第二十二期：Android 源码修改

eoe 特刊

eoe 开发者门户 | 优亿市场 Android应用发布与分享平台

目录

【Android 源码修改】

一、说说Android的基础问题

1. Android 源码分支概要

Android 源码在不断升级，所以会形成很多版本和分支，我们下载源码时候，可以按照自己需求来下载相应版本或是分支的 Android 源码，以下是目前最新的 Android 源码版本和分支信息：

```
From git://android.git.kernel.org/tools/repo

 * [new branch]      maint      -> origin/maint

 * [new branch]      master     -> origin/master

 * [new branch]      stable     -> origin/stable

 * [new tag]         v1.6.9.6   -> v1.6.9.6

From git://android.git.kernel.org/tools/repo

 * [new tag]         v1.0       -> v1.0

 * [new tag]         v1.0.1     -> v1.0.1

 * [new tag]         v1.0.2     -> v1.0.2

 * [new tag]         v1.0.3     -> v1.0.3

 * [new tag]         v1.0.4     -> v1.0.4

 * [new tag]         v1.0.5     -> v1.0.5

 * [new tag]         v1.0.6     -> v1.0.6

 * [new tag]         v1.0.7     -> v1.0.7

 * [new tag]         v1.0.8     -> v1.0.8

 * [new tag]         v1.0.9     -> v1.0.9

 * [new tag]         v1.1       -> v1.1
```

```
* [new tag]         v1.2       -> v1.2
* [new tag]         v1.3       -> v1.3
* [new tag]         v1.3.1     -> v1.3.1
* [new tag]         v1.3.2     -> v1.3.2
* [new tag]         v1.4       -> v1.4
* [new tag]         v1.4.1     -> v1.4.1
* [new tag]         v1.4.2     -> v1.4.2
* [new tag]         v1.4.3     -> v1.4.3
* [new tag]         v1.4.4     -> v1.4.4
* [new tag]         v1.5       -> v1.5
* [new tag]         v1.5.1     -> v1.5.1
* [new tag]         v1.6       -> v1.6
* [new tag]         v1.6.1     -> v1.6.1
* [new tag]         v1.6.2     -> v1.6.2
* [new tag]         v1.6.3     -> v1.6.3
* [new tag]         v1.6.4     -> v1.6.4
* [new tag]         v1.6.5     -> v1.6.5
* [new tag]         v1.6.6     -> v1.6.6
* [new tag]         v1.6.7     -> v1.6.7
* [new tag]         v1.6.7.1   -> v1.6.7.1
* [new tag]         v1.6.7.2   -> v1.6.7.2
* [new tag]         v1.6.7.3   -> v1.6.7.3
* [new tag]         v1.6.7.4   -> v1.6.7.4
```

```
* [new tag]         v1.6.7.5   -> v1.6.7.5
* [new tag]         v1.6.8     -> v1.6.8
* [new tag]         v1.6.8.1   -> v1.6.8.1
* [new tag]         v1.6.8.10  -> v1.6.8.10
* [new tag]         v1.6.8.11  -> v1.6.8.11
* [new tag]         v1.6.8.2   -> v1.6.8.2
* [new tag]         v1.6.8.3   -> v1.6.8.3
* [new tag]         v1.6.8.4   -> v1.6.8.4
* [new tag]         v1.6.8.5   -> v1.6.8.5
* [new tag]         v1.6.8.6   -> v1.6.8.6
* [new tag]         v1.6.8.7   -> v1.6.8.7
* [new tag]         v1.6.8.8   -> v1.6.8.8
* [new tag]         v1.6.8.9   -> v1.6.8.9
* [new tag]         v1.6.9     -> v1.6.9
* [new tag]         v1.6.9.1   -> v1.6.9.1
* [new tag]         v1.6.9.2   -> v1.6.9.2
* [new tag]         v1.6.9.3   -> v1.6.9.3
* [new tag]         v1.6.9.4   -> v1.6.9.4
* [new tag]         v1.6.9.5   -> v1.6.9.5
Getting manifest ...
From git://android.git.kernel.org/platform/manifest
 * [new branch]      android-1.5 -> origin/android-1.5
 * [new branch]      android-1.5r2 -> origin/android-1.5r2
```

```
* [new branch]      android-1.5r3 -> origin/android-1.5r3
* [new branch]      android-1.5r4 -> origin/android-1.5r4
* [new branch]      android-1.6_r1 -> origin/android-1.6_r1
* [new branch]      android-1.6_r1.1 -> origin/android-1.6_r1.1
* [new branch]      android-1.6_r1.2 -> origin/android-1.6_r1.2
* [new branch]      android-1.6_r1.3 -> origin/android-1.6_r1.3
* [new branch]      android-1.6_r1.4 -> origin/android-1.6_r1.4
* [new branch]      android-1.6_r1.5 -> origin/android-1.6_r1.5
* [new branch]      android-1.6_r2 -> origin/android-1.6_r2
* [new branch]      android-2.0.1_r1 -> origin/android-2.0.1_r1
* [new branch]      android-2.0_r1 -> origin/android-2.0_r1
* [new branch]      android-2.1_r1 -> origin/android-2.1_r1
* [new branch]      android-2.1_r2 -> origin/android-2.1_r2
* [new branch]      android-2.1_r2.1p -> origin/android-2.1_r2.1p
* [new branch]      android-2.1_r2.1p2 -> origin/android-2.1_r2.1p2
* [new branch]      android-2.1_r2.1s -> origin/android-2.1_r2.1s
* [new branch]      android-sdk-1.5-pre -> origin/android-sdk-1.5-pre
* [new branch]      android-sdk-1.5_r1 -> origin/android-sdk-1.5_r1
* [new branch]      android-sdk-1.5_r3 -> origin/android-sdk-1.5_r3
* [new branch]      android-sdk-1.6-docs_r1 -> origin/android-sdk-1.6-docs_r1
* [new branch]      android-sdk-1.6_r1 -> origin/android-sdk-1.6_r1
* [new branch]      android-sdk-1.6_r2 -> origin/android-sdk-1.6_r2
* [new branch]      android-sdk-2.0.1-docs_r1 ->
```

```
origin/android-sdk-2.0.1-docs_r1

* [new branch]      android-sdk-2.0.1_r1 -> origin/android-sdk-2.0.1_r1

* [new branch]      android-sdk-2.0_r1 -> origin/android-sdk-2.0_r1

* [new branch]      android-sdk-2.1_r1 -> origin/android-sdk-2.1_r1

* [new branch]      android-sdk-tools_r2 -> origin/android-sdk-tools_r2

* [new branch]      android-sdk-tools_r3 -> origin/android-sdk-tools_r3

* [new branch]      android-sdk-tools_r4 -> origin/android-sdk-tools_r4

* [new branch]      android-sdk-tools_r5 -> origin/android-sdk-tools_r5

* [new branch]      cdma-import -> origin/cdma-import

* [new branch]      cupcake     -> origin/cupcake

* [new branch]      cupcake-release -> origin/cupcake-release

* [new branch]      donut       -> origin/donut

* [new branch]      donut-plus-aosp -> origin/donut-plus-aosp

* [new branch]      eclair      -> origin/eclair

* [new branch]      master      -> origin/master

* [new branch]      release-1.0 -> origin/release-1.0

* [new tag]         android-1.5 -> android-1.5

* [new tag]         android-1.5r2 -> android-1.5r2

* [new tag]         android-1.5r3 -> android-1.5r3

* [new tag]         android-1.5r4 -> android-1.5r4

* [new tag]         android-1.6_r1 -> android-1.6_r1

* [new tag]         android-1.6_r1.1 -> android-1.6_r1.1

* [new tag]         android-1.6_r1.2 -> android-1.6_r1.2
```

```
* [new tag]         android-1.6_r1.3 -> android-1.6_r1.3
* [new tag]         android-1.6_r1.4 -> android-1.6_r1.4
* [new tag]         android-1.6_r1.5 -> android-1.6_r1.5
* [new tag]         android-1.6_r2 -> android-1.6_r2
* [new tag]         android-2.0.1_r1 -> android-2.0.1_r1
* [new tag]         android-2.0_r1 -> android-2.0_r1
* [new tag]         android-2.1_r1 -> android-2.1_r1
* [new tag]         android-2.1_r2 -> android-2.1_r2
* [new tag]         android-2.1_r2.1p -> android-2.1_r2.1p
* [new tag]         android-2.1_r2.1p2 -> android-2.1_r2.1p2
* [new tag]         android-2.1_r2.1s -> android-2.1_r2.1s
* [new tag]         android-sdk-1.5-pre -> android-sdk-1.5-pre
* [new tag]         android-sdk-1.5_r1 -> android-sdk-1.5_r1
* [new tag]         android-sdk-1.5_r3 -> android-sdk-1.5_r3
* [new tag]         android-sdk-1.6-docs_r1 -> android-sdk-1.6-docs_r1
* [new tag]         android-sdk-1.6_r1 -> android-sdk-1.6_r1
* [new tag]         android-sdk-1.6_r2 -> android-sdk-1.6_r2
* [new tag]         android-sdk-2.0.1-docs_r1 -> android-sdk-2.0.1-docs_r1
* [new tag]         android-sdk-2.0.1_r1 -> android-sdk-2.0.1_r1
* [new tag]         android-sdk-2.0_r1 -> android-sdk-2.0_r1
* [new tag]         android-sdk-2.1_r1 -> android-sdk-2.1_r1
* [new tag]         android-sdk-tools_r2 -> android-sdk-tools_r2
* [new tag]         android-sdk-tools_r3 -> android-sdk-tools_r3
```

```
 * [new tag]         android-sdk-tools_r4 -> android-sdk-tools_r4

 * [new tag]         android-sdk-tools_r5 -> android-sdk-tools_r5

From git://android.git.kernel.org/platform/manifest

 * [new tag]         android-1.0 -> android-1.0
```

使用[new tag]或是[new branch] 后面指定的版本或是分支就可以下载到对应的 Android 版本的源码。

初始化与配置 Repo

初始化--在终端中键入以下内容：

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

配置--在终端中键入以下内容：

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

或是

```
repo init -u git://android.git.kernel.org/platform/manifest.git -b [Android 版本/分支名]
```

如果不指定-b 后面的 Android 版本或分支名参数，那么就会下载到最新版本的 Android 源码。

配置成功后会显示一下字样：

```
repo initialized in /android
```

下载 Android 源码

直接在刚刚终端中键入：

```
$ repo sync
```

等待一至两小时，下载就会完成。

2.获取 Android 源代码及搭建开发环境

Supported Operating Systems：

Windows XP or Vista

Mac OS X 10.4.8 or later （x86 only）

Linux （tested on Linux Ubuntu Dapper Drake）

由于在 Linux 环境下 Android Source Code 的获取和编译更为方便，本文主要介绍基于 Linux Ubuntu 的搭建过程.

推荐的操作系统

Ubuntu 6.06 （Dapper）， 7.10 （Gutsy）， and 8.04 （Hardy）

安装和获取 Ubuntu CD 可参考 http：//www.ubuntu.org.cn/

Ubuntu 的安装过程不复杂，放入 CD 启动电脑，按照提示一步步安装即可.

安装好 Ubuntu 后，首先需要设置 apt-get 代理上网

方法一

这是一种临时的手段，如果您仅仅是暂时需要通过 http 代理使用 apt-get，您可以使用这种方式.

在使用 apt-get 之前，在终端中输入以下命令（根据您的实际情况替换 yourproxyaddress 和 proxyport）.

终端运行 export http_proxy="http：//用户名：密码@代理 IP：代理端口"也可

方法二

这种方法要用到/etc/apt/文件夹下的 apt.conf 文件.如果您希望 apt-get（而不是其他应用程序）一直使用 http 代理，您可以使用这种方式.

注意： 某些情况下，系统安装过程中没有建立 apt 配置文件.下面的操作将视情况修改现有的配置文件或者新建配置文件.

sudo gedit /etc/apt/apt.conf

在您的 apt.conf 文件中加入下面这行（根据你的实际情况替换 yourproxyaddress 和 proxyport）.

Acquire：：http：：Proxy “http：//yourproxyaddress：proxyport”;

保存 apt.conf 文件.

方法三

这种方法会在您的主目录下的.bashrc 文件中添加两行.如果您希望 apt-get 和其他应用程序如 wget 等都使用 http 代理，您可以使用这种方式.

gedit ~/.bashrc

在您的.bashrc 文件末尾添加如下内容（根据你的实际情况替换 yourproxyaddress 和 proxyport）.

http_proxy=http：//yourproxyaddress：proxyport

export http_proxy

保存文件.关闭当前终端，然后打开另一个终端.

使用 apt-get update 或者任何您想用的网络工具测试代理.我使用 firestarter 查看活动的网络连接.

如果您为了纠正错误而再次修改了配置文件，记得关闭终端并重新打开，否自新的设置不会生效.

apt-get 设置好后，按照下面的步骤安装相关组件

Required Packages （Ubuntu 8.04）

$ sudo apt-get install python2.5

$ sudo apt-get install sun-java6-jdk sun-java6-bin sun-java6-jre

Add/Edit /etc/bash.bashrc

export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.07

$ sudo apt-get install flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev zlib1g-dev zip curl valgrind libncurses5-dev build-essential

$ sudo apt-get install x-dev

$ sudo apt-get install libx11-dev

安装 git

Install Packages

$ sudo apt-get install git-core gnupg

Git 是一种版本管理的工具

安装好 git 后同样需要进行设置代理.

下载，编译 connect.c.

$ cd ~/bin

$ curl http：//www.meadowy.org/~gotoh/ssh/connect.c > connect.c

$ gcc -o connect connect.c

编写一个简单的脚本文件.

#！/bin/sh

Filename： ~/bin/http-proxy-gw

This script connects to an HTTP proxy using connect.c

connect -H http：//yourproxyaddress：proxyport $@

设置运行权限

$ chmod +x ~/bin/http-proxy-gw

现在，需要告诉 git 使用这个脚本访问代理服务器.

$ git config --global core.gitproxy "http-proxy-gw for kernel.org"

安装 repo

1.确认你有一个 bin 目录在你 home 目录下，没有则创建一个，并设置它到 PATH 环境变量中.

$ cd ~

$ mkdir bin

$ echo $PATH

2.下载 repo，并给它可执行权限：

$ curl http：//android.git.kernel.org/repo >~/bin/repo

$ chmod a+x ~/bin/repo

3.建立一个空目录保存 Android 代码.

$ mkdir mydroid

$ cd mydroid

4.获取 manifest 文件

$ repo init -u git：//android.git.kernel.org/platform/manifest.git

5.按照命令提示输入 name 和 address.如果你想提交 code，输入一个 Google account.

运行成功后应当提示如下信息.

repo initialized in /mydroid

开始下载 Android source code， run

$ repo sync

编译 Android source code

$ cd ~/mydroid

$ export ANDROID_JAVA_HOME=$JAVA_HOME

$ cd ~/mydroid

$ make

编译 Android Kernel

Building zImage

Go into kernel directory

$ cd ~/mydroid/kernel

$ make msm_defconfig ARCH=arm

$ make ARCH=arm CROSS_COMPILE=/prebuilt/linux-x86/toolchain/arm-

eabi-4.2.1/bin/arm-eabi-

可以在 kernel/arch/arm/boot/目录下看到 build 好的 zImage

3.三星 Galaxy S2 i9100 内核源代码下载

对于三星下一代高端 Android 手机 Galaxy S2 i9100 的内核源代码目前已经开始提供下载，对于 ROM 制作来说提早下载可以解决很多问题，由于目前 i9100 即将在本月上市这款使用双核 1.2GHz 处理器，1GB RAM 的怪兽级手机拥有超薄的 8.6mm 机身和 4.3 英寸触控屏.目前从代码上来看使用的是三星自己定制的 Android 2.3.3 版固件，相对于 i9000 来说性能几乎翻倍必将成为下一代机王.

https://opensource.samsung.com/reception/reception_main.do?method=reception_search&searchValue=I9100

二、说说源码结构。

1.Android 2.1源码结构

Android 2.1

|-- 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

Android 2.1
|-- 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               （？作用不明，java语言，网上有人说和JDK5有关）
|-- 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              （？）
|-- 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）

|-- e2fsprogs （EXT2文件系统工具）

|-- elfcopy （复制ELF的工具）

|-- elfutils （ELF工具）

|-- embunit （Embedded Unit Project）

|-- emma （java代码覆盖率统计工具）

|-- esd （Enlightened Sound Daemon，将多种音频流混合在一个设备上播放）

|-- expat （Expat is a stream-oriented XML parser.）

|-- fdlibm （FDLIBM (Freely Distributable LIBM)）

|-- freetype （字体）

|-- fsck_msdos （dos文件系统检查工具）

|-- gdata （google的无线数据相关）

|-- genext2fs （genext2fs generates an ext2 filesystem as a normal (non-root) user）

|-- giflib （gif库）

|-- googleclient （google用户库）

|-- grub （This is GNU GRUB, the GRand Unified Bootloader.）

|-- gtest （Google C++ Testing Framework）

|-- icu4c （ICU(International Component for Unicode)在C/C++下的版本）

|-- ipsec-tools （This package provides a way to use the native IPsec functionality ）

|-- iptables （防火墙）

|-- jdiff （generate a report describing the difference between two public Java APIs.）

|-- jhead （jpeg头部信息工具）

|-- jpeg （jpeg库）

|-- junit （JUnit是一个Java语言的单元测试框架）

|-- kernel-headers （内核的一些头文件）

|-- libffi （libffi is a foreign function interface library.）

|-- libpcap （网络数据包捕获函数）

|-- libpng （png库）

|-- libxml2 （xml解析库）

|-- mtpd （一个命令）
|-- 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）

frameworks 目录 （核心框架——java及C++语言）
.
|-- base （基本内容）
| |-- api （？都是xml文件，定义了java的api？）
| |-- awt （AWT库）
| |-- build （空的）
| |-- camera （摄像头服务程序库）
| |-- cmds （重要命令：am、app_proce等）
| |-- core （核心库）
| |-- data （字体和声音等数据文件）
| |-- docs （文档）
| |-- graphics （图形相关）

```
|     |-- include                 （头文件）
|     |-- keystore            （和数据签名证书相关）
|     |-- libs                    （库）
|     |-- location            （地区库）
|     |-- media                   （媒体相关库）
|     |-- obex                    （蓝牙传输库）
|     |-- opengl                  （2D-3D加速库）
|     |-- packages            （设置、TTS、VPN程序）
|     |-- sax                     （XML解析器）
|     |-- services            （各种服务程序）
|     |-- telephony           （电话通讯管理）
|     |-- test-runner         （测试工具相关）
|     |-- tests                   （各种测试）
|     |-- tools                   （一些叫不上名的工具）
|     |-- vpn                     （VPN）
|     `-- wifi                    （无线网络）
|-- opt                           （可选部分）
|     |-- com.google.android                                （有个framework.jar）
|     |-- com.google.android.googlelogin                （有个client.jar）
|     `-- emoji                 （standard message elements）
`-- policies                （Product policies are operating system directions aimed at specific uses）
      `-- base
            |-- mid           （MID设备）
            `-- phone         （手机类设备，一般用这个）

hardware 目录                   （部分厂家开源的硬解适配层HAL代码）
|-- broadcom                        （博通公司）
|     `-- wlan                      （无线网卡）
|-- libhardware                     （硬件库）
|     |-- include                   （头文件）
|     `-- modules               （Default (and possibly architecture dependents) HAL modules）
|           |-- gralloc             （gralloc显示相关）
|           `-- overlay             （Skeleton for the "overlay" HAL module.）
|-- libhardware_legacy          （旧的硬件库）
|     |-- flashlight              （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）
     |    |-- dspbridge                （DSP桥）
     |    |-- libopencorehw            （opencore硬件库）
     |    |-- liboverlay               （overlay硬件库）
     |    |-- libstagefrighthw         （stagefright硬件库）
     |    `-- omx                      （omx组件）
     `-- wlan                          （无线网卡）
```

packages 目录

```
.
|-- apps                               （应用程序库）
|    |-- AlarmClock                    （闹钟）
|    |-- Bluetooth                     （蓝牙）
|    |-- Browser                       （浏览器）
|    |-- Calculator                    （计算器）
|    |-- Calendar                      （日历）
|    |-- Camera                        （相机）
|    |-- CertInstaller                 （在Android中安装数字签名，被调用）
|    |-- Contacts                      （拨号(调用)、联系人、通话记录）
|    |-- DeskClock                     （桌面时钟）
|    |-- Email                         （Email）
|    |-- Gallery                       （相册，和Camera类似，多了列表）
|    |-- Gallery3D                     （？3D相册）
|    |-- GlobalSearch                  （为google搜索服务，提供底层应用）
|    |-- GoogleSearch                  （google搜索）
|    |-- HTMLViewer                    （浏览器附属界面，被浏览器应用调用，同时提供存储记录功能）
```

```
|     |-- IM                          （即时通讯，为手机提供信号发送、接收、通信的服务）
|     |-- Launcher                    （登陆启动项，显示图片框架等等图形界面）
|     |-- Launcher2                   （登陆启动项，负责应用的调用）
|     |-- Mms                         （？彩信业务）
|     |-- Music                       （音乐播放器）
|     |-- PackageInstaller            （安装、卸载程序的响应）
|     |-- Phone                       （电话拨号程序）
|     |-- Provision                   （预设应用的状态，使能应用）
|     |-- Settings                    （开机设定，包括电量、蓝牙、设备信息、界面、wifi等）
|     |-- SoundRecorder               （录音机，可计算存储所需空间和时间）
|     |-- Stk                         （接收和发送短信）
|     |-- Sync                        （空）    ------O1
|     |-- Updater                     （空）
|     `-- VoiceDialer                 （语音识别通话）
|-- inputmethods                      （输入法）
|     |-- LatinIME                    （拉丁文输入法）
|     |-- OpenWnn                     （OpenWnn输入法）
|     `-- PinyinIME                   （拼音输入法）
|-- providers                         （提供器，提供应用程序、界面所需的数据）
|     |-- ApplicationsProvider        （应用程序提供器，提供应用程序启动项、更新等）
|     |-- CalendarProvider            （日历提供器）
|     |-- ContactsProvider            （联系人提供器）
|     |-- DownloadProvider            （下载管理提供器）
|     |-- DrmProvider                 （创建和更新数据库时调用）
|     |-- GoogleContactsProvider      （联系人提供器的子类，用以同步联系人）
|     |-- GoogleSubscribedFeedsProvider（设置信息提供器）
|     |-- ImProvider                  （空）
|     |-- ManagementProvider          （空）
|     |-- MediaProvider               （媒体提供器，提供存储数据）
|     |-- TelephonyProvider           （彩信提供器）
|     |-- UserDictionaryProvider      （用户字典提供器，提供用户常用字字典）
|     `-- WebSearchProvider           （空）
|-- services
|     |-- EasService                  （空）
|     `-- LockAndWipe                 （空）
`-- wallpapers                        （墙纸）
      |-- Basic                       （基本墙纸，系统内置墙纸）
      |-- LivePicker                  （选择动态壁纸）
      |-- MagicSmoke                  （壁纸特殊效果）
      `-- MusicVisualization          （音乐可视化，图形随音乐而变化）
```

