9.ROS service client

Service service is also a common communication method between ros nodes. Different from the topic, the service has a server and a client. The client requests the server to provide services. After the server completes the service, it needs to return the service results. Also called a response. In this lesson, we will explain how to write a client node program.

9.1 Create a client

The general creation steps are as follows:

- Initialize ROS nodes
- Create handle
- Create a Client instance
- Initialize and publish service request data
- Wait for the response result after Server processing

9.2 Create function package

In order to distinguish it from the learn_topic function package, we re-create a function package learn_service and enter it in the terminal.

```
cd ~/ros_ws/src
catkin_create_pkg learn_service std_msgs rospy roscpp geometry_msgs turtlesim
```

Then compile,

```
cd ~/ros_ws
catkin_make
```

9.3 C++ version

9.3.1 Writing source code

In the src folder of the function package learn_service, create a C++ file (the file suffix is .cpp), name it a_new_turtle.cpp, and paste the following content into a_new_turtle.cpp,

```
/**
* This routine will request the /spawn service in the little turtle node, and a
new little turtle will appear at the specified location.
*/
#include <ros/ros.h>
#include <turtlesim/Spawn.h>
int main(int argc, char** argv)
{
    ros::init(argc, argv, "a_nes_turtle");// Initialize ROS node
    ros::NodeHandle node;
    ros::service::waitForService("/spawn"); // Wait/spawn service
    ros::ServiceClient new_turtle = node.serviceClient<turtlesim::Spawn>
("/spawn");//Create a service client and connect to the service named /spawn
    // Initialize turtlesim::Spawn's request data
    turtlesim::Spawn new_turtle_srv;
    new_turtle_srv.request.x = 6.0;
    new_turtle_srv.request.y = 8.0;
    new_turtle_srv.request.name = "turtle2";
    // Request the service to pass in the xy position parameters and name 
parameters
    ROS_INFO("Call service to create a new turtle name is %s,at the 
x:%.1f,y:%.1f", new_turtle_srv.request.name.c_str(),
    new_turtle_srv.request.x,new_turtle_srv.request.y);
    new_turtle.call(new_turtle_srv); //Request service
    ROS_INFO("Spwan turtle successfully [name:%s]", 
    new_turtle_srv.response.name.c_str());// Display service call results
    return 0;
};
```

9.3.2 Modify CMakeList.txt file

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

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

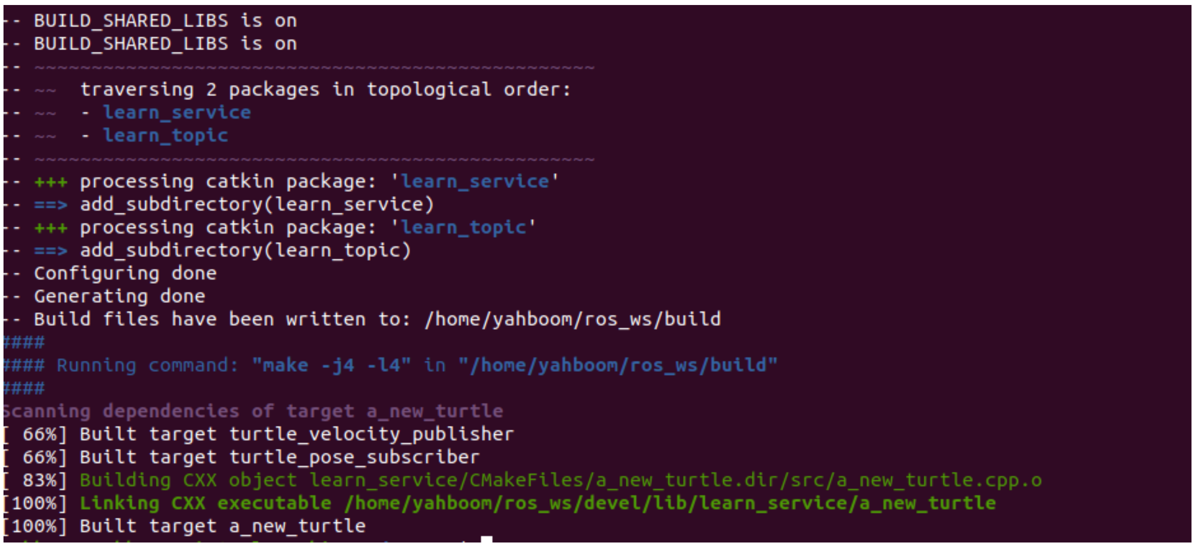
add_executable shows that the generated executable program file is a_new_turtle, and the compiled source code is a_new_turtle.cpp in the src directory.

target_link_libraries specifies the libraries that need to be linked when compiling and generating an executable file.

