Limx Oli EDU SDK 开发指南

生成时间：2026-11-04

文档版本	修订日期	修订内容	适用主控软件版本 （若不满足请前往官网下载中心获取最新版本主控软件包进行升级）
V1.0	2026.02.27	初版	V2.1.21 及以上
V1.1	2026.07.17	1. 删除 MCP Server 2. 新增关闭相机驱动自启动功能	V2.2.11 及以上
V1.2	2026.07.30	1. 补充获取相机数据步骤说明	V2.2.11 及以上
V1.3	2026.09.16	1. 新增音频设备接口	V2.2.11 及以上
V1.4	2026.10.26	1. 新增全身操作接口 2. 新增强脑2代触觉灵巧手接口	V2.5.9 及以上

1 大模型调用接口

1.1 内置大模型

模型	备注
qwen2.5:3b	由阿里云推出的通义千问 2.5 系列模型，参数量为 30 亿。具备较强的语言理解和生成能力，适用于文本生成、对话交互等场景。
qwen2.5:1.5b	通义千问 2.5 系列中参数量为 15 亿的模型，相对轻量，在一些对计算资源要求不高的场景中也能有较好表现 。
qwen2.5:0.5b	参数量为 5 亿的轻量级模型，便于在终端设备或资源有限的环境下运行。
llama3.2:3b	Meta 公司开发的大语言模型 LLaMA 3.2 版本中的 30 亿参数模型，在自然语言处理任务上表现出色，开源特性使得开发者可以基于它进行二次开发。
llama3.2:1b	LLaMA 3.2 系列的 10 亿参数模型，模型规模较小，训练和推理速度相对较快。

模型	备注
deepseek-r1:1.5b	DeepSeek 推出的模型，参数量 15 亿，在多种语言处理任务中具备一定的竞争力。

1.2 大模型调用

Oli 已通过 ollama 在本地完成上述大模型部署，只需将您的设备连接至与机器人相同的网络，即可按以下方法进行调用。

1.2.1 使用 Curl 调用

▼

复制代码

```
curl http://10.192.1.3:11434/api/generate \
  -H "Content-Type: application/json" \
  -d '{
        "model": "qwen2.5:3b",
        "prompt": "请写一首描述春天的四言绝句？",
        "temperature": 0.7,
        "max_tokens": 200,
        "stream": false
      }'
```

1.2.2 使用 Python 调用

▼

复制代码

```
import requests

url = "http://10.192.1.3:11434/api/generate"
data = {
    "model": "qwen2.5:3b",
    "prompt": "请写一首描述春天的四言绝句？",
    "temperature": 0.7,
    "max_tokens": 200,
    "stream": False
}

response = requests.post(url, json=data)
if response.status_code == 200:
    print(str(response.json()['response']))
else:
```

```
    print(f"Request failed with status code: {response.status_code}, Error: {response.tex
t}")
```

1.2.3 使用 C++ 调用

以 Ubuntu 20.04 及以上系统版本为例：

- **安装依赖**

▼

复制代码

```
sudo apt-get install libcurl4-openssl-dev nlohmann-json3-dev
```

- **代码实现(llm_demo.cpp)**

▶

复制代码

- **编译并运行**

▼

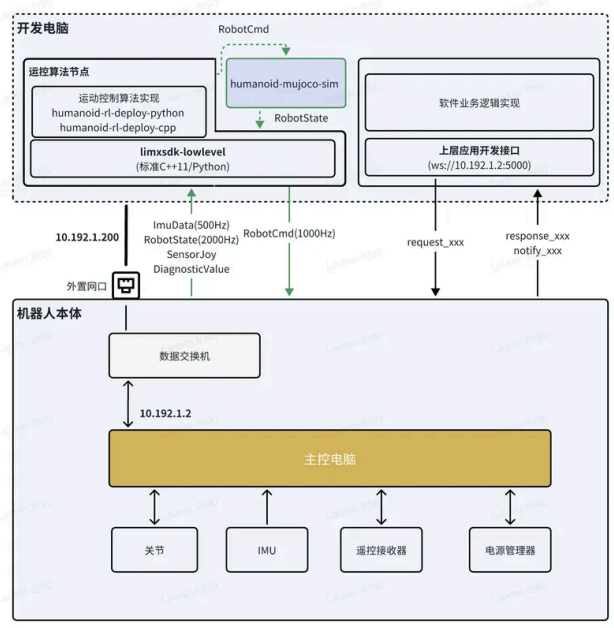
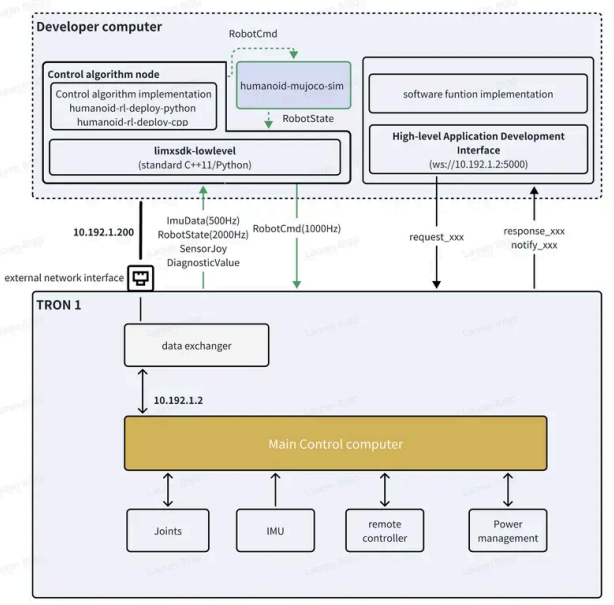
复制代码

```
# Compile with C++11 support
g++ -std=c++11 -o llm_demo llm_demo.cpp -lcurl

# Execute
./llm_demo
```

2 通讯架构图

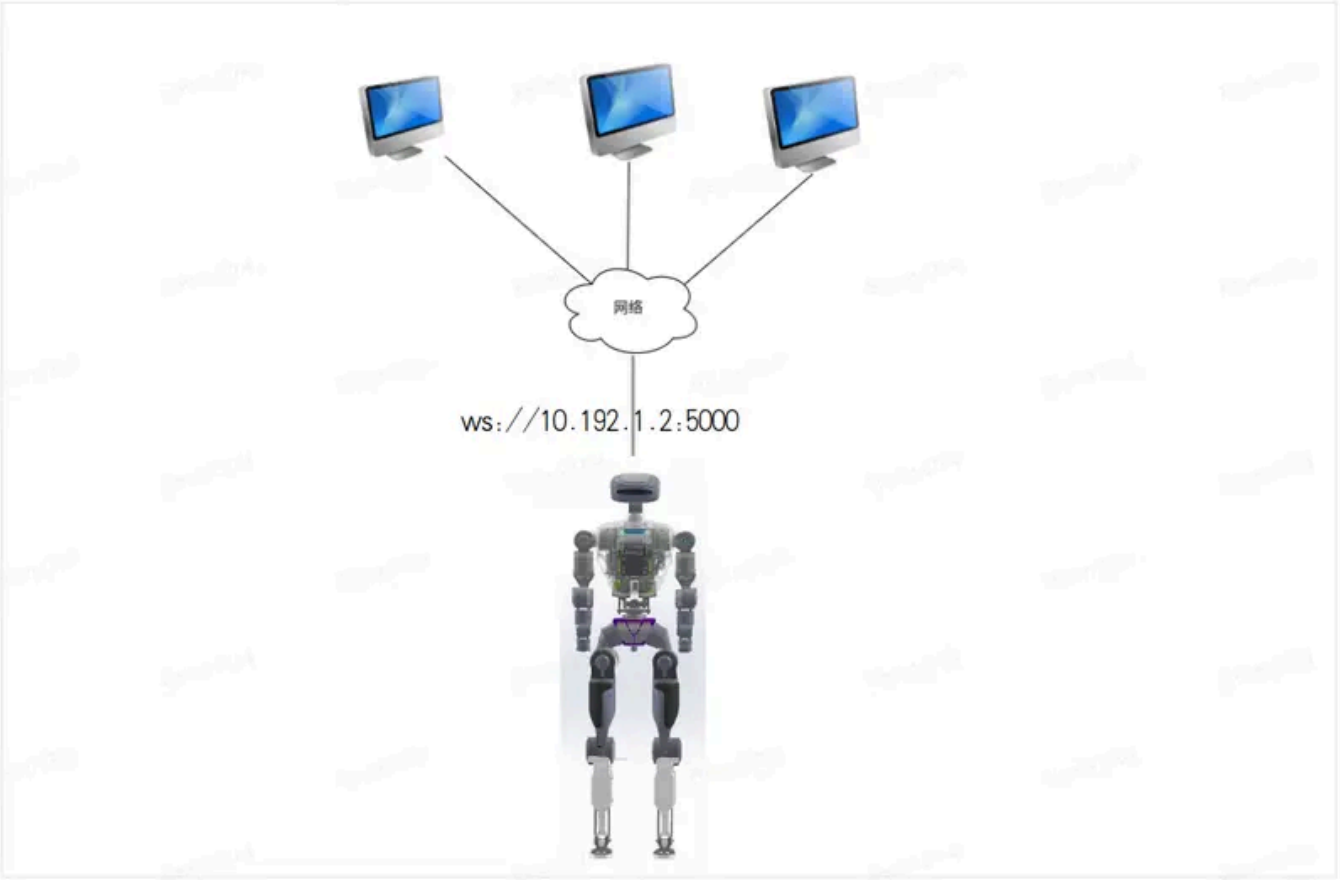
下图呈现了开发者的电脑与机器人本体的系统组成及交互关系。开发电脑部分涵盖运控算法节点和软件业务逻辑实现模块，通过 `上层应用协议接口` 和 `limxsdk-lowlevel` 的数据通讯控制机器人本体的运动。机器人本体由数据交换机、主控电脑及各类硬件组件构成，主控电脑负责协调各组件运行。

中文版	英文版
	

3 上层应用协议接口

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：



3.1 坐标说明

- 如无特别说明，双臂末端的位置、姿态，均基于机器人的 base 坐标系。
- SN 开头为 HU 的人型机器人 base 坐标系的定义。
- base 坐标系原点为 URDF 文件中定义的 base_link，坐标系标准为右手坐标系 / REP-103 坐标系。

3.2 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。

3.2.1 请求数据

请求数据格式包含字段	描述
`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。

请求数据格式包含字段	描述
`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。

请求数据代码示例：

▼

复制代码

```
{
  "accid": "HU_D02_001", # 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
}
```

3.2.2 响应数据

响应数据格式包含字段	描述
`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。

响应数据代码示例：

▼

json 复制代码

```
{
  "accid": "HU_D02_001",   # 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_xx
x”
  }
}
```

3.2.3 消息推送

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。

消息推送数据格式包含字段	描述
`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。

消息推送代码示例：

▼

json 复制代码

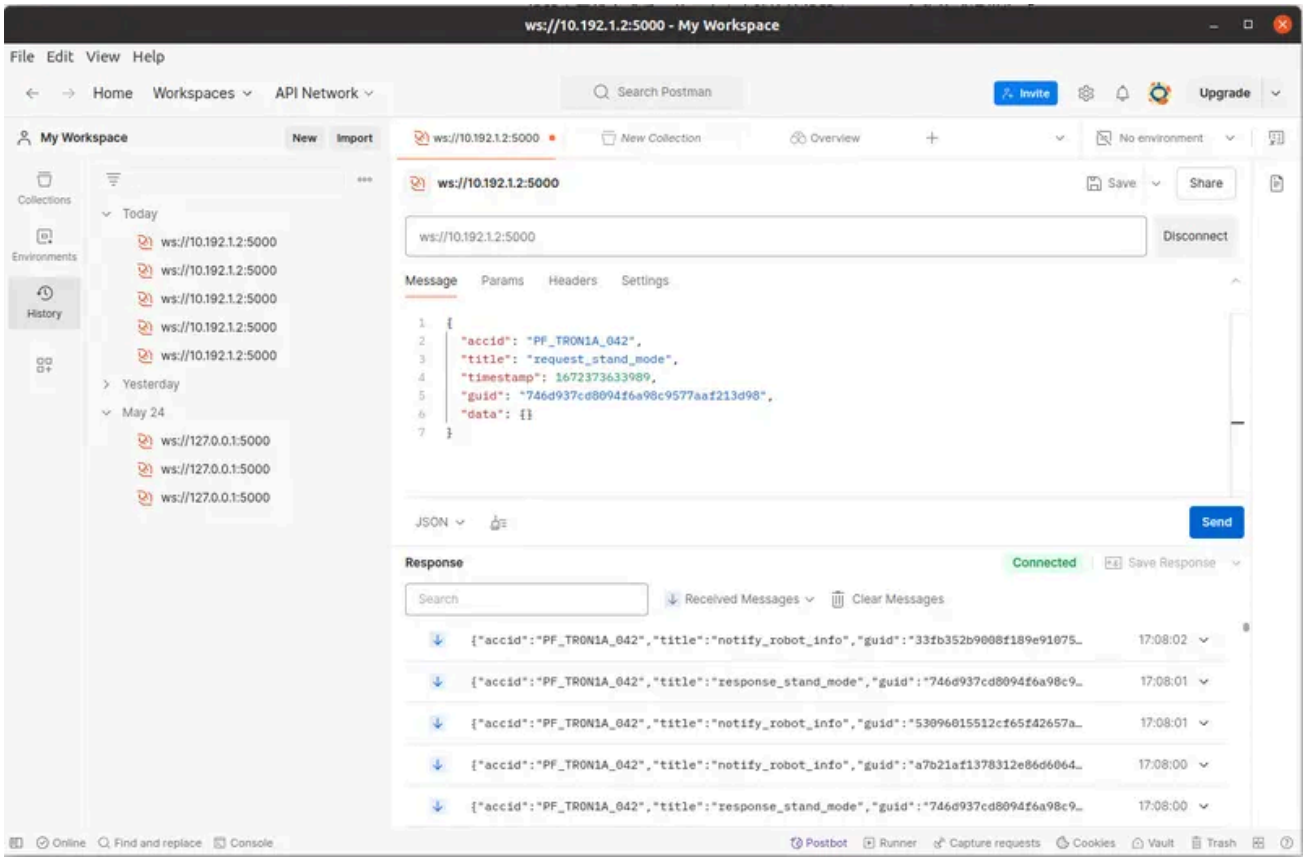
```
{
  "accid": "HU_D02_001",   # 机器人唯一序列号，标识机器人的唯一身份
  "title": "notify_xxx",  # 消息名称，以“notify_”为前缀
  "timestamp": 1672373633989, # 消息发出时间戳，单位为毫秒
  "guid": "746d937cd8094f6a98c9577aaf213d98", # 消息的guid值，唯一标识这条消息
  "data": { } # 存放消息数据内容
}
```

3.3 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 接口操作步骤：

1. 安装 postman，下载地址：https://www.postman.com/downloads/?utm_source=postman-home；
2. 打开 Postman，并创建一个 WebSocket 的请求；
3. 连接机器人无线网络 1. 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「HU_D02_xxx」
4. 输入 Wi-Fi 密码：`12345678`
5. 在请求的 URL 中输入 WebSocket 接口的地址，例如，“ws://10.192.1.2:5000”;
6. 在“Message”中，输入要发送的指令请求；
7. 单击“Send”按钮，发送请求指令；
8. 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。



3.4 基础功能协议接口

3.4.1 连接 Wi-Fi 热点

3.4.1.1 请求：request_connect_wifi

> 本协议用于向机器人路由器发起请求，指令路由器连接到指定 SSID 的 WiFi 热点并返回连接结果，支持机器人版本：2.1.3 及以上。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_connect_wifi",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "wifi_band": "0",  # WiFi频段：0=5GHz，1=2.4GHz
      "wifi_ssid": "Limx-Guests",  # 目标WiFi的SSID（WiFi名称），区分大小写，需与实际热点一致
      "wifi_password": "LimX2024",  # 目标WiFi的密码，WPA2-PSK加密方式的密码
      "router_admin_password": "12345678"  # 机器人路由器的管理员密码
  }
}
```

3.4.1.2 响应：response_connect_wifi

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_connect_wifi",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功
                           # fail_no_wifi_band: 没有指定频段
                           # fail_no_wifi_ssid: 没有指定ssid
                           # fail_no_wifi_password: 没有指定密码
                           # fail_no_router_admin_password: 没有指定密码
  }
}
```

3.4.1.3 消息推送：无

3.4.2 查询 Wi-Fi 连接状态

> 本协议用于客户端向机器人路由器发起 WiFi 连接状态查询请求，路由器接收请求后反馈当前已连接 WiFi 的核心状态信息，包括关联 SSID、信号强度及连接结果，支撑客户端实时感知设备网络连接状态。客户端发起 WiFi 连接状态查询，需携带机器人路由器管理员密码完成身份校验。

