《智能机器人设计》

机器人操作系统ROS

Part 1

ROS体系架构

ROS体系架构

ROS是用于机器人的一种开源的“后操作系统”，或者说次级操作系统。它提供了类似于操作系统所提供的功能，包含硬件抽象描述、设备管理与控制、通用功能实现、程序间的消息传递、程序发行包管理等，它也提供了一些工具程序和函数库用于获取、建立、编写和运行机器人程序。

ROS的主要设计目标是便于机器人研发过程中的代码复用。因此ROS采用了一种分布式架构，各个软构件可以各自独立地设计，松散、即时地组合起来。并可以按照功能包和功能包集的方式分组，因而可以更容易地分享和发布。

ROS的运行架构是一种基于Socket网络连接的松耦合架构。在ROS的架构中，一切可执行的程序被抽象为节点（node），它可以是传感器数据采集程序、执行器控制程序、规划算法程序、视觉识别程序，等等。各个节点之间均使用ROS提供的消息传递机制进行通信。ROS支持多种类型的通信，包括基于服务的同步RPC通信、基于Topic的异步数据流通信及参数服务器上的数据存储等。

ROS体系架构

作为一个开放源代码软件系统，ROS的宗旨是构建一个能够整合不同研究成果，实现算法发布、代码重用的通用机器人软件平台。由于这些功能可满足广大开发者的需求，因此，ROS得到了广泛应用。不少智能机器人研发者在ROS的基础上开发了许多如运动规划、定位导航、仿真、感知等高层功能软件包，使得这一软件平台的功能更加丰富，发展更加迅速。

ROS系统的应用

ROS体系架构

ROS具有如下主要特点。

1）分布式架构：

ROS将每个工作进程都看作一个节点，使用节点管理器进行统一管理，并提供了一套消息传递机制。这种架构可以分散由计算机视觉和语音识别等CPU密集型任务带来的实时计算压力，也能够适应多机器人系统遇到的挑战。基于套接字通信的大量采用，ROS具备分布式计算的实现能力。在任何具备网络连接的主机上都可运行节点程序，这些主机可以是小型的ARM嵌入式系统，或者是具备强大运算能力的基站服务器，甚至是基于Android系统的移动手机或平板电脑。这些异构计算平台均可成为智能机器人系统的一部分。

ROS体系架构

ROS具有如下主要特点。

2) 多语言支持：

由于所有节点的通信都是通过网络套接字来实现的，这意味着只要能够提供套接字接口，节点程序可以用任何编程语言来实现。ROS不依赖特定的编程语言，它目前已经支持多种现代编程语言，如C++、Python和Lisp已经在ROS中得到广泛应用，Java的测试性支持也已经实现。ROS采用了一种独立于编程语言的接口定义语言(IDL)，并实现了多种编程语言对IDL的封装，使得各个不同编程语言编写的“节点”之间也能透明地进行消息传递。

ROS体系架构

ROS具有如下主要特点。

3)良好的可伸缩性：

使用ROS进行机器人研发，既可以简单地编写一两个节点单独运行，又可以通过rospack、roslaunch将很多个节点组织成一个更大的工程，指定它们之间的依赖关系及运行时的组织形式。

4）源码开放：

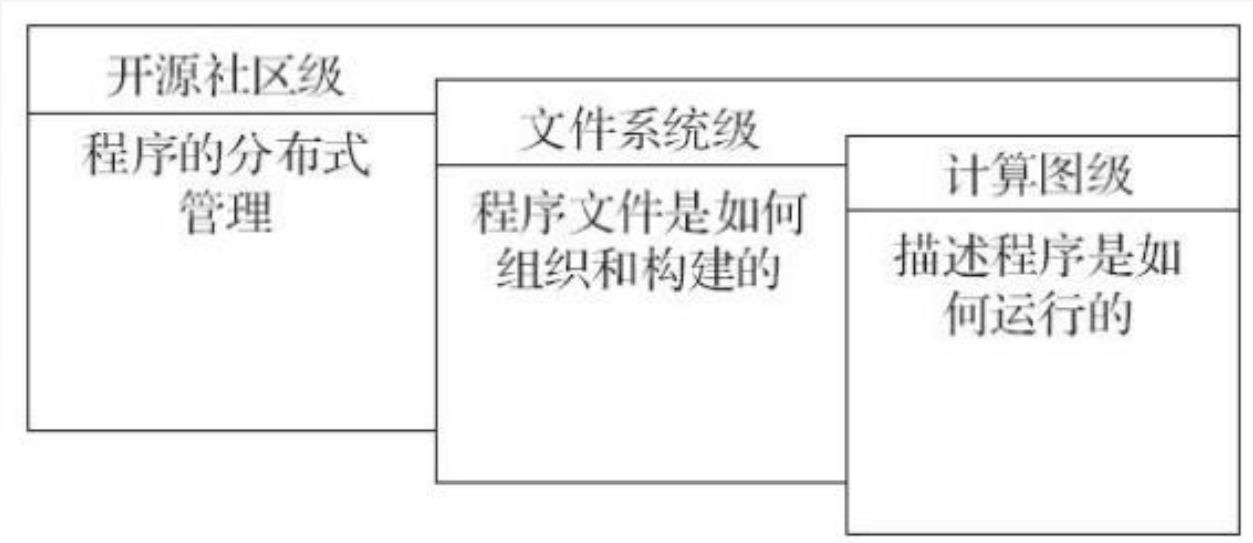
OS遵循BSD协议，实现源码开放，对个人、商业应用及修改完全免费，这也是使其具有更强生命力的主要原因之。

ROS体系架构

ROS整体架构分析

ROS系统架构主要被设计划分为三个级别，如图所示。

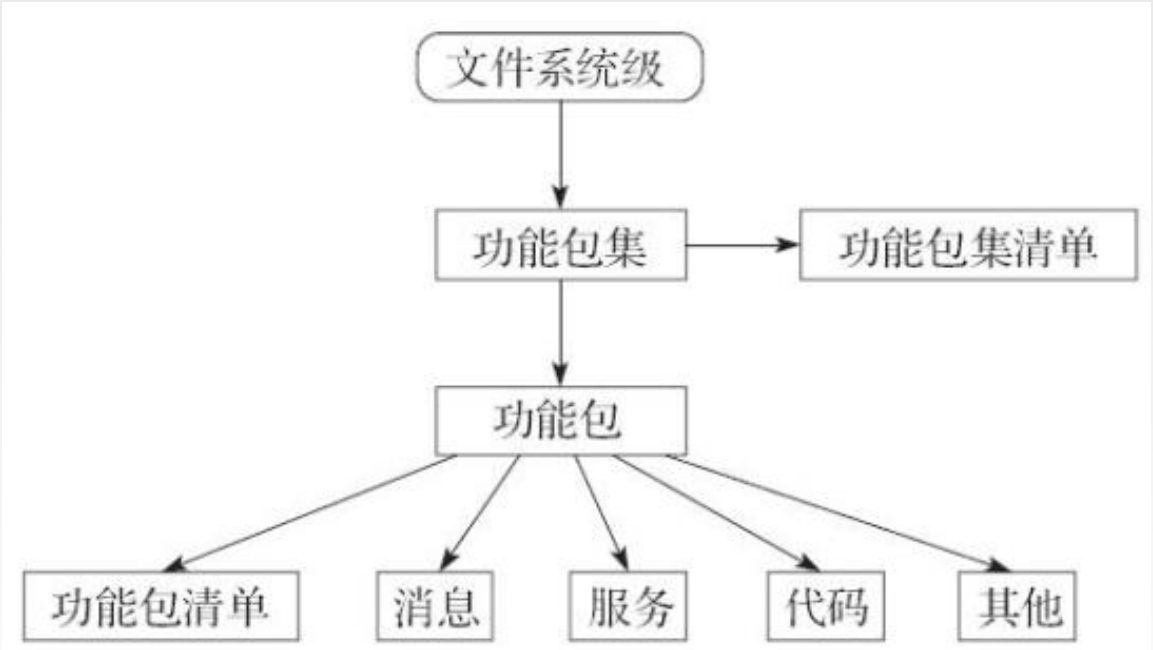
- 文件系统级（Filesystem level），用于描述可以在硬盘上查到的代码及可执行程序。
- 计算图级(Computation Graph level),体现的是进程与进程、进程与系统之间的通信。
- 开源社区级(Community level),主要包括在开发人员之间如何共享知识、算法和代码。



ROS体系架构

ROS整体架构分析——文件系统级

ROS中有众多的抽象节点及消息、服务、工具和库文件，需要采用有效的结构管理这些代码。一个ROS程序的不同构件被放在不同的文件夹下，这些文件夹是根据功能的不同来对文件进行组织的。ROS的软件是以Package（功能包）的方式进行组织的。ROS系统由众多的功能包组成，每个功能包中可能包含多个节点的可执行文件、消息接口定义文件、RPC服务接口定义文件及库文件等。此外还包含一个功能包清单，如图所示。



功能包之间可以配置依赖关系。如果设定了功能包A依赖功能包B，那么ROS在构建系统时，B一定要早于A构建，并且在编译A时，自动配置环境，使得A可以使用B中的头文件和库文件。

ROS体系架构

ROS整体架构分析——文件系统级

(1)功能包集(Stack)

将一些具有某些相关功能的功能包组织在一起，就是一个功能包集。在ROS中存在大量的不同用途的功能包集，如导航功能包集navigation、机械臂运动控制功能包集MoveIt。

功能包集必须包含三个文件：CMakeList.txt、Makefile和stack.xml。

(2)功能包(Package)

功能包是ROS中软件组织的基本形式。创建ROS程序时，功能包具有最小的结构和最少的内容，它可以包含ROS运行的进程（节点）、配置文件等。

ROS体系架构

ROS整体架构分析——文件系统级

(2)功能包(Package)

当提到功能包时，一般指的是一种特定的文件结构和文件夹组合。这种结构具体如下所示。

- bin/:编译和链接程序后，用于存储可执行文件的文件夹。
- include/package_name/：此目录包含了所需库的头文件。请记住导出功能包清单，因为它们还会被其他功能包所使用。
- msg/:程序开发者开发的非标准消息类型。
- scripts/：包括bash、Python或任何其他脚本的可执行脚本文件。
- src/:程序源文件，可以为节点创建一个文件夹，或者按照希望的方式组织它。
- srv/:程序开发者开发的非标准服务。
- CMakeLists.txt:CMake的生成文件。
- manifest.xml/package.xml：功能包清单文件，其必须包含在功能包中，用来说明此功能包相关的各类信息。如果你发现某个文件夹内包含了此文件，那么这个文件夹很可能是一个功能包。

ROS整体架构分析——文件系统级

(3)功能包清单(Manifest)

功能包清单提供关于功能包、许可信息、依赖关系、编译标志等的信息。功能包清单是一个manifests.xml/package.xml文件，通过这个文件能够实现对功能包的管理。打开一个功能包清单文件，可以看到包的名称、依赖关系等信息。功能包清单的作用就是为了更容易地安装和分发这些功能包。

ROS整体架构分析——文件系统级

(4)消息类型(Message/msg type)

消息是ROS数据传递的基本元素，消息类型需要用ROS的消息定义语言进行描述。每个功能包的msg文件夹中，都定义了这个功能包需要的消息类型。此外，如果该功能包依赖另外一个功能包，则该功能包可以使用另一功能包的所有消息类型。

ROS使用了一种简化的消息类型描述语言来描述ROS节点发布的数据值。通过这样的描述语言，ROS能够使用多种编程语言生成不同消息类型的代码。

ROS提供了很多预定义的消息类型。如果你创建了一种新型消息类型，那么就要把消息类型的定义放到功能包的msg文件夹下。该文件夹中包含了用于定义各种消息的文件，这些文件都以.msg为扩展名。

ROS体系架构

ROS整体架构分析——文件系统级

(5)服务类型(Service/srvtype)

ROS使用了一种简化的服务描述语言来描述ROS的服务类型。这直接借鉴了ROS消息的数据格式，以实现节点之间的请求/响应通信。服务的描述存储在功能包的srv子目录下扩展名为.srv的文件中。

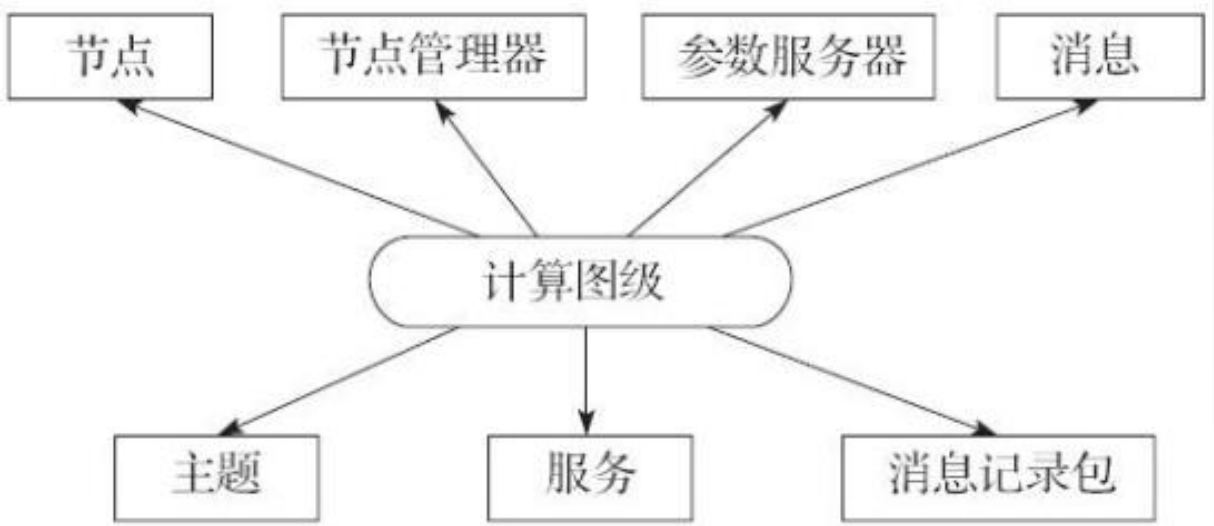
若要调用服务，调用者需要使用该服务功能包的名称及服务名称。例如，对于sample_package1/srv/sample1.srv文件，可以将它称为sample_package1/sample1服务。

ROS中有查看某些功能包名称与服务的相关工具。例如，rossrv工具能输出服务说明.sv文件所在的功能包名称，并可以找到使用某一服务类型的源代码文件。如果要在ROS中创建一个服务，可以使用服务生成器。这些工具能够从基本的服务说明中生成代码，只需要在CMakeLists.txt文件中加一行gensrv（）命令即可。

ROS体系架构

ROS整体架构分析——计算图级

ROS能够创建一个连接所有进程的抽象“网络”，该网络是进程(节点)之间通过主题或服务的连接所形成的。系统中的任何节点都可以访问此网络，并通过该网络与其他节点进行交互，获取其他节点发布的消息，并将自身数据发布到网络上。在这一层级中最基本的概念实体包括节点、节点管理器、参数服务器、消息、服务、主题和消息记录包，如图所示，它们均以不同的方式向计算图级提供数据。



ROS体系架构

ROS整体架构分析——计算图级

(1) 节点(Node)

节点是主要的计算执行进程。如果想要有一个可以与其他节点进行交互的进程，那么需要创建一个节点，并将此节点连接到ROS网络中。通常情况下，系统包含能够实现不同功能的多个节点。最好让每一个节点都具有特定的单一功能，而不是在系统中创建一个包罗万象的“大节点”。

节点都是各自独立的可执行文件，并能够通过主题、服务或参数服务器与其他节点进行通信。ROS通过使用节点概念将代码和功能解耦，提高了系统的容错能力和可维护性，使系统设计得以简化。同时，节点允许ROS能够布置在任意机器上并同时运行。

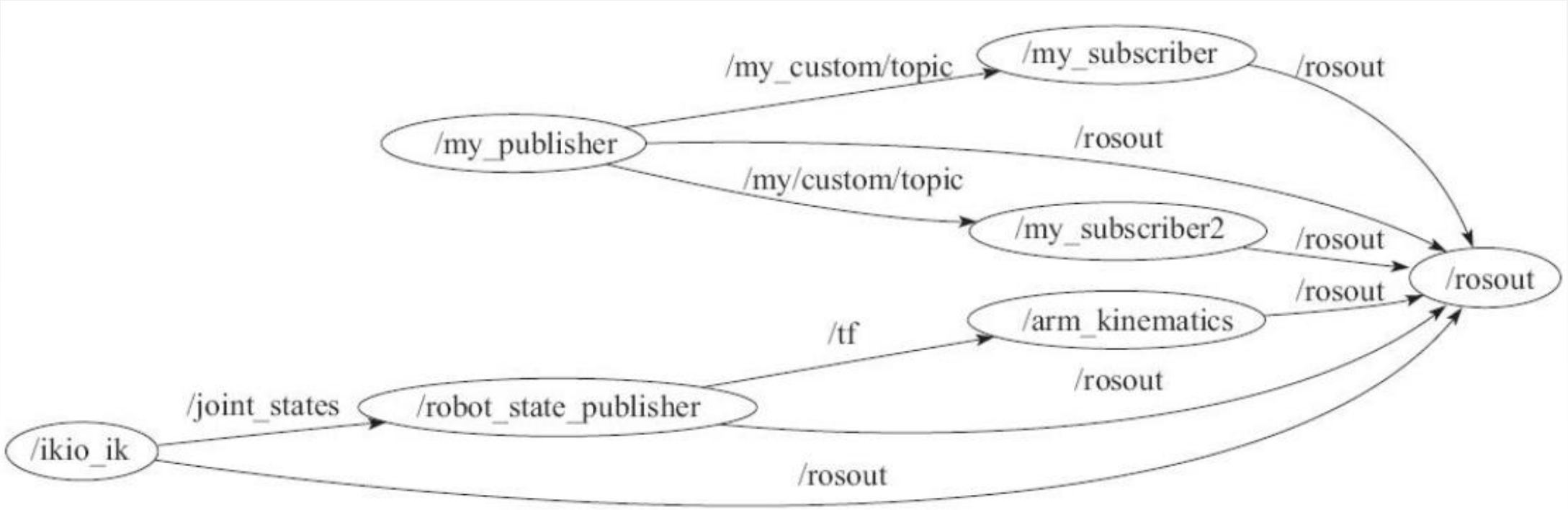
节点在系统中必须有唯一的名称，以便节点使用特定名称与其他节点进行通信而不产生歧义。节点可以使用不同的库进行编写，如roscpp和rospy，roscpp基于C++，rospy基于Python。

ROS体系架构

ROS整体架构分析——计算图级

(1) 节点(Node)

ROS是由很多节点组成的。在这里节点实际上是“软构件”。ROS使用节点使得基于ROS的系统在运行时更加形象化：当许多节点同时运行时，可以方便地使用rqt_graph工具将节点与节点之间的通信绘制成一个图表，其中进程就是图中的节点，节点间的消息连接关系就是其中的弧线连接，如图所示。



ROS体系架构

ROS整体架构分析——计算图级

(2)节点管理器(Master)

节点管理器用于主题、服务名称的注册和查找等。如果在整个ROS中没有节点管理器，就不会有节点间的通信。需要注意的是，由于ROS本身就是一个分布式网络系统，可以在某一台计算机上运行节点管理器，在其他计算机上运行由该管理器管理的节点。节点管理器通常使用roscore命令运行，它会加载ROS节点管理器及其他ROS核心构件。同时，节点管理器还提供了参数服务器。

(3)参数服务器（Parameter Server)

参数服务器是可通过网络访问的共享的多变量字典，通过关键字存储在节点管理器上。节点使用此服务器存储和检索运行时的参数，还可以改变节点的工作任务。参数服务器使用XMLRPC实现，并运行在ROS节点管理器上。这意味着其API可通过通用的XMLRPC库进行访问。

ROS体系架构

ROS整体架构分析——计算图级

(4)消息(Message)

节点可通过消息实现彼此的逻辑联系与数据交换。消息包含一个节点发送到其他节点的数据信息。消息具有多种标准类型，同时用户也可以基于标准消息开发自定义类型的消息。

(5)主题(Topic)

