具身智能开发平台 CloudRobo

SDK 参考

文档版本 01

发布日期 2026-06-30

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，负责连接物理机器人与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. 硬件设备准备
   准备一台上位机和一台可正常上电和通信的机器人。
2. 软件环境配置
   安装LeRobot与本项目R2C SDK所需的软件环境。
3. 机器人接入
   在CloudRobo平台完成真机创建。
4. 端侧配置
   使用R2C SDK将机器人接入CloudRobo平台。
5. 智能体调试
   真机执行任务并完成调试。

2 硬件设备准备

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
机械臂	节卡	Jaka Mini2
机械臂	Universal Robots	UR5e
机械臂	非夕科技	Flexiv Rizon4
人形机器人	星海图	GALAXEA R1
四足机器人	五八智能	天狼Q25

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	

主控设备	操作系统	配置	基础软件（共通）
		USB：至少4到6个，或备个扩展坞	

3 软件环境配置

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

  执行以下命令安装：

  ```
  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 配置conda国内源（可选）。

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**

执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，只需要执行 “conda activate lerobot” 即可。

步骤4 配置Python pypi国内源（可选）。

为了加快安装Python包时依赖包下载速度，可配置pypi的国内源。

自动配置命令（跨平台通用）。

执行以下命令，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：

```
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，执行以下命令。
  ```
  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下执行：

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

Windows下执行：

```
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，请执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

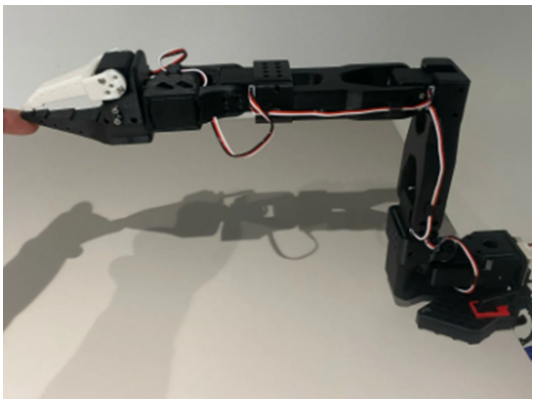
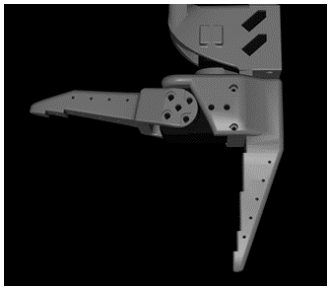
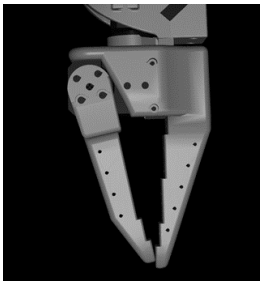
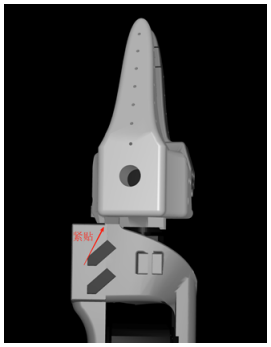
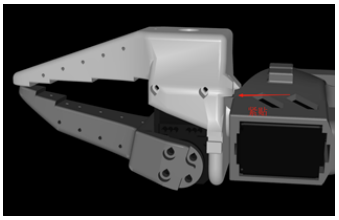
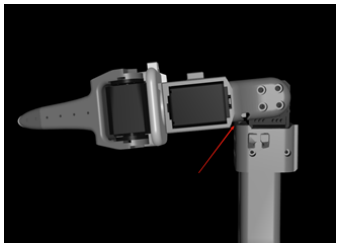
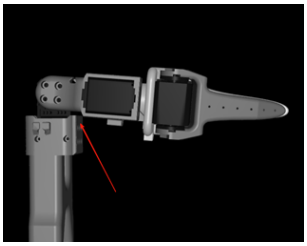
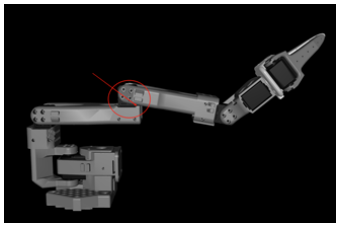
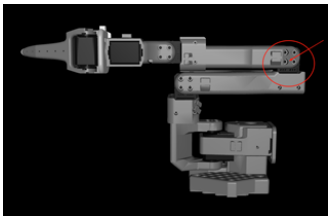
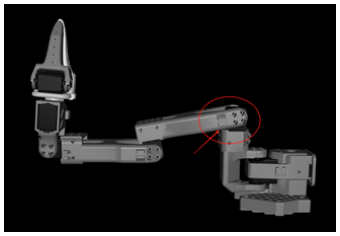
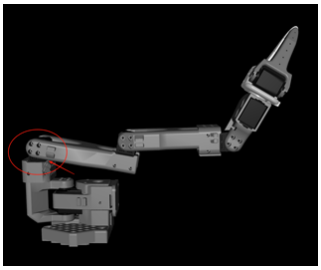
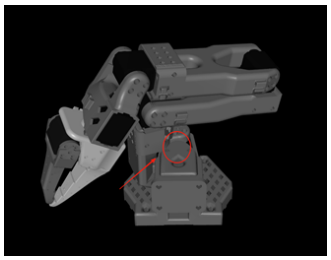
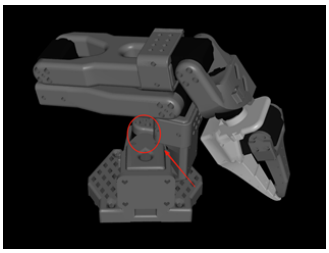
关节角	中间角度示例
Middle position	命令行窗口输出“Move so101_follower to the middle of its range of motion and press ENTER....” 后，机械臂摆成如图所示形态（动爪朝上，固定爪朝下），再按回车键。 

