目录

第 1 章　QBoxLayout

1.1　void QBoxLayout::addStretch(int stretch = 0)

Adds a stretchable space (a QSpacerItem) with zero minimum size and stretch factor stretch to the end of this box layout.

函数的作用是在布局器中增加一个伸缩量，里面的参数表示 QSpacerItem 的个数，默认值为零，会将你放在 layout 中的空间压缩成默认的大小。

例如：一个 layout 布局器，里面有三个控件，一个放在最左边，一个放在最右边，最后一个放在 layout 的 1/3 处，这就可以通过 addStretch 去实现。

例子：用 addStretch 函数实现将 nLayout 的布局器的空白空间平均分配：

```
QHBoxLayout *buttonLayout=new QHBoxLayout;
button1=new QPushButton();
button2=new QPushButton();
button3=new QPushButton();
buttonLayout->addStretch(1);  //增加伸缩量
buttonLayout->addWidget(button1);
buttonLayout->addStretch(1);  //增加伸缩量
buttonLayout->addWidget(button2);
buttonLayout->addStretch(1);  //增加伸缩量
buttonLayout->addWidget(button3);
buttonLayout->addStretch(6);  //增加伸缩量
//void QWidget::setContentsMargins(int left, int top,
   int right, int bottom)
//Sets the margins around the contents of the widget
   to have the sizes left, top, right, and bottom.
//The margins are used by the layout system, and may
   be used by subclasses to specify the area to draw
```

```
28        in (e.g. excluding the frame).
29    buttonLayout->setContentsMargins(0,0,0,0);
30    setLayout(buttonLayout);
```

31 其中四个 addStretch() 函数用于在 button 按钮间增加伸缩量，伸缩量的比例
32 为 1:1:1:6，意思就是将 button 以外的空白地方按设定的比例等分为 9 份并按
33 照设定的顺序放入 buttonLayout 布局器中。

第 2 章　QCheckBox

2.1　void setTristate(bool y = true)

2.2　bool isTristate() const

查询和设定 checkbox 是否具有三种状态，默认的情况下是只有两种状态。

2.3　void QCheck-Box::setCheckState(Qt::CheckState state)

将 checkbox 的状态设定为 state。如果我们在程序中不需要使用三状态的 checkbox，则我们可以调用 setChecked(bool state) 函数来替代。

第 3 章　QEvent

QEvent::Type 中事件的类型，每个事件类型和每个类型的专门类如下.

- QEvent::None: 不是一个事件
- QEvent::ActionAdded：一个新的 action 被添加 (QActionEvent)
- QEvent::ActionChanged：一个 action 被改变 (QActionEvent)
- QEvent::ActionRemoved：一个 action 被移除 (QActionEvent)
- QEvent::ActivationChange：Widget 的顶层窗口激活状态发生了变化
- QEvent::ApplicationFontChange：应用程序的字体发生了改变
- QEvent::ApplicationLayoutDirectionChange：应用程序的布局方向发生了改变
- QEvent::ApplicationPaletteChange：应用程序的默认调色板发生了变化
- QEvent::ApplicationStateChange：应用程序的状态发生了变化
- QEvent::ApplicationWindowIconChange：应用程序的图标发生了变化
- QEvent::ChildAdded：一个对象获得孩子（QChildEvent）
- QEvent::ChildPolished：一个部件的孩子被抛光（QChildEvent）
- QEvent::ChildRemoved：一个对象去除孩子（QChildEvent）
- QEvent::Clipboard：剪贴板的内容发生改变
- QEvent::Close：Widget 被关闭（QCloseEvent）
- QEvent::CloseSoftwareInputPanel：一个部件要关闭软件输入面板
- QEvent::ContentsRectChange：部件内容区域的外边距发生改变