主题是由ROS网络对消息进行路由和消息管理的“数据总线”。每一条消息都要发布到相应的主题上。当一个节点发送数据时，就说该节点正在向主题发布消息。节点可以通过订阅某个主题，接收来自其他节点的消息。节点可以订阅任何节点发布的主题，而不需要了解向该主题发送消息的节点。这就保证了消息的发布者和订阅者之间相互解耦，完全无需知晓对方的存在。主题的名称必须具有唯一性，否则在同名主题之间的消息路由就会发生错误。

ROS体系架构

ROS整体架构分析——计算图级

(5)主题(Topic)

每一个主题都是强类型的，发布到主题上的消息必须与主题的ROS消息类型相匹配，并且节点只能接收类型匹配的消息。一个节点要订阅一个主题，相关者必须具有相同的消息类型。ROS的主题消息可以使用TCP/IP或UDP传输。基于TCP传输称为TCPROS，它使用TCPIP长连接，这是ROS默认的传输方式。基于UDP传输称为UDPROS，它是一种低延迟、高效率的传输方式，但可能产生数据丢失，所以它适合于远程操控任务。

(6)服务(Service)

服务用于请求应答模型，也必须有一个唯一的名称。当一个节点提供某个服务时，所有的节点都可以通过使用ROS客户端库所编写的代码与之通信。

ROS整体架构分析——计算图级

(7)消息记录包(Bag)

消息记录包是一种用于保存和回放ROS消息数据的文件格式，保存在.bag文件中。消息记录包是一种用于存储数据的重要机制，它能够获取并记录各种难以收集的传感器数据。程序可以通过消息记录包反复获取实验数据，以进行必要的开发和算法测试。在使用复杂机器人进行实验工作时，需要经常使用消息记录包。

ROS体系架构

ROS整体架构分析——开源社区级

ROS开源社区级的概念主要用于ROS资源管理，其能够通过独立的网络社区分享软件和知识。这些资源具体如下。

- **发行版**（Distribution）：ROS发行版是可以独立安装、带有版本号的一系列功能包集。ROS发行版像Linux发行版一样可发挥类似的作用。这使得ROS软件安装更加容易，而且能够通过一个软件集合来维持一致的版本。
- **软件源**（Repositorie）：ROS依赖于共享代码与软件源的网站或主机服务，在这里不同的机构都能够发布和分享各自的机器人软件和程序。
- **ROSWiki**:ROS Wiki是用于记录有关ROS系统信息的主要论坛。任何人都可以注册账户和贡献自己的文件、提供更正或更新、编写教程及其他信息。
- **邮件列表**(Mailing list)：ROS用户邮件列表是关于ROS的主要交流渠道，能够交流从ROS软件更新到ROS软件使用中的各种疑惑或信息。

ROS体系架构

名称系统

当构建一个规模较大且结构复杂的智能机器人系统时，ROS框架通过名称来处理和提取复杂的信息。因此名称在ROS中具有非常重要的作用：节点、主题、服务和参数均有各自的名称。每一个客户端库都支持名称的命令行再映射，这意味着一个编译过的程序可以在运行时被重新构建，以便于其能够在不同的图级拓扑结构中操作。

名称系统——计算图资源名称

计算图资源名称提供了分层命名结构，用于在ROS中计算图中的所有“资源”，如节点、参数、主题和服务。计算图资源名称的机制在ROS中非常有用，对于构建复杂的机器人系统有很大的帮助。

计算图资源名称提供了封装作用。每一个资源被定义在一个命名空间内，该资源可以与其他资源共享。一般来说，资源可以在它们的命名空间内被创建，它们在自己的命名空间内或上一级命名空间内连接资源。连接可以在不同的命名空间的资源之间进行，不过这通常需要编写一段代码来实现。这种封装分离了系统的不同部分，从而避免了因为偶然的名称错误或全局的“名称劫持”而导致出现不期望的结果。

名称系统——计算图资源名称

资源名称的解析可以是相对的，所以资源并不一定需要知道它们在哪个命名空间内。这样可以简化节点的编写，在编写每个节点时可以假设它们就在根命名空间内。当将这样编写的节点集成到更大的系统中时，它们可以被放到另一个指定的命名空间内。

例如，有两个开发者A和B,合作开发一个智能机器人的软件系统。A开发的数个节点与B开发的数个节点中有一个节点名字重复(如/Camera_node)，如果要将A与B的成果部署到同一台智能机器人上，那么只需将A、B的节点放入不同的命名空间内，即可区分同名的节点。如:

/DeveloperA/Camera_node

/DeveloperB/Camera_node

名称系统——计算图资源名称

此外，这种机制也在很大程度上增强了ROS的可复用性。例如，我们为一个双目摄像头开发了一个图像采集节点，该节点发布了两个主题：/Camera_Left与/Camera_Right。若智能机器人需要安装两台这种摄像头同时工作，这时可以在两个不同的命名空间（例如CamA、CamB)内分别运行相同的节点，这样该节点发布的主题也在不同的命名空间内，如：

摄像头A：/CamA/Camera_Left与/CamA/Camera_Right

摄像头B：/CamB/Camera_Left与/CamB/Camera_Right

名称系统——名称的有效性

在ROS中，一个有效的资源名称应该具有如下特征。

- 第一个字符是字母、波浪号或斜线。
- 后续字符可以是字母、数字、下划线或斜线。

“基本名称”中不能有波浪号或斜线。

名称系统——名称的解析

ROS包含了4类型的计算图资源名称：基本名称、相对名称、全局名称和私有名称，其语法分别如下。

- 基本名称：base
- 相对名称：relative/name
- 全局名称：/global/name
- 私有名称：~private/name

默认情况下，名称解析是相对命名空间进行的。如节点/ur5/node1的命名空间为/ur5，因此若该节点使用了“res”资源，那么该名称实际上被解析为/ur5/res。

没有命名空间修饰符的名称都是基本名称。基本名称实际上是一种特殊的相对名称，因此具有相同的解析规则。基本名称用得最多的场合是用来初始化节点名称。

名称系统——名称的解析

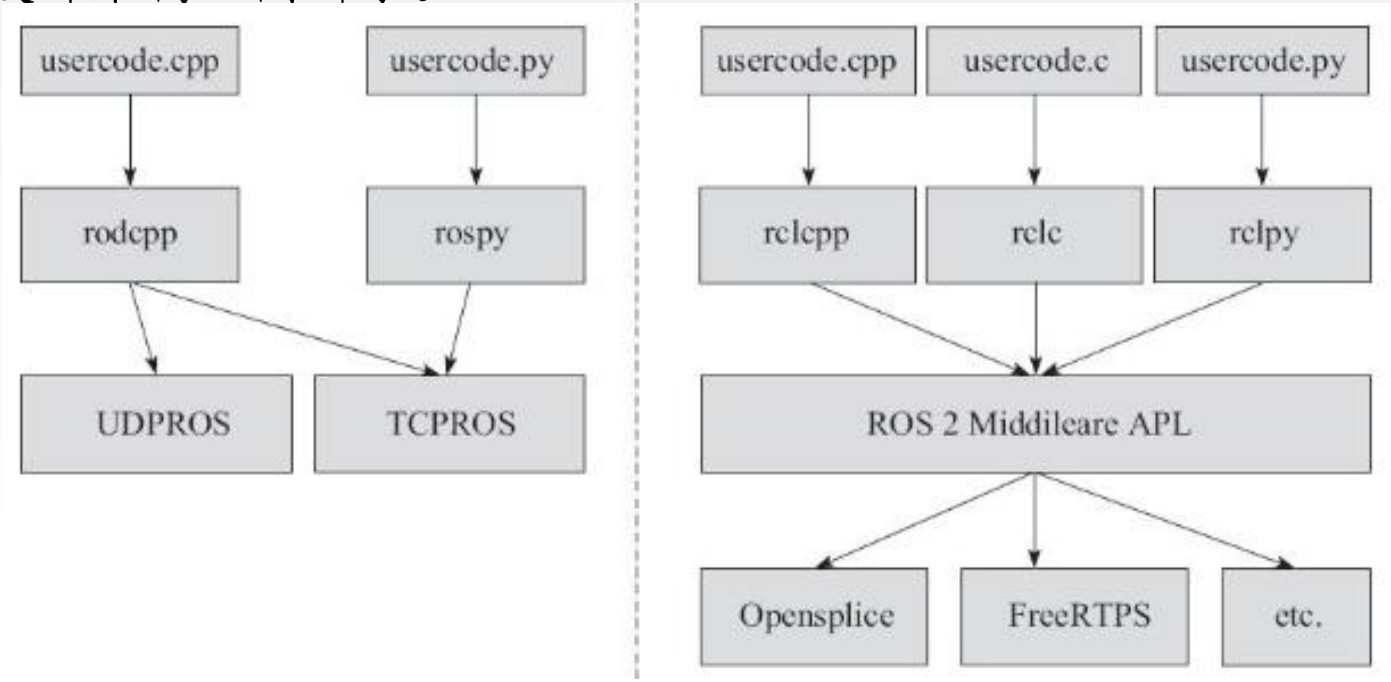
以斜线(/)开头的名称是全局名称，全局名称是被完整解析的。全局名称在一定程度上限制了代码的可移植性，所以应当尽量避免使用。

以波浪号(~)开头的名称是私有名称。私有名称把节点名称转换为一个命名空间。例如，若节点/ur5/node1中使用了“~ res”资源，那么该名称实际上是被解析为/ur5/node1/res。

ROS2.0框架

由于ROS已经存在了较长时间，已采用其开发机器人系统的开发者希望ROS仍然能保持现在的应用状态，而不会被ROS的不断发展所影响，所以ROS2.0是独立于ROS并行的包集，可以单独安装并且可以与ROS进行交互。

ROS2.0是一个与ROS类似的中间件系统，它使用的也是现成的开源库，这样开发者可以维护更少的代码，尤其是非机器人特定代码；充分利用超过开发者能力范围的现有库；受益于其他开发者未来对这些库的改进。ROS与ROS2.0的设计架构如图所示。



ROS2.0框架

ROS2.0的改进主要是为了让ROS更符合工业级的运行标准，采用了DDS(数据分发服务)这个工业级的中间件来负责可靠通信、通信节点动态发现，并用共享内存的方式使得通信效率更高。通过使用DDS,所有节点的通信拓扑结构都依赖于动态点对点的自发现模式，从而去掉了ROS Master这个中心节点。

ROS2.0改变了API。目前ROS代码中大量的API是在2009年发布的，虽然很稳定，但未必是最好的，所以ROS2.0设计了新的API,同时尽最大能力与第一代API结合。所以ROS2.0的关键概念（分布式处理、匿名的发布/订阅消息、带有反馈的RPC、语言无关性等)仍然是与ROS相同的，但ROS2.0API与现有的ROS代码并不兼容，好在目前已有开发者实现了ROS2.0与ROS之间的兼容，并在ROS2.0的软件包集下发布了一个ros1_bridge包，用于实现ROS与ROS2.0之间的双向通信。

ROS2.0框架

目前，ROS2.0已经发布了一部分功能包，世界各地的ROS开发者还在不断地为它作贡献，将新开发的功能包集发布到这个系统中。ROS2.0是ROS的功能扩展和性能优化，

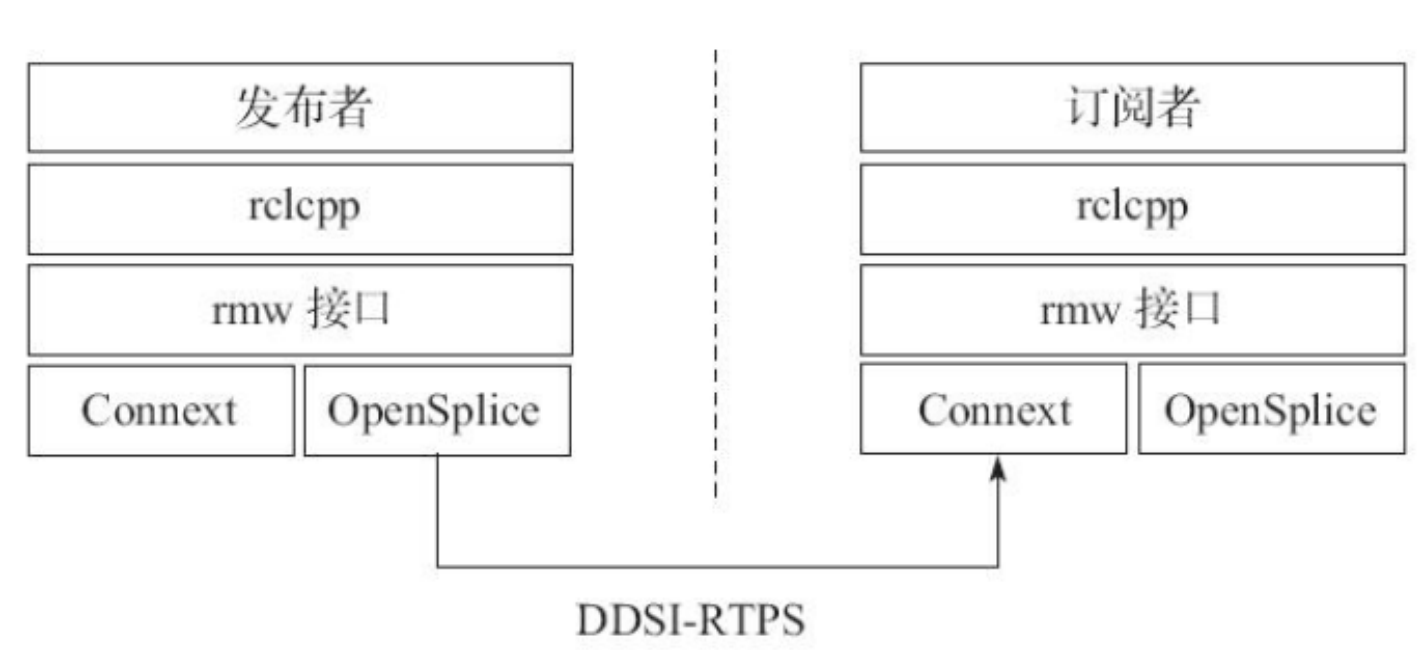
其设计目标主要体现在以下5个方面。

- 支持多机器人系统，包括不可靠的网络。
- 消除原型系统和最终产品之间的差距。
- 可以运行在小型嵌入式平台上。
- 支持实时控制。
- 支持交叉平台。

ROS与ROS2.0之间的主要区别

1发布订阅

下图是ROS2.0实现发布/订阅的结构，ROS2.0将ROS中原来的通信协议TCPROS和UDPROS替换为DDSI-RTPS。此外，API也做了相应的改变，例如，节点的名字没有传递给全局的init（）函数，而是传递给节点的构造函数；spin（）函数不再是全局的，而是通过节点调用等。



ROS与ROS2.0之间的主要区别

2.进程内通信

ROS2.0与ROS的进程内通信具有相同的功能，但是ROS2.0采用了更加安全的模式。

ROS2.0的进程内通信具有如下特点。

- 避免了序列化和逆序列化。
- 避免了网络栈、将数据分成包。
- 通过提供基于API的unique_ptr可安全避免复制。
- 进程内和进程间的通信更加一致。

ROS与ROS2.0之间的主要区别

3. 网络通信

下表列出了ROS和ROS2.0的一些QoS设置对比。

	ROS	ROS 2.0
消息传递	UDPROS/TCPROS	可靠传递
消息存储	队列	保存全部消息
消息保存	暂时保存	永久保存

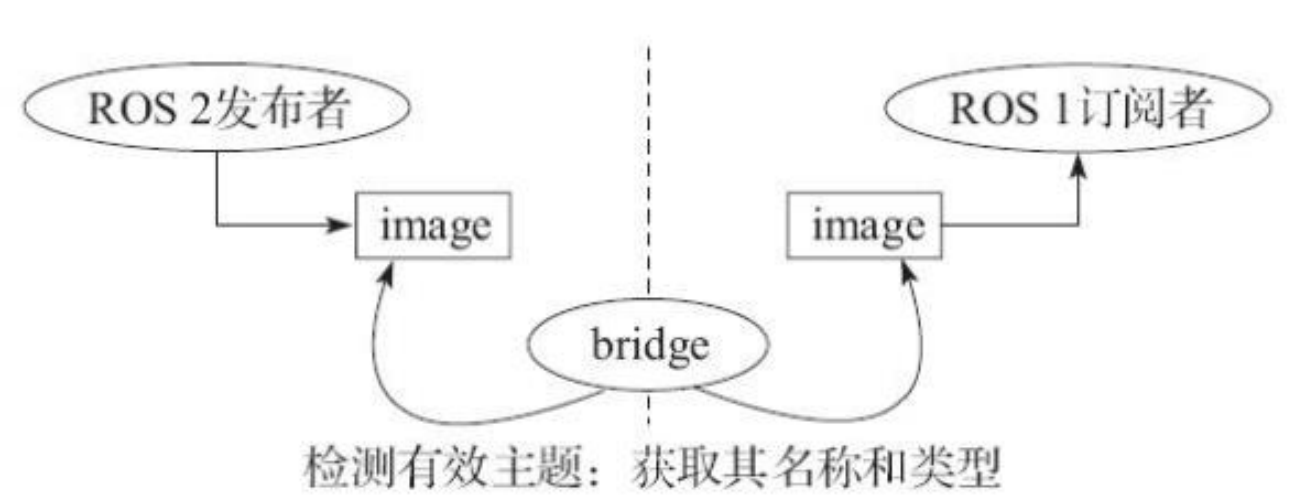
ROS消息传递采用尽量到达，尽最大可能丢失最少的消息，而ROS2.0会保证所有的消息都会到达另一端；ROS采用队列存储消息，队列的长度可以指定，而ROS2.0会存储所有的消息；ROS数据没有持久性，ROS2.0的数据会被发送者持续保存。此外，ROS2.0中的DDS也提供了一些已被工业应用证明有效的QoS:使用UDP(代替TCP)实现多播，这样发布者就不需要为每一个订阅者都复制一份数据了；支持不可靠网络，如蜂窝网络、IoT、高延时的连接等。

ROS与ROS2.0之间的主要区别

4.ROS和ROS2.0之间的软件桥

ROS和ROS2.0通过软件桥(bridge)进行数据通信，其实现的技术背景具体如下。

- 目前bridge是使用C++实现的。
- 消息定义：ROS使用rosmsg API，ROS2.0使用ament资源索引。
- 在ROS和ROS2.0类型间采用自动的映射规则；也可以选择指定通用的规则（在.yaml文件中进行定义）；对于任意的类型对，可以注册一个函数，用于产生ROS和ROS2.0之间的转换函数。



- 建立一个软件桥：ROS使用roscpp,通过pkg-config实现；而ROS2.0使用rclcpp,通过CMake find_package实现。
- 具体实现要求所有的头函数必须没有冲突。

ROS与ROS2.0之间的主要区别

5.基本开发环境的区别

1)ROS：Linux（Ubuntu）；OS X。

2)ROS2.0：Linux（Ubuntu）；OS X；Windows。

从上述所列开发环境可以看出，ROS2.0扩展了Windows基础平台，使其具有更广泛的适应性。

Part 2

ROS通信机制概述

ROS通信机制概述

ROS通信机制概念

ROS的核心功能是提供一种软件点对点通信机制，基于这一机制，开发人员可以非常灵活和高效地组织智能机器人的软件实现。ROS的运行架构是一种基于Socket网络连接的松耦合架构。在这个运行架构中包括一系列进程，这些进程可以驻留在多个不同的主机上并且在运行的过程中通过点对点的拓扑结构实现通信。

ROS通信机制概述

ROS通信机制概念

ROS将每个工作进程看作一个节点，使用节点管理器进行统一管理，并提供一套相应的消息传递机制。在ROS中，所有的消息通信都必须使用节点管理器。ROS的特殊性主要体现在消息通信层，而不是更深的层次。点对点的连接和配置通过XMLRPC机制实现；节点间的数据流通过网络套接字实现，数据流在ROS中被称为消息，模块间的消息传递采用简单的、语言无关的接口定义描述。

