第四章、ROS运行管理

一、ROS元功能包

1. 概念：

> MetaPackage是Linux的一个文件管理系统的概念。是ROS中的一个虚包，里面没有实质性的内容，但是它依赖了其他的软件包，通过这种方法可以把其他包组合起来，我们可以认为它是一本书的目录索引，告诉我们这个包集合中有哪些子包，并且该去哪里下载。

2. 作用：

> 方便用户的安装，我们只需要这一个包就可以把其他相关的软件包组织到一起安装了。

3. 实现

首先、创建一个没有依赖的功能包

```
catkin_creat_pkg 包名
    // catkin_creat_pkg my_bag
```

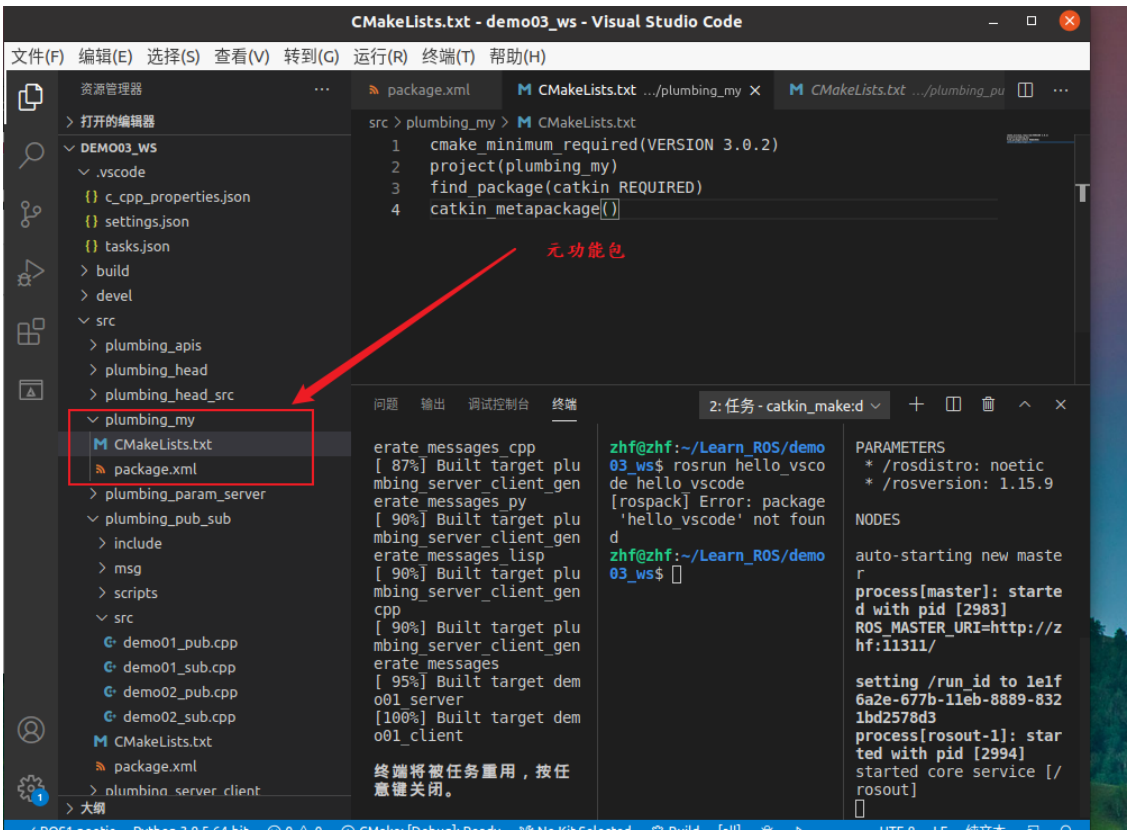
然后：添加依赖package.xml

```
<!-- 加下面这个标签 -->
<exec_depend>被集成的功能包</exec_depend>
    <!-- <exec_depend>plumbing_pub_sub</exec_depend> -->
 .....
 <export>
   <!-- 加下面这个标签 -->
   <metapackage />
 </export>
```

