

Android 系统的开发综述

Android 系统的开发综述

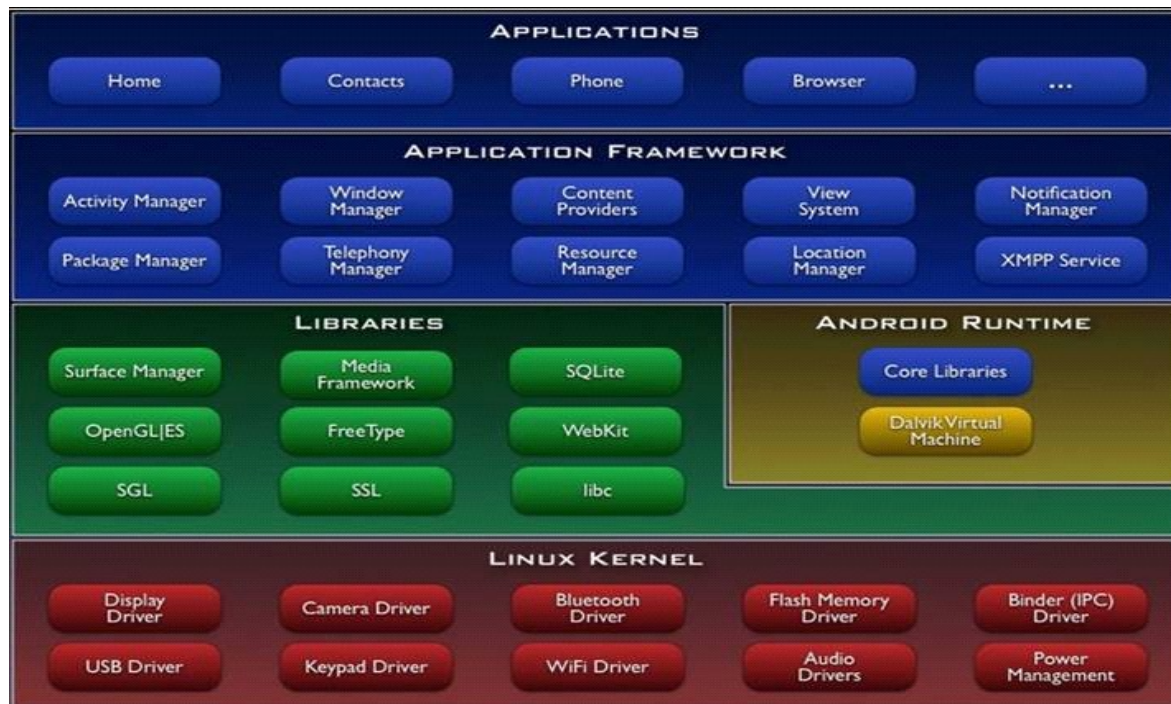
- 第一部分 **Android 的系统架构**
- 第二部分 **Android 源代码的开发环境**

第一部分 Android 的系统架构

1.1 软件结构

1.2 Android 的工具

1.1 软件结构



第四层:
Java应用程序

第三层:
Java框架

第二层:
本地框架和Java运行环境

第一层:
Linux操作系统及驱动

1.1 软件结构

Android 的软件结构的几个层次:

1. 操作系统层 (Linux, 相关驱动)
2. 库 (Libraries) 和运行环境 (RunTime)
3. 应用程序框架 (Application Framework)
4. 应用程序 (Application)

操作系统层使用 C 语言编写, 运行于内核空间。

底层库和 JAVA 虚拟机使用 C 语言编写, 运行于用户空间。

JAVA 框架和 JAVA 应用程序使用 C 语言编写, 运行于用户空间。

1.1 软件结构

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

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

第 2 层次和第 3 层次之间，是本地代码层和 Java 代码层的接口。

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

1.1 软件结构

Linux 操作系统和驱动:

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

1.1 软件结构

Android 本地框架（C/C++）：

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

- ❑ 系统 C 库：一个从 BSD 继承来的标准 C 系统函数库，专门为基于嵌入式 linux 的设备定制的。
- ❑ 媒体库：基于 PacketVideo OpenCORE；该库支持多种常用的音频、视频格式回放和录制。
- ❑ Surface Manager：对显示子系统的管理，图层功能。
- ❑ WebCore：一个最新的 web 浏览器引擎用，支持 Android 浏览器和一个可嵌入的 web 视图。
- ❑ SGL：Skia 的 2D 图形引擎
- ❑ 3D libraries：基于 OpenGL 实现；该库可以使用硬件 3D 加速（如果可用）或者使用高度优化的 3D 软加速。
- ❑ FreeType：位图（bitmap）和矢量（vector）字体显示。

1.1 软件结构

Android 运行库：

Android 包括了一个核心库，该核心库提供了 JAVA 编程语言核心库的大多数功能。

每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 被设计成一个设备可以同时高效地运行多个虚拟系统。

Dalvik 虚拟机执行（.dex）的 Dalvik 可执行文件，该格式文件针对小内存使用做了优化。同时虚拟机是基于寄存器的，所有的类都经由 JAVA 编译器编译，然后通过 SDK 中的 "dx" 工具转化成 .dex 格式由虚拟机执行。

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

1.1 软件结构

Android 应用程序框架：

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

隐藏在每个应用后面的是一系列的服务和系统，其中包括；

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

1.1 软件结构

Android 应用程序：

Android 会同一系列核心应用程序包一起发布，这些应用程序包也就是预置的应用程序，主要包括 **email** 客户端，**SMS** 短消息程序，日历，地图，浏览器，联系人管理程序等。所有的应用程序都是使用 **JAVA** 语言编写的。

