山东大学 计算机科学与技术 学院

课程实验报告

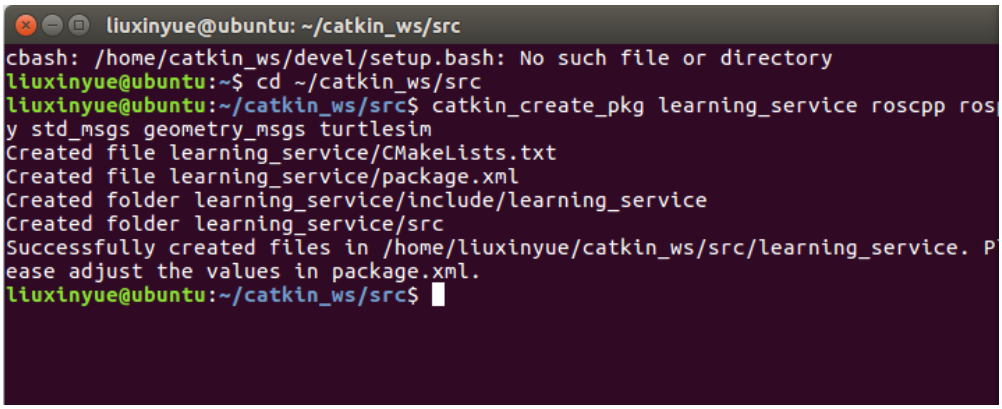
学号：202100130052	姓名：刘欣月	班级：人工智能班
实验题目：机器人操作系统 ROS 自定义服务的请求和服务		
实验学时：2	实验日期：20210510	
实验目的：学习自定义服务的请求和服务		
实验环境：Ubuntu１６　ＲＯＳ		

实验步骤：

实验六：自定义服务的请求和服务

１.客户端 Client 的编程实现，通过程序实现生成一个小海龟，用客户端节点产生一个服务请求，用服务端节点帮忙产生一个小海龟

（１）功能包的产生，使用如下命令来产生 learning_service 功能包文件夹



（２）创建 Client 源程序，进入 learning_sevice 功能包文件夹下 src 文件夹，建立 turtle_spawn.cpp 文件如下所示：

turtle_spawn.cpp (~/catkin_ws/src/learning_service/src) - gedit

```
/*********************************************************************
Copyright 2020 GuYueHome (www.guyuehome.com).
*********************************************************************/

/**
 * 该例程将请求/spawn服务，服务数据类型turtlesim::Spawn
 */

#include <ros/ros.h>
#include <turtlesim/Spawn.h>

int main(int argc, char** argv)
{
    // 初始化ROS节点
        ros::init(argc, argv, "turtle_spawn");

    // 创建节点句柄
        ros::NodeHandle node;

    // 发现/spawn服务后，创建一个服务客户端，连接名为/spawn的service
        ros::service::waitForService("/spawn");
        ros::ServiceClient add_turtle = node.serviceClient<turtlesim::Spawn>("/spa

    // 初始化turtlesim::Spawn的请求数据
        turtlesim::Spawn srv;
        srv.request.x = 2.0;
        srv.request.y = 2.0;
        srv.request.name = "turtle2";

    // 请求服务调用
        ROS_INFO("Call service to spwan turtle[x:%0.6f, y:%0.6f, name:%s]",
                         srv.request.x, srv.request.y, srv.request.name.c_str());

        add_turtle.call(srv);

        // 显示服务调用结果
        ROS_INFO("Spwan turtle successfully [name:%s]", srv.response.name.c_str()

        return 0;
};
```

（3）编译 Client 源程序，首先妹纸 learning_service 文件下 CMakeFile.txt 文件，在合适的位置插入如下代码，

CMakeLists.txt (~/catkin_ws/src/learning_service) - gedit

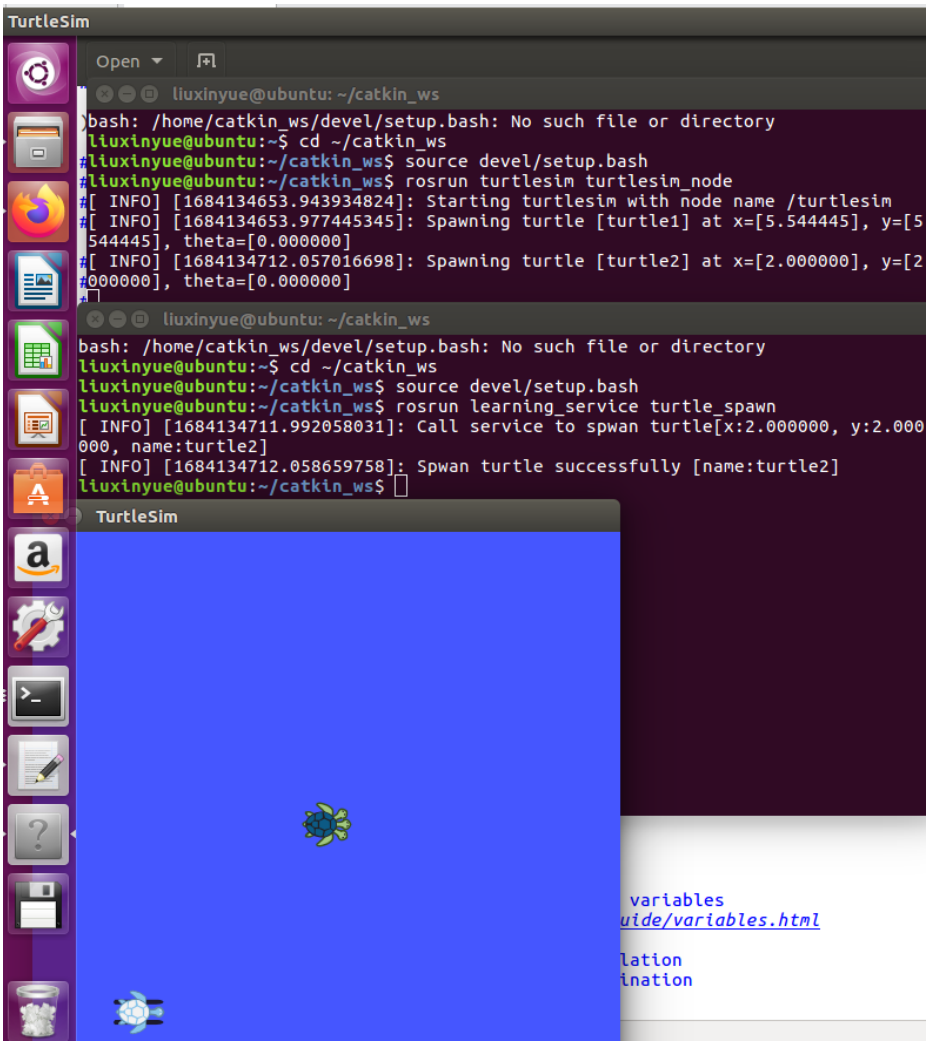
```
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/learning_service.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_E

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/learning_service_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames th
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_no
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX ""

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catk

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

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

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
```

