



ROS (Robot Operating System)

培训教材

北京维尔玛科技有限公司

北京维尔玛科技有限公司

2017 年 9 月 9 日



北京维尔玛科技有限公司

公司简介

尊敬的用户：

您好,非常感谢您关注和支持北京维尔玛公司，北京维尔玛将与您一起进行技术创新，让人工智能走进千家万户！

北京维尔玛科技有限公司成立于 2015 年，是一家专业从事机器人整机、配件代理销售以及机器人集成解决方案专业化定制的高科技公司，公司致力于将世界上最先进的机器人产品引入中国，帮助中国提升机器人技术研发实力。

公司主营业务分成三块，一是代理销售，主要代理品牌有 Kinova 轻量型仿生机械臂、Clearpath 移动平台、Righthand 三指柔性灵巧手、ARMDillo 智能复合机器人等，所有产品支持 ROS 机器人操作系统。

二是基于代理的机械臂、移动平台、传感器等部件，为科研单位、研究院所定制基于 ROS 的科研教育平台，配套相应教材。不定期分享 ROS 学习、开发经验。

三是基于机械臂关节模块，为用户量身定制不同构型、不同负载的机械臂，机械臂可运行在 Windows 下或 ROS 下。



目 录

北京维尔玛科技有限公司 公司简介	2
1. ROS 安装---PC	6
1.1. 安装 ubuntu 操作系统	6
1.2. 配置 Ubuntu 软件库	6
1.3. 添加 ROS 软件库到 source.list 文件中	7
1.4. 设置密钥	7
1.5. 更新软件包列表	7
1.6. 安装 ROS 软件包	8
1.7. 初始化 rosdep	8
1.8. 配置环境	8
1.9. 安装 rosinstall	9
1.10. 测试安装	9
2. ROS 架构和概念	10
2.1 理解 ROS 文件系统级	10
2.1.1 工作空间	11
2.1.2 功能包	12
2.1.3 功能包集	14
2.1.4 消息	14
2.1.5 服务	15
2.2 理解 ROS 计算图级	16
2.2.1 节点管理器	16



2.2.2 节点	16
2.2.3 主题	16
2.2.4 消息	17
2.2.5 服务	17
2.2.6 参数服务器	18
2.2.7 消息记录包	18
3. 创建 ROS 功能包	20
3.1 创建工作空间	20
3.1.1 新建工作空间	20
3.1.2 初始化新工作空间	20
3.1.3 编译工作空间	20
3.1.4 加入新工作空间的路径	20
3.2 创建功能包	20
3.3 新建节点文件	21
3.4 修改 CMakeLists.txt 文件	22
3.5 编译节点文件	23
3.6 运行节点文件	23
4. 基于 QT IDE 练习 ROS 编程	24
4.1. 安装 QT IDE	24
4.1.1. 安装 QtCreator	24
4.1.2. 设置快捷方式	25
4.1.3. 安装 ros_qtc_plugin 插件	26



4.2. 利用 ros_qtc_plugin 编程	27
4.2.1. 新建 Project	27
4.2.2. 新建功能包 (Package) :	32
4.2.3. 采用标准消息类型编写第一个 Publisher 节点	35
4.2.4. 采用标准消息类型编写第一个 Subscriber 节点	40
4.2.5. 编写第一个非标准消息文件	43
4.2.6. 采用非标准消息文件编写 Publisher 节点和 Subscriber 节点	48
4.2.7. 编写第一个服务文件 (.srv 文件)	50
4.2.8. 采用新建的服务文件编写服务器节点和客户端节点 :	54



1. ROS 安装---PC

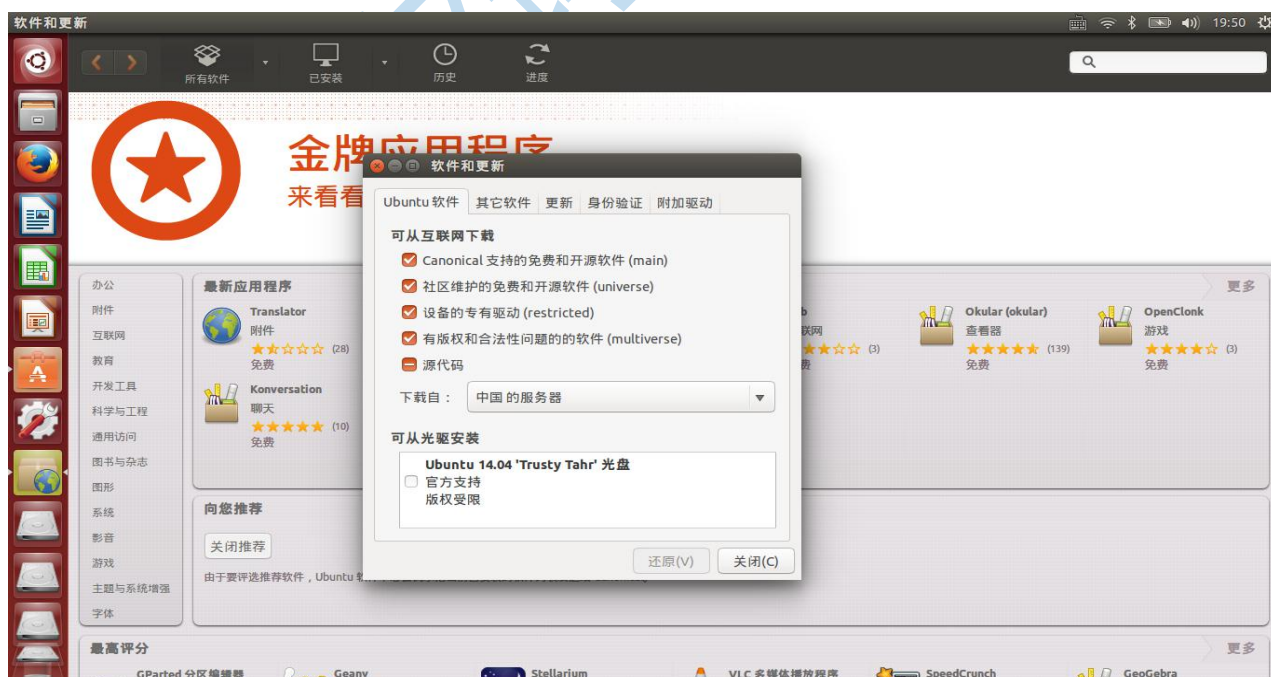
使用 ROS 之前，我们必须首先确认已经在计算机上成功安装了 ROS 软件，本节讲述 ROS (indigo 版本) 的安装过程。

1.1. 安装 ubuntu 操作系统

ROS indigo 对应的 unbutu 系统版本为 14.04，请自行下载安装。

1.2. 配置 Ubuntu 软件库

开始安装之前需要首先配置软件库，打开桌面左侧的 Ubuntu 软件中心 (Ubuntu Software Center)，单击 Edit|Software 标签页，保证各个选项与下图中一致：





1.3. 添加 ROS 软件库到 source.list 文件中

使用超级用户（Ubuntu 下使用 `sudo` 命令得到超级用户权限）添加 ROS 软件源，命令行输入：

```
sudo sh -c ' . /etc/lsb-release && echo "deb  
http://ros.exbot.net/rospackage/ros/ubuntu/ $DISTRIB_CODENAME main" >  
/etc/apt/sources.list.d/ros-latest.list'
```

一旦添加了正确的软件库，操作系统就知道在哪里下载程序，并根据命令自动安装软件。

1.4. 设置密钥

这一步是为了确认原始的代码是正确且没有被修改过的。命令行输入：

```
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv-key  
421C365BD9FF1F717815A3895523BAEEB01FA116
```

1.5. 更新软件包列表

一旦配置完软件版本仓库（`repositories`）之后，可以用下列命令得到最新的可用软件包列表：

```
sudo apt-get update
```

需要注意的是，这会更新你系统中所有的软件版本仓库，而不仅仅是新添加的 ROS 库。



1.6. 安装 ROS 软件包

现在开始安装 ROS 软件。最简单的方法是将 ROS 的核心系统进行完整安装：

```
sudo apt-get install ros-indigo-desktop-full
```

如果磁盘空间足够——至少要几个 GB——上述安装方案将是最好的选择。

1.7. 初始化 rosdep

安装完 ROS 包后，需要执行下面的命令：

```
sudo rosdep init
```

```
rosdep update
```

这个初始化步骤是一次性的，一旦 ROS 正常工作，多数用户不再需要访问 rosdep，该命令将在你的根目录下保存一些文件，文件夹名为 .ros。

1.8. 配置环境

ROS 要依据一些环境变量来定位文件。设置这些环境变量，你需要使用以下命令：

```
source /opt/ros/indigo/setup.bash
```

然后，用下列命令确认环境变量已经设置正确：

```
export | grep ROS
```

如果一切工作正常，你应该看到了一组值（显示 ROS_DISTRO 和 ROS_PACKAGE_PATH 等环境变量的值）。如果 setup.bash 尚未运行，则此命令的输出通常为空。

上面所述的设置环境变量的方法仅适用于当前的 shell 端口。用户在每次启动一个新的 shell，且要在这个 shell 中运行 ROS 时，就必须重新运行上



述 source 命令，ROS 才能顺利工作。如果想要配置自己的账户，使其每启动一个新的 shell 时，都自动运行脚本 setup.bash，可编辑账户根目录中的文件.bashrc，在该文件最下面一行添加前文的 source 命令即可。也可在终端输入：

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

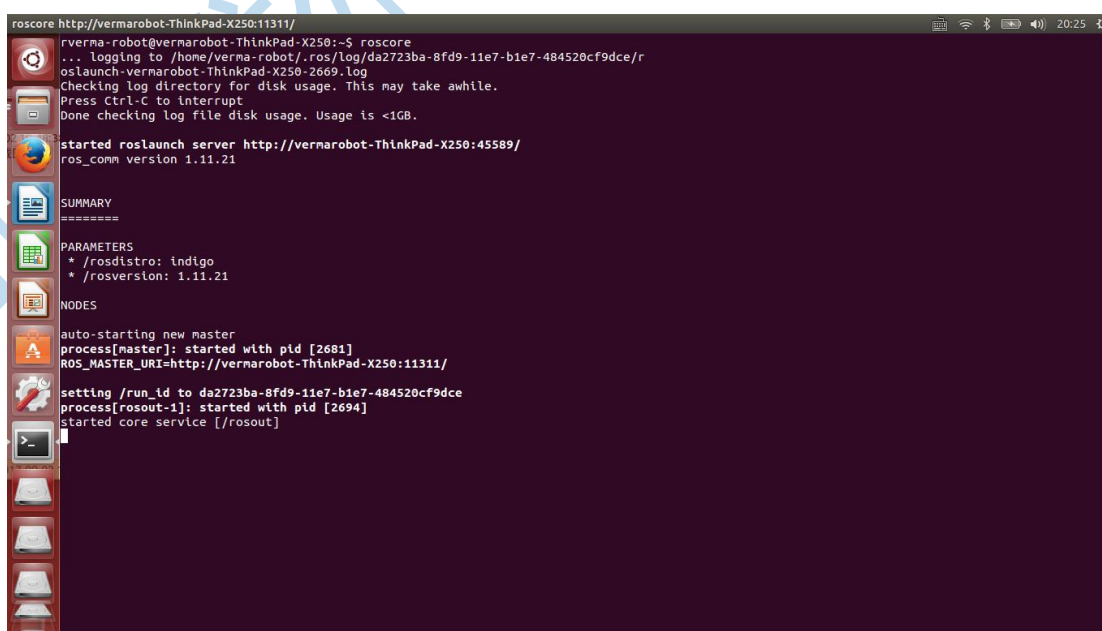
1.9. 安装 rosinstall

安装一个命令工具，以帮助我们使用一条命令安装其他功能包。命令：

```
sudo apt-get install python-roscpp
```

1.10. 测试安装

命令：roscore，若返回界面如下图所示，证明安装正确：





2. ROS 架构和概念

ROS 的架构经过设计被划分成了三个层级：

- 文件系统级--文件系统级体现 ROS 的内部构成、文件夹结构以及核心文件。
- 计算图级--进程与系统之间的通信
- 社区级

2.1 理解 ROS 文件系统级

在 ROS 中，所有的软件都以功能包的形式组织起来，ROS 软件包是一组用于实现特定功能的相关文件的集合，包括可执行文件和其他支持文件。

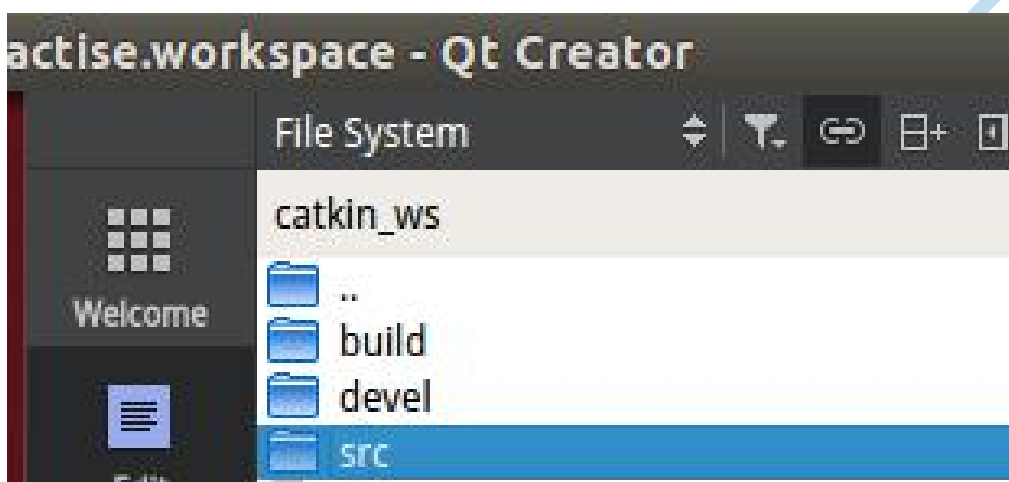
- 功能包：在 ROS 中，所有的软件都以功能包的形式组织起来，ROS 软件包是一组用于实现特定功能的相关文件的集合，包括可执行文件和其他支持文件。比如说，`turtlesim_node` 和 `turtle_teleop_key` 都属于 `turtlesim` 包。ROS 提供多个命令用于与已安装的软件包进行交互。
- 功能包清单：功能包清单提供关于功能包、许可证、依赖关系、编译标志等的信息。功能包清单由一个名为 `package.xml` 的文件管理。
- 功能包集：功能包集由几个功能包组成。ROS 中存在大量不同用途的功能包集，例如导航功能包集。
- 功能包集清单：功能包集清单类似于普通的功能包清单，
- 消息类型：消息是一个进程发送到其他进程的信息。ROS 有很多标准类型的消息。
- 服务类型：服务为每个进程提供定义请求和响应的数据结构。



2.1.1 工作空间

工作空间是一个文件夹，包含源文件空间（Source space）、编译空间（build space）和开发空间（Development（devel）space）。

如下图中，工作空间为 catkin_ws，源文件空间：src；编译空间：build；开发空间：devel；



- 源文件空间：在源空间（src 文件夹）中放置了功能包、项目、复制的包等。在这个空间中，最重要的一个文件是 CMakeLists.txt，这个文件是通过 catkin_init_workspace 命令创建的。

指令为：

```
cd workspace/src
```

```
catkin_init_workspace
```

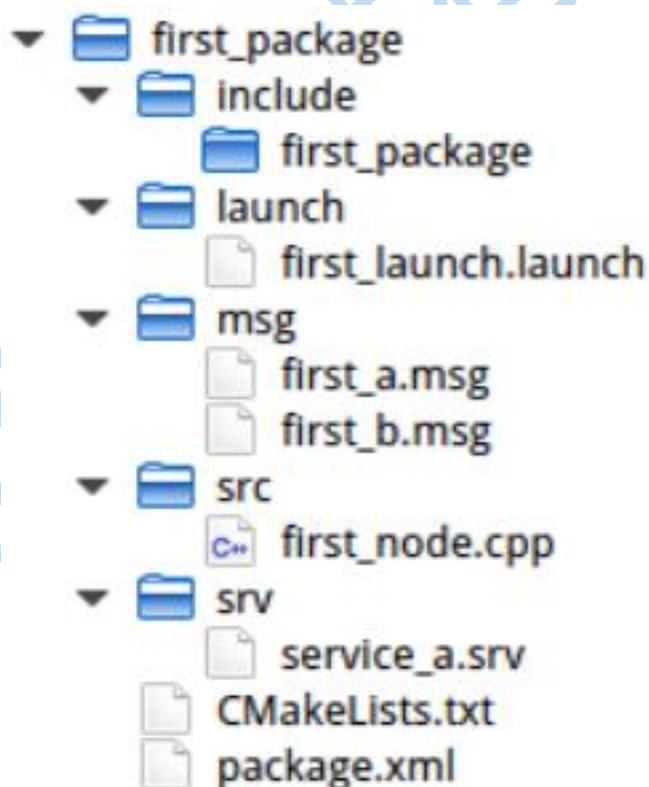
- 编译空间：在 build 文件夹里，catkin 为功能包保存缓存信息、配置和其他中间文件。
- 开发空间：devel 文件夹保存编译后的程序，这些是无需安装就能用来测试的程序。



2.1.2 功能包

功能包指的是一种特定结构的文件和文件夹组合。通常包含如下结构：

- include/package_name: 此目录包含了需要的库的头文件
- msg/：非标准消息文件的存放目录
- src/：程序源文件存放目录
- srv/：服务类型文件的存放目录
- CmakeLists.txt：CMake 的生成文件
- package.xml：功能包清单文件
- scripts/：脚本文件存放目录





