Qt5.9.1 qmake 中文手冊

來源：http://blog.csdn.net/qm843230255/article/details/77584969

qmake 手冊

qmake 工具有助於簡化跨平台項目的開發.它自動生成 Makefiles，只需要很少的信息就可以創建 Makefiles。無論是 Qt 項目或者是其他軟件項目，都可以使用 qmake。

qmake 通過項目文件(.pro)裡提供的信息自動生成 Makefiles.項目文件(.pro)由開發者創建,通常項目文件(.pro)都很簡單,大型復雜的項目也可以通過項目文件(.pro)創建。

qmake 包含的附加功能支持 Qt 跨平台開發,qmake 可以為 moc 和 uic 自動生成編譯規則。

qmake 可以在無需修改項目文件(.pro)的情況下自動生成 Microsoft Visual studio 項目。

目錄大綱

預覽

qmake 工具提供了一個管理應用程序，庫和其他組件的面向項目的系統。可以使你控制項目裡的所有源文件,並簡明描述了構建過程中每一步的信息。qmake 擴展了每個項目中在編譯和連接中運行必要命令的 Makefile 文件中的信息。

描述項目

Qt 項目通過項目文件(.pro)文件內容來描述這個項目。qmake 利用項目文件(.pro)中的信息生成 Makefiles，Makefiles 包含了編譯項目所需的所有命令。項目文件(.pro)通常包含源文件和頭文件的列表，通用的配置信息和特定應用程序細節, 例如外部庫鏈接列表,或者外部頭文件包含目錄列表等。

項目文件(.pro)可以包含許多不同類型的元素，包括注釋，變量聲明，內置函數和一些簡單的控制結構。一些最簡單的項目中, 只需聲明用於構建基本的配置選項的源文件和頭文件。更多關於如何創建一個簡單的項目文件，請參考快速入門。

你可以為更復雜的項目創建復雜的項目文件(.pro)。預覽項目文件，參見創建項目文件。關於項目文件(.pro)中使用變量和函數的詳細信息，請看參考.

你可以使用一個應用程序或者庫的項目模板，通過指定專門的配置來調整構建過程。 獲取更多信息請看構建通用項目.

你可以使用 Qt 新建項目向導，來新建項目文件(.pro)。擇項目模板,然後 Qt Creator 會創建一個帶默認參數的可以構建和運行的項目。你可以通過修改項目文件(.pro)來達到你的目的。

你也可以通過 qmake 生成項目文件(.pro)，獲取全部的 qmake 的描述和命令行參數，請參考運行 qmake.

基本配置特性的 qmake 可以處理絕大多數的跨平台項目。然而, 使用一些平台特性的變量是非常有效的，甚至的必須的。更多關於使用平台特性變量的信息，請參考平台說明.

構建項目

對於簡單項目，你只需在項目的根目錄運行 qmake 來生成 Makefile。你可以運行平台的 make 命令執行 Makefiles 來構建你的項目。

更多關於配置構建過程使用的環境變量的信息，請參考配置 qmake.

使用第三方庫

使用第三方庫的指南向你展示如何在你的 Qt 項目中使用簡單的第三方庫。

預編譯頭文件

在大型項目中，可以利用預編譯頭文件加快構建過程。更多信息請參考使用預編譯頭文件.

快速入門

本教程會介紹有關 qmake 的基礎知識，和使用 qmake 的詳細信息。

開始簡單實例

假設我們已經完成了一個應用程序的基本配置，並且創建了以下文件:

- hello.cpp
- hello.h
- main.cpp

這些文件存在於 examples/qmake/tutorial 文件夾中。你需要知道的另一件事情就是應用程序是由 Qt 編寫的。首先，使用你最喜歡的文本編輯器，新建一個文件名字叫 hello.pro 的文件在 examples/qmake/tutorial 文件夾中。 接下來你需要添加內容告訴 qmake 那些頭文件和源文件是你項目的一部分。

我們會首先添加源文件，這時候你就需要用到 SOURCES 這個變量。寫下這樣的一行 SOURCES += 後邊加上 hello.cpp。類似如下這樣:

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

一直重復添加到最後一個源文件:

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

如果你喜歡使用 make 一樣的語法,列出所有的文件並且用換行符取拼接它們，如下所示:

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

這樣源文件就被寫在工程文件裡面了，同樣的頭文件一樣需要被添加進來。和源文件一樣的添加方法,使用變量 HEADERS.

