

机器 人操 作 系 统

机 电 工 程 学 院

机器 人智 能制 造研 究团 队

2025 · 秋



课程考核

总分 = 平时成绩 (20%) + 实践成绩 (40%) + 大作业成绩 (40%)

平时成绩 = 考勤 (70%) + 课堂表现 (30%)

大作业成绩 (小组完成)

= 基础内容 (50%) + 创新性 (20%) + 报告内容 (15%) + 展示讲解 (15%)



课程安排

第9周周四 (10.30) 5-8节 → 第12周周四 (11.20) 5-8节

大作业安排：大作业要求 & ROS大作业展示PPT 要求 & 大作业样例（学在西电）

- 大作业选题提交 10.23
- 大作业项目研究方案提交 10.30
- 大作业项目录制视频提交 11.18
- 大作业项目汇报 11.20
- 大作业成果提交 11.27
- 大作业个人报告 11.27

课程 安排

- 01 第一章 ROS概述与环境搭建
- 02 第二章 ROS通信机制
- 03 第三章 ROS架构与运行管理
- 04 第四章 ROS常用组件
- 05 第五章 机器人建模与仿真
- 06 第六章 ROS进阶功能



第六章

ROS 进阶功能



机器人导航场景:

节点A：发布目标位置； 节点B：控制机器人移动

服务（Service）通信是否可行？

在任务执行过程中，客户端无法获得实时反馈，只能等待最终结果。

导航需要实时监控与中断，比如“暂停”“重新规划路径”。

Service 没有中途反馈机制，会造成“假死”现象。

更合理的方案应该是：导航过程中，可以连续反馈当前机器人状态信息，当导航终止时，再返回最终的执行结果。

在ROS中，该实现策略称之为：action 通信。

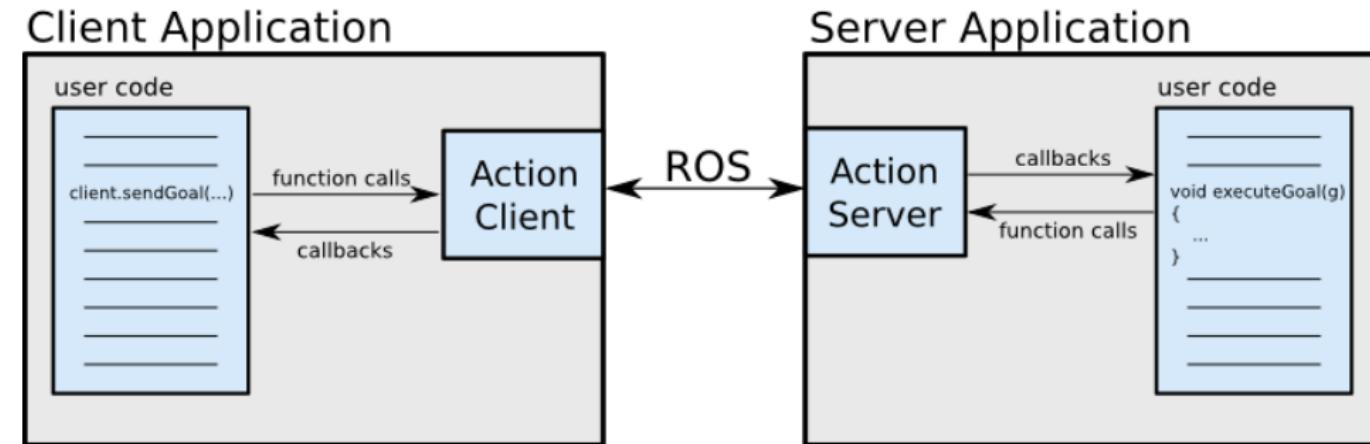


action通信

在ROS中提供了actionlib功能包集，用于实现 Action 通信。Action 是一种类似于服务通信的实现。

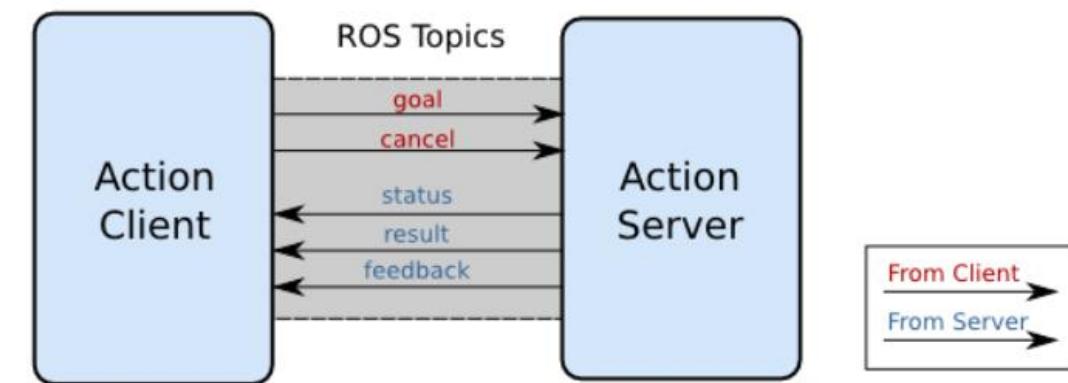
但是不同的是，在请求和响应的过程中，服务端还可以连续的反馈当前任务进度，客户端可以接收连续反馈并且还可以取消任务。

action结构图解:



action通信接口图解:

Action Interface



一般适用于耗时的请求响应场景,用以获取连续的状态反馈

- goal: 目标任务;
- cancel: 取消任务;
- status: 服务端状态;
- result: 最终执行结果(只会发布一次);
- feedback: 连续反馈(可以发布多次)。

