Week 7

2부 ROS 2 기본 프로그래밍

1장 ROS 프로그래밍 규칙

ROS 커뮤니티에서는 코드 스타일 가이드를 통해 소스코드 작업시 빈번히 생기는 개발자의 부가적인 선택을 줄여주고, 가독성을 높혀 다른 협업 개발자 및 이용자의 코드 이해를 높히고 있다. 이를 통해 상호간의 코드 리뷰를 보다 쉽게 할 수 있으며, 프로그래밍 언어의 특정 기능으로 인해 생길 수 있는 오류와 다양한 이슈를 피할 수 있다.

기본 이름 규칙

이름 규칙에는 snake_case, CamelCased, ALL_CAPITALS 와 같이 3종류의 네이밍을 기본으로 사용한다.

파일 이름 및 변수명, 함수명에는 모두 소문자로 snake_case 이름 규칙을 사용한다. 가독성을 해지는 축약어는 가능한 사용하지 않으며 확장자명은 모두 소문자로 표기한다. 타입 및 클래스는 CamelCased 이름 규칙을 사용하고 상수는 ALL_CAPITALS 이름 규칙을 사용한다.

snake_case : 이름의 띄어쓰기를 언더바(_)로 표기하는 관례인 네이밍 컨벤션(Naming convention)이다.

ex) snake_case(일반적인 변수 이름), is_snake_case(Boolean 타입)

CamelCased : 이름의 띄어쓰기 없이 단어의 첫 글자를 대문자로 표기하는 네이밍 컨벤션이다.

ex) camelCase(일반적인 변수 이름), isCamelCase(Boolean 타입)

ALL_CAPITALS : 이름의 띄어쓰기를 언더바(_)로 표기하고 단어를 전부 대문자로 표기하는 네이밍 컨벤션이다.

ex) MY_MACRO_THAT_SCARES_SMALL_CHILDREN_AND_ADULTS_ALIKE

단, ROS 인터페이스 파일은 /msg /srv /action 폴더에 위치시키며 인터페이스 파일명은 CamelCased 규칙을 따른다. *.msg, *.srv, *.action 파일은 *.h(pp) 및 모듈로 변환한 후 구조체 및 타입으로 사용되기 때문이다.

C++ style

ROS 에서 사용하는 C++ 코드 스타일은 오픈소스 커뮤니티에서 가장 널리 사용중인 Google C++ Style Guide 를 사용하고 있으며 ROS의 특성에 따라 일부를 수정해 사용한다.

더 자세한 내용은 REPs 및 Google C++ Style Guide 문서를 참고하자.

1. 기본규칙

   C++14 Standard 준수

2. 라인 길이

   최대 100문자

3. 이름 규칙(Naming)

   CamelCased : 타입, 클래스, 구조체, 열거형

   snake_case : 파일, 패키지, 인터페이스, 네임스페이스, 변수, 함수, 메소드

   ALL_CAPITALS : 상수, 매크로

   소스 파일은 cpp 확장자 사용

   헤더파일은 hpp 확장자 사용

전역변수(Global variable)는 접두어(g_)를 붙인다.

클래스 멤버 변수(Class member variable)는 마지막에 언더바(_)를 붙인다.

4. 공백 문자 대 탭(Spaces VS Tabs)

   기본 들여 쓰기(Indent)는 공백 문자(Space) 2개를 사용한다.(Tab 미사용)

   Class의 접근 지정자(public, protected, private)는 들여쓰기 미사용

5. 괄호(Brace)

   if, else, do, while, for 구문에 괄호를 사용한다.

   괄호 및 공백 사용은 책에 예제 참조

6. 주석(Comments)

   문서 주석은 /** */을 사용한다.

   구현 주석은 //을 사용한다.

7. 린터(Linters)

   C++ 코드 스타일의 자동 오류 검출을 위하여 ament_cpplint, ament_uncrustify를 사용하고 정적 코드 분석이 필요한 경우 ament_cppcheck를 사용한다,

8. 기타

   Boost 라이브러리의 사용은 가능한 피한다.

   포인터 구문은 char * c처럼 사용한다.

