SimOneAPI:

1. 初始化：

param[in]

hostVehicleId: host vehicle ID(from 0 to 9)
isFrameSync: synchronize frame or not
serverIP: BridgeIO server ip
port: BridgeIO server port
startCase: callback func which being called before case start
endCase: callback func which being called after case end
registerNodeId: not in use

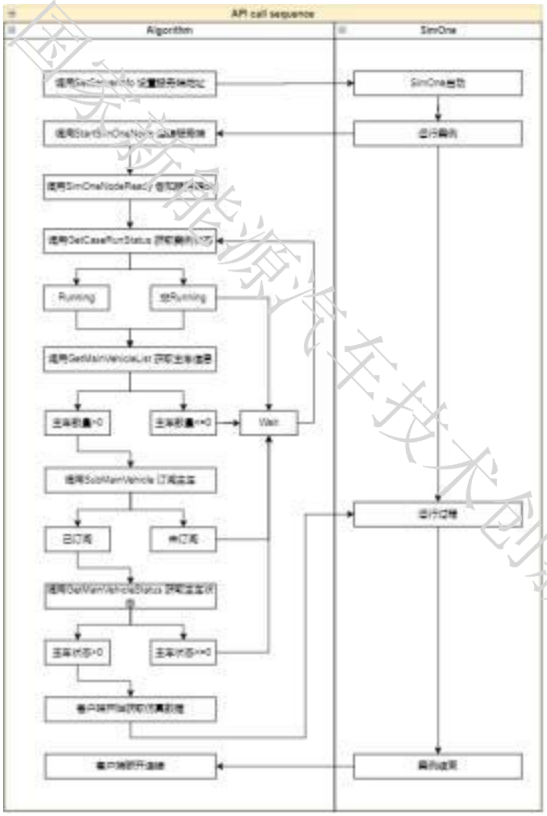
bool InitSimOneAPI(const char* mainVehicleId = "0", bool isFrameSync =false, const char *serverIP = "127.0.0.1", int port = 23789, void(*startCase)()=0, void(*endCase)()=0, int registerNodeId=0);

帧同步：isFrameSync 为 true:

intframe SimOneAPI::Wait();
SimOneAPI 调用
SimOneAPI::NextFrame(frame);

退出：

1. 获取案例运行情况（运行中，停止）
   Return：Stop,Running
   ESimOne_Case_Status GetCaseRunStatus();
2. 退出 API node
   bool TerminateSimOneAPI();



2. 感知车道线

回调方式：

```
auto function = [](const char* mainVehicleId, const char* sensorId, SimOne_Data_LaneInfo
*pDetections) {
  /* data processing */
};
SimOneAPI::SetSensorLaneInfoCB(function);
```

异步方式：

```
std::unique_ptr<SimOne_Data_LaneInfo> pDetections =
std::make_unique<SimOne_Data_LaneInfo>();
SimOneAPI::GetSensorLaneInfo(mainVehicleId.c_str(), "sensorFusion1", pDetections.get());
```

车道消息：

```
struct SimOne_Data_LaneInfo :public SimOne_Data
{
 int id = 0;//Lane ID
 ESimOne_Lane_Type laneType;//Lane type
 int laneLeftID = 0;//The Lane's leftLane ID
 int laneRightID = 0;//The Lane's rightLane ID
 int lanePredecessorID[SOSM_SENSOR_LANE_OBJECT_SIZE_MAX];//total of lane predecessor
ID,max 256
 int laneSuccessorID[SOSM_SENSOR_LANE_OBJECT_SIZE_MAX];//total of lane successor
ID,max 256
 SimOne_Data_LaneLineInfo l_Line;//lane left lineBoundary;
 SimOne_Data_LaneLineInfo c_Line;//lane center lineBoundary;
 SimOne_Data_LaneLineInfo r_Line;//lane right lineBoundary;
 SimOne_Data_LaneLineInfo ll_Line;//laneleft left lineBoundary;
 SimOne_Data_LaneLineInfo rr_Line;//laneright right lineBoundary;
};
```

3. 主车状态

回调方式：

```
auto function = []( const char* mainVehicleId, SimOne_Data_Gps *pGps){
  /* data processing */
};
SimOneAPI::SetGpsUpdateCB(function);
```

