SOCKET 通讯 SDK_插件使用文档

SOCKET 通讯 SDK_插件使用文档

本手册中包含的信息如有变更，恕不另行通知，且不应视为捷勃特的承诺。捷勃特对本手册中可能出现的错误概不负责。

除本手册中有明确陈述之外，本手册中的任何内容不应解释为捷勃特对个人损失、财产损坏或具体适用性等做出的任何担保或保证。

捷勃特对因使用本手册及其中所述产品而引起的意外或间接伤害概不负责。

可从捷勃特处获取此手册的额外复印件。

本出版物的原始语言为中文。

本出版物全部为国际单位制，GB 代表中国国家标准。

AgilebotRoboticsCo.,Ltd

中国上海

1. 文档概述

1.1 文档目的

本插件旨在使用 socket 通讯采取标准协议，实现只要能通过 socket 接受/发送指定格式字符就能控制/读取机器人的功能，来满足更方便的二次开发需求，降低客户配置和使用难度

1.2 适用产品与版本

机器人品牌与型号：**GBT-协作机器人系列**

机器人版本要求：**Copper7.6.F.0** 及以上，路由器版本 1.1.6 及以上

插件版本号：20251208

1.3 相关文档

机器人相关：https://www.sh-agilebot.com/

2. 用前需知

2.1 注意事项与更新记录

更新日期：2025/12/8

更新内容：初版发布

已知问题：

①插件只能自动获取 10.27.1.254 这一个 ip，故如果需要连接多台机器人请修改插件脚本或联系技术支持人员

②插件端口号不固定，有可能出现占用的情况，需要脚本指定或者重新分配，请联系技术支持人员

③插件偶发无法关闭的问题，表现为手动改变插件运行状态为未激活，插件底层仍在运行，此时接收命令仍有效。

④插件加载超时的问题

⑤当前插件是严格的串行处理，不支持并发指令处理。

2.2 安全相关

通用安全规范

操作前必须接受培训。

在自动运行前务必进行手动测试。

工作区域内确保**安全**。

3. 系统要求与准备工作

3.1 硬件要求

协作机器人控制器可正常开机使用。

3.2 软件要求

机器人操作系统版本正常可以使用。

机器人支持插件环境和插件安装。

3.3 安装与配置

插件安装:

https://dev.sh-agilebot.com/docs/extension/zh/02-development/04-package.html

4. 功能使用指南

需知:

GBT 机器人插件为支持用户进行二次开发而添加机器人控制器端口:

端口号	名称	功能
6101	控制接口	接收指定格式的 json 字符串

用户可通过 socket 通讯向机器人端口发送指定格式的 json 字符串来实现相关功能,如下所示。

发送:

{"id":id,"method":"方法名称,"params":{"参数"}}

```
{
    "id": 数字（整数或浮点数）,
    "method": "字符串",
    "params": {
```

```
        // 具体参数
    }
}
```

接受：

正常：

```
{
    "id":1,
    "method":"方法名称",
    "status":{
        // 具体状态
    }
 }
```

出错：

```
{
    "id":1,
    "method":"方法名称",
    "status":"false",
    "error":"出错信息"
 }
```

注：

- *"error"可能的结果:*

1.空

2.unknown method（未知/未定义方法）

3.missing required fields: id or method（发送的格式错误）

- 非正常返回则错误
- 出于方便阅读，本文档 json 作换行缩减处理。
- JSON 标准要求字符串值必须是连续的，不能包含未转义的换行符。
- 客户端在发送前应确保 JSON 字符串是有效的
- 发送 json 字符串时的 id 和接收结果时的 id 对应，id 值不同并不影响功能，，多机器使用建议根据 ip 分配 id。

```
# 正确的做法 - 转义特殊字符
import json

command = {
    "id": 1,
    "method": "set_tcs",
    "params": {
        "coordinate_system": "JOINT\n"   # 换行符会被正确转义
    }
}

json_string = json.dumps(command)
# 发送 json_string
```

当前功能:

4.0 连接功能：

连接和断开机器人

通过 socket 连接状态判断

上位机作客户端连接机器人参考 python 脚本

```
import socket
```

```
import json

# 机器人配置
robots = [
    {"ip": "192.168.110.2", "port": 6101, "name": "Robot_1"},
    {"ip": "192.168.110.3", "port": 6101, "name": "Robot_2"}
]

def check_robot_connection(robot_ip, port, robot_name):
    """检查单个机器人的连接状态"""
    try:
        # 创建 socket 连接
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(3)  # 3 秒超时

        # 尝试连接
        sock.connect((robot_ip, port))

        # 接收服务器确认消息
        response = sock.recv(1024).decode('utf-8').strip()

        # 关闭连接
        sock.close()

        print(f"✅ {robot_name} ({robot_ip}:{port}) - 连接成功")
```

```
            return True

        except Exception as e:
            print(f"❌ {robot_name} ({robot_ip}:{port}) - 连接失败: {e}")
            return False

# 检查所有机器人的连接状态
print("正在检查机器人连接状态...")
for robot in robots:
    check_robot_connection(robot["ip"], robot["port"], robot["name"])
```

运行结果显示:

```
正在检查机器人连接状态...
✅ Robot_1 (192.168.110.2:6101) - 连接成功
❌ Robot_2 (192.168.110.3:6101) - 连接失败: timed out
```

4.1 读取功能：

2.2.1 获取机器人伺服状态

原始：

方法名	**get_servo_status**() -> tuple[ServoStatusEnum, StatusCodeEnum]
请求参数	无参数
返回值	ServoStatusEnum: 伺服控制器状态 StatusCodeEnum: 函数执行结果

SOCKET

//发送

{

"id":id,

"method":"get_servo_status"

}

//返回

//伺服已开启

{

"id":id,

"method":"get_servo_status",

"status":"**true**"

}

//伺服已关闭

{

"id":id,

"method":"get_servo_status",

"status":"**false**"

}

参数：

无

示例：

发送：

{

"id":"1",

```
    "method":"get_servo_status"
}
```

返回：

伺服已开启

```
{
    "id":"1",
    "method":"get_servo_status",
    "status":"true"
}
```

伺服已关闭

```
{
    "id":"1",
    "method":"get_servo_status",
    "status":"false"
}
```

2.2.2 获取机器人当前关节角度

原始：

方法名	**motion.get_current_pose**(pose_type : PoseType, uf_index : int = 0, tf_index : int = 0) -> tuple[MotionPose, StatusCodeEnum]
请求参数	pose_type :PoseType 位姿类型 uf_index : 当使用 PoseType.CART 时需传入用户坐标系 id, 默认 0 tf_index : 当使用 PoseType.CART 时需传入工具坐标系 id, 默认 0
返回值	MotionPose: 机器人位姿

