目錄

引言 1.1

一. 伽利略导航系统HTTP协议说明 1.2

二. ROS通信协议 1.3

三. 串口通信协议 1.4

四. Modbus协议 1.5

五. UDP 协议 1.6

六. 伽利略导航系统SDK说明 1.7

- 引言
  - 整体工作流程
  - 跨地图导航
  - 跨楼层导航

引言

伽利略视觉导航系统是一个融合了多种传感器以视觉导航为主导的机器人定位和运动控制系统。通过加载本系统可以实现机器人的定位和导航功能。适用于自动巡检机器人工业AGV,服务机器人等多种应用场景。

建议在使用API之前先通过使用机器人了解整个工作原理和流程。机器人使用手册

整体工作流程

机器人工作主要分为两个部分，建图和导航。首先开启机器人建图状态后，遥控机器人在目标环境中移动。机器人会自动创建环境地图。创建地图后调用保存接口，地图保存在机器人上后可以供导航使用。 在创建的地图中添加机器人移动的路径和目标点数据。之后就可以启动导航了。

跨地图导航

在不同地图中添加目标点的关联信息后即可在不同的地图间导航。

跨楼层导航

跨楼层导航需要创建楼层地图和电梯地图。其基本原理和跨地图导航一样。通过添加连接点把楼层地图和电梯地图连接起来。机器人会自动根据目标点所在的楼层和当前位置找出应该采取的路线和动作。 注意电梯地图的0号点必须在电梯内部。

- 伽利略导航系统HTTP协议说明
  - 程序实例
  - 跨局域网调用API
  - HTTP 服务的参数配置
  - token API
  - 系统状态API
    - 获取系统当前状态
    - 获取系统基本信息
    - 获取Galileo Status
    - 获取系统日志
    - 获取机器人速度
    - 遥控机器人
    - 关闭机器人
    - 校正机器人陀螺仪
    - 开始校正摄像头角度位置参数
    - 取消摄像头角度位置校正参数
    - 获取摄像头角度位置校正状态
    - 提交摄像头角度位置校正参数
    - 开始标定雷达参数
    - 提交雷达校正参数
    - 取消校准雷达参数
    - 获取雷达校准状态
    - 检查系统更新状态
    - 开始自动更新程序
    - 取消自动更新程序
    - 获取软件更新日志
    - 获取当前调度服务器
    - 系统自检
    - 获取当前io电平情况
    - 控制当前io电平情况
    - 获取IO扩展板电平
    - 控制IO扩展板电平
    - 获取机器人当前配置参数
    - 修改机器人配置
    - 恢复机器人默认参数
    - 让机器人播放语音
    - 机器人当前是否正在播放语音
    - 停止正在播放的语音
    - 机器人的语音播放记录
    - 上传语音包(语音音频文件方式)
    - 创建或替换语音包（语音压缩包方式）
    - 替换语音文件
    - 删除语音包
    - 重置默认语音

        - 获取语音修改文本
        - 创建alias记录
        - 替换文本内容和语音
        - 迎宾模式开关
        - 获取对应话题数据
        - 获取语音模块IO电平
        - 设置语音模块输出端口电平
        - 设置语音模块输出端口PWM
        - 设置语音模块输出舵机信号
        - 检查机器人证书
        - 让机器人连接wifi
        - 是否使用预览版程序
        - 设置使用预览版
        - 是否开启跟随功能
        - 开启关闭跟随功能
        - 获取系统音量设置
        - 设置系统音量
        - 获取当前机器人是否连上互联网
        - 获取已经保存的照片文件
        - 拍照
        - 删除拍照文件
        - 获取已经保存的录像文件
        - 开始录像
        - 删除录像文件
        - 滑台控制相关API
        - 顶升机构控制API
        - onvif 监控摄像头相关API
    - 文件系统操作API
        - 获取文件
        - 创建文件
        - 文件或文件夹重命名
        - 删除文件
        - 获取文件夹内容
        - 创建文件夹
        - 删除文件夹
        - 获取文件或文件夹详细信息
        - 复制文件或文件夹
        - 批量删除文件或文件夹
        - 剪切文件或文件夹
    - 创建地图API
        - 启动创建地图
        - 结束创建地图线程
        - 更新地图
        - 当前机器人位置

        - 获取当前正在创建的地图信息
        - 保存当前创建的地图
        - 获取当前创建地图的png图片
        - 获取创建地图时机器人的运动轨迹
        - 获取机器人轨迹
    - 导航部分
        - 启动导航状态
        - 导航时开启或关闭更新地图
        - 获取是否在导航时更新地图
        - 启动调度导航
        - 停止导航
        - 停止调度导航
        - 机器人当前位置信息
        - 获取当前导航正在使用的地图和路径名称
        - 获取当前位置到目标点的距离
        - 获取已经保存的详细地图信息
        - 获取保存的地图中的机器人运动轨迹
        - 获取目标地图的PGM图片信息
        - 获取目标地图的png图片信息
        - 获取当前正在使用的地图
        - 上传当前机器人地图数据至调度服务器
        - 从调度服务器下载地图
        - 下载机器人地图
        - 上传地图至机器人
        - 开启导航任务
        - 开始自动充电
        - 停止自动充电
        - 移动到对应目标点
        - 取消当前导航任务
        - 获取导航循环任务信息
        - 开始导航循环任务
        - 停止导航循环任务
        - 获取充电桩位置
        - 保存充电桩位置
        - 动态切换地图
        - 获取导航路径点
        - 上传导航路径点
        - 删除导航路径点
        - 获取导航目标点
        - 上传导航目标点
        - 删除导航目标点
        - 获取地图连接点
        - 添加导航点连接信息
        - 删除导航点之间的连接

        - 切换当前地图
        - 获取忽略区域信息
        - 修改忽略信息
        - 删除忽略区域信息
        - 获取虚拟墙信息
        - 修改虚拟墙数据
        - 删除虚拟墙数据
        - 设置导航初始位置
        - 生成多地图导航任务（不执行）
        - 执行局部运动任务
        - 执行导航和局部运动
        - 执行导航任务，局部任务，io任务
        - 局部移动到指定位置
        - 导航到指定目标点
        - 风向传感器信息
        - 获取地图二维码信息
        - 生成覆盖路径
        - 获取电梯基本信息
        - 呼叫电梯到指定楼层
        - 控制电梯开关门
    - 任务相关API
        - 获取任务信息
        - 创建任务
        - 修改任务
        - 删除任务
        - 启动任务
        - 暂停任务
        - 继续任务
        - 取消任务
        - 循环执行任务
        - 获取动作Action信息
        - 创建动作Action信息
        - 创建action
        - 包含新创建的action的task
        - 修改动作Action信息
        - 删除动作Action信息
        - 触发等待动作
    - 自动化任务API
        - 获取自动化任务信息
        - 创建自动化任务
        - 修改自动化任务
        - 删除自动化任务
    - 行为树相关API
        - 什么是行为树

        - 当前支持的行为树控制节点
        - 创建和控制行为树
        - 黑板参数
    - Websocket相关API
        - 获取GalileoStatus
        - 获取温湿度及可燃气体数据
        - 获取光照度和空气传感器数据
    - 对于跨地图导航的说明

伽利略导航系统HTTP协议说明

程序实例

C# 版本

Python 版本

Java 版本

API的格式为API版本号加上对应的URL，以获取系统状态API为例，实际请求地址为/api/v1/system/status。API返回值都是json格式的数据。下面的文档将省略API前缀。API服务程序默认端口为3546。 机器人在启动后会向局域网发送UDP广播，端口为22002。通过此广播我们能够获取到局域网中的机器人基本信息。下面是一个具体的广播数据例子


```
{"id": "F072E1BA8162245572D2FAEEB2526C5DD916F5A0D6D0F8A14B67FA43DC501079461280B15BA5", "port": 3546, "mac": "00:e0:4c:68:6f:0f", "version": "5.0.0"}
```


广播数据包含了机器人ID，机器人Http服务端口号，机器人mac和机器人当前的http服务版本号。

对于Http调用的参数，GET和DELETE方法的参数放在URL query string里面。POST和PUT方法的参数放在body里面。

对于HTTP协议不熟悉的用户可以先参考这个文档

跨局域网调用API

通过伽利略网络代理我们可以实现远程跨局域网的机器人API调用

机器人远程代理设置网址

机器人远程调用说明

HTTP 服务的参数配置

http服务参数配置文件位于 `/home/xiaoqiang/Documents/ros/src/galileo_api/config.json`

其默认内容如下

```
{
    "username": "admin",
    "password": "admin",
    "allow_update": true,
    "no_token_check": true,
    "auto_charge": false,
}
```

参数	类型	说明
username	string	获取机器人token时的用户名。默认为admin
password	string	获取机器人token时的密码。默认为admin
allow_update	bool	是否允许自动更新。当为true时程序有更新时客户端会收到机器人更新提示。反之则不会有更新提示。
no_token_check	bool	是否开启token验证功能。默认不开启
auto_charge	bool	是否开启低电量自动充电功能，默认不开启。注意开启此功能要保证机器人导航地图中有正确的充电桩位置，同时导航地图能够正常工作

token API

为了系统的安全性，在调用机器人api的时候可以加上token验证的功能。此功能默认关闭，可以通过配置http api参数打开此功能。

URL: /token

请求方式: GET

请求参数:

参数	类型	说明
username	string	配置文件中的用户名，默认为admin
password	string	配置文件中的密码, 默认为admin

返回值

参数	类型	说明
result	bool	是否成功获取token
token	string	获取的token

获取token后在调用api时可以在url参数中加入token=xxx,xxx为你的token数据。 对于POST或PUT请求，token参数可以加在url中也可以在body的json数据里面。

下面是一个调用的例子

```
http://192.168.0.132:3546/api/v1/system/info?token=28b1c500400611ebb805493c9303c705
```

其中192.168.0.132为机器人IP，28b1c500400611ebb805493c9303c705为机器人token。

系统状态API

获取系统当前状态

URL: /system/status

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	状态String,可能的值为 Mapping, Navigating, Busy, Free

说明：系统可能处在互斥的几种状态中。通过不同的操作API系统在不同的状态间切换。只有在特定的状态下系统才能执行特定的API。

获取系统基本信息

URL: /system/info

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
battery	int	电池电量百分比
camera_rgb	int	rgb摄像头topic发布频率，0代表没有数据
camera_depth	int	深度摄像头topic发布频率，0代表没有数据
odom	int	编码器topic发布频率，0代表没有数据
imu	int	IMU topic的发布频率，0代表没有数据
driver_port	bool	急停按钮状态，true为按下，false为未按下
charge	bool	是否正在执行充电任务
loop	bool	是否正在执行循环任务
info	object	机器人基本信息,4.5.0和5.1.0之后版本才会有此返回

说明: charge状态和galileo_status中的状态并不一样。charge状态是整个充电任务的状态而galileo status中的充电状态为局部充电任务状态。

示例返回


```
{
    "battery": 100,
    "camera_rgb": 19,
    "camera_depth": 29,
    "odom": 26,
    "imu": 49,
    "camera_processed": 30,
    "driver_port": false,
    "info": {
        "version": "6.1.5",
        "code_name": "chitu-noetic",
        "id": "AE13B83EE2846276882EE47A99391C89CD2EF6B6878D5D309F80755F8E3B7D15CB0CB9BEF55
D",
        "mac": "00:15:00:b9:90:a6",
        "port": 3546
    },
    "charge": false,
    "loop": false,
    "slam_type": "camera"
}
```


注意第一次调用时，由于需要对数据统计，可能对应的数据返回为*0.*之后调用返回数据将为正常

获取Galileo Status

URL: /system/galileo_status

请求方式: GET

请求参数: 无

返回参数：当前的galileo status数据, galileo status是对系统整体状态的一个汇报参数。具体定义如下

例子


```
{
  "header": {
    "seq": 542928,
    "stamp": {
      "secs": 1624970817,
      "nsecs": 524690511
    },
    "frame_id": "map"
  },
  "navStatus": 0, // 导航服务状态，0表示没开启closed，1表示开启opened。
  "visualStatus": -1, // 视觉系统状态，-1标系视觉系统处于关闭状态，0表示没初始化uninit，1表示正在追踪
tracking,2表示丢失lost,1和2都表示视觉系统已经初始化完成。
  "mapStatus": 0, // 建图服务状态，0表示未开始建图，1表示正在建图
  "gcStatus": 0, // 内存回收标志，0表示未进行内存回收，1表示正在进行内存回收
  "gbaStatus": 0, // 闭环优化标志，0表示未进行闭环优化，1表示正在进行闭环优化
```

```
  "chargeStatus": 0, // 充电状态，0 free 未充电状态，1 charging 充电中，2 charged 已充满，但仍在小电流充电，3 finding 寻找充电桩，4 docking 停靠充电桩，5 error 错误
  "loopStatus": 0, // 是否处于自动巡检状态，1为处于，0为不处于。
  "power": 37.05362319946289, // 电源电压v。
  "targetNumID": -1, // 当前目标点编号,默认值为-1表示无效值，当正在执行无ID的任务是值为-2，比如通过Http API 创建的导航任务。
  "targetStatus": 0, // 当前目标点状态，0表示已经到达或者取消free，1表示正在前往目标点过程中working,2表示当前目标点的移动任务被暂停paused,3表示目标点出现错误error,默认值为-1表示无效值。
  "targetDistance": -1, // 机器人距离当前目标点的距离，单位为米，-1表示无效值，该值的绝对值小于0.01时表示已经到达。
  "angleGoalStatus": 1, // 目标角度达到情况，0表示未完成，1表示完成，2表示error,默认值为-1表示无效值。
  "controlSpeedX": 0, // 导航系统计算给出的前进速度控制分量,单位为m/s。
  "controlSpeedTheta": 0, // 导航系统计算给出的角速度控制分量,单位为rad/s。
  "currentSpeedX": -0.0000010095536708831787, // 当前机器人实际前进速度分量,单位为m/s。
  "currentSpeedTheta": 0.0002659947786014527,// 当前机器人实际角速度分量,单位为rad/s。
  "currentPosX": -1, // 当前机器人在map坐标系下的X坐标,此坐标可以直接用于设置动态插入点坐标
  "currentPosY": -1, // 当前机器人在map坐标系下的Y坐标
  "currentAngle": -1, // 当前机器人在map坐标系下的z轴转角(yaw)
  "busyStatus": 0 //当busy为true时系统将仍然后接收新指令，但是不会立即处理。当系统退出busy状态后再处理消息
}
```

获取系统日志

URL: /system/log

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
log	string	系统日志信息

获取机器人速度

URL: /system/speed

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
speed_x	float	x方向速度，单位为米每秒
speed_y	float	y方向速度，单位为米每秒
speed_angle	float	转动角速度, 单位是弧度每秒

说明:

机器人本体坐标系方向为，机器人正前方为x方向，机器人左方为y方向，机器人上方为z方向。y方向的速度是全向轮如麦克纳姆轮才有的。对于一般差速底盘y方向速度一直为0

遥控机器人

URL: /system/speed

请求方式: PUT

请求参数:

参数	类型	说明
speed_x	float	x方向速度
speed_y	float	y方向速度
speed_angle	float	转动角速度

返回参数:

参数	类型	说明
result	bool	遥控指令是否成功执行

说明: 调用此API后机器人会以指定的速度移动。移动的时间是不确定的，这个由驱动器决定，一般是几秒的时间。实际开发遥控功能时可以以5Hz左右的频率调用此API。 对于6.2.1之后版本的机器人，遥控也可以通过websocket实现，这样可以极大的提高响应速度，尤其对于使用跨网访问接口。通过websocket向/cmd_vel话题发送Twist消息即可。

关闭机器人

URL: /system/shutdown

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	bool	是否成功接收关机指令

校正机器人陀螺仪

URL: /system/calibrate_imu

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	bool	是否开始校准陀螺仪

开始校正摄像头角度位置参数

URL: /system/calibrate_camera/start

请求方式: GET

请求参数:

参数	类型	说明
camera_id	int	当没有此参数时为单摄像头标定，当为0时为前摄像头标定，当为1时为后摄像头标定

说明：执行启动校准方法后需要遥控机器人在环境中运动，使其路径形成一个闭环完成闭环优化。闭环优化完成后，继续移动一段距离，然后调用提交参数方法完成参数校准。

请求参数: 无

返回参数:

参数	类型	说明
result	string	是否开始校准陀螺仪

取消摄像头角度位置校正参数

URL: /system/calibrate_camera/cancel

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	string	是否取消校准陀螺仪

获取摄像头角度位置校正状态

URL: /system/calibrate_camera/status

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	标定状态，可以是CALIBRATING或CALIBRATED。分别表示标定中和标定完成
accuracy	float	在标定完成时候会包含此参数，表明标定准确度，最大值为100

提交摄像头角度位置校正参数

URL: /system/calibrate_camera/complete

请求方式: GET

说明： 只有在处于CALIBRATED状态下才能够提交标定参数

请求参数: 无

返回参数:

参数	类型	说明
status	string	标定参数是否提交成功

开始标定雷达参数

URL: /system/calibrate_laser/start

请求方式: GET

说明: 雷达校准需要在摄像头校准之后完成。开启后遥控机器人在具有明显边界的环境中移动。等待标定参数生成

返回参数:

参数	类型	说明
status	string	是否开始校准雷达

提交雷达校正参数

URL: /system/calibrate_laser/complete

请求方式: GET

说明: 在雷达标定状态变为 CALIBRATED 之后，调用此API将生成的标定参数提交到系统内。

返回参数:

参数	类型	说明
status	string	是否成功提交雷达校准参数

取消校准雷达参数

URL: /system/calibrate_laser/cancel

请求方式: GET

返回参数:

参数	类型	说明
status	string	是否成功取消雷达校准

获取雷达校准状态

URL: /system/calibrate_laser/status

请求方式: GET

返回参数:

参数	类型	说明
status	string	雷达校准状态，可能为CALIBRATING， CALIBRATED， FREE

检查系统更新状态

URL: /system/update/check

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	检查更新程序是否成功执行
need_update	bool	是否需要更新

开始自动更新程序

URL: /system/update/start

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	系统更新程序是否成功启动

取消自动更新程序

URL: /system/update/cancel

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	取消系统更新是否成功

获取软件更新日志

URL: /system/update/log

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	是否成功获取更新日志
log	string	系统更新日志内容
is_complete	bool	系统更新是否已经完成

获取当前调度服务器

URL: /system/schedule_manager

请求方式: GET

请求参数: 无

返回参数: 所有schedule manager数据，其中第一个是当前的schedule manager

返回数据示例


```
[
  {
    "ip": "192.168.0.23",
    "update_time": 1576115449217,
    "version": "1.0.0",
    "id": "9188d590-8508-47bd-a704-d34ddb803265",
    "port": 24958
  }
]
```


系统自检

URL: /system/self_test

请求方式: GET

返回参数:

参数	类型	说明
motor_driver	bool	底盘驱动状态是否正常
camera	bool	摄像头状态是否正常
charge	bool	自动充电模块是否正常
lidar	bool	雷达是否正常
bluetooth	bool	蓝牙是否工作正常
battery	bool	获取电池电压是否正常

获取当前io电平情况

注意io仍为电平输出，获取的是输出电平的状态，并不是输入信号

URL: /system/io

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	获取状态
out1	string	"-1"为未设置，"0"为低电平, "1"为高电平
out2	string	"-1"为未设置，"0"为低电平, "1"为高电平
out3	string	"-1"为未设置，"0"为低电平, "1"为高电平

控制当前io电平情况

URL: /system/io

请求方式: POST

请求参数:

参数	类型	说明
level	string	0 为低电平， 1为高电平
port	string	可以为1,2,3分别对应三个IO端口

返回参数:

参数	类型	说明
status	string	获取状态
status	string	设置电平状态

获取IO扩展板电平

说明: IO扩展板有16路IO，其中8路输入8路输出

URL: /system/bw_io

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
in	object	输入端口的电平情况
out	object	输出端口的电平情况

返回数据示例

```
{
    "in": {
        "0": 0,
        "1": 0,
        "2": 0,
        "3": 0,
        "4": 0,
        "5": 0,
        "6": 0,
        "7": 0,
    },
    "out": {
        "0": 0,
        "1": 0,
        "2": 0,
        "3": 0,
        "4": 0,
        "5": 0,
        "6": 0,
        "7": 0,
    }
}
```

控制IO扩展板电平

URL: /system/bw_io

请求方式: PUT

请求参数:

参数	类型	说明
level	int	0 为低电平， 1为高电平
port	int	可以为0-7分别对应8个IO端口

返回参数:

参数	类型	说明
status	string	设置电平状态

获取机器人当前配置参数

URL: /system/config

请求方式: GET

请求参数: 无

返回参数:

5.0 以前版本返回

参数	类型	说明
default_map	string	默认地图名称
maps	list	地图对应默认路径配置
navigation_speed	float	最大导航速度
max_control_speed	float	最大遥控速度
bar_distance_min	float	避障距离
k2	float	PID控制参数k2
kp	float	PID控制参数kp
ki	float	PID控制参数ki
kd	float	PID控制参数kd
look_ahead_dist	float	预估距离
theta_max	float	最大角速度
plan_width	float	车体宽度
path_change	float	避障时是否绕开
forward_width	float	预估距离
rot_width	float	车体旋转宽度
backtime	float	最大后退距离

5.0 以后版本返回

参数	类型	说明
default_map	string	默认地图名称
maps	list	地图对应默认路径配置
auto_charge	bool	是否开启低电量自动充电
slam_type	string	导航类型 "camera"为摄像头导航，"lidar"为雷达导航
lf_linear_v	float	最大导航速度
lf_theta_max	float	最大导航角速度
lf_k2	float	PID控制参数k2
lf_kp	float	PID控制参数kp
lf_ki	float	PID控制参数ki
lf_kd	float	PID控制参数kd
lf_look_ahead_dist	float	预估距离
robot_radius	float	机器人半径
lf_xy_goal_tolerance	float	位置允许误差
lf_yaw_goal_tolerance	float	角度允许误差
astar_path_changeability	bool	是否绕开
lf_bar_distance_min	float	避障距离
lf_forward_max_dist	float	避障最远检测距离
lf_move_direction	int	导航时机器人方向，0正向，1反向，2自动
go_charge_percentage	float	返回充电电量百分比，需要6.1.7及以上版本
use_smooth_switch_map	float	是否使用导航流畅切换，开启后机器人切换地图时会更加流畅，但是跨地图导航时机器人会消耗更多CPU

6.6.8 增加参数

参数	类型	说明
lagrange_id	string	拉格朗日调度服务器ID
weixin_id	string	微信消息发送目标
wifi_ssid	string	wifi名称
wifi_password	string	wifi密码, 当有此参数时机器人会自动连接对应WiFi，如果有多个同名WiFi机器人会自动选择其中信号较好的

