QT 中.pro 文件

qmake 变量	含义	
#xxxx	注释，从“#”开始，到这一行结束	
SOURCES	指定源文件	SOURCES = *.cpp
	对于多源文件，可用空格分开	SOURCES = 1.cpp 2.cpp 3.cpp
	或者每一个文件可以被列在一个分开的行里面，通过反斜线另起一行	SOURCES = hello.cpp \ main.cpp
	一个更冗长的方法是单独地列出每一个文件，就像这样	SOURCES += hello.cpp SOURCES += main.cpp
HEADERS	指定头文件	HEADERS = hello.h HEADERS += hello.h
CONFIG	配置信息	CONFIG+= qt warn_on release
	编译器标志： ● release - 应用程序将以 release 模式连编。如果“debug”被指定，它将被忽略。 ● debug - 应用程序将以 debug 模式连编。 ● warn_on - 编译器会输出尽可能多的警告信息。如果“warn_off”被指定，它将被忽略。 ● warn_off - 编译器会输出尽可能少的警告信息。 连编的库/应用程序的类型： ● qt - 应用程序是一个 Qt 应用程序，并且 Qt 库将会被连接。 ● thread - 应用程序是一个多线程应用程序。 ● x11 - 应用程序是一个 X11 应用程序或库。 ● windows - 只用于“app”模板：应用程序是一个 Windows 下的窗口应用程序。 ● console - 只用于“app”模板：应用程序是一个 Windows 下的控制台应用程序。 ● dll - 只用于“lib”模板：库是一个共享库（dll）。 ● staticlib - 只用于“lib”模板：库是一个静态库。 ● plugin - 只用于“lib”模板：库是一个插件，这将会使 dll 选项生效。	
TARGET	指定目标文件名 如果不设置该项目，目标名会被自动设置为跟项目文件一样的名称	TARGET = filename
INTERFACES	添加界面文件(ui)	INTERFACES = filename.ui
TEMPLATE	模块设置 app(生成应用程序,默认) subdirs(生成 makefile 文件编译 subdirs 指定的子文件夹) lib(生成库文件)	TEMPLATE = app
DESTDIR	指定生成的应用程序放置的目录	DESTDIR += ../bin

UI_DIR	指定 uic 命令将.ui 文件转化成 ui_*.h 文件的存放的目录	UI_DIR += forms
RCC_DIR	指定 rcc 命令将.qrc 文件转换成 qrc_*.h 文件的存放目录	RCC_DIR += ../tmp
MOC_DIR	指定 moc 命令将含 Q_OBJECT 的头文件转换成标准.h 文件的存放目录	MOC_DIR += ../tmp
OBJECTS_DIR	指定目标文件的存放目录	OBJECTS_DIR += ../tmp
DEPENDPATH	程序编译时依赖的相关路径	DEPENDPATH += . forms include qrc sources
INCLUDEPATH	头文件包含路径	INCLUDEPATH += .
CODECFORSRC	源文件编码方式	CODECFORSRC = GBK
FORMS	工程中包含的.ui 设计文件	FORMS += forms/painter.ui
RESOURCES	工程中包含的资源文件	RESOURCES += qrc/painter.qrc

win32{…}
unix{…}
	平台相关性处理	win32 {
		SOURCES += hello_win.cpp
		}

LANGUAGE		LANGUAGE = C++

exists
!exists
	如果一个文件不存在，停止 qmake	!exists(main.cpp) {
		error("No main.cpp file found")
		}

QT	加入库模块	QT += xml
LIBS	LIBS += -L folderPath //引入的 lib 文件的路径 -L：引入路径 LIBS += -lLibName //引入 lib 文件 -l：引入库	LIBS += -L"$(OutDir)" \ -L"$(SolutionDir)lib" \ -lopengl32 \ -lglu32 \ -lObjectDbAPI \ -lGraphicsLibD

更多其他变量：

app - 建立一个应用程序的 makefile。这是默认值，所以如果模板没有被指定，这个将被使用。

lib - 建立一个库的 makefile。

vcapp - 建立一个应用程序的 Visual Studio 项目文件。

vclib - 建立一个库的 Visual Studio 项目文件。

subdirs - 这是一个特殊的模板，它可以创建一个能够进入特定目录并且为一个项目文件生成 makefile 并且为它调用 make 的 makefile。

“app”模板
“app”模板告诉 qmake 为建立一个应用程序生成一个 makefile。当使用这个模板时，下面这些 qmake 系统变量是被承认的。你应该在你的.pro 文件中使用它们来为你的应用程序指定特定信息。

