ROS Notebook

Wu Yutian

2021.11.13

前言

主要参考了胡春旭的《ROS 机器人开发实践》一书。

Wu Yutian

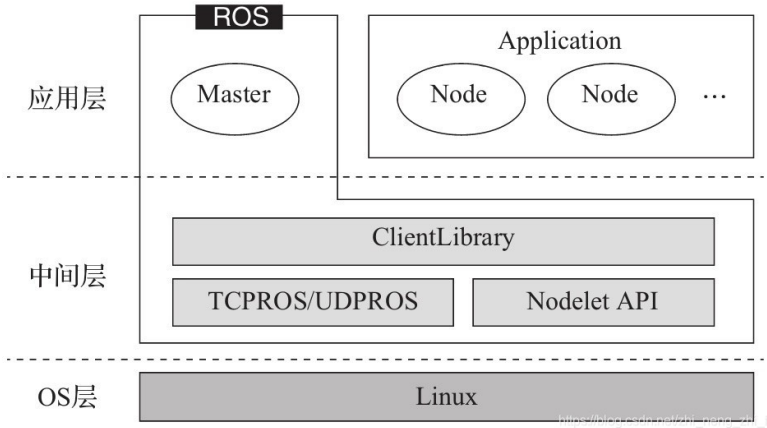
2021.11.13

Contents

Chapter 1

ROS 的基本架构——ROS1

1.1 整体架构



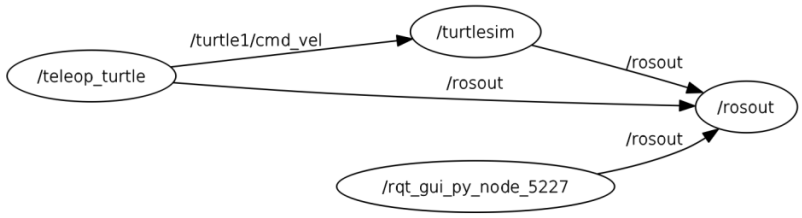
OS 层：是 ROS 依托的底层操作系统，一般是 Ubuntu。

中间层：最重要的就是基于 TCP/UDP 网络，进行封装形成的 TCPROS/UDPROS 通信系统，这其中包括了 Topic 的发布、订阅的通信方式，Service 的客户端、服务器的通信方式等。另外 ROS 还提供了一种进程内通信的方式——Nodelet，可以为多进程通信提供一种更优化的数据传输方式，适合对实时性要求较高的应用。

在通信机制的基础上，ROS 还在中间层提供了大量的机器人开发相关的实用功能，如：数据类型定义、坐标变换、运动控制等。

应用层：ROS 需要运行一个管理者——Matser，负责整个系统的正常运行。其他的一些相关的 ROS 功能包都是以节点（Node）的方式运行，一般来说，简单的开发工作只需要关注节点的标准输入输出接口，而不需要关注模块的内部实现。

1.2 计算图的视角



从计算图的视角来看 ROS 的功能模块，它们都是以节点为单位独立运行的，甚至可以分布于不同的主机中。

1.2.1 节点 Node

节点就是一些执行运算任务的进程，它们之间可以相互通信。

1.2.2 话题 Topic

消息以一种发布/订阅（publish/subscribe）的方式传递，发布者和订阅者并不了解彼此的存在，系统中可能有多个节点发布或者订阅同一个话题的消息。

1.2.3 服务 Service

对于双向的同步传输模式，采用基于客户端/服务器（Client/Server）的模型，包含请求和应答，类似于 Web 服务器，ROS 中只允许有一个节点提供指定命名的服务。