   중첩 템플릿은 set<list<string>>처럼 사용한다.(<간의 띄어쓰기 미사용)

Python style

ROS 에서 사용하는 Python 코드 스타일은 Python Enhancement Proposals (PEPs) 의 PEP 8을 준수한다.

1. 기본규칙

   파이썬 3(3.5이상) 사용

2. 라인 길이

   최대 100문자

3. 이름 규칙(Naming)

   CamelCased : 타입, 클래스

   snake_case : 파일, 패키지, 인터페이스, 모듈, 변수, 함수, 메소드

   ALL_CAPITALS : 상수

   소스 파일은 cpp 확장자 사용

   헤더파일은 hpp 확장자 사용

   전역변수(Global variable)는 접두어(g_)를 붙인다.

   클래스 멤버 변수(Class member variable)는 마지막에 언더바(_)를 붙인다.

4. 공백 문자 대 탭(Spaces VS Tabs)

   기본 들여 쓰기(Indent)는 공백 문자(Space) 4개를 사용한다.(Tab 미사용)

   Hanging indent(문장 중간에 들여쓰기를 사용하는 형식)의 사용 방법은 다음 예제를 참고하자.

5. 괄호(Brace)

   if, else, do, while, for 구문에 괄호를 사용한다.

괄호 및 공백 사용은 책에 예제 참조

6. 주석(Comments)

   문서 주석은 “”” ”””을 사용하고

   구현 주석은 #을 사용한다.

7. 린터(Linters)

   python 코드 스타일의 자동 오류 검출을 위하여 ament_flake8을 사용한다,

8. 기타

   모든 문자는 큰 따옴표(”)가 아닌 작은 따옴표(’)를 사용하여 표현한다.

2장 ROS 프로그래밍 기초(파이썬)

이번 장에서는 “Hello World” 라는 메시지를 전송하는 파이썬 패키지를 만드는 것에 초점을 둔다.

```
$ cd ~/ros2_ws/src/
$ ros2 pkg create my_first_ros_rclpy_pkg --build-type ament_python --dependencies rclpy std_msgs
```

의존하는 패키지로 rclpy와 std_msgs를 옵션으로 사용했다. 이는 ROS 파이썬 클라이언트 라이브러리(rclpy)와 ROS의 표준 메시지 패키지(std_msgs)를 사용하겠다는 의미이다.

패키지 설정

이제 pakcage.xml, setup.py, setup.cfg 를 작성하여 패키지를 설정하겠다.

패키지 설정 파일(package.xml)

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_first_ros_rclpy_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="42863043+jhc1000@users.noreply.github.com">chan</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

파이썬 패키지 설정 파일(setup.py)

파이썬 패키지 설정 파일에서는 entry_points 옵션의 console_scripts 키를 사용한 실행 파일 설정이다. 예를 들어 helloworld_publisher 콘솔 스크립트는 my_first_ros_rclpy_pkg.helloworld_publisher 모듈의 main 함수를 호출하게 된다. 이 설정을 통해 ros2 run 또는 ros2 launch 명령어로 해당 스크립트를 실행시킬 수 있다.

```
from setuptools import setup

package_name = 'my_first_ros_rclpy_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='chan',
    maintainer_email='42863043+jhc1000@users.noreply.github.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'helloworld_publsiher = my_first_ros_rclpy_pkg.hellowworld_publisher:main',
            'helloworld_subscriber = my_first_ros_rclpy_pkg.hellowworld_subsrciber:main',
        ],
    },
)
```

파이썬 패키지 환경설정 파일(setup.cfg)

setup.cfg 에서는 패키지이름을 기입하고 나중에 colcon build 후에 해당 workspace(ex)ros2_ws) 내부의 지정 폴더에 실행파일이 생성된다.

```
[develop]
script-dir=$base/lib/my_first_ros_rclpy_pkg
[install]
install-scripts=$base/lib/my_first_ros_rclpy_pkg
```

퍼블리셔 노드 작성

퍼블리셔 노드의 파이썬 코드는 ~/ros2_ws/src/my_first_ros_rclpy_pkg/my_first_ros_rclpy_pkg/ 폴더에 helloworld_publisher.py 소스코드 파일을 직접 생성하여 작성한다.