`actionlib` 是 ROS 中用于实现**异步任务管理**的一套工具包，提供了**客户端-服务器模式**的通信机制。它的设计主要用于处理那些需要较长时间完成的任务，同时允许用户在任务执行的过程中发送进度更新、取消任务或者接收结果。

`actionlib` 是对 ROS 基本通信机制（如话题、服务）的扩展和补充，解决了服务（service）由于同步阻塞而无法适应长时间任务的问题。

actionlib 的基本结构

1. Action 定义

每个 `action` 包括三个部分：

- **Goal** (目标)：指定任务的输入参数。
- **Feedback** (反馈)：任务执行过程中的实时进度信息。
- **Result** (结果)：任务完成后的最终输出。

2. Action 客户端 (Action Client)

客户端负责发送目标 (goal)，接收反馈 (feedback) 和最终结果 (result)，并可在任务执行过程中取消任务。

3. Action 服务器 (Action Server)

服务器负责接收客户端发送的目标 (goal)，执行具体任务，并向客户端发送反馈 (feedback) 和最终结果 (result)。

应用场景

1. 移动机器人导航

在移动机器人导航中（如 `move_base` 包），机器人需要执行较长时间的路径规划和导航任务。

`actionlib` 被用于处理导航目标点任务，允许客户端发送目标点并接收导航的实时进度反馈。

2. 机械臂控制

控制机械臂执行较复杂的动作（如抓取和放置物体），任务可能需要较长时间，且需要实时监控机械臂的动作状态。

3. 无人机任务

无人机的飞行任务（如送货、巡检），通常需要精确的目标输入、实时状态更新以及随时取消任务的功能。

4. 图像处理

在一些耗时的图像处理任务（如 3D 点云生成）中，可以通过 `actionlib` 实现实时进度监控和任务管理。

5. 多机器人协作

在多机器人系统中，机器人可能需要并行完成复杂的任务，而这些任务通常需要精确的反馈和灵活的控制。

与其他通信方式的比较

特性	ROS Topic	ROS Service	ROS Action (actionlib)
通信模式	发布/订阅	请求/响应	客户端/服务器
是否支持异步	是	否	是
是否支持实时反馈	否	否	是
是否支持取消任务	否	否	是
适用场景	数据流传输	短时间同步任务	长时间异步任务

示例一：

创建两个ROS 节点，服务器和客户端，客户端可以向服务器发送目标数据N(一个整型数据)服务器会计算 1 到 N 之间所有整数的和,这是一个循环累加的过程，返回给客户端，这是基于请求响应模式的，又已知服务器从接收到请求到产生响应是一个耗时操作，每累加一次耗时0.1s，为了良好的用户体验，需要服务器在计算过程中，每累加一次，就给客户端响应一次百分比格式的执行进度，使用 action实现。

