创建 ROS 程序包

一个程序包（package）由什么组成?

一个程序包必须符合以下要求:

- 该程序包必须包含 package.xml 文件
  - 这个 package.xml 文件提供有关程序包的元信息。
- 程序包必须包含一个 CMakeLists.txt 文件。
- 每个目录下只能有一个程序包。
  - 这意味着在同一个目录下不能有嵌套的或者多个程序包存在。

最简单的程序包也许看起来就像这样:

```
my_package/
  CMakeLists.txt
  package.xml
```

创建一个程序包

本部分将演示如何使用 catkin_create_pkg 命令来创建一个新的 catkin 程序包以及创建之后都能做些什么。

首先切换到创建的 catkin 工作空间中的 src 目录下:

```
# You should have created this in the Creating a Workspace Tutorial
$ cd ~/catkin_ws/src
```

现在使用 catkin_create_pkg 命令来创建一个名为'beginner_tutorials'的新程序包，这个程序包依赖于 std_msgs、roscpp 和 rospy:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

这将会创建一个名为 beginner_tutorials 的文件夹，这个文件夹里面包含一个 package.xml 文件和一个 CMakeLists.txt 文件，这两个文件都已经自动包含了部分你在执行 catkin_create_pkg 命令时提供的信息。

catkin_create_pkg 命令会要求你输入 package_name，如果有需要你还可以在后面添加一些需要依赖的其它程序包:

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

这些依赖包随后保存在 **package.xml** 文件中，可以在 beginner_tutorials 文件夹路径下通过以下命令查看 **package.xml** 文件。

```
$ cat package.xml
```

```
- <package>
- ...
-   <buildtool_depend>catkin</buildtool_depend>
-   <build_depend>roscpp</build_depend>
-   <build_depend>rospy</build_depend>
-   <build_depend>std_msgs</build_depend>
- ...
- </package>
```

编译 ROS 程序包

编译程序包

一旦安装了所需的系统依赖项，我们就可以开始编译刚才创建的程序包了。

记得事先 source 你的环境配置(setup)文件，在 Ubuntu 中的操作指令如下：

```
$ source /opt/ros/noetic/setup.bash
```

使用 catkin_make

catkin_make 是一个命令行工具，用于程序包的编译。

使用方法:

```
#  在 catkin 工作空间下

$ catkin_make [make_targets] [-DCMAKE_VARIABLES=...]
```

开始编译你的程序包

按照之前的创建一个 ROS 程序包教程，你应该已经创建好了一个 catkin 工作空间 和一个名为 beginner_tutorials 的 catkin 程序包。现在切换到 catkin workspace 并查看 src 文件夹：

```
$ cd ~/catkin_ws/

$ ls src
```

- beginner_tutorials/ CMakeLists.txt@

你可以看到一个名为 beginner_tutorials 的文件夹，这就是你在之前的 catkin_create_pkg 教程里创建的。现在我们可以使用 catkin_make 来编译它了：

```
$ catkin_make
```

你可以看到很多输出的信息。

写一个简单的消息发布器和订阅器 (Python)

写一个发布器节点

“节点” (Node) 是指 ROS 网络中可执行文件。接下来，我们将会创建一个发布器节点 (“talker”)，它将不断的在 ROS 网络中广播消息。

确保你在调用 catkin_make 后，已经 source 了 catkin 工作空间下的 setup.bash 文件，以更新环境配置信息：

```
# In your catkin workspace

$ cd ~/catkin_ws

$ source ./devel/setup.bash
```

切换到之前创建的 beginner_tutorials 功能包路径下：

```
$ roscd beginner_tutorials
```

源代码

先创建一个“scripts”文件夹用来存储我们的 scripts 脚本：

```
$ mkdir scripts

$ cd scripts
```

然后将示例脚本 talker.py 下载到新脚本目录中，并使其可执行：

```
$ wget https://github.com/cavayangtao/rmtt_ros/blob/main/learning_tf2/src/talker.py
```

