 Android 编译过程中通用的工程管理文件，本地程序、本地库和 Java 程序包都使用这个文件。这个文件仅仅在基于源代码开发的情况中使用，在 Java 应用程序工程的管理中，该文件不用定义过多的内容，其中关键的内容是使用 include $(BUILD_PACKAGE)表示从当前目录编译 Java 应用程序包。LOCAL_PACKAGE_ NAME 定义的是这个程序的 APK 包的名称。LOCAL_MODULE_TAGS 表示这个包的类型。

这个包的 LOCAL_MODULE_TAGS 定义成了 samples，这将编译 APK 包，但是不安装在系统中。使用不同的值，可以决定是否编译和安装，例如使用 eng，将安装到目标系统中。

最后一行的 include $(call all-makefiles-under，$(LOCAL_PATH))，表示包含本目录的子目录中的 Android.mk 文件，本例中也就是 tests 目录中的内容。

4.2.2. AndroidManifest.xml 文件

AndroidManifest.xml 文件是这个 Android 应用程序的工程描述文件，包含了宏观上的内容，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.helloactivity">
   <application android:label="Hello, Activity!">
      <activity android:name="HelloActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
         </intent-filter>
      </activity>
   </application>
</manifest>
```

application（表示应用程序）标签中包含了一个 activity（表示活动）。活动是应用程序中的一个组件，一个应用程序中也可以包含若干个组件。包名定义为 com.example. android.helloactivity ， 表 示 将 从 src 目 录 的

com/example/android/helloactivity 中寻找程序中的 Java 源代码。活动名称将被定义为 HelloActivity，表示活动的代码是上述源代码目录中的 HelloActivity.java 文件。intent-filter 中的内容指定了程序的启动方式，这里 category 中的 android.intent.category.LAUNCHER 表示活动将在 Android 的桌面（Android 默认的桌面程序名称也是 LAUNCHER）上出现。

这里指定 application 的 android:label 为"Hello，Activity!"，这和桌面图标下面的文字以及活动启动后上面的标题文字是一致的。本例没有指定图标，所以桌面上的图标使用的是默认图标。

在 AndroidManifest.xml 文件中为一个活动指定 label（标签）和 icon（图标）的方法　　如下所示：

```
    <activity android:name="HelloActivity"
           android:label="@string/label_name"
           android:icon="@drawable/icon_name" >
        <intent-filter>
           <action android:name="android.intent.action.MAIN"/>
           <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
```

android:label 指定为字符串、android:icon 指定为图标后，将使用 res/drawable 中对应名称的图片文件作为图标（本例中将使用 icon_name.png）。

activity 和 application 都具有 android:label 和 android:icon 等属性，由于活动是程序的单元，且应用可以包含多个活动，因此程序首先将使用 activity 中的这些标签，如果没有则使用上一级的 application 中标签的定义

4.2.3. 源代码文件

HelloActivity 工程只有一个源代码文件 HelloActivity.java，位于这个工程 src 目录下的 com/example/android/helloactivity 中，内容如下所示：

```
    package com.example.android.helloactivity;            // 定义包名
    import android.app.Activity;                          // 引入包含的包
    import android.os.Bundle;
    public class HelloActivity extends Activity {
        public HelloActivity() {  }
        @Override
        public void onCreate(Bundle savedInstanceState) {  // 重载 onCreate()方法
            super.onCreate(savedInstanceState);
            setContentView(R.layout.hello_activity);  // 使用 hello_activity.xml
布局文件
```

```
        }
    }
```

这里的类 HelloActivity 继承实现了 Android 系统 API 提供的活动类（Activity），使用 setContentView(R.layout.hello_activity)指定了当前活动的布局，这里表示将从 res/layout 目录中找到 hello_activity.xml 文件作为本例的布局文件使用。

4.2.4. 布局文件

hello_activity.xml 是本程序中的布局文件，在 Java 源文件中使用了此文件。本文件在 res/layout 目录中，其内容如下所示：

```
    <?xml version="1.0" encoding="utf-8"?>
    <EditText
    xmlns:android="http://schemas.android.com/apk/res/android"
ndroid:id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textSize="18sp"
    android:autoText="true"
    android:capitalize="sentences"
    android:text="@string/hello_activity_text_text" />
```

在这个布局文件中，只定义了一个 UI 元素——EditText，就是在界面上出现的占据全屏的可编辑文本框。在这里定义了这个可编辑文本框的初始化字符串为"@string/hello_activity_text_text"，这个值在另外的资源文件中被定义，本例就是 string.xml。

4.2.5. 其他资源文件

string.xml 是本例中的一个资源文件，其内容如下所示：

```
    <?xml version="1.0" encoding="utf-8"?>
    <resources>
        <string name="hello_activity_text_text">Hello, World!</string>
    </resources>
```

这里定义了名称为“hello_activity_text_text”的字符串的内容为 Hello，World!，这就是出现在屏幕上的字符串。

4.3 HelloActivity 的编译结构

在 Android 的 SDK 环境开发中，HelloActivity 工程经过编译后，SDK 环境下开发生成的所有目标文件均在当前工程目录中，包含了 assets、bin、gen 等目录。

在 gen 目录中，包含了以类的层次关系为结构的资源文件。例如，gen/com/example/android/helloactivity 目录中的 R.java 就是 HelloActivity 中的资源文件。

在 bin 目录中，目录结构按照类的关系组织，com/example/android/helloactivity 子目录包含了经过编译后的各个 Java 类，以.class 为后缀。

在 bin 目录中包含的 classes.dex 文件是编译后的，可以在 Dalvik 虚拟机上运行的 Java 的字节码文件，生成的 HelloActivity.apk 文件是最终的 APK 文件，可以在兼容的 Android API 的目标系统中安装，进而运行程序。

HelloActivity.apk 经过解压缩后，包含了下面的一些内容：

```
HelloActivity.apk/
|-- AndroidManifest.xml              （经过 aapt 处理的工程描述文件）
|-- META-INF
|     |-- CERT.RSA
|     |-- CERT.SF
|     `-- MANIFEST.MF
|-- classes.dex                      （Dalvik 的字节码）
|-- res
|     `-- layout
|           `-- hello_activity.xml   （经过 aapt 处理的布局文件）
`-- resources.arsc
```

4.4 SkeletonApp 的程序的运行

SkeletonApp 是 Android 中一个应用程序的框架，这个程序比 HelloActivity 复杂一些，这个程序的运行结果如图所示：

图 SkeletonApp 程序的运行

这个程序包含了两个按钮和菜单，两个按钮分别用于清除编辑文本框中的内容，菜单的功能和两个按钮时是相同的，点击菜单按钮将出现菜单，菜单是 Android 中的标准组件。

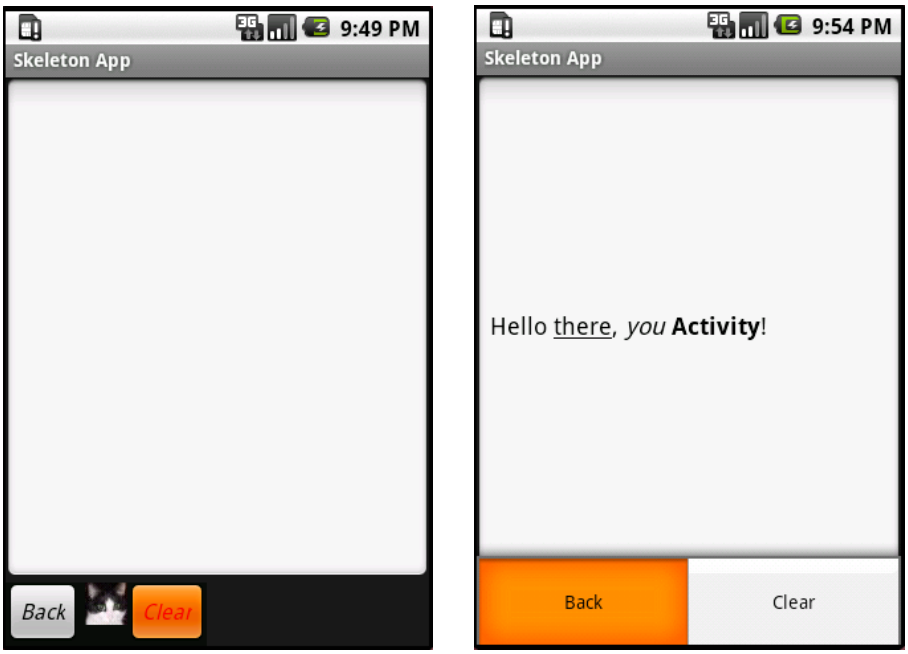


图 使用 SkeletonApp 程序中的按钮和菜单

4.5 SkeletonApp 的源文件结构

SkeletonApp 工程的源文件的结构按照目录树的方式如下所示：

```
SkeletonApp/
|-- Android.mk                          （工程管理文件）
|-- AndroidManifest.xml                 （工程描述文件）
|-- res                                 （资源文件）
|    |-- drawable
|    |    `-- violet.jpg                （图片文件）
|    |-- layout
|    |    `-- skeleton_activity.xml     （布局文件）
|    `-- values
|         |-- colors.xml                （颜色资源文件）
|         |-- strings.xml               （字串资源文件）
|         `-- styles.xml                （样式资源文件）
|-- src                                 （Java 源代码文件）
```

```
`-- com
    `-- example
        `-- android
            `-- skeletonapp
                `-- SkeletonActivity.java
```

在 SkeletonApp 中，资源目录 res 中的 values 目录中除了 strings.xml 文件，还包含了 colors.xml 和 styles.xml 文件，这两种文件也是 Android 中的标准资源文件。

colors.xml 文件的内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Retrieved via Resources.getColor() and friends. -->
    <color name="red">#f00</color>
    <!-- Retrieved via Resources.getDrawable() and friends. -->
    <drawable name="semi_black">#80000000</drawable>
</resources>
```

styles.xml 文件的内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="ActionButton">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textAppearance">
                @style/TextAppearance.ActionButton</item>
    </style>
    <style name="TextAppearance" parent="android:TextAppearance">
    </style>
    <style name="TextAppearance.ActionButton">
        <item name="android:textStyle">italic</item>
    </style>
</resources>
```

资源目录 res 还包含了 drawable 目录，表示可以绘制的内容，这里的 violet.jpg 是一个 jpeg 的文件。

在布局文件 skeleton_activity.xml 中的部分内容引用了以上的资源

```
    <LinearLayout
        android:layout_width="wrap_content"
android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:gravity="center_horizontal"
        android:orientation="horizontal"
        android:background="@drawable/semi_black">
        <Button android:id="@+id/back" style="@style/ActionButton"
            android:text="@string/back" />
```

```
        <ImageView android:id="@+id/image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingLeft="4dip" android_paddingRight="4dip"
            android:src="@drawable/violet" />
        <Button android:id="@+id/clear" style="@style/ActionButton"
            android:text="@string/clear" android:textColor="@color/red" />
    </LinearLayout>
```

布局文件中引用了上面的资源，颜色可以作为字体的颜色，style/ActionButton 作为按钮的样式，drawable/semi_black 表示了背景的内容，drawable/violet 表示引用 violet.jpg 图片作为图像的内容。例如根据上面 styles.xml 文件中的定义，两个按钮上的字体为斜体，第二个按钮的字体红色。

JAVA 源代码 SkeletonActivity.java 中构建了菜单、按钮的动作等功能。

4.6 SkeletonApp 的编译结构

Android 中程序的编译结构基本类似，SkeletonApp 的应用程序包 SkeletonApp.apk 经过解压缩后，包含了下面的一些内容：

```
SkeletonApp.apk/
|-- AndroidManifest.xml          （经过 aapt 处理的工程描述文件）
|-- META-INF
|    |-- CERT.RSA
|    |-- CERT.SF
|    `-- MANIFEST.MF
|-- classes.dex                  （Dalvik 的字节码）
|-- res
|    |-- drawable
|    |    `-- violet.jpg         （保持原状的图片文件）
|    `-- layout
|         `-- skeleton_activity.xml   （经过 aapt 处理的布局文件）
`-- resources.arsc
```

在这里 drawable 中图片文件保持原状，layout 中的布局文件经过 aapt 处理成为压缩的文本文件，其他的资源文件在最终的程序包中，不再单独存在。

第 5 章 Android 应用程序的内容

5.1 Android 应用程序的概念性描述

Android 应用程序包含了工程文件、代码和各种资源，主要由 Java 语言编写，每一个应用程序将被编译成 Android 的一个 Java 应用程序包（*.apk）。

由于 Android 系统本身是基于 Linux 操作系统运行的，因此 Android 应用程序也运行于 Linux 环境中，它们具有以下的特点：

- 在默认情况下，每一个应用程序运行于它们的 Linux 进程中；
- 每个进程具有自己的虚拟机（VM），所以每个应用程序运行于独立的环境中；
- 在默认情况下，每一个应用程序具有唯一的 Linux 用户 ID。通过设置权限让应用程序只对用户和应用程序本身可见，也有一些方法可以把它们暴露给其他的应用程序。

5.1.1. 应用程序的组成部分

一般情况下，Android 应用程序由以下 4 种组件构成：

- 活动（Activity）；
- 广播接收器（BroadcastReceiver）；
- 服务（Service）；
- 内容提供者（Content Provider）。

一个 Android 应用程序是一个包(Package)，包中可能包含一个或者多个 Android 组件（component）。

（1）活动（Activity）

活动是最基本的 Android 应用程序组件，在应用程序中，一个活动通常就是一个单独的用户界面。每一个活动都被实现为一个独立的类，并且从活动（Activity）基类中继承而来，活动类将会显示由视图（View）控件组成的用户接口，并对事件（Event）做出响应。大多数的应用程序都会有多个用户界面，因此便会有多个相应的活动。

Android 的一个活动一般对应界面中的一个屏幕显示，可以理解成一个界面，每一个活动在界面上可以包含按钮、文本框等多种可视的 UI 元素。

（2）广播接收器（BroadcastReceiver）

广播接收器用于让应用程序对一个外部事件做出响应。例如：电话呼入事件、数据网络可用通知或者到了晚上时进行通知。

（3）服务（Service）

一个服务是一个具有一段较长生命周期但没有用户界面的程序。例如：一个正在从播放列表中播放歌曲的媒体播放器在后台运行。

（4）内容提供者（Content Provider）

应用程序能够将它们的数据保存到文件或 SQLite 数据库中，甚至是任何有效的设备中。当需要将数据与其他的应用共享时，内容提供者将会很有用。一个内容提供者类实现了一组标准的方法，从而能够让其他应用程序保存或读取此内容提供者处理的各种数据类型。

5.1.2. 应用程序的生命周期

Android 系统中的不同组件具有不同的生命周期。Android 根据每个进程中运行的组件以及组件的状态把进程放入一个重要性分级（importance hierarchy）中。Android 进程的重要性分级，可以理解成执行的优先级。

Android 进程的类型包括（按重要性分级排序）：

（1）前台（Foreground）进程

与用户当前正在做的事情密切相关，不同的应用程序组件能够通过不同的方法使它的宿主进程移到前台。当下面任何一个条件满足时，都可以考虑将进程移到前台。

- 进程正在屏幕的最前端运行一个与用户交互的 Activity（它的 onResume()方法被调用）；
- 进程有一个正在运行的 BroadcastReceiver（它的 BroadcastReceiver.onReceive()方法正在执行）；
- 进程有一个 Service，并且在 Service 的某个方法（Service.onCreate()、Service.onStart()或者 Service.onDestroy()）内有正在执行的代码。

（2）可见（Visible）进程

它有一个可以被用户从屏幕上看到的 Activity，但不在前台——其 onPause()方法被调用。例如：如果前台的 Activity 是一个对话框，以前的 Activity 隐藏在

对话框之后，就可能出现这种进程。这样的进程很重要，一般不允许被杀死，除非为了保证前台进程的运行不得不这样做。

（3）服务（Service）进程

有一个已经用 startService() 方法启动的 Service，虽然这些进程用户无法直接看到，但它们做的事情却是用户所关心的（例如：后台 MP3 回放或后台网络数据的上传/下载）。因此，系统将一直运行这些进程，除非内存不足以维持所有的前台进程和可见进程。

（4）后台（Background）进程

拥有一个当前用户看不到的 Activity（它的 onStop()方法被调用），这些进程对用户体验没有直接的影响。如果它们正确执行了 Activity 生命周期，系统可以在任意时刻杀死进程来回收内存，并提供给前面 3 种类型的进程使用。系统中通常有很多这样的进程在运行，因此要将这些进程保存在 LRU 列表中，以确保当内存不足时用户最近看到的进程最后一个被杀死。

（5）空（Empty）进程

不包含任何处于活动状态的应用程序组件。保留这种进程的唯一原因是，当下次应用程序的某个组件需要运行时，不需要重新创建进程，这样可以提高启动速度。

以上所说的“进程”是从系统运行的角度考虑的，各种不同的进程可以理解成 Android 的各种组件的不同状态机（state machine）。如果从应用程序的代码以及运行情况考虑，可以关注 Android 的各种组件相对应的生命周期。

1. 活动的生命周期

活动是 Android 中最重要、最基础的组件，用户在界面上看到的一个个可以切换的屏幕界面就是 Android 中的活动。活动的生命周期如图 1 所示。

- 运行活动的情景：当一个活动被启动时，活动中的 onCreate()、onStart()和 onResume()这 3 个方法被依次调用，活动对应的界面出现在屏幕上。
- 活动被“覆盖”的情景：Android 的活动一般都占据一个完整的屏幕，从当前活动启动另外一个活动时，另一个活动将被启动到前台（Foreground），当前活动转入后台（Background），这时活动的 onPasuse()方法将被调用，活动转入后台运行。如果活动变为不可见，还将调用 onStop()方法。在转入后台时，onStop()是否被调用取决于活动是否被完全覆盖，在新的活动有透明部分时，转入后台的活动依然“可见”，其他情况下（较多数的情况）活动

均进入不可见状态（被完全覆盖）。

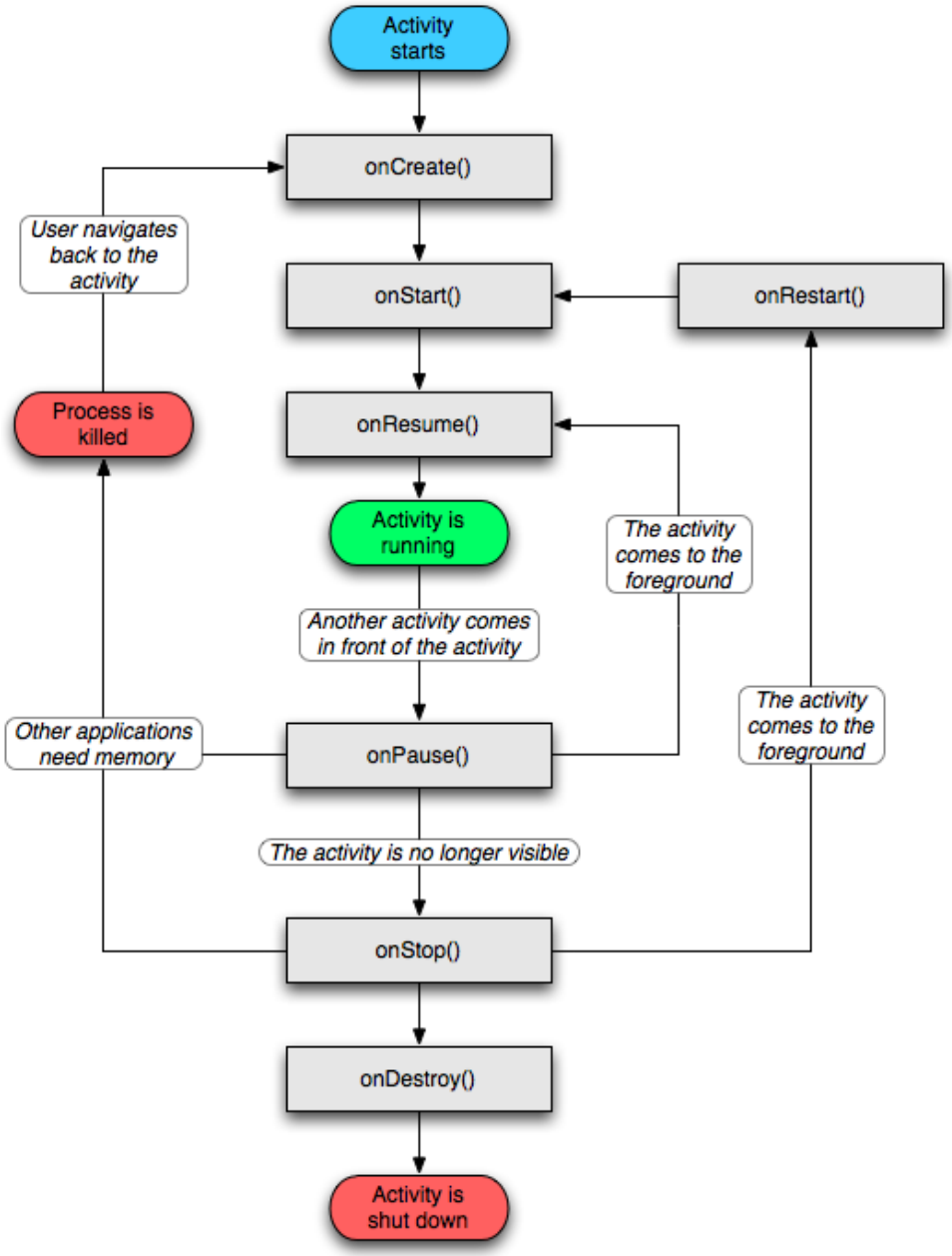


图 活动（Activity）的生命周期

- 活动被恢复的情景：当界面上最前面的活动退出后，它所覆盖的活动将被恢复，这时 onResume()方法将被调用，活动重新转入前台运行。
- 活动完全退出的情景：当使用回退（Back）按钮退出活动时，onDestroy()方法将被调用，活动关闭。如果系统缺少内存时，也会杀死（kill）后台的活动，其中优先杀死不可见的活动，可见的活动一般不会被杀死。

2. 服务的生命周期

服务可以长时间运行，它的特点是没有可视化界面，服务的生命周期如图 2 所示。

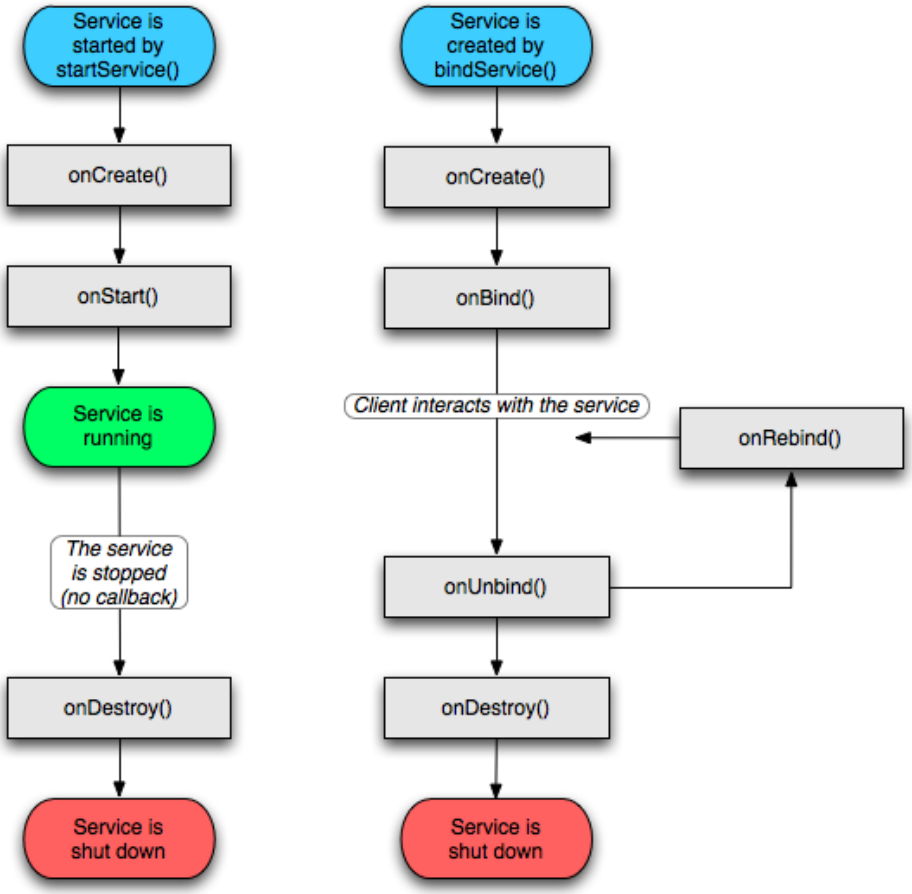


图 服务（Service）的生命周期

使用 StartService 运行服务的情景：使用这种方法启动服务，服务的 onCreate()和 onStart()这两个方法将被调用，服务会在后台运行直到退出，退出时将调用 onDestroy()方法。

使用 bindService 运行服务的情景：使用这种方法启动服务，调用者（也就是服务的客户端）将获得和服务交互的类，通过其调用时服务的相关内容会处于活动状态。

3. 广播接收器的生命周期

广播接收器有一个单一的回调方法 onReceive()，当广播消息到达接收器时，Android 将调用这个方法，并传递给包含在这个消息中的 Intent 对象。

广播接收器只有在这个方法的执行过程中才处于活动状态，当 onReceive()返回后，广播接收器将不再处于活动状态。广播接收器的功能类似于一个回调函数，只是单次运行时处于活动状态。

5.2 应用程序包含的各个文件

Android 应用程序一般包含在一个单一的文件夹中，即每一个 Android 应用程序是一个独立的工程，包含了以下文件：

- Android.mk：统一工程文件，在 SDK 开发中可以不需要；
- AndroidManifest.xml：工程描述文件，在其中定义了各种组件；
- Java 源代码：按照 Java 包的方式来组织目录结构，包括各个 Java 类的源代码；
- 资源文件：包含 XML 文件、图片、原始数据文件等，其中表示界面情况的布局（Layout）文件比较重要。

在编译 Android 应用程序的过程中，Java 源代码使用 Sun JDK 将 Java 源程序编译成 Java 字节码文件（多个后缀名为.class 的文件），这一步骤和标准的 Java 一致，然后通过 Android 自带的工具软件 dex 把所有的字节码文件转成 dex 文件（单一文件 classes.dex）。

AndroidManifest.xml 文件经过 Android 打包工具（aapt）处理后形成二进制格式 AndroidManifest.xml 文件，实质的内容与以前相同。

各个资源文件也经过 aapt 处理，其中布局等文本文件处理成二进制文件，图片等文件保持不变。

最后将这三个部分组合成一个应用程序包（*.apk）。AndroidManifest.xml 描述文件、Java 源文件、资源文件是 Android 应用程序的三个部分；在编译之前的工程中是这三个部分，在编译之后 APK 包依然是由这三个部分组成的。

Android 应用程序的编译过程如图所示：

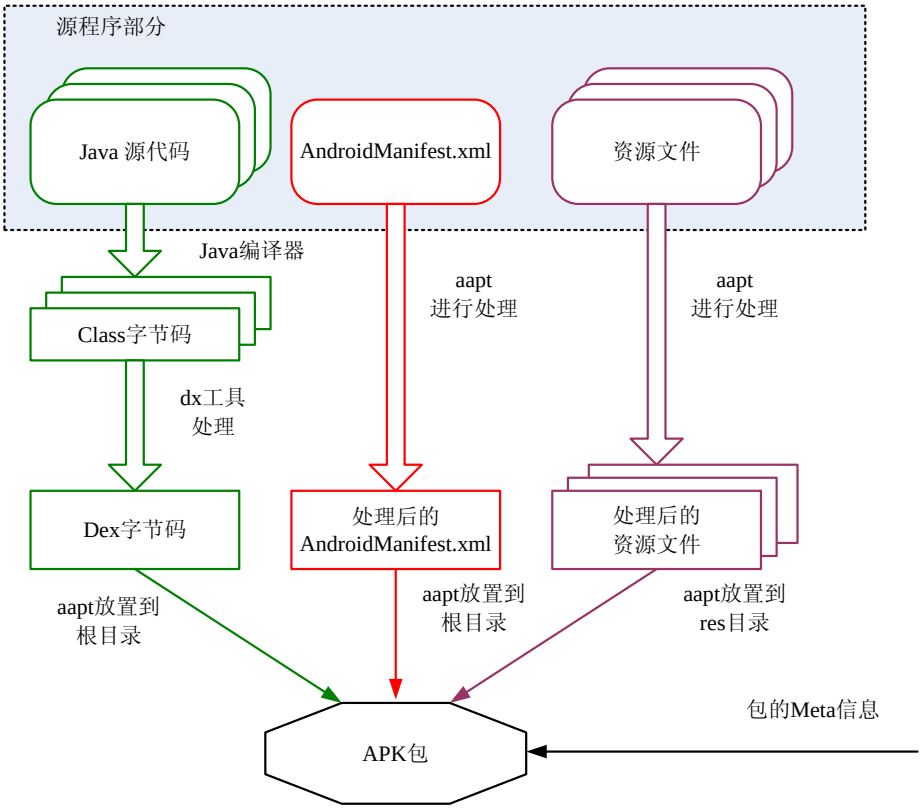


图 Android 应用程序的编译过程

如图所示，Android 源文件经过了标准的 Java 编译器的编译，又经过了 dx 工具的处理，标准的 Java 字节码作为整个 Android 编译的中间过程，最终生成的 dex 文件（classes.dex）是一个单一文件，将工程中所有的 Java 源代码文件对应的字节码集成在一起。资源文件和 AndroidManifest.xml 文件通过 aapt 工具进行处理。

在运行时，APK 包将首先进行“安装”，也就是将其中的 dex 文件进行优化，优化后的文件被保存到缓存区域，生成格式为 dey 的优化文件，然后 Dalvik 虚拟机将运行这些 dey 文件。如果应用程序包文件不发生变化，dey 文件不会被重新生成；在应用程序包发生更新的情况下，将重新由 dex 生成 dey。

Android 和标准 JAVA 开发的 JAR 包最大的不同在于，标准 JAVA 字节码是每个文件一个 Clash 文件，而 Android 中的一个包将生成一个 Dex 文件。

5.3 使用 am 工具启动 Android 应用程序

除了在 GUI 界面中启动应用程序之外，在 Android 的命令行终端（可以使用 adb shell 进行连接）也可以使用 am 工具启动应用程序。

am 命令的基本使用方法如下所示：

