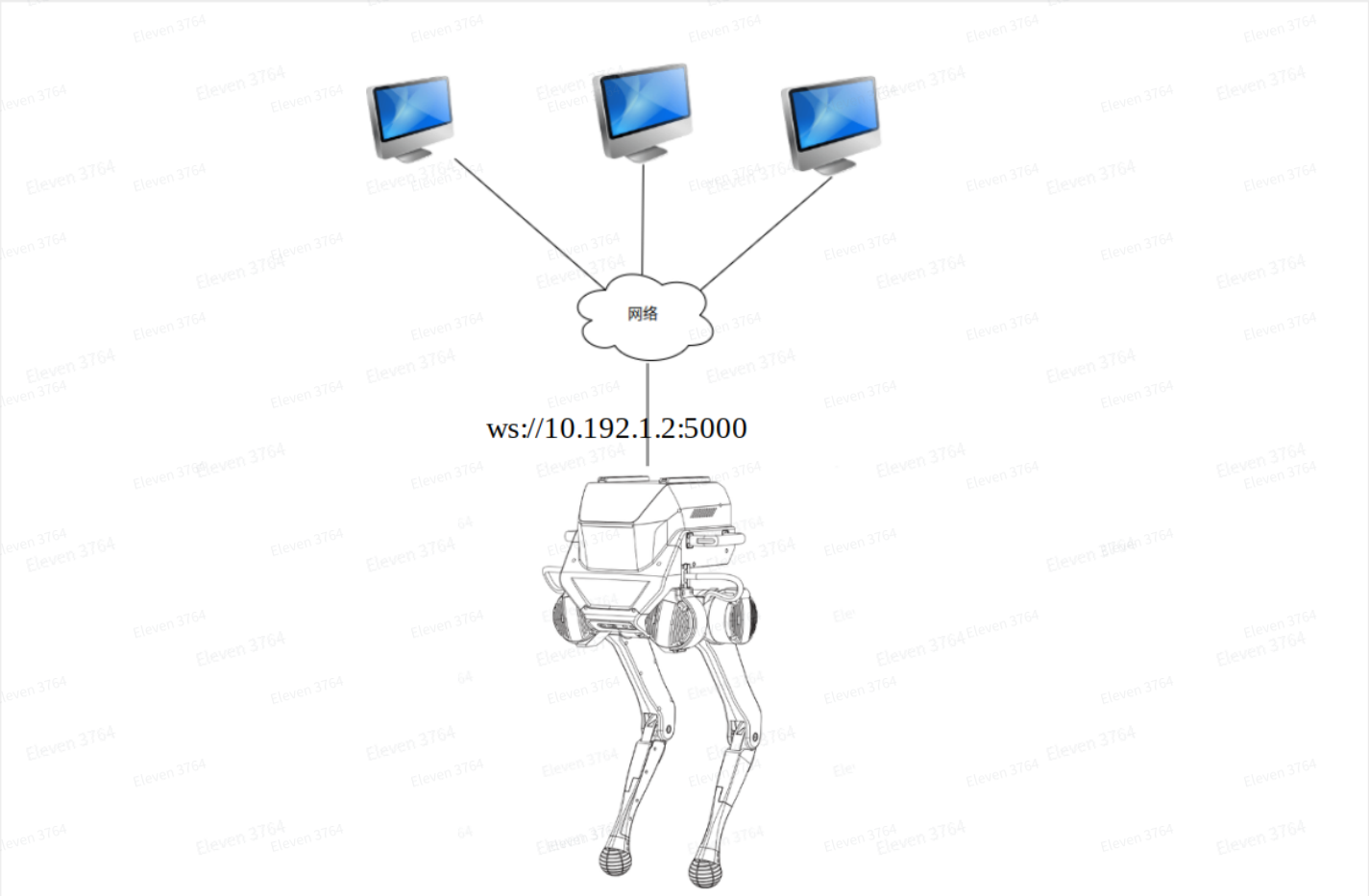
TRON1 WebSocket上层应用开发接口

1. 概述

在 `遥控模式` 下机器人通过WebSocket通信端口 `5000` 来接收用户端请求指令，例如让机器人站起、蹲下、行走等。WebSocket是一种实时通信协议，在机器人和用户端之间建立长连接，以便快速有效地传输控制信息和数据。如下图所示：



2. 通信协议格式

当机器人通过 WebSocket 接收客户端指令时，采用 JSON 数据协议进行信息传递。这种方式具有显著优势：WebSocket 是一种全双工通信协议，能够在客户端与服务器之间建立实时、低延迟的连接，特别适合频繁交互的应用场景。JSON 数据协议则以其简洁、可读性强的结构，确保数据传输直观明了，且具有跨平台、跨语言的兼容性。WebSocket 与 JSON 的结合不仅与编程语言无关，适用于各种设备和系统，还能提升开发的灵活性和维护的便利性。

- 请求数据格式包含以下字段：
  - `accid`：机器人唯一序列号，标识机器人的唯一身份；
  - `title`：指令名称，以“`request_`”为前缀；

- `timestamp`：指令发出时间戳，单位为毫秒；
- `guid`：指令的唯一标识符，用于区分不同的请求指令。如果是同步接口，则需要在“response_xxx”响应消息中通过guid字段将值带回给客户端。客户端接收到响应消息后，可以通过比较guid字段的值是否与请求指令中的值相同来判断指令是否执行完成；
- `data`：存放请求指令的数据内容。可以根据具体需求包含多个子字段，以存放请求指令所需的数据内容，例如执行动作的参数、发送消息的文本内容等等；
- 示例如下：

```
{
  "accid": "PF_TRON1A_042", # 机器人唯一序列号，标识机器人的唯一身份
  "title": "request_xxx",   # 指令名称，以“request_”为前缀
  "timestamp": 1672373633989, # 指令发出时间戳，单位为毫秒
  "guid": "746d937cd8094f6a98c9577aaf213d98", # 指令的唯一标识符，用于区分不同的请求指令
  "data": {}  # 存放请求指令的数据内容
}
```

- 响应数据格式包含以下字段：
  - `accid`：机器人唯一序列号，标识机器人的唯一身份；
  - `title`：指令名称，以“`response_`”为前缀；
  - `timestamp`：指令发出时间戳，单位为毫秒；
  - `guid`：与对应请求指令的`guid`值相同；
  - `data`：至少应该包含一个“result”子字段，用于存放请求指令的执行结果数据。如果有需要，还可以包含其他子字段，例如错误码、错误信息等用于描述操作结果的信息；
  - 示例如下：

