

Qt 6.4 > Qmake手册 > [测试功能](#)

测试功能

测试函数返回一个布尔值，您可以在作用域的条件部分中测试该值。测试函数可以分为内置函数和函数库。

另请参阅[替换函数](#)。

内置测试功能

基本测试功能作为内置函数实现。

`cache (variablename, [set|add|sub] [transient] [super|stash], [source variablename])`

这是您通常不需要的内部函数。

此功能在Qt 5.0中引入。

配置（配置）

此函数可用于测试放入`CONFIG`变量中的变量。这与作用域相同，但具有额外的优势，即可以传递第二个参数来测试活动配置。由于值的顺序是重要的变量（即，最后一个集合将被视为互斥值的活动配置），因此可以使用第二个参数来指定要考虑的值集。例如：`CONFIG`

```
CONFIG = debug
CONFIG += release
CONFIG(release, debug|release):message(Release build!) #will print
CONFIG(debug, debug|release):message(Debug build!) #no print
```

由于发布被视为活动设置（用于功能分析），因此它将是用于生成构建文件的`CONFIG`。在通常情况下，不需要第二个参数，但对于特定的互斥测试，它是无价的。

包含（变量名、值）

如果变量包含值，则成功；否则失败。可以为参数`value`指定正则表达式。`variablenamevalue`

您可以使用作用域检查此函数的返回值。

```
contains( drivers, network ) {
    # drivers contains 'network'
    message( "Configuring for network build..." )
    HEADERS += network.h
    SOURCES += network.cpp
}
```

仅当变量包含值时，才会处理范围的内容。如果是这种情况，则会将相应的文件添加到SOURCES和HEADERS变量中。driversnetwork

计数（变量名、数字）

如果变量包含具有指定值的列表，则成功;否则失败。variablenamenumbers

此函数用于确保仅当变量包含正确数量的值时，才处理作用域内的声明。例如：

```
options = $$find(CONFIG, "debug") $$find(CONFIG, "release")
count(options, 2) {
    message(Both release and debug specified.)
}
```

调试（级别，消息）

检查 qmake 是否在指定的调试级别运行。如果是，则返回 true 并打印调试消息。

已定义（名称[, 类型]）

测试是否定义了函数或变量。省略 ifis，检查所有功能。要仅检查变量或特定类型的函数，请指定。它可以具有以下值：nametypetype

- › test仅检查测试功能
- › replace仅检查替换函数
- › var仅检查变量

等于（变量名、值）

测试字符串是否相等。variablenamevalue

例如：

```
TARGET = helloworld
equals(TARGET, "helloworld") {
    message("The target assignment was successful.")
}
```

例如：

```
error(An error has occurred in the configuration process.)
```

评估（字符串）

使用 qmake 语法规则计算字符串的内容并返回 true。可以在字符串中使用定义和赋值来修改现有变量的值或创建新定义。

例如：

```
eval(TARGET = myapp) {  
    message($$TARGET)  
}
```

注意：引号可用于分隔字符串，如果不需要返回值，可以丢弃返回值。

存在（文件名）

测试具有给定的文件是否存在。如果文件存在，则函数成功；否则它将失败。filename

参数可能包含通配符。在这种情况下，如果任何文件匹配，此函数将成功。filename

例如：

```
exists( $(QTDIR)/lib/libqt-mt* ) {  
    message( "Configuring for multi-threaded Qt..." )  
    CONFIG += thread  
}
```

注意：“/”应用作目录分隔符，无论使用何种平台。

导出（变量名）

将函数的本地上下文中的当前值导出到全局上下文。variablename

for（迭代，列表）

启动循环访问中的所有值，依次设置每个值。为方便起见，ifis 1..10 将迭代值 1 到 10。listiteratelist

例如：

循环可以中断。该语句跳过循环主体的其余部分，并在下一次迭代中继续执行。break()next()

大于（变量名、值）

测试的值大于。首先，此函数尝试进行数值比较。如果至少一个操作数转换失败，此函数将执行字符串比较。
variablenamevalue

例如：

```
ANSWER = 42
greaterThan(ANSWER, 1) {
    message("The answer might be correct.")
}
```

不可能直接将两个数字作为字符串进行比较。解决方法是使用非数字前缀构造临时值并比较这些值。

例如：

```
VALUE = 123
TMP_VALUE = x$$VALUE
greaterThan(TMP_VALUE, x456): message("Condition may be true.")
```

另请参阅lessThan () 。

如果（条件）

评估。它用于对布尔表达式进行分组。condition

例如：

```
if(linux-g++|macx-g++):CONFIG(debug, debug|release) {
    message("We are on Linux or Mac OS, and we are in debug mode.")
}
```

包含（文件名）

包括当前项目中指定的文件的内容。此函数成功包含 ifis;否则它将失败。将立即处理包含的文件。
filenamefilename

可以通过使用此函数作为范围的条件来检查是否包含该文件。例如：

```
include( shared.pri )
OPTIONS = standard custom
!include( options.pri ) {
```

infile (文件名, var, val)

如果文件（由 qmake 本身解析时）包含值为 ; 否则失败。如果未指定，该函数将测试是否已在文件中分配。
filenamevarvalvalvar

isActiveConfig

这是函数的别名。CONFIG

isEmpty (变量名)

如果变量为空，则成功;否则失败。这相当于。variablenamecount(variablename, 0)

例如：

```
isEmpty( CONFIG ) {  
    CONFIG += warn_on debug  
}
```

是平等的

这是函数的别名。equals

lessThan (变量名、值)

测试 的值小于。工作为大于 () 。variablenamevalue

例如：

```
ANSWER = 42  
lessThan(ANSWER, 1) {  
    message("The answer might be wrong.")  
}
```

