

## 第一天 Android 基础

第一章 快速入门.....	2
1.1 Android 系统简介.....	2
1.1.1 1G-4G 介绍.....	2
1.1.2 Android 系统由来.....	2
1.1.3 Android 进化史.....	3
1.1.4 Android 系统架构.....	3
1.1.5 两种虚拟机的不同.....	4
1.1.6 ART 模式.....	5
1.2 搭建开发环境.....	6
1.2.1 ADT-Bundle 的下载和安装.....	6
1.2.2 SDK 目录结构.....	6
1.2.3 SDK Manager.....	8
1.2.4 AVD 的使用.....	9
1.2.5 DDMS 简介.....	12
1.3 Android 程序入门.....	15
1.3.1 Hello World 的创建.....	15
1.3.2 Android 工程目录结构.....	18
1.3.3 Android 的打包过程.....	20
1.3.4 ADB 简介.....	21
1.4 案例-电话拨号器.....	22
1.4.1 编写布局.....	22
1.4.2 编写代码.....	23
1.4.3 添加权限.....	25
1.5 案例-短信发送器.....	25
1.5.1 编写布局.....	26
1.5.2 编写代码.....	27
1.5.3 添加权限.....	29
1.6 点击事件的四种实现方式.....	29
1.7 Android 常用布局.....	30
1.7.1 相对布局.....	30
1.7.2 线性布局.....	32
1.7.3 帧布局.....	34
1.8 Android 中的长度单位.....	36
1.8.1 px.....	36
1.8.2 dip 或 dp.....	36
1.8.3 sp.....	36
1.8.4 dp 和 px 的区别.....	36

## 第一章 快速入门

- ◆ 了解 Android 操作系统
- ◆ 搭建 Android 开发工具
- ◆ 使用 Android 模拟器
- ◆ 案例-电话拨号器/短信发送器
- ◆ 掌握点击事件的四种实现方式
- ◆ 掌握 Android 常见布局
- ◆ 了解 Android 中的长度单位

### 1.1 Android 系统简介

#### 1.1.1 1G-4G 介绍

手机的发展根据通信技术大致可以划分为 4 个时代（G：Generation 的缩写）：

- 1、1G：模拟制式手机,手机类似于简单的无线双工电台,通话是锁定在一定的频率,所以使用可调频电台就可以实现窃听;
- 2、2G：手机使用 PHS，GSM 或者 CDMA 这些十分成熟的标准，具有稳定的通话质量和合适的待机时间，支持彩信业务的 GPRS 和上网业务的 WAP 服务，以及各式各样的 Java 程序等；
- 3、3G：将无线通信与国际互联网等多媒体通信结合的新一代移动通信系统。它能够处理图像、音乐、视频流等多种媒体形式，提供包括网页浏览、电话会议、电子商务等多种信息服务；
- 4、4G：该技术包括 TD-LTE 和 FDD-LTE 两种制式。4G 是集 3G 与 WLAN 于一体，并能够传输高质量视频图像。4G 理论上可以达到 100Mbps 的下载速度。此外，4G 可以在 DSL 和有线电视调制解调器没有覆盖的地方部署，然后再扩展到整个地区。

#### 1.1.2 Android 系统由来

Android 系统最初由安迪·鲁宾等人开发制作，最初开发这个系统的目的是创建一个数码相机的先进操作系统；后来发现市场需求不够大，加上智能手机市场快速成长，于是 Android 被改造为一款面向智能手机的操作系统，2005 年由 Google 收购注资，并组建开放手机联盟。2007 年 11 月 12 日，Android Beta 操作系统 SDK 正式发布。

### 1.1.3 Android 进化史

2008 年 9 月 23 日 Android 1.0 发布，代号 Bender（发条机器人），这也是 Android 系统最早的版本。

2009 年 9 月 15 日 Android 1.6 发布，代号 Donut（甜甜圈）该版本首次支持了 CDMA 网络。

2009 年 11 月 Android 2.0 发布，代号 Eclair（松饼）无论从哪个方面说，它都是 Android 发展历史上第二个重要的里程碑时刻（第一个是 Android1.5）。

2010 年 5 月 20 日 Android 2.2 发布，代号 Froyo（冻酸奶）为 Android 添加了很多企业级功能。

2011 年 10 月 19 日 Android 4.0 发布，代号 Ice Cream Sandwich（冰激凌三明治）是 Android 发展历史上最重大的一次升级。

2012 年 6 月 28 日 Android 4.1 发布，代号 Jelly Bean（果冻豆）是谷歌继蜂巢之后，一次全新的平板策略尝试。

2014 年 10 月 15 日 Android 5.0 发布，代号 Lollipop（棒棒糖），全新的 UI 设计，全新的操作系统。

2015 年 10 月 6 日 Android 6.0 发布，代号 Marshmallow（棉花糖），这次的新版系统在 UI 和交互上和 Android 5.X 保持高度一致。

### 1.1.4 Android 系统架构

Android 的系统架构采用了分层的设计。从架构图（图 1-1）看，Android 分为四层，从低层到高层分别是 Linux 内核层、系统运行库层、应用程序框架层和应用程序层。



图 1-1 Android 系统架构

#### 1. Linux 内核

Android 的核心系统服务依赖于 Linux 2.6 内核，如安全性，内存管理，进程管理，网络协议栈和驱动

模型。Linux 内核也同时作为硬件和软件栈之间的抽象层。

## 2. 系统运行库

### 1) 程序库

Android 包含一些 C/C++ 库，这些库能被 Android 系统中不同的组件使用。它们通过 Android 应用程序框架为开发者提供服务。以下是一些核心库：

- \* **Surface Manager** - 对显示子系统的管理，并且为多个应用程序提供了 2D 和 3D 图层的无缝融合。
- \* **媒体库** - 该库支持多种常用的音频、视频格式回放和录制，同时支持静态图像文件。编码格式包括 MPEG4, H.264, MP3, AAC, AMR, JPG, PNG。
- \* **SQLite** - 一个对于所有应用程序可用，功能强劲的轻型关系型数据库引擎。
- \* **OpenGL ES** - 该库可以使用硬件 3D 加速(如果可用)或者使用高度优化的 3D 软加速。
- \* **FreeType** - 位图(bitmap)和矢量(vector)字体显示。
- \* **WebKit** - 一个最新的 web 浏览器引擎，支持 Android 浏览器和一个可嵌入的 web 视图 (WebView)。
- \* **SGL** - 底层的 2D 图形引擎
- \* **SSL** - SSL(Secure Sockets Layer 安全套接层),在传输层对网络连接进行加密,为网络通信提供安全及数据完整性的一种安全协议。
- \* **Libc** - 一个标准 C 系统函数库( libc )。

### 2) Android Runtime 库

- \* **Core Libraries** - 该核心库提供了 Java 编程语言核心库的大多数功能。
- \* **Dalvik Virtual Machine** - 每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。

Dalvik 虚拟机依赖于 linux 内核的一些功能，比如线程机制和底层内存管理机制。

## 3. 应用程序框架

Android 系统中的每个应用都依赖于该框架提供的一系列服务和系统，其中包括：

- \* **活动管理器(Activity Manager)** 用来管理应用程序生命周期并提供常用的导航回退功能。
- \* **丰富而又可扩展的视图(Views)**，可以用来构建应用程序，它包括列表(lists)，网格(grid)，文本框(text boxes)，按钮(buttons)，甚至可嵌入的 web 浏览器。
- \* **内容提供者(Content Providers)**使得应用程序可以访问另一个应用程序的数据(如联系人数据库)，或者共享它们自己的数据。
- \* **资源管理器(Resource Manager)**提供非代码资源的访问，如本地字符串，图形，和布局文件(layout files)。
- \* **通知管理器(Notification Manager)** 使得应用程序可以在状态栏中显示自定义的提示信息。

## 4. 应用程序层

该层不仅包括系统内置的应用也包括用户自己安装的应用，比如 email 客户端，SMS 短消息程序，日历，地图，浏览器，联系人管理程序，QQ，微信，淘宝，美团等，该层所有的应用程序都是使用 Java 语言编写的。

## 1.1.5 两种虚拟机的不同

JavaSE 程序使用的虚拟机叫 Java Virtual Machine，简称 JVM，Android 应用也使用 Java 语言开发，但是使用的虚拟就是 Dalvik Virtual Machine，简称 DVM。

Dalvik 是 Google 公司自己设计用于 Android 平台的 Java 虚拟机。它执行的是已转换为 .dex(即 Dalvik Executable) 格式的 Java 应用程序的运行，.dex 格式是专为 Dalvik 设计的一种压缩格式，适合内存和处理速度有限的系统。

Dalvik 经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个 Dalvik 应用作为一个独立的 Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

Dalvik 和标准 Java 虚拟机(JVM)首要差别：

Dalvik 基于寄存器，而 JVM 基于栈。

基于寄存器的虚拟机对于编译后变大的程序来说，在它们执行的时候，花费的时间更短。

Dalvik 和 Java 运行环境的区别：

1. Dalvik 主要是完成对象生命周期管理，堆栈管理，线程管理，安全和异常管理，以及垃圾回收等重要功能；

2. Dalvik 负责进程隔离和线程管理，每一个 Android 应用在底层都会对应一个独立的 Dalvik 虚拟机实例，其代码在虚拟机的解释下得以执行；

3. 不同于 Java 虚拟机运行 Java 字节码，Dalvik 虚拟机运行的是其专有的文件格式 Dex；

4. dex 文件格式可以减少整体文件尺寸，提高 I/O 操作的类查找速度；

5. odex 是为了在运行过程中进一步提高性能，对 dex 文件的进一步优化；

6. 所有的 Android 应用的线程都对应一个 Linux 线程，虚拟机因而可以更多的依赖操作系统的线程调度和管理机制；

7. 有一个特殊的虚拟机进程 Zygote，他是虚拟机实例的孵化器。它在系统启动的时候就会产生，它会完成虚拟机的初始化，库的加载，预制类库和初始化的操作。如果系统需要一个新的虚拟机实例，它会迅速复制自身，以最快的数据提供给系统。对于一些只读的系统库，所有虚拟机实例都和 Zygote 共享一块内存区域；

8: Dalvik 是由 Dan Bornstein 编写的，名字来源于他的祖先曾经居住过名叫 Dalvík 的小渔村，村子位于冰岛。

## 1.1.6 ART 模式

一、ART 模式是什么？

ART 模式英文全称为：Android runtime，谷歌从 Android 4.4 系统开始新增的一种应用运行模式，与传统的 Dalvik 模式不同，ART 模式可以实现更为流畅的 Android 系统体验。

在 4.4 系统之前，Android 系统在 Linux 的底层下构筑 Dalvik 一层的虚拟机，通过其可以更好适应多样的硬件架构，开发者只需要按一套规则进行应用便可，无需因为不同的硬件架构而处理与底层的驱动关系，从而大大提高开发的效率，但因为应用均是运行在 Dalvik 虚拟机中，因此应用程序每次运行的时候，一部分代码都需要重新进行编译，这过程需要消耗一定的时间和降低应用的执行效率，最明显的便是拖延了应用的启动时间和降低了运行速度。

二、ART 模式有什么作用？

ART 模式最大的作用就是提升了 Android 系统流畅度，相比 Dalvik 模式中出现的耗电快、占用内存大、即使是旗舰机用久了也会卡顿严重等现象，ART 模式中这种问题得到了很好的解决，通过在安装应用程序时，自动对程序进行代码预读取编译，让程序直接编译成机器语言，免去了 Dalvik 模式要时时转换代码，实现高效率、省电、占用更低的系统内存、手机运行流畅。

三、ART 模式的缺点

ART 模式可以降低手机硬件配置要求，减少 RAM 内存依赖，不过在安卓 4.4 系统中，安装应用的时间比安卓 4.4 以下版本系统更长，这主要由于应用安装过程中需要先执行编码导致，并且安装应用更占存储空间（ROM）。

## 1.2 搭建开发环境

目前主流的开发工具有两个，一个是 Eclipse 另外一个 Android Studio。Eclipse 需要和 ADT (Android Develop Tool) 插件整合后才能使用，不过 Google 官方已经直接提供了 Eclipse 和 ADT 集成好的开发工具，叫 ADT-Bundle。

Android Studio 是 Google 基于 IntelliJ IDEA 开发的 Android 集成开发工具，目前国内使用该开发工具的企业也越来越多。Android 基础阶段我们依然使用 Eclipse 作为开发工具，在后面的课程中才会使用到 Android Studio。

### 1.2.1 ADT-Bundle 的下载和安装

第一步：JDK 的安装：

官方下载地址：<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

下载好以后安装，并配置系统环境变量，这里就不再赘述。

第二步：下载 ADT-Bundle for Windows

官方下载地址：<http://developer.android.com/sdk/index.html>

由于该网址在国内已经被屏蔽掉了，因此本人提供了一个百度网盘的下载地址。

网盘下载地址：<http://pan.baidu.com/s/1gdwIAER>

第三步：安装 ADT-Bundle

直接把下载下来的 adt-bundle-windows-xxx-xxx.zip 解压到需要安装的位置。

### 1.2.2 SDK 目录结构

将下载好的 ADT-Bundle 解压后根目录结构如图 1-2 所示，总共包括 3 个部分，eclipse、sdk、SDK Manager.exe。

名称	修改日期	类型	大小
eclipse	2015/10/19 15:19	文件夹	
sdk	2015/9/12 17:53	文件夹	
SDK Manager.exe	2013/9/18 10:08	应用程序	350 KB