最后：修改CMakeList.txt

只能有这四行，不能有换行和别的、括号内容根据情况修改

```
cmake_minimum_required(VERSION 3.0.2)
project(plumbing_my)
find_package(catkin REQUIRED)
catkin_metapackage()
```



二、launch文件

概念

> launch 文件是一个 XML 格式的文件，可以启动本地和远程的多个节点，还可以在参数服务器中设置参数。

作用

> 简化节点的配置与启动，提高ROS程序的启动效率。

使用

1. 在功能包下创建launch文件夹
2. 创建xxx.launch文件
3. 启动launch文件

```
roslaunch 包名 xxx.launch
```

launch文件的标签

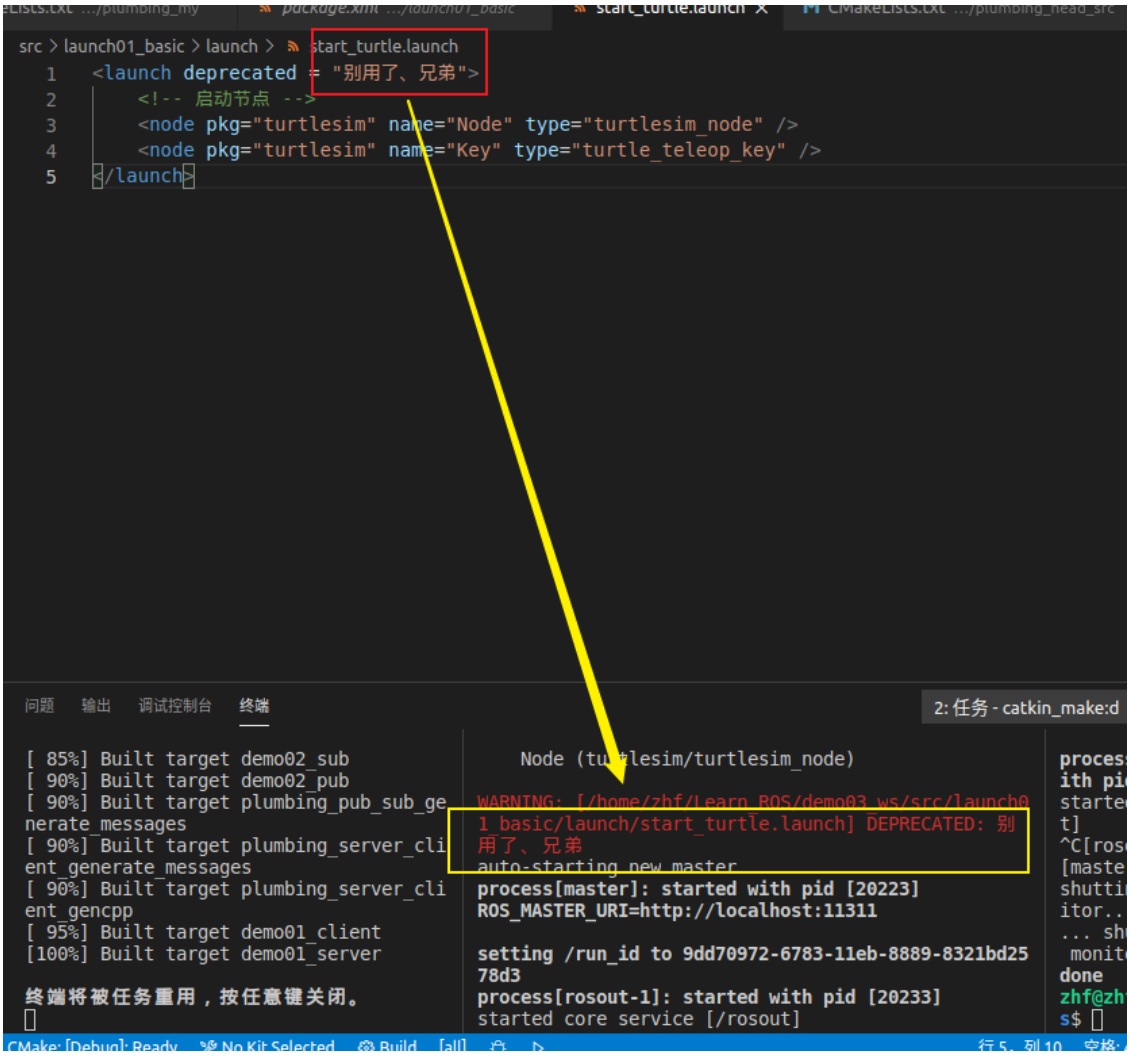
1. launch 标签:

