



## 第三章 作业分享



主讲人 周福  
南



# 纲要

➤ 第一题

➤ 第二题

➤ 第三题

➤ 总结

# 第一题

1. 创建一个节点，在其中实现一个订阅者和一个发布者，完成以下功能：

- 发布者：发布海龟速度指令，让海龟圆周运动
- 订阅者：订阅海龟的位置信息，并在终端中周期打印输出

● 首先考虑发布者，发布的是速度信息，可以在turtlesim运行时使用 `rostopic list` 确定海龟的速度topic为 `turtle1/cmd_vel`，知道了消息的名字还不够，我们还要进一步知道消息的类型和格式。

# 第一题

- 接着查阅**官方手册**。当然也可直接搜索相应包中的**头文件**，或者利用**命令**中的**自动补全**，或者利用**rostopic info**，得知**cmd\_vel**的类型为**Twist**，格式如左图，可以采用类似的方法得知位置的topic为**turtle1/pos**，类型为**Pose**，格式如右图。

## geometry\_msgs/Twist Message

File: `geometry_msgs/Twist.msg`

### Raw Message Definition

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

### Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

[http://docs.ros.org/en/melodic/api/geometry\\_msgs/html/msg/Pose.html](http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Pose.html)

## turtlesim/Pose Message

File: `turtlesim/Pose.msg`

### Raw Message Definition

```
float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity
```

### Compact Message Definition

```
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

# 第一题

- 然后就可以愉快地编程啦。需要注意在一个节点里同时实现订阅者和发布者。在此给出两种实现。注意在头文件中包含 geometry\_msgs/Twist 和 turtlesim/Pose.h，并在实现的时候根据消息的格式收发信息。

## 第一种

```
void chatterCallback(const turtlesim::Pose& msg){
    ROS_INFO("Turtle is at: [%f,%f,%f]", (double)(msg.x), (double)(msg.y), (double)(msg.theta));
}

int main(int argc, char **argv){
    ros::init(argc, argv, "turtle_circle_position");
    ros::NodeHandle n;
    ros::Publisher turtle_circle_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1);
    ros::Subscriber turtle_circle_sub = n.subscribe("/turtle1/pose", 1, chatterCallback);
    ros::Rate loop_rate(10);
    while(ros::ok()){
        geometry_msgs::Twist msg;
        msg.linear.x = 1;
        msg.linear.y = 0;
        msg.linear.z = 0;
        msg.angular.x = 0;
        msg.angular.y = 0;
        msg.angular.z = 1;
        ROS_INFO("Turtle is turning around with linear velocity(%f,%f,%f) angular velocity(%f,%f,%f)",
            (double)msg.linear.x, (double)msg.linear.y, (double)msg.linear.z, (double)msg.angular.x, (double)msg.angular.y, (double)msg.angular.z);
        turtle_circle_pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
    }

    return 0;
}
```

# 第一题

- 采用了面向对象的写法，参考了ROS社区论坛的帖子

## 第二种

```
class TurtleCirclePosition{
public:
    TurtleCirclePosition(){
        pub_ = n_.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1);
        sub_ = n_.subscribe("/turtle1/pose", 1, &TurtleCirclePosition::callback, this);
    }

    void callback(const turtlesim::Pose& input){
        ROS_INFO("Turtle is at: [%lf,%lf,%lf]", (double)(input.x), (double)(input.y), (double)(input.theta));
        geometry_msgs::Twist msg;
        msg.linear.x = 1;
        msg.linear.y = 0;
        msg.linear.z = 0;
        msg.angular.x = 0;
        msg.angular.y = 0;
        msg.angular.z = 1;
        ROS_INFO("Turtle is turning around with linear velocity(%lf,%lf,%lf) angular velocity(%lf,%lf,%lf)",
            [(double)]msg.linear.x, (double)msg.linear.y, (double)msg.linear.z, (double)msg.angular.x, (double)msg.angular.y, (double)msg.angular.z);
        pub_.publish(msg);
    }
private:
    ros::NodeHandle n_;
    ros::Publisher pub_;
    ros::Subscriber sub_;
};

