Android 开发之旅——完整版

目录

Android 开发之旅：环境搭建及 HelloWorld

——工欲善其事必先利其器

引言

本系列适合 0 基础的人员，因为我就是从 0 开始的，此系列记录我步入 Android 开发的一些经验分享，望与君共勉！作为 Android 队伍中的一个新人的我，如果有什么不对的地方，还望不吝赐教。

在开始 Android 开发之旅启动之前，首先要搭建环境，然后创建一个简单的 HelloWorld。本文的主题如下：

- 1、环境搭建
  - 1.1、JDK 安装
  - 1.2、Eclipse 安装
  - 1.3、Android SDK 安装
  - 1.4、ADT 安装
  - 1.5、创建 AVD
- 2、HelloWorld

1、环境搭建

1.1、JDK 安装

如果你还没有 JDK 的话，可以去这里下载，接下来的工作就是安装提示一步一步走。设置环境变量步骤如下：

1. 我的电脑->属性->高级->环境变量->系统变量中添加以下环境变量：
2. JAVA_HOME 值为： D:\Program Files\Java\jdk1.6.0_18（**你安装 JDK 的目录**）

3. CLASSPATH 值
为：.;%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\bin;
4. Path: 在开始追加 %JAVA_HOME%\bin;
5. **NOTE：**前面四步设置环境变量对搭建 Android 开发环境不是必须的，可以跳过。

安装完成之后，可以在检查 JDK 是否安装成功。打开 cmd 窗口，输入 java –version 查看 JDK 的版本信息。出现类似下面的画面表示安装成功了：

图 1、验证 JDK 安装是否成功

1.2、Eclipse 安装

如果你还么有 Eclipse 的话，可以去这里下载，下载如下图所示的 Eclipse IDE for Java Developers（92M）的 win 32bit 版：

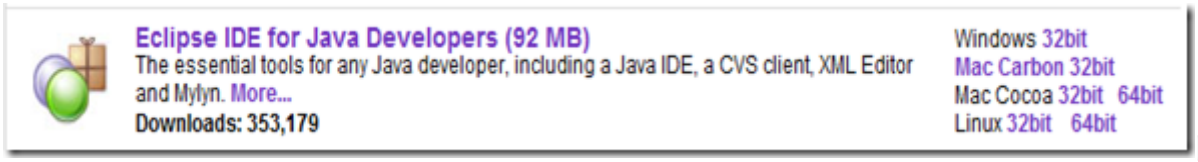


图 2、Eclipse 下载

解压之后即可使用。

1.3、Android SDK 安装

在 Android Developers 下载 android-sdk_r05-windows.zip，下载完成后解压到任意路径。

- 运行 SDK Setup.exe，点击 Available Packages。如果没有出现可安装的包，请点击 Settings，选中 Misc 中的"Force https://..."这项，再点击 Available Packages 。
- 选择希望安装的 SDK 及其文档或者其它包，点击 Installation Selected、Accept All、Install Accepted，开始下载安装所选包
- 在用户变量中新建 PATH 值为：Android SDK 中的 tools 绝对路径（本机为 D:\AndroidDevelop\android-sdk-windows\tools）。

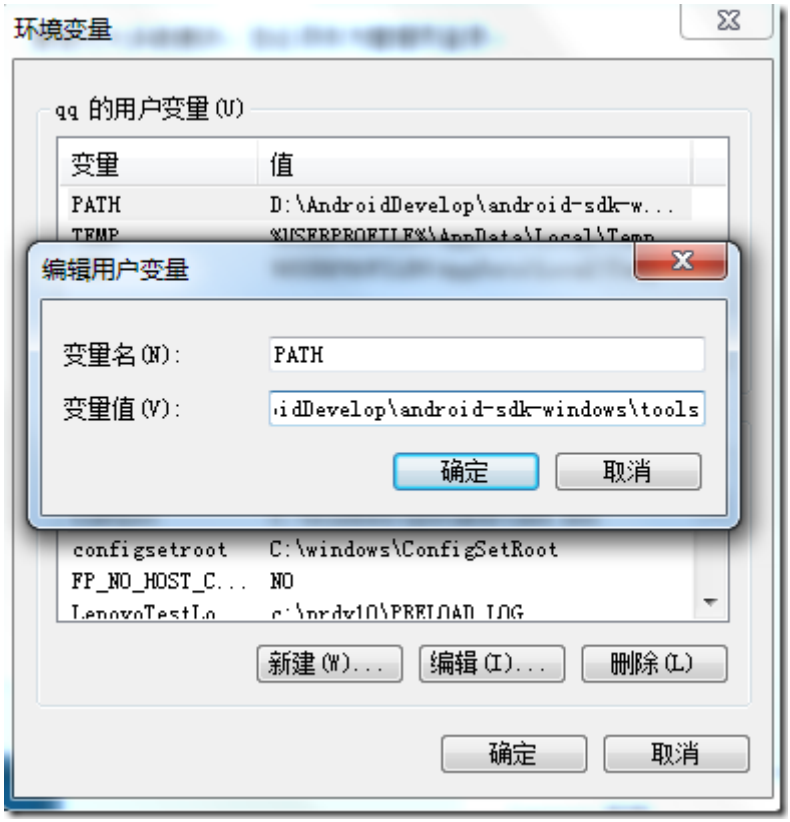


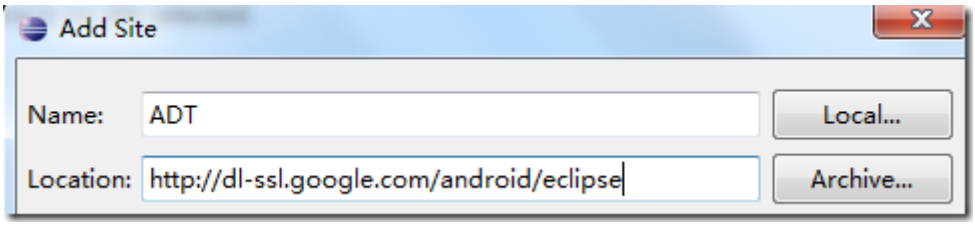
图 2、设置 Android SDK 的环境变量

“确定”后，重新启动计算机。重启计算机以后，进入 cmd 命令窗口，检查 SDK 是不是安装成功。运行 android –h 如果有类似以下的输出，表明安装成功：

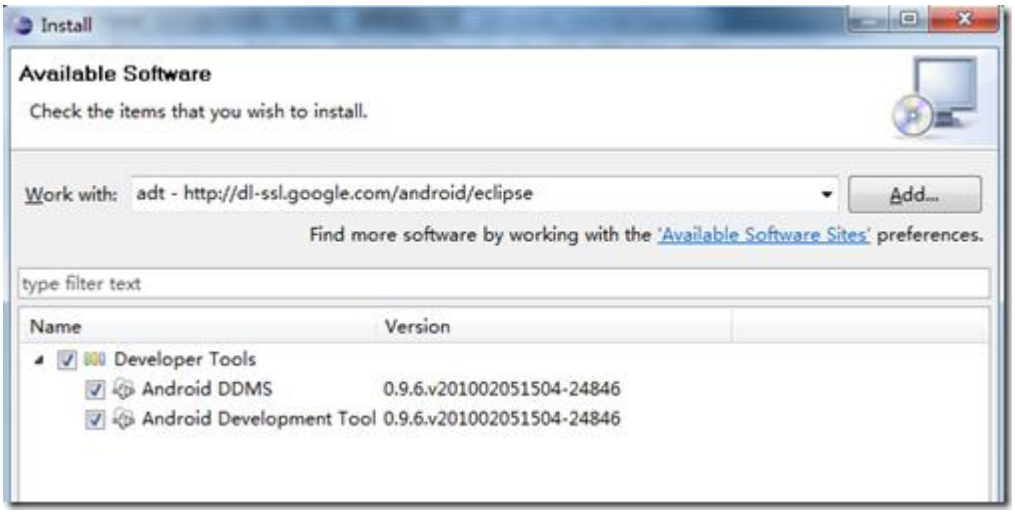
图 3、验证 Android SDK 是否安装成功

1.4、ADT 安装

- 打开 Eclipse IDE，进入菜单中的 "Help" -> "Install New Software"
- 点击 Add...按钮，弹出对话框要求输入 Name 和 Location：Name 自己随便取，Location 输入 http://dl-ssl.google.com/android/eclipse。如下图所示：



- 确定返回后，在 work with 后的下拉列表中选择我们刚才添加的 ADT，我们会看到下面出有 Developer Tools，展开它会有 Android DDMS 和 Android Development Tool，勾选他们。 如下图所示：



- 然后就是按提示一步一步 next。

完成之后：

- 选择 Window > Preferences...
- 在左边的面板选择 Android，然后在右侧点击 Browse...并选中 SDK 路径，本机为：D:\AndroidDevelop\android-sdk-windows
- 点击 Apply、OK。配置完成。

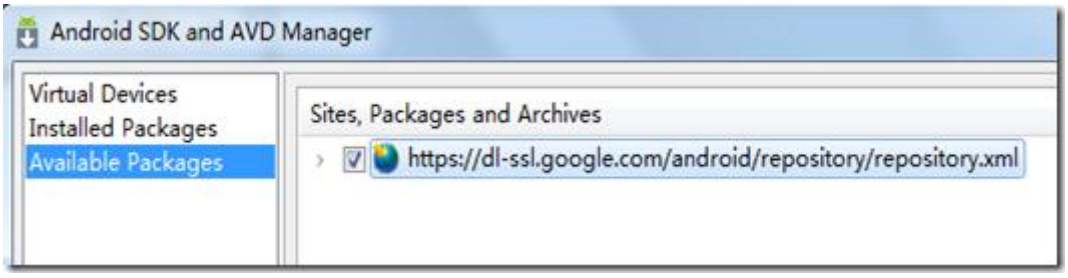
1.5、创建 AVD

为使 Android 应用程序可以在模拟器上运行，必须创建 AVD。

- 1、在 Eclipse 中。选择 Windows > Android SDK and AVD Manager
- 2、点击左侧面板的 Virtual Devices，再右侧点击 New
- 3、填入 Name，选择 Target 的 API，SD Card 大小任意，Skin 随便选，Hardware 目前保持默认值
- 4、点击 Create AVD 即可完成创建 AVD

注意：如果你点击左侧面板的 Virtual Devices，再右侧点击 New ，而 target 下拉列表没有可选项时，这时候你：

- 点击左侧面板的 Available Packages，在右侧勾选 https://dl-ssl.google.com/android/repository/repository.xml，如下图所示：

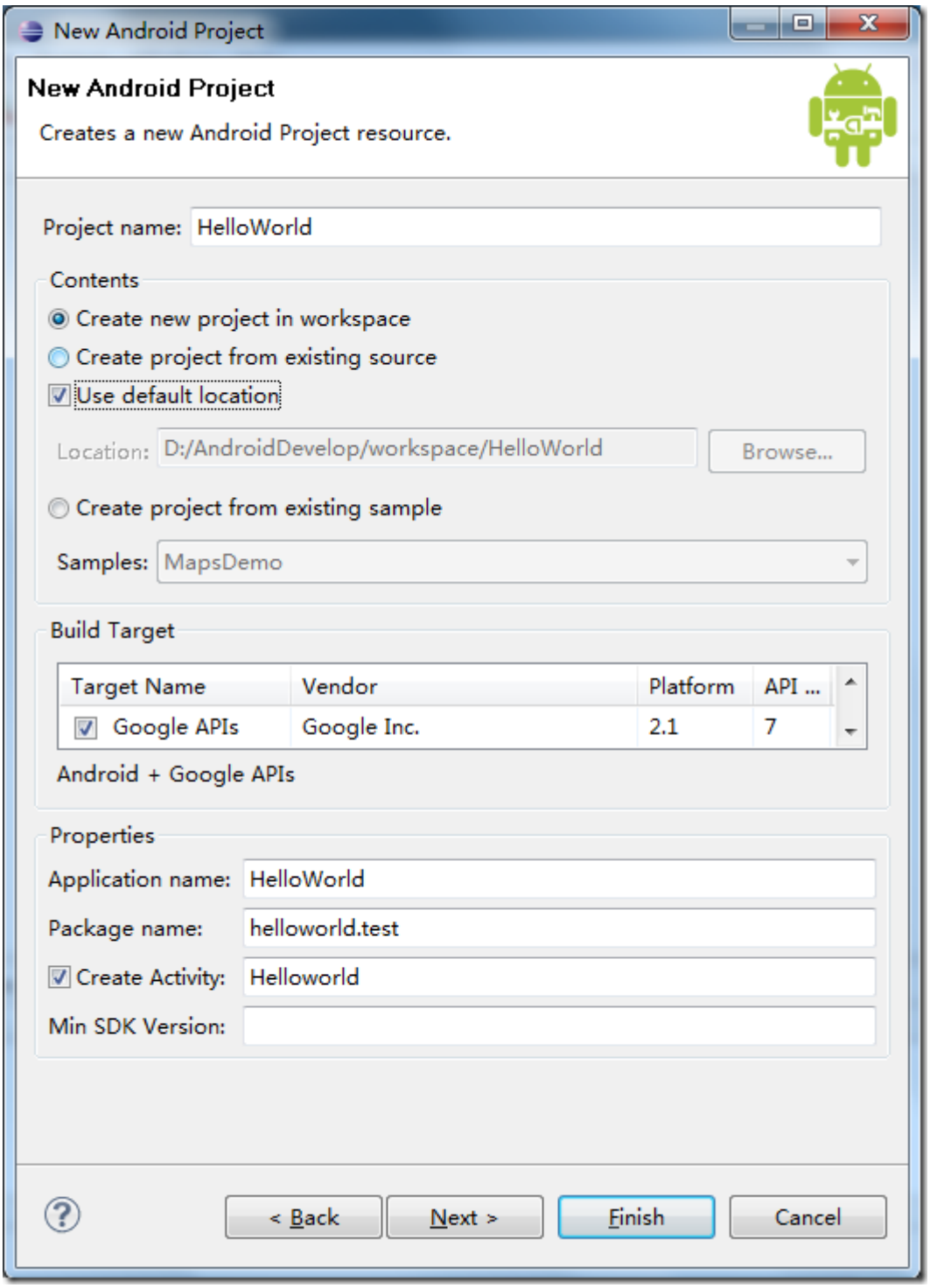


  - 然后点击 Install Selected 按钮，接下来就是按提示做就行了

要做这两步，原因是在 1.3、Android SDK 安装中没有安装一些必要的可用包（Available Packages）。

2、HelloWorld

- 通过 File -> New -> Project 菜单，建立新项目"Android Project"
- 然后填写必要的参数，如下图所示：（注意这里我勾选的是 Google APIs，你可以选你喜欢的，但你要创建相应的 AVD）

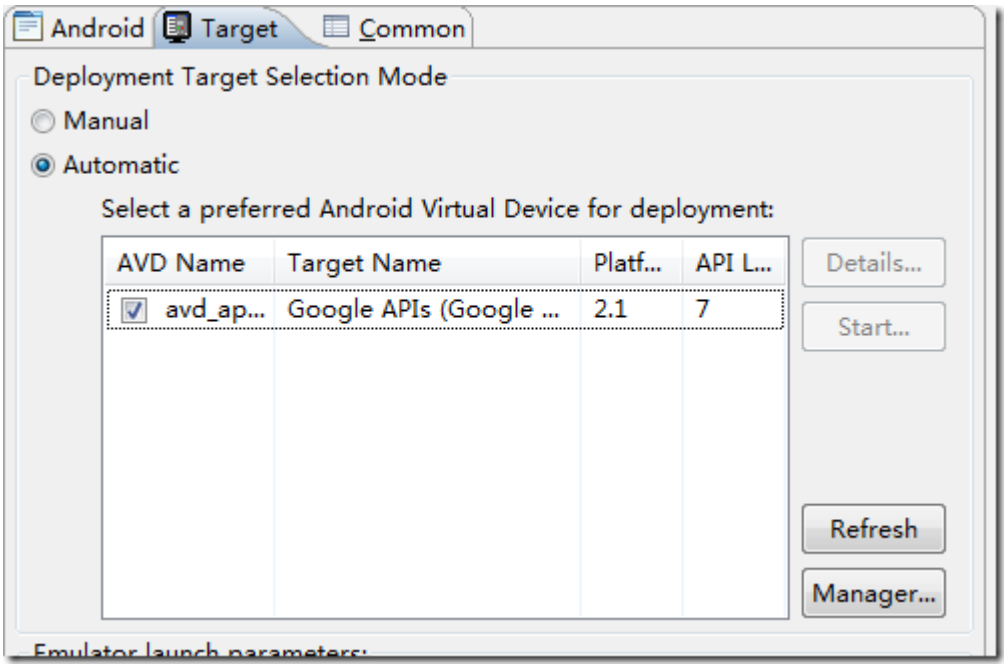


相关参数的说明：

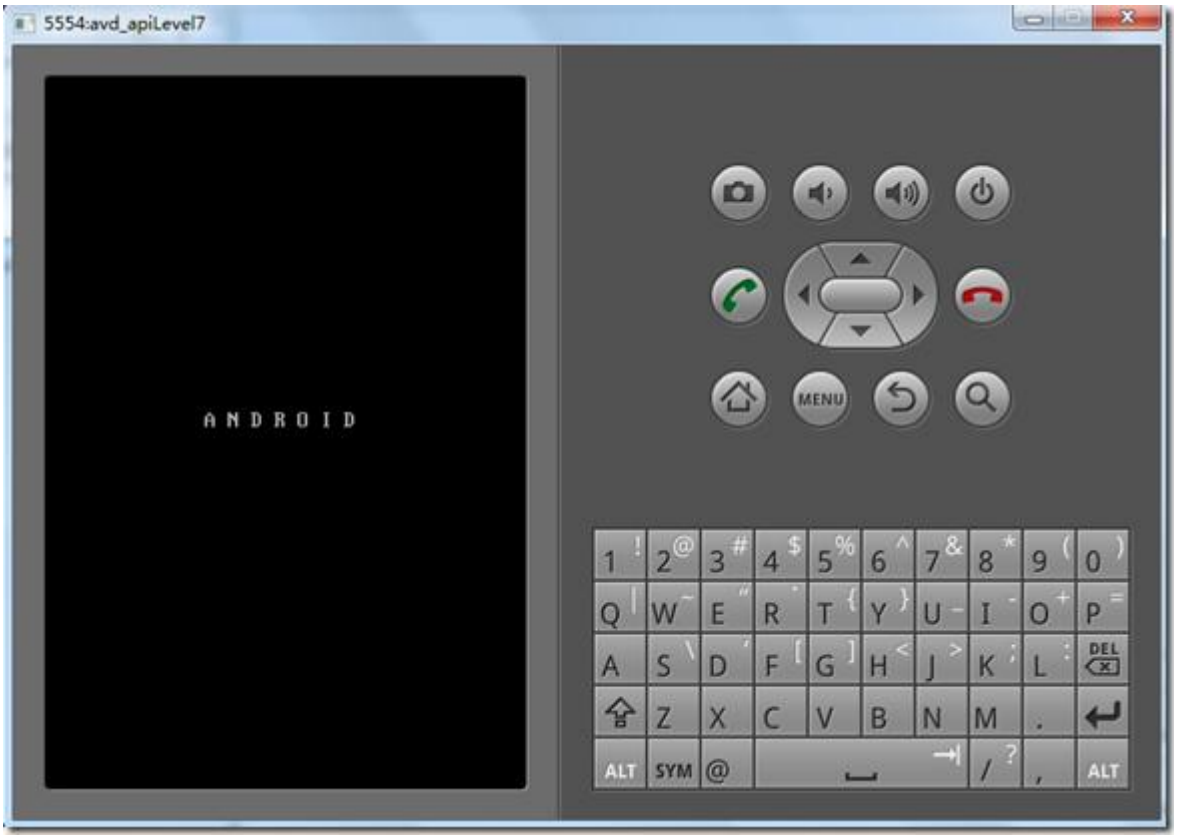
1. Project Name: 包含这个项目的文件夹的名称。
2. Package Name: 包名，遵循 JAVA 规范，用包名来区分不同的类是很重要的，我用的是 helloworld.test。

3. Activity Name: 这是项目的主类名，这个类将会是 Android 的 Activity 类的子类。一个 Activity 类是一个简单的启动程序和控制程序的类。它可以根据需要创建界面，但不是必须的。
4. Application Name: 一个易读的标题在你的应用程序上。
5. 在"选择栏"的 "Use default location" 选项，允许你选择一个已存在的项目。

- 点击 Finish 后，点击 Eclipse 的 Run 菜单选择 Run Configurations...
- 选择“Android Application”，点击在左上角（按钮像一张纸上有个“+”号）或者双击“Android Application”， 有个新的选项“New_configuration”（**可以改为我们喜欢的名字**）。
- 在右侧 Android 面板中点击 Browse...，选择 HelloWorld
- 在 Target 面板的 Automatic 中勾选相应的 AVD，**如果没有可用的 AVD 的话，你需要点击右下角的 Manager...，然后新建相应的 AVD**。如下图所示：



- 然后点 Run 按钮即可，运行成功的话会有 Android 的模拟器界面，如下图所示：



引言

前面 Android 开发之旅：环境搭建及 HelloWorld，我们介绍了如何搭建 Android 开发环境及简单地建立一个 HelloWorld 项目，本篇将通过 HelloWorld 项目来介绍 Android 项目的目录结构。本文的主要主题如下：

- 1、HelloWorld 项目的目录结构
  - 1.1、src 文件夹
  - 1.2、gen 文件夹
  - 1.3、Android 2.1 文件夹
  - 1.4、assets
  - 1.5、res 文件夹
  - 1.6、AndroidManifest.xml
  - 1.7、default.properties

1、HelloWorld 项目的目录结构

（这个 HelloWorld 项目是基于 Android 2.1 的）在 Eclipse 的左侧展开 HelloWorld 项目，可以看到如下图的目录结构：

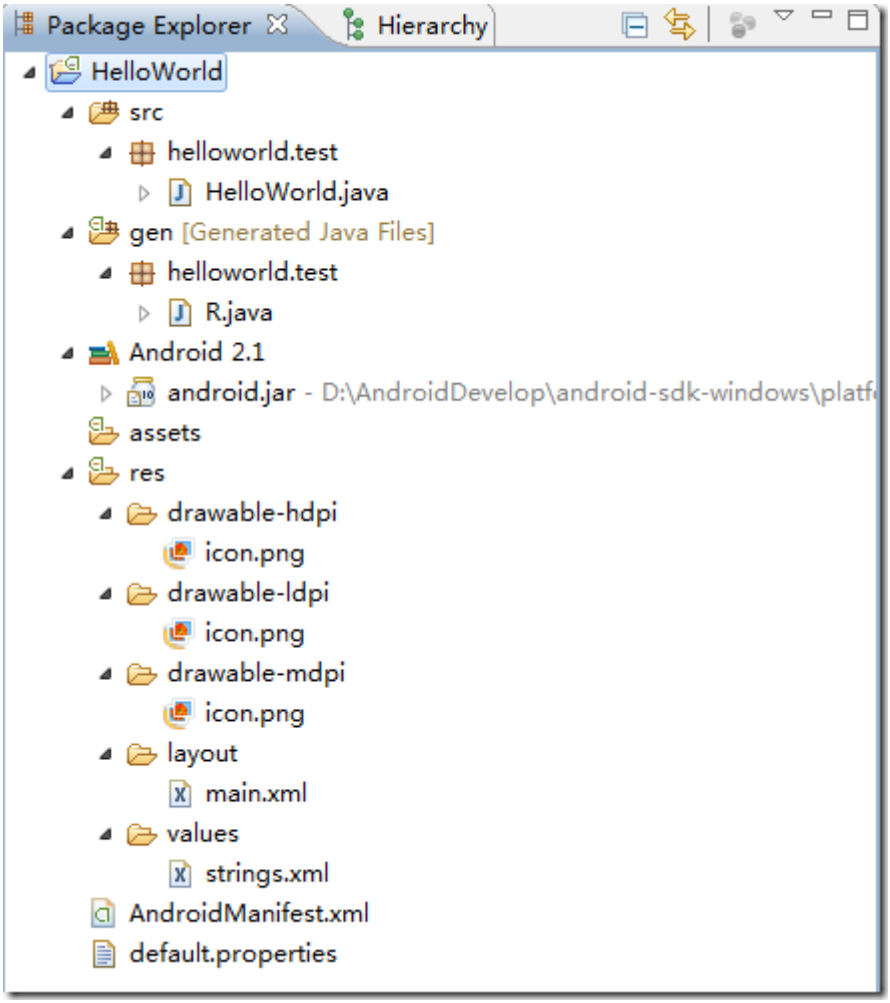


图 1、HelloWorld 项目目录结构

下面将分节介绍上面的各级目录结构。

1.1、src 文件夹

顾名思义（src, source code）该文件夹是放项目的源代码的。打开 HelloWorld.java 文件会看到如下代码：

```
HelloWorld.java
```

可以知道：我们新建一个简单的 HelloWorld 项目，系统为我们生成了一个 HelloWorld.java 文件。他导入了两个类 android.app.Activity 和 android.os.Bundle，HelloWorld 类继承自 Activity 且重写了 onCreate 方法。

以下说明针对没有学过 Java 或者 Java 基础薄弱的人

@Override

在重写父类的 onCreate 时，在方法前面加上@Override 系统可以帮你检查方法的正确性。例如，public void onCreate(Bundle savedInstanceState){.....}这种写法是正确的，如果你写成 public void oncreate(Bundle savedInstanceState){.....}这样编译器回报如下错误——The method oncreate(Bundle) of type HelloWorld must override or implement a supertype method，以确保你正确重写 onCreate 方法。（因为 oncreate 应该为 onCreate）

而如果你不加@Override，则编译器将不会检测出错误，而是会认为你新定义了一个方法 oncreate。

android.app.Activity 类：因为几乎所有的活动（activities）都是与用户交互的，所以 Activity 类关注创建窗口，你可以用方法 setContentView(View)将自己的 UI 放到里面。然而活动通常以全屏的方式展示给用户，也可以以浮动窗口或嵌入在另外一个活动中。有两个方法是几乎所有的 Activity 子类都实现的：

1. onCreate(Bundle)：初始化你的活动（Activity），比如完成一些图形的绘制。最重要的是，在这个方法里你通常将用布局资源（layout resource）调用 setContentView(int)方法定义你的 UI，和用 findViewById(int)在你的 UI 中检索你需要编程地交互的小部件（widgets）。setContentView 指定由哪个文件指定布局（main.xml），可以将这个界面显示出来，然后我们进行相关操作，我们的操作会被包装成为一个意图，然后这个意图对应有相关的 activity 进行处理。
2. onPause()：处理当离开你的活动时要做的事情。最重要的是，用户做的所有改变应该在这里提交（通常 ContentProvider 保存数据）。

更多的关于 Activity 类的详细信息此系列以后的文章将做介绍，如果你想了解更多请参阅相关文档。android.os.Bundle 类：从字符串值映射各种可打包的（Parcelable）类型（Bundle 单词就是捆绑的意思，所有这个类很好理解和记忆）。如该类提供了公有方法——public boolean containKey(String key)，如果给定的 key 包含在 Bundle 的映射中返回 true，否则返回 false。该类实现了 Parceable 和 Cloneable 接口，所以它具有这两者的特性。

1.2、gen 文件夹

该文件夹下面有个 R.java 文件，R.java 是在建立项目时自动生成的，这个文件是只读模式的，不能更改。R.java 文件中定义了一个类——R，R 类中包含很多静态类，且静态类的名字都与 res 中的一个名字对应，即 R 类定义该项目所有资源的索引。看我们的 HelloWorld 项目是不是如此，如下图：