或者：

```
$ wget https://gitee.com/cavayangtao/rmtt_ros/blob/main/learning_tf2/src/talker.py
```

获取脚本 talker.py 可执行权限：

```
$ chmod +x talker.py
```

你可以通过$ rosed beginner_tutorials talker.py 查看编辑文件，或者直接看下面：

```
1 #!/usr/bin/env python3
2 # license removed for brevity
3 import rospy
4 from std_msgs.msg import String
5
6 def talker():
7         pub = rospy.Publisher('chatter', String, queue_size=10)
8         rospy.init_node('talker', anonymous=True)
9         rate = rospy.Rate(10) # 10hz
10        while not rospy.is_shutdown():
11                hello_str = "hello world %s" % rospy.get_time()
12                rospy.loginfo(hello_str)
13                pub.publish(hello_str)
14                rate.sleep()
15
16 if __name__ == '__main__':
17        try:
18                talker()
19        except rospy.ROSInterruptException:
20                pass
```

代码说明

现在我们分段解释代码

```
1 #!/usr/bin/env python3
```

每个 python ROS 节点的顶部都有这个声明。第一行确保脚本作为 python 脚本执行。

```
3 import rospy
```

```
4 from std_msgs.msg import String
```

如果正在编写 ROS 节点，则需要导入 rospy。std_msgs.msg 的导入 std_msgs/String 消息类型（一个简单的字符串容器）进行发布。

```
7       pub = rospy.Publisher('chatter', String, queue_size=10)
8       rospy.init_node('talker', anonymous=True)
```

pub =rospy.Publisher("chatter"，String，queue_size=10)声明节点正在使用消息类型字符串发布到话题。这里的字符串实际上是类 std_msgs.msg.String。queue_size 参数是为了如果任何订阅者接收消息不够快，则限制排队消息的数量。

下一行 rospy.init_node(name,…)非常重要，因为它告诉 rospy 你的节点的名称在 rospy 获得此信息之前，它无法开始与 ROS 主节点（master）进行通信。在这种情况下，你的节点将使用名称 talker。注意：名称必须是 base name，即不能包含任何斜杠"/"。

anonymous = True 通过向名称末尾添加随机数来确保节点具有唯一的名称。

```
9       rate = rospy.Rate(10) # 10hz
```

这一行创建一个 Rate 对象 rate。借助 sleep()，它提供了一种以所需速度循环的方便方法。它的参数为 10，我们应该期望每秒循环 10 次(只要我们的处理时间不超过 1/10 秒！)

```
10      while not rospy.is_shutdown():
11          hello_str = "hello world %s" % rospy.get_time()
12          rospy.loginfo(hello_str)
13          pub.publish(hello_str)
14          rate.sleep()
```

此循环是一个相当标准的 rospy 构造：检查 rospy.is_shutdown() 标志，然后执行工作。要检查程序是否应该退出(例如，如果存在 ctrl-c 或其他情况)，您必须检查 is_shutdown()。在本例中， 关键是对 pub.publish(hello_str)的调用，它将字符串发布到我们的 chatter 话题中。循环调用 rate.sleep()，它的睡眠时间刚好足以通过循环保持所需的速率。

这个循环还调用 rospy.loginfo(str)，它执行三重任务：消息被打印到屏幕，它被写入节点的日志文件，并被写入 rosout。

你可能会对最后一点感到好奇：

```
17  try:
18      talker()
19  except rospy.ROSInterruptException:
20      pass
```

除了标准 Python __main__检查之外，它还捕获 rospy.ROSInterruptException 异常，当 ctrl-c 按下或节点被关闭时，rospy.sleep()和 rospy.Rate.sleep() 会引发该异常。

写一个订阅器节点

源代码

将 listener.py 文件下载到你的 scripts 目录中：

```
$ roscd beginner_tutorials/scripts/

$ wget https://github.com/cavayangtao/rmtt_ros/blob/main/learning_tf2/src/listener.py
```