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String

class HelloWorldPublisher(Node):

    def __init__(self):
        super.__init__('helloworld_publsiher')
        qos_profile = QoSProfile(depth=10)
        self.helloworld_publisher = self.create_publisher(String, 'helloworld', qos_profile)
        self.timer = self.create_timer(1, self.publish_helloworld_msg)
        self.count = 0

    def publish_helloworld_msg(self):
```

```
        msg = String()
        msg.data = 'Hello World: {0}'.format(self.count)
        self.helloworld_publsiher.publish(msg)
        self.get_logger().info('Published message: {0}'.format(msg.data))
        self.count += 1

def main(args=None):
    rclpy.init(args=args)

    node = HelloWorldPublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

서브스크라이버 노드 작성

퍼블리셔 노드의 파이썬 코드는 ~/ros2_ws/src/my_first_ros_rclpy_pkg/my_first_ros_rclpy_pkg/ 폴더에 helloworld_subscriber.py 소스코드 파일을 직접 생성하여 작성한다.

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String

class HelloWorldSubscriber(Node):

    def __init__(self):
        super.__init__('helloworld_subscriber')
        qos_profile = QoSProfile(depth=10)
        self.helloworld_subscriber = self.create_subscription(
            String,
            'helloworld',
            self.subscribe_topic_message,
            qos_profile)

    def subscribe_topic_message(self, msg):
        self.get_logger().info('Received message: {0}'.format(msg.data))

def main(args=None):
    rclpy.init(args=args)

    node = HelloWorldSubscriber()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

빌드

colcon 빌드 툴을 사용하여 패키지를 빌드한다.

```
(워크스페이스 내의 모든 패키지 빌드하는 방법)
$ cd ~/ros2_ws && colcon build --symlink-install

(특정 패키지만 빌드하는 방법)
$ cd ~/ros2_ws && colcon build --symlink-install --packages-select [패키지 이름1] ...

(특정 패키지 및 의존성 패키지를 함께 빌드하는 방법)
$ cd ~/ros2_ws && colcon build --symlink-install --packages-up-to [패키지 이름]
```

```
$ cd ~/ros2_ws
$ colcon build --symlink-install --packages-select my_first_ros_rclpy_pkg
```

패키지 빌드 후에는 환경설정 파일을 불러와서 실행 가능한 패키지의 노드 설정을 해줘야 빌드된 노드를 실행할 수 있다.

```
$ . ~/ros2_ws/install/local_setup.bash
```

실행

각 노드 실행은 ros2 run 명령어로 실행한다.

```
$ ros2 run my_first_ros_rclpy_pkg helloworld_subscriber
[INFO] [1692765697.363130949] [helloworld_subscriber]: Received message: Hello World: 0
[INFO] [1692765698.357796673] [helloworld_subscriber]: Received message: Hello World: 1
[INFO] [1692765699.357478002] [helloworld_subscriber]: Received message: Hello World: 2
```

```
$ ros2 run my_first_ros_rclpy_pkg helloworld_publisher
[INFO] [1692765697.362887911] [helloworld_publisher]: Published message: Hello World: 0
[INFO] [1692765698.357100956] [helloworld_publisher]: Published message: Hello World: 1
[INFO] [1692765699.357029072] [helloworld_publisher]: Published message: Hello World: 2
[INFO] [1692765700.357049266] [helloworld_publisher]: Published message: Hello World: 3
```

3장 ROS 프로그래밍 기초 (C++)

이번 장에서는 C++로 간단한 토픽 퍼블리셔, 서브스크라이버 패키지를 구현하는 방법을 알아보겠다.

패키지 생성

```
$ cd ~/ros2_ws/src/
$ ros2 pkg create my_first_ros_rclcpp_pkg --build-type ament_cmake --dependencies rclcpp std_msgs
```

rclcpp과 std_msgs를 의존 패키지로 설정하였다.

패키지 설정

패키지 설정 파일(package.xml) 과 빌드 설정 파일(CMakeLists.txt)을 통해 패키지를 설정한다.

