具身智能开发平台 CloudRobo

SDK 参考

文档版本 01

发布日期 2026-08-05

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 R2C SDK 使用简介

R2C SDK（Robot-to-Cloud SDK）是CloudRobo平台的机器人端通信组件，负责连接机器人与CloudRobo平台云端推理服务。基于R2C协议实现实时数据传输，支持机器人端观测数据上报和云端动作指令下发，内置TLS/mTLS认证，传输编码压缩，异步推理处理，异构机器人适配以及配置驱动的适配器/翻译器架构。支持用户快速使用R2C SDK适配自有机器人接入CloudRobo平台完成端侧设备接入、数据采集、数据处理、模型训练、模型部署到模型推理的全链路具身智能业务流程。

机器人接入 CloudRobo 平台流程

用于将一台真实的机器人（如机械臂、人形机器人等）接入CloudRobo平台。整体流程如下：

1. 准备硬件设备
   准备一台上位机和一台可正常上电和通信的机器人。
2. 配置软件环境
   安装LeRobot与本项目R2C SDK所需的软件环境。
3. 接入机器人
   在CloudRobo平台完成真机创建。
4. 配置端侧
   使用R2C SDK将机器人接入CloudRobo平台。
5. 调试智能体
   真机运行任务并完成调试。

2 准备硬件设备

首先准备一台上位机和一台机器人设备。CloudRobo平台上R2C SDK已适配支持的机器人设备如下表所示。

表 2-1 已适配的设备清单

类型	厂商	型号
机械臂	开源	SO-ARM101
	节卡	Jaka Mini2
	Universal Robots	UR5e
	非夕科技	Flexiv Rizon4
	拓斯达	X5
	华沿	S05
人形机器人	星海图	GALAXEA R1
	千寻智能	Moz1
	优艾智合	隙锋
复合机器人	暂无	暂无
四足机器人	五八智能	天狼Q25
轮式机器人	暂无	暂无
其他机器人	暂无	暂无

表 2-2 上位机设备要求

主控设备	操作系统	配置	基础软件（共通）
台式机/笔记本电脑	Ubuntu 22.04+/ Windows	CPU: 6核+，推荐8核	Python 3.10+、VS Code、Git
		内存：16GB+，推荐32GB	
		显卡：核显 / NVIDIA独显（本机推理必需），8GB+显存	
		硬盘：1TB+，推荐NVMe/SSD	
		USB：至少需要4个USB，但一般设备只有2~3个USB，就需要准备1个或者2个备用USB扩展坞	

3 配置软件环境

安装 LeRobot 环境

安装LeRobot环境的主要安装步骤如下所示。

步骤1 安装Miniforge。

- Windows系统

  Windows系统下载路径：https://mirrors.tuna.tsinghua.edu.cn/github-release/conda-forge/miniforge/LatestRelease/Miniforge3-Windows-x86_64.exe

  下载完后，双击安装，按照默认选项安装即可。

- Linux系统

  Linux系统下载路径： https://mirrors.tuna.tsinghua.edu.cn/github-release/conda-forge/miniforge/LatestRelease/Miniforge3-Linux-x86_64.sh

  运行以下命令安装：

  ```
  sh Miniforge3-Linux-x86_64.sh
  source ~/.bashrc
  ```

对于后续步骤的命令，无特殊声明时，Windows系统在**Miniforge Prompt**中运行（可在已安装应用列表中找到），Linux系统在终端中运行。

步骤2 （可选）配置conda国内源。

为加快conda环境安装速度，可选择配置国内源。

- **Windows系统**

  在Miniforge Prompt中运行以下命令。

  ```
  # 添加清华大学的conda-forge镜像
  conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
  # 设置搜索时显示通道地址，方便确认是否用上了国内源
  conda config --set show_channel_urls yes
  # 移除默认的国外conda-forge通道（可选，但推荐，确保只走国内镜像）
  conda config --remove channels conda-forge
  ```

- **Linux (Ubuntu) 系统**

  打开终端，运行以下命令。

  ```
  # 添加清华大学的conda-forge镜像
  conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
  # 设置搜索时显示通道地址，方便确认是否用上了国内源
  conda config --set show_channel_urls yes
  # 移除默认的国外conda-forge通道（可选，但推荐，确保只走国内镜像）
  conda config --remove channels conda-forge
  ```


- **清除缓存使配置生效**

  修改完文件后，运行以下命令。

  ```
  conda clean -i
  ```

- **验证配置**

  运行以下命令查看当前的channels是否已更换。

  ```
  conda config --show channels
  # 输出应该包含有 https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
  ```

步骤3 创建虚拟环境。

运行如下命令，对于创建环境中conda的判定请求，按照日志提示回复accept或y。

```
conda create -n lerobot python=3.12
conda activate lerobot
```

后续再次进入环境，只需要运行 “conda activate lerobot” 即可。

步骤4 配置Python pypi国内源（可选）。

为了加快安装Python包时依赖包的下载速度，可配置pypi的国内源。

自动配置命令（跨平台通用）。

运行以下命令，即可一键将pip默认源设置为华为源。

```
pip config set global.index-url https://repo.huaweicloud.com/repository/pypi/simple
pip config set global.trusted-host repo.huaweicloud.com
```

步骤5 获取lerobot代码。

- **方式一**：如果本地有配置git，可以直接克隆代码库。如果没有git，可先自行安装git，或采用方式二。

  说明

  克隆保存的路径，不能有中文字符，否则可能导致运行时找不到Python库的路径。

LeRobot的官方代码仓库更新频繁，选择tag为0.5.1版本，本文档匹配LeRobot-v0.5.1版本。

Windows系统：

```
set GIT_LFS_SKIP_SMUDGE=1
git clone https://gitee.com/mirrors/lerobot.git
cd lerobot
git checkout -b v051 v0.5.1
```

Linux系统：

```
export GIT_LFS_SKIP_SMUDGE=1
git clone https://gitee.com/mirrors/lerobot.git
cd lerobot
git checkout -b v051 v0.5.1
```

- **方式二**：在浏览器打开**lerobot仓库**，并下载v0.5.1版本。将压缩包放到个人工作目录下并解压。

步骤6 进入lerobot目录，安装lerobot的依赖包并指定添加飞特舵机相关的驱动。

```
pip install -e ".[feetech]"
```

步骤7 安装ffmpeg。

- Linux下需要安装，运行以下命令。

  ```
  conda install ffmpeg -c conda-forge
  ```

- Windows系统不需要安装。

----结束

安装 R2C SDK 软件环境

步骤1 登录CloudRobo控制台。

步骤2 在左侧导航栏选择“运行管理 > R2C SDK”，下载R2C SDK软件包，如hw_r2c_sdk-*0.1.33*.tar.gz，并放至工作目录。

步骤3 进入conda的lerobot环境。

```
conda activate lerobot
```

步骤4 在工作目录下，解压软件包。

```
tar -zxvf hw_r2c_sdk-0.1.33.tar.gz
cd r2c_sdk_python
```

步骤5 安装R2C SDK软件包。

```
pip install -e .
```

步骤6 验证安装：输出有版本号（如*0.1.33*）即表示安装成功。

Linux下运行：

```
python3 -c "import r2c_sdk; print(r2c_sdk.__version__)"
```

Windows下运行：

```
python -c "import r2c_sdk; print(r2c_sdk.__version__)"
```

----结束

机器人标定

本章节以标定SO-ARM101机械臂为例，其他型号机械臂标定可参考官方文档。

关键标定步骤如下：

步骤1 手动标定Follower臂。

确保Follower臂连接电源和USB信号线之后，请运行如下命令查看并记录端口号。

```
lerobot-find-port
```

请根据界面提示拔出Follower臂USB信号线并在界面单击“Enter”键，可查看到Follower臂的端口号。

运行如下命令启动标定。

```
lerobot-calibrate  --robot.type=so101_follower  --robot.port=/dev/ttyACM0  --robot.id=my_follower_arm
```

说明

- robot.type=so101_follower机械臂类型，可根据需要修改。
- robot.port=/dev/ttyACM0修改为查询到的端口号，Linux一般默认为/dev/ttyACM0，Windows一般为COM1，命令需要修改为robot.port=COM1。
- robot.id=my_follower_arm可根据需要修改，保持唯一即可。

首先是Middle Position，然后是各关节都旋转到最大最小限位。

表 3-1 Middle position

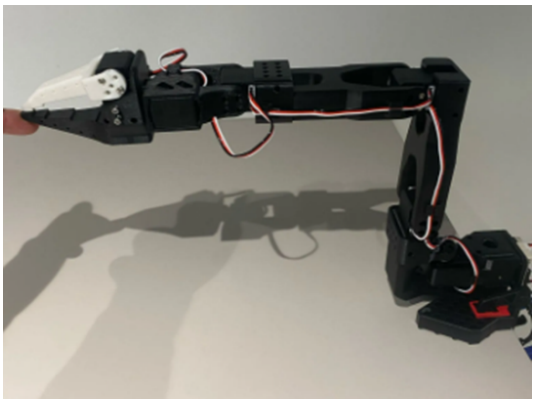
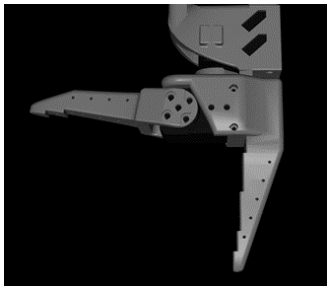
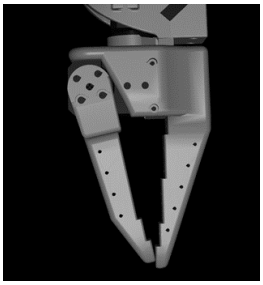
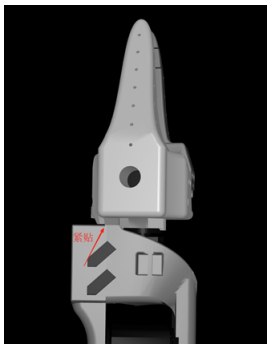
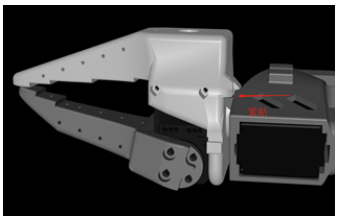
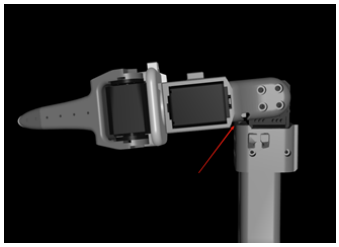
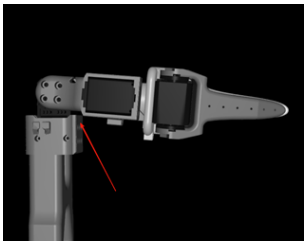
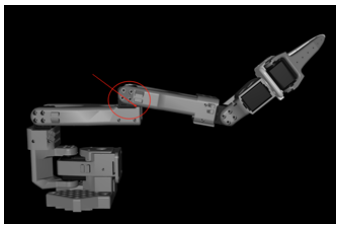
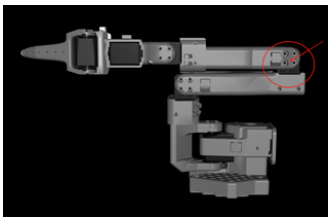
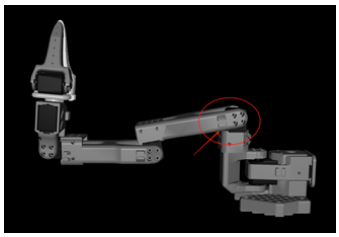
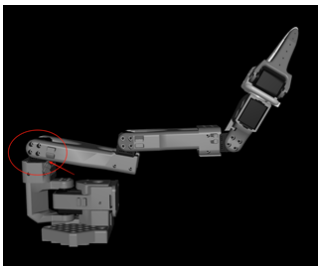
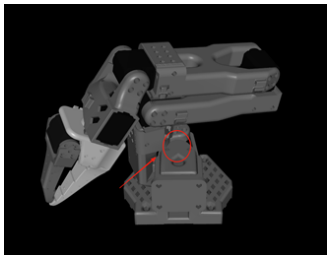
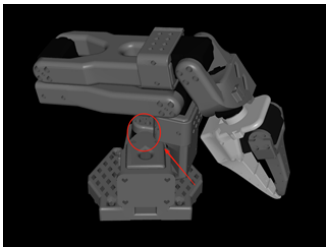
关节角	中间角度示例
Middle position	命令行窗口输出“Move so101_follower to the middle of its range of motion and press ENTER....” 后，机械臂摆成如图所示形态（动爪朝上，固定爪朝下），再按回车键。 

表 3-2 各关节最大最小张开角度

各关节限位	最大张开角度	最小张开角度
6号关节		
5号关节		

各关节限位	最大张开角度	最小张开角度
4号关节		
3号关节		
2号关节		
1号关节		

最终的参数大概在这个范围，当MIN、MAX达到最终状态不再变化之后，按回车保存。

图 3-1 标定参数范围

```
-------------------------------------------
NAME            |   MIN |   POS |   MAX
shoulder_pan    |   746 |  2035 |  3467
shoulder_lift   |   917 |   926 |  3296
elbow_flex      |   796 |  3009 |  3019
wrist_flex      |   833 |  2848 |  3180
gripper         |  2026 |  2059 |  3530
```