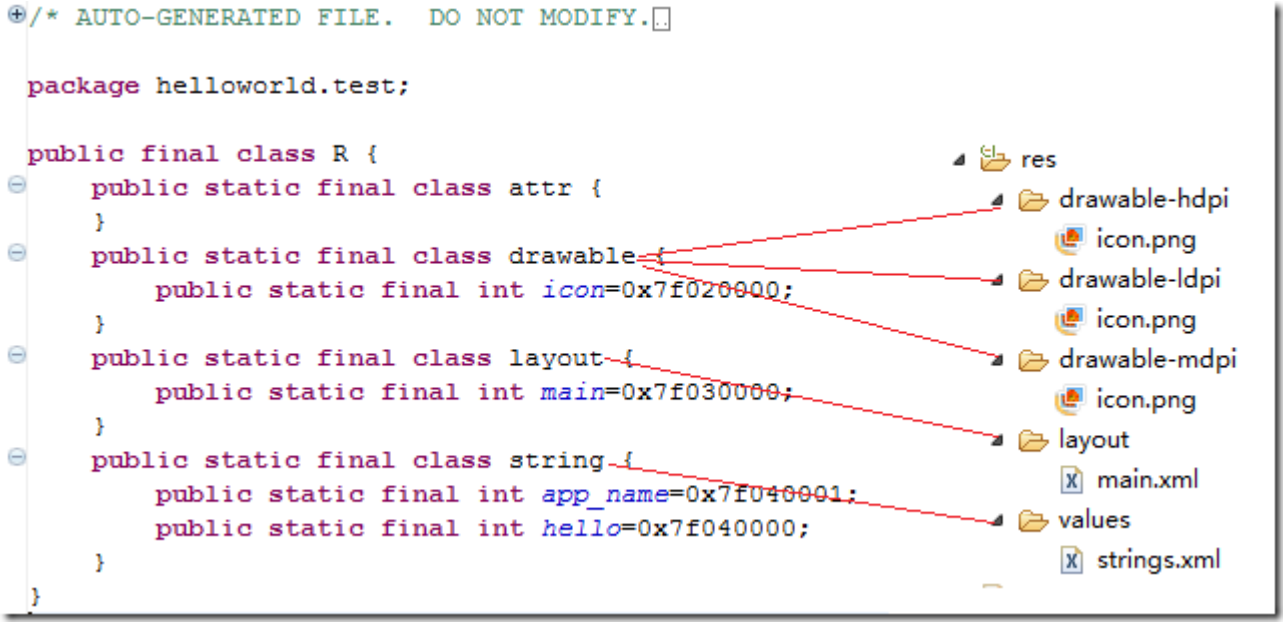


图 2、R.java 对应 res

通过 R.java 我们可以很快地查找我们需要的资源，另外编绎器也会检查 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绎进软件中，这样可以减少应用在手机占用的空间。

1.3、Android 2.1 文件夹

该文件夹下包含 android.jar 文件，这是一个 Java 归档文件，其中包含构建应用程序所需的所有的 Android SDK 库（如 Views、Controls）和 APIs。通过 android.jar 将自己的应用程序绑定到 Android SDK 和 Android Emulator，这允许你使用所有 Android 的库和包，且使你的应用程序在适当的环境中调试。例如上面的 HelloWorld.java 源文件中的：

import android.app.Activity;

import android.os.Bundle;

这里两行代码就是从 android.jar 导入包。

1.4、assets

包含应用系统需要使用到的诸如 mp3、视频类的文件。

1.5、res 文件夹

资源目录，包含你项目中的资源文件并将编译进应用程序。向此目录添加资源时，会被 R.java 自动记录。新建一个项目，res 目录下会有三个子目录：drawabel、layout、values。

- drawabel-?dpi：包含一些你的应用程序可以用的图标文件(*.png、*.jpg)
- layout：界面布局文件(main.xml)与 WEB 应用中的 HTML 类同，没修改过的 main.xml 文件如下（HelloWorld 的就没有修改过）：

```
main.xml
```

- values：软件上所需要显示的各种文字。可以存放多个*.xml 文件，还可以存放不同类型的数据。比如 arrays.xml、colors.xml、dimens.xml、styles.xml

1.6、AndroidManifest.xml

项目的总配置文件，记录应用中所使用的各种组件。这个文件列出了应用程序所提供的功能，在这个文件中，你可以指定应用程序使用到的服务(如电话服务、互联网服务、短信服务、GPS 服务等等)。另外当你新添加一个 Activity 的时候，也需要在这个文件中进行相应配置，只有配置好后，才能调用此 Activity。AndroidManifest.xml 将包含如下设置：application permissions、Activities、intent filters 等。

如果你跟我一样是 ASP.NET 出生或者学过，你会发现 AndroidManifest.xml 跟 web.config 文件很像，可以把它类同于 web.config 文件理解。

如果你不是，你可以这样理解——众所周知 xml 是一种数据交换格式，AndroidManifest.xml 就是用来存储一些数据的，只不过这些数据时关于 android 项目的配置数据。

HelloWorld 项目的 AndroidManifest.xml 如下所示：

```
AndroidManifest.xml
```

关于 AndroidManifest.xml 现在就讲这么多，此系列后面的文章将单独详细介绍。

1.7、default.properties

记录项目中所需要的环境信息，比如 Android 的版本等。 HelloWorld 的 default.properties 文件代码如下所示，代码中的注释已经把 default.properties 解释得很清楚了：

```
default.properties
```

引言

通过前面两篇：

- Android 开发之旅：环境搭建及 HelloWorld
- Android 开发之旅：HelloWorld 项目的目录结构

我们对 android 有了个大致的了解，知道如何搭建 android 的环境及简单地写一个 HelloWorld 程序，而且知道一个 android 项目包括哪些文件夹和文件及相应的作用。本篇将站在顶级的高度——架构，来看 android。我开篇就说了，这个系列适合 0 基础的人且我也是从 0 开始按照这个步骤来学的，谈架构是不是有点螳臂挡车，自不量力呢？我觉得其实不然，如果一开始就对整个 android 的架构了然于胸，就不会误入歧途，能够很好地把握全局。本文的主题如下：

- 1、架构图直观
- 2、架构详解
  - 2.1、Linux Kernel
  - 2.1、Android Runtime
  - 2.3、Libraries
  - 2.4、Application Framework
  - 2.5、Applications
- 3、总结

1、架构图直观

下面这张图展示了 Android 系统的主要组成部分：

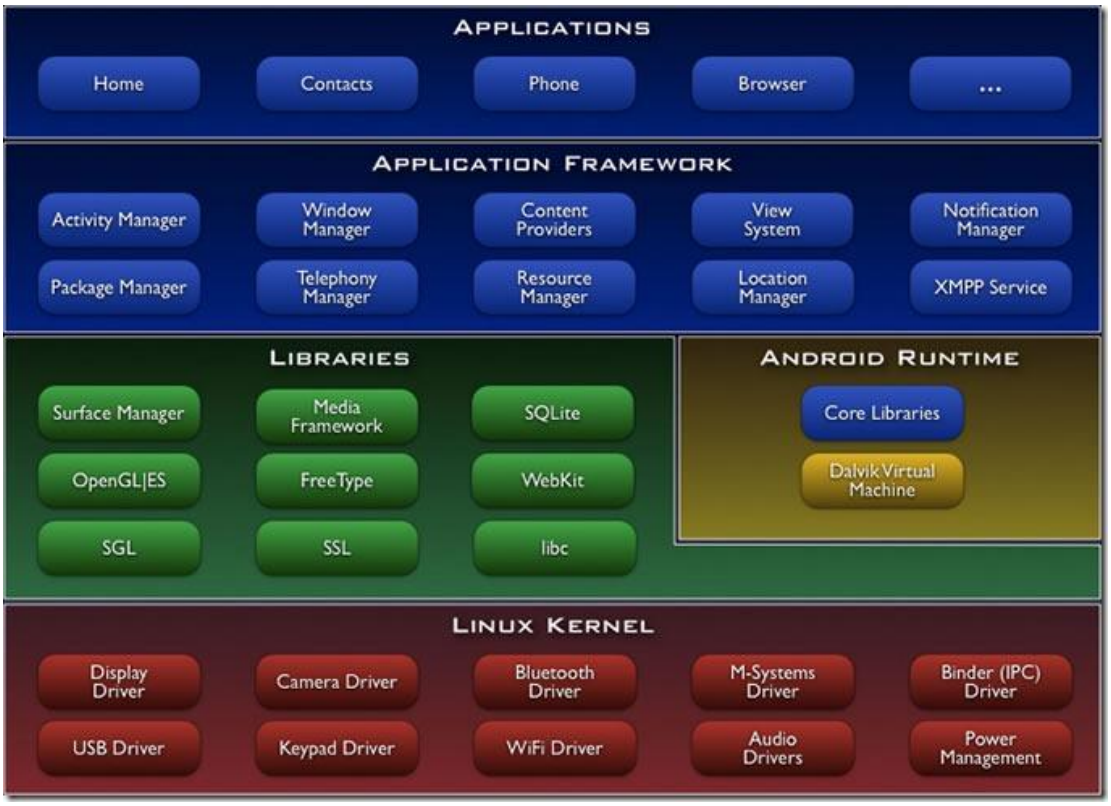


图 1、Android 系统架构（来源于：android sdk）

可以很明显看出，Android 系统架构由 5 部分组成，分别是：Linux Kernel、Android Runtime、Libraries、Application Framework、Applications。第二部分将详细介绍这 5 个部分。

2、架构详解

现在我们拿起手术刀来剖析各个部分。其实这部分 SDK 文档已经帮我们做得很好了，我们要做的就是拿来主义，然后再加上自己理解。下面自底向上分析各层。

2.1、Linux Kernel

Android 基于 Linux 2.6 提供核心系统服务，例如：安全、内存管理、进程管理、网络堆栈、驱动模型。Linux Kernel 也作为硬件和软件之间的抽象层，它隐藏具体硬件细节而为上层提供统一的服务。

如果你学过计算机网络知道 OSI/RM，就会知道分层的好处就是使用下层提供的服务而为上层提供统一的服务，屏蔽本层及以下层的差异，当本层及以下层发生了变化不会影响到上层。也就是说**各层各司其职，各层提供固定的 SAP（Service Access Point）**，专业点可以说是**高内聚、低耦合。**

如果你只是做应用开发，就不需要深入了解 Linux Kernel 层。

2.2、Android Runtime

Android 包含一个核心库的集合，提供大部分在 Java 编程语言核心类库中可用的功能。每一个 Android 应用程序是 Dalvik 虚拟机中的实例，运行在他们自己的进程中。Dalvik 虚拟机设计成，在一个设备可以高效地运行多个虚拟机。Dalvik 虚拟机可执行文件格式是.dex，dex 格式是专为 Dalvik 设计的一种压缩格式，适合内存和处理器速度有限的系统。

大多数虚拟机包括 JVM 都是基于栈的，而 Dalvik 虚拟机则是基于寄存器的。两种架构各有优劣，一般而言，基于栈的机器需要更多指令，而基于寄存器的机器指令更大。dx 是一套工具，可以将 Java .class 转换成 .dex 格式。一个 dex 文件通常会有多个.class。由于 dex 有時必须进行最佳化，会使文件大小增加 1-4 倍，以 ODEX 结尾。

Dalvik 虚拟机依赖于 Linux 内核提供基本功能，如线程和底层内存管理。

2.3、Libraries

Android 包含一个 C/C++库的集合，供 Android 系统的各个组件使用。这些功能通过 Android 的应用程序框架（application framework）暴露给开发者。下面列出一些核心库：

- **系统 C 库**——标准 C 系统库（libc）的 BSD 衍生，调整为基于嵌入式 Linux 设备
- **媒体库**——基于 PacketVideo 的 OpenCORE。这些库支持播放和录制许多流行的音频和视频格式，以及静态图像文件，包括 MPEG4、 H.264、 MP3、 AAC、 AMR、JPG、 PNG
- **界面管理**——管理访问显示子系统和无缝组合多个应用程序的二维和三维图形层
- **LibWebCore**——新式的 Web 浏览器引擎,驱动 Android 浏览器和内嵌的 web 视图
- **SGL**——基本的 2D 图形引擎
- **3D 库**——基于 OpenGL ES 1.0 APIs 的实现。库使用硬件 3D 加速或包含高度优化的 3D 软件光栅
- **FreeType** ——位图和矢量字体渲染
- **SQLite** ——所有应用程序都可以使用的强大而轻量级的关系数据库引擎

2.4、Application Framework

通过提供开放的开发平台，Android 使开发者能够编制极其丰富和新颖的应用程序。开发者可以自由地利用设备硬件优势、访问位置信息、运行后台服务、设置闹钟、向状态栏添加通知等等，很多很多。开发者可以完全使用核心应用程序所使用的框架 APIs。应用程序的体系结构旨在简化组件的重用，任何应用程序都能发布他的功能且任何其他应用程序可以使用这些功能（需要服从框架执行的安全限制）。这一机制允许用户替换组件。

所有的应用程序其实是一组服务和系统，包括：

- **视图（View）**——丰富的、可扩展的视图集合，可用于构建一个应用程序。包括包括列表、网格、文本框、按钮，甚至是内嵌的网页浏览器
- **内容提供者（Content Providers）**——使应用程序能访问其他应用程序（如通讯录）的数据，或共享自己的数据
- **资源管理器（Resource Manager）**——提供访问非代码资源，如本地化字符串、图形和布局文件
- **通知管理器（Notification Manager）**——使所有的应用程序能够在状态栏显示自定义警告
- **活动管理器（Activity Manager）**——管理应用程序生命周期,提供通用的导航回退功能

2.5、Applications

Android 装配一个核心应用程序集合，包括电子邮件客户端、SMS 程序、日历、地图、浏览器、联系人和其他设置。所有应用程序都是用 Java 编程语言写的。更加丰富的应用程序有待我们去开发！

3、总结

从上面我们知道 Android 的架构是分层的，非常清晰，分工很明确。Android 本身是一套软件堆叠(Software Stack)，或称为「软件叠层架构」，叠层主要分成三层：操作系统、中间件、应用程序。从上面我们也看到了开源的力量，一个个熟悉的开源软件在这里贡献了自己的一份力量。

现在我们对 android 的系统架构有了一个整体上的了解，我将用一个例子来深入体会一下，但是考虑到此系列希望 0 基础的人也能看懂，在介绍例子之前将介绍一下 **Android 应用程序的原理及一些术语**，可能需要几篇来介绍。敬请关注！

——成功属于耐得住寂寞的人，接下来几篇将讲述 Android 应用程序的原理及术语，可能会比较枯燥。如果能够静下心来看，相信成功将属于你。

引言

为了后面的例子做准备，本篇及接下来几篇将介绍 Android 应用程序的原理及术语，这些也是作为一个 Android 的开发人员必须要了解，且深刻理解的东西。本篇的主题如下：

- 1、应用程序基础
- 2、应用程序组件
  - 2.1、活动（Activities）
  - 2.2、服务（Services）

- 2.3、广播接收者（Broadcast receivers）
- 2.4、内容提供者（Content providers）

因为这些内容比较理论，且没有用例子来说明，看上去会比较枯燥，我就把这几篇写得算比较短，方便大家吸收。

1、应用程序基础

Android 应用程序是用 Java 编程语言写的。编译后的 Java 代码——包括应用程序要求的任何数据和资源文件，通过 aapt 工具捆绑成一个 Android 包，归档文件以.apk 为后缀。这个文件是分发应用程序和安装到移动设备的中介或工具，用户下载这个文件到他们的设备上。一个.apk 文件中的所有代码被认为是一个应用程序。

aapt：

aapt 是 Android Asset Packaging Tool 的首字母缩写，这个工具包含在 SDK 的 tools/目录下。查看、创建、更新与 zip 兼容的归档文件（zip、jar、apk）。它也能将资源文件编译成二进制包。

尽管你可能不会经常直接使用 appt，但是构建脚本（build scripts）和 IDE 插件会使用这个工具打包 apk 文件，构成一个 Android 应用程序。

如需更详细的使用细节，打开一个终端，进入 tools/目录下，运行命令：

- Linux 或 Mac 操作系统：./aapt
- Windows：aapt.exe

注意：tools/目录是指 android SDK 目录下的/platforms/android-X/tools/

在许多方面，每个 Android 应用程序生活在它自己的世界：

- 默认情况下，每一个应用程序运行在它自己的 Linux 进程中。当应用程序中的任何代码需要执行时，Android 将启动进程；当它不在需要和系统资源被其他应用程序请求时，Android 将关闭进程。
- 每个应用程序都有他自己的 Java 虚拟机（VM），因此应用程序代码独立于其他所有应用程序的代码运行。
- 默认情况下，每个应用程序分配一个唯一的 Linux 用户的 ID。权限设置为每个应用程序的文件仅对用户和应用程序本身可见——虽然也有一些方法可以暴露他们给其他应用程序。

有可能设置两个应用程序共享一个用户 ID，这种情况下，他们能够看到对方的文件。为了节省系统资源，具有相同 ID 的应用程序也可以安排在同一个 Linux 进程中，共享同一个 VM。

2、应用程序组件

Android 的一个主要特点是，一个应用程序可以利用其他应用程序的元素（假设这些应用程序允许的话）。例如，如果你的应用程序需要显示一个图像的滚动列表，且其他应用程序已经开发了一个合适的滚动条并可以提供给别的应用程序用，你可以调用这个滚动条来工作，而不用自己开发一个。你的应用程序不用并入其他应用程序的代码或链接到它。相反，当需求产生时它只是启动其他应用程序块。

对于这个工作，当应用程序的任何部分被请求时，系统必须能够启动一个应用程序的进程，并实例化该部分的 Java 对象。因此，不像其他大多数系统的应用程序，Android 应用程序没有一个单一的入口点（例如，没有 main()函数）。相反，**系统能够实例化和运行需要几个必要的组件**。有四种类型的组件：

1. 活动（Activities）
2. 服务（Services）
3. 广播接收者（Broadcast receivers）
4. 内容提供者（Content providers）

然而，并不是所有的应用程序都必须包含上面的四个部分，你的应用程序可以由上面的一个或几个来组建。当你决定使用以上哪些组件来构建 Android 应用程序时，你应该将它们列在 AndroidManifest.xml 文件中，在这个文件中你可以声明应用程序组件以及它们的特性和要求。关于 AndroidManifest.xml 在 Android 开发之旅：HelloWorld 项目的目录结构的 1.6、AndroidManifest.xml 简单介绍了一下，你可以参考一下，下篇也将介绍它。

2.1、活动（Activities）

一个活动表示一个可视化的用户界面，关注一个用户从事的事件。例如，一个活动可能表示一个用户可选择的菜单项列表，或者可能显示照片连同它的标题。一个文本短信应用程序可能有一个活动，显示联

系人的名单发送信息；第二个活动，写信息给选定的联系人；其他活动，重新查看旧信息或更改设置。虽然他们一起工作形成一个整体的用户界面，但是每个活动是独立于其他活动的。每一个都是作为 Activity 基类的一个子类的实现。

android.app.Activity 类：因为几乎所有的活动（activities）都是与用户交互的，所以 Activity 类关注创建窗口，你可以用方法 setContentView(View)将自己的 UI 放到里面。然而活动通常以全屏的方式展示给用户，也可以以浮动窗口或嵌入在另外一个活动中。有两个方法是几乎所有的 Activity 子类都实现的：

1. onCreate(Bundle)：初始化你的活动（Activity），比如完成一些图形的绘制。最重要的是，在这个方法里你通常将用布局资源（layout resource）调用 setContentView(int)方法定义你的 UI，和用 findViewById(int)在你的 UI 中检索你需要编程地交互的小部件（widgets）。setContentView 指定由哪个文件指定布局（main.xml），可以将这个界面显示出来，然后我们进行相关操作，我们的操作会被包装成为一个意图（Intent），然后这个意图对应有相关的 activity 进行处理。
2. onPause()：处理当离开你的活动时要做的事情。最重要的是，用户做的所有改变应该在这里提交（通常 ContentProvider 保存数据）。

一个应用程序可能只包含一个活动，或者像刚才提到的短信应用，它可能包含几个活动。这些活动是什么，以及有多少，当然这取决于它的应用和设计。一般来讲，当应用程序被启动时，被标记为第一个的活动应该展示给用户。从一个活动移动到另一个活动由当前的活动完成开始下一个。

每一个活动都有一个默认的窗口。一般来讲，窗口会填满整个屏幕，但是它可能比屏幕小或浮在其他窗口上。一个活动还可以使用额外的窗口——例如弹出式对话框，或当一用户选择屏幕上一个特定的项时一个窗口显示给用户重要的信息。

窗口的可视内容是由继承自 View 基类的一个分层的视图—对象提供。每个视图控件是窗口内的一个特定的矩形空间。父视图包含和组织子女视图的布局。叶子视图（在分层的底层）绘制的矩形直接控制和响应用户的操作。因此，一个视图是活动与用户交互发生的地方。例如，一个视图可能显示一个小的图片和当用户点击图片时发起一个行为。Android 有一些现成的视图你可以使用，包括按钮（buttons）、文本域（text fields）、滚动条（scroll bars）、菜单项（menu items）、复选框（check boxes）等等。通过 Activity.setContentView() 方法放置一个视图层次在一个活动窗口中。内容视图（*content view*）是层次结构的根视图对象。层次结构如下图所示：

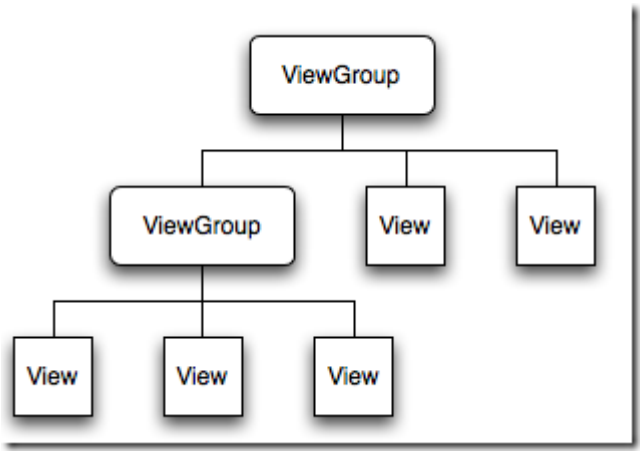


图 1、视图的层次结构

Activity.setContentView() 方法：

public void setContentView (int layoutResID)：根据布局资源设置活动的界面。 资源将被夸大，添加布局资源文件中所有的最高层的视图（ top-level views ）到活动.

2.2、 服务（Services）

一个服务没有一个可视化用户界面，而是在后台无期限地运行。例如一个服务可能是播放背景音乐而用户做其他一些事情，或者它可能从网络获取数据，或计算一些东西并提供结果给需要的活动（activities）。每个服务都继承自 Service 基类。

每个服务类在 AndroidManifest.xml 中有相应的<service>声明。服务可以通过 Context.startService()和 Context.bindService()启动。

一个典型的例子是一个媒体播放器播放一个播放列表中的歌曲。该播放器应用程序将可能有一个或多个活动（activities），允许用户选择歌曲和开始播放。然而，音乐播放本身不会被一个活动处理，因为用户希望保持音乐继续播放，当用户离开播放器去做其他事情时。为了保持音乐继续播放，媒体播放器活动可以启动一个服务运行在后台。系统将保持音乐播放服务运行，甚至媒体播放器离开屏幕时。
可以连接到（绑定到）一个持续运行的服务（并启动服务，如果它尚未运行）。连接之后，你可以通过服务暴露的接口与服务交流。对于音乐服务，这个接口可以允许用户暂停、倒带、停止和重新播放。
像活动（activities）和其他组件一样，服务（services）运行在应用程序进程中的主线程中。因此，他们将不会阻止其他组件或用户界面，他们往往产生其他一些耗时的任务（如音乐播放）。

2.3、广播接收者（Broadcast receivers）

一个广播接收者是这样一个组件，它不做什么事，仅是接受广播公告并作出相应的反应。许多广播源自于系统代码，例如公告时区的改变、电池电量低、已采取图片、用户改变了语言偏好。应用程序也可以发起广播，例如为了他其他程序知道某些数据已经下载到设备且他们可以使用这些数据。
一个应用程序可以有任意数量的广播接收者去反应任何它认为重要的公告。所有的接受者继承自 BroadcastReceiver 基类。
BroadcastReceiver 类：
是接受 sendBroadcast()发送的意图（intents）的基类。可以用 Context.registerReceiver()动态地注册这个类的实例，或者通过 AndroidManifest.xml 中<receiver>标签静态发布。**注意：如果你在 Activity.onResume() 注册一个接受者，你应该在 Activity.onPause()注销它。因为当暂停时你不会收到意图，注销它将削减不必要的系统开销。不要在 Activity.onSaveInstanceState()中注销它，因为它将不会被调用，如果用户移动到先前的堆栈。**
有两种主要的可接受广播类型：

1. **正常广播**（由 Context.sendBroadcast 发送）是完全异步的。所有的广播接收者以无序方式运行，往往在同一时间接收。这样效率较高，但是意味着接受者不能使用结果或终止广播数据传播。
2. **有序广播**（由 Context.sendOrderedBroadcast 发送）一次传递给一个接收者。由于每个接收者依次执行，因此它可以传播到下一个接收器，也可以完全终止传播以便他不会传递给其他接收者。接收者的运行顺序可由匹配的意图过滤器（intent-filter）的 android:priority 属性控制。

广播接收者不显示一个用户界面。然而，它们启动一个活动去响应收到的信息，或者他们可能使用 NotificationManager 去通知用户。通知可以使用多种方式获得用户的注意——闪烁的背光、振动设备、播放声音等等。典型的是放在一个持久的图标在状态栏，用户可以打开获取信息。

2.4、内容提供者（Content providers）

内容提供者（content provider）使一个应用程序的指定数据集提供给其他应用程序。这些数据可以存储在文件系统中、在一个 SQLite 数据库、或以任何其他合理的方式。内容提供者继承自 ContentProvider 基类并实现了一个标准的方法集，使得其他应用程序可以检索和存储数据。然而，应用程序并不直接调用这些方法。相反，替代的是它们使用一个 ContentResolver 对象并调用它的方法。ContentResolver 能与任何内容提供者通信，它与提供者合作来管理参与进来的进程间的通信。
内容提供者是 Android 应用程序的主要组成部分之一，提供内容给应用程序。他们封装数据且通过单个 ContentResolver 接口提供给应用程序。只有需要在多个应用程序间共享数据是才需要内容提供者。例如，通讯录数据被多个应用程序使用，且必须存储在一个内容提供者中。如果你不需要在多个应用程序间共享数据，你可以直接使用 SQLiteDataBase。
当 ContentResolver 发出一个请求时，系统检查给定的 URI 的权限并传递请求给内容提供者注册。内容提供者能理解 URI 想要的东西。UriMatcher 类用于帮组解析 URIs。
需要实现的方法主要如下：

- query(Uri, String[], String, String[], String) 返回数据给调用者
- insert(Uri, ContentValues) 插入数据到内容提供者
- update(Uri, ContentValues, String, String[]) 更新内容提供者已存在的数据
- delete(Uri, String, String[]) 从内容提供者中删除数据
- getType(Uri) 返回内容提供者中的 MIME 类型数据

更多的关于 ContentResolver 信息，请查看相关文档。
每当有一个应该由特定组件处理的请求，Android 可以确保该组件的应用程序正在运行，如果没有就启动它，而且一个适当的组件实例可用，如果没有就创建。

——成功属于耐得住寂寞的人，你离成功又近了一步了。

引言

上篇 Android 开发之旅：应用程序基础及组件介绍了应用程序的基础知识及 Android 的四个组件，本篇将介绍如何激活组关闭组件等。本文的主题如下：

- 1、激活组件：意图（Intents）
  - 1.1、活动（Activity）组件的激活
  - 1.2、服务（Service）组件的激活
  - 1.3、广播接收者（Broadcast receiver）组件的激活
- 2、关闭组件
- 3、清单文件
- 4、Intent 过滤器

1、激活组件：意图（Intents）

当接收到 ContentResolver 发出的请求后，内容提供者被激活。而其它三种组件——活动、服务和广播接收者，被一种叫做**意图（*intent*）的异步消息**激活。意图是一个保存着消息内容的 Intent 对象。**对于活动和服务来说**，Intent 对象指明了请求的操作名称以及作为操作对象的数据的 URI 和其它一些信息。例如，它可以传递对活动的一个请求，让它为用户显示一张图片，或者让用户编辑一些文本。**而对于广播接收者而言**，Intent 对象指明了广播的行为。例如当照相按钮被按下，它可以对所有感兴趣的对象广播。
对于每种组件来说，激活的方法是不同的。下面将分别介绍活动、服务、广播接收者组件的激活方法。

1.1、活动（Activity）组件的激活

通过传递一个 Intent 对象至 Context.startActivity()或 Activity.startActivityForResult()以载入（或指定新工作给）一个活动。相应的活动可以看到初始的意图，这个意图通过 getIntent() 方法来查看激活活动。Android 调用活动的 onNewIntent()方法传递任何后续的意图。
一个活动经常启动了下一个。如果它期望它所启动的那个活动返回一个结果，它会调用 startActivityForResult()而不是 startActivity()。例如，如果它启动了一个活动让用户挑选一张照片，它可能会返回被选中的照片。结果以一个 Intent 对象传递调用活动的 onActivityResult() 方法。

