10.Tf publishing and monitoring

10.1. tf Function Package

10.1.1. tf is a functional package that allows users to track multiple coordinate systems over time. It uses a tree shaped data structure to buffer and maintain coordinate transformation relationships between multiple coordinate systems based on time. It can help developers complete coordinate transformations such as points and vectors between coordinate systems at any time.

10.1.2 Usage Steps

1. Monitoring tf transformationReceive and cache all coordinate system transformation data published in the system, and query the required coordinate transformation relationships from it.
2. Broadcasting tf transformationBroadcast the coordinate transformation relationship between coordinate systems in the system. There may be multiple different parts of tf transformation broadcasting in the system. Each broadcast can directly insert coordinate transformation relationships into the tf tree without the need for synchronization.

10.2. Programming Implementation of Broadcasting and Monitoring in the tf Coordinate System

10.2.1 Creating and Compiling Function Packs

```
cd ~/catkin_ws/src
catkin_create_pkg learning_tf rospy roscpp turtlesim tf
cd ..
catkin_make
```

10.2.2 How to implement a tf broadcaster

1. Define the tf broadcaster (TransformBroadcast);
2. Initialize tf data and create coordinate transformation values
3. Publish coordinate transformation (sendTransform);

10.2.3 How to implement a tf listener

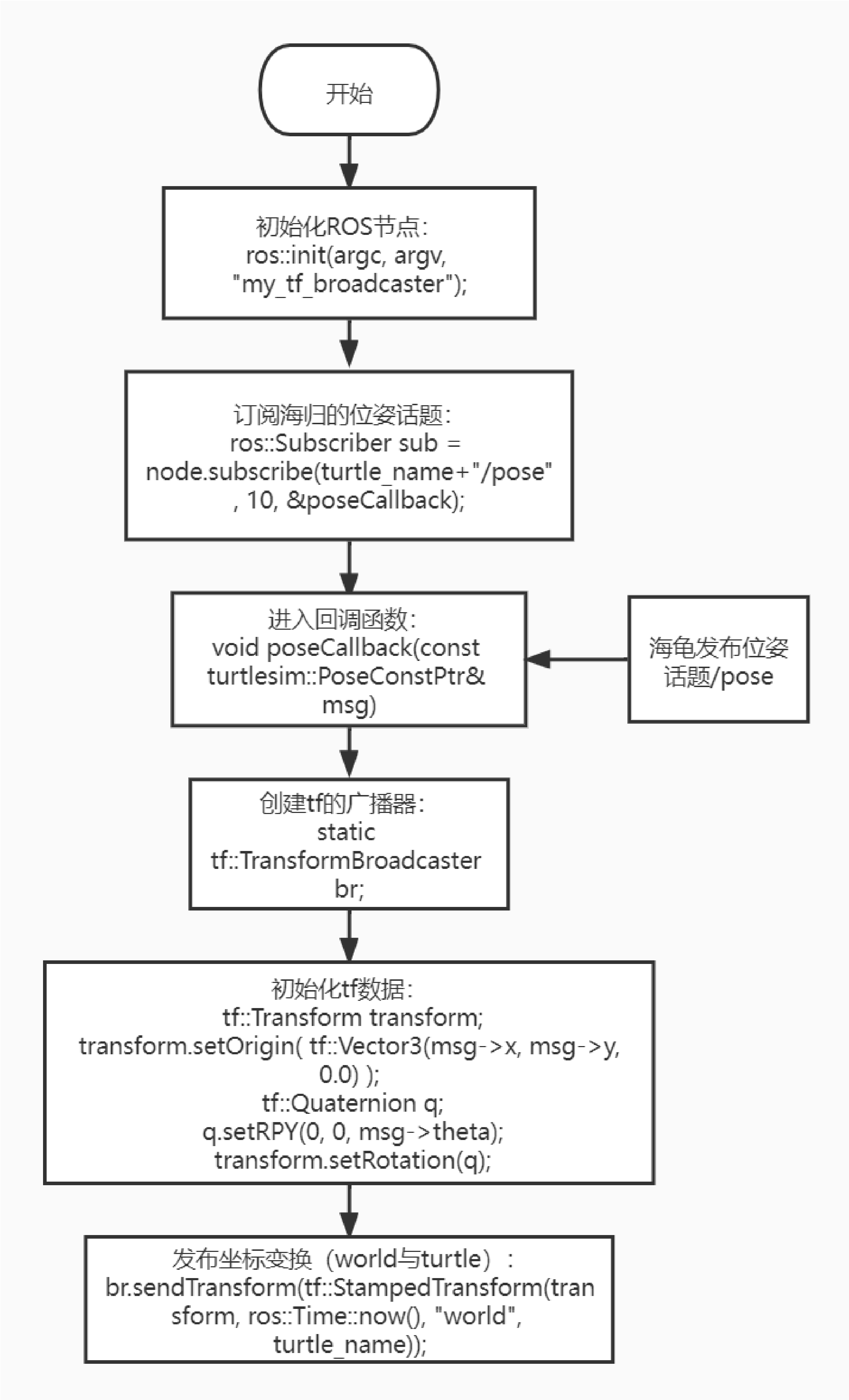
1. Define a TF listener (TransformListener);
2. Find coordinate transformations (waitForTransform, lookupTransform)

