将有内在联系的功能包 进行管理
关联不同的功能包

很多个节点 如何启动 ----> .launch 文件 进行启动

掌握元功能包的使用方法

重命名节点 和话题名称的处理方法

node 多个机器的分布式通信

元功能包

如果有多个功能包 实现一个系统,

这些功能包需要都逐一安装马

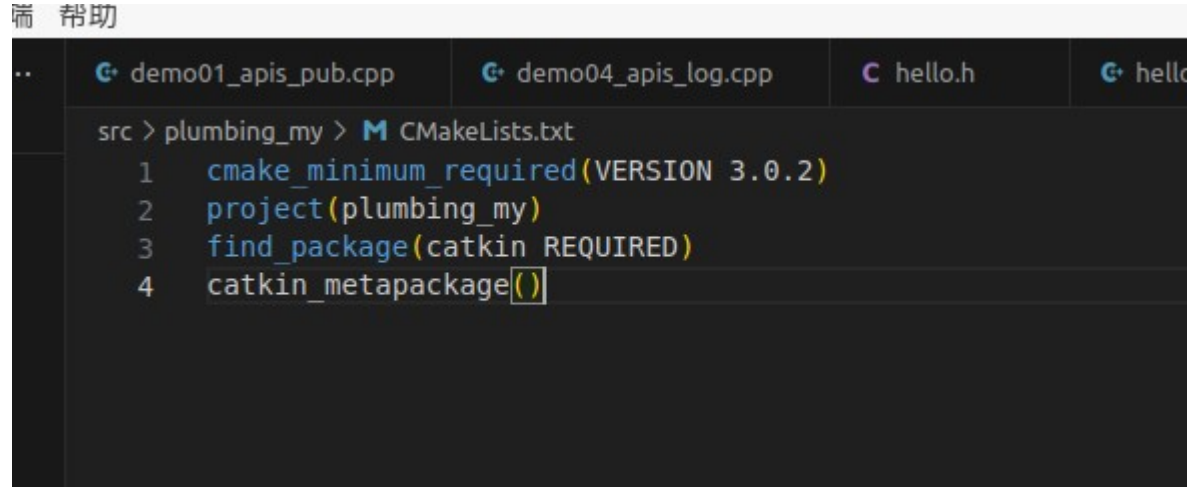
将多个功能包 打包成一个元功能包
将多个功能包组合在一起
一个虚包，将依赖的软件包 组合起来， 理解成一本书的目录索引，告诉我们需要哪些包
这些子包在那里下

sudo apt install ros-noetic-desktop-full
安装这个元功能包，会把依赖也下载下来

创间元功能包
在 package.xml
在创建功能包的时候 没有添加任何依赖如 roscpp std_msgs
在这里说我们输出的依赖 需要 53-55 这三个功能包的名字
再加上 59 行 <metapackage />

```
48    <!--   <test_depend>gtest</test_depend> -->
49    <!-- Use doc_depend for packages you need only for building documentation: -->
50    <!--   <doc_depend>doxygen</doc_depend> -->
51    <buildtool_depend>catkin</buildtool_depend>
52
53    <exec_depend>plumbing_pub_sub</exec_depend>
54    <exec_depend>plumbing_server_client</exec_depend>
55    <exec_depend>plumbing_param_Server</exec_depend>
56    <!-- The export tag contains other, unspecified, tags -->
57    <export>
58      <!-- Other tools can request additional information be placed here -->
59      <metapackage />
60    </export>
61  </package>
62
```

cmakelist.txt 的文件按照这个模式写就可以了，保留这四行
cmakelist 不可以用换行



赵虚左老师给了几个例子
展示了 package.xml 文件 和 cmakelist.list 文件

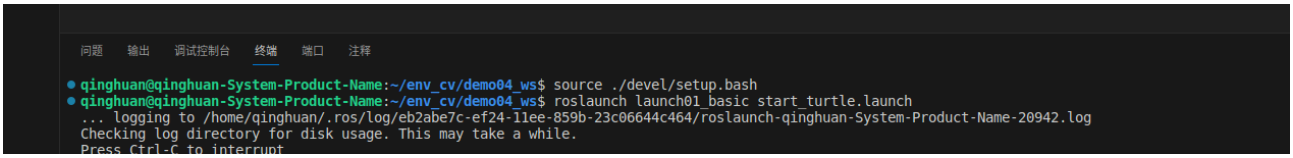
官方有参考在 4.1 赵虚左老师的课件

node 节点管理的 launch 文件

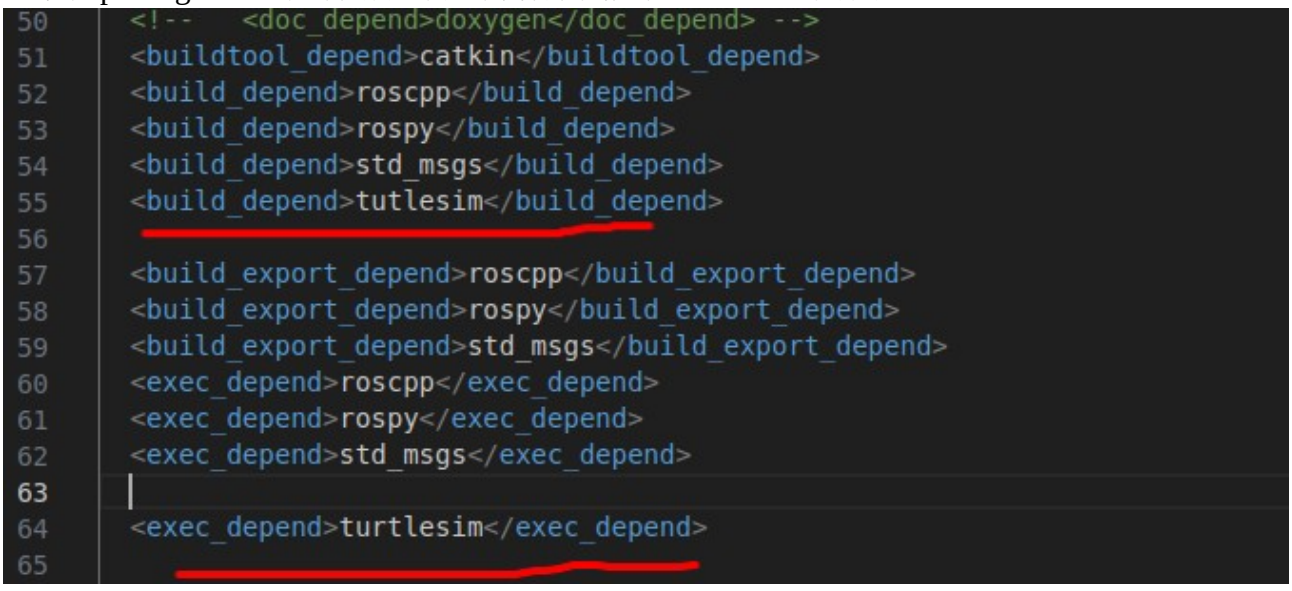
在功能包下创建一个 launch 文件夹
之后创建 launch 文件
可以自己启动 roscore 文件 他会判断 roscore 是否启动 没有启动他会自己启动

这个是 launch 文件和每个参数意义

文件的调用

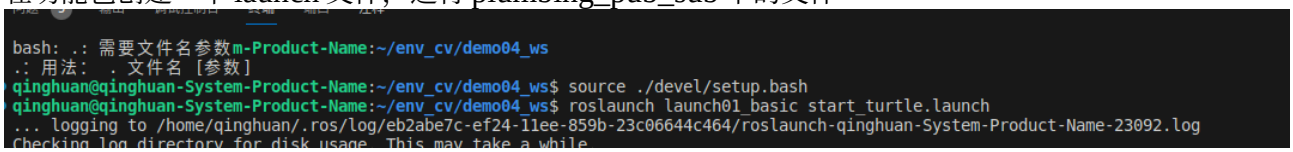


launch02_basic
是在创建依赖的时候，我故意没有加入 turtlesim 这个功能包
之后在 package.xml 的地方 添加了如下两行代码实现 turtlesim 的导入



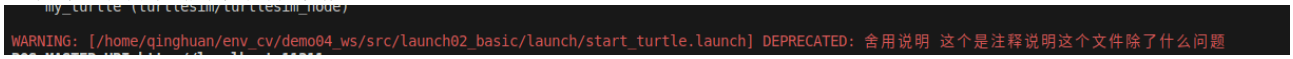
第 55 行 和第 64 行

在功能包创建一个 launch 文件，运行 plumbing_pub_sub 下的文件



在这里就算把我自己的功能包添加进去了

launch 文件
<launch>是所有文件的根标签
deprecated="弃用说明"
运行后的效果 就算给你个信息



node 标签
用于指定 node 节点
roslaunch 不能保证按照 node 的生成顺序启动

pkg="包名"

节点所属的包

- type="nodeType" 可执行文件的名字

  节点类型(与之相同名称的**可执行文件**)

- name="nodeName" 网络的节点名字 ros::init(argc,argv,"节点的名字")

  节点名称(在 ROS 网络拓扑中节点的名称)