添加完成後，你的項目文件就是如下類似的樣子:

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

生成目標文件的名字是自動生成的。和這個項目的名稱相同, 但是生成的目標文件的後綴是和平台相關的.比如, 項目的名稱是 hello.pro,在 Windows 平台下生成的目標文件就是 hello.exe，在 linux 下生成的就是 hello。如果你需要設置目標文件和項目是不同的名字，你可以這樣設置:

```
TARGET = helloworld
```

項目文件編輯完成後是這個樣子:

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

你可以在命令行下使用 qmake 為項目生成生成 Makefile，在項目的根目錄下，執行如下命令:

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

然後 make 或者 nmake 命令來編譯項目。

對於 Visual Studio 用戶，qmake 可以生成 Visual Studio 的項目文件 s.例如:

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

創建可調試的應用程序

release 版本的應用程序不包含調試符號表和其他調試信息，在開發的過程中，創建一個 debug 版本的應用程序是非常有用的。在 Qt 項目中，添加 debug 版本的程序非常簡單，通過添加 debug 到變量 CONFIG 中就可以。

For example:

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

像上面一樣生成新的 Makefiles，你就會在運行過程中獲得程序中有用的調試信息。

添加特定平台的源文件

一段時間的練習，也許會需要平台相關的代碼。並對不同平台的代碼分開處理。你現在有兩個文件包含到你的項目中:hellowin.cpp 和 hellounix.cpp。我們不能單獨這樣把它們添加到 SOURCES 中，這樣會把兩個文件度寫入 Makefile 中。所以，我們需要區分我們需要為那個平台構建程序。

為 windows 添加平台相關的文件是這樣的:

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

當 qmake 在 windows 平台運行的時候會添加 hellowin.cpp 到項目中。當為其他平台編譯的時候，qmake 會忽略這個文件。接下來我們為 unix 平台添加平台相關文件.

完成以上步驟後，項目文件變成下面的樣子:

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

```
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
```

還是像上面那樣生成 Makefiles。

如果缺少某個文件停止繼續執行 qmake

你一定不希望你創建的 Makefiles 所需要的文件是不存在的，這種情況下，我們可以使用 exists()這個內置函數來判斷。可以通過 error()這個內置函數使 qmake 停止構建項目。例子如下

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

符號!是取反得到意思，和 C++一樣，如果 main.cpp 存在，exists(main.cpp)返回 true，如果 main.cpp 不存在，!exists(main.cpp)返回 true。

```
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" )
}
```

U 按照之前的方法重新生成 Makefiles，如果你對 main.cpp 重命名後，你就會得到上面代碼裡返回的錯誤信息，qmake 會停止繼續構建項目並立即返回。

檢查多個條件

假如你使用的平台是 windows 同時你希望實時看到 qDebug()的輸出，當你在命令行下運行你的程序的時候，要想看到這些輸出，你必須在編譯你程序的時候進行適當的控制台設置。我們可以簡單的把 console 加到 CONFIG 並且當 debugdebug 已經添加到 CONFIG 變量中的時候，這時候就需要兩個相互嵌套的條件了。首先創建一段條件代碼，用花括號限定條件范圍，然後在第一個條件范圍內再增加第二個條件，如下:

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

嵌套的條件可以用冒號鏈接到一起，那麼最終的項目文件就是如下的樣子:

