**1 ROS初识**

**1.1 在终端中实现程序**

（1）创建工作空间

主目录打开终端：mkdir -p 工作空间名称\_ws/src

进入工作空间：cd 工作空间名称\_ws

编译工作空间：catkin\_make

（2）创建功能包

功能包名称加上依赖

catkin\_create\_pkg helloworld roscpp rospy std\_msgs

（3）编辑源文件

进入功能包中的src文件夹中，新建cpp文档：gedit helloworld.cpp

（4）编辑配置文件

编辑功能包下的配置文件cmakelist.txt文件：

add\_executable和target\_link\_libraries取消注释

然后修改其中的源文件名称

（5）编译并执行

进入工作空间并编译：catkin\_make

新开一个终端启动 ros 核心：roscore

source ./devel/setup.bash

# 这么写的source只能在该窗口下执行，新开终端则无法执行程序

rosrun 功能包名 c++节点

# 改变source局限性，在任意窗口下都可以执行：在主目录中的./bashrc文件中最后一行添加hellowoorld的source

**1.2 集成开发环境的搭建**

**1.3 launch文件的应用**

可以一次性启动多个节点

1、在功能包中新建launch文件夹

2、在launch文件夹中新建launch文件

3、编辑launch文件：声明需要启动的节点

在终端中启动乌龟：

需要三个单独终端

roscore

rosrun turtlesim turtlesim\_node

rosrun turtlesim turtle\_teleop\_key

在vscode中使用launch文件启动乌龟：

<launch>

<!-- 添加被执行的节点 -->

<!-- 乌龟GUI -->

<node pkg="turtlesim" type="turtlesim\_node" name="turtle" />

<node pkg="turtlesim" type="turtle\_teleop\_key" name="turtle\_key" />

</launch>

pkg：turtlesim

type：turtlesim\_node

name：给节点定义名称

4、运行launch文件

格式固定：roslaunch 包名 launch文件名

launch语法：roslaunch package\_name file\_name.launch

（1）、node：启动节点

<node pkg="package-name" type="execulate-name" name="node-name"/>

pkg ：功能包名称

type：节点可执行文件名称

name：节点运行时的名称

（2）、param：设置ROS系统运行参数，存储在参数服务器中

<param name="output\_name"   value="odom"/>

name：参数名

value：参数值

加载参数文件中的多个参数

<rosparam file="params.yaml" command="load"ns="params"/>

例：<rosparam file="$(find learning\_launch)/config/param.yaml" command="load"/>

（3）、arg：launch文件内部的局部变量，仅仅在launch文件中使用

<arg name="arg\_name" default="arg\_value"/>

name：参数名

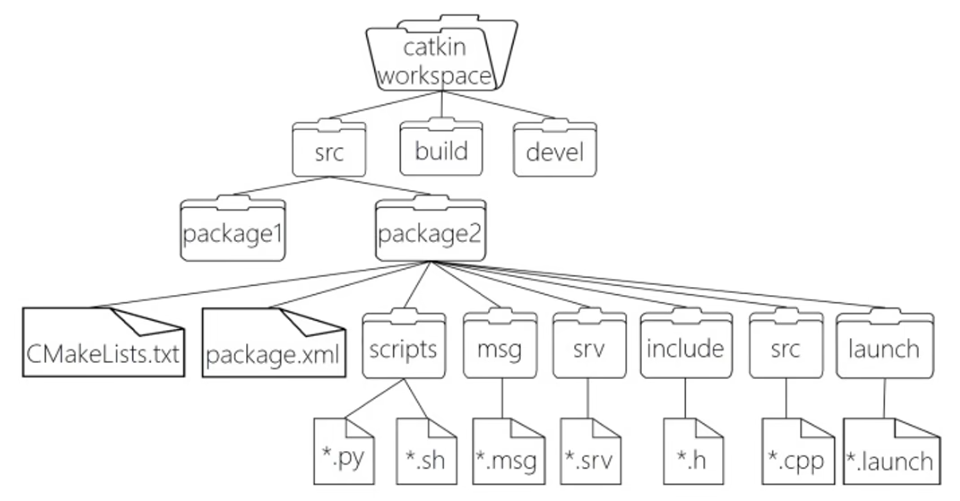
value：参数值

<param name="foo" value="$(arg arg\_name)"/>

<node name="node pkg="package-name" type="type" arg="$(arg arg\_name)"/>

**1.4 ROS架构**

**1.5 ROS文件系统**

build：编译空间

devel：开发空间

src：源码

功能包：ROS基本单元，包含多个节点、库、配置文件。包名只能用小写字母数字下划线组成。

Package.xml：自动生成的，位于功能包下，内容包含软件包的属性，例如软件包名称、作者、版本、维护者、依赖。

CmakeLists.txt：

**1.6 ROS命令**

roscore：启动核心