修改机器人配置

URL: /system/config

请求方式: POST

请求参数:

5.0 版本之前参数

参数	类型	说明
default_map	string	默认地图名称，可选参数
maps	list	地图对应默认路径配置，可选参数
navigation_speed	float	最大导航速度，可选参数
max_control_speed	float	最大遥控速度，可选参数
bar_distance_min	float	避障距离，可选参数
k2	float	PID控制参数k2，可选参数
kp	float	PID控制参数kp，可选参数
ki	float	PID控制参数ki，可选参数
kd	float	PID控制参数kd，可选参数
look_ahead_dist	float	预估距离，可选参数
theta_max	float	最大角速度，可选参数
plan_width	float	车体宽度，可选参数
path_change	float	避障时是否绕开，可选参数
forward_width	float	预估距离，可选参数
rot_width	float	车体旋转宽度，可选参数
backtime	float	最大后退距离，可选参数
deliver_wait_time	float	送餐最长等待时间，可选参数

5.0版本之后参数

参数	类型	说明
default_map	string	默认地图名称
maps	list	地图对应默认路径配置
auto_charge	bool	是否开启低电量自动充电
slam_type	string	导航类型 "camera"为摄像头导航，"lidar"为雷达导航
lf_linear_v	float	最大导航速度
lf_theta_max	float	最大导航角速度
lf_k2	float	PID控制参数k2
lf_kp	float	PID控制参数kp
lf_ki	float	PID控制参数ki

lf_kd	float	PID控制参数kd
lf_look_ahead_dist	float	预估距离
robot_radius	float	机器人半径
lf_xy_goal_tolerance	float	位置允许误差
lf_yaw_goal_tolerance	float	角度允许误差
astar_path_changeability	bool	是否绕开
lf_bar_distance_min	float	避障距离
lf_forward_max_dist	float	避障最远检测距离
lf_move_direction	int	导航时机器人方向，0正向，1反向，2自动
go_charge_percentage	float	返回充电电量百分比，需要6.1.7及以上版本
use_smooth_switch_map	float	是否使用导航流畅切换，开启后机器人切换地图时会更加流畅，但是跨地图导航时机器人会消耗更多CPU

返回参数:

修改后的机器人参数

5.0 以前版本返回

参数	类型	说明
default_map	string	默认地图名称
maps	list	地图对应默认路径配置
navigation_speed	float	最大导航速度
max_control_speed	float	最大遥控速度
bar_distance_min	float	避障距离
k2	float	PID控制参数k2
kp	float	PID控制参数kp
ki	float	PID控制参数ki
kd	float	PID控制参数kd
look_ahead_dist	float	预估距离
theta_max	float	最大角速度
plan_width	float	车体宽度
path_change	float	避障时是否绕开
forward_width	float	预估距离
rot_width	float	车体旋转宽度
backtime	float	最大后退距离

5.0 以后版本返回

参数	类型	说明
default_map	string	默认地图名称
maps	list	地图对应默认路径配置
navigation_speed	float	最大导航速度
auto_charge	bool	是否开启低电量自动充电
slam_type	string	导航类型 "camera"为摄像头导航，"lidar"为雷达导航
lf_linear_v	float	最大导航速度
lf_theta_max	float	最大导航角速度
lf_k2	float	PID控制参数k2
lf_kp	float	PID控制参数kp
lf_ki	float	PID控制参数ki
lf_kd	float	PID控制参数kd
lf_look_ahead_dist	float	预估距离
robot_radius	float	机器人半径
lf_xy_goal_tolerance	float	位置允许误差
lf_yaw_goal_tolerance	float	角度允许误差
astar_path_changeability	bool	是否绕开
lf_bar_distance_min	float	避障距离
lf_forward_max_dist	float	避障最远检测距离
lf_move_direction	int	导航时机器人方向，0正向，1反向，2自动
go_charge_percentage	float	返回充电电量百分比，需要6.1.7及以上版本
use_smooth_switch_map	float	是否使用导航流畅切换，开启后机器人切换地图时会更加流畅，但是跨地图导航时机器人会消耗更多CPU

恢复机器人默认参数

URL: /system/config

请求方式: DELETE

请求参数

参数	类型	说明
key	string	对应参数的名称,如k2,kp,ki等等

返回参数:

参数	类型	说明

status	string	恢复默认值操作状态

让机器人播放语音

URL: /system/tts

请求方式: GET

请求参数:

参数	类型	说明
text	string	需要机器人播放的语音对应的文本

返回参数:

参数	类型	说明
status	string	语音播放状态

说明: 机器人语音文件在第一次播放时需要联网下载，所以机器人未联网情况下无法播放新语音。成功播放后语音文件会自动缓存在机器人中，之后播放不再需要网络连接。

机器人当前是否正在播放语音

URL: /system/tts/playing

请求方式: GET

返回参数:

参数	类型	说明
status	bool	语音播放状态

停止正在播放的语音

URL: /system/tts/stop

请求方式: GET

返回参数:

参数	类型	说明
status	string	是否成功停止播放

机器人的语音播放记录

URL: /system/tts/records

请求方式: GET

请求参数:

参数	类型	说明
package_name	string	语音包名称，可选参数。当没有此参数时返回所有的语音记录
text	string	当有此参数时，返回对应文字对应的语音文件。当没有此参数时返回整个语音包文件

返回参数:

当没有package_name参数时返回的数据结构如下

参数	类型	说明
packages	list	语音包名称列表
records	list	语音播放记录
current_package	string	当前语音包名称

示例数据:


```
{
    "packages": [
        "\u6653\u6653",
        "zh-cn",
        "Jenny",
        "SunHi",
        "\u5c0f\u8587",
        "xx"
    ],
    "records": [
        {
            "text": "\u6b22\u8fce\u5149\u4e34",
            "md5sum": "d5bd516c5a6827d585b1a4b08e352241",
            "tags": [
                "\u8fce\u5bbe\u6a21\u5f0f"
            ],
            "codenames": [
                "all"
            ],
            "i18n": {
                "en": "Welcome",
                "ko": "\uc548\ub155\ud558\uc138\uc694",
                "zh-cn": "\u6b22\u8fce\u5149\u4e34"
            },
            "mode": [
                "all"
            ],
            "last_active": 1663219852937,
            "active_count": 81
        },
        {
            "text": "\u5f00\u59cb\u5145\u7535",
            "md5sum": "1572fba0bf9d48b8c086b5c83f10e319",
            "tags": [
                "\u5145\u7535"
            ],
```

```
            "codenames": [
                "all"
            ],
            "i18n": {
                "en": "Start charging",
                "ko": "\ucda9\uc804 \uc2dc\uc791",
                "zh-cn": "\u5f00\u59cb\u5145\u7535"
            },
            "mode": [
                "all"
            ],
            "last_active": 1663482021699,
            "active_count": 421
        },
        ...
    ],
    "current_package": "\u6653\u6653"
}
```


上传语音包(语音音频文件方式)

URL: /system/tts/records

请求方式: POST

请求参数:

参数	类型	说明
package_name	string	语音包名称

说明: 注意整个请求不是以json格式调用。由于需要上传语音文件，数据要通过Form的形式上传。 语音包文件要添加到Form里面。成功上传后机器人会自动切换到当前的语音包。

返回参数:

参数	类型	说明
status	string	是否上传成功

创建或替换语音包（语音压缩包方式）

URL: /system/tts/records

说明: 把之前导出的语音包文件导入机器人。注意由于需要上传文件此请求不是json格式。需要用Form方式上传。上传成功后机器人会自动切换到此语音包

请求参数:

参数	类型	说明
package_name	string	语音包名称
package_file	file	语音包文件

返回参数:

参数	类型	说明
status	string	是否上传成功

替换语音文件

URL: /system/tts/records

说明: 替换机器人中特定文字对应的语音。由于需要上传语音文件所以此请求的方式不是json.数据需要通过添加到Form里面进行上传。

请求参数:

参数	类型	说明
package_name	string	语音包名称
text	string	目标替换文字
audio_file	string	语音文件，可以为常见的音频文件格式

返回参数:

参数	类型	说明
status	string	是否替换成功

删除语音包

URL: /system/tts/records

请求参数:

参数	类型	说明
package_name	string	语音包名称

返回参数:

参数	类型	说明
status	string	是否删除成功

重置默认语音

URL: /system/tts/records

说明：把替换的语音重置为系统默认的语音。注意重置过程机器人需要联网下载语音。所以要保证机器人联网。

请求参数:

参数	类型	说明
text	string	需要重置的文字

返回参数:

参数	类型	说明
status	string	是否删除成功

获取语音修改文本

URL: /system/tts/alias

请求方式: GET

说明: 系统默认的语音可以替换成其他文字进行显示。注意替换并不会影响record记录。只是增加了一个alias记录。语音文本显示时可以判断是否有对应的alias记录，如果有则显示alias记录。

请求参数: 无

返回参数:

返回参数是一个dict，其中key是原文本，value是替换的文本

示例

```
{"\u4f60\u597d": "\u4f60\u597d\u554a"}
```

创建alias记录

URL: /system/tts/alias

请求方式: POST

说明: 这里上传的数据是所有的alias记录。上传后新数据会直接覆盖掉所有的老数据。

请求参数:

请求参数为一个dict，其中key是原文本，value是替换的文本

示例数据

```
{"\u4f60\u597d": "\u4f60\u597d\u554a"}
```

返回参数:

参数	类型	说明
status	string	状态说明，默认ok

替换文本内容和语音

URL: /system/tts/alias

请求方式: PUT

说明: 不同于上面全覆盖的方式，这个API只会修改发生变化的文本内容。同时生成新内容对应的语音文件，自动替换旧语音文件。

请求参数:

请求参数为一个dict，其中key是原文本，value是替换的文本

返回参数:

返回参数是一个dict，其中key是原文本，value是替换的文本

迎宾模式开关

URL: /system/greeting

请求方式: PUT

说明: 迎宾模式开启后，当有人走过机器人后机器人会说欢迎光临。你也可以通过语音相关的api替换成其他语句。

请求参数:

参数	类型	说明
greeting	bool	是否开启迎宾模式

返回参数:

参数	类型	说明
greeting	bool	是否开启迎宾模式

获取对应话题数据

URL: /system/topic

请求方式: GET

说明: 获取对应话题的数据，自动转换成json格式进行返回。由于订阅话题需要一个时间，第一个调用可能会返回空数据。

请求参数:

参数	类型	说明
topic_name	string	话题名称

返回参数:

话题对应的json数据。

获取语音模块IO电平

URL: /system/asr/io

请求方式: GET

说明: 语音模块共8个IO其中1-4是输出端口，5-8是输入端口。获取电平为获取输入端口电平。实际硬件端口序号从1开始，在程序中序号从0开始

请求参数: 无

返回参数:

参数	类型	说明
4	bool	是否为高电平
5	bool	是否为高电平
6	bool	是否为高电平
7	bool	是否为高电平

设置语音模块输出端口电平

URL: /system/asr/io

请求方式: PUT

说明: 语音模块共8个IO其中1-4是输出端口，5-8是输入端口。设置电平为设置输出端口电平。实际端口序号从1开始，在程序中序号从0开始。注意设置io有频率限制，过高的设置频率会导致部分设置被丢弃失效。最高设置频率100HZ

请求参数:

请求参数是一个dict，其中key是端口号，value是电平数据。

示例数据

```
{
  "0": true,
  "1": true,
  "2": true,
  "3": true,
}
```

返回数据:

```
{
```

```
  "status": "ok"
}
```

设置语音模块输出端口PWM

URL: /system/asr/pwm

请求方式: GET

请求参数:

参数	类型	说明
port	int	设置的目标端口
freq	int	pwm频率
percent	int	pwm占空比

返回参数:

```
{
  "status": "ok"
}
```

设置语音模块输出舵机信号

URL: /system/asr/servo

请求方式: GET

请求参数:

参数	类型	说明
port	int	输出端口号
angle	int	输出舵机角度

返回参数:

```
{
  "status": "ok"
}
```

检查机器人证书

URL: /system/check_cert

请求方式: GET

请求参数: 无

返回值

```
{
  "status": "ok"
}
```

让机器人连接wifi

URL: /system/wifi

请求方式: GET

请求参数:

参数	类型	说明
ssid	string	wifi的名称
password	string	wifi密码

返回参数:

参数	类型	说明
status	string	状态说明，默认ok

注意: wifi最后是否连接成功是以机器人能否访问互联网判断的。如果你所要连接的wifi不能连接互联网则即使机器人成功连接了wifi，此时机器人仍然会返回连接错误。

是否使用预览版程序

URL: /system/use_dev

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
use_dev	bool	是否使用预览版

设置使用预览版

URL: /system/use_dev

请求方式: PUT

请求参数:

参数	类型	说明

use_dev	bool	是否使用预览版

返回参数:

参数	类型	说明
use_dev	bool	是否使用预览版

是否开启跟随功能

URL: /system/tracking_mode

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
tracking_flag	bool	是否开启跟随功能

开启关闭跟随功能

URL: /system/tracking_mode

请求方式: PUT

请求参数:

参数	类型	说明
enable	bool	是否开启跟随功能

获取系统音量设置

URL: /system/volume

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
value	int	音量百分比

设置系统音量

URL: /system/volume

请求方式: PUT

请求参数:

参数	类型	说明
value	int	音量百分比

返回参数:

参数	类型	说明
status	string	ok

获取当前机器人是否连上互联网

URL: /system/is_online

请求方式: GET

请求参数:

无

返回参数

参数	类型	说明
result	bool	是否能连上互联网标志

获取已经保存的照片文件

URL: /system/saved_images/

说明: 获取已经保存的拍照图片文件。file_timestamp为文件的时间戳，拍照完成时会返回此参数。如果想要获取所有的图片列表可以不加file_timestamp访问此接口。

请求方式：GET

请求参数: 无

返回参数:

返回图片文件。

拍照

URL: /system/saved_images

说明: 拍照并保存图片文件。图片默认保存在 `/home/xiaoqiang/Pictures` 目录下，文件名为 `<时间戳>.png`

请求方式：POST

请求参数:

参数	类型	说明
topic	string	可选参数，拍照的ROS话题，默认为/camera_node/image_raw，前摄像头话题为/multi/front/image_raw，后摄像头话题为/multi/back/image_raw
filename	string	可选参数，保存的文件名，默认为时间戳

返回参数:

参数	类型	说明
image	string	图片的时间戳,可以用此时间戳获取到文件内容

删除拍照文件

URL: /system/saved_images/

请求方式: DELETE

请求参数: 无

返回参数:

参数	类型	说明
status	string	ok

获取已经保存的录像文件

URL: /system/saved_videos/

说明: 获取已经保存的录像文件。file_timestamp为文件的时间戳，录像完成时会返回此参数。录像文件默认保存在 `/home/xiaoqiang/Videos` 目录下，文件名为 `<时间戳>.mkv` 。如果想要获取所有的录像列表可以不加file_timestamp访问此接口。

请求方式：GET

请求参数: 无

返回参数:

返回录像文件。

开始录像

URL: /system/saved_videos

说明: 开始录像并保存视频文件。

请求方式：POST

请求参数:

参数	类型	说明
topic	string	可选参数，录像的ROS话题，默认为/camera_node/image_raw，前摄像头话题为/multi/front/image_raw，后摄像头话题为/multi/back/image_raw
duration	int	可选参数，录像时长，单位为秒，默认为10秒
filename	string	可选参数，保存的文件名，默认为时间戳

返回参数:

此API会自动创建一个包含录像动作的任务。返回参数为任务对象。

当通过任务控制API停止此任务时，录像会停止并保存文件。

删除录像文件

URL: /system/saved_videos/

请求方式: DELETE

请求参数: 无

返回参数:

参数	类型	说明
status	string	ok

滑台控制相关API

获取滑台当前位置

URL: /system/slide_pose

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
zero_status	int	是否已经归零, 0 没完成， 1 已经完成， -1 有错误
current_pose	float	当前位置，相对零点，zereo_status为1时才有效，单位米
pose_ratio	float	当前位置比例，相对零点，zereo_status为1时才有效, 0-1取值

移动滑台

URL: /system/slide_move_to

请求方式: GET

请求参数:

参数	类型	说明
pose	float	滑台高度百分比
speed	float	滑台速度单位米/秒，0表示使用默认速度

返回参数:

参数	类型	说明
task	object	滑台移动任务对象，滑台移动API内部会构造一个滑台移动任务

返回数据示例:


```
{
    "id": "bff2bfb0-b2eb-42b8-9324-d86920faed83",
    "name": "slide_task",
    "loop_flag": false,
    "current_task": {
        "id": "9fe85f29-52f6-4783-b313-e812149f53aa",
        "type": "slide_action",
        "mode": 3,
        "pose": 0.5,
        "speed": 0,
        "time": 0,
        "progress": 0
    },
    "state": "WORKING",
    "sub_tasks": [
        {
            "id": "9fe85f29-52f6-4783-b313-e812149f53aa",
            "type": "slide_action",
            "mode": 3,
            "pose": 0.5,
            "speed": 0,
            "time": 0,
            "progress": 0
        }
    ],
    "progress": 0.0,
    "loop_count": 0
}
```


顶升机构控制API

获取当前顶升位置

URL: /system/jacking_pose

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
zero_status	int	是否已经归零, 0 没完成， 1 已经完成， -1 有错误
current_pose	float	当前位置，相对零点，zereo_status为1时才有效，单位米
pose_ratio	float	当前位置比例，相对零点，zereo_status为1时才有效, 0-1取值

设置顶升位置

URL: /system/jacking_move_to

请求方式: GET

请求参数:

参数	类型	说明
pose	float	顶升高度百分比
speed	float	顶升速度单位米/秒，0表示使用默认速度，可选参数

返回参数:

参数	类型	说明
task	object	顶升移动任务对象，顶升移动API内部会构造一个顶升任务

返回数据示例:


```
{
    "id": "bff2bfb0-b2eb-42b8-9324-d86920faed83",
    "name": "jacking_task",
    "loop_flag": false,
    "current_task": {
        "id": "9fe85f29-52f6-4783-b313-e812149f53aa",
        "type": "jacking_action",
        "mode": 3,
        "pose": 0.5,
        "speed": 0,
        "time": 0,
        "progress": 0
    },
    "state": "WORKING",
    "sub_tasks": [
        {
            "id": "9fe85f29-52f6-4783-b313-e812149f53aa",
            "type": "jacking_action",
            "mode": 3,
            "pose": 0.5,
            "speed": 0,
            "time": 0,
            "progress": 0
```

```
            }
        ],
        "progress": 0.0,
        "loop_count": 0
    }
```

获取顶升角度

URL: /system/jacking_angle

请求方式: GET

请求参数:

参数	类型	说明
frame_id	string	坐标系ID，可选参数，不传就是机器人坐标系 可以是base_link, map

返回参数:

参数	类型	说明
zero_status	int	是否已经归零, 0 没完成， 1 已经完成， -1 有错误
current_angle	float	当前角度，单位弧度

设置顶升角度

URL: /system/jacking_angle_move_to

请求方式: GET

请求参数:

参数	类型	说明
angle	float	顶升角度，单位弧度
speed	float	顶升速度单位弧度/秒，0表示使用默认速度， 可选参数

返回参数:

参数	类型	说明
task	object	顶升角度任务对象，顶升角度API内部会构造一个顶升任务

获取UWB跟随状态

URL: /system/uwb_track

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
enable	bool	是否启用UWB跟随功能

启用UWB跟随功能

URL: /system/uwb_track

请求方式: PUT

请求参数:

参数	类型	说明
enable	bool	是否启用UWB跟随功能

返回参数:

参数	类型	说明
status	string	ok

获取ROS参数服务器参数

URL: /system/ros_params

请求方式: GET

请求参数:

参数	类型	说明
param	string	参数名称

返回参数:

参数	类型	说明
请求参数名称	object	参数值

设置ROS参数服务器参数

URL: /system/ros_params

请求方式: PUT

请求参数:

参数	类型	说明
param	string	参数名称

value	object	参数值

返回参数:

参数	类型	说明
请求参数名称	object	参数值

重启路由器

URL: /system/restart_router

请求方式: GET

请求参数:

参数	类型	说明
type	string	路由器类型，可选参数 oray蒲公英路由器，tplink，huawei，tcl
ip	string	路由器IP地址
username	string	路由器用户名
password	string	路由器密码

返回参数:

参数	类型	说明
status	string	ok

重启机器人

URL: /system/restart_robot

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	bool	true

重启novnc服务

URL: /system/restart_novnc

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	bool	true

onvif 监控摄像头相关API

获取局域网内所有摄像头信息

URL: /system/onvif_cameras

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
cameras	list	局域网内的摄像头ip和端口

返回数据例子

```
[
  {
    "ip": "192.168.0.13",
    "port": 9999
  }
]
```

获取特定摄像头参数

URL: /system/onvif_cameras

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码

返回参数:

参数	类型	说明
		摄像头的profile信息,由于profile有无法序列化的内容，所以这个返回内容是

		text

示例返回数据:

```
[{
    'Name': 'Stream_101',
    'VideoSourceConfiguration': {
        'Name': 'VideoSourceConfig_1',
        'UseCount': 2,
        'SourceToken': 'VideoSource_1',
        'Bounds': {
            'x': 0,
            'y': 0,
            'width': 2688,
            'height': 1520
        },
        '_value_1': None,
        'Extension': None,
        'token': 'VideoSourceToken_1',
        '_attr_1': {
    }
    },
    'AudioSourceConfiguration': {
        'Name': 'AudioSourceConfig_1',
        'UseCount': 4,
        'SourceToken': 'AudioSourceChannel_1',
        '_value_1': None,
        'token': 'AudioSourceConfigToken_1',
        '_attr_1': {
    }
    },
    'VideoEncoderConfiguration': {
        'Name': 'VideoEncoder_101',
        'UseCount': 1,
        'Encoding': 'H264',
        'Resolution': {
            'Width': 2688,
            'Height': 1520
        },
        'Quality': 3.0,
        'RateControl': {
            'FrameRateLimit': 25,
            'EncodingInterval': 1,
            'BitrateLimit': 6144
        },
        'MPEG4': None,
        'H264': {
            'GovLength': 50,
            'H264Profile': 'High'
        },
        'Multicast': {
            'Address': {
                'Type': 'IPv4',
                'IPv4Address': '239.0.0.1',
                'IPv6Address': None
            },
            'Port': 8860,
            'TTL': 128,
            'AutoStart': False,
```

```
            '_value_1': None,
            '_attr_1': None
        },
        'SessionTimeout': datetime.timedelta(seconds=5),
        '_value_1': None,
        'token': 'VideoEncoderToken_101',
        '_attr_1': {
    }
    },
    'AudioEncoderConfiguration': {
        'Name': 'AudioEncoderConfig_1',
        'UseCount': 4,
        'Encoding': 'G711',
        'Bitrate': 64,
        'SampleRate': 8,
        'Multicast': {
            'Address': {
                'Type': 'IPv4',
                'IPv4Address': '239.0.0.1',
                'IPv6Address': None
            },
            'Port': 8862,
            'TTL': 128,
            'AutoStart': False,
            '_value_1': None,
            '_attr_1': None
        },
        'SessionTimeout': datetime.timedelta(seconds=5),
        '_value_1': None,
        'token': 'MainAudioEncoderToken_1',
        '_attr_1': {
    }
......
```

