

代码导出pdf测试

```
1 #include "HelloWorldPubSubTypes.h" //自定义数据生成的序列化和反序列化文件
2
3 #include <fastdds/dds/domain/DomainParticipantFactory.hpp> //用于创建和销毁DomainParticipant对象
4 #include <fastdds/dds/domain/DomainParticipant.hpp>
5 #include <fastdds/dds/topic/TypeSupport.hpp>
6 #include <fastdds/dds/publisher/Publisher.hpp>
7 #include <fastdds/dds/publisher/DataWriter.hpp>
8 #include <fastdds/dds/publisher/DataWriterListener.hpp>
9 using namespace eprosima::fastdds::dds; //名称空间
10
11 class HelloWorldPublisher //发布者类
12 {
13 public:
14     HelloWorldPublisher(): participant_(nullptr), publisher_(nullptr),
15                             topic_(nullptr), writer_(nullptr),
16                             type_(new HelloWorldPubSubType()){}
17
18     virtual ~HelloWorldPublisher()
19     { //手动析构，清空指针成员
20         if (writer_ != nullptr){
21             publisher_>delete_datawriter(writer_);
22         }
23         if (publisher_ != nullptr){
24             participant_>delete_publisher(publisher_);
25         }
26         if (topic_ != nullptr){
27             participant_>delete_topic(topic_);
28         }
29         DomainParticipantFactory::get_instance()->delete_participant(participant_);
30     }
31
32     //初始化发布者
33     bool init()
34     {
35         hello_.index(0); //要发布的数据
36         hello_.message("HelloWorld");
37
38         //实例化participant_
39         DomainParticipantQos participantQos; //要为参与者设置的QOS
40         participantQos.name("Participant_publisher");
41         participant_ = DomainParticipantFactory::get_instance()->create_parti
```

```

42 cipant(0, participantQos);
43     if (participant_ == nullptr) return false;
44
45     //向参与者注册话题数据类型
46     type_.register_type(participant_);
47
48     //由参与者构造发布话题
49     topic_ = participant_->create_topic("HelloWorldTopic", "HelloWorld",
TOPIC_QOS_DEFAULT);
50     if (topic_ == nullptr) return false;
51
52     //由参与者构造发布者
53     publisher_ = participant_->create_publisher(PUBLISHER_QOS_DEFAULT, nu
llptr);
54     if (publisher_ == nullptr) return false;
55
56     //由发布者构造DataWriter
57     writer_ = publisher_->create_datawriter(topic_, DATAWRITER_QOS_DEFAULT, &listener_);
58     if (writer_ == nullptr) return false;
59
60     return true;
61 }
62
63 //执行发布动作
64 bool publish()
65 {
66     if (listener_.matched_ > 0)    //当出现匹配的订阅者时进行发布
67     {
68         hello_.index(hello_.index() + 1);
69         writer_->write(&hello_);
70         return true;
71     }
72     return false;
73 }
74
75 //发布函数接口
76 void run(uint32_t samples)
77 {
78     uint32_t samples_sent = 0;
79     while (samples_sent < samples)    //控制发布一定的次数
80     {
81         if (publish()){
82             samples_sent++;
83             std::cout << "Message: " << hello_.message() << " with index: " << hello_.index() << " SENT" << std::endl;
84         }
85         std::this_thread::sleep_for(std::chrono::milliseconds(1000));

```

```

85 //控制发布频率
86     }
87 }
88
89 private:
90     HelloWorld hello_; //idl文件自定义数据的类对象
91     DomainParticipant* participant_; //参与者(类似于ros中的node),用于构建
92     Publisher* publisher_; //发布者,负责创建DataWriters的对象
93     Topic* topic_; //话题
94     DataWriter* writer_; //向发布话题中写入数据
95     TypeSupport type_; //为参与者提供序列化、反序列化和获取特
96     定数据类型的密钥的功能。
97     class PubListener : public DataWriterListener //继承DataWriterListener
98     {
99     public:
100         PubListener(): matched_(0){}