1.2 Android 的工具

aapt (**Android Asset Packaging Tool**)

用于建立 Zip 兼容的包 (zip, jar, apk) , 也可用于编译资源到二进制的 assets 。

abd (**Android Debug Bridge** , **Android 调试桥**)

使用 Adb 工具可以在模拟器或设备上安装应用程序的 .apk 文件, 并从命令行访问模拟器或设备。也可以用它把 Android 模拟器或设备上的应用程序代码和一个标准的调试器连接在一起。

```
$/out/host/linux-x86/bin/adb shell  
$/out/host/linux-x86/bin/adb install XXX.apk  
$/out/host/linux-x86/bin/adb push {host_path} {target_path}  
$/out/host/linux-x86/bin/adb pull {target_path} {host_path}
```

android 工具

一个脚本用于创建和管理 Android Virtual Devices (AVDs) 。

1.2 Android 的工具

AIDL 工具

（ **Android Interface Description Language** ,
Android 接口描述语言）

可以生成进程间的接口的代码，诸如 **service** 可能使用的接口。

AVDs （ **Android Virtual Devices** , **Android 虚拟设备**）

用于配置仿真器的选项，使用实际的设备。

DDMS

（ **Dalvik Debug Monitor Service** , **Dalvik 调试监视器服务**）

这个工具集成了 **Dalvik**，能够在模拟器或者设备上管理进程并协助调试。可以使用它杀死进程，选择某个特定的进程来调试，产生跟踪数据，观察堆（**heap**）和线程信息，截取模拟器或设备的屏幕画面，还有更多的功能。

1.2 Android 的工具

dx

Dx 工具将 .class 字节码（bytecode）转换为 Android 字节码（保存在 .dex 文件中）。

Draw 9-patch

Draw 9-patch 工具允许使用所见即所得（WYSIWYG）的编辑器轻松地创建 NinePatch 图形。

Emulator（模拟器）

它是在的计算机上运行的一个虚拟移动设备。可以使用模拟器来在一个实际的 Android 运行环境下设计，调试和测试的应用程序。

Hierarchy Viewer（层级观察器）

层级观察器工具允许调试和优化的用户界面。它用可视的方法把的视图（view）的布局层次展现出来，此外还给当前界面提供了一个具有像素栅格（grid）的放大镜观察器。

1.2 Android 的工具

mksdcard

帮助创建磁盘映像（**disk image**），可以在模拟器环境下使用磁盘映像来模拟外部存储卡（例如 **SD 卡**）。

Monkey

Monkey 是在模拟器上或设备上运行的一个小程序，它能够产生为随机的用户事件流，例如点击 (**click**)，触摸 (**touch**)，挥手 (**gestures**)，还有一系列的系统级事件。可以使用 **Monkey** 来给正在开发的程序做随机的，但可重复的压力测试。

sqlite3

sqlite3 工具能够方便地访问 **SQLite** 数据文件。

Traceview

这个工具可以将的 **Android** 应用程序产生的跟踪日志（**trace log**）转换为图形化的分析视图

第二部分 Android 源代码的开发环境

2.1 源代码结构

2.2 编译 Android

2.3 运行 Android

2.1 源代码结构

开发 **Android** 主机环境的需求:

- ❑ **Git** 工具
- ❑ **Repo** 工具
- ❑ **Java** 的 **JDK**
- ❑ 主机编译工具

Ubuntu :

```
$ sudo apt-get install git-core gnupg sun-java5-jdk  
flex bison gperf libSDL-dev libesd0-dev libwxgtk2.6-  
dev build-essential zip curl libncurses5-dev zlib1g-  
dev
```

2.1 源代码结构

获取 **Android** 完全的源代码：

初始化代码仓库：

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

获取代码：

```
$ repo sync
```

初始化指定的版本：

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

-b release-1.0

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

-b android-sdk-1.5_r2

2.1 源代码结构

同步单个工程代码：

```
$ repo sync {project_name}
```

获取一个工程的代码：

```
$ git clone git://android.git.kernel.org/ + project path
```

例如，获取通用内核的代码：

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

2.1 源代码结构

```
$ repo init -u git://android.git.kernel.org/platform/manifest.git
Getting repo ...
  from git://android.git.kernel.org/tools/repo.git
Getting manifest ...
  from git://android.git.kernel.org/platform/manifest.git
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-1.6_r1 -> origin/android-1.6_r1
* [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_r1 -> origin/android-sdk-1.6_r1
* [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]      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-1.6_r1 -> android-1.6_r1
* [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_r1 -> android-sdk-1.6_r1
From git://android.git.kernel.org/platform/manifest
* [new tag]          android-1.0 -> android-1.0
```

2.1 源代码结构

`repo init` 之后，将生成隐藏目录 `.repo`，其中文件 `.repo/manifest.xml` 为 `repo` 工程的描述文件，表示 `repo` 时包含的各个工程，其片段如下所示：

```
<project path="dalvik" name="platform/dalvik" />
<project path="development" name="platform/development" />
<project path="frameworks/base" name="platform/frameworks/base" />
```

`.repo/manifest.xml` 中的 `path` 表示工程获取后的路径（基于当前目录），`name` 表示工程的名称。

2.1 源代码结构

提交代码 **Android** 代码:

<https://review.source.android.com>

提交代码的流程:

```
$ repo start {branch_name} {project_name}
```

```
$ git add {file_path}
```

```
$ git commit {file_path} -m"comment"
```

```
$ repo upload {project_name}
```