- QEvent::ContextMenu：上下文弹出菜单（QContextMenuEvent）
- QEvent::CursorChange：部件的鼠标发生改变
- QEvent::DeferredDelete：对象被清除后将被删除（QDeferredDeleteEvent）
- QEvent::DragEnter：在拖放操作期间鼠标进入窗口部件（QDragEnterEvent）
- QEvent::DragLeave：在拖放操作期间鼠标离开窗口部件（QDragLeaveEvent）
- QEvent::DragMove：拖放操作正在进行（QDragMoveEvent）
- QEvent::Drop：拖放操作完成（QDropEvent）
- QEvent::DynamicPropertyChange：动态属性已添加、更改或从对象中删除
- QEvent::EnabledChange：部件的 enabled 状态已更改
- QEvent::Enter：鼠标进入部件的边界（QEnterEvent）
- QEvent::EnterEditFocus：编辑部件获得焦点进行编辑
- QEvent::EnterWhatsThisMode：当应用程序进入 “What’ s This?” 模式，发送到 toplevel 顶层部件
- QEvent::Expose：当其屏幕上的内容无效，发送到窗口，并需要从后台存储刷新
- QEvent::FileOpen：文件打开请求（QFileOpenEvent）
- QEvent::FocusIn：部件或窗口获得键盘焦点（QFocusEvent）
- QEvent::FocusOut：部件或窗口失去键盘焦点（QFocusEvent）
- QEvent::FocusAboutToChange：部件或窗口焦点即将改变（QFocusEvent）
- QEvent::FontChange：部件的字体发生改变
- QEvent::Gesture：触发了一个手势（QGestureEvent）
- QEvent::GestureOverride：触发了手势覆盖（QGestureEvent）

- QEvent::GrabKeyboard：Item 获得键盘抓取（仅限 QGraphicsItem）
- QEvent::GrabMouse：项目获得鼠标抓取（仅限 QGraphicsItem）
- QEvent::GraphicsSceneContextMenu：在图形场景上的上下文弹出菜单（QGraphicsScene ContextMenuEvent）
- QEvent::GraphicsSceneDragEnter：在拖放操作期间，鼠标进入图形场景（QGraphicsSceneDragDropEvent）
- QEvent::GraphicsSceneDragLeave：在拖放操作期间鼠标离开图形场景（QGraphicsSceneDragDropEvent）
- QEvent::GraphicsSceneDragMove：在场景上正在进行拖放操作（QGraphicsSceneDragDropEvent）
- QEvent::GraphicsSceneDrop：在场景上完成拖放操作（QGraphicsSceneDragDropEvent）
- QEvent::GraphicsSceneHelp：用户请求图形场景的帮助（QHelpEvent）
- QEvent::GraphicsSceneHoverEnter：鼠标进入图形场景中的悬停项（QGraphicsSceneHoverEvent）
- QEvent::GraphicsSceneHoverLeave：鼠标离开图形场景中一个悬停项（QGraphicsSceneHoverEvent）
- QEvent::GraphicsSceneHoverMove：鼠标在图形场景中的悬停项内移动（QGraphicsSceneHoverEvent）
- QEvent::GraphicsSceneMouseDoubleClick：鼠标在图形场景中再次按下（双击）（QGraphicsSceneMouseEvent）
- QEvent::GraphicsSceneMouseMove：鼠标在图形场景中移动（QGraphicsSceneMouseEvent）
- QEvent::GraphicsSceneMousePress：鼠标在图形场景中按下（QGraphicsSceneMouseEvent）

