# 缩写

AMS: ActivityManagerService. android系统服务,Activity管理的服务端，用于管理activity的各种行为，控制activity的生命周期，派发消息事件，低内存管理等等。实现了IBinder接口，可以用于进程间通信。

# Android源码的编译

1. 下载源码

http://source.android.com/source/downloading.html

1. 搭建编译环境

http://source.android.com/source/initializing.html

1. 编译

$ source build/envsetup.sh

$ lunch full-eng

$ make -j8

G703d:

./autobuild.sh eng

Android编译选项：

eng：debug版本

user:release版本

userDebug版本：部分debug版本

build/envsetup.sh：该脚本的作用是初始化编译环境，并引入一些辅助的 Shell 函数，这其中就包括第二步使用 lunch 函数。

build/envsetup.sh 中定义的常用函数

|  |  |
| --- | --- |
| **croot** | 切换到源码树的根目录 |
| **m** | 在源码树的根目录执行 make |
| **mm** | Build 当前目录下的模块 |
| **mmm** | Build 指定目录下的模块 |
| **cgrep** | 在所有 C/C++ 文件上执行 grep |
| **jgrep** | 在所有 Java 文件上执行 grep |
| **resgrep** | 在所有 res/\*.xml 文件上执行 grep |
| **godir** | 转到包含某个文件的目录路径 |
| **printconfig** | 显示当前 Build 的配置信息 |
| **add\_lunch\_combo** | 在 lunch 函数的菜单中添加一个条目 |

lunch 函数：参数用来指定此次编译的目标设备以及编译类型。 “full”是 Android 源码中已经定义好的一种产品，是为模拟器而设置的。而编译类型会影响最终系统中包含的模块。

make 的参数“-j”指定了同时编译的 Job 数量，这是个整数，该值通常是编译主机 CPU 支持的并发线程总数的 1 倍或 2 倍。在调用 make 命令时，如果没有指定任何目标，则将使用默认的名称为“droid”目标，该目标会编译出完整的 Android 系统镜像。

整个 Build 系统中的 Make 文件可以分为三类：

第一类是 Build 系统核心文件，此类文件定义了整个 Build 系统的框架，而其他所有 Make 文件都是在这个框架的基础上编写出来的。Build 系统核心文件全部位于 /build/core目录下。

第二类是针对某个产品的 Make 文件，这些文件通常位于 device 目录下，该目录下又以公司名以及产品名分为两级目录，例如，/device/sony/it26 目录下的文件共同构成了对于 Sony LT26 型号手机的定义。

第三类是针对某个模块的 Make 文件。整个系统中，包含了大量的模块，每个模块都有一个专门的 Make 文件，这类文件的名称统一为“Android.mk”，该文件中定义了如何编译当前模块。Build 系统会在整个源码树中扫描名称为“Android.mk”的文件并根据其中的内容执行模块的编译。

整个 Build 系统的入口文件是源码树根目录下名称为“Makefile”的文件，当在源代码根目录上调用 make 命令时，make 命令首先将读取该文件。

# build.prop

build.prop文件生成在android\out\_703d\_ebox\target\product\gs703d\system目录下

build.prop的生成是由make系统解析build/core/Makefile完成。

Makefile中调用build/tools/buildinfo.sh执行脚本，并输出到build.prop

Makefile中直接把$(TARGET\_DEVICE\_DIR)/system.prop的内容追加到build.prop中

收集ADDITIONAL\_BUILD\_PROPERTIES中的属性，追加到build.prop中。

每个属性都有一个名称和值，他们都是字符串格式。属性被大量使用在Android系统中，用来记录系统设置或进程之间的信息交换。属性是在整个系统中全局可见的。每个进程可以get/set属性。

在系统初始化时，Android将分配一个共享内存区来存储的属性。这些是由“init”守护进程完成的，其源代码位于：device/system/init。“init”守护进程将启动一个属性服务。

每一个客户端想要设置属性时，必须连接属性服务，再向其发送信息。属性服务将会在共享内存区中修改和创建属性。任何客户端想获得属性信息，可以从共享内存直接读取。在Android运行时刻可以通过property\_get()[c/c++域] / SystemProperties\_get\*()[Java域]读取这些属性值。

