JNI层——GPS

HAL层——GPS

Android系统 GPS框架分析 (JNI-HAL)

Disigned By Chaiyun

```
static JNINativeMethod sMethods[] = {
     /* name, signature, funcPtr */
    {"class_init_native", "()V", (void *)android_location_GpsLocationProvider_class_init_native},
    {"native_is_supported", "()Z", (void*)android_location_GpsLocationProvider_is_supported},
    {"native_init", "()Z", (void*)android_location_GpsLocationProvider_init},
    {"native_cleanup", "()V", (void*)android_location_GpsLocationProvider_cleanup},
    {"native_set_position_mode", "(IIIII)Z", (void*)android_location_GpsLocationProvider_set_position_mode},
    {"native_start", "()Z", (void*)android_location_GpsLocationProvider_start},
    {"native_stop", "()Z", (void*)android_location_GpsLocationProvider_stop},
```

```
struct gps_device_t {
  struct hw_device_t common;
  const GpsInterface* (*get_gps_interface)(struct gps_device_t* dev);
};
```

gps_device_t

```
android_location_GpsLocationProvider_class_init_native:
//获取module
hw_get_module(GPS_HARDWARE_MODULE_ID, (hw_module_t const**)&module);
//调用HAL层的open函数获取device
module->methods->open(module, GPS_HARDWARE_MODULE_ID, &device);
//将device类型转化为 gps_device_t
gps_device_t* gps_device = (gps_device_t *)device;
//调用成员函数，返回一个结构体 GpsInterface(后面我们可以知道此结构体填充了函数)
sGpsInterface = gps_device->get_gps_interface(gps_device);

//获取上层应用程序中java里面的reportLocation函数的方法ID
method_reportLocation = env->GetMethodID(clazz, "reportLocation", "(IDDDFFFJ)V");
```

open

```
typedef struct {
  /** set to sizeof(GpsInterface) */
  size_t      size;
  int   (*init)( GpsCallbacks* callbacks );
  int   (*start)( void );
  int   (*stop)( void );
  void  (*cleanup)( void );
  int   (*inject_time)(GpsUtcTime time, int64_t timeReference, int uncertainty);
  int  (*inject_location)(double latitude, double longitude, float accuracy);
  void  (*delete_aiding_data)(GpsAidingData flags);
} GpsInterface;//全是函数指针
```

GpsInterface

```
GpsCallbacks sGpsCallbacks = {
      sizeof(GpsCallbacks),
      location_callback,
      status_callback,
      sv_status_callback,
      nmea_callback,
      set_capabilities_callback,
      acquire_wakelock_callback,
      release_wakelock_callback,
      create_thread_callback,
      request_utc_time_callback,
};
```

```
android_location_GpsLocationProvider_init:
//sGpsInterface是从HAL层获得的device从而得到的GpsInterface结构体
//调用HAL层的init函数，参数是sGpsInterface(里面全是定义的函数)
if (!sGpsInterface || sGpsInterface->init(&sGpsCallbacks) != 0)
```

init

```
static int qemu_gps_init(GpsCallbacks* callbacks)
{
    GpsState*  s = _gps_state;///开辟空间
    gps_state_init(s);//初始化GpsState结构体
    s->callbacks = *callbacks;//初始化了callbacks 成员，后面会调用此函数
}
```

```
//主要功能：返回填充本地数据device
static struct hw_module_methods_t gps_module_methods = {
    .open = open_gps
};
```

open_gps

```
open_gps(const struct hw_module_t* module, char const* name,struct hw_device_t** device)
{
    struct gps_device_t *dev;
    //填充结构体
    dev->common.tag = HARDWARE_DEVICE_TAG;
    dev->common.version = 0;
    dev->common.module = (struct hw_module_t*)module;
    dev->get_gps_interface = gps__get_gps_interface
    *device = (struct hw_device_t*)dev
}
```

```
static const GpsInterface  qemuGpsInterface = {
    sizeof(GpsInterface),
    qemu_gps_init,
    qemu_gps_start,
    qemu_gps_stop,
    qemu_gps_cleanup,
    qemu_gps_inject_time,
    qemu_gps_inject_location,
    qemu_gps_delete_aiding_data,
    qemu_gps_set_position_mode,
    qemu_gps_get_extension,
};
```

