TF 坐标变换代码详解

AI 航团队

1. 常用数据类型

参考 ROS Wiki 上面给出的 TF 数据类型，主要有以下 6 种基本类型，分别对应四元数，向量，点坐标，位姿和转换模板。

Type	tf
Quaternion	tf::Quaternion
Vector	tf::Vector3
Point	tf::Point
Pose	tf::Pose
Transform	tf::Transform

还包括 tf::Stamped，数据类型如下图所示：这个数据类型是在上述所有基本类型（除了 tf::Transform）的基础上具有元素 frame_id_和 stamp_模板化。

```
template <typename T>
class Stamped : public T{
 public:
  ros::Time stamp_;
  std::string frame_id_;

  Stamped() :frame_id_ ("NO_ID_STAMPED_DEFAULT_CONSTRUCTION"){}; //Default constructor used only for preallocation

  Stamped(const T& input, const ros::Time& timestamp, const std::string & frame_id);

```

同时还包括 tf::StampedTransform，该类型是 tf::Transform 的一个特例，同时具有 frame_id，stamp 和 child_frame_id。如下图所示：

```
/** \brief The Stamped Transform datatype used by tf */
class StampedTransform : public tf::Transform
{
public:
  ros::Time stamp_; ///< The timestamp associated with this transform
  std::string frame_id_; ///< The frame_id of the coordinate frame  in which this transform is defined
  std::string child_frame_id_; ///< The frame_id of the coordinate frame this transform defines
  StampedTransform(const tf::Transform& input, const ros::Time& timestamp, const std::string & frame_id, const std::string & child_frame_id):
    tf::Transform (input), stamp_ ( timestamp ), frame_id_ (frame_id), child_frame_id_(child_frame_id){ };

  /** \brief Default constructor only to be used for preallocation */
  StampedTransform() { };

  /** \brief Set the inherited Traonsform data */
  void setData(const tf::Transform& input){*static_cast<tf::Transform*>(this) = input;};

```

2. TF 编程

任何使用 TF 包的时，都需要编写两个程序，分别用来监听 TF 变换和广播 TF 变换的功能，称之为 TF 监听器和 TF 广播器

1》TF 监听器：接受缓存系统中发布的所有参考系坐标变换，并从中查询所需要的参考系变换。

2》TF 广播器：广播 TF 变换，向系统中广播参考系之间的坐标变换关系。系统中可能会存在多个不同部分的 tf 变化广播，但每个广播可以直接将参考系变换关系直接插入 tf 树中，不需要再进行同步。

2.1 TF 广播器

实现功能：创建 TF 广播器，创建坐标变换值并发布实时发布坐标变换

编程思路：

1.初始化 ROS 节点，并订阅 turtle 的位置消息；

2.循环等待话题消息，接收到之后进入回调函数，该回调函数用以处理并发布坐标变换；

3.在该回调函数内部定义一个广播器；

4.根据接收到的小海龟的位置消息，创建坐标变换值

5.通过定义的广播器发布坐标变换

具体代码部分如下：

在具体分析之前我们先来看一下 “transform_broadcaster.h” 里面的内容

注意到在命名空间 tf 内部定义了一个 TransformBroadcaster 类，这个类的内部的内容也很简单：

声明了一个无参构造函数 TransformBroadcaster()；

使用了函数重载的方法定义了多个同名函数 sendTransform；

声明了一个私有化的成员变量 tf2_broadcaster_

基于此内容，我们再将回调函数代码的层次分为：

（1）定义 tf 广播器

```
static tf::TransformBroadcaster br;
```

tf::TransformBroadcaster 有一个无参构造函数，因此初始化时直

（2）创建坐标变换

根据 tf 内部的数据类型，我们首先声明一个“Transform”数据结构，用来记录变换内容

```
tf::Transform transform;
```

以下两个函数可以实现我们的需求。内部参数各自要求为“Vector3”和“Quaternion”

Vector3 类型直接声明即可使用，Quaternion 类型需要先使用 setRPY 这个函数进行赋值。

由此，我们可以得到

```
transform.setOrigin(tf::Vector3(msg->x, msg->y, 0.0));
transform.setRotation(tf::Quaternion::setRPY(0, 0, msg->theta));
```

进一步简化得到

```
tf::Transform transform;
transform.setOrigin(tf::Vector3(msg->x, msg->y, 0.0));
tf::Quaternion q;
q.setRPY(0,0,msg->theta);
transform.setRotation(q);
```

（3） 广播器发布坐标变换

br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", turtle_name));

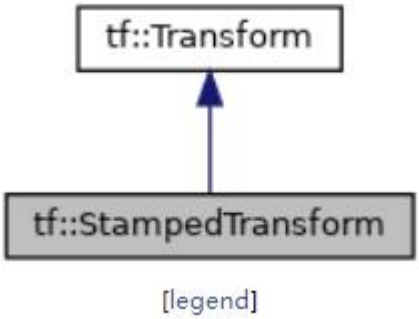
根据前面所解释的 “transform_broadcaster.h” 里面的内容应该不难理解

内部参数类型为“StampedTransform”，下图为继承关系图，也就是说 StampedTransform 这个类里面的内容继承自 Transform。

void tf::TransformBroadcaster::sendTransform (const **StampedTransform** & transform)

Send a **StampedTransform** The stamped data structure includes frame_id, and time, and parent_id already.