9.3.3 Compile

Terminal input,

```
cd ~/ros_ws
catkin_make
```



```
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 2 packages in topological order:
-- ~~  - learn_service
-- ~~  - learn_topic
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'learn_service'
-- ==> add_subdirectory(learn_service)
-- +++ processing catkin package: 'learn_topic'
-- ==> add_subdirectory(learn_topic)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/yahboom/ros_ws/build
####
#### Running command: "make -j4 -l4" in "/home/yahboom/ros_ws/build"
####
Scanning dependencies of target a_new_turtle
[ 66%] Built target turtle_velocity_publisher
[ 66%] Built target turtle_pose_subscriber
[ 83%] Building CXX object learn_service/CMakeFiles/a_new_turtle.dir/src/a_new_turtle.cpp.o
[100%] Linking CXX executable /home/yahboom/ros_ws/devel/lib/learn_service/a_new_turtle
[100%] Built target a_new_turtle
```

After the compilation is passed, you need to re-source the current environment variables to find or update the program. Enter in the terminal.

```
cd ~/ros_ws
source devel/setup.bash
```

9.3.4 Running the program

Open roscore,

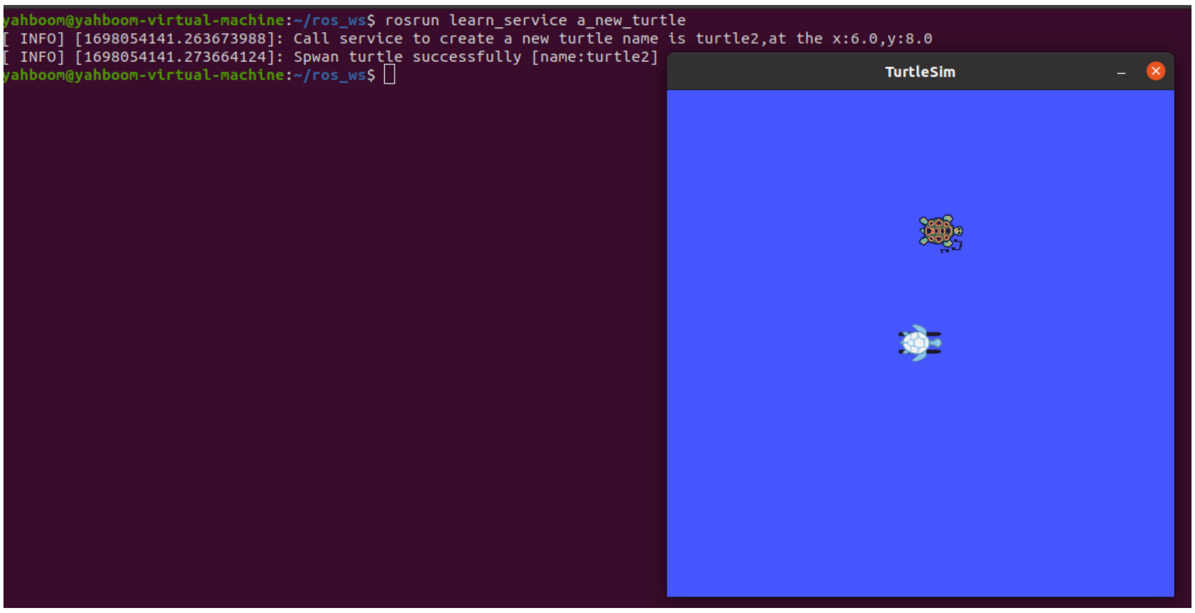
```
roscore
```

Run the little turtle node program,

```
rosrun turtlesim turtlesim_node
```

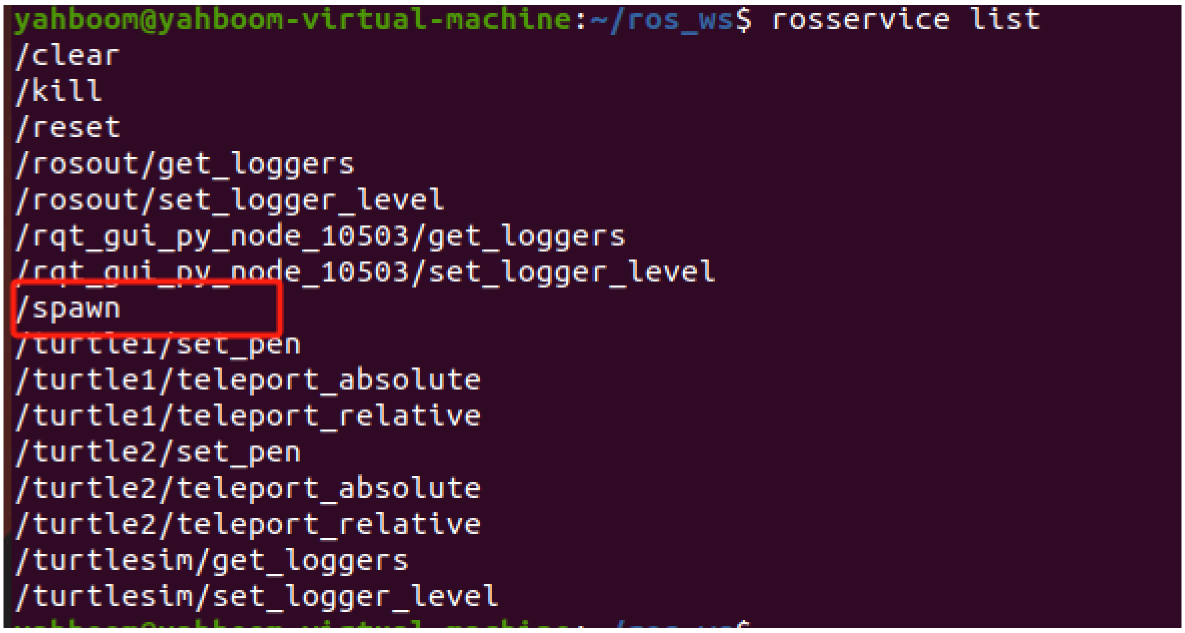
Run the client node program to generate a small turtle at the specified location.

```
rosrun learn_service a_new_turtle
```



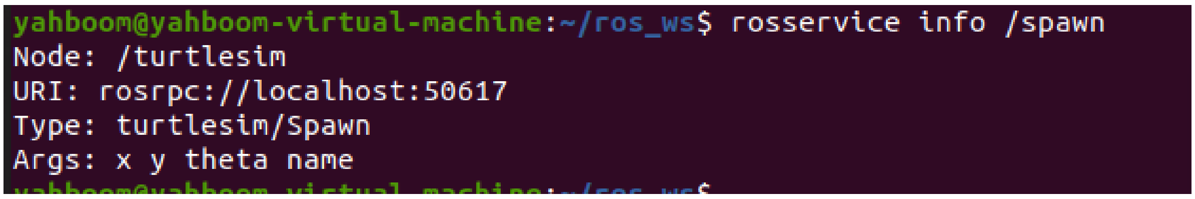
```
yahboom@yahboom-virtual-machine:~/ros_ws$ rosrun learn_service a_new_turtle
[ INFO] [1698054141.263673988]: Call service to create a new turtle name is turtle2,at the x:6.0,y:8.0
[ INFO] [1698054141.273664124]: Spwan turtle successfully [name:turtle2]
yahboom@yahboom-virtual-machine:~/ros_ws$
```