1.2.4 节点管理器 Master

节点管理器帮助 ROS 节点之间相互查找、建立连接，同时还为系统提供参数服务器，管理全局参数。

1.3 文件系统

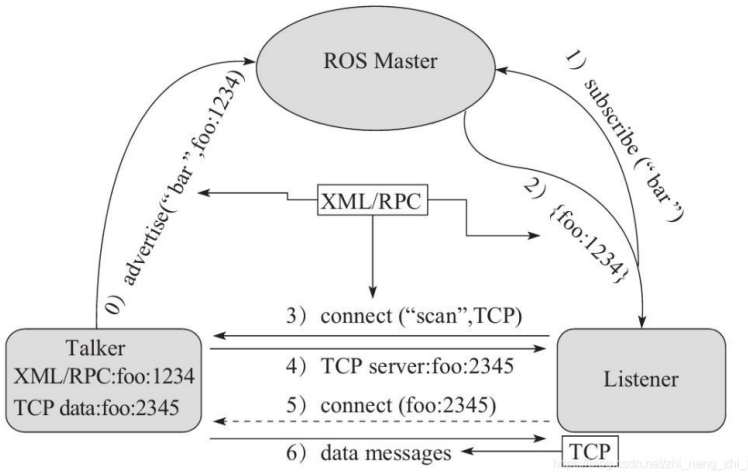
1.3.1 功能包

功能包相关的常用 ROS 命令：

命令	作用
catkin_create_pkg	创建功能包
rospack	获取功能包的信息
catkin_make	编译功能包的信息
rosdep	自动安装功能包依赖的其他包
roscd	功能包目录跳转
roscp	拷贝功能包中的文件
rosed	编辑功能包中的文件
rosrun	运行功能包中的可执行文件
roslaunch	运行启动文件

1.4 通信机制

1.4.1 话题通信机制——Topic



假设 Talker 首先启动，建立通信的详细过程：

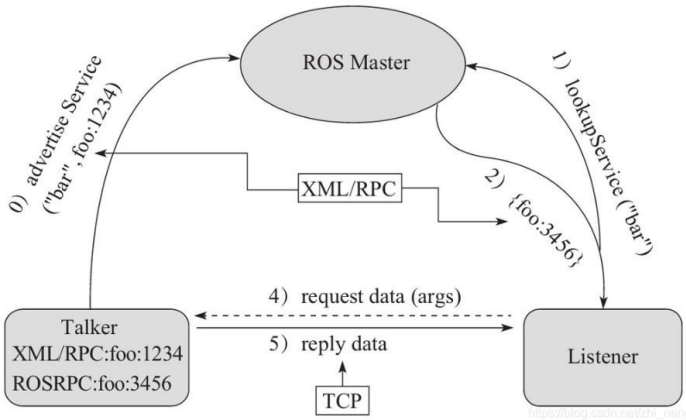
- 1、发布者（Talker）启动，通过 RPC 向 ROS Master 注册发布者的信息，包括：发布者节点信息，话题名，话题缓存大小等；Master 会将这些信息加入注册列表中；
- 2、订阅者（Listener）启动，通过 RPC 向 ROC Master 注册订阅者信息，包括：订阅者节点信息，话题名等；Master 会将这些信息加入注册列表；
- 3、Master 进行节点匹配：Master 会根据订阅者提供的信息，在注册列表中查找匹配的发布者；如果没有发布者（Talker），则等待发布者（Talker）的加入；如果找到匹配的发布者

（Talker），则会主动把发布者（Talker）（有可能是很多个 Talker）的地址通过 RPC 传送给订阅者（Listener）节点；

- 4、Listener 接收到 Master 的发出的 Talker 的地址信息，尝试通过 RPC 向 Talker 发出连接请求（信息包括：话题名，消息类型以及通讯协议（TCP/UDP））；
- 5、Talker 收到 Listener 发出的连接请求后，通过 RPC 向 Listener 确认连接请求（包含的信息为自身 TCP 地址信息）；
- 6、Listener 接收到 Talker 的确认消息后，使用 TCP 尝试与 Talker 建立网络连接；
- 7、成功连接之后，Talker 开始向 Listener 发布话题消息数据；

需要注意的是：有可能多个 Talker 连接一个 Listener，也有可能是一个 Talker 连接上多个 Listener（多对多）。

1.4.2 服务通信机制——Service



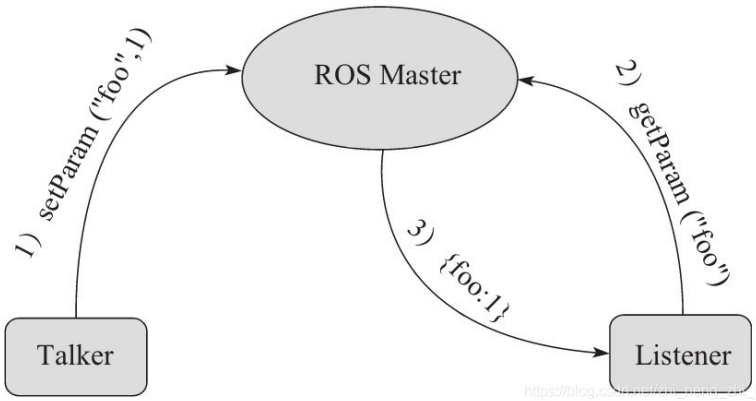
与话题的通信相比，其减少了 Listener 与 Talker 之间的 RPC 通信，建立通信的详细过程：

