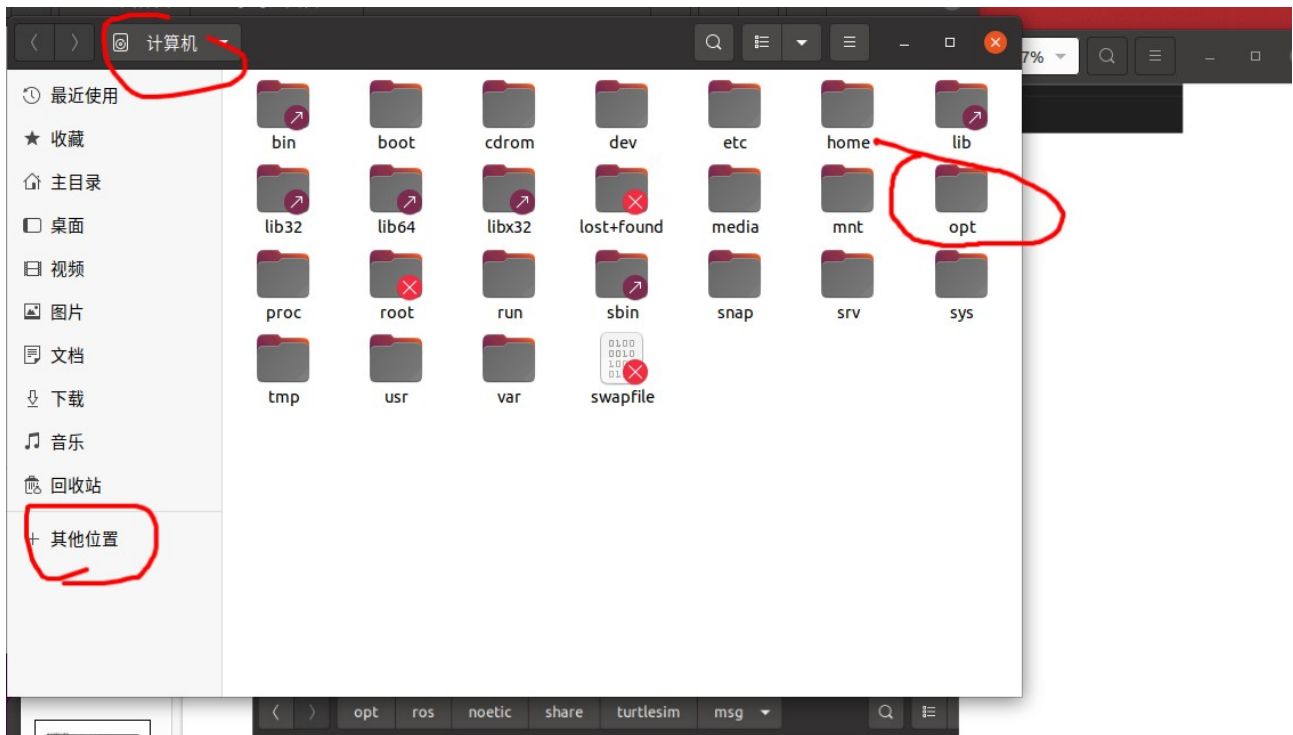


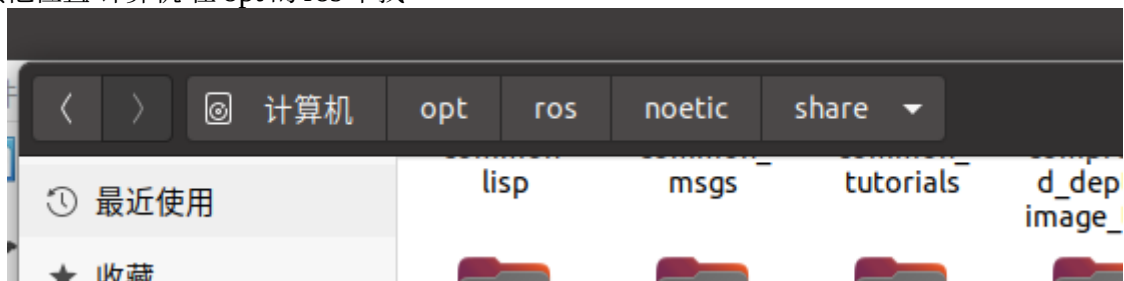
learn16 补充 嵌套 msg 的实现

静态坐标变化

查看消息类型



要先其他位置 计算机 在 opt 的 ros 下找



消息类型第一个是 header

第二个是 string

第三个是自定义的 transform

```
7
8 Header header
9 string child_frame_id # the frame id of the child frame
10 Transform transform
```

header 数据类型

## 5. std\_msgs/Header:

- 这个类型提供了一个消息头部，通常包含时间戳（`stamp`）和帧ID（`frame_id`）。
- 时间戳表示消息被创建的时间，而帧ID标识了消息的参考坐标系。
- `Header` 通常用作其他消息的头部信息，为消息提供上下文信息。

这里的成员变量包括：

- `seq`：一个无符号整数，表示消息的序列号。这个序列号是单调递增的，用于标识消息的顺序。
- `stamp`：一个 `ros::Time` 类型，表示消息的时间戳。这个时间戳记录了消息被创建或发布的时间。
- `frame_id`：一个字符串，用于指定消息的坐标系。在 ROS 中，坐标系通常用于指定传感器数据、位置信息等的参考框架。

```
1 #include <std_msgs/Header.h>
2 std_msgs::Header header;
3 header.stamp = ros::Time::now(); // 设置当前时间戳
4 header.frame_id = "base_link"; // 设置参考坐标系ID
5
6 //相应的源码
7 namespace std_msgs {
8     struct Header {
9         uint32 seq;           // 消息序列号
10        ros::Time stamp;      // 消息的时间戳
11        std::string frame_id; // 消息的坐标系ID
12    };
13 }
```

