搜索

qmake语言

许多 qmake 项目文件只是使用列表 and 定义来描述项目使用的源和头文件。qmake 还提供了其他运算符、函数和作用域，可用于处理变量声明中提供的信息。这些高级功能允许从单个项目文件为多个平台生成生成文件。`name = valuename += value`

运营商

在许多项目文件中，赋值 （） 和追加 （） 运算符可用于包含有关项目的所有信息。典型的使用模式是为变量分配值列表，并根据各种测试的结果附加更多值。由于 qmake 使用默认值定义某些变量，因此有时需要使用 remove （） 运算符来过滤掉不需要的值。以下部分介绍如何使用运算符操作变量的内容。`=+=-=`

分配值

运算符为变量赋值：`=`

```
TARGET = myapp
```

上行将目标变量设置为 。这将覆盖之前为其设置的任何值。`myappTARGETmyapp`

追加值

运算符将新值追加到变量中的值列表中：`+=`

```
DEFINES += USE_MY_STUFF
```

上面的行附加到要放入生成的生成文件中的预处理器定义列表中。`USE_MY_STUFF`

删除值

运算符从变量中的值列表中删除值：`-=`

上面的行从要放入生成的 Makefile 中的预处理器定义列表中删除。USE_MY_STUFF

添加唯一值

运算符将值添加到变量中的值列表中，但前提是该值尚不存在。这可以防止值多次包含在变量中。例如：*=

```
DEFINES *= USE_MY_STUFF
```

在上行中，只有在尚未定义的情况下才会添加到预处理器定义的列表中。请注意，unique () 函数也可用于确保变量仅包含每个值的一个实例。USE_MY_STUFF

替换值

运算符将匹配正则表达式的任何值替换为指定值：~=

```
DEFINES ~= s/QT_[DT].+/QT
```

在上行中，列表中以 or 开头的任何值都将替换为。QT_DQT_TQT

可变扩展

运算符用于提取变量的内容，并可用于在变量之间传递值或将它们提供给函数：$$

```
EVERYTHING = $$SOURCES $$HEADERS
message("The project contains the following files:")
message($$EVERYTHING)
```

变量可用于存储环境变量的内容。这些可以在运行 qmake 时进行评估，也可以包含在生成的 Makefile 中，以便在生成项目时进行评估。

若要在运行 qmake 时获取环境值的内容，请使用运算符：$$(...)

```
DESTDIR = $$(PWD)
message(The project will be installed in $$DESTDIR)
```

在上面的赋值中，环境变量的值是在处理项目文件时读取的。PWD

若要在处理生成的生成文件时获取环境值的内容，请使用运算符：$(...)

```
DESTDIR = $$(PWD)
message(The project will be installed in $$DESTDIR)
```

```
message(The project will be installed in the value of PWD)
message(when the Makefile is processed.)
```

在上面的赋值中，值 of 在处理项目文件时立即读取，但分配给生成的 Makefile。只要在处理生成文件时正确设置环境变量，生成过程就会更加灵活。PWD$(PWD)DESTDIR

访问 qmake 属性

特殊运算符可用于访问 qmake 属性：$$[...]

```
message(Qt version: $$[QT_VERSION])
message(Qt is installed in $$[QT_INSTALL_PREFIX])
message(Qt resources can be found in the following locations:)
message(Documentation: $$[QT_INSTALL_DOCS])
message(Header files: $$[QT_INSTALL_HEADERS])
message(Libraries: $$[QT_INSTALL_LIBS])
message(Binary files (executables): $$[QT_INSTALL_BINS])
message(Plugins: $$[QT_INSTALL_PLUGINS])
message(Data files: $$[QT_INSTALL_DATA])
message(Translation files: $$[QT_INSTALL_TRANSLATIONS])
message(Settings: $$[QT_INSTALL_CONFIGURATION])
message(Examples: $$[QT_INSTALL_EXAMPLES])
```

有关更多信息，请参阅配置 qmake。

使用此运算符可访问的属性通常用于使第三方插件和组件能够集成到Qt中。例如，*如果Qt*Designer的项目文件中进行了以下声明，则可以与*Qt Designer*的内置插件一起安装：

```
target.path = $$[QT_INSTALL_PLUGINS]/designer
INSTALLS += target
```

范围

作用域类似于过程编程语言中的语句。如果某个条件为 true，则处理作用域内的声明。if

作用域语法

作用域由一个条件组成，后跟同一行上的左大括号、一系列命令和定义以及新行上的右大括号：

```
<condition> {
    <command or definition>
    ...
}
```

范围和条件

作用域被写为一个条件，后跟一对大括号中包含的一系列声明。例如：

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

上面的代码会在为 Windows 平台构建时将该文件添加到生成的生成文件中列出的源中。为其他平台构建时，将忽略该定义。paintwidget_win.cpp

也可以对给定范围中使用的条件进行否定，以提供一组替代声明，仅当原始条件为 false 时才处理这些声明。例如，要在为*除 Windows 以外的*所有平台构建时处理某些内容，请像这样否定范围：

```
!win32 {
    SOURCES -= paintwidget_win.cpp
}
```

范围可以嵌套以组合多个条件。例如，要仅在启用调试时才包含特定平台的特定文件，请编写以下内容：

```
macx {
    CONFIG(debug, debug|release) {
        HEADERS += debugging.h
    }
}
```

若要节省写入许多嵌套作用域，可以使用运算符嵌套作用域。可以通过以下方式重写上述示例中的嵌套作用域：：

```
macx:CONFIG(debug, debug|release) {
    HEADERS += debugging.h
}
```

