Commander 机器人软件架构

FastRTPS 实现机制及使用文档

1. 文档说明

本文档主要用于介绍 Commander 机器人进程间通信的机制-DDS 及其中间件 FastRTPS 的原理和具体使用方式。

2. DDS 及 FastRTPS 简介

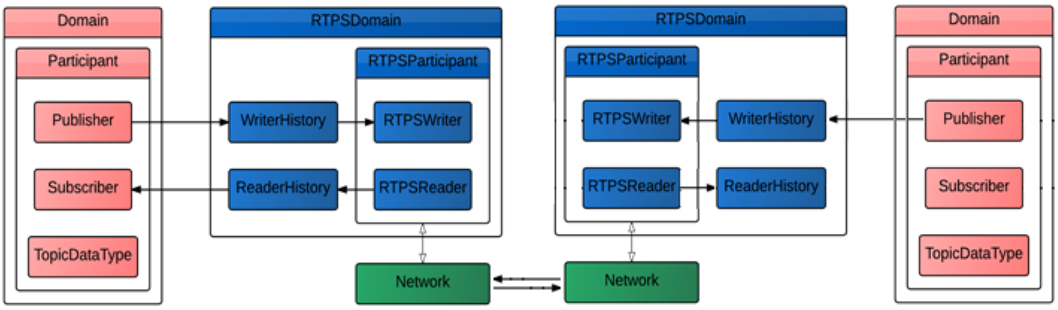
数据分发服务 DDS(DataDistributionService)是对象管理组织(OMG)在 HLA 及 CORBA 等标准的基础上制定的新一代分布式实时通信中间件技术规范，DDS 采用发布/订阅体系架构，强调以数据为中心，提供丰富的 QoS 服务质量策略，能保障数据进行实时、高效、灵活地分发，可满足各种分布式实时通信应用需求。DDS 信息分发中间件是一种轻便的、能够提供实时信息传送的中间件技术。

eProsima Fast RTPS 是 RTPS(实时发布-订阅)协议的 c++实现，该协议在不可靠的传输(如 UDP)上提供发布-订阅通信，由对象管理组(OMG)定义和维护。RTPS 也是由 OMG 为数据分发服务(DDS)标准定义的有线互操作性协议。

FastRTPS 的一些主要特点是:

- 可配置的 best-effort 和 reliable 发布-订阅通信策略
- 即插即用连接，使网络中的任何其他成员都能自动发现任何新应用
- 模块性和可伸缩性，允许网络中复杂和简单设备的持续增加
- 可配置的网络和可换的传输层:为每个实现选择最佳协议和系统输入/输出通道组合
- 两个 API 层：high-level 的 publisher-subscriber 层和 low-level 的 reader-writer 层

3. FastRTPS 实现



上图是 FastRTPS 的原理示意图，最底层有一个域的概念（domain），在一个 domain 内的所有 participant 才有互相通信的可能性，不同 domain 的 participant 无法互相通信（可以认为毫无关联）；domain 内以 Participant 为通信实体的单位，每一个 Participant 可以创建 n 个 subscriber 和 m 个 publisher（n,m 可以为 0），且每一个 subscriber/publisher 对应一个确定的 topic，负责 sub/pub 这个 topic 的消息。