After starting the little turtle node, and then running the a_new_turtle program, you will find that another little turtle will appear on the screen. This is because the little turtle node provides service/spawn, which corresponds to `ros::ServiceClient new_turtle in the code. = node.serviceClient<turtlesim::Spawn>("/spawn");//` Creating this service will generate another little turtle turtle2. To view the services provided by the little turtle, you can view it through the rosservice list command, as shown in the figure below.



```
yahboom@yahboom-virtual-machine:~/ros_ws$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rqt_gui_py_node_10503/get_loggers
/rqt_gui_py_node_10503/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtle2/set_pen
/turtle2/teleport_absolute
/turtle2/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

You can view the parameters required by this service through rosservice info /spawn, as shown in the figure below.

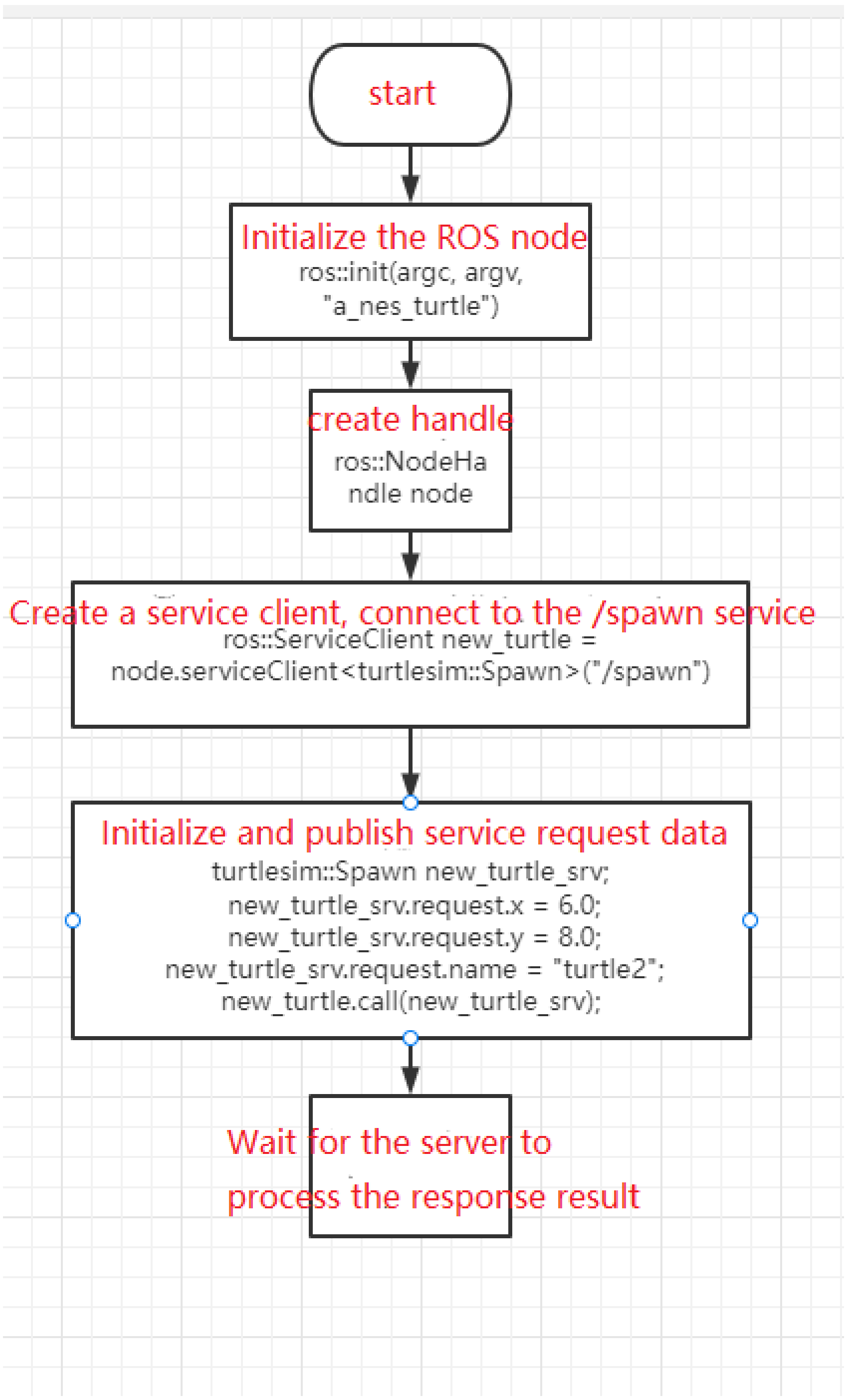


```
yahboom@yahboom-virtual-machine:~/ros_ws$ rosservice info /spawn
Node: /turtlesim
URI: rosrpc://localhost:50617
Type: turtlesim/Spawn
Args: x y theta name
```

It can be seen that 4 parameters are required: x, y, theta, name. These four parameters are initialized in a_new_turtle.cpp.

