第十二课 编写消息发布器和订阅器

1.编写发布器节点

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。

开始编写代码之前我们先创建一个新的 beginner_tutorials 包

```
cd ~/robot_ws/src/

catkin_create_pkg beginner_tutorials roscpp rospy std_msgs
```

在 beginner_tutorials package 路径下创建一个 src 文件夹：

```
mkdir -p ~/ robot _ws/src/beginner_tutorials/src
```

在 src 文件夹内创建新的 cpp 文件：

```
cd ~/ robot _ws/src/beginner_tutorials/src

touch talker.cpp
```

在 talker.cpp 文件内添加以下代码

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <iostream>

using namespace std;


int main(int argc, char **argv)
{

    ros::init(argc, argv, "talker");
    ros::NodeHandle n;
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

    ros::Rate loop_rate(10);

    while (ros::ok())
    {
        std_msgs::String msg;
        msg.data = "Hello Ros talker!";
        chatter_pub.publish(msg);

        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

代码解析：

```
1.  #include "ros/ros.h"
```

ros/ros.h 是一个实用的头文件，它引用了 ROS 系统中大部分常用的头文件。

```
2. #include "std_msgs/String.h"
```

这引用了 std_msgs/String 消息，存放在 std_msgs package 里，是自动生成的头文件。

```
3. ros::init(argc, argv, "talker");
```

初始化 ROS 。它允许 ROS 通过命令行进行名称重映射，也可以指定节点的名称（运行过程中），但是节点的名称必须唯一。

```
4. ros::NodeHandle n;
```

为这个进程的节点创建一个句柄。第一个创建的 NodeHandle 会为节点进行初始化，最后一个销毁的 NodeHandle 则会释放该节点所占用的所有资源。

```
5. ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
```

告诉 master 我们将要在 chatter（话题名） 上发布 std_msgs/String 消息类型的消息。这样 master 就会告诉所有订阅了 chatter 话题的节点，将要有数据发布。第二个参数是发布序列的大小。如果我们发布的消息的频率太高，缓冲区中的消息在大于 1000 个的时候就会开始丢弃先前发布的消息。

NodeHandle::advertise() 返回一个 ros::Publisher 对象，它有两个作用：

1) 它有一个 publish() 成员函数可以让你在 topic 上发布消息；

2) 如果消息类型不对，它会拒绝发布。

```
6. ros::Rate loop_rate(10);
```

ros::Rate 对象可以允许你指定自循环的频率。它会追踪记录自上一次调用 Rate::sleep() 后时间的流逝，并休眠直到一个频率周期的时间。

```
7. chatter_pub.publish(msg);
```

我们向所有订阅 chatter 话题的节点发送消息。

```
8. ros::spinOnce();
```

在这个例子中并不是一定要调用 ros::spinOnce()，因为我们不接受回调。然而，如果你的程序里包含其他回调函数，最好在这里加上 ros::spinOnce() 这一语句，否则你的回调函数就永远也不会被调用了。

```
9. loop_rate.sleep();
```

这条语句是调用 ros::Rate 对象来休眠一段时间以使得发布频率为 10Hz。

2. 编写订阅器节点

在 beginner_tutorials 目录下创建 src/listener.cpp 文件，并添加如下代码：

```
cd ~/ robot _ws/src/beginner_tutorials/src

touch listener.cpp
```

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"


void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{

    ros::init(argc, argv, "listener");
    ros::NodeHandle n;

    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

    ros::spin();
    return 0;
}
```

代码解析：

```
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

这是一个回调函数，当接收到 chatter 话题的时候就会被调用。消息是以 boost shared_ptr 指针的形式传输，这就意味着你可以存储它而又不需要复制数据。

```
ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
```

告诉 master 我们要订阅 chatter 话题上的消息。当有消息发布到这个话题时，ROS 就会调用 chatterCallback() 函数。第二个参数是队列大小，以防我们处理消息的速度不够快，当缓存达到 1000 条消息后，再有新的消息到来就将开始丢弃先前接收的消息。

NodeHandle::subscribe() 返回 ros::Subscriber 对象，你必须让它处于活动状态直到你不再想订阅该消息。当这个对象销毁时，它将自动退订 chatter 话题的消息。

有各种不同的 NodeHandle::subscribe() 函数，允许你指定类的成员函数，甚至是 Boost.Function 对象可以调用的任何数据类型。roscpp overview 提供了更为详尽的信息。

```
ros::spin();
```

ros::spin() 进入自循环，可以尽可能快的调用消息回调函数。如果没有消息到达，它不会占用很多 CPU，所以不用担心。一旦 ros::ok() 返回 false，ros::spin() 就会立刻跳出自循环。这有可能是 ros::shutdown() 被调用，或者是用户按下了 Ctrl-C，使得 master 告诉节点要终止运行。也有可能是节点被人为关闭的。

3. 编译和运行

在 CMakeLists.txt 文件末尾加入几条语句：

```
sudo vim ~/robot_ws/src/beginner_tutorials/CMakeLists.txt
```

```
include_directories(include ${catkin_INCLUDE_DIRS})

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

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

```
cd ~/robot_ws/

catkin_make

rospack profile
```

```
[ 96%] Linking CXX executable /home/huanyu/robot_ws/devel/lib/beginner_tutorials/talker
[ 96%] Built target talker
[100%] Linking CXX executable /home/huanyu/robot_ws/devel/lib/beginner_tutorials/listener
[100%] Built target listener
```

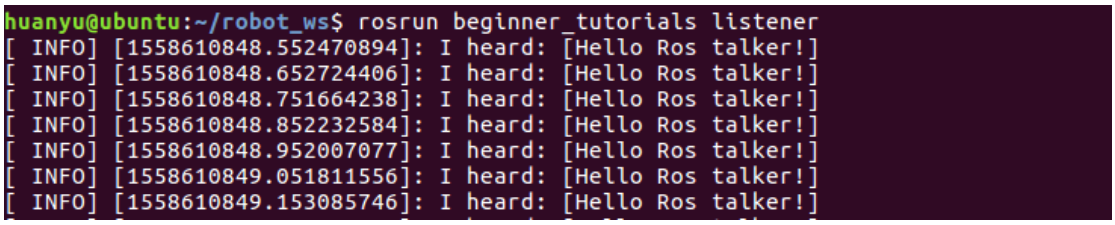
打开新的终端首先运行 roscore。再打开新的终端运行消息发布器节点：

```
rosrun beginner_tutorials talker
```

```
huanyu@ubuntu:~$ rosrun beginner_tutorials talker
```

在打开一个新的终端，运行消息订阅器节点：

```
rosrun beginner_tutorials listener
```



4. 总结

本节课程我们使用 c++编写了一个 ROS 发布器节点和一个订阅器节点，同时简单介绍了消息的发布和消息回调的编码方式。最后运行了两个节点，也成功看到订阅器订阅的内容。