通讯协议手册

(Ver 2.3)

深圳市微雪电子有限公司

目 录

一、通讯协议

1.1 通讯处理过程

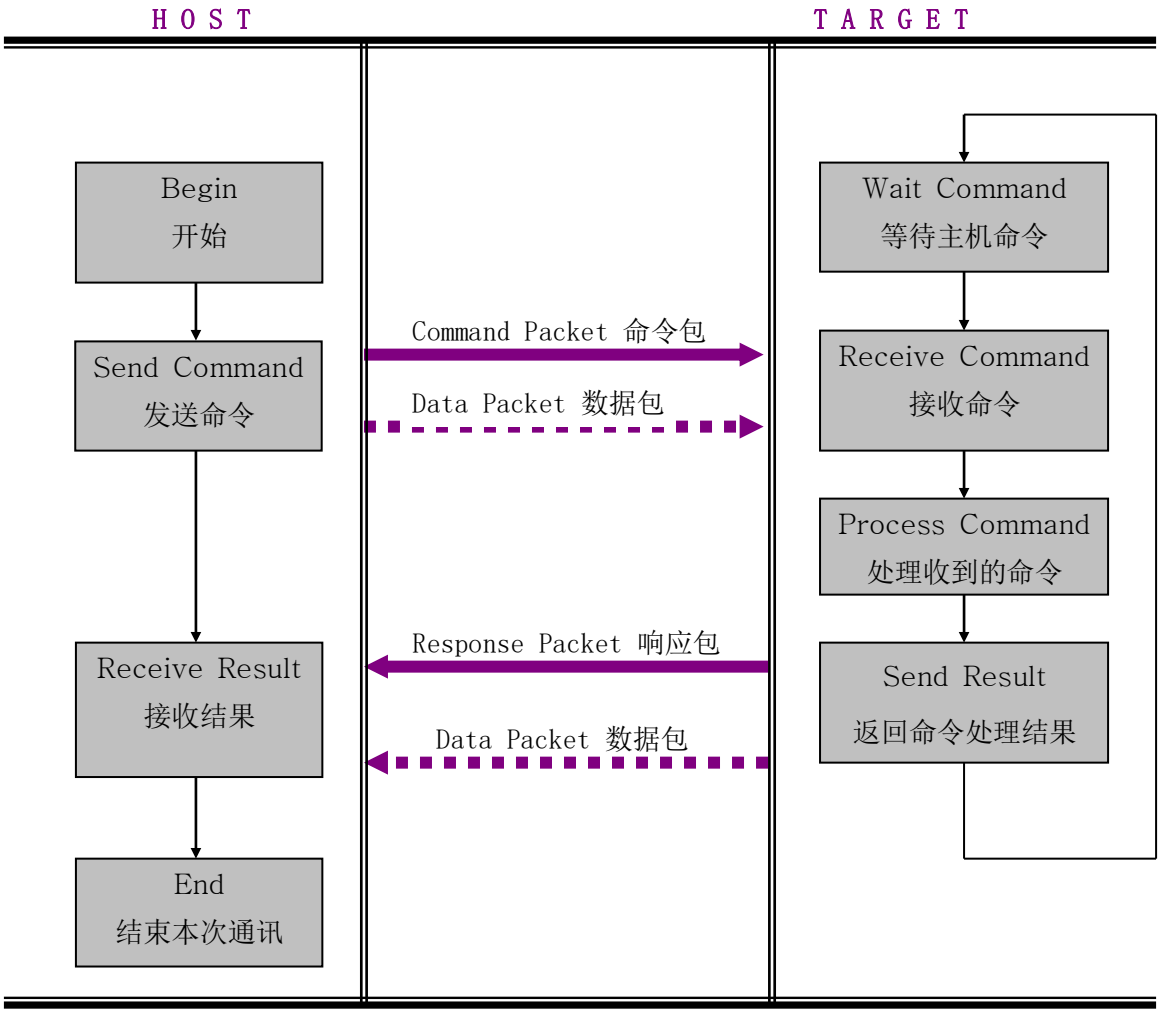


图4-1 通讯过程

注：

所有指令的发送、接收必须要遵循一发一收的原则。Host在没有收到应答时，请不要向 TARGET 发送指令。

1.2 数据结构

1.2.1 包识别码（小端模式）

```
#define XG_PREFIX_CODE                0xAABB //小端模式，先发BB后发AA
```

1.2.2 包结构

```
/*包数据结构24字节*/
typedef struct _XG_PACKET_
{
        unsigned short  wPrefix; //包标识xAABB或xBBAA，大端小端模式不同,2个字节
        unsigned char   bAddress; //设备地址,多个设备接总线时使用，单个设备时为,1个字节
        unsigned char   bCmd; //命令码,1个字节
        unsigned char   bEncode; //数据编码，个字节
        unsigned char   bDataLen; //有效数据长度,1个字节，取值范围-16
        unsigned char   bData[16]; //包数据,16字节
        unsigned short  wCheckSum; //包检验，字节，为-22字节的累加和，小端模式
} XG_PACKET, *PXG_PACKET;
```

例：

连接指令包数据：

发送：BB AA 00 01 00 08 30 30 30 30 30 30 30 30 00 00 00 00 00 00 00 00 EE 02

接收：BB AA 00 01 00 10 00 37 2D 31 B0 E0 00 00 00 00 00 00 00 00 00 00 02 9D 03

1.2.3 包校验

计算方法：

包校验和 wCheckSum为从wPrefix开始到bData数据结束按字节进行和运算，取计算结果的最低2 byte。

代码示例：

```
/*
功能:计算校验和
pBuf:需要计算校验和的缓存首地址
Size:计算校验和的缓存长度
返回值:
2字节的校验和，不同系统需要注意大小端区别
*/
UINT16 GetCheckSum(UINT8* pBuf, UINT16 Size)
```

```
{
	UINT16   w_Ret = 0;
	UINT16 i;
	UINT8* p = (UINT8*)&w_Ret;

	for (i = 0; i < Size; i++ )
		w_Ret += pBuf[i];
	//大端到小端的转换
	//w_Ret = p[0] + (p[1]<<8);
	return w_Ret;
}
```

1.2.4 发指令包

```
/*
功能:通过串口发送指令包
bCmd:指令码
pData:指令包需要发送的数据
Len:有效数据长度
返回值:
XG_ERR_SUCCESS发送成功
*/

UINT8 SendPack(UINT8 bCmd, UINT8* pData, UINT8 Len)
{
	XG_PACKET pack = { 0 };

	pack.wPrefix = XG_PREFIX_CODE;
	pack.bCmd = bCmd;
	pack.bDataLen = Len;
	if(pData && Len > 0)
	{
		UINT8 i;
		for(i = 0; i < Len; i++) pack.bData[i] = pData[i];
	}
	pack.wCheckSum = GetCheckSum((UINT8*)&pack, sizeof(XG_PACKET) - 2);
	UartSendBuf((UINT8*)&pack, sizeof(XG_PACKET));
```

```
        return XG_ERR_SUCCESS;
}
```

1.2.5 收指令包回包

```
/*
功能:从串口接收一个指令包
pCmd:返回指令码的指针
pData:指令包的数据输出缓存,16字节
Timeout:接收超时，单位毫秒
返回值:
XG_ERR_SUCCESS:接收成功
XG_ERR_TIME_OUT:接收超时
XG_ERR_DATA:数据错误
*/

UINT8 RecvPack(UINT8* pCmd, UINT8* pData, int Timeout)
{
        XG_PACKET pack = { 0 };

        UINT8 ret = UartRecvBuf((UINT8*)&pack, (UINT8)sizeof(XG_PACKET), Timeout);
        if(ret >= sizeof(XG_PACKET))
        {
                UINT16 CheckSum = 0;

                if(pack.wPrefix != XG_PREFIX_CODE) return XG_ERR_DATA;
                CheckSum = GetCheckSum((UINT8*)&pack, sizeof(XG_PACKET) - 2);
                if(CheckSum != pack.wCheckSum) return XG_ERR_DATA;
                if(pCmd) *pCmd = pack.bCmd;
                if(pData)
                {
                        UINT8 i;
                        for(i = 0; i < 16; i++) pData[i] = pack.bData[i];
                }
        } else {
                return XG_ERR_TIME_OUT;
```

```
        }
        return XG_ERR_SUCCESS;
}
```

1.2.6 读取数据

```
/*
功能:读取一个数据包
参数:
bCmd: 指令码
bDataBuf: 要读取的数据缓存首地址
iOffset: 要读取的数据地址偏移
iSize: 要读取的数据大小，不能大于一个包的大小
iTimeout: 读取一个包的超时
返回值: > 0读取成功的数据大小，=0读取失败
*/
static UINT16 ReadDataPack(UINT8 bCmd, UINT8* bDataBuf, UINT16 iOffset, UINT16 iSize, int iTimeout)
{
        UINT16 RecvSize = 0;
        //UINT8 bBuf[UART_RECV_BYTE_MAX] = { 0 }; //初始化数组
        UINT8* bBuf = NULL; //如果单片机为节省内存使用可直接用串口缓存
        UINT8 bCmdData[16] = { 0 }; //初始化数组

        if(iSize > COM_DATA_PACKET_SIZE) return 0;

        bCmdData[0] = bCmd;
        bCmdData[1] = (iOffset&0xff);
        bCmdData[2] = ((iOffset>>8)&0xff);
        bCmdData[3] = 0;
        bCmdData[4] = 0;
        bCmdData[5] = (iSize&0xff);
        bCmdData[6] = ((iSize>>8)&0xff);
        bCmdData[7] = 0;
        bCmdData[8] = 0;

        SendPack(XG_CMD_READ_DATA, bCmdData, 9);
        gUartByte = 0; //清除串口接收缓存
```

```
        //接收的数据要加上字节的校验和
        RecvSize = UartRecvBuf(bBuf, iSize+2, iTimeout);
        bBuf = gUartBuf; //如果单片机为节省内存使用可直接用串口缓存
        if(RecvSize >= iSize + 2)
        {
                UINT16 CheckSum2 = (UINT16)(bBuf[iSize] + (bBuf[iSize + 1]*256));
                UINT16 CheckSum1 = GetCheckSum(bBuf, iSize);
                if(CheckSum1 == CheckSum2)
                {
                        int i;
                        for(i = 0; i < iSize; i++)
                        {
                                bDataBuf[i] = bBuf[i];
                        }
                        return iSize;
                }
        }
        return 0;
}

/*
功能:分包读取数据
参数:
bCmd: 指令码
bDataBuf: 要读取的数据缓存首地址
iSize: 要读取的数据大小
iTimeout: 读取一个包的超时
返回值: = 0读取成功，非0读取失败
*/
static UINT8 ReadData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize, int iTimeout)
{
        int ret = 0;
        int  lPkNum, lPkRec, lDataMove, i;
        int reReadCnt = 0;
        int PackSize = COM_DATA_PACKET_SIZE;
```

```
        //分包
        lPkNum = (iSize)/PackSize;
        lPkRec = (iSize)%PackSize;
        lDataMove = 0;

        for(i = 0; i < lPkNum; i++)
        {
                ret = ReadDataPack(bCmd, bDataBuf, lDataMove, PackSize, iTimeout);
                if(ret == PackSize)
                {
                        reReadCnt = 0;
                        lDataMove += PackSize;
                } else {
                        //如果读取失败重试2次
                        i--;
                        if(reReadCnt++ > 2)
                        {
                                return XG_ERR_COM;
                        }
                }
        }

        if(lPkRec > 0)
        {
                ret = ReadDataPack(bCmd, bDataBuf, lDataMove, lPkRec, iTimeout);
                if(ret == lPkRec)
                {
                        lDataMove += lPkRec;
                }
        }
        if(lDataMove == iSize) return XG_ERR_SUCCESS;
        return XG_ERR_FAIL;
}
```

1.2.7 写入数据

```
/*
```

```
功能:写入一个数据包
参数:
bCmd: 指令码
bDataBuf: 要写入的数据缓存首地址
iOffset: 要写入的数据地址偏移
iSize: 要写入的数据大小，不能大于一个包的大小
返回值: = 0写入成功，非0写入失败
*/
static UINT8 WriteDataPack(UINT8 bCmd, UINT8* bDataBuf, UINT16 iOffset, UINT16 iSize)
{
	UINT8 ret = 0;
	UINT8 bBuf[UART_RECV_BYTE_MAX] = { 0 };
	UINT8 bCmdData[16] = { 0 };
	UINT16 i, CheckSum;

	if(iSize > COM_DATA_PACKET_SIZE) return XG_ERR_FAIL;

	bCmdData[0] = bCmd;
	bCmdData[1] = (iOffset&0xff);
	bCmdData[2] = ((iOffset>>8)&0xff);
	bCmdData[3] = 0;
	bCmdData[4] = 0;
	bCmdData[5] = (iSize&0xff);
	bCmdData[6] = ((iSize>>8)&0xff);
	bCmdData[7] = 0;
	bCmdData[8] = 0;

	ret = SendPack(XG_CMD_WRITE_DATA, bCmdData, 9);
	if(ret != XG_ERR_SUCCESS)
	{
		return ret;
	}
	for(i = 0; i < iSize; i++)
	{
		bBuf[i] = bDataBuf[i];
	}
```

```
        CheckSum = GetCheckSum(bBuf, iSize);
        bBuf[iSize] = CheckSum%256;
        bBuf[iSize+1] = CheckSum/256;
        UartSendBuf(bBuf, iSize+2);

        gUartByte = 0; //清除串口接收缓存
        ret = RecvPack(NULL, bCmdData, 1000);
        if(ret == XG_ERR_SUCCESS && bCmdData[0] == XG_ERR_SUCCESS)
        {
                return XG_ERR_SUCCESS;
        }

        return XG_ERR_FAIL;
}

/*
功能:分包写入数据
参数:
bCmd: 指令码
bDataBuf: 要写入的数据缓存首地址
iSize: 要写入的数据大小
返回值: = 0写入成功，非0写入失败
*/
static UINT8 WriteData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize)
{
        int ret = 0;
        int  lPkNum, lPkRec, lDataMove, i;
        int reWriteCnt = 0;
        int PackSize = COM_DATA_PACKET_SIZE;

        //分包
        lPkNum = (iSize)/PackSize;
        lPkRec = (iSize)%PackSize;
        lDataMove = 0;
```

```
        for(i = 0; i < lPkNum; i++)
        {
                ret = WriteDataPack(bCmd, bDataBuf, lDataMove, PackSize);
                if(ret == PackSize)
                {
                        reWriteCnt = 0;
                        lDataMove += PackSize;
                } else {
                        //如果发送失败重试2次
                        i--;
                        if(reWriteCnt++ > 2)
                        {
                                return XG_ERR_COM;
                        }
                }
        }

        if(lPkRec > 0)
        {
                ret = WriteDataPack(bCmd, bDataBuf, lDataMove, lPkRec);
                if(ret == lPkRec)
                {
                        lDataMove += lPkRec;
                }
        }

        if(lDataMove == iSize) return XG_ERR_SUCCESS;
        return XG_ERR_FAIL;
}
```