1.2、服务（Service）组件的激活

通过传递一个 Intent 对象至 Context.startService()以启动一个服务（或给予正在运行的服务以一个新的指令）。Android 调用服务的 onStart()方法并将 Intent 对象传递给它。
与此类似，一个 Intent 可以传递给 Context.bindService()以在调用的组件和目标服务之间建立持续的连接。这个服务会在调用 onBind() 方法中接受这个 Intent 对象（如果服务尚未启动，bindService() 会先启动它）。例如，一个活动可以连接至前面讲到的音乐播放服务，并提供给用户一个可操作的（用户界面）以对播放进行控制。这个活动可以调用 bindService()来建立连接，然后调用服务中定义的对象来控制播放。

1.3、广播接收者（Broadcast receiver）组件的激活

应用程序可以通过将 Intent 对象传递给

- Context.sendBroadcast()
- Context.sendOrderedBroadcast()
- Context.sendStickyBroadcast()

及其它类似方法来产生一个广播。Android 会通过 onReceive()方法将 intent 传递给所有对此广播有兴趣的广播接收者。

2、关闭组件

内容提供者仅在响应 ContentResolver 提出请求的时候激活。而一个广播接收者仅在响应广播信息的时候激活。所以，没有必要去显式的关闭这些组件。
而活动则不同，它提供了用户界面。与用户进行会话，所以只要会话依然持续，哪怕对话进程空闲，它都会一直保持激活状态。与此相似，服务也会在很长一段时间内保持运行。所以 Android 提供方法有序地关闭活动和服务。

- 可以通过调用它的 finish()方法来关闭一个活动。一个活动也可以通过调用 finishActivity()方法来关闭另外一个活动（它用 startActivityForResult() 启动的）。
- 服务可以通过调用它的 stopSelf()方法来停止，或者调用 Context.stopService()。

当组件不再被使用的时候或者 Android 必须要为更多活跃的组件回收内存时，组件也可能会被系统关闭。

3、清单（manifest）文件

当 Android 启动一个应用程序组件之前，它必须知道那个组件是存在的。所以，应用程序会在一个清单（manifest）文件中声明它的组件，这个文件会被打包到 Android 包中。这个.apk 文件还将包括应用程序的代码、文件以及其它资源。
这个清单文件是 XML 结构的文件，且所有的 Android 应用程序都把它叫做 AndroidManifest.xml。为声明一个应用程序组件，它还会做很多额外工作，比如指明应用程序所需链接到的库的名称（除了默认的 Android 库之外）以及声明应用程序期望获得的各种权限。
但清单文件的主要功能仍然是向 Android 声明应用程序的组件。举例说明，一个活动可以如下声明：

```
AndroidManifest.xml
```

<activity>元素的 name 属性指定了实现了这个活动的 Activity 类的子类，icon 和 label 属性指向了包含展示给用户的此活动的图标和标签的资源文件。
其它组件也以类似的方法声明——<service> 元素用于声明服务，<receiver> 元素用于声明广播接收者，而<provider>元素用于声明内容提供者。清单文件中未进行声明的活动、服务以及内容提供者将不为系统所见，从而也就不会被运行。然而，广播接收者既可以在清单文件中声明，也可以在代码中动态的创建（作为 BroadcastReceiver 对象）且调用 Context.registerReceiver()方式注册到系统。

4、Intent 过滤器

Intent 对象可以显式地指定目标组件。如果进行了这种指定，Android 会找到这个组件（依据清单文件中的声明）并激活它。但如果 Intent 没有进行显式的指定，Android 就必须为它找到对于 intent 来说最合适的组件。**这个过程是通过比较 Intent 对象和所有可能对象的 intent 过滤器完成的**。组件的 intent 过滤器会告知 Android 它所能处理的 intent 类型。如同其它关于组件的必要信息一样，它们在清单文件中进行声明的。这里是上面示例的一个扩展，其中加入了针对活动的两个 intent 过滤器声明：

```
AndroidManifest.xml
```

示例中的第一个过滤器——action：“android.intent.action.MAIN”和 category：“android.intent.category.LAUNCHER”的组合，是常见的。它标记这个活动显示在应用程序启动器中，用户在设备上看到的可启动的应用程序列表。换句话说，这个活动是应用程序的入口，是用户选择运行这个应用程序后所见到的第一个活动。第二个过滤器声明了这个活动针对特定类型的数据。
一个组件可以拥有任意数量的 intent 过滤器，每个声明一系列不同的能力。如果它没有包含任何过滤器，它将只能被显式声明了目标组件名称的意图激活。
对于广播接收者，它在代码中创建并注册 intent 过滤器，直接作为 IntentFilter 的对象实例化。其它过滤器则在清单文件中设置。

如果您现在对这些概念还没有完全理解，没关系这里我仅是让大家有个印象，知道这些概念或术语的存在，知道他们大概是做什么的。后面我还将陆续更详细地到这些东西并结合一些实例，到时候您就会清楚地知道这些东西。

——坚持就是胜利！关键是你能坚持吗？不能的话，你注定是个失败者。

引言

关于 Android 应用程序原理及术语，前面两篇：

- Android 开发之旅：应用程序基础及组件
- Android 开发之旅：应用程序基础及组件（续）

介绍了 Android 应用程序的进程运行方式：每一个应用程序运行在它自己的 Linux 进程中。当应用程序中的任何代码需要执行时，Android 将启动进程；当它不在需要且系统资源被其他应用程序请求时，Android 将关闭进程。而且我们还知道了 Android 应用程序不像别的应用程序那样（有 Main 函数入口点），它没有单一的程序入口点，但是它必须要有四个组件中的一个或几个：活动（Activities） 、服务（Services） 、广播接收者（Broadcast receivers） 、内容提供者（Content providers）。且分别介绍它们的作用，及如何激活和关闭它们、如何在清单文件（AndroidManifest.xml）中声明它们及 Intent 过滤器。

在简单回顾之后，本篇还是继续介绍 Android 应用程序原理及术语——活动与任务（Activities and Tasks）。

- 1、活动与任务概述
- 2、亲和度和新任务（Affinities and new tasks）
- 3、启动模式（Launch modes）
- 4、清除栈（Clearing the stack）
- 5、启动任务（Starting tasks）

1、活动与任务概述

如前所述，一个活动（activity）能启动另一个活动，包括定义在别的应用程序中的活动。再次举例说明，假设你想让用户显示某地的街道地图。而且已经有了一个活动能做这个事情（**假设**这个活动叫做地图查看器），因此你的活动要做的就是将请求信息放进一个 Intent 对象，然后将它传给 startActivity()。地图查看器就启动并显示出地图。当用户点击返回按钮之后，你的活动就会重新出现在屏幕上。

对用户来说，这个地图查看器就好像是你的应用程序的活动一样，虽然它定义在其他的应用程序中且运行在那个应用程序的进程中。Android 将这些活动保持在同一个**任务**（*task*）中以维持用户的体验。**简单地讲，任务是用户体验上的一个“应用程序”，是排成堆栈的一组相关活动**。栈底的活动（根活动）是起始活动——一般来讲，它是用户在应用程序启动器（也称应用程序列表，下同）中选择的一个活动。栈顶的活动是正在运行的活动——它关注用户的行为（操作）。当一个活动启动另一个，新的活动被压入栈顶，变为正在运行的活动。前面那个活动保存在栈中。当用户点击返回按钮时，当前活动从栈顶中弹出，且前面那个活动恢复成为正在运行的活动。（关于栈的**先进后出**特性不要我在这里讲吧！）

栈中包含对象，因此如果一个活动（**再次说明：活动是 Activity 的子类**）启动了多个实例——例如多个地图查看器，则栈对每个实例有一个独立的入口。（可以这样理解：假设有四个活动以这样的顺序排在栈中——A-B-C-D，现在又有一个 C 的实例，则栈变成 A-B-C-D-C，这两个 C 的实例是独立的。）栈中的活动从不会被重新排列，只会被压入、弹出。这点很好理解，因为活动的调用顺序是固定的。

任务是一栈的活动，而不是清单文件中声明的某个类或元素，因此无法独立于它的活动为任务赋值。整个任务的值是在栈底活动（根活动）设置的。例如，下节将讨论的“任务亲和度”，亲和度信息就是从任务的根活动中获取的。

一个任务的所有活动作为一个整体运行。整个任务（整个活动栈）可置于前台或发送到后台。例如，假设当前任务有四个活动在栈中——三个活动在当前活动下面。用户按下 HOME 键，切换到程序启动器，并选择一个新的应用程序（实际上是一个新的任务）。当前任务进入后台，新任务的根活动将显示。接着，过了一会，用户回到主屏幕并再次选择之前的应用程序（之前的任务）。那个任务栈中的所有四个活动都变为前台运行。当用户按下返回键时，不是离开当前任务回到之前任务的根活动。相反，栈顶的活动被移除且栈中的下一个活动将显示。

上面所描述的是活动和任务的默认行为，但是有方法来改变所有这些行为。活动与任务之间的联系及任务中活动的行为，是由启动活动的 Intent 对象的标志（flags）和清单文件中活动<activity>元素的属性共同决定的。

在这方面，主要的 Intent 标志有：

- FLAG_ACTIVITY_NEW_TASK
- FLAG_ACTIVITY_CLEAR_TOP
- FLAG_ACTIVITY_RESET_TASK_IF_NEEDED
- FLAG_ACTIVITY_SINGLE_TOP

主要的<activity>属性有：

- taskAffinity
- launchMode
- allowTaskReparenting
- clearTaskOnLaunch
- alwaysRetainTaskState
- finishOnTaskLaunch

接下来的小节将讨论这些标志和属性的作用，他们怎么交互，及使用的注意事项。

2、亲和度和新任务（Affinities and new tasks）

默认情况下，一个应用程序的所有活动互相之间都有一个**亲和度（*affinity*）**——也就是说，他们属于同一个任务的偏好（preference）。然而，也可以通过<activity>元素的 taskAffinity 属性为每个活动设置个体亲和度。定义在不同应用程序中的活动能够共享亲和度，同一个应用程序中的活动可以分配不一样的亲和度。亲和度发挥作用的两种情况：1）启动活动的 Intent 对象包含 FLAG_ACTIVITY_NEW_TASK 标志时；2）一个活动的 allowTaskReparenting 属性为"true"时。

- FLAG_ACTIVITY_NEW_TASK 标志

如前所述，默认情况下，一个新的活动被启动到调用 startActivity()方法的活动所在的任务。它被压入调用它的活动的栈中。但是，如果传递给方法的 Intent 对象包含 FLAG_ACTIVITY_NEW_TASK 标志，系统找一个不同的任务容纳活动。通常，顾名思义它表示一个新任务。但是，他并非一定如此。如果已经存在一个任务与新活动亲和度一样，该活动将启动到该任务。如果不是，则启动一个新任务。

- allowTaskReparenting 属性

如果一个活动的 allowTaskReparenting 属性为"true"，它可以从启动它的任务转移到与它有亲和度并转到前台运行的任务中。例如，假设一个天气预报的活动，但选择城市是一个旅游应用程序的一部分。它与同一个应用程序中的其他活动具有相同的亲和度，且允许重新选择父活动（reparenting）。你的一个活动启动天气预报活动，因此他初始是跟你的活动属于同一个任务。但是，当旅游应用程序切换到前台运行时，天气预报活动将被重新分配和显示到该任务。

如果一个.apk 文件，从用户的角度看包含不止一个“应用程序”，你可能要为与他们有关的些活动指定不一样的亲和度。

3、启动模式（Launch modes）

有四种不同的启动模式可以分配到<activity>元素的 launchMody 属性：

- "standard"(默认模式)
- "singleTop "
- "singleTask"
- "singleInstance"

这些模式的在以下四方面不同：

- **哪个任务将持有响应意图（intent）的活动**。对"standard"和"singleTop "模式，是产生意图的任务（调用 startActivity()方法）——除非 Intent 对象包含 FLAG_ACTIVITY_NEW_TASK 标志。在那种情况下，像上一节亲和度和新任务（Affinities and new tasks）所描述的那样选择一个不同的任务。相反，"singleTask"和"singleInstance"模式，总是将活动标记为一个任务的根活动。他们定义一个任务，而从不启动到其他任务。

- **活动是否可以实例化多次**。"standard"或"singleTop "活动可以实例化多次。这些实例可以属于多个任务，且一个给定任务可以包含同一个活动的多个实例。
  相反，"singleTask"和"singleInstance"活动仅可以被实例化一次。因为这些活动是一个任务的根，这个限制意味着设备上一个时间只有不多于一个任务的实例。
- **是否允许实例所在任务有其他活动**。"singleInstance"活动所在任务只有它一个活动。如果他启动别的活动，那些活动将启动到不同的任务中，无论它的模式如何——就好像 Intent 对象包含 FLAG_ACTIVITY_NEW_TASK 标志。在所有其他方面，"singleInstance"模式等同于"singleTask"模式。其它三种模式允许多个活动属于一个任务。"singleTask"活动总是任务的根活动，但是它能启动其他活动到它的任务。"standard"或"singleTop "活动的实例可以出现的栈中的任何位置。
- **响应一个意图时是否需要生成类的新实例**。对于默认的"standard"模式，创建新的实例去响应每一个新的意图。每个实例仅处理一个意图。对于"singleTop "模式，一个类已存在的实例可以重新用了处理新的意图，如果它位于目标任务的活动栈的栈顶。如果不是在栈顶，就不可以重用。相反，将创建一个新的实例并压入栈顶。
  例如，一个任务的活动栈由根活动 A、B、C 和 D 组成，顺序为 A-B-C-D。当一个意图到达请求类型 D 时，如果 D 是默认的"standard"模式，将产生 D 类的新实例且栈变为 A-B-C-D-D。然而，如果 D 的启动模式是"singleTop "，已存在的 D 实例将去处理新的意图（因为它在栈顶）且栈仍然是 A-B-C-D。如果，另一方面，到达的意图是请求类型 B 时，一个 B 的新实例将启动而不管 B 的模式是"standard"还是"singleTop "（因为 B 不是在栈顶），因此栈的结构为 A-B-C-D-B。
  如前所述，"singleTask"和"singleInstance"活动仅可以被实例化一次，因此他们的实例将处理所有的新意图。一个"singleInstance"活动总是在栈顶（因为仅有一个活动在任务中），因此它总是在可以处理意图的位置。然而，一个"singleTask"活动在栈中可能有或可能没有其他活动在它上面。如果有，即它不在处理意图的位置，意图会被丢弃（即使意图被丢弃了，它的到来使任务转到并保持在前台运行）

当一个已存在的活动被请求处理一个新的意图，Intent 对象将通过 onNewIntent()调用传到活动。（产生启动活动的意图对象可以由 getIntent()获取。）
注意到当一个活动的新实例被创建去处理新意图时，用户总是可以按返回键返回到之前的状态（之前的活动）。但是当一个已存在的活动实例去处理新意图是，用户不可以按返回键返回到意图到达之前的状态。

4、清除栈（Clearing the stack）

如果用户离开一个任务很长时间，系统将会清除根活动之外的活动。当用户再次返回到这个任务时，像用户离开时一样，仅显示初始的活动。这个想法是，一段时间后，用户可能已经放弃之前做的东西，及返回任务做新的事情。这是默认情况，有些活动属性可以用来控制和改变这个行为。

- alwaysRetainTaskState 属性
  如果在任务的根活动中这个属性被设置为"true"，刚才描述的默认行为将不会发生。任务将保留所有的活动在它的栈中，甚至是离开很长一段时间。
- clearTaskOnLaunch 属性
  如果在任务的根活动中这个属性被设置为"true"，只有用户离开就清除根活动之外的活动。换句话说，它与 alwaysRetainTaskState 截然相反。用户总是返回到任务的初始状态，甚至是只离开一会。
- finishOnTaskLaunch 属性
  这个属性类似于 clearTaskOnLaunch，但是它作用于单个活动，而不是整个任务。而且它能移除任何活动，包括根活动。当它被设置为"true"，任务本次会话的活动的部分还存在，如果用户离开并返回到任务时，它将不再存在。

有其他的方法强制从栈中移除活动。如果 Intent 对象包含 FLAG_ACTIVITY_CLEAR_TOP 标志，目标任务已经有一个指定类型的活动实例，栈中该实例上面的其它活动将被移除而使它置于栈顶响应意图。如果指定的活动的启动类型是"standard"，它自己也将被移除出栈，且一个新的实例将被启动去处理到来的意图。这是因为当模式是"standard"时，总是创建一个新的实例去处理新的意图。

FLAG_ACTIVITY_CLEAR_TOP 标志经常与 FLAG_ACTIVITY_NEW_TASK 一起使用。当一起使用时，这些标志的方式是定位到另一个任务中的已存在的活动并把它放到可以处理意图的位置。

5、启动任务（Starting tasks）

通过给定活动一个意图过滤器"android.intent.action.MAIN"作为指定行为（action）和"android.intent.category.LAUNCHER"指定种类（category），活动就被设置为任务的入口点了。上篇 Android 开发之旅：应用程序基础及组件（续）第四节 Intent 过滤器中我们举了这样一个例子，它将导致该活动的图标（icon）和标签（label）显示在应用程序启动器，给用户启动它或启动之后任意时候返回到它。

它的第二个功能非常重要：用户可以离开任务且之后可以返回到它。基于这个原因，两个启动模式"singleTask"和"singleInstance"标记活动总是初始化一个任务来响应意图，仅可以使用在有 MAIN 和 LAUNCHER 过滤器的活动中。想象一下，如果没有这个过滤器将会发生什么：一个意图启动一个"singleTask"活动，开始一个新任务，用户在任务中做一些操作。然后用户按下 HOME 键，任务现在退到后台运行且被主屏幕遮蔽住。而且，由于活动不在应用程序启动器中显示，用户无法再返回。

类似的困难也出现在 FLAG_ACTIVITY_NEW_TASK 标志。如果这个标志导致一个活动开始一个新的任务且用户按 HOME 键离开它，就必须要有某种方法是用户能够导航回来。一些实体（如通知管理器）总是在外部任务启动活动，从不作为他们自己的一部分，因此他们总是将带 FLAG_ACTIVITY_NEW_TASK 标志的意图传到 startActivity()方法启动活动。如果你有活动能调用外部实体，可以使用此标志，注意用户有一个独立的方式返回到开始的任务。

如果您希望用户离开活动后就不能再回到这个活动，可以将<activity>元素的 finishOnTaskLaunch 属性设置为"true"。可以参见清除栈那节。

引言

当应用程序的组件第一次运行时，Android 将启动一个只有一个执行线程的 Linux 进程。默认，应用程序所有的组件运行在这个进程和线程中。然而，你可以安排组件运行在其他进程中，且你可以为进程衍生出其它线程。本文从下面几点来介绍 Android 的进程与线程：

- 1、进程
- 2、线程
  - 2.1、远程过程调用（Remote procedure calls，RPCs）
  - 2.2、线程安全方法

1、进程

组件运行于哪个进程中由清单文件控制。组件元素——<activity>、<service>、<receiver>、<provider>，都有一个 process 属性可以指定组件运行在哪个进程中。这个属性可以设置为每个组件运行在自己的进程中，或者某些组件共享一个进程而其他的不共享。他们还可以设置为不同应用程序的组件运行在同一个进程中——假设这些应用程序共享同一个 Linux 用户 ID 且被分配了同样的权限。<application>元素也有 process 属性，为所有的组件设置一个默认值。

所有的组件都在特定进程的主线程中实例化，且系统调用组件是由主线程派遣。不会为每个实例创建单独的线程，因此，对应这些调用的方法——诸如 View.onKeyDown()报告用用户的行为和生命周期通知，总是运行在进程的主线程中。这意味着，没有组件当被系统调用时应该执行很长时间或阻塞操作（如网络操作或循环计算），因为这将阻塞进程中的其它组件。你可以为长操作衍生独立的线程。

public boolean **onKeyDown**(int keyCode,KeyEvent event)：默认实现 KeyEvent.Callback.onKeyMultiple()，当按下视图的 KEYCODE_DPAD_CENTER 或 KEYCODE_ENTER 然后释放时执行，如果视图可用且可点击。

参数

keyCode-表示按钮被按下的键码，来自 KeyEvent

event-定义了按钮动作的 KeyEvent 对象

返回值

如果你处理事件，返回 true；如果你想下一个接收者处理事件，返回 false。
当内存剩余较小且其它进程请求较大内存并需要立即分配，Android 要回收某些进程，进程中的应用程序组件会被销毁。当他们再次运行时，会重新开始一个进程。
当决定终结哪个进程时，Android 会权衡他们对用户重要性的相对权值。例如，与运行在屏幕可见的活动进程相比（前台进程），它更容易关闭一个进程，它的活动在屏幕是不可见（后台进程）。决定是否终结进程，取决于运行在进程中的组件状态。关于组件的状态，将在后面一篇——**组件生命周期**中介绍。

2、线程

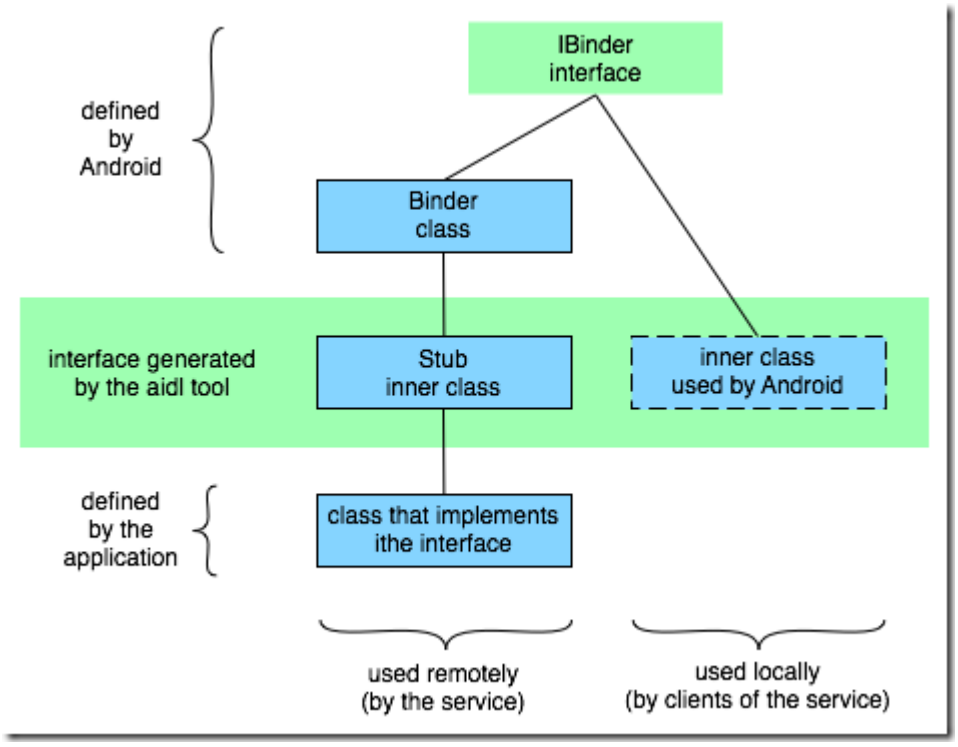
虽然你可能会将你的应用程序限制在一个进程中，但有时候你会需要衍生一个线程做一些后台工作。因为用户界面必须很快地响应用户的操作，所以活动寄宿的线程不应该做一些耗时的操作如网络下载。任何不可能在短时间完成的操作应该分配到别的线程。
线程在代码中是用标准的 Java 线程对象创建的，Android 提供了一些方便的类来管理线程——Looper 用于在线程中运行消息循环、Handler 用户处理消息、HandlerThread 用户设置一个消息循环的线程。
Looper 类
该类用户在线程中运行消息循环。线程默认没有消息循环，可以在线程中调用 prepare()创建一个运行循环；然后调用 loop()处理消息直到循环结束。大部分消息循环交互是通过 Handler 类。下面是一个典型的执行一个 Looper 线程的例子，分别使用 prepare()和 loop()创建一个初始的 Handler 与 Looper 交互：

```
class LooperThread extends Thread {
    public Handler mHandler;

    public void run() {
        Looper.prepare();

        mHandler = new Handler() {
            public void handleMessage(Message msg) {
                // process incoming messages here
            }
        };

        Looper.loop();
    }
}
```

更多的关于 Looper 的信息及 Handler、HandlerThread 请参阅相关资料。

2.1、远程过程调用（Remote procedure calls，RPCs）

Android 有一个轻量级的远程过程调用机制——方法在本地调用却在远程（另外一个进程中）执行，结果返回给调用者。这需要将方法调用和它伴随的数据分解为操作系统能够理解的层次，从本地进程和地址空间传输到远程进程和地址空间，并重新组装调用。返回值以相反方向传输。Android 提供了做这些工作的所有代码，这样我们可以专注于定义和执行 RPC 接口本身。
一个 RPC 接口仅包含方法。所有的方法同步地执行（本地方法阻塞直到远程方法执行完成），即使是没有返回值。简言之，该机制工作原理如下：首先，你用简单的 IDL（interface definition language，接口定义语言）声明一个你想实现的 RPC 接口。从这个声明中，aidl 工具生成一个 Java 接口定义，提供给本地和远程进程。它包含两个内部类，如下图所示：



内部类有管理你用 IDL 定义的接口的远程过程调用所需要的所有代码。这两个内部类都实现了 IBinder 接口。其中之一就是在本地由系统内部使用，你写代码可以忽略它。另外一个是 Stub，扩展自 Binder 类。除了用于有效地 IPC（interprocess communication）调用的内部代码，内部类在 RPC 接口声明中还包含方法声明。你可以定义 Stub 的子类实现这些方法，如图中所示。

通常情况下，远程过程有一个服务管理（因为服务能通知系统关于进程和它连接的其它进程的信息）。它有由 aidl 工具生成的接口文件和 Stub 子类实现的 RPC 方法。服务的客户端仅有由 aidl 工具生成的接口文件。

下面介绍服务如何与它的客户端建立连接：

- 服务的客户端（在本地端的）应该实现 onServiceConnected() 和 onServiceDisconnected() 方法，因此当与远程服务建立连接成功和断开连接是会通知它。然后调用 bindService() 建立连接。
- 服务的 onBind()方法将实现为接受或拒绝连接，者取决于它接受到的意图（该意图传送到 binServive()）。如果连接被接受，它返回一个 Stub 子类的实例。
- 如果服务接受连接，Android 调用客户端的 onServiceConnected()方法且传递给它一个 IBinder 对象，返回由服务管理的 Stub 子类的一个代理。通过代理，客户端可以调用远程服务。

这里只是简单地描述，省略了一些 RPC 机制的细节。你可以查阅相关资料或继续关注 Android 开发之旅，后面将为你奉上。

2.2、线程安全方法

在一些情况下，你实现的方法可能会被不止一个线程调用，因此必须写成线程安全的。这对远程调用方法是正确的——如上一节讨论的 RPC 机制。当从 IBinder 进程中调用一个 IBinder 对象中实现的一个方法，这个方法在调用者的线程中执行。然而，当从别的进程中调用，方法将在 Android 维护的 IBinder 进程中的线程池中选择一个执行，它不在进程的主线程中执行。例如，一个服务的 onBind()方法在服务进程的主线程中被调用，在 onBind()返回的对象中执行的方法（例如，实现 RPC 方法的 Stub 子类）将在线程池中被调用。由于服务可以有一个以上的客户端，所以同时可以有一个以上的线程在执行同一个 IBinder 方法。因此，IBinder 的方法必须是线程安全的。

同样，一个内容提供者可以接受其它进程产生的数据请求。虽然 ContentResolver 和 ContentProvider 类隐藏进程通信如何管理的，对应哪些请求的 ContentResolver 方法——query()、insert()、delete()、update()、getType()，在内容提供者的进程的线程池中被调用，而不是在这一进程的主线程中。因为这些方法可以同时从任意数量的线程中调用，他们也必须实现为线程安全的。

Android 开发之旅：组件生命周期（二）

2010-05-06 23:59 by 吴秦, 1360 visits, 网摘, 收藏, 编辑

引言

应用程序组件有一个生命周期——一开始 Android 实例化他们响应意图，直到结束实例被销毁。在这期间，他们有时候处于激活状态，有时候处于非激活状态；对于活动，对用户有时候可见，有时候不可见。组件生命周期将讨论活动、服务、广播接收者的生命周期——包括在生命周期中他们可能的状态、通知状态改变的方法、及这些状态的组件寄宿的进程被终结和实例被销毁的可能性。