10.2.4. C++language implementation of tf broadcaster

1. In the feature pack learning_ Create a C++file (with the suffix. cpp) in the src folder of tf and name it turtle_ tf_ broadcaster.cpp
2. Copy and paste the program code below into the title_ tf_ In the broadcaster.cpp file

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <turtlesim/Pose.h>

std::string turtle_name;

void poseCallback(const turtlesim::PoseConstPtr& msg)
{

    static tf::TransformBroadcaster br;// 创建tf的广播器

    // 初始化tf数据
    tf::Transform transform;
    transform.setOrigin( tf::Vector3(msg->x, msg->y, 0.0) );//设置xyz坐标
    tf::Quaternion q;
    q.setRPY(0, 0, msg->theta);//设置欧拉角：以x轴，y轴，z轴旋转
    transform.setRotation(q);

    br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world",
turtle_name));// 广播world与turtle坐标系之间的tf数据
}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "turtle_world_tf_broadcaster");// 初始化ROS节点

    if (argc != 2)
    {
        ROS_ERROR("Missing a parameter as the name of the turtle!");
        return -1;
    }

    turtle_name = argv[1];// 输入参数作为海龟的名字

    // 订阅海龟的位姿话题/pose
    ros::NodeHandle node;
    ros::Subscriber sub = node.subscribe(turtle_name+"/pose", 10,
&poseCallback);

        // 循环等待回调函数
    ros::spin();

