# 实现修改Android系统的开机动画

Android系统中，与开机动画相关的文件都放在一个叫bootanimation.zip的压缩包文件内，bootanimation.zip随system.img一起发布。需要放到system/media目录中.如果我们要查找系统自带的bootanimation.zip文件，可以将system目录还原，在其下的media目录下存在一个bootanimation.zip文件，替换该文件，我们就能完成开机动画的修改。bootanimation.zip文件解压看看里面的具体包含的内容：首先包含一个desc.txt文件，还会包含若干类似part0，part1的目录，其中desc.txt文件是必须的，part0、part1等目录至少要存在一个。part0、part1目录中存放的是图像文件，诸如001.png、002.png类似的命名有规律的图像文件。Android系统中读取这些静态图像，并按一定的显示规律，和频率产生动画效果，desc.txt文件就是用来描述加载规律和频率信息的文件。

为了使Android拥有Root权限，需要在代码中执行su命令。在执行su命令的过程中会创建一个新的拥有root权限的进程，通过该进程进行的任何操作都是在root权限下进行的

//执行su命令，并创建一个新进程（Process对象）

Process process = Runtime.getRuntime().exec("su");

//获取新进程的OutputStream对象，可以通过该对象发出要执行的命令

OutputStream os = process.getOutputStream();

//获取新进程的InputStream对象，可以通过该对象获取命令执行后返回的数据

InputStream is = process.getInputStream();

......

最后还要实现设备的重启，在Android系统中，要实现重启有两种方式：

1. 执行reboot命令.执行reboot命令需要root权限。也就是说，只要拥有了root权限，任何应用程序都可以重启Android设备

2. 调用PowerManager.reboot命令。该种重启方式只有System用户能使用，自由system用户才允许设置android.permission.REBOOT权限

PowerManager pm = (PowerManager)getSystemService(Context.POWER\_SERVICE);

pm.reboot("change boot animation");

最后执行重启设备命令需要在清单配置文件中配置android.permission.REBOOT权限

## 源码分析

### 调用入口

Android系统的开机动画是由应用程序bootanimation来实现的，它位于/system/bin目录下，bootanimation在init.rc中的定义如下：

1. service bootanim /system/bin/bootanimation
2. class core
3. user graphics
4. group graphics audio
5. disabled
6. oneshot

可见，由于设置为"disable"，该应用在init启动过程中是不会启动的，需要其他地方显示的调用才能启动。那是什么时候启动的呢？当SurfaceFlinger服务启动时，会修改系统属性值ctl.start，通知init进程启动bootanimation。

在早期的Android版本中，SurfaceFlinger服务是由SystemServer启动的。但在Android5.1中，该服务是init进程启动过程中就启动了。在init.rc中能看到对该服务的描述：

1. service surfaceflinger /system/bin/surfaceflinger
2. class core
3. user system
4. group graphics drmrpc
5. onrestart restart zygote

SurfaceFlinger服务源码路径为：frameworks\native\services\surfaceflinger

服务的入口在main\_surfaceflinger.cpp中，具体为：

1. int main(int, char\*\*) {
2. // When SF is launched in its own process, limit the number of
3. // binder threads to 4.
4. ProcessState::self()-**>**setThreadPoolMaxThreadCount(4);
6. // start the thread pool
7. sp**<ProcessState>** ps(ProcessState::self());
8. ps-**>**startThreadPool();
10. // instantiate surfaceflinger
11. sp**<SurfaceFlinger>** flinger = new SurfaceFlinger();
13. #if defined(HAVE\_PTHREADS)
14. setpriority(PRIO\_PROCESS, 0, PRIORITY\_URGENT\_DISPLAY);
15. #endif
16. set\_sched\_policy(0, SP\_FOREGROUND);
18. // initialize before clients can connect
19. flinger-**>**init();
21. // publish surface flinger
22. sp**<IServiceManager>** sm(defaultServiceManager());
23. sm-**>**addService(String16(SurfaceFlinger::getServiceName()), flinger, false);
25. // run in this thread
26. flinger-**>**run();
28. return 0;
29. }

