

《ROS机器人开发技术》

**课程名称:ROS机器人开发技术**

**教师姓名:孙乐璇**

**提交时间:2018年7月x日**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程信息 | 课程名称 | | 名称 | | 章数 | 节数 | | 课程类型 |
| 《ROS机器人开发技术》 | | ROS通信架构 | | 三 | 一 | | 授课(√ )  实训( ) |
| 教师 |  | | | 时长 |  | |
| 参考文献 | 1. ROS机器人程序设计（原书第二版） [西班牙]恩里克.费尔南德斯等著 ，刘锦涛 等译 | | | | | | | |
| 教学  目的  要求 | 掌握service、action通信机制，熟练操作rosservice、rosparam、rossrv等操作指令 | | | | | | | |
| 教学  重点  难点 | 重难点 | | | PPT页面 | | | 时间分配 | |
| 重点 | 1. Service通信方式 | | 3~6页 | | | 5分钟 | |
| 1. Service数据格式.srv文件 | | 7~12页 | | | 7分钟 | |
| 1. Parameter Server | | 13~18页 | | | 15分钟 | |
| 1. Action通信方式 | | 19~22页 | | | 10分钟 | |
| 1. Action数据格式.action文件 | | 23~30页 | | | 6分钟 | |
| 难点 | 1. Service通信方式 | | 3~6页 | | | 5分钟 | |
| 1. Action通信方式 | | 19~22页 | | | 10分钟 | |
| 教学方法 | 本授课以课堂讲授为主，与课堂演示方式相结合 | | | | | | | |

|  |  |  |
| --- | --- | --- |
| 教学内容 | 操作演示 | 知识点 |
| **PPT第1页:**  大家好，我是孙乐璇，今天由我带领大家学习ROS通信架构的下篇，即有关service、action通信机制以及相关操作指令的内容 |  |  |
| **PPT第2页:**  我们的教学内容主要分成5个部分：  3.2.1 Service通信方式  3.2.2 Service数据格式.srv文件  3.2.3 Parameter Server  3.2.4 Action通信方式  3.2.5 Action数据格式.action文件 |  |  |
| **PPT第3页:**  首先是3.2.1部分，service通信方式的介绍，让同学们来了解这一部分的内容。上一章我们介绍了ROS的通信方式中的topic(主题)通信，我们知道topic是ROS中的一种单向的异步通信方式。然而有些时候单向的通信满足不了通信要求，比如当一些节点只是临时而非周期性的需要某些数据，如果用topic通信方式时就会消耗大量不必要的系统资源，造成系统的低效率高功耗。  这种情况下，就需要有另外一种请求-查询式的通信模型。这节我们来介绍ROS通信中的另一种通信方式——service(服务)。它是ROS中的同步通信方式。 |  |  |
| **PPT第4页:**  为了解决以上问题，service方式在通信模型上与topic做了区别。Service通信是双向的，它不仅可以发送消息，同时还会有反馈。所以service包括两部分，一部分是请求方（Clinet），另一部分是应答方/服务提供方（Server）。这时请求方（Client）就会发送一个request，要等待server处理，反馈回一个reply，这样通过类似“请求-应答”的机制完成整个服务通信。  这种通信方式的示意图如下：Node B是server（即应答方），它提供了一个服务的接口，叫做 /Service ，我们一般都会用string类型来指定service的名称。Node A向Node B发起了请求，经过处理后得到了反馈。Service是同步通信方式，也就是说，此时Node A发布请求后会在原地等待reply，直到Node B处理完了请求并且完成了reply，Node A才会继续执行。Node A等待过程中，是处于阻塞状态的成通信。这样的通信模型没有频繁的消息传递，没有冲突与高系统资源的占用，只有接受请求才执行服务，简单而且高效。 | **Node间可以通过request-reply方式通信：** |  |
| **PPT第5页：**  下面我们对比一下这两种最常用的通信方式，加深我们对两者的理解和认识，具体见下表，这个表中从通信方式，实现原理，通信模型，应用场景及举例等几方面介绍二者的区别。从这个表中，我们可以看出Topic是异步通信方式，Service是同步通信，而实现原理方面都是基于TCP/IP协议，通信模型方面，topic是多对多的，service是多对一的，topic应用于连续、高频的数据发布，service应用于偶尔调用的功能/具体的任务。 |  |  |
