Android 7.1.1中SystemProperties详解

原创 2017年03月28日 20:02:01

标签： SystemProperties (http://so.csdn.net/so/search/s.do?q=SystemProperties&t=blog) / init (http://so.csdn.net/so/search/s.do?q=init&t=blog) / build.prop (http://so.csdn.net/so/search/s.do?q=build.prop&t=blog)

Android系统源码中，存在大量的SystemProperties.get或SystemProperties.set，通过这两个接口可以对系统的属性进行读取/设置，看着挺简单的就是调用get或set就能获取或设置系统属性，其实并不然。曾经也遇到过有关的坑，所以就总结了下，这样以后自己就不会在再次入坑了，接下来了正题吧

1、SystemProperties的使用

SystemProperties的使用很简单，在SystemProperties.java中所以方法都是static，直接通过SystemProperties.get(String key)或SystemProperties.set(String key, String val)就可以了，系统属性都是以键值对的形式存在即name和value

需要注意的是对name和value的length是有限制的，name的最大长度是31，value最大长度是91，具体定义如下：

[java]

```
1.  //frameworks/base/core/java/android/os/SystemProperties.java
2.  public class SystemProperties
3.  {
4.      public static final int PROP_NAME_MAX = 31;
5.      public static final int PROP_VALUE_MAX = 91;
6.      ...
7.      private static native String native_get(String key);
8.      private static native void native_set(String key, String def);
9.      /**
10.      * Get the value for the given key.
11.      * @return an empty string if the key isn't found
12.      * @throws IllegalArgumentException if the key exceeds 32 characters
13.      */
14.     public static String get(String key) {
15.         if (key.length() > PROP_NAME_MAX) {
16.             throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
17.         }
18.         return native_get(key);
19.     }
20.     ...
21.     /**
22.      * Set the value for the given key.
23.      * @throws IllegalArgumentException if the key exceeds 32 characters
24.      * @throws IllegalArgumentException if the value exceeds 92 characters
25.      */
26.     public static void set(String key, String val) {
27.         if (key.length() > PROP_NAME_MAX) {
28.             throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
29.         }
30.         if (val != null && val.length() > PROP_VALUE_MAX) {
31.             throw new IllegalArgumentException("val.length > " +
```

```
                PROP_VALUE_MAX);
        }
        native_set(key, val);
    }
    ...
}
```

在调用get或set时，都是通过调用native方法去操作的，开始本来想一笔带过的，还是看看native方法中的具体流程吧，如果不感兴趣可以直接看第三条

ps:在调用SystemProperties.set时所在的apk uid必须在system group，否则设置属性会报错，在manifest配置下行：
`android:sharedUserId="android.uid.system (http://androidxref.com/7.1.1_r6/s?path=android.uid.system&project=packages)"`

2、SystemProperties中方法具体实现

SystemProperties.java中所有native方法都是在android_os_SystemProperties.cpp里实现，先来看看java中个方法是怎么和cpp中方法对应起来的

```cpp
//frameworks/base/core/jni/android_os_SystemProperties.cpp
static const JNINativeMethod method_table[] = {
    { "native_get", "(Ljava/lang/String;)Ljava/lang/String;",
      (void*) SystemProperties_getS },
    { "native_get", "(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;",
      (void*) SystemProperties_getSS },
    { "native_get_int", "(Ljava/lang/String;I)I",
      (void*) SystemProperties_get_int },
    { "native_get_long", "(Ljava/lang/String;J)J",
      (void*) SystemProperties_get_long },
    { "native_get_boolean", "(Ljava/lang/String;Z)Z",
      (void*) SystemProperties_get_boolean },
    { "native_set", "(Ljava/lang/String;Ljava/lang/String;)V",
      (void*) SystemProperties_set },
    { "native_add_change_callback", "()V",
      (void*) SystemProperties_add_change_callback },
};

int register_android_os_SystemProperties(JNIEnv *env)
{
    return RegisterMethodsOrDie(env, "android/os/SystemProperties", method_table,
                                NELEM(method_table));
}
```

register_android_os_SystemProperties方法是在AndroidRuntime.cpp中调用的，RegisterMethodsOrDie可以简单的理解为把method_table数组里的方法一一对应起来。在android_os_SystemProperties.cpp中可以发现最终实现是不区分SystemProperties_get_int或 SystemProperties_getS，基本所有的方法都是调用property_get和property_set方法来实现的
property_get和property_set是在properties.c里实现的，如下

```cpp
//system/core/libcutils/properties.c
int property_get(const char *key, char *value, const char *default_value)
{
    int len;

    len = __system_property_get(key, value);
    if(len > 0) {
        return len;
    }
    if(default_value) {
        len = strlen(default_value);
        if (len >= PROPERTY_VALUE_MAX) {
            len = PROPERTY_VALUE_MAX - 1;
        }
        memcpy(value, default_value, len);
        value[len] = '\0';
    }
```

```
    return len;
}
int property_set(const char *key, const char *value)
{
    return __system_property_set(key, 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__,具体操作在system_properties.cpp中