rosrun：运行指定ROS节点

rosrun 包名 可执行文件名

roslaunch：执行包下的launch文件

roslaunch 包名 launch文件名

catkin\_create\_pkg 自定义包名 依赖包：创建新的ROS功能包

sudo apt install xxx：安装ROS功能包

sudo apt purge xxx：删除某个功能包

rospack list：列出所有功能包

rospack find：查找某个功能包是否存在，如果存在，返回功能包所在路径

roscd 包名：进入某个功能包

rosls 包名：列出某个包下文件

apt search xxx：搜索某个功能包

rosed 包名 文件名：修改功能包文件

**2、通讯机制**

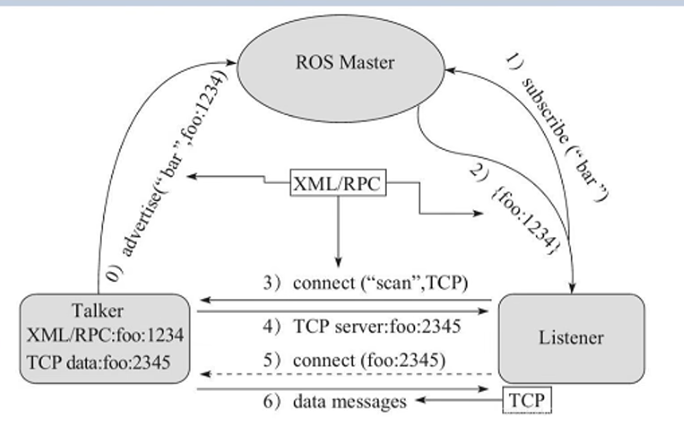
话题通信（发布订阅模式）

服务通信（请求响应模式）

参数服务器（参数共享模式）

**2.1 话题通信（发布订阅模式）**

发布方通过话题与订阅方相连接，

**角色：**

master：管理者

talker：发布者

listener：订阅者

**流程：**

master根据话题建立发布者与订阅者之间的链接。

0. （注册操作）发布者向管理者提交自身信息，包括话题RPC地址

1. （注册操作）订阅者向管理者提交自身信息，包括话题

2. 管理者比对两者的话题，如果一致，将发布者的RPC地址发送给订阅者

3. 订阅者通过RPC地址远程访问发布者

4. 发布者给订阅者响应，响应内容包含TPC地址

5. 订阅者通过TPC地址访问发布者

6. 发布者给订阅者发送消息

注意：

步骤0和步骤1没有先后顺序

发布者和订阅者都可以有多个

发布者和订阅者建立联系后，管理者可以关闭了

**2.1.1 实现流程（案例一）**

**1. 编写发布方实现**

#include "ros/ros.h"

#include "std\_msgs/String.h"

// 发布方实现

// 1、包含头文件

// ROS中文本类型--> std\_msgs/String.h

// 2、初始化ROS节点

// 3、创建节点句柄

// 4、创建发布者对象

// 5、编写发布者逻辑并发布数据

int main(int argc, char \*argv[])

{

// 2、初始化ROS节点,节点名称pub01

ros::init(argc,argv,"pub01");

// 3、创建节点句柄

ros::NodeHandle nh;

// 4、创建发布者对象

ros::Publisher pub = nh.advertise<std\_msgs::String>("topic01",10);

// 5、编写发布者逻辑并发布数据

//先创建发布的消息

std\_msgs::String msg;

//编写循环，循环中发布数据

while (ros::ok())

{

msg.data = "hello";

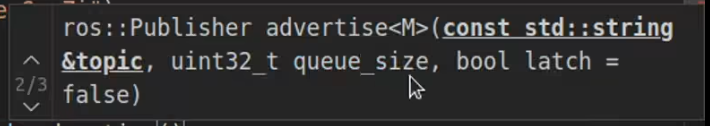
pub.publish(msg);

}

return 0;

}

ps：创建发布者对象的函数advertise，在此使用的是第二个重载函数

M：一会发布的数据的类型

std::string&topic：字符串类型，话题名称

queue\_size：消息长度，存放由于网络堵塞未发布成功的消息数量，如果堵塞的消息过多，会舍弃前面超出数量的消息。

**2. 编写订阅方实现**

#include "ros/ros.h"

#include "std\_msgs/String.h"

// 订阅方实现

// 1、包含头文件

// ROS中文本类型--> std\_msgs/String.h

// 2、初始化ROS节点

// 3、创建节点句柄

// 4、创建订阅者对象

// 5、编写发布者逻辑并发布数据

// 6、声明spin()函数

void doMsg(const std\_msgs::String::ConstPtr &msg){

// 通过msg获取并操作订阅到的数据

ROS\_INFO("订阅的数据：%s",msg->data.c\_str());

}

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL,"");

// 2、初始化ROS节点，节点名称保证唯一性

ros::init(argc,argv,"sub01");