用处std\_msgs/Header 通常作为其他消息类型的一个字段，例如 `sensor_msgs/Image` / `PoseStamped` / `geometry_msgs`等。这样，每个消息都可以携带自己的时间戳和坐标系信息，这对于数据的同步和处理非常重要。

string 是一个字符串类型

transform

还是一个嵌套



下面是这两个数据类型

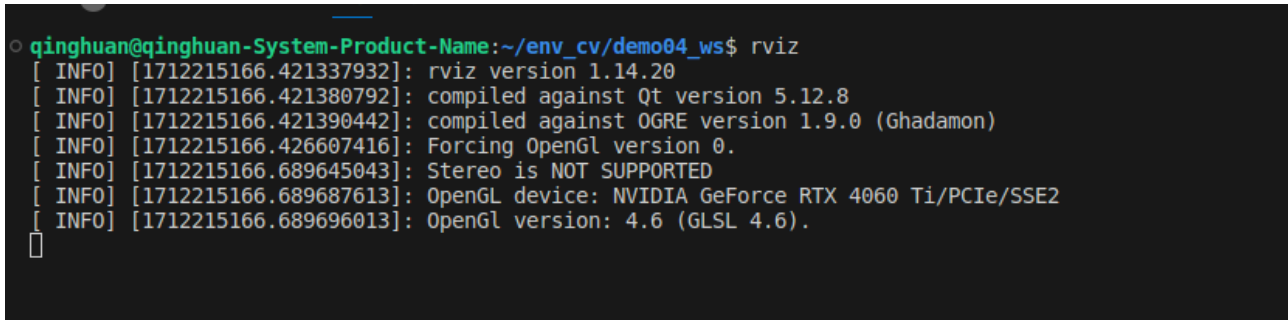
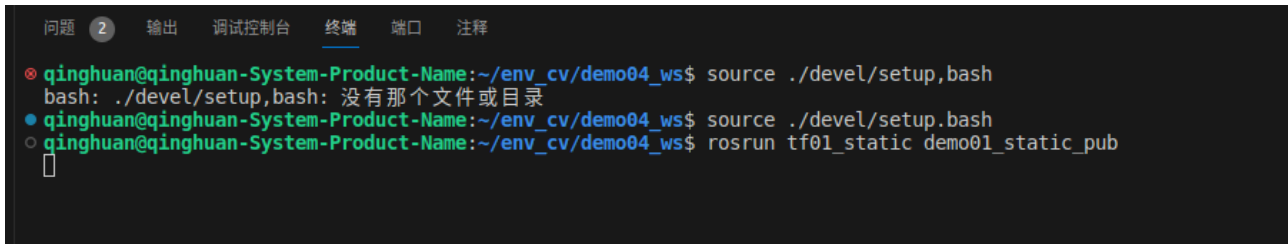
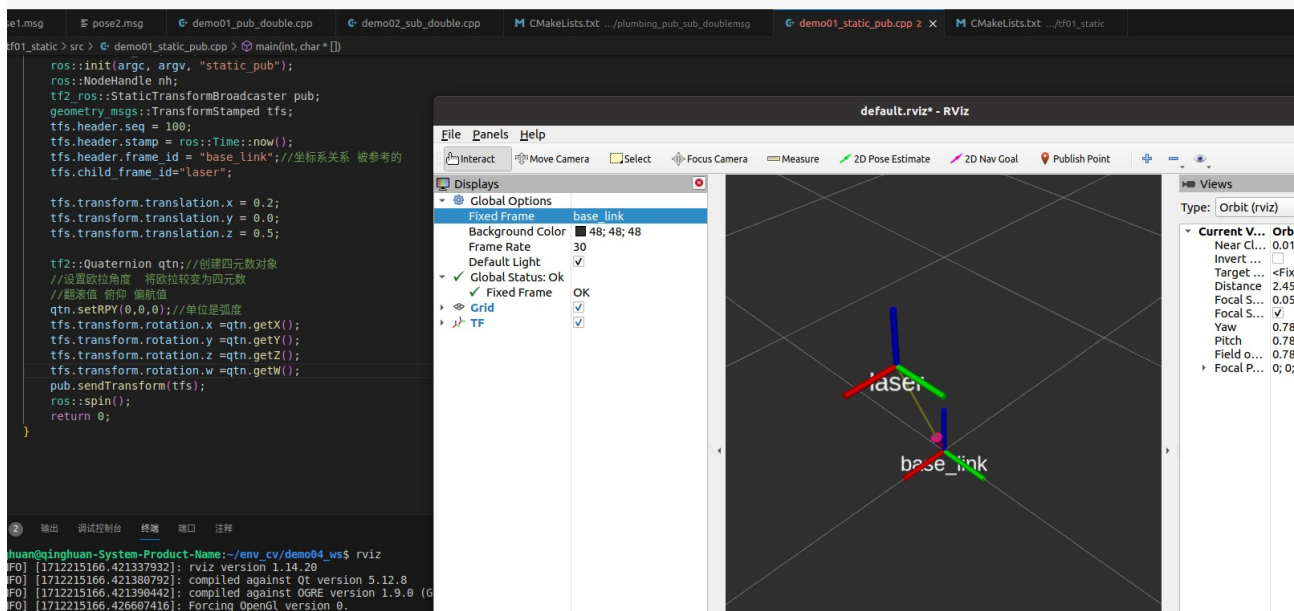
rotation 是四元数

translation 是 vector3



看这里

这个是在命令行用了 rviz



在转化 已经开始转化，但是发布的新坐标系 还没有发布

```
问题 2 输出 调试控制台 终端 端口 注释
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrn tf01_static demo01_static_sub
[rosrn] Couldn't find executable named demo01_static_sub below /home/qinghuan/env_cv/demo04_ws/src/tf01_static
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrn tf01_static demo02_static_sub
terminate called after throwing an instance of 'tf2::LookupException'
  what():  "base link" passed to lookupTransform argument target_frame does not exist.
已放弃 (核心已转储)
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ □
```

```
tf2_ros::Buffer buffer; //把数据存到这里
//订阅对象， 对象存在到 buff 里
tf2_ros::TransformListener listener(buffer);
```

在这个需要调用 tf2\_ros

订阅者 将订阅的消息缓存在 buffer 里 buffer.transform (要转换的数据， 转换到哪个坐标系)

```
tf2_ros::Buffer buffer; //把数据存到这里
//订阅对象， 对象存在到buff里
tf2_ros::TransformListener listener(buffer);
geometry_msgs::PointStamped ps;
ps.header.frame_id = "laser";
ps.header.stamp = ros::Time::now();

ps.point.x=2.0;
ps.point.y=3.0;
ps.point.z=5.0;
//转换坐标系的算法 tf的内置实现 通过循环转化数据
ros::Rate rate(10);
ros::Duration(2).sleep();
```

我们定义了一个 msg 的数据格式， 有 header 通过 point 嵌套的 msg point 是 float32 x/y/z 这三类

```
ros::Duration(2).sleep();
while(ros::ok())
{
    rate.sleep();
    //转换成 baselink的座标点 buffer有个函数transform
    geometry_msgs::PointStamped ps_out=buffer.transform(ps,"base_link");
    ROS_INFO("转换后的数据: (%.2f,%.2f,%.2f), 参考的坐标系是:", ps.point.x, ps.point.y, ps.point.z, ps.header.frame_id.c_str());

    ros::spinOnce();
}
return 0;
```



