【产品】Luna 二次开发指南

版本	修订日期	修订人	修改内容	备注
V1.0	2026-08-26	Eleven	初稿	

1. SDK介绍

1.1 上层应用协议接口

1.1.1 概述

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。

1.1.2 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：

```
代码块
1  {
2    "accid": "HU_D02_001", # 机器人唯一序列号，标识机器人的唯一身份
3    "title": "request_xxx",    # 指令名称，以“request_”为前缀
4    "timestamp": 1672373633989, # 指令发出时间戳，单位为毫秒
5    "guid": "746d937cd8094f6a98c9577aaf213d98", # 指令的唯一标识符，用于区分不同的请求指令
6    "data": {}  # 存放请求指令的数据内容
7  }
```

- 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：

```
代码块
1  {
2    "accid": "HU_D02_001",    # 机器人唯一序列号，标识机器人的唯一身份
3    "title": "response_xxx",  # 指令名称，以“response_”为前缀
4    "timestamp": 1672373633989, # 指令发出时间戳，单位为毫秒
5    "guid": "746d937cd8094f6a98c9577aaf213d98", # 与对应请求指令的guid值相同
6    "data": { # 存放响应指令的具体数据内容
7      "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”
8    }
9  }
```

- 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：

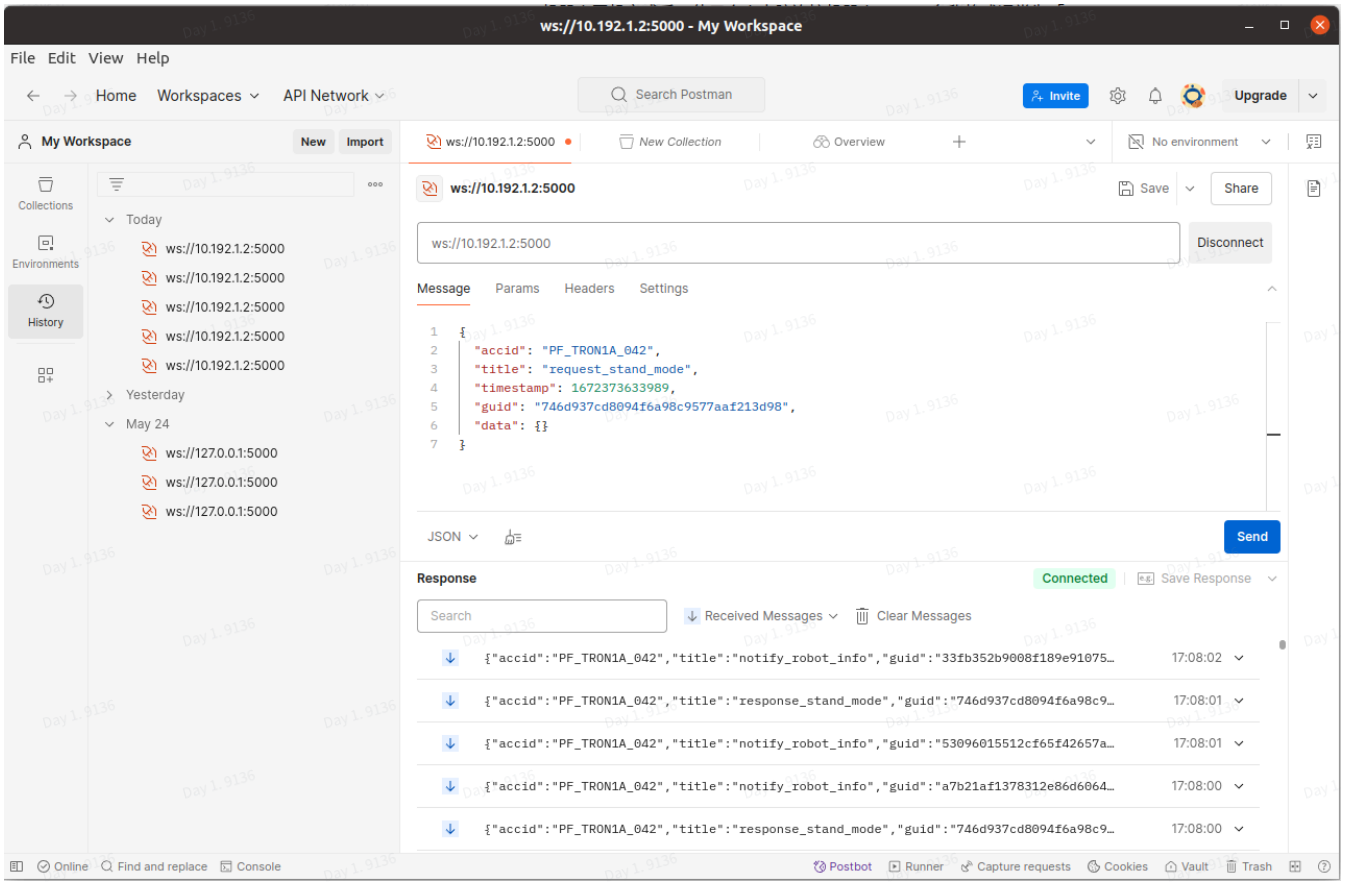
代码块

```
{
  "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
}
```

1.1.3 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，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「HU_D02_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。



1.2 基础功能协议接口

1.2.1 设置提示音语言

1.2.1.1 请求：request_set_audio_prompts_language

本协议用于设置机器人提示音语言。请求成功后，机器人后续语音提示将使用指定语言，并播放一段切换确认提示音，方便用户确认设置已经生效。请求参数，language 取值：

language	含义	用户感知
cn	中文提示音	后续提示音使用中文，并播放中文切换确认提示音
en	英文提示音	后续提示音使用英文，并播放英文切换确认提示音

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_audio_prompts_language",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "language": "cn"
  }
```

```
}
```

1.2.1.2 响应：response_set_audio_prompts_language

代码块

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

result 取值

result	含义	用户提示建议
success	设置成功	提示音语言已切换
fail_no_language	未传入目标语言	请选择提示音语言
fail_invalid_language	语言取值不支持	当前仅支持中文和 English
fail_write_robot_info	设置保存失败	设置失败，请稍后重试
fail_play_audio_prompt_service_not_ready	语音功能暂未就绪	语音功能暂未就绪，请稍后重试
fail_play_audio_prompt_call	切换确认提示音请求失败	切换确认失败，请重试
fail_play_audio_prompt	切换确认提示音播放失败	未听到确认提示音，请重试

1.2.1.3 消息推送：无

1.2.2 连接Wifi热点

1.2.2.1 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。

代码块

```
{
```

```
  "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
  }
}
```

1.2.2.2 响应：response_connect_wifi

代码块

```
{
  "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
  }
}
```

1.2.3 查询Wifi连接状态

1.2.3.1 请求：request_wifi_connection_status

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。

代码块

```
{
  "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
    }
}
```

1.2.3.2 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。

代码块

```
{
    "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
    }
}
```

1.2.4 进入准备状态

机器人缓慢摆出准备姿势。

1.2.4.1 请求：request_prepare

控制机器人进入站立状态。

代码块

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

1.2.4.2 响应：response_prepare

代码块

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

1.2.5 控制机器人行走

1.2.5.1 进入行走模式

机器人进入行走模式，可以接收速度指令。

1.2.5.2 请求：request_set_walk_mode

代码块

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

1.2.5.3 响应：response_set_walk_mode

代码块

```
{
  "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
  }
}
```

1.2.6 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（需> 10 Hz下发）。请注意，在全身操作模式下，此协议接口无效。

1.2.6.1 请求：request_set_walk_vel

代码块

```
{
  "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]
  }
}
```

1.2.6.2 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。

代码块

```
{
  "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
  }
}
```

1.2.6.3 消息推送：无

1.2.7 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。

1.2.7.1 请求：request_damping

代码块

```
{
```

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

1.2.7.2 响应：response_damping

代码块

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

1.2.8 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。

1.2.8.1 请求：request_zero_torque

代码块

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

1.2.8.2 响应：response_zero_torque

代码块

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

```
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "data": {
        "result": "success" # fail_motor: 电机错误
    }
}
```

1.2.9 进入站立指令

💡 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]

返回：已站起后返回

需求链接：启动与回收机器人产品需求文档_V1.0

1.2.9.1 请求：request_standup

代码块

```
{
    "accid": "HU_D04_01_001",
    "title": "request_standup",
    "timestamp": 1672373633989,
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "data": {
        "mode": "lying" // "lying": 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
    }
}
```

1.2.9.2 响应：response_standup

代码块

```
{
    "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
```

```
10                          # fail_timeout: 执行超时错误
11      }
12  }
```

1.2.10 进入躺着指令

1.2.10.1 请求：request_lie_down

💡 Walk状态下可以调用该接口

代码块

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

1.2.10.2 响应：response_lie_down

代码块

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

1.2.11 机器人舞蹈

1.2.11.1 切换机器人到舞蹈模式

1.2.11.1.1 请求：request_enter_dance_mode

代码块

```
{
```

```
  "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
  }
}
```

1.2.11.1.2 响应：response_enter_dance_mode

代码块

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

1.2.11.2 获取舞蹈列表

1.2.11.2.1 请求：request_get_dance_list

代码块

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

1.2.11.2.2 响应：response_get_dance_list

代码块

```
{
    "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",
                "duration": 10
            },
            {
                "id": "DAN-08",
                "index": 1,
                "name": "\u4f4e\u4fd7\u5c0f\u8bf4",
                "english_name": "Pulp Fiction Dance",
                "rc_mapping": "pulp_fiction_dance",
                "duration": 10
            }
        ]
    }
}
```

1.2.11.3 机器人跳舞

1.2.11.3.1 请求：request_dance

💡 执行前置条件：当前处于动作库模式

代码块

```
{
  "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
  }
}
```

1.2.11.3.2 响应：response_dance

代码块

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

1.2.11.3.3 消息推送：notify_dance

跳完舞蹈或执行过程中失败推送此消息。

代码块

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

1.2.12 机器人动作库

1.2.12.1 动作打断

1.2.12.1.1 请求：request_interrupt_action_joystick

代码块

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

1.2.12.1.2 响应：

代码块

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

1.2.12.2 获取动作库状态

💡 接口说明：

（1）机器人进入动作库之后，无论是处于动作库/原子执行/舞蹈中，"action_library_mode": "action_library"

（2）当机器人在执行原子动作中或者舞蹈中，"action_library_state": "running"

1.2.12.2.1 请求：request_get_action_library_status

代码块

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

1.2.12.2.2 响应：response_get_action_library_status

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "get_action_library_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "action_library_mode": "action_library" // or "remote_control"
```

```
        "action_library_state": "running"       //or "idle"
        "result": "success" # fail_motor
    }
}
```

