ROS使用流程

[SSH登录 3](#_Toc156571770)

[创建工作空间 4](#_Toc156571771)

[利用catkin工具创建功能包 5](#_Toc156571772)

[创建话题与发布话题C++ 7](#_Toc156571773)

[创建话题与发布话题Python 9](#_Toc156571774)

[订阅话题及回调处理 11](#_Toc156571775)

[利用launch文件批量运行节点 13](#_Toc156571776)

[ROS辅助工具使用 15](#_Toc156571777)

[ROS自定义消息包创建及添加新依赖包 17](#_Toc156571778)

[SLAM算法 24](#_Toc156571779)

[TF系统 26](#_Toc156571780)

[如何发布TF变换关系： 29](#_Toc156571781)

[里程计 33](#_Toc156571782)

[旋转（欧拉角与四元数） 34](#_Toc156571783)

[串口别名修改 36](#_Toc156571784)

[Gazebo地图构建 38](#_Toc156571785)

[机器人模型构建 40](#_Toc156571786)

[保存SLAM所建的地图 44](#_Toc156571787)

[ROS与STM32通信 46](#_Toc156571788)

[ROS图像与OpenCv图像转换 47](#_Toc156571789)

[脚本实现ROS程序开机自启动 49](#_Toc156571790)

[Rosbag记录/回放rostopic数据 49](#_Toc156571791)

[机器人位姿融合Robot\_pos\_ekf 50](#_Toc156571792)

[SLAM建图--Hector\_Mapping 52](#_Toc156571793)

[SLAM建图—Gmapping 53](#_Toc156571794)

SSH登录

* 简述：
  + SSH(Secure shell)是一种网络协议，用于计算机之间的加密登录。当两台电脑处于同一网段时，可以通过ssh来进行访问。
* 安装：

命令行安装：Sudo apt-get install openssh\*

创建工作空间

* 启动ROS：

Terminal中输入： “roscore”

* 在合适目录创建工作空间：

cd “你想创建工作空间文件夹的目录”

mkdir “工作空间名字” （创建一个工作空间文件夹）

cd “工作空间名字”

mkdir “src” （创建一个名为src的文件夹）

catkin\_make (编译“工作空间”文件夹)

source devel/setup.bash (设置环境变量)

echo $ROS\_PACKAGE\_PATH (检查环境变量)

catkin\_make install （创建install文件夹）

！至此，完整的工作空间已经建立！

你将获得以下目录结构的工作空间：

“工作空间名称”

Src （代码空间Source Space）

Build （编译空间 Bulid Space）

Devel （开发空间 Development Space）

Install （安装空间 Install Space）

！所有功能包均放置在工作空间的src文件夹中！

利用catkin工具创建功能包

* 使用catkin工具创建你的第一个功能包！
  + 进入工作空间中的src文件夹
  + “catkin\_create\_pkg <PackageName> <Depend1、2、3>”

例子: (catkin\_create\_pkg myWorkspace roscpp rospy std\_msgs)

其中myWorkspace—>功能包名字；roscpp/py,std\_msgs为依赖

功能包名字规范：1.全为小写字母；2.使用下划线连接

* 编译功能包：

Cd“工作空间目录”

“catkin\_make”

* 运行你的第一个节点：

打开终端，启动ROS： roscore

启动节点： rosrun “功能包名字” “节点名字”

若编译报错（显示无法寻找到package），则输入命令：source ~/“工作空间名字”/devel/setup.bash

完成！

* 完善第一个节点：

在代码中加入：(C++): ros::init(<参数1…2…>，“节点名字”)

(python): rospy.init\_node(“节点名字”)

若想实现键入 Ctrl+C 退出，则将循环条件改为：while(ros::ok())

此时大多会出现无法找到ros::init错误（“undefined reference to ros init ……”），此时需要将该函数库文件一块链接进来进行编译。方法：进入CMakeLists.txt 找到注释中build 章节，将Specify libraries下面三行复制到CMakeLists.txt 中的最后一行（注意将前面井号去掉， 将“$）{PROJECT\_NAME}”改为自己节点名称）。保存再进行编译即可。

* 再次打开终端，运行节点

启动节点： rosrun “功能包名字” “节点名字”。

注意，尽量不要将功能包名字和节点名字命名的一模一样，不然后续使用时容易起文件冲突。特别是py文件

创建话题与发布话题C++

* 创建话题与发布话题：

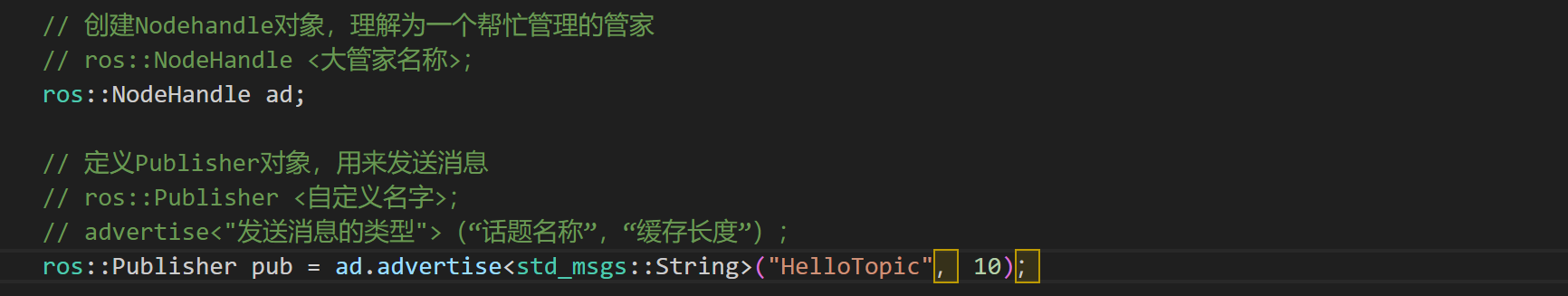
确定话题名称和消息类型，进入index.ros.org搜索std\_msgs ，找到使用的ros版本看里面有哪些类型

在代码中include消息类型对应的头文件：

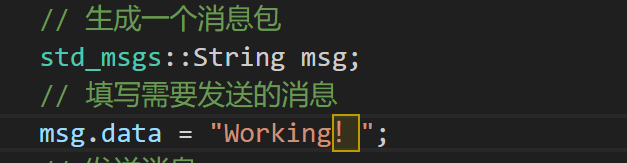
例如：#include <std\_msgs/String.h>

在main函数中通过 NodeHandle 发布一个话题：

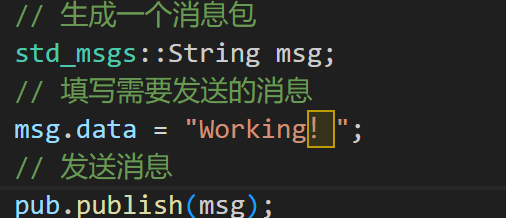
例：C++：



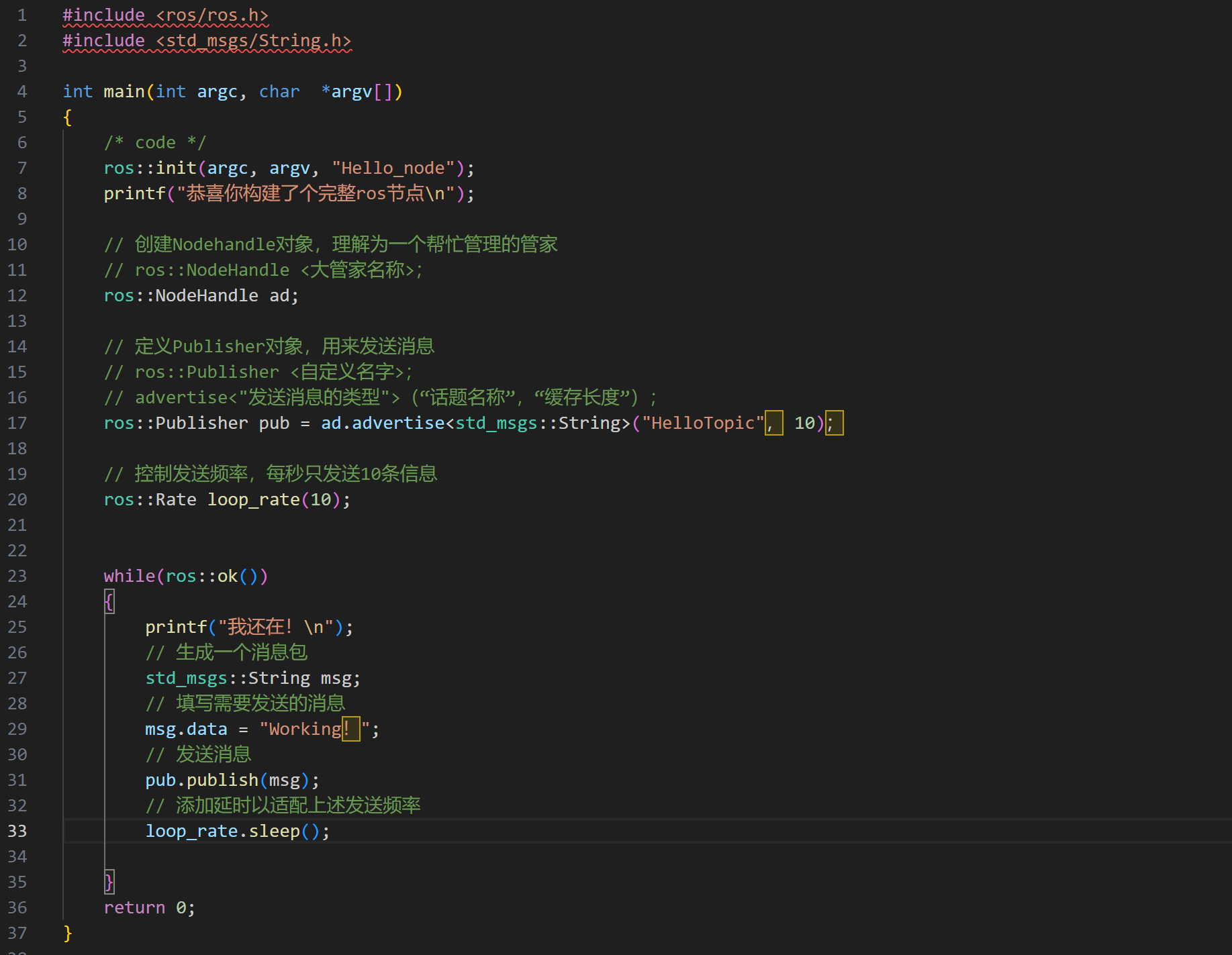
在消息发送前生成消息包并发送数据：



调用消息发送对象的publish（）函数将消息包发送至话题中：



完整代码（里边附带控制消息包发送频率方法）：



创建话题与发布话题Python

* 创建话题与发布话题：

确定话题名称和消息类型，进入index.ros.org搜索std\_msgs ，找到使用的ros版本看里面有哪些类型

在代码第一行指定python编译器和编码格式：

Line1：#!/usr/bin/env python

Line2：#coding=utf-8

（PS： Line 1 可能会报错，详情查看报错解决方案）

在代码中import消息类型对应的头文件：

例如：import rospy

from std\_msgs.msg import String

通过 rospy发布一个话题：

Pub = rospy.Publisher(<”话题名称”>, <话题中消息包的数据类型>,<缓存长度>)