패키지 설정 파일(package.xml)

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_first_ros_rclcpp_pkg</name>
  <version>0.6.0</version>
  <description>ROS 2 rclcpp basic package for the ROS 2 seminar</description>
  <maintainer email="passionvirus@gmail.com">Pyo</maintainer>
  <license>Apache License 2.0</license>
  <author>Mikael Arguedas</author>
  <author>Morgan Quigley</author>
  <author email="jacob@openrobotics.org">Jacob Perron</author>
  <author email="passionvirus@gmail.com">Pyo</author>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

빌드 설정 파일(CMakeLists.txt)

```
# Set minimum required version of cmake, project name and compile options
cmake_minimum_required(VERSION 3.5)
project(my_first_ros_rclcpp_pkg)

if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# Find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

# Build
add_executable(helloworld_publisher src/helloworld_publisher.cpp)
ament_target_dependencies(helloworld_publisher rclcpp std_msgs)

add_executable(helloworld_subscriber src/helloworld_subscriber.cpp)
ament_target_dependencies(helloworld_subscriber rclcpp std_msgs)

# Install
install(TARGETS
  helloworld_publisher
  helloworld_subscriber
  DESTINATION lib/${PROJECT_NAME})

# Test
if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()
```

```
# Macro for ament package
ament_package()
```

퍼블리셔 노드 작성

~/ros2_ws/src/my_first_ros_rclcpp_pkg/src/ 폴더에 helloworld_publisher.cpp 소스 파일을 직접 생성한다.

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;


class HelloworldPublisher : public rclcpp::Node
{
public:
  HelloworldPublisher()
  : Node("helloworld_publisher"), count_(0)
  {
    auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
    helloworld_publisher_ = this->create_publisher<std_msgs::msg::String>(
      "helloworld", qos_profile);
    timer_ = this->create_wall_timer(
      1s, std::bind(&HelloworldPublisher::publish_helloworld_msg, this));
  }

private:
  void publish_helloworld_msg()
  {
    auto msg = std_msgs::msg::String();
    msg.data = "Hello World: " + std::to_string(count_++);
    RCLCPP_INFO(this->get_logger(), "Published message: '%s'", msg.data.c_str());
    helloworld_publisher_->publish(msg);
  }
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr helloworld_publisher_;
  size_t count_;
};


int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<HelloworldPublisher>();
  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

서브스크라이버 노드 작성

마찬가지로 ~/ros2_ws/src/my_first_ros_rclcpp_pkg/src/ 폴더에 helloworld_subscriber.cpp 소스 파일을 직접 생성한다.

```
#include <functional>
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using std::placeholders::_1;
```

```
class HelloworldSubscriber : public rclcpp::Node
{
public:
  HelloworldSubscriber()
  : Node("Helloworld_subscriber")
  {
    auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
    helloworld_subscriber_ = this->create_subscription<std_msgs::msg::String>(
      "helloworld",
      qos_profile,
      std::bind(&HelloworldSubscriber::subscribe_topic_message, this, _1));
  }

private:
  void subscribe_topic_message(const std_msgs::msg::String::SharedPtr msg) const
  {
    RCLCPP_INFO(this->get_logger(), "Received message: '%s'", msg->data.c_str());
  }
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr helloworld_subscriber_;
};


int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<HelloworldSubscriber>();
  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

빌드

파이썬 빌드때와 마찬가지의 방법으로 빌드하고 환경설정을 불러온다.

```
$ cd ~/ros2_ws
$ colcon build --symlink-install --packages-select my_first_ros_rclpy_pkg
```

```
$ . ~/ros2_ws/install/local_setup.bash
```

실행

```
$ ros2 run my_first_ros_rclcpp_pkg helloworld_subscriber
[INFO] [1692765697.363130949] [helloworld_subscriber]: Received message: Hello World: 0
[INFO] [1692765698.357796673] [helloworld_subscriber]: Received message: Hello World: 1
[INFO] [1692765699.357478002] [helloworld_subscriber]: Received message: Hello World: 2
```

