# **网络接口协议 V3.22**

# 1、变更记录

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **时间** | **作者** | **记录** |
| V1.0 | 2023/08/01 | luffy，sky | 参展版本 |
| V2.0 | 2023/08/16 | luffy | 1、调整协议结构  2、新增协议校验功能 |
| V2.1 | 2023/08/24 | luffy | 1、新增2.4和2.5小节  2、增加了网络状态、系统状态、运行信息的数据上报接口 |
| V2.2 | 2023/08/30 | luffy | 新增4.4.4小节，响应超时处理 |
| V2.3 | 2023/09/05 | luffy | 1、修改CRC16校验初始值为0x9966  2、修改6.1小节，增加AES加密细节 |
| V2.4 | 2023/09/05 | 杨润 | 修改6.1小节，修改IV向量的类型 |
| V2.5 | 2023/09/07 | luffy | 修改4.5小节，细化消息加密方式，规定了内部加密key |
| V2.6 | 2023/09/13 | luffy | 修改6.2节，新增证书分段下发 |
| V2.7 | 2023/09/15 | luffy | 1、修改6.8小节，新增恢复出厂上报  2、删除6.5小节，将系统状态合并到6.7小节信息上报 |
| V2.8 | 2023/09/22 | luffy | 1、新增2.6、2.7小节，对TCP通道和SN做了规范说明  2、修改2.5小节，对MQTT的thing命名规则做了修改  3、修改6.11.3小节，新增OTA状态主动上报接口  4、修改6.6小节，新增设备信息MQTT通道开机上报一次 |
| V2.9 | 2023/09/25 | luffy | 1、修改6.9小节，透传指令增加机器地址字段，用于服务器识别机器数据  2、新增6.13小节，水质传感器数据读取上报 |
| V2.10 | 2023/10/09 | luffy | 1、修改6.13小节，新增传感器序列号字段  2、修改6.6小节，新增无线地址字段上报  3、新增6.14小节，新增无线模块地址的读取设置 |
| V2.11 | 2023/10/17 | luffy | 1、修改6.6小节，新增机器版本号合并规则  2、修改6.11.1小节，URL升级方式改为多个文件数组下发  3、修改6.11.2小节，块升级方式改为多个文件数组下发  4、修改6.11.3小节，升级状态上报新增机器序列号和版本号字段 |
| V2.12 | 2023/10/18 | 杨润 | 1.修改6.7小节，去掉了蓝牙信号字段，改为APP设备端获取。 |
| V2.13 | 2023/10/23 | 杨润 | 1.修改6.11.3小节，加入state字段区分下载进度和安装进度。 |
| V2.14 | 2023/10/25 | luffy | 1、修改6.11.1、6.11.2、6.11.2小节，支持大版本号  2、修改6.6，新增支持大版本号 |
| V2.15 | 2023/11/03 | luffy | 修改2.2、2.3小节，更新蓝牙wifi配网流程图 |
| V2.16 | 2023/11/14 | luffy、杨润 | 1、修改2.2、2.3小节，更新蓝牙wifi配网流程，证书变为设备端请求  2、修改6.1小节，配网命令添加用户token信息 |
| V2.179 app可以选择十字弓还是单边弓 | 2023/11/22 | luffy | 1、修改6.13小节，新增水质传感器校准指令  2、新增6.15小节，加入蓝牙身份验证  3、修改第3节，完善蓝牙和AP命名规则 |
| V2.18 | 2023/11/24 | luffy | 修改第3节，完善蓝牙和AP命名规则 |
| V2.19 | 2023/11/28 | 杨润 | 修改6.11.3 修改OTA的错误状态码 |
| V2.20 | 2023/12/01 | luffy | 修改6.11小节，支持OTA超时字段 |
| v2.21 | 2024/04/15 | 潘楼 | 修改协议格式，"data", "type"改成key-value对应，与shadow doc统一 |
| v2.22 | 2024/04/16 | 万卢海 | 添加与APP的接口 |
| v2.23 | 2024/4/23 | 万卢海 | 增加 需要 shadow 的请求  6.18 清洁动作、6.19 标准清洁、6.21 风扇、6.22 水泵、6.32 音量设置、6.34 速度设置、6.41 语音包设置 |
| v.2.24 | 2024/4/25 | 潘楼 | 增加遗言topic |
| v.2.25 | 2024/4/29 | 万卢海 | 增加了需要上报的消息(以shaow形式上报)  6.18 清洁动作、6.27 告警列表、6.28 获取地图、  6.29 运行轨迹、6.31 电量阈值、6.33 音量、  6.37 充电状态、6.38 清洁记录、6.40 语音包列表、  6.42 语音包、6.SetStatus43 当前位置 |
| v.2.26 | 2024/4/29 | 万卢海 | 修改：  6.7 修改机器状态包含的值，新增wifi名称  6.17 任务管理增加了任务相关逻辑的字段  新增：  6.44 任务查询/当前工作模式查询  6.45 是否关联充电钨  6.46 垃圾篓状态  6.47 在水信息 |
| v.2.27 | 2024/5/15 | 胡恒如 | 6.48 LORA ID设置规则 |
| v.2.28 | 2024/5/15 | 万卢海 | 修改  6.27 增加错误码对应的状态 |
| v.2.29 | 2024/5/21 | 李相龙 | 6.11 ota推送内容增加固件包字节大小 fileSize |
| v.2.30 | 2024/5/22 | 李相龙 | 6.11 UrlOta接口协议内容 sites类型变更 |
| v.2.31 | 2024/5/23 | 潘楼 | MQTT主题修改 |
| v.2.32 | 2024/5/28 | 万卢海 | 上报字段的key修改 |
| v.2.33 | 2024/5/29 | 潘楼 | 6.49 设备文件上传S3存储 |
| v.2.34 | 2024/5/30 | 李相龙 | 6.11.4 OTA 升级结果上报 6.6 设备信息上报增加充电钨版本及绑定状态 |
| v.2.35 | 2024/5/31 | 李相龙 | 6.6 设备信息上报新增cloudver |
| v.2.36 | 2024/6/4 | 李相龙 | 6.11.4 OtaResultReport 增加sn字段 |
| v.2.37 | 2024/6/6 | 万卢海 | 修改：  6.17-针快速任务的字段删减(功能改变) |
| v.2.38 | 2024/6/14 | 万卢海 | 修改：  6.27-增加错误码属性字段 |
| v.2.39 | 2024/6/19 | 万卢海 | 修改：  6.38-重构清扫数据记录 |
| v.2.40 | 2024/6/27 | 李相龙 | 6.47 在水状态 字段命名修改  6.6 devinfo协议字段 nocharge命名变更为hascharge  6.7 opinfo协议删除ip字段 |
| v2.42 | 2024/7/11 | 徐志勇 | 6.51 获取配网模式 |
| v2.43 | 2024/7/11 | 李相龙 | 6.8 恢复出厂设置 6.11 OTA请求 6.12 重启 6.25 关机 6.39 日志拉取 6.49 设备文件上传s3 新增controlLogId (远程控制时存在) |
| v2.44 | 2024/7/12 | 徐志勇 | 6.27 获取告警列表 修改数据格式 |
| v2.45 | 2024/7/15 | 万卢海 | 修改：  6.44 修改成：获取当天任务队列中的任务 |
| v2.46 | 2024/7/15 | 万卢海 | 修改：  6.17 清扫区域增加一个水线区域 |
| v2.47 | 2024/7/23 | 王清 | 6.39 删除无效key，重定义type用法。 |
| v2.48 | 2024/7/23 | 徐志勇 | 6.54 新增自检 |
| v2.49 | 2024/7/25 | 徐志勇 | 6.54 删除自检，改为重置 |
| v3.00 | 2024/8/13 | 李相龙 | 6.7 opInfo 增加 wifiName字段；6.6 devInfo增加hotSpot字段 |
| v3.01 | 2024/8/22 | 李相龙 | 6.55 6.56 速度设置 手动模式控制 上浮下潜相关接口定义 |
| V3.02 | 2024/8/27 | 万卢海 | 6.38 清洁记录新增(修改)   1. 各个细分区域耗电量 2. 当前任务清扫了哪些区域 3. 避障次数，脱困次数，异常次数 4. 异常终止原因(cleanResult) |
| V3.03 | 2024/09/11 | 万卢海 | 修改：  6.17.1 下发任务中新增一个多平台(临时版本) |
| mode  int  池底清洁时遍历模式  1 - 打开多平台模式  2 - 关闭多平台模式  V3.04 | 2024/09/18 | 万卢海 | 修改：  6.17 清洁强度定义修改 |
| V3.05 | 2024/09/23 | 王清 | 修改：  6.39 添加sensor\_data |
| V3.06 | 2024/09/25 | 徐志勇 | 新增：  6.57 获取部件寿命 |
| V3.07 | 2024/10/10 | 李相龙 | 6.6 DevInfo 新增 mcuver 和blever版本号 |
| V3.08 | 2024/10/22 | 徐志勇 | 修改：  6.7 status 字段加回 9（报警中） |
| V3.09 | 2024/10/28 | 万卢海 | 新增：  6.58 设置是否屏蔽按键  6.59 获取是否屏蔽按键  6.60 设置水面工作属性  6.61 获取水面工作属性 |
| V3.10 | 2024/10/30 | 万卢海 | 修改：  6.7 机器状态上报增加充放气状态  6.34 速度设置  6.35 速度获取 |
| V3.11 | 2024/10/31 | 万卢海 | 新增：  6.62 设置池底遍历模式  6.63 获取池底遍历模式 |
| V3.12 | 2024/11/1 | 万卢海 | 修改：  6.38 清洁记录内容删减与增加 |
| V3.13 | 2024/11/1 | 徐志勇 | 新增：  6.64 设置召回回到入水点  6.65 获取召回回到入水点 |
| V3.14 | 2024/11/5 | 万卢海 | 新增：  6.66 OTA中 按键触发  6.67 设置实验开关  6.68 获取实验开关 |
| V3.15 | 2024/11/6 | 徐志勇 | 修改：  6.27.2 增加ts字段 |
| V3.16 | 2024/11/7 | 万卢海 | 新增：  6.69 获取气囊状态 |
| V3.17 | 2024/11/16 | 徐志勇 | 修改：  6.8 标题由“恢复出厂设置&上报”改为“解除App绑定&上报”  新增：  6.70 恢复出厂设置 |
| V3.18 | 2024/11/27 | 万卢海 | 新增：  6.71 设置多平台  6.72 获取多平台 |
| V3.19 | 2024/11/28 | 徐志勇 | 新增：  6.7 status字段增加15表示关机 |
| V3.20 | 2024/11/29 | 徐志勇 | 新增：  6.7 增加w2自定协议 |
| V3.21 | 2024/12/03 | 万卢海 | 修改：  6.6 增加 ango 版本号 |
| V3.22 | 2024/12/3 | 潘楼 | 新增：  OtaResultReport增加失败原因，mcu主板，灯板，岸歌升级结果 |
| V3.23 | 2024/12/9 | 徐志勇 | 新增：  6.7 w2自定协议增加15 - 多平台清洁模式 |
| V3.24 | 2024/12/20 | 万卢海 | 增加：  6.74 设置浅水区策略  6.75 获取浅水区策略 |
| V3.25 | 2024/12/4 | 万卢海 | 增加：  6.76 脚本模式 |
| V3.26 | 2024/12/24 | 王清 | 6.77添加日志开关 |
| V3.27 | 2024/12/4 | 万卢海 | 增加：  6.34 恢复使用且修改水泵设置 |

# 2、简介

V3.17

2024/11/16

徐志勇

修改：

6.8 标题由“恢复出厂设置&上报”改为“解除App绑定&上报”

新增：

6.70 恢复出厂设置V3.17

2024/11/16

徐志勇

修改：

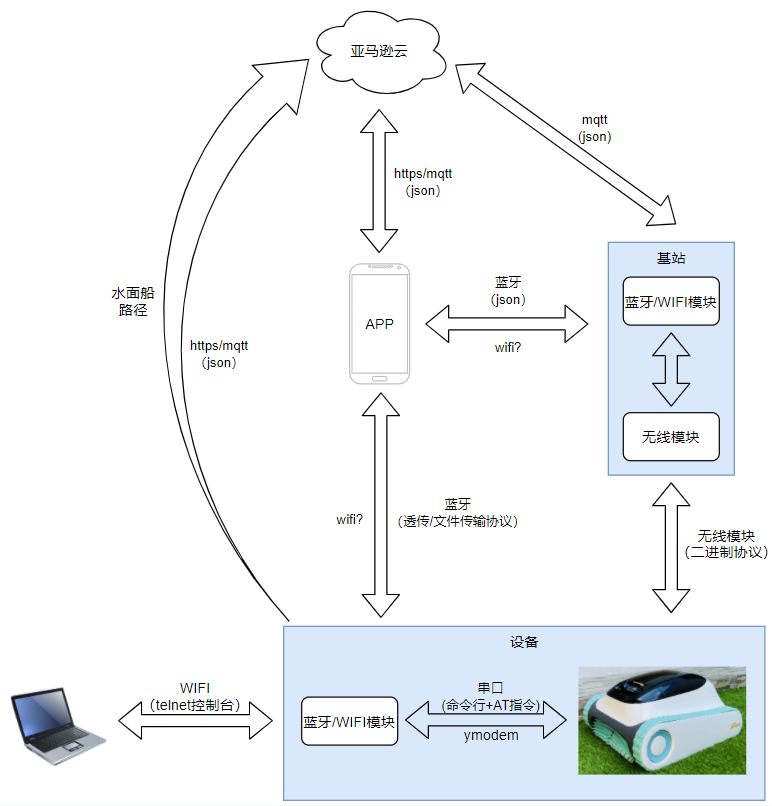
6.8 标题由“恢复出厂设置&上报”改为“解除App绑定&上报”