```
usage: am [start|broadcast|instrument]
       am start -D INTENT
       am broadcast INTENT
       am instrument [-r] [-e <ARG_NAME> <ARG_VALUE>] [-p <PROF_FILE>]
               [-w] <COMPONENT>
```

使用 am start 是其中的一个功能，INTENT 使用的选项如下所示：

```
[-a <ACTION>] [-d <DATA_URI>] [-t <MIME_TYPE>]
[-c <CATEGORY> [-c <CATEGORY>] ...]
[-e|--es <EXTRA_KEY> <EXTRA_STRING_VALUE> ...]
[--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE> ...]
[-e|--ei <EXTRA_KEY> <EXTRA_INT_VALUE> ...]
[-n <COMPONENT>] [-f <FLAGS>] [<URI>]
```

主要的参数是使用-a 指定使用的动作（action），使用-d 指定数据（data），使用 URI 的格式，使用-n 指定组件。

例如：使用 am 启动应用程序的格式如下所示：

```
# am start -n {包名}/{包名}.活动名
```

启动 Android 设置工具的命令如下所示：

```
# am start -n com.android.settings/com.android.settings.Settings
```

启动 Android 计算器程序的命令如下所示：

```
# am start -n com.android.calculator2/com.android.calculator2.Calculator
```

启动 Android 录音机程序的命令如下所示：

```
#                    am                    start                    -n
com.android.soundrecorder/com.android.soundrecorder.SoundRecorder
```

启动 Android 照相机程序的命令如下所示：

```
# am start -n com.android.camera/com.android.camera.Camera
```

启动 Android 摄像机程序的命令如下所示：

```
    # am start -n com.android.camera/com.android.camera.VideoCamera
```

启动 Android 音乐浏览器的命令如下所示：

```
    # am start -n com.android.music/com.android.music.MusicBrowserActivity
```

启动 Android 视频浏览器的命令如下所示：

```
    # am start -n com.android.music/com.android.music.VideoBrowserActivity
```

启动 Android 网络浏览器等的命令如下所示：

```
    # am start -n com.android.browser/com.android.browser.BrowserActivity
```

在上面的程序中，有些程序位于同一个包中，例如：音乐浏览器和视频浏览器都在 Music 包中，照相机和摄像机都在 Camera 包中。

对于某些具有附加数据的应用程序，还可以使用-d 选项增加数据 URL，示例如下所示：

```
    # am start -n com.android.music/com.android.music.MediaPlaybackActivity -d
/a.mp3
    # am start -n com.android.music/com.android.music.MediaPlaybackActivity -d
file: ///a.mp3
    #   am   start   -n   com.android.camera/com.android.camera.MovieView   -d
file:///b.mp4
    # am start -n com.android.camera/com.android.camera.MovieView -d /b.mp4
    #   am   start   -n   com.android.camera/com.android.camera.ViewImage   -d
file:///c.jpg
```

以上程序分别进行了音乐播放、视频播放、图片浏览等功能。com.android.music. MediaPlaybackActivity 、 com.android.camera.MovieView 和 com.android.camera.ViewImage 分别是对应的应用程序。

对于上述内容，还可以使用 mime type 方式启动程序，如下所示：

```
    # am start -a android.intent.action.VIEW -d file:///a.mp3 -t audio/*
    # am start -a android.intent.action.VIEW -d file:///b.mp4 -t video/*
    # am start -a android.intent.action.VIEW -d file:///c.jpg -t image/*
```

这里使用的是 am -a 参数，表示执行一个动作，后面的 audio/*、video/*和 image/*表示数据 mime 类型，Android 将自动找到支持相应数据 mime 类型的程序来打开对应的音乐、视频和图片文件。

第三篇 Android 的 UI 系统实现

第 6 章 UI 的基本外形和控制

Android UI 系统的知识结构如下图所示：

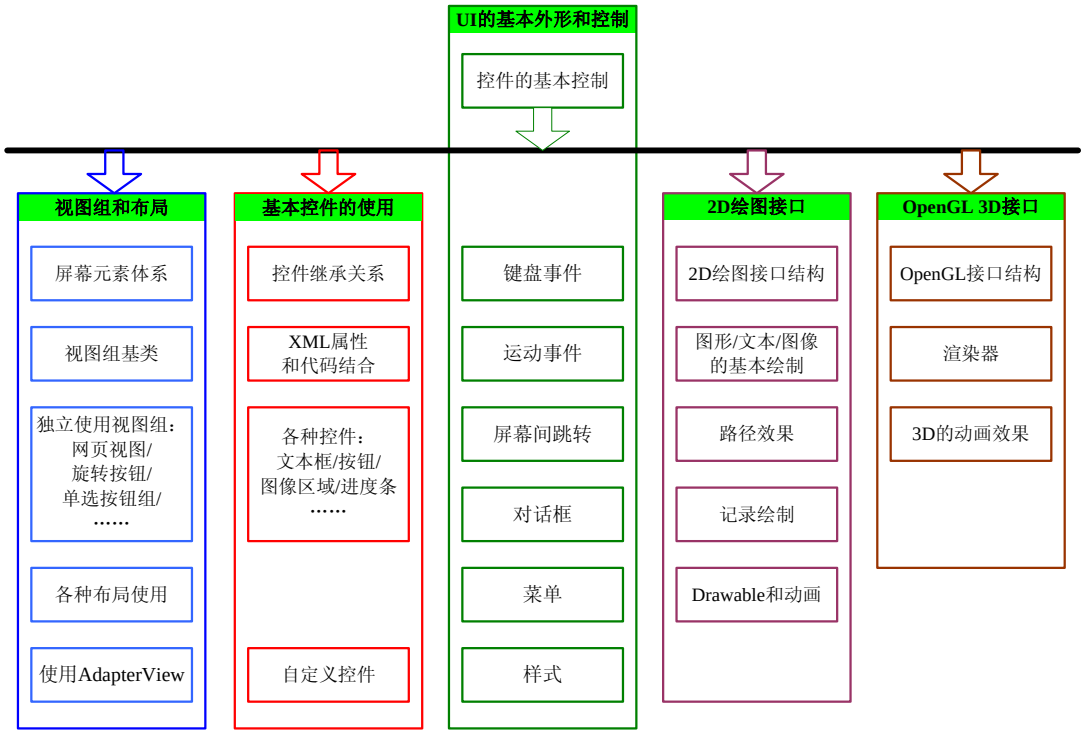


图　Android UI 系统的知识结构

对于 一个 GUI 系统地使用，首先是由应用程序来控制屏幕上元素的外观和行为，这在各个 GUI 系统中是不相同的，但是也具有相通性。Android 系统在这方面，包含了基本的控件控制，键盘事件响应，窗口间跳转、对话框、菜单、样式等内容，这是 GUI 系统所具有的通用内容。

6.1　控件和基本事件的响应

在任何一个 GUI 系统中，控制界面上的控件（通常称为控件）都是一个基本的内容。对于 Android 应用程序，控件称为 View。

在 Android 中，在处理 UI 中的各种元素的时候，两个程序中的要点为：

- 得到布局文件（XML）中的控件句柄
- 设置控件的行为

本小节介绍在 Android 中几种基本的程序控制方法，要获得的效果是通过 2 个按钮来控制一个文本框的背景颜色，其运行结果如图所示：

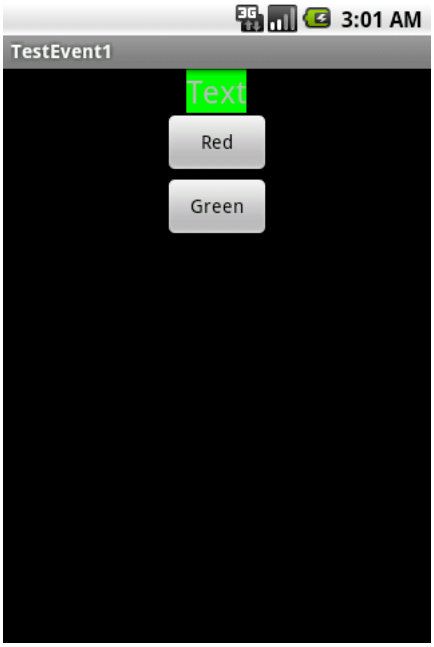


图　控件事件的响应

6.1.1. 事件响应方法

本例构建一个应用程序，其在 AndroidManifest.xml 描述文件中的内容如下所示：

```
        <activity android:name="TestEvent1" android:label="TestEvent1">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
```

本例定义了一个 Android 中基本的活动。

本例的布局文件（layout）的代码片段如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/screen"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
        android:orientation="vertical">
        <TextView android:id="@+id/text1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:textSize="24sp"
            android:text="@string/text1" />
        <Button android:id="@+id/button1"
            android:layout_width="80sp"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="@string/red"/>
        <Button android:id="@+id/button2"
            android:layout_width="80sp"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="@string/green"/>
    </LinearLayout>
```

根据以上的布局文件中定义的两个按钮和一个文本框，这个布局文件被活动设置为 View 后，显示的内容就如上图所示，只是行为还没有实现。

行为将在源代码文件 TestEvent1.java 中实现，这部分的代码如下所示：

```
    package com.android.basicapp;

    import android.app.Activity;
    import android.os.Bundle;

    import android.graphics.Color;
    import android.widget.Button;

    import android.widget.TextView;
    import android.view.View;
    import android.view.View.OnClickListener;

    import android.util.Log;

    public class TestEvent1 extends Activity {
        private static final String TAG = "TestEvent1";
        public TestEvent1() {
        }

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.testevent);
            final TextView Text = (TextView) findViewById(R.id.text1);   // 获得句
柄
            final Button Button1 = (Button) findViewById(R.id.button1);
            final Button Button2 = (Button) findViewById(R.id.button2);
```

```
        Button1.setOnClickListener(new OnClickListener() { // 实现行为功能
            public void onClick(View v) {
               Text.setBackgroundColor(Color.RED);
            }
        });

        Button2.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
               Text.setBackgroundColor(Color.GREEN);
            }
        });

    }
}
```

在创建的过程中，通过 findViewById 获得各个屏幕上面的控件（控件）的背景，这里使用的 R.id.button1 等和布局文件中各个元素的 id 是对应的。实际上，在布局文件中，各个控件即使不写 android:id 这一项也可以正常显示，但是如果需要在代码中进行控制，则必须设置这一项。

根据 Button 控件的 setOnClickListener()设置了其中的点击行为，这个方法的参数实际上是一个 View.OnClickListener 类型的接口，这个接口需要被实现才能够使用，因此在本例的设置中，实现了其中的 onClick()函数。这样既可实现点击的时候实现相应的功能，在点击的函数中，将通过 Text 的句柄对其进行控制。

在 Android 的控件使用方面，这两个编程方面要点是：

- 使用 findViewById()获取布局文件（XML）中控件的句柄；
- 使用 setOnXXXListener()设置事件处理函数。

在获取句柄时需要转换成相应的控件类型，findViewById()函数的参数是一个整数，返回值是一个 android.view.View 类型。通过 R.id.XXX 找到布局文件中定义的 ID，然后通过将基础类转换成其实际的类获得真正的句柄。注意：所转换类必须和布局文件中描述的控件一致。

SetOnXXXListener()等函数是 android.view.View 类的函数，各种控件（包括 Button、EditText）都扩展这个类，同族的函数包括：

```
void  setOnClickListener(View.OnClickListener l);
void  setOnFocusChangeListener(View.OnFocusChangeListener l);
void  setOnKeyListener(View.OnKeyListener l);
void  setOnLongClickListener(View.OnLongClickListener l);
void  setOnTouchListener(View.OnTouchListener l);
```

这些函数用于事件处理，它们由程序实现，通过设置这些内容也就设置了控件的行为。这些函数的参数都是所对应的 android.view.View 类中的方法。

Android 中 UI 基本控制内容：使用 findViewById()联系布局文件中控件和句柄，并通过 OnClickListener()等定制句柄的行为。

6.1.2. 第二种响应方法

除了上述的使用方法，在使用同样的布局文件和应用程序的情况下，实现同样的功能。本例中使用的是另外的一种方式实现。

本例使用的源代码文件如下所示：

```
package com.android.basicapp;

import android.app.Activity;
import android.os.Bundle;

import android.graphics.Color;

import android.widget.Button;
import android.widget.TextView;
import android.view.View;
import android.view.View.OnClickListener;

import android.util.Log;

public class TestEvent2 extends Activity implements OnClickListener {
                                    // 实现相关的接口
    private static final String TAG = "TestEvent2";
    private TextView mText;
    private Button mButton1;
    private Button mButton2;

    public TestEvent2() {
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.testevent);
        mText = (TextView) findViewById(R.id.text1);
        mButton1 = (Button) findViewById(R.id.button1);
        mButton1.setOnClickListener(this);  // 设置监听的类
        mButton2 = (Button) findViewById(R.id.button2);
        mButton2.setOnClickListener(this);  // 设置监听的类
    }
```

```
    public void onClick(View v) {
        Log.v(TAG, "onClick()");
        switch(v.getId()){        // 区分不同的控件
          case R.id.button1:
            mText.setBackgroundColor(Color.RED);
            break;
          case R.id.button2:
            mText.setBackgroundColor(Color.GREEN);
            break;
          default:
            Log.v(TAG, "other");
            break;
        }
    }
}
```

这个例子的主要变化是让活动实现（implements）了 OnClickListener()这个进口，也就是需要实现其中的 onClick()方法。然后通过 setOnClickListener()将其设置到按钮中的 参数就是 this，表示了当前的活动。

通过这种方式的设置，如果程序中有多个控件需要设置，那么所设置的也都是一个函数。为了保证对不同控件具有不同的处理，可以由 onClick()函数的参数进行判断，参数是一个 View 类型，通过 getId()获得它们的 ID，使用 switch…case 分别进行处理。

在本例中，通过将需要将文本框（TextView）句柄保存为类的成员（mText），这样就可以在类的各个函数中都能获得这个句柄进行处理。这和上一种方法是有区别的，因为上一个例子实现的接口和获得的 TextView 在同一个函数中，因此不需要保存 TextView 的句柄。

6.1.3. 第三种响应方法

本例介绍同样功能实现的第三种方法，区别也仅仅在于 JAVA 源代码中，实现的内容如下所示。

```
import android.view.View.OnClickListener;

import android.util.Log;

public class TestEvent3 extends Activity{

    private static final String TAG = "TestEvent3";
    private TextView mText;
    private       Button1 OnClickListener       mListener1       =       new
Button1_OnClickListener();
    private       Button2_OnClickListener       mListener2       =       new
```

```
Button2_OnClickListener();

        public TestEvent3() {
        }

        class Button1 OnClickListener implements OnClickListener { // 接口的第一个
实现
            public void onClick(View v) {
                mText.setBackgroundColor(Color.RED);
            }
        }
        class Button2_OnClickListener implements OnClickListener { // 接口的第一个
实现
            public void onClick(View v) {
                mText.setBackgroundColor(Color.GREEN);
            }
        }

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.testevent);
            mText = (TextView) findViewById(R.id.text1);
            final Button mButton1 = (Button) findViewById(R.id.button1);
            final Button mButton2 = (Button) findViewById(R.id.button2);

            mButton1.setOnClickListener(mListener1);  // 设置监听者的类
            mButton2.setOnClickListener(mListener2);  // 设置监听者的类

        }
    }
```

本例通过定义实现活动类中的 2 个子类，来实现 View.OnClickListener 这个接口，这种方式是一种最为直接的方式，即为不同的控件单独实现它的相应类。

6.2 键盘事件的响应

在应用的程序的控制方面，更多的使用的是屏幕上的控件，但是有的时候也需要直接对键盘事件来进行响应。键盘是 Android 中主要的输入设备，对按键的响应的处理是响应之间在程序中使用键盘的核心内容。

本例需要实现的内容是通过键盘来控制屏幕上的一个图片的 Alpha 值，使用上键和右键增加图片的 Alpha 值，使用下键和左键减少图片的 Alpha 值。显示内容如下所示：

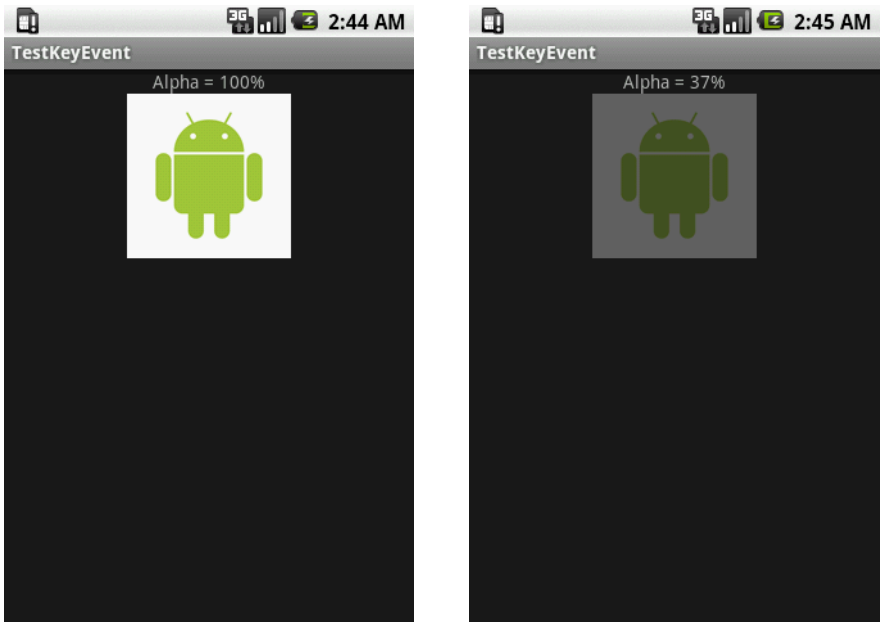


图　按键事件的响应

本例的布局文件 testkeyevent.xml 如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/screen"
    android:layout width="fill parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <TextView android:id="@+id/alphavalue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout gravity="center"/>
    <ImageView android:id="@+id/image"
           android:src="@drawable/robot"
           android:layout gravity="center"
           android:layout_width="wrap_content"
           android:layout height="wrap content" />
</LinearLayout>
```

本例包含了一个文本框和一个显示图片的控件，这样可以文本框用作显示当前的 Alpha 的比例值，显示图片的控件 ImageView 用于显示一个图片。

本例的源代码实现如下所示：

```
package com.android.basicapp;

import android.app.Activity;
import android.content.Context;
```

```
import android.graphics.*;

import android.os.Bundle;
import android.util.Log;

import android.view.KeyEvent;
import android.view.View;
import android.widget.TextView;
import android.widget.ImageView;

public class TestKeyEvent extends Activity {
    private static final String TAG = "TestKeyEvent";
    private ImageView mImage;
    private TextView  mAlphavalueText;
    private int mAlphavalue;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.testkeyevent);
        mImage = (ImageView) findViewById(R.id.image);
        mAlphavalueText = (TextView) findViewById(R.id.alphavalue);
        mAlphavalue = 100;
        mImage.setAlpha(mAlphavalue);
        mAlphavalueText.setText("Alpha = " + mAlphavalue*100/0xff + "%");
    }


    @Override
    public boolean onKeyDown(int keyCode, KeyEvent msg){
        Log.v(TAG, "onKeyDown: keyCode =  "+ keyCode);
        Log.v(TAG, "onKeyDown: String =  " + msg.toString());

        switch (keyCode) {
            case KeyEvent.KEYCODE_DPAD_UP:
            case KeyEvent.KEYCODE_DPAD_RIGHT:
                mAlphavalue += 20;
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
            case KeyEvent.KEYCODE_DPAD_LEFT:
                mAlphavalue -= 20;
                break;
            default:
                break;
        }
        if(mAlphavalue>=0xFF)mAlphavalue = 0xFF;
        if(mAlphavalue<=0x0)mAlphavalue = 0x0;
        mImage.setAlpha(mAlphavalue);
        mAlphavalueText.setText("Alpha = " + mAlphavalue*100/0xff + "%");
        return super.onKeyDown(keyCode, msg);
    }
}
```

本例子使用 onKeyDown()函数来获得按键的事件，同类的函数还包括 onKeyUp()函数，其参数 int keyCode 为按键码，KeyEvent msg 表示按键事件的消息（其中包含了更详细的内容）。

上面打出的 log 信息为：

```
    VERBOSE/TestKeyEvent(771): onKeyDown: keyCode =  20
    VERBOSE/TestKeyEvent(771): onKeyDown: String =  KeyEvent{action=0 code=20
repeat=0 meta=0 scancode=108 mFlags=8}
```

基本上通过 keyCode 可以获得是哪一个按键响应，而通过 msg 除了按键码之外，可以获得按键的动作（抬起、按下）、重复信息，扫描码等内容。

KeyEvent 主要包含以下一些接口：

```
    final int  getAction()          // 获得按键的动作
    final int  getFlags()           // 获得标志
    final int  getKeyCode()         // 获得按键码
    final int  getRepeatCount()     // 获得重复的信息
    final int  getScanCode()        // 获得扫描码
```

通过 KeyEvent 接口，可以获得按键相关的详细信息。

6.3 运动事件的处理

触摸屏（TouchScreen）和滚动球（TrackBall）是 Android 中除了键盘之外的主要输入设备。如果需要使用触摸屏和滚动球，主要可以通过使用运动事件（MotionEvent）用于接收它们的信息。

触摸屏和滚动球事件主要通过实现以下 2 个函数来接收：

```
        public boolean onTouchEvent(MotionEvent event)
        public boolean onTrackballEvent(MotionEvent event)
```

在以上 2 个函数中，MotionEvent 类作为参数传入，在这个参数中可以获得运动事件的各种信息。

本例介绍另外触摸屏事件的程序，这个程序在 UI 的界面中，显示当前的 MotionEvent 的动作和位置。

图 触摸屏程序的运行结果

本例的程序的代码如下所示：

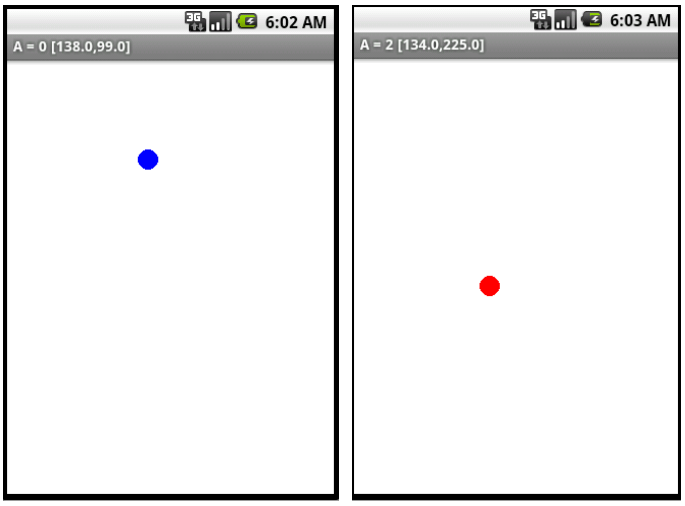
```
package com.android.basicapp;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;

import android.view.MotionEvent;
import android.widget.TextView;

public class TestMotionEvent extends Activity {
    private static final String TAG = "TestMotionEvent";
    TextView mAction;
    TextView mPostion;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.testmotionevent);
        mAction  = (TextView)findViewById(R.id.action);
        mPostion = (TextView)findViewById(R.id.postion);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int Action = event.getAction();
```

```
            float X = event.getX();
            float Y = event.getY();
            Log.v(TAG, "Action = "+ Action );
            Log.v(TAG, "("+X+","+Y+")");
            mAction.setText("Action = "+ Action);
            mPostion.setText("Postion = ("+X+","+Y+")");
            return true;
        }
    }
```

布局文件 testmotionevent.xml 的内容如下所示：

```
<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:id="@+id/action"
        android:textSize = "20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView android:id="@+id/postion"
        android:textSize = "20sp"
        android:layout_width="wrap_content"
        android:layout height="wrap content"/>
</LinearLayout>
```

另外一个示例程序，当触摸屏按下、移动、抬起的时候，在坐标处绘制不同颜色的点，在标题栏中显示当时的动作和坐标。程序的结果如图所示：



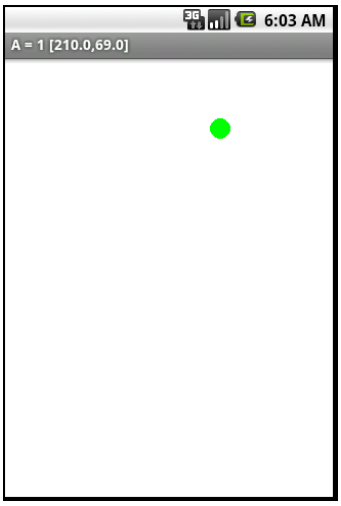


图 触摸屏程序的运行结果

这里使用的程序如下所示：

```
package com.android.basicapp;

import android.app.Activity;
import android.content.Context;
import android.graphics.*;
import android.os.Bundle;
import android.util.Log;

import android.view.MotionEvent;
import android.view.View;


public class TestMotionEvent2 extends Activity {
    private static final String TAG = "TestMotionEvent2";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new TestMotionView(this));
    }
    public class TestMotionView extends View {
        private Paint   mPaint = new Paint();
        private int    mAction;
        private float  mX;
        private float  mY;
        public TestMotionView(Context c) {
            super(c);
            mAction = MotionEvent.ACTION_UP;
            mX = 0;
```

