qmake 入门

本教程将教您 qmake 的基础知识。本手册中的其他主题包含有关使用 qmake 的更多详细信息。

从简单开始

假设您刚刚完成了应用程序的基本实现，并且已经创建了以下文件：

- 你好.cpp
- 你好.h
- 主要.cpp

您将在 Qt 发行版的目录中找到这些文件。关于应用程序的设置，您唯一知道的另一件事是它是用Qt编写的。首先，使用您喜欢的纯文本编辑器，创建一个名为 的文件。您需要做的第一件事是添加一行，告诉 qmake 有关开发项目一部分的源文件和头文件的信息。
`examples/qmake/tutorialhello.proexamples/qmake/tutorial`

我们将首先将源文件添加到项目文件中。为此，您需要使用源变量。只需开始一个新行，并在它之后输入你好.cpp。你应该有这样的东西：`SOURCES +=`

```
SOURCES += hello.cpp
```

我们对项目中的每个源文件重复此操作，直到我们最终得到以下内容：

```
SOURCES += hello.cpp
SOURCES += main.cpp
```

如果您更喜欢使用类似Make的语法，并且一次性列出所有文件，则可以使用换行符转义，如下所示：

```
SOURCES = hello.cpp \
          main.cpp
```

现在源文件已在项目文件中列出，必须添加头文件。这些变量的添加方式与源文件完全相同，只是我们使用的

```
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

目标名称是自动设置的。它与项目文件名相同，但具有适合平台的后缀。例如，如果项目文件被调用，则目标将在窗口和 Unix 上。如果要使用其他名称，可以在项目文件中进行设置：`hello.prohello.exehello`

```
TARGET = helloworld
```

完成的项目文件应如下所示：

```
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

现在，您可以使用 qmake 为您的应用程序生成生成文件。在命令行的项目目录中，键入以下内容：

```
qmake -o Makefile hello.pro
```

注意： 如果您通过包管理器安装了 Qt，则二进制文件可能是 。`qmake6`

然后键入 或取决于您使用的编译器。`makenmake`

对于视觉工作室用户，qmake 还可以生成可视化工作室项目文件。例如：

```
qmake -tp vc hello.pro
```

Making an Application Debuggable

The release version of an application does not contain any debugging symbols or other debugging information. During development, it is useful to produce a debugging version of the application that has the relevant information. This is easily achieved by adding to the CONFIG variable in the project file.`debug`

For example:

```
CONFIG += debug
HEADERS += hello.h
```

Use qmake as before to generate a Makefile. You will now obtain useful information about your application when running it in a debugging environment.

Adding Platform-Specific Source Files

After a few hours of coding, you might have made a start on the platform-specific part of your application, and decided to keep the platform-dependent code separate. So you now have two new files to include into your project file: and . We cannot just add these to the variable since that would place both files in the Makefile. So, what we need to do here is to use a scope which will be processed depending on which platform we are building for.`hellowin.cpp``hellounix.cpp``SOURCES`

A simple scope that adds the platform-dependent file for Windows looks like this:

```
win32 {
    SOURCES += hellowin.cpp
}
```

When building for Windows, qmake adds to the list of source files. When building for any other platform, qmake simply ignores it. Now all that is left to be done is to create a scope for the Unix-specific file.`hellowin.cpp`

When you have done that, your project file should look something like this:

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
```

Use qmake as before to generate a Makefile.

Stopping qmake If a File Does Not Exist

You may not want to create a Makefile if a certain file does not exist. We can check if a file exists by using the exists() function. We can stop qmake from processing by using the error() function. This works in the same way as scopes do. Simply replace the scope condition with the function. A check for a file called main.cpp looks like this:

```
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
```

The symbol is used to negate the test. That is, is true if the file exists, and is true if the file does not exist.`!exists( main.cpp )!exists( main.cpp )`

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
```

Use qmake as before to generate a makefile. If you rename temporarily, you will see the message and qmake will stop processing.`main.cpp`

Checking for More than One Condition

Suppose you use Windows and you want to be able to see statement output with when you run your application on the command line. To see the output, you must build your application with the appropriate console setting. We can easily put on the line to include this setting in the Makefile on Windows. However, let's say that we only want to add the line when we are running on Windows *and* when is already on the line. This requires using two nested scopes. First create one scope, then create the other inside it. Put the settings to be processed inside the second scope, like this:`qDebug()consoleCONFIGCONFIGdebugCONFIG`

```
win32 {
    debug {
        CONFIG += console
    }
}
```

Nested scopes can be joined together using colons, so the final project file looks like this:

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
```

Qt DOCUMENTATION

```
win32:debug {
    CONFIG += console
}
```

That's it! You have now completed the tutorial for qmake, and are ready to write project files for your development projects.

< Overview Creating Project Files >

The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum

DOCUMENTATION

Feedback Sign In