QT 中 pro文件的写法

1. QT常用变量说明

qmake 变量 含义

例子

#XXX 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

UI\_DIR += forms

void function(e,t){for(var n=t.getElementsByTagName("img"),a=+new Date,i=[],o=function(){this.removeEventListener&&this.removeEventListener("load",o,!1),i.push({img:this,time:+new Date})},s=0;s< n.length;s++)!function(){var e=n[s];e.addEventListener?!e.complete&&e.addEventListener("load",o,!1):e.attachEvent&&e.attachEvent("onreadystatechange",function(){"complete"==e.readyState&&o.call(e,o)})}();alog("speed.set",{fsItems:i,fs:a})}(window,document);

的目录

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

2. qmake高级概念

2.1 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

2.2 检查多于一个的条件

两个嵌套的作用域写法： win32 {    debug {

     CONFIG += console    } }

var cpro\_psid ="u2572954"; var cpro\_pswidth =966; var cpro\_psheight =120;

嵌套的作用域可以使用冒号连接起来，像这样： win32:debug {

CONFIG += console }

2.3 作用域

win32:thread {

DEFINES += QT\_THREAD\_SUPPORT  }

else:debug {

DEFINES += QT\_NOTHREAD\_DEBUG  }  else {

warning("Unknown configuration")  }

2.4 变量

除了系统变量，比如DEFINES、SOURCES和HEADERS，允许创建自己的变量，这样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。

例如： 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  访问系统变量，采用 $(VARNAME) 来引用。

例如：QTITAN\_LIB\_PATH = $(GLDRS)\ThirdPart\Qtitan\lib

2.5 函数

这里只举几个最常用的例子    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  }

表示： 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。

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  }

表示： 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。

/\* pc阅读页3-4页间（新） \*/

var cpro\_id = "u2845472";

var cpro\_psid = "u2787156";

var cpro\_pswidth = "966";

var cpro\_psheight = "120";