上篇 Android 开发之旅：组件生命周期（一）讲解了论活动的生命周期及他们可能的状态、通知状态改变的方法。本篇将介绍服务和广播接收者的生命周期：

- 服务生命周期
- 广播接收者生命周期

1、服务生命周期

一个服务可以用在两个方面：

- 它可以启动且允许一直运行直到有人停止它，或者它自己停止。在这种模式，通过调用 Context.startService()启动服务及通过调用 Context.stopService()停止服务。服务也可以通过调用 Service.stopSelf()或 Service.stopSelfResult()停止自己。仅需要调用一次 stopService()停止服务，而不管调用 startService()了多少次。
- 通过使用相关接口可以编程地操作服务。客户端建立与 Service 对象的一个连接及使用该连接调入服务。连接通过调用 Context.bindService()建立，通过调用 Context.unbindService()关闭。多个客户端可以绑定到同一个服务。如果服务尚未启动，bindService()可以选择启动它。

这两种模式并不是完全分离的。你可以绑定到一个用 startService()启动的服务。例如，一个后台音乐服务可以通过使用定义了音乐播放的 Intent 对象调用 startService()启动。直到后来，用户可能想对播放器做一些控制或者获取当前歌曲的一些信息，一个活动将调用 bindService()与服务建立连接。在这种情况下，实际上直到最后一个绑定关闭 stopService()并不会停止。

像活动一样，一个服务也有生命周期方法，你可以执行监视它的状态改变。但是比活动的生命周期方法更少，只有三个且它们是公有的（public）而不是受保护的（protected）（说明：活动的生命周期方法是 protected 的）：

- void onCreate()
- void onStart(Intent *intent*)
- void onDestory()

通过这三个方法，你可以监视服务生命周期的两个嵌套循环：

- **服务的整个生命时间（entire lifetime）**，从调用 onCreate()到相应地调用 onDestory()。像一个活动一样，服务在 onCreate()中做一些初始设置，且在中释放所有的资源。例如，一个音乐播放服务可以在 onCreate()中创建线程，然后在 onDestory()中停止线程。
- **服务的活跃生命时间（active lifetime）**，从调用 onStart()开始。这个方法传递参数是传送给 startService()的 Intent 对象。音乐服务将打开 Intent，了解播放哪个音乐并且开始播放。

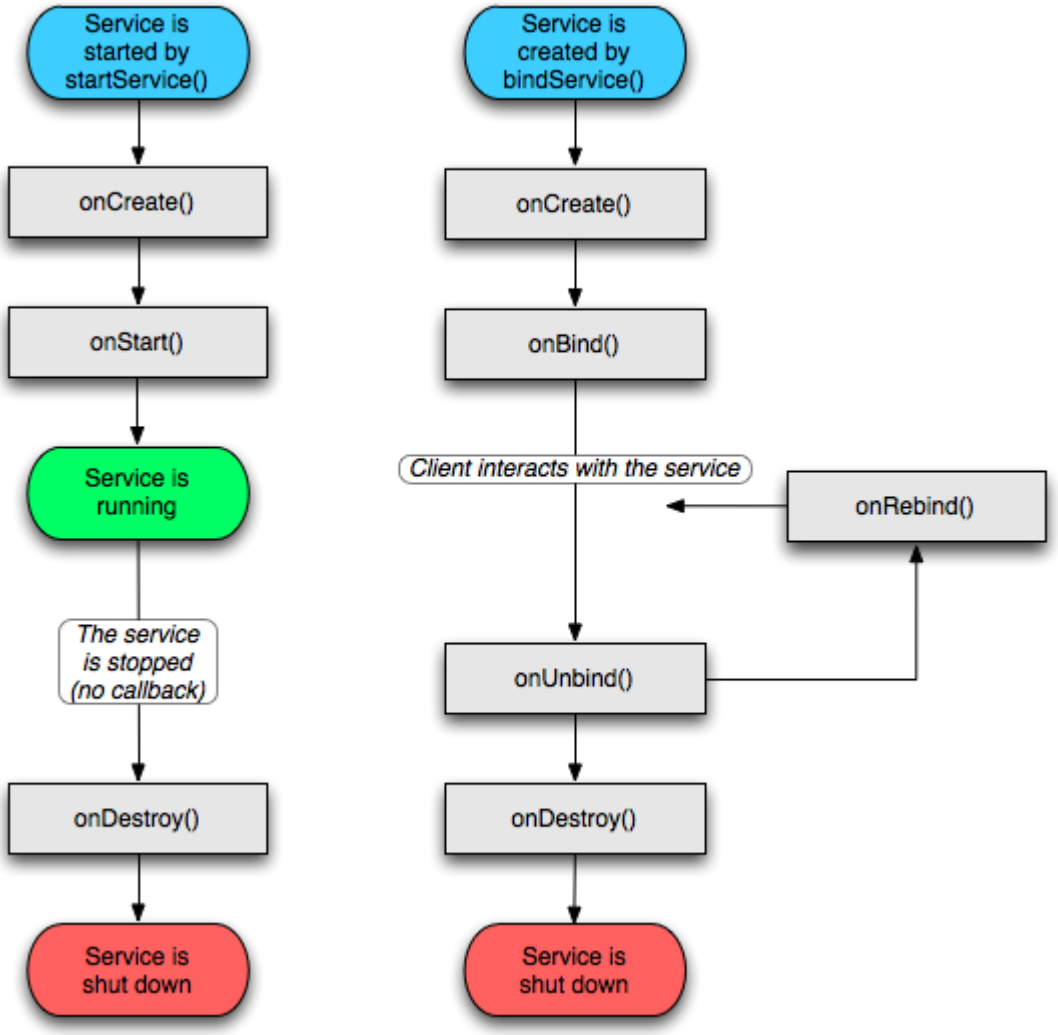
  没有相应的回调方法，因为服务停止没有 onStop()方法。

startService()和 onDestory()被所有服务调用，不管是通过 Context.startService()启动还是通过 Context.bindService()启动的。然而，onStart()仅被通过 startService()启动的服务调用。

如果一个服务允许别的绑定到它，有一些额外的回调方法来实现它：

- IBinder onBind(Intent *intent*)
- boolean onUnbind(Intent *intent*)
- void onRebind(Intent *intent*)

onBind()回调传递的参数是传给 bindService()的 Intent 对象，onUnbind()回调传递的参数是传给 unbindService()的 Intent 对象。如果服务允许绑定，onBind()返回客户端与服务交互的通信通道。onUnbind()方法可以要求调用 onRebind()，如果一个新的客户端连接到服务。
下图解释了服务的回调方法。虽然，它分离了由 startService()启动的服务和由 bindService()启动的服务，记住任何服务，无论它怎么启动的，都可能允许客户端绑定到它，因此任何服务可能接收 onBind()和 onUnbind()调用。



2、广播接收者生命周期

一个广播接收者有一个回调方法：void onReceive(Context *curContext*, Intent *broadcastMsg*)。当一个广播消息到达接收者是，Android 调用它的 onReceive()方法并传递给它包含消息的 Intent 对象。广播接收者被认为仅当它执行这个方法时是活跃的。当 onReceive()返回后，它是不活跃的。
有一个活跃的广播接收者的进程是受保护的，不会被杀死。但是系统可以在任何时候杀死仅有不活跃组件的进程，当占用的内存别的进程需要时。
这带来一个问题，当一个广播消息的响应时费时的，因此应该在独立的线程中做这些事，远离用户界面其它组件运行的主线程。如果 onReceive()衍生线程然后返回，整个进程，包括新的线程，被判定为不活跃的（除非进程中的其它应用程序组件是活跃的），将使它处于被杀的危机。解决这个问题的方法是 onReceive()启动一个服务，及时服务做这个工作，因此系统知道进程中有活跃的工作在做。

Android 开发之旅：组件生命周期（三）

2010-05-07 19:39 by 吴秦, 1456 visits, 网摘, 收藏, 编辑

Android 系统试图尽可能长地保持一个应用程序进程，但是当内存低时它最终还是需要移除旧的进程。为了决定保持哪个进程及杀死哪个进程，Android 将每个进程放入一个基于运行于其中的组件的重要性等级和这些组件的状态。重要性最低的进程首先被杀死，然后是次低，以此类推。总共有 5 个层次等级。下列清单按重要性顺序列出：

1. **前台进程**，用户当前工作所需要的。一个进程如果满足下列任何条件被认为是前台进程：
    1. 它正运行着一个正在与用户交互的活动（Activity 对象的 onResume()方法已经被调用）。
    2. 它寄宿了一个服务，该服务与一个与用户交互的活动绑定。
    3. 它有一个 Service 对象执行它的生命周期回调（onCreate()、onStart()、onDestroy()）。
    4. 它有一个 BroadcastReceiver 对象执行他的 onReceive()方法。

    在给定时间内仅有少数的前台进程存在。仅作为最后采取的措施他们才会被杀掉——如果内存太低以至于他们不能继续运行。一般来说，就在那时，设备达到一个内存？？？状态，因此杀掉某些前台进程以保持用户界面响应。
2. **可视进程**，他没有任何前台组件，但是仍然能影响用户在屏幕上看到东西。一个进程满足下面任何一个条件都被认为是可视的:
    1. 它寄宿着一个不是前台的活动，但是它对用户仍可见（它的 onPause()方法已经被调用）。举例来说，这可能发送在，如果一个前台活动是一个对话框且运行之前的活动在其后面仍可视。
    2. 它寄宿着一个服务，该服务绑定到一个可视的活动。

    一个可视进程被认为是及其重要的且不会被杀死，除非为了保持前台进程运行。
3. **服务进程**，是一个运行着一个用 startService()方法启动的服务，并且该服务并没有落入上面 2 种分类。虽然服务进程没有直接关系到任何用户可见的，它们通常做用户关心的事（诸如在后台播放 mp3 或者从网络上下载数据），因此系统保持它们运行，除非没有足够内存随着所有的前台进程和可视进程保持它们。
4. **后台进程**，是一个保持着一个当前对用户不可视的活动（已经调用 Activity 对象的 onStop()方法）。这些进程没有直接影响用户体验，并且可以在任何时候被杀以收回内存用于一个前台、可视、服务进程。一般地有很多后台进程运行着，因此它们保持在一个 LRU（least recently used，即最近最少使用，如果您学过操作系统的话会觉得它很熟悉，跟内存的页面置换算法 LRU 一样。）列表以确保最近使用最多的活动的进程最后被杀。如果一个活动执行正确地执行它的生命周期方法，且捕获它当前的状态，杀掉它对用户的体验没有有害的影响。
5. **空进程**，是一个没有保持活跃的应用程序组件的进程。保持这个进程可用的唯一原因是作为一个 cache 以提高下次启动组件的速度。系统进程杀死这些进程，以在进程 cache 和潜在的内核 cache 之间平衡整个系统资源。

Android 把进程标记为它可以的最高级，即进程中活跃的组件中重要性最高的那个（选取重要性最高的那个作为进程的重要性级别）。例如，有一个进程寄宿着一个服务和一个可视活动，进程的级别被设置为可视进程级别，而不是服务进程级别（因为可视进程级别比服务进程级别高）。

此外，一个进程的排名因为其他进程依赖它而上升。一个进程服务其它进程，它的排名从不会比它服务的进程低。例如，进程 A 中的一个内容提供者服务进程 B 中的一个客户，或者进程 A 中的一个服务绑定到进程 B 中的一个组件，进程 A 总是被认为比进程 B 重要。

因为一个运行一个服务进程排名比一个运行后台活动的进程排名高，一个活动启动一个服务来初始化一个长时间运行操作，而不是简单地衍生一个线程——特别是如果操作很可能会拖垮活动。这方面的例子是在后台播放音乐和上传相机拍摄的图片到一个网站。使用服务保证操作至少有“服务进程”的优先级，无论活动发生什么情况。

——量变产生质变，如果你从第一篇一直看到了这篇，可以说这就是你的质变点之一。

回顾及展望

经过数篇对 Android 应用程序的原理的讲述，现在我们大概回顾一下。

- 首先我们利用 Hello World 程序介绍了一个 Android 应用程序的目录结构，包括 src 文件夹、gen 文件夹、Android x 文件夹、assets 文件夹、AndroidMainifest.xml、default.properties；
- 接下来我们又站在架构的高度分析了一下 Android 系统的主要组成部分，包括 Linux Kernel、Android Runtime、Libraries、Application Framework、Application；
- 接下来我们又介绍了 Android 应用程序的运行及应用程序组件，包括活动（Activities）、服务（Services）、广播接收者（Broadcast receivers）、内容提供者（Content providers）等内容；
- 接着我们又介绍了如何激活及关闭组件，还有简单的介绍了 AndroidMainifest.xml、Intent 及其过滤器（这两者我们以后还要通过例子或者单独开篇深入分析）；
- 接着我们站在 Android 应用程序的角度分析 Android 中活动与任务，包括活动与任务概述、亲和度和新任务（Affinities and new tasks）、启动模式（Launch modes）、清除栈（Clearing the stack）、启动任务（Starting tasks）；
- 接着我们在 Android 应用程序运行的角度，简单分析了 Android 应用程序的进程与线程；
- 最后我们用分析了 Android 应用程序组件的生命周期，包括活动的生命周期及他们可能的状态、服务生命周期、广播接收者生命周期、Android 应用程序进程的分类及重要性等级。

至此，我们终于算是完全算是双脚步入 Android 开发的大门了，但我们现在还只能算是金字塔底端的那群人，还需要不断地实践、实践、再实践。而且上面所讲的是作为一个真正 Android 开发人员必须要深刻理解的东西，如果您还没有达到深刻的程度那请你**回去再浏览一遍**，然后跟着我的这个系列继续深入学习，在接下来的文章我将更多的是利用实例来解析这些东西。下面我再次用 Hello World 程序来分析一下 Android 应用程序，主要内容如下：

- “Hello World！”显示浅析
- “Hello World！”的手术（一）
- “Hello World！”的手术（二）
- “Hello World！”的手术（三）

1、“Hello World！”浅析

首先我们再次简单地新建一个 Hello World 项目，它的布局文件 res\layout\main.xml 的代码如下：

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

元素<TextView>的 android:text 就是我们在屏幕上显示的“Hello World, HelloWorld！”，android:text 的值是“@string/hello”，它是如何在屏幕上显示“Hello World, HelloWorld！ ” 的呢？。

在 main.xml 文件中以这种格式：

@[package:]string/some_name (where some_name is the name of a specific string)

引用 res/values/strings.xml 文件中的字符串，其中 some_name 是要引用的字符串的名字。strings.xml 文件代码如下：

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

```
    <string name="hello">Hello World, HelloWorld!</string>
    <string name="app_name">HelloWorld</string>
</resources>
```

由此可见，上面那个<TextView>的 android:text 引用的字符串是“Hello World, HelloWorld!”。接着想象一下，“Hello World, HelloWorld！”何时加载显示的呢？

Note：这种<TextView>的 text 的值是存储在 strings.xml 中的，而不是硬编码的。这样的好处是，当我们在 strings.xml 中修改 hello 的具体值时，不需要在 main.xml 中修改。

我们来看下 src 目录下 skynet.com.cnblogs.www 包（就是新建 Hello World 项目时定义的包名）中的 HelloWorld.java 文件，代码如下：

```
package skynet.com.cnblogs.www;

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

public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

可以看出在创建活动时，即 void onCreate(Bundle *saveInstanceState*)中，在 HelloWorld 中重写了它，在该方法中首先调用了父类的 onCreate 方法且接着调用了 setContentView(R.layout.main)方法。是的，就是在这个方法中，根据 main.xml 文件将其显示出来，因为 R.layout.main 是表示布局资源文件 main.xml 编译后的对象，Eclipse 插件会自动在 R.java 文件中创建这个引用。main.xml 中定义了<TextView>，然后根据它的 android:text 属性引用到 strings.xml 文件中的<string name="hello">Hello World, HelloWorld!</string>元素，然后将它显示到屏幕上。

从活动的生命周期可以知道，任何一个活动启动的一个方法是 onCreate()方法，在这里做一些初始化的工作，诸如创建视图、绑定数据列表等。在 HelloWorld 项目中，就是调用 setContentView 进行初始化工作，将 Hello World, HelloWorld!显示到屏幕上。

至此，我们对 Hello World 的认识更加深入了一点！下面我们让拿起手术刀对它进行一个手术。

2、Hello World 的手术（一）

我们将 main.xml 文件中的<TextView>元素删掉，取而代之用一个 Button 来显示“Hello World！”。代码如下：

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

</LinearLayout>
其它的地方不用修改，直接运行即可！

图 1、修改之前的

图 2、修改之后的

从这个例子我们可以看出我们上节的分析是对的，接下来我们继续对它做一个手术。

3、Hello World 的手术（二）

我们这次不用 Button 来显示“Hello World！”，我们这次用一个图片来显示。首先我们准备一张 png 的图片，如下：

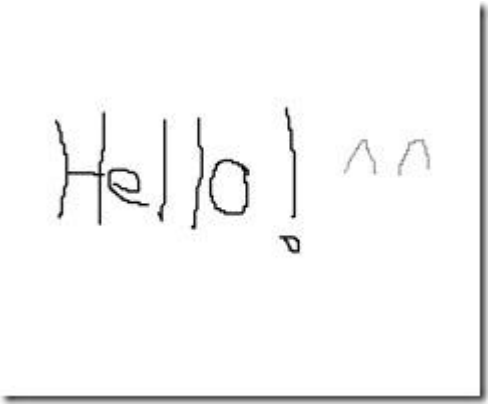


图 3、hello.png

接下来我们将它复制到， res/drawable-dhpi 中。然后我们查看 R.java 文件我们会发现，R 中自动加了一行：（关于 R 文件，你可以参考 Android 开发之旅：HelloWorld 项目的目录结构中的 **1.2、gen 文件夹**）

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found.  It
* should not be modified by hand.
*/

package skynet.com.cnblogs.www;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int hello=0x7f020000;//这行是 Android 自动新加的
        public static final int icon=0x7f020001;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

drawable- hdpi、drawable- mdpi、drawable-ldpi 的区别：

(1)drawable-hdpi 里面存放高分辨率的图片,如 WVGA (480x800),FWVGA (480x854)

(2)drawable-mdpi 里面存放中等分辨率的图片,如 HVGA (320x480)

(3)drawable-ldpi 里面存放低分辨率的图片,如 QVGA (240x320)

系统会根据机器的分辨率来分别到这几个文件夹里面去找对应的图片。

删除刚才我们添加的<Button>元素，添加一个<ImageView>元素。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ImageView
android:id="@+id/imageview"
android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/hello"
/>
</LinearLayout>
```

运行之后，结果如下所示：

图 4、图片显示 Hello World

我们看到上面的 ImageView 有一个 id 属性。其实，每个 View 对象都有一个关联的 ID，来唯一标识它。当应用程序被编译时，这个 ID 作为一个整数引用。但是 ID 通常是在布局 XML 文件中作为字符串分配的，在元素的 id 属性。这个 XML 属性对所有的 View 对象可用且会经常用到。XML 中的 ID 语法如下：**android:id="@+id/my_button"**。

字符串前的@符号表示 XML 解析器应该解析和扩展剩下的 ID 字符串，并把它作为 ID 资源。+符号表示这是一个新的资源名字，它必须被创建且加入到我们的资源（R.java 文件，R 是 Resource）。Android 框架提供一些其他的 ID 资源。当引用一个 Android 资源 ID 时，你不需要+符号，但是你必须添加 android 包名字空间，如下：**android:id="@android:id/empty"**。

4、Hello World 的手术（三）

上面两个手术都是基于 main.xml 的，在布局资源文件中声明相应的控件并设置相关属性，如 TextView、Button、ImageView 等，然后在 onCreate()方法中调用 setContentView(R.layout.main)方法来输出。其实上面的几个实验我们都可以编程地进行，下面就以编程地用 ImageView 显示“Hello World！”，别的也是类似来完成。

首先，我们将 main.xml 文件中的<ImageView>删掉，且将 res/values/strings.xml 中的<string name="hello">Hello World, HelloWorld!</string>删掉（其实第 3 节，用图片显示就没有用到它）；然后，在 HelloWorld.java 文件中 import android.widget.ImageView;且删掉 onCreate()方法中调用 setContentView(R.layout.main)方法，因为我们不需要根据 main.xml 布局资源文件来显示而是编程地 Hello World。

接下来我们在 onCreate()方法中声明一个 ImageView 对象，并设置它的图像资源，最后将这个 ImageView 对象传给 setContentView()方法，代码如下：

```
package skynet.com.cnblogs.www;

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

public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ImageView imageHello=new ImageView(this);
        imageHello.setImageResource(R.drawable.hello);
        setContentView(imageHello);
    }
}
```

运行之后，同样会得到 Hello World 的手术（二）的结果，只是图片的显示位置不太一样。**在这里注意上面那行粗体代码，如果是 TextView 或者 Button 的话，应该调用 setText()方法来设置”Hello World!”。**

图 5、编程地显示 Hello World！

Note：构建 ImageView 对象时传递了一个 ***this*** 参数，表示与当前上下文（context）关联。这个 Context 由系统处理，它提供诸如资源解析、获取访问数据库和偏好等服务。因为 Activity 类继承自 Context，且因为你的 HelloWorld 类是 Activity 的子类，它也是一个 Context。因此，你可以传递 ***this*** 作为你的 Context 给 ImageView 引用。

Android 开发之旅：深入分析布局文件&又是"Hello World！"

2010-05-20 18:45 by 吴秦, 2675 visits, 网摘, 收藏, 编辑

引言

上篇可以说是一个分水岭，它标志着我们从 Android 应用程序理论进入实践，我们拿起手术刀对默认的"Hello World！"程序进行了 3 个手术，我们清楚了"Hello world！"是如何实现显示在屏幕上的，而且我们知道不仅可以根据布局文件 main.xml 来初始化屏幕，还可编程地进行。以后基本我们都会以实践的方式来深入 Android 开发。我们这次深入分析 Android 应用程序的布局文件，主要内容如下：

- 1、用户界面及视图层次
- 2、Android 中布局定义方法
- 3、编写 XML 布局文件及加载 XML 资源
- 4、常用布局文件中元素的属性
  - 4.1、ID 属性
  - 4.2、布局参数
- 5、布局位置&大小&补距&边距
- 6、又是"Hello World！"

1、用户界面及视图层次

在通过“Hello World！”介绍 Android 中的布局问题之前，不得不先介绍一下 Android 中的用户界面，因为布局问题也是用户界面问题之一。在一个 Android 应用程序中，用户界面通过 View 和 ViewGroup 对象构建。Android 中有很多种 Views 和 ViewGroups，他们都继承自 View 类。View 对象是 Android 平台上表示用户界面的基本单元。

View 类：

extends Object

implements Drawable.Callback　KeyEvent.Callback　AccessibilityEventSource

这个类表示用户界面组件的基本构建块，一个 View 占据屏幕上的一个矩形区域，并负责绘图和事件处理。View 类是 *widgets* 的基类，*widgets* 用于创建交互式 UI 组件（buttons、text fields 等）。View 类的直接子类 ViewGroup 类是 *layouts* 的基类，*layouts* 是不可见的容器用户保持其他 Views 或者其他 ViewGroups 和定义它们的布局属性。

一个 View 对象是一个数据结构，它的属性存储屏幕上一个特定矩形区域的**布局参数**和**内容**。一个 View 对象处理它自己的测度、布局、绘图、焦点改变、滚动、键/手势等与屏幕上矩形区域的交互。作为用户界面中的对象，View 也是与用户交互的一个点且交互事件接收器。

在 Android 平台上，你定义活动的 UI 使用的 View 和 ViewGroup 节点的层次结构如下图所示。根据你的需要这个层次树可以是简单的或复杂的，并且你能使用 Android 预定义的 *widgets* 和 *layouts* 集合，或者使用自定义的 Views。

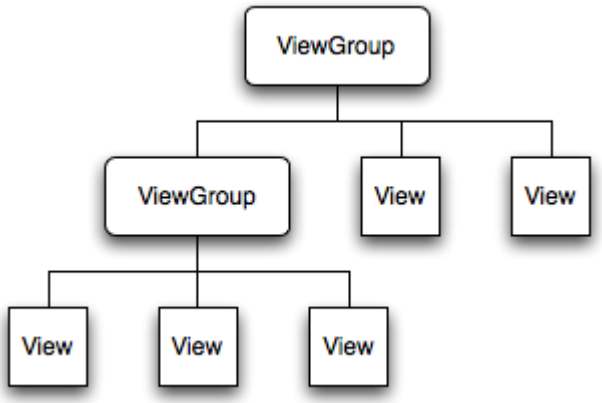


图 1、视图层次结构

为了将视图层次树呈现到屏幕上，你的活动必须调用 setContentView()方法并且传递到根节点对象的引用。Android 系统接收这个引用并使用它来验证、测度、绘制树。层次的根节点要求它的孩子节点绘制它自己——相应地每个试图组节点要求调用自己的孩子视图去绘制他们自己。子视图可能在父视图中请求指定的大小和位置，但是父视图对象有最终决定权（子视图在哪个位置及多大）。因为它们是按序绘制的，如果元素有重叠的地方，重叠部分后面绘制的将在之前绘制的上面。

2、Android 中布局定义方法

布局是一个活动中的用户界面的架构，它定义了布局结构且存储所有显示给用户的元素。有两种方式可以声明布局，这个我们在上文中已经用了（对应上文的“Hello World 的手术（二）”、“Hello World 的手术（三）”）。我们再重温总结一下：

- **方法一、在 XML 格式的布局文件中声明 UI**。Android 提供了简易的 XML 词汇表对应视图类和其子类，诸如 *widgets* 和 *layouts*。
- **方法二、在运行时实例化布局元素**。可以编程地创建 View 和 ViewGroup 对象，并操作他们的属性。

Android 框架给我们灵活地使用这两个方法之一或两个声明和管理你的应用程序的 UI。例如，你可以用 XML 格式的布局文件定义应用程序默认的布局，包括将显示在屏幕的元素和属性。然后你可以编程地修改屏幕上对象的状态，包括定义在 XML 文件中的元素。

最常用的是方法一，即用一个 XML 的布局文件定义自己的布局和表达层次视图。XML 提供一种直观的布局结构，类似 HTML。XML 中的每个元素是一个 View 或者 ViewGroup 对象（或继承自他们的对象）。View 对象是树中的叶子，ViewGroup 对象是树中的分支，这点可以从上面的视图层次树中可以看出。

在 XML 布局文件中声明 UI 的优点是：使**应用程序的界面**与**控制它行为的代码**更好地分离了。UI 描述在应用程序代码之外，这意味着你可以修改或调整它而不用修改你的源码并重新编译。例如，你可以为不同的屏幕方向、不同的屏幕大小、不同的语言创建 XML 布局文件。此外，在 XML 中声明布局更易地可视化你的 UI 结构，因此更容易调试问题。

一个元素 XML 元素的名字对应到一个 Java 类，因此一个<TextView>元素在你的 UI 中创建一个 TextView，一个<linearLayout>元素创建一个 LinearLayout 的视图组。当你加载一个布局资源时，Android 系统初始化这些运行时对象，对应你的布局中的元素。XML 元素的属性对应到一个 Java 类的方法。

3、编写 XML 布局文件及加载 XML 资源

使用 Android 的 XML 词汇，我们可以快速地设计 UI 布局及包含的屏幕元素，就像 web 页面的 HTML。每个布局文件必须包含一个根元素，根元素必须是一个 View 或 ViewGroup 对象。一旦你已经定义了根元素，你可以添加额外的 *layout* 对象或 *widgets* 作为子元素，逐步地构建一个视图层次定义你的布局。例如，下面的 XML 布局文件使用了纵向的 LinearLayout 保存一个 TextView 和一个 Button。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

布局文件以.xml 为扩展名，保存在 res/layout/下面，它将会被正确地编译。我们已经定义好了布局文件，那它是怎么被加载的呢？当我们编译应用程序时，每个 XML 布局文件被编译成一个 View 资源。我们应该在应用程序代码中加载布局资源，在 Activity.onCreate()回调中通过调用 setContentView()实现，以 **R.layout.layout_*file_name*** 形式传递给它布局资源的引用。例如，如果你的 XML 布局保存为 main_layout.xml，你应该这样加载它：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView.(R.layout.main_layout);
}
```

活动中的 onCreate()回调方法，当你的活动启动时被 Android 框架调用（详见 Android 开发之旅：组件生命周期（一），详细介绍了活动组件的生命周期）。

4、常用布局文件中元素的属性

每个 View 和 ViewGroup 对象支持他们自己的各种 XML 属性。一些属性特定于一个 View 对象（例如，TextView 支持 textSize 属性），但是这些属性也被继承自这个类的任何 View 对象继承。一些属性对所有 View 对象可用，因为他们从根 View 类继承（诸如 id 属性）。并且，其他属性被考虑为“布局参数”，这些属性描述特定 View 对象的特定布局方向，由对象的父 ViewGroup 对象定义。

4.1、ID 属性

每个 View 对象都有一个关联的 ID，来唯一标识它。当应用程序被编译时，这个 ID 作为一个整数引用。但是 ID 通常是在布局 XML 文件中作为字符串分配的，作为元素的 id 属性。这个 XML 属性对所有的 View 对象可用且会经常用到。XML 中的 ID 语法如下：

android:id="@+id/my_button"

字符串前的**@符号表示 XML 解析器应该解析和扩展剩下的 ID 字符串**，并把它作为 ID 资源。**+符号表示这是一个新的资源名字，它必须被创建且加入到我们的资源**（R.java 文件，R 是 Resource）。Android 框架提供一些其他的 ID 资源。当引用一个 Android 资源 ID 时，你不需要+符号，但是你必须添加 android 包名字空间，如下：

android:id="@android:id/empty"

为了创建视图和从应用程序引用他们，通常的模式是：

1. 首先在布局文件中定义一个视图/构件对象并分配一个唯一的 ID：