<launch>标签是所有 launch 文件的**根标签**，充当其他标签的容器

属性： 告知用户当前 launch 文件已经弃用

deprecated = "弃用声明"、告知用户，该launch文件已经是旧版本了，不建议使用，如果使用，会在命令行提示用户

```xml
<launch deprecated = "别用了、兄弟">
    <!-- 启动节点 -->
    <node pkg="turtlesim" name="Node" type="turtlesim_node" />
    <node pkg="turtlesim" name="Key" type="turtle_teleop_key" />
</launch>
```



子集标签：所有其它标签都是launch的子级

2. node 标签

<node> 标签用于指定 ROS 节点，是最常见的标签，需要注意的是: roslaunch 命令不能保证按照 node 的声明顺序来启动节点(节点的启动是多进程的)

- *属性*

```xml
<!-- node -->
<!-- pkg="包名" 节点所属的包 -->
<!-- type="节点类型" 节点类型(与之相同名称的可执行文件) -->
<!-- name="节点名称" 节点名称(在 ROS 网络拓扑中节点的名称) -->
<!-- output="log | screen" 日志发送目标，可以设置为 log 日志文件，或 screen 屏幕,默认是 log -->
<!-- ns="xxx" 在指定命名空间 xxx 中启动节点 -->

<!-- machine="机器名" 在指定机器上启动节点 -->
<!-- respawn="true | false" 如果节点退出，是否自动重启(默认是false) -->
<!-- respawn_delay="N" 如果 respawn 为 true，那么延迟 N 秒后启动节点 -->
<!-- args="xxx xxx xxx" 将参数传递给节点 -->
```

```
<!-- required="true | false" 该节点是否必须，如果为 true,那么如果该节点退出，将杀死整个 roslaunch -->

<!-- clear_params="true | false" 在启动前，删除节点的私有空间的所有参数 -->
```

3.include文件标签

`include`标签用于将另一个 xml 格式的 launch 文件导入到当前文件

- *属性*

```
<!-- file="$(find 包名)/xxx/xxx.launch" 要包含的文件路径 -->
<!-- ns="xxx" 在指定命名空间导入文件 -->
<launch>
    <include file="$(find launch01_basic)/launch/start_turtle.launch" />
</launch>
```

4.remap文件标签，node的子集标签

重映射节点名称

- *属性*

```
<!-- from="xxx" 原始话题名称 -->
<!-- to="yyy" 目标名称 -->
<node pkg="turtlesim" name="Node" type="turtlesim_node" respawn="true" respawn_delay="10">
    <remap from="/turtle1/cmd_vel" to="/cmd_vel" />
</node>
<node pkg="turtlesim" name="Key" type="turtle_teleop_key" />
```

5.param参数标签

`<param>`标签主要用于在参数服务器上设置参数，参数源可以在标签中通过 value 指定，也可以通过外部文件加载，在`<node>`标签中时，相当于私有命名空间。