任务描述

实现基于 `actionlib` 的两个 ROS 节点：

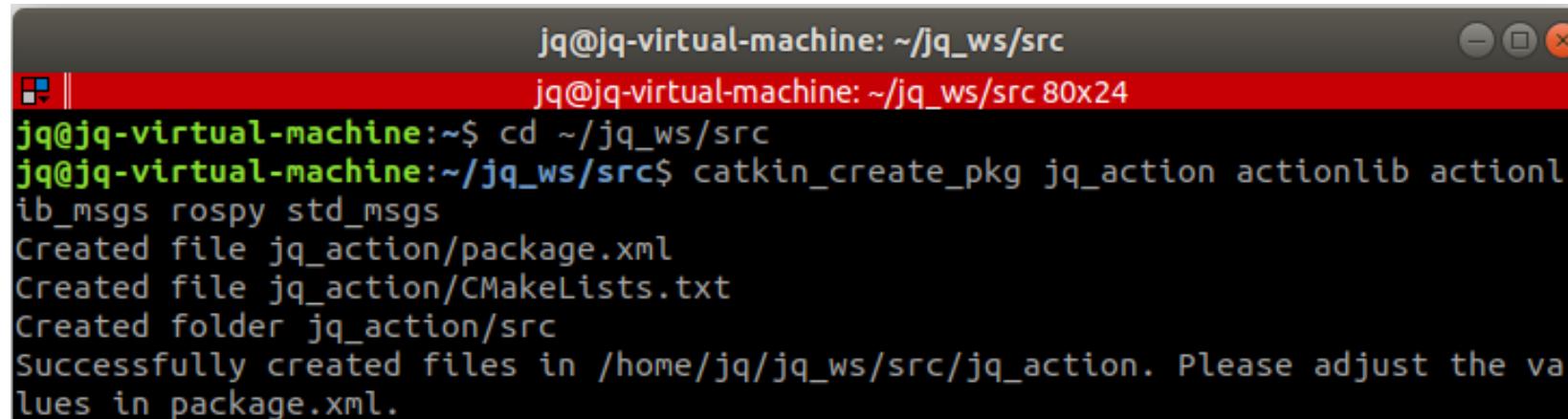
1. **客户端**: 发送目标数据 N 。
2. **服务器**: 接收 N ，计算 1 到 N 的累加和，在计算过程中，每次累加时耗时 $0.1s$ ，并将执行进度（百分比）返回客户端，最终返回累加结果。

1. 创建 ROS 包

生成 ROS 包

在工作空间中生成一个名为 `jq_action` 的 ROS 包:

```
cd ~/jq_ws/src  
catkin_create_pkg jq_action actionlib actionlib_msgs rospy std_msgs
```



The screenshot shows a terminal window with a dark background and light-colored text. The title bar says "jq@jq-virtual-machine: ~/jq_ws/src". The terminal prompt is "jq@jq-virtual-machine:~\$". The user runs the command "catkin_create_pkg jq_action actionlib actionlib_msgs rospy std_msgs". The terminal then displays the output: "Created file jq_action/package.xml", "Created file jq_action/CMakeLists.txt", "Created folder jq_action/src", and "Successfully created files in /home/jq/jq_ws/src/jq_action. Please adjust the values in package.xml.".

```
jq@jq-virtual-machine:~/jq_ws/src  
jq@jq-virtual-machine:~/jq_ws/src 80x24  
jq@jq-virtual-machine:~$ cd ~/jq_ws/src  
jq@jq-virtual-machine:~/jq_ws/src$ catkin_create_pkg jq_action actionlib actionlib_msgs rospy std_msgs  
Created file jq_action/package.xml  
Created file jq_action/CMakeLists.txt  
Created folder jq_action/src  
Successfully created files in /home/jq/jq_ws/src/jq_action. Please adjust the values in package.xml.
```

2. 定义 Action 文件

创建 Action 文件

在包的 `action` 目录下创建 `Sum.action` 文件:

```
mkdir -p ~/jq_ws/src/jq_action/action  
gedit ~/jq_ws/src/jq_action/action/Sum.action
```

