8 Server

In the last lesson, we talked about the client requesting the service, and then the server providing the service. In this lesson, we will talk about how the server implements the service.

8.1 C++ language implementation

8.1.1 Implementation steps

1. initialize the ROS node
2. create a Server instance
3. wait for the service request in a loop, enter the callback function
4. complete the functional processing of the service in the callback function, and feedback the response data

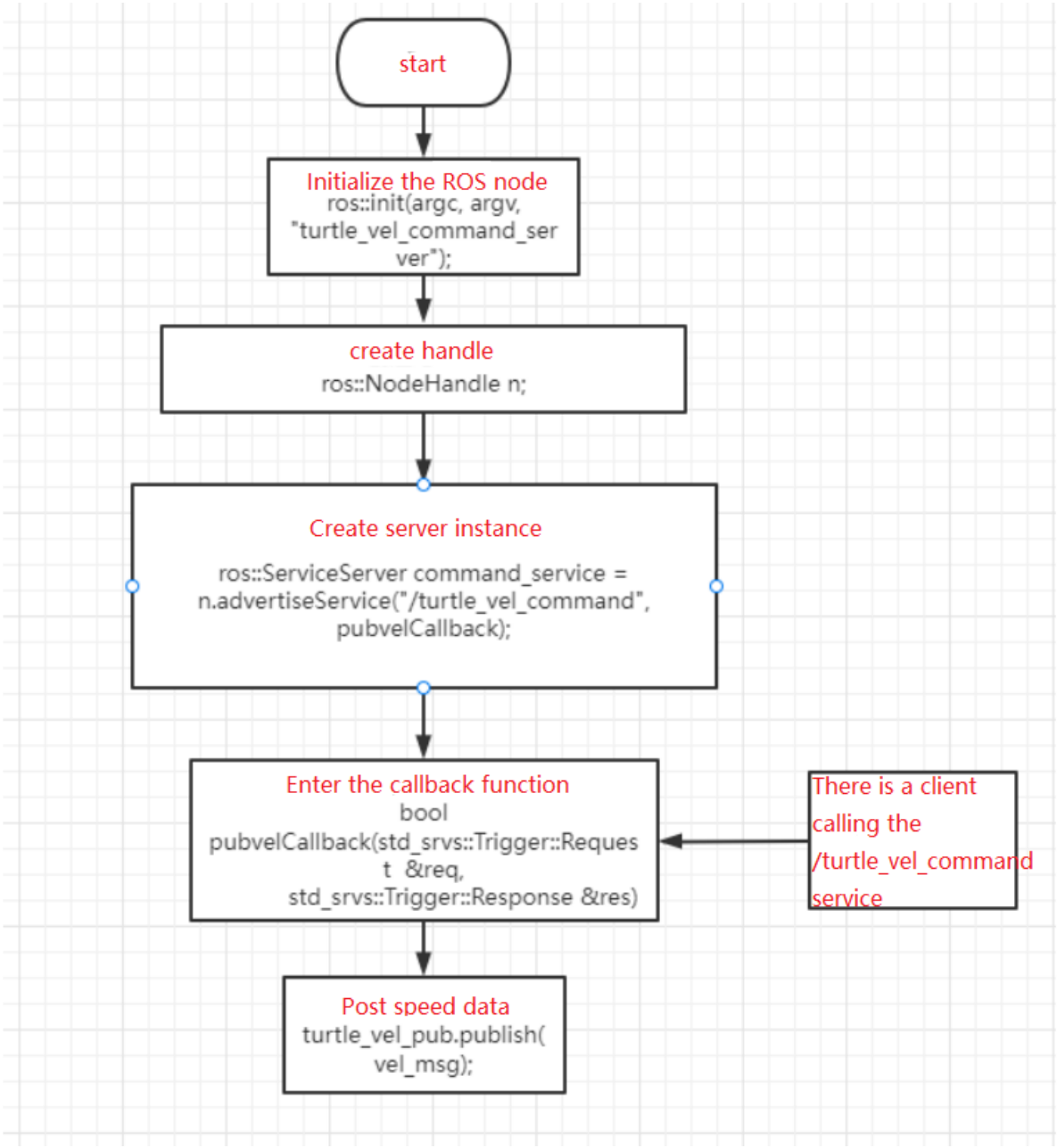
8.1.2 Switch to the ~/catkin_ws/src/learning_server/src directory, create a new .cpp file, name it turtle_vel_command_server, and paste the code below into it