- *属性*

```
<!-- name="命名空间/参数名" 参数名称，可以包含命名空间 -->
<!-- value="xxx" 定义参数值，如果此处省略，必须指定外部文件作为参数源 -->
<!-- type="str | int | double | bool | yaml" 指定参数类型，如果未指定，roslaunch 会尝试确定参数类型 -->
    <!-- 规则如下:
        如果包含 '.' 的数字解析未浮点型，否则为整型
        "true" 和 "false" 是 bool 值(不区分大小写)
        其他是字符串
    -->
<!-- 定义在node外 -->
<param name="param_A" type="int" value="100" />
<node pkg="turtlesim" name="Node" type="turtlesim_node" respawn="true" respawn_delay="10">
    <!-- 定义在node内 -->
    <param name="param_B" type="double" value="3.14" />
</node>
```

6.rosparam参数标签和yaml交互

`<rosparam>`标签可以从 YAML 文件导入参数，或将参数导出到 YAML 文件，也可以用来删除参数，`<rosparam>`标签在`<node>`标签中时被视为私有。

- *属性*

```
<launch>
<!-- command="load | dump | delete" (默认 load) 加载、导出或删除参数 -->
<!-- file="$(find 包名)/xxx.../yyy.yaml" 加载或导出到的 yaml 文件 -->
<!-- param="参数名称"  -->
<!-- ns="命名空间" -->
    <rosparam command="dump" file="$(find launch01_basic)/launch/params_out.yaml" />
    <rosparam command="delete" param="bg_B"/>
</launch>
<launch>
    <!-- 定义在node外 -->
    <!-- 从yaml中加载参数 -->
    <rosparam command="load" file="$(find launch01_basic)/launch/params.yaml" />
    <node pkg="turtlesim" name="Node" type="turtlesim_node" respawn="true" respawn_delay="10">
        <!-- 定义在node内 -->
        <rosparam command="load" file="$(find launch01_basic)/launch/params.yaml" />
    </node>
</launch>
```

7.group标签

`<group>`标签可以对节点分组，具有 ns 属性，可以让节点归属某个命名空间

- *属性*

```
<!-- ns="名称空间" -->
<!-- clear_params="true | false" 启动前，是否删除组名称空间的所有参数(慎用.... 此功能危险) -->
<launch>
    <group ns="first">
        <node pkg="turtlesim" name="Node" type="turtlesim_node" respawn="true" respawn_delay="10"/>
        <node pkg="turtlesim" name="Key" type="turtle_teleop_key" />
    </group>
    <group ns="second">
        <node pkg="turtlesim" name="Node" type="turtlesim_node" respawn="true" respawn_delay="10"/>
        <node pkg="turtlesim" name="Key" type="turtle_teleop_key" />
    </group>
</launch>
```

8.arg动态传参

`<arg>`标签是用于动态传参，类似于函数的参数，可以增强launch文件的灵活性

- *属性*

```
<launch>
    <!-- name="参数名称" -->
    <!-- default="默认值" (可选) -->
    <!-- value="数值" (可选) 不可以与 default 并存 -->
    <!-- doc="描述" 参数说明 -->
    <arg name="car_length" default="0.5"/>
    <param name="A" value="$(arg car_length)"/>
    <param name="B" value="$(arg car_length)"/>
    <param name="C" value="$(arg car_length)"/>
</launch>
```

三、工作空间覆盖

- 所谓覆盖就是存在不同工作空间下的相同功能包时，在调用时，会调用后刷新的

> ROS 会解析 .bashrc 文件，并生成 ROS_PACKAGE_PATH ROS包路径，该变量中按照 .bashrc 中配置设置工作空间优先级，在设置时需要遵循一定的原则:ROS_PACKAGE_PATH 中的值，和 .bashrc 的配置顺序相反--->后配置的优先级更高，如果更改自定义空间A与自定义空间B的source顺序，那么调用时，将进入工作空间A。

四、节点名称重名

1.rosrun设置命名空间与重映射（起别名）

1. 设置命名空间演示

> 语法: rosrun 包名 节点名 __ns:=新名称

```
rosrun turtlesim turtlesim_node __ns:=/xxx
rosrun turtlesim turtlesim_node __ns:=/yyy
```