提交的网页:

<https://review.source.android.com/#change,{change id.}>

2.1 源代码结构

Android 代码的工程分为三个部分：

- ❑ **核心工程 (Core Project)**

建立 Android 系统的基础，在根目录的各个文件夹中。

- ❑ **扩展工程 (External Project)**

使用其他开源项目扩展的功能，在 external 文件夹中。

- ❑ **包 (Package)**

提供 Android 的应用程序和服务，在 package 文件夹中。

2.1 源代码结构

核心工程:

bionic :	C 运行时支持 : libc,
libm, libdl, 动态 linker	
bootloader/legacy :	Bootloader 参考代码
Build :	Build 系统
dalvik :	Dalvik 虚拟机
development :	高层的开发和调试工具
frameworks/base :	Android 核心的框架库
frameworks/policies/base :	框架配置策略
hardware/libhardware :	硬件抽象层库
hardware/ril :	Radio interface layer
kernel :	Linux 内核
prebuilt :	对 Linux 和 Mac OS 编译
的二进制支持	
system/core :	最小化可启动的环境
system/extras :	底层调试和检查工具

2.1 源代码结构

扩展工程（1）：

aes	: Advanced Encryption Standard , 高级加密标准
apache-http	: (JAVA) Http 服务器
bison	: (主机) 自动生成语法分析器程序, 基本兼容 Yacc
bluez	: 蓝牙库
bsdiff	: (主机) 用于为二进制文件生成补丁
bzip2	: (主机 / 目标机) 压缩文件工具
clearsilver	: (主机) 模板语言, 包括 python, java, perl, c 的 lib 支持
dbus	: freedesktop 下开源的 Linux IPC 通信机制
dhcpcd	: 动态主机设定协定 的工具
dropbear	: ssh2 服务器和客户端
e2fsprogs	: (主机) Ext2/3/4 文件系统的工具
elfcopy	: (主机) ELF 工具
elfutils	: (主机) ELF 工具

2.1 源代码结构

扩展工程（2）：

- embunit : 嵌入式 C 系统的测试框架
- emma : (JAVA) Java 代码覆盖检查工具
- esd : (仅头文件)
- expat : (主机 / 目标机) XML Parser
- fdlibm : 精确实现 IEEE754 浮点数
- freetype : C 语言实现的字体光栅化引擎制作的的一个软件库。
- gdata : (JAVA) 用于数据操作
- genext2fs : (主机) Ext2 文件系统生成工具
- giflib : GIF 工具
- googleclient : (JAVA) Google 客户端
- grub : 多重操作系统启动管理器
- icu4c : IBM 的支持软件国际化的开源项目
- iptables : 建构在 Xtables 的架构下, 定义“表 (tables)”、“键 (chain)”、“规则 (rules)”三个资料来处理封包的运送。

2.1 源代码结构

扩展工程（3）：

jdifff	: （主机 JAVA 库）比较工具
jhead	: Jpeg 文件头（Exif）编辑修改软件
jpeg	: Jpeg 工具库
libffi	: a portable foreign function interface library
libpcap	: 网络数据包捕获函数包
libpng	: PNG 工具库
libxml2	: （主机 / 目标机）C 语言的 XML 解析库
netcat	: 用来对网路连线 TCP 或者 UDP 进行读写
netperf	: 网络性能的测量工具
neven	: 人脸识别的一套库
opencore	: 多媒体框架
openssl	: C 语言的 SSL（Secure Sockets Layer）工具
oprofile	: Linux 内核支持的一种性能分析机制
ping	: ping 工具
ppp	: ppp 工具
protobuf	: Google 工具，利用 .proto 文件自动生成代码

2.1 源代码结构

扩展工程（4）：

qemu	： （主机） 仿真环境
safe-iop	： 夸平台的整数运算
skia	： 一个图形库
sonivox	： Sonic 嵌入式的音乐合成器
sqlite	： 轻量级的 SQL 嵌入式数据库
srec	： （主机 / 目标机） motorola S-records 16 进制文件格式
工具	
strace	： 监视系统调用的工具
tagsoup	： （ JAVA ） HTML 解析工具
tcpdump	： 网络中传送的数据包的头完全截获下来提供分析的工具
tinyxml	： （主机 / 目标机） XML 工具
tremor	： Ogg Vorbis 的播放器
webkit	： 开源的浏览器引擎
wpa_supplicant	： 无线局域网 Wifi 的工具
xdelta3	： （主机 / 目标机） 二进制文件比较工具
yaffs2	： （主机） YAFFS 文件系统

2.1 源代码结构

包（包括应用程序，提供者和输入法）

:

Applications ([package/apps](#))

AlarmClock , Browser , Calculator , Calendar , Camera
Contacts , Email , GoogleSearch , HTML Viewer , IM
Launcher , Mms , Music , PackageInstaller , Phone
Settings , SoundRecorder , Stk , Sync , Updater ,
VoiceDialer

Providers ([package/Providers](#))

CalendarProvider , ContactsProvider , DownloadProvider
DrmProvider , GoogleContactsProvider ,
GoogleSubscribedFeedsProvider ,
ImProvider , MediaProvider , SettingsProvider ,
SubscribedFeedsProvider , TelephonyProvider

