

如何使用.h 头文件 .cpp 源文件的使用

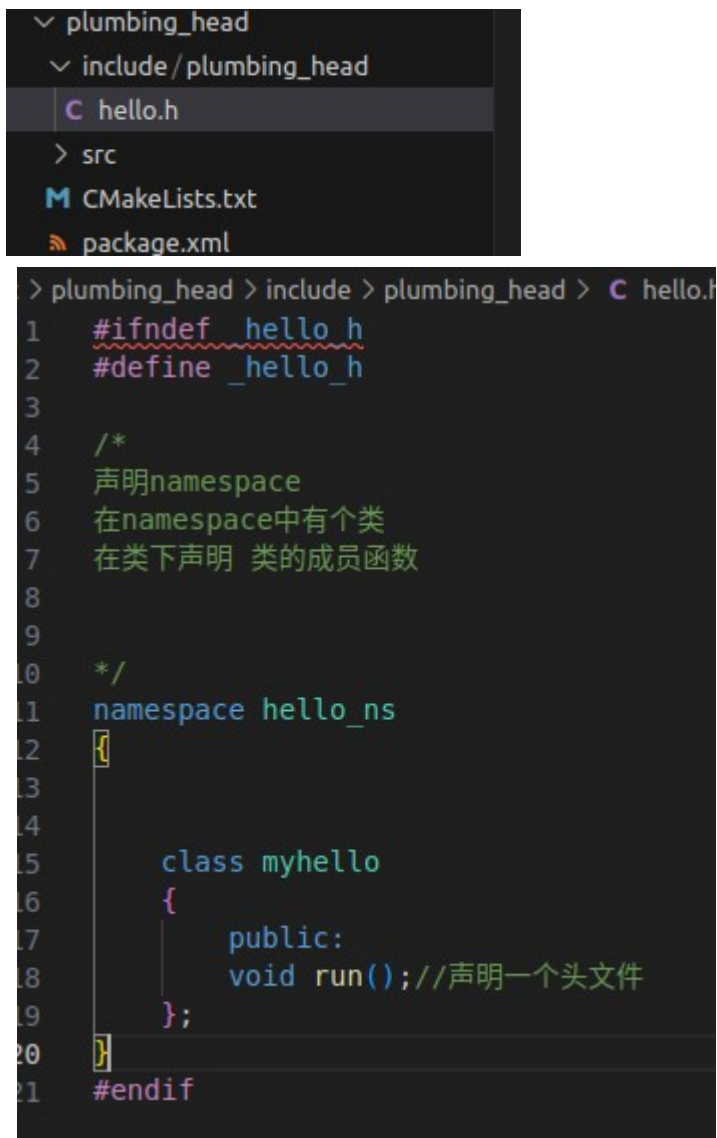
如何在 CMakeLists.txt 的配置

这个我会了一部分但是对于配置的文件只是；了解了一部分

- 1、编写头文件
- 2、编写执行文件
- 3、配置 CMakeLists.txt

1、编写头文件

在**功能包文件夹**下的 **include** 文件夹下 有个与**功能包**同名的文件夹，在这个文件夹下创建.h 的头文件
头文件的位置和实现



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure: `plumbing_head` (expanded) contains `include/plumbing_head` (expanded), which contains `hello.h` (selected). Below `include/plumbing_head` are `src`, `CMakeLists.txt`, and `package.xml`. The code editor shows the content of `hello.h` with line numbers 1 through 21. The code is as follows:

```
1  #ifndef hello_h
2  #define _hello_h
3
4  /*
5   声明namespace
6   在namespace中有个类
7   在类下声明 类的成员函数
8
9
10 */
11 namespace hello_ns
12 {
13
14
15     class myhello
16     {
17     public:
18         void run(); //声明一个头文件
19     };
20 }
21 #endif
```

2、编写 cpp 文件

这里是在 cpp 文件中，同时作为 main 文件执行程序

首先是看这个 cpp 文件的位置是在 src

这是我们的 cpp 文件

首先要办含头文件 include 后面的

```
rc > plumbing_head > src > hello.cpp > main(int, char * [])
1  #include "ros/ros.h"
2  #include "plumbing_head/hello.h" //include 下哪个文件家的 头文件
3  namespace hello_ns
4  {
5
6  void myhello::run()
7  {
8      ROS_INFO("run函数执行");
9  }
10 }
11
12 int main(int argc, char *argv[])
13 {
14     setlocale(LC_ALL, "");
15     ros::init(argc, argv, "hellomyclass");
16     hello_ns::myhello obj;
17     obj.run();
18
19     return 0;
20 }
```

功能包下的.h 文件在 cpp.json 包含这个头文件

```
"includePath": [
    "${workspaceFolder}/**",
    "/home/qinghuan/env_cv/demo04_ws/devel/include/**",
    "/home/qinghuan/env_cv/demo04_ws/devel/include/plumbing_server_client",
    "/home/qinghuan/env_cv/demo04_ws/src/helloclass/include/helloclass",
    "/home/qinghuan/env_cv/demo04_ws/src/hello_class2d/include/**",
    "/home/qinghuan/env_cv/demo04_ws/src/plumbing_head/include/**"
```

在 cmake.txt 的配置

1、是打开头文件 我们的 package 如果在其他文件夹 我们需要把地址放进去 我们的.h 头文件是在 include 下的 把这个地址给过去就 ok 了

```
6  ## Specify additional locations of header files
7  ## Your package locations should be listed before other locations
8  include_directories(
9      include
10     ${catkin_INCLUDE_DIRS}
11 )
12
```