	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method":"get_current_joint"
}
//返回
{
    "id":id,
    "method":"get_current_joint",
    "status":"J1,J2,J3,J4,J5,J6"//单位是度数，如需弧度自行转化
}
```

示例：

发送：

```
{
    "id":"1",
    "method":"get_current_joint"
}
```

返回：

```
{
    "id": "1",
    "method": "get_current_joint",
    "status":
```

"85.04792122512235,87.8551501447484,-91.6810844228238,87.67312188678159,-88.79555144187017,126.14060312621241"

}

2.2.3 获取机器人当前笛卡尔坐标

原始:

方法名	**motion.get_current_pose**(pose_type : PoseType, uf_index : int = 0, tf_index : int = 0) -> tuple[MotionPose, StatusCodeEnum]
请求参数	pose_type :PoseType 位姿类型 uf_index : 当使用 PoseType.CART 时需传入用户坐标系 id, 默认 0 tf_index : 当使用 PoseType.CART 时需传入工具坐标系 id, 默认 0
返回值	MotionPose: 机器人位姿 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method":"get_current_pose",
    "params": {
        "coordinate_num":coordinate_num,   //用户坐标编号
        "tool_num ": tool_num                  //工具坐标编号
     }
}
//返回
```

```
{
    "id":id,
    "method":"get_current_pose",
    "status":"X,Y,Z,RX,RY,RZ"//单位毫米和度
}
```

参数:

coordinate_num: 用户坐标

tool_num: 工具坐标

示例:

发送:

```
{"id":"1","method":"get_current_pose","params":{"coordinate_num":0,"tool_num ":0}}
```

返回:

```
{"id": "1", "method": "get_current_pose", "status": "-81.04858469662369,521.8703769410705,463.6358511136111,177.3312399031415,5.67480240107024,48.71022708536099"}
```

2.2.4 获取机器人全局速度

原始:

方法名	**motion.get_OVC**() -> tuple[float, StatusCodeEnum]
请求参数	无参数
返回值	float: 速度比率，结果在 0~1 之间 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{"id":"1","method":"get_OVC"}
```

//返回

{"id": "1", "method": "get_OVC", "status":"speed"}//速度范围（0-1]

参数：

无

示例：

发送：

```
{
    "id":"1",
    "method":"get_OVC"
}
```

返回：

```
{
    "id":"1",
    "method":"get_OVC",
    "status":"0.1"
}
```

2.2.5 获取机器人全局加速度

原始：

方法名	**motion.get_OAC**() -> tuple[float, StatusCodeEnum]
请求参数	无参数
返回值	float：速度比率，结果在 0~1.2 之间 StatusCodeEnum：函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "get_OAC",
}
//返回
{
    "id": id,
    "method": "get_OAC",
    "status": status//string,oac 值（0,1.2]
```

参数：无

示例：

发送：

```
{
    "id": 1,
    "method": "get_OAC",
}
```

返回：

```
{
    "id": 1,
    "method": "get_OAC",
    "status": "0.5"
}
```

2.2.6 获取当前示教坐标系

原始：

方法名	**motion.get_TCS**() -> tuple[TCSType, StatusCodeEnum]
请求参数	无参数
返回值	TCSType：示教坐标系编号 StatusCodeEnum：函数执行结果

SOCKET

//发送

{

"id": id,

"method": "get_TCS"

}

//返回

{

"id": id,

"method": "get_TCS"

"status": status//string,CART,BASE,JOINT,WORLD,TOOL,USER,RTCP_USER,RTCP_TOOL

}

参数：无

示例：

发送：

{

"id": 2,

```
    "method": "get_TCS"
}
```

返回：

```
{
    "id": 2,
    "method": "get_TCS",
    "status": "JOINT"
}
```

2.2.7 获取机器人 IO 状态

原始：

方法名	**signals.read**(signal_type : SignalType, index : int) -> tuple[float, StatusCodeEnum]
请求参数	signal_type : SignalType 要读取的 IO 类型 index : 要读取的 IO 的序号，从 1 开始
返回值	float: IO 值，1 代表高电平，0 代表低电平 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method":"signals_read",
```

```
    "params": {
        "signals_type":signals_type,
        "signals_num": signals_num
        }
}
//返回
{
    "id":id,
    "method":"signals_read",
    "params": {
    "signals_type":signals_type,
    "signals_num": signals_num
                }
    "status": status//io 值

}
```

参数:

signals_type:

信号类型 (DO, DI, UO, UI, RO, RI, GO, GI, TAI, TDI, TDO, AI, AO)， string 类型

signals_num:

信号编号，int 类型

示例：

发送：

```
{
    "id":1,
```

```
    "method":"signals_read",
    "params":{
        "signals_type":"DO",
        "signals_num":7
    }
}
```

返回：