图 1-2 ADT-Bundle 根目录结构

在本节中将主要介绍 sdk 目录结构。sdk 目录结构如图 1-3。

名称	修改日期	类型
add-ons	2015/9/11 15:14	文件夹
build-tools	2013/9/18 10:08	文件夹
docs	2015/9/11 14:51	文件夹
extras	2015/9/12 19:59	文件夹
platforms	2015/9/11 11:48	文件夹
platform-tools	2013/9/18 10:08	文件夹
sources	2015/10/13 12:05	文件夹
system-images	2015/9/11 11:50	文件夹
temp	2015/9/12 17:53	文件夹
tools	2015/9/23 17:24	文件夹

图 1-3 sdk 目录结构

1、add-ons 这里面保存着附加库，比如 GoogleMaps，当然你如果安装了 OphoneSDK，这里也会有一些类库在里面；

2、build-tools 这里保存着与编译相关的重要工具，比如 aapt、aidl、逆向调试工具 dexdump 和编译脚本 dx，目录结构如图 1-4 所示；

名称	修改日期	类型	大小
lib	2013/9/11 2:57	文件夹	
renderscript	2013/9/11 2:57	文件夹	
aapt.exe	2013/9/11 2:57	应用程序	836 KB
aidl.exe	2013/9/11 2:57	应用程序	271 KB
arm-linux-androideabi-ld.exe	2013/9/11 2:57	应用程序	3,702 KB
bcc_compat.exe	2013/9/11 2:57	应用程序	119 KB
dexdump.exe	2013/9/11 2:57	应用程序	125 KB
dx.bat	2013/9/11 2:57	Windows 批处理...	3 KB
i686-linux-android-ld.exe	2013/9/11 2:57	应用程序	3,702 KB
libbcc.dll	2013/9/11 2:57	应用程序扩展	335 KB
libbccinfo.dll	2013/9/11 2:57	应用程序扩展	388 KB
libclang.dll	2013/9/11 2:57	应用程序扩展	13,786 KB
libLLVM.dll	2013/9/11 2:57	应用程序扩展	18,503 KB
llvm-rs-cc.exe	2013/9/11 2:57	应用程序	1,247 KB
mipsel-linux-android-ld.exe	2013/9/11 2:57	应用程序	1,799 KB
NOTICE.txt	2013/9/11 2:57	TXT 文件	11 KB
source.properties	2013/9/11 2:57	PROPERTIES 文件	1 KB

图 1-4 build-tools 目录结构

3、docs 这里面是 Android SDKAPI 参考文档，所有的 API 都可以在这里查到；

4、extras 拓展开发包，这里面包括下向下兼容开发包以及 Intel 硬件加速程序等；

5、platforms 是每个平台的 SDK 真正的文件，里面会根据 API Level 划分 SDK 版本，这里就以 Android2.3 来说，进入后有一个 android-10 的文件夹，android-10 进入后是 Android2.3 SDK 的主要文件，其中 data 保存着一些系统资源，images 是模拟器映像文件，skins 则是 Android 模拟器的皮肤，templates 是工程创建的默认模板，android.jar 则是该版本的主要 framework 文件；



6、platform-tools 保存着一些通用工具，比如 adb.exe；

7、sources Android 系统源码目录，文件夹里可以包含多份源码，根据 API Level 划分为不同的文件夹；

8、system-images Android 系统镜像文件；

9、tools 这里包含了重要的工具，比如 ddms 用于启动 Android 调试工具，draw9patch 则是绘制 android 平台的可缩放 png 图片的工具，sqlite3 可以在 PC 上操作 SQLite 数据库，emulator 是 Android SDK 模拟器主程序，traceview 作为 android 平台上重要的调试工具；

### 1.2.3 SDK Manager

SDK Manager.exe 是 ADT 根目录下的一个 SDK 管理程序，该程序负责下载、更新、删除与 sdk 相关的文件。启动 SDK Manager 后程序主界面如图 1-5 所示。

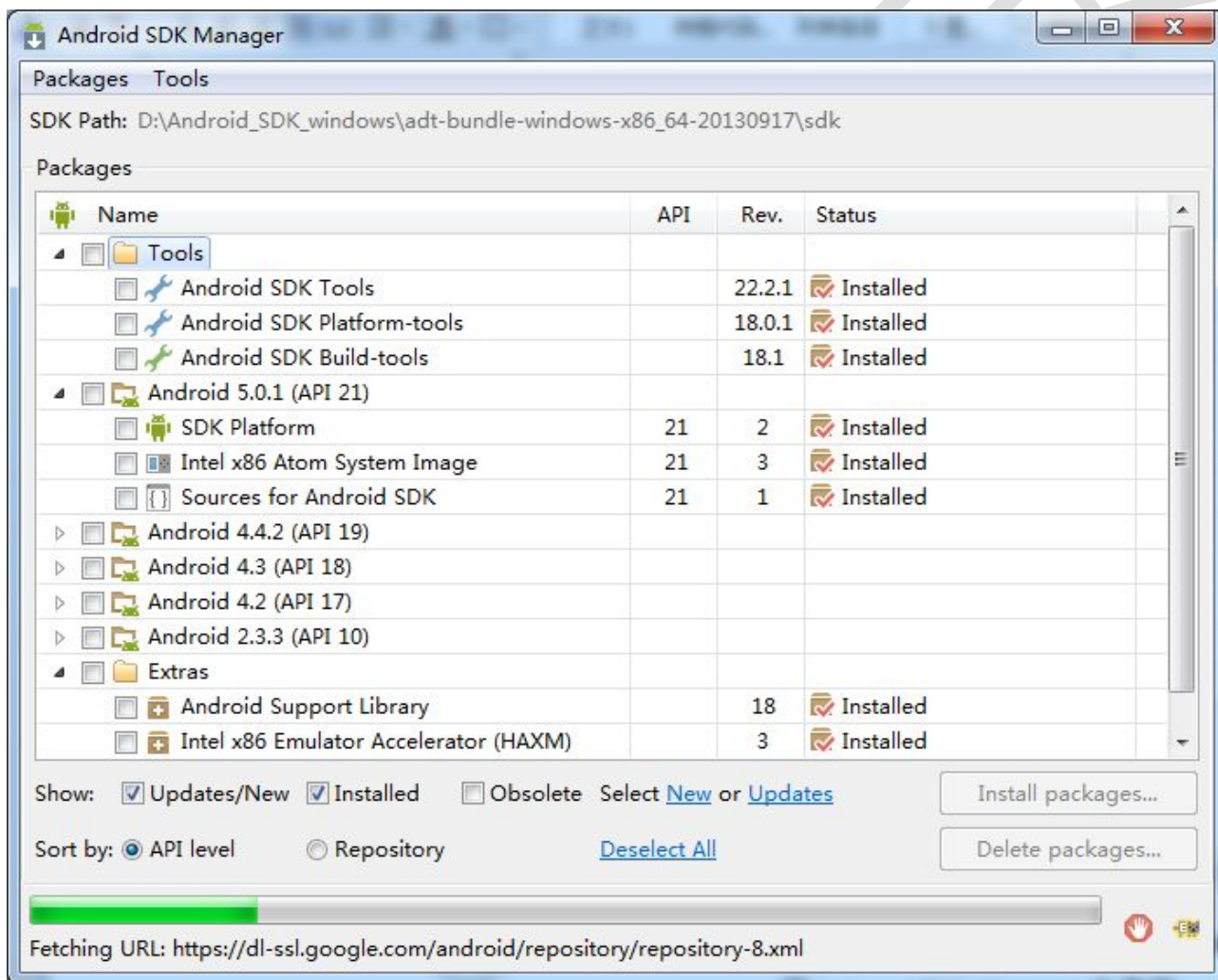


图 1-5 SDK Manager 主界面

该管理器主要分为以下几部分：

1、Tools 又分为 Android SDK Tools、Android SDK Platform-tools、Android SDK Build-tools。分别对应 sdk 目录中的 tools、platform-tools、build-tools 三个目录。其中的 Rev. 代表版本号，Status 代表状态。

2、Android x.x.x(API xx) 一般会有多个，一个 API 就有一个条目。该条目下分为 SDK Platform、System Image、Sources for Android SDK。分别对应 sdk 目录中的 platforms、system-images、sources 目录。

3、Extras 主要分为 Android Support Library 和 HAXM（Inter x86 模拟器加速器），对应 sdk 目录中的



tools 文件。

由于 SDK Manager 需要通过连接国外互联网才可以对 sdk 进行升级管理，但是国内是被屏蔽了的，因此大家可以直击从网站上下载现成的 sdk 文件，然后放到相应的目录下，这样就不需要 SDK Manager 进行额外的管理工作。

## 1.2.4 AVD 的使用

AVD 是 Android Virtual Machine 的简称，是用来管理 Android 虚拟机的程序。一般可以通过点击 eclipse 上的 AVD 图标（如图 1-6）直接启动。



