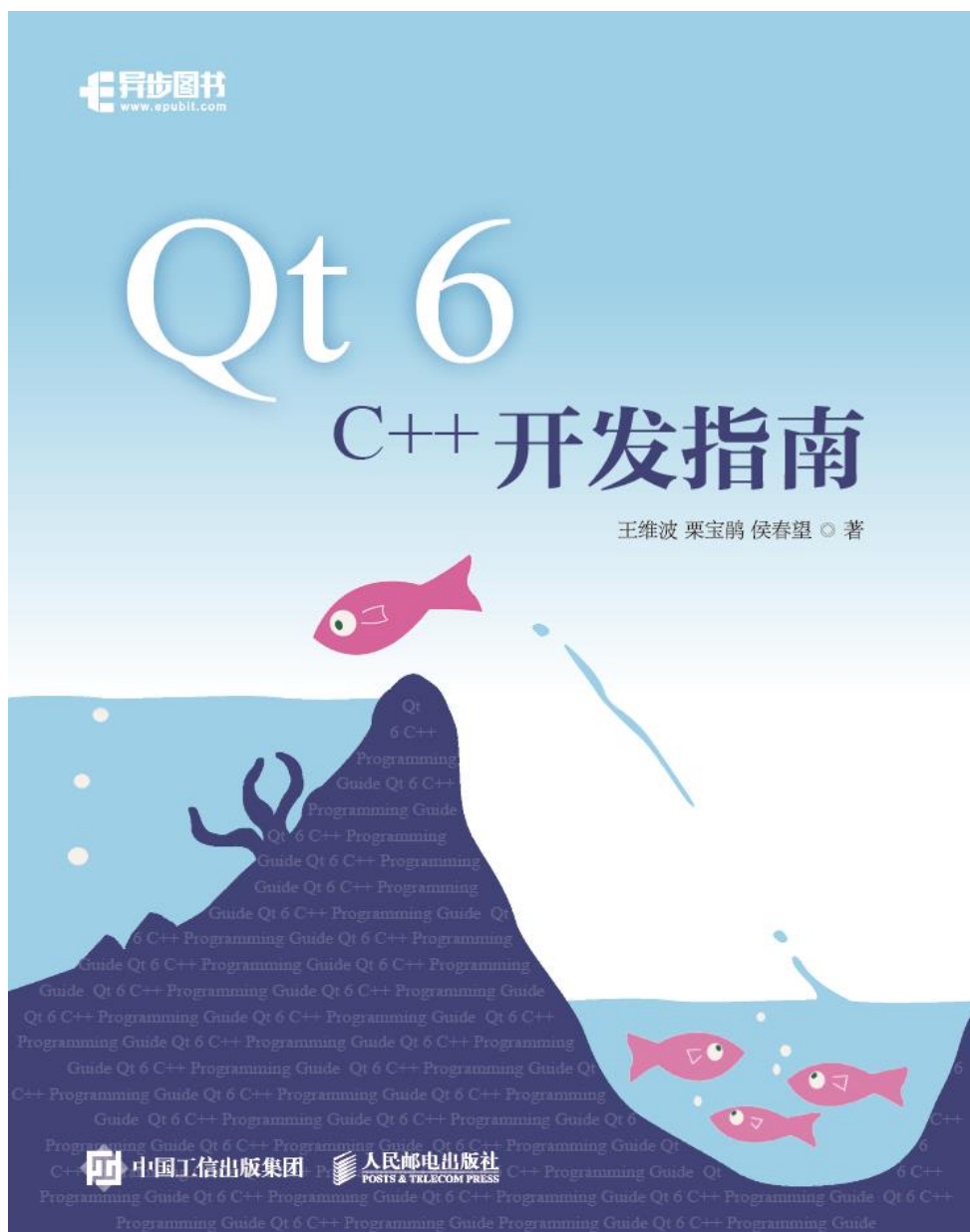


## CMake 管理项目使用方法汇总



王维波

2023-3-19

# 说 明

《Qt 6 C++开发指南》全书的内容是按照 qmake 管理项目来编写的，只是在 2.4 节介绍了 CMake 管理项目的方法和示例。所以，本书最初提供的全书示例源码几乎都是用 qmake 管理项目。《Qt 6 C++开发指南》之所以采用 qmake 介绍全书的示例，主要是基于如下的考虑。