二、通讯指令Command 的详细说明

2.1 Command列表

```
#define XG_CMD_CONNECTION              0x01 //连接设备
#define XG_CMD_CLOSE_CONNECTION        0x02 //关闭连接
#define XG_CMD_GET_SYSTEM_INFO         0x03 //获取版本号和设置信息
#define XG_CMD_FACTORY_SETTING         0x04 //恢复出厂设置
#define XG_CMD_SET_DEVICE_ID           0x05 //设置设备编号-255
#define XG_CMD_SET_BAUDRATE            0x06 //设置波特率-4
#define XG_CMD_SET_SECURITYLEVEL       0x07 //设置安全等级-4
#define XG_CMD_SET_TIMEOUT             0x08 //设置等待手指放入超时
#define XG_CMD_SET_DUP_CHECK           0x09 //设置重复登录检查-1
#define XG_CMD_SET_PASSWORD            0x0A //设置通信密码
#define XG_CMD_CHECK_PASSWORD          0x0B //检查密码是否正确
#define XG_CMD_REBOOT                  0x0C //复位重启设备
#define XG_CMD_SET_SAME_FV             0x0D //登记的时候检查是否为同一根手指
#define XG_CMD_SET_USB_MODE            0x0E //设置USB驱动模式
#define XG_CMD_GET_DUID                0x0F //获取设备序列号
#define XG_CMD_FINGER_STATUS           0x10 //检测手指放置状态
#define XG_CMD_CLEAR_ENROLL            0x11 //清除指定ID登录数据
#define XG_CMD_CLEAR_ALL_ENROLL        0x12 //清除所有ID登录数据
#define XG_CMD_GET_EMPTY_ID            0x13 //获取空（无登录数据）ID
#define XG_CMD_GET_ENROLL_INFO         0x14 //获取总登录用户数和模板数
#define XG_CMD_GET_ID_INFO             0x15 //获取指定ID登录信息
#define XG_CMD_ENROLL                  0x16 //指定ID登录
#define XG_CMD_VERIFY                  0x17 //1:1认证或:N识别
#define XG_CMD_IDENTIFY_FREE           0x18 //无超时1:1认证或:N识别，可发0x19指令中断
#define XG_CMD_CANCEL                  0x19 //中断当前执行的指令
#define XG_CMD_READ_DATA               0x20 //从设备读取数据
#define XG_CMD_WRITE_DATA              0x21 //写入数据到设备
#define XG_CMD_READ_ENROLL             0x22 //读取指定ID登录数据
#define XG_CMD_WRITE_ENROLL            0x23 //写入（覆盖）指定ID登录数据
#define XG_CMD_GET_CHARA               0x28 //读取当前采集的特征
#define XG_CMD_READ_USER_DATA          0x29 //从用户扩展存储区读取数据，最大K
```

```
#define XG_CMD_WRITE_USER_DATA        0x2A //写入数据到用户扩展存储区，最大K
#define XG_CMD_GET_SYS_SET            0x2E //获取D700设备系统设置数据结构
#define XG_CMD_SET_SYS_SET            0x2F //下载D700系统设置数据结构
#define XG_CMD_OPEN_DOOR              0x32 //开门
#define XG_CMD_READ_LOG               0x33 //读取门禁出入日志
#define XG_CMD_SET_DEVNAME            0x34 //设置设备名称
#define XG_CMD_GET_DATETIME           0x35 //获取门禁实时时钟
#define XG_CMD_SET_DATETIME           0x36 //设置门禁实时时钟
#define XG_CMD_ENROLL_EXT             0x38 //扩展语音登记
#define XG_CMD_VERIFY_EXT             0x39 //扩展语音验证
#define XG_CMD_DEL_LOG                0x3a //删除门禁日志
#define XG_CMD_PLAY_VOICE             0x3b //播放语音
#define XG_CMD_REPORT                 0x4E //串口输出验证信息包
#define XG_CMD_GET_USER_BATCH         0x51 //批量读取用户信息
#define XG_CMD_SET_USER_BATCH         0x52 //批量写入用户信息
#define XG_CMD_READ_CARDNO            0x53 //D700请求用户刷卡并读取卡号
```

2.2 Command的详细说明

2.2.1 设备连接

- **[命令码]**

```
#define XG_CMD_CONNECTION                0x01 //连接设备,必须带位以上密码，默认密码全为(0x30)
```

- **[功能]**

连接目标设备,必须带8位以上密码，出厂设置密码为8个0（0x30）。

注意：除了该指令，其他所有指令必须连接成功后才能使用。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x01	XG_CMD_CONNECTION
bEncode	0x00	数据编码，为0
bDataLen	0x08	密码位数
bData	“00000000”	设备连接密码
wCheckSum	0x02EE	校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x01	XG_CMD_CONNECTION
bEncode	0x00	数据编码，为0
bDataLen		
bData	连接成功	连接成功后返回设备名称
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[14]为设备名称	
	连接失败	连接失败返回出错代码
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_PWD	
wCheckSum		校验和

- **[实例]**

```
/*
功能:通过串口发送连接指令并接收回包
Timeout:接收超时，单位毫秒
返回值:
XG_ERR_SUCCESS:连接成功
XG_ERR_FAIL:连接失败
XG_ERR_INVALID_PWD:密码错误
*/

UINT8 ConnectDev(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bCmd = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_CONNECTION, "00000000", 8);
	gUartByte = 0; //清除串口接收缓存
	ret = RecvPack(&bCmd, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bCmd == XG_CMD_CONNECTION)
		{
			if(bData[0] == XG_ERR_SUCCESS) return XG_ERR_SUCCESS;
			else return bData[1];
		}
	}
	return XG_ERR_FAIL;
}
```

2.2.2 关闭连接

- **[命令码]**

  ```
  #define XG_CMD_CLOSE_CONNECTION       0x02 //关闭连接
  ```

- **[功能]**

  关闭当前连接。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x02	XG_CMD_CLOSE_CONNECTION
bEncode	0x00	数据编码，为0
bDataLen	0x00	
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x02	XG_CMD_CLOSE_CONNECTION
bEncode	0x00	数据编码，为0
bDataLen		
bData	连接关闭成功	
	bData[0] = XG_ERR_SUCCESS	
	连接关闭失败	
wCheckSum		校验和

- **[实例]**

```
/*
功能:发送关闭连接指令
Timeout:接收超时，单位毫秒
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 CloseConnectDev(int Timeout)
{
```

```
        UINT8 ret = 0;
        UINT8 bCmd = 0;
        UINT8 bData[16] = { 0 };

        SendPack(XG_CMD_CLOSE_CONNECTION, NULL, 0);
        gUartByte = 0; //清除串口接收缓存
        ret = RecvPack(&bCmd, bData, Timeout);
        if(ret == XG_ERR_SUCCESS)
        {
                if(bData[0] == XG_ERR_SUCCESS) return XG_ERR_SUCCESS;
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.3 获取版本号和系统设置

- **[命令码]**

```
#define XG_CMD_GET_SYSTEM_INFO        0x03 //获取版本号和设置信息
```

- **[功能]**

  获取设备固件版本号和当前设置参数：设备编号，波特率，安全等级，检测手指超时，ID重复检查。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x03	XG_CMD_GET_SYSTEM_INFO
bEncode	0x00	数据编码，为0
bDataLen	0x00	
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x03	XG_CMD_GET_SYSTEM_INFO
bEncode	0x00	数据编码，为0
bDataLen		
bData	获取成功	
	bData[0] = XG_ERR_SUCCESS bData[1] = 主版本号 bData[2] = 子版本号; bData[3] = 设备编号(0-255) bData[4] = 波特率(0-4) bData[5] = 安全等级(0-4) bData[6] = 检测超时(1-255) bData[7] = 重复登记检查(0-1) bData[8] = 相同手指登记检查	
	获取失败	

	bData[1] = 语音输出音量(0-17)	
wCheckSum		校验和

- **[实例]**

```
/*
功能:获取设置信息
Timeout:接收超时，单位毫秒
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 GetDevSetting(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bCmd = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_GET_SYSTEM_INFO, NULL, 0);
	gUartByte = 0; //清除串口接收缓存
	ret = RecvPack(NULL, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			int iBaud[5] = {9600, 19200, 38400, 57600, 115200};
			UINT8 Mver = bData[1];
			UINT8 Sver = bData[2];
			UINT8 DevID = bData[3];
			int Baud = iBaud[bData[4]];
			UINT8 Securty = bData[5];
			UINT8 Timeout = bData[6];
			UINT8 DupCheck = bData[7];
			UINT8 SameFingerCheck = bData[8];
			UINT8 Volume = bData[11]; //0-16,>16语音关闭
```

```
                        return XG_ERR_SUCCESS;
                }
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.4 恢复出厂设置

- **[命令码]**

```
#define XG_CMD_FACTORY_SETTING        0x04 //恢复出厂设置
```

- **[功能]**

  恢复设备设置为出厂值。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x04	XG_CMD_FACTORY_SETTING
bEncode	0x00	数据编码，为0
bDataLen	0x00	
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x04	XG_CMD_FACTORY_SETTING
bEncode	0x00	数据编码，为0
bDataLen		
bData	恢复出厂设置成功	
	bData[0] = XG_ERR_SUCCESS	
	恢复出厂设置失败	
wCheckSum		校验和

- **[实例]**

```
/*
功能:恢复出厂设置
Timeout:接收超时，单位毫秒
返回值:
```

```
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevFactory(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_FACTORY_SETTING, NULL, 0);
	gUartByte = 0; //清除串口接收缓存
	ret = RecvPack(NULL, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS) return XG_ERR_SUCCESS;
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.5 设置设备编号

- **[命令码]**

```
#define XG_CMD_SET_DEVICE_ID          0x05 //设置设备编号0-255
```

- **[功能]**

设置设备编号，设备编号取值范围为0-255,0为默认编号。

1-255表示只接收对应bAddress编号的包。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x05	XG_CMD_SET_DEVICE_ID
bEncode	0x00	数据编码，为0
bDataLen	0x01	
bData	bData[0] = 设备编号	
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x05	XG_CMD_SET_DEVICE_ID
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	
wCheckSum		校验和

- **[实例]**

```
/*
功能:设置设备编号
bDevID:设备编号
```

```
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevID(UINT8 bDevID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bDevID;
	SendPack(XG_CMD_SET_DEVICE_ID, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.6 设置串口波特率

- **[命令码]**

```
#define XG_CMD_SET_BAUDRATE          0x06 //设置波特率0-4
```

- **[功能]**

  设置串口波特率。

  取值范围为0-4，分别对应波特率为9600,19200,38400,57600,115200。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x06	XG_CMD_SET_BAUDRATE
bEncode	0x00	数据编码，为0
bDataLen	0x01	
bData	bData[0] = 波特率索引	0:9600 1:19200 2:38400 3:57600 4:115200
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x06	XG_CMD_SET_BAUDRATE
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	
wCheckSum		校验和

- **[实例]**

```
/*
功能:设置波特率
bBaud:波特率,0=9600,1=19200,2=38400,3=57600,4-115200
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevBaud(UINT8 bBaud)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bBaud;
	SendPack(XG_CMD_SET_BAUDRATE, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.7 设置安全等级

- **[命令码]**

```
#define XG_CMD_SET_SECURITYLEVEL       0x07 //设置安全等级0-4
```

- **[功能]**

设置验证安全等级。

安全等级	识别精确度	
0	认假率 FAR（False Acceptance Rate）	**0.001%**
	拒识率 FRR（False Rejection Rate）	**0.001%**
1	认假率 FAR（False Acceptance Rate）	**0.0005%**
	拒识率 FRR（False Rejection Rate）	**0.01%**
2	认假率 FAR（False Acceptance Rate）	**0.0001%**
	拒识率 FRR（False Rejection Rate）	**0.05 %**

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x07	XG_CMD_SET_SECURITYLEVEL
bEncode	0x00	数据编码，为0
bDataLen	0x01	
bData	bData[0] = 安全等级	
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x07	XG_CMD_SET_SECURITYLEVEL
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	

wCheckSum		校验和

- **[API实例]**

```
/*
功能:设置安全等级
bSecurity:安全等级,0,1,2
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevSecurity(UINT8 bSecurity)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bSecurity;
	SendPack(XG_CMD_SET_SECURITYLEVEL, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.8 设置等待手指放入超时

- **[命令码]**

```
#define XG_CMD_SET_TIMEOUT            0x08 //设置等待手指放入超时1-255秒
```