新增：

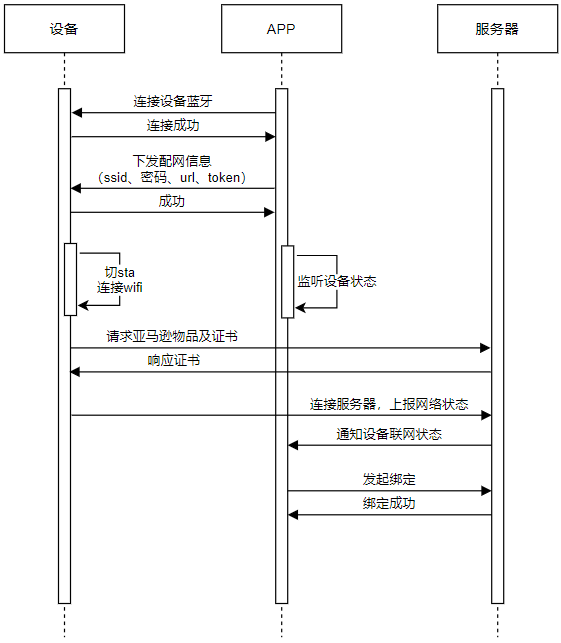
6.70 恢复出厂设置

**适用范围：所有json格式的通讯，包括：模组与APP通讯，模组与服务器通讯、**

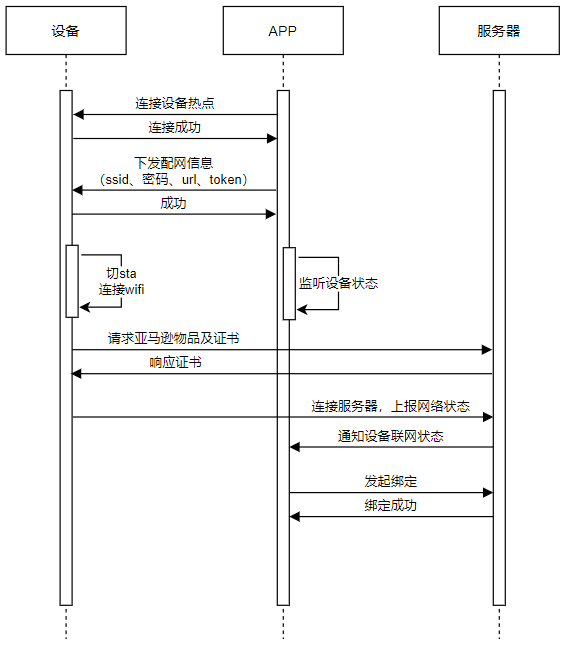
## **2.1、网络拓扑图**



## **2.2、蓝牙配网流程**

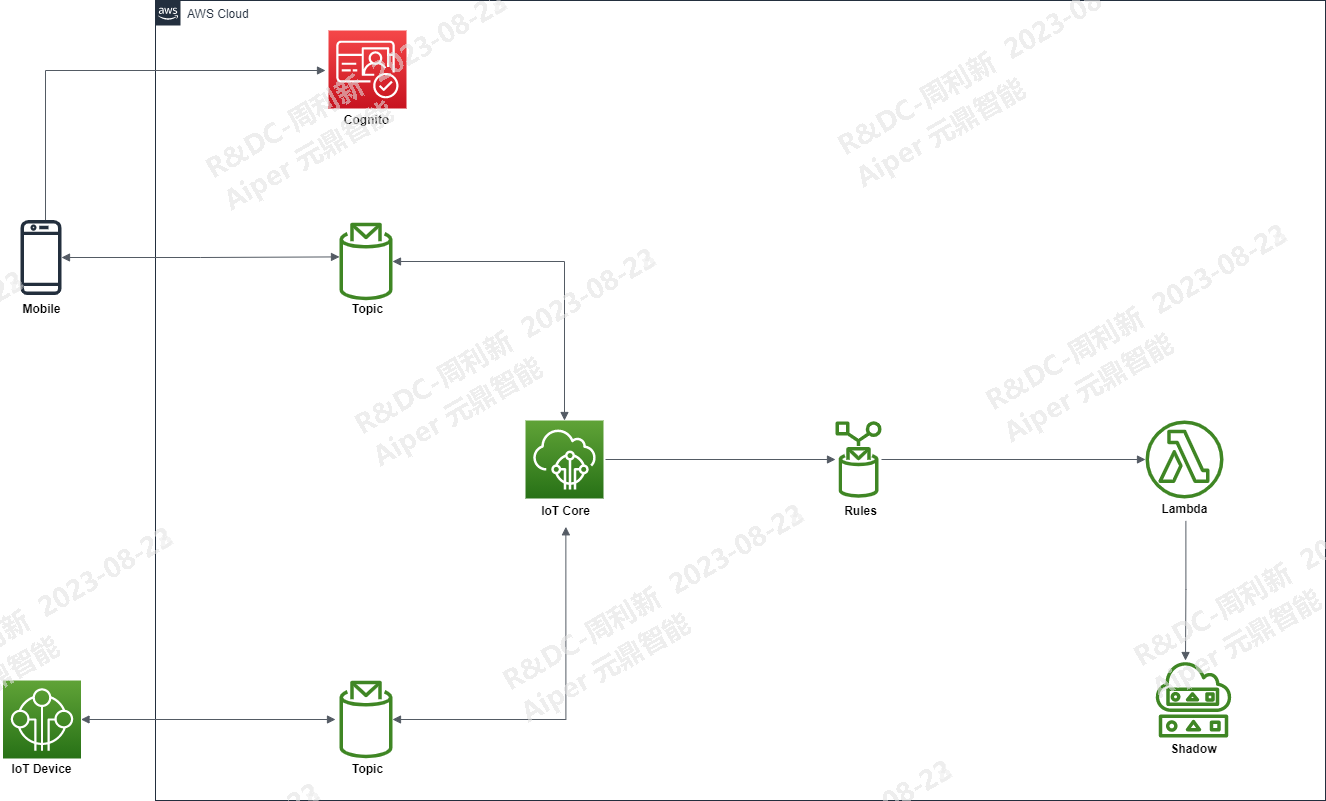


## **2.3、AP配网流程**

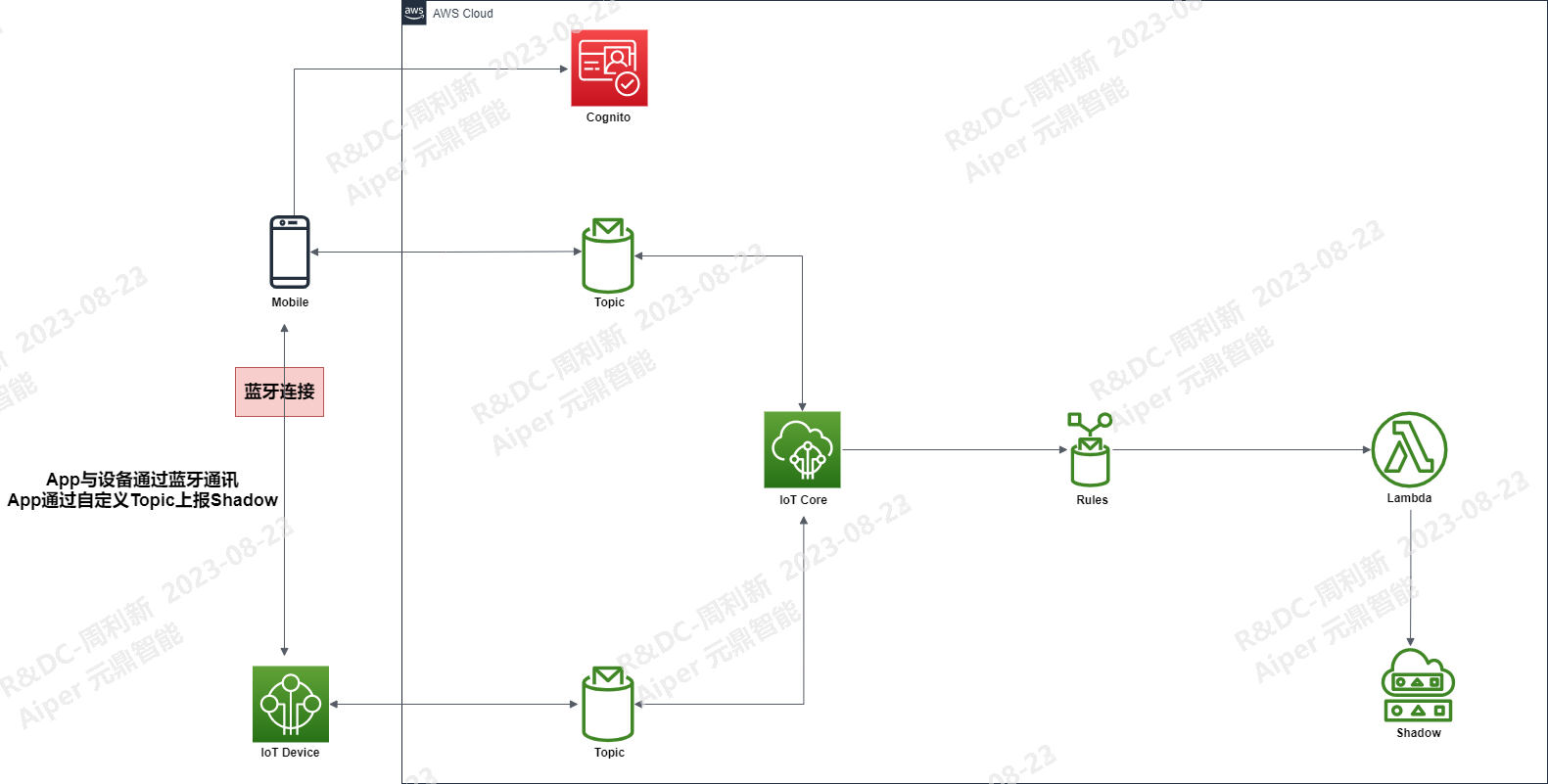


## **2.4、模组&服务器&APP交互方式**

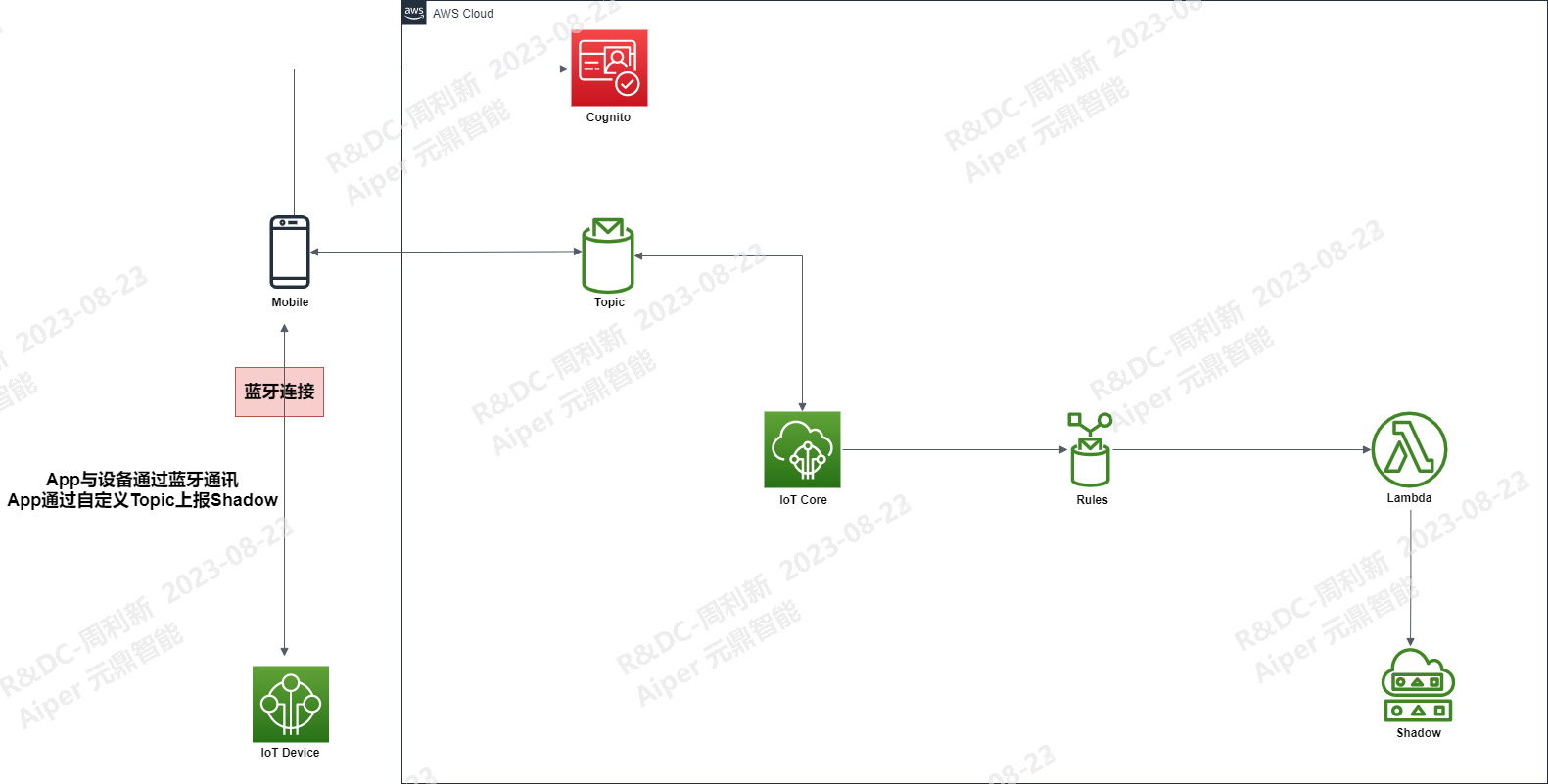
* 设备连上IOT，未连接蓝牙



* 设备连上IOT，同时连接蓝牙



* 设备连接蓝牙，未连接IOT



## 2.5、MQTT主题

下行通道：aiper/things/thingName/downChan

上行通道：aiper/things/thingName/upChan

上报通道：aiper/things/thingName/app/report

上报通道：aiper/things/thingName/cloud/report

上报通道：aiper/things/thingName/bigdata/report

设备断连：aiper/things/thingName/willChan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **下行通道** | **上行通道** | **上报通道** | **上报通道** | **上报通道** | **断连上报** |
| 主题名 | aiper/things/thingName/downChan | aiper/things/thingName/upChan | aiper/things/thingName/cloud/report | aiper/things/thingName/app/report | aiper/things/thingName/bigdata/report | aiper/things/thingName/willChan |
| 发布者 | app | 模组 | 模组 | 模组 | 模组 | 服务器 |
| 订阅者 | 模组 | app | 服务器 | app | 服务器 |  |
| 用途 | app请求 | 模组响应 | 模组主动上报，服务端监听 | 模组主动上报，app监听 | 大数据平台分析 | 设备在线时由服务器和设备保持连接 ，设备离线时由服务器发布 |

thingName命名规则：产品11位SN

## 2.6、TCP通道

用于APP与模组之间的局域网之间交互（包括AP配网），采用C/S模型，模组作为服务端，APP作为客户端，模组监听端口8888

## **2.7、SN规则**

SN组成：机型代码（2位）+ 工厂代码（1位拼音缩写）+生产年月（3位数）+序列号（5位）共**11**位

如：AABYWW00000 S/N 共**11**位

详细如下：

AA:机型代码（固定）,例如:

ZT2001=21,ZT2002=22,ZT2003=23;ZT3001=31,ZT3002=32,ZT6001=61,ZT6002=62,H2=H2,V2=V2,S1=S1,X5=X5,X5pro=XS,X1=X1,X3pro=X3,X6=X6,X7pro=X7,X9=X9，T1pro=TX，T1=T1后续机型以此类推

B: 代工厂代码，元鸿=Y, 欣智旺=X，**和而泰=H，和而泰越南=V，比亚迪=B**

Y: 年份（一位），2023=3,2024=4......以此类推

WW：周别，01,02,03,04,05,06,07,08,09,10，11.......

00000: 流水号，按照00001,00002……以此类推，每个周流水号不可重复，下个周重新开始从00001开始

# 3、蓝牙/AP命名规则

## **3.1、AP正常命名规则**

产品型号-SN后5位

|  |  |  |
| --- | --- | --- |
| **内部代号** | **产品型号** | **举例** |
| S1 | Surfer S1 | Surfer S1-12345 |
| Surfer M1 | Surfer M1-12345 |
| X6 | Scuba S1 Pro | Scuba S1 Pro-12345 |
| Scuba N1 Pro | Scuba N1 Pro-12345 |
| X7 | Scuba N1 Ultra | Scuba N1 Ultra-12345 |
| T1 | Scuba X1 | Scuba X1 -12345 |
| T1Pro | Scuba X1 Pro | Scuba X1 Pro-12345 |
| X9 | Scuba X1 Pro Max | Scuba X1 Pro Max-12345 |

## 3.2、AP生产命名规则

产品型号-SN

如：Surfer S1-AABYWW00000

## **3.3、蓝牙广播信息**

包含：品牌信息、产品型号、序列号

如：Aiper-Surfer S1-AABYWW00000

# 4、协议格式

## 4.1、请求

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **约束** | **说明** |
| type | 字符串 | 必备 | 消息类型 |
| data | 对象 | 可省 | 消息内容 |
| chksum | 数字 | 可省 | 采用CRC16校验，对数据内容进行校验 |

## 4.2、响应

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **约束** | **说明** |
| type | 字符串 | 必备 | 消息类型 |
| res | 数字 | 必备 | 返回结果 |
| data | 对象 | 可省 | 返回数据 |
| chksum | 数字 | 可省 | 采用CRC16校验，对数据内容进行校验 |

## 4.3、注意事项

💡 消息以换行符结束

## 4.4、示例

### 4.4.1、请求示例

{"NetConfig": {"ssid":"wifi名称", "passwd":"wifi密码", "iv": 1234}, "chksum":xxxx}

### 4.4.2、成功响应示例 - 标准响应

{"NetConfig":{}, "res": 0}

### 4.4.3、错误响应示例

{"NetConfig":{"errcode":"Verification failed"}, "res": -1, "chksum":xxxx}

### 4.4.4、响应超时处理

当请求发出3秒内未收到响应，可以再次发送请求，最多重复3次结束

## 4.5、消息加密方案（可选）

数据打包加密过程：数据 -> 计算chksum -> json包 ->【使用内部32位key对json整包异或加密 -> base64转码 -> 字符串数据】-> 添加换行符 -> 发送

数据取解密过程：以换行符取包 ->【字符串数据 -> base64解码 -> 使用内部32位key对字符串数据进行异或解密】 -> json包 -> 校验chksum -> 数据

注：内部key[] = {0x12, 0x34, 0x56, 0x78}

## **4.6、CRC16校验**

static const uint8\_t aucCRCHi[] = {  
 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  
};  
  
static const uint8\_t aucCRCLo[] = {  
 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  
};  
  
uint16\_t calc\_crc16( uint8\_t \* pucFrame, uint16\_t usLen )  
{  
 uint8\_t ucCRCHi = 0x99;  
 uint8\_t ucCRCLo = 0x66;  
 int iIndex;  
   
 while( usLen-- ) {  
 iIndex = ucCRCLo ^ \*( pucFrame++ );  
 ucCRCLo = ( uint8\_t )( ucCRCHi ^ aucCRCHi[iIndex] );  
 ucCRCHi = aucCRCLo[iIndex];  
 }  
 return ( uint16\_t )( ucCRCHi << 8 | ucCRCLo );  
}

# 5、返回值与错误信息

## 5.1、状态

|  |  |
| --- | --- |
| **res** | **说明** |
| 0 | 成功 |
| -1 | 失败 |
| >0 | 作为数据返回，根据具体命令定义 |
| <-1 | 作为错误反馈，根据具体命令定义 |

## 5.2、错误码

|  |  |
| --- | --- |
| **errcode** | **说明** |
| "System error" | 系统错误 |
| "Unknown message" | 未知消息 |
| "Verification failed" | 校验失败 |

# 6、命令列表

## **6.1、配网命令（蓝牙/TCP）**

请求（app -> 模组）：

{"NetConfig":{"mode":"ap", "ssid":"wifi名称", "passwd":"wifi密码","id":xxxx,"aeskey":"xxxx","aesiv":"xxxx","encryptkey":"xxxxx","url":"xxxx","token":"xxxx","timestamp":"xxxx","randdata":"xxxx"}, "chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| mode | 字符串 | “sta”，station模式  “ap”，ap模式 |
| ssid | 字符串 | 热点名字，使用AES - CBC模式加密 |
| passwd | 字符串 | 热点密码，使用AES - CBC模式加密 |
| id | 8位整数数组，长度16 | AES加密向量IV，随机数组 |
| aeskey | 字符串,长度16 | AES加密key |
| aesiv | 字符串,长度16 | AES加密向量IV |
| encryptkey | 字符串 | 服务器解析密钥 |
| url | 字符串 | 请求证书的链接 |
| token | 字符串 | 登录凭证 |
| timestamp | 整数 | 随机数 |
| randdata | 整数 | 随机数 |
| AES加密说明：采用128位，16字节对齐非填充方式  key[] = {0x02, 0x46, 0xb9, 0xc0, 0x0a, 0xf1, 0xee, 0x10, 0xfa, 0x0c, 0xd0, 0x0d, 0x48, 0x00, 0x88, 0x20} | | |

响应：

{"NetConfig":{}, "res": 0}

主动上报：

{"NetConfigReport":{}, "res": 0, "chksum":xxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | 整数 | 0：默认无错误。  1：wifi密码错误。  2：搜索不到wifi名字。  3：路由器无网络。  4：已在配网过程。 |

## **6.2、证书下发（蓝牙/TCP）**

### **6.2.1、合并证书下发**

请求（app -> 模组）：

{"AwsCert":{"thingName":"物品名称", "awsIotEndpoint":"终端节点", "awsCertCa":"根证书", "awsCertCrt":"物品证书", "awsCertPrivate":"私有秘钥"}, "chksum":xxxx}