2.2 Android 的编译

编译 **Android** 系统，在其根目录下中具有一个 **Makefile**，直接执行 **make** 即可。

```
$ make
```

make 的过程将递归找到各个目录中的 **Android.mk** 文件进行编译。

Android 的编译将搜索所有的目录，编译本身和目录的名称以及位置没有关系。

2.2 Android 的编译

Android 系统编译完成的结果全部在其根目录的 **out** 目录中，在其他目录中没有内容。

编译的结果：

- ❑ 主机工具
- ❑ 目标机程序
- ❑ 目标机映像文件
- ❑ 目标机 **Linux** 内核（需要单独处理）

```
out/target/product/generic/
|-- android-info.txt
|-- clean_steps.mk
|-- data
|-- installed-files.txt
|-- obj
|   |-- APPS
|   |-- ETC
|   |-- EXECUTABLES
|   |-- KEYCHARS
|   |-- NOTICE.html
|   |-- NOTICE.html.gz
|   |-- NOTICE_FILES
|   |-- PACKAGING
|   |-- SHARED_LIBRARIES
|   |-- STATIC_LIBRARIES
|   |-- include
|   |-- lib
|-- previous_build_config.mk
|-- ramdisk.img
|-- root
|-- symbols
|-- system
|-- system.img
|-- userdata-qemu.img
|-- userdata.img
```

[数据目录]

[中间目标文件目录]

JAVA 应用程序包

运行时配置文件

可执行程序

动态库（共享库）

静态库（归档文件）

根文件系统映像

[根文件系统目录]

[符号的目录]

[主文件系统目录]

主文件系统映像

为 QEMU 的数据映像

数据映像

2.3 运行 Android

Android 编译完成后可以在 QEMU 中运行，首先设置环境变量：

```
$ declare -x ANDROID_PRODUCT_OUT="{Android  
root}out/target/product/generic"
```

运行 Android：

```
$/out/host/linux-x86/bin/emulator -shell
```

按照这种方式运行后，在出现图形系统的同时，将会出现 Android 的 shell 界面。

默认皮肤为：**HVGA-P=320×480**。

2.3 运行 Android

Android Emulator

Android Emulator 基于 QEMU，这个仿真器支持 Android Virtual Device（Android 虚拟设备）以及很多的调试性能。

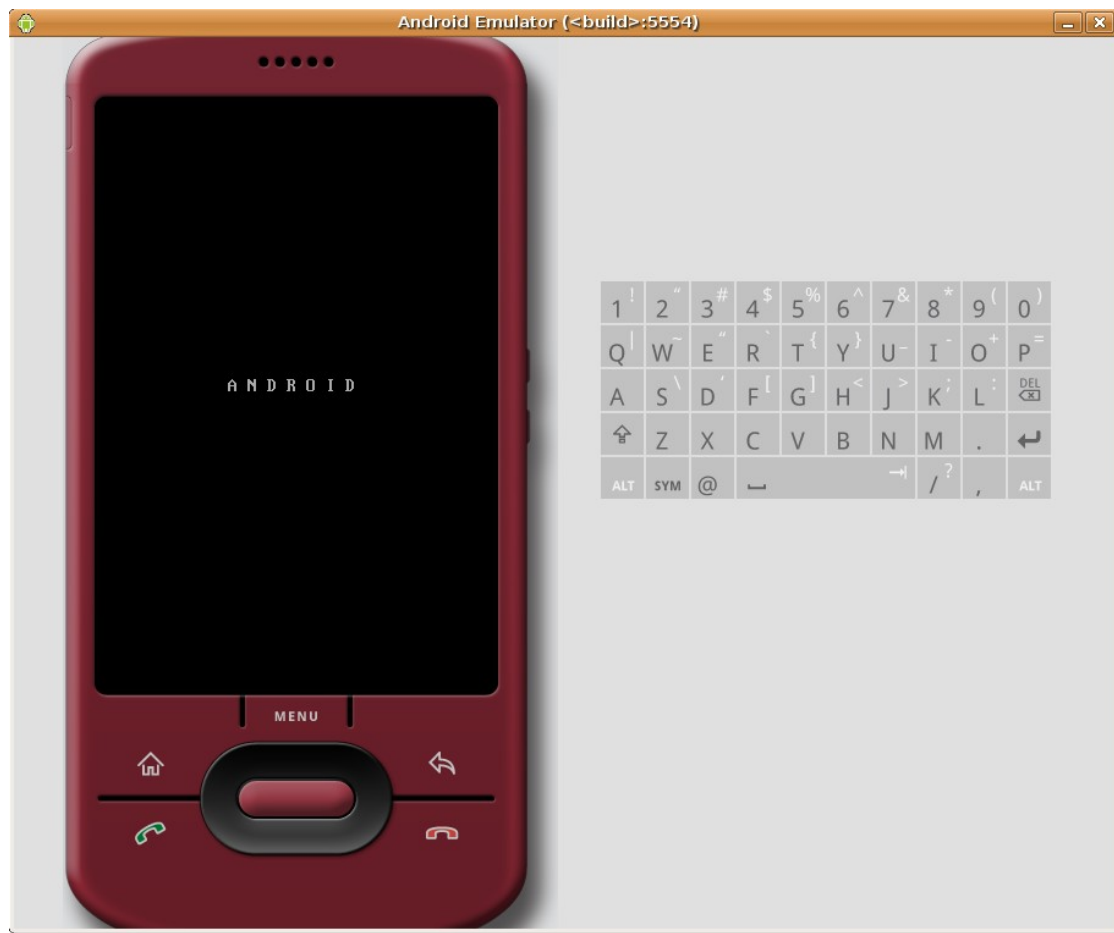
使用 Android Emulator 可以仿真 Android 整个系统运行，在运行的过程中，可以指定内核、主文件系统、用户文件系统等。

```
$ emulator -avd <avd_name> [-<option> [<value>]] ... [-<qemu args>]
```