- args="xxx xxx xxx" (可选) rosrun 功能包名 功能包的别名 add_execute 第一个参数，之后后面岁就传参数 空格 分割参数 这个 args 就不同参数空格分割

  将参数传递给节点

- machine="机器名" 启动不同设备上的节点

  在指定机器上启动节点

- respawn="true | false" (可选) 关闭这个节点后可以自动帮你重启

  如果节点退出，是否自动重启

- respawn_delay=" N" (可选)

  如果 respawn 为 true, 那么延迟 N 秒后启动节点

- required="true | false" (可选)

  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

- ns="xxx" (可选)

在指定命名空间 xxx 中启动节点

- clear_params="true | false" (可选)

  在启动前，删除节点的私有空间的所有参数

- output="log | screen" (可选)

  日志发送目标，可以设置为 log 日志文件，或 screen 屏幕,默认是 log

include 是包含的意思

重复启动两个节点,

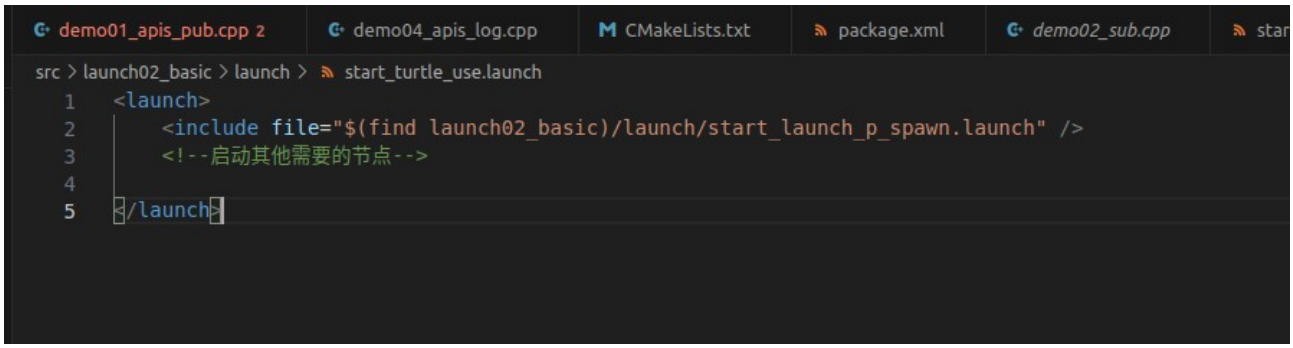
将另一个 xml 格式的 launch 文件导入当前文件

file 文件的路径

ns 是节点的命名空间

$(find 功能包名)功能包下 launch 文件的地址

子标签 env 环境便利 arg 传入的参数



remap 话题重命名

```
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
qinghuan@qinghuan-System-Product-Name:~/桌面$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
/turtle2/cmd_vel
qinghuan@qinghuan-System-Product-Name:~/桌面$
```

这就算 remap 重命名了话题名称 及节点间的通信方法 依据话题 话题名称会进行匹配通信

安装所需要的功能包

rosrun teleop_twist_keyboard teleop_twist_keyboard.py

src > launch02_basic > launch > start_turtle.launch

```
1  <launch deprecated="舍用说明 这个是注释说明这个文件除了什么问题">
2  <!--启动节点 -->
3  <node pkg="turtlesim" type="turtlesim_node" name="my_turtle"  output="screen" >
4      <remap from="/turtle1/cmd_vel" to="/cmd_vel"/>
5  </node>
6  <!--pgk是包名   type是具体的文件的别名  节点类型 name 我设置的节点名称 output日志输出到那里  -->
7  <node pkg="turtlesim" type="turtle_teleop_key" name="my_key"  output="screen" />
8
9  </launch >
```

<paran> 在参数服务器上设置参数
设置在 launch 标签下
设置在 node 标签下 在 node 下 会在 param 下 有个 node 的名称 相当于 是命名空间
name 设置参数名子 value 指向 param 的值 type value 的类型 value ="值"
type 是可选的 可以自动解析

4.2.5 launch文件标签之param

`<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。

1.属性

- name="命名空间/参数名"

  参数名称，可以包含命名空间

- value="xxx" (可选)

  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" (可选)

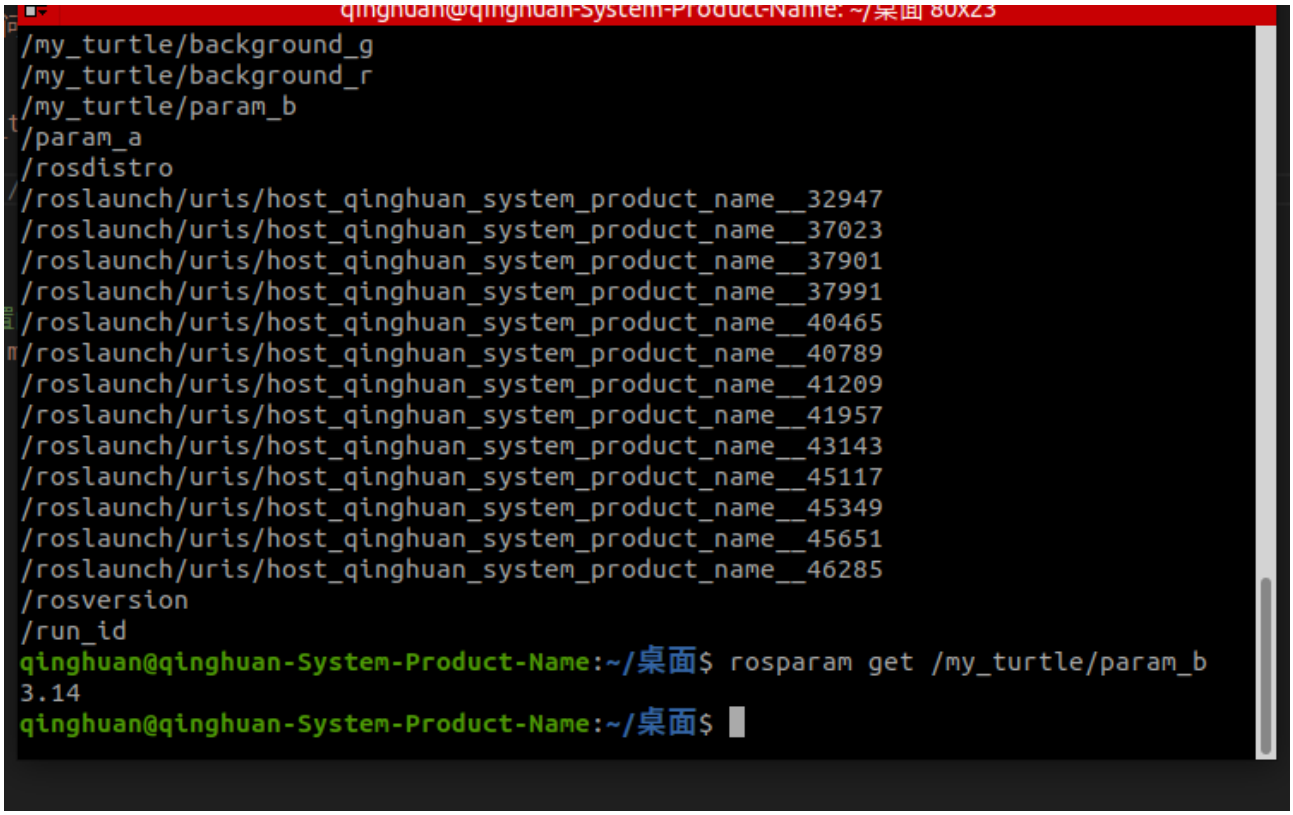
  指定参数类型，如果未指定，roslaunch 会尝试确定参数类型，规则如下:

  - 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

2.子级标签

- 无

两个格式 一个是 node 名字下的 param 是相当于给 param 参数 设置了作用域 前面相当于是 namespace 是个节点的名字 就算 launch 的 name
在 launch 的标签下定义 就算 一个全局的变量


qinghuan@qinghuan-System-Product-Name: ~/桌面 80x23
/my_turtle/background_g
/my_turtle/background_r
/my_turtle/param_b
/param_a
/rosdistro
/roslaunch/uris/host_qinghuan_system_product_name__32947
/roslaunch/uris/host_qinghuan_system_product_name__37023
/roslaunch/uris/host_qinghuan_system_product_name__37901
/roslaunch/uris/host_qinghuan_system_product_name__37991
/roslaunch/uris/host_qinghuan_system_product_name__40465
/roslaunch/uris/host_qinghuan_system_product_name__40789
/roslaunch/uris/host_qinghuan_system_product_name__41209
/roslaunch/uris/host_qinghuan_system_product_name__41957
/roslaunch/uris/host_qinghuan_system_product_name__43143
/roslaunch/uris/host_qinghuan_system_product_name__45117
/roslaunch/uris/host_qinghuan_system_product_name__45349
/roslaunch/uris/host_qinghuan_system_product_name__45651
/roslaunch/uris/host_qinghuan_system_product_name__46285
/rosversion
/run_id
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosparam get /my_turtle/param_b
3.14
qinghuan@qinghuan-System-Product-Name:~/桌面$

rosparam 的基本使用

4.2.6 launch文件标签之rosparam

`<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。

1.属性

- command="load | dump | delete" (可选，默认 load)

  加载、导出或删除参数

- file="$(find xxxxx)/xxx/yyy...."