响应：

{"AwsCert":{}, "res": 0}

### 6.2.2、根证书下发

请求（app -> 模组）：

{"RootCert":{"awsCertCa":"根证书"}, "chksum":xxxx}

响应：

{"RootCert":{}, "res": 0}

### 6.2.3、物品证书下发

请求（app -> 模组）：

{"ThingCert":{"thingName":"物品名称", "awsCertCrt":"物品证书"}, "chksum":xxxx}

响应：

{"ThingCert":{}, "res": 0}

### 6.2.4、证书密钥下发

请求（app -> 模组）：

{"CertSecret":{"awsIotEndpoint":"终端节点", "awsCertPrivate":"私有秘钥"}, "chksum":xxxx}

响应：

{"CertSecret":{}, "res": 0}

## **6.3、心跳信息（蓝牙**/TCP**/MQTT)**

请求（模组 -> app/服务器）：

{"Heart":{}}

响应：

{"Heart":{}, "res":0}

**注：模组上电或未收到回复时，心跳周期为1秒，当检测到心跳回复，调整为15秒**

## 6.4、查询&上报网络状态（蓝牙/TCP/MQTT)

请求（app/服务器 -> 模组）：

{"NetStat":{}}

响应：

{"NetStat":{"ble":0, "ap":0, "sta":0, "cert":0, "online":0}, "res":0, "chksum":xxxx}

主动上报：

{"NetStatReport":{"ble":0, "ap":0, "sta":0, "cert":0, "online":0}, "chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| ble | 整数 | 0，蓝牙关闭  1，蓝牙打开  2，蓝牙连接 |
| ap | 整数 | 0，热点关闭  1，热点打开  2，客户端连接 |
| sta | 整数 | 0，从站模式关闭  1，从站模式打开  2，连上热点 |
| cert | 整数 | 0，无证书  1，有证书 |
| online | 整数 | 0，未连接服务器  1，连上服务器 |

## ~~6.5、查询&上报系统状态（蓝牙/TCP/MQTT)~~

请求（app/服务器 -> 模组）：

{"SysStat":{}}

响应：

{"SysStat":{"status":0}, "res":0, "chksum":xxxx}

主动上报：

{"SysStatReport":{"status":0}, "chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| status | 整数 | 0，初始化状态  1，工作状态  2，OTA状态 |

## 6.6、获取&上报设备信息（蓝牙/TCP/MQTT)

请求 [app/cloud]：

{"DevInfo":{}}

响应：

{  
 "DevInfo": {  
 "cloudver":"1.0.0",  
 "model": "xxxx",  
 "sn": "xxxxx",  
 "socver": "x.x.x",  
 "mcuver":"x.x.x",  
 "blever":"x.x.x",  
 "ulver" :"x.x.x",  
 "charge": {  
 "version": "",  
 "hasCharge": 0  
 },  
 "wifiName": "",  
 "bleName": "",  
 "ip": "",  
 "waddr": "",  
 "hotSpot":""  
 },  
 "res": 0,  
 "chksum":xxxx  
}

上报：（仅开机上报一次）

{  
 "DevInfoReport": {  
 "cloudver":"1.0.0",  
 "model": "xxxx",  
 "sn": "xxxxx",  
 "socver": "x.x.x",  
 "mcuver":"x.x.x",  
 "blever":"x.x.x",  
 "ulver" :"x.x.x",  
 "charge": {  
 "version": "",  
 "hasCharge": 0  
 },  
 "wifiName": "",  
 "bleName": "",  
 "ip": "",  
 "waddr": "",  
 "hotSpot":""  
 },  
 "chksum":xxxx  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| cloudver | string | 默认  ;首次推送版本后 随下发版本号更新 |
| model | 字符串 | 对基站：基站型号  对机器：机器型号 |
| sn | 字符串 | 对基站：基站序列号  对机器：机器序列号 |
| socver | 字符串 | soc版本号 |
| mcuver | string | MCU Base版本号 |
| blever | string | 蓝牙版本号(默认0.0.0) |
| ulver | string | ango版本号 |
| charge/version | 字符串 | 充电钨版本号 |
| charge/hasCharge | 整型 | 0 无充电钨 1 有充电钨 |
| wifiName | 字符串 | wifi名称 |
| bleName | 字符串 | 蓝牙名称 |
| ip | 字符串 | 设备IP地址 |
| waddr | 整型 | 无线模块地址 |
| hotSpot | string | 热点名称 |

## 6.7、获取&上报运行信息（蓝牙/TCP/MQTT)

请求 [app/cloud]：

{"OpInfo":{  
}}

响应：

{  
 "OpInfo":{  
 "wifiRssi":xx,   
 "batCurrent":1.2,   
 "batVoltage":2.1  
 "batTemperature":20.0  
 "batQuantity":90   
 "status":0,  
 "wifiName":""  
 },   
 "res":0,   
 "chksum":xxxx  
}

主动上报：

{  
 "OpInfoReport":{  
 "wifiRssi":xx,   
 "batCurrent":1.2,   
 "batVoltage":2.1  
 "batTemperature":20.0  
 "batQuantity":90   
 "status":0,  
 "wifiName":""  
 },   
 "res":0,   
 "chksum":xxxx  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **值** | **说明** |
| wifi\_rssi | 整数 | -127 ~ 0 | WiFi信号强度 |
| batCurrent | float | 1.2 | 电池电流 |
| batVoltage | float | 2.1 | 电池电业 |
| batTemperature | float | 20.1 | 电池温度 |
| batQuantity | int32\_t | 90 | 电池电量 |
| status | 整数 | ~~0：初始化状态~~  1：清扫中  2：充电中  3：电池已充满  ~~4：回充中~~  5：停泊中  6：召回中  7：待机中  8：休眠中  9：报警中  10：OTA更新中  11：配网模式  ~~12: 遥控中~~  ~~13: 上浮中~~  ~~14: 下潜中~~  15: 关机 | 机器当前状态 |
| wifiName | string |  | wifi名称 |

W2 自定协议：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 值 | 说明 |
| status | 4 bits | 6: 休眠中  10: 停泊中  15: 多平台清洁模式 | 机器当前状态 |

## **6.8、解除App绑定&上报（**蓝牙/TCP/**MQTT）**

请求（app/服务器 -> 模组）：

{"FactoryRestore":{"controlLogId": 0}}

响应：

{"FactoryRestore":{"controlLogId": 0}, "res":0}

主动上报：

{"FactoryRestoreReport":{}, "chksum":xxxx}

## **6.9、设备透传指令**（蓝牙/TCP/MQTT）

**设备相关指令参考：《**[**设备AT指令集**](https://alidocs.dingtalk.com/i/nodes/7NkDwLng8ZK03NbXCbZX01DOWKMEvZBY?utm_scene=team_space)**》**

### **6.9.1、**AT指令请求透传

请求[app/cloud]：

{"Machine":{"cmd":"AT+指令=参数1,参数2,参数3...", "sn":"AABYWW00000"}, "chksum":xxxx}

注：sn内容填机器序列码，主要用于基站识别不同的机器

响应：

{"Machine":{"ack": "+OK", "sn":"AABYWW00000"}, "res":0, "chksum":xxxx}

### **6.9.2、主动上报**透传

{"MachineReport":{"report":"+指令:数据1,数据2,数据3...", "sn":"AABYWW00000"}, "chksum":xxxx}

### **6.9.3、异步消息透传**

双向（app/服务器 <-> 模组）：

{"Machine":{"msg":"数据", "sn":"AABYWW00000"}, "chksum":xxxx}

## 6.10、读取证书（蓝牙/TCP）

请求（app -> 模组）

{"ReadCert":{}}

响应：

{"ReadCert":{"thingName":"物品名称", "awsIotEndpoint":"终端节点", "awsCertCa":"根证书", "awsCertCrt":"物品证书", "awsCertPrivate":"私有秘钥"}, "res":0, "chksum":xxxx}

## **6.11、OTA升级**（蓝牙/TCP/MQTT）

### **6.11.1、URL升级**方式

请求 [app/cloud]]：

{"UrlOta":{"version":"x.x.x", "controlLogId": 0, "sites":{"target":xx, "type":xx, "subver":"x.x.x", "url":"xx", "md5":"xxxx","fileSize":100}}, "chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| version | string | 大版本号，由服务器管控，需要设备模组存NVS |
| controlLogId | int | 日志id |
| target | int | 0，模组  1，机器人 |
| type | int | 0 手动  1 自动 |
| subver | string | 子版本号 |
| url | string | 固件地址 |
| md5 | string | 固件md5校验值 |
| fileSize | int | 固件包字节大小 |

响应：

{"UrlOta":{"controlLogId": 0}, "res":0}

### ~~6.11.2、块传输升级方式~~

#### **a、**块**传输开始**

请求（app -> 模组）：

{"UpgradeStart":{"version":"x.x.x", "files":[{"target":xx, "type":xx, "subver":"x.x.x", "filename":"xx", "filesize":xxxx, "md5":"xxxx"},{"target":xx, "type":xx, "subver":"x.x.x", "filename":"xx", "filesize":xxxx, "md5":"xxxx", "timeout":xx}]}, "chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| version | 字符串 | 大版本号 |
| target | 整数 | 0，模组  1，机器人 |
| type | 整数 | 0，app提醒升级  1，设备静默升级  2，app强制升级  3，app检测升级 |
| subver | 字符串 | 子版本号 |
| filename | 字符串 | 固件名字 |
| filesize | 整数 | 固件大小 |
| md5 | 字符串 | 固件md5校验值 |
| block | 整数 | 块传输大小 |
| timeout | 整数 | 超时时间，单位秒 |

响应：

{"UpgradeStart":{}, "res":0,}

#### b、块传输中

请求（服务器 -> 模组）：

{"Upgrading":{"id":xx, "index":xx, "length":xx, "data":"xxx"}, "chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| id | 整数 | 数组索引 0、1 |
| index | 整数 | 块传输索引 |
| length | 整数 | 真实数据长度 |
| data | 字符串 | 块数据，经base64转换 |

注：打包时先对data计算chksum，再base64转码，解包时反过来

响应：

{"Upgrading":{}, "res":index}

#### c、块传输结束

请求（服务器 -> 模组）：

{"UpgradeEnd":{}}

响应：

{"UpgradeEnd":{}, "res":0}

注：收到结束命令后，模组对固件进行校验，发起本地升级

### 6.11.3、查询&上报升级状态

请求 【app/服务器】：

{"OtaStatus":{}}

响应：

{  
 "OtaStatus": {  
 "progress":xx,  
 "sn": "AABYWW00000",  
 "version": "1.0.0",  
 "state": 0,  
 "error":xx,  
 "msg":""  
 },  
 "res": 0,  
 "chksum":xxxx  
}

上报：（升级状态才上报）

{  
 "OtaStatusReport": {  
 "progress":xx,  
 "sn": "AABYWW00000",  
 "version": "1.0.0",  
 "state": 0,  
 "error":xx,  
 "msg":""  
 },  
 "chksum":xxxx  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| progress | int | 升级进度，百分比  0~100% |
| sn | string | 基站或者机器序列号 |
| version | string | ota下发请求时携带的版本号 |
| error | int | 0：默认  1：下载失败  2：安装失败  3：OTA检测失败 |
| msg | string | 失败时填充失败描述 |
| state | int | 0：空闲  1：下载中  2：安装中  3：成功  4：失败 |

### 6.11.4、上报升级结果

上报：[cloud]

{  
 "OtaResultReport": {  
 "sn": "",  
 "cloudver": "",  
 "state": 0, # 0：失败；1：成功  
 "msg":"", # 失败时填充失败描述  
 "controlLogId": 1,  
 "error":0,  
 "socver": {  
 "state": 0, # 0：失败；1：成功  
 "version": ""  
 },  
 "mcu": {  
 "state": 0, # 0：失败；1：成功  
 "version": ""  
 },  
 "led": {  
 "state": 0, # 0：失败；1：成功  
 "version": ""  
 },  
 "ango": {  
 "state": 0, # 0：失败；1：成功  
 "version": ""  
 }  
 },  
 "chksum":xxxx  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| sn | string | 设备sn |
| cloudver | string | ota下发请求时携带的版本号 |
| state | int | 整体升级结果 0：失败；1：成功 |
| msg | string | 失败时填充失败描述 |
| controlLogId | int | 默认 0，远程控制给实际值 |
| error | int | 0：默认  1：下载失败  2：安装失败  3：OTA检测失败 |
| socver/state | int | 0：失败；1：成功 |
| socver/version | string | soc当前版本号 |
| mcu/state | int | 0：失败；1：成功 |
| mcu/version | string | mcu主板当前版本号 |
| led/state | int | 0：失败；1：成功 |
| led/version | string | mcu灯板当前版本号 |
| ango/state | int | 0：失败；1：成功 |
| ango/version | string | 岸歌当前版本号 |

## **6.12、模组重启**（蓝牙/TCP/MQTT）

请求（app/服务器 -> 模组）：

{"SystemReset":{"controlLogId": 0, "type":0 // type 模组种类}}

响应：

{"SystemReset":{"controlLogId": 0}, "res":0}

## 6.13、水质传感器（蓝牙/TCP/MQTT）

### **6.13.1、读取&上报水质传感器数据[shadow]**

请求（app/服务器 -> 模组）：

{"WQS":{}}

响应：

{"WQS":{"temp":3524}，  
"res":0, "chksum":xxxx}

主动上报：

{"WQSReport":{"temp":3524}，"chksum":xxxx}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| temp | 整数 | 温度 |

### 6.13.2、水质传感器校准

请求（app/服务器 -> 模组）：

{"WQSCal":{}}

响应：

{"WQSCal":{}, "res":0}

## 6.14、读取&设置无线模块地址（蓝牙/TCP/MQTT）

### **6.14.1、设置地址**

请求（app/服务器 -> 模组）：

{"WAddrSet":{"addr":65535}, "chksum":xxxx}

响应：

{"WAddrSet":{}, "res":0}

### 6.14.2、读取地址

请求（app/服务器 -> 模组）：

{"WAddrGet":{}}

响应：

{"WAddrGet":{"addr":65535}, "res":0, "chksum":xxxx}

## 6.15、蓝牙身份验证（蓝牙）

请求（模组 -> app/遥控器）：

{"BLEID":{}}

响应：

{"BLEID":{}, "res":0}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | 整数 | <=0，为手机蓝牙  1，蓝牙遥控器 |

## 6.16、时间时区同步（蓝牙/MQTT）

请求（app/服务器 -> 模组）：

{"TimeZoneSet":{"timeZone":"UTC-8","time":"2023-06-30,18:07:00,5"}，"chksum":xxxx}

响应：

{"TimeZoneSet"：{}, "res":0}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| timeZone | 字符串 | 时区信息 |
| time | 字符串 | 时间 |

## **6.17、任务管理**

### **6.17.1 快速任务**

请求（app/服务器 -> 模组）：

{  
 "FastTask":{  
 "nameID":"F\_165485587",  
 "startTime":"2:10:5",  
 "smart":1,  
 "underWater"：0，  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| ~~operate~~ | ~~int~~ | ~~表示任务是增加还是覆盖~~  ~~0-增加~~  ~~1-覆盖~~ | ~~当任务队列中已有两个任务，只能覆盖，若不指明覆盖哪个任务或者覆盖的任务名字不正确，则随机覆盖~~ |
| ~~如果当前任务队列只有一个任务，则只能增加，不能覆盖~~ |
| ~~overrideID~~ | ~~string~~ | ~~要覆盖的任务名称~~ | ~~无论哪种类型任务(快速/计时/周期)，都可以覆盖~~ |
| nameID | string | 任务名称(唯一)  格式：F\_timestamp(按秒) | 优化项 |
| ~~isDelay~~ | ~~bool~~ | ~~是否设置开始时间~~  ~~true - 设置任务开始时间~~  ~~false - 默认立刻执行~~ |  |
| ~~startTime~~ | ~~string~~ | ~~任务开始时间，~~  ~~格式：2:10:5(两点十分5秒)~~ | ~~isDelay若为 false，可以不设置这个值~~ |
| ~~int32\_t~~ | ~~格式2：time(NULL)返回值(自纪元（通常是1970年1月1日00:00:00 UTC）以来经过的秒数)~~ |  |
| smart | int | 表示是否是智能模式  0-不是  1-是 | 如果是智能模式，则不需要填充下面的clean\_areas |
| underWater | int | 表示是否是水下模式  0-不是  1-是 | 如果是水下模式，则不需要填充下面的clean\_areas；  水下模式：池底+池壁  水下模式只针对X9机器 |
| cleanAreas | {int, int} | 清扫区域定义以及次数：  1 -水面  2- 池底  3- 池壁  4- 水线  5 - 多平台任务 |  |
| ~~isOverride~~ | ~~bool~~ | ~~是否覆盖任务~~  ~~true - 覆盖~~  ~~false - 不覆盖~~ | ~~默认为true~~ |
| ~~isWaterTreat~~ | ~~bool~~ | ~~水治理~~  ~~true-治理~~  ~~false- 不治理~~ |  |
| cleanMode | int | 1 - 变频清洁  2 - 深度清洁  3 - 标准清洁 |  |