1.2.12.3 切换机器人到动作库模式

1.2.12.3.1 请求：request_set_motion_engine

代码块

```
{
    "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
    }
}
```

1.2.12.3.2 响应：response_set_motion_engine

代码块

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

1.2.12.4 执行动作库

1.2.12.4.1 请求：request_action_sync

代码块

```
{
    "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
      "music": "bgm1.wav,bgm2.wav,"    # 可选,仅支持.wav格式的文件

  }
}
```

1.2.12.4.2 响应：response_action_sync

代码块

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

1.2.12.5 获取动作库列表

1.2.12.5.1 请求：request_get_atomic_motion_list

代码块

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

1.2.12.5.2 响应：response_get_atomic_motion_list

代码块

```
{
    "accid": "HU_D04_01_001",
    "title": "response_get_atomic_motion_list",
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "timestamp": 287883835,
    "data": {
        "result": "success",
        "motion_list": [
            {
                "id": "MON-01",
                "rc_mapping": "greeting1",
                "duration": 7,
                "motion_index": 0,
                "motion_name_cn": "侧身打招呼",
                "motion_name_en": "Greeting1"
            },
            {
                "id": "MON-02",
                "rc_mapping": "greeting2",
                "duration": 9,
                "motion_index": 1,
                "motion_name_cn": "抬腿打招呼",
                "motion_name_en": "Greeting2"
            }
            ......
        ],
        "count": 2
    }
}
```


1.2.13 全局消息协议接口

1.2.14 机器人状态信息

通过此协议定时上报机器人状态信息，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：

代码块

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

1.2.14.1 电池数据

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": [
        ......
        {
                "level": 0,
                "name": "peripheral",
                "message": "OK",
                "hardware_id": "peripheral",
                "values": [
                    {
                        "key": "bmsconn",
                        "value": "ON"
                    },
                    {
                        "key": "bat_chg",
                        "value": "OFF"
                    },
                    {
                        "key": "bat_off",
                        "value": "OFF"
                    },
                    {
                        "key": "bat_prt",
                        "value": "0"
                    },
                    {
                        "key": "bat_vol",
```

```
                    "value": "48830"
                },
                {
                    "key": "bat_cur",
                    "value": "2870"
                },
                {
                    "key": "battery",
                    "value": "29"
                },
                {
                    "key": "bat_temp0",
                    "value": "430"
                },
                {
                    "key": "bat_temp2",
                    "value": "430"
                },
                {
                    "key": "bat_temp4",
                    "value": "400"
                },
                {
                    "key": "battery_capacity",
                    "value": "9000mAh"
                }
            ]
        },
    ]
  }
}
```

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】
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℃

1.2.14.2 手柄信息数据（仅Lite机型包含该数据）

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": [
        ......
        {
                "level": 0,
                "name": "peripheral",
                "message": "OK",
                "hardware_id": "peripheral",
                "values": [
                    {
                        "key": "joystickconn",
                        "value": "ON"
                    },
                    {
                        "key": "joysticksignal",
                        "value": "1"
                    },
                    {
                        "key": "joystickbattery",
                        "value": "2"
                    }
                ]
            },
    ]
  }
}
```

字段	含义

joystickconn	手柄连接状态： 【OFF：未连接，ON：已连接】
joysticksignal	手柄信号强度： 【0-4 越高越强】
joystickbattery	手柄电量信息： 【0:0%~20% 1:21%~40% 2:41%~60% 3:61%~80% 4:81%~100%】

1.2.14.3 系统信息

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": [
          {
              "level": 0,
              "name": "system_info",
              "message": "system info",
              "hardware_id": "system_info",
              "values": [
                ####oli和luna共用字段
                {
                  "key": "ability_running",
                  "value": "ZeroTorque"
                },
                {
                  "key": "ecm_version",
                  "value": "1.1.2"
                },
                {
                  "key": "mode",
                  "value": "Remote"
                },
                {
                  "key": "motor_version",
                  "value": "1: 0.0.9; 2: 0.0.9; 3: 0.0.9; 4: 0.0.9; 5: 0.0.9; 6: 0.0.9; 7: 0.0.9; 8: 0.0.9; 9: 0.0.9; 10: 0.0.9; 11: 0.0.9; 12: 0.0.9; 13: 0.0.9; 14: 0.0.9; 15: 0.0.9; 16: 0.0.9; "
```

```
                },
                {
                  "key": "pms_version",
                  "value": "2.1.8"
                },
                {
                  "key": "robot_status",
                  "value": "ZeroTorque"
                },
                {
                  "key": "version",
                  "value": "robot-hu-d-2.1.0.20251225062343"
                },
                {
                  "key": "sn",
                  "value": "HU_D04_01_131"
                },
                ##########luna特有字段
                { "key": "sdk_lite_led_enable", "value": "1" },
                { "key": "sdk_lite_led_has_params", "value": "1" },
                { "key": "sdk_lite_led_mode", "value": "0" },
                { "key": "sdk_lite_led_state", "value": "1" },
                { "key": "sdk_lite_led_color", "value": "4" },
                { "key": "sdk_lite_led_brightness", "value": "3" },
                { "key": "sdk_lite_leds", "value": "" },
                {"key": "walk_gait", "value":"walk"}
              ]
          }
      ]
    }
}
```

oli和luna公用

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

luna特有字段

字段	含义
sdk_lite_led_enable()	luna 灯效控制开关： `0` =关闭， `1` =开启
sdk_lite_led_has_params	是否已有 luna 灯效参数： `0` =无， `1` =有
sdk_lite_led_mode	luna 灯效控制模式： `0` =整体灯效控制， `1` =单颗灯 RGB 控制
sdk_lite_led_state	整体灯效状态， `mode=0` 时有效
sdk_lite_led_color	整体灯效颜色， `mode=0` 时有效
sdk_lite_led_brightness	整体灯效亮度， `mode=0` 时有效，范围 `0-5`
sdk_lite_leds	单颗灯 RGB 扁平数组字符串， `mode=1` 时有效，例如 `255,0,0,0,255,0`

代码块

```
如果是     mode = 1     单颗 RGB 控制:
{ "key": "sdk_lite_led_mode", "value": "1" },
{ "key": "sdk_lite_led_state", "value": "" },
{ "key": "sdk_lite_led_color", "value": "" },
{ "key": "sdk_lite_led_brightness", "value": "" },
{ "key": "sdk_lite_leds", "value": "255,0,0,0,255,0,0,0,255" }
禁用时:
{ "key": "sdk_lite_led_enable", "value": "0" },
{ "key": "sdk_lite_led_has_params", "value": "0" },
{ "key": "sdk_lite_led_mode", "value": "" },
{ "key": "sdk_lite_led_state", "value": "" },
{ "key": "sdk_lite_led_color", "value": "" },
{ "key": "sdk_lite_led_brightness", "value": "" },
{ "key": "sdk_lite_leds", "value": "" }
```

1.2.14.4 电机状态信息

代码块

```
{
  "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

1.2.15 遥控器数据

通过此协议上报机器人遥控器数据：

- `accid`：机器人序列号
- `title`：notify_joy_data
- `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：

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

1.2.16 协议接口调用示例

1.2.16.1 Python 示例实现

- 环境准备：以Ubuntu 20.04系统为例，安装下面依赖

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

- 运行脚本

```
代码块
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

代码块

```python
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)
ACCID = None

# Replace it with the real IP address of the robot.
# Usually, for simulation, it is: 127.0.0.1
# for a real machine, it is: 10.192.1.2
ROBOT_IP = "10.192.1.2"

# 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):
    global ACCID
    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)

```

```python
        # 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 ('prepare', 'damping', 'zero') or 'exit' to quit:\n")

            if command == "exit":
                should_exit = True  # Set exit flag to stop the loop
                break
            elif command == "prepare":
                send_request("request_prepare")  # request_prepare
            elif command == "damping":
                send_request("request_damping")  # request_damping
            elif command == "zero":
                send_request("request_zero_torque")  # request_zero_torque

    # WebSocket on_open callback
    def on_open(ws):
        print("Connected!")
        # Start handling commands in a separate thread
        threading.Thread(target=handle_commands, daemon=True).start()

    # WebSocket on_message callback
    def on_message(ws, message):
        global ACCID
        root = json.loads(message)
        title = root.get("title", "")
        ACCID = root.get("accid", None)

        if title != "notify_robot_info":
            print(f"Received message: {message}")  # Print the received message

    # WebSocket on_close callback
    def on_close(ws, close_status_code, close_msg):
        print("Connection closed.")

    # Close WebSocket connection
    def close_connection(ws):
        ws.close()

    def main():
        global ws_client
```

```
89
90      # Create WebSocket client instance
91      ws_client = websocket.WebSocketApp(
92          f"ws://{ROBOT_IP}:5000",  # WebSocket server URI
93          on_open=on_open,
94          on_message=on_message,
95          on_close=on_close
96      )
97
98      # Configure socket send and receive buffer sizes
99      # Increase send buffer size to 2MB (default is typically much smaller)
100     # This helps prevent data loss when sending large messages or high-
    frequency data
101     ws_client.sock_opt = [("socket", "SO_SNDBUF", 2 * 1024 * 1024)]
102
103     # Increase receive buffer size to 2MB
104     # This allows handling larger incoming messages without truncation
105     ws_client.sock_opt.append(("socket", "SO_RCVBUF", 2 * 1024 * 1024))
106
107     # Run WebSocket client loop
108     print("Press Ctrl+C to exit.")
109     ws_client.run_forever()
110
111 if __name__ == "__main__":
112     main()
113
```

1.2.16.2 Linux C++ 示例

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

代码块

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

- 编译代码

代码块

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

- 运行程序

```
./humanoid
```

- humanoid.cpp 实现

代码块

```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 value with the actual serial number (SN) of the robot.
static std::string ACCID = "";

