

通用协议 SDK 使用说明

一、概述

本文档对充电桩使用通用协议进行联网中关于网络连接、数据收发、设备登录等流程进行可移植性的开发。

1.1 说明

本协议用于界定充电桩与充电运营管控系统服务器之间的通信协议规范。

1.2 接口定义

充电桩与充电运营管控系统服务器之间的通信接口采用基于 TCP/IP Socket 的通信方式实现，按照长连接工作模式。





1.3 通信方式

本接口采用 Server/Client 的通信方式，充电桩作为客户端 Client 方，充电运营管控系统作为服务器 Server 方。

1.4 通信规约

- 1) 通信数据报文采用 16 进制顺序表示，多字节段以低位在前，高位在后传输。
- 2) 由客户端主动向服务端发起连接请求，连接成功后开始数据报文通信。
- 3) 客户端的状态数据具有心跳功能，按周期上报，默认周期为 10s，可以设置。
- 4) 采用 CRC16 对接收的数据包进行合法性校验, CRC 校验如附件所述。
- 5) 客户端自动维护通信连接状态的有效性，在初始化和断链以后（断链以状态数据的回复作为判定依据，服务端 5 次未回复状态数据时，视为断链），自动进行连接尝试，直到连接恢复。

二、目录说明

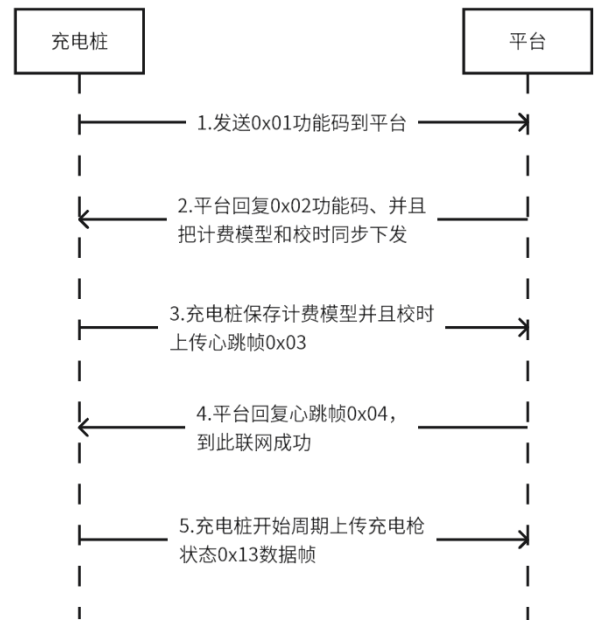
 Net_Deal.c	2023/10/17 23:12	C 文件	68 KB
 Net_Deal.h	2022/10/27 14:56	H 文件	2 KB
 DataDeal.h	2022/10/27 14:56	H 文件	2 KB
 DataDeal.c	2022/10/27 14:56	C 文件	15 KB

其中 Net_Deal.c 文件是设备联网的程序文件，DataDeal.c 是在联网时会用到的一些数据处

理函数。

三、设备登录

流程如下图所示



说明：在网络初始化成功后，开始调用 `void NetDealTask(void *pvParameters)` 函数创建 TCP/IP 连接。创建成功后就可以开始登录流程了。 `NetDeal_SendRecv(GUN_A);` 这个函数是所有网络功能的数据交互函数。在这个函数中会完成以下功能

- 1、创建一个 TCP/IP 连接，创建成功后开始数据交互
- 2、数据发送
- 3、数据接收解析

```

static void NetDeal_SendRecv(u8 GunNum)
{
    err_t err = ERR_OK;

    if (NetFlag.SocketCreateOk == FALSE)//socket未连接
    {
        SoftFeedWdg_Unactive(SOFTWDG_TASKID_NetDeal);//关闭看门狗监测
        err = CreateConnectForGun(GunNum); //创建连接
        if (err == ERR_OK)
        {
            NetFlag.NetFailTimes = 0;
            SoftFeedWdg_Active(SOFTWDG_TASKID_NetDeal);//打开看门狗监测
            NetFlag.SocketCreateOk = TRUE;//socket连接成功
            sy_adp_trace(SYADP_TRACE_INFO,"Gun[%d] Create Connect OK...\r\n", GunNum);
        }
    }
    else
    {
        JDataSend(GunNum); //数据发送处理
        //同步时间(每天一次)
        //Syn_Paramater_EveryDay(GunNum);
        //接收数据
        NetDeal_Recv(GunNum); //数据接收解析
        //如接受数据时发现连接已断开, 则删除连接, 清除状态
        if (NetFlag.SocketCreateOk == FALSE)
        {
            DeleteConnectForGun(); //连接状态判断, 如断联就开始重连准备
        }
    }
}
« end NetDeal_SendRecv »