赵虚左在这里加入了异常处理  
之后这个是用 try 之后 catch 捕获异常  
这是结果

```
[ INFO] [1712282369.801029544]: 转换后的数据:(2.20,3.00,5.50 ),参考的坐标系是: base_link
• qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
• qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch tf01 static demo02 static_sub
[ INFO] [1712282391.802201736]: 异常信息 "base_link" passed to lookupTransform argument target_frame does not exist.
error: "base_link" passed to lookupTransform argument target_frame does not exist.
[ INFO] [1712282391.902089893]: 异常信息 "base_link" passed to lookupTransform argument target_frame does not exist.
error: "base_link" passed to lookupTransform argument target_frame does not exist.
[ INFO] [1712282392.002067636]: 转换后的数据:(2.20,3.00,5.50 ),参考的坐标系是: base_link
[ INFO] [1712282392.102060640]: 转换后的数据:(2.20,3.00,5.50 ),参考的坐标系是: base_link
[ INFO] [1712282392.202060265]: 转换后的数据:(2.20,3.00,5.50 ),参考的坐标系是: base_link
[ INFO] [1712282392.302065677]: 转换后的数据:(2.20,3.00,5.50 ),参考的坐标系是: base_link
[ INFO] [1712282392.402063100]: 转换后的数据:(2.20,3.00,5.50 ),参考的坐标系是: base_link
```

这个是代码

```
while(ros::ok())
{
    rate.sleep();
    //转换成 baselink的座标点 buffer有个函数transform
    //我们将ps这个从laser坐标 变换到 base_link 发布是 发布了两个关系
    try
    {
        geometry_msgs::PointStamped ps_out=buffer.transform(ps,"base_link");
        ROS_INFO("转换后的数据:(%.2f,%.2f,%.2f ),参考的坐标系是: %s",ps_out.point.x, ps_out.point.y, ps_out.point.z, ps_out.header.frame_id.c_str());
    }

    //我们
    // [ INFO] [1712280321.756704367]: 转换后的数据:(2.00,3.00,5.00 ),参考的坐标系是: laser
    catch(const std::exception &e)
    {
        ROS_INFO("异常信息 %s",e.what());
        std::cout<<"error: "<<e.what()<<std::endl;
    }
    ros::spinOnce();
}
```

这个是静态坐标系，因为他的代码可复用性高，因此通过这种方式也可以实现  
调用 ros 的静态实现  
正值是顺实针转动

## 补充1:

当坐标系之间的相对位置固定时，那么所需参数也是固定的: 父系坐标名称、子级坐标系名称、x偏移量、y偏移量、z偏移量、x 翻滚角度、y俯仰角度、z偏航角度，实现逻辑相同，参数不同，那么 ROS 系统就已经封装好了专门的节点，使用方式如下:

```
roslaunch tf2_ros static_transform_publisher x偏移量 y偏移量 z偏移量 z偏航角度 y俯仰角度 x翻滚角度 父级坐标系 子级坐标系
```

示例: roslaunch tf2\_ros static\_transform\_publisher 0.2 0 0.5 0 0 0 /baselink /laser

也建议使用该种方式直接实现静态坐标系相对信息发布。

动态坐标系的变换  
坐标系的偏移量 坐标系的翻转角度

创建的功能包 依赖于 tf2 tf2\_ros tf2\_geometry\_msgs

乌龟的 msg pose 信息

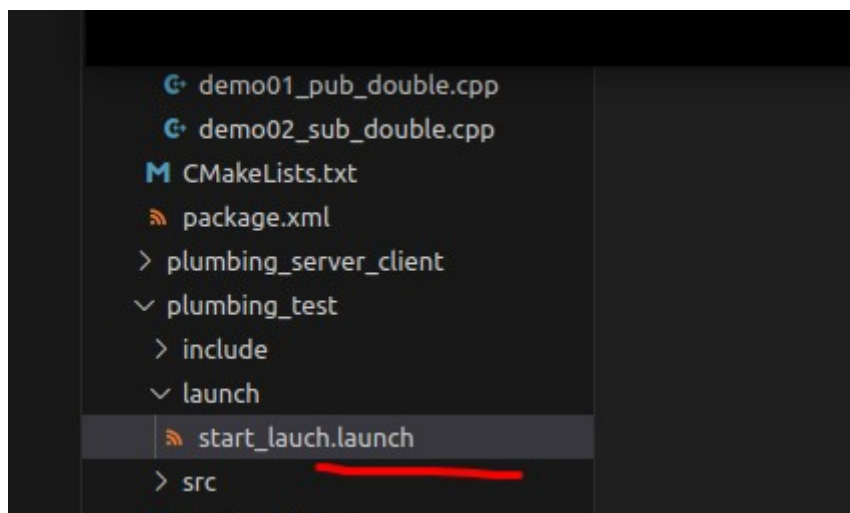
```
/opt/ros/noe
1 float32 x
2 float32 y
3 float32 theta
4
5 float32 linear_velocity
6 float32 angular_velocity
```

```
rosdemo@rosdemo-VirtualBox: ~ 49x11

rosdemo@rosdemo-VirtualBox:~$ rosmmsg info turtles
im/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

rosdemo@rosdemo-VirtualBox:~$
```

功能包下的 launch 文件就可以帮助我们打开 turtlesim 功能包的键盘节点和 显示节点



展示结果 动态发表数据

在这个功能包下 启动 turtlesim 的节点

```
问题 输出 调试控制台 终端 端口 注释
• qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
o qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch plumbing test start_launch.launch
... logging to /home/qinghuan/.ros/log/62b2b72a-f2e9-11ee-a73e-45bdd0184121/roslaunch-qinghuan-System-Product-Name-63829.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
```

运行动态发布数据



问题 输出 调试控制台 终端 端口 注释

```
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
○ qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ rosrn tf2_dynamic demo01_dynamic_pub
□
```

这是初始 u 信息

```
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0
---
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 1712296153
      nsecs: 102290423
    frame_id: "world"
  child_frame_id: "turtle1"
  transform:
    translation:
      x: 5.544444561004639
      y: 5.544444561004639
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
---
```

通过键盘调节 turtle 的位置 之后发布的数据也一起发生了变换

```
问题  输出  调试控制台  终端  端口  注释

b_double.cpp      x: 0.0
b_double.cpp      y: 0.0
t                  z: 0.0
                  w: 1.0
---
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 1712296190
      nsecs: 526565650
    frame_id: "world"
  child_frame_id: "turtle1"
  transform:
    translation:
      x: 10.344444274902344
      y: 5.5444444561004639
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
```