101         ~PubListener() override{}
102
103         //覆盖默认的DataWriter侦听器回调。当检测到新的DataReader侦听DataWriter发布
104         的话题时,重写回调函数来自定义一些操作
105         void on_publication_matched(DataWriter*,const PublicationMatchedStatu
106         s& info) override
107         {
108             if (info.current_count_change == 1){ //发布/订阅的匹配数
109                 matched_ = info.total_count; //注意这里用的是历史
110                 累计匹配数,不是当前匹配数
111                 std::cout << "Publisher matched." << std::endl;
112             }else if (info.current_count_change == -1){ //发布/订阅的匹配数
113                 matched_ = info.total_count;
114                 std::cout << "Publisher unmatched." << std::endl;
115             }else{
116                 std::cout << info.current_count_change << " is not a valid va
117                 lue for PublicationMatchedStatus current count change." << std::endl;
118             }
119         }
120         std::atomic_int matched_; //累计匹配的订阅者个
121         数
122     } listener_; //listener_对象是为了跟踪订阅状态的变
123     化
124 };
125
126 int main(int argc,char** argv)
127 {
128     std::cout << "Starting publisher." << std::endl;

```

```

123     int samples = 10;                //控制发布10次信息
124
125     HelloWorldPublisher* mypub = new HelloWorldPublisher();    //发布对象的指
126     针
127     if(mypub->init()){                //初始化
128         mypub->run(static_cast<uint32_t>(samples)); //执行具体的发布
129     }
130     delete mypub;                    //清空指针后退出
131     return 0;
    }

```

```

1  cmake_minimum_required(VERSION 3.2)    #要求最小版本3.2,和上面有些不同
2
3  project(cartographer)                  #工程名
4
5  set(CARTOGRAPHER_MAJOR_VERSION 1)      #设置版本号
6  set(CARTOGRAPHER_MINOR_VERSION 0)
7  set(CARTOGRAPHER_PATCH_VERSION 0)
8  set(CARTOGRAPHER_VERSION ${CARTOGRAPHER_MAJOR_VERSION}.${CARTOGRAPHER_MINOR_V
  ersion}.${CARTOGRAPHER_PATCH_VERSION})
9  set(CARTOGRAPHER_SOVERSION ${CARTOGRAPHER_MAJOR_VERSION}.${CARTOGRAPHER_MINOR
  _VERSION})
10 option(BUILD_GRPC "build Cartographer gRPC support" false) #是否构建gRPC, fals
11 e
12 set(CARTOGRAPHER_HAS_GRPC ${BUILD_GRPC})
13 option(BUILD_PROMETHEUS "build Prometheus monitoring support" false)
14
15 include("${PROJECT_SOURCE_DIR}/cmake/functions.cmake")    #引入functions.cmake
16 文件
17 google_initialize_cartographer_project()    #执行函数(引入文件中定义的函数)
18 # google_enable_testing()
19
20 find_package( absl REQUIRED )
21 set(BOOST_COMPONENTS iostreams)
22 if(WIN32)
23     list(APPEND BOOST_COMPONENTS zlib)
24     set(Boost_USE_STATIC_LIBS FALSE)
25 endif()
26 find_package(Boost REQUIRED COMPONENTS ${BOOST_COMPONENTS})
27 find_package(Ceres REQUIRED COMPONENTS SuiteSparse)
28 find_package(Eigen3 REQUIRED)
29 find_package(LuaGoogLe REQUIRED)
30 if(WIN32)

```

```

30     # On Windows, Protobuf is incorrectly found by the bundled CMake module, so
31     prefer native CMake config.
32     set(protobuf_MODULE_COMPATIBLE TRUE CACHE INTERNAL "")
33     find_package(Protobuf 3.0.0 CONFIG)
34 else()
35     find_package(Protobuf 3.0.0 REQUIRED)
36 endif()
37
38 if (${BUILD_GRPC})
39     find_package(async_grpc REQUIRED)
40 endif()
41
42 if (${BUILD_PROMETHEUS})
43     find_package( ZLIB REQUIRED )
44 endif()
45
46 include(FindPkgConfig)
47 if (NOT WIN32)
48     PKG_SEARCH_MODULE(CAIRO REQUIRED cairo>=1.12.16)
49 else()
50     find_library(CAIRO_LIBRARIES cairo)
51 endif()
52
53 # Only build the documentation if we can find Sphinx.
