第1篇 (基础) QT开发环境的搭建和HelloQt

代码地址：

```
https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Example01
```

一、软件下载

▼ 官方，**国内下载速度比较慢**

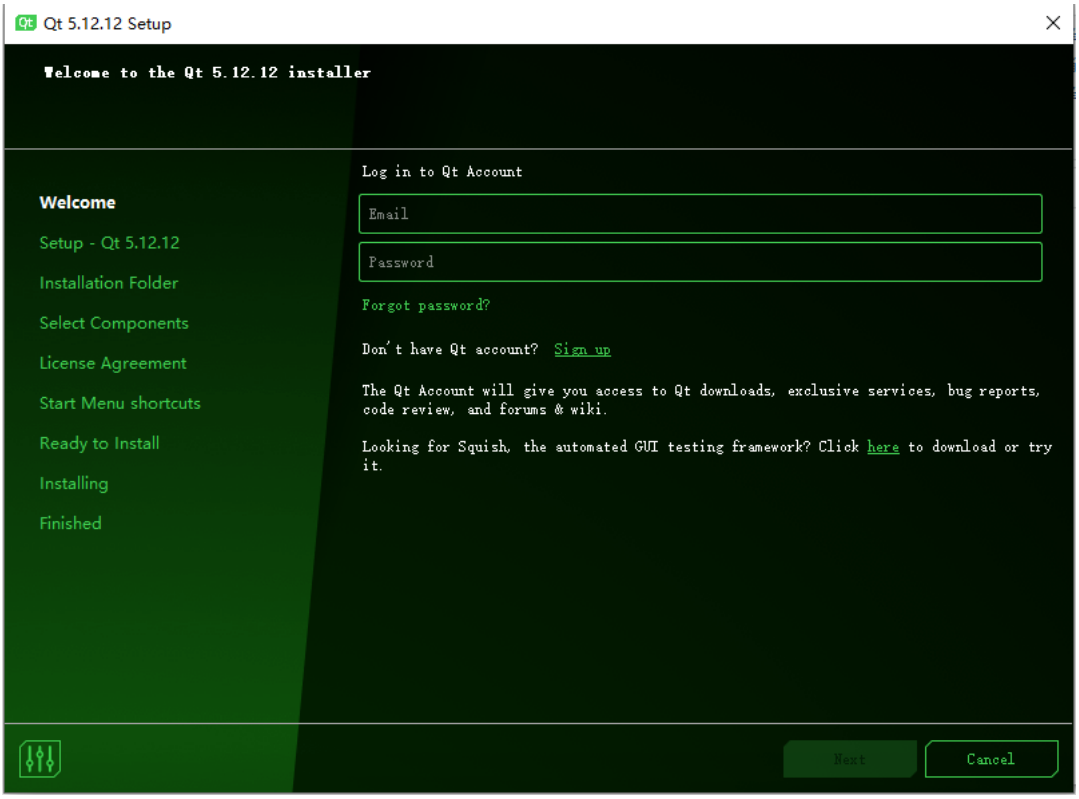
- Win：https://download.qt.io/archive/qt/5.12/5.12.12/qt-opensource-windows-x86-5.12.12.exe
- Mac：https://download.qt.io/archive/qt/5.12/5.12.12/qt-opensource-mac-x64-5.12.12.dmg
- Linux：https://download.qt.io/archive/qt/5.12/5.12.12/qt-opensource-linux-x64-5.12.12.run