保存好后，输入 ca ~/catkin_ws catkin_make 命令

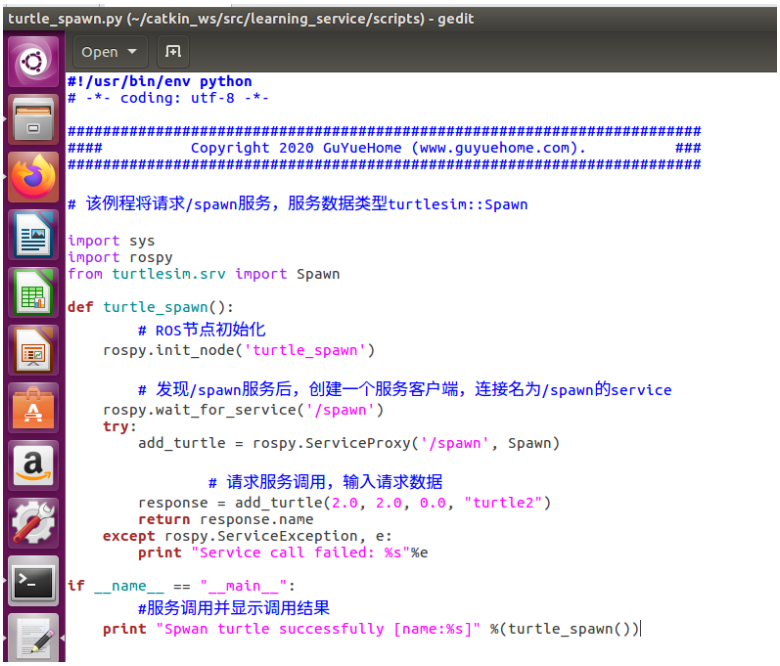
liuxinyue@ubuntu: ~/catkin_ws

```
Scanning dependencies of target turtle_spawn
[  5%] Building CXX object learning_service/CMakeFiles/turtle_spawn.dir/src/turtle_spawn.cpp.o
[ 15%] Built target my_hello_world_node
[ 15%] Built target _learning_topic_generate_messages_check_deps_Person
[ 15%] Built target std_msgs_generate_messages_cpp
[ 26%] Built target pose_subscriber
[ 36%] Built target velocity_publisher
[ 36%] Built target std_msgs_generate_messages_eus
[ 36%] Built target std_msgs_generate_messages_nodejs
[ 36%] Built target std_msgs_generate_messages_lisp
[ 36%] Built target std_msgs_generate_messages_py
[ 42%] Built target learning_topic_generate_messages_cpp
[ 52%] Built target learning_topic_generate_messages_eus
[ 57%] Built target learning_topic_generate_messages_nodejs
[ 63%] Built target learning_topic_generate_messages_lisp
[ 73%] Built target learning_topic_generate_messages_py
[ 84%] Built target person_publisher
[ 94%] Built target person_subscriber
[ 94%] Built target learning_topic_generate_messages
[100%] Linking CXX executable /home/liuxinyue/catkin_ws/devel/lib/learning_service/turtle_spawn
[100%] Built target turtle_spawn
liuxinyue@ubuntu:~/catkin_ws$
```

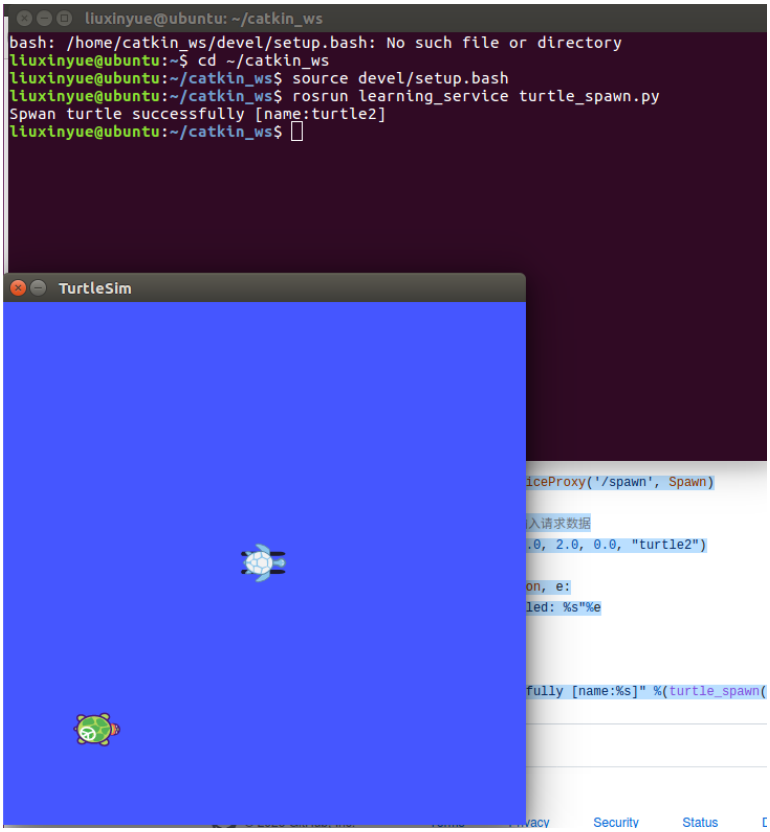
（4）执行 Client 源程序，终端输入 roscore,rosrun turtlesim turtlesim_node ,rosrun learning_service turtle_spawn