int main(int argc, char **argv){
    ros::init(argc, argv, "turtle_circle_position");
    TurtleCirclePosition TCPObject;
    ros::spin();
    return 0;
}
```

# 纲要

---

➤ 第一题

➤ 第二题

➤ 第三题

## 第二题

### 2. 创建另外一个节点，在其中实现一个客户端，完成以下功能：

- 客户端：请求海龟诞生的服务，在仿真器中产生一只新的海龟

● 参考第一题的方法定位到Spawn服务和它的消息类型及格式，如下图

#### [turtlesim/Spawn Service](#)

File: turtlesim/Spawn.srv

##### Raw Message Definition

```
float32 x
float32 y
float32 theta
string name # Optional. A unique name will be created and returned if this is empty
---
string name
```

##### Compact Message Definition

```
float32 x
float32 y
float32 theta
string name
string name
```



## 第二题

- 接着按照课程视频讲解的编写客户端即可，每启动一次客户端，就向服务器发送一次spawn服务请求。注意包含<turtlesim/Spawn.h>头文件。

```
int main(int argc, char** argv){
    ros::init(argc, argv, "turtle_spawner");
    ros::NodeHandle node;

    ros::service::waitForService("/spawn");
    ros::ServiceClient turtle_spawner = node.serviceClient<turtlesim::Spawn>("/spawn");

    //ROS_INFO("%d,%s", argc, argv[1]);

    turtlesim::Spawn srv;
    srv.request.x = 1.0;
    srv.request.y = 1.0;
    if(argc > 1) srv.request.name = argv[1];

    ROS_INFO("Call service to spawn a turtle at[%lf,%lf], named %s", (double)srv.request.x, (double)srv.request.y, argv[1]);
    turtle_spawner.call(srv);

    ROS_INFO("spawned a baby, OK");
    return 0;
}
```

# 纲要

---

➤ 第一题

➤ 第二题

➤ 第三题

## 第三题

### 3. 综合运用话题与服务编程、命令行使用，实现以下场景：

小R想要实现一个海龟运动控制的功能包，需要具备以下功能（以下指令的接收方均为该功能包中的节点）：

- 通过命令行发送新生海龟的名字，即可在界面中产生一只海龟，并且位置不重叠；
- 通过命令行发送指令控制界面中任意海龟圆周运动的启动/停止，速度可通过命令行控制；

你可以帮助小R实现这个功能包么？

- 要在命令行发送名字产生乌龟，其实就是向一个能产生乌龟的服务器请求服务。这个服务器需要能够产生一个位置不（与已有乌龟）重叠的海龟。实际上，对于 turtlesim，要产生新乌龟，我们只能通过它自带的 spawn service，所以我们需要编写的服务器，实际上只是个中介，本质还是要调用 spawn service，只不过这个中介同时确保了位置不重叠。在服务消息的格式上，因为只需要一个名字，我们可以自定义一个 TurtleSrv，request 只包含名字字符串，response 只包含结果字符串。

## 第三题

- 调用spawn服务的部分和第二题几乎一样，不过位置要通过一些方法确保不重叠，比如和已产生的乌龟比较或者预先设定好随机序列或网格。我采用的就是后一种办法。

```
//src/spawn_a_turtle.cpp
//该源文件可以提供服务来生成一只指定名字的海龟。
#include <ros/ros.h>
#include <turtlesim/Spawn.h>
#include <turtle_controller/TurtleSrv.h>
#include <sstream>
//#include <stdlib.h>
```

注意包含头文件

```
int main(int argc, char** argv){
    ros::init(argc, argv, "spawn_a_turtle");
    ros::service::waitForService("/spawn");
    SpawnATurtle SATObject;
    ros::spin();
    return 0;
}
```