▼ 国内第三方，**国内下载速度比较快**

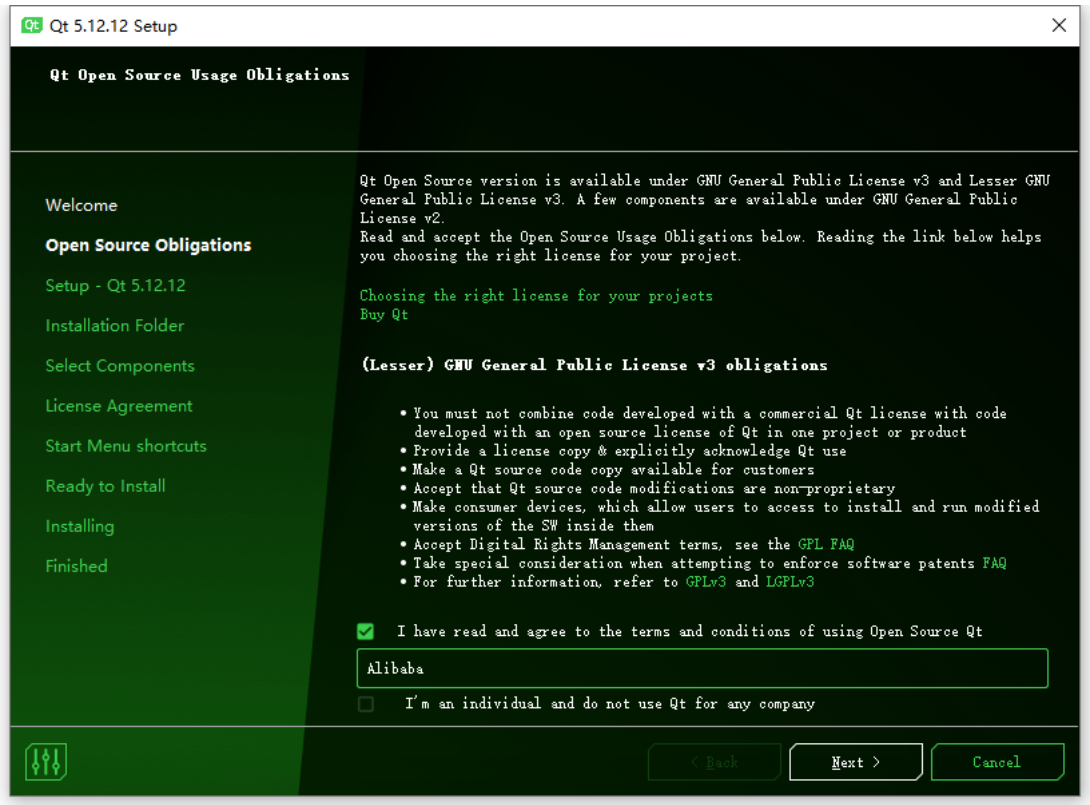
- Win：
- Mac：
- Linux：

二、安装Qt

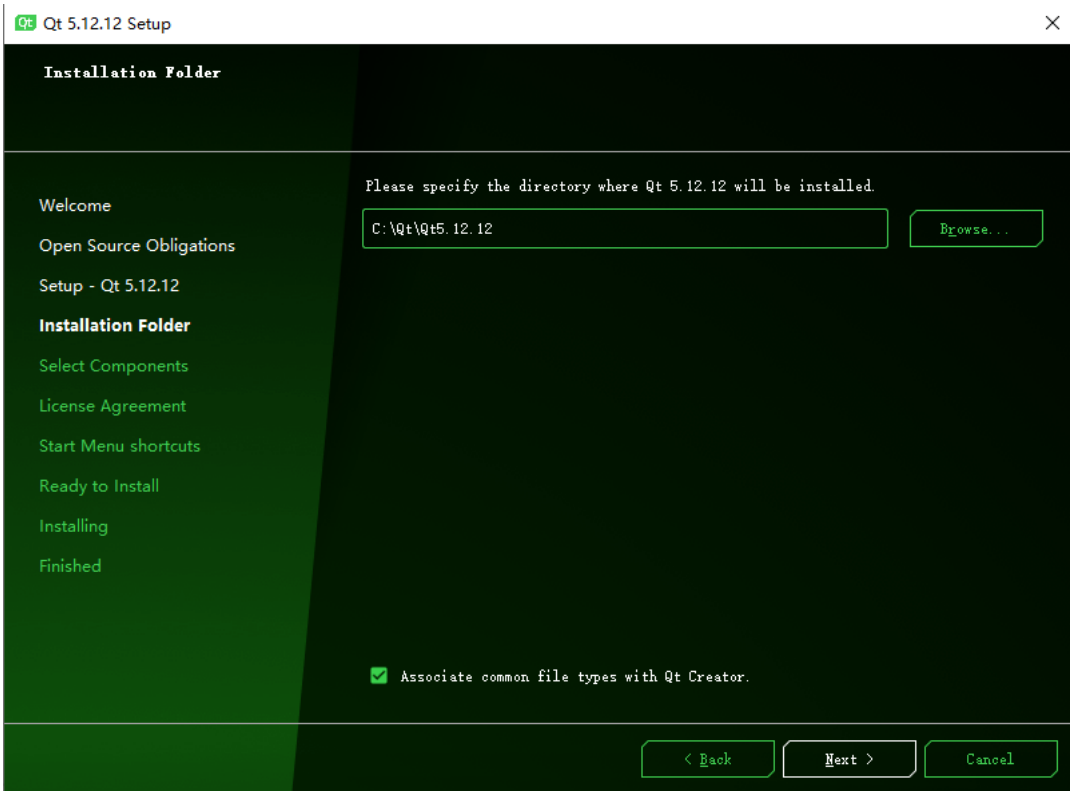
1. 双击软件安装包，exe结尾
2. 如下图所示，输入QT账号和密码，如果没有的话点击`Sign up`注册一个。