异步方式：

```
std::unique_ptr<SimOne_Data_Gps> pGps = std::make_unique<SimOne_Data_Gps>();
SimOneAPI::GetGps(mainVehicleId.c_str(), pGps.get());
```

主车状态消息：

```
struct SimOne_Data_Gps : public SimOne_Data
{
 float posX; // Position X on Opendrive (by meter)
 float posY; // Position Y on Opendrive (by meter)
 float posZ; // Position Z on Opendrive (by meter)
```

```
    float oriX; // Rotation X on Opendrive (by radian)
    float oriY; // Rotation Y on Opendrive (by radian)
    float oriZ; // Rotation Z on Opendrive (by radian)
    float velX; // MainVehicle Velocity X on Opendrive (by meter)
    float velY; // MainVehicle Velocity Y on Opendrive (by meter)
    float velZ; // MainVehicle Velocity Z on Opendrive (by meter)
    float throttle; //MainVehicle throttle
    float brake; //MainVehicle brake;
    float steering; //MainVehicle Wheel Steering angle (deg)
    int gear; // MainVehicle gear position
    float accelX; // MainVehicle Acceleration X on Opendrive (by meter)
    float accelY; // MainVehicle Acceleration Y on Opendrive (by meter)
    float accelZ; // MainVehicle Acceleration Z on Opendrive (by meter)
    float angVelX; // MainVehicle Angular Velocity X on Opendrive (by meter)
    float angVelY; // MainVehicle Angular Velocity Y onOpendrive (by meter)
    float angVelZ; // MainVehicle Angular Velocity Z on Opendrive (by meter)
    float wheelSpeedFL;  // Speed of front left wheel (by meter/sec)
    float wheelSpeedFR;  // Speed of front right wheel (by meter/sec)
    float wheelSpeedRL;  // Speed of rear left wheel (by meter/sec)
    float wheelSpeedRR;  // Speed of rear right wheel (by meter/sec)
    float engineRpm;// Speed of engine (by r/min)
    float odometer;      // odometer in meter.
    int extraStateSize;
    float extraStates[SOSM_EXTRA_STATES_SIZE_MAX];// vehicle states subscripted by
MainVehicleExtraDataIndics message
};
```

4. 目标及传感器

回调方式：

```
auto function = [](const char* MainVehicleID, const char* sensorId,
SimOne_Data_SensorDetections* pGroundtruth)
{
  /* data processing */
};
SimOneAPI::SetSensorDetectionsUpdateCB(function);
```

异步方式：

```
std::unique_ptr<SimOne_Data_SensorDetections> pGroundtruth =
std::make_unique<SimOne_Data_SensorDetections>();
// "sensorFusion1" "objectBasedCamera1" "objectBasedLidar1" "perfectPerception1"
SimOneAPI::GetSensorDetections(mainVehicleId.c_str(), "perfectPerception1",
pGroundtruth.get());
```

目标级传感器消息：

```
struct SimOne_Data_SensorDetections_Entry
{
int id;                          // Detection Object ID
ESimOne_Obstacle_Type type;      // Detection Object Type
float posX;                      // Detection Object Position X in meter
float posY;                      // Detection Object Position Y in meter
float posZ;                      // Detection Object Position Z in meter
float oriX;                 // Rotation X in radian
float oriY;                 // Rotation Y in radian
float oriZ;                 // Rotation Z in radian
float length;               // Detection Object Length in meter
float width;                // Detection Object Width in meter
float height;               // Detection Object Height in meter
float range;                // Detection Object relative range in meter
float velX;                      // Detection Object Velocity X
float velY;                      // Detection Object Velocity Y
float velZ;                      // Detection Object Velocity Z
float accelX;               // Detection Object accel X
float accelY;               // Detection Object accel Y
float accelZ;               // Detection Object accel Z
float probability;          // Detection probability
float relativePosX;         // Relative position X in sensor space
float relativePosY;         // Relative position Y in sensor space
float relativePosZ;         // Relative position Z in sensor space
float relativeRotX;         // Relative rotation X in sensor space
```

```
 float relativeRotY;        // Relative rotation Y in sensor space
 float relativeRotZ;        // Relative rotation Z in sensor space
 float relativeVelX;        // Relative velocity X in sensor space
 float relativeVelY;        // Relative velocity Y insensor space
 float relativeVelZ;        // Relative velocity Z in sensor space
 float bbox2dMinX = 0;  // bbox2d minX in pixel if have
 float bbox2dMinY = 0;  // bbox2d minY in pixel if have
 float bbox2dMaxX = 0;  // bbox2d maxX in pixel if have
 float bbox2dMaxY = 0;  // bbox2d maxY in pixel if have
};
```