```cpp
//bionic/libc/bionic/system_properties.cpp
int __system_property_get(const char *name, char *value)
{
    const prop_info *pi = __system_property_find(name);

    if (pi != 0) {
        //数据已经存储在内存中__system_property_area__ 等待读取完返回
        return __system_property_read(pi, 0, value);
    } else {
        value[0] = 0;
        return 0;
    }
}
```

设置属性通过异步socket通信，向property_service发送消息

```cpp
int __system_property_set(const char *key, const char *value)
{
    if (key == 0) return -1;
    if (value == 0) value = "";
    if (strlen(key) >= PROP_NAME_MAX) return -1;
    if (strlen(value) >= PROP_VALUE_MAX) return -1;

    prop_msg msg;
    memset(&msg, 0, sizeof msg);
    msg.cmd = PROP_MSG_SETPROP;
    strlcpy(msg.name, key, sizeof msg.name);
    strlcpy(msg.value, value, sizeof msg.value);

    const int err = send_prop_msg(&msg);
    if (err < 0) {
        return err;
    }

    return 0;
}
static int send_prop_msg(const prop_msg *msg)
{   //sokcet 通信 /dev/socket/property_service
    const int fd = socket(AF_LOCAL, SOCK_STREAM | SOCK_CLOEXEC, 0);
    if (fd == -1) {
        return -1;
    }
    //static const char property_service_socket[] = "/dev/socket/" PROP_SERVICE_NAME;
    const size_t namelen = strlen(property_service_socket);
    sockaddr_un addr;
    memset(&addr, 0, sizeof(addr));
    strlcpy(addr.sun_path, property_service_socket, sizeof(addr.sun_path));
    addr.sun_family = AF_LOCAL;
    socklen_t alen = namelen + offsetof(sockaddr_un, sun_path) + 1;
    if (TEMP_FAILURE_RETRY(connect(fd, reinterpret_cast<sockaddr*>(&addr), alen)) < 0) {
        close(fd);
        return -1;
    }

    const int num_bytes = TEMP_FAILURE_RETRY(send(fd, msg, sizeof(prop_msg), 0));

    ...
    close(fd);
    return result;
}
```

property_service是在init进程调用start_property_service启动的，在property_service.cpp中

[cpp]

```cpp
//system/core/init/property_service.cpp
void start_property_service() {
    //创建socket
    property_set_fd = create_socket(PROP_SERVICE_NAME, SOCK_STREAM | SOCK_CLOEXEC | SOCK_NONBLO
                                    0666, 0, 0, NULL);
    if (property_set_fd == -1) {
        ERROR("start_property_service socket creation failed: %s\n", strerror(errno));
        exit(1);
    }
   //监听socket
    listen(property_set_fd, 8);

    register_epoll_handler(property_set_fd, handle_property_set_fd);
}
```

还是在property_service.cpp的handle_property_set_fd()处理对应属性

[cpp]

```cpp
static void handle_property_set_fd()
{
    //等待建立通信
    if ((s = accept(property_set_fd, (struct sockaddr *) &addr, &addr_size)) < 0) {
        return;
    }

    /获取套接字相关信息 uid gid
    if (getsockopt(s, SOL_SOCKET, SO_PEERCRED, &cr, &cr_size) < 0) {
        close(s);
        ERROR("Unable to receive socket options\n");
        return;
    }
    ...
    //接收属性设置请求消息
    r = TEMP_FAILURE_RETRY(recv(s, &msg, sizeof(msg), MSG_DONTWAIT));
    //处理消息
    switch(msg.cmd) {
    case PROP_MSG_SETPROP:
        msg.name[PROP_NAME_MAX-1] = 0;
        msg.value[PROP_VALUE_MAX-1] = 0;
        //检查属性名是否合法
        if (!is_legal_property_name(msg.name, strlen(msg.name))) {
            ERROR("sys_prop: illegal property name. Got: \"%s\"\n", msg.name);
            close(s);
            return;
        }