标定好的参数文件保存：~/.cache/huggingface/lerobot/calibration/robots/so_follower/my_follower_arm.json

步骤2 手动标定Leader臂。

请勿断开Follower臂，并确保Leader臂连接电源和USB信号线之后，请运行如下命令查看并记录端口号。

```
lerobot-find-port
```

请根据界面提示拔出Leader臂USB信号线并在界面单击“Enter”键，可查看到Leader臂的端口号。

运行如下命令启动标定。

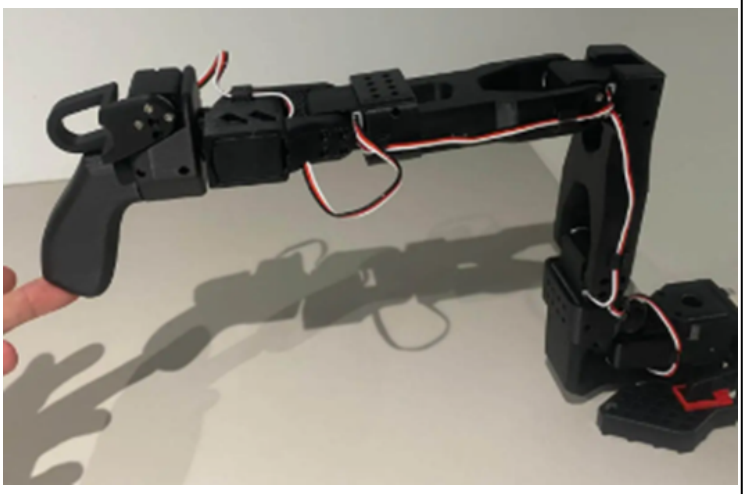
```
lerobot-calibrate --teleop.type=so101_leader --teleop.port=/dev/ttyACM1 --teleop.id=my_leader_arm
```

说明

- teleop.type=so101_leader机械臂类型，可根据需要修改
- teleop.port=/dev/ttyACM1修改为查询到的端口号，Linux一般默认为/dev/ttyACM1，Windows一般为COM2，命令需要修改为teleop.port=COM2
- teleop.id=my_leader_arm可根据需要修改，保持唯一即可

首先是Middle Position，然后是各关节都旋转到最大最小限位。(张开角度与Follower类似)

表 3-3 Middle position

关节角	角度示例
Middle position	命令行窗口输出 “Move so101_leader to the middle of its range of motion and press ENTER....” 之后，机械臂摆成如图所示形态，再按回车键。 **图 3-2** Middle position 

标定好的参数文件保存为： ~/.cache/huggingface/lerobot/calibration/teleoperators/so_leader/my_leader_arm.json

----结束

4 接入机器人

步骤1 登录CloudRobo控制台。

步骤2 在左侧导航栏单击“运行管理 > 机器人”，进入机器人管理界面。

步骤3 单击“接入机器人”，填写机器人名称、描述，选择机器人类型、厂家、型号，单击“立即接入”。

步骤4 创建完成后会弹出下载接入配置文件提示。

图 4-1 下载接入配置文件提示

- 勾选开启“私钥加密密码”，输入“加密密码”和“确认密码”后，单击“下载”按钮，将配置文件保存到本地。
  - 加密密码校验规则：支持使用英文大小写字母、数字、特殊字符（例如,.+-_#）等，长度为1~32个字符。
  - 确认密码校验规则：支持使用英文大小写字母、数字、特殊字符（例如,.+-_#）等，长度为1~32个字符。

- 可单击“返回机器人列表”按钮，返回到列表页。

----结束

5 配置端侧

R2C SDK 已适配本体

当前R2C SDK已内置有SO-ARM101等机械臂的相关配置，请将在CloudRobo平台下载的配置文件放置到r2c_sdk_python/config目录下，可快速接入机械臂。

步骤1 进入r2c_sdk_python目录下。

步骤2 进入conda的lerobot环境。

```
conda activate lerobot
```

步骤3 配置so101机械臂参数。

- 查看Follower臂的port并记录。

  ```
  lerobot-find-port
  ```

  请根据界面提示拔出Follower臂USB信号线并在界面单击“Enter”键，可查看到Follower臂的端口号。

  Linux系统下，端口类似：

  ```
  /dev/ttyACM0
  ```

  Windows系统下，端口类似：

  ```
  COM1
  ```

- 查看相机的设备号并记录。在当前运行命令的路径下会生成outputs/captured_images文件夹，可查看图片确定相机对应端口号。

  ```
  lerobot-find-cameras opencv
  ```

  Linux系统下，端口类似：

  ```
  /dev/video0
  ```

  Windows系统下，端口类似：

  ```
  0
  ```

- 在R2C SDK主目录下，打开机器人配置文件config/robot_so101_lerobot_config.yaml，修改机器人当前相机设备号和so101从臂的id和port。

  Linux系统下：

  ```
  hardware:
    lerobot_config:
      type: so101_follower
      id: my_follower_arm          --修改成标定时设备的机械臂id
      calibration_dir: null
      # Replace with the actual serial device used by your SO101 robot.
     port: /dev/ttyACM0            --修改成查到的port
      cameras:
  ```



```
    front:
      type: opencv
      index_or_path: /dev/video0       --修改成查到顶部相机的设备号
      width: 640
      height: 480
      fps: 30       --修改为相机实际的fps,如不知道或没显示就保持默认30
    wrist:
      type: opencv
      index_or_path: /dev/video2         --修改成查到的腕部相机的设备号
      width: 640
      height: 480
      fps: 30       --修改为相机实际的fps,如不知道或没显示就保持默认30
```

Windows系统下：

```
hardware:
  lerobot_config:
    type: so101_follower
    id: my_follower_arm          --修改成标定时设备的机械臂id
    calibration_dir: null
    # Replace with the actual serial device used by your SO101 robot.
    port: COM1           --修改成查到的port
    cameras:
      front:
        type: opencv
        index_or_path: 0      --修改成查到顶部相机的设备号
        width: 640
        height: 480
        fps: 30       --修改为相机实际的fps,如不知道或没显示就保持默认30
      wrist:
        type: opencv
        index_or_path: 2         --修改成查到的腕部相机的设备号
        width: 640
        height: 480
        fps: 30       --修改为相机实际的fps,如不知道或没显示就保持默认30
```

步骤4 运行如下命令完成机器人接入。

```
python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_so101_lerobot_config.yaml
```

如无报错，即表示机器人接入CloudRobo平台成功，在CloudRobo平台的机器人列表页，可查看到该机器人状态变为在线。

----结束

使用 skills 适配本体

场景介绍

使用**Coding Agent**可以实现将任意机器人接入R2C。

前提条件

1. 已经安装任意**Coding Agent**（如**码道**等）。
2. Python的最小样例（包括观测获取与动作运行），并能在机器人上运行，且提供三个安全的action样例。
   - 如果无最小样例，可以使用厂商提供的SDK示例让**Coding Agent**写一个。
   - 如果无Python版本的SDK或最小样例，可以让**Coding Agent**使用python重写。

通过skills快速接入步骤

步骤1 将已下载的**R2C SDK**中的skills/r2c添加到项目级skills文件夹（项目根目录/.codeartsdoer/skills）或个人级skills文件夹(~/.codeartsdoer/skills)。

步骤2 使用接入SKILL根据最小样例生成完整的R2C接入代码。（建议显式调用技能：/r2c根据@robot_minimal_test.py的样例来适配机械臂，机械臂的型号为{厂商的SDK或用户自定义型号}）

Coding Agent新增或修改文件应包含：

- config/robot_{机械臂型号}_config.yaml。
- examples/{机械臂型号}_cloud_adapter.py。
- src/r2c_sdk/robots/{机械臂型号}_hardware_adapter.py。

步骤3 打开调试开关。

为了避免Coding Agent代码有问题导致机器人发生碰撞，在测试前请确认config/robot_{机械臂型号}_config.yaml中的dry_run字段为True。

图 5-1 dry_run 字段

```
hardware:
  type: "custom"
  class_path: "r2c_sdk.robots.flexiv_hardware_adapter.FlexivHardwareAdapter"
  custom_config:
    # Flexiv 机器人序列号
    robot_sn: "Rizon4-062833"

    dry_run: true
```

步骤4 订阅观测验证，该流程用于验证机器人能正确运行动作。

打开两个终端，并依次分别运行下列命令：

```
python examples/observation_subscriber.py --project-id test_project --device-id {机械臂型号} --client-id obs_sub  --endpoints tcp/127.0.0.1:7447 --endpoint-role  listen
python -m r2c_sdk.cloudroboclient --project-id test_project --device-id {机械臂型号} --robot-config config/robot_{机械臂型号}_config.yaml
```

步骤5 发布动作测试，该流程用于验证机器人能正确运行动作。

```
python examples/{机械臂型号}_cloud_adapter.py --project-id test_project --device-id {机械臂型号} --client-id obs_sub
python -m r2c_sdk.cloudroboclient --project-id test_project --device-id {机械臂型号} --robot-config config/robot_{机械臂型号}_config.yaml
```

步骤6 测试完成后，关闭dry_run模式，然后启动机器人客户端(接入CloudRobo平台)。

```
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config config/robot_{机械臂型号}_config.yaml
```

----结束

6 调试智能体

智能体调试说明

- 智能体调试时，模型服务需处于运行中状态，机器人需在线。
- 同一本体，支持切换不同的模型服务进行技能调试，方便您选择效果更优的模型服务。
- 部分模型仅支持运行固定的模型技能，不支持泛化技能；其他模型可运行泛化技能，您可根据需要输入需要运行的Prompt，验证模型的泛化性。

部署模型服务

步骤1 在左侧导航选择“运行管理 > 模型部署”，进入模型部署页面。

步骤2 单击“部署模型服务”，进入部署模型服务页面。

步骤3 输入基础信息，选择**部署SO101**可使用的模型，并选择资源配置。

步骤4 完成后单击“立即部署”，在模型部署页面可以单击，实时刷新模型部署进展，并等待模型服务部署完成

----结束

调试模型技能

步骤1 在左侧导航选择“运行管理 > 机器人”，进入机器人页面。

步骤2 在机器人列表，选择**在线**机器人，单击对应的“智能体调试”，进入“智能体调试”页面。

步骤3 单击“选择模型技能”，选择处于**运行中**的模型服务、模型技能，单击“确定”。

步骤4 在智能体调试页面，输入机器人运行Prompt或者保持默认技能，单击，开始技能调试。

模型推理的默认步数为60 steps。

如果已经达到最大推理的步数，但实际任务并未完成，您可重新发送Prompt，会重新运行任务。

在**参数配置**中适当调大**技能最大推理步数**。该参数调整后仅对该机器人和当前选择的模型服务生效。您还可以在模型的**r2c.json**中修改，该修改对后续部署的推理服务生效。

----结束

切换模型服务

在智能体界面底部，可单击模型服务名称，切换其他模型服务来验证模型技能。您可在调试记录中查看两个模型的运行结果。

运行泛化性技能

在选择模型服务和模型技能时，如模型技能中有泛化技能选项，表示该模型支持自定义Prompt运行泛化技能。

选择泛化技能后，在输入框中可自定义输入Prompt，验证模型的泛化性。

配置 r2c.json 文件

预置模型、空间资产中类型为感知模型和规划模型均不适用于r2c.json配置，可以直接跳过相关部分。

本章节用于描述r2c.json配置文件的结构和含义。该配置文件定义了机器人观测数据到模型输入、以及模型输出到机器人动作之间的映射关系。您可以根据需要调整r2c.json配置文件，具体请参见**r2c.json文件配置说明**。