5. 物理级传感器

Camera:

回调方式：

```
auto function = [](SimOne_Streaming_Image *pImage)
{
  /* data processing */
};
SimOneAPI::SetStreamingImageUpdateCB(/*ip*/, /*port*/, function);
```

异步方式：

```
std::unique_ptr<SimOne_Streaming_Image > pImage =
std::make_unique<SimOne_Streaming_Image>();
GetStreamingImage(/*ip*/, /*port*/, pImage.get());
```

物理级摄像头数据：

```
struct SimOne_Streaming_Image : public SimOne_Streaming_Data
{
 int width; // Image resolution width 1920 max
 int height; // Image resolution height 1200 max
 ESimOne_Streaming_Image_Format format; // Image format, RGB only for now
 int imageDataSize; // image data size
 char imageData[SOSM_IMAGE_DATA_SIZE_MAX]; // 1920 x 1200 x 3 max
};
```

Lidar:

回调方式：

```
auto function = [](SimOne_Streaming_Point_Cloud *pPointCloud)
{
  /* data processing */
};
SimOneAPI::SetStreamingPointCloudUpdateCB(/*ip*/, /*port*/, /*InfoPort*/, function);
```

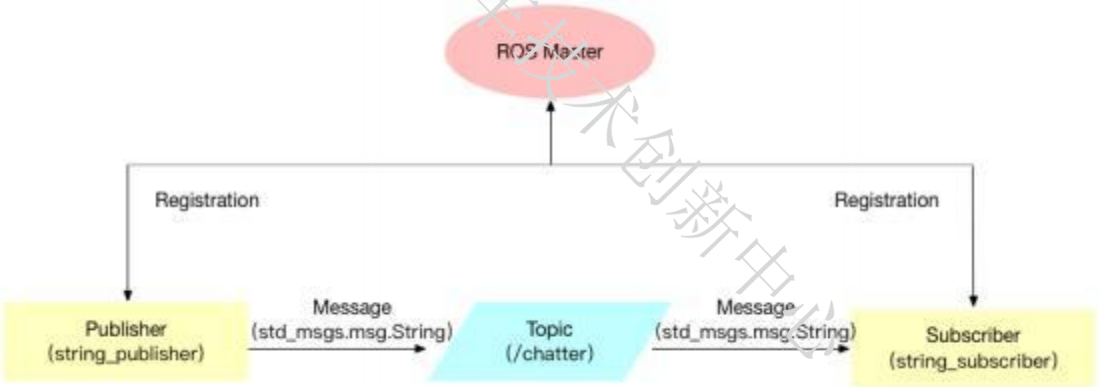
异步方式：

```
std::unique_ptr<SimOne_Streaming_Point_Cloud> pPointCloud =
std::make_unique<SimOne_Streaming_Point_Cloud>();
GetStreamingPointCloud(/*ip*/, /*port*/, /*InfoPort*/, pPointCloud.get());
```

物理级激光雷达数据：

```
struct SimOne_Streaming_Point_Cloud : public SimOne_Streaming_Data
{
 int width;
 int height;
 int pointStep;
 int pointCloudDataSize;
 char pointCloudData[SOSIM_POINT_DATA_SIZE_MAX];
};
```

第三方定制化通信接口（ROS 接口示例）



ROS Msg 消息转换

将 SimOne API 数据格式转为 ROS 可发布/订阅的标准数据格式：

1. 对照 SimOne API 接口数据，按照 C++ 数据类型与 ROS msg 数据类型的映射关系，编写 ROS API msg 消息文件

2. 创建 ROS 工程 Worksapce

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

3. 创建功能包

```
cd ~/catkin_ws/src
catkin_create_pkg <package_name> roscpp std_msgs
```