        getpeercon(s, &source_ctx);
        //处理ctl.开头消息
        if(memcmp(msg.name,"ctl.",4) == 0) {
            // Keep the old close-socket-early behavior when handling
            // ctl.* properties.
            close(s);
            //检查权限，处理以ctl开头的属性
            if (check_control_mac_perms(msg.value, source_ctx, &cr)) {
                handle_control_message((char*) msg.name + 4, (char*) msg.value);
            } else {
                ERROR("sys_prop: Unable to %s service ctl [%s] uid:%d gid:%d pid:%d\n",
                        msg.name + 4, msg.value, cr.uid, cr.gid, cr.pid);
            }
        } else {
            //检查权限，设置对应的属性
            if (check_mac_perms(msg.name, source_ctx, &cr)) {
                property_set((char*) msg.name, (char*) msg.value);
            } else {
                ERROR("sys_prop: permission denied uid:%d  name:%s\n",
                      cr.uid, msg.name);
            }
            close(s);
        }
        ...
```

```
53.     }
54. }
```

不继续看了，后面的坑还有很多，考虑到后面还有几个方面没讲，简单说下property_set最后是调用了property_set_impl实现的，有兴趣的可以继续去跟踪

3、系统属性怎么生成的

1

Android的build.prop文件是在Android编译时刻收集的各种property（LCD density/语言/编译时间, etc.），编译完成之后，文件生成在out/target/product/<board>/system/目录下，build.prop的生成是由make系统解析build/core/Makefile完成。

3.1 Makefile中直接把$(TARGET_DEVICE_DIR)/system.prop的内容追加到build.prop中
3.2 收集ADDITIONAL_BUILD_PROPERTIES中的属性，追加到build.prop中
3.3 ADDITIONAL_BUILD_PROPERTIES又会收集PRODUCT_PROPERTY_OVERRIDES中定义的属性

在配置系统属性时，如果是在*.prop文件中配置直接是在文件添加一行“persist.timed.enable=true”就行，但在*.mk配置时就需要加上PRODUCT_PROPERTY_OVERRIDES属性,需要注意最后一个没有“\”，下面提供了一个实例

```
[plain]

1. PRODUCT_PROPERTY_OVERRIDES += \
2.     persist.timed.enable=true \
3.     persist.timed.enable=true \
4.     ... \
5.     key=value
```

4、系统属性类别和加载优先级

属性名称以“ro.”开头，被视为只读属性。一旦设置，属性值不能改变。
属性名称以“persist.”开头，当设置这个属性时，其值也将写入/data/property。
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（此处在7.0上有改动，做了相关优化，后面会说到）。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。
需要注意的一点是在Android7.0以后在mk文件中提供了一个编译宏LOCAL_INIT_RC用于将服务相关的RC文件编译到相应位置。这能确保服务定义和服务的可执行文件同时存在，避免了之前出现的服务对应的可执行程序不存在的问题。
单一的init*.rc，被拆分，服务根据其二进制文件的位置（／system，／vendor，／odm）定义到对应分区的etc／init目录中，每个服务一个rc文件。与该服务相关的触发器、操作等也定义在同一rc文件中。
在init执行mount_all指令挂载分区时，会加载这些目录中的rc文件，并在适当的时机运行这些服务和操作，具体可以参考system/core/init/readme.txt文件里面有详细的介绍。
系统属性是在init.rc中加载的，具体启动方式如下：

```
[cpp]

1.  //system/core/rootdir/init.rc
2.  on property:sys.boot_from_charger_mode=1
3.      trigger late-init
4.
5.  # Load properties from /system/ + /factory after fs mount.
6.  on load_system_props_action
7.      load_system_props  #定义在property_service.cpp
8.
9.  on load_persist_props_action
10.     load_persist_props #定义在property_service.cpp
11.     start logd
12.     start logd-reinit
13.
14. # Mount filesystems and start core system services.
```

```
on late-init
    # Load properties from /system/ + /factory after fs mount. Place
    # this in another action so that the load will be scheduled after the prior
    # issued fs triggers have completed.
    trigger load_system_props_action

    # Load persist properties and override properties (if enabled) from /data.
    trigger load_persist_props_action
```

当属性值sys.boot_from_charger_mode为1时，会触发late-init，在late-init又会触发load_system_props_action和load_persist_props_action，具体看看property_service.cpp这两个方法对应干啥了。

