

「ROS」 学习笔记

陈策

版本：0.0

更新：2019 年 11 月 21 日



清華大學

表 1: 中文术语表 A

英文术语	中文术语
Node	节点
Master	主节点
Package	功能包
Metapackage	元功能包
Dependent Package	依赖包
Message	消息
Service	服务
Topic	话题
Service Server	服务服务器
Service Client	服务客户端
Parameter Server	参数服务器
Publisher	发布者
Subscriber	订阅者

表 2: 中文术语表 B

英文术语	中文术语
Launch	启动
Client Library	客户端库
Repositories	存储库
Namespace	命名空间
Base Name	基本名称
Global Name	全局名称
Private Name	私有名称
Relative Name	相对名称
Node Handle	节点句柄
Timer	计时器
Transform	变换
Master PC	总机
Host PC	主机
Build	构建

表 3: 中文术语表 C

英文术语	中文术语
Motion Planning	运动规划
Odometry	侧位
Pose	姿态
Play/Replay	回放
Introspection	自检
Icon	图标
Dead Reckoning	导航推测
Base	基座
Link	连杆
Joint	关节
End Effector	末端执行器

1 机器人软件平台

1.1 平台的组件

1.2 机器人软件平台

1.3 机器人软件平台的必要性

1.4 机器人软件平台将带来的未来

2 机器人操作系统 ROS

2.1 ROS 简介

2.2 元操作系统

2.3 ROS 的目的

2.4 ROS 的组件

2.5 ROS 的生态系统

2.6 ROS 的历史

2.7 ROS 的版本

2.7.1 版本规则

2.7.2 版本周期

2.7.3 选择版本

3 搭建 ROS 开发环境

3.1 安装 ROS

3.1.1 常规安装

3.1.2 简易安装

3.2 搭建 ROS 开发环境

3.2.1 ROS 配置

3.2.2 集成开发环境 (IDE)

3.3 ROS 操作测试

4 ROS 的重要概念

4.1 ROS 术语

4.1.1 ROS

ROS 是一个用于开发机器人应用程序的、类似操作系统的机器人软件平台。

4.1.2 主节点

主节点 (master) 负责节点到节点的连接和消息通信，类似于名称服务器 (Name Server)。

roscore 是它的运行命令，当您运行主节点时，可以注册每个节点的名字，并根据

需要获取信息。

没有主节点，就不能在节点之间建立访问和消息交流（如话题和服务）。

主节点使用 XML 远程过程调用（XML-RPC，XML-Remote Procedure Call）与节点进行通信。XMLRPC 是一种基于 HTTP 的协议，主节点不与连接到主节点的节点保持连接。换句话说，节点只有在需要注册自己的信息或向其他节点发送请求信息时才能访问主节点并获取信息。通常情况下，不检查彼此的连接状态。

当启动 ROS 时，主节点将获取用户设置的 `ROS_MASTER_URI` 变量中列出的 URI 地址和端口。除非另外设置，默认情况下，URI 地址使用当前的本地 IP，端口使用

11311。

4.1.3 节点

节点 (node) 是指在 ROS 中运行的最小处理器单元，可以把它看作一个可执行程序。

在 ROS 中，建议为一个目的创建一个节点，建议设计时注重可重用性。

节点在运行的同时，向主节点注册节点的名称，并且还注册发布者 (publisher)、订阅者 (subscriber)、服务服务器 (service server)、服务客户端 (service client) 的名称，且注册消息形式、URI 地址和端口。

基于这些信息，每个节点可以使用话

题和服务与其他节点交换消息。

节点使用 XMLRPC 与主站进行通信，并使用 TCP/IP 通信系列的 XMLRPC 或 TCPROS5 进行节点之间的通信。节点之间的连接请求和响应使用 XMLRPC，而消息通信使用 TCPROS，因为它是节点和节点之间的直接通信，与主节点无关。

URI 地址和端口则使用存储于运行当前节点的计算机上的名为 ROS_HOSTNAME 的环境变量作为 URI 地址，并将端口设置为任意的固有值。

4.1.4 功能包

功能包 (package) 是构成 ROS 的基本单元，ROS 应用程序是以功能包为单位开发

的。

功能包包括至少一个以上的节点或拥有用于运行其他功能包的节点的配置文件。它还包含功能包所需的所有文件，如用于运行各种进程的 ROS 依赖库、数据集和配置文件等。

4.1.5 元功能包

元功能包（metapackage）是一个具有共同目的的功能包的集合。

4.1.6 消息

节点之间通过消息（message）来发送和接收数据。

消息是诸如 integer、floating point 和 boolean 等类型的变量。

用户还可以使用诸如消息里包括消息的简单数据结构或列举消息的消息数组的结构。

使用消息的通信方法包括 TCPROS, UDPROS 等, 根据情况使用单向消息发送/接收方式的话题 (topic) 和双向消息请求 (request) /响应 (response) 方式的服务 (service)。

4.1.7 话题

话题 (topic) 就是“故事”。在发布者 (publisher) 节点关于故事向主节点注册之后, 它以消息形式发布关于该故事的广告。

希望接收该故事的订阅者 (subscriber) 节点获得在主节点中以这个话题注册的那个发布者节点的信息。基于这个信息，订阅者节点直接连接到发布者节点，用话题发送和接收消息。

4.1.8 发布与发布者

发布 (publish) 是指以与话题的内容对应的消息的形式发送数据。

为了执行发布，发布者 (publisher) 节点在主节点上注册自己的话题等多种信息，并向希望订阅的订阅者节点发送消息。

发布者在节点中声明自己是执行发布的个体。单个节点可以成为多个发布者。

4.1.9 订阅与订阅者

订阅是指以与话题内容对应的消息的形式接收数据。

为了执行订阅，订阅者节点在主节点上注册自己的话题等多种信息，并从主节点接收那些发布此节点要订阅的话题的发布者节点的信息。基于这个信息，订阅者节点直接联系发布者节点来接收消息。

订阅者在节点中声明自己执行订阅的个体。单个节点可以成为多个订阅者。

注意：发布和订阅概念中的话题是异步的，这是一种根据需要发送和接收数据的好方法。另外，由于它通过一次连接，发送和接收连续的消息，所以它经常被用

于必须连续发送消息的传感器数据。然而，在某些情况下，需要一种共同使用请求和响应的同步消息交换方案。因此，ROS 提供叫做服务（service）的消息同步方法。服务分为响应请求的服务服务器和请求后接收响应的服务客户端。与话题不同，服务是一次性的消息通信。当服务的请求和响应完成时，两个节点的连接被断开。

4.1.10 服务

服务（service）消息通信是服务客户端（service client）与服务服务器（service server）之间的同步双向消息通信。其中服务客户端请求对应于特定目的的任务的服务，而服务服务器则负责服务响应。

4.1.11 服务服务器

服务服务器（service server）是以请求作为输入，以响应作为输出的服务消息通信的服务器。

请求和响应都是消息，服务器收到服务请求后，执行指定的服务，并将结果下发给服务客户端。

服务服务器用于执行指定命令的节点。

4.1.12 服务客户端

服务客户端（service client）是以请求作为输出并以响应作为输入的服务消息通信的客户端。

请求和响应都是消息，并发送服务请求到服务服务器后接收其结果。

服务客户端用于传达给定命令并接收结果值的节点。

4.1.13 动作

动作（action）是在需要像服务那样的双向请求的情况下使用的消息通信方式，不同点是在处理请求之后需要很长的响应，并且需要中途反馈值。

动作文件也非常类似于服务，目标（goal）和结果（result）对应于请求和响应。此外，还添加了对应于中途的反馈（feedback）。它由一个设置动作目标（goal）的动作客户端（action client）和一个动作服务器（action

server) 组成，动作服务器根据目标执行动作，并发送反馈和结果。

动作客户端和动作服务器之间进行异步双向消息通信。

4.1.14 动作服务器

动作服务器 (action server) 以从动作客户端接收的目标作为输入并且以结果和反馈值作为输出的消息通信的服务器。

在接收到来自客户端的目标值后，负责执行实际的动作。

4.1.15 动作客户端

动作客户端 (action client) 是以目标作为输出并以从动作服务器接收待结果和反馈值作为输入的消息通信的客户端。

它将目标交付给动作服务器，收到结果和反馈，并给出下一个指示或取消目标。

4.1.16 参数

ROS 中的参数 (parameter) 是指节点中使用的参数。可以把它想象成一个 Windows 程序中的 *.ini 配置文件。

这些参数是默认 (default) 设置的，可以根据需要从外部读取或写入。尤其是，它可以通过使用外部的写入功能实时更改

设置值，因此非常有用。

4.1.17 参数服务器

参数服务器（parameter server）是指在功能包中使用参数时，注册各参数的服务器。

参数服务器也是主节点的一个功能。

4.1.18 catkin

catkin 是指 ROS 的构建系统。

ROS 的构建系统使用 CMake（Cross Platform Make），并在功能包目录中的 CMakeLists.txt 文件中描述构建环境。

在 ROS 中，我们将 CMake 修改成专为

ROS 定制的 catkin 构建系统。

catkin 构建系统让用户方便使用与 ROS 相关的构建、功能包管理以及功能包之间的依赖关系等。现在使用 ROS 的话，需要使用 catkin 而不是 rosbuidl。

4.1.19 rosbuidl

ROS 构建 (rosbuidl) 是在构建 catkin 构建系统之前使用的构建系统，虽然仍有一些用户可以使用，但这只是为 ROS 版本兼容性保留的，并不是官方推荐的。

如果您必须使用 rosbuidl 构建系统使用旧的功能包，我们建议您将 rosbuidl 更改为 catkin。

4.1.20 roscore

roscore 是运行 ROS 主节点的命令，也可以在另一台位于同一个网络内的计算机上运行它，但除了支持多 roscore 的某些特殊情况，roscore 在一个网络中只能运行一个。

运行 ROS 时，使用 ROS_MASTER_URI 变量中列出的 URI 地址和端口。

如果用户没有设置，会使用当前本地 IP 作为 URI 地址并使用端口 11311。

4.1.21 rosrun

roslaunch 是 ROS 的基本运行命令，它用于在功能包中运行一个节点。

节点使用的 URI 地址将存储在当前运行节点的计算机上的 `ROS_HOSTNAME` 环境变量作为 URI 地址，端口被设置为任意的固有值。

4.1.22 `roslaunch`

如果 `roslaunch` 是执行一个节点的命令，那么 `roslaunch` 是运行多个节点的概念。

该命令允许运行多个确定的节点。其他功能还包括一些专为执行具有诸多选项的节点的 ROS 命令。`roslaunch` 使用 *.launch 文件来设置可执行节点，它基于可扩展标记语言 (XML)，并提供 XML 标记形式的多种选项。

4.1.23 bag

用户可以保存 ROS 中发送和接收的消息的数据，这时用于保存的文件格式称为 bag，是以 *.bag 作为扩展名。

在 ROS 中，这个功能包可以用来存储信息并在需要时可以回放以前的情况。

特别的，如果利用 rosbag 的记录和回放功能，在开发那些需要反复修改程序的算法的时候会非常有用。

4.1.24 ROS Wiki

ROS 的基本说明是一个基于 wiki 的页面，它解释了 ROS 提供的每个功能包和功能。

4.1.25 存储库

每一个公开的功能包在该功能包的 wiki 上指定一个存储库 (repository)。

存储库是存储功能包的网站的 URL 地址，并使用源代码管理系统（如 svn、hg 和 git）来管理问题、开发、下载等。许多当前可用的 ROS 功能包将 github 用作存储库。

4.1.26 状态图

上面描述的节点、话题、发布者和订阅者之间关系可以通过状态图 (graph) 直观地表示，它是当前正在运行的消息通信的图形表示，但不能为一次性服务创建状态图。

执行它是通过运行 `rqt - graph` 功能包的 `rqt - graph` 节点完成的。有两种执行命令：`rqt - graph` 和 `roslaunch rqt - graph rqt - graph`。

4.1.27 名称

节点、参数、话题和服务都有名称 (name)。

当使用主节点参数、话题和服务时，向主节点注册该名称并根据名称进行搜索，然后发送消息。

名称非常灵活，它们可以在运行时被更改，对于一个节点、参数、话题和服务，也能给其设定多个不同的名称。

4.1.28 客户端库

ROS 是一个客户端库 (client library)，它为各种语言提供开发环境，以减少对所用语言的依赖性。主要的客户端库包括 C++、Python 和 Lisp。其他语言包括 Java、Lua、.NET、EusLisp 和 R。