```
<Button android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

2. 然后创建一个视图对象实例并从布局中获取它（典型的是在 onCreate()方法中）：

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

4.2、布局参数

名为 ***layout_something*** 的 XML 布局属性，为视图定义适合于它所驻留的 ViewGroup 的布局参数。每个 ViewGroup 类实现一个扩展自 ViewGroup.LayoutParams 的嵌套类。这个子类包含为每个子视图定义**大小**和**位置**的属性类型，以适合于该视图组。如下图所示，父视图组为每个子视图定义布局参数（包括子视图组）。

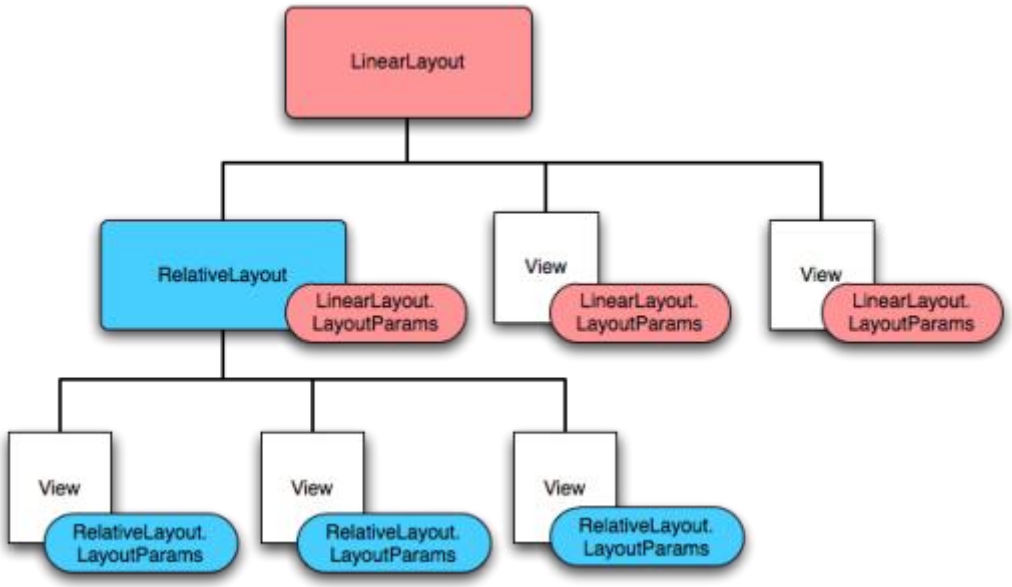


图 2、布局参数

注意每个 LayoutParams 子类有它自己的设置值的语法。每个子元素必须定义适合于它父视图的 LayoutParams，虽然它可能也为自己的子视图定义不同 LayoutParams。

所有的视图组包括宽带和高度（**layout_width** 和 **layout_height**），并且每个视图要求要定义它们。许多 LayoutParams 也包括可选的边距和边界。你可以指定宽度和高度的具体值，虽然你可能并不想这

样做。更多地你将告诉视图它的大小依据它内容要求或跟父视图组所允许的一样大（分别用 wrap_content 和 fill_parent 值）。

5、布局位置&大小&补距&边距

视图的几何形状是一个矩形。视图的位置表示为一个 left 和 top 的坐标对，尺寸（dimensions）表示为宽度和高度。位置和尺寸的单位是像素（pixel）。

可以通过调用 getLeft()和 getTop()检索视图的位置。前者返回视图矩形坐标的 left 或 x，后者返回视图矩形坐标的 right 或 y。这些方法返回相对于与其父视图的相对位置，例如当 getLeft()返回 20，即认为视图到其直接父视图的左边距离为 20 像素。

此外，提供了一些额外的方法如 getRight()和 getButtom()避免不必要的计算。这些方法返回视图坐标的右边距和底边距。例如，调用 getRight()等同于下面的计算：getLeft()+getWidth()。

视图的大小（size）也表示为宽度和高度，但跟上面尺寸（dimensions）是区别的。上面的尺寸定义视图想在父视图中占多大，视图的尺寸可以通过 getMeasureWidth()和 getMeasureHeight()获得。而视图的大小（size）则表示视图在屏幕上的实际大小，他们的值可以跟视图尺寸的不一样，但也不是非得这样。视图的大小可以通过 getWidth()和 getHeight()获得。

为了估量视图的尺寸，必须考虑它的补距（padding，即视图内容与视图边框的距离）。补距以像素表示视图的 left、top、right 和 bottom 部分的空白。补距可以用来按特定数量的像素偏移视图内容。例如，左补距是 2 将输出内容离左边框 2 像素。补距可以通过 setPadding(int, int, int, int)方法设置和通过 getPaddingLeft()、getPaddingRight()、getPaddingTop()、getPaddingBottom()来查询。虽然一个视图可以定义补距，但是它不支持边距（margins）。然而，视图组支持边距。

6、又是“Hello World！”

下面我们通过几个实验来验证和加深上述关于布局文件理解。

6.1、又是“Hello World！”（一）

验证名为 **layout_*something*** 的 XML 布局属性，用 **layout_width** 和 **layout_height** 定义 Button 的宽度和高度，这两个属性也是每个视图对象都必须要声明定义的。验证 wrap_content 与 fill_parent 的区别，其实区别从他们的单词组成就可以看出：wrap——包，包裹等而 content——内容，则 wrap_content 表示视图对象的高度/宽度正好包住内容；fill——填充等而 parent——父，则 fill_parent 表示填满父视图。

实验设置为：在布局资源文件中定义两个 Button，id 分别为 button1、button2，button1 的宽度和高度属性都是 wrap_content，button2 的宽度和高度属性都是 fill_parent。main.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
    <Button android:id="@+id/button2"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="Hello, I am a Button"
    />
</LinearLayout>
```

而 HelloWorld.java 文件代码为：

```
package skynet.com.cnblogs.www;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.*;//注意：导入此包，或者是 android.widget.Button;

public class HelloWorld extends Activity {
    private CharSequence text="new Hello!";

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

运行可以得到如下图结果：

图 3、验证 验证名为 **layout_*something*** 的 XML 布局属性

明显可以看出 button1 的大小是刚好包住内容“Hello，I am a Button”，而 button2 的大小则是填满父视图的大小。

6.2、又是“Hello World！”（二）

接下来我们视图对象的通用属性 ID 及再次验证定义布局的两种方式。（关于 ID 属性——首先在布局文件中定义一个视图/构件对象并分配一个唯一的 ID，然后创建一个视图对象实例并从布局中获取它（典型的是在 onCreate()方法中））。

属性实验设置为：基本跟上面一样，但是我们要编程地修改 button1 的 text 属性。main.xml 布局文件跟上面一样，而主要是 HelloWorld.java 文件不一样，它的代码如下：

```
package skynet.com.cnblogs.www;

import android.app.Activity;
import android.os.Bundle;
//import android.widget.*;
import android.widget.Button;

public class HelloWorld extends Activity {
    private CharSequence text="new Hello!";

/** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button myButton=(Button)findViewById(R.id.button2);
        myButton.setText(text);
    }
}
```

上面红色粗体的两行代码就是编程地改变 button2 的 text 属性，这体现了 Android 框架给我们灵活地使用上面两个方法之一或两个声明和管理你的应用程序的 UI。运行结果如下图所示:

图 4、验证视图对象的通用属性 Id 的常用模式及编程地改变定义在 XML 布局文件中的 Button 的属性

6.3、又是“Hello World！”（三）

验证视图对象的布局位置&大小&补距，实验设置为：在实验一的基础上在 main.xml 中修改 button1 的布局位置、大小、补距等属性。修改后的 mian.xml 文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button"
        android:width="250dp"
        android:height="80dp"
        android:padding="20dp"
        android:layout_margin="20dp"
        />
    <Button android:id="@+id/button2"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="Hello, I am a Button"
    />
</LinearLayout>
```

运行后结果如下所示：

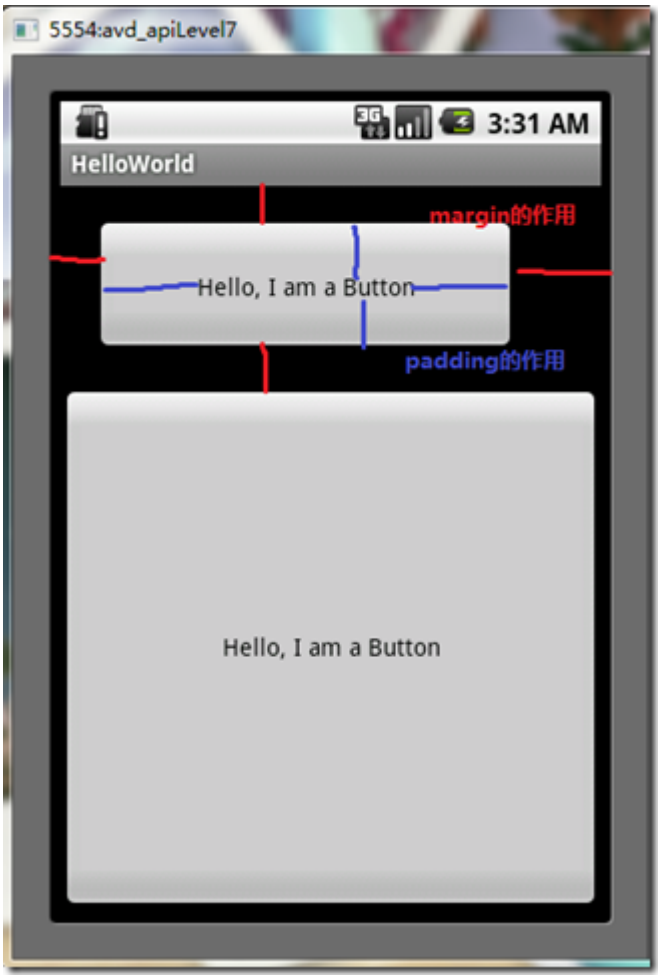


图 5、 验证视图对象的布局位置&大小&补距

NOTE：上述代码中布局位置&大小&补距的单位（如 width="250dp"）。单位可以为 px、in、mm、pt、dp、sp。

- px：pixels（像素）——对应屏幕上实际的像素
- in：inches（英寸）——基于物理屏幕的大小
- mm：millimeters（毫米）——基于物理屏幕的大小
- pt：points（点）——英寸的 1/72，基于基于物理屏幕的大小
- dp：density-independent pixels（独立于密度的像素）——一个抽象的基于物理屏幕密度的单位。这些单位是相对于一个 160dpi 的屏幕，所有一个 dp 是 160dpi 屏幕上的一个点。dp 到 px 的转换比率根据屏幕密度改变，但不一定是成正比。
- sp：scale-independent pixels（规模独立像素）——类似于 dp 单位，但是它也受用户字体大小设置的影响。当你指定字体大小时使用它，因为他们将根据屏幕和用户设置调整。

Android 开发之旅：view 的几种布局方式及实践

2010-06-06 14:14 by 吴秦, 4450 visits, 网摘, 收藏, 编辑

引言

通过前面两篇：

- Android 开发之旅：又见 Hello World！
- Android 开发之旅：深入分析布局文件&又是“Hello World！”

我们对 Android 应用程序运行原理及布局文件可谓有了比较深刻的认识和理解，并且用“Hello World！”程序来实践证明了。在继续深入 Android 开发之旅之前，有必要解决前两篇中没有介绍的遗留问题：View 的几种布局显示方法，以后就不会在针对布局方面做过多的介绍。View 的布局显示方式有下面几

种：**线性布局**（Linear Layout）、**相对布局**（Relative Layout）、**表格布局**（Table Layout）、**网格视图**（Grid View）、**标签布局**（Tab Layout）、**列表视图**（List View）、**绝对布局**（AbsoluteLayout）。本文虽然是介绍 View 的布局方式，但不仅仅是这样，其中涉及了很多小的知识点，绝对能给你带来 Android 大餐！

本文的主要内容就是分别介绍以上视图的七种布局显示方式效果及实现，大纲如下：

1、view 的布局显示概述

通过前面的学习我们知道：在一个 Android 应用程序中，用户界面通过 View 和 ViewGroup 对象构建。Android 中有很多种 View 和 ViewGroup，他们都继承自 View 类。View 对象是 Android 平台上表示用户界面的基本单元。

View 的布局显示方式直接影响用户界面，View 的布局方式是指一组 View 元素如何布局，准确的说是一个 ViewGroup 中包含的一些 View 怎么样布局。ViewGroup 类是布局（layout）和视图容器（View container）的基类，此类也定义了 ViewGroup.LayoutParams 类，它作为布局参数的基类，此类告诉父视图其中的子视图想如何显示。例如，XML 布局文件中名为 layout_*something* 的属性（参加上篇的 4.2 节）。我们要介绍的 View 的布局方式的类，都是直接或间接继承自 ViewGroup 类，如下图所示：

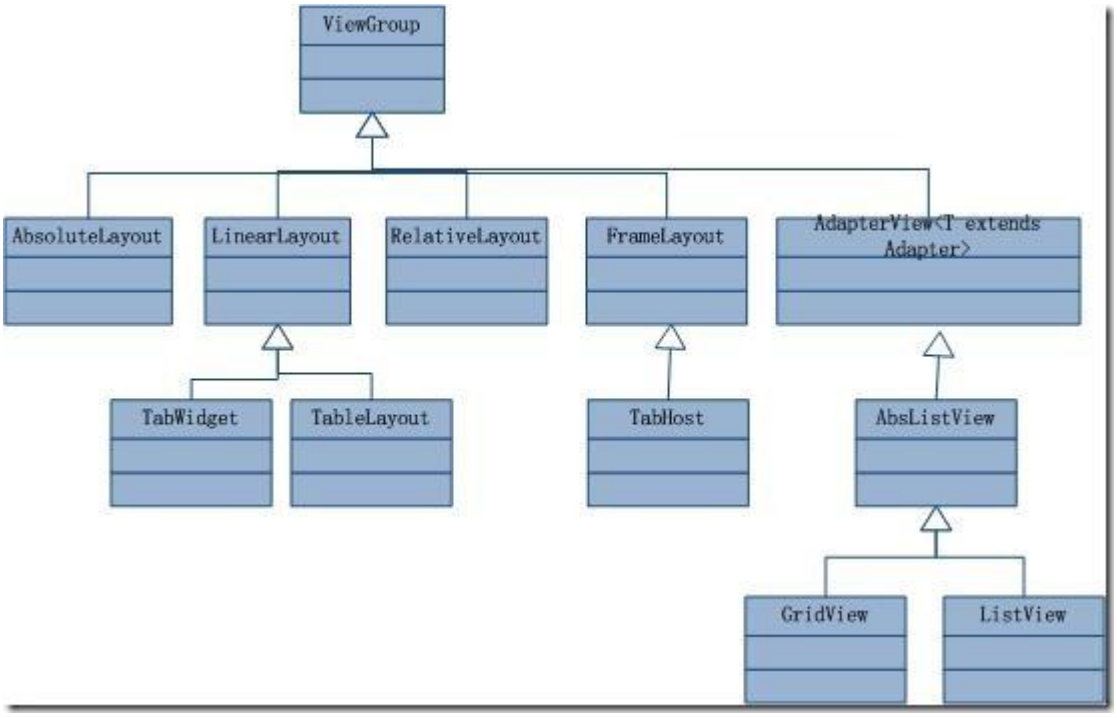


图 1、继承自 ViewGroup 的一些布局类

其实，所有的布局方式都可以归类为 ViewGroup 的 5 个类别，即 ViewGroup 的 5 个直接子类。其它的一些布局都扩展自这 5 个类。下面分小节分别介绍 View 的七种布局显示方式。

2、线性布局（Linear Layout）

线性布局：是一个 ViewGroup 以线性方向显示它的子视图（view）元素，即**垂直地**或**水平地**。之前我们的 Hello World！程序中 view 的布局方式就是线性布局的，一定不陌生！如下所示

res/layour/main.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal"><!-- have an eye on ! -->
    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button1"
        android:layout_weight="1"
        />
    <Button android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button2"
    android:layout_weight="1"
    />
    <Button android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button3"
    android:layout_weight="1"
    />
    <Button android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button4"
    android:layout_weight="1"
    />
    <Button android:id="@+id/button5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button5"
    android:layout_weight="1"
    />
</LinearLayout>
```

从上面可以看出根 LinearLayout 视图组（ViewGroup）包含 5 个 Button，它的子元素是以线性方式（horizontal，水平的）布局，运行效果如下图所示：

图 2、线性布局（水平或者说是横向）

如果你在 **android:orientation="horizontal"**设置为 **vertical**，则是是垂直或者说是纵向的，如下图所示：

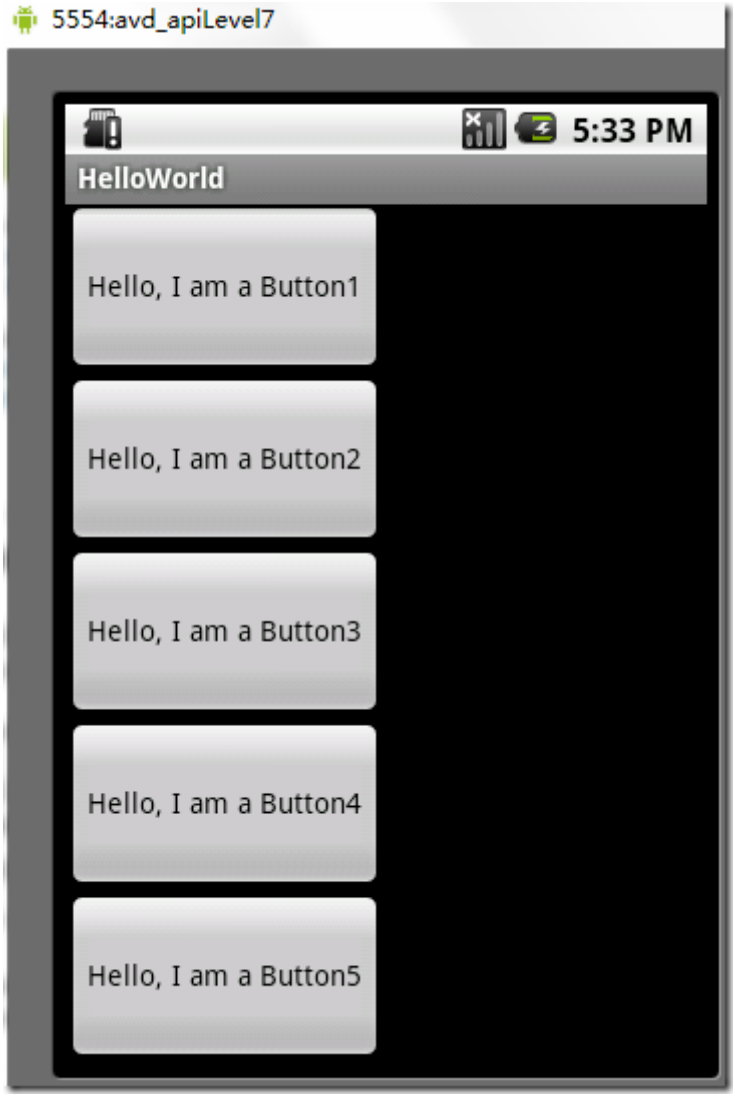


图 3、线性布局（垂直或者说是纵向）

2.1、Tips：android:layout_weight="1"

这个属性很关键，如果你没有显示设置它，它默认为 0。把上面布局文件（*水平显示的那个*）中的这个属性都去掉，运行会得出如下结果：

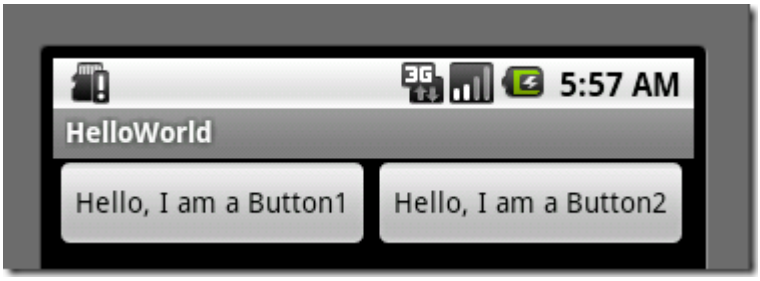


图 4、layout_weight 属性

没有了这个属性，我们本来定义的 5 个 Button 运行后却只显示了 2 个 Button，为什么呢？？

"weight"顾名思义是**权重**的意思，layout_weight 用于给一个线性布局中的诸多视图的重要程度赋值。所有的视图都有一个 layout_weight 值，默认为零，意思是需要显示多大的视图就占据多大的屏幕空间。这就不难解释为什么会造成上面的情况了：Button1~Button5 都设置了 layout_height 和 layout_width 属性为 wrap_content 即包住文字内容，他们都没有设置 layout_weight 属性，即默认为 0.，这样 Button1 和 Button2 根据需要的内容占据了整个屏幕，别的就显示不了啦！

若赋一个高于零的值，则将父视图中的可用空间分割，分割大小具体取决于每一个视图的 layout_weight 值以及该值在当前屏幕布局的整体 layout_weight 值和在其它视图屏幕布局的 layout_weight 值中所占的比率而定。举个例子：比如说我们在 水平方向上有一个文本标签和两个文本编辑元素。该文本标签并无指定 layout_weight 值，所以它将占据需要提供的最少空间。如果两个文本编辑元素每一个的 layout_weight 值都设置为 1，则两者平分在父视图布局剩余的宽度(因为我们声明这两者的重要度相等)。如果两个文本编辑元素其中第一个的 layout_weight 值设置为 1，而第二个的设置为 2，则剩余空间的三分之二分给第一个，三分之一分给第二个(**数值越小，重要度越高**)。

3、相对布局（Relative Layout）

相对布局：是一个 ViewGroup 以相对位置显示它的子视图（view）元素，一个视图可以指定相对于它的兄弟视图的位置（例如在给定视图的左边或者下面）或相对于 RelativeLayout 的特定区域的位置（例如底部对齐，或中间偏左）。

相对布局是设计用户界面的有力工具，因为它消除了嵌套视图组。如果你发现你使用了多个嵌套的 LinearLayout 视图组后，你可以考虑使用一个 RelativeLayout 视图组了。看下面的 res/layour/main.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
      android:id="@+id/label"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Type here:"/>
    <EditText
      android:id="@+id/entry"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="@android:drawable/editbox_background"
      android:layout_below="@id/label"/><!-- have an eye on ! -->
    <Button
      android:id="@+id/ok"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"   <!-- have an eye on ! -->
        android:layout_alignParentRight="true" <!-- have an eye on ! -->
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok" <!-- have an eye on ! -->
        android:layout_alignTop="@id/ok" <!-- have an eye on ! -->
        android:text="Cancel" />
</RelativeLayout>
```

从上面的布局文件我们知道，**RelativeLayout** 视图组包含一个 TextView、一个 EditView、两个 Button，注意标记了<!-- have an eye on ! -->（请注意运行代码的时候，请把这些注释去掉，否则会运行出错，上面加上是为了更加醒目！）的属性，在使用**相对布局**方式中就是使用这些类似的属性来定位视图到你想要的位置，它们的值是你参照的视图的 **id**。这些属性的意思很简单，就是英文单词的直译，就不多做介绍了。运行之后，得如下结果：

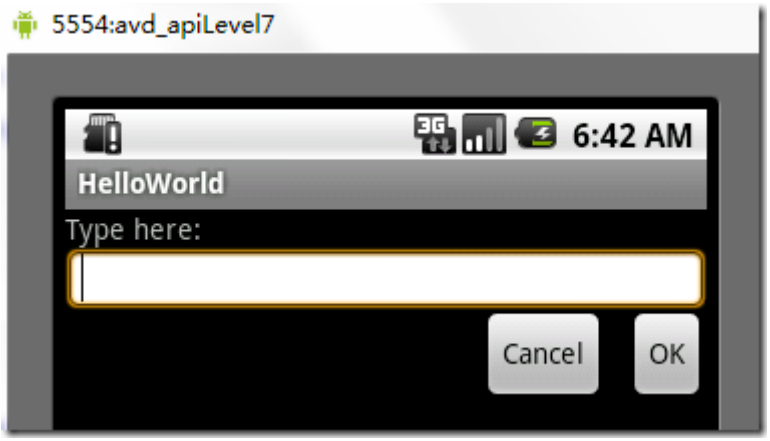


图 5、相对布局

4、 表格布局（Table Layout）

表格布局：是一个 ViewGroup 以表格显示它的子视图（view）元素，即行和列标识一个视图的位置。其实 Android 的表格布局跟 HTML 中的表格布局非常类似，TableRow 就像 HTML 表格的<tr>标记。

用表格布局需要知道以下几点：

- android:shrinkColumns，对应的方法：setShrinkAllColumns(boolean)，作用：设置表格的列是否收缩（列编号从 0 开始，下同），多列用逗号隔开（下同），如 android:shrinkColumns="0,1,2"，即表格的第 1、2、3 列的内容是收缩的以适合屏幕，不会挤出屏幕。
- android:collapseColumns，对应的方法：setColumnCollapsed(int,boolean)，作用：设置表格的列是否隐藏
- android:stretchColumns，对应的方法：setStretchAllColumns(boolean)，作用：设置表格的列是否拉伸

看下面的 res/layour/main.xml：

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

```
        android:shrinkColumns="0,1,2"><!-- have an eye on ! -->
    <TableRow><!-- row1 -->
    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button1"
        android:layout_column="0"
        />
     <Button android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button2"
    android:layout_column="1"
    />
    </TableRow>
    <TableRow><!-- row2 -->
    <Button android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button3"
        android:layout_column="1"
        />
<Button android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button4"
    android:layout_column="1"
    />
</TableRow>
<TableRow>
    <Button android:id="@+id/button5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button5"
    android:layout_column="2"
    />
</TableRow>
</TableLayout>
```

运行之后可以得出下面的结果：

图 6、表格布局

5、列表视图（List View）

列表布局：是一个 ViewGroup 以列表显示它的子视图（view）元素，列表是可滚动的列表。列表元素通过 ListAdapter 自动插入到列表。

ListAdapter：扩展自 Adapter，它是 ListView 和数据列表之间的桥梁。ListView 可以显示任何包装在 ListAdapter 中的数据。该类提供两个公有类型的抽象方法：

1. public abstract boolean areAllItemsEnabled () ：表示 ListAdapter 中的所有元素是否可激活的？如果返回真，即所有的元素是可选择的即可点击的。
2. public abstract boolean isEnabled (int position) ：判断指定位置的元素是否可激活的？

下面通过一个例子来，创建一个可滚动的列表，并从一个字符串数组读取列表元素。当一个元素被选择时，显示该元素在列表中的位置的消息。

1)、首先，将 res/layour/main.xml 的内容置为如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

这样就定义了元素在列表中的布局。

2)、src/skynet.com.cnblogs.www/HelloWorld.java 文件的代码如下：

```
package skynet.com.cnblogs.www;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;
```

```
public class HelloWorld extends ListActivity  {
    //注意这里 Helloworld 类不是扩展自 Acitvity，而是扩展自 ListAcitivty
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<String>(this, R.layout.main,
COUNTRIES));