为了创建、修改和使用功能包，ROS 为我们提供了一些工具：

- rospack：此命令通常用于获取信息或在系统中查找包。此工具的参数如下：

`rospack list`：列出所安装的 ROS 功能包

`rospack find package_name`：查找功能包所在路径

- catkin_create_pkg：此命令用于创建一个新的功能包。常用格式为：

`catkin_create_pkg new_package_name depend1 depend2.....`

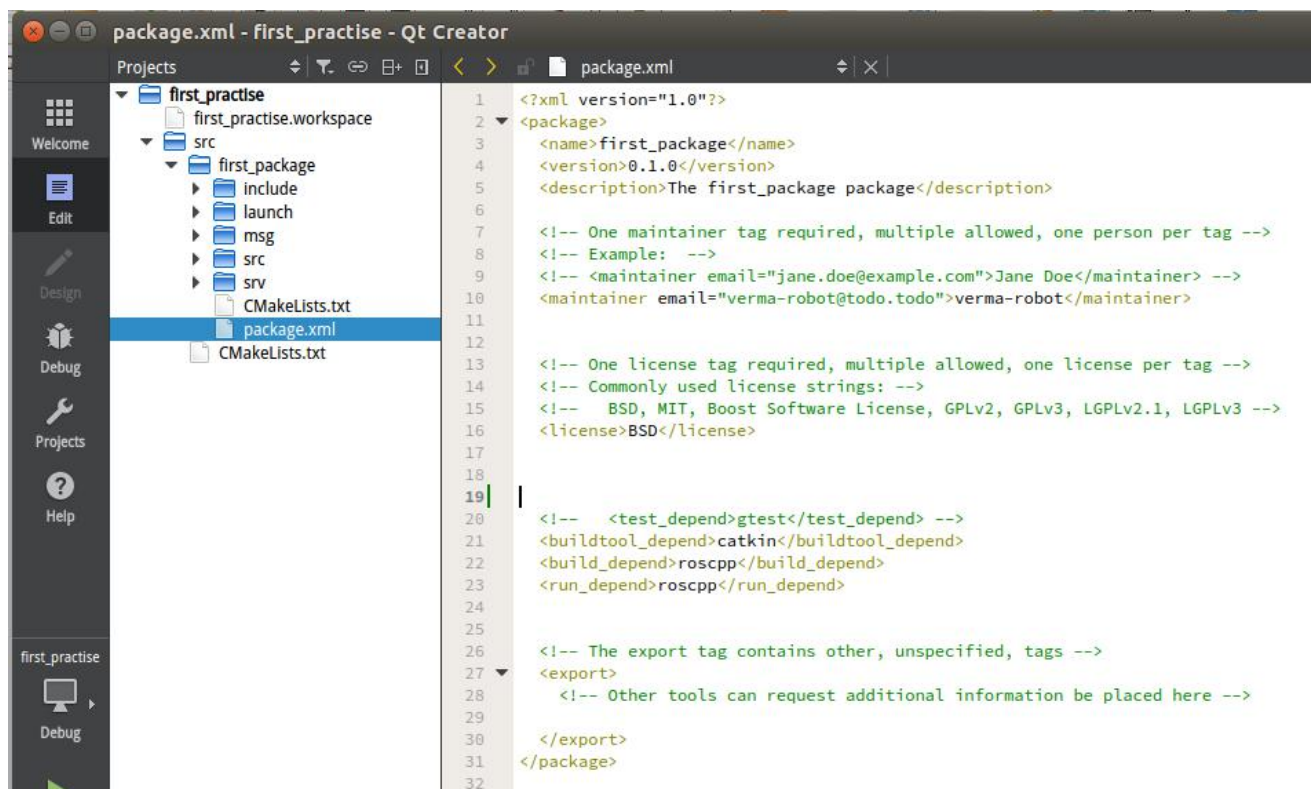
- catkin_make：此命令用于编译工作空间,命令：

`cd workspace`

`catkin_make`

- rosdep：此命令用于安装功能包的系统依赖项
- rqt_dep：此命令用于查看包的依赖关系图

文件 package.xml 必须在功能包内，他用来说明功能包相关的各类信息。在 package.xml 中有两个典型标志：<build_depend>和<run_depend>，<build_depend>标记当前功能包安装之前必须先安装哪些功能包，<run_depend>标记运行当前功能包中的代码需要哪些功能包。

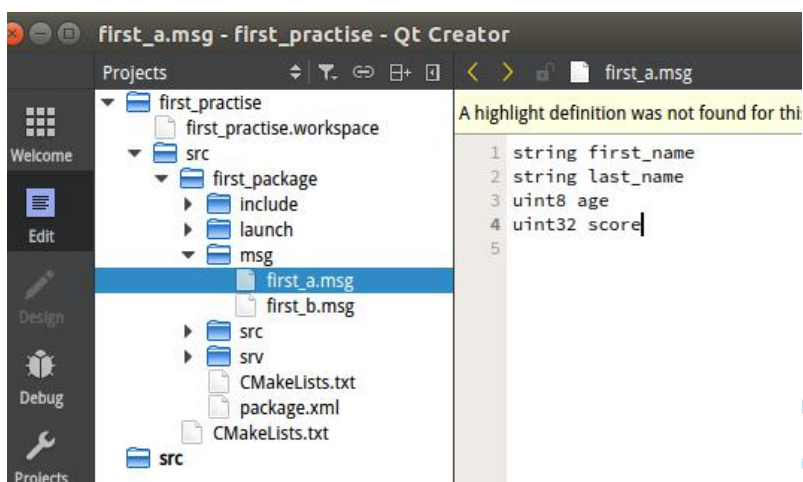


2.1.3 功能包集

功能包集是只有一个文件的特殊的包，这个文件就是 package.xml，功能包集常用的指令为：rosstack find stack_name

2.1.4 消息

ROS 节点之间进行通信所利用的最重要的机制就是消息传递。消息类型主要有两个部分：字段和常量。字段定义了要在消息中传输的数据类型，例如：int、float 等；常量用于定义字段的名称。消息的描述通常存放在 msg/目录下的.msg 文件中，例如，消息 first_a.msg:



ROS 中一种特殊的数据类型是标头类型，他主要用于添加时间、坐标系和序列号等。标头类型允许对消息进行编号。标头类型通常包含以下字段：

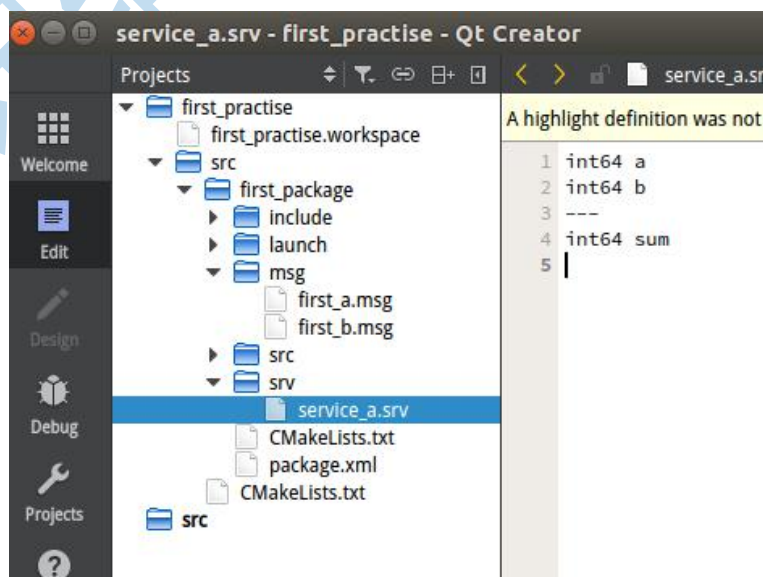
uint32 seq

time stamp

string frame_id

2.1.5 服务

在 ROS 中，服务的描述通常存放在 srv 子目录下的.srv 文件中，服务通常包含请求和响应两部分内容；通常上部分为请求，下部分为响应；例如：





2.2 理解 ROS 计算图级

ROS 计算图级中最基本的概念包括节点、节点管理器、参数服务器、消息、服务、主题和消息记录包。

2.2.1 节点管理器

节点管理器为 ROS 中其他节点提供命名和注册服务。节点管理器的作用是使 ROS 节点之间可以相互查找。一旦这些节点找到了彼此，就能建立点对点的通信。节点通过 `roscore` 指令启动。

2.2.2 节点

节点是各自独立的可执行文件，能够通过主题、服务或参数服务器与其他节点进行通信。节点在系统中必须有唯一的名称。ROS 提供了处理节点和显示节点的工具，如：

- `roscall` : 列出正在运行的 ROS 节点
- `roscall NODE_NAME` : 列出 `NODE_NAME` 节点的节点信息，包括该节点发出和接收的主题名称
- `roscall hostname` : 列出运行在 `hostname` 主机上的节点

ROS 允许在节点启动时更改主题名称、参数。命令为：

```
roscall package_name node name topic_old_name:=topic_new_name  
roscall package_name node name _param_old_name:=new_value
```

2.2.3 主题

主题 (topic) 是节点间用来传输数据的总线。通过主题进行消息传递，发布者和订阅者不需要知道彼此是否存在。一个主题可以有多个订阅者，也可以有多个发布者，但是用不同的节



点发布同样的主题时要慎重。这里请大家认真思考订阅者、发布者和节点、主题相互之间的关系。

ROS 的每个主题都是强类型的，发布到主题上的消息必须与主题的消息类型相匹配。

ROS 通过 rostopic 工具对主题进行操作，常用指令为：

- `rostopic list`：列出当前活动的主题
- `rostopic info topic_name`：列出 topic_name 主题的信息，包括发布者和订阅者的名称、消息类型
- `rostopic echo topic_name`：将在 topic_name 上传输的消息打印到屏幕上
- `rostopic pub topic_name type args`：通过命令行将数据发布到主题
- `rostopic type topic_name`：通过主题传输的消息的数据类型
- `rostopic hz topic-name`：主题发布频率
- `rostopic bw topic-name`：主题的带宽

2.2.4 消息

消息具有简单的数据结构，包括 ROS 提供的标准类型和用户自定义的类型，ROS 中的消息类型遵循以下命名方式：包名/文件名.msg；

ROS 通过 rosmmsg 工具来操作消息，命令如下：

- `rosmmsg list`：列出所有消息
- `rosmmsg show message-type-name`：列出消息类型的数据结构

2.2.5 服务

服务通过 RPC 方式获得应答，服务由用户开发，ROS 不提供标准服务。ROS 通过 rosservice 来操作服务，命令如下：



- `rosservice list` : 列出正在活动的服务
- `rosservice info service_name` : 列出服务的信息, 包括客户端和服务器的名称
- `rosservice type service_name` : 列出服务的数据类型
- `rosservice call service_name arg` : 根据命令行参数调用服务

2.2.6 参数服务器

参数服务器与节点管理器都是在执行 `roscore` 指令时启动的 (`roslaunch` 会自动运行 `roscore`) ; 参数服务器通常用来存储全局变量, 可通过 `rosparam` 指令进行存储、操作, 也可通过图形化的界面来设置参数服务器上的数据;

图形化界面的指令为:

```
roslaunch rqt_reconfigure rqt_reconfigure
```

注意: 有一些参数在图形化界面中是没有显示的, 这个时候我们只能通过 `rosparam` 指令进行操作

`rosparam` 相关的指令如下:

- `rosparam list` : 列出运行在参数服务器上的参数名称
- `rosparam get param_name` : 读取参数的值
- `rosparam set param_name value` : 给参数赋值
- `rosparam dump XXX.yaml` : 将参数服务器上的参数信息存储在 XXX.yaml
- `rosparam load XXX.yaml namespace` : 加载

2.2.7 消息记录包

消息记录包 (`bag`) 是 ROS 创造的一系列组件, 他使用 `.bag` 的形式保存消息、主题、服务



和其他数据信息

北京维尔玛科技有限公司



3. 创建 ROS 功能包

3.1 创建工作空间

我们创建的功能包，应该全部放到一个叫做工作空间的目录中。因此，在开始全部工作之前，我们需要首先创建工作空间。

3.1.1 新建工作空间

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

其中目录 catkin_ws 为我们新建的工作空间

3.1.2 初始化新工作空间

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

```
catkin_init_workspace
```

3.1.3 编译工作空间

```
cd ~/catkin_ws
```

```
catkin_make
```

3.1.4 加入新工作空间的路径

```
echo 'source ~/catkin_ws /devel/setup.bash' >> ~/.bashrc
```

```
source ~/.bashrc
```

3.2 创建功能包



使用 `catkin_create_pkg` 命令来创建一个名为 “first_example” 的新程序包，这个程序包依赖于 `std_msgs`、`roscpp` 功能包，命令如下：

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

```
catkin_create_pkg first_example std_msgs roscpp
```

备注：`catkin_create_pkg` 命令的使用格式为：`catkin_create_pkg [新建程序包的名称] [程序包依赖包 1] [程序包依赖包 2] ...`

这个命令创建了一个功能包（文件夹 `first_example`），并在 `first_example` 文件夹下创建了两个文件夹和两个配置文件：`package.xml` 和 `CMakeLists.txt`；新建的 `src` 文件夹是我们写节点程序的位置；`package.xml` 是功能包清单文件，记载了功能包的基本信息和依赖关系；`CMakeLists.txt` 是一个 Cmake 的脚本文件，这个文件包含了一系列的编译指令，包括应该生成哪种可执行文件，需要哪些源文件，以及在哪里可以找到所需的头文件和链接库；

`package.xml` 文件和 `CMakeLists.txt` 的内容值得认真研读，有兴趣的同学可仔细阅读并推敲其中的每一行代码（包括注释的内容）

3.3 新建节点文件

在 `first_example/src` 目录下新建节点文件 `first_node.cpp`，并输入代码如下图所示：

该节点文件只有简单的几行：



```
first_node.cpp
#include <ros/ros.h>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "first_node");
    ros::NodeHandle nh;

    ROS_INFO("Hello world!");
}
```



`#include<ros/ros.h>`：所要包含的头文件；该节点文件中所要调用的 ROS 库函数、类、工作空间等都是在这个头文件中声明、定义的；`ros/ros.h` 的具体内容可在 [/opt/ros/indigo/include/ros/ros.h](#) 中找到，可以发现，`ros/ros.h` 文件又 `include` 了很多其他头文件，其中就包括 `init.h`、`NodeHandle.h` 文件。

`ros::init(int argc,argv," first_node")`：初始化节点并设置节点名称为 `first_node`；ROS 中节点的名称必须是唯一的；`ros::init()` 函数调用的是域名 `ros` 下的 `init()` 函数，该函数的定义在 [/opt/ros/indigo/include/ros/init.h](#) 中

`ros::NodeHandle nh`：实例化了一个 `ros::NodeHandle` 类；`ros::NodeHandle` 是域名 `ros` 下的 `NodeHandle` 类，该类的定义在 `/opt/ros/indigo/include/ros/NodeHandle.h` 中，感兴趣的同学可以研究下 `NodeHandle.h` 都定义了哪些函数和变量

3.4 修改 CMakeLists.txt 文件

在 `CMakeLists.txt` 文件的末尾添加如下几行：

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

这几行代码的意思是把 `first_node.cpp` 编译成名为 `first_node` 的节点



3.5 编译节点文件

```
cd ~/catkin_ws
```

```
catkin_make
```

3.6 运行节点文件

终端输入 `roscore` ; 另开一个终端输入 : `roslaunch first_package first_node ;`

roslaunch 的调用格式为 : `roslaunch package_name node_name`



4. 基于 QT IDE 练习 ROS 编程

4.1. 安装 QT IDE

4.1.1. 安装 QtCreator

安装前需要安装相应的 GNU 开发工具集和 OpenGL 开发库：

```
sudo apt-get install build-essential libgl1-mesa-dev libevent-pthreads-2.0.5  
doxygen xorg-dev
```

下载 64 位 Linux 安装包 qt-opensource-linux-x64-5.8.0.run(安装目录为：/opt/QT)：

```
mkdir QT
```

```
cd QT
```

```
sudo wget http://download.qt.io/archive/qt/5.8/5.8.0/qt-opensource-linux-x64-  
5.8.0.run
```

下载的 run 文件不一定具有可执行权限，执行如下命令开启执行权限：

```
chmod +x opt/QT/qt-opensource-linux-x64-5.8.0.run
```

双击 .run 安装文件直接图形界面安装

默认完整安装,完成后点左上角的 Dash home，输入“qt”如果看到 Qt Creator 图标则安装成功



4.1.2. 设置快捷方式

打开 terminal，输入下面的命令：

```
gedit ~/.local/share/applications/DigiaQtOpenSource-qtcreator.desktop
```

修改文件内容如下：

```
[Desktop Entry]
```

```
Type=Application
```

```
Exec=/home/Qt5.8.0/Tools/QtCreator/bin/qtcreator
```

```
Name=Qt Creator (Community)
```

```
GenericName=The IDE of choice for Qt development.
```

```
Icon=QtProject-qtcreator
```

```
Terminal=false
```

```
Categories=Development;IDE;Qt;
```

```
MimeType=text/x-c++src;text/x-c++hdr;text/x-xsrc;application/x-
```

```
designer;application/vnd.qt.qmakeprofile;application/vnd.qt.xml.resource;text/x-
```



```
qml;text/x-qt.qml;text/x-qt.qbs;
```

- 修改 Exec 变量一行，中间添加 `bash -i -c` 即改为 `Exec=bash -i -c /home/Qt5.8.0/Tools/QtCreator/bin/qtcreator`，保存并退出。
- 添加 `bash -i -c` 是为了在通过快捷方式启动 Qt Creator 的同时加载 ROS 环境变量

4.1.3. 安装 ros_qtc_plugin 插件

Ubuntu 14.04 使用 apt-get 方式安装，会安装 5.7 版本的 Qt Creator

```
sudo add-apt-repository ppa:levi-armstrong/qt-libraries-trusty
```

```
sudo add-apt-repository ppa:levi-armstrong/ppa
```

```
sudo apt-get update && sudo apt-get install qt57creator-plugin-ros
```

安装完成在 Dash home 出现 Qt 5.7.1 Creator, 点击即可启动带插件的 QtCreator



修改系统配置文件让 Qt 启动器选择新版的 Qt

```
sudo gedit /usr/lib/x86_64-linux-gnu/qt-default/qtchooser/default.conf
```

内容为：

```
/usr/lib/x86_64-linux-gnu/qt4/bin
```



```
/usr/lib/x86_64-linux-gnu
```

修改成：

```
/home/Qt5.8.0/5.8/gcc_64/bin
```

```
/home/Qt5.8.0/5.8/gcc_64/lib
```

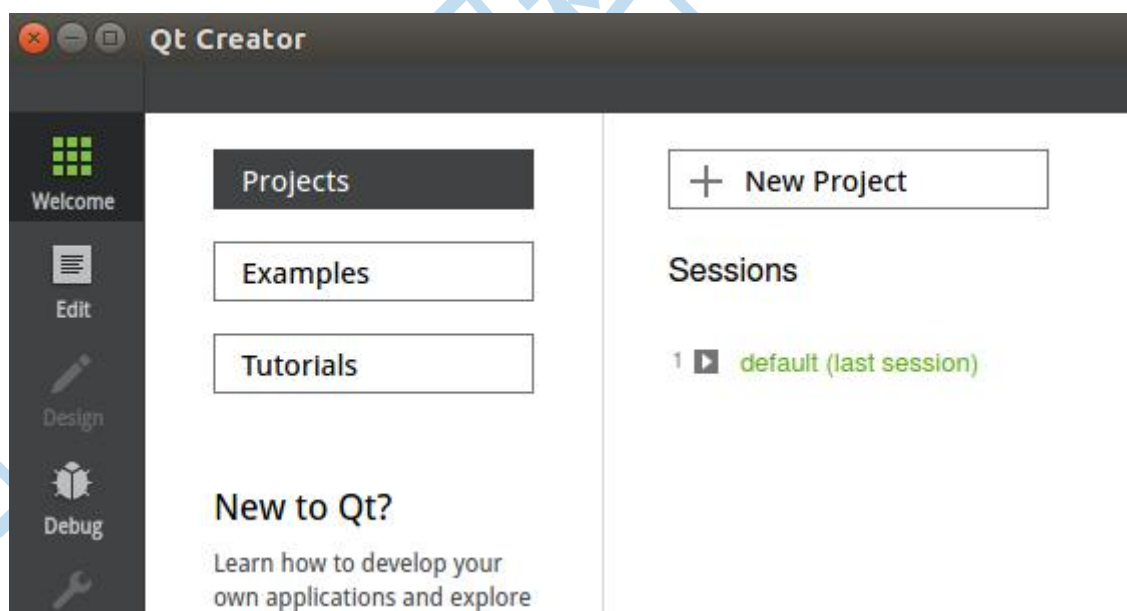
/home 是我的主文件夹绝对路径，请对应修改为自己的。

4.2. 利用 ros_qtc_plugin 编程

利用插件不仅可以新建工作空间还可以导入现有的工作空间，而“新建文件”中的“ROS”下面的“Package”、“Basic Node”等选项可以创建 package 和节点、launch 文件、urdf 文件等

