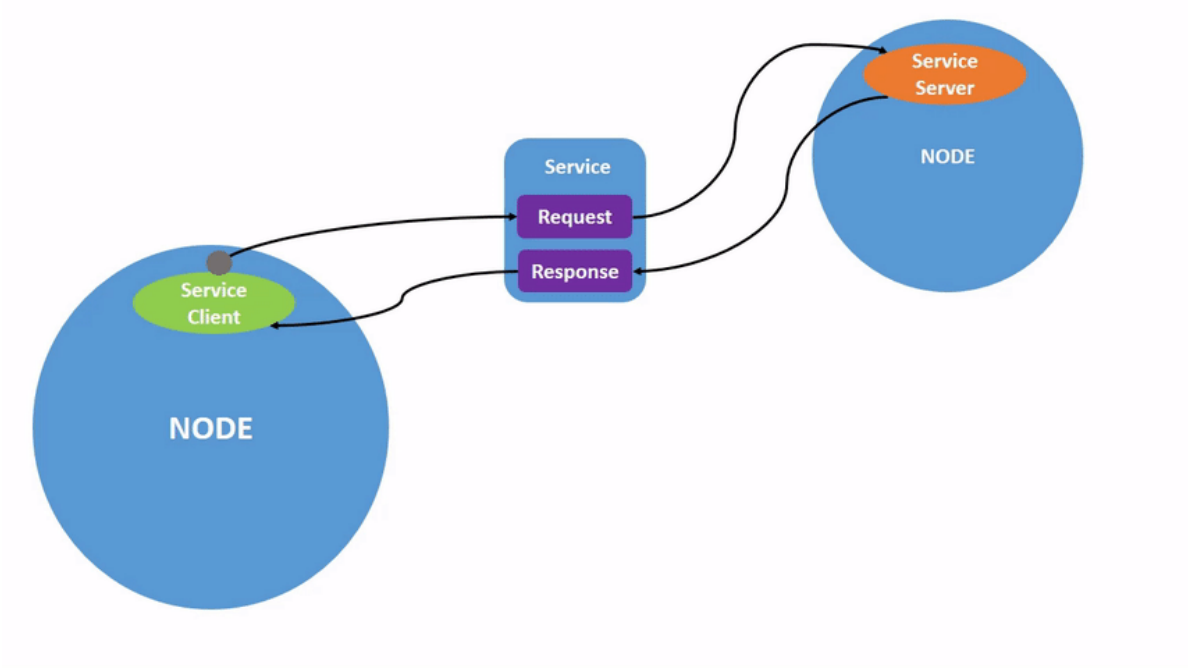
12.ROS2 action communication client

1. Introduction to action communication

Action communication is a communication model with continuous feedback. Between the communicating parties, the client sends request data to the server, and the server responds to the client.However, during the process from the server receiving the request to generating the final response, it will send continuous feedback information to the client.

The Action Communication client/server model is as follows:



2. Case introduction

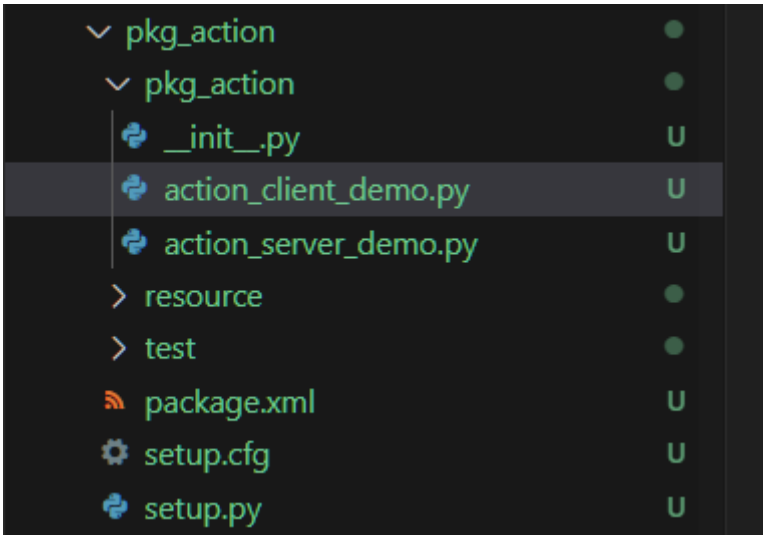
The action client submits an integer data N, the action server receives the request data and accumulates all integers between 1 and N, and returns the final result to the action client.And each time it is accumulated, the current operation progress is calculated and fed back to the action client.