```

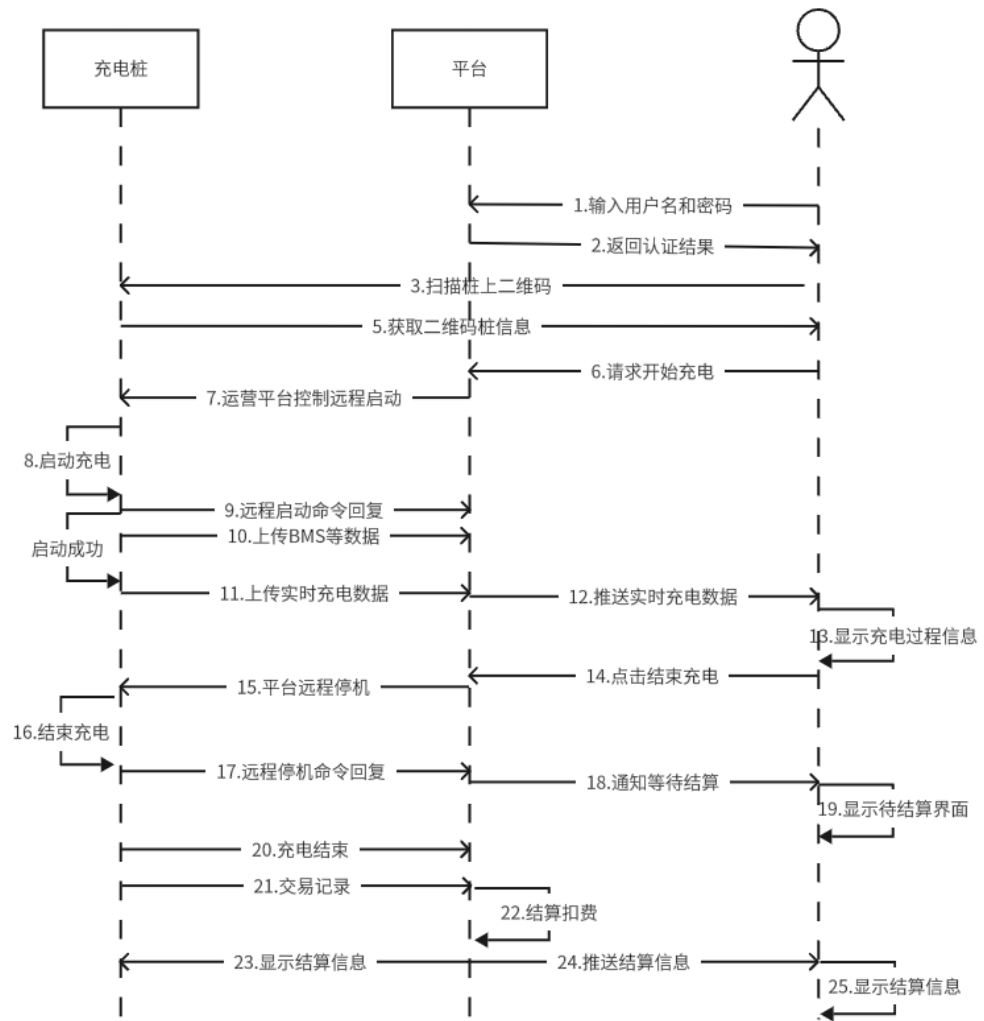
数据发送函数：在 Net_init_send_data 函数中进行设备登录、校时、计费模型获取等，完成后标识充电中登录成功，NetFlag.NetSuccessFlag 该标志位会置位，标识联网成功，可以开始正常数据交互。Net_StateHardData_Send 函数是心跳包周期发送以及充电枪状态变位上传或者空闲定期上传。

```

static void JDataSend(u8 GunNum)
{
    if(NetFlag.NetSuccessFlag == FALSE)//未联网开始网络初始化进行对时等工作
    {
        Net_init_send_data(GunNum);
    }
    else //网络连接成功了开始发送业务数据
    {
        Net_BusinessData_Send(GunNum);
        Net_StateHardData_Send();
    }
    //数据重发
    CheckTimeOut(GunNum);
}