- 运行结果：用`rosnode list`查看

```
/xxx/turtlesim
/yyy/turtlesim
```

2. 为节点起别名

> 语法: rosrun 包名 节点名 __name:=新名称

```
rosrun turtlesim  turtlesim_node __name:=t1
rosrun turtlesim  turtlesim_node __name:=t2
```

- 运行结果：用`rosnode list`查看

```
/t1
/t2
```

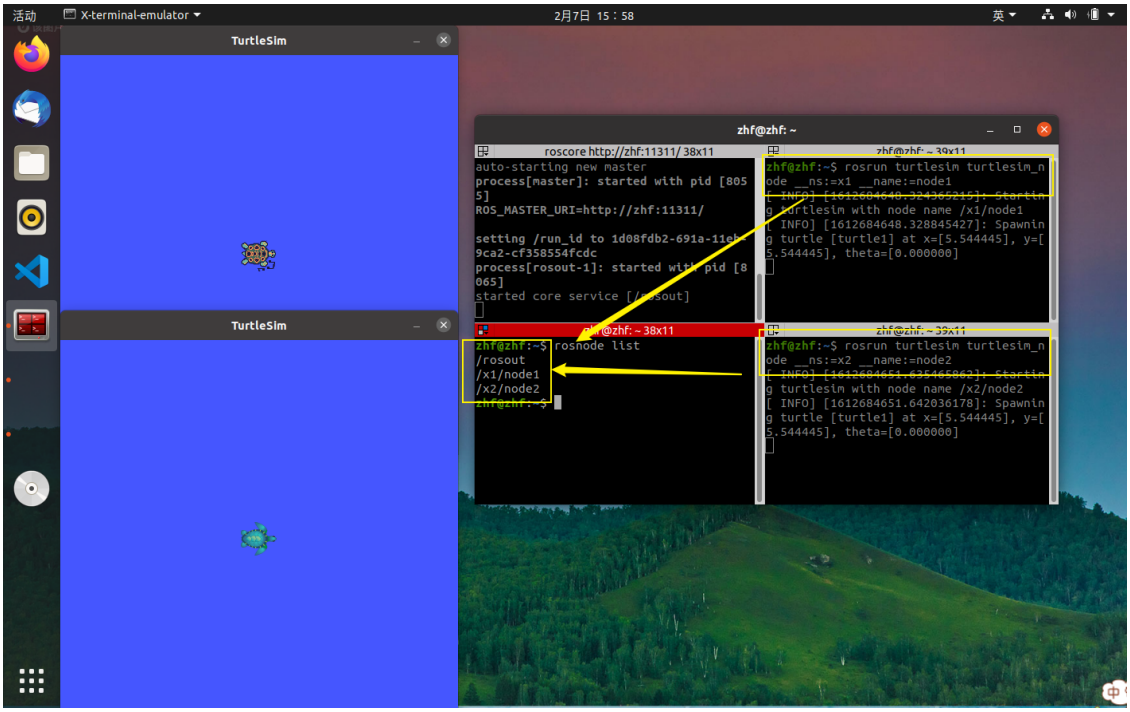
3. rosrun命名空间与名称重映射叠加

> 语法: rosrun 包名 节点名 __ns:=新名称 __name:=新名称

```
rosrun turtlesim turtlesim_node __ns:=/xxx __name:=tn
```