在消息发送前生成消息包并发送数据：

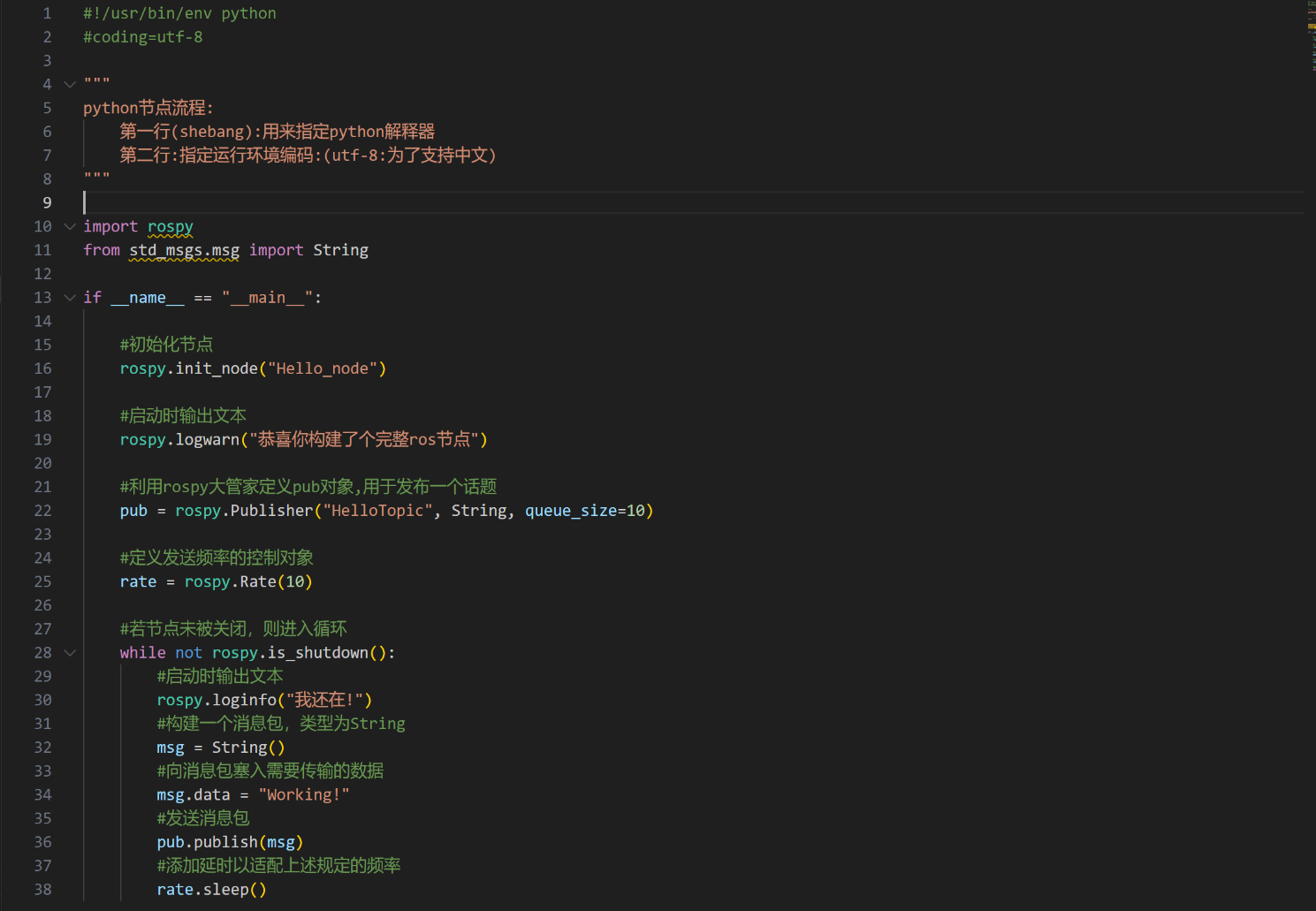
msg=String()

msg.data = “内容”

调用消息发送对象的publish（）函数将消息包发送至话题中：

Pub.pulish(msg)

完整代码（里边附带控制消息包发送频率方法）：



Ubuntu中问了安全考虑，该py节点不带有执行文件，现在要为它创建可执行为文件：

Command： cd节点所在目录

sudo chmod +x <节点名称>

订阅话题及回调处理

* 订阅话题及回调处理：

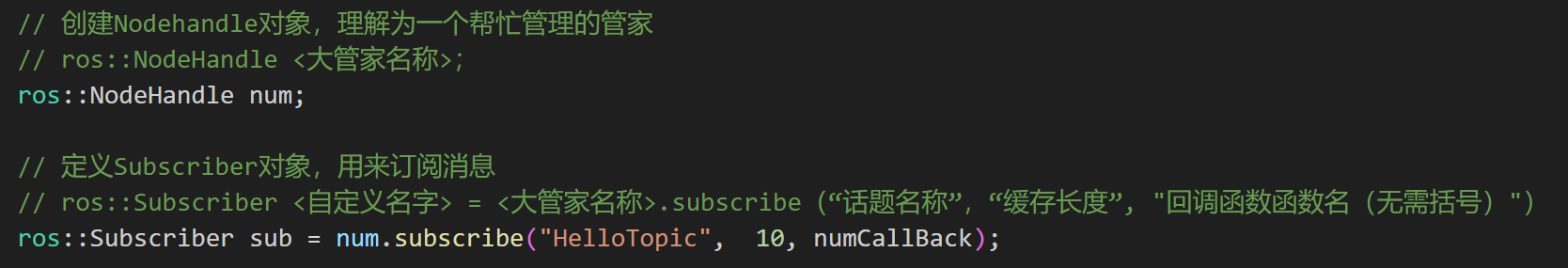
确定话题名称和消息类型，进入index.ros.org搜索std\_msgs ，找到使用的ros版本看里面有哪些类型

在代码中include消息类型对应的头文件：

例如：#include <std\_msgs/String.h>

在main函数中通过 NodeHandle 订阅一个话题并设置消息包的回调函数：

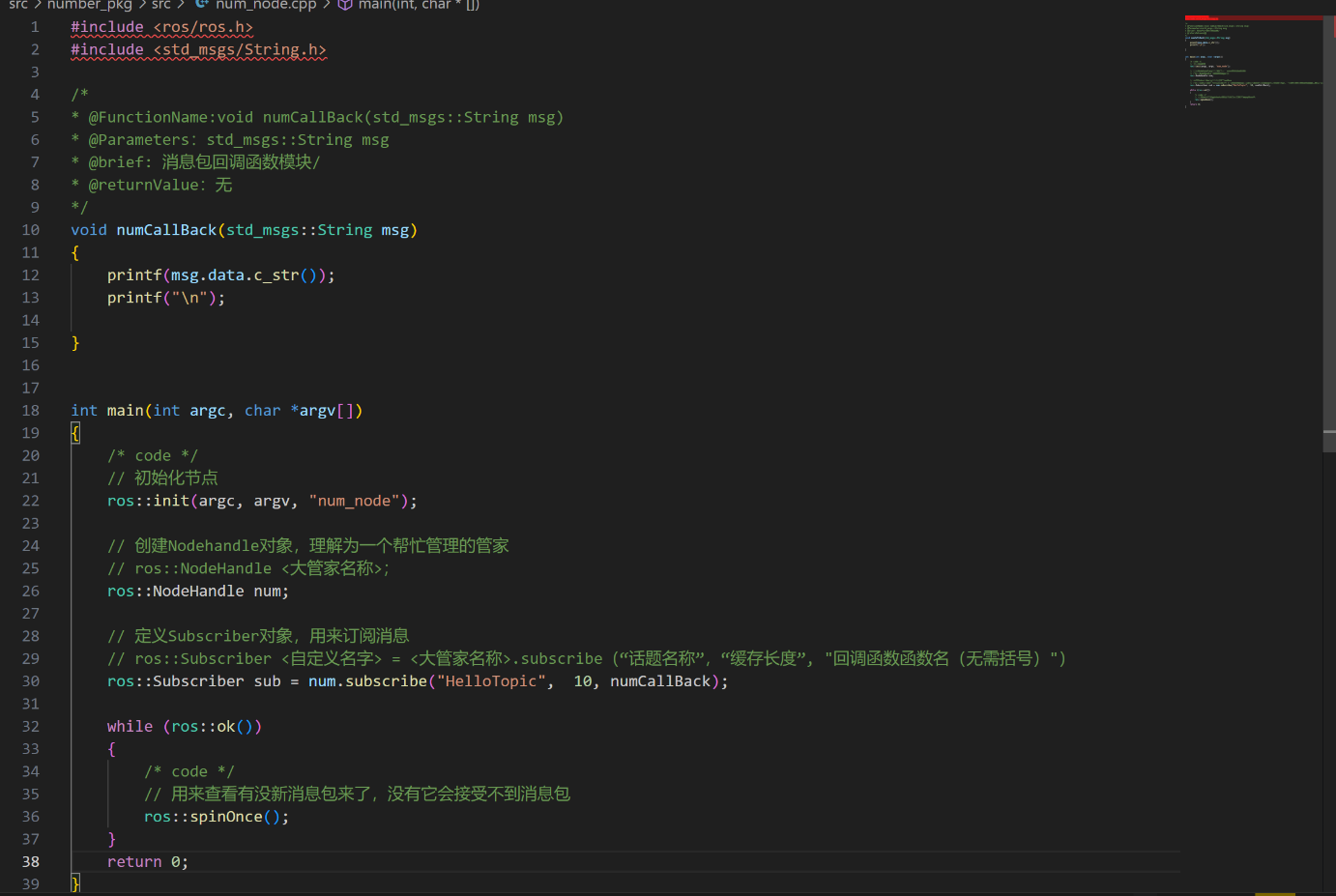
例：C++：



定义一个回调函数，对接收到的消息包进行处理

Main函数中加入ros::SpinOnce()函数，让回调函数能响应接收到的消息包。（没有它会接收不到消息包）

完整代码：



利用launch文件批量运行节点

* 利用launch文件批量运行节点：

lanunch文件概念：

其是一种遵循XML语法的描述文件。

XML语法：（带上“/”说明描述结束）

例子：<大纸盒>

<1号小纸盒 材质=“” 颜色=“”/>

<2号小纸盒 材质=“” 颜色=“”/>

<3号小纸盒 材质=“” 颜色=“”/>

………………………………………

</大纸盒>

进入任意功能包目录，创建一个launch目录:

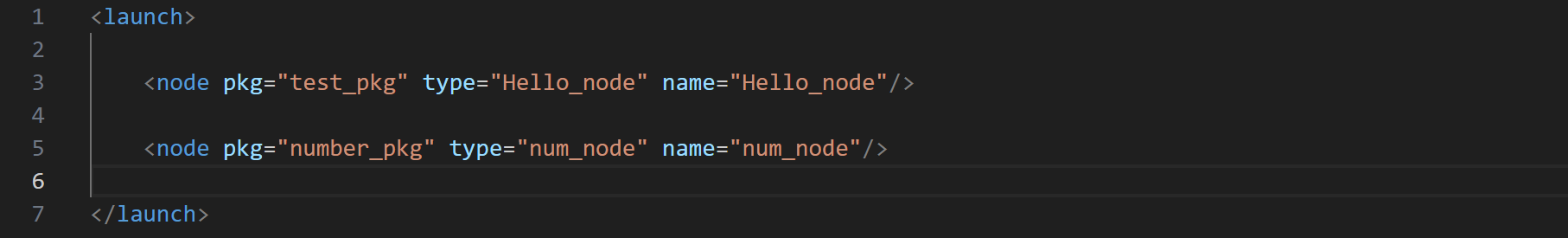
命令：cd <任意功能包>

mkdir launch

为lunch目录创建launch文件：

命令：cd launch

touch <自定义名字>.launch

开始描述，示例：

启动launch文件：

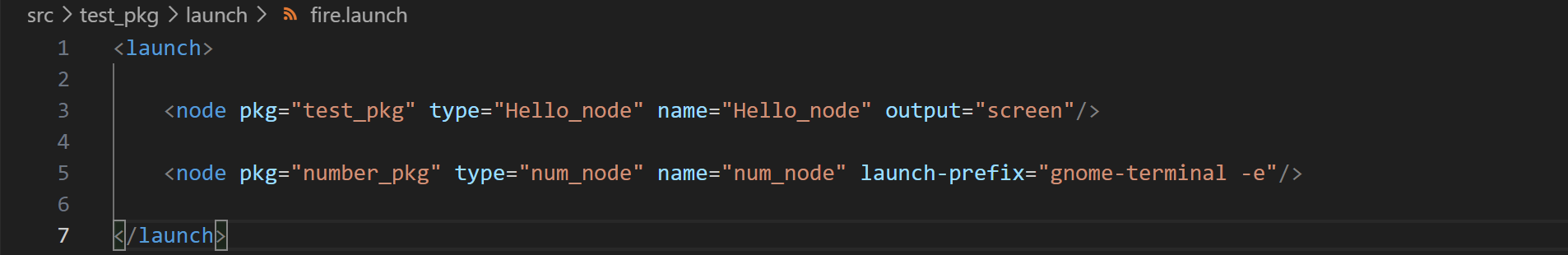
roslaunch <launch所存储在的功能包名字> <launch文件名>