当启动属性服务时，将从以下文件中加载默认属性：

/default.prop

/system/build.prop

/system/default.prop

/data/local.prop

后加载的属性将覆盖原先的值。这些属性加载之后，最后加载的属性会被保持在/data/property中。

如果属性名称以“ro.”开头，那么这个属性被视为只读属性。

如果属性名称以“persist.”开头，当设置这个属性时，其值也将写入/data/property

如果属性名称以“net.”开头，当设置这个属性时，“net.change”属性将会自动设置，以加入到最后修改的属性名。

属性“ ctrl.start ”和“ ctrl.stop ”是用来启动和停止服务。

每一项服务必须在/init.rc中定义。系统启动时，init守护进程将解析init.rc和启动属性服务。一旦收到设置“ ctrl.start ”属性的请求，属性服务将使用该属性值作为服务名找到该服务，启动该服务。这项服务的启动结果将会放入“ init.svc.<服务名>“属性中 。客户端应用程序可以轮询那个属性值，以确定结果

# AndroidManifest.xml

## android:persistent="true"

在Android系统中，有一种永久性应用。它们对应的AndroidManifest.xml文件里，会将persistent属性设为true

# 添加Android系统log

**Android.mk文件**

LOCAL\_SHARED\_LIBRARIES :+= \

liblog

**\*.c/\*.h文件**

#define LOG\_TAG "onvifsvr"

#define LOG\_NDEBUG 1

#include <utils/Log.h>

# Android属性（property）机制

## 开机时系统属性的加载

系统启动时以下面的次序加载预先设定属性：

/default.prop

/system/build.prop

/system/default.prop

/data/local.prop

/data/property/\*

后加载的如果有重名的则覆盖前面的。

persist.\* : 以persist开始的属性会在/data/property存一个副本。也就是说，如果程序调property\_set设了一个以persist为前缀的属性，系统会在/data/property/\*里加一个文件记录这个属性，重启以后这个属性还有。如果property\_set其它属性，因为属性是在内存里存，所以重启后这个属性就没有了。

ro.\* :以ro为前缀的属性不能修改。

## 应用程序属性使用方法

**java应用里设置属性：**

import android.os.SystemProperties;

SystemProperties.set("persist.sys.country",”china”);

String vmHeapSize = SystemProperties.get("dalvik.vm.heapgrowthlimit", "24m");

也可以用SystemProperties.getBoolean，getInt等。

**在native C中设置属性：**

#include "cutils/properties.h"

property\_set("vold.decrypt", "trigger\_load\_persist\_props");

char encrypted\_state[32];

property\_get("ro.crypto.state", encrypted\_state, "");

最后一个参数是默认值。

## 启动脚本中属性使用方法

一般property启动应该加在init.<your hardware>.rc而不是直接init.rc里。下面是一个init.rc里的例子：

on property:ro.kernel.qemu=1

start adbd

意思是如果ro.kernel.qemu=1，也就是当前是模拟器的话，则启动adb服务。

## 属性的权限限制

只有有权限的进程才能修改属性，要不随便写一个就改系统属性那当黑客也太容易了。

权限在system/core/init/property\_service.c里定义：

property\_perms[] = {

{ "net.rmnet0.", AID\_RADIO, 0 },

{ "net.gprs.", AID\_RADIO, 0 },

{ "net.ppp", AID\_RADIO, 0 },

{ "net.qmi", AID\_RADIO, 0 },

{ "ril.", AID\_RADIO, 0 },

{ "gsm.", AID\_RADIO, 0 },

{ "persist.radio", AID\_RADIO, 0 },

{ "net.dns", AID\_RADIO, 0 },

{ "net.", AID\_SYSTEM, 0 },

{ "dev.", AID\_SYSTEM, 0 },

{ "runtime.", AID\_SYSTEM, 0 },

{ "hw.", AID\_SYSTEM, 0 },

{ "sys.", AID\_SYSTEM, 0 },

...

其实一般应用程序都不会去修改系统属性，所以也不用太在意。

## 属性实现原理

属性初始化的入口点是property\_init ，在system/core/init/property\_service.c中定义。

它的主要工作是申请32k共享内存，其中前1k是属性区的头，后面31k可以存247个属性（受前1k头的限制）。

property\_init加载/default.prop的属性定义。

start\_property\_service中加载其它的系统属性（build.prop, local.prop,…）。

加载完属性服务创建一个socket和其他进程通信（设置或读取属性）。

Init进程poll属性的socket，等待和处理属性请求。如果有请求到来，则调用handle\_property\_set\_fd来处理这个请求。在这个函数里，首先检查请求者的uid/gid看看是否有权限，如果有权限则调property\_service.c中的property\_set函数。

在property\_set函数中，它先查找就没有这个属性，如果找到，更改属性。如果找不到，则添加新属性。更改时还会判断是不是“ro”属性，如果是，则不能更改。如果是persist的话还会写到/data/property/<name>中。

最后它会调property\_changed，把事件挂到队列里，如果有人注册这个属性的话（比如init.rc中on property:ro.kernel.qemu=1），最终会调它的会调函数。

property名字长度限制是32字节，值的限制是92字节。

# end