- QEvent::GraphicsSceneMouseRelease：鼠标在图形场景中释放（QGraphicsSceneMouseEvent）
- QEvent::GraphicsSceneMove：部件被移动（QGraphicsSceneMoveEvent）
- QEvent::GraphicsSceneResize：部件已调整大小（QGraphicsSceneResizeEvent）
- QEvent::GraphicsSceneWheel：鼠标滚轮在图形场景中滚动（QGraphicsSceneWheelEvent）
- QEvent::Hide：部件被隐藏（QHideEvent）
- QEvent::HideToParent：子部件被隐藏（QHideEvent）
- QEvent::HoverEnter：鼠标进入悬停部件（QHoverEvent）
- QEvent::HoverLeave：鼠标留离开悬停部件（QHoverEvent）
- QEvent::HoverMove：鼠标在悬停部件内移动（QHoverEvent）
- QEvent::IconDrag：窗口的主图标被拖走（QIconDragEvent）
- QEvent::InputMethod：正在使用输入法（QInputMethodEvent）
- QEvent::InputMethodQuery：输入法查询事件（QInputMethodQueryEvent）
- QEvent::KeyboardLayoutChange：键盘布局已更改
- QEvent::KeyPress：键盘按下（QKeyEvent）
- QEvent::KeyRelease：键盘释放（QKeyEvent）
- QEvent::LanguageChange：应用程序翻译发生改变
- QEvent::LayoutDirectionChange：布局的方向发生改变
- QEvent::LayoutRequst：部件的布局需要重做
- QEvent::Leave：鼠标离开部件的边界
- QEvent::LeaveEditFocus：编辑部件失去编辑的焦点

- QEvent::LeaveWhatsThisMode：当应用程序离开 “What’ s This?” 模式，发送到顶层部件
- QEvent::LocaleChange：系统区域设置发生改变
- QEvent::NonClientAreaMouseButtonDblClick：鼠标双击发生在客户端区域外
- QEvent::NonClientAreaMouseButtonPress：鼠标按钮按下发生在客户端区域外
- QEvent::NonClientAreaMouseButtonRelease：鼠标按钮释放发生在客户端区域外
- QEvent::NonClientAreaMouseMove：鼠标移动发生在客户区域外
- QEvent::MacSizeChange：用户更改了部件的大小（仅限 OS X）
- QEvent::MetaCall：通过 QMetaObject::invokeMethod() 调用异步方法
- QEvent::ModifiedChange：部件修改状态发生改变
- QEvent::MouseButtonDblClick：鼠标再次按下（QMouseEvent）
- QEvent::MouseButtonPress：鼠标按下（QMouseEvent）
- QEvent::MouseButtonRelease：鼠标释放（QMouseEvent）
- QEvent::MouseMove：鼠标移动（QMouseEvent）
- QEvent::MouseTrackingChange：鼠标跟踪状态发生改变
- QEvent::Move：部件的位置发生改变（QMoveEvent）
- QEvent::NativeGesture：系统检测到手势（QNativeGestureEvent）
- QEvent::OrientationChange：屏幕方向发生改变（QScreenOrientationChangeEvent）
- QEvent::Paint：需要屏幕更新（QPaintEvent）
- QEvent::PaletteChange：部件的调色板发生改变

- QEvent::ParentAboutToChange：部件的 parent 将要更改
- QEvent::ParentChange：部件的 parent 发生改变
- QEvent::PlatformPanel：请求一个特定于平台的面板
- QEvent::PlatformSurface：原生平台表面已创建或即将被销毁（QPlatformSurfaceEvent）
- QEvent::Polish：部件被抛光
- QEvent::PolishRequest：部件应该被抛光
- QEvent::QueryWhatsThis：如果部件有“What’s This?”帮助，应该接受事件
- QEvent::ReadOnlyChange：部件的 read-only 状态发生改变
- QEvent::RequestSoftwareInputPanel：部件想要打开软件输入面板
- QEvent::Resize：部件的大小发生改变（QKeyEvent）
- QEvent::ScrollPrepare：对象需要填充它的几何信息(QScrollPrepareEvent)
- QEvent::Scroll：对象需要滚动到提供的位置（QScrollEvent）
- QEvent::Shortcut：快捷键处理（QShortcutEvent）
- QEvent::ShortcutOverride：按下按键，用于覆盖快捷键（QKeyEvent）
- QEvent::Show：部件显示在屏幕上（QShowEvent）
- QEvent::ShowToParent：子部件被显示
- QEvent::SockAct：Socket 激活，用于实现 QSocketNotifier
- QEvent::StateMachineSignal：信号被传递到状态机(QStateMachine::SignalEvent)
- QEvent::StateMachineWrapped：事件是一个包装器，用于包含另一个事件（QStateMachine::WrappedEvent）
- QEvent::StatusTip：状态提示请求（QStatusTipEvent）