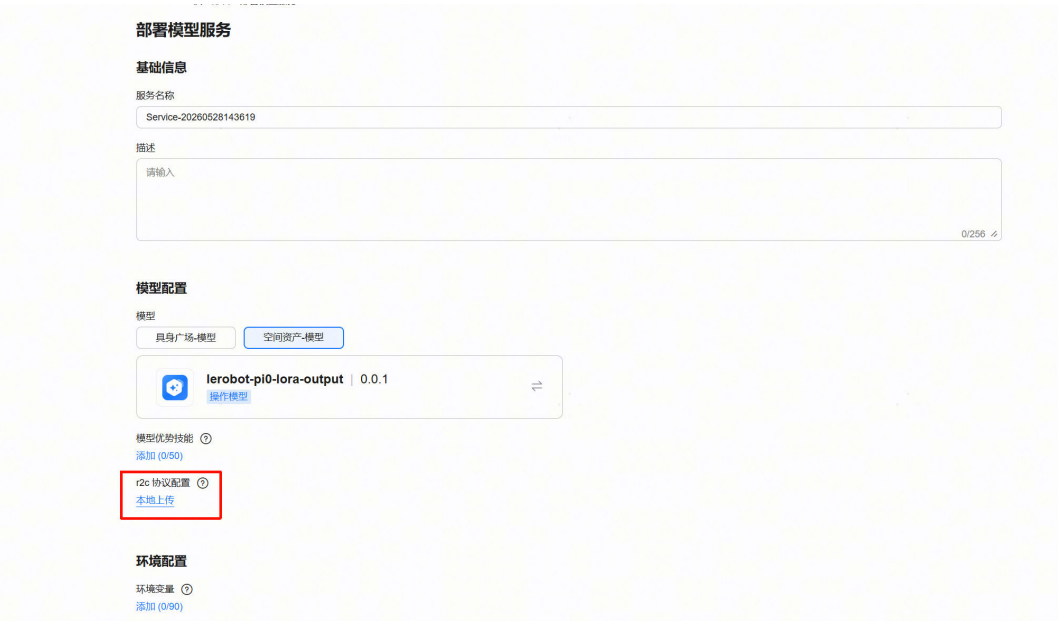
方法一：从模型部署页面中配置

说明

r2c.json仅允许部署模型时上传，部署模型后仅可查看。如果需要更改该配置，请重新部署。

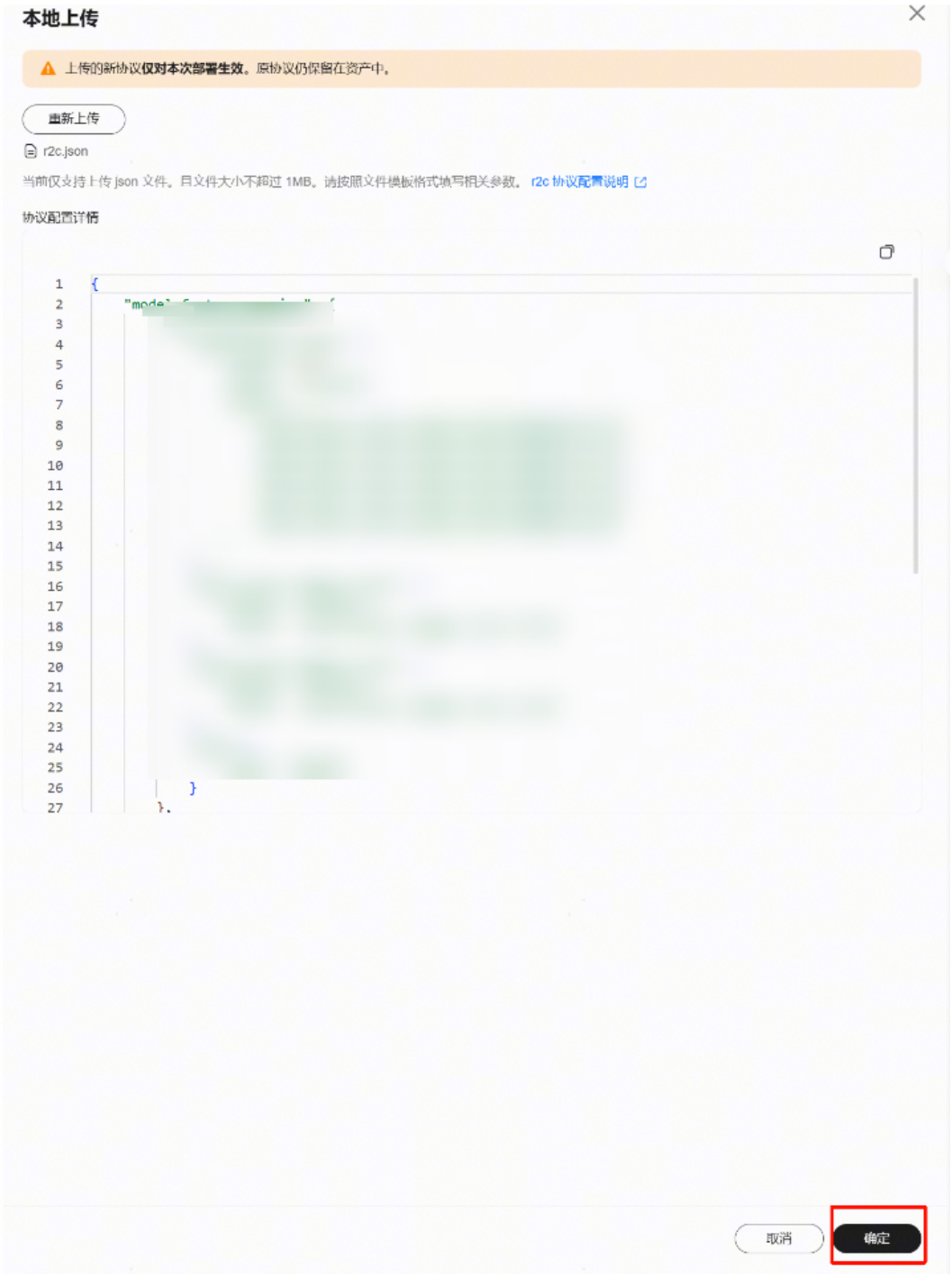
步骤1 在运行管理-模型部署中，可直接上传r2c.json配置。

图 6-1 模型部署页面



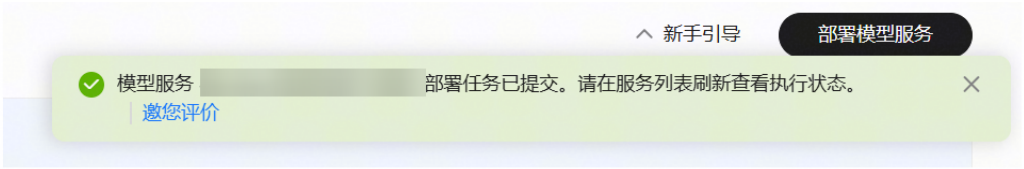
步骤2 上传后，可直接预览已上传的文件。

图 6-2 上传成功页面



步骤3 单击部署，即配置成功。如果显示r2c.json不正确，请确保上传的文件内容满足json格式，且符合配置结构要求。

图 6-3 部署成功页面



----结束

方法二：从模型资产所在的OBS中配置

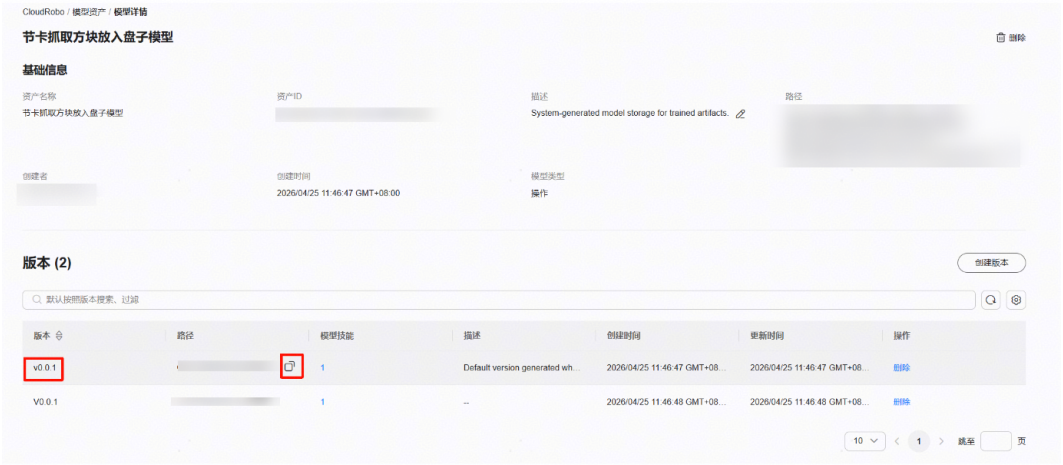
配置文件位于模型资产所在的OBS目录下。

说明

r2c.json仅在部署模型服务时会获取。如果通过OBS更新了r2c.json，已经部署的推理服务不会更新，重新启动也不会更新。因此请在部署模型服务之前，更新OBS中的r2c.json。

步骤1 在模型资产-模型详情中，复制需要部署的模型服务对应版本的obs路径。

图 6-4 复制模型资产所在的 OBS 路径



步骤2 登录OBS平台，找到对应的桶，并进入到OBS路径。

图 6-5 进入 OBS 存储模型路径

步骤3 单击上传对象，上传r2c.json文件。

图 6-6 上传文件

----结束

方法三：从数据所在的OBS中配置

配置文件位于数据资产所在的OBS目录下。

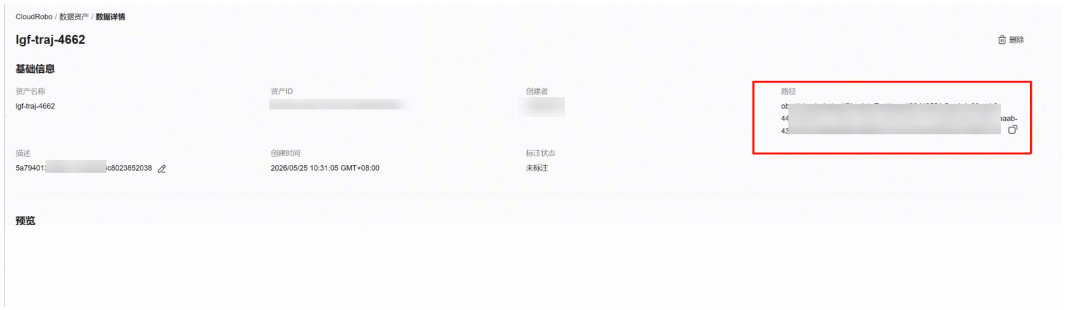
说明

r2c.json仅在训练模型终态（训练成功、手动终止、运行失败、任务失败）时获取。如果后续通过OBS更新了r2c.json，已经训练好的模型资产不会更新。因此请在训练模型结束之前，更新数据资产对应的OBS中的r2c.json。

如果数据集不存在r2c.json，而训练的源模型存在，则会在训练模型终态时将源模型的r2c.json复制到训练产物中。

步骤1 在数据资产-数据详情中，复制需要部署的模型服务对应版本的OBS路径。

图 6-7 数据资产详情页面



步骤2 登录OBS平台，找到对应的桶，并进入到OBS路径。

图 6-8 进入 OBS 存储模型路径

步骤3 单击上传对象，上传r2c.json文件。

图 6-9 上传文件

----结束

r2c.json 文件配置说明

r2c.json配置文件内容与**端侧机器人配置**（config/robot_*{机械臂型号}*_config.yaml）强相关。

编写r2c.json配置文件时请对应端侧机器人配置做映射修改。

两者映射关系：

- 机器人状态输出 > 端侧机器人配置 > r2c.json配置文件 > 推理模型输入
- 推理模型输出 > r2c.json配置文件 > 端侧机器人配置 > 机器人执行动作

r2c.json配置结构概览

```
{
   "model_feature_mapping": {
      "input_features": { ... },
      "output_features": { ... }
   },
   "stop_condition": { ... }
}
```

表 6-1 顶层字段说明

字段	类型	必填	说明
model_feature_mapping	object	是	模型特征映射配置对象。 该对象包含两个子对象**input_features**和**output_features**，分别定义模型输入和输出的映射规则，请参见**表6-2**。
stop_condition	object	否	模型运行停止条件配置，如果没有设置，则默认为{"max_iter_num": 100,"max_run_time": 10}，可通过前端修改推理服务的运行停止条件，请参见**停止条件（stop_condition）**。

表 6-2 模型特征映射（model_feature_mapping）

model_feature_mapping的子对象	说明
input_features	输入特征，该对象定义如何将机器人的观测数据映射到模型的输入格式，具体请参见**输入特征（input_features）**。
output_features	输出特征，该对象定义如何将模型输出的数据映射到机器人的动作指令，具体请参见**输出特征（output_features）**。

r2c.json配置文件示例

```
{
   "model_feature_mapping": {
      "input_features": {
         "observation.state": {
            "shape": [6],
            "values": [
               "observation.joint_states.position@{joint_1}",
               "observation.joint_states.position@{joint_2}",
               "observation.joint_states.position@{joint_3}",
               "observation.joint_states.position@{joint_4}",
               "observation.joint_states.position@{joint_5}",
               "observation.joint_states.position@{joint_6}"
            ]
         },
         "observation.images.front": {
            "value": "observations.images.color.front"
         },
         "observation.images.wrist": {
            "value": "observations.images.color.wrist"
         },
         "task":{
            "type": "PROMPT"
         }
      },
      "output_features": {
         "actions": {
            "chunk_size": 100,
            "shape": [6],
            "value": [
               "actions.joint_states.position@{joint_1}",
               "actions.joint_states.position@{joint_2}",
               "actions.joint_states.position@{joint_3}",
               "actions.joint_states.position@{joint_4}",
               "actions.joint_states.position@{joint_5}",
               "actions.joint_states.position@{joint_6}"
            ]
         }
      }
   },
   "stop_condition": {
      "max_iter_num": 100,
      "max_run_time": 10
   }
}
```