- 运行结果：用`rosnode list`查看

```
/xxx/tn
```



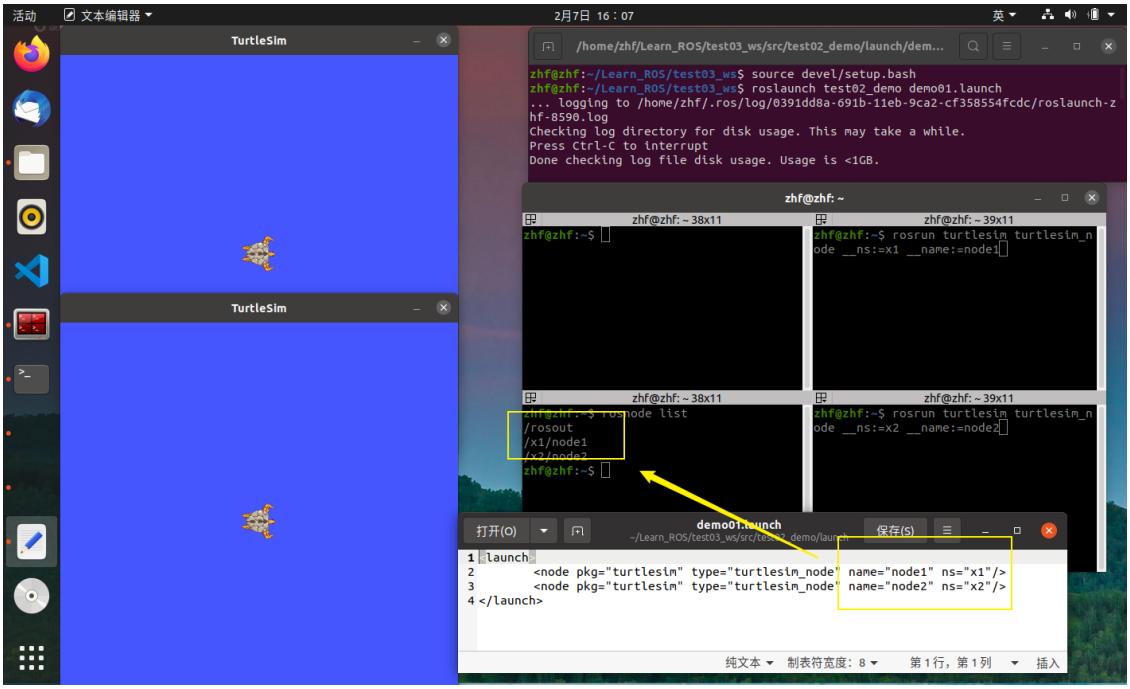
2.launch文件设置命名空间与重映射（起别名）

> 语法：
>
> `<!-- ns="xxx" 在指定命名空间 xxx 中启动节点 -->`
>
> `<!-- name="节点名称" 节点名称(在 ROS 网络拓扑中节点的名称) -->`

```
<launch>
    <node pkg="turtlesim" type="turtlesim_node" name="node1" ns="x1"/>
    <node pkg="turtlesim" type="turtlesim_node" name="node2" ns="x2"/>
</launch>
```

- 运行结果：用`rosnode list`查看

```
/rosout
/x1/node1
/x2/node2
```



3.编码设置命名空间与重映射（起别名）

1. 名称设别名

> 语法：为节点名随机添加一个后缀
>
> 核心代码: `ros::init(argc,argv,"zhangsan",ros::init_options::AnonymousName);`

2. 设置命令空间

> 语法：cpp

```cpp
std::map<std::string, std::string> map;
map["__ns"] = "xxxx";
ros::init(map,"wangqiang");
```