O1里面有一个隐藏的.git文件夹，内容都是一样的，没有有意义的代码，config看似乎是一个下载程序，因此认为这些文件夹下没有实质东西。

prebuilt 目录 （x86和arm架构下预编译的一些资源）
.
|-- android-arm （arm-android相关）
| |-- gdbserver （gdb调试器）
| `-- kernel （模拟的arm内核）
|-- android-x86 （x86-android相关）
| `-- kernel （空的）
|-- common （通用编译好的代码，应该是java的）
|-- darwin-x86 （drawin x86平台）
| `-- toolchain （工具链）
| |-- arm-eabi-4.2.1
| |-- arm-eabi-4.3.1
| `-- arm-eabi-4.4.0
|-- darwin-x86_64 （drawin x86 64bit平台）
|-- linux-x86 （linux x86平台）
| `-- toolchain （工具链，我们应该主要用这个）
| |-- arm-eabi-4.2.1
| |-- arm-eabi-4.3.1
| |-- arm-eabi-4.4.0
| `-- i686-unknown-linux-gnu-4.2.1 （x86版编译器）
|-- linux-x86_64 （linux x86 64bit平台）
|-- windows （windows平台）
`-- windows-x86_64 （64bit windows平台）
system 目录 （底层文件系统库、应用及组件——C语言）
.
|-- Bluetooth （蓝牙相关）
|-- 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）
| |-- libzipfile （zip库）
| |-- logcat （查看log工具）
| |-- logwrapper （log封装工具）

```
|     |-- mkbootimg          （制作启动boot.img的工具盒脚本）
|     |-- netcfg             （网络配置netcfg源码）
|     |-- nexus              （google最新手机的代码）
|     |-- rootdir            （rootfs，包含一些etc下的脚本和配置）
|     |-- sh                 （shell代码）
|     |-- toolbox            （toolbox，类似busybox的工具集）
|     `-- vold               （SD卡管理器）
|-- extras                   （额外工具）
|     |-- latencytop         （a tool for software developers ， identifying system latency happen）
|     |-- libpagemap         （pagemap库）
|     |-- librank            （Java Library Ranking System库）
|     |-- procmem            （pagemap相关）
|     |-- procrank           （Java Library Ranking System相关）
|     |-- showmap            （showmap工具）
|     |-- showslab           （showslab工具）
|     |-- sound              （声音相关）
|     |-- su                 （su命令源码）
|     |-- tests              （一些测试工具）
|     `-- timeinfo           （时区相关）
`-- wlan                     （无线相关）
      `-- ti                 （ti网卡相关工具及库）
```

```
vendor 目录                  （厂家定制内容）

|-- aosp                     （android open source project）
|     `-- products           （一些板级规则）
|-- htc                      （HTC公司）
|     |-- common-open        （通用部分）
|     |     `-- akmd         （解压img用的工具）
|     |-- dream-open         （G1开放部分）
|     |-- prebuilt-open      （预编译开放部分）
|     `-- sapphire-open      （sapphire这款型号开放内容）
|-- pv-open                  （没东西）
|-- qcom                     （里面基本是空的）
`-- sample                   （google提供的样例）
      |-- apps               （应用）
      |     |-- client       （用户）
      |     `-- upgrade      （升级）
      |-- frameworks         （框架）
      |     `-- PlatformLibrary （平台库）
      |-- products           （产品）
      |-- sdk_addon          （sdk添加部分）
      `-- skins              （皮肤）
```

`– WVGAMedDpi （WVGA适用的图片）

2.Android 2.2 源码结构

Android 2.2 源码结构分析

Google提供的Android包含了原始Android的目标机代码，主机编译工具、仿真环境，代码包经过解压缩后，第一级别的目录和文件如下所示：

.
|– Makefile （全局的Makefile）
|– bionic （Bionic含义为仿生，这里面是一些基础的库的源代码）
|– bootloader （引导加载器）
|– build （build目录中的内容不是目标所用的代码，而是编译和配置所需要的脚本和工具）
|– dalvik （JAVA虚拟机）
|– development （程序开发所需要的模板和工具）
|– external （目标机器使用的一些库）
|– frameworks （应用程序的框架层）
|– hardware （与硬件相关的库）
|– kernel （Linux2.6的源代码）
|– packages （Android的各种应用程序）
|– prebuilt （Android在各种平台下编译的预置脚本）
|– recovery （与目标的恢复功能相关）
`– system （Android的底层的一些库）

bionic目录展开一个级别的目录如下所示：
bionic/
|– Android.mk
|– libc
|– libdl
|– libm
|– libstdc++
|– libthread_db
`– linker

bootloader目录展开的两个级别目录：
bootloader/
`– legacy
|– Android.mk
|– README
|– arch_armv6
|– arch_msm7k
|– fastboot_protocol.txt

|– include
|– libboot
|– libc
|– nandwrite
`– usbloader

build目录展开的一个级别的目录如下所示：
build/
|– buildspec.mk.default
|– cleanspec.mk
|– core （各种以mk为结尾的文件，它们是编译所需要的Makefile）
|– envsetup.sh
|– libs
|– target （包含board和product两个目录，为目标所需要文件）
`– tools （编译过程中主机所需要的工具，一些需要经过编译生成）

其中，core中的Makefile是整个Android编译所需要的真正的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。
dalvik目录用于提供Android JAVA应用程序运行的基础————JAVA虚拟机。
development目录展开的一个级别的目录如下所示：
development
|– apps （Android应用程序的模板）
|– build （编译脚本模板）
|– cmds
|– data
|– docs
|– emulator （仿真相关）
|– host （包含windows平台的一些工具）
|– ide
|– pdk
|– samples （一些示例程序）
|– simulator （大多是目标机器的一些工具）
`– tools

在emulator目录中qemud是使用QEMU仿真时目标机器运行的后台程序，skins是仿真时手机的界面。
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。
external目录展开的一个级别的目录如下所示：
external/
|– aes
|– apache-http
|– bluez
|– clearsilver
|– dbus
|– dhcpcd

|– dropbear
|– elfcopy
|– elfutils
|– emma
|– esd
|– expat
|– fdlibm
|– freetype
|– gdata
|– giflib
|– googleclient
|– icu4c
|– iptables
|– jdiff
|– jhead
|– jpeg
|– libffi
|– libpcap
|– libpng
|– libxml2
|– netcat
|– netperf
|– neven
|– opencore
|– openssl
|– oprofile
|– ping
|– ppp
|– protobuf
|– qemu
|– safe-iop
|– skia
|– sonivox
|– sqlite
|– srec
|– strace
|– tagsoup
|– tcpdump
|– tinyxml
|– tremor
|– webkit
|– wpa_supplicant
|– yaffs2
`– zlib

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。其中：
opencore为PV（PacketVideo），它是Android多媒体框架的核心。
webkit是Android网络浏览器的核心。
sqlite是Android数据库系统的核心。
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。
frameworks目录展开的一个级别的目录如下所示：

```
frameworks/
|– base
|– opt
`– policies
```

frameworks是Android应用程序的框架。
hardware是一些与硬件相关的库
kernel是Linux2.6的源代码

packages目录展开的两个级别的目录如下所示：

```
packages/
|– apps
| |– AlarmClock
| |– Browser
| |– Calculator
| |– Calendar
| |– Camera
| |– Contacts
| |– Email
| |– GoogleSearch
| |– HTMLViewer
| |– IM
| |– Launcher
| |– Mms
| |– Music
| |– PackageInstaller
| |– Phone
| |– Settings
| |– SoundRecorder
| |– Stk
| |– Sync
| |– Updater
| `– VoiceDialer
`– providers
|– CalendarProvider
|– ContactsProvider
|– DownloadProvider
```

|– DrmProvider
|– GoogleContactsProvider
|– GoogleSubscribedFeedsProvider
|– ImProvider
|– MediaProvider
`– TelephonyProvider

packages中包含两个目录，其中apps中是Android中的各种应用程序，providers是一些内容提供者（在Android中的一个数据源）。

packages中两个目录的内容大都是使用JAVA编写的程序，各个文件夹的层次结构是类似的。
prebuilt目录展开的一个级别的目录如下所示：
prebuilt/
|– Android.mk
|– android-arm
|– common
|– darwin-x86
|– linux-x86
`– windows

system目录展开的两个级别的目录如下所示：
system/
|– bluetooth
| |– bluedroid
| `– brfpatch
|– core
| |– Android.mk
| |– README
| |– adb
| |– cpio
| |– debuggerd
| |– fastboot
| |– include （各个库接口的头文件）
| |– init
| |– libctest
| |– libcutils
| |– liblog
| |– libmincrypt
| |– libnetutils
| |– libpixelflinger
| |– libzipfile
| |– logcat
| |– logwrapper
| |– mkbootimg
| |– mountd

```
| |– netcfg
| |– rootdir
| |– sh
| `– toolbox
|– extras
| |– Android.mk
| |– latencytop
| |– libpagemap
| |– librank
| |– procmem
| |– procrank
| |– showmap
| |– showslab
| |– sound
| |– su
| |– tests
| `– timeinfo
`– wlan
`– ti
```

三、源码分析与研究

1.Android源码分析总结

在 Android 中进行处理 Android 源码中，hardware/ril 目录中包含着 Android 的 telephony 源码，下面就由我向大家介绍这些目录其中包含了三个子目录，下面是对三个子目录进行具体的分析说明.

一、目录 hardware/ril/include 分析：

只有一个头文件 ril.h 包含在此目录下.ril.h 中定义了 76 个如下类型的宏：这些宏代表着客户进程可以向 Android 源码 telephony 发送的命令，包括 SIM 卡相关的功能，打电话，发短信，网络信号查询等.好像没有操作地址本的功能？

二、目录 hardware/ril/libril 分析.本目录下代码负责与客户进程进行交互.在接收客户进程命令后，调用相应函数进行处理，然后将命令响应结果传回客户进程.在收到来自网络端的事件后，也传给客户进程.

文件 ril_commands.h：列出了 telephony 可以接收的命令；每个命令对应的处理函数；以及命令响应的处理函数. 文件 ril_unsol_commands.h：列出了 telephony 可以接收的事件类型；对每个事件的处理函数；

以及 WAKE Type 文件 ril_event.h/cpp：处理与事件源（端口，modem 等）相关的功能.ril_event_loop 监视所有注册的事件源，当某事件源有数据到来时，相应事件源的回调函数被触发（firePending -> ev->func（））

listenCallback 函数：当与客户进程连接建立时，此函数被调用.此函数接着调用 processCommandsCallback 处

理来自客户进程的命令请求 processCommandsCallback 函数：具体处理来自客户进程的命令请求.

对每一个命令，ril_commands.h 中都规定了对应的命令处理函数（dispatchXXX），processCommandsCallback 会调用这个命令处理函数进行处理.dispatch 系列函数：此函数接收来自客户进程的命令己相应参数，并调用 onRequest 进行处理.

RIL_onUnsolicitedResponse 函数：将来自网络端的事件封装（通过调用 responseXXX）后传给客户进程，RIL_onRequestComplete 函数：将命令的最终响应结构封装（通过调用 responseXXX）后传给客户进程.

response 系列函数：对每一个命令，都规定了一个对应的 response 函数来处理命令的最终响应；对每一个网络端的事件，也规定了一个对应的 response 函数来处理此事件.response 函数可被 onUnsolicitedResponse 或者 onRequestComplete 调用.

三、目录 hardware/ril/reference-ril 分析.本目录下代码主要负责与 modem 进行交互. 文件 reference-ril.c：此文件核心是两个函数：onRequest 和 onUnsolicited .

onRequest 函数：在这个函数里，对每一个 RIL_REQUEST_XXX 请求，都转化成相应的 AT command，发送给 modem，然后睡眠等待.当收到此 AT command 的最终响应后，线程被唤醒，将响应传给客户进程（RIL_onRequestComplete -> sendResponse）.

onUnsolicited 函数：这个函数处理 modem 从网络端收到的各种事件，如网络信号变化，拨入的电话，收到短信等.然后将时间传给客户进程（RIL_onUnsolicitedResponse -> sendResponse）文件 atchannel.c：负责向 modem 读写数据.其中，写数据（主要是 AT command）功能运行在主线程中，读数据功能运行在一个单独的读线程中.

Android 源码 at_send_command_full_nolock:运行在主线程里面.将一个 AT command 命令写入 modem 后进入睡眠状态（使用 pthread_cond_wait 或类似函数），直到 modem 读线程将其唤醒.唤醒后此函数获得了 AT command 的最终响应并返回.

函数 readerLoop 运行在一个单独的读线程里面，负责从 modem 中读取数据.读到的数据可分为三种类型：网络端传入的事件；modem 对当前 AT command 的部分响应；modem 对当前 AT command 的全部响应.对第三种类型的数据（AT command 的全部响应），读线程唤醒（pthread_cond_signal）睡眠状态的主线程.

2. Android 的消息处理机制

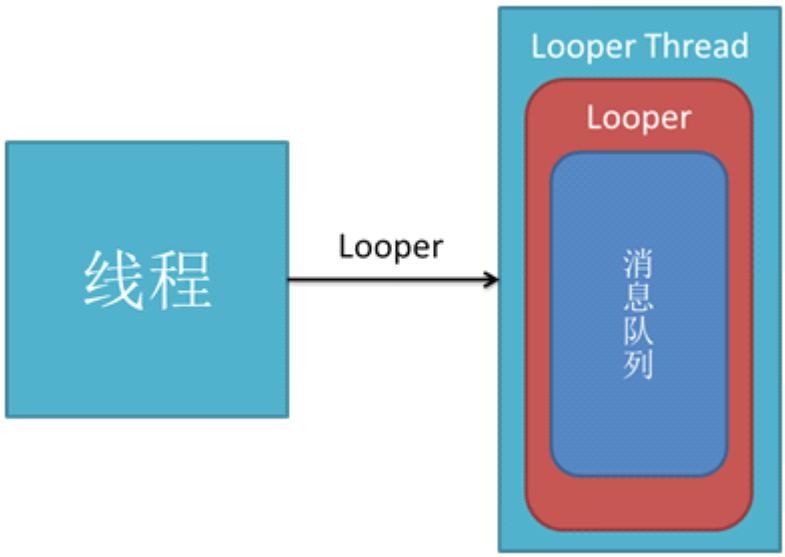
Looper 的字面意思是“循环者”，它被设计用来使一个普通线程变成 Looper 线程.所谓 Looper 线程就是循环工作的线程.在程序开发中（尤其是 GUI 开发中），我们经常会需要一个线程不断循环，一旦有新任务则执行，执行完继续等待下一个任务，这就是 Looper 线程.使用 Looper 类创建 Looper 线程很简单：

```
public class LooperThread extends Thread {
@Override
public void run（） {
// 将当前线程初始化为 Looper 线程
Looper.prepare（）;
```

```
// …其他处理，如实例化 handler
// 开始循环处理消息队列
Looper.loop（）;
}
}
```

通过上面两行核心代码，你的线程就升级为 Looper 线程了！！！是不是很神奇？让我们放慢镜头，看看这两行代码各自做了什么。

1) Looper.prepare()



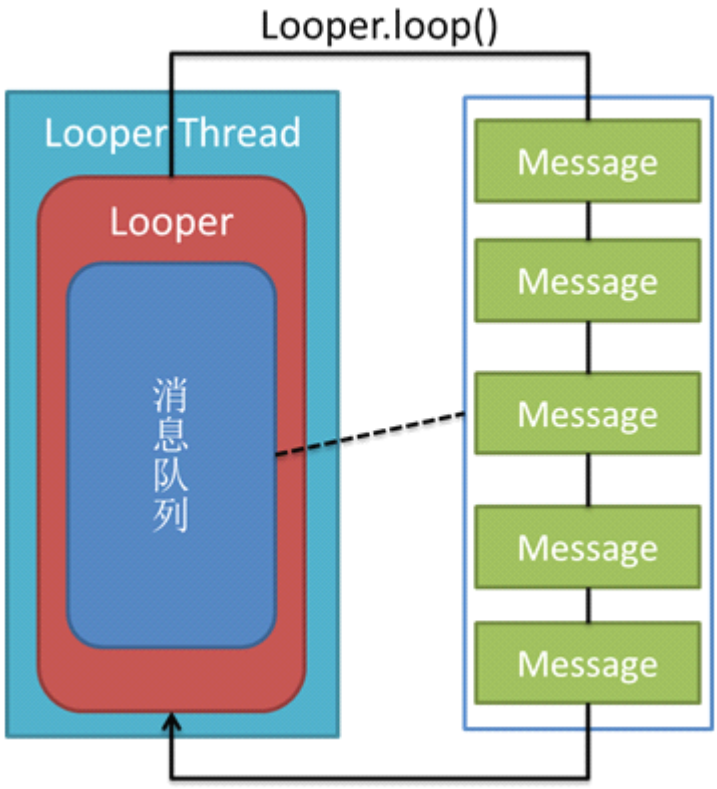
通过上图可以看到，现在你的线程中有一个 Looper 对象，它的内部维护了一个消息队列 MQ.注意，一个 Thread 只能有一个 Looper 对象，为什么呢？咱们来看源码.

```
public class Looper {
// 每个线程中的 Looper 对象其实是一个 ThreadLocal，即线程本地存储（TLS）对象
private static final ThreadLocal sThreadLocal = new ThreadLocal（）;
// Looper 内的消息队列
final MessageQueue mQueue;
// 当前线程
Thread mThread;
// ...其他属性
// 每个 Looper 对象中有它的消息队列，和它所属的线程
private Looper（） {
mQueue = new MessageQueue（）;
mRun = true;
mThread = Thread.currentThread（）;
}
```

```
// 我们调用该方法会在调用线程的 TLS 中创建 Looper 对象
public static final void prepare () {
if (sThreadLocal.get () != null) {
// 试图在有 Looper 的线程中再次创建 Looper 将抛出异常
throw new RuntimeException ("Only one Looper may be created per thread");
}
sThreadLocal.set (new Looper ());
}
// 其他方法
}
```

通过源码，prepare()背后的工作方式一目了然，其核心就是将 looper 对象定义为 ThreadLocal。如果你还不清楚什么是 ThreadLocal。

2）Looper.loop()



调用 loop 方法后，Looper 线程就开始真正工作了，它不断从自己的 MQ 中取出队头的消息（也叫任务）执行.其源码分析如下：

```
public static final void loop () {
Looper me = myLooper ();   //得到当前线程 Looper
MessageQueue queue = me.mQueue;   //得到当前 looper 的 MQ
// 这两行没看懂= = 不过不影响理解
```

```
Binder.clearCallingIdentity ();
final long ident = Binder.clearCallingIdentity ();
// 开始循环
while (true) {
Message msg = queue.next (); // 取出 message
if (msg != null) {
if (msg.target == null) {
// message 没有 target 为结束信号，退出循环
return;
}
// 日志...
if (me.mLogging != null) me.mLogging.println (
">>>>> Dispatching to " + msg.target + " "
+ msg.callback + ": " + msg.what
);
// 非常重要！将真正的处理工作交给 message 的 target，即后面要讲的 handler
msg.target.dispatchMessage (msg);
// 还是日志...
if (me.mLogging != null) me.mLogging.println (
"<<<<< Finished to " + msg.target + " "
+ msg.callback);
// 下面没看懂，同样不影响理解
final long newIdent = Binder.clearCallingIdentity ();
if (ident != newIdent) {
Log.wtf ("Looper", "Thread identity changed from 0x"
+ Long.toHexString (ident) + " to 0x"
+ Long.toHexString (newIdent) + " while dispatching to "
+ msg.target.getClass ().getName () + " "
+ msg.callback + " what=" + msg.what);
}
// 回收 message 资源
msg.recycle ();
}
}
}
```

除了 prepare（）和 loop（）方法，Looper 类还提供了一些有用的方法，比如 Looper.myLooper（）得到当前线程 looper 对象：

```
public static final Looper myLooper () {
// 在任意线程调用 Looper.myLooper () 返回的都是那个线程的 looper
return (Looper) sThreadLocal.get ();
}
public static final Looper myLooper () {
// 在任意线程调用 Looper.myLooper () 返回的都是那个线程的 looper
```

```
return （Looper）sThreadLocal.get（）;
}getThread（）得到 looper 对象所属线程：
public Thread getThread（） {
return mThread;
}quit（）方法结束 looper 循环：
public void quit（） {
// 创建一个空的 message，它的 target 为 NULL，表示结束循环消息
Message msg = Message.obtain（）;
// 发出消息
mQueue.enqueueMessage（msg， 0）;
}
public void quit（） {
// 创建一个空的 message，它的 target 为 NULL，表示结束循环消息
Message msg = Message.obtain（）;
// 发出消息
mQueue.enqueueMessage（msg， 0）;
}
```

到此为止，你应该对 Looper 有了基本的了解，总结几点：
1.每个线程有且最多只能有一个 Looper 对象，它是一个 ThreadLocal
2.Looper 内部有一个消息队列，loop（）方法调用后线程开始不断从队列中取出消息执行
3.Looper 使一个线程变成 Looper 线程.

3.Android 系统原理与源码分析

现在我们来看看第一个需求：如果某个应用需要弹出一个对话框.当单击“确定“按钮时完成某些工作，如果这些工作失败，对话框不能关闭.而当成功完成工作后，则关闭对话框.当然，无论何程度情况，单击“取消”按钮都会关闭对话框.

这个需求并不复杂，也并不过分（虽然我们可以自己弄个 Activity 来完成这个工作，也可在 View 上自己放按钮，但这显示有些大炮打蚊子了，如果对话框上只有一行文本，费这么多劲太不值了）.但使用过 AlertDialog 的读者都知道，无论单击的哪个按钮，无论按钮单击事件的执行情况如何，对话框是肯定要关闭的.也就是说，用户无法控制对话框的关闭动作.实际上，关闭对话框的动作已经在 Android SDK 写死了，并且未给使用者留有任何接口.但我的座右铭是“宇宙中没有什么是不能控制的”.

既然要控制对放框的关闭行为，首先就得分析是哪些类、哪些代码使这个对话框关闭的.进入 AlertDialog 类的源代码.在 AlertDialog 中只定义了一个变量：mAlert.这个变量是 AlertController 类型.AlertController 类是 Android 的内部类，在 com.android.internal.app 包中，无法通过普通的方式访问.也无法在 Eclipse 中通过按 Ctrl 键跟踪进源代码.但可以直接在 Android 源代码中找到 AlertController.java.我们再回到 AlertDialog 类中.AlertDialog 类实际上只是一个架子.象设置按钮、设置标题等工作都是由 AlertController 类完成的.因此，AlertController 类才是关键.

找到 AlertController.java 文件.打开后不要感到头晕哦，这个文件中的代码是很多地.不过这么多代码对本文的主题也没什么用处.下面就找一下控制按钮的代码.