订阅方的实现

有个坐标点 是相对与乌龟坐标系，我们改成世界坐标系

```
51   ros::spinOnce();
52   }
53   return 0;
```

```
问题  输出  调试控制台  终端  端口  注释

[ INFO] [1712297286.900485283]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.000499972]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.100497594]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.200482704]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.300496503]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.400474693]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.500478195]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.600483924]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.700482634]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.800482024]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297287.900481535]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.000498104]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.100483264]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.200486436]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.300488565]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.400483215]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.500484965]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.600478586]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.700481526]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.800496075]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297288.900500245]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297289.000798780]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297289.100463047]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297289.200489117]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
[ INFO] [1712297289.300485126]: 转换后的数据:(7.54,8.54,5.00),参考的坐标系是: world
```

```
33
34 rate.sleep();
35 //转换成 base_link 的坐标点 buffer有个函数transform

问题 输出 调试控制台 终端 端口 注释

[ INFO] [1712297307.300489450]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297307.400503241]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297307.500499051]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297307.600494050]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297307.700488670]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297307.800483860]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297307.900480380]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.000489569]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.100497859]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.200487239]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.300482919]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.400483918]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.500492298]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.600486738]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.700490769]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.800496698]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297308.900477708]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.000475208]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.100478078]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.200478327]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.300495797]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.400493686]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.500493286]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.600469556]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
[ INFO] [1712297309.700487135]: 转换后的数据: (12.09,8.54,5.00 ),参考的坐标系是: world
```

这是变化的结果

1、通过 plumbing\_launch 包的 launch 文件启动 launch 节点

```
done
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
○ qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch plumbing_test start_launch.launch
```

2、发布方

```
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch tf2_dynamic demo01_dynamic_pub
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
○ qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch tf2_dynamic demo01_dynamic_pub
```

3、是订阅方，之后变化的结果也体现了

4、在订阅方的改变 从 demo02\_static\_sub.cpp 粘贴并作了一下的变更

1、改变了时间戳用的 `ros::Time(0.0)`

静态是就一个值 发送完就 ok 了

动态的数据实时的，buffer 缓存发布方的数据是存在时间的，每个坐标都有时间戳，进入缓存延时，时间戳在变化，坐标点的时间戳和坐标关系的时间戳 比对，看时间十分接近，接近就变化，差别大就无法转换，如果用 `ros::Time(0.0)` 设置没有时间值，就不考虑拿到坐标的时间值

2、改变了我们的点坐标系，是 turtle1 的 我们的坐标系名称发生变化了，从 laser 变化为 turtle1

3、我们的基准坐标也不算 base\_link 了 也要改称 world

```
geometry_msgs::PointStamped ps_out=buffer.transform(ps,"world");
```

这个的意义

这个点 ps 参考的坐标系是 turtle1

这个坐标系 我们的 turtle 的坐标是一直变得，全局坐标系不便，

ps 是 turtle 的点 是一个固定的点，这个随着 turtle 变换的点 在全局坐标系中如何变化

类比与 车辆的在运动，但摄像头的位置不变 摄像头相对与车辆坐标系是不变的，但是车辆在全局坐标系是变得，那么我这个摄像头在全局坐标系的位置是多少

//我们计算的就算这个东西

## 多坐标系变换

父坐标系 world 子坐标系 son1 son2 son3

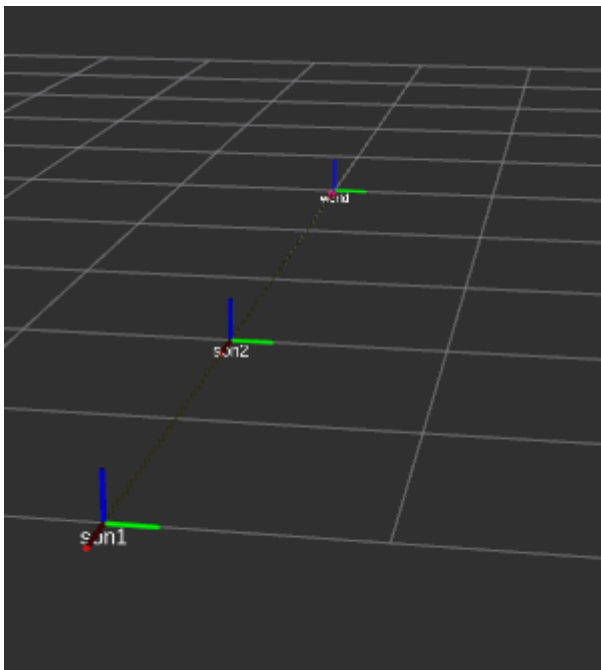
求 son1 下的点 在 son2 和 son3 的位置

借助于 tf2 实现 son1 和 son2 的转化

最后实现坐标点的转换

roscpp rospy std\_msgs tf2 tf2\_ros tf2\_geometry\_msgs geometry\_msgs

1、通过 launch 文件 发布了两个坐标系 rviz 的可视化



1、通过 launch 文件调用 ros 自带的静态包

**static\_transform\_publisher**

发布 x,y,z 以及 x y z 轴旋转的角度 父坐标系是 world 子坐标系是 son1 和 son2

```
tf3.launch - demo04_ws - Visual Studio Code

终端 帮助
demo01_dynamic_pub.cpp demo02_dynamic_sub.cpp tf3_c.launch x demo01_tf3.cpp CMakeLists.txt
src > tf3_tf3 > launch > tf3_c.launch
1 <launch>
2 <!-- 发布 son1相对于world son2相对于world的坐标关系 -->
3 <node pkg="tf2_ros" type="static_transform_publisher" name="son1" args="5 0 0 0 0 /world /son1" output="screen"/>
4 <node pkg="tf2_ros" type="static_transform_publisher" name="son2" args="3 0 0 0 0 /world /son2" output="screen"/>
5 </launch>
```

```
// target 的坐标系 源坐标系 以及 时间批评 time(0.0)就算最近的一个时间 一个取值 计算相对关系
geometry_msgs::TransformStamped son1_To_son2 =
buffer.lookupTransform("son2","son1",ros::Time(0.0));
这是计算坐标系之间的关系
```

计算在坐标系 son1 的点的位置 在 son2 的坐标系的位置

```
geometry_msgs::PointStamped point_s1_ats2 = buffer.transform(point_s1,"son2");
ROS_INFO("son1 的点在 son2 的值 %.2f %.2f %.2f",point_s1_ats2.point.x,
point_s1_ats2.point.y,
point_s1_ats2.point.z);
```

