技术文档 - Feature #400

无人船远程控制与数据交换协议

2019-04-19 13:39 - 坚 黄

状态:	New	**开始日期:**	2019-04-19
优先级:	中	**计划完成日期:**	
指派给:		**% 完成:**	0%
类别:		**预期时间:**	0.00 小时
目标版本:			

描述

1. 应用层网络协议
   无人船对外提供两种协议的控制协议：
   1. MQTT控制协议
      无人船上运行了一个Broker，基站等终端通过网络连接上无人船的Broker后，使用MQTT控制协议进行数据交换。
   2. TCP控制协议
      无人船作为服务端，基站等终端作为客户端，基站使用TCP协议连接上无人船后开始使用控制协议进行数据交换。
      以上两种协议，MQTT的消息名与命令ID一一对应，MQTT的负载部分的内容与对应的命令的参数内容一致。比如：
      MQTT消息 /ctrl 与命令 0x0102 功能一样，消息的负载内容为9个字节，命令的参数也是9个字节，也就是说，通过MQTT发送 /ctrl，包括9个字节的负载内容，与通过TCP发送命令
      0x0102,参数的内容与MQTT所发送的负载内容一样，那么这两种方式是完全等价的。
2. 设备发现
   主控会每隔2秒同时广播和组播自己的设备信息，设备信息为字符串类型，格式如下：
   ip=%s;id=%s;mqtt=%d;ver=%s
   ip: 表示主控的IP地址
   id: 表示船的ID，也作为MQTT客户端的ID
   mqtt: 表示MQTT服务的端口号
   ver: 表示当前主控程序的版本号
   广播的地址为：255.255.255.255
   广播的端口号为：10415
   组播的地址为：234.56.78.90
   组播的端口号为：10415
   无人船的某些专网设备，可能既不支持组播也不支持广播，这种情况无法使用设备发现功能。设备发现功能只能用于局域网，因此，4G远程控制也无法使用设备发现功能。
   无人船的设备IP默认情况下是192.168.1.230，mqtt服务的端口号为1883，TCP控制协议的服务端口号为3000，通常情况下，客户端只需要直接连接对应的IP和端口即可开始进行通信。
3. 命令格式规定
   命令的格式规定为：| 命令（2B）| 扩展（1B）| [序号（2B）] | [参数（>=0B）] | 校验（1B） |
   命令：取值范围 0 ～ 65535，大端模式
   扩展：0表示不需要接收方发送确认命令，1表示需要接收方发送确认命令
   序号：取值范围 0 ～ 65535，大端模式，发送方每发送一条需要确认的数据，序号相应加1。
   参数：格式和长度由命令决定
   校验：从“ 命令 ”～“ 参数 ”所有字节的CRC8值
   序号项不一定存在，取决于扩展项的值。当扩展项的值为0值时，命令中不能带序号；当扩展项的值为1时，命令中必须带序号。除非特别说明，每个命令都可以带序号也可以不带序号，取决与发送方，如果发送方觉得这个命令很重要，需要接收方确认，则需要带上序号，否则不用带序号以提高数据发送效率。
   如果多条命令之间的顺序不重要，那么发送端可以在接收到上一条命令的确认前，继续发送下一条命令。如果命令之间有顺序要求（如GPS数据和测深数据），那么发送端只有在接收到上一条命令的确认后才能发送下一条命令。
   如没有特别说明，协议内传输的数据统一使用大端模式。
4. 具体命令定义
   1. 命令-0x0000（Client <--> USV）
      确认命令已收到，扩展值必须为0，参数格式为：|所确认的命令的序号：（2B）|
      例（CRC8未计算）：

      A<------------------------------>B
      [0xXX,0xXX,0x01,0x12,0x34,CRC8] ->
                                  <- [0x00,0x00,0x00,0x12,0x34,CRC8]
                                  <- [0xXX,0xXX,0x01,0x11,0x22,CRC8]
      [0x00,0x00,0x00,0x11,0x22,CRC8] ->

