如何在 android 系统里面从驱动到 app 添加一个系统服务

首先，android 系统从下到上分为这么几层
kernel → HAL -->routime(虚拟机和一些其他的库)-->framework-->app

我们要添加的服务为 LedService
在 app 中直接可以这样使用

LedManager ledManage = getSystemService(Context.LED_SERVICE);
ledManager.setOn();//设置手机灯的开关
ledManager.setOff();

要达到这个上面的目的，需要在系统的 Context.java 和 ContextImpl.java 里面做如下工作
Context.java

```
public static final String LED_SERVICE = "led";//添加该行
```

在 contextImpl.java 里面添加如下代码:

```
 public Object getSystemService(String name) {
        if (WINDOW_SERVICE.equals(name)) {
            return WindowManagerImpl.getDefault();
        } else if (LAYOUT_INFLATER_SERVICE.equals(name)) {
            synchronized (mSync) {
                LayoutInflater inflater = mLayoutInflater;
                if (inflater != null) {
                    return inflater;
                }
                mLayoutInflater = inflater =
                    PolicyManager.makeNewLayoutInflater(getOuterContext());
                return inflater;
            }
        } else if (ACTIVITY_SERVICE.equals(name)) {
            return getActivityManager();
        } else if (INPUT_METHOD_SERVICE.equals(name)) {
            return InputMethodManager.getInstance(this);
        } else if (ALARM_SERVICE.equals(name)) {
            return getAlarmManager();
        } else if (ACCOUNT_SERVICE.equals(name)) {
            return getAccountManager();
        } else if (POWER_SERVICE.equals(name)) {
            return getPowerManager();
        } else if (CONNECTIVITY_SERVICE.equals(name)) {
            return getConnectivityManager();
        } else if (THROTTLE_SERVICE.equals(name)) {
            return getThrottleManager();
        } else if (WIFI_SERVICE.equals(name)) {
            return getWifiManager();
        } else if (NOTIFICATION_SERVICE.equals(name)) {
            return getNotificationManager();
        } else if (KEYGUARD_SERVICE.equals(name)) {
            return new KeyguardManager();
        } else if (ACCESSIBILITY_SERVICE.equals(name)) {
            return AccessibilityManager.getInstance(this);
        } else if (LOCATION_SERVICE.equals(name)) {
            return getLocationManager();
        } else if (SEARCH_SERVICE.equals(name)) {
            return getSearchManager();
        } else if (SENSOR_SERVICE.equals(name)) {
            return getSensorManager();
```

```
        } else if (STORAGE_SERVICE.equals(name)) {
            return getStorageManager();
        } else if (VIBRATOR_SERVICE.equals(name)) {
            return getVibrator();
        } else if (STATUS_BAR_SERVICE.equals(name)) {
            synchronized (mSync) {
                if (mStatusBarManager == null) {
                    mStatusBarManager = new StatusBarManager(getOuterContext());
                }
                return mStatusBarManager;
            }
        } else if (AUDIO_SERVICE.equals(name)) {
            return getAudioManager();
        } else if (TELEPHONY_SERVICE.equals(name)) {
            return getTelephonyManager();
        } else if (CLIPBOARD_SERVICE.equals(name)) {
            return getClipboardManager();
        } else if (WALLPAPER_SERVICE.equals(name)) {
            return getWallpaperManager();
        } else if (DROPBOX_SERVICE.equals(name)) {
            return getDropBoxManager();
        } else if (DEVICE_POLICY_SERVICE.equals(name)) {
            return getDevicePolicyManager();
        } else if (UI_MODE_SERVICE.equals(name)) {
            return getUiModeManager();
        } else if (DOWNLOAD_SERVICE.equals(name)) {
            return getDownloadManager();
        } else if (NFC_SERVICE.equals(name)) {
            return getNfcManager();
        }else if(LED_SERVICE.equals(name))
        {
           Log.d("ContextImpl", "get LedManager success");
           return getLedManager();
        }

        return null;
    }

private LedManager getLedManager()
    {
       synchronized (mSync) {
            if (mLedManager == null) {
              mLedManager = new LedManager();
            }
        }
        return mLedManager;
    }
```