ROS底层的通信是通过HTTP完成的，因此ROS内核本质上是一个HTTP服务器，它的地址一般是http:/localhost：11311/，即本机的11311端口，当需要连接到另一台计算机上运行的ROS时，只要连上该机的11311端口即可。

ROS通信机制概述

ROS通信机制的基本要素

ROS通信机制的基本要素主要是“计算图级”介绍中给出的概念，包括节点、节点管理器、参数服务器、消息、服务、主题，这些要素在介绍ROS通信机制中反复出现，为了便于理解，下面简要给出有关定义。

- 节点：节点是主要的计算执行进程。
- 节点管理器：节点管理器的主要目的是实现节点之间的通信，用于主题、服务的注册和查找等，参数服务器也运行在节点管理器之上。
- 参数服务器：用于存储和检索运行时参数。
- 消息：节点之间的通信是通过消息传递来实现的。ROS中包含多种标准类型的消息，同时编程人员也可以自定义消息类型。
- 主题：发布订阅采用主题实现消息通信，每一条消息都要发布到相应的主题中。通过主题实现发布者和订阅者之间的解耦，主题的名称必须是独一无二的。
- 服务：请求应答模型使用服务进行通信，提供一对一的连接。服务也必须有一个唯一的名称。当一个节点提供某个服务时，所有的节点都可以通过使用ROS客户端库所编写的代码与它进行通信。

ROS通信机制概述

ROS通信机制的分类

ROS的运行架构是一种使用ROS通信实现节点间点对点的松耦合网络连接的处理架构，通过提供多种类型通信机制，包括**基于主题的异步数据流通信**、**基于服务的同步RPC(远程过程调用)通信**及**基于参数服务器的数据传递**，支持实现不同需求的网络连接。虽然ROS通信机制目前不具有严格的实时性，但是也能够支撑不少类别的智能机器人开发。

ROS通信机制概述

ROS通信机制的分类

基于主题的异步数据流通信是ROS中最主要的通信方式，它实现了节点间多对多的连接，并且采用单向数据流传送数据。通过主题可以实现发布者和订阅者之间的解耦，因此，主题可以关联任意的发布者和订阅者。

基于服务的同步RPC通信实现一对一的连接，采用请求/应答模型。当一个节点需要向另一个节点发送数据并要从另一个节点中得到响应时，就需要使用该通信方式。

参数服务器是一个可以通过网络访问的多元共享的参数字典，节点在运行时可以在参数服务器上存储或获取参数。参数服务器运行在节点管理器上，它的应用程序编程接口通过通用XMLPRC库进行访问。参数服务器主要用来实现一些参数配置等。

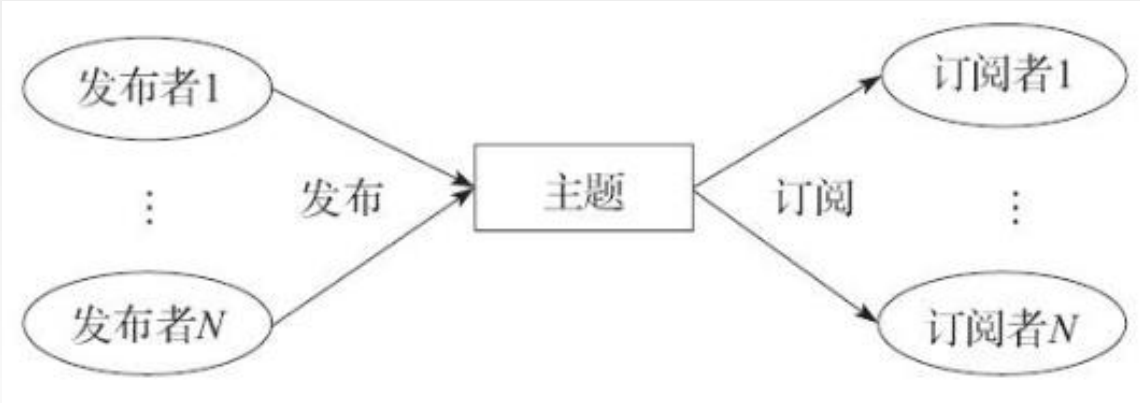
ROS通信机制概述

ROS通信机制的分类

ROS针对实时性要求高的场景还提供了一种通信方式—actionlib，它为优先级任务提供了标准的接口。actionlib与服务通信类似，不同的是它是可剥夺的。在一些情况下，节点请求的某些任务需要长时间执行，并且希望可以在请求执行过程中中断任务的执行，或者在任务执行过程中获得周期性的状态反馈。因此，ROS提供了可剥夺任务调度接口actionlib，该接口提供了服务节点在执行耗时任务时的中断能力和状态跟踪能力。

基于主题的异步数据流通信

主题用于发布订阅模型的消息传递，它拥有独一无二的名字。在智能机器人系统中，若一个节点需要与其他节点进行通信，则只需要在自己关心的主题上发布或订阅消息即可，发布者和订阅者无需知晓彼此的存在，从而实现了发布者与订阅者之间的解耦。发布者、订阅者和主题之间的关系如图所示。



在ROS中，一个主题上可以拥有多个发布者和订阅者，同时一个节点可以发布或订阅多个主题，并且可以随时创建发布者或订阅者。但是每个发布者只能向一个主题发布消息，每个订阅者只能订阅一个主题。

基于主题的异步数据流通信

ROS通过主题实现的节点间交换数据为单向数据流，发布者发布的消息通过数据流的方式传递给订阅者，订阅者不需要反馈给发布者是否收到消息。ROS的主题通信支持TCPROS和UDPROS两种通信协议，TCPROS采用标准的TCP/IP套接字，这是ROS默认的传输方式；UDPROS采用标准的UDP传输方式，它是一种低延时、高效率的传输方式，但可能产生数据丢失，所以它比较适用于远程操控任务。

在一个系统中，必须包含一个节点管理器Master。Master用来管理发布者和订阅者，使得发布者发布的消息可以正确地发送给订阅者，因此每当有一个新的发布者或订阅者产生时，都必须在Master上注册信息。当发布者在Master上注册时，Master会保存发布者的URI(统一资源标识符)和发布者发布的主题；当系统中有一个新的订阅者在Master上注册时，Master会根据订阅者订阅的主题，在保存的信息中寻找与主题匹配的发布者，然后将这些发布者的URI发送给订阅者。订阅者会根据发布者的URI与这些发布者建立一对多的连接，从而接收这些发布者在主题上发布的消息。

基于主题的异步数据流通信

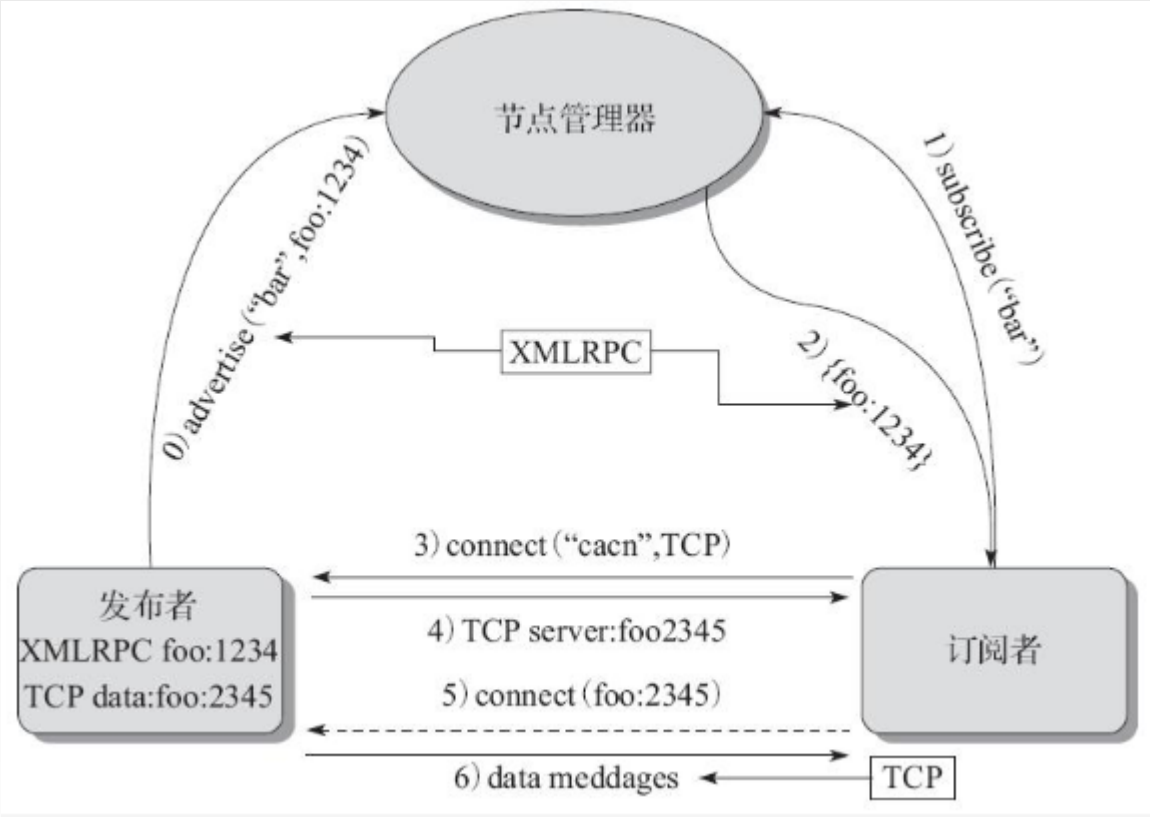
基于主题的数据流通信主要通过以下4步来完成，如图所示。

1）通过XMLRPC，发布者与订阅者在Master上注册，订阅者与发布者协商共同支持的协议(TCPROS或UDPROS)。

2)发布者和订阅者通过互相发送header，建立TCPROS/UDPROS连接。

3)经过以上两步，连接已经建立成功，可以进行数据传送。发布者向特定的主题发布消息，订阅者会在特定的主题上接收消息，然后将消息保存在回调函数队列中，等待处理。

4)最后调用回调函数队列中已经注册的回调函数来处理接收到的消息。



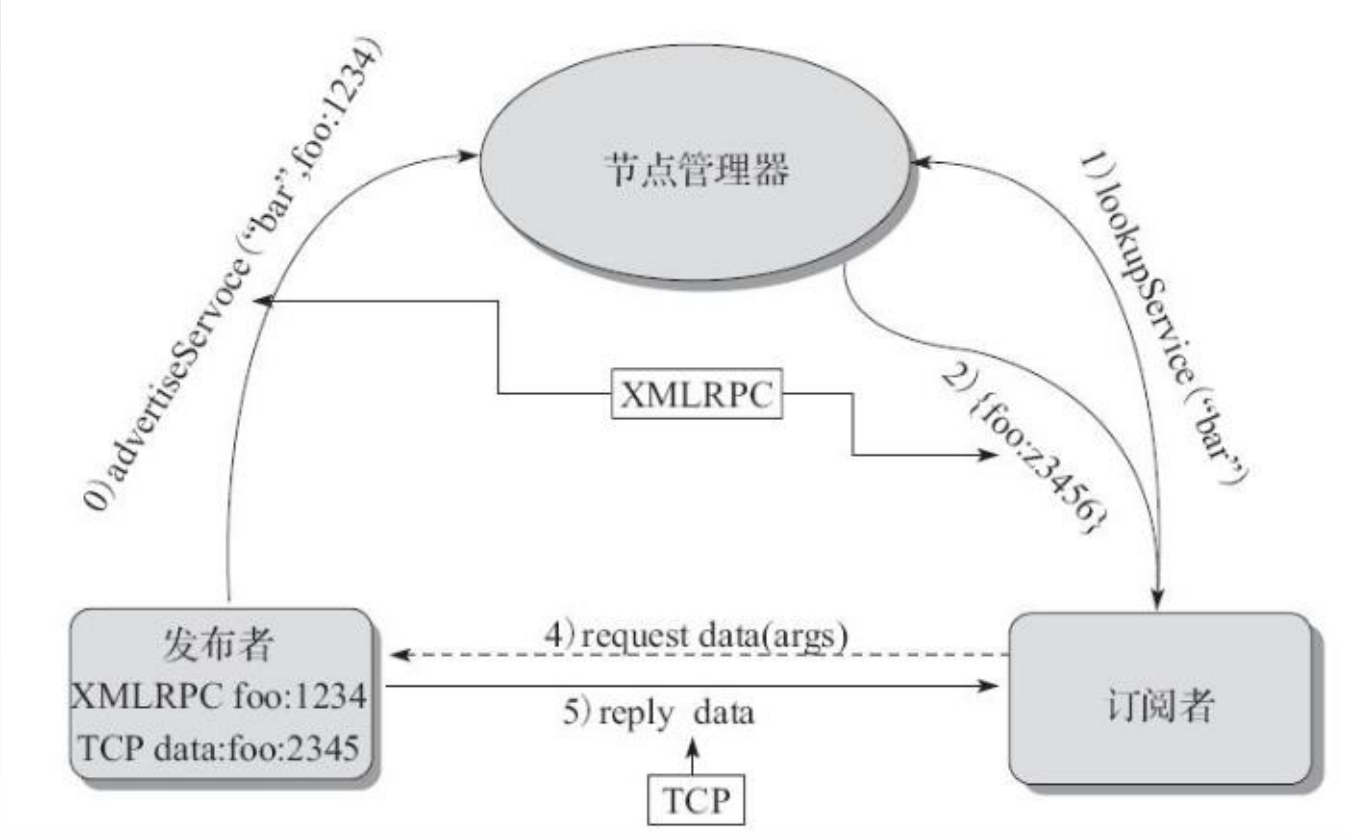
基于服务的同步RPC通信

发布/订阅模型是一个灵活的节点间交流方式，但是它属于多对多、单向传输方式，不适合RPC(远程过程调用)请求/应答模型。请求/应答是分布式系统常用的信息交互方式，所以，ROS提供了基于服务的同步RPC通信方式，用来实现RPC请求/应答模型。服务由一对消息组成，一个用于实现请求，另一个用来实现应答。节点用一字符名定义服务，客户端通过发送一个请求消息调用服务，然后等待响应。

在一个系统中，服务被定义在文件srv中，通过ROS客户端库转换为源码。一个客户端可以与服务器端建立持久连接，这样做有利于提高运行时性能，但当服务器端发生改变时，系统的健壮性就会较差。

基于服务的同步RPC通信

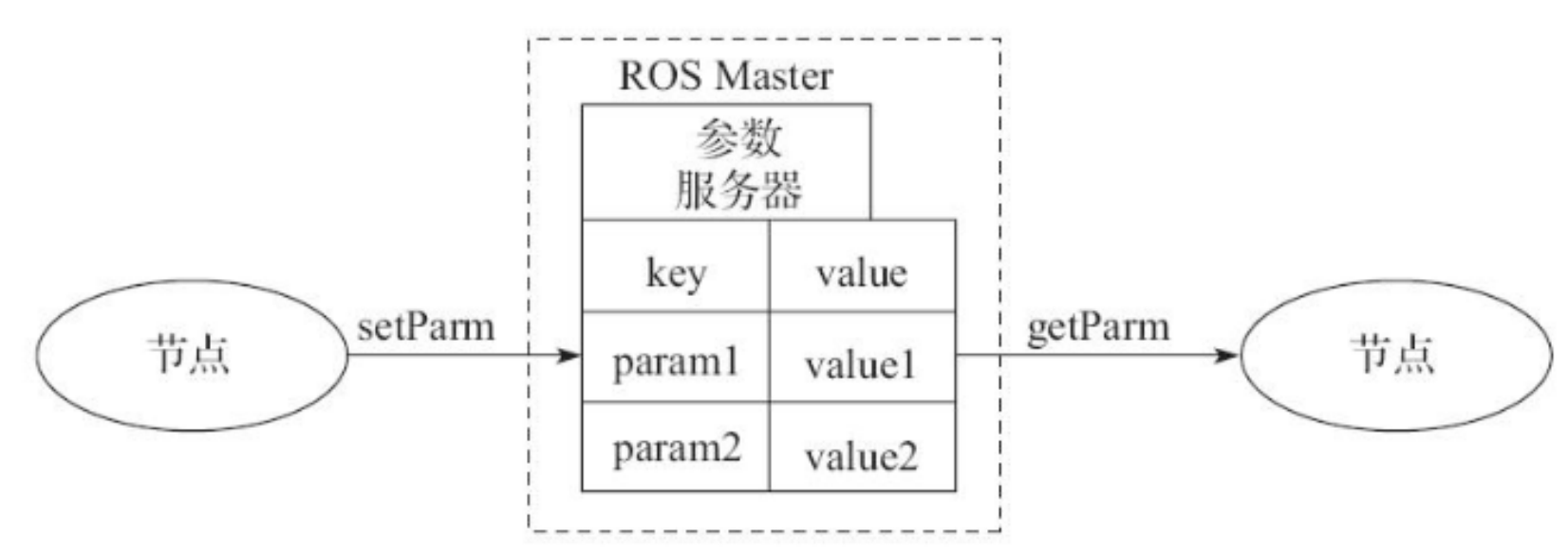
与发布订阅类似，基于服务的同步RPC通信也需要通过XMLRPC在Master上注册服务，客户端节点通过XMLRPC在Master上寻找与服务对应的服务器端节点，然后与其建立TCPROS连接，之后才能进行数据流通信，如图所示。



与发布/订阅不同的是，同步RPC服务通信只支持TCPROS协议，客户端不需要通过XMLRPC与服务器端协商共同支持的协议；RPC服务通信数据流是双向的，需要分别发送请求与响应消息来实现数据通信。

基于参数服务器的数据传递

参数服务器是ROS上一种共享的多变量参数字典，节点利用参数服务器获取和存储运行时参数。参数服务器并不是为了高性能数据存储而设计的，其通常用于存储一些静态的非二进制数据，如配置参数、机器人模型文件等。参数服务器是全局可见的，即所有节点都可以获取和存储参数，在必要时可以删除、修改参数。基于参服务器的数据传递过程如图所示。



参数属于参数服务器管理，也就是说，即使节点创建了参数，但在节点终止时参数仍然继续存在。虽然其设计思想比较简单，但其可提高ROS节点的灵活性和可配置性。参数服务器使用XMLRPC实现，并且本身运行于ROS节点管理器中，这就意味着它的API可通过通用的XMLRPC库进行访问。

基于参数服务器的数据传递

参数定义及特点

参数是存储在节点管理器上的全局变量，由索引和参数值构成。节点在运行时可以在参数服务器上存储和检索参数值。参数服务器的存储结构是map<key,value>的形式，其基本结构如图所示。

基于参数服务器的数据传递

参数定义及特点

参数使用一般的ROS命名规则，这就表明参数要是用于节点或主题，就会有与命名空间相匹配的层次关系，从而避免发生参数命名冲突。这种层次体制允许节点在参数服务器上获取单个参数或参数树。

例如对于下面的参数：

/camera/left/name: leftcamera

/camera/left/exposure: 1

/camera/right/name: rightcamera

/camera/right/exposure:1.1

其中，“：”之前为参数索引，之后为参数值。参数/camera/left/name的值为leftcamera，参数/camera/left/exposure的值为1，参数/camera/right/name的值为rightcamera，参数/camera/right/exposure的值为1.1。