```

四、远程充电



1、识别到插枪后调用函数发送枪的变位状态

```

void PileSendRealTimeInfo(INT8U GunNum)
{
    u8 buf[90]={0},len=0;
    u16 CRCdata=0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    // 包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(OFFLINEDATA, &buf[len]);
    //数据
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].OrderNumber,16);//交易流水号

    len += AddNewCommandBuf(&buf[len],NetFlag.NewPileID,7);//ID:不足7字节 位补 0

    len += AddNewCommandByte((GunNum+1), &buf[len]);//枪编号:A枪为1、B枪为2

    len += AddNewCommandByte(PileChargeInfo[GunNum].PileGunState, &buf[len]);//枪状态

    len += AddNewCommandByte(PileChargeInfo[GunNum].GunReturnState, &buf[len]);//枪归位状态

    len += AddNewCommandByte(PileChargeInfo[GunNum].IfGunInsert, &buf[len]);

    len += AddNewCommandInt16_LOW(PileChargeInfo[GunNum].OutVoltage, &buf[len]);

    len += AddNewCommandInt16_LOW(PileChargeInfo[GunNum].OutCurrent, &buf[len]);

    len += AddNewCommandByte(PileChargeInfo[GunNum].GunTemp, &buf[len]);

    //memcpy(&buf[len],0x00,8);//枪线编码直接赋值0x00
    len += 8;
}

```

2、收到远程命令后开启充电，并且在充电握手和参数配置配置阶段分别发送数据帧 0x15 和 0x17 到平台

```

void PileSendChargeStep1Info(INT8U GunNum)
{
    u8 buf[90]={0},len=0;
    u16 CRCdata=0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    // 包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(CHARGESTART, &buf[len]);
    //数据
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].OrderNumber,16);//交易流水号

    len += AddNewCommandBuf(&buf[len],NetFlag.NewPileID,7);//ID:不足7字节 位补 0

    len += AddNewCommandByte((GunNum+1), &buf[len]);//枪编号:A枪为1、B枪为2

    len += AddNewCommandBuf(&buf[len],&BatteryInfo[GunNum].ProtocolVer[0],3);

    len += AddNewCommandByte(BatteryInfo[GunNum].BatteryType, &buf[len]);

    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].CarPowerCapacity, &buf[len]);

    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].CarVoltageCapacity, &buf[len]);

    len += AddNewCommandInt32_LOW(BatteryInfo[GunNum].BatteryManufacturer, &buf[len]);

    len += AddNewCommandInt32_LOW(BatteryInfo[GunNum].BatteryNumber, &buf[len]);

    len += AddNewCommandByte(BatteryInfo[GunNum].BatteryCreateYear, &buf[len]);
}

```

```

void PileSendChargeStep2Info(INT8U GunNum)
{
    u8 buf[90]={0},len=0;
    u16 CRCdata=0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    //包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(CHARGEPARASET, &buf[len]);
    //数据
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].OrderNumber,16);//交易流水号

    len += AddNewCommandBuf(&buf[len],NetFlag.NewPileID,7);//ID:不足7字节 位补 0

    len += AddNewCommandByte((GunNum+1), &buf[len]);//枪编号:A枪为1、B枪为2

    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].SingleBatteryMaxVol, &buf[len]);
    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].MaxChargeCurrent, &buf[len]);
    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].BatteryAllPower, &buf[len]);
}