！！！！！！至此完成！！！！！！

在launch文件中，为节点添加 output=”screen”属性，可以让节点信息输出在终端中。（ROS\_WARN不受该属性控制）

在launch文件中，为节点添加 launch-prefix=“gnome-terminal -e”属性，可以让节点单独运行在一个独立终端中

示例：

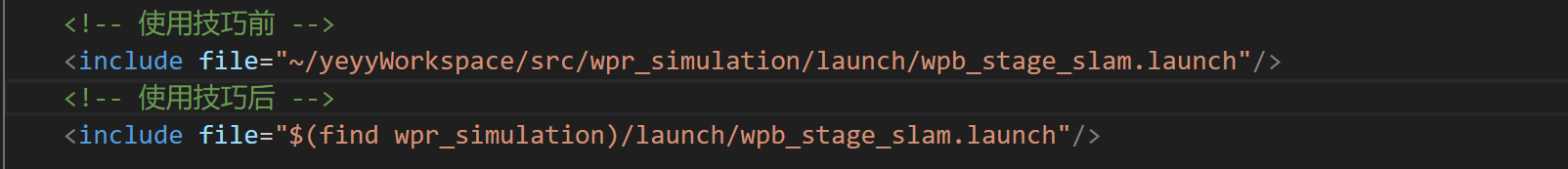


* 在launch文件中运行其它launch文件：

格式：<include file=”<launch文件具体路径>/”

小技巧：通过rospack指令来获取软件包的完整路径

* + 格式：$(find <软件包名称>/<接launch文件具体路径>)



利用ANSI 转义序列设置文本颜色

* ANSI转义序列是一种在终端或命令行界面中输出特定控制字符的方法。这些序列通常以ESC（Escape，ASCII码27）开始，后跟一个或多个字符。
* 本例程用其控制输出文本的颜色，适用于ROS\_INFO、ROS\_INFO\_STREAM、ROS\_WARN、ROS\_ERROR 等 ROS 的日志函数，以及普通的 C++ 或 Python 程序中的打印语句。
* ROS端：

默认(黄色)：ROS\_INFO\_STREAM(“HelloWorld!”)

ANSI：ROS\_INFO\_STREAM(“\033[1:32mHelloWorld!\033[0m”)

* 参数解释：

“**\033**”表示转义序列的开始。

**“[1;32m**”表示设置文本属性的部分。在这个序列中，**1** 表示粗体（即加粗），**32** 表示绿色。

**“\033[0m”**：表示重置文本样式和颜色的部分。

* 常见颜色代号：

30~37：黑/红/绿/黄/蓝/洋红/青/白

ROS辅助工具使用

* Gazebo (三维物理仿真平台):

下载源码包（功能包）：

Step1：进入github，搜索 wpr\_simulation

Step2：复制其HTTPS地址

Step3：终端cd <工作空间><src目录>

Step4：下载源码包：git clone <HTTPS地址>

Step5：cd wpr\_simulation/scripts/

Step6：./install\_for\_<ROS系统的版本>\_.sh

Step7：等待其自动下载和安装编译需要的依赖项的完成

Step:8：回到工作空间目录进行编译（catkin\_make）

Step9：启动软件：

roslaunch wpr\_simulation wpb\_simple.launch

* Rviz (三维可视化工具，把已有的数据可视化显示)

无需下载，ROS系统自带

终端输入 rviz 即可启动

若想保存当前数据设置：

Step1：点击右上角file按键

Step2：选择 Save config as

Step3：选个目录保存该文件即可

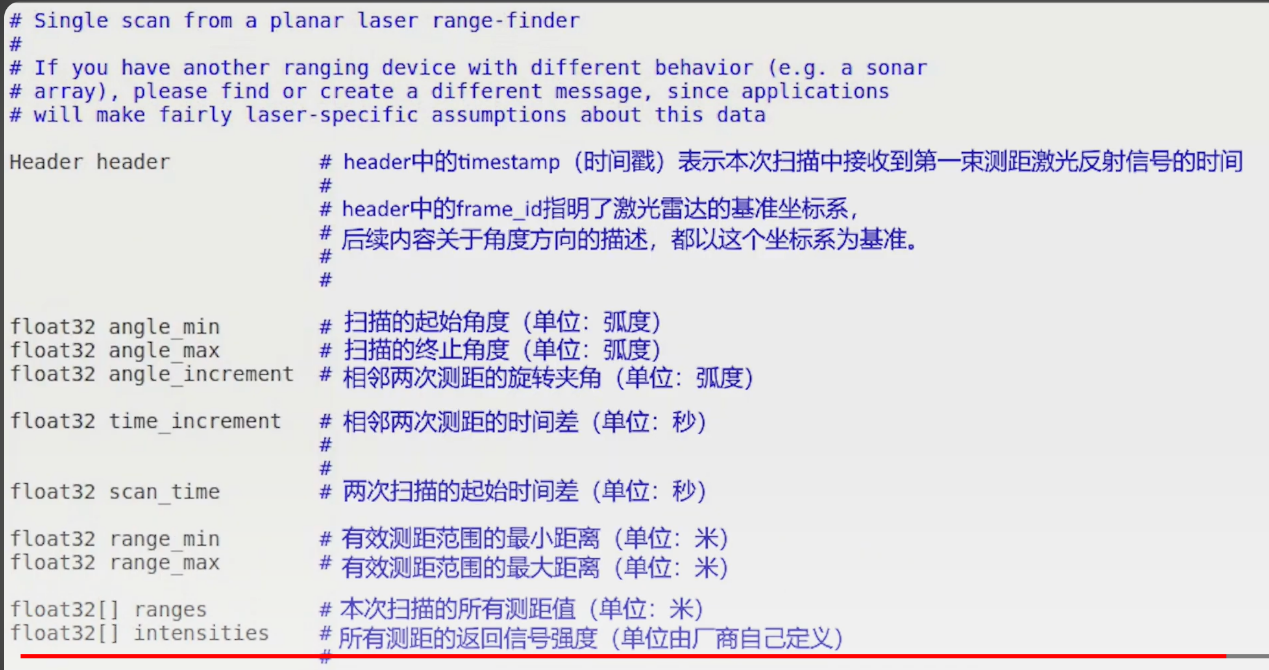
加载保存的数据文件：

方法一：进入rviz界面，点击右上角file按键，选择

open config 手动打开

方法二：在launch文件自动加载rviz配置文件

命令：“Roslaunch wpr\_simulation wpb\_rviz.launch”



ROS自定义消息包创建及添加新依赖包

* ROS消息包分类：

std\_msgs

(标准消息包)

ROS消息包

common\_msgs

(常用消息包)

* 消息包格式：



或者其它功能包定义的消息类型也可放进去！

* 创建自定义消息包：

Step1：进入工作空间src目录创建新功能包并添加依赖项

* + Command：catkin\_create\_pkg <功能包名字> roscpp rospy std\_msgs message\_generation message\_runtime

Step2：进入功能包目录，创建名为msg的新文件夹，并在里面创建自定义类型消息包文件（例如： Self.msg）

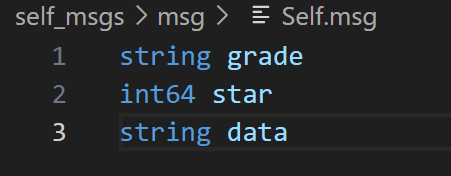
* + Command：cd <新创建功能包目录>

mkdir msg #创建名为msg的新文件夹

cd msg

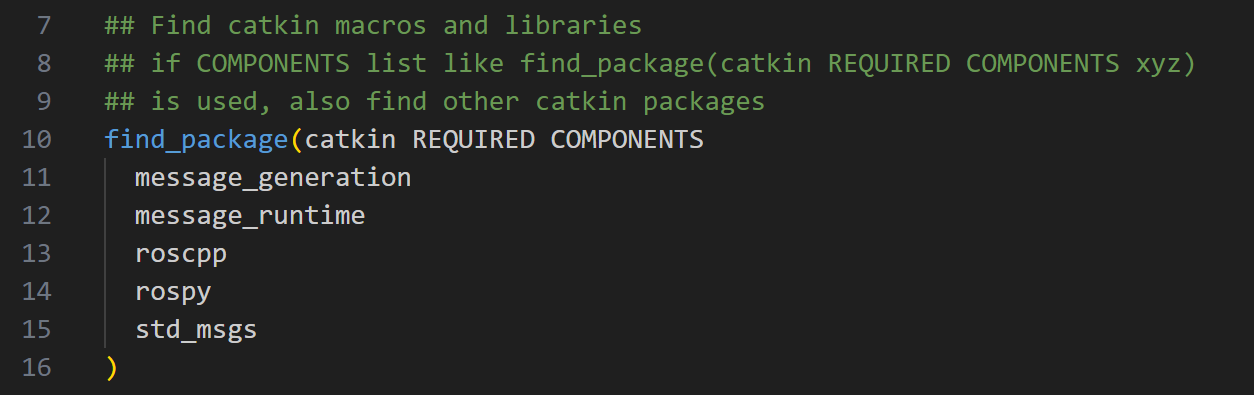
touch <自定义名字>.msg #例子为Self.msg

Step3：打开Self.msg 文件，根据消息包格式定义新消息结构

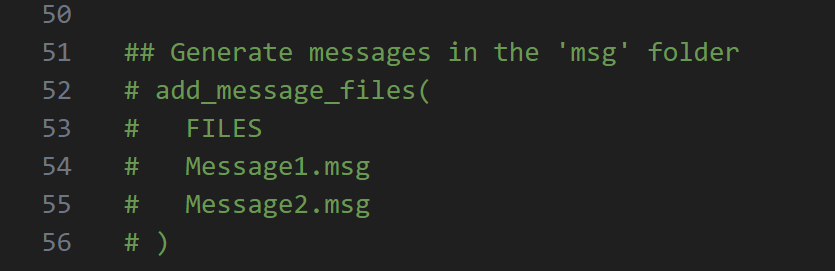


Step4：为新的消息类型设置编译规则：

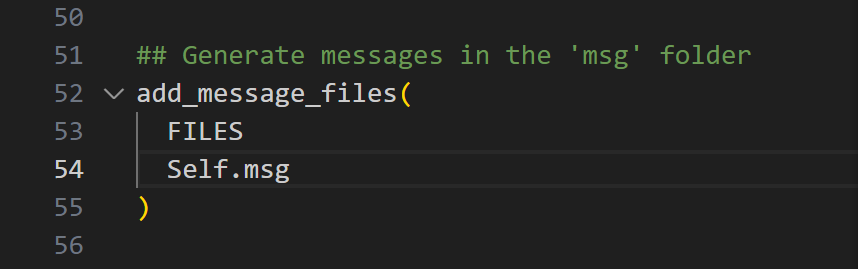
* + 打开CmakeLists.txt文件,确认find\_package里已包含message\_generation 和 message\_runtime



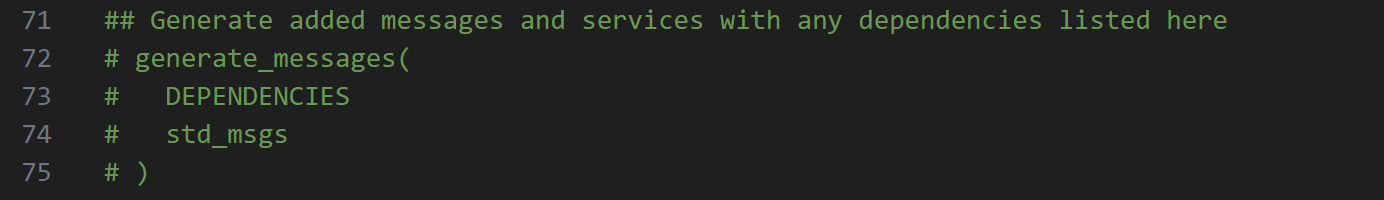
* + 将CmakeLists.txt中的 #add\_message\_files进行修改

