eoe特刊

第十三期：Android Architecture Analysis

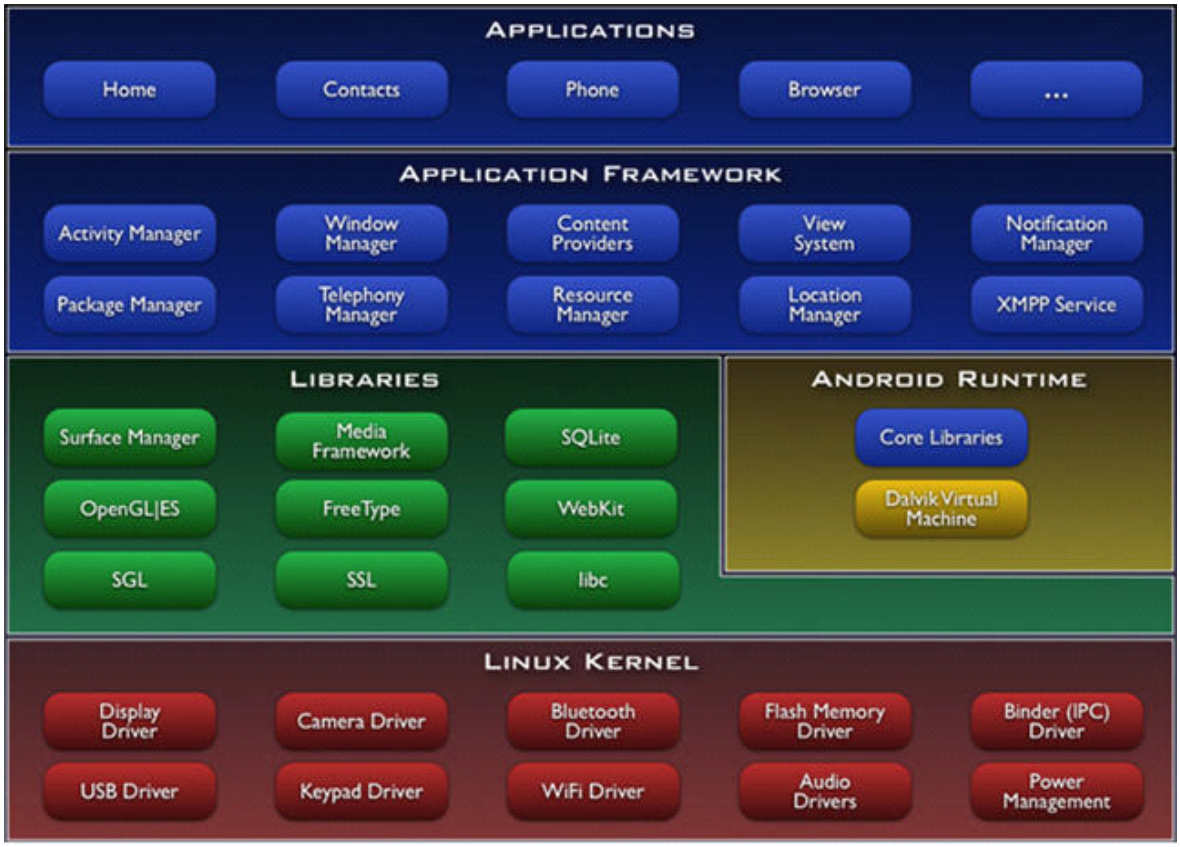
ANDROID 源码架构分析

eoeandroid.com
做最棒的Android开发社区

eoe | 优亿市场 Android应用发布与分享平台

目录

【Android 系统架构解析】



Android 系统架构图

1.1 Android 系统架构详解

Android 的系统架构和其操作系统一样，采用了分层的架构。从架构图看，android 分为四个层，从高层到低层分别是应用程序层、应用程序框架层、系统运行库层和 linux 核心层。

1.1.1 Android 应用程序

Android 会同一系列核心应用程序包一起发布，该应用程序包包括 email 客户端，SMS 短消息程序，日历，地图，浏览器，联系人管理程序等。所有的应用程序都是使用 JAVA 语言编写的。

1.1.2 Android 应用程序框架

开发人员可以完全访问核心应用程序所使用的 API 框架。该应用程序的架构设计简化了组件的重用;任何一个应用程序都可以发布它的功能块并且任何其它的应用程序都可以使用其所发布的功能块(不过得遵循框架的安全性限制)。同样，该应用程序重用机制也使用户可以方便的替换程序组件。

隐藏在每个应用后面的是一系列的服务和系统，其中包括：

* **丰富而又可扩展的视图**（Views），可以用来构建应用程序， 它包括列表(lists)，网格(grids)，文本框(text boxes)，按钮(buttons)， 甚至可嵌入的 web 浏览器。

* **内容提供器**（Content Providers），使得应用程序可以访问另一个应用程序的数据(如联系人数据库)， 或者共享它们自己的数据。

* **资源管理器**（Resource Manager），提供非代码资源的访问，如本地字符串，图形，和布局文件(layout files)。

* **通知管理器**（Notification Manager），使得应用程序可以在状态栏中显示自定义的提示信息。

* **活动管理器**(Activity Manager) 用来管理应用程序生命周期并提供常用的导航回退功能。

1.1.3 Android 系统运行库

1） Android 程序库

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

* **系统 C 库**：一个从 BSD 继承来的标准 C 系统函数库(libc)， 它是专门为基于 embedded linux 的设备定制的。

* **媒体库**： 基于 PacketVideo OpenCORE;该库支持多种常用的音频、视频格式回放和录制，同时支持静态图像文件。编码格式包括 MPEG4, H.264，MP3，AAC，AMR，JPG，PNG。

* **Surface Manager**： 对显示子系统的管理，并且为多个应用程序提 供了 2D 和 3D 图层的无缝融合。

* **LibWebCore**： 一个最新的 web 浏览器引擎用，支持 Android 浏览器和一个可嵌入的 web 视图。

* **SGL**： 底层的 2D 图形引擎。

* **3D libraries**： 基于 OpenGL ES 1.0 APIs 实现;该库可以使用硬件 3D 加速(如果可用)或者使用高度优化的 3D 软加速。

* **FreeType**： 位图(bitmap)和矢量(vector)字体显示。

* **SQLite**： 一个对于所有应用程序可用，功能强劲的轻型关系型数据库引擎。

2) Android 运行库

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。同时虚拟机是基于寄存器的，所有的类都经由 JAVA 编译器编译，然后通过 SDK 中 的 "dx" 工具转化成.dex 格式由虚拟机执行。

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 有时必须进行最佳化，会使文件大小增加 1-4 倍，以 ODEX 结尾。 Dalvik 虚拟机依赖于 linux 内核的一些功能，比如线程机制和底层内存管理机制。

1.1.4 Linux 内核

Android 的核心系统服务依赖于 Linux 2.6 内核，如安全性，内存管理，进程管理， 网络协议栈和驱动模型。 Linux 内核也同时作为硬件和软件栈之间的抽象层。

Linux 内核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。也就是说各层各尽其职，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。

1.2 Android 应用程序框架

1.2.1 Android Framework 框架介绍

我们在开发应用时都是通过框架来与 Android 底层进行交互，接触最多的就是应用框架层了。在 Android SDK 中内置了一些对象，其中最重要的组件要属 Activities、Intents、Services 以及 Content Providers 四个组件。

什么是应用程序框架呢？框架可以说是一个应用程序的核心，框架是所有参与开发的程序员共同使用和遵守的约定，大家在其约定上进行必要的扩展，但程序始终保持主体结构的一致性。其作用是让程序保持清晰、一目了然，在满足不同需求的同时又不互相影响。

1) Activities 活动

一个活动就是一个用户界面。一个应用程序可以定义一个或多个活动，每个活动都能够保存和恢复自身的状态。

2） Intents 意向

Intent 是描述一个特定活动的一种机制，比如“选取照片”、“拨打电话”等这类具体动作。在 Android 中，所有的东西都是通过 Intents 完成的，因此开发者有机会替代或重用大量的组件。比如有一个“发送邮件”的 intent，当你应用程序需要发送邮件时可以激活这个 intent。开发者甚至可以重新编写一个新的邮件应用程序，并注册为活动以处理这个 intent 代替标准的邮件应用程序。那么其他应用程序就可以使用新编写应用程序来发送邮件了。

3） Services 服务

一个服务 Service 就是运行在后台、没有用户直接交互的任务，与 Unix daemon 类似。比如要做一个音乐播放器，可能会被另一个活动激活，但音乐是需要作为背景音乐播放，那么这种程序就可以考虑作为一种服务 Service。然后别的活动可以来操作这个播放器。Android 中内置了很多服务，可以方便的使用 API 进行访问。

4） Content Providers 内容提供者

一个内容提供者 content Provider 就是由自定义的 API 封装读写操作的一套数据。Content Provider 是不同应用程序之间共享全局数据最好的方式。比如，Google 提供了联系人的 Content Provider，包括姓名、地址、电话等所有信息在内的联系方式能够被所有应用程序使用。

1.2.2 Android Framework 的功能介绍

Android 系统提供给应用开发者的本身就是一个框架，所有的应用开发都必须遵守这个框架的原则。我们在开发应用时就是在这个框架上进行扩展，下面是 Android 这个框架的功能介绍，供 eoe 的各位使用。

Android.app：	提供高层的程序模型和基本的运行环境。
android.content：	包含对各种设备上的数据进行访问和发布。
android.database：	通过内容提供者浏览和操作数据库。
android.graphics：	底层的图形库，包含画布、颜色过滤、点、矩形，可以将其直接绘制到屏幕上。
android.location：	定位和相关服务的类。
android.media：	提供一些类管理多种音频、视频的媒体接口。
android.net：	提供帮助网络访问的类，超过通常的 java.net.* 接口。
android.os：	提供了系统服务、消息传输和 IPC 机制。
android.opengl：	提供 OpenGL 的工具。
android.provider：	提供访问 Android 内容提供者的类。
android.telephony：	提供与拨打电话相关的 API 交互。
android.view：	提供基础的用户界面接口框架。
android.util：	涉及工具性的方法，例如时间日期的操作。
android.webkit：	默认浏览器操作接口。
android.widget：	包含各种 UI 元素（大部分是可见的）在应用程序的布局中使用。

1.3 Android 和 Android Linux kernel 源码获取

1.3.1 git 和 repo 简介

Git 是 Linux Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的分布式版本控制软件，它不同于 Subversion、CVS 这样的集中式版本控制系统。

在集中式版本控制系统中只有一个仓库（repository），许多个工作目录（working copy），而像 Git 这样的分布式版本控制系统中（其他主要的分布式版本控制系统还有 BitKeeper、Mercurial、GNU Arch、Bazaar、Darcs、SVK、Monotone 等），每一个工作目录都包含一个完整仓库，它们可以支持离线工作，本地提交可以稍后提交到服务器上。

分布式系统理论上也比集中式的单服务器系统更健壮，单服务器系统一旦服务器出现问题整个系统就不能运行了，分布式系统通常不会因为一两个节点而受到影响。

因为 Android 是由 kernel、Dalvik、Bionic、prebuilt、build 等多个 Git 项目组成，所以 Android 项目编写了一个名为 Repo 的 Python 的脚本来统一管理这些项目的仓库，使得 Git 的使用更加简单。

1.3.2 获取源码的工具安装

1、安装 git 和 curl：

```
apt-get install git-core curl
```

2、安装 repo：

创建存放 repo 目录

```
$ cd ~
$ mkdir bin
$ export PATH=~/bin:$PATH
```

3、下载 repo 并改变权限

```
$ curl http://android.git.kernel.org/repo >~/bin/repo
$ chmod a+x ~/bin/repo
```