图 1-6 AVD 图标

AVD 启动后的主界面如图 1-7 所示。

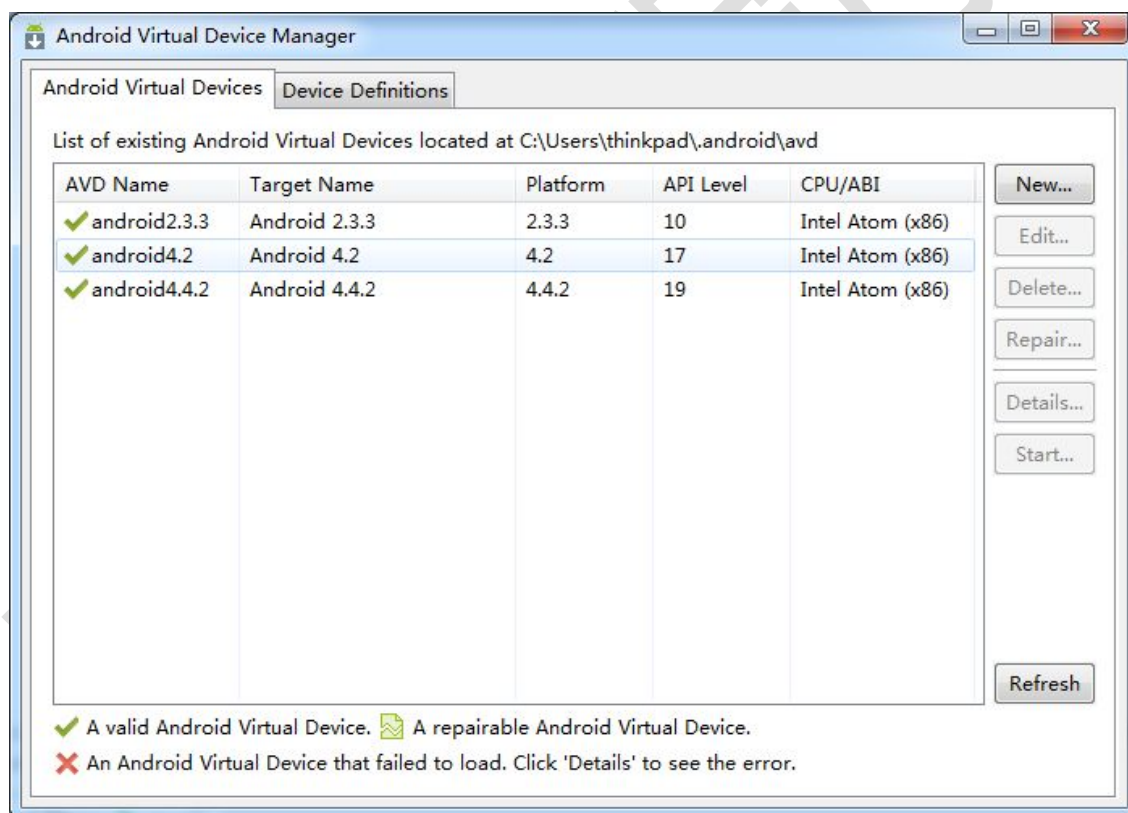


图 1-7 AVD 管理器界面

图 1-7 中的主界面列出了本人已经创建好的所有 Android 虚拟机。具体功能如下所示。点击右侧 New...按钮，弹出如图 1-8 界面。

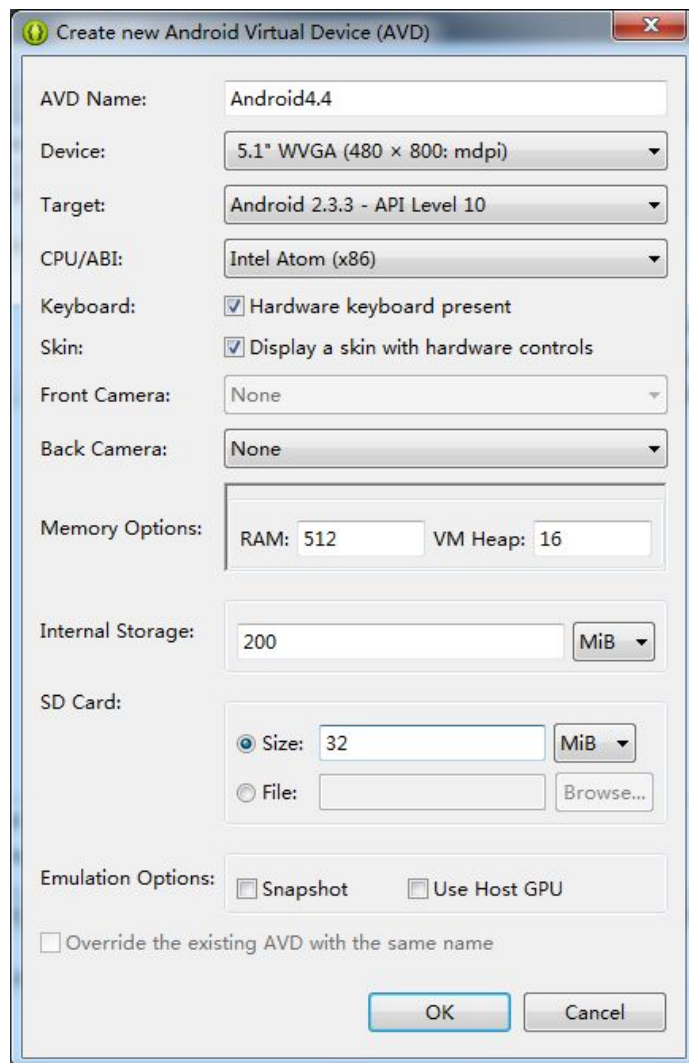


图 1-8 创建新的模拟器界面

上面的各个选项介绍如下：

- 1、AVD name 自定义的模拟器名称，建议见名知意的命名原则；
- 2、Device 选择设备屏幕分辨率；
- 3、Target 选择 Android 操作系统版本；
- 4、CPU/ABI 选择 CPU 架构。只有 x86 的才支持硬件加速。
- 5、Keyboard 勾选上后可以自己电脑键盘作为模拟器的键盘，不勾选的话默认使用模拟器键盘；
- 6、Skin 勾选上后显示模拟器实体按键，不勾选没有实体按键，建议勾选上；
- 7、Front Camera 选择前置摄像头设备，不用选；
- 8、Back Camera 选择后置摄像头设备，不用选；
- 9、Memory Options RAM 代表模拟器运行内存，VM Heap Dalvik 虚拟机堆内存大小，不建议改动；
- 10、Internal Storage 模拟器内置存储大小；
- 11、SD Card Size 设置 SDCard 的大小；
- 12、Emulation Options 模拟器选项，Snapshot 如果被勾选了，模拟器被关闭时相当于是使用了休眠的形式；下次开启时会重新恢复到关机前状态，不建议勾选。User Host GPU，勾选上代表使用电脑提供图形处理；一般如果我们的应用不涉及 3D 图像处理不需要勾选，甚至在有些电脑上勾选后反而导致模拟器无法运行。

当我们创建好模拟器之后，如果需要修改那么可以在图 1-7 中选中要修改的虚拟机，然后点击 Edit...，如果需要删除，点击 Delete...，如果需要启动点击 Start...

启动模拟器的的时候会弹出如图 1-9 所示的界面。

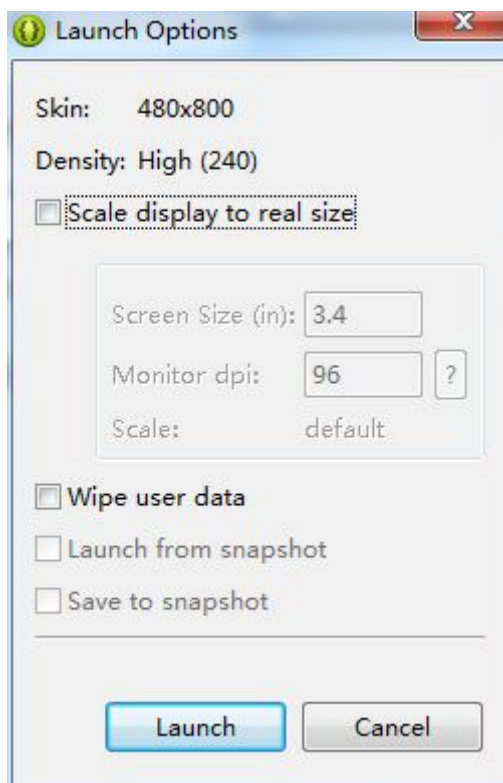


图 1-9 启动模拟器选项

在上图界面中有两个比较重要的选项。Scale display to real size 和 Wipe user data。后者很好理解，如果勾选上了，那么启动时相当于恢复出厂设置。前者用的比较多，是用来缩放我们的模拟器界面的，如果不勾选的话屏幕会显示模拟器的真实大小，可能会占据我们的整个屏幕，因此可以选择勾选，然后将 Screen Size 改为 6（这是我个人经常使用的，大家可以根据自己的喜欢调整）就是一个比较适合的大小。

点击 Launch，就开始启动模拟器，如果没有硬件加速可能需要 3 到 5 分钟的时间。启动好以后的模拟器界面如图 1-10。



图 1-10 模拟器主界面

## 1.2.5 DDMS 简介

DDMS 是 Dalvik Debug Monitor Service 的简称。DDMS 为 IDE 和 emulator 以及 Android 真机架起来了一座桥梁。开发人员可以通过 DDMS 看到目标机器上运行的进程/线程状态，可以看进程的 heap 信息，可以查看 logcat 信息，可以查看进程分配内存情况，可以向目标机发送短信以及打电话，可以向 Android 发送地理位置信息。下面以 Eclipse 的 DDMS perspective 为例简单介绍 DDMS 的功能。图 1-11 为 DDMS 透视图主界面。

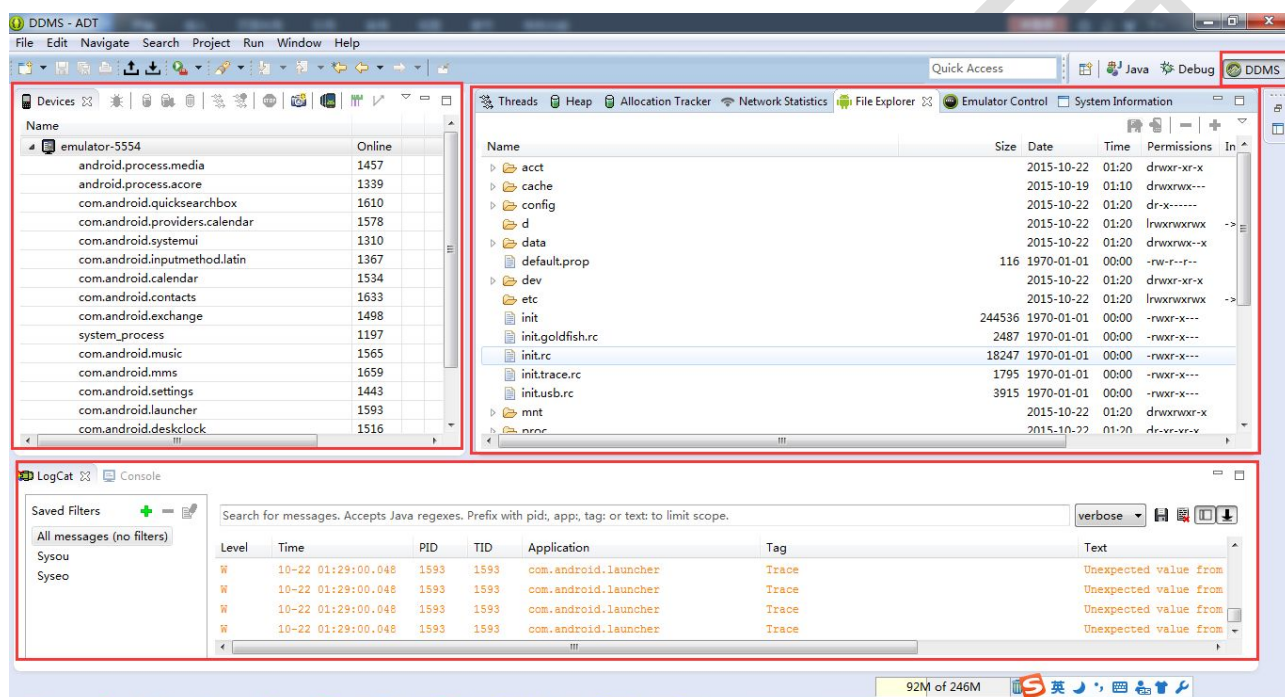


图 1-11

上图中我用红色线框将 DDMS 透视图分为 3 大部分。

左上部分为 Devices 窗口，列出了所有已经跟 adb 成功连接的模拟器（包括真机）以及各个模拟器中所有运行的进程。如图 1-12 所示，最上面一排从左到右一共有 9 个可用按钮，分别为： 调试某个进程， 更新进程堆栈信息， 下载进程堆栈数据到本地， 调用垃圾回收器， 更新线程， 开启方法性能分析数据收集， 停止某个进程， 抓取 Android 目前的屏幕， 查看当前界面视图树结构。

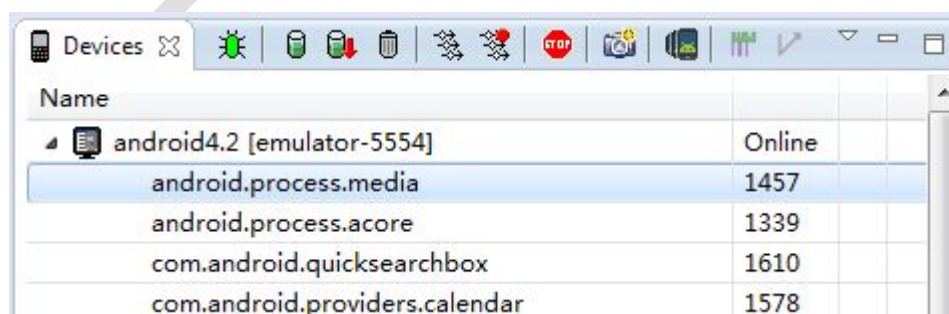






图 1-12 Devices 窗口

右上部分功能比较丰富，且大多数是需要跟左上部分按钮结合者使用。这里的功能一部分用于 Android 性能分析，比如 Threads（如图 1-13）用于查看某个进程中的所有线程运行信息，使用前必须先在左上部分选中某个进程然后开启  功能；Heap（如图 1-14）是堆内存分析工具，使用前必须在左上部分选中某个进程然后开启  功能；Allocation Tracker（如图 1-15）是内存分配查看器；Network Statistic 是网络统计工具；File Explorer（如图 1-16）是我们学习基础阶段用的最多的视图，该界面列出了整个 Android 操作系统的文件目录结构；Emulator Control（如图 1-17）是模拟器控制器，是用于给 Android 模拟器发送测试数据的工具；最后一个 System Information 展示了 CPU 加载、内存使用的情况，一般用不到。