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

以上就是 qmake 的教程，你可以試著為你自己的項目編寫一個項目文件並且測試了。下面是一個簡單的例子作為參考：

```
#-------------------------------------------------
#
# Project created by QtCreator xxxx-xx-xx
#
#-------------------------------------------------
QT += core gui network serialport concurrent

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = example
TEMPLATE = app
DEFINES += QT_DEPRECATED_WARNINGS
TRANSLATIONS+=\
        translate/example_EN.ts
win32 {
#-------------------------------------------------
# Windows Host x86
#-------------------------------------------------
     contains(QMAKE_TARGET.arch,x86){
            INCLUDEPATH += C:/example_3rdlib/x86/include
            LIBS +=-LC:/example_3rdlib/x86/lib/
            win32:CONFIG(release, debug|release){
                        LIBS +=-lexample_3rdlib
                      }
            else:win32:CONFIG(debug, debug|release){
                        LIBS +=-lexample_3rdlibd
                            }
           }
#-------------------------------------------------
# Windows Host x64
#-------------------------------------------------
     contains(QMAKE_TARGET.arch,x86_64){
            INCLUDEPATH +=C:/example_3rdlib/x64/include
            LIBS +=-LC:/example_3rdlib/x64/lib/
            win32:CONFIG(release, debug|release){
                        LIBS +=-lexample_3rdlib
                      }
```

```
            else:win32:CONFIG(debug, debug|release){
                        LIBS +=-lexample_3rdlibd
                            }
            }
}
#-------------------------------------------------
# Linux Host
#-------------------------------------------------
unix {
INCLUDEPATH +=/path/include
LIBS += -L/path/lib/ -lexample_3rdlib
}
SOURCES += main.cpp\
       example.cpp

HEADERS  +=example.h
RESOURCES += \
    example.qrc

FORMS += \
    example.ui
RC_FILE += \
     example.rc
```

目錄

創建項目文件

Qt 項目文件(.pro)包含了 qmake 構建應用程序，庫和插件需要的所有信息。通常，在一些需要使用特定資源的項目中，可以使用簡單的編程結構來描述不同的編譯平台，不同的構建過程，不同的環境變量。

項目文件元素

使用 qmake 的 Qt 項目文件(.pro)可以支持簡單或者是復雜的構建系統。簡單的項目文件(.pro)使用簡單直接的聲明樣式，定義標准變量來指示項目中使用的頭文件和源文件。復雜的項目可以使用控制流結構來調整構建過程。

以下部分詳細描述了項目文件(.pro)使用的不同類型的元素。

變量

在項目文件(.pro)中，變量是用來存儲字符串的列表。在簡單的項目中，這些變量告知 qmake 要使用的配置選項，或者提供在構建過程中使用的文件名和路徑。

qmake 會搜索每個項目文件中包含的變量，並利用這些變量來決定什麼內容應該被寫入 Makefile 文件中。比如，HEADERS and SOURCES 變量會把項目包含的頭文件和源文件列表告知給 qmake，頭文件和源文件的路徑是相對於項目文件(.pro)所在的目錄。

變量還可以用來存儲內部臨時列表值，現有的列表值可以擴展和覆蓋。

下面的片段說明了如何把列表的值賦值給變量：

```
HEADERS = mainwindow.h paintwidget.h
```

變量的列表值可以用以下的方法擴展：

```
SOURCES = main.cpp mainwindow.cpp \
          paintwidget.cpp
CONFIG += console
```

說明: 上面的兩個例子中，第一個例子中，HEADERS 變量包含了與 HEADERS 變量同一行的內容。第二個例子中通過一個反斜槓(\)拼接了 SOURCES 變量的內容。

The CONFIG 變量是另一個 qmake 需要的特殊變量來生成 Makefiles，它的詳細介紹在常規配置中。在上面的代碼中，console 會被添加到 CONFIG 的列表值中。

下面的列表給出了常用的變量及其具體描述，關於完整的變量的描述請參考變量。

變量	描述
CONFIG	常規項目的配置選項
DESTDIR	編譯生成的可執行文件或二進制文件存放的文件夾。
FORMS	用戶界面文件(後綴為.ui 文件)的列表，這些文件會被用戶界面編譯器(uic)編譯處理。
HEADERS	編譯項目所需要的頭文件(.h)的列表。
QT	項目中使用的 Qt 模塊的列表。
RESOURCES	項目包含的資源文件(.qrc)的列表，參考 Qt 資源系統獲取資源文件的更多信息。
SOURCES	編譯項目所需要的源文件的列表。
TEMPLATE	項目的模板，模板的選擇決定項目編譯輸出的是一個應用程序，或者一個庫，或者是一個插件。

