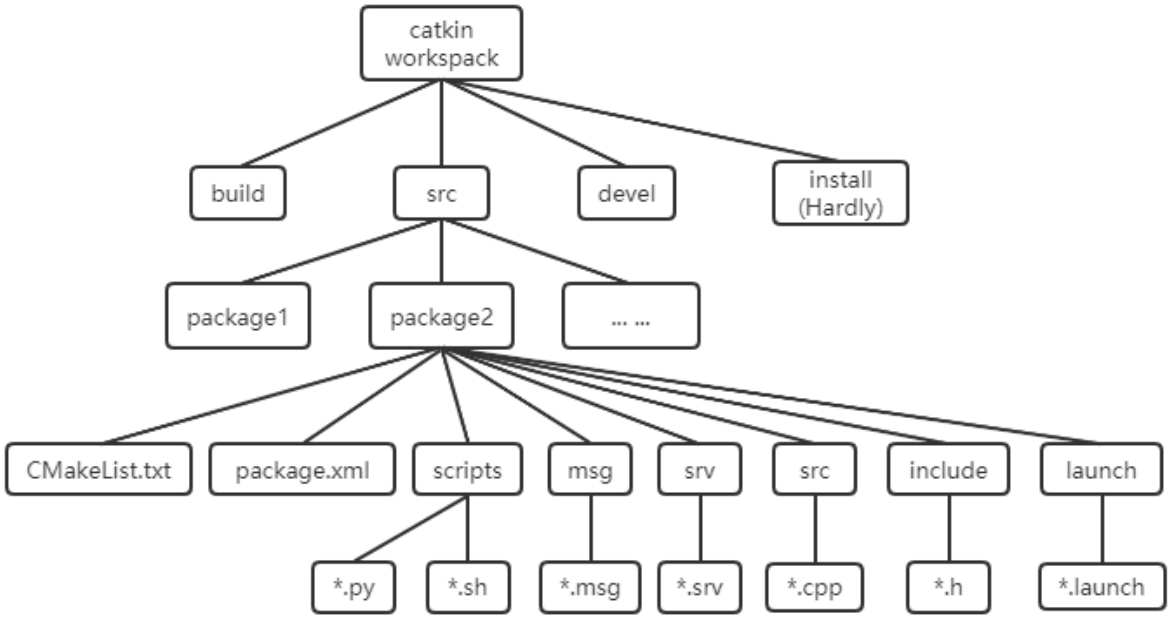
2.Project File Structure

2.1 Project File Structure

The file structure of ROS is not mandatory for every folder and is designed according to business needs.



2.2 Workplace

A workspace is a place to manage and organize ROS engineering project files, intuitively described as a warehouse that contains various ROS project projects, making it easy for the system to organize, manage, and call. In the visual graphical interface, it is a folder. The ROS code we write ourselves is usually stored in the workspace. There are four main primary directories under it:

- src：Source space; ROS Catkin software package (source code package)
- build：Compilation space; Cache information and intermediate files for catkin (CMake)
- devel：Development space; Generated target files (including header files, dynamic link libraries, static link libraries, executable files, etc.), environment variables
- install：Installation Space

The top-level workspace (can be named arbitrarily) and src (must be src) folders need to be created by oneself;

- The build and dev folders are created by Catkin_ The make command automatically creates;
- The install folder is created by Catkin_ The make install command is automatically created and is rarely used, usually not created.

Note: Using catkin_ Make sure to go back to the top-level workspace before compiling. In the same workspace, feature packages with the same name are not allowed to exist; Under different workspaces, feature packages with the same name are allowed to exist.

```
mkdir -p ~/catkin_ws/src      # create
cd catkin_ws/                 # enter the workspace
catkin_make                   # compile
source devel/setup.bash       # update the workspace environment
```

2.3. Package Function Pack

A package is a specific combination of file structures and folders. Usually, program code that implements the same specific function is placed in a package. Only CMakeLists.txt and package.xml are required, and the remaining paths are determined based on whether the software package is needed.Create Feature Pack

```
cd ~/catkin_ws/src
catkin_create_pkg my_pkg rospy rosmsg roscpp
```