或者：

```
$ wget https://gitee.com/cavayangtao/rmtt_ros/blob/main/learning_tf2/src/listener.py
```

同样我们在该 python 文件目录下需要开启权限：

```
$ chmod +x listener.py
```

下载的代码如下：

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

def callback(data):
        rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

        # In ROS, nodes are uniquely named. If two nodes with the same
        # name are launched, the previous one is kicked off. The
        # anonymous=True flag means that rospy will choose a unique
        # name for our 'listener' node so that multiple listeners can
        # run simultaneously.
        rospy.init_node('listener', anonymous=True)

        rospy.Subscriber("chatter", String, callback)

        # spin() simply keeps python from exiting until this node is stopped
        rospy.spin()

if __name__ == '__main__':
        listener()
```

代码说明

listener.py 的代码类似于 talker.py，只不过我们引入了一种新的基于回调的机制来订阅消息。

```
15      rospy.init_node('listener', anonymous=True)
16
17      rospy.Subscriber("chatter", String, callback)
18
19      # spin() simply keeps python from exiting until this node is stopped
20      rospy.spin()
```

这声明你的节点订阅了类型为 std_msgs.string 的 chatter 主题。当接收到新消息时，将以消息作为第一个参数调用回调。

我们也稍微改变了对 rospy.init_node()的调用。我们添加了 anonymous=True 关键字参数。ROS 要求每个节点具有唯一的名称。如果具有相同名称的节点出现，则会出现先前的节点。这使得故障节点能够容易地从网络中被踢开。 anonymous=True 标志告诉 rospy 为节点生成唯一的名称，这样就可以轻松地运行多个 listener.py 节点。

最后一个附加项 rospy.spin()只会阻止节点退出，直到节点被关闭为止。

编译节点

我们使用 CMake 作为我们的编译系统。

改变目录到 catkin workspace 然后运行 catkin_make 指令：

```
$ cd ~/catkin_ws

$ catkin_make
```

测试消息发布器和订阅器

启动发布器

确保 roscore 可用，并运行：

```
$ roscore
```

确保你在调用 catkin_make 后，在运行你自己的程序前，已经 source 了 catkin 工作空间下的 setup.bash 文件：

```
# In your catkin workspace

$ cd ~/catkin_ws
```

```
$ source ./devel/setup.bash
```

在上一个教程中我们制作了一个叫做"talker"的消息发布器，让我们运行它：

```
$ rosrun beginner_tutorials talker          (C++)

$ rosrun beginner_tutorials talker.py      (Python)
```

你将看到如下的输出信息:

- [INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
- [INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
- [INFO] [WallTime: 1314931833.778937] hello world 1314931833.78
- [INFO] [WallTime: 1314931834.782059] hello world 1314931834.78
- [INFO] [WallTime: 1314931835.784853] hello world 1314931835.78
- [INFO] [WallTime: 1314931836.788106] hello world 1314931836.79

发布器节点已经启动运行。现在需要一个订阅器节点来接受发布的消息。

启动订阅器

上一教程，我们编写了一个名为"listener"的订阅器节点。现在运行它：

```
$ rosrun beginner_tutorials listener       (C++)

$ rosrun beginner_tutorials listener.py    (Python)
```

你将会看到如下的输出信息:

- [INFO] [WallTime: 1314931969.258941] /listener_17657_13149319687951 heard hello world 1314931969.26
- [INFO] [WallTime: 1314931970.262246] /listener_17657_13149319687951 heard hello world 1314931970.26
- [INFO] [WallTime: 1314931971.266348] /listener_17657_13149319687951 heard hello world 1314931971.26
- [INFO] [WallTime: 1314931972.270429] /listener_17657_13149319687951 heard hello world 1314931972.27
- [INFO] [WallTime: 1314931973.274382] /listener_17657_13149319687951 heard hello world 1314931973.27
- [INFO] [WallTime: 1314931974.277694] /listener_17657_13149319687951 heard hello world 1314931974.28

- [INFO] [WallTime: 1314931975.283708] /listener_17657_13149319687951 heard hello world 1314931975.28

查看当前话题列表：

