

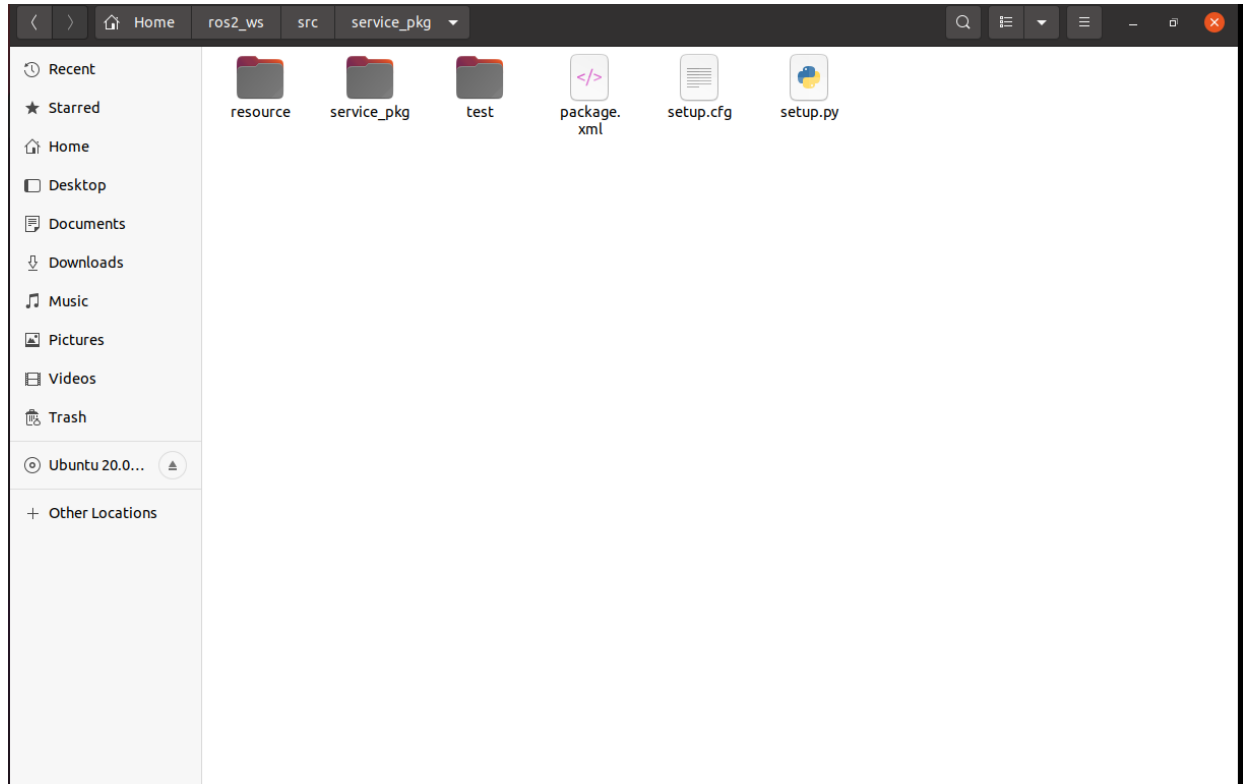
4、 ROS2 service communications

Service communication is one of the communication methods of ROS2 nodes, which is different from topic communication in that service communication has a feedback mechanism, which will feedback the results of the service. Therefore, service communication is mostly used in programs that need feedback results, such as the routine of a baby turtle, calling the Spawn service to generate a turtle, and after the service is completed (after the turtle is generated), the turtle's name will be printed. Next, it is explained how to use the Python language to achieve service communication between nodes.

1、 Create a new feature pack

src new function package in the previously created workspace directory, terminal input,

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python service_pkg
```



3、 Write server-side python files

3.1、 Program source code

Create a new file, named server_demo.py,

```
cd ~/ros2_ws/src/service_pkg/service_pkg
gedit server_demo.py
```

Copy the following sections into the file,