Pythone 的实现，代码如下所示



在终端输入：roscore ，rosrun turtlesim turtlesim_node rosrun learning_service turtle_spawn.py 结果如下：



2．服务端（server）的编程实现，server 端的主要功能时通过 topic 指令向小海龟发送速度的指令，Client 端通过发送服务请求控制 Server

端是否给小孩会发送指令。

（1）创建 server 源程序，在 learning_service 文件夹下的 src 文件夹下创建 turle_command_server.cpp 文件如下所示：

*turtle_comand_server.cpp (~/catkin_ws/src/learning_service/src) - gedit

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

ros::Publisher turtle_vel_pub;
bool pubCommand = false;

// service回调函数，输入参数req，输出参数res
bool commandCallback(std_srvs::Trigger::Request  &req,
                     std_srvs::Trigger::Response &res)
{
        pubCommand = !pubCommand;

    // 显示请求数据
    ROS_INFO("Publish turtle velocity command [%s]", pubCommand==true?"Yes":"No")

        // 设置反馈数据
        res.success = true;
        res.message = "Change turtle command state!";

    return true;
}

int main(int argc, char **argv)
{
    // ROS节点初始化
    ros::init(argc, argv, "turtle_command_server");

    // 创建节点句柄
    ros::NodeHandle n;

    // 创建一个名为/turtle_command的server，注册回调函数commandCallback
    ros::ServiceServer command_service = n.advertiseService("/turtle_command", co

        // 创建一个Publisher，发布名为/turtle1/cmd_vel的topic，消息类型为geometry_m
        turtle_vel_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 10

    // 循环等待回调函数
    ROS_INFO("Ready to receive turtle command.");

        // 设置循环的频率
        ros::Rate loop_rate(10);
```

```
bash: /home/catkin_ws/devel/setup.bash: No such file or directory
liuxinyue@ubuntu:~$ rossrv show std_srvs/Trigger
---
bool success
string message

liuxinyue@ubuntu:~$
```

（2）编译 server 源程序，在 CMakeFile.txt 文件中插入代码，

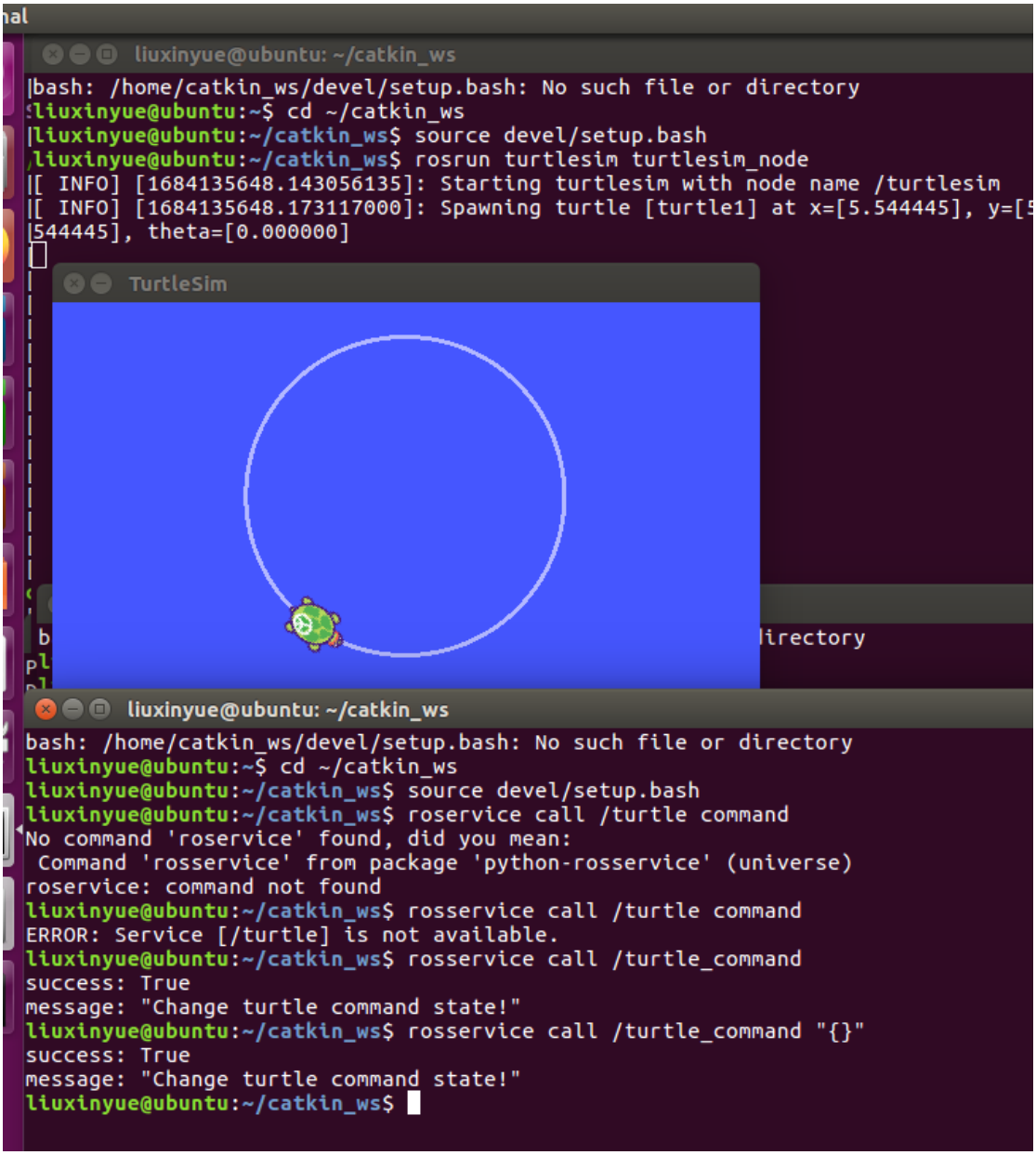
```
*CMakeLists.txt (~/catkin_ws/src/learning_service) - gedit
Open
## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collid
# add_executable(${PROJECT_NAME}_node src/learning_service_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${ca