响应：

{  
 "FastTask":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

### ~~6.17.2 计时任务(弃用)~~

~~请求（app/服务器 -> 模组）~~

~~{  
 "TimingTask":{  
 "operate":0  
 "overrideID":"xxxxx"  
 "nameID":"TimingTask\_1234",  
 "cleanTime":60,  
 "clean\_areas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 }  
}~~

|  |  |  |  |
| --- | --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |  |
| ~~operate~~ | ~~int~~ | ~~表示任务是增加还是覆盖~~  ~~0-增加~~  ~~1-覆盖~~ | ~~当任务队列中已有两个任务，只能覆盖，若不指明覆盖哪个任务或者覆盖的任务名字不正确，则随机覆盖~~ |
| ~~overrideID~~ | ~~string~~ | ~~要覆盖的任务名称~~ | ~~无论哪种类型任务(快速/计时/周期)，都可以覆盖~~ |
| ~~nameID~~ | ~~string~~ | ~~任务名称(唯一)~~  ~~格式：TimingTask\_ + （1-9999之间的随机值），~~  ~~比如：TimingTask\_1234~~ |  |
| ~~cleanTime~~ | ~~int32\_t~~ | ~~清扫时间，单位：分钟~~ |  |
| ~~clean\_areas~~ | ~~{int, int}~~ | ~~清扫区域定义以及次数：~~  ~~1 -水面~~  ~~2- 池底~~  ~~3- 池壁~~  ~~4- 水线~~ | ~~如：清扫水面1次，+池底2次+池壁2次，其值为:~~  ~~[~~  ~~{"area":1, "count":1},~~  ~~{"area":2, "count":2},~~  ~~{"area":3, "count":2}~~  ~~]~~ |
| ~~isOverride~~ | ~~bool~~ | ~~是否覆盖任务~~  ~~true - 覆盖~~  ~~false - 不覆盖~~ |  |
| ~~isWaterTreat~~ | ~~bool~~ | ~~水治理~~  ~~true-治理~~  ~~false- 不治理~~ |  |
| ~~cleanMode~~ | ~~int~~ | ~~1 - 变频清洁~~  ~~2 - 深度清洁~~  ~~3 - 标准清洁~~ |  |

~~响应：~~

~~{  
 "TimingTask":{},  
 "res":0  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| ~~res~~ | ~~int32\_t~~ | ~~结果返回~~  ~~0 - ok~~ |

### 6.17.3 每周任务

请求（app/服务器 -> 模组）

{  
 "WeekdayTask":{  
 "cancelNowTask":0,  
 "taskExecute":0,  
 "weekdayTask":[  
 {  
 "nameID":"PW\_1\_timestamp",  
 "weekday":1  
 "startTime":"2:10:5",  
 "taskExecute":0,  
 "smart":1,  
 "underWater"：0，  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 },  
 {  
 "nameID":"PW\_2\_timestamp",  
 "weekday":2  
 "startTime":"2:10:5",  
 "taskExecute":0,  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "areaCount":[],  
 "cleanMode":1   
 },  
 ...  
 ]  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | 备注 |
| cancelNowTask | int | 表示是否取消今天的周期的任务  0 - 取消  1 - 不取消，继续执行 | 机器正在运行周期任务时，修改周期任务；可以选择当前运行的周期任务是否继续，若继续则：  若在新的周期任务中，今天也有任务，则不再执行；  若不继续，则今天也有任务，且未过时，则会执行 |
| 机器未运行周期任务时，修改周期任务，这个字段不生效，都执行取消 |
| taskExecute | int | 表示周期任务是否生效  1 - 生效，即修改周期任务  非1 - 不生效,即删除所有任务 | 不生效意味着，删除机器端的周期任务；  生效意味着，会重新加载新的周期任务 |
| nameID | string | 任务名称(唯一)  格式：PW\_0\_timestamp(按秒)  比如：pw\_0\_xxxxxxxxx |  |
| weekday | int | 周几任务， 比如：  Sunday - 0  Monday -1  Tuesday - 2 |  |
| startTime | string | 任务开始时间，  格式：2:10:5(两点十分5秒)  或者  格式：23:10:05(两点十分5秒) |  |
| ~~int32\_t~~ | ~~格式2：time(NULL)返回值(自纪元（通常是1970年1月1日00:00:00 UTC）以来经过的秒数)~~ |  |
| smart | int | 表示是否是智能模式  0-不是  1-是 | 如果是智能模式，则不需要填充斜面的clean\_areas |
| underWater | int | 表示是否是水下模式  0-不是  1-是 | 如果是水下模式，则不需要填充下面的clean\_areas |
| cleanAreas | {int, int} | 清扫区域定义以及次数：  1 -水面  2- 池底  3- 池壁  4- 水线 | 如：清扫水面1次，+池底2次+池壁2次，其值为:  [  {"area":1, "count":1},  {"area":2, "count":2},  {"area":3, "count":2}  ] |
| ~~isWaterTreat~~ | ~~bool~~ | ~~水治理~~  ~~true-治理~~  ~~false- 不治理~~ |  |
| cleanMode | int | 1 - 变频清洁  2 - 深度清洁  3 - 标准清洁 |  |

响应：

{  
 "WeekdayTask":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

### 6.17.4 隔天任务

请求（app/服务器 -> 模组）

{  
 "IntervalTask":{  
 "cancelNowTask":0,  
 "taskExecute":0,  
 "nameID":"PI\_timestamp",  
 "createDate":"2024-4-3"  
 "startTime":"2:10:5",  
 "intervalDays":1,  
 "smart":1,  
 "underWater"：0，  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** |  |
| cancelNowTask | int | 表示是否取消今天的周期的任务  0 - 取消  1 - 不取消，继续执行 | 机器正在运行周期任务时，修改周期任务；可以选择当前运行的周期任务是否继续，若继续则：  若在新的周期任务中，今天也有任务，则不再执行；  若不继续，则今天也有任务，且未过时，则会执行 |
| 机器未运行周期任务时，修改周期任务，这个字段不生效，都执行取消 |
| taskExecute | int | 表示周期任务是否生效  1 - 生效，即修改周期任务  2 - 不生效,即删除所有任务 | 不生效意味着，删除机器端的周期任务；  生效意味着，会重新加载新的周期任务 |
| nameID | string | 任务名称(唯一)  格式：PI\_timestamp(按秒)  比如：PI\_xxxxxxxxx |  |
| createDate | string | 任务创建日期，  格式：2024-4-3(2024年4月3号) |  |
| startTime | string | 任务开始时间，  格式：2:10:5(两点十分5秒) |  |
| intervalDays | int32\_t | 间隔天数, 比如：  1 - 1天 |  |
| smart | int | 表示是否是智能模式  0-不是  1-是 | 如果是智能模式，则不需要填充斜面的clean\_areas |
| underWater | int | 表示是否是水下模式  0-不是  1-是workType | 如果是水下模式，则不需要填充下面的clean\_areas |
| cleanAreas | {int, int} | 清扫区域定义以及次数：  1 -水面  2- 池底  3- 池壁  4- 水线 | 如：清扫水面1次，+池底2次+池壁2次，其值为:  [  {"area":1, "count":1},  {"area":2, "count":2},  {"area":3, "count":2}  ] |
| ~~isWaterTreat~~ | ~~bool~~ | ~~水治理~~  ~~true-治理~~  ~~false- 不治理~~ |  |
| cleanMode | int | 1 - 变频清洁  2 - 深度清洁  3 - 标准清洁 |  |

响应：

{  
 "IntervalTask":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

### 6.17.5 停止任务

请求（app/服务器 -> 模组）

{  
 "StopAction":{}  
}

响应：

{  
 "StopAction":{},  
 "res":0  
}

## **6.18 清洁动作**(弃用)

请求（app/服务器 -> 模组）

~~{  
 "CleanAction": {"cleanAction":1}  
}~~

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| cleanAction | int32\_t | 清扫过程中可以执行的动作  1 - 暂停  2 - 恢复  3 - 终止任务 |

响应：

{  
 "CleanAction":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | (弃用) | **说明** |
| res | (弃用) | 结果返回  0 - ok |

上报

~~{  
 "CleanActionReport":{"cleanAction":1},  
 "res":0  
}~~

## **6.19 APP手动命令**(弃用)

请求（app/服务器 -> 模组）

{  
 "ManualAT": {"manualAT":1}  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| manualAT | int32\_t | 通过APP可以操作机器的动作  1 - 前进  2 - 后退  3 - 左拐  4 - 右拐  10 - 上浮  11 - 下潜 |

响应：

~~{  
 "ManualAt":{},  
 "res":0  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| ~~res~~ | ~~int32\_t~~ | ~~结果返回~~  ~~0 - ok~~ |

## 6.20 建图(弃用)

请求（app/服务器 -> 模组）

~~{  
 "BuildMap": {}  
}~~

响应：

~~{  
 "BuildMap":{},  
 "res":0  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| res | int32\_t | 结果返回  0 - ok |

## 6.21 风扇(弃用)

请求（app/服务器 -> 模组）

{  
 "Fan": {"isWork":true}  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| isWork | bool | 风扇开关  bool - 开  false - 关 |

响应：

{  
 "Fan":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.22 水泵(弃用)

请求（app/服务器 -> 模组）

{  
 "WaterPump": {"isWork":true}  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| isWork | bool | 水泵开关  bool - 开  false - 关 |

响应：

{  
 "WaterPump":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.23 回充(有桩)(弃用)

请求（app/服务器 -> 模组）

{  
 "BackCharge": {}  
}

响应：

{  
 "BackCharge":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.24 召回(无桩)

请求（app/服务器 -> 模组）

{  
 "CallBack": {}  
}

响应：

{  
 "CallBack":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.25 关机

请求（app/服务器 -> 模组）：

{  
 "Shutdown":{"controlLogId": 1}  
}

响应：

{  
 "Shutdown":{"controlLogId": 1},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.26 休眠(弃用)

请求（app/服务器 -> 模组）：

{  
 "ToSleep":{}  
}

响应：

{  
 "ToSleep":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.27 报警

### **6.27.1 清除报警**(弃用)

请求（app/服务器 -> 模组）：

{  
 "ClearAlarm":{}  
}

响应：

{  
 "ClearAlarm":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

### **6.27.2 获取告警列表**

渠道：Shadow

请求（app/服务器 -> 模组）：

{  
 "GetAlarm":{}  
}

响应：

{  
 "GetAlarm":{  
 "code":[1223, 1224, 2007]  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| code | int32\_t | 当前的错误码集合 |
| ts | string | 上报的毫秒级时间戳字符串 |
| res | int32\_t | 结果返回  0 - ok |
| controlLogId | int | 默认 0，远程控制给实际值 |

上报

{  
 "GetAlarmReport":{  
 "code":[1223, 1224, 2007],  
 "ts":"1727676583123"  
 },  
 "res":0  
}

{  
 "GetAlarmReport":{  
 "code":[1223, 1224, 2007],  
 "ts":"1727676583123" //ms  
 }  
}

## 6.28 获取地图(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetMap":{}  
}

响应：

{  
 "GetMap":{  
 "mapID":xx33  
 "area":23  
 "data":[  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 },  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 },  
 ...  
 ]   
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| mapID | string | 地图标识 |
| area | float32\_t | 地图面积 |
| x | float32\_t | 点X坐标 |
| y | float32\_t | 点Y坐标 |
| z | float32\_t | 点Z坐标 |
| pointID | uint64\_t | 点标识 |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetMapReport":{  
 "mapID":xx33  
 "area":23  
 "data":[  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 },  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 },  
 ...  
 ]   
 },  
 "res":0  
}

## 6.29 获取运行轨迹(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetRunLine":{}  
}

响应：

{  
 "GetRunLine":{  
 "getRunLine":[  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 }，  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 }，  
 ...  
 ]  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| x | float32\_t | 点X坐标 |
| y | float32\_t | 点Y坐标 |
| z | float32\_t | 点Z坐标 |
| pointID | uint64\_t | 点标识 |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetRunLineReport":{  
 [  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 }，  
 {  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "pointID":1  
 }，  
 ...  
 ]  
 },  
 "res":0  
}

## 6.30 设置电量阈值(弃用)

请求（app/服务器 -> 模组）：

{  
 "SetElecThreshold":{  
 "cleanThreshold":90,  
 "backChargeThreshold":20,  
 "shutdownThreshold":5  
 }  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| cleanThreshold | int32\_t | 在桩上充电时，到了任务时间，可以去工作的阈值 |
| backChargeThreshold | int32\_t | 机器电量过低需要回充的阈值 |
| shutdownThreshold | int32\_t | 机器低于这个电量就需要关机阈值 |

响应：

{  
 "SetElecThreshold":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.31 获取电量阈值(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetElecThreshold":{}  
}

响应：

{  
 "GetElecThreshold":{  
 "cleanThreshold":90,  
 "backChargeThreshold":20,  
 "shutdownThreshold":5  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| cleanThreshold | int32\_t | 在桩上充电时，到了任务时间，可以去工作的阈值 |
| backChargeThreshold | int32\_t | 机器电量过低需要回充的阈值 |
| shutdownThreshold | int32\_t | 机器低于这个电量就需要关机阈值 |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetElecThresholdReport":{  
 "cleanThreshold":90,  
 "backChargeThreshold":20,  
 "shutdownThreshold":5  
 },  
 "res":0  
}

## 6.32 音量设置(弃用)

请求（app/服务器 -> 模组）：

{  
 "SetVolume":{  
 "volume":90,  
 }  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| volume | int32\_t | 音量设置  0-100 |

响应：

{  
 "SetVolume":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

{  
 "SetVolumeReport":{  
 "volume":90,  
 },  
 "res":0  
}

## 6.33 音量获取(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetVolume":{  
 }  
}

响应：

{  
 "GetVolume":{  
 "volume":10  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| volume | int32\_t | 音量  0-100 |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetVolumeReport":{  
 "volume":10  
 },  
 "res":0  
}

## 6.34 速度设置

请求（app/服务器 -> 模组）：

{  
 "SetSpeed":{  
 "lineSpeed":0.06,  
 "angularSpeed":45,  
 "pumpSpeed": {  
 "type":1,  
 "min":800,  
 "max":1600  
 }  
 }  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| lineSpeed | float | 驱动轮速度设置，单位：m/s  0.06~0.3 |
| angularSpeed | int | 角速度, 单位： 度/s  45~120 |
| type | int | 水泵控制类型  1-转速(rpm)  2-占空比(%)  3-功率(W) |
| min | int | 类型--数值：  1 -- 800  2 -- 20  3 -- 3 |
| max | int | 类型--数值：  1 -- 1600  2 -- 55  3 -- 10 |

响应：

{  
 "SetSpeed":{},  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## 6.35 速度获取

请求（app/服务器 -> 模组）：

{  
 "GetSpeed":{}  
}

响应：

{  
 "GetSpeed":{  
 "lineSpeed":0.06,  
 "angularSpeed":45,  
 "pumpSpeed": {  
 "type":1,  
 "min":800,  
 "max":1600  
 }  
 },  
 "res":0  
}

{  
 "SpeedReport":{  
 "lineSpeed":0.06,  
 "angularSpeed":45,  
 "pumpSpeed": {  
 "type":1,  
 "min":800,  
 "max":1600  
 }  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| lineSpeed | float | 驱动轮速度设置，单位：m/s  0.06~0.2 |
| angularSpeed | int | 角速度, 单位： 度/s  0~90 |
| pumpSpeed | int | 流速(水泵速度), 单位： r/s  800~1600 |

## 6.37 获取充电状态(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetChargeState":{}  
}

响应：

{  
 "GetChargeState":{  
 "electricCurrent":1.2,  
 "voltage":2.1.  
 "temperature":20.0  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| electricCurrent | float | 电流大小，单位：mA  1.2mA |
| voltage | float | 电压大小，单位： V  2.0V |
| temperature | float | 温度， 单位：摄氏度  20.0 |
| ~~electricQuantity~~ | ~~int32\_t~~ | ~~电量多少~~  ~~90~~ |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetChargeStateReport":{  
 "electricCurrent":1.2,  
 "voltage":2.1.  
 "temperature":20.0  
 },  
 "res":0  
}

## 6.38 获取清洁记录

请求（app/服务器 -> 模组）：

{  
 "CleanRecord":{  
 }  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| date | string | 获取哪一天的清洁记录，格式 xx-xx-xx  2024-2-1 |