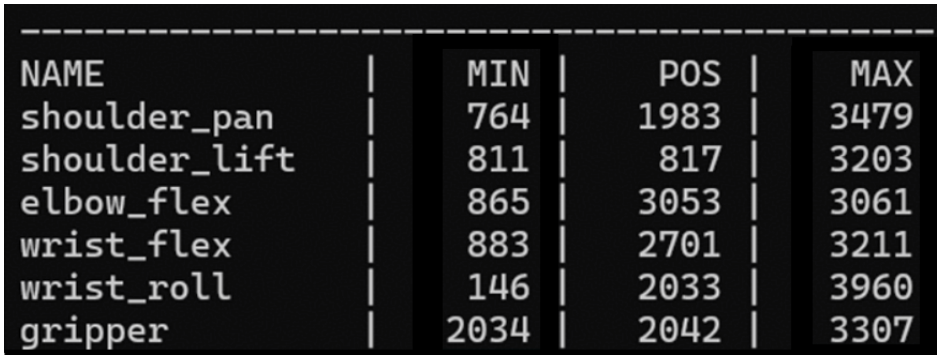
表 3-2 各关节最大最小张开角度

各关节限位	最大张开角度	最小张开角度
6号关节		

各关节限位	最大张开角度	最小张开角度
5号关节		
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	764	1983	3479
shoulder_lift	811	817	3203
elbow_flex	865	3053	3061
wrist_flex	883	2701	3211
wrist_roll	146	2033	3960
gripper	2034	2042	3307

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，请执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。

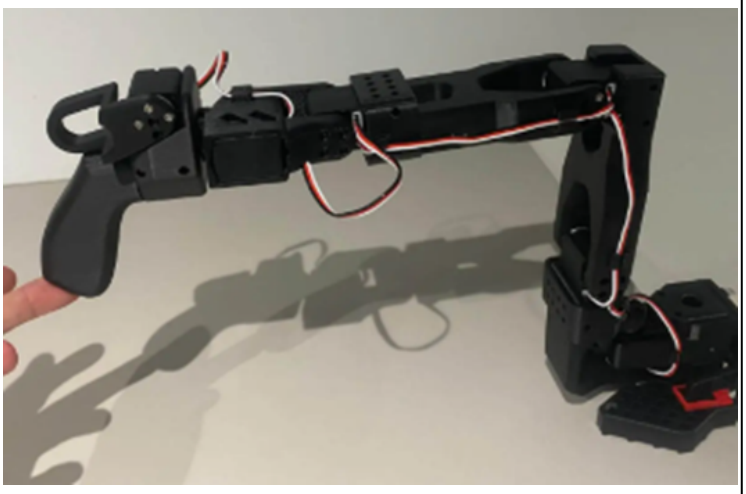
```
lerobot-calibrate --teleop.type=so101_leader --teleop.port=/dev/ttyACM1 --teleop.id=my_leader_arm
```

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 机器人接入

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 端侧配置

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Follow臂的port并记录。

  ```
  lerobot-find-port
  ```

  请根据界面提示拔出Follow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 执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执行），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执行动作。

打开两个终端，并依次分别执行下列命令：

```
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执行动作。

```
python examples/flexiv_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

手动适配本体

本文档以接入SO100为示例。

步骤1 选硬件接入路径（最重要的技术决策）。

基于您的SO100现有软件栈选择：

- 已有LeRobot兼容对象→选hardware.type: lerobot；
- 已有ROS2话题接口→选hardware.type: ros2；
- 只有厂商私有SDK/串口API→选hardware.type: raw_sdk；

步骤2 实现get_observation()/send_action()。

无论选哪条路径，本质都要保证：

- get_observation()返回Mapping/dict；
- send_action(command)能执行设备动作；

步骤3 写device_to_r2c与r2c_to_device映射。

该操作为“协议对齐”阶段：

- 上行：把硬件数据映射为R2C Observation features；
- 下行：把R2C Action features解包成设备命令；

步骤4 补齐相机、夹爪、安全回退。

先通关节，再补图像和高级feature，最后完善fallback。

步骤5 用python -m r2c_sdk.cloudroboclient端到端验证。

通过CLI加载--client-config/bundle + --robot-config验证完整链路。

----结束

必须准备的配置项

适配需要云上机器人配置文件和一份设备配置yaml文件：

- 通信连接配置

  这是SDK连接云侧的基础配置，在CloudRobo平台上创建机器人并下载其配置文件即可。
- robot_so100_config.yaml（设备适配的配置文件，文件名唯一即可）

重点字段如下：

表 5-1 边缘适配核心配置

配置信息	参数及说明
基础元信息	schema_version
runtime	• publish_hz：控制/发布频率 • max_duration_s：最长运行时长
hardware（核心）	• type: lerobot | ros2 | raw_sdk • 以及对应子配置（lerobot_config / ros2_config / raw_sdk_config）
device_to_r2c.mappings（上行映射）	• target_key：R2C features键名 • source_path：从硬件observation提取路径 • modality：vector/image • dtype、encode、quality等

配置信息	参数及说明
r2c_to_device.mappings（下行动作解包）	• source_key：Action中的feature key • target：映射到硬件的字段名称

三种硬件适配路径

- 路径A：lerobot

  适用场景：SO100在你的工程里已经能通过LeRobot风格对象直接读取状态和下发动作。

  你需要做：

  a. 在hardware.type设为lerobot；
  b. 填hardware.lerobot_config（如串口、相机等）；
  c. 调通observations_to_r2c / r2c_to_action；

  优点：最快，几乎不用写新适配器类。

  风险：受LeRobot对SO100支持度约束。
- 路径B：ros2（ROS生态首选）

  适用场景：SO100驱动已经有ROS2 topic/service/action。

  你需要做：

  a. hardware.type: ros2；
  b. 在ros2_config中定义：
     - 订阅（subscriptions或兼容字段joint_states_topic）；
     - 控制发布（command_publishers或兼容字段action_topic）；
     - default_command_publisher / action_targets；
  c. 确保send_action产出的command能转换为目标ROS消息结构。

  优点：与ROS2现有系统耦合最小。

  风险：消息类型、字段名、QoS不匹配会导致“看似连接成功但无数据”。
- 路径C：raw_sdk（厂商SDK / 串口 / 私有协议）

  适用场景：SO100只有厂商Python SDK或自研底层控制API。

  你需要做：

  a. hardware.type: raw_sdk；
  b. 填raw_sdk_config（如串口、波特率、IP、控制模式）；
  c. 在启动时传入raw_sdk_robot_factory，由它根据配置构造真实robot对象；
  d. 使用VendorSDKHardwareAdapter（直接用或继承）；