第1行：A向B发送某命令“ 0xXXXX ”，需要确认，序号为0x1234，不带参数
第2行：B向A发送确认，确认收到了序号为0x1234的命令
第3行：B向A发送某命令“ 0xXXXX ”，需要确认，序号为0x1122，不带参数
第4行：A向B发送确认，确认收到了序号为0x1122的命令
发送方在一定时间（比如2秒）内没有收到接收方的确认数据时，需要对数据进行重发，以保证数据最终能成功发送给对方。

2. 命令-0x0001（Client <--> USV）
   心跳PING命令，没有参数。当通信的一端有一阵时间（2秒）没有发送或者接收到数据时，可以通过发送PING命令来避免超时
3. 命令-0x0002 （Client <--> USV）
   心跳的响应命令，没有参数。当通信的一端接收到PING命令时，需要回复这个命令。通信的一端接收到这个命令时，直接忽略，不做处理。
4. 命令-0x00FF（Web <-- Cloud）
   Web客户端与云服务专用，无人船上线与离线命令， 参数为1个字节，0表示离线，1表示上线。
5. 命令-0x0100（Client --> USV），无对应的mqtt消息
   路径点发送命令，参数格式为：|路径点序号（2B）|标记（1B）|WGS84纬度（8B）|WGS84经度（8B）|
   路径点序号：16位整数，从0开始计算
   标记：第0位表示是否是开始点，第1位表示是否是结束点，第2位表示是否为测绘点
   WGS84纬度：双精度浮点数
   WGS84经度：双精度浮点数
6. 命令-0x0102（Client --> USV），对应的mqtt消息为 /ctrl
   控制船的速度和方向，通用方式，适用于带舵的船和差速控制的船，要求控制频率不小于2Hz,建议频率10Hz。参数格式为：|速度（4字节）|方向（4字节）|优先级（1字节）|
   速度：单精度浮点数，取值范围为-1 ～ 1，-1表示全速后退，1表示全速前进。
   方向：单精度浮点数，取值范围为-1 ～ 1，-1表示满舵左转，1表示满舵右转。
   优先级：0~100，遥控器具有最高的优先级100，建议其他终端不大于50。
7. 命令-0x0103（Client --> USV），对应的mqtt消息为 /mode/set
   改变船的控制模式，参数格式为：|控制模式（1B）
   控制模式：0表示自动，1表示手动
8. 命令-0x0104（Client --> USV），对应的mqtt消息为 /nav/start
   控制无人船，开始任务，参数可选，不带参数时默认为0，格式为：
   [|开始点（2B）|]
   开始点：16位整数，表示这个点和这个点之前的任务已经完成，无人船开始往下一个点航行。
9. 命令-0x0105（Client --> USV），对应的mqtt消息为 /nav/pause
   控制无人船，暂停任务，暂停后收到/nav/start命令，会继续从当前位置开始航行，无参数。
10. 命令-0x0106（Client --> USV），对应的mqtt消息为 /nav/stop
    控制无人船，停止任务，停止后收到/nav/start命令，会从第一个点开始航行，无参数。
11. 命令-0x0107（Client --> USV），无对应的mqtt消息
    往测深仪透传数据，参数部分就是要发送的数据
12. 命令-0x0108（Client --> USV），无对应的mqtt消息
    往RTK透传数据，参数部分就是要发送的数据
13. 命令-0x010A（Client <-- USV），无对应的mqtt消息
    测深仪数据透传，参数部分为测深仪输出的原始数据
14. 命令-0x010B（Client <-- USV），无对应的mqtt消息
    RTK数据透传，参数部分为RTK输出的原始数据
15. 命令-0x010C（Client <-- USV），对应的mqtt消息为 /status
    无人船状态反馈数据，发送频率为1Hz，参数格式为：|前往点（2B）|控制模式（1B）|任务类型（1B）|任务状态（1B）|工作模式（1B）|
    前往点序号：16位整数，表示无人船正前往这个点
    控制模式：0表示自动，1表示手动
    任务类型：扩展用，忽略
    任务状态：0表示停止，1表示暂停，2表示正在运行，3表示空闲
    工作模式：与0x04进行与操作，不为0表示正在测绘（比如第3个点是测绘点，那么往第4个点的过程中，工作模式的值就为4）
16. 命令-0x010D（Client <-- USV），对应的mqtt消息为 /gps
    无人船的当前位置，发送频率由传感器设备的频率决定，一般不低于5Hz，参数格式为：|WGS84纬度（8B）|WGS84经度（8B）|
    WGS84纬度：双精度浮点数
    WGS84纬度：双精度浮点数
17. 命令-0x010E（Client <-- USV），对应的mqtt消息为 /pose
    无人船的姿态数据，发送频率同传感器设备的频率，一般不低于5Hz，参数格式为：|艏向（4B）|纵摇（4B）|横摇（4B）|
    艏向：单精度浮点数，单位为度，正北为0，顺时针为正方向
    纵摇：单精度浮点数，单位为度
    横摇：单精度浮点数，单位为度
18. 命令-0x010F（Client <-- USV），对应的mqtt消息为 /vtg
    无人船的速度和航向，发送频率同传感器设备的频率，一般不低于5Hz，参数格式为：|速度（4B）|航向（4B）|
    速度：单精度浮点数，单位为米每秒
    航向：单精度浮点数，单位为度，正北为0，顺时针为正方向
19. 命令-0x0110（Client <-- USV），对应的mqtt消息为 /vel
    无人船的速度，发送频率同传感器设备的频率，一般不低于5Hz，参数格式为：|速度（4B）|