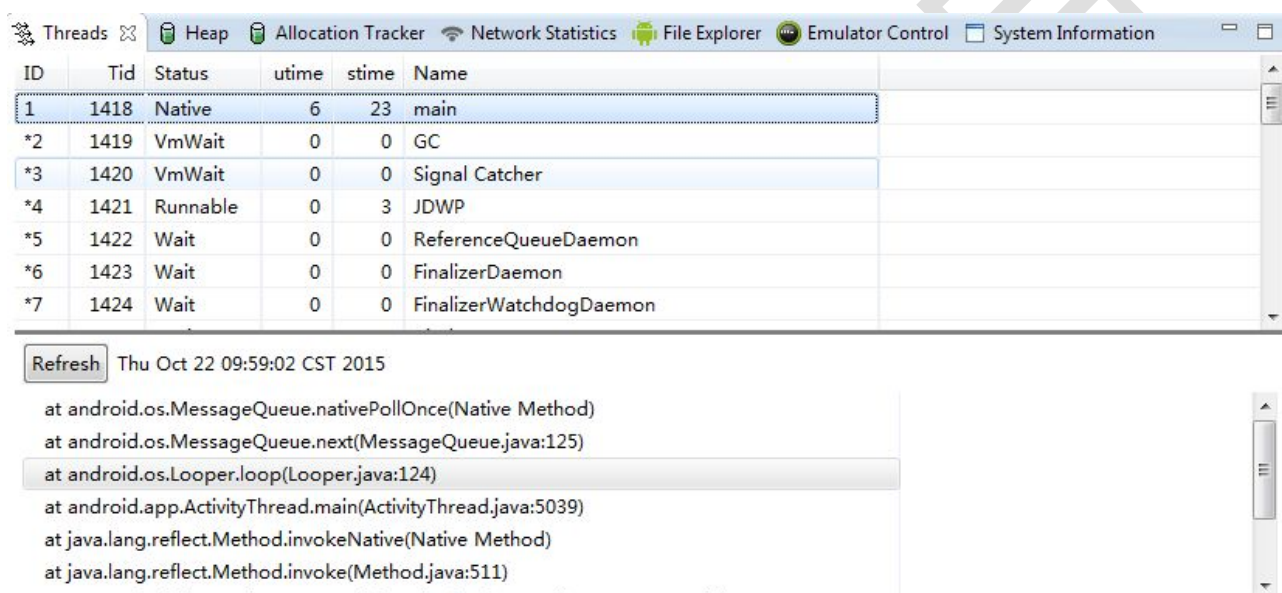


图 1-13 DDMS 右上部分视图

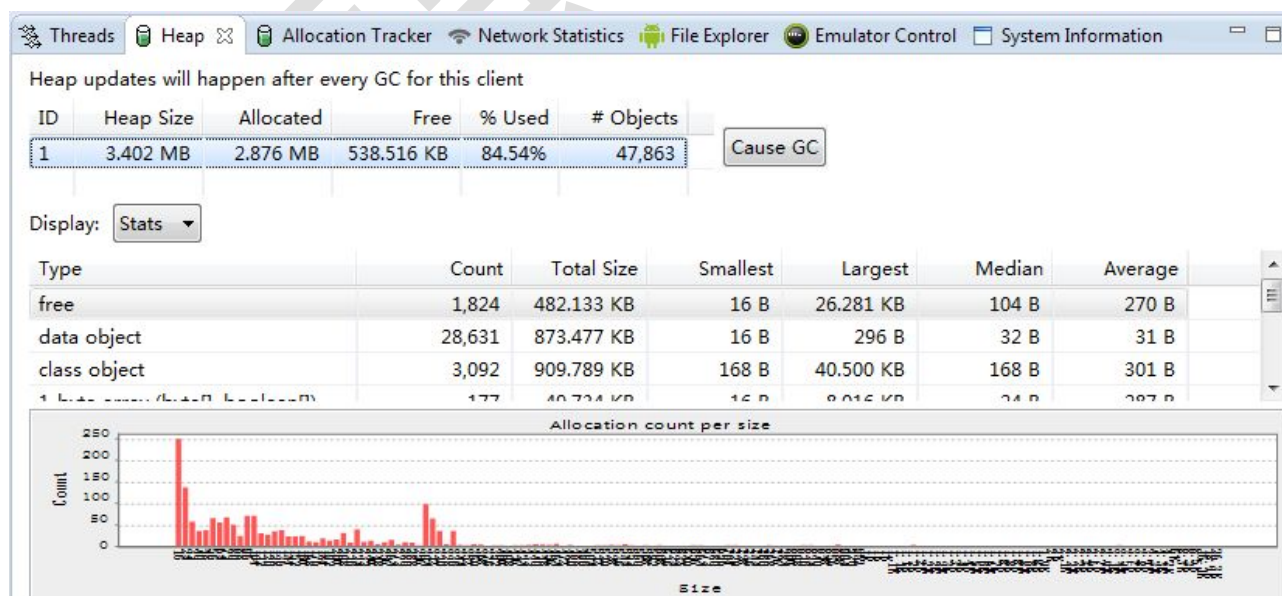
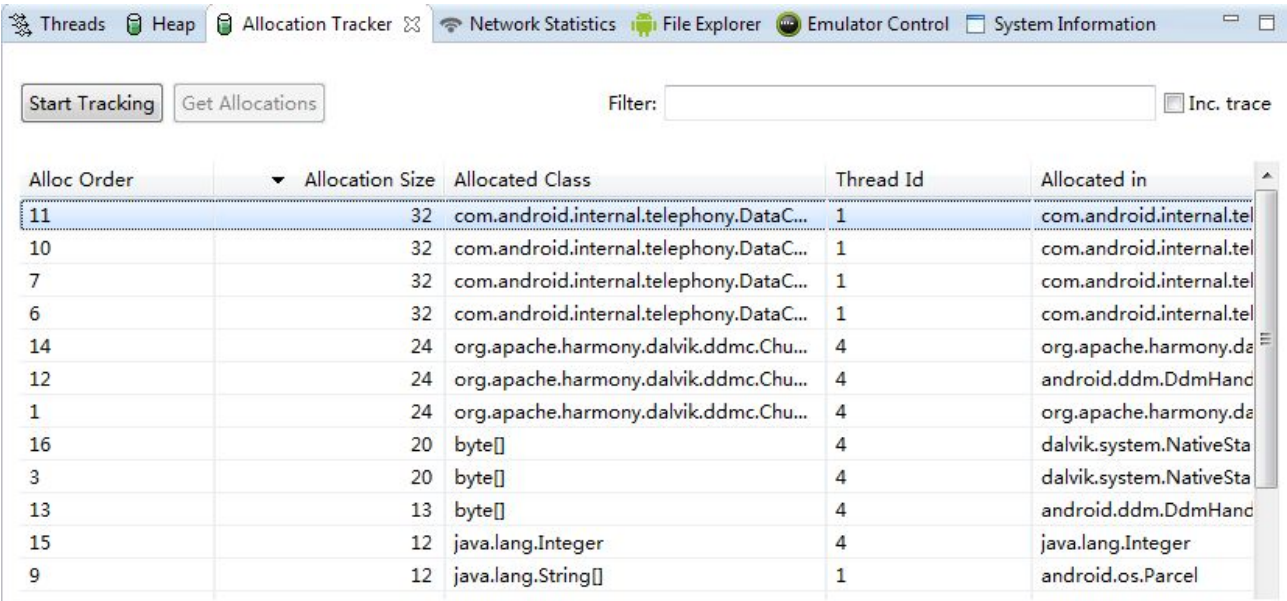
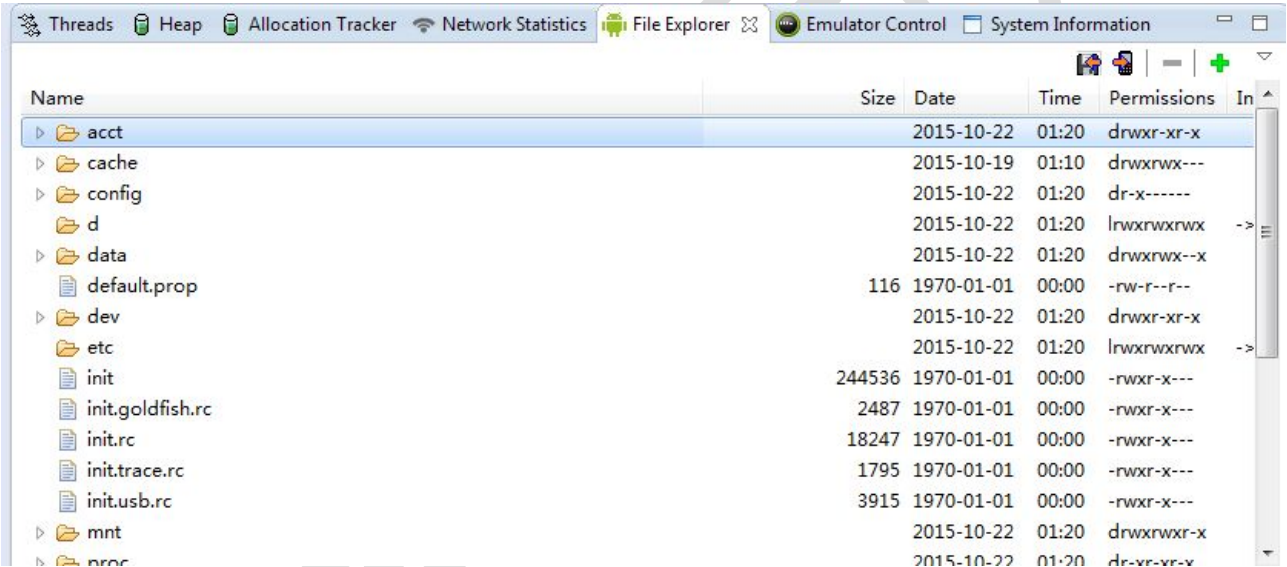


图 1-14 Heap 堆内存分析视图



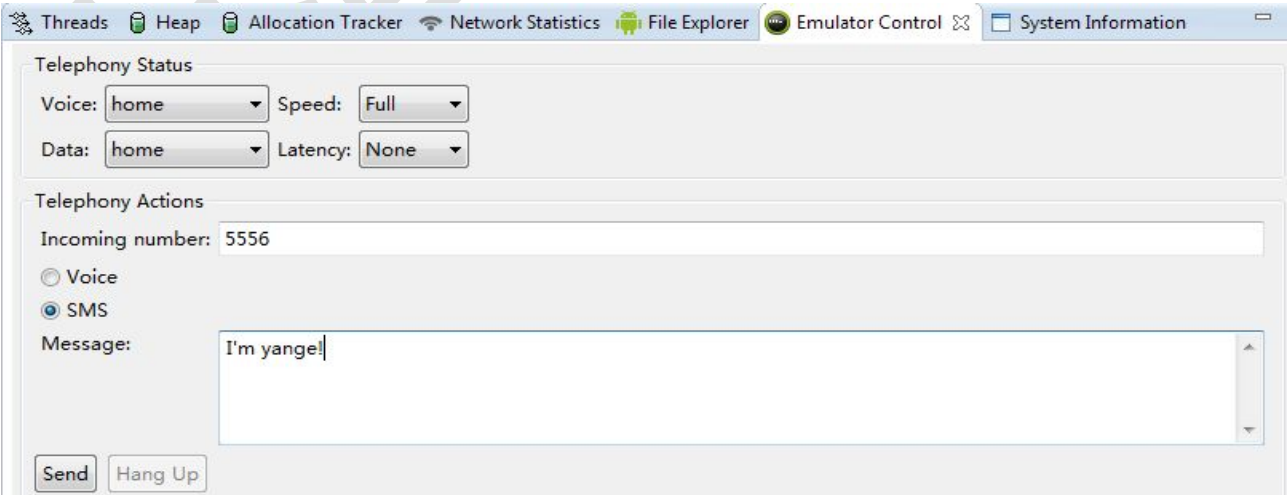
Alloc Order	Allocation Size	Allocated Class	Thread Id	Allocated in
11	32	com.android.internal.telephony.DataC...	1	com.android.internal.tel
10	32	com.android.internal.telephony.DataC...	1	com.android.internal.tel
7	32	com.android.internal.telephony.DataC...	1	com.android.internal.tel
6	32	com.android.internal.telephony.DataC...	1	com.android.internal.tel
14	24	org.apache.harmony.dalvik.ddmc.Chu...	4	org.apache.harmony.da
12	24	org.apache.harmony.dalvik.ddmc.Chu...	4	android.ddm.DdmHand
1	24	org.apache.harmony.dalvik.ddmc.Chu...	4	org.apache.harmony.da
16	20	byte[]	4	dalvik.system.NativeSta
3	20	byte[]	4	dalvik.system.NativeSta
13	13	byte[]	4	android.ddm.DdmHand
15	12	java.lang.Integer	4	java.lang.Integer
9	12	java.lang.String[]	1	android.os.Parcel

图 1-15 Allocation Tracker 内存分配查看视图



Name	Size	Date	Time	Permissions	In
acct		2015-10-22	01:20	drwxr-xr-x	
cache		2015-10-19	01:10	drwxrwx---	
config		2015-10-22	01:20	dr-x-----	
d		2015-10-22	01:20	lrwxrwxrwx	->
data		2015-10-22	01:20	drwxrwx--x	
default.prop	116	1970-01-01	00:00	-rw-r--r--	
dev		2015-10-22	01:20	drwxr-xr-x	
etc		2015-10-22	01:20	lrwxrwxrwx	->
init	244536	1970-01-01	00:00	-rwxr-x---	
init.goldfish.rc	2487	1970-01-01	00:00	-rwxr-x---	
init.rc	18247	1970-01-01	00:00	-rwxr-x---	
init.trace.rc	1795	1970-01-01	00:00	-rwxr-x---	
init.usb.rc	3915	1970-01-01	00:00	-rwxr-x---	
mnt		2015-10-22	01:20	drwxrwxr-x	
proc		2015-10-22	01:20	dr-xr-xr-x	

图 1-16 File Explorer 透视图



Telephony Status

Voice: home Speed: Full

Data: home Latency: None

Telephony Actions

Incoming number: 5556

☐ Voice ☒ SMS

Message: I'm yange!