  加载或导出到的 yaml 文件

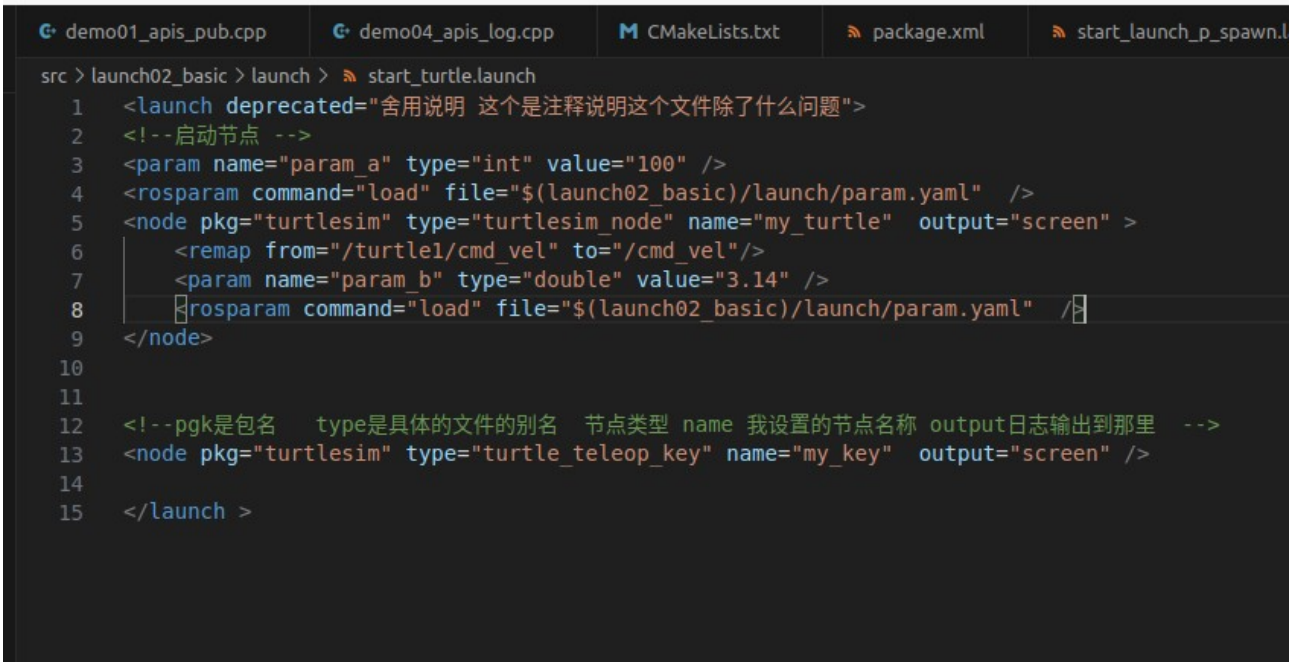
- param="参数名称"

- ns="命名空间" (可选)

2.子级标签

- 无

在 launch 下定义
在 node 下定义
有了一个 namespace 就算 这个 node 的 name 会在前头



src > launch02_basic > launch > ! param.yaml

```yaml
bgr: 100
bgg: 50
bgb: 255
#: 冒号后面必须有空格
```

设置参数

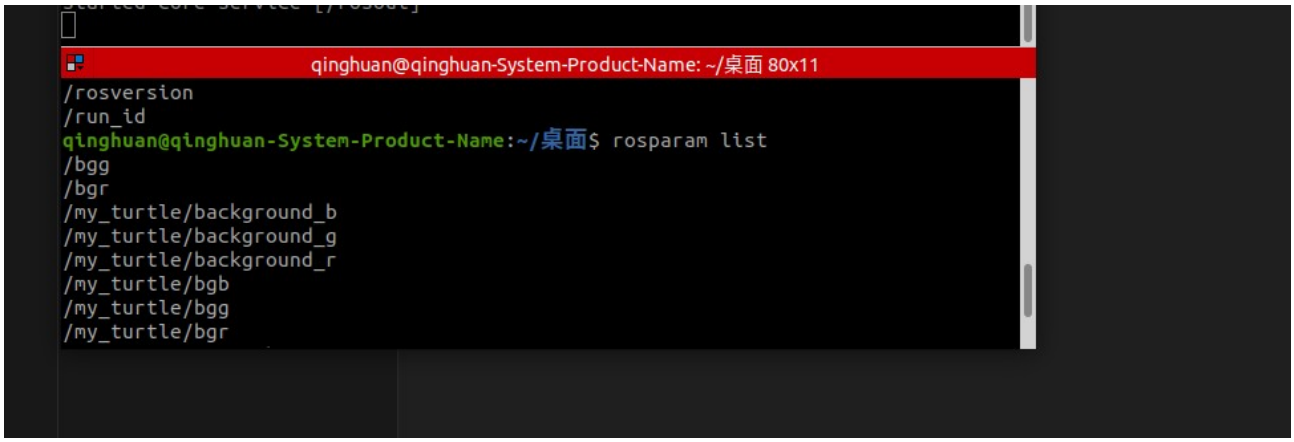
```
qinghuan@qinghuan-System-Product-Name:~/
/bgb
/bgg
/bgr
/my_turtle/background_b
/my_turtle/background_g
/my_turtle/background_r
/my_turtle/bgb
```

dump 导出参数 有可能不符合预期，是这样的 这个 node 和 param 的设在有一个先后顺序，
可以单独用一个 launch 文件导出文件

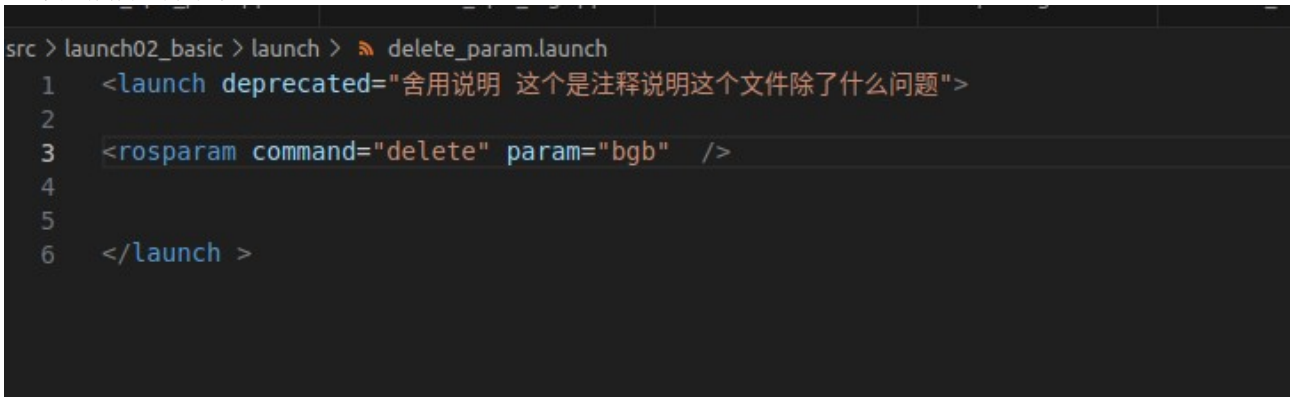
> launch02_basic > launch > start_turtle.launch

```
<launch deprecated="舍用说明 这个是注释说明这个文件除了什么问题">
<!--启动节点 -->
<param name="param_a" type="int" value="100" />
<rosparam command="load" file="$(find launch02_basic)/launch/param.yaml"  />
<rosparam command="dump" file="$(find launch02_basic)/launch/param1.yaml"  />
<node pkg="turtlesim" type="turtlesim_node" name="my_turtle"  output="screen" >
    <remap from="/turtle1/cmd_vel" to="/cmd_vel"/>
    <param name="param_b" type="double" value="3.14" />
    <rosparam command="load" file="$(find launch02_basic)/launch/param.yaml"  />
</node>


<!--pgk是包名   type是具体的文件的别名  节点类型 name 我设置的节点名称 output日志输出到那里  -->
<node pkg="turtlesim" type="turtle_teleop_key" name="my_key"  output="screen" />

</launch >
```