修改前：

取消注释，将.msg文件替换为自己创建的消息类型

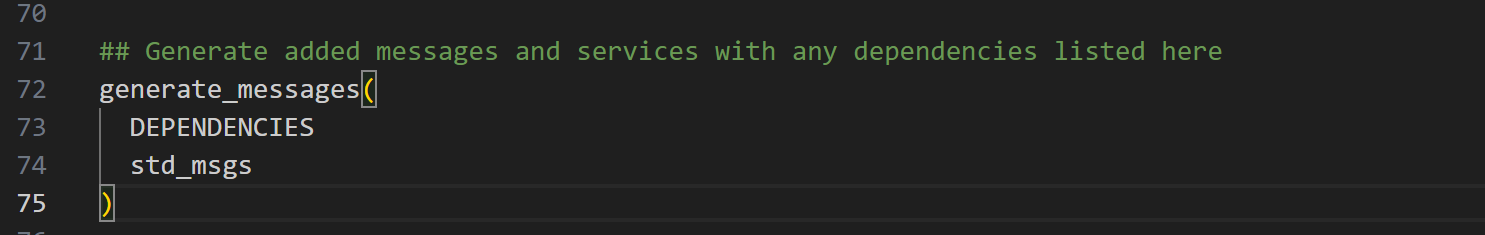
 修改后：

* + 将CmakeLists.txt中的 #generate\_messages进行修改

修改前：

取消注释

修改后：



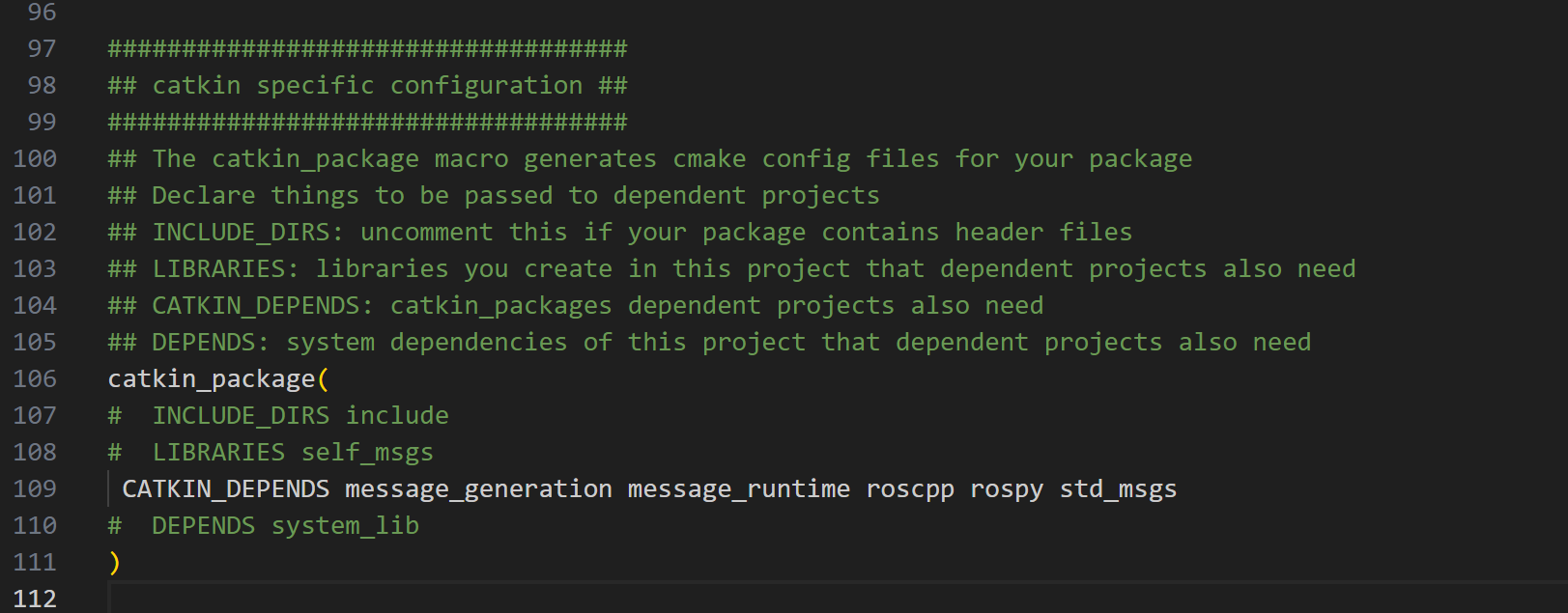
* + 将CmakeLists.txt中的 #generate\_messages进行修改：

修改前：

取消注释，并确认

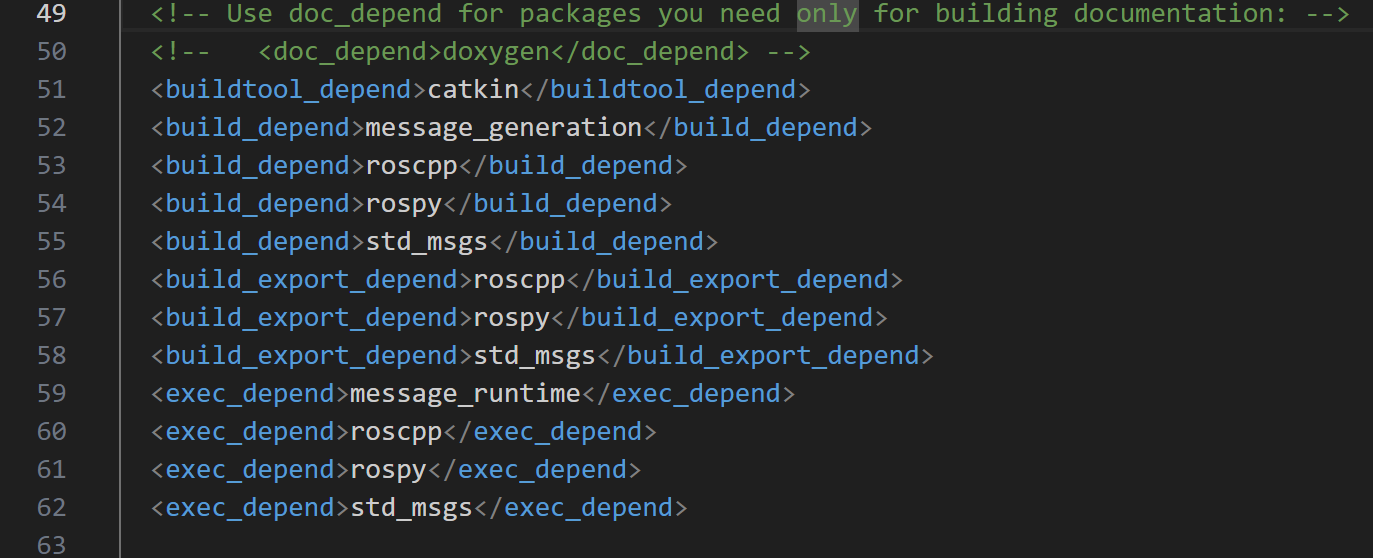
message generation、message runtime在其中

修改后：

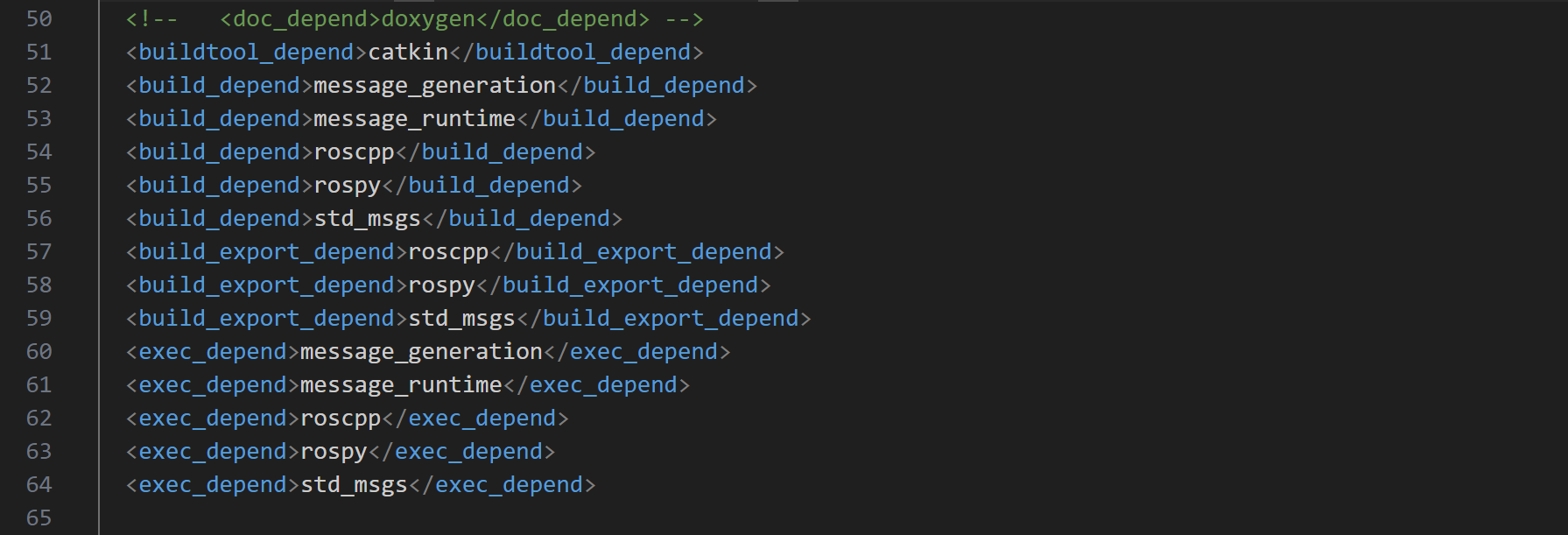


* + 至此，CmakeList.txt文件全部修改完成。接下来打开Package.xml文件进行修改。

修改前：



确保build\_depend 和 exec\_depend 中均包含message\_generation 、message\_runtime

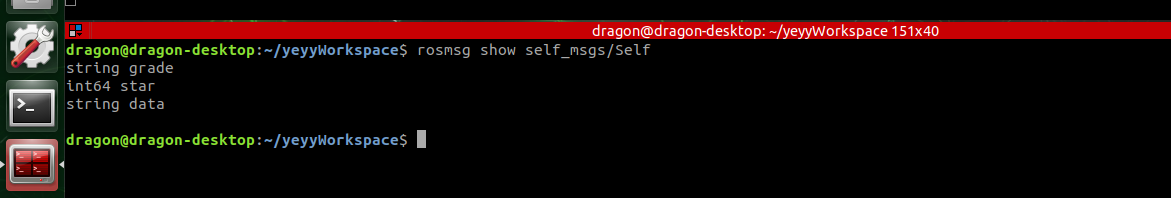
修改后：

至此全部修改完毕！

随后进入工作空间，进行编译！

查看新的消息类型是否进入ROS的消息列表

* + Command: rosmsg show <新消息包名称/消息类型名称>
  + 例如：rosmsg show self\_msgs/Self



恭喜你成功创建了自定义消息包类型！

SLAM算法

（Simultaneous Localization And Mapping）

（即时定位与地图构建）

* 简介：

以下介绍的都是SLAM的各类算法。同一台机器人可以运行多个 SLAM算法。

* 利用hector\_mapping算法进行激光雷达建图：

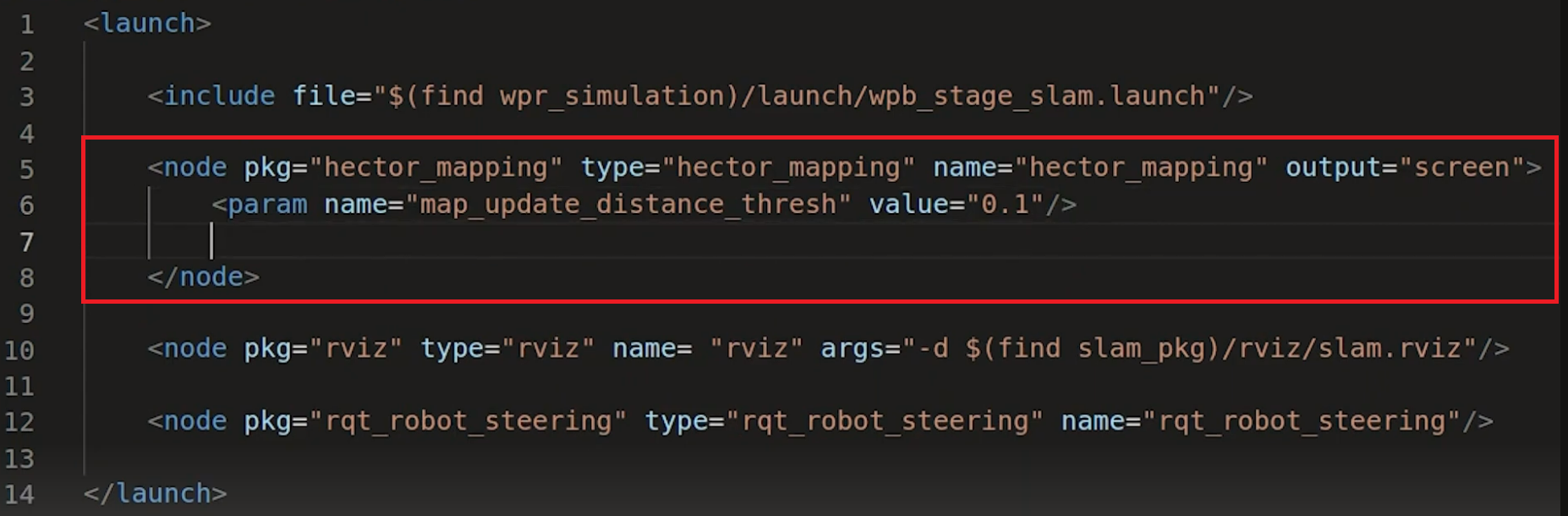
优点：仅依靠激光雷达即可进行建图，即它可以在没有里程计的情况在未知环境下构建当前环境的地图