输入特征（input_features）

input_features定义如何将机器人的观测数据映射到模型的输入格式，系统支持三种类型的输入特征，如**表6-3**所示。

表 6-3 输入特征类型

特征类型	特征类型分类标准	说明
状态数组(STATE)	配置中包含values字段（必填）。	从joint_states、end_effector_states、end_effector_poses提取数值组成状态向量。 状态数组类型的详细信息请参见**状态数组类型(STATE)**。
视觉图像(VISUAL)	配置中有value字段且value路径格式符合以下规则之一。 • 4段格式：observation(s).image(s).{color|depth}.{image_name}。 • 2段格式：{color|depth}.{image_name}。	从observations.images.color.*或observations.images.depth.*提取图像数据。 视觉图像类型的详细信息请参见**视觉图像类型(VISUAL)**。
任务提示(PROMPT)	需满足以下任一条件。 • 配置中type字段值为"PROMPT"。 • dtype为"string"且value以"task"开头。 • dtype为"string"且value路径的第一段以"observation"开头，第二段以"task"开头。	从observation.task获取任务描述文本。 任务提示类型的详细请参见**任务提示类型（PROMPT）**。

- **状态数组类型 (STATE)**

  状态数组特征用于从观测数据中提取多个值组成状态向量。

  说明

  状态数组的shape和values数量必须与实际端侧机器人配置一致。

```
"observation.state": {
            "shape": [6],
            "values": [
                "observation.joint_states.position@{joint_1}",
                "observation.joint_states.position@{joint_2}",
                "observation.joint_states.position@{joint_3}",
                "observation.joint_states.position@{joint_4}",
                "observation.joint_states.position@{joint_5}",
                "observation.joint_states.position@{joint_6}"
            ]
}
```

表 6-4 状态数组类型参数说明

字段	值	说明
shape	[N]	状态数组包含N个元素（关节数 + 夹爪数）。 该字段必须存在且是一维数组（如[6]，不能是[2，3]）。
dtype	float32	使用32位浮点数表示。 该字段必须是整数或浮点类型，不能是字符串。
values	数组	按顺序提取所有关节位置和夹爪位置，顺序取决于模型推理状态数组的具体含义（请参见**表6-5**）。 该字段不能为空。

表 6-5 数据来源类型

数据源	路径格式	说明
joint_states	observation.joint_states.position@{joint_name}	关节位置
end_effector_state	observation.end_effector_states.position@{ee_name}	末端运行器/夹爪位置
end_effector_poses	observation.end_effector_poses.pose@{ee_name}	末端运行器7D位姿（tx, ty, tz, qx, qy, qz, qw），返回完整7维
end_effector_poses[index]	observation.end_effector_poses.pose@{ee_name}[0]	提取7D位姿的第index维（0-6）

数据源路径解析规则

- 使用@{name}从names数组中查找对应的索引。
- 支持[index]语法从数组中提取特定元素。
- 支持组合格式：field@{name}[index]。
- 路径必须以"observation."开头。
- 特殊：仅end_effector_poses支持使用[index]方式提取7D位姿的特定维度（0-6，对应tx, ty, tz, qx, qy, qz, qw）。

数据源路径的占位符替换

- 路径配置中的{joint_1}、{joint_2}、{gripper_1}等占位符会在运行时查找关节/夹爪名称所对应的索引的取值。
- 具体的占位符名称（如joint_1、gripper_1）来自端侧机器人配置中定义的joint_states.names和end_effector_states.names。

- 不同机器人的关节/夹爪名称可能不同，配置时需确保占位符与实际端侧机器人配置匹配。

- **视觉图像类型 (VISUAL)**

  视觉特征用于从观测数据中提取图像数据（PNG格式）。

  图像的键名（如top、wrist）取决于端侧机器人配置中定义的图像配置，不同机器人可能使用不同的相机名称。

  说明

  图像的具体名称（如top、wrist、front）需与端侧机器人配置中的images相关定义保持一致。

```
"observation.images.front": {
    "dtype": "uint8",
    "value": "observations.images.color.front"
}
```

表 6-6 视觉图像类型参数说明

字段	值	说明
dtype	uint8	无符号8位整数（0-255）。该字段必须是整数或浮点类型，不能是字符串。
value	observations.images.color.front	端侧机器人配置中的图像数据路径，请参见**表6-7**。

表 6-7 支持的图像路径格式

格式	示例	说明
完整路径（4段）	observations.images.color.front	完整观测数据路径。 4段格式路径：observations.images.{color|depth}.{image_name} • 第1段必须以observations开头。 • 第2段必须以images开头。 • 第3段必须是color或depth。 • 第4段为图像名称。
短格式（2段）	color.front	仅包含类型、图像名称。 2段格式路径：{color|depth}.{image_name}。 • 第1段必须是color或depth。 • 第2段为图像名称。

其中，路径的图像类型（第3段或第1段）必须是color或depth，支持的**图像类型**：

- color.*：彩色图像（RGB）。
- depth.*：深度图像。

- **任务提示类型（PROMPT）**

  任务提示特征用于将任务描述传递给模型。

  说明

  - 系统只允许存在一个PROMPT类型特征。
  - 配置文件只定义任务描述的**来源路径**（即从哪里获取任务），而不是静态的任务描述文本。任务描述必须从外部传入（前端）。如果为空，会报错“Task prompt not found in both skill config and observation data”，运行失败。

```
"task": {
    "dtype": "string",
    "value": "observation.task"
}
```

或：

```
"task":{
    "type": "PROMPT"
}
```

表 6-8 任务提示类型参数说明

字段	值	说明
dtype	string	字符串类型。
value	observation.task	从观测数据中提取任务描述（该字段仅记录为任务提示词，并未使用r2c中的task字段）。
type	PROMPT	表示该字段对应模型的任务描述字段（会从前端传入）。

输出特征（output_features）

output_features目前仅允许存在一个item（item的键与模型保持一致即可）。

动作输出 (action)的shape和values数量必须与实际机器人配置一致，具体关节和夹爪名称来自端侧机器人配置。

```
"actions": {
    "chunk_size": 100,
    "shape": [6],
    "value": [
        "actions.joint_states.position@{joint_1}",
        "actions.joint_states.position@{joint_2}",
        "actions.joint_states.position@{joint_3}",
        "actions.joint_states.position@{joint_4}",
        "actions.joint_states.position@{joint_5}",
        "actions.joint_states.position@{joint_6}"
        ]
}
```

表 6-9 动作输出参数

字段	值	说明
chunk_size	100	模型一次性输出100个时间步的动作。

字段	值	说明
shape	[N]	每个时间步包含N个值（关节数 + 夹爪数）。
values	数组	动作值映射到机器人关节/夹爪的路径列表，顺序取决于端侧机器人配置内names数组的顺序。

输出数据形状

实际输出形状为 [chunk_size, shape]，即chunk_size个时间步，每个时间步N个值（N = 关节数 + 夹爪数）。

动作映射关系

动作值按顺序映射到端侧机器人配置中定义的关节和夹爪，其中：

- 关节名称来自joint_states.names。
- 夹爪名称来自end_effector_poses。
- 夹爪名称来自end_effector_states.names。

停止条件（stop_condition）

该字段定义模型运行在何时停止的条件，避免无限循环或超时运行。

```
"stop_condition": {
    "max_iter_num": 100,
    "max_run_time": 10
}
```

表 6-10 停止条件参数

字段	类型	值	说明
max_iter_num	integer	100	最大推理调用次数。
max_run_time	integer	10	最大运行时间（分钟）。

停止逻辑

- 参数max_iter_num和max_run_time的停止条件是“或”关系，即任一条件满足时都会停止运行。
- 默认最多运行100次推理调用或最多运行10分钟。

7 支持的机器人说明

7.1 支持的机器人概述

R2C SDK通过RobotFactory统一创建和管理机器人硬件适配器。所有适配器实现IRobotHardwareAdapter接口（connect / disconnect / get_observation / send_action），通过YAML配置文件驱动。

支持的机器人总表

表7-1列出了R2C SDK当前支持的机器人及其相关配置信息，包括各类机器人的名称、硬件类型、适配器类、关键依赖、配置样例文件路径以及参考章节。

表 7-1 支持的机器人相关配置信息

机器人	硬件类型	适配器类	关键依赖	配置样例	参考章节
Dummy 仿真模拟器	dummy	DummyRobotHardwareAdapter	无	config/robot_dummy_config.yaml	**Dummy 仿真**
LeRobot (SO101)	lerobot	LeRobotHardwareAdapter	lerobot, draccus	config/robot_so101_lerobot_config.yaml	**LeRobot (SO101)**
UR5e RTDE直驱	ur5e_rtde	UR5eHardwareAdapter	ur-rtde, pyserial, pyrealsense2	config/robot_ur5e_config.yaml	**UR5e**
Flexiv Rizon4	flexiv	FlexivHardwareAdapter	Flexiv RDK	config/robot_flexiv_config.yaml	**Flexiv Rizon4**
Jaka Mini2	ros2	Ros2HardwareAdapter	rclpy (ROS2)	config/robot_mini2_ros_config.yaml	**Jaka Mini2**

机器人	硬件类型	适配器类	关键依赖	配置样例	参考章节
星海图 GALAXEA R1	zenoh_ros1	ZenohRos1HardwareAdapter	zenoh-python	config/robot_r1_zenoh_ros1_config.yaml	**星海图 GALAXEA R1**
五八智能 Q25	q25_ros2	Q25HardwareAdapter	无	config/robot_q25_config.yaml	**五八智能 Q25**
华沿 S05	s05	S05HardwareAdapter	pyrealsense2, CPS SDK	config/robot_s05_config.yaml	**华沿S05**
优艾隙锋	xifeng	XifengHardwareAdapter	numpy (mock) / pypilot (real)	config/robot_xifeng_config.yaml	**优艾隙锋**
拓斯达X5	tsd	TSDHardwareAdapter	pypilot	config/robot_tsd_config.yaml	**拓斯达X5**
千寻 Moz1	moz1	Moz1HardwareAdapter	mozrobot SDK	config/robot_moz1_config.yaml	**千寻 Moz1**

7.2 Dummy 仿真

本章介绍如何使用Dummy仿真适配器（ hardware.type: "dummy" ），无需任何硬件即可运行完整的R2C SDK链路。

硬件说明

Dummy适配器是一个纯软件仿真器，模拟SO101风格的6关节机械臂 + 1夹爪 + 多路相机图像。关节运动按max_joint_speed_rad_s进行一阶速度限制跟踪。

表 7-2 硬件信息

组件	数量	说明
关节	6	旋转关节，可配置名称和初始位置。
夹爪	1	步进电机仿真，开度0~1。
相机	≥1	随机噪声图像或从文件加载。

依赖与环境

零额外依赖。Dummy适配器只使用numpy（已在核心依赖中）。

配置文件

使用config/robot_dummy_config.yaml。

关键配置项如下表所示。

表 7-3 关键配置项

参数	类型	默认值	说明
joint_names	list[str]	DEFAULT_SO101_JOINT_NAMES	关节名称列表。
initial_joint_positions	list[float]	[0.0]*n	初始关节角度(rad)。
max_joint_speed_rad_s	float	2.5	关节最大仿真速度。
image_specs	dict	{"front": {480,640,3}}	相机配置，支持随机噪声或文件加载。

图像配置

```
image_specs:
  front:
    h: 480          # 图像高度
    w: 640          # 图像宽度
    c: 3            # 通道数（3=RGB, 1=灰度）
  wrist:
    file: "images/wrist_sample.png"   # 从文件加载（可选，与 h/w/c 互斥）
```

启动命令

```
# 安装SDK
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .
uv run python -m r2c_sdk.cloudroboclient  --bundle config/配置文件名.zip --robot-config config/
robot_dummy_config.yaml
```

键盘控制

表 7-4 按键功能

按键	功能
Space键	暂停/恢复。
h键	复位go_home（左臂 6 关节 → 0 rad，夹爪 → 0）。

按键	功能
s键	打印当前关节状态。
q键	退出。

常用场景

调试配置

```
uv run python -m r2c_sdk.cloudroboclient \
  --bundle config/配置文件名.zip \
  --robot-config config/robot_dummy_config.yaml \
  --log-level DEBUG \
  --duration 30
```

7.3 LeRobot (SO101)

本章介绍如何通过LeRobot适配器（ hardware.type: "lerobot" ）接入SO101双臂机器人。

硬件说明

表 7-5 硬件信息

组件	数量	协议	说明
双臂	2×6 DOF	LeRobot 标准协议	通过 robot.send_action() 驱动。
夹爪	2	LeRobot 协议	步进电机夹爪。
相机	2	OpenCV USB	• 分辨率：640×480 • 频率：30fps

依赖与环境

参考**配置软件环境**安装LeRobot环境。

```
# 安装SDK
cd /home/robot/project/r2c_sdk_python
conda activate lerobot
pip install -e .
```

