搜索

CMake 入门

CMake是一组允许构建、测试和打包应用程序的工具。就像Qt一样，它可以在所有主要开发平台上使用。它也受到各种IDE的支持，包括Qt Creator。

在本节中，我们将展示在CMake项目中使用Qt的最基本方法。首先，我们创建一个基本的控制台应用程序。然后，我们将项目扩展到使用Qt Widgets的GUI应用程序中。

如果您想知道如何使用 Qt 构建现有的 CMake 项目，请参阅有关如何在命令行上使用 CMake 构建项目的文档。

构建C++控制台应用程序

项目由用 CMake 语言编写的文件定义。主文件被调用，通常与实际程序源放在同一个目录中。CMakeCMakeLists.txt

下面是使用 Qt 用 C++ 编写的控制台应用程序的典型文件：CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)

project(helloworld VERSION 1.0.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS Core)
qt_standard_project_setup()

add_executable(helloworld
    main.cpp
)

target_link_libraries(helloworld PRIVATE Qt6::Core)
```

让我们来看看内容。

```
cmake_minimum_required(VERSION 3.16)
```

cmake_minimum_required()指定应用程序所需的最低 CMake 版本。Qt本身至少需要CMake版本3.16。如果您使用静态构建的Qt（iOS版Qt和WebAssembly版Qt中的默认值），则需要CMake 3.21.1或更高版本。

```
project(helloworld VERSION 1.0.0 LANGUAGES CXX)
```

project()设置项目名称和默认项目版本。Theargument告诉CMake该程序是用C++编写的。LANGUAGES

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

Qt 6 需要支持 C++ 版本 17 或更高版本的编译器。通过设置变量来强制实施此操作，如果编译器太旧，CMake 将打印错误。
CMAKE_CXX_STANDARDCMAKE_CXX_STANDARD_REQUIRED

这告诉CMake查找Qt 6，并导入模块。如果找不到模块，继续下去是没有意义的，所以我们确实设置了标志，让 CMake 在这种情况下中止。CoreCMakeREQUIRED

如果成功，模块将设置模块变量中记录的一些 CMake 变量。它还导入了我们在下面使用的目标。Qt6::Core

要想成功，必须找到Qt安装。有不同的方法可以告诉Qt，但最常见和推荐的方法是设置CMake缓存变量以包含Qt 6安装前缀。请注意，Qt Creator将为您透明地处理这个问题。find_packageCMakeCMakeCMAKE_PREFIX_PATH

```
qt_standard_project_setup()
```

qt_standard_project_setup命令为典型的 Qt 应用程序设置项目范围的默认值。

除此之外，此命令将变量设置为 ，指示 CMake 自动设置规则，以便在需要时透明地调用 Qt 的元对象编译器 （moc）。CMAKE_AUTOMOCON

有关详细信息，请参阅qt_standard_project_setup的参考资料。

```
add_executable(helloworld
    main.cpp
)
```

add_executable()告诉 CMake 我们想要构建一个名为目标的可执行文件（所以不是库）。应从C++源文件生成目标。helloworldmain.cpp

请注意，您通常不会在此处列出头文件。这与 qmake 不同，在qmake 中，头文件需要显式列出，以便由元对象编译器 （moc） 处理它们。

对于不太琐碎的项目，您可能需要调用qt_add_executable （） 。它是内置命令的包装器，提供额外的逻辑来自动处理诸如静态Qt构建中的Qt插件链接，特定于平台的库名称自定义等事情。add_executable()

有关创建库的信息，请参阅qt_add_library。

```
target_link_libraries(helloworld PRIVATE Qt6::Core)
```

最后，告诉CMake，可执行文件通过引用上面调用导入的目标来利用Qt Core。这不仅会将正确的参数添加到链接器，还可以确保将正确的包含目录、编译器定义传递给C++编译器。关键字对于可执行目标不是绝对必需的，但最好指定它。如果是一个库而不是可执行文件，那么应该指定 bothor（如果库在其标头中提到任何内容，否则）。target_link_librarieshelloworldQt6::Corefind_package()PRIVATEhelloworldPRIVATEPUBLICPUBLICQt6::CorePRIVATE

构建C++ GUI 应用程序

在上一节中，我们展示了一个简单的控制台应用程序的 CMakeLists.txt 文件。我们现在将扩展它以创建一个使用Qt Widgets模块的 GUI 应用程序。

这是完整的项目文件：

