



Qt程序设计方法

QT基础

**School of Computer Science
and Engineering**

2024-2025-暑期课程

Qt 简介

■Qt是一个跨平台应用程序和UI开发框架。使用Qt只需一次性开发应用程序，无须重新编写源代码，便可跨不同桌面和嵌入式操作系统部署这些应用程序。Qt Software的前身为创始于1994年的Trolltech（奇趣科技），Trolltech于2008年6月被 Nokia 收购，加速了其跨平台开发战略，2011年3月Qt被芬兰的Digia公司收购。

■Qt Creator是全新的跨平台 Qt IDE（集成开发环境），可单独使用，也可与 Qt 库和开发工具组成一套完整的SDK（软件开发工具包）。包括：高级 C++ 代码编辑器、项目和生成管理工具、集成的上下文相关的帮助系统、图形化调试器、代码管理和浏览工具。

本章内容

- Qt Creator的下载和安装
- Qt Creator环境介绍
- Qt工具简介
- 小结



1.1 Qt Creator的下载和安装

为了避免由于开发环境的版本差异而产生不必要的问题，推荐在学习本书前下载和本书相同的软件版本。本书使用Qt 5.6.1版本，其中包含了Qt Creator 4.0.1。

➤ **地址：**

http://download.qt.io/official_releases/qt/5.6/5.6.1-1/

➤ **下载文件：** qt-opensource-windows-x86-mingw492-5.6.1-1.exe



版本介绍

Qt安装包：

qt-opensource-windows-x86-mingw492-5.6.1-1.exe

- **opensource**表示开源版本
- **windows-x86**表示Windows 32位平台
- **mingw**表示使用MinGW编译器
- **5.6.1-1**是当前版本号





提示

MinGW即Minimalist GNU For Windows，是将GNU开发工具移植到Win32平台下的产物，是一套Windows上的GNU工具集。用其开发的程序不需要额外的第三方DLL支持就可以直接在Windows下运行。更多内容请查看<http://www.mingw.org>。

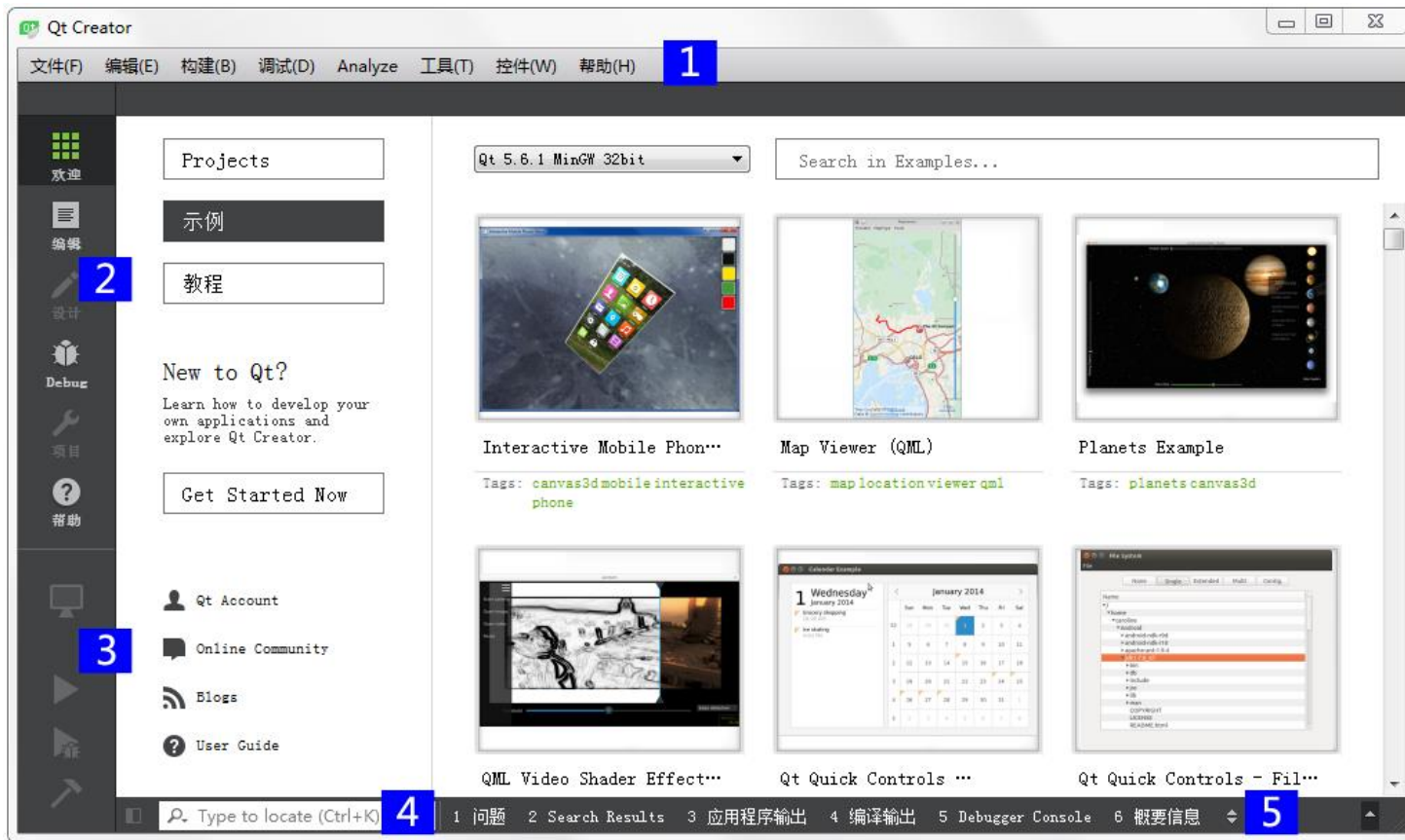


注意：

- 安装路径中不能有中文。
- 安装开始是否登陆或者注册Qt账号，不影响程序的安装，可以直接跳过。
- 在选择组件界面，可以选择安装一些模块，单击一个组件，可以在右侧显示该组件的详细介绍，初学者建议保持默认选择。



1.2 Qt Creator环境介绍



Qt Creator主要由主窗口区、菜单栏、模式选择器、构建套件选择器、定位器和输出窗格等部分组成



①**菜单栏 (Menu Bar)**。这里有8个菜单选项，包含了常用的功能菜单。

- **文件菜单**。其中包含了新建、打开和关闭项目和文件、打印文件和退出等基本功能菜单。

- **编辑菜单**。这里有撤销、剪切、复制、查找和选择编码等常用功能菜单，在高级菜单中还有标示空白符、折叠代码、改变字体大小和使用vim风格编辑等功能菜单。

- **构建菜单**。包含构建和运行项目等相关的菜单。

- **调试菜单**。包含调试程序等相关的功能菜单。

- **Analyze分析菜单**。包含QML分析器、Valgrind内存和功能分析器等相关菜单。

- **工具菜单**。这里提供了快速定位菜单、版本控制工具菜单和外部工具菜单等。这里的选项菜单中包含了Qt Creator各个方面的设置选项：环境设置、文本编辑器设置、帮助设置、构建和运行设置、调试器设置和版本控制设置等。

- **控制菜单**。这里包含了设置窗口布局的一些菜单，如全屏显示和隐藏边栏等。

- **帮助菜单**。包含Qt帮助、Qt Creator版本信息、报告bug和插件管理等菜单。



②**模式选择器 (Mode Selector)**。Qt Creator包含欢迎、编辑、设计、调试、项目和帮助6个模式，各个模式完成不同的功能，也可以使用快捷键来更换模式，它们对应的快捷键依次是Ctrl + 数字1~6。

- **欢迎模式**。这里主要提供了一些功能的快捷入口，如打开帮助教程、打开示例程序、打开项目、新建项目、快速打开以前的项目和会话、联网查看Qt官方论坛和博客等。
- **编辑模式**。这里主要用来查看和编辑程序代码，管理项目文件。也可以在“工具→选项”菜单项中对编辑器进行设置。
- **设计模式**。这里整合了Qt 设计师的功能。可以在这里设计图形界面，进行部件属性设置、信号和槽设置、布局设置等操作。可以在“工具→选项”菜单项中对设计师进行设置。
- **调试模式**。支持设置断点、单步调试和远程调试等功能，包含局部变量和监视器、断点、线程以及快照等查看窗口。可以在“工具→选项”菜单项中设置调试器的相关选项。
- **项目模式**。包含对特定项目的构建设置、运行设置、编辑器设置、代码风格设置和依赖关系等页面。也可以在“工具→选项”菜单项中对项目进行设置。
- **帮助模式**。在帮助模式中将Qt助手整合了进来，包含目录、索引、查找和书签等几个导航模式。可以在“工具→选项”菜单中对帮助进行相关设置。



③ **构建套件选择器 (Kit Selector)** 。包含了目标选择器 (Target selector)、运行按钮 (Run)、调试按钮 (Debug) 和构建按钮 (Building) 4 个图标。目标选择器用来选择要构建哪个项目，使用哪个Qt库，这对于多个Qt库的项目很有用。这里还可以选择编译项目的debug版本或是release版本。运行按钮可以实现项目的构建和运行；调试按钮可以进入调试模式，开始调试程序；构建按钮完成项目的构建。

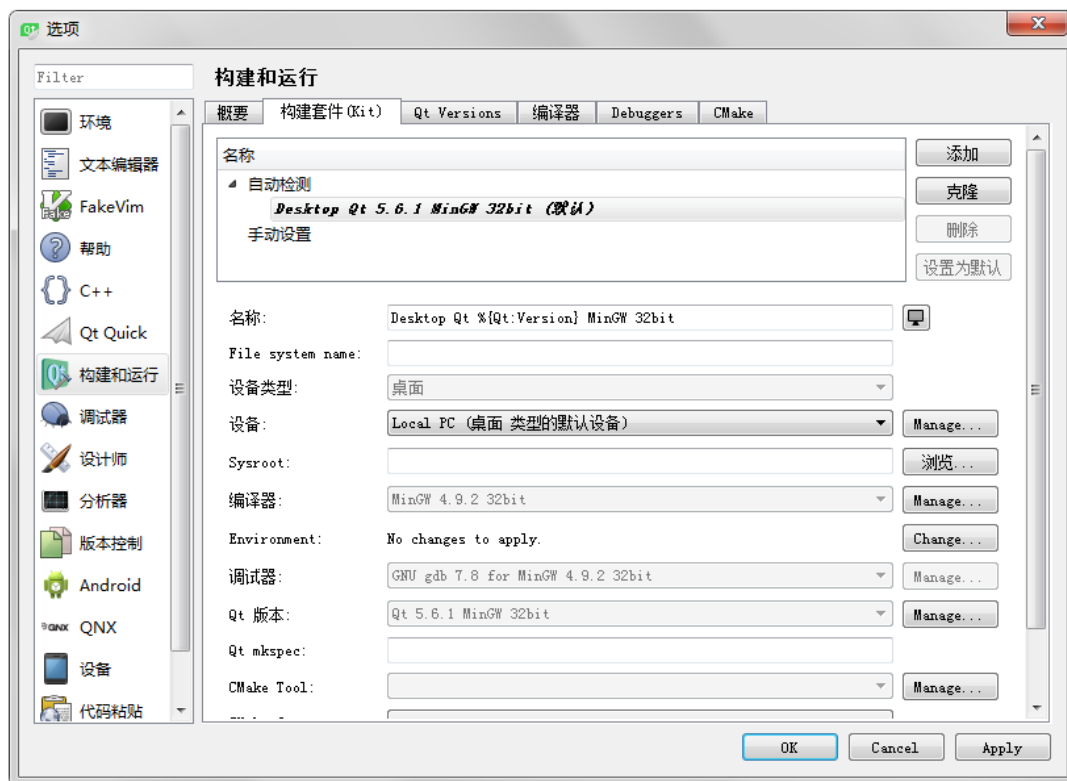
④ **定位器 (Locator)** 。在Qt Creator中可以使用定位器来快速定位项目、文件、类、方法、帮助文档以及文件系统。可以使用过滤器来更加准确地定位要查找的结果。可以在“工具→选项”菜单项中设置定位器的相关选项。

⑤ **输出窗格 (Output panes)** 。这里包含了问题、搜索结果、应用程序输出、编译输出、Debugger Console、概要信息、版本控制7个选项，它们分别对应一个输出窗口，相应的快捷键依次是Alt + 数字1~7。问题窗口显示程序编译时的错误和警告信息；搜索结果窗口显示执行了搜索操作后的结果信息；应用程序输出窗口显示在应用程序运行过程中输出的所有信息；编译输出窗口显示程序编译过程输出的相关信息；版本控制窗口显示版本控制的相关输出信息。



将Qt Creator与Qt库进行关联

选择“工具→选项”菜单项，然后选择“构建和运行”项。可以看到构建套件中已经自动检测到了Qt版本、编译器和调试器。如果以后需要添加其他版本的Qt，可以在这里先添加编译器，然后添加Qt版本，最后添加构建套件。



运行一个示例程序

设定Qt库以后，在欢迎界面已经可以看到所有示例程序了，它们几乎涉及到了Qt支持的所有功能。这里还提供了一个搜索栏，可以进行示例程序的查找，比如查找所有和对话框相关的例子，可以输入“dialog”关键字。

Projects

示例

教程

New to Qt?
Learn how to develop your own applications and explore Qt Creator.

Get Started Now

Qt Account

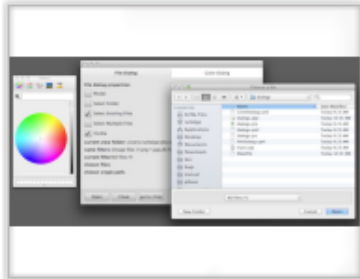
Online Community

Blogs

User Guide

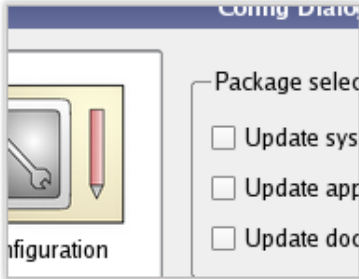
Qt 5.6.1 MinGW 32bit

dialog



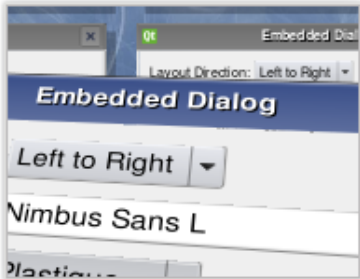
Qt Quick System Dialogs

Tags: `dialogs` `system` `quick` `dialog`



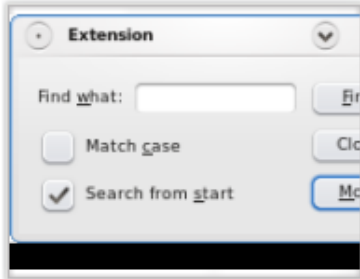
Config Dialog Example

Tags: `widgets` `dialog` `config`



Embedded Dialogs

Tags: `widgets` `embedded` `dialogs` `ios`

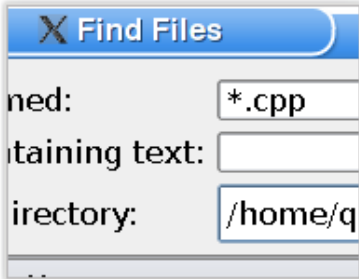


Find Files

Find what:

☐ Match case

☒ Search from start

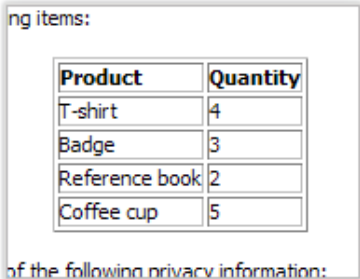


Find Files

Find what:

Containing text:

Directory:



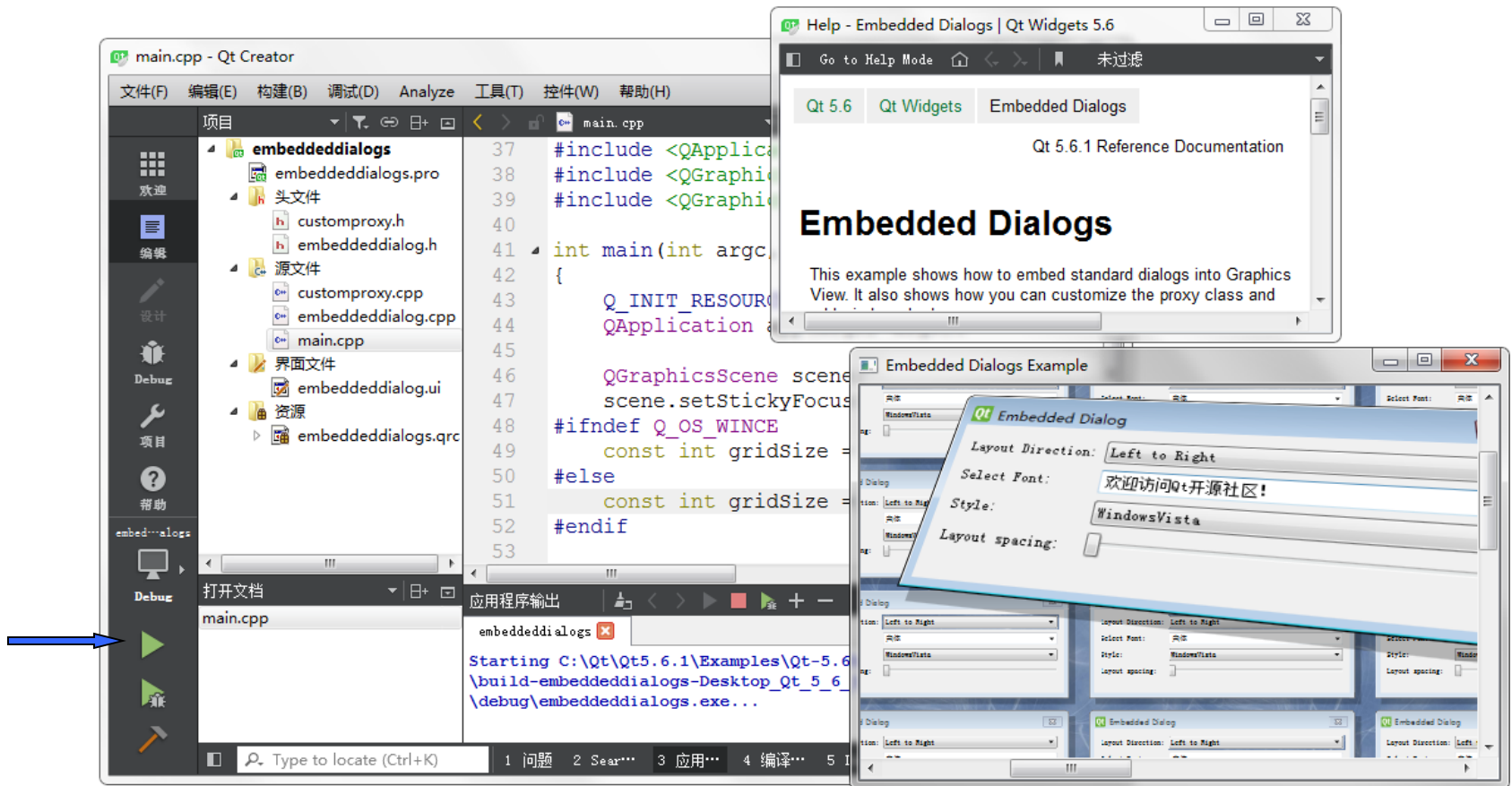
Find items:

Product	Quantity
T-shirt	4
Badge	3
Reference book	2
Coffee cup	5

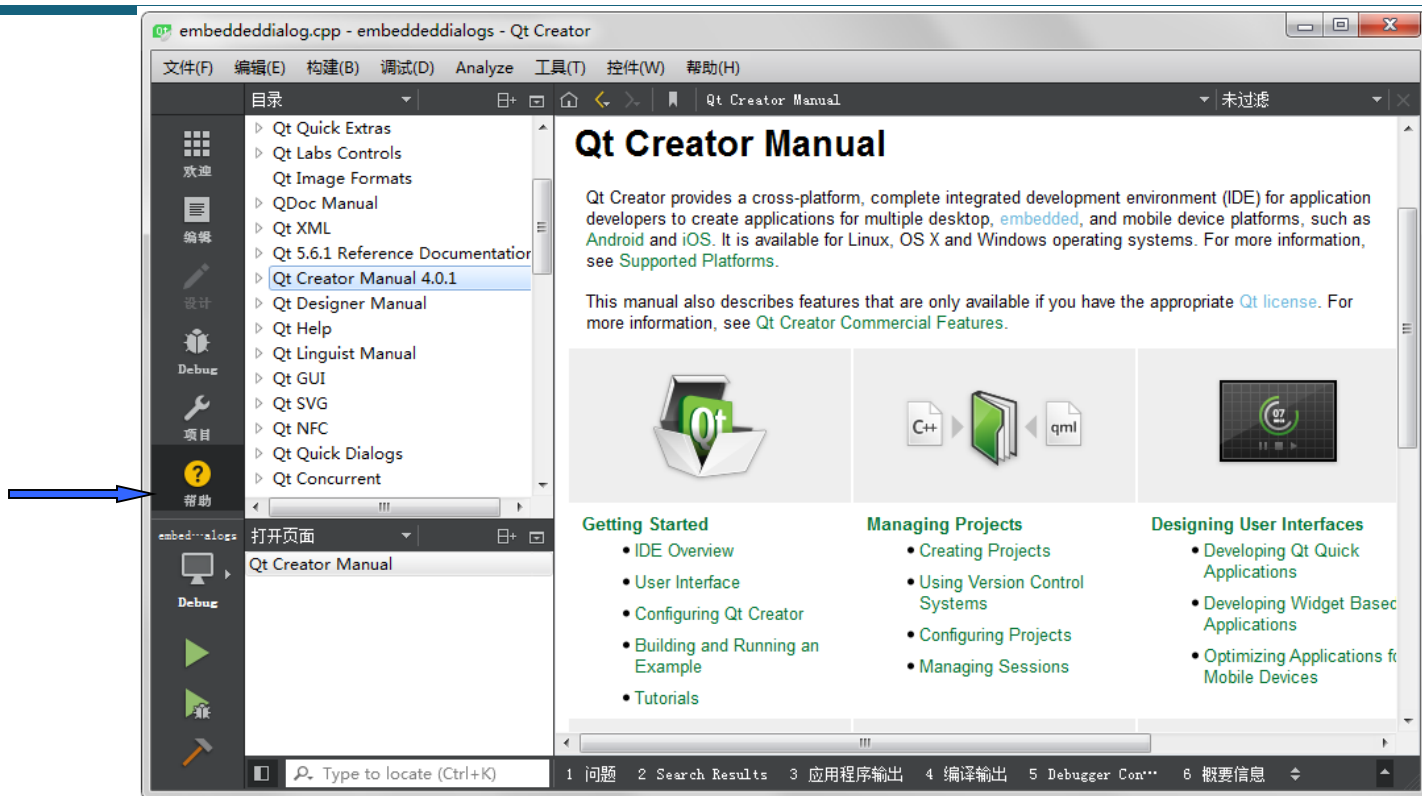
of the following privacy information:



选择Embedded Dialogs示例程序，这时便进入了编辑模式。单击左下角的运行按钮，程序便开始编译运行。



学习使用帮助模式

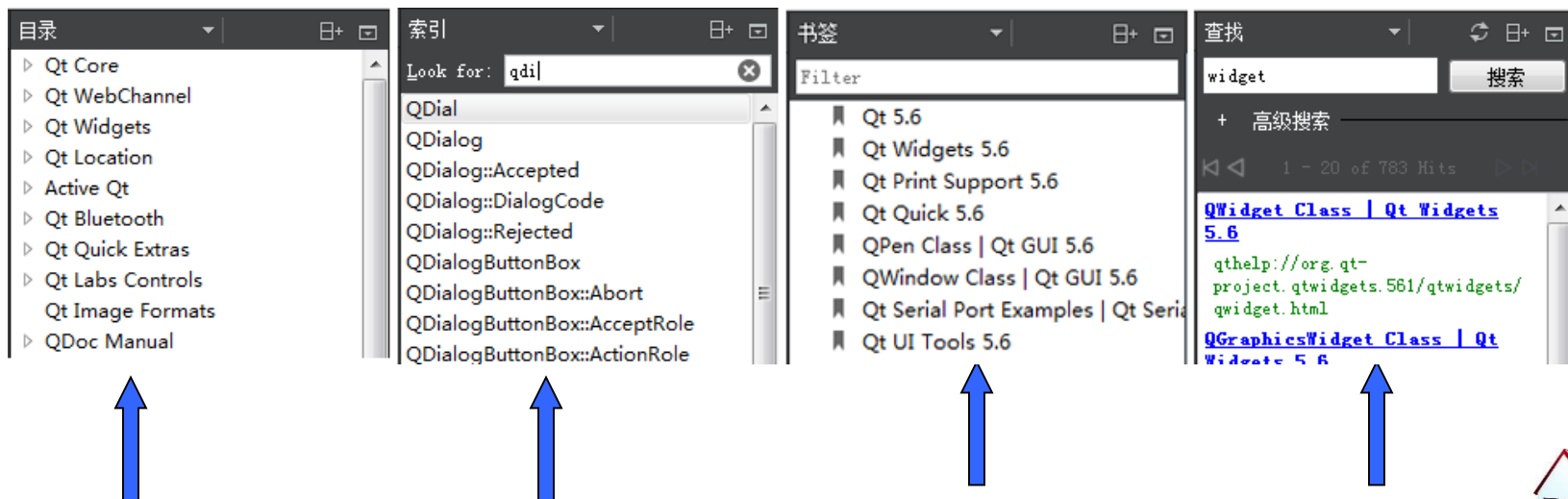


初学一个软件，无法马上掌握其全部功能，而且可能对某些功能很不理解，这时软件的帮助文档就很有用了，学习Qt也是如此。虽然Qt的帮助文档目前还是全英文的，但是我们必须掌握它，毕竟这才是原生的东西，而网上的一些中文版本是广大爱好者翻译的，效果差强人意，再说，如果要深入学习，以后接触到的也以英文文档居多。按下Ctrl+6组合键（当然也可以直接单击“帮助”图标）进入帮助模式。



在查看帮助时可能想为某一页面添加书签，以便以后再看，则可以按下快捷键Ctrl + M，或者单击界面上方边栏里的图标。打开帮助模式时默认是目录视图，其实帮助的工具窗口中还提供了“索引”、“查找”和“书签”3种方式对文档进行导航。

- 在**书签**方式下，可以看到刚才添加的书签；
- 在**查找**方式下，可以输入关键字进行全文检索，就是在整个文档的所有文章中进行查找；
- 在**索引**方式下，只要输入关键字，那么就可以罗列出相关的内容。



Qt工具简介

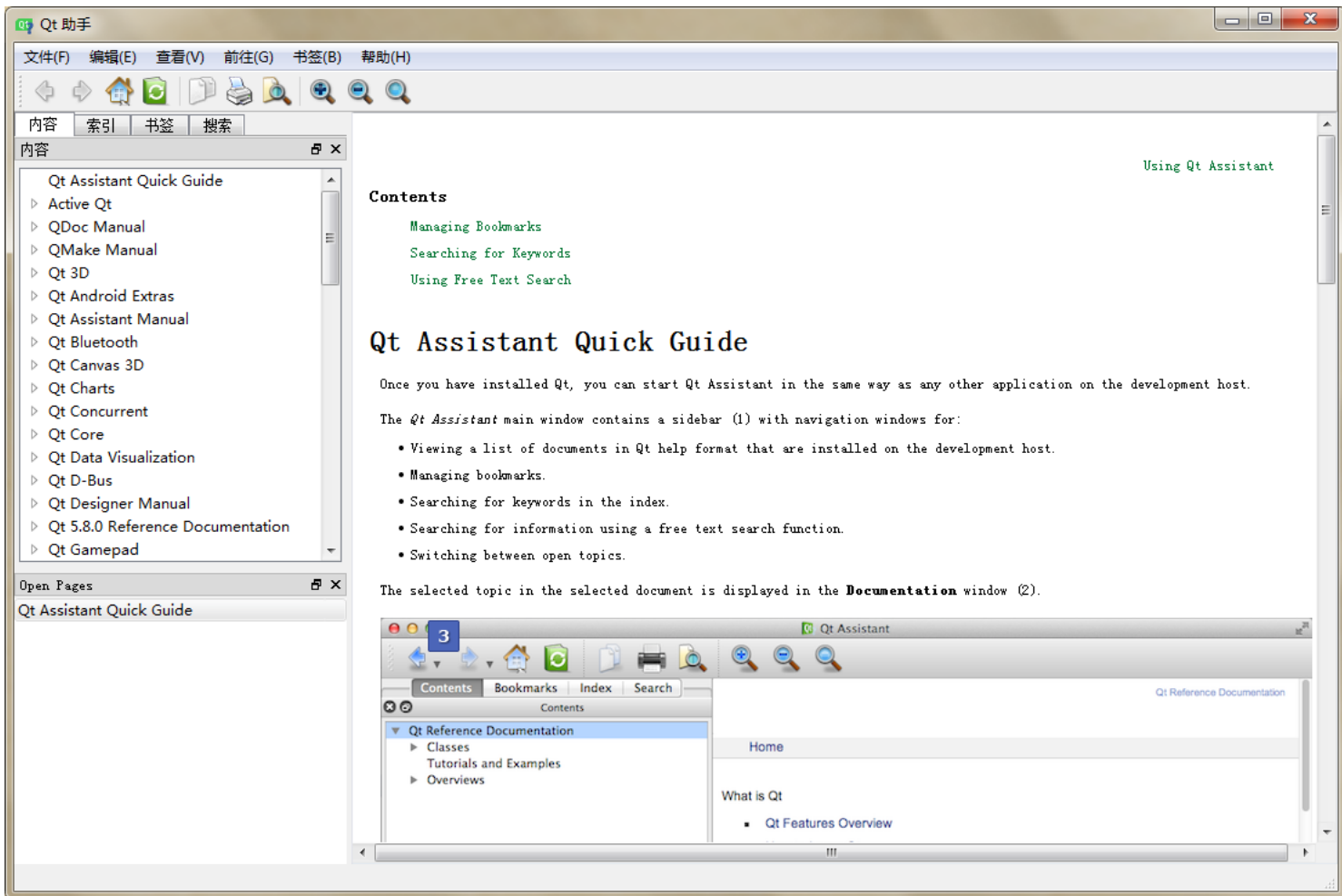
前面安装的Qt 5.6.1中包含了几个很有用的工具，分别是：

- Qt Assistant (Qt助手)
- Qt Linguist (Qt语言家)
- Qt Designer (Qt设计师)

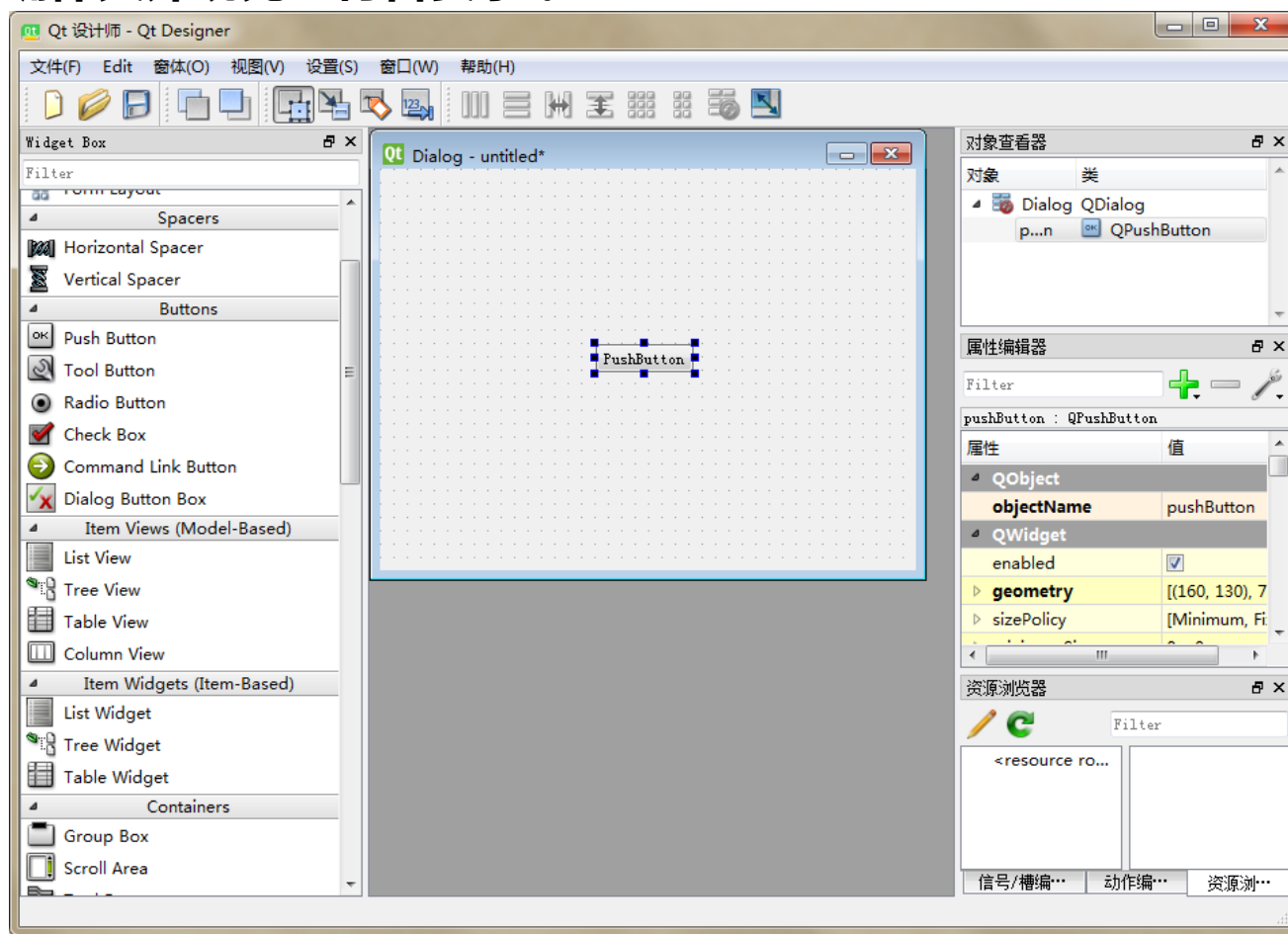
可以从开始菜单启动它们；也可以在安装目录下找到它们，例如C:\Qt\Qt5.6.1\5.6\mingw49_32\bin。



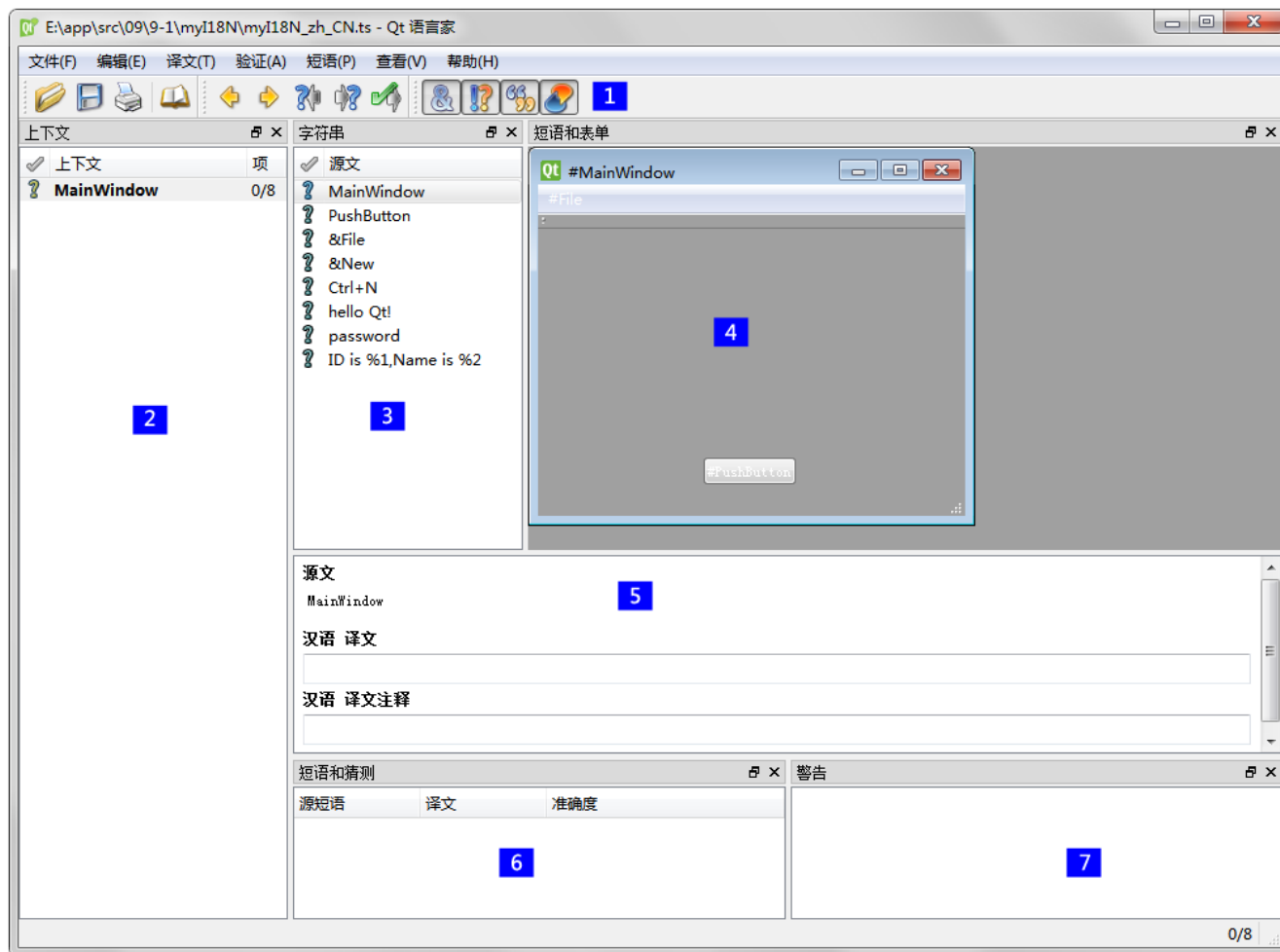
Qt Assistant是可配置且可重新发布的文档阅读器，可以方便地进行定制并与Qt应用程序一起重新发布。Qt Assistant已经被整合进Qt Creator，就是前面介绍的Qt帮助。