Send Hang Up

图 1-17 Emulator Control 视图



底部是 Logcat 信息，展示了系统以及应用程序输出的日志信息，对 debug 有无比重要的作用，Logcat 的使用在后面的章节会详细的介绍。

## 1.3 Android 程序入门

### 1.3.1 Hello World 的创建

点击 ADT 左上角的 File 按钮，然后选择 New，然后点击 Android Application Project（如图 1-18）开始 Android 工程的创建。（或者鼠标右击，在弹出的快捷菜单中创建也可以）。

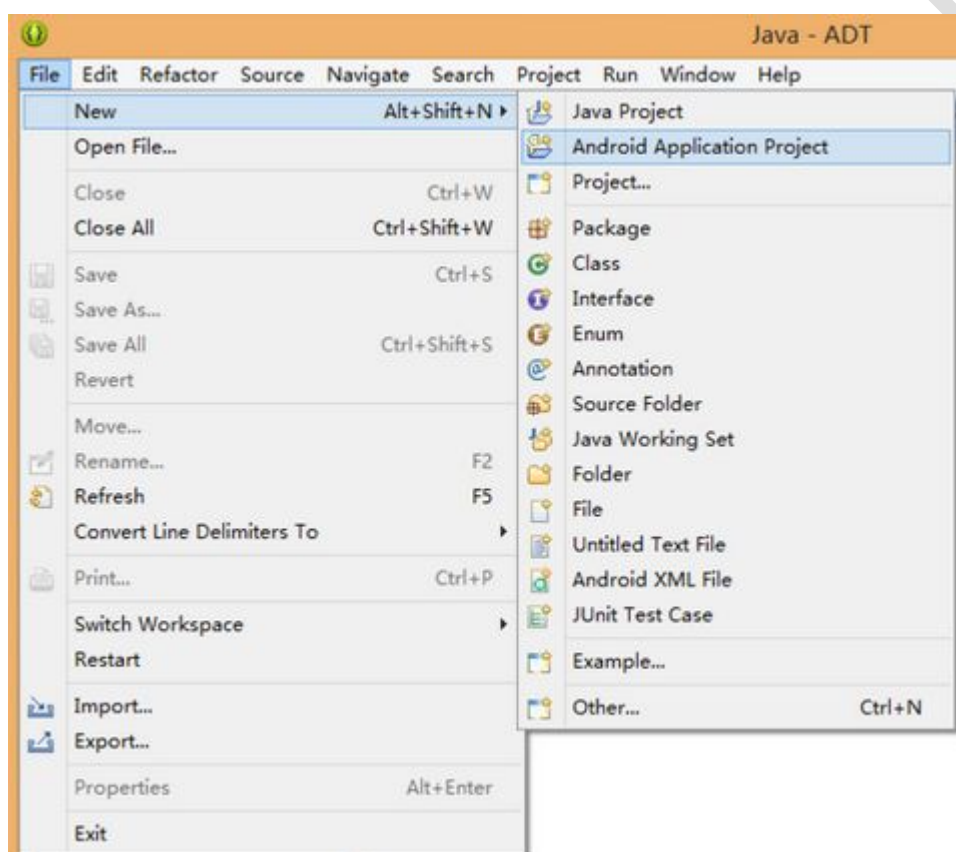


图 1-18 Android 工程的创建

弹出如图 1-19 向导界面 1。Application Name 是应用的名字，该名字会显示在 Android 操作系统桌面图标下方，比如 QQ、微信这样的文字。Project Name 是工程名称，该名称是 Eclipse 工程的名称，与 Android 应用程序没有太大的关系。Package Name 是包名，一个包名代表了唯一一个 Android 应用，在 Android 操作系统中是通过包名管理应用程序的。Minimum Required SDK 是指该应用最低兼容版本。Target SDK 是目标版本，也就是说我们创建的这个 Android 工程是针对哪个 Android 版本开发的。Compile With 是编译器版本，每个 Android 版本都有对应的编译器，建议使用跟 Target SDK 相同的编译器。Theme 是给应用选择主题，不同的主题会让应用显示出不同的样式。

以上各个配置设置好以后，点击 Next...

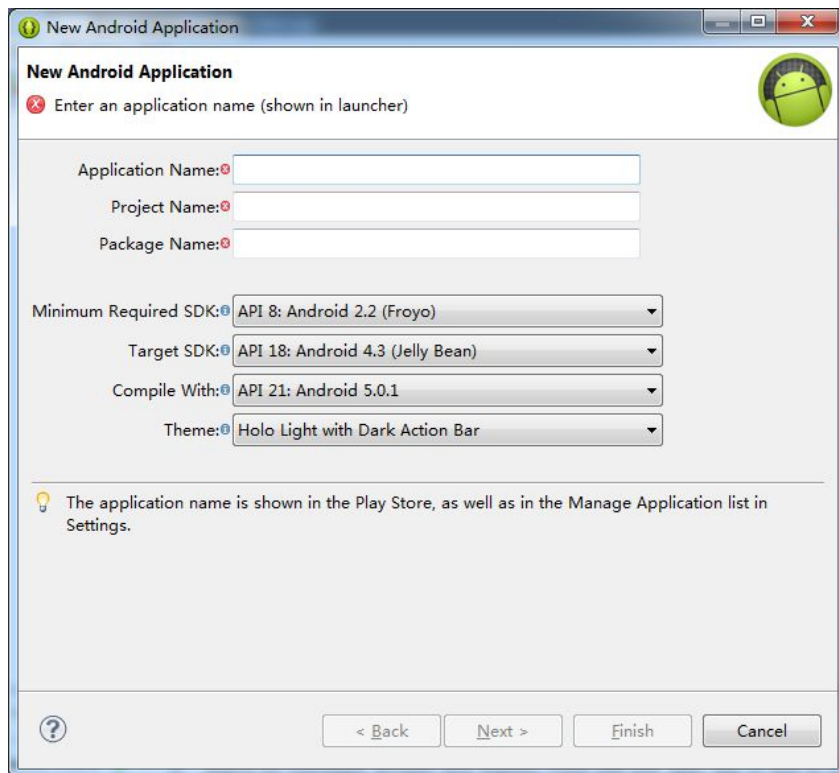


图 1-19 创建 Android 工程向导界面 1

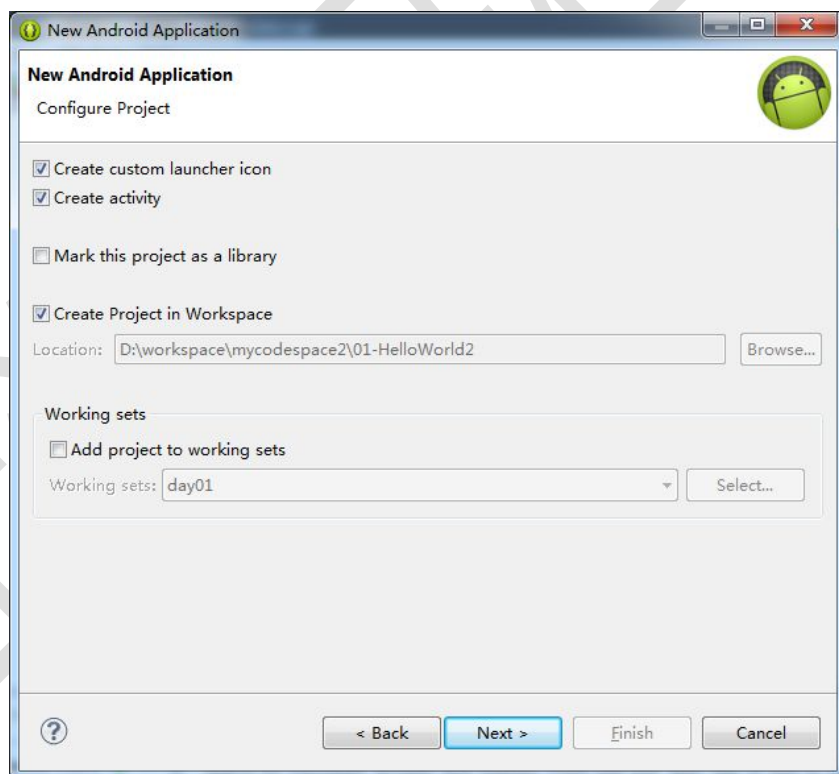


图 1-20 创建 Android 工程向导界面 2

如图 1-20 所示，该界面展示了是否创建图标，是否创建 activity，是否将该工程作为库工程，是否将该工程添加到当前工作空间，是否将该工程添加到指定工作集等信息，通常情况下不需要我们修改默认的配置，直接 Next 就行。

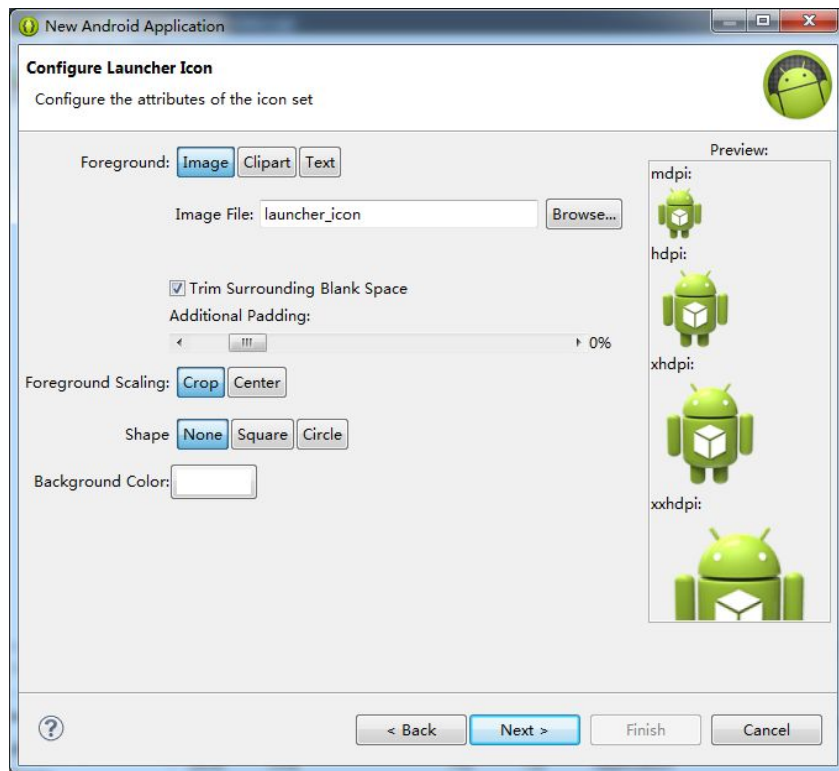


图 1-21 创建 Android 工程向导界面 3

如图 1-21 所示，该界面提供了配置图标选项，我们通过点击 **Browser** 按钮可以选择我们个性化的图标。在学习阶段不需要使用，然后点击 **Next...**。

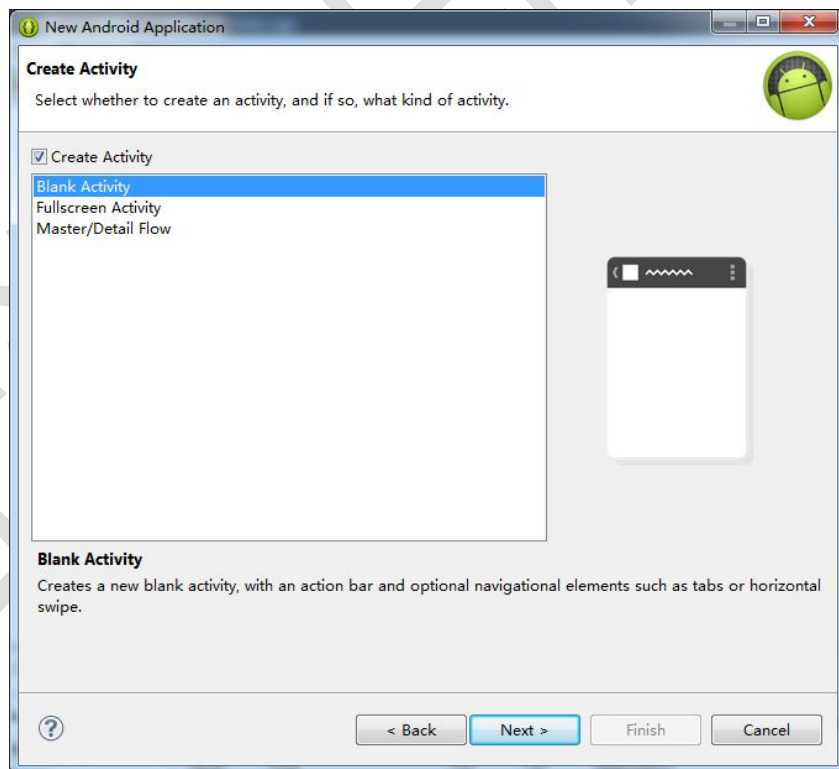


图 1-22 创建 Android 工程向导界面 4

如图 1-22 所示，该向导界面可以让我们选择创建一个什么样式的 Activity，一般使用系统默认的 Blank Activity 即可，然后点击 **Next...**。