```

#Import the relevant library files
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts

class Service_Server(Node):
    def __init__(self,name):
        super().__init__(name)
        #To create a server, use create_service function, and the parameters passed
        in are:
        #The data type of the service data, the name of the service, and the service
        callback function (that is, the content of the service)
        self.srv = self.create_service(AddTwoInts, '/add_two_ints',
self.Add2Ints_callback)
        #The content of the service callback function here is to add two integer numbers
        and return the result of the addition
        def Add2Ints_callback(self,request,response):
            response.sum = request.a + request.b
            print("response.sum = ",response.sum)
            return response
def main():
    rclpy.init()
    server_demo = Service_Server("publisher_node")
    rclpy.spin(server_demo)

```

Focus on the service callback function, Add2Ints_callback, in addition to self, there are also request and response, request is the parameter required by the service, and response is the feedback result of the service. request.a and request.b are the content of the request part, response.sum is the content of the response part, here first look at the AddTwoInts type of data is how, you can use the following command to view,

```
ros2 interface show example_interfaces/srv/AddTwoInts
```

```

yahboom@yahboom-virtual-machine:~$ ros2 interface show example_interfaces/srv/AddTwoInts
int64 a
int64 b
---
int64 sum

```

--- The section divides this type of data into two parts, the upper side represents the request and the lower side represents the response. Then the variables in each domain, such as int64 a and int64 b, all the parameters passed in need to specify what the value of a and b is. Similarly, the result of the feedback also needs to specify what the value of sum is.

3.2、 Modify the setup.py file

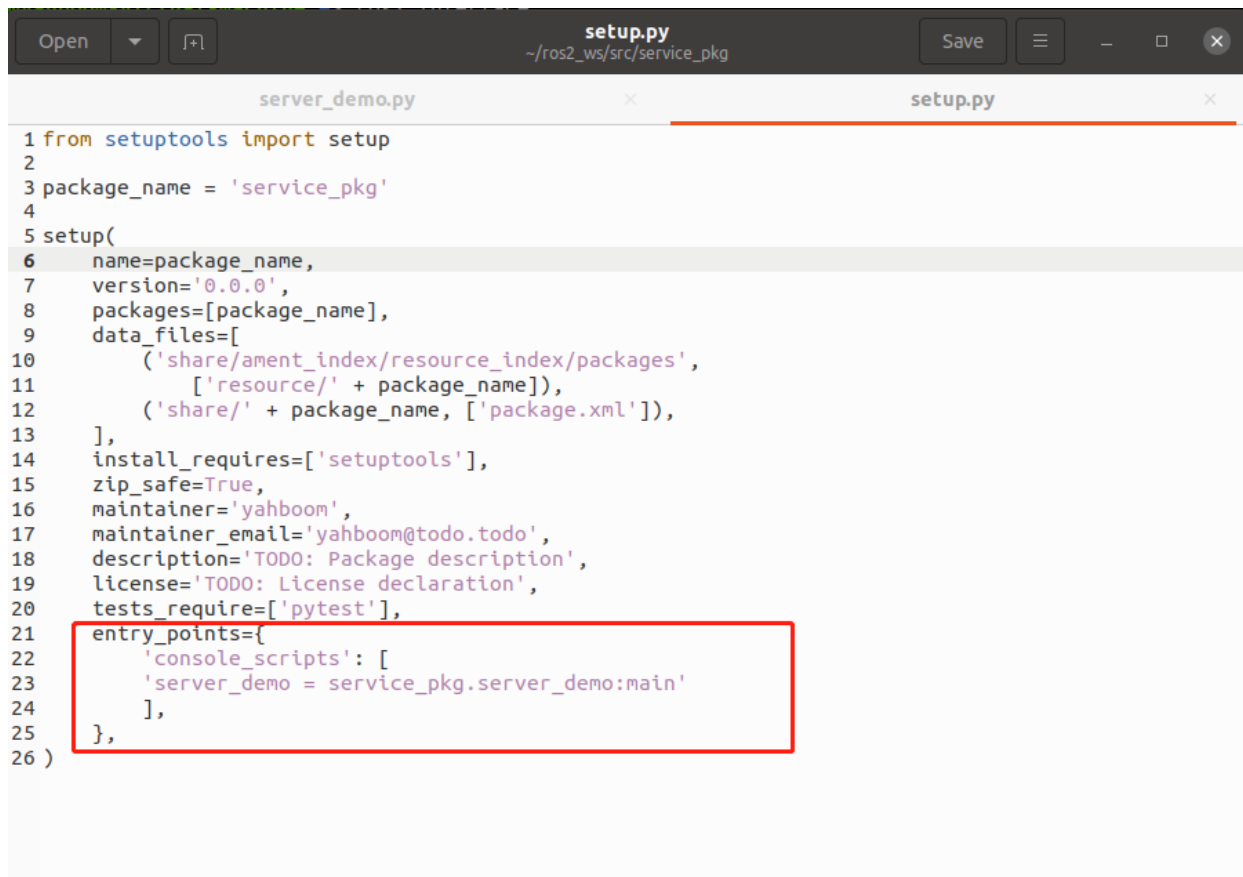
terminal input,