您也可以使用运算符来执行单行条件分配。例如：：

```
win32:DEFINES += USE_MY_STUFF
```

上行仅在为 Windows 平台构建时添加到DEFINE变量。通常，运算符的行为类似于逻辑 AND 运算符，将许多条件连接在一起，并要求所有条件都为真。USE_MY_STUFF：

还有运算符充当逻辑 OR 运算符，将许多条件连接在一起，并且只需要其中一个条件为真。 |

```
    HEADERS += debugging.h
}
```

如果需要混合使用这两个运算符，可以使用函数指定运算符优先级。`if`

```
if(win32|macos):CONFIG(debug, debug|release) {
    # Do something on Windows and macOS,
    # but only for the debug configuration.
}
win32|if(macos:CONFIG(debug, debug|release)) {
    # Do something on Windows (regardless of debug or release)
    # and on macOS (only for debug).
}
```

条件接受通配符以匹配一系列值或 mkspec 名称。`CONFIG`

```
win32-* {
    # Matches every mkspec starting with "win32-"
    SOURCES += win32_specific.cpp
}
```

> **注意：**从历史上看，使用上述通配符检查 mkspec 名称是 qmake 检查平台的方式。如今，我们建议使用由 `QMAKE_PLATFORM` 变量中的 mkspec 定义的值。

还可以使用 anscope 为范围内的声明提供替代声明。如果上述作用域的条件为 false，则处理每个作用域。这允许您在与其他作用域结合使用时编写复杂的测试（如上所述由运算符分隔）。例如：`elseelse:`

```
win32:xml {
    message(Building for Windows)
    SOURCES += xmlhandler_win.cpp
} else:xml {
    SOURCES += xmlhandler.cpp
} else {
    message("Unknown configuration")
}
```

配置和作用域

存储在CONFIG变量中的值由 qmake 专门处理。每个可能的值都可以用作范围的条件。例如，持有的值列表可以使用以下值进行扩展：`CONFIGopengl`

```
CONFIG += opengl
```

作为此操作的结果，将处理测试的任何范围。我们可以使用此功能为最终的可执行文件指定一个适当的名称：opengl

```
opengl {
    TARGET = application-gl
} else {
    TARGET = application
}
```

此功能使您可以轻松更改项目的配置，而不会丢失特定配置可能需要的所有自定义设置。在上面的代码中，处理第一个作用域中的声明，并调用最终的可执行文件。但是，如果未指定 ifis，则会改为处理第二个作用域中的声明，并调用最终的可执行文件。application-glopenglapplication

由于可以将自己的值放在线上，因此这为您提供了一种自定义项目文件和微调生成的 Makefile 的便捷方法。CONFIG

平台范围值

除了在许多作用域条件中使用的、、和值之外，还可以使用作用域测试其他各种内置平台和编译器特定的值。这些基于Qt目录中提供的平台规范。例如，项目文件中的以下行显示当前使用的规范并测试规范：win32macxunixmkspecslinux-g++


```
linux-g++ {
    message(Linux)
}
```

您可以测试任何其他平台编译器组合，只要目录中存在它的规范。mkspecs

变量

项目文件中使用的许多变量都是 qmake 在生成生成文件时使用的特殊变量，例如定义、源和标头。此外，您可以创建变量供自己使用。qmake 在遇到对该名称的赋值时会创建具有给定名称的新变量。例如：

```
MY_VARIABLE = value
```

对于您对自己的变量执行的操作没有任何限制，因为 qmake 将忽略它们，除非它在处理作用域时需要评估它们。

您还可以通过将当前变量的值作为变量名称的前缀 $$ 来将当前变量的值分配给另一个变量。例如：

```
MY_DEFINES = $$DEFINES
```

```
MY_DEFINES = $${DEFINES}
```

第二种表示法允许您将变量的内容附加到另一个值，而无需用空格分隔两者。例如，以下内容将确保最终可执行文件的名称包含正在使用的项目模板：

```
TARGET = myproject_$${TEMPLATE}
```

替换函数

qmake提供了一系列内置函数，以允许处理变量的内容。这些函数处理提供给它们的参数，并返回值或值列表作为结果。要将结果分配给变量，请将运算符与这种类型的函数一起使用，就像将一个变量的内容分配给另一个变量一样：$$

```
HEADERS = model.h
HEADERS += $$OTHER_HEADERS
HEADERS = $$unique(HEADERS)
```

这种类型的函数应该在赋值的右侧使用（即，作为操作数）。

您可以定义自己的函数来处理变量的内容，如下所示：

```
defineReplace(functionName){
    #function code
}
```

以下示例函数将变量名称作为其唯一参数，使用eval（） 内置函数从变量中提取值列表，并编译文件列表：

```
defineReplace(headersAndSources) {
    variable = $$1
    names = $$eval($$variable)
    headers =
    sources =

    for(name, names) {
        header = $${name}.h
        exists($$header) {
            headers += $$header
        }
        source = $${name}.cpp
        exists($$source) {
            sources += $$source
```

```
}
```

测试功能

QMAKE提供了内置函数，可以在编写作用域时用作条件。这些函数不返回值，而是指示*成功*或*失败*:

```
count(options, 2) {
    message(Both release and debug specified.)
}
```

这种类型的函数应仅在条件表达式中使用。

可以定义自己的函数来为作用域提供条件。下面的示例测试列表中的每个文件是否存在，如果它们都存在，则返回 true;如果不存在，则返回 false：

```
defineTest(allFiles) {
    files = $$ARGS

    for(file, files) {
        !exists($$file) {
            return(false)
        }
    }
    return(true)
}
```

‹ 平台说明 高级用法 ›

编辑部
职业
办公地点

常见问题

支持

支持服务
专业服务
合作 伙伴
训练

对于客户

支持中心
下载

Qt登录
联系我们
客户成功案例

社区

为Qt做贡献
论坛

维基
下载
市场

反馈 登录