缺点：当周围环境特征太过相似，即激光雷达扫描到的环境特征点前后之间过于相似，导致没有特征点没有位移变化。该算法会认为机器人没有在移动。从而出现地图与定位信息不准的情况。

实现方法：

* + Command：
* 未安装wpr\_simulation的请上翻文档进行安装，已安装的可略过。（此项只为仿真而用，可用自己的雷达数据代替）
* 运行仿真环境：roslaunch wpr\_simulation wpb\_sage\_slam.launch
* 运行SLAM节点：rosrun hector\_mapping hector\_mapping
* 运行rviz查看输出地图：rosrun rviz rviz
* 在rviz中添加机器人模型，添加雷达扫描测距点，添加地图，即可看到利用hector\_mapping算法建立的地图
* 若想让机器人仿真运动：rosrun rqt\_robot\_steering rqt\_robot\_steering 并将话题名称改为/cmd.vel即可。

hector\_mapping的参数设置：

* + 进入index.ros.org网站，搜索hector\_mapping。找到本机ros系统版本点击进入
  + 往下找到website，找到Parameters章节即可看所有参数。
  + 带波浪线的即为可以被设置的参数名。括号里为参数类型和默认值，下一行即为内容说明。
  + 修改格式：
    - <param name=”参数名” value=”修改后的值”/>
    - 示例：

TF系统

（Transform）

* 简介：

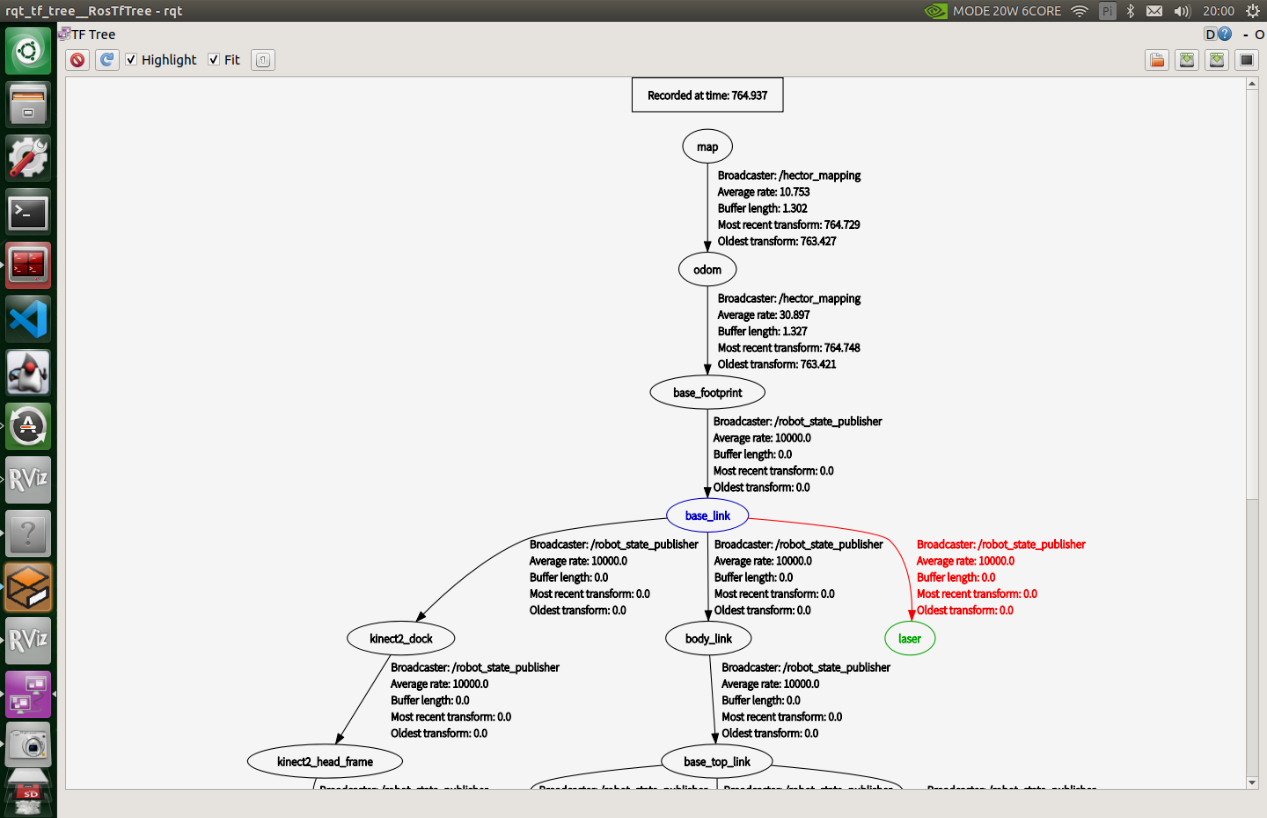
主要描述ROS系统中各个坐标系的空间关系。它可以帮你实时监控每个子坐标系与父坐标系的空间变换关系，前提是你要提前告诉它这俩坐标系初始时的位置关系

可以订阅/tf话题查看该ros系统中存在哪些坐标系以及它们之间的空间关系

* tf\_tree可视化:

通过命令：rosrun rqt\_tf\_tree rqt\_tf\_tree可以查看该ros系统中所有坐标系的层级关系，即tf树

* + 椭圆：代表一个坐标系
  + 上方椭圆是下方椭圆的父级坐标系

例子：

* 坐标系类型（仅列举六个）：

World：全局世界坐标系（”全局”代表它是固定的参考坐标系）

map：全局地图坐标系

base\_link：是机器人的底盘坐标系（机器人几何中心或者重心坐标系）

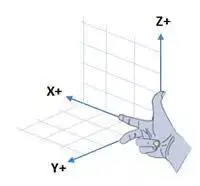
base\_footprint：通常被用作机器人的移动基准点。通常是机器人底部的中心点或底部的近似中心点。它是机器人在地面或参考平面上的投影点。

odom：里程计坐标系，用于估计机器人的位姿变化。通常由轮式里程计或惯性测量单元（IMU）提供

laser或者laser\_link：激光传感器坐标系

* Ros坐标系规则：

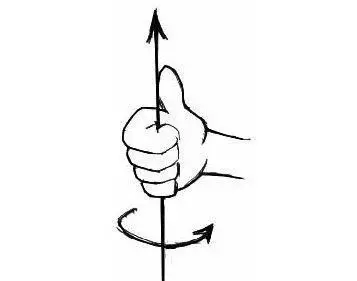
右手坐标系定义：

* + 把右手放在原点的位置，使大姆指，食指和中指互成直角，把 **大姆指** 指向 **Z轴** 的正方向，**食指**指向**X轴**的正方向时向时，**中指**所指的方向就是 **Y轴** 的正方向
  + 通常相对于我们的身体而言：

X 朝向前方

Y 朝向左方

Z 朝向上方

****绕坐标轴旋转定义：

* + 用右手握住坐标轴，**大拇指**的方向朝着坐标轴**朝向的正方向**，**四指环绕的方向**定义沿着这个坐标轴**旋转的正方向。**
  + 一般来说：

绕**Z轴**旋转：称之为 **航向角**，使用yaw表示

绕**X轴**旋转，称之为 **横滚角**，使用roll表示

绕**Y轴**旋转，称之为 **俯仰角**，使用pitch表示

如何发布TF变换关系：

* 发布静态TF坐标变换：

采用“static\_transform\_publisher”，直接写在launch文件中：

* + <node pkg="tf" type="static\_transform\_publisher" name="<自定义广播器名称>" args="<x> <y> <z> <yaw> <pitch> <roll> <frame\_id> <child\_frame\_id> <period\_in\_ms>" />
  + 参数解释：
  + 运行launch文件，用rosrun rqt\_tf\_tree rqt\_tf\_tree查看坐标系是否正确发布。也可使用rviz让坐标系可视化出来，fixed frame选择全局坐标点，然后点击add添加tf，即可看到可视化后的坐标系

例图：

* 动态TF变换发布：

采用tf广播器“**TransformBroadcaster”，动态的广播tf变换，需要编写动态tf变换节点。**

官方教程：进入index.ros.org搜索tf，寻找自己ros版本，找到website按钮点击即可寻找教程

注释：

* + Broadcaster(广播器）:是一个ROS节点，负责将坐标系之

间的变换信息广播出去

* + Average Rate(平均速率)：指在一定时间间隔内发布坐标

变换信息的平均速率。单位：秒

* + Buffer Length(缓冲区长度)：缓冲区长度指的是TF库中用

于存储变换数据的缓冲区的大小。它决定了可以保存多少历史变换数据，以便进行后续的坐标变换计算和处理

* + Most Recent Transform (最近的变换)：指从广播器接收到

的最新的坐标变换数据

* + Oldest Transform (最旧的变换)：指缓冲区中最老的坐标

变换数据

* 常用命令：

查看tf tree:

* + “rosrun rqt\_tf\_tree rosrun rqt\_tf\_tree”