获取特定摄像头视频流

URL: /system/onvif_camera/stream

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码
channel	int	摄像头图像通道,可选参数，没有就默认为1通道

返回参数:

参数	类型	说明

stream	string	摄像头视频流地址

返回数据示例:

```
{
    "stream": "rtsp://admin:SctecCam01@192.168.0.247:554/Streaming/Channels/101?transportmode=unicast&profile=Profile_101"
}
```

绝对位置移动监控摄像头

URL: /system/onvif_camera/move

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码
x	float	摄像头横向转动角度
y	float	摄像头纵向转动角度
zoom	float	摄像头缩放比例

返回参数:

参数	类型	说明
status	string	ok,执行状态
description	string	状态描述

监控摄像头移动到预置点

URL: /system/onvif_camera/goto_preset

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口

username	string	摄像头用户名
password	string	摄像头密码
preset	string	摄像头预置点名称

返回参数:

参数	类型	说明
status	string	ok,执行状态

速度模式移动监控摄像头

URL: /system/onvif_camera/move_speed

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码
speed_x	float	摄像头横向转动速度
speed_y	float	摄像头纵向转动速度
speed_zoom	float	摄像头缩放速度

返回参数:

参数	类型	说明
status	string	success,执行状态
description	string	状态描述

监控摄像头拍照

URL: /system/onvif_camera/snapshot

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip

port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码
channel	int	摄像头通道,可选参数，默认为1

返回参数:

参数	类型	说明
snapshot	string	拍照生成的照片位置, 可以通过文件API访问

获取监控摄像头温度

说明: 需摄像头支持红外测温并且支持海康的ISAPI协议。如果没有插入用户测温点则此API返回所有返回内的温度信息。如果添加了用户测温点则返回用户测温点的温度信息。

URL: /system/onvif_camera/temperature

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码

返回参数:

参数	类型	说明
temperature	object	摄像头温度信息

返回参数示例:

```
{
    "temperature": {
        "ThermometryRulesTemperatureInfoList": {
            "ThermometryRulesTemperatureInfo": [
                {
                    "id": 1,
                    "maxTemperature": 24.9,
                    "minTemperature": 24.0,
                    "averageTemperature": 24.5,
                    "MaxTemperaturePoint": {
                        "positionX": 0.308,
                        "positionY": 0.889
                    },
```

```
                    "MinTemperaturePoint": {
                        "positionX": 0.154,
                        "positionY": 0.93
                    },
                    "isFreezedata": false
                },
                {
                    "id": 2,
                    "maxTemperature": 24.0,
                    "minTemperature": 24.0,
                    "averageTemperature": 24.0,
                    "MaxTemperaturePoint": {
                        "positionX": 0.851,
                        "positionY": 0.321
                    },
                    "MinTemperaturePoint": {
                        "positionX": 0.851,
                        "positionY": 0.321
                    },
                    "isFreezedata": false
                }
            ]
        }
    }
}
```


获取监控摄像头当前位姿

URL: /system/onvif_camera/pose

请求方式: GET

请求参数:

参数	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码

返回参数:

参数	类型	说明
x	float	摄像头水平角度量
y	float	摄像头垂直角度量
zoom	float	摄像头缩放比例

文件系统操作API

注意为了系统安全，目前只开放了四个可操作的文件夹，分别是：

- /home/xiaoqiang/saved-slamdb
- /home/xiaoqiang/Videos
- /home/xiaoqiang/Pictures
- /home/xiaoqiang/slamdb

获取文件

URL: /fs/file_api

请求方式: GET

请求参数:

参数	类型	说明
path	string	文件路径

返回参数:

文件内容

创建文件

URL: /fs/file_api

请求方式: POST

说明: 注意path的参数放在GET里面，文件通过Form表单上传。文件参数名为file.

请求参数:

参数	类型	说明
path	string	文件路径

返回参数:

参数	类型	说明
status	string	ok

文件或文件夹重命名

URL: /fs/file_api

请求方式: PUT

请求参数:

参数	类型	说明

name	string	原始文件路径
new_name	string	新文件路径

返回参数:

参数	类型	说明
status	string	ok

删除文件

URL: /fs/file_api

请求方式: DELETE

请求参数:

参数	类型	说明
path	string	文件路径

返回参数:

参数	类型	说明
status	string	ok

获取文件夹内容

URL: /fs/dir_api

请求方式: GET

请求参数:

参数	类型	说明
path	string	文件夹路径

返回参数:

参数	类型	说明
files	list	文件列表，以创建时间排序
directories	list	文件夹列表，以创建时间排序

创建文件夹

URL: /fs/dir_api

请求方式: POST

请求参数:

参数	类型	说明
path	string	文件夹路径

返回参数:

参数	类型	说明
status	string	ok

删除文件夹

URL: /fs/dir_api

请求方式: DELETE

请求参数:

参数	类型	说明
path	string	文件夹路径

返回参数:

参数	类型	说明
status	string	ok

获取文件或文件夹详细信息

URL: /fs/get_file_info

请求方式: GET

请求参数:

参数	类型	说明
path	string	文件或文件夹路径

返回参数:

参数	类型	说明
mode	int	文件或文件夹权限
size	int	文件大小
atime	int	最后访问时间
mtime	int	最后修改时间
ctime	int	最后创建时间

mimetype	string	文件类型

复制文件或文件夹

URL: /fs/copy

请求方式: POST

请求参数:

参数	类型	说明
from	string	文件或文件夹路径
to	string	目标路径

返回参数:

参数	类型	说明
status	string	ok

批量删除文件或文件夹

URL: /fs/delete

请求方式: POST

请求参数:

参数	类型	说明
delete	list	文件或文件夹路径列表

返回参数:

参数	类型	说明
status	string	ok

剪切文件或文件夹

URL: /fs/cut

请求方式: POST

请求参数:

参数	类型	说明
from	string	文件或文件夹路径
to	string	目标路径

返回参数:

参数	类型	说明
status	string	ok

创建地图API

启动创建地图

URL: /map/start

请求方式: GET

说明: 接受到启动命令后系统会处于Busy状态，线程启动成功后系统进入Mapping状态。

请求参数:

参数	类型	说明
camera_id	int	可选参数，当没有此参数时为单摄像头建图，为0时为前摄像头建图，为1时为后摄像头建图

返回参数:

参数	类型	说明
result	bool	是否成功启动导航

结束创建地图线程

URL: /map/stop

请求方式: GET

说明: 接收到结束命令进入Busy状态。后先自动保存地图，然后开始关闭建图线程，线程关闭后系统进入Free状态。

请求参数: 无

返回参数

参数	类型	说明
result	bool	是否成功结束建图

更新地图

URL: /map/update

请求方式: GET

请求参数:

参数	类型	说明
map	string	可选参数，需要更新的地图，当没有此参数时则更新上次导航使用的地图
path	string	可选参数,配合start_index使用设置机器人初始位置，对于雷达导航是必须的
start_index	int	可选参数，设置机器人的初始位置，对于雷达导航是必须的

返回参数:

参数	类型	说明
result	bool	是否成功开启更新地图

当前机器人位置

URL: /map/pose

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
x	float	机器人当前x坐标，单位为米
y	float	机器人当前y坐标，单位为米
angle	float	机器人当前朝向角度， 单位为度

获取当前正在创建的地图信息

URL: /map/current_map_image

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	是否成功获取地图信息
map_params	object	地图的meta信息
map_image	string	获取当前地图图片的URL路径

返回数据示例:

```
{
    "status": "OK",
    "map_params": {
        "origin": [
            -7.621091365814209,
            -9.969059944152832,
            0.0
        ],
        "width": 306,
        "resolution": 0.05000000074505806,
        "height": 324
    },
    "map_image": "/api/v1/map/current_map_png"
}
```


说明：机器人图片像素坐标和实际坐标的转换关系为标准的ROS地图关系。像素坐标的原点为图片左上角，x正方向向右，y正方向向下。地图坐标原点为左下角，x正方向向右，y正方向向上。转换关系如下

```
x_map = x_origin + x_pixel * res
y_map = y_origin + h_map * res - y_pixel * res
```

其中 x_map, y_map为像素点对应的地图坐标

x_origin, y_origial 为左下角原点坐标

x_pixel, y_pixel 为像素坐标

h_map为地图像素高度,可以从地图图片文件去获取

保存当前创建的地图

URL: /map/current_map_image

请求方式: POST

请求参数:

参数	类型	说明
name	string	创建的地图名称
type	string	地图类型，可选参数。当为floor时代表此地图为普通楼层地图，当为elevator时代表此地图为电梯地图，默认为floor
levels	int[]	地图楼层，可选参数，此地图所代表的楼层。不同楼层可以共用一个地图，默认为1

返回参数:

参数	类型	说明
result	bool	地图保存是否成功

name	string	保存后的地图名称

获取当前创建地图的png图片

URL: /map/current_map_png

请求方式: GET

请求参数: 无

返回参数: 当前正在创建的地图的png图片

获取创建地图时机器人的运动轨迹

URL: /map/track

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
trajectory	list	创建地图时机器人运动轨迹, 只有在保存地图后才能获取数据

返回数据示例:

```
{
  "trajectory": [
    {
      "y": 0.3912552,
      "x": 0.3480716,
      "z": -0.0384934
    },
    {
      "y": 0.3972594,
      "x": 0.4466029,
      "z": -0.038688
    },
    {
      "y": 0.403202,
      "x": 0.5998781,
      "z": -0.0389315
    },
    ...
  ]
}
```

获取机器人轨迹

URL: /map/track

请求方式: GET

说明: 获取机器人轨迹只能在地图保存之后才行。

请求参数: 无

返回参数:

参数	类型	说明
trajectory	list	返回数据为一个列表。每个元素为包含x y z 属性的dict

示例数据

```
{
    "trajectory": [
        {
            "x": -0.000295,
            "y": 0.0008704,
            "z": 1e-07
        },
        {
            "x": 0.000326,
            "y": 0.000888,
            "z": -9.69e-05
        },
    ]
}
```

导航部分

启动导航状态

URL: /navigation/start

请求方式: GET

说明： 接收到启动命令后系统会处于Busy状态，线程启动成功后系统进入Navigating状态。启动命令参数中包含导航地图文件名字。

请求参数:

参数	类型	说明
map	string	导航所使用的地图名称，当没有此参数时机器人采用上次启动导航时的地图
path	string	导航所使用的路径名称，当没有此参数时机器人采用上次启动导航的路径
start_index	int	机器人初始位置，有此参数时机器人会采用对应目标点位置作为初始位置。如start_index为1时机器人采用1号导航点位置作为初始位置。 当没有此参数时机器人将采用上次启动导航时设置的start_index。对于视觉导航此参数不是必须的。对于雷达导航此参数是必须的。当设置为-1时将采用充电桩位置作为初始位置。当摄像头导航且没有此参数时采用自动定位模式

level	int	可选参数，当前机器人所在楼层。如果没有此参数默认为1

返回参数

参数	类型	说明
result	bool	是否成功启动导航
description	string	若启动失败，其原因说明

导航时开启或关闭更新地图

URL: /navigation/update_map

请求方式: PUT

说明: 在导航状态下开启或关闭更新地图。在这种情况下可以在导航移动的过程中更新地图

请求参数:

参数	类型	说明
enabled	bool	是否开启更新地图，true为开启，false为关闭

返回参数:

参数	类型	说明
enabled	bool	是否开启更新地图

获取是否在导航时更新地图

URL: /navigation/update_map

请求方式: GET

说明: 获取是否在导航时更新地图

请求参数: 无

返回参数:

参数	类型	说明
enabled	bool	是否开启更新地图

启动调度导航

URL: /navigation/start_schedule_mode

请求方式: GET

说明: 接收到启动命令后系统会处于Busy状态，线程启动成功后系统进入Navigating状态。此导航状态不同于普通导航状态。伽利略导航功能将不能使用，机器人将由拉格朗日调度系统控制。

请求参数: 无

返回参数:

参数	类型	说明
result	bool	是否成功启动调度导航
description	string	若启动失败，其原因说明

停止导航

URL: /navigation/stop

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	bool	是否成功启动调度导航
description	string	若启动失败，其原因说明

停止调度导航

URL: /navigation/stop_schedule_mode

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
result	bool	是否成功启动调度导航
description	string	若启动失败，其原因说明

机器人当前位置信息

URL: /navigation/pose

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
x	float	机器人x坐标，单位为米
y	float	机器人y坐标，单位为米
angle	float	机器人朝向角度，单位为度

获取当前导航正在使用的地图和路径名称

URL: /navigation/current_path

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
map	string	当前导航使用的地图名称
path	string	当前导航使用的路径名称

获取当前位置到目标点的距离

URL: /navigation/target_distance

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
distance	float	当前位置到目标点的距离

获取已经保存的详细地图信息

URL: /navigation/saved_maps

请求参数: GET

请求参数:

参数	类型	说明
name	string	目标地图的名称，可选参数。当没有此参数时返回所有已保存的地图信息。当有此参数时，返回目标地图的信息
type	string	地图类型，可选参数。为lidar时返回雷达地图，为camera时返回摄像头地图，为all时返回所有类型地图

返回参数:

参数	类型	说明
resolution	float	地图分辨率，即每像素点对应的实际距离
origin	object	地图原点坐标
occupied_thresh	float	占用阈值，一般没什么用
free_thresh	float	未占用阈值，一般没什么用
negate	int	是否反转占用和未占用，一般没什么用
name	string	目标地图名称
md5sum	string	目标地图数据的md5sum值，用于判断地图是否是一样的
map_type	string	地图类型，值为floor时表示为楼层地图，为elevator时表示为电梯地图
levels	int[]	地图代表的楼层
slam_type	string	地图建图类型，lidar表示为雷达地图，camera表示为摄像头地图

获取保存的地图中的机器人运动轨迹

URL: /navigation/robot_tracks

请求方式: GET

请求参数:

参数	类型	说明
map_name	string	目标路径的地图名称。可选参数，没有此参数时返回所有地图中的机器人路径数据。当有此参数时只返回目标地图的机器人路径数据

返回参数:

当请求参数有map_name时

参数	类型	说明
trajectory	list	机器人运动轨迹点列表

当请求参数有map_name时

返回所有地图的轨迹点列表，包含地图名称和轨迹点数据

返回数据示例:

当没有map_name参数时

```
[
  {
    "map_name": "map1",
    "track": [
```

```
      {
        "y": 0.0000016,
        "x": -0.0010667,
        "z": -0.0000129
      },
      {
        "y": 0.0084065,
        "x": 0.0657083,
        "z": -0.0001012
      },
      {
        "y": -0.0036619,
        "x": 0.1893717,
        "z": -0.0003028
      },
      ...
    ]
  }
]
```

获取目标地图的PGM图片信息

URL: /navigation/map_pgm

请求方式: GET

请求参数:

参数	类型	说明
name	string	目标地图名称

返回目标地图的pgm图片数据

获取目标地图的png图片信息

URL: /navigation/map_png

请求方式: GET

参数	类型	说明
name	string	目标地图名称

返回目标地图的png图片数据

获取当前正在使用的地图

URL: /navigation/current_map

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
name	string	目标地图名称
md5sum	string	目标地图数据的md5sum值

上传当前机器人地图数据至调度服务器

URL: /navigation/upload_map

请求方式: GET

说明: 此方法是异步方法，调用后会在系统中创建一个上传地图数据的线程。上传进度可以通过task相关api获取。使用前一定要保证调度服务器和机器人可以正常通信，且机器人处于被激活状态。

请求参数:

参数	类型	说明
server_id	string	调度服务器id

返回参数:

地图上传任务数据

示例返回数据:


```
{
    "name": "upload_map",
    "loop_flag": false,
    "id": "8bc1e6dd-df1f-4a78-a17b-c60911ce5a06",
    "state": "WORKING",
    "sub_tasks": [
        {
            "server_id": "9188d590-8508-47bd-a704-d34ddb803265",
            "state": "WORKING",
            "result": "",
            "progress": 0,
            "type": "upload_map_action",
            "id": "e21c743d-c692-4ed2-b2be-1470da60ba1e"
        }
    ],
    "progress": 0.0,
    "current_task": {
        "server_id": "9188d590-8508-47bd-a704-d34ddb803265",
        "state": "WORKING",
        "result": "",
        "progress": 0,
        "type": "upload_map_action",
        "id": "e21c743d-c692-4ed2-b2be-1470da60ba1e"
    }
}
```

从调度服务器下载地图

URL: /navigation/download_map

请求方式: GET

请求参数:

参数	类型	说明
server_id	string	调度服务器id
map_id	string	需要下载的地图id

返回参数:

地图下载任务数据

返回数据示例:


```
{
    "name": "download_map",
    "loop_flag": false,
    "id": "fb7669d2-5f03-4461-a992-95422f66d49d",
    "state": "WORKING",
    "sub_tasks": [
        {
            "server_id": "9188d590-8508-47bd-a704-d34ddb803265",
            "map_id": "04f777b8-ff9d-4303-87ac-334dab2e0ffe",
            "state": "WORKING",
            "result": "",
            "progress": 0,
            "type": "download_map_action",
            "id": "2fb7eebc-8048-47d8-b62a-60365c772f87"
        }
    ],
    "progress": 0.0,
    "current_task": {
        "server_id": "9188d590-8508-47bd-a704-d34ddb803265",
        "map_id": "04f777b8-ff9d-4303-87ac-334dab2e0ffe",
        "state": "WORKING",
        "result": "",
        "progress": 0,
        "type": "download_map_action",
        "id": "2fb7eebc-8048-47d8-b62a-60365c772f87"
    }
}
```


下载机器人地图

URL: /navigation/download_map_from_robot

请求方式: GET

请求参数:

参数	类型	说明
map_name	string	需要下载的地图名称

返回数据: 地图文件压缩包

上传地图至机器人

URL: /navigation/upload_map_to_robot

请求方式: POST

请求参数:

参数	类型	说明
map_name	string	上传地图名称
map_files	文件	下载地图API所产生的地图文件压缩包

注意: 此方法的上传数据不是json格式，是MultipartFormData格式

返回

参数	类型	说明
status	string	完成状态,默认ok

开启导航任务

URL: /navigation/start_nav_task

请求方式: POST

请求参数:

参数	类型	说明
x	float	导航目标点坐标x值，单位为米
y	float	导航目标点坐标y值，单位为米
theta	float	导航目标点角度值,单位为弧度
map	string	可选参数,目标点所在的地图
path	string	可选参数,目标点所在的路径

说明:

当没有map和path参数时，机器人在当前地图中移动至对应坐标。如果包含map和path参数，机器人将自动寻找地图的连接点，并在移动过程中自动切换地图，最终到达目标地图中的目标点。此功能可以用来跨地图导航，比如，在不同的楼层间，不同的区域中进行导航。

返回参数:

导航任务数据

示例返回数据:


```
{
  "name": "navigation task",
  "loop_flag": false,
  "id": "a6858132-5e84-4d37-b968-19808dd8fd96",
  "state": "WORKING",
  "sub_tasks": [
    {
      "state": "WAITTING",
      "result": "",
      "progress": 0,
      "theta": 1.0936689376831055,
      "y": -0.1287004053592682,
      "x": -0.28466343879699707,
      "current_location": {
        "y": -1,
        "x": -1,
        "theta": -1
      },
      "type": "nav_action",
      "id": "99bfb264-8f8f-4950-940b-6f6b883e7358"
    }
  ],
  "progress": 0,
  "current_task": {
    "state": "WAITTING",
    "result": "",
    "progress": 0,
    "theta": 1.0936689376831055,
    "y": -0.1287004053592682,
    "x": -0.28466343879699707,
    "current_location": {
      "y": -1,
      "x": -1,
      "theta": -1
    },
    "type": "nav_action",
    "id": "99bfb264-8f8f-4950-940b-6f6b883e7358"
  }
}
```


开始自动充电

URL: /navigation/go_charge

请求方式: GET

请求参数: 无

返回参数:

返回充电任务对象

停止自动充电

URL: /navigation/stop_charge

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	停止充电操作状态
task	object	充电任务对象

移动到对应目标点

URL: /navigation/move_to_index

请求方式: GET

请求参数:

参数	类型	说明
index	int	目标点index，即绘制路径时插入导航点的index
map	string	目标位置所在地图，可选参数
path	string	目标位置所在的路径
level	int	可选参数，当有此参数时会自动通过电梯移动到目标位置

说明： 当没有map和path参数时，机器人会在当前地图上移动到目标位置。如果包含map和path参数，机器人将自动寻找地图的连接点，并在移动过程中自动切换地图，最终到达目标地图中的目标点。此功能可以用来跨地图导航，比如，在不同的楼层间，不同的区域中进行导航。

返回参数:

返回导航任务对象

取消当前导航任务

URL: /navigation/stop_nav_task

请求方式: GET

请求参数: 无

返回参数:

返回当前导航任务对象

获取导航循环任务信息

导航循环任务为沿着插入的导航点依次移动的导航任务，其效果和点击客户端上的循环选项一样。

URL: /navigation/loop_task

请求方式: GET

请求参数: 无

返回参数:

返回循环导航任务状态

开始导航循环任务

URL: /navigation/loop_task

请求方式: POST

请求参数

参数	类型	说明
wait_time	int	循环到达目标点后的等待时间

返回参数:

返回新创建的导航循环任务，如果已有导航循环任务则会返回错误

停止导航循环任务

URL: /navigation/loop_task

请求方式: DELETE

请求参数: 无

返回参数:

返回当前的导航循环任务

获取充电桩位置

URL: /navigation/charge_pose

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明

x	float	充电桩位置坐标x
y	float	充电桩位置坐标y
theta	float	充电桩角度theta

保存充电桩位置

URL: /navigation/charge_pose

请求方式: POST

请求参数: 无

返回参数:

参数	类型	说明
x	float	充电桩位置坐标x
y	float	充电桩位置坐标y
theta	float	充电桩角度theta

动态切换地图

URL: /navigation/reload

method: GET

请求参数:

参数	类型	说明
map	string	想要切换的地图名称
path	string	切换的路径名称，可选参数，当没有此参数时，路径使用机器人设置里面的默认路径

说明:

此方法为动态切换地图，导航程序不会重启。所以切换速度会很快。推荐在机器人需要切换场景时使用，比如上下电梯后不同楼层地图的切换。

返回参数:

```
{
  "status": "ok",
  "map": "",
  "path": ""
}
```

获取导航路径点

URL: /navigation/path

请求方式: GET

请求参数

参数	类型	说明
map_name	string	可选参数，获取的目标地图名称
path_name	string	可选参数，获取的目标路径名称

说明:

当同时有map_name和path_name参数的时候，返回数据为对应路径文件的内容。如下


```
{
  "map_name": "map1",
  "path_name": "path1",
  "path_data": "0.575 -0.356 0\n6.98246 17.95874 0\n9.969936 15.64437 0\n-2.099531 2.19303
4 0\n-2.099531 2.193034 0\n-2.099531 2.193034 0\n-2.099531 2.193034 0\n-3.247468 4.031781
0\n-2.099531 2.193034 0\n6.98246 17.95874 0\n6.98246 17.95874 0\n0.575 -0.356 0\n9.969936
15.64437 0\n-2.099531 2.193034 0\n-2.099531 2.193034 0\n-2.099531 2.193034 0\n-2.099531 2.
193034 0\n-3.247468 4.031781 0\n-2.099531 2.193034 0\n-1.278543 -2.399499 0\n-3.247468 4.0
31781 0\n-2.099531 2.193034 0\n12.3693 12.11829 0\n9.969936 15.64437 0\n-1.928585 1.842135
 0\n-1.278543 -2.399499 0\n-3.247468 4.031781 0\n-2.099531 2.193034 0\n12.3693 12.11829 0\
n9.969936 15.64437 0\n-1.928585 1.842135 0\n-1.278543 -2.399499 0\n-6.600385 -5.06042 0\n-
6.555664 -5.038059 0\n-6.510942 -5.015699 0\n-6.466221 -4.993338 0\n-6.4215 -4.970977 0\n-
6.376779 -4.948617 0\n-6.332057 -4.926256 0\n-6.287336 -4.903895 0\n-6.242614 -4.881535 0\
n-6.197893 -4.859174 0\n-6.153172 -4.836813 0\n-6.10845 -4.814453 0\n"
}
```


当只有map_name参数时，返回的数据是对应地图的所有路径名称，如下


```
[
    "pathY01",
    "path4"
]
```


当没有map_name和path_name参数时，返回的数据为所有地图和其对应的路径名称


```
{
    "14": [
        "pathY01",
        "path\u673a\u573a\u5230\u8fbe\u53e31\u6b63",
        "path\u673a\u573a3",
        "path\u673a\u573a\u5230\u8fbe\u53e31\u53cd",
        "path\u673a\u573a2"
    ],
    "\u58f9\u667a\u4e91\u516c\u53f8": [
        "01"
    ],
    "12": [
        "pathY01",
        "path\u673a\u573a\u5230\u8fbe\u53e31\u6b63",
        "path\u673a\u573a3",
```

```
            "path\u673a\u573a\u5230\u8fbe\u53e31\u53cd",
            "path\u673a\u573a2"
        ],
        "13": [
            "pathY01",
            "path\u673a\u573a\u5230\u8fbe\u53e31\u6b63",
            "path\u673a\u573a3",
            "path\u673a\u573a\u5230\u8fbe\u53e31\u53cd",
            "path\u673a\u573a2"
        ],
        "map5": [
            "pathY01",
            "path4"
        ]
    }
```


上传导航路径点

URL: /navigation/path

请求方式: POST, PUT

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称
path_data	string	路径文件内容

说明:

路径文件内容格式为每个路径点的坐标 x y z，z一般为0，然后换行。可以参照上面API的返回值。如果对应路径已经存在则覆盖之前的路径。

返回参数:

和请求参数一致

删除导航路径点

URL: /navigation/path

请求方式: DELETE

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称

说明:

注意删除路径的时候会自动删除此路径对应的导航点

返回值:

```
{
  "status": "ok"
}
```

获取导航目标点

URL: /navigation/nav_points

请求方式: GET

请求参数:

参数	类型	说明
map_name	string	目标地图名称
path_name	string	目标路径名称

说明:

返回的path_data是导航点文件内容，其格式为 x y z theta errorX errorY direction name。其中x y z为导航点坐标，一般z为0。theta为导航点角度。errorX errorY为导航点所允许的误差。direction为循环方向，name为点的名称，可以为空。和客户端中的设置一致。

当没有传入map_name和path_name参数时，系统会自动选取上次导航的地图和路径，并返回上次导航的导航点数据。

返回参数

```
{
    "map_name": "map5",
    "path_name": "path4",
    "path_data": "0.575 -0.356 0 -1.117307 0.1 0.1 0\n6.98246 17.95874 0 -2.563412 0.1 0.1
 0\n9.969936 15.64437 0 2.045675 0.1 0.1 0\n-2.099531 2.193034 0 -1.078226 0.1 0.1 0\n-2.0
99531 2.193034 0 -1.078226 0.1 0.1 0\n-2.099531 2.193034 0 -1.078226 0.1 0.1 0\n-2.099531
2.193034 0 -1.078226 0.1 0.1 0\n-3.247468 4.031781 0 -0.9649882 0.1 0.1 0\n-2.099531 2.193
034 0 -1.078226 0.1 0.1 0\n6.98246 17.95874 0 -2.563412 0.1 0.1 0\n6.98246 17.95874 0 -2.5
63412 0.1 0.1 0\n"
}
```

上传导航目标点

URL: /navigation/nav_points

请求方式: POST, PUT

参数说明

参数	类型	说明
map_name	string	目标地图的名称
path_name	string	目标路径的名称
path_data	string	导航目标点文件内容

说明：当map_name或path_name没有传入时，将采用当前正常使用的地图和路径最为修改对象。

返回参数:

和请求参数一致

删除导航目标点

URL: /navigation/nav_points

请求方式: DELETE

参数说明

参数	类型	说明
map_name	string	目标地图名称
path_name	string	目标路径名称

说明:

删除导航点的时候并不会自动删除路径文件

返回值

```
{
  "status": "ok"
}
```

获取地图连接点

URL: /navigation/point_connections

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
points	list	当前所有地图中的所有导航点
connections	list	导航点之间的连接关系

示例数据

```
{
    "points": [
        {
            "x": 1.624922, // 导航点 x 坐标
            "y": -0.372011, // 导航点 y 坐标
            "theta": 0.0, // 导航点角度
            "name": "", // 导航点名称
            "map": "office", // 导航点所在地图名称
            "path": "path2", // 导航点所在地图路径
            "index": 0, // 导航点序号
            "type": "floor", // 导航点所属地图类型
            "levels": [ // 导航点所属楼层，可以为多层。和地图楼层一致
                1
            ]
        },
        {
            "x": 0.8710775,
            "y": -0.09214968,
            "theta": -0.2962013,
            "name": "",
            "map": "office",
            "path": "path2",
            "index": 1,
            "type": "floor",
            "levels": [
                1
            ]
        },
        {
            "x": -0.967,
            "y": -2.864,
            "theta": 1.570796,
            "name": "",
            "map": "out",
            "path": "path1",
            "index": 0,
            "type": "floor",
            "levels": [
                1
            ]
        },
        {
            "x": 5.470039,
            "y": -0.08699174,
            "theta": 0.0,
            "name": "",
            "map": "out",
            "path": "path1",
            "index": 1,
            "type": "floor",
            "levels": [
                1
            ]
        },
    ],
    "connections": [
        {
            "_id": {
                "$oid": "60ff74d3d78f798bc92b91d5"
```

```
            },
            // points为此连接所连接的目标点，表示可以通过一定方式在这两个点间切换
            "points": [
                {
                    "map": "office",
                    "path": "path2",
                    "index": 1,
                    "type": "floor",
                    "levels": [
                        1
                    ]
                },
                {
                    "map": "out",
                    "path": "path1",
                    "index": 0,
                    "type": "floor",
                    "levels": [
                        1
                    ]
                }
            ],
            "type": "direct",// 连接的方式，direct为直接连接，如不同地图中的同一个位置
            // elevator为电梯连接，用于跨楼层
            "is_available": true, // 此连接是否可用，当为false时，此连接将被排除于路径规划
            "id": "a05ee41f-86d3-4588-be99-4feb776848d6"
        }
    ]
}
```

添加导航点连接信息

URL: /navigation/point_connections

请求方式: POST

请求参数: |参数|类型|说明| |--|--|--| |points|list|目标点信息| |type|string|连接类型，direct直接连接，表示此点为不同地图中的同一个位置。elevator电梯连接|

请求例子

```
{
    "points": [
        {
             "map": "office", // 目标点所在的地图
             "path": "path2", // 目标点所在的路径
             "index": 1 // 目标点的序号
         },
        {
             "map": "out",
             "path": "path1",
             "index": 0
         }
    ],
    "type": "direct" // 连接方式
}
```

返回参数:

和请求参数一致

返回示例


```
{
  "points": [
    {
      "map": "office",
      "path": "path2",
      "index": 1
    },
    {
      "map": "out",
      "path": "path1",
      "index": 0
    }
  ],
  "type": "direct",
  "is_available": true,
  "id": "a05ee41f-86d3-4588-be99-4feb776848d6",
  "_id": {
    "$oid": "60ff74d3d78f798bc92b91d5"
  }
}
```


删除导航点之间的连接

URL: /navigation/point_connections

请求方式: DELETE

请求参数

名称	类型	说明
id	string	连接的id，可以通过获取连接信息查到

返回参数:

名称	类型	说明
status	string	完成状态

切换当前地图

注意：此切换地图API只是切换当前使用的地图文件。并不是在导航过程中切换地图。导航中切换地图可以使用/navigation/reload API

URL: /navigation/switch_map

请求方式: GET

请求参数:

参数	类型	说明
map_name	string	切换的地图名称

返回参数

```
{
  "status": "ok"
}
```

获取忽略区域信息

URL: /navigation/skip_area

请求方式: GET

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称

返回参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称
skip_area	string	忽略区域信息，每行以空格分隔，四个数据构成一个方格。一行即是一块忽略区域的坐标信息

示例数据:

```
{
    "map_name": "\u529e\u516c\u5ba4",
    "path_name": "1",
    "skip_area": "9.89896392822266 19.9832229614258 12.0267267227173 20.9590129852295\n"
}
```

修改忽略信息

URL: /navigation/skip_area

请求方式: POST

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称
skip_area	string	忽略区域信息，每行以空格分隔，四个数据构成一个方格。一行即是一块忽略区域的坐标信息

示例数据:


```
{
    "map_name": "\u529e\u516c\u5ba4",
    "path_name": "1",
    "skip_area": "9.89896392822266 19.9832229614258 12.0267267227173 20.9590129852295\n"
}
```


返回数据:

返回数据和请求数据一致

删除忽略区域信息

URL: /navigation/skip_area

请求方式: DELETE

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称

返回参数:

参数	类型	说明
status	string	默认为ok

获取虚拟墙信息

URL: /navigation/virtual_wall

请求方式: GET

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称

返回参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称
virtual_wall	string	每行两个数据以空格分隔，分别代表虚拟墙一个点的x和y坐标

示例数据:


```
{
    "map_name": "\u529e\u516c\u5ba4",
    "path_name": "1",
    "virtual_wall": "5.209754 18.43822\n5.2582 18.45059\n5.306646 18.46296\n5.355093 18.47532\n5.403539 18.48769\n5.451986 18.50006\n5.500432 18.51242\n5.548879 18.52479\n5.597325 18.53716\n5.645772 18.54952\n"
}
```


修改虚拟墙数据

URL: /navigation/virtual_wall

请求方式: POST

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称
virtual_wall	string	每行两个数据以空格分隔，分别代表虚拟墙一个点的x和y坐标

示例数据:


```
{
    "map_name": "\u529e\u516c\u5ba4",
    "path_name": "1",
    "virtual_wall": "5.209754 18.43822\n5.2582 18.45059\n5.306646 18.46296\n5.355093 18.47532\n5.403539 18.48769\n5.451986 18.50006\n5.500432 18.51242\n5.548879 18.52479\n5.597325 18.53716\n5.645772 18.54952\n"
}
```


返回数据:

返回数据和请求数据一致

删除虚拟墙数据

URL: /navigation/virtual_wall

请求方式: DELETE

请求参数:

参数	类型	说明
map_name	string	地图名称
path_name	string	路径名称

返回参数:

参数	类型	说明
status	string	默认为ok

设置导航初始位置

URL: /navigation/init_pose

请求方式: POST

请求参数:

参数	类型	说明
x	float	初始位置x坐标
y	float	初始位置y坐标
z	float	初始位置z坐标

返回参数:

参数	类型	说明
status	string	默认为ok

生成多地图导航任务（不执行）

URL: /navigation/multi_map_task

请求参数:

参数	类型	说明
from	object	初始位置对象，包含map,path,index属性。其中map为初始位置所在地图，path为初始位置所在路径，index为初始位置的序号。level为可选参数为初始位置所在楼层
to	object	目标位置对象，其属性和初始位置一致

示例数据

```
{
```

```
  "from": {
    "map": "map1",
    "path": "path1",
    "index": 2,
  },
  "to": {
    "map": "map2",
    "path": "path2",
    "index": 1,
  }
}
```

返回参数:

返回为多地图导航任务对象

执行局部运动任务

说明: 此API会根据对应参数自动创建一个包含LocalMoveAction的任务，然后自动执行

URL: /navigation/start_local_move

请求方式: GET

请求参数:

参数	类型	说明
distance	float	机器人局部运动距离，当distance为正时向前运动，当distance为负时向后运动，单位为米
angle	float	机器人局部运动转向角度，机器人先转动对应角度再直行，单位为弧度
method	int	精准对接辅助手段，0为无，1为使用雷达

返回参数:

返回此任务信息

执行导航和局部运动

说明: 机器人先导航至指定目标点然后执行局部运动，如导航至1号点然后倒退。机器人会自动创建包含一个导航动作和一个局部运动动作的任务。

URL: /navigation/start_nav_and_local_move

请求方式: GET

请求参数:

参数	类型	说明
index	int	目标点index

distance	float	机器人局部运动距离，当distance为正时向前运动，当distance为负时向后运动，单位为米
angle	float	机器人局部运动转向角度，机器人先转动对应角度再直行，单位为弧度
method	int	精准对接辅助手段，0为无，1为使用雷达

返回参数:

返回此任务信息

执行导航任务，局部任务，io任务

说明: 机器人先导航至指定目标点然后执行局部运动动作，然后再设置io电平。如导航到1号点然后倒退，再设置1号端口高电平。机器人会自动创建包含一个导航动作，一个局部运动动作和一个控制io动作的任务。

URL: /navigation/start_nav_local_move_io

请求方式: GET

请求参数:

参数	类型	说明
index	int	目标点index
distance	float	机器人局部运动距离，当distance为正时向前运动，当distance为负时向后运动，单位为米
angle	float	机器人局部运动转向角度，机器人先转动对应角度再直行，单位为弧度
method	int	精准对接辅助手段，0为无，1为使用雷达
level	string	0 为低电平， 1为高电平
port	string	可以为1,2,3分别对应三个IO端口

返回参数:

返回此任务信息

局部移动到指定位置

说明: 调用局部移动Action,局部移动到指定位置

URL: /navigation/start_local_move_target

请求方式: GET

请求参数:

参数	类型	说明
x	float	目标位置x坐标
y	float	目标位置y坐标

theta	float	目标位置角度
frame_id	string	目标位置坐标系,可以是map或base_link
xy_tolerance	float	目标位置x和y坐标允许的误差
yaw_tolerance	float	目标位置角度允许的误差
timeout	float	超时时间，单位为秒
start_rotate_enable	bool	是否允许在起点旋转
target_rotate_enable	float	是否允许在目标点旋转

返回参数:

返回此任务信息

导航到指定目标点

说明: 调用局部移动Action,局部移动到指定目标点位置.可用于精准对接

请求方式: GET

URL: /navigation/start_local_move_index

请求参数:

参数	类型	说明
index	int	目标点index
start_rotate_enable	bool	是否允许在起点旋转
target_rotate_enable	float	是否允许在目标点旋转

返回参数:

返回此任务信息

风向传感器信息

说明: 获取风向传感器信息

请求方式: GET

URL: /navigation/wind_info

请求参数:

无

返回参数:

参数	类型	说明
current_speed	float	当前风速,单位米/秒

max_speed	float	最大风速,单位米/秒
level	float	风速等级
direction	float	风向,单位弧度

获取地图二维码信息

请求方式: GET

URL: /navigation/landmarks

请求参数:

参数	类型	说明
map_name	string	地图名称,可选参数，如果没有此参数则返回所有地图的二维码信息

返回参数:

参数	类型	说明
map_name	string	地图名称
landmarks	list	二维码信息列表

生成覆盖路径

说明: 有些情况下需要机器人路径覆盖特定区域，比如清扫机器人。此API可以生成覆盖路径。

请求方式: POST

URL: /navigation/coverage_path

请求参数:

参数	类型	说明
swath_angle	float	生成路线的角度。单位度，-3 自动, -2 平行最短边，-1 平行最长边。0到180对应手动制定角度
op_width	float	生成路线的行间距，单位米
turn_radius	float	转弯半径，单位米
headland_width	float	专门用于掉头的区域宽度，这块区域通常不耕种，单位米
border	list	多边形边界，x、y两轴分量有效，z轴为0。
holes	list	多边形边界内的洞，代表障碍物之类，暂时不用未来扩展用

返回参数:

参数	类型	说明

status	string	状态
path	list	覆盖路径点列表

示例

请求参数

```
{
    "swath_angle": -3,
    "op_width": 0.5,
    "turn_radius": 0.5,
    "headland_width": 0.5,
    "border": [
        [0, 0, 0],
        [10, 0, 0],
        [10, 10, 0],
        [0, 10, 0],
        [0, 0, 0]
    ],
    "holes": [
        [
            [2, 2, 0],
            [2, 3, 0],
            [3, 3, 0],
            [3, 2, 0],
            [2, 2, 0]
        ]
    ]
}
```

返回参数

```
{
    "status": "ok",
    "path": [
        [0, 0, 0],
        [0, 0.5, 0],
        [0, 1, 0],
        [0, 1.5, 0],
        [0, 2, 0],
        [0, 2.5, 0],
        [0, 3, 0],
        [0, 3.5, 0],
        [0, 4, 0],
        [0, 4.5, 0],
        [0, 5, 0],
        [0, 5.5, 0],
        [0, 6, 0],
        [0, 6.5, 0],
        [0, 7, 0],
        ...
    ]
}
```

获取电梯基本信息

说明: 获取电梯的基本信息，包括电梯的楼层信息，电梯的状态信息

URL: /navigation/get_elevator_info

请求方式: GET

请求参数:

参数	类型	说明
elevator_name	string	电梯名字，可选参数，当没有此参数时返回所有电梯信息

返回参数:

参数	类型	说明
level	int	电梯所在楼层
direction	string	UP,DONW,STOP, 电梯运行方向
state	string	电梯状态，WAITING_OUT, NONE, WAITING_IN
front_door_open	bool	前门是否打开
back_door_open	bool	后门是否打开
name	string	电梯名字

呼叫电梯到指定楼层

说明: 通过此API可以呼叫电梯到指定楼层

URL: /navigation/elevator_to_level

请求方式: GET

请求参数:

参数	类型	说明
elevator_name	string	电梯名字
level	int	目标楼层

返回参数:

参数	类型	说明
result	bool	电梯呼叫是否成功

控制电梯开关门

说明: 通过此API可以控制电梯的前门和后门的开关

URL: /navigation/elevator_open_door

请求方式: GET

请求参数:

参数	类型	说明
elevator_name	string	电梯名字
open	int	0关门，1开门

返回参数:

参数	类型	说明
status	string	ok

任务相关API

机器人的一个动作被定义为Action。比如移动到[1,1]点。又如播放一段声音。一系列的Action组合在一起构成一个任务即Task。通过对Action和Task进行操作，我们可以轻松的控制机器人实现一系列动作。

Task 和 Action 有以下的一些状态

WAITTING 等待执行 WORKING 正在执行 PAUSED 暂停中 CANCELLED 被取消 ERROR 错误 COMPLETE 完成

获取任务信息

URL: /task

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标任务id，可选参数。若无此参数则返回所有的task

返回参数:

task 数据信息

创建任务

URL: /task

请求方式: POST

请求参数:

参数	类型	说明

name	string	创建任务的名称，可以是任意字符串
sub_tasks	list	包含子任务信息的列表，子任务可以是task也可以是action
loop_flag	bool	可选参数，是否自动循环任务

例子:

```
{
    "name": "task",
    "sub_tasks": [
      {
        "type": "nav_action",
        "x": 0,
        "y": 0,
        "theta": 0
      },
      {
        "type": "nav_action",
        "x": 1,
        "y": 1,
        "theta": 1
      },
      {
        "type": "nav_action",
        "x": 2,
        "y": 2,
        "theta": 2
      },
    ],
    "loop_flag": false,
}
```

返回参数:

成功创建的task数据

示例返回数据

```
{
  "name": "test task",
  "loop_flag": false,
  "id": "47a97e4e-9327-4214-a780-fd5a2ae39ee3",
  "state": "WAITTING",
  "sub_tasks": [],
  "progress": 0,
  "current_task": null
}
```

修改任务

URL: /task

请求方式: PUT

请求参数:

参数	类型	说明
id	string	目标任务id
name	string	可选参数，新的任务名称
loop_flag	bool	可选参数，是否循环任务
sub_tasks	list	包含子任务信息的列表

返回参数:

修改后的任务数据

示例返回数据

```
{
  "name": "test task",
  "loop_flag": false,
  "id": "47a97e4e-9327-4214-a780-fd5a2ae39ee3",
  "state": "WAITTING",
  "sub_tasks": [],
  "progress": 0,
  "current_task": null
}
```

删除任务

URL: /task

请求方式: DELETE

请求参数:

参数	类型	说明
id	string	目标删除任务的id

返回参数:

参数	类型	说明
status	string	目标任务是否删除成功

启动任务

URL: /task/start

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标任务的id

返回参数:

启动后的任务信息

暂停任务

URL: /task/pause

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标任务的id

返回参数:

暂停后的任务信息

继续任务

URL: /task/resume

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标任务的id

返回参数:

继续后的任务信息

取消任务

URL: /task/stop

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标任务的id，可选参数。当没有id时则取消所有任务

返回参数:

取消后的任务信息

循环执行任务

URL: /task/loop

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标任务的id
loop_flag	bool	是否循环任务