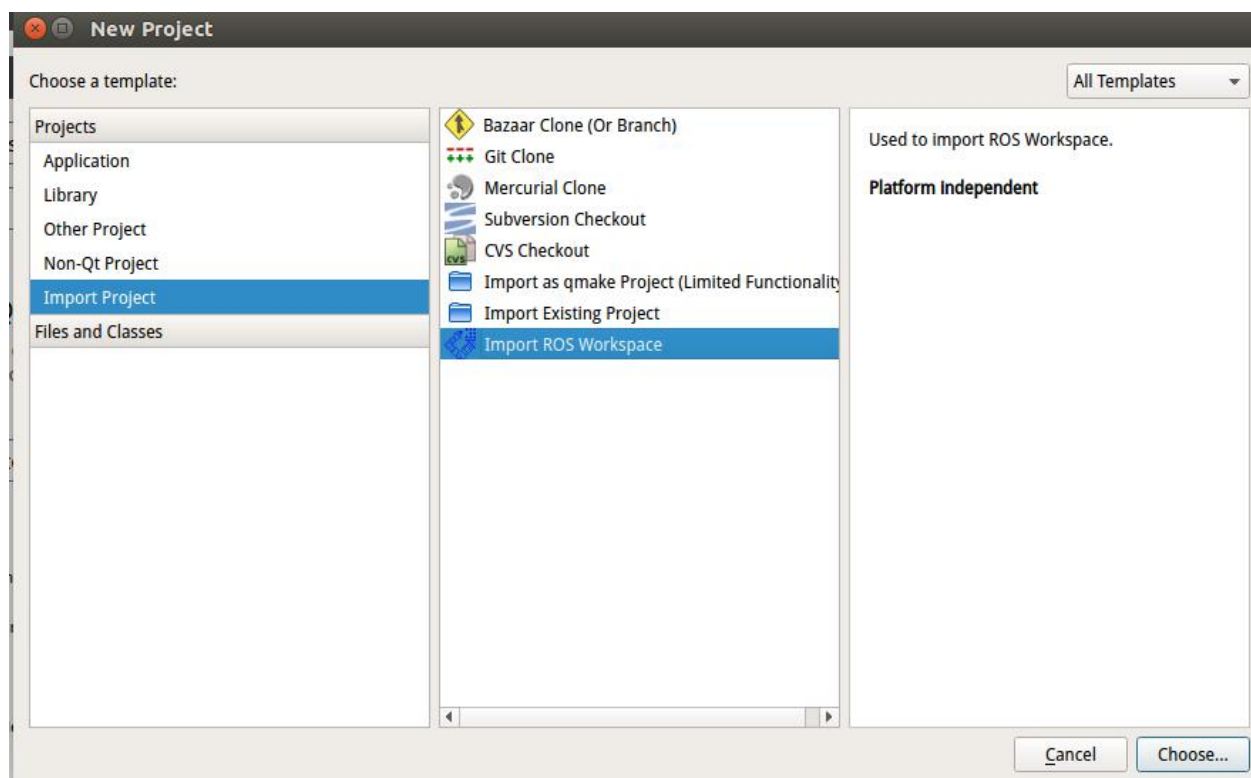
4.2.1. 新建 Project

打开 Qt，左侧选择 Projects，右侧单击 New Project：

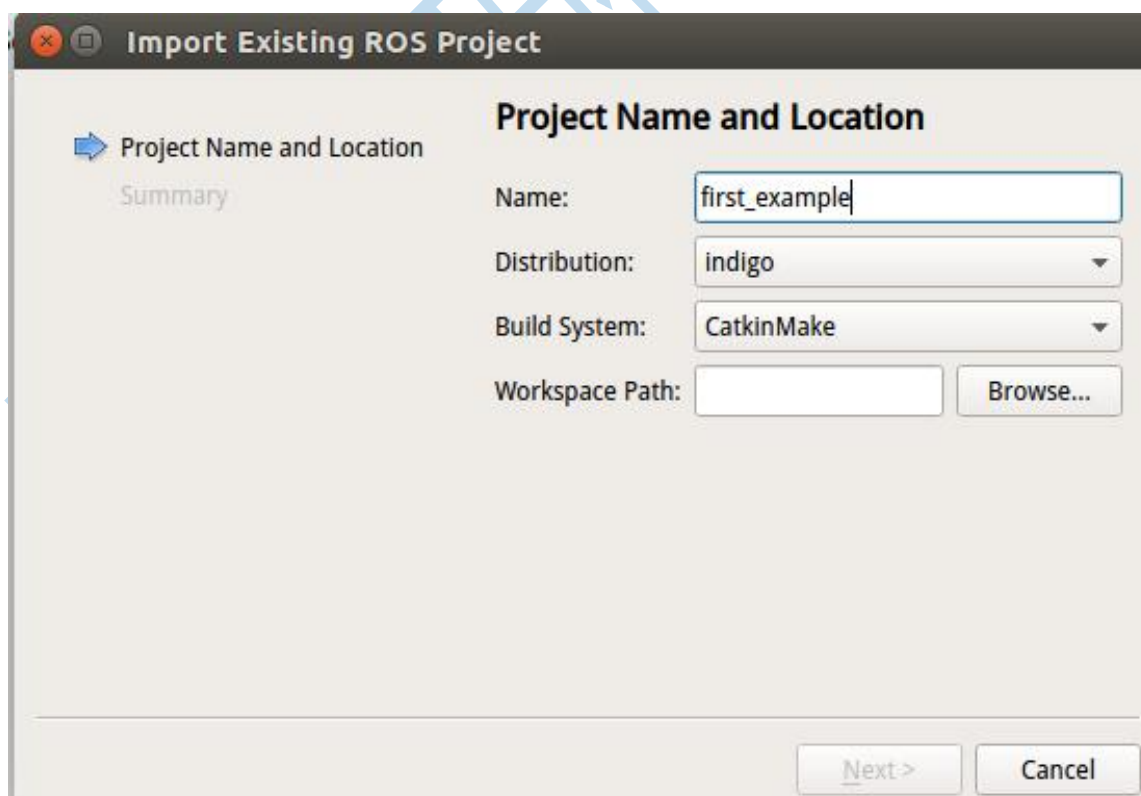




弹出对话框：

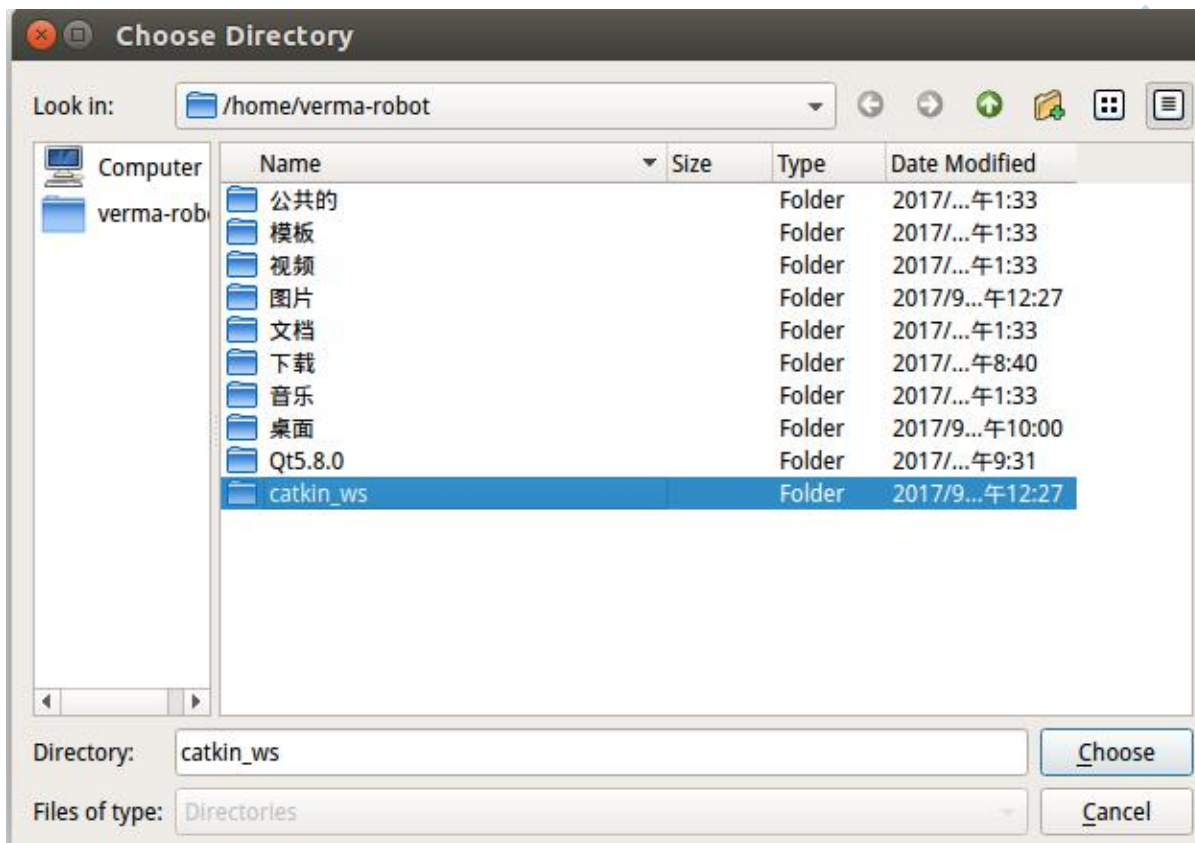


选择 Import ROS workspace, 点击 choose, 弹出对话框：



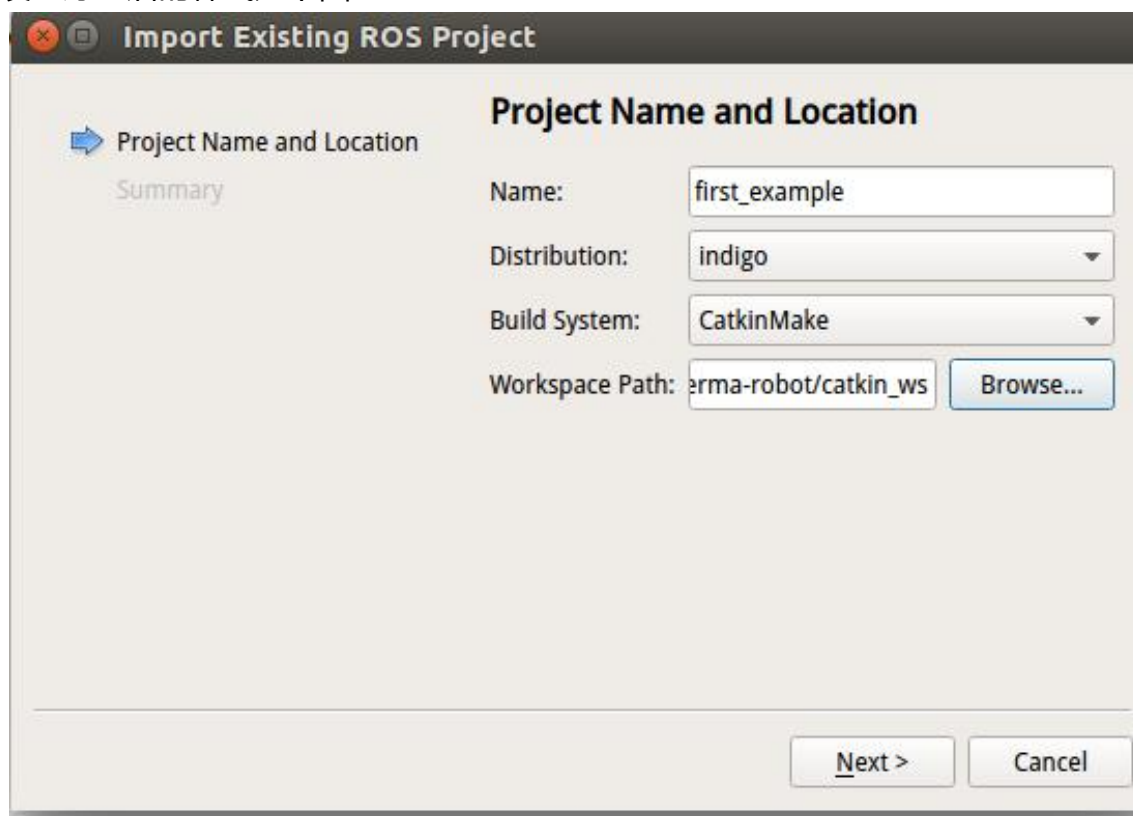


输入 project 的名字（我们第一个 project 命名为 first_example），选择 Distributor，和 Build System，单击 Browse 选择工作空间的目录（在弹出的对话框里右键新建文件夹 catkin_ws，把 ~/catkin_ws 设置成工作空间）：