```[cpp]
//system/core/init/property_service.cpp
void load_system_props() {
    load_properties_from_file(PROP_PATH_SYSTEM_BUILD, NULL);//加载system/build.prop
    load_properties_from_file(PROP_PATH_VENDOR_BUILD, NULL);//加载vendor/build.prop
    load_properties_from_file(PROP_PATH_FACTORY, "ro.*");//加载factory/factory.prop
    load_recovery_id_prop();//加载recovery相关prop
}
void load_persist_props(void) {
    load_override_properties();//如果"ro.debuggable"为1加载data/local.prop里的属性
    /* Read persistent properties after all default values have been loaded. */
    load_persistent_properties();//加载/data/property里的persistent properties
}
```

当同一属性在多个文件中都有配置，先加载的会被后加载的覆盖。

5、利用系统属性动态设置程序中Log的开关

android 动态控制logcat日志开关,通过Log.isLoggable（TAG,level）方法动态控制，以FlashlightController类为例

```
private static final String TAG = "FlashlightController";
private static final boolean DEBUG = Log.isLoggable(TAG, Log.DEBUG);
```

利用adb命令设置属性值来控制该日志开关

adb shell setprop log.tag.FlashlightController DEBUG 设置该TAG的输出级别为DEBUG，则 level为DEBUG以上的都返回true

需要注意的是通过adb shell setprop设置的属性值每次重启后都会恢复之前的值，log的级别如下：

