init

init是Linux启动后运行的第一个用户态进程。一般作用是启动系统，并管理其启动的进程（babysitting，比如重启崩溃的进程）。

Android的init，提供了**四个基本的服务**：

1. 添加/删除设备支持 (udev like)
2. 系统属性支持 (GConf like)
3. 管理服务进程 (babysitting)
4. keychord

Android的init，集成了“**触发器(Triggers)**触发事件 - **响应(Actions)**”的方式。在配置文件(init.rc)中，定义了对触发事件的响应(Actions)。

预定义的触发器(Trigger)有：

- *early-init*、*init*、*early-boot*和*boot*

  init启动过程中，依次触发上述事件。

- property:<name>=<value>

  当某个属性<name>修改成值<value>，触发相应Actions。

Android的init，还集成了**权限表**：

- **qemu_perms**, **devperms**和**devperms_partners**

  确定设备节点的所有者(组)和权限。

- **control_perms**，**property_perms**

  确定：进程是否能够设置某系统属性？

init.rc

init.rc语法:

- 逐行解析，空格隔开各个域（支持*双引号*或*反斜杠转义空格*的方式允许某域包含空格)
- 连续若干行组成一段（Section）。
- 行尾反斜杠 -- 续行
- 行首井号 -- 注释
- 注意：
  - init支持的配置文件（init.*rc、.prop），均只支持ASCII编码。
  - init.rc指定的命令，**不是系统中的命令**。只能由init解释成函数调用。

其中，每个段必然为下述类型之一：

1. **Action**

```
on <trigger>
   <command>
   <command>
   <command>
```

2. **Service**

```
service <name> <pathname> [ <argument> ]*
   <option>
   <option>
```

   服务可以属于一个class，通过**class选项**标记。未标识的则属于名为“default”的class。

   “class选项”提供了启动一组服务的机制， 参见**class_start**和**class_stop**命令。

   注意：目前可以有多个对应一个<trigger>的Action段，；但Service必须唯一（通过<name>标识）；第二个同名Service将会被忽略。

每个段的范围为：起始标记开始，直到遇到新段的起始标记。

init.rc示例

添加/删除设备支持

内核通过netlink套接字，通知init新设备接入/移除，init进而建立相应设备节点。设备节点的所有者和权限，依次检查**qemu_perms** -> **devperms** -> **devperms_partners**确定，或者为uid = 0, gid = 0, mode = 0600。

若设备需要加载固件，则固件需放置在**/etc/firmware**目录下。

实现细节

系统属性支持

init提供设置/读取各条系统属性服务，一条属性即"<属性名> = <值>"。通过<属性名>可以查询<属性值>。

其中，只有init能设置系统属性，因此更改系统属性需要向init请求，由init检查相应权限表后代为设置。

系统启动后，将初始化系统属性空间，加载磁盘上转储的属性。系统属性的内容来源（依次载入）：

1. (ramdisk)文件**/default.prop**
2. 内核参数：

内核参数	静态变量	系统属性	备注
qemu	**qemu**	导致**所有内核参数**转储为"ro.kernel.<para> = <value>"	
androidboot.console	**console**		若指定，则含有console选项的Services使用此控制台

androidboot.mode	**bootmode**	<bootmode>:
		• "factory"
		  ○ **ro.factorytest** = "1"
		  ○ **ro.bootmode** = <bootmode>
		• "factory2"
		  ○ **ro.factorytest** = "2"
		  ○ **ro.bootmode** = <bootmode>
		• ...
		  ○ **ro.factorytest** = "0"
		  ○ **ro.bootmode** = <bootmode> or "unknown"	

androidboot.serialno	**serialno**	**ro.serialno** = <serialno> or ""	
androidboot.baseband	**baseband**	**ro.baseband** = <baseband> or "unknown"	
androidboot.carrier	**carrier**	**ro.carrier** = <carrier> or "unknown"	
androidboot.bootloader	**bootloader**	**ro.bootloader** = <bootloader> or "unknown"	
androidboot.hardware	**hardware**	"ro.hardware" = <hardware> or hardware detected from/proc/cpuinfo	"ro.revision" <= snprintf(tmp, PROP_VALUE_MAX, "%d", hardware_revision);

android.ril	p = &**qemu_perms**[qemu_perm_count];
	p->name = "/dev/<value>";
	p->perm = 0660;
	p->uid = AID_RADIO;
	p->.gid = AID_ROOT;
	p->prefix = 0;
	qemu_perm_count++;		

3. 文件 :**/system/build.prop**，**/system/default.prop**，**/data/local.prop**
4. 目录**/data/property**下各个文件：persist.<name>，文件内容为属性值。

特殊的属性：

- **ro.***：创建后不能更改其值。
- **net.***(但不是net.change)：其被设置后，会在**net.change**标明此属性最近被更新。
- **persist.***：其值会被转储到磁盘上（上述提到的系统属性初始化内容来源4)
- **ctl.start**、**ctl.stop**：其值为"*服务名[:参数列表]*"。特殊的属性，**只写**，其作用仅限于启动/停止指定服务：
  - root用户或system用户可以使用这两个属性，启动/停止任意服务。
  - 其他用户，检查其安全上下文(*服务名，uid，gid*)，是否被**control_perms**所许可。
  - 这里特别指出，使用"*服务名:参数列表*"形式，仅对**ctl.start**，并且服务要有**oneshot**选项。其中，参数列表中，各参数由空格分隔。
- init.svc.<name>：State of a named service ("stopped","stopping"，"running","restarting")