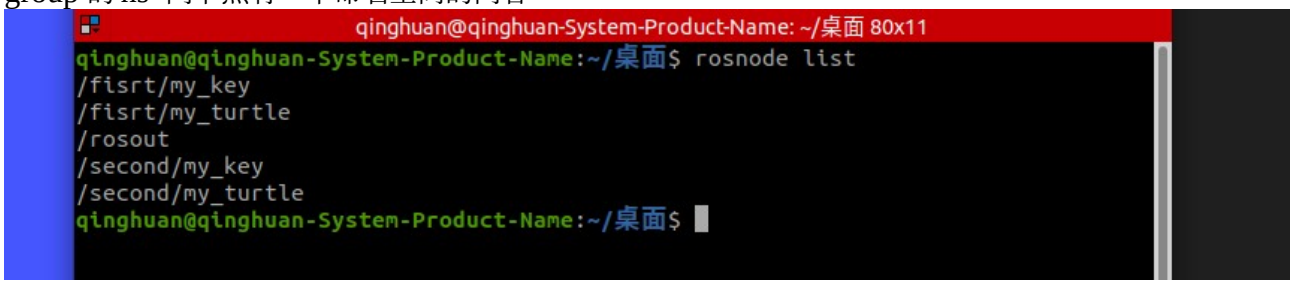
我又建立一个 launch 文件
这个是在第一个 start_turtle.launch 运行后运行
删除一个 param 这个是效果



这个是删除的代码



group 的 ns 向节点有一个命名空间的内容



arg 传递参数

`<arg>` 标签是用于动态传参，类似于函数的参数，可以

1.属性

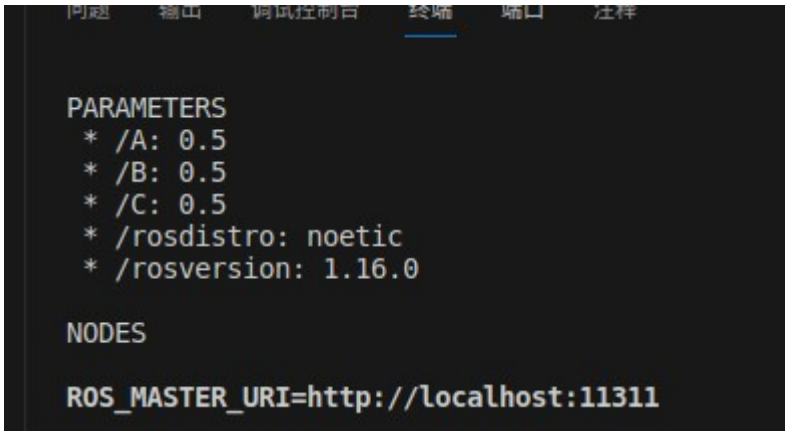
- name="参数名称"
- default="默认值" (可选)
- value="数值" (可选)

  不可以与 default 并存
- doc="描述"

  参数说明

2.子级标签

设置一组参数的显示



```
launch02_basic > launch > arg.launch
<launch>
    <!--设置多个参数 -->
    <!-- <param name="A" type="double" value="0.5" />
    <param name="B" type="double" value="0.5" />
    <param name="C" type="double" value="0.5" /> -->
    <arg name="car_length"  default="0.5" />
    <!--这个是使用arg的参数-->
    <param name="A" type="double" value="$(arg car_length)" />
    <param name="B" type="double" value="$(arg car_length)" />
    <param name="C" type="double" value="$(arg car_length)" />
</launch>
```

这个是 设置参数 a 和 b 都是 arg 的 $(arg 参数的名字)

ros 工作空间覆盖

在不同工作空间内 存在同名的功能包
ros 会自定义工作空间， 特定的工作空间 功能包 不能可以重名
自定义的功能包 和官方的功能包 重名
功能包 A 有 turtlesim 功能包 b 也有 turtlesim
那我调用的时候 是调用的哪个

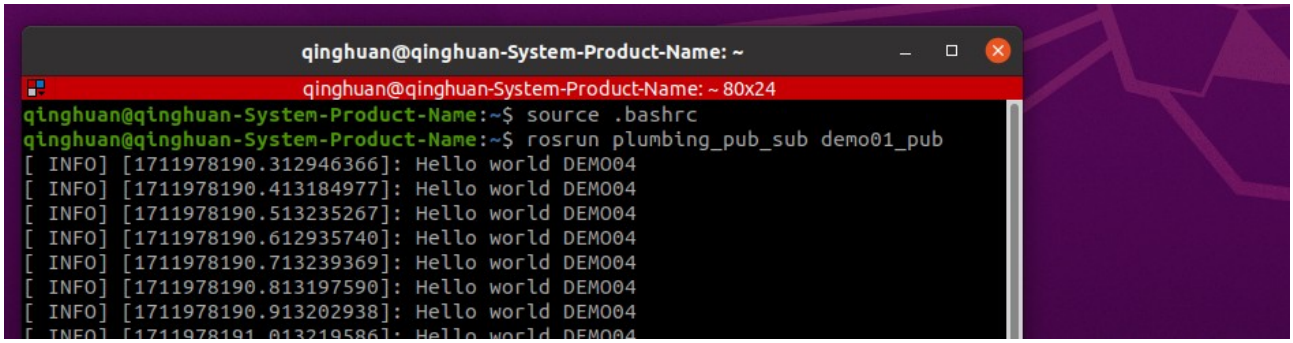
ros 的配置文件中多个环境变量

后刷新 的优先级高于前面的 demo01 的
两个文件的环境是不一样的 在 demo04 _ws 的空间 我们这里 ros_info 特意添加了 demo04 之后在全局运行 我们可以看到是调用 后写进去的，后写进去的优先级会更高

```
116   fi
117 fi
118 source /opt/ros/noetic/setup.bash
119 source /home/qinghuan/demo03_ws/devel/setup.bash
120 source /home/qinghuan/env_cv/demo04_ws/devel/setup.bash
```