- (1) qmake 是一个简单易用的构建系统，而 CMake 比较复杂，适合于管理大型的软件系统。对于初学者来说，使用 qmake 学习 Qt 编程更容易一些。本书的示例一般比较简单，没有很多的文件和层次结构，使用 qmake 足以，对初学者更友好一些。
- (2) qmake 已经存在了很多年，与 Qt 配合使用非常成熟。很多 Qt 的使用者可能基于使用习惯或向后兼容的考虑，还是愿意用 qmake，而不是轻易转到 CMake。
- (3) 本书的主要目的和内容是介绍 Qt 开发框架中各种架构、模块和类的编程使用方法。CMake 只是一个构建系统，不是本书的重点。将一般的 qmake 项目转换为 CMake 项目是很容易的事情。

但是 CMake 是技术发展趋势，部分读者也想学习使用 CMake 管理 Qt 的项目。所以，本书又提供了全书示例的 CMake 版本。对于 CMake 版本的源程序，需要注意以下问题。

- (1) 这些示例中只用到了 CMake 的一些基本功能，读者不要指望从中学到 CMake 的所有技巧，这不是本书的重点。要学习 CMake 的所有技术，应该研读 CMake 的专业文档或专门介绍 CMake 的书籍。
- (2) Qt 中某些类型的项目目前还只能用 qmake 管理，而不能用 CMake。如 11.2 节涉及到的 Qt Designer Widget 插件项目只能是 qmake 项目，18.3 节涉及的 Qt Installer Framework 安装项目只能是 qmake 项目。

本文档是针对本书示例源代码的 CMake 完整版整理的 CMake 使用方法汇总，以及一些示例的使用注意事项，只涉及到本书中的示例用到的 CMake 的一些功能。搞懂本书示例中用到的这些 CMake 功能后，用 CMake 管理一般的 Qt 项目就没有问题了。

# 第 2 章 GUI 程序设计基础

## 2.2 可视化 UI 设计

示例 samp2\_2 中涉及到使用资源文件、为应用程序设置图标等问题，文件 CMakeLists.txt 中的相关操作描述如下。

### 1. 创建和使用资源文件

(1) 使用新建文件或项目向导，创建一个 Qt 资源文件，例如 res.qrc。

(2) 编辑文件 CMakeLists.txt，将文件 res.qrc 添加到项目的源文件列表，如下所示。

```
set (PROJECT_SOURCES
    main.cpp
    dialog.cpp
    dialog.h
    dialog.ui
    res.qrc      #添加资源文件
)
```

保存 CMakeLists.txt 文件后，CMake 会把文件 res.rc 添加到项目节点下，如图 2-1 所示。

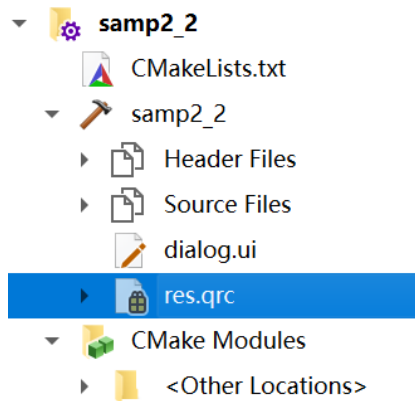


图 2-1

(3) 在 res.qrc 节点的右键快捷菜单中点击 Open in Editor，就可以编辑资源文件 res.qrc，为其添加图标等资源。在编辑 dialog.ui 文件时，就可以使用资源文件 res.qrc 中的图标。

### 2. 为应用程序设置图标

(1) 把为应用程序准备的图标文件 editor.ico 复制到 CMakeLists.txt 同级的目录下，也就是项目的根目录下。

(2) 使用新建文件或项目向导，选择 **General** 组里的 **empty file**，创建的文件命名为 **appIcon.rc**。注意，文件名可以自由设置，但文件名的后缀需要是 **rc**。在文件 **appIcon.rc** 中输入如下的内容

```
IDI_ICON1 ICON DISCARDABLE "editor.ico"
```