实现细节

管理服务进程

init管理的服务，按照其生命周期可分为两种类型:

1. 指定了**oneshot**选项的服务 -- 该服务运行一次退出。
2. 其他 -- 服务一直运行，直到由init关闭之。若在其他情形下服务退出，则init自动重启之。其中指定了**critical**选项的服务，属于关键服务，若init发现其频繁崩溃则重启系统。

启动服务前，init准备如下工作：

- 传递给全局属性空间的引用，以便服务访问属性。
- [**setenv**选项]：为服务准备环境变量。
- [**socket**选项]：为服务准备UNIX_DOMAIN的套接字（位置：/dev/socket/<sockname>，fd通过环境变量"ANDROID_SOCKET_<sockname>"告知）。
- [**ioprio**选项]：设置服务的IO调度器类型和优先级(通过ioprio_set系统调用)。
- [**console**选项]：为服务准备控制台。若系统没有可用的控制台，则该服务被禁用。
- [**user**选项、**group**选项]：设置服务的uid、gids。默认为uid = 0, gid = 0。
- 记录服务状态： 启动时间、pid以及通过属性通知：**init.svc.<name>** = "running"。

当服务退出时，init作如下工作：

- *非oneshot*的服务：杀死服务所在进程组中其他进程（SIGKILL)。
- 移除sockets
- 针对不同类型的服务，记录状态，或标记服务需重启：
  - **oneshot**：设置服务状态为*DISABLED* => 属性**init.svc.<name>** = "stopped"。
  - *非oneshot*：
    - **critical** -- 记录崩溃次数，崩溃时间；若4分钟内崩溃四次 => 立即同步磁盘并重启进入恢复模式。
    - 运行**onrestart**选项指定的<command>。
    - 设置服务状态为*RESTARTING* => 属性**init.svc.<name>** = "restarting"。待重启。
- 之后重启标记为*RESTARTING*的服务。

init根据init.rc中**stop**、**class_stop**命令的指示或者设置系统属性"ctl.stop=<service>"，来停止服务：

- 设置服务状态为*DISABLED* => 属性**init.svc.<name>** = "stopping"。
- 服务是否在运行？
  - 是：杀死服务所在进程组所有进程（SIGTERM） --> init侦测到服务进程已退出，则置属性**init.svc.<name>** = "stopped"。
- 否：置属性**init.svc.<name>** = "stopped"

实现细节

keychord

keychord是这样一种机制，服务通过选项**keycodes**，指定若干keycodes，之后收到任一指定的keycode，对应服务将被启动。

该启用该机制需要：

- 属性**ro.debuggable** == "1"
- adb服务已运行

实现细节

Hack

调试选项

1. 在Android上，可以指定**rd_init**内核参数，指定其他程序，作为“init”。

2. 在/default.prop，设置系统属性“ro.debuggable=1”
3. 在init.*rc中，通过**loglevel**命令，设置日志输入的级别。比如，为了尽早输出全部日志，将下述内容加入init.rc

```
On early-init
    loglevel 6 #INFO, NOTICE and Error
```

4. By default, programs executed by init will drop stdout and stderr into /dev/null. To help with debugging, you can execute your program via the Andoird program logwrapper. This will redirect stdout/stderr into the Android logging system (accessed via logcat). For example

```
service akmd /system/bin/logwrapper /sbin/akmd
```

移植注意

1. init.rc：挂载/system，/data分区，更改设备节点
2. init.rc：通过device命令，加入“第三方设备节点所有者及权限表”

启动流程

Android开机后，由U-BOOT加载内核及ramdisk。之后将控制权交给ramdisk中的init。一些初始化工作完成后， init挂载system和data分区，继续启动system下服务。

其中ramdisk的文件系统布局如下：

```
/init    /init.rc    /init.goldfish.rc    /init.<hardware>.rc    /init.logo.rle
/default.prop    #转储到ramdisk的系统属性
/dev/
/proc/
/sys/
/sbin/adbd
/lib/modules/    #包含一系列.ko文件
/etc/firmware/   #包含驱动所需固件
/system/
/data/
```

另外，Android将用户名(组名)与uid(gid)对应关系写入了数组**android_ids**中。从而避免依赖文件系统。

ramdisk阶段

在此阶段init工作流如下：

1. 设置对**SIGCHLD**信号的处理句柄
   - flags：*SA_NOCLDSTOP*，表示只关心子进程退出）。
   - 处理句柄：写入到socket fd -- 静态变量**signal_fd**。注意：signal_fd此时还未打开，因此此时若有子进程启动/退出，init不对其处理。(write调用会*默默地*失败）
2. umask(0) -- 创建文件的权限 = mod & (~umask)
3. 创建基本的文件系统布局

```
mkdir("/dev", 0755);
mkdir("/proc", 0755);
mkdir("/sys", 0755);

mount("tmpfs", "/dev", "tmpfs", 0, "mode=0755");
mkdir("/dev/pts", 0755);
mkdir("/dev/socket", 0755);
mount("devpts", "/dev/pts", "devpts", 0, NULL);
mount("proc", "/proc", "proc", 0, NULL);
mount("sysfs", "/sys", "sysfs", 0, NULL);
```

