

8. Server side

In the previous lesson, we talked about how the client requests services and then the server provides services. In this lesson, we will talk about how the server implements providing services.

8.1. C++ Language Implementation

8.1.1 Implementation steps

1. Initialize ROS node
2. Create Server Instance
3. Loop waiting for service request, entering callback function
4. Complete the functional processing of the service in the callback function and provide feedback on the response data

8.1.2. Switch to `~/catkin_ws/src/learning_` Create a new. `cpp` file under the `server/src` directory and name it `turtle_vel_command_Server`, paste the following code inside

`turtle_vel_command_server.cpp`

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

ros::Publisher turtle_vel_pub;
bool pubvel = false;

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");

    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;
```

```

    ros::ServiceServer command_service =
n.advertiseService("/turtle_vel_command", pubvelCallback);

    turtle_vel_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 8);

    ros::Rate loop_rate(10);

    while(ros::ok())
    {

        ros::spinOnce();// view a callback function queue

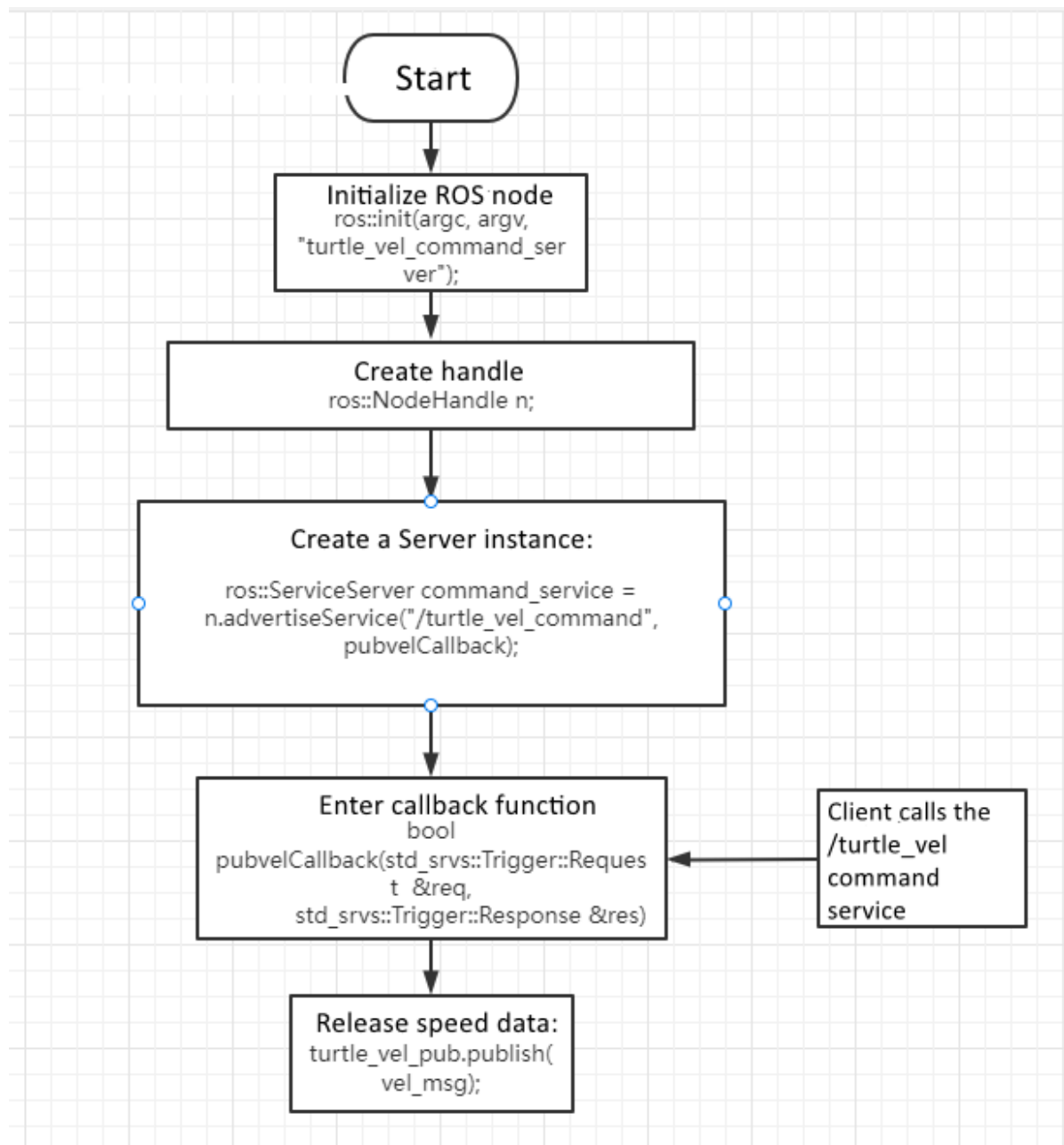
        //If pubvel is True, the turtle speed command is issued.
        if(pubvel)
        {
            geometry_msgs::Twist vel_msg;
            vel_msg.linear.x = 0.6;
            vel_msg.angular.z = 0.8;
            turtle_vel_pub.publish(vel_msg);
        }