添加以下内容: 也可以在VSCode中添加

```
1 # 目标 (Goal)  
2 int32 target_number # 输入目标数据 N  
3  
4 ---  
5 # 结果 (Result)  
6 int32 result_sum # 累加结果  
7  
8 ---  
9 # 反馈 (Feedback)  
10 int32 percent_complete # 当前执行进度百分比
```

 `sum_server.py`

 `sum_client.py`

 `Sum.action`

3. 配置 CMakeLists.txt

3.1 修改 CMakeLists.txt 文件

打开 jq_action 包中的 CMakeLists.txt 文件：

添加 add_action_files

确保指定了 Sum.action :

添加 generate_messages

确保生成消息依赖：

```
add_action_files(  
    FILES  
    Sum.action  
)
```

```
generate_messages(  
    DEPENDENCIES  
    std_msgs  
    actionlib_msgs  
)
```

配置 catkin_package

确保包含必要的依赖：

```
catkin_package(  
    CATKIN_DEPENDS actionlib_msgs rospy std_msgs  
)
```

4. 配置 package.xml

4.1 修改 package.xml 文件

打开 package.xml 文件：

4.2 添加依赖

在 <build_depend> 和 <exec_depend> 中添加以下内容：

```
<build_depend>actionlib_msgs</build_depend>
<exec_depend>actionlib_msgs</exec_depend>
<build_depend>std_msgs</build_depend>
<exec_depend>std_msgs</exec_depend>
<build_depend>rospy</build_depend>
<exec_depend>rospy</exec_depend>
```

5. 编译工作空间

```
cd ~/jq_ws  
catkin_make  
source devel/setup.bash
```

检查 `SumResult` 是否正确生成：

```
rosmsg show jq_action/SumResult
```

输出应为：

```
int32 result_sum
```

6. 编写服务器节点

6.1 创建服务器脚本

在 `jq_action` 的 `scripts` 文件夹下创建服务器脚本：

```
mkdir -p ~/jq_ws/src/jq_action/scripts  
nano ~/jq_ws/src/jq_action/scripts/sum_server.py
```

6.2 服务器脚本内容

也可以在VSCode中添加

6.3 设置可执行权限

```
chmod +x ~/jq_ws/src/jq_action/scripts/sum_server.py
```

7. 编写客户端节点

7.1 创建客户端脚本

在 `jq_action` 的 `scripts` 文件夹下创建客户端脚本：

```
nano ~/jq_ws/src/jq_action/scripts/sum_client.py
```

7.2 客户端脚本内容

也可以在VSCode中添加

7.3 设置可执行权限

```
chmod +x ~/jq_ws/src/jq_action/scripts/sum_client.py
```

8. 测试运行

8.1 启动 roscore

在一个终端中启动 roscore :

8.2 启动服务器节点

在另一个终端启动服务器：

8.3 启动客户端节点

在另一个终端启动客户端：

The screenshot shows two terminal windows on a Linux desktop. The top window, titled 'jq@jq-virtual-machine: ~', displays the output of running 'roscore'. It shows log messages about logging to a file, checking disk usage, starting a roslaunch server at http://jq-virtual-machine:43057, and the version of ros_comm (1.14.13). It also shows a 'SUMMARY' section and 'PARAMETERS' section with two entries: '/rosdistro: melodic' and '/rosversion: 1.14.13'. The bottom window, also titled 'jq@jq-virtual-machine: ~', shows the output of running 'rosrun jq_action sum_client.py'. It logs the start of the Sum Action Server, receiving a target number of 10, and calculating a result of 55. The right side of this window is highlighted with a red rectangle. The left window shows the progress of sending a goal to the server, with messages indicating 10% to 100% progress and finally the final result of 55.

```
jq@jq-virtual-machine:~$ roscore
... logging to /home/jq/.ros/log/1f3e4430-be0b-11ef-af74-000c2902c05e/roslaunch-jq-virtual-machine-38165.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://jq-virtual-machine:43057/
ros_comm version 1.14.13

SUMMARY
=====

PARAMETERS
* /rosdistro: melodic
* /rosversion: 1.14.13

NODES
jq@jq-virtual-machine:~ 54x16
jq@jq-virtual-machine:~$ rosrun jq_action sum_server.py
[INFO] [1734614078.198285]: Sum Action Server started!
[INFO] [1734614086.896668]: Received target number: 10
[INFO] [1734614087.984911]: Sum result: 55
]

jq@jq-virtual-machine:~ 59x16
jq@jq-virtual-machine:~$ rosrun jq_action sum_client.py
[INFO] [1734614086.761730]: Waiting for the server...
[INFO] [1734614086.880472]: Sending goal: 10
[INFO] [1734614087.003852]: Progress: 10%
[INFO] [1734614087.112846]: Progress: 20%
[INFO] [1734614087.222484]: Progress: 30%
[INFO] [1734614087.332458]: Progress: 40%
[INFO] [1734614087.440774]: Progress: 50%
[INFO] [1734614087.551483]: Progress: 60%
[INFO] [1734614087.658652]: Progress: 70%
[INFO] [1734614087.768305]: Progress: 80%
[INFO] [1734614087.876050]: Progress: 90%
[INFO] [1734614087.987023]: Progress: 100%
[INFO] [1734614087.999630]: Final result: 55
jq@jq-virtual-machine:~$
```