| **PPT第6页：**  下面列出了rosservice的相关命令：  使用$rosservice list可以列出当前所有活跃的service  $rosservice info service\_name命令显示某个service的属性信息  $rosservice call service\_name args命令可以调用某个service，并且传递相关的参数值  $rosservice type service\_name命令可以打印出服务类型  $rosservice uri能打印服务的uri  $rosservice find 可以按服务类型查找服务  $rosservice args 能打印服务参数 | **$rosservice list**    **$rosservice info /gazebo/delete\_light**    **$rossrv show gazebo\_msgs/DeleteLight** |  |
| **PPT第7页:**  下面我们来看这个srv，srv是什么呢？我们可以把srv和msg做对比，Service通信的数据格式就是srv，它定义在\*.srv文件中，类似于msg文件，是用来描述服务（service）数据类型的。所以srv与service的关系就好比msg与topic的关系。将两者放在一起类比，理解起来比较容易。 |  |  |
| **PPT第8页:**  下面我们来看一看srv文件格式的写法，它声明了一个服务，包括请求(request)和响应（reply）两部分。以人体关节检测为例，如下图所示，首先srv文件一定要放在srv文件目录下，其次文件的格式中间要用“---”隔开，“---”之上的部分是请求（request）要遵循的格式，下面是应答部分要遵循的格式，即“问-答”形式。正如例子中写的，在srv文件中，请求部分设置为bool型，当其为True时，我们可以开始进行检测，检测结束后，就回传HumanPose数组类型的数据，即应答部分的形式定义。定义为数组是因为我们检测到的可能不止一个人，而这里的HumanPose数组是我们自定义的，如下页ppt所示。 |  |  |
| **PPT第9页:**  HumanPose数组是定义在msg文件下的HumanPose.msg文件里，需要注意的是srv文件中的HumanPose必须是一个msg文件，因为srv只能嵌套msg。不能再去嵌套一个srv。如果嵌套的是srv文件，那就无法解析这个文件了。但如果嵌套的是msg文件，那么这个msg在srv中是所为一种自定义的数据类型存在的。这个HumanPose.msg中就可以定义一个人的骨骼信息了，其中有string类型的id，有int型定义的骨骼数目参数，还有我们自己定义的JointPos类型存放每一个关节的具体信息。  正如第二部分定义的那样，HumanPose.msg中嵌套了JointPos.msg文件，每一个关节信息的数据定义都放在JointPos.msg文件中。 |  |  |
| **PPT第10页:**  上述以DetectHUman.srv文件为例，该服务例子取自OpenNI的人体检测ROS软件包。它是用来查询当前深度摄像头中的人体姿态和关节数的。  再一次强调，.srv文件只能嵌套.msg文件，不能再嵌套.srv文件。 |  |  |
| PPT第11页: 定义完msg或srv文件后，还有一点需要注意的是，我们需要修改package.xml和CMakeList.txt中的内容，具体需要添加一些条目，正如图中所示，需要修改相关部分的内容。这些修改的内容就是指定了srv或msg文件在编译或运行时所需要的依赖，刚开始学，我们可以不去深究每一条代码所表示的含义，但我们需要知道无论是自定义了srv文件还是msg文件，都要修改这几部分内容。 |  |  |
| PPT第12页: 下面我们来介绍一下rossrv的一些相关命令：  $rossrv list 用来列出系统上所有srv  $rossrv show srv\_name 该命令用来显示某个srv内容  $rossrv md5 srv\_name 显示服务md5sum  $rossrv package package\_name 用来列出包中的服务  $rossrv packages 列出包含服务的包 |  |  |
| PPT第13页: 接下来我们将第三部分，即有关Parameter Server的内容。 |  |  |
| PPT第14页: 前文介绍了ROS中常见的两种通信方式——主题和服务，这节介绍另外一种通信方式——参数服务器（parameter server）。与前两种通信方式不同，参数服务器也可以说是特殊的“通信方式”。特殊点在于参数服务器是节点存储参数的地方、用于配置参数，全局共享参数。参数服务器使用互联网传输，在节点管理器中运行，实现整个通信过程。  参数服务器，作为ROS中另外一种数据传输方式，有别于topic和service，它更加的静态。参数服务器维护着一个数据字典，字典里存储着各种参数和配置。  何为字典，其实就是一个个的键值对，我们小时候学习语文的时候，常常都会有一本字典，当遇到不认识的字了我们可以查部首查到这个字，获取这个字的读音、意义等等，而这里的字典可以对比理解记忆。键值kay可以理解为语文里的“部首”这个概念，每一个key都是唯一的，参照下图：  每一个key不重复，且每一个key对应着一个value。也可以说字典就是一种映射关系，在实际的项目应用中，因为字典的这种静态的映射特点，我们往往将一些不常用到的参数和配置放入参数服务器里的字典里，这样对这些数据进行读写都将方便高效。  参数服务器的维护方式非常的简单灵活，总的来讲有三种方式：命令行维护、launch文件内读写和node源码读写 |  |  |