```
$ ros2 run my_first_ros_rclcpp_pkg helloworld_publisher
[INFO] [1692765697.362887911] [helloworld_publisher]: Published message: Hello World: 0
[INFO] [1692765698.357100956] [helloworld_publisher]: Published message: Hello World: 1
[INFO] [1692765699.357029072] [helloworld_publisher]: Published message: Hello World: 2
[INFO] [1692765700.357049266] [helloworld_publisher]: Published message: Hello World: 3
```

4장 ROS 2 Tips

설정 스크립트(setup script)

ROS 2 를 사용할 때의 빈번한 실수는 빌드후 설정스크립트를 사용하지 않고 패키지를 찾거나 노드를 실행시키는 것이다. 새로운 패키지를 빌드하였으면 잊지말고 설정 스크립트를 실행시키자. 편의를 위해 ~/.bashrc 에 저장한후 사용하는 것을 추천한다.

```
$ source ~/ros2_ws/install/local_setup.bash
```

setup.bash VS local_setup.bash

두 bash 파일의 차이를 설명하기 앞서서 underlay와 overlay 개념을 알아보자. 우리가 ROS 2를 바이너리 설치(Binary installation)하면 /opt/ros/foxy 설치 폴더에 모든 구성 파일이 위치하게 되고 소스코드를 내려받아 이를 빌드하여 설치(Source installation)하면 ~/ros2_foxy/와 같은 경로를 사용할 것이다. 이와 같은 개발환경을 underlay라고 한다. 그리고 개발자가 사용하는 ~/ros2_ws/ 와 같은 워크 스페이스가 있다. 이러한 개발환경은 overlay라고 한다.

즉 overlay 환경은 설치된 ROS 패키지들에 의존하기에 underlay 개발환경에 종속적이게 된다.

local_setup.bash 스크립트는 이 스크립트가 위치해있는 접두사(Prefix) 경로(~/ros2_ws/install/)의 모든 패키지에 대한 환경을 설정한다.

setup.bash 스크립트는 현재 작업공간이 빌드될 때 환경에 제공된 다른 모든 작업 공간에 대한 local_setup.bash를 포함한다. 즉 underlay 개발환경의 설정 정보도 포함한다.

따라서 일반적으로 underlay 개발환경의 setup.bash를 소싱한 후, 자신의 워크스페이스인 overlay 개발환경의 local_setup.bash를 소싱하는게 정석이다. 따라서 이 둘을 ~/.bashrc 에 넣어두는 것이 좋다.

```
source /opt/ros/foxy/setup.bash
source ~/ros2_ws/install/local_setup.bash
```

colcon_cd

터미널 창에서 colcon_cd 명령을 사용하면 셸의 현재 작업 디렉터리를 패키지 디렉터리로 빠르게 변경할 수 있다.

```
$ colcon_cd [패키지 이름]
```

이를 사용하기 위해서는 다음과 같은 설정을 ./bashrc에 추가하면 된다.

```
source /usr/share/colcon_cd/function/colcon_cd.sh
export _colcon_cd_root=~/ros2_ws
```

ROS_DOMAIN_ID VS Namespace

ROS 2를 사용하면서 동일 네트워크를 다른 사람들과 공유하고 있다면 다른 연구원들이 사용하고 있는 노드 정보에 쉽게 접근이 가능하며 관련된 데이터를 공유할 수 있게 된다. 이런 기능은 멀티 로봇 제어 및 협업 작업시 매우 편리하지만 독립

된 작업을 해야 할때는 역으로 불편하다. 이를 방지하는 방법은 다음과 같다.

1. 물리적으로 다른 네트워크 사용
2. ROS_DOMAIN_ID를 이용하여 DDS의 domain을 변경
3. 각 노드 및 토픽/서비스/액션의 이름에 Namespace 추가

ROS_DOMAIN_ID

ROS 2에서는 DDS의 도입으로 멀티캐스트 방식을 사용하여 전역 공간이라 불리는 DDS Global Space 공간에 있는 토픽들에 대해 퍼블리시 서브스크라이브할 수 있게 된다. 이 DDS Global Space 를 손쉽게 변경하는 방법이 ROS_DOMAIN_ID 환경변수를 설정하는 것이다.

각 터미널 창에서 ROS_DOMAIN_ID를 export 하면 동일하게 맞춘 노드들만 서로 통신하게 된다.

필자도 현재 동일 네트워크 상에서 다른 멤버들과 ROS를 사용하기 때문에 이 방법을 사용하고 있다.