// Replace it with the real IP address of the robot.
// Usually, for simulation, it is: 127.0.0.1
// for a real machine, it is: 10.192.1.2
const std::string ROBOT_IP = "10.192.1.2";

// 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);
}

```

```
41  // Send WebSocket request with title and data
42  static void send_request(const std::string& title, const json& data =
    json::object()) {
43    json message;
44
45    // Adding necessary fields to the message
46    message["accid"] = ACCID;
47    message["title"] = title;
48    message["timestamp"] =
    std::chrono::duration_cast<std::chrono::milliseconds>(
49
    std::chrono::system_clock::now().time_since_epoch()).count();
50    message["guid"] = generate_guid();
51    message["data"] = data;
52
53    std::string message_str = message.dump();
54
55    // Send the message through WebSocket
56    ws_client.send(current_hdl, message_str,
    websocketpp::frame::opcode::text);
57  }
58
59  // Handle user commands
60  void handle_commands() {
61    std::cout << "Enter command ('prepare', 'damping', 'zero') or 'exit' to
    quit:\n";
62    while (!should_exit) {
63        std::string command;
64        std::cin >> command;
65
66        if (command == "exit") {
67            should_exit = true;
68            return;
69        } else if (command == "prepare") {
70            send_request("request_prepare");
71        } else if (command == "damping") {
72            send_request("request_damping");
73        } else if (command == "zero") {
74            send_request("request_zero_torque");
75        }
76
77        sleep(1);
78
79        std::cout << "\nEnter command ('prepare', 'damping', 'zero') or
    'exit' to quit:\n";
80    }
81  }
```

```
82
83  // WebSocket open callback
84  static void on_open(connection_hdl hdl) {
85    std::cout << "Connected!" << std::endl;
86
87    // Save connection handle for sending messages later
88    current_hdl = hdl;
89
90    // Start handling commands in a separate thread
91    std::thread(handle_commands).detach();
92  }
93
94  // WebSocket TCP initialization handler
95  static void on_tcp_init(connection_hdl hdl)
96  {
97  auto con = ws_client.get_con_from_hdl(hdl);
98
99  // Obtain the underlying TCP socket
100 auto& socket = con->get_socket().lowest_layer();
101
102 // Configure socket options
103 try {
104   boost::system::error_code ec;
105
106   // Set send buffer size (e.g., 2MB)
107   const size_t sendBufferSize = 2 * 1024 * 1024;
108
    socket.set_option(websocketpp::lib::asio::socket_base::send_buffer_size(send
    BufferSize), ec);
109
110   if (ec)
111   {
112     printf("Failed to set send buffer size: %s", ec.message().c_str());
113   }
114
115   // Set receive buffer size (e.g., 2MB)
116   const size_t recvBufferSize = 2 * 1024 * 1024;
117
    socket.set_option(websocketpp::lib::asio::socket_base::receive_buffer_size(r
    ecvBufferSize), ec);
118
119   if (ec)
120   {
121     printf("Failed to set receive buffer size: %s", ec.message().c_str());
122   }
123
124   // Disable Nagle's algorithm to reduce latency
```

```
    socket.set_option(websocketpp::lib::asio::ip::tcp::no_delay(true), ec);

    if (ec)
    {
      printf("Failed to disable Nagle's algorithm: %s", ec.message().c_str());
    }
  } catch (const std::exception& e) {
    printf("Socket configuration exception: %s", e.what());
  }
  }

  // WebSocket message callback
  static void on_message(connection_hdl hdl,
  client<websocketpp::config::asio>::message_ptr msg) {
  // Parse JSON data from message payload
  json data = json::parse(msg->get_payload());

  // Extract 'accid' field if present
  if (data.contains("accid") && data["accid"].is_string() && ACCID.empty()) {
      ACCID = data["accid"].get<std::string>();
  }

  if (msg->get_payload().find("notify_robot_info") == std::string::npos) {
      std::cout << "Received message: " << msg->get_payload() << std::endl;
  }
  }

  // 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

    ws_client.set_access_channels(websocketpp::log::alevel::none);

    // 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
```

```cpp
    ws_client.set_tcp_init_handler(&on_tcp_init); // Set tcp init handler

    std::string server_uri = "ws://" + ROBOT_IP + ":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;
}
```

函数名	**subscribeImuData**
函数原型	void subscribeImuData(std::function<void(const ImuDataConstPtr&)> cb);
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：
	代码块
	`/**`
	`* @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

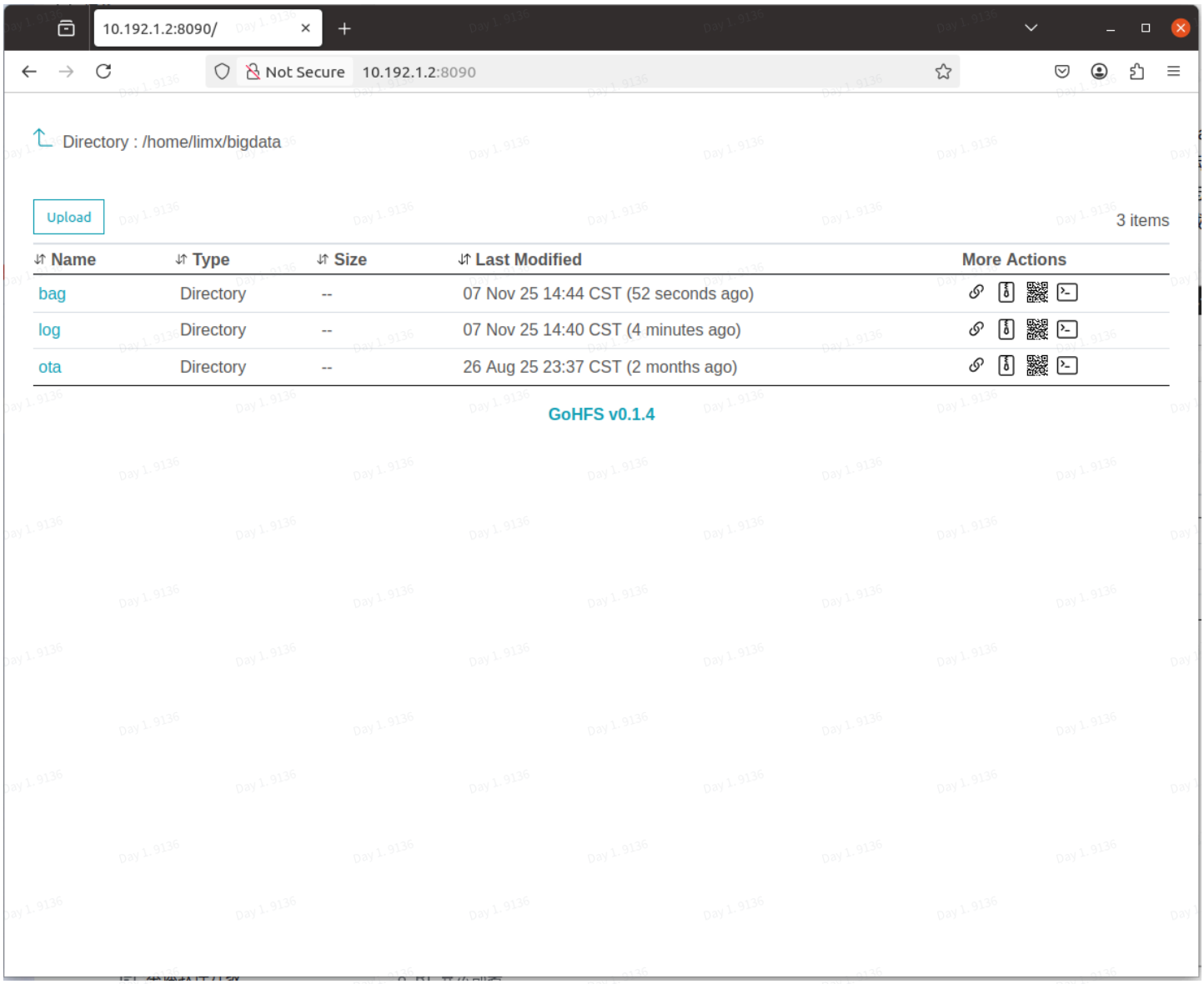
代码块

```
#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();

  // 默认机器人 IP 地址
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // 如果提供了命令行参数，则使用命令行参数作为机器人 IP 地址
    robot_ip = argv[1];
  }

  // 初始化运动控制算法程序的通信运行环境
  if (!robot->init(robot_ip))
  {
    // 如果初始化失败，则退出程序
    exit(1);
  }
```