3.4.2.1 请求：request_wifi_connection_status

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_wifi_connection_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "router_admin_password": "12345678"  # 机器人路由器的管理员密码
  }
}
```

3.4.2.2 响应：response_wifi_connection_status

> 机器人路由器接收查询请求后，返回当前 WiFi 实际连接状态，供客户端解析展示或后续业务处理。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_wifi_connection_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "ssid": "Limx-Guests",
      "signal": -56,        # 单位dBm
      "result": "success"   # success: 成功
                            # fail_disconnected
  }
}
```

3.4.2.3 消息推送：无

3.4.3 进入准备状态

> 机器人缓慢摆出准备姿势。

3.4.3.1 请求：request_prepare

> 控制机器人进入“站姿”，可接受速度指令控制行走。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_prepare",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": { }
}
```

3.4.3.2 响应：response_prepare

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_prepare",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.3.3 消息推送：无

3.4.4 控制机器人行走

3.4.4.1 进入行走模式

机器人进入行走模式，可以接收速度指令。

3.4.4.1.1 请求：request_set_walk_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.4.1.2 响应：response_set_walk_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.4.1.3 消息推送：无

3.4.4.2 控制机器人行走

在移动操作模式下，通过此协议控制机器人行走。请注意，在全身操作模式下，此协议接口无效。

3.4.4.2.1 请求：request_set_walk_vel

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_walk_vel",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "x": 0.0,   #  前进后退速度比值，取值范围[-1, 1]
```

```
    "y": 0.0,   #  横向行走速度比值，取值范围[-1, 1]
    "yaw": 0.0  #  旋转角速度比值，取值范围[-1, 1]
  }
}
```

3.4.4.2.2 响应：response_set_walk_vel

指令执行失败时返回此消息，成功执行则无返回。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_walk_vel",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "fail_motor"  # fail_imu: IMU 错误, fail_motor: 电机错误
  }
}
```

3.4.4.2.3 消息推送：无

3.4.5 进入阻尼模式

机器人所有电机停止主动运动，摆动时有明显阻尼感。

3.4.5.1 请求：request_damping

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_damping",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.5.2 响应：response_damping

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_damping",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: 电机错误
  }
}
```

3.4.5.3 消息推送：无

3.4.6 进入零力矩模式

机器人所有电机停止主动运动，摆动时没有阻尼感。

3.4.6.1 请求：request_zero_torque

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_zero_torque",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.6.2 响应：response_zero_torque

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_zero_torque",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: 电机错误
  }
}
```

3.4.6.3 消息推送：无

3.4.7 进入坐下指令

3.4.7.1 请求：request_from_stand_to_sit

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_from_stand_to_sit",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.7.2 响应：response_from_stand_to_sit

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_from_stand_to_sit",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: 电机错误
  }
}
```

3.4.7.3 消息推送：无

3.4.8 进入站立指令

> **提示：**
>
> 接口功能：启动机器人使用，机器人开机后调用该接口进入站立状态。
> 参数 mode：[lying：机器人躺着 hanging：机器人吊着 sit: 机器人坐着]
> 返回：已站起后返回

3.4.8.1 请求：request_standup

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_standup",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "mode": "lying" // "lying"/"sitting"：机器人当前躺着/坐着  or
                      // "hanging"：机器人当前状态吊着
                      // 若没有"mode" 字段 默认机器人状态是"sitting"坐着
  }
}
```

3.4.8.2 响应：response_standup

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_standup",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: 电机错误
                          # fail_invalid_cmd：参数错误
                          # fail_invalid_mode：机器人状态错误
                          # fail_timeout：执行超时错误
  }
}
```

3.4.8.3 消息推送：无

3.4.9 进入躺着指令

3.4.9.1 请求：request_lie_down

> Walk 状态下可以调用该接口

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_lie_down",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.9.2 响应：response_lie_down

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_lie_down",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: 电机错误
  }
}
```

3.4.9.3 消息推送：无

3.4.10 机器人校零指令

3.4.10.1 请求：request_calibrate

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_calibrate",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.10.2 响应：response_calibrate

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_calibrate",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: 电机错误
  }
}
```

3.4.10.3 消息推送：notify_calibrate

校零完成后，推送此消息。

▼

python 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_calibrate",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"
  }
}
```

3.4.11 机器人舞蹈

3.4.11.1 切换机器人到舞蹈模式

3.4.11.1.1 请求：request_enter_dance_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_enter_dance_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 0：退出舞蹈模式
    # 1：进入舞蹈模式
    "mode": 0
```

```
  }
}
```

3.4.11.1.2 响应：response_enter_dance_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_enter_dance_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.11.1.3 消息推送：无

3.4.11.2 获取舞蹈列表

3.4.11.2.1 请求：request_get_dance_list

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_dance_list",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.11.2.2 响应：response_get_dance_list

▼

json 复制代码

```
{
    "accid": "HU_D04_01_001",
    "title": "response_get_dance_list",
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "timestamp": 1672373633989,
```

```
    "data": {
        "result": "success",
        "code": 0,
        "dances": [
            {
                "id": "DAN-14",
                "index": 0,
                "name": "\u70ed\u70c8",
                "english_name": "One and Only Dance",
                "rc_mapping": "one_and_only_dance"
            },
            {
                "id": "DAN-08",
                "index": 1,
                "name": "\u4f4e\u4fd7\u5c0f\u8bf4",
                "english_name": "Pulp Fiction Dance",
                "rc_mapping": "pulp_fiction_dance"
            }
        ]
    }
}
```

3.4.11.3 机器人跳舞

3.4.11.3.1 请求：request_dance

> **提示：**
>
> 1. 执行前置条件：当前处于动作库模式
> 2. 适用于主控 V2.1.21 及以上版本

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_dance",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "name": "one_and_only_dance"  # rc_mapping 字段的舞蹈名称
  }
}
```

3.4.11.3.2 响应：response_dance

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_dance",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.11.3.3 消息推送：notify_dance

跳完舞蹈或执行过程中失败推送此消息。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_dance",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.12 机器人原地踏步

3.4.12.1 开启原地踏步

3.4.12.1.1 请求：request_start_walktoggle

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_start_walktoggle",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
  "data": {}
}
```

3.4.12.1.2 响应：response_start_walktoggle

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_start_walktoggle",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.12.1.3 消息推送：无

3.4.12.2 停止原地踏步

3.4.12.2.1 请求：request_stop_walktoggle

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_stop_walktoggle",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.12.2.2 响应：response_stop_walktoggle

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_stop_walktoggle",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.13 机器人动作库

3.4.13.1 动作打断

3.4.13.1.1 请求：request_interrupt_action_joystick

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_interrupt_action_joystick",
  "timestamp": 1779355330784,
  "guid": "32cef03a-5563-4b21-9bbb-3e65a8c9ae9e",
  "data": {}
}
```

3.4.13.1.2 响应：response_interrupt_action_joystick

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_interrupt_action_joystick",
  "guid": "32cef03a-5563-4b21-9bbb-3e65a8c9ae9e",
  "timestamp": 1779355330784,
  "data": {
    "result": "success"
  }
}
```

3.4.13.2 获取动作库状态

> 接口说明：
>
> 1. 机器人进入动作库之后，处于动作库/原子执行/舞蹈中将显示："action_library_mode": "action_library"
> 2. 当机器人在执行原子动作中或者舞蹈中将显示： "action_library_state": "running"

3.4.13.2.1 请求：request_get_action_library_status

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_action_library_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.13.2.2 响应：response_get_action_library_status

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "get_action_library_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "action_library_mode": "action_library" //"remote_control"
      "action_library_state": "running"       //"idle"
      "result": "success" # fail_motor
  }
}
```

3.4.13.3 执行动作库

> 接口说明：
>
> 1. 不在 Menu 会自动进入 Menu，动作执行完会保持 Menu；
> 2. 需配合 `request_get_action_library_status` 的 `action_library_state` 使用。

3.4.13.3.1 请求：request_action_sync_stay_menu

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
```

```
  "title": "request_action_sync_stay_menu",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "name": "one_and_only_dance"
  }
}
```

3.4.13.3.2 响应：response_action_sync_stay_menu

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_action_sync_stay_menu",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor
  }
}
```

3.4.13.4 执行动作库（同步接口）

> 接口说明：
>
> 1. 当机器人处于不可执行动作库时，在 100ms 内返回失败响应；
> 2. 当机器人处于可执行动作库时（walk/motion library），在执行动作库之后返回响应。
> 3. 同步接口，会自动回到Walk，结束状态通过判断是否再Walk结束。

3.4.13.4.1 请求：request_action_sync

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_action_sync",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "name": "one_and_only_dance,this_way_please"
                                    # name:多个舞蹈/多个动作/舞蹈与动作混合,用“，”隔开
```

```
  }
}
```

3.4.13.4.2 响应：response_action_sync

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_action_sync",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.13.5 切换机器人到动作库模式

3.4.13.5.1 请求：request_set_motion_engine

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_motion_engine",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 0：退出动作库模式
    # 1：进入动作库模式
    "mode": 0
  }
}
```

3.4.13.5.2 响应：response_set_motion_engine

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_motion_engine",
```

```
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.13.5.3 消息推送：无

3.4.13.6 获取动作库列表

3.4.13.6.1 请求：request_get_atomic_motion_list

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_atomic_motion_list",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.13.6.2 响应：response_get_atomic_motion_list

▼

json 复制代码

```
{
    "accid": "HU_D04_01_001",
    "title": "response_get_atomic_motion_list",
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "timestamp": 287883835,
    "data": {
        "result": "success",
        "motion_list": [
            {
                "motion_index": 0,
                "motion_name_cn": "静止站立",
                "motion_name_en": "stand"
            },
            {
                "motion_index": 1,
                "motion_name_cn": "指引请这边走",
                "motion_name_en": "this_way_please"
```

```
            }
            ......
        ],
        "count": 2
    }
}
```

3.4.13.7 执行机器人动作库动作

设置机器人为动作库模式后，可以执行机器人动作库中的动作。

3.4.13.7.1 请求：request_execute_atomic_motion

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_execute_atomic_motion",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 动作名称
    "motion_name": "wave_greet_bye"
  }
}
```

3.4.13.7.2 响应：response_execute_atomic_motion

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_execute_atomic_motion",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.13.7.3 消息推送：notify_execute_atomic_motion

动作执行完成或执行过程中失败推送此消息。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_execute_atomic_motion",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.14 机器人移动操作

3.4.14.1 移动操作模式切换

在移动操作模式下您还可以控制机器人行走，但不能控制机器人的身高及腰部运动。



3.4.14.1.1 请求：request_set_ub_manip_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_ub_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "mode": 0  # 0: 准备进入模式 1: 操作模式，开始跟踪末端位置 2: 准备退出模式
  }
}
```

3.4.14.1.2 响应：response_set_ub_manip_mode

▼

json 复制代码

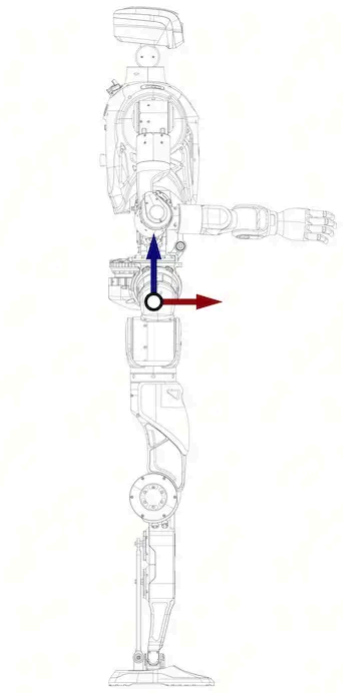
```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_ub_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

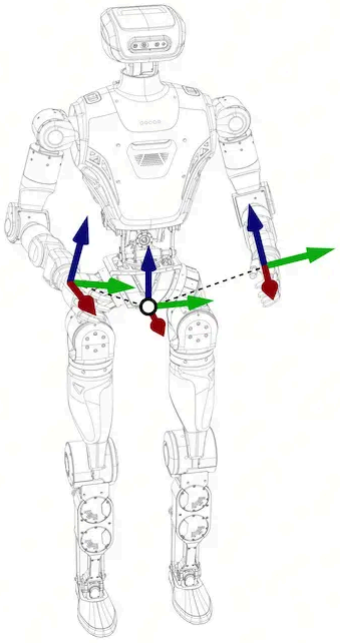
3.4.14.1.3 消息推送：无

3.4.14.2 移动操作控制

需要通过协议接口 `request_set_ub_manip_mode`，功能才生效。

- **参考坐标系示意图：**
  - 位置 —— base_link的原点（髋部下方正中间）
  - 坐标轴定义：红色为x方向：机器人前进方向；绿色为y方向：正方向向左；蓝色为z方向：竖直朝上





| | |

3.4.14.2.1 请求：request_set_ub_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
```

```
  "title": "request_set_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 参考坐标系定义：
      # 原点：base_link坐标系
      # 方向：x方向对齐机器人正前方，y方向朝向机器人左侧，z方向竖直向上
      #
      # 参数定义：
      # 头相对于参考坐标系的姿态，四元数[x,y,z,w]
      "head_quat": [0.0, 0.0, 0.0, 1.0],

      # 左手相对于参考坐标系的位置，单位为米
      "left_hand_pos": [0.0, 0.0, 0.0],

      # 左手相对于参考坐标系的姿态，四元数[x,y,z,w]
      "left_hand_quat": [0.0, 0.0, 0.0, 1.0],

      # 右手相对于参考坐标系的位置，单位为米
      "right_hand_pos": [0.0, 0.0, 0.0],

      # 右手相对于参考坐标系的姿态，四元数[x,y,z,w]
      "right_hand_quat": [0.0, 0.0, 0.0, 1.0]
  }
}
```

3.4.14.2.2 响应：response_set_ub_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误, fail_invalid_cmd: 非法指令
  }
}
```

3.4.14.2.3 消息推送：无

3.4.14.3 获取移动操作位姿信息

3.4.14.3.1 请求：request_get_ub_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.4.14.3.2 响应：response_get_ub_manip_ee_pose

▶

json 复制代码

3.4.15 机器人原地操作

3.4.15.1 进入原地操作模式

原地操作模式下，您可控制机器人的身体运动，但无法操控其行走功能。

3.4.15.1.1 请求：request_set_wb_manip_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_wb_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "mode": 0  # 0: 准备进入模式 1: 操作模式，开始跟踪末端位置 2: 准备退出模式
  }
}
```

3.4.15.1.2 响应：response_set_wb_manip_mode

▼

json 复制代码

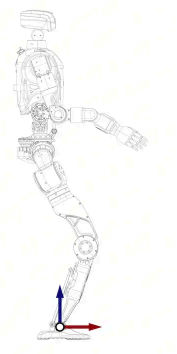
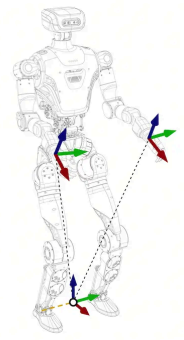
```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_wb_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.15.1.3 消息推送：无

3.4.15.2 原地操作控制

需要通过协议接口 `request_set_wb_manip_mode`，mode 为 1 进入原地操作模式下，功能才生效。

- **参考坐标系示意图：**
  - 位置 —— left_ankle_roll_link 与 right_ankle_roll_link 原点连线的中点
  - 姿态 —— yaw方向为 left_ankle_roll_link 与 right_ankle_roll_link 转向的中间值
  - 坐标轴定义：红色为x方向：由机器人双脚朝向决定；绿色为y方向：可根据右手坐标系规则确定；蓝色为z方向：竖直朝上。

3.4.15.2.1 请求：request_set_wb_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
```

```
  "title": "request_set_wb_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 参考坐标系定义：
      # 原点：为左脚以及右脚的中心位置在地面上的投影（z=0）
      # 方向：x方向对齐机器人正前方，y方向朝向机器人左侧，z方向竖直向上
      #
      # 参数定义：
      # 左手相对于参考坐标系的位置，单位为米
      "left_hand_pos": [0.0, 0.3, 0.8],

      # 左手相对于参考坐标系的姿态，四元数[x,y,z,w]
      "left_hand_quat": [0.0, 0.0, 0.0, 1.0],

      # 右手相对于参考坐标系的位置，单位为米
      "right_hand_pos": [0.0, -0.3, 0.8],

      # 右手相对于参考坐标系的姿态，四元数[x,y,z,w]
      "right_hand_quat": [0.0, 0.0, 0.0, 1.0]
  }
}
```

3.4.15.2.2 响应：response_set_wb_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_wb_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误, fail_invalid_cmd: 非法指令
  }
}
```

3.4.15.2.3 消息推送：无

3.4.15.3 获取原地操作位姿信息

3.4.15.3.1 请求：request_get_wb_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_wb_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.4.15.3.2 响应：response_get_wb_manip_ee_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_wb_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 参考坐标系定义：
    # 原点：为左脚以及右脚的中心位置在地面上的投影（z=0）
    # 方向：x方向对齐机器人正前方，y方向朝向机器人左侧，z方向竖直向上
    #
    # 参数定义：
    # 左手相对于参考坐标系的位置，单位为米
    "left_hand_pos": [0.0, 0.0, 0.0],