```
{
    "id":1,
    "method":"signals_read",
    "params":{
        "signals_type":"DO",
        "signals_num":7
        },
     "status":0
}
```

2.2.8 获取机器人寄存器值

2.2.6.1 数值寄存器 R

①写入 R 寄存器

原始:

方法名	**register.write_R**(index : int, value : float) -> StatusCodeEnum
请求参数	index : int R 寄存器的编号 value : float 需要更新的寄存器的值

返回值	StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"write_r","params": {"index_r":index_r,"num_r":num_r}}

//返回

//成功:

{"id":id,"method":"write_r","params": {"index_r":index_r,"num_r":num_r},"status":"**true**"}

//失败:

{"id":id,"method":"write_r","params": {"index_r":index_r,"num_r":num_r},"status":"**false**"}

参数:

示例:

发送:

{"id":1,"method":"write_r","params": {"index_r":1,"num_r":1}}

返回:

{"id":1,"method":"write_r","params": {"index_r":1,"num_r":1},"status":"**true**"}

②读取 R 寄存器

原始:

方法名	**register.read_R**(index : int) -> tuple[float, StatusCodeEnum]

请求参数	index : int 希望获取的寄存器编号
返回值	float: 寄存器信值 StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":"1","method":"read_r","params": {"index_r":index_r}}

//返回

//成功:

{"id":"1","method":"read_r","params":
{"index_r":index_r},"status":{"index_r":index_r,"value":value}}

//失败:

{"id":"1","method":"read_r","params": {"index_r":index_r},"status":"**false**"}

参数:

示例:

发送:

{"id":"1","method":"read_r","params": {"index_r":1}}

返回:

{"id":"1","method":"read_r","params": {"index_r":1},"status":{"index_r":1,"value":1}}

③删除 R 寄存器

原始:

方法名	**register.delete_R**(index : int) -> StatusCodeEnum
请求参数	index : int R 寄存器的编号
返回值	StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"delete_r","params": {"index_r":index_r}}

//返回

//成功:

{"id":id,"method":"delete_r","params": {"index_r":index_r},"status":"**true**"}

//失败:

{"id":id,"method":"delete_r","params": {"index_r":index_r},"status":"**false**"}

参数:

示例:

发送:

{"id":1,"method":"delete_r","params": {"index_r":1}}

返回:

{"id":1,"method":"delete_r","params": {"index_r":1},"status":"**false**"}

2.2.6.2 字符串寄存器 SR

①添加和设置 SR 寄存器

原始:

方法名	**register.write_SR**(index : int, value : str) -> StatusCodeEnum
请求参数	index : int SR 寄存器的编号 value : str 需要更新的寄存器值
返回值	StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"write_sr","params": {"index_sr":index_sr,"num_sr":num_sr}}

//返回

成功:

{"id":id,"method":"write_sr","params":
{"index_sr":index_sr,"num_sr":num_sr},"status":"**true**"}

失败:

{"id":id,"method":"write_sr","params":
{"index_sr":index_sr,"num_sr":num_sr},"status":"**false**"}

参数:

示例:

发送:

{"id":1,"method":"write_sr","params": {"index_sr":1,"num_sr":abc}}

返回:

{"id":1,"method":"write_sr","params": {"index_sr":1,"num_sr":abc},"status":"**true**"}

②读取 SR 寄存器

原始:

方法名	**register.read_SR**(index : int) -> tuple[str, StatusCodeEnum]
请求参数	index : int 希望获取的寄存器编号
返回值	str: 寄存器的值 StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"read_sr","params":{"index_sr":index_sr}}

//返回

//成功

{"id":id,"method":"read_sr","params":{"index_sr":index_sr},"status":{"index_sr":index_sr,"value":value}}

//失败

{"id":id,"method":"read_sr","params":{"index_sr":index_sr},"status":"false"}

参数:

示例:

发送:

{"id":1,"method":"read_sr","params":{"index_sr":1}}

返回：

{"id":1,"method":"read_sr","params":{"index_sr":1},"status":abc}}

③删除 SR 寄存器

原始:

方法名	**register.delete_SR**(index : int) -> StatusCodeEnum
请求参数	index : int SR 寄存器的编号
返回值	StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"delete_sr","params":{"index_sr":index_sr}}

//返回

//成功:

{"id":id,"method":"delete_sr","params":{"index_sr":index_sr},"status":"true"}

//失败:

{"id":id,"method":"delete_sr","params":{"index_sr":index_sr},"status":"false"}

参数：

示例：

发送：

{"id":"1","method":"delete_sr","params":{"index_sr":1}}

返回：

成功：{"id":"1","method":"delete_sr","params":{"index_sr":1},"status":"true"}

失败：{"id":"1","method":"delete_sr","params":{"index_sr":1},"status":"false"}

2.2.6.3 位置寄存器 PR

①添加和设置 PR

原始：

方法名	**register.write_PR**(value : PoseRegister) -> StatusCodeEnum
请求参数	index : int 希望获取的寄存器编号
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "write_pr",
    "params": {
        "index_pr": index_pr,//number
        "pose_type": pose_type,//"cart"or"joint"
        "pose_data": {
            "x": x,//number
            "y": y,//number
```

```
                "z": z,//number
                "a": a,//number
                "b": b,//number
                "c": c //number
            },
            "posture_arm_back_front": 1//预留，待研发定义完善
        }
}
//返回
{
    "id": id,
    "method": "write_pr",
    "params": {
        "index_pr": index_pr,
        "pose_type": pose_type,//"cart"or"joint"
        "posture_arm_back_front": "1"//预留，待研发定义完善
    },
    "status": status    //true or false
}
```

参数：

示例：

①写入 pr 寄存器关节点位

发送：

```
{
    "id": 1,
```

```
    "method": "write_pr",
    "params": {
        "index_pr": 6,
        "pose_type": "joint",
        "pose_data": {
            "j1": 10,
            "j2": 20,
            "j3": 30,
            "j4": 10,
            "j5": 20,
            "j6": 30
        },
        "posture_arm_back_front": 1
    }
}
```

返回：

```
{
    "id": 1,
    "method": "write_pr",
    "params": {
        "index_pr": "5",
        "pose_type": "joint",
        "posture_arm_back_front": "1"
    },
```

```
    "status": "true"
}
```


②写入 pr 寄存器笛卡尔点位

发送：


```
{
    "id": 1,
    "method": "write_pr",
    "params": {
        "index_pr": 5,
        "pose_type": "cart",
        "pose_data": {
            "x": 100,
            "y": 200,
            "z": 300,
            "a": 100,
            "b": 200,
            "c": 300
        },
        "posture_arm_back_front": 1
    }
}
```


返回：


```
{
```

```
    "id": 1,
    "method": "write_pr",
    "params": {
        "index_pr": "5",
        "pose_type": "cart",
        "posture_arm_back_front": "1"
    },
    "status": "true"
}
```

②读取 PR

原始:

方法名	**register.read_PR**(index : int) -> tuple[PoseRegister, StatusCodeEnum]
请求参数	index : int 希望获取的寄存器编号
返回值	PoseRegister 寄存器信息 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "read_pr",
    "params": {
```

```
            "index_pr": index_pr//number
        }
}
//返回
{
        "id": id,
        "method": "read_pr",
        "params": {
            "index_pr": index_pr//number
        },
        "status": {
            "pose_type": pose_type,//"cart"or"joint"
            "posture_arm_back_front":posture_arm_back_front
            "x": x,
            "y": y,
            "z": z,
            "a": a,
            "b": b,
            "c": c
        }
}
```

参数：

示例：

发送：

```
{
```

```
    "id": 3,
    "method": "read_pr",
    "params": {
        "index_pr": 5
    }
}
```

返回：

```
{
    "id": 3,
    "method": "read_pr",
    "params": {
        "index_pr": "5"
    },
    "status": {
        "index_pr": 5,
        "pose_type": "cart",
        "posture_arm_back_front": 1,
        "x": 100.0,
        "y": 200.0,
        "z": 300.0,
        "a": 100.0,
        "b": 200.0,
        "c": 300.0
    }
```

}

③删除 PR

原始:

方法名	**register.delete_PR**(index : int) -> StatusCodeEnum
请求参数	index : int PR 寄存器的编号
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "delete_pr",
    "params": {
        "index_pr": index_pr
    }
}
//返回
{
    "id": id,
    "method": "delete_pr",
    "params": {
        "index_pr": index_pr,
    },
```

```
    "status": status//ture or false
}
```

参数：

示例：

发送：

```
{
    "id": 4,
    "method": "delete_pr",
    "params": {
        "index_pr": 5
    }
}
```

返回：

```
{
    "id": 4,
    "method": "delete_pr",
    "params": {
        "index_pr": "5",
    },
    "status": "true"
}
```