在 AlertController 类的开头就会看到如下的代码：

```
View.OnClickListener mButtonHandler = new View.OnClickListener（） {
public void onClick（View v） {
Message m = null；
if （v == mButtonPositive && mButtonPositiveMessage ！ = null） {
m = Message.obtain（mButtonPositiveMessage）；
} else if （v == mButtonNegative && mButtonNegativeMessage ！ = null） {
m = Message.obtain（mButtonNegativeMessage）；
} else if （v == mButtonNeutral && mButtonNeutralMessage ！ = null） {
m = Message.obtain（mButtonNeutralMessage）；
}
if （m ！ = null） {
m.sendToTarget（）；
}
// Post a message so we dismiss after the above handlers are executed
mHandler.obtainMessage（ButtonHandler.MSG_DISMISS_DIALOG， mDialogInterface）
.sendToTarget（）；
}
}；
```

从这段代码中可以猜出来，前几行代码用来触发对话框中的三个按钮（Positive、Negative 和 Neutral）的单击事件，而最后的代码则用来关闭对话框（因为我们发现了 MSG_DISMISS_DIALOG、猜出来的）.

```
mHandler.obtainMessage（ButtonHandler.MSG_DISMISS_DIALOG， mDialogInterface）

.sendToTarget（）；
```

上面的代码并不是直接来关闭对话框的，而是通过一个 Handler 来处理，代码如下：

```
private static final class ButtonHandler extends Handler {
// Button clicks have Message.what as the BUTTON{1，2，3} constant
private static final int MSG_DISMISS_DIALOG = 1；
private WeakReference<DialogInterface> mDialog；
public ButtonHandler（DialogInterface dialog） {
mDialog = new WeakReference<DialogInterface>（dialog）；
}
@Override
public void handleMessage（Message msg） {
switch （msg.what） {
case DialogInterface.BUTTON_POSITIVE：
case DialogInterface.BUTTON_NEGATIVE：
case DialogInterface.BUTTON_NEUTRAL：
```

```
((DialogInterface.OnClickListener) msg.obj).onClick(mDialog.get(), msg.what);
break;
case MSG_DISMISS_DIALOG:
((DialogInterface) msg.obj).dismiss();
}
}
}
```

从上面代码的最后可以找到（（DialogInterface） msg.obj）.dismiss（）；.现在看了这么多源代码，我们来总结一下对话框按钮单击事件的处理过程.在 AlertController 处理对话框按钮时会为每一个按钮添加一个 onclick 事件.而这个事件类的对象实例就是上面的 mButtonHandler.在这个单击事件中首先会通过发送消息的方式调用为按钮设置的单击事件（也就是通过 setPositiveButton 等方法的第二个参数设置的单击事件），在触发完按钮的单击事件后，会通过发送消息的方式调用 dismiss 方法来关闭对话框.而在 AlertController 类中定义了一个全局的 mHandler 变量.在 AlertController 类中通过 ButtonHandler 类来对象来为 mHandler 赋值.因此，我们只要使用我们自己 Handler 对象替换 ButtonHandler 就可以阻止调用 dismiss 方法来关闭对话框.下面先在自己的程序中建立一个新的 ButtonHandler 类（也可叫其他的名）.

```
class ButtonHandler extends Handler
{
private WeakReference<DialogInterface> mDialog;
public ButtonHandler（DialogInterface dialog）
{
mDialog = new WeakReference<DialogInterface>（dialog）;
}
@Override
public
void handleMessage（Message msg）
{
switch （msg.what）
{
case DialogInterface.BUTTON_POSITIVE:
case DialogInterface.BUTTON_NEGATIVE:
case DialogInterface.BUTTON_NEUTRAL:
((DialogInterface.OnClickListener) msg.obj).onClick（mDialog
.get（）, msg.what）;
break;
}
}
}
```

我们可以看到，上面的类和 AlertController 中的 ButtonHandler 类很像，只是支掉了 switch 语句的最后一个 case 子句（用于调用 dismiss 方法）和相关的代码.

下面我们就要为 AlertController 中的 mHandler 重新赋值. 由于 mHandler 是 private 变量，因此，在这里需要使用 Java 的反射技术来为 mHandler 赋值. 由于在 AlertDialog 类中的 mAlert 变量同样也是 private，因此，也需要使用同样的反射技术来获得 mAlert 变量. 代码如下：

先建立一个 AlertDialog 对象

```
AlertDialog alertDialog = new AlertDialog.Builder（this）
.setTitle（"abc"）
.setMessage（"content"）
.setIcon（R.drawable.icon）
.setPositiveButton（ “确定”,
new OnClickListener（）
{
@Override
public void onClick（DialogInterface dialog，
int which）
{
}
}）.setNegativeButton（"取消"， new OnClickListener（）
{
@Override
public void onClick（DialogInterface dialog， int which）
{
dialog.dismiss（）;
}
}）.create（）;
```

上面的对话框很普通，单击哪个按钮都会关闭对话框. 下面在调用 show 方法之前来修改一个 mHandler 变量的值，OK，下面我们就来见证奇迹的时刻.

```
try{
Field field = alertDialog1.getClass（）.getDeclaredField（"mAlert"）;
field.setAccessible（true）;
//获得 mAlert 变量的值
Object obj = field.get（alertDialog1）;
field = obj.getClass（）.getDeclaredField（"mHandler"）;
field.setAccessible（true）;
//修改 mHandler 变量的值，使用新的 ButtonHandler 类
field.set（obj， new ButtonHandler（alertDialog1））;
}
catch （Exception e）
{
}
//显示对话框
```

```
alertDialog.show（）;
```

我们发现，如果加上 try

catch 语句，单击对话框中的确定按钮不会关闭对话框（除非在代码中调用 dismiss 方法），单击取消按钮则会关闭对话框（因为调用了 dismiss 方法）.如果去了 try…catch 代码段，对话框又会恢复正常了.

虽然上面的代码已经解决了问题，但需要编写的代码仍然比较多，为此，我们也可采用另外一种方法来阻止关闭对话框.这种方法不需要定义任何的类.

这种方法需要用点技巧.由于系统通过调用 dismiss 来关闭对话框，那么我们可以在 dismiss 方法上做点文章.在系统调用 dismiss 方法时会首先判断对话框是否已经关闭，如果对话框已经关闭了，就会退出 dismiss 方法而不再继续关闭对话框了.因此，我们可以欺骗一下系统，当调用 dismiss 方法时我们可以让系统以为对话框已经关闭（虽然对话框还没有关闭），这样 dismiss 方法就失效了，这样即使系统调用了 dismiss 方法也无法关闭对话框了.

下面让我们回到 AlertDialog 的源代码中，再继续跟踪到 AlertDialog 的父类 Dialog 的源代码中.找到 dismissDialog 方法.实际上，dismiss 方法是通过 dismissDialog 方法来关闭对话框的，dismissDialog 方法的代码如下：

```
Private void dismissDialog（） {
if （mDecor == null） {
if （Config.LOGV） Log.v（LOG_TAG，
"[Dialog] dismiss： already dismissed， ignore"）;
return;
}
if （！ mShowing） {
if （Config.LOGV） Log.v（LOG_TAG，
"[Dialog] dismiss： not showing， ignore"）;
return;
}
mWindowManager.removeView（mDecor）;
mDecor = null;
mWindow.closeAllPanels（）;
onStop（）;
mShowing = false;
sendDismissMessage（）;
}
```

该方法后面的代码不用管它，先看 if（！ mShowing）{…}这段代码.这个 mShowing 变量就是判断对话框是否已关闭的.因此，我们在代码中通过设置这个变量就可以使系统认为对话框已经关闭，就不再继续关闭对话框了.由于 mShowing 也是 private 变量，因此，也需要反射技术来设置这个变量.我们可以在对话框按钮的单击事件中设置 mShowing，代码如下：

```
try
```

```
{
Field field = dialog.getClass（）
.getSuperclass（）.getDeclaredField（
"mShowing"）；
field.setAccessible（true）；
//
将 mShowing 变量设为 false，表示对话框已关闭
field.set（dialog， false）；
dialog.dismiss（）；
}
catch （Exception e）{
```

4.关于 Android 技术的图片浏览源码分析

Android 手机操作系统的应用方式灵活，简单，深受广大编程爱好者的喜爱.尤其是它的开源代码，使得我们能够方便的得到自己想要的功能需求.今天我们就为大家带来了有关 Android 图片浏览的相关方法.

首先是 Android 图片浏览中 layout.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">
< ImageSwitcher Android： id="@+id/switcher"
Android：layout_width="fill_parent"
Android：layout_height="fill_parent"
Android：layout_alignParentTop="true"
Android：layout_alignParentLeft="true" />
< Gallery Android： id="@+id/gallery"
Android：background="#55000000"
Android：layout_width="fill_parent"
Android：layout_height="60dp"
Android：layout_alignParentBottom="true"
Android：layout_alignParentLeft="true"
Android：gravity="center_vertical"
Android：spacing="16dp" />
< /RelativeLayout>
```

layout 里面用到了前面所说的两个控件，ImageSwitcher 用啦显示全图，Gallery 用来显示缩略图。着重看看 ImageSwitcher,在 ImageSwitcher1 中需要实现 ViewSwitcher.ViewFactory 这个接口，这个接口里有个方法 makeView，这样就产生了用来显示图片的 view. ImageSwitcher 调用过程是这样的，首先要有一个 Factory 为它提供一个 View，然后 ImageSwitcher 就可以初始化各种资源了。注意在使用一个 ImageSwitcher 之前，一定要调用 setFactory 方法，要不 setImageResource 这个方法会报空指针异常。

下面是 Android 图片浏览代码：

```
import Android.app.Activity;
import Android.content.Context;
import Android.os.Bundle;
import Android.view.View;
import Android.view.ViewGroup;
import Android.view.animation.AnimationUtils;
import Android.widget.AdapterView;
import Android.widget.BaseAdapter;
import Android.widget.Gallery;
import Android.widget.ImageSwitcher;
import Android.widget.ImageView;
import Android.widget.ViewSwitcher;
import Android.widget.Gallery.LayoutParams;
public class ImageSwitcherTest extends Activity implements
AdapterView.OnItemSelectedListener, ViewSwitcher.ViewFactory{
private ImageSwitcher mSwitcher;
private Integer[] mThumbIds = {
R.drawable.sample_thumb_0, R.drawable.sample_thumb_1,
R.drawable.sample_thumb_2, R.drawable.sample_thumb_3,
R.drawable.sample_thumb_4, R.drawable.sample_thumb_5,
R.drawable.sample_thumb_6, R.drawable.sample_thumb_7};
private Integer[] mImageIds = {
R.drawable.sample_0, R.drawable.sample_1, R.drawable.sample_2,
R.drawable.sample_3, R.drawable.sample_4, R.drawable.sample_5,
R.drawable.sample_6, R.drawable.sample_7};
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
mSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
mSwitcher.setFactory(this);
mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this, Android.R.anim.fade_in));
mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,  Android.R.anim.fade_out));
Gallery g = (Gallery) findViewById(R.id.gallery);
g.setAdapter(new ImageAdapter(this));
g.setOnItemSelectedListener(this);
}

/*
* override for ViewSwitcher.ViewFactory#makeView()
*/
public View makeView() {
```

```
ImageView i = new ImageView(this);
i.setBackgroundColor(0xFF000000);
i.setScaleType(ImageView.ScaleType.FIT_CENTER);
i.setLayoutParams(new ImageSwitcher.LayoutParams(LayoutParams.FILL_PARENT,  LayoutParams.FILL_PARENT))
;
return i;
}

/*
* override for
* AdapterView.OnItemSelectedListener#onItemSelected()
*/

public void onItemSelected(AdapterView parent, View v, int position, long id) {
mSwitcher.setImageResource(mImageIds[position]);
}

/*
* override for AdapterView.OnItemSelectedListener #onNothingSelected()
*/

public void onNothingSelected(AdapterView<?> arg0) {
// TODO Auto-generated method stub
}
public class ImageAdapter extends BaseAdapter {
public ImageAdapter(Context c) {
mContext = c;
}
public int getCount() {
return mThumbIds.length;
}
public Object getItem(int position) {
return position;
}
public long getItemId(int position) {
return position;
}
public View getView(int position, View convertView, ViewGroup parent) {
ImageView i = new ImageView(mContext);
i.setImageResource(mThumbIds[position]);
i.setAdjustViewBounds(true);
i.setLayoutParams(new Gallery.LayoutParams(
LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
i.setBackgroundResource(R.drawable.picture_frame);
```

```
return i;
}
private Context mContext;
}
}
```

从 Android 图片浏览的代码中看到还实现了 AdapterView.OnItemSelectedListener，这样就需要重写 onItemSelected()方法，然后在该方法中：mSwitcher.setImageResource(mImageIds[position]);这样就实现了图片在 ImageSwitcher 中的切换。

5.Android系统原理与源码分析

在android里面用的smack包其实叫做asmack，该包提供了两种不同的连接方式：socket和httpclient.该并且提供了很多操作xmpp协议的API，也方便各种不同自定义协议的扩展.我们不需要自己重新去定义一套接收机制来扩展新的协议，只需继承然后在类里处理自己的协议就可以了.而本文今天主要说两点，一点就是消息是如何接收的，另一点就是消息是如何通知事件的.

总的思路

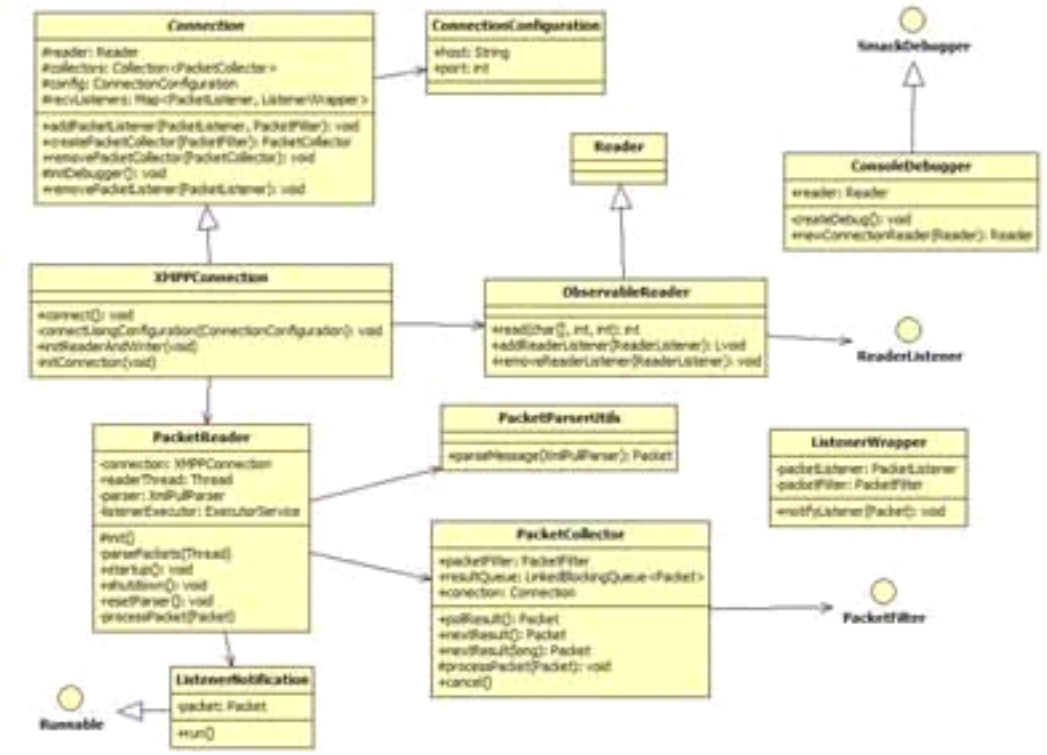
1.使用socket连接服务器

2.将XmlPullParser的数据源关联到socket的InputStream

3.启动线程不断循环处理消息

4.将接收到的消息解析xml处理封装好成一个Packet包

5.将包广播给所有注册事件监听的类



解析这块东西打算从最初的调用开始作为入口，抽丝剥茧，逐步揭开.

1. PacketListener packetListener = new PacketListener（） {
@Override
public void processPacket（Packet packet） {
System.out
.println（"Activity----processPacket" + packet.toXML（））；
}
};
PacketFilter packetFilter = new PacketFilter（） {
@Override
public boolean accept（Packet packet） {
System.out.println（"Activity----accept" + packet.toXML（））；
return true;
}
};

解释：创建包的监听以及包的过滤，当有消息到时就会广播到所有注册的监听，当然前提是要通过packetFilter的过滤.

2. connection = new XMPPConnection（）；

XMPPConnection 在这构造函数里面主要配置 ip 地址和端口（super（new ConnectionConfiguration（"169.254.141.109"， 9991））；）

3. connection.addPacketListener（packetListener， packetFilter）；
connection.connect（）；

注册监听，开始初始化连接.

```
4.  public void connect（） {
// Stablishes the connection， readers and writers
connectUsingConfiguration（config）；
}
5.  private void connectUsingConfiguration（ConnectionConfiguration config） {
String host = config.getHost（）；
int port = config.getPort（）；
try {
this.socket = new Socket（host， port）；
} catch （UnknownHostException e） {
e.printStackTrace（）；
} catch （IOException e） {
e.printStackTrace（）；
}
initConnection（）；
}通过之前设置的 ip 和端口，建立 socket 对象
6.  protected void initDebugger（） {
Class<? > debuggerClass = null；
try {
debuggerClass = Class.forName（"com.simualteSmack.ConsoleDebugger"）；
Constructor<? > constructor = debuggerClass.getConstructor（
Connection.class， Writer.class， Reader.class）；
debugger = （SmackDebugger） constructor.newInstance（this， writer，
reader）；
reader = debugger.getReader（）；
} catch （ClassNotFoundException e1） {
// TODO Auto-generated catch block
e1.printStackTrace（）；
} catch （Exception e） {
throw new IllegalArgumentException（
"Can't initialize the configured debugger！"， e）；
}
}private void initReaderAndWriter（） {
try {
reader = new BufferedReader（new InputStreamReader（socket
.getInputStream（）， "UTF-8"））；
} catch （UnsupportedEncodingException e） {
// TODO Auto-generated catch block
e.printStackTrace（）；
} catch （IOException e） {
// TODO Auto-generated catch block
e.printStackTrace（）；
}
```

```
initDebugger（）；
}private void initConnection（） {
// Set the reader and writer instance variables
initReaderAndWriter（）；
packetReader = new PacketReader（this）；
addPacketListener（debugger.getReaderListener（）， null）；
// Start the packet reader. The startup（） method will block until we
// get an opening stream packet back from server.
packetReader.startup（）；
}
```

从三个方法可以看出，建立reader和writer的对象关联到socket的InputStream，实例化ConsoleDebugger，该类主要是打印出接收到的消息，给reader设置了一个消息的监听.接着建立PacketReader对象，并启动.PacketReader主要负责消息的处理和通知

7. public class PacketReader {

```
private ExecutorService listenerExecutor;
private boolean done;
private XMPPConnection connection;
private XmlPullParser parser;
private Thread readerThread;
protected PacketReader（final XMPPConnection connection） {
this.connection = connection;
this.init（）；
}
/**
* Initializes the reader in order to be used. The reader is initialized
* during the first connection and when reconnecting due to an abruptly
* disconnection.
*/
protected void init（） {
done = false;
readerThread = new Thread（） {
public void run（） {
parsePackets（this）；
}
};
readerThread.setName（"Smack Packet Reader "）；
readerThread.setDaemon（true）；
// create an executor to deliver incoming packets to listeners.
// we will use a single thread with an unbounded queue.
listenerExecutor = Executors
.newSingleThreadExecutor（new ThreadFactory（） {
@Override
public Thread newThread（Runnable r） {
```

```
Thread thread = new Thread（r，
"smack listener processor"）；
thread.setDaemon（true）；
return thread；
}
}）；
resetParser（）；
}
/**
* Starts the packet reader thread and returns once a connection to the
* server has been established. A connection will be attempted for a maximum
* of five seconds. An XMPPException will be thrown if the connection fails.
*
*/
public void startup（） {
readerThread.start（）；
}
/**
* Shuts the packet reader down.
*/
public void shutdown（） {
done = true；
// Shut down the listener executor.
listenerExecutor.shutdown（）；
}
private void resetParser（） {
try {
parser = XmlPullParserFactory.newInstance（）.newPullParser（）；
parser.setFeature（XmlPullParser.FEATURE_PROCESS_NAMESPACES， true）；
parser.setInput（connection.reader）；
} catch （XmlPullParserException xppe） {
xppe.printStackTrace（）；
}
}
/**
* Parse top-level packets in order to process them further.
*
* @param thread
*                the thread that is being used by the reader to parse incoming
*                packets.
*/
private void parsePackets（Thread thread） {
try {
int eventType = parser.getEventType（）；
```

```
do {
if （eventType == XmlPullParser.START_TAG） {
if （parser.getName（）.equals（"message"）） {
processPacket（PacketParserUtils.parseMessage（parser））；
}
System.out.println（"START_TAG"）；
} else if （eventType == XmlPullParser.END_TAG） {
System.out.println（"END_TAG"）；
}
eventType = parser.next（）；
} while （！ done && eventType ！ = XmlPullParser.END_DOCUMENT
&& thread == readerThread）；
} catch （Exception e） {
e.printStackTrace（）；
if （！ done） {
}
}
}
private void processPacket（Packet packet） {
if （packet == null） {
return;
}
// Loop through all collectors and notify the appropriate ones.
for （PacketCollector collector ： connection.getPacketCollectors（）） {
collector.processPacket（packet）；
}
// Deliver the incoming packet to listeners.
listenerExecutor.submit（new ListenerNotification（packet））；
}
/**
* A runnable to notify all listeners of a packet.
*/
private class ListenerNotification implements Runnable {
private Packet packet;
public ListenerNotification（Packet packet） {
this.packet = packet;
}
public void run（） {
for （ListenerWrapper listenerWrapper ： connection.recvListeners
.values（））） {
listenerWrapper.notifyListener（packet）；
}
}
}
```

```
}
```

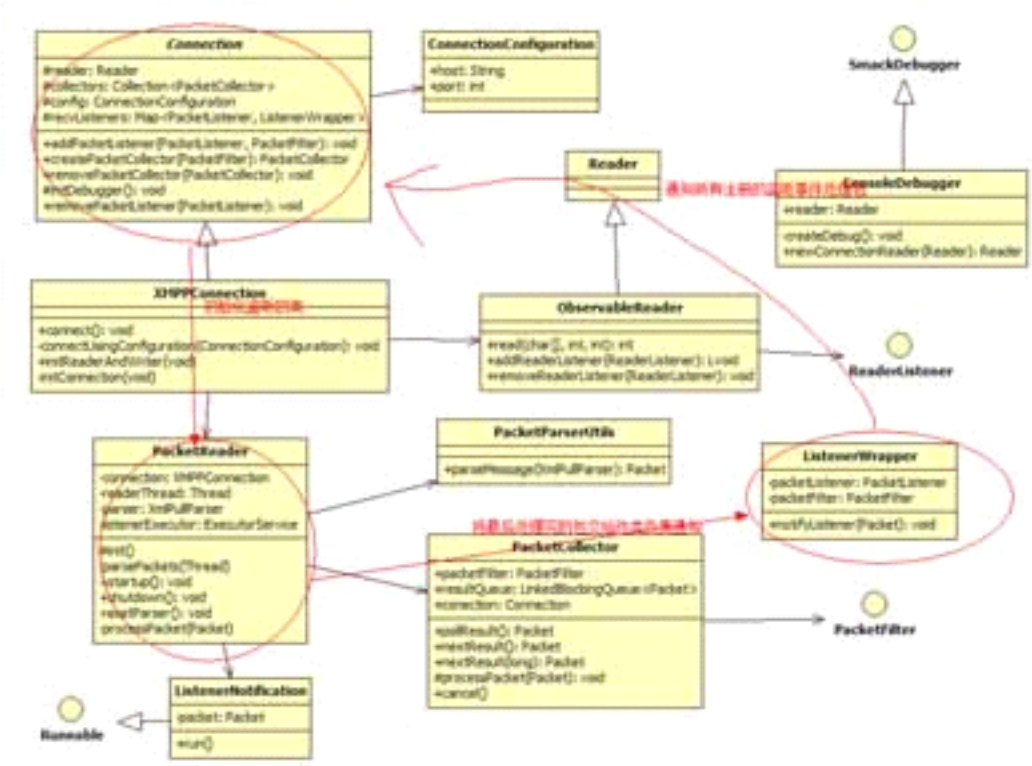
创建该类时就初始化线程和 ExecutorService ，接着调用 resetParser（） 方法为 parser 设置输入源（这里是重点，parser 的数据都是通过这里获取），调用 startup 启动线程，循环监听 parser，如果接收到消息根据消息协议的不同将调用 PacketParserUtils 类里的不同方法，这里调用 parseMessage（）该方法主要处理 message 的消息，在该方法里分析 message 消息并返回 packet 包.返回的包将调用 processPacket 方法，先通知所有注册了 PacketCollector 的监听，接着消息（listenerExecutor.submit（new ListenerNotification（packet））； ）传递给所有注册了 PacketListener 的监听.这样在 activity 开始之前注册的那个监听事件就会触发，从而完成了整个流程.

8.以上.剩下的就是一些辅助包，很简单.比如 PacketCollector 这个类，它的用处主要用来处理一些需要在发送后需要等待一个答复这样的请求.