    # 左手相对于参考坐标系的姿态，四元数[x,y,z,w]
    "left_hand_quat": [0.0, 0.0, 0.0, 1.0],

    # 右手相对于参考坐标系的位置，单位为米
    "right_hand_pos": [0.0, 0.0, 0.0],

    # 右手相对于参考坐标系的姿态，四元数[x,y,z,w]
    "right_hand_quat": [0.0, 0.0, 0.0, 1.0],
    "result": "success"  # success: 成功, fail_motor: 电机错误, fail_invalid_cmd: 非法指令
  }
}
```

3.4.15.3.3 消息推送：无

3.4.16 双臂协同 Move 控制

3.4.16.1 切换 Move 控制模式

3.4.16.1.1 请求：request_set_move_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_move_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 0: 退出Move控制
      # 1: 移动Move模式
      # 2: 原地Move模式
      "mode": 0
  }
}
```

3.4.16.1.2 响应：response_set_move_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_move_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.16.1.3 消息推送：无

3.4.16.2 MoveJ 控制指令

3.4.16.2.1 请求：request_moveJ

▶

python 复制代码

3.4.16.2.2 响应：response_moveJ

▼

python 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_moveJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.16.2.3 消息推送：notify_moveJ

执行完成或失败，主动推送此消息。

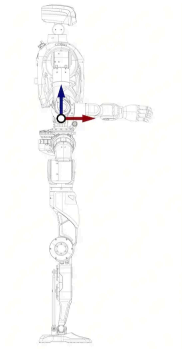
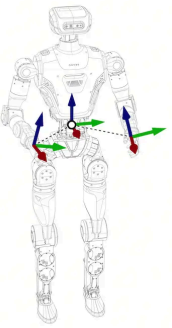
▼

python 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_moveJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 执行完成， fail_motor: 电机错误， fail_invalid_speed: 非法速度值
  }
}
```

3.4.16.3 MoveP 控制指令

- **参考坐标系示意图：**
  - 位置 —— waist_pitch_link 的原点
  - 坐标轴定义：红色为x方向：机器人前进方向；绿色为y方向：正方向向左；蓝色为z方向：竖直朝上

3.4.16.3.1 请求：request_moveP

▼

python 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_moveP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 如您同时给出以下数据，则会控制双臂运动
      "left_position": [0.0, 0.0, 0.0], # 表示左臂末端要移动到的目标位置，单位为米，顺序为 x, y,
z
      "left_quat": [0.0, 0.0, 0.0, 1.0], # 用四元数（x, y, z, w）表示左臂末端的目标姿态
      "right_position": [0.0, 0.0, 0.0], # 表示右臂末端要移动到的目标位置，单位为米，顺序为 x, y,
z
      "right_quat": [0.0, 0.0, 0.0, 1.0], # 用四元数（x, y, z, w）表示右臂末端的目标姿态

      # 仅在原地MoveP可控，如您同时给出以下数据，则会控制躯干姿态
      "torso_height": 0,  # 调整身高比例值，取值范围[-1, 1]
      "torso_pitch": 0,   # Pitch方向运动比例值，取值范围[-1, 1]
      "torso_roll": 0,    # Roll方向运动比例值，取值范围[-1, 1]
      "torso_yaw": 0,     # Yaw方向运动比例值，取值范围[-1, 1]

      # 如您同时给出以下数据，则会控制头运动
      "head_pitch": 0.0,  # pitch 关节的目标位置，单位为弧度
      "head_yaw": 0.0,    # yaw 关节的目标位置，单位为弧度

      "speed": 0.2  # 运动速度，取值范围为 0 到 0.5 弧度 / 秒，控制双臂运动的快慢
  }
}
```

3.4.16.3.2 响应：response_moveP

▼

python 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_moveP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.16.3.3 消息推送：notify_moveP

执行完成或失败，主动推送此消息。

▼

python 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_moveP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 执行完成， fail_motor: 电机错误， fail_invalid_speed: 非法速度值
  }
}
```

3.4.16.4 获取双臂末端位姿

3.4.16.4.1 请求：request_get_move_pose

通过此接口获取机器人双臂的末端位姿信息。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_move_pose",
  "timestamp": 1672373633989,
```

```
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.16.4.2 响应：response_get_move_pose

接收到请求后，返回双臂当前位姿的相关信息。

▼

复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_move_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "timestamp": 1672373633989, # 表示数据时戳，单位为毫秒
      "left_position": [0.0, 0.0, 0.0], # 表示左臂末端的位置，单位为米，顺序为 x, y, z
      "left_quat": [0.0, 0.0, 0.0, 1.0], # 表示左臂末端的姿态，以四元数表示，顺序为 x, y, z, w
      "right_position": [0.0, 0.0, 0.0], # 表示右臂末端的位置，单位为米，顺序为 x, y, z
      "right_quat": [0.0, 0.0, 0.0, 1.0], # 表示右臂末端的姿态，以四元数表示，顺序为 x, y, z, w
      "result": "success"  # fail_not_data
  }
}
```

3.4.17 双臂协同 Servo 控制

3.4.17.1 切换 Servo 控制模式

3.4.17.1.1 请求：request_set_servo_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_servo_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 0: 退出Servo控制
      # 1: 移动Servo模式
      # 2: 原地Servo模式
      "mode": 0
```

```
  }
}
```

3.4.17.1.2 响应：response_set_servo_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_servo_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.17.1.3 消息推送：无

3.4.17.2 ServoJ 控制指令

3.4.17.2.1 请求：request_servoJ

- 推荐在实时系统中按控制频率 >= 500Hz 要求来控制机械臂运动，以保证控制效果和稳定性。

▶

json 复制代码

3.4.17.2.2 响应：无

3.4.17.2.3 消息推送：notify_servoJ

当 ServoJ 控制操作执行失败时，服务器会主动推送此消息，告知客户端失败原因。

▼

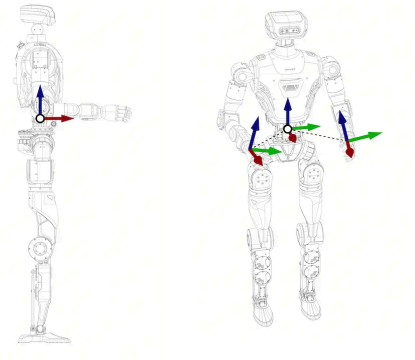
json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_servoJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "fail_invalid_cmd"  # fail_invalid_cmd: 非法指令， fail_motor: 电机错误
```

```
  }
}
```

3.4.17.3 ServoP 控制指令

- **参考坐标系示意图：**
  - 位置 —— waist_pitch_link 的原点
  - 坐标轴定义：红色为x方向：机器人前进方向；绿色为y方向：正方向向左；蓝色为z方向：竖直朝上。



3.4.17.3.1 请求：request_servoP

- 推荐在实时系统中按控制频率 >= 500Hz 要求来控制机械臂运动，以保证控制效果和稳定性。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_servoP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 如您同时给出以下数据，则会控制双臂运动
      "left_position": [0.0, 0.0, 0.0], # 表示左臂末端的目标位置，单位为米，顺序为 x，y, z
      "left_quat": [0.0, 0.0, 0.0, 1.0], # 以四元数（x, y, z, w）形式表示左臂末端的目标姿态
      "right_position": [0.0, 0.0, 0.0], # 表示右臂末端的目标位置，单位为米，顺序为 x，y, z
      "right_quat": [0.0, 0.0, 0.0, 1.0] # 以四元数（x, y, z, w）形式表示右臂末端的目标姿态

      # 仅在原地ServoP可控，如您同时给出以下数据，则会控制躯干姿态
      "torso_height": 0,  # 调整身高比例值，取值范围[-1, 1]
      "torso_pitch": 0,   # Pitch方向运动比例值，取值范围[-1, 1]
      "torso_roll": 0,    # Roll方向运动比例值，取值范围[-1, 1]
      "torso_yaw": 0,     # Yaw方向运动比例值，取值范围[-1, 1]

      # 如您同时给出以下数据，则会控制头运动
      "head_yaw": 0.0,    # yaw 关节的目标位置，单位为弧度
      "head_pitch": 0.0  # picth 关节的目标位置，单位为弧度
```

```
  }
}
```

3.4.17.3.2 响应：无

3.4.17.3.3 消息推送：notify_servoP

当 ServoP 控制操作执行失败时，服务器会主动推送此消息，告知客户端失败原因。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_servoP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "fail_invalid_cmd"  # fail_invalid_cmd: 非法指令， fail_motor: 电机错误
  }
}
```

3.4.17.4 获取双臂末端位姿

3.4.17.4.1 请求：request_get_servo_pose

通过此接口获取机器人双臂的末端位姿信息。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_servo_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.17.4.2 响应：response_get_servo_pose

接收到请求后，返回双臂当前位姿的相关信息。

▼

复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_servo_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "timestamp": 1672373633989, # 表示数据时戳，单位为毫秒
      "left_position": [0.0, 0.0, 0.0], # 表示左臂末端的位置，单位为米，顺序为 x, y, z
      "left_quat": [0.0, 0.0, 0.0, 1.0], # 表示左臂末端的姿态，以四元数表示，顺序为 x, y, z, w
      "right_position": [0.0, 0.0, 0.0], # 表示右臂末端的位置，单位为米，顺序为 x, y, z
      "right_quat": [0.0, 0.0, 0.0, 1.0], # 表示右臂末端的姿态，以四元数表示，顺序为 x, y, z, w
      "result": "success"  # fail_not_data
  }
}
```

3.4.17.4.3 消息推送：无

3.4.18 机器人全身操作

3.4.18.1 切换全身操作模式

3.4.18.1.1 请求：request_set_wbt_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_wbt_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 0: 退出全身操作模式
      # 1: 进入WBT四点式控制模式
      # 2: 进入WBT关节控制模式
      "mode": 0
  }
}
```

3.4.18.1.2 响应：response_set_wbt_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_wbt_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.4.18.1.3 消息推送：无

3.4.18.2 WBT 关节控制指令

3.4.18.2.1 请求：request_set_wbt_joint_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_wbt_joint_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "joint_q": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    "base_pos":         [0.0, 0.0, 0.95],
    "base_quat":        [0.0, 0.0, 0.0, 1.0]
  }
}
```

3.4.18.2.2 响应：response_set_wbt_joint_pose

注意：有错误才反馈

3.4.18.2.3 消息推送：无

3.4.18.3 WBT 四点式控制指令

3.4.18.3.1 请求：request_set_wbt_keybody_pose

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_wbt_keybody_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "base_pos":         [0.0, 0.0, 0.95],
    "base_quat":        [0.0, 0.0, 0.0, 1.0],

    "head_pos":         [0.0,  0.00, 0.50],
    "head_quat":        [0.0, 0.0, 0.0, 1.0],

    "left_wrist_pos":   [0.2,  0.25, 0.10],
    "left_wrist_quat":  [0.0, 0.0, 0.0, 1.0],

    "right_wrist_pos":  [0.2, -0.25, 0.10],
    "right_wrist_quat": [0.0, 0.0, 0.0, 1.0],

    "base_velocity": {
      "vx":       0.30,
      "vy":       0.00,
      "yaw_rate": 0.00
    }
  }
}
```

3.4.18.3.2 响应：response_set_wbt_keybody_pose

注意：有错误才反馈

3.4.18.3.3 消息推送：无

3.4.19 机器人关节状态

3.4.19.1 请求：request_get_joint_state

此请求用于获取机器人的各个关节状态。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_joint_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
  "data": {}
}
```

3.4.19.2 响应：response_get_joint_state

接收到请求后，返回机器人各个关节当前状态。

▼

复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_joint_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "names": [], # 各个关节的名称
      "q": [],     # 各个关节的位置
      "dq": [],    # 各个关节的速度
      "tau": [],   # 各个关节的扭矩
      "result": "success"  # fail_not_data
  }
}
```

3.4.19.3 消息推送：无

3.4.20 获取 IMU 数据

3.4.20.1 请求：request_get_imu_data

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_imu_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.4.20.2 响应：response_get_imu_data

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_imu_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success", # fail_no_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]
  }
}
```

3.4.20.3 消息推送：无

3.5 灵巧手及夹爪协议接口

3.5.1 逐际二指夹爪

3.5.1.1 夹爪控制指令

3.5.1.1.1 请求：request_set_limx_2fclaw_cmd

此协议控制夹爪的抓取动作。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_limx_2fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 如您同时给出以下数据，则会控制左夹爪运动
      "left_opening": 50,  # 开口度，0-100，无量纲（0对应最小闭合，100对应张开到最大）
      "left_speed": 50,    # 夹爪速度，0~100 无量纲（数值越大速度越快）
      "left_force": 50,   #力，夹爪夹持力，0~100 无单位（数值越大力越大）

      # 如您同时给出以下数据，则会控制右夹爪运动
      "right_opening": 50,  # 开口度，0-100，无量纲（0对应最小闭合，100对应张开到最大）
      "right_speed": 50,    # 夹爪速度，0~100 无量纲（数值越大速度越快）
      "right_force": 50,   #力，夹爪夹持力，0~100 无单位（数值越大力越大）
  }
}
```

3.5.1.1.2 响应：response_set_limx_2fclaw_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_limx_2fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.5.1.1.3 消息推送：无

3.5.1.2 获取夹爪状态信息

3.5.1.2.1 请求：request_get_limx_2fclaw_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_limx_2fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.5.1.2.2 响应：response_get_limx_2fclaw_state

返回夹爪状态信息

3.5.1.2.3 消息推送：无

3.5.2 逐际三指夹爪

3.5.2.1 夹爪控制指令

3.5.2.1.1 请求：request_set_limx_3fclaw_cmd

此协议控制夹爪的抓取动作。

3.5.2.1.2 响应：response_set_limx_3fclaw_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_limx_3fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.5.2.1.3 消息推送：无

3.5.2.2 获取夹爪状态信息

3.5.2.2.1 请求：request_get_limx_3fclaw_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_limx_3fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.5.2.2.2 响应：response_get_limx_3fclaw_state

返回夹爪状态信息

3.5.2.2.3 消息推送：无

3.5.3 因时二指夹爪

3.5.3.1 夹爪控制指令

3.5.3.1.1 请求：request_set_claw_cmd

此协议控制夹爪的抓取动作。

3.5.3.1.2 响应：response_set_claw_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_claw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.5.3.1.3 消息推送：无

3.5.3.2 获取夹爪状态信息

3.5.3.2.1 请求：request_get_claw_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_claw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.5.3.2.2 响应：response_get_claw_state

返回夹爪状态信息

3.5.3.2.3 消息推送：无

3.5.4 强脑 1 代灵巧手

3.5.4.1 灵巧手控制指令

3.5.4.1.1 请求：request_set_brainco_hand_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_brainco_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # 如您给定以下数据，则会控制左手运动
      "left_thumb": 50,        # 左手大拇指弯曲角度，0-100，无量纲
      "left_thumb_aux": 50,    # 左手大拇指内收角度，0-100，无量纲
      "left_index": 50,        # 左手食指弯曲角度，0-100，无量纲
      "left_middle": 50,       # 左手中指弯曲角度，0-100，无量纲
      "left_ring": 50,         # 左手无名指弯曲角度，0-100，无量纲
      "left_pinky": 50,        # 左手小指弯曲角度，0-100，无量纲
      "left_mode": 3,          # 力量等级 1：小 2：中 3：大。默认值为：2

      # 如您给定以下数据，则会控制右手运动
      "right_thumb": 50,        # 右手大拇指弯曲角度，0-100，无量纲
      "right_thumb_aux": 50,    # 右手大拇指内收角度，0-100，无量纲
      "right_index": 50,        # 右手食指弯曲角度，0-100，无量纲
      "right_middle": 50,       # 右手中指弯曲角度，0-100，无量纲
      "right_ring": 50,         # 右手无名指弯曲角度，0-100，无量纲
      "right_pinky": 50,        # 右手小指弯曲角度，0-100，无量纲
      "right_mode": 3           # 力量等级 1：小 2：中 3：大。默认值为：2
  }
}
```

3.5.4.1.2 响应：response_set_brainco_hand_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_brainco_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.5.4.1.3 消息推送：无

3.5.4.2 获取灵巧手状态

3.5.4.2.1 请求：request_get_brainco_hand_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_brainco_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.5.4.2.2 响应：response_get_brainco_hand_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_brainco_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "timestamp": 1672373633989, # 表示数据时戳，单位为毫秒
      "left_thumb": 50,       # 左手大拇指弯曲角度，0-100，无量纲
      "left_thumb_aux": 50,   # 左手大拇指内收角度，0-100，无量纲
      "left_index": 50,       # 左手食指弯曲角度，0-100，无量纲
      "left_middle": 50,      # 左手中指弯曲角度，0-100，无量纲
      "left_ring": 50,        # 左手无名指弯曲角度，0-100，无量纲
      "left_pinky": 50,       # 左手小指弯曲角度，0-100，无量纲

      "right_thumb": 50,       # 右手大拇指弯曲角度，0-100，无量纲
      "right_thumb_aux": 50,   # 右手大拇指内收角度，0-100，无量纲
      "right_index": 50,       # 右手食指弯曲角度，0-100，无量纲
      "right_middle": 50,      # 右手中指弯曲角度，0-100，无量纲
      "right_ring": 50,        # 右手无名指弯曲角度，0-100，无量纲
      "right_pinky": 50        # 右手小指弯曲角度，0-100，无量纲
      "result": "success"  # success: 成功, fail_motor: 电机错误
  }
}
```