turtle_vel_command_server.cpp

```
/**
 * This routine will execute the /turtle_vel_command service, service data type
std_srvs/Trigger
 */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <std_srvs/Trigger.h>

ros::Publisher  turtle_vel_pub;
bool  pubvel  =  false;

// service callback function, input parameter req, output parameter res
bool  pubvelCallback(std_srvs::Trigger::Request   & req,
                     std_srvs::Trigger::Response  & res)
{
    pubvel  =  !pubvel;

        ROS_INFO("Do you want to publish the vel?: [%s]", pubvel == true ? "Yes"
: "No"); // print client request data
```

```
    // set feedback data
    res.success  =  true;
    res.message  =  "The status is changed!";

    return  true;
}

int  main(int  argc, char  ** argv)
{

    ros::init(argc, argv, "turtle_vel_command_server");


    ros::NodeHandle  n;

    // Create a server named /turtle_vel_command and register the callback
function pubvelCallback
    ros::ServiceServer  command_service  =
n.advertiseService("/turtle_vel_command", pubvelCallback);

    // Create a Publisher, publish a topic named /turtle1/cmd_vel, the message
type is geometry_msgs::Twist, and the queue length is 8
    turtle_vel_pub  =  n.advertise < geometry_msgs::Twist >("/turtle1/cmd_vel",
8);

    ros::Rate  loop_rate(10); // set the frequency of the loop

    while(ros::ok())
    {

        ros::spinOnce(); // View the callback function queue once

        // Judging that pubvel is True, issue the small turtle speed command
        if(pubvel)
        {
            geometry_msgs::Twist vel_msg;
            vel_msg.linear.x  =  0.6;
            vel_msg.angular.z  =  0.8;
            turtle_vel_pub.publish(vel_msg);
        }

        loop_rate.sleep(); //Delay according to the loop frequency
    }

    return  0;
}
```

1. program flow chart



2. configure in CMakelist.txt, under the build area, add the following content

```
add_executable(turtle_vel_command_server src/turtle_vel_command_server.cpp)
target_link_libraries(turtle_vel_command_server ${catkin_LIBRARIES})
```

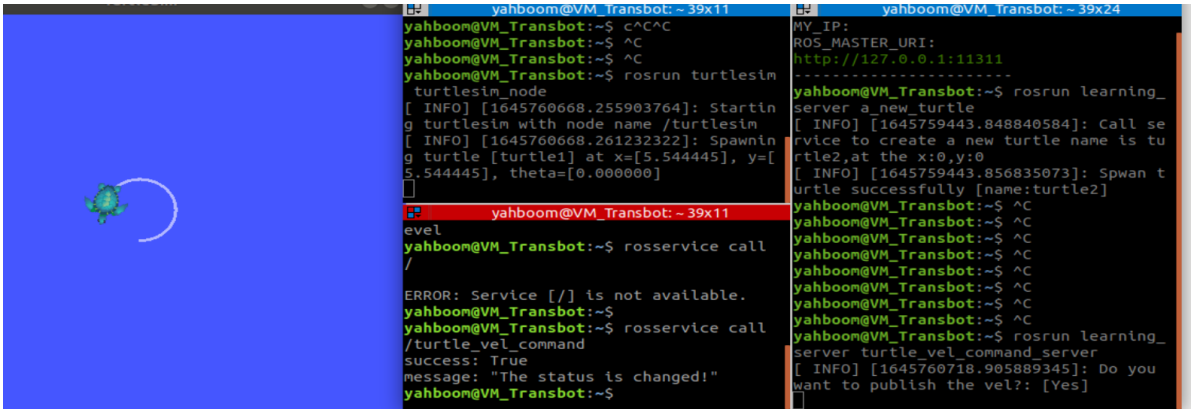
3. Compile the code in the workspace directory