- 1、发布者（Talker）启动，通过 RPC 向 ROS Master 注册发布者的信息，包括：发布者节点信息，话题名，话题缓存大小等；Master 会将这些信息加入注册列表中；
- 2、订阅者（Listener）启动，通过 RPC 向 ROC Master 注册订阅者信息，包括：订阅者节点信息，话题名等；Master 会将这些信息加入注册列表；
- 3、Master 进行节点匹配：Master 会根据订阅者提供的信息，在注册列表中查找匹配的发布者；如果没有发布者（Talker），则等待发布者（Talker）的加入；如果找到匹配的发布者（Talker），则会主动把发布者（Talker）（有可能是很多个 Talker）的地址通过 RPC 传送给订阅者（Listener）节点；

- 4、Listener 接收到 Talker 的确认消息后，使用 TCP 尝试与 Talker 建立网络连接；
- 5、成功连接之后，Talker 开始向 Listener 发布话题消息数据；

需要注意的是：有可能是一个 Talker 连接上多个 Listener（一对多）。

1.4.3 参数管理机制——Parameter



参数共享机制类似于程序中的全局变量，Talker 去更新全局变量（共享的参数），Listener 去获取更新后的全局变量（共享的参数）；这个通信过程不涉及 TCP/UDP 的通信；

- 1、Talker 更新全局变量；Talker 通过 RPC 更新 ROS Master 中的共享参数（包含参数名和参数值）；
- 2、Listener 通过 RPC 向 ROS Master 发送参数查询请求（包含要查询的参数名）；
- 3、ROS Master 通过 RPC 回复 Listener 的请求（包括参数值）；

需要注意的是：如果 Listener 向实时知道共享参数的变化，需要自己不停的去询问 ROS Master；

Chapter 2

ROS 基础

2.1 turtlesim 功能包

接触的第一个 ROS 功能包：turtlesim，其核心是 tuetlesim_node 节点。

其中包含的话题和服务如下：

	名称	类型	描述
话题订阅	turtleX/cmd_vel	geometry_msgs/Twist	控制乌龟角速度与线速度的输入指令
话题发布	turtleX/pose	turtlesim/Pose	乌龟的姿态信息：包括 x 与 y 坐标、角度、线速度和角速度
服务	clear	std_srvs/Empty	清楚仿真器中的背景颜色
	reset	std_srvs/Empty	复位仿真器到初始状态
	kill	turtlesim/Kill	删除一只乌龟
	spawn	turtlesim/Spawn	新生一只乌龟
	turtleX/set_pen	turtlesim/Setpen	设置画笔的颜色和线宽
	turtleX/teleport_absolute	turtlesim/TeleportAbsolute	移动乌龟到指定的姿态
	turtleX/teleport_realative	turtlesim/TeleportRealative	移动乌龟到指定的角度和距离

2.2 创建工作空间和功能包

2.2.1 创建工作空间

工作空间初始化：

```
mkdir ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

初始化后，可以编译整个工作空间：

```
cd ~/catkin_ws/
catkin_make
```

编译后，在工作空间的根目录下会产生 build 和 devel 两个文件夹，在 devel 文件夹中有 setup.bash 形式的环境变量设置脚本，则可以使用 source 命令运行这些脚本配置环境变量，如：

```
source devel/setup.bash
```

但是 source 命令设置的环境变量只在当前终端中有效，所以为了方便，可以讲终端的配置文件（ /.bashrc）中加入上面的环境变量的配置语句（要注意写全绝对路径）。

2.2.2 创建功能包

创建功能包的命令如下：

```
cd ~/catkin_ws/src
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

创建完成后，工作空间的 src 目录中会生成一个 <package_name> 的功能包，并且已经包含了 package.xml 和 CMakelist.txt 文件。其中 package.xml 文件提供描述功能包属性的信息，CMakelist.txt 文件记录功能包的编译规则。