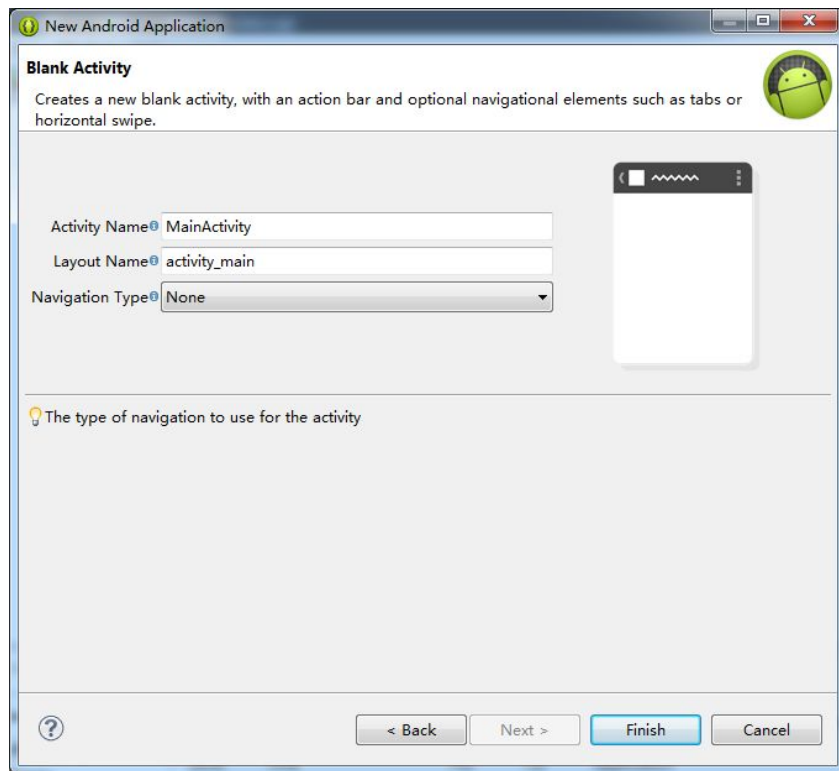


图 1-23 创建 Android 工程向导界面 5

如图 1-23 所示,这是最后一个向导界面了。Activity Name 和 Layout Name 分别是让我们设置主 Activity 和其布局的名字,通常情况下使用默认名字即可。Navigation Type 是主界面 Activity 切换类型,使用 None 即可。然后点击 Finish 完成 Android 工程的创建。

### 1.3.2 Android 工程目录结构

在 1.3.1 中创建好的工程目录视图如图 1-24 所示。该目录结构说明如下:

- 1、src java 源代码存放目录。
- 2、gen 自动生成的目录

gen 目录中存放所有由 Android 开发工具自动生成的文件。目录中最重要的就是 R.java 文件。这个文件由 Android 开发工具自动产生的。Android 开发工具会自动根据你放入 res 目录的资源,同步更新修改 R.java 文件。正因为 R.java 文件是由开发工具自动生成的,所以我们应避免手工修改 R.java。另外编译器也会检查 R.java 列表中的资源是否被使用到,没有被使用到的资源不会编译进软件中,这样可以减少应用在手机占用的空间。

- 3、bin 用于存放 ADT 编译时产生的临时文件,Android 工程最终会被打包成一个 xxx.apk
- 4、res 资源(Resource)目录

在这个目录中我们可以存放各种各样的资源,如 xml 界面文件,图片或数据,该目录下包含多个子目录。

#### 4.1、res/drawable

存放 png、jpg 等图标文件。在代码中使用 getResources().getDrawable(resourceId)获取该目录下的资源。

#### 4.2、res/layout

存放 xml 界面文件,xml 界面文件和 HTML 文件一样,主要用于显示用户操作界面。

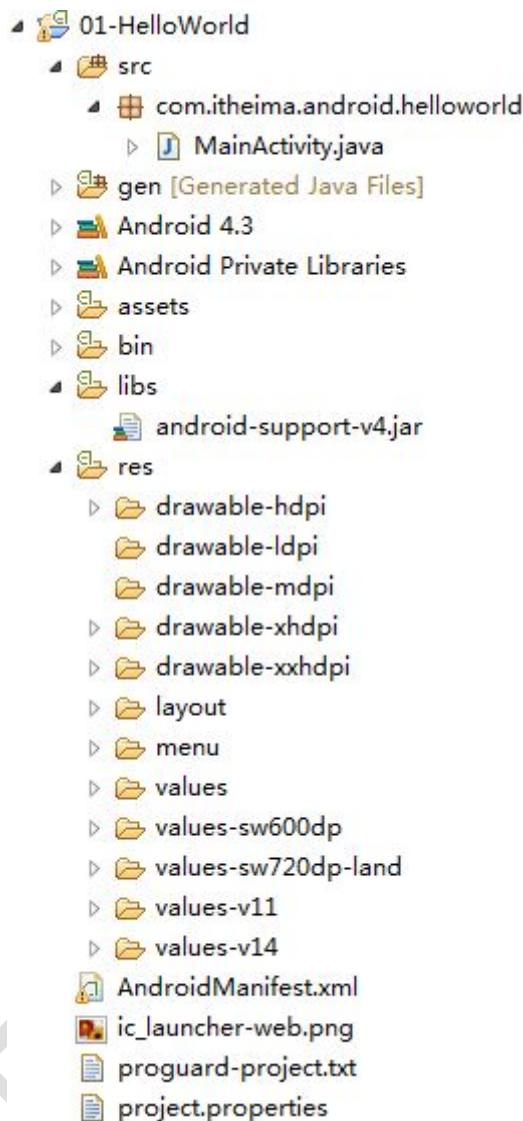


图 1-24 Android 工程基本目录结构

#### 4.3、res/values

存放应用使用到的各种类型数据。不同类型的数据存放在不同的文件中。比如：

4.3.1 strings.xml 定义字符串和数值

4.3.2 colors.xml 定义颜色和颜色字符串数值

4.3.3 dimens.xml 定义尺寸数据

4.3.4 styles.xml 定义样式

#### 4.4、res/anim/

存放自定义动画的 XML 文件。

#### 4.5 res/xml/

在 Activity 中使用 `getResources().getXML()` 读取该目录下的 XML 资源文件。

#### 4.6 res/raw/

该目录用于存放应用使用到的原始文件，如音效文件等。编译软件时，这些数据不会被编译，它们被直接加入到程序安装包里。

#### 5、libs 支持库目录

程序开发时需要的一些三方的 jar 包可以放在这个目录，通常系统会自动把里面的 jar 包，添加到环境变量，如果自动添加不了那么就需要手动添加。



## 6、assets 资源目录

Android 除了提供 res 资源文件外，在 assets 目录中可以存放资源文件，而且 assets 目录下的资源文件不会在 R.java 中自动生成 id。

## 7、AndroidManifest.xml 项目清单文件

该文件用于配置四大组件、声明权限、配置应用版本等参数。

8、project.properties 项目环境信息，一般是不需要修改此文件。

9、proguard-project.txt 用于配置代码混淆参数。

## 1.3.3 Android 的打包过程

### 一、部署 Android 工程

启动模拟器后，将 1.3.1 节中创建好的 HelloWorld 程序部署到模拟器上的步骤如下：

选中 01-HelloWorld 工程，然后右击鼠标，在弹出的菜单中选择 Run As，然后选择 Android Application，如图 1-25 所示。

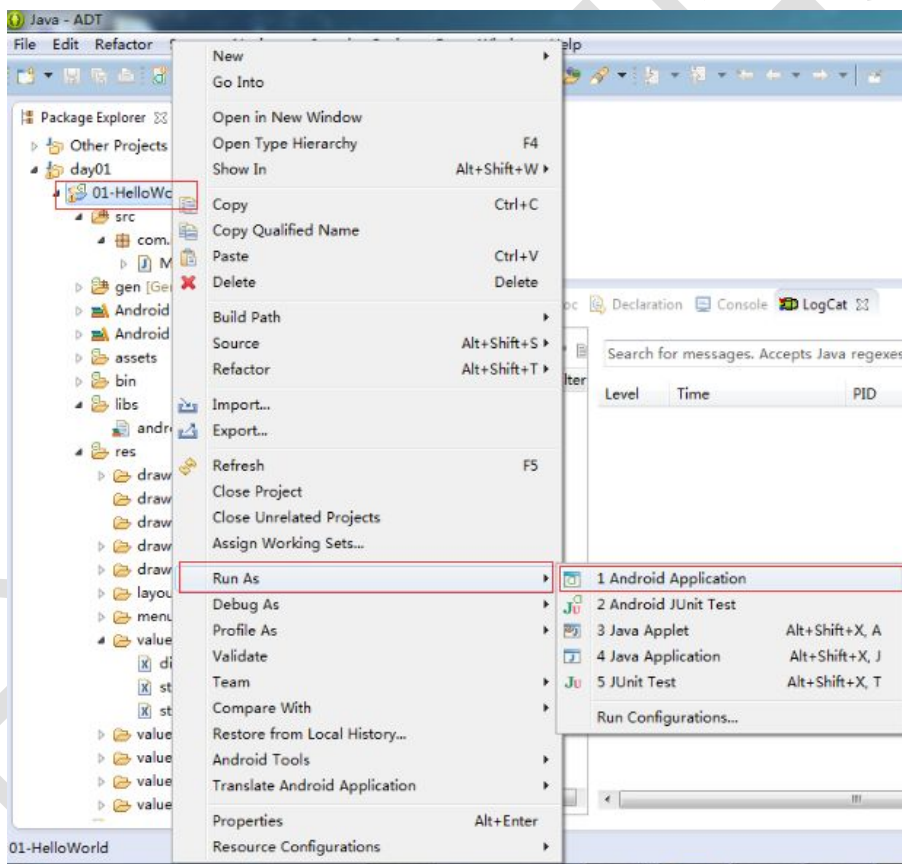


图 1-25 部署应用操作图

部署的时候我们从 Console 中可以观察到如图 1-26 内容：



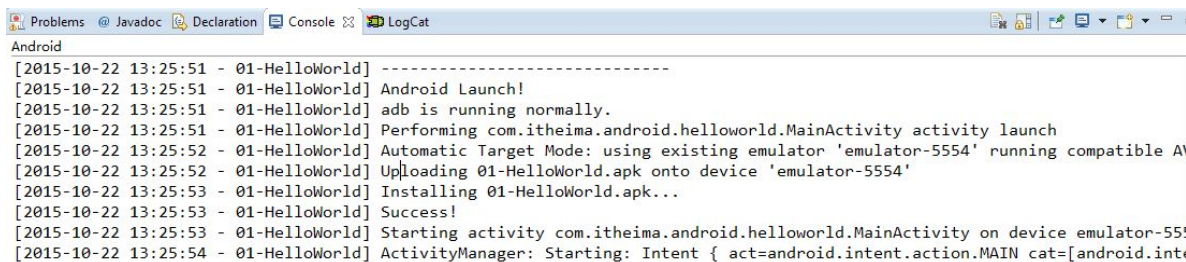


图 1-26 Android 部署过程

上图中的控制台输出了 Android 应用程序的部署过程，分别经过了 Uploading 阶段，Installing 阶段，最后是 Starting，成功之后我们可以看到如图 1-27 界面。

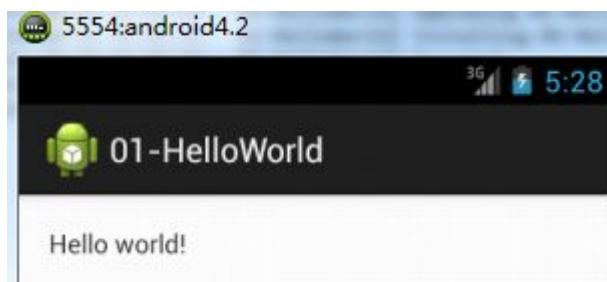


图 1-27 HelloWorld 程序主界面

## 二、Android 的打包过程

ADT 将 Android 工程编译成 APK 中间经历了一系列过程，如图 1-28。

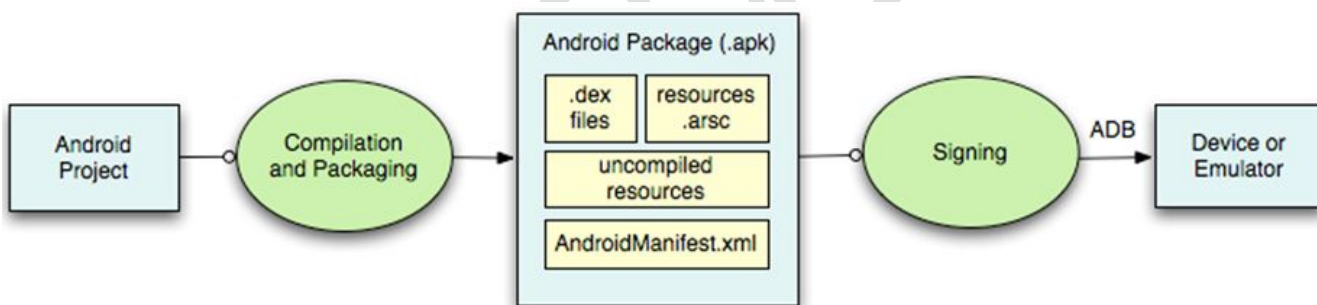


图 1-28 Android 打包过程

ADT 将 Java 源文件编译成.class，然后调用 dx 工具将所有的.class 抽取到一个叫 classes.dex 的文件中。然后调用 aapt 工具将 Android 中所有的资源，包括 res 和 assets 资源以及清单文件一起打包成 apk 文件，然后在部署的时候给 apk 进行签名操作，最后通过 ADB 工具将签名后的 APK 部署到模拟器上。

