ROS 十天学基础

罗振华
dreamluo
哈尔滨工业大学 中科院深圳先进技术研究院
支持：www.cnros.org 此网站希望可以互相探讨
百度贴吧：ros 机器人操作系统学习交流吧
机器人操作系统学习交流吧
文字教程参考 小菜鸟上校的博客

第一课 Linux 软件源更新

身为在中国的 ubuntu 使用者，当您费劲千辛万苦安装好 ubuntu，准备开始体验一下开源操作系统的魅力的时候，您会发现有很多的软件还没有安装或者是更新，这时候千万不要急着去安装或者更新，因为您还没有更新 ubuntu 的软件源，这样直接安装会非常的慢，慢到让人 eggache。如果您先更新了软件源，一定会事半功倍。

按 ctrl+alt+t 打开一个新的终端，在里面输入：

[html] view plaincopy

```
1. sudo gedit /etc/apt/sources.list
```

然后将打开的文件中的内容全部都清空，如果您不放心的话，可以先将这个文件备份一下。然后挑选如下的一个或者几个源复制进去就可以了。

搜狐源

[html] view plaincopy

```
1. deb http://mirrors.sohu.com/ubuntu/ precise main restricted
2. deb-src http://mirrors.sohu.com/ubuntu/ precise main restricted
3. deb http://mirrors.sohu.com/ubuntu/ precise-updates main restrict
   ed
4. deb-src http://mirrors.sohu.com/ubuntu/ precise-updates main rest
   ricted
5. deb http://mirrors.sohu.com/ubuntu/ precise universe
6. deb-src http://mirrors.sohu.com/ubuntu/ precise universe
7. deb http://mirrors.sohu.com/ubuntu/ precise-updates universe
8. deb-src http://mirrors.sohu.com/ubuntu/ precise-updates universe

9. deb http://mirrors.sohu.com/ubuntu/ precise multiverse
10.deb-src http://mirrors.sohu.com/ubuntu/ precise multiverse
11.deb http://mirrors.sohu.com/ubuntu/ precise-updates multiverse
```

```
deb-src http://mirrors.sohu.com/ubuntu/ precise-updates multiverse
deb http://mirrors.sohu.com/ubuntu/ precise-backports main restricted universe multiverse
deb-src http://mirrors.sohu.com/ubuntu/ precise-backports main restricted universe multiverse
deb http://mirrors.sohu.com/ubuntu/ precise-security main restricted
deb-src http://mirrors.sohu.com/ubuntu/ precise-security main restricted
deb http://mirrors.sohu.com/ubuntu/ precise-security universe
deb-src http://mirrors.sohu.com/ubuntu/ precise-security universe

deb http://mirrors.sohu.com/ubuntu/ precise-security multiverse
deb-src http://mirrors.sohu.com/ubuntu/ precise-security multiverse
deb http://extras.ubuntu.com/ubuntu precise main
deb-src http://extras.ubuntu.com/ubuntu precise main
```

163 的源

[html] view plaincopy

```
deb http://mirrors.163.com/ubuntu/ precise main restricted
deb-src http://mirrors.163.com/ubuntu/ precise main restricted
deb http://mirrors.163.com/ubuntu/ precise-updates main restricted
deb-src http://mirrors.163.com/ubuntu/ precise-updates main restricted
deb http://mirrors.163.com/ubuntu/ precise universe
deb-src http://mirrors.163.com/ubuntu/ precise universe
deb http://mirrors.163.com/ubuntu/ precise-updates universe
deb-src http://mirrors.163.com/ubuntu/ precise-updates universe
deb http://mirrors.163.com/ubuntu/ precise multiverse
deb-src http://mirrors.163.com/ubuntu/ precise multiverse
deb http://mirrors.163.com/ubuntu/ precise-updates multiverse
deb-src http://mirrors.163.com/ubuntu/ precise-updates multiverse

deb http://mirrors.163.com/ubuntu/ precise-backports main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ precise-backports main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ precise-security main restricted
```

```
16. deb-src http://mirrors.163.com/ubuntu/ precise-security main rest
    ricted
17. deb http://mirrors.163.com/ubuntu/ precise-security universe
18. deb-src http://mirrors.163.com/ubuntu/ precise-security universe

19. deb http://mirrors.163.com/ubuntu/ precise-security multiverse
20. deb-src http://mirrors.163.com/ubuntu/ precise-security multivers
    e
21. deb http://extras.ubuntu.com/ubuntu precise main
22. deb-src http://extras.ubuntu.com/ubuntu precise main
```

教育网的源

#兰州大学

[html] view plaincopy

```
1. deb http://mirror.lzu.edu.cn/ubuntu/ intrepid main multiverse res
   tricted universe
2. deb http://mirror.lzu.edu.cn/ubuntu/ intrepid-backports main mult
   iverse restricted universe
3. deb http://mirror.lzu.edu.cn/ubuntu/ intrepid-proposed main multi
   verse restricted universe
4. deb http://mirror.lzu.edu.cn/ubuntu/ intrepid-security main multi
   verse restricted universe
5. deb http://mirror.lzu.edu.cn/ubuntu/ intrepid-updates main multiv
   erse restricted universe
6. deb http://mirror.lzu.edu.cn/ubuntu-cn/ intrepid main multiverse
   restricted universe
```

#电子科技大学

[html] view plaincopy

```
1. deb http://ubuntu.uestc.edu.cn/ubuntu/ precise main restricted un
   iverse multiverse
2. deb http://ubuntu.uestc.edu.cn/ubuntu/ precise-backports main res
   tricted universe multiverse
3. deb http://ubuntu.uestc.edu.cn/ubuntu/ precise-proposed main rest
   ricted universe multiverse
4. deb http://ubuntu.uestc.edu.cn/ubuntu/ precise-security main rest
   ricted universe multiverse
```

```
5. deb http://ubuntu.uestc.edu.cn/ubuntu/ precise-updates main restr
   icted universe multiverse
6. deb-src http://ubuntu.uestc.edu.cn/ubuntu/ precise main restricte
   d universe multiverse
7. deb-src http://ubuntu.uestc.edu.cn/ubuntu/ precise-backports main
    restricted universe multiverse
8. deb-src http://ubuntu.uestc.edu.cn/ubuntu/ precise-proposed main
   restricted universe multiverse
9. deb-src http://ubuntu.uestc.edu.cn/ubuntu/ precise-security main
   restricted universe multiverse
10. deb-src http://ubuntu.uestc.edu.cn/ubuntu/ precise-updates main r
    estricted universe multiverse
```

#中国科技大学

[html] view plaincopy

```
1. deb http://debian.ustc.edu.cn/ubuntu/ precise main restricted uni
   verse multiverse
2. deb http://debian.ustc.edu.cn/ubuntu/ precise-backports restricte
   d universe multiverse
3. deb http://debian.ustc.edu.cn/ubuntu/ precise-proposed main restr
   icted universe multiverse
4. deb http://debian.ustc.edu.cn/ubuntu/ precise-security main restr
   icted universe multiverse
5. deb http://debian.ustc.edu.cn/ubuntu/ precise-updates main restri
   cted universe multiverse
6. deb-src http://debian.ustc.edu.cn/ubuntu/ precise main restricted
    universe multiverse
7. deb-src http://debian.ustc.edu.cn/ubuntu/ precise-backports main
   restricted universe multiverse
8. deb-src http://debian.ustc.edu.cn/ubuntu/ precise-proposed main r
   estricted universe multiverse
9. deb-src http://debian.ustc.edu.cn/ubuntu/ precise-security main r
   estricted universe multiverse
10. deb-src http://debian.ustc.edu.cn/ubuntu/ precise-updates main re
    stricted universe multiverse
```