```
27
28      // 订阅机器人状态更新，并指定回调函数
29      robot->subscribeImuData([&](const ImuDataConstPtr& msg) {
30        // 在这里处理接收到的 ImuData 数据
31        // 注意：回调函数会在收到ImuData时被调用
32      });
33
34      // 无限循环以保持程序运行
35      while (true)
36      {
37        // 休眠 1000 毫秒
38
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
39      }
40      return 0;
41  }
```

1.3 日志和数据包

机器人系统会自动录制：机器人的IMU数据(ImuData)、机器人状态数据（/joint/state）以及机器人控制数据（/joint/cmd）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。此外，机器人还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。当电脑与机器人WiFi热点连接时，可以通过浏览器输入 `http://10.192.1.2:8090` 进行访问，并从中下载这些数据。这一过程为机器人的监控、维护和调试提供了便利。

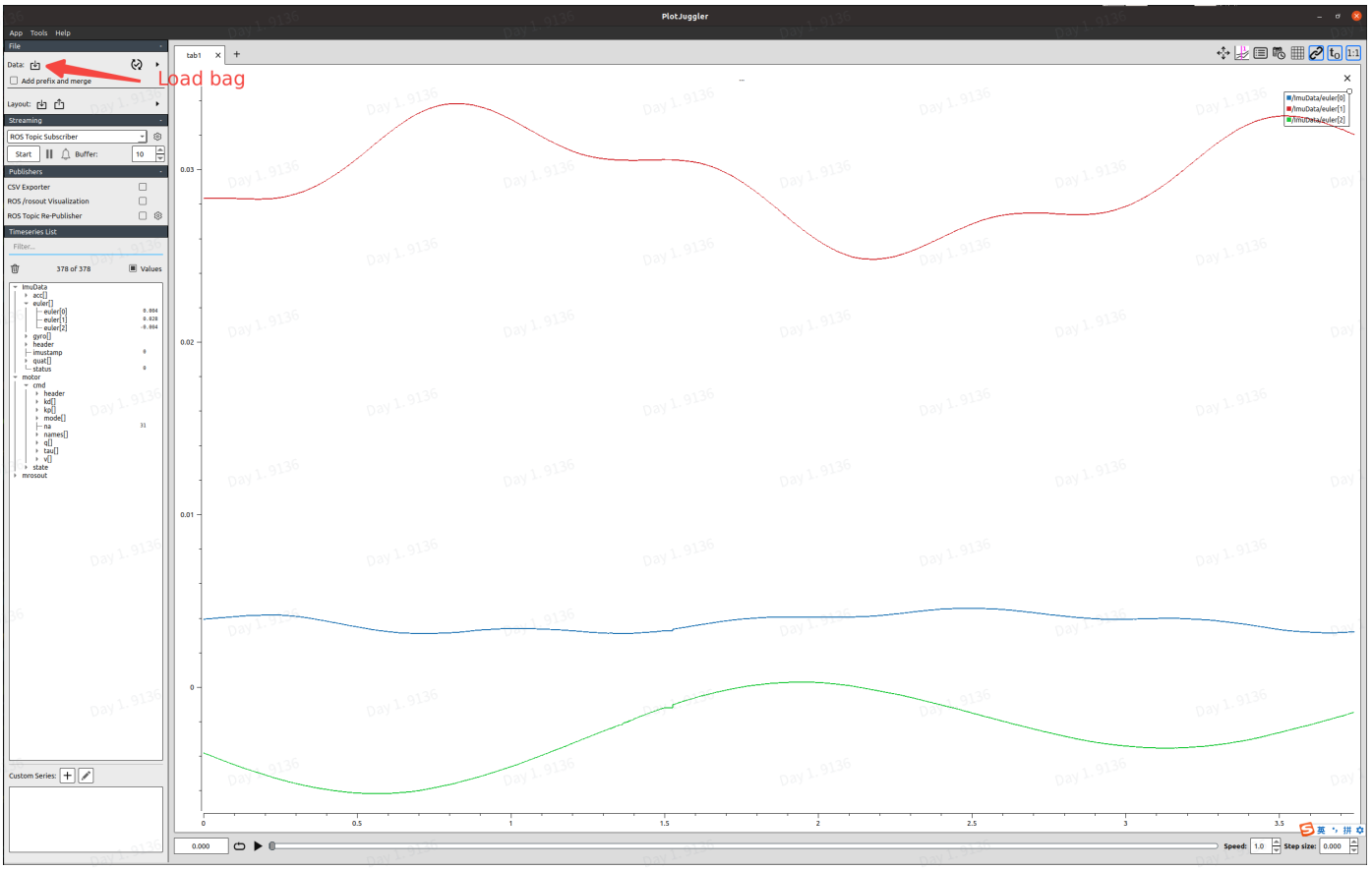


1.3.1 数据包可视化分析方法

- **数据包下载：**当您把 `.bag` 文件下载后，您可以使用PlotJuggler可视化工具加载这些包数据进行分析。特别需要注意的是，如果您下载的是 `.bag.active` 文件，您需要使用如下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。

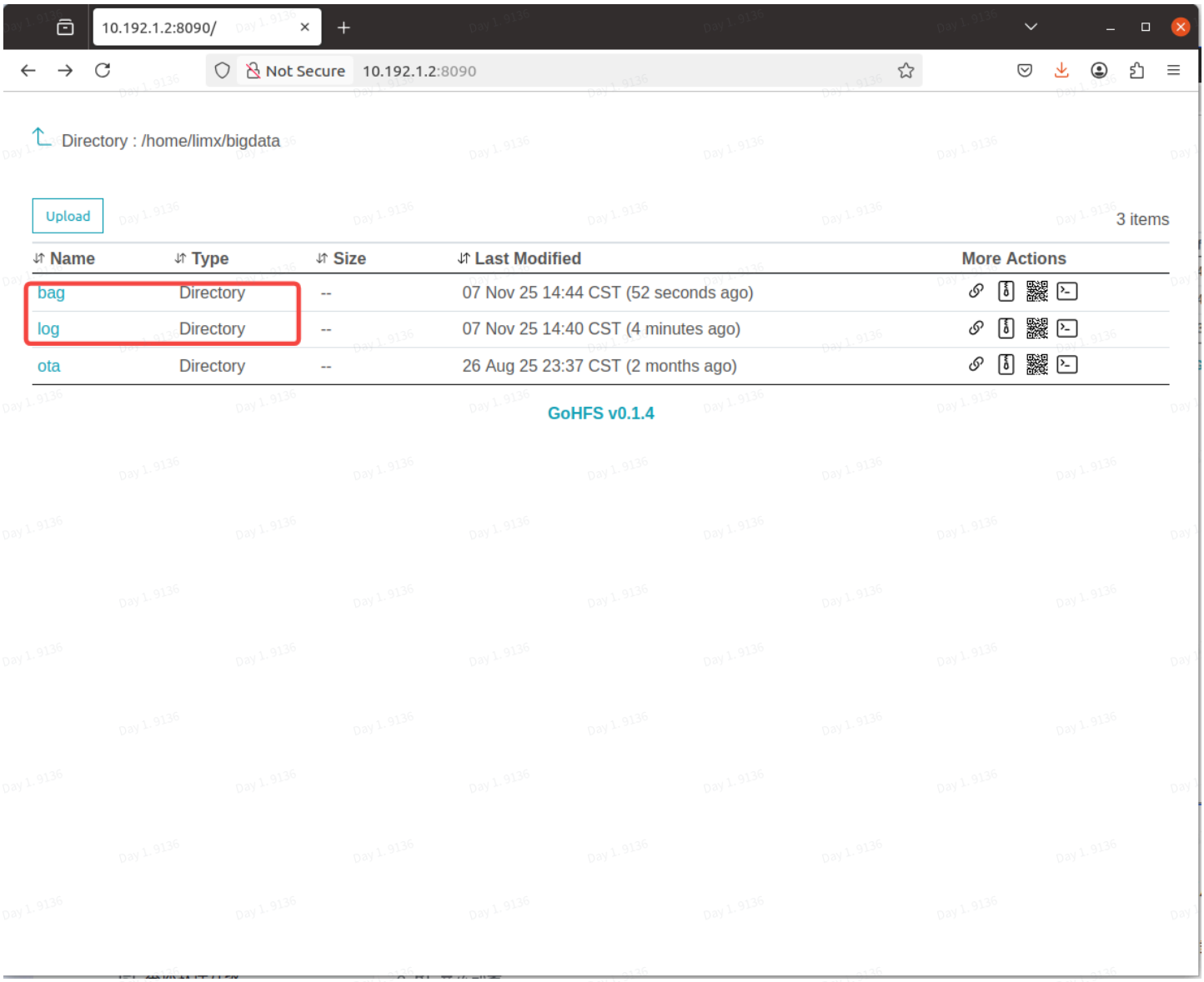
```
代码块
1   rosbag reindex your_file.bag.active
2   mv your_file.bag.active your_file.bag
```

- **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并分析数据。



1.3.2 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。4



1.4 机器人软件升级

我们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：

- 请选择并连接您机器人的 Wi-Fi 热点，密码为：`12345678`

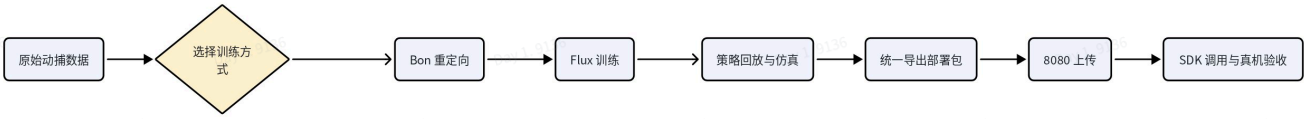
1. 访问管理页面：
   - 在浏览器地址栏输入：http://10.192.1.2:8080 进入机器人管理页面。
2. 选择和升级软件：
   - 依次选择“版本管理 -> 浏览 -> 升级”。
   - 升级完成后，机器人主控电脑将自动重启。

2. 动作训练方法参考

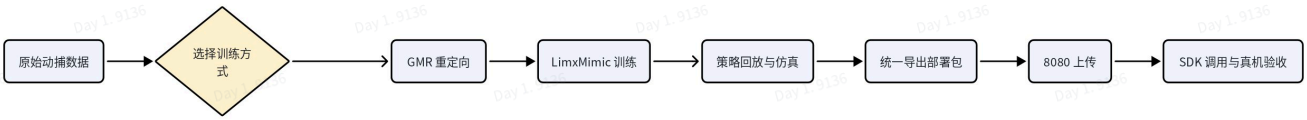
2.1 训练方式选择

方式	适用场景	主要流程	前置条件
Bon & Flux	可访问平台，希望减少本地环境搭建工作	Bon 重定向 → Flux 训练 → 云桌面/本地仿真 → 导出	Bon、Flux 账号及有效算力配额
本地离线训练	数据不能出域、平台不可达，或需要高频调参迭代	GMR 本地重定向 → LimxMimic 本地训练 → 本地仿真 → 导出	Ubuntu、NVIDIA GPU、官方仓库访问权限

云端训练链路



本地离线训练链路



> ❗ 两条路线的最终部署规范相同。训练、回放和导出必须使用同一次任务对应的动作数据与 checkpoint，不要混用不同动作、不同代码版本或不同任务配置的产物。

2.2 基于BON & Flux Training的平台训练

2.2.1 用户业务全流程

同步块 [该内容不支持导出查看]

注意：Bon未上线时可通过本地脚本完成重定向工作

2.2.2 BON & Flux Training 产品使用手册

同步块 [该内容不支持导出查看]

2.3 本地离线训练

2.3.1 电脑与软件要求

- 推荐 Ubuntu 22.04、NVIDIA 显卡和可用驱动。
- 最低建议：16 核 CPU、24 GB 显存、32 GB 内存；至少预留 25 GB 磁盘空间。
- 安装 Git、Git LFS、Miniconda 或 Anaconda。
- GitHub 账号需能访问 GMR、Luna 机器人描述仓库和官方 LimxMimic 训练仓库。

> 💡 测试确认 Luna L03 与 L04 的业务训练方法一致；当前公开导出工具固定使用 HU_L03_01 Parallel Deploy Gravity 的 141 维观测与 27 维动作。任务名、机器人配置和导出工具必须来自同一版本的官方交付包，不要自行混用 L03/L04 配置。

2.3.2 下载代码并准备目录

代码块