Qt Designer是强大的跨平台GUI布局和格式构建器。由于使用了与应用程序中将要使用的相同的部件，可以使用屏幕上的格式快速设计、创建部件以及对话框。使用Qt Designer创建的界面样式功能齐全并可以进行预览，这样就可确保其外观完全符合要求。



Qt Linguist 提供了一套加速应用程序翻译和国际化的工具。Qt 使用单一的源码树和单一的应用程序二进制包就可同时支持多个语言和书写系统。



➤ **Qt Creator的下载、安装以及Qt示例程序的运行，最重要的是要掌握Qt帮助的使用，因为在后面的章节里几乎每个知识点都要使用Qt的帮助索引来查找关键字。不仅是要掌握一个知识，更多的是要学会一种学习方法。**



- Qt的历史？
- Qt Creator的作用？
- Qt帮助有哪几种视图方式？
- Qt的常用工具有哪些？





Hello World

主要内容

- 2.1 编写HelloWorld程序
- 2.2 程序的运行与发布
- 2.3 helloworld程序源码与编译过程详解
- 2.4 项目模式和项目文件介绍
- 2.5 小结



2.1 编写HelloWorld程序

什么是Hello World程序？

Hello World程序就是让应用程序显示“Hello World”字符串。这是最简单的应用，但却包含了一个应用程序的基本要素，所以一般使用它来演示程序的创建过程。在本节中要讲的就是在Qt Creator中创建一个图形用户界面的项目，来生成一个可以显示“Hello World”字符串的程序。



新建Qt Widgets应用

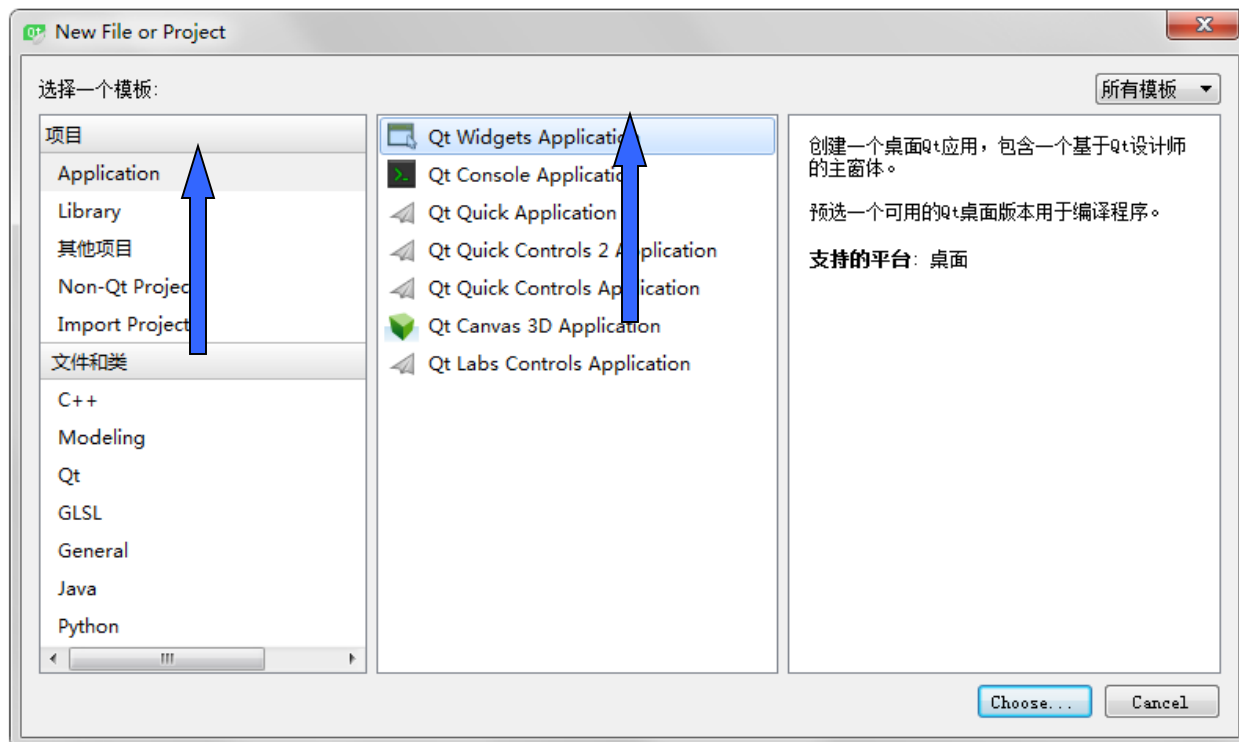
分五个步骤：

- **第一步，选择项目模板。**
- **第二步，输入项目信息。**
- **第三步，选择构建套件。**
- **第四步，输入类信息。**
- **第五步，设置项目管理。**



第一步，选择项目模板。

打开“文件→新建文件或项目”菜单项（也可以直接按下Ctrl+N快捷键，或者单击欢迎模式中的New Project按钮），在选择模板页面选择Application中的Qt Widgets Application一项，然后单击Choose按钮。



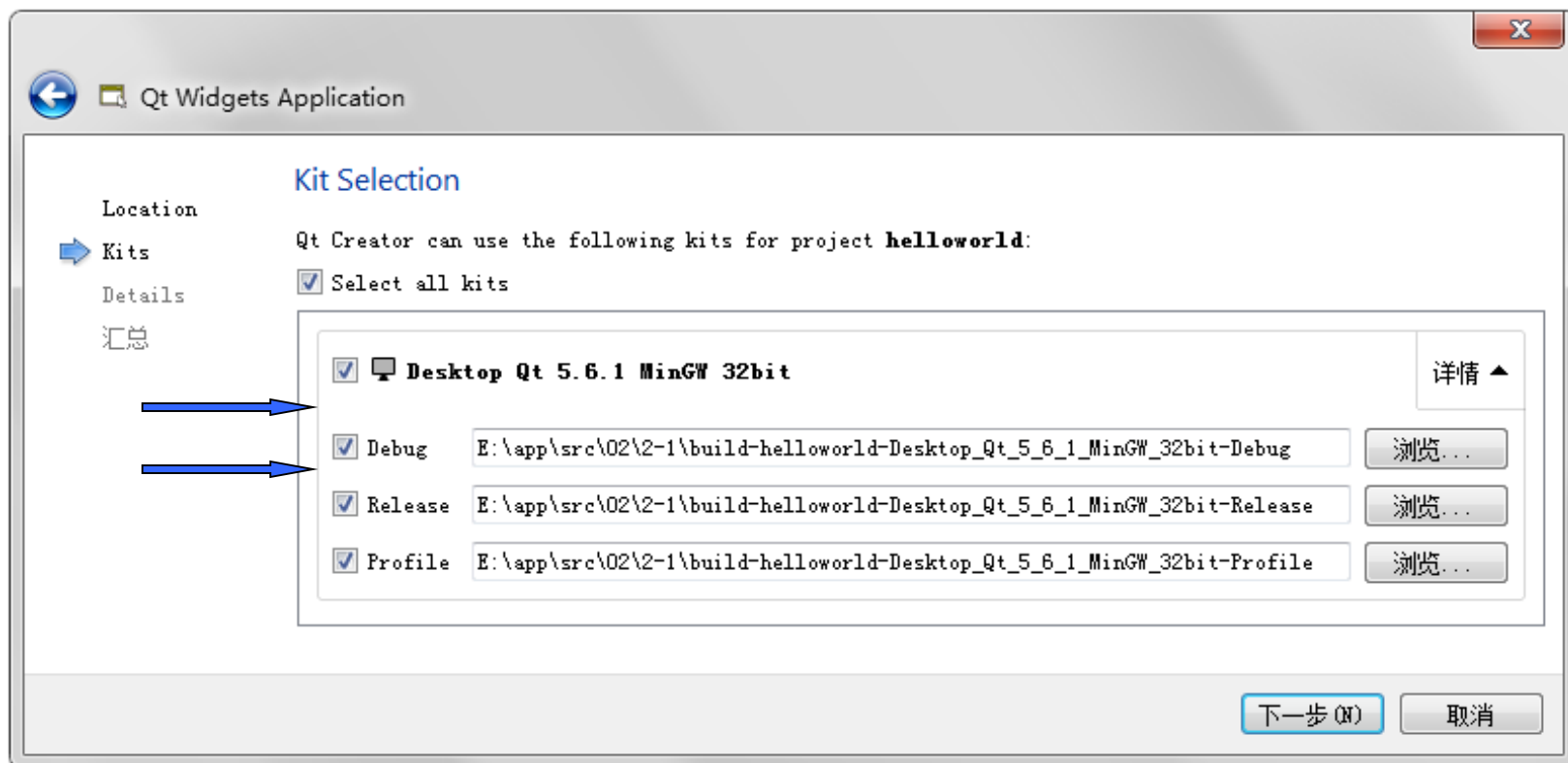
第二步，输入项目信息。

在“项目介绍和位置”页面输入项目的名称为helloworld，然后单击创建路径右边的“浏览”按钮选择源码路径，例如这里是“E:\app\src\02\2-1”。如果选中了这里的“设置默认的项目路径”，那么以后创建的项目会默认使用该目录。（注意：项目名和路径中都不能出现中文。）



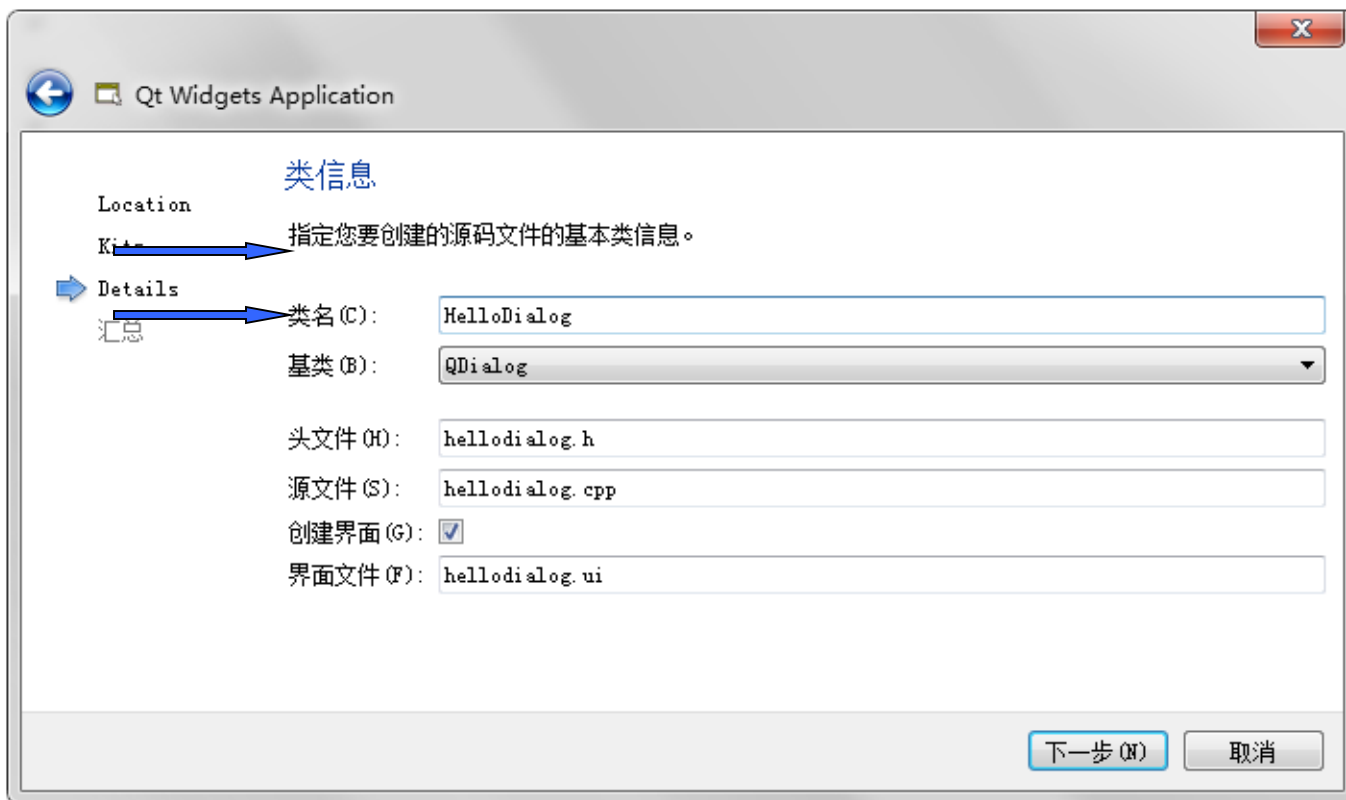
第三步，选择构建套件。

这里显示的Desktop Qt 5.6.1就是在第1章看到的构建套件，下面默认为Debug版本（调试版本）和Release版本（发布版本）分别设置了两个不同的目录。



第四步，输入类信息。

在“类信息”页面中创建一个自定义类。这里设定基类为QDialog，类名为HelloDialog，表明该类继承自QDialog类，使用这个类可以生成一个对话框界面。这时下面的头文件、源文件和界面文件都会自动生成，保持默认即可。



The image shows a screenshot of the "Qt Widgets Application" dialog box, specifically the "Class Information" (类信息) tab. The dialog has a sidebar on the left with three items: "Location", "Details", and "Summary" (汇总). The "Details" item is selected and highlighted with a blue arrow. The main area of the dialog contains the following fields:

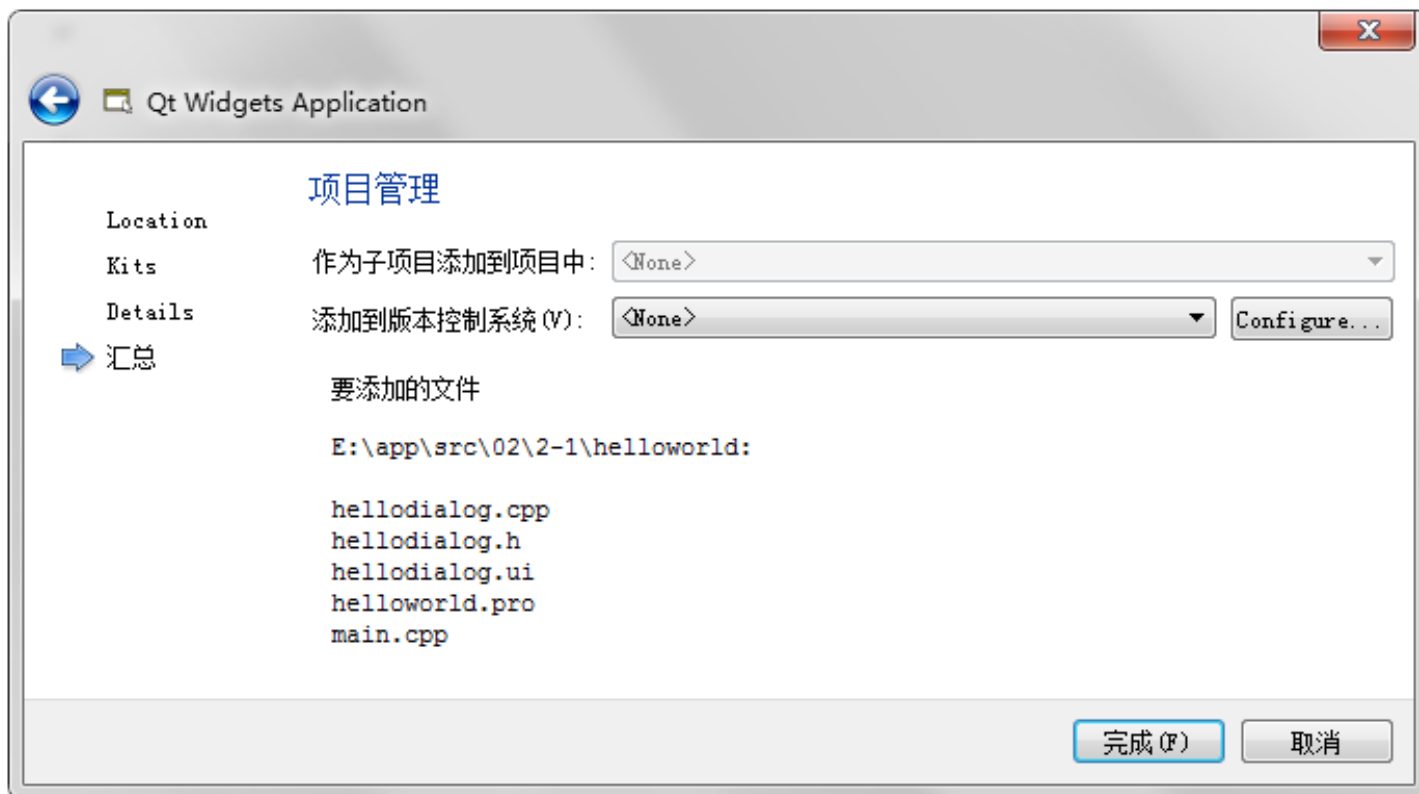
- Location:** A label with a blue arrow pointing to the "Details" section in the sidebar.
- Kind:** A label with a blue arrow pointing to the "Details" section in the sidebar.
- Details:** A label with a blue arrow pointing to the "Details" section in the sidebar.
- Summary:** A label with a blue arrow pointing to the "Details" section in the sidebar.
- 类名 (C):** A text box containing "HelloDialog".
- 基类 (B):** A dropdown menu showing "QDialog".
- 头文件 (H):** A text box containing "hellodialog.h".
- 源文件 (S):** A text box containing "hellodialog.cpp".
- 创建界面 (G):** A checkbox that is checked.
- 界面文件 (F):** A text box containing "hellodialog.ui".