- **[功能]**

在执行登录和认证指令时需要等待手指的输入，此参数就是设置等待超时，如超过此时间设置还未检测到手指输入，则会返回超时错误。

取值范围为1-255秒

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x08	XG_CMD_SET_TIMEOUT
bEncode	0x00	数据编码，为0
bDataLen	0x01	
bData	bData[0] = 超时秒数	1-255秒
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x08	XG_CMD_SET_TIMEOUT
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	
wCheckSum		校验和

- **[实例]**

```
/*
```

```
功能:设置手指检测超时
bTimeout:1-255秒
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/
UINT8 SetDevCheckFingerTimeout(UINT8 bTimeout)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bTimeout;
	SendPack(XG_CMD_SET_TIMEOUT, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.9 设置重复登记检查

- **[命令码]**

```
#define XG_CMD_SET_DUP_CHECK          0x09 //设置重复登记检查-1
```

- **[功能]**

在执行登记指令时，如果设置重复登记检查为是，则在登记的时候会同时进行1:N验证，如有相似用户，则会返回XG_ERR_DUPLICATION_ID错误。

取值范围为0（否）,1（是）

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x09	XG_CMD_SET_DUP_CHECK
bEncode	0x00	数据编码，为0
bDataLen	0x01	
bData	bData[0] = 0,1	否，是
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x09	XG_CMD_SET_DUP_CHECK
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	
wCheckSum		校验和

- **[实例]**

```
/*
```

```
功能:设置重复登记检查
bCheck:0不检查1检查
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevDupCheck(UINT8 bCheck)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bCheck;
	SendPack(XG_CMD_SET_DUP_CHECK, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.10 设置相同手指登记检查

- **[命令码]**

```
#define XG_CMD_SET_SAME_FV              0x0D //登记的时候检查是否为同一根手指
```

- **[功能]**

在执行登录指令时，如果设置相同手指登记检查为是，则在登记的时候会与此前登记信息比对，如不是同一根手指，则会返回XG_ERR_NO_SAME_FINGER错误。

取值范围为0（否）,1（是）

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x09	XG_CMD_SET_DUP_CHECK
bEncode	0x00	数据编码，为0
bDataLen	0x01	
bData	bData[0] = 0,1	否，是
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0D	XG_ERR_NO_SAME_FINGER
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	
wCheckSum		校验和

- **[实例]**

```
/*
```

```
功能:设置登记时是否同一根手指检查
bCheck:0不检查1检查
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevSameFingerCheck(UINT8 bCheck)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bCheck;
	SendPack(XG_CMD_SET_SAME_FV, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.11 设置新密码

- [命令码]

```
#define XG_CMD_SET_PASSWORD          0x0A //设置通信密码
```

- [功能]

设置新的连接密码，密码必须大于等于8位和小于等于14位，否则返回失败。

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0A	XG_CMD_SET_PASSWORD
bEncode	0x00	数据编码，为0
bDataLen	>= 8 && <= 14	密码位数
bData	新密码	新密码
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0A	XG_CMD_SET_PASSWORD
bEncode	0x00	数据编码，为0
bDataLen		
bData	设置成功	
	bData[0] = XG_ERR_SUCCESS	
	设置失败	
	bData[0] = XG_ERR_FAIL;	
wCheckSum		校验和

- [实例]

```
/*
功能:设置新的通讯密码
pPassword:需要设置的新密码
```

```
Len:密码的长度，>= 8 && <= 14
返回值:
XG_ERR_SUCCESS:设置成功
XG_ERR_FAIL:错误
*/

UINT8 SetCommPassword(UINT8* pPassword, UINT8 Len)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	if(Len >= 8 && Len <= 14)
	{
		memcpy(bData, pPassword, Len);
	} else {
		return XG_ERR_FAIL;
	}
	SendPack(XG_CMD_SET_PASSWORD, bData, Len);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.12 校验密码

- [命令码]

```
#define XG_CMD_CHECK_PASSWORD          0x0B //检查密码是否正确
```

- [功能]

  校验密码是否正确。

- [指令实例]

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0B	XG_CMD_CHECK_PASSWORD
bEncode	0x00	数据编码，为0
bDataLen	>= 8 && <= 14	密码位数
bData	密码	密码
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0B	XG_CMD_CHECK_PASSWORD
bEncode	0x00	数据编码，为0
bDataLen		
bData	校验成功	
	bData[0] = XG_ERR_SUCCESS	
	校验失败	
	bData[0] = XG_ERR_FAIL;	
wCheckSum		校验和

- [实例]

```
/*
功能:检查新设置的通讯密码是否正确
pPassword:密码
```

```
Len:密码长度
返回值:
XG_ERR_SUCCESS:设置成功
XG_ERR_FAIL:错误
*/

UINT8 CheckCommPassword(UINT8* pPassword, UINT8 Len)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	if(Len >= 8 && Len <= 14)
	{
		memcpy(bData, pPassword, Len);
	} else {
		return XG_ERR_FAIL;
	}
	SendPack(XG_CMD_CHECK_PASSWORD, bData, Len);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.13 复位重启设备

- [命令码]

```
#define XG_CMD_REBOOT                0x0C //复位重启设备
```

- [功能]

  复位并重新启动设备。

- [指令实例]

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0C	XG_CMD_REBOOT
bEncode	0x00	数据编码，为0
bDataLen		
bData		
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0C	XG_CMD_REBOOT
bEncode	0x00	数据编码，为0
bDataLen		
bData	复位成功	
	bData[0] = XG_ERR_SUCCESS	
	复位失败	
wCheckSum		校验和

- [实例]

```
/*
功能:复位重启设备
返回值:
```

```
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevReset()
{
	UINT8 ret = 0;

	SendPack(XG_CMD_SET_SAME_FV, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.14 检测手指输入状态

- [命令码]

```
#define XG_CMD_FINGER_STATUS          0x10 //检测手指放置状态
```

- [功能]

检测手指是否正确放置。

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x10	XG_CMD_FINGER_STATUS
bEncode	0x00	数据编码，为0
bDataLen		
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x10	XG_CMD_FINGER_STATUS
bEncode	0x00	数据编码，为0
bDataLen		
bData	检测成功	只有手指正确放置才返回1，否则返回0
	bData[0] = XG_ERR_SUCCESS bData[1] = 0无手指，1有手指	
wCheckSum		校验和

- [实例]

```
/*
功能:检测手指状态
```

```
返回值:
0:无手指
1:有手指
*/

UINT8 CheckFinger()
{
	UINT8 ret = 0;

	SendPack(XG_CMD_FINGER_STATUS, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return bData[1];
		}
	}
	return 0;
}
```

2.2.15 清除指定ID登录数据

- [命令码]

```
#define XG_CMD_CLEAR_ENROLL            0x11 //清除指定ID登录数据
```

- [功能]

  删除指定ID用户的登记数据。

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x11	XG_CMD_CLEAR_ENROLL
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
		ID为需要删除的用户编号

wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x11	XG_CMD_CLEAR_ENROLL
bEncode	0x00	数据编码，为0
bDataLen		
bData	删除成功	
	bData[0] = XG_ERR_SUCCESS	
	删除失败	删除失败： XG_ERR_INVALID_ID： 用户ID非法。 XG_ERR_EMPTY_ID： 此用户ID无登录数据
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_EMPTY_ID	
wCheckSum		校验和

- **[实例]**

```
/*
功能:清除指定用户数据
UserID:清除指定用户ID
返回值:
XG_ERR_SUCCESS:清除成功
XG_ERR_FAIL:清除失败
*/

UINT8 CleanUser(UINT16 UserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = (UserID)&0xff;
	bData[1] = ((UserID)>>8)&0xff;
	SendPack(XG_CMD_CLEAR_ENROLL, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.16 清除所有用户登录数据

- [命令码]

  ```
  #define XG_CMD_CLEAR_ALL_ENROLL        0x12 //清除所有ID登录数据
  ```

  注意：此指令执行时间比较长，不建议使用，建议一个个用户清除

- [功能]

  删除所有用户登记数据。

- [指令实例]

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x12	XG_CMD_CLEAR_ALL_ENROLL
bEncode	0x00	数据编码，为0
bDataLen		
bData		
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x12	XG_CMD_CLEAR_ALL_ENROLL
bEncode	0x00	数据编码，为0
bDataLen		
bData	删除成功	
	bData[0] = XG_ERR_SUCCESS	
wCheckSum		校验和

- [实例]

```
UINT8 CleanAllUser()
{
```

```
	UINT8 ret = 0;
	UINT16 UserNum = 0;
	UINT16 UserMax = 0;

	//先获取登记有多少用户,估算一下大概需要多长的清除时间
	ret = GetEnrollInfo(&UserNum, &UserMax);
	if(ret == XG_ERR_SUCCESS)
	{
		int timeout = 1000 + UserNum*300;
		UINT8 bData[16] = { 0 };

		SendPack(XG_CMD_CLEAR_ALL_ENROLL, NULL, 0);
		gUartByte = 0;
		ret = RecvPack(NULL, bData, timeout);
		if(ret == XG_ERR_SUCCESS)
		{
			if(bData[0] == XG_ERR_SUCCESS)
			{
				return XG_ERR_SUCCESS;
			}
			else return bData[1];
		}
	}
	return XG_ERR_FAIL;
}
```

2.2.17 获取空（可用）ID

- **[命令码]**

```
#define XG_CMD_GET_EMPTY_ID          0x13 //获取空（无登录数据）ID
```

- **[功能]**

  获取空的（无登录数据）的ID号。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x13	XG_CMD_GET_EMPTY_ID
bEncode	0x00	数据编码，为0

bDataLen	bData[0] = StartId&0xff;
	bData[1] = (StartId >>8)&0xff;
	bData[2] = (StartId >>16)&0xff;
	bData[3] = (StartId >>24)&0xff;
	bData[4] = EndId&0xff;
	bData[5] = (EndId >>8)&0xff;
	bData[6] = (EndId >>16)&0xff;
	bData[7] = (EndId >>24)&0xff;
		StartId为0或EndId为0默认是从所有用户中找空闲ID,否则从StartId到EndId指定范围的用户中找空闲ID

bData		
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x13	XG_CMD_GET_EMPTY_ID
bEncode	0x00	数据编码，为0
bDataLen		
bData	获取成功 bData[0] = XG_ERR_SUCCESS	ID = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]

	ID:bData[1]-bData[4]	<<24);
	获取失败	注册满，已无空ID可用
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_NO_NULL_ID	
wCheckSum		校验和

- **[实例]**

```
/*
功能:获取空ID，也就是没有登记的ID
pUserID:返回可登记ID的指针
返回值:
XG_ERR_SUCCESS:接收成功
XG_ERR_FAIL:通讯错误
XG_ERR_NO_NULL_ID:无空ID
*/

UINT8 GetNullID(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(XG_CMD_GET_EMPTY_ID, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.18 获取系统登录信息（登录用户数，模板数）

- **[命令码]**

```
#define XG_CMD_GET_ENROLL_INFO        0x14 //获取总登录用户数和模板数
```

- **[功能]**

  获取系统登录信息，包括已登录用户数，已登录模板数和允许最大登记用户数。每个用户最多登录3个模板，所以，允许最大的登记模板数为最大登记用户数*3。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x14	XG_CMD_GET_ENROLL_INFO
bEncode	0x00	数据编码，为0
bDataLen		
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x14	XG_CMD_GET_ENROLL_INFO
bEncode	0x00	数据编码，为0
bDataLen		
bData	获取成功	登录用户数 = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24); 最大用户数 = bData[9] + (bData[10]<<8) + (bData[11]<<16) + (bData[12]<<24);
	bData[0] = XG_ERR_SUCCESS EnrollUser:bData[1]-bData[4] EnrollMax:bData[9]-bData[12]	

wCheckSum		校验和

- **[实例]**

```
/*
功能:获取登记信息
pUserNum:已经登记的用户数
pUserMax:最大用户数
返回值:
XG_ERR_SUCCESS:获取成功
XG_ERR_FAIL:获取失败
*/

UINT8 GetEnrollInfo(UINT16* pUserNum, UINT16* pUserMax)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_GET_ENROLL_INFO, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pUserNum) *pUserNum = bData[1] + (bData[2]<<8);
			if(pUserMax) *pUserMax = bData[9] + (bData[10]<<8);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.19 获取指定ID登录信息

- **[命令码]**

```
#define XG_CMD_GET_ID_INFO              0x15 //获取指定ID登录信息
```

- **[功能]**

获取指定ID的登录模板数，模板数为0即为空（无登录）ID。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x15	XG_CMD_GET_ID_INFO
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;	

wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x15	XG_CMD_GET_ID_INFO
bEncode	0x00	数据编码，为0
bDataLen		
bData	获取成功	
	bData[0] = XG_ERR_SUCCESS bData[1] = 模板数	
	获取失败	非法ID
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID	
wCheckSum		校验和

- **[实例]**

```
/*
功能:获取指定用户DI的登记信息
UserID:指定ID
返回值:
0:此用户无登记
> 0:此用户已登记
*/

UINT8 GetIDEnroll(UINT16 UserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = (UserID)&0xff;
	bData[1] = ((UserID)>>8)&0xff;
	SendPack(XG_CMD_GET_ID_INFO, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return bData[1];
		}
	}
	return 0;
}
```

2.2.20 用户登记

- **[命令码]**