```
protected synchronized void processPacket（Packet packet） {
System.out.println（"PacketCollector---processPacket"）；
if （packet == null） {
return;
}
if （packetFilter == null || packetFilter.accept（packet）） {
while （！ resultQueue.offer（packet）） {
resultQueue.poll（）；
}
}
}public Packet nextResult（long timeout） {
long endTime = System.currentTimeMillis（） + timeout；
System.out.println（"nextResult"）；
do {
try {
return resultQueue.poll（timeout， TimeUnit.MILLISECONDS）；
} catch （InterruptedException e） { /* ignore */
}
} while （System.currentTimeMillis（） < endTime）；
return null；
}
```

该方法就是将获取到的包，先过滤然后放到队列里，最后通过 nextResult 来获取包，这样就完成一个请求收一个答复.

这样整个流程就完成了，最后总结一下，如图：



6.Android IPC 通讯机制源码分析

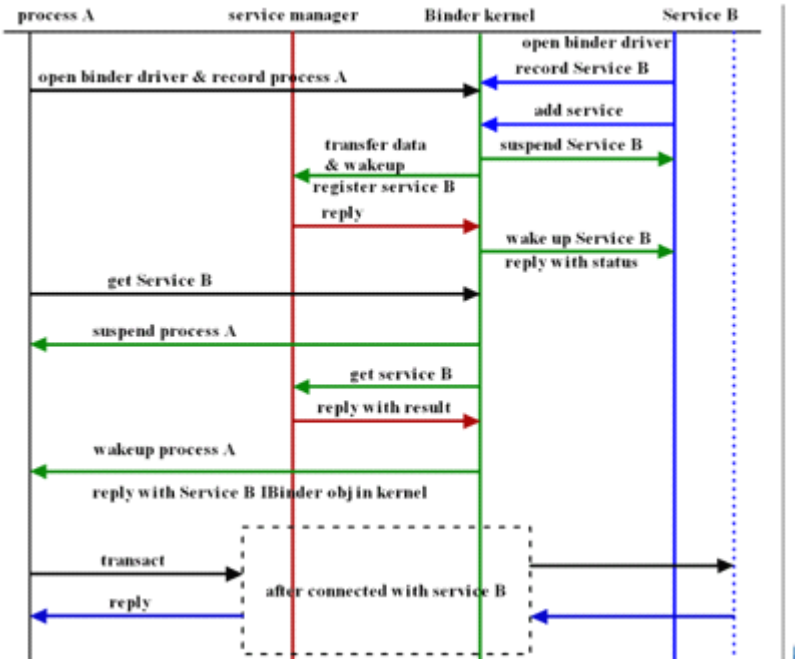
Linux 系统中进程间通信的方式有：socket， named pipe，message queque， signal，share memory。Java 系统中的进程间通信方式有 socket， named pipe 等，android 应用程序理所当然可以应用 JAVA 的 IPC 机制实现进程间的通信，但我查看 android 的源码，在同一终端上的应用软件的通信几乎看不到这些 IPC 通信方式，取而代之的是 Binder 通信。Google 为什么要采用这种方式呢，这取决于 Binder 通信方式的高效率。 Binder 通信是通过 linux 的 binder driver 来实现的，Binder 通信操作类似线程迁移（thread migration），两个进程间 IPC 看起来就象是一个进程进入另一个进程执行代码然后带着执行的结果返回。Binder 的用户空间为每一个进程维护着一个可用的线程池，线程池用于处理到来的 IPC 以及执行进程本地消息，Binder 通信是同步而不是异步。

Android 中的 Binder 通信是基于 Service 与 Client 的，所有需要 IBinder 通信的进程都必须创建一个 IBinder 接口，系统中有一个进程管理所有的 system service，Android 不允许用户添加非授权的 System service，当然现在源码开发了，我们可以修改一些代码来实现添加底层 system Service 的目的。对用户程序来说，我们也要创建 server，或者 Service 用于进程间通信，这里有一个 ActivityManagerService 管理 JAVA 应用层所有的 service 创建与连接（connect），disconnect，所有的 Activity 也是通过这个 service 来启动，加载的。ActivityManagerService 也是加载在 Systems Servcie 中的。

Android 虚拟机启动之前系统会先启动 service Manager 进程，service Manager 打开 binder 驱动，并通知 binder kernel 驱动程序这个进程将作为 System Service Manager，然后该进程将进入一个循环，等待处理来自其他进程的数据。用户创建一个 System service 后，通过 defaultServiceManager 得到一个远程 ServiceManager 的接口，通过这个接口我们可以调用 addService 函数将 System service 添加到 Service Manager 进程中，然后 client 可以通过 getService 获取到需要连接的目的 Service 的 IBinder 对象，这个 IBinder 是 Service 的 BBinder 在 binder kernel 的一个参考，所以 service IBinder 在 binder kernel 中不会存在相同的两个 IBinder 对象，每一个 Client 进程同样需要打开 Binder 驱动程序。对用户程序而言，我们获得这个对象就可以通过 binder kernel 访问 service 对象中的方

法。Client 与 Service 在不同的进程中，通过这种方式实现了类似线程间的迁移的通信方式，对用户程序而言当调用 Service 返回的 IBinder 接口后，访问 Service 中的方法就如同调用自己的函数。

下图为 client 与 Service 建立连接的示意图



首先从 ServiceManager 注册过程来逐步分析上述过程是如何实现的.

ServiceMananger 进程注册过程源码分析：

Service Manager Process（Service_manager.c）

Service_manager 为其他进程的 Service 提供管理，这个服务程序必须在 Android Runtime 起来之前运行，否则 Android JAVA Vm ActivityManagerService 无法注册.

```
int main（int argc， char **argv）
{
struct binder_state *bs;
void *svcmgr = BINDER_SERVICE_MANAGER;
bs = binder_open（128*1024）； //打开/dev/binder 驱动
if （binder_become_context_manager（bs）） {//注册为 service manager in binder kernel
LOGE（"cannot become context manager （%s）\n"， strerror（errno））；
return -1;
}
svcmgr_handle = svcmgr;
binder_loop（bs， svcmgr_handler）；
return 0;
}
```

首先打开 binder 的驱动程序然后通过 binder_become_context_manager 函数调用 ioctl 告诉 Binder Kernel 驱动程序这是一个服务管理进程，然后调用 binder_loop 等待来自其他进程的数据.BINDER_SERVICE_MANAGER 是服务管理进程的句柄，它的定义是：

```
/* the one magic object */
#define BINDER_SERVICE_MANAGER （（void*） 0）
```

如果客户端进程获取 Service 时所使用的句柄与此不符，Service Manager 将不接受 Client 的请求.客户端如何设置这个句柄在下面会介绍.

CameraSerivce 服务的注册（Main_mediaservice.c）

```
int main（int argc， char** argv）
{
sp<ProcessState> proc（ProcessState：：self（））；
sp<IServiceManager> sm = defaultServiceManager（）；
LOGI（"ServiceManager： %p"， sm.get（））；
AudioFlinger：：instantiate（）；                    //Audio 服务
MediaPlayerService：：instantiate（）；             //mediaPlayer 服务
CameraService：：instantiate（）；                   //Camera 服务
ProcessState：：self（）->startThreadPool（）；  //为进程开启缓冲池
IPCThreadState：：self（）->joinThreadPool（）；  //将进程加入到缓冲池
}
CameraService.cpp
void CameraService：：instantiate（） {
defaultServiceManager（）->addService（
String16（"media.camera"）， new CameraService（））；
}
```

创建 CameraService 服务对象并添加到 ServiceManager 进程中.

client 获取 remote IServiceManager IBinder 接口：

```
sp<IServiceManager> defaultServiceManager（）
{
if （gDefaultServiceManager ！= NULL） return gDefaultServiceManager；
{
AutoMutex _l（gDefaultServiceManagerLock）；
if （gDefaultServiceManager == NULL） {
gDefaultServiceManager = interface_cast<IServiceManager>（
ProcessState：：self（）->getContextObject（NULL））；
}
}
return gDefaultServiceManager；
}
```

任何一个进程在第一次调用 defaultServiceManager 的时候 gDefaultServiceManager 值为 Null，所以该进程会通过 ProcessState：：self 得到 ProcessState 实例.ProcessState 将打开 Binder 驱动.

```
ProcessState.cpp
sp<ProcessState> ProcessState：：self（）
{
if （gProcess ！= NULL） return gProcess；
AutoMutex _l（gProcessMutex）；
if （gProcess == NULL） gProcess = new ProcessState；
return gProcess；
}
ProcessState：：ProcessState（）
```

```
：  mDriverFD（open_driver（））  //打开/dev/binder 驱动
{
}
sp<IBinder> ProcessState：：getContextObject（const sp<IBinder>& caller）
{
if （supportsProcesses（））  {
return getStrongProxyForHandle（0）；
return getContextObject（String16（"default"），  caller）；
}
}
```

Android 是支持 Binder 驱动的所以程序会调用 getStrongProxyForHandle.这里 handle 为 0，正好与 Service_manager 中的 BINDER_SERVICE_MANAGER 一致.

```
sp<IBinder> ProcessState：：getStrongProxyForHandle（int32_t handle）
{
sp<IBinder> result；
AutoMutex _l（mLock）；
handle_entry* e = lookupHandleLocked（handle）；
if （e ！= NULL）  {
// We need to create a new BpBinder if there isn't currently one，  OR we
// are unable to acquire a weak reference on this current one. See comment
// in getWeakProxyForHandle（）  for more info about this.
IBinder* b = e->binder；  //第一次调用该函数 b 为 Null
if （b == NULL || ！e->refs->attemptIncWeak（this））  {
b = new BpBinder（handle）；
e->binder = b；
if （b）  e->refs = b->getWeakRefs（）；
result = b；
} else {
// This little bit of nastyness is to allow us to add a primary
// reference to the remote proxy when this team doesn't have one
// but another team is sending the handle to us.
result.force_set（b）；
e->refs->decWeak（this）；
}
}
return result；
}
```

第一次调用的时候 b 为 Null 所以会为 b 生成一 BpBinder 对象：

```
BpBinder：：BpBinder（int32_t handle）
：  mHandle（handle）
，  mAlive（1）
，  mObitsSent（0）
，  mObituaries（NULL）
{
```

```
LOGV（"Creating BpBinder %p handle %d\n"， this， mHandle）；
extendObjectLifetime（OBJECT_LIFETIME_WEAK）；
IPCThreadState：：self（）->incWeakHandle（handle）；
}
void IPCThreadState：：incWeakHandle（int32_t handle）
{
LOG_REMOTEREFS（"IPCThreadState：：incWeakHandle（%d）\n"， handle）；
mOut.writeInt32（BC_INCREFS）；
mOut.writeInt32（handle）；
}
```

getContextObject 返回了一个 BpBinder 对象.

```
interface_cast<IServiceManager>（
ProcessState：：self（）->getContextObject（NULL））；
template<typename INTERFACE>
inline sp<INTERFACE> interface_cast（const sp<IBinder>& obj）
{
return INTERFACE：：asInterface（obj）；
}
```

将这个宏扩展后最终得到的是：

```
sp<IServiceManager> IServiceManager：：asInterface（const sp<IBinder>& obj）
{
sp<IServiceManager> intr；
if （obj ！=NULL） {
intr = static_cast<IServiceManager*>（
obj->queryLocalInterface（
IServiceManager：：descriptor）.get（））；
if （intr == NULL） {
intr = new BpServiceManager（obj）；
}
}
return intr；
}
```

返回一个 BpServiceManager 对象，这里 obj 就是前面我们创建的 BpBInder 对象.

client 获取 Service 的远程 IBinder 接口

以 CameraService 为例（camera.cpp）：

```
const sp<ICameraService>& Camera：：getCameraService（）
{
Mutex：：Autolock _l（mLock）；
if （mCameraService.get（） ==0） {
sp<IServiceManager> sm = defaultServiceManager（）；
sp<IBinder> binder；
do {
binder = sm->getService（String16（"media.camera"））；
if （binder ！=0）
```

```
break;
LOGW（"CameraService not published， waiting…"）；
usleep（500000）； // 0.5 s
} while（true）；
if （mDeathNotifier == NULL） {
mDeathNotifier = new DeathNotifier（）；
}
binder->linkToDeath（mDeathNotifier）；
mCameraService = interface_cast<ICameraService>（binder）；
}
LOGE_IF（mCameraService==0， "no CameraService！？"）；
return mCameraService;
}
```

由前面的分析可知 sm 是 BpCameraService 对象：//应该为 BpServiceManager 对象

```
virtual sp<IBinder> getService（const String16& name） const
{
unsigned n;
for （n = 0； n < 5； n++）{
sp<IBinder> svc = checkService（name）；
if （svc ！= NULL） return svc;
LOGI（"Waiting for sevice %s…\n"， String8（name）.string（））；
sleep（1）；
}
return NULL;
}
virtual sp<IBinder> checkService（ const String16& name） const
{
Parcel data， reply;
data.writeInterfaceToken（IServiceManager：：getInterfaceDescriptor（））；
data.writeString16（name）；
remote（）->transact（CHECK_SERVICE_TRANSACTION， data， &reply）；
return reply.readStrongBinder（）；
}
```

这里的 remote 就是我们前面得到 BpBinder 对象.所以 checkService 将调用 BpBinder 中的 transact 函数：

```
status_t BpBinder：：transact（
uint32_t code， const Parcel& data， Parcel* reply， uint32_t flags）
{
// Once a binder has died， it will never come back to life.
if （mAlive） {
status_t status = IPCThreadState：：self（）->transact（
mHandle， code， data， reply， flags）；
if （status == DEAD_OBJECT） mAlive = 0;
return status;
}
```

return DEAD_OBJECT；

}

mHandle 为 0，BpBinder 继续往下调用 IPCThreadState：transact 函数将数据发给与 mHandle 相关联的 Service Manager Process.

```
status_t IPCThreadState：：transact（int32_t handle，
uint32_t code， const Parcel& data，
Parcel* reply， uint32_t flags）
{
if （err == NO_ERROR） {
LOG_ONEWAY（">>>> SEND from pid %d uid %d %s"， getpid（），getuid（），
（flags & TF_ONE_WAY） == 0 ？ "READ REPLY" ： "ONE WAY"）；
err = writeTransactionData（BC_TRANSACTION， flags， handle， code， data， NULL）；
}
if （err ！= NO_ERROR） {
if （err ！= NO_ERROR） {
if （reply） reply->setError（err）；
return （mLastError = err）；
}
if （（flags & TF_ONE_WAY） == 0） {
if （reply） {
err = waitForResponse（reply）；
} else {
Parcel fakeReply；
err = waitForResponse（&fakeReply）；
}
return err；
}
```

通过 writeTransactionData 构造要发送的数据

```
status_t IPCThreadState：：writeTransactionData（int32_t cmd， uint32_t binderFlags，
int32_t handle， uint32_t code， const Parcel& data， status_t* statusBuffer）
{
binder_transaction_data tr；
tr.target.handle = handle； //这个 handle 将传递到 service_manager
tr.code = code；
tr.flags = bindrFlags；
}
```

waitForResponse 将调用 talkWithDriver 与对 Binder kernel 进行读写操作.当 Binder kernel 接收到数据后，service_mananger 线程的 ThreadPool 就会启动，service_manager 查找到 CameraService 服务后调用 binder_send_reply，将返回的数据写入 Binder kernel，Binder kernel.

```
status_t IPCThreadState：：waitForResponse（Parcel *reply， status_t *acquireResult）
{
int32_t cmd；
int32_t err；
while （1） {
```

```
if （（err=talkWithDriver（））＜NO_ERROR） break；
}
status_t IPCThreadState：：talkWithDriver（bool doReceive）
{
#if defined（HAVE_ANDROID_OS）
if （ioctl（mProcess->mDriverFD， BINDER_WRITE_READ， &bwr） >= 0）
err = NO_ERROR；
else
err = -errno；
#else
err = INVALID_OPERATION；
#endif
}
```

通过上面的 ioctl 系统函数中 BINDER_WRITE_READ 对 binder kernel 进行读写.

7.Android源码分析：HeaderViewListAdapter

在Android开发的时候经常会有一些列表数据分组显示的需求，对于iPhone来说修改下模型数据并设置下风格就可以轻松搞定。但对于Android这种需求就会有些痛苦了。因为系统提供的ListAdapter无法提供这种支持，因此只能自定义Adapter。为了实现该功能，Adapter通常就开始充斥很多特殊处理以及各种硬编码。如何用Android的方式设计和实现一个自定义Adapter呢？源码中的HeaderViewListAdapter可能会提供些答案或者启发。

```
/*如果 List 需要有标题就会使用该 ListAdapter。该 ListAdapter 包装了另一个 ListAdapter 并保存着
标题视图和相关数据对象。该类是作为基类，在代码中可能不需要直接使用。
*/
// 类名有点不确切，其实是支持 Header 和 Footter 的
public class HeaderViewListAdapter implements WrapperListAdapter, Filterable {

    private final ListAdapter mAdapter;

    // These two ArrayList are assumed to NOT be null.
    // They are indeed created when declared in ListView and then shared.
    ArrayList<ListView.FixedViewInfo> mHeaderViewInfos;
    ArrayList<ListView.FixedViewInfo> mFooterViewInfos;

    // Used as a placeholder in case the provided info views are indeed null.
    // Currently only used by some CTS tests, which may be removed.
    static final ArrayList<ListView.FixedViewInfo> EMPTY_INFO_LIST =
        new ArrayList<ListView.FixedViewInfo>();

    boolean mAreAllFixedViewsSelectable;
```

```
/// 这是个很有趣的概念，之后我们会一起了解
private final boolean mIsFilterable;

public HeaderViewListAdapter(ArrayList<ListView.FixedViewInfo> headerViewInfos,
                             ArrayList<ListView.FixedViewInfo> footerViewInfos,
                             ListAdapter adapter) {
    mAdapter = adapter;
    mIsFilterable = adapter instanceof Filterable;
    /// 很好的处理，这里实际是使用了一种 Null 对象模式
    if (headerViewInfos == null) {
        mHeaderViewInfos = EMPTY_INFO_LIST;
    } else {
        mHeaderViewInfos = headerViewInfos;
    }
    if (footerViewInfos == null) {
        mFooterViewInfos = EMPTY_INFO_LIST;
    } else {
        mFooterViewInfos = footerViewInfos;
    }

    mAreAllFixedViewsSelectable =
            areAllListInfosSelectable(mHeaderViewInfos)
            && areAllListInfosSelectable(mFooterViewInfos);
}



/// 支持多个标题栏哦
public int getHeadersCount() {
    return mHeaderViewInfos.size();
}


/// 支持多个尾注栏哦
public int getFootersCount() {
    return mFooterViewInfos.size();
}

public boolean isEmpty() {
    return mAdapter == null || mAdapter.isEmpty();
}


/// FixedViewInfo 名字很贴切，这是相对于包装 Adapter 中的可变数据命名的
```

```
private boolean areAllListInfosSelectable(ArrayList<ListView.FixedViewInfo> infos) {
    if (infos != null) {
        for (ListView.FixedViewInfo info : infos) {
            if (!info.isSelectable) {
                return false;
            }
        }
    }
    return true;
}


/// 支持标题删减
public boolean removeHeader(View v) {
    for (int i = 0; i < mHeaderViewInfos.size(); i++) {
        ListView.FixedViewInfo info = mHeaderViewInfos.get(i);
        if (info.view == v) {
            mHeaderViewInfos.remove(i);
            mAreAllFixedViewsSelectable =
                    areAllListInfosSelectable(mHeaderViewInfos)
                    && areAllListInfosSelectable(mFooterViewInfos);
            return true;
        }
    }

    return false;
}


/// 支持尾注删减
public boolean removeFooter(View v) {
    for (int i = 0; i &lt; mFooterViewInfos.size(); i++) {
        ListView.FixedViewInfo info = mFooterViewInfos.get(i);
        if (info.view == v) {
            mFooterViewInfos.remove(i);

            mAreAllFixedViewsSelectable =
                    areAllListInfosSelectable(mHeaderViewInfos)
                    && areAllListInfosSelectable(mFooterViewInfos);

            return true;
        }
    }
```

```
        return false;
    }


    /// 标题视图、尾注视图和普通的数据视图，一视同仁
    public int getCount() {
        if (mAdapter != null) {
            return getFootersCount() + getHeadersCount() + mAdapter.getCount();
        } else {
            return getFootersCount() + getHeadersCount();
        }
    }

    public boolean areAllItemsEnabled() {
        if (mAdapter != null) {
            return mAreAllFixedViewsSelectable && mAdapter.areAllItemsEnabled();
        } else {
            return true;
        }
    }

    public boolean isEnabled(int position) {
        // Header (negative positions will throw an ArrayIndexOutOfBoundsException)
        int numHeaders = getHeadersCount();
        if (position &lt; numHeaders) {
            return mHeaderViewInfos.get(position).isSelectable;
        }

        // Adapter
        final int adjPosition = position - numHeaders;
        int adapterCount = 0;
        if (mAdapter != null) {
            adapterCount = mAdapter.getCount();
            if (adjPosition &lt; adapterCount) {
                return mAdapter.isEnabled(adjPosition);
            }
        }

        // Footer (off-limits positions will throw an ArrayIndexOutOfBoundsException)
        return mFooterViewInfos.get(adjPosition - adapterCount).isSelectable;
    }