1.3.3 下载源码

1、下载 Android 源码

```
$ mkdir mydroid
$ cd mydroid
$ repo init -u git://android.git.kernel.org/platform/manifest.git
$ repo sync
```

2、下载 Android Linux kernel 源码(具体见 http://android.git.kernel.org/)

git clone git://android.git.kernel.org/kernel/common.git(下载下来的内核源码在 common 文件夹中)

//注：如果过程中出现错误：fatal: Unable to look up (port 9418) (Name or service not known)，请检查下 Linux 的网络是否可用。

【Android2.1与Android2.2的源码目录】

2.1 Android2.1 源码目录

```
|-- Makefile
|-- bionic                (bionic C库)
|-- bootable              (启动引导相关代码)
|-- build                 (存放系统编译规则及generic等基础开发包配置)
|-- cts                   (Android兼容性测试套件标准)
|-- dalvik                (dalvik JAVA虚拟机)
|-- development           (应用程序开发相关)
|-- external              (android使用的一些开源的模组)
|-- frameworks            (核心框架——java及C++语言)
|-- hardware              (部分厂家开源的硬解适配层HAL代码)
|-- out                   (编译完成后的代码输出与此目录)
|-- packages              (应用程序包)
|-- prebuilt              (x86和arm架构下预编译的一些资源)
|-- sdk                   (sdk及模拟器)
|-- system                (底层文件系统库、应用及组件——C语言)
`-- vendor                (厂商定制代码)

bionic 目录
|-- libc                  (C库)
|   |-- arch-arm          (ARM架构，包含系统调用汇编实现)
|   |-- arch-x86          (x86架构，包含系统调用汇编实现)
|   |-- bionic            (由C实现的功能，架构无关)
|   |-- docs              (文档)
|   |-- include           (头文件)
|   |-- inet              (inet相关，具体作用不明)
|   |-- kernel            (Linux内核中的一些头文件)
|   |-- netbsd            (nesbsd系统相关，具体作用不明)
|   |-- private           (一些私有的头文件)
|   |-- stdio             (stdio实现)
|   |-- stdlib            (stdlib实现)
|   |-- string            (string函数实现)
```

```
|   |-- tools              （几个工具）
|   |-- tzcode             （时区相关代码）
|   |-- unistd             （unistd 实现）
|   `-- zoneinfo           （时区信息）
|-- libdl                  （libdl 实现，dl 是动态链接，提供访问动态链接
                             库的功能）
|-- libm                   （libm 数学库的实现，）
|   |-- alpha              （apaha 架构）
|   |-- amd64              （amd64架构）
|   |-- arm                （arm 架构）
|   |-- bsdsrc             （bsd 的源码）
|   |-- i386               （i386架构）
|   |-- i387               （i387架构）
|   |-- ia64               （ia64架构）
|   |-- include            （头文件）
|   |-- man                （数学函数，后缀名为.3，一些为 freeBSD 的库文
                             件）
|   |-- powerpc            （powerpc 架构）
|   |-- sparc64            （sparc64架构）
|   `-- src                （源代码）
|-- libstdc++              （libstdc++ C++实现库）
|   |-- include            （头文件）
|   `-- src                （源码）
|-- libthread_db           （多线程程序的调试器库）
|   `-- include            （头文件）
`-- linker                 （动态链接器）
`-- arch                   （支持 arm 和 x86两种架构）
```

bootable 目录

```
.
|-- bootloader             （适合各种 bootloader 的通用代码）
|   `-- legacy             （估计不能直接使用，可以参考）
|       |-- arch_armv6     （V6架构，几个简单的汇编文件）
|       |-- arch_msm7k     （高通7k 处理器架构的几个基本驱动）
|       |-- include        （通用头文件和高通7k 架构头文件）
|       |-- libboot        （启动库，都写得很简单）
```

```
|       |-- libc          （一些常用的 c 函数）
|       |-- nandwrite     （nandwirte 函数实现）
|       `-- usbloader     （usbloader 实现）
|-- diskinstaller         （android 镜像打包器，x86可生产 iso）
`-- recovery              （系统恢复相关）
    |-- edify             （升级脚本使用的 edify 脚本语言）
    |-- etc               （init.rc 恢复脚本）
    |-- minui             （一个简单的 UI）
    |-- minzip            （一个简单的压缩工具）
    |-- mtdutils          （mtd 工具）
    |-- res               （资源）
    |   `-- images        （一些图片）
    |-- tools             （工具）
    |   `-- ota           （OTA Over The Air Updates 升级工具）
`-- updater               （升级器）
```

build 目录

```
.
|-- core                  （核心编译规则）
|-- history               （历史记录）
|-- libs
|   `-- host              （主机端库，有 android “cp”功能替换）
|-- target                （目标机编译对象）
|   |-- board             （开发平台）
|   |   |-- emulator      （模拟器）
|   |   |-- generic       （通用）
|   |   |-- idea6410      （自己添加的）
|   |   `-- sim           （最简单）
|   `-- product           （开发平台对应的编译规则）
|       `-- security      （密钥相关）
`-- tools                 （编译中主机使用的工具及脚本）
    |-- acp               （Android "acp" Command）
    |-- apicheck          （api 检查工具）
    |-- applypatch        （补丁工具）
    |-- apriori           （预链接工具）
    |-- atree             （tree 工具）
```

```
    |-- bin2asm          (bin 转换为 asm 工具)
    |-- check_prereq     (检查编译时间戳工具)
    |-- dexpreopt        (模拟器相关工具，具体功能不明)
    |-- droiddoc         (java 语言，JDK5有相关文档)
    |-- fs_config        (This program takes a list of files and
                          directories)
    |-- fs_get_stats     (获取文件系统状态)
    |-- iself            (判断是否 ELF 格式)
    |-- isprelinked      (判断是否 prelinked)
    |-- kcm              (按键相关)
    |-- lsd              (List symbol dependencies)
    |-- releasetools     (生成镜像的工具及脚本)
    |-- rgb2565          (rgb 转换为565)
    |-- signapk          (apk 签名工具)
    |-- soslim           (strip 工具)
`-- zipalign           (zip archive alignment tool)

dalvik 目录             (dalvik 虚拟机)
.
|-- dalvikvm           (main.c 的目录)
|-- dexdump            (dex 反汇编)
|-- dexlist            (List all methods in all concrete classes in
                        A DEX file.)
|-- dexopt             (预验证与优化)
|-- docs               (文档)
|-- dvz                (和 zygote 相关的一个命令)
|-- dx                 (dx 工具，将多个 java 转换为 dex)
|-- hit                (java 语言写成)
|-- libcore            (核心库)
|-- libcore-disabled   (禁用的库)
|-- libdex             (dex 的库)
|-- libnativehelper    (Support functions for Android's class
                        libraries)
|-- tests              (测试代码)
|-- tools              (工具)
`-- vm                 (虚拟机实现)
```

```
development 目录          （开发者需要的一些例程及工具）
|-- apps                  （一些核心应用程序）
|   |-- BluetoothDebug    （蓝牙调试程序）
|   |-- CustomLocale      （自定义区域设置）
|   |-- Development       （开发）
|   |-- Fallback          （和语言相关的一个程序）
|   |-- FontLab           （字库）
|   |-- GestureBuilder    （手势动作）
|   |-- NinePatchLab
|   |-- OBJViewer         （OBJ 查看器）
|   |-- SdkSetup          （SDK 安装器）
|   |-- SpareParts        （高级设置）
|   |-- Term              （远程登录）
|   `-- launchperf        （装载前的预处理）
|-- build                 （编译脚本模板）
|-- cmds                  （有个 monkey 工具）
|-- data                  （配置数据）
|-- docs                  （文档）
|-- host                  （主机端 USB 驱动等）
|-- ide                   （集成开发环境）
|-- ndk                   （本地开发套件——c 语言开发套件）
|-- pdk                   （Plug Development Kit）
|-- samples                  （例程）
|   |-- AliasActivity
|   |-- ApiDemos          （API 演示程序）
|   |-- BluetoothChat     （蓝牙聊天）
|   |-- BrowserPlugin     （浏览器插件）
|   |-- BusinessCard      （商业卡）
|   |-- Compass           （指南针）
|   |-- ContactManager    （联系人管理器）
|   |-- CubeLiveWallpaper （动态壁纸的一个简单例程）
|   |-- FixedGridLayout   （像是布局）
|   |-- GlobalTime        （全球时间）
|   |-- HelloActivity     （Hello）
|   |-- Home              （Home）
|   |-- JetBoy            （jetBoy 游戏）
```

```
|   |-- LunarLander          (貌似又是一个游戏)
|   |-- MailSync             (邮件同步)
|   |-- MultiResolution      (多分辨率)
|   |-- MySampleRss          (RSS)
|   |-- NotePad              (记事本)
|   |-- RSSReader            (RSS 阅读器)
|   |-- SearchableDictionary (目录搜索)
|   |-- SimpleJNI            (JNI 例程)
|   |-- SkeletonApp          (空壳 APP)
|   |-- Snake                (snake 程序)
|   |-- SoftKeyboard         (软键盘)
|   |-- Wiktionary           (维基)
|   `-- WiktionarySimple     (维基例程)
|-- scripts                  (脚本)
|-- sdk                      (sdk 配置)
|-- simulator                (模拟器)
|-- testrunner               (测试用)
`-- tools                    (一些工具)
```