- QEvent::StyleChange：部件的样式发生改变
- QEvent::TabletMove：Wacom 写字板移动（QTabletEvent）
- QEvent::TabletPress：Wacom 写字板按下（QTabletEvent）
- QEvent::TabletRelease：Wacom 写字板释放（QTabletEvent）
- QEvent::TabletEnterProximity：Wacom 写字板进入接近事件(QTabletEvent)，发送到 QApplication
- QEvent::TabletLeaveProximity：Wacom 写字板离开接近事件(QTabletEvent)，发送到 QApplication
- QEvent::TabletTrackingChange：
- QEvent::ThreadChange：对象被移动到另一个线程。这是发送到此对象的最后一个事件在上一个线程中，参见：QObject::moveToThread()
- QEvent::Timer：定时器事件（QTimerEvent）
- QEvent::ToolBarChange：工具栏按钮在 OS X 上进行切换
- QEvent::ToolTip：一个 tooltip 请求（QHelpEvent）
- QEvent::ToolTipChange：部件的 tooltip 发生改变
- QEvent::TouchBegin：触摸屏或轨迹板事件序列的开始（QTouchEvent）
- QEvent::TouchCannel：取消触摸事件序列（QTouchEvent）
- QEvent::TouchEnd：触摸事件序列结束（QTouchEvent）
- QEvent::TouchUpdate：触摸屏事件（QTouchEvent）
- QEvent::UngrabMouse：Item 失去鼠标抓取（QGraphicsItem、QQuickItem）
- QEvent::UpdateLater：部件应该排队在以后重新绘制
- QEvent::UpdateRequest：部件应该被重绘

- QEvent::WhatsThis：部件应该显示“What’s This”帮助（QHelpEvent）
- QEvent::WhatsThisClicked：部件的“What’s This”帮助链接被点击
- QEvent::Wheel：鼠标滚轮滚动（QWheelEvent）
- QEvent::WinEventAct：发生了 Windows 特定的激活事件
- QEvent::WindowActivate：窗口已激活
- QEvent::WindowBlocked：窗口被模态对话框阻塞
- QEvent::WindowDeactivatge：窗户被停用
- QEvent::WindowStateChange：窗口的状态（最小化、最大化或全屏）发生改变（QWindowStateChangeEvent）
- QEvent::WindowTitleChange：窗口的标题发生改变
- QEvent::WindowUnblocked：一个模态对话框退出后，窗口将不被阻塞
- QEvent::WinIdChange：本地窗口的系统标识符发生改变
- QEvent::ZOrderChange：部件的 z 值发生了改变，该事件不会发送给顶层窗口

第 4 章　QGroupBox

4.1　void setFlat(bool flat)

This property holds whether the group box is painted flat or has a frame。决定组合框的垂直边缘是否进行绘画。如果为真，则不进行绘画，如果为假，则进行绘画，缺省值是 false。

第5章　Qss样式表

第 6 章　qmake Manual

The qmake tool helps simplify the build process for development projects across different platforms.It automates the generation of Makefiles so that only a few lines of information are needed to create each Makefile.You can use qmake for any software project, whether it is written with Qt or not.

qmake generates a Makefile based on the information in a project file. Project files are created by the developer, and are usually simple, but more sophisticated project files can be created for complex projects.

qmake contains additional features to support development with Qt, automatically including build rules for moc and uic.qmake can also generate projects for Microsoft Visual studio without requiring the developer to change the project file.

6.1　Overview

The qmake tool provides you with a project-oriented system for managing the build process for applications, libraries, and other components. This approach gives you control over the source files used, and allows each of the steps in the process to be described concisely, typically within a single file. qmake expands the information in each project file to a Makefile that executes the necessary commands for compiling and linking.