配置文件

使用config/robot_so101_lerobot_config.yaml。

关键配置项如下表所示。

表 7-6 关键配置项

参数	说明
robot.type	LeRobot 机器人类型（so101_follower 等）。
robot.port	串口设备路径（默认 /dev/ttyACM0）。
robot.cameras.<name>.type	相机驱动类型（opencv）。
robot.cameras.<name>.index_or_path	相机设备索引或路径。
robot.cameras.<name>.fps	相机帧率。

启动命令

```
cd /home/robot/project/r2c_sdk_python
conda activate lerobot
python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_so101_lerobot_config.yaml
```

go_home

LeRobot go_home通过LeRobotGoHomeCommand实现，支持两种joint格式：

- Dict 模式：joints: {"joint_1": 0.0, "joint_2": 0.1, ...} — 直接传给 robot.send_action()
- List 模式：joints: [0.0, 0.1, ...] + joint_names: ["joint_1", "joint_2", ...] — 按序映射

```
# 命令配置
commands:
  go_home:
    type: go_home
    joint_names:
      - shoulder_pan.pos
      - shoulder_lift.pos
      - elbow_flex.pos
      - wrist_flex.pos
      - wrist_roll.pos
      - gripper.pos
    joints: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

键盘控制

表 7-7 按键功能

按键	功能
Space键	暂停/恢复。
h键	go_home（左臂 6 关节 → 0 rad，夹爪 → 0）。
q键	退出。

7.4 UR5e

本章介绍如何通过RTDE协议直驱Universal Robots UR5e协作机器人（hardware.type: "ur5e_rtde"）。

硬件说明

表 7-8 硬件信息

组件	型号	协议	说明
机械臂	UR5e	RTDE协议 (port 30004)	6 DOF 协作机械臂
夹爪	DH夹爪	RS485串口 (pyserial)	USB 转 RS485 连接
相机	Intel RealSense	USB 3.0	pyrealsense2 SDK

依赖与环境

```
cd /home/robot/project/r2c_sdk_python
uv sync
# 安装 RTDE + 相机 SDK
uv pip install ur-rtde pyserial pyrealsense2
```

配置文件

使用config/robot_ur5e_config.yaml。

关键配置项如下表所示。

表 7-9 关键配置项

参数	类型	说明
robot.robot_ip	str	UR5e控制器IP地址。
robot.frequency	int	RTDE控制循环频率 (Hz)，推荐500。
robot.gain	int	servoL增益，越高越灵敏但可能振荡。
action_mode	str	"absolute" = 绝对角度控制，"delta" = 增量控制。
robot.max_pos_speed	float	位移速度限制 (m/s)。
robot.max_rot_speed	float	旋转速度限制 (rad/s)。

启动命令

```
cd /home/robot/project/r2c_sdk_python
uv run python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_ur5e_config.yaml
```

键盘控制

表 7-10 按键功能

按键	功能
Space键	暂停/恢复。
q键	退出。

安全注意事项

- **始终确保急停按钮可用**。
- 首次运行用dry_run: true验证数据流。
- gain参数从低值开始逐步调高。
- 确认max_pos_speed 和 max_rot_speed限速合理。

7.5 Flexiv Rizon4

本章介绍如何通过 Flexiv RDK 接入非夕 Flexiv Rizon4 自适应机器人（ hardware.type: "flexiv" ）。

硬件说明

表 7-11 硬件信息

组件	型号	协议	说明
机械臂	Flexiv Rizon4	Flexiv RDK	7 DOF 力控协作机械臂
夹爪	末端夹爪	RDK 内置接口	通过 RDK gripper states 获取开度
相机	外接 USB 相机	OpenCV	通过 capture thread 异步采集（ ADR-0006 ）

依赖与环境

```
# 安装 SDK
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .
```

```
# 安装 Flexiv RDK
uv pip install flexivrdk==1.9.0
```

Flexiv RDK包含 C++ SDK + Python binding，安装方式根据非夕提供的文档。

配置文件

使用 config/robot_flexiv_*.yaml。

关键配置项如下表所示。

表 7-12 关键配置项

参数	说明
robot_ip	Flexiv控制器IP。
local_ip	边缘设备 IP（与控制器同一子网）。
home.joint_positions	默认home位置（7 维关节 + 夹爪）。
camera.enabled	是否启用相机。

启动命令

```
cd /home/robot/project/r2c_sdk_python
uv run python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip  --robot-config config/
robot_flexiv_config.yaml
```

go_home

Flexiv 提供专用 FlexivGoHomeCommand，支持两种模式：

```
commands:
  go_home:
    type: go_home
    pose_euler: [0.5, 0.0, 0.3, 0.0, 3.14, 0.0]  # [x, y, z, roll, pitch, yaw]
    gripper: 0.02    # 可选,夹爪目标宽度(m)
```

- pose_euler：直接调用RDK移动到目标末端位姿。
- joints：调用RDK关节空间移动。

键盘控制

表 7-13 按键功能

按键	功能
Space键	暂停/恢复。
h键	复位（go_home）。
q键	退出。

安全注意事项

- **始终确保急停按钮可用。**
- 首次运行用dry_run: true验证。
- home位姿需在机器人工作空间内。

7.6 Jaka Mini2

本章介绍如何通过 ROS2 适配器（ hardware.type: "ros2" ）接入 Jaka Mini2 协作机械臂。

硬件说明

表 7-14 硬件信息

组件	型号	协议	说明
机械臂	Jaka Mini2 (JAKA MiniCobo)	ROS2 topic	6 DOF 协作机械臂，TCP 通信（默认 IP 192.168.1.xxx ）
夹爪	Wheeltec MS42DC 步进电机	ROS2 topic (step_motor/Motor)	开度 0～0.13 m，USB CDC-ACM 串口，115200 baud
相机 ×2	USB UVC	ROS2 topic (CompressedImage)	腕部 + 顶部，640×480 @ 30fps

关键 ROS2 话题

话题	消息类型	方向	说明
/jaka_driver/joint_position	sensor_msgs/JointState	硬件 → SDK	关节状态，10 Hz
/gripper_state	std_msgs/Float32	硬件 → SDK	夹爪开度百分比
/camera_top/image_raw/compressed	sensor_msgs/CompressedImage	硬件 → SDK	顶部相机
/camera_wrist/image_raw/compressed	sensor_msgs/CompressedImage	硬件 → SDK	腕部相机
/robot_command_servo_j	sensor_msgs/JointState	SDK → 硬件	机械臂 6 关节控制

话题	消息类型	方向	说明
/motor_control	step_motor/Motor	SDK → 硬件	夹爪控制

依赖与环境

```
# 安装 ROS2 Humblesudo
apt install ros-humble-desktop python3-colcon-common-extensions ros-humble-usb-cam

# 克隆并编译 r2c_jaka ROS2 workspace
git clone git@github.com:suhanwu/r2c_jaka.git

cd ~/jaka_mini2/r2c_jaka
colcon build
source install/setup.bash

# 安装 SDK
cd /home/robot/project/r2c_sdk_python
pip install -e .
```

环境激活顺序

```
source /opt/ros/humble/setup.bash
source ~/jaka_mini2/r2c_jaka/install/setup.bash
cd /home/robot/project/r2c_sdk_python
```

配置文件

使用 config/robot_mini2_ros_config.yaml

关键配置差异

Jaka Mini2配置与其他ROS2机器人的主要差异：

表 7-15 配置差异

差异点	Jaka Mini2	通用 ROS2
相机类型	CompressedImage	Image 或 CompressedImage
图像 transform	ros_compressed_image_to_ndarray → bgr_to_rgb → ndarray_to_jpeg	ros_image_to_jpeg
夹爪消息类型	step_motor.msg.Motor（自定义）	Float64MultiArray
控制话题	/robot_command_servo_j (JointState)	controller commands
动作目标	commands_by_publisher.jaka_control.data	直接 target: publisher_name

启动硬件节点

在跑SDK之前，需要先启动r2c_jaka的ROS2节点：

```
# 相机 + 夹爪（一键启动）
ros2 launch camera_c170 camera_and_motor.launch.py

# 机械臂控制（根据实际 IP 修改）
ros2 launch jaka_joint_control jaka_joint_control.launch.py robot_ip:=192.168.1.xxx
```

验证硬件话题：

```
ros2 topic hz /jaka_driver/joint_position
# 预期 10 Hz
ros2 topic hz /gripper_state
# 预期 10 Hz
ros2 topic hz /camera_top/image_raw/compressed
# 预期 30 Hz
```

启动SDK

```
cd /home/robot/project/r2c_sdk_python
source /opt/ros/humble/setup.bash
source ~/jaka_mini2/r2c_jaka/install/setup.bash python -m r2c_sdk.cloudroboclient 配置文件名.zip --robot-config config/robot_mini2_ros_config.yaml
```

键盘控制

表 7-16 按键功能

按键	功能
Space键	暂停 / 恢复
q键	退出

udev 规则（设备持久化）

```
# 夹爪串口持久化
sudo bash ~/jaka_mini2/r2c_jaka/src/step_motor/ch9102_udev.sh

# 用户加入 dialout 组
sudo usermod -aG dialout $USER
```

7.7 星海图 GALAXEA R1

本章介绍如何通过 Zenoh ROS1 适配器（hardware.type: "zenoh_ros1"）接入星海图 GALAXEA R1。SDK 通过 zenoh-bridge-ros1 桥接 ROS1 topic 到 Zenoh data plane。

已支持场景

表 7-17 支持的场景

机器人	配置样例	说明
星海图 GALAXEA R1	config/robot_r1_zenoh_ros1_config.yaml	双臂人形机器人 (HDAS)

依赖与环境

```
# 安装 zenoh-bridge-ros1（在机器人边缘设备上)

# 安装 SDK
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .
```

zenoh-bridge-ros1 配置

zenoh-bridge-ros1配置使用JSON5格式，具体请参考R2C SDK安装包中的文件config/r1.json5。

启动 zenoh-bridge-ros1

```
zenoh-bridge-ros1 -c config/r1.json5
```

边缘端配置

使用config/robot_r1_zenoh_ros1_config.yaml。

关键配置项

- **Subscriptions（上行）**

  表 7-18 Subscriptions 关键配置项

参数	说明
ros_topic	ROS1 原始 topic 名，需与 bridge 白名单一致
msg_type	ROS1 消息类型字符串
store_as	内部 streams 存储 key

- **Publishers（下行）**

表 7-19 Publishers 关键配置项

参数	说明
ros_topic	目标 ROS1 topic
msg_type	ROS1 消息类型
encoder	编码策略：motor_control / float32 / raw_bytes
go_home_role	arm 或 gripper，标记 go_home 中的角色

内置 ROS1 消息类型

表 7-20 内置 ROS1 消息类型

消息类型	用途
sensor_msgs/CompressedImage	相机JPEG。
sensor_msgs/JointState	关节状态。
sensor_msgs/Image	原始图像。
std_msgs/Float32	标量。
std_msgs/Float32MultiArray	浮点数组。
hdas_msg/motor_control	HDAS电机控制。

启动命令

```
# 终端 1：启动 zenoh-bridge-ros1（在 R1 边缘设备上）
zenoh-bridge-ros1 -c config/r1.json5

# 终端 2：启动 SDK 边缘端
cd /home/robot/project/r2c_sdk_python
uv run python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_r1_zenoh_ros1_config.yaml
```

键盘控制

表 7-21 按键功能

按键	功能
Space键	暂停 / 恢复
q键	退出

扩展双臂控制

当前推理路径只接左臂（6 关节 + 1 夹爪 = 7D）。如需接入双臂（14D），扩展 device_to_r2c 和 r2c_to_device 的映射。

7.8 五八智能 Q25

本章介绍如何接入五八智能天狼Q25四足机器人（hardware.type: "q25_ros2"）。

硬件说明

表 7-22 硬件信息

组件	协议	说明
机体	ROS2 topic 控制	通过 std_msgs/String JSON 轨迹命令控制
相机	ROS2 sensor_msgs/Image	板载相机，224×224

依赖与环境

Q25 适配器无额外 Python 依赖。需要 ROS2 运行环境（Humble+）以订阅/发布 topic。

```
cd /home/robot/project/r2c_sdk_python
source /opt/ros/humble/setup.bash
pip install -e .
```

配置文件

使用config/robot_q25_config.yaml。

关键配置项如下表所示。

表 7-23 关键配置项

参数	说明
class_path	必须为 r2c_sdk.robots.q25_hardware_adapter.Q25HardwareAdapter
image_topic	板载相机 ROS2 topic
trajectory_topic	轨迹指令发布 topic
use_mock_image	true = 使用灰色模拟图（无相机时调试用）
dry_run	true = 仅打印不执行实际控制

启动命令

```
source /opt/ros/humble/setup.bash
cd /home/robot/project/r2c_sdk_python
python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_q25_config.yaml
```

模拟摇杆控制

Q25也支持UDP直驱模式（config/robot_q25_config_joystick.yaml），通过UDP协议直接发送轨迹命令。

7.9 华沿 S05

本章介绍如何接入华沿（Huayan）S05 机器人（hardware.type: "s05"）。

硬件说明

表 7-24 硬件信息

组件	协议	说明
机械臂	华沿 CPS SDK	6 DOF 旋转关节，TCP 通信
夹爪	Modbus RTU	汉森夹爪，通过机器人末端串口控制
相机	RealSense	Intel RealSense 深度相机，彩色流采集

依赖与环境