    return 0;
};
```

3. Process Flow Chart



4. Code parsing

Firstly, subscribe to the Little Turtle's/pose pose topic first. If any of the topics are published, then enter the callback function. In the callback function, first create a broadcaster for tf, and then initialize tf data. The value of the data is the data passed through the subscription/pose topic. Finally, publish the transformation of the Little Turtle's world coordinates through br.sendTransform, and let's talk about the sendTransform function here. There are four parameters, the first parameter represents the coordinate transformation of tf:: Transform type (i.e. the previously initialized tf data), the second parameter is the timestamp, and the third and fourth parameters are the source and target coordinate systems of the transformation.

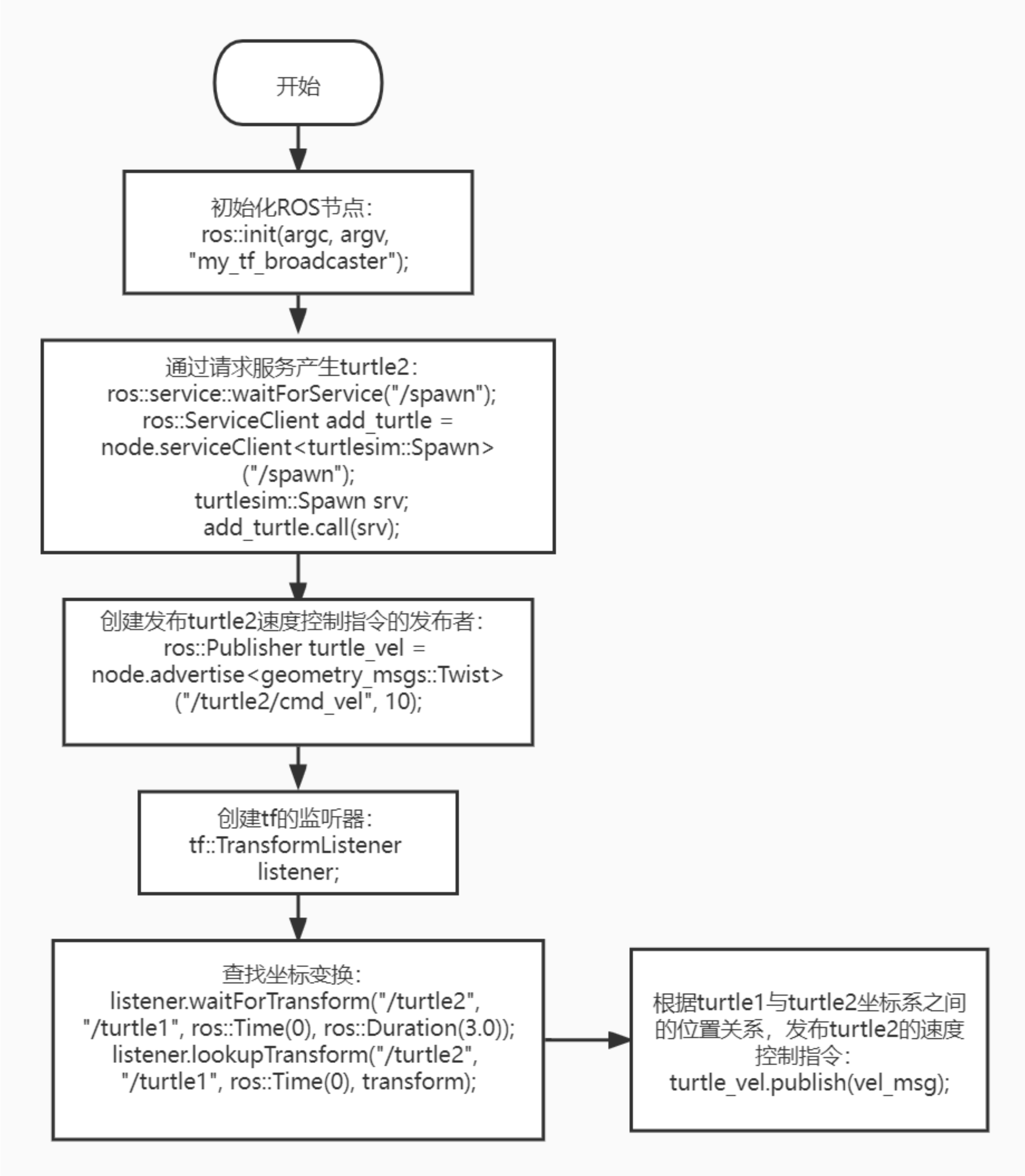
10.2.4. C++Language Implementation of tf Listener

1. In the feature pack learning_ Create a C++file (with the suffix. cpp) in the src folder of tf and name it turtle_ tf_ listener.cpp
2. Copy and paste the program code below into the title_ tf_ In the listener. cpp file

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/Twist.h>
#include <turtlesim/Spawn.h>

int main(int argc, char** argv)
{

    ros::init(argc, argv, "turtle1_turtle2_listener");// 初始化ROS节点


    ros::NodeHandle node; // 创建节点句柄

    // 请求服务产生turtle2
    ros::service::waitForService("/spawn");
    ros::ServiceClient add_turtle = node.serviceClient<turtlesim::Spawn>
("/spawn");
    turtlesim::Spawn srv;
    add_turtle.call(srv);

    // 创建发布turtle2速度控制指令的发布者
    ros::Publisher vel = node.advertise<geometry_msgs::Twist>("/turtle2/cmd_vel",
10);

    tf::TransformListener listener;// 创建tf的监听器

    ros::Rate rate(10.0);

    while (node.ok())
    {
        // 获取turtle1与turtle2坐标系之间的tf数据
        tf::StampedTransform transform;
        try
        {
            listener.waitForTransform("/turtle2", "/turtle1", ros::Time(0),
ros::Duration(3.0));
            listener.lookupTransform("/turtle2", "/turtle1", ros::Time(0),
transform);
        }
```

```
        catch (tf::TransformException &ex)
        {
            ROS_ERROR("%s",ex.what());
            ros::Duration(1.0).sleep();
            continue;
        }
![listener_cpp](C:\Users\Administrator\Desktop\nano资料更新\7. ROS advanced 
tutorial\10. TF release and monitoring\listener_cpp.jpg)
        // 根据turtle1与turtle2坐标系之间的位置关系，通过数学计算公式，得出角速度和线速度，
发布turtle2的速度控制指令
        geometry_msgs::Twist turtle2_vel_msg;

        turtle2_vel_msg.angular.z = 6.0 * atan2(transform.getOrigin().y(),
                                        transform.getOrigin().x());
        turtle2_vel_msg.linear.x = 0.8 * sqrt(pow(transform.getOrigin().x(), 2) 
+
                                      pow(transform.getOrigin().y(), 2));
        vel.publish(turtle2_vel_msg);

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

3. Process Flow Chart



4. Code parsing

Firstly, generate another little turtle named Turtle2 through service invocation, and then create a Turtle2 speed control publisher; Next, create a listener to listen and look for the left transformation of turnle1 and tuetle2, which involves two functions, waitForTransform and lookupTransformWaitForTransform (target_frame, source_frame, time, timeout): Two frames represent the target coordinate system and the source coordinate system respectively, while time represents the time to wait for the transformation between the two coordinate systems. Since coordinate transformation is a blocking program, a timeout needs to be set to represent the timeout time.

LookupTransform (target_frame, source_frame, time, transform): Given the source coordinate system (source_frame) and target coordinate system (target_frame), obtain the coordinate transformation (transform) between the two coordinate systems at the specified time (time).

After lookupTransform, we obtained the result of coordinate transformation, transform, and then obtained the values of x and y through transform. getOrigin(). y(), transform. getOrigin(). x(). Then, through mathematical operations, we obtained the angular velocity angular. z and linear velocity linear. x. Finally, we published it and let turnle2 move

10.2.5 Modifying CMakelist.txt and Compiling

1. Modify CMakelist.txtModify CMakelist.txt under the feature pack and add the following content

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

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

2. compile

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash
```