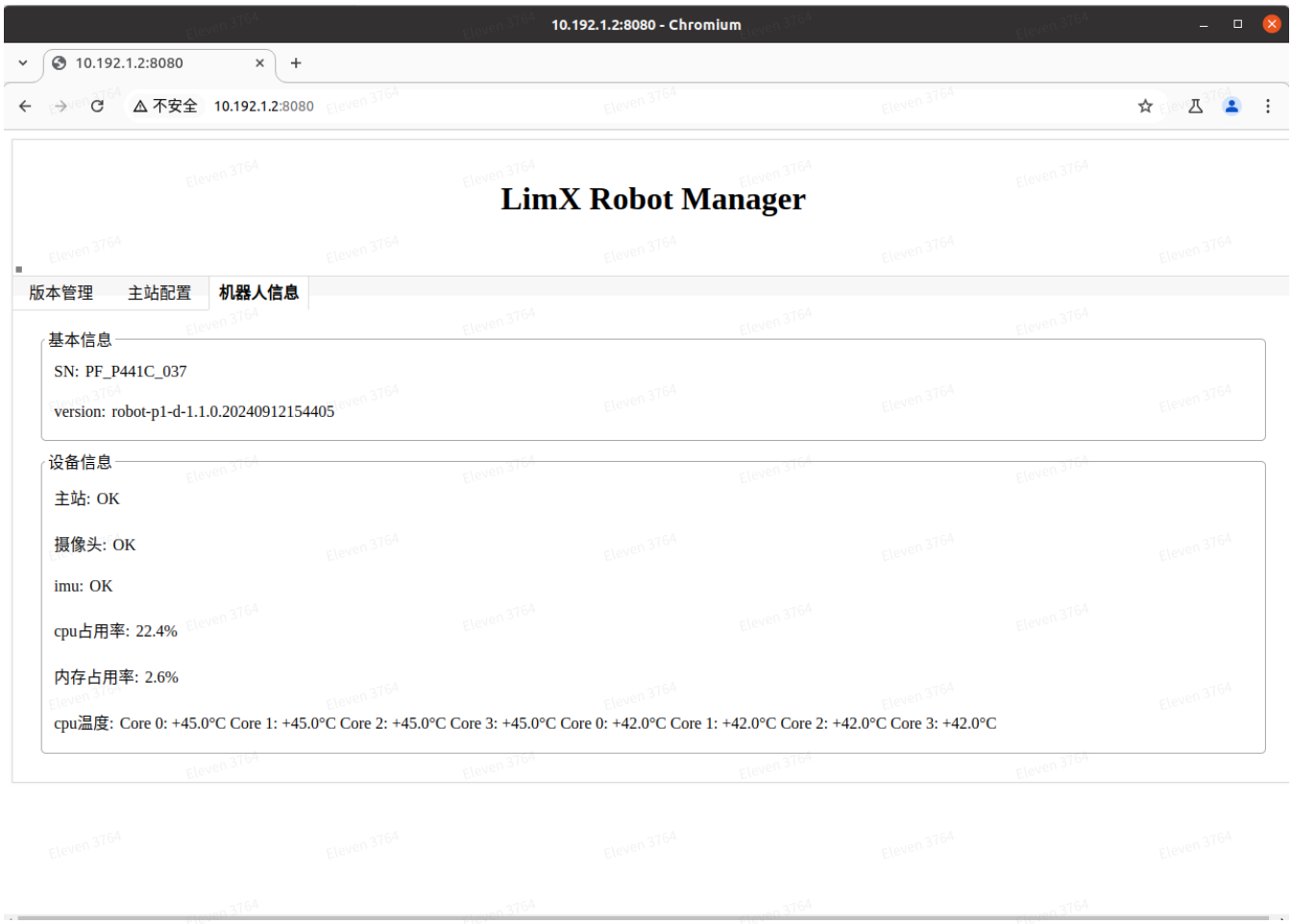
```
{
  "accid": "PF_TRON1A_042",   # 机器人唯一序列号，标识机器人的唯一身份
  "title": "response_xxx",  # 指令名称，以“response_”为前缀
  "timestamp": 1672373633989, # 指令发出时间戳，单位为毫秒
  "guid": "746d937cd8094f6a98c9577aaf213d98", # 与对应请求指令的guid值相同
  "data": { # 存放响应指令的具体数据内容
    "result": "success"  # “result” 用于存放请求指令处理是否成功，它的值为：“success 或 fail_xxx”
  }
}
```

- 消息推送：它是机器人主动向客户端发送信息的过程。这些信息可以包括机器人的序列号、当前运行状态、执行的操作等数据。通过及时地向客户端发送这些信息，机器人可以帮助客户端更好地理解它的工作状态，从而更好地使用它提供的服务。它的数据格式包含以下字段：
  - `accid`：机器人唯一序列号，标识机器人的唯一身份；
  - `title`：指令名称，以“`notify_`”为前缀；
  - `timestamp`：消息发出时间戳，单位为毫秒；
  - `guid`：消息的guid值，唯一标识这条消息；
  - `data`：存放消息数据内容。可以根据具体需求包含多个子字段，以存放请求指令所需的数据内容；
  - 示例如下：

```
{
  "accid": "PF_TRON1A_042",    # 机器人唯一序列号，标识机器人的唯一身份
  "title": "notify_xxx",  # 消息名称，以“notify_”为前缀
  "timestamp": 1672373633989, # 消息发出时间戳，单位为毫秒
  "guid": "746d937cd8094f6a98c9577aaf213d98", # 消息的guid值，唯一标识这条消息
  "data": { } # 存放消息数据内容
}
```

3. 查看软件序列号(ACCID)

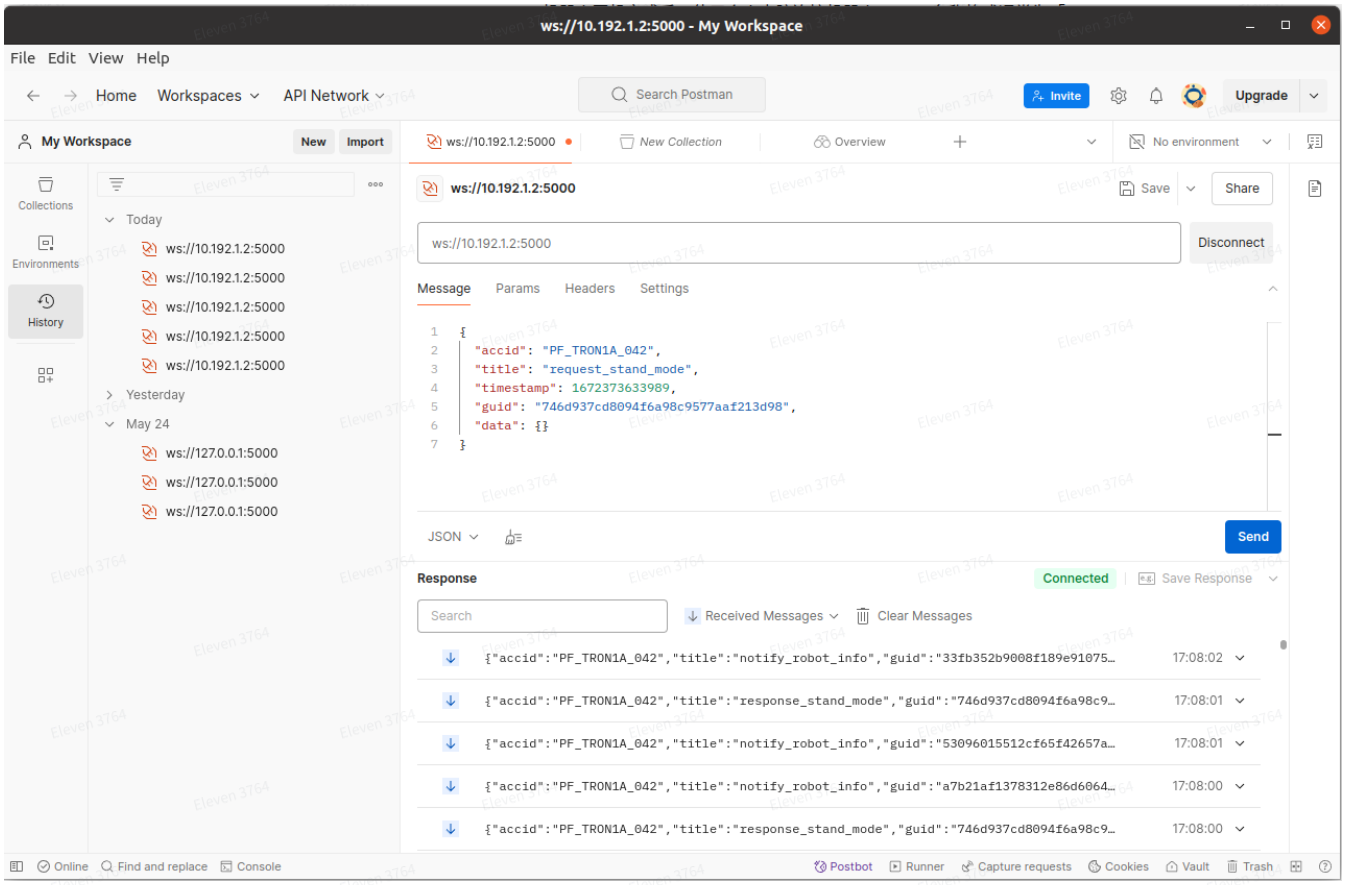
- 连接机器人无线网络
  - 机器人开机完成后，使用个人电脑连接机器人Wi-Fi，名称格式通常为「WF_TRON1A_xxx」
  - 输入Wi-Fi密码：`12345678`
- 在浏览器中输入`http://10.192.1.2:8080`可以进入“机器人信息页”，并查看机器人信息。如下图所示，页面中显示的SN (序列号) 为`PF_P441C_037`，其中`PF_P441C_037`便是此机器人的软件序列号。