负载 (功能)

加载 指定的功能部件文件 () , 除非已装入该要素。prffeature

日志 (消息)

在控制台上打印消息。与函数不同，它既不预置文本，也不附加换行符。message

此功能在Qt 5.0中引入。

另请参阅消息 () 。

始终成功，并显示为向用户发送的常规消息。与函数不同，此功能允许继续处理。stringerror()

```
message( "This is a message" )
```

上面的行会导致将“这是一条消息”写入控制台。引号的使用是可选的，但建议使用。

注意：默认情况下，将为 qmake 为给定项目生成的每个生成文件写出消息。如果要确保每个项目的消息只出现一次，请结合范围测试变量，以便在生成期间筛选出消息。例如：build_pass

```
!build_pass:message( "This is a message" )
```

mkpath (dirPath)

创建目录路径。此函数是QDir：mkpath函数的包装器。dirPath

此功能在Qt 5.0中引入。

要求（条件）

评估。如果条件为 false，qmake 在构建时会跳过此项目（及其SUBDIRS）。condition

注意：您也可以使用REQUIRE变量来实现此目的。但是，我们建议改用此功能。

系统（命令）

在辅助外壳中执行给定的。如果命令返回零退出状态，则成功;否则失败。您可以使用作用域检查此函数的返回值。command

例如：

```
system("ls /bin"): HAS_BIN = TRUE
```

另请参阅system () 的替换变体。

触摸（文件名，reference_filename）

更新时间戳以匹配的时间戳。filenamereference_filename

此功能在Qt 5.0中引入。

未设置（变量名）

从当前上下文中删除。variablename

```
NARF = zort
unset(NARF)
!defined(NARF, var) {
    message("NARF is not defined.")
}
```

版本至少（变量名，版本号）

测试版本号是否大于或等于。版本号被视为由“.”分隔的非负十进制数序列;字符串的任何非数字尾部都将被忽略。比较从左到右分段执行;如果一个版本是另一个版本的前缀，则认为它较小。

variablenameversionNumber

此功能在Qt 5.10中引入。

版本最多（变量名，版本号）

测试版本号是否小于或等于。作为版本至少（）工作。variablenameversionNumber

此功能在Qt 5.10中引入。

警告（字符串）

始终成功，并向用户显示警告消息。string

write_file（文件名，[变量名，[模式]]）

将 的值写入具有名称的文件，每个值在单独的行上。未指定 ifis，创建一个空文件。Ifis和文件已经存在，附加到它而不是替换它。variablenamefilenamevariablenamemodeappend

此功能在Qt 5.0中引入。

测试函数库

复杂的测试功能在.prf文件库中实现。

包存在（包）

使用 PKGCONFIG 机制来确定在项目解析时给定的包是否存在。

这对于选择性地启用或禁用功能非常有用。例如：

```
packagesExist(sqlite3 QtNetwork QtDeclarative) {
    DEFINES += USE_FANCY_UI
}
```

然后，在代码中：

```
#endif
```

prepareRecursiveTarget (target)

通过准备一个遍历所有子目录的目标，促进创建类似于目标的项目范围目标。例如：install

[Topics >](#)

```
prepareRecursiveTarget(check)
```

在其 .CONFIG 中指定或指定的子目录将从此目标中排除：have_no_defaultno_<target>_target

```
two.CONFIG += no_check_target
```

您必须手动将准备好的目标添加到QMAKE_EXTRA_TARGETS：

```
QMAKE_EXTRA_TARGETS += check
```

为了使目标全局化，需要将上面的代码包含在每个子目录中。此外，为了使这些目标执行任何操作，非子目录子项目需要包含相应的代码。实现此目的的最简单方法是创建自定义功能文件。例如：

```
# <project root>/features/mycheck.prf
equals(TEMPLATE, subdirs) {
    prepareRecursiveTarget(check)
} else {
    check.commands = echo hello user
}
QMAKE_EXTRA_TARGETS += check
```

功能文件需要注入到每个子项目中，例如通过 .qmake.conf：

```
# <project root>/ .qmake.conf
CONFIG += mycheck
```

此功能在Qt 5.0中引入。

qt编译测试 (测试)

生成测试项目 如果测试通过 则返回 true 并将其添加到CONFIG变量中 否则 返回 false


```
# <project root>/project.pro
load(configure)
```

这还会将变量QMAKE_CONFIG_TESTS_DIR设置为项目父目录的子目录。加载功能文件后可以覆盖此值。
config.tests

在测试目录中，每个测试必须有一个包含简单 qmake 项目的子目录。以下代码片段说明了项目的 .pro 文件：

```
# <project root>/config.tests/test/test.pro
SOURCES = main.cpp
LIBS += -ltheFeature
# Note that the test project is built without Qt by default.
```

以下代码片段说明了该项目的主.cpp文件：

```
// <project root>/config.tests/test/main.cpp
#include <TheFeature/MainHeader.h>
int main() { return featureFunction(); }
```

以下代码片段显示了测试的调用：

```
# <project root>/project.pro
qtCompileTest(test)
```

如果测试项目生成成功，则测试通过。

测试结果会自动缓存，这也使它们可用于所有子项目。因此，建议在顶级项目文件中运行所有配置测试。

要禁止重用缓存的结果，请传递给 qmake。CONFIG+=recheck

另请参阅[加载 \(\)](#)。

此功能在Qt 5.0中引入。

qtHaveModule (name)

检查指定的 Qt 模块是否存在。有关可能值的列表，请参阅[QT](#)。name

此功能在Qt 5.0.1中引入。

< 替换函数



联系我们

公司

- 关于我们
- 投资者
- 编辑部
- 职业
- 办公地点

发牌

- 条款和条件
- 开源
- 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