```

3、在正式开始充电的时候发送 0x23 和 0x25 到平台

```

void PileSendCharge1Info(INT8U GunNum)
{
    u8 buf[90]={0},len=0;
    u16 CRCdata=0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    //包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(CHARGE BMSREQ, &buf[len]);
}

void PileSendCharge2Info(INT8U GunNum)
{
    u8 buf[50]={0},len=0;
    u16 CRCdata=0;

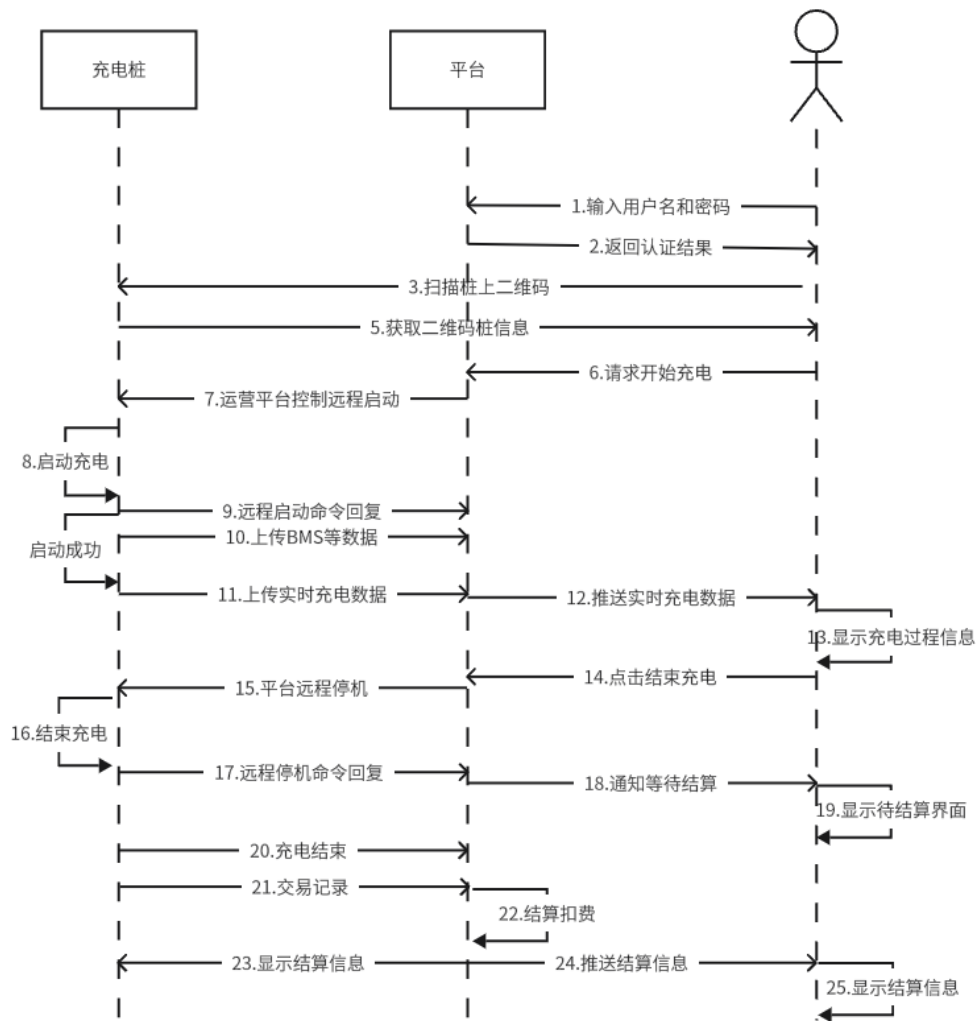
    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    //包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
}

```

4、正常按周期发送 0x13 数据帧

五、停止充电流程

平台停止充电有以下几种类型：APP 点击停止、平台设置限充（SOC 达到设定值），流程是一致的。



1、充电桩收到停止充电命令 0x35 和，先回复 0x36 数据帧，并且开始停止充电流程。包括向 BMS 发送 CST 报文开始停止能量输出，同时调用函数 `PileSendChargeStep3Info` 发送停止过程中的 BMS 数据到平台。

```

void PileSendChargeStep3Info(INT8U GunNum)
{
    u8 buf[60]={0},len=0;
    u16 CRCdata=0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    //包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(CHARGESTOP, &buf[len]);
    //数据
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].OrderNumber,16);//交易流水号

    len += AddNewCommandBuf(&buf[len],NetFlag.NewPileID,7);//ID:不足7字节 位补 0

    len += AddNewCommandByte((GunNum+1), &buf[len]); //枪编号:A枪为1、B枪为2

    len += AddNewCommandByte((BatteryInfo[GunNum].BatterySOC/10), &buf[len]);

    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].SingleBatteryMinVol, &buf[len]);
    len += AddNewCommandInt16_LOW(BatteryInfo[GunNum].SingleBatteryMaxVol, &buf[len]);

    len += AddNewCommandByte(BatteryInfo[GunNum].BatteryMinTemp, &buf[len]);
    len += AddNewCommandByte(BatteryInfo[GunNum].BatteryMaxTemp, &buf[len]);

    len += AddNewCommandInt16_LOW(PileChargeInfo[GunNum].GunChargeTime, &buf[len]);
    len += AddNewCommandInt16_LOW(PileChargeInfo[GunNum].ChargeMeterData/1000, &buf[len]);
}
  