external 目录

```
.
|-- aes                      (AES 加密)
|-- apache-http              (网页服务器)
|-- astl                     (ASTL (Android STL) is a slimmed-down
                              version of the regular C++ STL.)
|-- bison                    (自动生成语法分析器，将无关文法转换成 C、
                              C++)
|-- blktrace                 (blktrace is a block layer IO tracing
                              mechanism)
|-- bluetooth                (蓝牙相关、协议栈)
|-- bsdiff                   (diff 工具)
|-- bzip2                    (压缩工具)
|-- clearsilver              (html 模板系统)
|-- dbus                     (低延时、低开销、高可用性的 IPC 机制)
|-- dhcpcd                   (DHCP 服务)
|-- dosfstools               (DOS 文件系统工具)
|-- dropbear                 (SSH2的 server)
```

```
206 |-- e2fsprogs                (EXT2文件系统工具)
207 |-- elfcopy                  (复制 ELF 的工具)
208 |-- elfutils                 (ELF 工具)
209 |-- embunit                  (Embedded Unit Project)
210 |-- emma                     (java 代码覆盖率统计工具)
211 |-- esd                      (Enlightened Sound Daemon, 将多种音频流
                                  混合在一个设备上播放)
212 |-- expat                    (Expat is a stream-oriented XML parser.)
213 |-- fdlibm                   (FDLIBM (Freely Distributable LIBM))
214 |-- freetype                 (字体)
215 |-- fsck_msdos               (dos 文件系统检查工具)
216 |-- gdata                    (google 的无线数据相关)
217 |-- genext2fs                (genext2fs generates an ext2 filesystem as
                                  A normal (non-root) user)
218 |-- giflib                   (gif 库)
219 |-- googleclient             (google 用户库)
220 |-- grub                     (This is GNU GRUB, the GRand Unified
                                  Bootloader.)
221 |-- gtest                    (Google C++ Testing Framework)
222 |-- icu4c                    (ICU(International Component for Unicode)
                                  在 C/C++下的版本)
223 |-- ipsec-tools              (This package provides a way to use the
                                  native IPsec functionality )
224 |-- iptables                 (防火墙)
225 |-- jdiff                    (generate a report describing the
                                 difference between two public Java APIs.)
226 |-- jhead                    (jpeg 头部信息工具)
227 |-- jpeg                     (jpeg 库)
228 |-- junit                    (JUnit 是一个 Java 语言的单元测试框架)
229 |-- kernel-headers           (内核的一些头文件)
230 |-- libffi                   (libffi is a foreign function interface
                                   library.)
231 |-- libpcap                  (网络数据包捕获函数)
232 |-- libpng                   (png 库)
233 |-- libxml2                  (xml 解析库)
234 |-- mtpd                     (一个命令)
235 |-- netcat                   (simple Unix utility which reads and
                                   writes dataacross network connections)
```

```
|-- netperf                    (网络性能测量工具)
|-- neven                      (看代码和 JNI 相关)
|-- opencore                   (多媒体框架)
|-- openssl                    (SSL 加密相关)
|-- openvpn                    (VPN 开源库)
|-- oprofile                   (OProfile 是 Linux 内核支持的一种性能分析
                                机制。)
|-- ping                       (ping 命令)
|-- ppp                        (pppd 拨号命令，好像还没有 chat)
|-- proguard                   (Java class file shrinker, optimizer,
                                obfuscator, and preverifier)
|-- protobuf                   (a flexible, efficient, automated
                               mechanism for serializing structured data)
|-- qemu                       (arm 模拟器)
|-- safe-iop                   (functions for performing safe integer
                                operations)
|-- skia                       (skia 图形引擎)
|-- sonivox                    (sole MIDI solution for Google Android
                                Mobile Phone Platform)
|-- speex                      (Speex 编/解码 API 的使用(libspeex))
|-- sqlite                     (数据库)
|-- srec                       (Nuance 公司提供的开源连续非特定人语音识
                                别)
|-- strace                     (trace 工具)
|-- svox                       (Embedded Text-to-Speech)
|-- tagsoup                    (TagSoup 是一个 Java 开发符合 SAX 的 HTML
                                解析器)
|-- tcpdump                    (抓 TCP 包的软件)
|-- tesseract                  (Tesseract Open Source OCR Engine.)
|-- tinyxml                    (TinyXml is a simple, small, C++ XML
                                parser)
|-- tremor                     (I stream and file decoder provides an
                                embeddable,integer-only library)
|-- webkit                     (浏览器核心)
|-- wpa_supplicant             (无线网卡管理)
|-- xmlwriter                  (XML 编辑工具)
|-- yaffs2                     (yaffs 文件系统)
`-- zlib                       (a general purpose data compression
                                library)
```

```
265
266 frameworks 目录                    （核心框架——java 及 C++语言）
267 .
268 |-- base                           （基本内容）
269 |   |-- api                        （都是 xml 文件，定义了 java 的相关 api）
270 |   |-- awt                        （AWT 库）
271 |   |-- build                      （空的）
272 |   |-- camera                     （摄像头服务程序库）
273 |   |-- cmds                       （重要命令：am、app_proce 等）
274 |   |-- core                       （核心库）
275 |   |-- data                       （字体和声音等数据文件）
276 |   |-- docs                       （文档）
277 |   |-- graphics                   （图形相关）
278 |   |-- include                    （头文件）
279 |   |-- keystore                   （和数据签名证书相关）
280 |   |-- libs                       （库）
281 |   |-- location                   （地区库）
282 |   |-- media                      （媒体相关库）
283 |   |-- obex                       （蓝牙传输库）
284 |   |-- opengl                     （2D-3D 加速库）
285 |   |-- packages                   （设置、TTS、VPN 程序）
286 |   |-- sax                        （XML 解析器）
287 |   |-- services                   （各种服务程序）
288 |   |-- telephony                  （电话通讯管理）
289 |   |-- test-runner                （测试工具相关）
290 |   |-- tests                      （各种测试）
291 |   |-- tools                      （一些叫不上名的工具）
292 |   |-- vpn                        （VPN）
293 |   `-- wifi                       （无线网络）
294 |-- opt                            （可选部分）
295 |   |-- com.google.android                （有个 framework.jar）
296 |   |-- com.google.android.googlelogin    （有个 client.jar）
297 |   `-- emoji                             （standard message elements）
298 `-- policies                  （Product policies are operating system
                                   directions aimed at specific uses）
299     `-- base
```

```
300         |-- mid                 (MID 设备)
301         `-- phone               (手机类设备一般用这个，与锁屏有关的代码)
```