```
# 确认 SDK
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .
# 安装 RealSense SDK
uv pip install pyrealsense2
```

CPS SDK（CPS.py）和夹爪控制（gripper_control.py）由华沿提供，放置在 doc/hardware/ 目录下，适配器通过文件路径动态导入，无需安装为 Python 包。

配置文件

使用config/robot_s05_config.yaml。

关键配置项

表 7-25 关键配置项

参数	说明
s05_config.robot_ip	机器人 IP 地址
s05_config.robot_port	机器人端口号
s05_config.gripper_slave_id	夹爪 Modbus slave ID
sdk_paths.cps_sdk	CPS.py文件路径（联系厂商华沿S05获取）（相对配置文件或绝对路径）
sdk_paths.gripper_control	gripper_control.py 文件路径
cameras.width	相机图像宽度
cameras.height	相机图像高度
cameras.fps	相机帧率
dry_run	true = 空运行模式，不操作物理硬件

启动命令

```
cd /home/robot/project/r2c_sdk_python
uv sync
# 启动
uv run python -m r2c_sdk.cloudroboclient  --bundle config/配置文件名.zip  --robot-config config/
robot_s05_config.yaml
```

Dummy 测试

使用config/robot_s05_dummy_config.yaml（dry_run: true），无需连接物理硬件：

```
uv run python -m r2c_sdk.cloudroboclient  --bundle config/配置文件名.zip  --robot-config config/
robot_s05_dummy_config.yaml
```

键盘控制

表 7-26 按键功能

按键	功能
Space键	暂停 / 恢复
q键	退出

启用方式：runtime.keyboard_control.enabled: true。

参考

- R2C SDK安装包中的config/robot_s05_config.yaml配置文件。
- Dummy 测试配置：config/robot_s05_dummy_config.yaml

7.10 优艾隙锋

本章介绍如何接入隙锋人形机器人（ hardware.type: "xifeng" ）。

硬件说明

表 7-27 硬件信息

组件	协议	说明
左臂	pypilot (PilotSDK)	7 DOF 旋转关节
右臂	pypilot (PilotSDK)	7 DOF 旋转关节
左夹爪	串口 hex 命令	离散开/合控制
右夹爪	串口 hex 命令	离散开/合控制
相机	不支持	当前版本不采集 RGB 图像

隙锋机器人共 14 个手臂关节 + 2 个夹爪，observation 只含关节和夹爪状态（ 无图像 ）。

依赖与环境

```
# 安装 SDK
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .
# mock 模式：无需额外依赖（ 仅 numpy ）
# real 模式：安装 pypilot
uv pip install pypilot
```

配置文件

使用config/robot_xifeng_config.yaml。

关键配置项

表 7-28 关键配置项

参数	说明
source	"mock" 合成数据模式 / "real" 直连真机
dry_run	true = 只打印 observation/action 日志，不向真机下发动作

参数	说明
real.robot_host	机器人主机 IP
real.arm_local_ip	本地网卡 IP（与机器人通信用）
real.arm_ip	机械臂 IP
real.arm_port	机械臂端口
real.left_arm_id	左臂 ID
real.right_arm_id	右臂 ID
real.joint_poll_hz	关节状态轮询频率 (Hz)
mock.max_joint_speed_rad_s	mock 模式下关节最大角速度 (rad/s)

启动命令

Mock 模式（无需真机）

```
cd /home/robot/project/r2c_sdk_python
uv sync
uv run python -m r2c_sdk.cloudroboclient  --client-config config/client_xifeng_local.yaml --robot-config config/robot_xifeng_config.yaml
```

Real 模式（连接真机）

1. 修改配置：source: "real"，dry_run: true
2. 首次联调：保持 dry_run: true，比对关节名和数据格式
3. 测通后：将 dry_run 改为 false

```
uv run python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/robot_xifeng_config.yaml
```

键盘控制

表 7-29 按键功能

按键	功能
Space键	暂停 / 恢复
q键	退出

启用方式：runtime.keyboard_control.enabled: true。

参考

- R2C SDK安装包中的config/robot_xifeng_config.yaml配置文件
- 本地联调客户端配置：config/client_xifeng_local.yaml

7.11 拓斯达 X5

本章介绍如何接入拓斯达 X5 机械臂（ hardware.type: "tsd" ）。

硬件说明

表 7-30 硬件信息

组件	协议	说明
机械臂	拓斯达 SDK（ TCP ）	6 DOF 机械臂，通过拓斯达 SDK 直驱

依赖与环境

```
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .

# 文件名中的 `<version>`、`<py>`、`<platform>`、`<arch>`（ 如 `x86_64` ）咨询拓斯达科技客服人员。
uv pip install xapi-<version>-cp<py>-cp<py>m-<platform>_<arch>.whl
```

关键配置项如下表所示。

表 7-31 关键配置项

参数	说明
ip	机器人控制器 IP（ 可用 ${TSD_IP} 环境变量，默认192.168.1.xxx ）
default_speed	关节运动默认速度
reconnect_max_retries	断线重连次数
reconnect_delay_s	重连间隔（ 秒 ）
auto_enable_servo	连接时自动上伺服
auto_set_mode	自动设置控制模式
auto_mode_value	控制模式值
auto_set_speed	自动设置速度
dry_run	true = 仅打印不执行实际控制
mock_mode	true = 仿真模式（ 不连接真实硬件，使用模拟观测 ）

启动命令

```
cd /home/robot/project/r2c_sdk_python
uv run python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_tsd_config.yaml
```

7.12 千寻 Moz1

本章介绍如何接入千寻智能Moz1双臂人形机器人（ hardware.type: "moz1" ）。

硬件说明

表 7-32 硬件信息

组件	数量	协议	说明
双臂	2×7 DOF	Moz1 SDK (mozrobotSDK)	左臂 7 + 右臂 7 = 14 维关节
相机	3× RealSense	pyrealsense2	高位 + 左腕 + 右腕

依赖与环境

```
cd /home/robot/project/r2c_sdk_python
uv sync
uv pip install -e .
# 安装 Moz1 SDK
uv pip install mozrobotSDK pyrealsense2
```

配置文件

使用config/robot_moz1_config.yaml。

关键配置项如下表所示。

表 7-33 关键配置项

参数	说明
structure	机器人躯体结构：dualarm / wholebody_without_base / wholebody
robot_control_hz	控制频率 (Hz)，推荐 120
camera_mode	off / on / visualize
realsense_serials	三路 RealSense 序列号（逗号分隔）
camera_resolutions	三路相机分辨率（W*H 格式）
dry_run	true = 仅打印不执行实际动作

启动命令

```
cd /home/robot/project/r2c_sdk_python
uv run python -m r2c_sdk.cloudroboclient --bundle config/配置文件名.zip --robot-config config/
robot_moz1_config.yaml
```

Moz1 专用 Transform

表 7-34 Moz1 专用 Transform 的用途

Transform	用途
moz1_gripper_value_to_joints	将夹爪动作值转换为 Moz1 关节格式。
moz1_extract_gripper_states	从 joint_states 中提取夹爪状态。

8 R2C SDK 已支持机器人的通用配置

8.1 runtime 配置

请在本地**hw_r2c_sdk-*x.x.x*/r2c_sdk_python/config**文件夹找到机器人配置文件**robot_*{不同的机器人}*_config.yaml**，比如robot_flexiv_config.yaml。

机器人端配置文件中**runtime配置项**主要分为基础控制项、异步融合、心跳上报、键盘控制这几类。

runtime配置项示例：

```
runtime:
  publish_hz: 20.0              # 动作运行频率 (Hz)，同时决定观测发布检查频率
  max_duration_s: 0.0             # 最大运行时长 (秒)，0表示持续运行
  dry_run: false                # 调试模式：只发送观测，不运行动作
  action_response_timeout_s: 5.0  # 等待云端动作响应的超时时间 (秒)
  max_enqueue_actions_per_chunk: -1  # 单次入队的动作数上限，-1表示全部采用
  enable_action_chunk_alignment: false  # 启用chunk对齐
  skip_initial_observations: 1    # 启动后跳过的初始观测数，默认1

  # 心跳上报
  heartbeat:
    enabled: true
    interval_ms: 5000
    jitter_ms: 0
    status: "ONLINE"
    mode: "AUTO"

  # 键盘控制
  keyboard_control:
    enabled: false
  # ── 异步请求融合配置 ──────────────────────────────────────────────────────────
  # 控制是否启用异步请求与轨迹融合。启用后，队列未空即可提前发布新
  # observation（掩盖云端推理延迟），并用融合策略平滑新旧 chunk 衔接。
  async_request:
    enabled: false
    # 队列中剩余 action 数低于此阈值时提前请求新 observation。
    # 0 退化为队列空才发布。
    publish_trigger_threshold: 0
    # 轨迹融合子配置
    fusion:
      # 融合策略: replace (直接替换) / weighted_average (加权平均) / nearest_neighbor (最近邻)
      strategy: weighted_average
      # 融合窗口大小 (用于 weighted_average / nearest_neighbor)
      window_size: 10
      # 类型感知融合 (Type-Aware Fusion)：声明 joint_states.position 中每个元素的物理语义。
      # 可选值: joint_angle / position_xyz / euler / quaternion。
      # 需同时配置 strategy != replace 才能触发融合。
```

```
  state_types:
    - joint_angle
    - joint_angle
    - joint_angle
    - joint_angle
    - joint_angle
    - joint_angle
  # state_type_order — 为每个观测字段指定类型优先级排序 (可选)
  # state_type_order:
  #   euler: rpy
  #   quaternion: wxyz
```

runtime 基础控制项

表 8-1 基础控制项参数

参数	类型	默认值	说明
publish_hz	float	30	控制循环（动作运行）频率。 主循环每tick消费1个队列中的动作并运行，同时检查是否需要发布新观测。 实际观测发布速率取决于云端响应和队列消耗速度，一般为10Hz~30Hz。
max_duration_s	float	0	最大运行时长。 **0或负数**表示持续运行，需通过Ctrl+C停止运行。
dry_run	bool	false	该参数为**true**时正常发送观测数据，但不运行推理返回的动作。 适用于调试。
action_response_timeout_s	float	1.0	发送观测后等待云端动作的超时时间。 超时后重新发送观测。
skip_initial_observations	int	1	启动后跳过前N个观测不发布。 首个观测通常包含不完整数据（传感器未就绪），默认跳过1个。

动作序列与异步融合

云端返回的action chunk（动作序列）被拆分为多个action，将action逐个推入运行队列消费。当队列中仍有未消费的旧动作时，新chunk到达会触发融合逻辑，避免关节轨迹跳变。

表 8-2 动作队列与异步融合参数

参数	类型	默认值	说明
max_enqueue_actions_per_chunk	int	-1	单次入队的动作数上限，超出部分丢弃。 **-1**表示全部采用。
enable_action_chunk_alignment	bool	false	新chunk到达时，先对齐到上次已运行的最后一个动作位置。 适用于chunk之间存在重叠的推理场景。
publish_trigger_threshold	int	0	队列中剩余动作数低于此阈值时触发新一轮观测发布。 **0**表示队列清空才发布（不进行异步融合）。
fusion_strategy	str	replace	融合策略，当前仅支持对关节值的融合，末端位姿暂不支持，请参见**表8-3**。
fusion_window_size	int	10	融合窗口大小，仅weighted_average策略使用。

表 8-3 融合策略

策略	说明	适用场景
replace	直接替换旧队列（默认行为）。	低延迟、single-step推理
weighted_average	新旧动作前W步线性加权平均，逐步过渡。	chunk推理、要求平滑过渡
nearest_neighbor	以当前关节状态为锚点，在新chunk中匹配最近点作为起点。	关节快速运动、空间对齐

动作队列与异步融合的配置建议

```
# 低延迟模式（single-step 推理，不融合）
runtime:
  publish_trigger_threshold: 0
  fusion_strategy: "replace"

# 平滑过渡模式（chunk 推理，加权融合）
runtime:
  publish_trigger_threshold: 10
  fusion_strategy: "weighted_average"
  fusion_window_size: 10
```

心跳上报

runtime.heartbeat参数控制向云端定期发送心跳信号，用于监控设备在线状态。

表 8-4 heartbeat 参数

参数	类型	默认值	说明
enabled	bool	false	是否启用心跳
interval_ms	int	1000	心跳间隔
jitter_ms	int	0	随机抖动（毫秒），避免多设备同时上报
status	str	"ONLINE"	心跳状态值
mode	str	"AUTO"	心跳模式

心跳上报示例