```
where <tag> is a log component tag (or * for all) and priority is:
  V    Verbose (default for <tag>)
  D    Debug (default for '*')
  I    Info
  W    Warn
  E    Error
  F    Fatal
  S    Silent (suppress all output)
```

也可以将该属性添加在data/local.prop属性文件中,不同的是只要存在local.prop，该手机重启与否属性值都存在，另外需要注意的是user版ro.debuggable=0，系统启动时不会读取/data/local.prop文件

终于写得差不多了，可能比较啰嗦，不过感觉自己对系统属性的了解更深了，中间还学习到了不少的c中常用的函数，感觉还是值得。

参考文档：

http://www.cnblogs.com/bastard/archive/2012/10/11/2720314.html (http://www.cnblogs.com/bastard/archive/2012/10/11/2720314.html)

http://androidxref.com/7.1.1_r6/xref/system/core/init/readme.txt (http://androidxref.com/7.1.1_r6/xref/system/core/init/readme.txt)

相关文章推荐

Android SystemProperties设置/取得系统属性的用法总结 (http://blog.csdn.net/ameyum…

通过调查得知，Android系统中取得/设置系统属性的用法参考以下3篇文章就足够了。 1.Android SystemProperties简介介绍了设置属性需要的权限，已经设置权限的方法...

ameyume (http://blog.csdn.net/ameyume) 2012年10月10日 15:08 91862

SystemProperties与Settings.system (http://blog.csdn.net/xxxsz/article/details/7480354)

android源码开发中，常常要用到一些全局标志或者说变量，这时候我们可以给android系统添加自己想要的属性。 1.Settings.system 这种系统属性我们经常用到，例如飞行模式的开启...

xxxsz (http://blog.csdn.net/xxxsz) 2012年04月20日 10:18 15231

Android系统属性SystemProperties在应用层的用法 (http://blog.csdn.net/lilidejing/articl…

如果你看到这篇文章了，说明你已经是资深程序员，会发现整个Android系统有很多地方有用到这个SystemProperties这个系统属性文件。 关于SystemPropert...

lilidejing (http://blog.csdn.net/lilidejing) 2016年11月22日 18:04 2283

Android——SystemProperties的应用 (http://blog.csdn.net/jscese/article/details/18700903)

一.序前文分析了build.prop这个系统属性文件的生成http://blog.csdn.net/jscese/article/details/18699155，每个属性都有一...

jscese (http://blog.csdn.net/jscese) 2014年01月24日 10:23 5686

android SystemProperties--如何配置系统属性 (http://blog.csdn.net/Bing_ka/article/deta…

实现一个上层设置中的开关控制关机时是否在充电时打开呼吸灯的需求：明确如何实现：上层写入一个系统属性的变量，在开关打开和关闭时改变变量的值(bool类型即可)，底层会去读取这个系统属性的值，从而做控...

Bing_ka (http://blog.csdn.net/Bing_ka) 2015年06月17日 09:56 1328

Android 的系统属性(SystemProperties)设置分析 (http://blog.csdn.net/netpirate/article/…

作者：徐建祥(netpirate@gmail.com)日期：2009/11/11网址：http://www.anymobile.orgAndroid 的系统属性包括两部分：文件保存的持久属性和每次开机...

netpirate (http://blog.csdn.net/netpirate) 2009年11月11日 17:39 13914

如何使用**android.os.SystemProperties**的方法 **(http://blog.csdn.net/randyrhoads/article/…**

android.os.SystemProperties在SDK的库中是没有的，需要把Android SDK\platforms\android-xx\data\layoutlib.jar文件加到当前工...

randyrhoads (http://blog.csdn.net/randyrhoads) 2012年09月18日 10:17 12984

如何使用**android.os.SystemProperties (http://blog.csdn.net/gordonyui/article/details/70…**

最近在開發一些案子，需要使用SystemProperties去提取系統的資源，在網上找了一陣子，發現要將Android SDK裡的layoutlib.jar加進專案裡，就可以呼叫了。做法如下： 1....

gordonyui (http://blog.csdn.net/gordonyui) 2011年12月05日 12:08 19423

Systemproperties用法 **(http://blog.csdn.net/cupidove/article/details/7827778)**

Systemproperties类在android.os下，但这个类是隐藏的， 上层程序开发无法直接使用。其实用java的反射机制是可以使用这个类。何谓java反射机制，请自行研究学习，在此不做介绍，...

cupidove (http://blog.csdn.net/cupidove) 2012年08月03日 16:33 14132

Android : 反射机制获取或设置系统属性（**SystemProperties**） **(http://blog.csdn.net/dor…**

android.os.SystemProperties 提供了获取和设置系统属性的方法，但是这个类被隐藏了，应用开发时无法直接访问，可以通过反射的机制进行操作。...

doris_d (http://blog.csdn.net/doris_d) 2016年11月01日 18:01 4377

Android SystemProperties 深入分析 **(/wangbaochu/article/details/44647963)**

1.Android SystemProperties简介介绍了设置属性需要的权限，已经设置权限的方法。 Systemproperties类在android.os下，但这个类是隐藏的，上层...

wangbaochu (http://blog.csdn.net/wangbaochu) 2015-03-26 11:50 1425

Android SystemProperties系统属性详解 **(/wdong_love_cl/article/details/52404692)**

Systemproperties类在android.os下，但这个类是隐藏的，上层程序开发无法直接使用，用Java的反射机制就可以了。Java代码中创建与修改android属性用Systemprope...

wdong_love_cl (http://blog.csdn.net/wdong_love_cl) 2016-09-02 00:26 2583

Android SystemProperties设置/取得系统属性的用法总结 **(/shanghaibao123/article/deta…**

通过调查得知，Android系统中取得/设置系统属性的用法参考以下3篇文章就足够了。 1.Android SystemProperties简介介绍了设置属性需要的权限...

shanghaibao123 (http://blog.csdn.net/shanghaibao123) 2015-05-22 11:33 1323

Android7.0 init进程源码分析 **(/gaugamela/article/details/52133186)**

主要对Android 7.0中，init进程的源码进行分析。

Gaugamela (http://blog.csdn.net/Gaugamela) 2016-08-07 20:07 7040

《深入理解Android 卷1》读书笔记 （一）—— Android Init之属性服务 （property_ser...

老实说，觉得自己讲不好这部分，建议读者参考《深入理解Android 卷1》 或者网上其他文章。此处，我只是略提一下。看到这个property，让我想起了注册表，也想起来以前工作中保存用户...

happy08god (http://blog.csdn.net/happy08god) 2014-02-05 13:56 5495

Android 7.1 竖屏转横屏全过程实现-基于高通平台 (/eliot_shao/article/details/70766283)

背景：调试使用一款800x1280分辨率的屏，客户希望横屏使用（1280x800），且没有旋转过程，看起来就是横屏显示一样。修改主要涉及几个方面，第一是LK阶段的图片，第二是开机动画的显示（/sy...

seek_0380 (http://blog.csdn.net/seek_0380) 2017-04-25 19:39 1704

property_get/property_set (/xujianqun/article/details/6363318)

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...

xujianqun (http://blog.csdn.net/xujianqun) 2011-04-26 09:42 51028

SystemProperties.get(String key,String def)获取系统属性 (/ningchao328/article/details/...

一.SystemProperties.get(String key,String def)方法能够获取build.prop文件（即系统属性）的相关信息 1.获取build.prop文件的版本信息 St...

ningchao328 (http://blog.csdn.net/ningchao328) 2016-10-11 17:42 2143

Android启动文件init.rc文件实例语法分析 (/taanng/article/details/22653843)

1# Copyright (C) 2012 The Android Open Source Project 2# Copyright (C) 3# Copyright (C) 4# 5# IMPO...

taanng (http://blog.csdn.net/taanng) 2014-03-31 11:04 1365

Android init.rc文件解析过程详解(一) (/mk1111/article/details/16357327)

Android init.rc文件解析过程详解(一) init.c与init.rc在源码中的位置分别位于如下： 1 init.c : /system/core/in...

mk1111 (http://blog.csdn.net/mk1111) 2013-11-16 14:56 18566