4.1.29 URI

统一资源标识符 (URI, Uniform Resource Identifier) 是代表 Internet 上资源的唯一地址。

该 URI 被用作 Internet 协议中的标识符，是在 Internet 上所需的基本条件。

4.1.30 MD5

MD5 (Message-Digest algorithm 5) 是 128 位密码散列函数，它主要用于检查程序或文件的完整性，以查看它是否保持原样。

在使用 ROS 消息的通信中，使用 MD5 来检查消息发送/接收的完整性。

4.1.31 RPC

远程过程调用 (RPC, Remote Procedure Call) 意味着远程 (Remote) 计算机上的程序调用 (Call) 另一台计算机中的子程序 (Procedure)。

这个利用 TCP/IP、IPX 等传输协议的技术在不需要程序员一一进行编程的情况下

也能允许计算机在另一个地址空间通过远程控制运行函数或子程序。subsubsectionXML 可扩展标记语言 (XML, Extensible Markup Language) 是 W3C 推荐用于创建其他特殊用途标记语言的通用标记语言, 它是通过使用标签来指定数据结构的语言之一。

在 ROS 中用于 *.launch、*.urdf 和 package.xml 等各个部分。

4.1.32 XMLRPC

XML-Remote Procedure Call (XMLRPC) 是一种 RPC 协议, 其编码形式采用 XML 编码格式, 而传输方式采用既不保持连接状态、也不检查连接状态的请求和响应方式的 HTTP 协议。

XMLRPC 是一个非常简单的约定，仅用于定义小数据类型或命令，所以它比较简单。有了这个特点，XMLRPC 非常轻便，支持多种编程语言，因此非常适合支持各种硬件和语言的 ROS。

4.1.33 TCP/IP

传输控制协议（TCP，Transmission Control Protocol）是一种传输控制协议，通常被称为 TCP/IP。

从互联网协议层的角度来看，它基于 IP（Internet Protocol）且使用传输控制协议 TCP，以此保证数据传输，并按照发送顺序进行发送/接收。

TCPROS 消息和服务中使用的基于 TCP/

IP 的消息方式称为 TCPROS，而 UDPROS 消息及服务中使用的基于 UDP 的消息方式称为 UDPROS。

在 ROS 中，常用的是 TCPROS。

4.1.34 CMakeLists.txt

ROS 构建系统的 catkin 基本上使用了 CMake，因此在功能包目录的 CMakeLists.txt 文件中描述着构建环境。subsubsectionpackage.xml 包含功能包信息的 XML 文件，描述功能包名称、作者、许可证和依赖包。

4.2 消息通信

ROS 以节点的形式开发的，节点是根据其目的细分的可执行程序的最小单位，节点通过消息（message）与其他的节点交换数据。

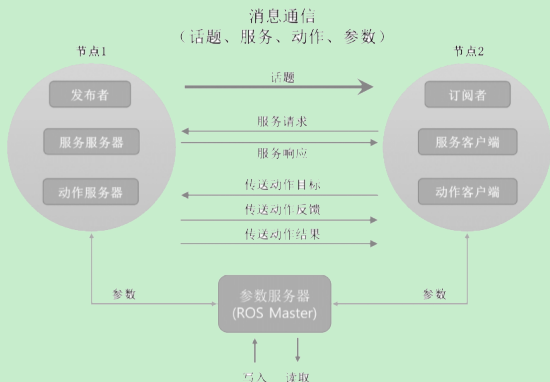


图 1: 节点间的消息通信

这里的关键概念是节点之间的消息通信，它分为三种：单向消息发送/接收方式的话题（topic）；双向消息请求/响应方式的服务（service）；双向消息目标（goal）/结果（result）/反馈（feedback）方式的动作（action）。

种类，时序，方向，区别

话题，异步，单向，连续单向地发送/接收数据的情况

服务，同步，双向，需要对请求给出即时响应的情况

动作，异步，双向，请求与响应之间需要太长的时间，所以难以使用服务的情况，或需要中途反馈值的情况

表 4: 话题、服务和动作之间的差异

节点中使用的参数可以从外部进行修改，这在大的框架中也可以被看作消息通信。

4.2.1 话题 (topic)

话题消息通信是指发送信息的发布者和接收信息的订阅者以话题消息的形式发送和接收信息。

希望接收话题的订阅者节点接收的是与在主节点中注册的话题名称对应的发布者节点的信息。基于这个信息，订阅者节点直接连接到发布者节点来发送和接收消息。

话题是单向的，适用于需要连续发送消息的传感器数据，因为它们通过一次连

接连续发送和接收消息。

单个发布者可以与多个订阅者进行通信，一个订阅者也可以在单个话题上与多个发布者进行通信。

4.2.2 服务 (service)

服务消息通信是指请求服务的服务客户端与负责服务响应的服务服务器之间的同步双向服务消息通信。

前述的发布和订阅概念的话题通信方法是一种异步方法，是根据需要传输和接收给定数据的一种非常好的方法。然而，在某些情况下，需要一种同时使用请求和响应的同步消息交换方案。因此，ROS 提供叫做服务的消息同步方法。

一个服务被分成服务服务器和服务客户端，其中服务服务器只在有请求（request）的时候才响应（response），而服务客户端会在发送请求后接收响应。

与话题不同，服务是一次性消息通信。因此，当服务的请求和响应完成时，两个连接的节点将被断开。

服务通常被用作请求机器人执行特定操作时使用的命令，或者用于根据特定条件需要产生事件的节点。由于它是一次性的通信方式，又因为它在网络上的负载很小，所以它也被用作代替话题的手段，是一种非常有用的通信手段。

4.2.3 动作 (action)

动作消息通信是在如下情况使用的消息通信方式：服务器收到请求后直到响应所需的时间较长，且需要中途反馈值。

反馈在动作客户端 (action client) 和动作服务器 (action server) 之间执行异步双向消息通信，其中动作客户端设置动作目标 (goal)，而动作服务器根据目标执行指定的工作，并将动作反馈和动作结果发送给动作客户端。

与服务不同，动作通常用于指导复杂的机器人任务。在这种情况下，发布者、订阅者、服务服务器、服务客户端、动作服务器和动作客户端都存在于不同的节点中，这些节点需要连接才能进行消息通信。

这时候，主节点帮助节点进行相互连接换，节点同时向主节点注册自己的信息，并从主节点获取其他节点希望通过主节点访问的节点的信息，然后，节点和节点直接连接进行消息通信。

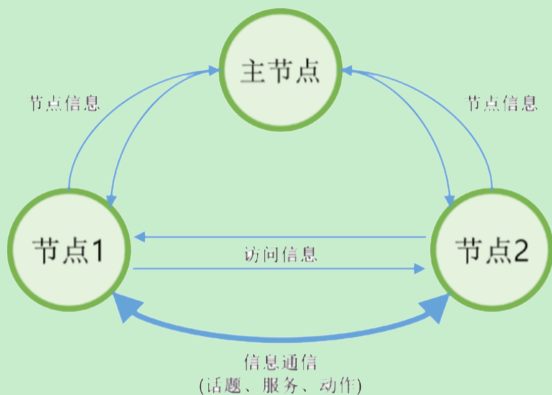


图 2: 消息通信

4.2.4 参数 (parameter)

信息通信主要分为话题、服务和动作，尽管参数严格的来说并不是消息通信，但从大的框架来看，参数也可以看作一种消息通信。

可以认为参数是节点中使用的全局变量。参数的用途与 Windows 程序中的 *.ini 配置文件非常类似。默认情况下，这些设置值是指定的，有需要时可以从外部读取或写入参数。特别是，由于可以通过使用来自外部的写入功能来实时地改变设置值，因此它是非常有用的，因为它可以灵活地应对多变的情况。

4.2.5 消息通信的过程

运行主节点：ROS 主节点使用 `roscore` 命令来运行，并使用 XMLRPC 运行服务器。主节点为了节点与节点的连接，会注册节点的名称、话题、服务、动作名称、消息类型、URI 地址和端口，并在有请求时将此信息通知给其他节点。

```
$ roscore
```

运行订阅者节点：订阅者节点使用 `roslaunch` 或 `roslaunch` 命令来运行。订阅者节点在运行时向主节点注册其订阅者节点名称、话题名称、消息类型、URI 地址和端口。主节点和节点使用 XMLRPC 进行通信。

```
$ roslaunch PACKAGE_NAME NODE_NAME
```

```
$ roslaunch PACKAGE_NAME LAUNCH_NAME
```

运行发布者节点：发布者节点（与订阅者节点类似）使用 `roslaunch` 或 `roslaunch` 命令来运行。发布者节点向主节点注册发布者节点名称、话题名称、消息类型、URI 地址和端口。主节点和节点使用 XMLRPC 进行通信。

```
$ roslaunch PACKAGE_NAME NODE_NAME
```

```
$ roslaunch PACKAGE_NAME LAUNCH_NAME
```

通知发布者信息：主节点向订阅者节点发送此订阅者希望访问的发布者的名称、话题名称、消息类型、URI 地址和端口等信息。主节点和节点使用 XMLRPC 进行通信。

订阅者节点的连接请求：订阅者节点

根据从主节点接收的发布者信息，向发布者节点请求直接连接。在这种情况下，要发送的信息包括订阅者节点名称、话题名称和消息类型。发布者节点和订阅者节点使用 XMLRPC 进行通信。

发布者节点的连接响应：发布者节点将 TCP 服务器的 URI 地址和端口作为连接响应发送给订阅者节点。发布者节点和订阅者节点使用 XMLRPC 进行通信。

TCPROS 连接：订阅者节点使用 TCPROS 创建一个与发布者节点对应的客户端，并直接与发布者节点连接。节点间通信使用一种称为 TCPROS 的 TCP/IP 方式。

发送消息：发布者节点向订阅者节点发送消息。节点间通信使用一种称为

TCPROS 的 TCP/IP 方式。

服务请求及响应：上述内容相当于消息通信中的话题。话题消息通信是只要发布者或订阅者不停止，会持续地发布和订阅。服务分为两种：服务客户端（请求服务后等待响应）、服务服务器（收到服务请求后执行指定的任务，并发送响应），服务服务器和客户端之间的连接与上述发布者和订阅者之间的 TCPPROS 连接相同，但是与话题不同，服务只连接一次，在执行请求和响应之后彼此断开连接。如果有必要，需要重新连接。

动作的目标、结果和反馈：动作（action）在执行的方式上好像是在服务（service）的请求（goal）和响应（result）之间仅仅多了中

途反馈环节，但实际的运作方式与话题相同。事实上，如果使用 `rostopic` 命令来查阅话题，那么可以看到该动作的 `goal`、`status`、`cancel`、`result` 和 `feedback` 等五个话题。动作服务器和客户端之间的连接与上述发布者和订阅中的 TCPROS 连接相同，但某些用法略有不同，动作客户端发送取消命令或服务器发送结果值会中断连接。

4.3 消息

消息 (message) 是用于节点之间的数据交换的一种数据形式。话题、服务和动作都使用消息。消息可以是简单的数据结构，或者是包含消息的简单的数据结构，也可以是消息数组结构。另外，ROS 中常用的

头 (header、std_msgs/Header) 也可以作为消息来使用。这些消息由两种类型组成：字段类型 (fieldtype) 和字段名称 (fieldname)。

```
fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3
```

字段类型应填入 ROS 数据类型，如下表所示。

ROS 数据类型	序列化 (Serialization)	C++ 数据类型	Python 数据类型
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int

表 5: 数据类型 A

ROS 数据类型	序列化 (Serialization)	C++ 数据类型	Python 数据类型
int/6	signed 16-bit int	int/6_t	int
uint/6	unsigned 16-bit int	uint/6_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string	std::string	str
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time

表 6: 数据类型 B

ROS 数据类型	序列化 (Serialization)	C++ 数据类型	Python 数据类型
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration
fixed-length	no extra serialization	boost::array, std::vector	tuple
variable-length	uint32 length prefix	std::vector	tuple
uint8[]	uint32 length prefix	std::vector	bytes
bool[]	uint32 length prefix	std::vector<uint8_t>	list of bool

表 7: 数据类型 C

字段名称要填入指示数据的名称。

前面讲到 ROS 中常用的头(header、std_msgs/Header) 可以作为消息来使用。更具体来说，正如 std_msgs32 的 Header.msg 文件中描述，会记录序列号、时间戳和框架 ID，利用它们在消息中记录消息的计数以及时间的计算。

std_msgs/Header.msg

序列号：是连续增加的ID，每个消息中依次递增+1。

uint32 seq

时间戳：具有两个子属性：以秒为单位的

stamp.sec和以纳秒为单位的stamp.nsec.

time stamp

记录框架ID。

string frame_id

4.3.1 Msg 文件

msg 文件是用于话题的消息文件，扩展名为 *.msg。geometry_msgs/Twist.msg

```
Vector3 linear
Vector3 angular
```