```
cd ~
git clone --branch limx --single-branch \
  https://github.com/limx-retarget/GMR.git GMR
git clone https://github.com/limx-luna/luna-beyondmimic.git whole_body_tracking
mkdir -p ~/data
```

后续统一使用 `~/GMR` 完成重定向，使用 `~/whole_body_tracking` 完成动作准备、训练、回放和导出。原始动作文件放入 `~/data`，文件名和路径尽量使用英文且不含空格。

2.3.3 安装 GMR 环境

代码块

```
cd ~/GMR
git lfs install
git lfs pull
git config submodule.assets/luna-description.url \
  https://github.com/limx-luna/luna-description.git
git submodule update --init assets/luna-description

conda create -n gmr python=3.10 -y
conda activate gmr
python -m pip install -e .
conda install -c conda-forge libstdcxx-ng -y
```

> ❗ 若子模块出现 `Permission denied` 或 `Repository not found`，说明账号缺少 Luna 描述仓库权限。应先申请权限，不要跳过子模块或复制不明版本的资产文件。

2.3.4 判断原始数据类型

输入	处理入口	关键确认项
Luna 重定向 NPY	直接预览和训练	必须由官方 GMR/Bon 流程生成，不是任意 NumPy 文件
SMPL-X NPZ	`smplx_to_robot.py`	pose、root、trans 和帧率字段完整
普通 BVH	`bvh_to_robot.py`	确认 lafan1、nokov、fzmotion、soma 或 noitom 格式及真实帧率
Xsens 3ds Max BVH	`xsens_bvh_to_robot.py`	确认位移单位；厘米用 0.01，毫米用 0.001
视频、FBX、CSV 等	先转换	当前官方流程不能直接读取

2.3.5 重定向为 Luna 动作

以下命令只执行与数据类型对应的一种：

代码块

```
conda activate gmr
cd ~/GMR
mkdir -p output

# 普通 BVH 示例：format 与 motion_fps 按真实数据修改
python scripts/bvh_to_robot.py \
  --bvh_file ~/data/walk.bvh \
  --format nokov \
  --motion_fps 50 \
  --robot limx_luna \
  --save_path output/my_motion.npy

# Xsens 3ds Max BVH 示例
python scripts/xsens_bvh_to_robot.py \
  --bvh_file ~/data/walk_xsens.bvh \
  --robot limx_luna \
  --bvh_format 3DSM \
  --scale 0.01 \
  --reset_to_zero \
```

```
20    --save_path output/my_motion.npy
```

不知道 BVH 类型时，应先向数据提供方确认采集设备与导出模板。格式、帧率或单位选错，常表现为机器人横向移动、身体离地、手脚方向异常。

2.3.6 预览重定向动作

代码块

```
conda activate gmr
cd ~/GMR
python scripts/vis_robot_motion.py \
  --robot limx_luna \
  --robot_motion_path output/my_motion.npy
```

确认整体方向、四肢姿态、脚底接触和动作尾部没有明显异常。纯 SSH 环境可暂时跳过预览，但必须在训练后补做仿真验证。

2.3.7 安装 LimxMimic 训练环境

代码块

```
conda create -n limxmimic python=3.10.15 -y
conda env config vars set PYTHONNOUSERSITE=1 -n limxmimic
conda env config vars set OMNI_KIT_ACCEPT_EULA=Y -n limxmimic
conda activate limxmimic

python -m pip install --upgrade pip
python -m pip install "setuptools==75.8.0" wheel
python -m pip install torch==2.5.1 torchvision==0.20.1 \
  --index-url https://download.pytorch.org/whl/cu118
python -m pip install --no-cache-dir "isaacsim[all,extscache]==4.5.0" \
  --extra-index-url https://pypi.nvidia.com

cd ~
git clone --branch v2.1.0 --depth 1 \
  https://github.com/isaac-sim/IsaacLab.git IsaacLab-2.1.0
cd ~/IsaacLab-2.1.0
echo "setuptools<81" > ~/isaaclab-build-constraints.txt
export PIP_CONSTRAINT=~/isaaclab-build-constraints.txt
export TERM=xterm
./isaaclab.sh --install

cd ~/whole_body_tracking
git submodule update --init --recursive
```

```
python -m pip install -e source/whole_body_tracking
```

GMR 与 LimxMimic 使用两个独立 Conda 环境。重定向命令在 `gmr` 中运行，训练与导出命令在 `limxmimic` 中运行。

2.3.8 训练前检查

代码块

```
conda activate limxmimic
cd ~/whole_body_tracking
python scripts/prepare_motion.py \
  --input ~/GMR/output/my_motion.npy \
  --output motions/my_motion_parallel_tail10.npz \
  --robot hu_l03_parallel \
  --linkage_report motions/my_motion_linkage_report.json \
  --force_tail
```

该步骤用于显式检查并联求解、尾帧和运动学结果。生成的 NPZ 是诊断中间文件；当前推荐训练、回放和组合导出仍统一传入同一份可信 NPY，由工具内部执行一致的动作准备流程。

2.3.9 启动训练

代码块

```
conda activate limxmimic
cd ~/whole_body_tracking
python scripts/rsl_rl/train.py \
  --task=Tracking-Flat-HU-L03-Parallel-Deploy-Gravity-v0 \
  --motion_file ~/GMR/output/my_motion.npy \
  --motion_force_tail \
  --num_envs 4096 \
  --headless \
  --max_iterations 15000 \
  --logger tensorboard
```

终端出现训练目录并持续更新迭代信息，即表示训练已正常启动。显存不足时，将 `--num_envs` 依次降为 2048 或 1024，不要修改任务名和观测/动作维度。

2.3.10 回放与导出

代码块

```
# 回放同一次训练的 checkpoint
```

```
python scripts/rsl_rl/play.py \
  --task=Tracking-Flat-HU-L03-Parallel-Deploy-Gravity-v0 \
  --checkpoint /path/to/model_14999.pt \
  --motion_file ~/GMR/output/my_motion.npy \
  --motion_force_tail \
  --num_envs 1 \
  env.commands.motion.debug_vis=false \
  env.scene.contact_forces.debug_vis=false

# 导出完整部署包
python tools/offline_onnx_exporter/export_policy_and_reference.py \
  --checkpoint /path/to/model_14999.pt \
  --motion ~/GMR/output/my_motion.npy \
  --output_path exported/my_motion
```

2.4 训练工程、可调参数与接口红线

2.4.1 仓库目录结构

仓库分两大块：`source/` 是安装成 Python 包的训练代码，`scripts/` 是直接执行的命令行工具。日常改配置在 `source/`，日常跑流程在 `scripts/`。

主要目录

```
whole_body_tracking/
├── scripts/                          命令行工具，直接执行
│   ├── rsl_rl/
│   │   ├── train.py                  启动训练
│   │   ├── play.py                   回放策略，顺带导出 ONNX
│   │   └── cli_args.py               训练/回放的命令行参数
│   ├── solve_parallel_linkage.py     解闭链自由度
│   ├── npy_to_npz.py                 重定向 npy 转训练用 npz
│   ├── append_motion_tail.py         给动作加保持终止姿态的尾帧
│   ├── verify_motion_npz.py          用正运动学校验 npz（硬门禁）
│   ├── dump_articulation_order.py    打印模型的关节/刚体规范顺序
│   └── replay_npz.py                 在 Isaac Sim 里直接回放参考动作
│
├── source/whole_body_tracking/whole_body_tracking/
│   ├── tasks/tracking/
│   │   ├── tracking_env_cfg.py       奖励、终止、域随机化、观测的总配置
│   │   ├── mdp/
│   │   │   ├── commands.py           参考动作加载、误差计算、自适应采样
│   │   │   ├── rewards.py            奖励函数实现
│   │   │   ├── terminations.py       终止条件实现
│   │   │   ├── events.py             域随机化实现
│   │   │   └── observations.py       观测项实现
```

```
23  │   │   └── config/hu_l03/
24  │   │       ├── parallel_env_cfg.py   闭链变体的环境配置
25  │   │       └── agents/
26  │   │           └── rsl_rl_ppo_cfg.py   PPO 超参与网络结构
27  │   ├── robots/
28  │   │   └── hu_l03_parallel.py        闭链模型的执行器增益、armature、动作缩放
29  │   ├── assets/HU_L03_description/    机器人 USD / URDF / MJCF
30  │   └── utils/exporter.py             ONNX 导出与元数据
31  │
32  ├── motions/                          转换好的 npz 参考动作
33  ├── logs/rsl_rl/                      训练产物：checkpoint、TensorBoard、导出的
    ONNX
34  ├── export_specs/                     离线 ONNX 导出的规格文件
35  └── tools/offline_onnx_exporter/      不依赖 Isaac Sim 的独立导出工具
```

2.4.2 配置文件速查

修改目标	对应文件
奖励、终止阈值、域随机化	`tasks/tracking/tracking_env_cfg.py`
课程采样参数	`tasks/tracking/mdp/commands.py` 的 `MotionCommandCfg`
新增一个奖励函数	`tasks/tracking/mdp/rewards.py`
PPO 超参、网络宽度	`config/hu_l03/agents/rsl_rl_ppo_cfg.py`
执行器刚度阻尼	`robots/hu_l03_parallel.py`
观测项	`config/hu_l03/parallel_env_cfg.py`，**但不要改**

2.4.3 接口红线：观测项和动作项不能改

❌ **禁止增删或调整观测项、观测顺序、动作维度和动作顺序。**这些内容构成策略与机器人下位机之间的固定接口契约，修改后策略无法正确部署。

项目	不能修改的原因
观测	

	每一项及其顺序都会写入 ONNX metadata。下位机按固定布局拼接输入向量，增删项目或改变顺序会导致输入错位。
动作	27 维动作分别对应具体电机槽位。改变宽度或顺序会使控制指令下发到错误关节。

当前任务 `Tracking-Flat-HU-L03-Parallel-Deploy-Gravity-v0` 使用 **141 维观测、27 维动作**：