2.2.6.4 运动寄存器 MR

①写入 MR

原始:

方法名	**register.read_MR**(index : int) -> tuple[int, StatusCodeEnum]
请求参数	index : int 希望获取的寄存器编号
返回值	int: 寄存器的值 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "write_mr",
    "params": {
        "index_mr": index_mr,//地址， int
        "num_mr": num_mr        //写入的值,int
    }
}
//返回
{
    "id":id,
```

```
    "method": "write_mr",
    "params": {
        "index_mr": index_mr,
        "num_mr": num_mr
    }
    "status":status//"true"or"false"
}
```

参数：

示例：

发送：

```
{
    "id": 1,
    "method": "write_mr",
    "params": {
        "index_mr": 5,
        "num_mr": 8
    }
}
```

返回：

```
{
    "id": 1,
    "method": "write_mr",
    "params": {
        "index_mr": 5,
        "num_mr": 8
```

```
            }
        "status":"true"
}
```

②读取 MR

```
{
  "id": 2,
  "method": "read_mr",
  "params": {
  "index_mr": 5
  }
}
```

③删除 MR

```
{
  "id": 3,
  "method": "delete_mr",
  "params": {
  "index_mr": 5
  }
}
```

2.2.6.5modbus 保持寄存器 MH

①写入 MH

原始:

方法名	**register.write_MH**(index : int, value : int) -> StatusCodeEnum
请求参数	index : int MH 寄存器的编号 value : int 需要更新的寄存器值
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "write_mh",
    "params": {
        "index_mh": index_mh,//int,i 地址
        "num_mh": num_mh//int,  写入的值
    }
}
//返回
{
    "id":id,
    "method": "read_mh",
    "params": {
        "index_mh": 1
        "num_mh": num_mh
    }
    "status":status//状态,"true" or "false"
```

```
}
```

参数：

示例：

发送：

```
{
    "id": 1,
    "method": "write_mh",
    "params": {
        "index_mh": 1,
        "num_mh": 16
    }
}
```

返回：

```
{
    "id": 1,
    "method": "write_mh",
    "params": {
        "index_mh": 1,
        "num_mh": 16
    }
"status":"true"
}
```

②读取 MH

原始:

方法名	**register.read_MH**(index : int) -> tuple[int, StatusCodeEnum]
请求参数	index : int 希望获取的寄存器编号
返回值	int: 寄存器值 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "read_mh",
    "params": {
        "index_mh": index_mh//int,地址
    }
}
//返回
{
    "id":id,
    "method": "read_mh",
    "params": {
        "index_mh": index_mh//int,地址
    }
    "status":status//值 int,错误为"false"
}
```

参数：无

示例：

发送：

```
{
    "id":1,
    "method": "read_mh",
    "params": {
        "index_mh": 1
    }
}
```

返回：

```
{
    "id":id,
    "method": "read_mh",
    "params": {
        "index_mh": 1
    }
    "status":16 //值是 16
}
```

2.2.6.6modbus 输入寄存器 MI

①写入 MI

原始:

方法名	**register.write_MI**(index : int, value : int) -> StatusCodeEnum
请求参数	index : int MI 寄存器的编号 value : int 需要更新的寄存器值
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "write_mi",
    "params": {
        "index_mi":index_mi,//地址
        "num_mi":num_mi//写入的值，int
    }
}
//返回
{
    "id":id,
    "method": "write_mi",
    "params": {
        "index_mi":index_mi,//地址
        "num_mi":num_mi//写入的值，int
    }
```

```
    "status":status//状态,"true" or "false"
}
```

参数：

示例：

发送：

```
{
    "id":id,
    "method": "write_mi",
    "params": {
        "index_mi":1,
        "num_mi":2
    }
}
```

返回：

```
{
    "id":id,
    "method": "write_mi",
    "params": {
        "index_mi":1,
        "num_mi":2
    }
    "status":"true"
}
```

②读取 MI

原始:

方法名	**register.read_MI**(index : int) -> tuple[int, StatusCodeEnum]
请求参数	index : int 希望获取的寄存器编号
返回值	寄存器值 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "read_mi",
    "params": {
        "index_mi": index_mi//int,地址
    }
}
//返回
{
    "id": id,
    "method": "read_mi",
    "params": {
        "index_mi":index_mi
    }
    "status":stutus//值 int,错误为"false"
}
```

参数：

示例：

发送：

```
{
    "id":4,
    "method": "read_mi",
    "params": {
        "index_mi": 1
    }
}
```

返回：

```
{
    "id":4,
    "method": "read_mi",
    "params": {
        "index_mi": 1
    }
    "status":16
}
```

寄存器类型	数据类型	支持操作	用途说明
R 寄存器	浮点数(float)	读/写/删除	通用数值寄存器
SR 寄存器	字符串(str)	读/写/删除	字符串寄存器
PR 寄存器	位姿数据	读/写/删除	位姿寄存器

MR 寄存器	整数(int)	读/写/删除	矩阵寄存器
MH 寄存器	整数(int)	读/写	特殊用途整数寄存器
MI 寄存器	整数(int)	读/写	特殊用途整数寄存器

2.2.9 获取当前使用的工具坐标系 TF

原始：

方法名	**motion.get_TF**() -> tuple[int, StatusCodeEnum]
请求参数	无
返回值	int: 工具坐标系编号 StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"get_TF"}

//返回

{"id":id, "method": "get_TF", "status":status}

参数：

status：string 类型，有数据如“1”表示获取成功的坐标系编号，为“false”则获取失败

示例：

发送：

{"id":1,"method":"get_TF"}

返回：

{"id": 1, "method": "get_TF", "status":"1"}

2.2.10 获取当前使用的用户坐标系 UF

原始：

方法名	**motion.get_UF**() -> tuple[int, StatusCodeEnum]
请求参数	无
返回值	int: 用户坐标系编号 StatusCodeEnum: 函数执行结果

SOCKET

//发送

{"id":id,"method":"get_UF"}

//返回

{"id":id, "method": "get_UF", "status":status}

参数：

status：string 类型，有数据如“1”表示获取成功的坐标系编号，为“false”则获取失败

示例：

发送：

{"id":1,"method":"get_UF"}

返回：

{"id": 1, "method": "get_UF", "status":"1"}

4.2 控制功能：

4.1.0 控制本体 led 灯开关

原始：

方法名	**switch_led_light**(mode : bool) -> StatusCodeEnum
请求参数	mode : bool，True 为打开，False 为关闭
返回值	StatusCodeEnum：函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method":"switch_led_light",
    "params":{
    "enable":enable
    }
}
//返回
{.
    "id":id,
    "method":"switch_led_light",
    "status":status
}
```