   这里注意，若mkdir的目录已经存在，其mkdir会*默默地*失败 -- 目录的权限未必如上面代码所述。

4. 将标准输入、输出出错重定向到null设备（1,3）
5. 初始化log，log输出到/dev/kmsg（1,11）
   - 向/dev/kmsg中写入数据，会作为内核日志输出
   - 后续INFO等函数，提供写入日志的功能。需要提供level和日志信息。其中，level用来决定是否输出日志（过滤 >静态变量**log_level**）。
   - ERROR - 3； NOTICE - 5； INFO - 6
   - 静态变量**log_level**默认为3
6. 解析文件init.rc
   - 基本的思想是逐行解析各域，将各域作为参数加入当前上下文(struct parse_state)。遇到换行:

        - 若为新section开始，则重载对应该section的parse_line。
        - 使用成员函数parse_line，解析一行。
    - 解析后的结果存储到两个链表中：
        - Service解析完成后，存储到全局双向链表 -- **service_list**
        - Action解析完后，存储到全局双向链表 -- **action_list**
7. 转储内核参数到**静态变量**，并chmod("/proc/cmdline", 0440)，防止非特权程序访问此文件
8. 解析硬件相关的"init.<hardware_name>.rc"
    - 硬件名称(静态变量**hardware**, **revision**<unsigned>) <== /proc/cpuinfo (匹配"nHardware"以及"nRevision")。注意，这里将覆盖从内核参数中传递的*androidboot.hardware*参数。
    - 解析/init.<hardware_name>.rc（这里注意：若**hardware**为空，则解析不存在的文件“init..rc”，从而*默默地*解析失败）
9. 触发"early-init"。
10. **准备netlink socket，并"冷启动"设备**
11. **初始化系统属性 -- 分配系统属性所在空间，加载/default.prop**
12. **初始化keychord** -- 加载服务的**keycodes**选项至"/dev/keychord"
13. 检查控制台是否可用 -- 尝试**console_name**。console_name为"/dev/<androidboot.console>"或“/dev/console”。
14. 载入位图"initlogo.rle"，并输出"A N D R O I D"等字样到**/dev/tty0**
15. **转储内核参数到系统属性中。**
16. 触发"early-init"。

system阶段

在Android提供的init.rc文件中，system分区和data分区被分别挂载。Android开始访问/system和/data下的资源。

1. **启动系统属性服务**
    - 载入system和data上转储的系统属性，并开启对"persist.*"属性转储磁盘的功能。
    - 建立公开socket，其他进程可以由此发出设置系统属性的请求。
2. **准备子进程退出通知通路** -- 打开一对socket，静态变量**signal_fd**(SIGCHLD信号的处理函数写入)；静态变量**signal_recv_fd**（事件循环读取）
3. 触发"early-boot"和"boot"。
4. **启用系统属性变更触发器。**首先，会对所有系统属性触发该触发器，然后启用该触发器。
5. poll -- 检测**device_fd**(设备更改事件)，**property_set_fd**(属性变更事件)，**signal_recv_fd**(子进程退出事件)[以及**keychord_fd**(监听keycodes)]。

附录

init.rc细节

Commands

- **loglevel** <log_level>

  <log_level>值越小，信息输出越少。典型取值：3(默认) - 仅输出ERROR; 5 - 输出ERROR和NOTICE；6 - 输出ERROR、NOTICE和INFO。

- **exec** <path> [<argument>]*

  该选项目前不支持。解析函数直接返回-1。

- **export** <name> <value>

  Set the environment variable <name> equal to <value> in the global environment (which will be inherited by all processes started after this command is executed)

- **ifup** <interface>

  Bring the network interface <interface> online.

- **import** <filename>

  Parse an init config file, extending the current configuration.

- **hostname** <name>

  Set the host name.

- **chdir** <directory>

  Change working directory.

- **chmod** <octal-mode> <path>

  Change file access permissions.

- **chown** <owner> <group> <path>

  Change file owner and group.

- **chroot** <directory>

  Change process root directory.

- **class_start** <serviceclass>

  Start all services of the specified class if they are not already running.

- **class_stop** <serviceclass>

  Stop all services of the specified class if they are currently running.

- **domainname** <name>

  Set the domain name.

- **insmod** <path>

  Install the module at <path>

- **mkdir** <path> [mode] [owner] [group]

  Create a directory at <path>, optionally with the given mode, owner, and group. If not provided, the directory is created with permissions 755 and owned by the root user and root group.

- **mount** <type> <device> <dir> [<mountoption>]*

  Attempt to mount the named device at the directory <dir> <device> may be of the form mtd@name to specify a mtd block device by name. <mountoption>s include "ro", "rw", "remount", "noatime", ...

- **setkey** <key_table> <scan_code> <value>

  对"/dev/tty0"设置某些键的返回值。参见man console_ioctl(KDSKBENT)

- **setprop** <name> <value>

  Set system property <name> to <value>.

- **setrlimit** <resource> <cur> <max>

  Set the rlimit for a resource.

- **start** <service>

  Start a service running if it is not already running.

- **stop** <service>

  Stop a service from running if it is currently running.

- **restart** <service> 重启某服务。

- **symlink** <target> <path>

  Create a symbolic link at <path> with the value <target>

- **sysclktz** <mins_west_of_gmt>

  Set the system clock base (0 if system clock ticks in GMT)

- **trigger** <event>

  Trigger an event. Used to queue an action from another action.

