智能无人机技术设计实践
--ROS通信

于超

联系方式: yc19@mails.tsinghua.edu.cn

时间：2019.10.12

目 录

- 1 ROS通信架构基础
  - 1.1 Node & Master
  - 1.2 Launch文件
- 2 ROS的三种主要通信方式
  - 2.1 Topic & Msg
  - 2.2 Service & Srv
  - 2.3 Parameter server

1 ROS通信架构基础

1.1 Node & Master

◆ Node

- 在ROS世界里，最小的进程单元就是节点（node）。
- 一个软件包里可以有多个可执行文件，可执行文件在运行之后就成了一个进程(process)，这个进程在ROS中就叫做节点。
- 从程序角度来说，node就是一个可执行文件（通常为C++编译生成的可执行文件、Python脚本）被执行，加载到了内存之中；从功能角度来说，通常一个node负责者机器人的某一个单独的功能。
- 由于机器人的功能模块非常复杂，我们往往不会把所有功能都集中到一个node上，而会采用分布式的方式，把鸡蛋放到不同的篮子里。
- 例如有一个node来控制底盘轮子的运动，有一个node驱动摄像头获取图像，有一个node驱动激光雷达，有一个node根据传感器信息进行路径规划。
- 这样可以很方便的对各个模块进行修改和异常处理。