  优点：灵活、可控。

  风险：需要你自己定义/约束机器人对象API一致性。

6 智能体调试

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。
- 部分模型仅支持执行固定的模型技能，不支持泛化技能；其他模型可执行泛化技能，您可根据需要输入需要执行的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，输入机器人执行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，会重新执行任务。

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执行结果。

执行泛化性技能

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执行泛化技能。

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。

r2c.json 配置说明

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。

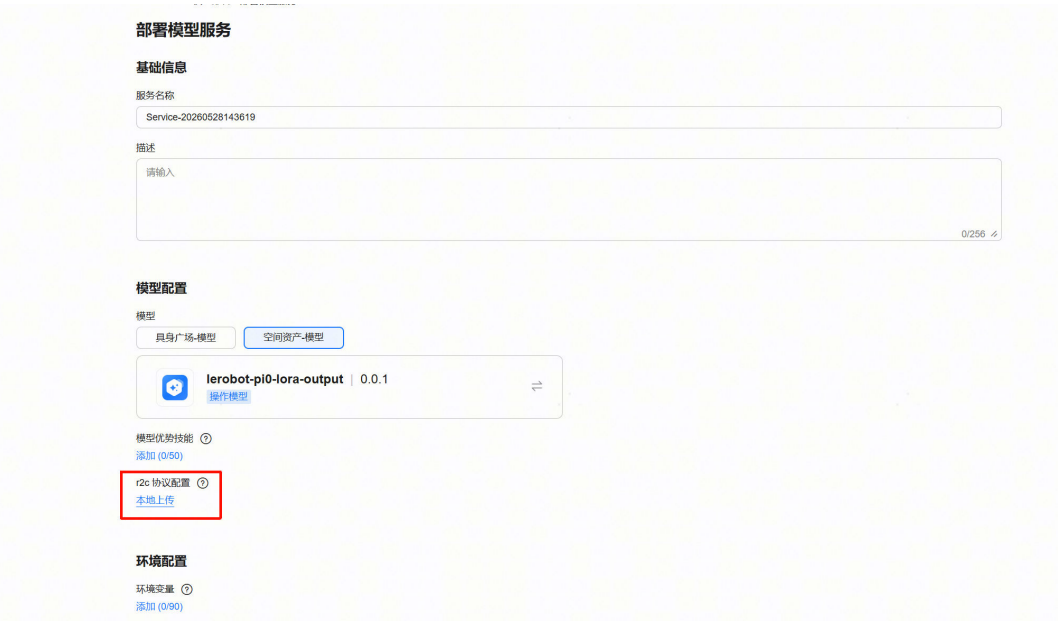
方法一：从模型部署页面中配置

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。

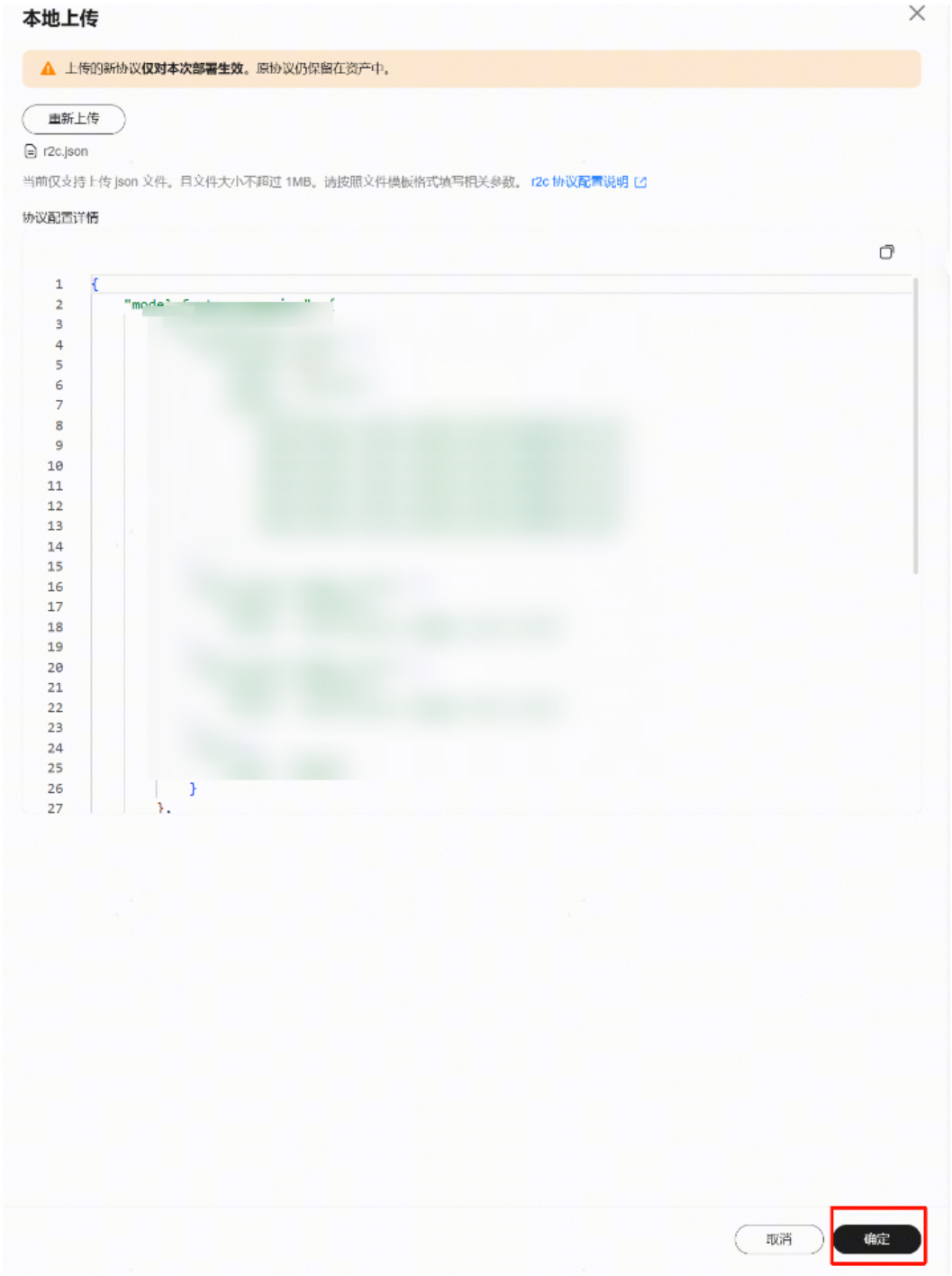
步骤1 在运行管理-模型部署中，可直接上传r2c.json配置。

图 6-1 模型部署页面



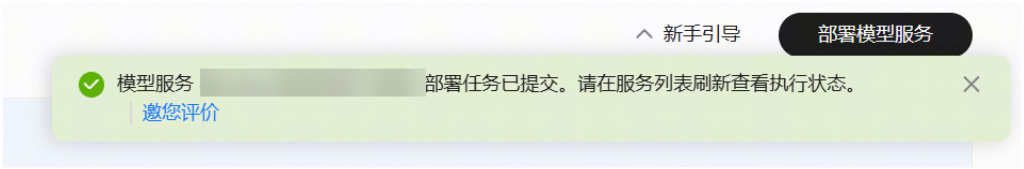
步骤2 上传后，可直接预览已上传的文件。

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

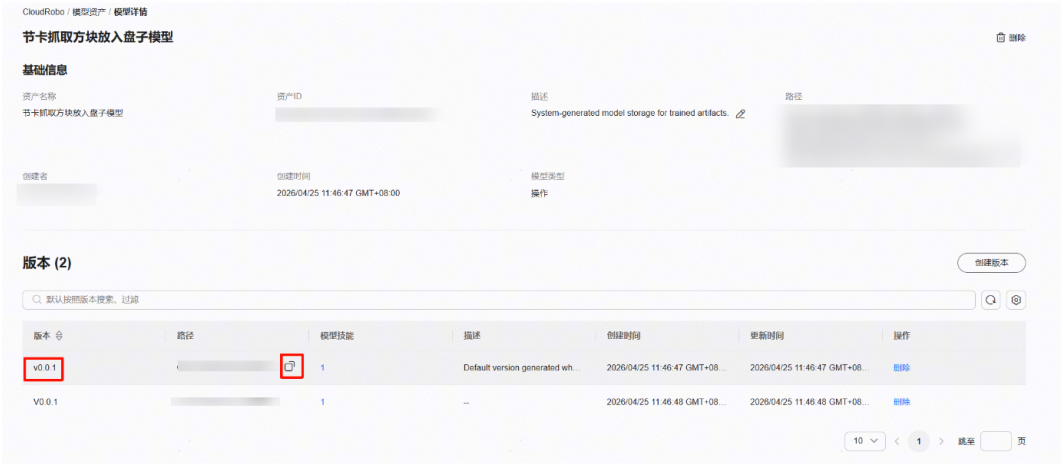
配置文件位于模型资产所在的OBS目录下。

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中目录下。

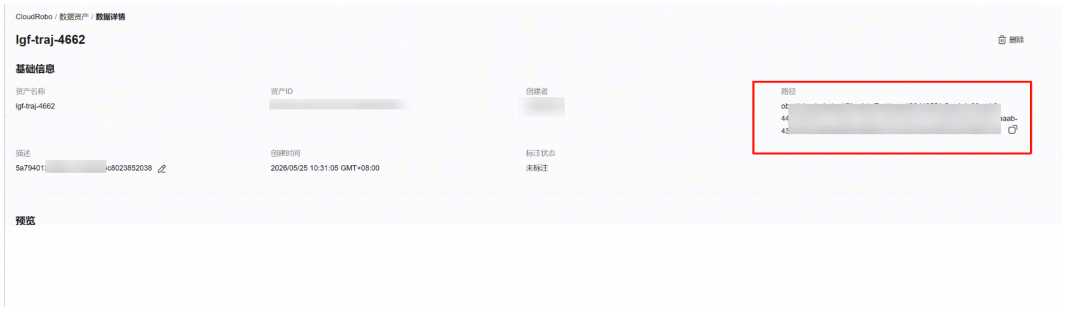
说明

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-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 6-7 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

配置结构概览

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

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
stop_condition	object	否	模型执行停止条件配置，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执行停止条件。

模型特征映射（model_feature_mapping）

该对象包含两个子对象：input_features和output_features，分别定义了模型输入和输出的映射规则。

输入特征（input_features）