  <event>不能是“property:<key>=<value>”之类的触发事件。

- **copy** <src> <dst>

  复制文件<src> => <dst>。注意复制是将<src>完全读入内存再写入<dst>，因此<src>大小不能太大。

- **write** <path> <string>

  Open the file at <path> and write one string to it with write(2)

- **device** <dev_node> <octal-mode> <uid> <gid>

  添加第三方的“设备节点所有者及权限表”。其中<dev_node>可为：1)“/dev/.../<node>[*]”；2)"mtd@name"，对应节点路径为"/dev/mtd/mtd<name to Number>"。

Options

注意：部分选项可以在服务中多次出现，部分选项多次出现无意义甚至导致init内存泄漏(比如**keycodes**)。

- **capability**

  non-op

- **critical**

  重要的服务，如果在4分钟内，崩溃4次。则Android会重启，进入恢复模式。.

- **disabled**

  当启动*其所属的class*时，该服务不启动。除非指定服务名来启动。

- **setenv** <name> <value>

  设置启动服务的环境变量。该选项在一个服务的Section中可以多次出现。

- **socket** <name> <type> <perm> [<user> [<group>]]

  建立一个unix domain的命名套接字(/dev/socket/<name>)，并将其fd传递给启动的服务。<type>可以是“dgram”或者“stream”。<user>和<group>默认为0。

- **user** <username>

  切换当前用户为<username>，然后再执行本服务。目前默认的用户为root。

  若某服务需要linux capabilities，则不要使用**本项**。必须在root时请求所需的权限(capabilities)，然后切换到所需的uid。

- **group** <groupname> [<groupname>]*

  切换当前组。可以指定不只一个组。目前默认的组为root。

- **oneshot**

  若服务退出，不要重启之。（比如Android的开机启动动画服务，就启用了这样一个选项）

- **class** <name>

  加入某个组。所有加入组的服务可以一起启动/停止。未指定此选项的服务，属于“default”组。

- **onrestart** <commond> [<argument>]*

  服务重启时，执行命令。该选项在一个服务的Section中可以多次出现。

- **keycodes** <keycode> [<keycode>]*

  参见：keychord

- **ioprio** <rt|be|idle> <ioprio 0-7>

  指定IO调度类和IO调度优先级。调度类分为rt(real-time)、be(best-effort)、idle。其中rt和be支持0-7的优先级，0最高优先。参见man setioprio

- **console**

  服务需要控制台。服务的stdout、stderr和stdin均定向到控制台。若系统没有可用的控制台，则该服务被禁用。

添加/删除设备支持[实现细节]

首先建立一个netlink的socket，保存到静态变量**device_fd**

```
device_fd = socket(PF_NETLINK, SOCK_DGRAM, NETLINK_KOBJECT_UEVENT);
```

在设备冷启动阶段，init递归地访问*/sys/class*、*/sys/class*以及*/sys/devices*下，名为uevent的文件：

1. 向uevent写入字符串“add
   n”，从而触发“设备添加事件”。
2. 从**device_fd**中，读取“设备添加事件”，添加设备节点。

在设备热插拔阶段，poll **device_fd**来检测设备“添加删除事件”。

每个“设备添加/删除事件”，由下述域组成(域的分隔符为'
0')：

- ACTION= (add|remove)
- DEVPATH=
- SUBSYSTEM=

- FIRMWARE=
- MAJOR=
- MINOR=

对于设备添加事件(ACTION=add)，处理如下：

1. 添加/dev下的设备节点：

```
#伪代码说明
name=$(basename $DEVPATH)

case $SUBSYSTEM in
  block*)
    mknod /dev/block/$name b $MAJOR $MINOR
    ;;
  usb*)
    if $SUBSYSTEM == "usb"; then
      let bus_id=$MINOR/128+1
      let device_id=$MINOR%128+1

      #格式化字符串 bus_id, device_id => %03d
      mknod /dev/bus/usb/$bus_id/$device_id c $MAJOR $MINOR
    else
      #IGNORE
    fi
    ;;
  graphics*)
    mknod /dev/graphics/$name c $MAJOR $MINOR
    ;;
  oncrpc*)
    mknod /dev/oncrpc/$name c $MAJOR $MINOR
    ;;
  adsp*)
    mknod /dev/adsp/$name c $MAJOR $MINOR
    ;;
  msm_camera*)
    mknod /dev/msm_camera/$name c $MAJOR $MINOR
    ;;
  input*)
    mknod /dev/input/$name c $MAJOR $MINOR
    ;;
  mtd*)
    mknod /dev/mtd/$name c $MAJOR $MINOR
    ;;
  sound*)
    mknod /dev/snd/$name c $MAJOR $MINOR
    ;;
  misc*)
    if expr match $name "^log_*"; then
      mknod /dev/log/$name c $MAJOR $MINOR
    else
      mknod /dev/$name c $MAJOR $MINOR
    fi
    ;;
  *)
    mknod /dev/$name c $MAJOR $MINOR
    ;;
esac
```

2. 设置设备节点的所有者和权限.
   - 首先检查**qemu_perms**数组
   - 上步无果，则进一步检查**devperms**数组。
   - 上步无果，则检查**devperms_partners**双向列表。
   - 上步无果，则uid = 0; gid = 0; mode = 0600
3. 载入固件
   - SUBSYSTEM == "firmware" && ACTION == "add"
   - fork子进程，在子进程中完成固件加载，避免在init进程分配太大内存。

```
#start transfer
echo -n 1 > /sys${DEVPATH}/loading
```

```
    if cat /etc/firmware${FIRMWARE} > /sys${DEVPATH}/data; then
      #successful end of transfer
      echo -n 0 > /sys${DEVPATH}/loading
    else
      #abort transfer
      echo -n -1 > /sys${DEVPATH}/loading
    fi
```