```
            mY = 0;
        }
        @Override
        protected void onDraw(Canvas canvas) {
            Paint paint = mPaint;
            canvas.drawColor(Color.WHITE);
            if(MotionEvent.ACTION_MOVE == mAction) {        // 移动动作
                paint.setColor(Color.RED);
            }else if(MotionEvent.ACTION_UP == mAction) {    // 抬起动作
                paint.setColor(Color.GREEN);
            }else if(MotionEvent.ACTION DOWN == mAction) { // 按下动作
                paint.setColor(Color.BLUE);
            }
            canvas.drawCircle(mX, mY,10, paint);
            setTitle("A = " + mAction +  " ["+ mX +","+ mY +"]");
        }
        @Override
        public boolean onTouchEvent(MotionEvent event) {
            mAction = event.getAction();    // 获得动作
            mX = event.getX();              // 获得坐标
            mY = event.getY();
            Log.v(TAG, "Action = "+ mAction );
            Log.v(TAG, "("+mX+","+mY+")");
            invalidate();                   // 重新绘制
            return true;
        }
    }
}
```

在程序中，在触摸屏事件到来之后，接收到它，并且纪录发生事件的坐标和动作，然后调用 invalidate()重新进行绘制。绘制在 onDraw()中完成，根据不同的事件，绘制不同颜色的点，并设置标题栏。

MotionEvent 是用于处理运动事件的类，这个类中可以获得动作的类型、动作的坐标，在 Android 2.0 版本之后，MotionEvent 中还包含了多点触摸的信息，当有多个触点同时起作用的时候，可以获得触点的数目和每一个触点的坐标。

6.4 屏幕间的跳转和事件的传递

在一般情况下，Android 的每一个屏幕基本上就是一个活动（Activity），屏幕之间的切换实际上就是在活动间互相调用的过程，Android 使用 Intent 完成这个动作。

Android 屏幕跳转的关系和方式如下图所示：

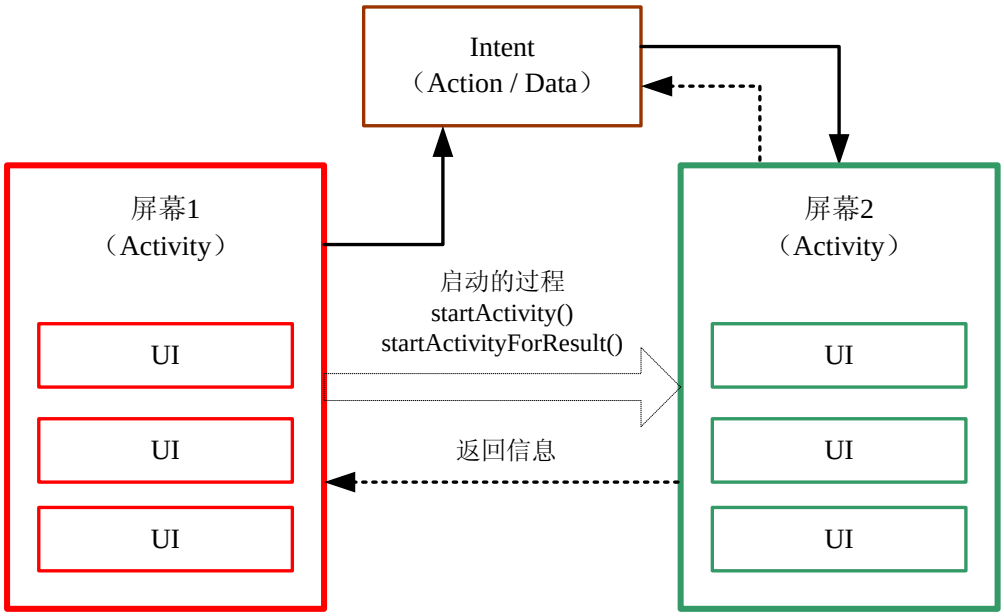


图　屏幕跳转的方式

事实上，在 Android 中，屏幕使用一个活动来实现，屏幕之间是相互独立的，屏幕之间的跳转关系通过 Intent 来实现。

6.4.1. 跳转的方法

本示例是一个简单的屏幕之间的跳转，从一个屏幕跳转到另一个屏幕，在启动第二个屏幕后，前一个屏幕消失。

参考示例程序：Forward（ApiDemo => App=>Activity=>Forward）

源代码：com/example/android/apis/app/Forward.java

com/example/android/apis/app/ForwardTarget.java

布局资源代码：forward_target.xml 和 forwarding.xml

本示例包含了两个活动，在 UI 上它们就是两个屏幕，分别为跳转的源和目的，因此在 AndroidManifest.xml 中分别定义。

```
<activity android:name=".app.Forwarding"
          android:label="@string/activity forward ding">
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.SAMPLE_CODE" />
```

```
        </intent-filter>
    </activity>
    <activity android:name=".app.ForwardTarget">  </activity>
```

两个活动的名称分别为 Forwarding 和 ForwardTarget，由于第二个活动没有 intent-filter，因此在程序中只能由第一个活动来启动。

Forward 程序的运行结果如图所示：

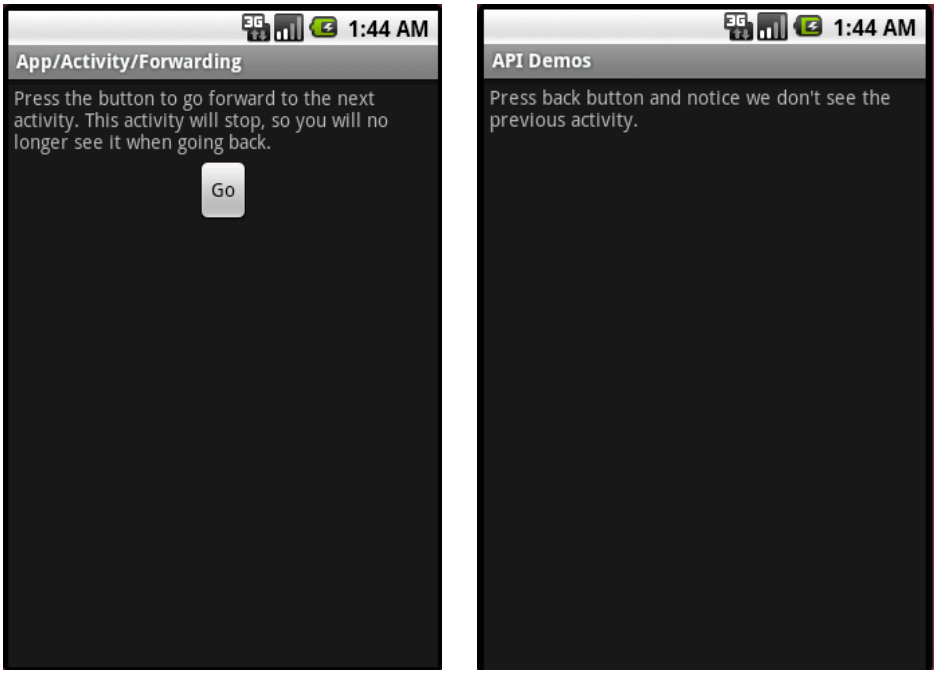


图 Forward 程序的运行结果

点击“Go”按钮从 Forward 跳转到 ForwardTarget，这个内容在 Java 源文件 Forward.java 的以下片段中处理：

```
public void onClick(View v)
{
    Intent intent = new Intent();                          // 建立 Intent
    intent.setClass(Forwarding.this, ForwardTarget.class);    // 设置活动
    startActivity(intent);
    finish();                                              // 结束当前活动
}
```

启动第二个活动需要使用 Intent，在其 setClass()函数中设置源和返回的内容，Intent 是 android.content 包中的类，用于启动活动、服务或者消息接收器。

这里使用的 Intent 的 setClass()的方法的原型如下所示：

```
Intent  setClass(Context packageContext, Class<?> cls)
```

第一个参数是当前的上下文类型 Context，因此把当前的活动设置过去即可（Activity 本身继承了 Context），第二个是 Intent 所包含的 JAVA 类，直接设置 ForwardTarget.class 类即可。

本例中使用了 finish()函数表示当前的活动结束，这样在第二个活动（ForwardTarget）启动时，第一个活动（Forward）已经不存在了。如果没有调用 finish()函数，第二个活动启动时，第一个活动就处于 OnPause 状态，当第二个活动退出后，第一个活动重新出现，也就是会调用活动的 onResume()函数。

6.4.2. 带有返回值的跳转

在某些时候，从跳转的对象返回时，跳转源头需要得到其返回的结果，这样两个屏幕才可实现一些交互。

参考示例程序：ReceiveResult（ApiDemo => App=>Activity=>ReceiveResult）

源代码：com/example/android/apis/app/ReceiveResult.java

com/example/android/apis/app/SendResult.java

布局资源代码：receive_result.xml 和 send_result.xml

```
<activity android:name=".app.ReceiveResult"
     android:label="@string/activity_receive_result">
  <intent-filter>
     <action android:name="android.intent.action.MAIN" />
     <category android:name="android.intent.category.SAMPLE_CODE" />
  </intent-filter>
</activity>
<activity android:name=".app.SendResult"> </activity>
```

ReceiveResult 程序的运行结果如图所示：

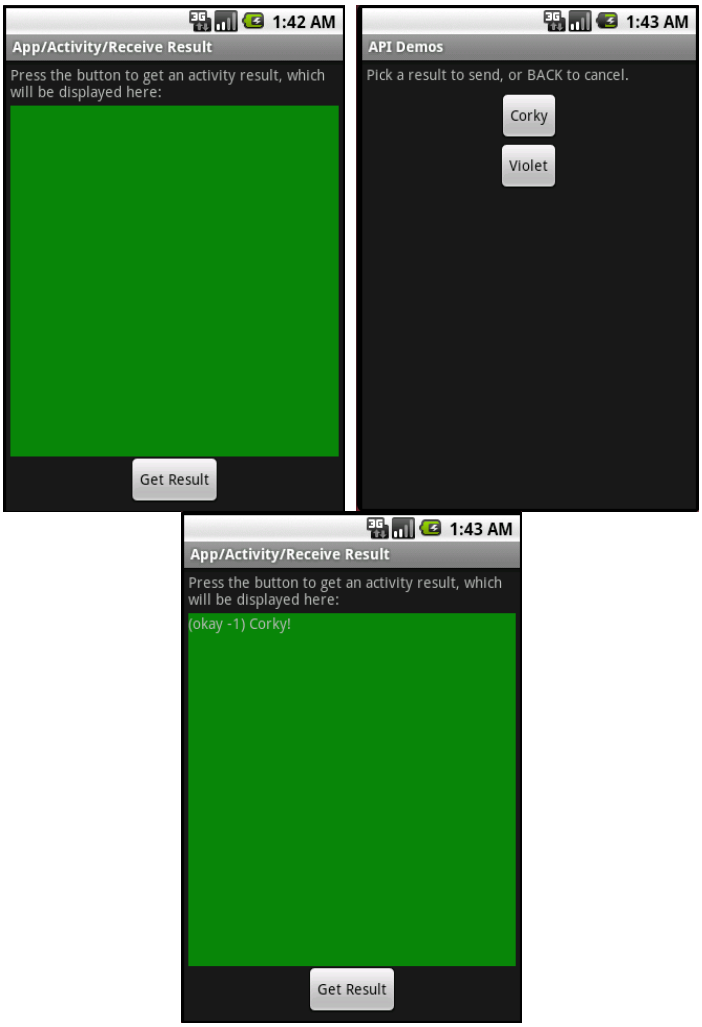


图 ReceiveResult 程序的运行结果

初始化界面如图所示，点击“Get Result”按钮将跳转到第二个屏幕，如中图所示；在第二个屏幕中点击“Corky”和“Violet”按钮将返回第一个屏幕，并获得对应显示，如右图所示。

Java 源文件 ReceiveResult.java 的代码片段如下所示：

```
    static final private int GET_CODE = 0;
        private OnClickListener mGetListener = new OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(ReceiveResult.this, SendResult.class);
```

```
            startActivityForResult (intent, GET_CODE);
        }
    };
```

这里调用的是 startActivityForResult()方法，设置一个 GET_CODE 为请求代码，这样可以获得目标活动的返回信息。这个函数的原型为：

```
public void startActivityForResult (Intent intent, int requestCode)
```

被跳转的目标的 Java 源文件 SendResult.java 的代码片段如下所示：

```
    private OnClickListener mCorkyListener = new OnClickListener()
    {
        public void onClick(View v)
        {
            setResult(RESULT_OK, (new Intent()).setAction("Corky!"));
            finish();
        }
    };
    private OnClickListener mVioletListener = new OnClickListener()
    {
        public void onClick(View v)
        {
            setResult(RESULT OK, (new Intent()).setAction("Violet!"));
            finish();
        }
    };
```

被跳转的目标程序将返回值返回，这里使用的依然是 Intent 作为交互的信息，通过 setAction()设置不同的活动。

由于被跳转的目标程序，是被显示 Intent 调用起来的。因此，返回后继续由 ReceiveResult.java 对返回值进行处理。返回的信息通过扩展 Activity 的 onActivityResult()函数来实现，两个整数类型的参数 requestCode 和 resultCode 分别代表请求代码和结果码，第三个参数 Intent（类型 data）表示活动间交互附加的数据信息。

```
    @Override
     protected void onActivityResult(int requestCode, int resultCode,
         Intent data) {
       if (requestCode == GET CODE) {
           Editable text = (Editable)mResults.getText();
           if (resultCode == RESULT_CANCELED) {
               text.append("(cancelled)");
           } else {
               text.append("(okay ");
               text.append(Integer.toString(resultCode));
```

```
                text.append(") ");
                if (data != null) {
                    text.append(data.getAction());
                }
            }
            text.append("\n");
        }
    }
```

这里 onActivityResult()是一个被继承的函数，其参数 data 就是这个活动作为返回值接受到，data.getAction()可以从返回的 Intent 中取回内容。

这里的参数 requestCode 也是根据当时的在调用 startActivityForResult()的时候指定的返回值。

Android 中使用 Intent 并使用 startActivity()和 startActivityForResult()调用一个新的活动，实现屏幕的跳转功能，调用者可以获得跳转对象的返回信息。

6.5 菜单的使用

菜单是屏幕中比较独立的一个元素，它和普通的控件略有不同，很多 GUI 系统都对菜单有单独的接口和运作方式。在 Android 中具有单独接口，用于在活动中使用菜单。

本例使用一个菜单来控制按钮的背景颜色，从其中可以了解如何在应用程序中使用菜单。

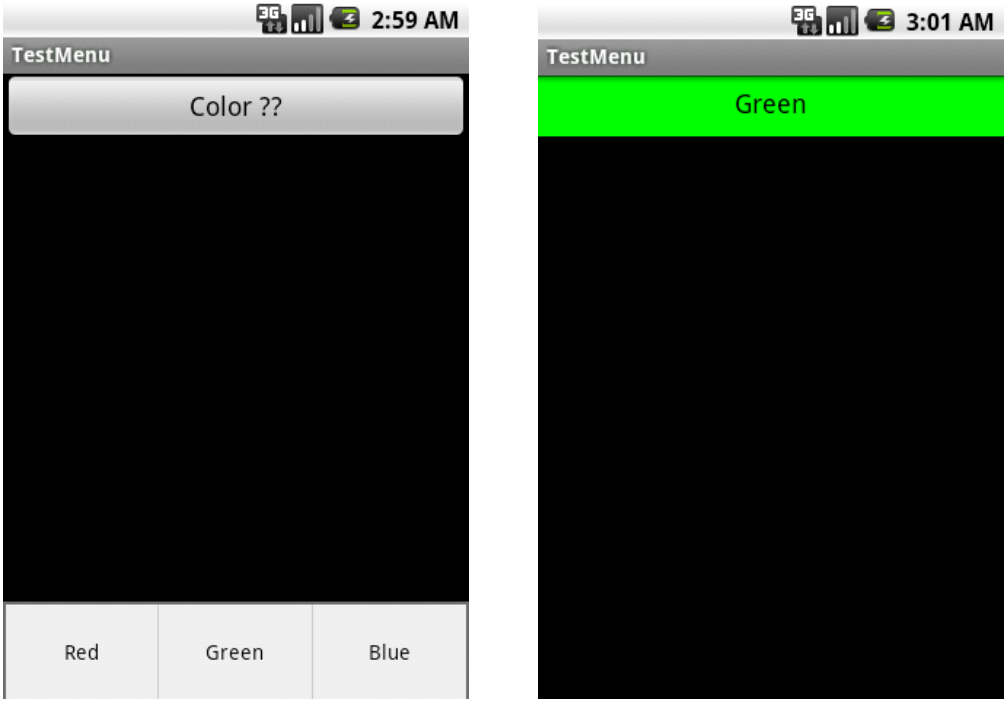


图　菜单示例程序的运行结果

建立菜单和调用的代码片段如下所示：

```
package com.android.basicapp;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

import android.graphics.Color;

import android.view.View;
import android.view.Menu;
import android.view.Gravity;
import android.view.MenuItem;

import android.widget.Button;
import android.widget.TextView;
import android.view.View.OnClickListener;

import android.util.Log;

public class TestMenu extends Activity {
```

```
    private static final String TAG = "TestMenu";
    private Button mButton;

    public static final int RED_MENU_ID   = Menu.FIRST;
    public static final int GREEN MENU ID = Menu.FIRST + 1;
    public static final int BLUE_MENU_ID  = Menu.FIRST + 2;
    public TestMenu() { }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.testmenu);
        mButton = (Button) findViewById(R.id.color button);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, RED MENU ID, 0, R.string.red);
        menu.add(0, GREEN_MENU_ID, 0, R.string.green);
        menu.add(0, BLUE_MENU_ID, 0, R.string.blue);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case RED_MENU_ID:
            mButton.setBackgroundColor(Color.RED);
            mButton.setText(R.string.red);
            return true;
        case GREEN MENU ID:
            mButton.setBackgroundColor(Color.GREEN);
            mButton.setText(R.string.green);
            return true;
        case BLUE_MENU_ID:
            mButton.setBackgroundColor(Color.BLUE);
            mButton.setText(R.string.blue);
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

使用菜单主要通过重载 Activity 中的两个函数来实现：

```
public boolean onCreateOptionsMenu(Menu menu)
public boolean onOptionsItemSelected(MenuItem item)
```

onCreateOptionsMenu()用于在建立菜单时进行设置，建立时为每一个按钮设置 ID，菜单项被选择时调用 onOptionsItemSelected()，通过 MenuItem 类的 getItemId() 函数获得这个菜单的 ID，继续进行处理。

菜单类在 Android 中表示为 android.view.Menu 类。使用这个类可以进行一些更为细节的设置和操作。

```
abstract MenuItem  add(int groupId, int itemId, int order, CharSequence title)
abstract MenuItem  add(int groupId, int itemId, int order, int titleRes)
```

add()的第 1、2 个参数是整数值，分别代表按钮项的组 ID 和选项 ID，第 3 个参数用于设置按钮上的文件。

6.6 弹出对话框

在 GUI 程序中，有时需要弹出对话框来提示一些信息。这些对话框比一个独立的屏幕简单，在 Android 中弹出式对话框不同于表示一个屏幕的活动，它通常用于简单的功能处理。

对话框的父类是 android.app.Dialog，通过构建类 android.app.AlertDialog 来实现弹出式对话框，可以使用 AlertDialog.Builder 和不同的参数来构建对话框。

参考示例程序：Dialog（ApiDemo => App=>Dialog）

源代码：com/example/android/apis/app/AlertDialogSamples.java

布局文件：alert_dialog.xml

Dialog 程序的运行结果如图所示：

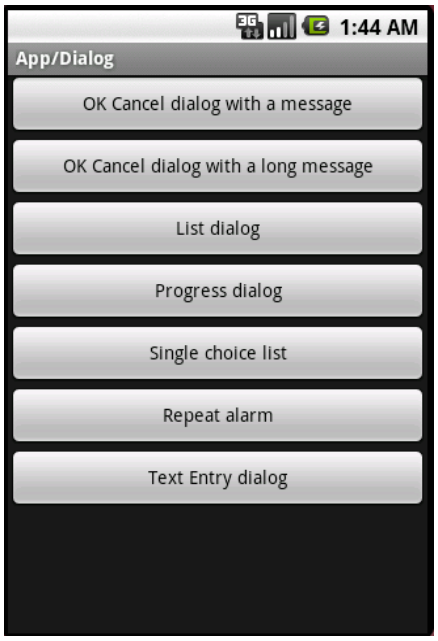


图 Dialog 程序的运行结果

通过点击屏幕上的不同按钮（第 4 个按钮除外）将会启动不同的对话框。

实现方法是继承 onCreateDialog()函数，返回一个 Dialog 类型：

```
@Override
protected Dialog onCreateDialog(int id) {
}
```

onCreateDialog()函数的参数 id 是区分对话框的标示，当调用对话框的时候需要调用 showDialog()。

```
public final void showDialog (int id)
```

showDialog()函数也是通过 id 来区分对话框。通过 showDialog()和 onCreateDialog()函数可以统一活动中的对话框。

6.6.1. 提示信息和两个按钮的对话框

第 1 个按钮（OK Cancel dialog with a message）启动一个提示信息和两个按钮的对话框，如图所示：

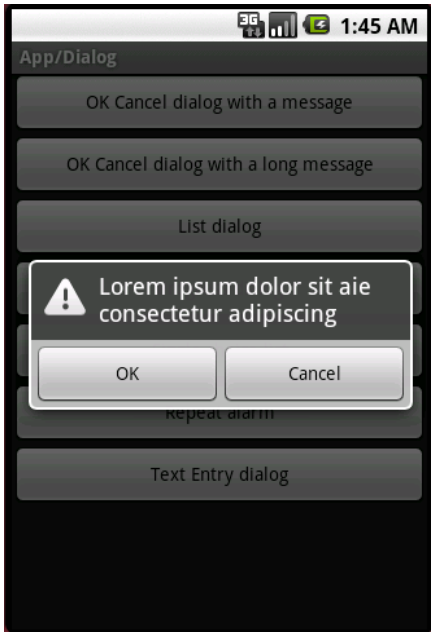


图 提示信息和两个按钮的对话框

代码实现的片断如下所示：

```
return new AlertDialog.Builder(AlertDialogSamples.this)
```

```
            .setIcon(R.drawable.alert_dialog_icon)
            .setTitle(R.string.alert dialog two buttons title)
            .setPositiveButton(R.string.alert_dialog_ok,                    new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    /* 左键事件 */
                }
            })
            .setNegativeButton(R.string.alert_dialog_cancel,                new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    /* 右键事件 */
                }
            })
```

其中，setPositiveButton 表示设置的左面的按钮，setNegativeButton 表示设置的右面的按钮，这两个按钮是确定的，但是可以设置其显示的字符和点击后的行为函数。

6.6.2. 提示信息和三个按钮的对话框

第 2 个按钮（OK Cancel dialog with a long message）启动一个提示信息和三个按钮的对话框，如图所示：

图　提示信息和三个按钮的对话框

代码实现的片断如下所示：

```
        return new AlertDialog.Builder(AlertDialogSamples.this)
            .setIcon(R.drawable.alert_dialog_icon)
            .setTitle(R.string.alert_dialog_two_buttons_msg)
            .setMessage(R.string.alert dialog two buttons2 msg)
            .setPositiveButton(R.string.alert_dialog_ok,                        new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    /* 左键事件 */
                }
            })
            .setNeutralButton(R.string.alert_dialog_something,                 new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    /* 中键事件 */
                }
            })
            .setNegativeButton(R.string.alert_dialog_cancel,                   new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    /* 右键事件 */
                }
            })
```

本对话框包含了 3 个按钮，与上一个例子的主要区别在于这里使用了setNeutralButton()表示的设置中间的按钮。

6.6.3. 列表项对话框

第 3 个按钮（List dialog）启动一个列表项对话框，如图所示；

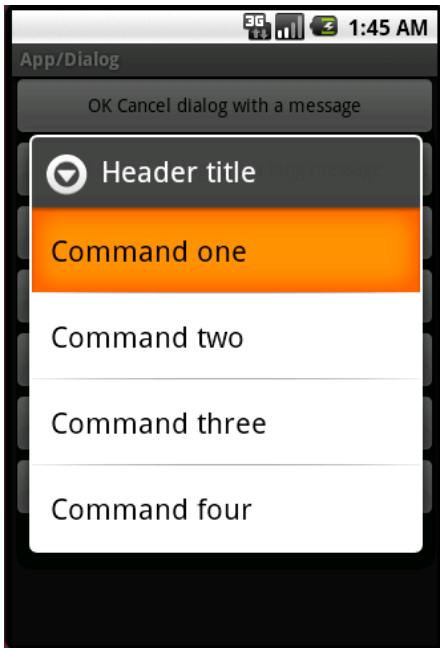


图 列表项对话框

代码实现的片断如下所示：

```
        return new AlertDialog.Builder(AlertDialogSamples.this)
            .setTitle(R.string.select_dialog)
            .setItems(R.array.select dialog items,                                    new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    String[] items =

getResources().getStringArray(R.array.select_dialog_items);
                    new AlertDialog.Builder(AlertDialogSamples.this)
                            .setMessage("You selected: " + which + " , " + items[which])
                            .show();
                }
            })
```

这里使用了 setItems()表示设置几个不同的项目，从 res/values/array.xml 文件中取得 select_dialog_items 的内容，这部分内容如下所示：

```
        <string-array name="select_dialog_items">
            <item>Command one</item>
            <item>Command two</item>
            <item>Command three</item>
            <item>Command four</item>
        </string-array>
```

这里的 Item 也设置了点击函数，因此它们被点击后，也会弹出新的对话框。

6.6.4. 单选项和按钮对话框

第 5 个按钮（Single choice list）启动一个单选项和按钮对话框；

图　单选项和按钮对话框

代码实现的片断如下所示：

```
       return new AlertDialog.Builder(AlertDialogSamples.this)
           .setIcon(R.drawable.alert dialog icon)
           .setTitle(R.string.alert_dialog_single_choice)
           .setSingleChoiceItems(R.array.select_dialog_items2,      0,      new
DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int whichButton) {
               }
           })
           .setPositiveButton(R.string.alert_dialog_ok,                    new
DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int whichButton) {
                   /* 左键事件 */
               }
           })
           .setNegativeButton(R.string.alert dialog cancel,                new
DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int whichButton) {
```

```
                /* 右键事件 */
            }
        })
```

本例是一个包含单选项的对话框，其中的选项使用了更简单的模式，从 res/values/array.xml 文件中取得 select_dialog_items2 中的内容作为单选项的项目。

这部分的内容如下所示：

```
    <string-array name="select_dialog_items2">
        <item>Map</item>
        <item>Satellite</item>
        <item>Traffic</item>
        <item>Street view</item>
    </string-array>
```

6.6.5. 复选项和按钮对话框

第 6 个按钮（Repeat alarm）启动一个复选项和按钮对话框；

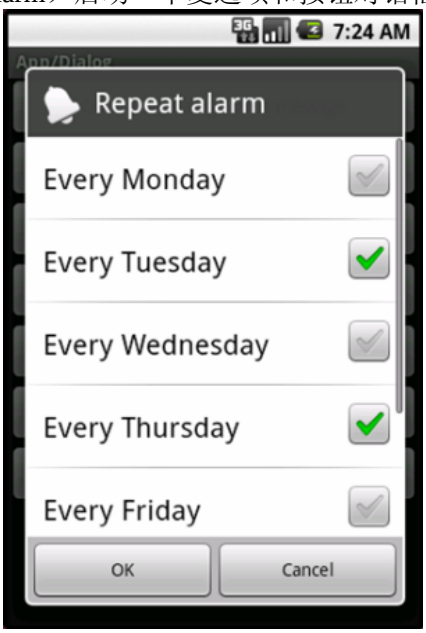


图　复选项和按钮对话框

代码实现的片断如下所示：

```
    return new AlertDialog.Builder(AlertDialogSamples.this)
        .setIcon(R.drawable.ic popup reminder)
        .setTitle(R.string.alert dialog multi choice)
        .setMultiChoiceItems(R.array.select_dialog_items3,
```

```
                new boolean[]{false, true, false, true, false, false, false},
                new DialogInterface.OnMultiChoiceClickListener() {
                   public    void    onClick(DialogInterface    dialog,    int
whichButton,
                           boolean isChecked) {
                       /* 点击复选框的响应 */
                   }
                })
           .setPositiveButton(R.string.alert_dialog_ok,                    new
DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int whichButton) {
                  /* 左键事件 */
               }
           })
           .setNegativeButton(R.string.alert_dialog_cancel,                new
DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int whichButton) {
                  /* 右键事件 */
               }
           })
          .create();
```

本例是一个包含复选项的对话框，从 res/values/array.xml 文件中取得 select_dialog_items3 中的内容作为单选项的项目：

```
    <string-array name="select_dialog_items3">
        <item>Every Monday</item>
        <item>Every Tuesday</item>
        <item>Every Wednesday</item>
        <item>Every Thursday</item>
        <item>Every Friday</item>
        <item>Every Saturday</item>
        <item>Every Sunday</item>
    </string-array>
```