#北京理工大学

[html] view plaincopy

```
1. deb http://mirror.bjtu.edu.cn/ubuntu/ precise main multiverse res
   tricted universe
2. deb http://mirror.bjtu.edu.cn/ubuntu/ precise-backports main mult
   iverse restricted universe
3. deb http://mirror.bjtu.edu.cn/ubuntu/ precise-proposed main multi
   verse restricted universe
4. deb http://mirror.bjtu.edu.cn/ubuntu/ precise-security main multi
   verse restricted universe
5. deb http://mirror.bjtu.edu.cn/ubuntu/ precise-updates main multiv
   erse restricted universe
6. deb-src http://mirror.bjtu.edu.cn/ubuntu/ precise main multiverse
    restricted universe
7. deb-src http://mirror.bjtu.edu.cn/ubuntu/ precise-backports main
   multiverse restricted universe
8. deb-src http://mirror.bjtu.edu.cn/ubuntu/ precise-proposed main m
   ultiverse restricted universe
9. deb-src http://mirror.bjtu.edu.cn/ubuntu/ precise-security main m
   ultiverse restricted universe
10.deb-src http://mirror.bjtu.edu.cn/ubuntu/ precise-updates main mu
   ltiverse restricted universe
```

关闭文件，然后输入：

[html] view plaincopy

```
1. sudo apt-get update
```

更新一下源，就可以了。

第二课　ROS 安装

学习 ROS 有两周了吧，发现网上的中文资料很少。再网上搜出来的好多 ROS 都是和路由器相关的一些东西，我们这里说的 ROS 是 robot operating system，也就是机器人操作系统，所以我的学习主要是基于 ROS 的官网：http://wiki.ros.org。当然也有网友古月在 csdn 上发表了一系列关于 ROS 的学习博客，很有参考价值，可以在百度中搜索之，今天主要来讲述一下如何安装 ROS。

打开 http://wiki.ros.org，点击超链接 install，在新的页面中点击 ubuntu 超链接，笔者所使用的操作系统是 ubuntu12.04，用户可以根据自己的需要安装需要的系统。不过还是推荐使用 ubuntu，因为 ubnutu 对 ros 的支持是最好的。之后就可以看到安装的步骤了，如果您的英文很好，那还是推荐您参照 ROS 的官网安装。不过在安装之前，有一个很重要的步骤，那就是更新软件源。可以参考笔者的另一篇博客，ubuntu 更新软件源的方法。笔者就是因为刚开始没有更新软件源，导致安装 ros 的速度非常慢，结果装了两天也没有装好，坑啊。

首先要查看你的 ubuntu 的版本：

按住 ctrl+alt+t 打开一个终端，在里面输入：

[plain] view plaincopy

```
1. cat /etc/issue
```

如果您的 ubuntu 版本是 ubuntu12.04，那么在终端输入如下命令：

[html] view plaincopy

```
1. sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise
   main" > /etc/apt/sources.list.d/ros-latest.list'
```

如果是 ubuntu12.10：

[html] view plaincopy

```
1. sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu quantal
   main" > /etc/apt/sources.list.d/ros-latest.list'
```

如果是 ubuntu13.04：

[html] view plaincopy

```
1. sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu raring m
   ain" > /etc/apt/sources.list.d/ros-latest.list'
```

在执行完上述命令后，需要在终端中以下命令：

[html] view plaincopy

```
1. wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

在上述命令完成后，需要输入以下命令：

[html] view plaincopy

```
1. sudo apt-get update
```

上面的这条命令的执行时间会比较长，请耐心等待。等这条指令执行完成后，请在终端输入如下指令：

[html] view plaincopy

```
1. sudo apt-get install ros-hydro-desktop-full
```

上面的这条指令的执行时间会更长，因为这是安装 ROS 的过程，请不要心急，耐心等待。在安装过程中，会出现选择 yes 和 no 的对话框，一定要选 yes，其他数据默认就可以了，不用修改。

待安装完成以后，可以使用以下命令来查看已安装的可以使用的包：

[html] view plaincopy

```
1. apt-cache search ros-hydro
```

然后需要对新安装的 ROS 进行初始化：

[html] view plaincopy

```
1. sudo rosdep init
```

待上面的这条命令执行完成以后，可以看到有提示需要执行 rosdep update 命令，那么继续输入：

[html] view plaincopy

```
1. rosdep update
```

然后需要进行环境变量的设置：

[html] view plaincopy

```
1. echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
2. source ~/.bashrc
```

以上的这个设置是永久性的，不需要每次打开一个终端都需要进行一次设置。当然也可以使用临时性的：

[html] view plaincopy

```
1. source /opt/ros/hydro/setup.bash
```

这个需要在每次打开一个终端之前，都需要执行一次上述的命令。推荐使用第一条指令。

接下来需要安装一个非常重要的工具 rosinstall，这个工具在以后会非常有用的，安装命令如下：

[html] view plaincopy

```
1. sudo apt-get install python-rosinstall
```

第三课　ROS 创建空间、包

在官网上本节的题目是 Creating a workspace for catkin，其中的 catkin 不知道是什么意思，在网上找到的结果是：（1）卡婷是一个广告公司，（2）茉莉花。这两种翻译显然都不太合适，不过不知道也没关系，影响不大。我们知道 catkin 是一个 ROS 中的工具就行了。本节的主要目的是创建一个 catkin 工作空间，在这个工作空间中，catkin 的包可以被编译。

如果您还没有安装 catkin 的话，请首先安装 catkin。不过如果按照前面的步骤的话，catkin 已经安装了。

首先需要修改环境变量，按 ctrl+alt+t 打开一个终端，在里面输入：

[html] view plaincopy

```
1. source /opt/ros/hydro/setup.bash
```

也可以按照前面所讲的，将其直接修改为永久性的。

创建一个工作空间：

[html] view plaincopy

```
1. mkdir -p ~/catkin_ws/src
```

[html] view plaincopy

```
1. cd ~/catkin_ws/src
```

通过上面两条命令，就可以创建一个工作空间，并转到已创建好的工作空间之下，尽管这个空间是空的，我们仍然可以构建（build）它：

[html] view plaincopy

```
1. cd ~/catkin_ws/
2. catkin_make
```

当时用 catkin 工作空间时，catkin_make 是一个非常方便的命令行工具。如果您看一下当前的工作目录，您会发现里面多了两个文件夹“build”和“devel”。在 devel 文件夹下，您可以看到很多 setup.*sh 文件。输入如下命令配置您的工作空间：

[html] view plaincopy

```
1. source devel/setup.bash
```

创建包

ROS 学习（三）中，笔者不知道 catkin 到底是个什么东东，后来终于在官方网站上找到了答案，原来 catkin 是 ROS 的一个官方的编译构建系统，是原本的 ROS 的编译构建系统 rosbuild 的后继者。catkin 的来源有点复杂，我们可以慢慢的讲一下其中的渊源。ROS 来源于 Willow Garage 这个公司，他们希望借助开源的力量，使 ROS 发扬光大。而在英语中，willow 的意思是柳树，catkin 是柳絮的意思，为了纪念的作用吧，因而为这个软件命名为 catkin。

这篇博客的主要内容是介绍如何使用 catkin 创建一个 ROS 包。一个 catkin 的包主要有以下几部分组成：（1）必须包括一个 package.xml 文件，（2）必须包括一个 CMakeLists.txt 文件，（3）在每一个文件夹下只能有一个包，且包不允许嵌套。一个最简单的包类似于如下的形式：

[html] view plaincopy

```
1. my_package/
2.   CMakeLists.txt
3.   package.xml
```

官网上推荐在 catkin 工作空间中使用 catkin 包，当然也可以独立使用 catkin 包，一个典型的工作空间结构如下：

[html] view plaincopy

