MPU6500

1. 设定电源管理模式——Power Management 1

static int sensor\_active(struct i2c\_client \*client, int enable, int rate)

1. 初始设置——static int sensor\_init(struct i2c\_client \*client)

1）检测芯片号

read\_data = sensor\_read\_reg(client,sensor->ops->id\_reg);

printk("mpu6500\_acc,%s,read\_data=0x%x.\n",\_\_func\_\_,read\_data);

if(read\_data != sensor->ops->id\_data)

{

printk("%s:check id err,read\_data:%d,ops->id\_data:%d\n",\_\_func\_\_,read\_data,sensor->ops->id\_data);

return -1;

}

2）设置电源MPU6500\_PWR\_MGMT\_1，MPU6500\_PWR\_MGMT\_2（待机）

res = sensor\_write\_reg(client, MPU6500\_PWR\_MGMT\_1,0x80);//芯片重置

res = sensor\_write\_reg(client, MPU6500\_PWR\_MGMT\_2,0x3F); //set accl and gyro all axis into standby mode待机

res = sensor\_write\_reg(client, MPU6500\_PWR\_MGMT\_1,0x41); //0x41，芯片待机，时钟源自动选择

3）设置陀螺仪和加速度计量程MPU6500\_GYRO\_CONFIG，MPU6500\_ACCEL\_CONFIG，MPU6500\_ACCEL\_CONFIG2

res = sensor\_write\_reg(client, MPU6500\_GYRO\_CONFIG,0x18); //config gyro for 2000dps

res = sensor\_write\_reg(client, MPU6500\_ACCEL\_CONFIG,0x00); //config Accel for +\_2G

1. 上报数据——陀螺仪、加速度计上报

static int gsensor\_report\_value(struct i2c\_client \*client, struct sensor\_axis \*axis)//上报加速度计数据

static int gyro\_report\_value(struct i2c\_client \*client, struct sensor\_axis \*axis)//上报陀螺仪数据

static int sensor\_report\_value(struct i2c\_client \*client)//原始数据

1. 芯片注册与注销

static int \_\_init gsensor\_mpu6500\_init(void)

static void \_\_exit gsensor\_mpu6500\_exit(void)

static int \_\_init gyro\_mpu6500\_init(void)

static void \_\_exit gyro\_mpu6500\_exit(void)

Linux Input系统

将输入设备抽象成文件形式，以打开、读写方式对设备进行操作

工作机制

输入设备工作机制： 输入动作——产生中断——CPU通过总线或者IO读取数据到缓冲区

构架层次

app

//--------------------

input\_event\_driver

//--------------------

input\_core

//--------------------

input\_device\_driver

//--------------------

Hardware

数据结构

定义在/linux/input.h下重要的结构体

input\_dev（输入设备对象，指出相关事件）、input\_handler（描述一类事件驱动处理）、input\_handle（将设备与处理器连接）input\_event（表示事件本身）

数据传递过程

首先在驱动层中，调用inport\_report\_abs，然后他调用了input core层的input\_event，input\_event调用了input\_handle\_event对事件进行分派，调用input\_pass\_event，在这里他会把事件传递给具体的handler层，然后在相应handler的event处理函数中，封装一个event，然后给evdev\_list结构体中的buffer成员赋上从设备中传过来的数据，等待用户空间来读取。

当用户空间打开设备节点/dev/input/event0~/dev/input/event4的时候，会使用input\_fops中的input\_open\_file()函数，input\_open\_file()->evdev\_open()(如果handler是evdev的话)->evdev\_open\_device()->input\_open\_device()->dev->open()。也就是struct file\_operations input\_fops提供了通用接口，最终会调用具体input\_dev的open函数