设备删除事件的处理，仅是简单地移除相应的设备节点。

设备节点所有者及权限表

- **devperms** :

```
static struct perms_ devperms[] = {
    /* 设备节点路径 */            /* 权限 */  /* 所有者 */     /* 所有组 */    /* 是否模糊匹配(即：前缀匹配<设备节点路径> */
    { "/dev/null",                0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/zero",                0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/full",                0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/ptmx",                0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/tty",                 0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/random",              0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/urandom",             0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/ashmem",              0666,     AID_ROOT,       AID_ROOT,       0 },
    { "/dev/binder",              0666,     AID_ROOT,       AID_ROOT,       0 },

            /* logger should be world writable (for logging) but not readable */
    { "/dev/log/",                0662,     AID_ROOT,       AID_LOG,        1 },

    /* the msm hw3d client device node is world writable/readable. */
    { "/dev/msm_hw3dc",           0666,     AID_ROOT,       AID_ROOT,       0 },

    /* gpu driver for adreno200 is globally accessible */
    { "/dev/kgsl",                0666,     AID_ROOT,       AID_ROOT,       0 },

        /* these should not be world writable */
    { "/dev/diag",                0660,     AID_RADIO,      AID_RADIO,      0 },
    { "/dev/diag_arm9",           0660,     AID_RADIO,      AID_RADIO,      0 },
    { "/dev/android_adb",         0660,     AID_ADB,        AID_ADB,        0 },
    { "/dev/android_adb_enable",  0660,     AID_ADB,        AID_ADB,        0 },
    { "/dev/ttyMSM0",             0600,     AID_BLUETOOTH,  AID_BLUETOOTH,  0 },
    { "/dev/ttyHS0",              0600,     AID_BLUETOOTH,  AID_BLUETOOTH,  0 },
    { "/dev/uinput",              0660,     AID_SYSTEM,     AID_BLUETOOTH,  0 },
    { "/dev/alarm",               0664,     AID_SYSTEM,     AID_RADIO,      0 },
    { "/dev/tty0",                0660,     AID_ROOT,       AID_SYSTEM,     0 },
    { "/dev/graphics/",           0660,     AID_ROOT,       AID_GRAPHICS,   1 },
    { "/dev/msm_hw3dm",           0660,     AID_SYSTEM,     AID_GRAPHICS,   0 },
    { "/dev/input/",              0660,     AID_ROOT,       AID_INPUT,      1 },
    { "/dev/eac",                 0660,     AID_ROOT,       AID_AUDIO,      0 },
    { "/dev/cam",                 0660,     AID_ROOT,       AID_CAMERA,     0 },
    { "/dev/pmem",                0660,     AID_SYSTEM,     AID_GRAPHICS,   0 },
    { "/dev/pmem_adsp",           0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/pmem_camera",         0660,     AID_SYSTEM,     AID_CAMERA,     1 },
    { "/dev/oncrpc/",             0660,     AID_ROOT,       AID_SYSTEM,     1 },
    { "/dev/adsp/",               0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/snd/",                0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/mt9t013",             0660,     AID_SYSTEM,     AID_SYSTEM,     0 },
    { "/dev/msm_camera/",         0660,     AID_SYSTEM,     AID_SYSTEM,     1 },
    { "/dev/akm8976_daemon",      0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/akm8976_aot",         0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/akm8973_daemon",      0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/akm8973_aot",         0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/bma150",              0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/cm3602",              0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/akm8976_pffd",        0640,     AID_COMPASS,    AID_SYSTEM,     0 },
    { "/dev/lightsensor",         0640,     AID_SYSTEM,     AID_SYSTEM,     0 },
    { "/dev/msm_pcm_out",         0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/msm_pcm_in",          0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/msm_pcm_ctl",         0660,     AID_SYSTEM,     AID_AUDIO,      1 },
```

```
    { "/dev/msm_snd",               0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/msm_mp3",               0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/audience_a1026",        0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/tpa2018d1",             0660,     AID_SYSTEM,     AID_AUDIO,      1 },
    { "/dev/msm_audpre",            0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/msm_audio_ctl",         0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/htc-acoustic",          0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/vdec",                  0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/q6venc",                0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/snd/dsp",               0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/snd/dsp1",              0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/snd/mixer",             0660,     AID_SYSTEM,     AID_AUDIO,      0 },
    { "/dev/smd0",                  0640,     AID_RADIO,      AID_RADIO,      0 },
    { "/dev/qemu_trace",            0666,     AID_SYSTEM,     AID_SYSTEM,     0 },
    { "/dev/qmi",                   0640,     AID_RADIO,      AID_RADIO,      0 },
    { "/dev/qmi0",                  0640,     AID_RADIO,      AID_RADIO,      0 },
    { "/dev/qmi1",                  0640,     AID_RADIO,      AID_RADIO,      0 },
    { "/dev/qmi2",                  0640,     AID_RADIO,      AID_RADIO,      0 },

        /* CDMA radio interface MUX */
    { "/dev/ts0710mux",             0640,     AID_RADIO,      AID_RADIO,      1 },
    { "/dev/ppp",                   0660,     AID_RADIO,      AID_VPN,        0 },
    { "/dev/tun",                   0640,     AID_VPN,        AID_VPN,        0 },
    { "/dev/bus/usb/",              0660,     AID_ROOT,       AID_USB,        1 },
    { NULL, 0, 0, 0, 0 },
}
```

- **devperms_partners** ：由**device**命令添加。

系统属性支持[实现细节]