主要工作是：新建一个SurfaceFlinger对象，然后调用其中的init()方法，最后调用其中的run()方法。

下面主要看一下SurfaceFlinger::init()方法，具体代码为：

1. void SurfaceFlinger::init() {
2. ALOGI(  "SurfaceFlinger's main thread ready to run. "
3. "Initializing graphics H/W...");
5. ......
7. // start boot animation
8. startBootAnim();
9. }

可以看到，最后调用了startBootAnim()。该函数代码如下：

1. void SurfaceFlinger::startBootAnim() {
2. // start boot animation
3. property\_set("service.bootanim.exit", "0");
4. property\_set("ctl.start", "bootanim");
5. }

可见，将系统属性ctl.start的值设置为"bootanim"。

回到init进程的init.c的main函数中：

1. **int** main(**int** argc, **char** \*\*argv) {
2. **for**(;;) {
3. nr = poll(ufds, fd\_count, timeout);
4. **if** (nr <= 0)
5. **continue**;
6. **for** (i = 0; i < fd\_count; i++) {
7. **if** (ufds[i].revents & POLLIN) {
8. **if** (ufds[i].fd == get\_property\_set\_fd())
9. handle\_property\_set\_fd();
10. **else** **if** (ufds[i].fd == get\_keychord\_fd())
11. handle\_keychord();
12. **else** **if** (ufds[i].fd == get\_signal\_fd())
13. handle\_signal();
14. }
15. }
16. }
17. }

可以看到，init进程会使用poll机制来轮询事件，其中一个事件是系统属性值被修改。得到该事件后，会执行handle\_property\_set\_fd()，代码如下：

1. if(memcmp(msg.name,"ctl.",4) == 0) {
2. // Keep the old close-socket-early behavior when handling
3. // ctl.\* properties.
4. close(s);
5. if (check\_control\_mac\_perms(msg.value, source\_ctx)) {
6. handle\_control\_message((char\*) msg.name + 4, (char\*) msg.value);
7. } else {
8. ERROR("sys\_prop: Unable to %s service ctl [%s] uid:%d gid:%d pid:%d\n",
9. msg.name + 4, msg.value, cr.uid, cr.gid, cr.pid);
10. }
11. }

该函数会进一步执行handle\_control\_message()，传入的参数msg.name=ctl.start，msg.value=bootanim。

1. void handle\_control\_message(const char \*msg, const char \*arg)
2. {
3. if (!strcmp(msg,"start")) {
4. msg\_start(arg);
5. } else if (!strcmp(msg,"stop")) {
6. msg\_stop(arg);
7. } else if (!strcmp(msg,"restart")) {
8. msg\_restart(arg);
9. } else {
10. ERROR("unknown control msg '%s'\n", msg);
11. }
12. }

由于msg == "start"，handle\_control\_message进一步执行msg\_start（），且传入的arg参数等于bootanim。msg\_start代码如下：

1. static void msg\_start(const char \*name)
2. {
3. struct service \*svc = NULL;
4. char \*tmp = NULL;
5. char \*args = NULL;
7. if (!strchr(name, ':'))
8. svc = service\_find\_by\_name(name);
9. else {
10. tmp = strdup(name);
11. if (tmp) {
12. args = strchr(tmp, ':');
13. \*args = '\0';
14. args++;
16. svc = service\_find\_by\_name(tmp);
17. }
18. }
20. if (svc) {
21. service\_start(svc, args);
22. } else {
23. ERROR("no such service '%s'\n", name);
24. }
25. if (tmp)
26. free(tmp);
27. }

该函数首先调用service\_find\_by\_name()，从service\_list中查询要启动的服务是否有存在，若存在，返回服务的相关信息。因为init.rc中有bootanimation的定义，因此在init进程执行parse\_config()时，会将该服务添加到service\_list中，所以bootanimation应用是存在的。然后，如果找到了该服务，就调用service\_start启动服务。

到此，bootanimation应用就启动了。

### 开机动画的显示过程

http://blog.csdn.net/u010753159/article/details/51325500