```
1. workspace_folder/        -- WORKSPACE
2.   src/                   -- SOURCE SPACE
3.     CMakeLists.txt       -- 'Toplevel' CMake file, provided by ca
   tkin
4.     package_1/
5.       CMakeLists.txt     -- CMakeLists.txt file for package_1
6.       package.xml        -- Package manifest for package_1
7.     ...
8.     package_n/
9.       CMakeLists.txt     -- CMakeLists.txt file for package_n
10.      package.xml        -- Package manifest for package_n
```

我们之前已经创建过一个空的工作空间：catkin_ws，下面我们来看一下如何在一个工作空间中创建一个包。在创建一个 catkin 包时需要使用 catkin_create_pkg 脚本。

首先进入到目录～/catkin_ws/src 下，使用如下命令：

[html] view plaincopy

```
1. cd ~/catkin_ws/src
```

接着创建一个名字为 beginner_tutorials 的包，它直接以来与一下三个包：std_msgs,rospy 以及 roscpp，使用如下命令：

[html] view plaincopy

```
1. catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

在创建的 beginner_tutorials 文件夹下可以看到 package.xml 和 CMakeLists.txt。catkin_create_pkg 要求您给出包的名字，及选择性的给出所创建的包依赖于哪一个包。他的使用方法如下：

[html] view plaincopy

```
1. catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

这样，我们的一个包就创建好了，我们可能会需要对包之间的依赖性做一下解释。我们可以使用 rospack 命令来查看包之间的依赖关系。查看直接依赖关系：

[html] view plaincopy

```
1. rospack depends1 beginner_tutorials
```

可以看到，返回的结果正是我们使用 catkin_create_pkg 时，所使用的参数。我们还可以直接在 beginner_tutorials 包下的 package.xml 中查看包的依赖关系。使用命令：

[html] view plaincopy

```
1. roscd beginner_tutorials
2. cat package.xml
```

结果如下：

[html] view plaincopy

```
1. <package>
2. ...
3.   <buildtool_depend>catkin</buildtool_depend>
4.   <build_depend>roscpp</build_depend>
5.   <build_depend>rospy</build_depend>
6.   <build_depend>std_msgs</build_depend>
7. ...
8. </package>
```

在通常情况下，一个包所依赖的包又会依赖许多其它的包，这称为间接依赖。我们使用如下命令来查看 rospy 依赖的包：

[html] view plaincopy

```
1. rospack depends1 rospy
```

返回结果如下：

[html] view plaincopy

```
1. genpy
2. rosgraph
3. rosgraph_msgs
4. roslib
5. std_msgs
```

一个包可以有很多的间接依赖关系，我们可以使用命令：

[html] view plaincopy

```
1. rospack depends beginner_tutorials
```

来进行查看。返回结果如下：

[html] view plaincopy

```
1. cpp_common
2. rostime
3. roscpp_traits
4. roscpp_serialization
5. genmsg
6. genpy
7. message_runtime
8. rosconsole
9. std_msgs
10.rosgraph_msgs
11.xmlrpcpp
12.roscpp
13.rosgraph
14.catkin
15.rospack
16.roslib
17.rospy
```

接下来配置您的 package.xml 包，我们将会一个标签一个标签的分析这个 xml 文件。首先是 description 标签：

[html] view plaincopy

```
1. <description>The beginner_tutorials package</description>
```

在这个标签中的内容可以改变为任何的内容，不过一般是对这个包的一个简述，尽量简单就行。接下来是 maintainer 标签：

[html] view plaincopy

```
1. <!-- One maintainer tag required, multiple allowed, one person pe
   r tag -->
2. <!-- Example:  -->
3. <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintaine
   r> -->
4. <maintainer email="user@todo.todo">user</maintainer>
```

这是一个很重要的标签，因为可以根据这个标签知道它的维护者，另外标签的 email 属性也是必须的，可有多个 maintainer 标签。接下来是 license 标签，

[html] view plaincopy

```
1. <!-- One license tag required, multiple allowed, one license per
   tag -->
2.   <!-- Commonly used license strings: -->
3.   <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1
   , LGPLv3 -->
4.   <license>TODO</license>
```

在后面的使用中，一般我们将 license 修改为 BSD。接下来是依赖性的标签：

[html] view plaincopy

```
1. <!-- The *_depend tags are used to specify dependencies -->
2. <!-- Dependencies can be catkin packages or system dependencies -
   -->
3. <!-- Examples: -->
4. <!-- Use build_depend for packages you need at compile time: -->

5. <!--   <build_depend>genmsg</build_depend> -->
6. <!-- Use buildtool_depend for build tool packages: -->
7. <!--   <buildtool_depend>catkin</buildtool_depend> -->
8. <!-- Use run_depend for packages you need at runtime: -->
```

```
<!--   <run_depend>python-yaml</run_depend> -->
<!-- Use test_depend for packages you need only for testing: -->

<!--   <test_depend>gtest</test_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
```

我们向里面添加了 run_depend:

[html] view plaincopy

```
<run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
```

好了，经过以上的步骤，我们的 package.xml 已经配置好了。

在这篇博客中将会介绍，如何在工作空间中构建和使用一个包。

首先，我们来看一下如何在 catkin 工作空间中，使用 catkin_make 工具从源文件构建和安装一个包。使用 catkin_make 来构建一个 catkin 工作空间是非常容易的，您必须在 catkin 工作空间的顶层使用 catkin_make 命令。下面的演示了一个典型的工作流程：

[html] view plaincopy

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

# Add/Edit source files

$ cd ~/catkin_ws/src/beginner_tutorials

# Update CMakeFiles.txt to reflect any changes to your sources

$ cd ~/catkin_ws

$ catkin_make
```

首先打开 beginner_tutorials 包下面的 src 文件夹，在里面添加或者编辑源文件。然后回到包 beginner_tutorials 的根目录下，更新一下 CMakeLists.txt 文件，最后回到工作空间的根目录下，使用 catkin_make 命令进行构建。

上面的流程会将~/catkin_ws/src 目录下的包构建到~/catkin_ws/build 目录下。任何的源文件、python 库、脚本，以及其他的静态文件，将会留在源空间~/catkin_ws/src 中。

然而所有产生的文件，像库文件、可执行文件以及产生的代码都被放置在 devel 中。使用如下命令来创建 install 工作空间：

[html] view plaincopy

```
1. cd ~/catkin_ws
2. catkin_make install
```

上面的两条命令可以用下面一条指令来代替：

[html] view plaincopy

```
1. cd ~/catkin_ws/build && make install
```

你可以使用 devel 或者是 install 空间，但不能同时使用。他们各有好处，具体情况，应该具体对待。在工作空间中还是推荐使用 devel。如果您之前已经有了一个编译好的工作空间，并且您在里面添加了一个新的包，可以使用如下命令将这个包添加进去：

[html] view plaincopy

```
1. catkin_make --force-cmake
```

这样我们就完成了一个包的构建。

第四课 ROS 节点、话题、消息、服务、参数的理解

经过前面的学习，我们已经知道了如何构建一个 ROS 的包，这篇博客将介绍 ROS 中的节点的概念。

在继续之前，请按 ctrl+alt+t 打开一个终端，在里面输入：

[html] view plaincopy

```
1. sudo apt-get install ros-<distro>-ros-tutorials
```