4.3.2 Srv 文件

srv 文件是服务使用的消息文件，扩展名为 *.srv。与 msg 文件的主要区别在于三个连字符（—）作为分隔符，上层消息是服务请求消息，下层消息是服务响应消息。

sensor_msgs/SetCameraInfo.srv

```
sensor_msgs/CameraInfo camera_info
---
bool success
```

```
string status\_message
```

4.3.3 Action 文件

action 消息文件是动作中使用的消息文件，它使用 *.action 扩展名。与 msg 和 srv 不同，它不是一个比较常见的消息文件，所以没有典型的官方消息文件，但是可以像下面的例子一样使用它。与 msg 和 srv 文件的主要区别在于，三个连字符（—）在两个地方用作分隔符，第一部分是 goal 消息，第二部分是 result 消息，第三部分是 feedback 消息。

```
geometry_msgs/PoseStamped start_pose
geometry_msgs/PoseStamped goal_pose
---
geometry_msgs/PoseStamped result_pose
```

float32 percent_complete

最大的区别是来自 action 文件的 feedback 信息。action 文件的 goal 消息和 result 消息与上述 srv 文件的请求消息和响应消息角色相同，但 action 文件的 feedback 消息用于传输指定进程执行过程中的中途值。

例子中，当机器人的出发点 start_pose 和目标点 goal_pose 的位置和姿态作为请求值被传送时，机器人移动到预定的目标位置并且将发送最终到达的 result_pose 的位置和姿态。另外，用 percent_complete 消息，周期性地发送到目标地点进程的百分比。

4.4 名称 (name)

ROS 有一个称为图 (graph) 的抽象数据类型作为其基本概念。这显示了每个节点的连接关系以及通过箭头表达发送和接收消息 (数据) 的关系。为此, 服务中使用的节点、话题、消息以及 ROS 中使用的参数都具有唯一的名称 (name)。

话题名称分为相对的方法、全局方法和私有方法, 用户可以在运行时更改节点的名称, 而不需要运行额外的程序或更改源代码。方法包括命名空间 (namespace) 和重新映射 (remapping)。

4.5 坐标变换 (TF)

ROS 中的坐标转换 TF 在描述组成机器人的每个部分、障碍物和外部物体时是最有用的概念之一。这些可以被描述为位置 (position) 和方向 (direction)，统称为姿态 (pose)。在此，位置由 x 、 y 、 z 这 3 个矢量表示，而方向是用四元数 (quaternion) x 、 y 、 z 、 w 表示。四元数并不直观，因为它们没有使用我们在日常生活中使用的三元数的角度表达方式：滚动角 (roll)、俯仰角 (pitch) 和偏航角 (yaw)。但这种四元数方式不存在滚动、俯仰和偏航矢量的欧拉 (Euler) 方式具有的万向节死锁 (gimbal lock) 问题或速度问题，因此在机器人工程中人们更喜欢用四元数 (quaternion) 的形式。因为同样的原因，ROS 中也大量使用四元数。当然，考虑

到方便，它也提供将欧拉值转换成四元数的功能。

类似上面说明的消息（message），TF 使用以下格式。

```
geometry_msgs/TransformStamped.msg
```

```
Header header  
string child_frame_id  
Transform transform
```

Header 用于记录转换的时间，并使用名为 `child_frame_id` 的消息来表示下位的坐标。

为了表达坐标的转换值，使用以下数据形式描述对方的位置和方向：

```
transform.translation.x
```

`transform.translation.y`

`transform.translation.z`

`transform.rotation.x`

`transform.rotation.y`

`transform.rotation.z`

`transform.rotation.w`

4.6 客户端库

编程语言多如繁星。为了更快的性能和更好地控制硬件，会选择 C++；为了更高的工作效率，会选择 Python 和 Ruby。由于 ROS 需要支持具有多重目的特点的机器人，因此 ROS 提供面向多种语言的便利接

口。具体来说，各个节点可以用各自的语言来编写，而节点间通过消息通信交换信息。其中，允许用各自的语言编写的软件模块就是客户端库（client library）。常用且具有代表性的库有支持 C++ 的 roscpp、支持 Python 的 rospy。

4.7 异构设备间的通信

ROS 基本上支持异构设备间的通信。节点间的通信不受各节点所在的 ROS 底层的操作系统的种类的影响，也不受编程语言的影响，因此可以很容易地进行通信。

4.8 文件系统

4.8.1 文件组织结构

在 ROS 中，组成软件的基本单位是功能包（package），因此 ROS 应用程序是以功能包为单位开发的。功能包包含一个以上的节点（node，ROS 中最小的执行处理器）或包含用于运行其他节点的配置文件。这些功能包也会以元功能包（metapackage）的形式来统一管理。元功能包是具有共同目的的功能包的集合体。每个功能包都包含一个名为 package.xml 的文件，该文件是一个包含功能包信息的 XML 文件，包括其名称，作者，许可证和依赖包。此外，ROS 构建系统 catkin 基本上使用 CMake，并在功能包目录中的 CMakeLists.txt 文件中描述构建环境。另外，它由节点的源代码和消息文件组成，用于节点之间的消息通信。

ROS 的文件系统分为安装目录和用户工作目录。

安装 ROS desktop 版本后，在 /opt 目录中会自动生成名为 ros 的安装目录，里面会安装有 roscore、rqt、RViz、机器人相关库、仿真和导航等核心实用程序。用户很少需要修改这个区域的文件。如果要修改以二进制文件形式分发的功能包，请找到包含源代码的功能包存储库之后利用在 “/catkin_ws/src” 目录下用 “git clone [存储库地址]” 命令直接复制源代码，而不是用 “sudo apt-get install ros-kinetic-xxx” 形式的功能包安装命令。

用户的工作目录可以在用户想要的位置创建，下面我们就使用 Linux 用户目录 “/

catkin_ws/（在 Linux 中，‘/’指‘/home/用户名/’目录）”。

ROS 功能包的安装方式有二进制文件安装和源代码安装两种。二进制文件安装是使用二进制形式的文件，无需额外的构建，而源代码安装是下载该功能包的源代码之后由用户进行构建之后使用的方式。根据功能包使用的目的不同选择不同的方式。如果需要修改安装包或者要检查源代码的内容，可以使用后一种安装方法。下面用 turtlebot3 功能包的例子，介绍两种安装方法的不同之处。

1. 二进制文件安装

```
$ sudo apt-get install ros-kinetic-  
turtlebot3
```

2. 源代码安装

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/ROBOTIS-  
    GIT/turtlebot3.git  
$ cd ~/catkin_ws/  
$ catkin_make
```

4.8.2 安装目录

ROS 安装在 “/opt/ros/[版本名称]” 目录中。

“/opt/ros/kinetic” 目录下包含 bin、etc、include、lib、share 目录和一些配置文件。

ROS 目录包含用户在安装 ROS 时选择的功能包和 ROS 运行程序。详情如下：

/bin 可执行的二进制文件

/etc 与 ROS 和 catkin 相关的配置文件

/include 头文件

/lib 库文件

/share ROS 功能包

env.* 配置文件

setup.* 配置文件

4.8.3 工作目录

用户可以在任意位置创建工作目录，但是为了方便，本书中将 Linux 用户目录“/catkin_ws/”用作工作目录。也就是说，您将使用“/home/用户名/catkin_ws”目录。

catkin_ws 的目录，由目录 build、devel 和 src 组成。请注意，build 和 devel 目录是在 catkin_make 之后创建的。

工作目录是对用户创建的功能包和其他开发人员公开的功能包进行存储和构建的空间。用户在该目录中执行与 ROS 有关的大部分操作。详情如下：

/build 构建相关的文件

/devel msg、srv 头文件、用户包库、可执行文件

/src 用户功能包

目录 “/catkin_ws/src” 是用户源代码的空间。在这个目录中，用户可以保存和建立自己的 ROS 功能包或其他开发者开发的

功能包。

下面列举了通常使用的目录和文件，但其文件组成会根据功能包的用途而有所不同：

/include 头文件

/launch 用于 roslaunch 的启动文件

/node 用于 rospy 的脚本

/msg 消息文件

/src 源代码文件

/srv 服务文件

CMakeLists.txt 构建配置文件

package.xml 功能包配置文件

4.9 构建系统

ROS 的构建系统默认使用 CMake (Cross Platform Make)，其构建环境在功能包目录中的 CMakeLists.txt 文件中描述。在 ROS 中，CMake 被修改为适合于 ROS 的“catkin”构建系统。在 ROS 中使用 CMake 的是为了在多个平台上构建 ROS 功能包。因为不同于只支持 Unix 系列的 Make，CMake 支持 Unix 类的 Linux、BSD 和 OS X 以外，还支持 Windows 系列。并且，它还支持 Microsoft Visual Studio，也还可以轻松应用于 Qt 开发。此外，catkin 构建系统可以轻松使用与 ROS 相关的构建、功能包管理和功能包之间的依赖关系。

4.9.1 创建功能包

创建 ROS 功能包的命令如下。

```
$ catkin_create_pkg [功能包名称] [依赖功能包1] [依赖功能包n]
```

“catkin_create_pkg”命令在创建用户功能包时会生成 catkin 构建系统所需的 CMakeLists.txt 和 package.xml 文件的包目录。

ROS 中的功能包名称全部是小写字母，不能包含空格。格式规则是将每个单词用下划线（_）而不是短划线（-）连接起来。

4.9.2 修改功能包配置文件 (package.xml)

必要的 ROS 配置文件之一的 package.xml 是一个包含功能包信息的 XML 文件，包括

功能包名称、作者、许可证和依赖功能包。

下面是对每个语句的说明：

`<?xml>` 这是一个定义文档语法的语句，随后的内容表明在遵循 xml 版本 1.0。

`<package>` 从这个语句到最后 `</package>` 的部分是 ROS 功能包的配置部分。

`<name>` 功能包的名称。使用创建功能包时输入的功能包名称。正如其他选项，用户可以随时更改。

`<version>` 功能包的版本。可以自由指定。

`<description>` 功能包的简要说明。通常用两到三句话描述。

`<maintainer>` 提供功能包管理者的姓名

和电子邮件地址。

`<license>` 记录版权许可证。写 BSD、MIT、Apache、GPLv3 或 LGPLv3 即可。

`<url>` 记录描述功能包的说明，如网页、错误管理、存储库的地址等。根据功能包的类型，用户可以填写网站、错误跟踪（bugtracker）或存储库的地址。

`<author>` 记录参与功能包开发的开发人员的姓名和电子邮件地址。如果涉及多位开发人员，只需在下一行添加 `<author>` 标签。

`<buildtool_depend>` 描述构建系统的依赖关系。我们正在使用 catkin 构建系统，因此输入 catkin。

`<build_depend>` 在编写功能包时写下您所依赖的功能包的名称。

`<run_depend>` 填写运行功能包时依赖的功能包的名称。

`<test_depend>` 填写测试功能包时依赖的功能包名称。

`<export>` 在使用 ROS 中未指定的标签名称时会用到 `<export>`。最广泛使用的情况是元功能包的情况，这时用 `<export> <metapackage/> </export>` 格式表明是元功能包。

`<metapackage>` 在 `export` 标签中使用的官方标签声明，当前功能包为一个元功能包时声明它。

以下为一个 Demo：

```
<?xml version="1.0"?>
<package>
<name>my_first_ros_pkg</name>
<version>0.0.1</version>
<description>The my_first_ros_pkg package
    </description>
<license>Apache License 2.0</license>
<author email="pyo@robotis.com">Yoonseok
    Pyo</author>
<maintainer email="pyo@robotis.com">
    Yoonseok Pyo</maintainer>
<url type="bugtracker">https://github.com
    /ROBOTIS-GIT/ros_turtorials/issues</
    url>
<url type="repository">https://github.com
    /ROBOTIS-GIT/ros_turtorials.git</url>
<url type="website">http://www.robotis.
    com</url>
<buildtool_depend>catkin</
    buildtool_depend>
```

```
<build_depend>std_msgs</build_depend>
<build_depend>roscpp</build_depend>
<run_depend>std_msgs</run_depend>
<run_depend>roscpp</run_depend>
<export></export>
</package>
```

4.9.3 修改构建配置文件 (CMakeLists.txt)

ROS 的构建系统 catkin 基本上使用 CMake，并在功能包目录中的 CMakeLists.txt 文件中描述构建环境。在这个文件中设置可执行文件的创建、依赖包优先构建、连接器 (linker) 的创建等等。

构建配置文件 (CMakeLists.txt) 中的每一项如下所示。

第一条是操作系统中安装的 `cmake` 的最低版本。由于它目前被指定为版本 2.8.3，所以如果使用低于此版本的 `cmake`，则必须更新版本。