运行结果

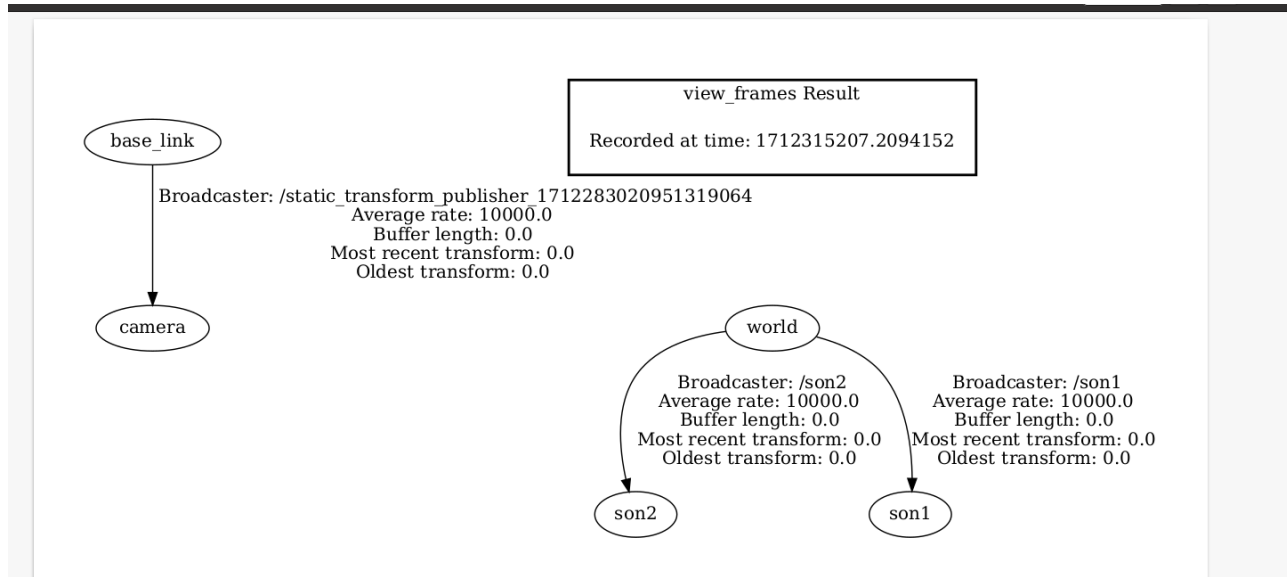
问题	输出	调试控制台	终端	端口	注释
	[ INFO ]	[1712315334.690917935]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315334.790882185]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315334.790913844]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315334.890880563]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315334.890912922]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315334.990879871]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315334.990912240]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.090881610]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.090916189]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.190894078]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.190926117]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.291066069]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.291098918]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.391061178]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.391092137]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.490911928]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.490943897]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.590892895]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.590924714]:	son1 的点在 son2 的值 3.00 2 3.00		
	[ INFO ]	[1712315335.690857612]:	son1 相对于 son2 的关系 父级 son2 子级 son1 2.00 0.00 0.00		
	[ INFO ]	[1712315335.690887051]:	son1 的点在 son2 的值 3.00 2 3.00		

```
问题 输出 调试控制台 终端 端口 注释
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ source ./devel/setup.bash
○ qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch tf3_tfs_tfs_c.launch
... logging to /home/qinghuan/.ros/log/62b2b72a-f2e9-11ee-a73e-45bdd0184121/roslaunch-qinghuan-System-Product-Name-125797.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://qinghuan-System-Product-Name:36599/
```

生成 pdf 文件

evince 查看 pdf

```
问题 输出 调试控制台 终端 端口 注释
● qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ roslaunch tf2_tools view_frames.py
[INFO] [1712315202.198378]: Listening to tf data during 5 seconds...
[INFO] [1712315207.204658]: Generating graph in frames.pdf file...
○ qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$ evince frames.pdf
□
```





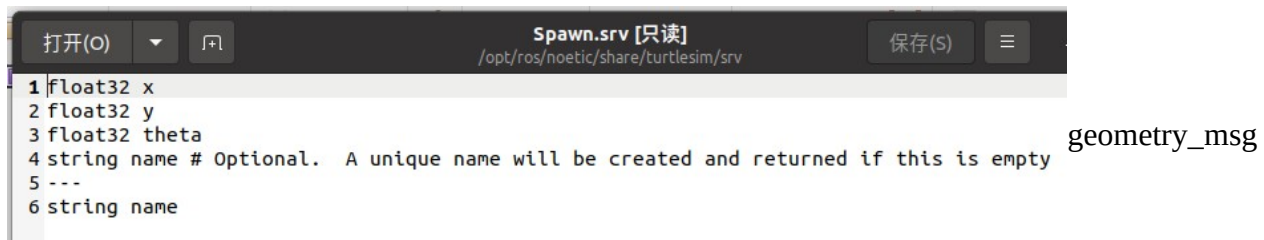
获取的话题

我们需要生成连两个乌龟 是服务消息类型

编写乌龟的发布位置信息的节点

编写订阅节点 生成新的

数据类型 msg 和 srv 的类型

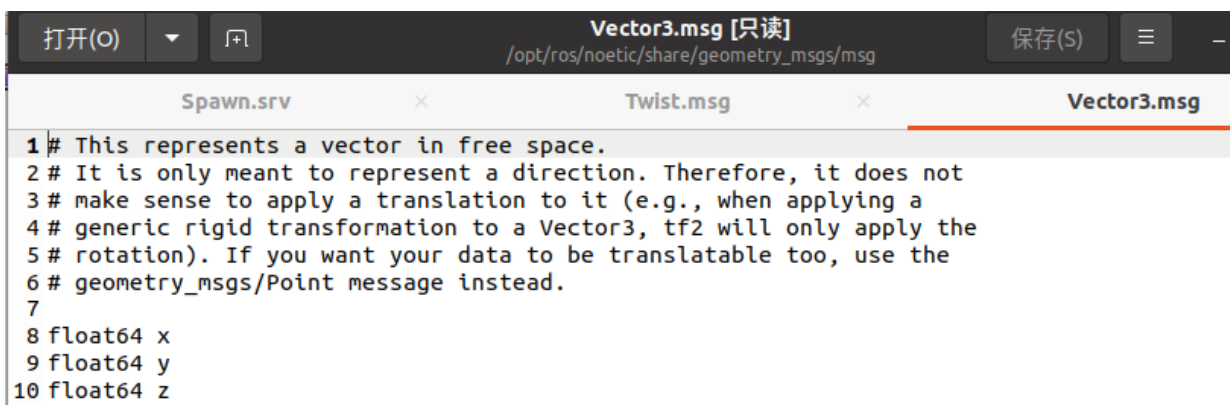


```
1 float32 x
2 float32 y
3 float32 theta
4 string name # Optional. A unique name will be created and returned if this is empty
5 ---
6 string name
```

geometry\_msg



```
1 # This expresses velocity in free space broken into its linear and angular parts.
2 Vector3 linear
3 Vector3 angular
```



```
1 # This represents a vector in free space.
2 # It is only meant to represent a direction. Therefore, it does not
3 # make sense to apply a translation to it (e.g., when applying a
4 # generic rigid transformation to a Vector3, tf2 will only apply the
5 # rotation). If you want your data to be translatable too, use the
6 # geometry_msgs/Point message instead.
7
8 float64 x
9 float64 y
10 float64 z
```