五、话题名称重名

1.rosrun设置重映射

1. 重映射

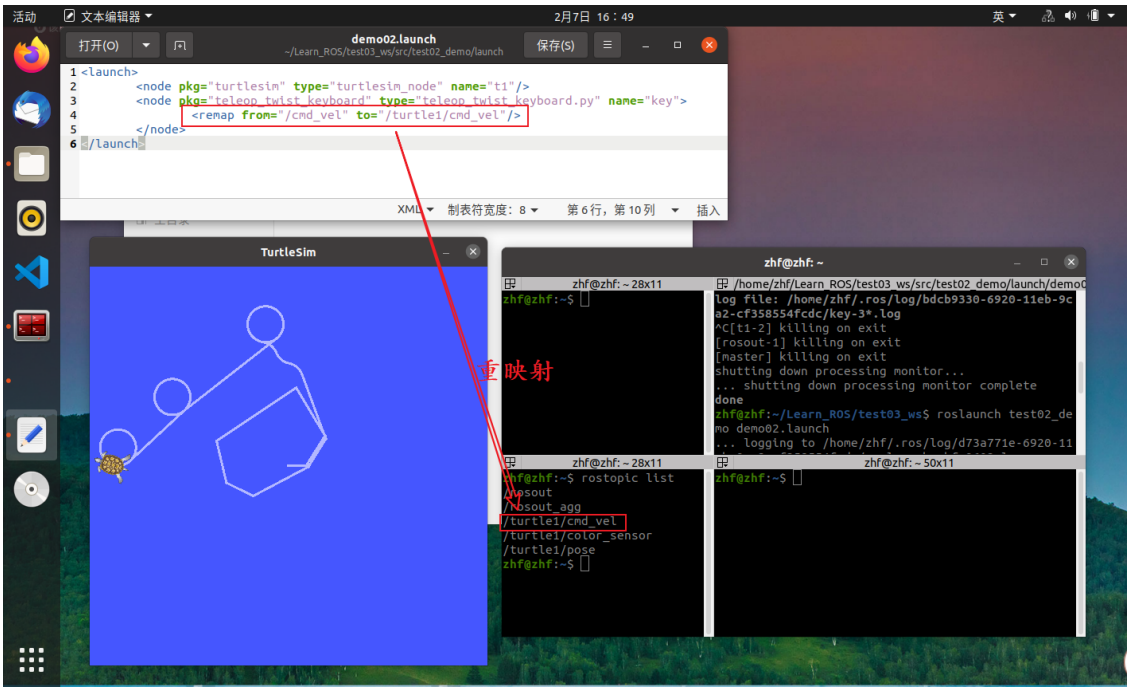
> 语法：rosrun名称重映射语法: rorun 包名 节点名 话题名:=新话题名称

```
rosrun turtlesim turtlesim_node /turtle1/cmd_vel:=/cmd_vel
```

2.launch文件设置重映射

> 语法：

```
<launch>
    <node pkg="turtlesim" type="turtlesim_node" name="t1"/>
    <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py" name="key">
        <remap from="/cmd_vel" to="/turtle1/cmd_vel"/>
    </node>
</launch>
```



3.编码设置命名空间与重映射

话题的名称与节点的命名空间、节点的名称是有一定关系的，话题名称大致可以分为三种类型：

- 全局(话题参考ROS系统，与节点命名空间平级)
- 相对(话题参考的是节点的命名空间，与节点名称平级)
- 私有(话题参考节点名称，是节点名称的子级)

1. **全局**

> 语法：以`/`开头的名称，和节点名称无关

```
ros::NodeHandle nh;
ros::Publisher pub = nh.advertise<std_msgs::String>("/chatter",1000);
```

- 结果

```
/chatter
```

2. **相对**

> 语法：非`/`开头的名称,参考命名空间(与节点名称平级)来确定话题名称

```
ros::NodeHandle nh;
ros::Publisher pub = nh.advertise<std_msgs::String>("chatter",1000);
```

- 结果：

```
xxx/chatter
```

3. **私有**

> 语法：以`~`开头的名称

```
ros::NodeHandle nh("~");
ros::Publisher pub = nh.advertise<std_msgs::String>("chatter",1000);
```

- 结果：

```
/xxx/hello/chatter
```

- 当全局和私有共同存在时，全局的优先级更高

六、参数名称重名

1.rosrun设置参数

- 设置的是私有的参数

> 语法： rosrun 包名 节点名称 _参数名:=参数值

```
rosrun turtlesim turtlesim_node _A:=100
```

- 结果：

```
/turtlesim/A
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
```

2.launch文件设置参数

第二部分讲过

3.编码设置参数

1. `ros::param` 设置

> 语法：

```
ros::param::set("/set_A",100); //全局,和命名空间以及节点名称无关
ros::param::set("set_B",100); //相对,参考命名空间
ros::param::set("~set_C",100); //私有,参考命名空间与节点名称
```