```
cmake_minimum_required(VERSION 2.8.3)
```

`project` 项是功能包的名称。只需使用用户在 `package.xml` 中输入的功能包名即可。请注意，如果功能包名称与 `package.xml` 中的 `<name>` 标记中描述的功能包名称不同，则在构建时会发生错误，因此需要注意。

```
project(my_first_ros_pkg)
```

`find_package` 项是进行构建所需的组件包。目前，`roscpp` 和 `std_msgs` 被添加为依赖包。如果此处没有输入功能包名称，则在

构建时会向用户报错。换句话说，这是让用户先创建依赖包的选项。

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)
```

以下是使用 ROS 以外的功能包时使用的方法。例如，使用 Boost 时，必须安装 system 功能包。功能如前面的说明，是让用户先创建依赖功能包的选项。

```
find_package(Boost REQUIRED COMPONENTS
  system)
```

catkin - python - setup() 选项是在使用 Python，也就是使用 rospy 时的配置选项。其功能是调用 Python 安装过程 setup.py。

```
catkin_python_setup()
```

`add_message_files` 是添加消息文件的选项。FILES 将引用当前功能包目录的 msg 目录中的 *.msg 文件，自动生成一个头文件 (*.h)。在这个例子中，我们将使用消息文件 Message1.msg 和 Message2.msg。

```
add_message_files(  
  FILES  
  Message1.msg  
  Message2.msg  
)
```

`add_service_files` 是添加要使用的服务文件的选项。使用 FILES 会引用功能包目录中的 srv 目录中的 *.srv 文件。在这个例子中，用户可以选择使用服务文件 Service1.srv

和 Service2.srv。

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

`generate_messages` 是设置依赖的消息的选项。此示例是将 `DEPENDENCIES` 选项设置为使用 `std_msgs` 消息包。

```
generate_messages(  
  DEPENDENCIES  
  std_msgs  
)
```

`generate_dynamic_reconfigure_options` 是使用 `dynamic_reconfigure` 时加载要引用的配置文件的设置。

```
generate_dynamic_reconfigure_options(  
  cfg/DynReconf1.cfg  
  cfg/DynReconf2.cfg  
)
```

以下是 catkin 构建选项。INCLUDE_DIRS 表示将使用 INCLUDE_DIRS 后面的内部目录 include 的头文件。LIBRARIES 表示将使用随后而来的功能包的库。CATKIN_DEPENDS 后面指定如 roscpp 或 std_msgs 等依赖包。目前的设置是表示依赖于 roscpp 和 std_msgs。DEPENDS 是一个描述系统依赖包的设置。

```
catkin_package(  
  INCLUDE_DIRS include  
  LIBRARIES my_first_ros_pkg  
  CATKIN_DEPENDS roscpp std_msgs  
  DEPENDS system_lib
```

)

`include_directories` 是可以指定包含目录的选项。目前设定为 `$catkin_INCLUDE_DIRS`，这意味着将引用每个功能包中的 `include` 目录中的头文件。当用户想指定一个额外的 `include` 目录时，写在 `$catkin_INCLUDE_DIRS` 的下一行即可。

```
include_directories(  
    ${catkin_INCLUDE_DIRS}  
)
```

`add_library` 声明构建之后需要创建的库。以下是引用位于 `my_first_ros_pkg` 功能包的 `src` 目录中的 `my_first_ros_pkg.cpp` 文件来创建 `my_first_ros_pkg` 库的命令。

```
add_library(my_first_ros_pkg
```

```
src/${PROJECT_NAME}/my_first_ros_pkg.cpp  
)
```

`add_dependencies` 是在构建该库和可执行文件之前，如果有需要预先生成的有依赖性的消息或 `dynamic_reconfigure`，则要先执行。以下内容是优先生成 `my_first_ros_pkg` 库依赖的消息及 `dynamic_reconfigure` 的设置。

```
add_dependencies(my_first_ros_pkg ${${  
    PROJECT_NAME}_EXPORTED_TARGETS} ${  
    catkin_EXPORTED_TARGETS})
```

`add_executable` 是对于构建之后要创建的可执行文件的选项。以下内容是引用 `src/my_first_ros_pkg_node.cpp` 文件生成 `my_first_ros_pkg_node` 可执行文件。如果有多个要引用的 `*.cpp` 文件，将其写入 `my_first_ros_pkg_node.cpp` 之

后。如果要创建两个以上的可执行文件，需追加 `add_executable` 项目。

```
add_executable(my_first_ros_pkg_node src/  
my_first_ros_pkg_node.cpp)
```

如前面描述的 `add_dependencies` 一样，`add_dependencies` 是一个首选项，是在构建库和可执行文件之前创建依赖消息和 `dynamic reconfigure` 的设置。下面介绍名为 `my_first_ros_pkg` 的可执行文件的依赖关系，而不是上面提到的库。在建立可执行文件之前，先创建消息文件的情况下会经常用到。

```
add_dependencies(my_first_ros_pkg_node ${  
  ${PROJECT_NAME}_EXPORTED_TARGETS} ${  
  catkin_EXPORTED_TARGETS})
```

`target_link_libraries` 是在创建特定的可

执行文件之前将库和可执行文件进行链接的选项。

```
target_link_libraries(  
    my_first_ros_pkg_node ${  
        catkin_LIBRARIES} )
```

以下为一个 Demo：

```
cmake_minimum_required(VERSION 2.8.3)  
project(my_first_ros_pkg)  
find_package(catkin REQUIRED COMPONENTS  
    roscpp std_msgs)  
catkin_package(CATKIN_DEPENDS roscpp  
    std_msgs)  
include_directories(${catkin_INCLUDE_DIRS}  
    })  
add_executable(hello_world_node src/  
    hello_world_node.cpp)  
target_link_libraries(hello_world_node ${  
    catkin_LIBRARIES})
```


4.9.4 编写源代码

在上述 CMakeLists.txt 文件的可执行文件创建部分（add_executable）中，进行了以下设置。

```
add_executable(hello_world_node src/  
               hello_world_node.cpp)
```

换句话说，是引用功能包的 src 目录中的 hello_world_node.cpp 源代码来生成 hello_world 可执行文件。

4.9.5 构建功能包

所有构建功能包的准备工作都已完成。在构建之前，使用以下命令更新 ROS 功能包的配置文件。这是一个将之前创建的功能包反映在 ROS 功能包列表的命令，这并不是必选操作，但在创建新功能包后更新的话使用时会比较方便。

```
$ rospack profile
```

下面是 catkin 构建。移动到 catkin 工作目录后进行 catkin 构建。

```
$ cd ~/catkin_ws && catkin_make
```

注意：如果在 .bashrc 文件中设置了 “alias cm = cd /catkin_ws && catkin_make”，则可以用终端窗口中的 cm 命令替换以前的命令。

4.9.6 运行节点

在运行节点之前先运行 `roscore`。请注意，运行 `roscore` 后，ROS 中的所有节点都可用，除非退出了 `roscore`，否则只需运行一次。

使用以下命令运行节点。这是在名为 `my_first_ros_pkg` 的功能包中运行名为 `hello_world_node` 的节点的命令。

```
$ rosruncat my_first_ros_pkg  
hello_world_node
```

5 ROS 命令

5.1 ROS 命令概述

ROS 可以通过在 shell 环境中输入命令来进行文件系统的使用、源代码编辑、构建、调试和功能包管理等。为了正确使用 ROS，除了基本的 Linux 命令之外，还需要熟悉 ROS 专用命令。

5.2 ROS shell 命令

ROS shell 命令又被称为 rosbash。这使我们可以 ROS 开发环境中使用 Linux 中常用的 bash shell 命令。我们主要使用前缀是 ros 且带有多种后缀的命令，例如 cd、pd、d、ls、ed、cp 和 run。相关命令如下。

roscd ★★★ ros+cd(changes directory) 移动

到指定的 ROS 功能包目录

rosls ★☆☆ ros+ls(lists files) 显示 ROS 功能包的文件和目录

roscd ★☆☆ ros+cd(editor) 编辑 ROS 功能包的文件

roscp ★☆☆ ros+cp(copies files) 复制 ROS 功能包的文件

rospd ☆☆☆ ros+pushd 添加目录至 ROS 目录索引

rosd ☆☆☆ ros+directory 显示 ROS 目录索引

5.3 ROS 执行命令

ROS 执行命令管理 ROS 节点的运行。最重要的是，roscore 被用作节点之间的名称服务器。执行命令是 rosrund 和 roslaunch。rosrund 运行一个节点，当运行多个节点或设置各种选项时使用 roslaunch。rosclean 是删除节点执行时记录的日志的命令。

roscore ★★★ roscore master (ROS 名称服务) + roslout (日志记录) + parameter server (参数管理)

rosrund ★★★ roscrun 运行节点

roslaunch ★★★ roslaunch 运行多个节点或设置运行选项

rosclean ★★★ rosclean 检查或删除 ROS

日志文件

5.4 ROS 信息命令

ROS 信息命令用于识别话题、服务、节点和参数等信息。尤其是 rostopic、rosservice、roscall 和 rosparam 经常被使用，并且 rosbag 是 ROS 的主要特征之一，它具有记录数据和回放功能，务必要掌握。

rostopic ★★★★★ rostopic 确认 ROS 话题信息

rosservice ★★★★★ rosservice 确认 ROS 服务信息

roscall ★★★★★ roscall 确认 ROS 节点信息

rosparam ★★★ rostopic param(parameter) 确认和修改 ROS 参数信息

rosviz ★★★ rostopic bag 记录和回放 ROS 消息

rostopic ★★★ rostopic type 显示 ROS 消息类型

rostopic ★★☆☆ rostopic type 显示 ROS 服务类型

rosversion ★☆☆ rostopic version 显示 ROS 功能包的版本信息

rosvtf ☆☆☆ rostopic tf 检查 ROS 系统

5.4.1 rostopic: ROS 节点

rostopic list 查看活动的节点列表

`rostopic ping [节点名称]` 与指定的节点进行连接测试

`rostopic info [节点名称]` 查看指定节点的信息

`rostopic machine [PC 名称或 IP]` 查看该 PC 中运行的节点列表

`rostopic kill [节点名称]` 停止指定节点的运行

`rostopic cleanup` 删除失连节点的注册信息

5.4.2 `rostopic`: ROS 话题

`rostopic list` 显示活动的话题目录

`rostopic echo [话题名称]` 实时显示指定话

题的消息内容

`rostopic find [类型名称]` 显示使用指定类型的消息的话题

`rostopic type [话题名称]` 显示指定话题的消息类型

`rostopic bw [话题名称]` 显示指定话题的消息带宽 (bandwidth)

`rostopic hz [话题名称]` 显示指定话题的消息数据发布周期

`rostopic info [话题名称]` 显示指定话题的信息

`rostopic pub [话题名称] [消息类型] [参数]` 用指定的话题名称发布消息

5.4.3 rosservice: ROS 服务

`rosservice list` 显示活动的服务信息

`rosservice info [服务名称]` 显示指定服务的
信息

`rosservice type [服务名称]` 显示服务类型

`rosservice find [服务类型]` 查找指定服务
类型的服务

`rosservice uri [服务名称]` 显示 ROSRPC URI
服务

`rosservice args [服务名称]` 显示服务参数

`rosservice call [服务名称] [参数]` 用输入
的参数请求服务

5.4.4 rosparam: ROS 参数

rosparam list 查看参数列表

rosparam get [参数名称] 获取参数值

rosparam set [参数名称] 设置参数值

rosparam dump [文件名称] 将参数保存到指定文件

rosparam load [文件名称] 获取保存在指定文件中的参数

rosparam delete [参数名称] 删除参数

5.4.5 rosmmsg: ROS 消息信息

rosmmsg list 显示所有消息

rosmmsg show [消息名称] 显示指定消息

rosmmsg md5 [消息名称] 显示 md5sum

rosmmsg package [功能包名称] 显示用于指定功能包的所有消息

rosmmsg packages 显示使用消息的所有功能包

5.4.6 rossrv: ROS 服务信息

rossrv list 显示所有服务

rossrv show [服务名称] 显示指定的服务信息

rossrv md5 [服务名称] 显示 md5sum

rossrv package [功能包名称] 显示指定的

功能包中用到的所有服务

`rossrv packages` 显示使用服务的所有功能包

5.4.7 rosbag: ROS 日志信息

`rosbag record [选项] [话题名称]` 将指定话题的消息记录到 bag 文件

`rosbag info [文件名称]` 查看 bag 文件的信息

`rosbag play [文件名称]` 回放指定的 bag 文件

`rosbag compress [文件名称]` 压缩指定的 bag 文件

rosvag decompress [文件名称] 解压指定的 bag 文件

rosvag filter [输入文件] [输出文件] [选项] 生成一个删除了指定内容的新的 bag 文件

rosvag reindex bag [文件名称] 刷新索引

rosvag check bag [文件名称] 检查指定的 bag 文件是否能在当前系统中回放

rosvag fix [输入文件] [输出文件] [选项] 将由于版本不同而无法回放的 bag 文件修改成可以回放的文件

5.5 ROS catkin 命令

ROS 的 catkin 命令用于使用 catkin 构建系统来构建功能包。

catkin - create - pkg ★★★ 自动生成功能包

catkin - make ★★★ 基于 catkin 构建系统的构建

catkin - eclipse ★★☆☆ 把用 catkin 构建系统生成的功能包修改，使得可以在 Eclipse 环境中使用

catkin - prepare - release ★★☆☆ 发布时用到的日志整理和版本标记

catkin - generate - changelog ★★☆☆ 发布时

生成或更新 CHANGELOG.rst 文件

catkin _ init _ workspace ★★☆☆ 初始化 catkin
构建系统的工作目录

catkin _ find ★☆☆ 搜索 catkin

5.6 ROS 功能包命令

ROS 功能包命令用于操作 ROS 功能包，
比如显示功能包信息、安装相关功能包等。

rospack ★★★★★ rospack(age) 显示与 ROS 功
能包相关的信息

roinstall ★★☆☆ roinstall 安装 ROS 附加
功能包

rosdep ★★☆☆ rosddep(endencies) 安装该功

能包的依赖性文件

`rosllocate` ☆☆☆ `rosllocate` 与 ROS 功能包信息有关的命令

`roscrcate-pkg` ☆☆☆ `roscrcate-pkg` 自动生成 ROS 功能包（用于旧的 `roscrcuild` 系统）

`rosmake` ☆☆☆ `rosmake` 构建 ROS 功能包（用于旧的 `roscrcuild` 系统）

6 ROS 工具

6.1 三维可视化工具 (RViz)