At the bottom right of the dialog, there are two buttons: "下一步 (N)" (Next) and "取消" (Cancel).



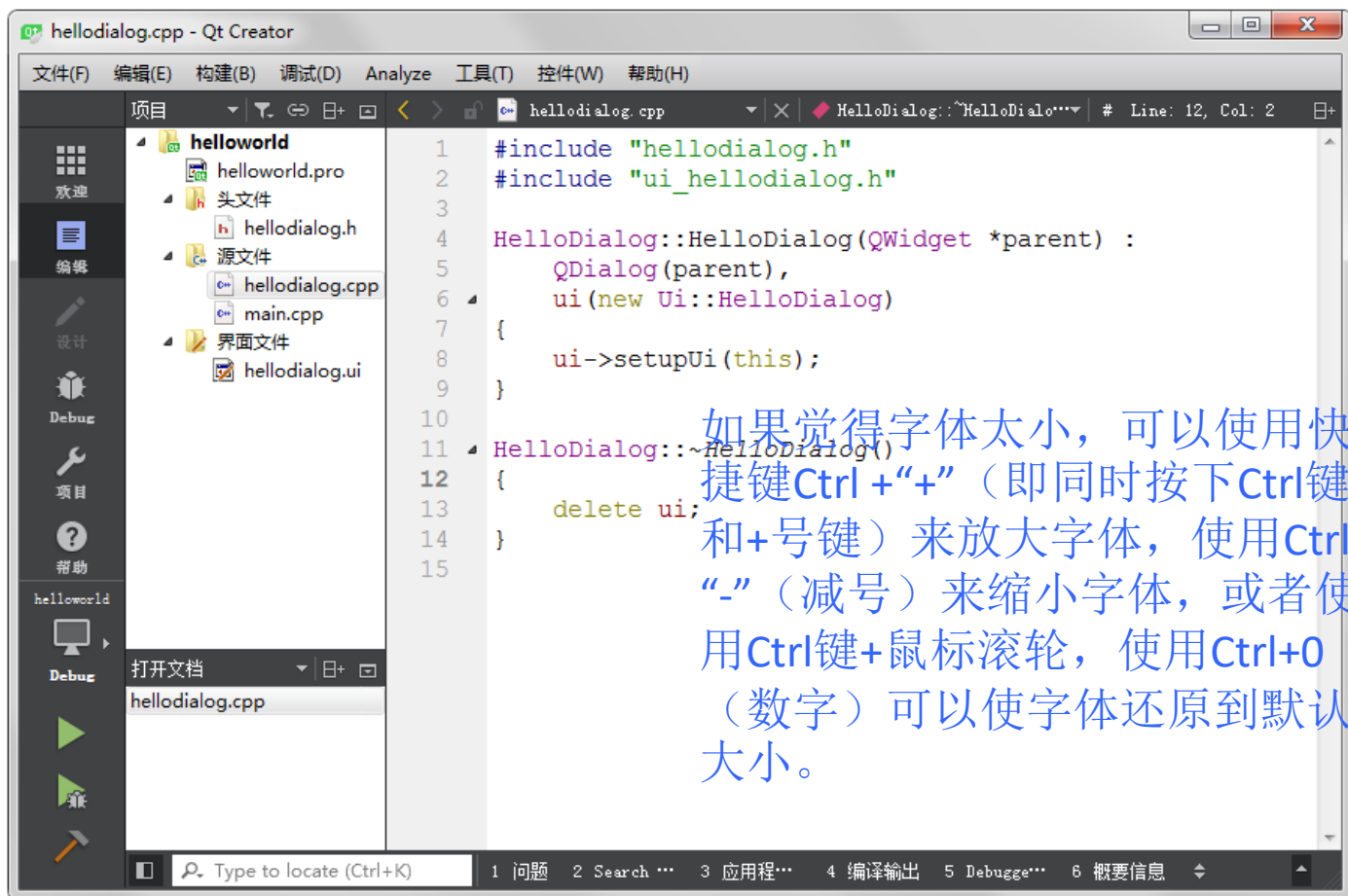
第五步，设置项目管理。

在这里可以看到这个项目的汇总信息，还可以使用版本控制系统，这个项目不会涉及，所以可以直接单击“下一步”



编辑模式

项目建立完成后会直接进入编辑模式。界面的右边是编辑器，可以阅读和编辑代码。在左边侧边栏，其中罗列了项目中的所有文件。



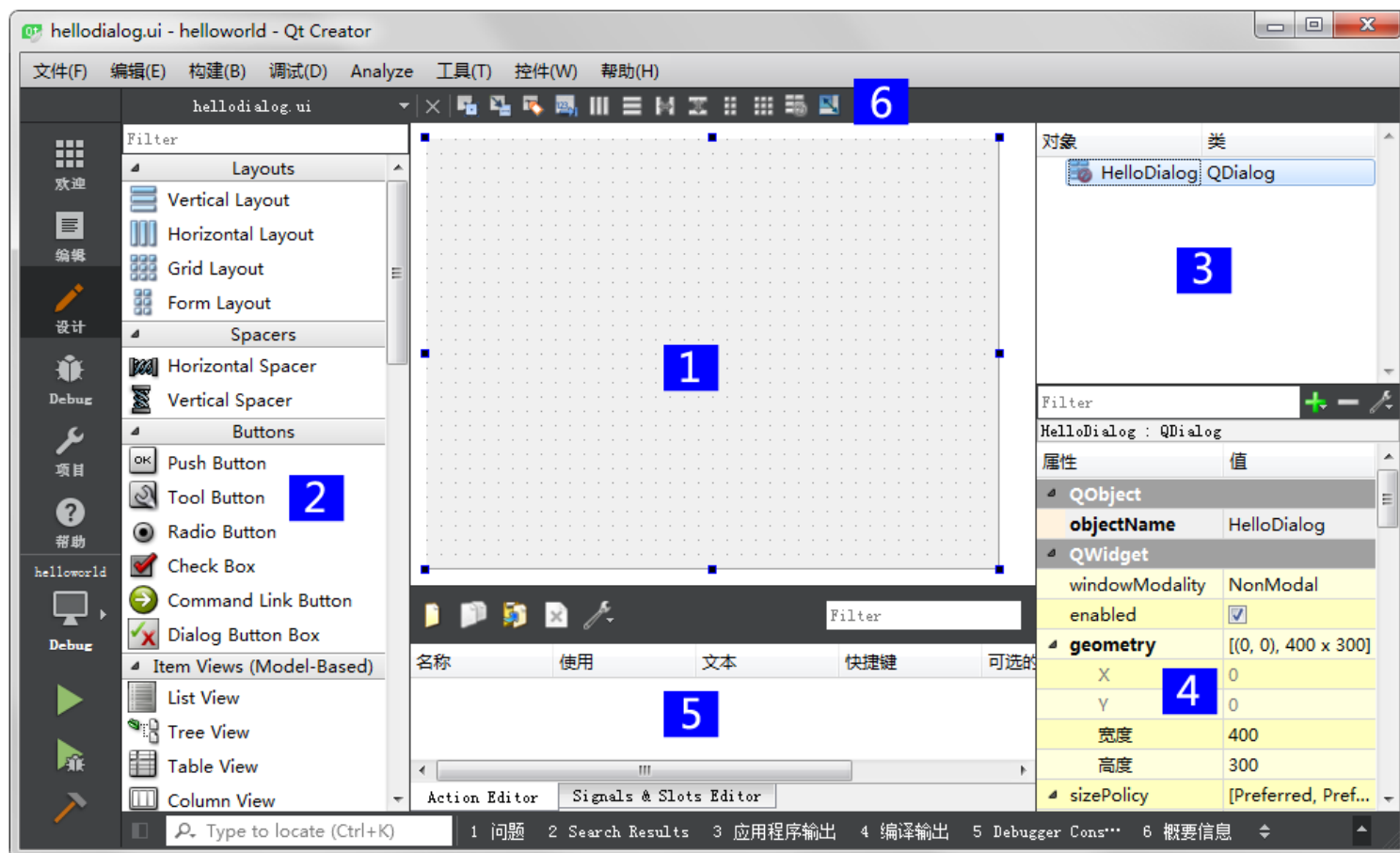
项目目录中的文件说明

文件	说明
<code>helloworld.pro</code>	该文件是项目文件，其中包含了项目相关信息
<code>helloworld.pro.user</code>	该文件中包含了与用户有关的项目信息
<code>hellodialog.h</code>	该文件是新建的 <code>HelloDialog</code> 类的头文件
<code>hellodialog.cpp</code>	该文件是新建的 <code>HelloDialog</code> 类的源文件
<code>main.cpp</code>	该文件中包含了 <code>main()</code> 主函数
<code>hellodialog.ui</code>	该文件是设计师设计的界面对应的界面文件



设计模式

在Qt Creator的编辑模式下双击项目文件列表中界面文件分类下的hellodialog.ui文件，这时便进入了设计模式。

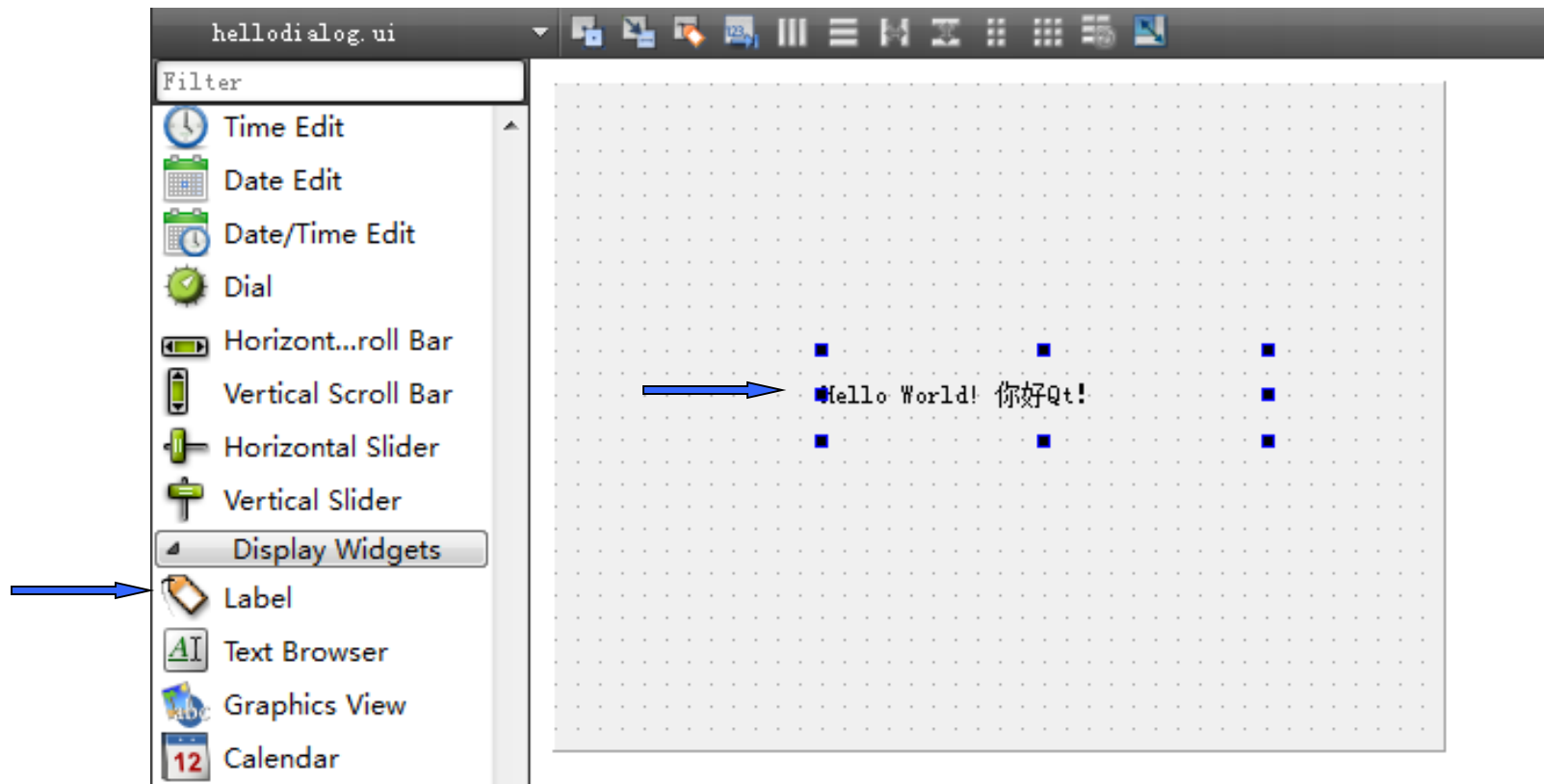


- ①主设计区。就是图中的中间部分，这里主要用来设计界面以及编辑各个部件的属性。
- ②部件列表窗口（Widget Box）。这里分类罗列了各种常用的标准部件，可以使用鼠标将这些部件拖入主设计区中，放到主设计区中的界面上。
- ③对象查看器（Object Inspector）。这里列出了界面上所有部件的对象名称和父类，而且以树形结构显示了各个部件的所属关系。可以在这里单击对象来选中该部件。
- ④属性编辑器（Property Editor）。这里显示了各个部件的常用属性信息，可以在这里更改部件的一些属性，如大小、位置等。这些属性按照从祖先继承的属性、从父类继承的属性和自己的属性的顺序进行了分类。
- ⑤动作（Action）编辑器与信号和/槽编辑器。在这里可以对相应的对象内容进行编辑。因为现在还没有涉及这些内容，所以放到以后使用时再介绍。
- ⑥常用功能图标。单击最上面的侧边栏中的前4个图标可以进入相应的模式，分别是窗口部件编辑模式（这是默认模式）、信号/槽编辑模式、伙伴编辑模式和Tab顺序编辑模式。后面的几个图标用来实现添加布局管理器以及调整大小等功能。