速度：单精度浮点数，单位为米每秒

20. 命令-0x0111（Client <-- USV），对应的mqtt消息为 /hdt
    无人船的艏向，发送频率同传感器设备的频率，一般不低于5Hz，参数格式为：|艏向（4B）|
    艏向：航向：单精度浮点数，单位为度，正北为0，顺时针为正方向
21. 命令-0x0112（Client <-- USV），对应的mqtt消息为 /bat
    无人船剩余电量的百分比，发送频率0.2Hz，参数格式为：|电量百分比（1B）|
    电量百分比：0 ～ 100，其他值表示没有采集到剩余电量数据。
22. 命令-0x0113（Client <-- USV），对应的mqtt消息为 /radar/object
    雷达探测到的目标信息，参数格式为：|编号（1B）|距离（4B）|方位（4B）|
    编号：0～255，雷达能同时探测多个障碍物，每个障碍物都有编号
    距离：单精度浮点数，障碍物相对船的距离，单位为米
    方位：单精度浮点数，障碍物相对船头指向的方位，顺时针为正，单位为度
23. 命令-0x0114（Client <-- USV），对应的mqtt消息为 /radar/status
    无人船当前的避障状态以及避障行为，发送频率0.5Hz，参数格式为：|避障使能（1B）|避障行为（1B）|
    避障使能：0表示已经禁用避障，1表示已经启用避障
    避障行为：0表示无避障，1表示向左避障，2表示向右避障，3表示向后避障
24. 命令-0x0115（Client --> USV），对应的mqtt消息为 /radar/set
    启用或关闭避障功能，参数格式为：|避障使能（1B）|
    避障使能：0表示禁用避障，1表示启用避障
25. 命令-0x0116（Client --> USV），对应的mqtt消息为 /status/get
    获取无人船当前的状态，无人船接收到这个指令后，会马上给客户端返回状态数据 /status，该命令无参数
26. 命令-0x0117（Client --> USV），对应的mqtt消息为 /wp/set
    设置无人船的航行任务，参数格式为：|任务类型（1B）|任务点数（2B）| 任务点数据0 ... 任务点数据n | 采样点数据0 ... 采样点数据x | 监测点数据0 ... 监测点数据 y |
    任务类型：0 工作任务，1 返航任务， 2 伴航任务
    任务点数：无符号整数，表示任务点的个数
    任务数据：任务数据的总长度由任务点数决定，每个任务点数据的长度是17个字节，依次为| 类型（1B）|纬度（8B）|经度（8B）|，类型的第0位表示当前点是否为采样点，第1位表示当前点是否为监测点，第2位表示当前点是否为测绘点，纬度为双精度浮点数，精度为双精度浮点数，单位为度。一个点可以为采样监测和测绘的任意组合。
    采样数据：采样数据的总长度由任务点中的采样点数决定，每个采样点的数据长度是9个字节，依次为| 采样瓶号（1B）|采样容量（4B）|
    采样深度（4B）|，采样瓶号的范围是0到255，采样容量为单精度浮点型，单位为升，采样深度为单精度浮点数，单位为米。