RViz 是 ROS 的三维可视化工具。它的主要目的是以三维方式显示 ROS 消息，可

以将数据进行可视化表达。例如，可以无需编程就能表达激光测距仪（LRF）传感器中的传感器到障碍物的距离，RealSense、Kinect 或 Xtion 等三维距离传感器的点云数据（PCD，Point Cloud Data），从相机获取的图像值等。

另外，利用用户指定的多边形（polygon）支持各种表现形式，交互标记（Interactive Markers）² 可以表达接收来自用户节点的命令和数据并互交的过程。在 ROS 中，机器人以 URDF（Unified Robot Description Format，统一机器人描述格式）³ 描述，它可以表示为三维模型，并且每个模型可以根据自由度进行移动或驱动，因此可以用于仿真或控制。

6.1.1 RViz 安装与运行

RViz 的运行命令如下。就像任何其他 ROS 工具一样，roscore 必须运行。作为参考，您也可以使用节点运行命令“roslaunch rviz rviz”运行它。

```
$ rviz
```

6.1.2 RViz 画面布局

- (1). 3D 视图 (3D view)：指屏幕的黑色部分。它是可以用三维方式查看各种数据的主屏幕。3D 视图的背景颜色、固定框架、网格等可以在左侧显示的全局选项 (Global Options) 和网格 (Grid) 项目中进行详细设置。

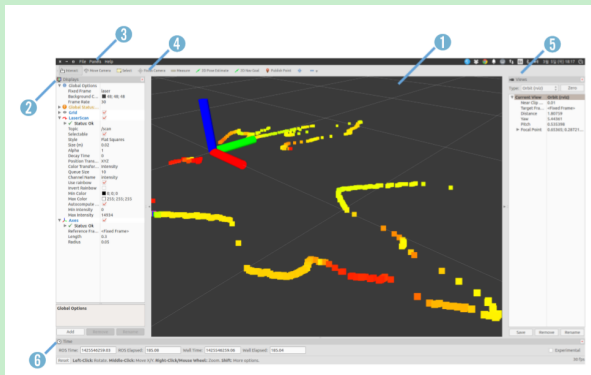


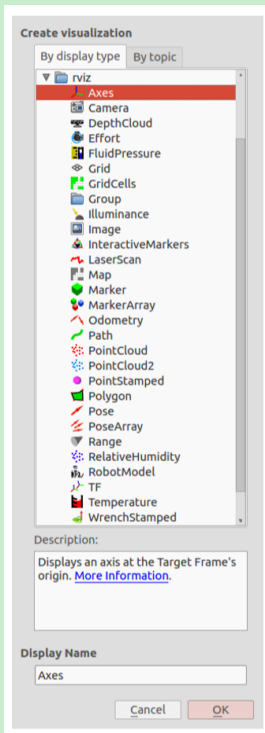
图 3: RViz 画面布局

- (2). 显示屏 (Displays)：左侧的显示屏是从各种话题当中选择用户所需的数据的视图的区域。如果单击屏幕左下方的 [Add]，选择屏幕将如 RViz 显示屏的选择画面所示。目前大约有 30 种不同的显示屏可供选择，我们将在下面的描述中详细介绍。
- (3). 菜单 (Menu)：菜单位于顶部。用户可以选择保存或读取显示屏状态的命令，还可以选择各种面板。
- (4). 工具 (Tools)：工具是位于菜单下方的按钮，允许用户用各种功能按键选择多种功能的工具，例如 Interact、Move Camera、Select、Focus Camera、Measure、2D Pose Estimate、2D Navigation Goal 以及 Publish Point 等。

(5). 视图 (Views)：设定三维视图的视点。

[Orbit：以指定的视点（在这里称为 Focus）为中心旋转。这是默认情况下最常用的基本视图。]、[FPS（第一人称）：显示第一人称视点所看到的画面。]、[ThirdPersonFollower：显示以第三人称的视点尾追特定目标的视图。]、[TopDownOrtho：这是 Z 轴的视图，与其他视图不同，以直射视图显示，而非透视法。]、[XYOrbit：类似于 Orbit 的默认值，但焦点固定在 Z 轴值为 0 的 XY 平面上。]

(6). 时间 (Time)：显示当前时刻 (wall time)、ROS Time 以及他们各自经过的时间。这主要用于仿真，如果需要重新启动，请点击底部的 [Reset] 按钮。



6.1.3 RViz 显示屏

使用 RViz 的过程中最常用的菜单应该是显示屏菜单。该显示屏菜单用于选择三维视图 (3D View) 画面所显示的信息，各项目的说明请参照图 E、F。

6.2 ROSGUI 开发工具 (rqt)

除了三维可视化工具 RViz 之外，ROS 还为机器人开发提供各种 GUI 工具。例如，有一个将每个节点的层次结构显示为图形，且显示当前节点和话题状态的 graph；将消息显示为二维图形的 plot 等。从 ROS Fuerte 版本开始，这些 GUI 开发工具被称为 rqt6，它集成了 30 多种工具，可以作为一个综合的 GUI 工具来使用。另外，RViz 也被集成到

图标	名称	说明
	Axes	显示x、y、z轴。
	Camera	从相机的角度创建一个新的渲染窗口，并将图像覆盖在窗口上。
	DepthCloud	显示基于Depth map的点云。将从相机获取的彩色图像覆盖在“点”上面，这里的“点”是通常基于DepthMap和ColorImage话题的传感器（如Kinect和Xtion）的距离值。
	Effort	显示施加在机器人的每个旋转关节的力。
	FluidPressure	显示流体的压力，如空气或水。
	Grid	显示二维或三维网格。
	Grid Cells	显示网格的每个单元格。主要用于显示导航的costmap中的障碍物。
	Group	这是一个对显示屏进行分组的容器。要使用的显示屏可以作为一个组进行管理。
	Illuminance	指示亮度。
	Image	在新的渲染窗口中显示图像。与camera显示屏不同，它不覆盖相机。
	InteractiveMarkers	显示一个或多个Interactive Marker。可以使用鼠标更改位置（x，y，z）和姿态（roll，pitch，yaw）。
	LaserScan	显示激光扫描值。
	Map	将导航中使用的占用地图（occupancy map）显示在地平面（ground plane）上。
	Marker	显示RViz提供的箭头、圆圈、三角形、矩形和圆柱等标志。
	MarkerArray	显示多个标记显示屏。
	Odometry	将按照时间推移的测位（odometry）信息以箭头的形式显示。例如，将随着机器人的移动所产生的连续路径（机器人的位置和方向），沿着时间间隔以箭头的形式显示。
	Path	显示导航中使用的机器人的路径。








图标	名称	说明
	Point Cloud	显示点云（point cloud）数据。它用于表示Depth camera系列的传感器数据，如RealSense、Kinect和Xtion等。由于PointCloud and PointCloud2是有区别的。由于PointCloud2是遵循最新的PCL（Point Cloud Library）中使用的格式，因此一般情况下使用PointCloud2即可。
	PointCloud2	
	PointStamped	显示圆形点。
	Polygon	显示一个线形式的多边形轮廓。主要用于在二维平面上简单显示机器人的轮廓。
	Pose	显示三维姿态（pose，位置+方向）。姿态是一个箭头，箭头的起点是位置（x，y，z），箭头的方向是机器人的方向（roll，pitch，yaw）。例如，机器人模型的位置和方向可以用姿态表示，而导航中可以用目标（goal）点表示。
	Pose Array	显示多个姿态。
	Range	它是圆锥喇叭形状，用来可视化超声波或红外传感器的测量范围。
	RelativeHumidity	显示相对湿度。
	RobotModel	显示机器人模型。
	TF	显示ROS中使用的坐标转换TF。显示方法与上述轴（Axes）一样用xyz轴表示，各轴根据相对坐标用箭头表现分层结构。
	Temperature	显示温度。
	WrenchStamped	用“箭头（力）”和“箭头+圆形（转矩）”表示扭转（wrench）的动作。

图 6: F

rqt 的插件中，这使 rqt 成为 ROS 的一个不可缺少的 GUI 工具。

另外，顾名思义，rqt 是基于 Qt 开发的，而 Qt 是一个广泛用于计算机编程的 GUI 编程的跨平台框架，用户可以方便自由地添加和开发插件。本节介绍 rqt 插件中的 rqt_image_view、rqt_graph、rqt_plot 和 rqt_bag。

6.2.1 rqt 安装与运行

运行 rqt 的命令如下。只需键入 rqt。作为参考，用户可以使用节点执行命令“roslaunch rqt rqt_gui”执行它。

```
$ rqt
```

运行 rqt 将显示 rqt 的 GUI 界面，如下图所示。如果是第一次，它将只显示菜单，此外没有任何内容。这是因为还没有指定 rqt 直接运行的插件程序。

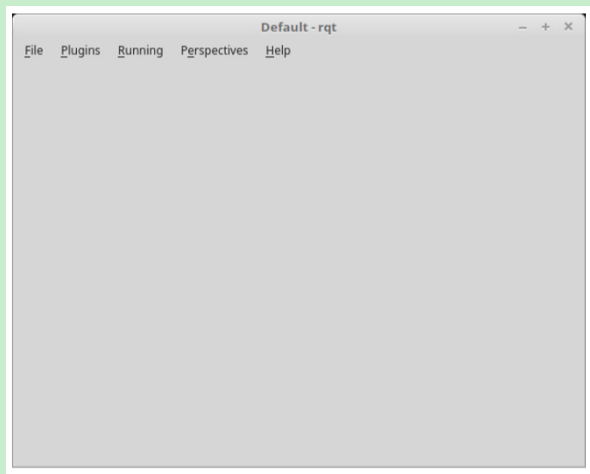


图 7: G

rqt 的各菜单如下。

文件 (File) 只有一个退出 rqt 的子菜单。

插件 (Plugins) 有 30 多个插件。可以选择并使用它。

动作 (Running) 显示当前运行的插件，在不需要的时候可以停止。

全景 (Perspectives) 用于保存当前运行的插件组，并在下次运行相同的插件组。

6.2.2 rqt 插件

如果从 rqt 的顶部菜单中选择 [插件 (Plugins)]，则可以看到大约 30 个插件。该插件具有以下功能。大部分是非常有用的

rqt 的默认插件。非官方的插件也可以添加到此，需要的话用户也可以添加自己开发的 rqt 插件。

动作 (Action)

Action Type Browser: 查看动作类型的数据结构的插件。

配置 (Configuration)

Dynamic Reconfigure: 这是用于更改节点参数值的插件。

Launch: roslaunch 的 GUI 插件，当不记得 roslaunch 的名称或配置时，它非常有用。

自检 (Introspection)