54 find_package(Sphinx)
55 if (SPHINX_FOUND)
56     add_subdirectory("docs")
57 endif()
58
59 #安装package.xml
60 install(FILES package.xml DESTINATION share/cartographer)
61
62 set(CARTOGRAPHER_CONFIGURATION_FILES_DIRECTORY ${CMAKE_INSTALL_PREFIX}/share/
63 cartographer/configuration_files
64 CACHE PATH ".lua configuration files directory")
65
66 install(DIRECTORY configuration_files DESTINATION share/cartographer/)    #安
67                                     装配置文件夹
68
69 install(DIRECTORY cmake DESTINATION share/cartographer/)
70
71 file(GLOB_RECURSE ALL_LIBRARY_HDRS "cartographer/*.h")    #找到目录下所有.h文
72                                     件，放到变量ALL_LIBRARY_HDRS中
73 file(GLOB_RECURSE ALL_LIBRARY_SRCS "cartographer/*.cc")
74 file(GLOB_RECURSE TEST_LIBRARY_HDRS "cartographer/fake*.h" "cartographer/*te
75 st_helpers*.h" "cartographer/mock*.h")
76 file(GLOB_RECURSE TEST_LIBRARY_SRCS "cartographer/fake*.cc" "cartographer/*t
77 est_helpers*.cc" "cartographer/mock*.cc")

```

```
72 file(GLOB_RECURSE ALL_TESTS "cartographer/*_test.cc")
73 file(GLOB_RECURSE ALL_EXECUTABLES "cartographer/*_main.cc")
74
75 # Remove dotfiles/-folders that could potentially pollute the build.
76 # 移除所有以.开头的文件
77 file(GLOB_RECURSE ALL_DOTFILES ".*/*")
78 if (ALL_DOTFILES)
79     list(REMOVE_ITEM ALL_LIBRARY_HDRS ${ALL_DOTFILES})
80     list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_DOTFILES})
81     list(REMOVE_ITEM TEST_LIBRARY_HDRS ${ALL_DOTFILES})
82     list(REMOVE_ITEM TEST_LIBRARY_SRCS ${ALL_DOTFILES})
83     list(REMOVE_ITEM ALL_TESTS ${ALL_DOTFILES})
84     list(REMOVE_ITEM ALL_EXECUTABLES ${ALL_DOTFILES})
85 endif()
86
87 list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_EXECUTABLES})
88 list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_TESTS})
89 list(REMOVE_ITEM ALL_LIBRARY_HDRS ${TEST_LIBRARY_HDRS})
90 list(REMOVE_ITEM ALL_LIBRARY_SRCS ${TEST_LIBRARY_SRCS})
91 file(GLOB_RECURSE ALL_GRPC_FILES "cartographer/cloud/*")
92     file(GLOB_RECURSE ALL_PROMETHEUS_FILES "cartographer/cloud/metrics/prometheu
93 s/*")
94 list(REMOVE_ITEM ALL_GRPC_FILES ${ALL_PROMETHEUS_FILES})
95 if (NOT ${BUILD_GRPC})
96     list(REMOVE_ITEM ALL_LIBRARY_HDRS ${ALL_GRPC_FILES})
97     list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_GRPC_FILES})
98     list(REMOVE_ITEM TEST_LIBRARY_HDRS ${ALL_GRPC_FILES})
99     list(REMOVE_ITEM TEST_LIBRARY_SRCS ${ALL_GRPC_FILES})
100     list(REMOVE_ITEM ALL_TESTS ${ALL_GRPC_FILES})
101     list(REMOVE_ITEM ALL_EXECUTABLES ${ALL_GRPC_FILES})
102 endif()
103 if (NOT ${BUILD_PROMETHEUS})
104     list(REMOVE_ITEM ALL_LIBRARY_HDRS ${ALL_PROMETHEUS_FILES})
105     list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_PROMETHEUS_FILES})
106     list(REMOVE_ITEM TEST_LIBRARY_HDRS ${ALL_PROMETHEUS_FILES})
107     list(REMOVE_ITEM TEST_LIBRARY_SRCS ${ALL_PROMETHEUS_FILES})
108     list(REMOVE_ITEM ALL_TESTS ${ALL_PROMETHEUS_FILES})
109     list(REMOVE_ITEM ALL_EXECUTABLES ${ALL_PROMETHEUS_FILES})
110 endif()
111
112 set(INSTALL_SOURCE_HDRS ${ALL_LIBRARY_HDRS} ${TEST_LIBRARY_HDRS})
113
114 # 移除internal文件夹内的头文件,不进行安装
115 file(GLOB_RECURSE INTERNAL_HDRS "cartographer/*/internal/*.h")
116 list(REMOVE_ITEM INSTALL_SOURCE_HDRS ${INTERNAL_HDRS})
117
118 file(GLOB_RECURSE ALL_PROTOS "cartographer/*.proto")
119 file(GLOB_RECURSE ALL_GRPC_SERVICES "cartographer/*_service.proto")
```

```

119 list(REMOVE_ITEM ALL_PROTOS ALL_GRPC_SERVICES)
120 if (NOT ${BUILD_GRPC})
121     list(REMOVE_ITEM ALL_PROTOS ${ALL_GRPC_FILES})
122 endif()
123
124 # TODO(cschuet): Move proto compilation to separate function.