```
srv.request.x = 6.0;
srv.request.y = 8.0;
srv.request.name = "turtle2";
Note: theta is not assigned a value and defaults to 0
```

9.3.5 Program flow chart


start
Initialize the ROS node
ros::init(argc, argv, "a_nes_turtle")
create handle
ros::NodeHandle node
Create a service client, connect to the /spawn service
ros::ServiceClient new_turtle = node.serviceClient<turtlesim::Spawn>("/spawn")
Initialize and publish service request data
turtlesim::Spawn new_turtle_srv;
new_turtle_srv.request.x = 6.0;
new_turtle_srv.request.y = 8.0;
new_turtle_srv.request.name = "turtle2";
new_turtle.call(new_turtle_srv);
Wait for the server to process the response result

9.4 Python version

9.4.1 Writing source code

Create a new scripts folder under the function package learn_service, then create a new python file (file suffix .py) in this scripts folder, name it a_new_turtle.py, copy and paste the following program code into the a_new_turtle.py file,

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import rospy
from turtlesim.srv import Spawn
def turtle_spawn():
    rospy.init_node('new_turtle')# ROS node initialization
    rospy.wait_for_service('/spawn')# wait/spawn service
    try:
        new_turtle = rospy.ServiceProxy('/spawn', Spawn)
        response = new_turtle(2.0, 2.0, 0.0, "turtle2")# Enter request data
        return response.name
    except rospy.ServiceException as e:
        print ("failed to call service : %s")
if __name__ == "__main__":
    #Call the service and display the call results
    print ("a new turtle named %s." %(turtle_spawn()))
```

The python program does not need to be compiled, but it needs to add executable permissions and enter it in the terminal.

```
cd ~/ros_ws/src/learn_service/scripts
sudo chmod a+x a_new_turtle.py
```

9.4.2 Run

Open roscore,

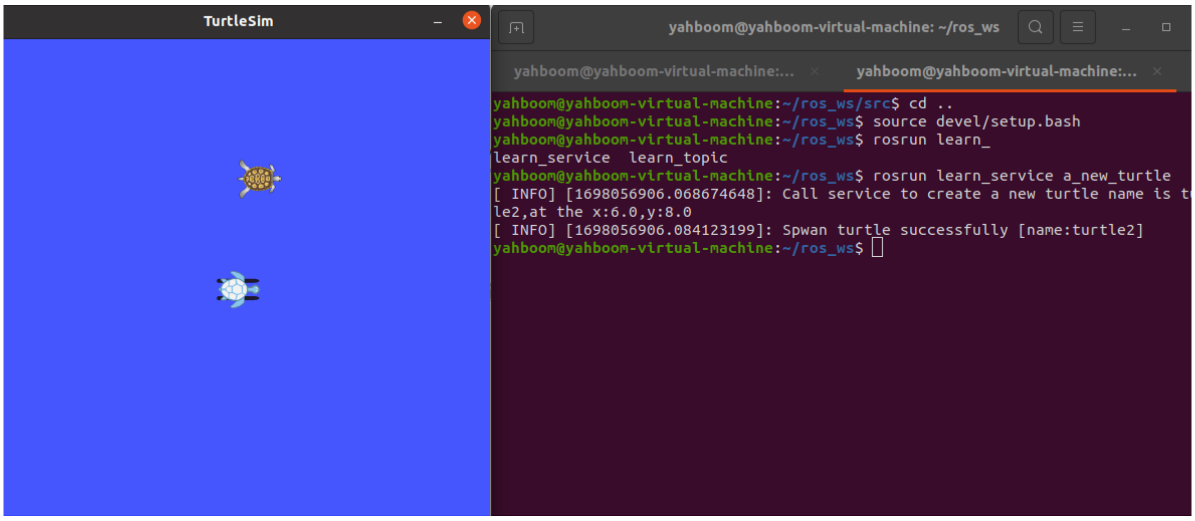
```
rocore
```

Run the little turtle node,

```
rosrun turtlesim turtlesim_node
```

Run the publisher node program and continue to send speed to the little turtle.