4. 通信测试方法

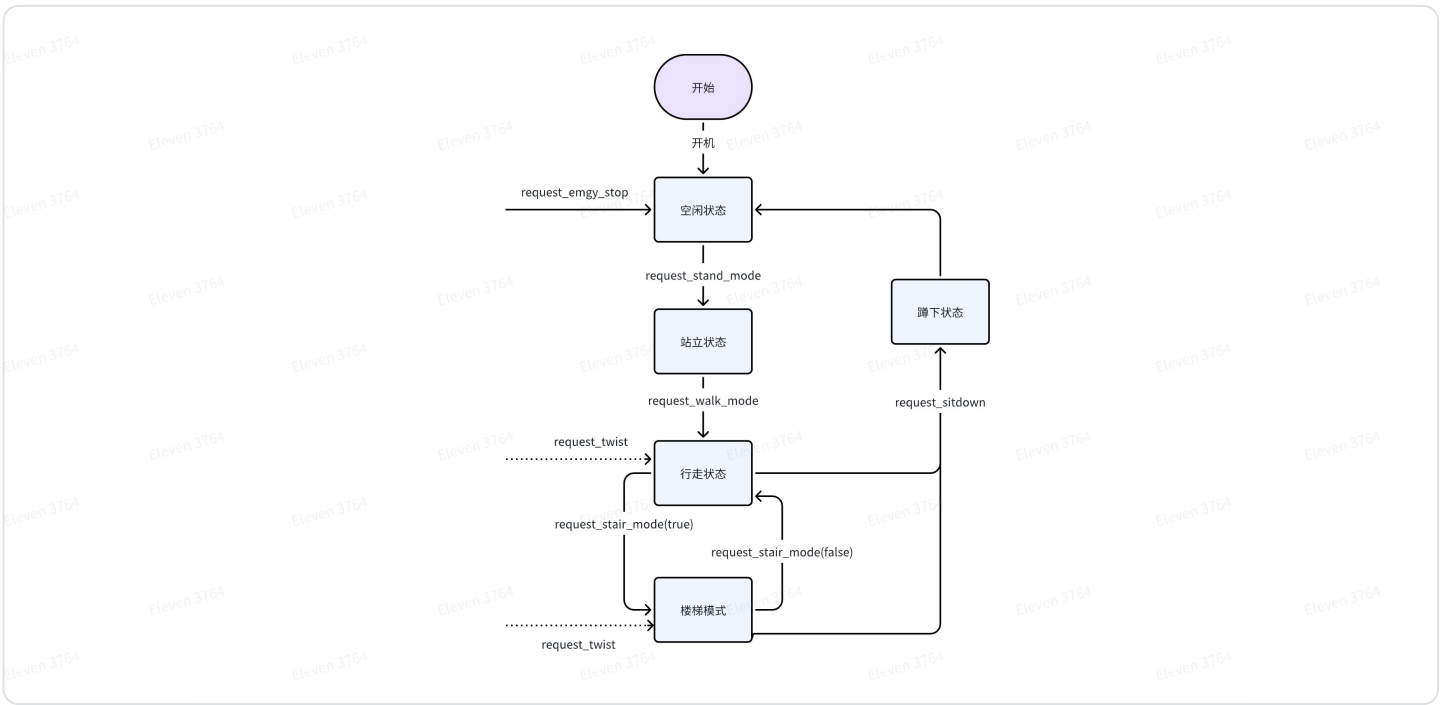
Postman是一个流行的API开发环境，可以用于测试WebSocket接口。使用Postman测试WebSocket接口，请按照以下步骤操作：

- 安装postman，下载地址：https://www.postman.com/downloads/?utm_source=postman-home；
- 打开Postman，并创建一个WebSocket的请求；
- 连接机器人无线网络
  - 机器人开机完成后，使用个人电脑连接机器人Wi-Fi，名称格式通常为「WF_TRON1A_xxx」
  - 输入Wi-Fi密码：`12345678`
- 在请求的URL中输入WebSocket接口的地址，例如，“ws://10.192.1.2:5000”；
- 在“Message”中，输入要发送的指令请求；
- 单击“Send”按钮，发送请求指令；
- 发送指令后，可以从服务器接收响应消息。使用Postman的响应窗口查看服务器返回的数据，并检查是否符合预期结果。



5. 协议接口定义

该机器人接口设计遵循与遥控器操控一致的流程和状态流转，确保调用顺序、响应时序及状态过渡与遥控器控制逻辑严格对齐。用户通过接口调用可获得如同使用遥控器的直观体验，同时支持遥控器与接口间的无缝切换，实现统一、稳定的机器人操控效果。



5.1 蹲起状态

5.1.1 请求：request_stand_mode

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_stand_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

5.1.2 响应：response_stand_mode

```
{
  "accid": "PF_TRON1A_042",
  "title": "response_stand_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
}
```

5.1.3 消息推送：notify_stand_mode

机器人站起过程失败或完成后，主动推送此消息。

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_stand_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.2 行走状态

5.2.1 请求：request_walk_mode

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

5.2.2 响应：response_walk_mode

```
{
  "accid": "PF_TRON1A_042",
  "title": "response_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
}
```

5.2.3 消息推送：notify_walk_mode

机器人站起过程失败或完成后，主动推送此消息。

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.3 控制行走

5.3.1 请求：request_twist

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_twist",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "x": 0.0,  // 前进后退速度，单位：m/s
    "y": 0.0,  // 横向行走速度，单位：m/s
    "z": 0.0   // 角速度，单位：rad/s
  }
}
```

5.3.2 响应：无

5.3.3 消息推送：notify_twist

机器人行走失败时，主动推送此消息。

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_twist",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "fail_motor"  // fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.4 蹲下

5.4.1 请求：request_sitdown

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_sitdown",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
```

```
}
```

5.4.2 响应：response_sitdown

```
{
  "accid": "PF_TRON1A_042",
  "title": "response_sitdown",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.4.3 消息推送：notify_sitdown

机器人蹲下过程失败或完成后，主动推送此消息。

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_sitdown",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.5 开启楼梯模式

5.5.1 请求：request_stair_mode

本功能仅适用于 TRON1 型号的双轮足机器人。

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_stair_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
  "data": {
      "enable": true  // true: 开启楼梯模式, false: 关闭楼梯模式
  }
}
```

5.5.2 响应：response_stair_mode

```
{
  "accid": "PF_TRON1A_042",
  "title": "response_stair_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.5.3 消息推送：无

5.6 紧急停止

5.6.1 请求：request_emgy_stop

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_emgy_stop",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

5.6.2 响应：response_emgy_stop

```
{
  "accid": "PF_TRON1A_042",
  "title": "response_emgy_stop",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
```

```
    "result": "success"  // success: 成功, fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