    监测数据：监测数据的总长度由任务点中的监测点数决定，每个监测点的数据长度是2个字节，为16位整数，表示在该点的监测时间长度，单位为秒。
27. 命令-0x0118（Client <-- USV），对应的mqtt消息为 /wp/info
    无人船返回当前的航行任务，参数格式为：|任务数据（不定长，最少20B）|
    任务数据：与/wp/set的内容一致。
28. 命令-0x0119（Client --> USV），对应的mqtt消息为 /wp/get
    获取无人船当前的任务数据，无人船接收到这个指令后，会马上给客户端返回任务数据/wp/info，该命令无参数。
29. 命令-0x011A（Client --> USV），对应的mqtt消息为 /ping
    基站与无人船建立连接后，会每隔3秒向无人船发送这个命令，表示基站与无人船的连接正常。该命令无参数。
30. 命令-0x011B（Client <-- USV），对应的mqtt消息为 /bat/info
    无人船返回实时的电池参数，参数如格式为：|功率（4B）|电压（4B）|电流（4B）|温度（2B）|
    功率：单精度浮点数，单位为瓦
    电压：单精度浮点数，单位为伏
    电流：单精度浮点数，单位为安
    温度：16位整数，单位为摄氏度
31. 命令-0x011C（Client --> USV），对应的mqtt消息为 /speed/set
    设置无人船自动航行的最小速度，最大速度和速度控制类型，参数格式为：|最小速度（4B）|最大速度（4B）|速度控制类型（1B）|
    最小速度：单精度浮点数，取值范围为0～1
    最大速度：单精度浮点数，取值范围为0～1
    控制类型：0 定功率方式，1 定速度方式
32. 命令-0x011D（Client --> USV），对应的mqtt消息为 /speed/get
    获取当前设置的自动航行的速度参数，无人船收到改命令后，马上会发送 /speed
33. 命令-0x011E（Client <-- USV），对应的mqtt消息为 /speed
    返回无人船当前设置的速度参数，参数格式与/speed/set一致
34. 命令-0x011F（Client --> USV），对应的mqtt消息为 /speed/pid/set
    设置无人船自动航行速度控制的PID参数，参数格式为：|P（4B）|I（4B）|D（4B）|MaxI（4B）|
    P：单精度浮点数
    I：单精度浮点数
    D：单精度浮点数
    MaxI：单精度浮点数
35. 命令-0x0120（Client --> USV），对应的mqtt消息为 /speed/pid/get
    获取当前设置的自动航行的速度PID参数，无人船收到改命令后，马上会发送 /speed/pid
36. 命令-0x0121（Client <-- USV），对应的mqtt消息为 /speed/pid
    返回无人船当前设置的速度PID参数，参数格式与/speed/pid/set一致