6.6.6. 文本的按键对话框（使用布局文件）

第 7 个按钮（Text Entry dialog）启动一个包含文本的按键对话框。

Dialog 程序中调用各个对话框的效果如图所示：

图　文本的按键对话框

代码实现的片断如下所示：

```
      LayoutInflater factory = LayoutInflater.from(this);
      final                View                textEntryView                =
factory.inflate(R.layout.alert_dialog_text_entry,
                                                  null);
      return new AlertDialog.Builder(AlertDialogSamples.this)
          .setIcon(R.drawable.alert_dialog_icon)
          .setTitle(R.string.alert dialog text entry)
          .setView(textEntryView)
          .setPositiveButton(R.string.alert_dialog_ok,                    new
DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int whichButton) {
                  /* 左键事件 */
              }
          })
          .setNegativeButton(R.string.alert_dialog_cancel,                new
DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int whichButton) {
                  /* 右键事件 */
              }
          })
          .create();
    }
```

alert_dialog_text_entry.xml 也是一个布局文件，其中包含了 2 个文本框和 2 个可

编辑文本，这就是显示在屏幕上的内容，由此根据这种模式，也可以在弹出的对话框中使用布局文件。

由此，在这个对话框中，包含了这些相应的控件。

如上面对话框的效果所示，对话框可以设置标题、图标、提示信息、最多 3 个按钮、单选项、复选项，甚至可以设置一个 View。最后一个对话框是通过设置一个 View 来实现的，设置的内容在布局文件 alert_dialog_text_entry.xml 中。

对话框的类为 android.app.Dialog，通过 android.app.AlertDialog.Builder 类来建立，在建立的过程中可以进行多项设置。

- setIcon()和 setTitle()：用于设置图标和标题；
- setMessage()：用于设置提示信息；
- setPositiveButton()、setNeutralButton()和 setNegativeButton()：用于设置左、中、右按钮；
- setSingleChoiceItems()和 setMultiChoiceItems()：用于设置单选项和复选项；
- setView()：用于设置一个 View 作为对话框的内容。

以上函数的返回类型均为 android.app.AlertDialog.Builder，也就是这个类本身，因此可以使用如下的方式进行连续调用来设置更多的内容。

设置完成后调用 create()函数返回 android.app.AlertDialog 类，这个类表示一个可以使用的对话框。

在 Android 中使用对话框，可以在没有 Activity 的情况下建立一个比较简易的窗体，基本界面可以通过直接设置得到，通过 setView()可以获得任意内容的界面。

6.7 样式的设置

在 Android 中，应用程序所呈现的样子不完全由布局文件和源代码决定。通过在 AndroidManifest.xml 中设置样式，也可以控制活动的外观，所设置的样式可以基于预定的样式，也可以自定义样式。

6.7.1. 预定样式对话框

在 Android 中，定义了一些具体的样式，它们可以在应用程序中被使用。本示例介绍如何使用 Android 中的预定义样式。

参考示例程序：DialogActivity（ApiDemo=>App=>Activity=>Dialog）

源代码：com/example/android/apis/app/DialogActivity.java
布局文件：custom_dialog_activity.xml

AndroidManifest.xml 中的定义如下所示：

```
<activity android:name=".app.DialogActivity"
       android:label="@string/activity dialog"
                    android:theme="@android:style/Theme. Dialog" >
       <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.SAMPLE_CODE" />
       </intent-filter>
</activity>
```

DialogActivity 程序的运行结果如图所示：

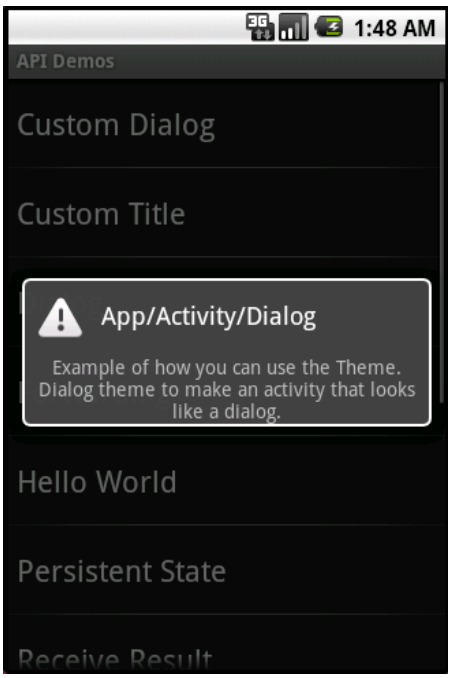


图 DialogActivity 程序的运行结果

这个程序本质上是一个活动，但是显示的结果类似于一个小对话框，而且背景是透明的。这个程序的布局文件和源代码都并无特别的地方，效果是通过在 AndroidManifest.xml 中设置其样式（android:theme）为 Theme.Dialog 来实现的，Theme.Dialog 是 Android 中的预定义样式。

6.7.2. 自定义样式对话框

除了使用 Android 系统中已有的样式，还可是使用自定义的样式。本示例介绍如何使用自定义样式。

参考示例程序：CustomDialogActivity（ApiDemo=>App=>Activity=>CustomDialog）

源代码：com/example/android/apis/app/CustomDialogActivity.java

布局文件：dialog_activity.xml

样式文件：values/styles.xml

AndroidManifest.xml 中的定义如下所示：

```
<activity android:name=".app.CustomDialogActivity"
        android:label="@string/activity custom dialog"
        android:theme="@style/Theme.CustomDialog " >
    <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.SAMPLE_CODE" />
   </intent-filter>
</activity>
```

CustomDialogActivity 程序的运行结果如图所示：

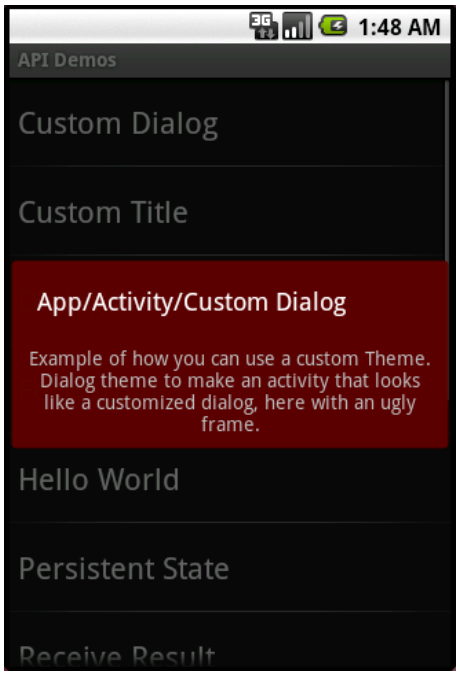


图 CustomDialogActivity 程序的运行结果

这个程序和上一个程序基本相同，区别在于样式被设置成了 CustomDialog，CustomDialog 是一个自定义样式，在 styles.xml 中进行定义，如下所示：

```
<style name="Theme.CustomDialog" parent="android:style/Theme.Dialog">
      <item name="android:windowBackground">@drawable/filled box</item>
</style>
```

CustomDialog 本身是“扩展”了预定的 Dialog 样式，重新定义了窗口的背景为 drawable 中的 filled_box，这里引用了 filled_box.xml 文件，这个文件在 res/drawable 中，其中定义了相关内容。

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
   <solid android:color="#f0600000"/>
   <stroke android:width="3dp" color="#ffff8080"/>
   <corners android:radius="3dp" />
   <padding android:left="10dp" android:top="10dp"
          android:right="10dp" android:bottom="10dp" />
</shape>
```

在定义的样式中，通过设置更多的值来获得不同的窗口效果。通过定义样式文件可以获得复用效果。

6.7.3. 窗口透明样式示例

在 Android 程序中，当某一个活动启动之后可能需要使用背景透明的效果，本例用于描述背景透明的应用。

参考示例程序：TranslucentActivity（ApiDemo=>App=>Activity=>Translucent）

TranslucentBlurActivity（App=>Activity=>TranslucentBlur）

源代码：com/example/android/apis/app/TranslucentActivity.java

com/example/android/apis/app/TranslucentBlurActivity.java

样式文件：values/styles.xml

AndroidManifest.xml 中的定义如下所示：

```
<activity android:name=".app.TranslucentActivity"
      android:label="@string/activity translucent"
      android:theme="@style/Theme.Translucent “ >
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.SAMPLE CODE" />
   </intent-filter>
</activity>
```

```
<activity android:name=".app.TranslucentBlurActivity"
        android:label="@string/activity_translucent_blur"
        android:theme="@style/Theme.Transparent " >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.SAMPLE CODE" />
    </intent-filter>
</activity>
```

TranslucentActivity 和 TranslucentBlurActivity 程序的运行结果如图所示：

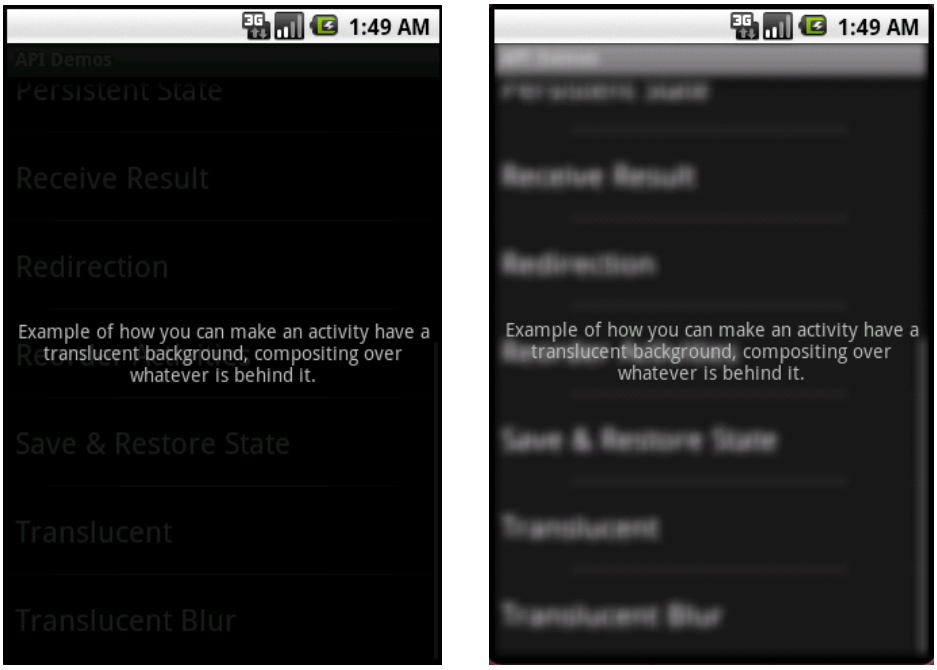


图 TranslucentActivity 和 TranslucentBlurActivity 程序的运行结果

这两个程序使用的都是窗口透明的效果，TranslucentActivity 获得的效果是背景普通的透明，TranslucentBlurActivity 获得的效果是背景模糊的透明。它们的样式被设置成了 Translucent，这是一个用于描述背景透明的自定义样式，在 styles.xml 中定义。

```
<style name="Theme.Translucent" parent="android:style/Theme.Translucent">
    <item name="android:windowBackground">
        @drawable/translucent_background
    </item>
    <item name="android:windowNoTitle">true</item>
    <item name="android:colorForeground">#fff</item>
</style>
```

translucent_background 值用于设置窗口的背景为透明，同时设置了 windowNoTitle 表示窗口不包含标题栏。

TranslucentBlurActivity 之所以能够获得背景模糊的效果，是因为在源代码中进行了进一步的设置，如下所示：

```
public class TranslucentBlurActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
```

```
            getWindow().
                setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
                        WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
            setContentView(R.layout.translucent_background);
        }
    }
```

设置模糊效果是通过窗口管理器（WindowManager）设置参数来完成的，这种设置只有在背景设置为透明后才能显示效果。

第 7 章 控件（Widget）的使用

在各个 GUI 系统中，控件一般都是占内容最多的部分，使用各种控件也是使用一个 GUI 系统的主要内容。

7.1 Android 中控件的层次结构

android.view.View 类（视图类）呈现了最基本的 UI 构造块。一个视图占据屏幕上的一个方形区域，并且负责绘制和事件处理。View 是 widgets 的基类，常用来创建交互式的图形用户界面（GUI）。

视图类有众多的扩展者，包括文本视图（TextView）、图像视图（ImageView）、进度条（ProgressBar）、视图组（ViewGroup）等。

Android 中控件类的扩展结构如图所示：

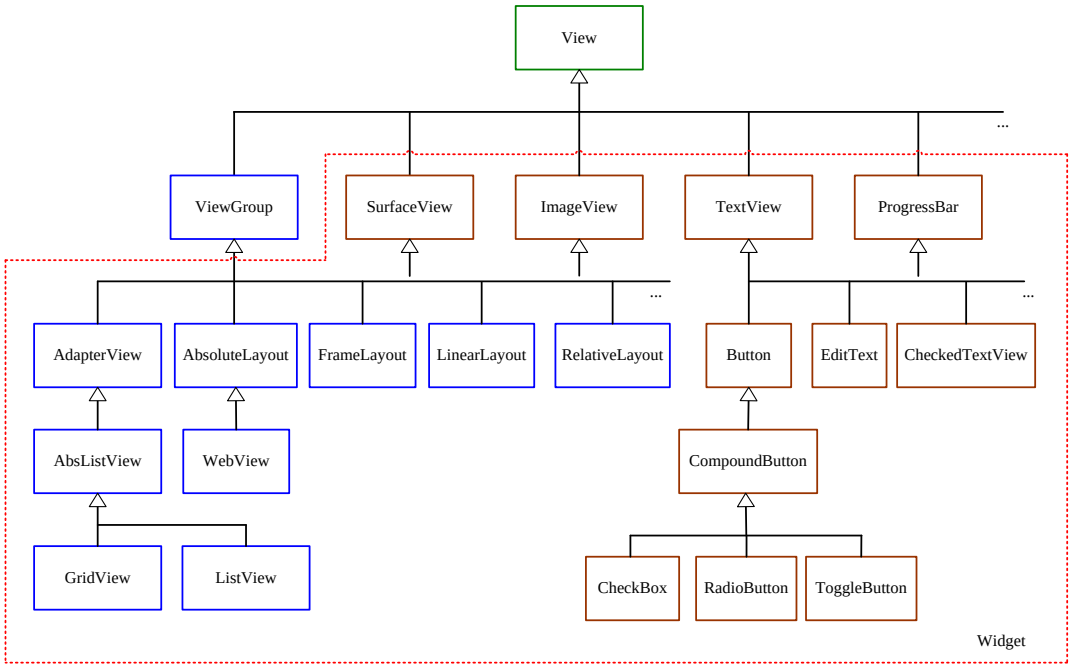


图 Android 中控件类的扩展结构

Android 中的控件常常在布局文件（Layout）中进行描述，在 Java 源代码中通过 findViewById()函数根据 ID 获得每一个 View 的句柄，并且转换成实际的类

型来使用。android.view.View 的扩展者也称作 Widget，通常包含在 android.widget 包中，也就是在 UI 中使用的控件。这些 android.view.View 的扩展者，通常可以在应用程序中直接使用，也可以应用程序再扩展一次使用。

在 Android 中各种 UI 类的名称也是它们在布局文件 XML 中使用的标签名称。

android.view.View 的一个重要的扩展者是 android.view.ViewGroup 类，这个类表示一个视图的集合，在这个视图的集合中可以包含众多的子视图。android.view.ViewGroup 类的扩展者既是多个视图的组合，本身也是一个视图。

7.2 基本控件的使用

Android 中的基本视图是 GUI 中通常直接使用的一些类，例如：字符区域、按钮、图像区域、图像按钮、进度条等。

7.2.1. 普通按钮

这里介绍普通按钮的使用，最普通的按钮是各种 GUI 系统中都类似的按钮，另外一种 ToggleButton 是具有开关两种状态的按钮。

参考示例程序：Buttons1（ApiDemo=>Views=>Buttons1）

源代码：com/example/android/apis/view/Buttons1.java

布局文件：buttons_1.xml

Buttons1 程序的运行结果如图所示：

图 Buttons1 程序的运行结果

界面比较简单，前两个按钮是 Button 类，表示普通的按钮；第三个按钮是 ToggleButton 类，表示可以进行开关操作的按钮。

这个活动的源代码很简单，实际上只有布局文件有特殊点。buttons_1.xml 的内容如下所示：

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <!-- Regular sized buttons -->
    <Button android:id="@+id/button_normal"
        android:text="@string/buttons 1 normal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <!-- Small buttons -->
    <Button android:id="@+id/button_small"
        style="?android:attr/buttonStyleSmall"
        android:text="@string/buttons 1 small"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <ToggleButton android:id="@+id/button toggle"
        android:text="@string/buttons_1_toggle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

```
    </LinearLayout>
```

这里主要引用了 3 个控件，2 个 Button 和 1 个 ToggleButton，它们都是 Android 中预定义的控件。

Button 类的扩展关系如下所示：

=> android.view.View

=> android.widget.TextView

=> android.widget.Button

Button 类扩展了 TextView 类，TextView 类是 View 的直接扩展者，表示一个文本区域，Android 中以文本为主要内容的各种控件均扩展自这个类。除了按钮之外，TextView 类的另外一个重要的扩展者是可编辑文本区域（EditText）。

按钮类（Button）作为 TextView 类的扩展者，主要的区别表现在外观和使用的方式上，Button 通常要设置处理点击动作的处理器（View.OnClickListener）；TextView 类虽然也可以设置这个内容，但是通常不需要这样做。

在本例的布局文件中，使用了 android:text 一个属性来定义在 Button 上面显示的文本，根据帮助，这其实是 TextView 中的一个 XML 属性，在这里被 Button 类继承使用，除了在布局文件中指定，还可以使用 setText(CharSequence)在 JAVA 源代码中进行设置。

ToggleButton 类的扩展关系如下所示：

=> android.view.View

=> android.widget.TextView

=> android.widget.Button

=> android.widget.CompoundButton

=> android.widget.ToggleButton

Button 类具有一个名为 CompoundButton（组合按钮）的扩展者，CompoundButton 又有了圆形按钮（RadioButton）、选择框（CheckBox）和开关按钮（ToggleButton）3 个扩展者。ToggleButton 比较简单，包含开关两个状态，可以显示不同的文本 textOn（开）和 textOff（关），在使用 ToggleButton 时主要根据 CompoundButton 的 isChecked()函数获得其是否选择的状态。

根据 ToggleButton 的帮助可以得知，其特定的 XML 属性包括了以下的内容：

android:disabledAlpha：禁止的时候的 Alpha 值，使用浮点数

android:textOff：定义开状态下显示的文本

android:textOn：定义开状态下显示的文本

Android 中的控件在使用上涉及的内容包括了：

在 JAVA 源代码中使用的方法

在布局文件中使用 XML 属性

每个控件本身涉及的内容包括它直接或者间接扩展的类，以及它自己的独特功能。例如，根据上述的继承关系，TextView 中能使用的所有内容，都可以在 Button 中使用，在 Button 中能使用的内容，都可以在 ToggleButton 中使用。

7.2.2. 图像区域

在 UI 界面上显示图片，是一个常常需要使用到的功能。在 Android 中可以使用图像区域是一个可以直接显示图片文件的控件，可以方便显示一个图片。

参考示例程序：ImageView（ApiDemo=>Views=>ImageView）

源代码：com/example/android/apis/view/ImageView1.java

布局文件：image_view_1.xml

ImageView 程序的运行结果如图所示：

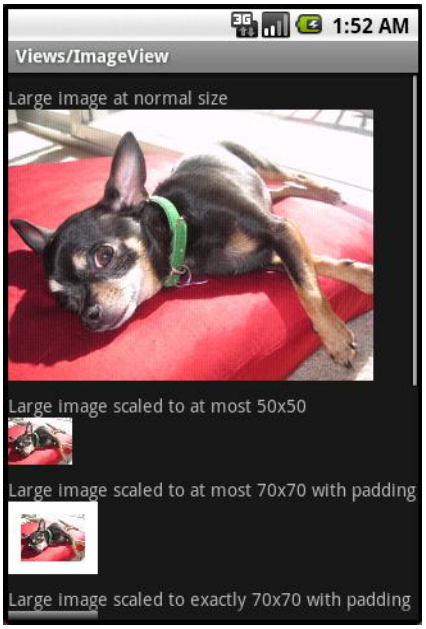


图 ImageView 程序的运行结果

程序中的图像都是通过 ImageView 类来实现显示的，ImageView 是 View 的直接扩展者，继承关系如下所示：

=> android.view.View

=> android.widget.ImageView

这里所使用的布局文件的一个片断如下所示：

```
<ImageView
    android:src="@drawable/sample_1"
    android:adjustViewBounds="true"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<ImageView
    android:src="@drawable/sample_1"
    android:background="#66FFFFFF"
    android:adjustViewBounds="true"
    android:maxWidth="70dip"
    android:maxHeight="70dip"
    android:padding="10dip"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

根据布局文件，可以得知，这里主要用来显示图片的内容是一个 ImageView 标签。它具有一个 android:src 属性，这个属性实际上就是用来设置所显示的图片的。

ImageView 又被称为图像视图，是 Android 中可以直接显示图形的控件，其中图像源是其核心。ImageView 有多种不同的设置图像源的方法：

```
void setImageResource (int resId)   // 设置图像源的资源 ID
void setImageURI(Uri uri)           // 设置图像源的 URI
void setImageBitmap(Bitmap bm)      // 设置一个 Bitmap 位图为图像源
```

使用 ID 的方式表示设置包中预置的图像资源，使用 URI 可以设置文件系统中存储在各种地方的图像等，使用 Bitmap 的方式可以设置一个已经表示为 Bitmap 格式的图像。

ImageView 还支持缩放、剪裁等功能，具有相关的方法进行设置。示例中的第二个图像通过指定最大的宽（android:maxWidth）和高（android:maxHeight）来实现缩小，第三个图像通过指定 android:padding 属性来实现为图像留出一个边缘。

7.2.3. 图像按钮

图像按钮是一个带有图片的按钮，从逻辑上可以实现普通按钮功能。图像按钮实际上是结合图像和按钮的双重特性。

参考示例程序：ImageButton（ApiDemo=>Views=>ImageButton）

源代码：com/example/android/apis/view/ImageButton.java

布局文件：image_button_1.xml

ImageButton 程序的运行结果如图所示：

图 ImageButton 程序的运行结果

这里的布局文件的主要内容如下所示：

```
<ImageButton
 android:layout width="100dip"
 android:layout_height="50dip"
 android:src="@android:drawable/sym_action_call" />

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

```
        android:src="@android:drawable/sym_action_chat" />

        <ImageButton
        android:layout_width="wrap_content"
        android:layout height="wrap content"
        android:src="@android:drawable/sym_action_email" />
```

示例中使用了 ImageButton 类作为显示一个带有图像的按钮，扩展关系如下所示：

=> aandroid.view.View

=> aandroid.widget.ImageView

=> aandroid.widget.ImageButton

图像按钮 ImageButton 扩展了 ImageView，它结合了图像和按钮的功能。ImageButton 除了可以当作按钮来使用，其他方面和 ImageView 基本一致。ImageButton 和 ImageView 的区别也仅在于外观和使用方式上，主要的图像设置方法和 ImageButton 中的一样。ImageButton 特定的是具有一个 onSetAlpha()函数：

```
    boolean  onSetAlpha(int alpha)
```

onSetAlpha()函数通过指定 0-255 来指定 Alpha 数值。

事实上，ImageButton 除了在外观上表现成一个按钮的状态，其他方面和 ImageView 基本一样。由于是按钮的功能，在 JAVA 源程序中，ImageButton 通常被设定 OnClickListener 来获得点击时候的响应函数。

由于 JAVA 语言不支持多重继承，因此，在 Android 中图像按钮 ImageButton 只是扩展了 ImageView，和普通按钮 Button 并没有继承（扩展）关系。

ImageButton 有一个扩展者是 ZoomButton，这是一个带有动态缩放功能的图像按钮。

7.2.4. 进度条

进度条可以用图形的方式显示一个百分比的效果。在 Android 中具有预定义的进度条可以使用。

参考示例程序：ProgressBar1（ApiDemo=>Views=>ProgressBar）

源代码：com/example/android/apis/view/ProgressBar1.java

布局文件：progressbar_1.xml

ProgressBar1 程序的运行结果如图所示：

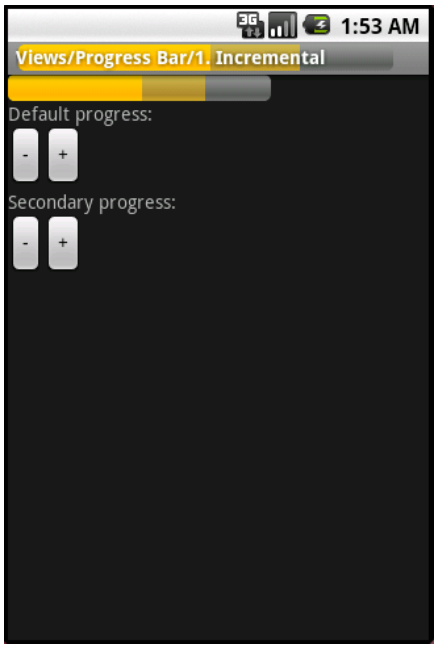


图 ProgressBar1 程序的运行结果

这里的布局文件 progressbar_1.xml 的主要内容如下所示：

```
<ProgressBar android:id="@+id/progress_horizontal"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout width="200dip"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="50"
    android:secondaryProgress="75" />
```

标签 ProgressBar 是表示了进度条的控件，扩展关系如下：

=> android.view.View

=> android.widget.ProgressBar

ProgressBar 是 android.view.View 的直接扩展者，在 GUI 界面中实现进度条的功能。ProgressBar 比较特殊的地方是这个类还支持第二个进度条，如示例所示，第二个进度条在第一个进度条的背后显示，两个进度条的最大值是相同的。

ProgressBar 的主要参数是进度条的当前值和最大值。

```
int  getMax()                                          // 获得进度条的最大值
void  setProgress(int progress)                        // 设置主进度条的进度
void  setSecondaryProgress(int secondaryProgress)      // 设置第二个进度条的进度
synchronized int  getProgress ()                        // 获得进度值
```

```
    synchronized int  getSecondaryProgress ()                // 获得第二个进度条的进度
```

ProgressBar 在使用的时候，要注意最大值和当前值的关系，在 UI 上所呈现的状态，其实是当前值和最大值的一个比例。

在本示例程序中，可以通过按钮来控制进度条，这部分内容是在 JAVA 源代码中实现的：

```
    protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           requestWindowFeature(Window.FEATURE_PROGRESS);
           setContentView(R.layout.progressbar_1);
           setProgressBarVisibility(true);

           final ProgressBar progressHorizontal
                        = (ProgressBar) findViewById(R.id.progress_horizontal);
           setProgress(progressHorizontal.getProgress() * 100);
           setSecondaryProgress(progressHorizontal.getSecondaryProgress()     *
100);
    }
```

由于这里使用了 requestWindowFeature(Window.FEATURE_PROGRESS)来获得了将进度条设置到标题栏的当中。因此这里调用了几个 Activity 中的函数，用于设置在标题栏中的进度条。

```
    final void  setProgress(int progress)
    final void  setSecondaryProgress(int secondaryProgress)
    final void  setProgressBarVisibility(boolean visible)
```

其中一个按钮的 onClick()调用如下所示：