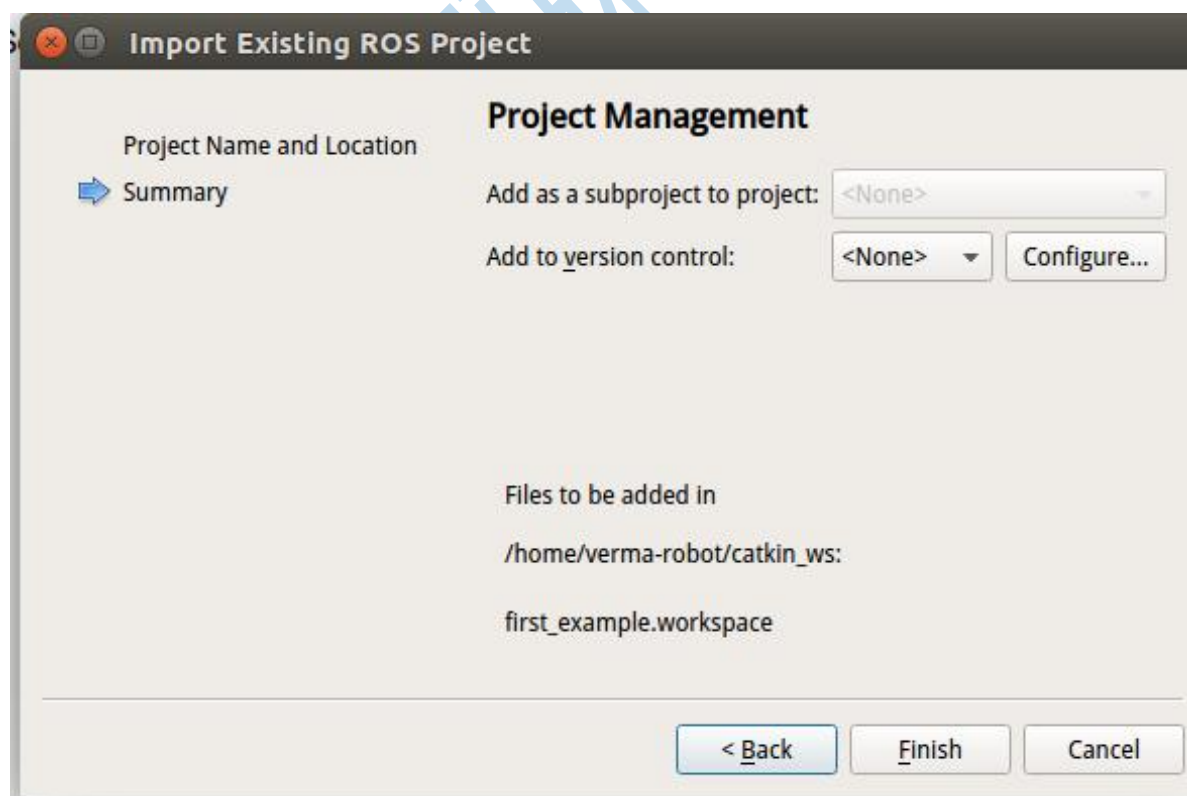




设置好之后的样式如下图：

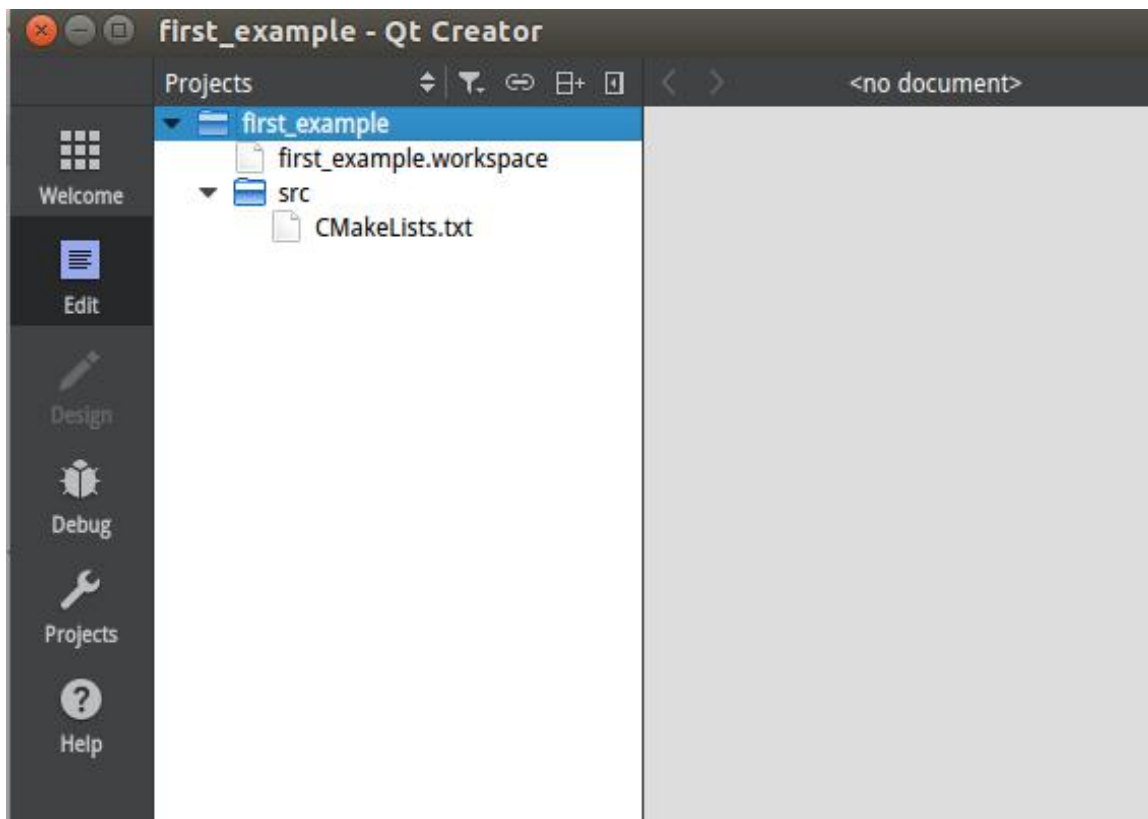


单击 Next，进入下一步：





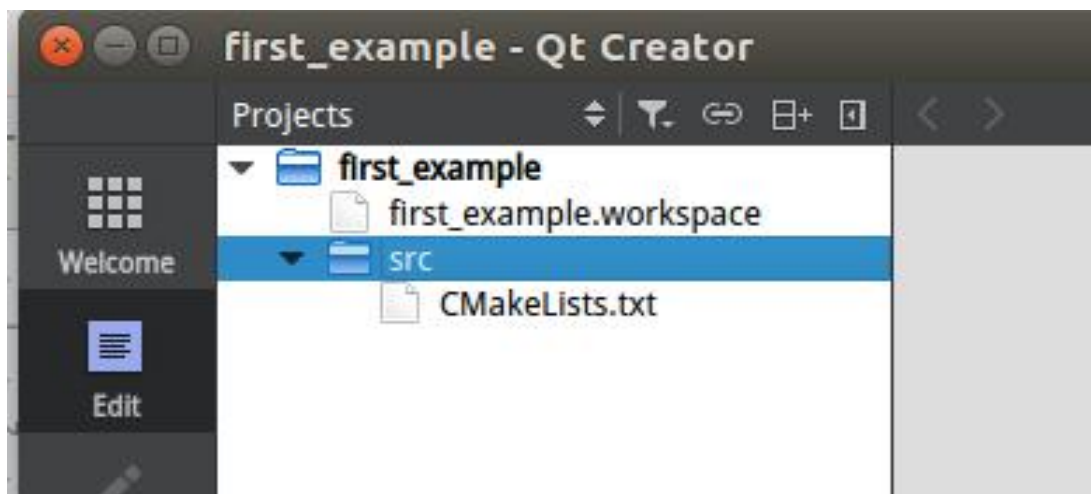
此处我们单击 Finish 即可，新建好的 project 如下图：



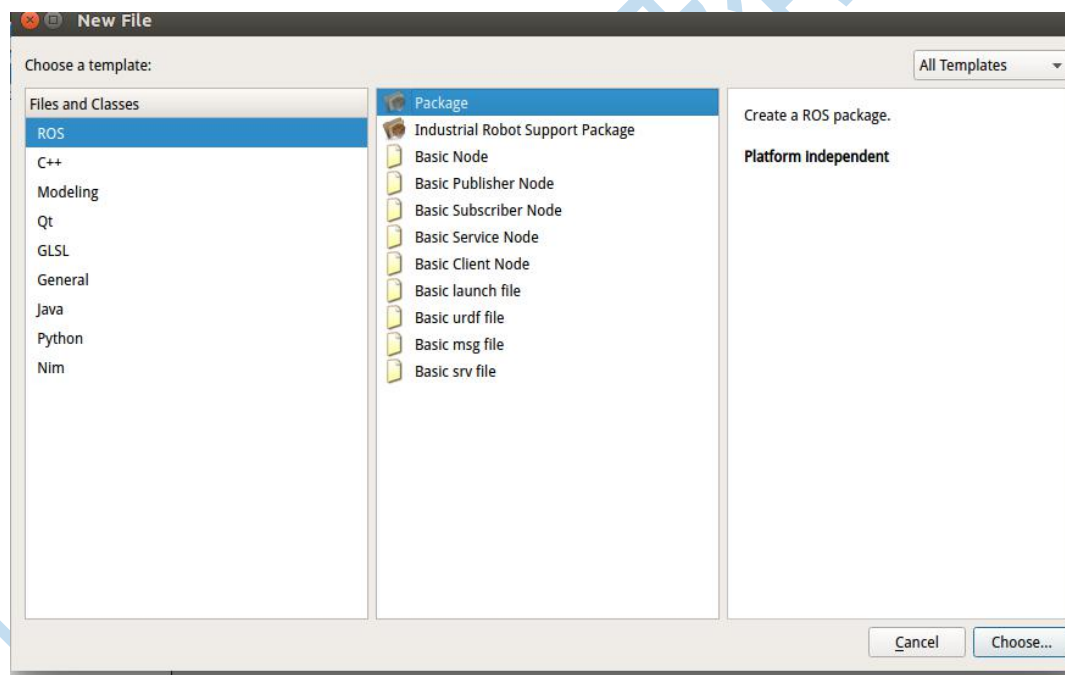


4.2.2. 新建功能包（Package）：

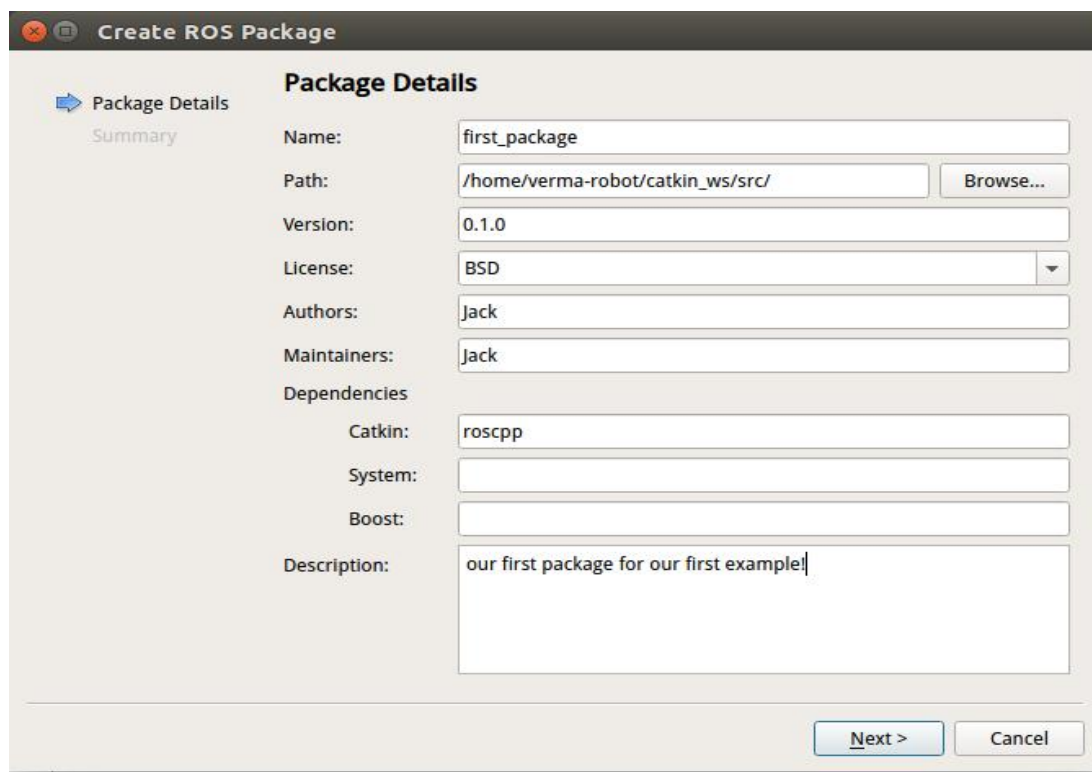
右键单击/src 文件夹，选中 Add New....:



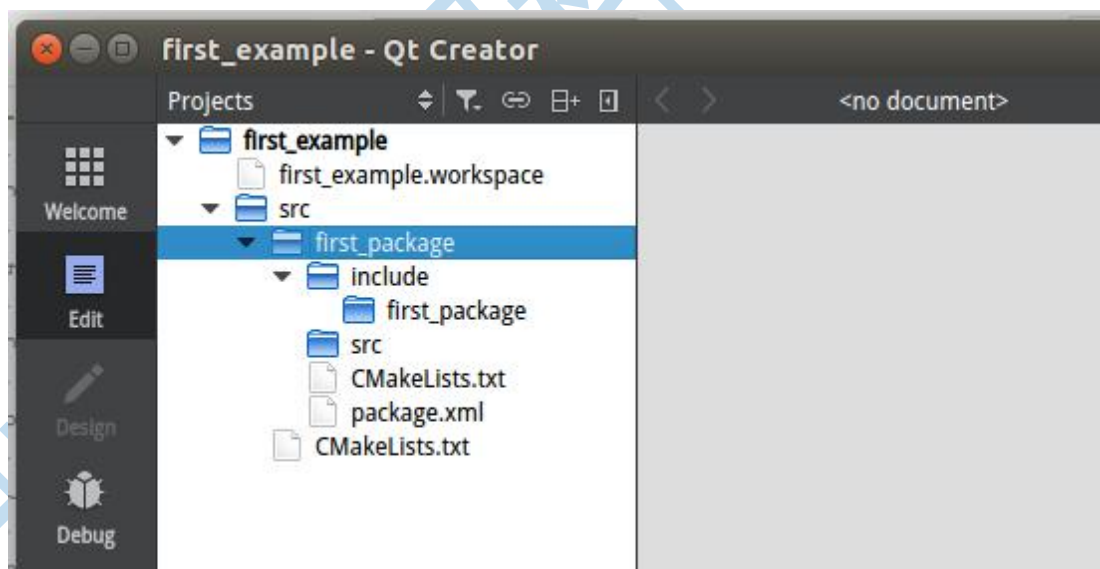
弹出对话框：



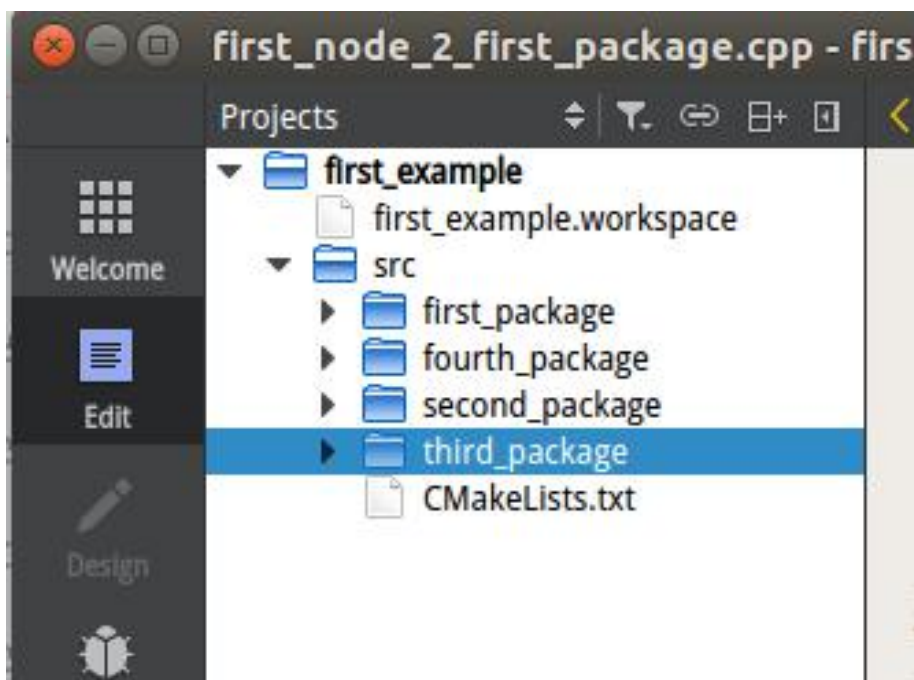
选中 Package，单击 choose 进入下一步，输入包的名字（first_package），选择包的路径（/home/verma-robot/catkin_ws/src），以及版本、作者、维护者、描述等；catkin 依赖项通常输入 roscpp：



单击 Next 进入下一步，单击 Finish 创建 first_package 功能包结束;



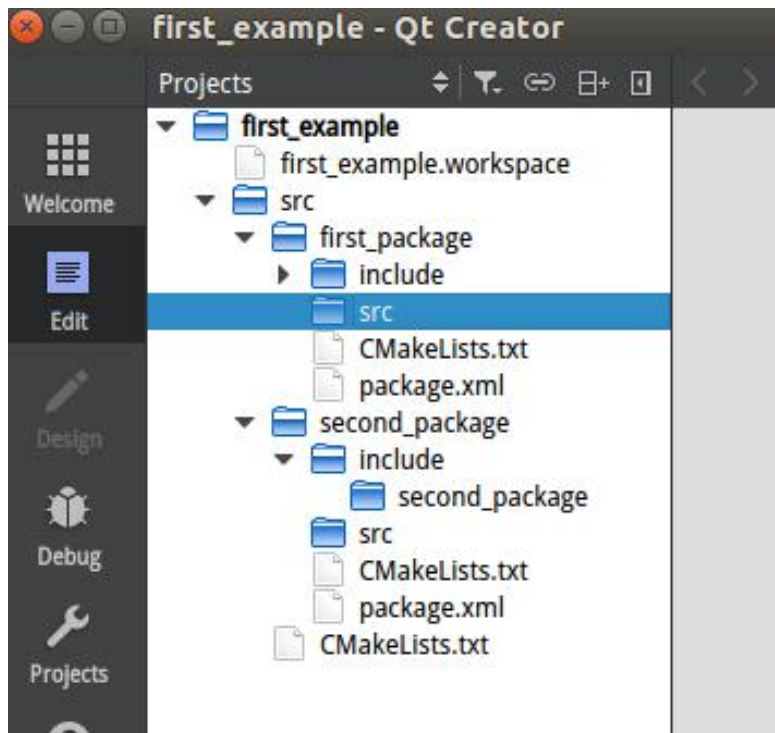
以同样的方法，我们在 first_example 的 project 中再创建几个功能包 second_package、third_package、fourth_package，创建结果如下图：



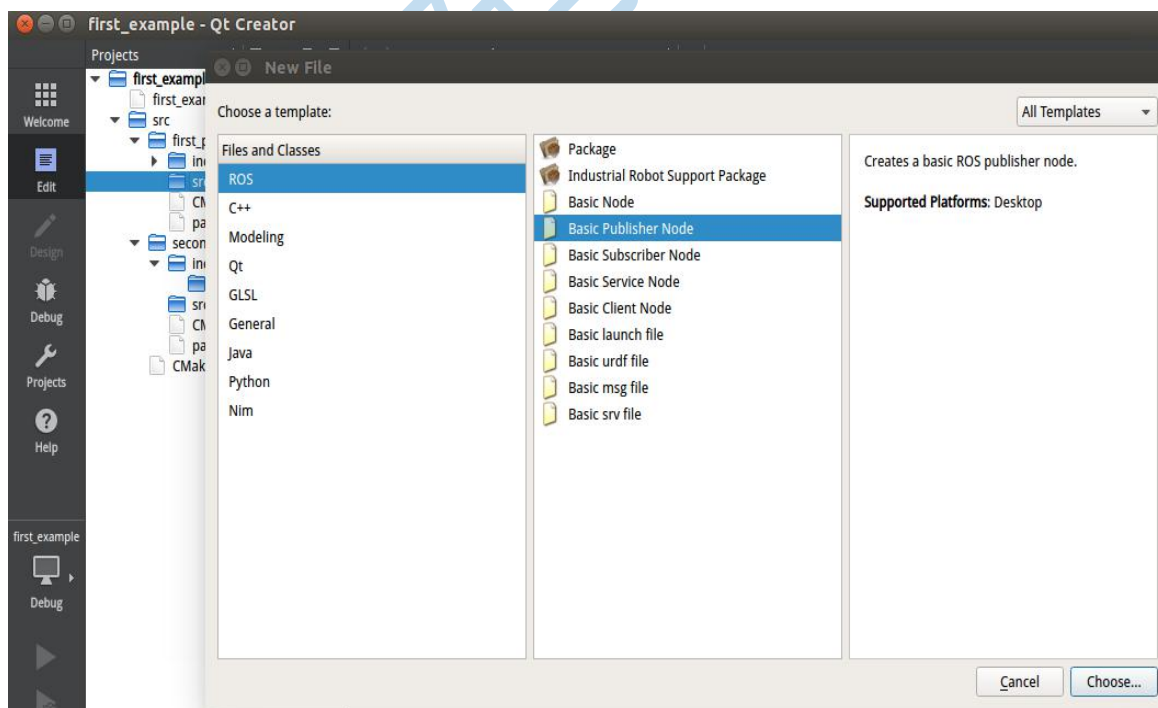


4.2.3. 采用标准消息类型编写第一个 Publisher 节点

右键单击 first_package 下面的/src，选择 Add New...，进入下一步：

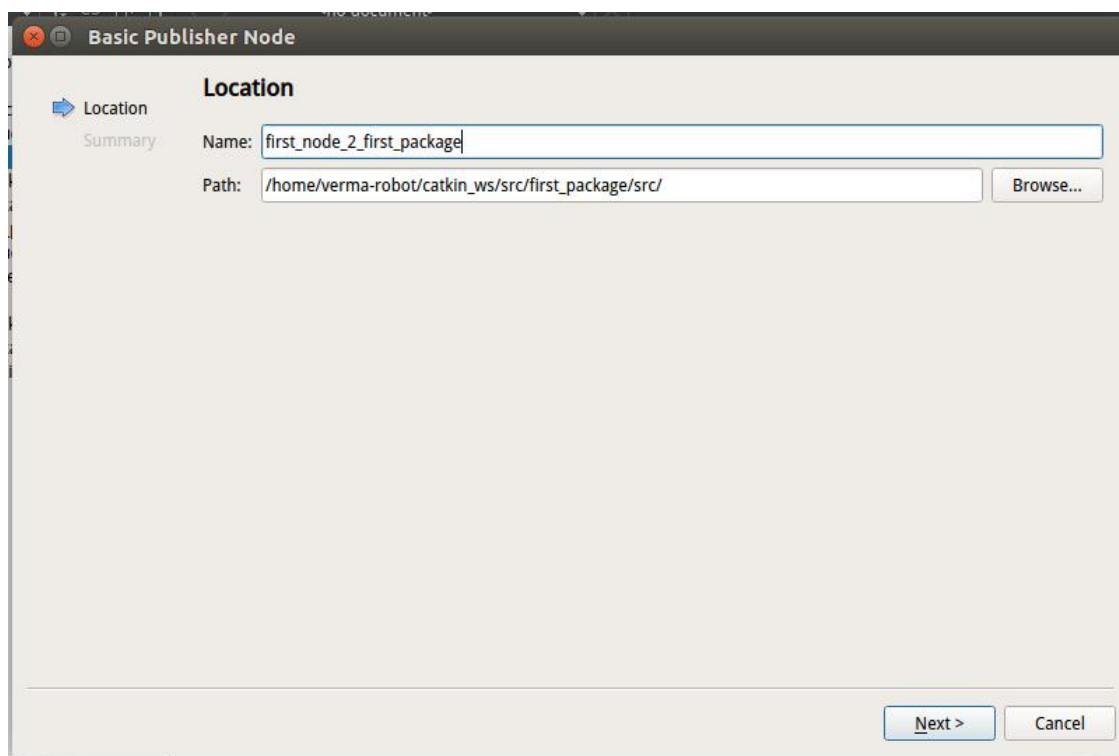


在弹出的对话框中选择第一个节点为 Basic Publisher Node，单击 choose 进入下一步：

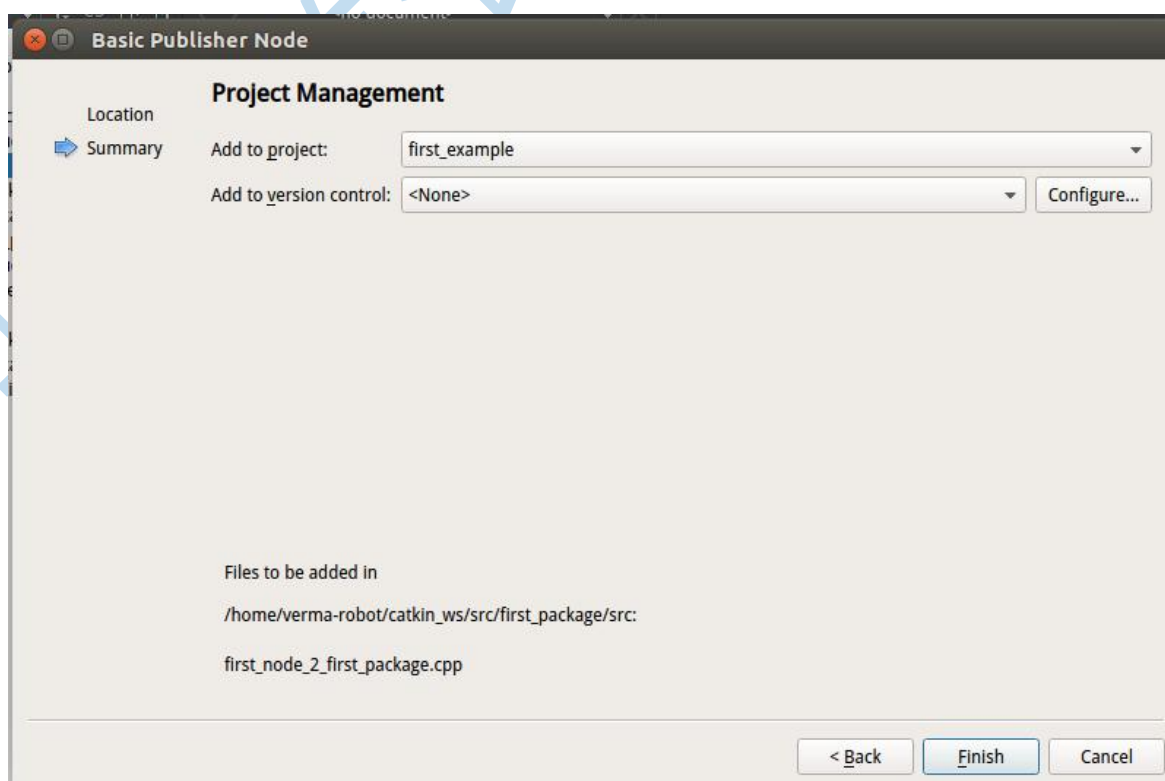




弹出的对话框中输入节点名称（ first_node_2_first_package ），选择节点文件储存路径 (/home/verma-robot/catkin_ws/src/first_package/src) ，然后单击 Next 进入下一步：