这是声明了一个图标资源 **IDI\_ICON1**，图标文件是 **editor.ico**。这属于 Windows API 编程的范畴。

(3) 编辑文件 **CMakeLists.txt**，将文件 **appIcon.rc** 添加到项目的源文件列表，如下所示。

```
set(PROJECT_SOURCES
    main.cpp
    dialog.cpp
    dialog.h
    dialog.ui
    res.qrc
    appIcon.rc      #添加应用程序图标资源文件
)
```

保存 **CMakeLists.txt** 文件后，CMake 会把文件 **appIcon.rc** 添加到 **Source Files** 分组里，如图 2-2 所示。

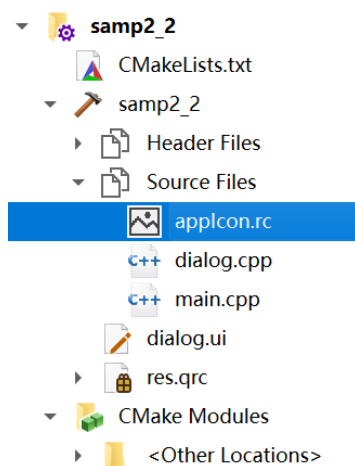


图 2-2

(4) 构建项目，生成的可执行文件 **smp2\_2.exe** 就具有了应用程序图标，如图 2-3 所示。

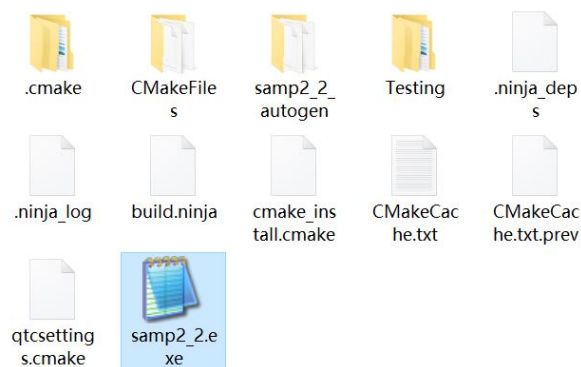


图 2-3

# 第 3 章 Qt 框架功能概述

## 向项目中添加文件

示例 samp3\_1 中需要创建 TPerson 类，创建后生成文件 tperson.h 和 tperson.cpp，但是这两个文件不会被自动添加到项目 samp3\_1 中，需要修改文件 CMakeLists.txt 的内容，手动添加这两个文件。

编辑文件 CMakeLists.txt，在如下的语句中添加文件 tperson.h 和 tperson.cpp。

```
set (PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
    #手动添加项目用到的文件
    tperson.h
    tperson.cpp
)
```

保存文件 CMakeLists.txt 后，文件 tperson.h 和 tperson.cpp 会被添加到项目的目录树中，如图 3-1 所示。

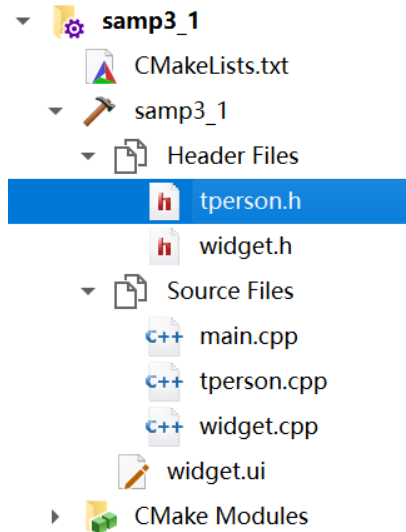


图 3-1

在其他项目中，向项目中添加源文件都是如此操作，包括添加 UI 文件。

# 第 4 章 常用界面组件的使用

## 定义 QT\_NO\_DEBUG\_OUTPUT, 禁止 qDebug()输出

函数 qDebug()通常用于在调试阶段输出一些信息,示例 samp4\_03 和 samp4\_07 中就用到 qDebug()。在发布 Release 版本程序时,可以通过项目的设置禁止 qDebug()的输出,而不用修改源程序。

在使用 qmake 管理项目时,在.pro 文件中加入如下的语句

```
DEFINES += QT_NO_DEBUG_OUTPUT
```

也就是定义符号 QT\_NO\_DEBUG\_OUTPUT,那么程序中的 qDebug()函数就无法输出信息,否则, qDebug()输出的信息会在 Qt Creator 的 Application Output 子窗口显示。