2.2 Android2.2 源码目录

```
1   |-- bionic                      (bioni c library)
2   |   |-- libc                    (C 库)
3   |   |   |-- arch-arm            (ARM 架构，包含系统调用汇编实现)
4   |   |   |-- arch-x86            (x86架构，包含系统调用汇编实现)
5   |   |   |-- bionic              (由 C 实现的功能，架构无关)
6   |   |   |-- docs                (文档)
7   |   |   |-- include             (头文件)
8   |   |   |-- inet                (inet 相关)
9   |   |   |-- kernel              (Linux 内核中的一些头文件)
10  |   |   |-- netbsd              (netbsd 系统相关)
11  |   |   |-- private             (一些私有的头文件)
12  |   |   |-- stdio               (stdio 实现)
13  |   |   |-- stdlib              (stdlib 实现)
14  |   |   |-- string              (string 函数实现)
15  |   |   |-- tools               (几个工具)
16  |   |   |-- tzcode              (时区相关代码)
17  |   |   |-- unistd              (unistd 实现)
18  |   |   `-- zoneinfo            (时区信息)
19  |   |-- libdl                   (动态链接接口库(dynamic linking interface
                                    Library)，提供了直接访问动态链接库的能力)
21  |   |-- libmC                   (数学函数库，提供了 System V,ANSI C,POSIX
                                    中定义的常见的基本数学函数和浮点运算，以
                                    及浮点运算的异常处理)
23  |   |   |-- alpha               (apaha 架构)
24  |   |   |-- amd64               (amd64架构)
25  |   |   |-- arm                 (arm 架构)
26  |   |   |-- bsdsrc              (bsd 的源码)
27  |   |   |-- i386                (i386架构)
28  |   |   |-- i387                (i387架构)
```

```
|   |   |-- ia64            (ia64架构)
|   |   |-- include         (头文件)
|   |   |-- man             (数学函数，后缀名为.3，一些为 freeBSD 的库文件)
|   |   |-- powerpc         (powerpc 架构)
|   |   |-- sparc64         (sparc64架构)
|   |   `-- src             (源代码)
|   |-- libstdc++           (GNU C++ 标准库)
|   |   |-- include         (头文件)
|   |   `-- src             (源码)
|   |-- libthread_db        (线程调试库(threads debugging library),可利用此库进行多线程程序的调试工作)
|   |   `-- include         (头文件)
|   `-- linker              (用来加载动态链接库的工具(替代了常用的ld.so))
|       `-- arch            (支持 arm 和 x86两种架构)
|-- bootable                (启动引导相关代码)
|   |-- bootloader
|   |   `-- legacy
|   |-- diskinstaller
|   |   |-- editdisklbl
|   |   `-- libdiskconfig
|   `-- recovery
|   |-- edify
|   |-- etc
|   |-- minui
|   |-- minzip
|   |-- mtdutils
|   |-- res
|   |-- tools
|   `-- updater
|-- build                   (存放系统编译规则以及 generic 等基础开包配置)
|   |-- core                (各种以 mk 为结尾的文件，它门是编译所需要的 Makefile，它被顶层目录的 Makefile 引用。 envsetup.sh 是一个在使用仿真器运行的时候，用于设置环境的脚本)
```

```
59 |   |   | Makefile          (是整个 Android 编译所需要的真正的
                                Makefile，被顶层目录的 Makefile 引用。)
60 |   |   | envsetup.sh       (是一个在使用仿真器运行的时候，用于设置
                                环境的脚本。)
61 |   |   |-- combo
62 |   |   `-- tasks
63 |   |-- history
64 |   |-- libs
65 |   |   `-- host
66 |   |-- target              (包含 board 和 product 两个目录，为目标所
                                需要文件)
67 |   `-- tools               (编译过程中主机所需要的工具，一些需要经
                                过编译生成)
68 |-- cts                     (android 兼容性测试套件标准)
69 |   |-- tests
70 |   |   |-- ApiDemosReferenceTest
71 |   |   |-- ProcessTest
72 |   |   |-- SignatureTest
73 |   |   |-- appsecurity-tests
74 |   |   |-- assets
75 |   |   |-- config_demo
76 |   |   |-- core
77 |   |   |-- res
78 |   |   |-- src
79 |   |   |-- tests
80 |   |   `-- vm-tests
81 |   `-- tools
82 |       |-- annotation-helper
83 |       |-- cts-reference-app-lib
84 |       |-- dasm
85 |       |-- device-setup
86 |       |-- dex-tools
87 |       |-- dx-tests
88 |       |-- host
89 |       |-- signature-tools
90 |       |-- spec-progress
91 |       |-- test-progress
92 |       |-- test-progress-new
```

```
93  |       |-- utils
94  |       `-- vm-tests
95  |-- dalvik                          (目录用于提供Android JAVA应用程序运行的
                                        基础————JAVA虚拟机。)
96  |   |-- dalvikvm
97  |   |-- dexdump
98  |   |-- dexlist
99  |   |-- dexopt
100 |   |-- docs
101 |   |   `-- opcodes
102 |   |-- dvz
103 |   |-- dx
104 |   |   |-- etc
105 |   |   |-- src
106 |   |   `-- tests
107 |   |-- hit
108 |   |   |-- samples
109 |   |   |-- src
110 |   |   `-- test
111 |   |-- libcore    核心库相关
112 |   |   |-- annotation
113 |   |   |-- archive
114 |   |   |-- auth
115 |   |   |-- awt-kernel
116 |   |   |-- concurrent
117 |   |   |-- crypto
118 |   |   |-- dalvik
119 |   |   |-- dom
120 |   |   |-- icu
121 |   |   |-- json
122 |   |   |-- junit
123 |   |   |-- logging
124 |   |   |-- luni
125 |   |   |-- luni-kernel
126 |   |   |-- math
127 |   |   |-- nio
128 |   |   |-- nio_char
```

```
129 |    |    |-- openssl
130 |    |    |-- prefs
131 |    |    |-- regex
132 |    |    |-- security
133 |    |    |-- security-kernel
134 |    |    |-- sql
135 |    |    |-- suncompat
136 |    |    |-- support
137 |    |    |-- text
138 |    |    |-- tools
139 |    |    |-- x-net
140 |    |    `-- xml
141 |    |-- libcore-disabled
142 |    |    |-- SoundTest
143 |    |    |-- instrument
144 |    |    `-- sound
145 |    |-- libdex
146 |    |-- libnativehelper
147 |    |    `-- include
148 |    |-- tests  测试代码
149 |    |    |-- 001-nop
150 |    |    |-- 002-sleep
151 |    |    |-- 003-omnibus-opcodes
152 |    |    |-- 004-annotations
153 |    |    |-- 005-args
154 |    |    |-- 006-count10
155 |    |    |-- 007-exceptions
156 |    |    |-- 008-instanceof
157 |    |    |-- 009-instanceof2
158 |    |    |-- 010-instance
159 |    |    |-- 011-array-copy
160 |    |    |-- 012-math
161 |    |    |-- 013-math2
162 |    |    |-- 014-math3
163 |    |    |-- 015-switch
164 |    |    |-- 016-intern
```

```
165 |   |   |-- 017-float
166 |   |   |-- 018-stack-overflow
167 |   |   |-- 019-wrong-array-type
168 |   |   |-- 020-string
169 |   |   |-- 021-string2
170 |   |   |-- 022-interface
171 |   |   |-- 023-many-interfaces
172 |   |   |-- 024-illegal-access
173 |   |   |-- 025-access-controller
174 |   |   |-- 026-access
175 |   |   |-- 027-arithmetic
176 |   |   |-- 028-array-write
177 |   |   |-- 029-assert
178 |   |   |-- 030-bad-finalizer
179 |   |   |-- 031-class-attributes
180 |   |   |-- 032-concrete-sub
181 |   |   |-- 033-class-init-deadlock
182 |   |   |-- 034-call-null
183 |   |   |-- 035-enum
184 |   |   |-- 036-finalizer
185 |   |   |-- 037-inherit
186 |   |   |-- 038-inner-null
187 |   |   |-- 039-join-main
188 |   |   |-- 040-miranda
189 |   |   |-- 041-narrowing
190 |   |   |-- 042-new-instance
191 |   |   |-- 043-privates
192 |   |   |-- 044-proxy
193 |   |   |-- 045-reflect-array
194 |   |   |-- 046-reflect
195 |   |   |-- 047-returns
196 |   |   |-- 048-server-socket
197 |   |   |-- 049-show-object
198 |   |   |-- 050-sync-test
199 |   |   |-- 051-thread
200 |   |   |-- 052-verifier-fun
```

```
201 |   |   |-- 053-wait-some
202 |   |   |-- 054-uncaught
203 |   |   |-- 055-enum-performance
204 |   |   |-- 056-const-string-jumbo
205 |   |   |-- 057-iteration-performance
206 |   |   |-- 058-enum-order
207 |   |   |-- 059-finalizer-throw
208 |   |   |-- 060-reflection-security
209 |   |   |-- 061-out-of-memory
210 |   |   |-- 062-character-encodings
211 |   |   |-- 063-process-manager
212 |   |   |-- 064-field-access
213 |   |   |-- 065-mismatched-implements
214 |   |   |-- 066-mismatched-super
215 |   |   |-- 067-preemptive-unpark
216 |   |   |-- 068-classloader
217 |   |   |-- 069-field-type
218 |   |   |-- 070-nio-buffer
219 |   |   |-- 071-dexfile
220 |   |   |-- 072-precise-gc
221 |   |   |-- 073-mismatched-field
222 |   |   |-- 074-gc-thrash
223 |   |   |-- 075-verification-error
224 |   |   |-- 076-boolean-put
225 |   |   |-- 077-method-override
226 |   |   |-- 078-polymorphic-virtual
227 |   |   `-- etc
228 |   |-- tools
229 |   |   |-- dexdeps
230 |   |   |-- dmtracedump
231 |   |   `-- hprof-conv
232 |   `-- vm
233 |       |-- alloc
234 |       |-- analysis
235 |       |-- arch
236 |       |-- compiler
```

```
237 |       |-- hprof
238 |       |-- interp
239 |       |-- jdwp
240 |       |-- mterp
241 |       |-- native
242 |       |-- oo
243 |       |-- reflect
244 |       `-- test
245 |-- development                 (应用程序开发相关)
246 |   |-- apps                    (Android 应用程序的模板)
247 |   |-- build                   (编译脚本模板)
248 |   |-- cmds
249 |   |-- data
250 |   |-- docs
251 |   |-- host                    (包含 windows 平台的一些工具)
252 |   |-- ide
253 |   |-- ndk
254 |   |-- pdk
255 |   |-- samples                 (samples 中包含了很多 Android 简单工程，这
                                    些工程为开发者学习开发 Android 程序提供了很
                                    大便利，可以作为模板使用)
256 |   |-- scripts
257 |   |-- sdk
258 |   |-- sdk_overlay
259 |   |-- simulator               (大多是目标机器的一些工具)
260 |   |-- testrunner
261 |   `-- tools
262 |-- external                    (android 使用的一些开源的模组)
                注：在 external 中，每个目录表示 Android 目标系统中的一
                    个模块，可能有一个或者若干个库构成
265 |   |-- aes
266 |   |-- alsa-lib
267 |   |-- alsa-utils
268 |   |-- apache-http
269 |   |-- astl
270 |   |-- bison
271 |   |-- blktrace
```

```
|   |-- Bluetooth
|   |-- bsdiff
|   |-- bzip2
|   |-- clearsilver
|   |-- dbus
|   |-- dhcpcd
|   |-- dosfstools
|   |-- dropbear
|   |-- e2fsprogs
|   |-- elfcopy
|   |-- elfutils
|   |-- embunit
|   |-- emma
|   |-- esd
|   |-- expat
|   |-- fdlibm
|   |-- freetype
|   |-- fsck_msdos
|   |-- gdata
|   |-- genext2fs
|   |-- giflib
|   |-- googleclient
|   |-- grub
|   |-- gtest
|   |-- icu4c
|   |-- ipsec-tools
|   |-- iptables
|   |-- jdiff
|   |-- jhead
|   |-- jpeg
|   |-- junit
|   |-- libaudio
|   |-- libffi
|   |-- libpcap
|   |-- libpng
|   |-- libxml2
```

```
308 |    |-- mtpd
309 |    |-- netcat
310 |    |-- netperf
311 |    |-- neven
312 |    |-- opencore            (为PV（PacketVideo），它是Android多媒体
                                  框架的核心。）
313 |    |-- openssl             (是Secure Socket Layer，一个网络协议层，
                                  用于为数据通讯提供安全支持。）
315 |    |-- oprofile
316 |    |-- ping
317 |    |-- ppp
318 |    |-- proguard
319 |    |-- protobuf
320 |    |-- qemu
321 |    |-- safe-iop
322 |    |-- skia
323 |    |-- sonivox
324 |    |-- speex
325 |    |-- sqlite              (sqlite是Android数据库系统的核心)
326 |    |-- srec
327 |    |-- strace
328 |    |-- svox
329 |    |-- tagsoup
330 |    |-- tcpdump
331 |    |-- tesseract
332 |    |-- tremor
333 |    |-- webkitwebkit        (是Android网络浏览器的核心。）
334 |    |-- wpa_supplicant
335 |    |-- xmlwriter
336 |    |-- yaffs2
337 |    `-- zlib
338 |-- frameworks               (核心框架————java及c++语言，是
                                  Android应用程序的框架。）
339 |    |-- base
340 |    |-- opt
341 |    `-- policies
342 |-- hardware                (主要是硬件 适配层HAL代码)
```

```
|   |-- broadcom
|   |   `-- wlan                (无线网卡)
|   |-- libhardware             (硬件库)
|   |   |-- include
|   |   `-- modules             (Default (and possibly Architecture
|                                 dependents) HAL modules)
|   |-- libhardware_legacy      (旧的硬件库)
|   |   |-- flashlight          (backlight 背光)
|   |   |-- gps                 (GPS)
|   |   |-- include             (头文件)
|   |   |-- mount               (旧的挂载器)
|   |   |-- power               (电源)
|   |   |-- qemu                (模拟器)
|   |   |-- qemu_tracing        (模拟器跟踪)
|   |   |-- tests               (测试)
|   |   |-- uevent              (uevent)
|   |   |-- vibrator            (震动)
|   |   `-- wifi                (无线)
|   |-- msm7k                   (高通7k 处理器开源抽象层)
|   |   |-- boot                (启动)
|   |   |-- libaudio            (声音库)
|   |   |-- libaudio-qsd8k      (qsd8k 的声音相关库)
|   |   |-- libcamera           (摄像头库)
|   |   |-- libcopybit          (copybit 库)
|   |   |-- libgralloc          (gralloc 库)
|   |   |-- libgralloc-qsd8k    (qsd8k 的 gralloc 库)
|   |   |-- liblights           (背光库)
|   |   `-- librpc              (RPC 库)
|   |-- ril                     (无线电抽象层)
|   |   |-- include             (头文件)
|   |   |-- libril              (库)
|   |   |-- reference-cdma-sms  (cdma 短信参考)
|   |   |-- reference-ril       (ril 参考)
|   |   `-- rild                (ril 后台服务程序)
|   `-- ti                      (ti 公司开源 HAL)
|       `-- omap3               (omap3处理器)
|-- out                         (编译完成后的代码输出在此目录)
```

```
379 |    |-- host
380 |    |    |-- common
381 |    |    `-- linux-x86
382 |    |-- target
383 |    |    |-- common
384 |    |    `-- product
385 |    `-- tmp
386 |         `-- org
387 |-- packages                    (应用程序包)
388 |    |-- apps                   (apps 中是 Android 中的各种 应用程序。)
389 |    |-- inputmethods
390 |    |-- providers              (providers 是一些内容提供者（在
                                     Android 中的一个数据源）)
391 |    `-- wallpapers
392 |-- prebuilt                    (x86 和 ARM 架构下预编译的一些资源)
393 |    |-- android-arm            (arm-android 相关)
394 |    |    |-- gdbserver         (gdb 调试器)
395 |    |    `-- kernel            (模拟的 arm 内核)
396 |    |-- android-x86            (x86-android 相关)
397 |    |-- common                 (通用编译好的代码，应该是 java 的)
398 |    |-- darwin-x86             (drawin x86平台)
399 |    |-- darwin-x86_64
400 |    |-- linux-x86
401 |    |-- linux-x86_64
402 |    |-- windows
403 |    `-- windows-x86_64
404 |-- sdk                          (sdk 及模拟器)
405 |    |-- androidprefs
406 |    |-- anttasks
407 |    |-- apkbuilder
408 |    |-- archquery
409 |    |-- ddms
410 |    |-- draw9patch
411 |    |-- dumpeventlog
412 |    |-- eclipse
413 |    |-- emulator
414 |    |-- eventanalyzer
```

```
|   |-- files
|   |-- hierarchyviewer
|   |-- jarutils
|   |-- layoutlib_api
|   |-- layoutlib_utils
|   |-- layoutopt
|   |-- ninepatch
|   |-- screenshot
|   |-- sdklauncher
|   |-- sdkmanager
|   |-- sdkstats
|   |-- templates
|   `-- traceview
|-- system                      (文件系统，应用及组件 ——c 语言)
|   |-- bluetooth               (蓝牙相关)
|   |   |-- bluedroid
|   |   |-- bluez-clean-headers
|   |   |-- brcm_patchram_plus
|   |   |-- brfpatch
|   |   |-- data
|   |   `-- tools
|   |-- core                    (系统核心工具盒接口)
|   |   |-- adb                 (adb 调试工具)
|   |   |-- cpio                (cpio 工具，创建 img)
|   |   |-- debuggerd           (调试工具)
|   |   |-- fastboot            (快速启动相关)
|   |   |-- include             (系统接口头文件)
|   |   |-- init                (init 程序源代码)
|   |   |-- libacc              (轻量级 C 编译器)
|   |   |-- libctest            (libc 测试相关)
|   |   |-- libcutils           (libc 工具)
|   |   |-- liblog              (log 库)
|   |   |-- libmincrypt         (加密库)
|   |   |-- libnetutils         (网络工具库)
|   |   |-- libpixelflinger     (图形处理库)
|   |   |-- libsysutils         (系统工具库)
```

```
451 |   |   |-- libzipfile          (zip 库)
452 |   |   |-- logcat              (查看 log 工具)
453 |   |   |-- logwrapper          (log 封装工具)
454 |   |   |-- mkbootimg           (制作启动 boot.img 的工具盒脚本)
455 |   |   |-- netcfg              (网络配置 netcfg 源码)
456 |   |   |-- nexus               (google 最新手机的代码)
457 |   |   |-- rootdir             (rootfs，包含一些 etc 下的脚本和配置)
458 |   |   |-- sh                  (shell 代码)
459 |   |   |-- toolbox             (toolbox，类似 busybox 的工具集)
460 |   |   `-- vold                (SD 卡管理器)
461 |   |-- extras                  (额外工具)
462 |   |   |-- latencytop          (a tool for software developers,
                                     identifying system latency happen)
463 |   |   |-- libpagemap          (pagemap 库)
464 |   |   |-- librank             (Java Library Ranking System 库)
465 |   |   |-- procmem             (pagemap 相关)
466 |   |   |-- procrank            (Java Library Ranking System 相关)
467 |   |   |-- showmap             (showmap 工具)
468 |   |   |-- showslab            (showslab 工具)
469 |   |   |-- sound               (声音相关)
470 |   |   |-- su                  (su 命令源码)
471 |   |   |-- tests               (一些测试工具)
472 |   |   `-- timeinfo            (时区相关)
473 |   `-- wlan                    (无线相关)
474 |       `-- ti                  (ti 网卡相关工具及库)
475 `-- vendor                      (厂商定制代码)
476     |-- sample
477     |   |-- apps
478     |   |-- frameworks
479     |   |-- products
480     |   |-- sdk_addon
481     |   `-- skins
482     `-- sec
483         |-- products
484         |-- sec_proprietary
485         |-- smdk6440
486         |-- smdkc100
```

```
        |-- smdkc110
        `-- smdkv210
```