sh ▾ 制表符宽度：8 ▾

试验完毕后 我把 119 120 删除了 避免污染环境变量



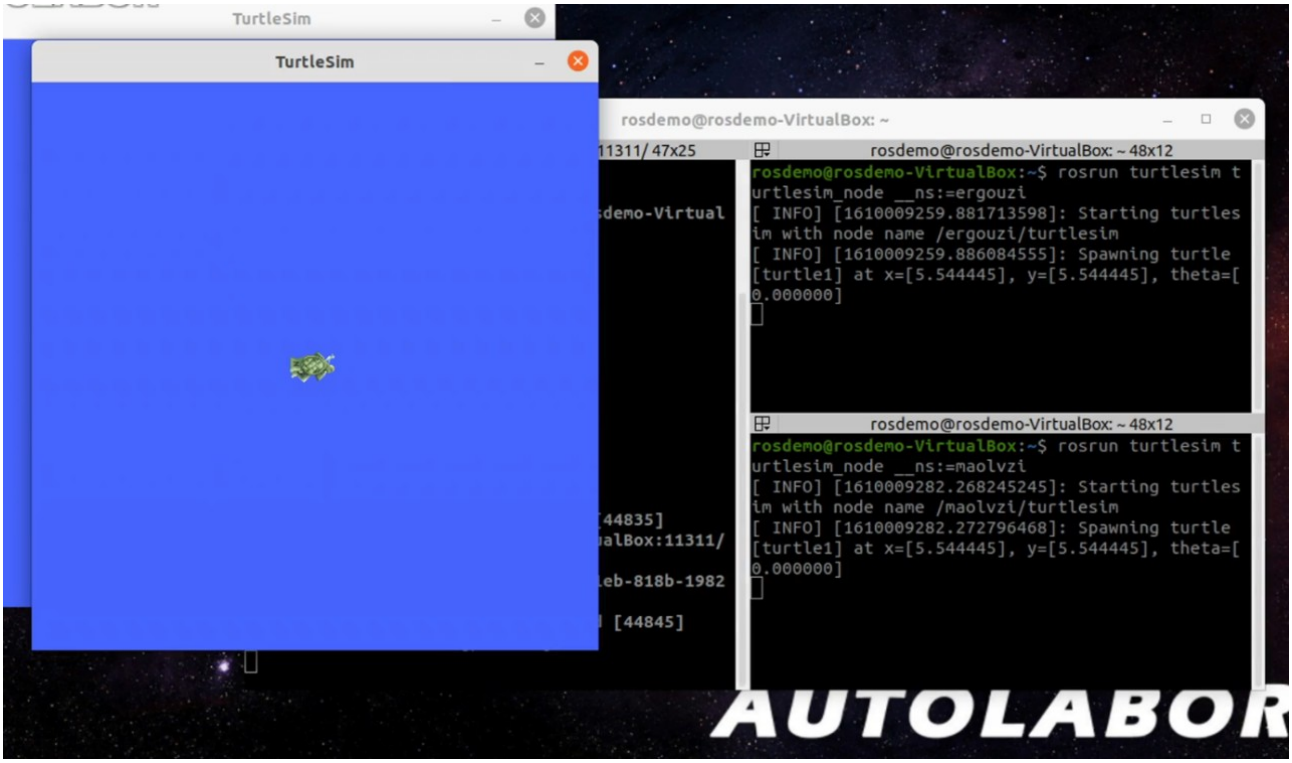
重命名包 先 source A 再 source B 的功能包 两个功能包 的

可能存在 bug 问题 需要避免

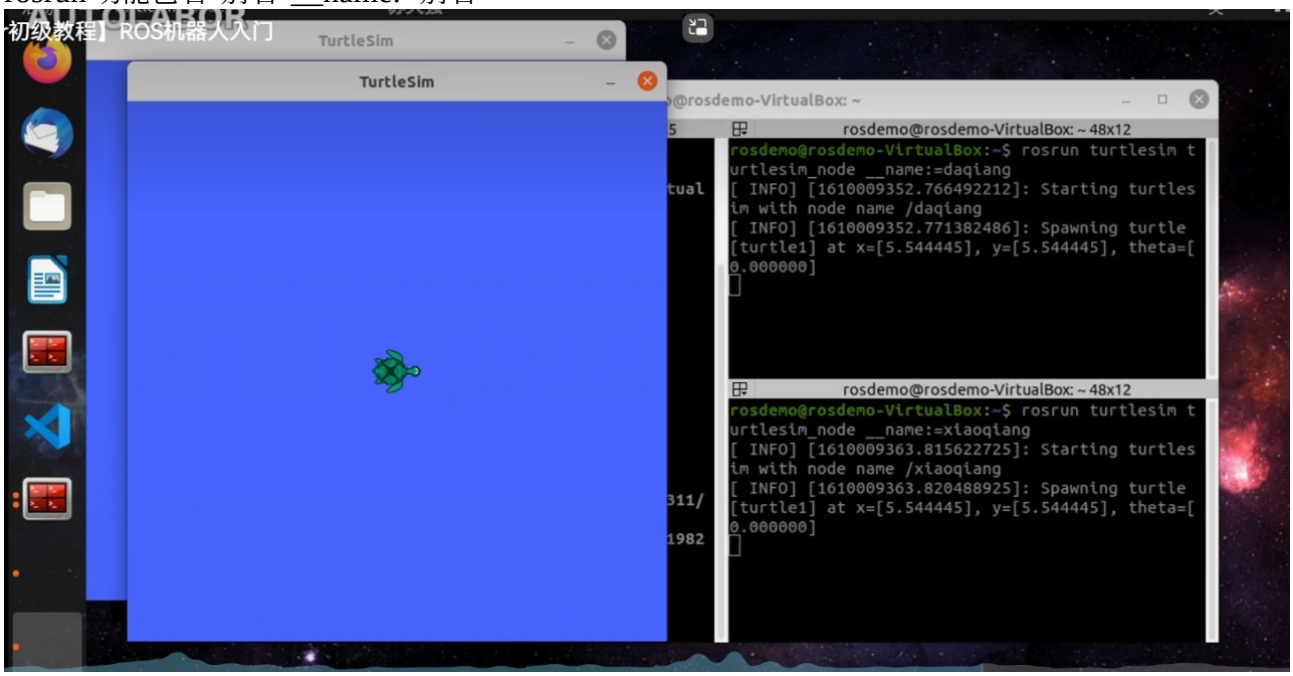
node 的重命名

解决的办法
起别名 /加前缀
rosrun 执行的时候 设置空间
launch 文件设置空间
编码设置空间

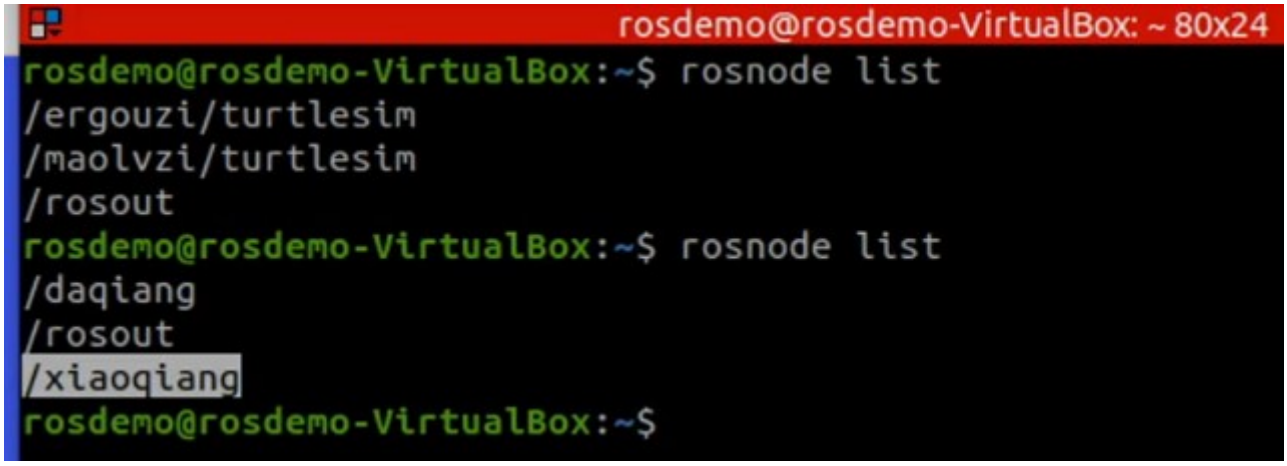
__ns:= 命名空间 通过 这个方法给节点加一个命名空间



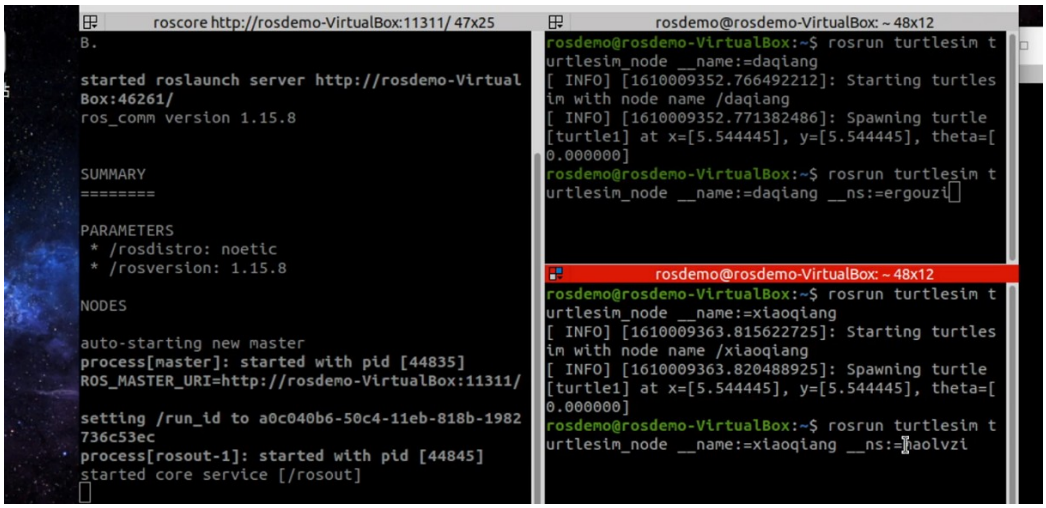
rosrun 功能包名 别名 __name:=别名



我们的名字是不一样的



效果叠加



重命名使用
1、第一个是直接使用
2、第二个是名称重映射
3、第三个是 ns 新加了一个命名空间
4、 第四个是命名空间和 name 重映射

start_turtle.launch

src > rename01_node > launch > start_turtle.launch

```
<!-- 需要启动多个乌龟GUI节点 -->
<launch>
    <node pkg="turtlesim" type="turtlesim_node" name="turtlesim"  />
    <!-- 名称重映射 -->
    <node pkg="turtlesim" type="turtlesim_node" name="t1"  />
    <!-- 命名空间 -->
    <node pkg="turtlesim" type="turtlesim_node" name="turtlesim" ns="ergouzi"  />
    <!-- 命名空间 + 名称重映射 -->
    <node pkg="turtlesim" type="turtlesim_node" name="t2" ns="maolvzi"  />

</launch>
```