其中LedManager类的定义如下：

```
package android.app;

import android.os.RemoteException;

import android.os.ServiceManager;
import android.util.Log;

public class LedManager
{
    private static final String TAG = "LedManager";
     private ILedManager mLedService;
```

```
     public LedManager() {
        mLedService =
ILedManager.Stub.asInterface(ServiceManager.getService("led"));

     if (mLedService != null) {
             Log.i(TAG, "The LedManager object is ready.");
     }
     }
     public boolean setOn(int n) {
         boolean result = false;
         try {
             result = mLedService.setOn(n);
         } catch (RemoteException e) {
             Log.e(TAG, "RemoteException in LedManager.LedOn:", e);
         }
         return result;
     }
     public boolean setOff(int n) {
         boolean result = false;
         try {
             result = mLedService.setOff(n);
         } catch (RemoteException e) {
             Log.e(TAG, "RemoteException in LedManager.LedOff:", e);
         }
         return result;
     }
 }
```

由于我们用用LedManager取得一个系统服务LedService，所以此处就用到了一个IPC机制，通过aidl，两个进程之间进行通信，这样当操作LedManager的时候就如同操作LedService了，在这里这个aidl文件如下：
IledManager.aidl

```
package android.app;

interface ILedManager
{
    boolean setOn(int led);
    boolean setOff(int led);
}
```

此时需要修改一下系统的一个mk文件，不然我们自己的aidl文件编不进去就出错

gingerbread_rel-M76XXTSNCJNLYA61601002/frameworks/base/Android.mk

```
LOCAL_SRC_FILES += \

core/java/android/accessibilityservice/IAccessibilityServiceConnection.aidl \
    core/java/android/accessibilityservice/IEventListener.aidl \
    core/java/android/accounts/IAccountManager.aidl \
    core/java/android/accounts/IAccountManagerResponse.aidl \
    core/java/android/accounts/IAccountAuthenticator.aidl \
    core/java/android/accounts/IAccountAuthenticatorResponse.aidl \
    core/java/android/app/IActivityController.aidl \
    core/java/android/app/IActivityPendingResult.aidl \
    core/java/android/app/IActivityWatcher.aidl \
    core/java/android/app/ILedManager.aidl \
    core/java/android/app/IAlarmManager.aidl \
    core/java/android/app/IBackupAgent.aidl \
    core/java/android/app/IInstrumentationWatcher.aidl \
    core/java/android/app/INotificationManager.aidl \
    core/java/android/app/ISearchManager.aidl \
```

```
    core/java/android/app/ISearchManagerCallback.aidl \
    core/java/android/app/IServiceConnection.aidl \
    core/java/android/app/IThumbnailReceiver.aidl \
    core/java/android/app/ITransientNotification.aidl \
    core/java/android/app/IUiModeManager.aidl \
…..................
```

LedService.java的内容如下

```
package com.android.server;


import android.content.Context;

import android.os.ServiceManager;

import android.util.Log;

import android.os.IBinder;

import android.app.ILedManager;


public final class LedService extends ILedManager.Stub {


         public LedService(Context context) {

             Log.i("LedService", "Go to get LED Stub...");

             ServiceManager.addService("led", LedService.this);

             _init();

         }

         /*

          * Mokoid LED native methods.

          */

        public boolean setOn(int led) {

             Log.i("MokoidPlatform", "LED On");

         return _setOn(led);

         }

         public boolean setOff(int led) {

             Log.i("MokoidPlatform", "LED Off");

         return _setOff(led);

         }
```

```
        private static native boolean _init();

        private static native boolean _setOn(int led);

        private static native boolean _setOff(int led);

    }
```