设计界面

从部件列表中找到Label（标签）部件，然后按着鼠标左键将它拖到主设计区的界面上，再双击它进入编辑状态后输入“Hello World! 你好Qt!”字符串。



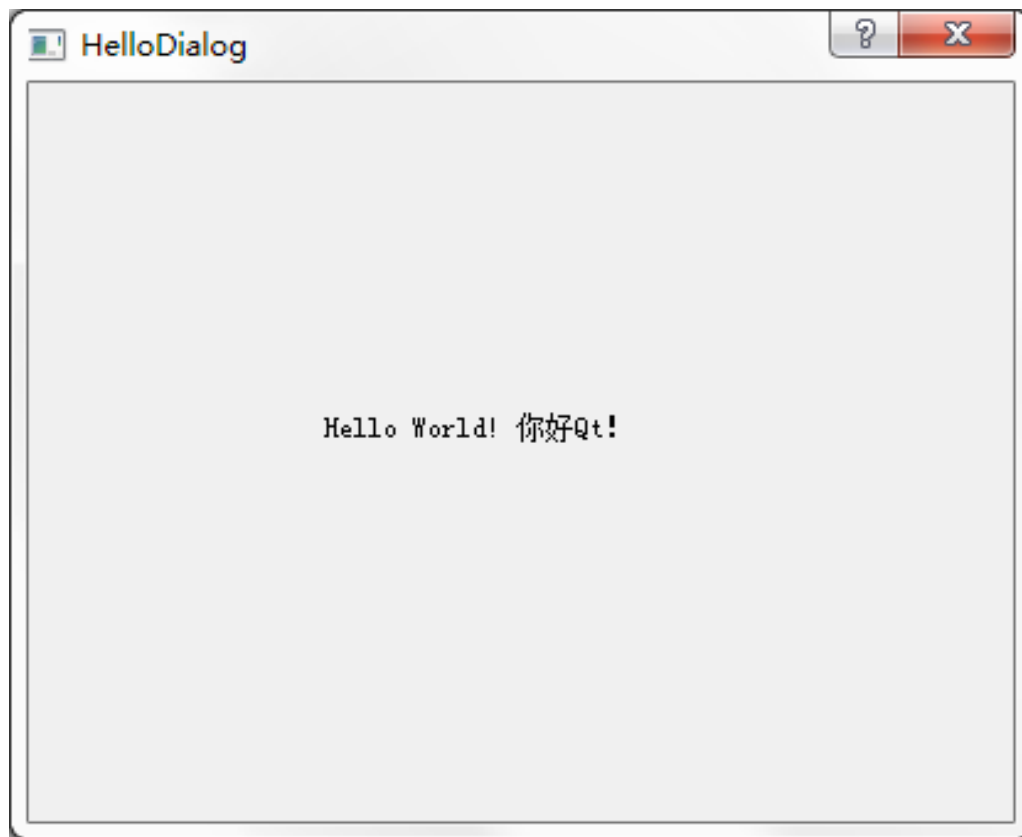
2.2 程序的运行与发布

- 程序的运行
- 程序的发布
- 设置应用程序图标



程序的运行

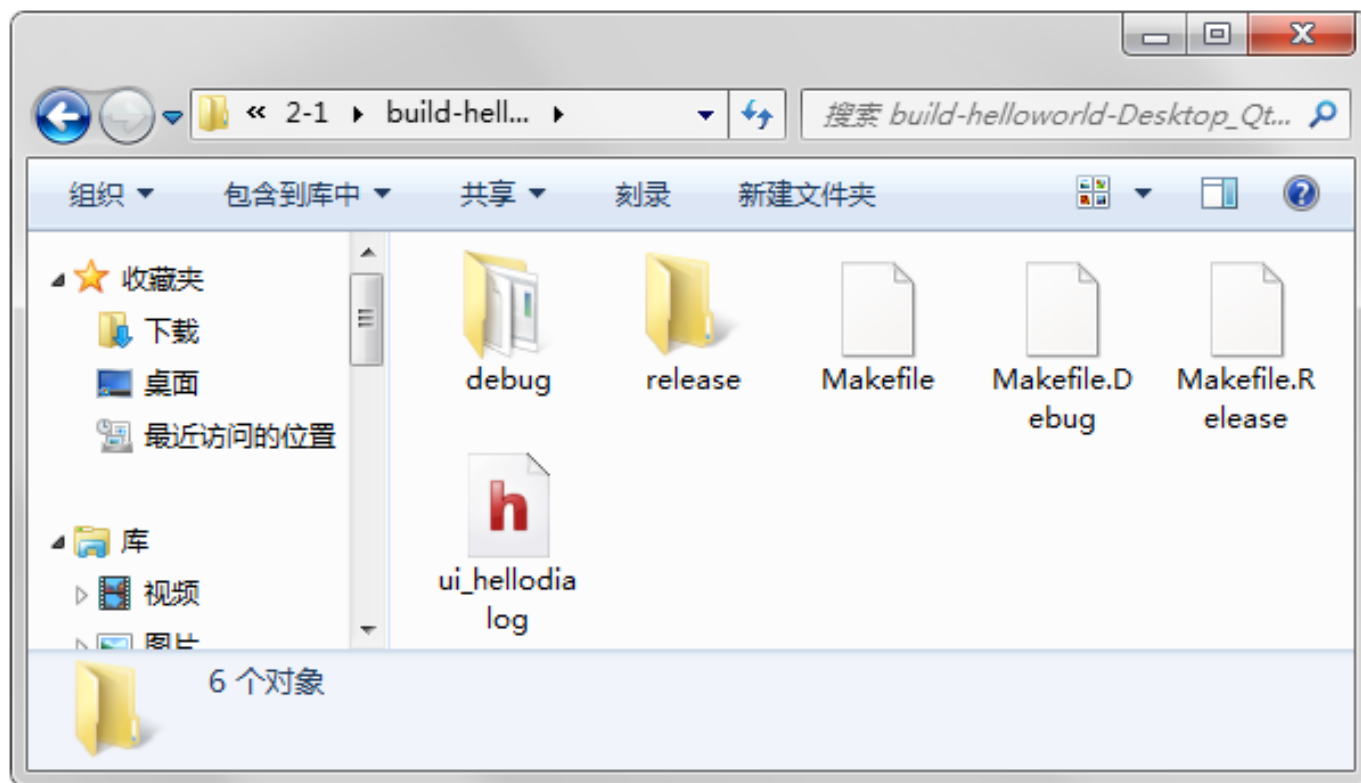
可以使用快捷键Ctrl+R或者通过按下左下角的运行按钮来运行程序。



生成的文件

现在项目目录中的文件可以发现，E:\app\src\02\2-1目录下又多了一个文件夹，这是默认的构建目录：

build-helloworld-Desktop_Qt_5_6_1_MinGW_32bit-Debug



文件说明

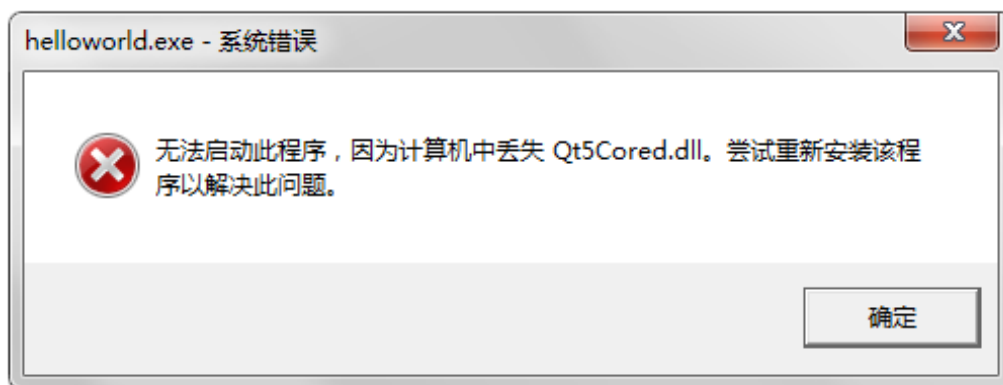
- Qt Creator将项目源文件和编译生成的文件进行了分类存放。
- helloworld文件夹中是项目源文件，而现在这个文件夹存放的是编译后生成的文件。进入该文件夹可以看到这里有3个Makefile文件和一个ui_hellodialog.h文件，还有两个目录debug和release。
- 现在release文件夹是空的，进入debug文件夹，这里有3个.o文件和一个.cpp文件，它们是编译时生成的中间文件，可以不必管它，而剩下的一个helloworld.exe文件便是生成的可执行文件。



直接运行生成的可执行文件

双击helloworld.exe运行，则弹出了下图所示的警告对话框，提示缺少Qt5Cored.dll文件：

原因：应用程序运行是需要dll动态链接库的。



解决办法一：复制DLL文件

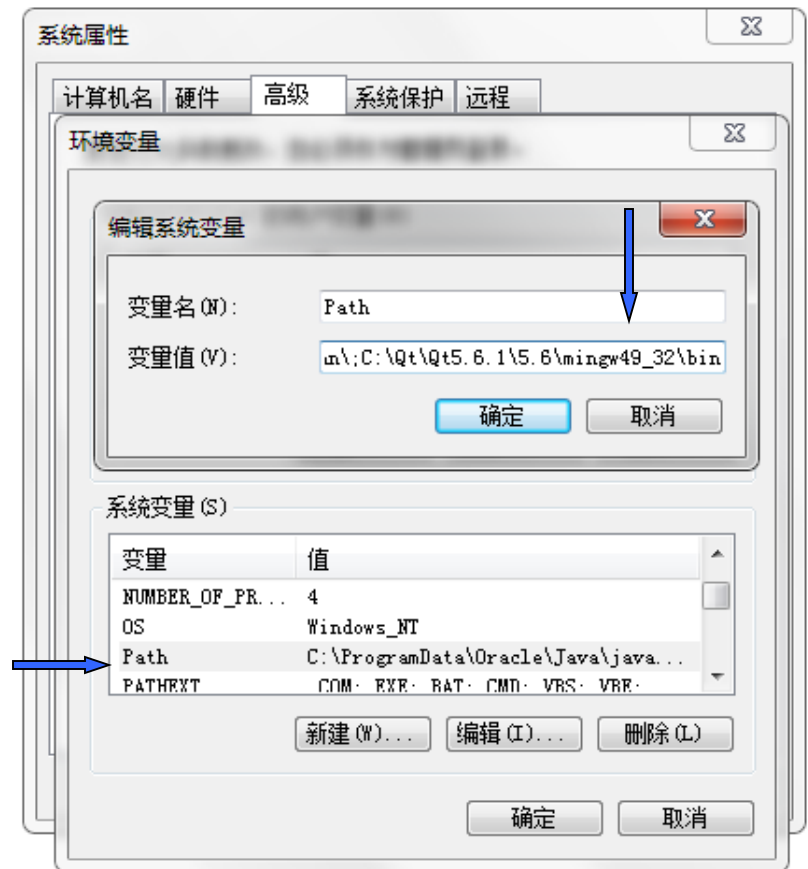
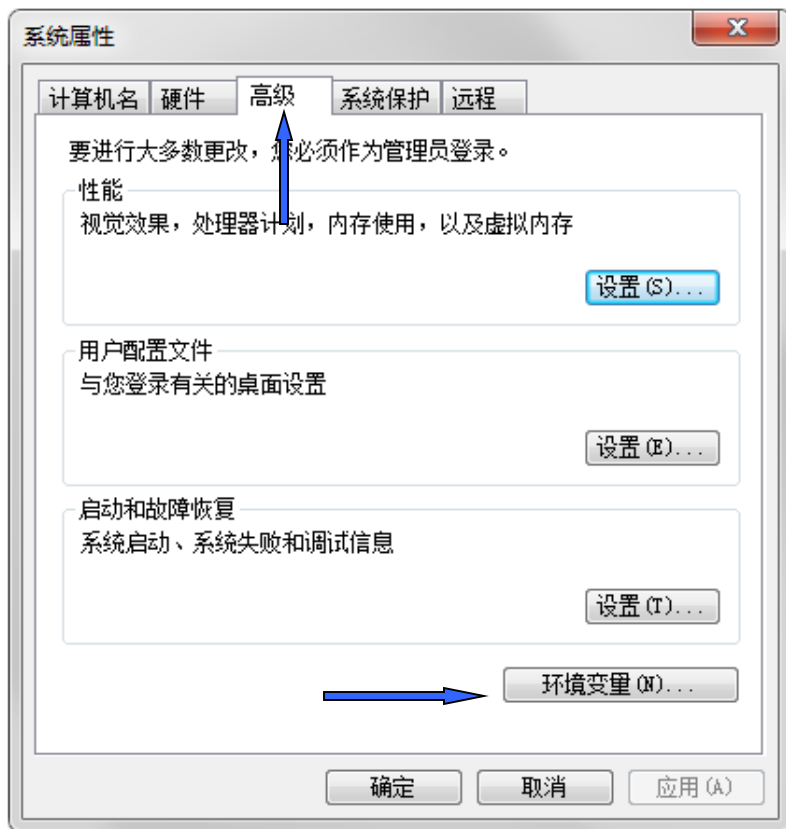
- 在Qt安装目录的bin目录（例如：
C:\Qt\Qt5.6.1\5.6\mingw49_32\bin）把这里的
Qt5Cored.dll文件复制到debug文件夹中。
- 运行程序提示缺少其他的文件，可以依次将它们复制过来，一共有6个文件。
- 再次运行程序发现已经没有问题了。



解决办法二：设置环境变量

可以直接将C:\Qt\Qt5.6.1\5.6\mingw49_32\bin目录加入到系统Path环境变量中去，这样程序运行时就可以自动找到bin目录中的dll文件了。

(注意：“; C:\Qt\Qt5.6.1\5.6\mingw49_32\bin”前面应该有个英文分号。)



程序的发布

现在程序已经编译完成，那么怎样来发布它，让它在别人的计算机上也能运行呢？

首先在Qt Creator中对helloworld程序进行release版本的编译。在左下角的目标选择器（Target selector）中将构建目标设置为Release。



将可执行文件和DLL文件放在一起

- 编译完成之后再看工程目录中build-helloworld-Desktop_Qt_5_6_1_MinGW_32bit-Release文件夹的release目录中，已经生成了helloworld.exe文件。
- 新建一个文件夹，重命名为“我的第一个Qt程序”，然后将release文件夹中的helloworld.exe复制过来，再去Qt安装目录的bin目录中将libgcc_s_dw2-1.dll、libstdc++-6.dll、libwinpthread-1.dll、Qt5Core.dll、Qt5Gui.dll和Qt5Widgets.dll这6个文件复制过来，。另外，还需要将C:\Qt\Qt5.6.1\5.6\mingw49_32\plugins目录中的platforms文件夹复制过来（不要修改该文件夹名称），里面只需要保留qwindows.dll文件即可。
- 现在整个文件夹一共有19MB，如果使用WinRAR等打包压缩软件对它进行压缩，就只有6MB了，已经到达了可以接受的程度，这时就可以将压缩包发布出去了。





提示

若程序中使用了png以外格式的图片，在发布程序时就要将Qt安装目录下的plugins目录中的imageformats文件夹复制到发布程序文件夹中，其中只要保留自己用到的文件格式的dll文件即可。

- 例如用到了gif文件，那么只需要保留qgif.dll。
- 而如果程序中使用了其他的模块，比如数据库，那么就要将plugins目录中的sqldrivers文件夹复制过来，里面保留自己用到的数据库驱动。



自学内容：静态编译

静态编译是相对于前面讲到的动态编译而言的。因为就像前面看到的一样，在Qt Creator默认的情况下，编译的程序要想发布就需要包含dll文件，这种编译方式被称为动态编译。

而静态编译就是将Qt的库进行重新编译，用静态编译的Qt库来链接程序，这样生成的目标文件就可以直接运行，而不再需要dll文件的支持了。不过这样生成的exe文件也就很大了，有好几MB，而且静态编译缺乏灵活性，也不能够部署插件。

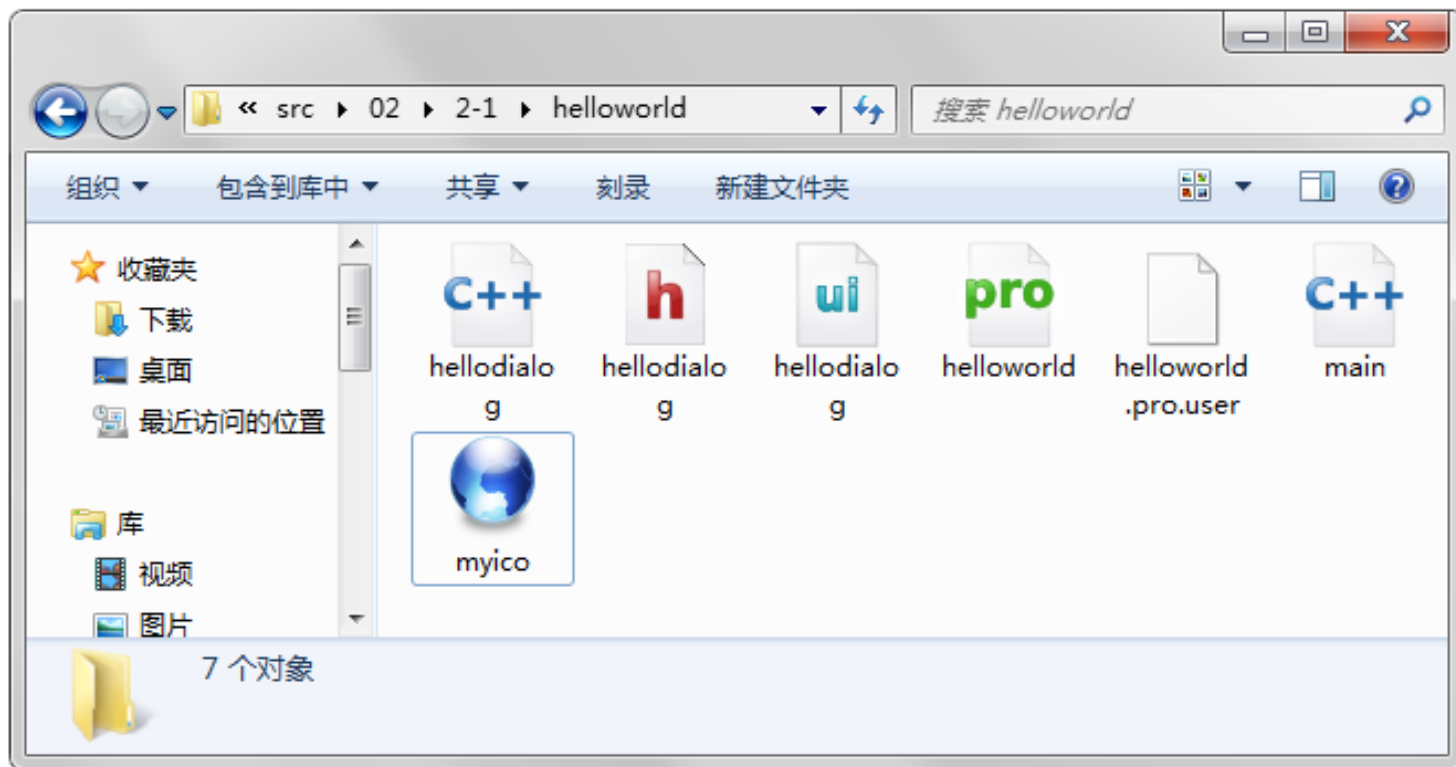
从前面的介绍可以看到，其实发布程序时带几个dll文件并不是很复杂的事情，而且如果要同时发布多个应用程序还可以共用dll文件，所以使用默认的方式就可以了。

想了解更多Qt发布的知识和静态编译的方法，可以在Qt Creator帮助的索引方式下查看Deploying Qt Applications关键字，Windows平台发布程序对应的关键字是Deploying an Application on Windows。



设置应用程序图标

第一步，创建.ico文件。将ico图标文件复制到工程文件夹的helloworld目录中，重命名为“myico.ico”。



第二步，修改项目文件。在Qt Creator中的编辑模式双击 helloworld.pro 文件，在最后面添加下面一行代码：

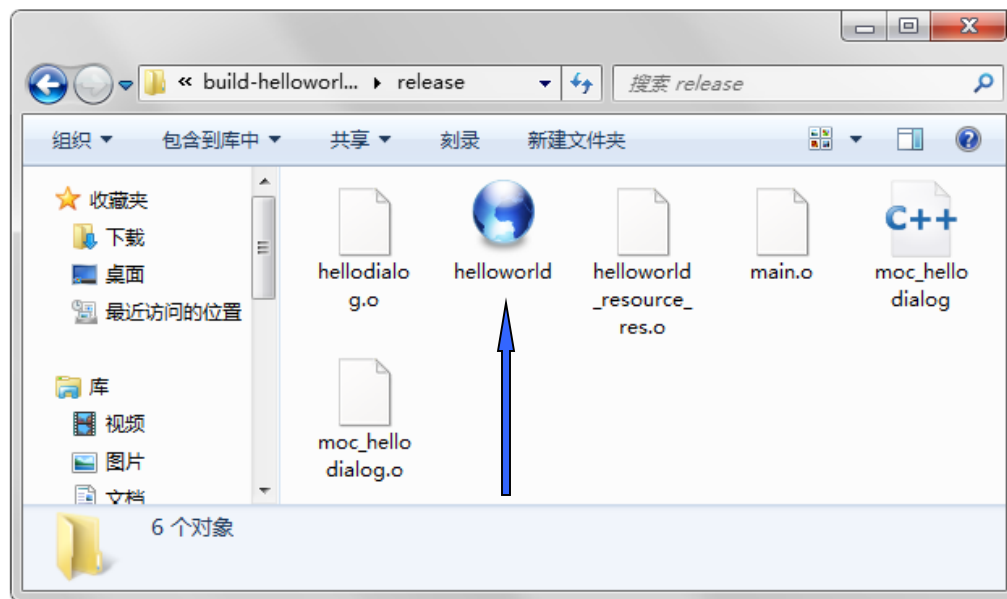
RC_ICONS = myico.ico

```
6
7 QT      += core gui
8
9 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
10
11 TARGET = helloworld
12 TEMPLATE = app
13
14
15 SOURCES += main.cpp\
16          hellodialog.cpp
17
18 HEADERS  += hellodialog.h
19
20 FORMS    += hellodialog.ui
21
22 RC_ICONS = myico.ico
23
```



