Week7

태그	
내용	

1장 ROS 프로그래밍 규칙 (코드 스타일)

로봇 운영체제 ROS는 커뮤니티의 공동 협업 결과물입니다. 따라서, 일관된 코드 스타일을 따릅니다.

기본 이름 규칙

- 파일, 변수, 함수명은 모두 소문자로 `snake_case` 를 사용합니다. 가독성을 해치는 축약어는 가급적 사용하지 않습니다. 확장자는 모두 소문자로 표기합니다.
- 타입 및 클래스는 `CamelCased` 규칙을 사용합니다.
- 상수는 `ALL_CAPTIALS` 이름 규칙을 사용합니다.
- ROS 인터페이스 파일은 msg, src, action 디렉토리에 위치합니다. 인터페이스 파일 명은 `CamelCased` 을 따릅니다. 그 이유는 *.msg, *.srv, *.action 파일은 *.h(pp) 모듈로 변환한 후 구조체 타입으로 사용되기 때문입니다.

C++ Style

ROS2 C++ 코드 스타일은 오픈소스 커뮤니티에서 널리 사용 중인 Google C++ Style Guide 를 사용합니다.

1. 기본 규칙은 C++14 Standard 를 준수
2. 라인의 길이는 최대 100문자
3. 소스파일은 cpp, 헤더파일은 hpp 확장자를 사용
4. 전역변수는 접두어 `g_` 사용
5. 클래스 변수는 마지막에 밑줄(_) 사용
6. 들여쓰기는 공백문자(space) 2개를 사용
7. Class 의 접근 지정자 (public, protected, private) 는 들여쓰기 않함
8. if, else, do, while, for 구문에 괄호 사용

다음 코드의 예시는 올바른 사용법 입니다.

```
int main(int argc, char **argv)
{
  if (condition) {
    return 0;
  } else {
    return 1;
  }
}

if (this && that || both) {
  ...
}

// 긴 조건식일 경우 소괄호, 중괄호 사용법
if (
  this && that || both && this && that || both && this && that || both && this && that)
{
  ...
}

// 짧은 함수의 호출
call_func(foo, bar);

// 긴 함수의 호출
call_func(
  foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar,
  foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar);

// 파라미터의 가독성을 위해 개행
call_func(
  bang,
  fooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo,
  bar, bat);

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
  Type par_name1,  // 2 space indent
  Type par_name2,
  Type par_name3)
{
  DoSomething();  // 2 space indent
  ...
}

MyClass::MyClass(int var)
: some_var_(var),
  some_other_var_(var + 1)
{
  ...
  DoSomething();
  ...
}
```

9. 주석은 /** */ 형식으로, 구현 주석은 // 를 사용
10. C++ 코드 스타일의 자동 오류 검출을 위해 ament_cpplint, ament_uncrustify 를 사용하고 정적분석이 필요한 경우 ament_cppcheck 를 사용

```
$ which ament_cpplint
/opt/ros/humble/bin/ament_cpplint
```

Python Style

ROS2 Python 코드 스타일은 Python Enhancement Proposals (PEPs) 중 PEP8을 준수합니다.

1. Python3 (3.5+) 를 사용
2. 라인의 길이는 최대 100문자
3. 이름규칙
    a. `CamelCased` : 타입, 클래스
    b. `snake_case` : 파일, 패키지, 인터페이스, 모듈, 변수, 함수, 메소드
    c. `ALL_CAPTITALS` : 상수
4. 공백문자 대 탭 (Spaces vs. Tabs)
    a. 들여쓰기는 공백문자(space) 4개를 사용
    b. Hanging indent 의 사용 방법은 아래 예제코드를 참고
    c. 괄호 및 공백 사용은 다음 예제를 참고

```
foo = function_name(var_one, var_two, var_three, var_four)

def long_long_long_long_function_name(
        var_one,
        var_two,
        var_three,
        var_four):
    print(var_one)
```

5. 괄호는 계산식 및 배열 인덱스로 사용

```
list = [1, 2, 3, 4, 5]
dictionary = {'age': 30, 'name': '홍길동'}
tupple = (1, 2, 3, 4, 5)
```

6. Python 코드 스타일의 자동 오류 검출을 위해 ament_flake8를 사용할 수 있습니다.

2장. ROS 프로그래밍 기초 (Python)

Python 버전의 Hello world 프로그램을 작성하고 실행합니다.

패키지 생성

ros2 pkg 명령어를 사용하여 패키지 구조를 생성할 수 있습니다. build-type은 ament_python 이며, dependencies를 통해 rclpy와 std_msgs를 추가합니다.

```
$ ros2 pkg create hello_topic_rclpy_pkg --build-type ament_python --depende
ncies rclpy std_msgs
```

ros2 pkg 명령어를 사용하여 패키지 구조를 생성한 결과입니다.

```
.
├── hello_topic_rclpy_pkg
│   └── __init__.py
├── package.xml
├── resource
│   └── hello_topic_rclpy_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

패키지 설정방법

setup.py 파일의 entry_points 옵션의 console_scripts 에 새로 만들 pub 와 sub 진입점을 추가합니다.

```
from setuptools import setup

package_name = 'hello_topic_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='makepluscode',
    maintainer_email='makepluscode@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
```

```
    entry_points={
        'console_scripts': [
            'hello_pub = hello_topic_rclpy_pkg.hello_pub:main',
            'hello_sub = hello_topic_rclpy_pkg.hello_sub:main'
        ],
    },
)
```

패키지 코드 작성

아래와 같이 hello_pub.py 와 hello_sub.py 파일을 생성하고 코드를 작성합니다.