可以通過把**$$**放在變量名稱之前來獲取變量的內容。這種方法可以使一個變量的值賦值給另一個變量:

```
TEMP_SOURCES = $$SOURCES
```

操作符**$$**和內置函數被廣泛的使用在處理字符串和列表的值，更多信息請參考 qmake 語法。

空格

通常情況下，空格用來在變量賦值的過程中分割內容，如果指定的內容包含空格，你必須附上雙引號包含這些內容:

```
DEST = "Program Files"
```

雙引號裡面包含的內容會被視為變量值的列表中的一個元素，類似的方法可以被用來處理帶空格的路徑，特別是當我們為 windows 平台定義 INCLUDEPATH 和 LIBS 變量的時候:

```
win32:INCLUDEPATH += "C:/mylibs/extra headers"
unix:INCLUDEPATH += "/home/user/extra headers"
```

注釋

你可以為項目文件添加注釋，注釋以#開始直到行結束，例如:

```
# usually start at the beginning of a line, but they
# can also follow other content on the same line.
```

如果你想要包含帶有#字符變量賦值，就需要使用內置的 LITERAL_HASH 變量。

內置函數和控制流

qmake 提供了許多內置函數來處理變量的內容，在一些常規項目中最常使用的內置函數就是 include()，這個函數以文件名為參數，這個給定的文件將被包含到項目文件中 include 被使用的地方，include 函數常用於一個項目文件包含另一個其他的項目文件的情況:

```
include(other.pro)
```

Qt 通過 scopes 支持條件結構體，類似與 if 語句:

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

在花括號內部的賦值只有在上面的條件成立的時候才會生效，在這種情況下，必須把 win32 設置到 CONFIG 變量中去，在 windows 平台下 win32 會自動被設置到 CONFIG 變量中，一對花括號的開括號必須與判斷條件在同一行。

更復雜的操作需要循環調用內置函數實現，比如 find(), unique(),和 count()。 這些函數和其他提供操作字符串和路徑的工具，支持用戶輸入和調用外部命令的功能。更多參考 qmake 語法. 詳細函數介紹參考替換函數 and 測試函數.

項目模板

項目目標 TEMPLATE 變量被用來定義編譯項目類型的。如果沒有指定，qmake 會默認編譯為應用程序類型。

下面的列表總結了 qmake 支持的項目類型模板和描述了不同類型模板下生成的文件類型。

項目模板	qmake 輸出
app (default)	Makefile 編譯生成應用程序
lib	Makefile 編譯生成一個庫。
aux	Makefile 不編譯生成任何東西， 不需要編譯器生成目標文件，比如項目是用解釋型語言編寫的。 **說明:** 此模板只適用於基於 makefiles 的生成器，不支持 vcxproj 和 Xcode 生成器。
subdirs	Makefile 包含指定的子目錄的規則，子目錄通過指定 SUBDIRS 變量來實現。每一個子目錄都包含自己的項目文件。
vcapp	Visual Studio 應用程序項目。
vclib	Visual Studio 庫項目
vcsubdirs	Visual Studio 多項目解決方案。

參見 構建通用項目中有關為使用應用程序 app 和庫 lib 模板項目的編寫建議。

當項目中配置了 subdirs，qmake 生成的 Makefiles 會檢查每一個子目錄，處理所有能找到的項目文件，執行平台的構建命令 make，SUBDIRS 變量被用來包含所有要被處理的子目錄的列表。

常規配置

CONFIG 變量指定了項目配置的選項和功能。

項目可以被編譯成 *release* 模式或者 *debug* 模式，或者全部。如果同時指定了 debug 和 release，最後指定的選項會生效，如果指定了 debug_and_release 選項為項目編譯 debug 和 release 兩種版本 qmake 生成的 Makefile 包含了編譯兩種模式的編譯規則，可以用下面的方法調用:

```
make all
```