2.3 Android2.1 与 Android2.2 的对比差异（eoe 特约撰稿人：高铜）

从中选取了以下三个具有代表性的对比，2.3.1 和 2.3.2 是 UI 方面，2.3.3 是 Android2.2 新添加的功能。

2.3.1 软件安装位置属性添加 auto

自动选择：

```
internalOnly  手机内存中
preferExternal  sd 卡中
.<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="string"
          android:sharedUserId="string"
          android:sharedUserLabel="string resource"
          android:versionCode="integer"
          android:versionName="string"
          android:installLocation=["auto" | "internalOnly" |
                  "preferExternal"] >
    . . .
</manifest>
```

2.3.2 match_parent 属性值取代了 fill_parent

```
<TextView android:text="@string/extras_text"
            android:textSize="20dp"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="10dp"
            android:layout_width="match_parent|fill_parent"
            android:layout_height="wrap_content"/>
```

2.3.3 添加了能够将程序数据备份到云端的功能，下面是演示代码：

```
/**
 * This is the backup/restore agent class for the BackupRestore sample
```

```
3   * application.  This particular agent illustrates using the backup and
4   * restore APIs directly, without taking advantage of any helper
        classes.
5   */
6   public class ExampleAgent extends BackupAgent {
7       /**
8        * We put a simple version number into the state files so that we
             can
9        * tell properly how to read "old" versions if at some point we
             want
10       * to change what data we back up and how we store the state blob.
11       */
12      static final int AGENT_VERSION = 1;
13
14      /**
15       * Pick an arbitrary string to use as the "key" under which the
16       * data is backed up.  This key identifies different data records
17       * within this one application's data set.  Since we only maintain
18       * one piece of data we don't need to distinguish, so we just pick
19       * some arbitrary tag to use.
20       */
21      static final String APP_DATA_KEY = "alldata";
22
23      /** The app's current data, read from the live disk file */
24      boolean mAddMayo;
25      boolean mAddTomato;
26      int mFilling;
27
28      /** The location of the application's persistent data file */
29      File mDataFile;
30
31      /** For convenience, we set up the File object for the app's data
            on creation */
32      @Override
33      public void onCreate() {
34          mDataFile = new File(getFilesDir(),
                        BackupRestoreActivity.DATA_FILE_NAME);
35      }
36
37      /**
38       * The set of data backed up by this application is very small:
             just
```

```
39      * two booleans and an integer.  With such a simple dataset, it's
40      * easiest to simply store a copy of the backed-up data as the
            state
41      * blob describing the last dataset backed up.  The state file
42      * contents can be anything; it is private to the agent class, and
43      * is never stored off-device.
44      *
45      * <p>One thing that an application may wish to do is tag the state
46      * blob contents with a version number.  This is so that if the
47      * application is upgraded, the next time it attempts to do a
            backup,
48      * it can detect that the last backup operation was performed by an
49      * older version of the agent, and might therefore require
            different
50      * handling.
51      */
52     @Override
53     public void onBackup(ParcelFileDescriptor oldState,
                    BackupDataOutput data,
54             ParcelFileDescriptor newState) throws IOException {
55         // First, get the current data from the application's file.
                This
56         // may throw an IOException, but in that case something has
                gone
57         // badly wrong with the app's data on disk, and we do not want
58         // to back up garbage data.  If we just let the exception go,
                the
59         // Backup Manager will handle it and simply skip the current
60         // backup operation.
61         synchronized (BackupRestoreActivity.sDataLock) {
62             RandomAccessFile file = new RandomAccessFile(mDataFile,
                    "r");
63             mFilling = file.readInt();
64             mAddMayo = file.readBoolean();
65             mAddTomato = file.readBoolean();
66         }
67
68         // If the new state file descriptor is null, this is the first
                time
69         // a backup is being performed, so we know we have to write the
70         // data.  If there <em>is</em> a previous state blob, we want
                to
```

```
71          // double check whether the current data is actually different
                from
72          // our last backup, so that we can avoid transmitting redundant
73          // data to the storage backend.
74          boolean doBackup = (oldState == null);
75          if (!doBackup) {
76              doBackup = compareStateFile(oldState);
77          }
78
79          // If we decided that we do in fact need to write our dataset,
                go
80          // ahead and do that.  The way this agent backs up the data is
                to
81          // flatten it into a single buffer, then write that to the
                backup
82          // transport under the single key string.
83          if (doBackup) {
84              ByteArrayOutputStream bufStream = new
                    ByteArrayOutputStream();
85
86              // We use a DataOutputStream to write structured data into
87              // the buffering stream
88              DataOutputStream outWriter = new
                    DataOutputStream(bufStream);
89              outWriter.writeInt(mFilling);
90              outWriter.writeBoolean(mAddMayo);
91              outWriter.writeBoolean(mAddTomato);
92
93              // Okay, we've flattened the data for transmission.  Pull
                    it
94              // out of the buffering stream object and send it off.
95              byte[] buffer = bufStream.toByteArray();
96              int len = buffer.length;
97              data.writeEntityHeader(APP_DATA_KEY, len);
98              data.writeEntityData(buffer, len);
99          }
100
101         // Finally, in all cases, we need to write the new state blob
102         writeStateFile(newState);
103     }
104
105     /**
```

```
106     * Helper routine - read a previous state file and decide whether to
107     * perform a backup based on its contents.
108     *
109     * @return <code>true</code> if the application's data has changed since
110     *   the last backup operation; <code>false</code> otherwise.
111     */
112    boolean compareStateFile(ParcelFileDescriptor oldState) {
113        FileInputStream instream = new FileInputStream(oldState.getFileDescriptor());
114        DataInputStream in = new DataInputStream(instream);
115
116        try {
117            int stateVersion = in.readInt();
118            if (stateVersion > AGENT_VERSION) {
119                // Whoops; the last version of the app that backed up
120                // data on this device was <em>newer</em> than the current
121                // version -- the user has downgraded.  That's problematic.
122                // In this implementation, we recover by simply rewriting
123                // the backup.
124                return true;
125            }
126
127            // The state data we store is just a mirror of the app's data;
128            // read it from the state file then return 'true' if any of
129            // it differs from the current data.
130            int lastFilling = in.readInt();
131            boolean lastMayo = in.readBoolean();
132            boolean lastTomato = in.readBoolean();
133
134            return (lastFilling != mFilling)
135                    || (lastTomato != mAddTomato)
136                    || (lastMayo != mAddMayo);
137        } catch (IOException e) {
138            // If something went wrong reading the state file, be safe
139            // and back up the data again.
140            return true;
```

```
141         }
142     }
143
144     /**
145      * Write out the new state file:  the version number, followed by
             the
146      * three bits of data as we sent them off to the backup transport.
147      */
148     void writeStateFile(ParcelFileDescriptor stateFile) throws
              IOException {
149         FileOutputStream outstream = new
                   FileOutputStream(stateFile.getFileDescriptor());
150         DataOutputStream out = new DataOutputStream(outstream);
151
152         out.writeInt(AGENT_VERSION);
153         out.writeInt(mFilling);
154         out.writeBoolean(mAddMayo);
155         out.writeBoolean(mAddTomato);
156     }
157
158     /**
159      * This application does not do any "live" restores of its own
             data,
160      * so the only time a restore will happen is when the application
             is
161      * installed.  This means that the activity itself is not going to
162      * be running while we change its data out from under it.  That, in
163      * turn, means that there is no need to send out any sort of
             notification
164      * of the new data:  we only need to read the data from the stream
165      * provided here, build the application's new data file, and then
166      * write our new backup state blob that will be consulted at the
             next
167      * backup operation.
168      *
169      * <p>We don't bother checking the versionCode of the app who
             originated
170      * the data because we have never revised the backup data format.
             If
171      * we had, the 'appVersionCode' parameter would tell us how we
             should
172      * interpret the data we're about to read.
```

```
173      */
174     @Override
175     public void onRestore(BackupDataInput data, int appVersionCode,
176             ParcelFileDescriptor newState) throws IOException {
177         // We should only see one entity in the data stream, but the
                safest
178         // way to consume it is using a while() loop
179         while (data.readNextHeader()) {
180             String key = data.getKey();
181             int dataSize = data.getDataSize();
182 
183             if (APP_DATA_KEY.equals(key)) {
184                 // It's our saved data, a flattened chunk of data all
                        in
185                 // one buffer.  Use some handy structured I/O classes
                        to
186                 // extract it.
187                 byte[] dataBuf = new byte[dataSize];
188                 data.readEntityData(dataBuf, 0, dataSize);
189                 ByteArrayInputStream baStream = new
                                ByteArrayInputStream(dataBuf);
190                 DataInputStream in = new DataInputStream(baStream);
191 
192                 mFilling = in.readInt();
193                 mAddMayo = in.readBoolean();
194                 mAddTomato = in.readBoolean();
195 
196                 // Now we are ready to construct the app's data file
                        based
197                 // on the data we are restoring from.
198                 synchronized (BackupRestoreActivity.sDataLock) {
199                     RandomAccessFile file = new
                                RandomAccessFile(mDataFile, "rw");
200                     file.setLength(0L);
201                     file.writeInt(mFilling);
202                     file.writeBoolean(mAddMayo);
203                     file.writeBoolean(mAddTomato);
204                 }
205             } else {
206                 // Curious!  This entity is data under a key we do not
207                 // understand how to process.  Just skip it.
208                 data.skipEntityData();
```

```
209                 }
210             }
211
212             // The last thing to do is write the state blob that describes
                    the
213             // app's data as restored from backup.
214             writeStateFile(newState);
215         }
216 }
```

【Android架构 实例教程】

3.1 应用框架入门实例

以HelloActivity程序为例，简单介绍Android应用程序的框架。希望eoe的各位，可以根据HelloActivity，自己写出一个Andorid的应用程序。

1、HelloActivity工程的源代码在Android目录的development/samples/HelloActivit-y/中，代码的结构如下所示：

```
development/samples/HelloActivity/
|-- Android.mk
|-- AndroidManifest.xml
|-- res
|   |-- layout
|   |   `-- hello_activity.xml
|   `-- values
|       `-- strings.xml
|-- src
    `-- com
        `-- example
            `-- android
                `-- helloactivity
                    `-- HelloActivity.java
```

其中tests是一个独立的项目，可以暂时不考虑。其他部分看作一个Android的一应用程序的工程。这个工程主要的组成部分如下所示：

Android.mk

2、Android.mk是整个工程的“Makefile”，其内容如下所示：

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := samples
# Only compile source java files in this apk.
LOCAL_SRC_FILES := $(call all-java-files-under, src)
LOCAL_PACKAGE_NAME := HelloActivity
LOCAL_SDK_VERSION := current
include $(BUILD_PACKAGE)
# Use the following include to make our test apk.
include $(call all-makefiles-under,$(LOCAL_PATH))
```

其中LOCAL_PACKAGE_NAME表示了这个包的名字。这个文件是最终生成的包（*.apk）的名称。**注意**，包的名称和应用程序目录的名称无关，而与这里的HelloActivity的名称有关。

AndroidManifest.xml

3、AndroidManifest.xml 工程的描述文件

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.android.helloactivity">
4      <application android:label="Hello, Activity!">
5          <activity android:name="HelloActivity">
6              <intent-filter>
7                  <action android:name="android.intent.action.MAIN"/>
8                  <category
                       android:name="android.intent.category.LAUNCHER"/>
9              </intent-filter>
10         </activity>
11     </application>
12 </manifest>
```