第三步，运行程序。可以看到窗口的左上角的图标已经更换了。

然后查看一下release文件夹中的文件，可以看到现在exe文件已经更换了新的图标。



2.3 helloworld程序源码与编译过程详解

在Qt Creator中创建的helloworld项目，可以看到绝大多数的工作都是自动完成的。但是需要思考以下几个问题：

- 生成的项目目录中的各个文件都是什么？
- 它们有什么作用？
- 相互之间有什么联系？
- 还有Qt程序到底是怎么编译运行的？

解决这些问题对于学习Qt编程至关重要。下面通过手动编写、编译程序来寻找答案。



创建Qt程序方式一：纯代码编写程序

使用下面两种方式：

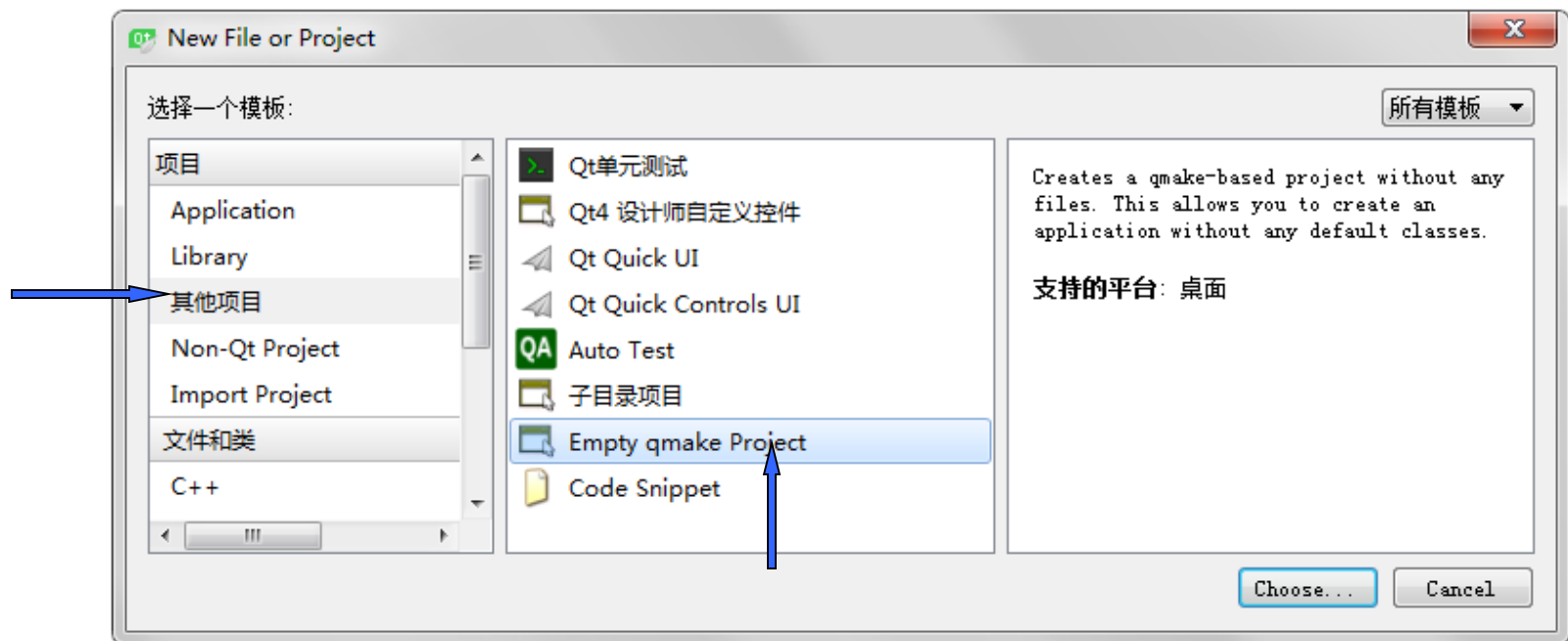
- 在Qt Creator中使用纯代码编写helloworld程序并编译运行。
- 使用普通文本编辑器编写helloworld程序，并在命令行中编译运行。



方式一：在Qt Creator中用纯代码编写程序

第一步，新建空项目。打开Qt Creator，并新建项目，选择“其他项目”中的“Empty qmake Project”。然后将项目命名为helloworld并设置路径，例如E:\app\src\02\2-2。完成后，双击helloworld.pro文件，添加如下一行代码：

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```



- **第二步，往项目中添加main.cpp文件。在项目文件列表中的项目文件夹helloworld上右击，选择“添加新文件”一项，然后选择C++ Source File，名称设置为main.cpp，路径就是默认的项目目录，后面的选项保持默认即可。**
- **第三步，编写源代码。向新建的main.cpp文件中添加如下代码：**

```
1  #include <QApplication>
2  #include <QDialog>
3  #include <QLabel>
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      QDialog w;
8      QLabel label(&w);
9      label.setText("Hello World! 你好Qt! ");
10     w.show();
11     return a.exec();
12 }
```

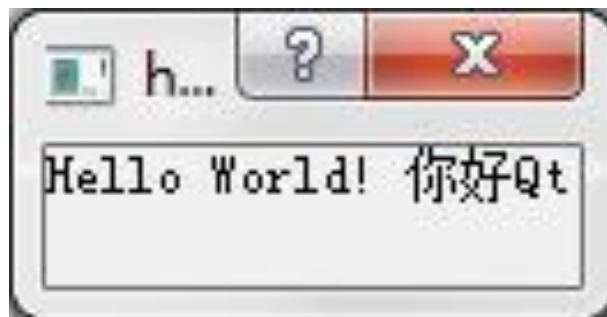


代码解释

- 前3行是头文件包含。在Qt中每一个类都有一个与其同名的头文件，因为后面用到了QApplication、QDialog和QLabel这3个类，所以这里要包含这些类的定义。
- 第4行就是在C++中最常见到的main()函数，它有两个参数，用来接收命令行参数。
- 在第6行新建了QApplication类对象，用于管理应用程序的资源，任何一个Qt GUI程序都要有一个QApplication对象。因为Qt程序可以接收命令行参数，所以它需要argc和argv两个参数。
- 第7行新建了一个QDialog对象，QDialog类用来实现一个对话框界面。
- 第8行新建了一个QLabel对象，并将QDialog对象作为参数，表明了对话框是它的父窗口，也就是说这个标签放在对话框窗口中。
- 第9行给标签设置要显示的字符。
- 第10行让对话框显示出来。在默认情况下，新建的可视部件对象都是不可见的，要使用show()函数让它们显示出来。
- 第11行让QApplication对象进入事件循环，这样当Qt应用程序在运行时便可以接收产生的事件，例如单击和键盘按下等事件。



第四步，编译运行。再看运行的程序，发现窗口太小，下面继续更改代码。



第五步，设置窗口大小。

```
1  #include <QApplication>
2  #include <QDialog>
3  #include <QLabel>
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      QDialog w;
8      w.resize(400, 300);
9      QLabel label(&w);
10     label.move(120, 120);
11     label.setText(QObject::tr("Hello World! 你好Qt! "));
12     w.show();
13     return a.exec();
14 }
```



- 如果想改变对话框的大小，可以使用QDialog类中的函数来实现。
- 在第7行代码下面另起一行，输入“w.”（注：w后面输入一个“.”），这时会弹出QDialog类中所有成员的列表，可以使用键盘的向下方向键↓来浏览列表，根据字面意思，这里选定了resize()函数。这时按下Enter键，代码便自动补全，并且显示出了resize()函数的原型。它有两个重载形式，用键盘方向键来查看另外的形式，这里的“int w, int h”应该就是宽和高了。
- 所以写出了第8行代码，设置对话框宽为400，高为300（它们的单位是像素）。还要说明的是，编写代码时所有的符号都要用输入法中的英文半角（当然，中文字符串中的除外）。然后在第10行代码中设置了label在对话框中的位置，默认的，对话框的左上角是(0, 0)点。
- 第11行添加的QObject::tr()函数可以实现多语言支持，一般建议程序中所有要显示到界面上的字符串都使用tr()函数括起来，这个在第9章国际化部分将会讲到。

提示

自动补全功能

在编辑器中敲入代码时可以发现当打完开头几个字母后就会出现相关的列表选项，这些选项都是以这些字母开头的。现在要说明的是，如果要输入一个很长的字符，比如setWindowTitle，那么可以直接输入swt三个字母（就是setWindowTitle中首字母加其中的大写字母）来快速定位它，然后按下Enter按键就可以完成输入。

也可以使用Ctrl+空格键来强制代码补全，需要注意它可能与使用的输入法的快捷键冲突。

```
QApplication a(argc, argv);
QDialog w;
w.resize(400, 300);
w.swt
  ◆ setWhatsThis
  ◆ setWindowTitle void setWindowTitle(const QString &)
w.show();
return a.exec();
```



列表中 各个图 标都代 表一种 数据类型：

Icon	Description	
	A class	类
	An enum	枚举类型
	An enumerator (value of an enum)	枚举类型值
	A function	函数
	A private function	私有函数
	A protected function	受保护函数
	A variable	变量
	A private variable	私有变量
	A protected variable	受保护变量
	A signal	信号
	A slot	槽
	A private slot	私有槽
	A protected slot	受保护槽
	A C++ keyword	C++关键字
	A C++ code snippet	C++代码段
	A QML type	QML类型
	A QML code snippet	QML代码段
	A macro	宏
	A namespace	命名空间

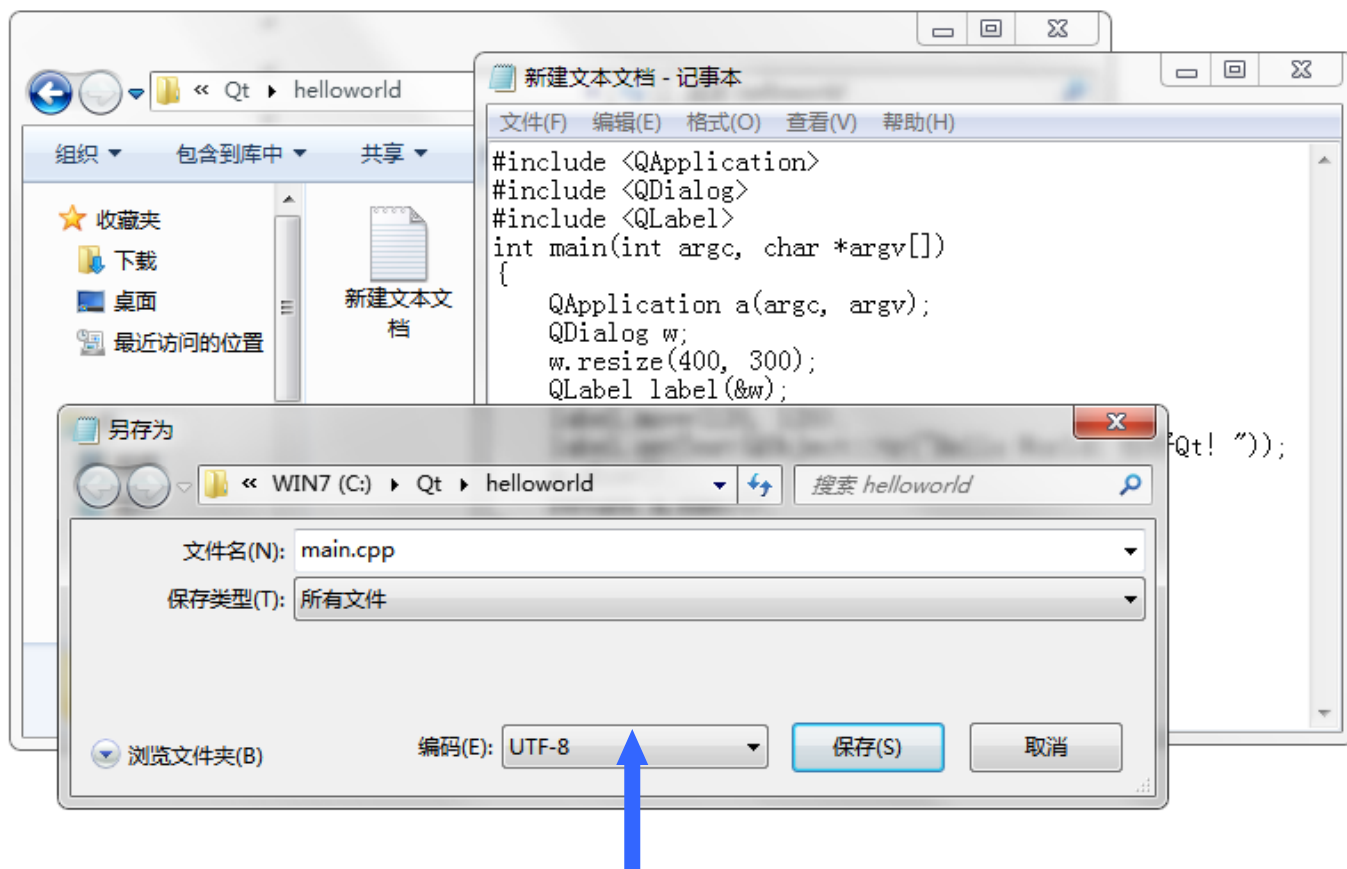


方式二：使用普通文本编辑器编写helloworld程序

前面在Qt Creator中使用纯代码实现了helloworld程序。下面不使用Qt Creator，而是在其他的编辑器（如Windows的记事本）中编写源码，然后再到命令行去编译运行该程序。



第一步，新建工程目录。在Qt的安装目录（例如：C:\Qt）中，新建文件夹helloworld，然后在其中新建文本文档，将Qt Creator中main.cpp文件中的所有内容复制过来，并将文件另存为main.cpp（在保存时要将编码选择为UTF-8，否则中文无法显示）。



第二步，使用命令编译程序。打开开始菜单中Qt安装目录下的命令提示符程序Qt 5.6 for Desktop (MinGW 4.9.2 32 bit)。这里已经配置好了编译环境。

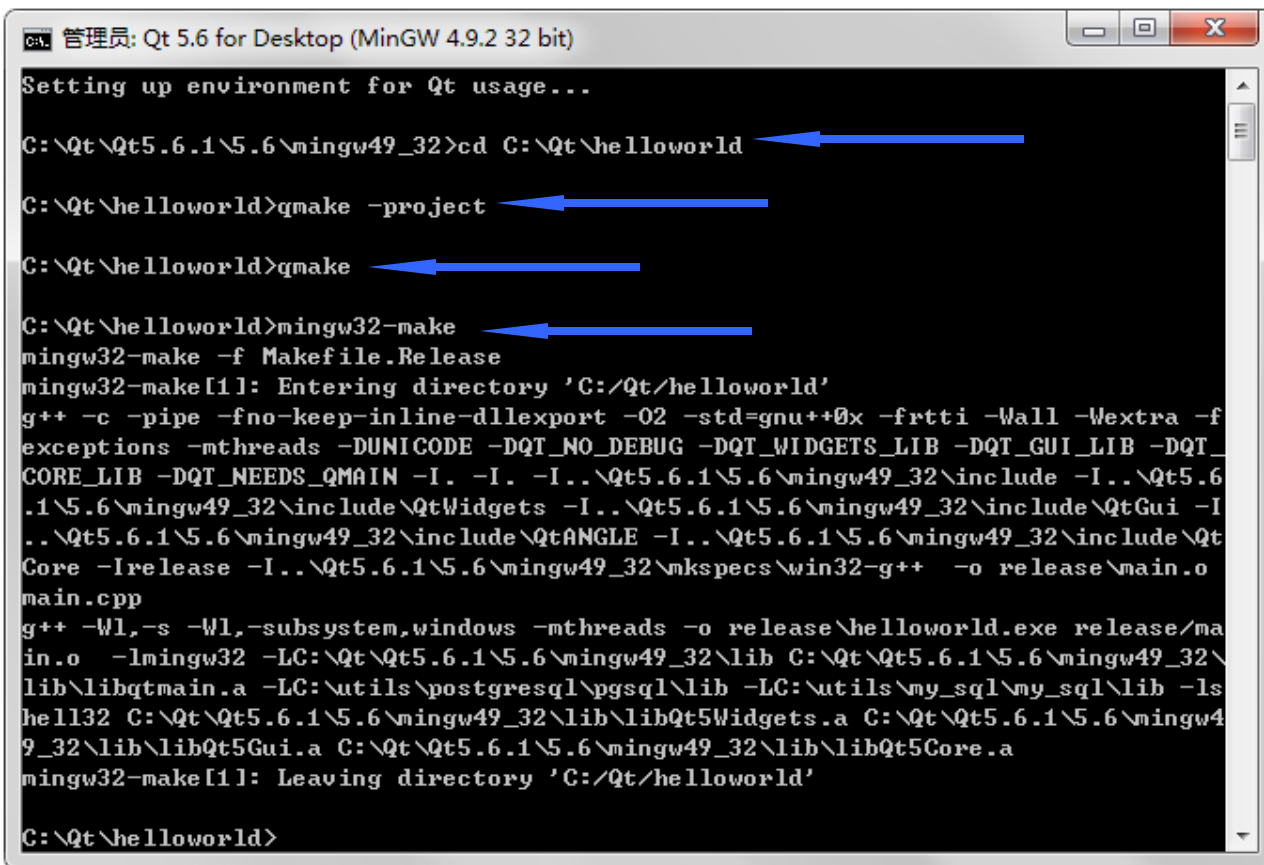
- 现在的默认路径在C:\Qt\Qt5.6.1\5.6\mingw49_32下
- 输入命令：“cd C:\Qt\helloworld”跳转到新建的helloworld目录中。
- 然后再输入“qmake -project”命令来生成pro工程文件，这时可以看到在helloworld目录中已经有了helloworld.pro文件了。
- 下面使用记事本打开该文件，然后在最后面添加如下一行代码：

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

- 下面接着输入“qmake”命令来生成用于编译的Makefile文件。这时在

helloworld目录中出现了Makefile文件和debug目录与release目录，当然这两个目录现在是空的。

- 接下来输入“mingw32-make”命令来编译程序，编译完成后会在release目录中出现helloworld.exe文件。



```
管理员: Qt 5.6 for Desktop (MinGW 4.9.2 32 bit)
Setting up environment for Qt usage...
C:\Qt\Qt5.6.1\5.6\mingw49_32>cd C:\Qt\helloworld
C:\Qt\helloworld>qmake -project
C:\Qt\helloworld>qmake
C:\Qt\helloworld>mingw32-make
mingw32-make -f Makefile.Release
mingw32-make[1]: Entering directory 'C:/Qt/helloworld'
g++ -c -pipe -fno-keep-inline-dllexport -O2 -std=gnu++0x -frtti -Wall -Wextra -fexceptions -mthreads -DUNICODE -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -DQT_NEEDS_QMAIN -I. -I. -I..\Qt5.6.1\5.6\mingw49_32\include -I..\Qt5.6.1\5.6\mingw49_32\include\QtWidgets -I..\Qt5.6.1\5.6\mingw49_32\include\QtGui -I..\Qt5.6.1\5.6\mingw49_32\include\QtANGLE -I..\Qt5.6.1\5.6\mingw49_32\include\QtCore -Irelease -I..\Qt5.6.1\5.6\mingw49_32\mkspecs\win32-g++ -o release\main.o main.cpp
g++ -Wl,-s -Wl,-subsystem,windows -mthreads -o release\helloworld.exe release\main.o -lmingw32 -LC:\Qt\Qt5.6.1\5.6\mingw49_32\lib C:\Qt\Qt5.6.1\5.6\mingw49_32\lib\libqtmain.a -LC:\utils\postgresql\pgsql\lib -LC:\utils\my_sql\my_sql\lib -ls hell32 C:\Qt\Qt5.6.1\5.6\mingw49_32\lib\libQt5Widgets.a C:\Qt\Qt5.6.1\5.6\mingw49_32\lib\libQt5Gui.a C:\Qt\Qt5.6.1\5.6\mingw49_32\lib\libQt5Core.a
mingw32-make[1]: Leaving directory 'C:/Qt/helloworld'
C:\Qt\helloworld>
```



