5.ROS2 function package

1. Introduction to function package

Each robot may have many functions, such as movement control, visual perception, autonomous navigation, etc. It is of course possible if we put the source code of these functions together.But when we want to share some of these functions with others, we will find that the codes are all mixed together and it is difficult to separate them.

This is the principle of function packages. We divide the codes of different functions into different function packages to try to reduce the coupling relationship between them.When you need to share it with others in the ROS community, you only need to explain how to use this function package, and others can use it quickly.

Therefore, the function package mechanism is one of the important methods to improve the software reuse rate in ROS.

2. Create function package

How to create a function package in ROS2? We can use this command:

```
ros2 pkg create <package_name> --build-type <build-type> --dependencies
<dependencies> --node-name <node-name>
```

In the ros2 command:

- **pkg**: Indicates functions related to the function package;
- **create**: indicates creating a function package;
- **package_name**: the name of the new function package;
- **build-type**: Indicates whether the newly created function package is C++ or Python. If C++ or C is used, then it will be followed by ament_cmake. If Python is used, it will be followed by ament_python;
- **dependencies**: Indicates the dependencies of the function package. The C++ function package must include rclcpp, the Python function package must include rclpy, and other required dependencies;
- **node-name**: The name of the executable program, the corresponding source file and configuration file will be automatically generated;

For example, create C++ and Python versions of function packages in the terminal:

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws/src
# Create C++ function package
ros2 pkg create pkg_helloworld_cpp --build-type ament_cmake --dependencies
rclcpp --node-name helloworld
# Create Python feature package
ros2 pkg create pkg_helloworld_py --build-type ament_python --dependencies rclpy
--node-name helloworld
```

3. Compile function package

In the created function package, we can continue to write the code. Afterwards, we need to compile and configure environment variables to run normally:

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
# Compile all function packages in the workspace
colcon build
# Compile the specified function package (one or more)
colcon build --packages-select Function pack list
source install/setup.bash
```

4. Complete workspace structure with function package

The directory structure of the ROS2 workspace is as follows:

```
WorkSpace --- Customized workspace.
     |--- build: The directory where intermediate files are stored. A separate
subdirectory will be created for each function package in this directory.
     |--- install: Installation directory, a separate subdirectory will be
created for each function package in this directory.
     |--- log: Log directory, used to store log files.
     |--- src: Directory used to store function package source code.
         |-- C++ function package
             |-- package.xml: package information, such as: package name,
version, author, dependencies.
             |-- CMakeLists.txt: Configure compilation rules, such as source
files, dependencies, and target files.
             |-- src: C++ source file directory.
             |-- include: header file directory.
             |-- msg: message interface file directory.
             |-- srv: Service interface file directory.
             |-- action: action interface file directory.
         |-- Python function package
             |-- package.xml: package information, such as: package name,
version, author, dependencies.
             |-- setup.py: similar to CMakeLists.txt of C++ function package.
             |-- setup.cfg: Function package basic configuration file.
             |-- resource: resource directory.
             |-- test: stores test-related files.
             |-- Directory with the same name of the function package: Python
source file directory.
```

In addition, whether it is a Python function package or a C++ function package, you can customize some directories related to configuration files.

```
|-- C++ or Python function package
     |-- launch: stores launch files.
     |-- rviz: stores rviz2 configuration related files.
     |-- urdf: stores robot modeling files.
     |-- params: storage parameter file.
     |-- world: stores files related to the simulation environment.
     |-- map: stores map files required for navigation.
    |-- ......
```

The above directories can also be defined with other names, or other directories can be created as needed.