5.6.3 消息推送：无

5.7 开启IMU数据

5.7.1 请求：request_enable_imu

该功能用于开启IMU数据推送，开启后系统将主动推送IMU数据。

```
{
  "accid": "PF_TRON1A_042",
  "title": "request_enable_imu",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "enable": true  // true: 开启IMU, false: 禁用IMU
  }
}
```

5.7.2 响应：response_enable_imu

```
{
  "accid": "PF_TRON1A_042",
  "title": "response_enable_imu",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: 成功, fail_imu: IMU
  }
}
```

5.7.3 消息推送：notify_imu

开启IMU数据后，系统将主动推送包含IMU状态的消息。

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_imu",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "euler": [0.0, 0.0, 0.0],      // 欧拉角 [roll, pitch, yaw] in degrees
      "acc": [0.0, 0.0, 0.0],        // 加速度 [x, y, z] in m/s²
      "gyro": [0.0, 0.0, 0.0],       // 陀螺仪角速度 [x, y, z] in rad/s
      "quat": [0.0, 0.0, 0.0, 0.0]  // 四元数 [w, x, y, z]
  }
}
```

5.8 全局消息

5.8.1 机器人基本信息

机器人基本信息每秒上报一次，包含以下内容：

- `accid`：机器人序列号
- `title`：notify_robot_info
- `timestamp`：消息发出时间戳，单位为毫秒
- `guid`：消息的guid值，唯一标识这条消息
- `data`：存放消息内容
- 示例如下：

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "accid": "PF_TRON1A_042",
    "sw_version": "robot-tron1-2.0.10.20241111103012",
    "imu": "OK",
    "camera": "OK",
    "motor": "OK",
    "battery": 95
  }
}
```

5.8.2 非法指令消息

当机器人收到非法格式的请求指令时，发送此消息，包含以下内容：

- `accid`：机器人序列号
- `title`：notify_invalid_request
- `timestamp`：消息发出时间戳，单位为毫秒
- `guid`：消息的guid值，唯一标识这条消息
- `data`：存放消息内容
- 示例如下：

```
{
  "accid": "PF_TRON1A_042",
  "title": "notify_invalid_request",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": "返回原请求指令内容，便于客户端排查问题"
}
```

6. 协议接口调用示例

6.1 Linux C++ 示例实现

- 安装依赖：以Ubuntu 20.04系统为例，安装websocketpp、nlohmann/json和boost依赖：

```
sudo apt-get install libboost-all-dev libwebsocketpp-dev nlohmann-json3-dev
```

- 编译代码

```
g++ -std=c++11 -o websocket_client websocket_client.cpp -lssl -lcrypto -
lboost_system -lpthread
```

- 运行程序

```
./websocket_client
```

- websocket_client.cpp 实现

```cpp
#include <iostream>
#include <atomic>
#include <string>
#include <thread>
#include <chrono>
#include <websocketpp/client.hpp>
#include <websocketpp/config/asio.hpp>
#include <nlohmann/json.hpp>
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <boost/uuid/uuid_io.hpp>

using json = nlohmann::json;
using websocketpp::client;
using websocketpp::connection_hdl;

// Replace this ACCID value with your robot's actual serial number (SN)
static const std::string ACCID = "PF_TRON1A_042";

// WebSocket client instance
static client<websocketpp::config::asio> ws_client;

// Atomic flag for graceful exit
static std::atomic<bool> should_exit(false);

// Connection handle for sending messages
static connection_hdl current_hdl;

// Generate dynamic GUID
static std::string generate_guid() {
    boost::uuids::random_generator gen;
    boost::uuids::uuid u = gen();
    return boost::uuids::to_string(u);
}

// Send WebSocket request with title and data
static void send_request(const std::string& title, const json& data =
json::object()) {
    json message;

    // Adding necessary fields to the message
    message["accid"] = ACCID;
    message["title"] = title;
```

```
    message["timestamp"] = std::chrono::duration_cast<std::chrono::milliseconds>(
std::chrono::system_clock::now().time_since_epoch()).count();
    message["guid"] = generate_guid();
    message["data"] = data;

    std::string message_str = message.dump();

    // Send the message through WebSocket
    ws_client.send(current_hdl, message_str, websocketpp::frame::opcode::text);
}

// Handle user commands
static void handle_commands() {
    while (!should_exit) {
        std::string command;
        std::cout << "Enter command ('stand', 'walk', 'twist', 'sit', 'stair', 'stop', 'imu') or 'exit' to quit:" << std::endl;
        std::getline(std::cin, command);  // Read user input

        if (command == "exit") {
            should_exit = true;  // Exit flag to stop the loop
            break;
        } else if (command == "stand") {
            send_request("request_stand_mode");  // Send stand mode request
        } else if (command == "walk") {
            send_request("request_walk_mode");  // Send walk mode request
        } else if (command == "twist") {
            float x, y, z;
            std::cout << "Enter x, y, z values:" << std::endl;
            std::cin >> x >> y >> z;  // Get twist values from user
            send_request("request_twist", {{"x", x}, {"y", y}, {"z", z}});
        } else if (command == "sit") {
            send_request("request_sitdown");  // Send sit down request
        } else if (command == "stair") {
            bool enable;
            std::cout << "Enable stair mode (true/false):" << std::endl;
            std::cin >> enable;  // Get stair mode enable flag from user
            send_request("request_stair_mode", {{"enable", enable}});
        } else if (command == "stop") {
            send_request("request_emgy_stop");  // Send emergency stop request
        } else if (command == "imu") {
            std::string enable;
            std::cout << "Enable IMU (true/false):" << std::endl;
```

```
            std::cin >> enable;  // Get IMU enable flag from user
            send_request("request_enable_imu", {{"enable", enable == "true" ? true : false}});
        }
    }
}

// WebSocket open callback
static void on_open(connection_hdl hdl) {
    std::cout << "Connected!" << std::endl;

    // Save connection handle for sending messages later
    current_hdl = hdl;

    // Start handling commands in a separate thread
    std::thread(handle_commands).detach();
}

// WebSocket message callback
static void on_message(connection_hdl hdl, client<websocketpp::config::asio>::message_ptr msg) {
    std::cout << "Received: " << msg->get_payload() << std::endl;  // Print received message
}

// WebSocket close callback
static void on_close(connection_hdl hdl) {
    std::cout << "Connection closed." << std::endl;
}

// Close WebSocket connection
static void close_connection(connection_hdl hdl) {
    ws_client.close(hdl, websocketpp::close::status::normal, "Normal closure");  // Close connection normally
}

int main() {
    ws_client.init_asio();  // Initialize ASIO for WebSocket client

    // Set WebSocket event handlers
    ws_client.set_open_handler(&on_open);  // Set open handler
    ws_client.set_message_handler(&on_message);  // Set message handler
    ws_client.set_close_handler(&on_close);  // Set close handler

    std::string server_uri = "ws://10.192.1.2:5000";  // WebSocket server URI

```

```
    websocketpp::lib::error_code ec;
    client<websocketpp::config::asio>::connection_ptr con =
ws_client.get_connection(server_uri, ec);  // Get connection pointer

    if (ec) {
        std::cout << "Error: " << ec.message() << std::endl;
        return 1;  // Exit if connection error occurs
    }

    connection_hdl hdl = con->get_handle();  // Get connection handle
    ws_client.connect(con);  // Connect to server
    std::cout << "Press Ctrl+C to exit." << std::endl;

    // Run the WebSocket client loop
    ws_client.run();

    return 0;
}

```

6.2 Python 示例实现

- 环境准备：以Ubuntu 20.04系统为例，安装下面依赖

```
sudo apt install python3-dev python3-pip
sudo pip3 install websocket-client
```