获取动作Action信息

URL: /action

请求方式: GET

请求参数:

参数	类型	说明
id	string	目标action的id。可选参数，当没有此参数时返回数据库中所有保存的action

返回参数:

目标action数据信息

创建动作Action信息

URL: /action

请求方式: POST

说明:

目前Action有以下几个类别。

callback_action 回调动作

当执行此动作时机器人将会向指定的地址(url),以指定的方式(method)，发送指定的数据(data)

属性	类型	说明
url	string	回调动作地址
method	string	回调请求方法
data	object	需要回调发送到的数据

sleep_action 等待动作

当执行此动作时机器人将等待对应的时间

属性	类型	说明
wait_time	float	等待的时间，单位秒

upload_map_action 上传地图至指定调度服务器动作

执行此任务时机器人将根据调用参数将地图数据上传至指定的调度服务器

属性	类型	说明
server_id	string	调度服务器id

download_map_action 从调度服务器下载地图动作

执行此动作时机器人将根据参数从指定的调度服务器下载地图数据

属性	类型	说明
server_id	string	调度服务器id
map_id	string	下载的地图id

nav_action 导航动作

执行此动作时机器人将移动到参数指定位置

属性	类型	说明
x	float	目标位置x坐标，单位为米
y	float	目标位置y坐标，单位为米
theta	float	目标位置机器人朝向角度，单位为弧度
map	string	目标位置所在地图名称, 可选参数
path	string	目标位置所在路径名称, 可选参数
index	int	目标位置序号, 可选参数
timeout	int	超时时间，可选参数。如果机器人在执行任务是停止超过timeout秒，则任务返回Fail或Error

charge_action 充电动作

在充电桩附近执行此动作机器人将自动对接充电桩并进行充电

属性	类型	说明
x	float	充电桩位置x坐标，单位为米
y	float	充电桩位置y坐标，单位为米

theta	float	充电正面方向，单位为弧度

local_move_action 局部运动动作

局部运动动作用于机器人精准对接过程。比如控制机器人倒车进入车库等等。

属性	类型	说明
distance	float	机器人局部运动距离，当distance为正时向前运动，当distance为负时向后运动，单位为米
angle	float	机器人局部运动转向角度，机器人先转动对应角度再直行，单位为弧度
method	int	精准对接辅助手段，0为无，1为使用雷达

wait_req_action 等待请求动作

等待http请求动作。此动作会一直等待直到超时或者 `/action/update_wait_req` 被调用

属性	类型	说明
timeout	int	超时时间，可选参数

switch_map_action 切换地图动作

属性	类型	说明
map	string	切换的目标地图名称
path	string	切换的目标路径名称
start_index	int	切换起始位置序号，用于地图切换后的位置初始化

change_setting_action 更改系统设置

属性	类型	说明
settings	object	修改的系统参数
init_settings	object	可选参数,当有此参数时，在包含change_setting_action的task执行完毕或取消后自动恢复成init_setting

roslaunch_action 启动ros程序

属性	类型	说明
launch_cmd	string	ros启动指令，如roslaunch xiaoqiang_track xiaoqiang_track.launch

io_action 控制io动作

属性	类型	说明
level	string	0 为低电平， 1为高电平
port	string	可以为1,2,3分别对应三个IO端口

elevator_req_action 电梯请求动作

说明：机器人和电梯交互的Action。可以通过此动作控制电梯上下和开关门

属性	类型	说明
level	int	目标楼层，可选参数，当有此参数时表示机器人要通过电梯去特定楼层。执行action时要保证机器人在电梯内
direction	string	控制电梯外机按钮，可选参数。当为up时自动按下电梯外机上按钮，为down时自动按下电梯下按钮。执行action时要保证机器人在电梯外

当电梯到达目标楼层且开门时此Action进入COMPLETE状态

none_stop_nav_action 不停止导航动作

说明: 到达目标点附近后不停止的导航动作。用于流畅切换目标点，连续执行none_stop_nav_action动作可以实现流畅的连续到达目标点中间不机器人不停止。

属性	类型	说明
x	float	目标点x坐标
y	float	目标点y坐标
radius	float	目标半径，当机器人到达目标点radius范围内，此任务返回complete状态

blockly_action 自动化任务动作

说明: 执行自动化任务的动作。自动化任务中的API模块可以定义通过API触发的行为。当此动作执行时会触发对应的自动化动作。

属性	类型	说明
api_key	string	自动化动作的触发string。注意创建自动化动作时api_key不要设置重复。否则可能会导致部分任务无法触发

video_record_action 录制视频动作

说明: 录制视频动作。执行此动作时机器人会开始录制视频，当任务结束时会自动停止录制。视频保存在机器人主目录的Videos文件夹下。注意当硬盘空间不足的时候，程序会自动删除旧的视频文件，只保留最新的部分内容。

属性	类型	说明
timestamp	int	视频开始的时间戳，视频保存时会采用这个时间戳作为文件名，如1683278160650.mkv

duration	int	视频录制时长，单位为秒。此参数为可选参数，默认为10秒
topic	string	视频录制的topic，此参数为可选参数，默认为/camera_node/image_raw。前摄像头话题为/multi/front/image_raw,后摄像头话题为/multi/back/image_raw
video_url	string	视频录制的地址，如果有此参数上面topic参数将不再生效。程序会自动从此地址下载视频进行保存。注意只支持http协议
filename	str	录音文件名，此参数为可选参数，默认为时间戳

image_action 拍照动作

说明: 拍照动作。执行此动作时机器人会拍照并保存在机器人主目录的Pictures文件夹下。

属性	类型	说明
topic	string	拍照的topic，此参数为可选参数，默认为/camera_node/image_raw。前摄像头话题为/multi/front/image_raw,后摄像头话题为/multi/back/image_raw
timestamp	int	拍照的时间戳，照片保存时会采用这个时间戳作为文件名，如1683278160650.png

audio_record_action 录音动作

说明: 录音动作。执行此动作时机器人会开始录音，录音文件保存至Videos文件夹下。

属性	类型	说明
timestamp	int	录音开始的时间戳，录音保存时会采用这个时间戳作为文件名，如1683278160650.wav
duration	int	录音时长，单位为秒。此参数为可选参数，默认为10秒
filename	str	录音文件名，此参数为可选参数，默认为时间戳

lbt_elevator_req_action

说明: 鲁邦通梯控请求动作

属性	类型	说明
from_level	int	起始楼层
to_level	int	目标楼层
cmd	string	enter或者leave，表示现在是进电梯请求还是出电梯请求

lbt_elevator_ack_action

说明: 鲁邦通梯控动作执行完成后的响应动作

slide_action

说明: 滑台移动动作

属性	类型	说明
mode	int	0 点动模式，1 位置模式， 2 回零, 3 百分比模式
speed	float	运行速度，单位米/秒，0值表示用默认值，点动模式位置可以取负值，位置模式取值范围0到正无穷，回零模式速度最小0.05，内部有限位保护。
time	float	点动模式运行时间，单位秒，其它模式时无效，取值范围0.1到正无穷，内部有限位保护。
pose	float	位置模式下相对零点的位置，单位米；点动模式下，这个值无效。取值范围不限，内部有限位保护。0值表示停止不动。百分比模式下，取值范围0到1。

jacking_action

说明: 顶升移动动作

属性	类型	说明
mode	int	0 点动模式，1 位置模式， 2 回零, 3 百分比模式
speed	float	运行速度，单位米/秒，0值表示用默认值，点动模式位置可以取负值，位置模式取值范围0到正无穷，回零模式速度最小0.05，内部有限位保护。
time	float	点动模式运行时间，单位秒，其它模式时无效，取值范围0.1到正无穷，内部有限位保护。
pose	float	位置模式下相对零点的位置，单位米；点动模式下，这个值无效。取值范围不限，内部有限位保护。0值表示停止不动。百分比模式下，取值范围0到1。

jacking_rotate_action

说明: 顶升旋转动作

属性	类型	说明
mode	int	0 点动模式，1 位置模式， 2 回零, 3 百分比模式
speed	float	运行速度，单位弧度/秒，0值表示用默认值
time	float	点动模式运行时间，单位秒，其它模式时无效，取值范围0.1到正无穷，内部有限位保护。
angle	float	位置模式下相对零点的位置，单位弧度
frame_id	string	坐标系，可以是map或者base_link

local_move_target_action

说明: 局部运动到指定位置动作

属性	类型	说明

x	float	目标位置x坐标
y	float	目标位置y坐标
theta	float	目标位置角度
frame_id	string	目标位置坐标系,可以是map或base_link
xy_tolerance	float	目标位置x和y坐标允许的误差
yaw_tolerance	float	目标位置角度允许的误差
timeout	float	超时时间，单位为秒
start_rotate_enable	bool	是否允许在起点旋转
target_rotate_enable	float	是否允许在目标点旋转

tts_action

说明: 语音合成动作

属性	类型	说明
text	string	合成的文本

onvif_camera_move_action

说明: onvif摄像头移动动作

属性	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口
username	string	摄像头用户名
password	string	摄像头密码
x	float	摄像头横向转动角度
y	float	摄像头纵向转动角度
zoom	float	摄像头缩放比例

onvif_camera_snapshot_action

说明: onvif摄像头拍照动作

属性	类型	说明
ip	string	摄像头IP地址，可以为空字符串。当为空字符串时就使用从局域网内找到的第一个摄像头ip
port	int	摄像头端口

username	string	摄像头用户名
password	string	摄像头密码
channel	int	摄像头图像通道,可选参数，没有就默认为1通道

创建action

请求参数:

参数	类型	说明
type	string	action类型，必须是以上几种aciton的一个
task_id	string	创建的action所属的task,可选参数。当有此参数时将新建action加入对应task，没有此参数时自动创建一个新的临时task(task不会被保存至数据库)并将action添加进task

其他参数和对应的Action类型相关

示例请求参数:

```
{
    "type": "sleep_action",
    "wait_time": 2
}
```

返回参数:

包含新创建的action的task

示例返回参数:

```
{
  "name": "auto task",
  "loop_flag": false,
  "id": "c50d355f-9155-4a17-ade6-a61b40820b43",
  "state": "WAITTING",
  "sub_tasks": [
    {
      "wait_time": 2,
      "state": "WAITTING",
      "type": "sleep_action",
      "id": "3bbbed8f-c0dc-4a94-8d6f-05006c943818"
    }
  ],
  "progress": 0,
  "current_task": null
}
```

修改动作Action信息

URL: /action

请求方式: PUT

请求参数:

参数	类型	说明
id	string	目标action的id

其他参数和Action类型相关

返回参数:

经过修改的Action数据

删除动作Action信息

URL: /action

请求方式: DELETE

请求参数:

参数	类型	说明
id	string	目标action的id

触发等待动作

说明: 对于 `wait_req_action`，机器人会一直等待http请求触发，直到触发后才继续执行下面的任务。

URL: /action/update_wait_req

请求方式: GET

请求参数: 无

返回参数:

参数	类型	说明
status	string	成功为OK

自动化任务API

获取自动化任务信息

URL: /blockly_task

请求方式: GET

请求参数: 无

返回参数:

所有的自动化任务，包括在执行的和未在执行的任务

创建自动化任务

URL: /blockly_task

请求方式: POST

请求参数:

参数	类型	说明
py	string	python程序内容，在后端自动化执行的程序。使用客户端时会自动把图形化程序转化成python程序
blocklyxml	string	图形化程序内容
title	string	自动化任务标题
description	string	自动化任务描述
is_enabled	bool	是否启用自动化任务

返回参数:

创建的自动化任务信息

修改自动化任务

URL: /blockly_task

请求方式: PUT

请求参数:

参数	类型	说明
id	string	目标自动化任务的id
py	string	python程序内容,可选参数，如果没有此参数则保持原有内容不变
blocklyxml	string	图形化程序内容,可选参数，如果没有此参数则保持原有内容不变
title	string	自动化任务标题,可选参数，如果没有此参数则保持原有内容不变
description	string	自动化任务描述,可选参数，如果没有此参数则保持原有内容不变
is_enabled	bool	是否启用自动化任务,可选参数，如果没有此参数则保持原有内容不变

返回参数:

修改后的自动化任务信息

删除自动化任务

URL: /blockly_task

请求方式: DELETE

请求参数:

参数	类型	说明
id	string	目标自动化任务的id

返回参数:

参数	类型	说明
status	string	目标自动化任务是否删除成功

行为树相关API

行为树API是在上面任务API的基础上进行扩展的API。因为在实际使用任务API的时候，我们会遇到很多无法满足要求的情况。比如以上的任务API是没有逻辑判断的，我们无法让机器人在某些情况下执行一个动作，在另一个情况下执行另一个动作。

什么是行为树

行为树是一种用于控制机器人行为的树形结构。机器人在执行行为树的时候，会从根节点开始，根据条件判断执行对应的行为，然后再根据行为执行结果执行下一个行为。

行为树分为控制节点和执行节点。控制节点就类似于之前的Task，用于对具体执行过程的控制。执行节点就类似于之前的Action，用于具体的执行动作。所有的节点的可能状态和Task一样，只不过Task状态基础上增加了FAIL状态。当节点执行失败时返回Fail状态。注意Fail状态和Error状态不一样。出现Fail状态时行为树会根据状态执行不同的流程。但是Error状态会导致行为树停止执行。

下面以一个例子作为说明

假设我们需要机器人乘坐电梯从一楼到二楼。使用Task我们的大致流程如下。

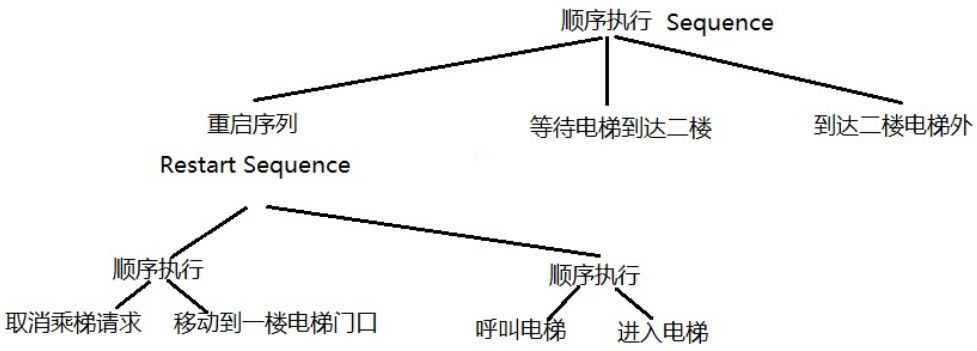
到达一楼电梯门口 -> 呼叫电梯 -> 进入电梯等待电梯到达二楼 -> 出电梯，到达二楼电梯门口

在实际使用中还需要考虑到电梯被人或货物占用机器人无法进入的情况。

更完善的流程如下

到达一楼电梯门口 -> {呼叫电梯 -> 进入电梯} -> 电梯被占用 -> {取消电梯呼叫 -> 回到一楼电梯外} -> {呼叫电梯 -> 进入电梯} -> 电梯被占用 -> {取消电梯呼叫 -> 回到一楼电梯外} -> ... -> 成功进入电梯 -> 等待电梯到达二楼 -> 出电梯，到达二楼电梯门口

这样我们可以重新组织一下



上面的行为树用到了以下几个控制节点。首先是顺序执行节点，此控制节点顺序执行自己的子任务。然后是重启序列节点，此节点会顺序执行所有子节点，如果有子节点失败则从头开始重新执行自己的子节点。

现在我们来看一下行为树的执行过程。

顺序执行节点 -> 重启序列节点 -> 顺序执行节点 -> 取消乘梯请求 -> 移动到一楼电梯门口 -> 顺序执行节点 -> 呼叫电梯 -> 进入电梯 -> 如果进入电梯失败，则重启序列节点会重新执行第一个顺序执行节点。也就是 取消乘梯请求->移动到一楼电梯门口，之后机器人会重新叫电梯，进入电梯。直到进入电梯成功。

从上面的例子可以看到，通过合理的组合各种节点类型，我们就可以让机器人在各种复杂的情况下做出正确的判断和动作。

当前支持的行为树控制节点

顺序（sequence）节点

control_type: sequence

顺序执行所有子节点，如果任意一个子节点返回Fail则返回Fail，如果任意子节点返回Error则返回Error，如果所有子节点都返回Complete则返回Complete

并行（parallel）节点

control_type: parallel

参数	类型	说明
success_count	int	并行执行的任务成功数超过success_count时状态变为COMPLETE

并行执行所有子节点，直到第success_count个子节点返回Complete。如果success_count为0则需要所有子节点都返回Complete才返回Complete。

fallback节点

control_type: fallback

包含两个子节点，首先执行第一个子节点，如果第一个子节点返回Fail则执行第二个子节点，如果第二个子节点返回Fail则返回Fail。

反转（flip）节点

control_type: flip

包含一个子节点，执行子节点，如果子节点返回Fail则返回Complete，如果子节点返回Complete则返回Fail，如果子节点返回Error则返回Error。

循环节点（loop）节点

control_type: loop

参数	类型	说明
target_loop_count	int	目标循环次数

循环依次执行所有子节点, 直到循环次数达到target_loop_count。如果target_loop_count为0则无限。如果有子节点返回Fail则返回Fail。

强制失败（force_fail）节点

control_type: force_fail

包含一个子节点，执行子节点，如果子节点返回Fail则返回Fail，如果子节点返回Complete则返回Fail，如果子节点返回Error则返回Error。

强制成功（force_success）节点

control_type: force_success

包含一个子节点，执行子节点，如果子节点返回Fail则返回Complete，如果子节点返回Complete则返回Complete，如果子节点返回Error则返回Error。

重启顺序（restart_sequence）节点

control_type: restart_sequence

依次执行所有子节点，如果子节点返回Fail则从头开始执行子节点，如果所有子节点返回Complete则返回Complete，如果子节点返回Error则返回Error。

重试顺序（retry_sequence）节点

control_type: retry_sequence

依次执行所有子节点，如果子节点返回Fail则重新执行此子节点，如果所有子节点返回Complete则返回Complete，如果子节点返回Error则返回Error。

重试到成功（retry_until_success）节点

control_type: retry_until_success

只能有一个子节点，如果子节点返回Fail则重新执行此子节点，直到子节点返回Complete。

创建和控制行为树

行为树的创建方式和Task一样。只是在Task的属性上增加了control_type，同时根据不同的control_type增加了不同的属性。下面是一个创建的例子

```
# 创建任务
task = requests.post(
    "http://127.0.0.1:3546/api/v1/task",
    json={
        "name": "test_sequence_task",
        "control_type": "sequence",
        "sub_tasks": [
            {
                "type": "nav_action",
                "map": "1",
                "path": "1",
                "index": 1,
                "x": 4.686,
                "y": -1.196,
                "theta": 0,
            },
            {
                "type": "nav_action",
                "map": "1",
                "path": "1",
                "index": 1,
                "x": -0.45657,
                "y": -3.551,
                "theta": 1.94446,
            },
            {
                "type": "charge_action",
                "x": -0.45657810000000004,
                "y": -3.551895,
                "theta": 1.94446,
            },
        ],
    },
)
# 启动任务
requests.get("http://127.0.0.1:3546/api/v1/task/start?id=" + task.json()["id"])
print(json.dumps(task.json(), indent=4))
```

上面的例子创建了一个序列控制节点，包含了三个子节点。机器人会依次执行三个节点。序列控制节点的效果和task基本是一样的。

上面的程序正常应该会有以下输出


```
{
    "id": "7976cd0d-fa77-485a-a1a3-af93fccad96f",
    "name": "test_sequence_task",
    "current_task": null,
    "state": "WAITTING",
    "sub_tasks": [
        {
            "type": "nav_action",
            "id": "e664c2dc-5168-40a5-9977-c5dbaf53d609",
            "x": 4.686,
            "y": -1.196,
            "theta": 0,
            "state": "WAITTING",
            "result": "",
            "index": 1,
            "map": "1",
            "path": "1",
            "progress": 0,
            "current_location": {
                "x": -1,
                "y": -1,
                "theta": -1
            },
            "current_distance": -1,
            "timeout": 0
        },
        {
            "type": "nav_action",
            "id": "4cdd1711-211a-4119-8384-e3de4ec5ad07",
            "x": -0.45657,
            "y": -3.551,
            "theta": 1.94446,
            "state": "WAITTING",
            "result": "",
            "index": 1,
            "map": "1",
            "path": "1",
            "progress": 0,
            "current_location": {
                "x": -1,
                "y": -1,
                "theta": -1
            },
            "current_distance": -1,
            "timeout": 0
        },
        {
            "type": "charge_action",
            "id": "71d60425-fef0-4577-a2c3-92f1ab05ebac",
            "x": -0.45657810000000004,
            "y": -3.551895,
            "theta": 1.94446,
            "state": "WAITTING",
            "result": "",
```

```
            "progress": 0
        }
    ],
    "progress": 0,
    "control_type": "sequence"
}
```

下面是一个更复杂的例子

```
task = requests.post(
  "http://127.0.0.1:3546/api/v1/task",
  json={
      "name": "test_nested_task",
      "control_type": "sequence",
      "sub_tasks": [
          {
              "name": "s1",
              "control_type": "sequence",
              "sub_tasks": [
                  {
                      "name": "s11",
                      "control_type": "sequence",
                      "sub_tasks": [
                          {
                              "type": "sleep_action",
                              "wait_time": 5,
                          },
                          {
                              "type": "tts_action",
                              "text": "测试",
                          },
                      ],
                  },
                  {
                      "name": "s12",
                      "control_type": "sequence",
                      "sub_tasks": [
                          {
                              "type": "sleep_action",
                              "wait_time": 5,
                          },
                          {
                              "type": "tts_action",
                              "text": "测试",
                          },
                      ],
                  },
              ],
          },
          {
              "name": "s2",
              "control_type": "sequence",
              "sub_tasks": [
                  {
                      "name": "s21",
                      "control_type": "sequence",
                      "sub_tasks": [
                          {
                              "type": "sleep_action",
```

```
                            "wait_time": 5,
                        },
                        {
                            "type": "tts_action",
                            "text": "测试",
                        },
                    ],
                },
                {
                    "name": "s22",
                    "control_type": "sequence",
                    "sub_tasks": [
                        {
                            "type": "sleep_action",
                            "wait_time": 5,
                        },
                        {
                            "type": "tts_action",
                            "text": "测试",
                        },
                    ],
                },
            ],
        },
    ],
  },
)
requests.get("http://127.0.0.1:3546/api/v1/task/start?id=" + task.json()["id"])
print(json.dumps(task.json(), indent=4))
```

上面的例子中包含了嵌套的行为树。机器人会依次执行s1,s2两个子任务。s1,s2两个子任务又会依次执行s11,s12和s21,s22两个子任务。最终机器人会依次执行s11,s12,s21,s22四个子任务。