基于参数服务器的数据传递

参数定义及特点

1)获取单个参数的值。例如可以获取/camera/left的值，那么将会获得一个字典：

name: leftcamera

exposure: 1

2)获取参数树。例如可获取/camera的值，将会得到一个参数树，

字典中包含内容为：

left: { name: leftcamera, exposure: 1 }

right: { name: rightcamera, exposure: 1.1 }

基于参数服务器的数据传递

参数类型

参数服务器使用XMLRPC数据类型为参数赋值，为了适应不同应用类型的数据传递，该模式定义了以下参数类型。

32-bit integers:32位整。

Booleans:布尔值。

Strings: 字符串。

Doubles:双精度浮点。

iso8601dates: ISO8601日。

Lists: 列表。

base64-encoded binary data: 基于64位编码的二进制数据。

基于参数服务器的数据传递

参数类型

虽然字典具有特殊的含义，但是也可以在参数服务器上存储字典(如结构体)。与ROS命名层次类似，可以在字典下嵌套字典来表示子命名空间。

例如，在参数gains上设置以下三个参数：

/gains/P = 10.0

/gains/I = 1.0

/gains/D= 0.1

可以单独地获取它们中的一个，如检测/gains/P将会返回10.0。

也可以获取整个字典，当检索/gains时，就会返回一个字典：

{ 'P': 10.0, I': 1.0, 'D' : 0.1 }

Part 3

ROS坐标变换体系ROS TF

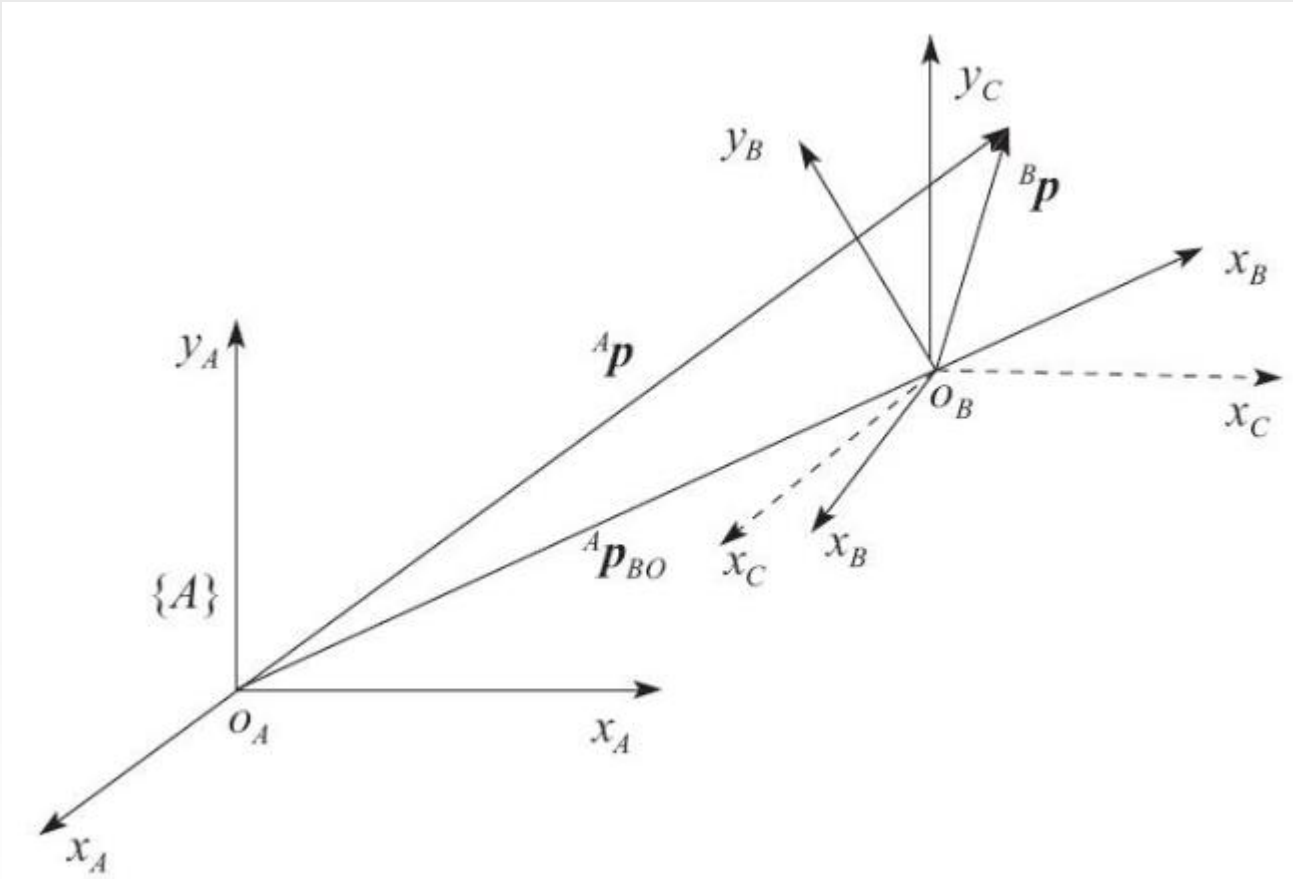
ROS坐标变换体系ROS TF

非齐次坐标系

$$\boldsymbol{R}(x,\theta)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix}$$

$$\boldsymbol{R}(y,\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}$$

$$\boldsymbol{R}(z,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ 0 & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$



$${}^A\boldsymbol{p}={}^A_B\boldsymbol{R}\cdot{}^B\boldsymbol{p}+{}^A\boldsymbol{p}_{BO}$$

ROS坐标变换体系ROS TF

齐次坐标系

$$Rot(x,\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$Rot(y,\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\ 0 & 1 & 0 & 0\\ -\sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$Rot(z,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

旋转矩阵

$$Trans(a,b,c)=\begin{bmatrix}1 & 0 & 0 & a\\ 0 & 1 & 0 & b\\ 0 & 0 & 1 & c\\ 0 & 0 & 0 & 1\end{bmatrix}$$

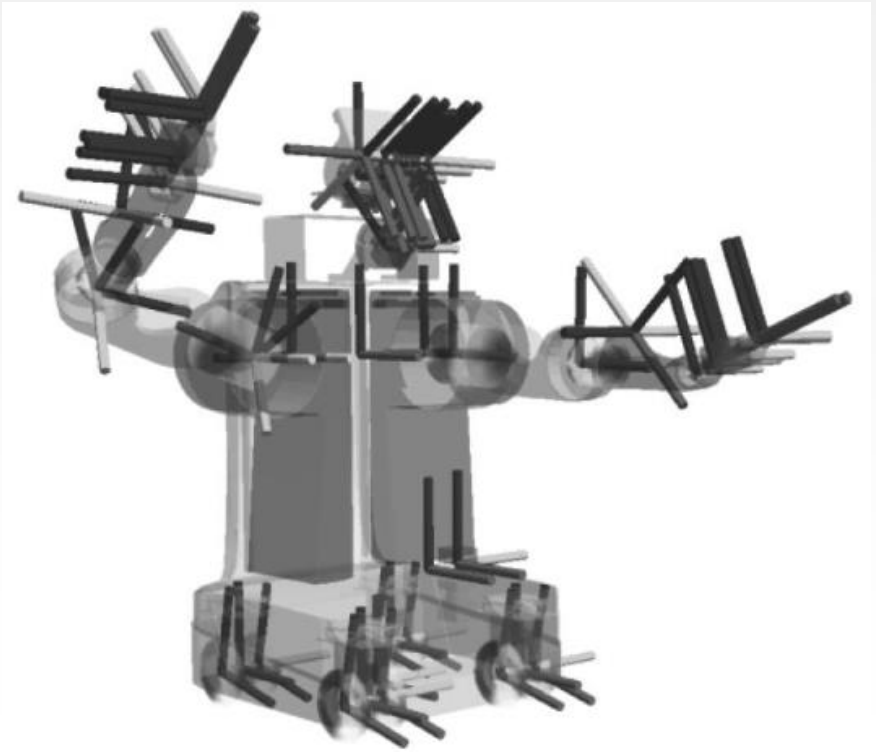
平移矩阵

如果让一物体绕z轴旋转90°，接着绕y轴旋转90°，再沿x轴方向平移5个单位，那么可以用下式来描述这一变换：

$$T=\text{Trans}(a,b,c)\,\text{Rot}(y,90)\,\text{Rot}(z,90)=\begin{bmatrix}0 & 0 & 1 & 4\\ 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

ROS坐标变换体系ROS TF

机器人想要完成预定的任务，必须清楚自身及周围环境的位置关系。例如，当我们需要让一个带有机械爪的机器人抓取一个球时，为了完成这一简单任务，就必须得到球和夹持器之间的位置关系。为了获取这一位置关系可能需要布置一些传感器。传感器得到了球相对于自身的位置，然后需要进行坐标转换，才能得到球相对于机器人基座的位置、相对于机器人手腕的位置，以及相对于最终夹持器的位置。机器人模型的坐标结构简图如图所示。



图中的每个三脚架均表示一个坐标系，它们之间存在着一定的转换关系。在ROS的早期发展阶段，坐标系相关的计算被认为是一个几乎所有机器人开发者都会遇到的难点，由于坐标系变换是一个比较复杂的问题，同时，在机器人系统中，坐标系之间的变换数据往往是分布式的，这也使其成为开发者面临的一个技术挑战。

ROS坐标变换体系ROS TF

ROS tf（transform）库的目的是为了提供一种标准的手段跟踪坐标系，以及整个系统内的坐标变换数据。tf使得其他构件可以方便、放心地使用坐标系中的数据，而不需要了解整个系统内的坐标系细节。机器人系统变得越来越复杂，对于任何一个构件，都能够专注于自身的任务，因此，提供相关的坐标系及其数据是很重要的。

ROS TF原理分析——相关数据结构

ROS tf为了实现坐标变换，设计了相关数据结构，描述坐标系及其转换关系。为了深入分析这些数据结构的组成及其作用，我们将其分为数学相关和转换相关两类数据结构。

数学相关的数据结构及其基本函数

(1)空间点和空间向量描述——Vector3

在tf功能包的头文件Vector3.h中定义了Vector3类，用来表示三维空间的一个点或一个向量。Vertor3定义的一个主要成员变量为：tfScalarm_floats[4]；/tfScalar相当double

这里用x、y、z、w分别来表示m_floats[0]、m_floats[1]、m_floats[2]、m_floats[3]。

其中x、y、z对应三维空间的三个坐标参数，而w则用于控制数据是否对齐，正常情况下为0.0。

成员变量m_floats表示---一个向量或一个点。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

向量主要用于表示坐标系转化中坐标原点之间的平移关系。

Vector3中定义了一组与向量相关的操作函数，具体的函数如表所示。

编号	函数名称	函数功能
1	tfScalar tfDot(const Vector3& v1, const Vector3& v2)	计算两个向量的点积（内积）
2	tfScalar length()	计算向量的长度（模）
3	Vector3& normalize()	求与已知向量同向的单位向量
4	tfScalartfAngle(const Vector3& v1, const Vector3& v2)	计算两个向量的夹角
5	tfScalartfDistance(const Vector3& v1, const Vector3v2)	计算两个向量之间的距离
6	Vector3 tfCross(const Vector3& v1, const Vector3& v2)	计算两个向量的叉积（外积）
7	void setInterpolate3(const Vector3& v0, const Vector3& v1, tfScalar rt)	利用已知的两个向量线性插值得到介于两者之间的一个近似向量
8	Vector3 rotate(const Vector3& wAxis, const tfScalar angle)	计算指定向量绕给定单位向量旋转角度 angle 之后得到的向量

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

除了表中所列的函数之外，tf还定义了一些向量基本运算的函数，如向量之间的加关系、减关系、乘关系（不同于内积和叉积）、除关系等。利用这些函数及表格中所述的函数，tf可以很好地处理不同向量之间的关系。已知两个向量的坐标，可以利用表格中的相关函数计算得到它们之间的夹角和距离，以便于后续操作的继续进行，进行旋转或平移。

Vector3中定义的成员变量m_floats不仅可以独立表示一个向量或是一个点，同时在定义后述的Matrix3x3(表示一个3×3矩阵)时，它也可以作为Matrix3x3的成员变量中的一行。在处理Matrix3x3的相关计算时，把其中的一行当成一个整体并且利用Vector3中定义的向量处理函数，也能使处理数据变得更加有效率，这也说明Vector3类一个好的设计对提升整个系统的计算效率是很有必要的。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

(2)旋转的四元数——Quaternion

在tf功能包的头文件Quaternion.h中定义了Quaternion类，它用来表示在三维空间当中的一次纯旋转。Quaternion类继承于它的父类QuadWord。在QuadWord类当中主要定义的成员变量如下:

tfScalar m_floats[4]; / tfScalar 相当于double

本节用x、y、z、w分别来表示 m_floats[0]、m_floats[1]、m_floats[2]、m_floats[3]。一般来说，有旋转矩阵、欧拉旋转及四元数三种方式表示旋转。

旋转矩阵是使用一个特定的矩阵来表示绕任意轴旋转。欧拉旋转则是通过定义坐标轴顺序（例如先x、再y、最后z)和绕每个轴旋转的角度来表示一次旋转，实际上是一系列坐标轴旋转的组合。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

四元数本质上是一种高阶复数，是一个四维空间，相当于复数的二维空间。复数由实部和虚部组成，即x=a+bi(i是虚数单位，$i^2=-1$)。四元数和复数的定义是类似的，不同的是，它的虚部包含了三个虚数单位，i、j、k,即四元数可以表示为Q=w+xi+yj+zk。QuadWord类当中的成员变量m_floats的4个变量分别对应这里的x,y,z,w。结合具体的例子来说，可以这样理解一个四元数。给定一个
单位长度的旋转轴(a,b,c)和一个角度θ，对应的四元数为：

q= (a*sin(θ/2)， b*sin(θ/2)， c*sin(θ/2)， cos(θ/2))

三种旋转的表示方法中，每种方法都有各自的优缺点。其中四元式表示方法有一个重要的优点是可以提供平滑的插值，这是ROS所必要且特有的。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

QuadWord类中仅仅定义了主要的成员变量，而Quaternion类中则定义了四元式的相关操作函数，具体函数如表所示。

编号	函数名称	函数功能
1	setRotation(const Vector3& axis, const tfScalar& angle)	已知旋转轴和旋转的度数，计算对应的旋转四元式
2	Quaternion(const tfScalar& yaw, const tfScalar& pitch, const tfScalar& roll)	已知欧拉旋转的三个角度，计算对应的旋转四元式
3	Quaternion inverse() const	计算逆旋转的旋转四元式
4	Quaternion operator+(const Quaternion& q2)	计算两个旋转叠加的旋转四元式
5	Quaternion operator*(const Quaternion& q1, const Quaternion& q2)	计算两个旋转的迭代
7	tfScalar angle(const Quaternion& q1, const Quaternion& q2)	计算两个旋转之间的角度差
7	slerp(const Quaternion&q1, const quaternion& q2, const tfScalar& t)	利用球面插值得到介于两个已知旋转之间的近似值
8	Vector3 getAxis()	利用四元式得到旋转的轴
9	Vector3 quatRotate(const Quaternion& rotation, const Vector3& v)	计算一个向量在旋转之前的数值，用于旋转和平移的迭代

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

除了表中所列的函数之外，Quaternion类中还定义了计算四元式的点积、计算两个旋转之间的差等相关函数。利用上述函数，tf可以解决与旋转相关的问题。

例如，从A坐标系到B坐标系有一次旋转，然后B又旋转一次得到C坐标系，要求计算A到C之间的旋转关系，此时可以利用旋转的迭代函数去得到所需的结果。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

(3)3x3的旋转矩阵—Matrix3x3

在tf功能包的头文件Matrix3x3.h中定义了Matrix3x3类，它用来表示三维空间中的旋转。它定义的主要成员变量如下：

```
Vector3 m_el[3]; //存储矩阵中的数据，
每一个Vector3变量代表矩阵的一行
```

tf定义旋转矩阵的主要目的是便于利用矩阵的乘法，从源数据计算得到旋转之后的目标数据。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

为了处理旋转矩阵的相关数据，Matrix3x3类定义了相应的函数，其中重要的函数如表所示。

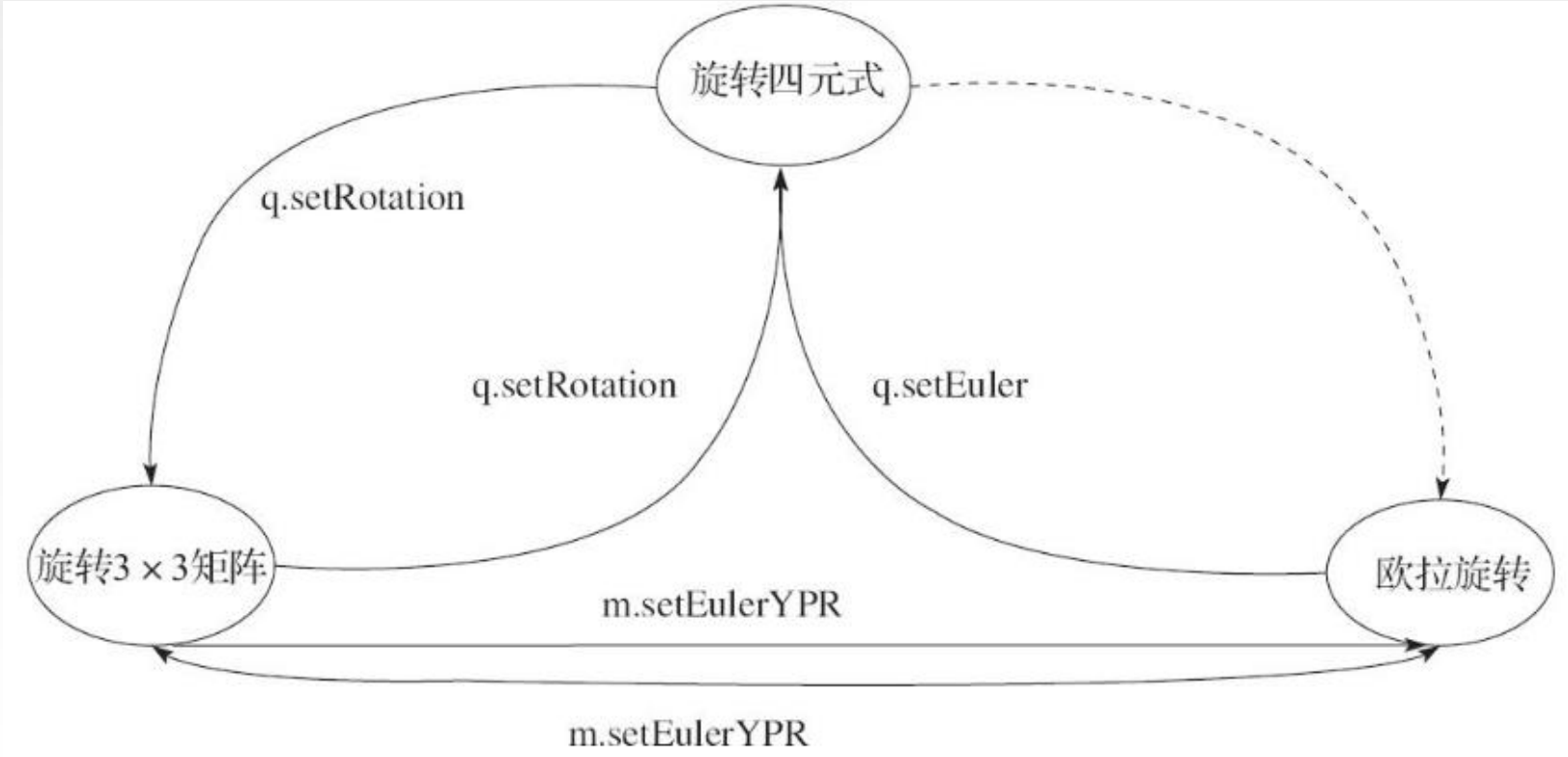
编号	函数名称	函数功能
1	void setRotation(const Quaternion& q)	利用旋转四元式得到一个旋转矩阵
2	void setEulerYPR(tfScalar eulerZ, tfScalar eulerY,tfScalar eulerX)	利用欧拉旋转的三个角度得到对应的旋转矩阵
3	void getRotation(Quaternion& q)	利用旋转矩阵得到旋转四元式
4	void getEulerYPR(tfScalar& yaw, tfScalar& pitch,tfScalar& roll)	利用旋转矩阵得到欧拉旋转的三个角度
5	Matrix3x3 operator*=(const Matrix3x3& m)	定义两个矩阵的乘法运算
6	Matrix3x3 transpose()	计算矩阵的转置
7	Matrix3x3 inverse()	计算矩阵的逆
8	Vector3 operator*(const Matrix3x3& m, const Vector3& v)	定义 3 × 3 矩阵和三维向量的乘法