```
              public void onClick(View v) {
                 progressHorizontal.incrementProgressBy(-1);
                 // Title progress is in range 0..10000
                 setProgress(100 * progressHorizontal.getProgress());
              }
```

事实上，这里调用的 progressHorizontal 是 ProgressBar 类的一个实例，而标题栏的进度条，是一个单独的内容。

在 Android 中还有一些其他类型的进度条。

参考示例程序：RatingBar1（Views=>RatingBar1）

源代码：com/example/android/apis/view/RatingBar1.java

布局文件：ratingbar_1.xml

RatingBar1 程序的运行结果如图所示：

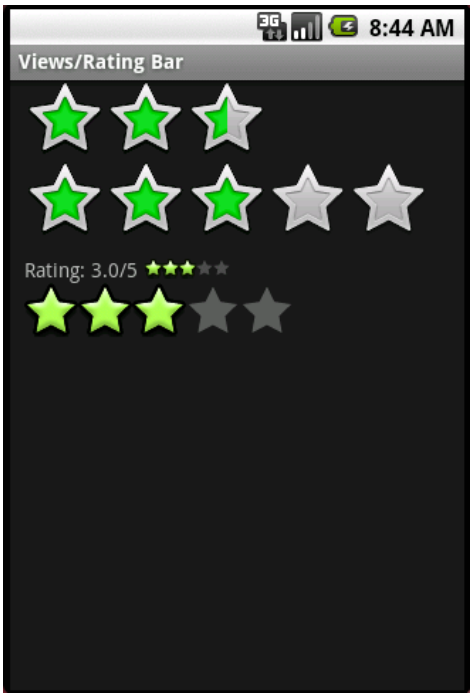


图 ProgressBar1 程序的运行结果

这里的布局文件 ratingbar _1.xml 的主要内容如下所示：

```
<RatingBar android:id="@+id/ratingbar1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="3"
    android:rating="2.5" />
<RatingBar android:id="@+id/ratingbar2"
    android:layout width="wrap content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:rating="2.25" />
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout marginTop="10dip">
    <TextView android:id="@+id/rating"
        android:layout_width="wrap_content"
        android:layout height="wrap content" />
    <RatingBar android:id="@+id/small_ratingbar"
        style="?android:attr/ratingBarStyleSmall"
        android:layout marginLeft="5dip"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
            android:layout_gravity="center_vertical" />
    </LinearLayout>
    <RatingBar android:id="@+id/indicator_ratingbar"
        style="?android:attr/ratingBarStyleIndicator"
        android:layout marginLeft="5dip"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical" />
```

这里，一共定义了 4 个 RatingBar 标签，RatingBar 的继承关系如下所示：

=> android.view.View

=> android.widget.ProgressBar

=> android.widget.AbsSeekBar

=> android.widget.RatingBar

AbsSeekBar 是 ProgressBar 的扩展者，这是一个表示绝对进度的类，由于使用的是绝对进度，因此主要区别是 AbsSeekBar 的进度最大值是可以设置的（对应 setMax()函数）。RatingBar 和 SeekBar 两个类又扩展了 AbsSeekBar，其中 RatingBar 可以直接用星星的方式来表示进度；SeekBar 可以使用可拖拽的小图标。

RatingBar 是 AbsSeekBar 的一个继承者，AbsSeekBar 和 ProgressBar 的一个主要扩展就是其最大值可以设置。在本例的布局文件中，android:numStars 和 android:rating 等几个属性是 RatingBar 自己的属性。

7.2.5. 多种控件

这里介绍一个具有多种控件的示例，它们被包含在一个活动中。

参考示例程序：Controls1（ApiDemo=>Views=>Controls1）

源代码：com/example/android/apis/view/Controls1.java

布局文件：controls_1.xml

Controls1 程序的运行结果如图所示：

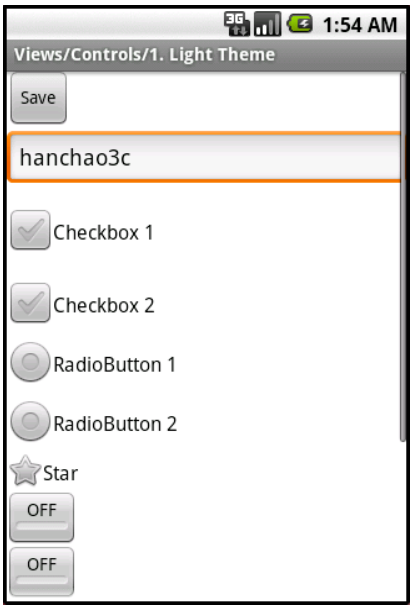


图　Controls1 程序的运行结果

在界面中包含了：Button（普通按钮）、EditText（可编辑文本区域）、CheckBox（复选框）、RadioGroup（单选按钮组）、ToggleButton（开关按钮）、TextView（文本区域）、Spinner（旋转按钮）等控件，这些内容均在布局文件中定义。

在 Android 中使用各种控件基本的原则是在布局文件中可以实现 UI 的外观，然后在 JAVA 文件中实现对各种的控件的控制动作。

7.3 自定义的视图

自定义的 View 的含义是通过扩展的方法，实现一个扩展 android.view.View 类的类，这个类的本质也是一个控件，通过它可以直接构建 UI。

参考示例程序：CustomView（ApiDemo=>Views=>CustomView ）

源代码：com/example/android/apis/view/CustomView1.java

com/example/android/apis/view/LabelView.java

布局文件：custom_view_1.xml

CustomView 程序的运行结果如图所示：

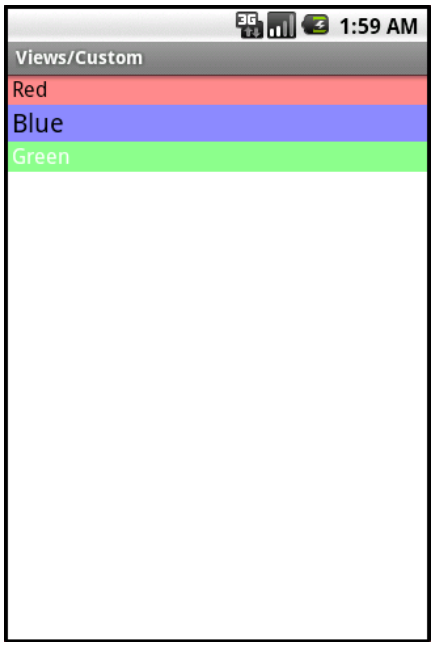


图 CustomView 程序的运行结果

布局文件 custom_view_1.xml 的如下所示：

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

xmlns:app="http://schemas.android.com/apk/res/com.example.android.apis"
          android:orientation="vertical"
          android:layout width="fill parent"
          android:layout_height="wrap_content">
       <com.example.android.apis.view.LabelView
             android:background="@drawable/red"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             app:text="Red"/>
       <com.example.android.apis.view.LabelView
             android:background="@drawable/blue"
             android:layout width="fill parent"
             android:layout_height="wrap_content"
             app:text="Blue" app:textSize="20dp"/>
       <com.example.android.apis.view.LabelView
             android:background="@drawable/green"
             android:layout_width="fill_parent"
             android:layout height="wrap content"
             app:text="Green" app:textColor="#ffffffff" />
    </LinearLayout>
```

这里使用的标签 com.example.android.apis.view.LabelView 不是 Android 框架层提供的 View 的一个子类，而是在自己的程序中实现的一个类。能在布局文件中使用的类，也都是 android.view.View 类的继承者。

这个 com.example.android.apis.view.LabelView，在源文件 LabView.java 中实现，其主要片段如下所示：

```
    import android.view.View;
    public class LabelView extends View {
        public LabelView(Context context, AttributeSet attrs) {
            super(context, attrs);
            initLabelView();
            TypedArray a = context.obtainStyledAttributes(attrs,
                   R.styleable.LabelView);
            CharSequence s = a.getString(R.styleable.LabelView text);
            if (s != null) {
                setText(s.toString());
            }
            setTextColor(a.getColor(R.styleable.LabelView_textColor,
0xFF000000));
            int textSize =
                      a.getDimensionPixelOffset(R.styleable.LabelView_textSize,
0);
            if (textSize > 0) {
                setTextSize(textSize);
            }
            a.recycle();
        }
        protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);
            canvas.drawText(mText, getPaddingLeft(),
                            getPaddingTop() - mAscent, mTextPaint);
        }
    }
```

在 LabelView 的构造函数中，通过 context.obtainStyledAttributes 获得 LabelView 所特有的几个属性。R.styleable.LabelView 这些内容在 res/values/的 attrs.xml 文件中进行了定义，内容如下所示：

```
      <declare-styleable name="LabelView">
          <attr name="text" format="string" />
          <attr name="textColor" format="color" />
          <attr name="textSize" format="dimension" />
       </declare-styleable>
```

根据 LabView.java 实现的类名称，这样自定义的控件也可以在布局文件中使用，使用标签与类名相一致。R.styleable.LabelView_text，R.styleable.LabelView_textColor 和 R.styleable.LabelView_textSize 是在源代码中使

用的属性，它们与引用 LabelView 的布局文件中的 app:text，app:textColor 和 app:textSize 等几个属性相对应。

作为公共的属性，LabelView 在实现上也应该具有公共的函数来设置这几个属性。这些函数如下所示：

```
public void setText(String text) {
    mText = text;
    requestLayout();
    invalidate();
}
public void setTextSize(int size) {
    mTextPaint.setTextSize(size);
    requestLayout();
    invalidate();
}
public void setTextColor(int color) {
    mTextPaint.setColor(color);
    invalidate();
}
```

以上的几个函数和几个 XML 中的属性对应的，如果没有他们，这些属性就只能在 XML 文件中指定，而不能在 JAVA 源文件中设置。

在 Android 的应用程序层，可以通过扩展 android.view.View 或者它的扩展者来实现自己的 View。

第 8 章 视图组（ViewGroup）和布局（Layout）的使用

在 Android 中视图组是集合若干个控件在一起的元素，ViewGroup 有两种用法，一种是像普通的控件一样使用（如网页视图、旋转按钮、文本切换器、图像切换器、单选按钮组等），另一种是作为布局容器使用（各种布局）。

8.1 Android 的屏幕元素体系

在屏幕中控件的组织上，可以将各个视图（控件）组成一个视图组（ViewGroup），视图组是一个包含了其他视图的视图。

android.view.ViewGroup 扩展了 android.view.View，它本身也具有 View 的特性，区别仅在于它可以包含其他的控件。

ViewGroup 视图组具有一系列的扩展者：AdapterView、AbsoluteLayout、FrameLayout 、 LinearLayout 、 RelativeLayout 、 AdapterView<T extends android.widget.Adapter>。

Android GUI 程序的屏幕体系结构的组织遵循以下原则：

- 一个屏幕可以包含一个视图；
- 视图组本身也是一个视图；
- 视图组可以包含若干个视图。

Android 视图和视图组的关系如图所示：

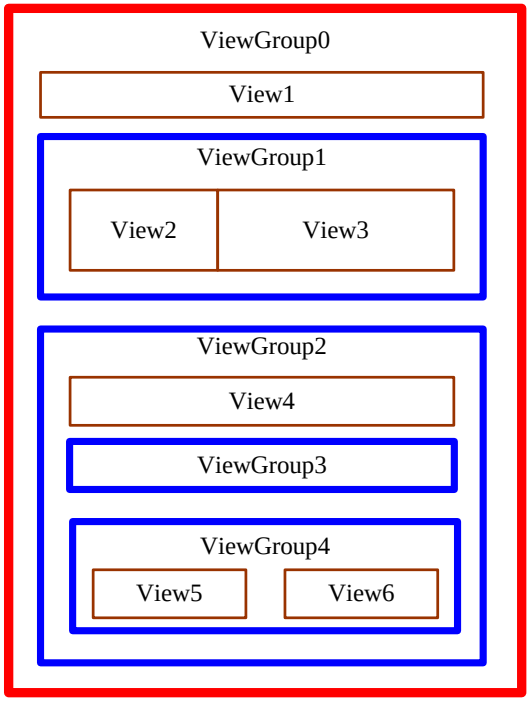


图　视图和视图组的关系

如图所示，外部最大的框表示整个屏幕，其中包含一个视图组 ViewGroup0，ViewGroup0 包含 3 个子视图，即 View1、ViewGroup1、ViewGroup2。ViewGroup1 本身也是视图组，以水平布局的方式包含了 View2 和 View3；ViewGroup2 本身也是视图组，以垂直的方式包含了 View4、ViewGroup3 和 ViewGroup4；ViewGroup4 本身也是视图组，以水平布局的方式包含了 View5 和 View6。

根据以上的原则，当屏幕需要包含多个视图时，必须组织在一个视图组中。由于视图组本身也是一个视图，因此视图组还可以包含视图组。在这里一个主要的限制是：在没有视图组的情况下，两个以上的视图（也包括视图组）是不能够并列的。

例如，在布局文件中，类似下面的写法是不可以的。

```
<?xml version="1.0" encoding="utf-8"?>
    <Button android:id="@+id/button"/>
    <EditText android:id="@+id/edit"/>
```

应该组织成以下的格式：

```
<?xml version="1.0" encoding="utf-8"?>
    <ViewGroup>
        <Button android:id="@+id/button"/>
        <EditText android:id="@+id/edit"/>
    </ViewGroup>
```

其中的 ViewGroup 可以是 ViewGroup 类，或者是它的扩展者，可以将 ViewGroup 及其扩展者统称为 ViewGroup。在 Android 中，有一些预置的 ViewGroup 可以直接像 View 一样使用（如 WebView），还有一些 ViewGroup 本身没有功能，只是提供屏幕上的各种布局（如 AbsoluteLayout、FrameLayout 等）。

8.2 几种独立使用的视图组

8.2.1. 网页视图

网页视图（WebView）是一个功能强大且常用的控件，它具有许多很好的特性，例如对 js 的支持，可用于制作简易浏览器等。

参考示例程序：WebView1（ApiDemo=>Views=>WebView）

源代码：com/example/android/apis/view/WebView1.java

布局文件：webview_1.xml

WebView 程序的运行结果如图所示：

图 WebView 程序的运行结果

WebView 的类扩展关系如下所示：

=> android.view.View

=> android.view.ViewGroup

=> android.widget.AbsoluteLayout

=> android.webkit.WebView

WebView 本身扩展了 AbsoluteLayout(绝对布局)，因此也是一个 ViewGroup，但是 WebView 不用于包含其他的视图，而是像一个普通的控件一样使用。

布局文件 webview_1.xml 的内容片断如下：

```
<WebView android:id="@+id/wv1"
    android:layout_height="wrap_content"
    android:layout width="fill parent"
    />
<WebView android:id="@+id/wv2"
    android:layout_height="wrap_content"
    android:layout width="fill parent"
    />
<WebView android:id="@+id/wv3"
    android:layout height="wrap content"
    android:layout_width="fill_parent"
    />
```

事实上，在本例中是使用了若干个 WebView 标签来实现显示的功能，每一个 WebView 标签显示为屏幕中一行的内容。

Java 源代码中的部分内容如下所示：

```
wv = (WebView) findViewById(R.id.wv1);
wv.loadData("<a href='x'>Hello World! - 1</a>", mimeType, encoding);
```

WebView 的作用是可以在 Android 中支持一个 HTML 格式的元素。由此，虽然 WebView 也是一个视图组，但是从使用上基本等同普通的控件。

8.2.2. 旋转按钮

旋转按钮（Spinner）是具有类似菜单的按钮，可以选择其中的一项，一般可以使用单向和双向的箭头进行选择。

参考示例程序：Spinner1（ApiDemo=>Views=>Spinner）

源代码：com/example/android/apis/view/Spinner1.java

布局文件：radio_group_1.xml

Spinner1 程序的运行结果如图所示：

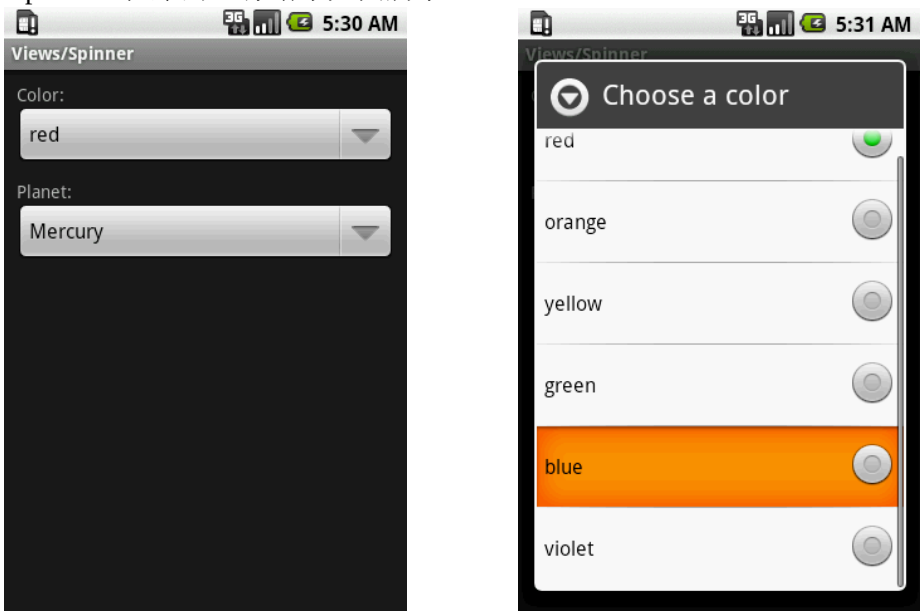


图　Spinner1 程序的运行结果

Android 中的旋转按钮做成了一个下拉菜单的形式，其功能和其他 GUI 系统中的旋转按钮类似。本例中直接使用 Spinner 类构造了两个可以具有若干个选项的旋转按钮，Spinner 类的扩展关系如下所示：

=> aandroid.view.View

=> aandroid.view.ViewGroup

=> aandroid.widget.AdapterView<T extends android.widget.Adapter>

=> aandroid.widget.AbsSpinner

=> aandroid.widget.Spinner

AdapterView<>是一个视图的模板，它本身扩展了 ViewGroup，具体的内容由其中定义的 android.widget.Adapter 类来确定。AbsSpinner 扩展了 AdapterView<>，Spinner 又扩展了 AbsSpinner。

在本例的 Java 源代码中，内容如下所示：

```
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.spinner 1);

            Spinner s1 = (Spinner) findViewById(R.id.spinner1);
            ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
                    this, R.array.colors, android.R.layout.simple_spinner_item);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
            s1.setAdapter(adapter);

            Spinner s2 = (Spinner) findViewById(R.id.spinner2);
            adapter = ArrayAdapter.createFromResource(this, R.array.planets,
                    android.R.layout.simple_spinner_item);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
            s2.setAdapter(adapter);
        }
```

setAdapter()是 AdapterView<>中的函数，通过建立一个 ArrayAdapter< CharSequence >将所需要的内容设置到其中。res/values/array.xml 中定义的字符串数组（string-array）colors 表示 Spinner 中的内容。

```
        <string-array name="colors">
            <item>red</item>
            <item>orange</item>
            <item>yellow</item>
```

```
            <item>green</item>
            <item>blue</item>
            <item>violet</item>
        </string-array>
```

simple_spinner_item 和 simple_spinner_dropdown_item 是 Android 中默认的样式，Android 中的 Spinner 在调用的时候，会显示为一弹出的窗口，其中包含了各个选项。

8.2.3. 文本切换器

文本切换器（TextSwitcher）是 Android 中一个集成化较高的控件，可以在多个文本之间切换，还可以设置动画的效果。

参考示例程序：TextSwitcher1（ApiDemo=>Views=>TextSwitcher）

源代码：com/example/android/apis/view/TextSwitcher1.java

布局文件：text_switcher_1.xml

TextSwitcher1 程序的运行结果如图所示：

图 TextSwitcher1 程序的运行结果

这个示例的布局文件 text_switcher_1.xml 如下所示：

```
        <Button android:id="@+id/next"
            android:layout width="wrap content"
            android:layout_height="wrap_content"
            android:text="@string/text_switcher_1_next_text" />

        <TextSwitcher android:id="@+id/switcher"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
```

图的中间部分表示了一个文字切换器，使用 TextSwitcher 来实现。具体显示的内容由当前的 Activity 实现 ViewSwitcher.ViewFactory 接口来完成，实现其中的 makeView()方法，返回一个 TextView 类型。

TextSwitcher 类的扩展关系如下所示：

=> android.view.View
=> android.view.ViewGroup
=> android.widget.FrameLayout
=> android.widget.ViewAnimator
=> android.widget.ViewSwitcher
=> android.widget.TextSwitcher

本示例中为 TextSwitcher 类设置了动画：

```
    public class TextSwitcher1 extends Activity implements
ViewSwitcher.ViewFactory,
            View.OnClickListener {
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            setContentView(R.layout.text_switcher_1);
            mSwitcher = (TextSwitcher) findViewById(R.id.switcher);
            mSwitcher.setFactory(this);
            Animation in = AnimationUtils.loadAnimation(this,
                    android.R.anim.fade_in);
            Animation out = AnimationUtils.loadAnimation(this,
                    android.R.anim.fade_out);
            mSwitcher.setInAnimation(in);
            mSwitcher.setOutAnimation(out);
            // ......

            updateCounter();
        }
    }
```

这里是 mSwitcher.setFactory(this)设置了所使用的 ViewSwitcher.ViewFactory 接口，这个接口由当前的 TextSwitcher1 活动来继承实现，主要实现其中的 makeView()函数：

```
        public View makeView() {
```

```
        TextView t = new TextView(this);
        t.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL);
        t.setTextSize(36);
        return t;
    }
```

makeView()函数的返回类型是 View，如果在 TextSwitcher1 中使用，要求返回的类型为 TextView。

这个示例中主要变化的内容，是通过 TextSwitcher 类的 setText()函数来完成的：

```
    private void updateCounter() {
        mSwitcher.setText(String.valueOf(mCounter));
    }
```

8.2.4. 图像切换器

图像切换器（ImageSwitcher）和文本切换器类似，但是显示的内容是多个图片中的一个。

参考示例程序：ImageSwitcher1（ApiDemo=>Views=>ImageSwitcher）

源代码：com/example/android/apis/view/ImageSwitcher1.java

布局文件：image_switcher_1.xml

ImageSwitcher1 程序的运行结果如图 26 所示。

图　ImageSwitcher1 程序的运行结果

这个示例的布局文件 text_switcher_1.xml 如下所示：

```
<ImageSwitcher android:id="@+id/switcher"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout alignParentTop="true"
    android:layout_alignParentLeft="true"
/>

<Gallery android:id="@+id/gallery"
    android:background="#55000000"
    android:layout width="fill parent"
    android:layout_height="60dp"
    android:layout_alignParentBottom="true"
    android:layout alignParentLeft="true"

    android:gravity="center_vertical"
    android:spacing="16dp"
/>
```

界面上面部分表示了一个图像切换器，使用 ImageSwitcher 来实现。

ImageSwitcher 类的扩展关系如下所示：

=> android.view.View
=> android.view.ViewGroup
=> android.widget.FrameLayout
=> android.widget.ViewAnimator
=> android.widget.ViewSwitcher
=> android.widget.ImageSwitcher

ImageSwitcher 具体显示的内容也是由当前的 Activity 实现 ViewSwitcher.ViewFactory 接口来完成的，实现其中的 makeView()方法，返回一个 ImageView 类型。

```
public class ImageSwitcher1 extends Activity implements
        AdapterView.OnItemSelectedListener, ViewSwitcher.ViewFactory {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.image switcher 1);
        mSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
        mSwitcher.setFactory(this);
        mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_in));
        mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade out));
        // ......
    }
```

```
        public View makeView() {
            ImageView i = new ImageView(this);
            i.setBackgroundColor(0xFF000000);
            i.setScaleType(ImageView.ScaleType.FIT_CENTER);
            i.setLayoutParams(new
mageSwitcher.LayoutParams(LayoutParams.FILL_PARENT,
                    LayoutParams.FILL_PARENT));
            return i;
        }
    }
```

这个示例的下面部分是一个 Gallery（android.widget.Gallery，Android 中另外一个控件，扩展 AbsSpinner 实现，与 Spinner 为兄弟关系）。为了实现这个类中的内容，本例中还实现了一个 ImageAdapter 类。

8.3 作为简单容器使用的视图组

8.3.1. 单选按钮组

单选按钮组（RadioButton）是一组逻辑上相关的按钮，它们之中只能有一个被选中，单选按钮通常单选按钮被设计成圆形的外观。因此需要一个类将各个单选按钮包含在一起。

参考示例程序：RadioGroup1（ApiDemo=>Views=>Radio Group）

源代码：com/example/android/apis/view/RadioGroup1.java

布局文件：radio_group_1.xml

RadioGroup1 程序的运行结果如图所示：

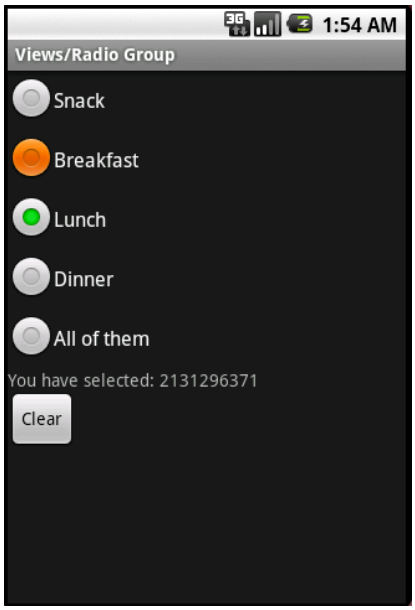


图 23　RadioGroup1 程序的运行结果

此程序使用 RadioGroup 将几个单选按钮组织在一起，RadioGroup 的扩展关系如下：

=> android.view.View
=> android.view.ViewGroup
=> android.widget.LinearLayout
=> android.widget.RadioGroup

RadioGroup 本身扩展了线性布局，它的功能比较单一，是为了保证多个 RadioButton 只有一个被选中，这种关系通常也被称为多选互斥（multiple-exclusion）。

使用 RadioGroup 组成一个单选列表，需要将 RadioButton 放置在一个 RadioGroup 中。本例的布局文件内容如下所示：

```
<RadioGroup android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" android:checkedButton="@+id/lunch"
    android:id="@+id/menu">
    <RadioButton android:text="@string/radio_group_1_breakfast"
              android:id="@+id/breakfast" />
    <RadioButton android:text="@string/radio group 1 lunch"
              android:id="@id/ lunch" />
    <RadioButton android:text="@string/radio_group_1_dinner"
              android:id="@+id/ dinner" />
```

```
        <RadioButton android:text="@string/radio_group_1_all"
                     android:id="@+id/ all" />
        <TextView android:text="@string/radio_group_1_selection"
                     android:id="@+id/ choice" />
    </RadioGroup>
```

RadioGroup 中的 XML 属性 android:checkedButton 表示这一组单选按钮 RadioButton 组中被选中的按钮，包含在一个 RadioGroup 之中的所有单选按钮只能有一个被选中。

根据扩展关系 RadioGroup 本身即是 ViewGroup，也是 LinearLayout，因此在 RadioGroup 中也可以包含 RadioButton 之外的其他控件。

8.3.2. 使用滚动条

当屏幕上控件的内容超过屏幕本身的尺寸时，一般可以通过出现滚动条（ScrollBar）供用户拖动来显示没有显示的内容。Android 使用滚动视图（ScrollView）来支持滚动条。

参考示例程序：ScrollView（ApiDemo=>Views=>ScrollView=>各个程序）

源代码：com/example/android/apis/view/ScrollBar1.java
com/example/android/apis/view/ScrollBar2.java
com/example/android/apis/view/ScrollBar3.java

布局文件：scrollbar1.xml、scrollbar2.xml 和 scrollbar3.xml

ScrollView 相关的程序的运行结果如图所示：

图 ScrollView 程序的运行结果

scrollbar1.xml 的内容如下所示：

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    <LinearLayout
        android:id="@+id/layout"
        android:orientation="vertical"
```

```
          android:layout_width="fill_parent"
android:layout height="wrap content">
          <TextView
              android:layout_width="fill_parent"
              android:layout height="wrap content"
              android:text="@string/scrollbar_1_text"/>
          <TextView
              android:layout width="fill parent"
              android:layout_height="wrap_content"
              android:text="@string/scrollbar_1_text"/>
       </LinearLayout>
   </ScrollView>
```

在 scrollbar2.xml 和 scrollbar3.xml 文件的内容也与之类似。

ScrollView 类的扩展关系如下所示：

=> android.view.ViewGroup

=> android.widget.FrameLayout

=> android.widget.ScrollView

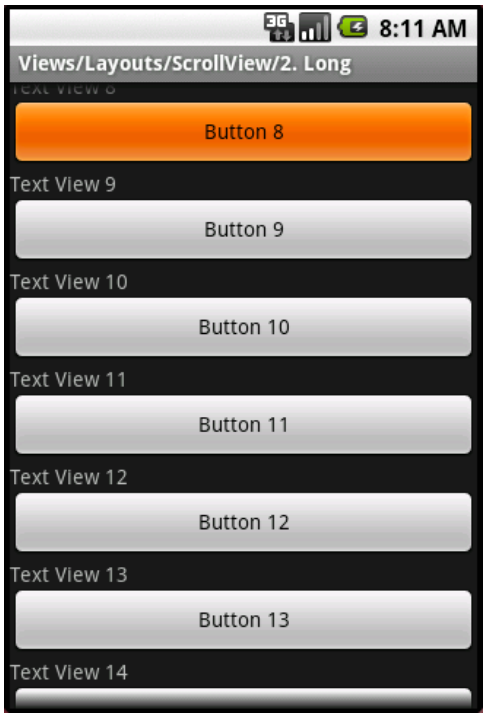
ScrollView 类通常在 XML 文件中使用，当屏幕上的内容预计超过屏幕尺寸时，用一个 ScrollView 将其他内容包含起来，这样就可以出现滚动条。

参考示例程序：ScrollView（Views=>Layout=>ScrollView=>2）

源代码：com/example/android/apis/view/ScrollView2.java

布局文件：scroll_view_2.xml

ScrollView2 程序的运行结果如图所示：



scroll_view_2.xml 布局文件的内容如下所示：