ros::init(argc,argv,"节点名字",ros::init_options::AnnoymousName）

名称重命名的映射

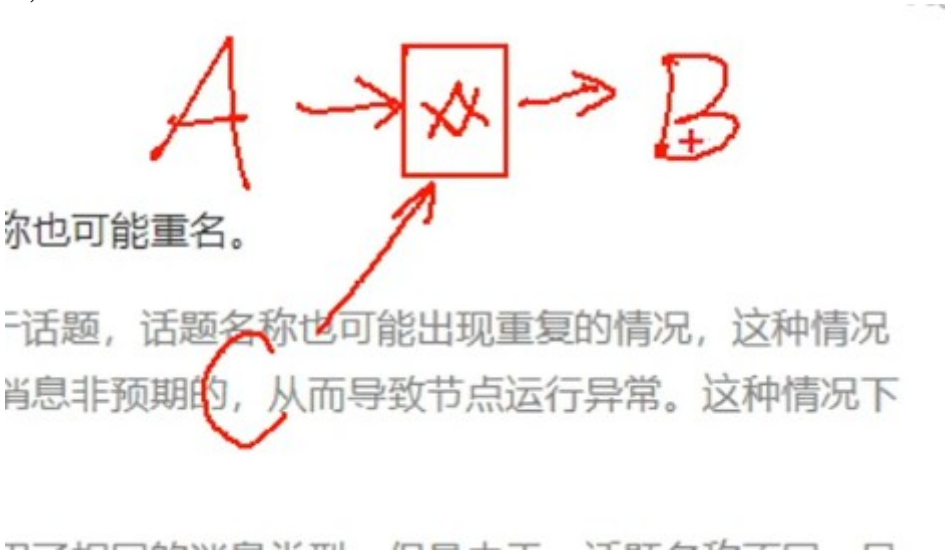
这个方法会自动加入一个 ns 在这个 node 名字为 wangqiang

2.1命名空间设置

核心代码

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

话题的重命名
不想重名 重名了
发布者可以多个， 多个向话题发送数据 B 可能不想接受这个数据

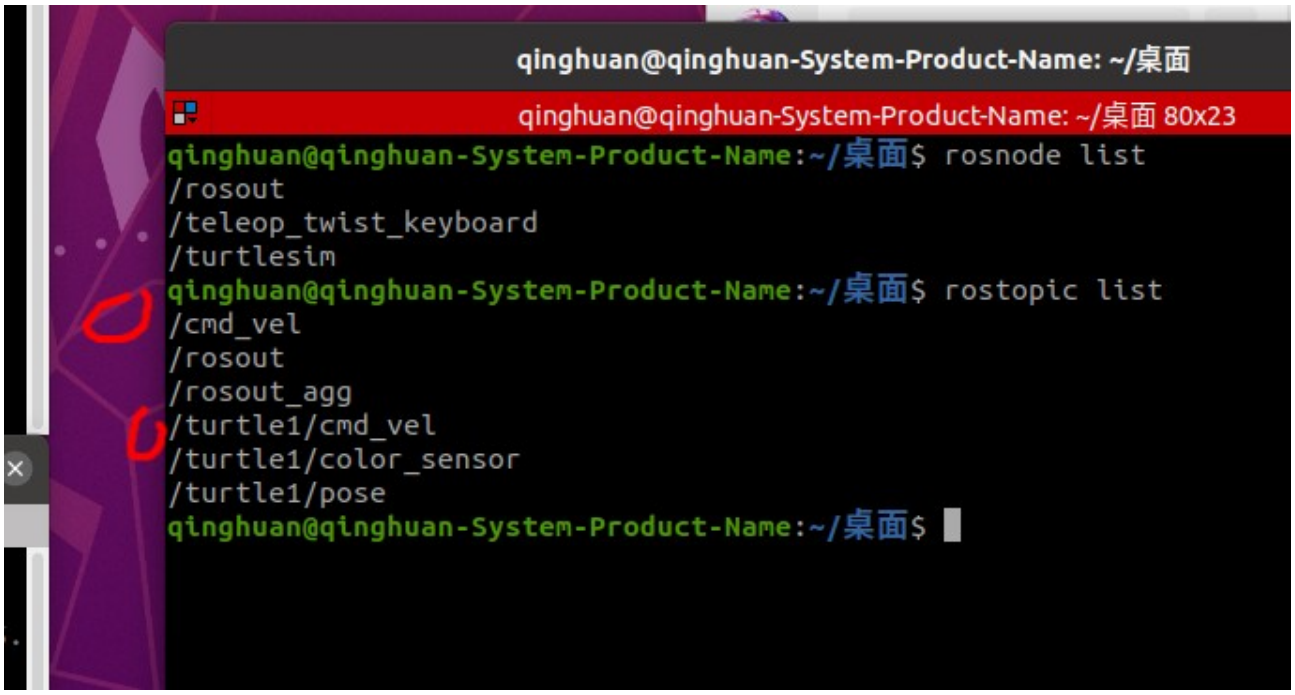


想重名不重名

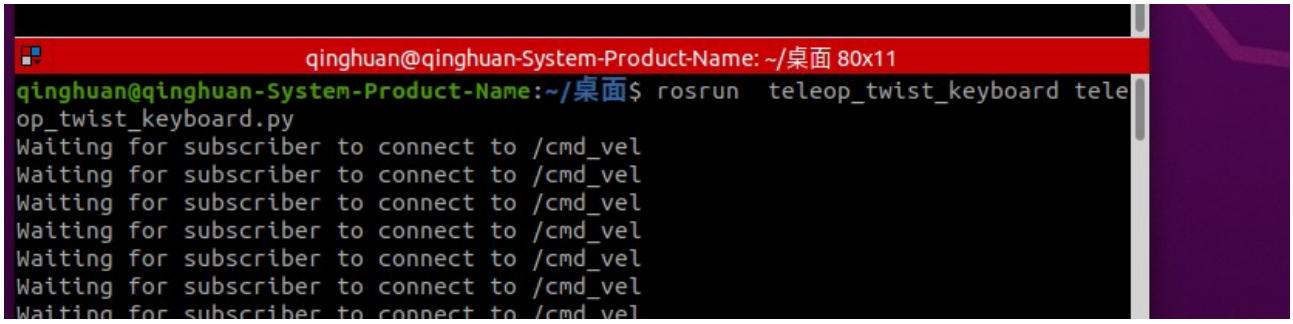
重映射 和 命名空间（全局/相对/私有)在 4.5 介绍

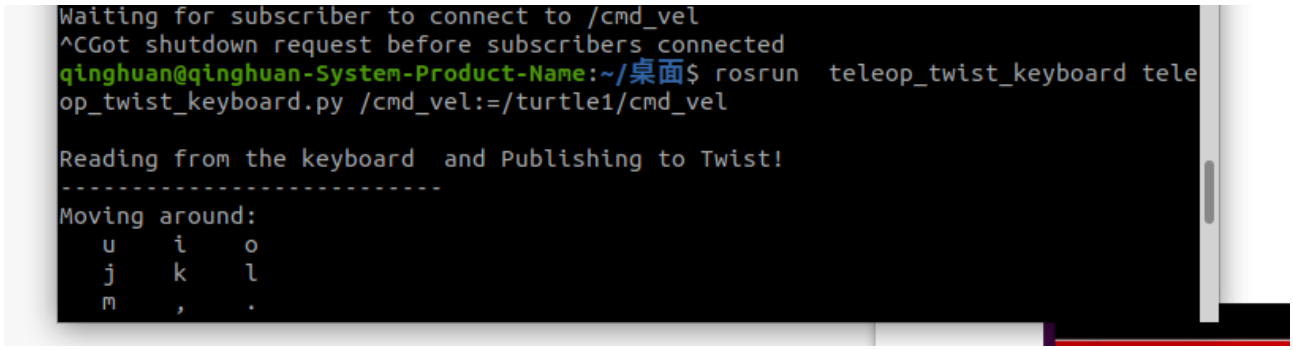
rosrun 命令 launch 文件 编码实现

rosrun teleop_twist_keyboard teleop_twist_keyboard.py /cmd_vel:=/turtle1/cmd_vel

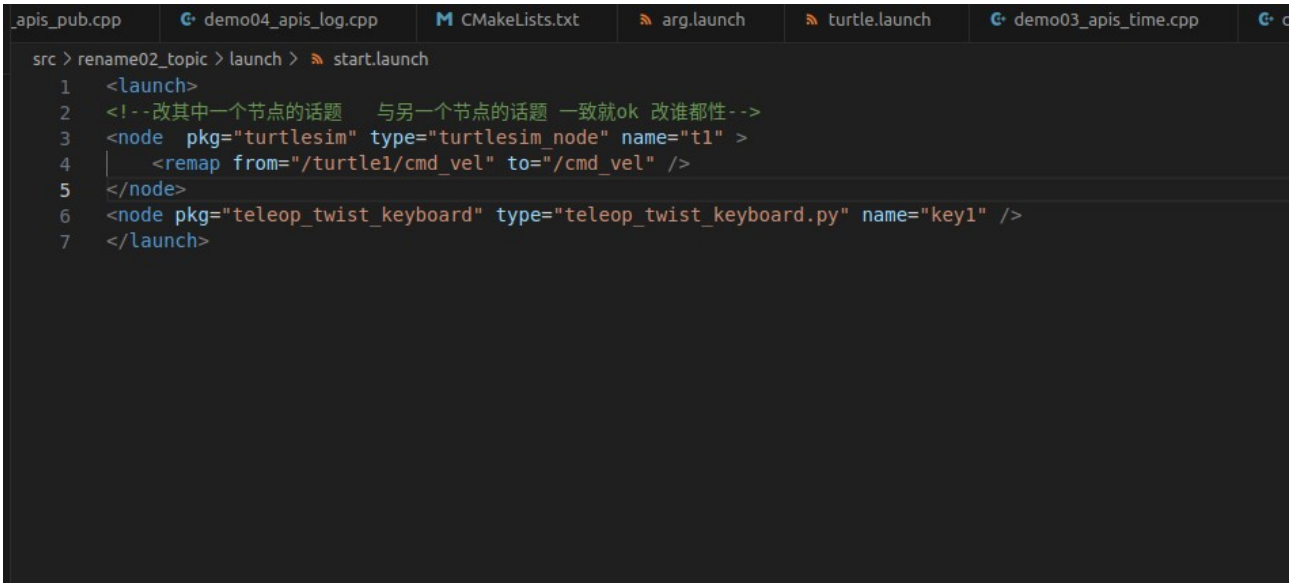


话题重命名了





通过 launch 实现 话题 topic 的重新映射



编码 避免话题重名

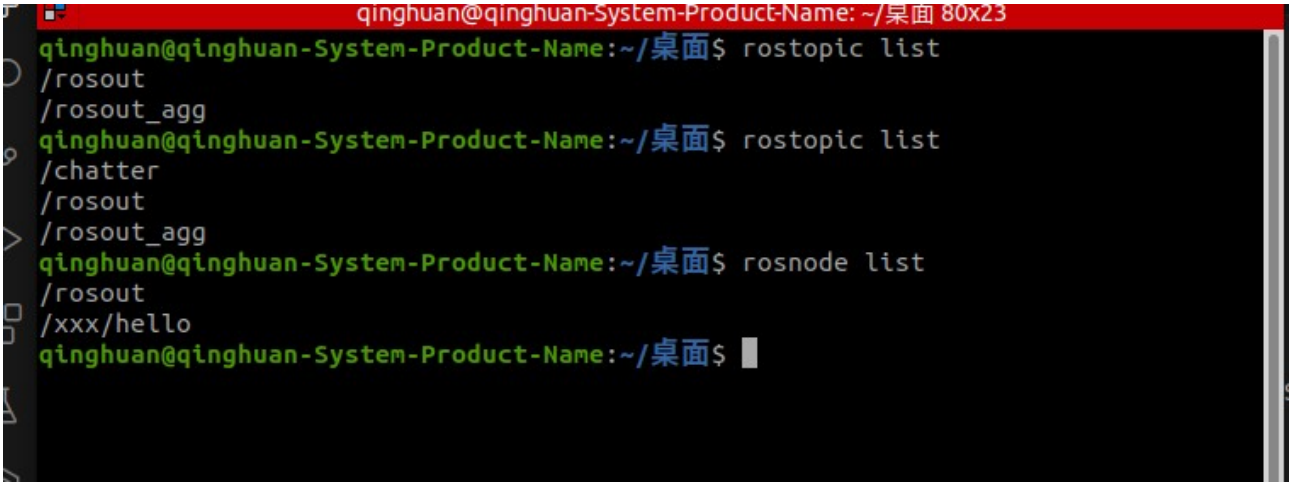
1、ros::int(argc,argv,"节点的名字")
话题名字 分为： 全局 相对 私有
