

ros-python学习样例

- ros-python学习样例
 - 1.通信基本原理介绍
 - 2.三种通信方式的程序样例(python版)
 - 2.1 topic 通信方式(非自定义和自定义)
 - 2.1.1 创建工作空间和topic功能包
 - 2.1.2 topic消息发布者(Publisher)部分
 - 2.1.3 topic消息接收者(Subscriber)部分
 - 2.1.3 编译并运行代码
 - 2.1.4 自定义消息
 - a.自定义.msg文件
 - b.修改 package.xml
 - c.修改 CMakeList.txt
 - d.修改消息发送者文件 (Publisher.py)
 - e.修改消息接收者文件 (Subscriber.py)
 - f.运行程序
 - 2.2 service 通信方式(自定义)
 - 2.2.1 自定义.srv文件
 - 2.2.2 编写服务器端(server)部分代码
 - 2.2.3 编写客户端(client)部分代码
 - 2.2.4 修改CMakeList.txt文件
 - 2.2.5 修改package.xml文件
 - 2.2.6 运行service_test程序
 - 2.3 action通信方式(不完整)
 - 2.3.1 创建 action_test功能包
 - 2.3.2 创建自定义 .action文件
 - 2.3.3 修改 CMakeList.txt 和 package.xml 文件
 - 2.3.4 编写action_server服务器端部分程序
 - 2.3.5 编写action_client客户端部分程序
 - 2.3.6 编译并运行程序

1.通信基本原理介绍

待写

2.三种通信方式的程序样例(python版)

2.1 topic 通信方式(非自定义和自定义)

2.1.1 创建工作空间和topic功能包

在ubuntu中打开命令行,输入下面的命令创建并初始化工作空间,一定要回到XXX_ws的目录下初始化工作空间

```

#创建工作空间文件夹my_ros(一般命名为XXX_ws)以及源程序目录src
$ mkdir -p ~/my_ros/src
#必须回到工作目录下,才开始初始化工作空间
$ cd ~/my_ros
#初始化工作空间,同时该命令还可以用于编译源代码
$ catkin_make
#进入src目录,创建自定义功能包
$ cd src
#创建功能包,指定了功能包的包名为"topic_test"
#引入的依赖有rospy(ros中python编程的必备接口),
#std_msgs(标准消息接口)和
#message_generation(用于自定义消息的接口)
$ catkin_create_pkg topic_test rospy std_msgs message_generation
#创建python源代码的目录,用于保存.py文件
$ mkdir script

```

主要运行的python程序分为两部分: **消息发布者(Publisher)** 和 **消息接受者(Subscriber)**

由于此例中只使用到ros中提供的std_msgs中的String类型消息,所以对于初始化时自动生成的 CMakeList.txt 和 package.xml 无需修改

2.1.2 topic消息发布者(Publisher)部分

在终端中输入如下代码,开始编辑 **消息发布者(Publisher)** 的代码文件

```

#进入script目录
$ cd ~/my_ros/topic_test/script
#打开编辑器编辑代码,若文件不存在将会创建该文件
$ gedit Publisher.py

```

注:也可以使用 RoboWare/eclipse 等的IDE做开发

输入如下所示的python代码 (运行前需删掉中文注释,否则将会报错!)

```

#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    # 初始化节点,向ros操作系统的master节点注册自己的信息,
    # 定义节点唯一的名称为publisher,同时保证名字唯一的参数(anonymous)为true
    # 即,当名字重复时将会在名字后面添加一串随机数字作为注册名字,从而保证名字唯一
    rospy.init_node('pyTalker', anonymous=True)
    # 定义本节点将向
    # rospy的 Publisher 类的构造方法介绍如下:
    # __init__(self, name, data_class, subscriber_listener=None,
    tcp_nodelay=False, latch=False, headers=None, queue_size=None
    # 其中 name (为str类型)代表本话题(topic)的名称

```

```

# data_class(message class) 为序列化的消息(message)类
# subscriber_listener(SubscribeListener) 指定订阅者, 默认值为None, 可以为空
# tcp_nodelay(bool) 为设置发布者(publisher)的socket是否采用Nagle算法(默认为False, 即采用Nagle算法), 通过该设置能够影响发布的效率\能耗
# latch(bool) 如果为真, 最后一条消息将设置为"占用信道", 即没有一个订阅者都会立即收到该消息
# headers(dict) 默认为None, 如果不为空, 将使用传入的"字典"的键值作为未来连接的参数
# queue_size(int) 默认为None(阻塞式同步收发), 指定消息队列的长度, 必须指定, 如果为0(无限缓冲区)是危险的, 如果缓冲队长太大会导致发送不同步
# 该方法抛出 ROSException 异常
pub = rospy.Publisher('test_chat', String, queue_size=1)
# 设定循环的周期为0.1s
rate = rospy.Rate(10)
count = 0
while not rospy.is_shutdown():
    words = 'Happy new year! %d' % count
    # 发布消息的 Publisher类 的核心方法, 参数如下:
    # publish(self, *args, **kwargs)
    # args 要发送的消息, 如果有 keyword 时可以为空
    # kwargs 消息的关键词, 如果 kwargs 被采用, 则 args 一定是未指定的
    pub.publish(words)
    rospy.loginfo('pyTalker: send msg = %s ', words)
    count = count+1
    pub.publish(words)
    rate.sleep()

# 进行标准的Python_main检查, 这里的try-except捕获rospy.ROSInterruptException异常(处理异常中断)
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass

```

2.1.3 topic消息接收者(Subscriber)部分

同理创建 **消息接受者(Subscriber)** 的代码文件

```

#打开编辑器编辑代码, 若文件不存在将会创建该文件
$ gedit Subscriber.py

```

输入如下代码 (运行代码前需将中文注释都删掉, 否则会出错!)

```

#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

# 用于处理从指定topic获取的消息, 尤其需要注意"message.data"这样使用, 才能正确获取全部消

```

```

息
def callback(message):
    rospy.loginfo('Listener: msg = %s', message.data)

# 主调用的方法, 负责向主服务器注册本节点, 以及注册, 链接, 使用指定话题
def listener():
    # 向服务器注册本节点名为 'pyListener', 允许使用避免重名的机制(若重复, 则在名字后加随机数)
    rospy.init_node('pyListener', anonymous=True)
    # 向服务器申请获得含有指定的 'test_chat' 的 topic 发布者的地址, 并连接
    # 该方法的参数中, 'test_chat' 为本节点需要链接的话题(topic)名, 'String' 为获取的消息的类型
    # 'callback' 为处理消息的方法名
    rospy.Subscriber('test_chat', String, callback)
    # 以下的 spin() 方法为 ROS 消息回调处理函数, 调用后将不会返回(下面的程序无法执行), 即阻塞本线程(使得本节点不能再执行其他操作, 只能处理消息)
    # 与这个函数相近功能的函数有 spinOnce(), 但 spinOnce() 会返回(下面的程序仍能执行)且只执行一次, 不会阻塞本线程
    rospy.spin()

# 进行标准的 Python_main 检查
if __name__ == '__main__':
    listener()

```

2.1.3 编译并运行代码

打开一个终端, 输入如下命令

```

#进入工作空间目录
$ cd ~/my_ros
#编译程序(实际上python为脚本语言无需编译)
$ catkin_make

```

这里所说的 " 编译 " 并不是严格意义上的 " 编译 ", 可以理解为是为 ros 运行做好准备(例如 catkin 的操作). 执行过该命令后, 修改 .py 文件后再运行将会出现修改后的结果 (无需再 catkin_make 一次)

开三个终端, 分别输入如下 3 条命令, 开始执行程序

```

#开启服务器, 将负责处理注册/连接请求等
$ roscore
#开始运行 topic_test 功能包下的 消息发送者(Publisher) 节点
$ rosrunc topic_test Publisher.py
#运行 topic_test 包下的 消息接收者(Subscriber) 节点
$ rosrunc topic_test Subscriber.py

```

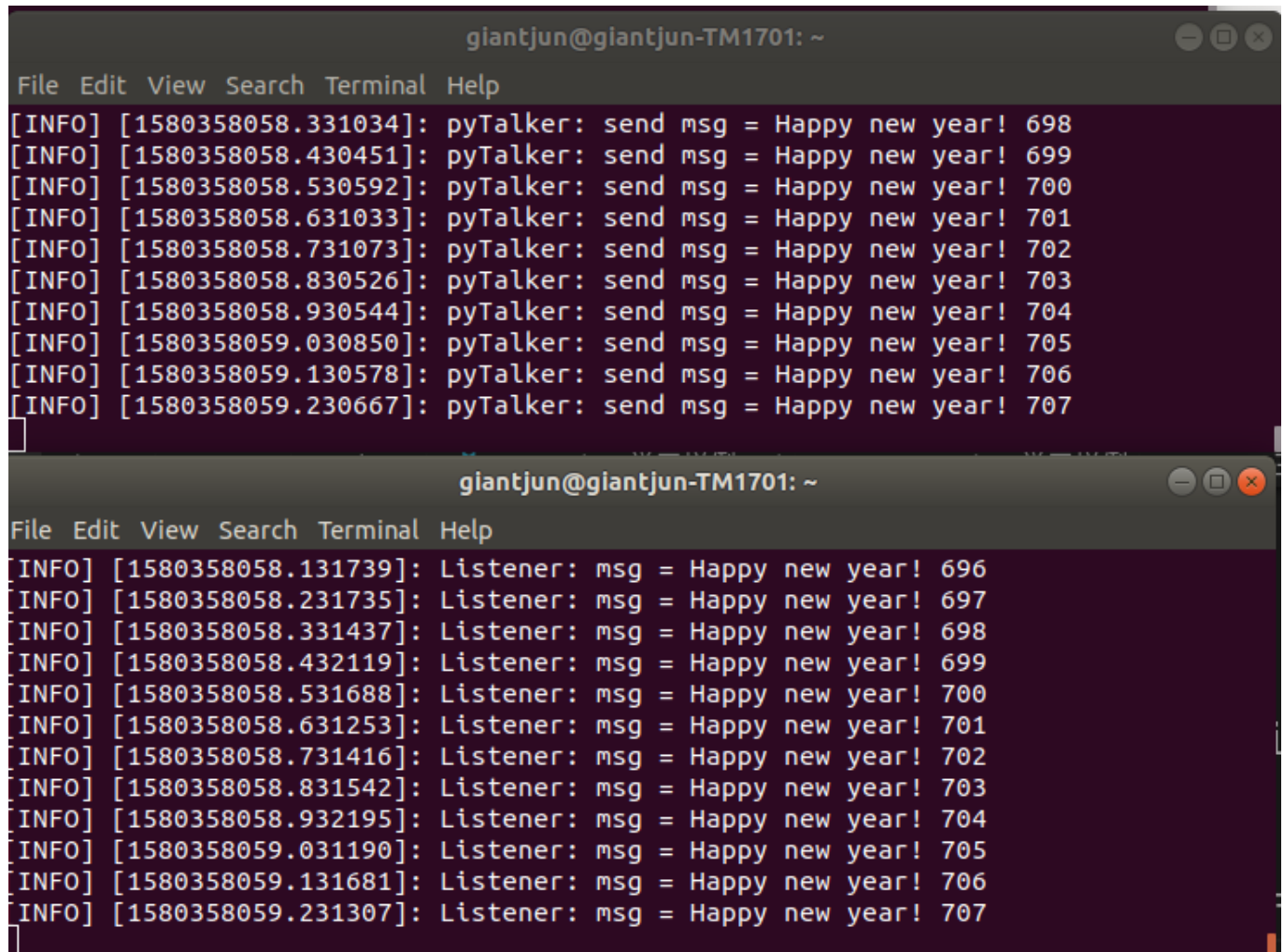
如果是完全按照上面的步骤操作, 如果过遇到程序无法执行/报错, 显示:

```
[roscrun] Coundn't find excutable named Publisher.py below ....
```