在工程描述文件中，package 的名称需要和 JAVA 文件中包的名称相同，activity 的名称必须和 JAVA 文件中 JAVA 类的名称相同，JAVA 文件的文件名也必须和其中类的名称相同。

而那个 android:label 的名字既是应用程序在菜单中的名字，也是应用程序启动后的标题。

HelloActivity.java

4、HelloActivity.java 这是 JAVA 类文件，这个文件的路径表示在 Andorid 的 JAVA 包的结构中的位置，这个包的使用方式为 com.example.android.helloactivity。

```
1  package com.example.android.helloactivity;
2  import android.app.Activity;
3  import android.os.Bundle;
4  public class HelloActivity extends Activity {
5      public HelloActivity() {
6      }
7      @Override
8      public void onCreate(Bundle savedInstanceState) {
9          super.onCreate(savedInstanceState);
10         setContentView(R.layout.hello_activity);
11     }
12 }
```

AndroidManifest.xml 和*.JAVA 的名称需要匹配（这里的 HelloActivity.java 和 HelloActivity），否则是找不到 JAVA 类的。

5、程序编译完成后，将生成 apk 包，将其放置在 system/app 中即可。除了使用界面启动之外，还可以在 QEMU 仿真器的启动界面中，使用如下的方式启动：

```
am start -n
    com.example.android.helloactivity/com.example.android.helloactivity
    .HelloActivity
```

事实上，启动的方法和工程描述文件中包的名称和类的名称有关。

几个注意点：

1、应用程序文件夹的名称并无实际的影响。
2、Android.mk 的 LOCAL_PACKAGE_NAME，决定 APK 包的名称。
3、AndroidManifest.xml 中需要包含包的名称和 activity 类的名称，并需要和 JAVA 文件中对应。
4、AndroidManifest.xml 中 application android:label 的名称既是应用程序在界面中的名称，也是启动后的标题。

3.2 用代理模式处理海量高频数据更新（开发者社区 ID：Ihavegotyou）

业务背景：海量高频数据(如股票实时报价)，更新的规则：被更新的对象和更新方法都不一样.

Java 代码 1

```
public interface CommonDefn {
    public static int  HIGHLIGHT_BACKGROUND_COLOR_INDEX = 0xff0033ff;
    public class DoThingsReturn{
        public Object cmd;
        public Object data;
        public int errCode;
        public DoThingsReturn(Object cmd, Object data, int errorCode) {
            super();
            this.cmd = cmd;
            this.data = data;
            this.errCode = errorCode;
        }
    }
}
```

Java 代码 2

```
1  /**
2   *  used  for updating  the  background of the view while the data
       changed
3   *
4   *
5   */
6  public  interface RefreshHandlerInterface {
7          public void updateBackground(Object handlerId, int color);
                 //the proxy updating method
8          public Handler getHandler();  //the proxy handler
9  }
```

Java 代码 3

```
1  /**
2   *
3   * decrease alpha channel value from 255 to 0.
4   *
5   */
6  public class ColorRefreshTask extends TimerTask {
7      private RefreshHandlerInterface refreshHandler;
8      private final static int  DELAY_ONCE =200;
9      private final static int TOTAL_RUNTIME = 1500;
10     private final static int POWER_16_16 = 16 * 16* 16 * 16 * 16 * 16;
11     private final static int INCREASE_ONCE =  0xff / (TOTAL_RUNTIME /
               DELAY_ONCE);
12     private int color;
13     private Object id;
14     private int startTime;
15     private int alphaChannel;
17     /**
18      *
19      * @param color ( current background color)
20      * @param id  id of the updated view
21      */
```

```
22      public ColorRefreshTask(RefreshHandlerInterface refreshHandler, int
              color, Object id) {
23          super();
24          Log.d("color", "ready to set color!");
25          this.color = color;
26          this.id = id;
27          this.startTime = 0;
28          this.alphaChannel = 0;
29          this.refreshHandler = refreshHandler;
30      }
32      public void run(){
34          int colorComm = color - 0xff000000;  //RGB color value;
35          int currColor = 0;
37          if(startTime < TOTAL_RUNTIME) {
38                  startTime +=  DELAY_ONCE;
39                  alphaChannel +=  INCREASE_ONCE;
40                  currColor = POWER_16_16 *  alphaChannel  + colorComm;
41                  //Log.d("color", Integer.toHexString(currColor));
42                  sendMsg(currColor);
43              refreshHandler.
                        getHandler().postDelayed(this,DELAY_ONCE);
45          }
46          else {
47              sendMsg(currColor <= color ? color : 0 );
48              cancel();
49          }
52      }
54      public void startTimer(){
55          refreshHandler.getHandler().postDelayed(this,DELAY_ONCE);
56      }
59      private void sendMsg(final int currColor){
60          final Runnable myUpdateResults = new Runnable() {
61              public void run() {
62                  refreshHandler.updateBackground( id, currColor);
63              }
64          };
```

```
66          new Thread() {
67              public void run() {
68                  refreshHandler.getHandler().post(myUpdateResults);
69              }
70          }.start();
72      }
74      public void stopTimer(){
75            this.cancel();
76      }
79  }
```

Java 代码 4

```
1   import android.app.Activity;
2   import android.os.Bundle;
3   import android.os.Handler;
4   import android.view.View;
5   import android.widget.TextView;
7   public class TestActivity extends Activity implements
    RefreshHandlerInterface{
8
9       private  Handler  _messageHandler = new Handler();
11      @Override
12      public Handler getHandler() {
13          return _messageHandler;
14      }
18      @Override
19      public void updateBackground(Object handlerId, int color) {
20          if(!( handlerId instanceof CommonDefn.DoThingsReturn)) {
21              return;    //invalid parameter
22          }
23          else {
26                  TextView  current =
                        (TextView)(((CommonDefn.DoThingsReturn)handlerId).
                        data);
27                  if(current.getVisibility() != View.VISIBLE) {
28                      return;         //no need to update  for the view
29                  }
```

```
30                      if(color<=0) {
31                          current.setBackgroundDrawable(null);
32              current.setText(((CommonDefn.DoThingsReturn)handlerId).
                    cmd.toString());
33                      }
34                      else {
35                          current.setBackgroundColor( color);
36                      }
37                      current.postInvalidate();
38          }
40      }
41      /** Called when the activity is first created. */
42      @Override
43      public void onCreate(Bundle savedInstanceState) {
44          super.onCreate(savedInstanceState);
45          setContentView(R.layout.main);
48          final TextView tv = (TextView)findViewById(R.id.txtview);
51          new Thread() {
52                java.util.Random rand =  new java.util.Random();
53                long lastUpdate = System.currentTimeMillis();
55                  public void run() {
56                    while(!Thread.interrupted() || !TestActivity.this.
                          isFinishing()) {
57                      //模拟高频更新数据
58                      if( System.currentTimeMillis() - lastUpdate <
                          20001)  { //设置两次动画的最少间隔时间
59                          continue;
60                      }
61                      else
62                          lastUpdate = System.currentTimeMillis();
63                      CommonDefn.DoThingsReturn  updateWrapper = new
                            CommonDefn.DoThingsReturn(String.valueOf(r
                            and.nextInt()) , tv,0);
64                      ColorRefreshTask refresh = new
                            ColorRefreshTask(TestActivity.this,CommonD
                            efn.HIGHLIGHT_BACKGROUND_COLOR_INDEX,
                            updateWrapper);
65                      _messageHandler.postDelayed(refresh, 200);
                            //ready to for the current updating
68                    }
```

```
69                      }
70                  }.start();
75      }
76  }
```

3.3 Android 中媒体播放器部分的架构（开发者社区 ID：tony）

Android MediaPlayer 的主要功能，具体实现在 OpenCore 的 Player 中。

在各个库中，libmedia.so 位于核心的位置，它对上层的提供的接口主要是 MediaPlayer 类，类 libmedia_jni.so 通过调用 MediaPlayer 类提供对 JAVA 的接口，并且实现了 android.media.MediaPlayer 类。

libmediaplayerservice.so 是 Media 的服务器，它通过继承 libmedia.so 的类实现服务器的功能，而 libmedia.so 中的另外一部分内容则通过进程间通讯和 libmediaplayerservice.so 进行通讯。

libmediaplayerservice.so，真正功能通过调用 OpenCore Player 来完成。MediaPlayer 部分的头文件在 frameworks/base/include/media/目录中，这个目录是和 libmedia.so 库源文件的目录 frameworks/base/media/libmedia/相对应的。

主要的头文件有以下几个：

```
1   notify_callback_f notifyFunc)
2   {
3       sp<MediaPlayerBase> p;
4       switch (playerType) {
5           case PV_PLAYER:
6               LOGV(" create PVPlayer");
7               p = new PVPlayer();
8               break;
9           case SONIVOX_PLAYER:
10              LOGV(" create MidiFile");
11              p = new MidiFile();
12              break;
13          case VORBIS_PLAYER:
14              LOGV(" create VorbisPlayer");
15              p = new VorbisPlayer();
16              break;
17      }
```

```
18  //……
19      return p;
20  }
```

在这些头文件 mediaplayer.h 提供了对上层的接口，而其他的几个头文件都是提供一些接口类（即包含了纯虚函数的类），这些接口类必须被实现类继承才能够使用。由于具有纯虚函数。

IMediaPlayerService 以及 BnMediaPlayerService 必须被继承实现才能够使用，在 IMediaPlayerService 定义的 create 和 decode 等 接口。事实上是必须被继承者实现的内容。

注意：create 的返回值的类型是 sp<IMediaPlayer>，这个 IMediaPlayer 正是提供实现功能的接口。值得注意的是 PVPlayer、MidiFile 和 VorbisPlayer 三个类都是继承 MediaPlayerInterface 得到的。

Android 架构是继承 MediaPlayerBase 得到的，因此三者具有相同接口类型。只有建立的时候会调用各自的构造函数，在建立之后，将只通过 MediaPlayerBase 接口来 MediaPlayerBase 控制它们。在 frameworks/base /media/libmediaplayerservice 目录中，MidiFile.h 和 MidiFile.cpp 的实现 MidiFile，VorbisPlayer.h 和 VorbisPlayer.cpp 实现一个 VorbisPlayer。

3.4 典型的Content Provider 代码架构（开发者社区ID：springfieldx）

我们平时在做Android开发的时候，一定经常会接触到数据库操作,android使用sqlite作为它的本地数据库，并提供了一种叫做Content Provider的数据访问机制，简单来说，它就像一个web服务，有自己的URI,我们也是通过URI的形式来访问它的数据，通过这种形式的接口，使得我们的数据不仅在我们自己的应用中可以访问，甚至还可以被系统中的其他应用所调用。 一个典型的例子就是我们手机中的通讯录，android给我们暴露了一个接口，我们只要申请到相应的权限，通过访问这个接口，就可以得到通讯录的信息了。 说了这么多，现在言归正传，这篇文章主要是和大家分享一下Content Provider的实现方式，通过一些更加标准的代码架构，可以使我们的项目的效率更高，并且提高可维护性。

例如，我们有一个记事本程序，需要记录每一条记事，那么我们就需要这样一个数据表：

```
note

_id         integer
content     varchar(2000)
pubDate     integer
```

为了说明问题，我们只位这个表设置了三个字段，分别使记录的id，记事内容，和编辑时间， 大家需要注意个是id字段需要在前面加一个下划线，否则在和ListView绑定的时候会出问题。