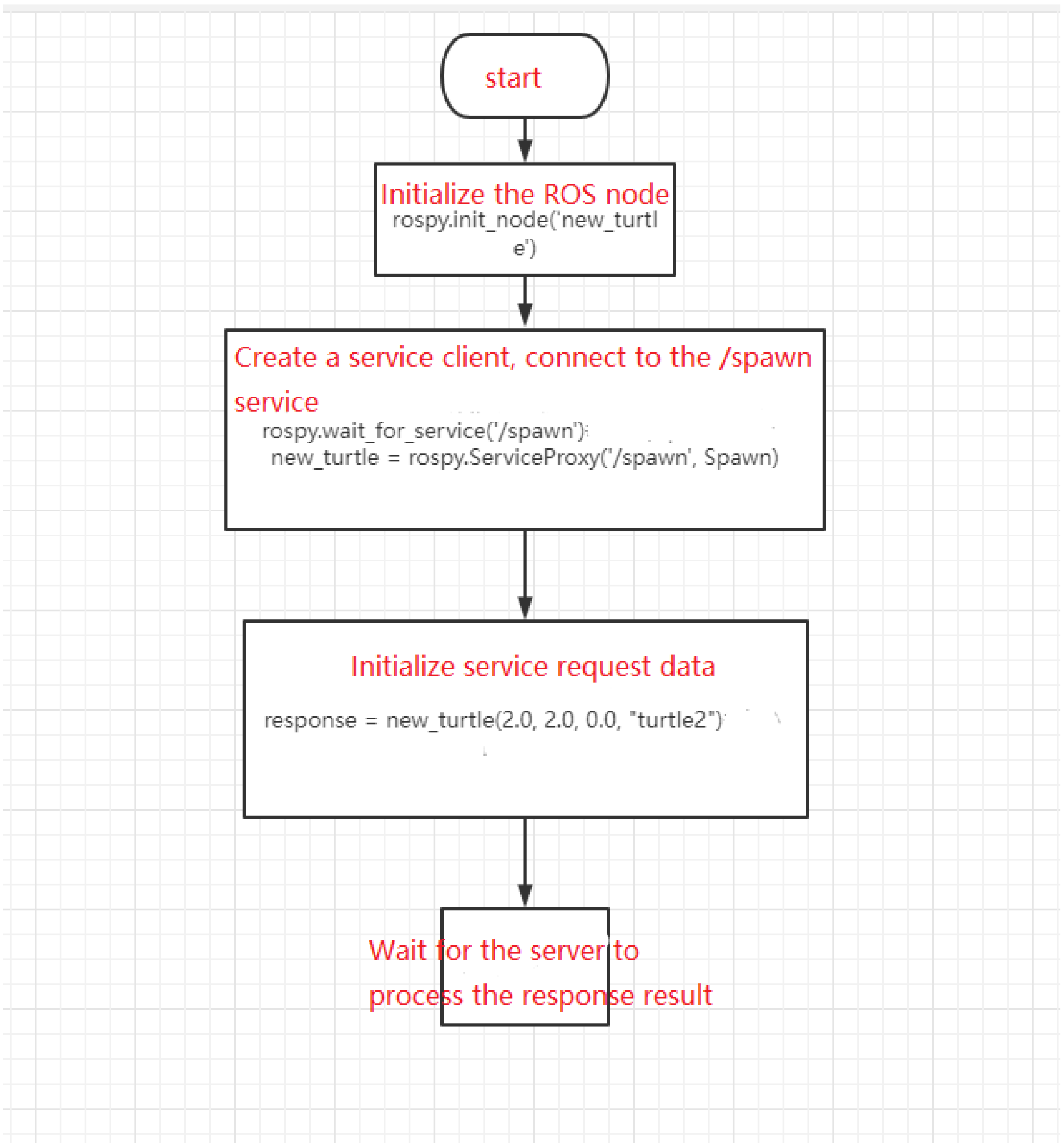
```
rosrun learn_service a_new_turtle.py
```


TurtleSim
yahboom@yahboom-virtual-machine: ~/ros_ws
yahboom@yahboom-virtual-machine:...
yahboom@yahboom-virtual-machine:...

```
yahboom@yahboom-virtual-machine:~/ros_ws/src$ cd ..
yahboom@yahboom-virtual-machine:~/ros_ws$ source devel/setup.bash
yahboom@yahboom-virtual-machine:~/ros_ws$ rosrun learn_
learn_service  learn_topic
yahboom@yahboom-virtual-machine:~/ros_ws$ rosrun learn_service a_new_turtle
[ INFO] [1698056906.068674648]: Call service to create a new turtle name is t
le2,at the x:6.0,y:8.0
[ INFO] [1698056906.084123199]: Spwan turtle successfully [name:turtle2]
yahboom@yahboom-virtual-machine:~/ros_ws$ 
```

Similarly, after running, a little turtle will appear, and the terminal will print the returned content.

9.4.3 Program flow chart


start
Initialize the ROS node
rospy.init_node('new_turtle')
Create a service client, connect to the /spawn service
rospy.wait_for_service('/spawn')
new_turtle = rospy.ServiceProxy('/spawn', Spawn)
Initialize service request data
response = new_turtle(2.0, 2.0, 0.0, "turtle2")
Wait for the server to process the response result