ROSPY, ROSMSG, and ROSCPP are dependency libraries that can be added according to business needs or other requirements. Adding them during creation eliminates the need for further configuration, and forgetting to add them requires self configuration.file structure

```
|-- CMakeLists.txt     #
|-- package.xml        #
|-- include folder
|-- config folder      #
|-- launch folder      #
|-- meshes folder      #
|-- urdf folder        #
|-- rviz folder        # rviz file
|-- src folder         # c++ source code
|-- scripts folder     #
|-- srv folder         # custom service
|-- msg folder         # custom topic
|-- action folder      # custom action
```

2.4 Introduction to CMakeLists.txt

2.4.1 Overview

CMakeLists.txt was originally a rule file for the Cmake compilation system, while the Catkin compilation system basically follows the CMake compilation style, only adding some macro definitions for the ROS project. So in terms of writing, Catkin's CMakeLists.txt is basically the same as CMake's.This file directly specifies which packages this package depends on, which targets to compile and generate, how to compile, and other processes. So CMakeLists.txt is very important as it specifies the rules from the source code to the target file. When the Catkin compilation system works, it first finds the CMakeLists.txt under each package, and then compiles and builds according to the rules.

2.4.2 Format

The basic syntax of CMakeLists.txt still follows CMake, and Catkin has added a few macros to it. The overall structure is as follows

```
cmake_minimum_required()    # Required CMake version
project()                   #
find_package()             # Find other CMake/Catkin packages required for
compilation
catkin_python_setup()       #
add_message_files()         # message generator
add_service_files()         # service generator
add_action_files()         # action generator
generate_message()       #
catkin_package()
add_library()            #
add_executable()         #
add_dependencies()       #
target_link_libraries()  #
catkin_add_gtest()       #
install()
```

2.4.3、Boost

If using C++and Boost, you need to call find on Boost_ Package() and specify which aspects of Boost to use as a component. For example, if you want to use Boost threads, you would say:

```
find_package (Boost REQUIRED COMPONENTS thread)
```

2.4.4、catkin_package（）

catkin_ Package() is a CMake macro provided by Catkin. This is necessary to specify catkin specific information for building the system, which in turn is used to generate pkg config and CMake files.

Using add_ Library() or add_ Before executable() can declare any target, * * this function must be called. This function has 5 optional * * parameters:

- `INCLUDE_DIRS` - The export of the package includes the path
- `LIBRARIES` -Library exported from project
- `CATKIN_DEPENDS` - -Other catkin projects that this project relies on
- `DEPENDS` -The non catkin CMake project that this project relies on. For a better understanding, please take a look at this explanation.
- `CFG_EXTRAS` - Other configuration options

The complete macro documentation can be found here.for instance:

```
catkin_package(
   INCLUDE_DIRS include
   LIBRARIES ${PROJECT_NAME}
   CATKIN_DEPENDS roscpp nodelet
   DEPENDS eigen opencv)
```

This indicates that the folder 'include' in the package folder is the location for exporting header files. The CMake environment variable ${PROJECT NAME} evaluates anything passed to the project() function before it is evaluated, in which case it will be "robot brain". 'roscpp'+'nodelet' are software packages that need to exist to build/run this package, while 'eigen'+'opencv' are system dependencies that need to exist to build/run this package.

2.4.5. Including paths and library paths

Before specifying the target, you need to specify the location where resources can be found, especially header files and libraries:

```
 include_directories (<dir1>, <dir2>, ..., <dirN>)
- link_directories (<dir1>, <dir2>, ..., <dirN>)
```

- include_directories ()

include_ The parameter for directories should be find_ Package calls and any other directories generated that need to be included *_ INCLUDE_ DIRS variable. If you use Catkin and Boost, your include_ The directories() call should look like this:

```
include_directories(include ${Boost_INCLUDE_DIRS} ${catkin_INCLUDE_DIRS})
```

The first parameter 'include' indicates that the include/directory in the package is also part of the path.

