ros service and parameter server

1 service

1.1 Service

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。

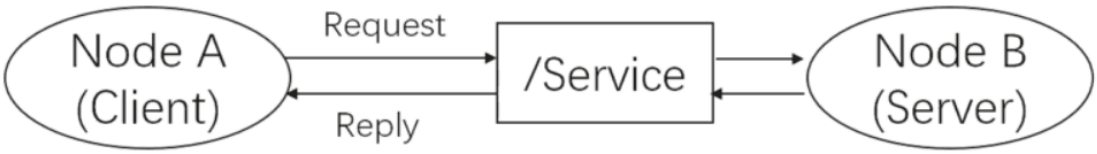
这种情况下，就需要有另外一种请求-查询式的通信模型。这节我们来介绍ROS通信中的另一种通信方式——service(服务)。

1.2 工作原理

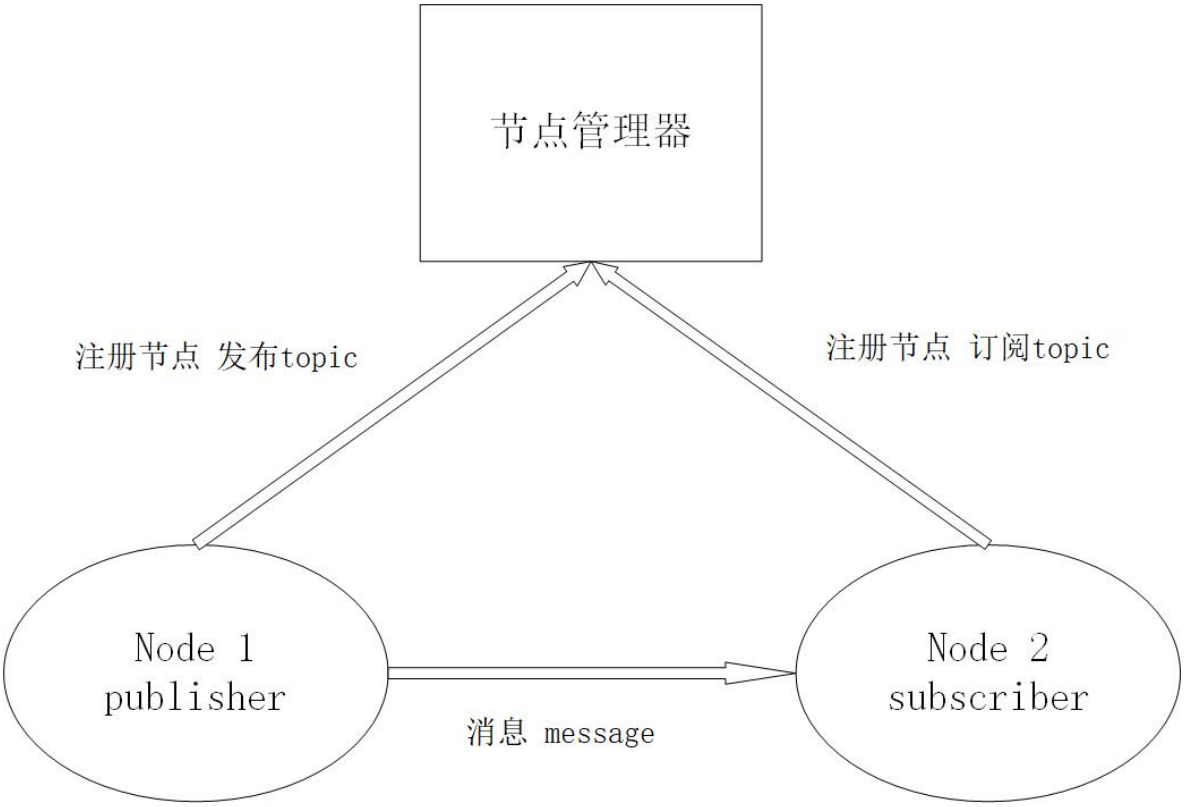
简介

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（应答方），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，类似于topic。Node A向Node B发起了请求，经过处理后得到了反馈。



回顾topic：



过程

Service是同步通信方式，所谓同步就是说，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。

1.3 topic VS service

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：

名称	Topic	Service
通信方式	异步通信	同步通信
实现原理	TCP/IP	TCP/IP
通信模型	Publish-Subscribe	Request-Reply
特点	接受者收到数据会回调（Callback）	远程过程调用（RPC）服务器端的服务
应用场景	连续、高频的数据发布	偶尔使用的功能/具体的任务
举例	激光雷达、里程计发布数据	开关传感器、拍照、逆解计算