Definition at line **54** of file **transform_broadcaster.cpp**.



使用时需要在内部声明：输入（即所需坐标变换），时间戳，框架 id 和子框架 id。

```
tf::StampedTransform::StampedTransform ( const tf::Transform &  input,
                                         const ros::Time &       timestamp,
                                         const std::string &     frame_id,
                                         const std::string &     child_frame_id
                                        )
```

1.

2.2 TF 监听器

创建 TF 监听器，创建第二只海龟，监听坐标变换并发布运动控制指令使第二只海龟向第一只海龟运动

编程思路：

2. 初始化 ROS 节点，并向 MASTER 注册节点信息；
3. 通过服务调用产生第二只海龟；
4. 创建 turtle2 的速度控制发布器；
5. 创建 tf 监听器并监听 turtle2 相对于 turtle1 的坐标变换；
6. 根据坐标变换发布速度控制指令；

代码如下：

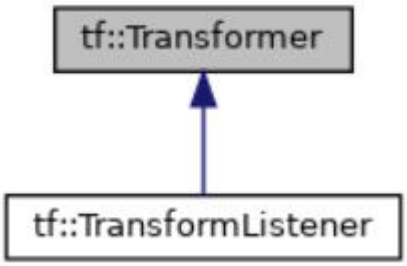
详见 ros 官网

http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20listener%20%28C%2B%2B%29

```cpp
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <turtlesim/Velocity.h>
#include <turtlesim/Spawn.h>

int main(int argc, char** argv){
  ros::init(argc, argv, "my_tf_listener");

  ros::NodeHandle node;

  ros::service::waitForService("spawn");
  ros::ServiceClient add_turtle =
    node.serviceClient<turtlesim::Spawn>("spawn");
  turtlesim::Spawn srv;
  add_turtle.call(srv);

  ros::Publisher turtle_vel =
    node.advertise<turtlesim::Velocity>("turtle2/command_velocity", 10);

  tf::TransformListener listener;

  ros::Rate rate(10.0);
  while (node.ok()){
    tf::StampedTransform transform;
    try{
      listener.lookupTransform("/turtle2", "/turtle1",
                               ros::Time(0), transform);
    }
    catch (tf::TransformException ex){
      ROS_ERROR("%s",ex.what());
      ros::Duration(1.0).sleep();
    }

    turtlesim::Velocity vel_msg;
    vel_msg.angular = 4.0 * atan2(transform.getOrigin().y(),
                                  transform.getOrigin().x());
    vel_msg.linear = 0.5 * sqrt(pow(transform.getOrigin().x(), 2) +
                                pow(transform.getOrigin().y(), 2));
    turtle_vel.publish(vel_msg);

    rate.sleep();
  }
  return 0;
};
```

（1）定义 tf 监听器

首先需要明确，TransformListener 的内容继承自 Transformer 类（注意这里不是 Transform 类哦），因此在使用时需要同时查看两者的使用文档，当然最简单的还是看 wiki 上面关于 listener 的使用。



内部构造和析构函数如下（截取自头文件）：

```
public:
  TransformListener(ros::Duration max_cache_time = ros::Duration(DEFAULT_CACHE_TIME), bool spin_thread = true);

  TransformListener(const ros::NodeHandle& nh,
                    ros::Duration max_cache_time = ros::Duration(DEFAULT_CACHE_TIME), bool spin_thread = true);

  ~TransformListener();
```

在大多数情况下，使用以下命令声明即可

```
tf::TransformListener listener
```

（2）监听坐标变换

声明一个空的变换

```
tf::StampedTransform transform;
```

2.3 实现效果

完成程序的编写之后我们在 CMakeLists 文件添加

```
add_executable(turtle_tf_listener
src/turtle_tf_listener.cpp)target_link_libraries(turtle_tf_listener ${catkin_LIBRARIES})
```

并配置 Launch 文件

```
 <launch>  ...  <node pkg="learning_tf" type="turtle_tf_listener"  name="listener"
/> </launch>
```

```
roslaunch learning_tf start_demo.launch
```

观察结果，发现一个小海龟可以正常的跟随另外一只海龟移动。

3. ros 中静态 tf 坐标关系发布方式

static_transform_publisher

static_transform_publisher 工具的功能是发布两个参考系之间的静态坐标变换，两个参考系一般不发生相对位置变化。

命令的格式如下：

static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period_in_ms

static_transform_publisher x y z qx qy qz qw frame_id child_frame_id period_in_ms

以上两种命令格式，需要设置坐标的偏移和旋转参数，偏移参数都使用相对于 xyz 三轴的坐标位移，而旋转参数第一种命令格式使用以弧度为单位的 yaw/pitch/roll 三个角度（yaw 是围绕 x 轴旋转的偏航角，pitch 是围绕 y 轴旋转的俯仰角，roll 是围绕 z 轴旋转的翻滚角），而第二种命令格式使用四元数表达旋转角度。发布频率以 ms 为单位，一般 100ms 比较合适。

该命令不仅可以在终端中使用，还可以在 launch 文件中使用，使用方式如下：

```
<launch>
<node pkg="tf" type="static_transform_publisher" name="link1_broadcaster" args="1 0 0 0 0 0 1 link1_parent link1 100" />
</launch>
```