1.开发中用到的qmake实用变量

变量	含义
TARGET	生成目标的名字
DESTDIR	目标输出文件路径
PWD	当前文件(.pro或.pri)所在的路径
OUT_PWD	Makefile生成的路径
PRO_FILE_PWD	pro项目文件所在的路径
QMAKE_HOST.arch	计算机架构
QMAKE_HOST.os	计算机系统
MAKE_HOST.cpu_count	计算机CPU核心数
QMAKE_HOST.name	计算机名
QMAKE_HOST.version	系统版本(数字形式)
QMAKE_HOST.version_string	系统版本(字符串形式)
QMAKE_POST_LINK	编译链接后自动执行命令
DEFINES	编译器定义的预处理器宏，类似于gcc -D 选项
INCLUDEPATH	包含头文件路径
LIBS	指定要链接到项目中的库

2.Qt pro文件中区分 linux/windows系统

```
win32{
//do something...
}
unix{
//do something...
}
```

3.Qt pro文件中区分debug/release版本

```
CONFIG += debug_and_release
CONFIG(debug, debug|release){  //处理debug
}else{  //处理release
}
```

4.Qt pro文件中判断 x86/arm(aarch64)交叉编译环境

```
contains(QT_ARCH, arm64){
message("arm64") #在这里处理arm64所需
}else{
message("x86")
}
```

5.Qt当中系统判断

```
//Windows系统
#ifdef Q_OS_WIN

#endif

//Linux系统
#ifdef Q_OS_LINUX

#endif

//MAC系统
#ifdef Q_OS_MAC

#endif
```

6.Qt当中编译器及其版本判断

```
Qt<qtglobal.h>定义了Q_OS_*和Q_WS_*系列用于判断操作系统。Q_CC_*系列判断编译器。
```

具体的可以在Qt Assistant里索引qtglobal.h查看。

编译器

GCC

```
#ifdef __GNUC__
#if __GNUC__ >= 3 // GCC3.0以上
```

Visual C++

```
#ifdef _MSC_VER
#if _MSC_VER >=1000 // VC++4.0以上
#if _MSC_VER >=1100 // VC++5.0以上
#if _MSC_VER >=1200 // VC++6.0以上
#if _MSC_VER >=1300 // VC2003以上
#if _MSC_VER >=1400 // VC2005以上
```

Borland C++

```
#ifdef __BORLANDC__
```

Cygwin

```
#ifdef __CYGWIN__
#ifdef __CYGWIN32__    //
```

MinGW

```
#ifdef __MINGW32__
```

操作系统

Windows

```
#ifdef _WIN32    //32bit
#ifdef _WIN64    //64bit
#ifdef _WINDOWS     //图形界面程序
#ifdef _CONSOLE     //控制台程序
//Windows（95/98/Me/NT/2000/XP/Vista）和Windows CE都定义了
#if (WINVER >= 0x030a)     // Windows 3.1以上
#if (WINVER >= 0x0400)     // Windows 95/NT4.0以上
#if (WINVER >= 0x0410)     // Windows 98以上
#if (WINVER >= 0x0500)     // Windows Me/2000以上
#if (WINVER >= 0x0501)     // Windows XP以上
#if (WINVER >= 0x0600)     // Windows Vista以上
//_WIN32_WINNT 内核版本
```

```
#if (_WIN32_WINNT >= 0x0500) // Windows 2000以上
#if (_WIN32_WINNT >= 0x0501) // Windows XP以上
#if (_WIN32_WINNT >= 0x0600) // Windows Vista以上

UNIX
#ifdef __unix
//or
#ifdef __unix__

Linux
#ifdef __linux
//or
#ifdef __linux__

FreeBSD
#ifdef __FreeBSD__

NetBSD
#ifdef __NetBSD__
```

7.QT中-O编译优化方式

选择编译release模式，在pro文件根据优化的需要添加下面的语句，根据需要优化的级别，任选一个添加即可

```
QMAKE_CXXFLAGS_RELEASE += -O        # Release -O
QMAKE_CXXFLAGS_RELEASE += -O1       # Release -O1
QMAKE_CXXFLAGS_RELEASE += -O2       # Release -O2
QMAKE_CXXFLAGS_RELEASE += -O3       # Release -O3
```

8.QT中设置程序图标

（1）在项目文件夹中，将图标文件cal.ico文件加入，新建一个logo.txt，然后在logo.txt文件中写入如下内容（cal.ico就是图标文件的名称）：

```
IDI_ICON1 ICON DISCARDABLE "cal.ico"
```

然后将logo.txt重命名为logo.rc，在.pro文件中加入写入如下内容：

```
RC_FILE = logo.rc
```

（2）在项目文件夹中，将图标文件cal.ico文件加入，然后在.pro文件中写入如下内容：

```
RC_ICONS=cal.ico
```

（3）PNG图片转换为ico图标

http://www.ico51.cn/

9.Qt程序增加公司信息、版本号、说明等

（1）在pro文件中增加对应的宏。

```

# 版本信息
VERSION = 4.0.2.666

# 图标
RC_ICONS = Images/MyApp.ico

# 公司名称
QMAKE_TARGET_COMPANY = "Digia"

# 产品名称
QMAKE_TARGET_PRODUCT = "Qt Creator"

# 文件说明
QMAKE_TARGET_DESCRIPTION = "Qt Creator based on Qt 5.7.0 (MSVC 2013, 32 bit)"

# 版权信息
QMAKE_TARGET_COPYRIGHT = "Copyright 2008-2016 The Qt Company Ltd. All rights
reserved."

# 中文（简体）
RC_LANG = 0x0004
```