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

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

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

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_execut
# install(TARGETS ${PROJECT_NAME}_node
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
```

然后进行编译，编译的结果如下所示：

```
liuxinyue@ubuntu: ~/catkin_ws
[ 19%] Built target my_hello_world_node
Scanning dependencies of target turtle_command_server
[ 23%] Building CXX object learning_service/CMakeFiles/turtle_command_server.dir
/src/turtle_command_server.cpp.o
[ 23%] Built target _learning_topic_generate_messages_check_deps_Person
[ 23%] Built target std_msgs_generate_messages_cpp
[ 33%] Built target pose_subscriber
[ 42%] Built target velocity_publisher
[ 42%] Built target std_msgs_generate_messages_eus
[ 42%] Built target std_msgs_generate_messages_nodejs
[ 42%] Built target std_msgs_generate_messages_lisp
[ 42%] Built target std_msgs_generate_messages_py
[ 47%] Built target learning_topic_generate_messages_cpp
[ 57%] Built target learning_topic_generate_messages_eus
[ 61%] Built target learning_topic_generate_messages_nodejs
[ 66%] Built target learning_topic_generate_messages_lisp
[ 76%] Built target learning_topic_generate_messages_py
[ 85%] Built target person_publisher
[ 95%] Built target person_subscriber
[ 95%] Built target learning_topic_generate_messages
[100%] Linking CXX executable /home/liuxinyue/catkin_ws/devel/lib/learning_servi
ce/turtle_command_server
[100%] Built target turtle_command_server
liuxinyue@ubuntu:~/catkin_ws$
```

（3）执行 server 源程序，再打开终端输入 roscore，rosrun turlesim turlesim_node ，rosrun learning_service turtle_command_server，Rosservice call /turtle command- “{}”