通过 launch 文件 向里面传入参数, 解析参数 是要在 init 后, 就算初始化 函数句柄之后, 解析参数

第一个是功能包 第二个是 参数

基于这个参数创建了一个全局变量, 用这个变量, 去改变相应的值

首先看到的是解析参数

```
ros::init(argc, argv, "dynamic_pub");

ros::NodeHandle nh;
//topic 是turtlepose
/*
修改订阅的话题名称 turtle1 动态传入的 frame_id也是动态传入的
*/
//放在初始化函数句柄的后面就是2个
if (argc!=2)
{
    //功能包名 和一个参数
    ROS_INFO("请传入一个参数,%d",argc);
    return 1;
}
else
{
    turtle_name = argv[1];
}
```

声明的变量 1 去 get 参数

在 serverclient 中向 server 发布一个新的乌龟，沿用的是 dynamic 动态发布的方法

在 sericeClient 中 `ros::ServiceClient client=nh.serviceClient<turtlesim::Spawn>("/spawn");`  
向/spawn 发布一个话题 我们这里有个乌龟的名字

我们这个消息类型有个乌龟的名字

srv.request.name 这个 name 是我们也会向 turtlesim 发布话的信息，基于 name 作后续的判断生成的相关话题就是 turtle22

```
int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "");
    ros::init(argc,argv,"service_client_call");
    ros::NodeHandle nh;
    ros::ServiceClient client=nh.serviceClient<turtlesim::Spawn>("/spawn");
    turtlesim::Spawn srv;
    srv.request.x = 1.0;
    srv.request.y = 4.0;
    srv.request.theta = 1.57;
    srv.request.name = "turtle22";
}
```

```
qinghuan@qinghuan-System-Product-Name: ~/桌面 80x11
/turtle2/pose
qinghuan@qinghuan-System-Product-Name: ~/桌面$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
/turtle22/cmd_vel
/turtle22/color_sensor
/turtle22/pose
qinghuan@qinghuan-System-Product-Name: ~/桌面$
```

因此我们需要把这个名字 保证和后续 test.launch 中 args 的参数 是要保证一样的  
红线的位置的 args="turtle2" 要和发布的乌龟名字一样

```
> tf04_test > launch > test.launch
1 <launch>
2 ....<!--启动turtlesim 生成一个新的乌龟-->
3 ....<!--name 相当于namespcae-->
4 ....<node pkg="turtlesim" type="turtlesim_node" name="turtle1" output="screen"..../>
5 ....<node pkg="tf04_test" type="test01_new_turtle" name="turtle2" output="screen"..../>
6 ....<node pkg="turtlesim" type="turtle_teleop_key" name="key" output="screen"..../>
7 ....<!--需要启动两个乌龟 相对与 世界的坐标关系的发布...
8 ....1、节点只编写一个
9 ....2、代码都一样 节点的名字不一样
0 ....3、节点需要启动两次
1 ....4、节点启动时 我们动态传参 第一次传turtle1 第二次传入 turtle2
2
3 .....>
4 ....<node pkg="tf04_test" type="test02_pub_turtle" name="pub1" args="turtle1" output="screen"..../>
5 ....<node pkg="tf04_test" type="test02_pub_turtle" name="pub2" args="turtle2" output="screen"..../>
6 </launch>
```

test01.launch 是将 发布的第二个乌龟的名字改成了 turtle22

之后我们运行 `rostopic list` `cmd_vel` 就都是 turtle22/pose 这些 因此注意名字的前后一致性

```
int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "");
    ros::init(argc,argv,"service_client_call");
    ros::NodeHandle nh;
    ros::ServiceClient client=nh.serviceClient<turtlesim::Spawn>("/spawn");
    turtlesim::Spawn srv;
    srv.request.x = 1.0;
    srv.request.y = 4.0;
    srv.request.theta = 1.57;
    srv.request.name = "turtle22";
```

```
qinghuan@qinghuan-System-Product-Name: ~/桌面 80x11
/turtle2/pose
qinghuan@qinghuan-System-Product-Name: ~/桌面$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
/turtle22/cmd_vel
/turtle22/color_sensor
/turtle22/pose
qinghuan@qinghuan-System-Product-Name: ~/桌面$
```

正确的写法是在 发布一个新的乌龟 我们给名字设置成 turtle2

```
demo01_dynamic_pub.cpp demo02_dynamic_sub.cpp tfs_c.launch demo01_tfs.cpp test.launch test01_new_turtle.cpp x
src > tf04_test > src > test01_new_turtle.cpp > main(int, char* [])
4 int main(int argc, char* argv[])
5 {
6     setlocale(LC_ALL, "");
7     ros::init(argc,argv,"service_client_call");
8     ros::NodeHandle nh;
9     ros::ServiceClient client=nh.serviceClient<turtlesim::Spawn>("/spawn");
10    turtlesim::Spawn srv;
11    srv.request.x = 1.0;
12    srv.request.y = 4.0;
13    srv.request.theta = 1.57;
14    srv.request.name = "turtle2";
15
16
17    client.waitForExistence();
18    bool flag1 = client.call(srv);
19    //发送数据 返回的一个bool 代表是否成功响应
20
21    if (flag1)
```

在这个状态下，我们的创建话题就算这些，我们需要给这写

```
qinghuan@qinghuan-System-Product-Name: ~/桌面 80x11
qinghuan@qinghuan-System-Product-Name:~/桌面$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
/turtle2/cmd_vel
/turtle2/color_sensor
/turtle2/pose
qinghuan@qinghuan-System-Product-Name:~/桌面$
```

我们是在第二个文件

test02\_pub\_turtle.cpp 文件中 我们从上面图 /turtle2/pose 中订阅乌龟的数据，之后把这个数据  
转换成一个坐标系 乌龟的动态坐标系 我们要对两个乌龟都生成一个坐标系，因此用的 launch 文件实现