对于暂停，取消，继续等操作和Task的API是一样的。

黑板参数

在有些情况下我们需要节点之间共享参数。比如语音识别节点的结果可以作为输出参数给语音播放节点使用。这时我们就可以使用黑板参数。黑板参数是一种全局的参数，所有节点都可以访问。下面是一个使用黑板参数的例子。

```
task = requests.post(
    "http://127.0.0.1:3546/api/v1/task",
    json={
        "name": "test_blackboard",
        "control_type": "sequence",
        "blackboard": {"test": "测试黑板数据"},
        "sub_tasks": [
            {
                "type": "tts_action",
                "text": "不使用黑板数据",
            },
            {
                "type": "sleep_action",
                "wait_time": 5,
            },
```

```
            {
                "type": "tts_action",
                "text": "{test}",
            },
        ],
    },
)
requests.get("http://127.0.0.1:3546/api/v1/task/start?id=" + task.json()["id"])
print(json.dumps(task.json(), indent=4))
```

所有的节点都共享一个黑板参数。在上面的例子中，我们在创建任务的时候传递了一个黑板参数。黑板参数是一个json对象。在tts_action节点中我们可以使用 `{test}` 的方式来使用黑板参数中的数据。在上面的例子中，机器人会先说出“不使用黑板数据”，然后等待5秒，最后说出“测试黑板数据”。在使用黑板参数变量的时候用大括号括起来即可。

Websocket相关API

对于需要高频率获取的数据我们提供了websocket api。比如我们可能需要很高频率的获取机器人的位置，速度等信息。

websocket默认端口3547

URL格式: ws://192.168.0.132:3547/topicName?token=28b1c500400611ebb805493c9303c705

其中192.168.0.132为机器人IP,28b1c500400611ebb805493c9303c705为机器人token。在开启token验证时此参数是必须的，反之则不需要。

topicName为数据对应的ros话题名称。理论上可以订阅机器人内部所有的ros话题。返回的数据为ros话题数据转换成的json数据

`注意在开启websocket连接后要定期发送ping消息，否则连接可能会自动断开。比如可以每秒发送一个ping消息。`

对于6.2.1及以后版本websock也可以用于向机器人发送ros话题。具体数据结构可以先打印订阅的话题参考一下。发送的数据结构和接收的数据结构一样。

下面是常用的接口

获取GalileoStatus

GalileoStatus数据定义可以参照串口api中GalileoStatus的说明。

topicName: /galileo/status

返回数据为json

获取温湿度及可燃气体数据

TopicName: /bw_env_sensors/EnvSensorData

返回数据

```
float temperature; // 温度，单位摄氏度
float rh; // 相对湿度 %RH
float smoke; // 烟雾 ppm
float pm1_0; // pm1.0 ug/m^3
float pm2_5; // pm2.5 ug/m^3
float pm10; // pm10 ug/m^3
float lel; // 可燃气体 ppm
float noise; // 噪声 db
```

获取光照度和空气传感器数据

TopicName: /bw_env_sensors/AirSensorData

返回数据

```
float temperature; // 温度，单位摄氏度
float rh; // 相对湿度 %RH
float co2; // 烟雾 ppm
float illuminance; // lux
```

对于跨地图导航的说明

在日常使用中，我们经常会遇到需要跨地图导航的情况。比如在不同楼层间进行导航，在不同的仓库区域中进行导航。Galileo API针对这种场景也进行了专门优化。具体的使用方法如下。

我们以跨楼层导航为例，首先我们要创建不同楼层的地图。然后在每个楼层中分别绘制导航点和导航路径。保证机器人可以在各个楼层正常使用。假设一层的地图为map1，二层的地图为map2。一层二层之间以电梯相连接。

为了使机器人能够通过电梯跨楼层，我们需要在地图中添加电梯信息。在一层和二层的电梯门口分别添加添加一个导航点。比如，一层中的电梯导航点为1号点，二层中的电梯导航点为0号点。然后我们需要给这两个点添加连接信息，告诉机器人我们可以通过一定方式从map1中的1号点到达map2中的0号点。

下面是一个具体的连接数据例子

```
{
    "points": [
        {
            "x": 1.624922, // 导航点 x 坐标
            "y": -0.372011, // 导航点 y 坐标
            "theta": 0.0, // 导航点角度
            "name": "", // 导航点名称
            "map": "map1", // 导航点所在地图名称
            "path": "path1", // 导航点所在地图路径
            "index": 0 // 导航点序号
        },
        {
            "x": 0.8710775,
            "y": -0.09214968,
            "theta": -0.2962013,
```

```
            "name": "",
            "map": "map1",
            "path": "path1",
            "index": 1
        },
        {
            "x": -0.967,
            "y": -2.864,
            "theta": 1.570796,
            "name": "",
            "map": "map2",
            "path": "path1",
            "index": 0
        },
        {
            "x": 5.470039,
            "y": -0.08699174,
            "theta": 0.0,
            "name": "",
            "map": "map2",
            "path": "path1",
            "index": 1
        },
    ],
    "connections": [
        {
            "_id": {
                "$oid": "60ff74d3d78f798bc92b91d5"
            },
            // points为此连接所连接的目标点，表示可以通过一定方式在这两个点间切换
            "points": [
                {
                    "map": "map1",
                    "path": "path1",
                    "index": 1
                },
                {
                    "map": "map2",
                    "path": "path1",
                    "index": 0
                }
            ],
            "type": "direct",// 连接的方式，direct为直接连接，如不同地图中的同一个位置
            // elevator为电梯连接，用于跨楼层
            "is_available": true, // 此连接是否可用，当为false时，此连接将被排除于路径规划
            "id": "a05ee41f-86d3-4588-be99-4feb776848d6"
        }
    ]
}
```

机器人在知道这些信息之后，我们通过api向机器人发布目标点，机器人就会自动根据连接信息切换地图并移动至目标点。

比如机器人现在在map1的0号点，我们需要机器人移动至map2中的1号点。机器人就会自动先移动到map1中的1号点。然后通过电梯移动至二层区域，移动至map2中的0号点，并切换至map2地图。然后移动至map2中的1号点。

机器人跨地图自动规划程序会考虑连接的可用性，总的移动距离，切换地图的次数，经过的目标点数等等细信息，最终为用户提供一个最优的路线。 地图的自动切换也并不仅限于两个，实际上只要地图之间有合适的连接信息，程序会自动的搜索最优的切换方式。比如map1和map2之间有连接，map2和map3之间有连接。那么机器人就会在运动过程中自动的由map1切换至map2然后切换至map3。

实际调用API时需要两个接口。`/navigation/point_connections` 用于添加点之间的连接信息。`/navigation/start_nav_task` 用于发布导航任务，注意在调用时添加map和path参数。详细说明请查看对应API文档。目前支持的连接方式只有direct连接，电梯支持尚在开发之中。

此程序会调用获取机器人当前电量，机器人当前位置，控制机器人向前移动0.5m的API。注意把代码里面的机器人IP换成自己的机器IP

- 伽利略视觉导航系统ROS通信协议
  - 1.命令发布话题 /galileo/cmds
  - 2.伽利略视觉导航系统状态话题 /galileo/status
  - ROS中控制导航系统的例子
    - 开启导航系统

伽利略视觉导航系统ROS通信协议

除了使用windows客户端或者串口来控制、使用伽利略视觉导航系统外，还可以通过ros的topic机制来实现。

在伽利略视觉导航系统串口通信协议里面，导航串口可以发送和接收串口数据，这是通过galileo_serial_server包实现的，而这个包的实现原理就是将导航串口接收的数据转换成ros话题后以 `/galileo/cmds` 话题发布给伽利略系统，同时通过订阅伽利略系统状态话题 `/galileo/status` 然后封装后下发给导航串口。

因此在ros系统里面，我们可以绕开串口直接发布 `/galileo/cmds` 话题就可以控制伽 利略系统，直接订阅 `/galileo/status` 就可以获取伽利略系统状态。

1.命令发布话题 /galileo/cmds

伽利略视觉导航系统会实时订阅这个话题，给这个话题发送数据就可以对应控制伽利略系统的状态。

这个话题所属类型为 `galileo_msg::GalileoNativeCmds` ，具体定义在 `/home/xiaoqiang/Documents/ros/src/galileo_msg/msg/GalileoNativeCmds.msg` 文件。

话题内容有三个成员：header、length、data。这个话题是为了封装串口数据设计的，在《伽利略视觉导航系统串口通信协议》里面，串口指令是由“包头+数据长度+数据内容”构成的。

因此根据串口通信协议，把包头用标准的std_msgs/Header替换成header，数据长度赋 值给length，数据内容赋值给data，就可以构造出有效的/galileo/cmds话题。

例如，下面cpp代码将构造一个开启导航系统的话题数据，数据内容请参考串口通信协议。

```
#include "galileo_msg/GalileoNativeCmds.h"
galileo_msg::GalileoNativeCmds currentCmds;
currentCmds.header.stamp = ros::Time::now();
currentCmds.header.frame_id = "galileo_msg";
currentCmds.length = 0x02;
currentCmds.data.resize(0x02);
currentCmds.data[0] = 0x6d;
currentCmds.data[1] = 0x00;
```

2.伽利略视觉导航系统状态话题 /galileo/status

伽利略视觉导航系统会持续自动发布 `/galileo/status` 话题，订阅这个话题查询系统状态。

这个话题所属类型为 `galileo_msg::GalileoStatus` ，具体定义
在 `/home/xiaoqiang/Documents/ros/src/galileo_msg/msg/GalileoStatus.msg` 文件。

```
std_msgs/Header header
int32 navStatus
int32 visualStatus
int32 chargeStatus
int32 loopStatus
float32 power
int32 targetNumID
int32 targetStatus
float32 targetDistance
int32 angleGoalStatus
float32 controlSpeedX
float32 controlSpeedTheta
float32 currentSpeedX
float32 currentSpeedTheta
```

本话题里面数据成员的意义请参考《伽利略视觉导航系统串口通信协议》中第一部分“导航串口下发的数据包”中的内容定义。

使用 `rostopic echo /galileo/status` 可以直接打印输出话题内容。

ROS中控制导航系统的例子

开启导航系统

首先查阅串口协议文档可知，开启导航对应的指令为 `m 0` ,这是一个两个字节的指令。则可以在终端输入以下指令开启导航

```
rostopic pub /galileo/cmds galileo_msg/GalileoNativeCmds "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
length: 2
data: [0x6d, 0]" -1
```

其中 length为指令长度，data为指令数据。0x6d对应字符'm'。 有一个比较简单的输入方法先输入 `rostopic pub /galileo/cmds` 之后双击tab键自动补全

```
rostopic pub /galileo/cmds galileo_msg/GalileoNativeCmds "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
length: 2
data: ''"
```

会出现上面的内容。对data进行修改就可以发布对应的指令了。

- 伽利略视觉导航系统串口通信协议
  - 1. 导航串口下发的数据包
  - 导航串口可以接收的指令
    - 开启、关闭、重载导航系统
    - 发布下一个目标点
    - 暂停、继续、取消当前目标点
    - 手动遥控速度指令
    - 主机关机指令
    - 自动巡检状态控制
    - 调度系统导航控制
    - 开始建立地图，结束建立地图,保存和更新地图
    - 自动充电相关指令
    - 迎宾模式
  - 版本历史

伽利略视觉导航系统串口通信协议

导航系统主机带一个usb转串口模块(可以选ttl电平或者rs232电平，默认ttl电平)，通过这个串口用户可以使用导航系统的功能和获取导航系统的状态，下文将这个串口命名为导航串口。

导航串口波特率为115200，8个数据位，1个停止位，无奇偶校验。

1. 导航串口下发的数据包

发布频率固定为30hz，即导航系统每秒自动通过导航串口下发30个数据包。

数据包格式：包头+长度+数据内容+结束符0x00

包头：占3个字节，0xcd 0xeb 0xd7

长度：占1个字节，长度不包括包头和长度本身字符，这个长度还包括字符串结束符0x00，当前值固定为21*4+1=85=0x55。

内容：由12个4字节小端模式二进制表示的数字串联在一起构成。

包头	长度	nav_status	visual_status	map_status	…	数据n	…	busy_
0xcd0xeb0xd7	0x55	4个字节	4个字节	4个字节	…	4个字节	…	4个

完整数据包内容构成一个c语言结构体，结构体具体构成如下所示：

```
typedef struct{
```

```
  int nav_status;// 导航服务状态，0表示没开启closed，1表示开启opened。
  int visual_status;// 视觉系统状态，-1标系视觉系统处于关闭状态，0表示没初始化uninit，1表示正在追踪tracking,2表示丢失lost,1和2都表示视觉系统已经初始化完成。
  int map_status; // 建图服务状态，0表示未开始建图，1表示正在建图
  int gc_status; // 内存回收标志，0表示未进行内存回收，1表示正在进行内存回收
  int gba_status; // 闭环优化标志，0表示未进行闭环优化，1表示正在进行闭环优化
  int charge_status; // 充电状态，0 free 未充电状态，1 charging 充电中，2 charged 已充满，但仍在小电流充电，3 finding
                     // 寻找充电桩，4 docking 停靠充电桩，5 error 错误
  int loop_status; // 是否处于自动巡检状态，1为处于，0为不处于。
  float power;// 电源电压【946】v。
  int target_numID;// 当前目标点编号,默认值为-1表示无效值，当正在执行无ID的任务是值为-2，比如通过Http API 创建的导航任务。
  int target_status;// 当前目标点状态，0表示已经到达或者取消free，1表示正在前往目标点过程中working,2表示当前目标点的移动任务被暂停paused,3表示目标点出现错误error,默认值为-1表示无效值。
  float target_distance;// 机器人距离当前目标点的距离，单位为米，-1表示无效值，该值的绝对值小于0.01时表示已经到达。
  int angle_goal_status;// 目标角度达到情况，0表示未完成，1表示完成，2表示error,默认值为-1表示无效值。

  float control_speed_x;// 导航系统计算给出的前进速度控制分量,单位为m/s。
  float control_speed_theta;// 导航系统计算给出的角速度控制分量,单位为rad/s。
  float current_speed_x;// 当前机器人实际前进速度分量,单位为m/s。
  float current_speed_theta;// 当前机器人实际角速度分量,单位为rad/s。
  unsigned int time_stamp;// 时间戳,单位为1/30毫秒，用于统计丢包率。对于ROS API时间戳在状态的header里面
  float current_pose_x; // 当前机器人在map坐标系下的X坐标,此坐标可以直接用于设置动态插入点坐标
  float current_pose_y; // 当前机器人在map坐标系下的Y坐标
  float current_angle; // 当前机器人在map坐标系下的z轴转角(yaw)
  int busy_status; //当busy为true时系统将仍然后接收新指令，但是不会立即处理。当系统退出busy状态后再处理消息
}Galileo_Status;
```

导航串口可以接收的指令

最大支持100hz上传频率，每条命令由包头+数据长度+数据内容构成。指令可以 串联一起发送但是不推荐这样使用，因为每条指令是否有效还要取决于导航系统状态。

开启、关闭、重载导航系统

0xcd	0xeb	0xd7	0x02	0x6d	0xXX
包头	包头	包头	数据长度	“m”	值为4表示关闭，0表示为开启

通过本命令可以开启、关闭导航系统，导航系统开启后需要对视觉系统进行初始化，视觉系统初始化过程机器人可能会原地旋转（当前视角无法初始化时需要调整视角），视觉初始化后才能使用导航系统的功能。

因为视觉系统初始化时间较长，所以不建议频繁地进行关闭、开启导航系统操作。导航系统的状态可以由nav_status获得、视觉系统的状态可以由visual_status获得。

现在导航系统支持地图的动态切换，不再需要重启导航系统便可以载入不同的导航地图。同时地图也支持自动切换，导航系统可根据当前环境自动切换至对应的地图。在地图切换后需要重新载入对应地图的导航路径，此时即可用到重载功能。

0xcd	0xeb	0xd7	0x02	0x6d	0xXX
包头	包头	包头	数据长度	“m”	9

重载时不会对导航地图进行处理，只重新载入当前的路径。所以调用重载之前需要设置好地图和路径。

发布下一个目标点

0xcd	0xeb	0xd7	0x02	0x67	0xXX
包头	包头	包头	数据长度	“g”	值为0到255，表示目标点编号

发布前需要检查视觉系统有没有完成初始化，如果没有完成初始化，系统会忽略这条指令。

视觉系统如果已经初始化了，主机接收这条指令后，先根据当前目标点状态判断是否要先执行取消当前移动任务的操作。当前目标点状态为free或者error时不执行任何动作直接继续下一步，当前目标点状态为working或paused时会自动取消当前目标点然后继续下一步。

接着系统会将当前目标点编号更新成设定值，再验证目标点编号是否有效。

如果目标点有效，立即控制机器人往目标点移动，同时目标点的状态更新成working

如果目标点无效，反馈的目标点状态更新成error，同时机器人保持不动

目标到达后，目标点状态会自动更新成free，机器人退出移动任务同时保持不动

暂停、继续、取消当前目标点

0xcd	0xeb	0xd7	0x02	0x69	0xXX
包头	包头	包头	数据长度	“i”	值为0表示暂停，值为1表示继续，值为2表示取消

发布暂停指令后，系统会检查目标点状态，状态为working时本命令才有效，为其它状态则忽略。指令判断有效后机器人会停止运动、保存移动任务同时目标点状态会更新成paused。

发布继续指令后，系统会检查目标点状态，状态为pause时本命令才有效，为其它状态则忽略。指令判断有效后机器人继续往目标点运动同时反馈的目标点状态变成working，

发布取消指令后，系统会检查目标点状态，状态不时free时本命令才有效，为其它状态则忽略。指令判断有效后机器人停止运动，退出移动任务,目标状态更新成free。

移动到动态目标点

添加目标点

0xcd	0xeb	0xd7	0x0a	0x67	0x69	0xXX	0xXX	0xXX	0xXX	0xX
包头	包头	包头	数据长度	“g”	"i"	x，y，theta的float32坐标，共4 * 3Byte				

机器人移动到x,y坐标，角度为theta

手动遥控速度指令

0xcd	0xeb	0xd7	0x02	0x66	0xXX
包头	包头	包头	命令长度	前进指令	速度大小，为最大速度百分比，数值范围为0到100

0xcd	0xeb	0xd7	0x02	0x62	0xXX
包头	包头	包头	命令长度	后退指令	速度大小，数值范围为0到100

0xcd	0xeb	0xd7	0x02	0x63	0xXX
包头	包头	包头	命令长度	左转指令	速度大小，数值范围为0到100

0xcd	0xeb	0xd7	0x02	0x64	0xXX
包头	包头	包头	命令长度	右转指令	速度大小，数值范围为0到100

0xcd	0xeb	0xd7	0x02	0x73	0xXX
包头	包头	包头	长度	停止指令	制动量大小，数值范围为0到100

可以控制机器人的移动，这些指令独立于视觉导航系统，导航系统工不工作都会执行这些指令，因此可以用来实现机器人的遥控功能。

这些指令生效后不会影响目标点状态，即遥控的速度指令和导航系统的速度指令是并行的会相互覆盖。如果导航运动过程中机器人发生错误需要切换到遥控，用户应该先发布目标点取消指令，否则可能会出现机器人还是不受自己控制的情况。

主机关机指令

0xcd	0xeb	0xd7	0x02	0xaa	0x44
包头	包头	包头	数据长度	数据1	数据2

主机已经配置成通电后自动开机，这样可以避免按键的不方便。使用本条指令可以实现串口关机。

自动巡检状态控制

自动巡检功能即为机器人按照当前所有的目标点循环移动。在到达目标点时停靠特定时间，此时间可以通过api进行设置。

开启自动巡检功能

注意只有在开启导航服务之后才可以启动自动巡检功能。当此功能成功开启之后galileo_status中的loopFlag会设置成True。

0xcd	0xeb	0xd7	0x02	0x6d	0x05
包头	包头	包头	数据长度	“m”	5

关闭自动巡检功能

0xcd	0xeb	0xd7	0x02	0x6d	0x04
包头	包头	包头	数据长度	“m”	6

设置自动巡检目标点停靠时间

0xcd	0xeb	0xd7	0x03	0x6d	0x05	0xXX
包头	包头	包头	数据长度	“m”	5	目标点停靠时间，单位为秒

调度系统导航控制

调度导航状态是配合拉格朗日调度系统使用的导航状态，在此状态下机器人的导航任务由调度系统发布。

开启调度导航

0xcd	0xeb	0xd7	0x02	0x6d	0x07
包头	包头	包头	数据长度	“m”	7

关闭调度系统导航

0xcd	0xeb	0xd7	0x02	0x6d	0x08
包头	包头	包头	数据长度	“m”	8

重载调度系统程序，不关闭导航地图，用于动态切换地图。在调用前需要设置好当前的地图和路径。
|0xcd|0xeb|0xd7|0x02|0x6d|0x0a| |:--:|:--:|:--:|:--:|:--:|:--:|:--:| |包头|包头|包头|数据长度|“m”|10|

开始建立地图，结束建立地图,保存和更新地图

0xcd	0xeb	0xd7	0x02	0x56	0xXX

包头	包头	包头	数据长度	“V”	0表示为开启，值为1表示关闭

自动充电相关指令

0xcd	0xeb	0xd7	0x02	0x6a	0xXX
包头	包头	包头	数据长度	“j”	0表示为开始充电，值为1表示停止充电，2表示保存充电桩位置

迎宾模式

0xcd	0xeb	0xd7	0x02	0x48	0xXX
包头	包头	包头	数据长度	"H"	0表示为开启迎宾模式，1未关闭迎宾模式

版本历史

版本	时间	更新说明
V1.0	2018-3-15	初稿，覆盖基本功能
V1.2	2018-3-22	修改开启导航系统指令
V1.3	2018-7-11	添加动态目标点功能
V1.4	2018-11-8	增加自动巡检功能
V1.5	2018-11-23	增加地图创建相关状态，增加创建地图控制指令
V1.6	2019-1-25	添加自动充电指令
V1.7	2019-10-11	添加调度系统导航控制指令
V1.8	2020-04-02	增加动态导航载入控制
V1.9	2020-08-07	增加无ID目标点任务状态
V2.0	2021-09-21	更新接口