结果如下所示：



Python 程序，在 scripts 文件夹下创建 turtle_command_server.py 文件修改文件权限，编写代码：

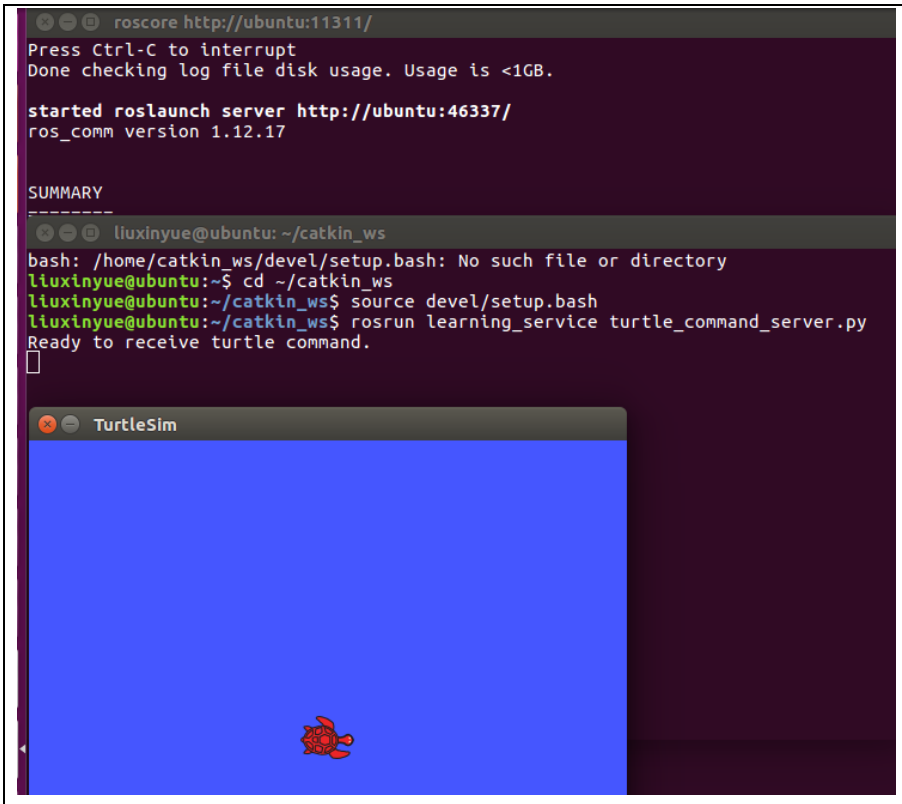
turtle_command_server.py (~/catkin_ws/src/learning_service/scripts) - gedit

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

pubCommand = False;
turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

def command_thread():
        while True:
                if pubCommand:
                        vel_msg = Twist()
                        vel_msg.linear.x = 0.5
                        vel_msg.angular.z = 0.2
                        turtle_vel_pub.publish(vel_msg)

                time.sleep(0.1)

def commandCallback(req):
        global pubCommand
        pubCommand = bool(1-pubCommand)

        # 显示请求数据
        rospy.loginfo("Publish turtle velocity command![%d]", pubCommand)

        # 反馈数据
        return TriggerResponse(1, "Change turtle command state!")

def turtle_command_server():
        # ROS节点初始化
    rospy.init_node('turtle_command_server')

        # 创建一个名为/turtle_command的server，注册回调函数commandCallback
    s = rospy.Service('/turtle_command', Trigger, commandCallback)

        # 循环等待回调函数
    print "Ready to receive turtle command."

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

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

在终端输入：roscore

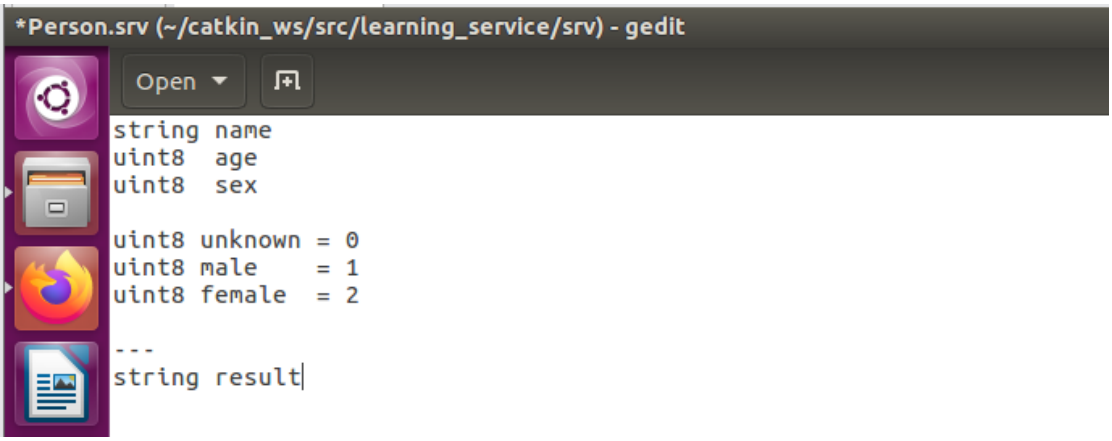
Rosrun turtlesim turtlesim_node

Rosrun learning_service turtle_command_server.py

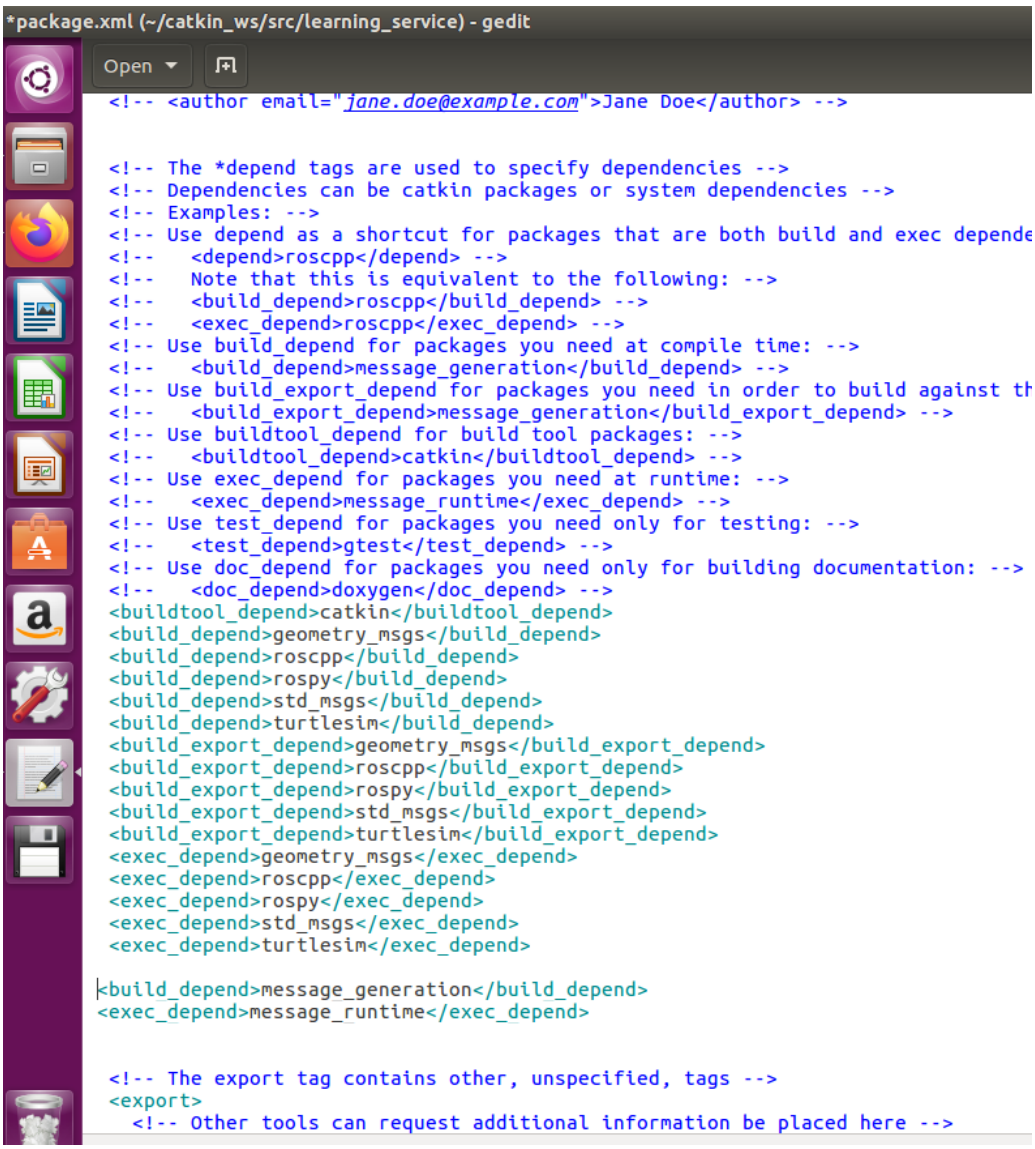


３．自定义服务数据类型的编程实现

（１）自定义服务数据类型文件，在 srv 文件下建立 Person.srv 文件，输入如下内容：

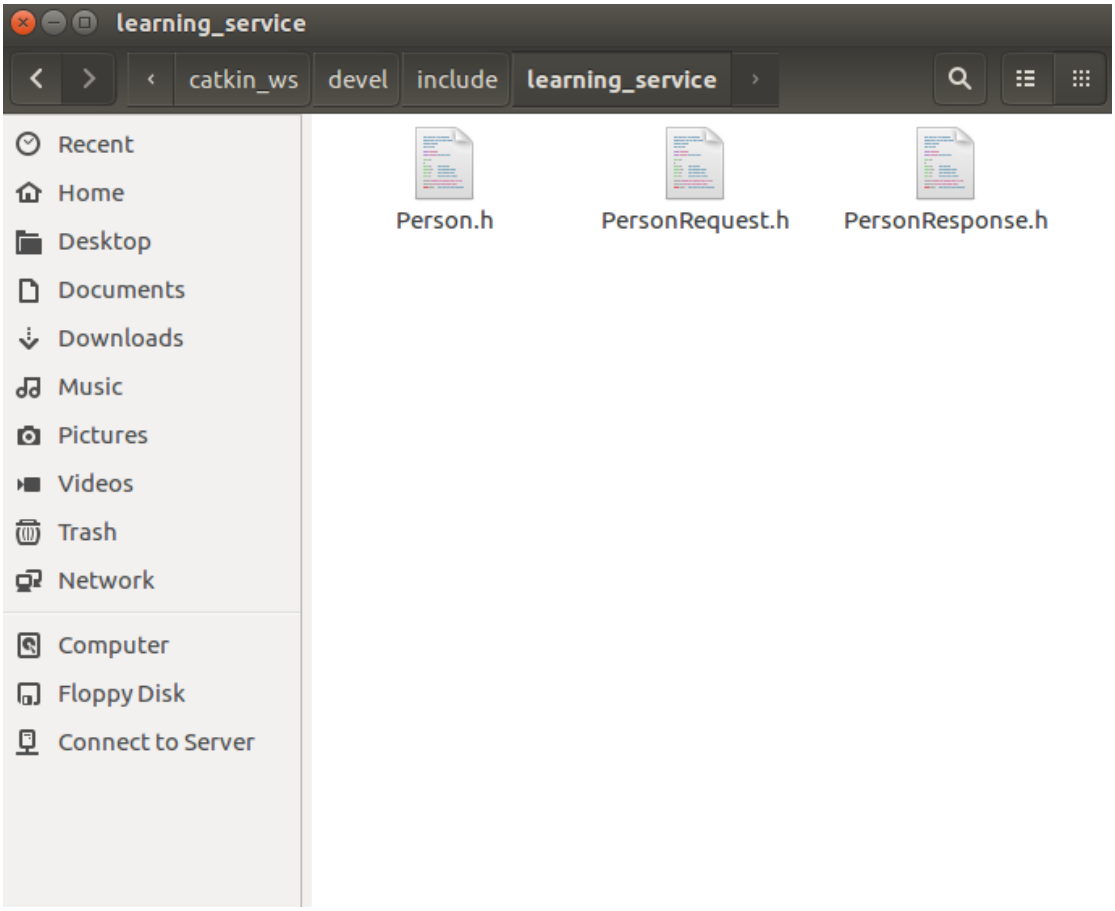


在打开 learning_service 文件夹下的 epackage.xml 文件，添加相关的编程依赖，在 package.xml 添加配置。



对功能包进行编译，编译结果如下所示：