4. 性能测试

（1）./LatencyTest publisher -n 5 -s 10000（1 发 5 收，besteffort）

```
Printing round-trip times in us, statistics for 10000 samples
   Bytes, Samples,   stdev,    mean,     min,     50%,     90%,     99%,  99.99%,     max
--------,--------,--------,--------,--------,--------,--------,--------,--------,--------,
      16,   10020,   99.00,  184.27,   -0.02,  173.13,  230.15,  510.48, 2851.85, 3650.09
      32,   10029,  110.00,  198.37,   -0.00,  175.39,  285.93,  614.67, 1393.75, 2353.27
      64,   10030,  148.00,  207.32,    0.01,  177.70,  296.61,  725.88, 3650.90, 4481.76
     128,   10026,  125.00,  206.66,    0.00,  179.81,  294.77,  732.35, 1933.83, 2890.97
     256,   10025,   92.00,  181.37,    0.00,  171.06,  227.63,  527.03,  431.47, 2547.63
     512,   10029,  105.00,  193.48,    0.00,  174.84,  258.45,  614.60, 1422.90, 1526.67
    1024,   10025,   60.00,  174.95,   -0.03,  170.93,  213.62,  439.46,  747.08, 2115.72
    2048,   10016,   77.00,  186.46,   -0.04,  181.31,  224.55,  466.13, 1409.63, 4156.32
    4096,   10034,   73.00,  192.66,   -0.08,  185.01,  227.08,  484.35, 1223.65, 3381.21
    8192,   10010,   58.00,  200.95,    0.24,  193.53,  234.35,  475.21,  728.16, 1353.60
   16384,   10045,   70.00,  216.43,   -0.07,  208.91,  251.17,  481.71, 1493.09, 3415.11
```

（2）./LatencyTest publisher -n 1 -s 10000（1 发 1 收，besteffort）

```
Printing round-trip times in us, statistics for 10000 samples
   Bytes, Samples,   stdev,    mean,     min,     50%,     90%,     99%,  99.99%,     max
--------,--------,--------,--------,--------,--------,--------,--------,--------,--------,
      16,   10000,   79.00,  206.97,  145.21,  190.51,  228.12,  553.38, 1797.33, 3400.24
      32,   10000,   63.00,  207.66,  131.00,  191.07,  235.80,  518.97, 1266.69, 1399.89
      64,   10000,   46.00,  178.85,  114.69,  167.05,  188.35,  432.47,  961.93, 1038.63
     128,   10000,   58.00,  194.24,  135.30,  188.21,  216.93,  494.74, 1099.85, 1423.95
     256,   10000,   46.00,  180.15,  129.55,  167.88,  191.03,  432.17,  724.09,  832.99
     512,   10000,   47.00,  179.97,  131.56,  167.61,  189.79,  427.10,  809.23, 1413.31
    1024,   10000,   50.00,  183.82,  135.51,  170.07,  195.51,  454.80, 1127.38, 1132.63
    2048,   10000,   44.00,  183.48,  143.79,  171.64,  194.37,  434.70,  747.15,  906.00
    4096,   10000,   57.00,  191.69,  139.96,  175.70,  202.52,  477.08, 1228.36, 1263.12
    8192,   10000,   51.00,  200.63,  147.76,  186.35,  213.57,  464.72, 1042.35, 1245.94
   16384,   10000,   56.00,  213.01,  113.64,  199.77,  224.30,  476.59, 2126.92, 2445.79
```

（3）./LatencyTest publisher -n 5 -s 10000（1 发 5 收，reliable）

```
Printing round-trip times in us, statistics for 10000 samples
   Bytes, Samples,   stdev,    mean,     min,     50%,     90%,     99%,  99.99%,     max
--------,--------,--------,--------,--------,--------,--------,--------,--------,--------,
      16,   10002,  170.00,  418.41,   -0.12,  385.70,  620.58,  906.79, 3190.96, 4094.31
      32,   10007,  144.00,  402.29,   -0.09,  376.75,  593.05,  840.47, 1695.13, 2348.44
      64,   10000,  145.00,  401.58,   -0.05,  374.76,  597.39,  837.34, 2039.62, 2363.37
     128,   10000,  154.00,  407.37,   -0.04,  377.77,  601.74,  884.95, 2266.07, 3923.77
     256,   10002,  156.00,  399.22,   -0.08,  371.00,  587.11,  833.92, 2406.59, 5174.71
     512,   10000,  146.00,  403.78,   -0.08,  376.14,  598.71,  840.12, 1354.92, 3630.85
    1024,   10001,  149.00,  403.82,   -0.11,  377.20,  592.29,  825.79, 2494.78, 3879.71
    2048,   10008,  177.00,  432.16,   -0.08,  399.39,  633.39,  964.29, 2624.59, 4729.69
    4096,   10003,  386.00,  441.09,   -0.12,  395.29,  641.17, 1010.84,13073.68,26852.80
    8192,   10002,  458.00,  488.93,   -0.14,  428.49,  715.31, 1271.65,13329.53,26506.40
   16384,   10002,  948.00,  513.54,   -0.11,  453.95,  732.59, 1188.91,14073.81,85691.26
```