该部分定义了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：

特征类型	说明
状态数组 (STATE)	从joint_states、end_effector_states、end_effector_poses提取数值组成状态向量
视觉图像 (VISUAL)	从observations.images.color.*或observations.images.depth.*提取图像数据

特征类型	说明
任务提示 (PROMPT)	从observation.task获取任务描述文本

特征类型分类标准：

- **STATE类型**：满足以下条件
  - 配置中包含values字段（必填）。
- **VISUAL类型**：满足以下条件
  - 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}。
- **PROMPT类型**：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。

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。

注意：状态数组的shape和values数量必须与实际机器人配置一致。

```
"observation.state": {
   "shape": [7],
   "dtype": "float32",
   "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.joint_states.position@{gripper_1}"
   ]
}
```

表 6-1 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），**必须是一维数组**
dtype	"float32"	使用32位浮点数表示（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

配置要求

- shape必须存在且是一维数组（如[7]，不能是[2，3]）。
- dtype必须是整数或浮点类型。
- values列表不能为空。

表 6-2 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}	末端执行器/夹爪位置
end_effector_poses	observation.end_effector_poses.pose@{ee_name}	末端执行器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）

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**：end_effector_poses支持使用[index]提取7D位姿的特定维度（0-6，对应tx, ty, tz, qx, qy, qz, qw）。

占位符替换

- 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。
- **关键**：具体的占位符名称（如joint_1、gripper_1）来自R2C协议中定义的observation_features.joint_states.names和observation_features.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）。