```
    <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout width="fill parent"
android:layout_height="wrap_content"
        android:scrollbars="none">
        <LinearLayout
            android:id="@+id/layout"
            android:orientation="vertical"
            android:layout width="fill parent"
            android:layout_height="wrap_content">
            <TextView
                android:layout width="fill parent"
                android:layout_height="wrap_content"
                android:text="@string/scroll_view_2_text_1"/>
            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/scroll view 2 button 1"/>
        </LinearLayout>
    </ScrollView>
```

这里指定了 android:scrollbars="none"表示本屏幕中没有滚动杆，即使这样依然可以使用上下键和触摸屏进行上下移动。

源文件 ScrollView2.java 中的主要内容如下所示：

```
public class ScrollView2 extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.scroll view 2);

        LinearLayout layout = (LinearLayout) findViewById(R.id.layout);
        for (int i = 2; i < 64; i++) {
            TextView textView = new TextView(this);
            textView.setText("Text View " + i);
            LinearLayout.LayoutParams p = new LinearLayout.LayoutParams(
                    LinearLayout.LayoutParams.FILL PARENT,
                    LinearLayout.LayoutParams.WRAP_CONTENT
            );
            layout.addView(textView, p);
            Button buttonView = new Button(this);
            buttonView.setText("Button " + i);
            layout.addView(buttonView, p);
        }
    }
}
```

在这里是直接获得了 LinearLayout 的句柄，在其中用循环的方式增加了若干组（2-64）文本框和按钮，这样就形成了一个在界面上的长列表。本例子的第一组文本框和按钮是在布局文件中指定的，其他是在代码中指定的。

Android 应用虽然支持滚动视图，但是在手机上，一般的界面并不一定适合使用这种的方式，在大多数情况下还是应该协调屏幕的尺寸和元素，保证一个屏幕可以完全显示内容。

8.4 布局（Layout）

布局（Layout）是各个控件在屏幕上的位置关系，视图组的几个扩展类与布局相关。在 Android 中布局通常有以下几种不同的情况：

- FrameLayout（框架布局）：系统默认的在屏幕上就有空白区显示它；
- LinearLayout（线性布局）：让所有的子视图都成为单一的方向，即垂直的或者水平的；
- AbsoluteLayout（绝对布局）：让子视图使用 x/y 坐标确定在屏幕上的位置；
- RelativeLayout（相对布局）：让子视图的位置和其他的视图相关；
- TableLayout（表单布局）：位置是它的子视图的行或列。

FrameLayout、LinearLayout、RelativeLayout、AbsoluteLayout、TableLayout 都是扩展了 ViewGroup 的类，因此这些视图可以用于包含其他的控件，并可以控制其他

的控件的位置关系。

布局的内容一般通过在布局文件中控制即可，在控制布局时 android:layout_width 和 android:layout_height 等表示尺寸属性，除了使用实际的尺寸值外，还有两个常用的选项：

- "fill_parent"：表示能填满父视图的最大尺寸；
- "wrap_content"：表示仅包裹子内容的最小尺寸。

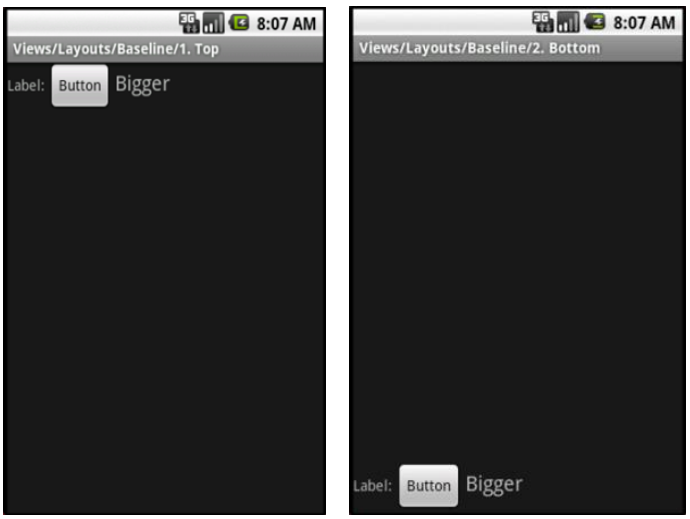
这两个值既可以在视图组中使用，也可以在普通视图中使用，如果在视图中使用"wrap_content"，表示包裹其中的内容，例如按钮需要包裹上面的文字。

8.4.1. 基本的布局内容

基本的布局内容用于控制每个元素的位置。示例程序位于 Views=>Layout=>Baseline 中：

布局文件：baseline_X.xml

其中的一些显示效果如图所示：



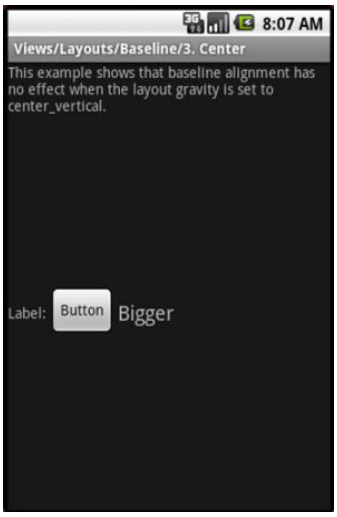


图　基本布局程序的运行结果

左图的程序使用了默认的布局参数，因此是上对齐和左对齐的效果，中图的程序使用了 android:layout_gravity 为底部对齐，右图中使用了两个布局嵌套的方式：

```
<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="wrap content"
        android:layout_height="wrap_content"
        android:text="@string/baseline_3_explanation" />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout weight="1.0"
        android:layout_height="0dip">
        <TextView
            android:layout width="wrap content"
            android:layout_height="wrap_content"
            android:layout_marginRight="3dip"
            android:layout gravity="center vertical"
            android:text="@string/baseline_3_label" />
        <Button
            android:layout width="wrap content"
            android:layout_height="wrap_content"
            android:layout_marginRight="3dip"
            android:layout gravity="center vertical"
            android:text="@string/baseline_3_button" />
        <TextView
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:textSize="20sp"
            android:text="@string/baseline_3_bigger" />
    </LinearLayout>
</LinearLayout>
```

以上几个程序实际上使用的是线性布局（LinearLayout）。以上不同元素位置的控制通过定义 android:layout_gravity 属性来完成，android:layout_gravity 可以在各个 View 中使用：top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical、clip_horizontal，这些选项用于处理竖直和水平方向的对齐方式。

8.4.2．线性布局（LinearLayout）

线性布局是 Android 中最常使用的布局，示例程序位于 Views=>Layout=>LinearLayout 中。
线性布局程序的运行结果如图所示：

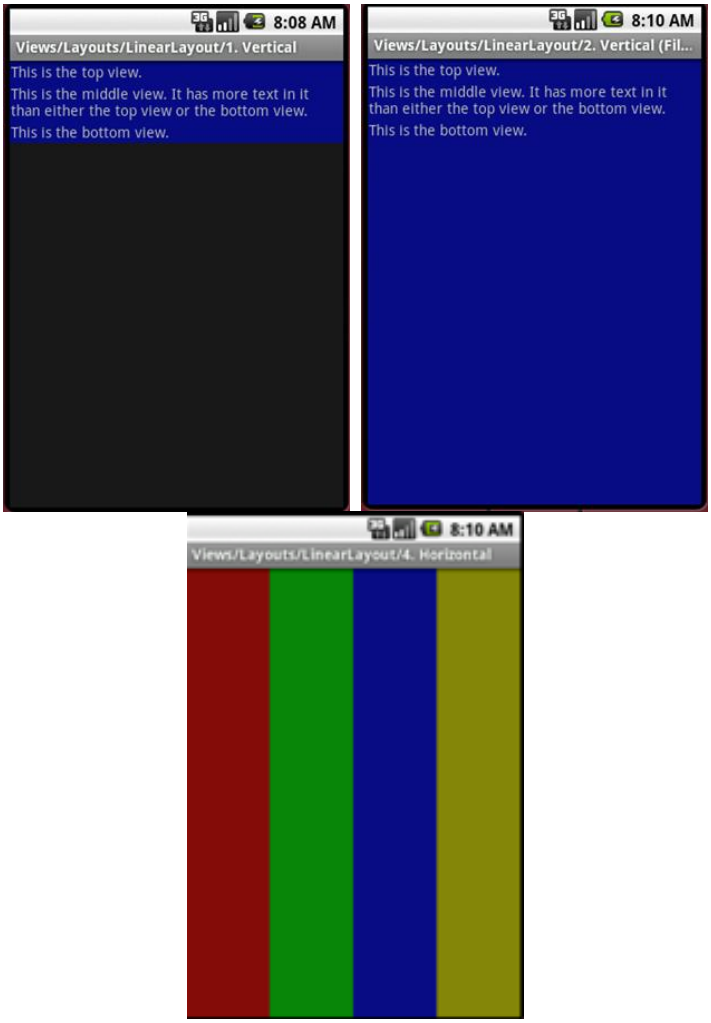


图　线性布局程序的运行结果

这几个示例程序的布局文件分别为 linear_layout_1.xml、linear_layout_2.xml 和 linear_ layout_4.xml。linear_ layout_4.xml 的内容如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout width="fill parent"
    android:layout_height="fill_parent">
    <TextView
        android:background="@drawable/red"
        android:layout_width="0dip"
```

```
            android:layout_height="fill_parent"
            android:layout_weight="1"/>
        <TextView
            android:background="@drawable/green"
            android:layout_width="0dip"
            android:layout_height="fill_parent"
            android:layout_weight="1"/>
        <!-- ……省略部分内容 -->
    </LinearLayout>
```

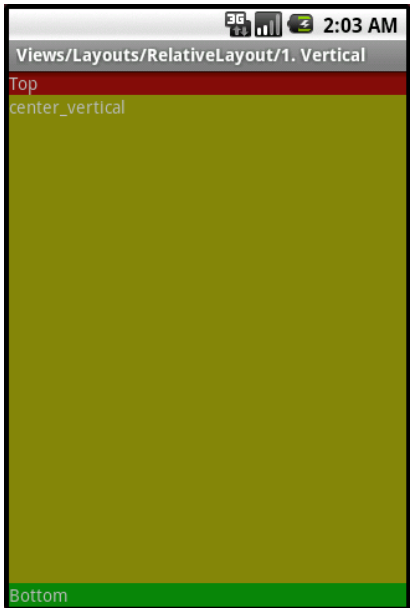
左图和中图的差别在于左图的竖直方向使用了"wrap_content"，中图使用了"fill_parent"；右图使用了 android:orientation="horizontal"定义屏幕中的方向为水平，并设置竖直方向为"fill_parent"，因此其中的内容以竖直方向显示。

8.4.3. 相对布局（RelativeLayout）

相对布局的特点是可以让控件之间互相确定关系，这样可以保证在屏幕的局部范围内几个控件之间的关系不受外部影响，

相对布局的示例程序位于 Views=>Layou=>RelativeLayout 中，其中的两个程序的运行结果如图所示：

这两个示例程序的布局文件分别为 relative_layout_1.xml 和 relative_layout_2.xml。



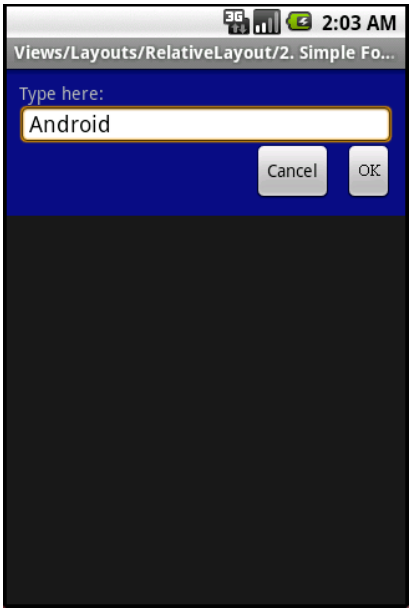


图　相对布局的运行结果

左图通过设置android:layout_alignParentTop和android:layout_alignParentBottom两个属性为"true"，让控件对齐到父UI的上端和下端。相关的属性还有android:layout_ alignParentRight和android:layout_alignParentLeft。

```
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView
            android:id="@+id/view1"
            android:background="@drawable/red"
            android:layout width="fill parent"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:text="@string/relative layout 1 top"/>
        <TextView
            android:id="@+id/view2"
            android:background="@drawable/green"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout alignParentBottom="true"
            android:text="@string/relative_layout_1_bottom"/>
        <TextView
            android:id="@+id/view3"
            android:background="@drawable/yellow"
            android:layout_width="fill_parent"
            android:layout height="0dip"
            android:layout_above="@id/view2"
            android:layout_below="@id/view1"
            android:text="@string/relative layout 1 center"/>
    </RelativeLayout>
```

右图中的两个按钮使用了相对对齐的方式，它们之间的关系如下所示：

```
    <Button android:id="@+id/ok"
        android:layout_width="wrap_content"
android:layout_height="wrap_content"
        android:layout_below="@id/entry" android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
android:text="@string/relative layout 2 ok" />
    <Button
        android:layout_width="wrap_content"
android:layout height="wrap content"
        android:layout_toLeftOf="@id/ok" android:layout_alignTop="@id/ok"
        android:text="@string/relative_layout_2_cancel" />
```

“Cancel”按钮的位置是相对“Ok”按钮来确定的，toLeftOf属性表示在“Ok”

按钮的左侧，layout_alignTop 属性表示和“Ok”按钮上对齐。

8.4.4. 表单布局（Table Layout）

一个表单布局（TableLayout）包含了若干个 TableRow 对象，每一个 TableRow 对象定义了其中一行。TableLayout 中也包含了不显示的行和列的边沿。

参考示例程序：TableLayout1（Views=>Layout=>TabLayout=>01.basic）

源代码：com/example/android/apis/view/TableLayout1.java

布局文件：table_layout_1.xml

表单布局程序的运行结果如图所示：

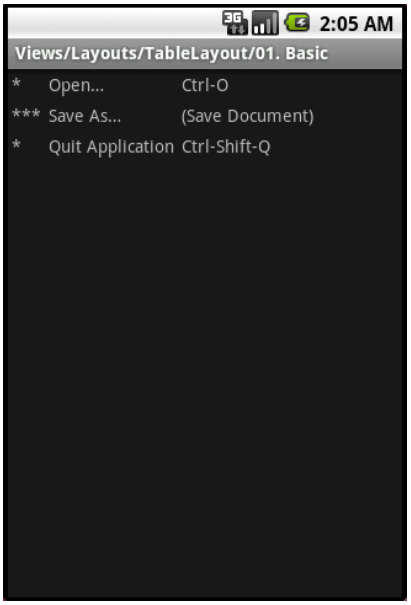


图 表单布局程序的运行结果

这个示例程序的布局文件 table_layout_1.xml 的内容如下所示：

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="fill parent" android:layout height="fill parent">
    <TableRow>
        <TextView android:text="@string/table_layout_1_star"
                  android:padding="3dip" />
        <TextView android:text="@string/table_layout_1_open"
                  android:padding="3dip" />
        <TextView android:text="@string/table_layout_1_open_shortcut"
```

```
                android:padding="3dip" />
        </TableRow>
        <TableRow>
            <TextView android:text="@string/table_layout_1_triple_star"
                android:padding="3dip" />
            <TextView android:text="@string/table_layout_1_save"
                android:padding=" 3dip" />
            <TextView android:text="@string/table layout 1 save shortcut"
                android:padding="3dip" />
        </TableRow>
        <!-- ……省略部分内容 -->
    </TableLayout>
```

TableLayout 中包含了若干个 TableRow，每个 TableRow 中又包含了若干个 TextView，这样在 UI 上实际上就形成了一个隐性的表格，表格中的每一个单元格的内容是一个 View。这种表单布局，其实是用了类似 HTML 中的表格的方式，这样可以准确地完成复杂的对齐问题。

8.5 网格（Grid）视图组

本节介绍的网格（Grid）视图组可以将某种控件按照网格的形式组织起来，平铺在屏幕上。

参考示例程序：Icon Grid（ApiDemo=>Views=>Grid =>Icon Grid）

源代码： com/example/android/apis/view/Grid1.java

布局文件：grid_1.xml

Icon Grid 程序的运行结果如图所示：

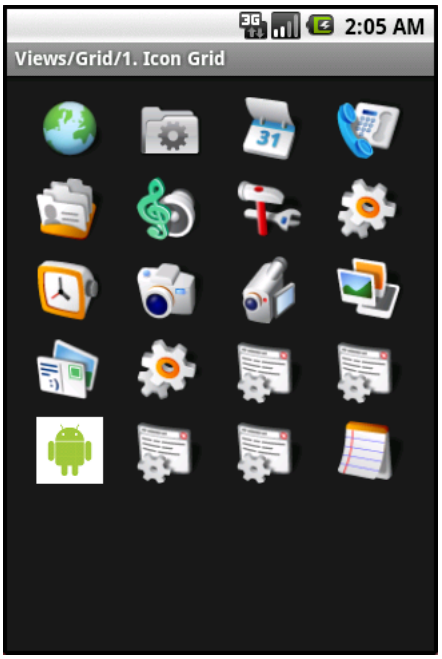


图 Icon Grid 程序的运行结果

本示例显示了当前系统中所包含的应用程序的图标。布局文件 grid_1.xml 的内容如下所示：

```
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myGrid"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:numColumns="auto_fit"
    android:columnWidth="60dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
    />
```

这里使用的是 GridView 标签，网格视图 GridView 的扩展关系如下所示：

=> android.view.View

=> android.view.ViewGroup

=> android.widget.AdapterView<T extends android. widget.Adapter>

=> android.widget.AbsListView

=> android.widget.GridView

本例中使用的 android:numColumns 、android:columnWidth 、android:horizontalSpacing 和 android:verticalSpacing 类是 GridView 的特定属性，分别表示了列的数目，列的宽度，水平间距和竖直间距，本例中的 android:numColumns 设置为"auto_fit"表示根据宽度和间距等信息，自动适应。

AbsListView 是 ListView 和 GridView 的共同父类，它使用 ListAdapter 作为其中的数据。ListAdapter 作为列表的 UI 和数据的桥梁，通过实现这个类来构建界面上的 AbsListView。android.widget.ListAdapter 实现了 android.widget.Adapter 接口。

在本示例程序中，在布局文件中定义了 GridView，在 Java 代码中设置一个 BaseAdapter 作为 GridView 中的数据。

JAVA 源代码中实现的主要内容如下所示：

```
    public class Grid1 extends Activity {
        GridView mGrid;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            loadApps();
            setContentView(R.layout.grid 1);
            mGrid = (GridView) findViewById(R.id.myGrid);
            mGrid.setAdapter(new AppsAdapter());      // 设置 GridView 后面的数据
        }
    }
```

这里调用的 setAdapter()方法是 android.widget.AdapterView<T extends android.widget.Adapter>中的方法，参数是所指定的一个模板类型 android.widget.Adapter。

为了实现 GridView 的效果，需要构建一个 BaseAdapter，也就是 android.widget.BaseAdapter。这个类表示了 Grid 中的所包含的内容，GridView 的实现如下所示：

```
        public class AppsAdapter extends BaseAdapter {
            public AppsAdapter() { }
            public View getView(int position, View convertView, ViewGroup parent)
{
                ImageView i;
                if (convertView == null) {
                    i = new ImageView(Grid1.this);
                    i.setScaleType(ImageView.ScaleType.FIT CENTER);
                    i.setLayoutParams(new GridView.LayoutParams(50, 50));
                } else {
                    i = (ImageView) convertView;
                }
                ResolveInfo info = mApps.get(position);

i.setImageDrawable(info.activityInfo.loadIcon(getPackageManager()));
                return i;
```

```
        }
        public final int getCount() {
            return mApps.size();
        }
        public final Object getItem(int position) {
            return mApps.get(position);
        }
        public final long getItemId(int position) {
            return position;
        }
    }
```

AppsAdapter 主要需要实现 getView()函数，其中的 position 参数表示需要取得的 View 的位置。本例中的实现是获取系统中所有的应用程序的图标，也就是分类为 Intent.CATEGORY_LAUNCHER 的应用程序。

AbsListView 继承了 AdapterView，这是一个类的模板，如果需要让 GridView 实现对事件的影响，需要继承一个 GridView，并且实现 AdapterView.OnItemClickListener 、 AdapterView.OnItemLongClickListener 和 AdapterView.OnItemSelectedListener 等几个接口。这几个接口如下所示：

```
AdapterView.OnItemClickListener {
    abstract void  onItemClick(AdapterView<?> parent, View view,
                               int position, long id) {}
}
AdapterView.OnItemLongClickListener {
    abstract boolean  onItemLongClick(AdapterView<?> parent, View view,
                                      int position, long id) {}
}
AdapterView.OnItemSelectedListener {
    abstract void  onItemSelected(AdapterView<?> parent, View view,
                                  int position, long id)  {}
    abstract void  onNothingSelected(AdapterView<?> parent) {}
}
```

参考示例程序：Photo Grid（Views=>Grid =>Photo Grid）

源代码： com/example/android/apis/view/Grid2.java

布局文件：grid_2.xml

Photo Grid 程序的运行结果如图所示：

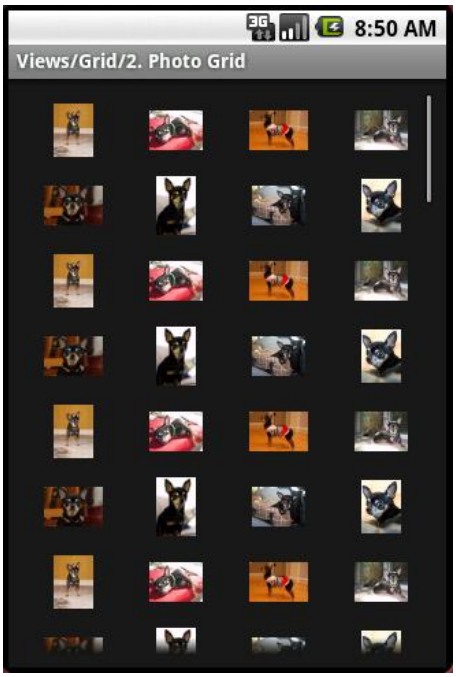


图 Photo Grid 程序的运行结果

布局文件 grid_2.xml 如下所示：

```
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myGrid"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:numColumns="auto_fit"
    android:columnWidth="60dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
    />
```

Grid2.java 使用了：

```
public class Grid2 extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.grid_2);
        GridView g = (GridView) findViewById(R.id.myGrid);
        g.setAdapter(new ImageAdapter(this));
```

```
        }
    }
```

这里定义的 ImageAdapter 继承了 BaseAdapter，内容如下所示：

```
    public class ImageAdapter extends BaseAdapter {
        public ImageAdapter(Context c) {
            mContext = c;
        }
        public int getCount() {
            return mThumbIds.length;
        }
        public Object getItem(int position) {
            return position;
        }
        public long getItemId(int position) {
            return position;
        }
        public View getView(int position, View convertView, ViewGroup parent)
{
            ImageView imageView;
            if (convertView == null) {
                imageView = new ImageView(mContext);
                imageView.setLayoutParams(new GridView.LayoutParams(45, 45));
                imageView.setAdjustViewBounds(false);
                imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                imageView.setPadding(8, 8, 8, 8);
            } else {
                imageView = (ImageView) convertView;
            }
            imageView.setImageResource(mThumbIds[position]);
            return imageView;
        }
        private Context mContext;
        private Integer[] mThumbIds = { // 图片的 id 数组
        }
```

这里是使用一系列的图片，作为 GridView 中的内容。

8.6 列表（List）视图组

本节介绍的列表（List）视图组可以将某种控件按照列表的形式组织起来，它与网格视图组类似，但是附加了更方便的组织方式。

参考示例程序：ListArray（ApiDemo=>Views=>List=>ListArray）

源代码：com/example/android/apis/view/List1.java

ListArray 程序的运行结果如图所示：

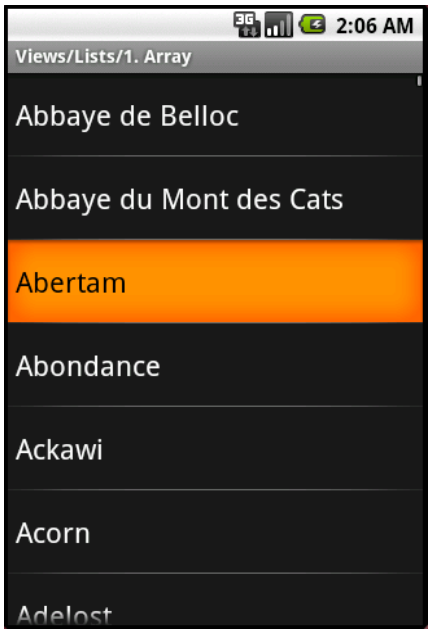


图 ListArray 程序的运行结果

本示例显示一系列的字符串，实现的方法是在代码中完成的，没有使用布局文件。

列表视图 ListView 的扩展关系如下所示：

=> android.view.View
=> android.view.ViewGroup
=> android.widget.AdapterView<T extends android. widget.Adapter>
=> android.widget.AbsListView
=> android.widget.ListView

ListView 也扩展了 AbsListView，列表视图的使用方法和网格视图具有很相似的共同点。

ListView 本身的使用方法可以和 GridView 一样，通过构建一个 android.widget.BaseAdapter 来完成。在实际的使用过程中，可以使用 ListActivity 这种更简单的方式。

在使用列表类 ListView 时通常使用 ListActivity 来代替 Activity，ListActivity 扩展了 Activity 可以方便 ListView 的使用，主要的方法包括以下几个：

```
void  setListAdapter(ListAdapter adapter)
        // 设置 ListAdapter 作为数据
void  onListItemClick(ListView l, View v, int position, long id)
```

```
            // Item 选择时的函数
```

本示例程序的实现是一个字符串列表，没有布局文件，直接使用 ListActivity 进行操作，主要的实现部分如下所示：

```
    public class List1 extends ListActivity {            // 扩展实现 ListActivity
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setListAdapter(new ArrayAdapter<String>(this,
                  android.R.layout.simple_list_item_1, mStrings));     //  设 置
ListAdapter
            getListView().setTextFilterEnabled(true);     //获得 ListView 并进行设
置
        }
        private String[] mStrings = {                     //....列表项文本};
    }
```

ArrayAdapter 是一个模板类（android.widget.ArrayAdapter<T>），它也是 android.widget. BaseAdapter 的实现者。如果需要实现对列表项选择的操作，可以通过实现 ListActivity 的 onListItemClick()等函数完成。

ListActivity 类实际上集成了 Activity 和 ListView 的功能，其内部包含了一个 ListView，使用这个类可以直接构造界面中的列表视图。

ListView 也可以有更灵活的方式进行使用：

参考示例程序：ListArray（Views=>List=>ListArray）

源代码：com/example/android/apis/view/List8.java

布局文件：list8.xml

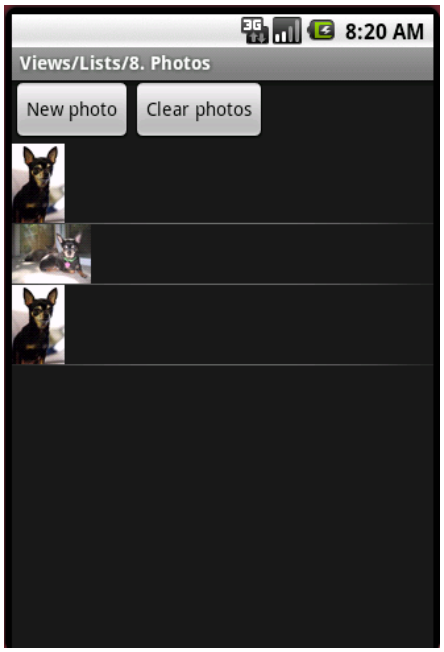


图 使用图片作为 Item 的 ListArray 程序的运行结果

布局文件 list8.xml 如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <Button android:id="@+id/add"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/list 8 new photo"/>
        <Button android:id="@+id/clear"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/list 8 clear photos"/>
    </LinearLayout>
    <FrameLayout
        android:layout width="fill parent"
        android:layout_height="0dip"
        android:layout_weight="1" >
        <ListView android:id="@android:id/list"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
```

```
            android:drawSelectorOnTop="false"/>
        <TextView android:id="@+id/empty"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="@string/list 8 no photos"/>
    </FrameLayout>
</LinearLayout>
```