10.2.6 Demonstration of startup and operation effects

1. firing

```
roscore
rosrun turtlesim turtlesim_node
rosrun learning_tf turtle_tf_broadcaster __name:=turtle1_tf_broadcaster /turtle1
rosrun learning_tf turtle_tf_broadcaster __name:=turtle2_tf_broadcaster /turtle2
rosrun learning_tf turtle_tf_listener
rosrun turtlesim turtle_teleop_key
```

2. SHOW

3. Program Description

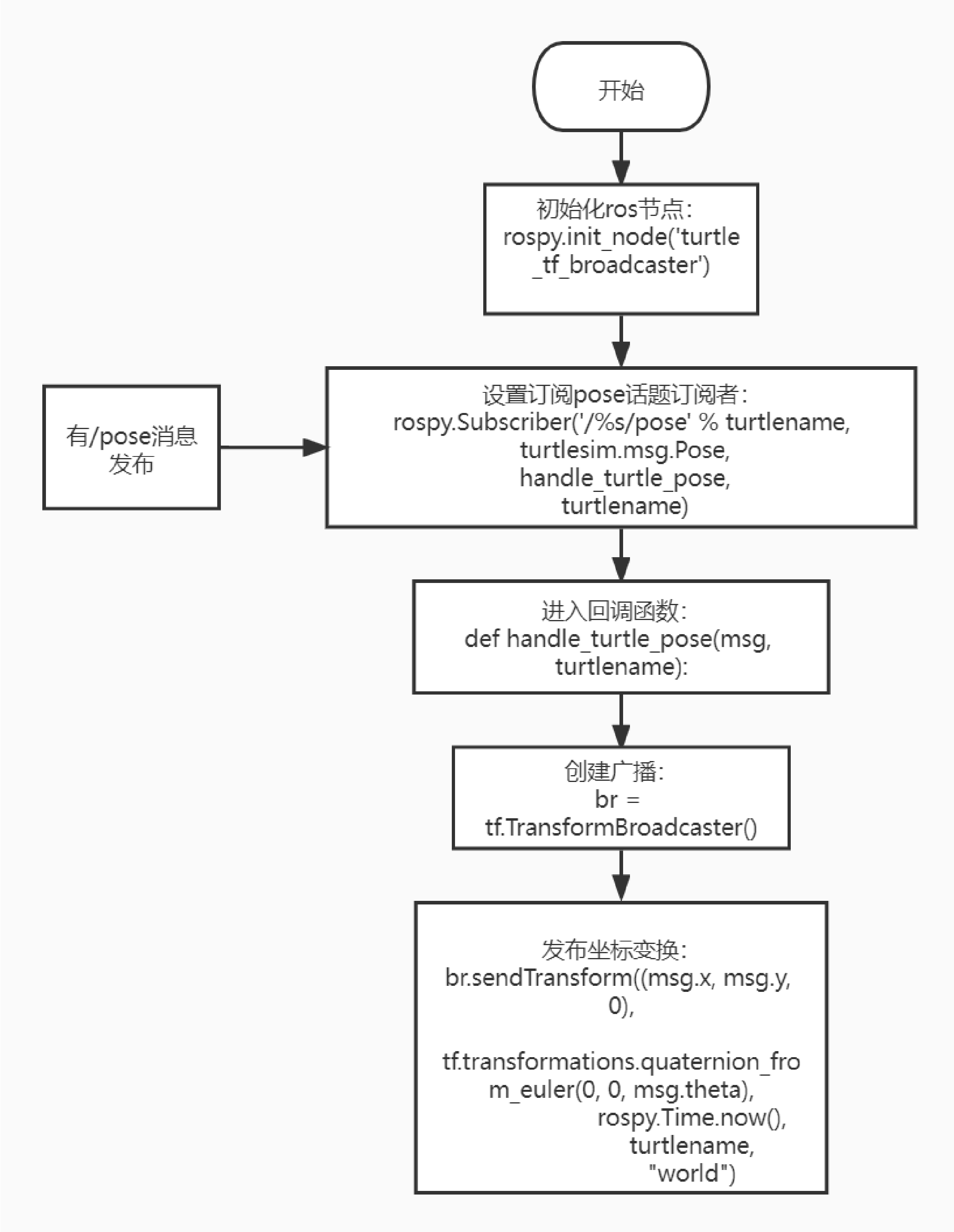
After starting the score, activate the Little Turtle node, and a little turtle will appear on the terminal; Then we will publish two tf transformations, turtle1->world and turtle2->world, because to know the changes between turtle2 and turtle1, we need to know the transformations between them and world; Then, start the tf listening program, and at this point, you will find that the terminal has generated another little turtle named Turtle2, and Turtle2 will move towards Turtle1; Then, we open the keyboard control and control the movement of turnle1 by pressing the directional keys, and then turnle2 will follow the movement of turnle1.

