|  |
| --- |
| **Qt pro工程文件 介绍** |
| |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 1. TEMPLATE  变量TEMPLATE描述了为建立目标文件而采用何种模板,即生成何种形式的Makefile文件。Qmake   工具定义了5种模板:   1. 应用程序App,为建立一个Qt应用程序创建Makefile文件;   2. 库lib,为建立引用程序库而创建Makefile文件;   3. 子工程 subdirs,为建立子目录下的目标文件创建一个Makefile文件,      子目录通过变量SUBDIRS指定(子目录下的工程文件也需要指出使用何种模板);   4. VC应用程序vcapp,为Visual Studio 生成一个应用程序工程,仅仅用语Windos操作系统.   5. VC库vclib,为Visual Studio生成一个应用程序库工程,仅仅用语Windows操作系统.  {  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。  }  2.HEADERS - 应用程序中的所有头文件的列表。  3.SOURCES - 应用程序中的所有源文件的列表。  4.FORMS / INTERFACES - 应用程序中的所有.ui文件（由Qt设计器生成）的列表。    eg.INTERFACES = filename.ui  5.LEXSOURCES - 应用程序中的所有lex源文件的列表。  6.YACCSOURCES - 应用程序中的所有yacc源文件的列表。  7.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。）  8.DESTDIR - 放置可执行程序目标的目录。  9.DEFINES - 应用程序所需的额外的预处理程序定义的列表。  10.INCLUDEPATH - 应用程序所需的额外的包含路径的列表（include文件路径列表）。  11.DEPENDPATH - 应用程序所依赖的搜索路径（描述了建立应用程序所依赖的其他文件所在的路 径）。  12.VPATH - 寻找补充文件的搜索路径。  13.DEF\_FILE - 只有Windows需要：应用程序所要连接的.def文件。  14.C\_FILE - 只有Windows需要：应用程序的资源文件。  15.RES\_FILE - 只有Windows需要：应用程序所要连接的资源文件。  16.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。  eg.  CONFIG += qt warn\_on release  在这里使用“+=”，是因为我们添加我们的配置选项到任何一个已经存在中。这样做比使用“=”那样替换已经指定的所有选项是更安全的。 A> qt部分告诉qmake这个应用程序是使用Qt来连编的。这也就是说qmake在连接和为编译添加所需的包含路径的时候会考虑到Qt库的。 B> warn\_on部分告诉qmake要把编译器设置为输出警告信息的。 C> release部分告诉qmake应用程序必须被连编为一个发布的应用程序。在开发过程中，程序员也可以使用debug来替换release  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（指定平台相关性等操作） 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 定义本项目的 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.h  SOURCES += 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，使用时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 | | | |