注意：图像的键名（如top、wrist）取决于R2C协议中定义的图像配置。不同机器人可能使用不同的相机名称。

```
"observation.images.top": {
   "dtype": "uint8",
```

```
    "value": "observations.images.color.top"
}
```

表 6-3 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）
value	"observations.images.color.top"	R2C协议中的图像数据路径

表 6-4 支持的图像路径格式

格式	示例	说明
完整路径（4段）	observations.images.color.top	完整观测数据路径
短格式（2段）	color.top	仅包含类型.图像名称
深度图	depth.top	深度图像

路径格式验证

- **4段格式**：observations.images.{color|depth}.{image_name}。
  - 第1段必须以observation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。
- **2段格式**：{color|depth}.{image_name}。
  - 第1段必须是color或depth；
  - 第2段为图像名称。
- 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。

图像名称配置：图像的具体名称（如top、wrist、front）需与R2C协议中的observation_features.images定义保持一致。

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。

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

或：

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

表 6-5 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）

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”，执行失败。

输出特征（output_features）

该部分定义了如何将模型输出的数据映射到机器人的动作指令。

动作输出 (action)

注意：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R2C协议。

```
"action": {
   "chunk_size": 100,
   "shape": [7],
   "values": [
      "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}",
      "actions.joint_states.position@{gripper_1}"
   ]
}
```

表 6-6 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
shape	[N]	每个时间步包含N个值（关节数 + 夹爪数）

字段	值	说明
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r2c.json

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r2c.json中定义的关节和夹爪。
- 关节名称来自action_features.joint_states.names。
- 夹爪名称来自action_features.end_effector_poses。
- 夹爪名称来自action_features.end_effector_states.names。

停止条件（ stop_condition ）

该字段定义模型执行在何时停止的条件，避免无限循环或超时执行。

```
"stop_condition": {
    "max_iter_num": 60,
    "max_run_time": 5
}
```

表 6-7 停止条件参数

字段	类型	值	说明
max_iter_num	integer	60	最大推理调用次数
max_run_time	integer	5	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执行。
- 最多执行60次推理调用或最多运行5分钟。

7 R2C SDK 已支持的机器人

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。

表 7-1 已支持的适配器

适配器类型	hardware.type	适用硬件	关键依赖	配置样例
Dummy仿真模拟器	dummy	无（纯软件）	无	**DummyRobot：Dummy仿真模拟器**
LeRobot驱动	lerobot	SO101	lerobot，draccus	**LeRobot（SO101）**
UR5e RTDE直驱	ur5e_rtde	UR5e+DH夹爪+RealSense	ur-rtde，pyserial，pyrealsense2	**UR5e RTDE（Universal Robots UR5e直驱）**
Flexiv Rizon4	flexiv	非夕Flexiv Rizon4机器人	非夕**Flexiv RDK**	**非夕Flexiv Rizon4机器人**
五八智能 天狼Q25 模拟摇杆控制	custom	五八智能 天狼Q25	无	**五八智能 天狼Q25 模拟摇杆控制**
Zenoh ROS1通用	zenoh_ros1	星海图GALAXEA R1、 任何ROS1平台	zenoh-python	**Zenoh ROS1：通用ROS1机器人（Zenoh Bridge）**
ROS2通用	ros2	Jaka Mini2、 任何ROS2平台	rclpy	**ROS2：通用ROS2机器人平台**

适配器类型	hardware.type	适用硬件	关键依赖	配置样例
回放	playback	无（数据回放）	无	Playback：回放录制数据

通用环境要求

R2C SDK运行的软件环境如表7-2所示。

表 7-2 基础系统要求

项目	要求
Python	>= 3.10（推荐3.12+）
架构	x86_64
操作系统	Linux（推荐Ubuntu 22.04/24.04）

通用CLI启动方式

所有适配器都通过统一的CloudRobo client命令行入口启动：

```
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config config/robot_xxx_config.yaml
```

- cert_xxx.zip接入配置文件是在CloudRobo平台控制台上“运行管理 >机器人”页面中，创建接入机器人后，可下载该接入配置文件。
- 机器人的配置文件robot_xxx_config.yaml在R2C SDK软件包中的config目录下。

表 7-3 常用 CLI 参数

参数	说明	默认值
--client-config	客户端通信配置YAML	config/client_config.yaml
--robot-config	机器人硬件配置YAML	config/robot_dummy_config.yaml
--bundle	平台证书包（zip或目录路径）	无
--duration	运行时长（秒），0=无限运行	0.0
--record	录制观测数据到指定.r2cr文件	无
--log-level	日志级别：DEBUG/INFO/WARNING/ERROR	INFO

参数	说明	默认值
--log-file	日志文件路径（轮转 100MB×5）	无
--hardware-class	自定义适配器类路径（覆盖配置）	无
--translator-class	自定义翻译器类路径（覆盖配置）	无

DummyRobot：Dummy 仿真模拟器

适用场景： 无真实硬件时进行端到端链路联调（ cloudroboclient ↔ cloudrobo(policy server)），或CI自动化测试。

硬件要求： 无。

依赖： 无需额外安装（仅使用numpy，SDK已包含）。

请参考config/robot_dummy_config.yaml文件的配置。

下面是该配置文件中的部分配置内容：

表 7-4 dummy_config 部分

参数	类型	说明	默认值
joint_names	List[str]	关节名称列表	6 个SO101样式的默认名
initial_joint_positions	List[float]	初始关节位置	全零
max_joint_speed_rad_s	float	最大关节速度 (rad/s)	2.5
image_specs	Dict	虚拟相机规格 {name: {h, w, c}}	1个480×640×3 的front相机

Dummy adapter生成数据：

- 关节位置：从initial_joint_positions开始，收到action后向目标移动（受max_joint_speed_rad_s限制）。
- 图像：根据image_specs生成随机像素的numpy数组。

```
# 启动 Dummy 机器人
python -m r2c_sdk.cloudroboclient  --bundle config/cert_xxx.zip --robot-config config/robot_dummy_config.yaml
```