安装一个轻量级的模拟器，命令中的"<distro>"需要替换为你自己的 ros 版本，若按照前面的教程的话，替换为 hydro。

下面来看一下 ROS 中图的相关概念：

节点（NODE）：一个节点就是一个可执行程序，它使用 ROS 可以和其他节点进行通信。

消息（Message）：当在一个话题上，发布或订阅时所使用的 ROS 的数据类型。

话题（Topic）：节点可以在一个话题上发布消息，同样也可以订阅一个话题来接收消息。

主机（Master）：是 ROS 的名字服务器。

ROS 的客户端库允许允许用不同的编程语言编写的节点之间相互通信。

roscore 是你在使用 ros 之前应该首先运行的程序。在终端中运行 roscore：

[html] view plaincopy

```
1. roscore
```

rosnode 命令显示了正在运行的 ros 的节点的信息。如下命令列出了活跃的 ros 节点，新打开一个终端输入：

[html] view plaincopy

```
1. rosnode list
```

你将会看到：

```
/rosout
```

着说明了当前只有一个节点 rosout 在运行。下面的这个命令可以返回活跃的节点的信息：

[html] view plaincopy

```
1. rosnode info /rosout
```

rosrun 允许你直接运行一个包里面的节点。使用方法如下：

[html] view plaincopy

```
1. rosrun [package_name] [node_name]
```

在终端中输入：

[html] view plaincopy

```
1. rosrun turtlesim turtlesim_node
```

将会看到在屏幕上出现了一只乌龟。这个命令的作用是运行 turtlesim 包下面的 turtlesim_node 节点，多次运行这个命令会看到乌龟可能会不同，这算不算是一个惊喜呢。

新打开一个终端，在里面输入：

[html] view plaincopy

```
1. rosnode list
```

可以看到我们刚刚运行的节点，出现在了列表中。我们还可以在命令行下给运行的节点直接指定名字，将刚刚打开的乌龟关闭，输入：

[html] view plaincopy

```
1. rosrun turtlesim turtlesim_node __name:=my_turtle
```

再使用 rosnode list 就可以看到我们所修改的名字。

还可以使用 ping 命令：

[html] view plaincopy

```
1. rosnode ping my_turtle
```

首先需要打开一个终端在里面运行 roscore：

[html] view plaincopy

```
1. roscore
```

再打开一个终端，在里面运行一个 turtlesim_node 节点：

[html] view plaincopy

```
1. rosrun turtlesim turtlesim_node
```

打开另一个终端，在里面输入：

[html] view plaincopy

```
1. rosrun turtlesim turtle_teleop_key
```

在运行完这条命令后，在这个终端下，按键盘上的方向键可以看到，之前我们运行的乌龟开始移动，:-)，很有意思吧。

turtlesim_node 和 turtle_teleop_key 通过 ROS 的话题来相互通信。turtle_teleop_key 把用户按下的键发布到话题上，turtlesim_node 也订阅了同一个话题用来接收用户按下的键，并做出相应的动作。我们有这个例子可以体会到话题的作用，更加深刻的认识到了节点的概念。

rqt_graph 创建了一个当前系统中运行的节点的动态图。如果你没有安装 rqt 的话，请首先安装：

[html] view plaincopy

```
1. sudo apt-get install ros-hydro-rqt
```

[html] view plaincopy

```
1. sudo apt-get install ros-hydro-rqt-common-plugins
```

首先按照 ROS 学习（六）博客中讲到的例子运行一下，即通过按键来控制乌龟的移动。我们在终端中输入命令：

[html] view plaincopy

```
1. rosrun rqt_graph rqt_graph
```

就可以看到一个表示节点之间的关系的图。将鼠标放在图片上面可以看到图片会变为高亮。可以看到两个节点是通过/turtle1/command_velocity 话题来进行通信的。

使用 rostopic echo 命令可以查看在一个话题上发布的数据。它的使用方法如下：

[html] view plaincopy

```
1. rostopic echo [topic]
```

例如，我们可以输入：

[html] view plaincopy

```
1. rostopic echo /turtle1/cmd_vel
```

之后我们在运行 turtle_teleop_key 节点的窗口下按方向键，就可以在刚才的窗口看到输出的数据如下：

[html] view plaincopy

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

现在我们重新运行：

[html] view plaincopy

```
rosrun rqt_graph rqt_graph
```

就可以看到话题刚才只有一个订阅者 现在又多了一个订阅者。

使用 rostopic list 可以列举出当前系统中的话题：

[html] view plaincopy

```
rostopic list
```

使用

[html] view plaincopy

```
rostopic list -v
```

可以查看每一个话题的发布者和订阅者的个数。

话题上的通信通过节点之间发送消息来完成。对于发布者（turtle_teleop_key）和接收者（turtlesim_node）的通信，发布者和接收者必须使用类型相同的消息。也就是说，一个话题的类型是通过在它上面发布的消息的类型来定义的。可以通过：

[html] view plaincopy

```
1. rostopic type /turtle1/cmd_vel
```

来查看/turtle1/cmd_vel 的消息类型。你得到的返回类型如下：

[html] view plaincopy

```
1. geometry_msgs/Twist
```

我们可以使用 rosmsg 来查看一个消息类型的详细信息：

[html] view plaincopy

```
1. rosmsg show geometry_msgs/Twist
```

得到的结果如下：

[html] view plaincopy

```
1. geometry_msgs/Vector3 linear
2.   float64 x
3.   float64 y
4.   float64 z
5. geometry_msgs/Vector3 angular
6.   float64 x
7.   float64 y
8.   float64 z
```

现在我们已经知道了 turtlesim 接收的消息类型，现在我们可以给它发布消息了。rostopic pub 命令用来发布消息，它的使用方法如下：

[html] view plaincopy

```
1. rostopic pub [topic] [msg_type] [args]
```

实例如下：

[html] view plaincopy

```
1. rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.
   0, 0.0]' '[0.0, 0.0, 1.8]'
```

我们可以看到，乌龟运动了一个弧度。这是一个非常复杂的命令，我们详细分析一下：

[html] view plaincopy

```
1. rostopic pub
```

这个命令发布话题，

[html] view plaincopy

```
1. -1
```

这个参数的意思是只发布一个命令，然后退出。

[html] view plaincopy

```
1. /turtle1/cmd_vel
```

这是要发布消息的话题，

[html] view plaincopy

```
1. --
2. 2.0 1.8
```

这两个参数我们可以先不用深究。我们可以使用 rostopic pub -r 命令让乌龟持续运动，如：

[html] view plaincopy

```
1. rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0,
   0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

我们可以看到，乌龟围绕着一个圈在不停的运动。

通过上面丰富的实例，相信大家会对 ROS 的话题概念会有一个清楚的认识。

服务是节点之间通信的另一种方式，服务允许节点发起一个请求和接收一个响应。

打开终端在里面输入：

[html] view plaincopy

```
1. roscore
```

另外打开一个终端在里面输入：

[html] view plaincopy

```
1. rosrun turtlesim turtlesim_node
```

使用 rosservice list 可以查看一个节点提供的服务：

[html] view plaincopy

```
1. rosservice list
```

返回结果如下：

[html] view plaincopy

```
1. /clear
2. /kill
3. /reset
4. /rosout/get_loggers
5. /rosout/set_logger_level
6. /spawn
7. /teleop_turtle/get_loggers
8. /teleop_turtle/set_logger_level
9. /turtle1/set_pen
10./turtle1/teleport_absolute
11./turtle1/teleport_relative
12./turtlesim/get_loggers
13./turtlesim/set_logger_level
```

使用 rosservice type 可以查看提供的服务的类型，它的使用方法如下：

[html] view plaincopy

```
1. rosservice type [service]
```

实例：

[html] view plaincopy

```
1. rosservice type clear
```

查看 clear 服务的类型，返回结果为：

[html] view plaincopy

```
1. std_srvs/Empty
```

返回的结果是 empty，这就说明当调用这个服务时，传递的参数为空，即没有参数。

我们可以使用 rosservice call 来调用一个服务，它的使用方法如下：

[html] view plaincopy

```
1. rosservice call [service] [args]
```

例如：

[html] view plaincopy

```
1. rosservice call clear
```

这条命令乌龟运动痕迹都清理掉了。

下面将演示一个带参数的服务 spawn，我们首先查看一下它的类型：

[html] view plaincopy

```
1. rosservice type spawn| rossrv show
```

返回的结果如下：

[html] view plaincopy

```
1. float32 x
2. float32 y
3. float32 theta
4. string name
5. ---
6. string name
```

spawn 命令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：

[html] view plaincopy