在使用 CMake 管理项目时,在文件 CMakeLists.txt 中加入如下的语句,可以禁止 qDebug()输出信息。

```
target_compile_definitions(samp4_07 PRIVATE QT_NO_DEBUG_OUTPUT)
```

若需要显示 qDebug()输出的信息,注释掉这行语句即可。

# 第 9 章 数据库

本章的示例项目都要用到 QtSQL 模块,需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Sql REQUIRED)
```

```
target_link_libraries(samp9_1 PRIVATE Qt${QT_VERSION_MAJOR}::Sql)
```

其中, target\_link\_libraries()中的第一个参数是示例项目名称,需要根据具体的项目名称修改。

# 第 10 章 绘图

## 10.4 图像处理

示例 samp10\_6 中用到打印功能,需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS PrintSupport REQUIRED)
```

```
target_link_libraries(samp10_6 PRIVATE Qt${QT_VERSION_MAJOR}::PrintSupport)
```

# 第 11 章 自定义插件和库

## 11.2 设计和使用 Qt Designer Widget

### 示例 BatteryPlugin

创建 Qt Custom Designer Widget 项目时，生成的项目类型自动使用 qmake 构建系统，不能使用 CMake 项目。所以，示例项目 BatteryPlugin 就是 qmake 项目，与书中介绍的完全一样。

但是要注意，创建此项目时开发套件(kit)只能选择 MSVC2019 64bit，也就是必须与构建 Qt Creator 的开发套件一致。

### 示例 BatteryUser

创建这个项目时可以选择使用 CMake 构建系统，使用时要注意以下事项。

- (1) 只能使用 MSVC2019 64bit 开发套件。
- (2) 在项目的根目录下创建文件夹 include，将项目 BatteryPlugin 中的文件 tpbattery.h 和 tpbatteryplugin.lib 复制到此文件夹下，如图 11-1 所示。

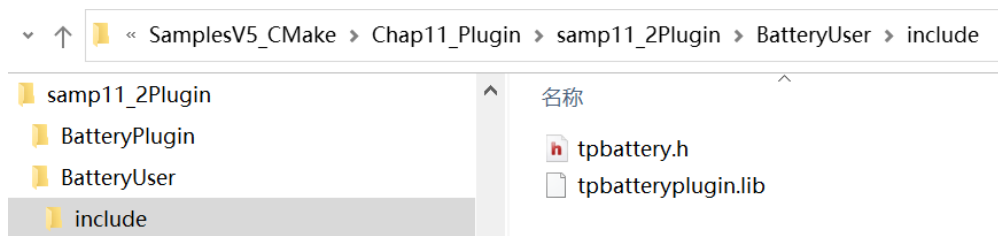


图 11-1

- (3) 编辑文件 CMakeLists.txt，添加头文件和库文件的搜索路径，也就是项目源程序目录下的子文件夹 include

```
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)    #设置头文件搜索路径
LINK_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)      #设置需要连接的库文件搜索路径
```

在文件 CMakeLists.txt 中再添加需要连接的动态库的名称，即 tpbatteryplugin.dll

```
target_link_libraries(BatteryUser PUBLIC tpbatteryplugin)    #添加需要连接的动态库
```

- (4) 构建此项目生成可执行文件 BatteryUser.exe，将示例项目 BatteryPlugin 中生成的动态链接库文件 tpbatteryplugin.dll 复制到可执行文件 BatteryUser.exe 所在的文件夹，才可以正常运行项目，否则提示找不到 DLL 文件。

注意，这些修改是针对 Release 模式的，使用的都是 Release 版本的 lib 和 dll 文件。

## 11.3 创建和使用静态库

### 示例 MyStaticLib

创建静态库项目 MyStaticLib 时可以选择 CMake 构建系统，创建过程的其他操作见书中解释。

创建项目后会自动生成 CMakeLists.txt 文件。文件 CMakeLists.txt 中，自动生成的与创建静态库相关的代码如下：

```
#add_library 用于生成静态库或动态库，STATIC 表示静态库
add_library(MyStaticLib STATIC
    tpendialog.cpp
    tpendialog.h
    tpendialog.ui
)
```

add\_library()使用指定的源文件生成静态库、动态库、模块等。第 1 个参数是库名称，第 2 个参数表示生成的库的类型，有 STATIC、SHARED、MODULE 等类型。后面的参数是生成库的源程序文件，可以设置多个需要的文件。