```
$ rostopic list
```

会看到 /chatter 话题，我们看一下该话题的信息：

```
$ rostopic info /chatter
```

- Type: std_msgs/String
- Publishers:
- * /talker_8665_1601020051089 (http://ros2go:42941/)
- Subscribers:
- * /listener_8903_1601020068142 (http://ros2go:46553/)

可以看到该话题对应的消息类型，以及它的发送者和订阅者。

录制并播放 rosbag

安装摄像头 ROS 驱动

下载并编译摄像头驱动

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/cavayangtao/usb_cam.git
```

或者：

```
git clone https://gitee.com/cavayangtao/usb_cam.git
```

回到工作空间目录进行编译：

```
$ cd ~/catkin_ws
$ catkin_make
```

启动摄像头

```
$ roslaunch usb_cam usb_cam-test.launch
```

rosrun 只能运行一个 node，roslaunch 可以同时运行多个 nodes。roslaunch 工具是 ros 中 python 实现的程序启动工具，可以通过读取启动文件（launch file）中的参数配置、属性配置等，同时启动节点管理器（master）和多个节点，在启动任何一个节点前，roslaunch 将会确定 roscore 节点（节点管理器） 是否已经在运行，如果没有，自动启动它。

利用 rviz 进行查看

我们首先看一下摄像头发布消息的主题:

```
$ rostopic list
```

- /rosout
- /rosout_agg
- /usb_cam/camera_info
- /usb_cam/image_raw

启动 ROS 数据查看器，并增加需要查看的主题。

```
$ rviz
```

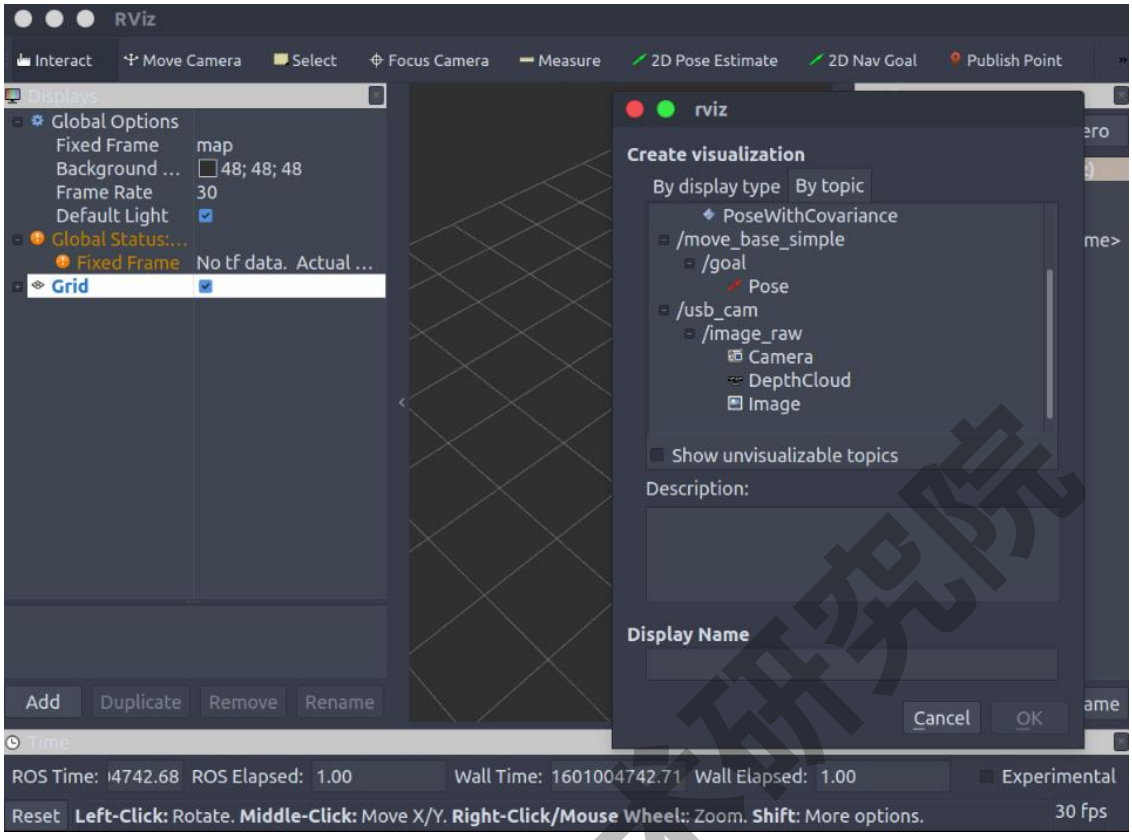

图 1：向 rivz 可视化工具中添加话题

录制 rosbag

```
$ rosbag record /usb_cam/image_raw
```

按 ctrl-c 可以停止当前的录制。关闭所有终端，然后重新开启一个 roscore。然后播放我们刚才保存的主题，如：

```
$ rosbag play -l 2020-09-25-11-35-47.bag
```

这里的 -l 是循环播放参数。然后可以类似的在 rviz 中查看一下。

查看计算图：

```
$ rqt_graph
```

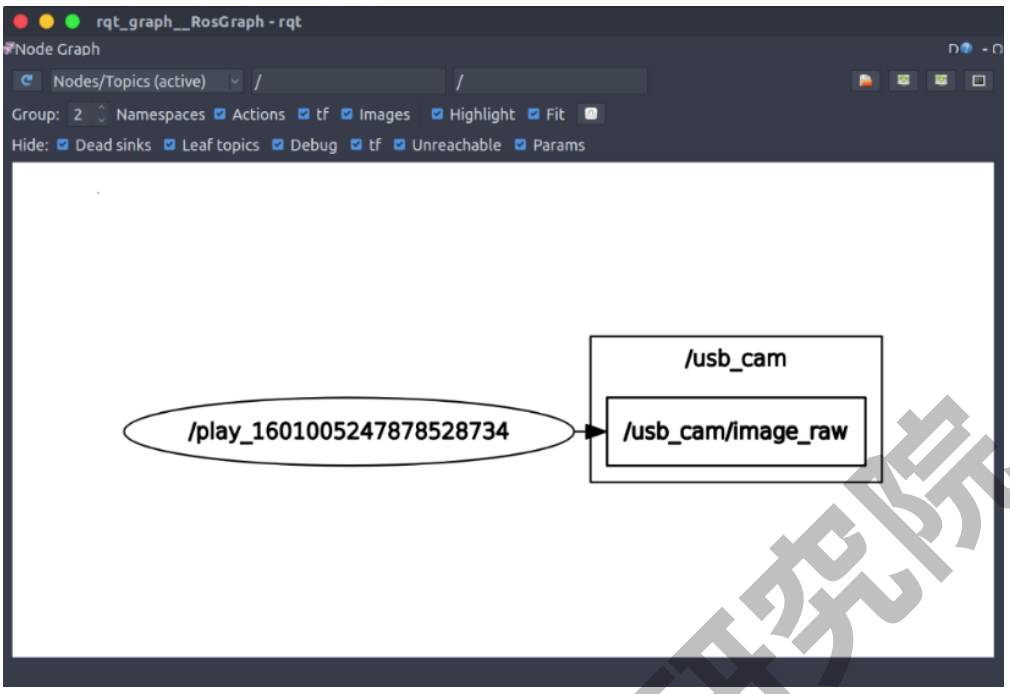

图 2：usb 摄像头运行时的节点话题关系图

调用一只新的小乌龟

我们首先关闭所有终端，并重新开启小乌龟模拟器：

```
$ roscore

$ rosrun turtlesim turtlesim_node
```

利用服务调用一只新的小乌龟

可以使用 rosservice call 调用服务：

```
$ rosservice call [service] [args]
```

spawn 服务允许我们根据给定的坐标和角度产生另一个乌龟，并且可以给这个新产生的乌龟起一个名字，也可以不起名字。如：

```
$ rosservice call spawn 2 2 0.1 “turtle2”
```