可能原因是python文件没有执行权限,输入如下命令赋予权限就可以运行了

```
#必须在.py文件的目录下
$ cd ~/my_ros/script
#赋予运行权限
$ chmod +x Publisher.py
$ chmod +x Subscriber.py
```

程序运行结果为



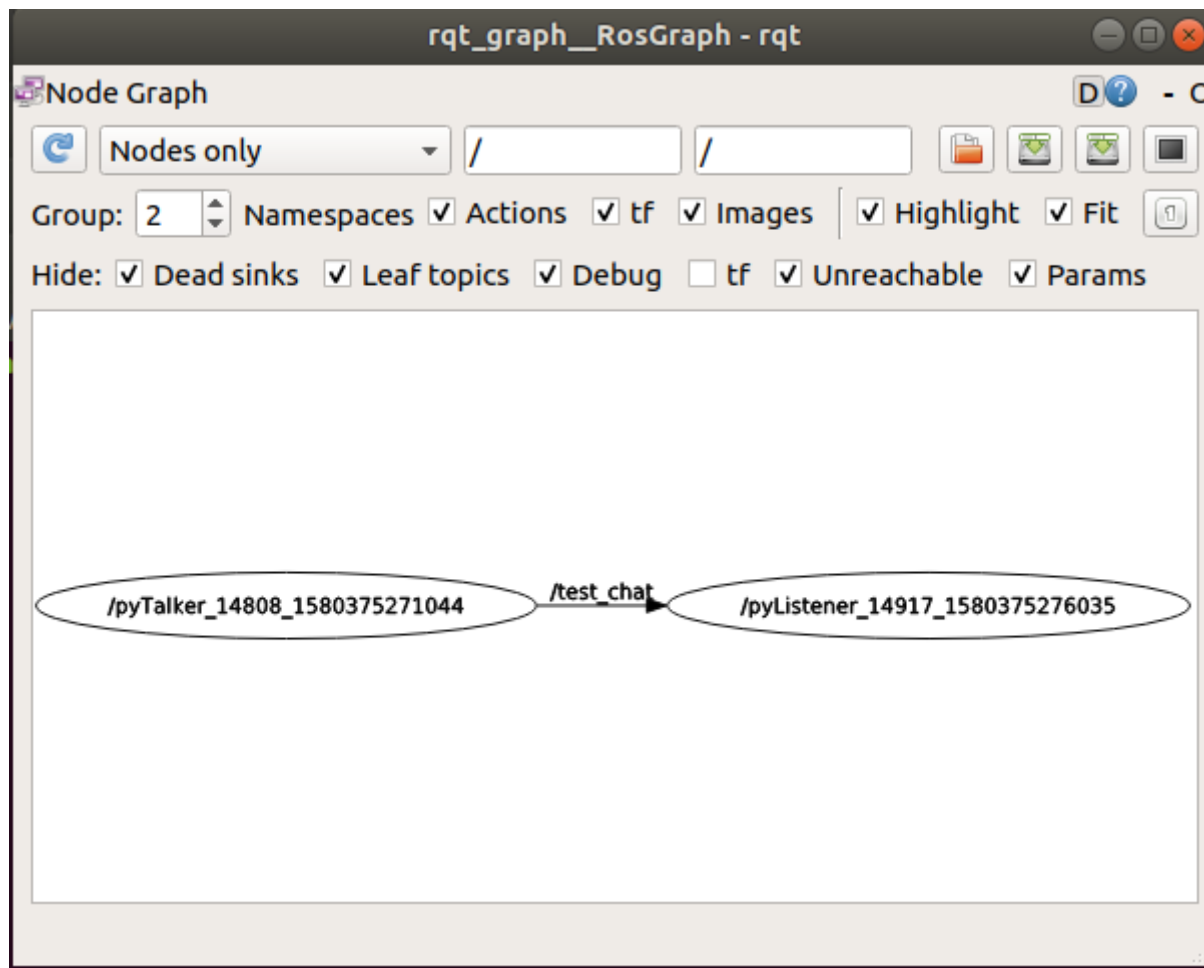
The image shows two terminal windows from a user named 'giantjun' on a machine 'giantjun-TM1701'. Both windows have a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The top window displays logs from a 'pyTalker' node, showing it sending the message 'Happy new year!' at 10 different timestamps, with the sequence number increasing from 698 to 707. The bottom window displays logs from a 'Listener' node, showing it receiving the message 'Happy new year!' at 10 different timestamps, with the sequence number increasing from 696 to 707.

```
giantjun@giantjun-TM1701: ~
File Edit View Search Terminal Help
[INFO] [1580358058.331034]: pyTalker: send msg = Happy new year! 698
[INFO] [1580358058.430451]: pyTalker: send msg = Happy new year! 699
[INFO] [1580358058.530592]: pyTalker: send msg = Happy new year! 700
[INFO] [1580358058.631033]: pyTalker: send msg = Happy new year! 701
[INFO] [1580358058.731073]: pyTalker: send msg = Happy new year! 702
[INFO] [1580358058.830526]: pyTalker: send msg = Happy new year! 703
[INFO] [1580358058.930544]: pyTalker: send msg = Happy new year! 704
[INFO] [1580358059.030850]: pyTalker: send msg = Happy new year! 705
[INFO] [1580358059.130578]: pyTalker: send msg = Happy new year! 706
[INFO] [1580358059.230667]: pyTalker: send msg = Happy new year! 707

giantjun@giantjun-TM1701: ~
File Edit View Search Terminal Help
[INFO] [1580358058.131739]: Listener: msg = Happy new year! 696
[INFO] [1580358058.231735]: Listener: msg = Happy new year! 697
[INFO] [1580358058.331437]: Listener: msg = Happy new year! 698