本项目使用 MSVC2019 64-bit 开发套件，以 Release 模式构建项目。构建项目后生成静态库文件 MyStaticLib.lib。

### 示例 StaticLibUser

创建此项目可以选择使用 CMake 构建系统，创建过程的其他操作见书中解释。选择 MSVC2019 64-bit 开发套件，并配置为 Release 模式。其他按如下步骤操作。

(1) 在项目文件夹下创建子文件夹 include，将项目 MyStaticLib 中的文件 tpendialog.h，以及编译生成的库文件 MyStaticLib.lib 复制到此文件夹里，如图 11-2 所示。

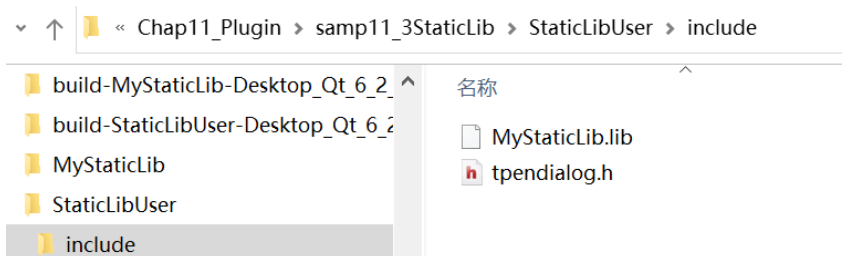


图 11-2

(2) 修改文件 CMakeLists.txt，加入如下的语句，各语句功能见注释

```
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)    #设置头文件搜索路径
LINK_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)      #设置需要连接的库文件搜索路径

target_link_libraries(StaticLibUser PRIVATE MyStaticLib) #添加需要链接的静态库
```



无需将文件 `tpendialog.h` 添加到项目的源文件列表。

(3) 以 **Release** 模式构建项目，即可运行项目。

## 11.4 创建和使用共享库

### 示例 MySharedLib

创建项目 `MySharedLib` 时可以选择 **CMake** 构建系统，创建过程的其他操作见书中解释。

创建项目后会自动生成 `CMakeLists.txt` 文件。文件 `CMakeLists.txt` 中，自动生成的与创建共享库相关的代码如下：

```
#add_library 用于生成静态库或动态库，SHARED 表示动态库（共享库）
add_library(MySharedLib SHARED
    MySharedLib_global.h
    tpendialog.cpp
    tpendialog.h
    tpendialog.ui
)
```

`add_library()` 中的第 2 个参数是 **SHARED**，表示生成共享库。使用 **MSVC2019** 开发套件时，生成的就是 **DLL** 文件。

本项目使用 **MSVC2019 64-bit** 开发套件，以 **Release** 模式构建项目。构建项目后生成文件 `MySharedLib.dll` 和 `MySharedLib.lib`。

### 示例 SharedLibUser

创建此项目可以选择使用 **CMake** 构建系统，创建过程的其他操作见书中解释。选择 **MSVC2019 64-bit** 开发套件，并配置为 **Release** 模式。其他按如下步骤操作。

(1) 在项目文件夹下创建子文件夹 `include`，将项目 `MySharedLib` 中的 3 个文件复制到此目录下，如图 11-3 所示。

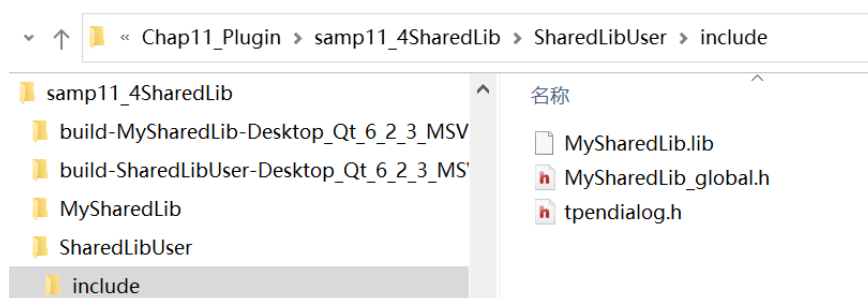


图 11-3

(2) 编辑文件 `CMakeLists.txt`，加入如下的语句，各语句功能见注释

```
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)    #设置头文件搜索路径
LINK_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)      #设置需要连接的库文件搜索路径

target_link_libraries(SharedLibUser PRIVATE MySharedLib)  #添加需要链接的动态库
```