6.1.1　Describing a project

Projects are described by the contents of project (.pro) files. qmake uses the information within the files to generate Makefiles that contain all the commands that are needed to build each project. Project files typically contain a list of source and header files, general configuration information, and any application-specific details, such as a list of extra libraries to link against, or a list of extra include paths to use.

Project files can contain a number of different elements, including comments, variable declarations, built-in functions, and some simple control structures. In most simple projects, it is only necessary to declare the source and header files that are used to build the project with some basic configuration options. For more information about how to create a simple project file, see Getting Started.

You can create more sophisticated project files for complex projects. For an overview of project files, see Creating Project Files. For detailed information about the variables and functions that you can use in project files, see Reference.

You can use application or library project templates to specify specialized configuration options to fine tune the build process. For more information, see Building Common Project Types.

You can use the Qt Creator new project wizard to create the project file. You choose the project template, and Qt Creator creates a project file with default values that enable you to build and run the project. You can modify the project file to suit your purposes.

You can also use qmake to generate project files. For a full description of qmake command line options, see Running qmake.

The basic configuration features of qmake can handle most cross-platform projects. However, it might be useful, or even necessary, to use some platform-specific variables. For more information, see Platform Notes.

6.1.2 创建一个项目文件

qmake 使用储存在项目（.pro）文件中的信息来决定 Makefile 文件中该生成什么。

一个基本的项目文件包含关于应用程序的信息，比如，编译应用程序需要哪些文件，并且使用哪些配置设置。

这里是一个简单的示例项目文件：

```
SOURCES = hello.cpp
HEADERS = hello.h
CONFIG += qt warn_on release
```

我们将会提供一行一行的简要解释，具体细节将会在手册的后面的部分解释。

```
SOURCES = hello.cpp
```

这一行指定了实现应用程序的源程序文件。在这个例子中，恰好只有一个文件，hello.cpp。大部分应用程序需要多个文件，这种情况下可以把文件列在一行中，以空格分隔，就像这样：

```
SOURCES = hello.cpp main.cpp
```

另一种方式，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，就像这样：

```
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
```

这种方法中使用“+=”比“=”更安全，因为它只是向已有的列表中添加新的文件，而不是替换整个列表。

HEADERS 这一行中通常用来指定为这个应用程序创建的头文件，举例来说：

```
HEADERS += hello.h
```

列出源文件的任何一个方法对头文件也都适用。

CONFIG 这一行是用来告诉 qmake 关于应用程序的配置信息。

```
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。

CONFIG 一行中的 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。

CONFIG 一行中的 warn_on 部分告诉 qmake 要把编译器设置为输出警告信息的。

CONFIG 一行中的 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，稍后会讨论这里的。

项目文件就是纯文本（比如，可以使用像记事本、vim 和 xemacs 这些编辑器）并且必须存为“.pro”扩展名。应用程序的执行文件的名称必须和项目文件

的名称一样，但是扩展名是跟着平台而改变的。举例来说，一个叫做“hello.pro”的项目文件将会在 Windows 下生成“hello.exe”，而在 Unix 下生成“hello”。

6.1.3 生成 Makefile

当你已经创建好你的项目文件，生成 Makefile 就很容易了，你所要做的就是先到你所生成的项目文件那里然后输入：

Makefile 可以像这样由“.pro”文件生成：

```
qmake -o Makefile hello.pro
```

对于 Visual Studio 的用户，qmake 也可以生成“.dsp”文件，例如：

```
qmake -t vcapp -o hello.dsp hello.pro
```

6.2 开始

这篇教程会讲述 qmake 的基础，这本手册的其他主题包含了有关使用 qmake 的更多信息。

6.2.1 简单的开始

假设你已经完成了你的应用的基本实现，并且你已经创建了如下文件：

- hello.cpp
- hello.h
- main.cpp

首先，在源代码所在的目录用纯文本编辑器创建一个叫 hello.pro 的文本文件，你要做的第一件事是在该文件中添加几行来告诉 qmake 这些源文件和头文件是你的项目中的一部分。我们首先要添加源文件到项目中，为达成这个目的你需要使用”SOURCES“变量，将 hello.cpp 加入项目可以这样做。