```
#define XG_CMD_ENROLL                0x16 //指定ID登录
```

- **[功能]**

  指定ID登记，一次登记可以采集单个模板，也可以采集多个模板（最多3个模板）。

  注意：

  此指令正常登记会返回多个返回包，只有接收到XG_ERR_SUCCESS或XG_ERR_FAIL返回值时才结束登记过程。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x16	XG_CMD_ENROLL
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
	bData[4] = 分组组号
	bData[5] = 此次登记模板数	

wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x16	XG_CMD_ENROLL
bEncode	0x00	数据编码，为0
bDataLen		
bData	登录成功	
	bData[0] = XG_ERR_SUCCESS	
	登录失败	登录失败： XG_ERR_INVALID_ID：非法ID
	bData[0] = XG_ERR_FAIL	

	bData[1] = XG_ERR_INVALID_ID/ XG_ERR_NOT_ENOUGH/ XG_ERR_TIME_OUT/ XG_ERR_DUPLICATION_ID	XG_ERR_NOT_ENOUGH：已无登录空间 XG_ERR_TIME_OUT：等待手指输入超时 XG_ERR_DUPLICATION_ID：有相似用户已经登录
	XG_INPUT_FINGER:	提示用户放入手指
	XG_RELEASE_FINGER	提示用户拿开手指
wCheckSum		校验和

- **[实例]**

```
/*
功能:发送登记指令增加指静脉用户
UserID:指定登记用户的ID
返回值:
XG_ERR_SUCCESS:登记成功
*/

UINT8 EnrollUser(UINT16 UserID)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };

        bData[0] = UserID&0xff;
        bData[1] = (UserID>>8)&0xff;
        bData[2] = 0;
        bData[3] = 0;
        bData[4] = 0;
        bData[5] = 3; //采集成功的次数，只要采集了次成功就登记成功
        bData[10] = 10; //如果采集不成功一共可重试多少次
        SendPack(XG_CMD_ENROLL, bData, 12);
        gUartByte = 0;
        for(;;)
        {
                ret = RecvPack(NULL, bData, 6000); //手指检测超时是秒
```

```
if(ret != XG_ERR_SUCCESS)
{
        PLAY_SOUND(VOICE_REG_FAIL);
        break;
}
if(bData[0] == XG_ERR_SUCCESS)
{
        PLAY_SOUND(VOICE_REG_OK);
        break;
} else if(bData[0] == XG_INPUT_FINGER) {
        //bData[1]是第几次采集
        if(bData[1] == 0) PLAY_SOUND(VOICE_INPUT);
} else if(bData[0] == XG_RELEASE_FINGER) {
        //指静脉采集完成，可以拿开手指了
        PLAY_SOUND(VOICE_KEY);
        DelayMs(200);
        if(bData[1] == 0 || bData[1] == 1)
        {
                PLAY_SOUND(VOICE_INPUT_FINGER_AGAIN);
        }
} else {
        //错误处理
        ret = bData[1];
        if(ret == XG_ERR_INVALID_ID)
        {
                //非法ID
                PLAY_SOUND(VOICE_PINPUT_ID); //ID错误
        } else if(ret == XG_ERR_NOT_ENOUGH) {
                //此用户已登记
                PLAY_SOUND(VOICE_USER_FULL);
        } else if(ret == XG_ERR_TIME_OUT) {
                //手指检测超时
                PLAY_SOUND(VOICE_REG_FAIL);
        } else if(ret == XG_ERR_DUPLICATION_ID) {
                //重复登记
                PLAY_SOUND(VOICE_OVER_REG);
```

```
                } else if(ret == XG_ERR_NO_SAME_FINGER) {
                        //不是同一根手指
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else if(ret == XG_ERR_NO_VEIN) {
                        //没有检测到指静脉
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else {
                        PLAY_SOUND(VOICE_REG_FAIL);
                }
                break;
            }
        }
        return ret;
}
```

2.2.21 用户验证

- **[命令码]**

```
#define XG_CMD_VERIFY                0x17 //1:1认证或:N识别
```

- **[功能]**

  如指定用户ID则为1:1验证模式。

  如指定用户ID为0则为1：N识别验证模式。

  注意：

  此指令正常验证会返回多个返回包，只有接收到XG_ERR_SUCCESS或XG_ERR_FAIL返回值时才结束验证过程。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x17	XG_CMD_VERIFY
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
	bData[4] = 分组组号
	bData[5] = 连续验证几个ID
		ID = 0为1：N验证方式
		ID > 0为1：1验证方式

wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x17	XG_CMD_VERIFY
bEncode	0x00	数据编码，为0
bDataLen		
bData	验证成功	识别的用户ID = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[4]	

	验证失败	
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID / XG_ERR_TIME_OUT	XG_ERR_TIME_OUT：等待手指输入超时
	XG_INPUT_FINGER:	提示用户放入手指
	XG_RELEASE_FINGER	提示用户拿开手指
wCheckSum		校验和

- **[实例]**

```
/*
功能:发送验证指令并获取返回的验证结果
pUserID:返回验证成功用户ID的指针
返回值:
XG_ERR_SUCCESS:验证成功，验证成功的用户ID通过pUserID返回
XG_ERR_NO_VEIN:没有成功采集指静脉，可能手指没放好
*/

UINT8 VerifyUser(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT32 CardNo = 0;

	if(pUserID)
	{
		//如果pUserID为非则:1验证
		bData[0] = (*pUserID)&0xff;
		bData[1] = ((*pUserID)>>8)&0xff;
		bData[2] = 0;
		bData[3] = 0;
		bData[4] = 0; //0为不分组
		bData[5] = 0; //0为单个用户验证,>0为UserID开始后的连续几个用户验证
		bData[6] = CardNo&0xff; //还可以按卡号进行:1验证
		bData[7] = (CardNo>>8)&0xff;
		bData[8] = (CardNo>>16)&0xff;
```

```
        bData[9] = (CardNo>>24)&0xff;
}

SendPack(XG_CMD_VERIFY, bData, 10);
gUartByte = 0;
for(;;)
{
        ret = RecvPack(NULL, bData, 6000); //手指检测超时是秒
        if(ret != XG_ERR_SUCCESS)
        {
                PLAY_SOUND(VOICE_IDENT_FAIL);
                break;
        }
        if(bData[0] == XG_ERR_SUCCESS)
        {
                PLAY_SOUND(VOICE_IDENT_OK);
                if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
                break;
        } else if(bData[0] == XG_INPUT_FINGER) {

        } else if(bData[0] == XG_RELEASE_FINGER) {
                //指静脉采集完成，可以拿开手指了
                PLAY_SOUND(VOICE_KEY);
        } else {
                //错误处理
                ret = bData[1];
                if(ret == XG_ERR_NO_VEIN)
                {
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else {
                        PLAY_SOUND(VOICE_IDENT_FAIL);
                }
                break;
        }
}
return ret;
```

```
}
```

2.2.22 扩展登记指令

- **[命令码]**

```
#define XG_CMD_ENROLL_EXT              0x38 //扩展语音登记
```

- **[功能]**

  获取设备的序列号

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x38	XG_CMD_ENROLL_EXT
bEncode	0x00	数据编码，为0
bDataLen		
bData		
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x38	XG_CMD_ENROLL_EXT
bEncode	0x00	数据编码，为0
bDataLen		
bData	登记成功	登记成功的ID = bData[1] + (bData[2]<<8)
	bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[2]	
	登记失败	
	bData[0] = XG_ERR_FAIL bData[1] = 错误码	
wCheckSum		校验和

- **[实例]**

```
/*
```

```
功能:发送扩展登记指令增加指静脉用户,登记过程通过语音提示用户操作过程，没有语音的设备不适用
UserID:指定登记用户的ID
返回值:
XG_ERR_SUCCESS:登记成功
*/

UINT8 EnrollUserEXT(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT16 UserID = *pUserID;

	bData[0] = UserID&0xff;
	bData[1] = (UserID>>8)&0xff;
	SendPack(XG_CMD_ENROLL_EXT, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 5000*3); //一次手指检测超时是秒，次至少要秒
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			//如果UserID为，此指令会自动找一个空ID进行登记，并返回此ID
			if(pUserID)
				*pUserID = bData[1] + bData[2]*256;
			return XG_ERR_SUCCESS;
		} else {
			return bData[1];
		}
	}
	return ret;
}
```

2.2.23 扩展验证指令

- **[命令码]**

```
#define XG_CMD_VERIFY_EXT              0x39 //扩展语音验证
```

- **[功能]**

  发指令验证只有一个回包

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x39	XG_CMD_VERIFY_EXT
bEncode	0x00	数据编码，为0
bDataLen		
bData		
wCheckSum		校验和

  返回包：

<table>
<tr><td>字段</td><td>数据</td><td>备注</td></tr>
<tr><td>wPrefix</td><td>0xAABB</td><td>包标识</td></tr>
<tr><td>bAddress</td><td>0x00</td><td>目标设备地址</td></tr>
<tr><td>bCmd</td><td>0x39</td><td>XG_CMD_VERIFY_EXT</td></tr>
<tr><td>bEncode</td><td>0x00</td><td>数据编码，为0</td></tr>
<tr><td>bDataLen</td><td></td><td></td></tr>
<tr><td rowspan="4">bData</td><td>验证成功</td><td rowspan="2">登记成功的ID = bData[1] + (bData[2]<<8)</td></tr>
<tr><td>bData[0] = XG_ERR_SUCCESS
ID:bData[1]-bData[2]</td></tr>
<tr><td>验证失败</td><td rowspan="2"></td></tr>
<tr><td>bData[0] = XG_ERR_FAIL</td></tr>
<tr><td>wCheckSum</td><td></td><td>校验和</td></tr>
</table>

- **[实例]**

```
/*
功能:发送扩展验证指令，此指令只有一个回包
```

```
pUserID:返回验证成功用户ID的指针
返回值:
XG_ERR_SUCCESS:验证成功，验证成功的用户ID通过pUserID返回
XG_ERR_NO_VEIN:没有成功采集指静脉，可能手指没放好
*/

UINT8 VerifyUserEXT(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT32 CardNo = 0;

	if(pUserID)
	{
		//如果pUserID为非则:1验证
		bData[0] = (*pUserID)&0xff;
		bData[1] = ((*pUserID)>>8)&0xff;
		bData[2] = 0;
		bData[3] = 0;
		bData[4] = 0; //0为不分组
		bData[5] = 0; //0为单个用户验证,>0为UserID开始后的连续几个用户验证
		bData[6] = CardNo&0xff; //还可以按卡号进行:1验证
		bData[7] = (CardNo>>8)&0xff;
		bData[8] = (CardNo>>16)&0xff;
		bData[9] = (CardNo>>24)&0xff;
	}

	//XG_CMD_VERIFY_EXT指令是没有提示放手指和拿开手指的回包的
	SendPack(XG_CMD_VERIFY_EXT, bData, 10);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 6000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			PLAY_SOUND(VOICE_IDENT_OK);
```

```
                        if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
                } else {
                        ret = bData[1];
                        if(ret == XG_ERR_NO_VEIN)
                        {
                                PLAY_SOUND(VOICE_RIGHT_INPUT);
                        } else {
                                PLAY_SOUND(VOICE_IDENT_FAIL);
                        }
                }
        }
        return ret;
}
```

2.2.24 读取数据

- **[命令码]**

```
#define XG_CMD_READ_DATA                0x20 //从设备读取数据
```

- **[功能]**

从设备读取数据,支持读取的数据类型有登录数据，图像数据和调试信息，必须与以下指令配合使用:

数据类型	命令码	备注
XG_CMD_READ_ENROLL	0x22	读取指定ID登录数据
XG_CMD_GET_CHARA	0x28	读取特征数据
XG_CMD_GET_DEBUG	0x26	读取调试信息

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x20	XG_CMD_READ_DATA
bEncode	0x00	数据编码，为0
bDataLen		
bData	bData[0] = 数据类型 bData[1] - bData[4]:数据偏移 bData[5] - bData[8]:数据大小，串口数据包大小不能超过512字节	数据类型： XG_CMD_READ_ENROLL XG_CMD_READ_CHARA XG_CMD_GET_DEBUG
wCheckSum		校验和

返回包：

数据 + 校验和(2字节)

- **[实例]**

```
/*
功能:分包读取数据
参数:
bCmd: 指令码
```

```
bDataBuf: 要读取的数据缓存首地址
iSize: 要读取的数据大小
iTimeout: 读取一个包的超时
返回值: = 0读取成功，非0读取失败
*/
static UINT8 ReadData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize, int iTimeout)
{
	int ret = 0;
	int  lPkNum, lPkRec, lDataMove, i;
	int reReadCnt = 0;
	int PackSize = COM_DATA_PACKET_SIZE;

	//分包
	lPkNum = (iSize)/PackSize;
	lPkRec = (iSize)%PackSize;
	lDataMove = 0;

	for(i = 0; i < lPkNum; i++)
	{
		ret = ReadDataPack(bCmd, bDataBuf, lDataMove, PackSize, iTimeout);
		 if(ret == PackSize)
		 {
			reReadCnt = 0;
			lDataMove += PackSize;
			bDataBuf += PackSize;
		 } else {
			//如果读取失败重试2次
			i--;
			if(reReadCnt++ > 2)
			{
				return XG_ERR_COM;
			}
		 }
	}

	if(lPkRec > 0)
```

```
	{
		ret = ReadDataPack(bCmd, bDataBuf, lDataMove, lPkRec, iTimeout);
		if(ret == lPkRec)
		{
			lDataMove += lPkRec;
		}
	}
	if(lDataMove == iSize) return XG_ERR_SUCCESS;
	return XG_ERR_FAIL;
}
```

2.2.25 写入数据

- **[命令码]**

