Android 系统属性SystemProperty分析

# Android System Property

**一 System Property**

　　代码中大量存在：SystemProperties.set()/SystemProperties.get()；通过这两个接口可以对系统的属性进行读取/设置，

顾名思义系统属性，肯定对整个系统全局共享。通常程序的执行以进程为单位各自相互独立，如何实现全局共享呢？

System Properties是怎么一回事，又是如何实现的呢？

       属性系统是android的一个重要特性。它作为一个服务运行，管理系统配置和状态。所有这些配置和状态都是属性。

每个属性是一个键值对（key/value pair），其类型都是字符串。

这些属性可能是有些资源的使用状态，进程的执行状态，系统的特有属性……

**可以通过命令adb shell ：**

　　　　getprop查看手机上所有属性状态值。

       或者 getprop init.svc.bootanim制定查看某个属性状态

       使用setprop init.svc.bootanim start 设置某个属性的状态

**特别属性 :**

　　如果属性名称以“ro.”开头，那么这个属性被视为只读属性。一旦设置，属性值不能改变。

　　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。

　　　　（这是很巧妙的。 netresolve模块的使用这个属性来追踪在net.\*属性上的任何变化。）

　　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，与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。

**二 framework访问系统服务流程**

　　framework通过SystemProperties接口操作系统属性，SystemProperties通过JNI调用访问系统属性。

　　\frameworks\base\core\java\android\os\ SystemProperties.java：

[复制代码](javascript:void(0);)

public class SystemProperties

{

//JNI

private static native String native\_get(String key, String def);

private static native void native\_set(String key, String def);

public static String get(String key, String def) {

return native\_get(key, def);

}

public static void set(String key, String val) {

native\_set(key, val);

}

}

[复制代码](javascript:void(0);)

**Jni代码位置：**

\frameworks\base\core\jni\android\_os\_SystemProperties.cpp

**获取系统属性** 阻塞方式：

static jstring SystemProperties\_getSS()

{

len = property\_get(key, buf, "");

}

**操作在\bionic\libc\bionic\system\_properties.c中：**

int \_\_system\_property\_get(const char \*name, char \*value)

{

//数据已经存储在内存中\_\_system\_property\_area\_\_ 等待读取完返回

const prop\_info \*pi = \_\_system\_property\_find(name);

return \_\_system\_property\_read(pi, 0, value);

}

       进程启动后数据已经将系统属性数据读取到相应的共享内存中，保存在全局变量\_\_system\_property\_area\_\_；

进程之间都是独立的，系统属性数据是如何读取到当前进程空间中的呢？后续介绍。

**设置属性**异步socket通信：

[复制代码](javascript:void(0);)

int \_\_system\_property\_set(const char \*key, const char \*value)

{

msg.cmd = PROP\_MSG\_SETPROP;

strlcpy(msg.name, key, sizeof msg.name);

strlcpy(msg.value, value, sizeof msg.value);

err = send\_prop\_msg(&msg);

}

static int send\_prop\_msg(prop\_msg \*msg)

{

//sokcet 通信 /dev/socket/property\_service

s = socket(AF\_LOCAL, SOCK\_STREAM, 0);

connect(s, (struct sockaddr \*) &addr, alen)

send(s, msg, sizeof(prop\_msg), 0)

close(s);

}

[复制代码](javascript:void(0);)

　　通过socket向property\_service发送消息，property\_service运行在哪里呢？

**三 Property Service创建服务端socket**

**init进程启动监听过程中：**\system\core\init\Init.c

[复制代码](javascript:void(0);)

int main(int argc, char \*\*argv)

{

//加入到action queue队列

queue\_builtin\_action(property\_service\_init\_action, "property\_service\_init");

for(;;)

//执行action queue队列

//接收通过socket向property service 发送的数据;

nr = poll(ufds, fd\_count, timeout);

……

handle\_property\_set\_fd();

}

static int property\_service\_init\_action(int nargs, char \*\*args)

{

start\_property\_service();

}

\system\core\init\property\_service.c：

void start\_property\_service(void)

{

//加载属性配置文件

load\_properties\_from\_file(PROP\_PATH\_SYSTEM\_BUILD);

load\_properties\_from\_file(PROP\_PATH\_SYSTEM\_DEFAULT);

load\_properties\_from\_file(PROP\_PATH\_LOCAL\_OVERRIDE);

load\_persistent\_properties();

//创建socket资源 并绑定

fd = create\_socket(PROP\_SERVICE\_NAME, SOCK\_STREAM, 0666, 0, 0);

//监听

listen(fd, 8);

}

[复制代码](javascript:void(0);)

**Property Service 是运行在init守护进程中。**

　　接收到消息之后干什么，还是要先弄清楚整个Property Service是如何实现的呢，后续介绍。

先看看Property Service接收到消息后的处理。

**四 Property Service 监听socket处理**

**Property Service监听socket消息的处理过程：**

[复制代码](javascript:void(0);)

void handle\_property\_set\_fd()