分配系统属性空间

通过Android的ashmem机制(/dev/ashmem，支持分配的内存pin在物理内存中)，分配一块名为“system_properties”空间：

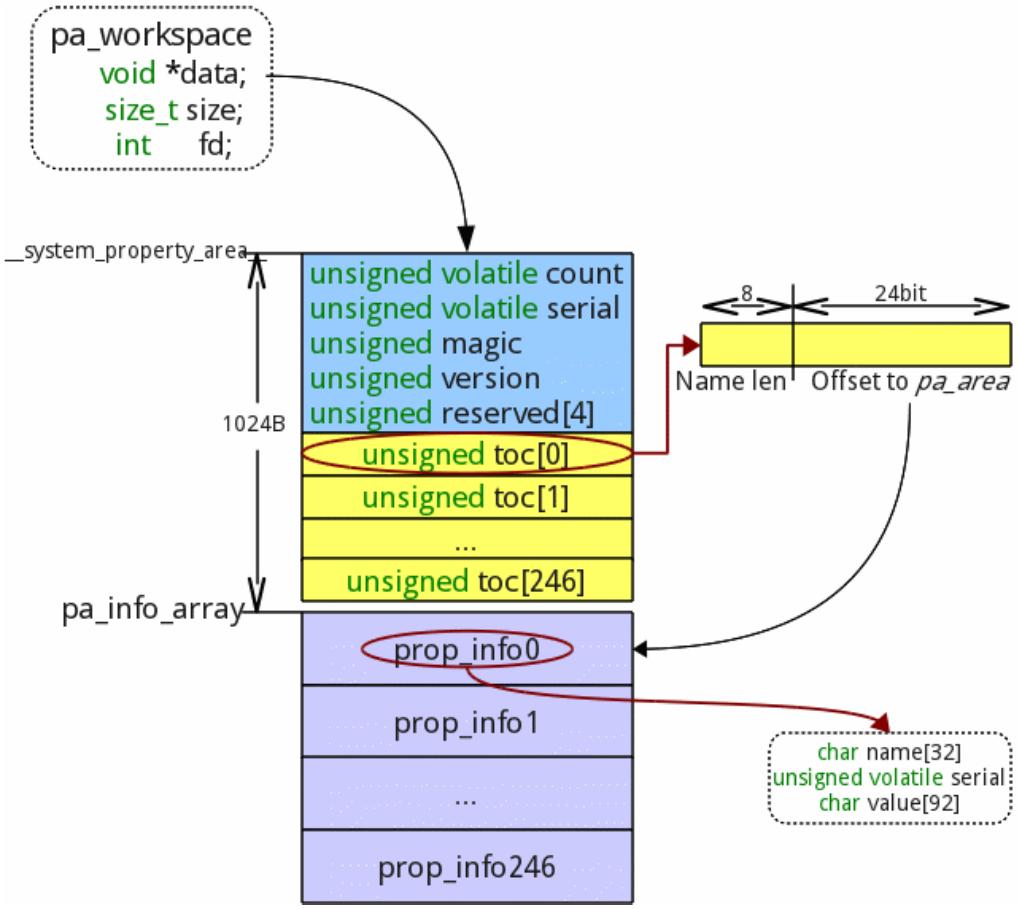
```
int fd = ashmem_create_region("system_properties", size);
if(fd < 0) return -1;

void *data = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if(data == MAP_FAILED) goto out;

/* 除了init以外的进程对此空间只读 */
ashmem_set_prot_region(fd, PROT_READ);
```

并将系统属性空间相关信息保存到静态变量**pa_workspace**和**pa_info_array**中。

系统属性空间分为三个部分：

- 空间头，包含了属性条目的数目(count），property空间的序列号(标记修改的次数)，magic号(当前值：0x504f5250)，version(当前值：0x4543476)
- 目录区(toc)，用于索引属性条目。包含*属性名的长度* 和 *属性条目地址(距离property空间起始的偏移)*。其中含有*属性名的长度*是用来优化属性名查找 (先比较字符串长度，不符合直接跳过）
- 属性条目。

property空间视图如下：



操作系统属性的低层API

- __system_property_find(代码来自bionic) -- 根据属性名，查找对应的prop_info

```
const prop_info *__system_property_find(const char *name)
{
    prop_area *pa = __system_property_area__;
    unsigned count = pa->count;
    unsigned *toc = pa->toc;
    unsigned len = strlen(name);
    prop_info *pi;

    while(count--) {
        unsigned entry = *toc++;
        if(TOC_NAME_LEN(entry) != len) continue;

        pi = TOC_TO_INFO(pa, entry);
        if(memcmp(name, pi->name, len)) continue;

        return pi;
    }

    return 0;
}
```

- 更新属性(代码来自init) -- 分为两种情况：
  - 属性不存在，需新建

```
/* name, value分别为属性名、属性值 */
namelen = strlen(name);
valuelen = strlen(value);

pi = (prop_info*) __system_property_find(name);
if (pi == 0) {
    pa = __system_property_area__;
    if(pa->count == PA_COUNT_MAX) return -1;
```

```
        /* 分配填充prop_info项 */
        pi = pa_info_array + pa->count;
        pi->serial = (valuelen << 24);
        memcpy(pi->name, name, namelen + 1);
        memcpy(pi->value, value, valuelen + 1);

        /* 分配、填充目录项 */
        pa->toc[pa->count] = (namelen << 24) | (((unsigned) pi) - ((unsigned) pa));

        pa->count++;
        /* 最后更新serial，通知更新完成 */
        /* serial数据成员被加上volatile限定，从而使得自加操作为原子操作
        pa->serial++;

        /* 唤醒 监视property空间更新事件的进程 */
        __futex_wake(&pa->serial, INT32_MAX);
    }
```