HEADERS - 应用程序中的所有头文件的列表。

SOURCES - 应用程序中的所有源文件的列表。

FORMS - 应用程序中的所有.ui 文件（由 Qt 设计器生成）的列表。

LEXSOURCES - 应用程序中的所有 lex 源文件的列表。

YACCSOURCES - 应用程序中的所有 yacc 源文件的列表。

TARGET - 可执行应用程序的名称。默认值为项目文件的名称。（如果需要扩展名，会被自动加上。）

DESTDIR - 放置可执行程序目标的目录。

DEFINES - 应用程序所需的额外的预处理程序定义的列表。

INCLUDEPATH - 应用程序所需的额外的包含路径的列表。

DEPENDPATH - 应用程序所依赖的搜索路径。

VPATH - 寻找补充文件的搜索路径。

DEF_FILE - 只有 Windows 需要：应用程序所要连接的.def 文件。

RC_FILE - 只有 Windows 需要：应用程序的资源文件。

RES_FILE - 只有 Windows 需要：应用程序所要连接的资源文件。

CONFIG 变量
配置变量指定了编译器所要使用的选项和所需要被连接的库。配置变量中可以添加任何东西，但只有下面这些选项可以被 qmake 识别。

下面这些选项控制着使用哪些编译器标志：

release - 应用程序将以 release 模式连编。如果“debug”被指定，它将被忽略。

debug - 应用程序将以 debug 模式连编。

warn_on - 编译器会输出尽可能多的警告信息。如果“warn_off”被指定，它将被忽略。

warn_off - 编译器会输出尽可能少的警告信息。

下面这些选项定义了所要连编的库/应用程序的类型：

qt - 应用程序是一个 Qt 应用程序，并且 Qt 库将会被连接。

thread - 应用程序是一个多线程应用程序。

x11 - 应用程序是一个 X11 应用程序或库。

windows - 只用于“app”模板：应用程序是一个 Windows 下的窗口应用程序。

console - 只用于“app”模板：应用程序是一个 Windows 下的控制台应用程序。

dll - 只用于“lib”模板：库是一个共享库（dll）。

staticlib - 只用于“lib”模板：库是一个静态库。

plugin - 只用于“lib”模板：库是一个插件，这将会使 dll 选项生效。

例如，如果你的应用程序使用 Qt 库，并且你想把它连编为一个可调试的多线程的应用程序，你的项目文件应该会有下面这行：

CONFIG += qt thread debug 注意，你必须使用“+=”，不要使用“=”，否则 qmake 就不能正确使用连编 Qt 的设置了，比如没法获得所编译的 Qt 库的类型了。

qmake 高级概念
操作符

“=”操作符　　分配一个值给一个变量
“+=”操作符　　向一个变量的值的列表中添加一个值
“-=”操作符　　从一个变量的值的列表中移去一个值
“*=”操作符　　仅仅在一个值不存在于一个变量的值的列表中的时候，把它添加进去
“~=”操作符　　替换任何与指定的值的正则表达式匹配的任何值　DEFINES ~= s/QT_[DT].+/QT

作用域
win32:thread {
DEFINES += QT_THREAD_SUPPORT } else:debug { DEFINES += QT_NOTHREAD_DEBUG } else { warning("Unknown configuration") } }变量
到目前为止我们遇到的变量都是系统变量，比如 DEFINES、SOURCES 和 HEADERS。你也可以为你自己创建自己的变量，这样你就可以在作用域中使用它们了。创建自己的变量很容易，只要命名它并且分配一些东西给它。比如：

MY_VARIABLE = value 你也可以通过在其它任何一个变量的变量名前加$$来把这个变量的值分配给当前的变量。例如：

MY_DEFINES = $$DEFINESMY_DEFINES = $${DEFINES}
第二种方法允许你把一个变量和其它变量连接起来，而不用使用空格。qmake 将允许一个变量包含任何东西（包括$(VALUE)，可以直接在 makefile 中直接放置，并且允许它适当地扩张，通常是一个环境变量）。无论如何，如果你需要立即设置一个环境变量，然后你就可以使用$$()方法。比如：

MY_DEFINES = $$(ENV_DEFINES)这将会设置 MY_DEFINES 为环境变量 ENV_DEFINES 传递给.pro 文件地值。另外你可以在替换的变量里调用内置函数。这些函数（不会和下一节中列举的测试函数混淆）列出如下：

join(variablename, glue, before, after)
这将会在 variablename 的各个值中间加入 glue。如果这个变量的值为非空，那么就会在值的前面加一个前缀 before 和一个后缀 after。只有 variablename 是必须的字段，其它默认情况下为空串。如果你需要在 glue、before 或者 after 中使用空格的话，你必须提供它们。

member(variablename, position)
这将会放置 variablename 的列表中的 position 位置的值。如果 variablename 不够长，这将会返回一个空串。variablename 是唯一必须的字段，如果没有指定位置，则默认为列表中的第一个值。

find(variablename, substr)
这将会放置 variablename 中所有匹配 substr 的值。substr 也可以是正则表达式，而因此将被匹配。