上面的过程完全是 ADT 自动帮我们完成的。签名是用于区分同一包名的时候应用身份的，也就是如果应用的包名相同签名不一样是不可以安装到同一个手机上的，如果包名相同签名也相同则可以覆盖安装。

## 1.3.4 ADB 简介

### 一、ADB 介绍

adb 是 Android Debug Bridge 的简称，通过 adb 可以在 Eclipse 中通过 DDMS 来调试 Android 程序，adb 启动时会占用 5554 端口，因此要避免其他应用跟该端口冲突。默认情况下所以当我们运行 Eclipse 时 adb 进程就会自动运行。

adb 还可以通过命令行使用，前提是将 adb.exe 所在路径已经配置到了系统环境变量。adb.exe 位于

sdk/platform-tools 目录下。环境变量的配置跟 JDK 相似，不再赘述。

## 二、ADB 常用命令

- 1、adb devices 列出当前连接上所有设备
- 2、adb install xxx.apk 将 xxx.apk 安装到模拟器上
- 3、adb uninstall 包名 卸载应用
- 4、adb push <本地路径><远程路径> 将本地文件上传到模拟器上
- 5、adb pull <远程路径><本地路径> 将模拟器上文件下载到本地
- 6、adb kill-server 杀死 adb 进程
- 7、adb start-server 启动 adb 进程
- 8、adb shell 进入 Linux shell 命令行

## 1.4 案例-电话拨号器

通过该案例可以学会①基本布局的编写②权限的声明③EditText 值的获取③Button 点击事件的监听④意图的使用⑤调用系统拨打电话功能。

需求：如图 1-29 所示，界面顶部有一个文本输入框，当用户输入号码后点击右侧的拨打按钮可以实现拨号功能。



图 1-29 电话拨号器界面

步骤：

### 1.4.1 编写布局

#### 【文件 1-1】activity\_main.xml

```
1. <!-- 使用垂直的线性布局 -->
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:orientation="vertical" >
7.
8.     <!-- 文本编辑框 -->
9.     <EditText
10.         android:id="@+id/et_num"
11.         android:layout_width="match_parent"
12.         android:layout_height="wrap_content"
13.         android:hint="请输入电话号码" />
```

```
14.
15.     <Button
16.         android:id="@+id/btn"
17.         android:layout_width="wrap_content"
18.         android:layout_height="wrap_content"
19.         android:layout_gravity="right"
20.         android:text="拨打" />
21.
22. </LinearLayout>
```

### 注意：

上面的布局文件使用的垂直的线性布局，线性布局有垂直 **vertical** 和水平 **horizontal** 两个方向，若不写默认后者，但是强烈建议大家写出来。

**android:layout\_width** 属性有三个固定的常量值，分别为：**wrap\_content** 包裹类型，控件内容有多大就尽量给分配多大的空间；**match\_parent**、**fill\_parent** 是填充父控件类型，后者已经被废弃，使用前者代替，使用该类型的控件尺寸跟父控件的尺寸保持一样大。控件的尺寸除了这三种常量外还可以输入自定的尺寸。

**android:id="@+id/et\_num"** 属性是为了给每一个控件分配一个编号 **id**，该 **id** 会记录在 **R** 类的子类 **id** 类中，这样方便于我们在代码中通过 **R.id** 获取到控件的引用。“@”后面跟的“+”号意思是该 **id** 对应的值 **et\_num** 需要新添加到 **R** 类中，而不是引用该值。

**android:hint="请输入电话号码"** 提示信息属性，当用户什么都没有输入的时候会显示提示信息。

**android:layout\_gravity="right"** 在 **Button** 控件中使用到了这个属性，表示该控件在父布局中的排列方式，这里的值为 **right**，显然是靠右排列。

**android:text="拨打"** **Button** 按钮显示的文本属性。

## 1.4.2 编写代码

在 **MainActivity** 中实现业务逻辑，如文件【1-2】所示。

### 【文件 1-2】 MainActivity.java

```
1. package com.itheima.android.phone;
2.
3. import android.net.Uri;
4. import android.os.Bundle;
5. import android.app.Activity;
6. import android.content.Intent;
7. import android.view.View;
8. import android.view.View.OnClickListener;
9. import android.widget.Button;
10. import android.widget.EditText;
11. /**
12.  *
13.  * @author wzy 2015-10-22 实现电话拨号器功能
14.  */
15. public class MainActivity extends Activity {
16. // 将两个控件声明为成员变量
```

```
17. private EditText et_num;
18. private Button btn;
19.
20. /**
21.  * 覆写 Activity 类的 onCreate 方法
22.  */
23. @Override
24. protected void onCreate(Bundle savedInstanceState) {
25.     // 必须调用一次父类的该方法，因为父类中做了大量的工作
26.     super.onCreate(savedInstanceState);
27.     /*
28.      * 该方法是从 Activity 类继承过来的，用于将 xml 布局文件跟当前 Activity 绑定起来，
29.      * 然后展现到用户界面
30.      * 通过 R 的内部类 layout（没错，layout 是一个类名，显然没有按照 Java 命令规范来，
31.      * 这里 Google 有其用意）获取到 activity_main.xml 文件的引用
32.      */
33.     setContentView(R.layout.activity_main);
34.     /*
35.      * 通过调用父类的 findViewById 方法，根据控件的 id 获取控件对象
36.      */
37.     et_num = (EditText) findViewById(R.id.et_num);
38.     btn = (Button) findViewById(R.id.btn);
39.     /*
40.      * 给 Button 设置点击监听器，通过该监听器才能监听到用户的点击事件，
41.      * 这里传入的实现 OnClickListener 接口的子类对象
42.      */
43.     btn.setOnClickListener(new MyOnClickListener());
44. }
45. /**
46.  * 自定义类继承 OnClickListener 类，覆写 onClick 方法，如果用户点击了按钮，
47.  * 那么系统会回调 onClick 方法
48.  *
49.  */
50. class MyOnClickListener implements OnClickListener {
51.
52.     @Override
53.     public void onClick(View v) {
54.         // 从 EditText 控件中获取用户输入的数据
55.         String num = et_num.getText().toString().trim();
56.         /*
57.          * 创建意图对象，因此拨打电话其实是调用 Android 系统的拨打电话功能，
58.          * 而且拨打电话是有界面的，因此需要通过 Intent 远程调启 Activity
59.          */
60.         Intent intent = new Intent();
```

```
61.      /*
62.      * 设置 Action，代表了你想干嘛，在这个案例中我们想打电话，这是一个很常用的动作，
63.      * 因此已经被作为 String 类型的常量封装到 Intent 类中了
64.      * 因此这里直接使用 Intent 的 ACTION_CALL 静态变量即可。
65.      *
66.      */
67.      intent.setAction(Intent.ACTION_CALL);
68.      /*
69.      * 给 Intent 设置数据，因为打电话肯定得有对方号码，setData 方法的形参是 Uri 类型的，
70.      * 因此需要使用 Uri 的 parse 方法
71.      * parse 方法中传入的是字符串，在这里字符串还必须按照 tel: xxxx 的数据个数，
72.      * 因此这是系统在配置文件中写死了。随着课程的深入我们会理解为何这里
73.      * 的数据格式必须这么写，这里大家先记住就行。
74.      */
75.      Uri uri = Uri.parse("tel:" + num);
76.      intent.setData(uri);
77.      // 调用父类 Activity 中的 startActivity 方法，传入意图，从而实现调用系统的打电话功能
78.      startActivity(intent);
79.  }
80. }
81. }
82.
```

#### 注意：

上面的代码其实已经实现了点击事件的第一种方式，通过自定义类实现 OnClickListener 接口，然后给 Button 设置 setOnClickListener () 方法中传递该类的对象即可。

### 1.4.3 添加权限

在 Android 系统中凡是可能会侵犯到用户隐私或者耗费用户钱财的行为都必须声明出来，以便让用户在安装 Android 应用时有知情权，然后选择是否安装。这是 Android 安全机制中的一部分。

在该案例中我们的应用实现了拨打电话的功能，那么这种行为肯定是需要声明权限的，因此我们必须在 AndroidManifest.xml 中声明如下权限。

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

添加过权限后部署应用，然后输入号码，测试拨打功能即可。

## 1.5 案例-短信发送器

通过该案例可以学会①短信发送器的使用②第二种方式实现点击事件③TextUtils 的使用④Toast 的使用。

需求：如图 1-30 所示，提供一个号码输入框和一个短信内容输入框，点击按钮的时候将该短信发送到指定手机上。



图 1-30 短信发送器主界面

## 1.5.1 编写布局

### 【文件 1-3】 activity\_main.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical"
6.     android:paddingBottom="@dimen/activity_vertical_margin"
7.     android:paddingLeft="@dimen/activity_horizontal_margin"
8.     android:paddingRight="@dimen/activity_horizontal_margin"
9.     android:paddingTop="@dimen/activity_vertical_margin" >
10.
11.     <EditText
12.         android:id="@+id/et_num"
13.         android:layout_width="match_parent"
14.         android:layout_height="wrap_content"
15.         android:hint="请输入电话" />
16.
17.     <EditText
18.         android:id="@+id/et_sms"
19.         android:layout_width="match_parent"
20.         android:layout_height="wrap_content"
21.         android:hint="请输入短信内容" />
22.
23.     <Button
24.         android:id="@+id/btn"
25.         android:layout_width="match_parent"
26.         android:layout_height="wrap_content"
27.         android:text="发送" />
28.
29. </LinearLayout>
```



### 注意：

上面布局中 LinearLayout 使用了如下属性：

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

上面四个属性分别代表控件在下、左、右、上四个方向的内边距。内边距的值引用自 res/values/dimen.xml 资源文件。

## 1.5.2 编写代码

在 MainActivity 中实现业务逻辑，如文件【1-4】所示。

### 【文件 1-4】 MainActivity.java

```
1. package com.com.itheima.android.sms;
2.
3. import java.util.ArrayList;
4. import android.os.Bundle;
5. import android.telephony.SmsManager;
6. import android.text.TextUtils;
7. import android.view.View;
8. import android.view.View.OnClickListener;
9. import android.widget.Button;
10. import android.widget.EditText;
11. import android.widget.Toast;
12. import android.app.Activity;
13.
14. /**
15.  *
16.  * @author wzy 2015-10-22 实现发送短信功能
17.  */
18. public class MainActivity extends Activity {
19.
20.     private EditText et_num;
21.     private EditText et_sms;
22.     private Button btn;
23.
24.     @Override
25.     protected void onCreate(Bundle savedInstanceState) {
26.         super.onCreate(savedInstanceState);
27.         setContentView(R.layout.activity_main);
28.         // 获取控件
29.         et_num = (EditText) findViewById(R.id.et_num);
30.         et_sms = (EditText) findViewById(R.id.et_sms);
31.         btn = (Button) findViewById(R.id.btn);
```

```
32.    /*
33.     * 给 Button 绑定点击事件
34.     * 这里通过一个匿名内部类来创建 OnClickListener 对象
35.     */
36.    btn.setOnClickListener(new OnClickListener() {
37.
38.        @Override
39.        public void onClick(View v) {
40.            // 获取数据
41.            String num = et_num.getText().toString().trim();
42.            String sms = et_sms.getText().toString().trim();
43.            /*
44.             * 对数据合法性进行校验
45.             * 这里的要求是如果电话号码或者短信内容有一个为空则就终止代码继续往下走
46.             * TextUtils 是 Android 提供的一个字符串操作工具类
47.             */
48.            if (TextUtils.isEmpty(num) || TextUtils.isEmpty(sms)) {
49.                /*
50.                 * Toast 是 Android 系统提供的工具类，
51.                 * 可以在手机屏幕上弹出提示信息并会自动消失
52.                 * 第一个参数 Context 上下文，MainActivity 继承了 Context，
53.                 * 因此其本身就是上下文对象
54.                 * 第二个参数 CharSequence 字符串文本内容
55.                 * 第三个参数 int 显示时长，只有两个常量，Toast.LENGTH_SHORT 和
56.                 * Toast.LENGTH_LONG
57.                 *
58.                 */
59.                Toast.makeText(MainActivity.this, " 电话 或 者 短 信 不 能 为 空 ",
Toast.LENGTH_SHORT).show();
60.                return;
61.            }
62.            /*
63.             * 发送短信是通过 API 提供的短信管理器实现的
64.             * 通过 SmsManager 的静态方法获取对象
65.             */
66.            SmsManager smsManager = SmsManager.getDefault();
67.            /*
68.             * 短信长度超过一定的限制后需要切割成多条分批发送
69.             * 一定要使用 SmsManager 对象提供的 divideMessage (String) 方法切割
70.             *
71.             */
72.            ArrayList<String> parts = smsManager.divideMessage(sms);
73.            /*
74.             * 发送短信
```