响应：

poolTemp{  
 "CleanRecord": {  
 "timeZone":"UTC+8",  
 "hasMapLine":true,  
 "cleanFile":"xxxxx.json",  
 "nameID":"FastTask\_1234",  
 "taskType":0,  
 "startTime"：222222222,  
 "endTime":333333333,  
 "cleanMode":1,  
 "cleanSpeed":0.25,  
 "cleanTime":60,  
 "multipleBottom":1,  
 "taskCleanArea":210.2,  
 "surfaceCleanArea":210.2,  
 "bottomCleanArea":0.0,  
 "wallCleanArea":0.0,  
 "waterLineCleanArea":0.0,  
 "avoidObstaclesCount":10,  
 "getOutCount":10,  
 "abnormalCount":10,  
 "getOutTime":10,  
 "getOutFailedCount":5,  
 "getOutSuccessCount":2,  
 "suppleCleanArea":20.2,  
 "poolArea":120.2,  
 "poolVolume":255.2,  
 "poolDepth":2.5,  
 "poolLittleDepth":1.5,  
 "poolTemp":28.2,  
 "usePower":20,  
 "poolBottomPower":10,  
 "poolWallPower":3,  
 "waterLinePower":3,  
 "surfaceWaterPower":1,  
 "cleanResult":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "errCode":[  
 {"error":1234, "count":2},  
 {"error":2345, "count":2},  
 {"error":3456, "count":2},  
 {"error":4567, "count":2}  
 ]  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** |  |
| timeZone | string | 时区  UTC-12 ~UTC+14 |  |
| hasMapLine | bool | 是否有地图轨迹  true- 有  false-没有 |  |
| cleanFile | string | 地图，清扫轨迹文件名 |  |
| nameID | string | 任务名称(唯一)  格式：FastTask\_ + （1-9999之间的随机值），  比如：FastTask\_1234 |  |
| taskType | int | 任务模式  0 - 快速任务  1 - 计时任务  2 - 周期任务  3 - 手动任务  4 - 预设任务 |  |
| startTime | int | 格式2：time(NULL)返回值(自纪元（通常是1970年1月1日00:00:00 UTC）以来经过的秒数) |  |
| endTime | int | 格式2：time(NULL)返回值(自纪元（通常是1970年1月1日00:00:00 UTC）以来经过的秒数) |  |
| cleanMode | int | 1-变频清洁  3-标准清洁  2-深度清洁 |  |
| cleanSpeed | float | 清洁速度(运行速度)  单位：M/S | **弃用** |
| cleanTime | int | 清洁时长  单位：分钟 |  |
| multipleBottom | int | 表示是否是多平台  1-不是多平台  2-是多平台 | 如果等于2，即多平台，则池底面积有缺失，不准确 |
| taskCleanArea | float | 任务总清洁面积  单位：m2 |  |
| surfaceCleanArea | float | 数显水面清洁面积  单位：m2 | **弃用** |
| bottomCleanArea | float | 数显池底清洁面积  单位：m2 |  |
| wallCleanArea | float | 数显池壁清洁面积  单位：m2 | **弃用** |
| waterLineCleanArea | float | 数显水线清洁面积  单位：m2 | **弃用** |
| avoidObstaclesCount | int | 避障次数 | **弃用** |
| getOutCount | int | 脱困次数 | **弃用** |
| abnormalCount | int | 异常次数 | **弃用** |
| getOutTime | int | 脱困时长  单位： 秒 | **弃用** |
| getOutFailedCount | int | 脱困失败次数 | **弃用** |
| getOutSuccessCount | int | 脱困成功次数 | **弃用** |
| suppleCleanArea | float | 补扫面积  单位： m2 | **弃用** |
| poolArea | float | 泳池面积 |  |
| poolVolume | float | 泳池体积 | 如果有图，有体积值，无图无体积值 |
| poolDepth | float | 泳池最深深度 | 如果无图，有最大深度，有图无最大审图 |
| poolLittleDepth | float | 泳池最浅深度 | 如果无图，有最小深度，有图无最大审图 |
| poolTemp | float | 泳池温度 |  |
| usePower | int | 耗电量 |  |
| poolBottomPower | int | 池底耗电量 |  |
| poolWallPower | int | 池壁耗电量 |  |
| waterLinePower | int | 水线耗电量 |  |
| surfaceWaterPower | int | 水面耗电量 |  |
| cleanResult | int | 清扫结果(异常终止原因)  1-完成  2-异常终止  3-低电终止  4-人为放弃  5-超时放弃 | 后续有错误码补充 |
| cleanAreas | {int, int} | 清扫区域定义以及次数：  1 -水面  2- 池底  3- 池壁  4- 水线 |  |
| errCode | {int, int} | 在清扫期间触发的错误码以及次数  error：错误码  count：次数 | **弃用** |

上报

poolTemp{  
 "CleanRecordReport": {  
 "hasMapLine":1,  
 "cleanFile":"xxxxx.json",  
 "nameID":"FastTask\_1234",  
 "taskType":0  
 "startTime"：222222222,  
 "endTime":333333333,  
 "cleanMode":1,  
 "cleanSpeed":0.25,  
 "cleanTime":60,  
 "taskCleanArea":210.2,  
 "displayCleanArea":210.2,  
 "avoidObstaclesCount":10,  
 "getOutTime":10,  
 "getOutFailedCount":5,  
 "getOutSuccessCount":2,  
 "suppleCleanArea":20.2,  
 "poolArea":120.2,  
 "poolVolume":255.2,  
 ""  
 "poolTemp":28.2,  
 "usePower":20,  
 "cleanResult":1,  
 "errCode":[  
 {"error":1234, "count":2},  
 {"error":2345, "count":2},  
 {"error":3456, "count":2},  
 {"error":4567, "count":2},  
 ]  
   
 },  
 "res":0  
}

文件格式(包含了清洁记录，轨迹，地图)

{  
 "CleanRecord": {  
 "hasMapLine":true,  
 "cleanFile":"xxxxx.json",  
 "nameID":"FastTask\_1234",  
 "taskType":0,  
 "startTime"：222222222,  
 "endTime":333333333,  
 "cleanMode":1,  
 "cleanSpeed":0.25,  
 "cleanTime":60,  
 "taskCleanArea":210.2,  
 "surfaceCleanArea":210.2,  
 "bottomCleanArea":0.0,  
 "wallCleanArea":0.0,  
 "poolArea":120.2,  
 "poolVolume":255.2,  
 "poolTemp":28.2,  
 "usePower":20,  
 "cleanResult":1  
 },  
 "Map": {  
 "map\_data": [  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2]  
 ]  
 },  
 "InWaterPoint":[2,2,2]  
 "Line":{  
 "line\_data": [  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2]  
 ]  
 },  
 "CleanLine":{  
 "clean\_line\_data": [  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2]  
 ]  
 },  
 "SuppleArea":{  
 "count":2,  
 "supple\_area\_data": [  
 [  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2]  
 ],  
 [  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2],  
 [2,2,2]  
 ]  
 ]  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** |  |
| Map | int | 地图点数据 |  |
| InWaterPoint | int | 入水点坐标 |  |
| Line | int | 清扫轮廓的点数据 |  |
| CleanLine | int | 清扫轨迹数据 |  |
| SuppleArea-count | int | 补扫区域个数 |  |
| SuppleArea-supple\_area\_data | int | 补扫区域的点数据 |  |

## 6.39 获取日志列表(弃用)

请求（app/服务器 -> 模组）：

~~{  
 "GetLogRecord":{  
 "controlLogId": 0,  
 "type":1  
 }  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| controlLogId | long | 日志ID |
| startTime | string | 获取的日志开始时间，  格式：2:10:5(两点十分5秒) |
| endTime | string | 获取的日志结束时间，  格式：8:10:5(两点十分5秒) |
| type | int | 关键词  全日志：1  soc：2  mcu：3  中间件：4  定位：5  规划：6  应用：7  工具: 8  sensor\_data:9 |

响应：

{  
 "GetLogRecord":{  
 "controlLogId": 0,  
 "logs":[  
 "xxx",  
 "xxx",  
 ...  
 ]  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| controlLogId | long | 日志ID |
| logs | array | 日志信息 |
| res | int32\_t | 结果返回  0 - ok |

## 6.40 获取语音包列表(弃用)

请求（app/服务器 -> 模组）：

~~{  
 "GetSoundPackList":{}  
}~~

~~响应：~~

~~{  
 "GetSoundPackList":{  
 [  
 {  
 "nameID":"name1",  
 "language":"chinese",  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 },  
 {  
 "nameID":"name1",  
 "language":"english",  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 },  
 ...   
 ]  
 },  
 "res":0  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| nameID | string | 语音包name |
| language | string | 语音包语言 |
| fileSize | int32\_t | 语音包大小， 单位: KB |
| url | string | 语音包 下载 url |
| md5 | string | 语音包 md5值 |
| res | int32\_t | 结果返回  0 - ok |

~~上报~~

~~{  
 "GetSoundPackListReport":{  
 [  
 {  
 "nameID":"name1",  
 "language":"chinese",  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 },  
 {  
 "nameID":"name1",  
 "language":"english",  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 },  
 ...   
 ]  
 },  
 "res":0  
}~~

## 6.41 语音包设置(弃用)

请求（app/服务器 -> 模组）：

~~{  
 "SetSoundPack":{  
 "nameID":"name1",  
 "language":"chinese",  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 }  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| nameID | string | 语音包name |
| language | string | 语音包语言 |
| fileSize | int32\_t | 语音包大小， 单位: KB |
| url | string | 语音包 下载 url |
| md5 | string | 语音包 md5值 |
| res | int32\_t | 结果返回  0 - ok |

~~响应：~~

~~{  
 "SetSoundPack":{},  
 "res":0  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| ~~res~~ | ~~int32\_t~~ | ~~结果返回~~  ~~0 - ok~~ |

## 6.42 语音包获取(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetSound":{}  
}

响应：

{  
 "GetSound":{  
 "nameID":"name1",  
 "language":"chinese"  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| nameID | string | 语音包name |
| language | string | 语音包语言 |
| fileSize | int32\_t | 语音包大小， 单位: KB |
| url | string | 语音包 下载 url |
| md5 | string | 语音包 md5值 |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetSoundReport":{  
 "nameID":"name1",  
 "language":"chinese"  
 "fileSize":123,  
 "url":"http://xxxxx",  
 "md5":"1123434xss"  
 },  
 "res":0  
}

## 6.43 当前位置获取(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetCurPose":{}  
}

响应：

{  
 "GetCurPose":{  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "yaw":2.1,  
 "pitch":3.1,  
 "roll":4.1,  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| x | float32\_t | 点X坐标 |
| y | float32\_t | 点Y坐标 |
| z | float32\_t | 点Z坐标 |
| yaw | float32\_t | 点偏航角度 |
| pitch | float32\_t | 点俯仰角度 |
| roll | float32\_t | 点翻转角度 |
| res | int32\_t | 结果返回  0 - ok |

上报

{  
 "GetCurPoseReport":{  
 "x":1.0,  
 "y":1.0,  
 "z":1.0,  
 "yaw":2.1,  
 "pitch":3.1,  
 "roll":4.1,  
 },  
 "res":0  
}

## 6.44 获取当前任务队列

渠道：Shadow

请求（app/服务器 -> 模组）：

{  
 "GetWorkMode":{}  
}

响应：

{  
 "GetWorkMode":{  
 "fastTask": {  
 "exist":0,  
 "nameID":"FastTask\_1234",  
 "startTime":"2:10:5",  
 "smart":1,  
 "underWater"：0，  
 "cleanAreas":[  
 {"area":1,"count":1, },  
 {"area":3, "count":2,}  
 ],  
 "cleanMode":1,  
 "status":1  
 },  
 "intervalTask":{  
 "exist":0,  
 "nameID":"PI\_timestamp",  
 "createDate":"2024-4-3"  
 "startTime":"2:10:5",  
 "intervalDays":1,  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1,  
 "status":2  
 },  
 "weekdayTask":{  
 "exist":0,  
 "tasks":[  
 {  
 "nameID":"PW\_1\_timestamp",  
 "weekday":1  
 "startTime":"2:10:5",  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1,  
 "status":2  
 },  
 {  
 "nameID":"PW\_2\_timestamp",  
 "weekday":2  
 "startTime":"2:10:5",  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "areaCount":[],  
 "cleanMode":1,  
 "status":2  
 },  
 ...  
 ]  
 }  
 },  
 "res":0  
}

{  
 "GetWorkMode":{  
 "working":0,  
 "workingTask":{  
 "nameID":"FastTask\_1234",  
 "workType":0,(0-快速任务,1-计时任务 2-周期任务， 3-手动任务, 4-预设任务)  
 "startTime":"2:10:5",  
 "smart":1,  
 "underWater"：0，  
 "clean\_areas":[  
 {"area":1,"count":1, "time":60},  
 {"area":3, "count":2,"time":60}  
 ]  
 },  
 "readyTasks":[  
 {  
 "nameID":"FastTask\_1234",  
 "workType":0,  
 "startTime":"2:10:5",  
 "smart":1,  
 "underWater"：0，  
 "clean\_areas":[  
 {"area":1,"count":1, "time":60},  
 {"area":3, "count":2,"time":60}  
 ]  
 },  
 ]  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| status | int | 是否正在执行清扫任务  1-是  2-没有 |  |
| exist | int | 任务是否存在  0 - 不存在  1 - 存在 |  |
| workType | int | 0-快速任务,  1-计时任务  2-周期任务，  3-手动任务,  4-预设任务 |  |
| 其余字段 |  |  | 参考6.17 |

上报

{  
 "GetWorkModeReport":{  
 "fastTask": {  
 "nameID":"FastTask\_1234",  
 "workType":0,  
 "startTime":"2:10:5",  
 "smart":1,  
 "clean\_areas":[  
 {"area":1,"count":1, "time":60},  
 {"area":3, "count":2,"time":60}  
 ],  
 "cleanMode":1  
 },  
 "intervalTask":{  
 "nameID":"PI\_timestamp",  
 "createDate":"2024-4-3"  
 "startTime":"2:10:5",  
 "intervalDays":1,  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 },  
 "weekdayTask":[  
 {  
 "nameID":"PW\_1\_timestamp",  
 "weekday":1  
 "startTime":"2:10:5",  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 },  
 {  
 "nameID":"PW\_2\_timestamp",  
 "weekday":2  
 "startTime":"2:10:5",  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "areaCount":[],  
 "cleanMode":1   
 },  
 ...  
 ]  
 }  
}

"GetWorkModeReport":{  
 "working":0,  
 "workingTask":{  
 "nameID":"FastTask\_1234",  
 "workType":0,  
 "startTime":"2:10:5",  
 "smart":1,  
 "underWater"：0，  
 "clean\_areas":[  
 {"area":1,"count":1, "time":60},  
 {"area":3, "count":2,"time":60}  
 ]  
 },  
 "readyTasks":[  
 {  
 "nameID":"FastTask\_1234",  
 "workType":0,  
 "startTime":"2:10:5",  
 "smart":1,  
 "underWater"：0，  
 "clean\_areas":[  
 {"area":1,"count":1, "time":60},  
 {"area":3, "count":2,"time":60}  
 ]  
 },  
 ]  
 }  
}

## 6.45 是否关联充电钨(弃用)

渠道：Shadow

请求（app/服务器 -> 模组）：

{  
 "HasCharge":{}  
}

响应：

{  
 "HasCharge":{  
 "charge":0  
 }  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| charge | int | 是否关联充电钨  0-没有  1-是 |  |

上报

{  
 "HasChargeReport":{  
 "charge":0  
 }  
}

## 6.46 获取/上报垃圾篓状态

渠道：Shadow

请求（app/服务器 -> 模组）：

{  
 "GetRubbishBoxStatus":{}  
}

响应：

{  
 "GetRubbishBoxStatus":{  
 "blockage":0,  
 "inPlace":1,  
 "health":0  
 }  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| blockage | int | 是否堵塞  0-不堵塞  1-堵塞 |  |
| inPlace | int | 是否在位  0-在位  1-不在位 |  |
| health | int | 健康状况  0-健康  1-中等  2-不健康 |  |

上报

{  
 "GetRubbishBoxStatusReport":{  
 "blockage":0,  
 "inPlace":1,  
 "health":0  
 }  
}

## 6.47 获取/上报入水状态

渠道：Shadow

请求（app/服务器 -> 模组）

{  
 "Inwater":{}  
}

响应：

{  
 "Inwater":{  
 "inWater":0  
 }，  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| inWater | int | 机器是否在泳池中  0-在，  1-不在 |  |

上报

{  
 "InwaterReport":{  
 "inWater":0  
 }  
}

## **6.48 lora ID规则**

ID组成：机型代码（1byte）+工厂（1byte）+生产年月（1byte）+序列号（3byte）(0x000000000000为异常)

AA:机型代码（固定）：依据SN的代码，进行查表。

|  |  |  |
| --- | --- | --- |
| **机型** | **SN机型代码** | **LORA ID机型代码** |
| X9 | x9(暂定) | 0x01 |
| X9station | 9s(暂定) | 0x02 |
|  |  |  |

B: 代工厂代码，依据SN代工厂进行查表

|  |  |  |
| --- | --- | --- |
| **工厂** | **SN代码** | **LORA ID 代码** |
| 元鸿 | Y | 0x01 |
| 欣智旺 | X | 0x02 |
| 和而泰 | H | 0x03 |
| 和而泰越南 | V | 0x04 |

生产年月：为sn年月的HEX，如：sn：301 = 0x12D

序列号：为sn序列号的HEX，如：sn ：00001 = 0x000001

## **6.49 设备文件上传S3存储**

请求URL（模组->服务器）

{  
 "uploadFileUrl":{  
 "controlLogId": 0,  
 "type":0,  
 "uuid":"1234",  
 "file":"map1-1716946233.json"  
 }  
}

URL返回 （服务器->模组）

{  
 "uploadFileUrl":{  
 "controlLogId": 0,  
 "type":0,  
 "uuid":"1234",  
 "url":"xxxx",  
 "urlExpire": 120  
 }  
}

~~上传完成上报 （模组->服务器）~~  （使用S3事件通知）

{  
 "uploadResultReport":{  
 "type":0,  
 "file":"map1-1716946233.json"  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| controlLogId | int | 日志id | type为3时存在 |
| type | int | 文件类型 | 0-地图数据  1-运动轨迹  2-清洁日志  3-设备日志  4-设备埋点 |
| uuid | string | 事件唯一ID | 设备请求URL时给服务器，服务器返回时原值下发 |
| file | string | 需要上传的文件名 |  |
| url | string | 上传文件URL |  |
| urlExpire | int | url过期时间，单位秒 |  |

## 6.50 泳池信息(弃用)

渠道：Shadow

请求（app/服务器 -> 模组）

{  
 "PoolInfo":{}  
}

响应：

{  
 "PoolInfo":{  
 "poolArea":100.0,  
 "poolVolume":100.0  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| poolArea | float | 泳池面积 |  |
| poolVolume | float | 泳池容积 |  |

上报

{  
 "PoolInfoReport":{  
 "cleanArea":100.0,  
 "poolVolume":100.0  
 }  
}

## 6.51 获取配网模式

请求 [app/cloud]：

{  
 "InNetConfig": {}  
}

响应：

{  
 "InNetConfig": {  
 "inNetConfig": false  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| inNetConfig | 布尔型 | 机器是否处于配网模式  false - 不处于  true - 处于 |  |