1、有了表接口，那么我们就可以将这张表抽象程一个数据结构，代码如下：

```
1  public class NoteProvider extends ContentProvider{
2      private static HashMap<String, String> noteProjectionMap;
3  static {
4          noteProjectionMap = new HashMap<String, String>();
```

```
        noteProjectionMap.put(NoteTableMetaData.NOTE_ID,
                NoteTableMetaData.NOTE_ID);
        noteProjectionMap.put(NoteTableMetaData.NOTE_CONTENT,
        NoteTableMetaData.NOTE_CONTENT);
        noteProjectionMap.put(NoteTableMetaData.NOTE_PUB_DATE,
                NoteTableMetaData.NOTE_PUB_DATE);
    }
}
```

这个HashMap其实就相当于我们select语句中的别名，在后面的sqlite操作中会用到这个数据，一般情况下，我们不需要更改别名，所以在map中将键和值都设置为同样的就可以了。

2、由于我们的NoteProvider要处理两种形式的URI，所以我们需要一个机制来区分不同的URI，这就要用到UriMatcher，代码如下：

```
private static UriMatcher matcher;
    private static final int QUERY_LIST = 1;
    private static final int QUERY_ITEM = 2;
    static {
        matcher = new UriMatcher(UriMatcher.NO_MATCH);
        matcher.addURI(NoteMetaData.AUTHORITY, "notes", QUERY_LIST);
        matcher.addURI(NoteMetaData.AUTHORITY, "notes/#", QUERY_ITEM);

    }
```

UriMatcher 用来区分不同的 URI，首先我们将它定义为一个静态属性，然后在静态初始化块中，使用它的 addURI 方法，为它添加了两个 URI 规则，并为每个规则设置了一个表示常量，这里有 QUERY_LIST 和 QUERY_ITEM，而这个方法的前两个参数就是用来构造这个 URI 规则的，例如第一条规则中，第一个参数我们用到了元数据中的 AUTHORITY 和一个 notes 字符串， 这样，这条规则最终就会成为这种形式 org.spring.provider.NoteProvider/notes 而另外一条规则就是这样 org.spring.provider.NoteProvider/notes/# ,注意到第二条规则中的井号。它是一个占位符，在实际的场景中，这个位置会用一个数字来代替。

说了这么多，我们为什么要用两个 URI 来为这个 Provider 来提供接口呢，相信只要做过 web 开发的朋友就会知道，假如我们要做一个 CRUD 功能，我们首先需要一个页面来显示数据，这个页面是不接受 ID 参数的。但我们还需要有一个编辑功能的页面，而这个页面就需要接受一个 ID 来区分要编辑哪一条记录。这样就不难理解我们为什么要用两种 URI 了，这里的第一条 URI 规则，就相当于那个显示数据的页面，而第二个 URI 中的井号的位置就相当于编辑页面中的 ID。有了这个 matcher 后，我们就可以根据不同的 URI 开执行各自所需的操作。

3、现在 Provider 的基本信息已经基本完成了，因为我们的 Provider 需要和数据库进行交互，所以我们还需要一个中间层，可以使用 SQLiteOpenHelper。

```
1  private DatabaseHelper dbHelper;
3     class DatabaseHelper extends SQLiteOpenHelper{
5         DatabaseHelper(Context context) {
7             super(context,    NoteMetaData.DATABASE_NAME, null,
                  NoteMetaData.DATABASE_VERSION);
8         }
10        @Override
11        public void onCreate(SQLiteDatabase db) {
13            db.execSQL(
15                    "create table " + NoteTableMetaData.TABLE_NAME + " ( "
16                    + NoteTableMetaData.NOTE_ID + " integer primary key, "
17                    + NoteTableMetaData.NOTE_CONTENT + " varchar(2000), "
18                    + NoteTableMetaData.NOTE_PUB_DATE + " integer"
19                    + ");"
21            );
22        }
24        @Override
25        public void onUpgrade(SQLiteDatabase db, int oldVersion, int
                newVersion) {
26
27            db.execSQL("drop table if exists " +
                  NoteTableMetaData.TABLE_NAME);
28            onCreate(db);
29
30        }
31    }
```

如上代码，我们扩展了自己的Helper，并将它定义为Provider 的私有属性，这个类的实现中我们还是使用元数据来进行操作，例如创建数据库和更新数据库的操作，字段名和表名使用的是元数据中的属性。

4、现在有了这些基础架构后，我们就可以实现相应的数据库操作方法了，先从query说起：

```
1     public Cursor query(Uri uri, String[] projection, String selection,
2             String[] selectionArgs, String sortOrder) {
3         SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
4         switch (matcher.match(uri)) {
5             case QUERY_LIST:{
6                 qb.setTables(NoteTableMetaData.TABLE_NAME);
```

```
7                   qb.setProjectionMap(noteProjectionMap);
8               }break;
9               case QUERY_ITEM:{
10                  qb.setTables(NoteTableMetaData.TABLE_NAME);
11                  qb.setProjectionMap(noteProjectionMap);
12                  qb.appendWhere(NoteTableMetaData.NOTE_ID + "=" +
                        uri.getPathSegments().get(1));
13              }break;
14          default:
15              throw new IllegalArgumentException("Unknown URI " + uri);
16          }
17          String orderBy;
18          if(TextUtils.isEmpty(sortOrder)) {
19              orderBy = NoteTableMetaData.DEFAULT_SORT_ORDER;
20          }else {
21              orderBy = sortOrder;
22          }
23          SQLiteDatabase db = dbHelper.getWritableDatabase();
24                  Cursor c = qb.query(db, projection, selection,
                          selectionArgs, null, null, orderBy);
25               c.setNotificationUri(getContext().getContentResolver(),
                          uri);
26          return c;
27      }
```

这里使用了 SQLiteQueryBuilder 来构建数据库查询，这样可使我们从更抽象的层次来处理数据库交互，在这里我们之前定义的 matcher 就派上用场拉，我们判断了一下 URI 的类型，如果使 QUERY_LIST，就查询所有的数据，而如果是 QUERY_ITEM 类型，就只查询特定 ID 下的记录。这一点在我们的代码中应该很明显，当然如果给出的 URI 不符合任意一条规则，那么就直接抛出异常。

接下来我判断了一下这次查询是否明确指定了排序规则，如果没指定就使用我们之前定一个默认规则来对数据进行排序，随后就是获取数据库引用，并执行插叙操作了。

这里要注意一下 c.setNotificationUri 方法，这个方法为当的 URI 注册了一下通知，简单来说就是这样，如果有其他的调用改变了底层数据库并发送了更改消息，那么这些注册了通知的 URI 会自动更新他们的数据集，而不用我们手动的进行刷新，这样可以省去很多繁琐的编码工作。最典型的例子就是为 ListView 绑定数据时，如果使用了这种机制，我们在修改数据库中的数据后，ListView 的显示也会自动的刷新。

5、接下来介绍一下insert方法：

```
1     public Uri insert(Uri uri, ContentValues values) {
2         if(matcher.match(uri) != QUERY_LIST) {
3             throw new IllegalArgumentException("Unknown URI " + uri);
4         }
5         if(values.containsKey(NoteTableMetaData.NOTE_PUB_DATE) == false)
              {
6             Long now = Long.valueOf(System.currentTimeMillis());
7             values.put(NoteTableMetaData.NOTE_PUB_DATE, now);
8         }
9         SQLiteDatabase db = dbHelper.getWritableDatabase();
10        long rowID = db.insert(NoteTableMetaData.TABLE_NAME,
                NoteTableMetaData.NOTE_CONTENT, values);
11                if(rowID > 0) {
12            Uri insertedUri =
                  ContentUris.withAppendedId(NoteTableMetaData.CONTENT_URI
                  , rowID);
13            getContext().getContentResolver().notifyChange(insertedUri,
                  null);
14            return insertedUri;
15        }
16        throw new android.database.SQLException("Failed to insert row
                  into " + uri);
17    }
```

先说一下这个方法传进来的参数，第一个参数时调用的URI，接下来的values就相当于insert语句中的列名和值的一对组合，由于这个插入方法只接受不带ID的URI所以我们在一开始的时候进行了一下判断，然后我们检测了一下是否指定了日期，如果没有就以当前时间作为默认值，随后就是数据库调用了。

我们通过返回rowID来判断该条记录是否插入成功，如果成功就返回带着这条记录ID的URI，否则就抛出异常。注意到这里的notifyChange方法，这个正好和前面的注册通知相对应，它会通知所有注册的URI，数据库已经改变。

6、下面再来介绍一下update方法：

```
1     @Override
2     public int update(Uri uri, ContentValues values, String selection,
3             String[] selectionArgs) {
4         SQLiteDatabase db = dbHelper.getWritableDatabase();
5         int count;
6         switch (matcher.match(uri)) {
7             case QUERY_LIST:{
```

```
8                 count = db.update(NoteTableMetaData.TABLE_NAME,
                        values,selection, selectionArgs);
9             }break;
10            case QUERY_ITEM:{
11               String rowID = uri.getPathSegments().get(1);
12               count = db.update(NoteTableMetaData.TABLE_NAME, values,
13                           NoteTableMetaData.NOTE_ID + "=" + rowID
14                          + (!TextUtils.isEmpty(selection) ? ("and ( "
                            + selection + " ) ") : ""), selectionArgs);
15            }break;
16        default:
17            throw new IllegalArgumentException("Unknown URI " + uri);
18        }
19        getContext().getContentResolver().notifyChange(uri, null);
20        return count;
21    }
```


7、这里个方法里的内容和前面很类似，唯一的区别就是通过URI来确定更新的方式。这一点在代码中写的也比较明白，所以就不赘述了。最后再介绍一下delete 方法：

```
1     @Override
2     public int delete(Uri uri, String selection, String[] selectionArgs)
                  {
3         SQLiteDatabase db = dbHelper.getWritableDatabase();
4         int count;
5         switch (matcher.match(uri)) {
6                   case QUERY_LIST:{
7             count = db.delete(NoteTableMetaData.TABLE_NAME,
                        selection, selectionArgs);
8             }break;
9             case QUERY_ITEM:{
10                String rowID = uri.getPathSegments().get(1);
11                count = db.delete(NoteTableMetaData.TABLE_NAME,
12                        NoteTableMetaData.NOTE_ID + "="  + rowID +
13                        (!TextUtils.isEmpty(selection) ? (" and ( " +
                          selection + " ) ") : "" ), selectionArgs);
14            }break;
```

```
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
```

这个方法也是判断两种不同的URI，如果时QUERY_LIST类型的URI，那么它就会删除所有的记录（当然，如果我们不需要这种操作，也可以忽略这种URI），另一种就是根据ID来删除相应的记录。这个方法应该也不难理解。

大家注意到，我们在这些方法中，大量的用到了我们元数据类中的信息，这样做的好处前面也说过了，隔离了底层的表结构后，让数据库结构的修改变得非常容易。

8、当然，我们还需要实现getType方法，来返回不同URI对应的MIME类型，这个信息，在我们的元数据中已有定义：

```
    @Override
    public String getType(Uri uri) {
        switch (matcher.match(uri)) {
            case QUERY_LIST:{
                return NoteTableMetaData.CONTENT_TYPE;
            }
            case QUERY_ITEM:{
                return NoteTableMetaData.CONTENT_ITEM_TYPE;
            }
            default:{
                throw new IllegalArgumentException("Unknown URI " + uri);
            }
        }
    }
```

9、到现在位置我们的Provider就已经实现好了，最后不要忘了在manifest中注册这个Provider：

```
<provider android:name=".NoteProvider"
      android:authorities="org.spring.provider.NoteProvider"  />
```