示例二：

使用 **ROS Actionlib** 实现小乌龟在 `turtlesim` 仿真环境中绘制一个对称的轨迹，具体任务如下：

任务功能

1. 多目标点路径规划：

- 客户端发送多个目标点的坐标给服务器，目标点包含：
 - 一个等边三角形的三个顶点。
 - 与该三角形关于 Y 轴对称的三个顶点。
- 小乌龟按照目标点列表依次移动，完成路径的绘制。

2. 实时反馈：

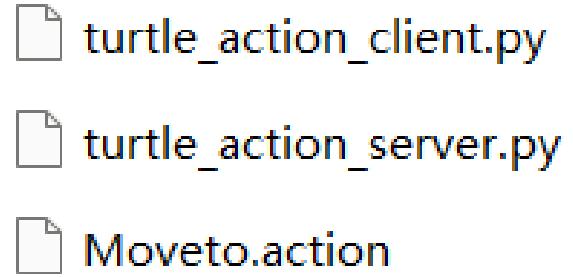
- 服务器在小乌龟移动过程中，实时计算当前位置，并反馈：
 - 当前目标点的索引。
 - 小乌龟的当前位置坐标 (`x`, `y`)。
- 客户端显示实时反馈信息，便于监控任务进展。

3. 任务完成反馈：

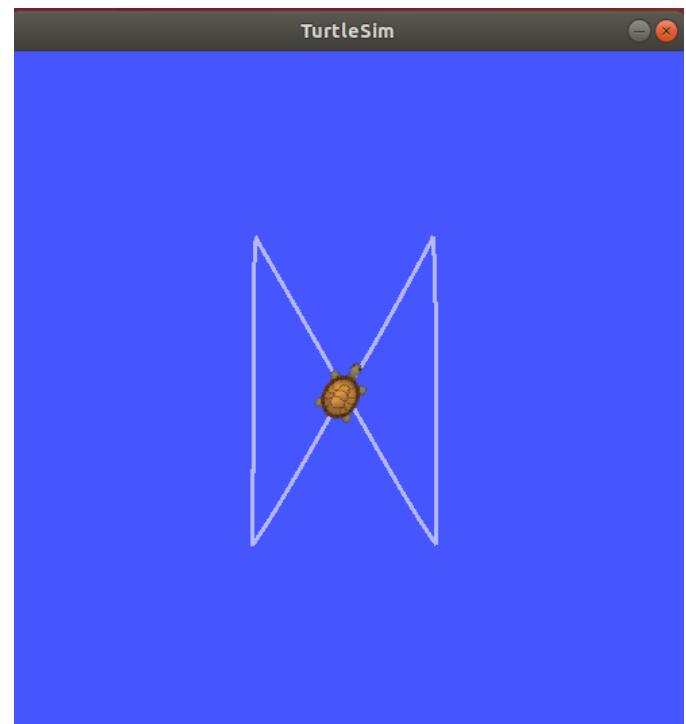
- 当小乌龟依次到达所有目标点后，服务器向客户端返回任务完成状态（成功/失败）。