参数：enable：string 类型，"true"为打开，"false"为关闭

status：string 类型，"true"为成功，"false"为失败

示例：

发送：

开灯

```
{
    "id":1,
    "method":"switch_led_light",
    "params":{
        "enable":"true"
        }
}
```

关灯

```
{
    "id":1,
    "method":"switch_led_light",
    "params":{
        "enable":"false"
        }
}
```

返回：

成功

```
{
    "id":1,
    "method":"switch_led_light",
    "status":"true"
}
```

失败

```
{
    "id":1,
```

"method":"switch_led_light",

"status":"false"

}

4.1.1 关节运动到指定位置

原始：

方法名	**motion.move_joint**(pose : MotionPose, vel : float = 1, acc : float = 1) -> StatusCodeEnum
请求参数	pose : MotionPose 笛卡尔空间或关节坐标系上的一个点的坐标 vel : float 运动的速度，范围是 0~1，表示最大速度的倍数 acc : float 范围是 0~1.2，表示最大加速度的倍数
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "move_joint",
    "params": {
        "pose_type": "joint"或"cart",//选用的点位关节还是笛卡尔
        "pose_data": "j1,j2,j3,j4,j5,j6"或"X,Y,Z,A,B,C",  // 字符串，单位度数和毫米
        "vel": 0.5,  // 数字,范围（0,1]
        "acc": 0.5    // 数字,范围（0,1.2]
    }
}
//返回
```

{"id":id,"method":"move_joint","status":status}

参数：

pose_type：string 类型，“joint”或“cart”

pose_data：string 类型，"J1,J2,J3,J4,J5,J6"或“X,Y,Z,A,B,C”，单位度数和毫米

vel：number 类型，范围（0,1]，速度

acc：number 类型，范围（0,1.2],加速度，注意实际机器人速度需要×全局速度

status：string 类型，"true"为成功，"false"为失败

示例：

①使用关节点位，关节运动至指定位置

发送：

```
{
   "id": 1,
   "method": "move_joint",
   "params": {
      "pose_type": "joint",
      "pose_data": "90,90,-90,90,-90,0",
      "vel": 0.5,
      "acc": 0.2
   }
}
```

返回：

```
//成功:
{
      "id":1,
      "method":"move_joint",
```

```
    "status":"true"
}
//失败:
{
    "id":1,
    "method":"move_joint",
    "status":"false"
}
```

②使用笛卡尔点位，关节运动至指定位置

发送：

```
{
  "id": 1,
  "method": "move_joint",
  "params": {
    "pose_type": "cart",
    "pose_data": "66,607,233,-144,17,-150",
    "vel": 0.5,
    "acc": 0.2
  }
}
```

返回：

```
//成功:
{
    "id":1,
    "method":"move_joint",
```

"status":"**true**"

}

//失败:

{

"id":1,

"method":"move_joint",

"status":"**false**"

}

4.1.2 直线运动到指定位置

原始:

方法名	**motion.move_line**(pose : MotionPose, vel : float = 100, acc : float = 1) -> StatusCodeEnum
请求参数	pose : MotionPose 笛卡尔空间或关节坐标系上的一个点的坐标 vel : float 运动的速度，范围是 0~5000mm/s，表示机械臂末端移动速度 acc : float 范围是 0~1.2，表示最大加速度的倍数
返回值	StatusCodeEnum: 函数执行结果

SOCKET

//发送

{

"id": id,

"method": "move_line",

"params": {

"pose_type": "joint" 或 "cart",//选用的点位关节还是笛卡尔

"pose_data": "j1,j2,j3,j4,j5,j6" 或 "X,Y,Z,A,B,C",// 字符串，单位度数和毫米

"vel": 0.5, // 数字，范围（0,5000]

"acc": 0.5 // 数字，范围（0,1.2]

}

}

//返回

{

"id":id,

"method":"move_joint",

"status":status

}

参数：

pose_type：string 类型，“joint”或“cart”

pose_data：string 类型，"J1,J2,J3,J4,J5,J6"或“X,Y,Z,A,B,C”，单位度数和毫米

vel：number 类型，范围（0,5000]，速度

acc：number 类型，范围（0,1.2],加速度，注意实际机器人速度需要×全局速度

status：string 类型，"true"为成功，"false"为失败

示例：

①关节点位，直线运动至指定位置

发送：

```
{
  "id": 1,
  "method": "move_line",
  "params": {
    "pose_type": "joint",
    "pose_data": "90,90,-90,90,-90,0",
```

```
        "vel": "500",
        "acc": "0.2"
    }
}
```

返回：

```
//成功:
{
        "id":1,
        "method":"move_line",
        "status":"true"
}
//失败:
{
        "id":1,
        "method":"move_line",
        "status":"false"
}
```

②笛卡尔点位直线运动至指定位置

发送：

```
{
    "id": 1,
    "method": "move_line",
    "params": {
        "pose_type": "joint",
        "pose_data": "90,82,-113,121,-90,0",
```

```
        "vel": "500",
        "acc": "0.2"
    }
}
```

返回:

```
//成功:
{
        "id":1,
        "method":"move_line",
        "status":"true"
}
//失败:
{
        "id":1,
        "method":"move_line",
        "status":"false"
}
```

4.1.3 设置当前使用的工具坐标系编号 TF

原始:

方法名	**motion.set_TF**(value : int) -> StatusCodeEnum
请求参数	value : int，工具坐标系编号，序号在 0~10 之间
返回值	StatusCodeEnum: 函数执行结果

SOCKET

//发送:

```
{
    "id": id,
    "method": "set_TF",
    "params": {
        "tf_num": tf_num        //工具坐标系编号，number or string 都可，
    }
}
```

//返回

```
{
    "id": id,
    "method": "set_TF",
    "params": {
    "tf_num": tf_num
    },
    "status": status
}
```

参数：

tf_num : //工具坐标系编号，number or string 都可，需要在现有坐标系编号内

status：string 类型，"true"为成功，"false"为失败

示例：

发送：

```
{
    "id": 1,
    "method": "set_TF",
```

```
        "params": {
            "tf_num": "0"
        }
    }
```

返回：

```
//成功
{
    "id": 1,
    "method": "set_TF",
    "params": {
        "tf_num": "0"
        },
    "status": "true"
}
//失败
{
    "id": 1,
    "method": "set_TF",
    "params": {
        "tf_num": "0"
        },
    "status": "false"
}
```

4.1.4 设置当前使用的用户坐标系编号 UF

原始：

方法名	**motion.set_UF**(value : int) -> StatusCodeEnum
请求参数	value : int，用户坐标系编号
返回值	StatusCodeEnum：函数执行结果

SOCKET