编译说明：

上面使用的qmake是Qt提供的一个编译工具，它可以生成与平台无关的.pro文件，然后利用该文件生成与平台相关的Makefile文件。

Makefile文件中包含了要创建的目标文件或可执行文件、创建目标文件所依赖的文件和创建每个目标文件时需要运行的命令等信息。最后使用mingw32-make命令来完成自动编译，通过读入Makefile文件的内容来执行编译工作。使用mingw32-make命令时会为每一个源文件生成一个对应的.o目标文件，最后将这些目标文件进行链接来生成最终的可执行文件。

第三步，运行程序。在命令行接着输入cd release命令，跳转到release目录下，然后再输入“helloworld.exe”，按下回车键，就会运行helloworld程序了。



创建Qt程序方式二：使用.ui文件

.ui文件是使用设计模式生成的文件，包含了界面设计的内容。下面通过两种方式来使用该文件：

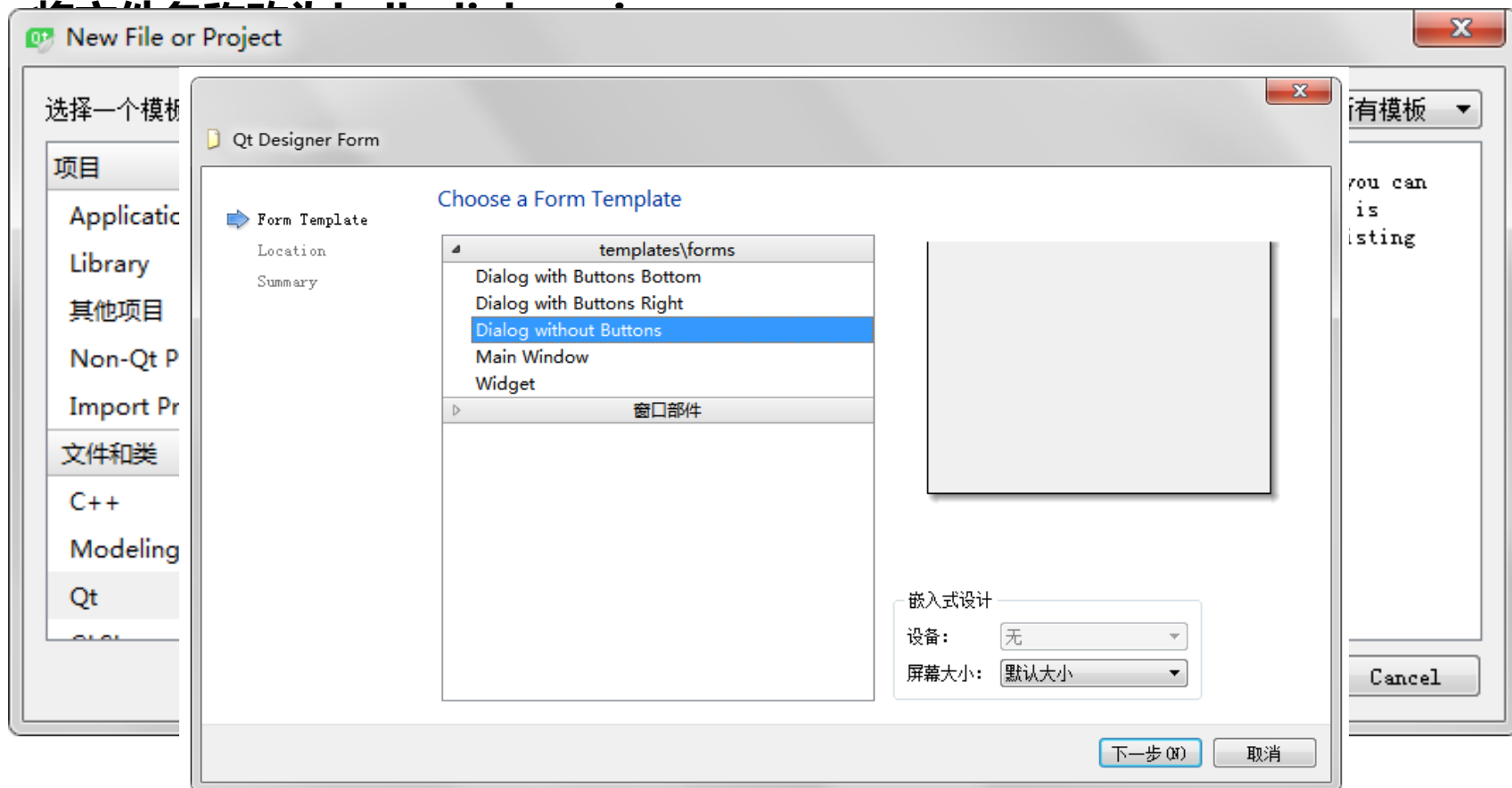
- 在Qt Creator中使用.ui界面文件
- 在命令行手动编译ui文件和程序



方式一：在Qt Creator中使用.ui界面文件

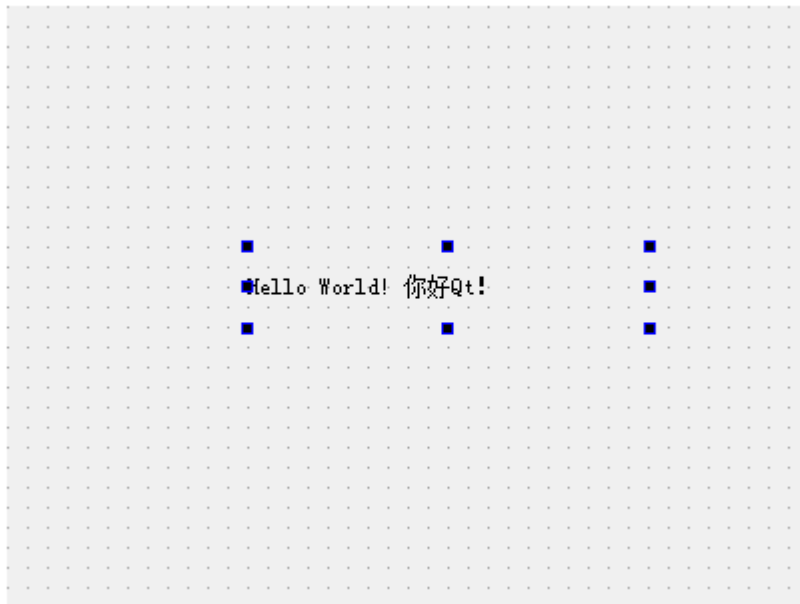
第一步，添加.ui文件。向工程中继续添加文件。在模板中选择Qt中的“Qt Designer Form”。

在选择界面模板时选择Dialog without Buttons项。再单击“下一步”，



第二步，设计界面。生成好文件后便进入了设计模式，在界面上添加一个Label部件，并且更改其显示内容为“Hello World! 你好Qt!”。

- **在右侧的属性栏的geometry属性中更改其坐标位置为“X: 120, Y: 120”。这样就与那行代码“label.move(120, 120);”起到了相同的作用。**
- **在右上角的类列表中选择QDialog类对象，并且在下面的属性中更改它的对象名objectName为“HelloDialog”。**




label : QLabel	
属性	值
QObject	
objectName	label
QWidget	
enabled	<input checked="" type="checkbox"/>
geometry	[(120, 120), 201 x 41]
X	120
Y	120

对象	类
label	QLabel
HelloDialog	QDialog

Filter	
HelloDialog : QDialog	
属性	值
QObject	
objectName	HelloDialog



第三步，生成ui头文件。这时按下Ctrl+S快捷键保存修改，然后按下Ctrl+2快捷键回到编辑模式，那么就会看到.ui文件的内容了，它是一个XML文件，里面是界面部件的相关信息。使用Ctrl+Shift+B快捷键或者左下角的  来构建工程。然后到本地磁盘的项目目录的build-helloworld-Desktop_Qt_5_6_1_MinGW_32bit-Debug目录中，就可以看到由.ui文件生成的ui_hellodialog.h头文件了。

自学内容：研究hellodialog.ui文件和ui_hellodialog.h文件的内容。



第四步，更改main.cpp文件。将main.cpp文件中的内容更改如下：

```
#include "ui_hellodialog.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QDialog w;
    Ui::HelloDialog ui;
    ui.setupUi(&w);
    w.show();
    return a.exec();
}
```

第五步，运行程序。

第1行代码是头文件包含。因为在ui_hellodialog.h中已经包含了其他类的定义，所以这里只需要包含这个文件就可以了。对于头文件的包含，使用“<>”时，系统会到默认目录（编译器及环境变量、工程文件所定义的头文件寻找目录，包括Qt安装的include目录，即“C:\Qt\Qt5.6.1\5.6\mingw49_32\include”）查找要包含的文件，这是标准方式；用双引号时，系统先到用户当前目录（即项目目录）中查找要包含的文件，找不到再按标准方式查找。因为ui_hellodialog.h文件在我们自己的项目目录中，所以使用了双引号包含。

第6行代码使用命名空间Ui中的HelloDialog类定义了一个ui对象。

在第7行中使用了setupUi()函数，并将对话框类对象作为参数，这样就可以将设计好的界面应用到对象w所表示的对话框上



方式二：在命令行手动编译ui文件和程序

- 第一步，新建工程目录。在C:\Qt目录中新建文件夹helloworld 2，然后将上面的项目文件夹helloworld目录下的hellodialog.ui和main.cpp两个文件复制过来。
 - 第二步，编译.ui文件。打开命令提示符程序，然后输入：
cd C:\Qt\helloworld 2命令进入helloworld_2文件夹中。再使用uic编译工具，从ui文件生成头文件。具体命令是：
uic -o ui_hellodialog.h hellodialog.ui
就像前面看到的那样，ui文件生成的默认头文件的名称是“ui_”加ui文件的名称。这时在helloworld_2目录中已经生成了相应的头文件。
 - 第三步，编译运行程序。输入如下命令：
 - qmake -project
 - 这时在helloworld_2.pro文件中添加QT += widgets，然后依次执行如下命令：
 - qmake
 - mingw32-make
 - cd release
 - helloworld 2.exe
- 这样就完成了整个编译运行过程。可以看到ui文件是使用uic编译工具来编译的，这里的Qt程序通过调用相应的头文件来使用ui界面文件。



创建Qt程序方式三：自定义C++类

首先新建空工程并且建立自己定义的一个C++类，然后再使用上一小节的.ui文件。

- **第一步，新建空的Qt项目Empty qmake Project。项目名称为helloworld，添加如下代码并保存该文件：**
- **greaterThan(QT_MAJOR_VERSION, 4): QT += widgets**
- **第二步，添加文件。向项目中添加新文件，选择C++ Class。类名为HelloDialog，基类Base class选择自定义<Custom>，然后在下面手动填写为QDialog，其他保持默认。添加完成后再往项目中添加main.cpp文件。**



- **第三步，编写源码。在main.cpp中添加如下代码：**

```
#include <QApplication>
#include "hellodialog.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    HelloDialog w;
    w.show();
    return a.exec();
}
```

这里在第2行添加了新建的HelloDialog类的头文件，然后在第6行定义了一个该类的对象。这时运行程序，它会显示一个空白的对话框。

- **第四步，添加ui文件。将前面建立的hellodialog.ui文件复制到项目目录下，然后在Qt Creator中的项目文件列表中的项目文件夹上右击，在弹出的菜单中选择“添加现有文件”，然后在弹出的对话框中选择helloworld.ui文件，将其添加到项目中。**



第五步，更改C++类文件。这次不在main()函数中使用ui文件，而是在新建立的C++类中使用。先将头文hellodialog.h，修改如下：

```
1  #ifndef HELLODIALOG_H
2  #define HELLODIALOG_H
3
4  #include <QDialog>
5
6  namespace Ui{
7  class HelloDialog;
8  }
9
10 class HelloDialog : public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     explicit HelloDialog(QWidget *parent = 0);
16     ~HelloDialog();
17
18 private:
19     Ui::HelloDialog *ui;
20 };|
21
22 #endif // HELLODIALOG_H
```

第1，2和22行是预处理指令，避免该头文件多重包含。

第6~8行定义了命名空间Ui，并在其中前置声明了HelloDialog类，这个类就是在ui_hellodialog.h文件中看到的那个类。因为它与新定义的类同名，所以使用了Ui命名空间。而前置声明是为了加快编译速度，也可以避免在一个头文件中随意包含其他头文件而产生错误。因为这里只使用了该类对象的指针，如第19行定义了该类对象的指针，这并不需要该类的完整定义，所以可以使用前置声明。这样就不用在这里添加ui_hellodialog.h的头文件包含，而可以将其放到hellodialog.cpp文件中进行。

第10行是新定义的HelloDialog类，继承自QDialog类。

第12行定义了Q_OBJECT宏，扩展了普通C++类的功能，比如下一章要讲的信号和槽功能。必须在类定义的最开始处来定义这个宏。

第15行是显式构造函数，参数是用来指定父窗口的，默认是没有父窗口的。

第16行是析构函数。



然后在hellodialog.cpp文件中添加代码：

```
#include "hellodialog.h"
#include "ui_hellodialog.h"
HelloDialog::HelloDialog(QWidget *parent) :
    QDialog(parent)
{
    ui = new Ui::HelloDialog;
    ui->setupUi(this);
}
```

- 第2行添加了ui头文件，因为在hellodialog.h文件中只是使用了前置声明，所以头文件在这里添加。第6行创建Ui::HelloDialog对象。第7行设置setupUi()函数的参数为this，表示为现在这个类所代表的对话框创建界面。

也可以将ui的创建代码放到构造函数首部，代码如下：

```
HelloDialog::HelloDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::HelloDialog)
{
    ui->setupUi(this);
}
```

- 这样与前面的代码效果是相同的。现在已经写出了和前面使用Qt Creator创建的helloworld项目中相同的文件和代码。此时，可以再运行程序。



创建Qt程序方式四：使用Qt设计师界面类

- 再次新建空项目，名称仍为helloworld。完成后在项目文件中添加widgets模块调用代码，然后向该项目中添加新文件，模板选择Qt中的“Qt设计师界面类”。界面模板依然选择Dialog without Buttons一项，类名为HelloDialog。完成后在设计模式往窗口上添加一个Label，更改显示文本为“Hello World! 你好Qt! ”。然后再往项目中添加main.cpp文件，并更改其内容如下：

```
#include <QApplication>
#include "hellodialog.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    HelloDialog w;
    w.show();
    return a.exec();
}
```

- 这里将前面的内容进行了简化，因为Qt设计师界面类就是前面讲到的C++类和ui文件的结合，它将这两个文件一起生成了，而不用再一个的添加。



思考?

到这里就已经把Qt Creator自动生成Qt Widgets项目进行了分解再综合，一步一步地讲解了整个项目的组成和构建过程，可以看到Qt Creator做了很多事情，但是读者最好也学会自己建立空项目，然后依次往里面添加各个文件，这种方式更灵活。

前面讲到的几种方式有哪些相同、不同之处？从整个过程中你对编写、编译Qt程序了解了多少？



2.4 项目模式和项目文件介绍

helloworld

构建和运行 | 编辑器 | 代码风格 | 依赖关系 | Clang Static Analyzer

添加构建套件 ▼
管理构建套件...

Desktop Qt 5.6.1 MinGW 32bit
构建 运行

 **构建设置**
编辑构建配置: Debug ▼ 添加 ▼ 删除 重命名...

概要
Shadow build: ☒
构建目录: E:\app\src\2-1\build-helloworld-Desktop_Qt_5_6_1_MinGW_32bit-Debug 浏览...