默认使用的内核是：

prebuilt/android-arm/kernel/kernel-qemu

2.3 运行 Android



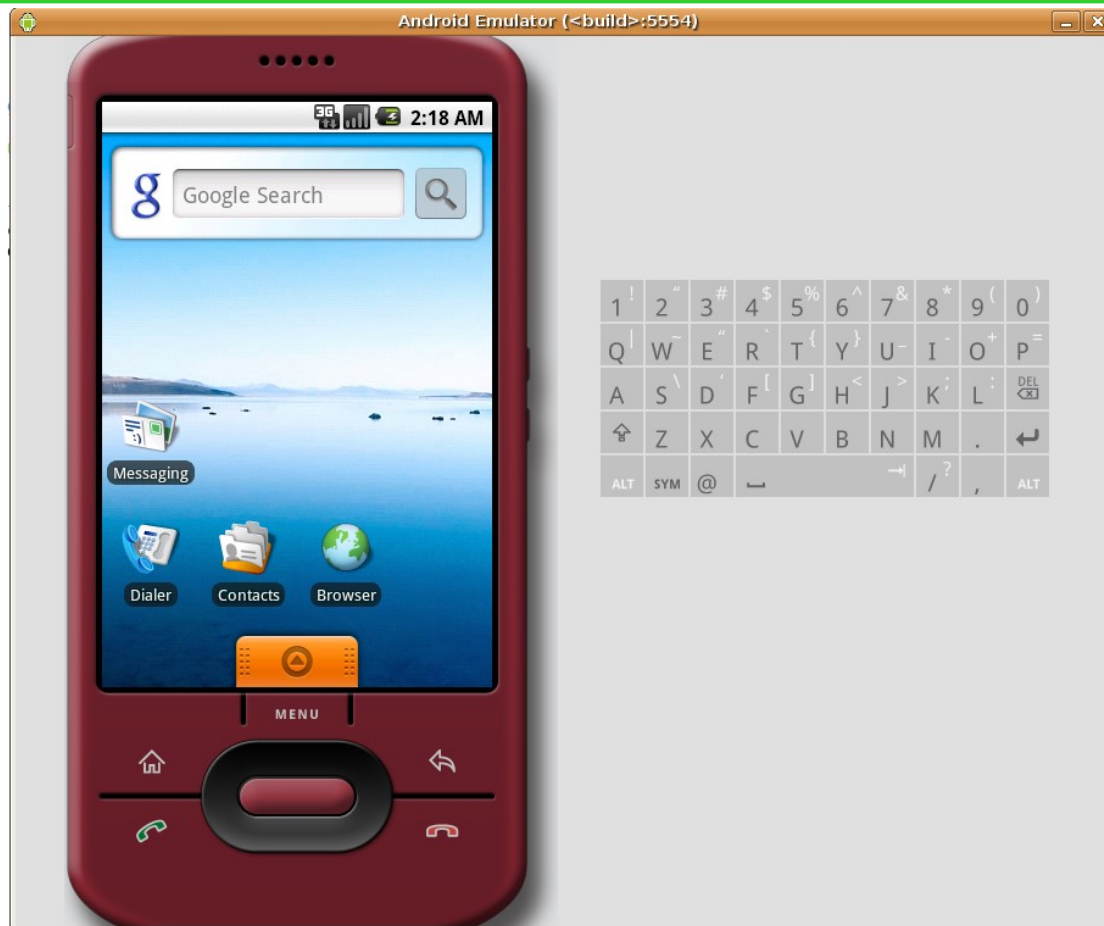
启动界面

2.3 运行 Android