- link_directories ()

```
link_directories(~/my_libs)
```

CMake link_ The directories() function can be used to add additional library paths, but it is not recommended to do so. All catkin and CMake software packages in find_ When packaged, link information will be automatically added, just link to the target_ link_ Libraries in libraries().Please refer to the threads in this cmake for detailed examples of target usage_ link_ Libraries() in link_ directories () 。

2.4.6 Executable Objectives

To specify executable targets that must be built, we must use add_ The executable() CMake function.

```
add_executable(myProgram src/main.cpp src/some_file.cpp src/another_file.cpp)
```

This will build a target executable file called myProgram, which is built from three source files: src/main.cpp, src/some_ File.cpp and src/another_ file.cpp。

2.4.7 Library files

The add_ The function of library() CMake is to specify libraries to build. By default, catkin builds shared libraries.

```
add_library ($ {PROJECT_NAME} $ {$ {PROJECT_NAME} _SRCS})
```

2.4.8、target_link_libraries

Using target_ link_ The libraries() function specifies the library that can execute the target link. This is usually done in add_ Completed after executable() call. If ros cannot be found, add ${catkin_LIBRARIES}.Syntactic:

```
target_link_libraries (<executableTargetName>, <lib1>, <lib2>, ... <libN>)
```

exam

```
add_executable(foo src/foo.cpp)
add_library(moo src/moo.cpp)
target_link_libraries(foo moo)  -- This links foo against libmoo.so
```

Please note that in most use cases, links are not required_ Directories() because the information is found through_ Package() is automatically introduced.

2.5. Introduction to package.xml

- overview

The package manifest is an XML file named package.xml that must include the root folder of any compatible packages. Package.xml is also a mandatory package file for Catkin, which is the description file for this software package. In earlier ROS versions (ROSBuild compilation system), this file was called manifest.xml, which is used to describe the basic information of the package. If you see some ROS projects online that include manifest.xml, then it is likely a project before the hydro version. Package.xml contains information such as the name, version number, content description, maintenance personnel, software license, compilation and construction tools, compilation dependencies, and runtime dependencies of the package.

Package.xml file build_ The dependent must contain a message_ generation，run_ The dependent must contain a message_ runtime。

- format

```
<pacakge>
<name>
<version>
<description>
<maintainer>
<license>
<buildtool_depend>
<depend>
<build_depend>
<build_export_depend>
<exec_depend>
<test_depend>
<doc_depend>
```

- Dependencies

The package manifest with the minimum label does not specify any dependencies on other packages. Software packages can have six types of dependencies:Build Dependencies<build_ Specify the package required to build this package. This is only the case when any files in these software packages are needed during construction. This can include header files at compile time, library files linked to these packages, or any other resources required at build time (especially when these packages are find_package() in CMake). In cross compilation scenarios, build dependency relationships targeting the target architecture.

Build export dependencies<build_ export_ Specify the package required to construct the library based on this package. This is the case when you include this header file in a common header file in this package (especially when the catkin_package() in CMake is declared as (CATKIN_DEPENDS).Execution dependency<exec_ Specify the software package required to run the code in this package. This is the case when you rely on shared libraries in this package (especially when catkin_package() in CMake is declared as (CATKIN_DEPENDS)).

Test dependencies<test_ Only specify additional dependencies for unit tests. They should not duplicate any dependencies already mentioned as building or running dependencies.Build tool dependencies<buildtool_ Specify that this software package needs to build its own build system tools. Usually, the only build tool is catkin. In cross compilation scenarios, building tool dependencies is used to execute the compiled architecture.Document tool dependencies<doc_ Specify the documentation tool required to generate documents for this software package.

- Additional labels

`<url>` - The URL for information about this software package is usually a wiki page on ros.org.

`<author>` - Author of the package

```
  <url type="website">http://www.ros.org/wiki/turtlesim</url>
  <author>Yahboom</author>
```