可用的Dummy预设配置：

- config/robot_dummy_config.yaml — SO101样式（6关节，front+wrist相机）。
- config/robot_ur5e_dummy_config.yaml — UR5e样式（7维状态：x/y/z/rx/ry/rz/gripper_position，wrist+third_person相机）。

- config/robot_jaka_dummy_config.yaml — JAKA样式（7关节，top+wrist相机）。

LeRobot（SO101）

适用场景： 使用LeRobot库驱动的SO101系列机械臂（Follower臂）等。

硬件要求：

- SO101系列机械臂（或其他LeRobot兼容硬件）。
- 通过USB串口 (/dev/ttyACM0或类似) 连接。
- USB相机（可选，用于观测）。

表 7-5 系统/软件要求

项目	要求
操作系统	Linux（推荐Ubuntu 22.04/24.04）/ Windows 11
Python	>= 3.10
LeRobot	pip install lerobot
draccus	pip install draccus（LeRobot依赖）
opencv-python	pip install opencv-python（相机采集）

```
# 安装 LeRobot 核心库
pip install lerobot draccus
# 如使用 USB 相机
pip install opencv-python
```

请参考config/robot_so101_lerobot_config.yaml文件的配置。

下面是该配置文件中的部分配置内容：

表 7-6 lerobot_config 部分

参数	类型	说明
type	str	Robot类型，如so101_follower, koch_follower, omx_follower, bi_so_follower
id	str	Robot实例标识符
port	str	串口设备路径（如 /dev/ttyACM0）
calibration_dir	str/null	标定数据目录，null表示不加载标定
cameras	Dict	相机配置，每项包含 {type, index_or_path, width, height, fps}
runtime.robot_connect	Dict	可选，传递给robot.connect(**kwargs)的额外参数

参数	类型	说明
runtime.robot_calibrate	bool	是否在连接时执行标定

```
# Step 1: 确认串口设备
ls /dev/ttyACM* /dev/ttyUSB*
# Step 2: 确认相机设备
ls /dev/video*
# Step 3: 修改配置中的port和相机路径
vim config/robot_so101_lerobot_config.yaml
# Step 4: 启动
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config  config/
robot_so101_lerobot_config.yaml
```

UR5e RTDE（Universal Robots UR5e 直驱）

适用场景： 通过RTDE(Real-Time Data Exchange)协议直接控制UR5e机械臂，无需ROS2。适用于需要低延迟、高频控制的研究与工业场景

硬件要求：

- Universal Robots UR5e机械臂（CB3/e-Series控制器）。
- DH Robotics夹爪（AG-95或兼容型号，通过USB-RS485连接）。
- Intel RealSense D400系列相机（D435/D435I，通过USB连接）。
- 所有设备需在同一局域网。

表 7-7 系统/软件要求

项目	要求
操作系统	Linux (Ubuntu 22.04/24.04)
Python	>= 3.10
ur-rtde	pip install ur-rtde>=1.5
pyserial	pip install pyserial>=3.5
pyrealsense2	pip install pyrealsense2>=2.56

```
# 安装 UR5e 可选依赖
pip install -e ".[ur5e]"
# 或手动安装
pip install ur-rtde>=1.5 pyserial>=3.5 pyrealsense2>=2.56
```

- **UR5e机械臂**：
  - UR5e控制器IP：192.168.5.152（默认，可在配置中修改）。
  - RTDE端口：默认30004。
  - 确认控制器已启动，示教器显示 "RUNNING" 状态。
- **DH夹爪**：
  - 通过USB转RS485适配器连接。

  - 设备路径：/dev/ttyUSBDH_（默认，可在配置中修改）。
- **RealSense相机**：
  - D435I（腕部）：序列号045322075954。
  - D435（第三人称）：序列号134222076113。
  - 通过USB 3.0连接。

请参考config/robot_ur5e_config.yaml文件的配置。

下面是该配置文件中的部分配置内容：

表 7-8 ur5e_config.robot 部分

参数	类型	说明	默认值
robot_ip	str	UR5e控制器IP	"192.168.5.152"
frequency	int	RTDE控制循环频率 (Hz)	500
lookahead_time	float	servoL前瞻时间 (s)	0.1
gain	int	servoL增益	300
max_pos_speed	float	最大位置速度 (m/s)	0.25
max_rot_speed	float	最大旋转速度 (rad/s)	0.6
tcp_offset	float	工具中心点偏移 (m)	0.21
init_joints	List[float]	可选初始关节位置	无
verbose	bool	打印状态信息	true

action_mode：

- "absolute"： 命令值即为目标具体值。
- "delta"： 命令值为相对于当前状态的偏移量。

表 7-9 ur5e_config.gripper 部分

参数	类型	说明	默认值
port	str	串口设备路径	"/dev/ttyUSBDH_"
baudrate	int	波特率	115200
max_width	float	最大开度 (m)	0.08
max_speed	float	最大速度 (m/s)	0.07273

参数	类型	说明	默认值
max_force	float	最大力 (N)	140.0

ur5e_config.cameras： 每项为{name, serial, width, height, fps}。serial值为RealSense相机序列号。

```
# Step 1: 确认 UR5e 控制器可达
ping 192.168.5.152
# Step 2: 确认串口设备
ls /dev/ttyUSBDH_* /dev/ttyUSB*
# Step 3: 确认 RealSense 相机连接（可选）
rs-enumerate-devices
# Step 4: 修改配置中的 IP、串口、相机序列号
vim config/robot_ur5e_config.yaml
# Step 5: 启动
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config config/robot_ur5e_config.yaml
```

- 启动前确保机械臂周围无障碍物。
- 首次运行建议设置较小的max_pos_speed和max_rot_speed。
- 随时可通过键盘e键或Ctrl+C紧急停止。
- 示教器上的急停按钮始终可用。
- 建议首次使用dry_run: true验证数据流，再切换为false执行物理动作。

非夕 Flexiv Rizon4 机器人

硬件设备：

- 仅在ubuntu22.04测试，使用**Flexiv RDK**作为底层实现。
- 使用opencv-python的VideoCapture捕获/dev/video6和/dev/video14两个相机。
- 夹爪为Flexiv-GN01。