```
#define XG_CMD_WRITE_DATA              0x21 //写入数据到设备
```

- **[功能]**

写入指定类型数据到设备,支持写入的数据类型有登录数据，图像数据和升级程序，必须与以下指令配合使用:

数据类型	命令码	备注
XG_CMD_WRITE_ENROLL	0x23	写入指定ID登录数据
XG_CMD_WRITE_IMAGE	0x25	写入图像数据
XG_CMD_UPGRADE	0x27	写入升级程序（在线升级）

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x21	XG_CMD_WRITE_DATA
bEncode	0x00	数据编码，为0
bDataLen		
bData	bData[0] = 数据类型 bData[1] - bData[4]:数据偏移 bData[5] - bData[8]:数据大小，串口数据包大小不能超过512字节	数据类型： XG_CMD_READ_ENROLL XG_CMD_READ_IMAGE XG_CMD_GET_DEBUG
wCheckSum		校验和
WriteData	Data[0]…Data[N]	写入的数据
DataCheckSum	Data[0]+Data[1]…+Data[N]	数据校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x21	XG_CMD_WRITE_DATA
bEncode	0x00	数据编码，为0
bDataLen	0x01	

bData	成功	
	bData[0] = XG_ERR_SUCCESS	
	失败	
	bData[0] = XG_ERR_FAIL	
	数据校验错误	
	bData[0] = XG_ERR_DATA	
wCheckSum		校验和

- **[实例]**

```
/*
功能:分包写入数据
参数:
bCmd: 指令码
bDataBuf: 要写入的数据缓存首地址
iSize: 要写入的数据大小
返回值: = 0写入成功，非0写入失败
*/
static UINT8 WriteData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize)
{
	int ret = 0;
	int  lPkNum, lPkRec, lDataMove, i;
	int reWriteCnt = 0;
	int PackSize = COM_DATA_PACKET_SIZE;

	//分包
	lPkNum = (iSize)/PackSize;
	lPkRec = (iSize)%PackSize;
	lDataMove = 0;

	for(i = 0; i < lPkNum; i++)
	{
		ret = WriteDataPack(bCmd, bDataBuf, lDataMove, PackSize);
		if(ret == XG_ERR_SUCCESS)
		{
			reWriteCnt = 0;
```

```
                lDataMove += PackSize;
        } else {
                //如果发送失败重试2次
                i--;
                if(reWriteCnt++ > 2)
                {
                        return XG_ERR_COM;
                }
        }
    }

    if(lPkRec > 0)
    {
        ret = WriteDataPack(bCmd, bDataBuf, lDataMove, lPkRec);
        if(ret == XG_ERR_SUCCESS)
        {
                lDataMove += lPkRec;
        }
    }

    if(lDataMove == iSize) return XG_ERR_SUCCESS;
    return XG_ERR_FAIL;
}
```

2.2.26 读取指定ID登记数据

- [命令码]

```
#define XG_CMD_READ_ENROLL              0x22 //读取指定ID登录数据
```

- [功能]

读取指定ID登记数据，读取前先发送此指令检测是否满足读取条件，如返回成功则可以读取数据，并返回需要读取数据的大小，再利用XG_CMD_READ_DATA指令读取具体数据。

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x22	XG_CMD_READ_ENROLL
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	buf[0] = ID&0xff;
	buf[1] = (ID >>8)&0xff;
	buf[2] = (ID >>16)&0xff;
	buf[3] = (ID >>24)&0xff;	

wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x22	XG_CMD_READ_ENROLL
bEncode	0x00	数据编码，为0
bDataLen		
bData	成功	size =bData[1] + (bData[2] << 8) + (bData[3] << 16) + (bData[4] << 24);
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[4]:数据大小	
	失败	XG_ERR_INVALID_ID：非法ID XG_ERR_EMPTY_ID：空ID，此ID无登录数据
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_EMPTY_ID	

wCheckSum		校验和

- **[实例]**

```
/*
功能:读取指定用户ID的模板
参数:
iUserID: 用户ID
bTempBuf: 模板数据缓存，至少要有字节的缓存空间
返回值: > 0模板数据的大小= 0读取失败
*/
UINT16 ReadDevTemp(UINT16 iUserID, UINT8* bTempBuf)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = iUserID%256;
	bData[1] = iUserID/256;
	bData[2] = 0;
	bData[3] = 0;
	ret = SendPack(XG_CMD_READ_ENROLL, bData, 4);
	if(ret == XG_ERR_SUCCESS) {
		ret = RecvPack(NULL, bData, 1000);
		if (ret == XG_ERR_SUCCESS) {
			if(bData[0] == XG_ERR_SUCCESS) {
				UINT16 TempSize = bData[1] + bData[2]*256;
				ret = ReadData(XG_CMD_READ_ENROLL, bTempBuf, TempSize, 1000);
				if (ret == XG_ERR_SUCCESS) {
					return TempSize;
				}
			} else {
				//bData[1]是错误代码
			}
		}
	}
	return 0;
```

```
}
```

2.2.27 写入指定ID登记数据

- [命令码]

```
#define XG_CMD_WRITE_ENROLL          0x23 //写入（覆盖）指定ID登录数据
```

- [功能]

写入指定ID登记数据，写入前先发送此指令检测是否满足写入条件，如返回成功则可以写入数据，再利用XG_CMD_WRITE_DATA指令写入具体数据。

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x23	XG_CMD_WRITE_ENROLL
bEncode	0x00	数据编码，为0
bDataLen	0x08	

bData	buf[0] = ID&0xff;
	buf[1] = (ID >>8)&0xff;
	buf[2] = (ID >>16)&0xff;
	buf[3] = (ID >>24)&0xff;
	buf[4] = SIZE&0xff;
	buf[5] = (SIZE >>8)&0xff;
	buf[6] = (SIZE >>16)&0xff;
	buf[7] = (SIZE >>24)&0xff;
		指定用户ID和需要写入的数据大小

wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x23	XG_CMD_WRITE_ENROLL
bEncode	0x00	数据编码，为0
bDataLen		
bData	成功	
	bData[0] = XG_ERR_SUCCESS	
	失败	XG_ERR_INVALID_ID：非法ID

	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_INVALID_PARAM	XG_ERR_INVALID_PARAM：参数错误，可能是数据的大小不匹配
wCheckSum		校验和

- **[实例]**

```
/*
功能:写入指定用户ID的模板
参数:
iUserID: 用户ID
bTempBuf: 要写入的模板数据缓存
iSize: 模板数据的大小
返回值: = 0写入成功非写入失败
*/
UINT8 WriteDevTemp(UINT16 iUserID, UINT8* bTempBuf, UINT16 iSize)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = iUserID%256;
	bData[1] = iUserID/256;
	bData[2] = 0;
	bData[3] = 0;
	bData[4] = iSize%256;
	bData[5] = iSize/256;
	bData[6] = 0;
	bData[7] = 0;
	ret = SendPack(XG_CMD_WRITE_ENROLL, bData, 8);
	if(ret == XG_ERR_SUCCESS) {
		ret = RecvPack(NULL, bData, 1000);
		if (ret == XG_ERR_SUCCESS) {
			if(bData[0] == XG_ERR_SUCCESS) {
				ret = WriteData(XG_CMD_WRITE_ENROLL, bTempBuf, iSize);
				return ret;
			}
```

```
                } else {
                        //bData[1]是错误代码
                }
        }
        return XG_ERR_FAIL;
}
```

2.2.28 采集并读取特征到主机

- [命令码]

```
#define XG_CMD_GET_CHARA                0x28 //采集并读取特征到主机
```

- [功能]

采集当前手指的静脉特征数据，如采集成功，再利用XG_CMD_READ_DATA指令读取具体数据。

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x28	XG_CMD_GET_CHARA
bEncode	0x00	数据编码，为0
bDataLen	0x00	
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x28	XG_CMD_GET_CHARA
bEncode	0x00	数据编码，为0
bDataLen		
bData	成功	会返回多个包，提示放入手指和拿开手指
	bData[0] = XG_INPUT_FINGER / MSG_OUTPUT_FINGER /XG_ERR_SUCCESS	
	失败	XG_ERR_TIME_OUT：没有检测到手指超时返回
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_TIME_OUT	
wCheckSum		校验和

- [实例]

```
/*
```

```
功能:采集指静脉特征
参数:
bCharaBuf: 特征数据缓存，至少要有字节的缓存空间
lTimeout: 等待手指放入的超时
返回值: > 0特征数据的大小= 0采集失败
*/
UINT16 GetVeinChara(UINT8* bCharaBuf, int lTimeout)
{
	//发送采集特征指令
	if(SendPack(XG_CMD_GET_CHARA, NULL, 0) == XG_ERR_SUCCESS)
	{
		for(;;)
		{
			UINT8 bData[16] = { 0 };
			int ret = RecvPack(NULL, bData, lTimeout);
			if (ret == XG_ERR_SUCCESS) { //接收到回包
				if(bData[0] == XG_ERR_SUCCESS) { //采集成功
					int CharaSize = bData[1] + bData[2]*255;
					ret = ReadData(XG_CMD_GET_CHARA, bCharaBuf, CharaSize, 1000);
					if (ret == XG_ERR_SUCCESS) {
						return CharaSize;
					}
					break;
				} else if(bData[0] == XG_ERR_FAIL) {
					//bData[1]是采集失败的错误代码
					return 0;
				} else if(bData[0] == XG_INPUT_FINGER) {
				} else if(bData[0] == XG_RELEASE_FINGER) {
				} else {
					break;
				}
			} else {
				break;
			}
		}
	}
```

```
        return 0;
}
```

2.2.29 获取设备序列号

- **[命令码]**

```
#define XG_CMD_GET_DUID                0x0F //获取设备序列号
```

- **[功能]**

  获取设备序列号

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0F	XG_CMD_GET_DUID
bEncode	0x00	数据编码，为0
bDataLen	0x05	
bData		
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x0F	XG_CMD_GET_DUID
bEncode	0x00	数据编码，为0
bDataLen		
bData	成功	
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[14] 序列号	
	失败	
	bData[0] = XG_ERR_FAIL	
wCheckSum		校验和

- **[实例]**

```
/*
功能:获取设备序列号
```

```
pSN:返回设备序列号
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:错误
*/

UINT8 GetDevSN(UINT8* pSN)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(XG_CMD_GET_DUID, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pSN) memcpy(pSN, bData[1], 14);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.30 播放采集仪语音

- [命令码]

```
#define XG_CMD_PLAY_VOICE              0x3b //播放语音
```

- [功能]

播放采集仪语音，只有支持语音的设备才适用

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x3B	XG_CMD_PLAY_VOICE
bEncode	0x00	数据编码，为0
bDataLen	0x05	
bData	bData[0]=语音序号，见附录 bData[1]=模式,0异步，1同步	
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x3B	XG_CMD_PLAY_VOICE
bEncode	0x00	数据编码，为0
bDataLen		
bData	成功	
	bData[0] = XG_ERR_SUCCESS	
	失败	
	bData[0] = XG_ERR_FAIL	
wCheckSum		校验和

- [实例]

```
/*
功能:播放设备语音，只要支持语音的设备才适用
```

```
bSound:语音序号
bMode:0异步，同步
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 PlayDevSound (UINT8 bSound, UINT8 bMode)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };

        bData[0] = bSound;
        bData[1] = bMode;
        SendPack(XG_CMD_PLAY_VOICE, bData, 2);
        gUartByte = 0;
        ret = RecvPack(NULL, bData, 1000); //异步的话此超时要延长，具体时间根据语音的长短
        if(ret == XG_ERR_SUCCESS)
        {
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        return XG_ERR_SUCCESS;
                }
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.31 门禁扩展开门指令

- [命令码]

```
#define XG_CMD_OPEN_DOOR                0x32 //开门指令
```

- [功能]

  发送指令开门，在门禁或锁上适用

- [指令实例]

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x32	XG_CMD_OPEN_DOOR
bEncode	0x00	数据编码，为0
bDataLen	0x05	
bData	bData[0-3] = 开门ID	
wCheckSum		校验和

- [实例]

```
/*
功能:指令开门
UserId:以那个用户ID的身份去开门
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 DevOpenDoor(UINT8 UserId)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };

        bData[0] = UserId%256;
        bData[1] = UserId/256;
        SendPack(XG_CMD_OPEN_DOOR, bData, 2);
        gUartByte = 0;
```

```
        ret = RecvPack(NULL, bData, 1000); //异步的话此超时要延长，具体时间根据语音的长短
        if(ret == XG_ERR_SUCCESS)
        {
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        return XG_ERR_SUCCESS;
                }
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.32 修改设备时间

- **[命令码]**

  ```
  #define XG_CMD_SET_DATETIME          0x36 //修改设备时间
  ```

- **[功能]**

  设置修改设备的系统时间，锁、门禁等产品适用

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x36	XG_CMD_SET_DATETIME
bEncode	0x00	数据编码，为0
bDataLen	0x06	
bData	buf[0] = 年-2000 buf[1] = 月 buf[2] = 日 buf[3] = 时 buf[4] = 分 buf[5] = 秒	
wCheckSum		校验和

- **[实例]**

```
/*
功能:修改设备时间

返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetDevDateTime(UINT16 year, UINT8 mon, UINT8 day, UINT8 hour, UINT8 min, UINT8 sec)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = year - 2000;
	bData[1] = mon;
	bData[2] = day;
	bData[3] = hour;
	bData[4] = min;
	bData[5] = sec;
	SendPack(XG_CMD_SET_DATETIME, bData, 6);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.33 无超时用户验证

- **[命令码]**

```
#define XG_CMD_IDENTIFY_FREE          0x18 //无超时1:1认证或:N识别，可发0x19指令中断
```

- **[功能]**

  发送此指令会一直等待手指放入直到验证成功或验证失败后才会退出此指令，或可以通过发送0x19指令中断退出此指令，此指令的参数同0x17指令。

  如指定用户ID则为1:1验证模式。

  如指定用户ID为0则为1：N识别验证模式。

  注意：

  此指令正常验证会返回多个返回包，只有接收到XG_ERR_SUCCESS或XG_ERR_FAIL或XG_ERR_BREAK_OFF返回值时才结束验证过程。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x18	XG_CMD_IDENTIFY_FREE
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
	bData[4] = 分组组号
	bData[5] = 连续验证几个ID
		ID = 0为1：N验证方式
		ID > 0为1：1验证方式

wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x18	XG_CMD_IDENTIFY_FREE
bEncode	0x00	数据编码，为0
bDataLen		
bData	验证成功	识别的用户ID = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[4]	

	验证失败 bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID / XG_ERR_TIME_OUT	XG_ERR_TIME_OUT：等待手指输入超时
	XG_INPUT_FINGER:	提示用户放入手指
	XG_RELEASE_FINGER	提示用户拿开手指
wCheckSum		校验和

- **[实例]**