里程计

（Odometry）

* 简介：

里程计并不是一个硬件设备，是一个软件算法！

以TF消息包的格式发送到/tf话题中

在tf\_tree 中为 odom坐标系

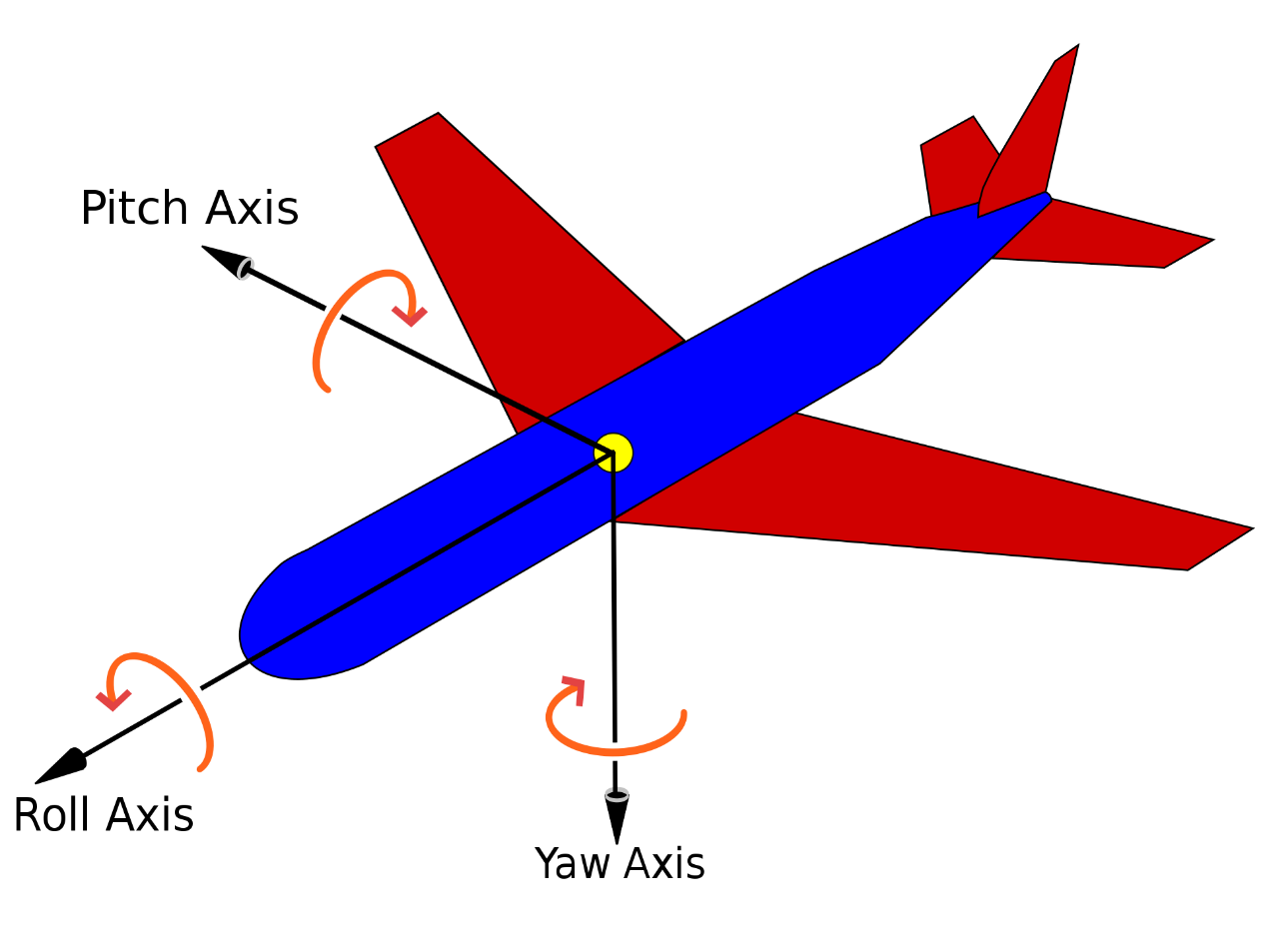
旋转（欧拉角与四元数）

* 简述：

三维空间中，描述刚体的旋转通常有很多方法：

* + 欧拉角（**Euler Angles）**
  + 轴角（**Axis-Angle**）
  + 四元数（**Quaternion**）
  + **旋转矩阵（Rotation Matrix）**
  + **姿态矩阵（Direction Cosine Matrix）**

欧拉角（**Euler Angles**）：

* + 使用三个角度（通常是绕三个坐标轴的旋转角度）来描述旋转。
  + 欧拉角顺序：Roll-Pitch-Yaw（滚动-俯仰-偏航）。
  + 优点：直观
  + 缺点：存在万向节锁！
  + “万向节锁”视频解析：

前往B站“https://www.bilibili.com”，搜索Up主“Ele实验室”，在其主页搜索“欧拉角”即可。

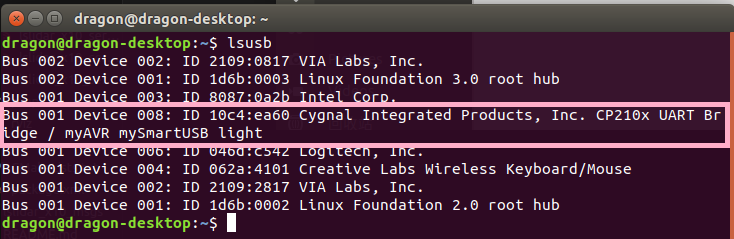
串口别名修改

* 目的:

为了将串口映射到一个别名上，从而避免因为插拔usb接口导致的串口号变化。

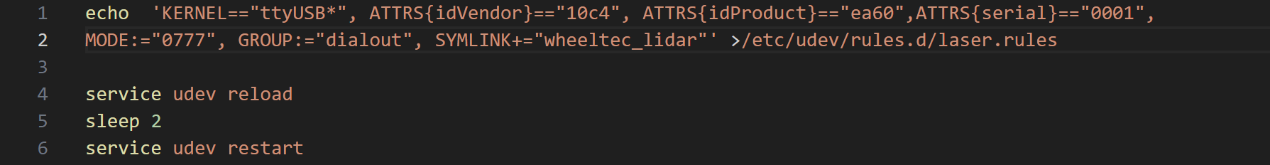
* 示例：通过sh脚本为串口配置雷达别名

Step1：通过命令 “lsusb”查看串口号

* + 

Step2：通过创建.sh脚本进行修改

* + 脚本示例



* + 注释：

Step3：进入创建好的.sh脚本文件目录

* + 给予权限： sudo chmod 777 <脚本名称>
  + 执行脚本：sudo ./<脚本名称>

Step4：键入命令“ll /dev/ | grep ttyUSB”查看修改是否成功！

Step5：修改成功后，即可使用别名代替原来的端口地址。

* + 例如：别名为：yyy666
    - * 原来端口地址：/dev/ttyUSB0
      * 配置别名后：/dev/yyy666
* 若存在多个相同芯片设备导致id识别号一致：

通过命令查看id识别号：

* + udevadm info -a -p $(udevadm info -q path -n /dev/ttyUSB0

找到ATTRS（devpath）一行

为文件脚本echo一行添加一个：

* + ATTRS{devpath}==”<查询到的识别码>”
  + devpath：为本机设别USB端口的固定端口号

重新按上述执行脚本即可

此时别名已经绑定到固定的usb端口，只要记住哪一个USB端口对应哪一个别名设备，将正确的设备插入原先设定好的USB端口，即可正确匹配！（错了换一换端口）

Gazebo地图构建

* 目的：

使用gazebo搭建自己绘制的模型地图，并用launch文件加载

* 步骤

Step1：打开终端，键入gazebo

Step2：点击左上角Edit，选择building Editor

Step3：此时可以开始绘制地图模型

Step4：点击左上角file，选择Save As 。保存至提前在工作空间创建的地图功能包的models文件夹（未提前创建此刻创建即可）

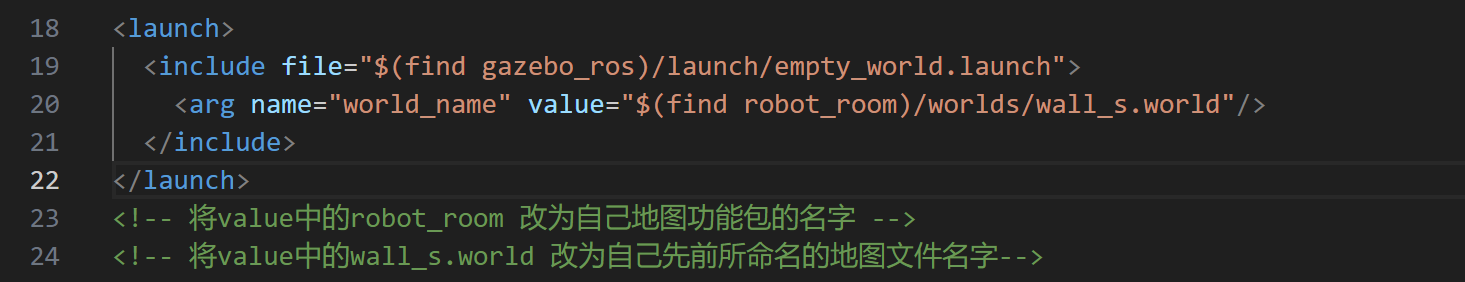
Step5：按ctrl+x退出模型建构 或 重新打开gazebo点击左边功能栏insert，选择add path将刚刚的（models）模型文件夹路径添加进去

Step6：随后单击左侧功能栏的模型文件路径，将模型放置地图中，自行调整坐标位置。

Step7：点击左上角file，选择save world as，将文件保存至提前在工作空间创建的地图功能包的worlds文件夹（未提前创建此刻创建即可）。注意文件名尾缀一定得为“.world”

Step8：然后我们在同工作空间下建立world.launch文件（可将放至任意功能包目录下，本例子放入先前所创建的地图功能包目录的launch文件夹下）

Step9：为launch文件添加如下代码：

* + 

Step10：使用roslaunch执行我们刚才编写的launch文件

* + 进入工作空间目录
  + 键入：roslaunch <地图功能包名称> <地图文件名称>
  + 例如：roslaunch map\_pkg world.launch