（2）自定义rc文件

在pro文件中增加：

```
RC_FILE += XXX.rc
```

xxx.rc文件内容

```
#include <windows.h>

//中文的话增加下面这一行

#pragma code_page(65001)

VS_VERSION_INFO VERSIONINFO
FILEVERSION 4,0,2,666
PRODUCTVERSION 4,0,2,666
FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
FILEFLAGS VS_FF_DEBUG
#else
FILEFLAGS 0x0L
#endif
FILEOS VOS__WINDOWS32
FILETYPE VFT_DLL
FILESUBTYPE 0x0L
BEGIN
BLOCK "StringFileInfo"
BEGIN
BLOCK "000404b0"
BEGIN
VALUE "CompanyName", "Digia\0"
VALUE "FileDescription", "Qt Creator based on Qt 5.7.0 (MSVC 2013, 32 bit)\0"
VALUE "FileVersion", "4.0.2.666\0"
VALUE "LegalCopyright", "Copyright 2008-2016 The Qt Company Ltd. All rights
reserved.\0"
```

```
VALUE "OriginalFilename", "test_rc.exe\0"
VALUE "ProductName", "Qt Creator\0"
VALUE "ProductVersion", "4.0.2.666\0"
END
END
BLOCK "VarFileInfo"
BEGIN
VALUE "Translation", 0x0004, 1200
END
END
```

（3）设置方法

在QMake Manual手册中搜索关于QMAKE_TARGET内容可以看到有如下QMake变量：

- QMAKE_TARGET_COMPANY：用于指定生产商
- QMAKE_TARGET_DESCRIPTION：用于描述应用程序
- QMAKE_TARGET_COPYRIGHT：用于声明版权
- QMAKE_TARGET_PRODUCT：用于指定产品名称

所以在.pro文件中设置相关变量即可。

需要注意：

- 如果变量值是中文，需要将.pro文件以system本地编码保存。

前提：需要在pro中设置VERSION变量才能生效。

10.QMake手册

Qt5 qmake手册：https://doc.qt.io/qt-5/qmake-manual.html

Qt6 qmake手册：https://doc.qt.io/qt-6/qmake-manual.html

（1）使应用程序可调试

应用程序的发布版本不包含任何调试符号或其他调试信息。在开发过程中，生成具有相关信息的应用程序的调试版本非常有用。这可以通过添加到项目文件中的变量来轻松实现。debug

例如：

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

像以前一样使用 qmake 生成 Makefile。现在，在调试环境中运行应用程序时，您将获得有关应用程序的有用信息。

（2）区分 linux/windows系统

```
win32{
//do something...
}
unix{
//do something...
}
```

（3）如果文件不存在，则停止 qmake

如果某个文件不存在，您可能不想创建 Makefile。我们可以通过使用该函数来检查文件是否存在。我们可以通过使用该函数来阻止 qmake 处理。其工作方式与作用域相同。只需将范围条件替换为函数即可。对名为 main.cpp 的文件的检查如下所示：

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

该符号用于否定测试。也就是说，如果文件存在，则为 true，如果文件不存在，则为true。!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" )
}
```

像以前一样使用 qmake 生成 makefile。如果临时重命名，您将看到该消息，并且 qmake 将停止处理。main.cpp

（4）检查多个条件

假设您使用 Windows，并且希望在命令行上运行应用程序时能够查看语句输出。若要查看输出，必须使用适当的控制台设置生成应用程序。我们可以轻松地将此设置包含在 Windows 上的 Makefile 中。但是，假设我们只想在 Windows 上运行并且已经在线时添加该行。这需要使用两个嵌套作用域。首先创建一个范围，然后在其中创建另一个范围。将要处理的设置放在第二个作用域内，如下所示：

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

嵌套作用域可以使用冒号连接在一起，因此最终的项目文件如下所示：

```
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
}
```

（5）变量

下表列出了一些常用的变量，并描述了它们的内容。有关变量及其说明的完整列表，请参见Variables | qmake Manual (qt.io)

变量	内容
CONFIG	常规项目配置选项。
DESTDIR	可执行文件或二进制文件所在的目录。
FORMS	要由 处理的 UI 文件的列表。
HEADERS	生成项目时使用的标头 （.h） 文件的文件名列表。
QT	项目中使用的Qt模块列表。
RESOURCES	要包含在最终项目中的资源 （.qrc） 文件的列表。有关这些文件的更多信息，请参见 。
SOURCES	生成项目时要使用的源代码文件列表。
TEMPLATE	要用于项目的模板。这决定了构建过程的输出是应用程序、库还是插件。
TARGET	生成目标的名字