3.5.4.2.3 消息推送：无

3.5.5 强脑 2 代灵巧手

3.5.5.1 灵巧手控制指令

3.5.5.1.1 请求：request_set_brainco2_hand_cmd

▶

json 复制代码

3.5.5.1.2 响应：response_set_brainco2_hand_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_brainco2_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误, fail_invalid_cmd: 非法指令
  }
}
```

3.5.5.1.3 消息推送：无

3.5.5.2 获取灵巧手状态

3.5.5.2.1 请求：request_get_brainco2_hand_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_brainco2_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.5.5.2.2 响应：response_get_brainco2_hand_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_brainco2_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "timestamp": 1672373633989,
      "left_mode": 1,
      "left_pos": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
      "left_vel": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
      "left_current": [500, 500, 500, 500, 500, 500],
      "left_time": [1000, 1000, 1000, 1000, 1000, 1000],
      "right_mode": 1,
      "right_pos": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
      "right_vel": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
      "right_current": [500, 500, 500, 500, 500, 500],
      "right_time": [1000, 1000, 1000, 1000, 1000, 1000]
  }
}
```

3.5.5.2.3 消息推送：无

3.5.6 强脑 2 代触觉灵巧手

3.5.6.1 灵巧手控制指令

3.5.6.1.1 请求：request_set_brainco2_touch_hand_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_brainco2_touch_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "left_mode": 1,
    "right_mode": 2,

    "left_pos":  [0.5, 0.8, 0.7, 0.7, 0.7, 0.7],
```

```
    "left_time": [500, 500, 500, 500, 500, 500],

    "right_pos": [0.5, 0.8, 0.7, 0.7, 0.7, 0.7],
    "right_vel": [1.0, 1.0, 1.0, 1.0, 1.0, 1.0],

    "left_tactile_switch":       [0, 0, 0, 0, 0],
    "left_channel_reset":        [0, 0, 0, 0, 0],
    "left_calibration_trigger":  [0, 0, 0, 0, 0],

    "right_tactile_switch":      [0, 0, 0, 0, 0],
    "right_channel_reset":       [0, 0, 0, 0, 0],
    "right_calibration_trigger": [0, 0, 0, 0, 0]
  }
}
```

字段说明：

字段	类型	必填	说明
left_mode / right_mode	int	至少一个	控制模式：0=退出控制, 1=位置时间模式, 2=位置速度模式, 3=力控模式

各 mode 所需字段：

mode	所需字段	数组长度	说明
0	无	-	退出控制
1	left_pos + left_time	6	pos 单位 rad，time 单位 ms (1-2000)
2	left_pos + left_vel	6	pos 单位 rad，vel 单位 rad/s
3	left_current	6	单位 mA，范围 ±1000

右手字段对应为 `right_pos`、`right_time`、`right_vel`、`right_current`。

6 个手指索引对应：

索引	手指	pos 范围 (rad)	vel 范围 (rad/s)
0	拇指尖 (thumb)	0 ~ 1.0297	0 ~ 2.5367
1	拇指根 (thumb aux)	0 ~ 1.5707	0 ~ 2.6180
2	食指 (index)	0 ~ 1.4137	0 ~ 2.2689

索引	手指	pos 范围 (rad)	vel 范围 (rad/s)
3	中指 (middle)	0 ~ 1.4137	0 ~ 2.2689
4	无名指 (ring)	0 ~ 1.4137	0 ~ 2.2689
5	小指 (pinky)	0 ~ 1.4137	0 ~ 2.2689

可选，预留功能。5 个触觉传感器索引：0=拇指, 1=食指, 2=中指, 3=无名指, 4=小指。

字段	类型	数组长度	说明
left_tactile_switch / right_tactile_switch	int[5]	5	触觉开关，0=关闭，1=开启
left_channel_reset / right_channel_reset	int[5]	5	通道复位，执行时手指不可受力，0=关闭，1=开启
left_calibration_trigger / right_calibration_trigger	int[5]	5	参数校准，执行约 10s 内数据不可用，0=关闭，1=开启

错误码说明：

result	说明
fail_no_ctrl_mode	未提供 left_mode 和 right_mode
fail_no_left_ctrl_mode	提供了左手数据但未提供 left_mode
fail_no_right_ctrl_mode	提供了右手数据但未提供 right_mode
fail_invalid_left_mode / fail_invalid_right_mode	mode 值不在 0-3 范围
fail_invalid_left_pos_and_time	mode=1 时 pos 或 time 数组长度不为 6
fail_no_left_pos_or_time	mode=1 时缺少 pos 或 time
fail_invalid_left_pos_and_vel	mode=2 时 pos 或 vel 数组长度不为 6
fail_no_left_pos_or_vel	mode=2 时缺少 pos 或 vel
fail_invalid_left_current	mode=3 时 current 数组长度不为 6
fail_no_left_current	mode=3 时缺少 current

3.5.6.1.2 响应：response_set_brainco2_touch_hand_cmd

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_brainco2_touch_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: 成功, fail_motor: 电机错误, fail_invalid_cmd: 非法指令
  }
}
```

3.5.6.1.3 消息推送：无

3.5.6.2 获取强脑 2 代触觉灵巧手状态

3.5.6.2.1 请求：request_get_brainco2_touch_hand_state

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_brainco2_touch_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.5.6.2.2 响应：response_get_brainco2_touch_hand_state

▶

json 复制代码

响应字段说明：

字段	类型	说明
left_mode / right_mode	int	当前控制模式
left_names / right_names	string[6]	手指名称

字段	类型	说明
left_pos / right_pos	float[6]	实际位置 (rad)
left_vel / right_vel	float[6]	实际速度 (rad/s，负值表示张开)
left_current / right_current	float[6]	实际电流 (mA)
left_time / right_time	float[6]	控制时间 (ms)
left_tactile_names / right_tactile_names	string[5]	传感器名称
left_normal_force / right_normal_force	float[5]	法向力 (N)
left_tangential_force / right_tangential_force	float[5]	切向力 (N)
left_direction_angle / right_direction_angle	float[5]	切向力方向角 (rad, 0~6.26)
left_approximate_value / right_approximate_value	uint32[5]	接近值
left_tactile_state / right_tactile_state	uint16[5]	触觉状态

3.5.6.2.3 消息推送：无

3.6 音频设备接口

3.6.1 概述

机器人提供音频设备相关的 WebSocket API，支持麦克风采集、喇叭播放、唤醒词检测和音量控制等功能。

音频服务支持以下核心能力：

- 音频采集：实时音频流推送和一次性录音
- 音频播放：支持 PCM 原始数据播放和音频文件播放（本地文件或远程 URL，支持 PCM/WAV/MP3）
- 唤醒词检测：支持语音唤醒词检测，检测到唤醒词时主动推送事件
- 音量控制：全局播放音量调节

3.6.2 音频流推送控制

音频流推送开关。开启后，系统通过 `notify_audio_capture` 将 PCM 音频数据持续推送至客户端；关闭后停止推送。

3.6.2.1 请求：request_audio_capture

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_capture",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "streaming": 1  # 1: 开启推送, 0: 关闭推送
  }
}
```

3.6.2.2 响应：response_audio_capture

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_capture",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功
  }
}
```

3.6.2.3 消息推送：notify_audio_capture

开启音频流推送后，系统会持续下发采集到的 PCM 音频数据片段。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_audio_capture",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "sample_rate": 16000,  # 采样率，单位 Hz
      "channels": 1,         # 声道数
      "samples": [...]       # PCM 数据，int16 数组
  }
}
```

3.6.2.4 代码示例：audio_capture_ws.py

audio_capture_ws.py # 默认录制 5 秒

audio_capture_ws.py -d 10 -o test.wav # 录制 10 秒，保存到 test.wav

audio_capture_ws.py --host 10.192.1.2 # 指定 IP

▶

python 复制代码

3.6.3 一次性录音

一次性录音功能。系统在指定时长内录制音频，录制完成后将完整 PCM 数据通过响应一次性返回。

3.6.3.1 请求：request_audio_capture_record

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_capture_record",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "duration": 5.0  # 录音时长，单位秒
  }
}
```

3.6.3.2 响应：response_audio_capture_record

录制完成后返回完整的 PCM 音频数据。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_capture_record",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success",    # success: 成功
      "sample_rate": 16000,   # 采样率，单位 Hz
      "channels": 1,          # 声道数
      "samples": [...]        # PCM 数据，int16 数组
```

```
  }
}
```

3.6.3.3 消息推送：无

3.6.3.4 代码示例：audio_capture_record_ws.py

audio_capture_record_ws.py # 录制 5 秒

audio_capture_record_ws.py -d 10 # 录制 10 秒

audio_capture_record_ws.py -d 3 -o test.wav # 录制 3 秒，保存到 test.wav

▶

python 复制代码

3.6.4 播放控制

喇叭播放全局开关。开启后初始化播放设备并开始消费播放队列；关闭时停止播放并清空队列。

系统启动后，喇叭播放默认为关闭状态，需要通过此接口手动开启。

3.6.4.1 请求：request_audio_playback_control

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_playback_control",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "enable": 1  # 1: 开启播放, 0: 关闭播放
  }
}
```

3.6.4.2 响应：response_audio_playback_control

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_playback_control",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
  "data": {
      "result": "success"  # success: 成功
  }
}
```

3.6.4.3 消息推送：无

3.6.5 播放 PCM 数据

向播放队列发送 PCM 原始数据片段。支持分块连续发送，系统按顺序依次播放。

注意：调用前需先通过 `request_audio_playback_control` 开启播放功能。

3.6.5.1 请求：request_audio_play_data

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_play_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "sample_rate": 16000,  # 采样率，单位 Hz
      "channels": 1,         # 声道数
      "samples": [...]       # PCM 数据，int16 数组
  }
}
```

3.6.5.2 响应：response_audio_play_data

正常成功时不返回响应；仅失败时返回响应，客户端不应等待成功响应。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_play_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "fail_no_samples"  # fail_no_samples: 缺少 PCM 数据
  }
}
```

3.6.5.3 消息推送：无

3.6.5.4 代码示例：audio_playback_ws.py

audio_playback_ws.py test.wav # 播放 WAV 文件

audio_playback_ws.py --tone 1000 -d 3 # 播放 1000Hz 正弦波 3 秒

audio_playback_ws.py --sweep -d 5 # 播放 200~8000Hz 扫频 5 秒

audio_playback_ws.py --noise -d 3 # 播放白噪声 3 秒

audio_playback_ws.py --tone 1000 -v 50 # 50% 音量播放

audio_playback_ws.py --tone 1000 -a 0.1 # 10% 振幅播放 (喇叭保护)

▶

python 复制代码

3.6.6 播放 PCM 数据并生成手势

3.6.6.1 Gesture 模式控制

控制机器人进入或退出 gesture 模式。只有先调用 `request_audio_gesture_control` 且 `enable=1` 成功后，后续 `request_audio_play_with_gesture` 才会在播放 PCM 的同时驱动上肢动作。调用 `enable=0` 用于退出 gesture 模式。未进入 gesture 模式时，`request_audio_play_with_gesture` 只播放音频，不生成动作。

3.6.6.1.1 请求：request_audio_gesture_control

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_gesture_control",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "enable": 1  # 1: 进入 gesture 模式；0: 退出 gesture 模式
  }
}
```

3.6.6.1.2 响应：response_audio_gesture_control

▼

json 复制代码

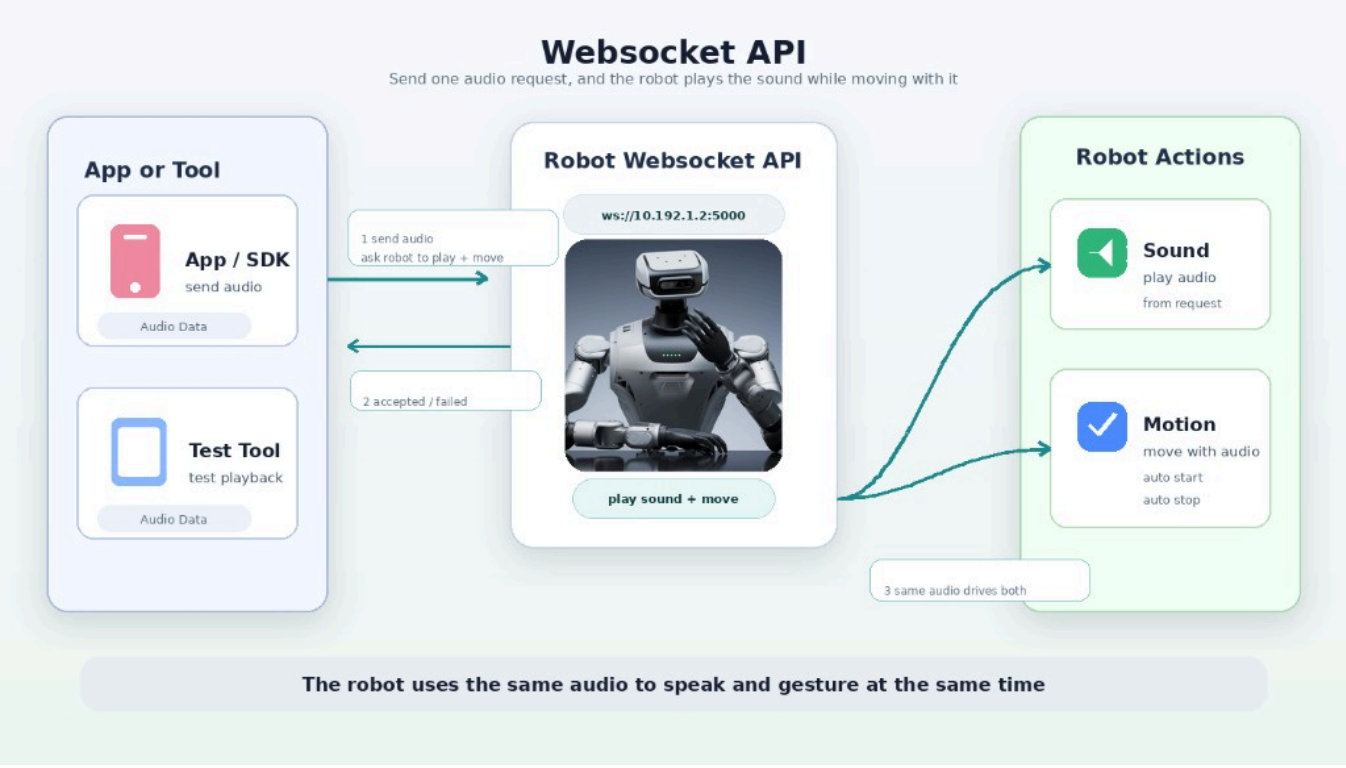
```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_gesture_control",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success",  # success: 成功
                            # fail_no_enable: 缺少 enable
                            # fail_gesture_control_service_not_ready: gesture 服务未就绪
                            # fail_gesture_control_call: 调用失败
                            # fail: gesture 服务拒绝执行
      "message": "gesture mode entered"  # enable=1: gesture mode entered
                                         # enable=0: gesture mode exited
  }
}
```

3.6.6.1.3 消息推送：无

3.6.6.2 发送 PCM 数据并生成手势

向机器人发送 PCM 原始数据片段。系统使用同一份音频数据进行声音播放，并同步驱动机器人生成对应手势。支持分块连续发送，系统按顺序依次处理。

注意：调用前需确保音频播放服务和手势服务已启动。



3.6.6.2.1 请求：request_audio_play_with_gesture

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_play_with_gesture",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "sample_rate": 16000,  # 采样率，单位 Hz
      "channels": 1,         # 声道数
      "samples": [...],      # PCM 数据，int16 数组
      "enable_head": 1       # 可选；1/缺省: 生成头部动作，0: 不生成头部动作
  }
}
```

3.6.6.2.2 响应：response_audio_play_with_gesture

正常成功时不返回响应；仅失败时返回响应，客户端不应等待成功响应。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_play_with_gesture",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "fail_no_samples"  # fail_no_samples: 缺少 PCM 数据
  }
}
```

3.6.6.2.3 消息推送：无

3.6.6.2.4 代码示例：audio_playback_ws.py

audio_playback_ws.py --gesture-enter # 进入手势模式

audio_playback_ws.py test.wav --gesture # 已进入模式时，带手势播放

audio_playback_ws.py --gesture-exit # 退出手势模式

▶

python 复制代码

3.6.7 播放音频文件

请求播放音频文件。支持机器人本地文件路径或远程 HTTP URL，支持 WAV、MP3、PCM 格式。

3.6.7.1 请求：request_audio_play_file

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_play_file",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "file_path": "/path/to/audio.wav"  # 本地文件路径或 HTTP URL
  }
}
```

3.6.7.2 响应：response_audio_play_file

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_play_file",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功
  }
}
```

