

9. Custom Service Messages and Usage

9.1 Customized Service Messages

Switch to `~/catkin_ws/src/learning_` Under the server function package directory, create a new folder named `srv` to store custom service messages.

9.1.1 Define SRV files

Switch to the `srv` directory and create a new blank `srv` file, with `srv` as the suffix to indicate that it is an `srv` file. Here we use `IntPlus.srv` as an example to copy the following code into the just created `srv` file.

```
uint8  a
uint8  b

---
uint8  result
```

Here is an explanation of the composition of the SRV file, which is divided into two parts by the symbol `---`. The upper side represents the request and the lower side is the response.

9.1.2 Add feature pack dependencies in `package.xml`

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

9.1.3. Add compilation options in `CMakeLists.txt`

```
add_service_files(FILES IntPlus.srv)
generate_messages(DEPENDENCIES std_msgs)
```

9.1.4 Compile and generate language related files

```
cd ~/catkin_ws
catkin_make
```

9.1.5 C++ Language Implementation

1) Switch to `~/catkin_ws/src/learning_` Under `server/src`, create two new `cpp` files named `IntPlus_Server.cpp` and `IntPlus_Client.cpp`, copy the following code into it separately,

`IntPlus_server.cpp`

```
#include <ros/ros.h>
#include "learning_server/IntPlus.h"

// Service callback function, input parameter req, output parameter res
bool IntPlusCallback(learning_server::IntPlus::Request &req,
```

```

        learning_server::IntPlus::Response &res)
{
    ROS_INFO("number 1 is:%d ,number 2 is:%d ", req.a, req.b); //Display request
data

    res.result = req.a + req.b ; // The feedback result is the sum of two numbers

    return res.result;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "IntPlus_server"); // ROS node initialization

    ros::NodeHandle n; // Create node handle

    // Create a server and register the callback function IntPlusCallback
    ros::ServiceServer Int_Plus_service = n.advertiseService("/Two_Int_Plus",
IntPlusCallback);

    // Loop waiting callback function
    ROS_INFO("Ready to caculate.");
    ros::spin();

    return 0;
}

```

IntPlus_client.cpp

```

#include <ros/ros.h>
#include "learning_server/IntPlus.h"
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
    int i,k;
    cin>>i;
    cin>>k;

    ros::init(argc, argv, "IntPlus_client"); // ROS node initialization

    ros::NodeHandle node; // Create node handle

    // Create a service client after discovering the/Two_int-Plus service
    ros::service::waitForService("/Two_Int_Plus");
    ros::ServiceClient IntPlus_client =
node.serviceClient<learning_server::IntPlus>("/Two_Int_Plus");

    // Initialize learningservice:: IntPlus request data
    learning_server::IntPlus srv;
    srv.request.a = i;
    srv.request.b = k;

```

```

    ROS_INFO("Call service to plus %d and %d", srv.request.a, srv.request.b); //
Request service invocation

    IntPlus_client.call(srv);

    // Display service call results
    ROS_INFO("Show the result : %d", srv.response.result); // Display service call
results

    return 0;
}

```

2. Modify the CMakeLists.txt file

```

add_executable(IntPlus_server src/IntPlus_server.cpp)
target_link_libraries(IntPlus_server ${catkin_LIBRARIES})
add_dependencies(IntPlus_server ${PROJECT_NAME}_generate_messages_cpp)

add_executable(IntPlus_client src/IntPlus_client.cpp)
target_link_libraries(IntPlus_client ${catkin_LIBRARIES})
add_dependencies(IntPlus_client ${PROJECT_NAME}_generate_messages_cpp)

```

3. Core part

The implementation process here is the same as before, with the main difference being the introduction of header files and the use of custom service files: The import header file is

```
#include "learning_server/IntPlus.h"
```

Front learning_Server is the name of the feature pack, followed by IntPlus.h, which is the header file name generated by the previously created srv file Using custom service files is

```

client:
learning_server::IntPlus srv;
srv.request.a = i;
srv.request.b = k;
#i. k is the addend input by the terminal
ros::ServiceClient IntPlus_client = node.serviceClient<learning_server::IntPlus>
("/Two_Int_Plus");
IntPlus_client.call(srv);
server:
ros::ServiceServer Int_Plus_service = n.advertiseService("/Two_Int_Plus",
IntPlusCallback);
bool IntPlusCallback(learning_server::IntPlus::Request &req,
learning_server::IntPlus::Response &res)