- 结果：假设设置的 namespace 为 xxx，节点名称为 yyy，使用 rosparam list 查看

```
/set_A
/xxx/set_B
/xxx/yyy/set_C
```

2. `ros::NodeHandle` 设置参数

> 语法：

```
ros::NodeHandle nh;
nh.setParam("/nh_A",100); //全局,和命名空间以及节点名称无关

nh.setParam("nh_B",100); //相对,参考命名空间

ros::NodeHandle nh_private("~");
nh_private.setParam("nh_C",100);//私有,参考命名空间与节点名称
```

- 结果：假设设置的 namespace 为 xxx，节点名称为 yyy，使用 rosparam list 查看

```
/nh_A
/xxx/nh_B
/xxx/yyy/nh_C
```

七、分布式通信

ROS是一个分布式计算环境。一个运行中的ROS系统可以包含分布在多台计算机上多个节点。根据系统的配置方式，任何节点可能随时需要与任何其他节点进行通信。

因此，ROS对网络配置有某些要求：

- 所有端口上的所有机器之间必须有完整的双向连接。
- 每台计算机必须通过所有其他计算机都可以解析的名称来公告自己。

实现

1.准备

先要保证不同计算机处于同一网络中，最好分别设置固定IP，如果为虚拟机，需要将网络适配器改为桥接模式；

2.配置文件修改

分别修改不同计算机的 /etc/hosts 文件，在该文件中加入对方的IP地址和计算机名：

IP地址查看名：`ifconfig`

计算机名称查看：`hostname`

主机端：

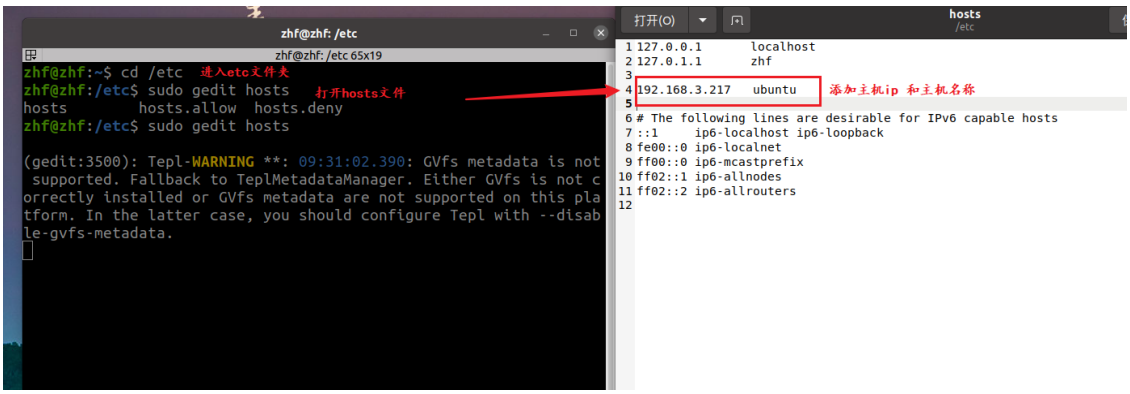
```
从机的IP    从机计算机名
```

从机端：

```
主机的IP    主机计算机名
```

设置完毕，可以通过 ping 命令测试网络通信是否正常。

```
ping 主机ip
```



3.配置主机IP

配置主机的 IP 地址，~/.bashrc 追加

```
export ROS_MASTER_URI=http://主机IP:端口号
export ROS_HOSTNAME=主机IP
```

4.配置从机IP

配置从机的 IP 地址，从机可以有多台，每台都做如下设置，~/.bashrc 追加

```
export ROS_MASTER_URI=http://主机IP:端口号
export ROS_HOSTNAME=从机IP
```

测试

1.主机启动 roscore(必须)

2.主机启动订阅节点，从机启动发布节点，测试通信是否正常

3.反向测试，主机启动发布节点，从机启动订阅节点，测试通信是否正常