链滴

[ROS]编写简单的发布器和订阅器

作者：lixiang0

原文链接：https://ld246.com/article/1472197957435

来源网站：链滴

####1.编写发布器节点

节点是ROS中连接ROS网络的可执行程序的专业术语。这里我们创建一个发布器节点talker，它将连的广播消息。

首先进入beginner_tutorials包：

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

#####1.1代码

在beginner_tutorials包目录下创建src目录：

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

src目录包含了beginner_toturials包的所有的源文件。

在src目录下创建talker.cpp文件，添加一下内容：

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

#include <sstream>

/**
 *
 *本教程演示简单的发送消息到ROS系统
 */
int main(int argc, char **argv)
{
  /**
   * ros::init()方法需要argc和argv参数，这样就可以接收任意在命令行提供的ROS参数和名字。
   * 对于程序化的映射，你可以使用不同版本的init()来直接进行映射，
   *
   * 在使用ROS系统的其他部分之前，你必须调用一个版本的ros::init()。
   */
  ros::init(argc, argv, "talker");

  /**
   * NodeHandle是与ROS系统交流的main入口。
   * 第一个NodeHandle构造器将完全的初始化节点，最后一个NodeHandle析构器将关闭节点。
   */
  ros::NodeHandle n;

  /**
   * advertise()方法提供方式告诉ROS在给定名字的主题上发布（消息）。
   * 这将产生一个对ROSmaster节点的调用，ROS的master节点保存了发布器和订阅器的注册。
   * 在advertise（）执行完后，master节点将通知每一个向这个主题订阅的订阅器，订阅成功之后，们将与主题所在的节点进行点对点的通信。
   * advertise()返回一个发布器对象，这个发布器对象允许通过主题调用publish()方法发布消息。一发布器的所有副本都销毁了，这个主题将不会再自动的发布消息。

   * advertise()方法的第二个参数是用来发布消息的队列的大小。如果发布消息比发送消息的速度快这个参数的作用就是指定了缓存的消息数目。
   */
```

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

  ros::Rate loop_rate(10);

  /**
   * 已经发送的消息的数目. 这个用来为每一个消息创建一个唯一的字符串
   */
  int count = 0;
  while (ros::ok())
  {
    /**
     * 消息对象. 赋值之后发布.
     */
    std_msgs::String msg;

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

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

    /**
     * publish()方法用来发送消息。参数是消息对象。对象的类型必须符合上面advertise<>()构造器申明的<>中的参数类型。
     */
    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```

#####1.2.代码解析

接下来分析上面的代码。

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

ros/ros.h头文件包含了ROS系统的大部分公共类库。

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

上面的头文件包含了std_msgs包中的std_msgs/String消息。这个头文件自动地根据std_msgs包中的tring.msg文件生成。更多的信息参考http://wiki.ros.org/msg。

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

初始化ROS。这句话允许ROS从命令行完成名字的映射，同时这句代码也指定了节点的名字。节点的字在系统运行的时候必须唯一，并且取名字的时候名字必须符合基本的取名规则，比如说名字就不能/。

```
ros::NodeHandle n;
```

创建节点的handle。首次创建的NodeHandle将完成节点的初始化，最后的一次析构将节点使用的任资源。

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

向master节点声明我们将要通过chatter主题发布std_msgs/String类型的数据。master节点将通知有监听chatter的节点chatter主题将要发布数据。第二个参数是发布队列的大小。如果发布的消息太，超过了1000的最大缓存消息之后将会开始丢失旧的消息。

NodeHandle::advertise()方法返回ros::Publisher对象，这个方法提供了两种用途：

1)使用publis()方法通过创建的主题发布消息；

2)一旦超过了范围将自动的停止发布。

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

ros::Rate对象允许你指定具体的发布频率。它将追踪距离上一次调用Rate::sleep()已经过去多久，保正确的休眠时间。这里我们指定它的速率是10HZ。

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

默认的，roscpp将安装一个监听ctrl+c组合键的SIGINT handler，按下组合键之后将导致ros::ok()返false。

下面情况将导致ros::ok()返回false。

，　，，

1.SIGINT handler接收到组合键ctrl+c;

2.被另外一个重名节点踢出网络；

3.程序其他部分调用了ros::shutdown();

4.所有的ros::NodeHandles都被销毁。

，，　，

一旦ros::ok()返回false，所有的ROS调用将失败。