- 更新已有属性的值

```
    /* name，value分别为属性名、属性值 */
    namelen = strlen(name); /* namelen < PROP_NAME_MAX, PROP_NAME_MAX = 32 */
    valuelen = strlen(value); /* valuelen < PROP_VALUE_MAX, PROP_VALUE_MAX = 92 */

    pi = (prop_info*) __system_property_find(name);
    if (pi != 0) {

        pa = __system_property_area__;

        /* 标记修改开始 */
        pi->serial = pi->serial | 1;
        memcpy(pi->value, value, len + 1);
        /* 标记修改结束，serial最后一位不为1 */
        pi->serial = (len << 24) | ((pi->serial + 1) & 0xffffff);
        /* 唤醒 监视该prop_info更新事件的进程 */
        __futex_wake(&pi->serial, INT32_MAX);

        pa->serial++;
        /* 唤醒 监视property空间更新事件的进程 */
        __futex_wake(&pa->serial, INT32_MAX);
    }
```

- __system_property_get(代码来自bionic) -- 获取属性值

```
int __system_property_get(const char *name, char *value)
{
    const prop_info *pi = __system_property_find(name);

    if(pi != 0) {
        return __system_property_read(pi, 0, value);
    } else {
        value[0] = 0;
        return 0;
    }
}

int __system_property_read(const prop_info *pi, char *name, char *value)
{
    unsigned serial, len;

    for(;;) {
        serial = pi->serial;
        while(SERIAL_DIRTY(serial)) /* while(serial & 1)*/ {
            /* 交由内核仲裁：
               if (pi->serial == serial) => 立即返回
               else => 睡眠，直到被唤醒
            */
            __futex_wait(&pi->serial, serial, 0);
            serial = pi->serial;
```

```
            }
            len = SERIAL_VALUE_LEN(serial); /* (serial) >> 24 */
            memcpy(value, pi->value, len + 1);
            if(serial == pi->serial) {
                if(name != 0) {
                    strcpy(name, pi->name);
                }
                return len;
            }
            /* 走到这里：serial已经发生变化，预示读取期间有更新，再次读取... */
        }
    }
```

- __system_property_wait(代码来自bionic) -- 监视property空间或指定prop_info更新

```
int __system_property_wait(const prop_info *pi)
{
    unsigned n;
    if(pi == 0) {
        prop_area *pa = __system_property_area__;
        n = pa->serial;
        do {
            __futex_wait(&pa->serial, n, 0);
        } while(n == pa->serial);
    } else {
        n = pi->serial;
        do {
            __futex_wait(&pi->serial, n, 0);
        } while(n == pi->serial);
    }
    return 0;
}
```

这里可以看到，多个属性读操作与单个属性写操作能安全并发。

init以外进程读取系统属性

将系统属性空间对应的fd，通过环境变量告知子进程(一般是服务)：

```
int fd = dup(pa_workspace.fd);
sprintf(tmp, "%d,%d", fd, pa_workspace.size);
add_environment("ANDROID_PROPERTY_WORKSPACE", tmp);
```

子进程的C库初始化时，自动读取该环境变量，并映射系统属性空间，空间入口保存到变量system_property_area中。

之后，子进程再次fork子进程时，其子进程将*继承*系统属性空间的fd和环境变量"ANDROID_PROPERTY_WORKSPACE"。

init以外进程设置系统属性

init对外公布一个UNIX DOMAIN的socket：/dev/socket/property_service(0666, uid=0, gid=0)，接受来自其他进程的设置系统属性请求。

假设请求设置的系统属性为“<prop_name> = <value>”，请求进程的uid和gid可以如下获得：

```
struct ucred cr; /* cr.pid, cr.uid, cr.gid */
socklen_t cr_size = sizeof(cr);

getsockopt(s, SOL_SOCKET, SO_PEERCRED, &cr, &cr_size)
```

决策是否允许设置系统属性的过程如下：

- 对于前缀为"ctl.*"的<prop_name>：
  - 允许**root或system用户**
  - 其他用户：用三元组(<value>, uid, gid)，检查**control_perms**来判断是否许可。
  - *注意*：该属性并不存储到系统属性空间上，一旦许可，则执行指定的服务（由<value>指定）。
- 其他<prop_name>
  - 允许**root用户**
  - 其他用户：用三元组(<prop_name>, uid, gid)，检查**property_perms**来判断是否许可。

系统属性设置权限表

- **control_perms** ：

```
struct {
    const char *service;
    unsigned int uid;
    unsigned int gid;
} control_perms[] = {
    { "dumpstate",AID_SHELL, AID_LOG },
     {NULL, 0, 0 }
};
```

- **property_perms** :

```
struct {
    const char *prefix;
    unsigned int uid;
    unsigned int gid;
} property_perms[] = {
    { "net.rmnet0.",      AID_RADIO,    0 },
    { "net.gprs.",        AID_RADIO,    0 },
    { "net.ppp",          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 },
    { "service.",         AID_SYSTEM,   0 },
    { "wlan.",            AID_SYSTEM,   0 },
    { "dhcp.",            AID_SYSTEM,   0 },
    { "dhcp.",            AID_DHCP,     0 },
    { "vpn.",             AID_SYSTEM,   0 },
    { "vpn.",             AID_VPN,      0 },
    { "debug.",           AID_SHELL,    0 },
    { "log.",             AID_SHELL,    0 },
    { "service.adb.root", AID_SHELL,    0 },
    { "persist.sys.",     AID_SYSTEM,   0 },
    { "persist.service.", AID_SYSTEM,   0 },
    { "persist.security.", AID_SYSTEM,   0 },
    { NULL, 0, 0 }
```