```
/*
功能:发送验证指令并获取返回的验证结果
pUserID:返回验证成功用户ID的指针
返回值:
XG_ERR_SUCCESS:验证成功，验证成功的用户ID通过pUserID返回
XG_ERR_NO_VEIN:没有成功采集指静脉，可能手指没放好
*/

UINT8 FreeVerifyUser(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT32 CardNo = 0;

	if(pUserID)
	{
		//如果pUserID为非则:1验证
		bData[0] = (*pUserID)&0xff;
		bData[1] = ((*pUserID)>>8)&0xff;
		bData[2] = 0;
		bData[3] = 0;
		bData[4] = 0; //0为不分组
		bData[5] = 0; //0为单个用户验证,>0为UserID开始后的连续几个用户验证
		bData[6] = CardNo&0xff; //还可以按卡号进行:1验证
		bData[7] = (CardNo>>8)&0xff;
		bData[8] = (CardNo>>16)&0xff;
```

```
        bData[9] = (CardNo>>24)&0xff;
}

SendPack(XG_CMD_IDENTIFY_FREE, bData, 10);
gUartByte = 0;
for(;;)
{
        ret = RecvPack(NULL, bData, 0); //此处是无限超时，一直等待手指放入返回验证结果
        if(ret != XG_ERR_SUCCESS)
        {
                PLAY_SOUND(VOICE_IDENT_FAIL);
                break;
        }
        if(bData[0] == XG_ERR_SUCCESS)
        {
                PLAY_SOUND(VOICE_IDENT_OK);
                if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
                break;
        } else if(bData[0] == XG_INPUT_FINGER) {

        } else if(bData[0] == XG_RELEASE_FINGER) {
                //指静脉采集完成，可以拿开手指了
                PLAY_SOUND(VOICE_KEY);
        } else {
                //错误处理
                ret = bData[1];
                if(ret == XG_ERR_NO_VEIN)
                {
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else {
                        PLAY_SOUND(VOICE_IDENT_FAIL);
                }
                break;
        }
}
return ret;
```

```
}
```

2.2.34 中断或取消登记和验证指令

- **[命令码]**

```
#define XG_CMD_CANCEL                    0x19 //中断当前执行的指令
```

- **[功能]**

  取消当前执行的指令，如登记、验证和无超时验证指令。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x19	XG_CMD_CANCEL
bEncode	0x00	数据编码，为0
bDataLen	0x00	
bData		
wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0xXX	当前中断的指令或XG_CMD_CANCEL
bEncode	0x00	数据编码，为0
bDataLen		
bData	指令中断	
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_BREAK_OFF	
	指令执行中断	
wCheckSum		校验和

- **[实例]**

```
void BreakOffCmd()
```

```
{
        SendPack(XG_CMD_CANCEL, NULL, 0);
}
```

2.2.35 验证成功后串口输出验证信息包

- **[命令码]**

  ```
  #define XG_CMD_REPORT                0x4E //串口输出验证信息包
  ```

- **[功能]**

  此指令只有门禁、锁控制器有此指令，需设置为串口输出，验证成功后会通过串口发送此指令包，此指令由控制器发出，无需应答。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x4E	XG_CMD_REPORT
bEncode	0x00	数据编码，为0
bDataLen	16	
bData	bData[0] = 1; bData[1-4] = ID;	
wCheckSum		校验和

2.2.36 读取刷卡卡号

- **[命令码]**

  ```
  #define XG_CMD_READ_CARDNO           0x53 //刷卡并读取当前卡的卡号
  ```

- **[功能]**

  请求刷卡并读取当前卡的卡号。

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x53	XG_CMD_READ_CARDNO
bEncode	0x00	数据编码，为0

bDataLen	0	
bData		
wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x53	XG_CMD_READ_CARDNO
bEncode	0x00	数据编码，为0
bDataLen		
bData	获取成功	CardNo = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	bData[0] = XG_ERR_SUCCESS 卡号:bData[1]-bData[4] bData[5]-bData[15]为物理卡号原始数据	
	获取失败	刷卡超时
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_TIME_OUT	
wCheckSum		校验和

- **[实例]**

```
UINT8 GetCardNo(int* pCardNo)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(XG_CMD_READ_CARDNO, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 6000); //刷卡等待时间为5秒
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pCardNo) * pCardNo = bData[1] + (bData[2]<<8) + (bData[3]<<16) +
```

```
(bData[4]<<24);
                        return XG_ERR_SUCCESS;
                }
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.37 获取用户名

- **[命令码]**

```
#define XG_CMD_GET_USERNAME                0x3C //获取用户名
```

- **[功能]**

  获取用户名

- **[指令实例]**

  发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x3C	XG_CMD_GET_USERNAME
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
		用户ID

wCheckSum		校验和

  返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x3C	XG_CMD_GET_USERNAME
bEncode	0x00	数据编码，为0
bDataLen	16	

bData	成功	
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[14] 用户名	
	失败	
	bData[0] = XG_ERR_FAIL	
wCheckSum		校验和

- **[实例]**

```
/*
功能:获取用户名
pName:返回用户名
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:错误
*/
UINT8 GetUserName(UINT8* pName)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(0x3C, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pName) memcpy(pName, &bData[1], 14);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.38 设置用户名

- [命令码]

```
#define XG_CMD_SET_USERNAME              0x3D //设置用户名
```

- [功能]

  设置用户名

- [指令实例]

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x3D	XG_CMD_SET_USERNAME
bEncode	0x00	数据编码，为0
bDataLen	16	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2-15] = 用户名	

wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x3D	XG_CMD_SET_USERNAME
bEncode	0x00	数据编码，为0
bDataLen		
bData	成功	
	bData[0] = XG_ERR_SUCCESS	
	失败	
	bData[0] = XG_ERR_FAIL	
wCheckSum		校验和

- [实例]

```
/*
```

```
功能:设置用户名
UserId:用户ID
pName:用户名
返回值:
XG_ERR_SUCCESS:成功
XG_ERR_FAIL:失败
*/

UINT8 SetUserName (UINT16 UserId, UINT8* pName)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = UserId&0xff;
	bData[1] = (UserId>>8)&0xff;
	memcpy(&bData[2], pName, 14);
	SendPack(0x3D, bData, 16);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.39 滑动登记

- **[命令码]**

```
#define XG_CMD_ENROLL_HD                0x5A //滑动登录
```

- **[功能]**

可以在手指不离开的情况下转动或滑动手指连续采集，也可以多次重复放入手指完成采集登记

注意：

此指令会连续返回多个回包，只有接收到XG_ERR_SUCCESS或XG_ERR_FAIL返回值时才结束登记过程。

- **[指令实例]**

发送包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x5A	XG_CMD_ENROLL_HD
bEncode	0x00	数据编码，为0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;	

wCheckSum		校验和

返回包：

字段	数据	备注
wPrefix	0xAABB	包标识
bAddress	0x00	目标设备地址
bCmd	0x5A	XG_CMD_ENROLL_HD
bEncode	0x00	数据编码，为0
bDataLen		
bData	登录成功	
	bData[0] = XG_ERR_SUCCESS	
	登录失败	登录失败： XG_ERR_INVALID_ID：非法ID XG_ERR_NOT_ENOUGH：已无登录空
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/	

	XG_ERR_NOT_ENOUGH/ XG_ERR_TIME_OUT/ XG_ERR_DUPLICATION_ID bData[3], bData[0]为触摸状态 =0为无触摸，1为有触摸	间 XG_ERR_TIME_OUT：等待手指输入超时 XG_ERR_DUPLICATION_ID：有相似用户已经登录
	XG_INPUT_FINGER:	提示用户放入手指
	XG_RELEASE_FINGER	提示用户滑动或拿开手指
wCheckSum		校验和

- **[实例]**

```
/*
功能:发送登记指令增加指静脉用户
UserID:指定登记用户的ID
返回值:
XG_ERR_SUCCESS:登记成功
*/

UINT8 EnrollUser_HD(UINT16 UserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT8 bTouch1 = 0xff, bTouch2 = 0xff;

	bData[0] = UserID&0xff;
	bData[1] = (UserID>>8)&0xff;
	bData[2] = 0;
	bData[3] = 0;
	SendPack(0x5A, bData, 4); //滑动登记指令
	gUartByte = 0;
	for(;;)
	{
		ret = RecvPack(NULL, bData, 6000); //手指检测超时是5秒
		if(ret != XG_ERR_SUCCESS)
		{
			PLAY_SOUND(VOICE_REG_FAIL);
```

```
                break;
            }
            if(bData[0] == XG_ERR_SUCCESS)
            {
                PLAY_SOUND(VOICE_REG_OK);
                break;
            } else if(bData[0] == XG_INPUT_FINGER) {
                //bData[3]和bData[4]是两个触摸状态，触摸状态改变提示信息
                if(bTouch1 != bData[3] || bTouch2 != bData[4])
                {
                    if(bData[3] == 0 && bData[4] == 0) PLAY_SOUND(VOICE_INPUT);//Message("请放入手指");
                    if(bData[3] == 1 && bData[4] == 0) PLAY_SOUND(VOICE_BEEP3);//Message("触摸1未贴紧");
                    if(bData[3] == 0 && bData[4] == 1) PLAY_SOUND(VOICE_BEEP4);//Message("触摸2未贴紧");
                    if(bData[3] == 1 && bData[4] == 1) PLAY_SOUND(VOICE_BEEP1);//Message("请滑动手指...");
                    bTouch1 = bData[3];
                    bTouch2 = bData[4];
                }
            } else if(bData[0] == XG_RELEASE_FINGER) {
                PLAY_SOUND(VOICE_KEY);//Message("请滑动手指...");
            } else {
                //错误处理
                ret = bData[1];
                if(ret == XG_ERR_INVALID_ID)
                {
                    //非法ID
                    PLAY_SOUND(VOICE_PINPUT_ID); //ID错误
                    break;
                } else if(ret == XG_ERR_NOT_ENOUGH) {
                    //此用户已登记
                    PLAY_SOUND(VOICE_USER_FULL);
                    break;
                } else if(ret == XG_ERR_TIME_OUT) {
```

```
                        //手指检测超时
                        PLAY_SOUND(VOICE_REG_FAIL);
                        break;
                    } else if(ret == XG_ERR_DUPLICATION_ID) {
                        //重复登记
                        PLAY_SOUND(VOICE_OVER_REG);
                        break;
                    } else if(ret == XG_ERR_NO_SAME_FINGER) {
                        //不是同一根手指,如果没有采集到最大次数可以继续采集
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                    } else if(ret == XG_ERR_NO_VEIN) {
                        //没有检测到指静脉,如果没有采集到最大次数可以继续采集
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                    } else {
                        PLAY_SOUND(VOICE_REG_FAIL);
                        break;
                    }
                }
            }
            return ret;
        }
```

四、附录一：错误代码列表

```
/*****************************错误代码*****************************************/
#define XG_ERR_SUCCESS                  0x00 //操作成功
#define XG_ERR_FAIL                     0x01 //操作失败
#define XG_ERR_COM                      0x02 //通讯错误
#define XG_ERR_DATA                     0x03 //数据校验错误
#define XG_ERR_INVALID_PWD              0x04 //密码错误
#define XG_ERR_INVALID_PARAM            0x05 //参数错误
#define XG_ERR_INVALID_ID               0x06 //ID错误
#define XG_ERR_EMPTY_ID                 0x07 //指定ID为空（无登录数据）
#define XG_ERR_NOT_ENOUGH               0x08 //无足够登录空间
#define XG_ERR_NO_SAME_FINGER           0x09 //不是同一根手指
#define XG_ERR_DUPLICATION_ID           0x0A //有相同登录ID
#define XG_ERR_TIME_OUT                 0x0B //等待手指输入超时
#define XG_ERR_VERIFY                   0x0C //认证失败
#define XG_ERR_NO_NULL_ID               0x0D //已无空ID
#define XG_ERR_BREAK_OFF                0x0E //操作中断
#define XG_ERR_NO_CONNECT               0x0F //未连接
#define XG_ERR_NO_SUPPORT               0x10 //不支持此操作
#define XG_ERR_NO_VEIN                  0x11 //无静脉数据
#define XG_ERR_MEMORY                   0x12 //内存不足
#define XG_ERR_NO_DEV                   0x13 //设备不存在
#define XG_ERR_ADDRESS                  0x14 //设备地址错误
#define XG_ERR_NO_FILE                  0x15 //文件不存在
#define XG_ERR_VER                      0x16 //版本错误
#define XG_ERR_BREAK_CARD               0x17 //刷卡中断

/*****************************状态代码*****************************************/
#define XG_INPUT_FINGER                 0x20 //请求放入手指
#define XG_RELEASE_FINGER               0x21 //请求拿开手指
```