```
std_msgs::String msg;

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

在ROS上发布的消息是使用消息适用类，这个类是根据msg文件生成的。复杂的数据类型也是可能的但是现在，我们只使用标准的String消息，这个消息只有一个成员：” data “。

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

现在我们可以向任意连接了的节点广播消息了。

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

ROS_INFO以及类似的代替了printf/cout。更多的参考：http://wiki.ros.org/rosconsole

```
ros::spinOnce();
```

调用ros::spinOnce()在简单编程的时候是没必要的，因为我们没有接收任何的回调。然而，如果我们要在程序中添加订阅器，但是有没有在这里调用ros::spinOnce()，回调将不会执行。所以这里最好将句加上。

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

上面这句话让ros::Rate对象休眠一会儿，以保证我们是以10HZ的速率发布消息。

让我们总结下之前的工作：

1.初始化ROS系统；
2.向master节点注册：之后将通过chatter主题发布std_msgs/String消息；
3.以一秒为周期，循环发布十次消息。

现在我们继续编写一个节点来接收消息。

####2.编写订阅器节点

#####2.1源码

在beginner_tutorials包下的src下创建listener.cpp文件，并粘贴下面的内容：

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

/**
 * 本教程编写了简单的从ROS系统接收消息的消息接收器。
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  /**
   * 初始化节点，名称为listener
   */
  ros::init(argc, argv, "listener");


  ros::NodeHandle n;

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

  /**
   *循环处理回调，所有的回调都是主线程中调用。按ctrl+c退出或者节点呗master关闭
   */
  ros::spin();

  return 0;
}
```

2.2 代码解释

接下来一点点的解析代码，这里只解析之前没有的：

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

这个回调方法将会被调用，当有新的消息到达chatter主题。消息是通过一种http://www.boost.org/doc/libs/1_37_0/libs/smart_ptr/shared_ptr.htm的方法。这意味着你可以保存消息，而不用担心需你去删除，不用复制底层数据。

有很多中方式处理回调的实现。roscpp_tutorlals包有一些例子。参考：http://wiki.ros.org/roscpp/verview。

总结：

1.初始化ROS系统
2.订阅chatter主题；
3.等待消息；
4.当消息达到，chatterCallback()方法将被调用。

3.构建节点

通过之前的教程，包初始化会自动生成package.xml和CMakeList.txt文件。

在CMakeList.txt低端加上：

```
include_directories(include ${catkin_INCLUDE_DIRS})

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

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

上面的代码会创建2个可执行程序，talker和listener，默认的位置是~/catkin_ws/devel/lib/<package_name>.

之后执行：catkin_make。

关于依赖和构建安装程序，参考：http://wiki.ros.org/catkin/CMakeLists.txt

编译完成之后运行：

```
rosrun beginner_tutorials talker
```

可以看到：

```
[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
```

运行：

```
rosrun beginner_tutorials listener
```

可以看到：

```
[INFO] [WallTime: 1314931969.258941] /listener_17657_13149319687951 heard hello world 1
14931969.26
[INFO] [WallTime: 1314931970.262246] /listener_17657_13149319687951 heard hello world 1
14931970.26
[INFO] [WallTime: 1314931971.266348] /listener_17657_13149319687951 heard hello world 1
14931971.26
[INFO] [WallTime: 1314931972.270429] /listener_17657_13149319687951 heard hello world 1
14931972.27
[INFO] [WallTime: 1314931973.274382] /listener_17657_13149319687951 heard hello world 1
14931973.27
[INFO] [WallTime: 1314931974.277694] /listener_17657_13149319687951 heard hello world 1
14931974.28
[INFO] [WallTime: 1314931975.283708] /listener_17657_13149319687951 heard hello world 1
14931975.28
```

这样就完成了简单的发布器和订阅器的编写了。