机器人模型构建

* 利用URDF构建机器人模型：

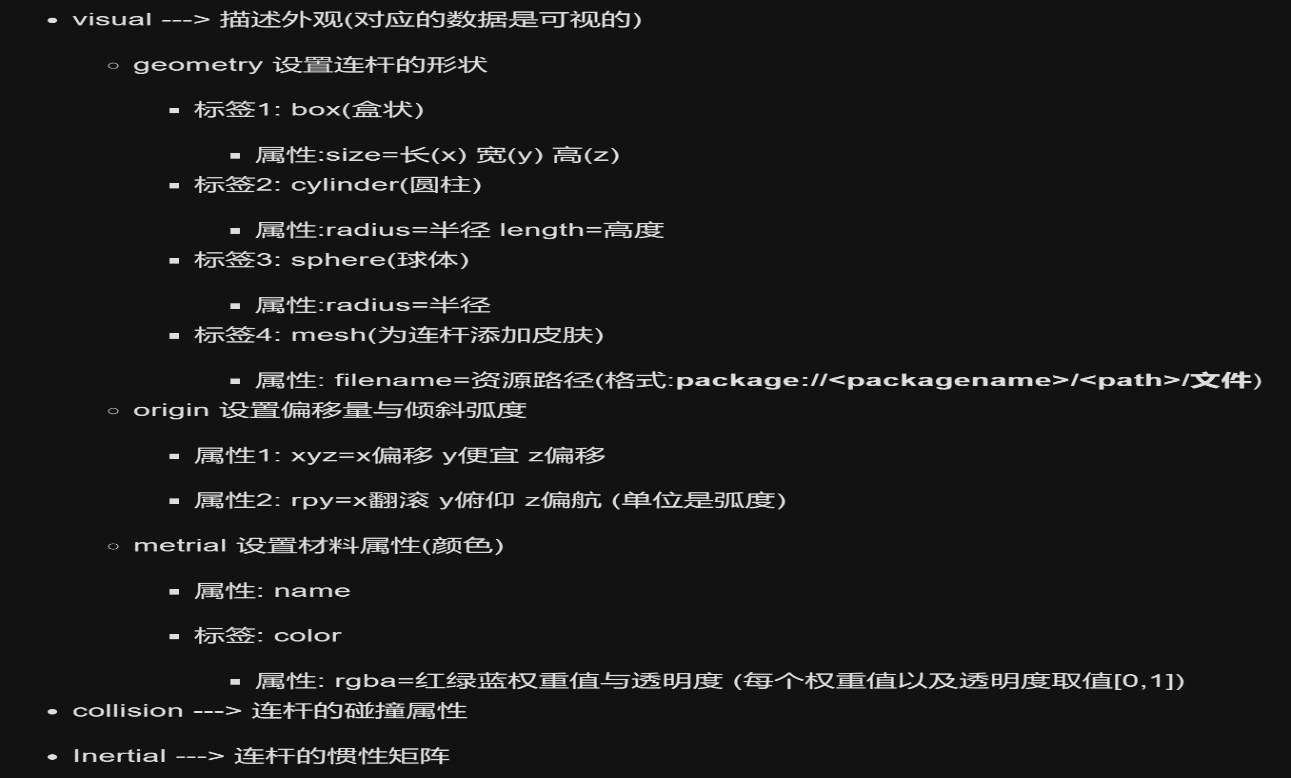
URDF主要用于机器人模型建模。后缀为urdf的文件本质上仍是xml文件。

* URDF语法：

标签<robot> …………………… </robot>

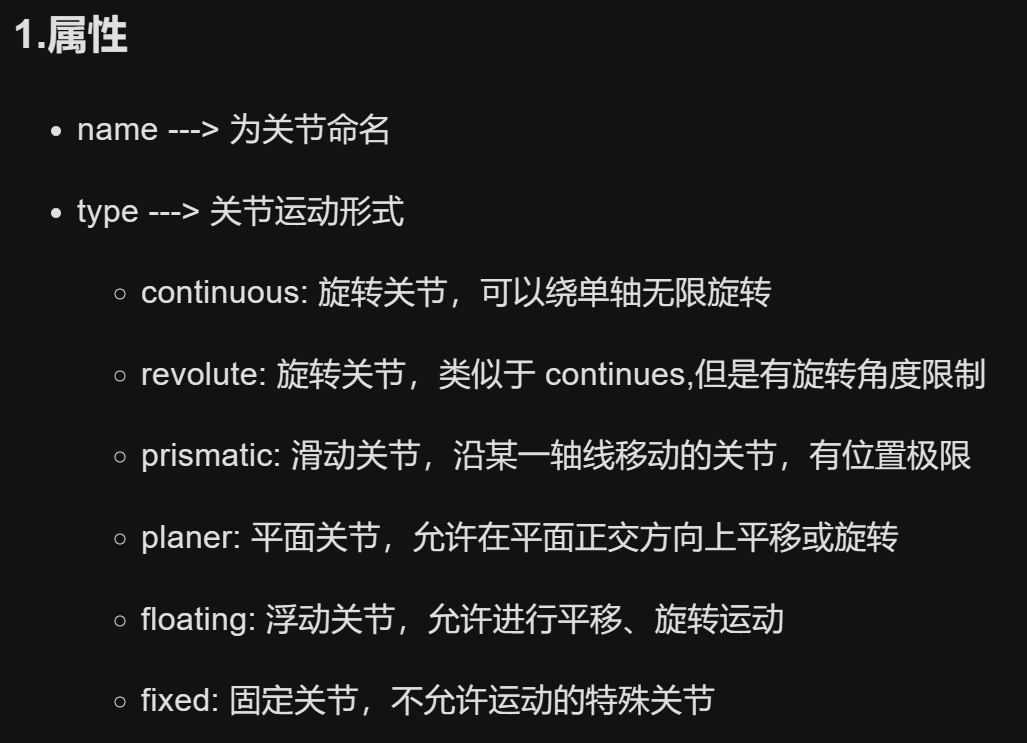
* + 本质仍为xml文件，URDF文件的根标签为<robot>
  + 属性：<robot name=”………”>（用于设置机器人模型名称）
  + 子标签：其余标签均为其的子标签

标签<link> …………………………</link>

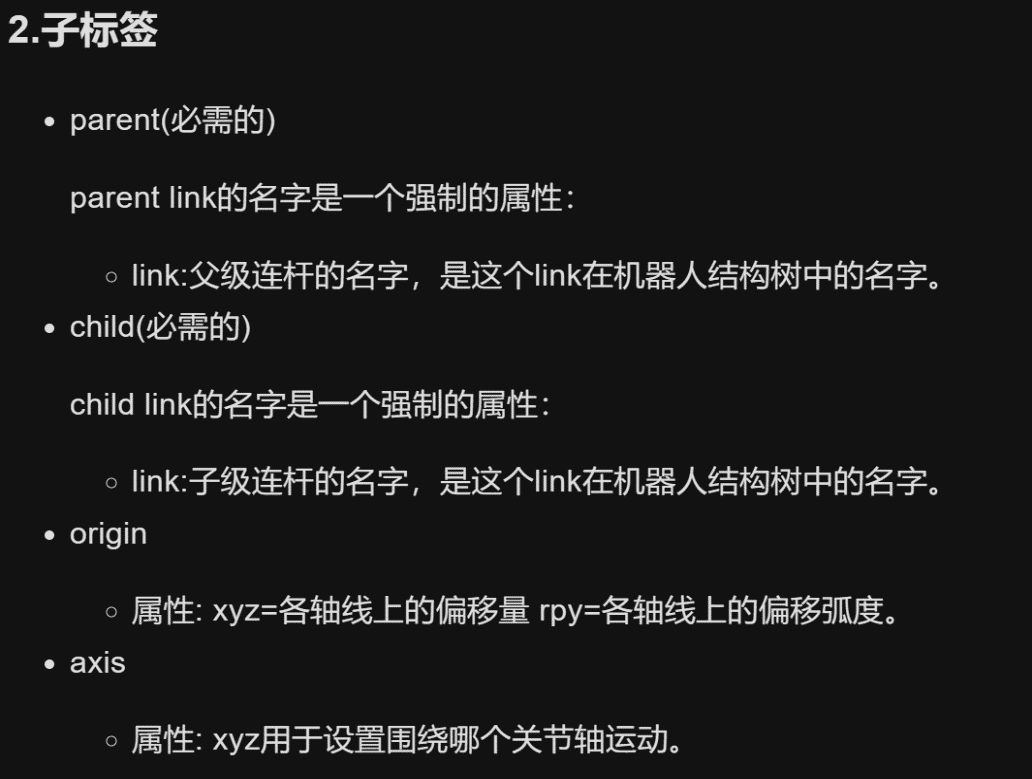
* + link 标签用于描述机器人某个部件的外观和物理属性(即刚体部分)
  + 属性：<link name=”……”>(用于设置link标签名称)
  + 子标签：（metrial改为 material）

标签<joint> …………………………</joint>

* + joint 标签用于描述机器人关节的运动学和动力学属性，还可以指定关节运动的安全极限
  + 属性：<joint name=”……”> 和 <joint type=”……”>



* + 子标签：



* 使用流程：

Step1：创建一个功能包，除必要的依赖外，加入urdf依赖。

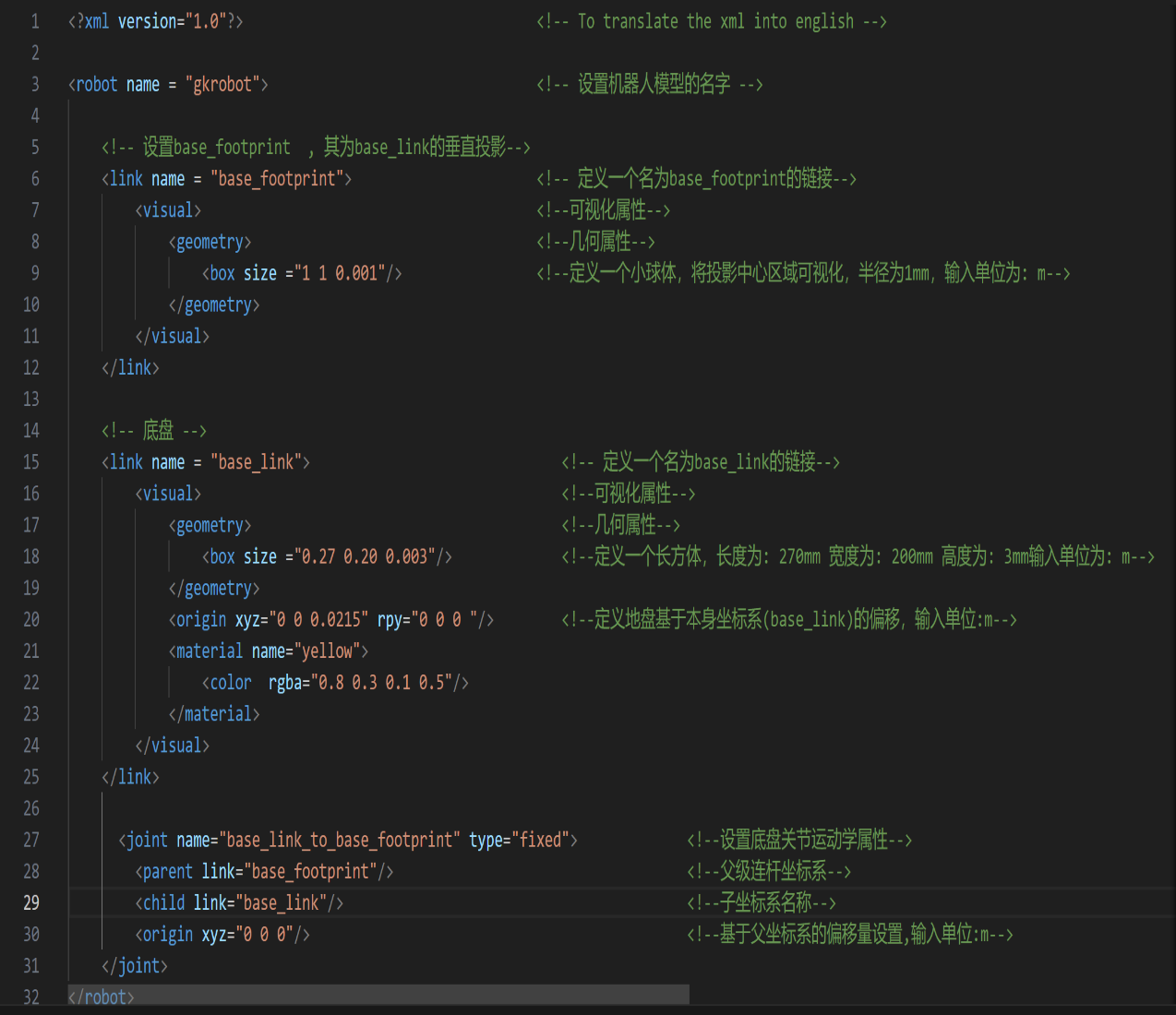
Step2：在功能包构建一个名为model的文件夹用于存放“.urdf”模型文件。

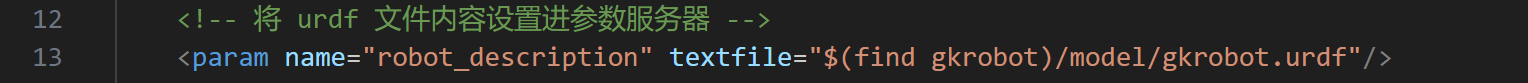
Step3：在model目录创建：<自定义名称>.urdf文件

Step4：“.urdf文件”首行加入如下声明（声明xml版本）

* + <?xml version=”1.0”?>

Step5：根据上述URDF语法构建机器人模型。

* + 例如：

Step6：在launch文件加载urdf文件：

注意！！在urdf文件中出现的坐标系，必须提前构建好！如何发布tf坐标，见本文档目录《TF系统》一文

* 利用XML Macros（Xacro）机器人模型：

本质仍为XML文件，文件后缀为“.xacro”。URDF 中的部分内容是高度重复的，可能只是相关参数不同。较之于纯粹的 URDF 实现，Xacro可以编写更安全、精简、易读性更强的机器人模型文件，且可以提高编写效率，避免重复造轮子。

保存SLAM所建的地图

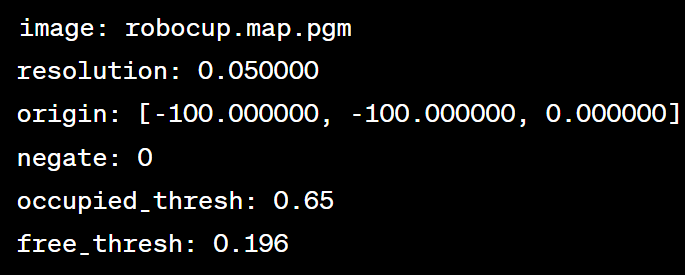
* 利用map\_server功能包保存地图

使用前提：已经对地图完全扫描！

Step1：cd进入需要保存地图的所在文件夹

Step2：执行“rosrun map\_server map\_saver -f <自定义地图名称>.map”

Step3:随后即可在文件夹内看到“.pgm”和“.yaml”文件。

* + .pgm为图片格式文件，其显示的是所构建地图的照片
  + .yaml为地图元数据文件的样本
  + 
    - Image：地图使用的图像文件的名称
    - resolution：地图分辨率（单位：m/Pixel 即 米/像素）
    - origin：地图坐标系的原点位置和偏转角[x,y,yaw]
    - negate：表示地图中障碍物的值是否被取反(0否1是)
    - occupied\_thresh：被认为是"occupied"（占用）栅格的阈值，这里的值是0.65。
    - free\_thresh：被认为是"free"（可通行）栅格的阈值，这里的值是0.196。

.yaml文件详情查看ros.wiki 的map\_server功能包

* 加载地图并显示：

运行“rosrun map\_server map\_server <地图名称>.yaml”