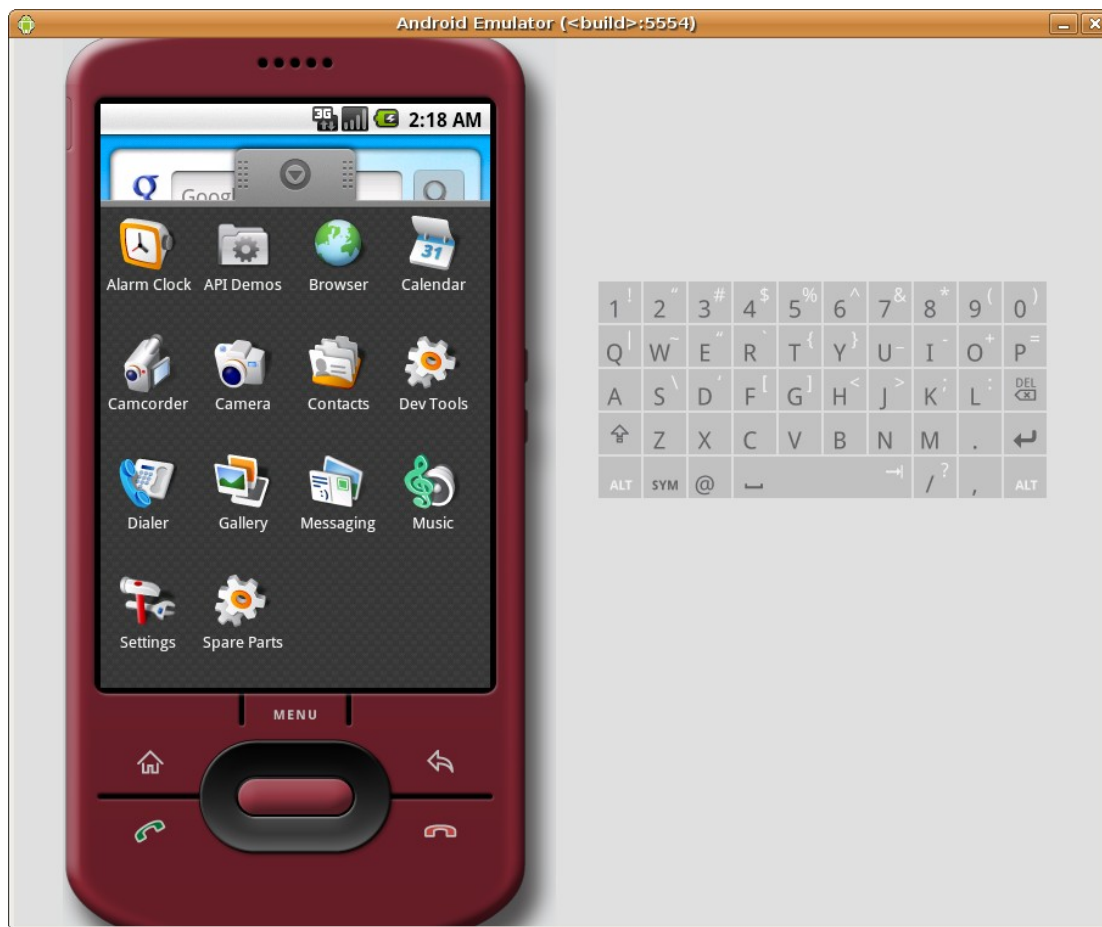
启动界面

2.3 运行 Android



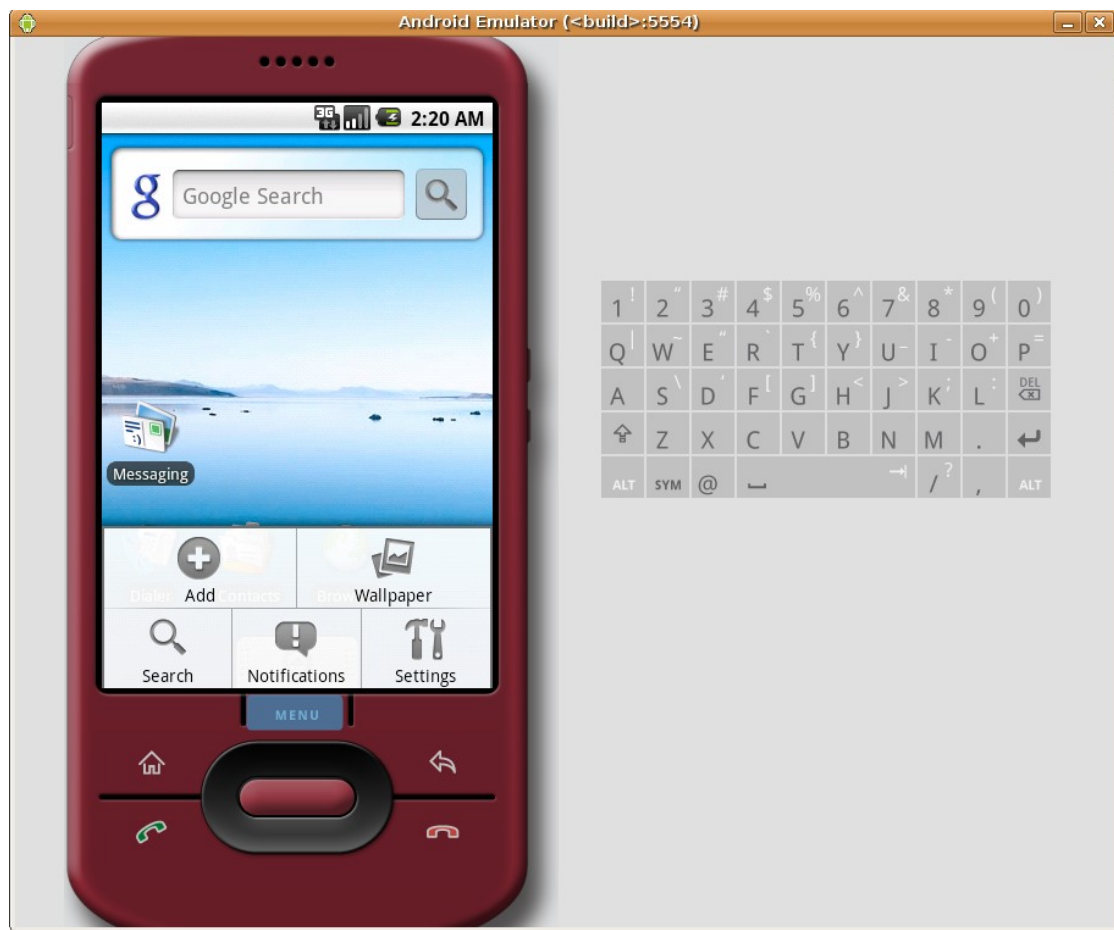
Android 的桌面
(HVGA-P: 320×480)