一个 turtle1 的乌龟动态坐标系

一个是 turtle2 的乌龟坐标系哦

我们是通过 launch 文件的 args 向 tf04\_test 功能包中的 test02\_pub\_turtle.cpp 文件传入参数  
传入的参数是 turtle1 和 turtle2 这两个乌龟，

```
src > tf04_test > launch > test.launch
1  <launch>
2  <!--启动turtlesim 生成一个新的乌龟-->
3  <!--name 相当于namespace-->
4  <node pkg="turtlesim" type="turtlesim_node" name="turtle1" output="screen" />
5  <node pkg="tf04_test" type="test01_new_turtle" name="turtle2" output="screen" />
6  <node pkg="turtlesim" type="turtle_teleop_key" name="key" output="screen" />
7  <!--需要启动两个乌龟 相对与 世界的坐标关系的发布
8  1、节点只编写一个
9  2、代码都一样 节点的名字不一样
10 3、节点需要启动两次
11 4、节点启动时 我们动态传参 第一次传turtle1 第二次传入 turtle2
12
13  -->
14  <node pkg="tf04_test" type="test02_pub_turtle" name="pub1" args="turtle1" output="screen" />
15  <node pkg="tf04_test" type="test02_pub_turtle" name="pub2" args="turtle2" output="screen" />
16 </launch>
```

通过解析 argv 来活得传入的参数

记住这个解析参数要在 ros::init 后面， 否则是不一样的

roslaunch tf04\_test test02\_pub\_turtle turtle1

放在前面四个参数，我怀疑是这四个

下面的截图是打印出了我们的参数 显示了功能包的一些文件 还有 name 是 pub1 以及 log 的信息  
传入的参数信息



```

process[key-3]: started with pid [82835]
process[pub1-4]: started with pid [82836]
process[pub2-5]: started with pid [82837]
*****
/home/qinghuan/env_cv/demo04_ws/devel/lib/tf04_test/test02_pub_turtle
turtle1
__name:=pub1
__log:=/home/qinghuan/.ros/log/ffc94032-f3da-11ee-9138-8b04c9b05374/pub1-4.log
Reading from keyboard
-----
Use arrow keys to move the turtle. 'q' to quit.
[ INFO] [1712384940.935612164]: waitForService: Service [/spawn] could not connect to host
36079] waiting...

```

在 ros::init 后解析

argv[0] 是功能包的名字

argv[1]是传入的参数

相当于少了后面的两个参数

完整的响应结果

```

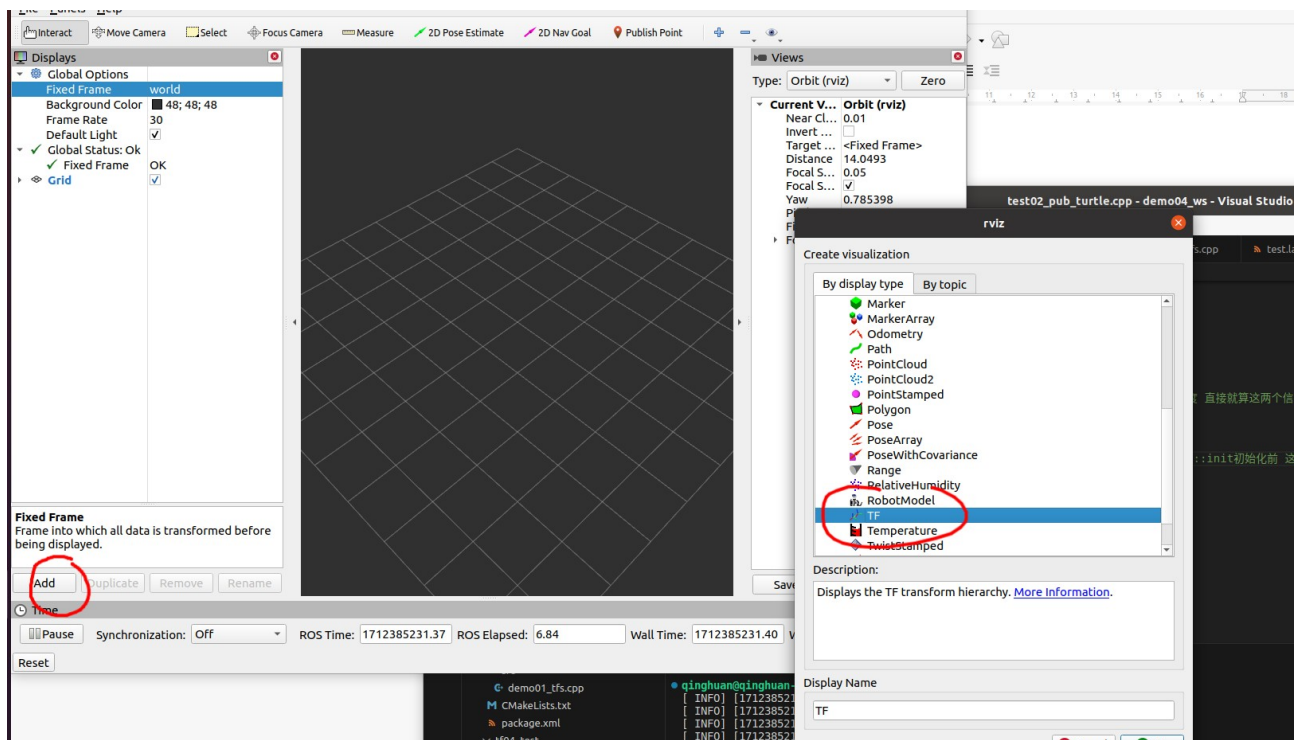
问题 输出 调试控制台 终端 端口 注释
process[pub2-5]: started with pid [82837]
*****
/home/qinghuan/env_cv/demo04_ws/devel/lib/tf04_test/test02_pub_turtle
turtle1
__name:=pub1
__log:=/home/qinghuan/.ros/log/ffc94032-f3da-11ee-9138-8b04c9b05374/pub1-4.log
Reading from keyboard
-----
Use arrow keys to move the turtle. 'q' to quit.
[ INFO] [1712384940.935612164]: waitForService: Service [/spawn] could not connect to host [qinghuan-System-Product-Name:
36079], waiting...
*****
/home/qinghuan/env_cv/demo04_ws/devel/lib/tf04_test/test02_pub_turtle
turtle2
__name:=pub2
__log:=/home/qinghuan/.ros/log/ffc94032-f3da-11ee-9138-8b04c9b05374/pub2-5.log
turtle1
turtle2
[ INFO] [1712384941.006990452]: Starting turtlesim with node name /turtle1
[ INFO] [1712384941.008679487]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
[ INFO] [1712384941.018150168]: waitForService: Service [/spawn] is now available.
[ INFO] [1712384941.021862398]: Spawning turtle [turtle2] at x=[1.000000], y=[4.000000], theta=[1.570000]
[ INFO] [1712384941.021955452]: 响应成功 turtle2
[turtle2-2] process has finished cleanly
log file: /home/qinghuan/.ros/log/ffc94032-f3da-11ee-9138-8b04c9b05374/turtle2-2*.log
[pub2-5] killing on exit