37. 命令-0x0122（Client --> USV），对应的mqtt消息为 /rudder/pid/set
    设置无人船自动航行舵角控制的PID参数，参数格式为：|P（4B）|I（4B）|D（4B）|MaxI（4B）|
    P：单精度浮点数
    I：单精度浮点数
    D：单精度浮点数
    MaxI：单精度浮点数
38. 命令-0x0123（Client --> USV），对应的mqtt消息为 /rudder/pid/get
    获取当前设置的自动航行的舵角PID参数，无人船收到改命令后，马上会发送 /rudder/pid
39. 命令-0x0124（Client <-- USV），对应的mqtt消息为 /rudder/pid
    返回无人船当前设置的舵角PID参数，参数格式与/rudder/pid/set一致。
40. 命令-0x0125（Client <-- USV），对应的mqtt消息为 /target
    返回当前的追踪点位置，调试专用，参数格式同 /gps。
41. 命令-0x0126（Client --> USV），对应的mqtt消息为 /home/pos/set
    设置无人船的返航点，参数格式同 /gps。
42. 命令-0x0127（Client <-- USV），对应的mqtt消息为 /home/pos
    无人船返回当前的返航点，发送频率为0.2Hz，参数格式同 /gps。
43. 命令-0x0128（Client --> USV），对应的mqtt消息为 /back/set
    设置无人船的自动返航条件，参数格式为：|返航条件（1B）|
    返航条件：第0位表示是否允许低电量自动返航，第1位表示是否允许通信失联自动返航。
44. 命令-0x0129（Client <-- USV），对应的mqtt消息为 /back/status
    返回无人船当前设置的返航条件。参数格式同 /back/set
45. 命令-0x012A（Client <-- USV），对应的mqtt消息为 /device/status
    返回无人船上各个外设的设备状态，参数格式如下：|设备ID（1B）|状态数据（不定长）|
    设备ID：外部设备的唯一ID
    状态数据：由设备ID决定数据内容以及长度（待完善）
46. 命令-0x012B（Client <-- USV），对应的mqtt消息为 /datetime
    当前的时间，精确到秒，该时间从GPS信息中获取，参数格式如下：|年（1B）|月（1B）|日（1B）|时（1B）|分（1B）|秒（1B）|
    年：8位整数，当前年份减去2000，比如2018年时，数值为0x12
    月：8位整数，1 ~ 12
    日：8位整数，1 ~ 31
    时：8位整数，0 ~ 59
    分：8位整数，0 ~ 59
    秒：8位整数，0 ~ 59
47. 命令-0x012C（Client --> USV），对应的mqtt消息为 /device/set
    给无人船的外部设备发送控制命令，参数格式如下：|设备ID（1B）|命令数据（不定长）|
    设备ID：外部设备的唯一ID
    状态数据：由设备ID决定数据内容以及长度（待完善）
48. 命令-0x0200（Client --> Remoter），无对应的mqtt消息
    设置遥控器的信道和地址命令，参数格式为：|信道（1B）|地址（2B）|
    信道：遥控器的通信信道，取值范围是0～9
    地址：遥控器的通信地址，取值范围为0x11 ~ 0x7FFF
49. 命令-0x0201（Client <-- Remoter），无对应的mqtt消息|左边-X轴（2B）|左边-Y轴（2B）|右边-X轴（2B）|右边-Y轴（2B）|旋钮（2B）|模式（1B）|采样（1B）|警报（1B）|菜单键（1B）|确定键（1B）|取消键（1B）|
    左边-X轴：0 ～ 4095
    左边-Y轴：0 ～ 4095
    右边-X轴：0 ～ 4095
    右边-Y轴：0 ～ 4095
    旋钮：0 ～ 4095
    模式：0 ～ 2
    采样：0或1
    警报：0或1
    菜单键：0或1
    确定键：0或1
    取消键：0或1
50. 命令-0x0300（Remoter --> USV），对应的mqtt消息为 /sample/start
    手动采样命令，参数格式为：|采样点号（2B）|采样瓶号（1B）|采样容量（4B）|采样深度（4B）|
    采样点号：必须设置为0xFFFF
    采样瓶号：0～255，实际最大瓶号以实际船的配置为准
    采样容量：单精度浮点数，单位为升
    采样深度：单精度浮点数，单位为米
51. 命令-0x0301（Remoter --> USV），对应的mqtt消息为 /sample/cancel
    取消正在执行的采样命令，无参数
52. 命令-0x0302（Remoter --> USV），对应的mqtt消息为 /monitor/start
    手动监测命令，参数格式为：|监测点号（2B）|监测时间（2B）|
    监测点号：必须设置为0xFFFF
    监测时间：16位整数，单位为秒