除了表中所列举的函数，Matrix3x3类当中还定义了计算矩阵余子式，计算伴随矩阵，矩阵对角化，矩阵相乘的转置，和转置后相乘的函数，基本上包括了对矩阵的所有的操作。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

从本质上来说，Quatemion类和Matrix3x3都可以用来表示旋转，在实际运用中还会用到欧拉旋转，在tf中定义了它们之间的相互转换关系，如图所示。



q和m分别表示一个四元式的类对象和3x3矩阵的类对象。图中的实线表示在tf中给出了它们之间的转换关系，实线箭头上的函数表示在转换中调用的具体函数。目前tf尚未给出旋转四元式到欧拉旋转之间的转换关系。

ROS TF原理分析——相关数据结构

数学相关的数据结构及其基本函数

前文已经论述过，Quaternion的优点是便于插值。而旋转矩阵的优点则是计算方便，利用矩阵乘法，我们很容易就能得到想要的数据，但它的缺点刚好就是不适合插值。欧拉旋转描述的特点是更加直接，易被人理解。三者同时存在并且存在相互转换的关系，使得tf可以根据不同的情况选择不同的表示方式，以便更好、更快、更准确地处理实际的问题。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

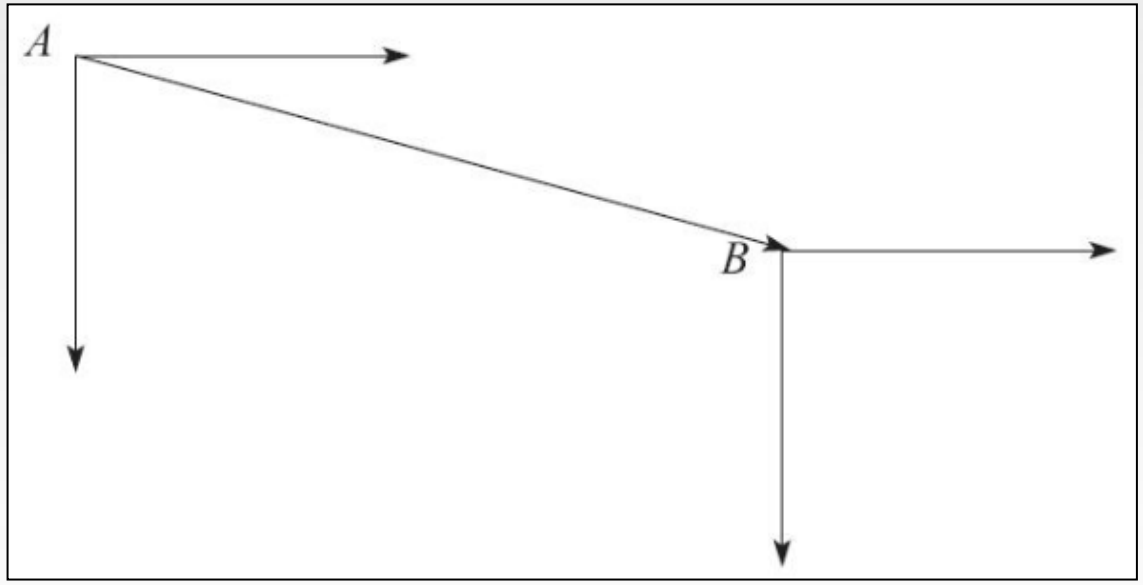
(1)齐次坐标变化——Transform在tf功能包的头文件Transform.h中定义了Transform类，它用来表示只包括旋转和平移（没有缩放和修剪）的变化。在该类中定义的主要成员变量为：

```
Matrix3x3 m_basis;
Vector3 m_origin;
```

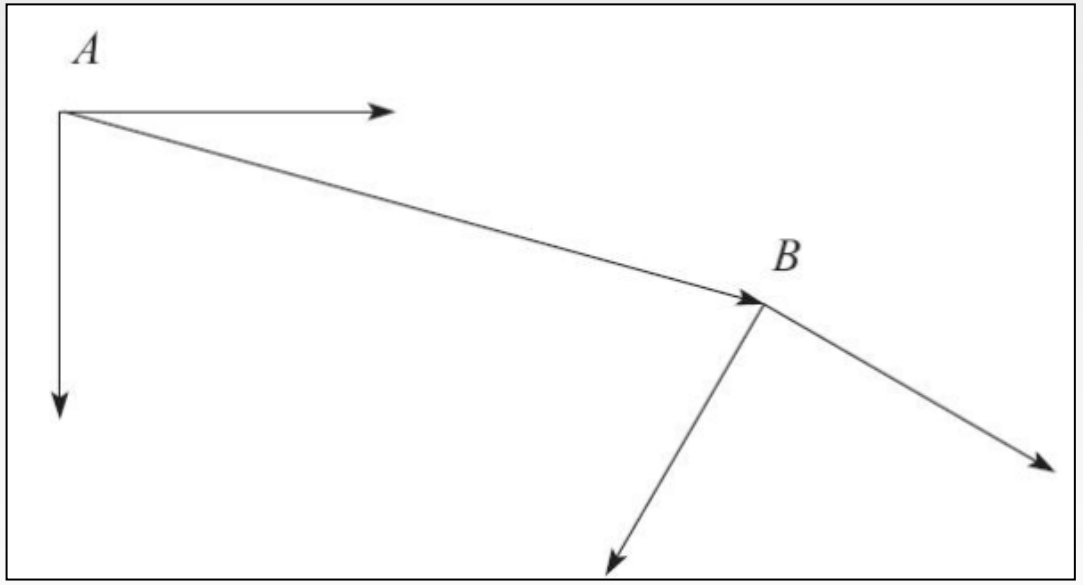
其中矩阵m basis表示坐标系之间的一个旋转关系，这个旋转矩阵可由旋转四元式或欧拉旋转的三个角度值计算得来。另外一个成员变量m_origin是一个向量，表示一个坐标原点和另外一个坐标系原点之间的位移关系。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数



坐标系之间的平移关系



坐标系之间的平移和旋转关系

ROS TF原理分析——相关数据结构

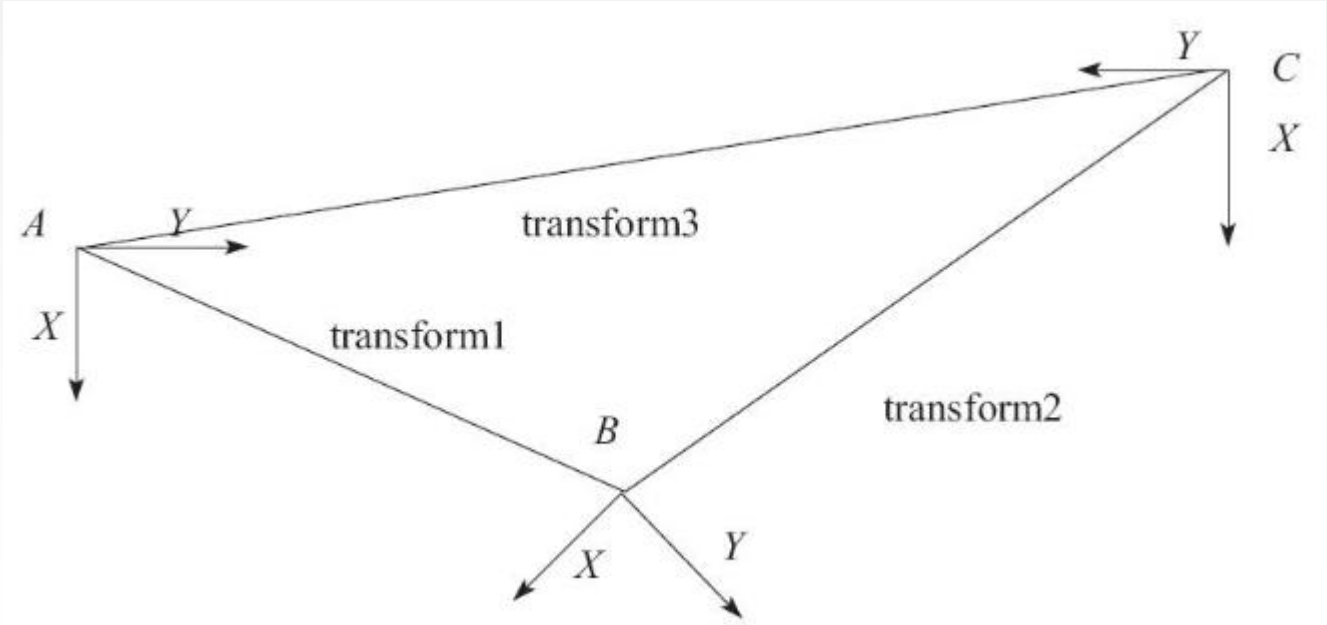
与转换相关的数据结构及其基本函数

Transform类包含以下两个重要的成员函数：

void mult(const Transform& t1, const Transform& t2) //计算t1,t2的迭代变换

Transform inverse()//返回当前变换的逆变换

利用上述函数，tf可以很容易地解决下面的问题，如图所示。



已知坐标系A到坐标系B的转换为transform1，以及坐标系B到坐标系C的转换为transform2，此时利用mult函数可以计算出A到C的转换：

transform3=transform1*transform2

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

除了上述两个函数之外，Transform类中还定义了与它的成员变量相关的一些函数。例如，已知一个变化，求其中的旋转关系，或者是平移关系。反之，给出两个坐标系之间的旋转角度和平移向量也能构建一个标准的Transform对象。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

(2)三维坐标系中对象的类模板——Stamped

在tf功能包的头文件transform_datatypes.h中定义了该类模板，它用来泛指三维空间中的一个点、一个向量或一个位姿。它的两个成员变量为：

ros: Time stamp_; 与这个数据对象对应的时间戳

std:string frame_id_;与这个数据对象对应的坐标系

Stamped类模板当中一个重要的函数如下：

void setData(const T& input) //填充具体的数据对象对应的相关参数

Stamped类主要用于数据转换函数的参数和消息的通信，利用模板的设计，可以使后续设计转化为函数更加简洁，缩减代码量，从而提升系统的效率。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

(3)齐次变换的存储结构——TransformStorage在tf功能包的头文件time_cache.h中定义了TransformStorage类，它定义了齐次变化在系统中的存储结构，主要成员变量如下：

```
tf: Quaternion rotation_;//两个坐标系之间的旋转四元式
tf: Vector3 translation_;//两个坐标系原点平移的向量
ros: Time stamp_;//与这个齐次变换相关联的时间戳
CompactFrameID frame_id_;//齐次变换中的父亲坐标系ID,指的是初始坐标系
CompactFrameID child frame id_;//齐次变换中的子坐标系ID,指的是转换到的坐标系
```

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

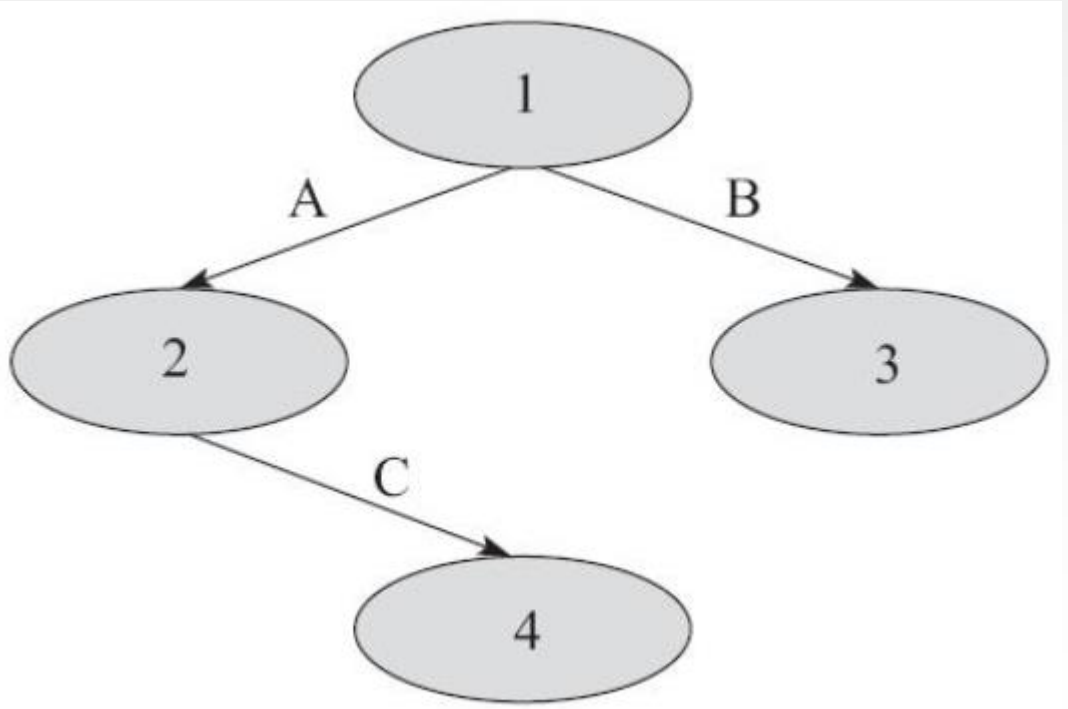
利用TransformStorage类，可以轻松建立坐标系转换之间的树形结构，下面是一个简单的例子。假设在系统当中同时存在三个TransformStorage的对象，分别是A、B和C并且A、B、C满足下面的条件：

A.frame id =1,A.child frame id =2

B.frame_id =1,B.child frame id =3

C.frame id =2,C.child frame id =4

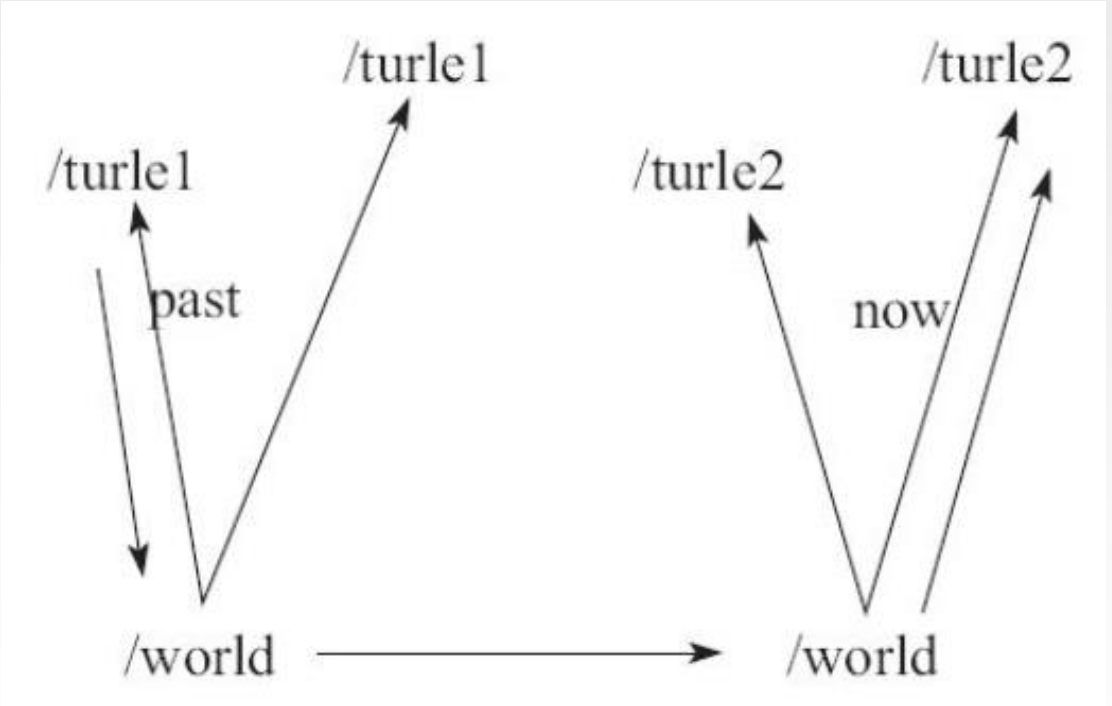
根据A、B、C的成员变量，可以得到坐标系1、2、3、4，满足如图所示的关系。通过对图所示结构树的遍历，tf可以获取1、2、3、4号坐标系任意两者之间的转化关系。



ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

除了增加齐次转换对应的父子坐标系之外，TransformStorage类当中还增加了时间戳成员变量，时间戳的作用是区分不同时刻两个坐标系之间不同的转换关系，如图所示，在past时刻有一个数据从坐标系turtle1转换到坐标系world；而在now这个时刻，又有一个数据从坐标系world转换到turtle2上，在此转换过程当中，时间参数就起到了非常重要的作用，它可以区分不同时刻的转换数据。从整体上来看，tf功能包的主要功能是让用户可以跟随时间的变化跟踪多个不同的坐标系。时间上的要求使得时间戳成员变量必须存在，并且是一个非常重要的部分，在其他部分的一些设计当中，时间参数也有着非常重要的作用。



ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

(4)坐标转换信息的消息类—StampedTransform

在tf功能包的头文件transform_datatypes.h中定义了该类，它是Transform类的一个子类。它与TransformStorage类的设计类似，在Transform的基础上增加了以下几个成员变量：

ros: Time stamp_;//与这次转换相对应的时间戳

std:string frame_id_;//与这次转换相对应的父亲坐标系的名称

std: string child frame id_; //与这换次转换相对应的子坐标系的名称

从TransformStorage和StampedTransform新增的成员变量的不同点来看，Stamped-Transform表示转换关系的父子坐标系是它们的名称，而在TransformStorage中使用的是坐标系内部存储的ID，导致这个差异的原因正是这两个类的用途不同，前文已经介绍过，TransformStorage是用于转化数据的内部存储，而Stamped-Transform更多的是用作消息类型。在具体节点调用tf功能包中的有关模块与其他节点进行通信时，StampedTransform往往作为其中的消息类型进行传输。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

(5)坐标系转换类——Transformer

在tf功能包的头文件tf.h中定义了Transformer的基本结构和框架，在源文件tf.cpp中对具体的功能模块进行了具体实现。由于Transformer类中的功能模块是整个tf中较为核心的部分，因此本章只列举它的模块和结构，功能模块的具体实现将在后续的章节中单独讲解。首先，Transformer类中所定义重要成员变量如下：

```
static const unsigned int MAX_GRAPH_DEPTH = 100UL;
static const double DEFAULT_CACHE_TIME;
static const int64_t DEFAULT_MAX_EXTRAPOLATION_DISTANCE=0ULL
```

其中，成员变量MAX_GRAPH_DEPTH的作用是设定了一个在递归搜索坐标系树形结构时最大深度，这样的设置是为了防止树形结构中可能出现的环状异常，防止程序出现死循环的现象。成员变量DEFAULT CACHE TIME用于设置一个默认的发布坐标系转换的频率。而成员变量DEFAULT_MAX_EXTR-APOLATION DISTANCE定义的是一个外插值的默认最大间隔，用于在插值时使用。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

除了这几个主要的成员变量之外，Transformer类当中还定义了如表所示的重要的成员函数。

编号	函数名称	函数功能
1	bool Transformer::setTransform(const StampedTransform& transform, const std::string& authority)	添加一个新的坐标系转化信息到特定的数据结构当中，若添加成功则返回true值，反之为false
2	void lookupTransform(const std::string& target_frame, const std::string& source_frame, const ros::Time& time, StampedTransform& transform)	寻找在给定时刻，源坐标系和目标坐标系之间的坐标转换关系
3	bool canTransform(const std::string& target_frame, const std::string& source_frame,const ros::Time& time, std::string* error_msg = NULL)	判断在给定时刻，源坐标系和目标坐标系是否存在可行的转换关系
4	bool waitForTransform(const std::string& target_frame, const std::string& source_frame, const ros::Time& time, const ros::Duration& timeout,const ros::Duration& polling_sleep_duration = ros::Duration(0.01), std::string* error_msg = NULL)	每隔一段时间间隔检测在给定时刻两个坐标系之间的转化是否存在，并且会处于可阻塞的状态