2.3 运行 Android



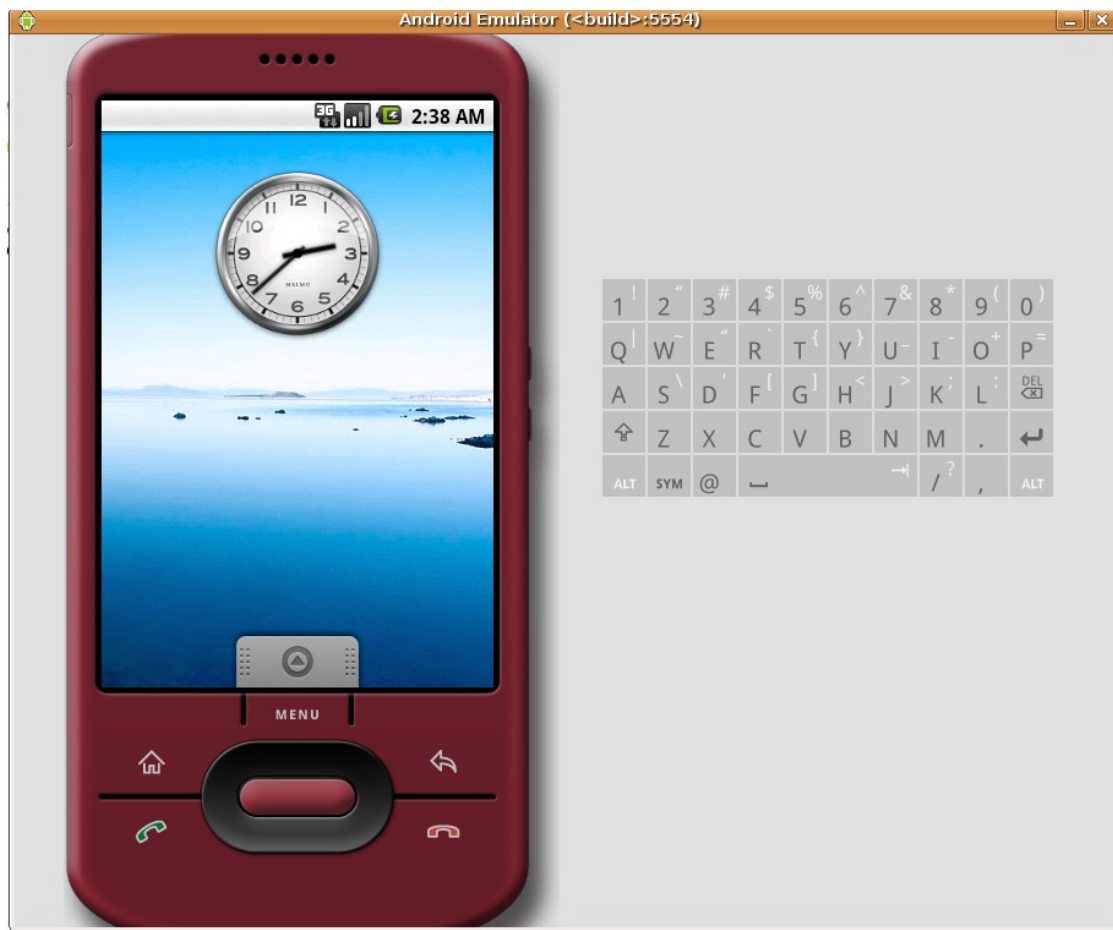
Android 的应用主菜单

2.3 运行 Android



在 **Android** 中使用菜单

2.3 运行 Android



拖拉到主界面的另外一屏

2.3 运行 Android

在 Shell 提示符查看目标系统根目录:

```
# ls -l
drwxrwxrwt root      root      2009-06-15 02:17 sqlite_stmt_journals
drwxrwx--- system    cache    2009-06-15 02:18 cache
d----- system      system    2009-06-15 02:17 sdcard
lrwxrwxrwx root      root      2009-06-15 02:17 etc -> /system/etc
drwxr-xr-x root      root      2009-05-28 02:16 system
drwxr-xr-x root      root      1970-01-01 00:00 sys
drwxr-x--- root      root      1970-01-01 00:00 sbin
dr-xr-xr-x root      root      1970-01-01 00:00 proc
-rwxr-x--- root      root      9075 1970-01-01 00:00 init.rc
-rwxr-x--- root      root      1677 1970-01-01 00:00 init.goldfish.rc
-rwxr-x--- root      root      106568 1970-01-01 00:00 init
-rw-r--r-- root      root      118 1970-01-01 00:00 default.prop
drwxrwx--x system    system    2009-05-28 02:49 data
drwx----- root      root      1970-01-01 00:00 root
drwxr-xr-x root      root      2009-06-15 02:18 dev
```

2.3 运行 Android

在 Shell 提示符查看目标系统的进程:

```
# ps
USER      PID    PPID    VSIZE  RSS      WCHAN    PC      NAME
root       1       0       280    188     c008de04 0000c74c S  /init
root       2       0        0      0      c004b334 00000000 S  kthreadd
root       3       2        0      0      c003cf68 00000000 S  ksoftirqd/0
root       4       2        0      0      c00486b8 00000000 S  events/0
root       5       2        0      0      c00486b8 00000000 S  khelper
root      10      2        0      0      c00486b8 00000000 S  suspend
root      42      2        0      0      c00486b8 00000000 S  kblockd/0
root      45      2        0      0      c00486b8 00000000 S  cqueue
root      47      2        0      0      c016f13c 00000000 S  kseriod
root      51      2        0      0      c00486b8 00000000 S  kmcd
root      96      2        0      0      c0065c7c 00000000 S  pdflush
root      97      2        0      0      c0065c7c 00000000 S  pdflush
root      98      2        0      0      c006990c 00000000 S  kswapd0
root     100      2        0      0      c00486b8 00000000 S  aio/0
root     269      2        0      0      c016c884 00000000 S  mtblockd
root     304      2        0      0      c00486b8 00000000 S  rpciod/0
```