gps__get_gps_interface

```
const GpsInterface* gps__get_gps_interface(struct gps_device_t* dev)
{
    //返回一个GpsInterface类型的已经填充好的全是函数的结构体
    return &qemuGpsInterface;
}
```

```
typedef struct {
    /** set to sizeof(GpsCallbacks) */
    size_t      size;
    gps_location_callback location_cb;
    gps_status_callback status_cb;
    gps_sv_status_callback sv_status_cb;
    gps_nmea_callback nmea_cb;
    gps_set_capabilities set_capabilities_cb;
    gps_acquire_wakelock acquire_wakelock_cb;
    gps_release_wakelock release_wakelock_cb;
    gps_create_thread create_thread_cb;
    gps_request_utc_time request_utc_time_cb;
} GpsCallbacks;
```

```
gps_state_init( GpsState*  state )//参数是上层传递的参数GpsState结构体
{
    //填充state
    state->init       = 1;
    state->control[0] = -1;
    state->control[1] = -1;
    state->fd         = -1;
    //#define  GPSHARDWARE_CHANNEL_NAME  "/dev/ttyUSB0"
    state->fd = gps_usb_channel_open(&state->channel,GPSHARDWARE_CHANNEL_NAME,
O_RDONLY);
    close(state->fd);//关闭  说明之前打开只是为了检测
    socketpair( AF_LOCAL, SOCK_STREAM, 0, state->control );//创建双向管道
    pthread_create( &state->thread, NULL, gps_state_thread, state )
            //创建线程，执行线程体函数: gps_state_thread，程序另一部分继续向下执行
}
```

gps_state_init

gps_state_thread

gps_usb_channel_open

```
int gps_usb_channel_open( QemuChannel*  channel,
                          const char*   name,
                          int           mode )
{
    printf(p, 20, "/dev/ttyUSB%d", i);
    fd = open(p, mode);//打开设备节点 /dev/ttyUSB0
    set_opt(fd,4800,8,'N',1)//初始化串口
    read(fd, p, 1024);
```

```
typedef struct {
    int                 init;
    int                 fd;
    GpsCallbacks        callbacks;
    pthread_t           thread;
    pthread_t           thread_gps_read;
    int                 control[2];
    QemuChannel         channel;
} GpsState;
```

```
static void* gps_state_thread( void*  arg )//参数是线程传来的
{
    GpsState*   state = (GpsState*) arg;
    int  epoll_fd   = epoll_create(2);//创建epoll
    int control_fd = state->control[1];
    epoll_register( epoll_fd, control_fd );
    for (;;)
    {
        //等待返回，如果返回，说明另一端向这端写了数据 。。。。。
        nevents = epoll_wait( epoll_fd, events, 2, -1 );
        do {
            //读取数据，另一端发来的,后面我们可以知道第一次cmd是START
            //CMD_QUIT 退出
            //CMD_START 开始
            //CMD_STOP 停止
            ret = read( fd, &cmd, 1 );
        } while (ret < 0 && errno == EINTR);
        if (cmd == CMD_QUIT)
        else if (cmd == CMD_START) {
            state->thread_gps_read =
                    //state是GpsState结构体调用callbacks函数，callbacks是GpsCallbacks结构体，
是从上层传递下来的结构体参数已经被赋值给了该state，再调用GpsCallbacks里的create_thread_cb函数
即回调了JNI层的create_thread_callback函数，参数是gps_read_thread线程体
                    state->callbacks.create_thread_cb("gps_read", gps_read_thread, state);
        else if (cmd == CMD_STOP)
    }
}
```

epoll_register

```
epoll_register( int  epoll_fd, int  fd );
    ev.events  = EPOLLIN;
    ev.data.fd = fd;
    do {
        //fd是control_fd是state->control[1] 监听管道的1端
        ret = epoll_ctl( epoll_fd, EPOLL_CTL_ADD, fd, &ev );
    } while (ret < 0 && errno == EINTR);
```