```
SORUCES += hello.cpp
```

如果你更喜欢 make 风格的语法，即一次列出所有文件，你可以使用换行符，就像这样：

```
SOURCES += hello.cpp \
           main.cpp
```

既然源文件已经被列入项目文件中了，那么头文件也必须被添加。他们的添加

方式与源文件完全相同，除了变量名叫 “HEADERS”
当你做完这些事后，你的项目文件应当是这样的：

```
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

目标名称会自动设置，它与项目文件名相同，只是会添加适合与平台的后缀。例如，如果项目文件叫做 hello.pro，在 Windows 上目标将会叫做 hello.exe，而在 Unix 上叫 hello。如果你想使用一个不同的名字的话可以在项目文件中设置：

```
TARGET = helloworld
```

你可以使用 qmake 来为你的应用生产 makefile，在命令行界面，进入项目目录，输入下列命令：

```
qmake -o Makefile hello.pro
```

然后根据你使用的编译器输入’ make ‘或’ nmake ‘。

6.2.2 使应用支持调试

6.2.3 Starting Off Simple

Let’s assume that you have just finished a basic implementation of your application, and you have created the following files:

- hello.cpp
- hello.h
- main.cpp

You will find these files in the examples/qmake/tutorial directory of the Qt distribution. The only other thing you know about the setup of the application is that it’s written in Qt. First, using your favorite plain text editor, create a file called hello.pro in examples/qmake/tutorial. The first thing you need to do is add the lines that tell qmake about the source and header files that are part of your development project.

We’ll add the source files to the project file first. To do this you need to use the SOURCES variable. Just start a new line with SOURCES += and put hello.cpp after it. You should have something like this:

```
SOURCES += hello.cpp
```

We repeat this for each source file in the project, until we end up with the following:

```
SOURCES += hello.cpp
SOURCES += main.cpp
```

If you prefer to use a Make-like syntax, with all the files listed in one go you can use the newline escaping like this:

```
SOURCES = hello.cpp \
          main.cpp
```

Now that the source files are listed in the project file, the header files must be added. These are added in exactly the same way as source files, except that the variable name we use is HEADERS.
Once you have done this, your project file should look something like this:

```
SOURCES = hello.cpp \
          main.cpp
HEADERS += hello.h
```

The target name is set automatically. It is the same as the project filename, but with the suffix appropriate for the platform. For example, if the project file is called hello.pro, the target will be hello.exe on Windows and hello on Unix. If you want to use a different name you can set it in the project file:

```
TARGET = helloworld
```

The finished project file should look like this:

```
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

You can now use qmake to generate a Makefile for your application. On the command line, in your project directory, type the following:

```
qmake -o Makefile hello.pro
```

Then type make or nmake depending on the compiler you use.

For Visual Studio users, qmake can also generate Visual Studio project files. For example:

```
qmake -tp vc hello.pro
```

6.3 Making an Application Debuggable

The release version of an application does not contain any debugging symbols or other debugging information. During development, it is useful to produce a debugging version of the application that has the relevant information. This is easily achieved by adding debug to the CONFIG variable in the project file.

For example:

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
```

Use qmake as before to generate a Makefile. You will now obtain useful information about your application when running it in a debugging environment.

6.4 Adding Platform-Specific Source Files

After a few hours of coding, you might have made a start on the platform-specific part of your application, and decided to keep the platform-dependent code separate. So you now have two new files to include into your project file: hellowin.cpp and hellounix.cpp. We cannot just add these to the SOURCES variable since that would place both files in the Makefile. So, what we need to do here is to use a scope which will be processed depending on which platform we are building for.

A simple scope that adds the platform-dependent file for Windows looks like this:

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

When building for Windows, qmake adds hellowin.cpp to the list of source files. When building for any other platform, qmake simply ignores it. Now all that is left to be done is to create a scope for the Unix-specific file.