```
1. rosservice call spawn 2 2 0.2 ""
```

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。

rosparam 命令允许你在 ROS 的参数服务器上操作和存储数据，参数服务器可以存储整数，浮点数，布尔类型，字典，列表。ROS 使用 YAML 标记语言作为语法，在简单的情况下，YAML 看起来是非常自然的：1 是整数，1.0 是浮点数，one 是字符串，true 是布尔类型，[1,2,3]是一个列表，{a:b,c:d}是一个字典。我们使用 rosparam list 命令可以查看参数服务器上的内容：

[html] view plaincopy

```
1. rosparam list
```

返回的结果如下：

[html] view plaincopy

```
1. /background_b
2. /background_g
3. /background_r
4. /roslaunch/uris/aqy:51932
5. /run_id
```

我们可以看到 turtlesim_node 节点在参数服务器上存储了三个表示北京颜色的参数。

使用 rosparam set 可以改变参数服务器上的参数，而 rosparam get 可以获取参数服务器上参数的值，它的使用方法如下：

[html] view plaincopy

```
1. rosparam set [param_name]
2. rosparam get [param_name]
```

我们改变背景颜色如下：

[html] view plaincopy

```
1. rosparam set background_r 150
```

这条命令执行完成后，我们会发现北京颜色并没有发生变化，对了我们得刷新一下，怎么刷新呢，这条命令我们已经学过了哦：

[html] view plaincopy

```
1. rosservice call /clear
```

使用下面的命令：

[html] view plaincopy

```
1. rosparam get /background_r
```

可以获得我们刚刚修改的/background_r 的值，当然我们也可以将/background_r 替换为/background_g 或/background_b。我们可以使用：

[html] view plaincopy

```
1. rosparam get /
```

一次性获得所有的参数的值。

使用命令 rosparam dump 可以将参数服务器的内容写到一个文件中，它的使用方法如下：

[html] view plaincopy

```
1. rosparam dump [file_name]
```

例如：

[html] view plaincopy

```
1. rosparam dump params.yaml
```

我们将参数服务器的内容写到 params.yaml 的文件中。

第五课　建立自己的 roslaunch msg srv

打开一个新的终端在里面输入：

[html] view plaincopy

```
1. sudo apt-get install ros-hydro-rqt ros-hydro-rqt-common-plugins r
   os-hydro-turtlesim
```

安装使用 rqt_console 所需要的插件，如果你不知道之前是否安装，没关系，把这个命令运行一下就行了，它不会对你之前的系统有任何的伤害。在终端中运行 rqt_console：

[html] view plaincopy

```
1. rosrun rqt_console rqt_console
```

再新打开一个终端运行：

[html] view plaincopy

```
1. rosrun rqt_logger_level rqt_logger_level
```

在运行这两条命令的时候，你会看到弹出了两个窗口，在一个新的终端中输入：

[html] view plaincopy

```
1. rosrun turtlesim turtlesim_node
```

运行一只乌龟节点，就可以看到在 rqt_console 中有信息输出。我们点击一下 logger_level 窗口中的 refresh 按钮，就可以看到在最左边的框中出现了 turtlesim，选中 turtlesim，选中右边的 warn，运行下面指令：

[html] view plaincopy

```
1. rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0,
   0.0, 0.0]' '[0.0, 0.0, 0.0]'
```

就可以看到 console 窗口中不断的输出警告信息，如过我们将 logger_level 窗口中的级别修改为 fatal，就会发现信息的输出停止。如果再将它选为 warn，信息就会继续输出。

在 ros 中将信息分为五个等级，从高到低如下：

[html] view plaincopy

```
1. Fatal
2. Error
```

```
3. Warn
4. Info
5. Debug
```

fatal 有最高的优先级，而 debug 的优先级最低。如果你选择的优先级是 debug，那么就会接收所有的消息，如果你选择的是 warn，就会接收 warn 及高于它的优先级的信息。

roslaunch 命令从 launch 文件中启动一个节点，它的使用方法如下：

[html] view plaincopy

```
1. roslaunch [package] [filename.launch]
```

首先切换到 beginner_tutorials 文件下：

[html] view plaincopy

```
1. roscd beginner_tutorials
```

创建 launch 文件夹，切换到该文件夹下：

[html] view plaincopy

```
1. mkdir launch
2. cd launch
```

输入

[html] view plaincopy

```
1. gedit turtlemimic.launch
```

将下面的内容复制到文件中：

[html] view plaincopy

```
1. <launch>
2.
3.   <group ns="turtlesim1">
4.     <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
5.   </group>
6.
7.   <group ns="turtlesim2">
8.     <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
9.   </group>
10.
```

```
11.   <node pkg="turtlesim" name="mimic" type="mimic">
12.     <remap from="input" to="turtlesim1/turtle1"/>
13.     <remap from="output" to="turtlesim2/turtle1"/>
14.   </node>
15.
16. </launch>
```

保存退出就可以了。

现在使用如下命令载入：

[html] view plaincopy

```
1. roslaunch beginner_tutorials turtlemimic.launch
```

就可以看到新打开一只乌龟节点。

在一个新打开的终端中输入：

[html] view plaincopy

```
1. rostopic pub /turtlesim1/turtle1/cmd_vel geometry_msgs/Twist -r 1
    -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

就可以看到两只乌龟都在运动。下面来解释一下原因：<lanuch>标签用来识别这是一个 launch 文件，

[html] view plaincopy

```
1. <group ns="turtlesim1">
2.   <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
3. </group>
4.
5. <group ns="turtlesim2">
6.   <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
7. </group>
```

上面这些内容表示我们启动了两只乌龟，

[html] view plaincopy

```
1. <node pkg="turtlesim" name="mimic" type="mimic">
2.   <remap from="input" to="turtlesim1/turtle1"/>
3.   <remap from="output" to="turtlesim2/turtle1"/>
4. </node>
```

上面的这些被荣使 turtlesim2 模仿 turtlesim1。所以虽然我们只向 turtlesim1 发布了话题，turtlesim2 也做出了反应。

msg 是一个描述 ROS 中消息的域的简单的文本文件，它用来为消息产生不同语言的源代码。

一个 srv 文件描述一个服务，它由两部分组成，请求和服务。

msg 文件被存储在一个包的 msg 目录下，srv 文件被存储在 srv 目录下。msg 是简单的文本文件，它的每一行由一个与的类型和域的名字组成。你可以使用的域的类型有：

[html] view plaincopy

```
1. int8, int16, int32, int64 (plus uint*)
2. float32, float64
3. string
4. time, duration
5. other msg files
6. variable-length array[] and fixed-length array[C]
```

ROS 中还有一个特殊的类型 Header，header 包括了一个时间戳和一个经产在 ROS 中使用的坐标框架信息。你经常会看到在 msg 文件的第一行代码是：

[html] view plaincopy

```
1. Header header
```

下面是一个使用了 Header 的 msg 的例子：

[html] view plaincopy

```
1. Header header
2. string child_frame_id
3. geometry_msgs/PoseWithCovariance pose
4. geometry_msgs/TwistWithCovariance twist
```

srv 文件和 msg 文件是一样的，除了它们包括两部分：请求和响应，这两部分通过'---'分隔。下面是一个 srv 文件的例子：

[html] view plaincopy

```
1. int64 A
2. int64 B
3. ---
4. int64 Sum
```

在上面的例子中 A 和 B 是请求，sum 是响应。

打开之前，我们所建立的包：

[html] view plaincopy

```
1. cd ~/catkin_ws/src/beginner_tutorials
```

在里面创建一个 msg 目录：

[html] view plaincopy

```
1. mkdir msg
```

新建一个 msg 文件，并写入数据：

[html] view plaincopy

```
1. echo "int64 num" > msg/Num.msg
```

这是一个最简单的例子，msg 文件只包括一行，当然也可以在文件中写入更多的数据，打开 package.xml:

[html] view plaincopy

```
1. vim package.xml
```

在里面添加下面两行：

[html] view plaincopy

```
1. <build_depend>message_generation</build_depend>
2. <run_depend>message_runtime</run_depend>
```

之后保存退出。打开 CMakeLists.txt：

[html] view plaincopy

```
1. vim CMakeLists.txt
```

添加 message_generation 到如下代码片，添加后结果如下：

[html] view plaincopy

```
1. # Do not just add this line to your CMakeLists.txt, modify the ex
   isting line
2. find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs mes
   sage_generation)
```

添加 CATKIN_DEPENDS message_runtime，如下：

[html] view plaincopy

```
1. catkin_package(
2.   ...
3.   CATKIN_DEPENDS message_runtime ...
4.   ...)
```

找到下面的代码片：

[html] view plaincopy

```
1. # add_message_files(
2. #   FILES
3. #   Message1.msg
4. #   Message2.msg
5. # )
```

将它修改为：

[html] view plaincopy

```
1. add_message_files(
2.   FILES
3.   Num.msg
4. )
```

确保文件中有如下代码：

[html] view plaincopy

```
1. generate_messages()
```

我们可以使用 rosmsg show 命令来查看，消息的详细类型，rosmsg show 的使用方法如下：

[html] view plaincopy

```
1. rosmsg show [message type]
```

查看 Num 消息类型如下：

[html] view plaincopy

```
1. rosmsg show Num
```

你将会看到：

[html] view plaincopy

```
[beginner_tutorials/Num]:
int64 num
```

下面介绍如何使用 srv，打开一个终端，在里面输入：

[html] view plaincopy

```
roscd beginner_tutorials
```

[html] view plaincopy

```
mkdir srv
```

除了可以手工创建一个 srv 包以外，我们还可以从其他的包中复制，这时候 roscp 是一个非常有用的命令，它的使用方法如下：

[html] view plaincopy

```
roscp [package_name] [file_to_copy_path] [copy_path]
```

我们从 rospy_tutorials 这个包中复制，命令如下：

[html] view plaincopy

```
roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
```

接下来就是配置 CMakeLists.txt 文件，打开 CMakeLists.txt:

[html] view plaincopy

```
vim CMakeLists.txt
```

在里面添加 message_generation，我们之前已经添加过了，如下：

[html] view plaincopy

```
# Do not just add this line to your CMakeLists.txt, modify the existing line
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs <span style="color:#FFFF00">message_generation</span>)
```

找到下面的代码片段：

[html] view plaincopy

```
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )
```

将其修改为：

[html] view plaincopy

```
add_service_files(
  FILES
  AddTwoInts.srv
)
```

保存退出。

使用 rossrv show 可以查看我们刚刚建立的服务类型，它的使用方法和 rosmsg show 相似，用法如下：

[html] view plaincopy

```
rossrv show <service type>
```

例如：

[html] view plaincopy

```
rossrv show beginner_tutorials/AddTwoInts
```

你将会看到返回的结果如下：

[html] view plaincopy

```
int64 a
int64 b
---
int64 sum
```

接下来我们看一下如何将上述的文件生成为 ros 支持的语言代码，打开 CMakeLists.txt：

[html] view plaincopy

```
1. vim CMakeLists.txt
```

找到下面这部分代码：

[html] view plaincopy

```
1. # generate_messages(
2. #   DEPENDENCIES
3. # #  std_msgs  # Or other packages containing msgs
4. # )
```

将起修改为如下：

[html] view plaincopy

```
1. generate_messages(
2.   DEPENDENCIES
3.   std_msgs
4. )
```

将我们之前的 generate_messages()去掉，保存退出。

现在重新构建一下这个包：

[html] view plaincopy

```
1. cd ../..
```

[html] view plaincopy

```
1. catkin_make
```

[html] view plaincopy

```
1. cd -
```

所有在 msg 目录下的.msg 文件都会产生 ros 所支持的语言的源文件。C++消息的头文件产生在：

[html] view plaincopy

```
1. ~/catkin_ws/devel/include/beginner_tutorials/
```

python 的脚本产生在：

[html] view plaincopy

```
1. ~/catkin_ws/devel/lib/python2.7/dist-packages/beginner_tutorials/
   msg
```

lisp 文件产生在：

[html] view plaincopy

```
1. ~/catkin_ws/devel/share/common-lisp/ros/beginner_tutorials/msg/
```

第六课　C++写简单的发布者和接收

节点是一个可执行程序，它连接到了 ROS 的网络系统中。我们将会创建一个发布者，也就是说话者节点，它将会持续的广播一个信息。

改变目录到之前所建立的那个包下：

[html] view plaincopy

```
1. cd ~/catkin_ws/src/beginner_tutorials
```

在 beginner_tutorials 包下面建立一个 src 文件夹：

[html] view plaincopy

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

创建文件 src/talker.cpp：

[html] view plaincopy

```
1. vim src/talker.cpp
```

将下面的内容复制进去：

[html] view plaincopy

```
1. #include "ros/ros.h"
2. #include "std_msgs/String.h"
3.
4. #include <sstream>
5.
6. /**
7.  * This tutorial demonstrates simple sending of messages over the
    ROS system.
8.  */
9. int main(int argc, char **argv)
10.{
11.  /**
12.   * The ros::init() function needs to see argc and argv so that
   it can perform
13.   * any ROS arguments and name remapping that were provided at t
   he command line. For programmatic
14.   * remappings you can use a different version of init() which t
   akes remappings
15.   * directly, but for most command-line programs, passing argc a
   nd argv is the easiest
```

```
16.    * way to do it.  The third argument to init() is the name of t
   he node.
17.    *
18.    * You must call one of the versions of ros::init() before usin
   g any other
19.    * part of the ROS system.
20.    */
21.   ros::init(argc, argv, "talker");
22.
23.   /**
24.    * NodeHandle is the main access point to communications with t
   he ROS system.
25.    * The first NodeHandle constructed will fully initialize this
   node, and the last
26.    * NodeHandle destructed will close down the node.
27.    */
28.   ros::NodeHandle n;
29.
30.   /**
31.    * The advertise() function is how you tell ROS that you want t
   o
32.    * publish on a given topic name. This invokes a call to the RO
   S
33.    * master node, which keeps a registry of who is publishing and
    who
34.    * is subscribing. After this advertise() call is made, the mas
   ter
35.    * node will notify anyone who is trying to subscribe to this t
   opic name,
36.    * and they will in turn negotiate a peer-to-peer connection wi
   th this
37.    * node.  advertise() returns a Publisher object which allows y
   ou to
38.    * publish messages on that topic through a call to publish().
    Once
39.    * all copies of the returned Publisher object are destroyed, t
   he topic
40.    * will be automatically unadvertised.
41.    *
42.    * The second parameter to advertise() is the size of the messa
   ge queue
43.    * used for publishing messages.  If messages are published mor
   e quickly
```

```
44.    * than we can send them, the number here specifies how many me
   ssages to
45.    * buffer up before throwing some away.
46.    */
47.   ros::Publisher chatter_pub = n.advertise<std_msgs::String>("cha
   tter", 1000);
48.
49.   ros::Rate loop_rate(10);
50.
51.   /**
52.    * A count of how many messages we have sent. This is used to c
   reate
53.    * a unique string for each message.
54.    */
55.   int count = 0;
56.   while (ros::ok())
57.   {
58.     /**
59.      * This is a message object. You stuff it with data, and then
    publish it.
60.      */
61.     std_msgs::String msg;
62.
63.     std::stringstream ss;
64.     ss << "hello world " << count;
65.     msg.data = ss.str();
66.
67.     ROS_INFO("%s", msg.data.c_str());
68.
69.     /**
70.      * The publish() function is how you send messages. The param
   eter
71.      * is the message object. The type of this object must agree
   with the type
72.      * given as a template parameter to the advertise<>() call, a
   s was done
73.      * in the constructor above.
74.      */
75.     chatter_pub.publish(msg);
76.
77.     ros::spinOnce();
78.
79.     loop_rate.sleep();
80.     ++count;
```

```
81.   }
82.
83.
84.   return 0;
85.}
```