```
command(54) + base_ang_vel(3) + projected_gravity(3)
            + joint_pos(27) + joint_vel(27) + actions(27) = 141
```

Deploy 表示已移除下位机无法直接获得的全局位置、状态估计线速度和无传感器连杆自由度；**Gravity** 表示保留 IMU 可直接提供的重力方向。

2.4.4 奖励参数

奖励是相对安全的调节入口，定义在 `tracking_env_cfg.py` 的 `RewardsCfg`。

奖励项	权重	std	调大后的主要影响
motion_global_anchor_pos	0.5	0.3	躯干世界位置贴得更紧，但会牺牲局部姿态自由度
motion_global_anchor_ori	0.5	0.4	躯干朝向更准确
motion_body_pos	1.0	0.3	四肢位置更准确，是动作相似度的主要约束
motion_body_ori	1.0	0.4	四肢朝向更准确
motion_body_lin_vel	1.0	1.0	线速度和动作节奏更贴近参考
motion_body_ang_vel	1.0	3.14	转动节奏更贴近参考
action_rate_l2	−0.1	—	动作更平滑、抖动更少，但响应会变慢
joint_limit	−10.0	—	更强地避开关节限位，可能与贴近限位的参考动作冲突
undesired_contacts	−0.1	—	减少脚和手以外部位触地

> 💡 **std 通常比权重更值得优先调整。**跟踪奖励采用 `exp(-error² / std²)` 形式；std 越小约束越严格，std 越大容忍范围越高。六个 `motion_*` 项定义了任务本身，不建议删除。

2.4.5 终止条件

终止条件定义在 `TerminationsCfg`，它直接决定策略是否有机会学习完整动作。

终止项	默认阈值	含义
time_out	10 s（500 步）	正常结束，不是失败
anchor_pos	0.25 m	躯干锚点高度偏离参考的最大允许值
anchor_ori	0.8	躯干倾斜与参考的偏差
ee_body_pos	0.25 m	双踝、双腕任一末端高度偏离参考的最大允许值

可使用“阈值 ÷ 末端峰值垂直速度”估算容错窗口。例如峰值速度为 2 m/s、阈值为 0.25 m，策略只允许落后约 0.125 秒。大幅抬腿、跳跃或起身动作完成率长期偏低，且失败原因集中在 `ee_body_pos` 时，可评估将阈值放宽到 0.4–0.5 m。

2.4.6 域随机化

增大随机化可提高实机鲁棒性，但会降低训练速度和仿真跟踪精度。

参数	默认范围	作用
physics_material 静摩擦	0.3–1.6	覆盖不同地面摩擦条件
physics_material 动摩擦	0.3–1.2	影响接触后的滑动特性
physics_material 弹性	0.0–0.5	控制触地回弹
add_joint_default_pos	±0.01 rad	模拟关节零位标定误差
base_com	x ±0.025 m，y/z ±0.05 m	模拟躯干质心偏差，对平衡影响较大
push_robot 间隔	1–3 s	控制外部扰动频率
push_robot 速度	xy ±0.5 m/s，yaw ±0.78 rad/s	增大可提高抗扰能力，但会干扰精细动作

2.4.7 课程采样

课程采样定义在 `MotionCommandCfg`，用于决定每次复位从动作的哪个时间点开始。

参数	默认值	影响

adaptive_kernel_size	1	为 1 时失败只记录在当前时间点；长动作卡在固定难点时可调至 5，让采样覆盖进入难点前的过程
adaptive_uniform_ratio	0.1	保证所有时间段具有最低采样量，避免算力集中在单一难点
adaptive_alpha	0.001	控制失败直方图更新速度；调大响应更快但波动更大
pose_range / velocity_range	以配置为准	控制复位时初始状态扰动，增大可提高鲁棒性

2.4.8 PPO 超参与网络

定义在 `config/hu_l03/agents/rsl_rl_ppo_cfg.py`。默认配置适用于大多数动作，没有明确证据时不建议修改。

参数	默认值	说明
网络	[512, 256, 128]，ELU	仅在观测维度显著增加时考虑加宽；当前观测维度禁止修改
num_steps_per_env	24	每轮每个环境采样步数
learning_rate	1e-3，adaptive	实际学习率由 desired KL 反馈调度
desired_kl	0.01	主要更新步长控制参数；调小更稳定但训练更慢
entropy_coef	0.005	调大增强探索，过大会导致动作抖动
gamma / lam	0.99 / 0.95	折扣因子与 GAE 参数
max_iterations	30000	命令行可覆盖，实践中 15000 轮通常已进入收敛区间

2.4.9 执行器增益

执行器增益定义在 `robots/hu_l03_parallel.py`，由自然频率、阻尼比和厂商 MJCF 中的 armature 统一计算：

代码块