（4）./LatencyTest publisher -n 1 -s 10000（1 发 1 收，reliable）

```
Printing round-trip times in us, statistics for 10000 samples
   Bytes, Samples,   stdev,    mean,     min,     50%,     90%,     99%,  99.99%,     max
--------,--------,--------,--------,--------,--------,--------,--------,--------,--------,
      16,   10000,  102.00,  329.11,  208.69,  295.81,  408.52,  723.65, 1562.77, 1835.78
      32,   10000,   86.00,  310.94,  149.80,  289.99,  355.07,  701.25, 1626.82, 1650.79
      64,   10000,   63.00,  302.40,  175.66,  288.64,  337.16,  604.46, 1323.16, 1351.99
     128,   10000,   83.00,  347.39,  195.36,  328.10,  438.96,  690.00, 1190.42, 1747.00
     256,   10000,   68.00,  303.82,  123.01,  290.10,  339.25,  617.66, 1300.64, 2850.79
     512,   10000,   83.00,  381.30,  207.74,  369.48,  457.43,  725.51, 1071.67, 1098.36
    1024,   10000,   80.00,  384.02,  219.92,  368.79,  454.32,  739.70, 1442.62, 1536.14
    2048,   10000,   82.00,  342.95,  216.94,  321.73,  427.34,  698.55, 1087.39, 2449.97
    4096,   10000,   80.00,  334.69,  220.74,  305.26,  418.68,  682.40, 1387.64, 1525.48
    8192,   10000,   64.00,  324.93,  226.41,  312.10,  356.16,  638.18, 1360.03, 1379.71
   16384,   10000,   61.00,  344.10,  240.25,  329.66,  388.58,  641.16, 1155.24, 1311.40
```

5. 例程

（1）概述

该例程共包含 4 个进程：applefarmer、pearfarmer、retailer、warehouse。其中 applefarmer 向 appletopic 发布 apple 消息，消息内容表示苹果数量和大小（本例程中苹果大小的数据不考虑）；pearfarmer 向 peartopic 发布 pear 消息，消息内容表示梨的数量和大小（本例程中梨大小的数据不考虑）；retailer 订阅 appletopic 和 peartopic 两个话题，当接收到任何一个消息时，把接收到的数量通过 storagetopic 发布一条 storage 消息；warehouse 订阅 storagetopic，对所有 storage 消息中的数量进行累加，每接收到一条消息输出一次总数量。具体代码见 dds_example.tar.gz->1_farmer_retailer_warehouse；

（2）实现 applefarmer 和 pearfarmer。

这两个进程的实现基本一样，唯一的区别是，main 函数中设定 applefarmer 每秒发布一次 apple 消息，pearfarmer 每 3 秒发布一次 pear 数据。

1> 新建一个文件夹 applefarmer
2> 拷贝 CMakeLists.txt、dds_participant.h、dds_publisher.h、dds_subscriber.h 四个文件
3> 新建 apple.idl 文件，键入文件内容见源码
4> 在该文件夹下执行 shell 命令 fastrtpsgen apple.idl –replace 。会生成 applePubSubTypes.cxx，applePubSubTypes.h, apple.h, apple.cxx 四个文件，其中前两个可认为是 topic 文件，后两个可以认为是消息文件
5> 创建 main.cc 文件，并键入源码中的内容（注意看注释部分内容）
6> 编辑 CMakeLists.txt 如源码
7> 新建 build 文件夹
8> 进入 build，执行 shell 命令：cmake ..
9> 成功后，执行 make
10> 编译成功后会在当前文件夹下看到可执行文件
11> 按照相同的操作即可实现 pearfarmer