MY_VAR = one two three four MY_VAR2 = $$join(MY_VAR, " -L", -L) -Lfive MY_VAR3 = $$member(MY_VAR, 2) $$find(MY_VAR, t.*)MY_VAR2 将会包含“-Lone -Ltwo -Lthree -Lfour -Lfive”，并且 MYVAR3 将会包含“three two three”。

system(program_and_args)
这将会返回程序执行在标准输出/标准错误输出的内容，并且正像平时所期待地分析它。比如你可以使用这个来询问有关平台的信息。

UNAME = $$system(uname -s) contains(UNAME, [lL]inux):message(This looks like

Linux ($$UNAME) to me)测试函数
qmake 提供了可以简单执行，但强大测试的内置函数。这些测试也可以用在作用域中（就像上面一样），在一些情况下，忽略它的测试值，它自己使用测试函数是很有用的。

contains(variablename, value)
如果 value 存在于一个被叫做 variablename 的变量的值的列表中，那么这个作用域中的设置将会被处理。例如：

contains(CONFIG, thread) { DEFINES += QT_THREAD_SUPPORT }如果 thread 存在于 CONFIG 变量的值的列表中时，那么 QT_THREAD_SUPPORT 将会被加入到 DEFINES 变量的值的列表中。

count(variablename, number)
如果 number 与一个被叫做 variablename 的变量的值的数量一致，那么这个作用域中的设置将会被处理。例如：

count(DEFINES, 5) { CONFIG += debug }error(string)
这个函数输出所给定的字符串，然后会使 qmake 退出。例如：

error("An error has occured")文本“An error has occured”将会被显示在控制台上并且 qmake 将会退出。

exists(filename)
如果指定文件存在，那么这个作用域中的设置将会被处理。例如：

exists(/local/qt/qmake/main.cpp) { SOURCES += main.cpp }如果 /local/qt/qmake/main.cpp 存在，那么 main.cpp 将会被添加到源文件列表中。

注意可以不用考虑平台使用“/”作为目录的分隔符。

include(filename)
项目文件在这一点时包含这个文件名的内容，所以指定文件中的任何设置都将会被处理。例如：

include(myotherapp.pro)myotherapp.pro 项目文件中的任何设置现在都会被处理。

isEmpty(variablename)
这和使用 count(variablename, 0)是一样的。如果叫做 variablename 的变量没有任何元素，那么这个作用域中的设置将会被处理。例如：

isEmpty(CONFIG) { CONFIG += qt warn_on debug }message(string)
这个函数只是简单地在控制台上输出消息。

message("This is a message")文本“This is a message”被输出到控制台上并且对于项目

文件的处理将会继续进行。

system(command)
特定指令被执行并且如果它返回一个 1 的退出值，那么这个作用域中的设置将会被处理。例如：

system(ls /bin) { SOURCES += bin/main.cpp HEADERS += bin/main.h }
所以如果命令 ls /bin 返回 1，那么 bin/main.cpp 将被添加到源文件列表中并且 bin/main.h 将被添加到头文件列表中。

infile(filename, var, val)
如果 filename 文件（当它被 qmake 自己解析时）包含一个值为 val 的变量 var，那么这个函数将会返回成功。你也可以不传递第三个参数（val），这时函数将只测试文件中是否分配有这样一个变量 var。

以下为我的一个项目举例
项目目标：为一个库文件

TEMPLATE = lib # 编译项目文件所需头文件的路径
INCLUDEPATH += ../common # 目标文件路径
DESTDIR=../lib # 条件依赖：Unix 平台上 定义本想目的 UI 目录， MOC 目录， 目的目录
unix { UI_DIR = ../.ui MOC_DIR = ../.moc OBJECTS_DIR = ../.obj} # 本项目配置：
CONFIG += qt warn_on release thread# Input 头文件，源文件 HEADERS += COMControllerThread.h \ DecodeSMS.h \ monitor_common.h \ monitor_interface.h \ MonitorThread.h \ UserEvent.h \ MyCOM.h \ MySMS.h \ MyTagHandle.h \ SMSParseThread.h \ tag_dict.hSOURCES += COMControllerThread.cpp \ DecodeSMS.cpp \ monitor_common.cpp \ monitor_interface.cpp \ MonitorThread.cpp \ MyCOM.cpp \ MySMS.cpp \ MyTagHandle.cpp \ SMSParseThread.cpp \ tag_dict.cpp 注：qmake -project 可以生成 pro 文件（可以根据项目需要，编辑改文件）
qmake 可以生成 Makefile 文件

make 编译
使用 qmake -project 时，会把本目录及其子目录内所有.cpp .h 文件加入到项目输入文件中，使用是注意移去其他无用的文件。
qmake 生成的 Makefile 文件，可以根据需要做相应修改