2、第2步是 给 cpp 文件起一个别名，这个一般就算.cpp 前面的文件名 用于在 rosrun 的时候用 rosrun 功能包名 文件的别名

add_dependencies 是添加依赖，说这个 hello 这个文件 在编译前，先把需要的依赖的给编译了
target_link_libraries 是把别名和编译空间联系起来

```
36 add_executable(hello src/hello.cpp)
37
38 ## Rename C++ executable without prefix
39 ## The above recommended prefix causes long target names, the following renames the
40 ## target back to the shorter version for ease of user use
41 ## e.g. "roslaunch someones_pkg node" instead of "roslaunch someones_pkg someones_pkg_node"
42 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
43 |
44 ## Add cmake target dependencies of the executable
45 ## same as for the library above
46 add_dependencies(hello ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
47
48 ## Specify libraries to link a library or executable target against
49 target_link_libraries(hello
50 |   ${catkin_LIBRARIES}
51 )
52
```

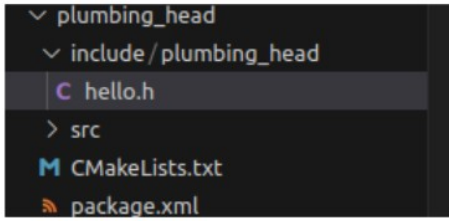
自定义源文件

- 1、编写头文件
- 2、编写源文件
- 3、编写执行文件
- 4、编写 CMakeLists.txt

1、头文件

1、编写头文件

在功能包文件夹下的 **include** 文件夹下有个与功能包同名的文件夹，在这个文件夹下创建.h 的头文件
头文件的位置和实现





```
> plumbing_head > include > plumbing_head > C hello.h
1  #ifndef hello_h
2  #define _hello_h
3
4  /*
5   声明namespace
6   在namespace中有个类
7   在类下声明 类的成员函数
8
9
10 */
11 namespace hello_ns
12 {
13
14     class myhello
15     {
16     public:
17         void run();//声明一个头文件
18     };
19
20 }
21 #endif
```

2、编写源文件

同样需要在 CMakeLists.txt 配置下 我们的头文件路径

```
},
"includePath": [
    "${workspaceFolder}/**",
    "/home/qinghuan/env_cv/demo04_ws/devel/include/**",
    "/home/qinghuan/env_cv/demo04_ws/devel/include/plumbing_server_client",
    "/home/qinghuan/env_cv/demo04_ws/src/helloclass/include/helloclass",
    "/home/qinghuan/env_cv/demo04_ws/src/hello_class2d/include/**",
    "/home/qinghuan/env_cv/demo04_ws/src/plumbing_head/include/**",
    "/home/qinghuan/env_cv/demo04_ws/src/plumbing_head_src/include/**"
],
```

这里我们是使用 ros 的 api 了 需要包含 ros 库

```
src > plumbing_head_src > src >  hello.cpp > {} hello_ns >  run()
1  #include "ros/ros.h"
2  #include "plumbing_head_src/hello.h"
3
4  namespace hello_ns
5  {
6
7  void myhello::run()
8  {
9      ROS_INFO("源文件的成员函数执行");//使用ros的文件需要把ros包含在内
10 }
11
12 }
```

在主函数文件中实现

```
> plumbing_head_src > src >  use_hello.cpp >  main(int, char *[])
1  #include "ros/ros.h"
2  #include "plumbing_head_src/hello.h"
3
4  int main(int argc, char* argv[])
5  {
6
7      setlocale(LC_ALL, "");
8      ros::init(argc, argv, "helloheadsrc");
9      hello_ns::myhello obj;|
10     obj.run();//需要把class文件与use_hello cpp关联
11 }
```

编译配置文件

1、首先配置头文件

class 的.h 文件在哪里

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

添加 c++ 的库 定义一个别名 这个别名与.h 头文件和.cpp 源文件关联

```
## Declare a C++ library
add_library(head_src
  include/plumbing_head_src/hello.h ##头文件的
  src/hello.cpp##源文件
)
```

148 行是给我们的 这个 class 函数编译一个类， 这个文件编译前， 要把相应的依赖提前编译号
152-154 我们把 这个 head_src 在编译的过程中， 需要把这个放进去

```
138 add_executable(use_hello src/use_hello.cpp)
139
140 ## Rename C++ executable without prefix
141 ## The above recommended prefix causes long target names, the following renames the
142 ## target back to the shorter version for ease of user use
143 ## e.g. "roslaunch someones_pkg node" instead of "roslaunch someones_pkg someones_pkg_node"
144 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
145
146 ## Add cmake target dependencies of the executable
147 ## same as for the library above
148 add_dependencies(head_src ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
149 add_dependencies(use_hello ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
150 ## Specify libraries to link a library or executable target against
151
152 target_link_libraries(head_src
153 | ${catkin_LIBRARIES}
154 )
155 target_link_libraries(use_hello ##use hello要连接到 c++库上 因此添加head_src
156 | head_src
157 | ${catkin_LIBRARIES}
158 )
```

138 我们 roslaunch plumbing_head_src use_hello

use_hello 是一个别名 与.cpp 文件关联 这个是主文件

149 我们的这个文件的依赖， 要在这个文件编译前 提前编译好

155-157 我们的这个文件也要关联到 c++的库上 就是我们自定义的 head_src 这个别名
指向了 .h 的头文件和 .cpp 的源文件