53. 命令-0x0303（Remoter --> USV），对应的mqtt消息为 /monitor/cancel
    取消正在执行的监测命令，无参数
54. 命令-0x0304（Client <-- USV），对应的mqtt消息为 /sample/record
    采样数据，某个点的采样完成后，主控会发送数据到基站，参数格式为：|序列号（4B）|时间戳（4B）|纬度（8B）|
    经度（8B）|采样点号（2B）|采样瓶号（1B）|采样容量（4B）|采样深度（4B）|
    序列号：采样序列号，无符号整数
    时间戳：开始采样时的时间戳
    纬度：双精度浮点数
    经度：双精度浮点数
    采样点号：表示在某个路径点进行的采样，0xFFFF表示是手动采样。
    采样瓶号：0～255，实际最大瓶号以实际船的配置为准
    采样容量：单精度浮点数，单位为升
    采样深度：单精度浮点数，单位为米
55. 命令-0x0305（Client <-- USV），对应的mqtt消息为 /sample/progress
    采样的进度，参数格式为：|进度百分比（1B）|
    进度百分比：0 ～ 100
56. 命令-0x0306（Client <-- USV），对应的mqtt消息为 /monitor/record
    监测数据，某个点监测完成后，主控会发送数据到基站，参数格式为：|序列号（4B）|时间戳（4B）|纬度（8B）|
    经度（8B）|监测点号（2B）|数据个数（1B）|数据ID（4B）|数据值（4B）|
    序列号：监测序列号，无符号整数
    时间戳：开始监测时的时间戳
    纬度：双精度浮点数
    经度：双精度浮点数
    监测点号：表示在某个路径点进行的监测，0xFFFF表示是手动监测。
    数据个数：表示后边带的数据个数，每个数据有ID和值，每个数据占8个字节
    数据ID：表示数据的类型，32位正整数，ID关联的数据项请参考监测棒的文档
    数据值：单精度浮点数，表示具体的数值。
57. 命令-0x0307（Client <-- USV），对应的mqtt消息为 /monitor/progress
    监测的进度，参数格式为：|进度百分比（1B）|
    进度百分比：0 ～ 100
58. 命令-0x0500（Remoter --> USV），对应的mqtt消息为 /xtend/params/set
    设置数传设备的参数，参数格式如下：|修改主控（1B）|修改遥控（1B）|数传信道（1B）|数传ID（2B）|
    修改主控：是否修改主控的数传参数，0或者1
    修改遥控：是否修改遥控器的数传参数，0或者1
    数传信道：0～9
    数传ID: 17 ～ 32767
59. 命令-0x0501（Remoter --> USV），对应的mqtt消息为 /xtend/params/get
    获取数传设备的参数，该命令无参数
60. 命令-0x0502（Remoter <-- USV），对应的mqtt消息为 /xtend/status
    无人船返回传设备的参数，参数格式为：|数传信道（1B）|数传ID（2B）|
    数传信道：0～9
    数传ID: 17 ～ 32767
61. 命令-0xFF00（Client <--> USV）
    数据包拆分命令，由于通过串口进行数据传输的时候，太大的数据包会由于丢数据导致整包数据无效，而后边的重传会浪费带宽，故需要把较大的数据包拆分成较小的数据包进行传输。目前设置的每包数据最大为50字节，也就是说如果发送的数据（命令 + 扩展 + 序号 +
    参数）超过50字节，那么它会被拆成多包发送，拆包的时候会检查扩展的值，如果是需要回复，那么分拆后的每一包都会使用同样的扩展值，但是原数据中的扩展值会被设置成0，序号也会被移除掉（分拆后，原来包的扩展值已经没有意义，而且还能省2个字节）。接收端接收到数据时，需要把数据进行合并处理，合并后的数据没有序号项，扩展项也一定是0。参数格式为：|包序号（1B）|数据（不定长）|
    包序号：0～127，最高位为1时，表示最后一包数据，发送端按照序号顺序发送。
    数据：拆分的数据，接收端需要根据序号重组。