4. 将编写好的 msg 消息文件放入功能包

```
mkdir ~/catkin_ws/src/<package_name>/msg
```

5. 在~/catkin_ws/src/<package_name>/package.xml 添加功能依赖

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

6. 在~/catkin_ws/src/<package_name>/CMakeLists.txt 添加编译选项

```
find_package(catkin REQUIRED COMPONENTS
    roscpp
    std_msgs
    message_generation
)
```

```
add_message_files(FILES
    <ROS API  msg 消息文件 .msg>
)
generate_messages(DEPENDENCIES
    std_msgs
)
catkin_package(
    CATKIN_DEPENDS roscpp std_msgs message_runtime
)
```

7. 编译功能包

```
cd  ~/catkin_ws
catkin_make –DCATKIN_WHITELIST_PACKAGES=”  package_name”
```

8. 编译工作空间

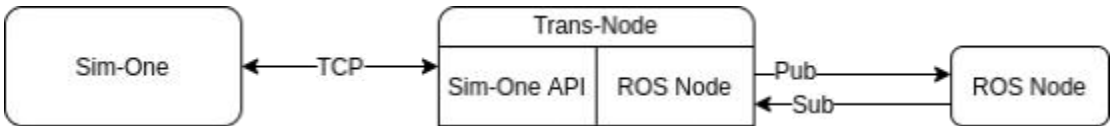
```
cd  ~/catkin_ws
catkin_make
```

9. 检查 msg 文件生成的 C++ 头文件

```
~/catkin_ws/devel/include/<package_name>/
```

10. 将生成的 C++ 头文件融入 ROS API 工程

通信节点架构图：



ROS API 工程结构

ROS/

```
├── CMakeLists.txt   # 工程编译脚本
├── gen_make_debug.sh  # debug 工程编译环境生成脚本
├── gen_make_release.sh  # release 工程编译环境生成脚本
├── include    # 工程头文件
├── lib    # 工程依赖库
├── run    # ROS API 节点可执行程序生成目录
└── src    # ROS API 工程源文件
```

ROS API 工程编译：

执行 gen_make_debug.sh / gen_make_release.sh

cd build_debug / build_release

make # ROS API 节点程序生成路径：run/trans_node_ros

ROS API 节点运行：

1. 启动 roscore

2. 执行 trans_node_ros 运行时程序将读取 config.ini 配置文件参数

config.ini 运行参数配置文件：

```
[BridgeIO]# Sim-One API 客户端连接设置
BridgeIO_IP=10.66.9.111# SimOne BridgeIO 节点 IP
[HostVehicle]# Sim-One 仿真主车设置
Vehicle_ID=0# Sim-One 仿真主车 ID
[Sensor]# Sim-One 传感器通信配置
IMG_IP=10.66.9.244# 图像数据 UDP 接收端 IP
IMG_PORT=13944# 图像数据 UDP 接收端 Port
```

PCD_IP=10.66.9.244# 点云数据 UDP 接收端 IP

PCD_PORT=6699# 点云数据 UDP 接收端 Port

PCD_PORT_INFO=7788# 点云数据 UDP 接收端 InfoPort

[ROS]# ROS 消息配置

GPS_Topic=/gps# Gps 消息发布 Topic

GroundTruth_Topic=/ground_truth# 感知物体真值消息发布 Topic

Image_Topic=/image# 图像数据消息发布 Topic

PointCloud_Topic=/point_cloud# 点云数据消息发布 Topic

Radar_Topic=/radar_detection# 毫米波雷达数据消息发布 Topic

Sensor_Topic=/sensor_detection# 目标及传感器真值数据消息发布 Topic

LaneInfo_Topic=/lane_info# 感知车道/车道线数据消息发布 Topic

CTL_Topic=/control# 主车控制（油门/刹车/方向）数据消息订阅Topic

POSE_CTL_Topic=/pose_control# 主车控制（离散点）数据消息订阅 Topic

ROS API 节点 Trans-Node：

1. 通过 Sim-One API 获取仿真感知数据

主车 Gps 消息回调

bool SetGpsUpdateCB(void(*cb)(const char* mainVehicleId,
SimOne_Data_Gps *pGps));

获取仿真感知物体真值回调

bool SetGroundTruthUpdateCB(void(*cb)(const char* mainVehicleId,
SimOne_Data_Obstacle *pObstacle));