进入rviz，点击add，选择map，将topic 选择/map，即可查看地图。

* 关于map\_server功能包

当使用map\_server时，其会发布两个话题，分别为：

* + map\_metadata
  + map

当使用map\_server时，其会订阅一个话题，其为：

* + map（读取地图并显示时不需要提前发布此话题）

ROS与STM32通信

* 采用UART串口通信方式

简述：

* + 上位机和下位机采用USB转TTL串口通信模块连接。

ROS端：

* + 安装serial串口通信库，
    - 命令：“sudo apt-get install ros-<ROS版本号>-serial”
  + C++：为功能包添加serial依赖。修改CmakeLists.txt和package.xml文件即可。
  + Python：有可能需要使用pip安装pyserial后才能使用，本例程已安装上述serial库以及使用pip安装pyseial。随后在python文件import serial 可正常使用。

Stm32端：

* + 按正常串口通信配置进行配置即可。
  + HAL库串口接收函数介绍：
    - **HAL\_UART\_Receive（轮询）（同步阻塞方式）**
      * 该函数是一个阻塞的同步接收函数。它会一直等待，直到接收到指定数量的数据或超时。
    - **HAL\_UART\_Receive\_IT（接收定长数据）**
      * 该函数是一个异步的接收函数，使用中断触发接收。在接收完成时，将调用用户定义的回调函数，从而实现非阻塞的接收。
    - **HAL\_UART\_Receive\_DMA（接收不定长数据）**

上下位机串口通信协议编写：

ROS图像与OpenCv图像转换

* 简述：

**ROS以自己的sensor\msgs/Image消息格式传递图像**，但用户希望将图像与OpenCV结合使用。**在OpenCV中，图像以Mat矩阵的形式存储**，与ROS定义的图像消息的格式有一定的区别，CvBridge是一个ROS库，**提供ROS和OpenCV之间的接口**

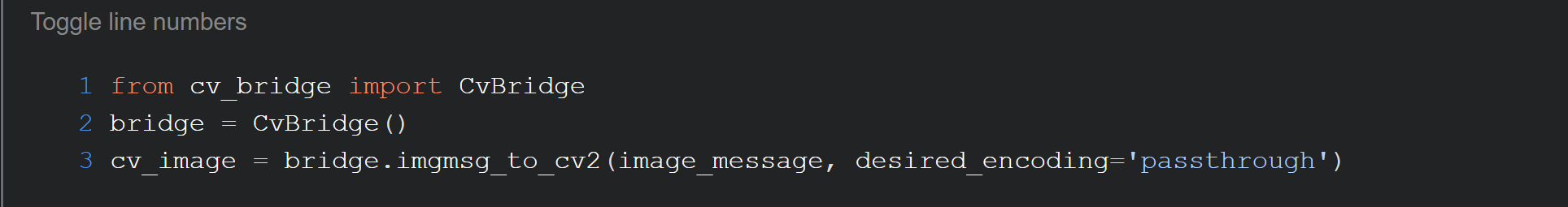
* 关于cvbridge：

Cvbridge是一个ROS库，故可在index.ros.org中搜索到。里边有详细的指导教程

* ROS图像消息转换至Opencv

当用户希望在ROS中使用Opencv进行图像处理时，且图像源格式为Ros的图像消息格式时，采用该转换

实现方法：

* + Step1：为功能包添加cvbridge依赖，详情见目录“**ROS自定义消息包创建及添加新依赖包**”
  + 注意！cvbridge要求必须依赖项为：sensor\_msgs；std\_msgs；rospy；cvbridge
  + Step2：
  + 第一行：导入cvbridge包
  + 第二行：实例化cvbridge包
  + 第三行：image\_message：需要被转换Opencv的图像

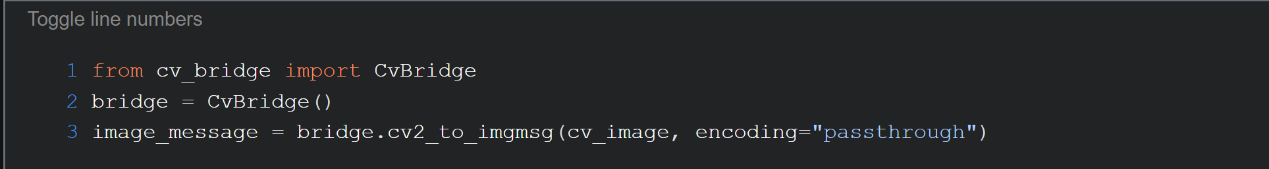
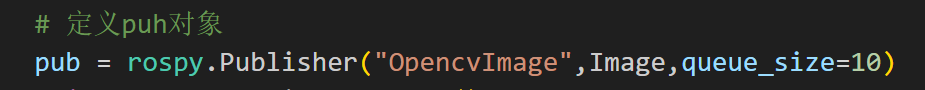
Desired\_encoding：图像编码格式：使用“brg8”

cv\_image：转换好的Opencv图像。

* Opencv图像转换至ROS图像消息：

当用户希望在ROS的不同节点中使用Opencv图像，采用该转换，并将转换好的图像消息发布话题即可

实现方式：

* + Step1：为功能包添加cvbridge依赖，详情见目录“**ROS自定义消息包创建及添加新依赖包**
  + 注意！cvbridge要求必须依赖项为：sensor\_msgs；std\_msgs；rospy；cvbridge
  + Step2：采用cv2.VideoCapture()获取摄像头画面，将画面输入至cv\_image中进行转换，encoding=“rgb8”：
  + Step3：定义pub发布者对象：

“OpencvImage”：用户自定义话题名称

“Image”：发布的消息类型

“queue\_size”：缓存

脚本实现ROS程序开机自启动

Rosbag记录/回放rostopic数据

机器人位姿融合Robot\_pos\_ekf

* 简介：
  + 该软件包可用于评估机器人的3D位置姿态，是ros navigation stack 中的一个包。通过扩展卡尔曼滤波器对 imu、里程计 odom、视觉里程计 vo 的数据进行融合,来估计平面移动机器人的真实位置姿态,输出 odom\_combined 消息。详情见roswiki。

安装：sudo apt-get install ros-<ros版本号>-robot-pos-ekf

* 该软件包会发布的tf坐标关系：

Odom\_combined->base\_footprint（动态tf坐标）

* 注意！！：

该软件包发布的话题名称可更改，关联的子坐标系也可以更改，为了与原始odom区分，这里使用了odom\_combined。base\_footprint也可改为base\_link。

* 该软件包需要订阅的话题（即输入该软件包的消息）

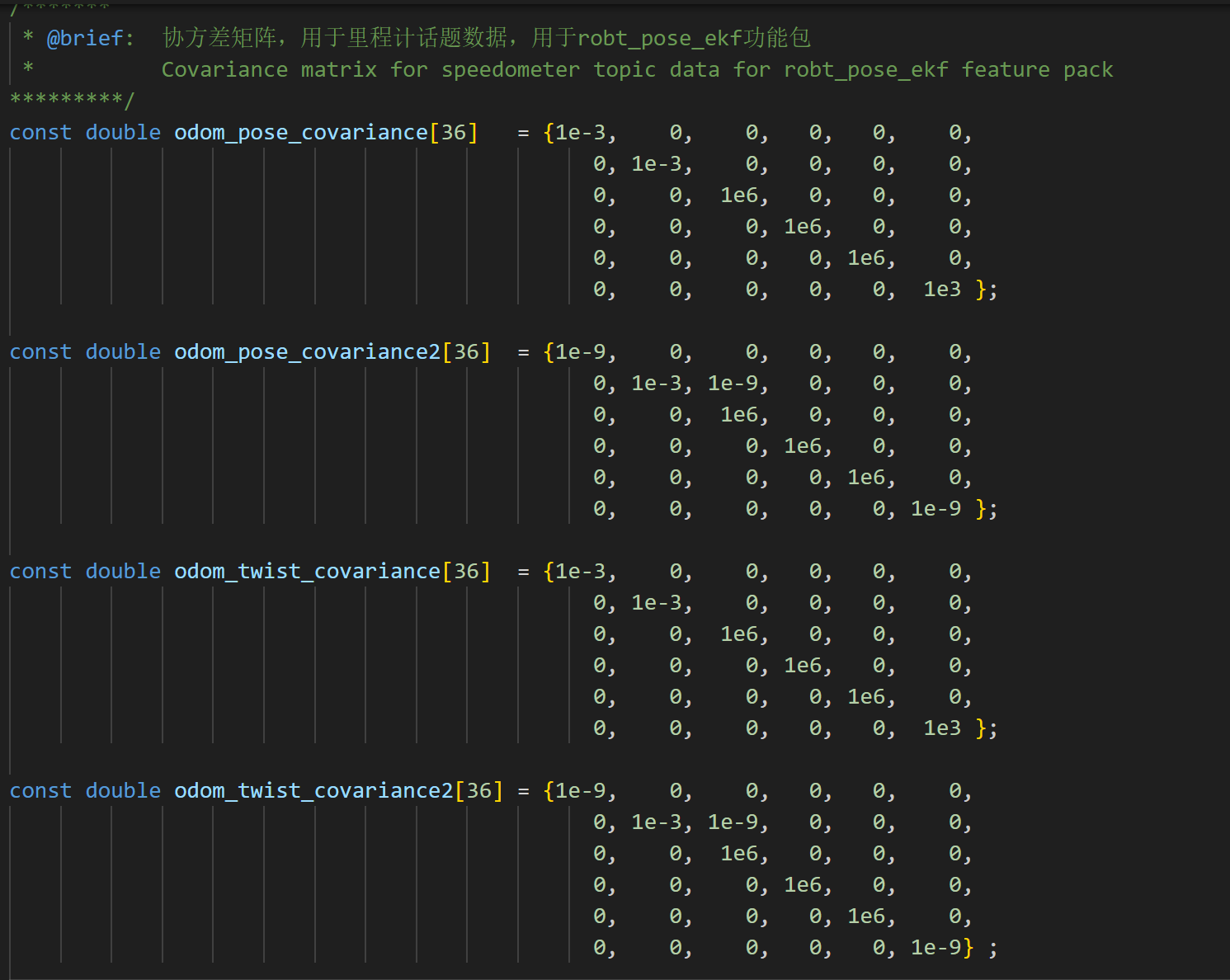
odom(nav\_msgs/Odometry)

imu\_data(sensor\_msgs/Imu)

* + vo(nav\_msgs/Odometry)
* 详细使用方法（更详细见roswiki）：



设置协方差矩阵（Covariance Matrix）！！！！！！！！！！！！！！

* + 因为使用该软件包即使用了卡尔曼滤波，所以必须要有协方差矩阵，故在odom消息发布中必须带上协方差矩阵。
  + 带2标签用于动态，无标签用于静态。

SLAM建图--Hector\_Mapping

* 简述：

Hector\_Mapping是一个无需里程计数据的SLAM算法，相关参数已经说明可在index.ros.org搜索获取。

* 如何使用：

安装Hector\_Mappipng：

* + “sudo apt-get install ros-melodic-hector-mapping”

需要自行发布的tf坐标：

* + Laser->base\_link；
  + base\_link->odom

示例：

SLAM建图—Gmapping

* 简介：

Gmapping是一种基于粒子滤波的SLAM算法。本文旨在提供实现建图思路，Gmapping详情信息见roswiki，源码见github。

安装：“sudo apt-get install ros-<ros版本号>-gmapping”

* Gmapping会发布的tf变换

map->odom（动态坐标变换）

* Gmapping需要用户发布的tf变换

Odom->base\_footprint (里程计到机器人地盘基坐标)

Scans->base\_link（激光雷达坐标系到机器人机身坐标系）

* 注意点！！：

Scans->base\_link通常为静态tf坐标变换。

Odom->base\_footprint通常为动态tf坐标变换。

* Odom->base\_footprint的动态tf坐标如何发布.

1st：采用robot\_pos\_ekf包，计算机器人的三位空间姿态。该包会替用户发布odom->base\_footprint的动态tf变换。

2nd：用户采用自写节点，发布odom->base\_footprint的动态tf坐标变换。通过引入头文件< tf/transform\_broadcaster.h>实现，通过geometry中的TransformStamped实现动态发布odom->base\_footprint的tf关系。

* 实现Gmapping建图的tf树图：