- 运行脚本

```
python3 websocket_client.py
```

- websocket_client.py 实现

```
import json
import uuid
import threading
import time
import websocket
from datetime import datetime

# Replace this ACCID value with your robot's actual serial number (SN)
```

```python
ACCID = "PF_TRON1A_042"

# Atomic flag for graceful exit
should_exit = False

# WebSocket client instance
ws_client = None

# Generate dynamic GUID
def generate_guid():
    return str(uuid.uuid4())

# Send WebSocket request with title and data
def send_request(title, data=None):
    if data is None:
        data = {}

    # Create message structure with necessary fields
    message = {
        "accid": ACCID,
        "title": title,
        "timestamp": int(time.time() * 1000),  # Current timestamp in milliseconds
        "guid": generate_guid(),
        "data": data
    }

    message_str = json.dumps(message)

    # Send the message through WebSocket if client is connected
    if ws_client:
        ws_client.send(message_str)

# Handle user commands
def handle_commands():
    global should_exit
    while not should_exit:
        command = input("Enter command ('stand', 'walk', 'twist', 'sit', 'stair', 'stop', 'imu') or 'exit' to quit:\n")

        if command == "exit":
            should_exit = True  # Set exit flag to stop the loop
            break
        elif command == "stand":
            send_request("request_stand_mode")  # Send stand mode request
        elif command == "walk":
            send_request("request_walk_mode")  # Send walk mode request
```

```
54          elif command == "twist":
55              # Get twist values from user
56              x = float(input("Enter x value:"))
57              y = float(input("Enter y value:"))
58              z = float(input("Enter z value:"))
59              send_request("request_twist", {"x": x, "y": y, "z": z})
60          elif command == "sit":
61              send_request("request_sitdown")  # Send sit down request
62          elif command == "stair":
63              # Get stair mode enable flag from user
64              enable = input("Enable stair mode
(true/false):").strip().lower() == 'true'
65              send_request("request_stair_mode", {"enable": enable})
66          elif command == "stop":
67              send_request("request_emgy_stop")  # Send emergency stop request
68          elif command == "imu":
69              # Get IMU enable flag from user
70              enable = input("Enable IMU (true/false):").strip().lower() ==
'true'
71              send_request("request_enable_imu", {"enable": enable})
72
73  # WebSocket on_open callback
74  def on_open(ws):
75      print("Connected!")
76      # Start handling commands in a separate thread
77      threading.Thread(target=handle_commands, daemon=True).start()
78
79  # WebSocket on_message callback
80  def on_message(ws, message):
81      print(f"Received message: {message}")  # Print the received message
82
83  # WebSocket on_close callback
84  def on_close(ws, close_status_code, close_msg):
85      print("Connection closed.")
86
87  # Close WebSocket connection
88  def close_connection(ws):
89      ws.close()
90
91  def main():
92      global ws_client
93
94      # Create WebSocket client instance
95      ws_client = websocket.WebSocketApp(
96          "ws://10.192.1.2:5000",  # WebSocket server URI
97          on_open=on_open,
98          on_message=on_message,
```

```
        on_close=on_close
    )

    # Run WebSocket client loop
    print("Press Ctrl+C to exit.")
    ws_client.run_forever()

if __name__ == "__main__":
    main()

```

6.3 JavaScript 示例实现

- HTML 页面：为了便于与用户交互，您可以在 HTML 页面中添加一个输入框，用户可以在其中输入命令。以下是 index.html 实现：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>WebSocket Robot Control</title>
    <style>
        #commandInput {
            width: 400px; /* Adjust the width to make it wider */
            padding: 10px;
            font-size: 14px;
        }
    </style>
</head>
<body>
    <h2>Robot Control Commands</h2>
    <input type="text" id="commandInput" placeholder="Enter command ('stand', 'walk', 'twist', 'sit', 'stair', 'stop', 'imu')">
    <p>Type a command and press Enter.</p>

    <script src="robotControl.js"></script>
</body>
</html>

```

- robotControl.js 实现：

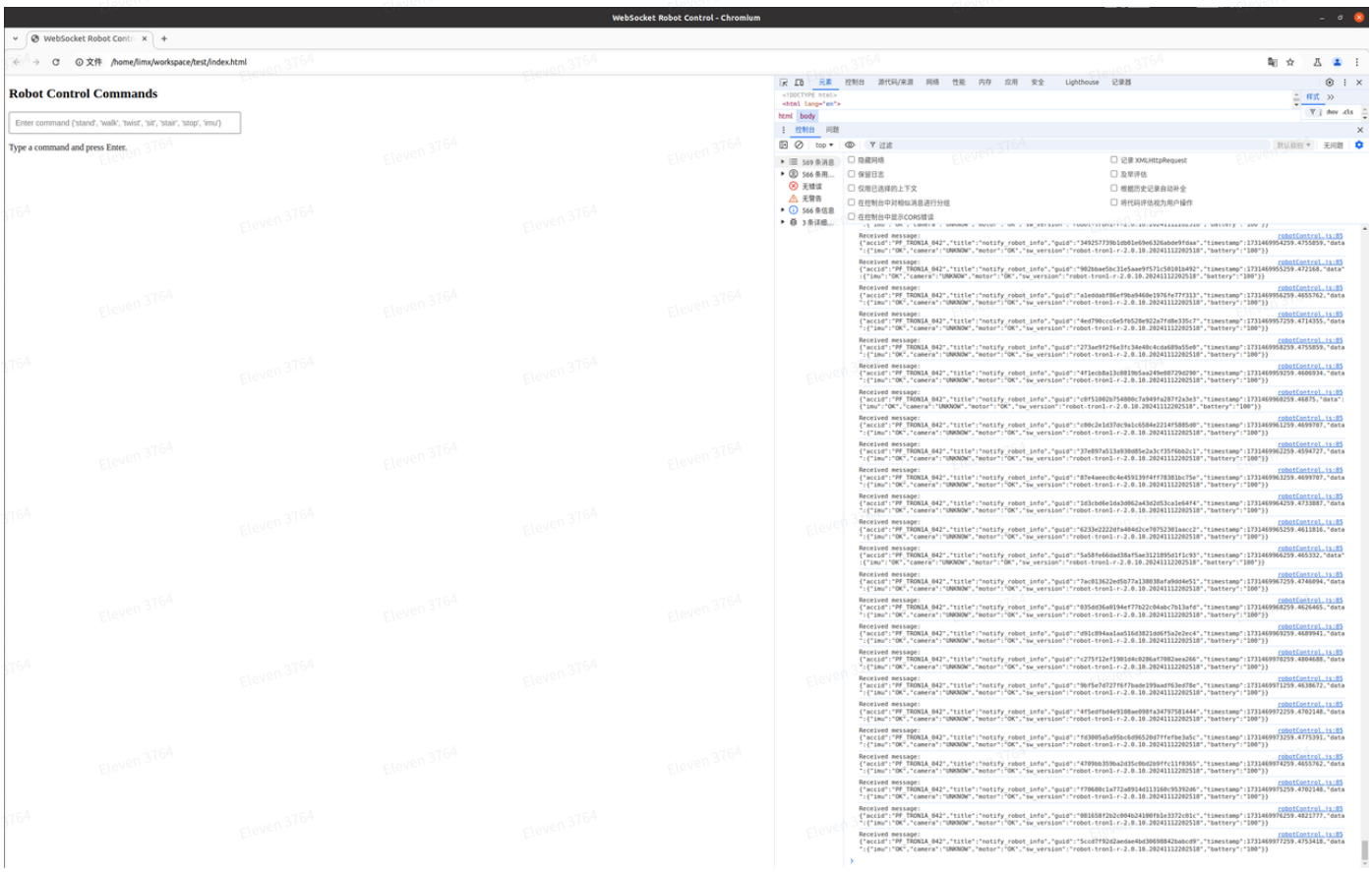
```javascript
// Replace this ACCID value with your robot's actual serial number (SN)
const ACCID = "PF_TRON1A_042";

// WebSocket client instance
let wsClient = null;