[INFO] [1580358058.432119]: Listener: msg = Happy new year! 699
[INFO] [1580358058.531688]: Listener: msg = Happy new year! 700
[INFO] [1580358058.631253]: Listener: msg = Happy new year! 701
[INFO] [1580358058.731416]: Listener: msg = Happy new year! 702
[INFO] [1580358058.831542]: Listener: msg = Happy new year! 703
[INFO] [1580358058.932195]: Listener: msg = Happy new year! 704
[INFO] [1580358059.031190]: Listener: msg = Happy new year! 705
[INFO] [1580358059.131681]: Listener: msg = Happy new year! 706
[INFO] [1580358059.231307]: Listener: msg = Happy new year! 707
```

在新的终端输入如下命令,打开rqt工具,可以看到各个节点之间的关系

```
$ rqt_graph
```



2.1.4 自定义消息

自定义消息的使用与使用ros提供的消息一样,不过有几个部分需要注意:

- 需自定义.msg文件
- 在package.xml中添加build_depend和exec_depend
- 在CMakeList.txt中,添加有关.msg文件的依赖包和说明

注: 如果是使用 Roboware,使用"右键->add Msg Folder"或"右键->add Msg File"将会自动修改package.xml和CMakeList.txt文件

a.自定义.msg文件

```
#在包中创建msg文件夹,专门用来存放.msg文件,并开始编辑Hello.msg文件
$ mkdir ~/my_ros/topic_test/msg
$ gedit ~/my_ros/topic_test/msg/Hello.msg
```

在编辑器中输入(类型+属性名)

```
String message
int32 index
```

b.修改 package.xml

为了能够使ros生成我们需要的python/c++的消息代码,我们需要在功能包中添加message_generation依赖,在package.xml文件中添加如下两行代码:

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

c.修改 CMakeList.txt

(1)在find_package()中添加message_generation,作用是让catkin在编译时链接message_generation功能包,修改后为:

```
find_package(catkin REQUIRED COMPONENTS
  message_generation
  rospy
  std_msgs
)
```

(2)在add_message_files()中添加Hello.msg. 作用是向catkin指明需要用到的.msg文件包含Hello.msg

```
add_message_files(FILES
  Hello.msg
)
```

(3)去掉generate_messages()的注释

(4)在catkin_package()中添加message_runtime

```
catkin_package(
  CATKIN_DEPENDS
  message_runtime
)
```

d.修改消息发送者文件 (Publisher.py)