```
//发送:
{
    "id": id,
    "method": "set_UF",
    "params": {
        "uf_num": uf_num        //用户坐标系编号，number or string 都可，
    }
}
//返回
  {
      "id": 1,
      "method": "set_UF",
      "params": {
          "uf_num": uf_num
      },
      "status": status
  }
```

参数：

uf_num ： 用户坐标系编号，number or string 都可，需要在现有的坐标系最大编

号范围内

status：string 类型，"true"为成功，"false"为失败

示例：

发送：

```
{
    "id": 1,
    "method": "set_UF",
    "params": {
        "uf_num": "0"
    }
}
```

返回：

```
//成功
{
    "id": 1,
    "method": "set_UF",
    "params": {
        "uf_num": "0"
    },
    "status": "true"
}
//失败
  {
    "id": 1,
    "method": "set_UF",
```

```
    "params": {
        "uf_num": "0"
    },
    "status": "false"
}
```

4.1.5 设置机器人全局速度 OVC

原始：

方法名	**motion.set_OVC**(value : float) -> StatusCodeEnum
请求参数	value : float，速度比率范围是 0~1
返回值	StatusCodeEnum：函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "set_OVC",
    "params": {
        "ovc_value":ovc_value
    }
}
//返回
{
    "id": id,
    "method": "set_OVC",
```

```
    "params": {
        "ovc_value": ovc_value
    },
    "status": false
}
```

参数：

ovc_value:number/string 类型，范围(0-1]

status：string 类型，"true"为成功，"false"为失败

示例：

发送：

```
{
    "id": 1,
    "method": "set_OVC",
    "params": {
        "ovc_value": 0.7
    }
}
```

返回：

```
//成功
{
    "id": 1,
    "method": "set_OVC",
    "params": {
        "ovc_value": "0.7"
    },
```

```
    "status": "true"
}
//错误：
{
    "id": 1,
    "method": "set_OVC",
    "params": {
        "ovc_value": "1.5"
    },
    "status": "false",
    "error": "ovc_value must be between 0.0 and 1.0"
}
```

4.1.6 设置机器人全局加速度 OAC

原始：

方法名	**motion.set_OAC**(value : float) -> StatusCodeEnum
请求参数	value : float，加速度比率，0~1.2 之间
返回值	StatusCodeEnum：函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "set_OAC",
    "params": {
```

```
            "oac_value": oac_value
        }
}
//返回
{
        "id": id,
        "method": "set_OAC",
        "params": {
            "oac_value": oac_value
        },
        "status": status
}
```

参数：

oac_value:number/string 类型，范围(0-1.2]

status：string 类型，"true"为成功，"false"为失败

示例：

发送：

```
 {
        "id": 1,
        "method": "set_OAC",
        "params": {
            "oac_value": 0.7
        }
 }
```

返回：

成功：

```
{
    "id": 1,
    "method": "set_OAC",
    "params": {
        "oac_value": "0.7"
    },
    "status": "true"
}
```

错误：

```
{
    "id": 1,
    "method": "set_OAC",
    "params": {
        "oac_value": "1.5"
    },
    "status": "false",
    "error": "oac_value must be between 0.0 and 1.0"
}
```

4.1.7 设置示教坐标系 TCS

原始：

方法名	**motion.set_TCS**(value : TCSType) -> StatusCodeEnum
请求参数	value : TCSType, , TCS 示教坐标系

返回值	StatusCodeEnum：函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "set_tcs",
    "params": {
        "coordinate_system": coordinate_systemt
    }
}
//返回
{
    "id": id,
    "method": "set_tcs",
    "params": {
        "coordinate_system": coordinate_system
    },
    "status": status//true or false
}
```

参数：

coordinate_system：string 类型;JOINT/BASE/WORLD/USER/TOOL/RTCP_UESR/RTCP_TOOL

status：string 类型，"true"为成功，"false"为失败

示例：

发送：

```
{
    "id": 1,
    "method": "set_tcs",
    "params": {
        "coordinate_system": "JOINT"
    }
}
```

返回：

```
{
    "id": 1,
    "method": "set_tcs",
    "params": {
        "coordinate_system": "JOINT"
    },
    "status": "true"
}
```

4.1.8 机器人示教运动

原始：

方法名	**jogging.move**(aj_num : int, move_mode : MoveMode = None, step_length : float = 0) -> StatusCodeEnum
请求参数	aj_num : int 填写数值 1-6 数值 1~6 对应关系根据当前示教的坐标系来决定 关节坐标系下 关节值号 [1 ~ 6] 笛卡尔坐标系下 x, y, z, rx, ry, rz

	move_mode : MoveMode 机械臂运动模式 增量运动 or 连续运动,Continuous,Stepping step_length : float 单次步进量，单位为 mm 或 °，不给值即不修改步进值
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method":"jogging_move",//"jogging_stop"为停止
    "params": {
        "direction":direction,//±1，±2，±3，±4，±5，±6，
        "move_mode":move_mode,//"stepping","continuous"
        "step": step//步长，"stepping"下有效，number 类型
    }
}
//返回
{
    "id": id,
    "method":"jogging_move",
    "params": {
        "direction":direction,
        "move_mode":move_mode,
        "step": step
    }
```

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

参数：

status：string 类型，"true"为成功，"false"为失败

示例：

jogging 步进

发送：

```
{
    "id":1,
    "method": "jogging_move",
    "params": {
        "direction": -1,
        "move_mode": "stepping",
        "step": 1.0
    }
}
```

返回：

```
{
    "id": 1,
    "method": "jogging_move",
    "params": {
        "direction": -1,
        "move_mode": "stepping",
        "step": 1.0
    }
```

```
    "status":"true"


}
```

jogging 持续

发送：

```
{
    "id": 1,
    "method": "jogging_move",
    "params": {
        "direction": 1,
        "move_mode": "continuous"
    }
}
```

返回：

```
{
    "id": 1,
    "method": "jogging_move",
    "params": {
        "direction": 1,
        "move_mode": "continuous"
    }
    "status":"true"
}
```

jogging 停止运动

发送：

```
{
   "id": 1,
   "method": "jogging_stop"
}
```

返回：

```
{
   "id": 1,
    "method": "jogging_stop"
    "status":"true"
}
```

注意：

jogging 坐标系需要选择示教坐标系 TCS

4.1.9 报警重置

原始：

方法名	**alarm_reset**() -> StatusCodeEnum
请求参数	无参数
返回值	StatusCodeEnum：函数执行结果

SOCKET

```
//发送
{
      "id":id,
      "method": "reset_alarm"
}
```

//返回

{

"id": id,

"method": "reset_alarm",

"status": status//"true"or "false"

}

参数：无

示例：

发送：

{

"id": 1,

"method":"reset_alarm"

}

返回：

成功:

{

"id":1,

"method":"reset_alarm",

"status":"true"

}

4.1.10 伺服上电

原始：

方法名	**servo_on**() -> StatusCodeEnum

请求参数	无参数
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id": id,
    "method": "servo_on"
}
//返回
{
    "id": id,
    "method": "servo_on"
    "status":status//"true"or"false"
}
```