保存退出。

解释一下代码：

[html] view plaincopy

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

ros/ros.h 包括了使用 ROS 系统最基本的头文件。

[html] view plaincopy

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

这条代码包括了 std_msgs/String 消息，它存在于 std_msgs 包中。这是有 std_msgs 中的 String.msg 文件自动产生的。

[html] view plaincopy

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

初始化 ROS，它允许 ROS 通过命令行重新命名，现在还不太重要。这里也是我们确切说明节点名字的地方，在运行的系统中，节点的名字必须唯一。

[html] view plaincopy

```
1. ros::NodeHandle n;
```

为处理的节点创建了一个句柄，第一个创建的节点句柄将会初始化这个节点，最后一个销毁的节点将会释放节点所使用的所有资源。

[html] view plaincopy

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

告诉主机，我们将会在一个名字为 chatter 的话题上发布一个 std_msgs/String 类型的消息，这就使得主机告诉了所有订阅了 chatter 话题的节点，我们将在这个话题上发布数据。第二

个参数是发布队列的大小，它的作用是缓冲。当我们发布消息很快的时候，它将能缓冲 1000 条信息。如果慢了的话就会覆盖前面的信息。

NodeHandle::advertise()将会返回 ros::Publisher 对象，该对象有两个作用，首先是它包括一个 publish()方法可以在制定的话题上发布消息，其次，当超出范围之外的时候就会自动的处理。

[html] view plaincopy

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

一个 ros::Rate 对象允许你制定循环的频率。它将会记录从上次调用 Rate::sleep()到现在为止的时间，并且休眠正确的时间。在这个例子中，设置的频率为 10hz。

[html] view plaincopy

```
1. int count = 0;
2. while (ros::ok())
3. {
```

默认情况下，roscpp 将会安装一个 SIGINT 监听，它使当 Ctrl-C 按下时，ros::ok()将会返回 false。

ros::ok()在以下几种情况下也会返回 false：（1）按下 Ctrl-C 时（2）我们被一个同名同姓的节点从网络中踢出（3）ros::shutdown()被应用程序的另一部分调用（4）所有的 ros::NodeHandles 都被销毁了。一旦 ros::ok()返回 false，所有的 ROS 调用都会失败。

[html] view plaincopy

```
1. std_msgs::String msg;
2.
3. std::stringstream ss;
4. ss << "hello world " << count;
5. msg.data = ss.str();
```

我们使用 message-adapted 类在 ROS 中广播信息，这个类一般是从 msg 文件中产生的。我们现在使用的是标准的字符串消息，它只有一个 data 数据成员，当然更复杂的消息也是可以的。

[html] view plaincopy

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

现在我们向话题 chatter 发布消息。

[html] view plaincopy

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

ROS_INFO 是 cout 和 printf 的替代品。

[html] view plaincopy

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

在这个简单的程序中调用 ros::spinOnce();是不必要的，因为我们没有收到任何的回调信息。然而如果你为这个应用程序添加一个订阅者，并且在这里没有调用 ros::spinOnce()，你的回调函数将不会被调用。所以这是一个良好的风格。

[html] view plaincopy

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

休眠一下，使程序满足前面所设置的 10hz 的要求。

下面总结一下创建一个发布者节点的步骤：（1）初始化 ROS 系统（2）告诉主机我们将要在 chatter 话题上发布 std_msgs/String 类型的消息（3）循环每秒发送 10 次消息。
打开一个终端，进入到 beginner_tutorials 包下面：

[html] view plaincopy

```
1. cd ~/catkin_ws/src/beginner_tutorials
```

编辑文件 src/listener.cpp：

[html] view plaincopy

```
1. vim src/listener.cpp
```

将下面的代码复制到文件中：

[html] view plaincopy

```
1. #include "ros/ros.h"
2. #include "std_msgs/String.h"
3.
4. /**
5.  * This tutorial demonstrates simple receipt of messages over the
    ROS system.
6.  */
7. void chatterCallback(const std_msgs::String::ConstPtr& msg)
8. {
9.   ROS_INFO("I heard: [%s]", msg->data.c_str());
```

```
10.}
11.
12.int main(int argc, char **argv)
13.{
14.  /**
15.   * The ros::init() function needs to see argc and argv so that it can perform
16.   * any ROS arguments and name remapping that were provided at the command line. For programmatic
17.   * remappings you can use a different version of init() which takes remappings
18.   * directly, but for most command-line programs, passing argc and argv is the easiest
19.   * way to do it.  The third argument to init() is the name of the node.
20.   *
21.   * You must call one of the versions of ros::init() before using any other
22.   * part of the ROS system.
23.   */
24.  ros::init(argc, argv, "listener");
25.
26.  /**
27.   * NodeHandle is the main access point to communications with the ROS system.
28.   * The first NodeHandle constructed will fully initialize this node, and the last
29.   * NodeHandle destructed will close down the node.
30.   */
31.  ros::NodeHandle n;
32.
33.  /**
34.   * The subscribe() call is how you tell ROS that you want to receive messages
35.   * on a given topic.  This invokes a call to the ROS
36.   * master node, which keeps a registry of who is publishing and who
37.   * is subscribing.  Messages are passed to a callback function, here
38.   * called chatterCallback.  subscribe() returns a Subscriber object that you
39.   * must hold on to until you want to unsubscribe.  When all copies of the Subscriber
```

```
40.   * object go out of scope, this callback will automatically be
   unsubscribed from
41.   * this topic.
42.   *
43.   * The second parameter to the subscribe() function is the size
    of the message
44.   * queue.  If messages are arriving faster than they are being
   processed, this
45.   * is the number of messages that will be buffered up before be
   ginning to throw
46.   * away the oldest ones.
47.   */
48.  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallb
   ack);
49.
50.  /**
51.   * ros::spin() will enter a loop, pumping callbacks.  With this
    version, all
52.   * callbacks will be called from within this thread (the main o
   ne).  ros::spin()
53.   * will exit when Ctrl-C is pressed, or the node is shutdown by
    the master.
54.   */
55.  ros::spin();
56.
57.  return 0;
58.}
```

保存退出。下面看一下代码的解释，

[html] view plaincopy

```
1. void chatterCallback(const std_msgs::String::ConstPtr& msg)
2. {
3.   ROS_INFO("I heard: [%s]", msg->data.c_str());
4. }
```

当一个消息到达 chatter 话题时，这个回调函数将会被调用。

[html] view plaincopy

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

订阅 chatter 话题，当一个新的消息到达时，ROS 将会调用 chatterCallback()函数。第二个参数是对列的长度，如果我们处理消息的速度不够快，会将收到的消息缓冲下来，一共可以缓冲 1000 条消息，满 1000 之后，后面的到达的消息将会覆盖前面的消息。NodeHandle::subscribe()将会返回一个 ros::Subscriber 类型的对象，当订阅对象被销毁以后，它将会自动从 chatter 话题上撤销。

[html] view plaincopy

```
1. ros::spin();
```

ros::spin()进入了一个循环，可以尽快的调用消息的回调函数。不要担心，如果它没有什么事情可做时，它也不会浪费太多的 CPU。当 ros::ok()返回 false 时，ros::spin()将会退出。这就意味着，当 ros::shutdown()被调用，或按下 CTRL+C 等情况，都可以退出。下面总结一下写一个订阅者的步骤：（1）初始化 ROS 系统（2）订阅 chatter 话题（3）Spin，等待消息的到来（4）当一个消息到达时，chatterCallback()函数被调用。