125 set(ALL_PROTO_SRCS)
126 set(ALL_PROTO_HDRS)
127 foreach(ABS_FIL ${ALL_PROTOS})
128     file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${ABS_FIL})
129     get_filename_component(DIR ${REL_FIL} DIRECTORY)
130     get_filename_component(FIL_WE ${REL_FIL} NAME_WE)
131
132     list(APPEND ALL_PROTO_SRCS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc")
133     list(APPEND ALL_PROTO_HDRS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h")
134
135     add_custom_command(
136         OUTPUT "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc"
137             "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h"
138         COMMAND ${PROTOBUF_PROTOC_EXECUTABLE}
139         ARGS --cpp_out ${PROJECT_BINARY_DIR} -I
140             ${PROJECT_SOURCE_DIR} ${ABS_FIL}
141         DEPENDS ${ABS_FIL}
142         COMMENT "Running C++ protocol buffer compiler on ${ABS_FIL}"
143         VERBATIM
144     )
145 endforeach()
146     set_source_files_properties(${ALL_PROTO_SRCS} ${ALL_PROTO_HDRS} PROPERTIES GENERATED TRUE)
147
148 # 添加.pb.h与.pb.cc到头文件源文件列表
149 list(APPEND ALL_LIBRARY_HDRS ${ALL_PROTO_HDRS})
150 list(APPEND ALL_LIBRARY_SRCS ${ALL_PROTO_SRCS})
151
152 if(${BUILD_GRPC})
153     set(ALL_GRPC_SERVICE_SRCS)
154     set(ALL_GRPC_SERVICE_HDRS)
155     foreach(ABS_FIL ${ALL_GRPC_SERVICES})
156         file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${ABS_FIL})
157         get_filename_component(DIR ${REL_FIL} DIRECTORY)
158         get_filename_component(FIL_WE ${REL_FIL} NAME_WE)
159
160         list(APPEND ALL_GRPC_SERVICE_SRCS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc")
161         list(APPEND ALL_GRPC_SERVICE_HDRS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h")
162
163         add_custom_command(

```

```

164     OUTPUT "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc"
165           "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h"
166     COMMAND ${PROTOBUF_PROTOC_EXECUTABLE}
167     ARGS --cpp_out ${PROJECT_BINARY_DIR}
168          -I ${PROJECT_SOURCE_DIR}
169          ${ABS_FIL}
170     DEPENDS ${ABS_FIL}
171     COMMENT "Running C++ protocol buffer compiler on ${ABS_FIL}"
172     VERBATIM
173 )
174 endforeach()
175     set_source_files_properties(${ALL_GRPC_SERVICE_SRCS} ${ALL_GRPC_SERVICE_HDR
176 s} PROPERTIES GENERATED TRUE)
177     list(APPEND ALL_LIBRARY_HDRS ${ALL_GRPC_SERVICE_HDRS})
178     list(APPEND ALL_LIBRARY_SRCS ${ALL_GRPC_SERVICE_SRCS})
179 endif()
180 set(INSTALL_GENERATED_HDRS ${ALL_PROTO_HDRS} ${ALL_GRPC_SERVICE_HDRS})
181 # 生成cartographer库文件
182 add_library(${PROJECT_NAME} STATIC ${ALL_LIBRARY_HDRS} ${ALL_LIBRARY_SRCS})
183
184 configure_file(
185     ${PROJECT_SOURCE_DIR}/cartographer/common/config.h.cmake
186     ${PROJECT_BINARY_DIR}/cartographer/common/config.h)
187
188 google_binary(cartographer_autogenerate_ground_truth
189     SRCS
190         cartographer/ground_truth/autogenerate_ground_truth_main.cc
191 )
192
193 google_binary(cartographer_compute_relations_metrics
194     SRCS
195         cartographer/ground_truth/compute_relations_metrics_main.cc
196 )
197
198 google_binary(cartographer_pbstream
199     SRCS
200         cartographer/io/pbstream_main.cc
201 )
202
203 google_binary(cartographer_print_configuration
204     SRCS
205         cartographer/common/print_configuration_main.cc
206 )
207
208 if(${BUILD_GRPC})
209     google_binary(cartographer_grpc_server
210         SRCS

```



```

211     cartographer/cloud/map_builder_server_main.cc
212 )
213 target_link_libraries(cartographer_grpc_server PUBLIC grpc++)