```
.
└── src
    └── hello_topic_rclpy_pkg
        ├── hello_topic_rclpy_pkg
        │   ├── __init__.py
        │   ├── hello_pub.py
        │   └── hello_sub.py
        ├── package.xml
        ├── resource
        │   └── hello_topic_rclpy_pkg
        ├── setup.cfg
        ├── setup.py
        └── test
            ├── test_copyright.py
            ├── test_flake8.py
            └── test_pep257.py
```

hello_pub.py

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


class HelloPub(Node):
    def __init__(self):
        super().__init__('hello_pub')
        qos_profile = QoSProfile(depth=10)
        self.hello_pub = self.create_publisher(String, 'hello', qos_profile)
        self.timer = self.create_timer(1, self.publish_hello_msg)
        self.count = 0

    def publish_hello_msg(self):
        msg = String()
        msg.data = "Hello: {0}".format(self.count)
        self.hello_pub.publish(msg)
        self.get_logger().info('Published message: {0}'.format(msg.data))
        self.count += 1


def main(args=None):
```

```
    rclpy.init(args=args)
    node = HelloPub()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('KeyboardInterrupt')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

hello_sub.py

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


class HelloSub(Node):
    def __init__(self):
        super().__init__('hello_sub')
        qos_profile = QoSProfile(depth=10)
        self.hello_sub = self.create_subscription(
            String,
            'hello',
            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 = HelloSub()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('KeyboardInterrupt')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

패키지 실행방법

다음과 같이 패키지를 빌드합니다.

```
$ colcon build --symlink-install --packages-select hello_topic_rclpy_pkg
```

다음과 같이 패키지를 실행합니다.

```
$ ros2 run hello_topic_rclpy_pkg hello_pub
```

```
$ ros2 run hello_topic_rclpy_pkg hello_sub
```

3장. ROS 프로그래밍 기초 (C++)

C++ 버전의 Hello world 프로그램을 작성하고 실행합니다.

패키지 생성

ros2 pkg 명령어를 사용하여 패키지 구조를 생성할 수 있습니다. build-type은 ament_cmake 이며, dependencies를 통해 rclcpp와 std_msgs를 추가합니다.

패키지 설정방법

CMakelist.txt 를 다음과 같이 수정합니다.

```
cmake_minimum_required(VERSION 3.8)
project(hello_topic_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 excutables
add_executable(hello_pub src/hello_pub.cpp)
ament_target_dependencies(hello_pub rclcpp std_msgs)

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

# install excutables
install(TARGETS
```

```
  hello_pub
  hello_sub
  DESTINATION lib/${PROJECT_NAME}
)

ament_package()
```

패키지 코드 작성

아래와 같이 hello_pub.cpp 와 hello_sub.cpp 파일을 생성하고 코드를 작성합니다.

```
.
├── CMakeLists.txt
├── include
│   └── hello_topic_rclcpp_pkg
├── package.xml
└── src
    ├── hello_pub.cpp
    └── hello_sub.cpp
```

hello_pub.cpp

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

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

using namespace std::chrono_literals;

class HelloPub : public rclcpp::Node
{
public:
    HelloPub() : Node("hello_pub"), count_(0)
    {
        auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
        hello_pub_ = this->create_publisher<std_msgs::msg::String>("hello", qos_profile);
        timer_ = this->create_wall_timer(1s, std::bind(&HelloPub::publish_hello_msg, this));
    }

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

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

hello_sub.cpp

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

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

using std::placeholders::_1;

class HelloSub : public rclcpp::Node
{
public:
    HelloSub() : Node("hello_sub")
    {
        auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
        hello_sub_ = this->create_subscription<std_msgs::msg::String>(
            "hello",
            qos_profile,
            std::bind(&HelloSub::subscribe_hello_msg, this, _1)
        );
    }

private:
    void subscribe_hello_msg(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 hello_sub_;
};

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

패키지 실행방법

다음과 같이 패키지를 빌드합니다.

```
$ colcon build --symlink-install --packages-select hello_topic_rclcpp_pkg
```

다음과 같이 패키지를 실행합니다.

```
$ ros2 run hello_topic_rclcpp_pkg hello_pub
```

```
$ ros2 run hello_topic_rclcpp_pkg hello_sub
```

4장 ROS2 Tips

다음 설정 스크립트 실행 구문을 .bashrc 에 넣는 것을 추천합니다.

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

underlay 와 overlay 의 개념

1. underlay 란 ROS2 프레임워크를 소스에서 빌드하여 ~/ros2_humble 과 같은 경로를 사용하는 개발환경을 의미합니다.
2. overlay 란 개발자가 임의의 디렉토리를 사용하여 ~/ros2_ws 와 같은 경로는 사용하는 개발환경을 의미 합니다.

setup.bash vs local_setup.bash

1. setup.bash 와 local_setup.bash 는 설정 스크립트라고 부릅니다. underlay 와 overlay 관계없이 모든 워크스페이스에 존재 합니다.
2. setup.bash 는 모든 작업 공간에 대한 local_setup.bash 를 포함합니다. 즉, underlay 개발환경의 설정 정보까지 포함합니다.
3. local_setup.bash 는 이 스크립트가 위치해 있는 접두사(prefix) 경로의 모든 패키지에 대한 환경을 설정합니다.
4. 위크스페이스가 하나면 차이가 없으나, 둘 이상이면 목적에 맞게 사용되어야 합니다.

ROS_DOMAIN_ID vs Namespace

ROS_DOMAIN_ID를 통해 ROS2 RMW의 네트워크를 구분하여 사용할 수 있습니다.