```

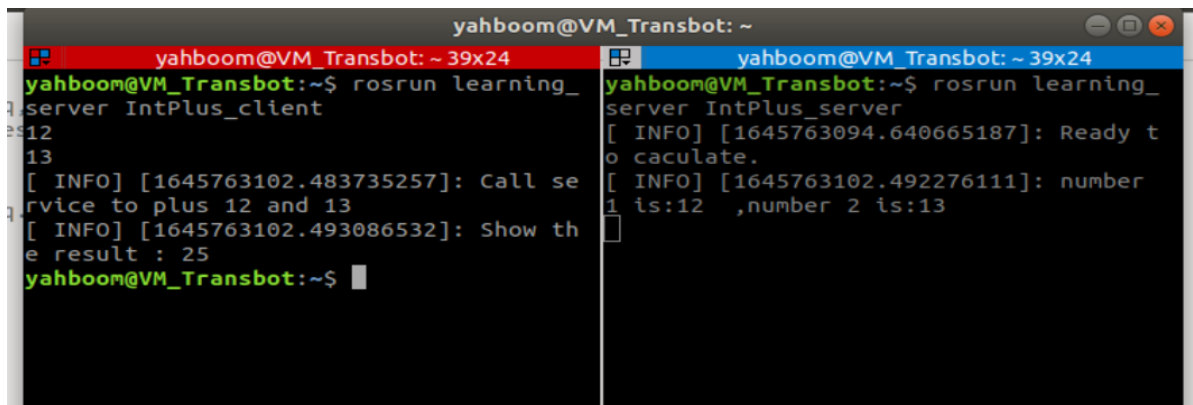
4. run a program

```

roscore
roslaunch learning_server IntPlus_client
roslaunch learning_server IntPlus_server

```

5. Run screenshot



```
yahboom@VM_Transbot: ~ 39x24
yahboom@VM_Transbot:~$ roslaunch learning_
server IntPlus_client
12
13
[ INFO] [1645763102.483735257]: Call se
rvice to plus 12 and 13
[ INFO] [1645763102.493086532]: Show th
e result : 25
yahboom@VM_Transbot:~$
```

```
yahboom@VM_Transbot: ~ 39x24
yahboom@VM_Transbot:~$ roslaunch learning_
server IntPlus_server
[ INFO] [1645763094.640665187]: Ready t
o caculate.
[ INFO] [1645763102.492276111]: number
1 is:12 ,number 2 is:13
```

6. Program Description

Running IntPlus_ After the server, it will prompt to prepare for calculation; Running IntPlus_ After the client, the terminal inputs two integer numbers, followed by IntPlus_ The server accountant calculates the result and returns it to IntPlus_ Client, and then print out the results.

9.1.6 Python Language Implementation

1) Switch to~/catkin_ws/src/learning_ Under server/script, create two new py files and name them IntPlus_Server.py and IntPlus_Client.py, copy the following code into it separately,

IntPlus_server.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy

from learning_server.srv import IntPlus, IntPlusResponse

def IntPlusCallback(req):

    rospy.loginfo("Ints: a:%d b:%d", req.a, req.b)# Display request data

    return IntPlusResponse(req.a+req.b)# Feedback data

def IntPlus_server():

    rospy.init_node('IntPlus_server')# ROS node initialization

    # Create a server and register the callback function IntPlusCallback
    s = rospy.Service('/Two_Int_Plus', IntPlus, IntPlusCallback)

    print "Ready to caculate two ints."# Loop waiting callback function

    rospy.spin()

if __name__ == "__main__":
    IntPlus_server()
```

IntPlus_client.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```

import sys
import rospy
from learning_server.srv import IntPlus, IntPlusRequest

def Plus_client():
    # ROS node initialization
    rospy.init_node('IntPlus_client')

    rospy.wait_for_service('/Two_Int_Plus')
    try:
        plus_client = rospy.ServiceProxy('/Two_Int_Plus', IntPlus)

        response = plus_client(22, 20) # Request service call, input request data

        return response.result
    except rospy.ServiceException, e:
        print "failed to call service : %s"%e

if __name__ == "__main__":
    #Call the service and display the call result
    print "Show two_int_plus result : %s" %(plus_client())

```

2. Core part

Here is mainly an explanation of how to import a custom service message module and use it: Import

```

server:
from learning_server.srv import IntPlus, IntPlusResponse
client:
from learning_server.srv import IntPlus, IntPlusRequest

```

use

```

server:
s = rospy.Service('/Two_Int_Plus', IntPlus, IntPlusCallback)
return IntPlusResponse(req.a+req.b) # Feedback data
client:
response = plus_client(12, 20) # Request service call, input request data
return response.result

```

3) Run program

Before running the program, add executable permissions to the py file

```

sudo chmod a+x IntPlus_server.py
sudo chmod a+x IntPlus_client.py

```

run a program

```

roscore
roslaunch learning_server IntPlus_client.py
roslaunch learning_server IntPlus_server.py

```

4. Program operation instructions

What is inconsistent with the C++ version here is that the addend is set in the program (12 and 20), so once the service is started, the result can be returned immediately.