When you have done that, your project file should look something like this:

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
  SOURCES += hellowin.cpp
}
unix {
  SOURCES += hellounix.cpp
}
```

Use qmake as before to generate a Makefile.

The ! symbol is used to negate the test. That is, exists(main.cpp) is true if the file exists, and !exists(main.cpp) is true if the file does not exist.

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
  SOURCES += hellowin.cpp
```

```
455         }
456         unix {
457           SOURCES += hellounix.cpp
458         }
459         !exists( main.cpp ) {
460           error( "No main.cpp file found" )
461         }
```

462 Use qmake as before to generate a makefile. If you rename main.cpp temporar-
463 ily, you will see the message and qmake will stop processing.

464 6.5 Checking for More than One Condition

465 Suppose you use Windows and you want to be able to see statement out-
466 put with qDebug() when you run your application on the command line. To
467 see the output, you must build your application with the appropriate console
468 setting. We can easily put console on the CONFIG line to include this setting
469 in the Makefile on Windows. However, let's say that we only want to add the
470 CONFIG line when we are running on Windows and when debug is already
471 on the CONFIG line. This requires using two nested scopes. First create one
472 scope, then create the other inside it. Put the settings to be processed inside
473 the second scope, like this:

474

```
475         win32 {
476           debug {
477             CONFIG += console
478           }
479         }
```

480 Nested scopes can be joined together using colons, so the final project file looks
481 like this:

482

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
  SOURCES += hellowin.cpp
}
unix {
  SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
  error( "No main.cpp file found" )
}
win32:debug {
  CONFIG += console
}
```

That's it! You have now completed the tutorial for qmake, and are ready to write project files for your development projects.

6.6 创建项目文件

项目文件通常包含创建应用程序，库和插件所需要的一切信息。Generally, you use a series of declarations to specify the resources in the project, but support for the simple programming constructs enables you to describe different build processes for different platforms and evvironments.

6.6.1 项目文件的 elements

The project file format used by qmake can be to support both simple and fairly complex build systems(简单复杂的通吃)。对于简单的项目文件而言，使用直接的变量声明的方式，定义标准的变量来表明哪些头文件和源文件会被用到。复杂的项目文件或许用到控制流结构来 fine-tune the build process.

下面的 sections 描述项目文件中被用到的不同元素类型。

6.6.1.1 变量

在项目文件中，变量被用来保存字符串的一个列表。在最简单的项目中，这些变量告诉 qmake 哪些配置选项会被使用，或者在创建过程中要被用到的文件名或者路径。

qmake 在每一个项目文件中寻找确定的变量，并用它们的内容来决定在哪些东西应该被用来写到 Makefile 文件中。例如，HEADERS 和 SOURCES变量被用来告诉 qmake 在相同的目录下哪些头文件和源文件会被当前的项目文件用到。

下面的 snippets 展示了如何给变量进行赋值:

```
HEADERS = mainwindow.h paintwidget.h
```

多行变量内容采用下面的方式进行书写:(反斜线连接)

```
SOURCES = main.cpp mainwindow.cpp \
          paintwidget.cpp
CONFIG += console
```

下表是经常被用到的变量以及对它们变量内容的讨论。

变量	内容
CONFIG	一般项目配置选项
DESTID	生成的可执行文件的存放目录
FORMS	UI 文件的列表
HEADERS	头文件列表
QT	模块内容列表
RESOURCES	资源文件列表
SOURCES	源文件列表
TEMPLATE	项目文件的类型 (可能是可执行程序，库或者插件)

其中 $$ 两个美元符号被用来作为内建函数，变量的引用的展开。

6.6.1.2 空格

通常空格是被忽略的，如果想显式的使用空格，必须用双引号进行括起来。如:

```
DEST = "Program Files"
```

```
win32:INCLUDEPATH += "C:/mylibs/extra headers"
unix:INCLUDEPATH += "/home/user/extra headers"
```

6.6.1.3 注释

注释使用 shape(#) 符号进行注释

6.6.1.4 内建函数和控制流

qmake 提供了一组内建的函数来处理变量的内容，如最常用的 include 函数，其中 include 函数被用来包含其它的项目文件:

```
include(other.pro)
```

对于变量更加复杂的操作需要可能要用到循环，如 find(), unique() 和 count().

6.6.2 项目 template

变量TEMPLATE被用来指定该项目的生成类型，缺省的项目类型是 app。下图是项目类型列表:

6.6.3 一般配置

变量CONFIG指定项目的 options 和特征。

项目可以被指定为 release 版本，也可以指定为 debug 版本，如果两个都指定将以最后一个为创建的依据，如果两个版本的方式都想指定，可以将值改为debug_and_release

Template	qmake Output
app(default)	生成 application
lib	创建库
aux	Makefile to build nothing
subdirs	subdirs 指定包含的子目录，每个子目录必须有自己的项目文件
vcapp	创建 Visual Studio 的应用程序
vclib	创建 Visual Studio 的库
vcsubdirs	Visual Studio Solution file to build project in sub-directories

Note: Each of the options specialized in the **CONFIG** variable can also be used as a scope condition.you can test for the presence of certain configuration options by using the built-in CONFIG() function.For example, the following lines show the function as the condition in a scope to test whether only the opengl option is in use:

```
CONFIG(opengl) {
    message(Building with OpenGL support.)
} else {
    message(OpenGL support is not available.)
}
```

下面的 options 定义项目创建的类型：

选项	描述
qt	该项目是一个 Qt 应用程序，且应该连接 Qt 库。
x11	该项目是一个 x11 应用程序。

如果你的应用程序使用 Qt 库也想是 debug 版本，你可以采用以下的配置：

6.6.4 声明 Qt 库

我们可以使用变量QT来确定要使用哪些模块，如下面的例子:
其中 QT 缺省值是包含 core 和 gui 这两个模块的。
使用下面的方式，将会忽略掉 core 和 gui 两个模块。
但是我们也可以显式的去掉缺省的模块，如下面的方式：

```
QT += network xml
```

```
QT = network xml # This will omit the core and gui modules.
```

6.6.5 声明其它的库

我们可以使用变量LIBS来添加其它的库，如：

其它头文件的指定方式我们可以指定其搜索的目录，如：

创建项目配置文件完。


```
QT -= gui # Only the core module is used.
```

```
LIBS += -L/usr/local/lib -lmath
```



```
INCLUDEPATH = c:/msdev/include d:/stl/include
```

第7章　QWidget

QWidget 的受保护的事件类成员

virtual void actionEvent(QActionEvent *event):

virtual void changeEvent(QEvent *event):

virtual void closeEvent(QCloseEvent *event):

virtual void contextMenuEvent(QContexMenuEvent *event):

virtual void dragEnterEvent(QDragEnterEvent *event):

virtual void dragLeaveEvent(QDragLeaveEvent *event):

virtual void dragMoveEvent(QDragMoveEvent *event):

virtual void dropEvent(QDropEvent *event):

virtual void enterEvent(QEvent *event):

virtual void focusInEvent(QFocusEvent *event):

virtual void focusOutEvent(QFocusEvent *event):

virtual void hideEvent(QHideEvent *event):

virtual void inputMethodEvent(QInputMethodEvent *event):

virtual void keyPressEvent(QKeyEvent *event):

virtual void keyReleaseEvent(QKeyEvent *event):

virtual void leaveEvent(QEvent *event):

virtual void mouseDoubleClickEvent(QMouseEvent *event):

virtual void mouseMoveEvent(QMouseEvent *event):

virtual void mousePressEvent(QMouseEvent *event):

virtual void mouseReleaseEvent(QMOuseEvent *event):

virtual void moveEvent(QMoveEvent *event):

virtual void nativeEvent(const QByteArray &eventType, void *message, long *result):

virtual void paintEvent(QPaintEvent *event):

virtual void resizeEvent(QResizeEvent *event):

virtual void showEvent(QShowEvent *event):

virtual void tabletEvent(QTabletEvent *event):

591 virtual void wheelEvent(QWheelEvent *event):