下面看一下如何构建节点。这时候你的 CMakeLists.txt 看起来应该是下面这个样子，包括前面所做的修改，注释部分可以除去：

[html] view plaincopy

```
1. cmake_minimum_required(VERSION 2.8.3)
2. project(beginner_tutorials)
3.
4. ## Find catkin and any catkin packages
5. find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs gen
   msg)
6.
7. ## Declare ROS messages and services
8. add_message_files(DIRECTORY msg FILES Num.msg)
9. add_service_files(DIRECTORY srv 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()
```

将下面几行代码添加到 CMakeLists.txt 的最后。最终你的 CMakeLists.txt 文件看起来样该是下面这个样子：

[html] view plaincopy

```
1. cmake_minimum_required(VERSION 2.8.3)
```

```
project(beginner_tutorials)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs gen
msg)

## Declare ROS messages and services
add_message_files(FILES Num.msg)
add_service_files(FILES AddTwoInts.srv)

## Generate added messages and services
generate_messages(DEPENDENCIES std_msgs)

## Declare a catkin package
catkin_package()

## Build talker and listener
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_cp
p)
```

这将会创建两个可执行文件，talker 和 listener。它们将会产生在 ~/catkin_ws/devel/lib/share/<package name>目录下，下面开始构建，在你的工作空间根目录下输入：

[html] view plaincopy

```
catkin_make
```

在前面的两篇博客中我们用 C++在 ROS 中创建了一个发布者和接收者，并使用 catkin_make 构建了新的节点，下面就需要验证一下，我们写的是否正确。

首先运行 roscore

[html] view plaincopy

```
roscore
```

打开一个新的终端在里面运行 talker：

[html] view plaincopy

```
1. rosrun beginner_tutorials talker
```

看到的运行结果如下：

[html] view plaincopy

```
1. [INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
2. [INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
3. [INFO] [WallTime: 1314931833.778937] hello world 1314931833.78
4. [INFO] [WallTime: 1314931834.782059] hello world 1314931834.78
5. [INFO] [WallTime: 1314931835.784853] hello world 1314931835.78
6. [INFO] [WallTime: 1314931836.788106] hello world 1314931836.79
```

打开一个新的终端在里面运行 listener：

[html] view plaincopy

```
1. rosrun beginner_tutorials listener
```

第七课　C++写简单的服务和客户端

我们将创建一个服务器节点 add_two_ints_server，它将会收到两个整数，并且返回它们的和。切换目录到之前建立的 beginner_tutorials 包下：

[html] view plaincopy

```
1. cd ~/catkin_ws/src/beginner_tutorials
```

编辑 src/add_two_ints_server.cpp 文件：

[html] view plaincopy

```
1. vim src/add_two_ints_server.cpp
```

将下面的代码复制到文件中，保存后退出：

[html] view plaincopy

```
1. #include "ros/ros.h"
2. #include "beginner_tutorials/AddTwoInts.h"
3.
4. bool add(beginner_tutorials::AddTwoInts::Request  &req,
5.          beginner_tutorials::AddTwoInts::Response &res)
6. {
7.   res.sum = req.a + req.b;
8.   ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)re
   q.b);
9.   ROS_INFO("sending back response: [%ld]", (long int)res.sum);
10.  return true;
11.}
12.
13.int main(int argc, char **argv)
14.{
15.  ros::init(argc, argv, "add_two_ints_server");
16.  ros::NodeHandle n;
17.
18.  ros::ServiceServer service = n.advertiseService("add_two_ints",
    add);
19.  ROS_INFO("Ready to add two ints.");
20.  ros::spin();
21.
22.  return 0;
23.}
```

下面解释一下代码：

[html] view plaincopy

```
1. #include "ros/ros.h"
2. #include "beginner_tutorials/AddTwoInts.h"
```

beginner_tutorials/AddTwoInts.h 是由之前我们创建的 srv 文件自动产生的头文件。

[html] view plaincopy

```
1. bool add(beginner_tutorials::AddTwoInts::Request  &req,
2.          beginner_tutorials::AddTwoInts::Response &res)
```

这个函数为两个整数相加提供了服务，它使用了在 srv 文件中定义的请求和响应类型，并且返回一个布尔类型的值。

[html] view plaincopy

```
1. {
2.   res.sum = req.a + req.b;
3.   ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)re
   q.b);
4.   ROS_INFO("sending back response: [%ld]", (long int)res.sum);
5.   return true;
6. }
```

在这里两个整数相加并存储在 response 中，然后输出了一些关于 request 和 response 的信息，最后返回一个真值。

[html] view plaincopy

```
1. ros::ServiceServer service = n.advertiseService("add_two_ints", a
   dd);
```

这行代码创建了一个服务。

接下来创建一个客户端，打开一个终端输入：

[html] view plaincopy

```
1. vim src/add_two_ints_client.cpp
```

将下面的代码复制进去，保存后退出：

[html] view plaincopy

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_client");
  if (argc != 3)
  {
    ROS_INFO("usage: add_two_ints_client X Y");
    return 1;
  }

  ros::NodeHandle n;
  ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
  beginner_tutorials::AddTwoInts srv;
  srv.request.a = atoll(argv[1]);
  srv.request.b = atoll(argv[2]);
  if (client.call(srv))
  {
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  }
  else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }

  return 0;
}
```

下面看一下代码解释：

[html] view plaincopy

```
ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
```

这行代码为 add_two_ints 创建了一个客户端，ros::ServiceClient 对象之后被用来调用服务。

[html] view plaincopy

```
1. beginner_tutorials::AddTwoInts srv;
2. srv.request.a = atoll(argv[1]);
3. srv.request.b = atoll(argv[2]);
```

我们示例了一个自动产生的服务类，并且在它的请求成员中分配值。一个服务类包括了两个成员，请求和服务。它同样包括了两个类的定义，请求和响应。

[html] view plaincopy

```
1. if (client.call(srv))
```

这句代码才开始真正调用了服务，因为服务调用被阻塞，当调用完成后就立即返回。如果服务调用成功，call()将会返回真及 srv 中的值。否则，call()将会返回假及 srv 中的值。

打开~/catkin_ws/src/beginner_tutorials/CMakeLists.txt，把下面代码复制到文件末尾，保存，退出：

[html] view plaincopy

```
1. add_executable(add_two_ints_server src/add_two_ints_server.cpp)
2. target_link_libraries(add_two_ints_server ${catkin_LIBRARIES})
3. add_dependencies(add_two_ints_server beginner_tutorials_gencpp)
4.
5. add_executable(add_two_ints_client src/add_two_ints_client.cpp)
6. target_link_libraries(add_two_ints_client ${catkin_LIBRARIES})
7. add_dependencies(add_two_ints_client beginner_tutorials_gencpp)
```

这将会创建两个可执行文件，add_two_ints_server 和 add_two_ints_client，默认将会产生在~/catkin_ws/devel/lib/share/<package name>目录下，你可以直接运行它们，也可以通过 rosrun 来运行。

现在开始构建：

[html] view plaincopy

```
1. cd ~/catkin_ws
```

[html] view plaincopy

```
1. catkin_make
```

ok 了，一切准备就绪，现在就开始检验一下程序是否正确吧。打开一个终端，运行服务器：

[html] view plaincopy

```
1. rosrun beginner_tutorials add_two_ints_server
```

会出现一下提示：

[html] view plaincopy

```
1. Ready to add two ints.
```

再打开一个终端，运行客户端：

[html] view plaincopy

```
1. rosrun beginner_tutorials add_two_ints_client 1 3
```

我们可以看到返回的结果如下：

[html] view plaincopy

```
1. Requesting 1+3
2. 1 + 3 = 4
```

说明你已经成功了。

小罗老师最后祝大家学习有成，多多探讨

dreamluo

huazailuo@126.com