| PPT第15页: 首先我们来讲一下命令行设置参数的方法，使用命令行来维护参数服务器，主要使用 rosparam 语句来进行操作的各种命令，如下表：  $rosparam list 该命令将列出当前所有参数  $rosparam get param\_key 显示某个参数的值  $rosparam set param\_key param\_value设置某个参数的值  $rosparam dump file\_name 保存参数到文件  $rosparam load file\_name 从文件读取参数  $rosparam delete param\_key 删除参数 |  |  |
| **PPT第16页：**  其中load和dump的文件必须遵从YAML格式，YAML格式如下图所示，从图中我们可以看出，这种定义格式类似于字典的形式，即由多个“键-值”组成，左边是key，右边是value值，中间用“:”隔开，这样就能定义一系列的参数，这就是一个YAML文件的格式。以上我们就展示了如何利用命令行来操作参数服务器。 |  |  |
| PPT第17页: 除此之外，我们还可以在launch文件中设置参数，首先我们要清楚，与参数服务器相关的标签有两个：一个是<param>，另一个是<rosparam>，在这里我们打开 robot\_sim\_demo/launch/include/xbot2.launch.xml文件，查看两种标签的定义方式。  如图所示：  该文件中使用了<param>和<rosparam>两个标签来分别进行参数定义，这两个标签的功能较为将近，但<param>一般只设置一个参数，正如文件中设置的那样，该文件设置了两个<param>标签，最下面的那个设置了一个名为“publish\_frequency”，值为“100.0”的参数，并把它交给参数服务器来维护，另外一个<param>标签定义在了第十二行，它的参数name是“robot\_description”，但它并没有直接写value值是多少，代替的是command，等号后面的意思是用xacro.py这个脚本去执行robot.xacro这个文件，得到的结果作为value，赋给robot\_description。  另外一个标签就是<rosparam>，这个用法就比较固定了，一般先指定一个yaml文件，然后对它施加一个command，这里的command是load，所以整句标签可以理解为rosparam load命令，即$rosparam load xbot2\_control.yaml，这就是参数服务器在launch文件中的用法，除了上面两种方法，还有一种是利用ROS源代码对其进行读写操作，不过在今天我们就不详细赘述了，等到后续学到rospy，roscpp的时候再详细讲解。 |  |  |
| PPT第18页: ROS参数服务器为参数值使用XMLRPC数据类型，其中包括:strings, integers, floats, booleans, lists, dictionaries, iso8601 dates, and base64-encoded data |  |  |
| PPT第19页: 下面我们介绍另一种通信方式，即Action通信方式。  ROS中常用的通讯机制是topic和service，但是在很多场景下，这两种通讯机制往往满足不了我们的需求，比如关于机械臂控制的，如果用topic发布一个运动目标，由于topic没有反馈，还需要另外订阅一个机械臂状态的topic，来获得运动过程的状态。如果用service来发布运动目标，虽然可以获得反馈，但是反馈只有一次，对于我们的控制来讲数据太少了，而且如果反馈迟迟没收到，也只能傻傻等待，干不了其他事情。那么有没有一种更加适合的通讯机制，来满足类似这样场景的需求呢？当然是有的，就是我们这篇要讲到的action通讯机制。首先我们要清楚Actionlib库，Actionlib是ROS中一个很重要的库，类似service通信机制，actionlib也是一种请求响应机制的通信方式，actionlib主要弥补了service通信的一个不足，就是当机器人执行一个长时间的任务时，假如利用service通信方式，那么publisher会很长时间接受不到反馈的reply，致使通信受阻。当service通信不能很好的完成任务时候，actionlib则可以比较适合实现长时间的通信过程，actionlib通信过程可以随时被查看过程进度，也可以终止请求，这样的一个特性，使得它在一些特别的机制中拥有很高的效率。  **PPT第20页:**  Action的工作原理是client-server模式，也是一个双向的通信模式。通信双方在ROS Action Protocol下通过消息进行数据的交流通信。client和server为用户提供一个简单的API来请求目标（在客户端）或通过函数调用和回调来执行目标（在服务器端）。  工作模式的结构示意图如下： |  |  |