## **6.52 状态更新[远程控制]**

请求【cloud】：

{  
 "FlushStatus": {"controlLogId":0}  
}

响应【cloud】：

{  
 "FlushStatus": {  
 "controlLogId": 0  
 },  
 "res": 0  
}

## 6.53 获取周期任务

渠道：Mqtt,Ble

{  
 "GetCycleWork":{  
   
}

上报

{  
 "GetCycleWork":  
 {  
 "PeriodWork":{  
 "taskExecute":0,  
 "weekdayTask":[  
 {  
 "nameID":"PW\_1\_timestamp",  
 "weekday":1  
 "startTime":"2:10:5",  
 "taskExecute":0,  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 },  
 {  
 "nameID":"PW\_2\_timestamp",  
 "weekday":2  
 "startTime":"2:10:5",  
 "taskExecute":0,  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "areaCount":[],  
 "cleanMode":1   
 },  
 ...  
 ]  
 },  
 "IntervalWork":{  
 "taskExecute":0,  
 "nameID":"PI\_timestamp",  
 "createDate":"2024-4-3"  
 "startTime":"2:10:5",  
 "intervalDays":1,  
 "smart":1,  
 "cleanAreas":[  
 {"area":1,"count":1},  
 {"area":3, "count":2}  
 ],  
 "cleanMode":1  
 }  
 }  
}

## **6.54 重置（远程控制）**

请求（app/服务器 -> 模组）：

{"Reset":{"controlLogId": 0}}

响应：

{"Reset":{"controlLogId": 0}, "res":0}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| res | int32\_t | 结果返回  0 - ok |

## **6.56 手动控制**

### **6.56.1 进入/退出手动模式**

请求（app/服务器 -> 模组）：

{"ManualControl":{"type": 0 # 0 退出手动模式 1 进入手动模式 }}

响应（ 模组 -> app/服务器）：

{"ManualControl":{}, "res":0 # -1 已在当前控制中 0 成功}

### **6.56.2 上浮/下潜 （上浮、下潜状态上报OpInfo）**

请求（app/服务器 -> 模组）：

{"AscentDiveControl":{"type": 0 # 0 下潜 1 上浮 }}

响应（ 模组 -> app/服务器）：

{"AscentDiveControl":{}, "res":0}

### **6.56.3 手动控制前后左右（1次/300ms；超时300ms未收到自动停止）**

请求（app/服务器 -> 模组）：

{"DirectionControl":{"type": 0 # 0 前 1 后 2 左 3 右 4 左前 5 左后 6 右前 7 右后 }}

响应（ 模组 -> app/服务器）：

{"DirectionControl":{}, "res":0}

### **6.56.4 清洁任务模式设置**

请求（app/服务器 -> 模组）：

{"DirectionControl":{"type": 0 # 0 前 1 后 2 左 3 右 4 左前 5 左后 6 右前 7 右后 }}

## 6.57 获取部件寿命

请求（app/服务器 -> 模组）：

{  
 "LifeTime": {}  
}

响应（模组 -> app/服务器）：

{  
 "LifeTime": {  
 "battery": 99  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| battery | int | 电池健康度  0 - 100 |  |

## 6.58 是否屏蔽按键(弃用)

请求（app/服务器 -> 模组）：

{  
 "SetLimitBtn": {  
 "limit":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetLimitBtn": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| limit | int | 机器入水3米深后，  1：屏蔽按键功能；  2：不屏蔽； | 默认：不屏蔽 |

## 6.59 查询是否屏蔽按键(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetLimitBtn": {}  
}

响应（模组 -> app/服务器）：

{  
 "GetLimitBtn": {  
 "limit":1  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| limit | int | 机器入水3米深后，  1：屏蔽按键功能；  2：不屏蔽； | 默认：不屏蔽 |

## 6.60 设置水面工作属性

请求（app/服务器 -> 模组）：

{  
 "SetSurfaceWater": {  
 "workTime":70  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetSurfaceWater": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| workTime | int | 水面工作时长；  单位：分钟 | 默认：70分钟 |

## 6.61 获取水面工作属性变更记录

版本

时间

作者

记录

V1.0

2024/03/20

万卢海

初定版本

V1.1

2024/4/1

吉安

清扫任务的topic增加回充请求

V1.2

2024/5/8

万卢海

1. 清扫任务的topic的msg增加变量，表示不同区域清扫次数;

1. 清扫任务的topicd的clean\_area数组支持FIFO

V1.3

2024/5/8

万卢海

发送清扫动作的topic增加建图/重定位的请求

V1.4

2024/5/15

万卢海

增加：

1. "/slam\_navi\_action\_service"

1. "/slam\_navi\_action\_result\_service"

使用6和7中定义的service代替之前的topic

V1.5

2024/6/20

万卢海

修改”

1. 获取清扫相关数据

V1.6

2024/8/21

徐志勇

增加：

1. "/slam\_action\_service"增加26、27、28

1. "/navi\_action\_service"增加20、26、27

1. "/slam\_navi\_action\_result\_service"增加26、27

V1.7

2024/8/23

徐志勇

增加：

"/navi\_action\_service"增加29、125

V1.8

2024/9/6

万卢海

修改：

7.清洁强度的value修改

V1.9

2024/9/24

徐志勇

增加：

8."/slam\_navi\_action\_result\_service"的action\_result 增加33

V2.0

2024/9/29

徐志勇

增加：

8."/slam\_navi\_action\_result\_service"的action\_result 增加43、44

V2.1

2024/11/2

徐志勇

修改：

6."/slam\_action\_service"增加25，修改26、27描述

7."/navi\_action\_service"增加25，修改26、27描述

8."/slam\_navi\_action\_result\_service"增加25，修改26、27描述

V2.2

2024/11/27

万卢海

新增：

9.通过topic通知navi某些事件;

10.通过topic通知业务某些事件；

"/navi\_clean\_work"(弃用)

hj\_interface/ActionInfo.h

int32 action\_cmd # 1-水面 2-池底 3-池壁

# 11-建图 12-定位

# 21-召回 22-回充

# 31-导航状态查询 32-清洁模式

# 101-pause, 102-resume, 103-stop, 104-success, 105-fail

int32 clean\_mode # 1-变频清洁 2-标准清洁 3-深度清洁

功能

用于接受清扫任务

"/navi\_clean\_action"(弃用)

std\_msgs/Int32.h

std\_msgs::Int32

//data包含以下这些值

enum class TaskCmd {

START = 100, //开始

PAUSE =101, //暂停

RESUME=102, //继续

STOP =103, //停止

SUCCESS=104, //成功

FAIL =105, //失败

};

功能

用于接受任务执行期间的动作

描述：

a. TaskCmd::SUCCESS 是针对于计时任务定义的(即定时运行多长时间的任务), 当业务层计算运行的时间结束，会主动发送这个指令；

b. 当需要任务暂停，恢复，停止，会主动发送定义的对应指令；

c. 算法模块接收到 PAUSE/RESUME/STOP/SUCCESS 指令后，处理完具体任务，要返回消息，返回内容以及TOPIC见下一个TOPIC定义；

d. 算法模块接收到BUILD\_MAP/LOCALIZE指令后，处理完具体任务，要返回消息，返回内容以及TOPIC见下一个TOPIC定义；

"/navi\_work\_result\_return"(弃用)

hj\_interface/WorkResult.h

hj\_interface::WorkResult {

int32 cmd # 101-pause, 2-resume, 3-stop, 4-success, 5-fail

# 7-running, 8-standby, 10-build\_map, 11-localize

int32 errcode # error code - 任务失败填充这个错误码

string msg # message - 有错误码可以不填这个msg

}

功能

任务工作结果上报，或者接受业务在任务执行期间发送的动作执行后上报结果；

描述：

a. 如果任务成功结束，cmd 返回 TaskCmd::STOP或者TaskCmd::SUCCESS

b. 如果任务返回失败，cmd 返回 TaskCmd::FAIL；msg 填充失败原因的信息补充；

c. 如果在任务执行期间，接受到了 PAUSE/RESUME/STOP，执行完对应的任务动作后，

cmd分别填充： PAUSE/RESUME/STOP 信息；

d. 如果在任务执行期间，收到了 SUCCESS 命令，说明正在执行的是一个计时任务，请求结束当前任务，并返回消息，cmd 填充： SUCCESS

e. 如果是建图，cmd返回 build\_map(10)，建图成功，errcode =0， 建图失败，errcode !=0, 如果有具体原因，可以填充msg，或者告知业务层errcode与msg的映射

f. 如果是定位，cmd返回 localize，定位成功，errcode = 0，定位失败， rrcode !=0, 如果有具体原因，可以填充msg，或者告知业务层errcode与msg的映射

"/navi\_working\_state"(弃用)

hj\_interface/WorkingState.h

hj\_interface::WorkingState {

}

功能

在任务执行期间，定期返回任务的状态；

"/slam\_action\_service"

"hj\_interface/SlamAction.h"

"hj\_interface/SlamActionRequest.h"

"hj\_interface/SlamActionResponse.h"

uint8 action\_cmd # 1: slam建图 2: slam入水重定位

# 11: 水面任务 13: 池壁任务 14：水线任务 15：池底任务

200:多平台任务

# 22: 回充 25: 最近边召回 26: 入水点召回 27: 入水侧召回 28：召回重定位

# 31: slam状态请求

# 35: 获取清扫记录(清扫成功) 36：获取清扫记录(清扫失败)

# 103: slam停止建图 104: slam停止重定位

# 115: navi 停止延边 116：navi停止极限延边

# 123: 停止召回 124: 停止回充

---

uint8 result # 0: 收到

# 1: slam待机 2: slam建图中 3: slam定位中

CleanRecord CleanRecord # 清扫记录

功能

业务层根据通过这个Service通知slam模块执行具体的动作；

"/navi\_action\_service"

"hj\_interface/NaviAction.h"

"hj\_interface/NaviActionRequest.h"

"hj\_interface/NaviActionResponse.h"

uint8 action\_cmd # 1: slam建图 2: slam重定位

# 11: 清扫水面 12: 池底清扫 13: 池壁清扫 14:水线清洁

# 15：池底任务

# 16：建图/重定位延边 17：极限延边 200:多平台任务

# 18: 姿态调整

# 19: 清洁模式

# 20: 召回延边

# 21: 召回 22: 回充 25: 最近边召回 26：入水点召回

# 27: 入水侧召回 29: 停泊

# 31: slam状态请求 32: navi状态请求

# 41: 下潜 42：上浮回充

# 103: slam停止建图 104: slam停止定位

# 111: 暂停清扫 112: 继续清扫 113: 停止清扫

# 115：停止建图/重定位延边 116：停止极限延边

# 117：停止姿态调整

# 123: 停止召回 124: 停止回充 125: 停止停泊

# 下面是脚本模式

# 222: 脚本停止

# 211: 防跌落脚本(高台)

# 212: 水线横移脚本

# 213: 上下浮脚本

uint8 clean\_mode # 1: 变频清洁 2:深度清洁 3: 标准清洁

bool has\_map # true:有 false:没有

bool right\_angle # true:是-整个任务包含池底和池壁 false:否-池底池壁不一起出现在任务中

uint8 stop\_reason # 1: 出水停止， 2：其他原因停止

---

uint8 result # 0: 收到

# 11: navi待机 12: navi清扫中

# 16：navi建图/重定位延边中 17：navi极限延边中

# 18：navi姿态调整中

# 21: navi召回中 22: navi回充中

CleanRecord CleanRecord # 清扫记录

功能

业务层根据通过这个Service通navi模块执行具体的动作；

待讨论：

池底清洁单纯指弓字形清洁；

"/slam\_navi\_action\_result\_service"

8.1 service相关的srv文件

"hj\_interface/SlamNaviWorkResult.h"

"hj\_interface/SlamNaviWorkResultRequest.h"

"hj\_interface/SlamNaviWorkResultResponse.h"

"hj\_interface/CleanRecord.h"

uint8 action\_cmd # 1: 建图 2: 重定位

# 11: 清扫结束 15：池底任务流程再走一遍(针对多平台)

# 16：navi建图/重定位延边 17：navi极限延边

# 18: 姿态调整

# 21: 召回 22: 回充 25: 最近边召回 26: 入水点召回

# 27: 入水侧召回 29：停泊

# 41：下潜 42： 上浮

uint8 action\_result # 1: slam建图成功 2: slam建图失败

# 11: slam重定位成功 12: slam重定位失败

# 21: navi清扫成功 22: navi清扫失败

# 26：建图/重定位延边停止 27：极限延边停止

# 31：navi姿态调整成功 32：navi姿态调整失败

# 33: navi 有图转无图召回

# 41: navi召回成功 42: navi召回失败

# 43: navi停泊成功 44: navi停泊失败

# 51: navi回充成功 52: navi回充失败

# 61：下潜成功 62: 下潜失败

# 63：上浮成功 64: 上浮失败

CleanRecord CleanRecord # 清扫记录

---

uint8 result # 0: 收到

8.1.1 功能

slam和navi模块把具体任务执行后的结果通知业务层

PS：失败的具体的错误码待补充

8.2 清洁记录的msg定义

float32 clean\_speed # 清洁速度 单位：m/s

float32 surface\_clean\_area # 清洁水面面积 单位：m2

float32 bottom\_clean\_area # 清洁池底面积

float32 wall\_clean\_area # 清洁池壁面积

float32 water\_line\_clean\_area # 清洁水线面积

float32 pool\_area # 泳池面积

float32 pool\_volume # 泳池体积

float32 pool\_depth # 泳池最大深度

float32 pool\_little\_depth # 泳池最小深度

int32 pool\_shape # 泳池形状

string map\_line\_file\_path # 地图轨迹文件路径

int32 avoidObstaclesCount # 避障次数

int32 getOutCount # 脱困次数

int32 multiple\_bottom # 多层池底 1-单层 2-多层

8.3 地图数据共享内存使用

const int MAX\_POINT = 10000+10;

//地图数据

struct MapData {

float x;

float y;

float z;

};

// 地图数据的数组，单位是mm

MapData map\_data[MAX\_POINT];

std::string map\_data\_shm = "map\_data\_shm"; //存放地图数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(map\_data\_shm.c\_str(), map\_data);

/\*

\* map\_data的说明

\* 1. 地图有效数据，从 map\_data[10] 开始

\* 2. map\_data[0].x ---> 存放地图数据一共有多少点，范围：0~10000

\*/

/\*

\* map\_data[0].x // 点云大小

\* map\_data[0].y // 入水点成功失败标志

\*

\* map\_data[1].x // 入水点

\* map\_data[1].y

\* map\_data[1].z

map\_data[2].x //底面积， 除以10之后变成平方米

\*

\*

\*/

8.4 清扫轨迹共享内存使用

const int MAX\_POINT = 10000+10;

//清扫轨迹数据 单位是mm

struct CleanTraceData {

float x;

float y;

float z;

};

CleanTraceData clean\_trace\_data[MAX\_POINT]; //清扫轨迹的数组

std::string clean\_trace\_data\_shm = "clean\_trace\_data\_shm"; //存放清扫轨迹数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(clean\_trace\_data\_shm.c\_str(), clean\_trace\_data);

/\*

\* clean\_trace\_data的说明

\* clean\_trace\_data[0].x ---> 存放清扫轨迹的点数，沿边

\* clean\_trace\_data[1].x ---> 存放清扫轨迹的点数，单边弓

\* clean\_trace\_data[2].x ---> 存放补扫面积1的点数，目前没有补扫

\* clean\_trace\_data[3].x ---> 存放补扫面积2的点数

\* ...

\* clean\_trace\_data[n].x ---> 20000作为补扫区域结束的标志位

\*clean\_trace\_data[10] ---> 开始存放清扫轨迹有效的数据

\*/

8.5 预规划路径共享内存使用

const int MAX\_POINT = 10000+10;

//清扫轨迹数据 单位是mm

struct PlanLineData {

float x;

float y;

float z;

};

PlanLineData plan\_line\_data[MAX\_POINT]; //预规划路径的数组

std::string plan\_line\_data\_shm = "plan\_line\_data\_shm"; //存放预规划路径数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(plan\_line\_data\_shm.c\_str(), plan\_line\_data);

/\*

\* clean\_trace\_data的说明

\* clean\_trace\_data[0].x ---> 存放预规划路径的点数

\* clean\_trace\_data[10] ---> 开始存放预规划路径有效的数据

\*/

8.6 地图路径

地图存放路径：/userdata/hj/maps/

地图名称：point\_cloud.bin

9."/to\_navi\_slam"

9.1 描述：

如果有持续的某些事件要通知navi或者slam，无需回复，则通过此 topic 通知

9.2 topic描述：

#include <std\_msgs/String.h>

{

"KEY":{

"key1":"value1",

"key2":"value2"

}

}

说明：字符串转化成json，再解析json，获取响应内容

9.3设置池底遍历方式

{

"bottomRunMode" {

"mode":1

}

}

功能

字段

对象中字段

类型

描述

设置池底遍历方式

bottomRunMode

mode

int

1 - 十字弓

2 - 有图单弓，无图十字弓；

9.4 机器是否在充电

{

"isCharge" {

"state":1

}

}

功能

字段

对象中字段

类型

描述

机器是否在充电

isCharge

state

int

1 - 充电；

2 - 未充电；

9.4 业务对水泵控制传达

{

"ctlPump" {

"cmd":1

}

}

功能

字段

对象中字段

类型

描述

业务是否需要控制水泵

ctlPump

cmd

int

1 - 需要控制；

2 - 结束控制；

9.5 浅水区屏蔽与否

{

"littleWater" {

"cmd":1

}

}

功能

字段

对象中字段

类型

描述

是否屏蔽浅水区

littleWater

cmd

int

1 - 屏蔽；

2 - 不屏蔽；

默认值2：不屏蔽

10."/from\_navi\_slam"

10.1 描述：

navi或者slam有消息要通知业务，且无需回复，通过此topic传达

10.2 topic描述：

#include <std\_msgs/String.h>

{

"KEY":{

"key1":"value1",

"key2":"value2"

}

}

说明：字符串转化成json，再解析json，获取响应内容

10.3 告知机器在池底

{

"isBottom" {

"state":1

}

}

功能

字段

对象中字段

类型

描述

机器是否在池底

isBottom

state

int

1 - 在池底；

2 - 不在池底

变更记录

版本

时间

作者

记录

V1.0

2024/03/20

万卢海

初定版本

V1.1

2024/4/1

吉安

清扫任务的topic增加回充请求

V1.2

2024/5/8

万卢海

1. 清扫任务的topic的msg增加变量，表示不同区域清扫次数;