Node Graph: 一种图形视图类型的插件，

可以检查当前运行中的节点间的关系图与消息的流动。

Package Graph: 这是一个图形视图插件，显示功能包的依赖关系。

Process Monitor: 可以检查当前正在运行的节点的 PID（进程 ID）、CPU 利用率、内存使用情况和线程数。

日志（Logging）

Bag: 是一个与 ROS 数据记录相关的插件。

Console: 它是一个允许用户在一个屏幕中查看来自节点的警告（Warning）和错误（Error）等消息的插件。

Logger Level: 通过选择负责发布日志的

记录器节点来设置 Debug、Info、Warn、Error 和 Fatal 等日志信息（称为记录器级别）的工具。调试时选择 Debug 会非常方便。

多种工具（Miscellaneous Tools）

Python Console: Python 控制台屏幕插件。

Shell: 它是一个运行 shell 的插件。

Web: 运行 Web 浏览器的插件。机器人工具（Robot Tools）

Controller Manager: 这是一个允许用户检查机器人控制器的状态、类型和硬件接口信息插件。

Diagnostic Viewer: 这是一个检查机器人设备和错误的插件。

MoveIt! Monitor: 用于查看运动规划的 MoveIt! 数据的插件。

Robot Steering: 手动控制机器人的 GUI 工具。在远程控制时，利用此 GUI 工具进行机器人遥控会非常有用。

Runtime Monitor: 它是一个可以实时查看节点中发生的警告或错误的插件。

服务 (Services)

Service Caller: 它是一个 GUI 插件，可以连接到正在运行中的服务服务器，并请求服务。这对测试服务 (Service) 很有用。

Service Type Browser: 这是一个用于检查服务类型的数据结构的插件。

话题 (Topics)

Easy Message Publisher: 这是一个允许用户在 GUI 环境中发布话题的插件。

Topic Publisher: 这是一个可以发布话题的 GUI 插件，这对话题测试很有用。

Topic Type Browser: 这是检查话题类型的数据结构的插件。这对于检查话题类型很有用。

Topic Monitor: 这是一个列出当前正在使用的话题，并确认用户选择的话题信息的插件。

可视化 (Visualization)

Image View: 这是一个可以检查相机的图像数据的插件。这对于简单的照相机数据测试非常有用。

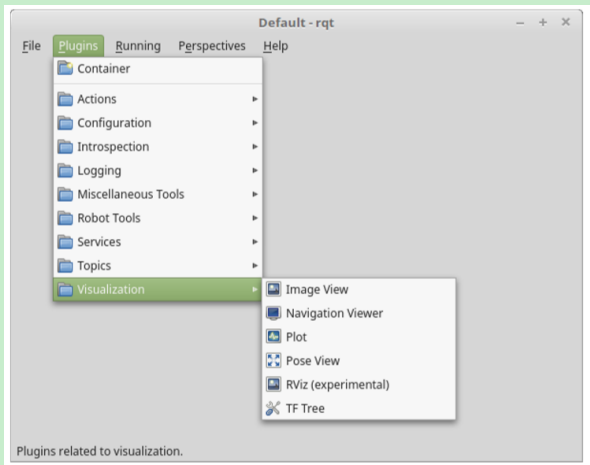
Navigation Viewer: 这是一个用于在导航中检查机器人的位置和目标点的插件。

Plot: 这是一个绘图二维数据的 GUI 插件。这对于二维数据绘图非常有用。

Pose View: 它是一个显示机器人模型和 TF 的姿态 (pose, 位置和方向) 的插件。

RViz: 这是 RViz 插件, 是一个 3D 可视化工具插件。

TF Tree: 这是一个图形视图插件, 它用树形图显示了通过 TF 收集的每个坐标之间的关系。



8: H

6.2.3 Rqt - image - view

这是一个显示相机的图像数据的插件。这不是一个图像处理过程，它只是在简单地查看图像时非常有用。一般的 USB 摄像头支持 UVC，所以用户可以使用 ROS 的 `uvc_camera` 功能包。首先，使用以下命令安装 `uvc_camera` 功能包。

```
$ sudo apt-get install ros-kinetic-uvc-camera
```

将 USB 摄像头连接到计算机的 USB 接口，然后使用以下命令运行 `uvc_camera` 功能包中的 `uvc_camera_node` 节点。

```
$ rosrun uvc_camera uvc_camera_node
```

然后用“`rqt`”命令运行 `rqt`，之后从菜单

中选择 [Plugins] → [Image View]。如果在左上方的消息选择下拉列表中选择 “/image - raw”，则可以看到如图 6-10 所示的图像。

```
$ rqt
```

除了从 rqt 菜单中选择插件之外，还可以使用专用的运行命令，如下所示。

```
$ rqt_image_view
```

6.2.4 rqt - graph

rqt - graph 是用图形表示当前活动中的节点与在 ROS 网络上传输的消息之间的相关性的工具。这对了解当前 ROS 网络情况非常有用。用法很简单。如以下示例所示，为了查看第 3.3 节中描述的 turtlesim 功能包

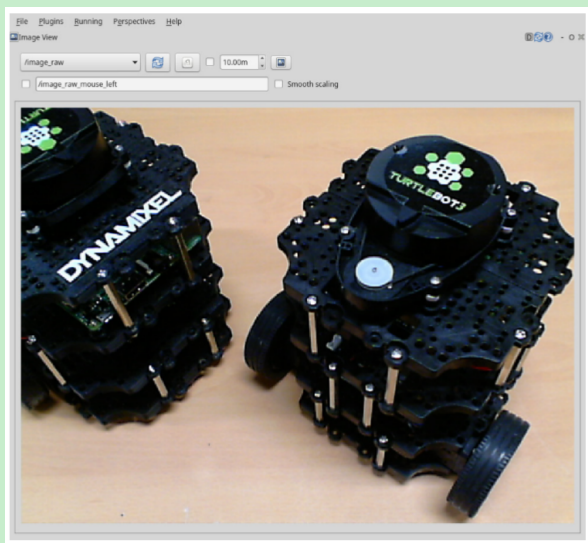


图 9: 1

中的 `turtlesim_node` 和 `turtle_teleop_key`，以及第 6.2.3 节中描述的 `uvc_camera` 功能包中的 `uvc_camera_node` 节点，将他们分别在不同的终端中运行。

```
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
$ rosrun uvc_camera uvc_camera_node
$ rosrun image_view image_view image:=
    image_raw
```

然后如下例所示，使用命令“`rqt`”运行 `rqt` 并从菜单中选择 [Plugins]→[Node Graph] 即可。请注意，用户也可以在终端中运行“`rqt_graph`”，而无需直接从菜单中选择插件。运行 `rqt_graph` 时，节点和话题的相关性会如图 6-11 所示显示。

```
$ rqt
```

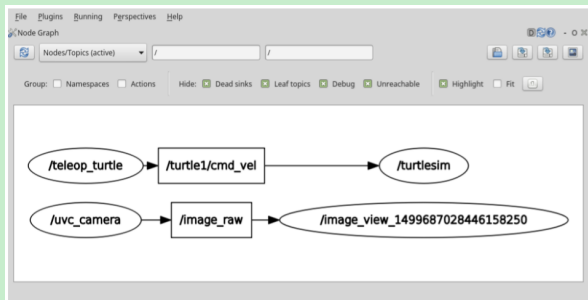


图 10: J

在图 6-11 中，椭圆表示节点（/teleop - turtle、/turtlesim），方块（/turtle1/cmd - vel）表示话题消息。箭头表示发送和接收消息。在前面的例子中，运行 turtle - teleop - key 时，会运行 teleop - turtle 节点；运行 turtlesim - node 节点时，会运行 turtlesim 节点。可以看到，这两个节点正在以平移速度和旋转速度的消息类型（话题名称：/turtle1/cmd - vel）发

送和接收键盘的方向键值。uvc_camera 功能包也可以通过 rqt_graph 确认 uvc_camera 节点在发出/image_raw 话题消息，并且 image_view_xxx 节点在订阅它。这已经通过简单的几个节点确认过，但在实际的 ROS 编程中，会有数十个节点发送和接收各种话题消息。此时，rqt_graph 对于检查当前 ROS 网络上节点的相关性会非常的有用。

6.2.5 rqt_plot

这一次，我们使用下面的命令运行 rqt_plot，而不是在 rqt 界面中选择插件。作为参考，用户可以使用节点执行命令 rosrun rqt_plot rqt_plot 运行它。

```
$ rqt_plot
```

`rqt - plot` 运行后，点击程序右上角的齿轮形状的选项图标。可以如图 6-12 所示选择一个选项，其默认设置为“MatPlot”。除 MatPlot 外，还提供了 PyQtGraph 和 QwtPlot。请参阅相关的安装说明并使用所需的图形库。例如，如果要使用 PyQtGraph 作为默认绘图而不是 MatPlot，请从以下下载地址下载并安装最新的 `python-pyqtgraph - 0.9.xx-x - all.deb` 文件。安装 PyQtGraph 之后，用户可以使用 PyQtGraph。

`rqt - plot` 是一个二维数据绘图工具。绘图意味着绘制坐标。换句话说，它接收到 ROS 消息并将其撒在坐标系上。例如，假设要标记 turtlesim 节点 pose 消息的 x 和 y 坐标。首先，运行 turtlesim 功能包中的 `turtlesim - node`。

Plot Type

☐ PyQtGraph

Based on PyQtGraph

- installer: <http://luke.campagnola.me/code/pyqtgraph>

☒ MatPlot

Based on Matplotlib

- needs most CPU

- needs matplotlib >= 1.1.0

- if using PySide: PySide > 1.1.0

☐ QwtPlot

Based on QwtPlot

- does not use timestamps

- uses least CPU

- needs Python Qwt bindings

Plot Markers

☐ Show Plot Markers

Warning: Displaying markers in rqt_plot may cause
high cpu load, especially using PyQtGraph

Cancel

OK

 //: K

```
$ rosrun turtlesim turtlesim_node
```

然后在 `rqt - plot` 上方的 Topic 栏中输入 `/turtle1/pose/`，则会在二维（x 轴：数据值，y 轴：时间）坐标系中绘制 `/turtle1/pose/` 节点。或者，您可以使用下一个命令立即运行它，包括指定要图示的话题。

```
$ rqt_plot /turtle1/pose/
```

接下来，运行 `turtlesim` 功能包中的 `turtle - teleop - key` 来移动屏幕上的乌龟。

```
$ rosrun turtlesim turtle_teleop_key
```

如图 6-13 所示，可以看到在显示龟的 x 位置、y 位置、theta 方向和平移转速。这是显示二维数据坐标的有用的工具。这里

表示了 turtlesim 功能包，但 rqt - plot 也可以用于表达用户开发的节点的二维数据。特别地，适合于随着时间的推移显示传感器值，例如速度和加速度。

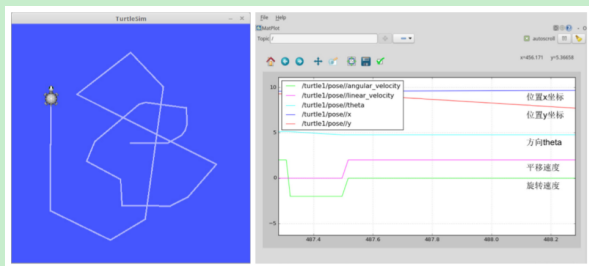


图 12: L

6.2.6 rqt - bag

rqt - bag 是一个可以将消息进行可视化的 GUI 工具。在 5.4.8 节中，ROS 日志信息

中的 `rosvbag` 是基于文本的，但是 `rqt - bag` 对于图像数据类的消息管理是非常有用的，因为 `rqt - bag` 多了可视化功能，因此可以立即查看摄像机的图像值。在测试之前，运行 `rqt - image - view` 和 `rqt - graph` 的工具说明中提到的 `turtlesim` 和 `uvc - camera` 中的所有相关节点。接下来，用如下命令生成为一个 `bag` 文件，记录相机的 `/image - raw` 和 `turtlesim` 的 `/turntlesim/turtle1/cmd - vel` 值。

在第 5.4 节中，曾使用 `rosvbag` 程序将 ROS 上的各种话题消息作为 `bag` 文件进行保存、回放和压缩。`rqt - bag` 是 `rosvbag` 的 GUI 版本，和 `rosvbag` 一样，它可以存储、回放和压缩话题消息。另外，由于它是一个 GUI 程序，所有的命令都是用按钮制作的，所以它很容易使用，并且用户可以用类似使用

视频编辑器一样，在时间轴上来回查看摄像机图像。

为了利用 `rqt - bag` 的特点，将 USB 摄像头图像保存为一个 `bag` 文件，然后使用 `rqt - bag` 进行播放。

```
$ rosrun uvc_camera uvc_camera_node  
$ rosbag record /image_raw  
$ rqt
```

使用“`rqt`”命令运行 `rqt`，然后从菜单中选择[插件 (Plugins)]→[日志 (Logging)]→[包 (Bag)]。然后选择左上方的文件夹图标 (Load Bag) 加载刚才录制的 *.bag 文件。然后，如图 6-14 所示，用户可以在时间轴上查看相机图像的变化。还可以进行放大、回放和查看各时间点的数据值。如果右键