3.6.7.3 消息推送：无

3.6.7.4 代码示例：audio_file_player_ws.py

audio_file_player_ws.py /path/to/audio.wav

audio_file_player_ws.py https://example.com/audio.wav

▶

python 复制代码

3.6.8 唤醒词检测控制

唤醒词检测的启停开关。开启后，系统会持续检测音频流中的唤醒词，检测到匹配时向客户端推送 `notify_audio_wakeup` 事件。

3.6.8.1 请求：request_audio_wakeup_control

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_wakeup_control",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "enable": 1  # 1: 开启唤醒检测, 0: 关闭唤醒检测
  }
}
```

3.6.8.2 响应：response_audio_wakeup_control

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_wakeup_control",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功
  }
}
```

3.6.8.3 消息推送：notify_audio_wakeup

当唤醒检测开启时，检测到用户唤醒词后触发此事件。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_audio_wakeup",
  "timestamp": 1672373633989,
```

```
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "word": "hai4 o1 li3",  # 触发的唤醒词
      "doa": 180              # 声源方向角度 (DOA)，单位度
  }
}
```

3.6.9 设置唤醒词

动态设置唤醒词。设置成功后立即生效并持久化存储，重启后自动恢复。

此功能需要机器人配备语音唤醒模块。

3.6.9.1 请求：request_audio_set_wakeup_word

字段	说明	是否必填
word	拼音加声调（谐音）	必填

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_set_wakeup_word",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "word": "ni3 hao3 zhu2 ji4"  # 使用拼音加声调
  }
}
```

3.6.9.2 响应：response_audio_set_wakeup_word

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_set_wakeup_word",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功
```

```
  }
}
```

3.6.9.3 消息推送：无

3.6.9.4 代码示例：audio_wakeup_ws.py

audio_wakeup_ws.py set "ni3 hao3 zhu2 ji4" # 简单唤醒词，带声调拼音

audio_wakeup_ws.py get # 获取当前唤醒词

audio_wakeup_ws.py enable # 开启唤醒检测

audio_wakeup_ws.py disable # 关闭唤醒检测

audio_wakeup_ws.py listen # 监听唤醒事件（Ctrl+C 退出）

audio_wakeup_ws.py listen -d 30 # 监听 30 秒

audio_wakeup_ws.py --host 10.192.1.4 listen

▶

python 复制代码

3.6.10 查询唤醒词

查询当前设定的唤醒词。

3.6.10.1 请求：request_audio_get_wakeup_word

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_get_wakeup_word",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.6.10.2 响应：response_audio_get_wakeup_word

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_get_wakeup_word",
```

```
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success",
      "word": "ni3 hao3 zhu2 ji4",
      "backend": "aispeech"
  }
}
```

3.6.10.3 消息推送：无

3.6.11 设置音量

设置播放的全局音量。

3.6.11.1 请求：request_audio_set_volume

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_set_volume",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "volume": 50  # 音量值，取值范围 [0, 100]，0 为静音，100 为最大音量
  }
}
```

3.6.11.2 响应：response_audio_set_volume

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_set_volume",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success: 成功
  }
}
```

3.6.11.3 消息推送：无

3.6.12 查询音量

查询当前播放的全局音量。

3.6.12.1 请求：request_audio_get_volume

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_get_volume",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

3.6.12.2 响应：response_audio_get_volume

▼

json 复制代码

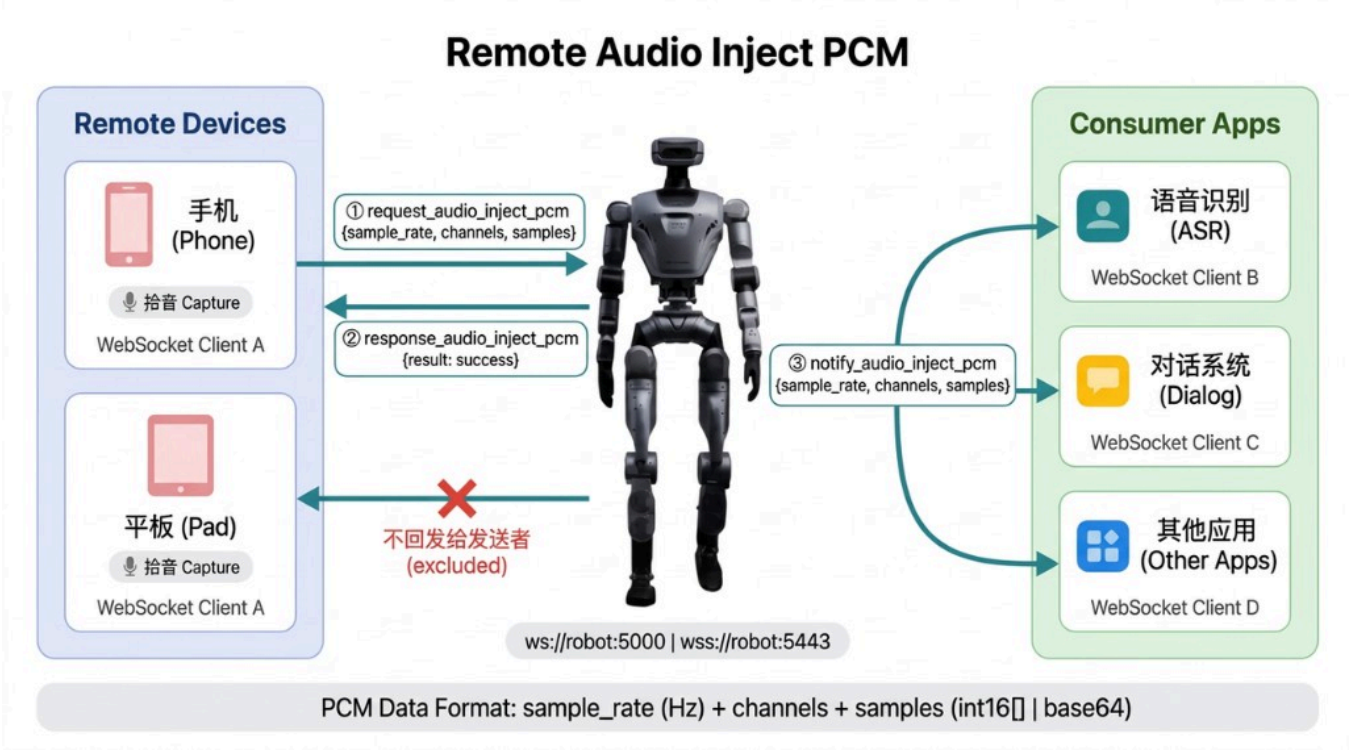
```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_get_volume",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success",
      "volume": 50,
      "message": "volume=50"
  }
}
```

3.6.12.3 消息推送：无

3.6.13 远程音频注入

外部设备（如手机、Pad）将拾取的 PCM 音频数据注入到机器人系统中，机器人服务会将该数据转发给除发送者以外的其他已连接 WebSocket 客户端。

典型场景：手机端拾音后，将音频数据发送给机器人，机器人上运行的其他应用（如语音识别、对话系统）通过 WebSocket 接收并处理该音频流，从而代替本地麦克风拾音。



3.6.13.1 请求：request_audio_inject_pcm

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_audio_inject_pcm",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "sample_rate": 16000,    # 采样率，单位 Hz
      "channels": 1,           # 声道数
      "samples": "<数据格式，由收发双方自行约定，如：PCM int16 或 base64>"
  }
}
```

3.6.13.2 响应：response_audio_inject_pcm

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_audio_inject_pcm",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
  "data": {
      "result": "success"     # success: 成功
  }
}
```

错误码说明：

result	说明
success	成功
fail_no_sample_rate	缺少 sample_rate 字段
fail_no_channels	缺少 channels 字段
fail_no_samples	缺少 samples 字段

3.6.13.3 消息推送：notify_audio_inject_pcm

请求成功后，会将音频数据推送给除发送者以外的所有已连接 WebSocket 客户端。

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_audio_inject_pcm",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "sample_rate": 16000,   # 采样率，单位 Hz
      "channels": 1,          # 声道数
      "samples": "<与请求中完全一致>"
  }
}
```

3.6.13.4 代码示例：audio_inject_pcm_ws.py

audio_inject_pcm_ws.py --host 10.192.1.2 # 注入 1 秒 440Hz 正弦波（int16 数组，默认）

python3 audio_inject_pcm_ws.py --host 10.192.1.2 --format base64 # 使用 base64 编码注入

audio_inject_pcm_ws.py --host 10.192.1.2 --wav /path/to/audio.wav # 从 WAV 文件注入

audio_inject_pcm_ws.py --host 10.192.1.2 --listen -o received.wav # 监听 notify_audio_inject_pcm 并保存为 WAV

▶

python 复制代码

3.7 全局消息协议接口

3.7.1 机器人状态信息

此协议将定时上报机器人状态信息。

上报状态信息	描述
`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：

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": []
  }
}
```

3.7.1.1 电池数据

▶

json 复制代码

字段	含义
bmsconn	电池连接状态 ：【OFF：未连接，ON：已连接】
bat_chg	电池充电器状态 ：【OFF：未连接，ON：已连接】

字段	含义
bat_off	电池预关机状态 ：【OFF：1s 后断电，ON：正常】
bat_prt	电池故障码：【 0：正常， 非 0：异常】
bat_vol	电池实时电压 单位：mV
bat_cur	电池实时电流 单位：mA
battery	电池电量百分比 0~100
bat_temp0	电池温度 0~100 单位：x10℃
bat_temp2	电池温度 0~100 单位：x10℃
bat_temp4	电池温度 0~100 单位：x10℃

3.7.1.2 系统信息

▶

json 复制代码

字段	含义
version	主控版本
ecm_version	主站版本
pms_version	分电板版本
motor_version	电机版本
sn	机器人序列号
robot_status	机器人当前状态
ability_running	机器人当前运行的控制器

3.7.1.3 电机状态信息

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
```

```
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": [
          {
                "level": 1,
                "name": "ethercatCommunicationExp",
                "message": "WARN",
                "hardware_id": "ethercat",
                "values": [
                    {
                        "key": "ethercatCommunicationExp",
                        "value": "motor 17 MOTOR_LOST triggered HALF_STAND"
                    },
                    {
                        "key": "ethercatResetNormal",
                        "value": "ok!"
                    }
                ]
          }
    ]
  }
}
```

字段	含义
level	异常等级[0:ok 1:warn 2:error]
name	异常类型
message	等级字符串
hardware_id	硬件 id
values	该硬件的所有异常集合

3.7.2 遥控器数据

此协议将上报机器人遥控器数据。

上报数据信息	描述
`accid`	机器人序列号
`title`	notify_joy_data

上报数据信息	描述
`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：

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_joy_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "axes": [],      # 遥感数据
    "buttons": []    # 按键数据
  }
}
```

3.8 协议接口调用示例

3.8.1 Python 示例

- **环境准备：** 以 Ubuntu 20.04 系统为例，安装下面依赖

▼

bash 复制代码

```
sudo apt install python3-dev python3-pip
sudo pip install websocket-client==1.8.0
```

- **运行脚本**

▼

bash 复制代码

```
python humanoid.py
```

- **humanoid.py 实现**

- ACCID：替换为真实的软件 SN
- ROBOT_IP: 一般情况，仿真为 127.0.0.1，真机为 10.192.1.2

▶

python 复制代码

3.8.2 Linux C++ 示例

- **环境准备：** 以 Ubuntu 20.04 系统为例，安装 websocketpp、nlohmann/json 和 boost 依赖：

▼

bash 复制代码

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

- **编译代码**

▼

bash 复制代码

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

- **运行程序**

▼

bash 复制代码

```
./humanoid
```

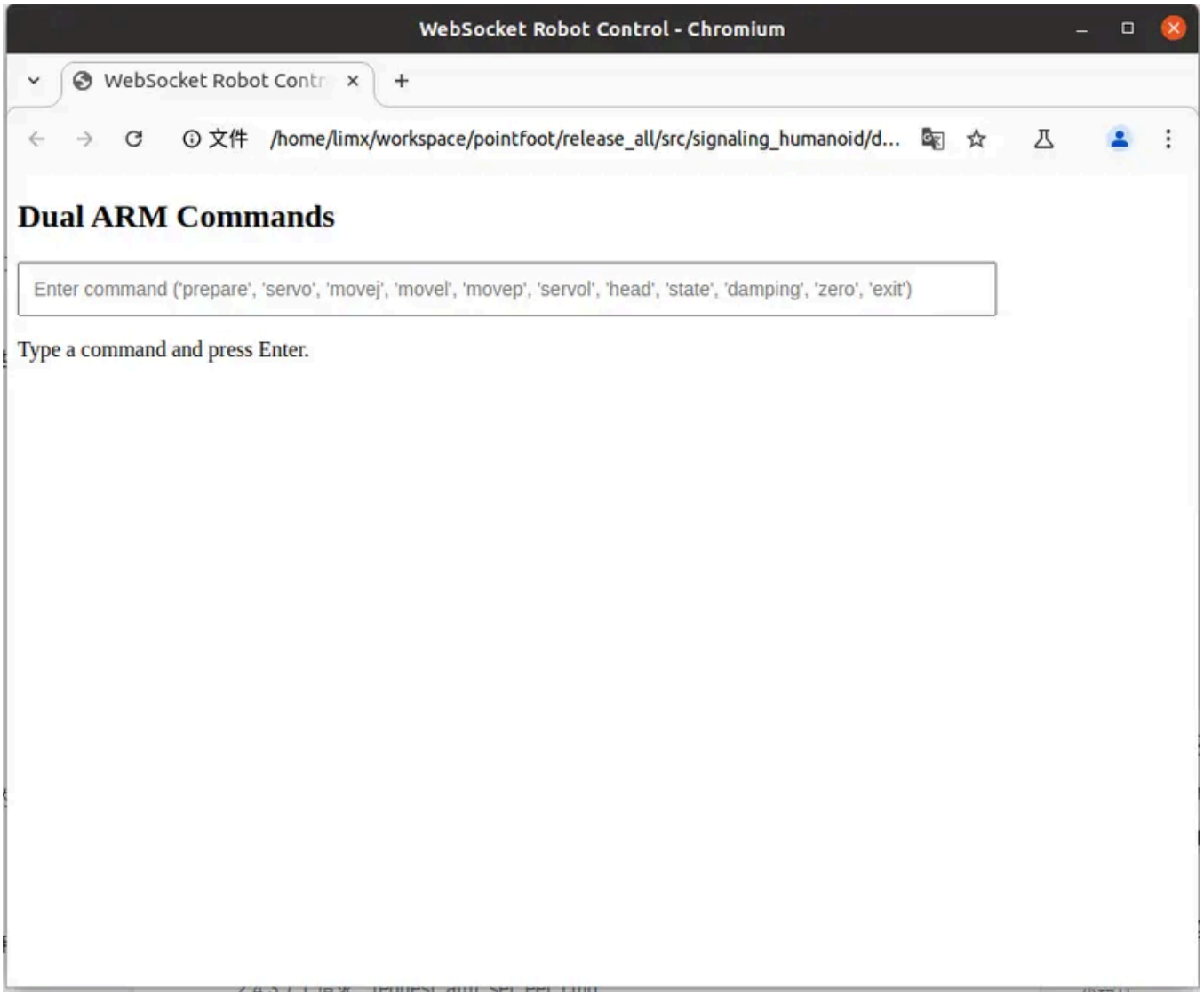
- **humanoid.cpp 实现**

▶

cpp 复制代码

3.8.3 JavaScript 示例

- **运行 humanoid.html：** 将 humanoid.html 文件保存到电脑中，然后在浏览器中打开 humanoid.html 运行。



- **humanoid.html 实现**

▶

html 复制代码

4 底层运动控制开发接口

跨平台底层运动控制开发接口库提供统一的 C++/Python API，兼容 ROS1、ROS2 及非 ROS 系统，实现运动控制算法的快速移植与部署。通过硬件抽象层和标准化通信协议，开发者可无缝切换仿真与真实硬件环境，显著降低多平台适配成本。

> **注意：**
>
> 1. 使用底层控制开发接口时，需通过按键 `R1+START` 切换**开发者模式**，此时高层开发接口会被禁用，机器人只响应上下电和校零遥控器指令。
> 2. 底层接口代码调用示例可参考 RL 部署训练。
> 3. 切换到开发者模式后，掉电模式会保留，退出开发者模式按键：`L2+○`

4.1 C++ 运动控制开发接口

4.1.1 getInstance 接口

函数名	getInstance
函数原型	static Humanoid* getInstance();
功能概述	获取 Humanoid 机器人类单例实例的指针
参数	无
返回值	Humanoid*，指向 Humanoid 实例的指针
备注	使用了单例模式，确保 Humanoid 类只有一个实例存在于程序中