```
heartbeat:
  enabled: true
  interval_ms: 5000
  status: "ONLINE"
  mode: "AUTO"
```

键盘控制

runtime.keyboard_control参数启用终端键盘快捷键控制机器人。

表 8-5 keyboard_control 参数

参数	类型	默认值	说明
enabled	bool	false	是否启用键盘控制。 键盘控制默认快捷键如**表8-6**。

表 8-6 键盘控制**默认快捷键**

按键	功能
Space键	暂停/恢复。
h键	复位（go_home）。
q键	停止。

8.2 传输数据压缩配置

请在本地**hw_r2c_sdk-*x.x.x*/r2c_sdk_python/config**文件夹找到机器人配置文件**robot_*{不同的机器人}*_config.yaml**，比如robot_flexiv_config.yaml。

传输数据压缩主要通过图像压缩配置实现，图像压缩通过机器人配置文件的**device_to_r2c.mappings.transforms**配置。

transforms 图像转换器

在**device_to_r2c.mappings.transforms**指定转换器，对原始图像数据进行编码。

说明

ndarray_to_* 转换器要求**BGR**输入。如果原始数据为RGB，需要先加rgb_to_bgr。

表 8-7 transforms 可用转换器

注册名	输入	输出	可调参数
ndarray_to_jpeg	BGR ndarray	JPEG bytes	quality（默认95）
ndarray_to_webp	BGR ndarray	WebP bytes	quality（默认80）
ndarray_to_png	BGR ndarray	PNG bytes	无（无损）
ros_image_to_jpeg	ROS Image	JPEG bytes	quality（默认95）
ros_image_to_webp	ROS Image	WebP bytes	quality（默认80）
ros_compressed_image_to_jpeg	ROS CompressedImage	JPEG bytes	quality（默认95）
ros_compressed_image_to_webp	ROS CompressedImage	WebP bytes	quality（默认80）
rgb_to_bgr	RGB ndarray	BGR ndarray	无

transforms写法示例

```
# 写法 1: 单字符串
transforms: ndarray_to_jpeg

# 写法 2: 列表串联
transforms:
  - rgb_to_bgr
```

```
  - ndarray_to_webp

# 写法 3: 带 quality 参数
transforms:
  - ndarray_to_webp:
      quality: 70
```

图像压缩格式对比

表 8-8 图像压缩格式对比

特性	JPEG	WebP	PNG
压缩方式	有损	有损	无损
quality范围	0~100	0~100	-
默认quality	95	80	-
压缩比（640×480）	≈20×	≈32×	≈2×
编码耗时	≈1.4ms	≈42ms	≈47ms
体积（vs JPEG）	基准	小35%~40%	大9×
适用场景	通用生产	带宽受限	调试/精确比对

不同编码方式的带宽参考

以2路640×480相机20Hz为例，带宽参考如表8-9。

表 8-9 带宽参考

编码方式	带宽	适用网络
raw（无压缩）	≈282Mbps	仅本地测试
PNG	≈64Mbps	本地/有线
JPEG q=80	≈15Mbps	宽带/4G
WebP q=80	≈9Mbps	4G/弱宽带
WebP q=40	≈4Mbps	极端低带宽

推荐在生产环境使用ndarray_to_webp（quality70~80），兼顾画质与带宽。

传输数据压缩典型场景配置示例

- **ROS2场景**（sensor_msgs/CompressedImage输入）

```
device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.wrist"
```

```
    source_path: "streams.wrist"
    transforms:
      - ros_compressed_image_to_ndarray
      - bgr_to_rgb
      - ndarray_to_jpeg:
          quality: 80
```

- **LeRobot/SDK直驱场景**（numpy ndarray输入，RGB格式）

```
device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.front"
      source_path: "front"
      transforms:
        - rgb_to_bgr
        - ndarray_to_webp:
            quality: 70
```

- **Dummy仿真场景**

```
device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.wrist"
      source_path: "wrist"
      transforms:
        - ndarray_to_webp:
            quality: 70
```

8.3 机器人通用配置典型片段

ROS2 机器人（Jaka Mini2）

```
runtime:
  publish_hz: 20.0
  max_duration_s: 0.0
  dry_run: false
  action_response_timeout_s: 5.0
  max_enqueue_actions_per_chunk: -1
  enable_action_chunk_alignment: true
  heartbeat:
    enabled: true
    interval_ms: 5000
    status: "ONLINE"
    mode: "AUTO"

device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.wrist"
      source_path: "streams.wrist"
      transforms:
        - ros_compressed_image_to_ndarray
        - bgr_to_rgb
        - ndarray_to_jpeg
```

LeRobot 机器人（SO101）

```
runtime:
  publish_hz: 10.0
  publish_trigger_threshold: 10
  fusion_strategy: "weighted_average"
  fusion_window_size: 10
  heartbeat:
    enabled: true
    interval_ms: 5000
    status: "ONLINE"
    mode: "AUTO"
```

```
device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.front"
      source_path: "front"
      transforms:
        - rgb_to_bgr
        - ndarray_to_webp:
            quality: 70
```

Flexiv SDK 直驱

```
runtime:
  publish_hz: 20.0
  dry_run: false
  action_response_timeout_s: 5.0
  keyboard_control:
    enabled: true

hardware:
  type: "flexiv"
  flexiv_config:
    robot_sn: "Rizon4-XXXXXX"
    cameras:
      wrist:
        source: 0
        width: 640
        height: 480
        fps: 30

device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.wrist"
      source_path: "wrist"
      transforms:
        - ndarray_to_webp:
            quality: 70
```

Dummy 测试

```
runtime:
  publish_hz: 10.0
  dry_run: true
  max_duration_s: 60.0

hardware:
  type: "dummy"
  dummy_config:
    image_width: 640
    image_height: 480

device_to_r2c:
  image_encoding: jpeg
  mappings:
    - target_key: "images.color.wrist"
      source_path: "wrist"
      transforms: ndarray_to_jpeg
```

9 R2C SDK 通过命令机制控制机器人

9.1 命令机制总览

在机器人使用者选定机器人型号后，需要写一份YAML配置，让cloudroboclient启动后键盘按h/空格就能让**机械臂归位**。

请在本地**hw_r2c_sdk-*x.x.x*/r2c_sdk_python/config**文件夹找到机器人配置文件**robot_*{不同的机器人}*_config.yaml**，比如robot_flexiv_config.yaml，机器人配置中包含命令机制配置。

命令机制配置**commands.go_home**只接受 4个标准键，其余都会被拒绝。

表 9-1 支持的标准键

键名	类型	含义	适用适配器
pose_euler	[x, y, z, roll, pitch, yaw] 6 floats	米 + 弧度，TCP目标位姿（欧拉角）	flexiv（其它适配器报NotImplementedError）
pose_quat	[x, y, z, qw, qx, qy, qz] 7 floats	米 + 单位四元数，TCP目标位姿	-
joints	[j1, j2, ..., jN] N floats	各关节目标角度(弧度)，N = 机器人 DOF	dummy/lerobot/flexiv
gripper	float或"open" / "close"	夹爪目标宽度（m）或语义动作	flexiv（其它适配器走no-op）

关键约束

- pose_euler/pose_quat/joints**三选一**，只允许一个存在。配置里同时放多个键，启动时报ValueError。
- 三个运动键**都不填**，不会报错，SDK打印WARNING日志后跳过本次执行（软失败）。
- gripper是可选的，只填夹爪键不动臂是不允许的（软失败），但是**臂 + 夹爪组合**是允许的。

- 调用顺序固定为**先臂后夹爪**：臂没动成功就不会动夹爪（atomic语义，中间态不安全）。

触发方式

按键盘按键进行命令触发，默认情况下，键盘按键触发的命令如表9-2。

表 9-2 按键触发的命令

按键	命令	说明	可重映射?
空格键	pause_resume	暂停/恢复推理循环	保留键
h键	go_home	复位（需暂停态）	支持
q键	graceful_stop	退出进程	保留键

键盘触发条件：在runtime段打开keyboard_control.enabled。

典型配置片段

```
runtime:
  keyboard_control:
    enabled: true   # 空格 触发暂停/恢复，在暂停时按h 触发 go_home
hardware:
  type: dummy
  config:
    commands:
      go_home:      # 命令实例名称 键盘按键映射时的目标
        type: go_home  # 命令类型 go_home 表示复位命令 可定义多个go_home 命令实例，从而支持多个复位点
        joints: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
      go_home_ready:      # 命令实例名称 键盘按键映射时的目标
        type: go_home  # 命令类型 go_home 表示复位命令 可定义多个go_home 命令实例，从而支持多个复位点
        joints: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
```

9.2 各机器人配置详解

本章节介绍各机器人支持的运动键、典型配置、运行时调用的目标SDK函数、是否实现set_gripper。

说明

✓ 代表支持，✗代表抛NotImplementedError/ValueError（运动键）或走no-op（夹爪键）

表 9-3 各机器人与标准键对照表

适配器	pose_euler	pose_quat	joints	gripper数字	gripper字符串
dummy	✗	✗	✓	✗（no-op）	✗（no-op）
lerobot	✗	✗	✓	✗（no-op）	✗（no-op）

适配器	pose_euler	pose_quat	joints	gripper数字	gripper字符串
flexiv	✓	✗	✓	✓	✓

dummy-仿真/单元测试用

- **支持键**joints（只此一种，任意长度）
- **不实现**set_gripper(走no-op)

典型配置

```
hardware:
  type: "dummy"
  config:
    joint_names: ["j1", "j2", "j3", "j4", "j5", "j6"]
    commands:
      go_home:
        type: go_home
        joints: [0.0, 0.0, 0.0, 0.0, 0.0, 0.5]
```

go_home触发后调用DummyRobotHardwareAdapter.move_to(joints=...)，内部走send_action({"joint_target": [v, ...]})。

仿真模式下不会真实驱动硬件，适合端到端流程联调。

lerobot-LeRobot 系列（so101 等）

- **支持键**joints（list 形式，顺序对应joint_names）
- **不实现**set_gripper（走no-op）

典型配置

```
hardware:
  type: "lerobot"
  config:
    robot:
      type: "so101"
      port: "/dev/ttyACM0"
    joint_names: ["shoulder_pan", "shoulder_lift", "elbow", "wrist_1", "wrist_2", "wrist_3", "gripper"]
    commands:
      go_home:
        type: go_home
        joint_names:
          - shoulder_pan.pos
          - shoulder_lift.pos
          - elbow_flex.pos
          - wrist_flex.pos
          - wrist_roll.pos
          - gripper.pos
        joints: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

LeRobot的gripper是机械结构上的一个独立关节，不需要单独走set_gripper 接口，直接给到joints列表的对应位置即可。

flexiv-Flexiv Rizon 4（唯一带夹爪的实现）

- **支持键**pose_euler/joints
- **实现**set_gripper（走vendor的Flexiv-GN01夹爪）

典型配置（pose + 夹爪宽度）

```
hardware:
  type: "flexiv"
  config:
    robot_sn: "Rizon4-062833"
    gripper_name: "Flexiv-GN01"
    gripper_open_width: 0.09      # 夹爪完全打开(m)
    gripper_close_width: 0.012    # 夹爪完全闭合(m)
    gripper_velocity: 0.10
    gripper_force_limit: 20.0
    commands:
      go_home:
        type: go_home
        pose_euler: [0.5, 0.0, 0.3, 0.0, 3.14, 0.0]   # 米 + 弧度
        gripper: 0.02                                  # 可选,目标宽度(m)
```

gripper键接受两种值，分为：

- **数字**

  set_gripper(width=<float>)，移动到指定宽度。

- **字符串**

  "open" / "close":set_gripper(action="open"|"close")，移动到gripper_open_width或gripper_close_width。

move_to走Flexiv RDK的execute_primitive调用；set_gripper内部调用_move_gripper(width)。

flexiv是SDK内**唯一**实现set_gripper的内置适配器。

9.3 常见问题排查

常见问题排查

表 9-4 常见问题排查

问题	原因	修复
ERROR [__main__] keyboard_control.keymap['j'] references command 'go_home2', but no such command is registered.	不存在的命令实例名称。	修改为正确的命令实例名称，查看commands配置。
WARNING: go_home: no motion key in config ...; skipping.	4个键都没填。	填pose_euler/pose_quat/joints 中的一个。

软失败示例

下列配置合法，启动OK，但按下h不会动，启动时不会报错。

```
hardware:
  type: "jaka"
  config:
    jaka:
      ip: "192.168.1.xxx"
      commands:
```

```
    type: go_home
    go_home: {}   # 4 个键都没有
```

运行时日志

```
WARNING  r2c_sdk.robots.commands.generic: go_home: no motion key in config {}; skipping.
```

这种用法适合“我已经预留了go_home配置位，具体目标运行时再传”的场景，通过 adapter.execute_command("go_home", joints=[...]) 在代码侧注入。

10 R2C SDK 通过键盘控制机器人

10.1 键盘控制概述

请在本地**hw_r2c_sdk-*x.x.x*/r2c_sdk_python/config**文件夹找到机器人配置文件**robot_*{不同的机器人}*_config.yaml**，比如robot_flexiv_config.yaml，机器人配置中包含键盘控制。

键盘控制通过stdin轮询实现机器人控制，仅在**终端窗口有键盘焦点**时响应按键，支持Linux（X11/Wayland/TTY）、macOS、Windows。