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

除了这几个主要的成员变量之外，Transformer类当中还定义了如表所示的重要的成员函数。

编号	函数名称	函数功能
5	bool frameExists(const std::string& frame_id_str)	检测一个坐标系是否在坐标系树形结构当中
6	bool getParent(const std::string& frame_id, ros::Time time, std::string& parent)	得到给定坐标系在树形结构中的父亲坐标系
7	void transformX(const std::string& target_frame, const Stamped<X>& stamped_in, Stamped<X>& stamped_out)	计算给定数据类型在目标坐标系中的数据值，X代表不同的数据类型，如点、向量、旋转等
8	std::string allFramesAsString()	查看所有存在于树形结构中的坐标系名称

Transformer类一般不会单独使用，其设计的主要目的是为特定的功能模块提供相应的函数功能。

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

(6)与时间相关的坐标系转换关系——Time_Cache

在tf功能包的头文件time_cache.h和源文件cache.cpp中定义了Time_Cache类。该类的主要功能是将坐标系之间的转换与时间紧密地联系起来。可以这样理解该类，它利用特殊的结构将一个坐标系与另外一个坐标系在不同时刻的转换关系存储起来，并且提供了相应的函数来操作这些数据。TransformStorage主要用于转换关系的存储，在Time_Cache类中，它有着重要的作用。首先Time_Cache当中最重要的一个成员变量如下：

```
typedef std:set<TransformStorage> L_TransformStorage;
L_TransformStorage storage_;
```

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

其中storage_中存放的每一个TransformStorage对象都有一个共同的特点，它们对应转换的父坐标系和子坐标系都是同一个，不同之处在于转换所对应的时刻不一致。有些节点会以一定的频率不断地发布坐标转换关系，正因为如此，需要记录不同时刻的坐标转换关系。由此可以推知，每一个Time_Cache对象存储的都是一对坐标系在不同时刻的转换关系。由于tf中的坐标系之间是以树形结构组织的，每一个坐标系只有一个父坐标系，具有唯一性，所以Time_Cache类的设计思想是，每一个坐标系都对应一个Time_Cache对象（根节点除外，它用0表示），而该对象存储了它与它的父坐标系之间在不同时刻的转化关系。正因为这个特点的存在，通过对Time_Cache对象的访问，tf可以获取坐标系之间的父子关系，同时也可以完成系统中整个坐标系树的遍历。此功能是实现查找在任意时刻任意两个坐标系之间的转换的前提和基础。Time_Cache类中定义了很多与时间相关的

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

Time_Cache类中定义了很多与时间相关的间的转换的前提和基础。Time_Cache类中定义了很多与时间相关的函数，并且也是整个系统的核心部分，具体函数列表如表所示。

编号	函数名称	函数功能
1	uint8_t findClosest(const TransformStorage*& one, const TransformStorage*& two, ros::Time target_time, std::string* error_str)	查找最接近给定时刻的两个转换关系数据
2	bool getData(ros::Time time, TransformStorage & data_out, std::string* error_str)	得到在给定时刻两个坐标系之间的转换关系数据
3	void interpolate(const TransformStorage& one, const TransformStorage& two, ros::Time time, TransformStorage& output)	利用两个对应时刻接近的坐标系转换数据，利用插值得到近似的转换数据
4	CompactFrameID getParent(ros::Time time, std::string* error_str)	获取在给定时刻坐标系转化关系中的父亲坐标系 ID，不存在就取 0
5	bool insertData(const TransformStorage& new_data)	插入一次新的坐标系转化数据
6	void pruneList()	删除存储时间超过最大存在时间间隔的坐标系转化数据

ROS TF原理分析——相关数据结构

与转换相关的数据结构及其基本函数

(7) 异常类——Exception

在tf处理数据的过程中，在不同的阶段和不同的模块当中，不可避免地会出现各种各样的异常。通过对异常进行记录和处理，可以让tf更好地掌控各个模块的运行状态，这对维持整个系统会有很重要的作用。tf中定义的异常如表所示。

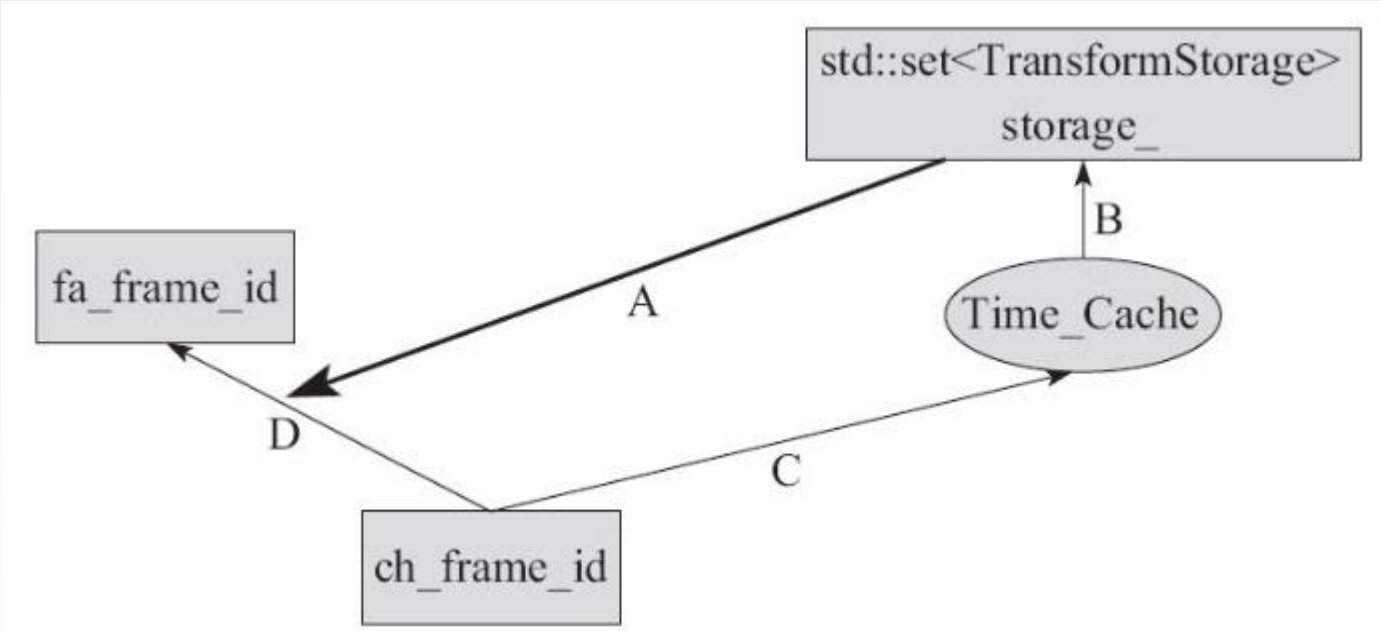
编号	异常名称	具体含义
1	LookupException	在结构树上找不到指定的坐标系
2	ExtrapolationException	在需要进行插值计算时，数据出现异常
3	InvalidArgumentException	基本数据结构中的参数不满足条件
4	TimeoutException	可阻塞的函数中出现超时的情况
5	ConnectivityException	在查找两个坐标系之间的转化关系时，因两个坐标系不在同一棵坐标系树上的异常

ROS TF原理分析——关键模块的实现

1. 与时刻相关联的坐标系转换存储管理

(1) 与时刻紧密相连的tf坐标系转换设计

Time-Cacha与坐标系的转关系如图所示。



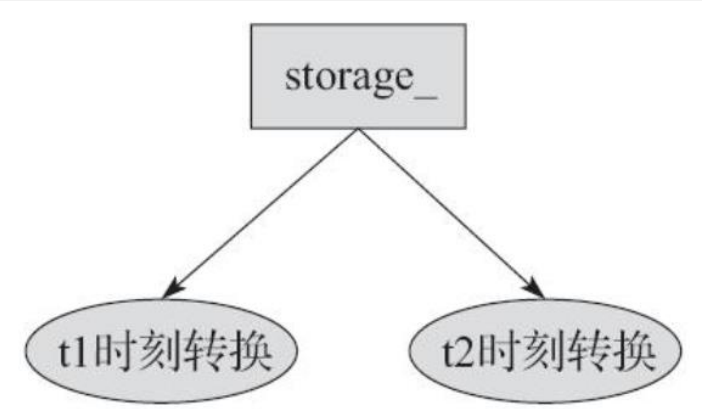
如图所示，每一个坐标系对应一个Time_Cache对象，图中的箭头C表明了这一点，箭头B表示Time_Cache有一个成员变量storage_，而粗箭头A表示storage_存储着多组坐标系ch_frame_id和fa_frame_id之间的转换，并且它们是以时间为序排列的。通过对storage_的相关访问和操作，可以获取两个坐标系在任意时刻转换的数据，并且可以对它们进行删除和修改。

ROS TF原理分析——关键模块的实现

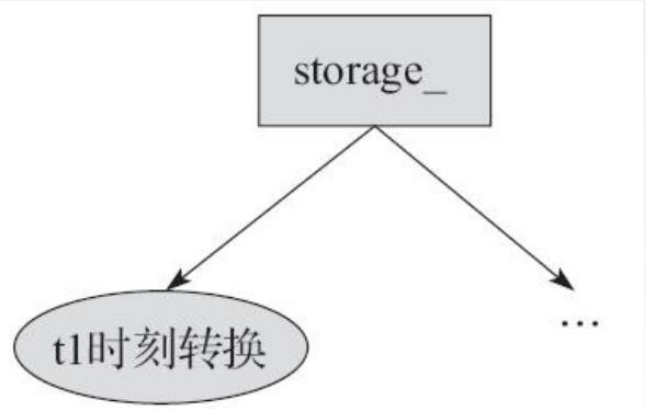
1. 与时刻相关联的坐标系转换存储管理

(2) 成员变量storage_访问

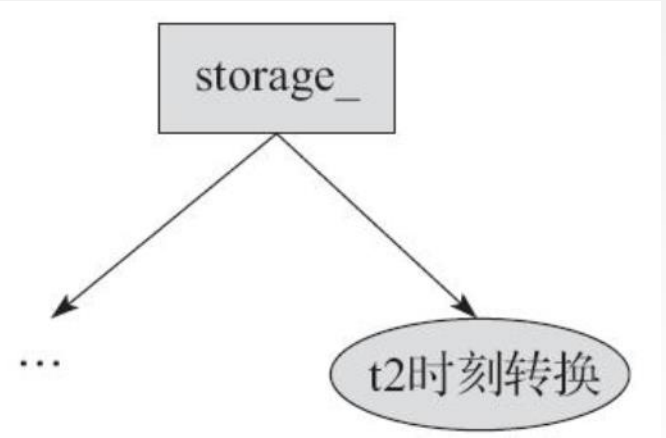
在对storage_的访问过程中，会遇到这样一个问题：需要知道在t时刻两个坐标系之间的转化数据。tf会以时间为索引，从storage_中获取所需要的数据。这时可能会出现如下的5种情形，分别如图所示（图中的省略号表示还有若干个节点）。



情景1



情景2

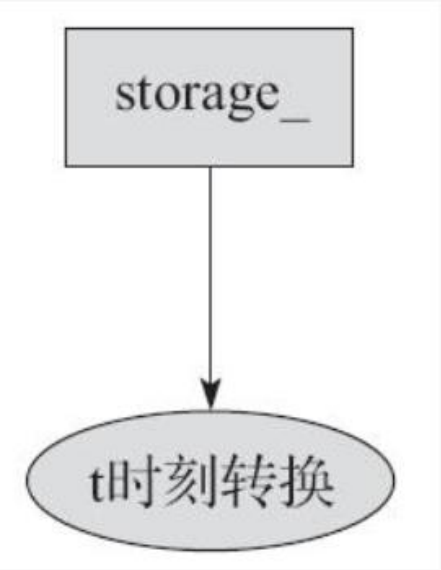


情景3

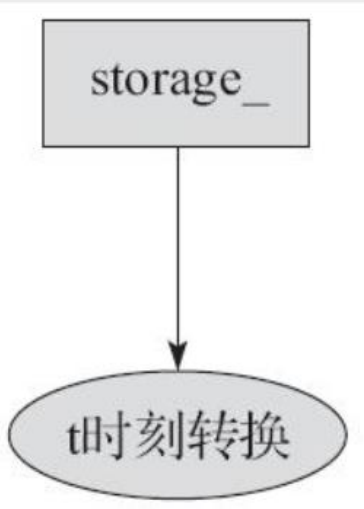
ROS TF原理分析——关键模块的实现

1. 与时刻相关联的坐标系转换存储管理

(2) 成员变量storage_访问



情景4



情景5

ROS TF原理分析——关键模块的实现

1. 与时刻相关联的坐标系转换存储管理

(2) 成员变量storage_访问

这5种情形的特点描述和处理方式分别如下。

情形1：存储结构当中有在t时刻前后两个时刻的两次数据转换，这种情况下，Time_Cache会调用函数interpolate进行插值计算，得到在近似t时刻的转换数据。其中interpolate函数是利用两个转化中的旋转四元式和平移向量分别进行球面插值和三次插值得到近似的数据。此处，前文提到的用四元式表示旋转的必要性和优点也得到了体现。

情形2：存储结构中最早的变换数据都比给定的时刻t要晚，会返回Time_Cache类中提到的第三种异常。

情形3：存储结构中最晚的变换数据都比给定的时刻t要早，会返回Time_Cache类中提到的第二种异常。

情形4：存储结构中刚好存在与t时刻对应的转换数据，直接获取数据即可。

情形5：存储结构中只存在一个时刻对应的转换数据，并且不是t时刻，由于只有一个数据，无法进行插值，因此会返回Time_Cache类中提到的第一种异常。

1. 与时刻相关联的坐标系转换存储管理

(3) 成员变量新数据的插入

在storage_中插入新数据时，Time_Cache会根据给定插入转换数据所对应的时刻，插入到storage_相应的位置当中。在插入的流程完成之后，还需要对整个storage_进行遍历，将其中转换数据对应的时刻超过系统最长存储时间的存储数据予以删除，以节省存储空间和提升后续的查找效率。

ROS TF原理分析——关键模块的实现

2坐标系树形结构处理

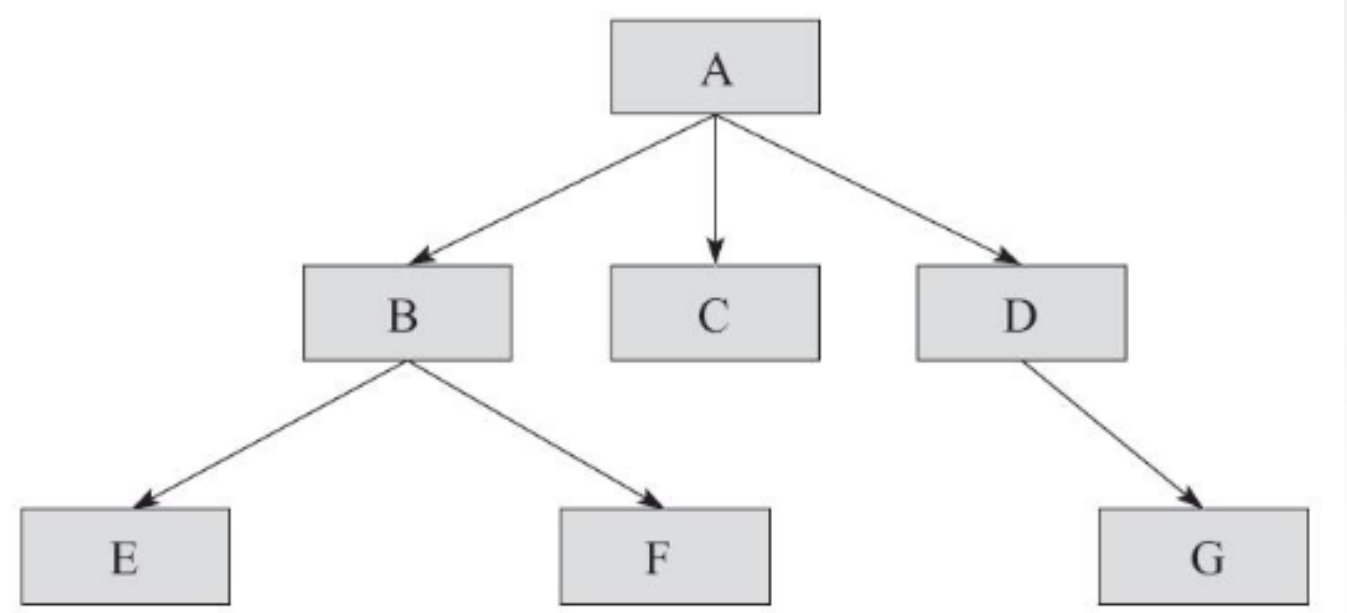
(1)坐标系树形结构的形成

前文的论述中已经提到过，tf中坐标系之间的转换关系用TransformStorage来存储，在每一个TransformStorage对象中都对应着两个坐标系，一个是转换中的初始坐标系，定义为父坐标系；另外一个是转换之后得到的坐标系，定义为子坐标系。由于tf限定了每个坐标系只有唯一的来源，从最初的自然坐标系开始转换生成不同的坐标，这样就形成了一个树形结构的坐标系关系图。

ROS TF原理分析——关键模块的实现

2坐标系树形结构处理

事实上在内存中，并没有真正存在一个这样的结构，是通过分析和整合各个坐标系之间的关系而得到的一个树形的结构图，其中一个例子的具体结构如图所示（每个大写字母均表示一个坐标系）。



2坐标系树形结构处理

(2)坐标系树遍历

上文中已经提到过，在系统内存中并不存在一块存放坐标系树的空间。事实上，在tf系统当中是利用每个坐标系对应的Time Cache对象的成员变量storage_来获取坐标系的父坐标系的，此时得到的父坐标系又可以找到与它对应的父坐标系，以此类推，可以实现从一个节点到根节点的访问。再从另外一个底端的坐标系开始访问，也可以访问到树的根节点，以此类推，可以完成整棵坐标系树的遍历。

2坐标系树形结构处理

(2)坐标系树遍历

上文中已经提到过，在系统内存中并不存在一块存放坐标系树的空间。事实上，在tf系统当中是利用每个坐标系对应的Time Cache对象的成员变量storage_来获取坐标系的父坐标系的，此时得到的父坐标系又可以找到与它对应的父坐标系，以此类推，可以实现从一个节点到根节点的访问。再从另外一个底端的坐标系开始访问，也可以访问到树的根节点，以此类推，可以完成整棵坐标系树的遍历。

2坐标系树形结构处理

(3)坐标系转换关系查找

假设有两个坐标系X（source_frame）和Y（target_frame），需要计算从X到Y的坐标系转换数据，则X坐标系和Y坐标系在坐标系树形结构中可能会出现以下几种情形。