代码示例：

▼

cpp 复制代码

```cpp
#include <thread>

// 包含 limxsdk::Humanoid 头文件，用于引入 Humanoid 类
#include "limxsdk/humanoid.h"

// 使用 limxsdk 命名空间，简化对 Humanoid 类的引用
using namespace limxsdk;

int main(int argc, char *argv[]){
  // 获取 Humanoid 类的单例实例
  Humanoid* robot = Humanoid::getInstance();

  // 无限循环以保持程序运行
  while (true)
  {
    // 休眠 1000 毫秒
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
  }

  return 0;
}
```

4.1.2 init 接口

函数名	init
函数原型	bool init(const std::string& robot_ip_address = "127.0.0.1");
功能概述	初始化运动控制算法程序的通信运行环境，通常在主函数中调用其它接口之前调用，完成初始化工作。
参数	robot_ip_address：机器人的 IP 地址。对于仿真，通常设置为 "127.0.0.1"，而对于真实机器人，可能设置为 "10.192.1.2"。
返回值	如果初始化成功，则返回 true；否则返回 false。
备注	无

代码示例：

```
cpp 复制代码
```

4.1.3 getMotorNumber 接口

函数名	getMotorNumber
函数原型	uint32_t getMotorNumber();
功能概述	获取机器人中的电机数量。
参数	无
返回值	返回一个无符号整数，表示机器人中的总电机数量。
备注	无

代码示例：

```
cpp 复制代码
```

4.1.4 subscribeImuData 接口

函数名	subscribeImuData
函数原型	void subscribeImuData(std::function<void(const ImuDataConstPtr&)> cb);

函数名	subscribeImuData
功能概述	订阅机器人的 IMU数据，并在接收到新的 IMU 数据时调用指定的回调函数。
参数	cb: 用于处理新 IMU 数据的回调函数。
返回值	无

备注：ImuData 数据结构原型如下：

▼

cpp 复制代码

```cpp
/**
 * @struct ImuData
 *
 * @brief 表示基于传感器反馈的机器人 IMU 数据的结构体。
 *
 * 此结构体封装了 IMU 数据，包括加速度计、陀螺仪和四元数。
 */
struct ImuData {
  uint64_t stamp; // 时间戳，以纳秒为单位，通常表示记录或生成此数据时的时间。
  float acc[3];   // 用于存储 IMU 加速度计数据，以跟踪沿三个轴（X、Y、Z）的线性加速度。
  float gyro[3];  // 用于存储 IMU 陀螺仪数据，以跟踪沿三个轴（X、Y、Z）的角速度或旋转速度。
  float quat[4];  // 用于存储 IMU 四元数数据，表示在三维空间中的方向（w、x、y、z）。
};

// 智能指针类型别名
typedef std::shared_ptr<ImuData> ImuDataPtr;
typedef std::shared_ptr<ImuData const> ImuDataConstPtr;
```

代码示例：

▶

cpp 复制代码

4.1.5 subscribeRobotState 接口

函数名	subscribeRobotState
函数原型	void subscribeRobotState(std::function<void(const RobotStateConstPtr&)> cb);
功能概述	订阅接收关于机器人状态的更新。

函数名	subscribeRobotState
参数	cb：回调函数，当接收到机器人状态更新时将被调用。回调函数参数指向 RobotState 对象的常量指针。
返回值	无

备注：RobotState 数据结构原型如下：

▼

cpp 复制代码

```cpp
/**
 * @struct RobotState
 *
 * @brief 代表基于传感器反馈的机器人状态的结构体。
 *
 * 此结构封装了各种数据点，可用于监控和控制机器人，包括 IMU 数据（加速度计、陀螺仪、四元数）、输出扭矩、当前角度和速度等。
 */
struct RobotState {
  // 默认构造函数
  RobotState() { }

  // 带参数的构造函数，用于初始化向量大小为 motor_num 的 tau、q、dq 向量，初始值均为 0.0
  RobotState(int motor_num)
  : tau(motor_num, 0.0)
  , q(motor_num, 0.0)
  , dq(motor_num, 0.0)
  , motor_names(motor_num, "") { }

  uint64_t stamp;              // 时间戳，通常表示记录或生成这些数据的时间，以纳秒为单位
  std::vector<float> tau;      // 用于存储当前估计的输出扭矩（以牛顿米为单位）的向量
  std::vector<float> q;        // 用于存储当前角度（以弧度为单位）的向量
  std::vector<float> dq;       // 用于存储当前速度（以弧度每秒为单位）的向量
  std::vector<std::string> motor_names; // 用于存储机器人各个关节名称的向量
};

// 智能指针类型别名
typedef std::shared_ptr<RobotState> RobotStatePtr;
typedef std::shared_ptr<RobotState const> RobotStateConstPtr;
```

代码示例：

▶

cpp 复制代码

4.1.6 publishRobotCmd 接口

函数名	publishRobotCmd
函数原型	bool publishRobotCmd(const RobotCmd& cmd);
功能概述	发布一个命令来控制机器人的动作。
参数	cmd：表示所需机器人命令的 RobotCmd 对象。
返回值	无

备注：RobotCmd 数据结构原型如下：

▶

cpp 复制代码

代码示例：

▶

cpp 复制代码

4.1.7 subscribeSensorJoy 接口

函数名	subscribeSensorJoy
函数原型	void subscribeSensorJoy(std::function<void(const SensorJoyConstPtr&)> cb);
功能概述	在真机部署中，该方法用于订阅来自机器人遥控器的数据。当机器人接收到遥控器的数据时，将会调用指定的回调函数，并传递包含遥控器数据的 SensorJoy 结构体常量指针给回调函数进行处理。
参数	cb: 表示回调函数，用于接收机器人遥控器的数据。回调函数的参数类型为 SensorJoyConstPtr，即指向 SensorJoy 结构体常量的共享指针。
返回值	无

备注：SensorJoy 数据结构原型如下：

▼

cpp 复制代码

```
/**
 * @struct SensorJoy
```

```
 *
 * @brief 机器人遥控器数据的结构体。
 *
 * 该结构体包含了与遥控器相关的时间戳信息，以及摇杆和按钮值。
 */
struct SensorJoy {
    uint64_t stamp;                     // 与传感器输入相关的时间戳，单位为纳秒。
    std::vector<float> axes;            // 表示操纵摇杆的值。
    std::vector<int32_t> buttons;       // 表示操纵按钮状态的值。
};

// SensorJoy 智能指针类型的定义
typedef std::shared_ptr<SensorJoy> SensorJoyPtr;
typedef std::shared_ptr<const SensorJoy> SensorJoyConstPtr;
```

代码示例：

▶

cpp 复制代码

4.1.8 subscribeDiagnosticValue 接口

函数名	subscribeDiagnosticValue
函数原型	void subscribeDiagnosticValue(std::function<void(const DiagnosticValueConstPtr&)> cb);
功能概述	在真机部署中，该方法用于订阅机器人的诊断值和状态信息。当机器人发出诊断值时，系统会调用指定的回调函数，并传递包含诊断值的 DiagnosticValue 结构体常量指针给回调函数进行处理。这可以帮助实时监控机器人的健康状态，并及时做出反应以处理可能的问题。
参数	cb: 用于接收机器人诊断值的回调函数，其参数类型为 DiagnosticValueConstPtr，即指向 DiagnosticValue 结构体常量的共享指针。DiagnosticValue 结构体包含了机器人诊断值的信息，包括时间戳、级别、名称、代码和消息字段。
返回值	无

备注：DiagnosticValue 数据结构原型如下：

▼

cpp 复制代码

```
/**
 * @struct DiagnosticValue
```

```
 *
 * @brief 结构体，表示诊断值。
 *
 * 此结构体包含有关诊断级别、名称、代码和消息的信息。
 */
struct DiagnosticValue {
  enum { OK = 0 };          // 正常状态的诊断级别
  enum { WARN = 1 };        // 警告状态的诊断级别
  enum { ERROR = 2 };       // 错误状态的诊断级别

  uint64_t stamp;           // 时间戳，单位为纳秒。
  int32_t level;            // 与诊断值相关联的级别。
  std::string name;         // 标识诊断值的名称。
  int32_t code;             // 与诊断值对应的代码。
  std::string message;      // 与诊断值相关的详细消息。
};

// DiagnosticValue 智能指针类型的定义
typedef std::shared_ptr<DiagnosticValue> DiagnosticValuePtr;
typedef std::shared_ptr<DiagnosticValue const> DiagnosticValueConstPtr;
```

代码示例：

▶

cpp 复制代码

4.1.9 publishJsonMessage 接口

函数名	publishJsonMessage
函数原型	void publishJsonMessage(const std::string &json_payload);
功能概述	向机器人发送 "上层应用协议接口" 的 JSON 格式消息。
参数	json_payload: 符合 "上层应用协议接口" 协议的 JSON 字符串。 示例：{"accid": "xxx", "title": "request_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
返回值	无
备注	**高阶开发模式下有效**

代码示例

▶

cpp 复制代码

4.1.10 subscribeJsonMessage 接口

函数名	subscribeJsonMessage
函数原型	void subscribeJsonMessage(std::function<void(const std::string &)> cb);
功能概述	注册一个回调函数，用于处理来自机器人的 "上层应用协议接口" 调用的响应和通知。 该回调函数会在以下两种场景中被调用： 1. 当机器人对之前发送的 JSON 命令（publishJsonMessage）返回响应时 2. 当机器人主动发起通知（未经请求的消息）时
参数	cb: 回调函数，原型为 void(const std::string &json_payload) 其中 json_payload 包含： - 命令响应：{"accid": "xxx", "title": "response_xxx", "timestamp": xxx, "guid": "xxx", "data": {}} - 通知：{"accid": "xxx", "title": "notify_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
返回值	无
备注	**高阶开发模式下有效**

代码示例：

▼

cpp 复制代码

```cpp
#include <thread>

#include "limxsdk/humanoid.h"

using namespace limxsdk;

int main(int argc, char *argv[]) {
  Humanoid* robot = Humanoid::getInstance();

  std::string robot_ip = "10.192.1.2";
  if (argc > 1) {
    robot_ip = argv[1];
  }

  if (!robot->init(robot_ip)) {
    exit(1);
  }

  robot->subscribeJsonMessage([&](const std::string & json_payload) {
    std::cout << json_payload << std::endl;
```

```
  });

  while (true) {
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
  }

  return 0;
}
```

4.1.11 参考例程

Github: https://github.com/limxdynamics/humanoid-rl-deploy-ros

4.2 Python 运动控制开发接口

提供与 C++ 相同功能的 Python 运动控制算法开发接口，**使得不熟悉 C++ 编程语言的开发者能够使用 Python 进行运动控制算法的开发。**

4.2.1 安装运动控制开发库

- **Linux x86_64 环境**

▼

bash 复制代码

```
pip install python3/amd64/limxsdk-*-py3-none-any.whl
```

- **Linux aarch64 环境**

▼

bash 复制代码

```
pip install python3/aarch64/limxsdk-*-py3-none-any.whl
```

- **Windows 环境**

▼

bash 复制代码

```
pip install python3/win/limxsdk-*-py3-none-any.whl
```

4.2.2 init 接口

函数名	init
函数原型	def **init**(self, robot_type: robot.RobotType)
功能概述	在初始化时，指定机器人的类型，并创建一个相应类型的本地机器人实例。
参数	robot_type：表示机器人类型的枚举值，点足为RobotType.Humanoid。
返回值	无
备注	无

代码示例：

▼

python 复制代码

```
import sys
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType

if __name__ == '__main__':
  # 创建一个类型为Humanoid的Robot实例
  robot = Robot(RobotType.Humanoid)
```

4.2.3 init 接口

函数名	init
函数原型	def init(self, robot_ip: str = "127.0.0.1")
功能概述	初始化运动控制算法程序的通信运行环境，通常在主函数中调用其它接口之前调用，完成初始化工作。
参数	robot_ip：机器人的 IP 地址。对于仿真，通常设置为 "127.0.0.1"，而对于真实机器人，可能设置为 "10.192.1.2"。
返回值	成功：返回True 失败：返回False
备注	无

代码示例：

▼

python 复制代码

```
import sys
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType

if __name__ == '__main__':
  # 创建一个类型为Humanoid的Robot实例
  robot = Robot(RobotType.Humanoid)

  robot_ip = "127.0.0.1"
  # 检查是否提供了命令行参数作为机器人IP
  if len(sys.argv) > 1:
    robot_ip = sys.argv[1]

  # 使用IP地址初始化机器人的通信运行环境
  if not robot.init(robot_ip):
    sys.exit()
```

4.2.4 getMotorNumber 接口

函数名	getMotorNumber
函数原型	def getMotorNumber(self)
功能概述	获取机器人中的电机数量。
参数	无
返回值	返回一个无符号整数，表示机器人中的总电机数量。
备注	通常情况下，点足机器人的电机数量为6个

代码示例：

▼

python 复制代码

```
import sys
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType

if __name__ == '__main__':
  # 创建一个 Humanoid 类型的 Robot 实例
  robot = Robot(RobotType.Humanoid)
```

```
robot_ip = "127.0.0.1"
# 检查是否提供了机器人 IP 的命令行参数
if len(sys.argv) > 1:
  robot_ip = sys.argv[1]

# 使用 robot_ip 初始化机器人
if not robot.init(robot_ip):
  sys.exit()

# 获取机器人中的电机数量
motor_number = robot.getMotorNumber()
```

4.2.5 subscribeImuData 接口

函数名	subscribeImuData
函数原型	def subscribeImuData(self, callback: Callable[[datatypes.ImuData], Any])
功能概述	订阅机器人的 IMU数据，并在接收到新的 IMU 数据时调用指定的回调函数。
参数	callback：用于处理新 IMU 数据的回调函数。
返回值	成功：返回True 失败：返回False

备注：datatypes.ImuData 数据结构原型如下：

▼

python 复制代码

```
import sys

class ImuData(object):
  __slots__ = ['stamp','acc','gyro','quat']
  def __init__(self):
    self.stamp = 0 # 时间戳，通常表示记录或生成这些数据的时间，以纳秒为单位
    self.acc = [0. for x in range(0, 3)]  # 用于存储 IMU（惯性测量单元）加速度计数据，用于跟踪三个轴上的线性加速度
    self.gyro = [0. for x in range(0, 3)] # 用于存储 IMU 陀螺仪数据，用于跟踪角速度或旋转速度
    self.quat = [0. for x in range(0, 4)] # 用于存储 IMU 四元数数据，表示在三维空间中的方向
```

代码示例：

▶

python 复制代码

4.2.6 subscribeRobotState 接口

函数名	subscribeRobotState
函数原型	def subscribeRobotState(self, callback: Callable[[datatypes.RobotState], Any])
功能概述	订阅接收关于机器人状态的更新。
参数	callback：回调函数，当接收到机器人状态更新时将被调用。回调函数参数指向datatypes.RobotState对象。 - datatypes.RobotState数据结构字段： - stamp：时间戳，通常表示记录或生成这些数据的时间。 - tau：用于存储当前估计的输出扭矩（以牛顿米为单位）的向量。 - q：用于存储当前角度（以弧度为单位）的向量。 - dq：用于存储当前速度（以弧度每秒为单位）的向量。 - motor_names：用于存储对应的关节名称。
返回值	成功：返回True 失败：返回False

备注：datatypes.RobotState 数据结构原型如下：

▼

python 复制代码

```
import sys

class RobotState(object):
  __slots__ = ['stamp','tau','q','dq']
  def __init__(self):
    self.stamp = 0 # 时间戳，通常表示记录或生成这些数据的时间，以纳秒为单位
    self.tau = []  # 用于存储当前估计的输出扭矩（以牛顿米为单位）的向量
    self.q = []    # 用于存储当前角度（以弧度为单位）的向量
    self.dq = []   # 用于存储当前速度（以弧度每秒为单位）的向量
    self.motor_names = []   # 用于存储对应的关节名称。
```

代码示例：

▶

python 复制代码

4.2.7 publishRobotCmd 接口

函数名	publishRobotCmd
函数原型	def publishRobotCmd (self, cmd: datatypes.RobotCmd)
功能概述	发布一个命令来控制机器人的动作。
参数	cmd：表示所需机器人命令的 datatypes.RobotCmd 对象，包含以下字段： - stamp：记录或生成数据时的时间戳，以纳秒为单位。 - q：存储所需的关节角度（以弧度为单位）的向量。 - dq：存储所需的关节速度（以弧度每秒为单位）的向量。 - tau：存储所需的输出扭矩（以牛顿米为单位）的向量。 - Kp：存储所需的位置刚度（以牛顿米每弧度为单位）的向量。 - Kd：存储所需的速度刚度（以牛顿米每弧度每秒为单位）的向量。 - motor_names：存储需要控制的关节名称。
参数取值	- q、dq、tau：在urdf中已定义范围，举例如下： - joint的字段中描述了每个关节的q、dq、tau取值范围： - lower和upper共同对应"q"，表示控制位置范围（单位为弧度）； - effort对应"tau"，表示扭矩峰值（单位为牛顿米）； - velocity对应"dq"，表示速度峰值（单位为弧度每秒）； - Kp、Kd可用值（用户自行开发控制器时需根据实际效果调整取值）：
返回值	成功：返回True 失败：返回False