```
cmake_minimum_required(VERSION 3.16)

project(helloworld VERSION 1.0.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS Widgets)
qt_standard_project_setup()

add_executable(helloworld
    mainwindow.ui
    mainwindow.cpp
    main.cpp
)

target_link_libraries(helloworld PRIVATE Qt6::Widgets)

set_target_properties(helloworld PROPERTIES
    WIN32_EXECUTABLE ON
    MACOSX_BUNDLE ON
)
```

```
find_package(Qt6 REQUIRED COMPONENTS Widgets)
```

在调用中，我们替换。这将找到模块并提供我们稍后链接的目标。`find_package`CoreWidgetsQt6WidgetsQt6::Widgets

请注意，应用程序仍将链接到它，因为依赖于它。Qt6::CoreQt6::Widgets

```
qt_standard_project_setup()
```

除了，qt_standard_project_setup将变量设置为。这将自动创建规则来调用Qt的用户界面编译器（uic）onsource文件。
CMAKE_AUTOMOCCMAKE_AUTOUICON.ui

```
add_executable(helloworld
    mainwindow.ui
    mainwindow.cpp
    main.cpp
)
```

我们将Qt Designer文件（）及其相应的C++源文件（）添加到应用程序目标的源中。mainwindow.uimainwindow.cpp

```
target_link_libraries(helloworld PRIVATE Qt6::Widgets)
```

在命令中，我们链接反对。target_link_librariesQt6::WidgetsQt6::Core

```
set_target_properties(helloworld PROPERTIES
    WIN32_EXECUTABLE ON
    MACOSX_BUNDLE ON
)
```

最后，我们在应用程序目标上设置属性，效果如下：

- 阻止在 Windows 上创建控制台窗口。
- 在 macOS 上创建应用程序捆绑包。

有关这些目标属性的详细信息，请参阅CMake 文档。

构建项目

包含多个目标的项目将受益于清晰的项目文件结构。我们将使用 CMake 的子目录功能。

当我们计划用更多目标扩展项目时，我们将应用程序的源文件移动到一个子目录中，并在其中创建一个 newin。CMakeLists.txt

```
<project root>
├── CMakeLists.txt
└── src
    └── app
        ├── CMakeLists.txt
        ├── main.cpp
        ├── mainwindow.cpp
        ├── mainwindow.h
        └── mainwindow.ui
```

顶层包含整个项目设置，并调用：CMakeLists.txtfind_packageadd_subdirectory

```
cmake_minimum_required(VERSION 3.16)
```

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS Widgets)
qt_standard_project_setup()

add_subdirectory(src/app)
```

在此文件中设置的变量在子目录项目文件中可见。

应用程序的项目文件包含可执行目标：src/app/CMakeLists.txt

```
add_executable(helloworld
    mainwindow.ui
    mainwindow.cpp
    main.cpp
)

target_link_libraries(helloworld PRIVATE Qt6::Widgets)

set_target_properties(helloworld PROPERTIES
    WIN32_EXECUTABLE ON
    MACOSX_BUNDLE ON
)
```

这样的结构将很容易向项目添加更多目标，例如库或单元测试。

构建库

随着项目的增长，您可能希望将部分应用程序代码转换为供应用程序使用的库，并可能使用单元测试。本节介绍如何创建此类库。

我们的应用程序当前直接包含业务逻辑。我们将代码提取到子目录中调用的新静态库中，如上一节所述。
main.cppbusinesslogic"src/businesslogic"

为简单起见，该库仅包含一个C++源文件及其相应的头文件，该文件包含在应用程序的：main.cpp

```
<project root>
├── CMakeLists.txt
└── src
    ├── app
    │   ├── ...
    │   └── main.cpp
    └── businesslogic
        ├── CMakeLists.txt
        ├── businesslogic.cpp
        └── businesslogic.h
```