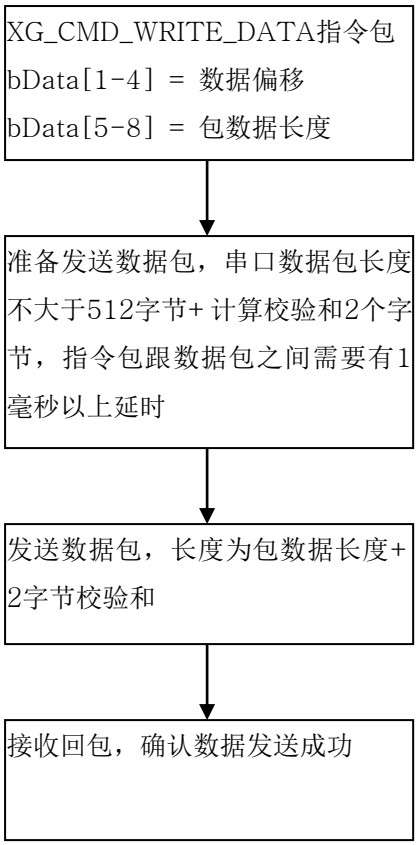
五、附录二：语音播放列表

```
VOICE_REG_OK = 0,//登记成功
VOICE_USER_FULL = 1,//记录已满
VOICE_REG_FAIL = 2,//登记失败
VOICE_OVER_REG = 3, //登记重复
VOICE_ADMIN_AUTH_OK=8, //管理员验证成功
VOICE_ADMIN_AUTH_FAIL=9, //管理员验证失败
VOICE_ERROR_CARD=10, //卡号错误
VOICE_PWD_ERROR=13, //密码错误
VOICE_PWD_REG_OK = 14,//密码登记成功
VOICE_PWD_REG_FAIL = 15,//密码登记失败
VOICE_ENROLL_ADMIN = 18,//请登记管理员
VOICE_PINPUT_PWD = 19,//请输入密码
VOICE_PINPUT_ID = 20,//请输入用户ID
VOICE_ADMIN_AUTH = 21,//请验证管理员
VOICE_INPUT_FINGER_AGAIN = 23,//请再放一次
VOICE_RETRY = 24,//请再试一次
VOICE_INPUT_AGAIN = 25,//请再输一次
VOICE_RIGHT_INPUT = 26, //请正确放置手指
VOICE_INPUT = 27,//请自然轻放手指
VOICE_DEL_OK = 28,//删除成功
VOICE_DEL_FAIL = 29,//删除失败
VOICE_THANKS = 30, //谢谢
VOICE_DELING = 31,//正在删除
VOICE_IDENT_FAIL = 32,//验证失败
VOICE_IDENT_OK = 33, //验证成功
VOICE_UNLOCK = 34, //已开锁
VOICE_BEEP1 = 35,
VOICE_KEY = 36,
VOICE_BEEP3 = 37,
VOICE_BEEP4 = 38,
VOICE_ALARM = 39, //39 报警音
VOICE_OK_KEY_CONFIRM = 40, //按OK键确认
VOICE_POWER_LOW = 41, //电量不足请更换电池
```

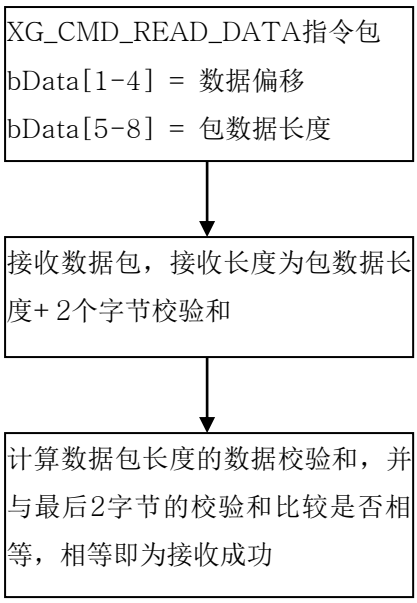
```
VOICE_ADMIN = 42,//管理员
VOICE_PWD = 43,//密码
VOICE_CARDNO = 44,//刷卡
VOICE_INPUT_CARD = 45, //请刷卡
VOICE_LOCK = 46, //已关锁
VOICE_PLS_INPUT_ADMIN_PWD = 47, //请输入管理员密码
VOICE_ADMIN_FULL = 49, //管理员已满
VOICE_OK_KEY_DEL_CONFIRM = 50, //请按OK键确认删除
VOICE_HELP = 51,
VOICE_HELP1 = 52,
```

六、附录三：功能实现示例

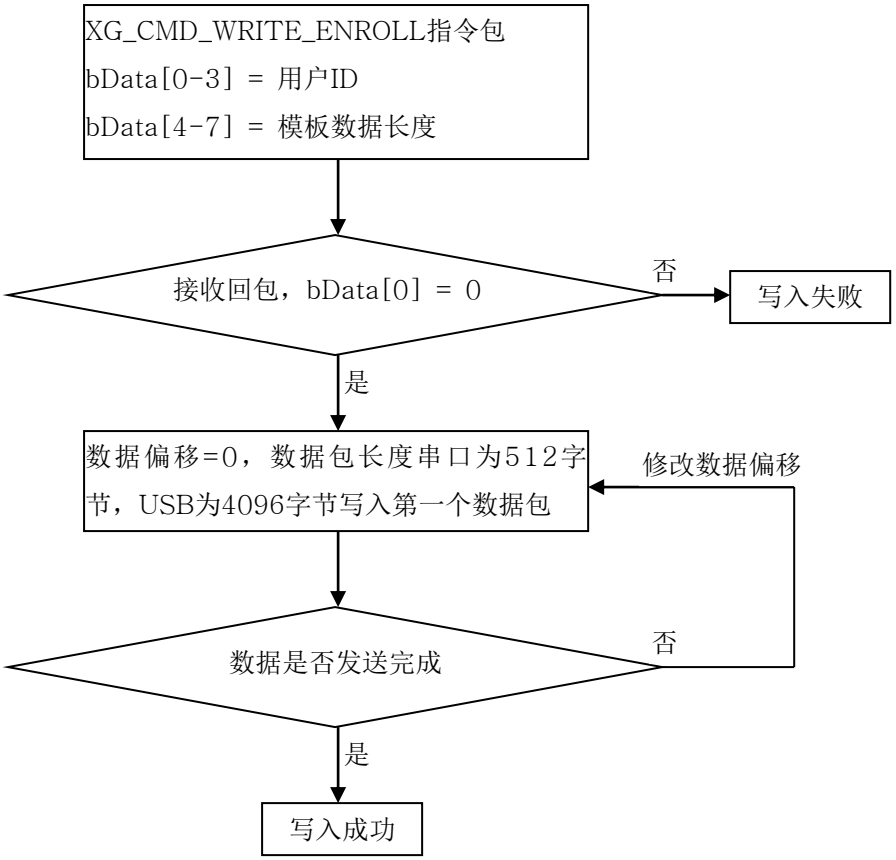
1、数据包的发送流程



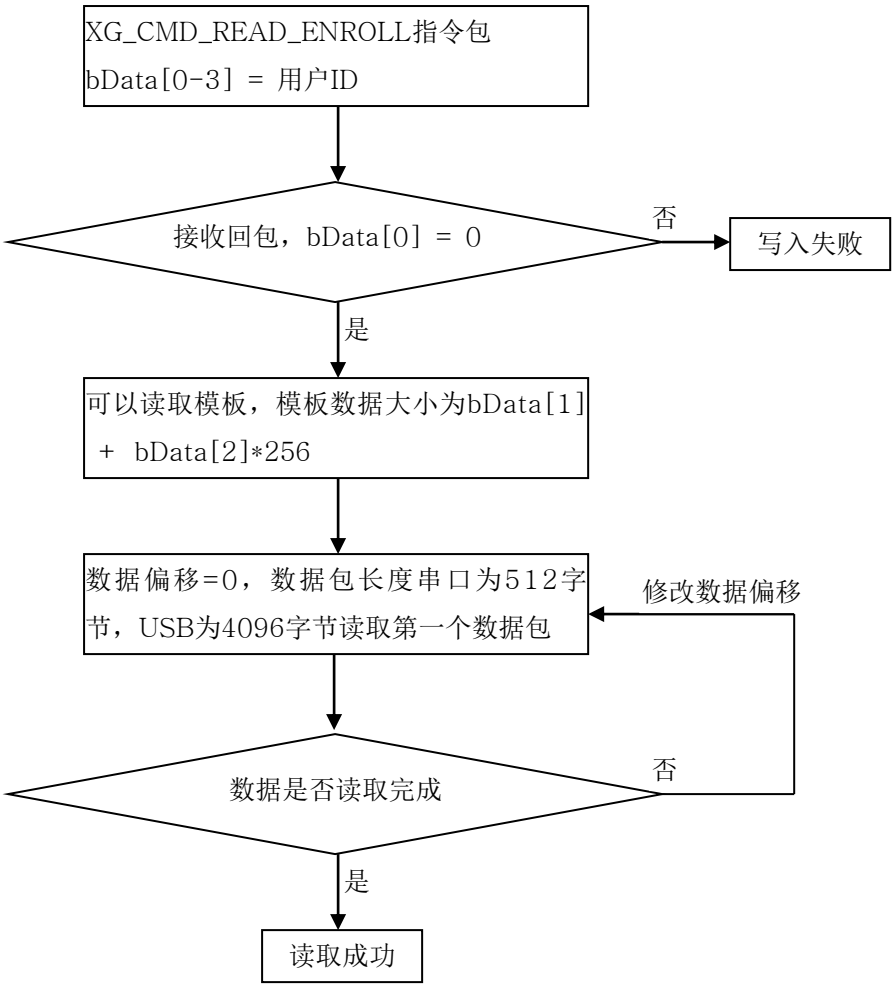
2、 数据包的接收流程


XG_CMD_READ_DATA指令包
bData[1-4] = 数据偏移
bData[5-8] = 包数据长度
接收数据包，接收长度为包数据长度+2个字节校验和
计算数据包长度的数据校验和，并与最后2字节的校验和比较是否相等，相等即为接收成功

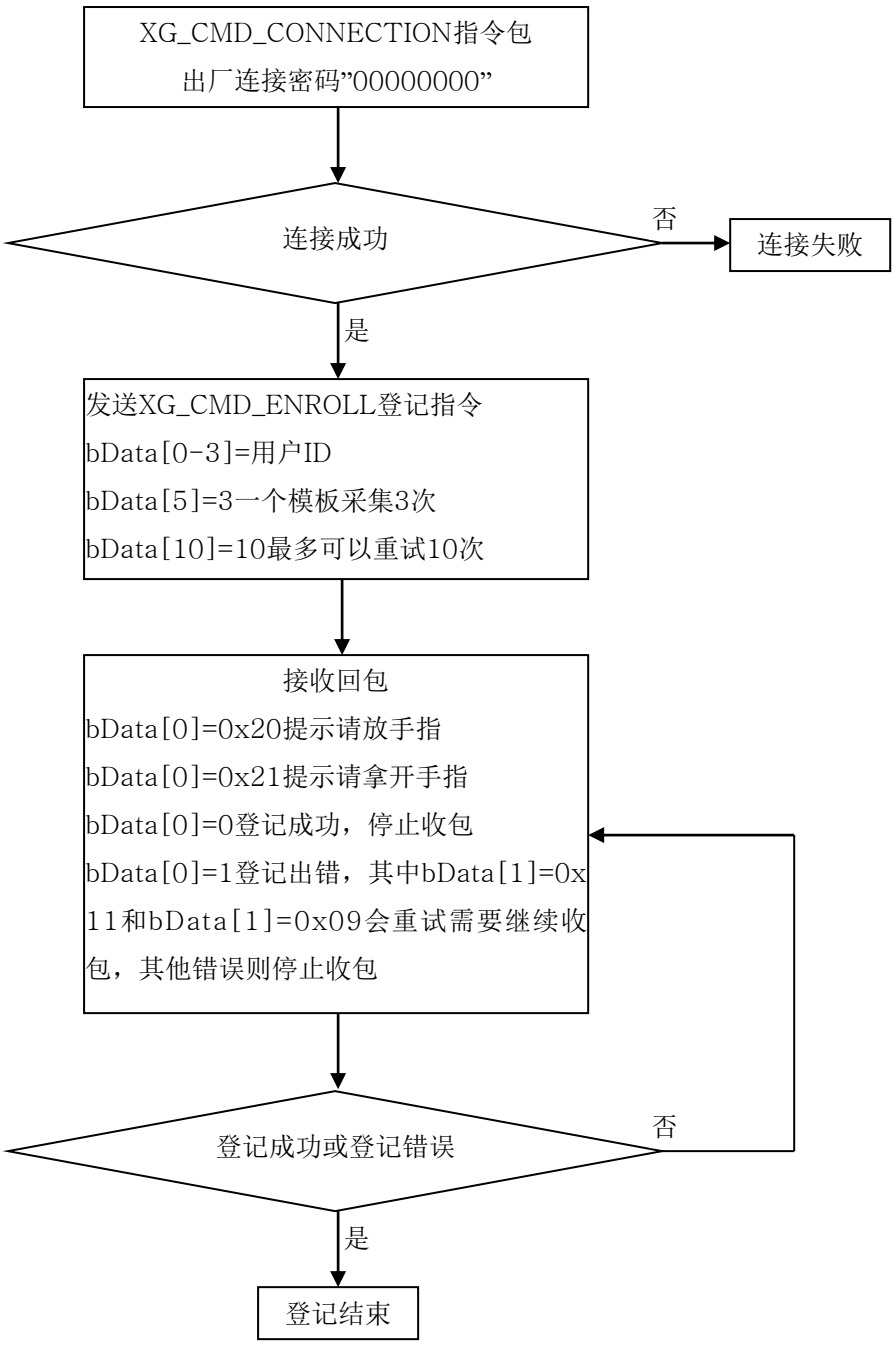
3、模板写入流程


XG_CMD_WRITE_ENROLL指令包
bData[0-3] = 用户ID
bData[4-7] = 模板数据长度
接收回包，bData[0] = 0
否
写入失败
是
数据偏移=0，数据包长度串口为512字节，USB为4096字节写入第一个数据包
修改数据偏移
数据是否发送完成
否
是
写入成功