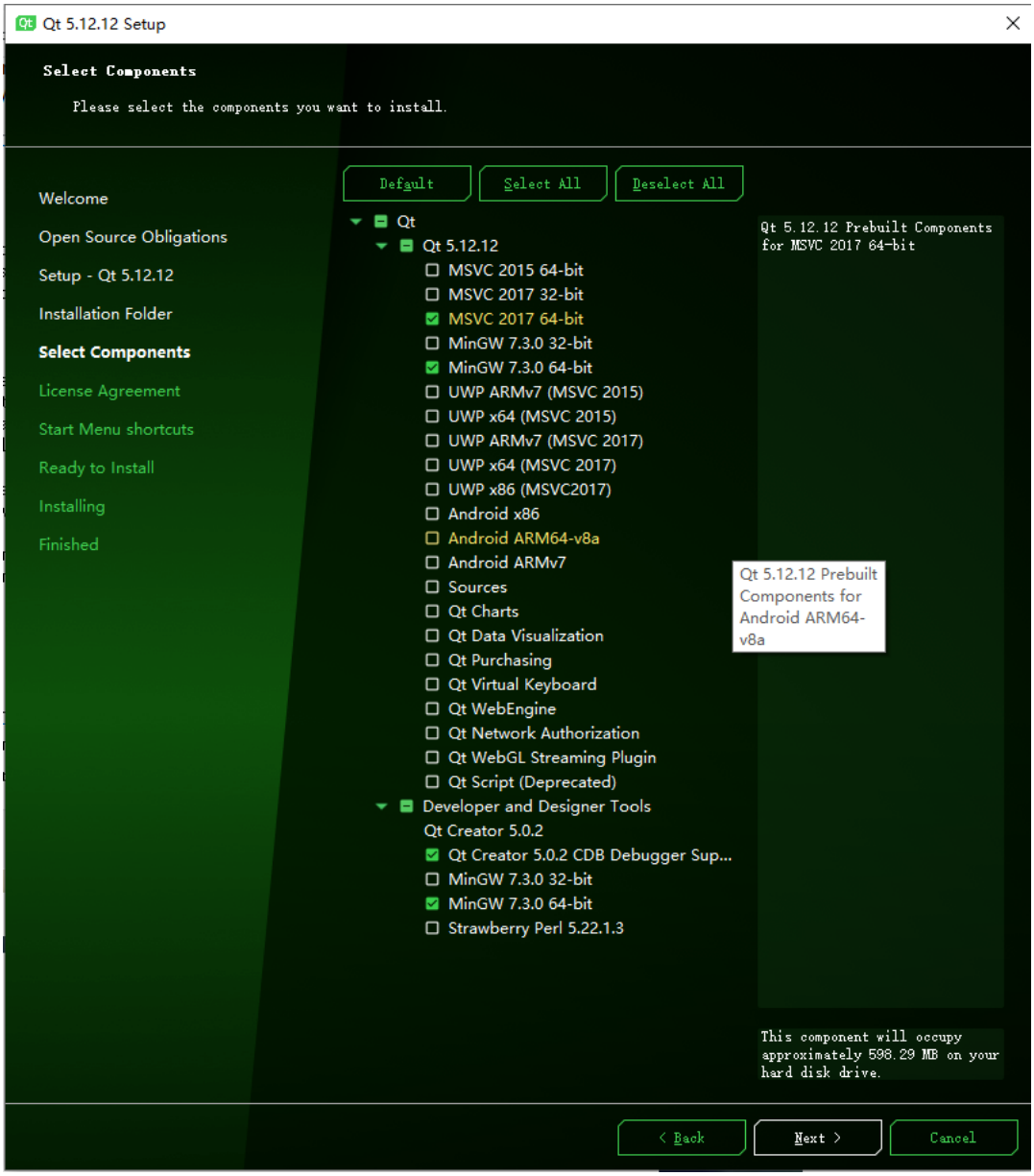
3. 如下图所示，随意输入一个Alibaba的公司便于注册。



4. 如下图所示，参考下列安装位置。

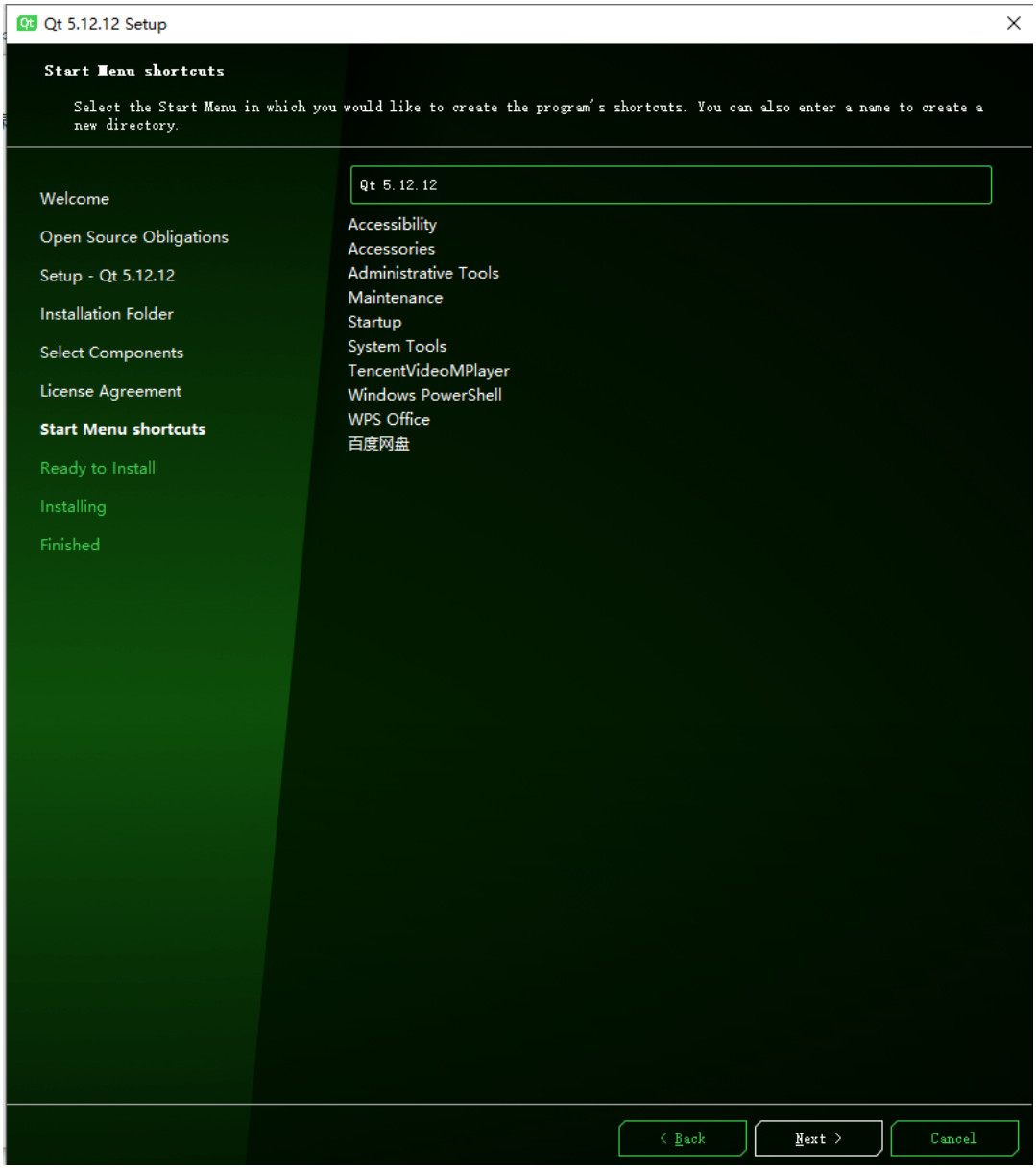


5. 如下图所示，安装一些开发过程中必要的包。

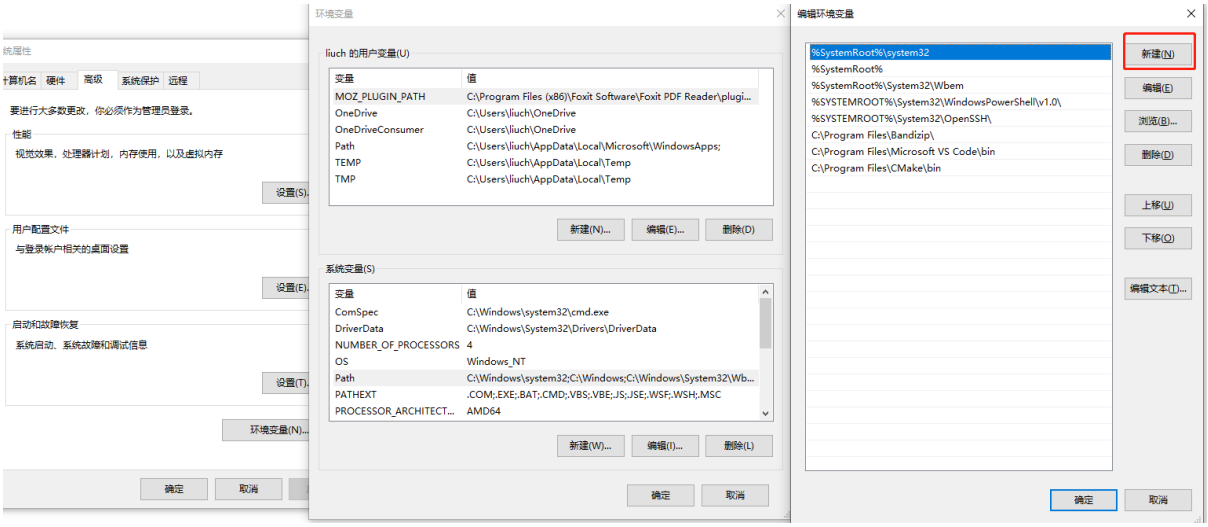


6. 如下图所示，同意许可证。

7. 如下图所示，安装QT后的软件名称，可选，默认即可。



8. 点击`Install`，耐心等待安装完成。
9. 打开“控制面板”并点击“系统和安全”（或“系统”）选项。
10. 点击“高级系统设置”链接。
11. 在“系统属性”对话框中，点击“高级”选项卡并点击“环境变量”按钮。
12. 如下图所示，在“环境变量”对话框中，找到“系统变量”区域找到“Path”按钮。

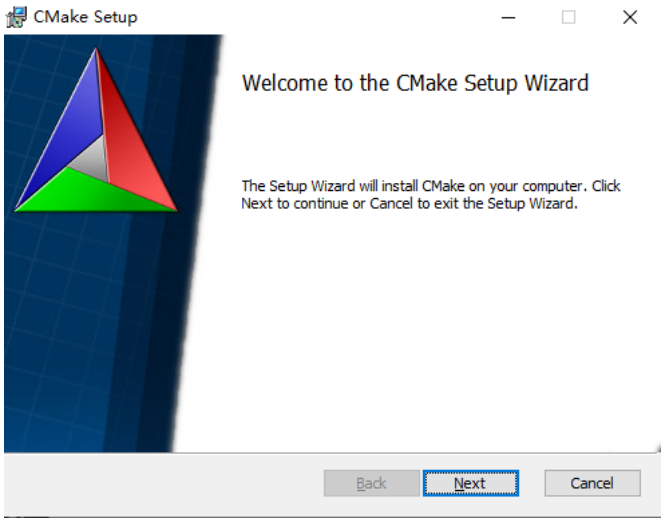


13. 如下图所示，在新建文本框中输入`C:\Qt\Qt5.12.12\5.12.12\mingw73_64\bin`

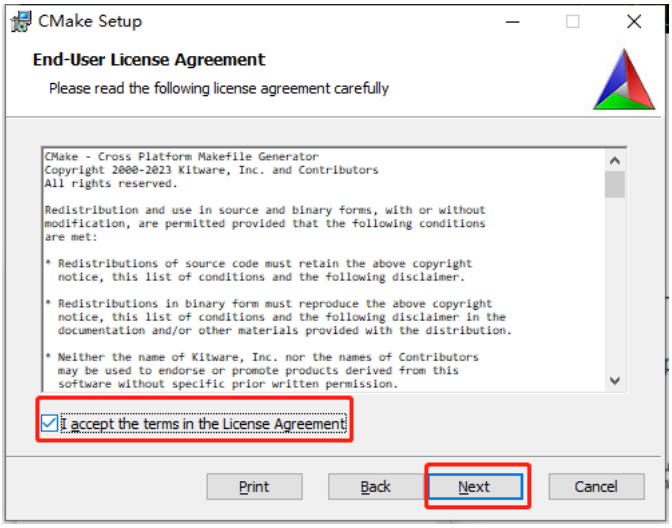
14.