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

    return 0;
}

```

1. Process 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. Compiling code under workspace directory

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

4. run a program

```
roscore
roslaunch turtlesim turtlesim_node
roslaunch learning_server turtle_vel_command_server
```

5. Running effect screenshot

Then, we input rosservice call/tour at the terminal_ vel_ command_ When the server calls this service, it will find that the little turtle is doing circular motion. If the service is called again, the little turtle stops moving. This is because in the service callback function, we invert the value of pubvel and provide feedback. The main function will determine the value of pubvel. If it is true, we will issue speed instructions, and if it is false, we will not issue instructions.

8.2. Python Language Implementation

8.2.1. Switch to~/catkin_ws/src/learning_ Under the server directory, create a new script folder, cut it in, and create a new py file named turtle_vel_command_Server, paste the following code inside

turtle_vel_command_server.py

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

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

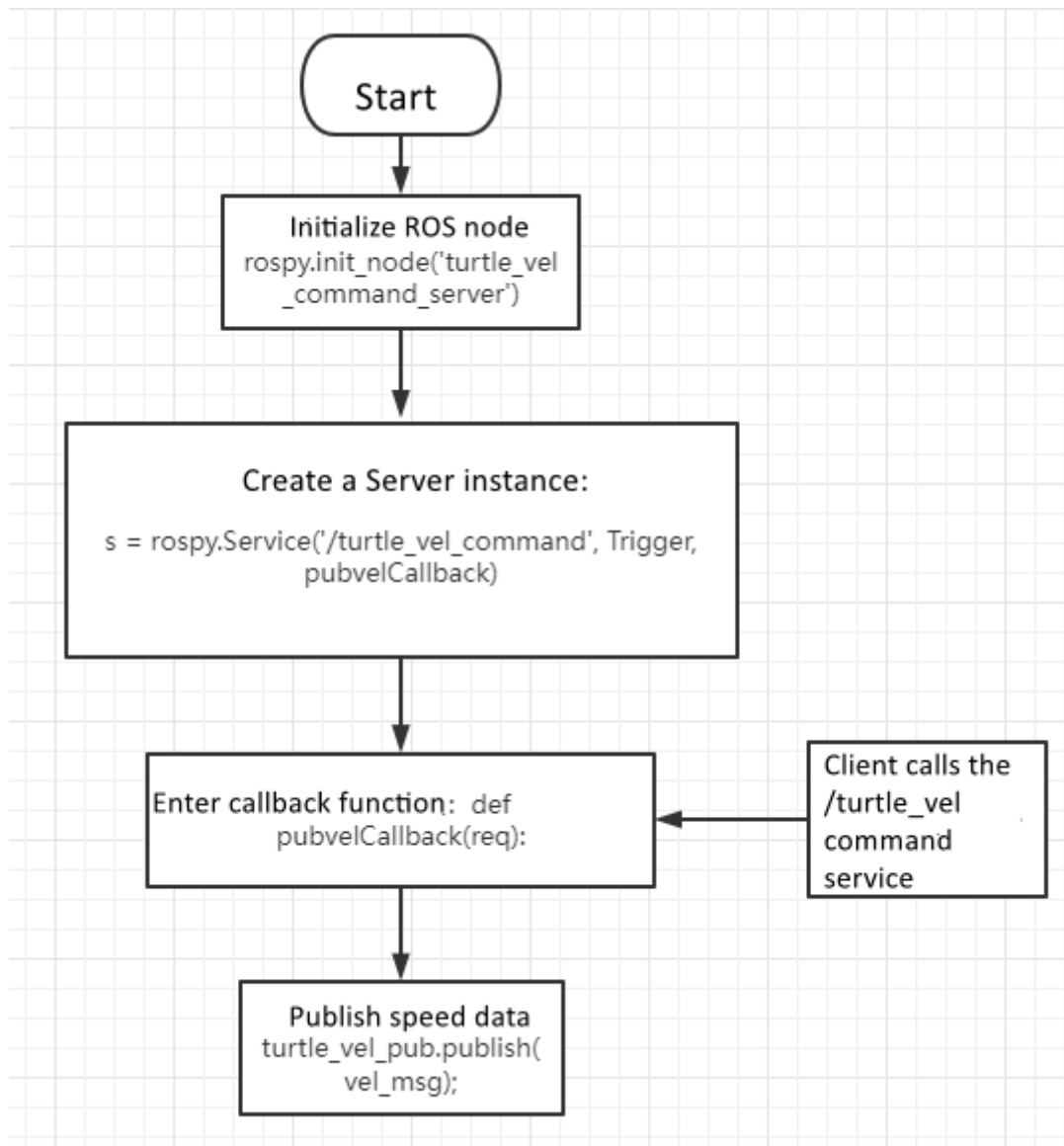
    s = rospy.Service('/turtle_vel_command', Trigger, pubvelCallback)
```

```
# Loop waiting for callback function
print "Ready to receive turtle_vel_command."

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

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

1. Process Flow Chart



2. run a program

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

3. The program operation effect and program description are consistent with the implementation effect in C++.