配置：

```
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
    # 夹爪配置（从配置读取，不硬编码）
    gripper_name: "Flexiv-GN01"  # 夹爪类型
    gripper_open_width: 0.09    # 夹爪完全打开宽度 (m)
    gripper_close_width: 0.012  # 夹爪完全闭合宽度 (m)
    gripper_velocity: 0.10
    gripper_force_limit: 20.0
    # 力传感器
    zero_ft_on_start: false
    force_limit_n: 20.0
    # 摄像头配置
    cameras:
      wrist:
        source: 6
        width: 640
        height: 480
        fps: 30
      third:
        source: 14
```

```
    width: 640
    height: 480
    fps: 30
```

前置安装命令

```
pip install numpy spdlog flexivrdk
```

启动命令

```
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config config/
robot_flexiv_config.yaml
```

测试命令

在开始测试前，请务必在配置中设置**runtime.dry_run**为**true**，这样只会打印action日志而不执行实际动作，在确认链路正常后再改为**false**。

- 机器人启动指令
  ```
  python -m r2c_sdk.cloudroboclient --project-id test_project --device-id flexiv --client-id robot --robot-config config/robot_flexiv_config.yaml
  ```
- 观测订阅指令
  ```
  python examples/observation_subscriber.py --project-id test_project --device-id flexiv --client-id obs_sub --target-device-id robot
  ```
- 动作发布指令
  ```
  python examples/flexiv_cloud_adapter.py --project-id test_project --device-id flexiv  --client-id obs_sub
  ```

五八智能 天狼 Q25 模拟摇杆控制

硬件设备：

- 仅在ubuntu 22.04测试，使用UDP直接与Q25交互，基于**58智能开放的SDK（C++）**重写。
- 使用摄像头RTSP流捕获main_cam、front_cam和back_cam三个相机。

配置：

```
hardware:
  type: "custom"
  class_path: "r2c_sdk.robots.q25.q25_hardware_adapter.Q25HardwareAdapter"
  custom_config:
    # Q25 Ultra 四足机器人配置
    robot_ip: "192.168.3.20"       # 机器人控制指令IP
    camera_ip: "192.168.1.102"     # 摄像头RTSP流IP
    robot_port: 43893
    listen_port: 43893
    # 观测数据发布频率 (Hz)
    observation_rate_hz: 10.0
    # 摄像头配置
    camera_enabled: true
    camera_locations: ["main_cam", "front_cam", "back_cam"]  # 启用的摄像头列表
    rtsp_port: 8554  # RTSP流端口
    camera_timeout_ms: 5000  # 摄像头连接超时 (毫秒)
    # 调试模式：只接受并打印action，不执行实际动作
    dry_run: true
```

安装命令

```
pip install numpy
```

启动命令

```
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config config/
robot_q25_config_joystick.yaml
```

测试命令

在开始测试前，请务必在配置中设置**runtime.dry_run**为**true**，这样只会打印action日志而不执行实际动作，在确认链路正常后再改为**false**。

- 机器人启动指令
  ```
  python -m r2c_sdk.cloudroboclient --project-id test_project --device-id q25 --client-id robot --robot-config config/robot_q25_config_joystick.yaml
  ```
- 观测订阅指令
  ```
  python examples/observation_subscriber.py --project-id test_project --device-id q25 --client-id obs_sub --target-device-id robot
  ```
- 动作发布指令
  ```
  python examples/q25_cloud_adapter_joystick.py --project-id test_project --device-id q25  --client-id obs_sub
  ```

ROS2：通用 ROS2 机器人平台

适用场景： 任何通过ROS2(Humble/Jazzy/Rolling)运行的机器人平台，包括JAKA Mini2、SO101，仿真MetaEngine提供的仿真机器人等。

硬件要求：

- 任何运行ROS2的机器人平台。
- ROS2 topics发布关节状态和相机图像。

表 7-10 系统/软件要求

项目	要求
操作系统	Linux (Ubuntu 22.04/24.04)
ROS2	Humble (22.04), Jazzy (24.04), 或 Rolling
Python	>= 3.10
rclpy	随ROS2一起安装
message_filters	可选，需pip install message_filters（时间同步观测）

```
# ROS2 依赖由系统 ROS2 安装提供，无需额外 pip 安装
# 确认 rclpy 可用：
python3 -c "import rclpy; print('rclpy OK')"
# 可选：时间同步订阅
pip install message_filters
```

说明

使用ros2 control list_controllers查看可用的控制器列表。

ros2_config.subscriptions：

表 7-11 每项定义订阅一个 ROS2 topic 参数说明

参数	类型	说明	必需
topic	str	ROS2 topic名称	是
msg_type	str	ROS2消息类型（如 sensor_msgs.msg.JointState）	是
qos	int	QoS队列深度	否
store_as	str	在observation dict中的存储键名	是
max_update_hz	float	最大更新频率 (Hz)，用于降采样	否
include_fields	list[str]	仅提取消息中的指定字段	否
field_aliases store_raw_message	dict[str,str] bool	字段名重命名映射 是否存储原始ROS2消息对象	否 否

ros2_config.command_publishers：

表 7-12 每项定义发布一个 ROS2 topic 参数说明

参数	类型	说明
topic	str	发布的目标topic
msg_type	str	ROS2消息类型
qos	int	QoS队列深度

ROS2 命令通道选择：

- **Publisher（推荐）**：直接发布topic，延迟最低。使用default_command_publisher指定。
- **Service**：使用command_services和default_command_service指定。

表 7-13 命令通道参数说明

配置文件	机器人	订阅Topics	发布Topics
robot_so101_ros_config.yaml	SO101（仿真场景）	/joint_states, /top/camera_image_color, /wrist/camera_image_color	/position_controller/commands（Float64MultiArray）

配置文件	机器人	订阅Topics	发布Topics
robot_mini2_ros_config.yaml	JAKA Mini2	/jaka_driver/joint_position, /gripper_state, /camera_top/image_raw/compressed, /camera_wrist/image_raw/compressed	/robot_command_servo_j（JointState）, /motor_control（step_motor/Motor）