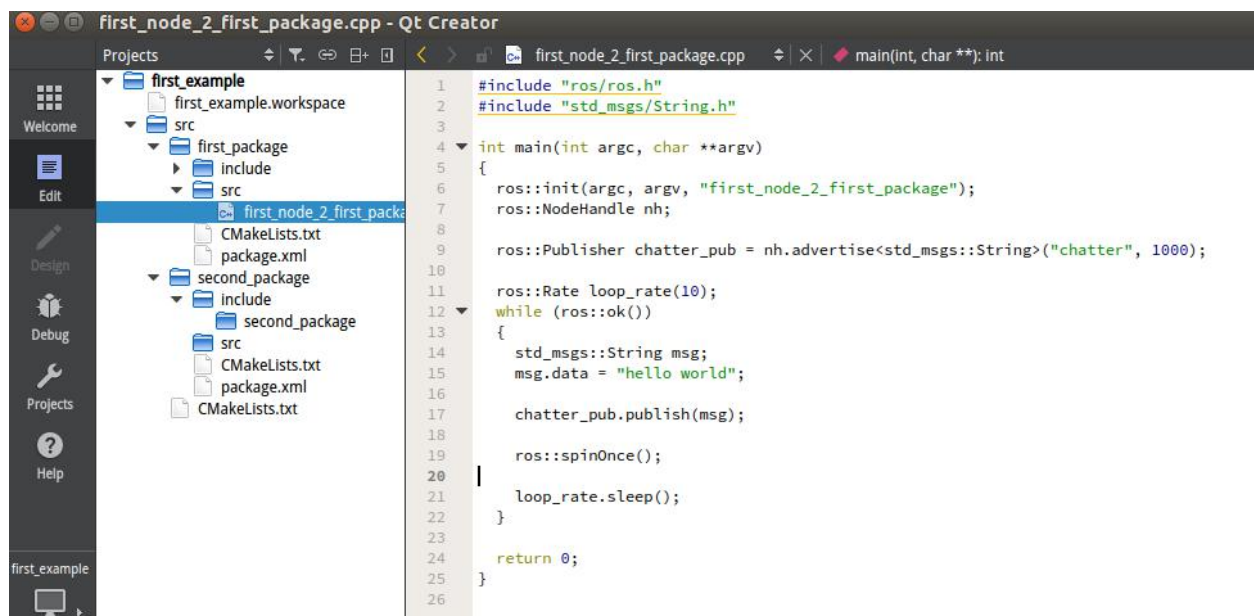


弹出的对话框直接单击 Finish ，完成节点框架创建：

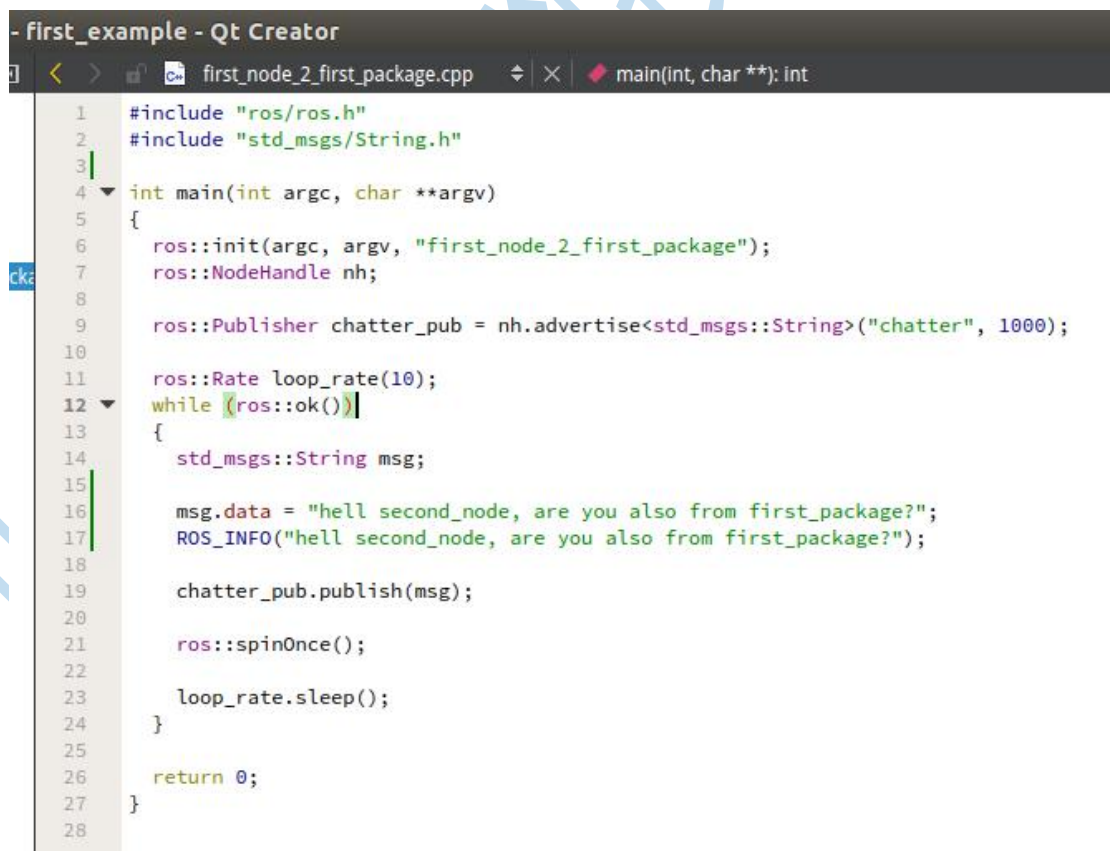




创建好的节点文件如图：



编辑代码如下：

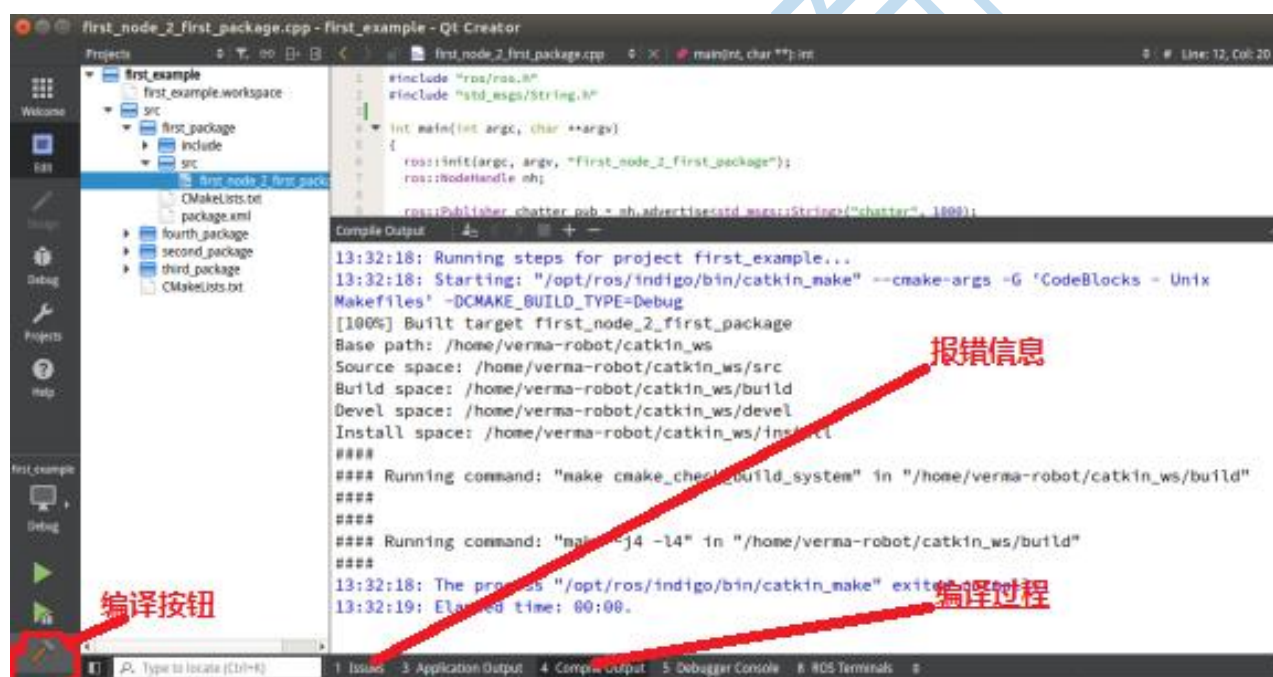




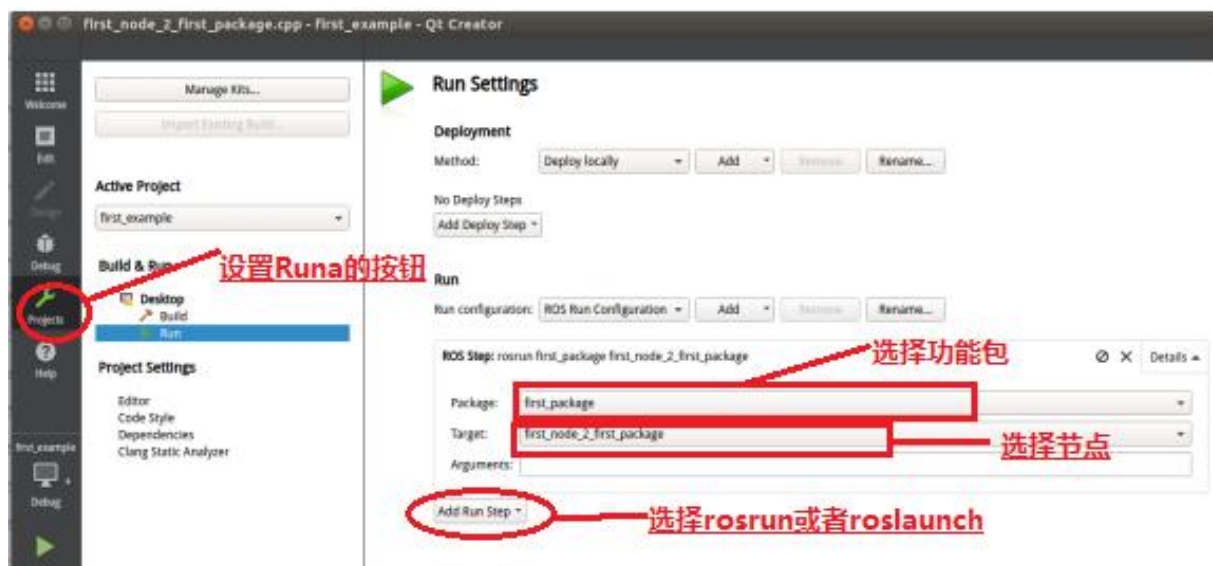
修改完代码之后，我们通常需要修改 CMakeLists 文件，双击 first_package 下 CMakeLists.txt 文件，并在最下一行加入如下代码：

```
add_executable(first_node_2_first_package src/first_node_2_first_package)
target_link_libraries(first_node_2_first_package ${catkin_LIBRARIES})
```

最后，单击左下角的“锤子”按钮完成编译，通过底端的 Compile Output 可以看到编译的过程，Issues 可列出程序中出错的地方：

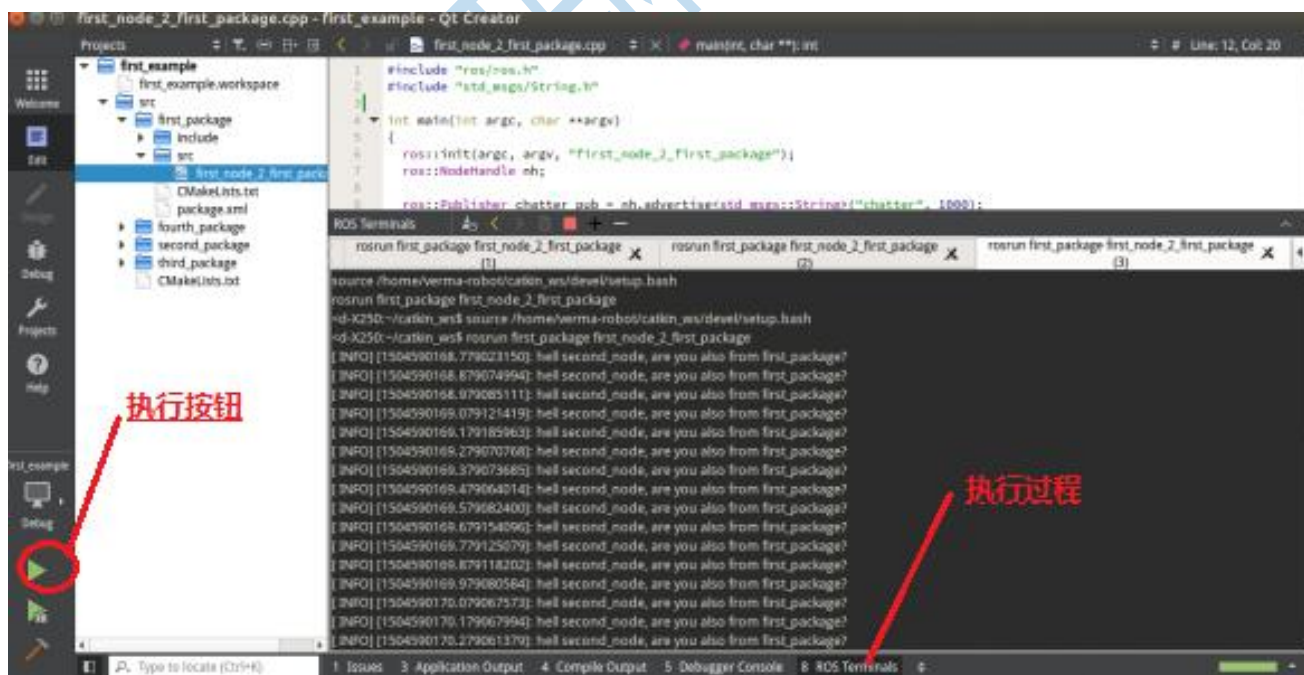


单击左侧 Project 扳手图标，设置 Run 的选项，如下图，Add Run Setup 选项中选择 ROS Run Step 选项，Package 选项中选择 first_package，Target 选择 first_node_2_first_package，因为我们不需要传递参数，所以 Arguments 选项为空：



设置好之后，我们新开 Ubuntu 终端，输入：roscore，启动 ROS 节点管理器。

然后单击 Qt 左下角的绿色三角按钮，即执行该节点文件，下方的 ROS TERMINAL 会显示执行过程中输出到屏幕的信息，如图：



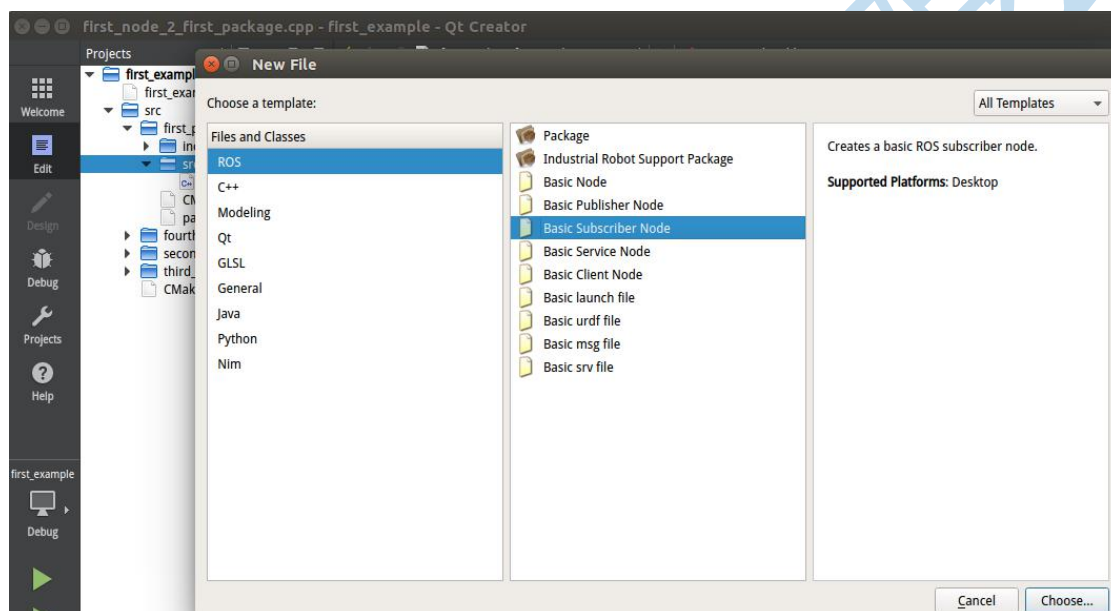
至此，我们第一个节点文件，编写就编写成功了。



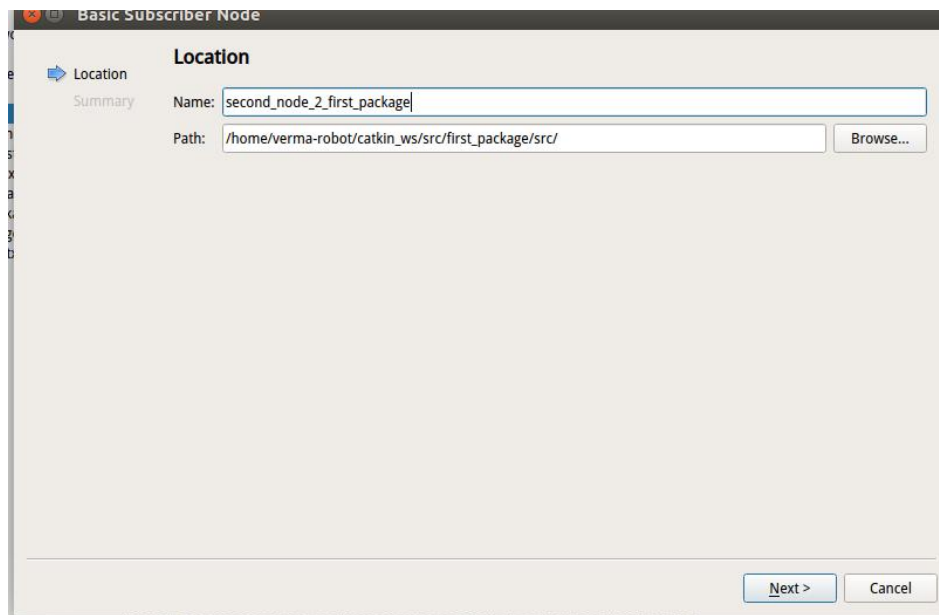
4.2.4. 采用标准消息类型编写第一个 Subscriber 节点

下面我们将为 first_package 编写第二个节点，第二个节点订阅第一个节点发布的主题，当接收到第一个节点的消息时，向屏幕输出一条消息。

如新建第一个 Publisher 节点时，我们右键单击 first_package 下的/src 文件夹，选择 Add New...，此处我们选择新建的节点类型为：Basic Subscriber Node：