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash #Environment      variables need to be configured,
otherwise the system cannot find the running program
```

4. run the program

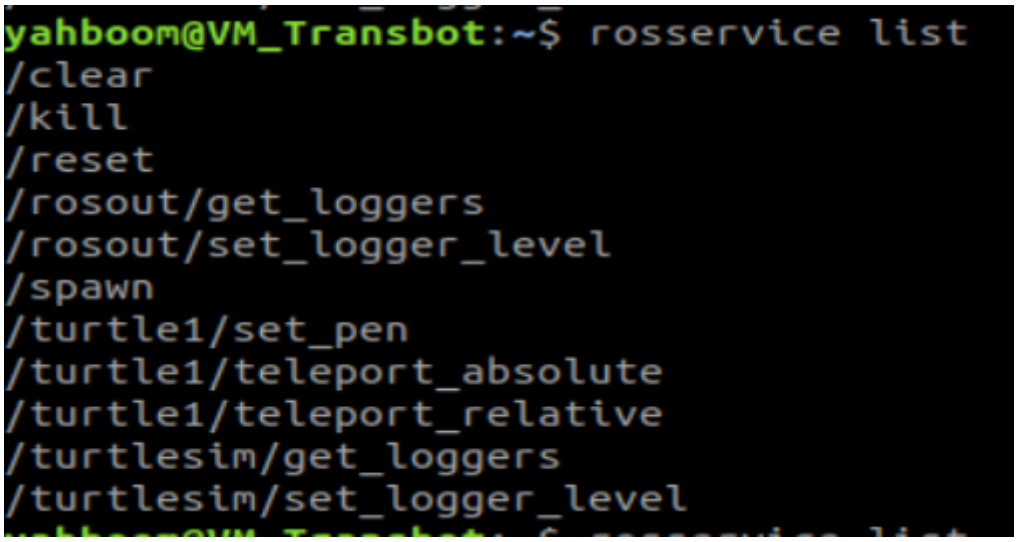
```
roscore
rosrun turtlesim turtlesim_node
rosrun learning_server turtle_vel_command_server
```

5. Screenshot of running effect

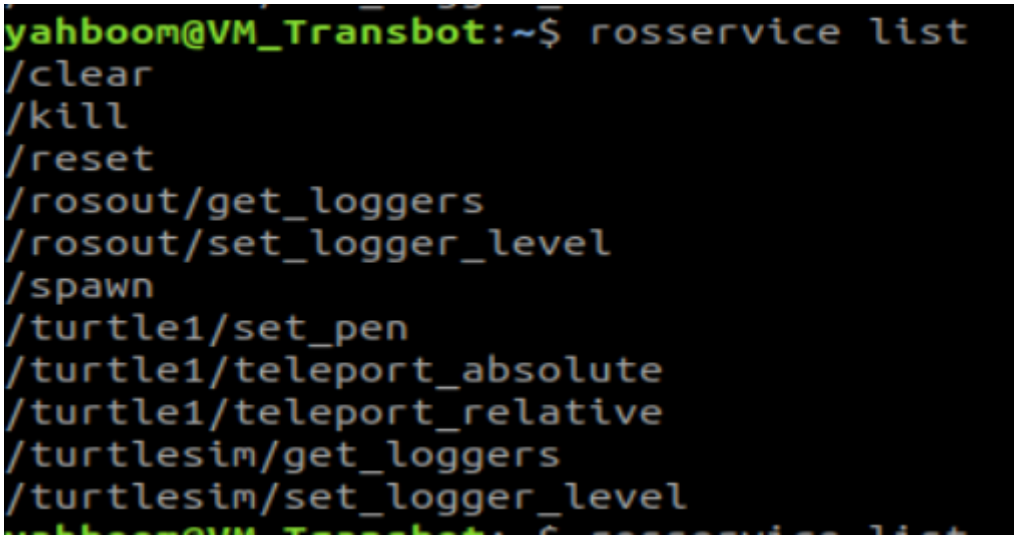


6. program description

First, after running the little turtle node, you can enter rosservice list in the terminal to see what the current services are. The results are as follows



Then, we run the turtle_vel_command_server program, and then enter the rosservice list, we will find an additional turtle_vel_command_server, as shown in the following figure



Then, we call this service by entering rosservice call /turtle_vel_command_server in the terminal, and we will find that the small turtle moves in a circle. If the service is called again, the small turtle stops moving. This is because in the service callback function, we invert the value of pubvel, and then feed it back. The main function will judge the value of pubvel. If it is True, it will issue a speed command, and if it is False, no command will be issued.

8.2 Python language implementation

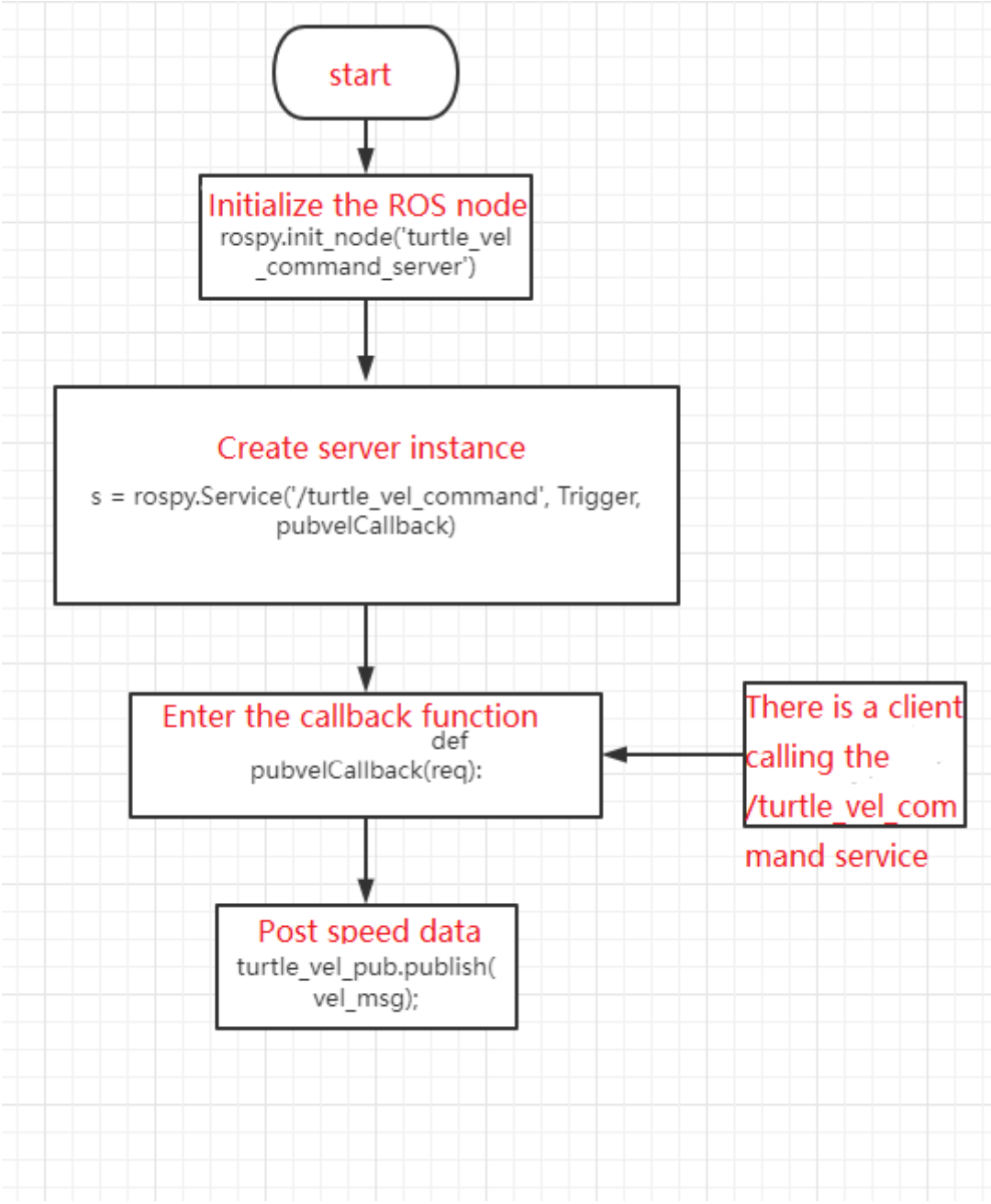
8.2.1 Switch to the ~/catkin_ws/src/learning_server directory, create a new script folder, cut it in, create a new py file, name it turtle_vel_command_server, and paste the code below into it

turtle_vel_command_server.py