```
NATURAL_FREQ = 10 * 2 * pi    # 10 Hz
DAMPING_RATIO = 2.0

stiffness = armature * NATURAL_FREQ ** 2
damping = 2 * DAMPING_RATIO * armature * NATURAL_FREQ
```

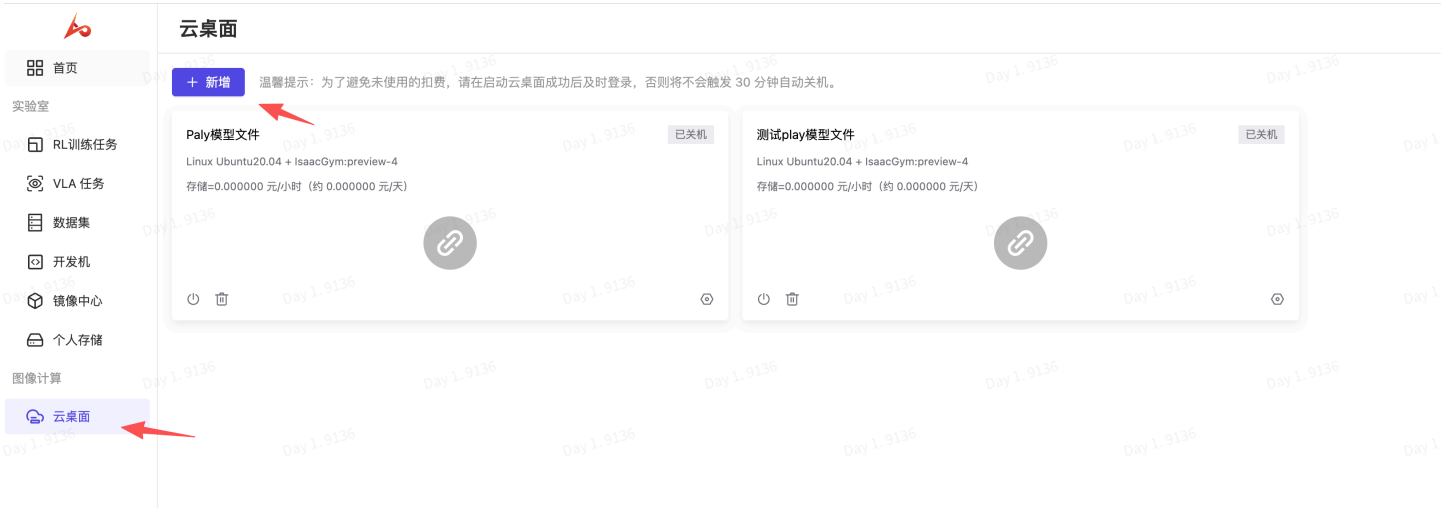
- **NATURAL_FREQ 调高**：跟踪更硬、更准确，但实机更容易产生抖动。
- **DAMPING_RATIO 调高**：系统更稳定，但响应更迟钝。
- **soft_joint_pos_limit_factor** 默认 0.9；参考动作需要接近限位时可谨慎提高。

> ❗ 调整增益应修改统一常数，不要逐关节任意改数值，以免破坏各关节之间的相对关系。任何调参结果都必须重新完成仿真和真机安全验收。

3. 舞蹈动作仿真云桌面验证

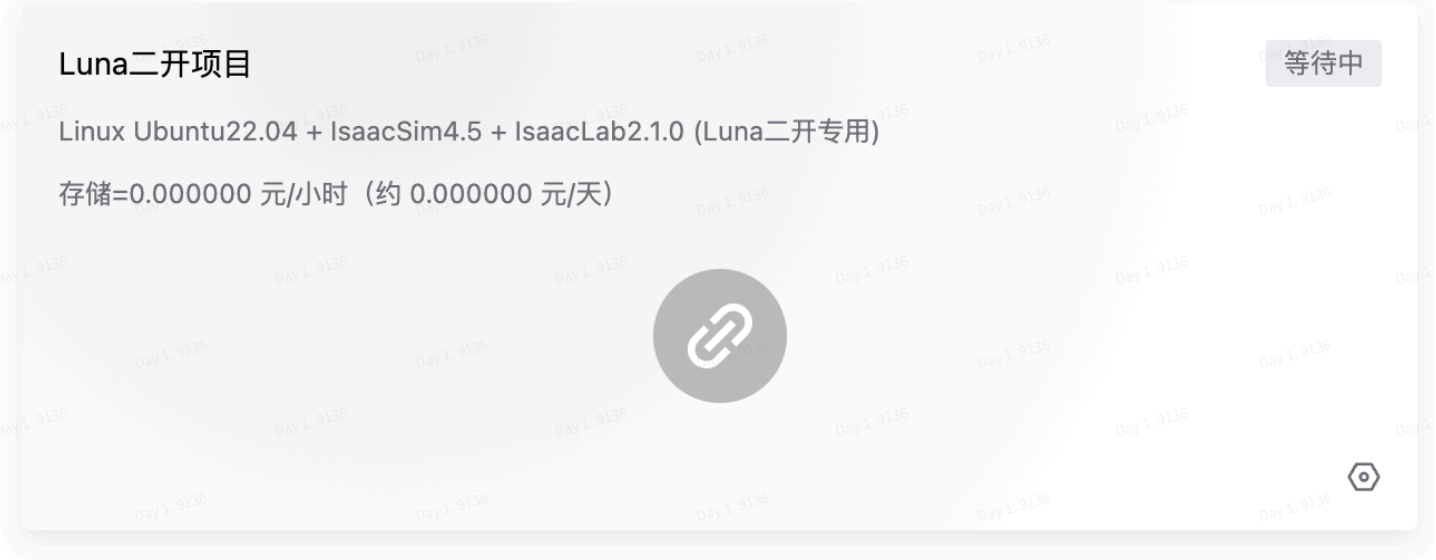
3.1 云桌面新增

1. 注册并登陆Flux Training （https://internal.limxdynamics.com/user/login）
2. 充值账户（兑换码请联系逐际动力销售同事）
3. 选择“云桌面”，点击新增



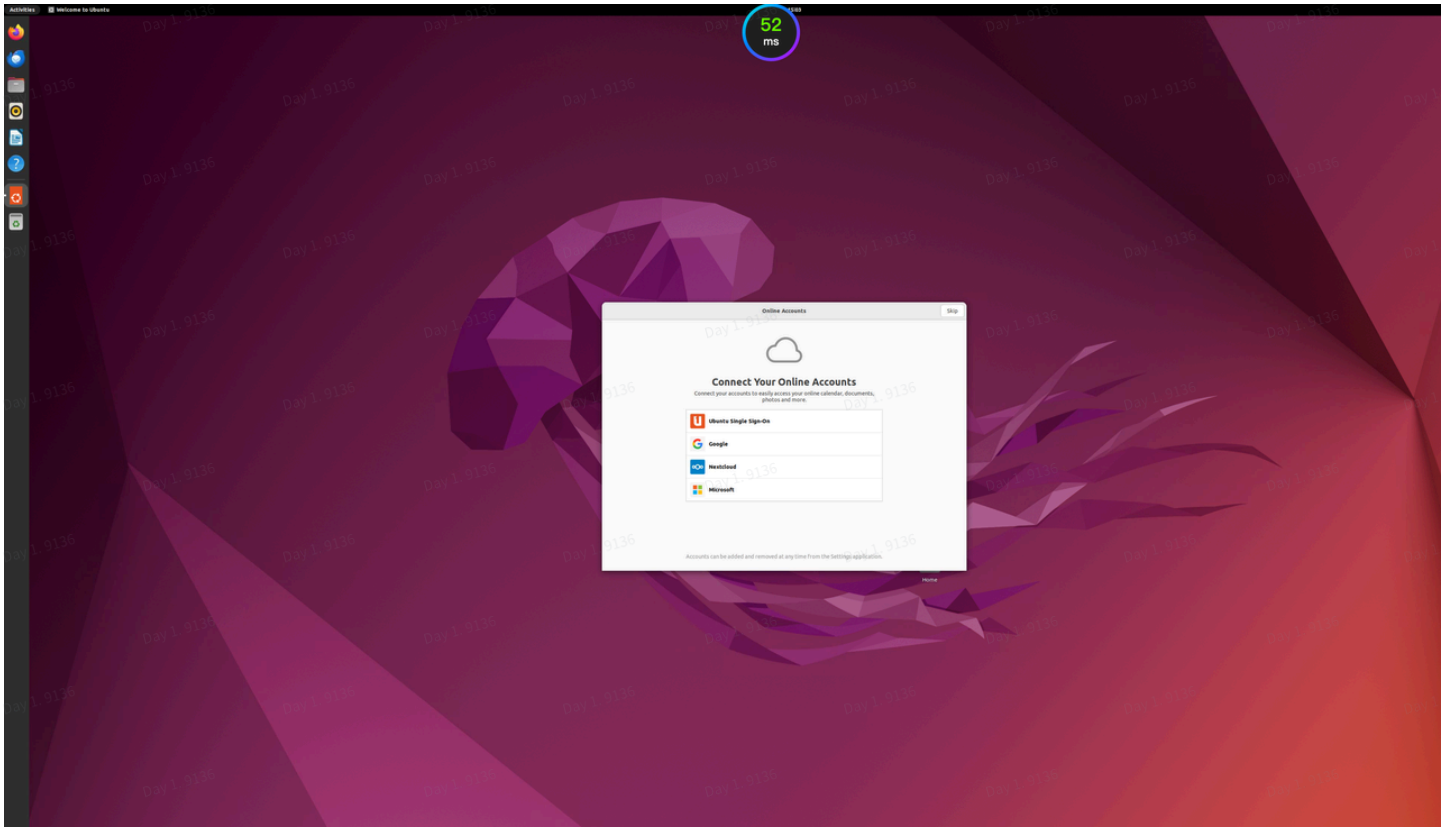
4. 新建云桌面，选择配置参数
    a. 算力：按需选择，3档可选，5880 16G/24G/48G
    b. 镜像系统：滑到底部，选择带Luna二开专用的镜像
    c. 系统盘：按需选择，2档可选，100G和200G
    d. 自动关机：按需选择，识别标准为是否移动鼠标

5. 创建并开机，等待开机



6. 点击登陆

7. 登陆使用



3.2 仿真运行

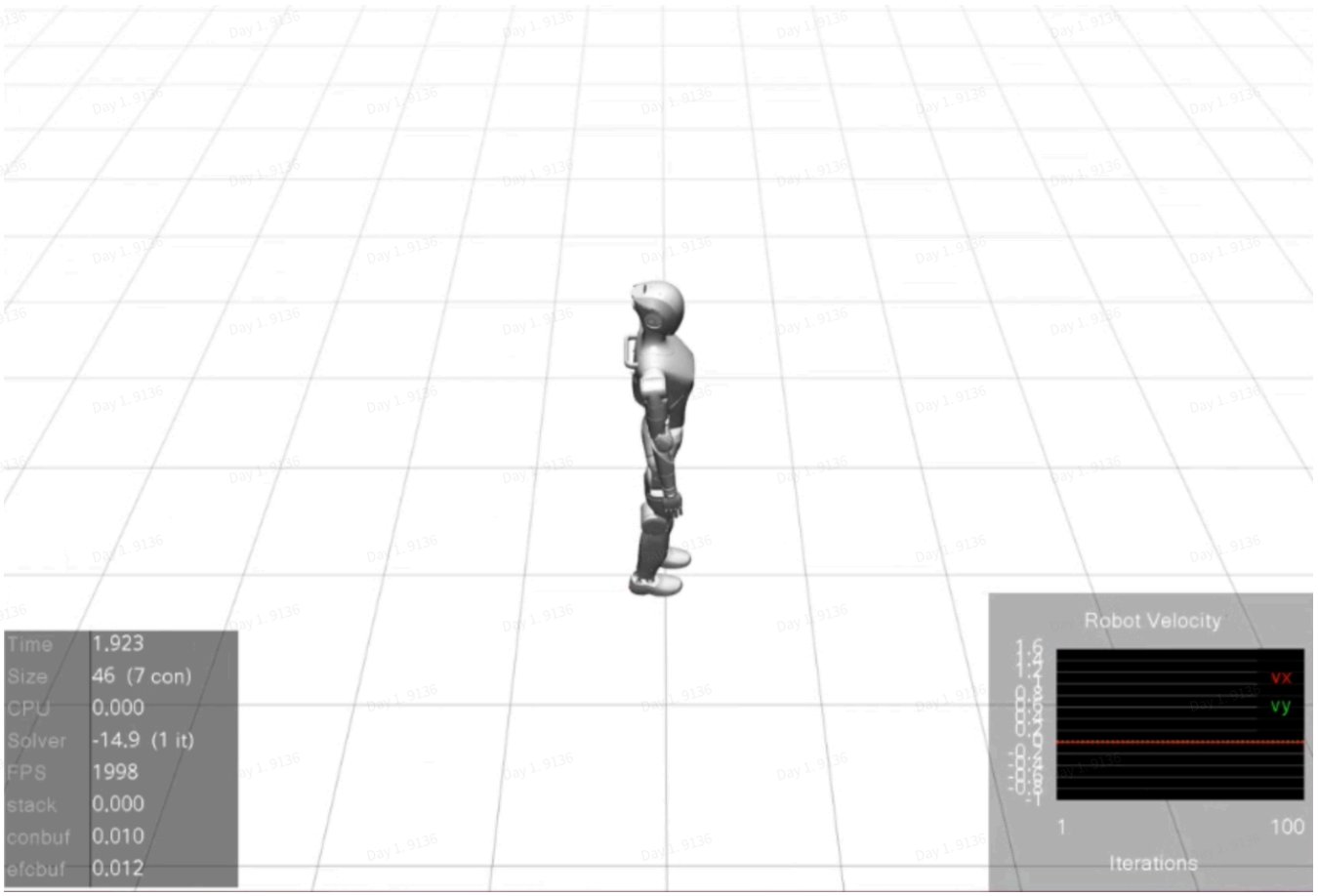
在主目录中，有个LunaSim文件夹，进入该文件夹中：