```
liuxinyue@ubuntu: ~/catkin_ws
[ 57%] Built target learning_topic_generate_messages_py
[ 60%] Generating Javascript code from learning_service/Person.srv
Scanning dependencies of target learning_service_generate_messages_py
[ 64%] Generating Python code from SRV learning_service/Person
[ 64%] Built target learning_service_generate_messages_nodejs
Scanning dependencies of target learning_service_generate_messages_lisp
[ 67%] Generating Lisp code from learning_service/Person.srv
[ 67%] Built target learning_service_generate_messages_lisp
Scanning dependencies of target learning_service_generate_messages_eus
[ 71%] Generating Python srv __init__.py for learning_service
[ 75%] Generating EusLisp code from learning_service/Person.srv
[ 78%] Generating EusLisp manifest code for learning_service
[ 78%] Built target learning_service_generate_messages_py
Scanning dependencies of target learning_service_generate_messages_cpp
[ 82%] Generating C++ code from learning_service/Person.srv
[ 82%] Built target learning_service_generate_messages_cpp
[ 85%] Built target learning_topic_generate_messages_cpp
[ 92%] Built target person_publisher
[100%] Built target person_subscriber
[100%] Built target learning_topic_generate_messages
[100%] Built target learning_service_generate_messages_eus
Scanning dependencies of target learning_service_generate_messages
[100%] Built target learning_service_generate_messages
liuxinyue@ubuntu:~/catkin_ws$
```



（2） 设计服务端 server，发布者的文件夹名为 person_sever.cpp