```

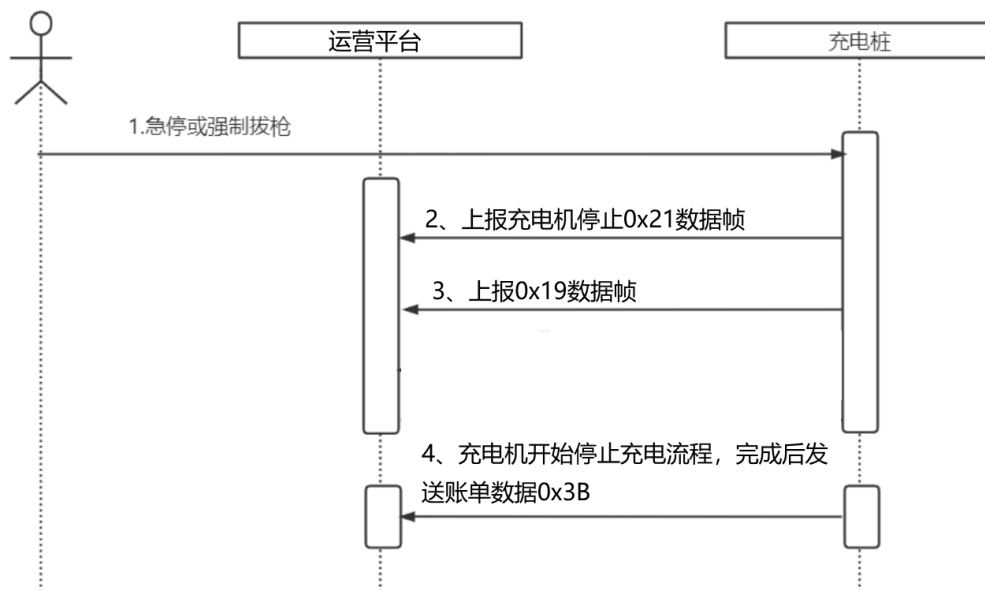
2、停止完成后把订单数据保存到本地，调用 0x3B 数据帧将充电账单发送到平台，在收到回复后停止上传订单，失败的时候继续上传直到收到平台的回复

```
void FileSendRecordData(INT8U GunNum)
{
    u8 buf[200]={0},len=0;
    u16 CRCdata=0;
    u8 i = 0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    //包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(CHARGEILL_S, &buf[len]);

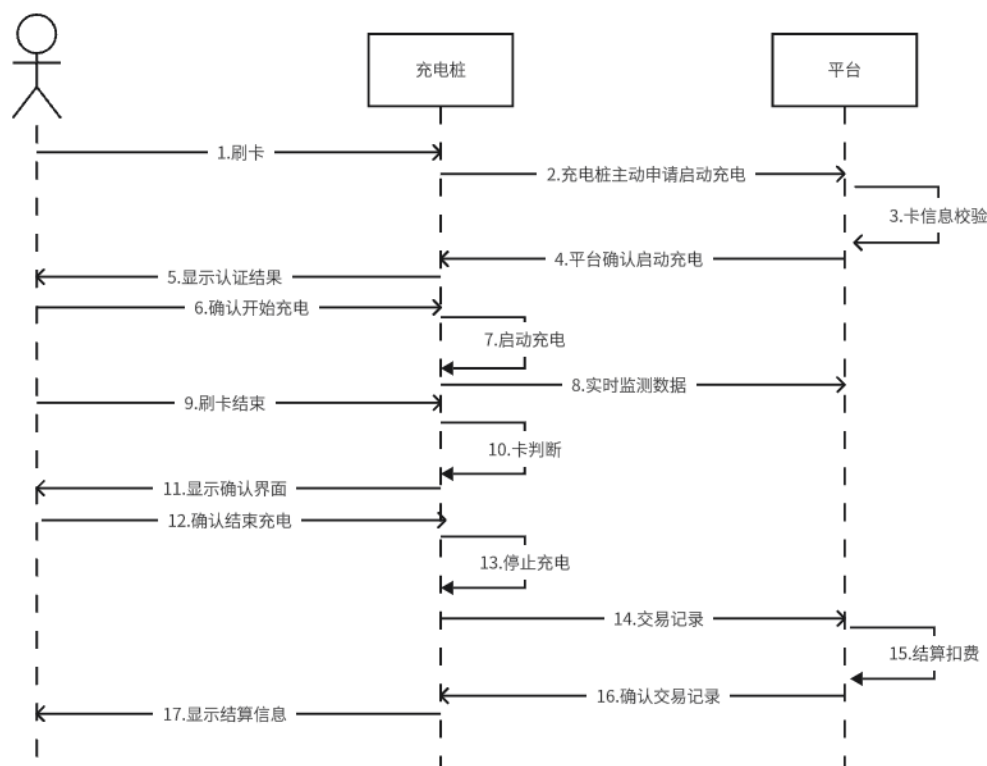
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].OrderNumber,16);//交易流水号
    len += AddNewCommandBuf(&buf[len],NetFlag.NewPileID,7);//ID:不足7字节 位补 0
    len += AddNewCommandByte((GunNum+1), &buf[len]);//枪编号:A枪为1、B枪为2
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].StartTime,7);
    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].StopTime,7);
    //充电金额信息
    for(i=0;i<4;i++)
    {
        len += AddNewCommandInt32_LOW(PileChargeInfo[GunNum].ChargeMoneyInfo.ChargeRateInfo.ElecRate[i], &buf[len]);
        len += AddNewCommandInt32_LOW(PileChargeInfo[GunNum].ChargeMoneyInfo.ChargeMeterData[i], &buf[len]);
        len += AddNewCommandInt32_LOW(PileChargeInfo[GunNum].ChargeMoneyInfo.ChargeLossData[i], &buf[len]);
        len += AddNewCommandInt32_LOW(PileChargeInfo[GunNum].ChargeMoneyInfo.ChargeMoney[i], &buf[len]);
    }
}
```