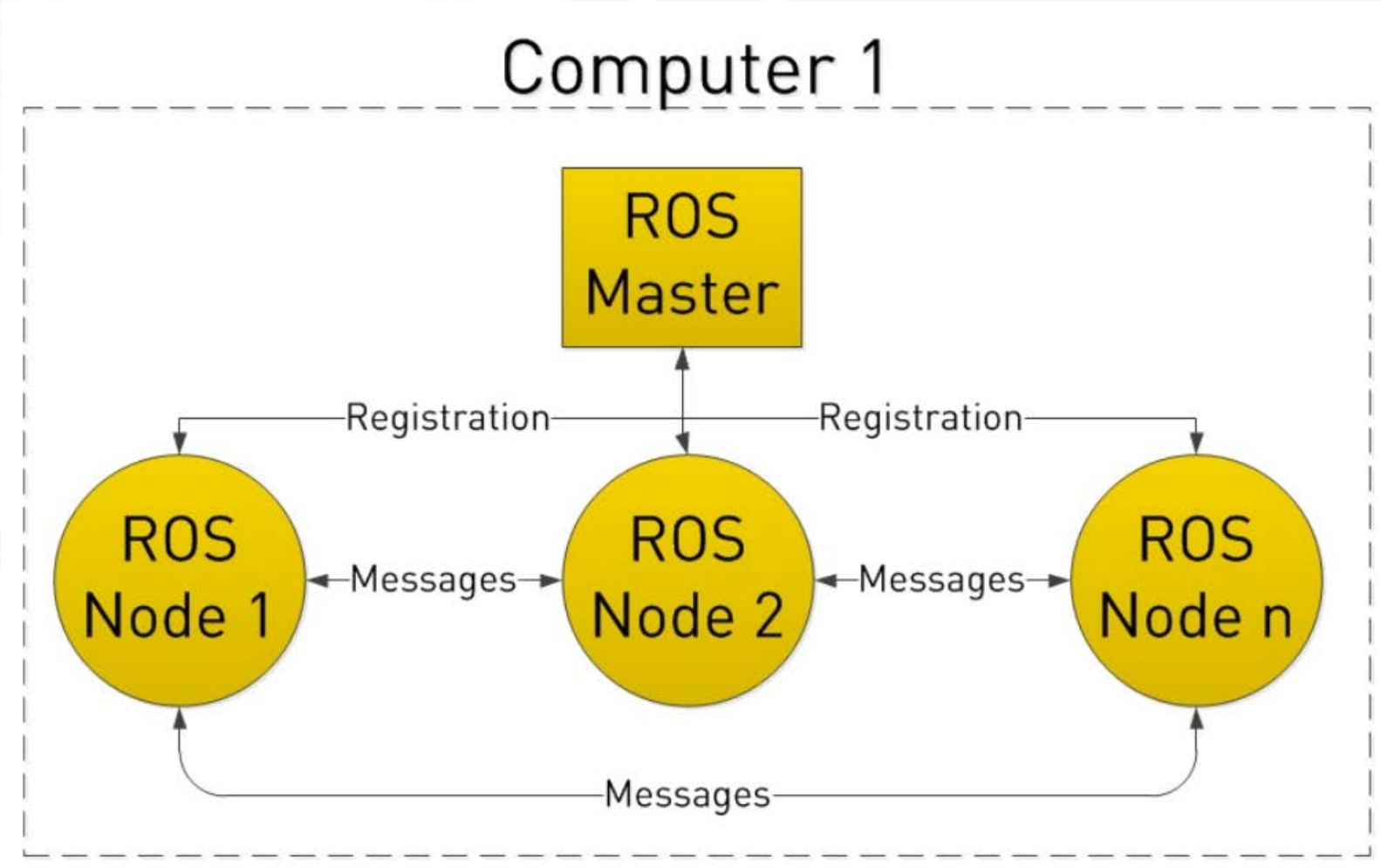
1.1 Node & Master

◆ Master

- 由于机器人的元器件很多，功能庞大，因此实际运行时往往会运行众多的node，负责感知世界、控制运动、决策和计算等功能，这就需要一个管理器来调配、管理这些node。
- ROS提供给我们一个节点管理器master, master在整个网络通信架构里相当于管理中心，管理着各个node。
- node首先在master处进行注册，之后master会将该node纳入整个ROS程序中。
- node之间的通信也是先由master进行“牵线”，才能两两的进行点对点通信。
- 当ROS程序启动时，第一步首先启动master，由节点管理器处理依次启动node。

1.1 Node & Master

- Node和Master的关系图



1.2 Launch文件

- launch文件
  - 机器人是一个系统工程，通常一个机器人运行操作时要开启很多个node。
  - ROS提供了launch文件来避免每次启动某个package时依次开启其中每一个node的繁琐操作。
  - launch文件储存在package的/launch目录下，每一个launch文件指定了对于这个package的一种启动方式，包括开启哪些节点、读取哪些配置文件，预置哪些参数等等。
  - roslaunch xxx yyy.launch
  - 不需要提前在一个终端中运行 roscore 命令。roslaunch 会自动启动；如果已经启动了roscore，这个roslaunch就不会再启动roscore了。

```
<launch>
        <group ns="turtlesim1">
                <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
        </group>

        <group ns="turtlesim2">
                <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
        </group>
</launch>
```

1.2 Launch文件

- 节点元素
  - 每个node的必须属性
    - ✓ pkg： 表示该节点的package，相当于 rosrun 命令后面的第一个参数；
    - ✓ type： 可执行文件的名字，rosrun 命令的第二个参数；
    - ✓ name： 该节点的名字，相当于代码中 ros::init() 中设置的信息，有了它代码中的名称会被覆盖。

```
<node
pkg=” turtlesim”                    //功能包名
 type=” sim”                        //节点名
name=” turtlesim_node”              //节点重命名
 />
```

    - ✓ 可选属性

```
respawn=” true”        //自动重启
required=” true”       //必要节点，这个出问题了，整个launch结束
 output=” screen”       //输出打印信息
launch-prefix=” xterm -e”    //单独维护一个窗口  比如要输入或者单独查看日志信息
ns=” name-space”  //定义命名空间，同一命名空间下节点的话题才能通信
<remap from="original-name" to "new-name"> //话题重映射
```

1.2 Launch文件

◆ 组元素

- 若干节点放在同一个组内

```
<group ns="name-space">
……
</group>
```

- group可以设置一些规则

```
<group if="$arg arg-name">
<group unless="$arg arg-name"> //可以通过这个参数的值，决定执行这个组，还是不执行这个组
```