{

//等待建立通信

s = accept(property\_set\_fd, (struct sockaddr \*) &addr, &addr\_size)

//获取套接字相关信息 uid gid

getsockopt(s, SOL\_SOCKET, SO\_PEERCRED, &cr, &cr\_size);

//接收属性设置请求消息

recv(s, &msg, sizeof(msg), 0);

//处理消息

switch(msg.cmd) {

case PROP\_MSG\_SETPROP:

//通过设置系统属性 处理ctl.开头消息

if(memcmp(msg.name,"ctl.",4) == 0)

{

//权限检测

if (check\_control\_perms(msg.value, cr.uid, cr.gid))

{

handle\_control\_message((char\*) msg.name + 4, (char\*) msg.value);

}

} else

{

//更改系统属性值

if (check\_perms(msg.name, cr.uid, cr.gid))

{

property\_set((char\*) msg.name, (char\*) msg.value);

}

}

break;

}

close(s);

}

[复制代码](javascript:void(0);)

**通过设置系统属性启动/关闭Service：**

权限判断：

[复制代码](javascript:void(0);)

static int check\_control\_perms(const char \*name, unsigned int uid, unsigned int gid)

{

// system /root用户直接有权限

if (uid == AID\_SYSTEM || uid == AID\_ROOT)

return 1;

//查询用户名单，判断是否存在表中并具有对应权限

for (i = 0; control\_perms[i].service; i++) {

if (strcmp(control\_perms[i].service, name) == 0) {

if ((uid && control\_perms[i].uid == uid) ||

(gid && control\_perms[i].gid == gid)) {

return 1;

}

}

}

return 0;

}

[复制代码](javascript:void(0);)

**所以如果想要应用有权限启动/关闭某Native Service：**

　　需要具有system/root权限

　　找到对应应用uid gid，将应用名称加入到control\_perms列表中

**处理消息 可以通过设置系统属性 改变服务的执行状态 start/stop：**

[复制代码](javascript:void(0);)

void handle\_control\_message(const char \*msg, const char \*arg)

{

if (!strcmp(msg,"start")) {

msg\_start(arg);

} else if (!strcmp(msg,"stop")) {

msg\_stop(arg);

} else if (!strcmp(msg,"restart")) {

msg\_stop(arg);

msg\_start(arg);

}

}

static void msg\_start(const char \*name)

{

service\_start(svc, args);

}

void service\_start(struct service \*svc, const char \*dynamic\_args){

//创建进程启动服务

pid = fork();

execve(svc->args[0], (char\*\*) svc->args, (char\*\*) ENV);

//修改服务的系统属性 执行状态

notify\_service\_state(svc->name, "running");

}

[复制代码](javascript:void(0);)

**连着前面就是ctr.start和ctr.stop系统属性：**用来启动和停止服务的。

例如：

　　　　// start boot animation

　　　　property\_set("ctl.start", "bootanim");

在init.rc中表明服务是否在开机时启动：

service adbd /sbin/adbd

　　　　class core

　　　　disabled //不自动启动

**启动服务的时候会判断：**

[复制代码](javascript:void(0);)

static void service\_start\_if\_not\_disabled(struct service \*svc)

{

//判断是否启动

if (!(svc->flags & SVC\_DISABLED)) {

service\_start(svc, NULL);

}

}

[复制代码](javascript:void(0);)

**修改系统属性值：**

[复制代码](javascript:void(0);)

static int check\_perms(const char \*name, unsigned int uid, unsigned int gid)

{

//进行权限检测

for (i = 0; property\_perms[i].prefix; i++) {

int tmp;

if (strncmp(property\_perms[i].prefix, name,

strlen(property\_perms[i].prefix)) == 0) {

if ((uid && property\_perms[i].uid == uid) ||

(gid && property\_perms[i].gid == gid)) {

return 1;

}

}

}

return 0;

}

[复制代码](javascript:void(0);)

**看这个修改系统属性权限表：**

[复制代码](javascript:void(0);)

property\_perms[] = {

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

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

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

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

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

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

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

{ "persist.service.", AID\_SYSTEM, 0 },

……

{ NULL, 0, 0 }

};

[复制代码](javascript:void(0);)