六、桩异常停止充电



该过程具体报文请参考远程停止充电流程的调用函数

七、刷卡充电



1、刷卡后立即上传卡信息到平台，数据帧 0x31，调用函数 PileSendStartCharge。

```

void PileSendStartCharge(INT8U GunNum)
{
    u8 buf[90]={0},len=0;
    u16 CRCdata=0;

    //包头
    len += AddNewCommandByte(HeadCmd, &buf[len]);
    // 包长
    len += 1;
    //序列号域
    len += AddNewCommandInt16_LOW(0x0000, &buf[len]);
    //加密标志
    len += AddNewCommandByte(CODETYPE_1, &buf[len]);
    //数据帧类型
    len += AddNewCommandByte(PILENEEDSTARTCHARGE_S, &buf[len]);
    //数据
    len += AddNewCommandBuf(&buf[len],NetFlag.NewPileID,7);//ID:不足7字节 位补 0

    len += AddNewCommandByte((GunNum+1), &buf[len]); //枪编号:A枪为1、B枪为2

    len += AddNewCommandByte(PileChargeInfo[GunNum].StartChargeMode, &buf[len]); //是否需要密码 0x00:不需要 0x01:需要

    len += AddNewCommandByte(0x00, &buf[len]); //是否需要密码 0x00:不需要 0x01:需要

    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].CardID,8);

    len += AddNewCommandBuf(&buf[len],PileChargeInfo[GunNum].CardCode,16);

    if(PileChargeInfo[GunNum].StartChargeMode == 3)
        len += AddNewCommandBuf(&buf[len],&BatteryInfo[GunNum].CarVINCode[0],17);
    else
        len += 17;
}
  