- 伽利略导航系统Modbus通信协议
  - 触点寄存器说明
  - 线圈寄存器说明
  - 输入寄存器说明
  - 保持寄存器说明
  - 程序例子

伽利略导航系统Modbus通信协议

注意此协议需要后端版本 6.6.3及以上

默认端口为3551。注意数据大小端规则是 byteorder=Endian.Big, wordorder=Endian.Little

触点寄存器说明

触点寄存器是只读的bool变量

地址	长度	类型	说明
1	1	bool	当前导航状态，1时说明开启了导航，0说明未开启导航
2	1	bool	当前建图状态，1说明开启了建图，0说明未开启
3	1	bool	GC状态，1说明正在进行内存回收, 0说明状态正常
4	1	bool	GBA状态，1说明正在进行闭环优化，0说明状态正常
5	1	bool	局部充电状态，1说明正在执行局部充电，0说明未在执行局部充电动作
6	1	bool	充电任务状态, 1说明正在执行充电任务，0说明未执行充电任务
7	1	bool	循环状态，1说明正在执行循环任务，0说明未执行循环任务
8	1	bool	保留位
9	1	bool	保留位
10	1	bool	保留位
11	1	bool	保留位
12	1	bool	保留位
13	1	bool	保留位
14	1	bool	保留位
15	1	bool	保留位
16	1	bool	保留位
17	1	bool	虚拟io0
18	1	bool	虚拟io1
19	1	bool	虚拟io2

20	1	bool	虚拟io3
21	1	bool	虚拟io4
22	1	bool	虚拟io5
23	1	bool	虚拟io6
24	1	bool	虚拟io7
25	1	bool	IO模块输入0
26	1	bool	IO模块输入1
27	1	bool	IO模块输入2
28	1	bool	IO模块输入3
29	1	bool	IO模块输入4
30	1	bool	IO模块输入5
31	1	bool	IO模块输入6
32	1	bool	IO模块输入7

线圈寄存器说明

线圈寄存器是具有读写功能的bool变量

地址	长度	类型	说明
1	1	bool	导航开关，1开始导航，0关闭导航
2	1	bool	建图开关，1开启建图，0关闭建图
3	1	bool	关机开关，1关机
4	1	bool	循环开关，1开启循环，0关闭循环
5	1	bool	充电开关，1开始自动充电任务，0停止自动充电任务
6	1	bool	保留位
7	1	bool	保留位
8	1	bool	保留位
9	1	bool	保留位
10	1	bool	保留位
11	1	bool	保留位
12	1	bool	保留位
13	1	bool	保留位
14	1	bool	保留位
15	1	bool	虚拟io8

16	1	bool	虚拟io9
17	1	bool	虚拟io10
18	1	bool	虚拟io11
19	1	bool	虚拟io12
20	1	bool	虚拟io13
21	1	bool	虚拟io14
22	1	bool	虚拟io15
23	1	bool	IO模块输出0
24	1	bool	IO模块输出1
25	1	bool	IO模块输出2
26	1	bool	IO模块输出3
27	1	bool	IO模块输出4
28	1	bool	IO模块输出5
29	1	bool	IO模块输出6
30	1	bool	IO模块输出7

输入寄存器说明

输入寄存器是只读的16位变量

地址	长度	类型	说明
1	1	int16	视觉系统状态，-1标系视觉系统处于关闭状态，0表示没初始化uninit，1表示正在追踪tracking,2表示丢失lost,1和2都表示视觉系统已经初始化完成。
2	2	float32	电池电压
4	1	int16	当前目标点编号,默认值为-1表示无效值，当正在执行无ID的任务是值为-2，比如通过Http API 创建的导航任务。
5	1	int16	当前目标点状态，0表示已经到达或者取消free，1表示正在前往目标点过程中working,2表示当前目标点的移动任务被暂停paused,3表示目标点出现错误error,默认值为-1表示无效值。
6	2	float32	机器人距离当前目标点的距离，单位为米，-1表示无效值，该值的绝对值小于0.01时表示已经到达。
8	2	float32	导航系统计算给出的前进速度控制分量,单位为m/s
10	2	float32	导航系统计算给出的角速度控制分量,单位为rad/s
12	2	float32	当前机器人实际前进速度分量,单位为m/s
14	2	float32	当前机器人实际角速度分量,单位为rad/s
			当前机器人在map坐标系下的X坐标,此坐标可以直接用于设置动态插入点

			坐标
18	2	float32	当前机器人在map坐标系下的Y坐标
20	2	float32	当前机器人在map坐标系下的z轴转角(yaw)
22	1	int16	当前追踪点数
24	2	uint32	当前信息时间戳，单位为秒
26	2	uint32	当前信息时间戳，剩余小于1秒的部分，单位为纳秒

保持寄存器说明

保持寄存器是具有读写功能的16位变量

地址	长度	类型	说明
1	2	float32	机器人遥控速度前进方向线速度，单位m/s，用于遥控机器人
3	2	float32	机器人遥控速度旋转方向角速度，单位rad/s,用于遥控机器人
5	1	int16	导航目标index，设置此值后，机器人会导航至对应目标
6	1	int16	1暂停任务，0继续任务
7	1	int16	1取消任务
8	6	3个float32	三个float32分别为 x,y,theta，即目标点的坐标和角度。设置此值后机器人会导航至对应坐标
14	1	int16	循环等待时间，单位为秒

程序例子

```python
#!/usr/bin/env python3

"""
Test Modbus
"""

from pymodbus.client import ModbusTcpClient
from pymodbus.version import version
from pymodbus.constants import Endian
from pymodbus.payload import BinaryPayloadDecoder
import time

ADDR_POWER = 2  # 电池电压, float32

if __name__ == '__main__':
    print(f"pymodbus version: {version}")
    address = ("127.0.0.1", 3551)
    client = ModbusTcpClient(
        host=address[0],
        port=address[1],
```

```
        port=address[1],
    )
    client.connect()
    while True:
        time.sleep(1)
        try:
            # pymodbus默认的地址会有一个偏移
            result = client.read_input_registers(ADDR_POWER - 1, 2, unit=1)
            print(f"result: {result.registers}")
            # 转换为浮点数
            decoder = BinaryPayloadDecoder.fromRegisters(
                result.registers, byteorder=Endian.Big, wordorder=Endian.Little)
            voltage = decoder.decode_32bit_float()
            print(f"voltage: {voltage}")
        except Exception as e:
            print(e)
```

- UDP 协议说明
  - 1. 连接
  - 2. 发送数据
  - 3. 接收数据
  - 客户端例子

UDP 协议说明

对于有些使用场景以上的协议都不太适合。比如大数据量传输，音频视频传输，高实时性数据传输，或者网络环境不稳定的场景下的数据传输。这时候就需要使用UDP协议了。

UDP协议类似于使用UDP模拟了一个websocket协议。首先我们定义了几个操作码，用于区分不同的操作。

操作码	操作
0x00	OPCODE_CONTINUATION，连续数据，数据尚未传输完成
0x01	OPCODE_TEXT，文本数据
0x02	OPCODE_BINARY，二进制数据
0x08	OPCODE_CLOSE_CONN，关闭连接
0x09	OPCODE_PING，ping
0x0A	OPCODE_PONG，pong

1. 连接

第一帧数据格式如下

0	1-x	结束位
0x01，标记此帧为连接数据	udp://[host]:[port]/[path]?token=[token], 字符串数据，用于表示自己的连接信息，其中host为机器人ip，如果使用galileo_proxy则host表示机器人的以ID开头的注册地址。port为UDP端口号，默认和机器人websocket端口号一致，3547。path是你要订阅或发送的ros话题地址。token为机器人token，向后可以扩展添加其他参数，比如galileo_proxy_token。	0x00

如果连接成功，机器人会返回一帧数据，其格式为UTF-8编码的json字符串，如下

```
{
    "status": true,
    "client_id": []
}
```

其中status标志连接是否成功，client_id为机器人分配的客户端id，用于标识客户端。client_id为一个长度为4的数组，每个元素为一个字节，如[0, 0, 0, 1]。

2. 发送数据

成功建立连接后就可以继续发送数据帧了，数据帧格式如下

0	1-4	5	6	7-...
0x02, 标记此帧为数据帧	client id	数据类型比如OPCODE_BINARY表示发送的是二进制数据	package_index，为uint32，表示数据包的序号，从0开始，每发送一帧数据加1	具体数据

同时需要注意的是，如果数据量较大，需要分包发送，每个包的数据长度不能超过1024字节，即每个包的数据长度不能超过1024-8=1016字节。一般采用512个字节作为最大长度。

注意建立连接后机器人会不断向客户端发送ping数据，客户端需要定时发送pong数据，以保持连接。同时客户端也可以发送ping数据，机器人会返回pong数据。一般为10hz左右。

3. 接收数据

接收到的数据帧和发送的数据帧格式一致。如果是文本数据，可以直接转换为字符串，如果是二进制数据，可以直接转换为字节数组。

客户端例子

```
using Newtonsoft.Json;
using SuperSocket.ClientEngine;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using WebSocket4Net;

namespace ChiTuClient.libs
{

    internal class UdpFastClient
    {

        byte OPCODE_CONTINUATION = 0x0;
        byte OPCODE_TEXT = 0x1;
        byte OPCODE_BINARY = 0x2;
        byte OPCODE_CLOSE_CONN = 0x8;
        byte OPCODE_PING = 0x9;
```

```
        byte OPCODE_PONG = 0xA;
        int MAX_DATA_LENGTH = 512;


        private Socket serverSocket;
        private EndPoint remotePoint;
        private EventHandler onConnectedCB = null;
        private EventHandler onDisconnectedCB = null;
        private EventHandler<MessageReceivedEventArgs> onMessageReceivedCB = null;
        private EventHandler<DataReceivedEventArgs> onDataReceivedCB = null;
        private EventHandler<ErrorEventArgs> onErrorCB = null;
        private string host;
        private int port;
        private string path;
        private string token;
        private byte[] clientID = null;
        private object socketLock = new object();
        long lastPongTime = 0;
        private UInt32 packageIndex = 0;
        public UdpFastClient(string host, int port, string path, string token)
        {
            // 创建UDP Client
            serverSocket = new Socket
                (AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            serverSocket.ReceiveTimeout = 5000;
            remotePoint = new IPEndPoint(IPAddress.Parse(host), port);
            this.path = path;
            this.host = host;
            this.port = port;
            this.token = token;
            serverSocket.Bind(new IPEndPoint(IPAddress.Parse("0.0.0.0"), 0));
        }

        public UdpFastClient(string hostAndPort, string path, string token)
        {
            this.host = hostAndPort.Split(':')[0];
            this.port = int.Parse(hostAndPort.Split(':')[1]);
            this.path = path;
            this.token = token;
            serverSocket = new Socket
                (AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            serverSocket.ReceiveTimeout = 5000;
            IPAddress hostIP;
            int serverPort = port;
            try
            {
                hostIP = IPAddress.Parse(host);
            }
            catch(Exception)
            {
                // WAN情况的连接，port应该是服务器的port
                hostIP = Dns.GetHostAddresses(host)[0];
                serverPort = 10427;
            }
            remotePoint = new IPEndPoint(hostIP, serverPort);
            serverSocket.Bind(new IPEndPoint(IPAddress.Parse("0.0.0.0"), 0));
        }

        public void SendPing()
        {
```

```
            lock (socketLock)
            {
                if (clientID == null)
                {
                    return;
                }
                List<byte> data = new List<byte>();
                data.Add(0x02);
                data.AddRange(clientID);
                data.Add(OPCODE_PING);
                try {
                    serverSocket.SendTo(data.ToArray(), remotePoint);
                }catch (Exception e) { }

            }
        }

        public void SendPong()
        {
            lock (socketLock)
            {
                if (clientID == null)
                    return;
                List<byte> data = new List<byte>();
                data.Add(0x02);
                data.AddRange(clientID);
                data.Add(OPCODE_PONG);
                try {
                    serverSocket.SendTo(data.ToArray(), remotePoint);
                }catch(Exception e) { }
            }
        }

        public void Send(byte[] buf, int offset, int length)
        {
            lock (socketLock)
            {
                if (clientID == null)
                    return;
                int currentIndex = offset;
                List<byte> data = new List<byte>();
                while (currentIndex + MAX_DATA_LENGTH < offset + length)
                {
                    // 数据太大需要分割数据
                    data.Clear();
                    data.Add(0x02);
                    data.AddRange(clientID);
                    data.Add(OPCODE_BINARY);
                    data.AddRange(BitConverter.GetBytes(packageIndex));
                    packageIndex += 1;
                    data.AddRange(buf.Skip(currentIndex).Take(MAX_DATA_LENGTH));
                    try {
                        serverSocket.SendTo(data.ToArray(), remotePoint);
                    }catch(Exception e) { }
                    currentIndex = currentIndex + MAX_DATA_LENGTH;
                }
                data.Clear();
                data.Add(0x02);
                data.AddRange(clientID);
                data.Add(OPCODE_BINARY);
```

```
                data.AddRange(BitConverter.GetBytes(packageIndex));
                packageIndex += 1;
                data.AddRange(buf.Skip(currentIndex).Take(offset + length - currentIndex))
;
                try {
                    serverSocket.SendTo(data.ToArray(), remotePoint);
                }catch(Exception e) { }

            }
        }

        public void Send(string msg)
        {
            lock (socketLock)
            {
                if (clientID == null)
                    return;
                List<byte> data = new List<byte>();
                data.Add(0x02);
                data.AddRange(clientID);
                data.Add(OPCODE_TEXT);
                data.AddRange(Encoding.UTF8.GetBytes(msg));
                if (data.Count > MAX_DATA_LENGTH)
                    MapUtils.Log("data too big");
                try {
                    serverSocket.SendTo(data.ToArray(), remotePoint);
                }catch(Exception e) { }
            }
        }

        public void Start()
        {

            try
            {
                lock(socketLock)
                {
                    // 发送第一帧数据
                    List<byte> data = new List<byte>();
                    data.Add(0x01);
                    data.AddRange(Encoding.UTF8.GetBytes($"udp://{host}:{port}/{path}?toke
n={token}"));
                    data.Add((byte)'\0');
                    serverSocket.SendTo(data.ToArray(), remotePoint);
                    EndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any, 0);
                    byte[] receiveBytes = new byte[1024 * 2];
                    int length = serverSocket.ReceiveFrom(receiveBytes, ref RemoteIpEndPoi
nt);
                    string receivedMsg = Encoding.UTF8.GetString(receiveBytes, 6, length);
                    UDPClientResponse res = JsonConvert.DeserializeObject<UDPClientResponse
>(receivedMsg);
                    if (res.status)
                    {
                        clientID = res.client_id;
                    }
                    else
                    {
                        clientID = null;
                    }
                }
```

```
                if(clientID == null && onDisconnectedCB != null)
                    onDisconnectedCB(this, null);
            }
            catch (Exception ignore)
            {
                lock(socketLock)
                {
                    clientID = null;
                }
                if (onDisconnectedCB != null)
                    onDisconnectedCB(this, null);
                return;
            }
            if(onConnectedCB != null)
            {
                onConnectedCB(this, null);
            }
            // 开启读取线程
            Task.Run(() => {
                while (true)
                {
                    lock(socketLock)
                    {
                        if (clientID == null)
                            return;
                    }
                    try
                    {
                        EndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any, 0);
                        byte[] receiveBytes = new byte[1024 * 2];
                        int length = serverSocket.ReceiveFrom(receiveBytes, ref RemoteIpEn
dPoint);
                        // 检测前几个字节是否正确
                        if (receiveBytes[0] != 2)
                            continue;
                        bool idCheckFlag = true;
                        lock(socketLock)
                        {
                            for (int i = 0; i < 4; i++)
                            {
                                if (clientID == null || receiveBytes[i + 1] != clientID[i]
)
                                {
                                    idCheckFlag = false;
                                    break;
                                }
                            }
                        }
                        if (!idCheckFlag)
                            continue;
                        // 任意类型的消息都可以更新pong时间
                        lastPongTime = GetTimeStamp();
                        // 处理各种数据类型
                        if (receiveBytes[5] == OPCODE_TEXT && onMessageReceivedCB != null)
                        {
                            string message = Encoding.UTF8.GetString(
                                receiveBytes, 6, length);
                            MessageReceivedEventArgs e = new MessageReceivedEventArgs(mess
age);
                            onMessageReceivedCB(this, e);
```

```
                        }
                        if (receiveBytes[5] == OPCODE_BINARY && onDataReceivedCB != null)
                        {
                            DataReceivedEventArgs e = new DataReceivedEventArgs(
                                receiveBytes.Skip(6).Take(receiveBytes.Length - 6).ToArray
());
                            onDataReceivedCB(this, e);
                        }
                        if (receiveBytes[5] == OPCODE_PING)
                        {
                            SendPong();
                            continue;
                        }
                        if (receiveBytes[5] == OPCODE_PONG || receiveBytes[5] == OPCODE_CO
NTINUATION)
                        {
                            continue;
                        }
                        if (receiveBytes[5] == OPCODE_CLOSE_CONN)
                        {
                            lock(socketLock)
                            {
                                clientID = null;
                            }
                            if (onDisconnectedCB != null)
                                onDisconnectedCB(this, null);
                            serverSocket.Close();
                            continue;
                        }
                    }
                    catch (Exception ignore)
                    {

                    }
                }
            });
            Task.Run(() => {
                // 定时发送ping
                while (true)
                {
                    lock(socketLock)
                    {
                        if (clientID == null)
                            return;
                    }
                    SendPing();
                    Thread.Sleep(100);
                    if (lastPongTime == 0)
                        lastPongTime = GetTimeStamp();
                    if (GetTimeStamp() - lastPongTime > 10 * 1000)
                    {
                        // 对面服务器可能已经下线
                        Stop();
                    }
                }
            });
        }

        public void Stop()
        {
```

```
            lock (socketLock)
            {
                if (clientID == null)
                    return;
                try
                {
                    List<byte> data = new List<byte>();
                    data.Add(0x02);
                    data.AddRange(clientID);
                    data.Add(0x08);
                    serverSocket.SendTo(data.ToArray(), remotePoint);
                    // 服务端将不会返回信息
                    clientID = null;
                }
                catch (Exception ignore)
                {
                    return;
                }
            }
            if (onDisconnectedCB != null)
                onDisconnectedCB(this, null);
            return;
        }

        public void OnOpen(EventHandler cb)
        {
            onConnectedCB = cb;
        }

        public void OnClose(EventHandler cb)
        {
            onDisconnectedCB = cb;
        }

        public void OnMessageReceived(EventHandler<MessageReceivedEventArgs> cb)
        {
            onMessageReceivedCB = cb;
        }

        public void OnError(EventHandler<ErrorEventArgs> cb)
        {
            onErrorCB = cb;
        }

        public void OnDataReceived(EventHandler<DataReceivedEventArgs> cb)
        {
            onDataReceivedCB = cb;
        }

        public bool IsConnected()
        {
            lock(socketLock)
            {
                return clientID != null;
            }
        }

        public static long GetTimeStamp()
        {
            var timeSpan = (DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0));
```

```
            return (long)timeSpan.TotalMilliseconds;
        }

        internal class UDPClientResponse
        {
            public bool status = false;
            public byte[] client_id;
        }
    }
}

// 创建局域网连接
var client = new UdpFastClient("192.168.0.132:3547", "cmd_vel", token);
// 创建云端连接
var client = new UdpFastClient("AAAAAAAA.3547.robot1.bwbot.org:3547", "cmd_vel", token);
// 收到txt消息
client.OnMessageReceived(new EventHandler<MessageReceivedEventArgs>(GalileoMessageReceived
));
// 收到二进制消息
client.OnDataReceived(new EventHandler<DataReceivedEventArgs>(GalileoDataReceived));
client.OnError(new EventHandler<ErrorEventArgs>(GalileoWebsocketOnError));
client.OnClose(new EventHandler(GalileoWebsocketOnClose));
client.OnOpen(new EventHandler(GalileoWebsocketOnOpen));
client.Start();


// 发送速度指令
TwistMsg msg = new TwistMsg(); // 根据TwistMsg定义的格式，构造消息
msg.linear.x = linear;
client.Send(JsonConvert.SerializeObject(msg));
```

```
// 此代码是在react native环境下运行的，需要安装react-native-udp
import UdpSocket from 'react-native-udp/lib/types/UdpSocket';
import dgram from '../components/native/UdpDgram';
import 'text-encoding-polyfill';
const OPCODE_CONTINUATION = 0x0;
const OPCODE_TEXT = 0x1;
const OPCODE_BINARY = 0x2;
const OPCODE_CLOSE_CONN = 0x8;
const OPCODE_PING = 0x9;
const OPCODE_PONG = 0xA;
const MAX_DATA_LENGTH = 512;

function Uint32ToUint8Array(value: number) {
    return [value & 0xFF, (value >> 8) & 0xFF, (value >> 16) & 0xFF, (value >> 24) & 0xFF];

}

type UDPClientResponse = {
    status: boolean,
    client_id: Uint8Array
};


// 是否是有效的的ipv4地址
function isIPv4(ip: string) {
    const reg = /^((25[0-5]|2[0-4]\d|[01]?\d\d?)($|(?!\.$)\.)){4}$/;
    return reg.test(ip);
```

```
}


class UdpFastClient {
    socket: UdpSocket;
    host: string;
    port: number;
    path: string;
    token: string;
    clientID: Uint8Array | null;
    onConnectedCB: (() => void) | null;
    onDisconnectedCB: (() => void) | null;
    onMessageReceivedCB: ((data: string) => void) | null;
    onDataReceivedCB: ((data: Uint8Array) => void) | null;
    onErrorCB: (() => void) | null;
    lastPongTime: number;
    packageIndex: number;
    constructor(hostAndPort: string, path: string, token: string) {
        this.host = hostAndPort.split(':')[0];
        this.port = parseInt(hostAndPort.split(':')[1]);
        // host 是否是有效ip地址
        if (!isIPv4(this.host)) {
            this.port = 10427;
        }
        this.socket = dgram.createSocket({ type: 'udp4', reusePort: true });
        this.path = path
        this.token = token;
        this.clientID = null;
        this.onConnectedCB = null;
        this.onDisconnectedCB = null;
        this.onMessageReceivedCB = null;
        this.onDataReceivedCB = null;
        this.onErrorCB = null;
        this.lastPongTime = 0;
        this.packageIndex = 0;
    }



    sendPing() {
        if (this.clientID == null) {
            return;
        }
        const data = [0x02, ...this.clientID, OPCODE_PING];
        this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e) =>
 {
            if (typeof e != 'undefined') {
                console.log("send ping error");
                console.log(e);
            }
        });
    }

    sendPong() {
        if (this.clientID == null) {
            return;
        }
        const data = [0x02, ...this.clientID, OPCODE_PONG];
        this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e) =>
 {
```

```
            if (typeof e != 'undefined') {
                console.log("send pong error");
                console.log(e);
            }
        });
    }

    sendBuf(buf: Uint8Array, offset: number, length: number) {
        if (this.clientID == null) {
            return;
        }
        let currentIndex = offset;
        let data = []
        while (currentIndex + MAX_DATA_LENGTH < offset + length) {
            data = [0x02, ...this.clientID, OPCODE_BINARY, ...Uint32ToUint8Array(this.pack
ageIndex)];
            this.packageIndex += 1;
            data = [...data, ...buf.slice(currentIndex, currentIndex + MAX_DATA_LENGTH)];
            this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e
) => {
                if (typeof e != 'undefined') {
                    console.log("send buf error 1");
                    console.log(e);
                }
            });
            currentIndex += MAX_DATA_LENGTH;
        }
        data = [0x02, ...this.clientID, OPCODE_BINARY, ...Uint32ToUint8Array(this.packageI
ndex)];
        this.packageIndex += 1;
        data = [...data, ...buf.slice(currentIndex, offset + length)];
        this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e) =>
 {
            if (typeof e != 'undefined') {
                console.log("send buf error 2");
                console.log(e);
            }
        });
    }

    sendMsg(msg: string) {
        if (this.clientID == null) {
            return;
        }
        const data = [0x02, ...this.clientID, OPCODE_TEXT, ...(new TextEncoder().encode(msg
))];
        if (data.length > MAX_DATA_LENGTH) {
            console.log("data too large");
        }
        this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e) =>
 {
            if (typeof e != 'undefined') {
                console.log("send msg error");
                console.log(e);
            }
        });
    }

    async start() {
        await this.waitBind(0);
```

```
        let data = [0x01, ...(new TextEncoder().encode(`udp://${this.host}:${this.port}/${
this.path}?token=${this.token}`)),
            '\0'.charCodeAt(0)];
        this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e) =>
 {
            if (typeof e != 'undefined') {
                console.log("send init error");
                console.log(e);
            }
        });
        try {
            let receiveBytes = await this.receive();
            // bytes to string
            let receivedMsg = new TextDecoder().decode(receiveBytes.slice(6));
            let res: UDPClientResponse = JSON.parse(receivedMsg);
            if (res.status) {
                this.clientID = res.client_id;
            } else {
                this.clientID = null;
            }
        } catch (e) {
            this.clientID = null;
        }
        if (this.clientID == null && this.onDisconnectedCB != null) {
            this.onDisconnectedCB();
            return;
        }
        if (this.onConnectedCB != null) {
            this.onConnectedCB();
        }
        this.socket.on('message', (receiveBytes: Uint8Array,
            rinfo: { address: string, port: number }) => {
            if (this.clientID == null) {
                // 关闭socket
                this.socket.close();
                return;
            }
            if (receiveBytes.length < 2) {
                return;
            }
            if (receiveBytes[0] != 0x02) {
                return;
            }
            let idCheckFlag = true;
            // 检测clientID
            for (let i = 0; i < 4; i++) {
                if (receiveBytes[i + 1] != this.clientID[i]) {
                    idCheckFlag = false;
                    break;
                }
            }
            if (!idCheckFlag) {
                return;
            }
            this.lastPongTime = new Date().valueOf();
            if (receiveBytes[5] == OPCODE_TEXT && this.onMessageReceivedCB != null) {
                const msg = new TextDecoder().decode(receiveBytes.slice(6));
                this.onMessageReceivedCB(msg);
            }
            if (receiveBytes[5] == OPCODE_BINARY && this.onDataReceivedCB != null) {
```

```
                this.onDataReceivedCB(receiveBytes.slice(6));
            }
            if (receiveBytes[5] == OPCODE_PING) {
                this.sendPong();
            }
            if (receiveBytes[5] == OPCODE_PONG || receiveBytes[5] == OPCODE_CONTINUATION) {

                return;
            }
            if (receiveBytes[5] == OPCODE_CLOSE_CONN) {
                this.clientID = null;
                if (this.onDisconnectedCB != null) {
                    this.onDisconnectedCB();
                }
                this.socket.close();
            }

        });

        // 定时发送ping
        const pingTimer = setInterval(() => {
            if (this.clientID == null) {
                clearInterval(pingTimer);
                return;
            }
            this.sendPing();
            if (this.lastPongTime == 0)
                this.lastPongTime = new Date().valueOf();
            if (new Date().valueOf() - this.lastPongTime > 10000) {
                this.stop();
            }
        }, 100);

    }

    async receive() {
        return new Promise<Uint8Array>((resolve, reject) => {
            let resoleved = false;
            const messageCB = (data: any, rinfo: { address: string, port: number }) => {
                this.socket.off('message', messageCB);
                resolve(data);
                resoleved = true;

            }
            this.socket.on('message', messageCB);
            setTimeout(() => {
                if (!resoleved) {
                    this.socket.off('message', messageCB);
                    reject();

                }
            }, 5000);
        });
    }

    async waitBind(port: number) {
        return new Promise((resolve, reject) => {
            // 等待socket绑定
            let binded = false;
            const bindCB = () => {
```

```
                this.socket.off('listening', bindCB);
                binded = true;
                resolve(true);
            }
            this.socket.on('listening', bindCB);
            this.socket.bind(port)
            setTimeout(() => {
                if (!binded) {
                    this.socket.off('listening', bindCB);
                    reject();
                }
            }, 5000);
        })
    }

    stop() {
        if (this.clientID == null) {
            return;
        }
        const data = [0x02, ...this.clientID, OPCODE_CLOSE_CONN];
        this.socket.send(new Uint8Array(data), 0, data.length, this.port, this.host, (e) =>
{
            if (typeof e != 'undefined') {
                console.log("send close error");
                console.log(e);
            }
        });
        this.clientID = null;
        if (this.onDisconnectedCB != null) {
            this.onDisconnectedCB();
        }
        this.socket.close();
    }

    onOpen(cb: () => void) {
        this.onConnectedCB = cb;
    }

    onClose(cb: () => void) {
        this.onDisconnectedCB = cb;
    }

    onMessageReceived(cb: (msg: string) => void) {
        this.onMessageReceivedCB = cb;
    }

    onDataReceived(cb: (data: Uint8Array) => void) {
        this.onDataReceivedCB = cb;
    }

    onError(cb: () => void) {
        this.onErrorCB = cb;
    }

    isConnected() {
        return this.clientID != null;
    }

}
```

```
export default UdpFastClient;

// 创建局域网连接
let udpclient = new UdpFastClient("192.168.0.132:3547", "cmd_vel", token);
// 创建公网连接
let udpclient = new UdpFastClient("aaaaaaa.3547.robot1.bwbot.org:3547", "cmd_vel", token);
udpclient.onOpen(() => {
    options.onConnect && options.onConnect();
});

udpclient.onMessageReceived((data) => {
    options.onMessage && options.onMessage(data);
});

udpclient.onDataReceived((data) => {
    options.onData && options.onData(data);
});

udpclient.onError(() => {
    options.onError && options.onError();
});

udpclient.onClose(() => {
    options.onClose && options.onClose();
});

udpclient.start();
```