三、安装CMake

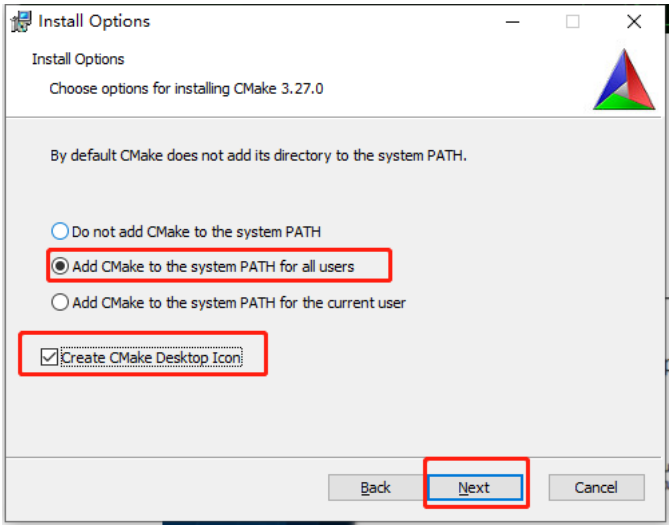
1. 双击软件安装包，msi结尾
2. 打开后如下图所示，点击`Next`。



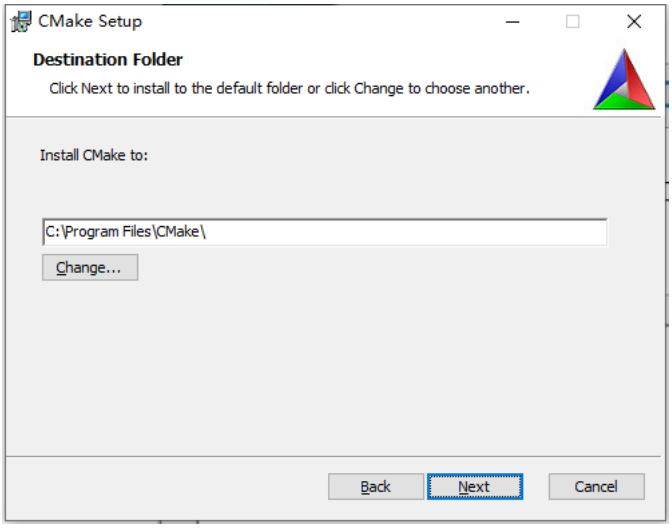
3. 如下图所示，点击`I agree`和`Next`。



4. 如下图所示，按照红框表示勾选，最后点击`Next`。

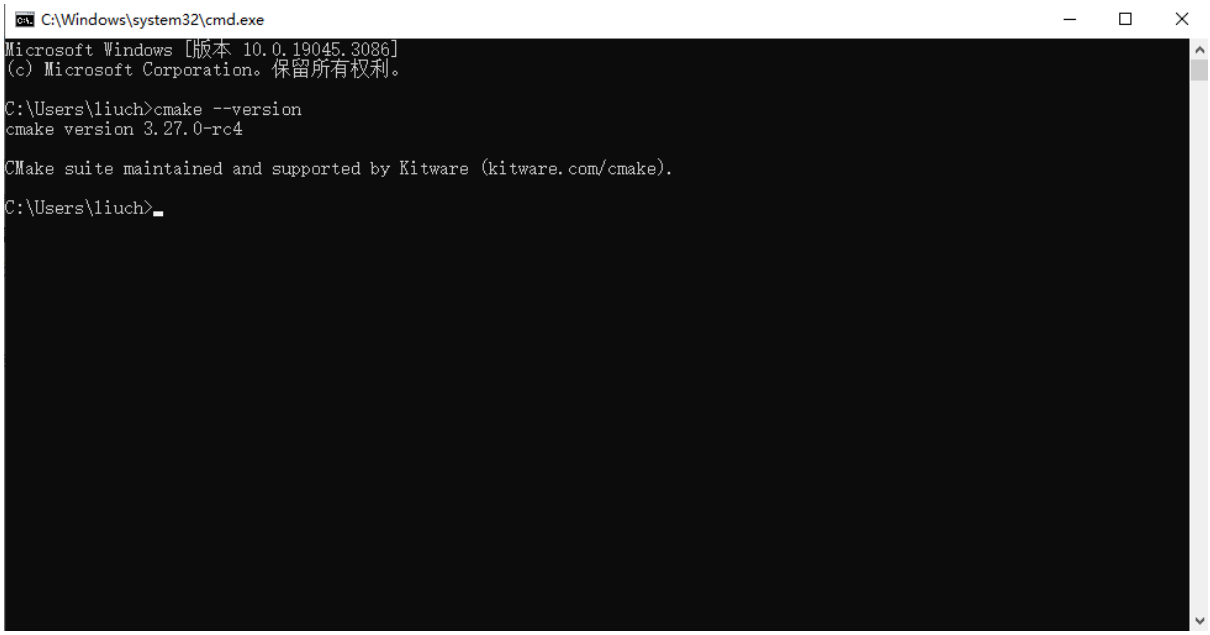


5. 如下图所示，按照默认路径安装即可。



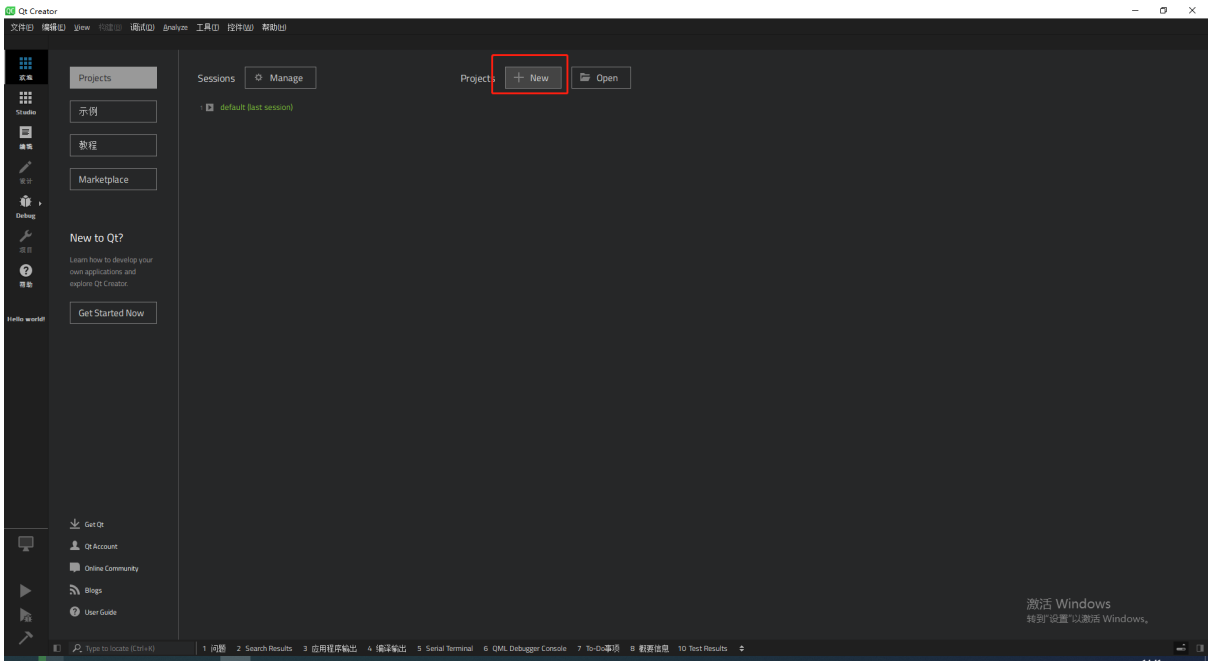
6. 等待安装完毕。

7. 如下图所示，验证CMake安装是否完成，`Win+R`输入`cmd`，输入`cmake - -version`，输出结果应该类似这样。

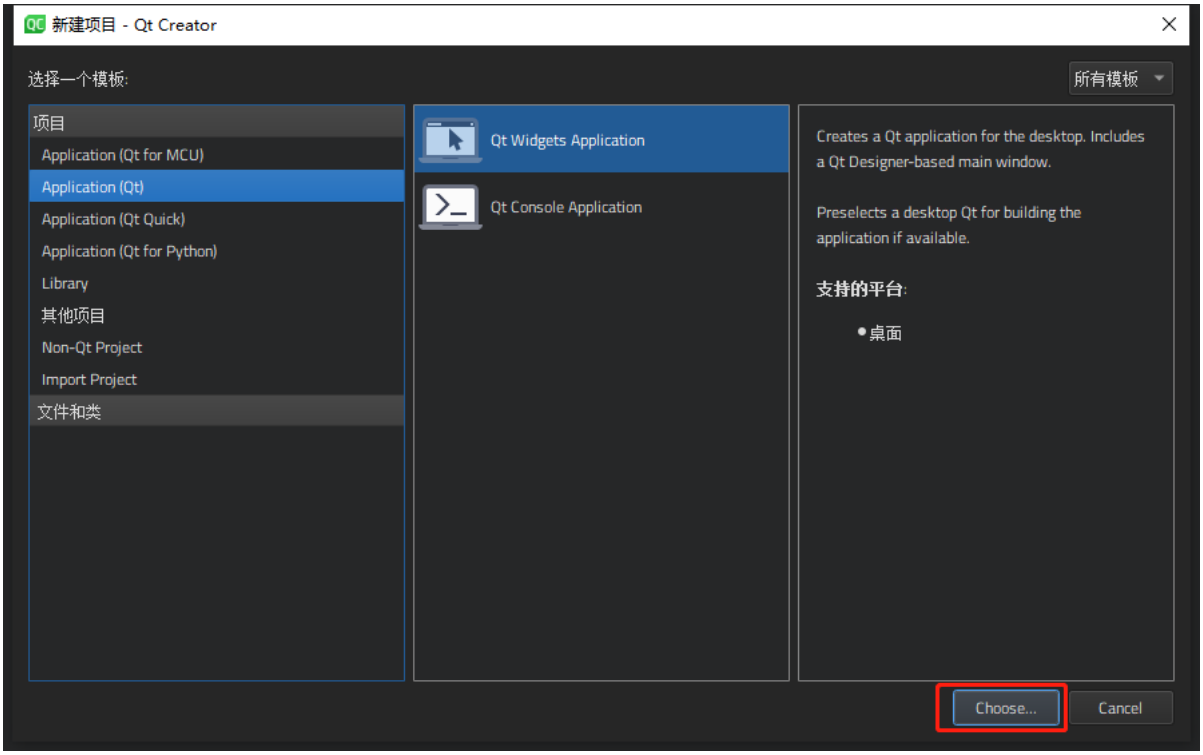


四、创建HelloQt程序

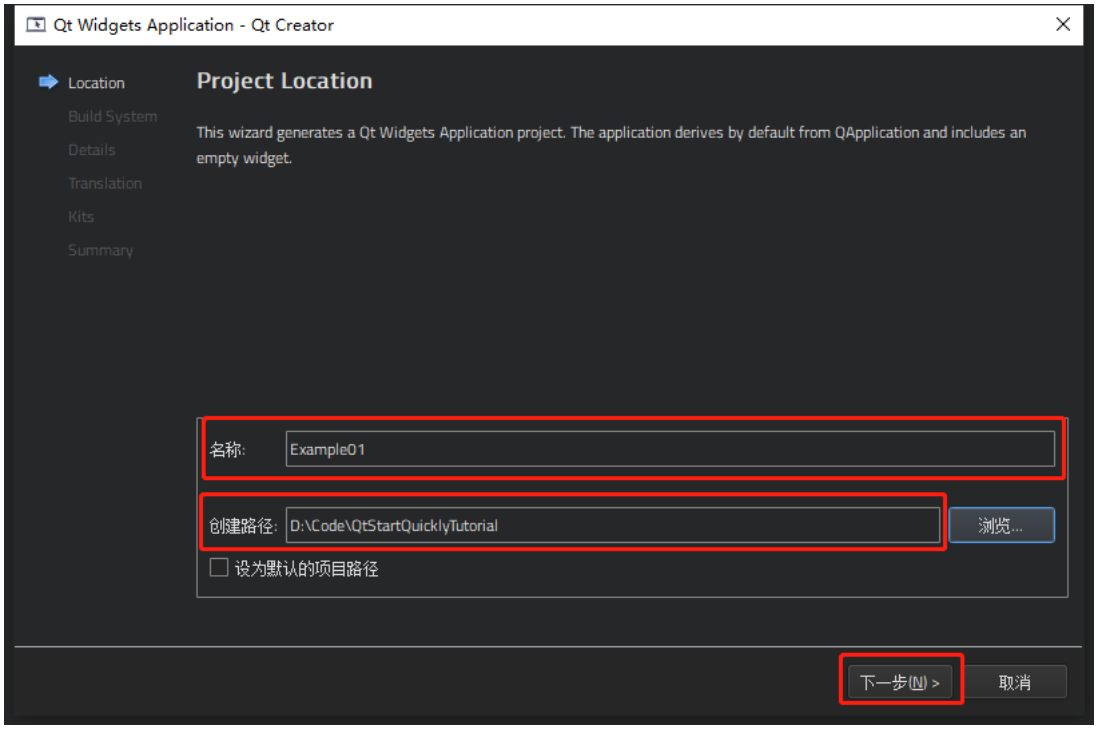
1. 如下图所示，创建一个新项目工程。



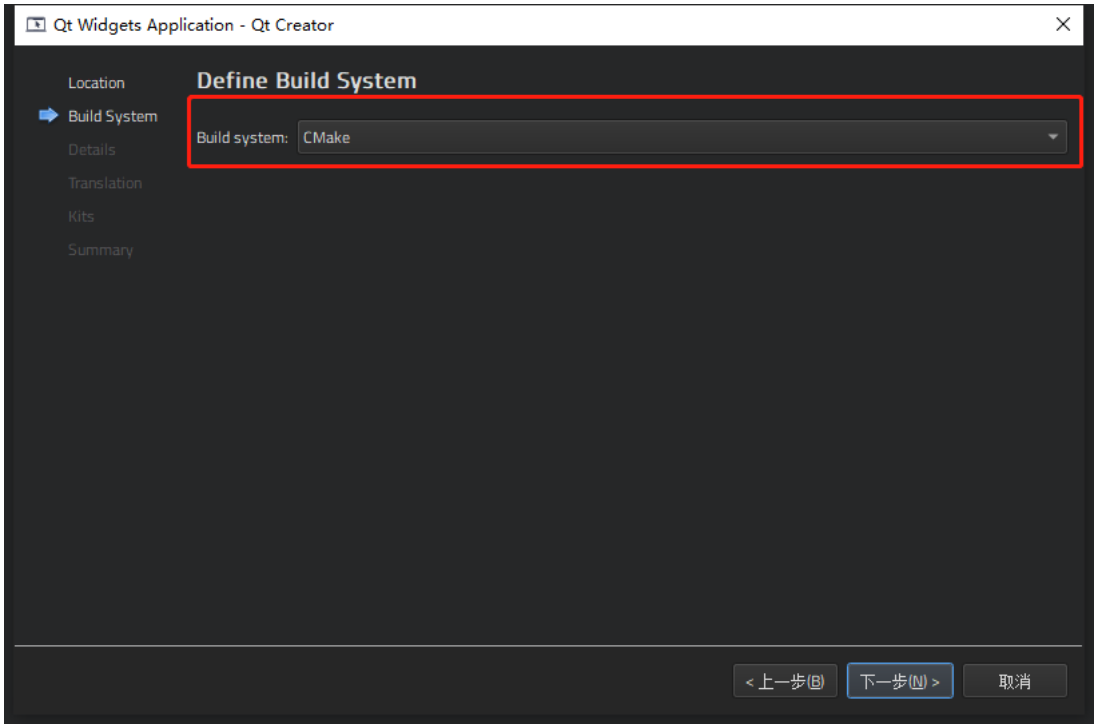
2. 如下图所示，选择QT控件的桌面端开发。



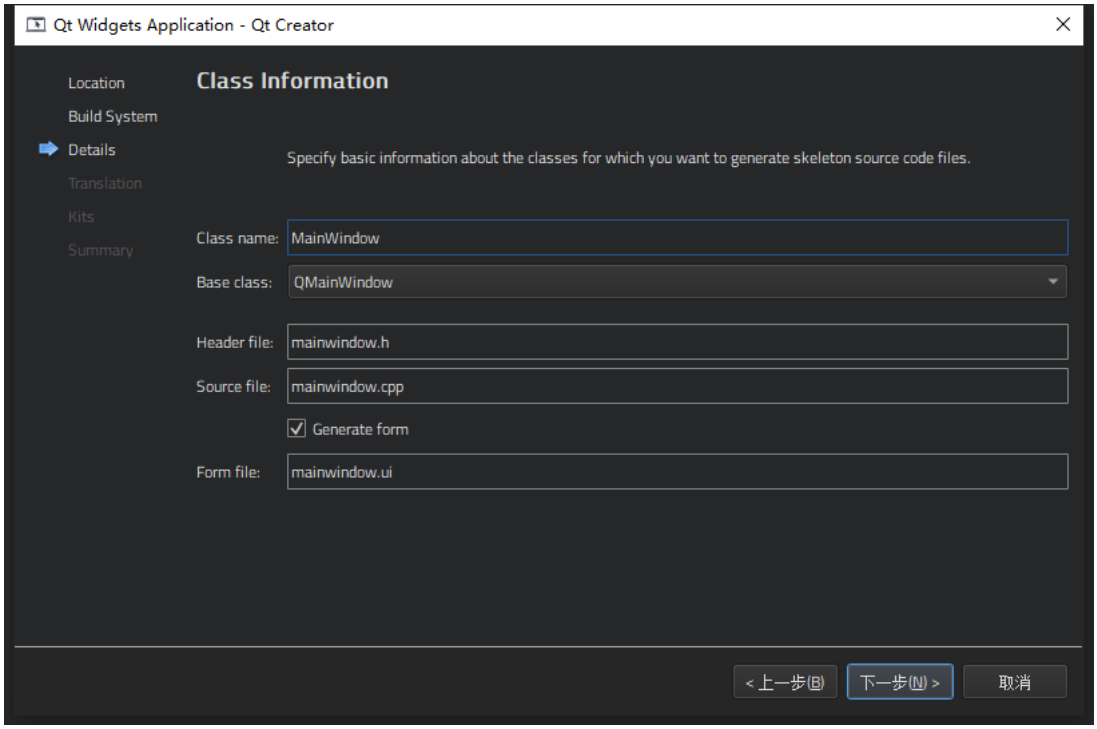
3. 如下图所示，配置好路径。



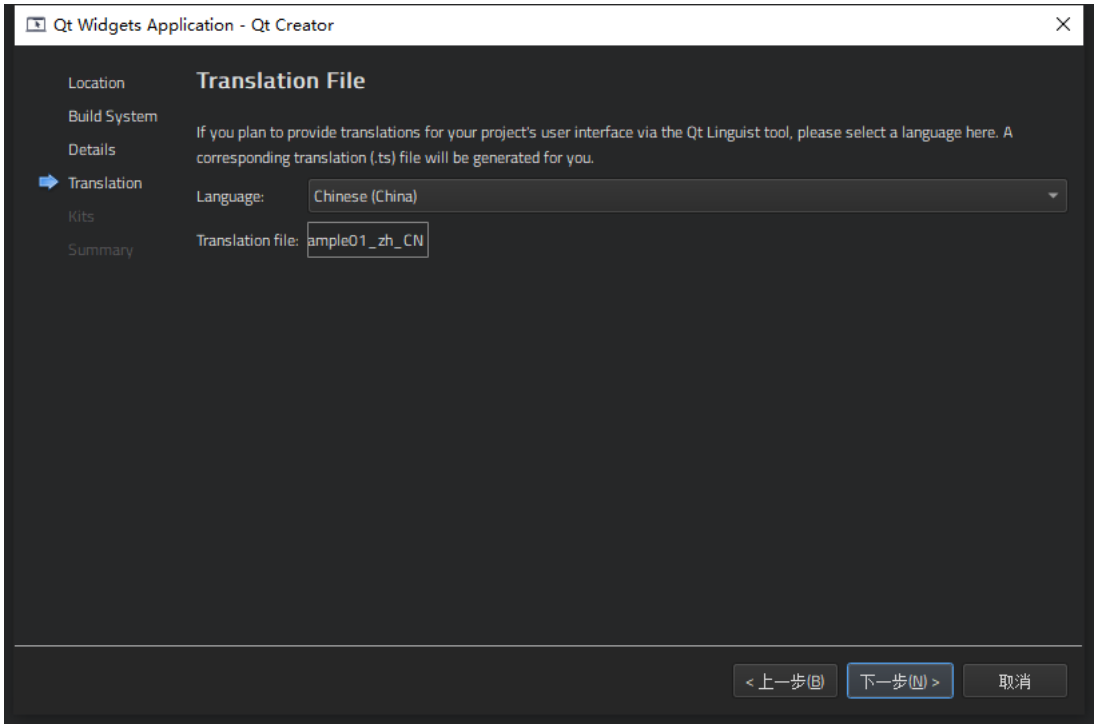
4. 如下图所示，选择CMake作为编译工具。



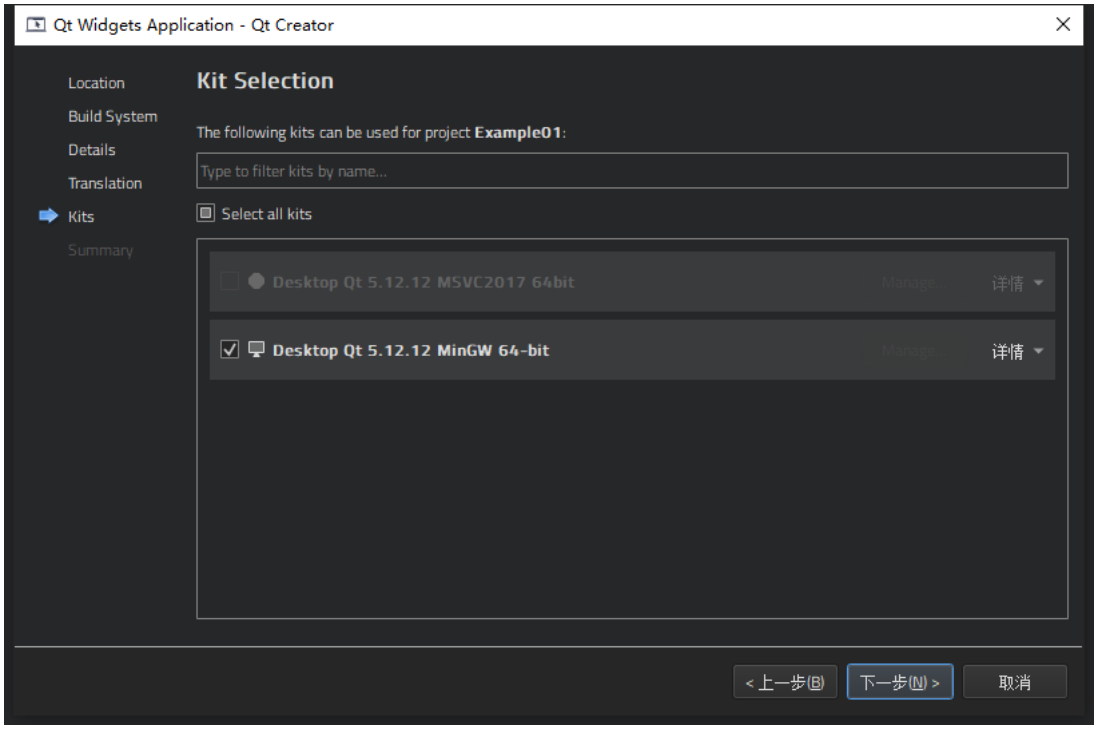
5. 如下图所示，定义头文件、UI和源文件。



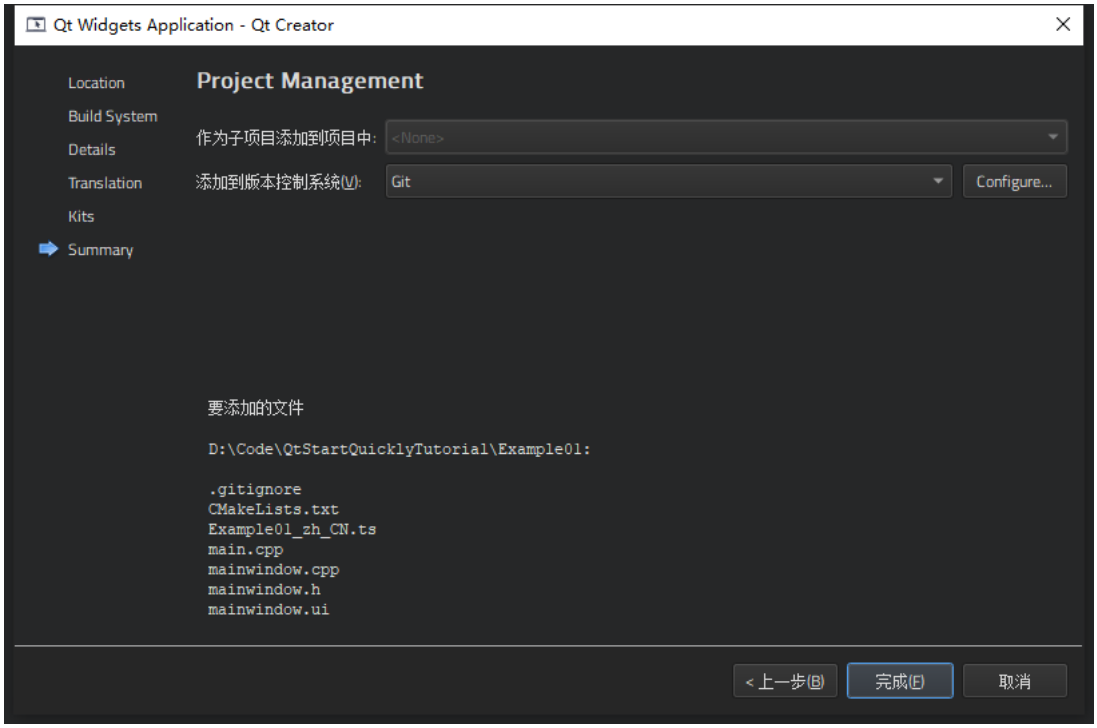
6. 如下图所示，翻译文件选择中文。



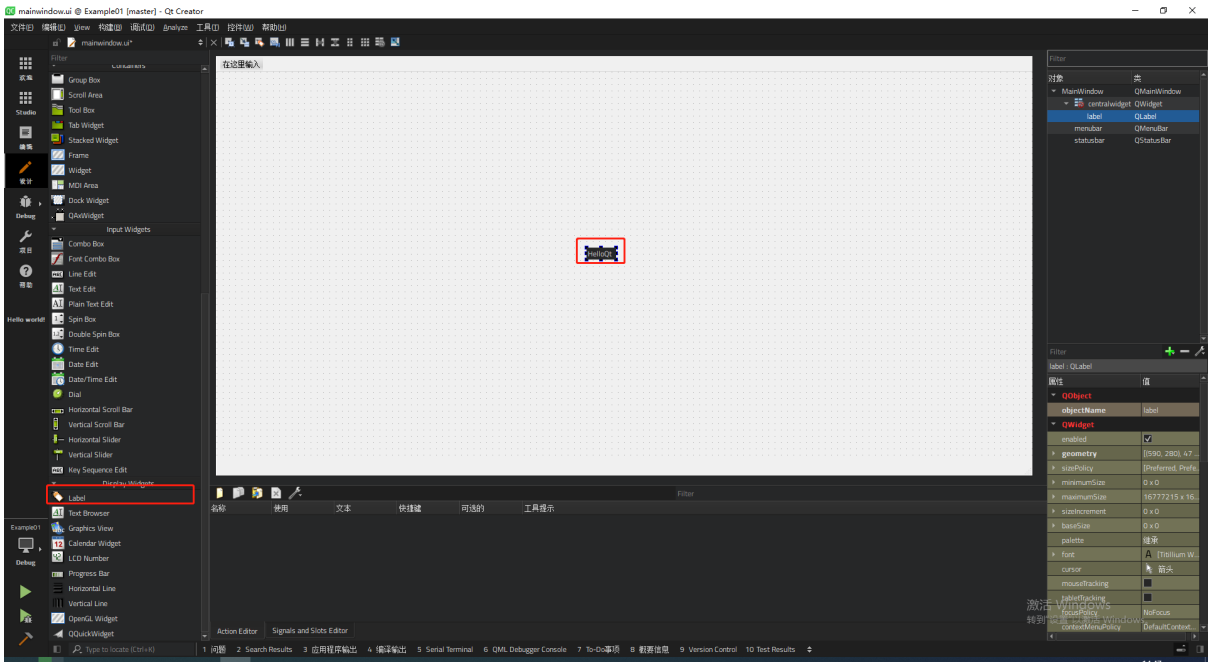