全局： 与节点命名空间评选
相对: 与节点名称评级
私有: 当前节点下的话题

节点名称
/ergouzi/wangqiang
/是代表根
ergouzi 命名空间

/wangqiang 节点名字

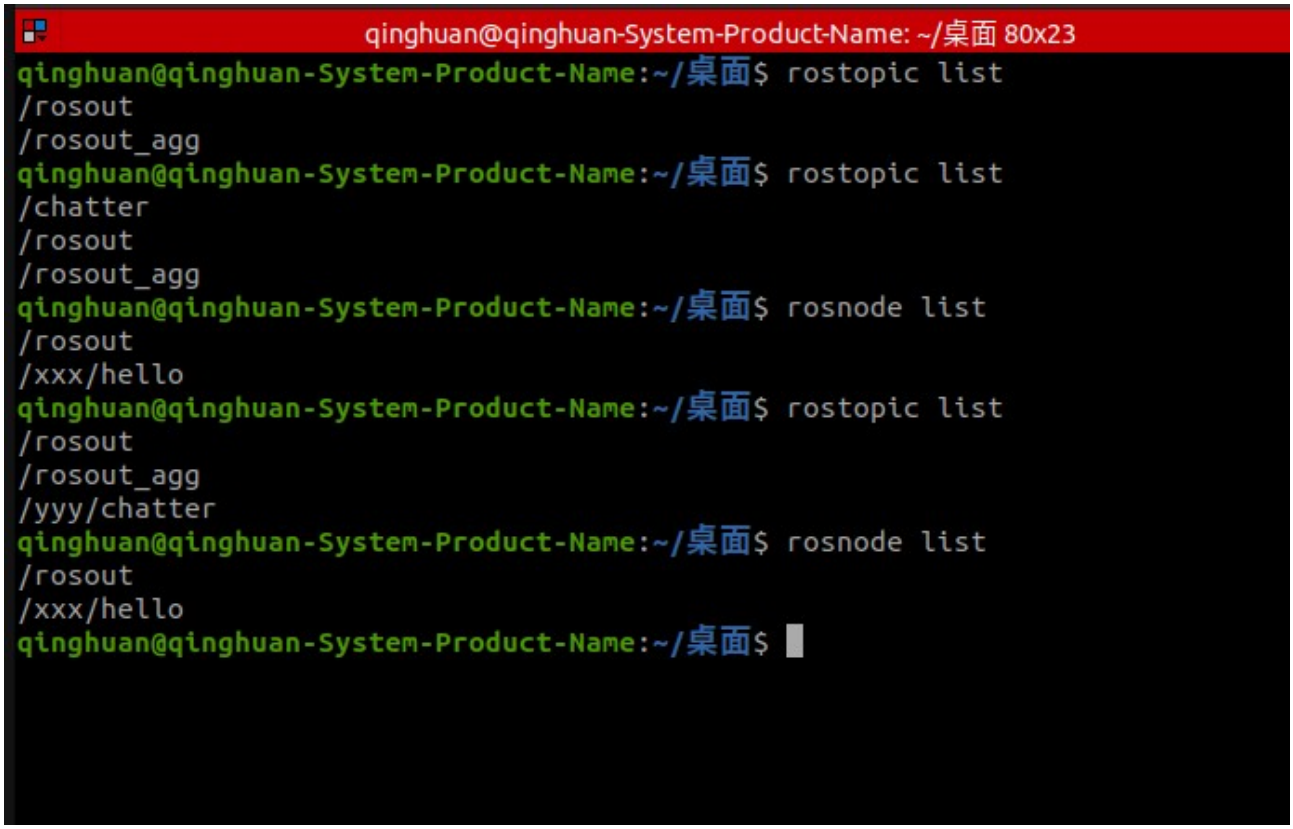
/liaotian /话题 参考根目录
/ergouzi/话题 参考命名空间
/ergouzi/wangqiang 参考 节点名字

全局话题
在话题前面加上/ 反斜杠代表根目录
node 会有一个 xxx 的命名空间
但是我们 topic 是基于全局的

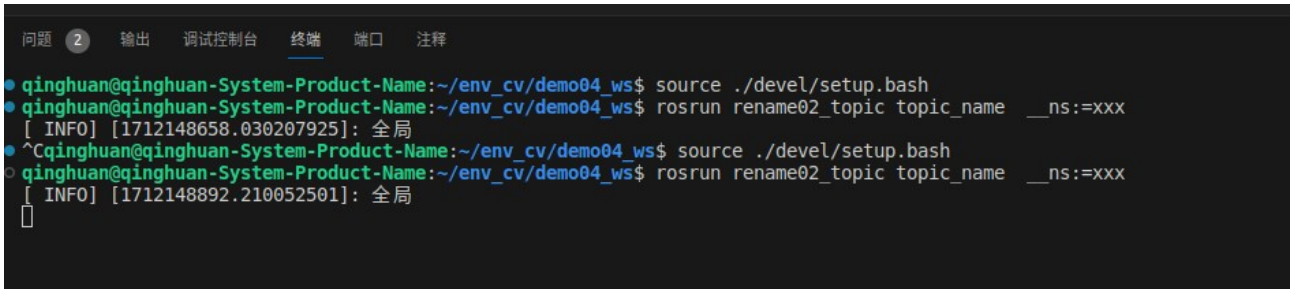



```cpp
#include"ros/ros.h"
#include"std_msgs/String.h"


int main(int argc, char*argv[])
{
    setlocale(LC_ALL, "");
    ros::init(argc, argv,"hello");
    ros::NodeHandle nh;
    //全局话题
    ros::Publisher pub = nh.advertise<std_msgs::String>("/chatter",1000);
    ROS_INFO("全局");
    while(ros::ok())
    {

    }

    return 0;
}
```

给话题设置 命名空间
设置了全局话题的命名
node 的命名空间是设置成 xxx



__ns:=xxx 这是运行时候给 node 添加了 namespace



.