- 伽利略导航系统SDK使用说明
  - 快速开始
    - 获取当前局域网内所有机器人
    - 连接机器人
    - 连接机器人，异步方式
    - 通过物联网跨局域网连接机器人
    - 获取机器人当前电压
    - 控制机器人前进
    - 控制机器人移动到特定坐标
  - SDK 说明
  - 注意事项

伽利略导航系统SDK使用说明

`目前SDK基于ros通信，在网络不稳定情况下稳定性较差。同时SDK功能相对较少，推荐使用HTTP API进行开发。SDK有两个版本，机器人在5.0.0以下版本用v1，以上采用v2`

为了降低用户开发和使用伽利略导航系统的难度，我们提供了伽利略导航系统SDK。目前对以下语言提供了支持

C++ pipeline failed

Python pipeline passed

C# pipeline passed

Java & Android pipeline passed

所有语言的SDK都是基于 `C++ SDK` 进行开发。下面对 `C++ SDK` 进行说明。其他语言可以进行参考，SDK的使用方法都是一致的。对于不同语言SDK的详细信息可以参考每个SDK的README文件。

快速开始

获取当前局域网内所有机器人

```
#include "GalileoSDK.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化SDK,注意只能实例化一个SDK对象
    while (true)
    {
        auto servers = sdk.GetServersOnline(); // 获取当前局域网内所有机器人
        if(servers.size() == 0){
            std::cout << "No server found" << std::endl;
        }
        for (auto it = servers.begin(); it < servers.end(); it++)
        {
```

```
            std::cout << it->getID() << std::endl; // 输出机器人ID
        }
        Sleep(1000);
    }
}
```

连接机器人

下面这种连接方式是阻塞的，sdk会等待10000ms连接局域网内任意一个机器人。如果局域网内有多个机器人，程序会返回MULTI_SERVER_FOUND错误。当局域网内有且仅有唯一的一个机器人时，sdk自动连接。 如果10000ms没有成功连接机器人，Connect方法会返回超时TIMEOUT错误。

```
#include "GalileoSDK.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化SDK
    auto res = sdk.Connect("", true, 10000, NULL, NULL); // 连接局域网内任意一个机器人
    if(res == GalileoSDK::GALILEO_RETURN_CODE::OK)
        std::cout << "Connect Success" << std::endl;
    std::cout << "Connect Failed" << std::endl;
}
```

连接机器人，异步方式

下面的连接方式是异步的，可以通过设置回调函数处理机器人连接状态。这个方法是非阻塞的，Connect方法会立即返回。当连接成功或超时时OnConnect回调会被调用。具体的状态信息可以通过回调函数的第一个参数获取

```
#include "GalileoSDK.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化SDK
    sdk.Connect("", true, 10000,
        // OnConnect回调函数
        [](GalileoSDK::GALILEO_RETURN_CODE res, std::string id) -> void {
            std::cout << "OnConnect Callback: result " << res << std::endl;
            std::cout << "OnConnect Callback: connected to " << id << std::endl;
        },
        // OnDisconnect回调函数
        [](GalileoSDK::GALILEO_RETURN_CODE res, std::string id) -> void {
            std::cout << "OnDisconnect Callback: result " << res << std::endl;
            std::cout << "OnDisconnect Callback: server " << id << std::endl;
    });
    while(true){
        Sleep(10000);
    }
}
```

通过物联网跨局域网连接机器人

只有v1版本支持此功能

```
#include "GalileoSDK.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化sdk
    // 通过物联网连接机器人，可以跨局域网连接。
    // 参数为目标机器人id，超时时间timeout，目标机器人密码，回调函数
    if (sdk.Connect("71329A5B0F2D68364BB7B44F3F125531E4C7F5BC3BCE2694DFE39B505FF9C730A614F
F2790C1", 10000, "xiaoqiang", NULL, NULL) != GalileoSDK::GALILEO_RETURN_CODE::OK)
    {
        std::cout << "Connect to server failed" << std::endl;
    }
    while (true)
    {
        if (sdk.PublishTest() == GalileoSDK::GALILEO_RETURN_CODE::OK) {
            std::cout << "pub succeed" << std::endl;
        }
        else {
            std::cout << "pub failed" << std::endl;
        }
        Sleep(1000);
    }
}
```

获取机器人当前电压

在连接机器人后，机器人的状态信息可以通过 `GetCurrentStatus` 方法获取。具体有哪些状态信息可以参照 `galileo_msg/GalileoStatus.h` 文件定义。

```
#include "GalileoSDK.h"
#include "galileo_msg/GalileoStatus.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化SDK
    sdk.Connect("", true, 10000, NULL, NULL); // 连接局域网内任意一个机器人
    galileo_msg::GalileoStatus status; // 创建伽利略状态对象
    while (true)
    {
        if (sdk.GetCurrentStatus(&status) == GalileoSDK::OK) // 获取当前系统状态
        {
            std::cout << status.power << std::endl; // 输出系统电压
        }
        else
        {
            std::cout << "Get status failed" << std::endl;
        }
        Sleep(1000);
    }
}
```

控制机器人前进

```
#include "GalileoSDK.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化SDK
```

```
    sdk.Connect("", true, 10000, NULL, NULL); // 连接局域网内任意一个机器人
    sdk.SetSpeed(0.1, 0); // 设置机器人速度，以0.1m/s速度前进
}
```

控制机器人移动到特定坐标

对于V1版本

假设机器人已经建立好地图且绘制好相关路径，我们可以通过MoveTo方法控制机器人移动到特定坐标位置。下面代码中的posx和posy即为目标点坐标。你可以替换成自己的实际目标点坐标。

对于V2版本可以直接设置目标点角度

```
#include "GalileoSDK.h"
#include "galileo_msg/GalileoStatus.h"

int main(){
    GalileoSDK::GalileoSDK sdk; // 实例化SDK
    sdk.Connect("", true, 10000, NULL, NULL); // 连接局域网内任意一个机器人
    sdk.StartNav();
    galileo_msg::GalileoStatus status;
    sdk.GetCurrentStatus(&status);
    // 等待正常追踪,正常追踪时visualStatus和navStatus都为1
    while (status.visualStatus != 1 || status.navStatus != 1)
    {
        std::cout << "Wait for navigation initialization" << std::endl;
        sdk.GetCurrentStatus(&status);
        Sleep(1000);
    }
    // 以下为V1例子
    uint8_t goalNum;
    sdk.MoveTo( posx, posy, &goalNum); // 移动到特定目标点v1
    // 以下为V2例子
    sdk.MoveTo( posx, posy, theta); // 移动到特定目标点v2
    // 等待移动开始
    sdk.GetCurrentStatus(&status);
    while (status.targetStatus != 1)
    {
        std::cout << "Wait for goal start" << std::endl;
        sdk.GetCurrentStatus(&status);
        Sleep(1000);
    }
    std::cout << "Goal started" << std::endl;
    // 等待移动完成
    while (status.targetStatus != 0)
    {
        std::cout << "Wait for goal complete" << std::endl;
        sdk.GetCurrentStatus(&status);
        Sleep(1000);
    }
}
```

SDK 说明

```
GALILEO_RETURN_CODE
Connect(std::string targetID, bool auto_connect, int timeout,
        void (*OnConnect)(GALILEO_RETURN_CODE, std::string),
        void (*OnDisconnect)(GALILEO_RETURN_CODE, std::string));
```

连接机器人

输入

`std::string targetID` 目标机器人ID

`bool auto_connect` 是否开启自动连接功能。当targetID为空字符串，且auto_connect为true时。机器人会自动连接局域网内的机器人。当局域网内有多台机器人时，此方法会返回MULTI_SERVER_FOUND错误。

`int timeout` 超时时间，单位毫秒。当在此时间内没有成功连接机器人则此方法返回超时错误。

`void (*OnConnect)(GALILEO_RETURN_CODE, std::string)` 连接回调函数。当此值为NULL时，Connect会以同步阻塞方式执行。Connect会等待连接成功直至超时。当此值不是NULL时，Connect会以非阻塞方式执行。当机器人成功连接或连接超时时，SDK会调用此回调函数。

`void (*OnDisconnect)(GALILEO_RETURN_CODE, std::string)` 连接断开回调函数。当此值非NULL,机器人连接断开时SDK会调用此回调函数。

返回

GALILEO_RETURN_CODE

```
std::vector<ServerInfo> GetServersOnline()
```

获取当前局域网内所有机器人信息

输入 无

输出

局域网内所有机器人列表

```
ServerInfo *GetCurrentServer();
```

获取当前连接的机器人信息

输入 无

输出 当前连接的机器人信息

在SDK未连接机器人时，此方法返回NULL。在SDK成功连接机器人后此方法返回机器人信息

```
GALILEO_RETURN_CODE SendCMD(uint8_t[] cmd, int length);
```

发送伽利略指令，具体信息参考伽利略串口说明文档

输入

`uint8_t[] cmd` 伽利略导航指令

`int length` 指令长度

输出

`GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StartNav();
```

开启导航服务。此方法只会发送开始导航指令，并不会等待导航开启完成。只能在机器人未处于导航状态且未处于建图状态下才可以执行此方法。

输入 空 输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StopNav();
```

关闭导航服务。此方法只会发送关闭导航指令，并不会等待导航关闭完成。只能在机器人处于导航状态下才可以执行此方法。

输入 空 输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SetGoal(int goalIndex);
```

设置导航目标点。此方法只会发送设置导航点指令，并不会等待目标点完成。只能在机器人处于导航状态下执行此方法。

输入

`int goalIndex` 目标点标号

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE PauseGoal();
```

暂停当前导航任务

输入 无

输出 GALILEO_RETURN_CODE

```
GALILEO_RETURN_CODE ResumeGoal();
```

继续当前导航任务

输出 无

输出 GALILEO_RETURN_CODE

```
GALILEO_RETURN_CODE CancelGoal();
```

取消当前导航任务

输入 无

输出 GALILEO_RETURN_CODE

```
GALILEO_RETURN_CODE InsertGoal(float x, float y);
```

只有v1版本支持此功能

插入导航点

输入

`float x` 插入点x坐标，此坐标为机器人map坐标系下坐标 `float y` 插入点y坐标，此坐标为机器人map坐标系下坐标

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE ResetGoal();
```

只有v1版本支持此功能

重置目标点。所有通过InsertGoal插入的目标点都会重置。只保留原始目标点。

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SetSpeed(float vLinear, float vAngle);
```

设置机器人速度

输入

`float vLinear` 直线运动速度，单位m/s `float vAngle` 转动速度， 单位 rad/s

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE Shutdown();
```

关闭机器人主机

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SetAngle(uint8_t sign, uint8_t angle);
```

只有v1版本支持此功能

设置机器人角度

输入

`uint8_t sign` 转向0表示正向旋转，1表示反向旋转 `uint8_t angle` 转动角度

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StartLoop();
```

开始循环导航。开启后机器人会按照导航点的顺序依次执行。

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StopLoop();
```

停止导航循环

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SetLoopWaitTime(uint8_t time);
```

设置循环等待时间。机器人在循环过程中到达目标点的等待时间

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StartMapping();
```

开始建图服务。机器人只能在不处于导航状态或建图状态下才能执行此方法。此方法只是发送开启建图指令，并不会等待建图服务开始运行。

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StopMapping();
```

停止建图服务。机器人只能在建图状态下才能执行此方法。此方法只是发送停止建图指令，并不会等待建图服务停止。

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SaveMap();
```

只有v1版本支持此功能

保存当前地图。当前地图会被保存在机器人的 `slamdb` 文件夹，但是在Windows客户端并不会显示。想要在Windows客户端显示需要把当前地图复制到 `saved-slamdb` 文件夹

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE UpdateMap();
```

只有v1版本支持此功能

更新地图

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StartChargeLocal();
```

开始局部充电功能。局部充电采用惯导定位，利用充电桩传感器对准充电桩。局部充电只能在机器人在充电桩附近使用。此方法只是发送开启局部充电指令，并不会等待充电动作完成

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StopChargeLocal();
```

停止局部充电功能

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SaveChargeBasePosition();
```

保存充电桩位置

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StartCharge(float x, float y); // V1
GALILEO_RETURN_CODE StartCharge(); // V2
```

开始充电。此方法采用融合导航定位算法，可以在保证机器人处于导航状态下，在地图任意位置执行此指令。此指令首先会控制机器人移动到充电桩附近,然后再启动局部充电指令。此指令是一个阻塞指令，程序会等待机器人运动到充电桩后再退出。V1版本需要自己传入充电桩坐标位置，V2版本不需要传入充电桩位置，程序可以自动获取。

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE StopCharge();
```

取消充电指令

输入 无

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE MoveTo(float x, float y, uint8_t *goalNum); // V1
GALILEO_RETURN_CODE MoveTo(float x, float y, float theta); // V2
```

控制机器人移动到指定坐标

输入

`float x` 目标点x坐标 `float y` 目标点y坐标 `goalNum` 新插入的目标点id，此参数相当于一个返回值。在成功插入点后此值会被设置为新插入点的ID `theta` 目标点角度

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE GetGoalNum(uint8_t *goalNum);
```

`只有v1版本支持此功能`

获取当前目标点总数

输入

`uint8_t *goalNum` 相当于返回值，成功调用后此值会被设置为当前目标点总数

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE GetCurrentStatus(galileo_msg::GalileoStatus *);
```

获取当前机器人伽利略状态

输入

`galileo_msg::GalileoStatus * status` 相当于返回值，成功调用后此值会被设置为当前伽利略系统状态

输出 `GALILEO_RETURN_CODE`

```
void SetCurrentStatusCallback(void (*callback)(
        GALILEO_RETURN_CODE, galileo_msg::GalileoStatus));
```

设置状态更新回调函数，当伽利略系统状态更新时会执行设置的回调函数并将最新的系统状态传递给此回调函数。

```
void SetGoalReachedCallback(
        void (*callback)(int goalID, galileo_msg::GalileoStatus));
```

设置到达目标点函数。当机器人到达任意目标点时会执行此处设置的回调函数。并且将目标点ID和当前系统状态传递给此回调函数

```
GALILEO_RETURN_CODE WaitForGoal(int goalID);
```

等待到达目标点。此方法会阻塞直至到达目标点或目标点被取消。

输入

`int goalID` 目标点ID

输出 `GALILEO_RETURN_CODE`

```
bool CheckServerOnline(std::string targetid);
```

检查目标机器人是否在线

输入

`std::string targetid` 目标机器人ID

输出

`bool` 目标机器人是否在线

```
GALILEO_RETURN_CODE SendAudio(char audio[]);
```

向机器人发布语音

输入

`char audio[]` 语音uft8字符串

输出 `GALILEO_RETURN_CODE`

```
void Dispose();
```

释放SDK占用的资源

```
GALILEO_RETURN_CODE EnableGreeting(bool flag);
```

控制迎宾模式。在迎宾模式下当有人在机器人前走过时机器人会播放欢迎光临语音。

输入

`bool flag` 是否开启迎宾模式

输出 `GALILEO_RETURN_CODE`

```
GALILEO_RETURN_CODE SendGalileoBridgeRequest(std::string method, std::string url, std::str
ing body, HttpBridgeResponse& response, int timeout)
```

只有v1版本支持此功能

调用Galileo Http API。Galileo Http API说明文档

输入

method http请求方式，如get，post等 url http请求url body http请求body response http请求响应，其数据结构为 std::string uuid，请求的id。 uint16_t status_code，响应的http状态码。std::string body， http响应body。 timeout 发送请求的超时值。

输出 `GALILEO_RETURN_CODE`

例子

```
GalileoSDK::HttpBridgeResponse res;
status = sdk.SendGalileoBridgeRequest("get", "/api/v1/system/status", "", res, 10 * 1000);
if (status != GalileoSDK::GALILEO_RETURN_CODE::OK)
{
    std::cout << "send http get req to server failed" << std::endl;
    std::cout << "status: " << status << std::endl;
    return;
}
```

注意事项

1. 在机器人刚启动时可能SDK会连接失败。机器人需要在启动后等待初始化完成才能成功连接（可能要十秒左右）。连接失败时可以重试连接。