```
$ export ROS_DOMAIN_ID=10
```

Namespace

ROS 2의 노드는 고유 이름을 가진다. 각 노드에서 사용하는 토픽, 서비스, 액션, 파라미터 또한 고유 이름으로 설정된다. 이러한 고유 이름에 Namespace를 붙여주면 독립적으로 자신만의 네트워크를 그룹화 할 수 있다. 이는 각 노드를 실행할 때 ROS 변수중 하나인 ns(namespace)를 입력하여 변경하는 방법이 있고, 런치 파일로 실행시킬 때 node_namespace 항목을 추가하는 방법이 있다.

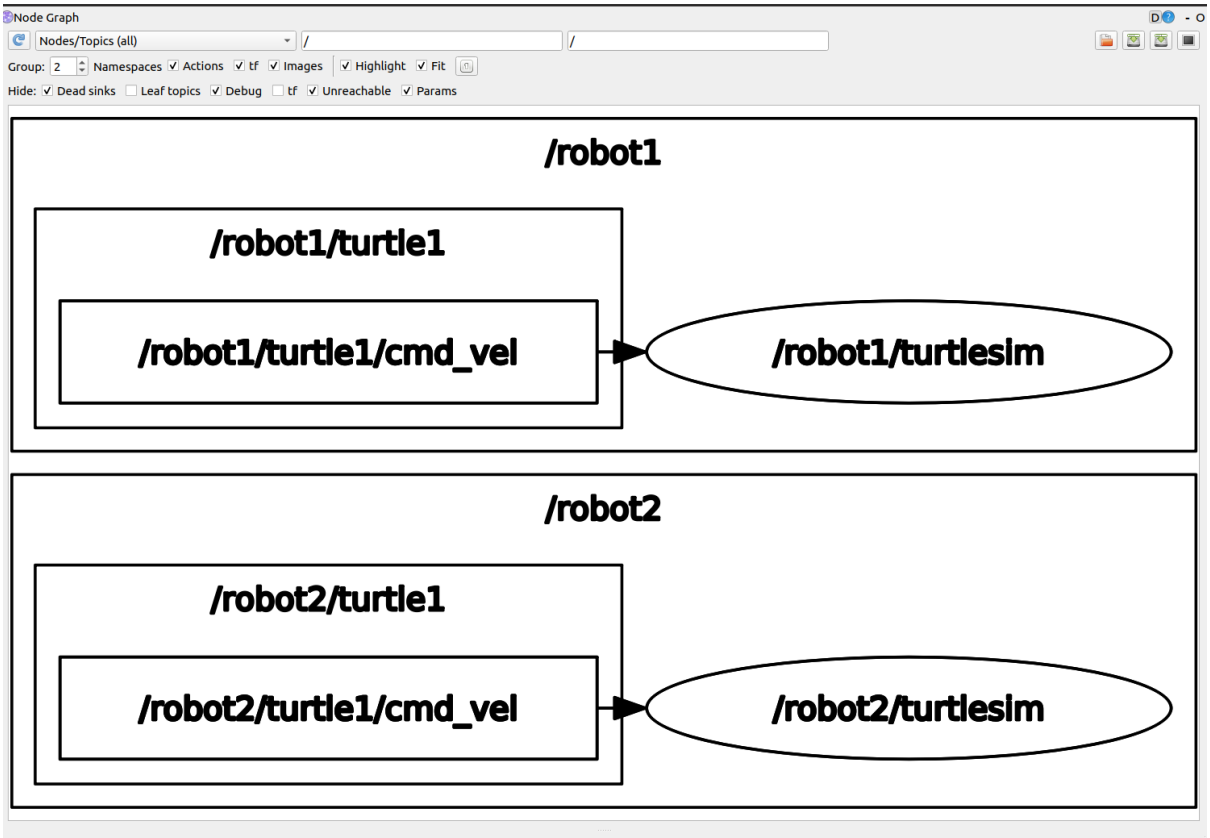
여기서는 노드 실행시 ns를 추가하는 방법을 설명하겠다.

처음에 노드를 실행시 —ros-args 옵션을 이용하여 각 노드에 Namespace를 붙여 주었다. 이후 rqt_graph를 통해 /robot1, /robot2 두 그룹으로 묶여져 있는 모습을 볼 수 있다.

