#### 继电器操作接口

虽然can协议中，can帧的id部分是用来进行优先级仲裁的，但是pwrctrl代码中把地址也写进了can帧的id域，只用id域的前几位来进行仲裁，所以挂在can总线上的模块，pdu控制板以及A7板应该都会对接收到的can帧的id部分做解析。

id部分的其他位用来表示帧标识（A7板发送给PDU的控制帧，PDU控制板返回的PDU状态帧等等），PDU地址（比如A7发送的PDU控制帧，地址嵌在0x□\*\*，PDU控制板返回的状态PDU地址嵌在0x\*\*□）；

can帧的id部分：仲裁 + 帧标识符 + 地址（如果有的话）；具体的还有很多细节，比如控制功率模块的can帧的id部分各个位都用得很细致，当然也包括模块地址在里面，**总之**：模块，PDU控制板，A7板是通过can通信的（创建can接收线程，传入接收处理回调函数），具体的帧组织方式不用看，只看调用接口即可；

###### 底层

typedef struct

{

unsigned char id; // 继电器编号//lhm: 应该是PDU编号

unsigned char sw[SINGLE\_PDU\_RELAY\_NUM]; // 继电器反馈状态

unsigned int swTimes[SINGLE\_PDU\_RELAY\_NUM]; // 继电器操作次数

int temp[2]; // 温度

unsigned char ctrlBack; // 反馈状态

//RESAY\_BACK\_OK

//RELAY\_BACK\_ERR

//RELAY\_BACK\_WAIT

unsigned char online; // 0-掉线，1-在线

}RELAY\_INFO\_STRUCT;（relayCtrl.h文件定义的结构）

RELAY\_FUNCTIONS\_STRUCT Relay\_SCII\_Functions =

{

.Init = Init, //lhm: 创建RelayDeal线程 **+** 一些初始化（给can接口传入本c文件定义

//的接收回调函数并创建底层can接收线程）；

.Uninit = Uninit, //lhm: 销毁can接收线程以及RelayDeal线程；

.SetSwitch = SetSwitch, //lhm: **给编号relayNo的PDU发送继电器控制（6字节数组pSW，每个字**

//**节对应PDU的一个继电器的闭合/断开控制）**，具体的发送由RelayDeal

//线程处理，调取GetInfo获得结果；

// static int SetSwitch(unsigned char \*pSW, int relayNo)

.GetInfo = GetInfo, //lhm: 获取编号relayNo的PDU信息g\_relayInfo[relayNo - 1]

//static int GetInfo(RELAY\_INFO\_STRUCT \*pInfo, int relayNo)

};

**做法：**接口函数负责修改 / 读取中间变量（中间变量的数据类型放在h文件中对外开放），而线程不断地将中间变量的内容发送出去以及不断接收数据并赋值给中间变量（线程一般在接口函数Init中创建，在Uinit函数中销毁）：

接口函数------SetSwitch和GetInfo

线程------------RelayDeal和can接收线程

（如果增加了一个PDU怎么办，直接在SetSwitch的PDU编号参数加1即可，因为用的是热插拔的can协议，定好了协议，PDU直接挂在can总线上即可，通信协议包括心跳在线检测，“地址”分配等等这些都定好了）

###### 上层

typedef struct

{

unsigned char pduNum; // pdu数量

unsigned char pduRlyNum; //单个pdu的继电器数量

unsigned char pduTempLevel; // pdu温度阈值

unsigned char pduType; // pdu类型

}RELAY\_PARA\_STRUCT;

int RelayCtrl\_GetCtrlBack(int relayNo);

int RelayCtrl\_SetSwitch(unsigned char \*pSW, int relayNo);

int RelayCtrl\_GetInfo(RELAY\_INFO\_STRUCT \*pInfo, int relayNo);

void RelayCtrl\_SetPara(RELAY\_PARA\_STRUCT para);//lhm: 除了一些参数设置还包括

//g\_pRelayFun = Get\_RelaySCII\_Functions()并调用//g\_pRelayFun->Init()

void RelayCtrl\_Init();

void RelayCtrl\_Uninit();//lhm: g\_pRelayFun->Uninit()

#### 策略算法matrix接口（matrix.c和matrix.h）

基本做法是将继电器断开闭合情况，电压电流需求情况等等整合到各种结构数据类型当中，调用者将策略分配的各个参数赋值给这些数据类型的变量，然后作为参数传入给策略分配算法接口，而接口根据这些参数计算出分配策略，再以变量的形式传出，接口不负责具体的操作，而是通过变量描述这些操作，调用者得到传出变量后再调用其他接口部署这些操作：比如1号PDU的第3个继电器要断开，则传出参数是PDU[0][2] = 0，然后调用者根据传出参数再调用继电器接口让具体的继电器断开（策略部署）。

美标应该只涉及到枪功率分切入切出策略的修改（通过调用模块接口获得当前模块的状态，通过通信接口获得枪的需求，然后进行功率分配）

.h文件提供接口函数（或者接口函数集合即成员变量全是函数指针的结构体，这个接口函数集合是数据类型而不是全局变量，应该是再提供一个函数来返回这个接口函数集合类型的变量 ）

.h文件提供数据类型（不算接口函数集合），调用者使用这些数据类型变量（调用者给其赋值）作为接口函数的输入参数，然后调用者使用被修改后的输入参数或者返回值

接口函数可能会创建一个线程，这个线程会处理.h文件提供的数据类型的变量（线程也会用到.c文件的私有数据类型），然后接口函数需要操作仅仅是读取.c文件的私有变量（但是这个变数据类型量是公有的，即数据类型定义在.h文件当中）或者私有数据类型的私用变量，并将需要的部分放到输入参数当中

尽量杜绝全局变量

按照这个思路去看C代码，就不会有疑惑“这个结构的成员变量的值从哪里来的？”，调用该接口的调用者置值的。

###### 输入

初始化：

枪的充电需求（1、正在充电的枪发生功率改变；2、空置的枪位发起充电请求）：

###### 输出

所有枪的继电器通断情况（CHANGE\_NO、CHANGE\_IN和CHANGE\_OUT）