（3）实现 warehouse

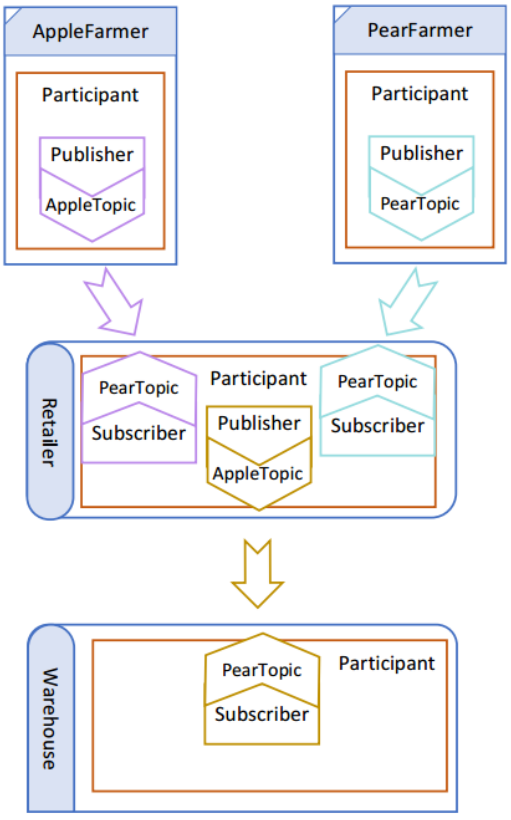
1> 新建一个文件夹 warehouse
2> CMakeLists.txt、dds_participant.h、dds_publisher.h、dds_subscriber.h 四个文件
3> 新建 storage.idl 文件，键入文件内容见源码
4> 在该文件夹下执行 shell 命令 fastrtpsgen storage.idl –replace 。会生成 storagePubSubTypes.cxx，storagePubSubTypes.h, storage.h, storage.cxx 四个文件
5> 以下步骤同（2）中所述

（4）实现 retailer

6> 新建一个文件夹 retailer
7> 拷贝 CMakeLists.txt、dds_participant.h、dds_publisher.h、dds_subscriber.h 四个文件
8> 将之前实现的 apple.idl、pear.idl 和 storage.idl 文件拷贝过来
9> 在该文件夹下执行 shell 命令 fastrtpsgen apple.idl pear.idl storage.idl –replace。会生成 applePubSubTypes.cxx, applePubSubTypes.h, apple.h, apple.cxx, pearPubSubTypes.cxx, pearPubSubTypes.h, pear.h, pear.cxx，storagePubSubTypes.cxx，storagePubSubTypes.h, storage.h, storage.cxx 共 12 个文件
10> 以下步骤同（2）中所述
11> main.cc 文件流程：
创建一个全局 Storage 类的对象用于存储数据；
创建一个 participant；
创建一个 publisher，用于在 StorageTopic 发送 Storage 消息；
创建一个 subscriber，用于订阅 AppleTopic 的 Apple 消息；
创建一个 subscriber，用于订阅 PearTopic 的 Pear 消息；
每隔 5 秒向 StorageTopic publish 一个 Storage 消息；
当接收到 AppleTopic 或 PearTopic 消息时调用对应的回调函数

（5）例程测试

分别在上述 4 个程序的可执行文件所在的文件夹中执行 shell 指令运行程序（无需额外参数），即可运行例程。程序流程见下图：



6. fastrtpsgen 及 idl 文件的使用

fastrtpsgen 是 eProsima 推出的一个 Java 应用程序，它可以通过 IDL 文件中定义的数据

类型生成源代码。这些生成的源代码可以用于应用程序中，以便发布和订阅该类型的 topic。

下图是 fastrtpsgen 在使用过程中的一些选项，通常只需要使用到其中的-replace 选项。例如，需要利用一个名为 hello.idl 的文件生成.cxx 和.h 文件的源码，可以在该文件所在的位置使用 shell 指令：fastrtpsgen hello.idl –replace ，即可生成 hello.cxx，hello.h，helloPubSubTypes.cpp，helloPubSubTypes.h 四个文件，其中可认为前两个是消息文件，后两个是 topic 文件。