```
$ ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/robot1
```

```
$ ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/robot2
```

```
rqt_graph
```



5장 토픽, 서비스, 액션 인터페이스

ROS 2 인터페이스(Interface) 신규 작성

ROS 2 프로그래밍은 메시지 통신을 위해 기본적인 std_msgs 인터페이스나 속도, 회전 등에 대한geometry_msgs 인터페이스, 센서 값을 담을 수 있는 sensor_msgs 인터페이스를 사용하는 것이 일반적이다. 하지만 사용자가 사용하고자 하는 인터페이스가 없을 경우 새로 만들어 사용해야 한다.

참고로 ROS 2 인터페이스는 이 인터페이스를 사용하려는 패키지에 포함시켜도 되지만 인터페이스로만 구성된 패키지를 별도로 만들어 사용하는 것이 의존성면에서 관리하기 쉽다.

이번 장에서는 msg_srv_action_interface_example 패키지를 만들고 이 인터페이스 전용패키지에 다음과 같은 msg, srv, action 인터페이스를 포함시켜보자.

- ArithmeticArgument.msg
- ArithmeticOperator.srv
- ArithmeticChecker.action

인터페이스 패키지 만들기

```
$ cd ~/ros2_ws/src
$ ros2 pkg create --build-type ament_cmake msg_srv_action_interface_example
$ cd msg_srv_action_interface_example
$ mkdir msg, srv, action
```

ArithmeticArgument.msg 생성

```
# Messages
builtin_interfaces/Time stamp
float32 argument_a
float32 argument_b
```

ArithmeticOperator.srv 생성

```
# Constants
int8 PLUS = 1
int8 MINUS = 2
int8 MULTIPLY = 3
int8 DIVISION = 4

# Request
int8 arithmetic_operator
---
# Response
float32 arithmetic_result
```

ArithmeticChecker.action 생성

```
# Goal
float32 goal_sum
---
# Result
string[] all_formula
float32 total_sum
---
# Feedback
string[] formula
```

패키지 설정 파일(package.xml)

일반적인 패키지와 인터페이스 패키지의 차이점은 빌드 시에 DDS에 사용되는 IDL(Interface Definition Language) 생성과 관련한 rosidl_default_generators가 사용되는 점과 실행 시에 builtin_interfaces와 rosidl_default_runtime이 사용되는 점이다.

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>msg_srv_action_interface_example</name>
  <version>0.6.0</version>
  <description>
    ROS 2 example for message, service and action interface
  </description>
  <maintainer email="passionvirus@gmail.com">Pyo</maintainer>
  <license>Apache 2.0</license>
  <author email="passionvirus@gmail.com">Pyo</author>
  <author email="routiful@gmail.com">Darby Lim</author>

  <buildtool_depend>ament_cmake</buildtool_depend>
```

```
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>builtin_interfaces</exec_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

빌드 설정 파일(CMakeLists.txt)

일반적인 패키지와 다르게 CMake의 set 명령어로 msg, srv, action 파일을 지정하고 rosidl_generate_interfaces에 해당 set들을 기입하면 된다.

```
cmake_minimum_required(VERSION 3.5)
project(msg_srv_action_interface_example)

if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Wpedantic")
endif()

find_package(ament_cmake REQUIRED)
find_package(builtin_interfaces REQUIRED)
find_package(rosidl_default_generators REQUIRED)

set(msg_files
  "msg/ArithmeticArgument.msg"
)

set(srv_files
  "srv/ArithmeticOperator.srv"
)

set(action_files
  "action/ArithmeticChecker.action"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)

ament_export_dependencies(rosidl_default_runtime)
ament_package()
```

빌드

```
$ cd ~/ros2_ws
$ colcon build --symlink-install --packages-select msg_srv_action_interface_example
```