获取摄像头图像数据回调

```
bool SetStreamingImageUpdateCB(const char* ip, unsigned short port,
void(*cb)(SimOne_Streaming_Image *pImage));
```

获取激光雷达点云数据回调

```
bool SetStreamingPointCloudUpdateCB(const char* ip, unsigned short
port, unsigned short infoPort, void(*cb)(SimOne_Streaming_Point_Cloud
*pPointCloud));
```

获取毫米波雷达目标信息回调

```
bool SetRadarDetectionsUpdateCB(void(*cb)(const char* mainVehicleId,
const char* sensorId, SimOne_Data_RadarDetection *pDetections));
```

获取目标及传感器真知数据回调

```
bool SetSensorDetectionsUpdateCB(void(*cb)(const char* mainVehicleId,
const char* sensorId, SimOne_Data_SensorDetections *pGroundtruth));
```

获取感知车道与车道线数据回调

```
bool GetSensorLaneInfo(const char* mainVehicleId, const char* sensorId,
SimOne_Data_LaneInfo *pLaneInfo);
```

2. 通过 ROS Publisher 将感知消息发布到相应 Topic 上

发布 Gps 消息

```
pub_gps = handle_gps.advertise<msg_gen::gps>(gps_topic.c_str(), 1);
pub_gps_p->publish(gps_d);
```

发布仿真感知物体真值消息

```
pub_ground_truth =
handle_ground_truth.advertise<msg_gen::obstacle>(ground_truth_topic.c_str(), 1);
```

```
            pub_ground_truth_p->publish(obstacle_d);
```

发布摄像头图像数据消息

```
            pub_image =
handle_image.advertise<sensor_msgs::Image>(image_topic.c_str(), 1);
            pub_image_p->publish(img_d);
```

发布激光雷达点云数据消息

```
            pub_point_cloud =
handle_point_cloud.advertise<sensor_msgs::PointCloud2>(point_cloud_topic.c_str(
), 1);
            pub_point_cloud_p->publish(point_cloud_d);
```

发布毫米波雷达目标信息

```
            pub_radar =
handle_radar.advertise<msg_gen::radardetection>(radar_topic.c_str(), 1);
            pub_radar_p->publish(radar_detection_d);
```

发布目标及传感器真值消息

```
            pub_sensor =
handle_sensor.advertise<msg_gen::sensordetections>(sensor_topic.c_str(), 1);
            pub_sensor_p->publish(sensor_detections_d);
```

发布感知车道/车道线消息

```
            pub_laneinfo =
handle_laneinfo.advertise<msg_gen::laneinfo>(lane_info_topic.c_str(), 1);
            pub_laneinfo_p->publish(lane_info_d);
```

3. 通过 ROS Subscriber 订阅主车控制相关 Topic

订阅主车控制消息（离散点 控制）

```
sub_ctl = handle_ctl.subscribe(ctl_topic.c_str(), 1,
&ros_trans_node::rcv_ctl_cb, this);
```

订阅主车控制消息（油门/刹车/方向 控制）

```
sub_pose_ctl = handle_pose_ctl.subscribe(pose_ctl_topic.c_str(), 1,
&ros_trans_node::rcv_pose_ctl_cb, this);
```