架构组件

表 10-1 架构组件

模块	职责
KeyboardController	stdin轮询 + 命令分派
KeyboardCommandMapper	按键 与命令名映射（支持merge + 大小写fallback）
SyncRobotClient	注入pause/stop回调

核心特性

- **焦点感知**：仅终端有焦点时响应，全局无干扰
- **零依赖**：纯Python stdlib（sys.stdin + select/msvcrt）
- **跨平台**：Linux（X11/Wayland/TTY）、macOS、Windows统一
- **安全门控**：命令可选择“暂停态才执行”，防止运动中误触
- **大小写不敏感**：按j或J都匹配

平台兼容性

表 10-2 平台兼容性

平台	终端	实现	终端恢复
Linux X11	GNOME Terminal / Konsole	os.read(fd) + select + tty.setcbreak	√
Linux Wayland	同上	同上	√
Linux TTY	纯终端	同上	√
macOS	Terminal.app / iTerm2	同上	√
Windows	cmd / PowerShell / Windows Terminal	msvcrt.kbhit() + getwch()	无需

10.2 快速开始

最小配置

```
runtime:
  keyboard_control:
    enabled: true
```

启动后日志输出当前keymap

```
[KB] Keyboard control started. keymap: [ ] pause_resume, [h] go_home, [q] graceful_stop
```

默认按键

表 10-3 按键

按键	命令	说明	可重映射?
空格键	pause_resume	暂停/恢复推理循环	保留键
h键	go_home	复位（需暂停态）	支持
q键	graceful_stop	退出进程	保留键

终端恢复

程序退出时自动恢复终端模式（try/finally + atexit + stop()三层保护）。

Unix下终端切到cbreak模式（逐字符读取），退出后恢复cooked模式。

10.3 按键配置

自定义按键映射

```
runtime:
  keyboard_control:
    enabled: true
    keymap:
      "j": go_home          # j → go_home
      "s": state            # s → get_state
      "r": home_ready       # r → 另一个命令实例
```

禁用默认按键

```
keymap:
  "h": ""                   # 禁用默认的 h → go_home
```

保留键（空格和q）不可禁用或重映射，配置中尝试操作会报错

```
ValueError: key 'q' is reserved (graceful_stop) and cannot be remapped or disabled.
```

Merge 语义

自定义keymap**合并**到默认键之上，未提到的键保持默认行为。

例如：

```
keymap:
  "j": go_home
```

空格、h、q仍然有效，只有j被新增。

大小写规则

单字母按键先精确匹配，失败后对调大小写再试：

配置	按键	结果
`"j": go_home`	j	触发
`"j": go_home`	J（Shift+J）	触发（fallback）
`"J": go_home`	j	触发（fallback）
多字符键	—	不适用fallback

10.4 命令与按键关联

关联到已注册的命令实例

keymap的value是命令实例名（YAML commands段的key）

```
hardware:
  config:
    commands:
```

```
    go_home:              # ← 实例名
      type: go_home
      joints: [0, 0, 0, 0, 0, 0]
    state:                # ← 实例名
      type: get_state

runtime:
  keyboard_control:
    enabled: true
    keymap:
      "h": go_home          # → 执行 go_home 实例
      "s": state            # → 执行 state 实例
```

关联到生命周期命令

pause_resume和graceful_stop是内置保留命令，不需要在commands段声明，只能通过保留键触发。

启动时校验

- **配置校验**（from_config）： keymap结构、保留键、类型正确性，有错立即报错退出。
- **命令存在性校验**（from_config）： keymap值对应的命令实例是否存在，有错立即报错退出。

```
keymap:
  "x": unknown_cmd     # ← 如果 unknown_cmd 未注册 → ValueError 退出
```

错误消息示例

```
ValueError: keymap key 'x' → 'unknown_cmd': no such command.
  Available commands: ['go_home', 'state']
  Fix: register a command named 'unknown_cmd' or change the keymap.
```

10.5 安全门控

requires_pause

每个AdapterCommand子类可设置requires_pause。

```
class GoHomeCommand(AdapterCommand):
    requires_pause = True   # 默认值，需暂停态

class EmergencyStopCommand(AdapterCommand):
    requires_pause = False  # 随时可执行
```

行为

表 10-4 行为

命令类型	暂停态	非暂停态
pause_resume	执行	始终执行
graceful_stop	执行	始终执行
requires_pause=True	执行	日志提示（需先暂停）
requires_pause=False	执行	执行

日志示例

```
[KB] h → go_home: requires pause — press space to pause first.
[KB] h → go_home: executed.
```

10.6 常见问题排查

表 10-5 常见问题排查

问题	原因	解决
按键无反应	终端未聚焦。	单击终端窗口。
按键无反应	窗口管理器抢占快捷键。	修改keymap换键。
h键没反应	自定义keymap覆盖了默认。	改为merge或显式添加 "h": go_home。
q键不能改	保留键。	不可更改，使用其他按键。
退出后终端箭头键失效	cbreak未恢复。	已通过三层保护修复（v2）。
pynput not available	旧日志残留。	当前版本不再依赖pynput。
按了按键一直无响应	keymap值指向了不存在的命令实例。	检查启动日志中是否有ValueError。

11 R2C SDK 通过 Entry Point 插件扩展机制集成机器人

11.1 Entry Point 插件扩展机制概述

R2C SDK使用Python标准库importlib.metadata.entry_points机制发现和加载第三方插件。第三方开发者只需在pyproject.toml中声明entry_point，无需修改SDK核心代码。

表 11-1 可扩展入口

Entry Point Group	注册内容	用途
r2c_sdk.adapters	工厂函数	接入新的机器人硬件后端。
r2c_sdk.transformers	IValueTransformer子类	自定义数据转换逻辑。

核心概念

Entry Point插件扩展机制有以下三类核心概念：

AdapterFactory

entry_point注册的是**工厂函数**，不是适配器类本身。

签名

```
def create_xxx_adapter(
    config: Mapping[str, Any], **extra_kwargs: Any
) -> IRobotHardwareAdapter:
    ...
```

- config：YAML中hardware.config的dict。
- extra_kwargs：运行时注入参数（保留字段）。

AdapterRegistry

r2c_sdk.robots.robot_factory.AdapterRegistry类级缓存：

- **懒加载**：仅在首次get(type_name)时import对应工厂。
- **错误缓存**：加载失败后缓存ValueError，后续直接抛错。
- **可用类型**：AdapterRegistry.available_types()列出所有已注册适配器类型。

TransformerRegistry

r2c_sdk.core.transformers.TransformerRegistry内置优先：

- **优先级**：调用方overrides > 内置builtins > entry_points。
- **冲突检测**：模块导入时自动检测同名冲突，warning日志提示。
- **内置优先**：同名时内置覆盖外部。

Entry Point 插件扩展机制向后兼容

- hardware.type：custom + class_path，继续可用（DeprecationWarning）
- xxx_config格式：继续可用（DeprecationWarning）
- class_path 安全检查：load_class()的安全策略不变
- entry_point 路径：安全性由pip安装环节保证，不走 load_class()

11.2 使用第三方适配器（作为用户）

安装第三方适配器包

```
pip install my-robot-r2c-adapter
```

验证安装

```
from r2c_sdk.robots.robot_factory import AdapterRegistry
print(AdapterRegistry.available_types())
# 输出: ['dummy', 'flexiv', 'jaka', 'lerobot', 'my_robot', ...]
```

配置 YAML

```
hardware:
  type: my_robot                    # ← 对应 entry_point name
  config:                           # ← 传给工厂函数的 config
    ip: "192.168.1.xxx"
    port: 502
    joint_names: ["j1", "j2", "j3", "j4", "j5", "j6"]
    commands:                       # 可选：命令配置
      go_home:
        type: go_home
        joints: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

程序化使用

```
from r2c_sdk.robots.robot_factory import RobotFactory

adapter = RobotFactory.create_hardware_adapter({
    "hardware": {
        "type": "my_robot",
        "config": {"ip": "192.168.1.xxx", "port": 502},
    }
})
adapter.connect()
adapter.get_observation()
```

使用第三方 Transformer

安装包含transformer 的包，在YAML mapping规则中按名引用。

```
transforms:
  - name: my_custom_transform    # entry_point name
    config:
      param1: value1
```

11.3 注册自定义适配器（作为开发者）

实现 IRobotHardwareAdapter

```
from dataclasses import dataclass
from typing import Any, Mapping
from r2c_sdk.core.interfaces import IRobotHardwareAdapter

@dataclass
class MyRobotHardwareAdapter(IRobotHardwareAdapter):
    config: Mapping[str, Any]

    def __post_init__(self) -> None:
        from my_robot_commands import MyRobotHomeCommand
        self.register_command_class("go_home", MyRobotHomeCommand)

    def connect(self) -> None:
        """初始化硬件连接。幂等。"""
        ...

    def disconnect(self) -> None:
        """释放硬件资源。"""
        ...

    def get_observation(self) -> Mapping[str, Any]:
        """返回最新原始观测数据。"""
        ...

    def send_action(self, command: Mapping[str, Any]) -> None:
        """下发控制指令到硬件。"""
        ...
```

创建工厂函数

```
def create_my_robot_adapter(
    config: Mapping[str, Any], **extra_kwargs: Any
) -> IRobotHardwareAdapter:
    """Entry_point 工厂。"""
    return MyRobotHardwareAdapter(config=dict(config))
```

声明 entry_point

```
# pyproject.toml
[project]
name = "my-robot-r2c-adapter"
version = "0.1.0"
requires-python = ">=3.12"
dependencies = ["hw-r2c-sdk"]

[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[tool.setuptools]
py-modules = ["my_robot_adapter", "my_robot_commands"]
```

```
[project.entry-points."r2c_sdk.adapters"]
my_robot = "my_robot_adapter:create_my_robot_adapter"
```

包结构

```
my-robot-r2c-adapter/
├── pyproject.toml
├── my_robot_adapter.py      # 工厂函数 + Adapter 类
└── my_robot_commands.py     # 自定义 AdapterCommand 子类
```

实现自定义命令

```
from r2c_sdk.robots.commands.base import AdapterCommand

class MyRobotHomeCommand(AdapterCommand):
    requires_pause = True   # 是否需要在暂停态执行

    def execute(self, **kwargs):
        target = self.config.get("joints")
        if target is not None:
            self.adapter.send_action({"joint_target": list(target)})
```

发布流程

1. 实现适配器和命令。
2. 编写pyproject.toml，声明entry_point。
3. pip install build && python -m build构建。
4. 发布到PyPI或内网pip源。
5. 用户pip install my-robot-r2c-adapter后即可使用。

相关文档

请在本地R2C SDK的文件夹的**examples/third_party_adapter/**获取完整示例。

11.4 注册自定义 Transformer（作为开发者）

实现 IValueTransformer

```
from typing import Any, Mapping
from r2c_sdk.core.interfaces import IValueTransformer

class MyScaleTransformer(IValueTransformer):
    """将输入值乘以配置中的缩放因子。"""

    def transform(self, value: Any, config: Any = None, context: Any = None) -> Any:
        if isinstance(config, Mapping):
            factor = float(config.get("factor", 1.0))
        elif isinstance(config, (int, float)):
            factor = float(config)
        else:
            factor = 1.0
        return float(value) * factor

    @staticmethod
    def validate_config(config: Any) -> None:
        if config is not None and not isinstance(config, (int, float, dict)):
            raise ValueError("config must be a number or {factor: N}")
```

声明 entry_point

```
[project.entry-points."r2c_sdk.transformers"]
my_scale = "my_package.transforms:MyScaleTransformer"
```

在 YAML 中使用

```
transforms:
  - name: my_scale
    config:
      factor: 0.001
```

注意事项

- **注册的是类**，不是实例，TransformerRegistry会在首次查找时实例化。
- **内置优先**：如果注册的name与内置同名，内置保留，第三方被忽略并日志warning。
- 可通过TransformerRegistry.available_transformers()查看当前可用的transformer。

11.5 配置校验与错误排查

Entry_point 错误场景

表 11-2 错误场景

错误	原因	解决
`ValueError: Unsupported hardware.type 'xxx'`	type未注册。	确认包已安装，检查AdapterRegistry.available_types()。
`ValueError: Failed to load adapter entry_point 'xxx'`	工厂函数import失败。	检查模块路径、函数名是否正确、依赖是否安装。
`TypeError: Adapter factory returned ...`	工厂返回值类型错误。	工厂必须返回IRobotHardwareAdapter实例。
entry_point不生效	同名被覆盖。	使用独特前缀命名（如vendor_robot）。
transformer不生效	与内置同名冲突。	检查日志is ignored because a builtin。

从 class_path 偏移

```
# 旧格式（已弃用）
hardware:
  type: custom
  class_path: "mypackage.MyAdapter"
  custom_config: {ip: "192.168.1.xxx"}

# 新格式
hardware:
  type: my_robot
  config: {ip: "192.168.1.xxx"}
```

xxx_config 迁移

```
# 旧格式（已弃用）
hardware:
  type: dummy
  dummy_config: {...}

# 新格式
hardware:
  type: dummy
  config: {...}
```