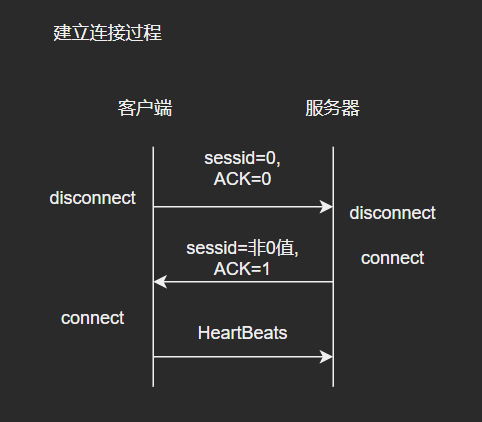
# 协议文档定义说明

上位机客户端和下位机服务器基于串口和socket实现一套支持接口切换(初步不支持自动切换，根据配置选择使用的接口)的通讯协议接口，在未建立连接时，两边session\_id均为0， 高1位表示是否位ACK包，整个协议的工作流大概如下所示。

建立连接过程如下所示，首先客户端发送同步包(session=0, ack=0)，服务器回复应答包(session=!0, ack=1)，至此连接建立，开始由客户端维持心跳包，当超时一段时间后(暂定义3s)，表示连接断开，重新进行连接.



# 协议格式

## 协议结构

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 功能 | 起始位 | ACK | Session\_Id | Sequence | 长度 | 数据区 | CRC16 |
| 长度(B) | 2 | 1bit | 7bit | 1 | 1 | n | 2 |
| 说明 | 0x5AA5 | 是否ACK | 1~0x7f, 0x00表示未连接 | 包序列号，用于重发机制 | 0~255单个包最大255 | 实际数据 | CRC校验位 |

ACK:分为通讯的数据包和应答的ACK包，主要用于建立连接和重发机制。

Session\_ID:默认session\_id为0x00, 表示未连接，其它session\_id表示已经连接

Sequence:每个包发送都递增生成序列号，用于重发机制等待ack

长度:允许0x00~0x255的数据长度

数据区:包含数据的信息, 可以支持加密和非加密模式，长度不超过255

CRC16:支持ACK-数据区末位的crc校验

## CRC算法

查表法, 服务器和客户端使用相同公式处理(x^16 + x^15 + x^2 + 1).

static uint16\_t const crc16\_table[256] = {

    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,

    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,

    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,

    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,

    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,

    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,

    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,

    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,

    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,

    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,

    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,

    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,

    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,

    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,

    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,

    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,

    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,

    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,

    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,

    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,

    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,

    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,

    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,

    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,

    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,

    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,

    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,

    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,

    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,

    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,

    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,

    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040

};

static inline uint16\_t crc16\_byte(uint16\_t crc, const uint8\_t data)

{

    return (crc >> 8) ^ crc16\_table[(crc ^ data) & 0xff];

}

uint16\_t crc16(uint16\_t crc, uint8\_t const \*buffer, uint16\_t len)

{

    while (len--)

        crc = crc16\_byte(crc, \*buffer++);

    return crc;

}