```
class SpawnATurtle{
public:
    SpawnATurtle(){
        cli_ = n_.serviceClient<turtlesim::Spawn>("/spawn");
        ser_ = n_.advertiseService("/spawn_a_turtle", &SpawnATurtle::callback, this);
        for(int i=0;i<100;i++){
            posRand[i]=i;
        }
        std::random_shuffle(posRand,posRand+100);
        posIdx = rand()%100;
        posIdy = rand()%100;
    }

    bool callback(turtle_controller::TurtleSrv::Request &req,
        turtle_controller::TurtleSrv::Response &res){
        turtlesim::Spawn srv;
        srv.request.x = 0.5 + 0.1*posRand[posIdx];
        srv.request.y = 0.5 + 0.1*posRand[posIdy];
        srv.request.name = req.name;
        posIdx = (posIdx+1)%100; //防止索引溢出
        posIdy = (posIdy+1)%100;
        if(cli_.call(srv)){
            std::stringstream ss;
            ss<<"successfully spawn a turtle "<<srv.request.name<<" at ("<<srv.request.x<<","<<srv.request.y<<");
            res.result = ss.str();
        }else{
            res.result = "wrong! see errors in turtlesim console.";
        }
        return true;
    }

private:
    ros::NodeHandle n_;
    ros::ServiceServer ser_;
    ros::ServiceClient cli_;
    int posRand[100];
    int posIdx,posIdy;
};
```

## 第三题

- 至于控制指定乌龟的运动停止和速度，就需要写一个服务器，能向对应的cmd\_vel发送速度指令。这里就要自定义一个TurtleControlSrv, 然后我们的写的服务器收到名字和速度调整的命令，再根据名字发布到对应的cmd\_vel即可。

```
string name
string state
float64 c_velocity
---
string result
```

## 第三题

- 主要代码如下，根据不同的state命令来调整速度，然后发给对应名字的cmd\_vel。

//该源文件可以提供服务来控制任意一只海龟的启动停止，并调节速度。

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <turtle_controller/TurtleControlSrv.h>
#include <sstream>
//#include <stdlib.h>
```

```
int main(int argc, char** argv){
    ros::init(argc, argv, "turtle_control_circle");
    TurtleControlCircle TCCObject;
    ros::spin();
    return 0;
}
```

```
class TurtleControlCircle{
public:
    TurtleControlCircle(){
        ser_ = n_.advertiseService("/turtle_control_circle", &TurtleControlCircle::callback, this);
    }
    bool callback(turtle_controller::TurtleControlSrv::Request &req,
        turtle_controller::TurtleControlSrv::Response &res){
        //根据req.name和state来给对应的turtle发布信息
        geometry_msgs::Twist msg;
        if(req.state=="start"){
            msg.linear.x = 1.0;msg.linear.y = 0.0;msg.linear.z = 0.0;
            msg.angular.x = 0.0;msg.angular.y = 0.0;msg.angular.z = 1.0;
        }else if(req.state=="stop"){
            msg.linear.x = 0.0;msg.linear.y = 0.0;msg.linear.z = 0.0;
            msg.angular.x = 0.0;msg.angular.y = 0.0;msg.angular.z = 0.0;
        }else if(req.state=="adjust"){
            msg.linear.x = req.c_velocity;msg.linear.y = 0.0;msg.linear.z = 0.0;
            msg.angular.x = 0.0;msg.angular.y = 0.0;msg.angular.z = req.c_velocity;
        }
        pub_ = n_.advertise<geometry_msgs::Twist>("/"+req.name+"/cmd_vel", 10);
        ros::Rate loop_rate(10);
        while(ros::ok()){
            pub_.publish(msg);
            ros::spinOnce();
            loop_rate.sleep();
        }
        return true;
    }
private:
    ros::NodeHandle n_;
    ros::ServiceServer ser_;
    ros::Publisher pub_;
    ros::Subscriber sub_;
};
```

# 总结

- 通信编程的核心还是节点间的关系（发布、订阅；服务、客户）以及话题消息的格式。调试的时候耐心一点，总能搞定的！







深蓝学院  
shenlanxueyuan.com

感谢各位聆听 !  
Thanks for Listening