节点名称设置为：second_node_2_first_package:



节点代码修改成如下：

```
second_node_2_first_package.c... | chatterCallback(const std_msgs::String::ConstPtr & msg)
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("is first_node ask me:[%s]? ", msg->data.c_str());

    ROS_INFO("yes, I am second_node, I am from first_package too!");
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "second_node_2_first_package");
    ros::NodeHandle nh;

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

    ros::spin();

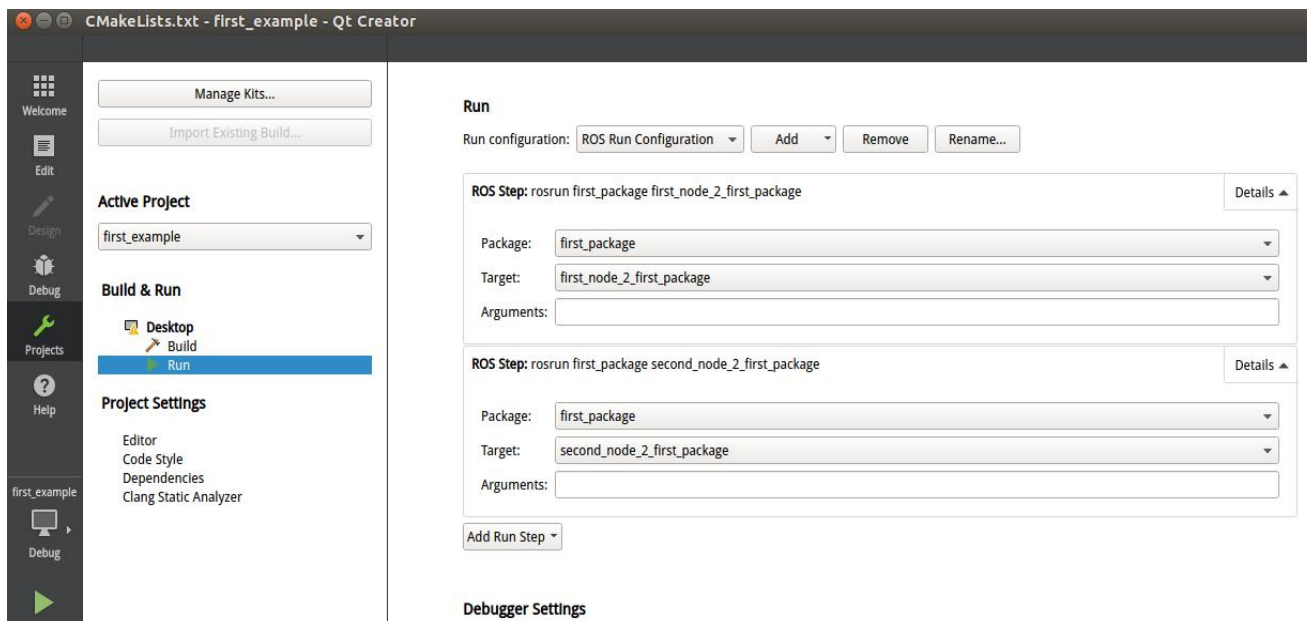
    return 0;
}
```

节点代码修改好之后，我们需要修改 CMakeLists.txt 文件，在 CMakeLists.txt 文件末端加入：

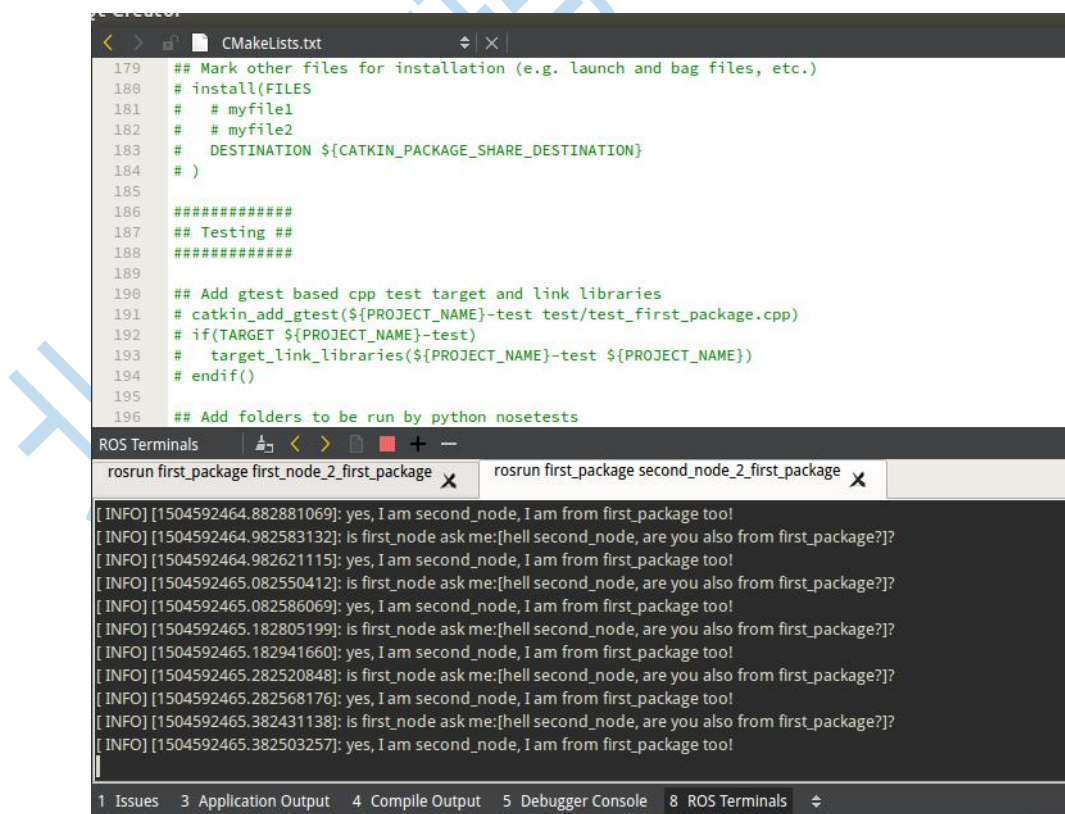


```
add_executable(second_node_2_first_package src/second_node_2_first_package)
target_link_libraries(second_node_2_first_package ${catkin_LIBRARIES})
```

编译完成后需要配置 Run，配置结果如下图：



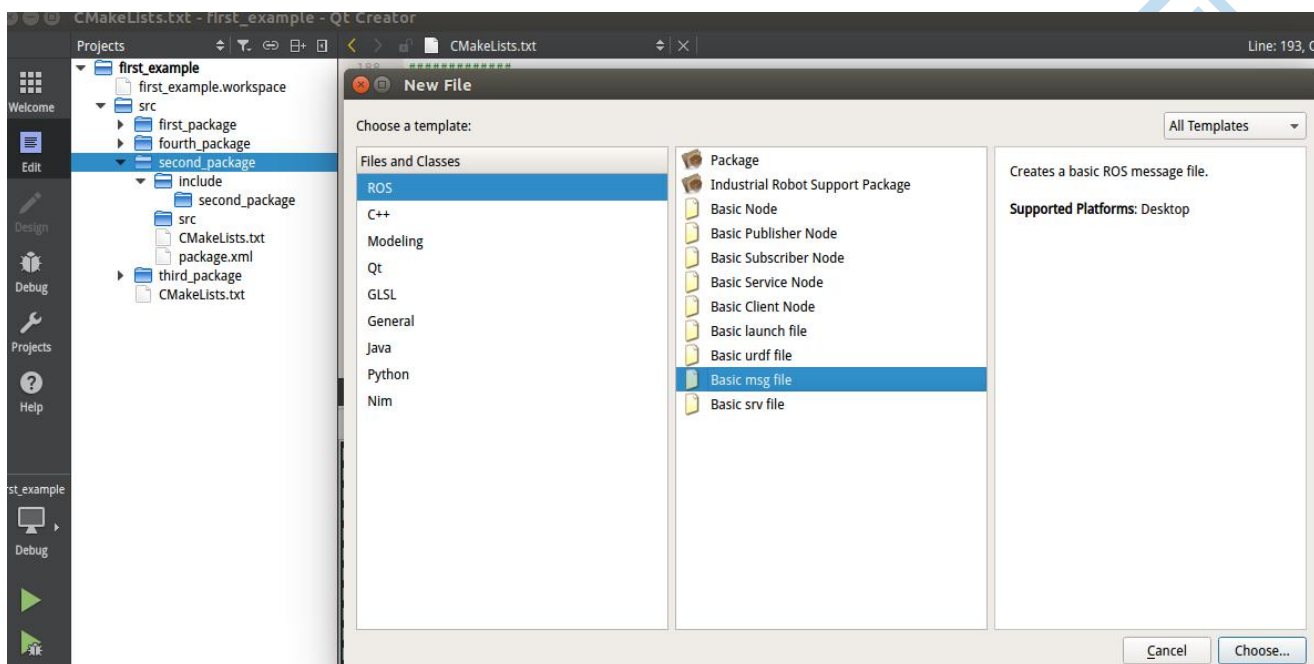
然后 ubuntu 终端输入：roscore；单击 Qt 左下角绿色三角按钮，执行节点文件，执行结果如图：





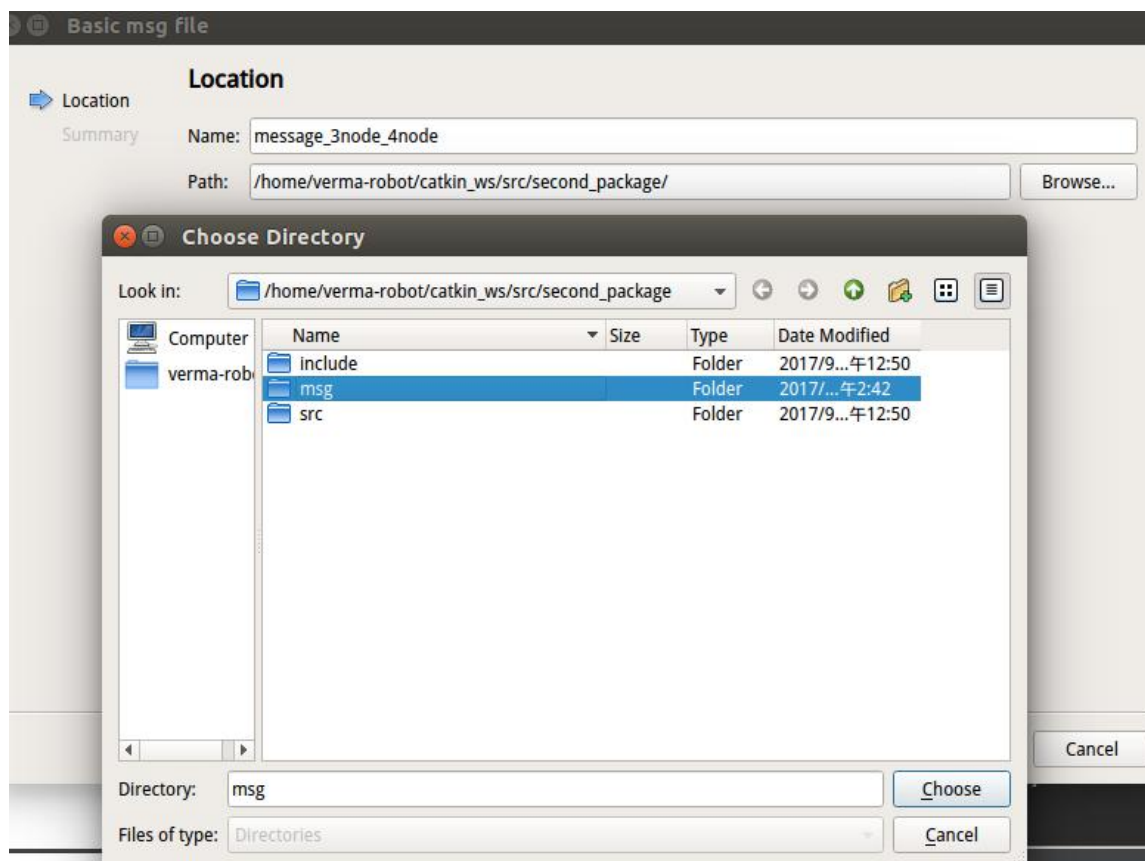
4.2.5. 编写第一个非标准消息文件

我们将在 second_package 功能包中新建第一个非标准消息文件，右键单击 second_package，选择 Add New,弹出对话框：

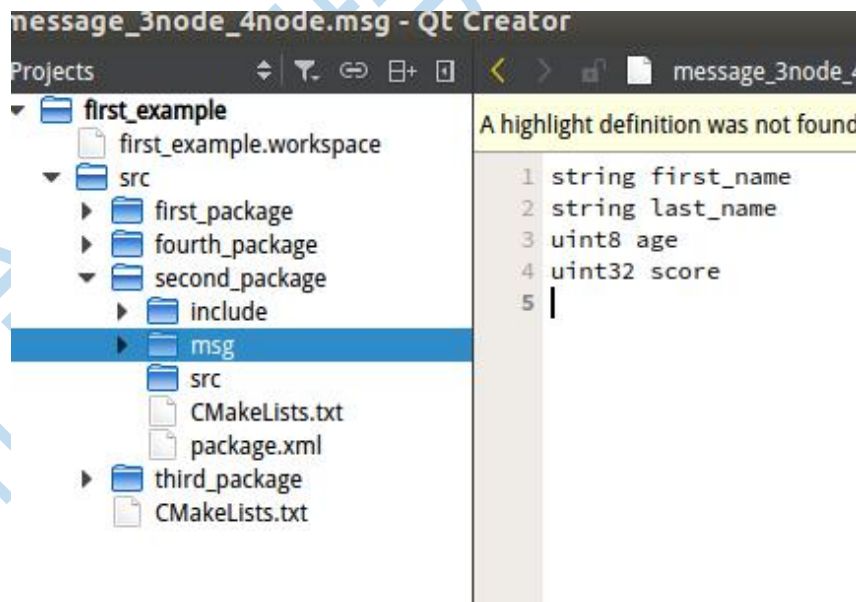


选择 Basic msg Files，单击 choose 进入下一步；输入消息文件的名称，并选择消息文件储存的路径（通过新建文件 msg，使路径为 /home/verma-robot/catkin_ws/src/second_package/msg）：

:

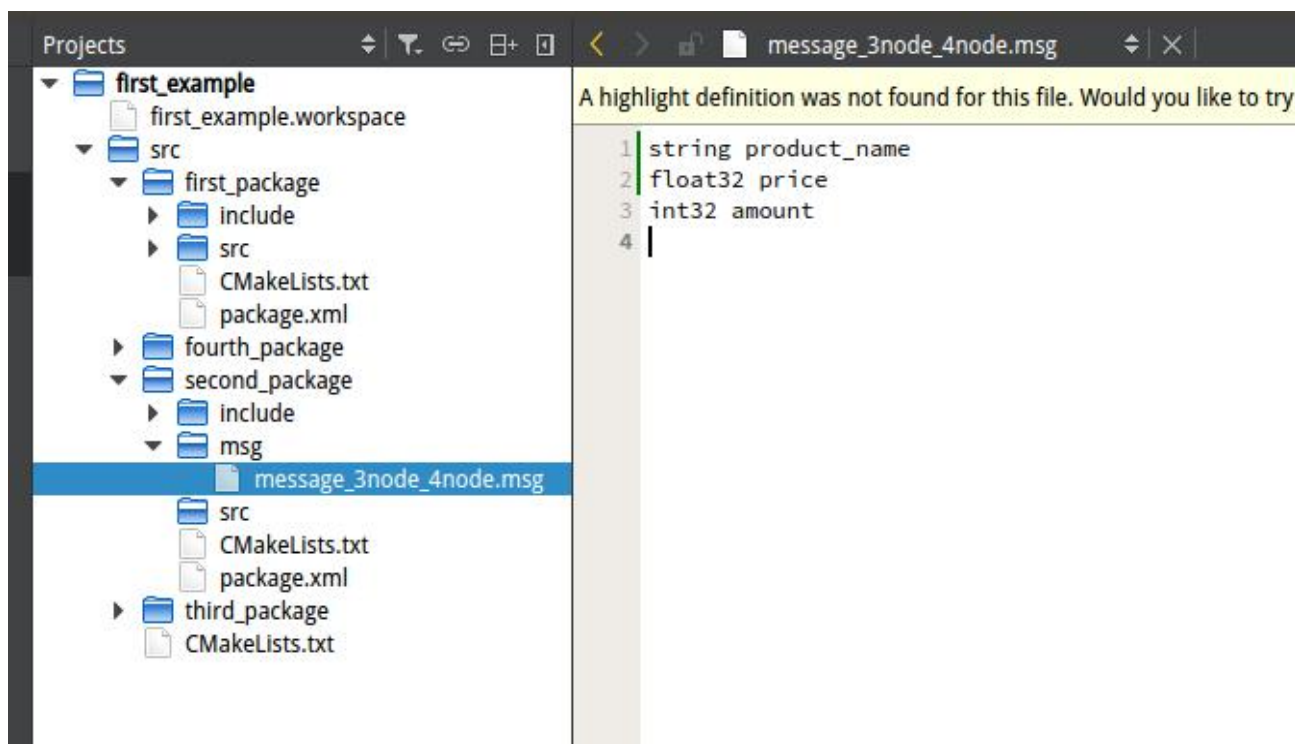


创建完毕之后，默认的消息类型为：



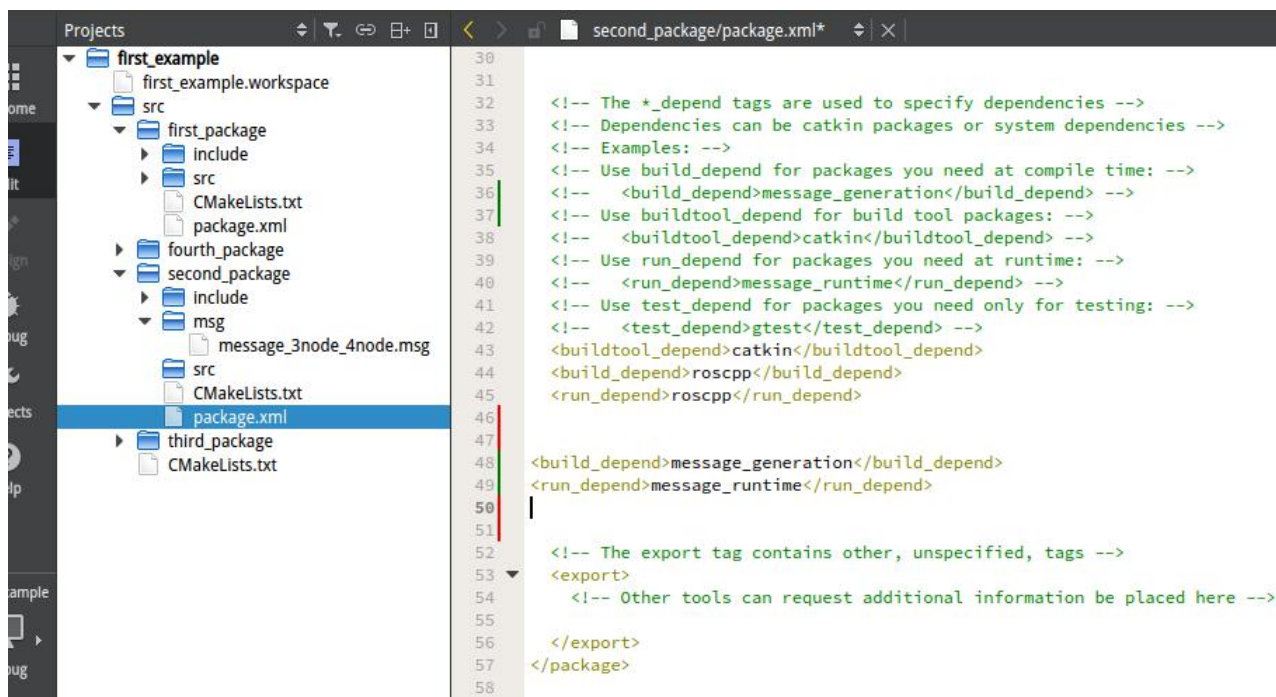


修改消息文件，将 message_3node_4node.msg 修改成如下：



修改 package.xml 文件，双击 second_package 下的 package.xml 文件，添加：

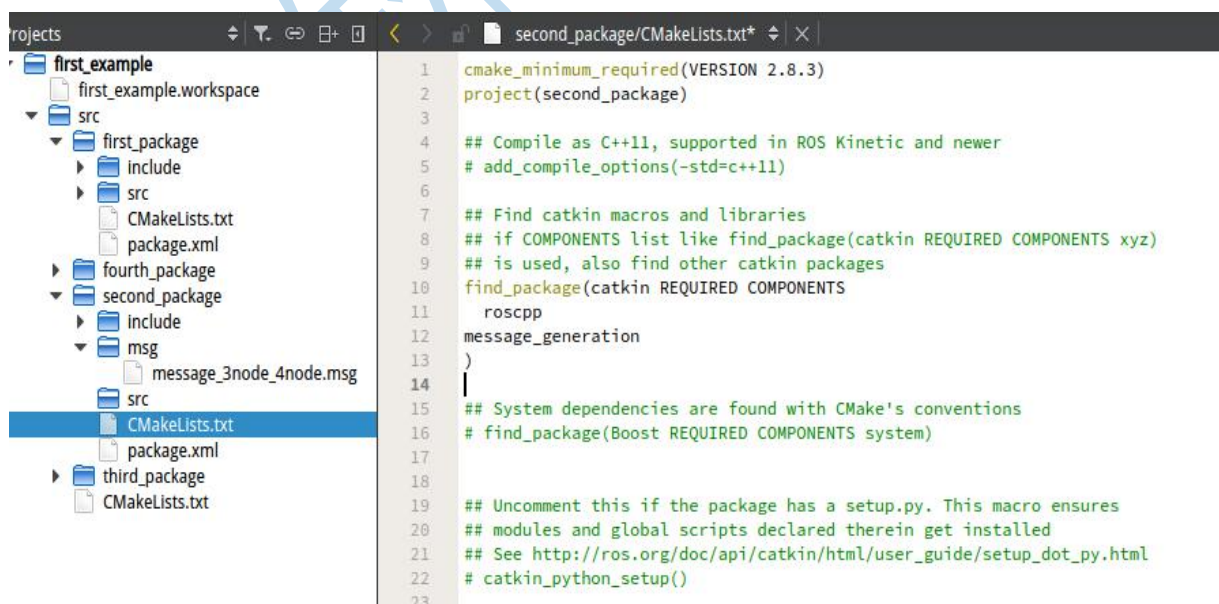
```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

修改 CMakeLists.txt 文件：

- 修改 find_package()函数，使之变成：

```
find_package(catkin
  roscpp
  message_generation
)
```





- 取消对 add_message()和 generate_message()函数的注释，并做修改：

```
49 ## Generate messages in the 'msg' folder
50 add_message_files(
51     FILES
52     message_3node_4node.msg
53 )
54
55
56 ## Generate services in the 'srv' folder
57 # add_service_files(
58 #     FILES
59 #     Service1.srv
60 #     Service2.srv
61 # )
62
63 ## Generate actions in the 'action' folder
64 # add_action_files(
65 #     FILES
66 #     Action1.action
67 #     Action2.action
68 # )
69
70 ## Generate added messages and services with any d
71 generate_messages(
72     DEPENDENCIES
73     std_msgs
74 )
```

- 修改 catkin_package()函数：

```
94
95 #####
96 ## catkin specific configuration ##
97 #####
98 ## The catkin_package macro generates cmake config files for your pack
99 ## Declare things to be passed to dependent projects
100 ## INCLUDE_DIRS: uncomment this if you package contains header files
101 ## LIBRARIES: libraries you create in this project that dependent proj
102 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
103 ## DEPENDS: system dependencies of this project that dependent project
104 catkin_package(
105     CATKIN_DEPENDS message_runtime
106 )
107
108
109 #####
110 ## Build ##
111 #####
112
113
114 ## Specify additional locations of header files
115 ## Your package locations should be listed before other locations
116 include_directories(
117     # include
118     ${catkin_INCLUDE_DIRS}
119 )
```



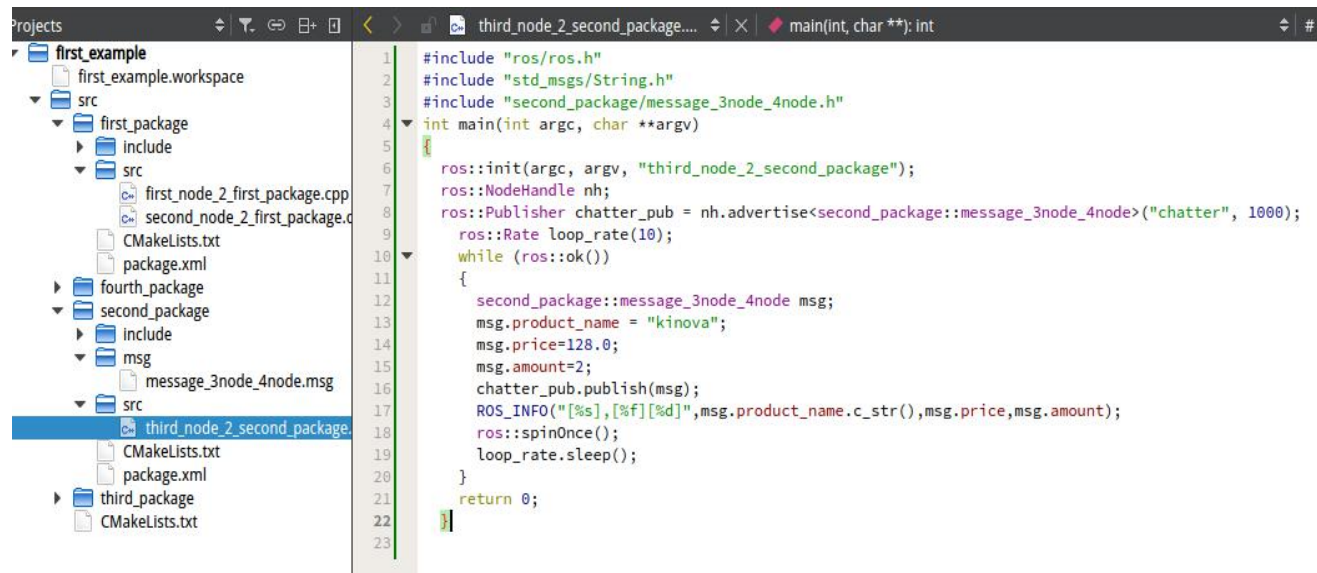
编译通过即完成非标准消息文件的编制。

北京维尔玛科技有限公司

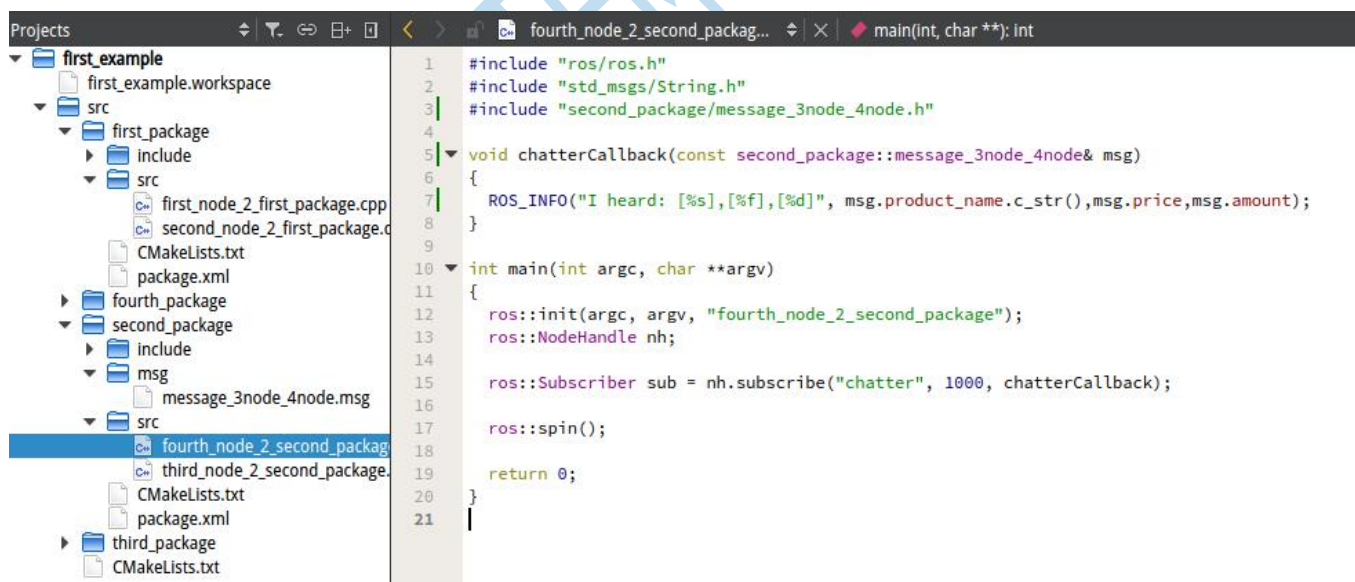


4.2.6. 采用非标准消息文件编写 Publisher 节点和 Subscribe 节点

我们新建一个 Publisher 节点和 Subscribe 节点，创建方法见 4.2.3，并分别修改代码如下：



```
1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3 #include "second_package/message_3node_4node.h"
4 int main(int argc, char **argv)
5 {
6     ros::init(argc, argv, "third_node_2_second_package");
7     ros::NodeHandle nh;
8     ros::Publisher chatter_pub = nh.advertise<second_package::message_3node_4node>("chatter", 1000);
9     ros::Rate loop_rate(10);
10    while (ros::ok())
11    {
12        second_package::message_3node_4node msg;
13        msg.product_name = "kinova";
14        msg.price=128.0;
15        msg.amount=2;
16        chatter_pub.publish(msg);
17        ROS_INFO("[%s],[%f],[%d]",msg.product_name.c_str(),msg.price,msg.amount);
18        ros::spinOnce();
19        loop_rate.sleep();
20    }
21    return 0;
22 }
```



```
1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3 #include "second_package/message_3node_4node.h"
4
5 void chatterCallback(const second_package::message_3node_4node& msg)
6 {
7     ROS_INFO("I heard: [%s],[%f],[%d]", msg.product_name.c_str(),msg.price,msg.amount);
8 }
9
10 int main(int argc, char **argv)
11 {
12     ros::init(argc, argv, "fourth_node_2_second_package");
13     ros::NodeHandle nh;
14
15     ros::Subscriber sub = nh.subscribe("chatter", 1000, chatterCallback);
16
17     ros::spin();
18
19     return 0;
20 }
21
```

修改 CMakeLists.txt 文件在末尾添加如下代码：

```
add_executable(third_node_2_second_package src/third_node_2_second_package)
```

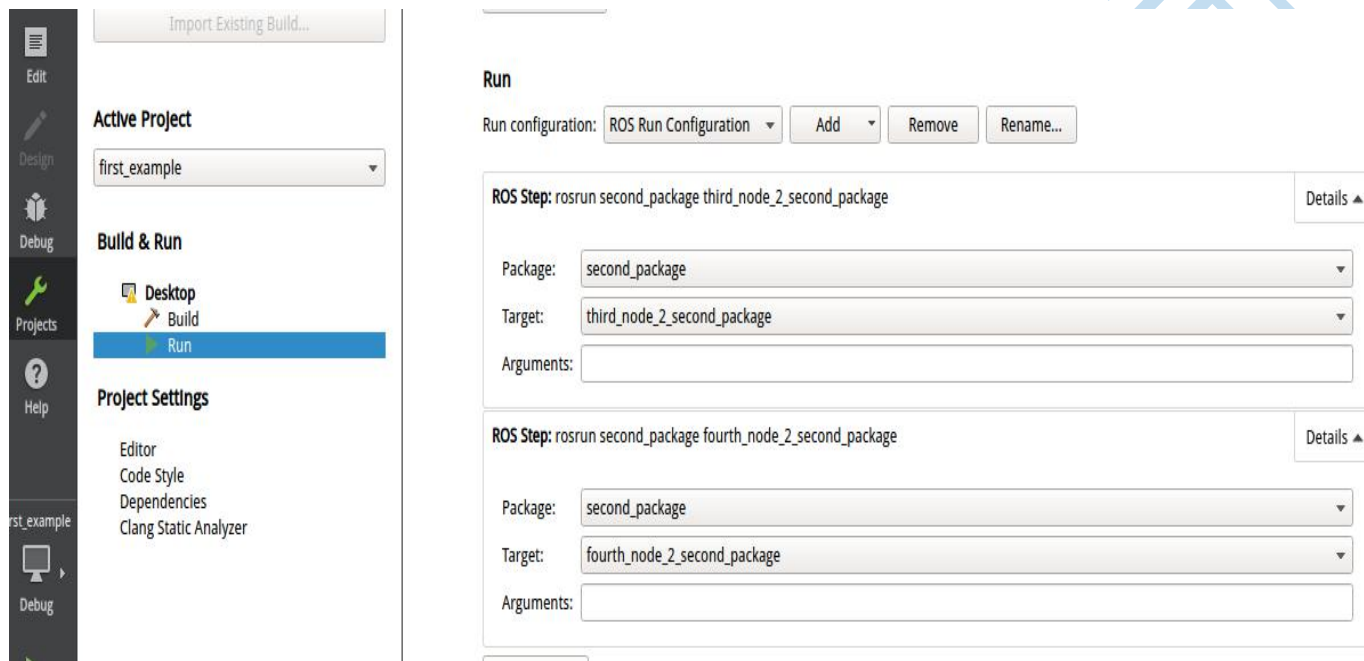


```
target_link_libraries(third_node_2_second_package ${catkin_LIBRARIES})

add_executable(fourth_node_2_second_package src/fourth_node_2_second_package)