    /// 需要分段处理
    public Object getItem(int position) {
```

```
        // Header (negative positions will throw an ArrayIndexOutOfBoundsException)
        int numHeaders = getHeadersCount();
        if (position &amp;lt; numHeaders) {
            return mHeaderViewInfos.get(position).data;
        }

        // Adapter
        final int adjPosition = position - numHeaders;
        int adapterCount = 0;
        if (mAdapter != null) {
            adapterCount = mAdapter.getCount();
            if (adjPosition &amp;lt; adapterCount) {
                return mAdapter.getItem(adjPosition);
            }
        }

        // Footer (off-limits positions will throw an ArrayIndexOutOfBoundsException)
        return mFooterViewInfos.get(adjPosition - adapterCount).data;
    }

    /// 这可不包括 Header 和 Footer
    public long getItemId(int position) {
        int numHeaders = getHeadersCount();
        if (mAdapter != null && position >= numHeaders) {
            int adjPosition = position - numHeaders;
            int adapterCount = mAdapter.getCount();
            if (adjPosition < adapterCount) {
                return mAdapter.getItemId(adjPosition);
            }
        }
        return -1;
    }

    public boolean hasStableIds() {
        if (mAdapter != null) {
            return mAdapter.hasStableIds();
        }
        return false;
    }

    /// 需要分段处理，其中 Header 和 Footer 视图已经事先存储在 ListView.FixedViewInfo 对象中了
    public View getView(int position, View convertView, ViewGroup parent) {
        // Header (negative positions will throw an ArrayIndexOutOfBoundsException)
        int numHeaders = getHeadersCount();
```

```
        if (position &lt; numHeaders) {
            return mHeaderViewInfos.get(position).view;
        }

        // Adapter
        final int adjPosition = position - numHeaders;
        int adapterCount = 0;
        if (mAdapter != null) {
            adapterCount = mAdapter.getCount();
            if (adjPosition &lt; adapterCount) {
                return mAdapter.getView(adjPosition, convertView, parent);
            }
        }

        // Footer (off-limits positions will throw an ArrayIndexOutOfBoundsException)
        return mFooterViewInfos.get(adjPosition - adapterCount).view;
    }

    /// 支持同一个 List 中不同类型的视图
    public int getItemViewType(int position) {
        int numHeaders = getHeadersCount();
        if (mAdapter != null && position >= numHeaders) {
            int adjPosition = position - numHeaders;
            int adapterCount = mAdapter.getCount();
            if (adjPosition &lt; adapterCount) {
                return mAdapter.getItemViewType(adjPosition);
            }
        }

        return AdapterView.ITEM_VIEW_TYPE_HEADER_OR_FOOTER;
    }

    public int getViewTypeCount() {
        if (mAdapter != null) {
            return mAdapter.getViewTypeCount();
        }
        return 1;
    }

    public void registerDataSetObserver(DataSetObserver observer) {
        if (mAdapter != null) {
            mAdapter.registerDataSetObserver(observer);
        }
    }
```

```
    public void unregisterDataSetObserver(DataSetObserver observer) {
        if (mAdapter != null) {
            mAdapter.unregisterDataSetObserver(observer);
        }
    }

    /// 这有点意思，有待发掘
    public Filter getFilter() {
        if (mIsFilterable) {
            return ((Filterable) mAdapter).getFilter();
        }
        return null;
    }
    public ListAdapter getWrappedAdapter() {
        return mAdapter;
    }
}
```

8.Android移植：wifi设计原理(源码分析)

初始化在 SystemServer 启动的时辰，会生成一个 ConnectivityService 的实例，

```
try {
Log.i（TAG， "Starting Connectivity Service."）;
ServiceManager.addService（Context.CONNECTIVITY_SERVICE， new
ConnectivityService（context））;
} catch （Throwable e） {
Log.e（TAG， "Failure starting Connectivity Service"， e）;
}
```

ConnectivityService 的结构函数会创立 WifiService，

```
if （DBG） Log.v（TAG， "Starting Wifi Service."）;
mWifiStateTracker = new WifiStateTracker（context， handler）;
WifiService wifiService = new WifiService（context， mWifiStateTracker）;
ServiceManager.addService（Context.WIFI_SERVICE， wifiService）;
```

WifiStateTracker 会创立 WifiMonitor 吸收来自底层的事务，WifiService 和 WifiMonitor 是整个模块的焦点.WifiService 负责启动封锁 wpa_supplicant、启动封锁 WifiMonitor 监督线程和把呼吁下发给 wpa_supplicant，而 WifiMonitor 则负责从 wpa_supplicant 吸收事务通知.

毗连 AP

1. 使能 WIFI

WirelessSettings 在初始化的时辰设置装备摆设了由 WifiEnabler 来措置赏罚 Wifi 按钮，

```
private void initToggles（） {
mWifiEnabler = new WifiEnabler（
this，
（WifiManager） getSystemService（WIFI_SERVICE），
```

（CheckBoxPreference） findPreference（KEY_TOGGLE_WIFI））；

当用户按下 Wifi 按钮后， Android 会挪用 WifiEnabler 的 onPreferenceChange， 再由 WifiEnabler 挪用 WifiManager 的 setWifiEnabled 接口函数，经由过程 AIDL，现实挪用的是 WifiService 的 setWifiEnabled 函数，WifiService 接着向自身发送一条 MESSAGE_ENABLE_WIFI 动静，在 措置赏罚该动静的代码中做真正的使能工作：首先装载 WIFI 内核模块（该模块的地位硬编码为 "/system/lib/modules/wlan.ko" ）， 然 后 启 动 wpa_supplicant （ 配 置 文 件 硬 编 码 为 "/data/misc/wifi/wpa_supplicant.conf"） 再经由过程 WifiStateTracker 来启动 WifiMonitor 中的监督 ，

线程.

```
private boolean setWifiEnabledBlocking（boolean enable） {
final int eventualWifiState = enable ? WIFI_STATE_ENABLED :
WIFI_STATE_DISABLED;
updateWifiState（enable ? WIFI_STATE_ENABLING : WIFI_STATE_DISABLING);
if （enable） {
if （! WifiNative.loadDriver（）） {
Log.e（TAG， "Failed to load Wi-Fi driver."）;
updateWifiState（WIFI_STATE_UNKNOWN）;
return false;
}
if （! WifiNative.startSupplicant（）） {
WifiNative.unloadDriver（）;
Log.e（TAG， "Failed to start supplicant daemon."）;
updateWifiState（WIFI_STATE_UNKNOWN）;
return false;
}
mWifiStateTracker.startEventLoop（）;
}
// Success!
persistWifiEnabled（enable）;
updateWifiState（eventualWifiState）;
return true;
}
```

当使能成功后，会广播发送 WIFI_STATE_CHANGED_ACTION 这个 Intent 通知外界 WIFI 已 经 成 功 使 能 了 .WifiEnabler 创 建 的 时 候 就 会 向 Android 注 册 接 收 WIFI_STATE_CHANGED_ACTION，是以它会收到该 Intent，从而初步扫描.

```
private void handleWifiStateChanged（int wifiState） {
if （wifiState == WIFI_STATE_ENABLED） {
loadConfiguredAccessPoints（）;
attemptScan（）;
}
```

2. 查找 AP

扫描的进口函数是 WifiService 的 startScan，它其实也就是往 wpa_supplicant 发送 SCAN 命令.

```
static jboolean android_net_wifi_scanCommand（JNIEnv* env， jobject clazz）
{
```

```
jboolean result;
// Ignore any error from setting the scan mode.
// The scan will still work.
（void）doBooleanCommand（"DRIVER SCAN-ACTIVE"， "OK"）;
result = doBooleanCommand（"SCAN"， "OK"）;
（void）doBooleanCommand（"DRIVER SCAN-PASSIVE"， "OK"）;
return result;
}
```

当 wpa_supplicant 措置赏罚完 SCAN 呼吁后，它会向独霸通道发送事务通知扫描完成，从而 wifi_wait_for_event 函数会吸收到该事务，由此 WifiMonitor 中的 MonitorThread 会被履行来出来这个事务，

```
void handleEvent（int event， String remainder） {
case SCAN_RESULTS:
mWifiStateTracker.notifyScanResultsAvailable（）;
break;
```

WifiStateTracker 则接着广播发送 SCAN_RESULTS_AVAILABLE_ACTION 这个 Intent

```
case EVENT_SCAN_RESULTS_AVAILABLE:
mContext.sendBroadcast（new
Intent（WifiManager.SCAN_RESULTS_AVAILABLE_ACTION））;
```

WifiLayer 注册了吸收 SCAN_RESULTS_AVAILABLE_ACTION 这个 Intent，所以它的相干措置赏罚函数 handleScanResultsAvailable 会被挪用，在该函数中，先会往拿到 SCAN 的功效（最终是往 wpa_supplicant 发送 SCAN_RESULT 呼吁并读取返回值来实现的），

List<ScanResult> list = mWifiManager.getScanResults（）;

对每一个扫描返回的 AP，WifiLayer 会挪用 WifiSettings 的 onAccessPointSetChanged 函数，从而最终把该 AP 加到 GUI 显示列表中.

```
public void onAccessPointSetChanged（AccessPointState ap， boolean added） {
AccessPointPreference pref = mAps.get（ap）;
if （added） {
if （pref == null） {
pref = new AccessPointPreference（this， ap）;
mAps.put（ap， pref）;
} else {
pref.setEnabled（true）;
}
mApCategory.addPreference（pref）;
}
}
```

3. 设置装备摆设 AP 参数

当用户在 WifiSettings 界面上选择了一个 AP 后，会显示设置装备摆设 AP 参数的一个对话框，

```
public boolean onPreferenceTreeClick（PreferenceScreen preferenceScreen， Preference
preference） {
if （preference instanceof AccessPointPreference） {
AccessPointState state = （（AccessPointPreference）
preference）.getAccessPointState（）;
```

```
showAccessPointDialog（state， AccessPointDialog.MODE_INFO）；
}
}
```

4. 毗连

当用户在 AcessPointDialog 中选择好加密方法和输进密钥之后，再点击毗连按钮，Android 就会往毗连这个 AP.

```
private void handleConnect（） {
String password = getEnteredPassword（）；
if （！ TextUtils.isEmpty（password）） {
mState.setPassword（password）；
}
mWifiLayer.connectToNetwork（mState）；
}
```

WifiLayer 会先检测这个 AP 是不是之前被设置装备摆设过，这个是经由过程向 wpa_supplicant 发送 LIST_NETWORK 呼吁而且斗劲返回值来实现的，

```
// Need WifiConfiguration for the AP
WifiConfiguration config = findConfiguredNetwork（state）；
```

假如 wpa_supplicant 没有这个 AP 的设置装备摆设信息， 则会向 wpa_supplicant 发送 ADD_NETWORK 呼吁来添加该 AP，

```
if （config == null） {
// Connecting for the first time， need to create it
config = addConfiguration（state，
ADD_CONFIGURATION_ENABLE|ADD_CONFIGURATION_SAVE）；
}
```

ADD_NETWORK 命令会返回一个 ID， WifiLayer 再用这个返回的 ID 作为参数向 wpa_supplicant 发送 ENABLE_NETWORK 呼吁，从而让 wpa_supplicant 往毗连该 AP.

```
// Make sure that network is enabled， and disable others
mReenableApsOnNetworkStateChange = true；
if （！ mWifiManager.enableNetwork（state.networkId， true）） {
Log.e（TAG， "Could not enable network ID " + state.networkId）；
error（R.string.error_connecting）；
return false；
}
```

5. 设置装备摆设 IP 地址

当 wpa_supplicant 成功毗连上 AP 之后，它会向独霸通道发送事务通知毗连上 AP 了，从而 wifi_wait_for_event 函数会吸收到该事务，由此 WifiMonitor 中的 MonitorThread 会被履行来出来这个事务，

```
void handleEvent（int event， String remainder） {
case CONNECTED：
handleNetworkStateChange（NetworkInfo.DetailedState.CONNECTED，
remainder）；
break；
```

WifiMonitor 再挪用 WifiStateTracker 的 notifyStateChange，WifiStateTracker 则接着会往自身发送 EVENT_DHCP_START 动静来启动 DHCP 往获取 IP 地址，

```
private void handleConnectedState（） {
setPollTimer（）;
mLastSignalLevel = -1;
if （！ mHaveIPAddress && ！ mObtainingIPAddress） {
mObtainingIPAddress = true;
mDhcpTarget.obtainMessage（EVENT_DHCP_START）.sendToTarget（）;
}
}
```

然后再广播发送 NETWORK_STATE_CHANGED_ACTION 这个 Intent

```
case EVENT_NETWORK_STATE_CHANGED:
if （result.state ！= DetailedState.DISCONNECTED || ！ mDisconnectPending） {
intent = new
Intent（WifiManager.NETWORK_STATE_CHANGED_ACTION）;
intent.putExtra（WifiManager.EXTRA_NETWORK_INFO，
mNetworkInfo）;
if （result.BSSID ！= null）
intent.putExtra（WifiManager.EXTRA_BSSID， result.BSSID）;
mContext.sendStickyBroadcast（intent）;
}
break;
```

WifiLayer 注册了吸收 NETWORK_STATE_CHANGED_ACTION 这个 Intent，所以它的相干措置赏罚函数 handleNetworkStateChanged 会被挪用，当 DHCP 拿到 IP 地址之后，会再发送 EVENT_DHCP_SUCCEEDED 动静，

```
private class DhcpHandler extends Handler {
public void handleMessage（Message msg） {
switch （msg.what） {
case EVENT_DHCP_START:
if （NetworkUtils.runDhcp（mInterfaceName， mDhcpInfo））{
event = EVENT_DHCP_SUCCEEDED;
}
```

WifiLayer 处理 EVENT_DHCP_SUCCEEDED 消息，会再次广播发送 NETWORK_STATE_CHANGED_ACTION 这个 Intent，此次带上完整的 IP 地址信息.

```
case EVENT_DHCP_SUCCEEDED:
mWifiInfo.setIpAddress（mDhcpInfo.ipAddress）;
setDetailedState（DetailedState.CONNECTED）;
intent = new
Intent（WifiManager.NETWORK_STATE_CHANGED_ACTION）;
intent.putExtra（WifiManager.EXTRA_NETWORK_INFO， mNetworkInfo）;
mContext.sendStickyBroadcast（intent）;
break;
```

9. Android Prelink 实现的源码分析

1. 原理简介

1） Prelink

Prelink 即预链接技术是利用事先链接以代替运行时链接的技术，以加快共享库的加载速度，它不仅能加快程序启动时间，还可以减少部分内存开销（它能使 KDE 的启动时间减少 50%）.每次程序执行时，进行的链接动作都是一样的，链接相对来说开销很大，尤其是嵌入式系统.

2） 普通 Linux 系统的 Prelink

Redhat 系统中 prelink 工具（/etc/cron.dialy/prelink）会修改可执行程序，把它与所需库的链接信息加入可执行程序.在程序运行时，使用 glibc（glibc > 2.3.1-r2）中的 ld-linux.so 来进行链接.用此方式，每次更新动态库后，使用它的程序都需要重新 prelink，因为新库中的符号信息，地址等很可能与原来不同了.

3） Android 的 Prelink

Android 源码中有一组 map 文件，其中定义了需要预连接的动态库，其 Prelink 信息以及对应的逻辑地址（4G 地址空间中位置），在动态库编译时，预处理程序 apriori 根据 map 文件中的定义，生成预链接信息重定向信息，并加入这些二进制文件 lib*.so 的末尾.它主要节约了查询函数地址等工作所用的时间，动态库重定位的开销.在运行程序，动态库加载时，加载程序 linker 判断动态库是否为 Prelink 的，如果是的话，就在首次使用时将其加载到指定的内存空间，直接使用预编译信息.

2. 源码分析

1） 动态库的编译脚本

a） 源代码

frameworks/base/media/libmedia/Android.mk 等库的编译选项文件

b） 配置

可在 Android.mk 中设置该库是否需要 Prelink，默认是使用 Prelink 的，也可设置成否，方法如下：

LOCAL_PRELINK_MODULE ：= false

c） 分析

此设置只用于动态库的编译，编译时用 showcommands 参数即可看到具体编译使用到的命令行，如在某个库的目录中运行 mm showcommands，即可看到相应的 Prelink 操作，示例如下：

target Prelink： libxxx

out/host/linux-x86/bin/apriori --prelinkmap build/core/prelink-linux-arm-2G.map --locals-only --quiet xxx.so --output xxx.so

如果该库设置为需要 prelink，则也需要在 map 文件中加入相应项，否则编译不通过

2） 内核

a） 源码

kernel/arch/arm/configs/xxx_defconfig

arch/arm/mach-msm/include/mach/vmalloc.h

b） 配置

CONFIG_VMSPLIT_2G=y 一般默认为 3G/1G，此项即设置为 2G/2G

c） 分析

xxx_defconfig 为默认的内核配置文件（修改其中的 CONFIG_VMSPLIT_*），也可通过 make menuconfig 配置，与 Prelink 相关的主要是指定用户空间和内核空间内存如何分配 4G 的虚拟内存空间（Memory split），一般有三种方式：3G/1G，2G/2G，1G/3G（user/kernel），一般默认的是用户空间 3G（0x0-0xBFFFFFFF），内核空间 1G（0xC0000000 - 0xFFFFFFFF）

d） 内存分析示例，以 3G/1G 为例（见 build/core/prelink-linux-arm.map）

0xC0000000 - 0xFFFFFFFF Kernel

0xB0100000 - 0xBFFFFFFF Thread 0 static

0xB0000000 - 0xB00FFFFF Linker

0xA0000000 - 0xBFFFFFFF Prelinked System Libraries
0x90000000 - 0x9FFFFFFF Prelinked App Libraries
0x80000000 - 0x8FFFFFFF Non-prelinked Libraries
0x40000000 - 0x7FFFFFFF mmap’s stuff
0x10000000 - 0x3FFFFFFF Thread Stacks
0x00000000 - 0x0FFFFFFF .text / .data / heap

3） map 文件

a） 源代码

build/core/prelink-linux-<arch>*.map

b） 配置

编译时有多个 map 文件可先，根据不同的硬件平台及内存分配（3G/1G， 2G/2G）修改系统配置 deviceBoardConfig.mk 中设置

TARGET_USES_2G_VM_SPLIT ：=true 以配置 Prelink 的地址空间

c） 分析

apriori 中的 prelinkmap.c 它用根据整个系统设置 device/*/BoardConfig.mk 的内存分配规则（3G/1G， 2G/2G）来判断 map 中指定地址是否符合 Prelink 的地址空间范围，如果正常，则在 so 的末尾加入 prelink 信息和标识（文件以 PRE 结束）

apriori 可以预先为若干共享库确定加载地址，并为有依赖关系的共享库做静态重定位和连接， 该命令加入参数--verbose，即可显示出 prelink 的细节.

5） linker 程序

a） 源代码

bionic/linker/*

（bionic 目录中存放一些基础的库，如 libc， libstdc++， libthread_db， linker 等）

b） 分析

linker是 Android的专用动态链接库键接器，Linker和传统 Linux 使用的 linker（ld.so，ld-linux.so.2，ld-linux.so.3）有所不同.库的编译参数-dynamic-linker 指定了键接器为/system/bin/linker（也可以手动换成别的），该信息将被存放在 ELF 文件的.interp 节中，内核执行目标映像文件前将通过该信息加载并运行相应的解释器程序 linker，并链接相应的共享库，共享库以 ELF 文件的形式保存在文件系统中.核心的 load_elf_binary 会首先将其映像文件映射到内存，然后映射并执行其解释器也就是 linker 的代码.linker 的代码段是进程间共享的，但数据段为各进程私有.

所有外部过程引用都在映像执行之前解析， Android 中的共享库和可执行映像都默认采用 ELF 格式的文件.程序头表包含了加载到内存中的各种段的索引及属性信息，它将告诉加载器如何加载映像，初始化时，动态链接器首先解析出外部过程引用的绝对地址，一次性的修改所有相应的 GOT 表项.

linker 会在共享库加载时，调用 is_prelinked 查看该库是否是 prelink 的，并在 alloc_mem_region 中检查目的地址是否被占用.如果该库不是 prelink 的，则库加载的起始地址为零.

10. Evan’s Android 源代码研究

下载源码

首先一个问题是 http：//source.android.com/在国内访问有时，甚至是大多时候，是有问题的.那么怎么翻墙就需要大家自己解决了.其实只是为了看源码的话，问题并不大，源码更新的站点还是可以正常连接的，但是如果有打算要提交 patch，那么用 tor 都有些麻烦，我是找了个 VPN 出去的.获取源码请参照

http：//source.android.com/download

嗯，我这里也可以给一些简单的步骤介绍吧：

1. 你的系统需要是 Linux 或者是 Mac OS X.需要先安装 git-core 和 gnupg，在 Debian/Ubuntu Linux 上，可以用 apt-get 获取，Mac 上使用 Macports 吧.额外的，在 mac 上你需要先安装 XCode 2.4 以上版本，另外再用 macports 安装 gmake， libsdl. 事实上编译期还会有问题，这在 http：//source.android.com/download 页文档中没有提及，缺失 ncursor 和 zlib 库，也需要单独安装（这个在 mail list 里有讨论，或许我晚些可以补全些）.复制内容到剪贴板代码：

正好干净安装了 Ubuntu Linux 8.10 记录一下需要的东西

注意可能 blog 中会自动折行，请以$来区分新行

$ sudo apt-get install git-core gnupg （gnupg 实际已自带）

$ sudo apt-get install flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl

$ sudo apt-get install zlib1g-dev

$ sudo apt-get install valgrind （可选，有 21M 大，我觉得一般人是用不到的）

$ sudo apt-get install python2.5 （实际上不用装，Ubuntu 8.10 已经自带）

$ sudo apt-get install sun-java5-jdk

注意，不要用 sun-java6-jdk， 不然在 make sdk，具体来说是 make doc 这一步中，遇到这个错误：

Docs droiddoc： out/target/common/docs/dx

javadoc： error - In doclet class DroidDoc， method start has thrown an

exception java.lang.reflect.InvocationTargetException

com.sun.tools.javac.code.Symbol$CompletionFailure： class file for

sun.util.resources.OpenListResourceBundle not found2. 下载 repo 脚本，放到/bin 目录下，加上可执行权限

$ curl http：//android.git.kernel.org/repo > repo

$ sudo mv repo /bin

$ sudo chmod a+x /bin/repo

其实文档中是把这个放在个人用户的~/bin 目录下，但是要改 PATH，我嫌麻烦而已就放/bin 下了

3. 创建一个放置源码的文件夹，如叫 myandroid.这里要注意下，需要区分大小写的分区，Linux 下一般是用 ext3 的，是符合要求的，而 mac 下默认的分区是不区分大小写的，请自行创建分区.但是事实上，我以前测试过，在非大小写敏感分区上，直接 make 会报错，但修改 Makefile 后直接跳过检查这一步，最终是可以成功编译的.

4. 在 myandroid 目录中执行

$ repo init -u git：//android.git.kernel.org/platform/manifest.git

中间会提示输入电子邮件什么的，如果你打算要提交 patch 的话，用 google accounts 注册过的邮箱

5. 在 myandroid 中执行 repo sync 就可以开始下载源码了

但是这一步非常慢，这里有些文档中没有的技巧了：

测试发现，主站点是限三线程连接的，也就是说你可以同时开三个窗口做不同部分下载，

我建议是第一个窗口直接执行 repo sync，第二个窗口执行 repo sync kernel，第三个窗口执行 repo sync prebuilt

因为 kernel 和 prebuilt 这两个工程比较大.

有时因为多次频繁连接，会有服务器端拒绝连接的错误，此时等两三分钟再重试就好了

repo 的使用请看 repo help.如果只是要学习源码的话，那只用 repo sync 一个命令就行.repo sync 不带参数的话会更新所有子项目，可以 repo sync project_path 来指定更新项目. 那么 project_path 在哪可以找到呢？myandroid/.repo/manifests/default.xml （你至少需要先 repo init 过才有这些东西）

编译源码

当 repo sync 全部完毕时，进入 myandroid 目录，首先执行 make，大约会耗时一个小时左右（在我的一代

macbook pro 上大概是这时间）.make 最后会生成 system.img.看到 mail list 里有人问这个干嘛用，其实是文档里没有完全写清楚.一般来说，我想普通人还需要的是 make sdk 这一步，会在 myandroid/out/host/YOUR_OS/sdk 中生成 sdk.和从 google 官方下载下来的 sdk 差不多，可以直接运行了

在进行过一次完整的 make 后，以后对一些程序的修改，大可不必重新 make sdk，因为 make sdk 实在太慢了.

先在 myandroid 目录下执行

$. build/envsetup.sh

然后你就会多出几个可用的命令.在改了 Contacts 项目后，可以简单的执行 mmm packages/apps/Contacts/来单独编译这个部分.为了可以直接测试改动，需要生成新的 system.img，在 myandroid 目录下执行

make snod 即可 当然，如果你改动的是 emulator 或者其它外围相关的，而非系统内部的东西，就不只是要重新生成 system.img 了

修改测试

这个周末，我提交了大约四到五个 patch.主要是联系人程序相关的一些 bug 修正.如果你是想修改 build in 的程序，像联系人这样的，最好先在编译前把 Contacts 包删去，这样编译出来的 system.img 不带 Contacts 程序会方便很多，可以直接用 eclipse 开发 Contacts 程序，在这个模拟器上调试.像一般情况下，Contacts 是带在 system.img 内的，对程序修改后想测试，比较麻烦，需要先运行 emulator，然后 adb uninstall 掉原先的程序，再安装自己的，并且如果数据没卸干净，会有 permission 出错或 uid 不否而报错的情况.最最麻烦的是，每次重启 emulator，build in 的程序在被 adb uninstall 后又会出现，这个问题在真机上不会出现.所以我建议你直接去掉 Contacts 再编译个 system.img 出来，不过今天的几个修改都比较简单，我全部都是改一点就用 mmm 重新编译 Contacts 程序，然后用 make snod 来出一个新的 system.img，覆盖掉 sdk 目录下旧有的文件，来测试.这个过程熟练了一般两分多钟可以测一把.

11.Android 2.2新增Widget之ProtipWidget源码

在 Android 2.2 SDK 中我们可能首次启动模拟器可以看到和以前不一样的是多出了一个绿色的小机器人提示信息，Google 给我们了演示了 Android 中如何通过 RemoteView 和简单的图片轮换方式实现动画效果在桌面小工具中，appWidget 的基类时 AppWidgetProvider 类，不过 Widget 本身的生命周期管理并非 Activity，相对于的而是 BroadcastReceiver 广播方式处理的，Android 2.2 新增的 Widget 的实现大家可以从中学习到很多有用的知识.

