编写简单的消息发布器和订阅器（C++）

AI 航 团队

1.编写节点发布器

『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。

切换到之前创建的 beginner_tutorials package 路径下：

```
$ cd ~/catkin_ws/src/beginner_tutorials
```

1.1 源代码

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

```
mkdir -p ~/catkin_ws/src/beginner_tutorials/src
```

这个文件夹将会用来放置 beginner_tutorials package 的所有源代码。

在 beginner_tutorials package 里创建 src/talker.cpp 文件，并将如下代码粘贴到文件内：

代码如下：

~~~~~~

```
1 #include "ros/ros.h"
2
3 /* 这引用了 std_msgs/String 消息, 它存放在 std_msgs package 里，是由 String.msg 文件自动生成的头文件 */
4 #include "std_msgs/String.h"
5
6 #include <sstream>
7
8 int main(int argc, char **argv)
9 {
10   /* 初始化 ROS 可以指定节点的名称——运行过程中，节点的名称必须唯一 */
11   ros::init(argc, argv, "talker");
12   /* 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。*/
13   ros::NodeHandle n;
14   /* 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。
15   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。 */

```
16  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter",1000);
17
18  /* 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。
19  在这个例子中，我们让它以 10Hz 的频率运行。*/
20  ros::Rate loop_rate(10);
21
22  int count  = 0;
23  /* roscpp 会默认生成一个 SIGINT 句柄，它负责处理 Ctrl-C 键盘操作——使得 ros::ok() 返回 false。
24  如果下列条件之一发生，ros::ok() 返回 false：
25  SIGINT 被触发 (Ctrl-C)
26  被另一同名节点踢出 ROS 网络
27  ros::shutdown() 被程序的另一部分调用
28  节点中的所有 ros::NodeHandles 都已经被销毁
29  一旦 ros::ok() 返回 false, 所有的 ROS 调用都会失效。*/
30  while(ros::ok())
31  {
32    /* 使用标准的 String 消息，它只有一个数据成员 "data" */
33    std_msgs::String msg;
34
35    std::stringstream  ss;
36    ss << "hello world" << count;
37    msg.data = ss.str();
38    /* ROS_INFO 和其他类似的函数可以用来代替 printf/cout 等函数*/
39    ROS_INFO("%s",msg.data.c_str());
```

```
40        /* 这里，我们向所有订阅 chatter 话题的节点发送消息 */
41        chatter_pub.publish(msg);
42        /* 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。 */
43        ros::spinOnce();
44        /* 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。 */
45        loop_rate.sleep();
46        ++count;
47    }
48    return 0;
49 }
50
```

1.2 代码说明

现在，我们来分段解释代码。

```
#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。

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

这引用了 std_msgs/String 消息, 它存放在 std_msgs package 里，是由 String.msg 文件自动生成的头文件。需要关于消息的定义，可以参考 msg 页面。

```
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——然而这并不是现在讨论的重点。在这里，我们也可以指定节点的名称——运行过程中，节点的名称必须唯一。

这里的名称必须是一个 base name ，也就是说，名称内不能包含 / 等符号。

```
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。

```
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。

```
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。

在这个例子中，我们让它以 10Hz 的频率运行。

```
int count = 0;
 while (ros::ok())
{
```

roscpp 会默认生成一个 SIGINT 句柄，它负责处理 Ctrl-C 键盘操作——使得 ros::ok() 返回 false。

如果下列条件之一发生，ros::ok() 返回 false： SIGINT 被触发 (Ctrl-C)

被另一同名节点踢出 ROS 网络

ros::shutdown() 被程序的另一部分调用

节点中的所有 ros::NodeHandles 都已经被销毁

一旦 ros::ok() 返回 false, 所有的 ROS 调用都会失效。

```
 std_msgs::String msg;

 std::stringstream ss;
 ss << "hello world " << count;
msg.data = ss.str();
```

我们使用一个由 msg file 文件产生的『消息自适应』类在 ROS 网络中广播消息。现在我们使用标准的 String 消息，它只有一个数据成员 "data"。当然，你也可以发布更复杂的消息类型。

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

这里，我们向所有订阅 chatter 话题的节点发送消息。

```
ROS_INFO("%s", msg.data.c_str());
```

ROS_INFO 和其他类似的函数可以用来代替 printf/cout 等函数。具体可以参考 rosconsole documentation，以获得更多信息。