        ListView lv = getListView();
        lv.setTextFilterEnabled(true);

        lv.setOnItemClickListener(new OnItemClickListener() {
          public void onItemClick(AdapterView<?> parent, View view,
              int position, long id) {
            // When clicked, show a toast with the TextView text
            Toast.makeText(getApplicationContext(), ((TextView)
view).getText(),
                Toast.LENGTH_SHORT).show();
          }
        });
    }
    static final String[] COUNTRIES = new String[] {
            "1", "2", "3", "4", "5",
            "6", "7", "8", "9", "10",
            "11", "12", "13", "14", "15",
            "16", "17", "18", "19", "20",
            "21", "22", "23", "24"
          };
}
```

Note：onCreate()函数中并不像往常一样通过 setContentView()为活动（Activity）加载布局文件，替代的是通过 setListAdapter(ListAdapter)自动添加一个 ListView 填充整个屏幕的 ListActivity。在此文件中这个方法以一个 ArrayAdapter 为参数：setListAdapter(new ArrayAdapter<String>(this, R.layout.main, COUNTRIES))，这个 ArrayAdapter 管理填入 ListView 中的列表元素。ArrayAdapter 的构造函数的参数为：this（表示应用程序的上下文 context）、表示 ListViewde 布局文件（这里是 R.layout.main）、插入 ListView 的 List 对象对数组（这里是 COUNTRES）。

setOnItemClickListener(OnItemClickListener)定义了每个元素的点击（on-click）的监听器，当 ListView 中的元素被点击时，onItemClick()方法被调用，在这里是即一个 Toast 消息——每个元素的位置将显示。

3)、运行应用程序得如下结果（点击 1 之后，在下面显示了 1）：

图 7、列表布局

NOTE:如果你改了 HelloWorld extends **ListActivity** 而不是 **Activity** 之后，运行程序是提示："Conversion to Dalvik format failed with error 1"。可以这么解决：解决办法是 Project > Clean... > Clean project selected below > Ok

5.1、一个小的改进

上面我们是把要填充到 ListView 中的元素硬编码到 HelloWorld.java 文件中，这样就缺乏灵活性！也不符合推荐的**应用程序的界面**与**控制它行为的代码**更好地分离的准则！

其实我们可以把要填充到 ListView 的元素写到 res/values/strings.xml 文件中的 <string-array>元素中，然后再源码中动态地读取。这样 strings.xml 的内容类似下面：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>1</item>
        <item>2</item>
        <item>3</item>
        <item>4</item>
        <item>5</item>
        <item>6</item>
        <item>7</item>
```

```
    </string-array>
</resources>
```

然而 HelloWorld.java 文件中的 onCreate()函数，则这样动态访问这个数组及填充到 ListVies：

```
String[] countries = getResources().getStringArray(R.array.countries_array);
setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item, countries));
```

5.2、补充说明

首先总结一下列表布局的关键部分：

- 布局文件中定义 ListView
- Adapter 用来将数据填充到 ListView
- 要填充到 ListView 的数据，这些数据可以字符串、图片、控件等等

其中 Adapter 是 ListView 和数据源之间的桥梁，根据数据源的不同 Adapter 可以分为三类：

- String[]: ArrayAdapter
- List<Map<String,?>>: SimpleAdapter
- 数据库 Cursor: SimpleCursorAdapter

使用 ArrayAdapter（数组适配器）顾名思义，需要把数据放入一个数组以便显示，上面的例子就是这样的；SimpleAdapter 能定义各种各样的布局出来，可以放上 ImageView（图片），还可以放上 Button（按钮），CheckBox（复选框）等等；SimpleCursorAdapter 是和数据库有关的东西。篇幅有限后面两种就不举例实践了。你可以参考 android ListView 详解 orArrayAdapter ,SimpleAdapter ,SimpleCursorAdapter 区别。

6、网格视图（Grid View）

网格布局：是一个 ViewGroup 以网格显示它的子视图（view）元素，即二维的、滚动的网格。网格元素通过 ListAdapter 自动插入到网格。ListAdapter 跟上面的列表布局是一样的，这里就不重复累述了。

下面也通过一个例子来，创建一个显示图片缩略图的网格。当一个元素被选择时，显示该元素在列表中的位置的消息。

1)、首先，将上面实践截取的图片放入 res/drawable/

2)、res/layour/main.xml 的内容置为如下：这个 GridView 填满整个屏幕，而且它的属性都很好理解，按英文单词的意思就对了。

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

3)、然后，HelloWorld.java 文件中 onCreate()函数如下：

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        GridView gridview = (GridView) findViewById(R.id.gridview);
```

```
        gridview.setAdapter(new ImageAdapter(this));

        gridview.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View v, int position,
long id) {
                Toast.makeText(HelloWorld.this, "  " + position,
Toast.LENGTH_SHORT).show();
            }
        });
    }
```

onCreate()函数跟通常一样，首先调用超类的 onCreate()函数函数，然后通过 setContentView()为活动（Activity）加载布局文件。紧接着是，通过 GridView 的 id 获取布局文件中的 gridview，然后调用它的 setListAdapter(ListAdapter)函数填充它，它的参数是一个我们自定义的 ImageAdapter。后面的工作跟列表布局中一样，为监听网格中的元素被点击的事件而做的工作。

4)、实现我们自定义 ImageAdapter，新添加一个类文件，它的代码如下：

```
package skynet.com.cnblogs.www;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
```

```
        if (convertView == null) {  // if it's not recycled, initialize some
attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = {
            R.drawable.linearlayout1, R.drawable.linearlayout2,
            R.drawable.linearlayout3, R.drawable.listview,
            R.drawable.relativelayout, R.drawable.tablelayout
    };
}
```

ImageAdapter 类扩展自 BaseAdapter，所以首先得实现它所要求必须实现的方法。构造函数和 getcount()函数很好理解，而 getItem(int)应该返回实际对象在适配器中的特定位置，但是这里我们不需要。类似地，getItemId(int)应该返回元素的行号，但是这里也不需要。这里重点要介绍的是 getView()方法，它为每个要添加到 ImageAdapter 的图片都创建了一个新的 View。当调用这个方法时，一个 View 是循环再用的，因此要确认对象是否为空。如果是空的话，一个 ImageView 就被实例化且配置想要的显示属性：

- setLayoutParams(ViewGroup.LayoutParams)：设置 View 的高度和宽度，这确保不管 drawable 中图片的大小，每个图片都被重新设置大小且剪裁以适应这些尺寸。
- setScaleType(ImageView.ScaleType)：声明图片应该向中心剪裁（如果需要的话）。
- setPadding(int, int, int, int)：定义补距，如果图片有不同的横纵比，小的补距将导致更多的剪裁以适合设置的 ImageView 的高度和宽度。

如果 View 传到 getView()不是空的，则本地的 ImageView 初始化时将循环再用 View 对象。在 getView()方法末尾，position 整数传入 setImageResource()方法以从 mThumbIds 数组中选择图片。

运行程序会得到如下结果（点击第一张图片之后）：

图 8、网格布局

7、绝对布局（AbsoluteLayout）

绝对布局：是一个 ViewGroup 以绝对方式显示它的子视图（view）元素，即以坐标的方式来定位在屏幕上位置。

这种布局方式很好理解，在布局文件或编程地设置 View 的坐标，从而绝对地定位。如下所示布局文件：

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/AbsoluteLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView android:id="@+id/txtIntro"
      android:text="绝对布局"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_x="20dip"<!-- have an eye on ! -->
      android:layout_y="20dip"><!-- have an eye on ! -->
    </TextView>
</AbsoluteLayout>
```

简单吧，这里不在深入了！

8、标签布局（Tab Layout）

标签布局：是一个 ViewGroup 以标签的方式显示它的子视图（view）元素，就像在 Firefox 中的一个窗口中显示多个网页一样。

为了狂创建一个标签 UI（tabbed UI），需要使用到 TabHost 和 TabWidget。TabHost 必须是布局的根节点，它包含为了显示标签的 TabWidget 和显示标签内容的 FrameLayout。
可以有两种方式实现标签内容：使用标签在同一个活动中交换视图、使用标签在完全隔离的活动之间改变。根据你的需要，选择不同的方式，但是如果每个标签提供不同的用户活动，为每个标签选择隔离的活动，因此你可以更好地以分离的组管理应用程序，而不是一个巨大的应用程序和布局。下面还有一个例子来创建一个标签 UI，每个标签使用隔离的活动。
1)、在项目中建立三个隔离的 Activity 类：ArtistisActivity、AlbumActivity、SongActivity。它们每个表示一个分隔的标签。每个通过 TextView 显示简单的一个消息，例如：

```
public class ArtistsActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView textview = new TextView(this);
        textview.setText("This is the Artists tab");
        setContentView(textview);
    }
}
```

其它两个类也类似。
2)、设置每个标签的图标，每个图标应该有两个版本：一个是选中时的，一个是未选中时的。通常的设计建议是，选中的图标应该是深色（灰色），未选中的图标是浅色（白色）。
现在创建一个 state-list drawable 指定哪个图标表示标签的状态：将图片放到 res/drawable 目录下并创建一个新的 XML 文件命名为 ic_tab_artists.xml，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- When selected, use grey -->
    <item android:drawable="@drawable/ic_tab_artists_grey"
          android:state_selected="true" />
    <!-- When not selected, use white-->
    <item android:drawable="@drawable/ic_tab_artists_white" />
</selector>
```

3)、res/layour/main.xml 的内容置为如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
```

```
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

这个布局将显示标签和提供上面创建的活动之间的导航。TabHost 要求包含一个 TabWidget 和一个 FrameLayout。TabWidget 和 FrameLayoutTabHost 以线性垂直地显示。

4)、HelloWorld.java 文件源码如下：

```
package skynet.com.cnblogs.www;

import android.widget.TabHost;
import android.app.TabActivity;
import android.content.Intent;
import android.content.res.Resources;
import android.os.Bundle;

public class HelloWorld extends TabActivity{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Resources res = getResources(); // Resource object to get Drawables
        TabHost tabHost = getTabHost();  // The activity TabHost
        TabHost.TabSpec spec;  // Resusable TabSpec for each tab
        Intent intent;  // Reusable Intent for each tab

        // Create an Intent to launch an Activity for the tab (to be reused)
        intent = new Intent().setClass(this, ArtistsActivity.class);

        // Initialize a TabSpec for each tab and add it to the TabHost
        spec = tabHost.newTabSpec("artists").setIndicator("Artists",
                          res.getDrawable(R.drawable.ic_tab_artists))
                      .setContent(intent);
        tabHost.addTab(spec);

        // Do the same for the other tabs
        intent = new Intent().setClass(this, AlbumsActivity.class);
        spec = tabHost.newTabSpec("albums").setIndicator("Albums",
                          res.getDrawable(R.drawable.ic_tab_artists))
                      .setContent(intent);
        tabHost.addTab(spec);
```

```
        intent = new Intent().setClass(this, SongsActivity.class);
        spec = tabHost.newTabSpec("songs").setIndicator("Songs",
                          res.getDrawable(R.drawable.ic_tab_artists))
                      .setContent(intent);
        tabHost.addTab(spec);

        tabHost.setCurrentTab(2);
    }
}
```

设置每个标签的文字和图标，并分配每个标签一个活动（这里为了方便三个标签都有相同的图标）。TabHost 的引用第一次通过 getTabHost()获取。然后，为每个标签，创建 TabHost.TabSpec 定义标签的属性。newTabSpec(String)方法创建一个新的 TabHost.TabSpec 以给定的字符串标识标签。调用 TabHost.TabSpec，setIndicator(CharSequence, Drawable)为每个标签设置文字和图标，调用 setContent(Intent)指定 Intent 去打开合适的活动。每个 TabHost.TabSpec 通过调用 addTab(TabHost.TabSpec)添加到 TabHost。
最后，setCurrentTab(int)设置打开默认显示的标签，通过索引标签的位置。
5)、打开 Android 的清单文件 AndroidManifest.xml，添加 NoTitleBar 主题到 HelloWorld 的<activity>标记。这将移除默认应用程序的标题和顶端布局，给标签腾出位置。<activity>标记应该像这样：

```
        <activity android:name=".HelloWorld"
                  android:label="@string/app_name"
                  android:theme="@android:style/Theme.NoTitleBar">
```

你运行这个程序能够得到什么结果呢？请自行检查。不过我在这里告诉你很有可能会运行不了，报“java.lang.NullPointerException”错！我想运行这个例子的很多人都会有这个问题，不信你试试！
PS：其实这也算是 Android 的一个 bug，而且这个 bug 在 2.2 中还没有解决，这个问题全球 N 多人都碰到了，并在 http://code.google.com/p/android/issues 中挂号了，相关问题的编号有不止一个。
接着往下看……
如果你看了我这篇文章，你一定会是个幸运儿！经过我艰苦的调试+找资料，我找到了解决方法：
在清单文件 AndroidManifest.xml，添加下面三个 Activity：

```
<activity android:name=".AlbumsActivity"   android:label="@string/app_name"></activity>
<activity android:name=".ArtistsActivity" android:label="@string/app_name"></activity>
<activity android:name=".SongsActivity"   android:label="@string/app_name"></activity>
```

现在运行可以看到如下结果：

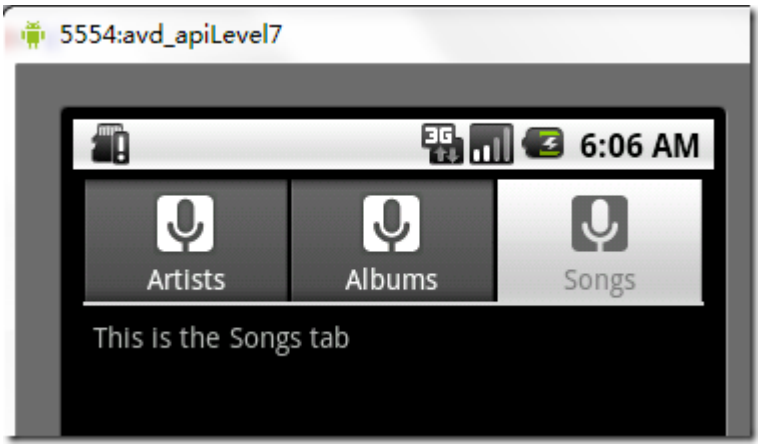


图 9、标签布局

Android 开发之旅：短信的收发及在 android 模拟器之间实践（一）

引言

本文通过运行两个 Android 模拟器，介绍在 Android 中如何实现短信服务（SMS，short message service）的功能。通过这个例子，我想带给大家的是：更加熟悉之前介绍过的 Android 应用程序的概念及技术细节，且通过实例调度大家的兴趣。我之所以选择 SMS 为例子，主要原因是 SMS 已经非常成熟了，从中可以发掘更多的信息和技术细节，而且我相信大部分人发短信比打电话多。

本文的主要内容如下：

1、温故知新

广播接收者：一个广播接收者是这样一个组件，它不做什么事，仅是接受广播公告并作出相应的反应。许多广播源自于系统代码，例如公告时区的改变、电池电量低、已采取图片、用户改变了语言偏好。应用程序也可以发起广播，例如为了他其他程序知道某些数据已经下载到设备且他们可以使用这些数据 BroadcastReceiver 类：是接受 sendBroadcast()发送的意图（intents）的基类。可以用 Context.registerReceiver()动态地注册这个类的实例，或者通过 AndroidManifest.xml 中<receiver>标签静态发布。

广播接收者不显示一个用户界面。然而，它们启动一个活动去响应收到的信息，或者他们可能使用 NotificationManager 去通知用户。通知可以使用多种方式获得用户的注意——闪烁的背光、振动设备、播放声音等等。典型的是放在一个持久的图标在状态栏，用户可以打开获取信息。

2、准备工作：SMS 涉及的主要类 SmsManager

实现 SMS 主要用到 SmsManager 类，该类继承自 java.lang.Object 类，下面我们介绍一下该类的主要成员。

公有方法：

- ArrayList<String> **divideMessage**(String text)

  当短信超过 SMS 消息的最大长度时，将短信分割为几块。

参数：*text*——初始的消息，不能为空
返回值：有序的 *ArrayList<String>*，可以重新组合为初始的消息

- static SmsManager **getDefault**()
  获取 SmsManager 的默认实例。
  返回值：*SmsManager* 的默认实例
- void **SendDataMessage**(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)
  发送一个基于 SMS 的数据到指定的应用程序端口。
  参数:
  1)、*destinationAddress*——消息的目标地址
  2)、*scAddress*——服务中心的地址 or 为空使用当前默认的 SMSC 3)*destinationPort*——消息的目标端口号
  4)、*data*——消息的主体，即消息要发送的数据
  5)、*sentIntent*——如果不为空，当消息成功发送或失败这个 PendingIntent 就广播。结果代码是 Activity.RESULT_OK 表示成功，或 RESULT_ERROR_GENERIC_FAILURE、RESULT_ERROR_RADIO_OFF、RESULT_ERROR_NULL_PDU 之一表示错误。对应 RESULT_ERROR_GENERIC_FAILURE，*sentIntent* 可能包括额外的“错误代码”包含一个无线电广播技术特定的值，通常只在修复故障时有用。
  每一个基于 SMS 的应用程序控制检测 *sentIntent*。如果 *sentIntent* 是空，调用者将检测所有未知的应用程序，这将导致在检测的时候发送较小数量的 SMS。
  6)、*deliveryIntent*——如果不为空，当消息成功传送到接收者这个 PendingIntent 就广播。
  异常：如果 *destinationAddress* 或 *data* 是空时，抛出 IllegalArgumentException 异常。
- void **sendMultipartTextMessage**(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliverIntents)
  发送一个基于 SMS 的多部分文本，调用者应用已经通过调用 **divideMessage**(String text)将消息分割成正确的大小。
  参数:
  1)、*destinationAddress*——消息的目标地址
  2)、*scAddress*——服务中心的地址 or 为空使用当前默认的 SMSC
  3)、*parts*——有序的 *ArrayList<String>*，可以重新组合为初始的消息
  4)、*sentIntents*——跟 **SendDataMessage** 方法中一样，只不过这里的是一组 PendingIntent
  5)、*deliverIntents*——跟 **SendDataMessage** 方法中一样，只不过这里的是一组 PendingIntent
  异常：如果 *destinationAddress* 或 *data* 是空时，抛出 IllegalArgumentException 异常。
- void **sendTextMessage**(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)
  发送一个基于 SMS 的文本。参数的意义和异常前面的已存在的一样，不再累述。

常量：

- public static final int **RESULT_ERROR_GENERIC_FAILURE**
  表示普通错误，值为 1(0x00000001)
- public static final int **RESULT_ERROR_NO_SERVICE**
  表示服务当前不可用，值为 4 (0x00000004)
- public static final int **RESULT_ERROR_NULL_PDU**
  表示没有提供 pdu，值为 3 (0x00000003)
- public static final int **RESULT_ERROR_RADIO_OFF**
  表示无线广播被明确地关闭，值为 2 (0x00000002)

- public static final int **STATUS_ON_ICC_FREE**
  表示自由空间，值为 0 (0x00000000)
- public static final int **STATUS_ON_ICC_READ**
  表示接收且已读，值为 1 (0x00000001)
- public static final int **STATUS_ON_ICC_SENT**
  表示存储且已发送，值为 5 (0x00000005)
- public static final int **STATUS_ON_ICC_UNREAD**
  表示接收但未读，值为 3 (0x00000003)
- public static final int **STATUS_ON_ICC_UNSENT**
  表示存储但为发送，值为 7 (0x00000007)

3、简单的 SMS 发送程序

1)、首先，编辑布局文件 res/layout/main.xml，达到我们想要的结果，界面如下：

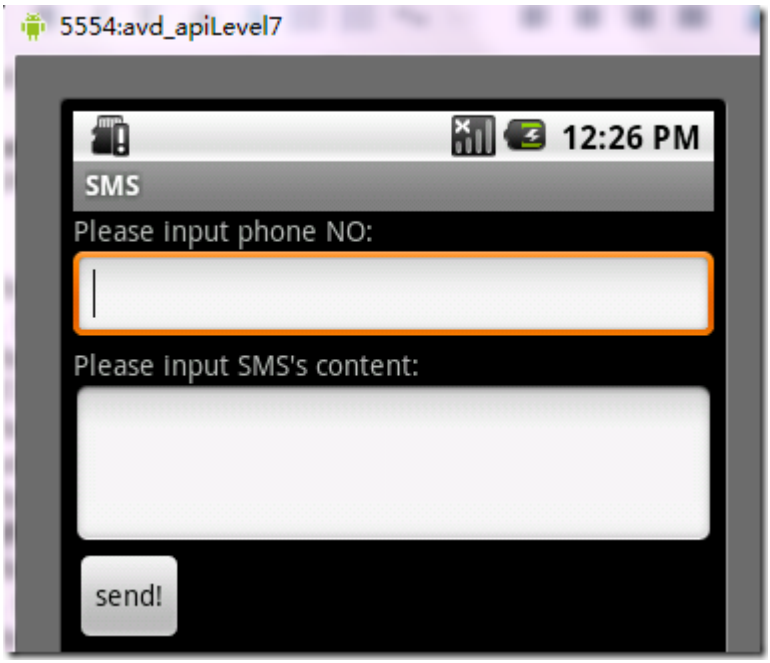


图 1、程序运行界面

对应的 xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
        android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:text="@string/txtPhoneNo"/>
    <!-- text's value define in res/values/strings.xml -->

    <EditText android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:id="@+id/edtPhoneNo"/>

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

```
                android:text="@string/txtContent"/>

    <EditText android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:minLines="3"
                android:id="@+id/edtContent"/>

    <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/btnText"
                android:id="@+id/btnSend"/>
</LinearLayout>
```

相应的要在 res/values/strings.xm 中添加上面定义的视图的 text 的值，如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="txtPhoneNo">Please input phone NO:</string>
    <string name="txtContent">Please input SMS\'s content:</string>
    <string name="btnText">send!</string>
    <string name="app_name">SMS</string>
</resources>
```

2)、做完这些准备工作之后，我么要开始编写代码实现简单的短信发送了。

通过第一步我们构建好界面之后，现在要在上面的基础上编写业务逻辑了。大致过程为：在 java 源文件中，获取用户在 edtPhoneNo 中输入的电话号码，edtContent 中输入要发送的内容；然后点击 btnSend 按钮发送短信，要达到这个目的我们要设置 btnSend 的 OnClickListener 以达到当点击它触发发送短信的功能，而且要发送短信就要用到我们前面介绍的 SmsManager 类提供的方法接口。

设置 btnSend 的 OnClickListener 的代码如下：

```
btnSend.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
                String phoneNo = edtPhoneNo.getText().toString();
                String message = edtContent.getText().toString();
                if (phoneNo.length() > 0 && message.length() > 0){
                        //call sendSMS to send message to phoneNo
                        sendSMS(phoneNo, message);
                }
                else
                        Toast.makeText(getBaseContext(),
                                "Please enter both phone number and message.",
                                Toast.LENGTH_SHORT).show();
        }
});
```

发送短信的功能的代码如下：

```
private void sendSMS(String phoneNumber, String message) {
        // ---sends an SMS message to another device---
        SmsManager sms = SmsManager.getDefault();
        PendingIntent pi = PendingIntent.getActivity(this, 0,
                new Intent(this,TextMessage.class), 0);
        //if message's length more than 70 ,
        //then call divideMessage to dive message into several part
        //and call sendTextMessage()
        //else direct call sendTextMessage()
        if (message.length() > 70) {
                ArrayList<String> msgs = sms.divideMessage(message);
                for (String msg : msgs) {
                        sms.sendTextMessage(phoneNumber, null, msg, pi, null);
                }
        } else {
                sms.sendTextMessage(phoneNumber, null, message, pi, null);
        }
        Toast.makeText(TextMessage.this, "短信发送完成",
Toast.LENGTH_LONG).show();
}
```

如果你已经看了第 2 节介绍的 SmsManager 类的介绍，代码应该很好理解。在这里要说明的是，sendTextMessage 方法中的第 4 个和第 5 个参数 PendingIntent 设为 null，这样的话不能根据短信发出之后的状态做相应的事情，如短信发送失败后的提醒、接收者成功接收后的回执.....完整的流程源码如下：

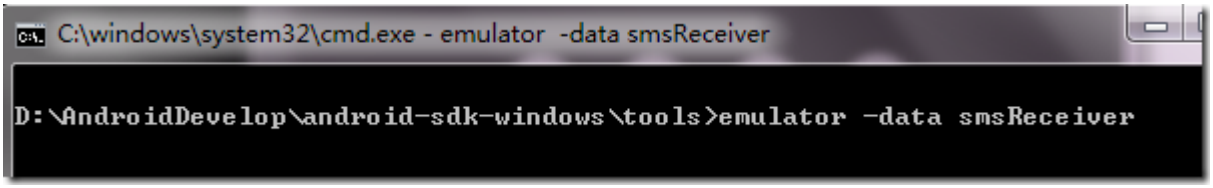
⊞⊟TextMessage.java 源文件全部代码

3）运行前，还要在清单文件 AndroidManifest.xml 中加入允许发送短信的权限：

⊞⊟AndroidManifest.xml

3.1、运行 SMS 程序给另一个 android 模拟器发短信

运行上面我们编写的 TextMessage 程序，另外在 Windows 的命令行下切换到 tools 目录下，并输入 emulator –data smsReceiver，我的如下：



这样就会启动一个 android 模拟器，如下所示：（**注意它的编号：5556，就是用这个编号与它通信的**）

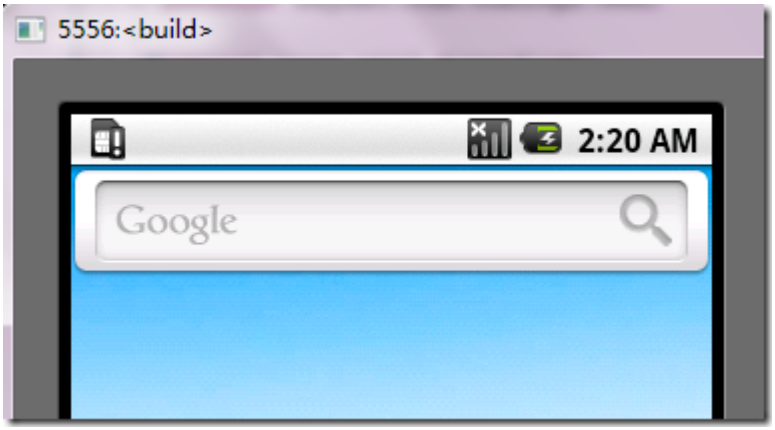


图 2、通过 emulator 启动一个 android 模拟器

通过我们 TextMessage 程序启动的 android 模拟器，编写短信：

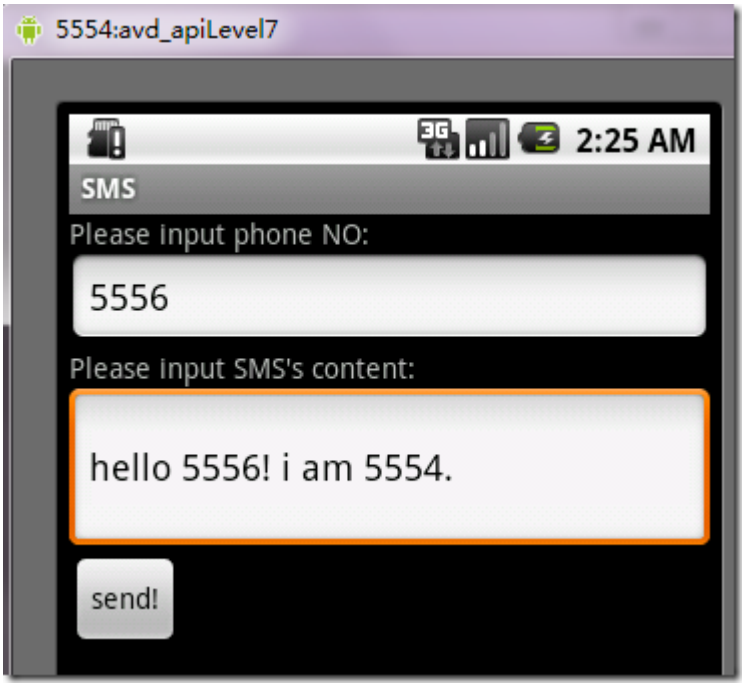


图 3、TextMessage 程序个 5556 模拟器发短信

点击发送之后，通过命令行启动的 5556 号 android 模拟器会收到我们刚才发送的短信，如下所示：

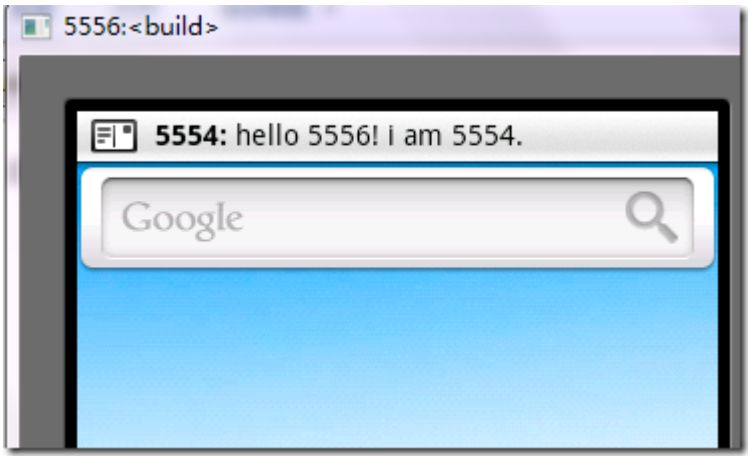


图 4、收到短信的提示

tips：

如果通过命令行的 emulator 启动 android 模拟器提示“NO DNS servers found！”，这时我们发的短信模拟器是收不到的。

- 在 Windows 下，如果电脑没有介入网络，即找不 DNS 服务器的话会出现这种情况！
- 在 Mac 下，如果提示这个警告的话，可以这样解决：检查你是否有/etc/resolv.conf 文件，如果没有的话，通过下面的命令行
  ln -s /private/var/run/resolv.conf /etc/resolv.conf 可以解决。