7. 如下图所示，使用GCC编译。



8. 如下图所示，添加版本控制文件，因为安装了Git所以参考如下配置，没有版本控制文件也是可以的。



9. 如下图所示，点击`设计`按钮，将Label控件拖入空白区域，并输入`HelloQt`。



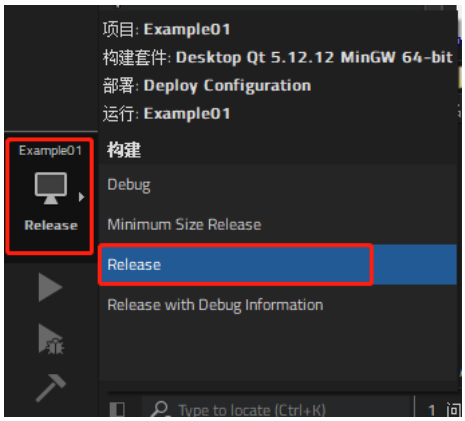
10. CMakeLists.txt文件内容解读

```
# 设置所需的最低CMake版本为3.26。
cmake_minimum_required(VERSION 3.26)
# 设置项目名称为"Example01"，版本号为0.1，使用的编程语言为C++。
project(Example01 VERSION 0.1 LANGUAGES CXX)

# 自动添加 CMAKE_CURRENT_BINARY_DIR 和 CMAKE_CURRENT_SOURCE_DIR 到当前目录
set(CMAKE_INCLUDE_CURRENT_DIR ON)

# QT独有参数
set(CMAKE_AUTOUIC ON)  # 启用自动UI编译（AUTOUIC）
set(CMAKE_AUTOMOC ON)  # 自动元对象编译（AUTOMOC）
set(CMAKE_AUTORCC ON)  # 自动资源编译（AUTORCC）
```

```
# CMAKE_AUTOUIC参数用于启用自动处理Qt User Interface (UI) 文件的功能。
# 当我们使用Qt进行GUI应用程序开发时，通常会创建UI文件（以.ui扩展名结尾），这些文件描述了用户界面的布局和组件。
# 为了将UI文件与应用程序的代码连接起来，我们需要将它们转换为相应的C++代码。通过将CMAKE_AUTOUIC设置为 ON ，CMake将自动在构建过程中查找并处理项目中的UI文件。
# 具体而言，当我们启用CMAKE_AUTOUIC后，CMake会在构建过程中自动调用uic工具（UI编译器），将UI文件转换为对应的C++代码，并将生成的C++文件添加到构建系统中，
# 以便编译器将其编译为最终的可执行文件。