搜索

高级用法

添加新的配置功能

qmake 允许您创建自己的可以通过将项目文件的名称添加到CONFIG变量指定的值列表中来包含在项目文件中。功能是文件中的自定义函数和定义的集合，这些文件可以驻留在许多标准目录之一中。这些目录的位置在许多地方定义，qmake 在查找文件时按以下顺序检查每个目录：features.prf.prf

1. 在环境变量中列出的目录中，该目录包含由平台的路径列表分隔符（Unix 的冒号，Windows 的分号）分隔的目录列表。QMAKEFEATURES
2. 在属性变量中列出的目录中，该变量包含由平台的路径列表分隔符分隔的目录列表。QMAKEFEATURES
3. 在位于 adirectory.directory 中的功能目录中，目录可以位于环境变量中列出的任何目录下，该变量包含由平台的路径列表分隔符分隔的目录列表。例如：。mkspecsmkspecsQMAKEPATH$QMAKEPATH/mkspecs/<features>
4. 位于QMAKESPEC环境变量提供的目录下的功能目录中。例如：。$QMAKESPEC/<features>
5. 在驻留在目录中的功能目录中。例如：。data_install/mkspecsdata_install/mkspecs/<features>
6. 在作为环境变量指定的目录的同级存在的功能目录中。例如：。QMAKESPEC$QMAKESPEC/../<features>

在以下功能目录中搜索功能文件：

1. features/unix，，或，具体取决于所使用的平台features/win32features/macx
2. features/

例如，考虑项目文件中的以下分配：

```
CONFIG += myfeatures
```

通过对变量的添加，qmake 将在完成项目文件解析后在上面列出的位置搜索文件。在Unix系统上，它将查找以下文件：CONFIGmyfeatures.prf

1. $QMAKEFEATURES/myfeatures.prf（对于环境变量中列出的每个目录）QMAKEFEATURES
2. $$QMAKEFEATURES/myfeatures.prf（对于属性变量中列出的每个目录）QMAKEFEATURES
3. myfeatures.prf（在项目的根目录中）。项目根目录由顶级文件确定。但是，如果将文件放在子目录或子项目的目录中，则项目根目录将成为子目录本身。.pro.qmake.cache
4. $QMAKEPATH/mkspecs/features/unix/myfeatures.prfand（对于环境变量中列出的每个目录）$QMAKEPATH/mkspecs/features/myfeatures.prfQMAKEPATH
5. $QMAKESPEC/features/unix/myfeatures.prf和$QMAKESPEC/features/myfeatures.prf
6. data_install/mkspecs/features/unix/myfeatures.prf和data_install/mkspecs/features/myfeatures.prf
7. $QMAKESPEC/../features/unix/myfeatures.prf和$QMAKESPEC/../features/myfeatures.prf

> **注意：**文件的名称必须为小写。.prf

安装文件

有关项目的一部分安装方式的说明。例如，文档文件的集合可以通过以下方式描述：make installinstall set

```
documentation.path = /usr/local/program/doc
documentation.files = docs/*
```

成员通知 qmake 文件应该安装在（路径成员），成员指定应该复制到安装目录的文件。在这种情况下，目录中的所有内容都将复制到。path/usr/local/program/docfilesdocs/usr/local/program/doc

完整描述安装集后，您可以使用如下行将其附加到安装列表中：

```
INSTALLS += documentation
```

qmake将确保将指定的文件复制到安装目录。如果需要对此过程进行更多控制，还可以为对象的成员提供定义。例如，以下行告诉 qmake 为此安装集执行一系列命令：extra

```
unix:documentation.extra = create_docs; mv master.doc toc.doc
```

该范围确保这些特定命令仅在 Unix 平台上执行。可以使用其他范围规则定义适用于其他平台的命令。unix

成员中指定的命令在执行对象的其他成员中的指令之前执行。extra

如果您将内置安装集附加到变量并且不指定成员，qmake 将决定需要为您复制的内容。目前，支持和安装集。例如：INSTALLSfilesextratargetdlltarget

```
target.path = /usr/local/myprogram
INSTALLS += target
```

在上面的行中，qmake知道需要复制什么，并且会自动处理安装过程。

添加自定义目标

Topics >

Makefile 输出的自定义是通过对象样式的 API 执行的，就像在 qmake 中的其他地方一样。通过*指定对象的成员*自动定义对象。例如：

```
mytarget.target = .buildfile
mytarget.commands = touch $$mytarget.target
mytarget.depends = mytarget2

mytarget2.commands = @echo Building $$mytarget.target
```

上面的定义定义了一个调用的 qmake 目标，其中包含一个调用的 Makefile 目标，而该目标又是用命令生成的。最后，成员指定依赖于另一个目标，该目标 afterwards.is 虚拟目标定义。它仅定义为将某些文本回显到控制台。mytarget.buildfiletouch.dependsmytargetmytarget2mytarget2

最后一步是使用变量来指示 qmake 此对象是要构建的目标：QMAKE_EXTRA_TARGETS

```
QMAKE_EXTRA_TARGETS += mytarget mytarget2
```