4. 通过 Sim-One API 设置主车控制参数

根据离散点设置主车位置

```
SimOneAPI::SetPose(0, &pose_ctl);
```

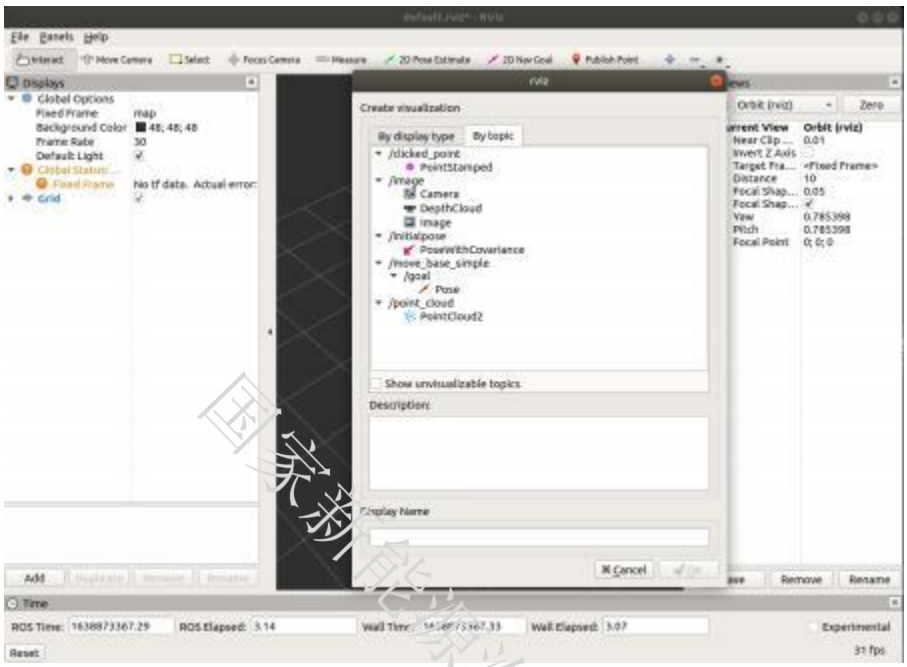
设置主车控制参数

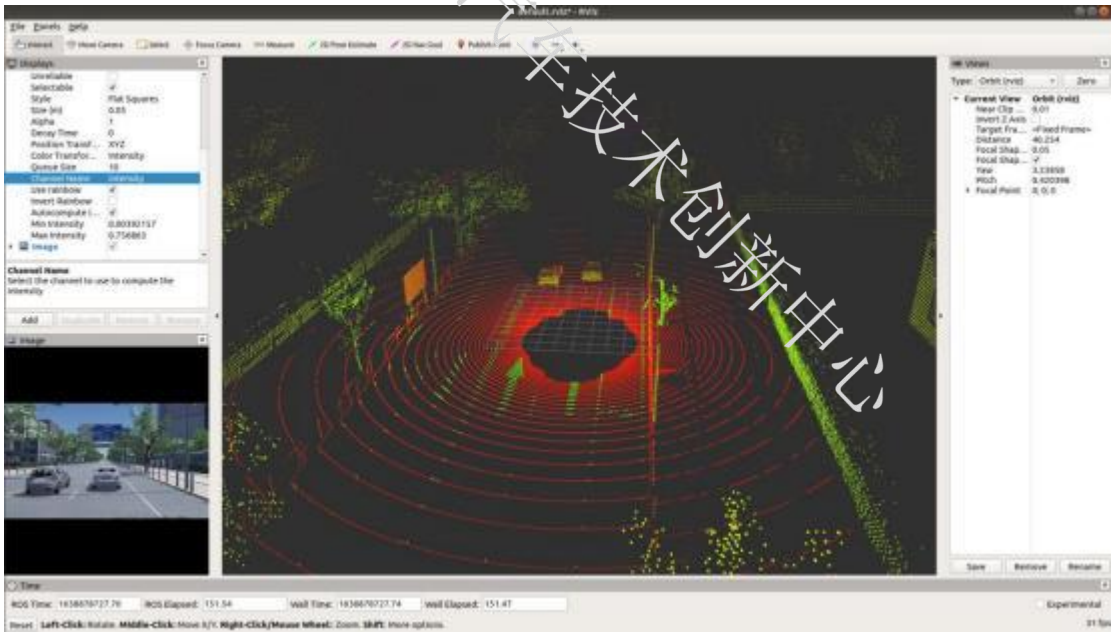
```
SimOneAPI::SetDrive(vehicle_id.c_str(), pCtrl.get());
```