进而可以回到工作空间的根目录下进行编译，并设置环境变量。

2.3 工作空间的覆盖

所有工作空间的路径会依次在 ROS_PACKAGE_PATH 环境变量中记录，当设置多个工作空间的环境变量后，新设置的路径在 ROS_PACKAGE_PATH 中会自动放在最前端。在运行时，ROS 会优先查找最前端的工作空间中是否存在指定的功能包，如果不存在，就顺序向后查找其他工作空间，知道最后一个工作空间为止。

2.4 Topic 中的 Publisher 和 Subscriber

2.4.1 Publisher 的创建

```
#include <sstream>
#include "ros/ros.h"
#include "std_msgs/String.h"
int main(int argc, char **argv){
    // ROS节点初始化
    ros::init(argc, argv, "talker");
    // 创建节点句柄
    ros::NodeHandle n;
    // 创建一个Publisher，发布名为chatter的topic，消息类型为std_msgs::String
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
    // 设置循环的频率
    ros::Rate loop_rate(10);
    int count = 0;
    // 一旦发生异常，ros::ok()就会返回false，跳出循环
    while (ros::ok()){
        // 初始化std_msgs::String类型的消息
        std_msgs::String msg;
        std::stringstream ss;
        ss << "hello world " << count;
        msg.data = ss.str();
        // 发布消息
        ROS_INFO("%s", msg.data.c_str());
        chatter_pub.publish(msg);
        // 循环等待回调函数
        // ros::spinOnce()函数用来处理节点订阅话题的所有回调函数
        // 虽然目前的发布节点并没有任何订阅信息，ros::spinOnce()不是必须的
        // 但是为了保证功能无误，建议所有节点都默认加入该函数
        ros::spinOnce();
        // 按照循环频率延时
        loop_rate.sleep();
        ++count;
    }
    return 0;
```

```
}
```

2.4.2 Subscriber 的创建

```
#include "ros/ros.h"
#include "std_msgs/String.h"
// 接收到订阅的消息后，会进入消息回调函数
// 当有消息到达时，会自动以消息指针作为参数
void chatterCallback(const std_msgs::String::ConstPtr& msg){
    // 将接收到的消息打印出来
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}
int main(int argc, char **argv){
    // 初始化ROS节点
    ros::init(argc, argv, "listener");
    // 创建节点句柄
    ros::NodeHandle n;
    // 创建一个Subscriber，订阅名为chatter的topic，注册回调函数chatterCallback
    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
    // 循环等待回调函数
    ros::spin();
    return 0;
}
```

2.4.3 自定义话题消息

编写 msg 文件

使用 msg 文件定义自己的消息类型，一般放置在功能包根目录下的 msg 文件夹中。msg 文件中既可以定义消息类型的变量，也可以定义常量：

```
string name
uint8  sex
uint8  age

uint8 unknown = 0
uint8 male    = 1
uint8 female  = 2
```

对于稍复杂一些的 ROS 自定义消息，还会包含一个标准格式的头信息 std_msgs/Header:

```
unint32 seq
time stamp
string frame_id
```

其中：seq 是消息的顺序标识，不需要手动设置，Publisher 在发布消息时会自动累加；stamp 是消息中与数据相关联的时间戳，可以用于时间同步；frame_id 是消息中与数据相关联的参考坐标系 id。

编译 msg 文件

（1）在 package.xml 中添加功能包依赖

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

（2）在 CMakeLists.txt 文件中添加编译选项
在 find_package 中添加消息生成依赖的功能包 message_generation：

```
find_package(catkin REQUIRED COMPONENTS
    geometry_msgs
    roscpp
    rospy
    std_msgs
    message_generation
)
```

设置 catkin 依赖：

```
catkin_package(
    # INCLUDE_DIRS include
    # LIBRARIES learning_communication
    CATKIN_DEPENDS geometry_msgs roscpp rospy std_msgs message_runtime
    # DEPENDS system_lib
)
```

设置需要编译的 msg 文件：

```
add_message_files(FILES Person.msg)
generate_messages(DEPENDENCIES std_msgs)
```