2.3 运行 Android

```
root      540    1      740    328    c003aa1c afe0d08c S /system/bin/sh
system    541    1      808    264    c01654b4 afe0c45c S /system/bin/servicemanager
root      542    1      836    364    c008e3f4 afe0c584 S /system/bin/vold
root      543    1      668    264    c0192c20 afe0cdec S /system/bin/debuggerd
radio     544    1      5392   684    ffffffff afe0cacc S /system/bin/rild
root      545    1      72256  20876  c008e3f4 afe0c584 S zygote
media     546    1      17404  3496   ffffffff afe0c45c S /system/bin/mediaserver
bluetooth 547    1      1168   568    c008de04 afe0d25c S /system/bin/dbus-daemon
root      548    1      800    300    c01f3b04 afe0c1bc S /system/bin/install-d
root      551    1      840    356    c00ae7b0 afe0d1dc S /system/bin/qemud
root      554    1      1268   116    ffffffff 0000e8f4 S /sbin/adbd
system    570    545    175652 23972  ffffffff afe0c45c S system_server
radio     609    545    105704 17584  ffffffff afe0d3e4 S com.android.phone
app_4     611    545    113380 19492  ffffffff afe0d3e4 S android.process.acore
app_12    632    545    95392  13228  ffffffff afe0d3e4 S com.android.mms
app_4     645    545    97192  12964  ffffffff afe0d3e4 S com.android.inputmethod.latin
app_5     655    545    95164  13376  ffffffff afe0d3e4 S android.process.media
app_7     668    545    97700  14264  ffffffff afe0d3e4 S com.android.calendar
app_11    684    545    94132  12624  ffffffff afe0d3e4 S com.android.alarmclock
root      702    540    888    340    00000000 afe0c1bc R ps
```

2.3 运行 Android

logcat 是 Android 中一个命令行工具，可以用于得到程序的 log 信息。

logcat 使用方法如下所示：

logcat [options] [filterspecs]

logcat 的选项包括：

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

2.3 运行 Android



(HVGA-L: 480×320)

2.3 运行 Android



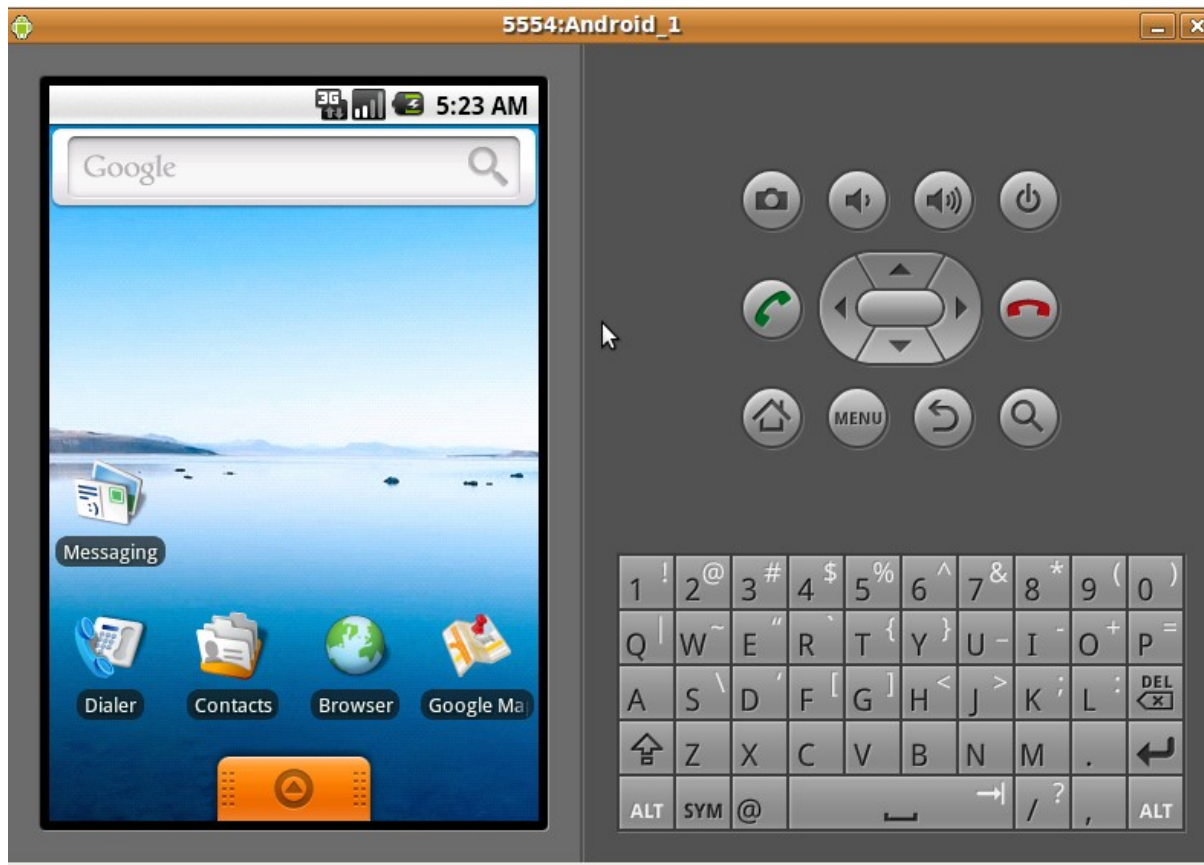
(QVGA-P: 320×240)

2.3 运行 Android



(QVGA-L: 320×240)

3.3 Linux Android SDK 环境



Android 1.6 的模拟器环境



谢谢！