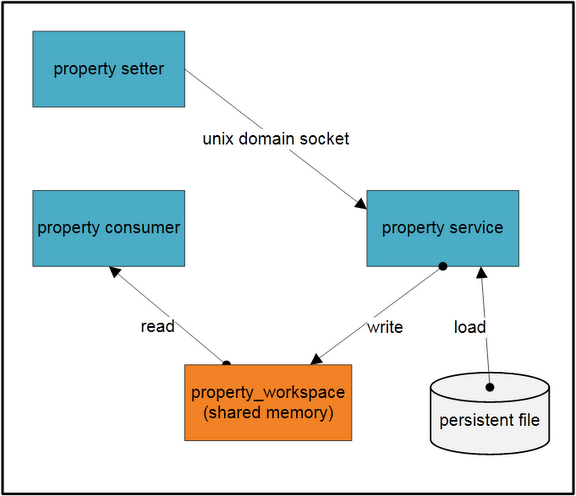
**指定了特定的用户有用修改 带有某些前缀的系统属性值。**

　　到这里基本就是Property对外的基本工作流程，Property Service内部具体如何实现，操作运行，

　　跨进程空想内存等问题仍未清除是如何处理的。

**五 属性系统设计**

         属性系统的上层架构如下图所示：



　　Property Service运行在init进程中，开机从属性文件中加载到共享内存中；设置系统属性通过socket与Property Service通信。

　　Property Consumer进程将存储系统属性值的共享内存，加载到当前进程虚拟空间中，实现对系统属性值的读取。

　　Property Setter进程修改系统属性，通过socket向Property Service发送消息，更改系统属性值。

**六 属性系统实现**

       属性系统设计的关键就是：跨进程共享内存的实现。

**下面将看看属性系统实现具体过程：**

**Init进程执行：**

[复制代码](javascript:void(0);)

int main(int argc, char \*\*argv){

//将属性系统初始化函数加入action queue

　　queue\_builtin\_action(property\_init\_action, "property\_init");

　　for(;;)

}

static int property\_init\_action(int nargs, char \*\*args)

{

property\_init(load\_defaults);

}

[复制代码](javascript:void(0);)

**初始化Property Service：**

       \system\core\init\property\_service.c

[复制代码](javascript:void(0);)

void property\_init(bool load\_defaults)

{

//初始化共享内存空间

init\_property\_area();

//加载属性文件

　　load\_properties\_from\_file(PROP\_PATH\_RAMDISK\_DEFAULT);

}

[复制代码](javascript:void(0);)

**初始化共享内存空间：**

[复制代码](javascript:void(0);)

static int init\_property\_area(void)

{

//创建匿名内存空间PA\_SIZE = 32768

init\_workspace(&pa\_workspace, PA\_SIZE)

//将内存区域分成两部分：属性系统基本信息和属性键值对

pa\_info\_array = (void\*) (((char\*) pa\_workspace.data) + PA\_INFO\_START);

//初始化属性系统信息

pa = pa\_workspace.data;

memset(pa, 0, PA\_SIZE);

pa->magic = PROP\_AREA\_MAGIC;

pa->version = PROP\_AREA\_VERSION;

/\* plug into the lib property services \*/

\_\_system\_property\_area\_\_ = pa;

}

[复制代码](javascript:void(0);)

**\_\_system\_property\_area\_\_：**

　　每个进程都会使用此变量，指向系统属性共享内存区域，访问系统属性，很重要。

位于：\bionic\libc\bionic\system\_properties.c中，属于bionic库。后面将介绍各进程如何加载共享内存。

**将文件作为共享内存映射到进程空间内存使用：**

[复制代码](javascript:void(0);)

static int init\_workspace(workspace \*w, size\_t size)

{

//dev is a tmpfs是一种虚拟内存文件系统

int fd = open("/dev/\_\_properties\_\_", O\_RDWR | O\_CREAT, 0600);

//将文件映射为共享进程空间内存 使其可以与操作内存方式一致

void \*data = mmap(NULL, size, PROT\_READ | PROT\_WRITE,

**MAP\_SHARED**, fd, 0);

close(fd);

//删除文件

fd = open("/dev/\_\_properties\_\_", O\_RDONLY);

unlink("/dev/\_\_properties\_\_");

//保存fd size 将作为环境变量传递给每个进程

w->data = data; w->size = size; w->fd = fd;

}

[复制代码](javascript:void(0);)

**加载系统属性默认数据文件：**

[复制代码](javascript:void(0);)

#define PROP\_PATH\_RAMDISK\_DEFAULT "/default.prop"

static void load\_properties\_from\_file(const char \*fn)

{

//读取系统属性键值对数据写入到共享内存中

data = read\_file(fn, &sz);

load\_properties(data);