```

cd ~/ros2_ws/src/service_pkg
gedit setup.py

```

Locate the location as shown in the image below,



```
1 from setuptools import setup
2
3 package_name = 'service_pkg'
4
5 setup(
6     name=package_name,
7     version='0.0.0',
8     packages=[package_name],
9     data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='yahboom',
17     maintainer_email='yahboom@todo.todo',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'server_demo = service_pkg.server_demo:main'
24         ],
25     },
26 )
```

Add the following to 'console_scripts': [].

```
'server_demo = service_pkg.server_demo:main'
```

3.3、 Compile the workspace

```
cd ~/ros2_ws
colcon build
```

After the compilation is completed, refresh the environment variables of the workspace.

```
source ~/ros2_ws/install/setup.bash
```

3.4、 Run the program

terminal input,

```
ros2 run service_pkg server_demo
```

After running, because the service is not called, there is no feedback data, you can call the service through the command line, first query what services are currently available, terminal input,

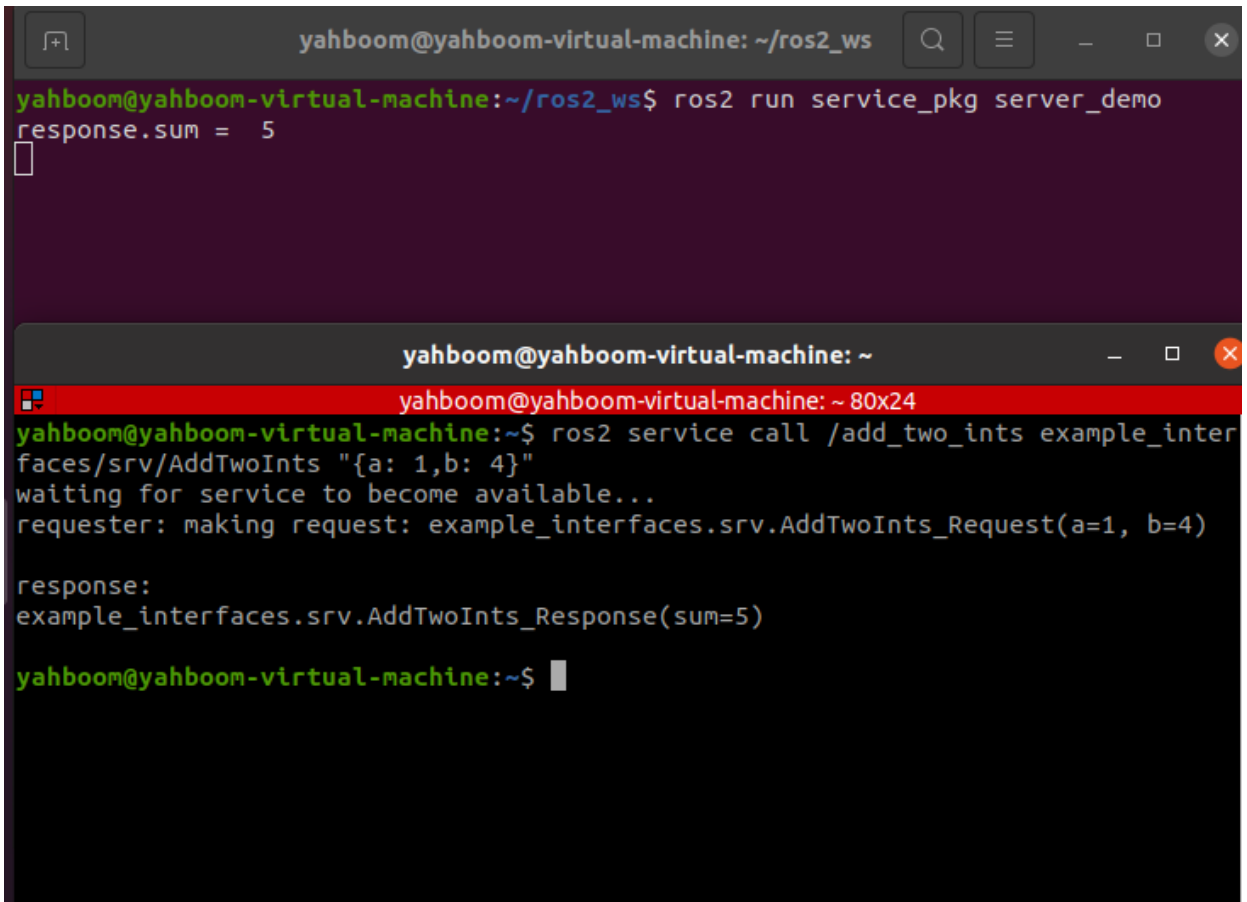
```
ros2 service list
```