注：首次生成的时候可以不加-replace，而当需要重新生成时必须使用

Option	Description
-help	Shows the help information.
-version	Shows the current version of eProsima FASTRTPSGEN.
-d <directory>	Sets the output directory where the generated files are created.
-I <directory>	Add directory to preprocessor include paths.
-t <directory>	Sets a specific directory as a temporary directory.
-example <platform>	Generates an example and a solution to compile the generated source code for a specific platform. The help command shows the supported platforms.
-replace	Replaces the generated source code files even if they exist.
-ppDisable	Disables the preprocessor.
-ppPath	Specifies the preprocessor path.
-typeobject	Generates *TypeObject* files for the IDL provided and modifies MyType constructor to register the TypeObject representation into the factory.

下图是 idl 文件中的数据类型与 c++中数据类型的对照表，表中只列出了基础数据类型，如果涉及到更负责的数据类型，可以参见参考文档部分的内容。

IDL	C++11
char	char
octet	uint8_t
short	int16_t
unsigned short	uint16_t
long	int32_t
unsigned long	uint32_t
long long	int64_t
unsigned long long	uint64_t
float	float
double	double
long double	long double
boolean	bool
string	std::string

7. 通过 XML 文件对 publisher 和 subscriber 进行详细配置

根据 FastRTPS 的官方文档，FastRTPS 在使用过程中可以对很多内容进行详细的配置，可以通过 XML 文件的形式来实现。示例代码见 dds_example.tar.gz->2_imu_with_xml_conf

代码中 ImuPub 每隔 1 秒 publish 一条 Imu 消息，ImuSub 订阅该消息，当消息到达时在屏幕输出该消息。与上一个例程最主要的差别集中在 publisher/subscriber 的创建的过程中：

```
13  using namespace eprosima;
14  using namespace fastrtps;
15  using namespace rtps;
16  using namespace std;
17
18  int main(int argc, char** argv)
19  {
20      //构造一个Imu的对象，这个对象即publish出去的内容
21      Imu my_imu;
22      //初始化数据
23      my_imu.yaw(3.0);
24      //载入配置用的XML文件（默认路径是可执行文件所在的路径）
25      xmlparser::XMLProfileManager::loadXMLFile("imu.xml");
26      //创建一个participant
27      DdsParticipant my_participant;
28      eprosima::fastrtps::Participant* my_part_ptr = my_participant.Create("helloparticipant", 100);
29      //创建一个publisher，并指定topic的名字和topic的数据类型
30      DdsPublisher<Imu, ImuPubSubType> my_pub;
31      /*第一个参数是participant的指针，第二个参数是xml文件中的profile*/
32      my_pub.Create(my_part_ptr, "imu_publisher_profile");
33      for(;;)
34      {
35          //publish数据
36          my_pub.PublishDds(&my_imu);
37          cout << "ImuPub published imu.yaw: " << my_imu.yaw() << endl;
38          my_imu.yaw(my_imu.yaw() + 1);
39          sleep(1);
40      }
41      Domain::stopAll();
42      return 0;
43  }
```

ImuPub 的 main.cc 中 line 25 载入 XML 文件，line 32 从 XML 文件中读取 profile，完成对 publisher 的配置，ImuSub 的过程完全一样。

对于 XML 文件的编写，请参见以下文档：

https://eprosima-fast-rtps.readthedocs.io/en/latest/xmlprofiles.html

8. 参考文档

（1）FastRTPS 用户文档：

https://eprosima-fast-rtps.readthedocs.io/en/latest/introduction.html

（2）FASTRTPSGEN 使用手册

https://eprosima-fast-rtps.readthedocs.io/en/latest/geninfo.html

（3）FastRTPS 源码：https://github.com/eprosima

（4）本文例程源码：https://github.com/gaoyadianta/sharefiles