// Generate dynamic GUID
function generateGuid() {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
        const r = Math.random() * 16 | 0,
              v = c === 'x' ? r : (r & 0x3 | 0x8);
        return v.toString(16);
    });
}

// Send WebSocket request with title and data
function sendRequest(title, data = {}) {
    const message = {
        accid: ACCID,
        title: title,
        timestamp: Date.now(),  // Current timestamp in milliseconds
        guid: generateGuid(),
        data: data
    };

    // Send the message through WebSocket if client is connected
    if (wsClient && wsClient.readyState === WebSocket.OPEN) {
        wsClient.send(JSON.stringify(message));
    }
}

// Handle user commands
function handleCommands() {
    const commandInput = document.getElementById('commandInput');
    commandInput.addEventListener('keydown', function(event) {
        if (event.key === 'Enter') {
            const command = commandInput.value.trim();
            commandInput.value = '';  // Clear input field

            switch (command) {
                case 'stand':
                    sendRequest('request_stand_mode');
                    break;
                case 'walk':
                    sendRequest('request_walk_mode');
                    break;
```

```
                case 'twist':
                    const x = parseFloat(prompt("Enter x value:"));
                    const y = parseFloat(prompt("Enter y value:"));
                    const z = parseFloat(prompt("Enter z value:"));
                    sendRequest('request_twist', {x, y, z});
                    break;
                case 'sit':
                    sendRequest('request_sitdown');
                    break;
                case 'stair':
                    const enableStair = prompt("Enable stair mode (true/false):").toLowerCase() === 'true';
                    sendRequest('request_stair_mode', {enable: enableStair});
                    break;
                case 'stop':
                    sendRequest('request_emgy_stop');
                    break;
                case 'imu':
                    const enableImu = prompt("Enable IMU (true/false):").toLowerCase() === 'true';
                    sendRequest('request_enable_imu', {enable: enableImu});
                    break;
                case 'exit':
                    wsClient.close();
                    break;
                default:
                    alert("Invalid command. Try again.");
            }
        }
    });
}

// WebSocket onOpen callback
function onOpen() {
    console.log("Connected!");
    handleCommands();
}

// WebSocket onMessage callback
function onMessage(event) {
    console.log("Received message:", event.data);
}

// WebSocket onClose callback
function onClose(event) {
    console.log("Connection closed.");
```

```
}

// Initialize WebSocket client
function initWebSocket() {
    // Replace this URL with your WebSocket server URI
    wsClient = new WebSocket('ws://10.192.1.2:5000');

    wsClient.onopen = onOpen;
    wsClient.onmessage = onMessage;
    wsClient.onclose = onClose;

    console.log("Press Ctrl+C to exit.");
}

// Start WebSocket connection when the page loads
window.onload = initWebSocket;
```

- 运行程序

  将 `index.html` 和 `robotControl.js` 文件保存到同一目录下，然后在浏览器中打开 `index.html` 运行。你可以通过浏览器的开发者工具查看接收到的详细信息。



6.4 Go 示例实现

- 首先需要安装 `gorilla/websocket` 包：

```
go get github.com/gorilla/websocket
```

- Go实现的代码如下：

```
package main

import (
        "encoding/json"
        "fmt"
        "github.com/gorilla/websocket"
        "time"
        "strings"
        "github.com/google/uuid"
)

// Global variables
var wsClient *websocket.Conn
var shouldExit bool

// Replace with your robot's serial number
var ACCID = "PF_TRON1A_042"

// Generate dynamic GUID
func generateGUID() string {
        return uuid.New().String()
}

// Send WebSocket request with title and data
func sendRequest(title string, data map[string]interface{}) {
        if data == nil {
                data = make(map[string]interface{})
        }

        message := map[string]interface{}{
                "accid":     ACCID,
                "title":     title,
                "timestamp": time.Now().UnixMilli(),
                "guid":      generateGUID(),
                "data":      data,
        }

        messageBytes, err := json.Marshal(message)
        if err != nil {
                fmt.Println("Error marshaling message:", err)
```

```
                return
        }

        // Send the message through WebSocket if client is connected
        if wsClient != nil {
                err = wsClient.WriteMessage(websocket.TextMessage,
messageBytes)
                if err != nil {
                        fmt.Println("Error sending message:", err)
                }
        }
}

// Handle user commands
func handleCommands() {
        var command string
        for !shouldExit {
                fmt.Println("Enter command ('stand', 'walk', 'twist',
'sit', 'stair', 'stop', 'imu') or 'exit' to quit:")
                fmt.Scanln(&command)

                command = strings.TrimSpace(command)
                switch command {
                case "exit":
                        shouldExit = true
                        return
                case "stand":
                        sendRequest("request_stand_mode", nil)
                case "walk":
                        sendRequest("request_walk_mode", nil)
                case "twist":
                        var x, y, z float64
                        fmt.Println("Enter x value:")
                        fmt.Scanln(&x)
                        fmt.Println("Enter y value:")
                        fmt.Scanln(&y)
                        fmt.Println("Enter z value:")
                        fmt.Scanln(&z)
                        sendRequest("request_twist", map[string]interface{}
{"x": x, "y": y, "z": z})
                case "sit":
                        sendRequest("request_sitdown", nil)
                case "stair":
                        var enable bool
                        fmt.Println("Enable stair mode (true/false):")
                        var input string
                        fmt.Scanln(&input)
```

```
85                          enable = strings.ToLower(input) == "true"
86                          sendRequest("request_stair_mode",
    map[string]interface{}{"enable": enable})
87                  case "stop":
88                          sendRequest("request_emgy_stop", nil)
89                  case "imu":
90                          var enable bool
91                          fmt.Println("Enable IMU (true/false):")
92                          var input string
93                          fmt.Scanln(&input)
94                          enable = strings.ToLower(input) == "true"
95                          sendRequest("request_enable_imu",
    map[string]interface{}{"enable": enable})
96                  }
97          }
98  }
99
100 // WebSocket onOpen callback
101 func onOpen(ws *websocket.Conn) {
102         fmt.Println("Connected!")
103         wsClient = ws
104         go handleCommands()
105 }
106
107 // WebSocket onMessage callback
108 func onMessage(ws *websocket.Conn, message []byte) {
109         fmt.Println("Received message:", string(message))
110 }
111
112 // WebSocket onClose callback
113 func onClose(ws *websocket.Conn, code int, text string) {
114         fmt.Println("Connection closed. Code:", code, "Message:", text)
115 }
116
117 // Connect to the WebSocket server
118 func connectWebSocket() {
119         url := "ws://10.192.1.2:5000" // WebSocket server URI
120
121         conn, _, err := websocket.DefaultDialer.Dial(url, nil)
122         if err != nil {
123                 fmt.Println("Error connecting to WebSocket server:", err)
124                 return
125         }
126
127         onOpen(conn)
128
129         // Start receiving messages from the server
```

```
130         go func() {
131                 for {
132                         _, message, err := conn.ReadMessage()
133                         if err != nil {
134                                 fmt.Println("Error reading message:", err)
135                                 break
136                         }
137                         onMessage(conn, message)
138                 }
139         }()
140
141         // Wait until WebSocket connection is closed
142         select {}
143 }
144
145 // Main function
146 func main() {
147         defer func() {
148                 if wsClient != nil {
149                         wsClient.Close()
150                 }
151         }()
152
153         // Connect to WebSocket server
154         go connectWebSocket()
155
156         // Block main goroutine to allow handling commands
157         select {}
158 }
159
```

6.5 Java 示例实现

```
1  import org.java-websocket.client.WebSocketClient;
2  import org.java-websocket.handshake.ServerHandshake;
3  import org.json.JSONObject;
4
5  import java.net.URI;
6  import java.util.Scanner;
7  import java.util.UUID;
8
9  public class WebSocketExample {
10
11     // Replace this ACCID value with your robot's actual serial number (SN)
12     private static final String ACCID = "PF_TRON1A_042";
```