```
yahboom@yahboom-virtual-machine:~$ ros2 service list
/add_two_ints
/publisher_node/describe_parameters
/publisher_node/get_parameter_types
/publisher_node/get_parameters
/publisher_node/list_parameters
/publisher_node/set_parameters
/publisher_node/set_parameters_atomically
```

/add_two_ints is the service we need to call, call it through the following command, terminal input,

```
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 1,b: 4}"
```

Here we assign the value of a to 1 and the value of b to 4, that is, call the service to calculate the sum of 1 and 4,



```
yahboom@yahboom-virtual-machine: ~/ros2_ws
yahboom@yahboom-virtual-machine:~/ros2_ws$ ros2 run service_pkg server_demo
response.sum = 5

yahboom@yahboom-virtual-machine: ~
yahboom@yahboom-virtual-machine: ~ 80x24
yahboom@yahboom-virtual-machine:~$ ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 1,b: 4}"
waiting for service to become available...
requester: making request: example_interfaces.srv.AddTwoInts_Request(a=1, b=4)

response:
example_interfaces.srv.AddTwoInts_Response(sum=5)

yahboom@yahboom-virtual-machine:~$
```

As can be seen from the above figure, after calling the service, the result of the feedback is 5, and the terminal running the server also prints the value of the feedback.

4、 Write a client python file

4.1、 Program source code

Create a new file, named client_demo.py,

```
cd ~/ros2_ws/src/service_pkg/service_pkg
gedit client_demo.py
```

Copy the following into it,

```
#Import the relevant libraries
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts

class Service_Client(Node):
    def __init__(self,name):
        super().__init__(name)
        #To create a client, you use create_client function, and the parameters
        #passed in are the data type of the service data and the topic name of the service
        self.client = self.create_client(AddTwoInts,'/add_two_ints')
        #Cycle through and wait for the server side to start successfully
        while not self.client.wait_for_service(timeout_sec=1.0):
            print("service not available, waiting again...")
        #Create a data object for the service request
        self.request = AddTwoInts.Request()

    def send_request(self):
        self.request.a = 10
        self.request.b = 90
        #Send a service request
        self.future = self.client.call_async(self.request)

def main():
    rclpy.init() #Node initialization
    service_client = Service_Client("client_node") #Create an object
    service_client.send_request() #Send a service request
    while rclpy.ok():
        rclpy.spin_once(service_client)
        #Determine whether the data processing is complete
        if service_client.future.done():
            try:
                #Get service feedback information and print it
                response = service_client.future.result()
                print("Result = ",response.sum)
            except Exception as e:
                service_client.get_logger().info('service call failed %r' % (e,))

        break
```

4.2. Modify the setup.py file

terminal input,

```
cd ~/ros2_ws/src/topic_pkg
gedit setup.py
```

Locate the location as shown in the image below,



```
1 from setuptools import setup
2
3 package_name = 'service_pkg'
4
5 setup(
6     name=package_name,
7     version='0.0.0',
8     packages=[package_name],
9     data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='yahboom',
17     maintainer_email='yahboom@todo.todo',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'server_demo = service_pkg.server_demo:main',
24             'client_demo = service_pkg.client_demo:main'
25         ],
26     },
27 )
```

Add the following to 'console_scripts': [],