```
代码块
1   cd ~/LunaSim
```

运行一键启动仿真脚本：

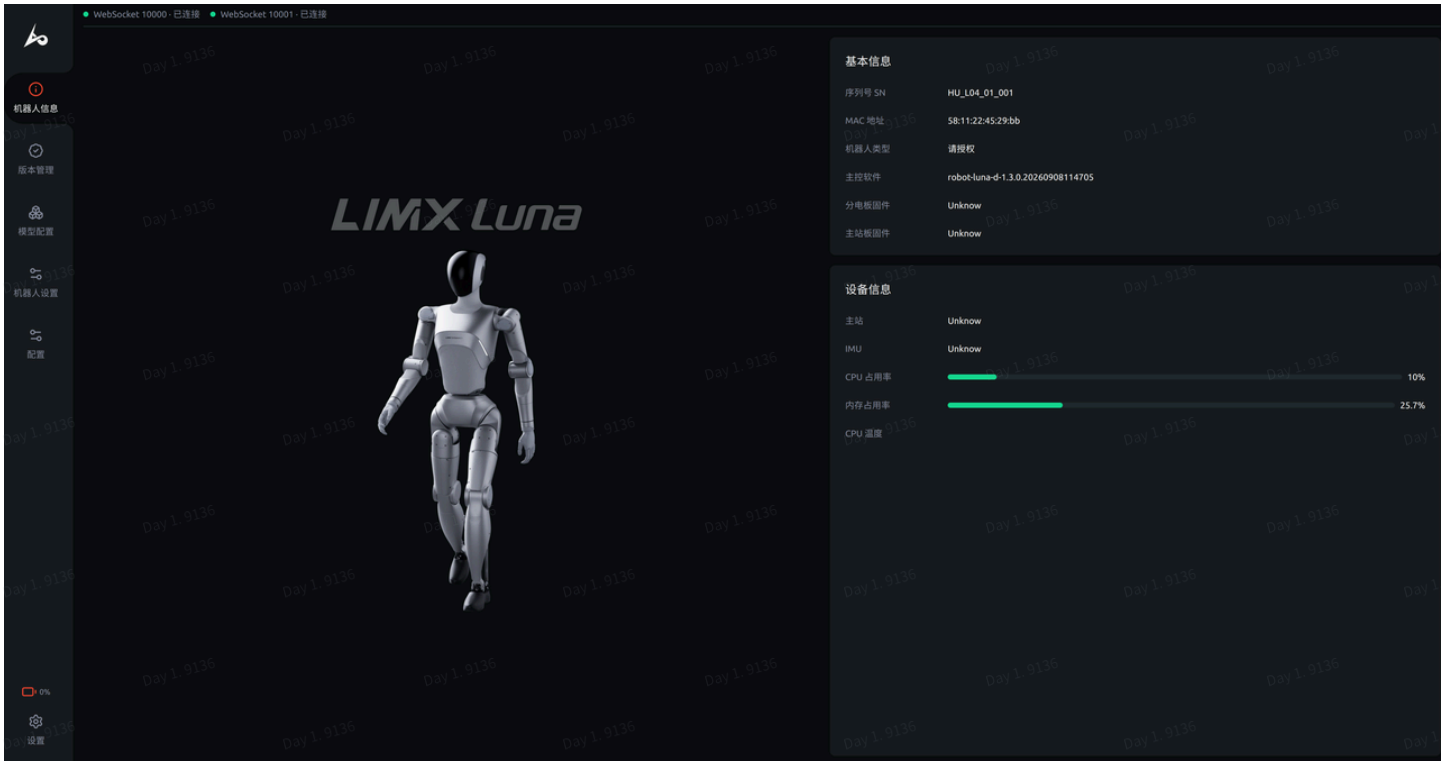
```
代码块
1   ./start_sim.sh
```

等待仿真启动，机器人站立在仿真环境中

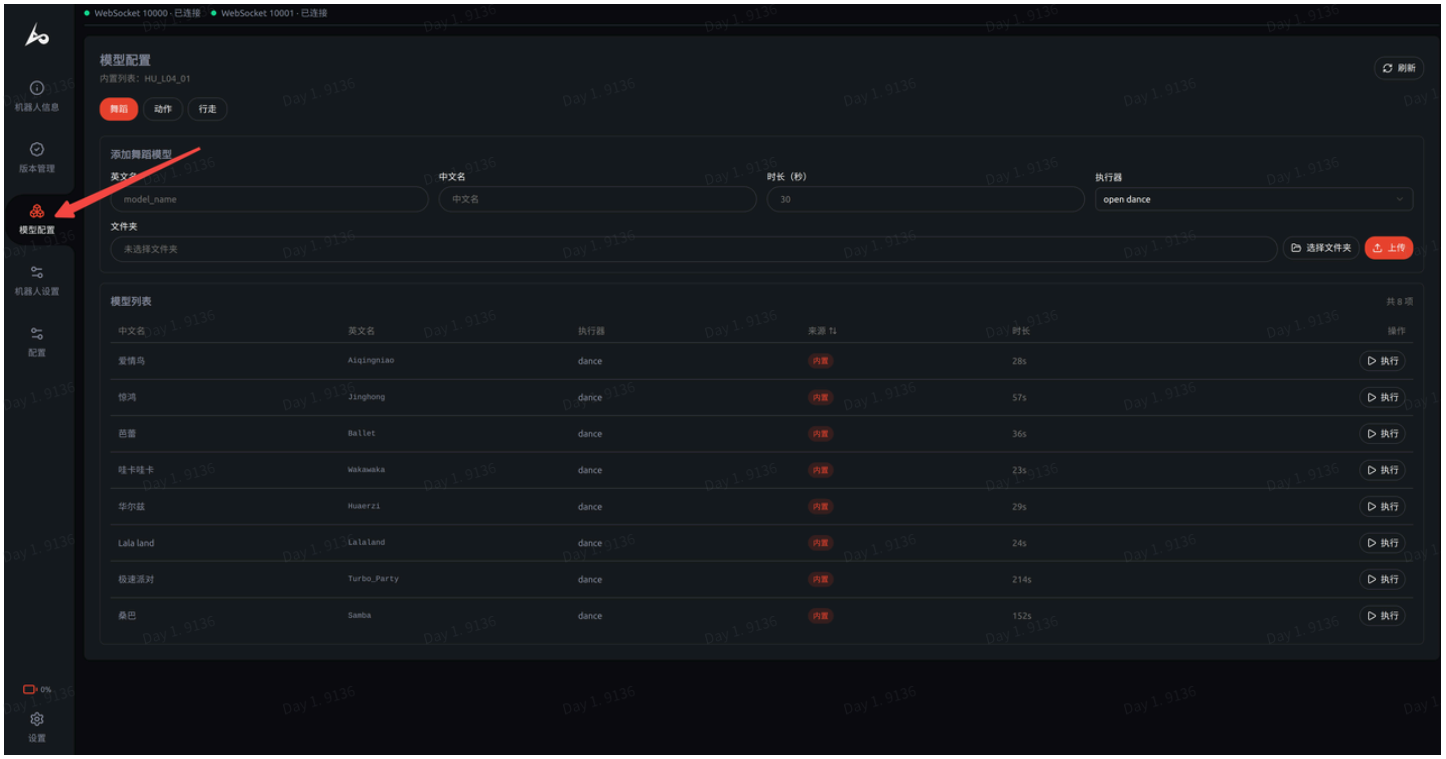


3.3 舞蹈验证

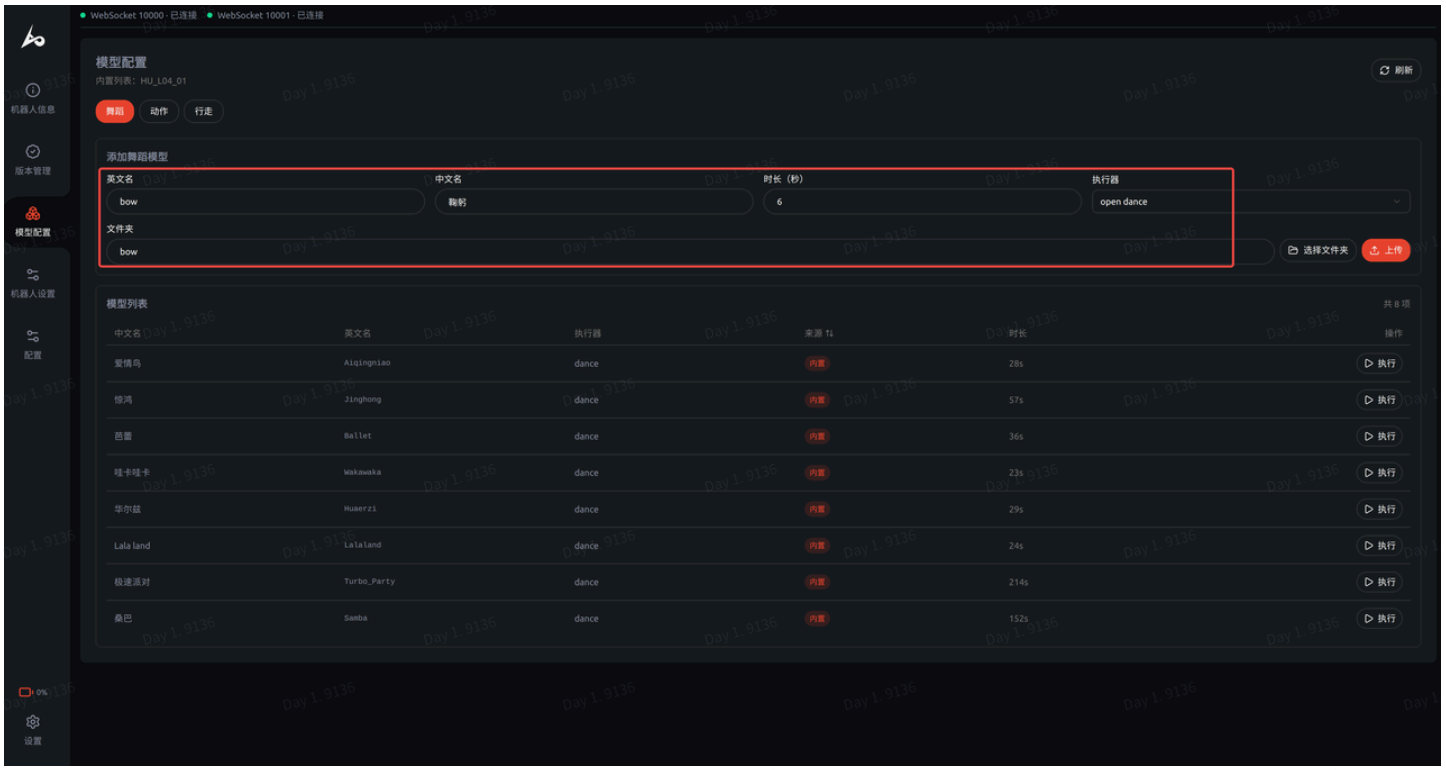
待仿真启动后，打开浏览器，在浏览器地址栏输入 `127.0.0.1:8080` ，进入机器人配置界面



观察仿真连接正常后，选择模型配置



选择“选择文件夹”，将训练好的policy.onnx，参考轨迹reference.txt以及配置文件param.yaml所在的文件夹选中，并给自己添加的动作命名，点击上传：



模型列表显示新增动作如下图，则可以点击执行，仿真中Luna将执行该动作


模型列表
共 9 项
中文名
英文名
执行器
来源
时长
操作
鞠躬
bow
open dance
添加
6s
执行
删除
爱情鸟
Aiqingniao
dance
内置
28s
执行
惊鸿
Jinghong
dance
内置
57s
执行
芭蕾
Ballet
dance
内置
36s
执行
哇卡哇卡
Wakawaka
dance
内置
23s
执行
华尔兹
Huaerzi
dance
内置
29s
执行
Lala land
Lalaland
dance
内置
24s
执行
极速派对
Turbo_Party
dance
内置
214s
执行
桑巴
Samba
dance
内置
152s
执行