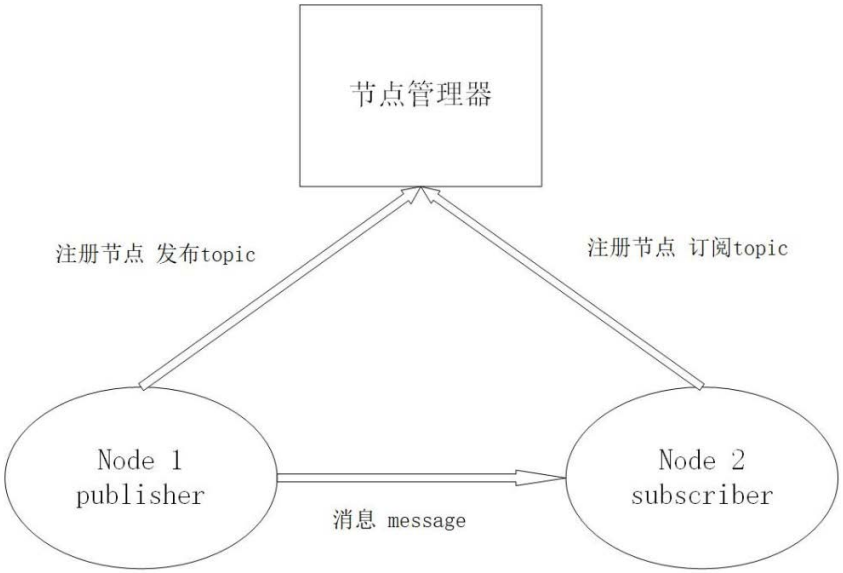
◆ 其他操作

```
<arg name="art-name" />   //参数声明
<arg name="fcu_url“ default="/dev/ttyACM0:57600" /> //参数赋值（也可以在启动launch时赋值 arg-name:="arg-value"）
<include file=“$(find mavros)/launch/node.launch”> //在这个launch文件中启动其它launch文件
<rosparam file=“$(find my_dynamixel_tutorial)/tilt.yaml” command=“load”/>
//rosparam：一次性将多个参数加载到参数服务器中. 我们将需要设置的参数放到 .yaml 文件中。
```

2 ROS的三种主要通信方式

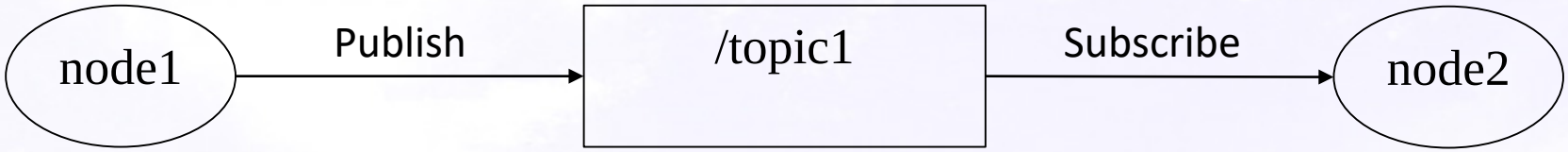
2.1 Topic & Msg

- topic
  - 话题（topic）是ROS通信方式中最常用的一种，通常用来传输实时性、周期性的消息。
  - topic是一种node和node之间的单向通信方式。
  - 一个topic通信模型中包含两类节点——Publisher和Subscriber。两种节点都需要在master处注册。Publisher会以一定频率把信息发布到与其绑定的topic上，Subscriber可以订阅一个或数个topic。
  - 整个通信过程是单向的，信息从Publisher单向流向Subscriber。



2.1 Topic & Msg

- topic



- topic是一种异步通信方式。

node1是Publisher，它以一定的频率将新的信息更新到/topic1这个topic上。而node2作为Subscriber，在检测到/topic1更新的时候，调用回调函数（callback）拿到/topic1中的数据并进行处理。一个topic可能同时存在多个Publisher和Subscriber。

对于Publisher来说，它并不关心当前topic的状态和接受的情况，它只负责按照自己的设定更新topic。同样，对于Subscriber来说，它也不关心当前topic由谁写入，它只负责在每次topic更新的时候进行接收和处理。两者各司其职，不存在协同工作，故称为异步通信。