This case is located in the factory docker container. The source code location is:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/pkg_interfaces
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/pkg_action
```

3. Client implementation

Create a new file [action_client_demo.py] in the same directory as [action_server_demo.py]

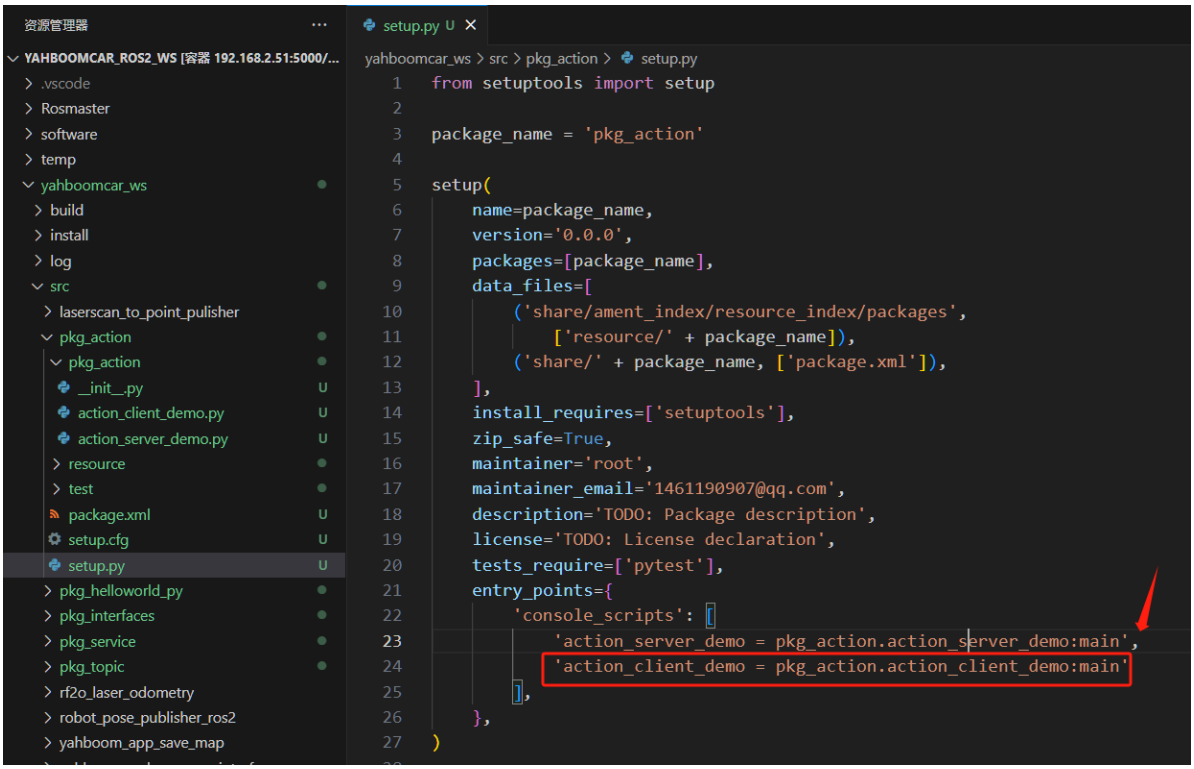


Next edit [action_client_demo.py] to implement the server-side functions and add the following code:

```
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
from pkg_interfaces.action import Progress

class Action_Client(Node):
    def __init__(self):
        super().__init__('progress_action_client')
        #Create action client;
        self._action_client = ActionClient(self, Progress, 'get_sum')

    def send_goal(self, num):
        # Send request;
        goal_msg = Progress.Goal()
        goal_msg.num = num
        self._action_client.wait_for_server()
        self._send_goal_future = self._action_client.send_goal_async(goal_msg,
feedback_callback=self.feedback_callback)
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        # Handle feedback after the target is sent;
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Request denied')
            return

        self.get_logger().info('The request is received and the task begins! ')

        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    # Handle the final response.
    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Final calculation result:sum = %d' % result.sum)
        # 5.Release resources.
        rclpy.shutdown()
```