到此为止 我们就可以使用LedManager来使用LedService的方法了
由于LedManager是在我们的app里面，是属于我们程序的进程的，但是LedService是系统启动的时候就启动的服务程序，是属于systemServer进程，不同的进程是不能通信的，android采用了aidl的方式，底层是采用binder驱动，通过共享内存来达到通信目的的。

在ledService.java里面有几个native方法，这几个方法在java层调用，但是实现是在c++层的，这里是通过jni调用来达到java代码调用c++代码的

这里的jni文件如下:
gingerbread_rel-M76XXTSNCJNLYA61601002/frameworks/base/services/jni/com_android_server_LedService.cpp

这个文件名不能随便命名，要遵循 包名_类名.cpp的原则

```
#define LOG_TAG "Dragon"
#include "jni.h"
#include "JNIHelp.h"
#include "android_runtime/AndroidRuntime.h"

#include <utils/misc.h>
#include <utils/Log.h>
#include <hardware/hardware.h>
#include <hardware/led.h>

#define LOG_NDDEBUG 0  // 定义后，LOGD能够输出
#define LOG_NIDEBUG 0  // 定义后，LOGI能够输出
#define LOG_NDEBUG 0   // 定义后，LOGV能够输出


namespace android
{

struct led_control_device_t *sLedDevice = NULL;

static jboolean setOn(JNIEnv * env ,jobject clazz,jint led)
{
    LOGI("LedService JNI: setOn() is invoked");
    if(sLedDevice == NULL)
    {
        return -1;

    }else
    {
        return sLedDevice->set_led_on(sLedDevice,led);
    }
}

static jboolean setOff(JNIEnv * env ,jobject clazz,jint led)
{
       LOGI("LedService JNI: setOff() is invoked.");

        if (sLedDevice == NULL) {
```

```
            LOGI("LedService JNI: sLedDevice was not fetched correctly.");
            return -1;
        } else {
            return sLedDevice->set_led_off(sLedDevice, led);
        }
}

/** helper APIs */
static inline int led_control_open(const struct hw_module_t* module,
         struct led_control_device_t** device) {
    return module->methods->open(module,
            LED_HARDWARE_MODULE_ID, (struct hw_device_t**)device);
}


static jboolean init(JNIEnv * evn ,jclass clazz)
{
    led_module_t * module;
    if(hw_get_module(LED_HARDWARE_MODULE_ID,(const hw_module_t **)&module) == 0)
    {
        LOGI("LedService JNI: LED Stub found.");
        if (led_control_open(&module->common, &sLedDevice) == 0) {
                    LOGI("LedService JNI: Got Stub operations.");
                     return 0;
                }
    }

    LOGE("LedService JNI: Get Stub operations failed.");
    return -1;
}

//传统的jni调用，函数名也是有一定规则的，但是android改变了这种方式，通过一下这种方式就可以把java层和c++层的代码映射起来

//第一个参数是java层要调用的方法，第二个参数是第一个参数(也就是java层调用的函数)的参数和返回值,第三个参数是c++层要调用的函数

static const JNINativeMethod gMethods[] = {
    { "_init",        "()Z", (void *)init },
    { "_setOn",        "(I)Z", (void *)setOn },
    { "_setOff",       "(I)Z", (void *)setOff },
};


//这个函数很重要，这里是注册我们上面定义的那些方法的
int register_android_server_LedService(JNIEnv *env)
{
    return jniRegisterNativeMethods(env, "com/android/server/LedService",
           gMethods, NELEM(gMethods));
}
}

这里要注意的是，
必须在下面这个函数里面调用我们上面的那个注册函数，不然编译能过，最后手机也跑不起来

gingerbread_rel-M76XXTSNCJNLYA61601002/frameworks/base/services/jni/onload.cpp

using namespace android;


extern "C" jint JNI_OnLoad(JavaVM* vm, void* reserved)
```

```
{
    JNIEnv* env = NULL;
    jint result = -1;

    if (vm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        LOGE("GetEnv failed!");
        return result;
    }
    LOG_ASSERT(env, "Could not retrieve the env!");

    register_android_server_PowerManagerService(env);
    register_android_server_InputManager(env);
    register_android_server_LightsService(env);
    register_android_server_AlarmManagerService(env);
    register_android_server_BatteryService(env);
    register_android_server_VibratorService(env);
    register_android_server_SystemServer(env);
    register_android_server_location_GpsLocationProvider(env);
    register_android_server_LedService(env);
//加上这句就好了

    return JNI_VERSION_1_4;
}
```