这样，我们就不需要手动执行uic工具来转换UI文件，CMake会在构建时自动完成这个步骤，简化了项目的配置和构建过程。

# CMAKE_AUTOMOC参数用于启用自动处理启用自动处理Qt元对象编译（MOC）的功能。
# Qt的元对象编译器（Meta Object Compiler，MOC）是一个工具，用于处理Qt中的特殊C++扩展，例如信号和槽、动态属性和反射机制。
# MOC会解析源代码中的这些扩展，并生成额外的C++代码，用于支持这些特性的运行时行为。
# 当我们使用到Qt进行应用程序开发时，一般会使用到信号和槽、Q_OBJECT宏或其他需要MOC处理的Qt特性，我们需要确保这些代码被MOC处理后再进行编译。
# 通过将CMAKE_AUTOMOC设置为ON，我们告诉CMake自动查找需要MOC处理的源文件，并在构建过程中自动调用MOC来生成相应的额外C++代码。
# 具体而言，启用CMAKE_AUTOMOC后，CMake会在构建过程中自动检测源文件中的需要MOC处理的特殊Qt扩展，并为这些源文件生成对应的MOC输出文件。
# 然后，这些生成的MOC输出文件将被添加到构建系统中，以便编译器将其编译为最终的可执行文件。

# CMAKE_AUTORCC参数用于启用自动处理Qt资源文件的功能。
# Qt资源文件（.qrc）是一种用于将非代码资源（如图像、字体、样式表等）集成到应用程序中的方式。
# 资源文件可以在运行时被动态加载和使用，使应用程序的资源管理更加方便和灵活。
# 通过将CMAKE_AUTORCC设置为ON，我们将告诉CMake自动查找并处理项目中的Qt资源文件。
# 具体而言，启用CMAKE_AUTORCC后，CMake会在构建过程中自动查找项目中的资源文件（.qrc文件），并调用rcc工具（资源编译器）来将这些资源文件编译为二进制数据。
# 然后，生成的二进制资源数据将被添加到构建系统中，以便在应用程序中使用。
# 这样，我们就无需手动调用rcc工具来处理资源文件，CMake会在构建时自动处理这些步骤，简化了项目的配置和构建过程。