```
    # handling continuous feedback;
    def feedback_callback(self, feedback_msg):
        feedback = (int)(feedback_msg.feedback.progress * 100)
        self.get_logger().info('Current progress: %d%%' % feedback)


def main(args=None):
    rclpy.init(args=args)
    action_client = Action_Client()
    action_client.send_goal(10)
    rclpy.spin(action_client)
```

4. Edit configuration file



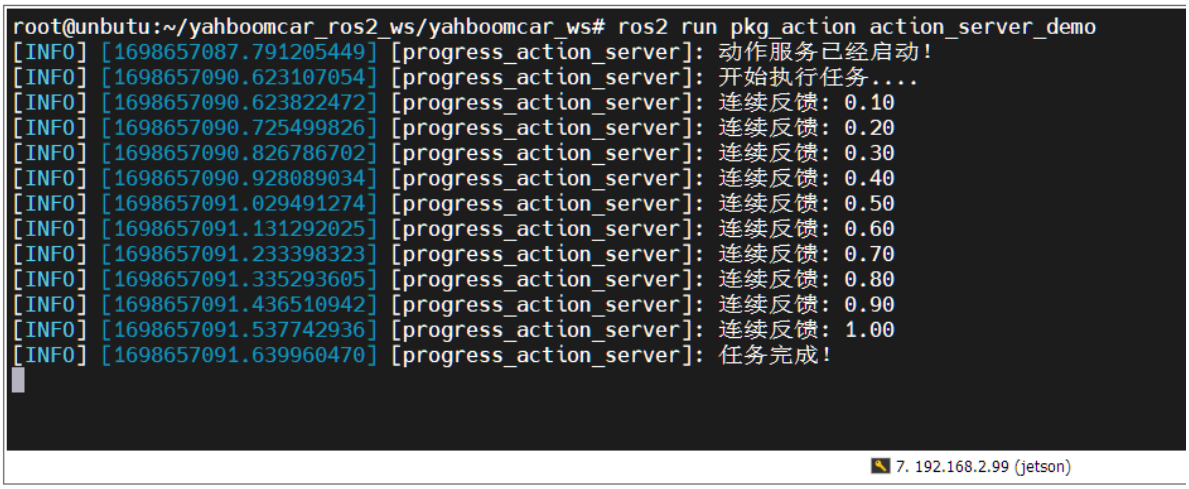
5. Compile workspace

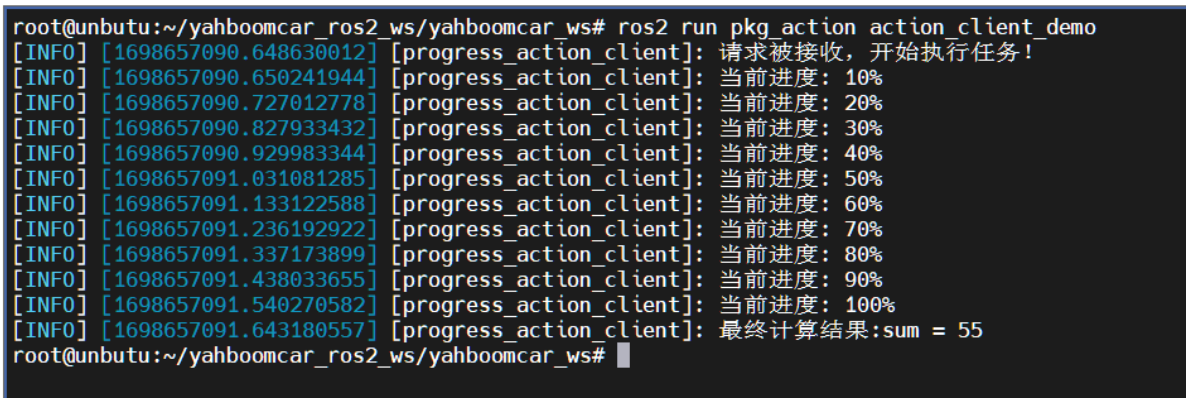
```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_action
source install/setup.bash
```

6.Run program

The sub-terminal execution is as follows:

```
#Start server node
ros2 run pkg_action action_server_demo
#Start client node
ros2 run pkg_action action_client_demo
```





The top of the picture above is the server, and the bottom is the client. Here we find the sum from 1 to 10. You can see that during the calculation of the sum from 1 to 10, the server has been feeding back the progress of the calculation. Finally, it shows that the task is completed, and the client also received feedback that the sum is 55.