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from topic_test.msg import Hello

def talker():
```

```

rospy.init_node('pyTalker', anonymous=True)
pub = rospy.Publisher('test_chat', Hello, queue_size=1)
rate = rospy.Rate(10)
count = 0
while not rospy.is_shutdown():

    greeting = Hello()
    greeting.index = count
    greeting.message = 'Happy new year!'
    pub.publish(greeting)
    rospy.loginfo('pyTalker: send msg = %s', greeting.message)
    rospy.loginfo('pyTalker: send index = %d', greeting.index)
    count = count+1
    rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass

```

e.修改消息接收者文件 (Subscriber.py)

```

#!/usr/bin/env python
# license removed for brevity
import rospy
from topic_test.msg import Hello

def callback(message):
    rospy.loginfo('pyListener: msg = %s' % message.message)
    rospy.loginfo('pyListener: index = %d' % message.index)

def listener():
    rospy.init_node('pyListener', anonymous=True)
    rospy.Subscriber('test_chat', Hello, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()

```

f.运行程序

按照上面的操作,先到工作空间目录下, catkin_make 一下,然后再输入命令运行

运行结果为:

```

giantjun@giantjun-TM1701: ~
File Edit View Search Terminal Help
[INFO] [1580375029.708902]: pyTalker: send msg = Happy new year!
[INFO] [1580375029.714026]: pyTalker: send index = 277
[INFO] [1580375029.808751]: pyTalker: send msg = Happy new year!
[INFO] [1580375029.813341]: pyTalker: send index = 278
[INFO] [1580375029.908830]: pyTalker: send msg = Happy new year!
[INFO] [1580375029.915222]: pyTalker: send index = 279
[INFO] [1580375030.008516]: pyTalker: send msg = Happy new year!
[INFO] [1580375030.014078]: pyTalker: send index = 280
[INFO] [1580375030.108847]: pyTalker: send msg = Happy new year!
[INFO] [1580375030.113572]: pyTalker: send index = 281

giantjun@giantjun-TM1701: ~
File Edit View Search Terminal Help
[INFO] [1580375029.515095]: pyListener: index = 275
[INFO] [1580375029.610306]: pyListener: msg = Happy new year!
[INFO] [1580375029.615133]: pyListener: index = 276
[INFO] [1580375029.709332]: pyListener: msg = Happy new year!
[INFO] [1580375029.714361]: pyListener: index = 277
[INFO] [1580375029.809316]: pyListener: msg = Happy new year!
[INFO] [1580375029.814596]: pyListener: index = 278
[INFO] [1580375029.909000]: pyListener: msg = Happy new year!
[INFO] [1580375029.913357]: pyListener: index = 279
[INFO] [1580375030.009342]: pyListener: msg = Happy new year!
[INFO] [1580375030.013778]: pyListener: index = 280
[INFO] [1580375030.109511]: pyListener: msg = Happy new year!
[INFO] [1580375030.114899]: pyListener: index = 281

```

2.2 service 通信方式(自定义)

这里的小demo实现的功能是由 **客户端(client)** 向 **服务器端(server)** 发送一条消息,消息里包含两个参数 a 和 b, **服务器端(server)** 收到这两个参数后,将他们相加后返回.

由于已经有了之前的基础,这里就不再展示完整操作了,只展示使用的核心部分,比较基础的读者可以自行查看 2.1 中的例子.

完整的基本步骤如下:

- 创建 service_test 功能包(与上例 2.1 一样)
- 创建自定义的.srv文件
- 修改 CMakeList.txt 和 package.xml
- 新建script文件夹,并编写 server.py 和 client.py
- 编译并运行

2.2.1 自定义.srv文件

在srv文件夹中,前两行为要传送的参数,"---"后的一行为返回的参数

```

int64 a
int64 b

```

```
---  
int64 result
```

2.2.2 编写服务器端(server)部分代码

如下为服务器端(server.py)部分代码

```
#!/usr/bin/env python  
import rospy  
from service_test.srv import Sum  
  
def handle(req):  
    rospy.loginfo("Successful recieved number %s and %s" % (req.a, req.b))  
    rospy.loginfo("Sending result: %s" % (req.a+req.b))  
    # 有些资料和官方文档个上写, 这里应该return的是"SumResponse(req.a+req.b)", 但实际  
    # 运行时不知道为什么出错, 改为直接返回结果后程序正常运行。  
    return req.a+req.b  
  
def server():  
    rospy.init_node("sum_server")  
    # 向master节点注册为service,  
    # 这个service的名字为"pyServer_Client"  
    # 传送消息类为Sum  
    # 处理函数的函数名为handle  
    rospy.Service("pyServer_Client", Sum, handle)  
    rospy.loginfo("Ready to recieve and handle request:")  
    # 消息回调处理函数, 将阻塞本线程  
    rospy.spin()  
  
if __name__ == "__main__":  
    server()
```

2.2.3 编写客户端(client)部分代码

如下为客户端(client.py)部分代码

```
#!/usr/bin/env python  
import sys  
import rospy  
from service_test.srv import Sum  
  
def client(x, y):  
    rospy.init_node("pyClient")  
    # 等待名为"pyServer_Client"的service空闲, 当其空闲/可用时程序继续执行  
    rospy.wait_for_service("pyServer_Client")  
    try:  
        # 向master节点注册为客户端,  
        # 申请链接的service名为"pyServer_Client",  
        # 消息类为Sum
```