消息验证

图像数据：rviz 可视化工具订阅 Image Topic 消息显示图像

点运数据：rviz 可视化工具订阅 PointCloud Topic 消息显示点云图像

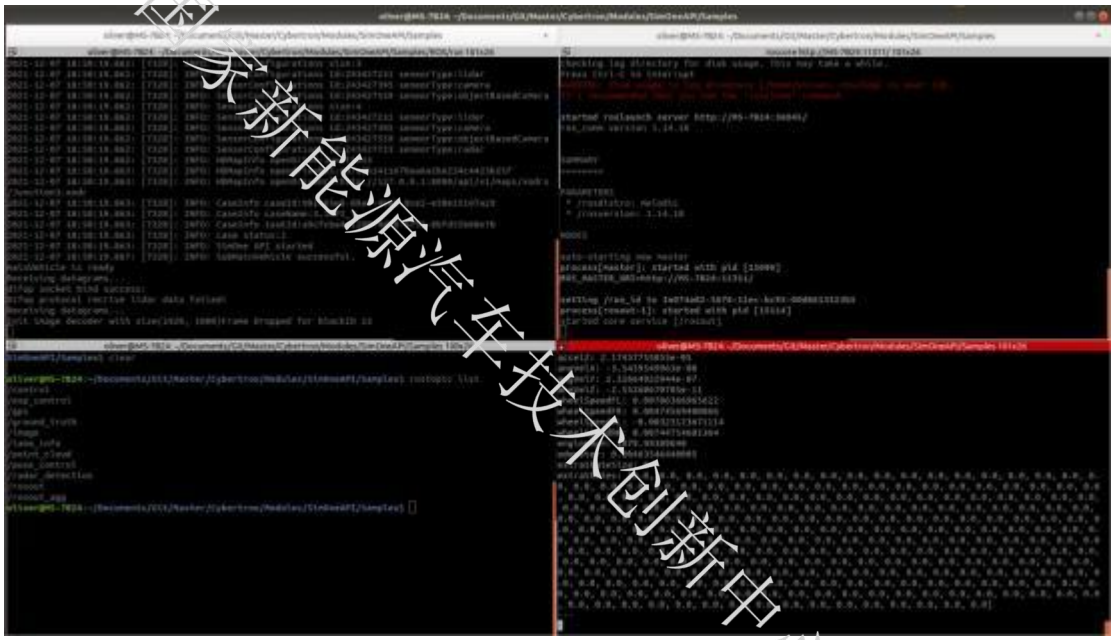




结构化数据：rostopic echo 命令监听相应 Topic 查看数据输出

source ~/catkin_ws/devel/setup.bash

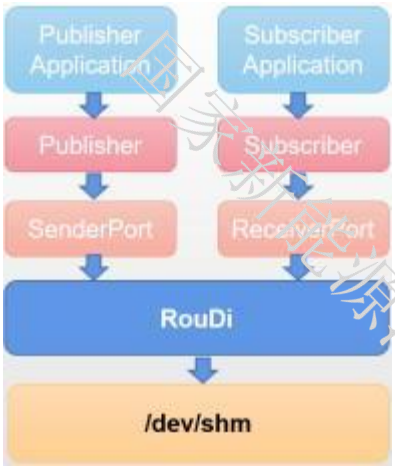
rostopic echo /gps



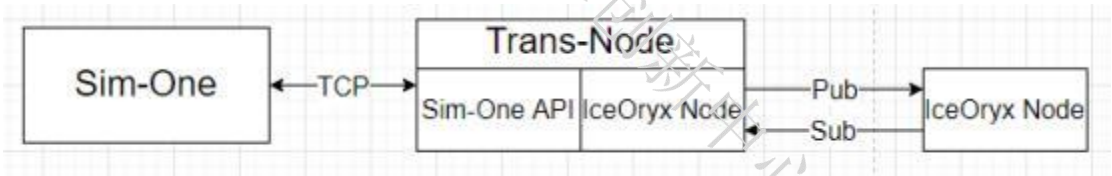
其它第三方接口：

1. IceOryx 接口

   IceOryx 传输机制：

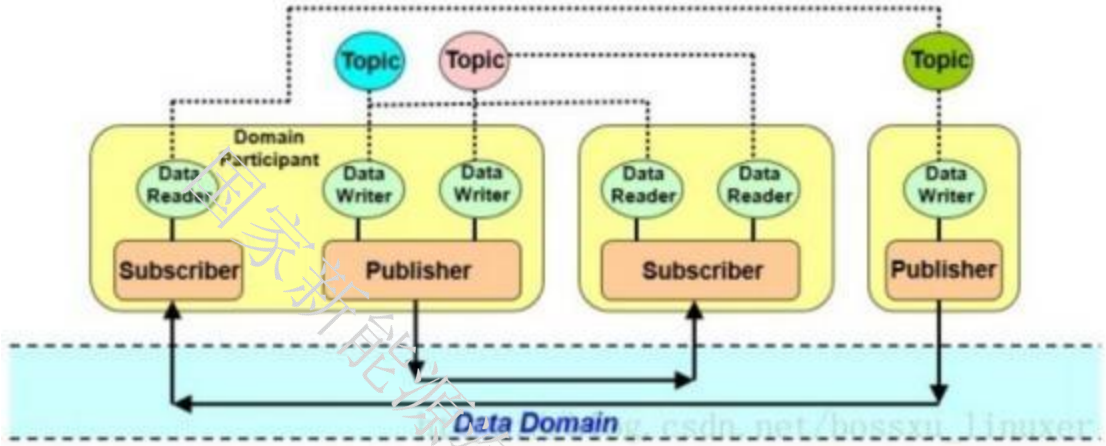


   SimOne 接口消息节点：



2. DDS 接口

   DDS 传输机制：

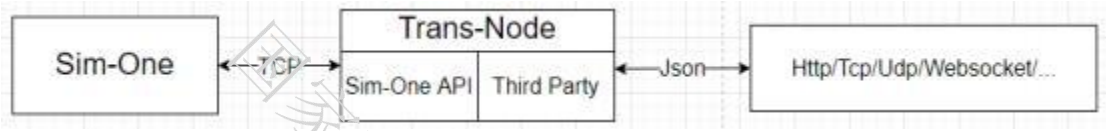


1. 发布者（publisher）设置发布的主题（topic），数据读者(subscriber)订阅感兴趣的主题。

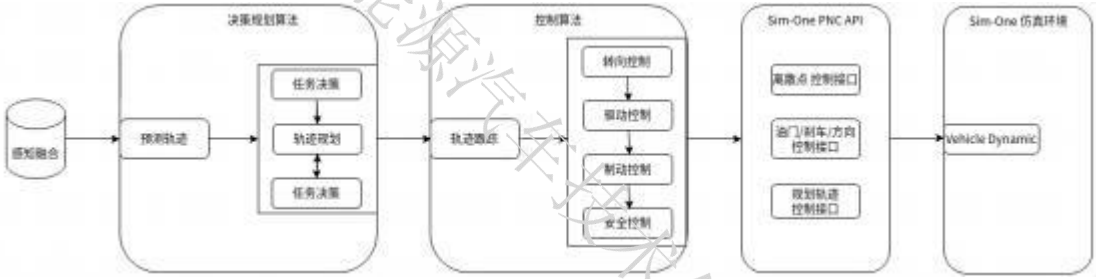