5. 数据包封包/解包协议
   使用TCP发送数据时，上边描述的所有内容（命令 + 扩展 + 序号 + 参数 +
   校验）需要经过打包后才能发送，每条完整的命令打包成一个数据包，数据包以字节PACKET_START开始，以字节PACKET_END结束，PACKET_START和PACKET_END之间的数据为数据内容。如果数据内容中包含PACKET_START，PACKET_END，PACKET_ESCAPE，则需要对这些字节做特殊处理，使用两个字节替换原字节，这两个字节的第1个字节是PACKET_ESCAPE，第2个字节是原字节与PACKET_ESCAPE_MASK异或后的值。解包的时候逆向应用规则即可。封包解包用到的宏定义
   如下：

```
#define   PACKET_START                    0xAC
#define   PACKET_END                      0xAD
#define   PACKET_ESCAPE                   0xAE
#define   PACKET_ESCAPE_MASK              0x80
```

封包与解包实例代码如下：

```
#define APPEND_BYTE(b, d, idx, max) \
do { \
        if(PACKET_START == (b) || PACKET_END == (b) || PACKET_ESCAPE == (b)) \
        { \
                if ((idx) >= max) return -1; \
                (d)[(idx)++] = PACKET_ESCAPE; \
                if ((idx) >= max) return -1; \
                (d)[(idx)++] = (b) ^ PACKET_ESCAPE_MASK; \
        } \
        else \
        { \
                if ((idx) >= max) return -1; \
                (d)[(idx)++] = (b); \
        } \
} while(0)

uint8_t Protocol_CRC8(const uint8_t *data, uint32_t size)
{
        uint32_t i = 0;
        uint32_t j = 0;
        uint8_t crc = 0;

        for (i = 0; i < size; ++i)
        {
                crc = crc ^ data[i];
                for (j = 0; j < 8; ++j)
                {
                        if ((crc & 0x01) != 0)
                        {
                                crc = (crc >> 1) ^ 0x8C;
                        }
                        else
                        {
                                crc >>= 1;
                        }
                }
        }
        return crc;
}

// 将数据封包，data是要封包的数据，不包括CRC8的值
// 结果数据保存在buffer中，buffer的大小应该大于size * 2 + 3才能保证空间足够。
// 返回值大于0表示结果数据的长度，-1表示buffer空间不够
int32_t Protocol_Pack(const uint8_t *data, uint32_t size, uint8_t *buffer,
uint32_t buf_len)
{
        uint32_t index = 0;
        uint32_t ret_len = 0;
        uint8_t crc = Protocol_CRC8(data, size);

        buffer[ret_len++] = PACKET_START;
        for (index = 0; index < size; ++index)
        {
                APPEND_BYTE(data[index], buffer, ret_len, buf_len);
        }
        APPEND_BYTE(crc, buffer, ret_len, buf_len);

        if (ret_len >= buf_len)
        {
                return -1;
        }

        buffer[ret_len++] = PACKET_END;
        return ret_len;
```

```
}

//  对数据进行解包，解包后的数据就保存在data中
//  返回值大于0表示解包后的数据长度，-1表示解包错误，CRC不匹配。
int32_t  Protocol_UnPack(uint8_t  *data,  uint32_t  size)
{
        uint32_t  index  =  0;
        int32_t  pack_len  =  0;
        uint8_t  crc  =  0;

        for  (index  =  0;  index  <  size;  ++index)
        {
                if  ((PACKET_START  ==  data[index])  ||  (PACKET_END  ==  data[index]))
                {
                        continue;
                }

                if  (PACKET_ESCAPE  ==  data[index])
                {
                        ++index;
                        if  (index  >=  size)
                        {
                                return  -1;
                        }

                        data[pack_len]  =  data[index]  ^  PACKET_ESCAPE_MASK;
                        ++pack_len;
                        continue;
                }

                data[pack_len]  =  data[index];
                ++pack_len;
        }

        if  (pack_len  >  0)
        {
                crc  =  Protocol_CRC8(data,  pack_len  -  1);
                if  (crc  !=  data[pack_len  -  1])
                {
                        return  -1;  //  CRC  error
                }
        }

        return  pack_len  -  1;
}
```

客户端通过TCP接收到无人船的数据后，需要根据通过包头字节与包尾字节对每一包数据进行拆分，然后把拆分后的数据传递给上边的解包函数，就可以得到实际的数据。

历史记录

#1 - 2019-05-08 20:08 - 坚 黄

- 描述 已更新。