// 3、创建节点句柄

ros::NodeHandle nh;

// 4、创建订阅者对象，话题名称要与发布方一致

ros::Subscriber sub = nh.subscribe("topic01",10,doMsg);

// 5、编写发布者逻辑并发布数据

ros::spin();

/\* code \*/

return 0;

}

**3. 编辑配置文件**

在功能包下 的CmakeList.txt文件中，第136行左右，发布方和订阅方都需要编辑。

# add\_executable(${PROJECT\_NAME}\_node src/talker\_listener\_node.cpp)

add\_executable(demo01\_pub src/demo01\_pub.cpp)

add\_executable(demo02\_sub src/demo02\_sub.cpp)

以及

target\_link\_libraries(demo01\_pub

${catkin\_LIBRARIES}

)

target\_link\_libraries(demo02\_sub

${catkin\_LIBRARIES}

)

**4. 编译并执行**

执行时，先source，两个终端窗口都要

:~/aslam\_ws$ rosrun talker\_listener demo01\_pub

:~/aslam\_ws$ rosrun talker\_listener demo02\_sub

**2.1.2 话题通信自定义msg（案例二）**

流程：

a. 按照固定格式创建msg文件

b. 编辑配置文件

c. 编译生成可以被C++调用的中间文件

定义msg文件

功能包下新建msg文件夹，msg文件夹中新建 .msg 文件

string name

int32 age

float32 height

编辑配置文件

功能包下的package.xml 文件中添加（50+行）

<build\_depend>message\_generation</build\_depend>

<exec\_depend>message\_runtime</exec\_depend>

功能包下的CmakeList.txt 文件中添加

10：find\_package中添加message\_generation；

51：add\_message\_files 中将message文件修改为自己建立的message文件的名称，意味着编译的是add中的msg文件；

71：将generation\_messages 注释放开，不需要修改，这一部分意思是编译需要靠std\_msgs；

105：catkin\_package中放开CATKIN这一行，并添加依赖 message\_runtime，CATKIN中的依赖包必须依赖第10行find\_package中的依赖包

编译

编译成功后会生成中间文件

aslam\_ws/devel/include/person.h：C++使用的头文件

aslam\_ws/devel/lib/python2.7/msg/\_person.py：python使用的头文件

**2.1.3 话题通讯自定义msg（C++）**

**流程：**

1. 编写发布方实现

2. 编写订阅方实现

3. 编辑配置文件

4. 编译并执行

准备工作：在 .vscode 文件夹中的 c\_cpp\_properties.json 文件中添加头文件路径，配置vscode，以免运行时报错

**1. 发布方实现**

#include "ros/ros.h"

#include "talker\_listener/person.h"

// 发布方消息

// 1 包含头文件

// #include "talker\_listener/person.h"

// 2 初始化ROS节点

// 3 创建ROS节点句柄

// 4 创建发布者对象

// 5 编写发布逻辑，发布数据

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL,"");

ROS\_INFO("这是消息发布方");

// 2 初始化ROS节点

ros::init(argc,argv,"publisher01");

// 3 创建节点句柄

ros::NodeHandle nh;

// 4 创建发布者对象

ros::Publisher pub = nh.advertise<talker\_listener::person>("liaotian",10);

// 5 编写发布逻辑，发布数据

// 5.1 创建被发布的数据

talker\_listener::person person;

person.name = "张三";

person.age = 1;

person.height = 1.8;

// 5.2 设置发布频率

ros::Rate rate(1);

// 5.3 循环发布数据

while(ros::ok()){

// 修改数据

person.age += 1;

// 核心：发布数据

pub.publish(person);

// 休眠

rate.sleep();

// 建议

ros::spinOnce();

}

/\* code \*/

return 0;

}

**2. 编写订阅方实现**

订阅方：订阅消息

1. 包含头文件；

# include “plumbing\_pub\_sub/

2. 初始化ROS节点；

3. 创建节点句柄；

4. 创建订阅者对象；

5. 处理订阅的数据；

6. 调用spin（）函数；

**3. 编辑配置文件**

发布方：CmakeList.txt文件中

add\_executable(demo03\_pub\_person src/demo03\_pub\_person.cpp)

// 在target上两行，当编辑好msg文件后没有进行编译，然后编写cpp文件后一起编译的话可能会出现，先编译cpp文件，后编译msg文件的情况，此时编译和运行会出错，因此需要在此添加${PROJECT\_NAME}\_generate\_message\_cpp，然后一起编译和执行时不会出错。

add\_dependencies(demo03\_pub\_person ${PROJECT\_NAME}\_generate\_messages\_cpp)

target\_link\_libraries(demo03\_pub\_person

${catkin\_LIBRARIES}

)

**4. 编译并执行**

**2.2 服务通信**

**2.3 参数服务器**

d

**2.4 常用命令**