添加 build_allCONFIG 變量中，在構建項目中設置為默認編譯選項。

說明: 每一個在 CONFIG 變量中指定的選項可以同樣被用於條件選項。 你可以使用內置函數 CONFIG()測試某些配置選項是否存在，例如 the 下面代碼測試了是否 opengl 被配置使用了:

```
CONFIG(opengl) {
    message(Building with OpenGL support.)
} else {
    message(OpenGL support is not available.)
}
```

Qt 允許在構建 release 和 debug 版本時候使用不同的配置選項。更過信息請參考 Using Scopes.

下面定義的項目選項蔣被構建到項目中去

說明: 其中歐謝選項只會在相關的平台上才會生效。

選項	描述
qt	這是一個 Qt 應用程序，必須鏈接 Qt 庫文件。你可以使用 QT 變量控制程序額外需要的任何一個的 qt 的庫。這個值是默認添加的，但是你可以通過使用 qmake 去除它，變成一個不適用 qt 庫的應用程序。
x11	這是一個 X11 的應用程序或庫文件。如果使用 Qt 的情況下，這個值是不必要的。.

應用程序和庫項目模板提供了更專業的配置選項對構建項目進行調整。這些選項的詳細說明在構建通用項目中。

比如，你的項目使用了 Qt 庫，並且你想構建這個項目的 debug 模式,那麼你的項目文件應該包含如下片段。

```
CONFIG += qt debug
```

說明: 你必須使用"+=",而不是"=", 否則 qmake 將不能通過 Qt 的配置確定項目所需要的設置。

聲明 Qt 庫

如果 CONFIG 變量中包含 qt 這個值，qmake 將支持 Qt 應用程序的編譯。這樣會使得你靈活調整你項目程序中使用的 Qt 模塊。這是通過 QT 變量實現對外部模塊的聲明。比如:我們通過下面的方法實現添加 network 和 xml 模塊的添加：

```
QT += network xml
```

說明:QT 默認包含 core 和 gui 模塊,所以以上的片段*添加了* network 和 XML 模塊到默認的列表中。下面的賦值*省略*默認的模塊, 當編譯的時候會發生錯誤。

```
QT = network xml # This will omit the core and gui modules.
```

如果你想編譯你的程序*不使用* gui 模塊,你需要添加"-="操作符：在默認情況下,QT 同時包含了 core 和 gui,那麼下面的代碼將會生成一個不包含 gui 的項目:

```
QT -= gui # Only the core module is used.
```

關於 QT 變量可添加的 Qt 的模塊列表詳情,參考 QT.

配置功能

qmake 可以設置額外的配置選項，這些配置選項是在(.prf)文件中定義的。這些額外的配置選項，通常是為了支持在構建過程中使用自定義的工具。為構建過程添加一個特性, 將特性的名字(特性文件的豬名)添加到 CONFIG 變量中。

例如，qmake 可以通過配置項目使用 pkg-config 支持的外部庫,比如 D-Bus 和 ogg 庫等。 如下所示:

```
CONFIG += link_pkgconfig
PKGCONFIG += ogg dbus-1
```

關於更多使用和添加特性的內容，請參考添加配置特性.

聲明其他庫

如果你在項目中使用其他的非 qt 提供的庫，你需要在你的項目文件中指定它們。

qmake 庫搜索路徑和指定庫的鏈接可以通過 LIBS 變量添加。你可以指定庫搜索路徑或者使用 Unix 風格的標記法指定庫的搜索路徑和庫。

下面的例子顯示了如何指定第三方庫:

```
LIBS += -L/usr/local/lib -lmath
```

包含頭文件的路徑也可以用類似的方式指定 INCLUDEPATH 變量的內容。

例如添加頭文件搜索路徑:

```
INCLUDEPATH = c:/msdev/include d:/stl/include
```