```
public class ProtipWidget extends AppWidgetProvider {
public static final String ACTION_NEXT_TIP = "com.android.misterwidget.NEXT_TIP";
//定义 action 切换到下一条提示
public static final String ACTION_POKE = "com.android.misterwidget.HEE_HEE";
//唤醒小机器人
public static final String EXTRA_TIMES = "times";
public static final String PREFS_NAME = "Protips";
public static final String PREFS_TIP_NUMBER = "widget_tip";
private static Random sRNG = new Random ();
//轮换图片用到的，生成一个静态的随机数生成器
private static final Pattern sNewlineRegex = Pattern.compile (" *\\n *");
private static final Pattern sDrawableRegex = Pattern.compile (" *@ (drawable/[a-z0-9_]+)  *");
// 初始化时 Droid 是眼睛没有睁开，同时没有信息提示
```

```
private int mIconRes = R.drawable.droidman_open;
private int mMessage = 0;
private AppWidgetManager mWidgetManager = null;
private int[] mWidgetIds;
private Context mContext;
private CharSequence[] mTips;
private void setup (Context context) {
mContext = context;
mWidgetManager = AppWidgetManager.getInstance (context);
mWidgetIds = mWidgetManager.getAppWidgetIds (new ComponentName (context, ProtipWidget.class));
SharedPreferences pref = context.getSharedPreferences (PREFS_NAME, 0);
mMessage = pref.getInt (PREFS_TIP_NUMBER, 0);
mTips = context.getResources () .getTextArray (R.array.tips);
if (mTips ! = null) {
if (mMessage >= mTips.length) mMessage = 0;
} else {
mMessage = -1;
}
}
public void goodmorning () {
//Android 开发网提示线程中轮换图片，使用了 500，200，100 等这样的 0.5 秒休眠，0.2 秒休眠实现了动画的间隔效果
mMessage = -1;
try {
setIcon (R.drawable.droidman_down_closed);
Thread.sleep (500);
setIcon (R.drawable.droidman_down_open);
Thread.sleep (200);
setIcon (R.drawable.droidman_down_closed);
Thread.sleep (100);
setIcon (R.drawable.droidman_down_open);
Thread.sleep (600);
} catch (InterruptedException ex) {
}
mMessage = 0;
mIconRes = R.drawable.droidman_open;
refresh ();
}
@Override
public void onReceive (Context context, Intent intent) {
//上面已经讲到了，appWidget 是基于 broadcastreceiver 类的，所以说需要响应 onReceive 通过 action 来驱动事件.
setup (context);
if (intent.getAction () .equals (ACTION_NEXT_TIP)) {
```

```
mMessage = getNextMessageIndex ();
SharedPreferences.Editor pref = context.getSharedPreferences (PREFS_NAME,  0) .edit ();
pref.putInt (PREFS_TIP_NUMBER,  mMessage);
pref.commit ();
refresh ();
} else if  (intent.getAction () .equals (ACTION_POKE))  {
blink (intent.getIntExtra (EXTRA_TIMES,  1));
} else if  (intent.getAction () .equals (AppWidgetManager.ACTION_APPWIDGET_ENABLED))  {
goodmorning ();
} else {
mIconRes = R.drawable.droidman_open;
refresh ();
}
}
private void refresh ()  {  //管理如果有多个本 widget 执行需要逐个更新
RemoteViews rv = buildUpdate (mContext);
for  (int i :  mWidgetIds)  {
mWidgetManager.updateAppWidget (i,  rv);
}
}
private void setIcon (int resId)  {
mIconRes = resId;
refresh ();
}
private int getNextMessageIndex ()  {
return  (mMessage + 1)  % mTips.length;
}
private void blink (int blinks)  {
if  (mMessage < 0)  return;
//这里使用的是图片轮换方式来实现动画效果，在 appWidget 中我们可以用的控件十分少
setIcon (R.drawable.droidman_closed);
try {
Thread.sleep (100);
while  (0<--blinks)  {
setIcon (R.drawable.droidman_open);
Thread.sleep (200);
setIcon (R.drawable.droidman_closed);
Thread.sleep (100);
}
} catch  (InterruptedException ex)  { }
setIcon (R.drawable.droidman_open);
}
public RemoteViews buildUpdate (Context context)  {
RemoteViews updateViews = new RemoteViews (
```

```
context.getPackageName (),  R.layout.widget);
//映射布局，widget.xml 文件的源码在下面可以找到
// 按下 bubble 的事件，对应 action_next_tip 动作
Intent bcast = new Intent (context,  ProtipWidget.class);
bcast.setAction (ACTION_NEXT_TIP);
PendingIntent pending = PendingIntent.getBroadcast (
context,  0,  bcast,  PendingIntent.FLAG_UPDATE_CURRENT);
updateViews.setOnClickPendingIntent (R.id.tip_bubble,  pending);
//这里为 action_poke
bcast = new Intent (context,  ProtipWidget.class);
bcast.setAction (ACTION_POKE);
bcast.putExtra (EXTRA_TIMES,  1);
pending = PendingIntent.getBroadcast (
context,  0,  bcast,  PendingIntent.FLAG_UPDATE_CURRENT);
updateViews.setOnClickPendingIntent (R.id.bugdroid,  pending);
// Tip bubble text
if  (mMessage >= 0)  {
String[] parts = sNewlineRegex.split (mTips[mMessage],  2);
String title = parts[0];
String text = parts.length > 1 ?  parts[1] :  "";
// Look for a callout graphic referenced in the text
Matcher m = sDrawableRegex.matcher (text);
if  (m.find ())  {
String imageName = m.group (1);
int resId = context.getResources () .getIdentifier (
imageName,  null,  context.getPackageName ());
updateViews.setImageViewResource (R.id.tip_callout,  resId);
updateViews.setViewVisibility (R.id.tip_callout,  View.VISIBLE);
text = m.replaceFirst ("");
} else {
updateViews.setImageViewResource (R.id.tip_callout,  0);
updateViews.setViewVisibility (R.id.tip_callout,  View.GONE);
}
updateViews.setTextViewText (R.id.tip_message,
text);
updateViews.setTextViewText (R.id.tip_header,
title);
updateViews.setTextViewText (R.id.tip_footer,
context.getResources () .getString (
R.string.pager_footer,
 (1+mMessage),  mTips.length));
updateViews.setViewVisibility (R.id.tip_bubble,  View.VISIBLE);
} else {
updateViews.setViewVisibility (R.id.tip_bubble,  View.INVISIBLE);
```

```
}
updateViews.setImageViewResource (R.id.bugdroid,  mIconRes);
return updateViews;
}
}
```

有关 AndroidManifest.xml 中详细的 recevier 代码如下

```
<receiver android: name=".ProtipWidget" android: label="@string/widget_name">
<intent-filter>
<action android: name="android.appwidget.action.APPWIDGET_UPDATE" />
<action android: name="com.android.protips.NEXT_TIP" />
<action android: name="com.android.protips.HEE_HEE" />
</intent-filter>
<meta-data android: name="android.appwidget.provider" android: resource="@xml/widget_build" />
</receiver>
```

有关 res/xml/widget_build.xml 的代码如下

```
<appwidget-provider xmlns: android="http: //schemas.android.com/apk/res/android"
android: minWidth="294dip"
android: minHeight="72dip"
android: updatePeriodMillis="0"
android: initialLayout="@layout/widget" />
```

有关 res/layout/widget.xml 的代码如下，注意下面使用了布局文件套嵌的 include 方式

```
<RelativeLayout xmlns: android="http: //schemas.android.com/apk/res/android"
android: id="@+id/widget"
android: layout_width="fill_parent"
android: layout_height="wrap_content"
android: orientation="vertical"
android: padding="5dip"
>
<include layout="@layout/droid" />
<include layout="@layout/bubble" />
</RelativeLayout>
```

有关 res/layout/droid.xml 的代码如下

```
<ImageView xmlns: android="http: //schemas.android.com/apk/res/android"
android: id="@+id/bugdroid"
android: src="@drawable/droidman_down_closed"
android: scaleType="center"
android: layout_width="wrap_content"
android: layout_height="wrap_content"
android: layout_alignParentRight="true"
android: layout_centerVertical="true"
/>
```

有关 res/layout/bubble.xml 的代码如下

```
<RelativeLayout xmlns: android="http: //schemas.android.com/apk/res/android"
android: id="@+id/tip_bubble"
```

```
android: layout_width="fill_parent"
android: layout_height="wrap_content"
android: layout_toLeftOf="@+id/bugdroid"
android: layout_centerVertical="true"
android: gravity="center_vertical|left"
android: layout_marginRight="2dip"
android: visibility="invisible"
android: background="@drawable/droid_widget"
android: focusable="true"
>
<TextView
android: layout_width="0dip"
android: layout_height="0dip"
android: layout_alignParentTop="true"
android: layout_marginTop="-100dip"
android: text="@string/widget_name"
/>
<TextView
android: layout_width="0dip"
android: layout_height="0dip"
android: layout_alignParentTop="true"
android: layout_marginTop="-90dip"
android: text="@string/tts_pause"
/>
<TextView
android: id="@+id/tip_footer"
style="@style/TipText.Footer"
android: layout_width="wrap_content"
android: layout_height="wrap_content"
android: layout_alignParentBottom="true"
android: layout_alignParentRight="true"
android: layout_marginRight="1dip"
/>
<ImageView
android: id="@+id/tip_callout"
android: layout_width="wrap_content"
android: layout_height="fill_parent"
android: gravity="center"
android: layout_alignParentTop="true"
android: layout_alignParentRight="true"
android: layout_above="@id/tip_footer"
android: visibility="gone"
android: padding="4dip"
/>
```

```
<TextView
android: id="@+id/tip_header"
style="@style/TipText.Header"
android: layout_width="fill_parent"
android: layout_height="wrap_content"
android: layout_alignParentTop="true"
android: layout_toLeftOf="@id/tip_callout"
android: layout_alignWithParentIfMissing="true"
android: layout_marginTop="0dip"
android: layout_marginLeft="3dip"
/>
<TextView
android: id="@+id/tip_message"
style="@style/TipText.Message"
android: layout_width="fill_parent"
android: layout_height="wrap_content"
android: layout_below="@id/tip_header"
android: layout_alignLeft="@id/tip_header"
android: layout_alignRight="@id/tip_header"
android: layout_marginTop="1dip"
/>
</RelativeLayout>
```

有关上面 bubble.xml 中的 drawable 对象 droid_widget 的代码如下