Zenoh ROS1：通用 ROS1 机器人（Zenoh Bridge）

Zenoh ROS1适配器适用于任何通过zenoh-bridge-ros1接入网络的ROS1机器人，并无需在上位机安装rospy或任何ROS1依赖，该适配器直接通过Zenoh数据平面订阅/发布ROS1消息。

Zenoh ROS1适配器硬件要求：

- 任意运行ROS1的机器人平台，如R1 (HDAS)双臂机器人等。
- 机器人端运行zenoh-bridge-ros1实例。
- 网络可达Zenoh peer或router。

表 7-14 上位机系统/软件要求

项	要求
操作系统	Linux（推荐Ubuntu 22.04/24.04）
Python	>= 3.10
zenoh-python	`pip install zenoh`
opencv-python	可选，用于解码CompressedImage（pip install opencv-python）

适配器通过zenoh-bridge-ros1与ROS1系统通信，工作流程为：

```
ROS1 机器人 ←→ zenoh-bridge-ros1 ←→ Zenoh 网络 ←→ ZenohRos1HardwareAdapter
```

需确保机器人端zenoh-bridge-ros1正常运行，且其Zenoh配置（mode、router address）与适配器一致。

表 7-15 支持的订阅消息类型（内置）

消息类型	用途
sensor_msgs/CompressedImage	相机图像（自动解码为JPEG bytes）
sensor_msgs/Image	相机图像（解码为numpy数组）
sensor_msgs/JointState	关节状态

消息类型	用途
std_msgs/Float32	单精度浮点（如夹爪位置）
std_msgs/Float32MultiArray	浮点数组

表 7-16 支持的发布消息类型（内置）

消息类型	encoder	用途
hdas_msg/motor_control	motor_control	电机控制（含位置/速度/力矩参数）
std_msgs/Float32	float32	浮点控制（如夹爪位置）
任意自定义	raw_bytes	透传原始字节（非标准消息）

自定义消息类型： 可通过ros1_type_overrides注册新的ROS1消息类型，需提供datatype和md5值。

请参考config/robot_r1_zenoh_ros1_config.yaml文件的配置。

下面是该配置文件中的**配置参数说明**：

表 7-17 zenoh_ros1_config 顶层参数

参数	类型	说明	默认值
mode	str	Zenoh连接模式："peer"(P2P)或"client"(连接router)	"peer"
connect_endpoints	List[str]	client模式下的router地址列表	[]
namespace	str	Zenoh key中的命名空间段（与bridge默认行为一致）	""
rate	float	控制循环目标频率(Hz)	30.0

表 7-18 scouting 部分

参数	类型	说明	默认值
multicast.enabled	bool	是否启用组播发现（client模式建议禁用）	true

表 7-19 subscriptions 每项

参数	类型	说明	必需
ros_topic	str	ROS1 topic名称	是
msg_type	str	ROS1消息类型	是
store_as	str	在observation dict中的存储键名	否（默认使用订阅名）

表 7-20 publishers 每项

参数	类型	说明	必需
ros_topic	str	发布的ROS1 topic名称	是
msg_type	str	ROS1消息类型	是
encoder	str	序列化方式：motor_control/float32/raw_bytes	是

ros1_type_overrides（可选）：

用于注册内置类型表之外的ROS1消息类型。每项为：

```
ros1_type_overrides:
  "custom_msgs/MyMessage":
    datatype: "custom_msgs/MyMessage"
    md5: "abc123..."
```

使用步骤

```
# Step 1: 安装依赖
pip install zenoh opencv-python
# Step 2: 确保机器人端zenoh-bridge-ros1正常运行
#（在机器人端启动bridge，确认其Zenoh配置与本适配器一致）
# Step 3: 修改配置文件中的connect_endpoints、subscriptions、publishers
vim config/robot_r1_zenoh_ros1_config.yaml
# Step 4: 启动适配器
python -m r2c_sdk.cloudroboclient --bundle config/cert_xxx.zip --robot-config config/
robot_r1_zenoh_ros1_config.yaml
```

适配其他ROS1机器人

Zenoh ROS1适配器高度可配置，更换机器人时只需修改subscriptions和publishers中的ros_topic：

- **修改订阅：** 将subscriptions中的ros_topic改为目标机器人实际的topic名称（如/joint_states、/camera/rgb/image_raw/compressed）。
- **修改发布：** 将 publishers中的ros_topic改为目标机器人接受的控制topic。
- **自定义消息类型：** 如使用了非内置类型，通过ros1_type_overrides注册。
- **调整translator映射：** 修改device_to_r2c和r2c_to_device中的source_path/target以匹配新的数据字段名。

Playback：回放录制数据

Playback适配器适用于离线调试cloudroboclient ↔ cloudrobo(policy server)链路，不需要真实硬件，通过回放之前录制的观测数据来模拟机器人。

该适配器无需额外安装（使用SDK内置的safe_serialization）依赖，请参考config/robot_playback_config.yaml文件的配置。

表 7-21 robot_playback_config.yaml 部分配置内容

参数	类型	说明	默认值
recording_file	str	录制文件的路径（.r2cr格式）	**必需**
loop	bool	播完后是否循环回开头	true

说明

- Playback adapter的send_action()是空操作（no-op），收到cloud的action也不会执行。
- device_to_r2c和r2c_to_device的映射配置需与被录制的真实机器人的配置保持一致。

录制与回放

R2C SDK支持将观测数据录制为.r2cr二进制文件（安全格式，JSON结构 + 原始numpy字节，无代码执行风险），用于后续离线回放和调试。

- **录制**：在cloudroboclient启动时添加--record参数。
  ```
  python -m r2c_sdk.cloudroboclient --client-config config/client_config.yaml --robot-config config/robot_ur5e_config.yaml --record recordings/ur5e_demo.r2cr
  ```
  正常运行时按Ctrl+C停止，录制文件会自动保存。
- **回放**：使用robot_playback_config.yaml配置回放已录制的数据。
  ```
  python -m r2c_sdk.cloudroboclient --client-config config/client_config.yaml --robot-config config/robot_playback_config.yaml
  ```
  确保配置文件中playback_config.recording_file指向正确的录制文件。