单击鼠标按钮，则会出现“Publish”选项来重新发送消息。



图 13: M

7 ROS 编程基础

7.1 ROS 编程前须知事项

7.1.1 标准单位

对 ROS 中所使用的消息（message），推荐使用世界上最广泛运用的标准单位 SI。

为了确保这一点，REP-0103 也明确了各物理量的单位。

例如，长度 (Length) 使用米 (meter)、质量 (Mass) 使用千克 (Kilogram)、时间 (Time) 使用秒 (Second)、电流 (Current) 使用安培 (Ampere)、角度 (Angle) 使用弧度 (Radian)、频率 (Frequency) 使用赫兹 (Hertz)、力 (Force) 使用牛顿 (Newton)、功率 (Power) 使用瓦 (Watt)、电压 (Voltage) 使用伏特 (Volt)、温度 (Temperature) 使用摄氏度 (Celsius)。其他所有单位都是这些单位的组合。

消息鼓励重用 ROS 提供的方式，但也可以根据需要使用用户全新定义的新的类型的消息。然而，消息用到的单位却必须要遵守使用 SI 单位，这是为了让其他用户

使用这种消息的时候不需要转换单位。

REP (ROS Enhancement Proposals): REP 是一份建议书, 它的内容包含由用户们在 ROS 社区提出的规则、新功能和和管理方法。它用于以民主方式创建 ROS 的规则的情况, 还用于在协商 ROS 的开发、运营和管理所需的内容的情况。收到建议书后, 许多 ROS 用户可以查看, 并通过互相协商继续修改。REP 就是通过这样的过程成为 ROS 标准文档的。REP 文件的目录可以在 <http://www.ros.org/reps/rep-0000.html> 找到。

7.1.2 坐标表现方式

如图左侧所示, ROS 中的旋转轴使用 x , y 和 z 轴。正面是 x 轴的正方向, 轴是红

色 (R)。左边是 y 轴的正方向，轴用绿色 (G) 表示。最后，上方是 z 轴的正方向，轴用蓝色 (B) 表示。为了便于记忆，您可以将 x 轴视为食指，将 y 轴视为中指，将 z 轴视为拇指。顺序是 x 、 y 、 z ，且颜色是 RGB 颜色顺序。机器人的旋转方向是右手定则，用右手卷住的方向是正 (+) 方向。这种坐标表示法在 ROS 编程中经常使用，必须以 x : forward, y : left, z : up 的形式进行编程。

7.1.3 编程规则

为了最大化每个程序的源代码的可重用性，ROS 指定了编程风格指南，并建议开发者遵守该指南。这减少了开发人员在处理源代码时频繁发生的额外的选项，也提高了其他协作开发人员和用户们的代码

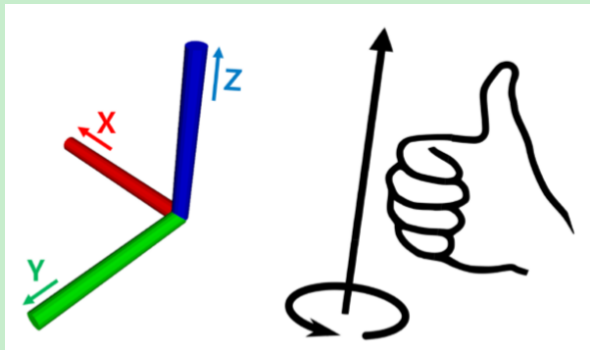


图 14: N

理解程度，并降低了他们之间的代码分析难度。这不是一个要求，但为了代码共享，笔者想鼓励 ROS 的许多用户遵守这一规则。编程规则在 wiki (C++, Python) 中按各个编程语言有详细解释。下面的表格中整理了基本的命名规则。

对象	命名规则	举例
功能包	under_scored	Ex) first_ros_package
话题、服务	under_scored	Ex) raw_image
文件	under_scored	Ex) turtlebot3_fake.cpp
注意，使用ROS消息和服务时，放置在/msg和/srv目录中的消息、服务和动作文件的名称遵循CamelCased规则。这是因为*.msg、*.srv和*.action被转换为头文件后用作结构体和数据类型（例如TransformStamped.msg，SetSpeed.srv）。		
命名空间	under_scored	Ex) ros_awesome_package
变量	under_scored	Ex) string table_name;
数据类型	CamelCased	Ex) typedef int32_t PropertiesNumber;
类	CamelCased	Ex) class UrlTable
结构体	CamelCased	Ex) struct UrlTableProperties
枚举型	CamelCased	Ex) enum ChoiceNumber
函数	camelCased	Ex) addTableEntry();
函数方法	camelCased	Ex) void setNumEntries(int32_t num_entries)
常数	ALL_CAPITALS	Ex) const uint8_t DAYS_IN_A_WEEK = 7;
宏定义	ALL_CAPITALS	Ex) #define PI_ROUNDED 3.0

7.2 发布者节点和订阅者节点的创建和运行

ROS 消息通信中使用的发布者 (Publisher) 和订阅者 (Subscriber) 可以被发送和接收所代替。在 ROS 中，发送端称为发布者，接收端称为订阅者。本节旨在创建一个简单的 msg 文件，并创建和运行发布者和订阅者节点。

7.2.1 创建功能包

以下命令是创建 `ros_tutorials_topic` 功能包的命令。这个功能包依赖于 `message_generation` `std_msgs` 和 `roscpp` 功能包，因此将这些用作依赖选项。第二行命令意味着将使用创建新的功能包时用到的 `message_generation` 表示

将使用创建新消息的功能包 `std_msgs`（ROS 标准消息功能包）和 `roscpp`（在 ROS 中使用 C/C++ 的客户端程序库）必须在创建功能包之前安装。用户可以在创建功能包时指定这些相关的功能包设置，但也可以在创建功能包之后直接在 `package.xml` 中修改。

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg ros_tutorials_topic  
    message_generation std_msgs roscpp
```

创建功能包时，将在 `/catkin_ws/src` 目录中创建 `ros_tutorials_topic` 功能包目录，并在该功能包目录中创建 ROS 功能包的默认目录和 `CMakeLists.txt` 和 `package.xml` 文件。可以使用下面的 `ls` 命令检查它，并使用基于 GUI 的 Nautilus（类似 Windows 资源管理器）来检查功能包的内部。

```
$ cd ros_tutorials_topic
$ ls
include  → 头文件目录
src      → 源代码目录
CMakeLists.txt  → 构建配置文件
package.xml    → 功能包配置文件
```

7.2.2 修改功能包配置文件

ROS 的必备配置文件 `package.xml` 是一个包含功能包信息的 XML 文件，其中包含用于描述功能包名称、作者、许可证和依赖包的信息。使用以下命令，利用编辑器（`gedit`、`vim`、`emacs` 等）打开文件，并修改它以匹配当前节点。以下是本次 Demo：

```
<?xml version="1.0"?>
```

```
<package>
<name>ros_tutorials_topic</name>
<version>0.1.0</version>
<description>ROS tutorial package to
    learn the topic</description>
<license>Apache License 2.0</license>
<author email="pyo@robotis.com">Yoonseok
    Pyo</author>
<maintainer email="pyo@robotis.com">
    Yoonseok Pyo</maintainer>
<url type="bugtracker">https://github.com
    /ROBOTIS-GIT/ros_tutorials/issues</
    url>
<url type="repository">https://github.com
    /ROBOTIS-GIT/ros_tutorials.git</url>
<url type="website">http://www.robotis.
    com</url>
<buildtool_depend>catkin</
    buildtool_depend>
<build_depend>roscpp</build_depend>
```

```
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</
    build_depend>
<run_depend>roscpp</run_depend>
<run_depend>std_msgs</run_depend>
<run_depend>message_runtime</run_depend>
<export></export>
</package>
```

7.2.3 修改构建配置文件 (CmakeLists.txt)

ROS 的构建系统 catkin 基本上使用 CMake，它在功能包目录中的 CMakeLists.txt 文件中描述了构建环境。该文件设置可执行文件的创建、依赖包优先构建、链接创建等。以下是本此的 Demo：

```
cmake_minimum_required(VERSION 2.8.3)
```

```
project(ros_tutorials_topic)
```

```
## catkin构建时需要的组件包.
```

```
## 是依赖包，是message_generation、  
std_msgs和roscpp.
```

```
## 如果这些功能包不存在，在构建过程中会发生错误.
```

```
find_package(catkin REQUIRED COMPONENTS  
message_generation std_msgs roscpp)
```

```
## 消息声明: MsgTutorial.msg
```

```
add_message_files(FILES MsgTutorial.msg)
```

```
## 这是设置依赖性消息的选项.
```

```
## 如果未安装std_msgs，则在构建过程中会发生错误.
```

```
generate_messages(DEPENDENCIES std_msgs)
```

```
## catkin功能包选项，描述了库、catkin构建  
依赖项和系统依赖的功能包.
```

```
catkin_package(  
    LIBRARIES ros_tutorials_topic  
    CATKIN_DEPENDS std_msgs roscpp  
)
```

设置包含目录。

```
include_directories(${catkin_INCLUDE_DIRS}  
    })
```

topic_publisher 节点的构建选项。

配置可执行文件、目标链接库和其他依赖项。

```
add_executable(topic_publisher src/  
    topic_publisher.cpp)  
add_dependencies(topic_publisher ${${  
    PROJECT_NAME}_EXPORTED_TARGETS} ${  
    catkin_EXPORTED_TARGETS})  
target_link_libraries(topic_publisher ${  
    catkin_LIBRARIES})
```

```
## topic_subscriber 节点的构建选项。  
add_executable(topic_subscriber src/  
    topic_subscriber.cpp)  
add_dependencies(topic_subscriber ${${  
    PROJECT_NAME}_EXPORTED_TARGETS} ${  
    catkin_EXPORTED_TARGETS})  
target_link_libraries(topic_subscriber ${  
    catkin_LIBRARIES})
```

7.2.4 创建消息文件

在上述的 CMakeLists.txt 文中添加了如下选项。

```
add_message_files(FILES MsgTutorial.msg)
```

这意味着在构建时要包含消息 MsgTutorial.msg，该消息将在此节点中被使用。现在

我们还没有创建 `MsgTutorial.msg`，因此按以下顺序创建它：

```
$ roscd ros_tutorials_topic → 移动到功能包目录
$ mkdir msg → 功能包中创建新的msg消息目录
$ cd msg → 转到创建的msg目录
$ gedit MsgTutorial.msg → 创建新的MsgTutorial.msg文件并修改内容
```

内容很简单。如下所示，创建一个 `time` 消息类型的 `stamp` 消息和一个 `int32` 消息类型的 `data` 消息，除了这些消息类型之外，还有一些如 `bool`、`int8`、`int16`、`float32`、`string`、`time`、`duration` 和 `common_msgs` 等基本消息类型，以及在 ROS 中收集常用消息的 `common_msgs`。这里我们只是为了创建一个简单的例子，

因此用了 `time` 和 `int32`。以下是文件内容：

```
time stamp
```

```
int32 data
```

消息（`msg`、`srv`、`action`）功能包的独立化：一般情况下，建议将消息文件 `msg` 和服务文件 `srv` 制作成一个只包含消息文件的单独的包，而不是将其包含在可执行节点中。其原因是，假设订阅者节点和发布者节点在不同的计算机上运行，存在的问题是必须安装不必要的节点，因为这两个节点只有在它们具有相互依赖性时才能使用。如果您独立创建消息功能包，则只需将独立于消息的功能包添加到依赖性选项，从而消除功能包之间不必要的依赖关系。但是，在本书中，我们为了简化代码已经将消息文件包含在可执行节点中。

7.2.5 创建发布者节点

在前面的 CMakeLists.txt 文件中，给了生成以下可执行文件的选项。

```
add_executable(topic_publisher src/  
    topic_publisher.cpp)
```

换句话说，是在 src 目录中构建 topic_publisher 文件以创建 topic_publisher 可执行文件。那么我们按如下顺序创建一个执行发布者节点函数的源代码。

```
$ roscd ros_tutorials_topic/src → 移至  
    src 目录，该目录是功能包的源代码目录  
$ gedit topic_publisher.cpp → 新建源文  
    件并修改内容
```

以下是此 cpp 文件内容：

```
#include "ros/ros.h"      // ROS默认头文件
#include "ros_tutorials_topic/MsgTutorial
    .h" // MsgTutorial消息头文件（构建后
    自动生成）

int main(int argc, char **argv)    // 节
    点主函数
{
    ros::init(argc, argv, "
        topic_publisher"); // 初始化节点
        名称
    ros::NodeHandle nh;      // 声明一个节
        点句柄来与ROS系统进行通信

