ROS2迁移指南

1、API迁移

1、初始化变化

```
ros1:
  ros::init(argc, argv, MODULE_NAME);
  auto& msg_server = ROSServer::instance();

ros2:
  rclcpp::init(argc, argv);
  auto& msg_server = ROSServer::instance(std::string(MODULE_NODE));

```

2、部分静态API变化

```
ros1          ->            ros2
ros::spin();
rclcpp::spin(msg_server.get_node());//std::shared_ptr<rclcpp::Node> node
ros::spinOnce();
rclcpp::spin_some(msg_server.get_node());//std::shared_ptr<rclcpp::Node>
node
ros::shutdown();           rclcpp::shutdown();
ros::ok();    rclcpp::ok();
ros::Rate loop_rate(10) rclcpp::Rate loop_rate(10)
ros::Time    rclcpp::Time
ros::Time::now()    rclcpp::Clock(RCL_ROS_TIME).now()
```

3、定时器变化

```
ros1:
msg_server.create_timer([](){},ros::Duration(1.0 / 20));
ros2:
msg_server.create_timer([](){},50ms);
```

4、参数读取变化

```
1、获取其他节点参数:
ros1:
```

```
msg_server.getParam("/vcu_to_ros_node/vcu_can_topic_path", can_topic_path);
ros2:
msg_server.getGlobalParam("/vcu_to_ros_node/vcu_can_topic_path",
can_topic_path);
msg_server.getGlobalParam("/mqtt_bridge_node/"mqtt.client.protocol",protocol
);
```

5、获取包的share路径

```
ros1:
1、cpp:
#include <ros/package.h>
std::string package_share_directory =
ros::package::getPath(package_name.toStdString())
2、CmakeList.txt:
find_package(roslib)
catkin_package括号内增加roslib
3、package.xml
<build_depend>roslib</build_depend>
<exec_depend>roslib</exec_depend>

ros2:
0、安装对应ros包:
sudo apt install ros-foxy-ament-index-cpp
1、cpp:
#include <ament_index_cpp/get_package_share_directory.hpp>
std::string package_share_directory =
ament_index_cpp::get_package_share_directory(package_name.toStdString());
2、CmakeList.txt:
find_package(ament_index_cpp REQUIRED)
ament_target_dependencies括号内增加ament_index_cpp
3、package.xml
<build_depend>ament_index_cpp</build_depend>
<exec_depend>ament_index_cpp</exec_depend>
```

6、注意

```
ROS2 初始化接口会解析部分参数使用，如代码中使用gflags解析参数，需做如下修改:
google::ParseCommandLineFlags(&argc, &argv, true);
===>
//google::ParseCommandLineFlags(&argc, &argv, true);
```

2、Msg迁移

1、文件名命名为大写开头的驼峰命名

2、内部字段为小写+下划线格式

3、生成的头文件名

称为小写+下划线格式

4、内部类名大写开头的驼峰命名

5、消息类使用须增加msg命名空间

ex:

```
文件名称：VcuDetail.msg
.msg内部自动定义：
    std_msgs/Header header

    float32 acc_depth         #加速踏板深度，0-100%
    float32 vehicle_speed     #整车车速，-80到80km/h
头文件使用：#include <truck_msgs/msg/vcu_detail.hpp>
消息类使用：truck_msgs::msg::VcuDetail
```

3、CmakeList.txt迁移

1、cmake版本要求

```
ros1:
cmake_minimum_required(VERSION 3.0.2)
ros2:
cmake_minimum_required(VERSION 3.5)
```

2、find_package

```
ros1:
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
  can_msgs
  truck_msgs
)

ros2:
```

```
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(can_msgs REQUIRED)
find_package(truck_msgs REQUIRED)

如需编译消息，替换find_package中的 message_generation
为 rosidl_default_generators
```

3、catkin_package

```
ros1:
catkin_package(
 INCLUDE_DIRS include
 LIBRARIES vcu_to_ros
 CATKIN_DEPENDS roscpp std_msgs can_msgs truck_msgs
#  DEPENDS system_lib
)

ros2:
ament_export_include_directories(include)
ament_export_libraries(${PROJECT_NAME}_node)
ament_export_dependencies( rclcpp std_msgs can_msgs truck_msgs)
#  ament_export_dependencies(system_lib)
```

4、generate_messages

```
ros1:
add_message_files(
  FILES
  object.msg
  object_list.msg
)
generate_messages(
  DEPENDENCIES
  std_msgs
  truck_msgs
)

ros2:
set(msg_files
  "msg/Object.msg"
  "msg/ObjectList.msg"
)
 rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  DEPENDENCIES std_msgs geometry_msgs
)

```

5、include_directories

```
ros1:
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
)

ros2:
include_directories(include)
```

6、add_dependencies(ROS2仅用于添加ROS包编译依赖)

```
ros1:
add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
ros2:
ament_target_dependencies(${PROJECT_NAME}_node
  rclcpp
  std_msgs
  can_msgs
  truck_msgs
)
```

7、target_link_libraries(ROS2仅用于添加系统第三方库编译依赖)

```
 ros1:
 target_link_libraries(${PROJECT_NAME}_node
  ${catkin_LIBRARIES}
   glog
   gflags
 )