10.2.7. Implementing tf broadcaster in Python language

1. In the feature pack learning_ Tf, create a folder script, switch to that directory, create a new. py file, and name it turtle_ tf_ broadcaster.py
2. Copy and paste the program code below into the title_ tf_ In the broadcaster.py file

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import roslib
roslib.load_manifest('learning_tf')
import rospy

import tf
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()#定义一个tf广播

    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0, msg.theta),
                     rospy.Time.now(),
                     turtlename,
                     "world")

if __name__ == '__main__':

    rospy.init_node('turtle1_turtle2_tf_broadcaster')#初始化ros节点

    turtlename = rospy.get_param('~turtle')

    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

3. Program flowchart



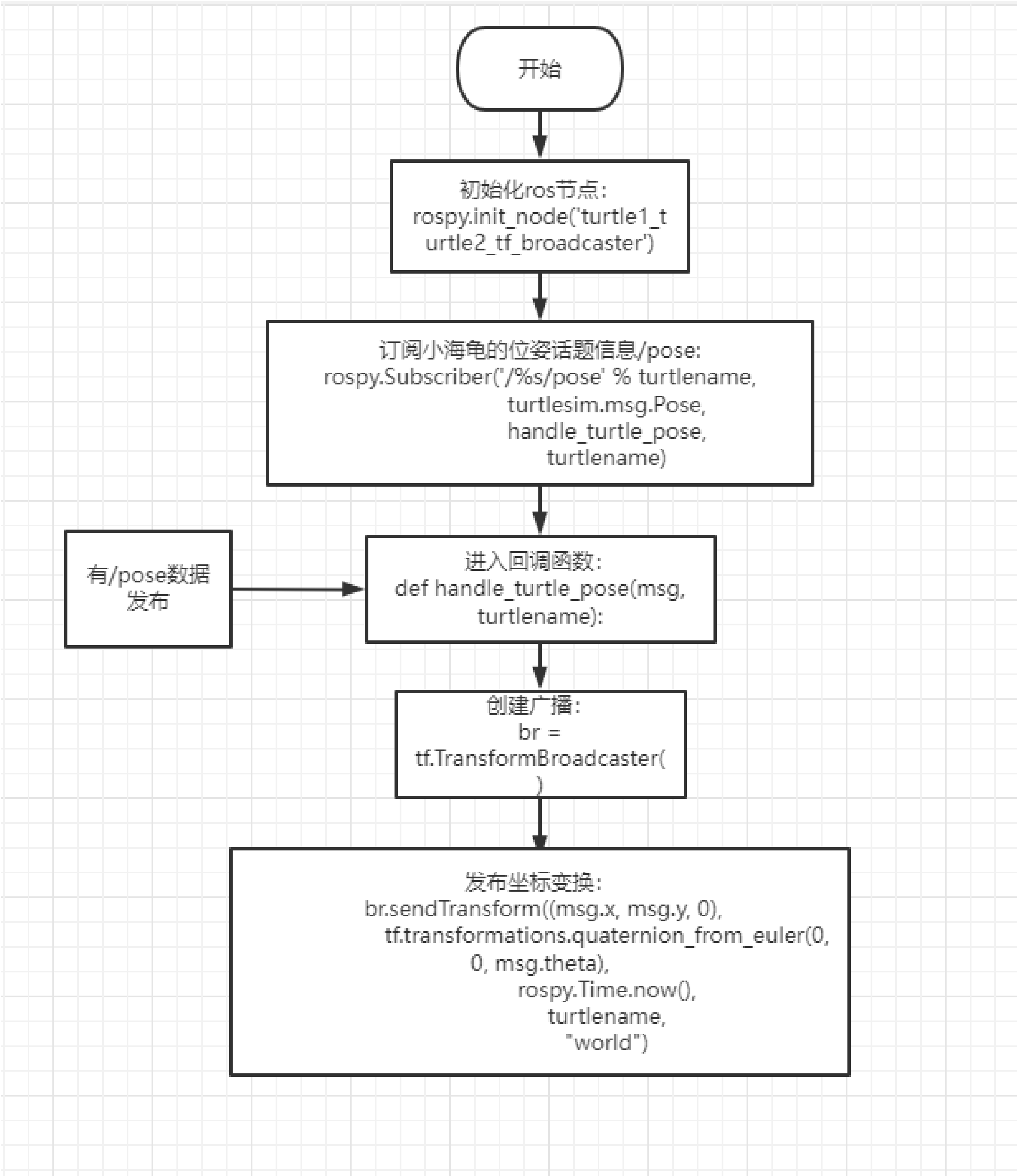
10.2.8. Python language implementation of tf listener