target_link_libraries(fourth_node_2_second_package ${catkin_LIBRARIES})
```

编译，然后设置 run:

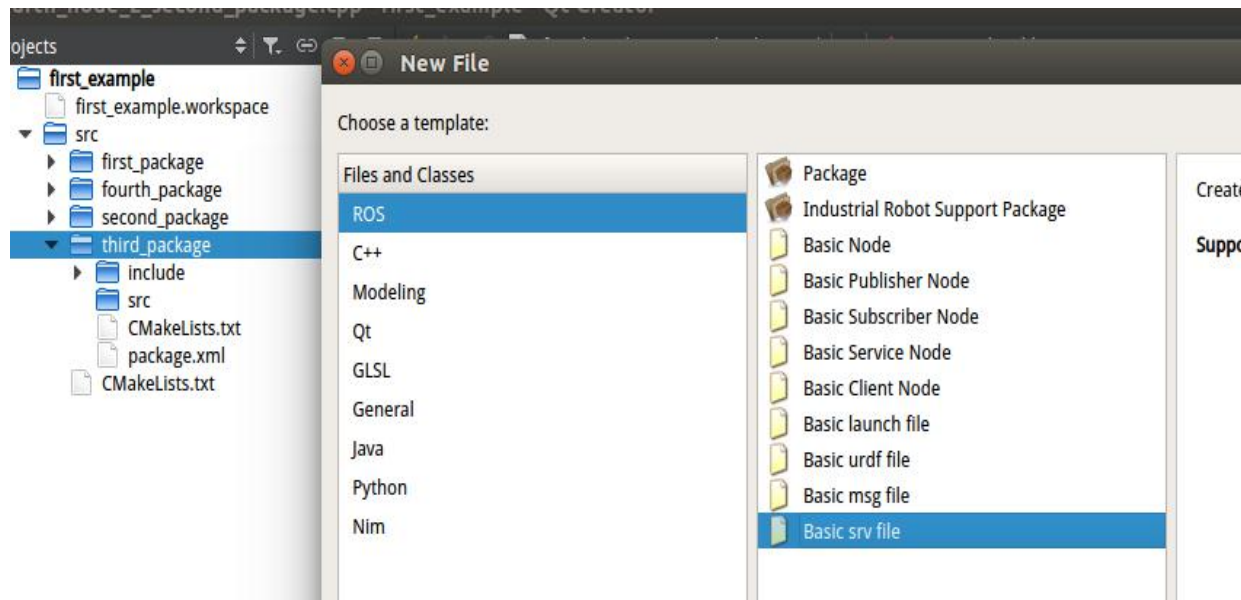


运行即可

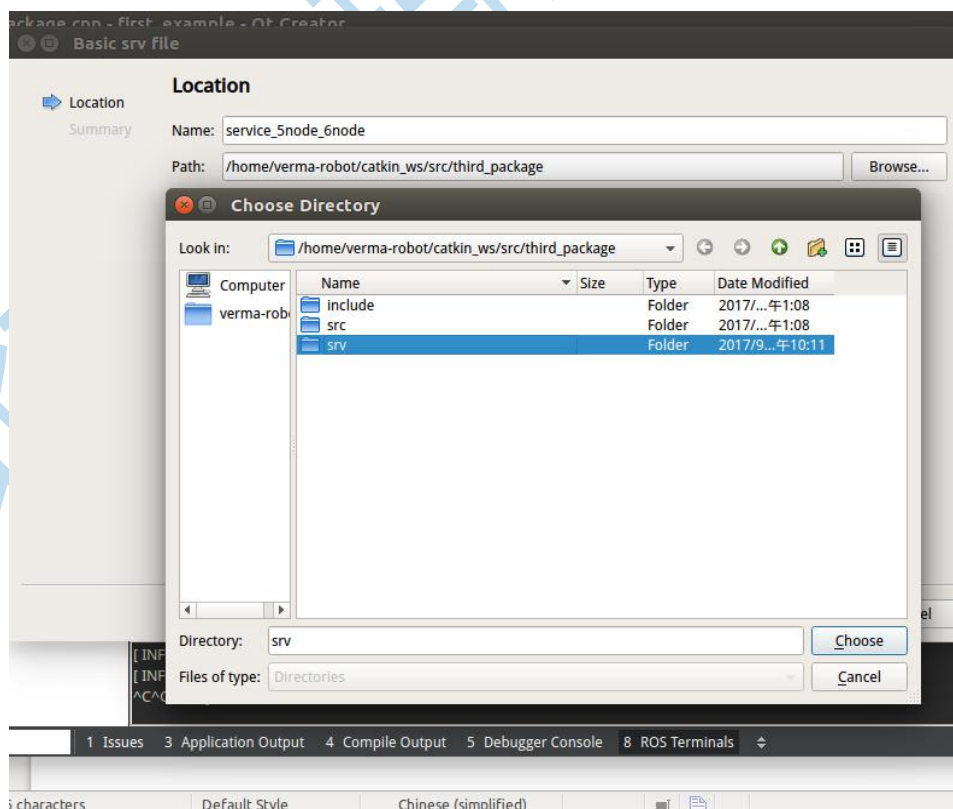


4.2.7. 编写第一个服务文件（.srv 文件）

我们将在 third_package 下新建我们第一个 srv 文件；右键单击 third_package，单击 Add New,选择 Basic Srv file：

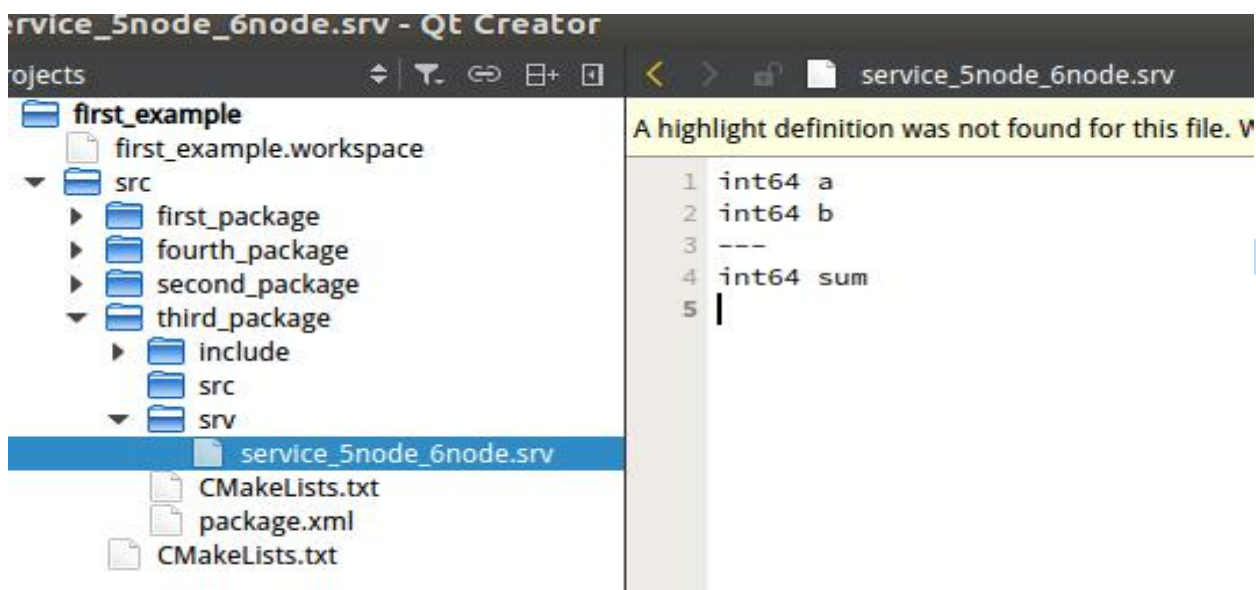


,单击 choose 进入下一步，输入服务文件的名称：service_5node_6node,服务文件保存的路径为：/home/verma-robot/catkin_ws/src/third_package/srv：





单击 choose 进入下一步，最后创建的服务文件如图：



修改 package.xml 文件，在 package.xml 文件中加入：

```
<build_depend>message_generation</build_depend>

<run_depend>message_runtime</run_depend>
```



```
third_package/package.xml* - first_example - Qt Creator
Projects
  first_example
    first_example.workspace
    src
      first_package
      fourth_package
      second_package
      third_package
        include
        src
          service_5node_6node.srv
          CMakeLists.txt
          package.xml
          CMakeLists.txt
29 <author>Jack</author>
30
31
32 <!-- The *depend tags are used to specify dependencies -->
33 <!-- Dependencies can be catkin packages or system dependencies -->
34 <!-- Examples: -->
35 <!-- Use build_depend for packages you need at compile time: -->
36 <!--   <build_depend>message_generation</build_depend> -->
37 <!-- Use buildtool_depend for build tool packages: -->
38 <!--   <buildtool_depend>catkin</buildtool_depend> -->
39 <!-- Use run_depend for packages you need at runtime: -->
40 <!--   <run_depend>message_runtime</run_depend> -->
41 <!-- Use test_depend for packages you need only for testing: -->
42 <!--   <test_depend>gtest</test_depend> -->
43 <buildtool_depend>catkin</buildtool_depend>
44 <build_depend>roscpp</build_depend>
45 <run_depend>roscpp</run_depend>
46
47 <build_depend>message_generation</build_depend>
48 <run_depend>message_runtime</run_depend>
49
50
51 <!-- The export tag contains other, unspecified, tags -->
52 <export>
53   <!-- Other tools can request additional information be placed here -->
54
55 </export>
```

修改 CMakeLists.txt 文件，参照 second_package 创建消息时的修改方法进行修改，修改后的文件如图：

```
third_package/CMakeLists.txt - first_example - Qt Creator
Projects
  first_example
    first_example.workspace
    src
      first_package
      fourth_package
      second_package
      third_package
        include
        src
          service_5node_6node.srv
          CMakeLists.txt
          package.xml
          CMakeLists.txt
1 cmake_minimum_required(VERSION 2.8.3)
2 project(third_package)
3
4
5
6 find_package(catkin REQUIRED COMPONENTS
7   roscpp
8   std_msgs
9   message_generation
10 )
11
12 add_service_files(
13   FILES
14   service_5node_6node.srv
15 )
16
17 generate_messages(
18   DEPENDENCIES
19   std_msgs
20 )
21
22
23 catkin_package(
24   CATKIN_DEPENDS message_runtime
25 )
26
27
28
29 include_directories(
30   # include
31   ${catkin_INCLUDE_DIRS}
32 )
33
34
35
```




编译通过之后，新创建的服务就可以用了

北京维尔玛科技有限公司



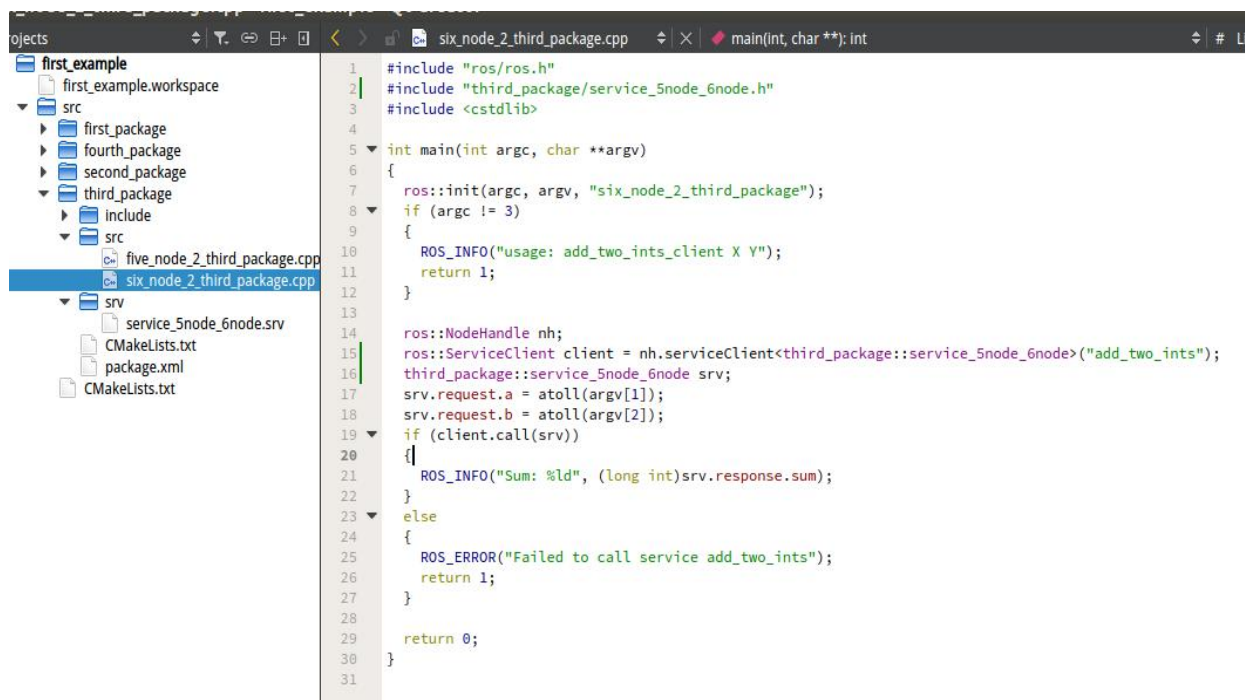
4.2.8. 采用新建的服务文件编写服务器节点和客户端节点：

右键单击 third_package 下的 src 文件夹，选择 Add New,弹出对话框：

选择 Basic Service Node，单击 Choose 进入下一步，输入服务器节点名称及保存路径：

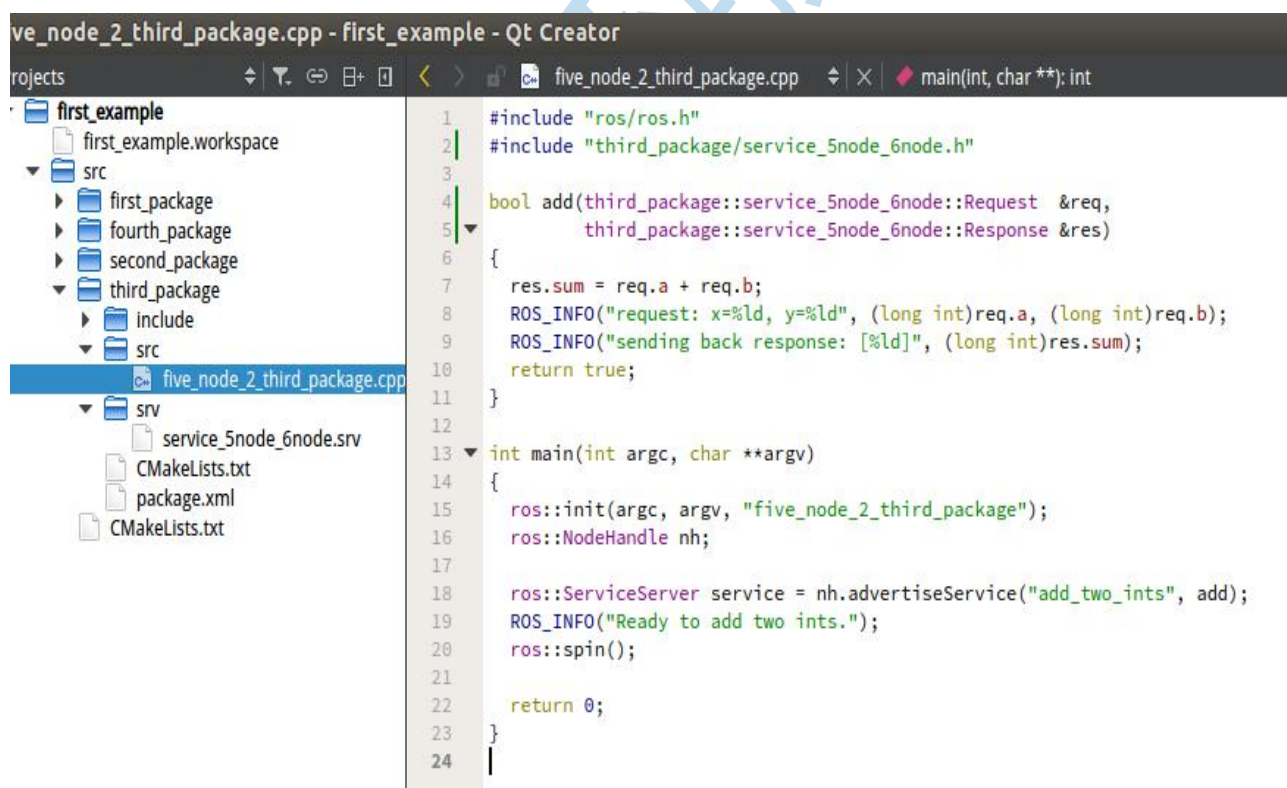
The screenshot shows a window titled "Basic Service Node". On the left, there are two tabs: "Location" (selected) and "Summary". Under the "Location" tab, there are two input fields: "Name:" with the text "five_node_2_third_package" and "Path:" with the text "/home/verma-robot/catkin_ws/src/third_package/src/". To the right of the "Path:" field is a "Browse..." button. At the bottom right of the window, there are two buttons: "Next >" and "Cancel".

节点创建完成之后修改节点文件代码如下：



```
1 #include "ros/ros.h"
2 #include "third_package/service_5node_6node.h"
3 #include <cstdlib>
4
5 int main(int argc, char **argv)
6 {
7     ros::init(argc, argv, "six_node_2_third_package");
8     if (argc != 3)
9     {
10         ROS_INFO("usage: add_two_ints_client X Y");
11         return 1;
12     }
13
14     ros::NodeHandle nh;
15     ros::ServiceClient client = nh.serviceClient<third_package::service_5node_6node>("add_two_ints");
16     third_package::service_5node_6node srv;
17     srv.request.a = atoll(argv[1]);
18     srv.request.b = atoll(argv[2]);
19     if (client.call(srv))
20     {
21         ROS_INFO("Sum: %ld", (long int)srv.response.sum);
22     }
23     else
24     {
25         ROS_ERROR("Failed to call service add_two_ints");
26         return 1;
27     }
28
29     return 0;
30 }
31
```

同样的方法，新建节点 six_node_2_third_package (six_node_2_third_package 为 Basic Client Node) ,节点代码如下：



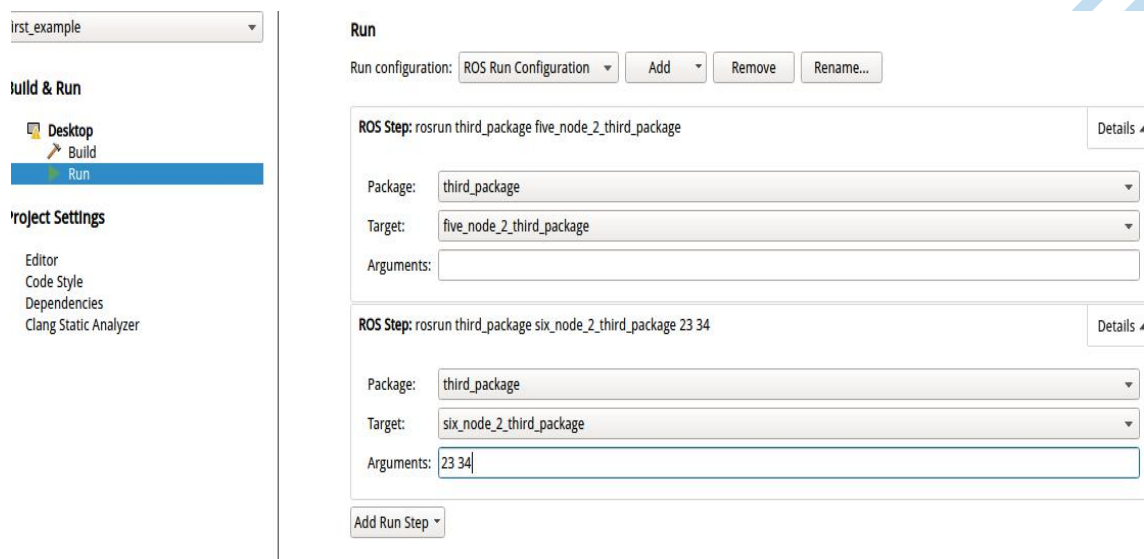
```
ve_node_2_third_package.cpp - first_example - Qt Creator
1 #include "ros/ros.h"
2 #include "third_package/service_5node_6node.h"
3
4 bool add(third_package::service_5node_6node::Request &req,
5         third_package::service_5node_6node::Response &res)
6 {
7     res.sum = req.a + req.b;
8     ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.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, "five_node_2_third_package");
16     ros::NodeHandle nh;
17
18     ros::ServiceServer service = nh.advertiseService("add_two_ints", add);
19     ROS_INFO("Ready to add two ints.");
20     ros::spin();
21
22     return 0;
23 }
24
```




修改 CMakeLists.txt 文件，在末端加入如下几行：

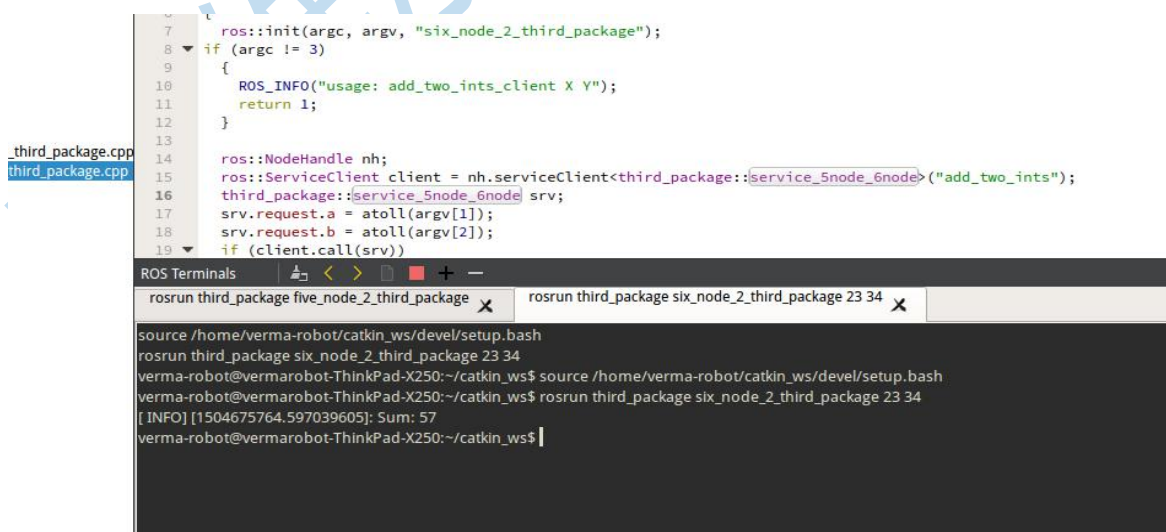
```
add_executable(five_node_2_third_package src/five_node_2_third_package)
target_link_libraries(five_node_2_third_package ${catkin_LIBRARIES})
add_executable(six_node_2_third_package src/six_node_2_third_package)
target_link_libraries(six_node_2_third_package ${catkin_LIBRARIES})
```

编译，配置 run：



可以看到，我们在 ROS Step:roslaunch third_package six_node_2_third_package 中的 Arguments 中添加了两个参数：23 和 34

执行结果：





北京维尔玛科技有限公司