    // 声明发布者，创建一个使用
        ros_tutorials_topic功能包的
        MsgTutorial消息文件的
    // 发布者ros_tutorial_pub. 话题名称是
        "ros_tutorial_msg",
    // 消息文件发布者队列（queue）的大小
        设置为100
```

```
ros::Publisher ros_tutorial_pub = nh.  
    advertise<ros_tutorials_topic::  
    MsgTutorial>("ros_tutorial_msg",  
    100);
```

```
// 设定循环周期。"10"是指10Hz，是以0  
    .1秒间隔重复
```

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

```
ros_tutorials_topic::MsgTutorial msg;  
    // 以MsgTutorial消息文件格式声明  
    一个叫作msg的消息
```

```
int count = 0;
```

```
// 声
```

```
明要在消息中使用的变量
```

```
while (ros::ok()) {
```

```
    msg.stamp = ros::Time::now();
```

```
    // 把当前时间传给msg的下级消
```

息、stamp

```
msg.data = count;    // 将变量
```

count的值传给下级消息data

```
ROS_INFO("send msg = %d", msg.
```

```
stamp.sec);    // 显示stamp.
```

sec消息

```
ROS_INFO("send msg = %d", msg.
```

```
stamp.nsec);    // 显示stamp.
```

nsec消息

```
ROS_INFO("send msg = %d", msg.
```

```
data);    // 显示data消息
```

```
ros_tutorial_pub.publish(msg);
```

// 发布消息。

```
loop_rate.sleep();    // 按照上面
```

定义的循环周期进行暂歇

```
++count;    // 变量count增加1
```

```
}
```

```
return 0;
```

```
}
```

7.2.6 创建订读者节点

在 CMakeLists.txt 文件中添加以下选项来生成可执行文件。

```
add_executable(topic_subscriber src/  
    topic_subscriber.cpp)
```

也就是说，通过构建 topic_subscriber.cpp 文件来创建 topic_subscriber 可执行文件。我们创建一个按照以下顺序执行订阅节点功能的源代码。

```
$ roscd ros_tutorials_topic/src → 移动到  
    src 目录，该目录是功能包的源代码目录  
$ gedit topic_subscriber.cpp → 创建和修  
    改新的源代码文件
```

以下是此 cpp 文件内容：

```
#include "ros/ros.h"      // ROS的默认头文件
```

```
#include "ros_tutorials_topic/MsgTutorial.h" // MsgTutorial消息头文件（构建后自动生成）
```

```
// 这是一个消息后台函数，
```

```
// 此函数在收到一个下面设置的名为
```

```
    ros_tutorial_msg的话题时候被调用。
```

```
// 输入的消息是从ros_tutorials_topic功能包接收MsgTutorial消息。
```

```
void msgCallback(const
```

```
    ros_tutorials_topic::MsgTutorial::
```

```
    ConstPtr& msg)
```

```
{
```

```
    ROS_INFO("recieve msg = %d", msg->
```

```
        stamp.sec); // 显示stamp.sec消息
```

```
    ROS_INFO("recieve msg = %d", msg->
```

```
        stamp.nsec); // 显示stamp.nsec消
```

息

```
ROS_INFO("recieve msg = %d", msg->  
data); // 显示data消息
```

```
}
```

```
int main(int argc, char **argv) // 节点  
主函数
```

```
{
```

```
ros::init(argc, argv, "  
topic_subscriber"); // 初始化节  
点名称
```

```
ros::NodeHandle nh;
```

```
// 声明用
```

```
于ROS系统和通信的节点句柄
```

```
// 声明订阅者，创建一个订阅者
```

```
ros_tutorial_sub,
```

```
// 它利用ros_tutorials_topic功能包的  
的MsgTutorial消息文件。
```

```
// 话题名称是"ros_tutorial_msg", 订阅
```


者队列 (queue) 的大小设为 100.

```
ros::Subscriber ros_tutorial_sub = nh
    .subscribe("ros_tutorial_msg",
        100, msgCallback);
```

// 用于调用后台函数，等待接收消息。在
接收到消息时执行后台函数。

```
ros::spin();
```

```
return 0;
```

```
}
```

之后执行：

```
$ cd ~/catkin_ws    → 移动到catkin目录
```

```
$ catkin_make    → 执行catkin构建
```

7.2.7 构建 (build) 节点

现在使用以下命令在 `ros - tutorials - topic` 功能包中构建消息文件、发布者节点和订阅者节点。`ros - tutorials - topic` 功能包的源代码位于 “`/catkin - ws/src/ros - tutorials - topic/src`” 中，而 `ros - tutorials - topic` 功能包中的消息文件位于 “`/catkin - ws/src/ros - tutorials - topic/msg`” 中。基于此的构建将分别在 “`/catkin - ws`” 的 “`/build`” 和 “`/devel`” 目录中生成文件。“`/build`” 目录中保存 catkin 构建用到的配置内容，而 “`/devel/lib/ros - tutorials - topic`” 目录中保存可执行文件。另外，“`/devel/include/ros - tutorials - topic`” 目录存储着从消息文件自动生成的消息头文件。如果您想了解生成文件，请根据目录查看文件。

7.2.8 运行发布者

以下是使用 `roslaunch` 命令运行 `ros_tutorials_topic_publisher` 功能包的 `ros_tutorial_msg_publisher` 节点。运行节点之前，请确保要从另一个终端运行 `roscore`。在以下示例中，即使没有解释，`roscore` 也必须在运行节点之前运行。

```
$ roscore
$ roslaunch ros_tutorials_topic
  topic_publisher
```

下面，我们使用 `rostopic` 命令获取 `topic_publisher` 发布的话题吧。首先，我们来看一下 ROS 网络当前正在使用的话题列表。通过将 `list` 选项添加到 `rostopic` 命令来查看是否存在 `ros_tutorial_msg` 话题。

```
$ rostopic list
```

```
/ros_tutorial_msg  
/rosout  
/rosout_agg
```

接下来，让我们看看我们运行的发布者节点发布的信息。换句话说，是检查 `ros_tutorial_msg` 话题消息。您可以看到发布的信息。

```
$ rostopic echo /ros_tutorial_msg
```

7.2.9 运行订阅者

以下是为了运行订阅者使用 ROS 节点命令 `roslaunch` 来运行 `ros_tutorials_topic` 功能包的 `topic_subscriber` 节点的过程。

```
$ roslaunch ros_tutorials_topic  
topic_subscriber
```

订阅者接收到了发布者发布的 `ros_tutorial_msgs` 话题的消息，并在屏幕上显示该值。

7.2.10 检查运行中的节点的通信状态

让我们用 6.2 节中介绍的 `rqt` 命令来查看运行中的节点的通信状态。您可以使用 `rqt_graph` 或 `rqt`，如下所示。执行 `rqt` 时，在菜单中选择 [Plugins] → [Introspection] → [Node Graph] 可以确认当前在 ROS 上运行的节点和消息。

```
$ rqt_graph 或 $ rqt
```

在当前的 ROS 网络上，发布者节点 (`topic_publisher`) 正在传输话题 (`ros_tutorial_msgs`)，

并且可以确认它正在接收订阅者节点(topic - subscriber)

7.3 创建和运行服务服务器与客户端节点

服务由服务服务器(service server)和服务客户端(service client)组成, 其中服务服务器仅在收到请求(request)时才会响应(response), 而服务客户端则会发送请求并接收响应。与话题不同, 服务是一次性消息通信。因此, 当服务的请求和响应完成时, 两个节点的连接会被断开。这种服务通常在让机器人执行特定任务时用到。或者用于需要在特定条件下做出反应的节点。由于它是一次性的通信方式, 因此对网络的负载很小, 所以是一种非常有用的通信

手段，例如被用作一种代替话题的通信手段。本节旨在创建一个简单的服务文件，并创建和运行一个服务服务器（server）节点和一个服务客户端（client）节点。

7.3.1 创建功能包

以下命令是创建 `ros_tutorials_topic` 功能包的命令。这个功能包依赖于 `message_generation`、`std_msgs` 和 `roscpp` 功能包，因此将这些用作依赖选项。第二行命令意味着将使用创建新的功能包时用到的 `message_generation` 表示将使用创建新消息的功能包 `std_msgs`（ROS 标准消息功能包）和 `roscpp`（在 ROS 中使用 C/C++ 的客户端程序库）必须在创建功能包之前安装。用户可以在创建功能包时指定这些相关的功能包设置，但也可以在创

建功能包之后直接在 `package.xml` 中修改。

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg ros_tutorials_topic
  message_generation std_msgs roscpp
```

创建功能包时，将在 `/catkin_ws/src` 目录中创建 `ros_tutorials_topic` 功能包目录，并在该功能包目录中创建 ROS 功能包的默认目录和 `CMakeLists.txt` 和 `package.xml` 文件。可以使用下面的 `ls` 命令检查它，并使用基于 GUI 的 Nautilus（类似 Windows 资源管理器）来检查功能包的内部。

```
$ cd ros_tutorials_topic
$ ls
include  → 头文件目录
src      → 源代码目录
CMakeLists.txt  → 构建配置文件
package.xml  → 功能包配置文件
```


7.3.2 修改功能包配置文件 (package.xml)

内容如下：

```
<?xml version="1.0"?>
<package>
<name>ros_tutorials_service</name>
<version>0.1.0</version>
<description>ROS tutorial package to
    learn the service</description>
<license>Apache License 2.0</license>
<author email="pyo@robotis.com">Yoonseok
    Pyo</author>
<maintainer email="pyo@robotis.com">
    Yoonseok Pyo</maintainer>
<url type="bugtracker">https://github.com
    /ROBOTIS-GIT/ros_tutorials/issues</
    url>
```

```
<url type="repository">https://github.com  
    /ROBOTIS-GIT/ros_tutorials.git</url>  
<url type="website">http://www.robotis.  
    com</url>  
<buildtool_depend>catkin</  
    buildtool_depend>  
<build_depend>roscpp</build_depend>  
<build_depend>std_msgs</build_depend>  
<build_depend>message_generation</  
    build_depend>  
<run_depend>roscpp</run_depend>  
<run_depend>std_msgs</run_depend>  
<run_depend>message_runtime</run_depend>  
<export></export>  
</package>
```

7.3.3 修改构建配置文件 (CMakeLists.txt)

内容如下：

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_service)

## 这是进行catkin构建时所需的组件包。
## 依赖包是message_generation、std_msgs和
    roscpp。如果这些包不存在，在构建过程
    中会发生错误。  find_package(catkin
    REQUIRED COMPONENTS
    message_generation std_msgs roscpp)

## 服务声明： SrvTutorial.srv
add_service_files(FILES SrvTutorial.srv)

## 这是一个设置依赖消息的选项。
## 如果未安装std_msgs，则在构建过程中会发
    生错误。
```

```
generate_messages(DEPENDENCIES std_msgs)
```

```
## 这是catkin功能包选项，它描述了库、  
    catkin构建依赖和依赖系统的功能包。
```

```
catkin_package(  
    LIBRARIES ros_tutorials_service  
    CATKIN_DEPENDS std_msgs roscpp  
)
```

```
## 设置包含目录。
```

```
include_directories(${catkin_INCLUDE_DIRS}  
    })
```

```
## 这是service_server节点的构建选项。
```

```
## 设置可执行文件、目标链接库和附加依赖  
    项。
```

```
add_executable(service_server src/  
    service_server.cpp)  
add_dependencies(service_server ${${  
    PROJECT_NAME}_EXPORTED_TARGETS} ${
```

```
    catkin_EXPORTED_TARGETS})

target_link_libraries(service_server ${
    catkin_LIBRARIES})

## 这是节点的构建选项。
add_executable(service_client src/
    service_client.cpp)
add_dependencies(service_client ${${
    PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})
target_link_libraries(service_client ${
    catkin_LIBRARIES})
```

7.3.4 创建服务文件

CMakeLists.txt 文件中加了下面的选项。

```
add_service_files(FILEs SrvTutorial.srv)
```

这是在构建本次的节点中使用的 `SrvTutorial.srv` 时所包含的内容。现在您还没有创建 `SrvTutorial.srv`，请按以下顺序创建它。

```
$ roscd ros_tutorials_service → 移动到功能包目录
$ mkdir srv → 在功能包中创建一个名为srv的新服务目录
$ cd srv → 转到创建的srv目录
$ gedit SrvTutorial.srv → 新建和修改SrvTutorial.srv文件
```

内容很简单。让我们以 `int64` 格式设计服务请求（request）`a`、`b`，和结果服务响应（response）`result`，如下所示。“—”是分隔符，用于分隔请求和响应。除了请求和响应之间有一个分隔符之外，它与上述话题的消息相同。