在这条命令执行完成后，就会出现另一只乌龟。

ROS 中的服务是基于请求和响应机制的，在上面的例子中，我们通过终端发送请求，节点接收后，做出响应。

下面来看一下服务的参数：

```
$ rosservice type spawn
```

返回结果：

- turtlesim/Spawn

查看这个服务类型的具体细节：

```
$ rossrv show turtlesim/Spawn
```

返回：

- float32 x
- float32 y
- float32 theta
- string name
- ---
- string name

编写一个 ROS 包，让两只小乌龟转圈

现在使用 catkin_create_pkg 命令来创建一个名为 turtle_control 的新程序包，这个程序包依赖于 std_msgs、geometry_msgs 和 rospy：

```
$ cd ~/catkin_ws/src/

$ catkin_create_pkg turtle_control rospy std_msgs geometry_msgs
```

记得 source 一下 catkin 工作空间下的 setup.bash 文件，以配置工作空间，刷新环境。然后进入创建包的 src 文件夹下，创建一个名为 walk.py 的 python 脚本文件：

```
$ source ./devel/setup.bash

$ roscd turtle_control/src/

$ gedit walk.py
```

我们将编写一个发布器节点，向两只小乌龟发送指令，使他们能同时转圈。将以下代码写入脚本文件，也可以复制之前的 talker.py 中的代码进行修改：

```
#!/usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist
```

```
def turtle_publisher():
        rospy.init_node('turtle_publisher', anonymous = True)
         pub1 = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
         pub2 = rospy.Publisher('/turtle2/cmd_vel', Twist, queue_size=10)
         rate = rospy.Rate(1)

         while not rospy.is_shutdown():
                msg1 = Twist()
                msg1.linear.x = 0.5
                msg1.angular.z = 0.2
                pub1.publish(msg1)

                msg2 = Twist()
                msg2.linear.x = 0.3
                msg2.angular.z = 0.1
                pub2.publish(msg2)

                rospy.loginfo('publish turtle message command')
                rate.sleep()
if __name__ == '__main__':
        try:
                turtle_publisher()
        except rospy.ROSInterruptException:
                pass
```

保存完编写的 python 脚本后，记得开放该脚本的可执行权限：

```
$ chmod +x walk.py
```

退回到 catkin_ws 文件夹下进行编译，然后运行一下刚刚编写的节点：

```
$ cd ~/catkin_ws

$ catkin_make

$ source ./devel/setup.bash

$ rosrun turtle_control walk.py
```

可以看到如下的结果：

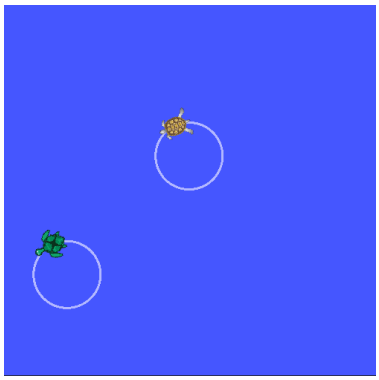

图 3：两只小乌龟同时运动

录制 rosbag，并播放

我们可以将当前的话题录制成一个 rosbag 文件：

```
$ rosbag record -a -0 cmd_vel_record
```

-a 是指录制所有的话题，-0 用来改变所录制 rosbag 的名称，我们命名为 cmd_vel_record。

现在关闭正在执行的发布器，回到 catkin_ws 文件夹下，播放一下刚刚录制的 rosbag：

```
$ rosbag play cmd_vel_record.bag
```

看看小乌龟是否开始运动。