```
<selector xmlns: android="http: //schemas.android.com/apk/res/android">
<item android: state_pressed="true" android: drawable="@drawable/droid_widget_pressed" />
  <item android : state_focused="true" android : state_window_focused="true" android : drawable="@drawable/droid_widget_focused" />
  <item android : state_focused="true" android : state_window_focused="false" android : drawable="@drawable/droid_widget_normal" />
<item android: drawable="@drawable/droid_widget_normal" />
</selector>
```

四、源码编译

1.Ubuntu 8.04下编译Android源码全过程

一、获取 Android 源代码

Git 是 Linux Torvalds（Linux 之父）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），而在 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，它们支持离线工作，本地提交可以稍后提交到服务器上.

因为 Android 是由 Kernel、Dalvik、Bionic、Prebuilt、build 等多个项目组成，如果我们分别使用 Git 来逐个获取显得很麻烦，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，使得项目的获取更加简单.

在 Ubuntu 8.04 上安装 Git 只需要设定正确的更新源，然后使用 apt-get 就可以了，apt-get 是一条 Linux 命令，主要用于自动从互联网的软件仓库中搜索、安装、升级、卸载软件或操作系统.

apt-get 命令一般需要 root 权限执行，所以一般跟着 sudo 命令.

sudo apt-get install git-core curl

这条命令会从互联网的软件仓库中安装 git-core 和 curl.

其中 curl 是一个利用 URL 语法在命令行方式下工作的文件传输工具，它支持很多协议，包括 FTP、FTPS、HTTP、HTTPS、TELENT 等，我们需要安装它从网络上获取 Repo 脚本文件.

curl http：//android.git.kernel.org/repo >~/bin/repo

这句命令会下载 repo 脚本文件到当前主目录的/bin 目录下，并保存在文件 repo 中.

最后我们需要给 repo 文件可执行权限

chmod a+x ~/bin/repo

接下来我们就可以利用 repo 脚本和 Git、curl 软件获取 Android 的源代码了：）

首先建一个目录，比如~/android.

然后使用下面命令获取源码：

repo init －u git：//android.git.kernel.org/platform/manifest.git

这个过程会持续很长的时间（笔者下载了一天），下载完毕后会看到 repo initialized in /android 这样的提示，说明本地的版本库已经初始化完毕，并且包含了当前最新的 sourcecode.

如果我们想拿某个分支版本的代码，而不是主线代码，需要使用-b 参数指定 branch 的名字，比如：

repo init －u git：//android.git.kernel.org/platform/manifest.git －b cupcake

如果我们只是想获取某一个 project 的代码，比如 kernel/common，就不需要 repo 脚本了，直接使用 Git 工具即可，如果仔细研究 repo 脚本会发现，repo 脚本其实就是组织 Git 工具去获取各个 Project 并把它们组织到同一个项目 Android 内.

git clone git：//android.git.kernel.org/kernel/common.git

我们上面使用 repo 脚本获取了各个项目，那么接下来就需要把整个 Android 代码树同步到本地，如下：

repo sync project1 project2 …

笔者使用 repo sync 命令直接同步全部项目到本地.

二、源码编译

全部同步完毕后，进入到 Android 目录下，使用 make 命令编译，你会发现出现如下错误提示：

host C： libneo_cgi <= external/clearsilver/cgi/cgi.c

external/clearsilver/cgi/cgi.c：22：18： error： zlib.h： No such file or directory

这个错误是因为我们缺少 zlib1g-dev，需要使用 apt-get 命令从软件仓库中安装，如下：

sudo apt-get install zlib1g-dev

同理，我们还需要依次安装如下软件

sudo apt-get install flex

sudo apt-get install bison

sudo apt-get install gperf

sudo apt-get install libsdl-dev

sudo apt-get install libesd0-dev

sudo apt-get install libncurses5-dev

sudo apt-get install libx11-dev

以上软件全部安装完毕后，运行 make 命令再次编译 Android 源码.

这个时候你会发现出现很多 java 文件无法编译的错误，打开 Android 的源码我们可以看到在 android/dalvik/libcore/dom/src/test/java/org/w3c/domts 下有很多 java 源文件，这意味着编译 Android 之前需要先安装 JDK.

首先从 sun 官方网站下载 jdk-6u16-linux-i586.bin 文件并安装它.

在 Ubuntu 8.04 中，/etc/profile 文件是全局的环境变量配置文件，它适用于所有的 shell.在我们登陆 Linux 系统时，首先启动/etc/profile 文件，然后再启动用户目录下的~/.bash_profile、~/.bash_login 或~/.profile 文件中的其中一个，执行的顺序和上面的排序一样.如果~/.bash_profile 文件存在的话，一般还会执行~/.bashrc 文件.

所以我们只需要把 JDK 的目录放到/etc/profile 里即可，如下：

JAVA_HOME=/usr/local/src/jdk1.6.0_16

PATH=$PATH：$JAVA_HOME/bin：/usr/local/src/android-sdk-linux_x86-1.1_r1/tools：~/bin

然后重新启动一下机器，输入 java －version 命令，提示如下信息代表配置成功：

java version "1.6.0_16"

Java（TM） SE Runtime Environment （build 1.6.0_16-b01）

Java HotSpot（TM） Client VM （build 14.2-b01， mixed mode， sharing）

在编译完整个项目后，终端会出现如下提示：

Target system fs image ：out/target/product/generic/obj/PACKAGING/systemimage_unopt_intermediates/system.img

Install system fs image： out/target/product/generic/system.img

Target ram disk： out/target/product/generic/ramdisk.img

Target userdata fs image： out/target/product/generic/userdata.img

Installed file list： out/target/product/generic/installed-files.txt

root@dfsun2009-desktop：/bin/android#

三、源码运行

在编译完整个项目后，如果我们需要观看编译后的运行效果，那么就需要在系统中安装模拟器 android-sdk-linux_x86-1.1_r1，这个 SDK 的下载地址为：

linux： http：//dl.google.com/android/android-sdk-linux_x86-1.1_r1.zip

mac： http：//dl.google.com/android/android-sdk-mac_x86-1.1_r1.zip

windows： http：//dl.google.com/android/android-sdk-windows-1.1_r1.zip

解压后需要把/usr/local/src/android-sdk-linux_x86-1.1_r1/tools 目录加入到系统环境变量/etc/profile 中.

然后找到编译后 android 的目录文件 out，我们发现在 android/out/host/linux-x86/bin 下多了很多应用程序，这些应用程序就是 android 得以运行的基础，所以我们需要把这个目录也添加到系统 PATH 下，在$HOME/.profile 文件中加入如下内容：

PATH=”$PATH：$HOME/android/out/host/linux-x86/bin”

接下来我们需要把 android 的镜像文件加载到 emulator 中，使得 emulator 可以看到 android 运行的实际效果，在$HOME/.profile 文件中加入如下内容：

ANDROID_PRODUCT_OUT=$HOME/android/out/target/product/generic

export ANDROID_PRODUCT_OUT

然后重新启动机器.

下面就可以进入到模拟器目录中并启动模拟器

cd $HOME/android/out/target/product/generic

emulator -image system.img -data userdata.img -ramdisk ramdisk.img

总结一下安装过程中的关键点：

1：JDK 版本必须安装

2：利用下面的命令确保所需软件都被正确安装

sudo apt-get install flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential python valgrind curl git

3：内存及虚拟内存保证在 2GB 以上，可以采用命令行 free -m 查看内存是否足够，如果内存不够终端停滞

2. 详解Android源码的编译

1、安装一些环境

01.sudo apt-get install build-essential 02. sudo apt-get install make 03. sudo apt-get install gcc 04. sudo apt-get install g++ 05. sudo apt-get install libc6-dev 06. 07. sudo apt-get install patch 08. sudo apt-get install texinfo 09. sudo apt-get install libncurses-dev 10. 11. sudo apt-get install git-core gnupg 12. sudo apt-get install flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl 13. sudo apt-get install ncurses-dev 14. sudo apt-get install zlib1g-dev 15. sudo apt-get install valgrind 16. sudo apt-get install python2.5 如果是 64 位系统还需安装：

sudo apt-get install libc6-dev-i386

apt-get install g++-multilib lib32z1-dev lib32ncurses5-dev

安装 java 环境

01.sudo apt-get install sun-java6-jre sun-java6-plugin sun-java6-fonts sun-java6-jdk 注：官方文档说如果用 sun-java6-jdk 可出问题，得要用 sun-java5-jdk.经测试发现，如果仅仅 make（make 不包括 make sdk），用 sun-java6-jdk 是没有问题的.而 make sdk，就会有问题，严格来说是在 make doc 出问题，它需要的 javadoc 版本为 1.5.

因此，我们安装完 sun-java6-jdk 后最好再安装 sun-java5-jdk，或者只安装 sun-java5-jdk.这里 sun-java6-jdk 和 sun-java5-jdk 都安装，并只修改 javadoc.1.gz 和 javadoc.因为只有这两个是 make sdk 用到的.这样的话，除了 javadoc 工具是用 1.5 版本，其它均用 1.6 版本：

01.sudo apt-get install sun-java5-jdk 修改 javadoc 的 link

01.cd /etc/alternatives 02.sudo rm javadoc.1.gz 03.sudo ln -s /usr/lib/jvm/java-1.5.0-sun/man/man1/javadoc.1.gz javadoc.1.gz 04.sudo rm javadoc 05.sudo ln -s /usr/lib/jvm/java-1.5.0-sun/bin/javadoc javadoc 2、设置环境变量

01.vim ~/.bashrc 在.bashrc 中新增或整合 PATH 变量，如下

01.#java 程序开发/运行的一些环境变量 02.JAVA_HOME=/usr/lib/jvm/java-6-sun 03.JRE_HOME=${JAVA_HOME}/jre 04.export Android_JAVA_HOME=$JAVA_HOME 05.export CLASSPATH=. : ${JAVA_HOME}/lib : $JRE_HOME/lib : $CLASSPATH 06.export JAVA_PATH=${JAVA_HOME}/bin：${JRE_HOME}/bin 07.export JAVA_HOME； 08.export JRE_HOME； 09.export CLASSPATH； 10.HOME_BIN=~/bin/ 11.export PATH=${PATH}：${JAVA_PATH}：${JRE_PATH}：${HOME_BIN}； 12.#echo $PATH； 最后，同步这些变化：

01.source ~/.bashrc 3、安装 repo（用来更新 Android 源码）

创建~/bin 目录，用来存放 repo 程序，如下：

01.$ cd ~ 02.$ mkdir bin 并加到环境变量 PATH 中，在第 2 步中已经加入

下载 repo 脚本并使其可执行：

01.$ curl http：//Android.git.kernel.org/repo >~/bin/repo 02.$ chmod a+x ~/bin/repo 4、下载 Android 源码并更新之

建议不要用 repo 来下载（Android 源码超过 1G，非常慢），直接在网上下载 http：//www.Androidin.com/bbs/pub/cupcake.tar.gz.而且解压出来的 cupcake 下也有.repo 文件夹，可以通过 repo sync 来更新 cupcake 代码：

01.tar -xvf cupcake.tar.gz repo sync（更新很慢，用了 3 个小时）

5、编译 Android 源码，并得到~/project/Android/cupcake/out 目录

进入 Android 源码目录：

make

这一过程很久（2 个多小时）

6、在模拟器上运行编译好 Android

Android SDK 的 emulator 程序在 Android-sdk-linux_x86-1.0_r2/tools/下，emulator 是需要加载一些 image 的，默认加载 Android-sdk-linux_x86-1.0_r2/tools/lib/images 下的 kernel-qemu（内核） ramdisk.img system.img userdata.img

编译好 Android 之后，emulator 在~/project/Android/cupcake/out/host/linux-x86/bin 下，ramdisk.img system.img userdata.img 则在~/project/Android/cupcake/out/target/product/generic 下

cd ~/project/Android/cupcake/out/host/linux-x86/bin

增加环境变量

01.vim ~/.bashrc 在.bashrc 中新增环境变量，如下

01.#java 程序开发/运行的一些环境变量 02.export Android_PRODUCT_OUT=~/project/Android/cupcake2/out/target/product/generic 03.Android_PRODUCT_OUT_BIN=~/project/Android/cupcake2/out/host/linux-x86/bin 04.export PATH=${PATH}:${Android_PRODUCT_OUT_BIN}； 最后，同步这些变化：

01.source ~/.bashrc 02.emulator -image system.img -data userdata.img -ramdisk ramdisk.img 最后进入 Android 桌面，就说明成功了.

out/host/linux-x86/bin 下生成许多有用工具（包括 Android SDK/tools 的所有工具），因此，可以把 eclipse 中 Android SDK 的路径指定到 out/host/linux-x86/bin 进行开发

7、编译 linux kernel

直接 make Android 源码时，并没有 make linux kernel.因此是在运行模拟器，所以不用编译 linux kernel.如果要移值 Android，或增删驱动，则需要编译 linux kernel

linux kernel 的编译将在以后的文章中介绍.

8、编译模块

Android 中的一个应用程序可以单独编译，编译后要重新生成 system.img

在源码目录下执行

01 build/envsetup.sh （.后面有空格） 就多出一些命令：

01.- croot： Changes directory to the top of the tree. 02.- m： Makes from the top of the tree. 03.- mm： Builds all of the modules in the current directory. 04.- mmm： Builds all of the modules in the supplied directories. 05.- cgrep： Greps on all local C/C++ files. 06.- jgrep： Greps on all local Java files. 07.- resgrep： Greps on all local res/*.xml files. 08.- godir： Go to the directory containing a file.

可以加—help 查看用法

我们可以使用 mmm 来编译指定目录的模块，如编译联系人：

01.mmm packages/apps/Contacts/

编完之后生成两个文件：

01.out/target/product/generic/data/app/ContactsTests.apk 02.out/target/product/generic/system/app/ Contacts.apk

可以使用 make snod 重新生成 system.img

再运行模拟器

9、编译 SDK

直接执行 make 是不包括 make sdk 的.make sdk 用来生成 SDK，这样，我们就可以用与源码同步的 SDK 来开发 Android 了.

1）修改/frameworks/base/include/utils/Asset.h

‘UNCOMPRESS_DATA_MAX = 1 * 1024 * 1024’ 改为 ‘UNCOMPRESS_DATA_MAX = 2 * 1024 * 1024’

原因是 Eclipse 编译工程需要大于 1.3M 的 buffer

2）编译 ADT.

注意，我们是先执行 2），再执行 3）.因为在执行./build_server.sh 时，会把生成的 SDK 清除了.

用上了新的源码，adt 这个调试工具也得自己来生成，步骤如下：

进入 cupcake 源码的 development/tools/eclipse/scripts 目录，执行：

export ECLIPSE_HOME=你的 eclipse 路径

./build_server.sh 你想放 ADT 的路径

3）执行 make sdk.

注意，这里需要的 javadoc 版本为 1.5，所以你需要在步骤 1 中同时安装 sun-java5-jdk

make sdk

编译很慢.编译后生成的 SDK 存放在 out/host/linux-x86/sdk/，此目录下有 Android-sdk_eng.xxx_linux-x86.zip 和 Android-sdk_eng.xxx_linux-x86 目录.Android-sdk_eng.xxx_linux-x86 就是 SDK 目录

实际上，当用 mmm 命令编译模块时，一样会把 SDK 的输出文件清除，因此，最好把 Android-sdk_eng.xxx_linux-x86 移出来

4）关于环境变量、Android 工具的选择

目前的 Android 工具有：

A、我们从网上下载的 SDK（ tools 下有许多 Android 工具，lib/images 下有 img 映像）

B、我们用 make sdk 编译出来的 SDK（ tools 下也有许多 Android 工具，lib/images 下有 img 映像）

C、我们用 make 编译出来的 out 目录（ tools 下也有许多 Android 工具，lib/images 下有 img 映像）

那么我们应该用那些工具和 img 呢？

首先，我们不会用 A 选项的工具和 img，因为一般来说它比较旧，也源码不同步.测试发现，如果使用 B 选项的工具和 img，Android 模拟器窗口变小（可能是 skin 加载不了），而用 C 选项的工具和 img 则不会有此问题.

有些 Android 工具依赖 Android.jar（比如 Android），因此，我们在 eclipse 中使用 B 选项的工具（SDK），使用 C 选项的 img.其实，从 emulator -help-build-images 也可以看出，Android_PRODUCT_OUT 是指向 C 选项的 img 目录的

不过，除了用 A 选项的工具和 img，用 B 或 C 的模拟器都不能加载 sdcard，原因还不清楚.

5）安装、配置 ADT

安装、配置 ADT 请参考官方文档

6）创建 Android Virtual Device

编译出来的 SDK 是没有 AVD（Android Virtual Device）的，我们可以通过 Android 工具查看：

Android list

输出为：

01.Available Android targets： 02.[1] Android 1.5 03. API level： 3 04. Skins： HVGA-P，QVGA-L， HVGA-L， HVGA （default）， QVGA-P 05.Available Android Virtual Devices：

表明没有 AVD.如果没有 AVD，eclipse 编译工程时会出错（Failed to find a AVD compatible with target 'Android 1.5'. Launch aborted.）

创建 AVD：

Android create avd -t 1 -c ~/sdcard.img -n myavd

可以 Android －help 来查看上面命令选项的用法.创建中有一些选项，默认就行了

再执行 Android list，可以看到 AVD 存放的位置

以后每次运行 emulator 都要加-avd myavd 或@myavd 选项，这里 eclipse 才会在你打开的 emulator 中调试程序

3.Ubuntu9.10下编译Android源码

在 Ubuntu8.04 下，应该一切很顺利，但是我装的是 9.10 版本，所以会有些问题.

先装 jdk1.5，网上说 1.6 的有点小问题，所以我没装 1.6，选择了 1.5. 然后一边装必须的一些程序，apt-get install **** 很多乱七八糟的，网上能搜到，然后一边下载 android 源码.

源码的下载需要 git，curl 和 repo，git 和 curl 可以直接用 apt-get install git-core curl 就可以了

然后下载 repo

curl http：//android.git.kernel.org/repo >~/bin/repo

添加可执行权限

sudo chmod +x repo

初始化 repo

repo init -u git：//android.git.kernel.org/platform/manifest.git

如果只需要某一个版本，比如我只要 cupcake

repo init -u git：//android.git.kernel.org/platform/manifest.git -b cupcake

然后建个目录，

mkdir android

进入

cd android

下载

repo sync

然后就等吧，大概两个 G 左右的内容，在获取 39%的时候和 95%的时候是最慢的，因为 39%的有一百多 M，

95%的有差不多 400M.

下载完之后，先不要 make，因为还有几个问题：

1.如果直接 make，会出现 frameworks/policies/base/PolicyConfig.mk：22： *** No module defined for the given PRODUCT_POLICY （android.policy_phone）. Stop.错误.

解决办法：

在 build/tools/findleaves.sh 中的第 89 行，

这一句 find "${@：0：$nargs}" $findargs -type f -name "$filename" -print |

改为 find "${@：1：$nargs-1}" $findargs -type f -name "$filename" -print |

2.frameworks/base/tools/aidl/AST.cpp：10： error： 'fprintf' was not declared in this scope 的错误

解决办法：

下载 gcc-4.3 和 g++-4.3

apt-get install gcc-4.3 g++-4.3

大约十多兆，然后

进入/usr/bin

cd /usr/bin

建个软连接

ln -s gcc-4.3 gcc

ln -s g++-4.3 g++

然后进入 android 目录下，执行 make，就可以了.

我在虚拟机里面 make 的，分了 1G 内存给他，处理器 50%，一共花了两个小时才编译完成.编译完成后，在 android 目录下的 out 文件夹中，就是结果，也有两个 G 左右.

在 out/target/product/generic 目录下，有三个 img 文件，system.img，ramdisk.img，userdata.img，这三个就是需要的 android 系统镜像文件，放到模拟器中运行就行了.

4.ubuntu 11.10（32位）下android2.2 源码编译

源码编译镜像文件

开始正式编译，在源码目录下 make 即可.

make

我完全不知道跑了个啥，只知道超级卡.完成后产生的 out 文件夹有 3.9G.

镜像生成在 out/target/product/generic 下：android 源码编译后得到 system.img，ramdisk.img，userdata.img 映像文件.其中， ramdisk.img 是 emulator 的文件系统，system.img 包括了主要的包、库等文件，userdata.img 包括了一些用户数据，emulator 加载这 3 个映像文件后，会把 system 和 userdata 分别加载到 ramdisk 文件系统中的 system 和 userdata 目录下.

当然编译是件很痛苦的事情：

错误 1：

****

You are attempting to build on a 32-bit system.

Only 64-bit build environments are supported beyond froyo/2.2.

****

解决：

需要进行如下修改即可，将

./external/clearsilver/cgi/Android.mk

./external/clearsilver/java-jni/Android.mk
./external/clearsilver/util/Android.mk
./external/clearsilver/cs/Android.mk
四个文件中的
LOCAL_CFLAGS += -m64
LOCAL_LDFLAGS += -m64
注释掉，或者将“64”换成“32”
LOCAL_CFLAGS += -m32
LOCAL_LDFLAGS += -m32
然后，将./build/core/main.mk 中的
ifneq （64，$（findstring 64，$（build_arch）））
改为：
ifneq （i686，$（findstring i686，$（build_arch）））
错误 2：
host C++： libutils <= frameworks/base/libs/utils/RefBase.cpp
frameworks/base/libs/utils/RefBase.cpp：In member function ‘void android::RefBase::weakref_type::trackMe（bool， bool）’：
frameworks/base/libs/utils/RefBase.cpp：483：67： error： passing ‘const android:: RefBase:: weakref_impl’ as ‘this’ argument of ‘void android:: RefBase:: weakref_impl:: trackMe（bool， bool）’ discards qualifiers [-fpermissive]
make： *** [out/host/linux-x86/obj/STATIC_LIBRARIES/libutils_intermediates/RefBase.o] 错误 1
解决：
gedit frameworks/base/libs/utils/Android.mk
Change the line：
LOCAL_CFLAGS += -DLIBUTILS_NATIVE=1 $（TOOL_CFLAGS）
To：
LOCAL_CFLAGS += -DLIBUTILS_NATIVE=1 $（TOOL_CFLAGS） -fpermissive
错误 3：
host Executable： aapt （out/host/linux-x86/obj/EXECUTABLES/aapt_intermediates/aapt）
out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/libcutils.a （ threads.o ）： In function `thread_store_get'：
/home/leno/works/android_dev/bin/system/core/libcutils/threads.c:27： undefined reference to `pthread_getspecific'
out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/libcutils.a （ threads.o ）： In function `thread_store_set'：
/home/leno/works/android_dev/bin/system/core/libcutils/threads.c:36： undefined reference to `pthread_key_create'
/home/leno/works/android_dev/bin/system/core/libcutils/threads.c:44： undefined reference to `pthread_setspecific'
collect2： ld returned 1 exit status
make： *** [out/host/linux-x86/obj/EXECUTABLES/aapt_intermediates/aapt] 错误 1
解决：
打开 frameworks/base/tools/aapt/Android.mk
然后打开文件 Android.mk，编辑下面一行：
ifeq （$（HOST_OS），linux）
#LOCAL_LDLIBS += -lrt 把这行注释掉，改为下面一行.
LOCAL_LDLIBS += -lrt -lpthread

endif
错误 4：
target Dex： core
#
An unexpected error has been detected by HotSpot Virtual Machine：
#
SIGSEGV （0xb） at pc=0x4003d848， pid=7668， tid=2889534320
#
Java VM： Java HotSpot（TM） Client VM （1.5.0_22-b03 mixed mode）
Problematic frame：
C [libpthread.so.0+0xa848] pthread_cond_timedwait+0x168
#
An error report file with more information is saved as hs_err_pid7668.log
#
If you would like to submit a bug report， please visit：
http：//java.sun.com/webapps/bugreport/crash.jsp
#
make： *** [out/target/common/obj/JAVA_LIBRARIES/core_intermediates/classes.dex] 已放弃 （core dumped）
解决：
虚拟机给的内存 512 太小，给个 1G 试试.然后在本机里打开任务管理器，找到虚拟机进程，优先级设置为实时.给他最多东西.
错误 5：
out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/libcutils.a （ threads.o ）： In function `thread_store_get'：
/home/leno/works/android_dev/bin/system/core/libcutils/threads.c：27： undefined reference to `pthread_getspecific'
out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/libcutils.a （ threads.o ）： In function `thread_store_set'：
/home/leno/works/android_dev/bin/system/core/libcutils/threads.c：36： undefined reference to `pthread_key_create'
/home/leno/works/android_dev/bin/system/core/libcutils/threads.c：44： undefined reference to `pthread_setspecific'
collect2： ld returned 1 exit status
make： *** [out/host/linux-x86/obj/EXECUTABLES/localize_intermediates/localize] 错误 1
解决：类似问题 3
修改./framework/base/tools/localize/Android.mk 文件
ifeq （$（HOST_OS），linux）
#LOCAL_LDLIBS += -lrt 把这行注释掉，改为下面一行.
LOCAL_LDLIBS += -lrt -lpthread
endif
编译 SDK
执行$sudo make PRODUCT-sdk-sdk 命令，生成对应于该版本源代码的用于生产环境的 sdk.
编译的 SDK 版本，实际位置是./out/host/linux-x86/sdk/android-sdk_eng.root_linux-x86.
因为它比较常用，我们给它高优先级：
update-alternatives --install /usr/bin/AndroidSDK AndroidSDK ./out/host/linux-x86/sdk/android-sdk_eng.root_linux-x86 255
然后使用 update-alternatives --display AndroidSDK 查看当前配置情况；

如果要切换配置，使用 update-alternatives --config AndroidSDK.
配置 AndroidSDK 环境变量.终端中执行 gedit ~/.bashrc
在文件最后添加下面三行：
set android environment
export ANDROID_SDK_HOME=/usr/bin/AndroidSDK
export PATH=$ANDROID_SDK_HOME/tools：$PATH
保存文件.在终端中执行 source ~/.bashrc
在模拟器中运行编译的镜像
打开终端，执行 android 脚本：
$android
选择左边第一项 Virtual Devices，然后在右边选择 New，新建一个 AVD.
Name：AVD 的名称，随便取，但只能包含字母和数字以及点、下划线和连字符，这里取名 test.
Traget：目标 SDK 版本，这里选自己编译的 2.1-r1 版 Android SDK.
SD Card：SD 卡.暂且不填，待需要时再设置.
Skin：皮肤（模拟器屏幕分辨率）.
Hardware：使用默认即可
然后点 Create AVD，就创建好了一个 AVD.点击 Start 即
可启动模拟器，其运行的是自己编译的镜像.

5. Android源码编译出错的原因

由于 android 源码编译要求为 4.3，如果你的 gcc 版本为 4.4，那你的编译可能会失败的！我的系统是 ubuntu 10.04，默认的 gcc 版本为 4.4，gcc-4.4 太严格，那么怎样从 gcc-4.4 降到 gcc-4.3 呢？

1、安装 gcc-4.3
$ sudo apt-get install gcc-4.3 g++-4.3
2、修 gcc 相关链接
$ cd /usr/bin
$sudo ln -snf gcc-4.3 gcc
$sudo ln -snf g++-4.3 g++
$sudo ln -snf cpp-4.3 cpp
3、如是 64bit 系统，则还需装如下包.
$ sudo apt-get install g++-multilib g++-4.3-multilib
注：Android1.6 用 gcc-4.4.3 编译时会出现如下错误.
error.o out/host/linux-x86/obj/STATIC_LIBRARIES/libtinyxml_intermediates/tinystr.o
/usr/bin/ld： skipping incompatible /usr/lib/gcc/x86_64-linux-gnu/4.3.4/libstdc++.so when searching for -lstdc++
/usr/bin/ld： skipping incompatible /usr/lib/gcc/x86_64-linux-gnu/4.3.4/libstdc++.a when searching for -lstdc++
/usr/bin/ld： skipping incompatible /usr/lib/gcc/x86_64-linux-gnu/4.3.4/libstdc++.so when searching for -lstdc++
/usr/bin/ld： skipping incompatible /usr/lib/gcc/x86_64-linux-gnu/4.3.4/libstdc++.a when searching for -lstdc++
/usr/bin/ld： cannot find -lstdc++
这时只要照我上面操作，将 gcc-4.4.3 降到 gcc-4.3 即可.
经过验证，即使降到 gcc-4.3 后 android2.1 或高通（android）的其它代码编译也均正常！！！
（2）android 源码编译对 jdk 也有特定的要求，一般情况下在 jdk1.5 下可以成功编译！1.6 下好像编译出错，那么怎样 ubuntu9.10 版本及以上版本安装 jdk1.5 呢？
安装 jdk-1.5， 在 ubuntu 9.10 上默认的是 jdk1.6， 所以用 sudo apt-get install sun-java5-jdk 会提示找不到相

应的软件包

解决办法： 将 9.10 的源替换成 9.04 的源

sudo -i

cd /etc/apt/

cp sources.list sources.list_bak

gedit sources.list

将该文件的内容全部替换成如下：

deb http：//run.hit.edu.cn/ubuntu/ jaunty main restricted universe multiverse

deb-src http：//run.hit.edu.cn/ubuntu/ jaunty main restricted universe multiverse

deb http：//run.hit.edu.cn/ubuntu/ jaunty-updates main restricted universe multiverse

deb-src http：//run.hit.edu.cn/ubuntu/ jaunty-updates main restricted universe multiverse

deb http：//run.hit.edu.cn/ubuntu/ jaunty-backports main restricted universe multiverse

deb-src http：//run.hit.edu.cn/ubuntu/ jaunty-backports main restricted universe multiverse

deb http：//run.hit.edu.cn/ubuntu/ jaunty-security main restricted universe multiverse

deb-src http：//run.hit.edu.cn/ubuntu/ jaunty-security main restricted universe multiverse

apt-get update

apt-get install sun-java5-jdk

至此，jdk1.5 就安装完成了.

收尾工作，

1，将 sources.list 还原回来 ：

mv sources.list_bak sources.list

apt-get update

如果你先前安装了 jdk1.6，卸载吧：

apt-get remove sun-java6-jdk sun-java6-jre sun-java6-bin

如果是 ubuntu10.04 的话，请在新立得软件包管理器删除关于 openjdk 的包

6. Android 源码开发环境及源码编译

下面是重点

如何在 Ubuntu 上构建 Android 源码开发环境

Android 核心编译技术

快速开发系统级应用程序

添加应用程序到 Android 源码中

如何构建和定制 Android 桌面（Home）

我们要在这里学到

如何安装基于 Ubuntu 上的 Android 源码开发环境

如何下载 Android 源码

初级 Android 源码编译技术

如何结合 Eclipse 进行源码修改

快速克隆和构建系统级应用程序

Android 模块编译和应用程序编译技术

往 Android 源码环境中加入自己的程序
编译后的 Android 产物目录及结构
构建简单的 HOME 程序
定制客制化的 Android 桌面

Android 编译代码环境

关于 Android 的源码编译环境，官方推荐在 Ubuntu 下进行搭建，从 Ubuntu 6.06 或之后的版本，Android 编译都经过了测试，某些较新的 Ubuntu 发行版本可能存在一定问题，本书采用 Ubuntu9.10 进行 Android 源码开发环境的搭建和编译。由于 Android 源码编译比较费时和消耗 CPU 及内存，建议您的机器配置 2G 及以上内存，至少大于 20G 的硬盘存储，一些好的电脑配置，可以采用 VMWare 或是 VirtualBox 进行安装，普通配置的电脑，采用硬盘安装的方式，这样加快您电脑编译时候的速度,不至于电脑卡住或死机。此章节推荐您使用 Wubi 或是 UNetbootin 来快速安装 Ubuntu 系统。

Wubi 的特性：

- 简易：不需要刻录光盘、直接运行安装器，键入密码创建一个新用户即可，一盏茶功夫就安装好。
- 安全：保持您的 Windows 系统不变，仅添加 Ubuntu 启动的额外选项。不需要修改硬盘分区，可使用其他启动器，不需要安装驱动，工作方式类似一个应用程序。Wubi 远离病毒和恶意软件，并且开源。
- 独立：保持多文件保存到一个文件夹，如果不喜欢，可以像普通应用程序一样卸载。
- 免费：Wubi 和 Ubuntu 完全免费，提供了艺术性、全功能的操作系统。免费、自由。

1.下载 Wubi：

您可以从 Wubi 官方下载最新版本的 Wubi。

官方地址：

http://wubi-installer.org/

下载地址：

http://releases.ubuntu.com/10.04/wubi.exe

2.安装 Ubuntu：

点击下载好的 wubi.exe

图 Wubi

运行 Wubi 安装器，如图：

图 使用 Wubi 快速安装 Ubuntu

按照上图所示，可以设置基本的安装设置：

- 目标驱动器：安装 Ubuntu 的硬盘
- 安装大小：分配给 Ubuntu 的硬盘大小
- 桌面环境：备选的可用 Ubuntu 系统
- 语言：Ubuntu 的系统语言
- 用户名：创建登陆用户
- 口令：登陆密码

配置好安装设置后，点击“安装”。然后等待下载 Ubuntu，Wubi 会帮您自动安装好，安装成功后按照提示，直接重启计算机，在启动界面会有 Ubuntu 启动选项，选择 Ubuntu 并启动。

使用 UNetbootin安装 Ubuntu

UNetbootin 是一个 Live USB Drive 制作工具，他可以创建一个可以启动的 Live USB 启动。 它可以自动加载 ISO 启动文件，可以在线下载 ISO 或是使用本地 ISO 光盘镜像，通常用它来使用系统 ISO 镜像、软盘、硬盘镜像、

内核或启动文件创建 USB 系统，直接把系统镜像安装在 USB 设备上，然后使用 USB 设备启动 Live 系统或是利用 USB 设备替代光盘进行安装系统。

1.下载 UNetbootin,从官方网站下载：

官方网址：

http://unetbootin.sourceforge.net/

下载地址：

Windows 版本

http://downloads.sourceforge.net/project/unetbootin/UNetbootin/442/unetbootin-windows-442.exe?use_mirror=cdnetworks-kr-1

Linux 版本

http://cdnetworks-kr-2.dl.sourceforge.net/project/unetbootin/UNetbootin/442/unetbootin-linux-442

2.下载 Ubuntu9.10 CD ISO 光盘镜像

Ubuntu 官方网址：

http://www.ubuntu.com/

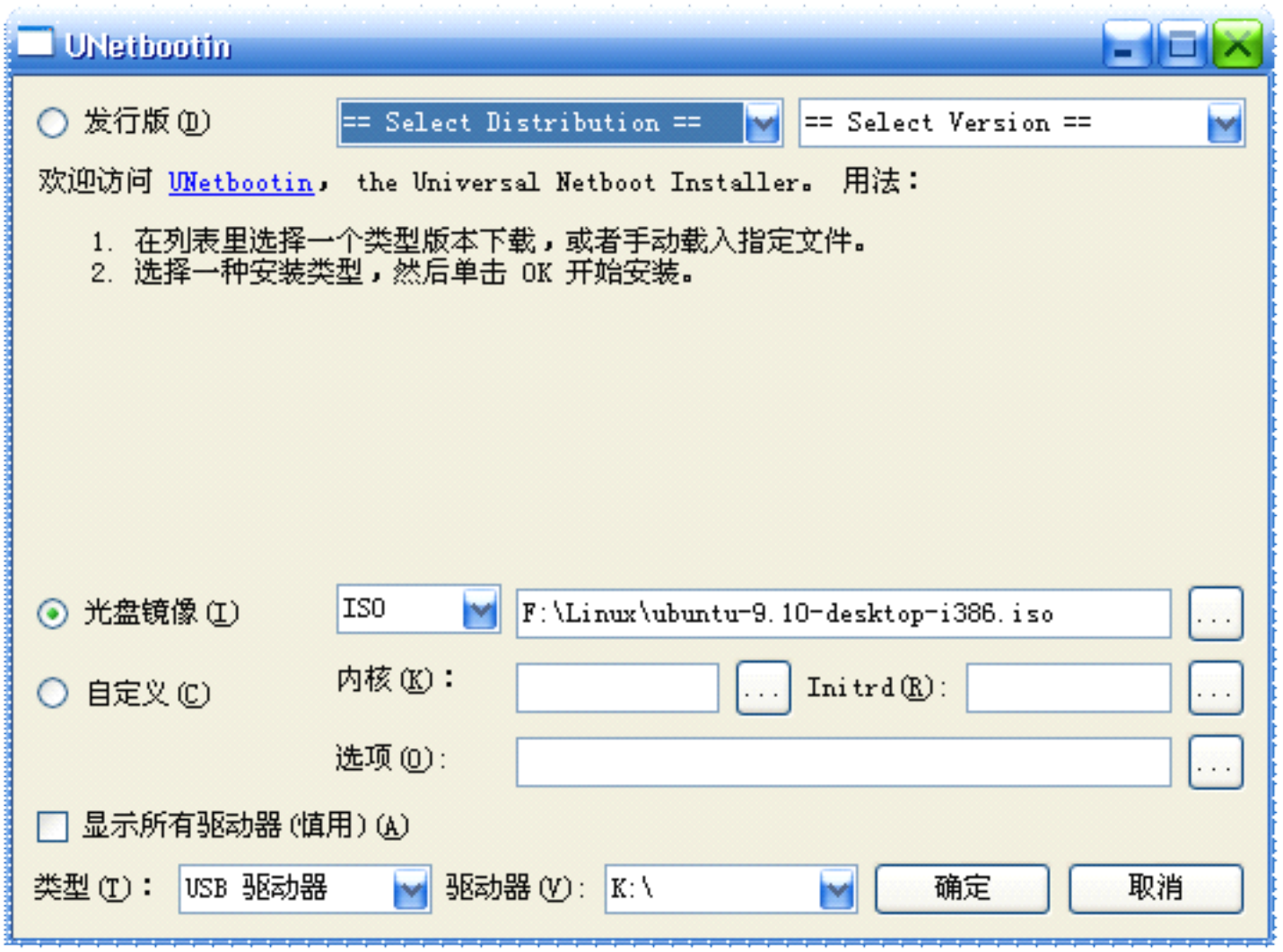
Ubuntu9.10 CD ISO 下载地址：

http://mirrors.cat.pdx.edu/ubuntu-releases/9.10/ubuntu-9.10-desktop-i386.iso

下载 Ubuntu 9.10 CD ISO 光盘镜像，放到任意的方便目录下。

3.制作 USB Live 系统

插入 USB 设备，例如 U 盘等，打开 UNetbootin，选择使用本地系统 ISO 镜像，选中刚刚下载的 Ubuntu 9.10 CD ISO 光盘镜像，然后选择刚刚插入的 USB 设备驱动器，点击确定。待镜像文件都安装到了 USB 上，点击重启。



图使用 UNetbootin 制作 USB 启动系统

4.修改 BIOS,使用 USB 启动

在开机时候，点击 Del 键，打开电脑 Bios 界面，选择 Boot，然后选择启动优先级，设置 USB 启动为第一优先级，然后按 F10 键，保存设置，然后重启。重启后，出现 Ubuntu 安装界面，其安装过程不再复述，可参考官方网站的安装流程。

7. Android 编译完成后的代码结构

通常，在 Android 源码根目录下，在命令行中使用：