注意：远程过程调用(Remote Procedure Call，RPC),可以简单通俗的理解为在一个进程里调用另一个进程的函数。

1.4 操作命令

在实际应用中，service通信方式的命令时`rosservice`，具体的命令参数如下表：

rosservice 命令	作用
`rosservice list`	显示服务列表
`rosservice info`	打印服务信息
`rosservice type`	打印服务类型
`rosservice uri`	打印服务ROSRPC uri
`rosservice find`	按服务类型查找服务
`rosservice call`	使用所提供的args调用服务
`rosservice args`	打印服务参数

2 Srv

2.1 简介

类似msg文件，srv文件是用来描述服务（service数据类型的，service通信的数据格式定义在*.srv中。它声明了一个服务，包括请求(request)和响应（reply）两部分。其格式声明如下：

举例：

msgs_demo/srv/DetectHuman.srv

```
bool start_detect
---
my_pkg/HumanPose[] pose_data
```

msgs_demo/msg/HumanPose.msg

```
std_msgs/Header header
string uuid
int32 number_of_joints
my_pkg/JointPose[]joint_data
```

msgs_demo/msg/JointPose.msg

```
string joint_name
geometry_msgs/Pose pose
floar32 confidence
```

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。srv文件格式很固定，第一行是请求的格式，中间用**---**隔开，第三行是应答的格式。在本例中，请求为是否开始检测，应答为一个数组，数组的每个元素为某个人的姿态（HumanPose）。而对于人的姿态，其实是一个msg，所以srv可以嵌套msg在其中，但它不能嵌套srv。

2.2 操作命令

具体的操作指令如下表：

rossrv 命令	作用
`rossrv show`	显示服务描述
`rossrv list`	列出所有服务
`rossrv md5`	显示服务md5sum
`rossrv package`	列出包中的服务
`rossrv packages`	列出包含服务的包

3 Parameter server

3.1 简介

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。

字典简介

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：

Key	/rosdistro	/rosversion	/use_sim_time	…
Value	‘kinetic’	‘1.12.7’	true	…

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。

维护方式

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
- node源码

下面我们来一一介绍这三种维护方式。

3.2 命令行维护

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 命令	作用
`rosparam set param_key param_value`	设置参数
`rosparam get param_key`	显示参数
`rosparam load file_name`	从文件加载参数
`rosparam dump file_name`	保存参数到文件
`rosparam delete`	删除参数
`rosparam list`	列出参数名称

load&&dump文件

load和dump文件需要遵守YAML格式，YAML格式具体示例如下：

```
name:'Zhangsan'
age:20
gender:'M'
score{Chinese:80,Math:90}
score_history:[85,82,88,90]
```

简明解释。就是“名称+：+值”这样一种常用的解释方式。一般格式如下：

```
key : value
```

遵循格式进行定义参数。其实就可以把YAML文件的内容理解为字典，因为它也是键值对的形式。

3.3 launch文件内读写

例子：

1、file方式

```
<launch>
    <node pkg="commu_node" type="commu_node.py" name="commu_node"
output="screen">
        <rosparam file="$(find commu_node)/config/wifi_param.yaml"
command="load" />
    </node>
</launch>
```

2、直接给值：

```
  <node name="mk_gimbalctr" pkg="mk_gimbalctr" type="mk_gimbalctr"
output="screen">
      <param name="stepRoll" type="double" value="0.0"/>
      <param name="stepPitch" type="double" value="0.0"/>
      <param name="stepYaw" type="double" value="0.0"/>
  </node>
```

3.4 node源码方式

利用API来对参数服务器进行操作。主要是代码内如何读取参数：

```
#include<ros/ros.h>
ros::NodeHandle nh;
int parameter1, parameter2, parameter3;
//Get Param的三种方法
//① ros::param::get()获取参数“param1”的value，写入到parameter1上
bool ifget1 = ros::param::get("param1", parameter1);
//② ros::NodeHandle::getParam()获取参数，与①作用相同
bool ifget2 = nh.getParam("param2",parameter2);
//③ ros::NodeHandle::param()类似于①和②
//但如果get不到指定的param，它可以给param指定一个默认值(如33333)
nh.param("param3", parameter3, 33333);
```