2.1 Topic & Msg

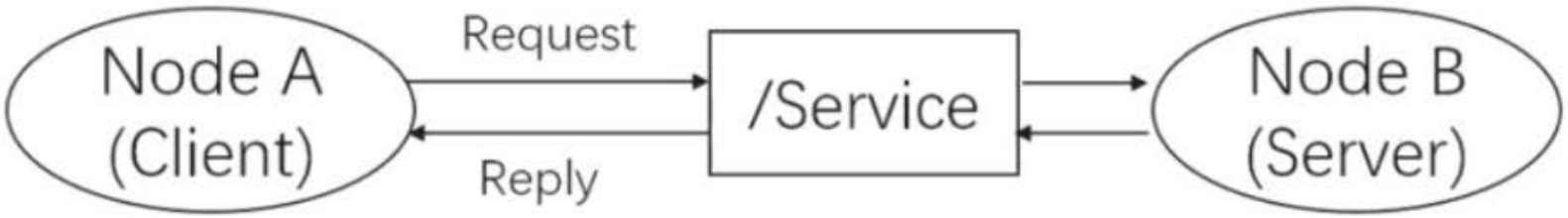
◆ Message

- topic有很严格的格式要求，保证发送方和接收方能够通过topic正常地进行通信。这个格式我们称为消息（message，文件名为msg）。每一个topic都需要一个message来对其格式进行规范，而一个message格式可以指导多个topic。
- msg可以涵盖的基本类型包括bool、int8、int16、int32、int64(以及uint)、float、float64、string、time、duration、header、可变长数组array[]、固定长度数组array[C]。一个msg里可以包含数个变量，每次信息传递的过程中所有的变量都需要被赋值。
- msg以.msg文件的形式存放在/package/msg路径下。每一种不同的msg都需要在CMakeLists.txt中进行登记，这样编译的时候ROS系统会对所有的.msg进行编译并生成一个对应的格式，从而用于topic的通信。

2.2 Service & Srv

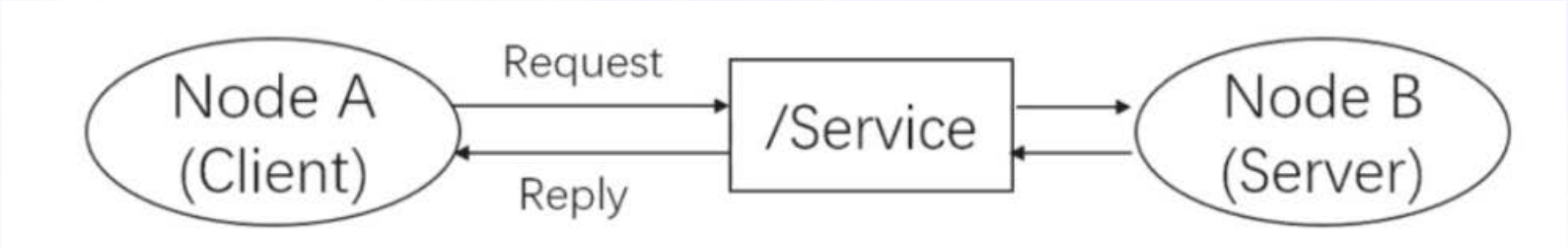
◆ Service

- 当单向的通信不能满足设计需求或不需要周期性传递信息时，ROS提供了服务（service）这一通信方式，来实现一种请求-查询式的双向同步通信模型。
- Service的通信模型中包含请求方（client）和应答方（server）两类节点。当client有需求时，它会发送一个请求（request），并等待回应。server收到请求并处理后返回一个回应（reply），client接收到reply并对其进行处理，完成通信。整个通信过程是双向的。
- 通常一个service只有一个server，并且允许多个client对其进行询问。



2.2 Service & Srv

◆ Service



- service是一种同步通信方式。当client发送request后，它会进入阻塞状态，即在原地进行等待，直到接收到reply才继续执行，故称之为同步通信。当server接收到request时，它会调用处理函数（handle_function）对接收到的request中的信息进行处理并返回一个reply。这个过程对于server是FIFO（First in First out，先入先出）的，server会按照接收request的次序处理每一个request并返回reply。
- service相对于topic的特点是双向、同步。它的通信较为稀疏，在不需要频繁通信的情况下可以节省通信资源，简单而高效。

2.3 Service & Srv

◆ Srv

- 与topic相同，service也有相应的格式要求。这个格式我们称为服务（service，文件名为srv）。每一个service都需要一个srv来对其格式进行规范，一个srv格式可以指导多个topic。
- srv涵盖的变量格式与topic相同。一个srv里可以包含数个变量，每次信息传递的过程中所有的变量都需要被赋值。
- 与topic不同的是，srv文件分为request和reply两部分，分别规定了request和reply的格式，。
- srv以.srv文件的形式存放在/package/srv路径下。与topic相同，每一种不同的srv都需要在CMakeLists.txt中进行登记。

2.4 Parameter Server

◆ Parameter Server

- 与service和topic不同，参数服务器（Parameter Server）是一种特殊的通信方式。
- Parameter Server是node存储参数的地方，用于配置参数和全局共享参数。相比于前两种通信方式，Parameter Server更加静态，它维护了一个数据字典，存储着各种参数和配置。
- 这里的字典其实就是指键值对。每一个键（key）对应着一个值（Value）。当node想要查询某个参数值时，只需要给出key，就可以在Parameter Server中索引得到这个key对应的参数值。
- Parameter Server可以通过命令行、launch文件和node源码三种方式进行维护和更改。

Key	/rosdistro	/rosversion	/use_sim_time	…
Value	'kinetic'	'1.12.7'	true	…

谢谢！