备注：datatypes.RobotCmd 数据结构原型如下：

▼

python 复制代码

```
import sys

class RobotCmd(object):
  __slots__ = ['stamp','mode','q','dq','tau','Kp','Kd']
  def __init__(self):
    self.stamp = 0 # 时间戳（以纳秒为单位），表示记录或生成数据时的时间。
    self.mode = [] # 机器人的期望工作模式。
    self.q = []    # 存储期望角度的向量（以弧度为单位）
    self.dq = []   # 存储期望速度的向量（以弧度每秒为单位）。
    self.tau = []  # 存储期望输出扭矩的向量（以牛顿米为单位）。
    self.Kp = []   # 存储期望位置刚度的向量（以牛顿米每弧度为单位）。
    self.Kd = []   # 存储期望速度刚度的向量（以牛顿米每弧度每秒为单位）。
    self.motor_names = []   # 存储需要控制的关节名称。
```

代码示例：

▶

python 复制代码

4.2.8 subscribeSensorJoy 接口

函数名	subscribeSensorJoy
函数原型	def subscribeSensorJoy(self, callback: Callable[[datatypes.SensorJoy], Any])
功能概述	在真机部署中，该方法用于订阅来自机器人遥控器的数据。当机器人接收到遥控器的数据时，将会调用指定的回调函数，并传递包含遥控器数据的 datatypes.SensorJoy 结构体常量指针给回调函数进行处理。
参数	callback: 表示回调函数，用于接收机器人遥控器的数据。回调函数的参数类型为 datatypes.SensorJoy
返回值	成功：返回True 失败：返回False

备注: datatypes.SensorJoy 数据结构原型如下：

▼

python 复制代码

```
import sys

class SensorJoy(object):
  __slots__ = ['stamp','axes','buttons']
  def __init__(self):
    self.stamp = 0     # 与传感器输入相关的时间戳，单位为纳秒。
    self.axes = []     # 表示操纵摇杆的值。
    self.buttons = []  # 表示操纵按钮状态的值。
```

代码示例：

▶

python 复制代码

4.2.9 subscribeDiagnosticValue 接口

函数名	subscribeDiagnosticValue
函数原型	def subscribeDiagnosticValue(self, callback: Callable[[datatypes.DiagnosticValue], Any])
功能概述	在真机部署中，该方法用于订阅机器人的诊断值和状态信息。当机器人发出诊断值时，系统会调用指定的回调函数，并传递包含诊断值的 datatypes.DiagnosticValue 结构对象给回调函数进行处理。这可以帮助实时监控机器人的健康状态，并及时做出反应以处理可能的问题。
参数	callback: 用于接收机器人诊断值的回调函数，其参数类型为 datatypes.DiagnosticValue。datatypes.DiagnosticValue 结构体包含了机器人诊断值的信息，包括时间戳、级别、名称、代码和消息字段。
返回值	成功：返回True 失败：返回False

备注: datatypes.DiagnosticValue 数据结构原型如下：

▼

python 复制代码

```
import sys

class DiagnosticValue(object):
  __slots__ = ['stamp','level','name','code','message']
  def __init__(self):
    self.stamp = 0 # 时间戳，单位为纳秒。
    self.level = 0 # 与诊断值相关联的级别 - 0: OK, 1: WARN, 2: ERROR
    self.name = '' # 标识诊断值的名称。
    self.code = 0  # 与诊断值对应的代码。
    self.message = ''  # 与诊断值相关的详细消息。
```

代码示例：

▶

python 复制代码

4.2.10 publishJsonMessage 接口

函数名	publishJsonMessage
函数原型	def publishJsonMessage(self, json_payload: str)

函数名	publishJsonMessage
功能概述	向机器人发送 "上层应用协议接口" 的 JSON 格式消息。
参数	json_payload: 符合 "上层应用协议接口" 协议的 JSON 字符串。 示例：{"accid": "xxx", "title": "request_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
返回值	无
备注	高阶开发模式下有效

代码示例：

▶

python 复制代码

4.2.11 subscribeJsonMessage 接口

函数名	subscribeJsonMessage
函数原型	def subscribeJsonMessage(self, callback: Callable[[str], Any])
功能概述	注册一个回调函数，用于处理来自机器人的 "上层应用协议接口" 调用的响应和通知。 该回调函数会在以下两种场景中被调用： 1. 当机器人对之前发送的 JSON 命令（publishJsonMessage）返回响应时 2. 当机器人主动发起通知（未经请求的消息）时
参数	callback: 回调函数, 其中 json_payload 包含： - 命令响应：{"accid": "xxx", "title": "response_xxx", "timestamp": xxx, "guid": "xxx", "data": {}} - 通知：{"accid": "xxx", "title": "notify_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
返回值	无
备注	高阶开发模式下有效

代码示例：

▶

python 复制代码

4.2.12 参考例程

Github: https://github.com/limxdynamics/humanoid-rl-deploy-python

5 查看/设置机器人型号

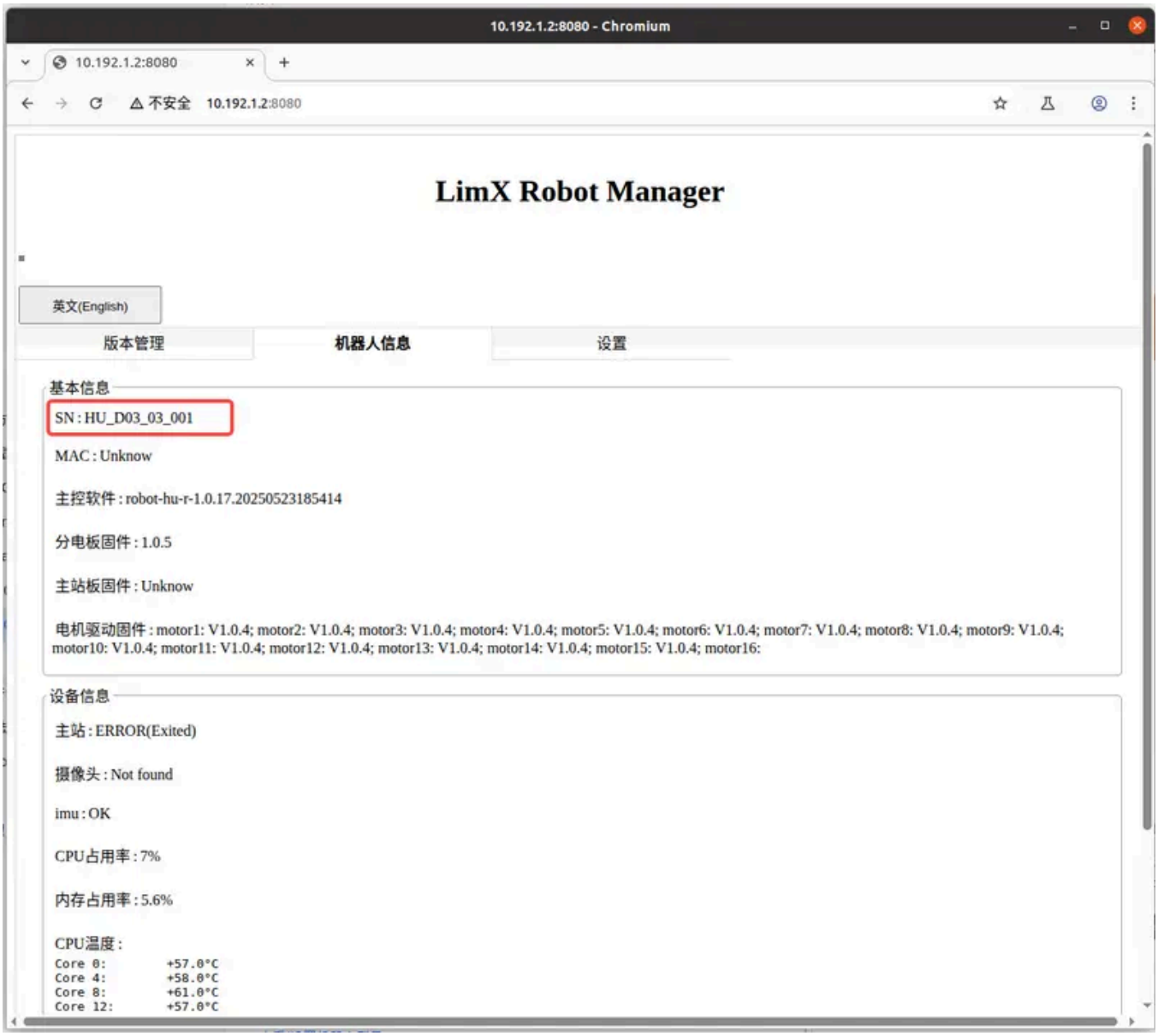
在编译和运行RL训练、控制算法及仿真器程序时，选择正确的机器人型号至关重要。通过查看机器人型号并将其设置到环境变量 `ROBOT_TYPE` 中，确保在不同任务中准确识别并应用相应的机器人模型。

查看和配置机器人型号步骤：

1. 选择并连接您机器人Wi-Fi 热点，密码为：`12345678`



1. 在浏览器中输入 `http://10.192.1.2:8080` 进入“机器人信息页”，查看机器人信息。如下图所示，页面中显示的SN (序列号) 为HU_D03_03_001，其中HU_D03_03为机器人型号。



1. 设置机器人型号：打开 Bash 终端，输入以下 Shell 命令来设置机器人型号。这样在二次开发时，您将能获取到正确的机器人型号信息。

▼

bash 复制代码

```
echo 'export ROBOT_TYPE=HU_D03_03' >> ~/.bashrc && source ~/.bashrc
```

6 机器人仿真器

MuJoCo 是一款轻量级且高性能的物理仿真器，专为多关节机器人和机械系统设计。它具备高效的物理引擎，能够精确处理接触和摩擦，且无需依赖 ROS，可独立运行。凭借其高速计算能力，MuJoCo 被广泛应用于机器人仿真和强化学习，尤其适合对仿真效率要求较高的场景。

6.1 运行仿真器步骤

1. 运行环境：推荐 Pyhon 3.8 及以上版本
2. 打开一个 Bash 终端。
3. 下载 MuJoCo 仿真器代码：

▼

bash 复制代码

```
git clone --recurse git@github.com:limxdynamics/humanoid-mujoco-sim.git
```

4. 安装运动控制开发库：

- Linux x86_64 环境

▼

bash 复制代码

```
pip install humanoid-mujoco-sim/limxsdk-lowlevel/python3/amd64/limxsdk-*-py3-none-any.whl
```

- Linux aarch64 环境

▼

bash 复制代码

```
pip install humanoid-mujoco-sim/limxsdk-lowlevel/python3/aarch64/limxsdk-*-py3-none-any.whl
```

1. 设置机器人型号：请参考"查看/设置机器人型号"章节，查看您的机器人型号。如果尚未设置，请按照以下步骤进行设置。

- 通过 Shell 命令 `tree -L 3 -P "meshes" -I "urdf|world|xml|usd" humanoid-mujoco-sim/humanoid-description` 列出可用的机器人类型：

▼

bash 复制代码

```
limx@limx:~$ tree -L 3 -P "meshes" -I "urdf|world|xml|usd" humanoid-mujoco-sim/humanoid-description
humanoid-mujoco-sim/humanoid-description
├── HU_D03_description
│   └── meshes
│       └── HU_D03_03
└── HU_D04_description
```

```
    └── meshes
        └── HU_D04_01
```

- 以`HU_D04_01`（请根据实际机器人类型进行替换）为例，设置机器人型号类型：

▼

bash 复制代码

```
echo 'export ROBOT_TYPE=HU_D04_01' >> ~/.bashrc && source ~/.bashrc
```

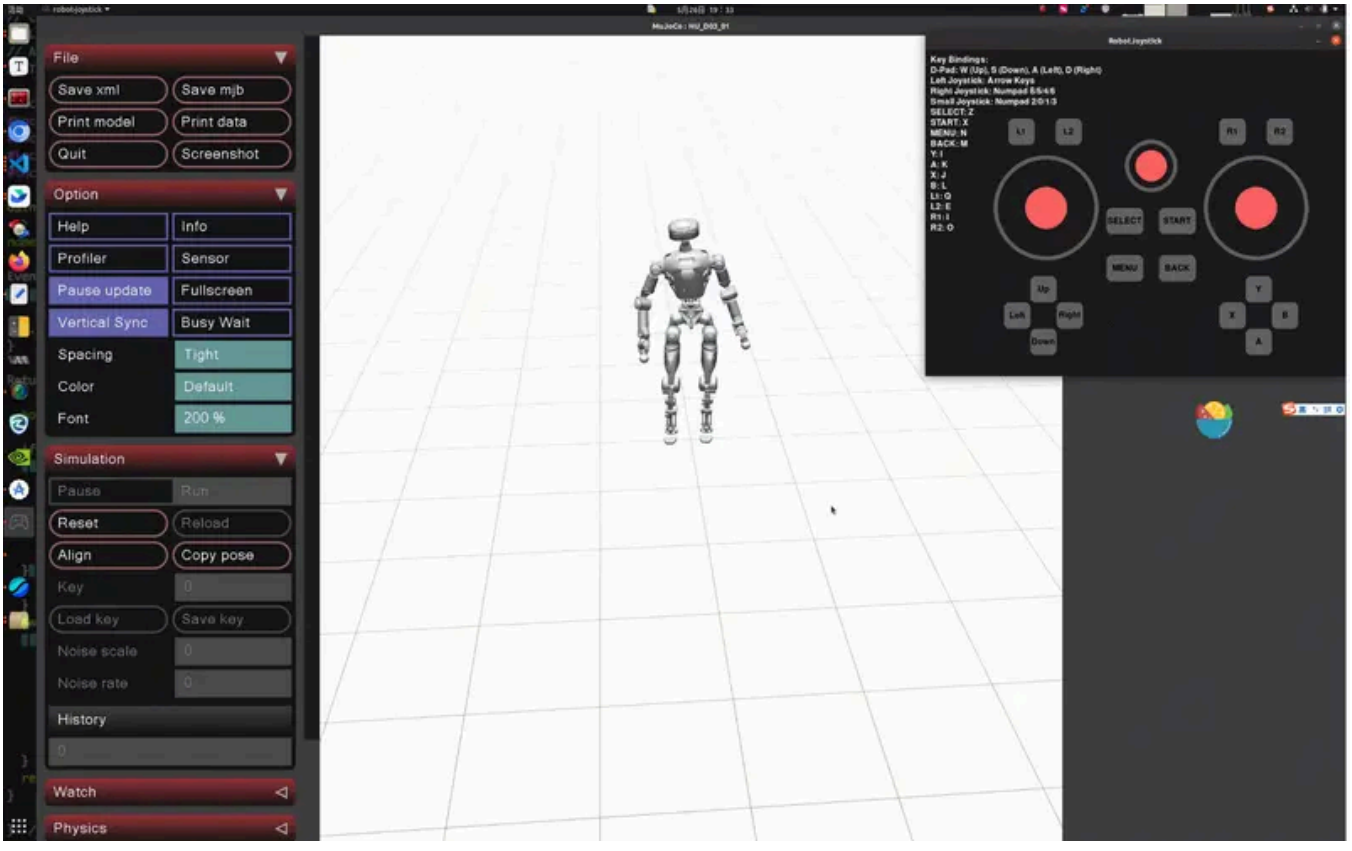
1. 运行 MuJoCo 仿真器：

▼

bash 复制代码

```
python humanoid-mujoco-sim/simulator.py
```

6.2 演示效果(请以实际为准)



7 RL 算法部署

7.1 基于标准C++进行部署

Github项目地址：https://github.com/limxdynamics/humanoid-rl-deploy-cpp，它是一个在标准C++中实现的轻量级算法框架。无需 ROS1/ROS2 即可快速部署训练的模型。

7.2 基于Python进行部署

Github项目地址：https://github.com/limxdynamics/humanoid-rl-deploy-python，它是一种基于Python的强化学习部署算法框架，它简化了在Oli机器人上部署训练模型的过程。

7.3 基于ROS2 进行部署

Github项目地址：https://github.com/limxdynamics/humanoid-rl-deploy-ros2，它是基于 ROS2 的强化学习部署框架，可在 Oli 机器人上快速部署经过训练的模型。

7.4 基于ROS1 进行部署

Github项目地址：https://github.com/limxdynamics/humanoid-rl-deploy-ros，它是基于[ROS1](https://www.ros.org)的强化学习部署框架，可以在Oli机器人上快速部署经过训练的模型。

8 日志和数据包

- **机器人系统自动录制数据：** IMU数据(ImuData)、状态数据（/joint/state）、控制数据（/joint/cmd）、运行日志数据等重要数据，这些数据对于机器人的运动控制分析至关重要。
- **访问并下载机器人数据：** 机器人将会记录运行时的日志数据，以便在需要时进行故障排查和性能优化，电脑与机器人 WiFi 热点连接后浏览器输入 `http://10.192.1.2:8090`，可自行选择下载所需数据。