目标包含在PRE_TARGETDEPS列表中。

自定义目标规范支持以下成员：

成员	描述
命令	用于生成自定义生成目标的命令。
配置	自定义生成目标的特定配置选项。可以设置为 以指示应在生成文件中创建规则，以调用子目标特定生成文件内的相关目标。此成员默认为每个子目标创建一个条目。recursive
取决于	自定义生成目标所依赖的现有生成目标。
递归	指定在生成文件中创建规则以在子目标特定的生成文件中调用时应使用哪些子目标。此成员仅在设置时使用。典型值为“调试”和“发布”。recursiveCONFIG
recurse_target	指定应通过子目标生成文件为生成文件中的规则生成的目标。此成员添加类似内容。此成员仅在设置时使用。$(MAKE) -f Makefile.[subtarget] [recurse_target]recursiveCONFIG
目标	自定义生成目标的名称。

添加编译器

可以自定义 qmake 以支持新的编译器和预处理器：

```
new_moc.output  = moc_${QMAKE_FILE_BASE}.cpp
new_moc.commands = moc ${QMAKE_FILE_NAME} -o ${QMAKE_FILE_OUT}
new_moc.depend_command = g++ -E -M ${QMAKE_FILE_NAME} | sed "s,^.*: ,,"
new_moc.input = NEW_HEADERS
QMAKE_EXTRA_COMPILERS += new_moc
```

使用上述定义，您可以使用 moc 的直接替代品（如果有）。该命令在给定变量的所有参数上执行（来自成员），结果将写入成员定义的文件。此文件将添加到项目中的其他源文件中。此外，qmake 将执行以生成依赖项信息，并将此信息也放在项目中。NEW_HEADERSinputoutputdepend_command

自定义编译器规范支持以下成员：

成员	描述
命令	用于从输入生成输出的命令。
配置	自定义编译器的特定配置选项。有关详细信息，请参阅配置表。
depend_command	指定用于生成输出依赖项列表的命令。
dependency_type	指定输出的文件类型。如果它是已知类型（如 TYPE_C、TYPE_UI、TYPE_QRC），则会将其作为这些类型的文件之一进行处理。
取决于	指定输出文件的依赖项。
输入	指定应使用自定义编译器处理的文件的变量。
名字	自定义编译器正在执行的操作的说明。这仅在某些后端使用。
输出	从自定义编译器创建的文件名。
output_function	指定用于指定要创建的文件名的自定义 qmake 函数。
变量	指示此处指定的变量在 pro 文件中称为 $（VARNAME） 时替换为 $（QMAKE_COMP_VARNAME）。
variable_out	应将从输出创建的文件添加到的变量。

CONFIG 成员支持以下选项：

选择	描述
合	指示将所有输入文件合并为单个输出文件。
target_predeps	指示应将输出添加到PRE_TARGETDEPS列表中。

dep_existing_only	作为.depend_command结果的每个依赖项都会检查是否存在。不存在的依赖项将被忽略。此值在 Qt 5.13.2 中引入。
dep_lines	.depend_command的输出被解释为每行一个文件。默认值是在空格上拆分，并且仅出于向后兼容性原因而保留。
no_link	指示不应将输出添加到要链接的对象列表中。

库依赖项

通常，当链接到库时，qmake 依赖于底层平台来了解该库链接到哪些其他库，并让平台将它们拉入。然而，在许多情况下，这还不够。例如，静态链接库时，不会链接到其他库，因此不会创建对这些库的依赖项。但是，稍后链接到此库的应用程序需要知道在哪里可以找到静态库所需的符号。如果您明确启用跟踪，qmake 会尝试在适当的情况下跟踪库的依赖项。

第一步是在库本身中启用依赖项跟踪。为此，您必须告诉 qmake 保存有关库的信息：

```
CONFIG += create_prl
```

这仅与模板相关，对于所有其他模板，将被忽略。启用此选项后，qmake 将创建一个以 .prl 结尾的文件，该文件将保存有关库的一些元信息。此图元文件就像普通的项目文件一样，但仅包含内部变量声明。安装此库时，通过在INSTALLS声明中将其指定为目标，qmake 会自动将 .prl 文件复制到安装路径。lib

此过程的第二步是在使用静态库的应用程序中读取此元信息：

```
CONFIG += link_prl
```

启用此功能后，qmake 将处理应用程序链接到的所有库并查找它们的元信息。qmake 将使用它来确定相关的链接信息，特别是将值添加到应用程序项目文件的DEFINE和LIBS 列表中。一旦 qmake 处理了这个文件，它将查看变量中新引入的库，并找到它们的依赖 .prl 文件，继续直到解析完所有库。此时，将照常创建生成文件，并且库将针对应用程序显式链接。LIBS

.prl文件应仅由qmake创建，不应在操作系统之间传输，因为它们可能包含与平台相关的信息。

‹ qmake语言 使用预编译标头 ›

支持服务
专业服务
合作 伙伴
训练

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