让我们看一下库的项目文件 （） 。src/businesslogic/CMakeLists.txt

```
add_library(businesslogic STATIC
    businesslogic.cpp
)
target_link_libraries(businesslogic PRIVATE Qt6::Core)
target_include_directories(businesslogic INTERFACE ${CMAKE_CURRENT_SOURCE_DIR})
```

让我们来看看内容。

```
add_library(businesslogic STATIC
    businesslogic.cpp
)
```

add_library命令创建库。稍后，我们将让应用程序针对此目标进行链接。businesslogic

我们有一个静态库，实际上不必链接其他库。但是由于我们的库使用类，我们添加了一个链接依赖项。这将拉入必要的包含路径和预处理器定义。
QtCoreQt6::CoreQtCore

```
target_include_directories(businesslogic INTERFACE ${CMAKE_CURRENT_SOURCE_DIR})
```

库 API 在头文件中定义。通过调用target_include_directories，我们确保目录的绝对路径自动添加为使用我们的库的所有目标的包含路径。
businesslogic/businesslogic.hbusinesslogic

这使我们不必使用相对路径进行定位。相反，我们可以写main.cppbusinesslogic.h

```
#include <businesslogic.h>
```

最后，我们必须将库的子目录添加到顶级项目文件中：

```
add_subdirectory(src/app)
add_subdirectory(src/businesslogic)
```

使用库

要使用我们在上一节中创建的库，我们指示 CMake 链接到它：

```
target_link_libraries(helloworld PRIVATE
    businesslogic
    Qt6::Widgets)
```

这可确保在编译 main.cpp 时找到它。此外，业务逻辑静态库将成为可执行文件的一部分。businesslogic.hhelloworld

在 CMake 术语中，库指定了我们库（应用程序）的每个使用者都必须满足的使用*要求*（包含路径）。命令负责处理。
businesslogictarget_link_libraries

添加资源

我们希望在应用程序中显示一些图像，因此我们使用Qt资源系统添加它们。

```
qt_add_resources(helloworld imageresources
    PREFIX "/images"
    FILES logo.png splashscreen.png
)
```

qt_add_resources命令会自动创建包含引用图像的Qt资源。从C++源代码中，您可以通过在指定的资源前缀前面加上前缀来访问图像：

```
logoLabel->setPixmap(QPixmap(":/images/logo.png"));
```

qt_add_resources命令将变量名称或目标名称作为第一个参数。我们建议使用此命令的基于目标的变体，如上例所示。

添加翻译

Qt项目中字符串的翻译是编码在文件中的。有关详细信息，请参阅Qt 的国际化。.ts

若要将文件添加到项目中，请使用qt_add_translations命令。.ts

以下示例将德语和法语翻译文件添加到目标：helloworld

```
    TS_FILES helloworld_de.ts helloworld_fr.ts)
```

这将创建构建系统规则以从文件自动生成文件。默认情况下，这些文件嵌入到资源中，并且可以在 theresource 前缀下访问。`.qm.ts.qm"/i18n"`

要更新文件中的条目，请构建目标：`.tsupdate_translations`

```
$ cmake --build . --target update_translations
```

要手动触发文件的生成，请构建目标：`.qmrelease_translations`

```
$ cmake --build . --target release_translations
```

有关如何影响文件处理和资源嵌入的详细信息，请参阅qt_add_translations文档。`.ts`

qt_add_translations命令是一个方便的包装器。若要更精细地控制文件和文件的处理方式，请使用基础命令qt_add_lupdate和qt_add_lrelease。`.ts.qm`

延伸阅读

官方CMake 文档是使用 CMake 的宝贵资源。

官方CMake 教程涵盖了常见的构建系统任务。

《专业 CMake：实用指南》一书很好地介绍了最相关的 CMake 功能。

‹ 使用 CMake 构建

在命令行上生成项目 ›

下载
市场

反馈 登录