8.1 数据包可视化分析方法

- **数据包下载：** 下载 `.bag` 文件后，可以使用PlotJuggler可视化工具加载这些数据包进行分析。需要特别注意的是，如果下载的是 `.bag.active` 文件，需要使用如下Shell命令把它重新索引 `.bag.active` 文件，生成一个新的 `.bag` 文件，以便PlotJuggler加载。

▼

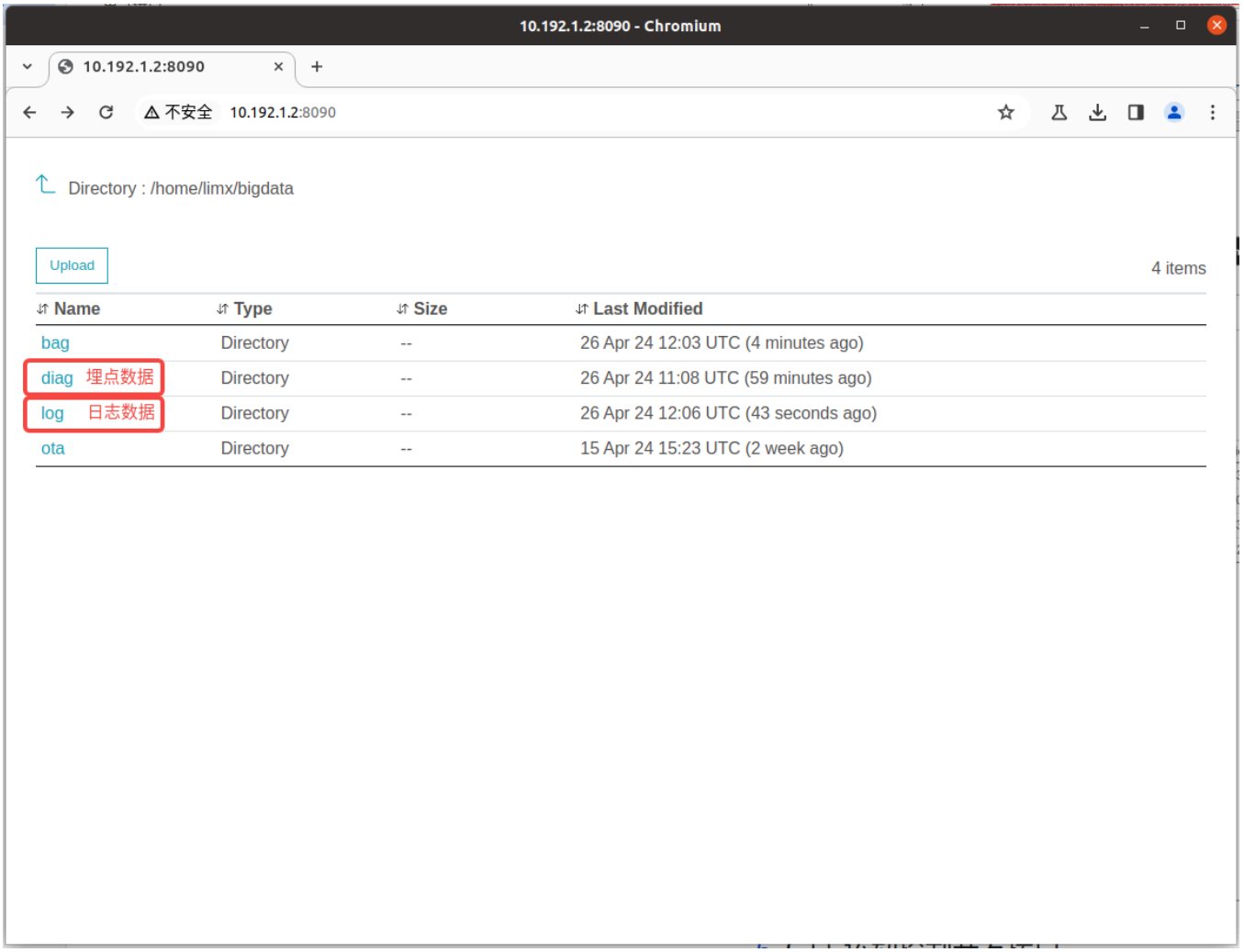
bash 复制代码

```
rosbag reindex your_file.bag.active
mv your_file.bag.active your_file.bag
```

- **可视化查看：** 通过Shell命令 `rosrun plotjuggler plotjuggler -n` 启动PlotJuggler可视化工具。如下图所示加载数据包进行可视化分析。

8.2 日志及诊断埋点数据

如下图所示分别为日志和埋点结构化数据。在需要时可以用于故障排查和性能优化。



9 机器人软件升级

软件升级注意事项：

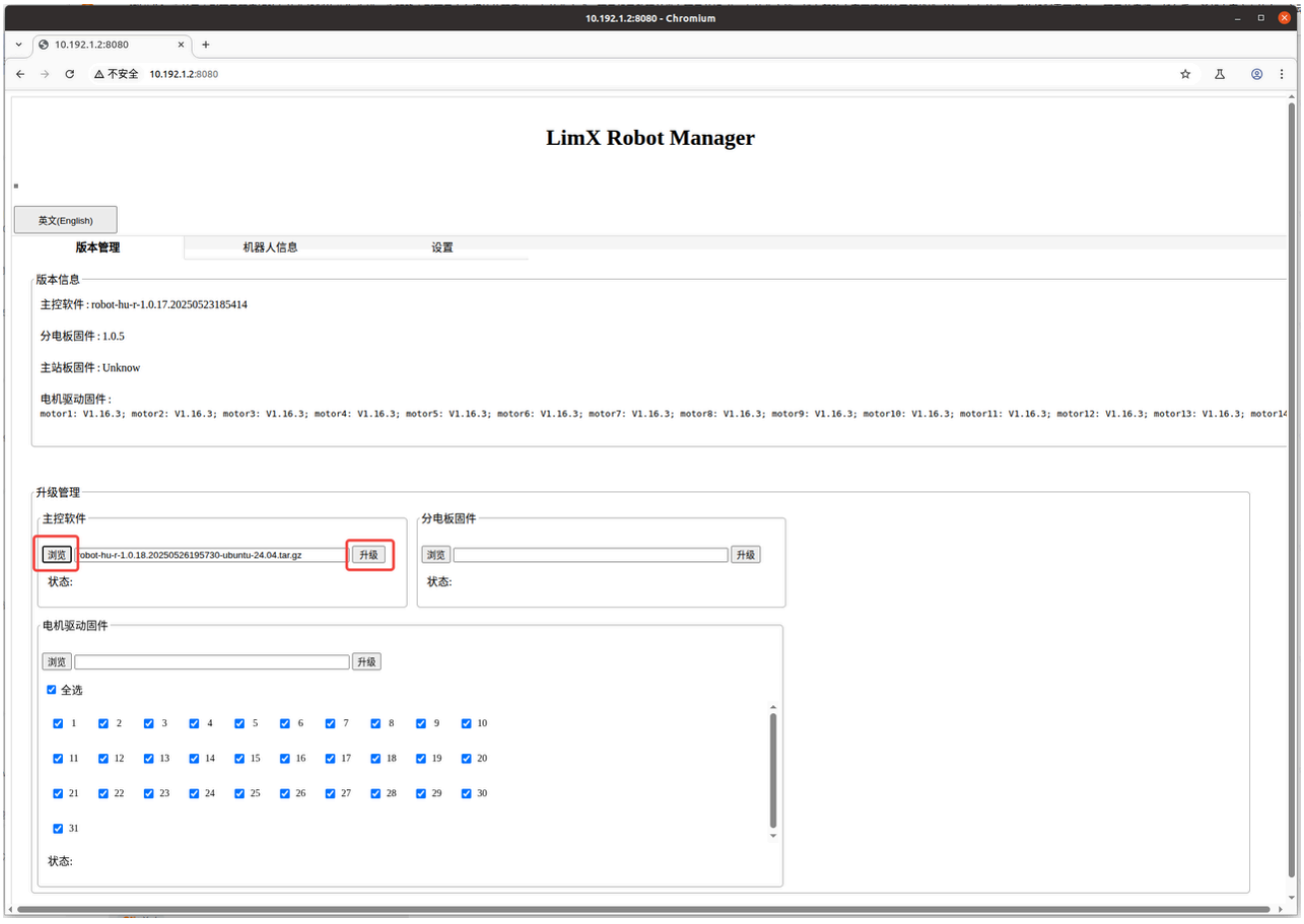
1. 请确保设备电量不低于 30%。升级过程中如发生断电，可能导致设备无法正常使用
2. 请提前将机器人切换至零力矩模式或阻尼模式。升级完成后设备将自动重启，若机器人处于站立状态,可能存在跌倒风险。

通过浏览器进入机器人管理页面,选择本地提前下载好的机器人软件版本进行升级。具体步骤如下：

1. 连接 Wi-Fi：
    - 选择并连接机器人的 Wi-Fi 热点，密码为：12345678



2. 访问管理页面：
    - 在浏览器地址栏输入： `http://10.192.1.2:8080` 进入机器人管理页面。
3. 选择并升级软件：
    - 依次选择"版本管理 -> 浏览 -> 升级"。
    - 升级完成后，机器人主控电脑将自动重启。



10 开发者电脑

开发者电脑主要用于开发机器人相关算法及应用程序。可以通过 WiFi 连接机器人本体系统登录到开发者电脑，具体步骤如下：

1. 连接Wi-Fi：
    - 选择并连接机器人的 Wi-Fi 热点，密码为：12345678



2. 通过 SSH 登录开发者电脑系统：
    - 登录地址：10.192.1.3
    - 登录密码：123456
    - 在终端中输入以下命令（首次连接需确认指纹）：

▼

bash 复制代码

```
ssh guest@10.192.1.3
```

- 开发者电脑系统配置：
    - 操作系统：Ubuntu 22.04 （Jetpack 6.2.1）
    - ROS2：系统默认安装ROS2 Humble 版本机器人系统
    - ROS1：系统默认安装ROS1 Noetic 版本机器人系统Docker，进入方法如下：

▼

bash 复制代码

```
sudo docker exec -it ros_noetic /bin/bash
```

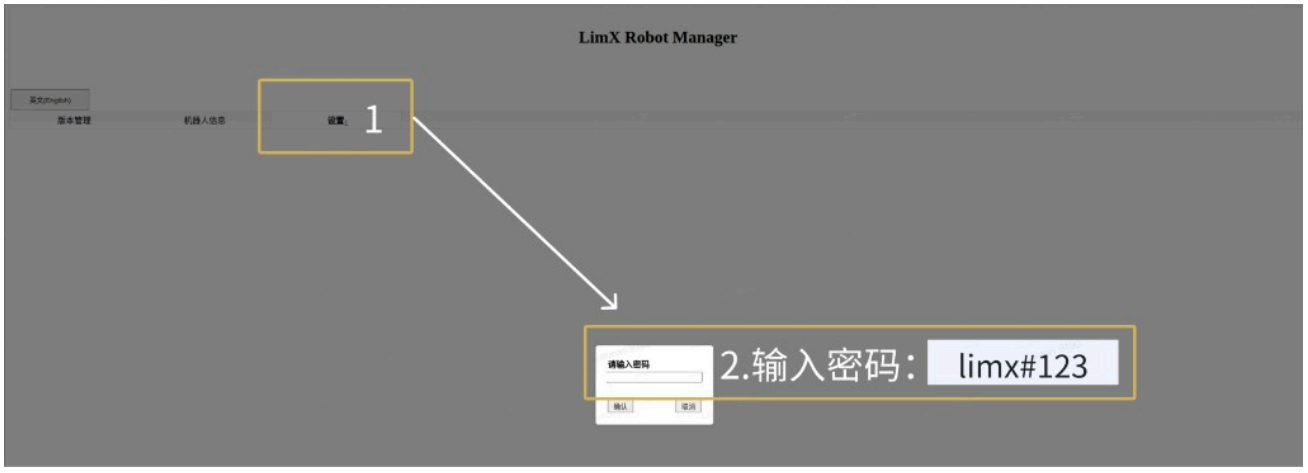
11 Realsense 相机数据获取

注意事项：
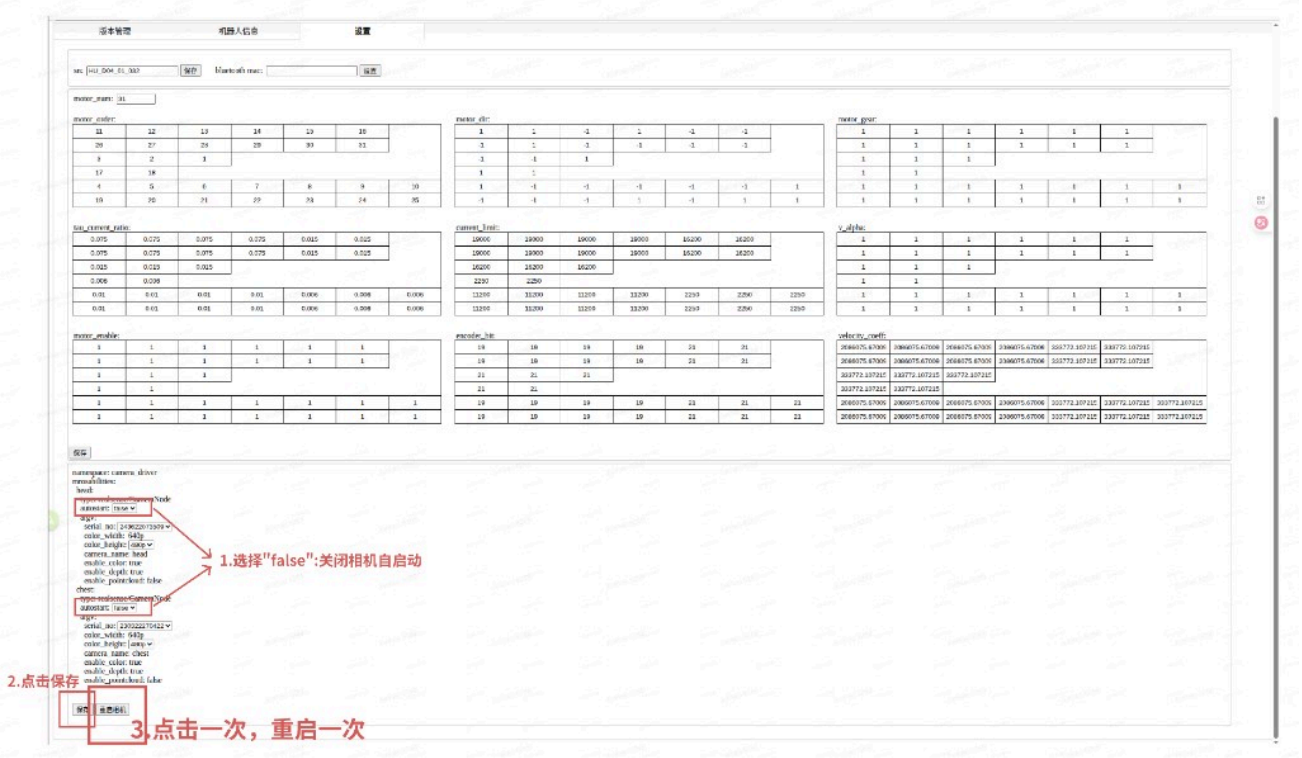
1. 机器人开机后默认会自动启动相机驱动。因此，在进行以下操作前，请先关闭相机驱动的自动启动功能。关闭方法请参考下文说明。该功能仅支持主控版本 V2.0.33 及以上。
2. 关闭相机驱动自动启动功能后，我司配套的数据采集套件（数采套件）将无法正常使用。请根据实际需求决定是否执行该操作。

11.1 关闭相机驱动自启动

1. 在浏览器地址栏输入 `http://10.192.1.2:8080`，进入以下界面。



2. 设置关闭并保存。



11.2 获取相机数据

1. 登录开发者电脑：
   - 请按照“开发者电脑”章节中的步骤登录该电脑。
2. 启动 Realsense 的 ROS 节点以获取相机数据：
   - 启动方式参考链接：realsense-ros
   - 电脑已预装 Realsense 相机的 SDK：
     - 版本为 v2.56.3，官方下载链接：librealsense v2.56.3，可基于此官方 SDK 自主开发应用程序来获取数据。
3. 示例代码说明：
   - 相机命名规则：
     - 默认命名：脚本会将多个相机的 topic 前缀命名为 `camera` 加上序号（如 `camera0`、`camera1`）。
     - 自定义命名：可通过修改脚本，根据相机的序列号（Serial Number）指定 topic 前缀，而非使用默认的 `camera + counter` 命名规则。

以下代码示例展示了如何获取多个相机的数据：

1. 通过 SSH 登录开发者电脑系统
2. 登录 ROS1 系统

▼

bash 复制代码

```
sudo docker exec -it ros_noetic /bin/bash
```

3. 将下面的脚本保存为 `rs_camera.sh`：

▶

bash 复制代码

4. 在终端执行脚本，启动相机节点：

▼

bash 复制代码

```
/bin/bash rs_camera.sh
```

5. 在另一个终端中，通过 `rostopic list` 验证结果：

▼

bash 复制代码

```
rostopic list
```