```
$ make
```

此命令会完整的编译 Android 各个项目和包，然后在 Android 源码根目录中生成 out 目录，存放编译结果内容。

out 目录

Android 编译完成后，将在根目录中生成一个 out 文件夹，所有生成的内容均放置在这个文件夹中。out 文件夹如下所示：

```
<Android-src>
    |
    out/
        |--> CaseCheck.txt
        |--> casecheck.txt
        |--> versions_checked.mk
        |--> temp
        |--> host
        |      |--> common
        |       `--> linux-x86
        `--> target
               |--> common
               `--> product
```

核心目录：

host：存放主机（x86）生成的工具
target：存放目标机（模认：ARMv5）运行的内容

其他：

- temp：临时文件目录
- casecheck.txt
- CaseCheck.txt
- versions_checked.mk：版本检测 make 文件

host 目录

host 中的工具可以在主机上直接使用，包括一些二进制工具和 Java 程序。以下是 host 目录结构：

```
out/host/

|--> common

|   `--> obj

`--> linux-x86

    |--> bin

    |--> framework

    |--> lib

    `--> obj
```

核心目录：

- common：存放公共组件
- obj：存放 Java 库
- linux-x86：存放主机相关工具和库
- bin：存放 linux-x86 Android 开发调试工具
- framework：Android 框架类库
- lib：共享库，底层 so 库
- obj：存放中间生成目标文件

target 目录

target 目录是和目标机相关的目录，里面存放了共用组件，中间目标文件，Android 文档和产品。

target 目录结构如下：

```
out/target/

|--> common

|   |--> R
```

```
|   |--> docs
|   |--> cts
|   `--> obj
|             |--> APPS
|             |--> JAVA_LIBRARIES
|               `--> PACKAGING
`--> product
        `--> generic
```

核心目录：

common：存放公共组件
product：存放产品产物

其他：

R：存放资源引用类库
docs：存放 Android 文档
obj：存放 Java 库
generic:目标机相关产品目录
APPS：JAVA 应用程序生成的目标目录，和应用程序相对应，里面存有其 Android 应用程序 APK 包。
JAVA_LIBRARIES：存放 Java 类库，和库中的结构一 一对应。
PACKAGING：存放公共的 API 打包信息

generic 目录

generic 是对应产品名为 generic 的产品目录，其中包括了编译后 generic 产品所有的产物，一般会包括常用的系统镜像，系统默认自带的内置程序，系统数据，系统工具，Android 框架库和资源等等。

以下是 generic 的目录结构：

```
out/target/product/generic/
|--> android-info.txt
|--> clean_steps.mk
```

```
|--> data

|--> obj

|--> ramdisk.img

|--> root

|--> symbols

|--> system

|--> system.img

|--> userdata-qemu.img

`--> userdata.img
```

核心目录：

data:存放 Android 系统数据
obj：存放中间产物，包括 Android 共享库，静态库，inlude 库，打包信息，应用程序的打包 apk，未签名版本。
root：Android 系统 root 目录
symbols：存放 Android 系统的符号库，包括 init 程序
system：存放 Android 系统相关资源，app 应用程序，工具，配置，字体，框架，常用库，用户目录，Xbin 等。

核心文件：

ramdisk.img：内存分区文件系统镜像
system.img：系统分区文件系统镜像
data.img:数据分区内容镜像
userdata-qemu.img：用户 Android 模拟器仿真系统数据内容镜像
userdata.img：用户数据分区内容镜像

8. 构建自己的 Android 桌面

目前，各大手机厂商都在定制自己的 Android MID 设备或是手持设备，当然，要让用户真正记住自己的产品，在感官上就需要和普通的 Android 原生系统 UI 和功能有所差异化，这样就需要定制 HOME。目前已经有很成功的 HOME 定制产品，例如 HTC 的 HTC Sense UI，起初搭载到了 HTC Hero 上，后来生产的 Tattoo,Sprint Hero 以及 Eris/Desire 都使用了 Sense UI。其他还有很多成功定制的 Android 桌面，以下是目前比较成功的定制桌面及开发商：

定制桌面 UI 名称	开发商
HTC Sense UI	HTCMotorola

MOTOBLUR UI	Motorola
Lephone UI	Lenovo
TouchWiz UIA	Samsung
Archos UI	Archos
Ophone UI	China Mobile

表 14-4-1 目前优秀的 Android 定制 Home 及其厂商

那么对于小厂商或是个人爱好者来说，他们也希望定制适合自己特色和爱好的 Android 桌面，Android 开源项目让定制 Android 桌面成为可能，下面介绍一下两种级别的 HOME 定制，一种是自己重写的普通应用程序类别的 Android 桌面，另一种是基于 Android Launcher 桌面的定制 HOME 桌面。对于一个 Android 桌面，至少需要提供以下功能：

1. 设置壁纸
2. 显示系统安装应用图标
3. 提供应用程序入口
4. 提供手机设置入口
5. 桌面切换
6. 添加 widget
7. 添加快捷方式
8. 添加 folders

那我们具体来看一下如何定制自己的 Android 桌面。 9

9. 构建普通的 HOME 类型应用程序

普通的 HOME 类型的应用程序，这种程序不需要在 Android 源码环境编译，可以直接在 Eclipse 中，类似于其他应用程序的开发和编译，也可以定制成为 HOME 程序，但是从功能上讲，这样的 Home 不能达到强大的交互效果，也不能提供用户更好地桌面体验。

1.在 Eclipse 中新建 Android 桌面项目 MyHome，配置信息如下：

Project Name	MyHome
Target Name	Android 2.0 或是 Android 2.1
Application name	MyHome
Package name	com.demo.home
Create Activity	MyHome
Min SDK Version	不填

表 14-4-2 新建 MyHome 项目信息

2.修改 AndroidManifest.xml 项目配置文件：

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

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

  package="com.demo.home" android:versionCode="1"
android:versionName="1.0">

  <application android:icon="@drawable/icon"
android:label="@string/app_name">

    <activity android:name=".MyHome" android:label="@string/app_name"

        android:theme="@android:style/Theme.Wallpaper.NoTitleBar">

         <intent-filter>

              <action android:name="android.intent.action.MAIN"/>

              <category android:name="android.intent.category.DEFAULT"/>

              <category android:name="android.intent.category.HOME"/>

         </intent-filter>

    </activity>

    </application>

</manifest>
```

解释：

- android:theme="@android:style/Theme.Wallpaper.NoTitleBar"：说明这个应用程序的风格样式主题，没有标题栏，并且可以直接看到壁纸，壁纸就是此页面（Activity）的背景。
- <action android:name="android.intent.action.MAIN"/>：说明此程序的 Activity 可以没有任何 Data 参数而直接启动，此 Activity 是主入口。
- <category android:name="android.intent.category.HOME"/>：说明此 Activity 的类型是属于 Home 类型

当然，这个 intent filter （Intent 过滤器）的 Action 是 Intent.ACTION_MAIN,过滤的 Intent 需要带有 Intent.CATEGORY_HOME 和 Intent.CATEGORY_DEFAULT 类型才能匹配，然后系统的 Activity Chooser 才能列出符合此 Intent 过滤器的 Activity。

这样，这个应用程序变成了 HOME 类型的桌面，当然，目前没有任何桌面的功能，需要您具体实现，当我们安装这个程序后，点击 Android 手持设备或是模拟器的 HOME 键，会列出默认的 HOME 桌面和我们定制的 Demo 桌面程序，具体的桌面功能需要您具体实现。

10. 定制自己的 Android HOME 桌面

为了定制功能更强大，性能更稳定的 Android 桌面，我们可以从 Android Launcher 来衍生出一个桌面项目，基于 Android Launcher 来定制桌面，这样会加快开发流程，以及提高代码重用的效果。定制一个优秀的 Android 桌面，需要结合人体交互习惯，艺术设计，人机交互的多种学科，这样设计出的 Android 桌面才能更加迎合客户的需求，满足更具有美感的体验。

在 Android 完整源码中加入自己的 Package

1.Home 来源

通过 Eclipse Android DEV 环境在<android-src-home>/packages/apps 目录新建一个 Android 项目：

Project Name	ZhangYunfang_Home
Location	<android-src-home>/packages/apps/ZhangYunfang_Home
Build Target	Android 2.1
Application Name	ZhangYunfang Home
Package Name	com.zhangyunfang.android.home
Create Activity	不选
Min SDK Version	不填

表 14-4-2 Clone Launcher 项目，新建 HOME 项目信息

先删除 ZhangYunfang_Home 下的 res 目录。然后拷贝 Android Launcher 里面（覆盖并替换）所有内容放到 ZhangYunfang_Home 中。

2.重构成自己的 Home

把 com.android.launcher 里面所有类拷贝到 com.zhangyunfang.android.home 中。

修改所有 com.android.launcher 地方相关的，改成 com.zhangyunfang.android.home 。然后进行调整重构。但注意 ContentProvider 的子类的 AUTHORITY 值，这个在 Android 系统中是唯一的，需要重点注意。

3.修正 Android.mk 文件

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

include $(CLEAR_VARS)

#LOCAL_MODULE_TAGS := user

LOCAL_SRC_FILES := $(call all-subdir-java-files)
```

```
LOCAL_PACKAGE_NAME := ZhangYunfang_Home

LOCAL_CERTIFICATE := zhangyunfang

LOCAL_OVERRIDES_PACKAGES := Home 群讨论(0)

您目前还没有群留言，请从右侧"我的群"中选择群发起聊天

include $(BUILD_PACKAGE)
```

建立自己的 Build System

1.创建 Build 脚本

建立一个自己产品发布时候签名的 System Key，这个非常重要和有用，防止盗版/版权/反编译等等问题。

在 ZhangYunfang_Home 根目录下面建立 build 目录，在 ZhangYunfang_Home 根目录创建 build.sh 脚本：

```
#!/bin/bash

# setup path

CURRENT_PATH=`pwd`

cd `dirname $0`

ZHANGYUNFANGHOME_PATH=`pwd`

cd "${ZHANGYUNFANGHOME_PATH}/../../../"

ANDROID_PATH=`pwd`

PRODUCT_PATH="${ANDROID_PATH}/build/target/product"

SECURITY_PATH="${PRODUCT_PATH}/security/"

INTERMEDIATE_DIR="${ANDROID_PATH}/out/target/product/generic/obj/APPS/ZhangYunfang_Home_intern

# verify path

if [ ! -d "${ANDROID_PATH}" ] || [ ! -d "${PRODUCT_PATH}" ] || [ ! -d "${SECURITY_PATH}" ] || [ ! -f
"${PRODUCT_PATH}/AndroidProducts.mk" ]; then
```

```
    echo "Please put ZhangYunfang_Home project and this script to currect path"
    exit 1
fi
# verify env
if [ ! -f "${PRODUCT_PATH}/zhangyunfang_home.mk" ]; then
    if [ -f "${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang_home.mk" ]; then
        echo "Copy zhangyunfang_home.mk to product setting dir"
        cp -v "${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang_home.mk"
"${PRODUCT_PATH}/zhangyunfang_home.mk"
    else
        echo "Miss the file ${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang_home.mk"
        exit 1
    fi
fi
if [ ! -f "${SECURITY_PATH}/zhangyunfang.pk8" ]; then
    if [ -f "${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang.pk8" ]; then
        echo "Copy zhangyunfang.pk8 to security dir"
        cp -v "${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang.pk8" "${SECURITY_PATH}/zhangyunfang.
    else
        echo "Miss the file ${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang.pk8"
        exit 1
    fi
fi
```

```
if [ ! -f "${SECURITY_PATH}/zhangyunfang.x509.pem" ]; then
    if [ -f "${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang.x509.pem" ]; then
        echo "Copy zhangyunfang.x509.pem to security dir"
        cp -v "${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang.x509.pem"
"${SECURITY_PATH}/zhangyunfang.x509.pem"
    else
        echo "Miss the file ${ZHANGYUNFANGHOME_PATH}/build/zhangyunfang.x509.pem"
        exit 1
    fi
fi
GREP_RESULT=`grep "\\\$(LOCAL_DIR)/zhangyunfang_home.mk" "${PRODUCT_PATH}/AndroidProducts.mk
if [ "${GREP_RESULT}" = "" ]; then
    echo "AndroidProducts.mk not contain ZhangYunfang_Home product"
    echo "Add porduct setting to AndroidProducts.mk"
    sed 's/.mk/.mk \\/;s/.mk \\ \\/.mk S/;s:.mk \\:.mk \\\n    \$(LOCAL_DIR)/zhangyunfang_home.mk:;s/.m
\\/' "${PRODUCT_PATH}/AndroidProducts.mk" > "${PRODUCT_PATH}/AndroidProducts.mk.new"
    rm -rf "${PRODUCT_PATH}/AndroidProducts.mk"
    mv "${PRODUCT_PATH}/AndroidProducts.mk.new" "${PRODUCT_PATH}/AndroidProducts.mk"
fi
# set android env
. "${ANDROID_PATH}/build/envsetup.sh"
chooseproduct zhangyunfang_home
# clean
echo "Clean ..."
```

```
rm -rv "${ZHANGYUNFANGHOME_PATH}/bin" "${ZHANGYUNFANGHOME_PATH}/gen"
"${ZHANGYUNFANGHOME_PATH}/ZhangYunfang_Home.apk"

# build

echo "Start build ..."

if mmm; then

    echo "Build success!"

    if [ -f "${INTERMEDIATE_DIR}/package.apk" ]; then

        cp -v "${INTERMEDIATE_DIR}/package.apk" "${ZHANGYUNFANGHOME_PATH}/ZhangYunfang_Home.a

    fi

    if [ -d "/mnt/hgfs/WindowsFolder" ] && [ -f "${ZHANGYUNFANGHOME_PATH}/ZhangYunfang_Home.apk" ]

        cp -v "${ZHANGYUNFANGHOME_PATH}/ZhangYunfang_Home.apk" "/mnt/hgfs/WindowsFolder/"

    fi

else

    echo "Build failed!"

fi
```

2.创建发布 System Key

在 ZhangYunfang_Home 目录下的 build 目录中，创建 key.sh 脚本：

```
#!/bin/bash

openssl genrsa -3 -out $1.pem 2048

openssl req -new -x509 -key $1.pem -out $1.x509.pem -days 20000 \

    -subj '/C=CN/ST=Bei Jing/L=Bei
Jing/O=Zhangyunfang/OU=Zhangyunfang/CN=Zhangyunfang/emailAddress=zhangyunfang@eoemobile.com'

openssl pkcs8 -in $1.pem -topk8 -outform DER -out $1.pk8 -nocrypt
```

C=国家

ST=地区

L=地方

O=组织

OU=组织

CN=创建者

emailAddress=邮件地址

创建 key：

```
$ ./key.sh zhangyunfang
```

这样就创建了系统 System app 用的私人 key 了。

3.创建项目 mk Make 文件

在 build 目录中创建 zhangyunfang_home.mk：

```
# This is a generic product that isn't specialized for a specific device.

# It includes the base Android platform. If you need Google-specific features,

# you should derive from generic_with_google.mk

PRODUCT_PACKAGES := \

    framework-res \

    ZhangYunfang_Home

#$(call inherit-product, $(SRC_TARGET_DIR)/product/core.mk)

PRODUCT_POLICY := android.policy_phone

PRODUCT_PROPERTY_OVERRIDES := \

    ro.config.notification_sound=OnTheHunt.ogg \

    ro.config.alarm_alert=Alarm_Classic.ogg

# Overrides

PRODUCT_BRAND := generic
```

```
PRODUCT_DEVICE := generic

PRODUCT_NAME := zhangyunfang_home

PRODUCT_LOCALES := \

    ldpi \

    hdpi \

    mdpi \

    en_US \

    en_GB \

    zh_CN \

    zh_TW
```

【附录】

1 提交 BUG

亲爱的开发者，如果您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。感谢您对 eoe 的支持！我们的进步离不开大家的互勉！

2 关于 eoeAndroid

eoeAndroid 是国内成立最早，规模最大的 Android 开发者社区，拥有海量的 Android 学习资料。分享、互助的氛围，让 Android 开发者迅速成长，逐步从懵懂到了解，从入门到开发出属于自己的应用，eoeAndroid 为广大 Android 开发者奠定坚实的技术基础。从初级到高级，从环境搭建到底层架构，eoeAndroid 社区为开发者精挑细选了丰富的学习资料和实例代码。让 Android 开发者在社区中迅速的成长，并在此基础上开发出更多优秀的 Android 应用。

北京易联致远无线技术有限公司

责任编辑：果子狸

美术支持：金明根　　**技术支持：**&麦#兜

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