1. 清扫任务的topicd的clean\_area数组支持FIFO

V1.3

2024/5/8

万卢海

发送清扫动作的topic增加建图/重定位的请求

V1.4

2024/5/15

万卢海

增加：

1. "/slam\_navi\_action\_service"

1. "/slam\_navi\_action\_result\_service"

使用6和7中定义的service代替之前的topic

V1.5

2024/6/20

万卢海

修改”

1. 获取清扫相关数据

V1.6

2024/8/21

徐志勇

增加：

1. "/slam\_action\_service"增加26、27、28

1. "/navi\_action\_service"增加20、26、27

1. "/slam\_navi\_action\_result\_service"增加26、27

V1.7

2024/8/23

徐志勇

增加：

"/navi\_action\_service"增加29、125

V1.8

2024/9/6

万卢海

修改：

7.清洁强度的value修改

V1.9

2024/9/24

徐志勇

增加：

8."/slam\_navi\_action\_result\_service"的action\_result 增加33

V2.0

2024/9/29

徐志勇

增加：

8."/slam\_navi\_action\_result\_service"的action\_result 增加43、44

V2.1

2024/11/2

徐志勇

修改：

6."/slam\_action\_service"增加25，修改26、27描述

7."/navi\_action\_service"增加25，修改26、27描述

8."/slam\_navi\_action\_result\_service"增加25，修改26、27描述

V2.2

2024/11/27

万卢海

新增：

9.通过topic通知navi某些事件;

10.通过topic通知业务某些事件；

"/navi\_clean\_work"(弃用)

hj\_interface/ActionInfo.h

int32 action\_cmd # 1-水面 2-池底 3-池壁

# 11-建图 12-定位

# 21-召回 22-回充

# 31-导航状态查询 32-清洁模式

# 101-pause, 102-resume, 103-stop, 104-success, 105-fail

int32 clean\_mode # 1-变频清洁 2-标准清洁 3-深度清洁

功能

用于接受清扫任务

"/navi\_clean\_action"(弃用)

std\_msgs/Int32.h

std\_msgs::Int32

//data包含以下这些值

enum class TaskCmd {

START = 100, //开始

PAUSE =101, //暂停

RESUME=102, //继续

STOP =103, //停止

SUCCESS=104, //成功

FAIL =105, //失败

};

功能

用于接受任务执行期间的动作

描述：

a. TaskCmd::SUCCESS 是针对于计时任务定义的(即定时运行多长时间的任务), 当业务层计算运行的时间结束，会主动发送这个指令；

b. 当需要任务暂停，恢复，停止，会主动发送定义的对应指令；

c. 算法模块接收到 PAUSE/RESUME/STOP/SUCCESS 指令后，处理完具体任务，要返回消息，返回内容以及TOPIC见下一个TOPIC定义；

d. 算法模块接收到BUILD\_MAP/LOCALIZE指令后，处理完具体任务，要返回消息，返回内容以及TOPIC见下一个TOPIC定义；

"/navi\_work\_result\_return"(弃用)

hj\_interface/WorkResult.h

hj\_interface::WorkResult {

int32 cmd # 101-pause, 2-resume, 3-stop, 4-success, 5-fail

# 7-running, 8-standby, 10-build\_map, 11-localize

int32 errcode # error code - 任务失败填充这个错误码

string msg # message - 有错误码可以不填这个msg

}

功能

任务工作结果上报，或者接受业务在任务执行期间发送的动作执行后上报结果；

描述：

a. 如果任务成功结束，cmd 返回 TaskCmd::STOP或者TaskCmd::SUCCESS

b. 如果任务返回失败，cmd 返回 TaskCmd::FAIL；msg 填充失败原因的信息补充；

c. 如果在任务执行期间，接受到了 PAUSE/RESUME/STOP，执行完对应的任务动作后，

cmd分别填充： PAUSE/RESUME/STOP 信息；

d. 如果在任务执行期间，收到了 SUCCESS 命令，说明正在执行的是一个计时任务，请求结束当前任务，并返回消息，cmd 填充： SUCCESS

e. 如果是建图，cmd返回 build\_map(10)，建图成功，errcode =0， 建图失败，errcode !=0, 如果有具体原因，可以填充msg，或者告知业务层errcode与msg的映射

f. 如果是定位，cmd返回 localize，定位成功，errcode = 0，定位失败， rrcode !=0, 如果有具体原因，可以填充msg，或者告知业务层errcode与msg的映射

"/navi\_working\_state"(弃用)

hj\_interface/WorkingState.h

hj\_interface::WorkingState {

}

功能

在任务执行期间，定期返回任务的状态；

"/slam\_action\_service"

"hj\_interface/SlamAction.h"

"hj\_interface/SlamActionRequest.h"

"hj\_interface/SlamActionResponse.h"

uint8 action\_cmd # 1: slam建图 2: slam入水重定位

# 11: 水面任务 13: 池壁任务 14：水线任务 15：池底任务

200:多平台任务

# 22: 回充 25: 最近边召回 26: 入水点召回 27: 入水侧召回 28：召回重定位

# 31: slam状态请求

# 35: 获取清扫记录(清扫成功) 36：获取清扫记录(清扫失败)

# 103: slam停止建图 104: slam停止重定位

# 115: navi 停止延边 116：navi停止极限延边

# 123: 停止召回 124: 停止回充

---

uint8 result # 0: 收到

# 1: slam待机 2: slam建图中 3: slam定位中

CleanRecord CleanRecord # 清扫记录

功能

业务层根据通过这个Service通知slam模块执行具体的动作；

"/navi\_action\_service"

"hj\_interface/NaviAction.h"

"hj\_interface/NaviActionRequest.h"

"hj\_interface/NaviActionResponse.h"

uint8 action\_cmd # 1: slam建图 2: slam重定位

# 11: 清扫水面 12: 池底清扫 13: 池壁清扫 14:水线清洁

# 15：池底任务

# 16：建图/重定位延边 17：极限延边 200:多平台任务

# 18: 姿态调整

# 19: 清洁模式

# 20: 召回延边

# 21: 召回 22: 回充 25: 最近边召回 26：入水点召回

# 27: 入水侧召回 29: 停泊

# 31: slam状态请求 32: navi状态请求

# 41: 下潜 42：上浮回充

# 103: slam停止建图 104: slam停止定位

# 111: 暂停清扫 112: 继续清扫 113: 停止清扫

# 115：停止建图/重定位延边 116：停止极限延边

# 117：停止姿态调整

# 123: 停止召回 124: 停止回充 125: 停止停泊

# 下面是脚本模式

# 222: 脚本停止

# 211: 防跌落脚本(高台)

# 212: 水线横移脚本

# 213: 上下浮脚本

uint8 clean\_mode # 1: 变频清洁 2:深度清洁 3: 标准清洁

bool has\_map # true:有 false:没有

bool right\_angle # true:是-整个任务包含池底和池壁 false:否-池底池壁不一起出现在任务中

uint8 stop\_reason # 1: 出水停止， 2：其他原因停止

---

uint8 result # 0: 收到

# 11: navi待机 12: navi清扫中

# 16：navi建图/重定位延边中 17：navi极限延边中

# 18：navi姿态调整中

# 21: navi召回中 22: navi回充中

CleanRecord CleanRecord # 清扫记录

功能

业务层根据通过这个Service通navi模块执行具体的动作；

待讨论：

池底清洁单纯指弓字形清洁；

"/slam\_navi\_action\_result\_service"

8.1 service相关的srv文件

"hj\_interface/SlamNaviWorkResult.h"

"hj\_interface/SlamNaviWorkResultRequest.h"

"hj\_interface/SlamNaviWorkResultResponse.h"

"hj\_interface/CleanRecord.h"

uint8 action\_cmd # 1: 建图 2: 重定位

# 11: 清扫结束 15：池底任务流程再走一遍(针对多平台)

# 16：navi建图/重定位延边 17：navi极限延边

# 18: 姿态调整

# 21: 召回 22: 回充 25: 最近边召回 26: 入水点召回

# 27: 入水侧召回 29：停泊

# 41：下潜 42： 上浮

uint8 action\_result # 1: slam建图成功 2: slam建图失败

# 11: slam重定位成功 12: slam重定位失败

# 21: navi清扫成功 22: navi清扫失败

# 26：建图/重定位延边停止 27：极限延边停止

# 31：navi姿态调整成功 32：navi姿态调整失败

# 33: navi 有图转无图召回

# 41: navi召回成功 42: navi召回失败

# 43: navi停泊成功 44: navi停泊失败

# 51: navi回充成功 52: navi回充失败

# 61：下潜成功 62: 下潜失败

# 63：上浮成功 64: 上浮失败

CleanRecord CleanRecord # 清扫记录

---

uint8 result # 0: 收到

8.1.1 功能

slam和navi模块把具体任务执行后的结果通知业务层

PS：失败的具体的错误码待补充

8.2 清洁记录的msg定义

float32 clean\_speed # 清洁速度 单位：m/s

float32 surface\_clean\_area # 清洁水面面积 单位：m2

float32 bottom\_clean\_area # 清洁池底面积

float32 wall\_clean\_area # 清洁池壁面积

float32 water\_line\_clean\_area # 清洁水线面积

float32 pool\_area # 泳池面积

float32 pool\_volume # 泳池体积

float32 pool\_depth # 泳池最大深度

float32 pool\_little\_depth # 泳池最小深度

int32 pool\_shape # 泳池形状

string map\_line\_file\_path # 地图轨迹文件路径

int32 avoidObstaclesCount # 避障次数

int32 getOutCount # 脱困次数

int32 multiple\_bottom # 多层池底 1-单层 2-多层

8.3 地图数据共享内存使用

const int MAX\_POINT = 10000+10;

//地图数据

struct MapData {

float x;

float y;

float z;

};

// 地图数据的数组，单位是mm

MapData map\_data[MAX\_POINT];

std::string map\_data\_shm = "map\_data\_shm"; //存放地图数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(map\_data\_shm.c\_str(), map\_data);

/\*

\* map\_data的说明

\* 1. 地图有效数据，从 map\_data[10] 开始

\* 2. map\_data[0].x ---> 存放地图数据一共有多少点，范围：0~10000

\*/

/\*

\* map\_data[0].x // 点云大小

\* map\_data[0].y // 入水点成功失败标志

\*

\* map\_data[1].x // 入水点

\* map\_data[1].y

\* map\_data[1].z

map\_data[2].x //底面积， 除以10之后变成平方米

\*

\*

\*/

8.4 清扫轨迹共享内存使用

const int MAX\_POINT = 10000+10;

//清扫轨迹数据 单位是mm

struct CleanTraceData {

float x;

float y;

float z;

};

CleanTraceData clean\_trace\_data[MAX\_POINT]; //清扫轨迹的数组

std::string clean\_trace\_data\_shm = "clean\_trace\_data\_shm"; //存放清扫轨迹数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(clean\_trace\_data\_shm.c\_str(), clean\_trace\_data);

/\*

\* clean\_trace\_data的说明

\* clean\_trace\_data[0].x ---> 存放清扫轨迹的点数，沿边

\* clean\_trace\_data[1].x ---> 存放清扫轨迹的点数，单边弓

\* clean\_trace\_data[2].x ---> 存放补扫面积1的点数，目前没有补扫

\* clean\_trace\_data[3].x ---> 存放补扫面积2的点数

\* ...

\* clean\_trace\_data[n].x ---> 20000作为补扫区域结束的标志位

\*clean\_trace\_data[10] ---> 开始存放清扫轨迹有效的数据

\*/

8.5 预规划路径共享内存使用

const int MAX\_POINT = 10000+10;

//清扫轨迹数据 单位是mm

struct PlanLineData {

float x;

float y;

float z;

};

PlanLineData plan\_line\_data[MAX\_POINT]; //预规划路径的数组

std::string plan\_line\_data\_shm = "plan\_line\_data\_shm"; //存放预规划路径数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(plan\_line\_data\_shm.c\_str(), plan\_line\_data);

/\*

\* clean\_trace\_data的说明

\* clean\_trace\_data[0].x ---> 存放预规划路径的点数

\* clean\_trace\_data[10] ---> 开始存放预规划路径有效的数据

\*/

8.6 地图路径

地图存放路径：/userdata/hj/maps/

地图名称：point\_cloud.bin

9."/to\_navi\_slam"

9.1 描述：

如果有持续的某些事件要通知navi或者slam，无需回复，则通过此 topic 通知

9.2 topic描述：

#include <std\_msgs/String.h>

{

"KEY":{

"key1":"value1",

"key2":"value2"

}

}

说明：字符串转化成json，再解析json，获取响应内容

9.3设置池底遍历方式

{

"bottomRunMode" {

"mode":1

}

}

功能

字段

对象中字段

类型

描述

设置池底遍历方式

bottomRunMode

mode

int

1 - 十字弓

2 - 有图单弓，无图十字弓；

9.4 机器是否在充电

{

"isCharge" {

"state":1

}

}

功能

字段

对象中字段

类型

描述

机器是否在充电

isCharge

state

int

1 - 充电；

2 - 未充电；

9.4 业务对水泵控制传达

{

"ctlPump" {

"cmd":1

}

}

功能

字段

对象中字段

类型

描述

业务是否需要控制水泵

ctlPump

cmd

int

1 - 需要控制；

2 - 结束控制；

9.5 浅水区屏蔽与否

{

"littleWater" {

"cmd":1

}

}

功能

字段

对象中字段

类型

描述

是否屏蔽浅水区

littleWater

cmd

int

1 - 屏蔽；

2 - 不屏蔽；

默认值2：不屏蔽

10."/from\_navi\_slam"

10.1 描述：

navi或者slam有消息要通知业务，且无需回复，通过此topic传达

10.2 topic描述：

#include <std\_msgs/String.h>

{

"KEY":{

"key1":"value1",

"key2":"value2"

}

}

说明：字符串转化成json，再解析json，获取响应内容

10.3 告知机器在池底

{

"isBottom" {

"state":1

}

}

功能

字段

对象中字段

类型

描述

机器是否在池底

isBottom

state

int

1 - 在池底；

2 - 不在池底

请求（app/服务器 -> 模组）：

{  
 "GetSurfaceWater": {  
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetSurfaceWater": {  
 workTime":70  
 },  
 "res":0  
}

上报

{  
 "SurfaceWaterReport": {  
 workTime":70  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| workTime | int | 水面工作时长；  单位：分钟 | 默认：70分钟 |

## 6.62 设置池底遍历模式变更记录

版本

时间

作者

记录

V1.0

2024/03/20

万卢海

初定版本

V1.1

2024/4/1

吉安

清扫任务的topic增加回充请求

V1.2

2024/5/8

万卢海

1. 清扫任务的topic的msg增加变量，表示不同区域清扫次数;

1. 清扫任务的topicd的clean\_area数组支持FIFO

V1.3

2024/5/8

万卢海

发送清扫动作的topic增加建图/重定位的请求

V1.4

2024/5/15

万卢海

增加：

1. "/slam\_navi\_action\_service"

1. "/slam\_navi\_action\_result\_service"

使用6和7中定义的service代替之前的topic

V1.5

2024/6/20

万卢海

修改”

1. 获取清扫相关数据

V1.6

2024/8/21

徐志勇

增加：

1. "/slam\_action\_service"增加26、27、28

1. "/navi\_action\_service"增加20、26、27

1. "/slam\_navi\_action\_result\_service"增加26、27

V1.7

2024/8/23

徐志勇

增加：

"/navi\_action\_service"增加29、125

V1.8

2024/9/6

万卢海

修改：

7.清洁强度的value修改

V1.9

2024/9/24

徐志勇

增加：

8."/slam\_navi\_action\_result\_service"的action\_result 增加33

V2.0

2024/9/29

徐志勇

增加：

8."/slam\_navi\_action\_result\_service"的action\_result 增加43、44

V2.1

2024/11/2

徐志勇

修改：

6."/slam\_action\_service"增加25，修改26、27描述

7."/navi\_action\_service"增加25，修改26、27描述

8."/slam\_navi\_action\_result\_service"增加25，修改26、27描述

V2.2

2024/11/27

万卢海

新增：

9.通过topic通知navi某些事件;

10.通过topic通知业务某些事件；

"/navi\_clean\_work"(弃用)

hj\_interface/ActionInfo.h

int32 action\_cmd # 1-水面 2-池底 3-池壁

# 11-建图 12-定位

# 21-召回 22-回充

# 31-导航状态查询 32-清洁模式

# 101-pause, 102-resume, 103-stop, 104-success, 105-fail

int32 clean\_mode # 1-变频清洁 2-标准清洁 3-深度清洁

功能

用于接受清扫任务

"/navi\_clean\_action"(弃用)

std\_msgs/Int32.h

std\_msgs::Int32

//data包含以下这些值

enum class TaskCmd {

START = 100, //开始

PAUSE =101, //暂停

RESUME=102, //继续

STOP =103, //停止

SUCCESS=104, //成功

FAIL =105, //失败

};