```
'client_demo = service_pkg.client_demo:main'
```

```
'client_demo = service_pkg.client_demo:main'
```

4.3. Compile the workspace

```
cd ~/ros2_ws
colcon build
```

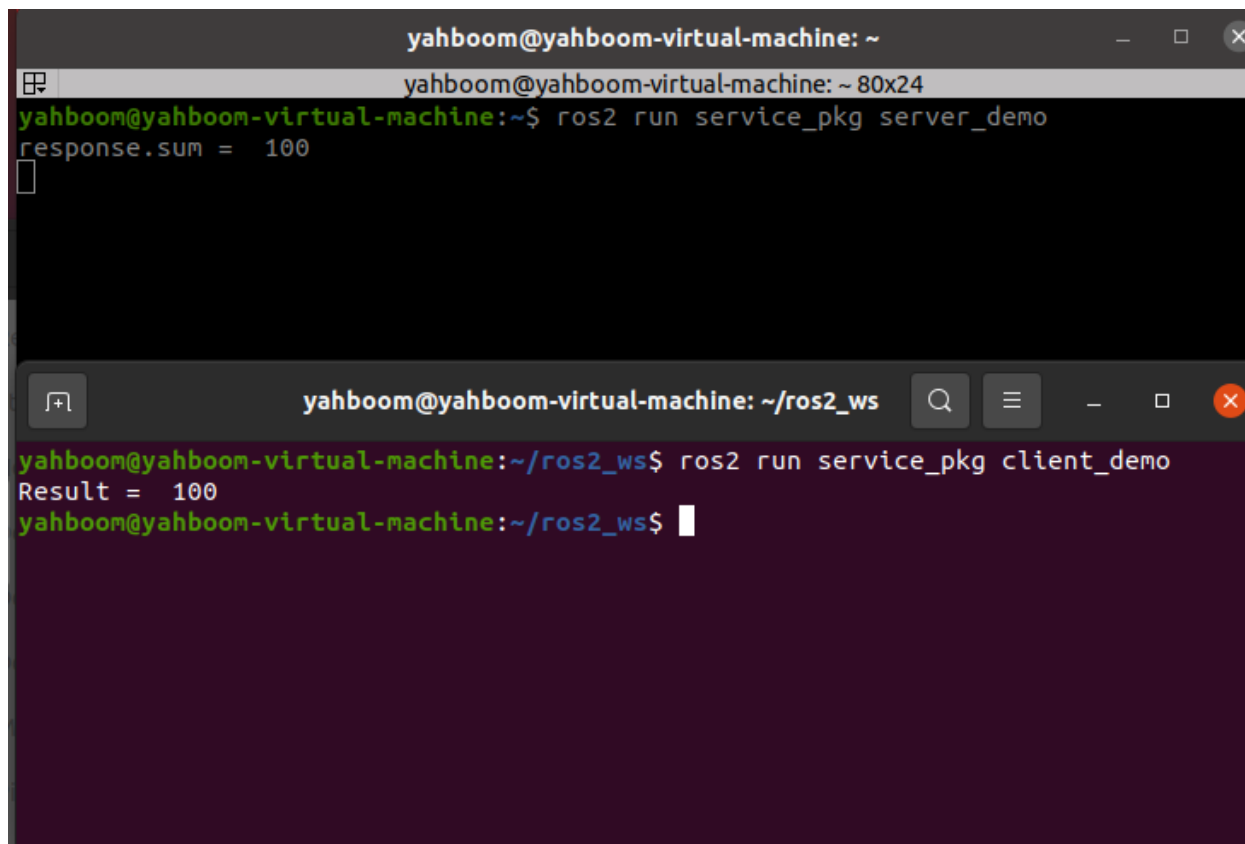
After the compilation is completed, refresh the environment variables of the workspace.

```
source ~/ros2_ws/install/setup.bash
```

4.4. Run the program

terminal input,

```
#Start the server
ros2 run service_pkg server_demo
#Start the client
ros2 run service_pkg client_demo
```



The image shows two terminal windows from a virtual machine. The top window, titled 'yahboom@yahboom-virtual-machine: ~', has a dark background and shows the command 'ros2 run service_pkg server_demo' being executed. The output is 'response.sum = 100'. The bottom window, titled 'yahboom@yahboom-virtual-machine: ~/ros2_ws', has a dark purple background and shows the command 'ros2 run service_pkg client_demo' being executed. The output is 'Result = 100'.

```
yahboom@yahboom-virtual-machine: ~  
yahboom@yahboom-virtual-machine: ~ 80x24  
yahboom@yahboom-virtual-machine:~$ ros2 run service_pkg server_demo  
response.sum = 100  
  
yahboom@yahboom-virtual-machine: ~/ros2_ws  
yahboom@yahboom-virtual-machine:~/ros2_ws$ ros2 run service_pkg client_demo  
Result = 100  
yahboom@yahboom-virtual-machine:~/ros2_ws$
```

First run the server, then run the client, the client provides a=10, b=90, the server summes, the result is 100, and the result is printed at the two terminals.