| PPT第21页: 通信双方在ROS Action Protocal下进行交流通信是通过接口来实现,如下图，那什么是ROS Action Protocal呢？action protocol就是预定义的一组ROS message，这些message被放到ROS topic上在 ActionClient 和ActionServer之间进行传实现二者的沟通。在此ROS message包括goal、cancel、status、feedback、result这几部分，下面我们将具体针对这几个方面做一下介绍。 |  |  |
| PPT第22页: 首先，如下所示：  goal:任务目标  cancel:请求取消任务  status:通知client当前的状态  feedback:周期反馈任务运行的监控数据  result:向client发送任务的执行结果，这个topic只会发布一次。  goal topic: Sending Goals  goal topic使用自动创建的ActionGoal message (example: [actionlib/TestActionGoal](http://docs.ros.org/api/actionlib/html/msg/TestActionGoal.html)), goal topic被用于向action server发送新的goal。 ActionGoal message封装了一条goal message并包含Goal ID。当acttion client发送goal时候将创建独一无二的goal ID和时间戳。但是也允许留空它们，这时action server将填充它们：   * **Empty *stamp***: action server收到这个时候，时间戳将设置为now() * **Empty *id***: action server收到这个时候，ID自动随机创建。不过这个ID由于action client无法知道其对应的goal，所以意义不大。   cancel topic: Cancelling Goals      cancel topic使用消息 [actionlib\_msgs/GoalID](http://docs.ros.org/api/actionlib_msgs/html/msg/GoalID.html) , 每个cancel message都有一个时间戳和goal ID，基于这些字段信息对应goal将会被取消。  status topic: Server 端goal状态更新      status topic使用消息[actionlib\_msgs/GoalStatusArray](http://docs.ros.org/api/actionlib_msgs/html/msg/GoalStatusArray.html" \t "_blank), 通知action clients有关 *server goal status信息。* action server将会以固定速率发送该消息(一般是10 Hz)。action server将一直跟踪actionclient请求的每一个目标，知道目标状态为完成。但是为了提高通信可靠性，通常在目标状态变为完成后几分钟，server才发送完成状态。  feedback topic: Asynchronous goal information       feedback topic使用自动创建的*ActionFeedback*消息, 在处理goal过程中server周期性通知client状态。因为*ActionFeedback也有一个* goal ID, 所以action client 能决定是否丢弃收到的feedback messages，发送feedback  message非强制是可选的。  result topic: Goal information upon completion     result topic使用自动生成的*ActionResult* message (如[actionlib/TestActionResult](http://docs.ros.org/api/actionlib/html/msg/TestActionResult.html" \t "_blank)), 通过它server端实现者可以发送goal完成结果给client。 *ActionResult也有一个 goal ID, 所以action client 能决定是否丢弃收到的*result messages。 虽然result可以是empty message, 但在goal完成后是强制要求必须发送的。 因此，当server端goal状态机过渡到完成状态 (Rejected, Recalled, Preempted, Aborted, Succeeded)时，result message必须发送出去。 | cancel_policy.png |  |
| PPT第23页: 下面我们将介绍Action数据格式.action文件，之前我们讲过了topic数据格式.msg文件，以及service数据格式.srv文件，这里我们就可以类比以上两种形式来理解action数据格式.action文件。 |  |  |