到这里我们就从java层彻底到了c/c++层了，在上面那个cpp文件里面的头文件有个叫做hardware/led.h的文件，这个文件的路径是
gingerbread_rel-M76XXTSNCJNLYA61601002/hardware/libhardware/include/hardware/led.h

对应的led.c路径是
gingerbread_rel-M76XXTSNCJNLYA61601002/hardware/msm7k/libdragon-led/led.c文件

这两个文件就是属于android的HAL层了

HAL层主要是为了让硬件厂商不许要公开起驱动源代码

led.c 文件的内容如下：

```
#define LOG_TAG "DragonLedStub"
```

```
#include <hardware/hardware.h>
#include <fcntl.h>
#include <errno.h>
#include <cutils/log.h>
#include <cutils/atomic.h>
#include <hardware/led.h>

#define GPG3DAT2_ON  0x4800;
#define GPG3DAT2_OFF 0x4801;

#define LOG_NDDEBUG 0  // 定义后，LOGD 能够输出
#define LOG_NIDEBUG 0  // 定义后，LOGI 能够输出
#define LOG_NDEBUG 0   // 定义后，LOGV 能够输出

int fd;

int led_device_close(struct hw_device_t * device)
{
    struct led_control_device_t * ctx = (struct led_control_device_t *)device;
    if(ctx)
    {
        free(ctx);
    }
    close(fd);
    return 0;
}
```

```
int led_on(struct led_control_device_t *dev ,int32_t led)

{

    LOGI("LED Stub: set *d off.",led);

    return 0;

}



int led_off(struct led_control_device_t *dev ,int32_t led)

{

    char buf[32] ;
    int n ;
    LOGI("LED Stub: set %d off.",led);

//这里就调用了led驱动程序
    if((fd = open("/dev/led",O_RDWR))== -1)
    {
            LOGI("LED open error");
    }
    else
    {
            LOGI("LED open ok");
            n = read(fd,buf,32);
            LOGI("LED Stub: read  %s data from led driver",buf);
            close(fd);

    }
    return 0;

}



static int led_device_open(const struct hw_module_t * module,const char*
name ,struct hw_device_t ** device)

{

//下面是填充 led_control_device_t 结构体
    struct led_control_device_t *dev;

    dev = (struct led_control_device_t *)malloc(sizeof(*dev));

    memset(dev,0,sizeof(*dev));

    dev->common.tag = HARDWARE_DEVICE_TAG;
```

```
    dev->common.version = 0;
    dev->common.module = module;
    dev->common.close = led_device_close;
    dev->set_led_on = led_on;
    dev->set_led_off = led_off;
    *device = &dev->common;

    success:
    return 0;
}

static struct hw_module_methods_t led_module_methods =
{
        open: led_device_open
};

const struct led_module_t HAL_MODULE_INFO_SYM =
{
        common:
        {
            tag:HARDWARE_MODULE_TAG,
            version_major:1,
            version_minor:0,
            id:LED_HARDWARE_MODULE_ID,
            name:"Sample LED Stud",
            author:"The Dragon Open Source Project",
```

```
            methods:&led_module_methods,

        }

};
```

下面是led驱动程序的实现代码led_driver.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/device.h>
#include <linux/kernel.h> /* printk() */
#include <linux/slab.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */
#define DRIVER_AUTHOR "ZHANG FEI LONG"
#define DRIVER_DESC   "HELLO WORLD DRIVER"