```

```
ros2:
target_link_libraries(${PROJECT_NAME}_node
  glog
  gflags
)
```

8、install

```
ros1:
install(TARGETS ${PROJECT_NAME}_node
  ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
  LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
  RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

## Mark cpp header files for installation
install(DIRECTORY include/${PROJECT_NAME}/
  DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
  FILES_MATCHING PATTERN "*.h" PATTERN "*.hpp"
  PATTERN ".svn" EXCLUDE
)

install(DIRECTORY launch config
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
)

ros2:
install(TARGETS ${PROJECT_NAME}_node
  ARCHIVE DESTINATION lib/
  LIBRARY DESTINATION lib/
  RUNTIME DESTINATION lib/${PROJECT_NAME})

## Mark cpp header files for installation
install(DIRECTORY include/
  DESTINATION include/${PROJECT_NAME}/
  FILES_MATCHING PATTERN "*.h" PATTERN "*.hpp"
  PATTERN ".svn" EXCLUDE
)

install(DIRECTORY launch config
  DESTINATION share/${PROJECT_NAME}/)
```

9、test

```
ros1:
catkin_add_gtest(${PROJECT_NAME}-test test/test_vcu_to_ros.cpp)
if(TARGET ${PROJECT_NAME}-test)
  target_link_libraries(${PROJECT_NAME}-test  ${catkin_LIBRARIES} gtest)
endif()

ros2:
find_package(ament_cmake_gtest REQUIRED)
ament_add_gtest(${PROJECT_NAME}-test src/test/test_vcu_to_ros.cpp)
ament_target_dependencies(${PROJECT_NAME)-test
"rclcpp"
"std_msgs")
target_link_libraries(${PROJECT_NAME}-test gtest)
```

10、其他

```
ros2末尾须添加:
ament_package()

可参考复制vcu_ro_ros模块下的CmakeList.txt
```

4、package.xml迁移

1、编译依赖

```
ros1:
<buildtool_depend>catkin</buildtool_depend>
<build_type>catkin</build_type>
<build_depend>roscpp</build_depend>
如需编译msg，添加:
<build_depend>message_generation</build_depend>

ros2:
<buildtool_depend>ament_cmake</buildtool_depend>
<build_depend>rclcpp</build_depend>
如需编译msg，添加:
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
如需使用launch启动，添加:
```

```
<exec_depend>ros2launch</exec_depend>
如需测试，添加
<test_depend>ament_cmake_gtest</test_depend>
如需lint,添加
<test_depend>ament_lint_auto</test_depend>
<test_depend>ament_lint_common</test_depend>
C++包添加：<export>
  <build_type>ament_cmake</build_type>
</export>
python包添加：
<export>
  <build_type>ament_python</build_type>
</export>
```

5、launch文件迁移

ROS2支持xml,yaml和py文件作为launch，为便于从ROS1迁移，我们这里使用xml：

1、修改原.launch文件为.xml文件

2、标签变化说明：

```
1、type -> exec
2、ns -> namespace
3、rosparam -> param（且仅可放在node节点标签内）
4、file -> from
5、find -> find-pkg-share
6、include 必须放在<group>标签内
```

3、参考示例：

```
ros1:
<launch>
    <include file="$(find lidar_perception)/launch/lidar_perception.launch"/>
    <rosparam file="$(find vcu_to_ros)/config/config.yaml" command="load"/>
    <node pkg="vcu_to_ros" type="vcu_to_ros_node" name="vcu_to_ros_node" output="screen">
    </node>
</launch>
```

```



ros2:
<launch>
     <group>
       <include file="$(find-pkg-share lidar_perception)/launch/lidar_perception.xml" />
   </group>
    <node pkg="vcu_to_ros" exec="vcu_to_ros_node" name="vcu_to_ros_node" output="screen">
        <param from="$(find-pkg-share vcu_to_ros)/config/config.yaml"/>
   </node>
</launch>
```

6、参数文件迁移

ROS2参数有2点大变化：

1、没有全局参数概念，所有节点参数都在节点内加载

2、不支持yaml文件中复杂类型的数组、列表等

3、yaml文件需要增加2个层级标签：

```
ROS1:
mqtt:
  client:
    protocol: 4      # MQTTv311
  connection:
    host: 117.160.210.2
    port: 1883

ROS2:
mqtt_bridge_node:
  ros__parameters:
    mqtt:
      client:
        protocol: 4      # MQTTv311
      connection:
        host: 117.160.210.2
        port: 1883
```

针对变化1，虽然没有全局参数概念，但由于ROS2参数是使ROS srv的形式实现，我们可通过srv客户端读取，ros_server.h中已提供读取其他节点参数的接口getGlobalParam，使用方法和原getParam一致。

针对变化2，目前须拆取复杂类型的数组参数部分到独立yaml文件，并使用yaml库直接读文件获取，或直接整个参数文件使用yaml库读取的形式实现。

针对变化2，为简化yaml库的使用，ROSServer封装了部分接口，ROSServer内 yaml参数使用示例：

```
1、CmakeList.txt修改:
add_definitions(-DSUPPORT_YAML_CONFIG) #使能yaml config支持

target_link_libraries(${PROJECT_NAME}_node
  glog
  gflags
  yaml-cpp #增加yaml依赖
)

2、launch文件修改:
#node节点内添加
<param name="yaml_config" value="$(find-pkg-share error_handle)/config/config.yaml"/>

3、代码参考:
AlgParam param;
auto params = msg_server.get_yaml_config()["alg_params"];
//读取（如不存在对应key，会报异常）
for(size_t i = 0; i<params.size(); ++i)
{
    param.seq_threshold = params[i]["seq_threshold"].as<int>();
    param.duration  = params[i]["duration"].as<double>();
    AlgParams[params[i]["error_code"].as<int>()] = param;
}
//设置(添加)
params[0]["duration"] = 3.0;
params[0]["seq_threshold"] = 50;
//保存
msg_server.save_yaml();
```