214 target_link_libraries(cartographer_grpc_server PUBLIC async_grpc)
215 if(${BUILD_PROMETHEUS})
216     target_link_libraries(cartographer_grpc_server PUBLIC ${ZLIB_LIBRARIES})
217     target_link_libraries(cartographer_grpc_server PUBLIC prometheus-cpp-cor
218 e)
219     target_link_libraries(cartographer_grpc_server PUBLIC prometheus-cpp-pu
220 l)
221 endif()
222 endif()
223
224 target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
225     "${EIGEN3_INCLUDE_DIR}")
226 target_link_libraries(${PROJECT_NAME} PUBLIC ${EIGEN3_LIBRARIES})
227
228 target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
229     "${CERES_INCLUDE_DIRS}")
230 target_link_libraries(${PROJECT_NAME} PUBLIC ${CERES_LIBRARIES})
231
232 target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
233     "${LUA_INCLUDE_DIR}")
234 target_link_libraries(${PROJECT_NAME} PUBLIC ${LUA_LIBRARIES})
235
236 target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
237     "${Boost_INCLUDE_DIRS}")
238 target_link_libraries(${PROJECT_NAME} PUBLIC ${Boost_LIBRARIES})
239
240 if (WIN32)
241     find_package(glog REQUIRED)
242     set(GLOG_LIBRARY glog::glog)
243 else()
244     set(GLOG_LIBRARY glog)
245 endif()
246
247 target_link_libraries(${PROJECT_NAME} PUBLIC ${GLOG_LIBRARY})
248 target_link_libraries(${PROJECT_NAME} PUBLIC gflags)
249 if(WIN32)
250     # Needed to fix conflict with MSVC's error macro.
251     target_compile_definitions(${PROJECT_NAME} PUBLIC -DGLOG_NO_ABBREVIATED_SEV
252 ERITIES)
253 endif()
254 if(MSVC)
255     # Needed for VS 2017 5.8
256     target_compile_definitions(${PROJECT_NAME} PUBLIC -D_ENABLE_EXTENDED_ALIGNE
257 D_STORAGE -D_USE_MATH_DEFINES)
258 endif()

```

```
255
256 if("${CAIRO_INCLUDE_DIRS}")
257     target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
258         "${CAIRO_INCLUDE_DIRS}")
259 endif()
260 target_link_libraries(${PROJECT_NAME} PUBLIC ${CAIRO_LIBRARIES})
261
262 target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
263     ${PROTOBUF_INCLUDE_DIR})
264 # TODO(hrapp): This should not explicitly list pthread and use
265 # PROTOBUF_LIBRARIES, but that failed on first try.
266 target_link_libraries(${PROJECT_NAME} PUBLIC ${PROTOBUF_LIBRARY}
267     absl::algorithm
268     absl::base
269     absl::debugging
270     absl::flat_hash_map
271     absl::memory
272     absl::meta
273     absl::numeric
274     absl::str_format
275     absl::strings
276     absl::synchronization
277     absl::time
278     absl::utility
279 )
280 if (NOT WIN32)
281     target_link_libraries(${PROJECT_NAME} PUBLIC pthread)
282 endif()
283 if(${BUILD_GRPC})
284     target_link_libraries(${PROJECT_NAME} PUBLIC grpc++)
285     target_link_libraries(${PROJECT_NAME} PUBLIC async_grpc)
286 endif()
287 if(${BUILD_PROMETHEUS})
288     target_link_libraries(${PROJECT_NAME} PUBLIC ${ZLIB_LIBRARIES})
289     target_link_libraries(${PROJECT_NAME} PUBLIC prometheus-cpp-core)
290     target_link_libraries(${PROJECT_NAME} PUBLIC prometheus-cpp-pull)
291     target_compile_definitions(${PROJECT_NAME} PUBLIC USE_PROMETHEUS=1)
292 endif()
293
294 set(TARGET_COMPILE_FLAGS "${TARGET_COMPILE_FLAGS} ${GOOG_CXX_FLAGS}")
295 set_target_properties(${PROJECT_NAME} PROPERTIES
296     COMPILE_FLAGS ${TARGET_COMPILE_FLAGS})
297
298 # 不编译test_library
299
300 # set(TEST_LIB
301 #     cartographer_test_library
302 # )
```

```

303 # add_library(${TEST_LIB} ${TEST_LIBRARY_HDRS} ${TEST_LIBRARY_SRCS})
304 # target_include_directories(${TEST_LIB} SYSTEM PRIVATE
305 #   "${GMOCK_INCLUDE_DIRS}")
306 # # Needed for dynamically linked GTest on Windows.