然后对功能包进行编译，自定义的消息类型就生效了。

2.4.4 CMakeLists 的编写

几个常用的编译选项：

（1）include_directories

用于设置头文件的相对路径。功能包的一些头文件会放在功能包根目录下的 include 文件夹中，所以需要添加该文件夹。

（2）add_executable

用于设置需要编译的代码和生成的可执行文件。第一个参数为期望生成的可执行文件的名称，后面的参数为参与的源码文件 (cpp)，如果需要多个代码文件，可以在后面依次列出，中间用空格分隔。

（3）target_link_libraries

用于设置链接库。第一个参数为期望生成的可执行文件的名称，后面依次列出需要链接的库，如果没有使用其他库，添加默认链接库（${catkin_LIBRARIES}）即可。

（4）add_dependencies

用于设置依赖。在很多应用中，我们需要定义语言无关的消息类型，消息类型会在编译过程中产生相应语言的代码，如果编译的可执行文件依赖这些动态生成的代码，则需要使用 add_dependencies 添加 ${PROJECT_NAME}_generate_messages_cpp 配置，即该功能包动态产生的消息代码。

对于我们的这个例子，CMakeLists.txt 文件如下：

```
include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker ${PROJECT_NAME}_generate_messages_cpp)

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(talker ${PROJECT_NAME}_generate_messages_cpp)
```

2.5 Service 中的 Client 和 Server

2.5.1 创建 Client

```
#include <cstdlib>
#include "ros/ros.h"
#include "learning_communication/AddTwoInts.h"
```

```
int main(int argc, char **argv){
    ros::init(argc, argv, "add_two_ints_client");
    // 从终端命令行获取两个加数，argv[0]是路径，argv[1]和[2]是两个输入参数
    if (argc != 3){
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }
    ros::NodeHandle n;
    // 创建一个client，请求add_two_int service
    // service消息类型是learning_communication::AddTwoInts
    ros::ServiceClient client = n.serviceClient\
        <learning_communication::AddTwoInts>("add_two_ints");
    // 创建learning_communication::AddTwoInts类型的service消息
    // 该变量包含两个成员：request和response
    learning_communication::AddTwoInts srv;
    // atoll()函数将字符串转化为整数
    srv.request.a = atoll(argv[1]);
    srv.request.b = atoll(argv[2]);
    // 发布service请求，等待加法运算的应答结果
    // 调用过程会发生阻塞，调用成功后返回true
    if (client.call(srv)){
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    }
    else{
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }
    return 0;
}
```

2.5.2 创建 server

```
#include "ros/ros.h"
#include "learning_communication/AddTwoInts.h"
// service回调函数，输入参数req，输出参数res
bool add(learning_communication::AddTwoInts::Request  &req,
         learning_communication::AddTwoInts::Response &res){
```

```
    // 将输入参数中的请求数据相加，结果放到应答变量中
    res.sum = req.a + req.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
    ROS_INFO("sending back response: [%ld]", (long int)res.sum);
    return true;
}
int main(int argc, char **argv){
    ros::init(argc, argv, "add_two_ints_server");
    ros::NodeHandle n;
    // 创建一个名为add_two_ints的server，注册回调函数add()
    ros::ServiceServer service = n.advertiseService("add_two_ints", add);
    // 循环等待回调函数
    ROS_INFO("Ready to add two ints.");
    ros::spin();
    return 0;
}
```

2.5.3 自定义服务数据

编写 srv 文件

使用 srv 文件定义自己的消息类型，一般放置在功能包根目录下的 srv 文件夹中。该文件包含 request 和 response 两个数据域，两个数据域之间用”—”（三个减号）分隔，如：

```
int64 a
int64 b
---
int64 sum
```

编译 srv 文件

（1）在 package.xml 中添加功能包依赖（与自定义话题消息相同）

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

（2）在 CMakeLists.txt 文件中添加编译选项

与自定义话题消息相同也是添加 message_generation 包，

```
find_package(catkin REQUIRED COMPONENTS
```

```
    geometry_msgs
    roscpp
    rospy
    std_msgs
    message_generation
)
add_service_files(FILES AddTwoInts.srv)
```

2.5.4 CMakeLists 的编写

与 Topic 类似:

```
include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(server src/server.cpp)
target_link_libraries(server ${catkin_LIBRARIES})
add_dependencies(server ${PROJECT_NAME}_gencpp)

add_executable(client src/client.cpp)
target_link_libraries(client ${catkin_LIBRARIES})
add_dependencies(client ${PROJECT_NAME}_gencpp)
```