```

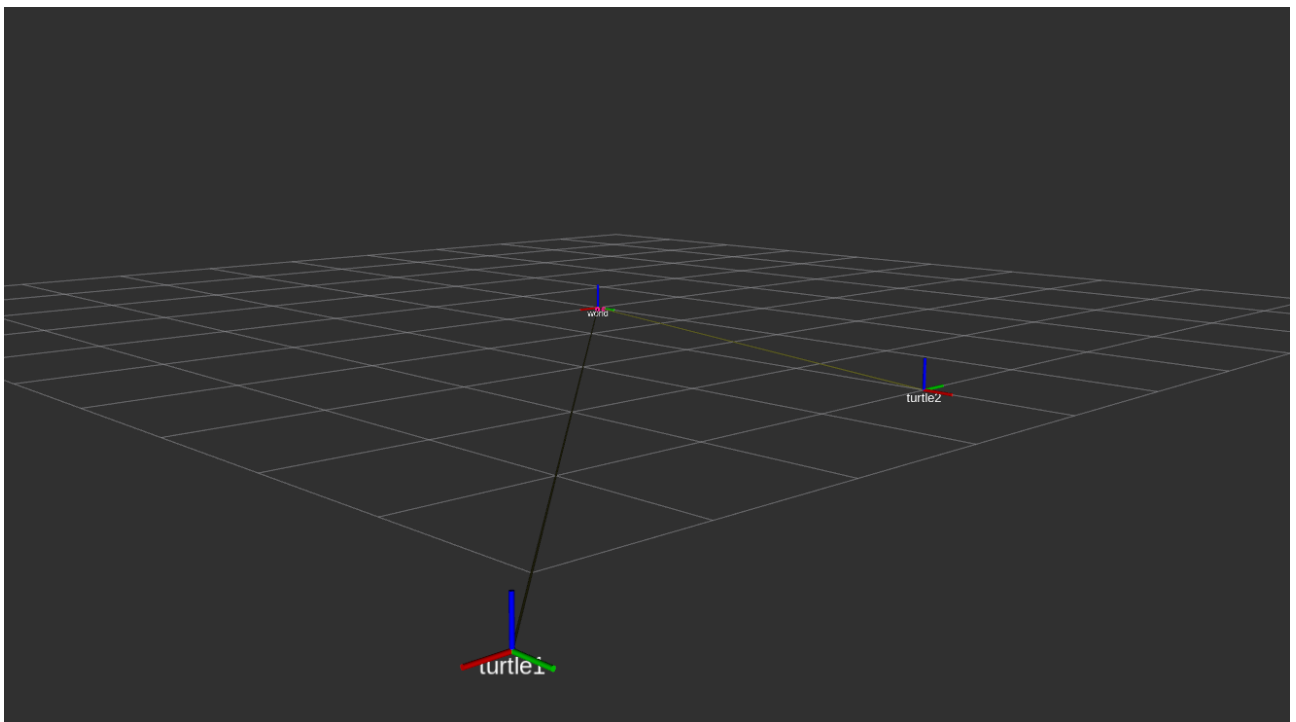
rviz 来显示 一个图像 这个更加直观

add 添加 tf 数据 我们记住要把 fixed frame 改称 world 这个我们的基础坐标系 我自定义的大地坐标系, 我们的两个 turtle 是基于这个坐标系作的





这个是坐标系的结果



订阅方实现，订阅两个乌龟的坐标系  
订阅 turtle1 和 turtle2 的两个坐标系，  
将 turtle1 转化成 turtle2 的坐标  
之后计算 线速度和角速度 把这个发布到 turtle2  
这样我们就可以控制 turtle1 turtle2 跟着一起走

两个乌龟的距离 就是通过订阅 turtle1 和 turtle2 两个坐标系  
之后把 turtle1 转换到 turtle2 上  
这个输出后的结果就算距离  
之后自定义一个系数跟踪 turtle1

1、serviceclient 通过 serviceClient 向 server 的话题/spawn 发布一个 turtle2  
通过 launch 文件 启动一个 turtle1 和 keyboard 控制节点 这个是 turtlesim 的包

2、我们需要订阅两个乌龟的位置信息，subscribe 订阅两个乌龟的位置信息  
我们将两个乌龟的位置信息 转化成坐标系 基于 world 的基础坐标系  
这个基础坐标系是大家自定义的。类似 ad4che 定义左上角为坐标原点

我们为了保证代码的复用性，我们通过 **launch** 文件 调用一个 cpp 文件活得乌龟的位置信息并生成坐标系 传入了 args 参数 代表是哪个乌龟  
这个传入参数 要和 1 中 serviceclient 中 发布的乌龟的名字，调用 turtlesim 包 生成一个新乌龟的  
话题的名字要保持一致的，这样我们才能订阅到乌龟的位置信息  
乌龟的位置信息的话题 名字 是 乌龟的名/pose /cmd\_vel

我们首先 subscribe /turtle1/pose /turtle2/pose 这个是话题 从话题 上订阅乌龟的位置信息，  
通过回调函数实现 生成坐标系， 这个生成的是动态坐标系， 并将这个动态坐标系发布出去

发布的时候，不仅有坐标系的位置信息，还有坐标系的名字和 父坐标系

1、给出坐标系的 x y z x 和 y 就算乌龟的位置信息 2 维平面没有 z  
2、小乌龟只有 z 轴的旋转 所以 通过欧拉角角度 和 tf2 自带的函数 生成四元数 的坐标系  
用 sendTransform 发布出去

3、我们是接收 乌龟的两个坐标系 这个话题就算 tf2 这个库的 直接建立个 listener  
之后把话题的数据保存到 buffer  
通过 lookupTransform 将 turtle1 的坐标系转换到 turtle2 上  
turtle2 相当于 原点 之后 turtle1 在 turtle2 的位置信息 就是两个乌龟的位置信息  
计算二者的速度和 角速度

自定义一个系数，  
通过 publish 发布出去 这个是要发布到 turtle2/cmd\_vel 上 这个上面是  
linear\_velocity 和 angular\_velocity 代表的 x y z 的位置信息 和 x y z 轴旋转的角度  
乌龟只有 x 的运动和 z 周的角度  
我们就把自定义的系数 乘以 两个乌龟的距离 作为下一个时刻乌龟的距离  
系数乘以 两个乌龟的系数 发布出去

就完成了控制