1. In the feature pack learning_ Create a Python file (with a. py file suffix) in the script folder of tf and name it turtle_ tf_ listener.py
2. Copy and paste the program code below into the title_ tf_ In the listener. py file

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import math
import tf
import geometry_msgs.msg
```

```
import turtlesim.srv

if __name__ == '__main__':
    rospy.init_node('turtle_tf_listener')#初始化ros节点

    listener = tf.TransformListener()#初始化一个监听者

    rospy.wait_for_service('spawn')
    #调用服务产生另一只海龟turtle2
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    spawner(8, 6, 0, 'turtle2')
    #声明一个发布者，用来发布turtle2的速度
    turtle_vel = rospy.Publisher('turtle2/cmd_vel',
geometry_msgs.msg.Twist,queue_size=1)

    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        try:
            #查找turtle2和turtle1之间的tf变化
            (trans,rot) = listener.lookupTransform('/turtle2', '/turtle1',
rospy.Time(0))
        except (tf.LookupException, tf.ConnectivityException,
tf.ExtrapolationException):
            continue
        #通过数学计算，算出线速度和角速度，然后发布出去
        angular = 6.0 * math.atan2(trans[1], trans[0])
        linear = 0.8 * math.sqrt(trans[0] ** 2 + trans[1] ** 2)
        cmd = geometry_msgs.msg.Twist()
        cmd.linear.x = linear
        cmd.angular.z = angular
        turtle_vel.publish(cmd)
        rate.sleep()
```

3. Process Flow Chart



10.2.9 Demonstration of startup and operation effects

1. Write a launch fileIn the feature pack directory, create a new folder launch, switch to launch, and create a new launch file named start_ tf_ demo_ Py. launch, copy the following content into it,

```
<launch>

    <!-- 海龟节点-->
    <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
    <!--广播turtle1->world-->
    <node name="turtle1_tf_broadcaster" pkg="learning_tf"
type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle1" />
    </node>
    <!--广播turtle2->world-->
```

```
    <node name="turtle2_tf_broadcaster" pkg="learning_tf"
type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle2" />
    </node>
    <!--监听-->
    <node pkg="learning_tf" type="turtle_tf_listener.py" name="listener" />
    <!--海龟键盘控制节点-->
    <node pkg="turtlesim" type="turtle_teleop_key" name="teleop"
output="screen"/>
</launch>
```

2. firing

```
roslaunch learning_tf start_tf_demo_py.launch
```

After the program runs, click on the launch window with the mouse, press the arrow keys, and Turtle2 will follow Turtle1.

3. The operation results are shown in the following figure