```
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。

```
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。

对上边的内容进行一下总结：

初始化 ROS 系统

在 ROS 网络内广播我们将要在 chatter 话题上发布 std_msgs/String 类型的消息

以每秒 10 次的频率在 chatter 上发布消息

接下来我们要编写一个节点来接收这个消息。

2. 编写订阅器节点

在 beginner_tutorials package 目录下创建 src/listener.cpp 文件，并粘贴如下代码：

```
1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3
4 /* 这是一个回调函数，当接收到 chatter 话题的时候就会被调用 */
5 void chatterCallback(const std_msgs::String::ConstPtr& msg)
6 {
7    ROS_INFO("I heard: [%s]", msg->data.c_str());
8 }
9
10
```

```
11 int main(int argc, char **argv)

12 {

13     ros::init(argc, argv, "listener");

14

15     ros::NodeHandle n;

16

17     /* 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。*/

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

19     /* 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。 */

20     ros::spin();

21

22     return 0;

23 }

24
```

下面对相关代码进行解释

```
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. 编译节点

~~~~~~

```
cmake_minimum_required(VERSION 2.8.3)

project(beginner_tutorials)


## Find catkin and any catkin packages
```
~~~~~~

```
5 find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs genmsg)
6
7 ## Declare ROS messages and services
8 add_message_files(FILES Num.msg)
9 add_service_files(FILES AddTwoInts.srv)
10
11 ## Generate added messages and services
12 generate_messages(DEPENDENCIES std_msgs)
13
14 ## Declare a catkin package
15 catkin_package()
16
17 ## Build talker and listener
18 include_directories(include ${catkin_INCLUDE_DIRS})
19
20 add_executable(talker src/talker.cpp)
21 target_link_libraries(talker ${catkin_LIBRARIES})
22 add_dependencies(talker beginner_tutorials_generate_messages_cpp)
23
24 add_executable(listener src/listener.cpp)
25 target_link_libraries(listener ${catkin_LIBRARIES})
26 add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```

现在运行 catkin_make

```
catkin_make
```

```
rosrun beginner_tutorials talker
```

```
yt@yt-UNO-2483G-453AE:~/catkin_ws$ catkin_make
Base path: /home/yt/catkin_ws
Source space: /home/yt/catkin_ws/src
Build space: /home/yt/catkin_ws/build
Devel space: /home/yt/catkin_ws/devel
Install space: /home/yt/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/yt/catkin_ws/bui
ld"
####
-- Using CATKIN_DEVEL_PREFIX: /home/yt/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yt/catkin_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/yt/catkin_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yt/catkin_ws/build/test_results
-- Found gmock sources under '/usr/src/gmock': gmock will be built
```

```
yt@yt-UNO-2483G-453AE:~$ cd ~/catkin_ws
yt@yt-UNO-2483G-453AE:~/catkin_ws$ source ./devel/setup.bash
yt@yt-UNO-2483G-453AE:~/catkin_ws$ rosrun beginner_tutorials talker
文件 3067155.447136654]: hello world 0
[ INFO] [1588067155.546999653]: hello world 1
[ INFO] [1588067155.646988347]: hello world 2
[ INFO] [1588067155.746984827]: hello world 3
[ INFO] [1588067155.846992127]: hello world 4
[ INFO] [1588067155.947024503]: hello world 5
[ INFO] [1588067156.047008014]: hello world 6
[ INFO] [1588067156.147000280]: hello world 7
[ INFO] [1588067156.246999430]: hello world 8
[ INFO] [1588067156.346999012]: hello world 9
[ INFO] [1588067156.446976824]: hello world 10
[ INFO] [1588067156.547009537]: hello world 11
[ INFO] [1588067156.647010888]: hello world 12
[ INFO] [1588067156.747013974]: hello world 13
[ INFO] [1588067156.847012155]: hello world 14
[ INFO] [1588067156.947003632]: hello world 15
[ INFO] [1588067157.047013716]: hello world 16
[ INFO] [1588067157.147016165]: hello world 17
[ INFO] [1588067157.247011400]: hello world 18
[ INFO] [1588067157.347009888]: hello world 19
[ INFO] [1588067157.447008097]: hello world 20
```