4、SMS 增强（一）

上面我们实现了一个简单的 SMS 程序，下面我们要对它进行增强！你肯定已经注意到了，我们上面的 SMS 程序的 sendTextMessage 方法中的第 4 个和第 5 个参数 PendingIntent 设为 null，即 sentIntent 和 deliveryIntent。

第 4 个参数-sendIntent，当消息成功发送或发送失败都将被触发。广播接收者的结果码，Activity.RESULT_OK 表示成功，或 RESULT_ERROR_GENERIC_FAILURE、RESULT_ERROR_RADIO_OFF、RESULT_ERROR_NULL_PDU 之一表示错误。对应 RESULT_ERROR_GENERIC_FAILURE，*sentIntent* 可能包括额外的“错误代码”包含一个无线电广播技术特定的值，通常只在修复故障时有用。第 5 个参数-deliveryIntent，仅当目标接收到你的 SMS 消息才触发。

为了跟踪发出的短信的状态，实现和注册 Broadcast Receiver（广播接收者）监听传递给 sendTextMessage 方法的参数 Pending Intents。下面我们就实现和注册这个广播接收者：

```
String SENT_SMS_ACTION="SENT_SMS_ACTION";
String DELIVERED_SMS_ACTION="DELIVERED_SMS_ACTION";

//create the sentIntent parameter
Intent sentIntent=new Intent(SENT_SMS_ACTION);
PendingIntent sentPI=PendingIntent.getBroadcast(
                this,
                0,
                sentIntent,
                0);

//create the deilverIntent parameter
Intent deliverIntent=new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI=PendingIntent.getBroadcast(
                this,
                0,
                deliverIntent,
                0);

//register the Broadcast Receivers
registerReceiver(new BroadcastReceiver(){
        @Override
        public void onReceive(Context _context,Intent _intent)
        {
                switch(getResultCode()){
                        case Activity.RESULT_OK:
                                Toast.makeText(getBaseContext(),
                                                "SMS sent success actions",
                         Toast.LENGTH_SHORT).show();
                                break;
                        case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
```

```
                                Toast.makeText(getBaseContext(),
                                        "SMS generic failure actions",
                                 Toast.LENGTH_SHORT).show();
                                        break;
                                case SmsManager.RESULT_ERROR_RADIO_OFF:
                                        Toast.makeText(getBaseContext(),
                                        "SMS radio off failure actions",
                                 Toast.LENGTH_SHORT).show();
                                        break;
                                case SmsManager.RESULT_ERROR_NULL_PDU:
                                        Toast.makeText(getBaseContext(),
                                        "SMS null PDU failure actions",
                                 Toast.LENGTH_SHORT).show();
                                        break;
                        }
                }
        },
new IntentFilter(SENT_SMS_ACTION));
registerReceiver(new BroadcastReceiver(){
        @Override
        public void onReceive(Context _context,Intent _intent)
        {
                Toast.makeText(getBaseContext(),
                                "SMS delivered actions",
                 Toast.LENGTH_SHORT).show();
        }
},
new IntentFilter(DELIVERED_SMS_ACTION));
```

在基本完成了要做的工作，接下来要做的就是将 sendTextMessage 的第 4 个和第 5 个参数改为 sentPI、deliverPI，这样工作基本完成，修改后的 sendSMS 方法如下：

⊞⊟修改后的 sendSMS 方法完整代码

运行之后的，发送短信成功的话就可以看到如下界面：

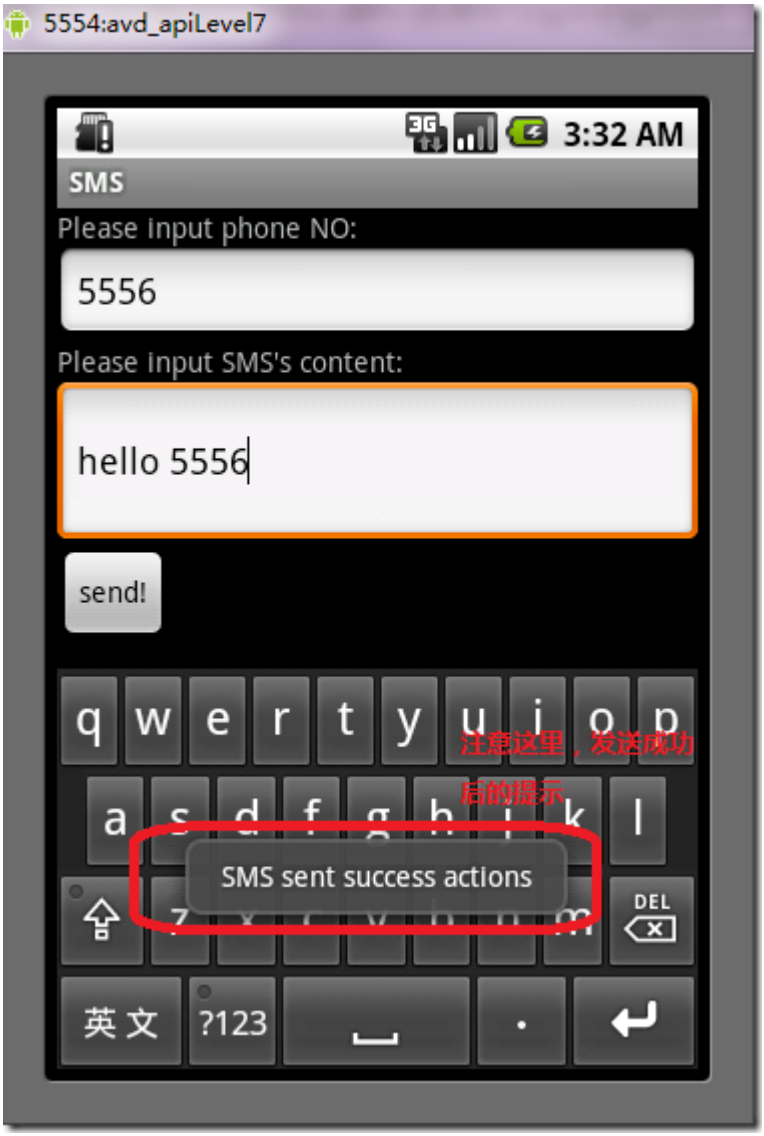


图 5、增强 SMS（一）

5、SMS 增强（二）

下面这个增强是使 SMS 能够发送二进制数据。要发送数据要使用 SmsManager 类的 sendDataMessage 方法，跟 sendTextMessage 方法类似，只不过该方法多了一个目标端口的参数，构建该 SMS 的过程跟前面的类似这里就不在累述。

Android 开发之旅：短信的收发及在 android 模拟器之间实践（二）

2010-07-11 12:36 by 吴秦, 2259 visits, 网摘, 收藏, 编辑

引言

前面我们介绍都只是如何发送 SMS 消息，接下来我们介绍如何接收 SMS 消息，及另一种发短信的方式并增强为可以发生图片等，最后介绍一下 emulator 工具。本文的主要内容如下：

- 1~5 见 Android 开发之旅：短信的收发及在 android 模拟器之间实践（一）
- 6、温故知新之 Intent
- 7、准备工作：SmsMessage 类
- 8、SMS 接收程序
- 9、另一种发送短信的方式：使用 Intent
- 10、增强 SMS 为 MMS

6、温故知新之 Intent

此系列前面简单地接受过意图（Intent），这里再次简单介绍一下，在短信接收程序和使用 Intent 发送 SMS 中我们要用到。android 应用程序的三大组件——Activities、Services、Broadcast Receiver，通过消息触发，这个消息就称作意图（Intent）。下面以 Acitvity 为例，介绍一下 Intent。Android 用 Intent 这个特殊的类实现在 Activity 与 Activity 之间的切换。**Intent 类用于描述应用的功能。在 Intent 的描述结构中，有两个最重要的部分：动作和动作对应的数据。**典型的动作类型有 MAIN、VIEW、PICK、EDIT 等，我们在短信接收程序中就用到从广播意图中提取动作类型并判断是否是"android.provider.Telephony.SMS_RECEIVED"，进而作深一步的处理。而动作对应的数据则以 URI 的形式表示。例如，要查看一个人的联系方式，需要创建一个动作为 VIEW 的 Intent，以及表示这个人的 URI。

通过解析各种 Intent，从一个屏幕导航到另一个屏幕是很简单的。当向前导航时，Activity 将会调用 startActivity("指定一个 Intent")方法。然后，系统会在所有已安装的应用程序中定义的 IntentFilter 中查找，找到最匹配的 Intent 对应的 Activity。新的 Activity 接收到指定的 Intent 的通知后，开始运行。当 startActivity()方法被调用时，将触发解析指定 Intent 的动作，该机制提供了两个关键的好处：

- Activity 能够重复利用从其他组件中以 Intent 形式产生的请求。
- Activity 可以在任何时候被具有相同 IntentFilter 的新的 Activity 取代。

7、准备工作：SmsMessage 类

顾名思义，SmsMessage 类是一个表示短信的类，为了更好地了解 Android 的短信机制及以后更好地编写短信相关程序，这里介绍一下该类的公有方法和常量，及嵌套枚举、类成员。

公有方法：

- public static int[] calculateLength (CharSequence msgBody, boolean use7bitOnly)

  参数:

  msgBody-要封装的消息、*use7bitOnly*-如果为 TRUE，不是广播特定 7-比特编码的部分字符被认为是单个空字符；如果为 FALSE，且 *msgBody* 包含非 7-比特可编码字符，长度计算使用 16-比特编码。

  返回值:

  返回一个 4 个元素的 int 数组，int[0]表示要求使用的 SMS 数量、int[1]表示编码单元已使用的数量、int[2]表示剩余到下个消息的编码单元数量、int[3]表示编码单元大小的指示器。
- public static int[] calculateLength (String messageBody, boolean use7bitOnly)

  *参数*和*返回值*跟上面类似
- public static SmsMessage createFromPdu (byte[] pdu)

  从原始的 PDU（protocol description units）创建一个 SmsMessage。这个方法很重要，在我们编写短信接收程序要用到，它从我们接收到的广播意图中获取的字节创建 SmsMessage。
- public String getDisplayMessageBody()

  返回短信消息的主体，或者 Email 消息主体（如果这个消息来自一个 Email 网关）。如果消息主体不可用，返回 null。这个方法也很重要，在我们编写短信接收程序也要用到。
- public String getDisplayOriginatingAddress ()

  返回信息来源地址，或 Email 地址（如果消息来自 Email 网关）。如果消息主体不可用，返回 null。这个方法在来电显示，短信接收程序中经常用到。
- public String getEmailBody ()

  如果 isEmail 为 TRUE，即是邮件，返回通过网关发送 Email 的地址，否则返回 null。
- public int getIndexOnIcc ()

  返回消息记录在 ICC 上的索引（从 1 开始的）
- public String getMessageBody ()

  以一个 String 返回消息的主体，如果它存在且是基于文本的。
- public SmsMessage.MessageClass getMessageClass ()

  返回消息的类。

- public String getOriginatingAddress ()
  以 String 返回 SMS 信息的来电地址，或不可用时为 null。
- public byte[] getPdu ()
  返回消息的原始 PDU 数据。
- public int getProtocolIdentifier ()
  获取协议标识符。
- public String getPseudoSubject ()
- public String getServiceCenterAddress ()
  返回转播消息 SMS 服务中心的地址，如果没有的话为 null。
- public int getStatus ()
  GSM：为一个 SMS-STATUS-REPORT 消息，它返回状态报告的 status 字段。这个字段表示之前提交的 SMS 消息的状态。
  CDMA：为不影响来自 GSM 的状态码，值移动到 31-16 比特。这个值由一个 error 类（25-16 比特）和一个状态码（23-16 比特）组成。
  如果是 0，表示之前发送的消息已经被收到。
- public int getStatusOnIcc ()
  返回消息在 ICC 上的状态（已读、未读、已发送、未发送）。有下面的几个值：SmsManager.STATUS_ON_ICC_FREE、SmsManager.STATUS_ON_ICC_READ、SmsManager.STATUS_ON_ICC_UNREAD、SmsManager.STATUS_ON_ICC_SEND、SmsManager.STATUS_ON_ICC_UNSENT 这几个值在上篇的 SmsManager 类介绍有讲到。
- public static SmsMessage.SubmitPdu getSubmitPdu (
  String scAddress, String destinationAddress,
  short destinationPort, byte[] data,
  boolean statusReportRequested)
  参数：*scAddress* - 服务中心的地址（Sercvice Centre address，为 null 即使用默认的）、*destinationAddress* - 消息的目的地址、*destinationPort*- 发送消息到目的的端口号、*data* - 消息数据。
  返回值：一个包含编码了的 SC 地址（如果指定了的话）和消息内容的 SubmitPdu，否则返回 null，如果编码错误。
- public static SmsMessage.SubmitPdu getSubmitPdu (
  String scAddress, String destinationAddress,
  String message, boolean statusReportRequested)
  和上面类似。
- public static int getTPLayerLengthForPDU (String pdu)
  返回指定 SMS-SUBMIT PDU 的 TP-Layer-Length，长度单位是字节而不是十六进字符。
- public long getTimestampMillis ()
  以 currentTimeMillis()格式返回服务中心时间戳。
- public byte[] getUserData ()
  返回用户数据减去用户数据头部（如果有的话）
- public boolean isCphsMwiMessage ()
  判断是否是 CPHS MWI 消息
- public boolean isEmail ()
  判断是否是 Email，如果消息来自一个 Email 网关且 Email 发送者（sender）、主题（subject）、解析主体（parsed body）可用，则返回 TRUE。
- public boolean isMWIClearMessage ()
  判断消息是否是一个 CPHS 语音邮件或消息等待 MWI 清除（clear）消息。

- public boolean isMWISetMessage ()
  判断消息是否是一个 CPHS 语音邮件或消息等待 MWI 设置（set）消息。
- public boolean isMwiDontStore ()
  如果消息是一个“Message Waiting Indication Group：Discard Message”通知且不应该保存，则返回 TRUE，否则返回 FALSE。
- public boolean isReplace ()
  判断是否是一个“replace short message”SMS
- public boolean isReplyPathPresent ()
  判断消息的 TP-Reply-Path 位是否在消息中设置了。
- public boolean isStatusReportMessage ()
  判断是否是一个 SMS-STATUS-REPORT 消息。

常量值：

- public static final int ENCODING_16BIT ：值为 3(0x00000003)
- public static final int ENCODING_8BIT ：值为 2 (0x00000002)
- public static final int ENCODING_UNKNOWN ：值为 0 (0x00000000) ，用户数据编码单元的大小。
- public static final int MAX_USER_DATA_BYTES ：值为 140 (0x0000008c)，表示每个消息的最大负载字节数。
- public static final int MAX_USER_DATA_BYTES_WITH_HEADER ：134 (0x00000086)，如果一个用户数据有头部，该值表示它的最大负载字节数，该值假定头部仅包含 CONCATENATED_8_BIT_REFENENCE 元素。
- public static final int MAX_USER_DATA_SEPTETS ：值为 160 (0x000000a0) ，表示每个消息的最大负载 septets 数。
- public static final int MAX_USER_DATA_SEPTETS_WITH_HEADER ：值为 153 (0x00000099)，如果存在用户数据头部，则该值表示最大负载 septets 数该值假定头部仅包含 CONCATENATED_8_BIT_REFENENCE 元素。

嵌套枚举成员 SmsMessage.MessageClass 的枚举值：

- public static final SmsMessage.MessageClass CLASS_0
- public static final SmsMessage.MessageClass CLASS_1
- public static final SmsMessage.MessageClass CLASS_2
- public static final SmsMessage.MessageClass CLASS_3
- public static final SmsMessage.MessageClass CLASS_UNKNOWN

嵌套枚举成员 SmsMessage.MessageClass 的公有方法：

- public static SmsMessage.MessageClass valueOf (String name)：返回值的字符串的值
- public static final MessageClass[] values ()：返回 MessageClass 的值数组

嵌套类成员 SmsMessage.SubmitPdu 的字段：

- public byte[] encodedMessage ：编码了的消息
- public byte[] encodedScAddress ：编码的服务中心地址

嵌套类成员 SmsMessage.SubmitPdu 的公有方法：

- public String toString ()
  返回一个包含简单的、可读的这个对象的描述字符串。鼓励子类去重写这个方法，并提供实现对象的类型和数据。默认实现简单地连接类名、@、十六进制表示的对象哈希码，即下面的形式：
  getClass().getName() + '@' + Integer.toHexString(hashCode())

8、SMS 接收程序

当一个 SMS 消息被接收时，一个新的广播意图由 android.provider.Telepony.SMS_RECEIVED 动作触发。注意：这个一个字符串字面量（string literal），但是 SDK 当前并没有包括这个字符串的引用，因此当要在应用程序中使用它时必须自己显示的指定它。现在我们就开始构建一个 SMS 接收程序：

1)、跟 SMS 发送程序类似，要在清单文件 AndroidManifest.xml 中指定权限允许接收 SMS：
<uses-permission android:name="android.permission.RECEIVER_SMS"/>
为了能够回发短信，还应该加上发送的权限。
2)、应用程序监听 SMS 意图广播，SMS 广播意图包含了到来的 SMS 细节。我们要从其中提取出 SmsMessage 对象，这样就要用到 pdu 键提取一个 SMS PDUs 数组（protocol description units—封装了一个 SMS 消息和它的元数据），每个元素表示一个 SMS 消息。为了将每个 PDU byte 数组转化为一个 SMS 消息对象，需要调用 SmsMessage.createFromPdu。
每个 SmsMessage 包含 SMS 消息的详细信息，包括起始地址（电话号码）、时间戳、消息体。下面编写一个接收短信的类 SmsReceiver 代码如下：

上面代码的功能是从接收到的广播意图中提取来电号码、短信内容，然后将短信加上@echo 头部回发给来电号码，并在屏幕上显示一个 Toast 消息提示成功。

9、另一种发送短信的方式：使用 Intent

上篇我们使用 SmsManager 类实现了发送 SMS 的功能，且并没有用到内置的客户端。实际上，我们很少这样做，自己在应用程序中去完全实现一个完整的 SMS 客户端。相反我们会去利用它，将需要发送的内容和目的手机号传递给内置的 SMS 客户端，然后发送。
下面我就向大家介绍如何利用 Intent 实现利用将我们的东西传递给内置 SMS 客户端发送我们 SMS。为了实现这个功能，就要用到 startActivity("指定一个 Intent")方法，且指定 Intent 的动作为 Intent.ACTION_SENDTO，用 sms：指定目标手机号，用 sms_body 指定信息内容。java 源文件如下所示：

```
⊞⊟java 源文件完整代码
```

注意代码中的红色粗体部分，就是实现这个功能的核心代码！布局文件 maim.xml 和值文件 string.xml 跟上篇中的一样，这里不再累述。运行结果如下图：

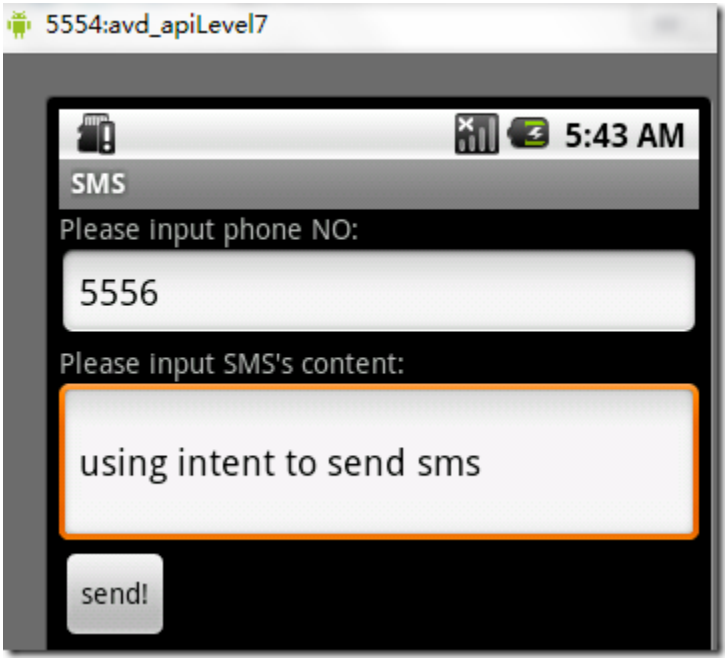


图 2、程序主界面

点击 send 按钮之后，转到内置的 SMS 客户端并且将我们输入的值传入了，如下图：

图 3、内容传至内置 SMS 客户端

发送之后，5556 号 android 模拟器会收到我们发送的消息，如下图：

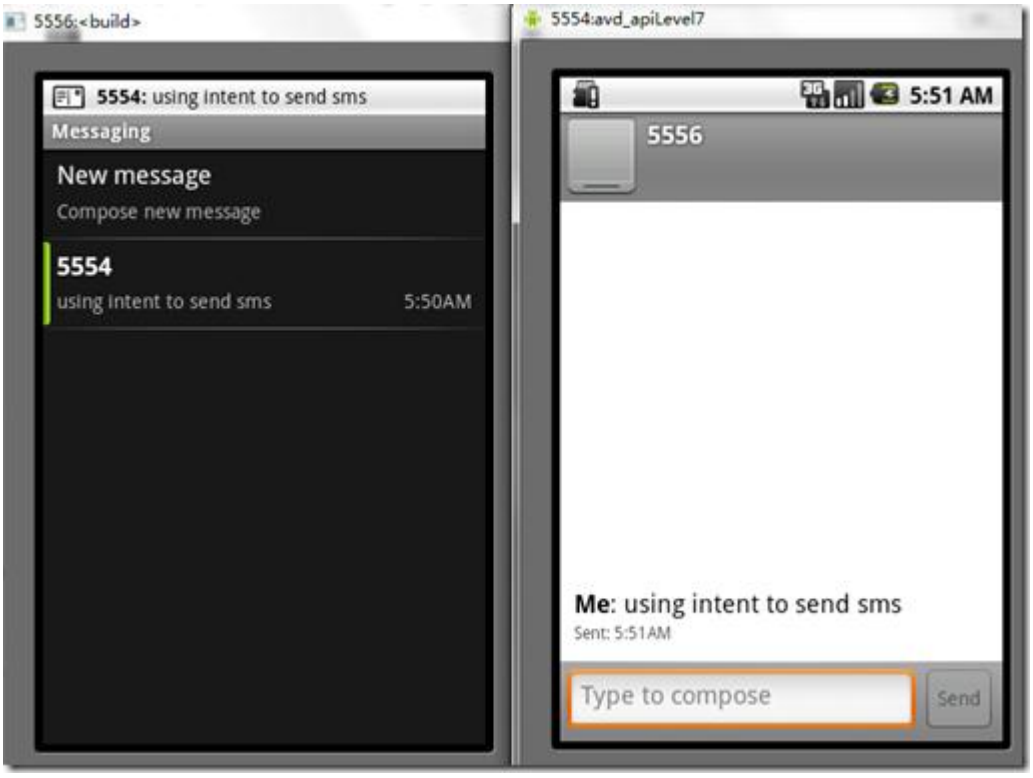


图 5、发送之后 5556 号 android 模拟器收到消息

10、增强 SMS 为 MMS

我们讲了这么多，都还只是实现了简单的发生 SMS 的功能，如果我们想发送图片、音频怎么办(⊙o⊙)？不急，现在我们就将第 9 节介绍的 SMS 发送程序改造为 MMS。

我们可以附加一个文件到我们的消息做为附件发送，用 Intent.EXTRA_STREAM 和附件资源的 Uri 做为参数调用 putExtra()方法，附加到信息。并设置 Intent 的类型为 mime-type。**要注意的是：内置的 MMS 并不包括一个 ACTION_SENDTO 动作的 Intent 接收器，我们需要使用的动作类型是 ACTION_SEND，并且目标手机号不在是使用 sms：而是 address。**主要代码如下：

```
// Get the URI of a piece of media to attach.
Uri attached_Uri = Uri.parse("content://media/external/images/media/1");
// Create a new MMS intent
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attached_Uri);
mmsIntent.putExtra("sms_body", edtContent.getText().toString());
mmsIntent.putExtra("address", edtPhoneNo.getText().toString());
mmsIntent.putExtra(Intent.EXTRA_STREAM, attached_Uri);
mmsIntent.setType("image/png");
startActivity(mmsIntent);
```

将这段代码替换第 9 节中的红色粗体代码，就完成而来一个 MMS 的构建。

PS：这篇文章本应该很早就该发出来了，在 6 月 20 号就写好了，但由于现在的工作环境完全隔离了 Internet，家里又还没有开通网络；还有一个原因是现在工作比较忙！请大家见谅，能够继续支持我，让我有动力写下去。还有一点，这篇文章在我电脑上放久了，对当时的状态有些忘了，不知道文中有什么遗漏和错误，请大家指出！

Android 开发之旅: Intents 和 Intent Filters（理论部分）

2010-07-20 20:07 by 吴秦, 2217 visits, 网摘, 收藏, 编辑

引言

大部分移动设备平台上的应用程序都运行在他们自己的沙盒中。他们彼此之间互相隔离，并且严格限制应用程序与硬件和原始组件之间的交互。 我们知道交流是多么的重要，作为一个孤岛没有交流的东西，一定毫无意义！Android 应用程序也是一个沙盒，但是他们能够使用 Intent、Broadcast Receivers、Adapters、Content Providers、Internet 去突破他们的边界互相交流。有交流还会和谐，由此可见这些交流手段有多重要。

上篇文章中我们在 SMS 接收程序和使用 Intent 发送 SMS 程序中用到了 Intent，并做了简单的回顾和总结：android 应用程序的三大组件——Activities、Services、Broadcast Receiver，通过消息触发，这个消息就称作意图（Intent）。然后以活动为例简单介绍了 Intent 了并说明 Intent 机制的好处。既然在 SMS 程序中用到了 Intent，这里我就借机顺着这条线，彻底详细地介绍一下 Intent。分两篇文章介绍：

1. Android 开发之旅: Intents 和 Intent Filters（理论部分）
2. Android 开发之旅: Intents 和 Intent Filters（实例部分）

本文的主要内容如下：

1、概述

一个应用程序的三个核心组件——activities、services、broadcast receivers，都是通过叫做 intents 的消息激活。Intent 消息是一种同一或不同应用程序中的组件之间延迟运行时绑定的机制。intent 本身（是一个 Intent 对象），是一个被动的数据结构保存一个将要执行的操作的抽象描述，或在广播的情况下，通常是某事已经发生且正在宣告。对于这三种组件，有独立的传送 intent 的机制：

- **Activity**：一个 intent 对象传递给 Context.startActivity()或 Activity.startActivityForRestult()去启动一个活动或使一个已存在的活动去做新的事情。
- **Service**：一个 intent 对象传递给 Context.startService()去初始化一个 service 或传递一个新的指令给正在运行的 service。类似的，一个 intent 可以传递给 Context.bindService()去建立调用组件和目标服务之间的连接。
- **Broadcast Receiver**：一个 intent 对象传递给任何广播方法（如 Context.sendBroadcast()，Context.sendOrderedBroadcast()，Context.sendStickyBroadcast()），都将传递到所有感兴趣的广播接收者。

在每种情况下，Android 系统查找合适的 activity、service、broadcast receivers 来响应意图，如果有必要的话，初始化他们。这些消息系统之间没有重叠，即广播意图仅会传递给广播接收者，而不会传递活动或服务，反之亦然。

下面首先描述 intent 对象，然后介绍 Android 将 intent 映射到相应组件的规则——如何解决哪个组件应该接收 intent 消息。对于没有指定目标组件名字的 intent，这个处理过程包括按照 intent filters 匹配每个潜在的目标对象。

2、Intent 对象

一个 Intent 对象是一个捆信息，包含对 intent 有兴趣的组件的信息（如要执行的动作和要作用的数据）、Android 系统有兴趣的信息（如处理 intent 组件的分类信息和如何启动目标活动的指令）。下面列出它的主要信息：

2.1、组件名字

处理 intent 的组件的名字。这个字段是一个 ComponentName 对象——是目标组件的完全限定类名（如 "com.example.project.app.FreneticActivity"）和应用程序所在的包在清单文件中的名字（如 "com.example.project"）的组合。其中组件名字中的包部分不必一定和清单文件中的包名一样。

组件名字是可选的，如果设置了，intent 对象传递到指定类的实例；如果没有设置，Android 使用 intent 中的其它信息来定位合适的目标组件（见下面的 Intent 解析）。组件的名字通过 setComponent()，setClass()或 setClassName()设置，通过 getComponent()读取。

2.2、动作

一个字符串命名的动作将被执行，或在广播 intent 中，已发生动作且正被报告。Intent 类定义了一些动作常量，如下：

Constant	Target component	Action
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity of a task, with no data input and no returned output.