```

    pyServer_Client = rospy.ServiceProxy("pyServer_Client", Sum)
    # 向服务器发送请求, 该请求包含2个参数
    resp = pyServer_Client.call(x,y)
    # 返回得到Sum.srv里result字段的加法运算结果
    return resp.result
except rospy.ServiceException, e:
    rospy.logwarn("Service call failed ! %s! " % e)

if __name__ == "__main__":
    if len(sys.argv) == 3:
        # 获取输入的两个参数
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        rospy.logwarn("The parameters inputed count is not right! ")
        sys.exit(1)
    rospy.loginfo("pyClient : Requesting %s + %s" % (x,y))
    rospy.loginfo("pyClient : The Result is %s" % client(x,y))

```

2.2.4 修改CMakeList.txt文件

修改的地方与2.1.4 c 中修改CMakeList.txt文件大致相同,只是将2.1.4 c(2) 中需要修改"add_message_file"的部分,改为修改"add_service_file"

[点这里跳转到2.1.4 c](#)

由于此次使用的是.srv文件,而不是.msg文件,所以catkin应该处理的是我们自定义的.srv文件.

我们需要将"add_message_file"部分注释掉,修改"add_service_file"部分为:

```

add_service_files(FILES
  Sum.srv
)

```

2.2.5 修改package.xml文件

这部分与2.1.4 (topic自定义msg的例子)中修改package.xml的一样

[点这里跳转 2.1.4 b](#)

2.2.6 运行service_test程序

运行需要的命令与前面提到的相同

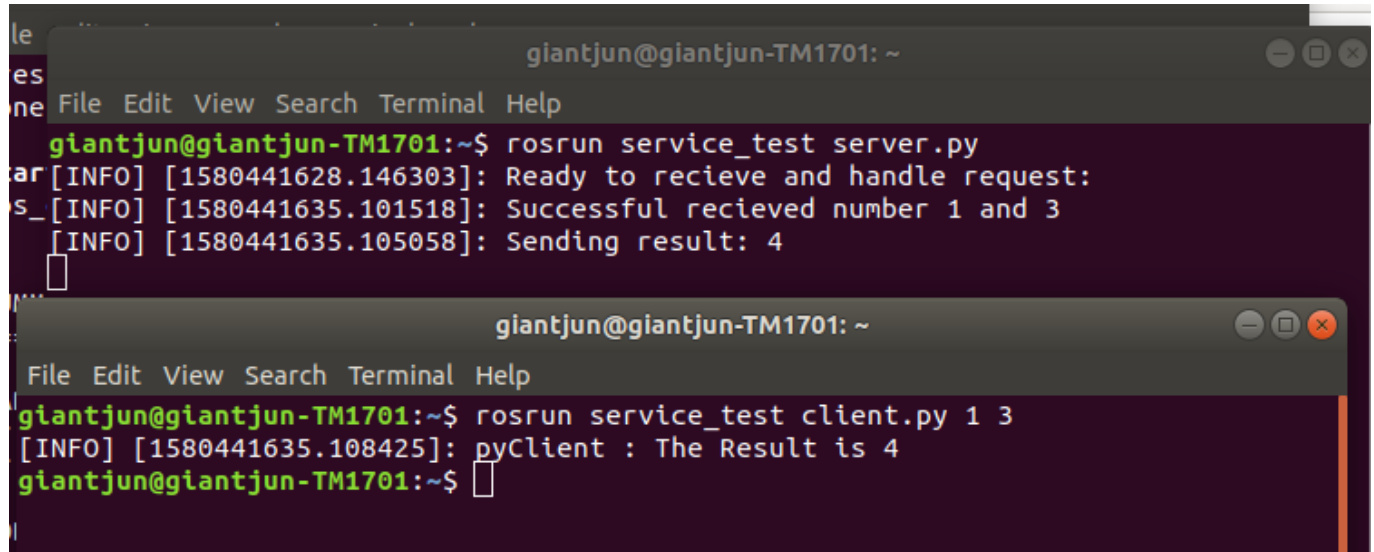
```

$ cd ~/my_ros/src/service_test/script
$ chmod +x server.py
$ chmod +x client.py
$ roscore
$ rosrn service_test server.py

```