参数：无

示例：

发送：

```
{
    "id": 1,
    "method": "servo_on"
}
```

返回

```
{
    "id": 1,
```

"method": "servo_on"

"status": "true"

}

4.1.11 伺服下电

原始：

方法名	**servo_off**() -> StatusCodeEnum
请求参数	无参数
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "servo_off"
}
//返回
{
    "id": id,
    "method": "servo_off"
    "status":status//"true"or"false"
}
```

参数：无

示例：

发送：

```
{
    "id": 1,
    "method": "servo_off"
}
```

返回：

```
{
    "id": 1,
    "method": "servo_off"
    "status":"true"
}
```

4.1.12 执行某个指定程序

原始：

方法名	**execution.start**(program_name : str) -> StatusCodeEnum
请求参数	program_name : str 需要执行的程序名称
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "start_program",
    "params": {
        "program_name": program_name//string,程序名称
    }
```

```
}
//返回
{
    "id":id,
    "method": "start_program",
    "params": {
        "program_name": program_name//string,程序名称
    }
    "status":status//"true"or"false"
}
```

参数：

program_name:string 类型,程序名称

示例：

发送：

```
{
    "id": 1,
    "method": "start_program",
    "params": {
        "program_name": "test"
    }
}
```

返回：

```
{
    "id": 1,
    "method": "start_program",
```

```
    "params": {
        "program_name": "test"
    }
    "status":"true"
}
```

4.1.13 停止正在执行的程序

原始：

方法名	**execution.stop**(program_name : str = '') -> StatusCodeEnum
请求参数	program_name : str 需要停止的程序名称，默认为空字符串，表示停止当前正在运行的程序或运动指令
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "stop_program",
    "params": {
        "program_name": program_name//string,程序名称
    }
}
//返回
{
```

```
    "id":id,
    "method": "stop_program",
    "params": {
        "program_name": program_name//string,程序名称
    }
    "status":status//"true"or"false"
}
```

参数：

program_name：string 类型,程序名称

示例：

发送：

```
{
    "id": 1,
    "method": "stop_program",
    "params": {
        "program_name": "test"
    }
}
```

返回：

```
{
    "id": 1,
    "method": "stop_program",
    "params": {
        "program_name": "test"
    }
```

"status":"true"

}

4.1.14 暂停程序运行

原始：

方法名	**execution.pause**(program_name : str = '') -> StatusCodeEnum
请求参数	program_name : str 需要暂停的程序名称，默认为空字符串，表示暂停当前正在运行的程序或运动指令
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "pause_program",
    "params": {
        "program_name": program_name//string,程序名
    }
}
//返回
{
    "id":id,
    "method": "pause_program",
    "params": {
```

```
        "program_name": program_name//string,程序名
    }
    "status":status//"true"or"false"
}
```

参数：

示例：

发送：

```
{
    "id": 1,
    "method": "pause_program",
    "params": {
        "program_name": "test"
    }
}
```

返回：

```
{
    "id": 1,
    "method": "pause_program",
    "params": {
        "program_name": "test"
    }
    "status":"true"
}
```

4.1.15 恢复程序运行

原始：

方法名	**execution.resume**(program_name : str = '') -> StatusCodeEnum
请求参数	program_name : str 需要继续运行的程序名称，默认为空字符串，表示继续运行当前处于暂停状态的程序或运动指令
返回值	StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "resume_program",
    "params": {
        "program_name":program_name//string,需要恢复的程序名称
    }
}
//返回
{
    "id":id,
    "method": "resume_program",
    "params": {
        "program_name": program_name
    },
    "status":status//"true"or"false"
}
```

参数：

program_name:string 类型,需要恢复的程序名称

示例：

发送：

```
{
    "id": 1,
    "method": "resume_program",
    "params": {
        "program_name": "test"
    }
}
```

返回：

```
{
    "id": 1,
    "method": "resume_program",
    "params": {
        "program_name": "test"
    },
    "status":"true"
}
```

4.1.16 返回所有正在运行的程序信息

原始：

方法名	**execution.all_running_programs**() -> tuple[list, StatusCodeEnum]

请求参数	无参数
返回值	list: 运行的程序详细信息列表 StatusCodeEnum: 函数执行结果

SOCKET

```
//发送
{
    "id":id,
    "method": "get_running_programs"
}
//返回
{
    "id":id,
    "method": "get_running_programs"
    "status":status//string,当前运行程序名，空值为无程序运行
}
```

参数：

无

示例：

发送：

```
{
    "id": 1,
    "method": "get_running_programs"
}
```

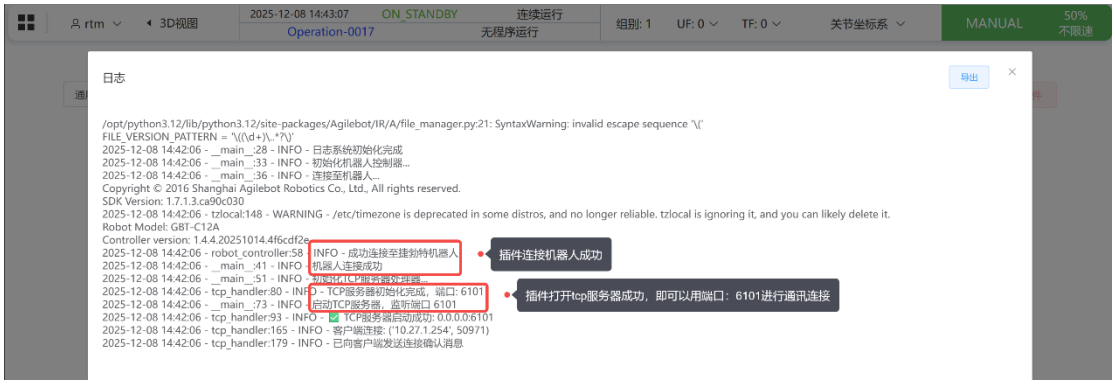
返回：