#define DEVICE_NAME "led"

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);

int led_open(struct inode *inode, struct file *filp);
int led_release(struct inode *inode, struct file*filp);
ssize_t led_read(struct file *filp, char *buf, size_t count, loff_t *f_pos);
ssize_t led_write(struct file *filp, __user char *buf,size_t     count, loff_t *f_pos);
void    led_exit(void);
int led_init(void);

struct file_operations led_fops = {
    read:    led_read,
     write:   led_write,
    open:    led_open,
    release: led_release
};

module_init(led_init);
module_exit(led_exit);

int led_major = 60;
char * led_buffer;

static struct class *led_class;
```

```
int led_init()
{
    int result;
    result = register_chrdev(led_major,DEVICE_NAME,&led_fops);

    if(result<0)
    {
        printk("<1> : LED cannot obtain major number %d \n",led_major);
        return result;
    }

    led_buffer = kmalloc(2,GFP_KERNEL);
    if(!led_buffer)
    {
        result = -ENOMEM;
        goto fail;
    }
    memset(led_buffer,0,2);
    printk("<1>Inserting  module\n");

    //注册一个类，使 mdev 可以在"/dev/"目录下面建立设备节点

    led_class = class_create(THIS_MODULE, DEVICE_NAME);

    if(IS_ERR(led_class))
    {
        printk("Err: failed in EmbedSky-leds class. \n");
        return -1;
    }

    //创建一个设备节点，节点名为 DEVICE_NAME

    device_create(led_class, NULL, MKDEV(led_major, 0), NULL,
DEVICE_NAME);

    //官方文档为 class_device_create(led_class, NULL, MKDEV(LED_MAJOR, 0),
NULL, DEVICE_NAME);

    printk(DEVICE_NAME " initialized\n");

    return result;
}

void led_exit()
{
    unregister_chrdev(led_major,"");
    if(led_buffer)
    {
        kfree(led_buffer);
    }
    printk("<1>LED Removing  module \n");
}
```

```
//在 HAL 层调用 open()函数就相当与调用下面这个 led_open 函数
int led_open(struct inode *inode,struct file *filp)
{
    printk("<1>Open device successfully\n");
    return 0;
}

int led_release(struct inode * inode,struct file * filp)
{
    printk("<1>Release device successfully\n");
    return 0;
}
//在 HAL 层调用 read()函数就相当与调用下面这个 led_read 函数
ssize_t led_read(struct file *filp,char * buf,size_t count,loff_t * f_pos)
{
    int n  = 0;
    n = copy_to_user(buf,led_buffer,sizeof(led_buffer)/sizeof(char));
      printk("<1>read from user char is %s\n",led_buffer);
    if(*f_pos ==0)
    {
        *f_pos += n;
        return n;
    }
    else
    {
        return *f_pos + n;
    }
}

ssize_t led_write(struct file *filp,char * buf,size_t count,loff_t * f_pos)
{
    int n = 0;
    n = copy_from_user(led_buffer,buf,sizeof(buf)/sizeof(char));
      printk("<1>write from user char is %s\n",led_buffer);
    return 1;
}
```

通过以上步骤就完成了从 kernel 层到 app 层添加一个服务的功能，对于上面的驱动模块的话，是通过单独编译成.ko 文件，然后 push 到手机里面通过 insmod 动态加载到内核里面的,要看内核 log 信息可以通过 adb shell dmesg 查看，上层 log 信息通过 adb shell logcat 查看,当然驱动程序也可以放在 kernel/driver/新建 led 文件夹/led.c 下面直接编译到 kernel 里面，这样就可以每次启动系统的时候自动运行驱动了，不过这样的话需要修改源代码的配置文件，这里就不说了，可以参考这里去配置
http://www.linuxidc.com/Linux/2011-04/34541.htm