```
#这里传入了连个参数1和3  
$ rosrn service_test client.py 1 3
```

程序运行情况为:

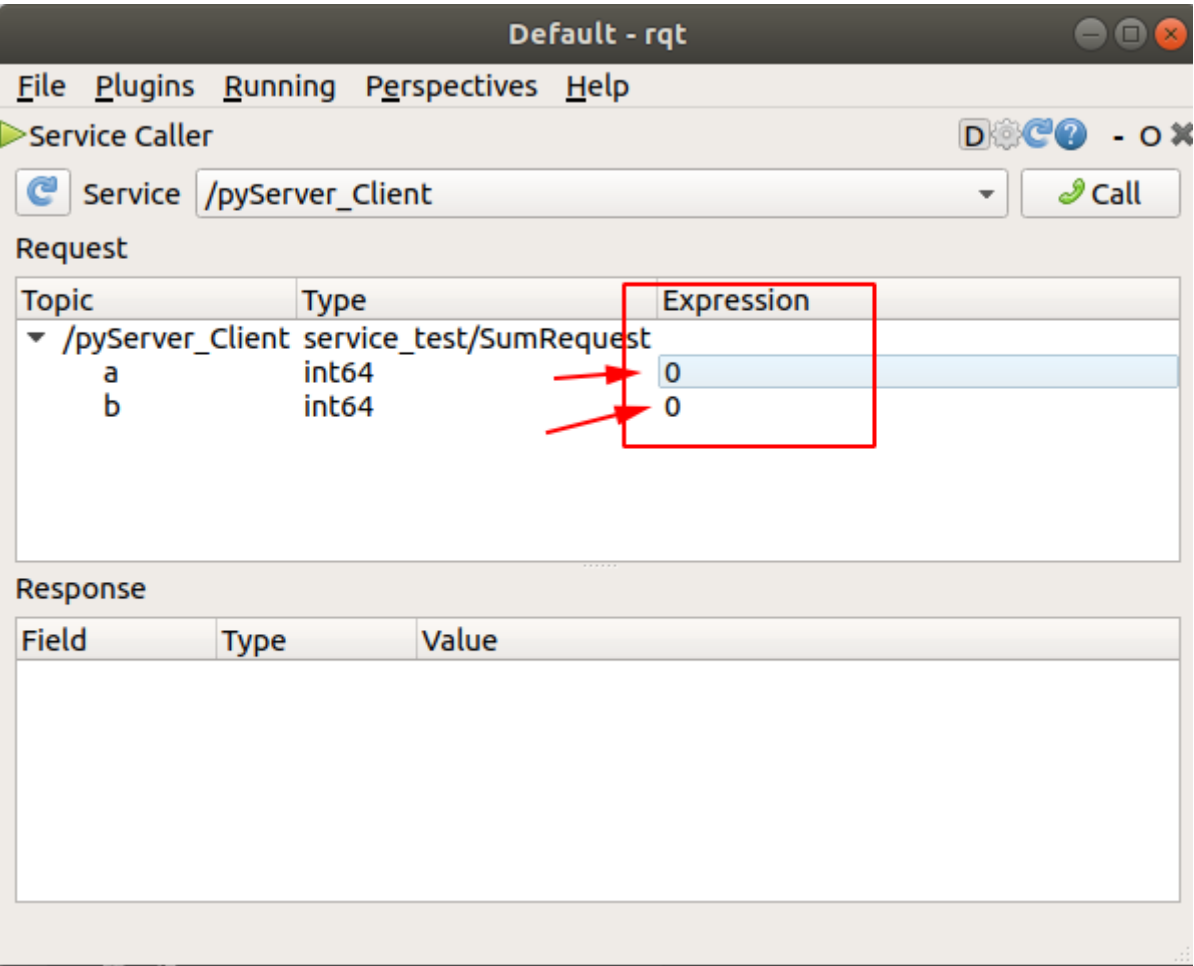