管理服务进程[实现细节]

子进程退出处理的流程为：

1. init通过处理SIGCHLD信号(flags:SA_NOCLDSTOP)，来获得子进程退出的事件。
2. 信号的处理函数写入数据到**signal_fd**(signal_fd, signal_recv_fd <= socketpair(...))
3. **signal_recv_fd**有数据，从而init从poll中醒来。
4. init用non-block的waitpid，回收所有退出的子进程，并根据pid查询到服务，进行相关处理。需要重启的服务标记状态为“RESTARTING”
5. 在下一轮poll前，init重启 标记为“RESTARTING”的服务：

```
process_needs_restart = 0;
service_for_each_flags(SVC_RESTARTING, ^ {
    time_t next_start_time = svc->time_started + 5;

    /* 若服务退出至今已逾5秒，则重启服务并返回 */
    if (next_start_time <= gettime()) {
        svc->flags &= (~SVC_RESTARTING);
        service_start(svc, NULL);
        return;
    }

    /* 若服务启动不久就退出了(<5s) */
    if ((next_start_time < process_needs_restart) ||
        (process_needs_restart == 0)) {
        process_needs_restart = next_start_time;
    }
});
```

```
    /* 由process_needs_restart计算poll的timeout */
```

keychord[实现细节]

初始化keychord时：将多个二元组（服务的一组keycodes，分配的id)写入**keychord_fd**（/dev/keychord）

poll **keychord_fd**，若存在数据，则其内容为id，通过id定位到服务并启动之。

{用户|组名}到{uid|gid}映射

```
#define AID_ROOT             0  /* traditional unix root user */

#define AID_SYSTEM        1000  /* system server */

#define AID_RADIO         1001  /* telephony subsystem, RIL */
#define AID_BLUETOOTH     1002  /* bluetooth subsystem */
#define AID_GRAPHICS      1003  /* graphics devices */
#define AID_INPUT         1004  /* input devices */
#define AID_AUDIO         1005  /* audio devices */
#define AID_CAMERA        1006  /* camera devices */
#define AID_LOG           1007  /* log devices */
#define AID_COMPASS       1008  /* compass device */
#define AID_MOUNT         1009  /* mountd socket */
#define AID_WIFI          1010  /* wifi subsystem */
#define AID_ADB           1011  /* android debug bridge (adbd) */
#define AID_INSTALL       1012  /* group for installing packages */
#define AID_MEDIA         1013  /* mediaserver process */
#define AID_DHCP          1014  /* dhcp client */
#define AID_SDCARD_RW     1015  /* external storage write access */
#define AID_VPN           1016  /* vpn system */
#define AID_KEYSTORE      1017  /* keystore subsystem */
#define AID_USB           1018  /* USB devices */
#define AID_DRM           1019  /* DRM server */
#define AID_DRMIO         1020  /* DRM IO server */

#define AID_SHELL         2000  /* adb and debug shell user */
#define AID_CACHE         2001  /* cache access */
#define AID_DIAG          2002  /* access to diagnostic resources */

/* The 3000 series are intended for use as supplemental group id's only.
 * They indicate special Android capabilities that the kernel is aware of. */
#define AID_NET_BT_ADMIN  3001  /* bluetooth: create any socket */
#define AID_NET_BT        3002  /* bluetooth: create sco, rfcomm or l2cap sockets */
#define AID_INET          3003  /* can create AF_INET and AF_INET6 sockets */
#define AID_NET_RAW       3004  /* can create raw INET sockets */
#define AID_NET_ADMIN     3005  /* can configure interfaces and routing tables. */

#define AID_MISC          9998  /* access to misc storage */
#define AID_NOBODY        9999

#define AID_APP          10000 /* first app user */

static struct android_id_info {
    const char *name;
    unsigned aid;
} android_ids[] = {
    { "root",      AID_ROOT, },
    { "system",    AID_SYSTEM, },
    { "radio",     AID_RADIO, },
    { "bluetooth", AID_BLUETOOTH, },
    { "graphics",  AID_GRAPHICS, },
    { "input",     AID_INPUT, },
    { "audio",     AID_AUDIO, },
    { "camera",    AID_CAMERA, },
    { "log",       AID_LOG, },
    { "compass",   AID_COMPASS, },
    { "mount",     AID_MOUNT, },
    { "wifi",      AID_WIFI, },
    { "dhcp",      AID_DHCP, },
```

```
    { "adb",       AID_ADB, },
    { "install",   AID_INSTALL, },
    { "media",     AID_MEDIA, },
    { "drm",       AID_DRM, },
    { "drmio",     AID_DRMIO, },
    { "shell",     AID_SHELL, },
    { "cache",     AID_CACHE, },
    { "diag",      AID_DIAG, },
    { "net_bt_admin", AID_NET_BT_ADMIN, },
    { "net_bt",    AID_NET_BT, },
    { "sdcard_rw", AID_SDCARD_RW, },
    { "vpn",       AID_VPN, },
    { "keystore",  AID_KEYSTORE, },
    { "usb",       AID_USB, },
    { "inet",      AID_INET, },
    { "net_raw",   AID_NET_RAW, },
    { "net_admin", AID_NET_ADMIN, },
    { "misc",      AID_MISC, },
    { "nobody",    AID_NOBODY, },
```

附件