子文件夹 `include` 中的两个头文件无需添加到项目的源文件列表。

(3) 以 `Release` 模式构建项目，生成可执行文件 `SharedLibUser.exe`。将动态库文件 `MySharedLib.dll` 复制到可执行文件 `SharedLibUser.exe` 所在的文件夹，才可以正常运行项目，否则提示找不到 `DLL` 文件。

## 第 12 章 Qt Charts

本章的示例项目都要使用 Charts 模块，需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Charts REQUIRED)

target_link_libraries(samp12_1 PRIVATE Qt${QT_VERSION_MAJOR}::Charts)
```

其中，target\_link\_libraries()中的第一个参数是示例项目名称，需要根据具体的项目名称修改。

## 第 13 章 Qt Data Visualization

本章的示例项目都要使用 Data Visualization 模块，需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS DataVisualization REQUIRED)

target_link_libraries(samp13_1 PRIVATE Qt${QT_VERSION_MAJOR}::DataVisualization)
```

其中，target\_link\_libraries()中的第一个参数是示例项目名称，需要根据具体的项目名称修改。

## 第 15 章 网络

本章的示例项目都使用 Network 模块，需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Network REQUIRED)

target_link_libraries(samp15_1 PRIVATE Qt${QT_VERSION_MAJOR}::Network)
```

其中，target\_link\_libraries()中的第一个参数是示例项目名称，需要根据具体的项目名称修改。

## 第 16 章 多媒体

本章所有示例都要使用 Multimedia 模块，需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Multimedia REQUIRED)

target_link_libraries(samp16_1 PRIVATE Qt${QT_VERSION_MAJOR}::Multimedia)
```

其中，target\_link\_libraries()中的第一个参数是示例项目名称，需要根据具体的项目名称修改。

### 示例 samp16\_2

对于此示例，需要将项目源文件目录下的 **sound** 文件夹复制到编译后生成的可执行文件所在的文件夹，也就是作为 **samp16\_2.exe** 所在目录的子目录，因为 **samp16\_2.exe** 在运行时需要插在子目录 **sound** 里的图片和声音文件。

## 示例 samp16\_4

这个示例要用到 **Charts** 和 **Multimedia** 两个模块，所以在文件 **CMakeLists.txt** 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Multimedia REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Charts REQUIRED)

target_link_libraries(samp16_4 PRIVATE Qt${QT_VERSION_MAJOR}::Multimedia)
target_link_libraries(samp16_4 PRIVATE Qt${QT_VERSION_MAJOR}::Charts)
```

## 示例 samp16\_5、samp16\_6 和 samp16\_7

这几个示例中需要用到 **Multimedia** 和 **Multimedia Widgets** 两个模块，所以在文件 **CMakeLists.txt** 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Multimedia REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS MultimediaWidgets REQUIRED)

target_link_libraries(samp16_5 PRIVATE Qt${QT_VERSION_MAJOR}::Multimedia)
target_link_libraries(samp16_5 PRIVATE Qt${QT_VERSION_MAJOR}::MultimediaWidgets)
```

注意，**target\_link\_libraries()**要使用具体的项目名称，如 **samp16\_5** 和 **samp16\_6**。

# 第 17 章 串口编程

本章的示例要使用 Serial Port 模块，需要在文件 CMakeLists.txt 中加入如下的语句

```
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS SerialPort REQUIRED)

target_link_libraries(samp17_1 PRIVATE Qt${QT_VERSION_MAJOR}::SerialPort)
```

# 第 18 章 其他工具软件和技术

## 示例 Demo18\_1

编译项目后，需要将项目根目录下的文件 samp18\_1\_cn.qm 和 samp18\_1\_en.qm 复制到可执行文件 samp18\_1.exe 所在的目录下，因为程序运行时需要加载这两个文件。

## 示例 Demo18\_4

Qt Install Framework 的安装项目只有 qmake 项目，没有 CMake 项目。