情形1： X和Y在同一个坐标系，它们之间的转换关系非常明确，不需再对树进行访问，两者之间的转换关系就是X=Y。

情形2： X和Y在坐标系树的同一棵子树上，并且X是Y的父亲节点或祖先节点，如图1或图2中的A、B之间的关系。

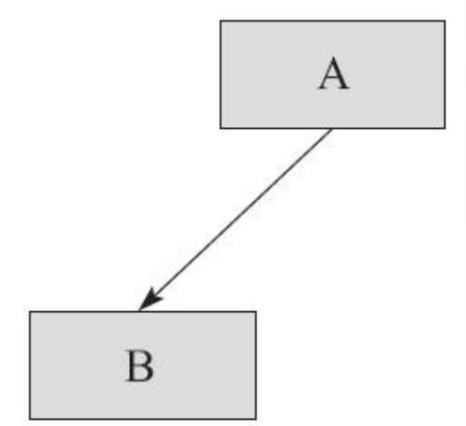


图1

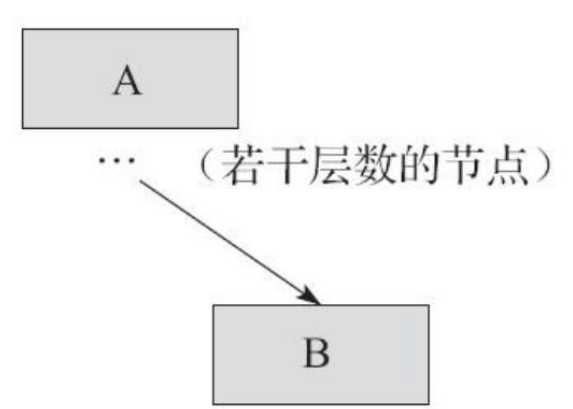


图2

ROS TF原理分析——关键模块的实现

2坐标系树形结构处理

(3)坐标系转换关系查找

情形3： X和Y在坐标系树的同一棵子树上，并且Y是X的父亲节点或祖先节点，如图1或图2中B、A之间的关系。

情形4： X和Y不在坐标系树的同一棵子树上，它们有共同的祖先节点或父亲节点，如图3或图4中X和Y的关系。

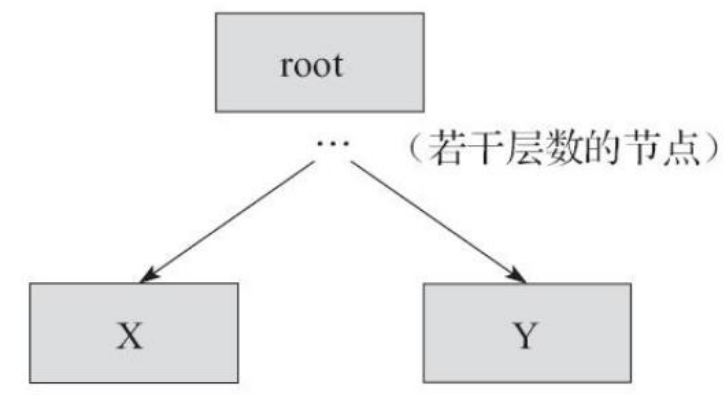


图3

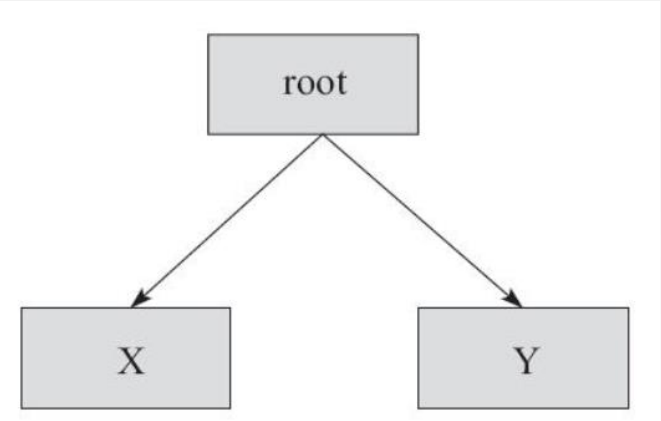


图4

情形5： X和Y不在同一棵坐标系树上，f会报告ConnectivityException异常。

2坐标系树形结构处理

(3)坐标系转换关系查找

通过下面的步骤可以解决查找两个坐标系之间转换数据的问题，具体步骤如下。

1)判断X和Y是否相等，如果相等则直接终止程序，正好对应情形1。否则，转到步骤2)执行。

2)从Y坐标系开始向上依次访问其父亲节点直到根节点为止，并迭代计算Y与当前访问坐标之间的转换关系数据，如果在访问过程当中存在一个坐标系为X,则访问立即终止，并且计算出从Y到X的坐标系转换数据，设为Tyx。如果到根节点还未能终止，则记录下这个根节点和坐标系转换数据，坐标系转换数据设为TR1,根节点设为R1执行步骤3)。

ROS TF原理分析——关键模块的实现

2坐标系树形结构处理

(3)坐标系转换关系查找

3)从X坐标系开始向上，依次访问其父亲节点直到根节点为止，并迭代计算X与当前访问坐标系之间的转换关系数据，如果在访问过程当中存在一个坐标系为Y,则访问立即终止，并且计算出从X到Y的坐标系转换数据，设为Ty。如果到根节点还未能终止，则记录下这个根节点坐标系转换数据,坐标系转换数据设为TXR2,根节点设为R2,执行步骤4)。

4)如果R1等于R2,则说明X和Y在同一颗坐标系树上，此时，计算坐标系变换T*xy=TxR2*（TyR1的逆），正常终止。如果不等于则表示X和Y不在同一棵坐标系树上，返回异常Connectivity-Exception。

通过上述三个步骤的执行，可以看出步骤2)的终止正好对应着情形2，此时可以得出结论：X到Y的坐标系转化数据为T的逆；步骤3)的终止正好对应情形3，此时可以得出结论：X到Y的坐标系转化数据为T;步骤4)的正常终止正好对应情形4，此时可以得出结论：X到Y的坐标系转化数据为T*。步骤4)报出的异常则正好对应情形5。

3.Transformer主要函数的具体实现

(1)添加新坐标系转换关系

在添加新的坐标转换关系时，其必要前提是坐标转换关系中的父坐标系已经存在于系统结构中。在添加新的坐标系转换关系时，只需要利用转换关系中的子坐标系的名称，在系统中构建一个Cache_Time的对象，同时把两个坐标系之间的转换数据和对应的时刻以TransformStorage的结构插入到Cache_Time对象的成员变量storage_当中即可，这就完成了新的坐标系转换的加入过程。当然，在给子坐标系构建Cache_Time对象时，需要判断它与父亲坐标系是否相同，为真则报出相应的错误。

3. Transformer主要函数的具体实现

(2) 源坐标系和目标坐标系之间的转换数据查找

上文关于Transformer类的介绍中已经提到，寻找源坐标系和目标坐标系之间的转换数据，分为两种情形，一种是含有一个固定不变的中间坐标系，另外一种是直接计算两个坐标系之间的转换数据。前一种情形可以转换为计算源坐标系——中间坐标系——目标坐标系转换关系的数据相乘，相当于两次第二种情形的计算，因此，这里仅讲述两个坐标系之间有直接转换关系的情形。在已知两个坐标系名称的前提之下，只需要利用坐标系转换关系查找中的方法处理即可，得出相应的结果后，一般会以消息的形式将得到的结果传送给其他模块使用。

3. Transformer主要函数的具体实现

(3) 坐标转换关系周期性检测

Transformer类中定义了一个模块，它的功能是在一定时间段内每隔一小段时间就检测一次两个坐标系之间是否存在转换关系，该模块平时处于阻塞状态，检测到转换关系之后，模块开始运行，这样做的目的是为了防止后面的程序出现异常症状。与此同时，如果在规定的时间段内均没有检测到转换关系，则会报出相应的异常并且终止程序的继续执行。检测转换关系的过程正是不断调用上文中提到的方法，查看是否存在转换关系。

3. Transformer主要函数的具体实现

(4) 源坐标系和目标坐标系转换关系判断

同样，在Transformer类的介绍中已经提到过，判断源坐标系和目标坐标系之间是否存在转换关系与寻找源坐标系和目标坐标系之间的转化换数据一样，都包含两种不同的类型，同理，这里只须分别判断源坐标系和中间坐标系是否存在转换关系及中间坐标系及目标坐标系是否存在转换关系即可，将两者的结果做与运算就可以得到最终的结果。两个坐标系之间直接的坐标转换关系只需调用上文中步骤2)中的方法，如果可以得到一个没有异常的结果，则表明存在这个转换，判断的结果就为真，反之亦然。

4. 时间参数及其作用

(1)时间参数的含义

在tf功能包的设计理念中，时间参数指的是具体的时刻。为了处理在时间先后关系上不同坐标系之间的关系，tf定义了坐标系转换与时刻之间一一对应的关系，从上文的介绍中可以看出，每个齐次变化都对应了一个时间成员变量。时间参数的定义，是tf能够实现随时间变化追踪坐标系变换的基本条件。

(2)时间参数与坐标系转换的关系

时间参数和坐标系存在多对多的关系。同一个时刻可以对应多个坐标系之间的转换关系，同时，两个坐标系在不同的时刻也可以对应多种不同的转换关系。在现实的应用当中，tf主要是利用当前时刻两个坐标系之间的转换关系来计算相关的数据。

4. 时间参数及其作用

(3)时间参数在具体应用实现中的作用

时间参数一般是在定义一个具体的齐次变化时随之确定的。它的应用主要有以下几种情形。

- 给出具体时刻，查找此时刻两个坐标系之间的转换关系。

- 在查找转换关系时，若没有找到准确对应时刻的转换，则需要利用时间参数进行三次插值和球面插值，以得到近似的数据。同时，也可以根据时间关系给出可能出现的异常。

- 当给定的时间是ROS当前时刻时，tf需要调用专门的函数来处理时间关系。

Part 4

ROS任务调度

ROS任务调度接口设计

基于ROS平台的应用系统，通过ROS的“服务”可以实现任何请求响应模型的交互，即客户端发送任务请求到服务器，服务器执行相应任务，将任务执行结果返回客户端。但是某些时候，客户端请求的任务需要长时间执行，客户端希望能够周期性的获取任务的执行状态，或者能够在任务执行过程时中断任务的执行，因此，ROS提供了抢占式的任务调度接口actionlib，该接口不但可以调度任务的执行，而且具备中断任务、任务状态跟踪与周期性状态反馈、执行过程中间结果反馈的能力。

这里的“任务”指的就是用户自定义的机器人任务，如智能机器人自主识别及抓取物体、机器人依据规划路径自主行走等，具体任务的程序由用户来实现，actionlib接口只是提供了相应的调度接口。

ROS任务调度接口设计

在actionlib中，客户端发送请求到服务器端，服务器端根据相应的任务调度策略进行任务调度，被选择的任务将会抢占当前任务的执行权。同时，客户端还可以发送取消请求，服务器端依照一定的原则选择取消一个或多个任务的执行。

actionlib接口中提供了Simple调度策略，具体策略为：服务器端一次只选择一个任务进行调度执行，后来的任务总是抢占当前任务的执行，不管当前任务是否执行完毕。

ROS任务调度接口设计

action的设计与编译

action用于定义任务，任务定义包括**目标**(Goal)、**任务执行过程状态反馈**（Feedback）和**结果**（Result）等。action是一种与平台、编程语言无关的接口定义语言。图是一个简单的action定义示例。

```
#goal definition
int32 order
time endtime
---
#result definition
int32[] sequence
---
#feedback
int32[] sequence
```

ROS任务调度接口设计

action的设计与编译

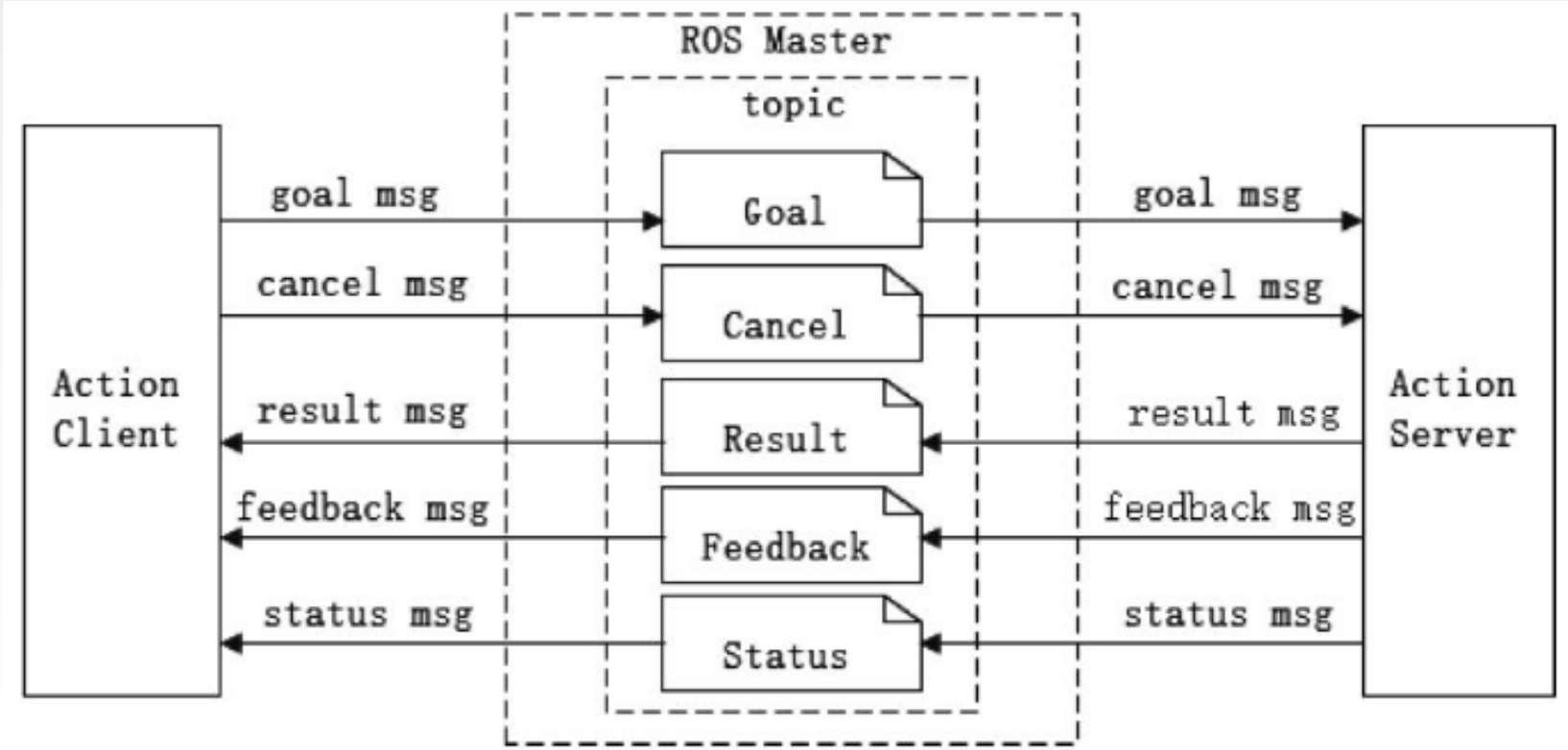
action的本质是消息。action文件依次定义了 goal、 result、feedback，定义顺序不能交换。action通过ROS中的catkin或rosbuild两种编译系统，可以生成多种语言（C++、Python、Java),以及屏蔽平台数据类型差异性的类、结构体及消息。编译action将会自动产生7个结构体，分别为Action、ActionGoal 、 ActionFeedback 、 ActionResult 、 Goal 、Feedback、Result结构体。图是编译action生成的消息。

```
testGoal.msg
#goal definition
int32 order
time endtime

testResult.msg
#result definition
int32[] sequence

testFeedback.msg
#feedback
int32[] sequence
```

基于主题的ActionClient与ActionServer的交互设计

1.基本架构

actionlib接口中定义了ActionClient与ActionServer两种角色，分别代表任务请求的客户端和任务调度的服务器端。下图是ActionClient与ActionServer的交互基本框架。



基于主题的ActionClient与ActionServer的交互设计

ActionClient与ActionServer的交互采用了“发布/订阅”模型，即基于主题的异步数据流通信模型，具体实现中依赖XMLRPC进行远程过程调用，XMLRPC的底层实现采用的是select异步网络模型。两者的交互原理具体如下。

1）ActionClient发布Goal和Cancel主题，并向主题中发送goal消息和cancel请求消息。

2）ActionClient订阅主题Result、Feedback及Status，并监听这三个主题上的消息。

3）ActionServer订阅主题Goal及Cancel，并监听这两个主题上的消息。

4）ActionServer发布主题Result、Feedback、Status，并分别向这三个主题发送result消息、周期性的feedback消息及任务执行状态消息。

基于主题的ActionClient与ActionServer的交互设计

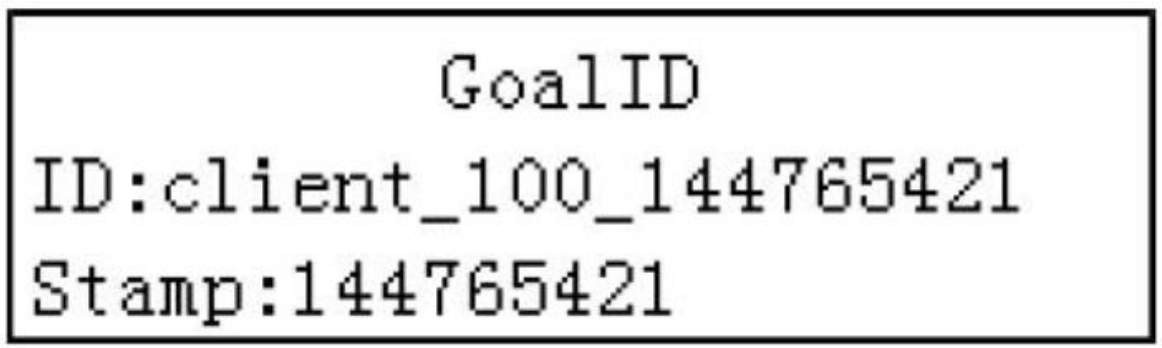
2 任务区分设计

当多个ActionClient在一定时间内同时发送Goal消息时，ActionServer需要区分每一个Goal属于哪一个ActionClient，因此必须保证每一个Goal都有唯一的标识，Action-Server才能将执行结果发送给相应的客户端。

actionlib接口设计了专用于区分目标的GoalID类。GoalID类中有ID和Stamp两个成员，ID是唯一标识每一个Goal的字符串，Stamp代表每一个Goal发送的时间戳。GoalID类为ActionServer与ActionClient的交互提供了一个稳健的方式，来保证两者之间的任务请求和执行结果的一一对应关系。

基于主题的ActionClient与ActionServer的交互设计

GoalID的ID由节点名、节点计数及时间戳构成。节点名本身具有全局唯一性；节点计数采用锁机制，以保证全局唯一；再结合不同的时间戳，来保证每一个Goal均有唯一标识。图是简单的GoalID示例。



基于主题的ActionClient与ActionServer的交互设计

3. 主题及主题消息设计

ActionClient与ActionServer的交互涉及以下5个主题。

- **Goal主题：**使用编译action产生的ActionGoal消息，该消息绑定有Goal消息及GoalID。
- **Cancel主题：**使用GoalID消息，客户端根据GoalID消息取消相应的目标。
- **Result主题：**使用编译action产生的ActionResult类型消息，该主题用来传递任务执行结果。
- **Feedback主题：**使用编译action产生的ActionFeedback消息，服务器可以在目标执行中周期性地发送ActionFeedback消息到客户端，客户端根据ActionFeedback消息跟踪任务执行的中间过程。
- **Status主题：**使用GoalStatusArray类型消息，服务器将任务执行状态通过该主题发送给客户端，客户端完成相关任务的状态更新。

ActionClient与ActionServer的交互过程

ActionClient与ActionServer在交互之前，需要事先定义action文件，通过catkin和rosbuild编译系统自动产生Goal、Feedback、Result消息及对应的Goal、Feedback、Result结构体，这些消息及结构体信息对ActionClient和 ActionServer均是可见的，即ActionClient和ActionServer共享这些消息类的结构、成员等信息。具体交互过程如下。

1）ActionClient创建Goal消息并初始化，将Goal消息发送到主题Goal上。

2）节点管理器（即ROS Master）通知ActionServer接收相应消息。

3）ActionServer将Goal加入到任务列表，经过一定的策略，一次选择列表中的一个任务进行执行，执行开发者提供的处理函数。

4）ActionServer将执行结果发布到Result主题中，ActionClient接收任务的执行结果。

ActionServer在调度任务的过程中会更新任务状态并周期性地反馈给ActionClient,而开发者提供的处理函数可以选择发送中间计算结果给ActionClient，需要说明的是，ActionServer并不会自己发送中间结果，因为ActionServer并不清楚开发者提供的处理函数过程。

Part 5

ROS有限状态机的SMACH

有限状态机的基本原理

1有限状态机的定义

有限状态机是由有限的状态及其相互之间的转换构成。有限状态机在任何时候都只能处于某一给定的状态。当接收到输入事件时，状态机产生相应的输出，同时也可能伴随着状态的转换。

有限状态机M可以形式化表示为一个五元组：M=(Q, Σ, δ,q0,F),其中各项含义如下。

- Q是一个有限的状态集合。
- Σ表示系统能接收的所有事件的集合。
- δ是状态转移函数。
- q0∈Q是系统的特殊状态，一般是系统启动时的初始状态。
- F∈Q是终止状态的集合。