```

75.          * 第一个参数 String 目标手机号码
76.          * 第二个参数 String 短信中心号码，建议设置为 null，
77.          * 使用 sim 卡提供的默认短信中心
78.          * 第三个参数 ArrayList 短信内容
79.          * 第四个参数 ArrayList<PendingIntent>> 短信发送后如果发送成功了，
80.          * 那么回调该参数，通过延时意图和广播才能实现，这里设置为 null 即可
81.          * 第五个参数 ArrayList<PendingIntent>> 短信发送后如果被对方收到了，
82.          * 那么回调该参数。
83.          */
84.          smsManager.sendMultipartTextMessage(num, null, parts, null, null);
85.          //执行过发送后提示用户
86.          Toast.makeText(MainActivity.this, " 发 送 成 功 ",
Toast.LENGTH_SHORT).show();
87.      }
88.  });
89. }
90. }

```

#### 注意：

上面代码第 59 和 86 行使用了 MainActivity.this。在 Toast 中使用的参数类型是 Context，因为 Activity 正是 Context 的子类，但是这行代码是在内部类中使用的，如果直接用 this 那么他代表的是当前内部类对象，在内部类中如果想引用外部类对象就需要通过类名.this 调用。

### 1.5.3 添加权限

在 AndroidManifest.xml 中添加发送短信权限。

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

#### 注意：

在测试上面工程的时候需要启动一个新的模拟器，模拟器左上角的数字就代表他的电话号码。

## 1.6 点击事件的四种实现方式

按钮的点击事件有如下四种实现方式，前两种在 1.4 和 1.5 案例中均已实现，就不再赘述。

#### 一、通过内部类

#### 二、通过匿名内部类

#### 三、通过让当前 Activity 实现 OnClickListener 接口

以 1.4 案例来说，可以让当前的 MainActivity implements OnClickListener 接口，然后在 MainActivity 类中覆写 OnClickListener 接口中的 onClick 方法。在 onClick 方法中完成拨号功能。同时呢就不需要使用内部类了，而是给 Button 设置点击事件监听器是通过 btn.setOnClickListener(this)即可。这个应该很好理解，因为 this 代表当前 MainActivity 对象，MainActivity 又是 OnClickListener 的子类。

#### 四、通过布局文件中控件的属性

在编写布局文件时可以给 Button 控件添加 android:onClick="sendSMS"属性。其中 sendSMS 是自定义的方法名，见名知意即可。然后需要在 MainActivity.java 中声明一个 public void sendSMS(View view) 的方法，该方法权限修饰符必须为 public，形参必须为 View 类，在该方法中完成发送短信的核心功能。底

部的原理是通过反射来实现的。

## 1.7 Android 常用布局

Android 有 5 大布局，分别是 RelativeLayout、LinearLayout、FrameLayout、AbsoluteLayout、TableLayout。不过前 3 种布局才是最常用的布局，AbsoluteLayout 已经被 Google 废除，TableLayout 可以被 GridView 替代，因此也很少用。这里只介绍前 3 种，如果对后两种感兴趣可以自己去尝试研究。

### 1.7.1 相对布局

相对布局 RelativeLayout 允许子元素指定它们相对于其父元素或兄弟元素的位置，这是实际布局中最常用的布局方式之一。

通过编写如图 1-31 所示的布局来学习使用相对布局。



图 1-31 相对布局

#### 【文件 1-5】relativelayout.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent" >
5.
6.     <Button
7.         android:layout_width="wrap_content"
8.         android:layout_height="wrap_content"
9.         android:text="左上" />
```

```
10.
11.     <Button
12.         android:layout_width="wrap_content"
13.         android:layout_height="wrap_content"
14.         android:layout_alignParentRight="true"
15.         android:text="右上" />
16.
17.     <Button
18.         android:layout_width="wrap_content"
19.         android:layout_height="wrap_content"
20.         android:layout_alignParentBottom="true"
21.         android:text="左下" />
22.
23.     <Button
24.         android:layout_marginRight="10dp"
25.         android:layout_width="wrap_content"
26.         android:layout_height="wrap_content"
27.         android:layout_alignParentBottom="true"
28.         android:layout_alignParentRight="true"
29.         android:text="右下" />
30.
31.
32.     <Button
33.         android:id="@+id/btn"
34.         android:layout_width="wrap_content"
35.         android:layout_height="wrap_content"
36.         android:layout_centerInParent="true"
37.         android:text="居中" />
38.     <Button
39.         android:layout_above="@id/btn"
40.         android:layout_width="wrap_content"
41.         android:layout_height="wrap_content"
42.         android:layout_alignLeft="@id/btn"
43.         android:text="居上" />
44.
45.     <Button
46.         android:layout_below="@id/btn"
47.         android:layout_width="wrap_content"
48.         android:layout_height="wrap_content"
49.         android:layout_alignLeft="@id/btn"
50.         android:text="居下" />
51.
52.     <Button
53.         android:layout_toLeftOf="@id/btn"
54.         android:layout_alignTop="@id/btn"
```

```
55.         android:layout_width="wrap_content"
56.         android:layout_height="wrap_content"
57.         android:text="居左" />
58.     <Button
59.         android:layout_toRightOf="@id/btn"
60.         android:layout_alignTop="@id/btn"
61.         android:layout_width="wrap_content"
62.         android:layout_height="wrap_content"
63.         android:text="居右" />
64.
65.
66.
67. </RelativeLayout>
```

### 注意：

在上面布局中左上、左下、右上、右下使用的分别是 `android:layout_alignParentXxxx="true"` 属性，该属性可以设置当前控件跟父控件的对其方式。

居中使用的是 `android:layout_centerInParent="true"` 属性，该属性让该控件位于父控件的水平和垂直两个方向同时居中，如果只想让其位于父控件的一个方向居中则可以使用 `android:layout_centerHorizontal="true"` 或者 `android:layout_centerVertical="true"`。

`android:layout_toRightOf="@id/btn"` 该属性可以让当前控件位于指定控件的右侧，左侧雷同。如果想让该控件位于某控件的顶部则使用 `android:layout_above="@id/btn"` 底部使用 `android:layout_below="@id/btn"`。

`android:layout_alignTop="@id/btn"`、`android:layout_alignLeft="@id/btn"` 属性分别是让该控件相对于某控件顶部或者左侧对齐。注意：跟某个控件左侧对齐跟在某个控件左侧是不同的意思。

## 1.7.2 线性布局

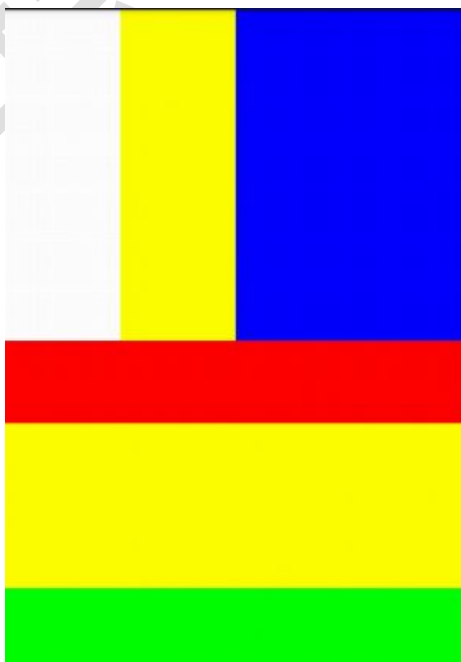




图 1-32 线性布局

线性布局 `LinearLayout` 是根据水平 `Horizontal` 或者垂直 `Vertical` 方向排列的布局，其最大的特点是可以给予控件按照权重分配空间。

通过编写如图 1-32 所示的布局来学习线性布局。

**【文件 1-6】** `linearlayout.xml`

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.
7.     <LinearLayout
8.         android:layout_width="match_parent"
9.         android:layout_height="0dp"
10.        android:layout_weight="4"
11.        android:orientation="horizontal" >
12.
13.        <TextView
14.            android:layout_width="0dp"
15.            android:layout_height="match_parent"
16.            android:layout_weight="1"
17.            android:background="#ffffff" />
18.
19.        <TextView
20.            android:layout_width="0sp"
21.            android:layout_height="match_parent"
22.            android:layout_weight="1"
23.            android:background="#ffff00" />
24.
25.        <TextView
26.            android:layout_width="0dp"
27.            android:layout_height="match_parent"
28.            android:layout_weight="2"
29.            android:background="#0000ff" />
30.    </LinearLayout>
31.
32.    <LinearLayout
33.        android:layout_width="match_parent"
34.        android:layout_height="0dp"
35.        android:layout_weight="4"
36.        android:orientation="vertical" >
37.
38.        <TextView
39.            android:layout_width="match_parent"
40.            android:layout_height="0dp"
```

```

41.         android:layout_weight="1"
42.         android:background="#ff0000" />
43.
44.     <TextView
45.         android:layout_width="match_parent"
46.         android:layout_height="0dp"
47.         android:layout_weight="2"
48.         android:background="#ffff00" />
49.
50.     <TextView
51.         android:layout_width="match_parent"
52.         android:layout_height="0dp"
53.         android:layout_weight="1"
54.         android:background="#00ff00" />
55. </LinearLayout>
56.
57. </LinearLayout>

```

#### 注意：

在上面布局中，将整体划分为上下两部分。上下两部分对应连个 LinearLayout，这两个 LinearLayout 的高度的权重设置为相同即可均分高度。

上面的 LinearLayout 是水平布局的，被水平分为 3 部分，第一部分和第二部分宽度相同，第三部分的宽度是第一个和第二部分之和，因此也使用水平方向的权重，给第三部分设置权重为 2，其他两个权重设置为 1 即可。

下面的 LinearLayout 是垂直布局的，被垂直分为 3 部分。第一部分和第三部分的高度相同，第二部分的高度是第一个和第三部分之和，因此也使用垂直方向的权重，给第一和第三部分权重设置为 1，第二部分权重设置为 2 即可。

### 1.7.3 帧布局

帧布局 FrameLayout 中的子视图总是被绘制到相对于屏幕的左上角上，所有添加到这个布局中的视图都是以层叠的方式显示，第一个添加到帧布局中的视图显示在最底层，最后一个被放在最顶层，上一层的视图会覆盖下一层的视图，类似于 html 中的 div。

通过编写如图 1-33 所示的布局来学习帧布局。

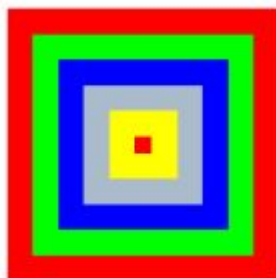


图 1-33 帧布局

【文件 1-7】 framelayout.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent" >
5.
6.     <TextView
7.         android:layout_width="160dp"
8.         android:layout_height="160dp"
9.         android:layout_gravity="center"
10.        android:background="#ff0000" />
11.
12.    <TextView
13.        android:layout_width="130dp"
14.        android:layout_height="130dp"
15.        android:layout_gravity="center"
16.        android:background="#00ff00" />
17.
18.    <TextView
19.        android:layout_width="100dp"
20.        android:layout_height="100dp"
21.        android:layout_gravity="center"
22.        android:background="#0000ff" />
23.
24.    <TextView
25.        android:layout_width="70dp"
26.        android:layout_height="70dp"
27.        android:layout_gravity="center"
28.        android:background="#aabbcc" />
29.
30.    <TextView
31.        android:layout_width="40dp"
32.        android:layout_height="40dp"
33.        android:layout_gravity="center"
34.        android:background="#ffff00" />
35.
36.    <TextView
37.        android:id="@+id/tv_6"
38.        android:layout_width="10dp"
39.        android:layout_height="10dp"
40.        android:layout_gravity="center"
41.        android:background="#ff0000" />
42.
43. </FrameLayout>
```

**注意：**

在上面的布局中所有的子控件都居中，然后多个子控件一层叠加一层就显示出了层层放大的效果。

## 1.8 Android 中的长度单位

### 1.8.1 px

pixels 的意思，是屏幕的物理像素点，与密度相关，密度大了，单位面积上的 px 会比较多。不推荐使用的单位。

### 1.8.2 dip 或 dp

Density independent pixels 设备无关像素，简称 dip 也叫 dp。一般情况下，在不同分辨率下都不会有缩放的感觉。在运行时，Android 系统会根据使用的屏幕的实际密度，透明地处理任何所需 dp 单位的缩放。推荐使用的单位。

**注意：**

dpi : dots per inch，直接来说就是一英寸多少个像素点。常见取值 120, 160, 240。一般称作像素密度，简称密度

density : 直接翻译的话也叫密度，但是他在 Android 中特指 dp 和 px 的比例关系。常见取值 1.5 , 1.0。

### 1.8.3 sp

与刻度无关的单位，同 dip/dp 相似，会根据用户的字体大小偏好来缩放，主要用于设置字体的大小。

### 1.8.4 dp 和 px 的区别

Android 官方定义 dip 等价于 160dpi 屏幕下的一个物理像素点，即 1dip=1px。转换公式如下：

$$\text{dip} = (\text{dpi} / (160 \text{ 像素/英寸})) \text{ px} = \text{density px};$$

举例来说，在 240 dpi 的屏幕上，1dip 等于 1.5px。

在 Android 的应用包 apk 中，系统会根据各个设备的具体情况引用相应的资源文件（注：不加任何标签的资源是各种分辨率情况下共用的）：

当屏幕 density=240 时，使用 hdpi 标签的资源；

当屏幕 density=160 时，使用 mdpi 标签的资源；

当屏幕 density=120 时，使用 ldpi 标签的资源。