Write

gps_state_start

```
gps_state_start(s);
```

```
static void gps_state_start( GpsState*  s )
        char  cmd = CMD_START;
        do { ret=write( s->control[0], &cmd, 1 ); }
        while (ret < 0 && errno == EINTR);//对control[0]//有写的操作,init函数里的epoll_wait不再等待，cmd=start，会继续执行
```

gps_read_thread

```
static void* gps_read_thread(void * arg
    state->fd = gps_usb_channel_open(
                    &state->channel,
                    GPSHARDWARE_CHANNEL_NAME,
                    O_RDONLY);  //打开设备节点，不关闭
    NmeaReader  reader[1];
    nmea_reader_init( reader );//初始化
    ret = read( state->fd, buff, sizeof(buff) );   //读取数据
    nmea_reader_addc( reader, buff[nn] );//数据的解析
}
```

nmea_reader_init

```
static void nmea_reader_init( NmeaReader*  r )
{
    memset( r, 0, sizeof(*r) );

    r->pos      = 0;
    r->overflow = 0;
    r->utc_year = -1;
    r->utc_mon  = -1;
    r->utc_day  = -1;
    r->callback = NULL;
    r->fix.size = sizeof(r->fix);
    //初始化时差
    nmea_reader_update_utc_diff( r );
}
```

```
typedef struct {
        int     pos;
        int     overflow;
        int     utc_year;
        int     utc_mon;
        int     utc_day;
        int     utc_diff;
        GpsLocation  fix;//此结构体里面存放经纬度信息
        GpsSvStatus    sv;
        GpsStatus  status;
        gps_sv_status_callback sv_callback;
        gps_status_callback status_cb;
        gps_location_callback  callback;
        char    in[ NMEA_MAX_SIZE+1 ];
} NmeaReader;
```

nmea_reader_addc

```
nmea_reader_addc( r );
Token  tok_time          = nmea_tokenizer_get(tzer,1);
Token  tok_latitude      = nmea_tokenizer_get(tzer,2);
Token  tok_latitudeHemi  =
Token  tok_longitude     = nmea_tokenizer_get(tzer,4);
Token  tok_longitudeHemi = nmea_tokenizer_get(tzer,5);
Token  tok_hdop = nmea_tokenizer_get(tzer,8);
Token  tok_altitude      = nmea_tokenizer_get(tzer,9);
Token  tok_altitudeUnits = nmea_tokenizer_get(tzer,10);
r->callback( &r->fix );//解析完成之后，回调调上层JNI 的location_callback 函数，参数就是经纬度数据
```

回调

```
static void location_callback(GpsLocation* location)
{
    JNIEnv* env = AndroidRuntime::getJNIEnv();
    env->CallVoidMethod(mCallbacksObj, method_reportLocation, location->flags,
            (jdouble)location->latitude, (jdouble)location->longitude,
            (jdouble)location->altitude,
            (jfloat)location->speed, (jfloat)location->bearing,
            (jfloat)location->accuracy, (jlong)location->timestamp);
    checkAndClearExceptionFromCallback(env, __FUNCTION__);
}
//通过一开始获取到用户层的方法ID，在此回调该函数，并将位置信息作为参数
```

create_thread_callback

```
create_thread_callback:
→ createJavaThread(name, start, arg);
  → javaCreateThreadEtc(android_thread_func_t) start, arg, name,ANDROID_PRIORITY_DEFAULT, 0, &threadId);
    → result = androidCreateRawThreadEtc(AndroidRuntime::javaThreadShell, args,threadName,
threadPriority, threadStackSize, threadId);
                //参数是gps_read_thread线程体
    → int result = pthread_create(&thread, &attr,(android_pthread_entry)entryFunction, userData);
细心的人可能会有疑问，为什么要在JNI层创建线程，而不在HAL层，因为我们要读开始数据了
```

```
//android_location_GpsLocationProvider_start:
            sGpsInterface->start()
```

Android系统 GPS框架分析 (JNI-HAL)

Disigned By Chaiyun

Android系统 GPS框架分析 (JNI-HAL)

Disigned By Chaiyun