```
13
14      // Atomic flag for graceful exit
15      private static volatile boolean shouldExit = false;
16
17      // WebSocket client instance
18      private static WebSocketClient wsClient = null;
19
20      // Generate dynamic GUID
21      private static String generateGuid() {
22          return UUID.randomUUID().toString();
23      }
24
25      // Send WebSocket request with title and data
26      private static void sendRequest(String title, JSONObject data) {
27          if (data == null) {
28              data = new JSONObject();
29          }
30
31          // Create message structure with necessary fields
32          JSONObject message = new JSONObject();
33          message.put("accid", ACCID);
34          message.put("title", title);
35          message.put("timestamp", System.currentTimeMillis());  // Current
    timestamp in milliseconds
36          message.put("guid", generateGuid());
37          message.put("data", data);
38
39          String messageStr = message.toString();
40
41          // Send the message through WebSocket if client is connected
42          if (wsClient != null && wsClient.isOpen()) {
43              wsClient.send(messageStr);
44          }
45      }
46
47      // Handle user commands
48      private static void handleCommands() {
49          Scanner scanner = new Scanner(System.in);
50          while (!shouldExit) {
51              System.out.println("Enter command ('stand', 'walk', 'twist',
    'sit', 'stair', 'stop', 'imu') or 'exit' to quit:");
52              String command = scanner.nextLine().trim();
53
54              switch (command) {
55                  case "exit":
56                      shouldExit = true;
57                      break;
```

```
                case "stand":
                    sendRequest("request_stand_mode", null);
                    break;
                case "walk":
                    sendRequest("request_walk_mode", null);
                    break;
                case "twist":
                    System.out.print("Enter x value: ");
                    double x = scanner.nextDouble();
                    System.out.print("Enter y value: ");
                    double y = scanner.nextDouble();
                    System.out.print("Enter z value: ");
                    double z = scanner.nextDouble();
                    scanner.nextLine();  // Consume the newline
                    JSONObject twistData = new JSONObject();
                    twistData.put("x", x);
                    twistData.put("y", y);
                    twistData.put("z", z);
                    sendRequest("request_twist", twistData);
                    break;
                case "sit":
                    sendRequest("request_sitdown", null);
                    break;
                case "stair":
                    System.out.print("Enable stair mode (true/false): ");
                    boolean enableStair =
    scanner.nextLine().trim().equalsIgnoreCase("true");
                    JSONObject stairData = new JSONObject();
                    stairData.put("enable", enableStair);
                    sendRequest("request_stair_mode", stairData);
                    break;
                case "stop":
                    sendRequest("request_emgy_stop", null);
                    break;
                case "imu":
                    System.out.print("Enable IMU (true/false): ");
                    boolean enableImu =
    scanner.nextLine().trim().equalsIgnoreCase("true");
                    JSONObject imuData = new JSONObject();
                    imuData.put("enable", enableImu);
                    sendRequest("request_enable_imu", imuData);
                    break;
                default:
                    System.out.println("Invalid command. Try again.");
                    break;
            }
        }
```

```java
    }

    // WebSocket onOpen callback
    private static void onOpen() {
        System.out.println("Connected!");
        // Start handling commands in a separate thread
        new Thread(WebSocketExample::handleCommands).start();
    }

    // WebSocket onMessage callback
    private static void onMessage(String message) {
        System.out.println("Received message: " + message);
    }

    // WebSocket onClose callback
    private static void onClose(int code, String reason, boolean remote) {
        System.out.println("Connection closed. Reason: " + reason);
    }

    public static void main(String[] args) {
        // WebSocket server URI
        URI serverUri = URI.create("ws://10.192.1.2:5000");

        // Create WebSocket client instance
        wsClient = new WebSocketClient(serverUri) {

            @Override
            public void onOpen(ServerHandshake handshakedata) {
                onOpen();
            }

            @Override
            public void onMessage(String message) {
                onMessage(message);
            }

            @Override
            public void onClose(int code, String reason, boolean remote) {
                onClose(code, reason, remote);
            }

            @Override
            public void onError(Exception ex) {
                System.out.println("Error: " + ex.getMessage());
            }
        };

```

```
        // Connect to the WebSocket server
        wsClient.connect();
        System.out.println("Press Ctrl+C to exit.");
    }
}

```

6.6 Windows C++ 示例实现

- 在 Windows 上，确保您已经安装并配置了 `websocketpp`、`nlohmann/json` 和 `boost` 库。如果使用 `vcpkg`，您可以运行以下命令安装依赖：

```
vcpkg install websocketpp boost nlohmann-json
```

- 代码实现

```
#include <iostream>
#include <atomic>
#include <string>
#include <thread>
#include <chrono>
#include <websocketpp/client.hpp>
#include <websocketpp/config/asio.hpp>
#include <nlohmann/json.hpp>
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <boost/uuid/uuid_io.hpp>

using json = nlohmann::json;
using websocketpp::client;
using websocketpp::connection_hdl;

// Replace this ACCID value with your robot's actual serial number (SN)
static const std::string ACCID = "PF_TRON1A_042";

// WebSocket client instance
static client<websocketpp::config::asio> ws_client;

// Atomic flag for graceful exit
static std::atomic<bool> should_exit(false);

// Connection handle for sending messages
static connection_hdl current_hdl;
```

```cpp

// Generate dynamic GUID
static std::string generate_guid() {
    boost::uuids::random_generator gen;
    boost::uuids::uuid u = gen();
    return boost::uuids::to_string(u);
}

// Send WebSocket request with title and data
static void send_request(const std::string& title, const json& data = json::object()) {
    json message;

    // Adding necessary fields to the message
    message["accid"] = ACCID;
    message["title"] = title;
    message["timestamp"] = std::chrono::duration_cast<std::chrono::milliseconds>(

std::chrono::system_clock::now().time_since_epoch()).count();
    message["guid"] = generate_guid();
    message["data"] = data;

    std::string message_str = message.dump();

    // Send the message through WebSocket
    ws_client.send(current_hdl, message_str, websocketpp::frame::opcode::text);
}

// Handle user commands
static void handle_commands() {
    while (!should_exit) {
        std::string command;
        std::cout << "Enter command ('stand', 'walk', 'twist', 'sit', 'stair', 'stop', 'imu') or 'exit' to quit:" << std::endl;
        std::getline(std::cin, command);  // Read user input

        if (command == "exit") {
            should_exit = true;  // Exit flag to stop the loop
            break;
        } else if (command == "stand") {
            send_request("request_stand_mode");  // Send stand mode request
        } else if (command == "walk") {
            send_request("request_walk_mode");  // Send walk mode request
        } else if (command == "twist") {
            float x, y, z;
```

```
70            std::cout << "Enter x, y, z values:" << std::endl;
71            std::cin >> x >> y >> z;  // Get twist values from user
72            send_request("request_twist", {{"x", x}, {"y", y}, {"z", z}});
73        } else if (command == "sit") {
74            send_request("request_sitdown");  // Send sit down request
75        } else if (command == "stair") {
76            bool enable;
77            std::cout << "Enable stair mode (true/false):" << std::endl;
78            std::cin >> enable;  // Get stair mode enable flag from user
79            send_request("request_stair_mode", {{"enable", enable}});
80        } else if (command == "stop") {
81            send_request("request_emgy_stop");  // Send emergency stop
    request
82        } else if (command == "imu") {
83            std::string enable;
84            std::cout << "Enable IMU (true/false):" << std::endl;
85            std::cin >> enable;  // Get IMU enable flag from user
86            send_request("request_enable_imu", {{"enable", enable == "true"
    ? true : false}});
87        }
88    }
89 }
90
91 // WebSocket open callback
92 static void on_open(connection_hdl hdl) {
93     std::cout << "Connected!" << std::endl;
94
95     // Save connection handle for sending messages later
96     current_hdl = hdl;
97
98     // Start handling commands in a separate thread
99     std::thread(handle_commands).detach();
100 }
101
102 // WebSocket message callback
103 static void on_message(connection_hdl hdl,
    client<websocketpp::config::asio>::message_ptr msg) {
104     std::cout << "Received: " << msg->get_payload() << std::endl;  // Print
    received message
105 }
106
107 // WebSocket close callback
108 static void on_close(connection_hdl hdl) {
109     std::cout << "Connection closed." << std::endl;
110 }
111
112 // Close WebSocket connection
```