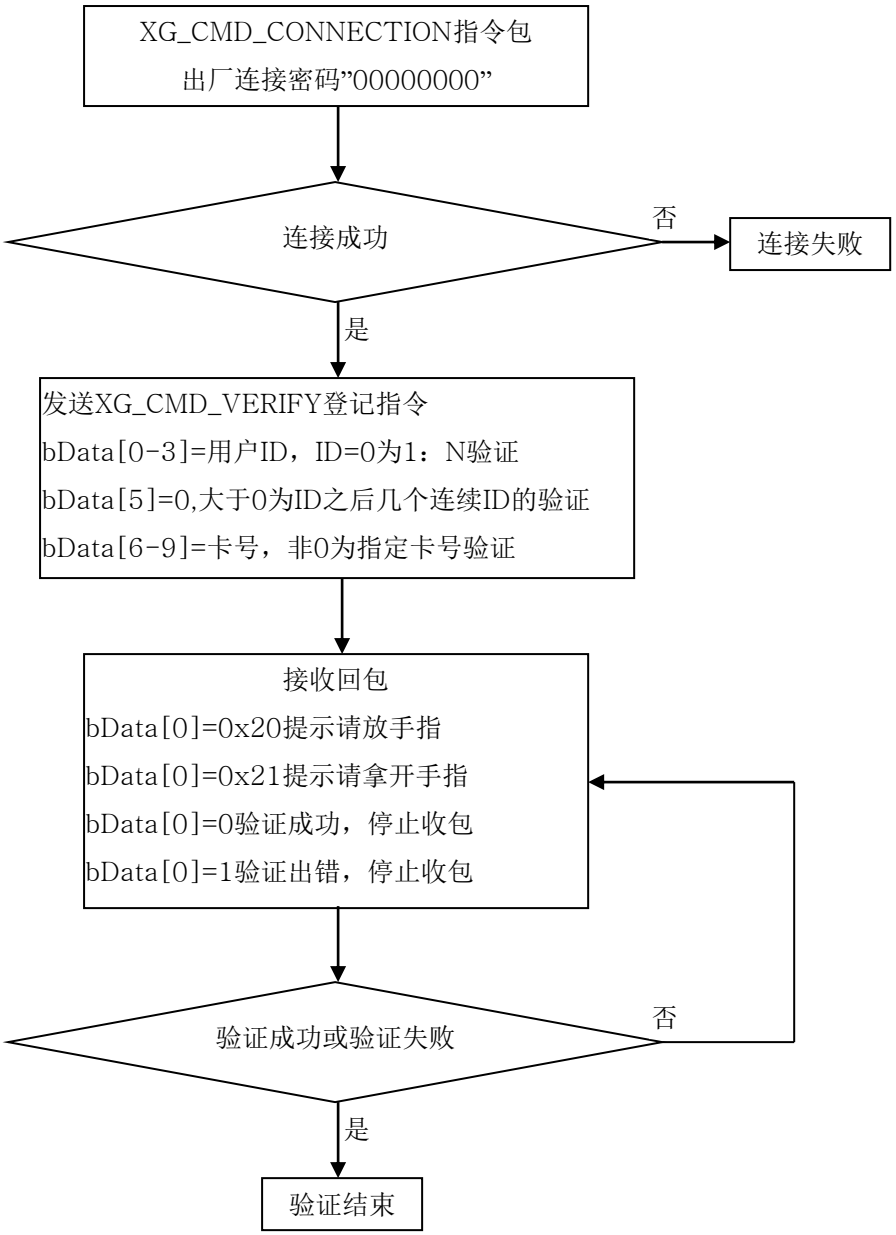
4、模板读取流程


XG_CMD_READ_ENROLL指令包
bData[0-3] = 用户ID
接收回包，bData[0] = 0
否
写入失败
是
可以读取模板，模板数据大小为bData[1] + bData[2]*256
数据偏移=0，数据包长度串口为512字节，USB为4096字节读取第一个数据包
修改数据偏移
数据是否读取完成
否
是
读取成功

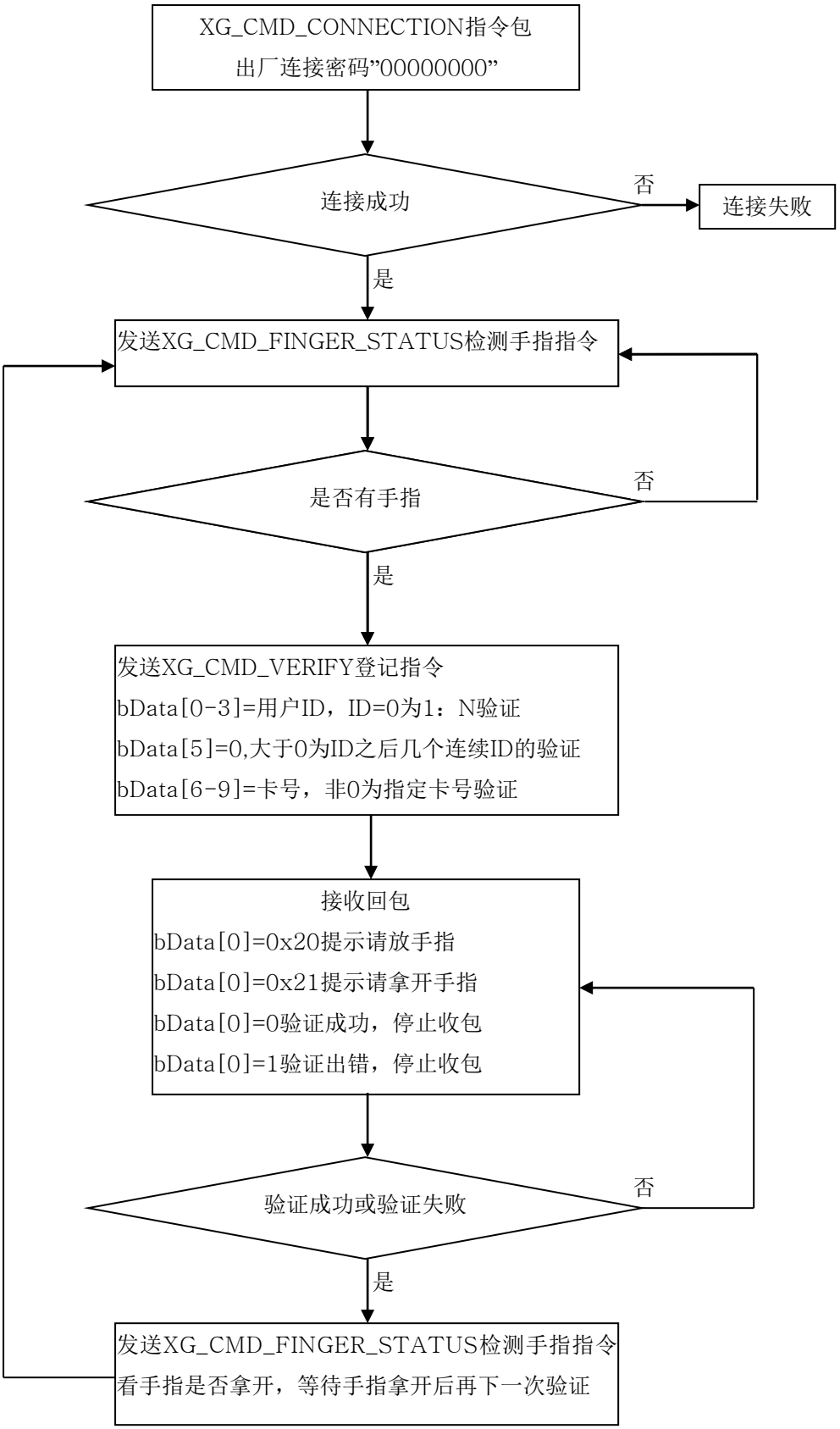
5、登记流程


XG_CMD_CONNECTION指令包
出厂连接密码”00000000”
连接成功
否
连接失败
是
发送XG_CMD_ENROLL登记指令
bData[0-3]=用户ID
bData[5]=3一个模板采集3次
bData[10]=10最多可以重试10次
接收回包
bData[0]=0x20提示请放手指
bData[0]=0x21提示请拿开手指
bData[0]=0登记成功，停止收包
bData[0]=1登记出错，其中bData[1]=0x11和bData[1]=0x09会重试需要继续收包，其他错误则停止收包
登记成功或登记错误
否
是
登记结束

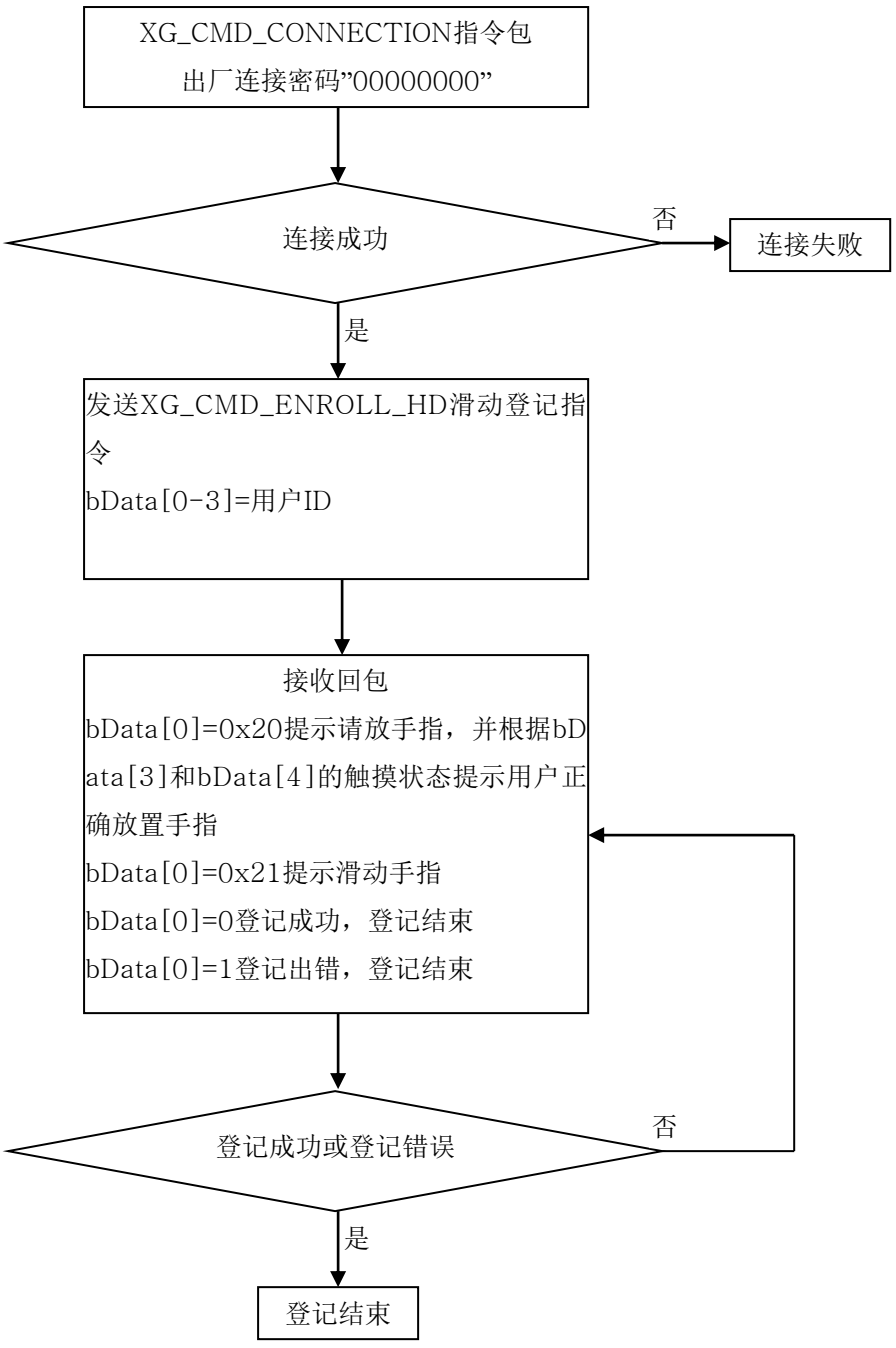
6、验证流程


XG_CMD_CONNECTION指令包
出厂连接密码”00000000”
连接成功
否
连接失败
是
发送XG_CMD_VERIFY登记指令
bData[0-3]=用户ID，ID=0为1：N验证
bData[5]=0,大于0为ID之后几个连续ID的验证
bData[6-9]=卡号，非0为指定卡号验证
接收回包
bData[0]=0x20提示请放手指
bData[0]=0x21提示请拿开手指
bData[0]=0验证成功，停止收包
bData[0]=1验证出错，停止收包
验证成功或验证失败
否
是
验证结束

7、在线1：N验证流程


XG_CMD_CONNECTION指令包
出厂连接密码”00000000”
连接成功
否
连接失败
是
发送XG_CMD_FINGER_STATUS检测手指指令
是否有手指
否
是
发送XG_CMD_VERIFY登记指令
bData[0-3]=用户ID，ID=0为1：N验证
bData[5]=0,大于0为ID之后几个连续ID的验证
bData[6-9]=卡号，非0为指定卡号验证
接收回包
bData[0]=0x20提示请放手指
bData[0]=0x21提示请拿开手指
bData[0]=0验证成功，停止收包
bData[0]=1验证出错，停止收包
验证成功或验证失败
否
是
发送XG_CMD_FINGER_STATUS检测手指指令
看手指是否拿开，等待手指拿开后再下一次验证

8、滑动登记流程


XG_CMD_CONNECTION指令包
出厂连接密码”00000000”
连接成功
否
连接失败
是
发送XG_CMD_ENROLL_HD滑动登记指令
bData[0-3]=用户ID
接收回包
bData[0]=0x20提示请放手指，并根据bData[3]和bData[4]的触摸状态提示用户正确放置手指
bData[0]=0x21提示滑动手指
bData[0]=0登记成功，登记结束
bData[0]=1登记出错，登记结束
登记成功或登记错误
否
是
登记结束