```
/**
 * 该例程将执行/show_person服务，服务数据类型learning_service::Person
 */

#include <ros/ros.h>
#include "learning_service/Person.h"

// service回调函数，输入参数req，输出参数res
bool personCallback(learning_service::Person::Request  &req,
                    learning_service::Person::Response &res)
{
    // 显示请求数据
    ROS_INFO("Person: name:%s  age:%d  sex:%d", req.name.c_str(), req.age, req

        // 设置反馈数据
        res.result = "OK";

    return true;
}

int main(int argc, char **argv)
{
    // ROS节点初始化
    ros::init(argc, argv, "person_server");

    // 创建节点句柄
    ros::NodeHandle n;

    // 创建一个名为/show_person的server，注册回调函数personCallback
    ros::ServiceServer person_service = n.advertiseService("/show_person", per

    // 循环等待回调函数
    ROS_INFO("Ready to show person informtion.");
    ros::spin();

    return 0;
}
```

（3） 设计客户端 client，文件名 person_client.cpp，如下所示：

person_client.cpp (~/catkin_ws/src/learning_service/src) - gedit

```
/**
 * 该例程将请求/show_person服务，服务数据类型learning_service::Person
 */

#include <ros/ros.h>
#include "learning_service/Person.h"

int main(int argc, char** argv)
{
    // 初始化ROS节点
        ros::init(argc, argv, "person_client");

    // 创建节点句柄
        ros::NodeHandle node;

    // 发现/spawn服务后，创建一个服务客户端，连接名为/spawn的service
        ros::service::waitForService("/show_person");
        ros::ServiceClient person_client = node.serviceClient<learning_servic

    // 初始化learning_service::Person的请求数据
        learning_service::Person srv;
        srv.request.name = "Tom";
        srv.request.age  = 20;
        srv.request.sex  = learning_service::Person::Request::male;

    // 请求服务调用
        ROS_INFO("Call service to show person[name:%s, age:%d, sex:%d]",
                         srv.request.name.c_str(), srv.request.age, srv.reque

        person_client.call(srv);

        // 显示服务调用结果
        ROS_INFO("Show person result : %s", srv.response.result.c_str());

        return 0;
};
```

（4）编译服务端 server 和客户端 client，修改 CmakeFile.txt 问价将如下的六句代码插入。

CMakeLists.txt (~/catkin_ws/src/learning_service) - gedit

Open

person_client.cpp

```
## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following rename
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pk
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFI

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )
add_executable(turtle_spawn src/turtle_spawn.cpp)
target_link_libraries(turtle_spawn ${catkin_LIBRARIES})

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

add_executable(person_server src/person_server.cpp)
target_link_libraries(person_server ${catkin_LIBRARIES})
add_dependencies(person_server ${PROJECT_NAME}_gencpp)

add_executable(person_client src/person_client.cpp)
target_link_libraries(person_client ${catkin_LIBRARIES})
add_dependencies(person_client ${PROJECT_NAME}_gencpp)
#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_exec
```

然后进行编译：

```
liuxinyue@ubuntu: ~/catkin_ws
[ 56%] Built target learning_topic_generate_messages_cpp
[ 62%] Built target learning_topic_generate_messages_eus
[ 65%] Built target learning_topic_generate_messages_nodejs
[ 68%] Built target learning_topic_generate_messages_lisp
[ 75%] Built target learning_topic_generate_messages_py
Scanning dependencies of target learning_service_gencpp
[ 75%] Built target learning_service_gencpp
[ 75%] Built target learning_service_generate_messages
[ 81%] Built target person_publisher
[ 87%] Built target person_subscriber
Scanning dependencies of target person_server
[ 87%] Built target learning_topic_generate_messages
Scanning dependencies of target person_client
[ 90%] Building CXX object learning_service/CMakeFiles/person_server.dir/src/
son_server.cpp.o
[ 93%] Building CXX object learning_service/CMakeFiles/person_client.dir/src/
son_client.cpp.o
[ 96%] Linking CXX executable /home/liuxinyue/catkin_ws/devel/lib/learning_se
ce/person_client
[ 96%] Built target person_client
[100%] Linking CXX executable /home/liuxinyue/catkin_ws/devel/lib/learning_se
ce/person_server
[100%] Built target person_server
liuxinyue@ubuntu:~/catkin_ws$
```

（5）执行发布者 publisher 和订阅者 subscriber，在终端输入

Roscore

Rosrun learning_service person_server

Rosrun learning_serive person_client

```
liuxinyue@ubuntu: ~/catkin_ws
bash: /home/catkin_ws/devel/setup.bash: No such file or directory
liuxinyue@ubuntu:~$ cd ~/catkin_ws
liuxinyue@ubuntu:~/catkin_ws$ source devel/srtup.bash
bash: devel/srtup.bash: No such file or directory
liuxinyue@ubuntu:~/catkin_ws$ source devel/setup.bash
liuxinyue@ubuntu:~/catkin_ws$ rosrun learning_service person_server
[ INFO] [1684139978.565589292]: Ready to show person informtion.
[ INFO] [1684140015.320981359]: Person: name:Tom  age:20  sex:1
```