10、到此为止，我们的数据访问接口就实现完成了，我们可以用下面的方式很容易的进行数据访问：

```
1         //插入数据
2         ContentValues values = new ContentValues();
3         values.put(NoteTableMetaData.NOTE_CONTENT, "test");
4         Uri insertedUri =
              getContentResolver().insert(NoteTableMetaData.CONTENT_URI,
              values);
5         //更新数据
6         values.put(NoteTableMetaData.NOTE_CONTENT, "test updated");
7 getContentResolver().update(Uri.withAppendedPath(NoteTableMetaData.CONTE
            NT_URI, "/" + insertedUri.getPathSegments().get(1)), values,
            null, null);
8         //查询数据
9         Cursor c =
                getContentResolver().query(NoteTableMetaData.CONTENT_URI,
                null, null, null, null);
10        //绑定 ListView
11        SimpleCursorAdapter adapter =
12            new SimpleCursorAdapter(this,
                  android.R.layout.simple_list_item_1, c,
13                    new String[] {"content" }, new int[]
                          {android.R.id.text1});
14        list.setAdapter(adapter);
15        //删除数据
16        getContentResolver().delete(insertedUri, null, null);
```

这篇文章主要是给大家提供了一种 ContentProvider 的架构方法，通过一些良好的代码组织，可以让我们的开发工作变得更加轻松，并且有效的增强应用的建壮性，对我们日常的开发还是很有帮助的。

当然这种方法并不是唯一的，更不敢说这个是最好的。更希望它能够起到一种抛砖引玉的作用，大家可以在这个基础之上进一步的探索，找出更加适合自己的架构方式。

以下是 Provider 的完整代码(省去 import 部分)：

```
1 public class NoteProvider extends ContentProvider{
2     private static HashMap<String, String> noteProjectionMap;
3     static {
```

```
4           noteProjectionMap = new HashMap<String, String>();
5           noteProjectionMap.put(NoteTableMetaData.NOTE_ID,
                NoteTableMetaData.NOTE_ID);
6           noteProjectionMap.put(NoteTableMetaData.NOTE_CONTENT,
                NoteTableMetaData.NOTE_CONTENT);
7           noteProjectionMap.put(NoteTableMetaData.NOTE_PUB_DATE,
                NoteTableMetaData.NOTE_PUB_DATE);
8       }
9       private static UriMatcher matcher;
10      private static final int QUERY_LIST = 1;
11      private static final int QUERY_ITEM = 2;
12      static {
13          matcher = new UriMatcher(UriMatcher.NO_MATCH);
14          matcher.addURI(NoteMetaData.AUTHORITY, "notes", QUERY_LIST);
15          matcher.addURI(NoteMetaData.AUTHORITY, "notes/#", QUERY_ITEM);
16      }
17      private DatabaseHelper dbHelper;
18      class DatabaseHelper extends SQLiteOpenHelper{
19          DatabaseHelper(Context context) {
20              super(context,   NoteMetaData.DATABASE_NAME, null,
                    NoteMetaData.DATABASE_VERSION);
21          }
22          @Override
23          public void onCreate(SQLiteDatabase db) {
24              db.execSQL(
25                      "create table " + NoteTableMetaData.TABLE_NAME + " ( "
26                      + NoteTableMetaData.NOTE_ID + " integer primary key, "
27                      + NoteTableMetaData.NOTE_CONTENT + " varchar(2000), "
28                      + NoteTableMetaData.NOTE_PUB_DATE + " integer"
29                      + ");"
30              );
31          }
32          @Override
33          public void onUpgrade(SQLiteDatabase db, int oldVersion, int
                newVersion) {
34              db.execSQL("drop table if exists " +
                    NoteTableMetaData.TABLE_NAME);
```

```
35              onCreate(db);
36          }
37      }
38      @Override
39      public String getType(Uri uri) {
40          switch (matcher.match(uri)) {
41              case QUERY_LIST:{
42                  return NoteTableMetaData.CONTENT_TYPE;
43              }
44              case QUERY_ITEM:{
45                  return NoteTableMetaData.CONTENT_ITEM_TYPE;
46              }
47              default:{
48                  throw new IllegalArgumentException("Unknown URI " + uri);
49              }
50          }
51      }
52      @Override
53      public Cursor query(Uri uri, String[] projection, String selection,
54              String[] selectionArgs, String sortOrder) {
55          SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
56          switch (matcher.match(uri)) {
57              case QUERY_LIST:{
58                  qb.setTables(NoteTableMetaData.TABLE_NAME);
59                  qb.setProjectionMap(noteProjectionMap);
60              }break;
61              case QUERY_ITEM:{
62                  qb.setTables(NoteTableMetaData.TABLE_NAME);
63                  qb.setProjectionMap(noteProjectionMap);
64                  qb.appendWhere(NoteTableMetaData.NOTE_ID + "=" +
                        uri.getPathSegments().get(1));
65              }break;
66          default:
67              throw new IllegalArgumentException("Unknown URI " + uri);
68          }
69          String orderBy;
```

```
70          if(TextUtils.isEmpty(sortOrder)) {
71              orderBy = NoteTableMetaData.DEFAULT_SORT_ORDER;
72          }else {
73              orderBy = sortOrder;
74          }
75          SQLiteDatabase db = dbHelper.getWritableDatabase();
76          Cursor c = qb.query(db, projection, selection, selectionArgs,
                  null, null, orderBy);
77          c.setNotificationUri(getContext().getContentResolver(), uri);
78          return c;
79      }
80      @Override
81      public int delete(Uri uri, String selection, String[] selectionArgs) {
82          SQLiteDatabase db = dbHelper.getWritableDatabase();
83          int count;
84          switch (matcher.match(uri)) {
85              case QUERY_LIST:{
86                  count = db.delete(NoteTableMetaData.TABLE_NAME, selection,
                          selectionArgs);
87              }break;
88              case QUERY_ITEM:{
89                  String rowID = uri.getPathSegments().get(1);
90                  count = db.delete(NoteTableMetaData.TABLE_NAME,
91                          NoteTableMetaData.NOTE_ID + "="  + rowID +
92                          (!TextUtils.isEmpty(selection) ? (" and ( " +
                                selection + " ) ") : "" ), selectionArgs);
93              }break;
94          default:
95              throw new IllegalArgumentException("Unknown URI " + uri);
96          }
97          getContext().getContentResolver().notifyChange(uri, null);
98          return count;
99      }
100     @Override
101     public Uri insert(Uri uri, ContentValues values) {
102         if(matcher.match(uri) != QUERY_LIST) {
```

```
103             throw new IllegalArgumentException("Unknown URI " + uri);
104         }
105         if(values.containsKey(NoteTableMetaData.NOTE_PUB_DATE) == false) {
106             Long now = Long.valueOf(System.currentTimeMillis());
107             values.put(NoteTableMetaData.NOTE_PUB_DATE, now);
108         }
109         SQLiteDatabase db = dbHelper.getWritableDatabase();
110         long rowID = db.insert(NoteTableMetaData.TABLE_NAME,
                NoteTableMetaData.NOTE_CONTENT, values);
111         if(rowID > 0) {
112             Uri insertedUri =
                    ContentUris.withAppendedId(NoteTableMetaData.CONTENT_URI,
                    rowID);
113 getContext().getContentResolver().notifyChange(insertedUri, null);
114             return insertedUri;
115         }
116         throw new android.database.SQLException("Failed to insert row into
                " + uri);
117     }
118     @Override
119     public boolean onCreate() {
120         dbHelper = new DatabaseHelper(getContext());
121         return true;
122     }
123     @Override
124     public int update(Uri uri, ContentValues values, String selection,
125             String[] selectionArgs) {
126         SQLiteDatabase db = dbHelper.getWritableDatabase();
127         int count;
128         switch (matcher.match(uri)) {
129             case QUERY_LIST:{
130                 count = db.update(NoteTableMetaData.TABLE_NAME,
                        values,selection, selectionArgs);
131             }break;
132             case QUERY_ITEM:{
133                String rowID = uri.getPathSegments().get(1);
134                count = db.update(NoteTableMetaData.TABLE_NAME, values,
```

```
135                             NoteTableMetaData.NOTE_ID + "=" + rowID
136                             + (!TextUtils.isEmpty(selection) ? ("and ( " +
                                    selection + "  ) ") : ""), selectionArgs);
137                 }break;
138             default:
139                 throw new IllegalArgumentException("Unknown URI " + uri);
140             }
141             getContext().getContentResolver().notifyChange(uri, null);
142             return count;
143         }
144 }
```

【其他】

4.1 BUG提交

如果你发现文档中有不妥的地方，请发邮件至 eoeandroid@eoemobile.com 进行反馈，我们会定期更新、发布更新后的版本。

4.2 关于eoeAndroid

eoeAndroid团队是一个有远大的梦想，有充沛的激情，有高效执行力的团队。北京易联致远无线技术有限公司于2009年8月成立。eoeAndroid的核心成员有在摩托罗拉、卓望科技、T3g和手机design house的工作经历和相关丰富的行业经验。eoeAndroid致力于让移动互联网的软件开发变得容易，发布变得更加方便，传播变得更加迅捷,让用户以最快的速度获取最适合自己的移动互联网应用。

4.3 eoe首届Android移动峰会在线报名

——开放能力，携手创赢

Android平台在经历“纸杯蛋糕”、“甜甜圈”、“冷冻奶酪”等一系列版本的演化后，终于迎来了全新的版本：“姜饼小人”。很难想象，仅三年时间，Android就成为增长速度最快的移动平台，市场占有率达到了全球第二。

与此同时，越来越多的开发者正投入到Android大潮中。这一次，中国的开发者们也没有落下速度，他们在这个强大的平台上，用自己精湛的技术，开发出越来越多的精良应用及游戏。据11月的数据统计，在eoeandroid这个中国最大的开发者社区里，有将近14万的开发者，大家每天在里面活跃着，不停的发帖，讨论技术，分享心得，每当开发出了新的应用便第一时间上传到优亿市场。

但是，在风风火火的背后，Android的未来是什么？赢利模式是什么？面对这些需要大家进行探讨的问题，eoe将结合自身在Android领域的丰富经验，及前瞻性思考，邀请业内精英，携手Forst&Sulliavn，举办“eoe首届Android移动峰会”。

报名地址：http://www.eoeandroid.com/thread-48972-1-1.html
活动时间： 2010-12-22（星期三） 下午 13:00-17:30
活动地点： 北京中关村皇冠假日酒店皇冠宴会厅（三层）
地　　址： 北京市海淀区知春路 106 号
电　　话： 010-59938888

本次峰会参会人员将包括：硬件厂商、VC、明星开发者和电信运营商等，演讲嘉宾有 Forst&Sulliavn 中国区首席顾问王煜全、Mobile 2.0 论坛创始人 Leo、Google Adsense 中国区负责人。

eoe 有和 Google、联想等一流公司举办开发者大会的经验，相信这次也一定会给大家带来惊喜。在本次大会上，除了参会者皆有 eoe 特别定制礼品赠送，更有新款 Android 手机抽奖环节。当然，远不止这些，对一个会议来说，每个人都会有不一样的收获，提升开发技能，获得全新的产品开发思路是绝对的，你甚至有可能收获成就你事业的第一笔投资，结识你人生道路上又一个志趣相投的盟友。

所有结果，都将在 12 月 22 日北京中关村皇冠假日酒店展现。欢迎各位同学报名前来参会！

峰会详情及报名请点击→我要参加

北京易联致远无限技术有限公司

责任编辑：莫言默语

美术支持：tiantian　技术支持：gaotong86

电话：（010）82176766

E-mail：eoeandroid@eoemobile.com

中国最大的 Android 开发者社区：www.eoeandroid.com

中国本土的 Android 软件下载平台：www.eoemarket.com