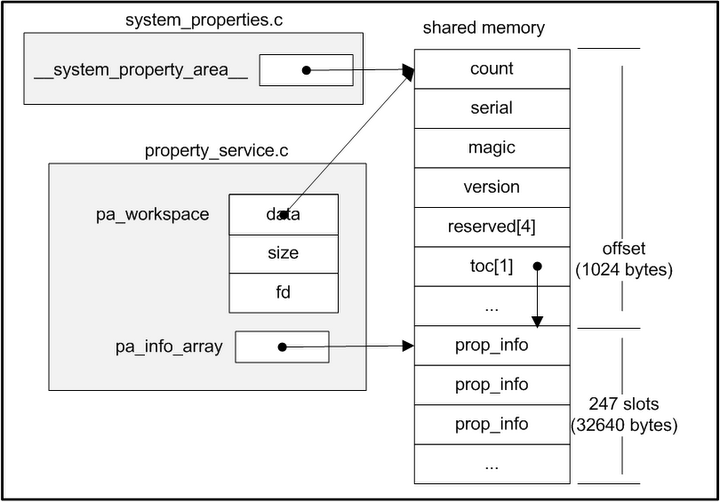
}

[复制代码](javascript:void(0);)

　　加上上面所述：Property Service Socket资源的创建，来监听socket通信连接设置系统属性，

　　在Init进程中Property Service完成了初始化。

**将得到该内存区域数据结构：**



**七 进程共享系统属性内存空间实现**

　　Property Service运行于init进程中，将文件映射为创建一块共享内存空间，但在整个系统中，

其他进程也能够读取这块内存映射到当前进程空间中，是如何实现的呢？

**Service进程启动：**将共享内存空间fd size作为环境变量传递给新创建进程

[复制代码](javascript:void(0);)

void service\_start(struct service \*svc, const char \*dynamic\_args)

{

//创建进程

pid = fork();

if (pid == 0) {

if (properties\_inited()) {

//获取系统属性空间文件描述

get\_property\_workspace(&fd, &sz);

//dup最小的可用文件描述符

sprintf(tmp, "%d,%d", dup(fd), sz);

//加入ANDROID\_PROPERTY\_WORKSPACE环境变量到ENV

　　　　　　　　//包含共享内存fd

add\_environment("ANDROID\_PROPERTY\_WORKSPACE", tmp);

}

//执行程序 传递环境变量ENV

execve(svc->args[0], (char\*\*) svc->args, (char\*\*) ENV)

//设置Service系统属性

notify\_service\_state(svc->name, "running");

}

}

void get\_property\_workspace(int \*fd, int \*sz)

{

\*fd = pa\_workspace.fd;

\*sz = pa\_workspace.size;

}

[复制代码](javascript:void(0);)

共享内存空间fd size作为环境变量传递给新创建进程后，将在何处使用呢？

**将系统属性内存空间映射到当前进程虚拟空间：**

**进程在启动时，会加载动态库bionic libc库：**

　　\bionic\libc\bionic\libc\_init\_dynamic.c中：

　　void \_\_attribute\_\_((constructor)) \_\_libc\_preinit(void);

**根据GCC的constructor/destructor属性：**

　　　　给一个函数赋予constructor或destructor，其中constructor在main开始运行之前被调用，

　　destructor在main函数结束后被调用。如果有多个constructor或destructor，可以给每个constructor

　　或destructor赋予优先级，对于constructor，优先级数值越小，运行越早。destructor则相反。

**多个constructor需要加优先级：**

[复制代码](javascript:void(0);)

\_\_attribute\_\_((**constructor**(1))) void func1()

{

printf("in constructor of foo\n");

}

\_\_attribute\_\_((**constructor**(2))) void func2()

{

printf("in constructor of foo1\n");

}

\_\_attribute\_\_((**destructor**)) void bar()

{

printf("in constructor of bar\n");

}

[复制代码](javascript:void(0);)

**\_\_libc\_preinit在bionic libc库加载的时候会被调用：**

[复制代码](javascript:void(0);)

void \_\_libc\_preinit(void)

{

\_\_libc\_init\_common(elfdata);

}

void \_\_libc\_init\_common(uintptr\_t \*elfdata)

{

\_\_system\_properties\_init();

}

int \_\_system\_properties\_init(void)

{

prop\_area \*pa; int s, fd; unsigned sz; char \*env;

//获取环境变量ANDROID\_PROPERTY\_WORKSPACE

　　//与上面init进程中设置对应

env = getenv("ANDROID\_PROPERTY\_WORKSPACE");

//共享内存文件描述符 内存大小

fd = atoi(env);

sz = atoi(env + 1);

//将文件描述符映射到当前进程虚拟空间内存，实现共享内存

pa = mmap(0, sz, PROT\_READ, MAP\_SHARED, fd, 0);

//全局变量指向共享系统属性内存首地址

\_\_system\_property\_area\_\_ = pa;

}

[复制代码](javascript:void(0);)

　　这就是整个System Property的访问交互和实现过程，具体请参考源码。

参考文档：

<http://blog.chinaunix.net/uid-20459533-id-3168973.html>

<http://www.cnblogs.com/simonshi/archive/2010/04/08/1707516.html>

<http://blog.csdn.net/zhangchiytu/article/details/7539101>