```
liuxinyue@ubuntu: ~/catkin_ws
bash: /home/catkin_ws/devel/setup.bash: No such file or directory
liuxinyue@ubuntu:~$ cd ~/catkin_ws
liuxinyue@ubuntu:~/catkin_ws$ source devel/setup.bash
liuxinyue@ubuntu:~/catkin_ws$ rosrun learning_service person_client
[ INFO] [1684140015.315047826]: Call service to show person[name:Tom, age:20,
x:1]
[ INFO] [1684140015.322109787]: Show person result : OK
liuxinyue@ubuntu:~/catkin_ws$
```

Python 实现创建一个服务端的源文件 person_server.py 内容如下所

示：

person_server.py (~/catkin_ws/src/learning_service/scripts) - gedit

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

##########################################################################
####          Copyright 2020 GuYueHome (www.guyuehome.com).          ###
##########################################################################

# 该例程将执行/show_person服务，服务数据类型learning_service::Person

import rospy
from learning_service.srv import Person, PersonResponse

def personCallback(req):
        # 显示请求数据
    rospy.loginfo("Person: name:%s  age:%d  sex:%d", req.name, req.age, req.se

        # 反馈数据
    return PersonResponse("OK")

def person_server():
        # ROS节点初始化
    rospy.init_node('person_server')

        # 创建一个名为/show_person的server，注册回调函数personCallback
    s = rospy.Service('/show_person', Person, personCallback)

        # 循环等待回调函数
    print "Ready to show person informtion."
    rospy.spin()

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

创建客户端的源文件 person_client.py 修改文件权限，编写内容如下：

person_client.py (~/catkin_ws/src/learning_service/scripts) - gedit

Open

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

##########################################################################
####          Copyright 2020 GuYueHome (www.guyuehome.com).          ###
##########################################################################

# 该例程将请求/show_person服务，服务数据类型learning_service::Person

import sys
import rospy
from learning_service.srv import Person, PersonRequest

def person_client():
        # ROS节点初始化
    rospy.init_node('person_client')

        # 发现/spawn服务后，创建一个服务客户端，连接名为/spawn的service
    rospy.wait_for_service('/show_person')
    try:
        person_client = rospy.ServiceProxy('/show_person', Person)

                # 请求服务调用，输入请求数据
        response = person_client("Tom", 20, PersonRequest.male)
        return response.result
    except rospy.ServiceException, e:
        print "Service call failed: %s"%e

if __name__ == "__main__":
        #服务调用并显示调用结果
    print "Show person result : %s" %(person_client())
```

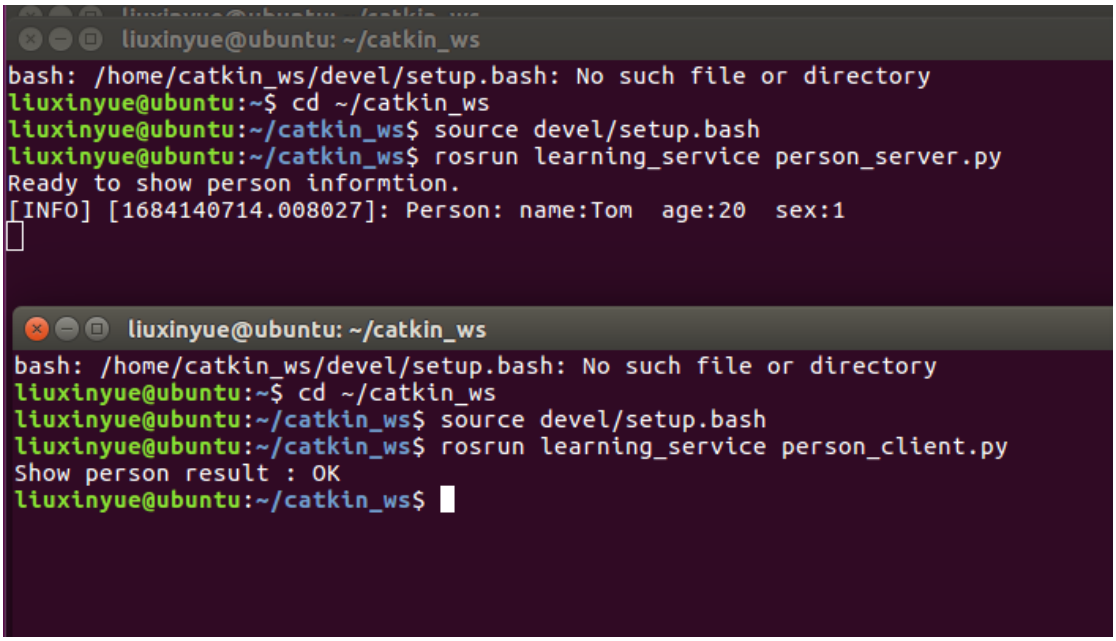
在终端输入；

Roscore

Rosrun learning_service person_server.py

Rosrun learning_service person_client.py

结果如下所示：


liuxinyue@ubuntu: ~/catkin_ws
bash: /home/catkin_ws/devel/setup.bash: No such file or directory
liuxinyue@ubuntu:~$ cd ~/catkin_ws
liuxinyue@ubuntu:~/catkin_ws$ source devel/setup.bash
liuxinyue@ubuntu:~/catkin_ws$ rosrun learning_service person_server.py
Ready to show person informtion.
[INFO] [1684140714.008027]: Person: name:Tom age:20 sex:1
liuxinyue@ubuntu: ~/catkin_ws
bash: /home/catkin_ws/devel/setup.bash: No such file or directory
liuxinyue@ubuntu:~$ cd ~/catkin_ws
liuxinyue@ubuntu:~/catkin_ws$ source devel/setup.bash
liuxinyue@ubuntu:~/catkin_ws$ rosrun learning_service person_client.py
Show person result : OK
liuxinyue@ubuntu:~/catkin_ws$