在这个布局文件，定义了上面的 2 个按钮，主体部分 ListView 部分是空着的，需要在 JAVA 源文件中设置其中的内容。其中的 ListView 的 id 是"@android:id/list"，这样当活动是一个 ListActivity 的时候，将可以直接使用这个 ListView 作为默认的 ListView，进而获得和 ListActivity 的交互。如果不是用这样的 id 也可以，但是需要在源代码中使用普通的 Activity 并调用函数进行设置。

List8.java 中的构造函数如下所示：

```
public class List8 extends ListActivity {
    PhotoAdapter mAdapter;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list_8);
        getListView().setEmptyView(findViewById(R.id.empty)); //获得 ListView
并设置
        mAdapter = new PhotoAdapter(this);
        setListAdapter(mAdapter);
        // ......省略部分内容
    }
}
```

本例的活动是一个 ListActivity，这里使用的 getListView()将返回布局文件中定义 id 是"@android:id/list"的 ListView。

其中 mAdapter 是自定义的一个 BaseAdapter 类型，也是在这个类中实现的。

```
    public class PhotoAdapter extends BaseAdapter {
        private Integer[] mPhotoPool = {
                // 图片的资源 ID 数组 };
        private ArrayList<Integer> mPhotos = new ArrayList<Integer>();
        // ...... 省略部分内容
        public long getItemId(int position) {
            return position;
        }
        public View getView(int position, View convertView, ViewGroup parent)
{
            ImageView i = new ImageView(mContext);
            i.setImageResource(mPhotos.get(position));
            i.setAdjustViewBounds(true);
            i.setLayoutParams(
                    new AbsListView.LayoutParams(LayoutParams.WRAP CONTENT,
                    LayoutParams.WRAP_CONTENT));
            i.setBackgroundResource(R.drawable.picture_frame);
```

```
            return i;
        }
        // ..... 省略部分内容
    }
```

这里的 getView()函数所返回的是 ImageView 类型，这样在列表中显示的内容就可以是一组图片了。

8.7 使用 Tab 组织 UI

Tab 用于在一个屏幕中将不同的子屏幕组织到一起，用不同的 Tab 区分。

参考示例程序：Content By Intent（ApiDemo=>Views=>Tabs=>Content By Intent）

源代码：com/example/android/apis/view/Tab3.java

Tab3 程序的运行结果如图所示：

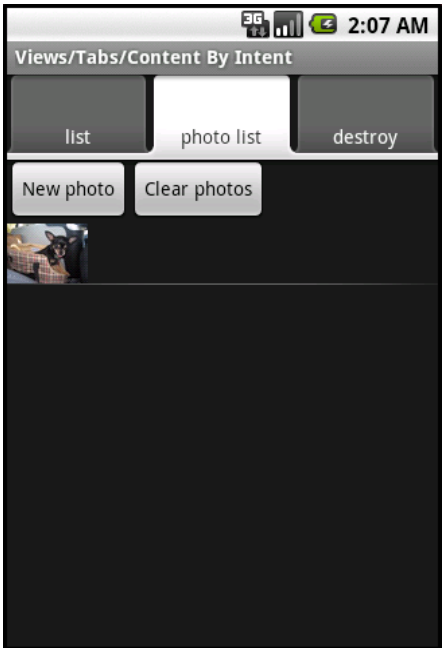


图 Tab（Content By Intent）程序的运行结果

在这个程序中使用了 3 个标签，每个标签启动一个活动作为其中的内容。主要的代码如下所示：

```
public class Tabs3 extends TabActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        final TabHost tabHost = getTabHost();
        tabHost.addTab(tabHost.newTabSpec("tab1")
                .setIndicator("list")
                .setContent(new Intent(this, List1.class)));

        tabHost.addTab(tabHost.newTabSpec("tab2")
                .setIndicator("photo list")
                .setContent(new Intent(this, List8.class)));

        tabHost.addTab(tabHost.newTabSpec("tab3")
                .setIndicator("destroy")
                .setContent(new Intent(this, Controls2.class)
                        .addFlags(Intent.FLAG ACTIVITY CLEAR TOP)));
    }
}
```

这里使用的 List1.class、List8.class 和 Controls2.class 都是已经实现的活动类，在这里被直接引用，出现在一个屏幕的几个 Tab 中。

TabActivity 是一个 Activity 的继承者，它主要包含以下几个方法：

```
TabHost  getTabHost()
   //  返回这个活动的 TabHost
TabWidget  getTabWidget()
   //  返回这个活动的 TabWidget the activity is using to draw the actual tabs.
void  onContentChanged()
   // 当内容变化的时候，更新屏幕的状态
```

TabHost 表示了 Tab 的框架，TabWidget 而表示了其中包含的内容，这 2 个类的继承关系如下所示：

=> android.view.View
  => android.view.ViewGroup
    => android.widget.FrameLayout
      => android.widget.TabHost
=> android.view.View
  => android.view.ViewGroup
    => android.widget.LinearLayout
      => android.widget.TabWidget

在本例中 newTabSpec()函数返回 TabHost.TabSpec 在其中可以设置每个 Tab 的内容。其中的 setContent(Intent intent)函数表示通过设置一个 Intent 启动一个活动。

TAB 其实包含了两方面的一个是上面的指示 indicator（包含了字串标签和图标两

方面的内容），另一个方面是 Tab 中的内容，在设置内容的时候，可以用三种选择：

1. 使用 View 的 id
2. 使用 TabHost.TabContentFactory
3. 使用 Intent 启动一个活动

Tab 的另外一种方式是使用 TabHost.TabContentFactory 类。

图 Tab（Content By Factory）程序的运行结果

参考示例程序：Content By Intent（Views=>Tabs=>Content By Factory）

源代码：com/example/android/apis/view/Tab2.java

```
    public class Tabs2 extends TabActivity implements TabHost.TabContentFactory
{

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            final TabHost tabHost = getTabHost();
            tabHost.addTab(tabHost.newTabSpec("tab1")
                    .setIndicator("tab1",

getResources().getDrawable(R.drawable.star_big_on))
                    .setContent(this));
            tabHost.addTab(tabHost.newTabSpec("tab2")
```

```
                .setIndicator("tab2")
                .setContent(this));
        tabHost.addTab(tabHost.newTabSpec("tab3")
                .setIndicator("tab3")
                .setContent(this));
    }

    public View createTabContent(String tag) {
        final TextView tv = new TextView(this);
        tv.setText("Content for tab with tag " + tag);
        return tv;
    }
```

createTabContent 是 TabHost.TabContentFactory 接口中的函数，实现这个函数来完成 Tabs 中的 View。

参考示例程序：Content By Id（Views=>Tabs=>Content By Id）

源代码：com/example/android/apis/view/Tab1.java

图 Tab（Content By ID）程序的运行结果

Tab1.java 的内容如下所示：

```
public class Tabs1 extends TabActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TabHost tabHost = getTabHost();
```

```
        LayoutInflater.from(this).inflate(R.layout.tabs1,
                                          tabHost.getTabContentView(), true);
        tabHost.addTab(tabHost.newTabSpec("tab1")
                .setIndicator("tab1")
                .setContent(R.id.view1));
        tabHost.addTab(tabHost.newTabSpec("tab3")
                .setIndicator("tab2")
                .setContent(R.id.view2));
        tabHost.addTab(tabHost.newTabSpec("tab3")
                .setIndicator("tab3")
                .setContent(R.id.view3));
    }
}
```

调用的 R.layout.tabs1 是资源文件 tab1.xml，其内容如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/view1"
        android:background="@drawable/blue"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/tabs_1_tab_1"/>
    <TextView android:id="@+id/view2"
        android:background="@drawable/red"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/tabs_1_tab_2"/>
    <TextView android:id="@+id/view3"
        android:background="@drawable/green"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/tabs_1_tab_3"/>
</FrameLayout>
```

在这里例子中，布局文件不是直接被设置到其中的。

第 9 章 2D 图形接口的使用

在 GUI 系统中，图形 API 是比较底层的接口。Android 系统的图形 API 包括 2D 和 3D 两部分：2D 部分使用 android.graphics 类，也作为上层控件的构建基础；3D 部分使用 OpenGL 作为标准接口。

9.1 使用 2D 图形接口的程序结构。

2D 图形的接口实际上是 Android 图形系统的基础， GUI 上的各种可见元素也是基于 2D 图形接口构建的。因此，Android GUI 方面的内容分为两层，下层是图形的 API，上层是各种控件，各种控件实际上是基于图形 API 绘制出来的。

使用 2D 图形接口的结构如下图所示：

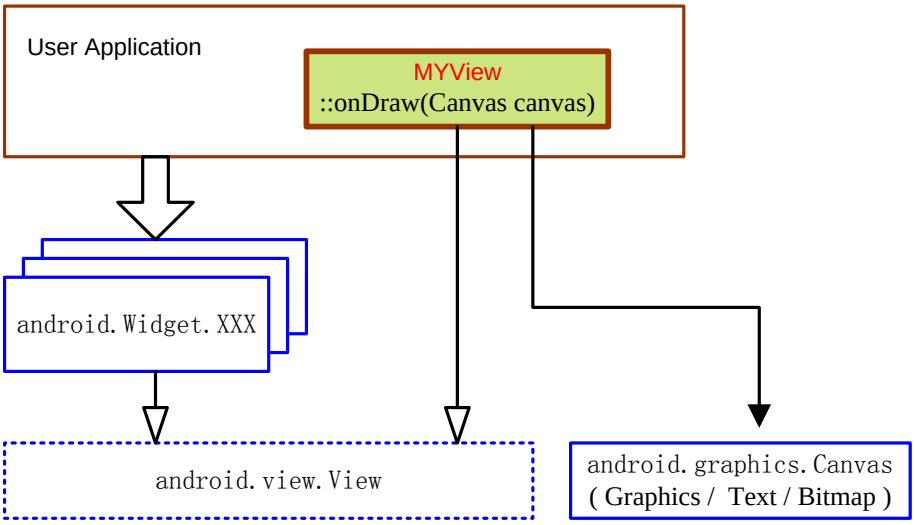


图 Android 2D 绘图接口结构

通过继承 android.view.View 类，并实现其中的 onDraw()函数来实现绘制的工作，绘制的工作主要由 android.graphics 包来实现。android.graphics 包中的内容是 Android 系统的 2D 图形 API，其中主要类的内容包含以下一些内容：

- Point、Rect 和 Color 等：一些基础类，分别定义顶点、矩阵、颜色的基础信息元素；
- Bitmap：表示内存中的位图，可以从图像文件中建立，可以指定依靠颜色来建立，也可以控制其中的每一个像素；

- Paint：画笔，用于控制绘制的样式（style）和颜色（color）等信息；
- Canvas：画布，2D 图形系统最核心的一个类，处理 onDraw()调用

主要绘制的设置和操作在 Paint（画笔）和 Canvas（画布）2 个类当中，使用这两个类就可以完成所有的绘制。

Canvas 类包含了一系列用于绘制的方法，方法分为 3 种类型：

- 几何图形
- 文本
- 位图

Canvas 类的几何图形（Geometry）方面的方法用于绘制点、绘制线、绘制矩形、绘制圆弧等。其中一些主要的方法如下所示：

```
    void drawARGB(int a, int r, int g, int b)                  // 将整体填充为某种颜色
    void drawPoints(float[] pts, Paint paint)                  // 绘制一个点
    void drawLines(float[] pts, Paint paint)                   // 绘制一条线
    void drawRect(RectF rect, Paint paint)                     // 绘制矩形
    void drawCircle(float cx, float cy, float radius, Paint paint)    // 绘制圆形
    void drawArc(RectF oval, float startAngle, float sweepAngle,      // 绘制圆弧
                boolean useCenter, Paint paint)
```

Canvas 类的文本（Text）方面的方法用于直接绘制文本内容，文本通常用一个字符串来表示。其中一些主要的方法如下所示：

```
    void drawText(String text, int start, int end, float x, float y, Paint paint)
    void drawText(char[] text, int index, int count, float x, float y, Paint paint)
    void drawText(String text, float x, float y, Paint paint)
    void drawText(CharSequence text, int start, int end, float x, float y, Paint paint)
```

Canvas 类的位图（Bitmap）方面的方法用于直接绘制位图，位图通常用一个 Bitmap 类来表示。其中一些主要的方法如下所示：

```
    void drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint) // 指定 Matrix 绘制位图
    void drawBitmap(int[] colors, int offset, int stride, // 指定数组作为 Bitmap 绘制
                    float x, float y, int width, int height,
                    boolean hasAlpha, Paint paint)
    void drawBitmap(Bitmap bitmap, Rect src, RectF dst, Paint paint)// 自动缩放到目标矩形的绘制
```

Canvas 是 Android 的 2D 图形绘制的中枢，绘制方法的参数中通常包含一个 Paint 类型，它作为附加绘制的信息来使用。

在使用 2D 的图形 API 方面，步骤通常如下所示：

1、扩展实现 android.view.View 类。

2、实现 View 的 OnDraw()函数，在其中使用 Canvas 的方法进行绘制。

使用 2D 的图形 API 的场合，自定义实现的 View 类型作为下层的绘制和上层的 GUI 系统中间层。

android.graphics.drawable 包是 Android 中一个绘制相关的包，表示一些可以被绘制的东西。在 Android 中 Drawable 的含义就是可以仅仅是为了显示来使用的，与 View 的主要区别就在于 Drawable 不能从用户处获得事件的反馈。

事实上，使用 Android 的 2D API 的程序结构和实现一个自定义控件类似，但是它们的目的略有不同：使用 2D API 主要是为了实现自由的绘制；自定义控件的目的是在应用程序中使用这些控件，包括可以在布局文件中使用甚至使用其属性。

9.2 图像、图形、文本的基本绘制

Android 中基本的绘制包括了图像、图形和文本的绘制。

参考示例程序：ApiDemo 的 AlphaBitmap（ApiDemo=>Graphics=>AlphaBitmap）

源代码：android/apis/graphics/AlphaBitmap.java

AlphaBitmap 程序的运行结果如图所示：

图 AlphaBitmap 程序的运行结果

本程序在界面上自上而下一共绘制了 3 个内容，第一个是一个原始位图，第二个是经过变化的位图，第三个是几何图形。

在这个示例程序中，主要通过将一个自定义的 SampleView 设置成活动的 View 作为其中的 ContentView。onCreate()函数如下所示：

```
    public class AlphaBitmap extends GraphicsActivity { // GraphicsActivity 相当于 Activity
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(new SampleView(this));// 设置实现中的 SampleView
        }
    }
```

SampleView 是其中扩展了 View 的实现，主要的内容在类的构造函数和 OnDraw()函数中，内容如下所示：

```
    private static class SampleView extends View {
            private Bitmap mBitmap;
            private Bitmap mBitmap2;
            private Bitmap mBitmap3;
            private Shader mShader;
        public SampleView(Context context) {
```

```
            super(context);
            setFocusable(true);
            InputStream is = context.getResources().
                                openRawResource(R.drawable.app_sample_code);
            mBitmap = BitmapFactory.decodeStream(is);     // 解码位图文件到 Bitmap
            mBitmap2 = mBitmap.extractAlpha();            // 提取位图的透明通道
            // 创建一个位图
            mBitmap3 = Bitmap.createBitmap(200, 200, Bitmap.Config.ALPHA_8);
            drawIntoBitmap(mBitmap3);                 //  调 用 自 己 实 现 的
drawIntoBitmap()
            mShader = new LinearGradient(0, 0, 100, 70, new int[] {
                                        Color.RED, Color.GREEN, Color.BLUE },
                                        null, Shader.TileMode.MIRROR);
        }
        private static void drawIntoBitmap(Bitmap bm) {
            float x = bm.getWidth();
            float y = bm.getHeight();
            Canvas c = new Canvas(bm);
            Paint p = new Paint();
            p.setAntiAlias(true);

            p.setAlpha(0x80);
            c.drawCircle(x/2, y/2, x/2, p);

            p.setAlpha(0x30);
            p.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.SRC));
            p.setTextSize(60);
            p.setTextAlign(Paint.Align.CENTER);
            Paint.FontMetrics fm = p.getFontMetrics();
            c.drawText("Alpha", x/2, (y-fm.ascent)/2, p);
        }
        @Override protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            Paint p = new Paint();
            float y = 10;                               // 设置纵坐标
            p.setColor(Color.RED);                      // 设置画笔为红色
            canvas.drawBitmap(mBitmap, 10, y, p);       // 绘制第 1 个位图（原始图像）
            y += mBitmap.getHeight() + 10;              // 纵坐标增加
            canvas.drawBitmap(mBitmap2, 10, y, p);      // 绘制第 2 个位图（根据红色的画
笔）
            y += mBitmap2.getHeight() + 10;             // 纵坐标增加
            p.setShader(mShader);                           // 设置阴影
            canvas.drawBitmap(mBitmap3, 10, y, p);      // 绘制第 3 个位图
        }
    }
```

第 1 个图是直接对原始的图像进行了绘制；第 2 个图是在原始图像的基础上抽取了透明通道，所以绘制时画笔（Paint）的颜色起到了作用；第 3 个图是调用 drawIntoBitmap()绘制了一个具有渐变颜色的圆，并附加了文字。

9.3 文本的对齐方式

在 Android 中文本的绘制可以使用一些效果，其中比较智能的方面是可以让文本的对齐操作。对齐操作不仅有水平和竖直上的对齐问题，甚至可以让文本在曲线的路径上实现对齐。

参考示例程序：ApiDemo 的 TextAlign（ApiDemo=>Graphics=>TextAlign）

源代码：android/apis/graphics/TextAlign.java

TextAlign 程序的运行结果如图所示：

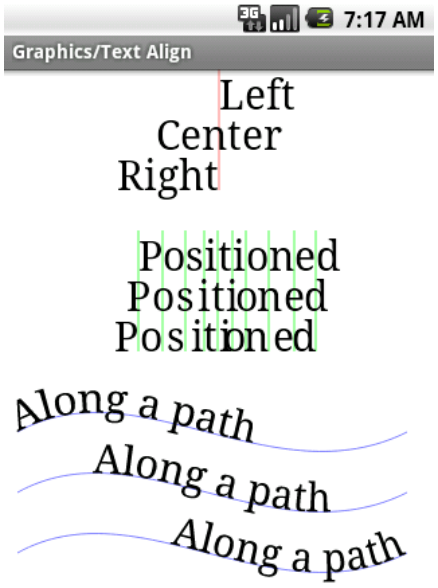


图 PathEffects 程序的运行结果

```
    private static class SampleView extends View {
        private Paint  mPaint;
        private float  mX;
        private float[] mPos;

        private Path   mPath;
        private Paint  mPathPaint;

        private static final int DY = 30;
        private static final String TEXT_L = "Left";
        private static final String TEXT_C = "Center";
        private static final String TEXT_R = "Right";
```

```
        private static final String POSTEXT = "Positioned";
        private static final String TEXTONPATH = "Along a path";

        private static void makePath(Path p) {
            p.moveTo(10, 0);
            p.cubicTo(100, -50, 200, 50, 300, 0);
        }
        private float[] buildTextPositions(String text, float y, Paint paint)
{
            // 省略，计算位置信息等内容
            return pos;
        }

        public SampleView(Context context) {
            super(context);
            setFocusable(true);
            mPaint = new Paint();
            mPaint.setAntiAlias(true);
            mPaint.setTextSize(30);
            mPaint.setTypeface(Typeface.SERIF);

            mPos = buildTextPositions(POSTEXT, 0, mPaint);
            mPath = new Path();
            makePath(mPath);         // 建立路径

            mPathPaint = new Paint();
            mPathPaint.setAntiAlias(true);
            mPathPaint.setColor(0x800000FF);
            mPathPaint.setStyle(Paint.Style.STROKE);
        }
        @Override protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            Paint p = mPaint;                        // 获得画笔
            float x = mX;
            float y = 0;
            float[] pos = mPos;
        // 绘制正常的字串
            p.setColor(0x80FF0000);                 // 绘制一条线
            canvas.drawLine(x, y, x, y+DY*3, p);
            p.setColor(Color.BLACK);
            canvas.translate(0, DY);
            p.setTextAlign(Paint.Align.LEFT);      // 绘制左对齐的文本
            canvas.drawText(TEXT_L, x, y, p);
            canvas.translate(0, DY);
            p.setTextAlign(Paint.Align.CENTER);    // 绘制中对齐的文本
            canvas.drawText(TEXT_C, x, y, p);
            canvas.translate(0, DY);
            p.setTextAlign(Paint.Align.RIGHT);     // 绘制右对齐的文本
            canvas.drawText(TEXT_R, x, y, p);

        // 绘制根据位置的的字串
            canvas.translate(100, DY*2);
```

```
            p.setColor(0xBB00FF00);
            for (int i = 0; i < pos.length/2; i++) {
                canvas.drawLine(pos[i*2+0], pos[i*2+1]-DY,
                        pos[i*2+0], pos[i*2+1]+DY*2, p);
            }                                       // 绘制若干条线
            p.setColor(Color.BLACK);
            p.setTextAlign(Paint.Align.LEFT);
            canvas.drawPosText(POSTEXT, pos, p); // 绘制左对齐的文本
            canvas.translate(0, DY);
            p.setTextAlign(Paint.Align.CENTER);
            canvas.drawPosText(POSTEXT, pos, p);  // 绘制中对齐的文本
            canvas.translate(0, DY);
            p.setTextAlign(Paint.Align.RIGHT);
            canvas.drawPosText(POSTEXT, pos, p);  // 绘制右对齐的文本

        // 绘制在路径上的的字串
            canvas.translate(-100, DY*2);       // 重定画布的位置
            canvas.drawPath(mPath, mPathPaint);
            p.setTextAlign(Paint.Align.LEFT);
            canvas.drawTextOnPath(TEXTONPATH, mPath, 0, 0, p);  // 绘制对齐路径
的文本
            canvas.translate(0, DY*1.5f);
            canvas.drawPath(mPath, mPathPaint);
            p.setTextAlign(Paint.Align.CENTER);
            canvas.drawTextOnPath(TEXTONPATH, mPath, 0, 0, p);  // 绘制对齐路径
的文本
            canvas.translate(0, DY*1.5f);
            canvas.drawPath(mPath, mPathPaint);
            p.setTextAlign(Paint.Align.RIGHT);
            canvas.drawTextOnPath(TEXTONPATH, mPath, 0, 0, p);  // 绘制对齐路径
的文本
        }
        // 省略部分内容
    }
```

文本的对其操作主要通过以下两点来完成：

1．通过画笔（Paint）的 setTextAlign()函数设置绘制过程中的对齐方式。

2．drawText()，drawPosText()，drawTextOnPath()几个函数表示了文本的几种绘制方式。drawText()在指定的坐标上进行文本绘制；drawPosText()在一个表示为位置信息的数组上进行文本绘制（其中的 float[] pos 参数表示交替的 x 和 y 表示的坐标）；drawTextOnPath()表示在一个路径（Path）进行文本绘制。

9.4 使用路径效果（PathEffect）

路径表示一条曲线，在 Android 中通过路径可以更灵活地实现一些效果。

参考示例程序：ApiDemo 的 PathEffects（ApiDemo=>Graphics=>PathEffects）
源代码：android/apis/graphics/PathEffects.java
PathEffects 程序的运行结果如图所示：

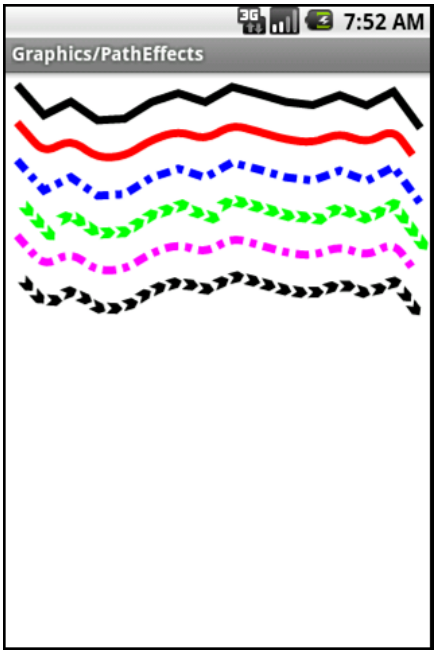


图 PathEffects 程序的运行结果

图中的几个路径的曲线的基本走向一致，但是细节的方面各不相同，例如线的方式

代码主要是实现了 SampleView，核心部分如下所示：

```
    private static class SampleView extends View {
        private Paint mPaint;
        private Path mPath;
        private PathEffect[] mEffects;
        private int[] mColors;
        private float mPhase;

        private static PathEffect makeDash(float phase) {
            return new DashPathEffect(new float[] { 15, 5, 8, 5 }, phase);
        }

        private static void makeEffects(PathEffect[] e, float phase) {
            e[0] = null;     // 没有效果
            e[1] = new CornerPathEffect(10);           // 拐角路径效果
            e[2] = new DashPathEffect(new float[] {10, 5, 5, 5}, phase);
```

```
            e[3] = new PathDashPathEffect(makePathDash(), 12, phase,
                                    PathDashPathEffect.Style.ROTATE);
                                                        // 破折号式效果
            e[4] = new ComposePathEffect(e[2], e[1]);  // 组合路径效果（内外各不
同）
            e[5] = new ComposePathEffect(e[3], e[1]);  // 组合路径效果
        }

        public SampleView(Context context) {
            super(context);
            setFocusable(true);
            setFocusableInTouchMode(true);
            mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
            mPaint.setStyle(Paint.Style.STROKE);
            mPaint.setStrokeWidth(6);
            mPath = makeFollowPath();
            mEffects = new PathEffect[6];               // 定义路径效果
            mColors = new int[] { Color.BLACK, Color.RED, Color.BLUE,
                              Color.GREEN, Color.MAGENTA, Color.BLACK
                            };
        }
        @Override protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            RectF bounds = new RectF();
            mPath.computeBounds(bounds, false);
            canvas.translate(10 - bounds.left, 10 - bounds.top);
            makeEffects(mEffects, mPhase);          // 创建几种效果
            mPhase += 1;
            invalidate();
            for (int i = 0; i < mEffects.length; i++) {
                mPaint.setPathEffect(mEffects[i]);  // 设置路径效果
                mPaint.setColor(mColors[i]);        // 使用不同的颜色
                canvas.drawPath(mPath, mPaint);     // 进行路径的绘制
                canvas.translate(0, 28);
            }
        }
        // 省略部分内容
    }
```

Path 是在 Android 中表示路径的类，路径可以理解成一条曲线，曲线可以由直线、圆弧等组成。Android 中提供了几种不同的绘制效果，PathEffect 是关于几何图形绘制的一个基类，它具有几个继承者：ComposePathEffect, CornerPathEffect, DashPathEffect, DiscretePathEffect, PathDashPathEffect, SumPathEffect。

9.5 剪裁效果

Android 中当几个绘制的内容重叠的时候，可以使用剪裁效果进行控制在重叠的情况下，显示哪个部分的内容。

参考示例程序：ApiDemo 的 Clipping（ApiDemo=>Graphics=>Clipping）

源代码：android/apis/graphics/Clipping.java

Clipping 程序的运行结果如图所示：

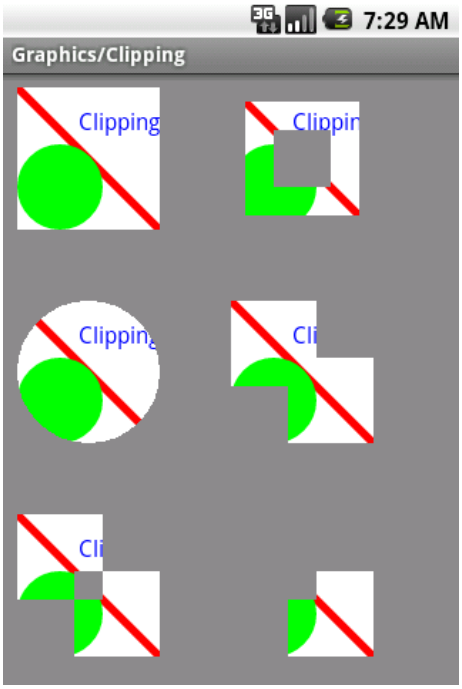


图 Clipping 程序的运行结果

图中的 6 个绘制效果各不相同，每个部分都是在一个白色矩形区域中，绘制一个条红线、一个绿色的园和一个蓝色的文本组成。