功能

用于接受任务执行期间的动作

描述：

a. TaskCmd::SUCCESS 是针对于计时任务定义的(即定时运行多长时间的任务), 当业务层计算运行的时间结束，会主动发送这个指令；

b. 当需要任务暂停，恢复，停止，会主动发送定义的对应指令；

c. 算法模块接收到 PAUSE/RESUME/STOP/SUCCESS 指令后，处理完具体任务，要返回消息，返回内容以及TOPIC见下一个TOPIC定义；

d. 算法模块接收到BUILD\_MAP/LOCALIZE指令后，处理完具体任务，要返回消息，返回内容以及TOPIC见下一个TOPIC定义；

"/navi\_work\_result\_return"(弃用)

hj\_interface/WorkResult.h

hj\_interface::WorkResult {

int32 cmd # 101-pause, 2-resume, 3-stop, 4-success, 5-fail

# 7-running, 8-standby, 10-build\_map, 11-localize

int32 errcode # error code - 任务失败填充这个错误码

string msg # message - 有错误码可以不填这个msg

}

功能

任务工作结果上报，或者接受业务在任务执行期间发送的动作执行后上报结果；

描述：

a. 如果任务成功结束，cmd 返回 TaskCmd::STOP或者TaskCmd::SUCCESS

b. 如果任务返回失败，cmd 返回 TaskCmd::FAIL；msg 填充失败原因的信息补充；

c. 如果在任务执行期间，接受到了 PAUSE/RESUME/STOP，执行完对应的任务动作后，

cmd分别填充： PAUSE/RESUME/STOP 信息；

d. 如果在任务执行期间，收到了 SUCCESS 命令，说明正在执行的是一个计时任务，请求结束当前任务，并返回消息，cmd 填充： SUCCESS

e. 如果是建图，cmd返回 build\_map(10)，建图成功，errcode =0， 建图失败，errcode !=0, 如果有具体原因，可以填充msg，或者告知业务层errcode与msg的映射

f. 如果是定位，cmd返回 localize，定位成功，errcode = 0，定位失败， rrcode !=0, 如果有具体原因，可以填充msg，或者告知业务层errcode与msg的映射

"/navi\_working\_state"(弃用)

hj\_interface/WorkingState.h

hj\_interface::WorkingState {

}

功能

在任务执行期间，定期返回任务的状态；

"/slam\_action\_service"

"hj\_interface/SlamAction.h"

"hj\_interface/SlamActionRequest.h"

"hj\_interface/SlamActionResponse.h"

uint8 action\_cmd # 1: slam建图 2: slam入水重定位

# 11: 水面任务 13: 池壁任务 14：水线任务 15：池底任务

200:多平台任务

# 22: 回充 25: 最近边召回 26: 入水点召回 27: 入水侧召回 28：召回重定位

# 31: slam状态请求

# 35: 获取清扫记录(清扫成功) 36：获取清扫记录(清扫失败)

# 103: slam停止建图 104: slam停止重定位

# 115: navi 停止延边 116：navi停止极限延边

# 123: 停止召回 124: 停止回充

---

uint8 result # 0: 收到

# 1: slam待机 2: slam建图中 3: slam定位中

CleanRecord CleanRecord # 清扫记录

功能

业务层根据通过这个Service通知slam模块执行具体的动作；

"/navi\_action\_service"

"hj\_interface/NaviAction.h"

"hj\_interface/NaviActionRequest.h"

"hj\_interface/NaviActionResponse.h"

uint8 action\_cmd # 1: slam建图 2: slam重定位

# 11: 清扫水面 12: 池底清扫 13: 池壁清扫 14:水线清洁

# 15：池底任务

# 16：建图/重定位延边 17：极限延边 200:多平台任务

# 18: 姿态调整

# 19: 清洁模式

# 20: 召回延边

# 21: 召回 22: 回充 25: 最近边召回 26：入水点召回

# 27: 入水侧召回 29: 停泊

# 31: slam状态请求 32: navi状态请求

# 41: 下潜 42：上浮回充

# 103: slam停止建图 104: slam停止定位

# 111: 暂停清扫 112: 继续清扫 113: 停止清扫

# 115：停止建图/重定位延边 116：停止极限延边

# 117：停止姿态调整

# 123: 停止召回 124: 停止回充 125: 停止停泊

# 下面是脚本模式

# 222: 脚本停止

# 211: 防跌落脚本(高台)

# 212: 水线横移脚本

# 213: 上下浮脚本

uint8 clean\_mode # 1: 变频清洁 2:深度清洁 3: 标准清洁

bool has\_map # true:有 false:没有

bool right\_angle # true:是-整个任务包含池底和池壁 false:否-池底池壁不一起出现在任务中

uint8 stop\_reason # 1: 出水停止， 2：其他原因停止

---

uint8 result # 0: 收到

# 11: navi待机 12: navi清扫中

# 16：navi建图/重定位延边中 17：navi极限延边中

# 18：navi姿态调整中

# 21: navi召回中 22: navi回充中

CleanRecord CleanRecord # 清扫记录

功能

业务层根据通过这个Service通navi模块执行具体的动作；

待讨论：

池底清洁单纯指弓字形清洁；

"/slam\_navi\_action\_result\_service"

8.1 service相关的srv文件

"hj\_interface/SlamNaviWorkResult.h"

"hj\_interface/SlamNaviWorkResultRequest.h"

"hj\_interface/SlamNaviWorkResultResponse.h"

"hj\_interface/CleanRecord.h"

uint8 action\_cmd # 1: 建图 2: 重定位

# 11: 清扫结束 15：池底任务流程再走一遍(针对多平台)

# 16：navi建图/重定位延边 17：navi极限延边

# 18: 姿态调整

# 21: 召回 22: 回充 25: 最近边召回 26: 入水点召回

# 27: 入水侧召回 29：停泊

# 41：下潜 42： 上浮

uint8 action\_result # 1: slam建图成功 2: slam建图失败

# 11: slam重定位成功 12: slam重定位失败

# 21: navi清扫成功 22: navi清扫失败

# 26：建图/重定位延边停止 27：极限延边停止

# 31：navi姿态调整成功 32：navi姿态调整失败

# 33: navi 有图转无图召回

# 41: navi召回成功 42: navi召回失败

# 43: navi停泊成功 44: navi停泊失败

# 51: navi回充成功 52: navi回充失败

# 61：下潜成功 62: 下潜失败

# 63：上浮成功 64: 上浮失败

CleanRecord CleanRecord # 清扫记录

---

uint8 result # 0: 收到

8.1.1 功能

slam和navi模块把具体任务执行后的结果通知业务层

PS：失败的具体的错误码待补充

8.2 清洁记录的msg定义

float32 clean\_speed # 清洁速度 单位：m/s

float32 surface\_clean\_area # 清洁水面面积 单位：m2

float32 bottom\_clean\_area # 清洁池底面积

float32 wall\_clean\_area # 清洁池壁面积

float32 water\_line\_clean\_area # 清洁水线面积

float32 pool\_area # 泳池面积

float32 pool\_volume # 泳池体积

float32 pool\_depth # 泳池最大深度

float32 pool\_little\_depth # 泳池最小深度

int32 pool\_shape # 泳池形状

string map\_line\_file\_path # 地图轨迹文件路径

int32 avoidObstaclesCount # 避障次数

int32 getOutCount # 脱困次数

int32 multiple\_bottom # 多层池底 1-单层 2-多层

8.3 地图数据共享内存使用

const int MAX\_POINT = 10000+10;

//地图数据

struct MapData {

float x;

float y;

float z;

};

// 地图数据的数组，单位是mm

MapData map\_data[MAX\_POINT];

std::string map\_data\_shm = "map\_data\_shm"; //存放地图数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(map\_data\_shm.c\_str(), map\_data);

/\*

\* map\_data的说明

\* 1. 地图有效数据，从 map\_data[10] 开始

\* 2. map\_data[0].x ---> 存放地图数据一共有多少点，范围：0~10000

\*/

/\*

\* map\_data[0].x // 点云大小

\* map\_data[0].y // 入水点成功失败标志

\*

\* map\_data[1].x // 入水点

\* map\_data[1].y

\* map\_data[1].z

map\_data[2].x //底面积， 除以10之后变成平方米

\*

\*

\*/

8.4 清扫轨迹共享内存使用

const int MAX\_POINT = 10000+10;

//清扫轨迹数据 单位是mm

struct CleanTraceData {

float x;

float y;

float z;

};

CleanTraceData clean\_trace\_data[MAX\_POINT]; //清扫轨迹的数组

std::string clean\_trace\_data\_shm = "clean\_trace\_data\_shm"; //存放清扫轨迹数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(clean\_trace\_data\_shm.c\_str(), clean\_trace\_data);

/\*

\* clean\_trace\_data的说明

\* clean\_trace\_data[0].x ---> 存放清扫轨迹的点数，沿边

\* clean\_trace\_data[1].x ---> 存放清扫轨迹的点数，单边弓

\* clean\_trace\_data[2].x ---> 存放补扫面积1的点数，目前没有补扫

\* clean\_trace\_data[3].x ---> 存放补扫面积2的点数

\* ...

\* clean\_trace\_data[n].x ---> 20000作为补扫区域结束的标志位

\*clean\_trace\_data[10] ---> 开始存放清扫轨迹有效的数据

\*/

8.5 预规划路径共享内存使用

const int MAX\_POINT = 10000+10;

//清扫轨迹数据 单位是mm

struct PlanLineData {

float x;

float y;

float z;

};

PlanLineData plan\_line\_data[MAX\_POINT]; //预规划路径的数组

std::string plan\_line\_data\_shm = "plan\_line\_data\_shm"; //存放预规划路径数据的共享内存名称

//把地图数据塞进共享内存

#include "shm\_interface.h"

hj\_bf::setVariable(plan\_line\_data\_shm.c\_str(), plan\_line\_data);

/\*

\* clean\_trace\_data的说明

\* clean\_trace\_data[0].x ---> 存放预规划路径的点数

\* clean\_trace\_data[10] ---> 开始存放预规划路径有效的数据

\*/

8.6 地图路径

地图存放路径：/userdata/hj/maps/

地图名称：point\_cloud.bin

9."/to\_navi\_slam"

9.1 描述：

如果有持续的某些事件要通知navi或者slam，无需回复，则通过此 topic 通知

9.2 topic描述：

#include <std\_msgs/String.h>

{

"KEY":{

"key1":"value1",

"key2":"value2"

}

}

说明：字符串转化成json，再解析json，获取响应内容

9.3设置池底遍历方式

{

"bottomRunMode" {

"mode":1

}

}

功能

字段

对象中字段

类型

描述

设置池底遍历方式

bottomRunMode

mode

int

1 - 十字弓

2 - 有图单弓，无图十字弓；

9.4 机器是否在充电

{

"isCharge" {

"state":1

}

}

功能

字段

对象中字段

类型

描述

机器是否在充电

isCharge

state

int

1 - 充电；

2 - 未充电；

9.4 业务对水泵控制传达

{

"ctlPump" {

"cmd":1

}

}

功能

字段

对象中字段

类型

描述

业务是否需要控制水泵

ctlPump

cmd

int

1 - 需要控制；

2 - 结束控制；

9.5 浅水区屏蔽与否

{

"littleWater" {

"cmd":1

}

}

功能

字段

对象中字段

类型

描述

是否屏蔽浅水区

littleWater

cmd

int

1 - 屏蔽；

2 - 不屏蔽；

默认值2：不屏蔽

10."/from\_navi\_slam"

10.1 描述：

navi或者slam有消息要通知业务，且无需回复，通过此topic传达

10.2 topic描述：

#include <std\_msgs/String.h>

{

"KEY":{

"key1":"value1",

"key2":"value2"

}

}

说明：字符串转化成json，再解析json，获取响应内容

10.3 告知机器在池底

{

"isBottom" {

"state":1

}

}

功能

字段

对象中字段

类型

描述

机器是否在池底

isBottom

state

int

1 - 在池底；

2 - 不在池底

请求（app/服务器 -> 模组）：

{  
 "SetBottomRunMode": {  
 "mode":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetBottomRunMode": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 池底清洁时遍历模式  1 - 纯十字弓清扫  2 - 场景自适应 | 1 - 无论是否建图成功，都十字弓清扫；  2 - 建图成功，单边弓清扫，失败，十字弓清扫  默认：十字弓清扫 |

## 6.63 获取池底遍历模式

请求（app/服务器 -> 模组）：

{  
 "GetBottomRunMode": {  
   
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetBottomRunMode": {  
 "mode":1  
 },  
 "res":0  
}

上报

{  
 "BottomRunModeReport": {  
 "mode":1  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 池底清洁时遍历模式  1 - 纯十字弓清扫  2 - 场景自适应 | 1 - 无论是否建图成功，都十字弓清扫；  2 - 建图成功，单边弓清扫，失败，十字弓清扫  默认：自适应 |

## 6.64 设置召回回到入水点

请求（app/服务器 -> 模组）：

{  
 "SetEntryPoint": {  
 "type":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetEntryPoint": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| type | int | 召回是否回到入水点  0 - 回到入水侧  1 - 回到入水点 | 0 - 召回回到入水侧  1 - 召回回到入水点 |

## 6.65 获取召回回到入水点

请求（app/服务器 -> 模组）：

{  
 "GetEntryPoint": {  
   
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetEntryPoint": {  
 "type":1  
 },  
 "res":0  
}

上报

{  
 "EntryPointReport": {  
 "type":1  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| type | int | 召回是否回到入水点  0 - 回到入水侧  1 - 回到入水点 | 0 - 召回回到入水侧  1 - 召回回到入水点 |

## 6.66 OTA中按键触发

上报（模组 -> app/服务）：

{  
 "BtnTriggerReport": {  
 "type":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| type | int | 是否按键触发了 | 在OTA过程中，按键触发了不响应，且通知app；  type字段暂无内容-占位符 |

## 6.67 设置实验功能(弃用)

请求（app/服务器 -> 模组）：

{  
 "SetExperimentFunction": {  
 "mode":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetExperimentFunction": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 是否启用实验功能  1 - 启用  2 - 不启用 | 实验功能包含了：多平台清扫，回到入水点； |

## 6.68 获取实验功能(弃用)

请求（app/服务器 -> 模组）：

{  
 "GetExperimentFunction": {  
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetExperimentFunction": {  
 "mode":1  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 是否启用实验功能  1 - 启用  2 - 不启用 | 实验功能包含了：多平台清扫，回到入水点； |

## 6.69 获取气囊状态

请求（app/服务器 -> 模组）：

{  
 "GetAirBagStatus": {  
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetAirBagStatus": {  
 "status":1  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| status | int | 气囊状态  0 - 放气完成  1 - 放气中  2 - 充气完成  3 - 充气中 |  |

上报

{  
 "AirBagStatusReport": {  
 "status":1  
 }  
}

## 6.70 恢复出厂设置

请求（app/服务器 -> 模组）：

{  
 "FactoryReset": {  
 }  
}

响应（模组 -> app/服务器）：

{  
 "FactoryReset": {  
 },  
 "res":0  
}

## 6.71 设置多平台模式

请求（app/服务器 -> 模组）：

{  
 "SetMultiBottom": {  
 "mode":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetMultiBottom": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 池底清洁时遍历模式  1 - 打开多平台模式  2 - 关闭多平台模式 | 默认：打开多平台 |

## 6.72 获取多平台模式

请求（app/服务器 -> 模组）：

{  
 "GetMultiBottom": {  
   
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetMultiBottom": {  
 "mode":1  
 },  
 "res":0  
}

上报

{  
 "MultiBottomReport": {  
 "mode":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 池底清洁时遍历模式  1 - 打开多平台模式  2 - 关闭多平台模式 | 默认：打开多平台 |

## 6.74 设置机器端ango功能

请求（app/服务器 -> 模组）：

{  
 "SetAngoEnable": {  
 "mode":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetAngoEnable": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | ango功能是否打开  1 - 打开(机器端)  2 - 屏蔽(机器端) | 默认：打开(待定) |

## 6.72 获取机器端ango功能

请求（app/服务器 -> 模组）：

{  
 "GetAngoEnable": {  
   
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetAngoEnable": {  
 "mode":1  
 },  
 "res":0  
}

上报

{  
 "AngoEnableReport": {  
 "mode":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | ango功能是否打开  1 - 打开(机器端)  2 - 屏蔽(机器端) | 默认：屏蔽 |

## 6.73 下发配置指令

请求（app/服务器 -> 模组）：

{  
 "processCmd": {  
   
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetAngoEnable": {  
 "mode":1  
 },  
 "res":0  
}

上报

{  
 "AngoEnableReport": {  
 "mode":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | ango功能是否打开  1 - 打开(机器端)  2 - 屏蔽(机器端) | 默认：屏蔽 |

## 6.74 设置浅水区策略

请求（app/服务器 -> 模组）：

{  
 "SetLittleWater": {  
 "mode":1  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetLittleWater": {  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 是否屏蔽浅水区  1 - 屏蔽；  2 - 不屏蔽； | 默认值：2-不屏蔽 |

## 6.75 获取浅水区策略

请求（app/服务器 -> 模组）：

{  
 "GetLittleWater": {  
   
 }  
}

响应（模组 -> app/服务器）：

{  
 "GetLittleWater": {  
 "mode":1  
 },  
 "res":0  
}

上报

{  
 "LittleWaterReport": {  
 "mode":1  
 }  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 是否屏蔽浅水区  1 - 屏蔽；  2 - 打开； | 默认值：2-打开 |

## 6.76 脚本模式

请求（app/服务器 -> 模组）：

{  
 "SetRunDemo": {  
 "controlLogId": 173509796132699,  
 "mode":201  
 }  
}

响应（模组 -> app/服务器）：

{  
 "SetRunDemo": {  
 "controlLogId": 173509796132699  
 },  
 "res":0  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **类型** | **说明** | **备注** |
| mode | int | 脚本运行模式  222 - 停止  201 - 变频演示  202 - 直线往返演示  203 - 矢量喷口演示  211 - 防跌落演示(高台)  212 - 水线横移演示  213 - 上下浮演示 |  |
| controlLogId | uint64 |  |  |

## 6.77 下发日志开关

请求（app/服务器 -> 模组）：

~~{  
 "switchLog":{  
 "logVersion": "debug"  
 }  
}~~

|  |  |  |
| --- | --- | --- |
| **~~字段~~** | **~~类型~~** | **~~说明~~** |
| logVersion | string | 开关，debug：打开debug日志。  release：关闭debug日志。 |

响应：

{  
 "switchLog":{  
 "logVersion": "debug"  
 },  
 "res":0  
}

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **说明** |
| logVersion | string | 开关，debug：打开debug日志。  release：关闭debug日志。 |
| res | int | 0：正确，-1：错误 |