| **PPT第24页:**  Action文件格式主要包括目标、反馈和结果这三部分，以利用动作库进行请求响应为例解释这三部分内容：  **目标**  机器人执行一个动作，应该有明确的移动目标信息，包括一些参数的设定，方向、角度、速度等等。从而使机器人完成动作任务。  **反馈**  在动作进行的过程中，应该有实时的状态信息反馈给服务器的实施者，告诉实施者动作完成的状态，可以使实施者作出准确的判断去修正命令。  **结果**  当运动完成时，动作服务器把本次运动的结果数据发送给客户端，使客户端得到本次动作的全部信息，例如可能包含机器人的运动时长，最终姿势等等。 |  |  |
| PPT第25页: Action文件格式如下图所示：  Action文件分为三段，中间用“---”来分隔，第一部分就是goal，由client发布目标，相当于我们在service中的request，然后是result部分，它是由我们server回传给client的，最后就是feedback部分，同样也是由server发布的。而result和feedback的区别是什么呢？result它只可能传一次，也就是当我们动作执行完了，就传回来一个result。而feedback返回的是实时的状态，所以它可能多次或是一直不停的回传给client。  接下来我们具体看一下这个文件的内容：  它展示的是这么一个场景：假设我们现在有很多洗碗机，给洗碗机依次编上号（1,2,3……），那这些洗碗机能提供洗碗的动作就相当于放在server端，现在有一个client想要调用这个功能，那就需要client端发一个goal，这个goal的定义类型是“uint32 dishwasher\_id”，即指定哪个洗碗机工作。在洗碗的工程中，server会回传feedback，feedback的定义类型是“float32 percent\_complete”，也就是会回传给client端洗碗的进度。当任务执行完的时候，还会回传一个result，告诉client洗碗机总共洗了多少个碗。以上就是整个action文件所表达的内容。 |  |  |
| PPT第26页: 下面我们来用一个action实例进行讲解。  首先我们要了解 Actionlib是一个用来实现action的一个功能包集。我们在demo中设置一个场景，执行一个搬运的action，搬运过程中客户端会不断的发回反馈信息，最终完成整个搬运过程．  我们先写一个DoDishes.action文件，包括三个部分，目标，结果，反馈，如下图所示：  实现这个.action之后，还需要将这个文件加入编译…… |  |  |
| PPT第27页: 那么就需要在CMakeLists.txt文件中添加如下的编译规则，如下图所示改写文件内容：   1. find\_package(catkin REQUIRED genmsg actionlib\_msgs actionlib) 2. add\_action\_files(DIRECTORY action FILES DoDishes.action) generate\_messages(DEPENDENCIES actionlib\_msgs) |  |  |
| PPT第28页: 接下来我们修改package.xml文件的内容，添加所需要的依赖如下：  1. <build\_depend>actionlib </build\_depend>  2. <build\_depend>actionlib\_msgs</build\_depend>  3. <run\_depend>actionlib</run\_depend>  4. <run\_depend>actionlib\_msgs</run\_depend>  然后回到工作空间catkin\_ws进行编译。  本例中设置的action，定义了一个洗碗机的例子，首先写客户端，实现功能发送action请求,包括进行目标活动,或者目标活动.之后写服务器,实验返回客户端活动当前状态信息,结果信息,和反馈信息.从而实现action。 |  |  |
| PPT第29页: 现在就可以进行编译了，编译完成后会产生一系列的.msg文件，我们可以看到有以下几个.msg文件：  DoDishesAction.msg  DoDishesActionFeedback.msg  DoDishesActionGoal.msg  DoDishesActionResult.msg  DoDishesFeedback.msg  DoDishesGoal.msg  DoDishesResult.msg |  |  |
| PPT第30页: 实现后，我们可以使用rosmsg list命令查看，结果如下图所示：  从这里我们也可以看到，action确实是基于message实现的。  至此，ROS通信架构的四种通信方式就介绍结束，我们可以对比学习这四种通信方式，去思考每一种通信的优缺点和适用条件，在正确的地方用正确的通信方式，这样整个ROS的通信会更加高效，机器人也将更加的灵活和智能。机器人学会了通信，也就相当于有了“灵魂”。 |  |  |
| PPT第31页: 这一讲我们就结束了，谢谢大家聆听。 |  |  |
|  |  |  |
|  |  |  |
|  | | |
|  | | |