```
{
    "id": 1,
    "method": "get_running_programs"
    "status":"test"
}
```

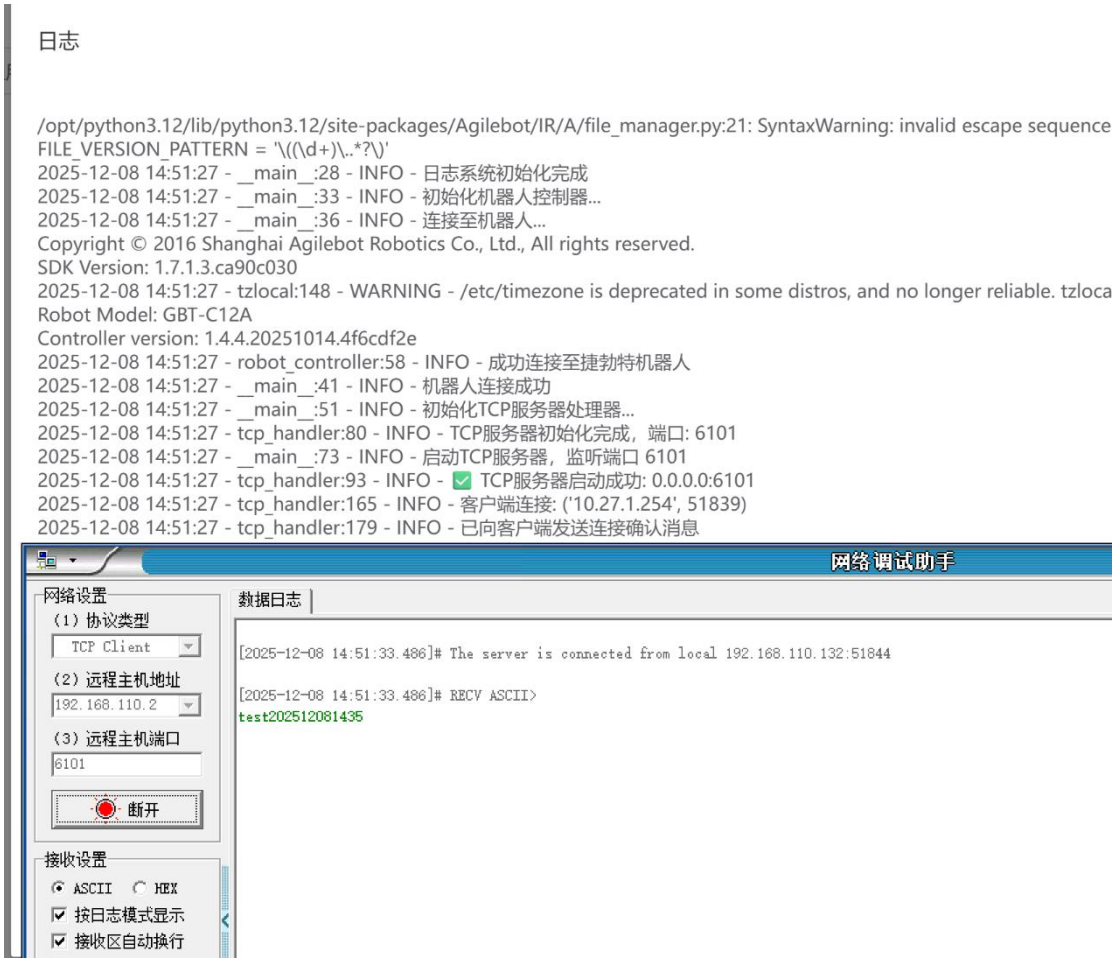
4.3 使用：

建议使用 socket 调试助手测试与机器人的通讯，由于编程语言和上位机品牌等多样性，理论上调试助手可以与机器人建立正常稳定的通讯则可视为机器人功能正常。

①确保插件安装成功，点击插件日志栏目可以查看插件启动状态

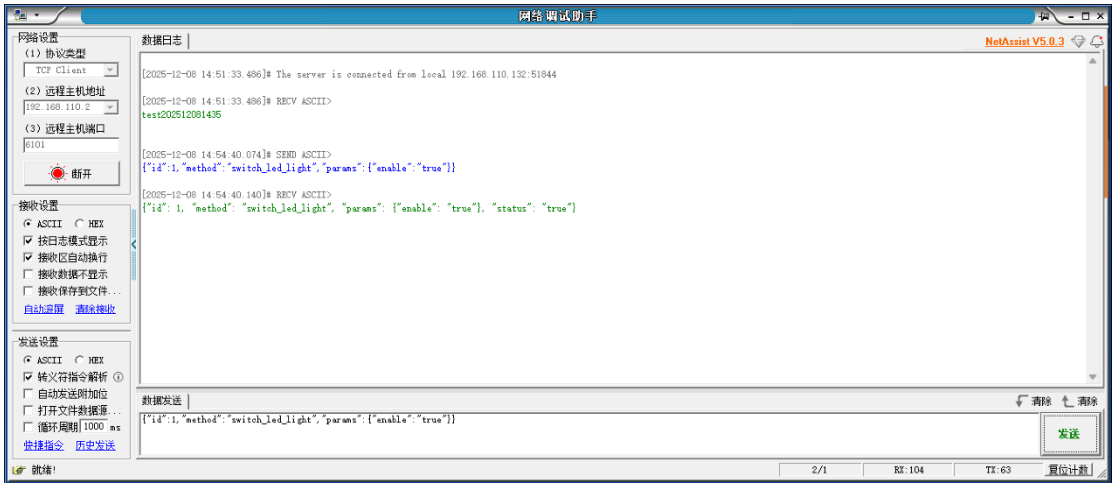


②助手连接后，机器人会主动发送一段字符“test202512081435”(具体字符根据版本有所区别)，用户也可根据此判断首次连接是否成功



③使用调试助手发送:

开本体 led 灯指令如下图



④其余功能可如上述测试

5. 维护与故障排除

5.1 常见问题与解决方案（FAQ）

问题：无法连接机器人

可能原因：线缆松动、ip 地址或网段错误、端口被占用。

解决方案：检查物理连接，确认配置参数。

5.2 错误代码说明

联系技术支持人员

6. 附录

6.1 免责声明

本插件及相关文档均按“现状”提供，不附带任何明示或暗示的保证，包括但不限于对商品适销性、特定用途适用性以及不侵犯第三方权利的暗示保证。开发商不保证本插件完全无缺陷或错误，也不保证其操作不会中断。

在任何情况下，开发商及其关联方均不对因使用或无法使用本插件而导致的任何直接、间接、附带、特殊、惩罚性或后果性损失承担责任，包括但不限于：

- 数据丢失或损坏。

- 生产中断或利润损失。
- 业务中断。
- 设备损坏或人身伤害。
- 任何第三方提出的索赔。

任何对本插件的未经授权的修改、反向工程、反编译或反汇编均被严格禁止。因用户或第三方对插件进行的任何未授权改动而导致的任何问题，开发商概不负责。

用户有责任确保在使用本产品时，完全遵守其所在国家或地区的所有适用的安全标准、法律、法规，包括但不限于机械指令、电气安全法规、职业健康与安全规定等。开发商提供的安全信息仅为一般性指导，不能替代用户根据其具体应用场景进行的全面风险评估