   publisher 作为发布者角色，至少包含一个 DataWriter，并负责创建，删除和管理 datawriter。同样，subscriber 作为订阅者，至少与一个 datareader 关联，并负责发布数据，数据发布者通过调用 datawriter 的write 函数发布数据，但数据不会立刻被送出，实际的消息产生是通过 publisher 和 Qos 综合控制的。datareader 负责订阅数据，订阅方式可采用异步方式（listener），同步方式和非阻塞三种

2. DDS 以 IDL（接口描述语言）作为定义交换结构的格式，根据 SimOneAPI 提供的接口数据结构编写 IDL
3. QoS（Quality of Service）服务质量：指一个网络能够利用各种基础技术，为指定的网络通信提供更好的服务能力，是网络的一种安全机制， 是用来解决网络延迟和阻塞等问题的一种技术

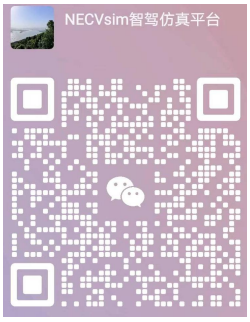
3. 其它



算法联合仿真训练优化



接口程序通过 API 获取 SimOne 仿真感知数据，并将数据转换为第三方消息格式发布到指定订阅节点上。驾驶算法订阅到相应消息，将数据解析并传入到规控/感知模块做算法相关训练。算法最终将解算后的控制消息发布到仿真主车控制消息订阅节点，该节点将控制参数通过 API 传入仿真主车动力学节点来驱动 SimOne 场景中的主车行使。驾驶算法可单独订阅个别消息验证局部模块的正确性，也可形成算法、仿真的完整闭环实现驾驶算法的整体验证。



（相关咨询可联系）　　**微信公众号：工创大赛智能网联竞赛系统**

竞赛平台相关信息中心地址：https://dc.nevc.com.cn:4430/competition/icvsim/srs/