```

2、收到平台确认函数后，会得到用户的信息以及充电流水号，并且开始充电流程。然后实时发送充电过程中 BMS 的数据到平台

3、刷卡结束充电或者 BMS 主动要求停止充电，发送交易记录 0x3B 到平台等待平台回复 0x40。

八、网络其他重要函数

7.1、判断超时

```
static void NetTimeOutInit(void)
{
    int i = 0, j = 0;
    for (i = 0; i < CHARGEPILE_GUN_NUM; i++)
    {
        for (j = 0; j < NETTIMEOUTMAX; j++)
        {
            memset(&(s_TimeOut[i][j]), 0, sizeof(Time_Out));
        }
    }
}

void NetTimeOutSet(u8 GunNum, u8 item, u8* buf, int len, int repeat)
{
    if (item < 0 || item >= NETTIMEOUTMAX)
        return;
    if (buf == NULL || len == 0 || len >= TIMEOUT_BUFFER)
        return;

    memcpy(s_TimeOut[GunNum][item].ptr, buf, len);
    s_TimeOut[GunNum][item].Len = len;
    s_TimeOut[GunNum][item].TimeData = xTaskGetTickCount();
    s_TimeOut[GunNum][item].RepeatCycle = repeat;
}

void NetTimeOutClear(u8 GunNum, int item)
{
    if (item < 0 || item >= NETTIMEOUTMAX)
        return;

    memset(&(s_TimeOut[GunNum][item]), 0, sizeof(Time_Out));
}
```

其中 NetTimeOutInit 为初始化函数，在初始化中调用，将需要判断重发的重要数据通道初始化。NetTimeOutSet 函数是在重要报文发送的时候调用，设置通道值、重发次数值、延时判断时间、重发数据等形参。NetTimeOutClear 函数是成功收到回复就清除 NetTimeOutSet 时设置的参数，此时就不再判断超时了。

```
static void CheckTimeOut(u8 GunNum)
{
    u8 i=0;
    u32 tmr=0,nowtime=0;
    Time_Out *temp = &(s_TimeOut[GunNum][0]);
    if(NetFlag.NetSuccessFlag != TRUE)
    { //网络初始化未完成
        return;
    }
    //轮询检查有没有超时
    for(i=0; i<NETTIMEOUTMAX; i++)
    {
        if(temp[i].Len != 0)
        {
            WORKCACL(tmr,temp[i].TimeData,nowtime);
            if(tmr > temp[i].RepeatCycle) // 超时时间到
            {
                //连续五次没有成功退出
                if(temp[i].Count > TIMEOUT_RETRY_MAX) // 发送五次 就认为超时
                {
                    NetTimeOutClear(GunNum, i);
                }
                else //发送记录
                {
                    NetSendData(GunNum, temp[i].ptr, temp[i].Len);
                    temp[i].TimeData = xTaskGetTickCount();
                    temp[i].Count++;
                }
            }
        }
    }
} // end for i=0;i<NETTIMEOUTMAX;i... »
if(NetFlag.NetFailTimes > 5)
{
    NetFlag.NetSuccessFlag = FALSE;
    NetFlag.SocketCreateOk = FALSE;
    NetFlag.Init_step = NET_LOGIN_STEP;
    NetFlag.RequestedFlag = FALSE;
    DeleteConnectForGun();
    NetFlag.NetFailTimes = 0;
    sy_adp_trace(SYADP_TRACE_INFO,"Net Time Out...\r\n");
}
```

该函数是判断重发的主体函数，按照通道值进行每个通道的判断，如果延时时间到了但是还是没有收到回复就继续发送，如果连续 5 次没有收到回复那就要清除该通道数据，认为网络断开了并进行网络重连。