4. 任务中断：

- 在任务执行过程中，客户端可以随时取消任务，服务器会停止当前操作。



修改 `CMakeLists.txt` 文件



```
jq@jq-virtual-machine: ~/jq_ws
roscore http://jq-virtual-machine:11311/ 53x13
* /rosversion: 1.14.13

NODES

auto-starting new master
process[master]: started with pid [42555]
ROS_MASTER_URI=http://jq-virtual-machine:11311/
setting /run_id to 967c59fc-be0f-11ef-af74-000c2902c0
5e
process[rosout-1]: started with pid [42566]
started core service [/rosout]

jq@jq-virtual-machine: ~/jq_ws 56x13
jq@jq-virtual-machine:~/jq_ws$ rosrun turtlesim turtlesi
m_node
[ INFO] [1734620432.641254076]: Starting turtlesim with
node name /turtlesim
[ INFO] [1734620432.643904849]: Spawning turtle [turtle1]
at x=[5.544445], y=[5.544445], theta=[0.000000]

jq@jq-virtual-machine: ~/jq_ws 53x15
jq@jq-virtual-machine:~/jq_ws$ rosrun jq_action turtl
e_action_server.py
[INFO] [1734620463.355598]: Moving to target 0: (7.00
, 8.10)
[INFO] [1734620468.656116]: Moving to target 1: (7.00
, 2.90)
[INFO] [1734620476.456682]: Moving to target 2: (5.50
, 5.50)
[INFO] [1734620482.055893]: Moving to target 3: (4.00
, 8.10)
[INFO] [1734620486.356743]: Moving to target 4: (4.00
, 2.90)
[INFO] [1734620494.257194]: Moving to target 5: (5.50
, 5.50)
[INFO] [1734620499.957292]: All goals completed!

jq@jq-virtual-machine: ~/jq_ws 56x15
5.34)
[INFO] [1734620499.357570]: Currently at goal 5: (5.42,
5.35)
[INFO] [1734620499.457189]: Currently at goal 5: (5.42,
5.37)
[INFO] [1734620499.556978]: Currently at goal 5: (5.43,
5.38)
[INFO] [1734620499.657695]: Currently at goal 5: (5.44,
5.39)
[INFO] [1734620499.757006]: Currently at goal 5: (5.45,
5.40)
[INFO] [1734620499.857474]: Currently at goal 5: (5.45,
5.41)
[INFO] [1734620499.981034]: Result: success: True
jq@jq-virtual-machine:~/jq_ws$
```

使用 ROS Service 的实现

核心逻辑

1. 定义一个 ROS Service:

- 客户端发送单个目标点的请求，服务器接收目标点并控制小乌龟运动到目标点。
- 每个目标点完成后，客户端发送下一个目标点。

2. 多目标点的处理：

- 通过客户端逻辑按顺序发送目标点，逐个调用 Service。
- 客户端需要主动管理目标点队列，逐步发送目标点。

3. 实时反馈：

- 使用额外的 Topic 订阅小乌龟的位置（`/turtle1/pose`），在客户端实时显示当前位置。

4. 任务完成：

- 最后一个目标点到达后，客户端结束任务。

对比分析

功能	Actionlib 实现	请求-响应实现 (Service)
实时反馈	内置支持，通过反馈机制实时更新客户端状态。	不直接支持，需要客户端手动订阅 <code>/turtle1/pose</code> 。
任务中断	客户端可随时取消任务，Actionlib 自动处理。	不直接支持，需客户端逻辑中断后续目标点的发送。
多目标任务	一次性发送所有目标点，服务器按顺序处理。	客户端逐个目标点发送，每个点都需独立服务调用。
实现复杂度	较高，需要定义 <code>.action</code> 文件和反馈逻辑。	较低，只需定义一个简单的 Service 和逻辑即可。

如果不用 **Actionlib**，只用 **Service** 的方式也能实现相同的轨迹运动，但以下功能会受到影响：

- 实时反馈**：需要额外订阅 `/turtle1/pose` 话题。
- 多目标任务**：需要客户端管理目标点队列，逐个发送请求。
- 任务中断**：需要手动中断目标点的发送。