# ROS编程之多线程

以下例子采用在服务端中发布两个主题，在客户端中监听这两个主题，由于其中一个主题监听处理函数耗时较长，而造成另一个监听处理函数不能及时被处理，程序演示如下：

服务端：

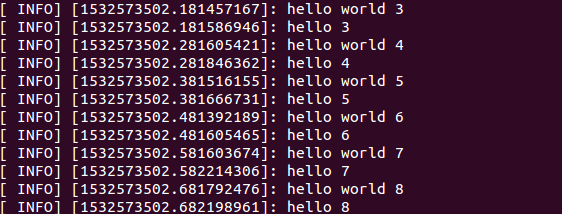
|  |
| --- |
| #include "ros/ros.h"  #include "std\_msgs/String.h"  #include <sstream>  int main(int argc, char \*\*argv)  {  ros::init(argc, argv, "multi\_pub");  ros::NodeHandle n;  ros::Publisher chatter\_pub = n.advertise<std\_msgs::String>("chatter1", 1);  ros::Publisher pub2 = n.advertise<std\_msgs::String>("chatter2", 1);  ros::Rate loop\_rate(10);  int count = 0;  while (ros::ok())  {  std\_msgs::String msg;  std::stringstream ss;  ss << "hello world " << count;  msg.data = ss.str();  std\_msgs::String msg2;  std::stringstream ss2;  ss2 << "hello " << count;  msg2.data = ss2.str();  ROS\_INFO("%s", msg.data.c\_str());  ROS\_INFO("%s", msg2.data.c\_str());  chatter\_pub.publish(msg);  pub2.publish(msg2);  ros::spinOnce();  loop\_rate.sleep();  ++count;  }  return 0;  } |

客户端：

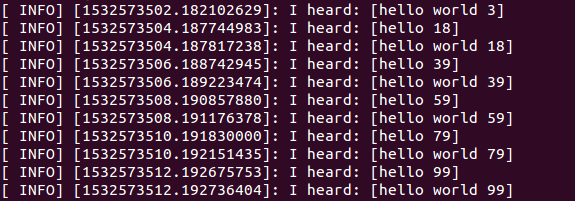
|  |
| --- |
| #include "ros/ros.h"  #include "std\_msgs/String.h"  class multiReceiver  {  public:  multiReceiver()  {  sub = nh.subscribe("chatter1", 1, &multiReceiver::chatterCallback1,this);  sub2 = nh.subscribe("chatter2", 1, &multiReceiver::chatterCallback2,this);  }  void chatterCallback1(const std\_msgs::String::ConstPtr& msg);  void chatterCallback2(const std\_msgs::String::ConstPtr& msg);  private:  ros::NodeHandle nh;  ros::Subscriber sub;  ros::Subscriber sub2;    };  void multiReceiver::chatterCallback1(const std\_msgs::String::ConstPtr& msg)  {  ROS\_INFO("I heard: [%s]", msg->data.c\_str());  ros::Rate loop\_rate(0.5);//block chatterCallback2()  loop\_rate.sleep();  }  void multiReceiver::chatterCallback2(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, "multi\_sub");  multiReceiver recOb;  ros::spin();  return 0;  } |

在客户端中，chatterCallback1使用了ros::Rate loop\_rate(0.5)以实现阻塞功能。

服务端输出：



客户端输出：



可见，由于chatterCallback1中被阻塞，处理时间较快的chatterCallback2不能及时得到执行。

为此使用多线程解决这种多发布多订阅的情况，多线程版本的客户端版本如下：

|  |
| --- |
| #include "ros/ros.h"  #include "std\_msgs/String.h"  #include <boost/thread.hpp>  class multiThreadListener  {  public:  multiThreadListener()  {  sub = n.subscribe("chatter1", 1, &multiThreadListener::chatterCallback1,this);  sub2 = n.subscribe("chatter2", 1, &multiThreadListener::chatterCallback2,this);  }  void chatterCallback1(const std\_msgs::String::ConstPtr& msg);  void chatterCallback2(const std\_msgs::String::ConstPtr& msg);  private:  ros::NodeHandle n;  ros::Subscriber sub;  ros::Subscriber sub2;    };  void multiThreadListener::chatterCallback1(const std\_msgs::String::ConstPtr& msg)  {  ROS\_INFO("I heard: [%s]", msg->data.c\_str());  ros::Rate loop\_rate(0.5);//block chatterCallback2()  loop\_rate.sleep();  }  void multiThreadListener::chatterCallback2(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, "multi\_sub");  multiThreadListener listener\_obj;  ros::AsyncSpinner spinner(2); // Use 2 threads  spinner.start();  ros::waitForShutdown();  return 0;  } |

在多线程版本中，仅是将ros::spin()修改成以下代码：



多线程版本的客户端运行效果如下，可见chatterCallback2得到了及时执行。