307 # if (WIN32)
308     #   target_compile_definitions(${TEST_LIB} PUBLIC -DGTEST_LINKED_AS_SHARED_LI
309 BRARY)
310 # endif()
311 # target_link_libraries(${TEST_LIB} PUBLIC ${GMOCK_LIBRARY})
312 # target_link_libraries(${TEST_LIB} PUBLIC ${PROJECT_NAME})
313 # set_target_properties(${TEST_LIB} PROPERTIES
314 #   COMPILE_FLAGS ${TARGET_COMPILE_FLAGS})
315 # 不编译_test.cc文件
316
317 # foreach(ABS_FIL ${ALL_TESTS})
318 #   file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${ABS_FIL})
319 #   get_filename_component(DIR ${REL_FIL} DIRECTORY)
320 #   get_filename_component(FIL_WE ${REL_FIL} NAME_WE)
321 #   # Replace slashes as required for CMP0037.
322 #   string(REPLACE "/" "." TEST_TARGET_NAME "${DIR}/${FIL_WE}")
323 #   google_test("${TEST_TARGET_NAME}" ${ABS_FIL})
324 #   if(${BUILD_GRPC})
325 #       target_link_libraries("${TEST_TARGET_NAME}" PUBLIC grpc++)
326 #       target_link_libraries("${TEST_TARGET_NAME}" PUBLIC async_grpc)
327 #   endif()
328 #   if(${BUILD_PROMETHEUS})
329 #       target_link_libraries("${TEST_TARGET_NAME}" PUBLIC ${ZLIB_LIBRARIES})
330 #       target_link_libraries("${TEST_TARGET_NAME}" PUBLIC prometheus-cpp-core)
331 #       target_link_libraries("${TEST_TARGET_NAME}" PUBLIC prometheus-cpp-pull)
332 #   endif()
333 #   target_link_libraries("${TEST_TARGET_NAME}" PUBLIC ${TEST_LIB})
334 # endforeach()
335
336 # Add the binary directory first, so that port.h is included after it has
337 # been generated.
338 target_include_directories(${PROJECT_NAME} PUBLIC
339     $<BUILD_INTERFACE:${PROJECT_BINARY_DIR}>
340     $<BUILD_INTERFACE:${PROJECT_SOURCE_DIR}>
341     $<INSTALL_INTERFACE:include>
342 )
343
344 # 安装cartographer库
345 install(
346     TARGETS ${PROJECT_NAME}
347     EXPORT CartographerExport
348     ARCHIVE DESTINATION lib
349     LIBRARY DESTINATION lib

```

```
350     RUNTIME DESTINATION bin
351 )
352
353 # 安装头文件
354 foreach(HDR ${INSTALL_SOURCE_HDRS})
355     file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${HDR})
356     get_filename_component(DIR ${REL_FIL} DIRECTORY)
357     install(
358         FILES ${HDR}
359         DESTINATION include/${DIR}
360     )
361 endforeach()
362
363 foreach(HDR ${INSTALL_GENERATED_HDRS})
364     file(RELATIVE_PATH REL_FIL ${PROJECT_BINARY_DIR} ${HDR})
365     get_filename_component(DIR ${REL_FIL} DIRECTORY)
366     install(
367         FILES ${HDR}
368         DESTINATION include/${DIR}
369     )
370 endforeach()
371
372 set(CARTOGRAPHER_CMAKE_DIR share/cartographer/cmake)
373 include(CMakePackageConfigHelpers)
374 configure_package_config_file(
375     cartographer-config.cmake.in
376     ${PROJECT_BINARY_DIR}/cmake/cartographer/cartographer-config.cmake
377     PATH_VARS CARTOGRAPHER_CMAKE_DIR
378     INSTALL_DESTINATION ${CMAKE_INSTALL_PREFIX}/share/cartographer
379 )
380
381 install(
382     EXPORT CartographerExport
383     DESTINATION share/cartographer/cmake/
384     FILE CartographerTargets.cmake
385 )
386
387 install(
388     FILES ${PROJECT_BINARY_DIR}/cmake/cartographer/cartographer-config.cmake
389     DESTINATION share/cartographer/
390 )
```