```
giantjun@giantjun-TM1701: ~  
File Edit View Search Terminal Help  
giantjun@giantjun-TM1701:~$ rosrn service_test server.py  
[INFO] [1580441628.146303]: Ready to recieve and handle request:  
s_[INFO] [1580441635.101518]: Successful recieved number 1 and 3  
[INFO] [1580441635.105058]: Sending result: 4  
█  
giantjun@giantjun-TM1701: ~  
File Edit View Search Terminal Help  
giantjun@giantjun-TM1701:~$ rosrn service_test client.py 1 3  
[INFO] [1580441635.108425]: pyClient : The Result is 4  
giantjun@giantjun-TM1701:~$ █
```

当然,运行server.py节点后,可以通过rqt的图形界面工具发送处理请求,这就不需要客户端(client.py)了.在新终端中输入

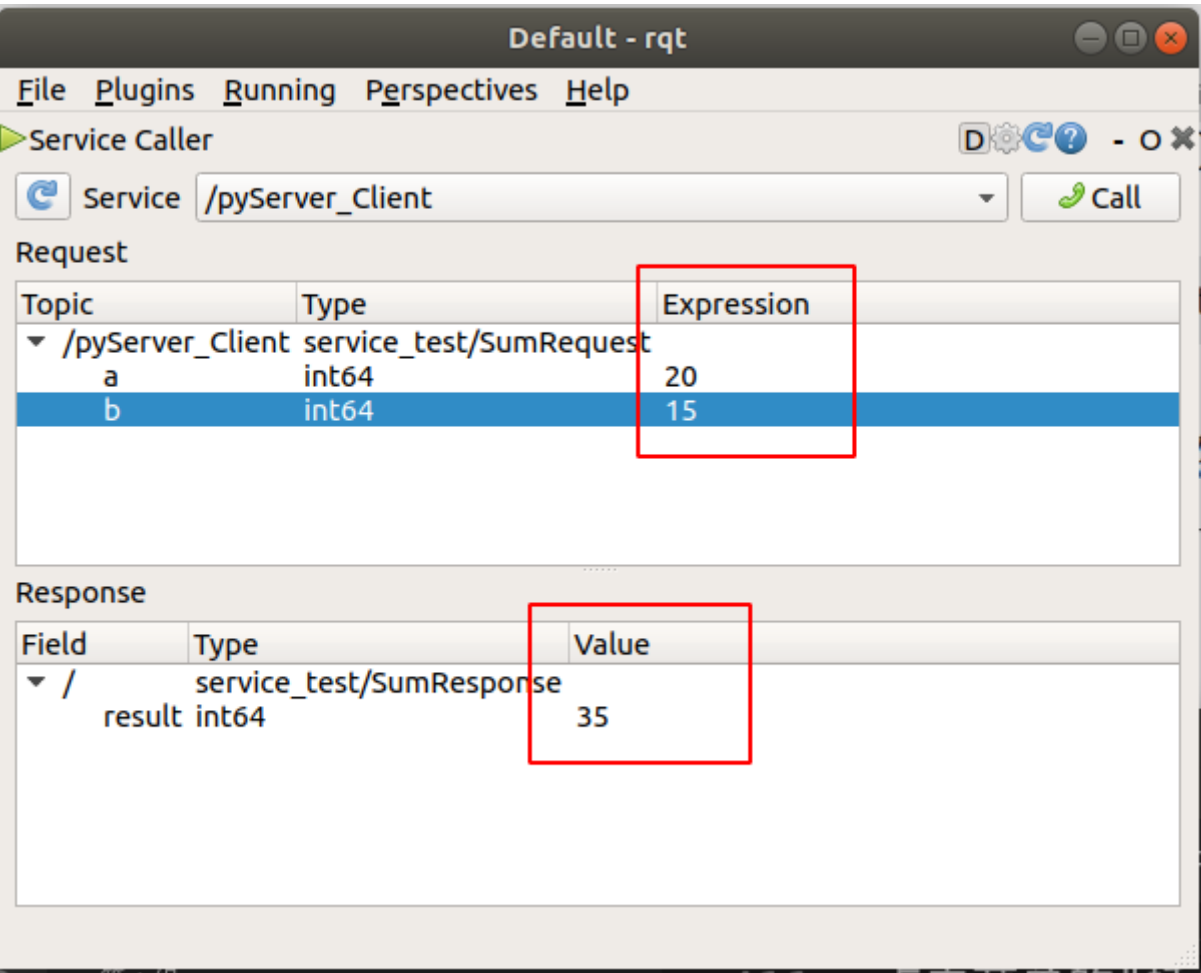
```
$ rqt
```

点击菜单的"插件[Plugins]->[Service]->[Service Call]"就可以得到下面的界面



修改如上图

箭头指向的地方为自己想传入的参数,点击"Call"可以得到服务器端返回的结果



2.3 action通信方式(不完整)

这里实现的例子是输入3个数,计算3个数的乘积作为最终结果,中间反馈结果为第1个数和第2个数的成绩(中间结果)

完整的基本步骤如下:

- 创建 action_test 功能包
- 创建自定义的.action文件
- 修改 CMakeList.txt 和 package.xml
- 新建script文件夹,并编写 action_server.py 和 action_client.py
- 编译并运行

action_server部分可参考:

http://wiki.ros.org/actionlib_tutorials/Tutorials/Writing%20a%20Simple%20Action%20Server%20using%20the%20Execute%20Callback%20%28Python%29

action_client部分可参考:

http://wiki.ros.org/actionlib_tutorials/Tutorials/Writing%20a%20Simple%20Action%20Client%20%28Python%29

2.3.1 创建 action_test功能包

```
$ catkin_create_pkg action_test rospy actionlib actionlib_msgs std_msgs  
message_generation
```

2.3.2 创建自定义.action文件

在 action_test 目录下创建 action 文件夹后,新建 multiply.action 文件,输入如下内容:

```
#a,b为定义的两个输入参数(goal definition)  
int32 a  
int32 b  
---  
#final_result为定义的最终结果(result definition)  
int32 final_result  
---  
#in_result为定义的中间反馈结果(feedback)  
int32 in_result
```

实际上动作(.action)可以包含5中基本消息:目标(goal),结果(result),反馈(feedback),取消(cancel),状态(status). 本例中未涉及的取消(cancel)消息使用actionlib_msgs/GoalID,作用是在动作运行时取消客户端和单独节点上的动作的执行. 状态(status)消息可以根据状态转换(如PENDING,ACTIVE,PREEMPTED和SUCCEEDED)检查当前动作的状态

2.3.3 修改 CMakeList.txt 和 package.xml 文件

package.xml中的内容实际上在指定依赖,创建功能包的时候,已经修改好了,所以不需要修改

CMakeList.txt的内容,如果是使用Roboware创建action Folder,action File,那么这一部分也不需要修改,因为已经修改好了.要修改的地方主要是如下所示的地方:

```
add_action_files(FILES
  multiply.action
)
```

2.3.4 编写action_server服务器端部分程序

后面再补充,可参考前面的链接

2.3.5 编写action_client客户端部分程序

后面再补充,可参考前面的链接

2.3.6 编译并运行程序

后面再补充,可参考前面的链接