```
    private static class SampleView extends View {
        private Paint mPaint;
        private Path mPath;

        public SampleView(Context context) {
            super(context);
            setFocusable(true);
            mPaint = new Paint();
            mPaint.setAntiAlias(true);
```

```
            mPaint.setStrokeWidth(6);
            mPaint.setTextSize(16);
            mPaint.setTextAlign(Paint.Align.RIGHT);

            mPath = new Path();
        }

        private void drawScene(Canvas canvas) {
            canvas.clipRect(0, 0, 100, 100);        // 选择区域

            canvas.drawColor(Color.WHITE);
            mPaint.setColor(Color.RED);       // 红色直线
            canvas.drawLine(0, 0, 100, 100, mPaint);
            mPaint.setColor(Color.GREEN);     // 绿色的圆
            canvas.drawCircle(30, 70, 30, mPaint);
            mPaint.setColor(Color.BLUE);       // 蓝色文本
            canvas.drawText("Clipping", 100, 30, mPaint);
        }

        @Override protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.GRAY);

            canvas.save();                // 左上部分绘制
            canvas.translate(10, 10);
            drawScene(canvas);
            canvas.restore();
            canvas.save();                // 右上部分绘制
            canvas.translate(160, 10);
            canvas.clipRect(10, 10, 90, 90);
            canvas.clipRect(30, 30, 70, 70, Region.Op.DIFFERENCE); // 中间矩形
被去除
            drawScene(canvas);
            canvas.restore();
            canvas.save();                // 左中部分绘制
            canvas.translate(10, 160);
            mPath.reset();
            canvas.clipPath(mPath); // 做一个圆
            mPath.addCircle(50, 50, 50, Path.Direction.CCW);
            canvas.clipPath(mPath, Region.Op.REPLACE); // 圆外部分被去除
            drawScene(canvas);
            canvas.restore();
            canvas.save();                // 右中部分绘制
            canvas.translate(160, 160);
            canvas.clipRect(0, 0, 60, 60);
            canvas.clipRect(40, 40, 100, 100, Region.Op.UNION);
            drawScene(canvas);
            canvas.restore();
            canvas.save();                // 左下部分绘制
            canvas.translate(10, 310);
            canvas.clipRect(0, 0, 60, 60);
            canvas.clipRect(40, 40, 100, 100, Region.Op.XOR);
```

```
            drawScene(canvas);
            canvas.restore();
            canvas.save();                  // 右下部分绘制
            canvas.translate(160, 310);
            canvas.clipRect(0, 0, 60, 60);
            canvas.clipRect(40, 40, 100, 100, Region.Op.REVERSE_DIFFERENCE);
            drawScene(canvas);
            canvas.restore();
        }
    }
```

Region.Op 表示图层的混合方法，包括以下的几个枚举值：

- DIFFERENCE（差）
- INTERSECT（加入）
- REPLACE（替代）
- REVERSE_DIFFERENCE（保留差异）
- UNION（和）
- XOR（异或）

clipPath()和 clipRect ()等几个函数用于在画布的范围内将几个区域剪裁掉。

```
    boolean  clipPath(Path path)
    boolean  clipPath(Path path, Region.Op op)
    boolean  clipRect(Rect rect)
    boolean  clipRect(float left, float top, float right, float bottom)
    boolean  clipRect(float left, float top, float right, float bottom, Region.Op
op)
```

剪裁的功能可以丰富绘制的最终效果。

9.6 记录绘制的过程

在图形界面的绘制过程中，绘制是一个分阶段的复杂工作，如果可以将某一次绘制的过程纪录，就可以在其他的地方重现这个绘制。

参考示例程序：ApiDemo 的 Pictures（ApiDemo=>Graphics=>Pictures）

源代码：android/apis/graphics/ Pictures.java

Pictures 程序的运行结果如图所示：

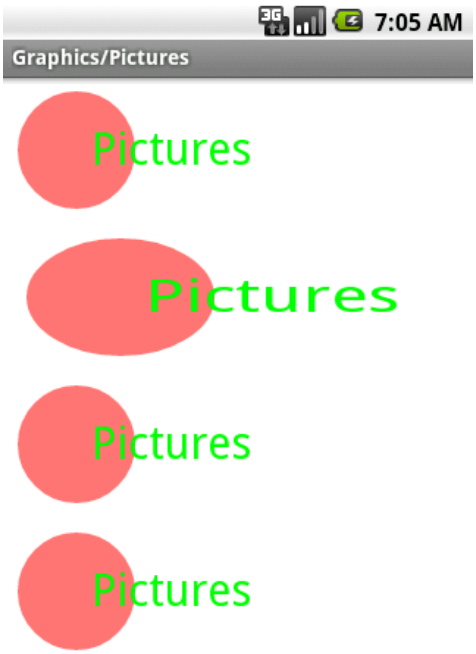


图 记录和重现绘制效果的运行结果

本例子的核心部分如下所示：

```
    private static class SampleView extends View {
        private Picture mPicture;
        private Drawable mDrawable;
        static void drawSomething(Canvas canvas) {    // 绘制的过程
            Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
            p.setColor(0x88FF0000);
            canvas.drawCircle(50, 50, 40, p);
            p.setColor(Color.GREEN);
            p.setTextSize(30);
            canvas.drawText("Pictures", 60, 60, p);
        }

        public SampleView(Context context) {
            super(context);
            setFocusable(true);
            setFocusableInTouchMode(true);

            mPicture = new Picture();
            drawSomething(mPicture.beginRecording(200, 100));  // 开始记录绘制
的过程
            mPicture.endRecording();                           // 结束记录绘制
的过程
```

```
            mDrawable = new PictureDrawable(mPicture);        // 创建一个可绘制物
        }

        @Override protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            canvas.drawPicture(mPicture);    // 绘制第 1 次
            canvas.drawPicture(mPicture, new RectF(0, 100, getWidth(), 200));
                                             // 绘制第 2 次
            mDrawable.setBounds(0, 200, getWidth(), 300);
            mDrawable.draw(canvas);           // 绘制第 3 次

            ByteArrayOutputStream os = new ByteArrayOutputStream();
            mPicture.writeToStream(os);
            InputStream is = new ByteArrayInputStream(os.toByteArray());
            canvas.translate(0, 300);
            canvas.drawPicture(Picture.createFromStream(is));   // 绘制第 4 次
        }
    }
```

本例子进行的 4 次绘制情况为：先在一个 Picture 中进行绘制，第 1 次和第 2 次使用这个纪录的 Picture 进行绘制，第 3 次使用由 Picture 转换成的 PictureDrawable 进行绘制，第 4 次将 Picture 转成一个通用的流，然后在转回成 Picture 进行绘制。

Pictures 是一个可以记录绘制过程的类，通常情况下 Android 的绘制工作需要被绘制到画布（Cavans）上，但是能显示的画布只有一个。Pictures 的功能就在于可以让绘制的东西绘制到一个虚拟的画布上，这个虚拟的画布由 Pictures 的 beginRecording()函数返回，picture.draw(canvas)和 canvas.drawPicture()函数用于将一个记录好的绘制过程，重现到画布上。Pictures 绘制的内容也可以被记录到一个流当中，然后在重新构造一个 Pictures。

9.7 动画效果

Android 中可以容易地实现绘制的动画效果。

参考示例程序：ApiDemo 的 AnimateDrawables（ApiDemo=>Graphics=>AnimateDrawables）

源代码：android/apis/graphics/ AnimateDrawables.java

AnimateDrawables 程序的运行结果如图所示：

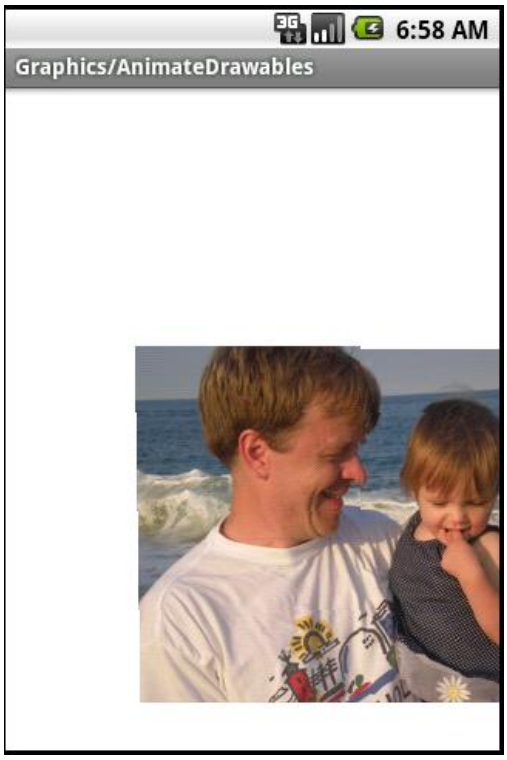


图 动画效果

核心的代码部分如下所示：

```
     private static class SampleView extends View {
          private AnimateDrawable mDrawable;
          public SampleView(Context context) {
              super(context);
              setFocusable(true);
              setFocusableInTouchMode(true);
              Drawable                              dr                              =
context.getResources().getDrawable(R.drawable.beach);
              dr.setBounds(0,            0,            dr.getIntrinsicWidth(),
dr.getIntrinsicHeight());

              Animation an = new TranslateAnimation(0, 100, 0, 200); // 创建一个
动画
              an.setDuration(2000);                          // 持续时间
              an.setRepeatCount(-1);
              an.initialize(10, 10, 10, 10);
              mDrawable = new AnimateDrawable(dr, an);    // 创建 Drawable
              an.startNow();
          }
```

```
        @Override protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            mDrawable.draw(canvas);                    // 通过 Drawable 进行绘制
            invalidate();
        }
    }
```

本例子中使用 TranslateAnimation 是使用了位置变化的动画效果。

android.view.animation 包中的 Animation 类表示了一个动画效果，它有几个继承者：TranslateAnimation（位置动画）、RotateAnimation（旋转动画）、ScaleAnimation（缩放动画）、AlphaAnimation（透明度动画）、AnimationSet（动画组）。

AnimationDrawable 是 Drawable 的一个继承者，其中包含了几个主要的方法：

```
Canvas  beginRecording(int width, int height)
void  endRecording()
void  draw(Canvas canvas)
void  writeToStream(OutputStream stream)
```

通过 AnimationDrawable，可以将 Animation 类转化成 Drawable，然后实现直接绘制的工作。

第 10 章 OpenGL 3D 图形的使用

10.1 使用 OpenGL 图形接口的程序结构。

在 Android 中，可以直接支持 3D 图形的绘制，主要使用 OpenGL 标准的类 javax.microedition.khronos.egl，但是需要结合 Android GUI 系统使用。Android 中 OpenGL 接口使用的结构如图所示：

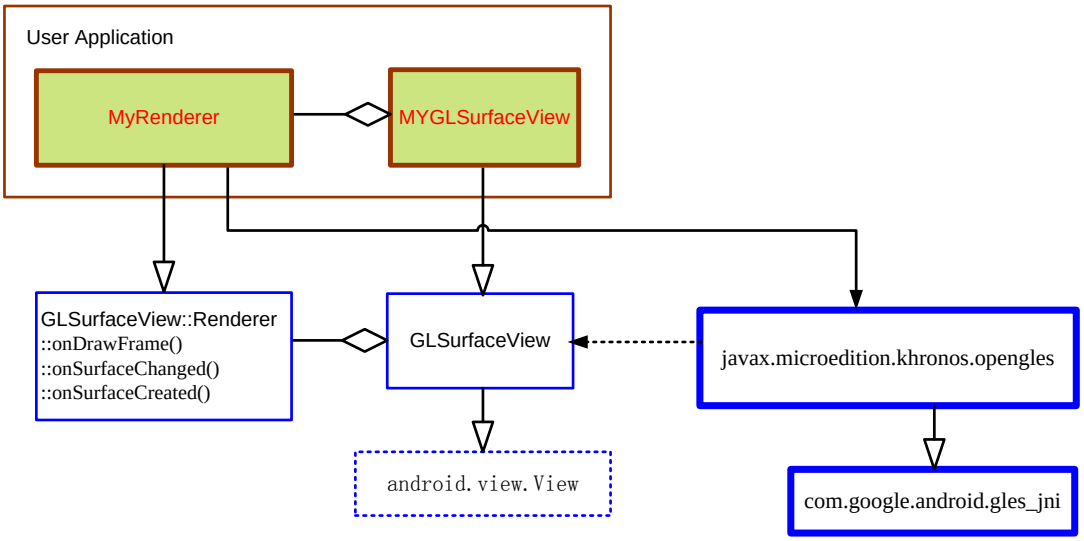


图 Android OpenGL 绘图接口结构

在使用 3D 的图形 API 方面，主要的步骤通常如下所示：

1．扩展实现 android.view.GLSurfaceView 类。

2．扩展实现 android.opengl.GLSurfaceView 中的 Renderer（渲染器）。

3．实现 GLSurfaceView::Renderer 中的 onDrawFrame()等函数。

android.opengl.GLSurfaceView 扩展了 android.view.SurfaceView，android.view.SurfaceView 扩展了 android.view.View，因此 GLSurfaceView 本身可以作为 android. view.View 来使用。

GLSurfaceView::Renderer 是一个接口，其中主要定义了以下几个方法：

```
    abstract void  onDrawFrame(GL10 gl)                          // 绘制当前帧
    abstract void  onSurfaceChanged(GL10 gl, int width, int height)
                                                                 // Surface 变化
时调用
    abstract void  onSurfaceCreated(GL10 gl, EGLConfig config)
```

```
                                                    // Surface 创建
时调用
```

各个方法的参数 GL10 是 javax.microedition.khronos.egl 包中的通用函数。GLSurface View::Renderer 中的 onSurfaceChanged()和 onSurfaceCreated()方法实际上是和 SurfaceView 中的两个方法对应的。实现的 GLSurfaceView::Renderer，通过 GLSurfaceView 的 setRenderer()方法将其设置到 GLSurfaceView 中。

在 ApiDemo 的示例程序中，android/apis/graphics/ 中的 GLSurfaceViewActivity、TouchRotateActivity、TriangleActivity 等程序和 spritetext/及/Kube/目录中的程序是 OpenGL 的示例程序。

10.2 基本的绘制

参考示例程序：Touch Rotate（Graphics=>OpenGL ES=>Touch Rotate）
源代码：android/apis/graphics/TouchRotateActivity.java
Touch Rotate 程序的运行结果如图所示：

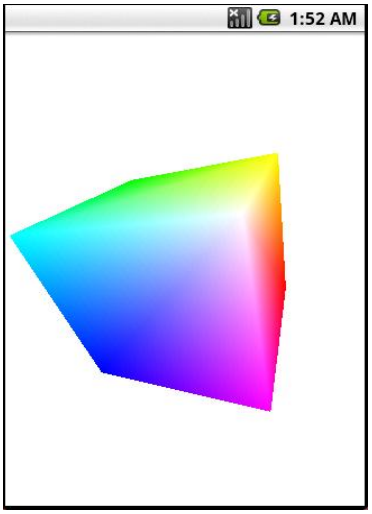


图 Touch Rotate 程序的运行结果

本程序显示了一个可以旋转的立方体，TouchRotate Activity 类的结构如下所示：

```
    public class TouchRotateActivity extends Activity {
        @Override
```

```
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            mGLSurfaceView = new TouchSurfaceView(this);  // 建立 GLSurfaceView
            setContentView(mGLSurfaceView);               // 设置 View 到活动中
            mGLSurfaceView.requestFocus();                    //       配       置
GLSurfaceView
            mGLSurfaceView.setFocusableInTouchMode(true);
        }
        //......Resume()和 onPause()方法调用 GLSurfaceView 的对应方法
        private GLSurfaceView mGLSurfaceView;             // 配置 GLSurfaceView
    }
```

TouchSurfaceView 是一个扩展 GLSurfaceView 类的实现，其中的 CubeRenderer 是扩展了 GLSurfaceView::Renderer 接口的实现，其主要内容如下所示：

```
    class TouchSurfaceView extends GLSurfaceView {
        public TouchSurfaceView(Context context) {
            super(context);
            mRenderer = new CubeRenderer();               // 建立渲染器
            setRenderer(mRenderer);                       // 设置渲染器
            setRenderMode(GLSurfaceView.RENDERMODE WHEN DIRTY);
        }
        private class CubeRenderer implements GLSurfaceView.Renderer {
                                                              // 实现渲染器接口
            public void onDrawFrame(GL10 gl) {
                // 调用 OpenGL 的标准接口进行操作
                gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
                gl.glMatrixMode(GL10.GL_MODELVIEW);
                gl.glLoadIdentity();
                gl.glTranslatef(0, 0, -3.0f);
                gl.glRotatef(mAngleX, 0, 1, 0);           // 对绘制的图形进行旋转
                gl.glRotatef(mAngleY, 1, 0, 0);
                gl.glEnableClientState(GL10.GL VERTEX ARRAY);
                gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
                mCube.draw(gl);                           // 调用 draw()进行绘制
            }
        }
    }
```

CubeRenderer 渲染器中的 onSurfaceChanged()和 onSurfaceCreated()两个函数进行了 Surface 变化及创建时的操作。

```
            public void onSurfaceChanged(GL10 gl, int width, int height) {
                 gl.glViewport(0, 0, width, height);
                 float ratio = (float) width / height;
                 gl.glMatrixMode(GL10.GL PROJECTION);
                 gl.glLoadIdentity();
                 gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
            }
```

```
        public void onSurfaceCreated(GL10 gl, EGLConfig config)
            gl.glDisable(GL10.GL DITHER);
            gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
                   GL10.GL_FASTEST);
            gl.glClearColor(1,1,1,1);
            gl.glEnable(GL10.GL_CULL_FACE);
            gl.glShadeModel(GL10.GL_SMOOTH);
            gl.glEnable(GL10.GL DEPTH TEST);
        }
```

移动的效果：

```
    @Override public boolean onTrackballEvent(MotionEvent e) {
        mRenderer.mAngleX += e.getX() * TRACKBALL_SCALE_FACTOR;
        mRenderer.mAngleY += e.getY() * TRACKBALL_SCALE_FACTOR;
        requestRender();
        return true;
    }

    @Override public boolean onTouchEvent(MotionEvent e) {
        float x = e.getX();
        float y = e.getY();
        switch (e.getAction()) {
        case MotionEvent.ACTION_MOVE:
            float dx = x - mPreviousX;
            float dy = y - mPreviousY;
            mRenderer.mAngleX += dx * TOUCH_SCALE_FACTOR;
            mRenderer.mAngleY += dy * TOUCH SCALE FACTOR;
            requestRender();
        }
        mPreviousX = x;
        mPreviousY = y;
        return true;
    }
```

10.3 渲染器的实现

在 Android 中，也可以不扩展 GLSurfaceView 类，只是实现 GLSurfaceView::Renderer 接口。实现渲染器可以在 GLSurfaceView 中直接使用。

参考示例程序：

GLSurfaceViewActivity（Graphics=>OpenGL ES=>GLSurfaceViewActivity）

TranslucentGLSurfaceViewActivity

（Graphics=>OpenGL ES=>TranslucentGLSurfaceViewActivity）

源代码：

src/com/example/android/apis/graphics/GLSurfaceViewActivity.java

src/com/example/android/apis/graphics/TranslucentGLSurfaceViewActivity.java
src/com/example/android/apis/graphics/CubeRenderer.java
src/com/example/android/apis/graphics/Cube.java

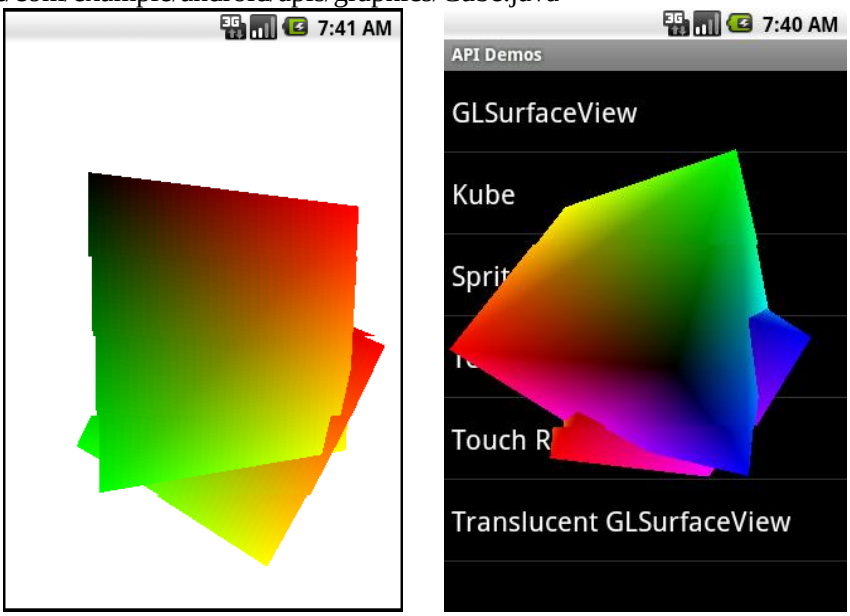


图 立方体和透明的立方体

Cube 实现的是一个使用 OpenGL 绘制的立方体，CubeRenderer 表示基于立方体实现的渲染器，GLSurfaceViewActivity 和 TranslucentGLSurfaceViewActivity 分别是图中左右两个效果的界面。

CubeRenderer.java 的内容：

```
class CubeRenderer implements GLSurfaceView.Renderer {
    public CubeRenderer(boolean useTranslucentBackground) {
        mTranslucentBackground = useTranslucentBackground;
        mCube = new Cube();
    }
    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0, 0, -3.0f);
        gl.glRotatef(mAngle,        0, 1, 0);
        gl.glRotatef(mAngle*0.25f,  1, 0, 0);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        mCube.draw(gl);
        gl.glRotatef(mAngle*2.0f, 0, 1, 1);
```

```
            gl.glTranslatef(0.5f, 0.5f, 0.5f);
            mCube.draw(gl);
            mAngle += 1.2f;
        }
    // 省略部分内容
        private boolean mTranslucentBackground;
        private Cube mCube;
        private float mAngle;
    }
```

AndroidManifest.xml 中两个活动的内容：

```
            <activity android:name=".graphics.GLSurfaceViewActivity"
                    android:label="Graphics/OpenGL ES/GLSurfaceView"
                    android:theme="@android:style/Theme.NoTitleBar"
                    android:configChanges="orientation|keyboardHidden">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category  android:name="android.intent.category.SAMPLE_CODE"
/>
                </intent-filter>
            </activity>
            <activity android:name=".graphics.TranslucentGLSurfaceViewActivity"
                    android:label="Graphics/OpenGL ES/Translucent GLSurfaceView"
                    android:theme="@style/Theme.Translucent"
                    android:configChanges="orientation|keyboardHidden">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category  android:name="android.intent.category.SAMPLE_CODE"
/>
                </intent-filter>
            </activity>
```

2 个程序主要的区别是 TranslucentGLSurfaceViewActivity 通过 android:theme 将背景颜色设置成了 Theme.Translucent，表示透明的背景。

GLSurfaceViewActivity.java 的内容如下所示：

```
    public class GLSurfaceViewActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            mGLSurfaceView = new GLSurfaceView(this);
            mGLSurfaceView.setRenderer(new CubeRenderer(false));
            setContentView(mGLSurfaceView);
        }
    }
```

TranslucentGLSurfaceViewActivity.java 的内容：

```
    public class TranslucentGLSurfaceViewActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            mGLSurfaceView = new GLSurfaceView(this);
```

```
            // 使用 RGB8888 的像素格式
            mGLSurfaceView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
            // 定义下层为透明
            mGLSurfaceView.setRenderer(new CubeRenderer(true));
            // 设置 Surface 的 Alpha 通道
            mGLSurfaceView.getHolder().setFormat(PixelFormat.TRANSLUCENT);
            // 设置 Surface 透明
            setContentView(mGLSurfaceView);
        }
    }
```

以上的 2 个程序中，重了后者设置了透明效果外基本相同，在这 2 个实现中，不需要重新实现 GLSurfaceView，而只是实现 GLSurfaceView 中::Renderer 即可，

10.4 3D 动画效果的实现

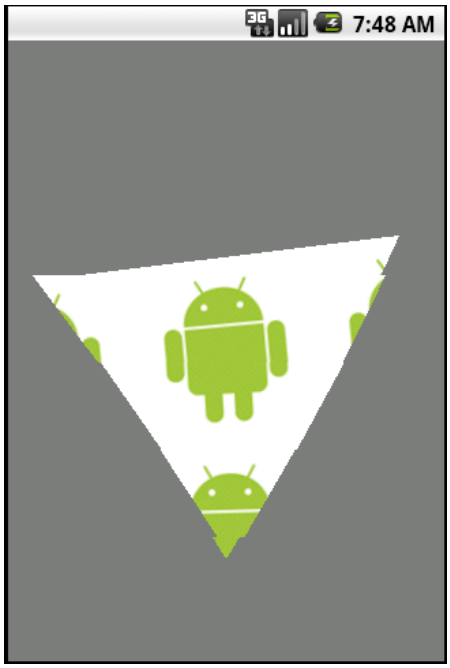
OpenGL 本身可以强大的实现动画的效果，在 Android 中实现动画，实际上只是将 OpenGL 动画需要使用的元素在 Android 中重新实现。

参考示例程序：Touch Rotate（Graphics=>OpenGL ES=>Textured Triangle）

源代码：src/com/example/android/apis/graphics/TriangleActivity.java

src/com/example/android/apis/graphics/TriangleRenderer.java

Touch Rotate 程序的运行结果如图所示：



TriangleActivity.java 的内容如下所示：

```
public class TriangleActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mGLView = new GLSurfaceView(this);
        mGLView.setEGLConfigChooser(false);
        mGLView.setRenderer(new TriangleRenderer(this));
        setContentView(mGLView);
    }
    @Override
    protected void onPause() {    // 使其中的 View 停止
        super.onPause();
        mGLView.onPause();
    }
    @Override
    protected void onResume() {    // 使其中 View 重新开始
        super.onResume();
        mGLView.onResume();
    }
    private GLSurfaceView mGLView;
}
```

程序的在 onPause()和 onResume()中调用 View 的对应函数。

TriangleRenderer.java 的主要函数的内容如下所示：

```
    public TriangleRenderer(Context context) {
```

```
        mContext = context;
        mTriangle = new Triangle();
    }
    public void onDrawFrame(GL10 gl) {
        gl.glDisable(GL10.GL DITHER);
        gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
              GL10.GL_MODULATE);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
// 开始绘制
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glActiveTexture(GL10.GL_TEXTURE0);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
                           GL10.GL_REPEAT);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
                           GL10.GL_REPEAT);
// 获取系统时间
        long time = SystemClock.uptimeMillis() % 4000L;
        float angle = 0.090f * ((int) time);
        gl.glRotatef(angle, 0, 0, 1.0f);       // 实现旋转
        mTriangle.draw(gl);
    }
```

OpenGL 本身带回动画绘制的功能，这里使用的 glRotatef()是进行旋转。mTriangle 是一个三角形，在 onDrawFrame()根据 OpenGL 的上下文进行动画的绘制。