```
# !/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will execute the /turtle_command service, service data type
std_srvs/Trigger

import  rospy
import  thread, time
from  geometry_msgs.msg  import  Twist
from  std_srvs.srv  import  Trigger, TriggerResponse

pubvel  =  False;
turtle_vel_pub  =  rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size = 8)

 def pubvel_thread():
    while  True :
        if  pubvel :
            vel_msg  =  Twist()
            vel_msg.linear.x  =  0.6
            vel_msg.angular.z  =  0.8
            turtle_vel_pub.publish(vel_msg)

        time.sleep(0.1)

 def pubvelCallback(req):
    global  pubvel

    pubvel  =  bool(1 - pubvel)

    rospy.loginfo("Do you want to publish the vel?[%s]", pubvel) # Display
request data

    return  TriggerResponse(1, "Change state!") # Feedback data

 def turtle_pubvel_command_server():

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

    # Create a server named /turtle_command and register the callback function
pubvelCallback
    s  =  rospy.Service('/turtle_vel_command', Trigger, pubvelCallback)

    # loop waiting for the callback function
    print  "Ready to receive turtle_pub_vel_command."

    thread.start_new_thread(pubvel_thread,())
    rospy.spin()
```

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

1. program flow chart



2. run the program

```
roscore
rosrun turtlesim turtlesim_node
rosrun learning_server turtle_vel_command_server.py
```

3. The program operation effect and program description are consistent with the effect realized by C++.