```cpp
static void close_connection(connection_hdl hdl) {
    ws_client.close(hdl, websocketpp::close::status::normal, "Normal closure");  // Close connection normally
}

int main() {
    ws_client.init_asio();  // Initialize ASIO for WebSocket client

    // Set WebSocket event handlers
    ws_client.set_open_handler(&on_open);  // Set open handler
    ws_client.set_message_handler(&on_message);  // Set message handler
    ws_client.set_close_handler(&on_close);  // Set close handler

    std::string server_uri = "ws://10.192.1.2:5000";  // WebSocket server URI

    websocketpp::lib::error_code ec;
    client<websocketpp::config::asio>::connection_ptr con = ws_client.get_connection(server_uri, ec);  // Get connection pointer

    if (ec) {
        std::cout << "Error: " << ec.message() << std::endl;
        return 1;  // Exit if connection error occurs
    }

    connection_hdl hdl = con->get_handle();  // Get connection handle
    ws_client.connect(con);  // Connect to server
    std::cout << "Press Ctrl+C to exit." << std::endl;

    // Run the WebSocket client loop
    ws_client.run();

    return 0;
}

```

6.7 Windows C# 示例实现

```csharp
using System;
using System.Net.WebSockets;
using System.Text;
using System.Text.Json;
using System.Threading;
using System.Threading.Tasks;

```

```
class WebSocketClient
{
    // Replace this ACCID value with your robot's actual serial number (SN)
    private static readonly string ACCID = "PF_TRON1A_042";
    private static ClientWebSocket ws;

    public static async Task Main(string[] args)
    {
        ws = new ClientWebSocket();
        Uri serverUri = new Uri("ws://10.192.1.2:5000");  // WebSocket server URI

        // Handle program exit to close WebSocket connection
        Console.CancelKeyPress += async (sender, e) =>
        {
            e.Cancel = true;
            await CloseConnection();
            Environment.Exit(0);
        };

        try
        {
            await ws.ConnectAsync(serverUri, CancellationToken.None);
            Console.WriteLine("Connected to WebSocket server!");

            _ = Task.Run(async () => await ReceiveMessages());  // Start listening for messages

            while (true)
            {
                Console.WriteLine("Enter a command ('stand', 'walk', 'twist', 'sit', 'stair', 'stop', 'imu') or 'exit' to quit:");
                string command = Console.ReadLine();

                if (command == "exit")
                {
                    await CloseConnection();
                    break;
                }
                else if (command == "stand")
                {
                    await RequestStandMode();
                }
                else if (command == "walk")
                {
                    await RequestWalkMode();
                }
```

```
52                 else if (command == "twist")
53                 {
54                     Console.WriteLine("Enter x, y, z values:");
55                     float x = float.Parse(Console.ReadLine());
56                     float y = float.Parse(Console.ReadLine());
57                     float z = float.Parse(Console.ReadLine());
58                     await RequestTwist(x, y, z);
59                 }
60                 else if (command == "sit")
61                 {
62                     await RequestSitDown();
63                 }
64                 else if (command == "stair")
65                 {
66                     Console.WriteLine("Enter stair mode enable
   (true/false):");
67                     bool enable = bool.Parse(Console.ReadLine());
68                     await RequestStairMode(enable);
69                 }
70                 else if (command == "stop")
71                 {
72                     await RequestEmgyStop();
73                 }
74                 else if (command == "imu")
75                 {
76                     Console.WriteLine("Enable IMU (true/false):");
77                     bool enable = bool.Parse(Console.ReadLine());
78                     await RequestEnableImu(enable);
79                 }
80             }
81         }
82         catch (Exception ex)
83         {
84             Console.WriteLine($"Exception: {ex.Message}");
85         }
86         finally
87         {
88             ws?.Dispose();
89         }
90     }
91 
92     private static async Task RequestStandMode()
93     {
94         // Sends a stand mode request
95         var request = new
96         {
97             accid = ACCID,
```

```
            title = "request_stand_mode",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
            guid = Guid.NewGuid().ToString("N"),
            data = new { }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task RequestWalkMode()
    {
        // Sends a walk mode request
        var request = new
        {
            accid = ACCID,
            title = "request_walk_mode",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
            guid = Guid.NewGuid().ToString("N"),
            data = new { }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task RequestTwist(float x, float y, float z)
    {
        // Sends a twist control request with x, y, z values
        var request = new
        {
            accid = ACCID,
            title = "request_twist",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
            guid = Guid.NewGuid().ToString("N"),
            data = new
            {
                x = x,
                y = y,
                z = z
            }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task RequestSitDown()
    {
        // Sends a sit down request
        var request = new
        {
            accid = ACCID,
```

```
            title = "request_sitdown",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
            guid = Guid.NewGuid().ToString("N"),
            data = new { }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task RequestStairMode(bool enable)
    {
        // Sends a stair mode request, enabling or disabling
        var request = new
        {
            accid = ACCID,
            title = "request_stair_mode",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
            guid = Guid.NewGuid().ToString("N"),
            data = new
            {
                enable = enable
            }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task RequestEmgyStop()
    {
        // Sends an emergency stop request
        var request = new
        {
            accid = ACCID,
            title = "request_emgy_stop",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
            guid = Guid.NewGuid().ToString("N"),
            data = new { }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task RequestEnableImu(bool enable)
    {
        // Sends a request to enable/disable IMU data streaming
        var request = new
        {
            accid = ACCID,
            title = "request_enable_imu",
            timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds(),
```

```
            guid = Guid.NewGuid().ToString("N"),
            data = new
            {
                enable = enable
            }
        };
        await SendMessage(JsonSerializer.Serialize(request));
    }

    private static async Task SendMessage(string message)
    {
        // Sends a message over WebSocket
        byte[] buffer = Encoding.UTF8.GetBytes(message);
        await ws.SendAsync(new ArraySegment<byte>(buffer),
WebSocketMessageType.Text, true, CancellationToken.None);
        Console.WriteLine($"Sent: {message}");
    }

    private static async Task ReceiveMessages()
    {
        // Receives messages from WebSocket server
        byte[] buffer = new byte[1024];

        while (ws.State == WebSocketState.Open)
        {
            WebSocketReceiveResult result = await ws.ReceiveAsync(new
ArraySegment<byte>(buffer), CancellationToken.None);

            if (result.MessageType == WebSocketMessageType.Close)
            {
                await ws.CloseAsync(WebSocketCloseStatus.NormalClosure,
"Server closed", CancellationToken.None);
                Console.WriteLine("WebSocket server closed connection.");
                break;
            }
            else
            {
                string message = Encoding.UTF8.GetString(buffer, 0,
result.Count);
                Console.WriteLine($"Received: {message}");
            }
        }
    }

    private static async Task CloseConnection()
    {
        // Closes the WebSocket connection gracefully
```

```
235         if (ws.State == WebSocketState.Open)
236         {
237             await ws.CloseAsync(WebSocketCloseStatus.NormalClosure, "Client
    closing", CancellationToken.None);
238             Console.WriteLine("WebSocket connection closed.");
239         }
240     }
241 }
242
```