相对命名空间
给 node 节点一个 xxx 的 namespace 之后
topic 和 node 是平级 都是相对于 node 的 namespace 进行定义的

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

```
[1]+  已停止               rosrun rename02_topic topic_name __ns:=xxx
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun rename02_topic topic_name  __ns:=xxx
[ INFO] [1712193574.160279682]: 全局
```

```
roscore http://qinghuan-System-Product-Name:11311/ 80x11

NODES

auto-starting new master
process[master]: started with pid [5866]
ROS_MASTER_URI=http://qinghuan-System-Product-Name:11311/

setting /run_id to 5eab3fee-f221-11ee-a4ea-2b8ea106113b
process[rosout-1]: started with pid [5876]
started core service [/rosout]
```

```
qinghuan@qinghuan-System-Product-Name: ~/桌面 80x11
qinghuan@qinghuan-System-Product-Name:~/桌面$ rostopic list
/rosout
/rosout_agg
/xxx/chatter
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosnode list
/rosout
/xxx/hello
qinghuan@qinghuan-System-Product-Name:~/桌面$
```

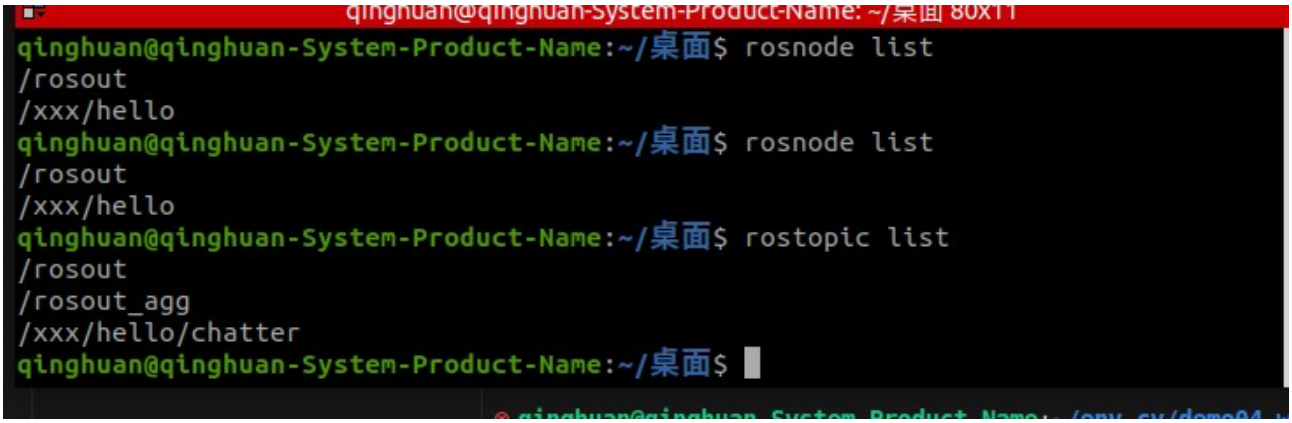
在 xxx 就算 node 的命名空间 我们再给出一个相对的命名方法

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

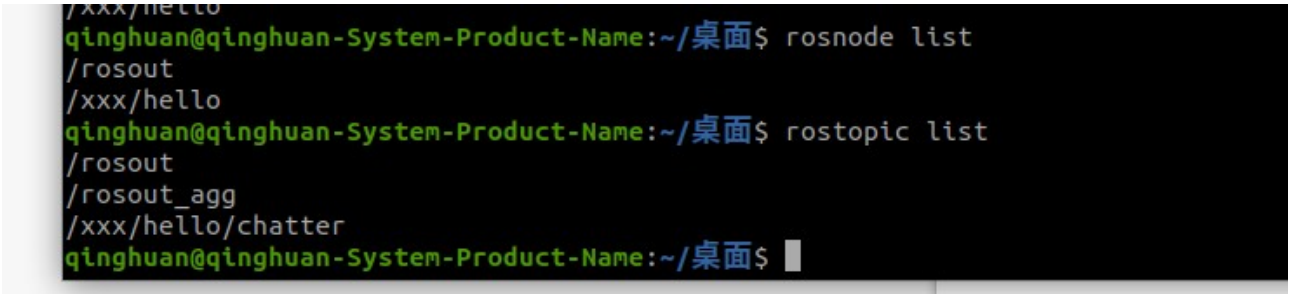
```
qinghuan@qinghuan-System-Product-Name: ~/桌面 80x11
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosnode list
/rosout
/xxx/hello
qinghuan@qinghuan-System-Product-Name:~/桌面$ rostopic list
/rosout
/rosout_agg
/xxx/yyy/chatter
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosnode list
/rosout
/xxx/hello
qinghuan@qinghuan-System-Product-Name:~/桌面$
```

生成私有的话题
～ 后续的生存是

```
setlocale(LC_ALL, "");
ros::init(argc, argv,"hello");
ros::NodeHandle nh1("~");
ros::Publisher pub = nh1.advertise<std_msgs::String>("chatter",1000);
```



看到 topic 就算在 hello 这个节点下面



相对命名，在节点的名字 hello 下面加入了额外的 yyy 命名空间
hello 的命名空间是添加了一个 hello 的 rosrun rename02_topic topic_name __ns:=xxx

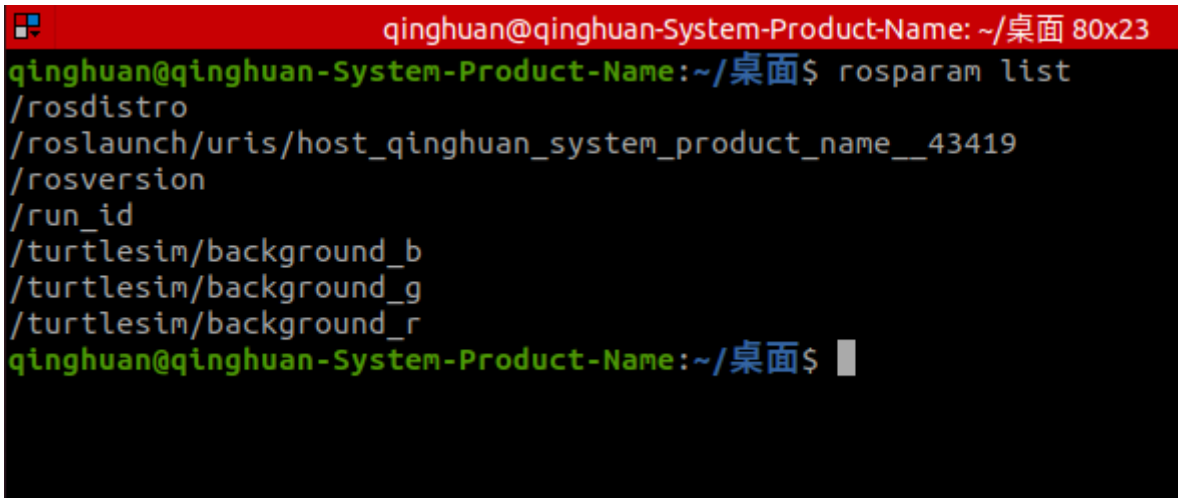
```
setlocale(LC_ALL, "");
ros::init(argc, argv,"hello");
ros::NodeHandle nh1("~");
ros::Publisher pub = nh1.advertise<std_msgs::String>("yyy/chatter",1000);
```

私有的 nodehandle 创建 一个/(斜杠)开头的全局话题，生成的结果 是 1 全局的话题 非私有的话题

ros 参数名的重命名

参数重命名会产生覆盖

通过设置全局参数/相对参数/私有参数
rosrun 包名 执行文件的名字 之后__radius :=3.545 就算设置参数 param 里面有个 key 为 radisu 的



.
rosrun turtlesim turtlesim_node __radius:=3.565

.

通过 launch 设置私有参数
在 node 里面就是 node 私有的，在<launch>标签下就三全局的

```
<launch deprecated="舍用说明 这个是注释说明这个文件除了什么问题">
<!--启动节点 -->
<param name="param_a" type="int" value="100" />
<rosparam command="load" file="$(find launch02_basic)/launch/param.yaml"  />
<rosparam command="dump" file="$(find launch02_basic)/launch/param1.yaml"  />
<rosparam command="delete" param="/bgb"  />
<node pkg="turtlesim" type="turtlesim_node" name="my_turtle"  output="screen" >
    <remap from="/turtle1/cmd_vel" to="/cmd_vel"/>
    <param name="param_b" type="double" value="3.14" />
    <rosparam command="load" file="$(find launch02_basic)/launch/param.yaml"  />

</node>


<!--pgk是包名   type是具体的文件的别名  节点类型 name 我设置的节点名称 output日志输出到那里  -->
<node pkg="turtlesim" type="turtle_teleop_key" name="my_key" output="screen" />

</launch >
```

```
/my_turtle/background_r
/my_turtle/bgb
/my_turtle/bgg
/my_turtle/bgr
/my_turtle/param_b
/param_a
/rosdistro
```

```
ros::init(argc,argv,"setparam");
ros::param::set("/paramA",52);
ros::param::set("paramB",1);
ros::param::set("~paramC",1);
ros::NodeHandle nh;
nh.setParam("setparamA",1);
nh.setParam("setparamB",1);
ros::NodeHandle nh1("~");
nh1.setParam("setparamC",1);
return 0;
```

我们给文件有个 namespace 是为了区分 A 和 B
paramA / 斜杠开头 对于他就算全局
paramB 就算根节点平级 这个节点我们加入了 namespace xxx 所以他就算在这里建立的
paramC 是基于节点建立的相对参数 是节点的子参数

```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosparam list
/paramA
/rosdistro
/roslaunch/uris/host_qinghuan_system_product_name__42141
/rosversion
/run_id
/xxx/paramB
/xxx/setparam/paramC
/xxx/setparam/setparamC
/xxx/setparamA
/xxx/setparamB
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
/src  Debug  0
```

nh 的全局和相对创建是不便的
但是创建私有 需要在这个节点下 建立一个私有的 Nodehandle nh("~")

```
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrun rename03_param param_set __ns:=xxx
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosparam list
/A
```