ACTION_SYNC	activity	Synchronize data on a server with data on the mobile device.
ACTION_BATTERY_LOW	broadcast receiver	A warning that the battery is low.
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.
ACTION_SCREEN_ON	broadcast receiver	The screen has been turned on.
ACTION_TIMEZONE_CHANGED	broadcast receiver	The setting for the time zone has changed.

查看更多的动作请参考 Intent 类。其它的动作定义在 Android API 中，我们还可以定义自己的动作字符串一再我们的应用程序中激活组件。自定义动作字符串应该包含应用程序报名前缀，如"com.example.project.SHOW_COLOR"。

动作很大程度上决定了剩下的 intent 如何构建，特别是数据（data）和附加（extras）字段，就像一个方法名决定了参数和返回值。正是这个原因，应该尽可能明确指定动作，并紧密关联到其它 intent 字段。换句话说，应该定义你的组件能够处理的 Intent 对象的整个协议，而不仅仅是单独地定义一个动作。

一个 intent 对象的动作通过 setAction()方法设置，通过 getAction()方法读取。

2.3、数据

数据（data）是将作用于其上的数据的 URI 和数据的 MIME 类型。不同的动作有不同的数据规格。例如，如果动作字段是 ACTION_EDIT，数据字段将包含将显示用于编辑的文档的 URI；如果动作是 ACTION_CALL，数据字段将是一个 tel:URI 和将拨打的号码；如果动作是 ACTION_VIEW，数据字段是一个 http:URI，接收活动将被调用去下载和显示 URI 指向的数据。

当匹配一个 intent 到一个能够处理数据的组件，通常知道数据的类型（它的 MIME 类型）和它的 URI 很重要。例如，一个组件能够显示图像数据，不应该被调用去播放一个音频文件。

在许多情况下，数据类型能够从 URI 中推测，特别是 content:URIs，它表示位于设备上的数据且被内容提供者（content provider）控制。但是类型也能够显示地设置，setData()方法指定数据的 URI，setType()指定 MIME 类型，setDataAndType()指定数据的 URI 和 MIME 类型。通过 getData()读取 URI，getType()读取类型。

2.4、种类

此外，还包含关于应该处理 intent 的组件类型信息。可以在一个 Intent 对象中指定任意数量的种类描述。Intent 类定义的一些种类常量，如下这些：

Constant	Meaning
CATEGORY_BROWSABLE	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message.
CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.
CATEGORY_HOME	The activity displays the home screen, the first screen the user sees when the device is turned on or when the HOME key is pressed.
CATEGORY_LAUNCHER	The activity can be the initial activity of a task and is listed in the top-level application launcher.
CATEGORY_PREFERENCE	The target activity is a preference panel.

更多的种类常量请参考 Intent 类。

addCategory()方法添加一个种类到 Intent 对象中，removeCategory()方法删除一个之前添加的种类，getCategories()方法获取 Intent 对象中的所有种类。

2.5、附加信息

额外的键值对信息应该传递到组件处理 intent。就像动作关联的特定种类的数据 URIs，也关联到某些特定的附加信息。例如，一个 ACTION_TIMEZONE_CHANGE intent 有一个"time-zone"的附加信息，

标识新的时区，ACTION_HEADSET_PLUG 有一个"state"附加信息，标识头部现在是否塞满或未塞满；有一个"name"附加信息，标识头部的类型。如果你自定义了一个 SHOW_COLOR 动作，颜色值将可以设置在附加的键值对中。

Intent 对象有一系列的 put...()方法用于插入各种附加数据和一系列的 get...()用于读取数据。这些方法与 Bundle 对象的方法类似，实际上，附加信息可以作为一个 Bundle 使用 putExtras()和 getExtras()安装和读取。

2.6、标志

有各种各样的标志，许多指示 Android 系统如何去启动一个活动（例如，活动应该属于那个任务）和启动之后如何对待它（例如，它是否属于最近的活动列表）。所有这些标志都定义在 Intent 类中。

3、Intent 解析

Intent 可以分为两组：

- **显式 intent**：通过名字指定目标组件。因为开发者通常不知道其它应用程序的组件名字，显式 intent 通常用于应用程序内部消息，如一个活动启动从属的服务或启动一个姐妹活动。
- **隐式 intent**：并不指定目标的名字（组件名字字段是空的）。隐式 intent 经常用于激活其它应用程序中的组件。

Android 传递一个显式 intent 到一个指定目标类的实例。Intent 对象中只用组件名字内容去决定哪个组件应该获得这个 intent，而不用其他内容。

隐式 intent 需要另外一种不同的策略。由于缺省指定目标，Android 系统必须查找一个最适合的组件（一些组件）去处理 intent——一个活动或服务去执行请求动作，或一组广播接收者去响应广播声明。这是通过比较 Intent 对象的内容和 intent 过滤器（intent filters）来完成的。intent 过滤器关联到潜在的接收 intent 的组件。过滤器声明组件的能力和界定它能处理的 intents，它们打开组件接收声明的 intent 类型的隐式 intents。如果一个组件没有任何 intent 过滤器，它仅能接收显示的 intents，而声明了 intent 过滤器的组件可以接收显示和隐式的 intents。

只有当一个 Intent 对象的下面三个方面都符合一个 intent 过滤器：action、data（包括 URI 和数据类型）、category，才被考虑。附加信息和标志在解析哪个组件接收 intent 中不起作用。

3.1、Intent 过滤器

活动、服务、广播接收者为了告知系统能够处理哪些隐式 intent，它们可以有一个或多个 intent 过滤器。每个过滤器描述组件的一种能力，即乐意接收的一组 intent。实际上，它筛掉不想要的 intents，也仅仅是不想要的隐式 intents。一个显式 intent 总是能够传递到它的目标组件，不管它包含什么；不考虑过滤器。但是一个隐式 intent，仅当它能够通过组件的过滤器之一才能够传递给它。

一个组件的能够做的每一工作有独立的过滤器。例如，记事本中的 NoteEditer 活动有两个过滤器，一个是启动一个指定的记录，用户可以查看和编辑；另一个是启动一个新的、空的记录，用户能够填充并保存。

一个 intent 过滤器是一个 IntentFilter 类的实例。因为 Android 系统在启动一个组件之前必须知道它的能力，但是 intent 过滤器通常不在 java 代码中设置，而是在应用程序的清单文件（AndroidManifest.xml）中以<intent-filter>元素设置。但有一个例外，广播接收者的过滤器通过调用 Context.registerReceiver()动态地注册，它直接创建一个 IntentFilter 对象。

一个过滤器有对应于 Intent 对象的动作、数据、种类的字段。过滤器要检测隐式 intent 的所有这三个字段，其中任何一个失败，Android 系统都不会传递 intent 给组件。然而，因为一个组件可以有多个 intent 过滤器，一个 intent 通不过组件的过滤器检测，其它的过滤器可能通过检测。

3.1.1、动作检测

清单文件中的<intent-filter>元素以<action>子元素列出动作，例如：

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
```

```
</intent-filter>
```
像例子所展示，虽然一个 Intent 对象仅是单个动作，但是一个过滤器可以列出不止一个。这个列表不能够为空，一个过滤器必须至少包含一个<action>子元素，否则它将阻塞所有的 intents。
要通过检测，Intent 对象中指定的动作必须匹配过滤器的动作列表中的一个。如果对象或过滤器没有指定一个动作，结果将如下：

- 如果过滤器没有指定动作，没有一个 Intent 将匹配，所有的 intent 将检测失败，即没有 intent 能够通过过滤器。
- 如果 Intent 对象没有指定动作，将自动通过检查（只要过滤器至少有一个过滤器，否则就是上面的情况了）

3.1.2、种类检测

类似的，清单文件中的<intent-filter>元素以<category>子元素列出种类，例如：

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```
注意本文前面两个表格列举的动作和种类常量不在清单文件中使用，而是使用全字符串值。例如，例子中所示的"android.intent.category.BROWSABLE"字符串对应于本文前面提到的 BROWSABLE 常量。类似的，"android.intent.action.EDIT"字符串对应于 ACTION_EDIT 常量。
对于一个 intent 要通过种类检测，intent 对象中的每个种类必须匹配过滤器中的一个。即过滤器能够列出额外的种类，但是 intent 对象中的种类都必须能够在过滤器中找到，只有一个种类在过滤器列表中没有，就算种类检测失败！
因此，原则上如果一个 intent 对象中没有种类（即种类字段为空）应该总是通过种类测试，而不管过滤器中有什么种类。但是有个例外，Android 对待所有传递给 Context.startActivity()的隐式 intent 好像它们至少包含"android.intent.category.DEFAULT"（对应 CATEGORY_DEFAULT 常量）。因此，活动想要接收隐式 intent 必须要在 intent 过滤器中包含"android.intent.category.DEFAULT"。
注意："android.intent.action.MAIN" 和 "android.intent.category.LAUNCHER"设置，它们分别标记活动开始新的任务和带到启动列表界面。它们可以包含"android.intent.category.DEFAULT"到种类列表，也可以不包含。

3.1.3、数据检测

类似的，清单文件中的<intent-filter>元素以<data>子元素列出数据，例如：

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```
每个<data>元素指定一个 URI 和数据类型（MIME 类型）。它有四个属性 scheme、host、port、path 对应于 URI 的每个部分：
scheme://host:port/path
例如，下面的 URI：
content://com.example.project:200/folder/subfolder/etc
scheme 是 content，host 是"com.example.project"，port 是 200，path 是"folder/subfolder/etc"。host 和 port 一起构成 URI 的凭据（authority），如果 host 没有指定，port 也被忽略。这四个属性都是可选的，但它们之间并不都是完全独立的。要让 authority 有意义，scheme 必须也要指定。要让 path 有意义，scheme 和 authority 也都必须要指定。
当比较 intent 对象和过滤器的 URI 时，仅仅比较过滤器中出现的 URI 属性。例如，如果一个过滤器仅指定了 scheme，所有有此 scheme 的 URIs 都匹配过滤器；如果一个过滤器指定了 scheme 和 authority，

但没有指定 path，所有匹配 scheme 和 authority 的 URIs 都通过检测，而不管它们的 path；如果四个属性都指定了，要都匹配才能算是匹配。然而，过滤器中的 path 可以包含通配符来要求匹配 path 中的一部分。

<data>元素的 type 属性指定数据的 MIME 类型。Intent 对象和过滤器都可以用"*"通配符匹配子类型字段，例如"text/*"，"audio/*"表示任何子类型。

数据检测既要检测 URI，也要检测数据类型。规则如下：

- 一个 Intent 对象既不包含 URI，也不包含数据类型：仅当过滤器也不指定任何 URIs 和数据类型时，才不能通过检测；否则都能通过。
- 一个 Intent 对象包含 URI，但不包含数据类型：仅当过滤器也不指定数据类型，同时它们的 URI 匹配，才能通过检测。例如，mailto:和 tel:都不指定实际数据。
- 一个 Intent 对象包含数据类型，但不包含 URI：仅当过滤也只包含数据类型且与 Intent 相同，才通过检测。
- 一个 Intent 对象既包含 URI，也包含数据类型（或数据类型能够从 URI 推断出）：数据类型部分，只有与过滤器中之一匹配才算通过；URI 部分，它的 URI 要出现在过滤器中，或者它有 content:或 file: URI，又或者过滤器没有指定 URI。换句话说，如果它的过滤器仅列出了数据类型，组件假定支持 content:和 file: 。

如果一个 Intent 能够通过不止一个活动或服务的过滤器，用户可能会被问那个组件被激活。如果没有目标找到，会产生一个异常。

3.2、通用情况

上面最后一条规则表明组件能够从文件或内容提供者获取本地数据。因此，它们的过滤器仅列出数据类型且不必明确指出 content:和 file: scheme 的名字。这是一种典型的情况，一个<data>元素像下面这样：

```
<data android:mimeType="image/*" />
```

告诉 Android 这个组件能够从内容提供者获取 image 数据并显示它。因为大部分可用数据由内容提供者（content provider）分发，过滤器指定一个数据类型但没有指定 URI 或许最通用。

另一种通用配置是过滤器指定一个 scheme 和一个数据类型。例如，一个<data>元素像下面这样：

```
<data android:scheme="http" android:type="video/*" />
```

告诉 Android 这个组件能够从网络获取视频数据并显示它。考虑，当用户点击一个 web 页面上的 link，浏览器应用程序会做什么？它首先会试图去显示数据（如果 link 是一个 HTML 页面，就能显示）。如果它不能显示数据，它将把一个隐式 Intent 加到 scheme 和数据类型，去启动一个能够做此工作的活动。如果没有接收者，它将请求下载管理者去下载数据。这将在内容提供者的控制下完成，因此一个潜在的大活动池（他们的过滤器仅有数据类型）能够响应。

大部分应用程序能启动新的活动，而不引用任何特别的数据。活动有指定"android.intent.action.MAIN"的动作的过滤器，能够启动应用程序。如果它们出现在应用程序启动列表中，它们也指定"android.intent.category.LAUNCHER"种类：

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

3.3、使用 intent 匹配

Intents 对照着 Intent 过滤器匹配，不仅去发现一个目标组件去激活，而且去发现设备上的组件的其他信息。例如，Android 系统填充应用程序启动列表，最高层屏幕显示用户能够启动的应用程序：是通过查找所有的包含指定了"android.intent.action.MAIN"的动作和"android.intent.category.LAUNCHER"种类的过滤器的活动，然后在启动列表中显示这些活动的图标和标签。类似的，它通过查找有"android.intent.category.HOME"过滤器的活动发掘主菜单。

我们的应用程序也可以类似的使用这种 Intent 匹配方式。PackageManager 有一组 query...()方法返回能够接收特定 intent 的所有组件，一组 resolve...()方法决定最适合的组件响应 intent。例如，queryIntentActivities()返回一组能够给执行指定的 intent 参数的所有活动，类似的

queryIntentServices()返回一组服务。这两个方法都不激活组件，它们仅列出所有能够响应的组件。对应广播接收者也有类似的方法——queryBroadcastReceivers()。
注：本文的主要内容意译自 SDK 文档。抱歉此系列的发文速度越来越来，主要是因为现在空闲时间很少，每天下班之后除了要写此系列的文章，还有其它知识需要总结并且还有为每天的工作做知识准备（大家应该也发现了我的 blog 最近发布了一些关于 C++的文章）。其实我一直告诫自己，信息时代效率是关键（当然专注作为一种品质，任何时候都非常重要），如果效率一直这么低而且不够专注，将来 Android 2.2、2.3 甚至更高版本都发布了，我这系列的基础部分都还没有写完。我会尽量加快此系列的发文速度，当然我会保持一贯的作风——文章的内容要经过验证才会发布，尽可能保证内容的正确性，不过由于一个人的见识和天生的会陷入自我思维，错误在所难免！还希望大家及时指出，以免文章误导他人和让我一直错下去。

Android 开发之旅: Intents 和 Intent Filters（实例部分）

2010-07-31 15:38 by 吴秦, 1481 visits, 网摘, 收藏, 编辑

引言

上篇我们介绍了 Intents 和 Intent Filters 的理论部分，主要是介绍了：activities、services、broadcast receivers 三种组件的 Intent 机制两种 Intent（显式和隐式）及它们如何去匹配目的组件、Intent 对象包含哪些信息、Intent Filters 的 action & category & data。
Intent 的重要性，我不再着重介绍了，但我还是要说：Intent 能够使应用程序突破沙盒与外界交流，者这使得 Android 的世界变得丰富多彩！本篇将用实例来介绍，如何应用 Intent，而且继续用 SMS 方面的例子来阐述。本文的主要内容如下：

- 例子（需求）描述
- STEP1、添加用于显示通讯录的布局文件
- STEP2、添加 Button 的点击事件
- STEP3、添加通讯录活动
- STEP4、解析通讯录返回的数据
- STEP5、在清单文件 AndroidManifest.xml 中注册通讯录活动和读取 Contact 数据库的权限
- 总结

例子（需求）描述

用手机发过 SMS 的人都知道：

- 用户可以先编辑短信，然后再去通讯录中选择相应的人并发生给他。
- 用户可以在短信内容中插入通讯录中联系人的号码。

我们的这个例子就是要说明如何实现这个功能。要实现这个功能，即是创建一个新的 Activity 选择（ACTION_PICK）通讯录中的数据，它会显示通讯录中的所有联系人并让用户选择，然后关闭并返回一个联系人的 URI 给短信程序。下面介绍如何一步一步实现类似的功能，而且是在之前 **Android 开发之旅：短信的收发及在 android 模拟器之间实践（一）**中发送 SMS 的例子（TextMessage）基础上加上从通讯录中选择联系人的功能。

STEP1、添加用于显示通讯录的布局文件

我们用一个 ListView 来显示整个通讯录，其中用 TextView 显示每一记录。它们的 xml 文件分别为 contact.xml、listitemlayout，如下所示：

==========contact.xml

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

```
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<ListView android:id="@+id/contactListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```
==========listitemlayout
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/itemTextView"
android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:padding="10px"
    android:textSize="16px" android:textColor="#FFF" />
</LinearLayout>
```
为了能够打开通讯录，我们还需要在 TextMessage 程序中加入一个 Button btnContact，通过点击 btnContact 激活显示通讯录的活动。这只需在 main.xml 文件中加入如下代码：
```
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="@string/btnContact"

        android:id="@+id/btnContact"
                                                    />
```
记得还有在 values/strings.xml 中相应的加入`<string name="btnContact">contact</string>`。

STEP2、添加 Button 的点击事件

在上面准备工作做完之后，我们需要监听 btnContact 的点击事件，当用户点击 btnContact 时，跳转显示通讯录界面，当用户选择一个联系人之后，返回 SMS 程序的主界面。**这里就要用到了伟大的 Intent 啦**！
```
btnContact = (Button) findViewById(R.id.btnContact);
btnContact.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        Intent intent = new Intent(Intent.ACTION_PICK,
                ContactsContract.Contacts.CONTENT_URI);
        startActivityForResult(intent, PICK_CONTACT);
    }

});
```
STEP3、添加通讯录活动

添加一个类文件，类名为 ContactPick（表示通讯录活动名）继承 Activity。它的主要功能就是获取从 SMS 主程序传递来的 Intent 并提取数据；然后去查询通讯录数据库，取出数据并填充到 STEP1 中定义的 ListView；最后，还需要添加当用户选择一个联系人的事件 onItemClick，将结果返回给 SMS 主程序，**这里也用到了我们伟大的 Intent 啦**！代码如下：

```
package skynet.com.cnblogs.www;

import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.AdapterView.OnItemClickListener;

public class ContactPick extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Intent orgIntent=getIntent();
        Uri queryUri=orgIntent.getData();
        final Cursor c = managedQuery(queryUri,
                null,
                null,
                null,
                null);

        String[] fromColumns=new
String[]{ContactsContract.Contacts.DISPLAY_NAME};
        int[] toLayoutIDs = new int[] { R.id.itemTextView };
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
                R.layout.listitemlayout, c, fromColumns, toLayoutIDs);
        ListView lv = (ListView) findViewById(R.id.contactListView);
        lv.setAdapter(adapter);
        lv.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int pos,
                    long id) {
                c.moveToPosition(pos);
                int rowId =
c.getInt(c.getColumnIndexOrThrow(ContactsContract.Contacts._ID));
```

```
                Uri outURI =
Uri.parse(ContactsContract.Contacts.CONTENT_URI.toString() + rowId);
                Intent outData = new Intent();
                outData.setData(outURI);
                setResult(Activity.RESULT_OK,outData);
                finish();
            }
        });
    }
}
```

STEP4、解析通讯录返回的数据

从通讯录活动返回之后，我们从返回的 Intent 中提取数据并填充到填写电话号码的 EditView 中。代码主要如下：

```
@Override
public void onActivityResult(int reqCode, int resCode, Intent data) {
    super.onActivityResult(reqCode, resCode, data);

    switch (reqCode) {
    case (PICK_CONTACT): {
        if (resCode == Activity.RESULT_OK) {
            String name;
            Uri contactData = data.getData();
            Cursor c = managedQuery(contactData, null, null, null, null);
            c.moveToFirst();
            name =
c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
            TextView tv;
            tv = (TextView)findViewById(R.id.edtPhoneNo);
            tv.setText(name);
        }
        break;
    }
    }
}
```

STEP5、在清单文件 AndroidManifest.xml 中注册通讯录活动和读取 Contact 数据库的权限

主要工作基本做完了，现在我们只需要注册通讯录活动和读取 Contact 数据的权限了。完整的清单文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="skynet.com.cnblogs.www" android:versionCode="1"
    android:versionName="1.0">
    <application>
        <activity android:name=".TextMessage" android:label="@string/app_name">
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ContactPick" android:label="@string/app_name">
            <action android:name="android.intent.action.PICK" />
            <category android:name="android.intent.category.DEFAULT" />
        </activity>
    </application>
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
</manifest>
```

注意通讯录活动的 Intent Filters，它的 action 是 *android.intent.action.PICK*；category 是 *android.intent.category.DEFAULT*。现在我们分析一下这个 Intent Filter：

- `<action android:name="android.intent.action.PICK" />`：使用户能够可以在通讯录列表中选择一个，然后将选择的联系人的 URL 返回给调用者。
- `<category android:name="android.intent.category.DEFAULT" />`：这是默认的 category，如果不知道 category 系统会自动加上。这个属性是让使其能够被像 Context.startActivity() 等找到。要说明的的是，如果列举了多个 category，这个活动仅会去处理那些 Intent 中都包含了所有列举的 category 的组件。

我们还可以在清单文件中看到 TextMessage 活动的 Intent Filter：

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

它指 TextMessage 活动定是真个程序的入口并且 TextMessage 会列举在 Launcher 即启动列表中。程序运行结果如下图所示：

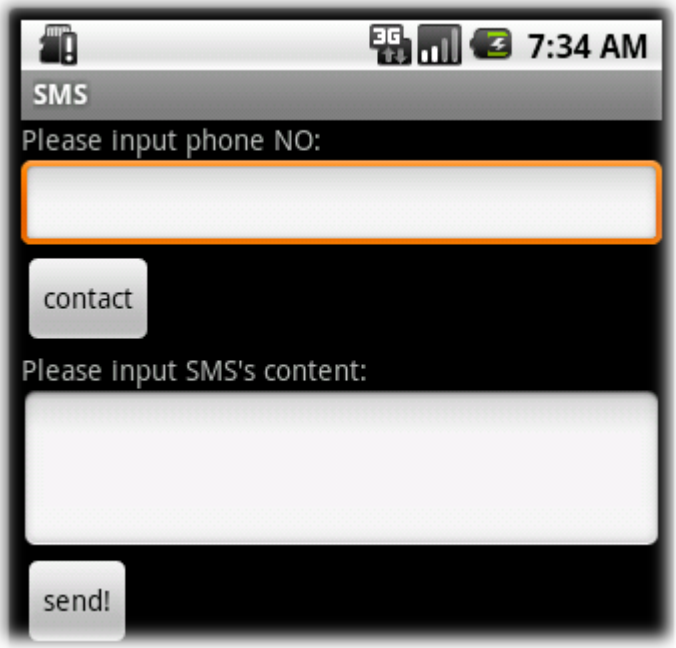


图 1、主界面

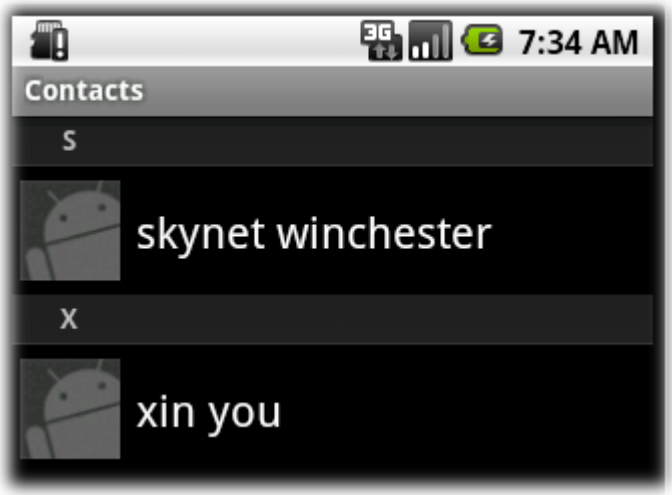


图 2、点击 contact 按钮之后

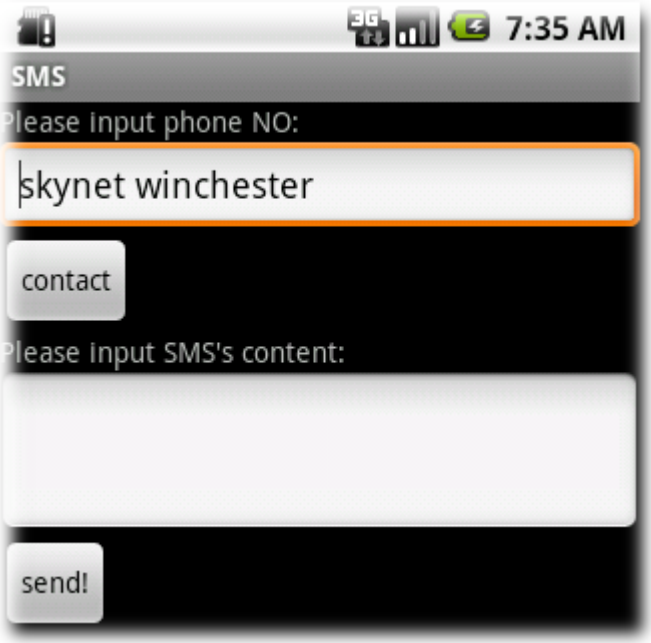


图 3、选择一个联系人之后

总结

我们用发短信中选择联系人的例子说明 Intent 和 Intent Filter，这里体现了两个活动之间如何通过 Intent 和 Intent Filter 来交互，这也是我们在编写 Android 应用程序的时候经常遇到了。本文除了上述的主要内容之外，还涉及别的知识点，下面列举几个个人认为比较有用的知识点：

- Cursor 类它跟我们平时用的数据库中的游标类似，它提供了对从数据库返回的结果的随机读写操作。如我们例子中用到的，通过 managedQuery 方法 查询数据库并返回结果，然后利用 Cursor 对它进行操作。下面介绍 Cursor 类的几个方法（我们例子中用到的，更多的方法请自行查阅相关资料）：
  - public abstract int getColumnIndexOrThrow (String columnName)：返回给定列名的索引（注意：从 0 开始的），或者当列名不存在时抛出 llegalArgumentException 异常；
  - public abstract boolean moveToFirst ()：移动到第一行。如果 Cursor 为空，则返回 FALSE
  - public abstract boolean moveToPosition (int position)：将游标移动到一个指定的位置，它的范围在-1 <= position <= count。如果 position 位置不可达，返回 FALSE

- managedQuery 方法：根据指定的 URI 路径信息返回包含特定数据的 Cursor 对象，应用这个方法可以使 Activity 接管返回数据对象的生命周期。参数：
  URI: Content Provider 需要返回的资源索引
  Projection: 用于标识有哪些 columns 需要包含在返回数据中
  Selection: 作为查询符合条件的过滤参数，类似于 SQL 语句中 Where 之后的条件判断
  SelectionArgs: 同上
  SortOrder: 用于对返回信息进行排序
- SimpleCursorAdapter 允许你绑定一个游标的列到 ListView 上，并使用自定义的 layout 显示每个项目。SimpleCursorAdapter 的创建，需要传入当前的上下文、一个 layout 资源，一个游标和两个数组：一个包含使用的列的名字，另一个（相同大小）数组包含 View 中的资源 ID，用于显示相应列的数据值。