# 当我们在代码中使用Qt的资源管理相关函数（如QResource）时，这些资源数据将被加载和使用。

# 设置C++标准为C++14，并要求编译器支持此标准。
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# 查找并加载外部依赖包
find_package(QT NAMES Qt6 Qt5 COMPONENTS Widgets LinguistTools REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Widgets LinguistTools REQUIRED)

# QT翻译资源文件
set(TS_FILES Example01_zh_CN.ts)

# 导入项目文件
set(PROJECT_SOURCES
        main.cpp
        mainwindow.cpp
        mainwindow.h
        mainwindow.ui
        ${TS_FILES}
)

# 如果Qt的主要版本号大于等于6，则使用"qt_add_executable"来创建可执行文件。否则，根据目标平台选择创建动态库（Android）或可执行文件。
if(${QT_VERSION_MAJOR} GREATER_EQUAL 6)
    qt_add_executable(Example01
        MANUAL_FINALIZATION
        ${PROJECT_SOURCES}
    )
    qt_create_translation(QM_FILES ${CMAKE_SOURCE_DIR} ${TS_FILES})
else()
    if(ANDROID)
        add_library(Example01 SHARED
            ${PROJECT_SOURCES}
        )
    else()
        add_executable(Example01
            ${PROJECT_SOURCES}
        )
    endif()

    qt5_create_translation(QM_FILES ${CMAKE_SOURCE_DIR} ${TS_FILES})
endif()

# 将Qt模块链接到目标可执行文件或动态库。
target_link_libraries(Example01 PRIVATE Qt${QT_VERSION_MAJOR}::Widgets)

# 设置目标可执行文件的属性，如MacOSX的Bundle标识符、版本号和短版本字符串，以及在Windows下作为可执行文件运行。
set_target_properties(Example01 PROPERTIES
    MACOSX_BUNDLE_GUI_IDENTIFIER my.example.com
    MACOSX_BUNDLE_BUNDLE_VERSION ${PROJECT_VERSION}
    MACOSX_BUNDLE_SHORT_VERSION_STRING ${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION_MINOR}
)

# 安装目标可执行文件到指定的目录。
install(TARGETS Example01
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)

# 如果使用的是Qt 6，则使用"qt_finalize_executable"进行最后的可执行文件处理。
if(QT_VERSION_MAJOR EQUAL 6)
    qt_finalize_executable(Example01)
endif()
```

五、运行HelloQt程序

1. 如下图所示，将debug模式修改成Release模式。



2. 点击下方绿色运行按钮，看到如下界面表示成功。