有限状态机描述了某个特定对象的所有状态及其转换的过程，对象的状态转换由事件驱动，对象通过对事件接收的响应来改变当前状态，状态转换图是有限状态机的确切描述方法。

有限状态机的基本原理

2 有限状态机的要素

有限状态机包含4个要素，具体如下。

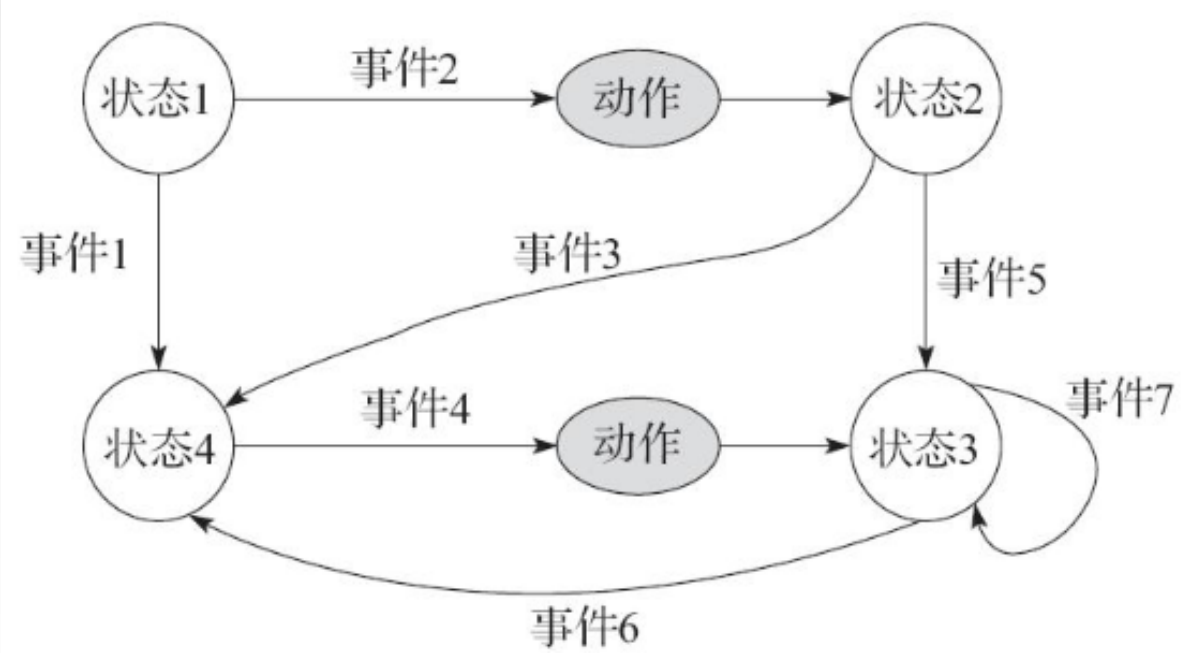
- 现态：指状态机当前所处的状态。
- 条件：又被称为“事件”。当某一条件满足时，将触发相应的事件，或者执行相应状态的迁移。
- 动作：条件满足后执行的动作。动作执行完毕后，可以迁移到新的状态，或者仍旧保持原状态。条件满足时，也可以不执行任何动作，直接迁移到新状态。
- 次态：条件满足后要转换到的状态。“次态”是相对于“现态”而言的，“次态”一旦被激活，就会成为新的“现态”。

如果我们做进一步的归纳，把“现态”和“次态”统一起来，而忽略“动作”的话，那么状态机就有两个基本且关键的要素：状态和迁移条件。

有限状态机的基本原理

3 有限状态机的状态迁移

动作迁移图是一种描述系统状态、状态转换关系的形象化图形表示方式，如图所示。



有限状态机的基本原理

4. 有限状态机的应用条件

有限状态机的优点在于简单易用，状态间的关系简明直观。一般情况下，可以应用有限状态机的前提包括如下几点。

- 被模型化的系统且具有有限个状态。
- 系统在一定状态下的行为响应是唯一的。
- 系统在任何时间段内总会停留在某一状态下。
- 系统状态迁移的条件是有限的。
- 迁移是系统对事件的反应。
- 迁移所用的时间近似为零。

有限状态机的基本原理

5. 智能机器人的有限状态机应用

机器人是以微控制器为控制核心的，微控制器内嵌程序的运行决定了机器人的动作或行为。如何为不同应用的机器人设计控制程序一直是困扰着设计人员的难题之一。在明确需求的基础上，将工作任务细分，确定各个任务之间的内在联系，就可以用有限状态机模型来指导不同类型机器人控制程序的设计。有限状态机是一种记录给定时刻状态的数学模型，根据输入而改变其状态或引发一个动作。应用有限状态机或状态转换方法实现机器人控制程序设计，可以理清控制思路，简化设计过程。

ROS作为面向机器人的操作系统，控制程序设计支持尤为重要。为了提高代码复用，适应不同应用场景的机器人控制要求，有限状态机SMACH应运而生。SMACH提供了机器人任务控制所需的不同状态和状态机容器，使得设计人员在面对特定的机器人控制程序设计时，能够组合不同类型的状态机，不但可提高代码的复用性，同时设计也更加简单高效。

有限状态机的基本原理

5. 智能机器人的有限状态机应用

在通用有限状态机模型中，事件是状态转移的触发因素，某一事件的出现使得相应的动作被执行，或者发生状态转移，状态机从当前状态转移到下一个状态。状态反映的是系统从开始到现在时刻的输入变化。转移是两个状态之间的关系，它由某个事件触发，然后执行特定的操作或评估并导致特定的结束状态。动作是在给定时刻要进行的活动描述，是系统此刻需要执行的活动。而SMACH设计的初衷是用于描述具有层次结构的机器人高层复杂任务。机器人的复杂任务可以被逐层分解成可执行的子任务，然后用SMACH的状态明确定义这些可执行的子任务，状态之间通过状态转移条件关联，这样就可以用状态机的形式明确表示一个复杂的机器人任务。因此可以看出，SMACH中的状态是对机器人任务执行过程中某一子任务的描述，而状态转移则表示了当前状态到下一状态的对应关系及转移条件，因此当前状态执行结束的情形决定了状态转移的方向。

SMACH概述

1.SMACH简介

SMACH是StateMachine的缩写。SMACH是利用独立于ROS的Python库构建的层次状态机库，用于快速创建健壮、可维护、模块化的机器人行为代码。它具有任务级执行和协调的功能，并且提供了多种状态和状态机容器，而容器又是特殊的状态，因此可以作为其他容器中的状态。由于某些复杂的结构化任务可以明确定义而且能够利用状态机清晰地描述，所以SMACH可以用于实现任务级的机器人控制，而无结构化的任务可以在不同系统的更高层处理。

由于Python语言对于智能机器人编程具有优势，因此ROS采用Python应用编程接口开发层次任务的状态机。SMACH作为ROS的核心之一,不仅能够用来建立层次化的状态机,而且可以建立任何其他遵守接口约定的任务状态容器。尽管SMACH是独立于ROS的库，但是相当多的内容已经写入到smach_ros包中，用于实现与ROS系统通信，比如主题、服务和动作库等。

SMACH库的核心是精简的，并且增加了日志和实用函数，提供了状态和容器接口。

SMACH概述

2. SMACH特点

SMACH有利于控制机器人复杂动作的具体实现。在SMACH中，所有可能的状态和状态转移都能够明确定义。SMACH具有以下特点。

- 快速原型：利用基于Python的状态机语法，能够快速构建可执行的状态机。
- 复杂状态机：SMACH支持设计、维护和调试大型的复杂的层次状态机。
- 自省：SMACH可以创建状态机的自省程序，包括状态转移、用户数据等要素的初始化。

由于SMACH旨在提供任务级的状态机描述，因此其并不适用于无结构性的任务，以及需要较高执行效率的底层系统。

3. SMACH设计的相关概念

(1)输出语义

对于状态机容器，其中的状态接口都是通过状态输出来定义的。状态输出类型是状态的一个属性，并且必须在执行前明确声明。不同类型的容器中，不同状态输出会触发不同的事件，当状态输出确定之后，从状态的视角来看后续的事件是不相关的，所以对于给定的状态而言，输出可以认为是“局部”数据。

状态能够提供“成功”、“失败”、“抢占”等类型的输出，对于状态而言，这些输出是状态之间的交互方式；对于状态机而言，这些输出可能和其他的状态相联系，从而导致状态间转移；对于容器而言，输出会有不同的处理方式，可能仅仅是状态和容器之间的普通接口。

3. SMACH设计的相关概念

(2) 用户数据

容器包含对应的平面数据库，用于不同状态之间的数据传递，可以存储状态生成的某些中间结果或是传感器返回的传感数据。数据库支持在状态执行过程中保存修改的数据，而且这些数据可以被其他任务或进程利用。应用可以通过状态的输入索引和输出索引访问用户数据，输入索引和输出索引是状态的属性，并且必须在执行前被声明，以防止在调试或设计复杂状态机时出现错误。

用户数据通常是和单一的容器相关联的，容器均会有对应的用户数据结构，从而实现容器所包含子状态的数据访问、数据共享及状态间通信等。用户数据也可以通过接口函数传递给其他关联的容器。

3. SMACH设计的相关概念

(3)抢占

SMACH也提供了抢占的概念，状态利用接口来处理容器，以及其所含状态的抢占请求。不同的容器类型对应不同的行为来回应抢占请求。因此，系统不仅能够对某一结束操作的用户信号给出明确响应，同时也应能够通过更高层的程序化的操作取消某些动作。

抢占是SMACH中的一个重要概念，在状态的执行过程中要不断检查状态抢占标志位。尤其是在并发的容器中，抢占显得尤为重要，当某个并发的分支终结时，也就意味着并发容器执行的结束，那么当前正在执行的其他分支状态都要被抢占。

3. SMACH设计的相关概念

(4) 自省

SMACH容器提供了调试接口，支持开发者通过SMACH树形结构来设置所有的初始状态和用户数据。此外，SMACH窗口工具实现了操作的可视化。可视化工具包含每个容器初始的状态标签，以及每层的用户数据。自省操作通过客户端和服务器模式实现，通过订阅相关主题，实现客户端和服务器的通信，以此来实现容器的用户数据和状态的初始化操作。

SMACH状态描述

1. 状态(State)及其属性

(1)设计思想

State是SMACH的状态基类。所有状态和容器都是从State继承而来的，而且在状态机的设计和执行过程中均是以状态State为单位来处理的。State定义了状态的接口，状态通过两种方式与容器进行交互，第一种方式是通过状态输出，第二种方式是在运行时读出或写入用户数据。这些交互方式必须在状态执行前声明，并且在构建时进行一致性检查。状态实现了阻塞的execute(函数，只有在状态返回给定输出时，才会终止状态的执行。

(2)基本属性

状态内置成员函数主要包括_init_（）和execute（）。其中_init_（）函数主要用于状态的初始化，包括参数存储，输出声明以及状态转移。execute（）方法在状态执行时调用，可以根据具体需要实现不同的功能。

SMACH状态描述

1. 状态(State)及其属性

(3) 重要属性

抢占是State中的重要概念，定义一个状态变量标识状态的抢占状况，通过内置的成员函数实现对抢占标识的访问。表中所列的函数通过维护特殊的标志位来指明状态的抢占情况。

函数名称	实现方式
request_preempt	设置抢占标志位 preempt_requested 为 True，抢占该状态
service_preempt	设置抢占标志位 preempt_requested 为 False，表明抢占生效，但执行该函数后，状态可再次被抢占
recall_preempt	设置抢占标志位 preempt_requested 为 False，解除抢占
preempt_requested	判断当前状态是否被抢占，如果被抢占，则函数返回 True

通过以上四个函数，每个状态都可以维护自己的抢占标志位，一旦状态被抢占，它并不是马上停止执行，而是在执行到抢占检测点时，才会判断标志位，状态执行才会结束。

SMACH状态描述

2. 状态(State)分类

State可被细分为以下6种状态类别，具体如下。

(1) 通用状态（Generic State）

该状态是从State状态继承而来的，所以不必定义基本状态接口。Generic State并不属于SMACH提供的基本状态，之所以在这里提出这个概念，是因为它在做数值运算和其他单一的非功能性调用时非常重要，不必再单独重写代码来实现与状态机的交互，仅仅重载它的execute（就可以实现相关功能，这样做不仅提高了代码的复用，也增强了代码的清晰度。设计完善的状态不会在创建时被阻塞，如果需要在创建时等待其他的状态完成，可以用多线程的方法实现该状态。

SMACH状态描述

2. 状态(State)分类

(2)回调状态(CBState)

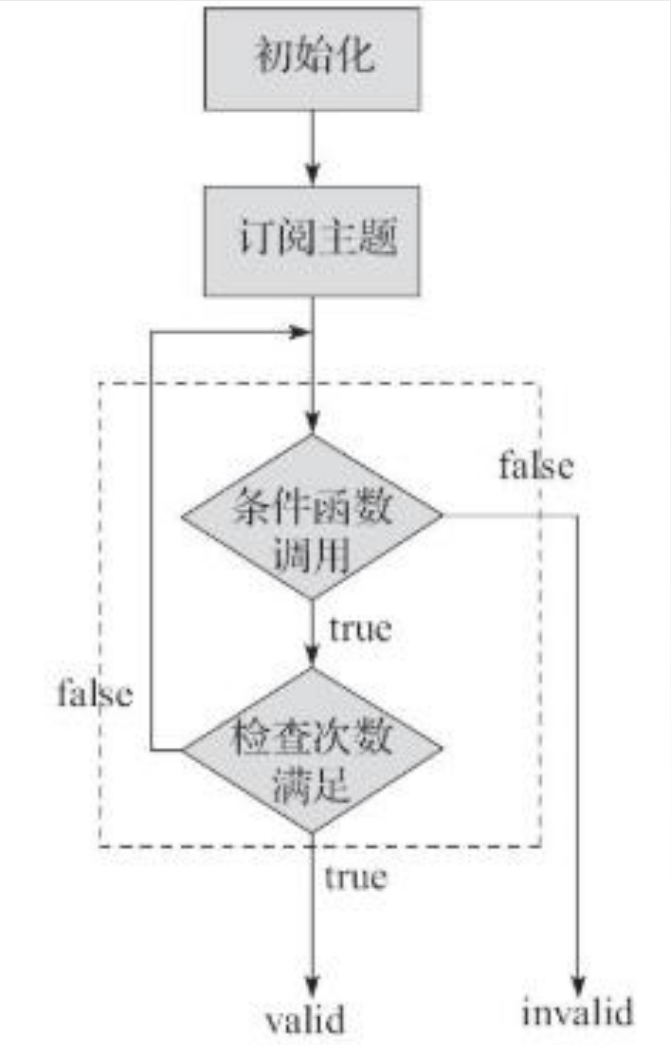
顾名思义，回调状态主要是把回调函数封装成状态，其继承了状态的基本属性。执行该状态时仅执行单一的函数调用即可，可以通过smach.cb_interface拓展接口的功能，其作用是在状态输出之后拓展状态的输出语义，也就是使得状态能够具有附加的属性，从这个意义上说，该接口就是回调函数，不过这个回调函数需要在构建时定义。CBState调用时至少要传递一个名为用户数据（userdata）的参数，附加的参数可以在构建时给出，这些参数将在状态执行时传递给回调函数。

SMACH状态描述

2. 状态(State)分类

(3) 监控状态 (MonitorState)

该状态主要是通过订阅给定的主题来实现对某一主题的监控。在每次接收到主题时执行用户自定义的回调函数。该状态通过条件函数检测给定的主题。通过订阅相关主题，利用python的线程概念使其阻塞，阻塞时不允许抢占出现。当订阅的主题消息出现时，判断函数调用和检查次数等条件，条件成立时则唤醒线程，取消订阅该主题，具体流程如图所示。

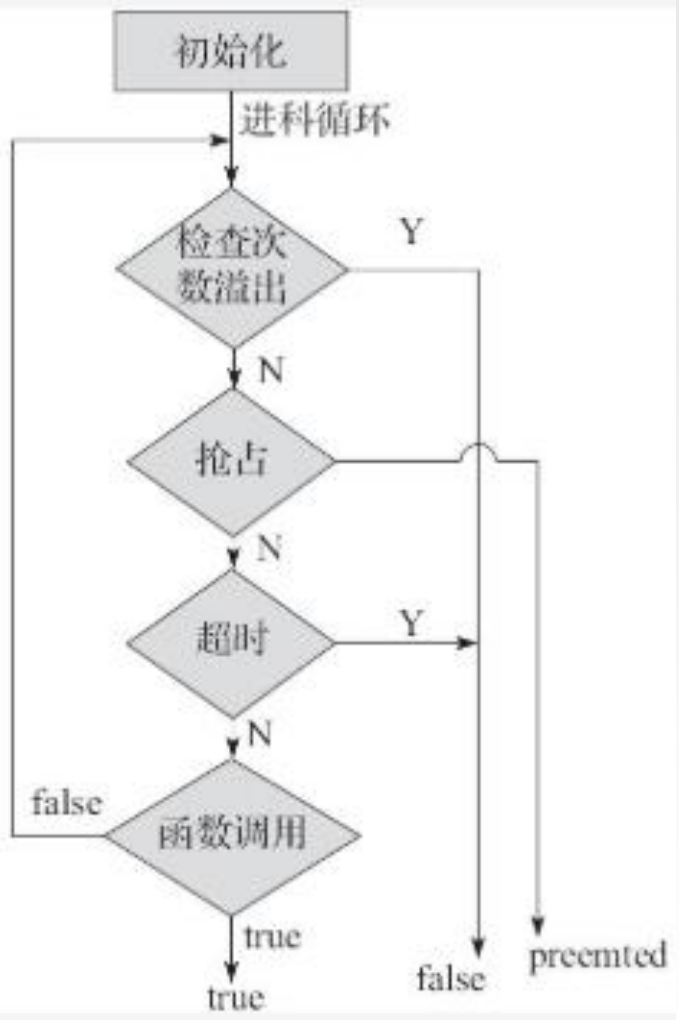


SMACH状态描述

2. 状态(State)分类

(4) 条件状态(ConditionState)

该状态主要通过函数调用实现对某个条件的多次检查。当参数中的max checks大于1时，状态就循环检测某个条件。其在循环过程中可以被抢占，一旦状态被抢占，就终止状态执行。当条件为False,或者检查次数超过最大检查次数时，则表明某个条件未能满足，状态返回False,具体流程如图所示。



SMACH状态描述

2. 状态(State)分类

(5) 简单动作状态 (SimpleActionState)

该状态的创建主要是封装actionlib中的动作，虽然可以从一般的程序中调用actionlib中的动作，但是需要编写比较明确的动作调用代码，由于SMACH已经明确实现了动作状态的调用，因此实现动作调用更加简单，而且更节省代码。SMACH提供了作为actionlib动作代理服务器的状态类，实现了某个节点的具体动作。在传递参数时，只需要提供具体动作目标参数，状态执行时就会通过传递参数来生成具体的动作目标，通过SendGoal发送目标消息，对应的节点就会通过目标消息实现该动作。在实例化时需要包含主题名称、动作类型和生成目标的参数，简单动作状态的输出是成功、失败或抢占。

SMACH状态描述

2. 状态(State)分类

(6) 服务状态 (ServiceState)

该状态用来封装节点提供的服务。虽然一般可以从给定程序中调用服务，但需要明确服务调用的过程，这一点对于不太清楚服务调用的编程人员而言会比较麻烦。因此，SMACH通过把服务封装成状态，支持服务调用，从而使得服务调用变得简单。SMACH提供了服务状态，同样在该状态创建时，需要request和result参数，其分别代表服务的请求消息和结果消息，在状态执行时会根据传递的请求参数生成请求消息，在状态完成时会接收结果消息并存储在用户数据中。在实例化该状态时,需要具体给出服务名称、服务类型和生成请求的参数。服务器的状态输出可以是成功、失败或抢占。