附录 1：CRC 校验

```
unsigned char gabyCRCHI[] =  
{ 0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x01,0xc0,  
  0x80,0x41,0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,  
  0x00,0xc1,0x81,0x40,0x00,0xc1,0x81,0x40,0x01,0xc0,  
  
  0x80,0x41,0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,  
  0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x00,0xc1,  
  0x81,0x40,0x01,0xc0,0x80,0x41,0x01,0xc0,0x80,0x41,  
  0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x00,0xc1,  
  0x81,0x40,0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,  
  0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x01,0xc0,  
  0x80,0x41,0x00,0xc1,0x81,0x40,0x00,0xc1,0x81,0x40,  
  0x01,0xc0,0x80,0x41,0x01,0xc0,0x80,0x41,0x00,0xc1,  
  0x81,0x40,0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,  
  0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x01,0xc0,  
  0x80,0x41,0x00,0xc1,0x81,0x40,0x00,0xc1,0x81,0x40,  
  0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,0x01,0xc0,  
  0x80,0x41,0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,  
  0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x01,0xc0,  
  0x80,0x41,0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
```

```
0x00,0xc1,0x81,0x40,0x00,0xc1,0x81,0x40,0x01,0xc0,  
0x80,0x41,0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,  
0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,0x01,0xc0,  
0x80,0x41,0x00,0xc1,0x81,0x40,0x00,0xc1,0x81,0x40,  
0x01,0xc0,0x80,0x41,0x01,0xc0,0x80,0x41,0x00,0xc1,  
0x81,0x40,0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,  
0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,0x01,0xc0,  
0x80,0x41,0x00,0xc1,0x81,0x40
```

```
};
```

```
unsigned char gabyCRCLo[] =  
{ 0x00,0xc0,0xc1,0x01,0xc3,0x03,0x02,0xc2,0xc6,0x06,  
0x07,0xc7,0x05,0xc5,0xc4,0x04,0xcc,0x0c,0x0d,0xcd,  
0x0f,0xcf,0xce,0x0e,0x0a,0xca,0xcb,0x0b,0xc9,0x09,  
0x08,0xc8,0xd8,0x18,0x19,0xd9,0x1b,0xdb,0xda,0x1a,  
0x1e,0xde,0xdf,0x1f,0xdd,0x1d,0x1c,0xdc,0x14,0xd4,  
0xd5,0x15,0xd7,0x17,0x16,0xd6,0xd2,0x12,0x13,0xd3,  
0x11,0xd1,0xd0,0x10,0xf0,0x30,0x31,0xf1,0x33,0xf3,  
0xf2,0x32,0x36,0xf6,0xf7,0x37,0xf5,0x35,0x34,0xf4,  
0x3c,0xfc,0xfd,0x3d,0xff,0x3f,0x3e,0xfe,0xfa,0x3a,  
0x3b,0xfb,0x39,0xf9,0xf8,0x38,0x28,0xe8,0xe9,0x29,  
0xeb,0x2b,0x2a,0xea,0xee,0x2e,0x2f,0xef,0x2d,0xed,  
0xec,0x2c,0xe4,0x24,0x25,0xe5,0x27,0xe7,0xe6,0x26,  
0x22,0xe2,0xe3,0x23,0xe1,0x21,0x20,0xe0,0xa0,0x60,  
0x61,0xa1,0x63,0xa3,0xa2,0x62,0x66,0xa6,0xa7,0x67,  
0xa5,0x65,0x64,0xa4,0x6c,0xac,0xad,0x6d,0xaf,0x6f,  
0x6e,0xae,0xaa,0x6a,0x6b,0xab,0x69,0xa9,0xa8,0x68,  
0x78,0xb8,0xb9,0x79,0xbb,0x7b,0x7a,0xba,0xbe,0x7e,  
0x7f,0xbf,0x7d,0xbd,0xbc,0x7c,0xb4,0x74,0x75,0xb5,  
0x77,0xb7,0xb6,0x76,0x72,0xb2,0xb3,0x73,0xb1,0x71,  
0x70,0xb0,0x50,0x90,0x91,0x51,0x93,0x53,0x52,0x92,  
0x96,0x56,0x57,0x97,0x55,0x95,0x94,0x54,0x9c,0x5c,  
0x5d,0x9d,0x5f,0x9f,0x9e,0x5e,0x5a,0x9a,0x9b,0x5b,  
0x99,0x59,0x58,0x98,0x88,0x48,0x49,0x89,0x4b,0x8b,  
0x8a,0x4a,0x4e,0x8e,0x8f,0x4f,0x8d,0x4d,0x4c,0x8c,  
0x44,0x84,0x85,0x45,0x87,0x47,0x46,0x86,0x82,0x42,  
0x43,0x83,0x41,0x81,0x80,0x40  
};
```

```
/******
```

```
***
```

```
* 描 述 : 通用协议使用的 CRC 校验计算
```

* 输 入 : pBuffer 校验数据
uLen 校验数据长度

* 输 出 : crc 值

* 返 回 :

***/
INT16U ModbusCRC(INT8U * pData, INT16U len)

```
{
    INT8U byCRCHi = 0xff;
    INT8U byCRCLo = 0xff;
    INT8U byIdx;
    INT16U crc;
    while(len--)
    {
        byIdx = byCRCHi ^* pData++;
        byCRCHi = byCRCLo ^ gabyCRCHi[byIdx];
        byCRCLo = gabyCRCLo[byIdx];
    }
    crc = byCRCHi;
    crc <<= 8;
    crc += byCRCLo;
    return crc;
}
```