构建步骤

qmake: qmake.exe helloworld.pro -r -spec win32-g++ "CONFIG+=debug" "CONFIG+=qml_debug" 详情 ▼

Make: mingw32-make.exe in E:\app\src\2-1\build-helloworld-Desktop_Qt_5_6_1_MinGW_32bit-Debug 详情 ▼

添加构建步骤 ▼

清除步骤

Make: mingw32-make.exe clean in E:\app\src\2-1\build-helloworld-Desktop_Qt_5_6_1_MinGW_32bit-Debug 详情 ▼

添加清理步骤 ▼

构建环境
使用 系统环境变量 详情 ▼



- 在构建和运行页面可以设置要构建的版本，如Debug版或是Release版本。还可以设置所使用的Qt版本。这里有一个Shadow build选项，就是所谓的“影子构建”，作用是将项目的源码和编译生成的文件分别存放，就像前面讲到的，helloworld项目经编译后会生成build-helloworld-Desktop Qt 5 6 1 MinGW 32bit-Debug文件夹，里面放着编译生成的所有文件。将编译输出与源代码分别存放是个很好的习惯，尤其是在使用多个Qt版本进行编译时更是如此。Shadow build选项默认是选中的，如果想让源码和编译生成的文件放在一个目录下，比如前面在命令行编译时那样，那么也可以将这个选项去掉。“构建步骤”、“清除步骤”和“构建环境”等选项，前面都已经提及过相关内容了，如果对编译命令不是很熟悉，这里保持默认就可以了，不用修改。
- 在编辑器设置中可以设置默认的文件编码、制表符和缩进、鼠标和键盘的相关功能，这些都是默认的全局设置，一般不建议修改，当然也可以按照自己的习惯进行自定义设置；在代码风格设置页面可以自定义代码风格，还可以将代码风格文件导出或者导入，这里默认使用了Qt的代码风格；在依赖关系中，如果同时打开了多个项目，可以设置它们之间的依赖关系；Qt Creator集成的Clang Static Analyzer可以用来发现C、C++和Objective-C程序中的问题，具体使用方法可以在帮助中索引Using Clang Static Analyzer关键字。在这些选项一般都不需要更改。



项目文件

下面是建立的helloworld项目的helloworld.pro文件的内容:

```
1 #-----
2 #
3 # Project created by QtCreator 2016-07-16T21:45:42
4 #
5 #-----
6
7 QT      += core gui
8
9 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
10
11 TARGET = helloworld
12 TEMPLATE = app
13
14
15 SOURCES += main.cpp\
16          hellodialog.cpp
17
18 HEADERS  += hellodialog.h
19
20 FORMS    += hellodialog.ui
21
22 RC_ICONS = myico.ico
```



第1~5行是注释信息，说明这个文件生成的时间。

第7行表明了这个项目使用的模块。**core**模块包含了Qt的核心功能，其他所有模块都依赖于这个模块；而**gui**模块提供了窗口系统集成、事件处理、OpenGL和OpenGL ES集成、2D图形、基本图像、字体和文本等功能。当使用**qmake**工具来构建项目时，**core**模块和**gui**模块是被默认包含的，这也是为什么前面手动编写项目文件时不添加这两个模块也可以编译的原因。其实所谓的模块，就是很多相关类的集合，读者可以在Qt帮助中查看Qt Core和Qt GUI关键字。

第9行添加了**widgets**模块，在Qt Widgets模块中提供了经典的桌面用户界面的UI元素集合，简单来说，所有C++程序用户界面部件都在该模块中。

第11行是生成的目标文件的名称，就是生成的**exe**文件的名称，默认的是项目的名称，当然也可以在这里改为别的名称。第12行使用**app**模板，表明这是个应用程序。

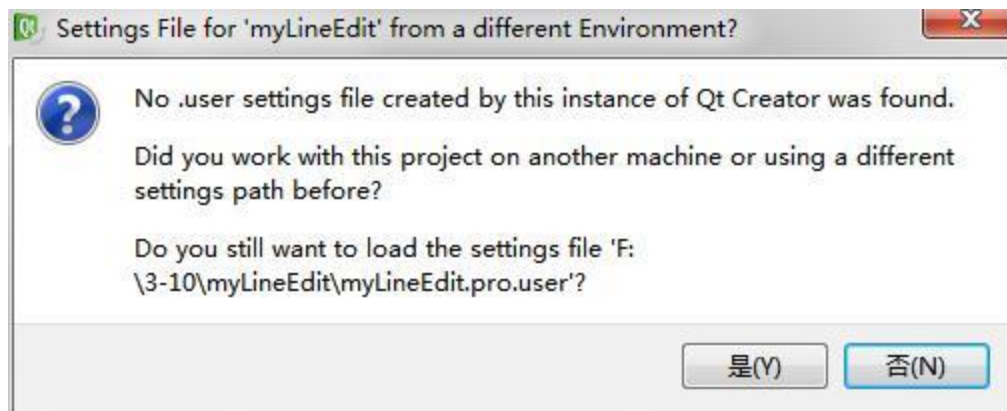
第15，18和20行分别是工程中包含的源文件、头文件和界面文件。

第22行就是添加的应用程序图标。这里这些文件都使用了相对路径，因为都在项目目录中，所以只写了文件名。



.pro.user文件

在项目文件夹中生成的.pro.user文件，它包含了本地构建信息，包含Qt版本和构建目录等。可以用记事本或者写字板将这个文件打开查看其内容。当使用Qt Creator打开一个.pro文件时会自动生成一个.pro.user文件。因为读者的系统环境都不太一样，Qt的安装于设置也不尽相同，所以如果要将自己的源码公开，一般不需要包含这个user文件。如果要打开别人的项目文件，但里面包含了user文件，Qt Creator则会弹出提示窗口，询问是否载入特定的环境设置，这时应该选择“否”，然后选择自己的Qt版本即可。



2.5 小结

本章虽然只是讲了一个最简单的Hello World程序，不过讲解了Qt项目从建立到编译运行，再到发布的全过程。其中还讲解了整个项目的组成与编译过程。而穿插在这些内容之中的是Qt Creator的一些基本操作和使用流程，比如建立各种应用、添加各种文件、设计界面和更改属性等等。



创建、编辑、编译、发布Qt项目的整个流程？

如何给Qt程序添加应用程序图标？

创建Qt程序的几种方式？

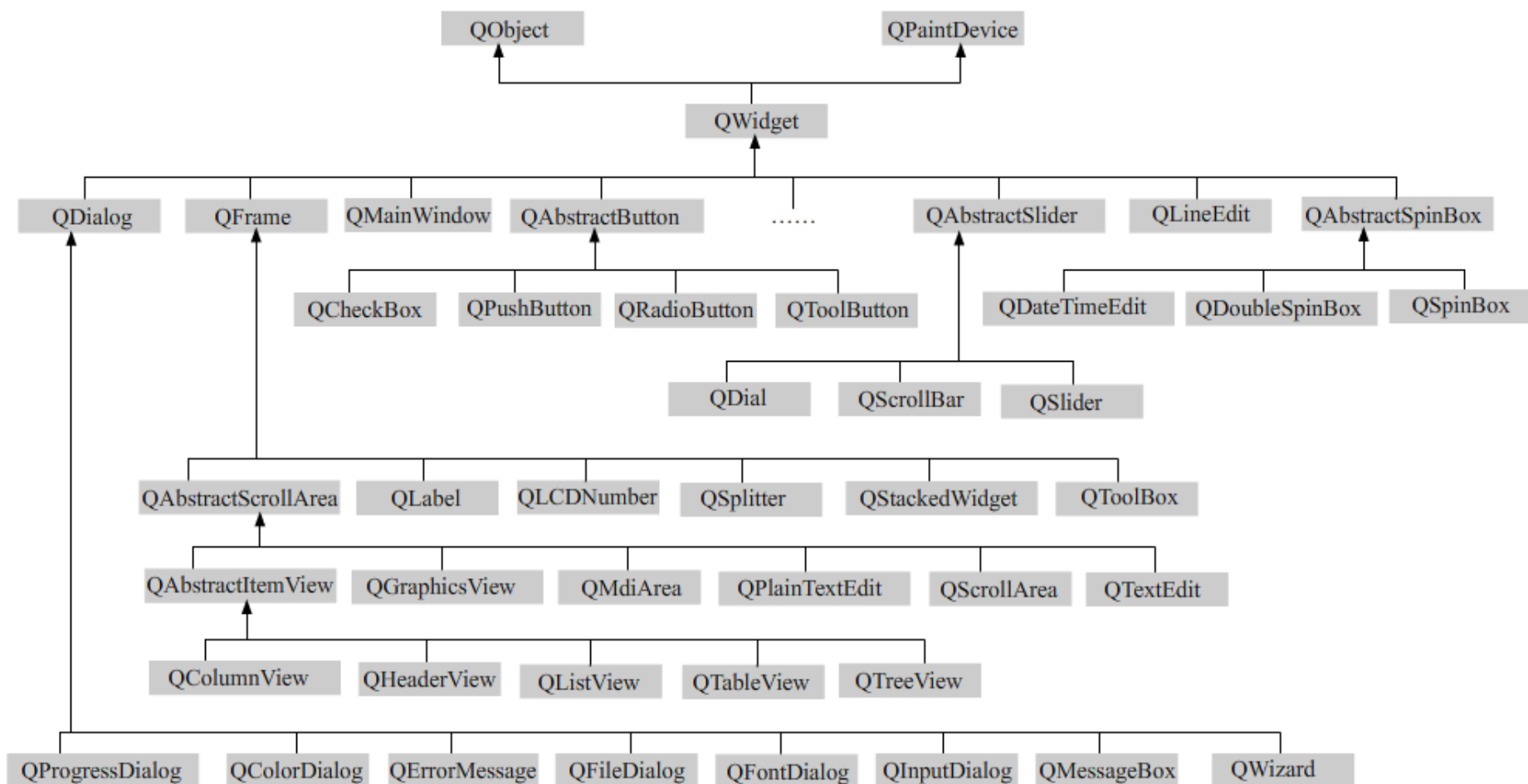
Qt项目文件的基本内容？





窗口部件

窗口部件



主要内容

- 基础窗口部件QWidget
- 对话框QDialog
- 其他窗口部件
- 小结



基础窗口部件QWidget

QWidget类是所有用户界面对象的基类，被称为基础窗口部件。QWidget继承自QObject类和QPaintDevice类，其中QObject类是所有支持Qt对象模型（Qt Object Model）的Qt对象的基类，QPaintDevice类是所有可以绘制的对象的基类。本节内容：

- 窗口、子部件以及窗口类型
- 窗口几何布局
- 程序调试



窗口、子部件以及窗口类型

来看一个代码片段：

```
// 新建QWidget类对象，默认parent参数是0，所以它是个窗口
QWidget *widget = new QWidget();
// 设置窗口标题
widget->setWindowTitle(QObject::tr("我是widget"));
// 新建QLabel对象，默认parent参数是0，所以它是个窗口
QLabel *label = new QLabel();
label->setWindowTitle(QObject::tr("我是label"));
// 设置要显示的信息
label->setText(QObject::tr("label:我是个窗口"));
// 改变部件大小，以便能显示出完整的内容
label->resize(180, 20);
// label2指定了父窗口为widget，所以不是窗口
QLabel *label2 = new QLabel(widget);
label2->setText(QObject::tr("label2:我不是独立窗口，只是widget的子部件"));
label2->resize(250, 20);
// 在屏幕上显示出来
label->show();
widget->show();
```



- 在程序中定义了一个QWidget类对象的指针widget和两个QLabel对象指针label与label2，其中label没有父窗口，而label2在widget中，widget是其父窗口。
- **窗口部件** (Widget) 这里简称部件，是Qt中建立用户界面的主要元素。像主窗口、对话框、标签、还有以后要介绍到的按钮、文本输入框等都是窗口部件。
- 在Qt中，把没有嵌入到其他部件中的部件称为**窗口**，一般的，窗口都有边框和标题栏，就像程序中的widget和label一样。
- QMainWindow和大量的QDialog子类是最一般的窗口类型。窗口就是没有父部件的部件，所以又称为**顶级部件** (top-level widget)。与其相对的是非窗口部件，又称为**子部件** (child widget)。在Qt中大部分部件被用作子部件，它们嵌入在别的窗口中，例如程序中的label2。



窗口类型

前面讲到窗口一般都有边框和标题栏，其实这也不是必需的：

- **QWidget的构造函数有两个参数：**`QWidget * parent = 0`和
`Qt::WindowFlags f = 0;`
- **前面的parent就是指父窗口部件，默认值为0，表明没有父窗口；**
- **而后面的f参数是Qt::WindowFlags类型的，它是一个枚举类型，分为窗口类型（WindowType）和窗口标志（WindowFlags。前者可以定义窗口的类型，比如我们这里f=0，表明使用了Qt::Widget一项，这是QWidget的默认类型，这种类型的部件如果有父窗口，那么它就是子部件，否则就是独立的窗口。**



例如：使用其中的Qt::Dialog和Qt::SplashScreen，更改程序中的新建对象的那两行代码：

```
QWidget *widget = new QWidget(0, Qt::Dialog);
```

```
QLabel *label = new QLabel(0, Qt::SplashScreen);
```

当更改窗口类型后，窗口的样式发生了改变，一个是对话框类型，一个是欢迎窗口类型。

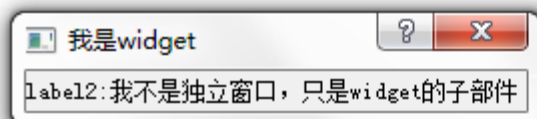
而对于窗口标志，它主要的作用是更改窗口的标题栏和边框，而且它们可以和窗口类型进行位或操作。下面再次更改那两行代码：

```
QWidget *widget = new QWidget(0, Qt::Dialog | Qt::FramelessWindowHint);
```

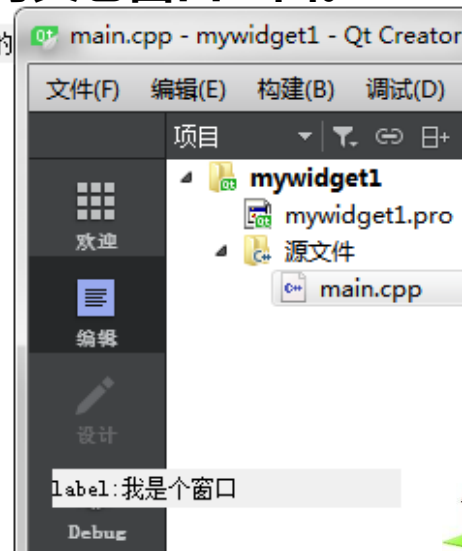
```
QLabel *label = new QLabel(0, Qt::SplashScreen | Qt::WindowStaysOnTopHint);
```

Qt::FramelessWindowHint用来产生一个没有边框的窗口，而Qt::WindowStaysOnTopHint用来使该窗口停留在所有其它窗口上面。

label2:我不是独立窗口，只是widget的

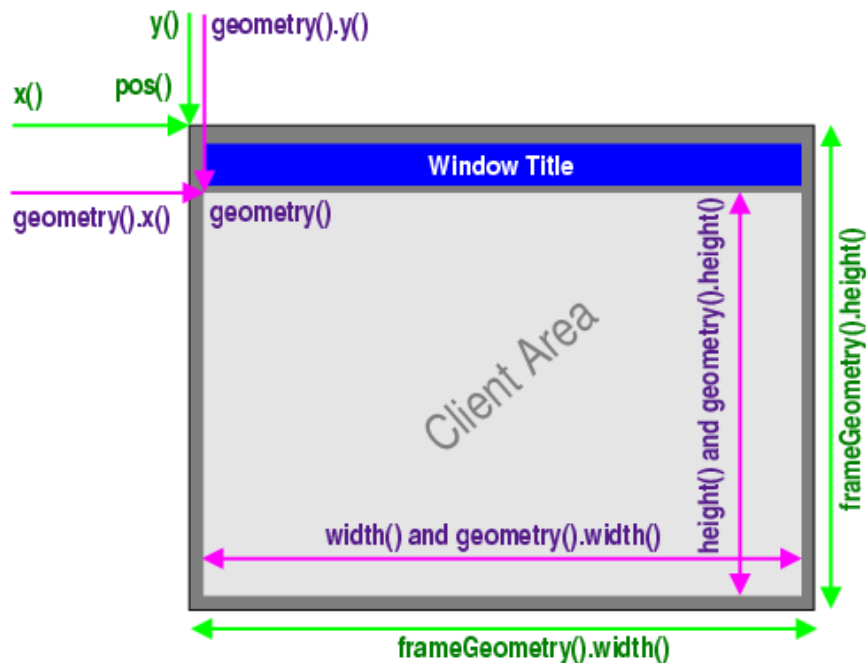


label:我是个窗口



窗口几何布局

对于窗口的大小和位置，根据是否包含边框和标题栏两种情况，要用不同的函数来获取它们的数值。



这里的函数分为两类，一类是包含框架的，一类是不包含框架的：

- 包含框架：x()、y()、frameGeometry()、pos()和move()等函数；
- 不包含框架：geometry()、width()、height()、rect()和size()等函数。



程序调试

下面在讲解窗口几何布局的几个函数的同时，讲解一下程序调试方面的内容。

将主函数内容更改如下：

```
#include <QApplication>
#include <QWidget>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QWidget widget;
    int x = widget.x();
    int y = widget.y();
    QRect geometry = widget.geometry();
    QRect frame = widget.frameGeometry();
    return a.exec();
}
```

`x()`、`y()` 分别返回部件的位置坐标的 `x`、`y` 值，它们的默认值为 0。

而 `geometry()` 和 `frameGeometry()` 函数分别返回没有边框和包含边框的窗口框架矩形的值，其返回值是 `QRect` 类型的，就是一个矩形，它的形式是（位置坐标，大小信息），也就是（`x`，`y`，宽，高）。



下面在`int x = widget.x();` 一行代码的标号前面点击鼠标左键来设置断点。

所谓**断点**，就是程序运行到该行代码时会暂停下来，从而可以查看一些信息，如变量值等。要取消断点，只要在那个断点上再点击一下就可以了。设置好断点后便可以按下F5或者左下角的调试按钮开始调试。这时程序会先进行构建再进入调试模式，这个过程可能需要一些时间。在程序构建时可能会出现警告，那是因为我们定义了变量却没有使用造成的，不用管它。



调试模式

main.cpp - mywidget2 - Qt Creator

文件(F) 编辑(E) 构建(B) 调试(D) Analyze 工具(T) 控件(W) 帮助(H)

项目 mywidget2
mywidget2.pro
源文件
main.cpp

```
1 #include <QApplication>
2 #include <QWidget>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7
8     QWidget widget;
9     int x = widget.x();
10    int y = widget.y();
11    QRect geometry = widget.geometry();
12    QRect frame = widget.frameGeometry();
13
14    return a.exec();
15 }
16
```

名称 值 类型

a	@0x28fe3c	QApplication
argc	1	int
argv	<1 个项>	char **
frame	5618992x2686...	QRect
geometry	-1990283138x...	QRect
widget	@0x28fe24	QWidget
x	1990422896	int
y	2686916	int

1 2 3 4 5 6 7 9

打开文档
main.cpp

Debugger 线程: #1

级别	函数	文件	行号	编号	函数	文件	行号
➔ 1	qMain	main.cpp	9	● 1	qMain(int, char **)	E:\app\src\03...	9
2	WinMain *16	qtmain_win.cpp	113				
3	main						

8

Type to locate (Ctrl+K) 1 问题 5 2 Search Results 3 应用程序输出 4 编译输出 5 Debugger Console 6 概要信息



调试模式的按钮和窗口介绍

- ①继续按钮。程序在断点处停了下来，按下继续按钮后，程序便会像正常运行一样，执行后面的代码，直到遇到下一个断点，或者程序结束。
- ②停止调试按钮。按下该按钮后结束调试。
- ③单步跳过按钮。直接执行本行代码，然后指向下一行代码。
- ④单步进入按钮。进入调用的函数内部。
- ⑤单步跳出按钮。当进入函数内部时，跳出该函数，一般与单步进入配合使用。
- ⑥重新启动调试会话。
- ⑦显示源码对应的汇编指令，并可以单步调试。
- ⑧堆栈视图。这里显示了从程序开始到断点处，所有嵌套调用的函数所在的源文件名和行号。
- ⑨其它视图。这里可以选择多种视图。



单步调试

- 点击一下“单步进入”按钮，或者按下F11，这时，程序会跳转到QWidget类的x()函数的源码处，这里对这个函数不做过多讲解，下面直接按下“单步跳出”按钮回到原来的断点处。然后便开始一直按“单步跳过”按钮，单步执行程序，并查看局部变量和监视器视图中相应变量值的变化情况。等执行到最后一行代码`return a.exec();`时，按下“停止调试”按钮，结束调试。
- 这里要补充说明一下，我们在程序调试过程中可以进入到Qt类的源码中，其实还有一个很简单的方法也可以实现这个功能，就是在编辑器中将鼠标光标定位到一个类名或者函数上，然后按下F2键，或者点击鼠标右键，选择“跟踪光标位置的符号”，这时编辑器就会跳转到其源码处。
- 从变量监视器中可以看到x、y、geometry和frame四个变量初始值都是一个随机未知数。等到调试完成后，x、y的值均为0，这是它们的默认值。而geometry的值为640x480+0+0，frame的值为639x479+0+0。



- 现在对这些值还不是很清楚，不过，为什么x、y的值会是0呢？我们可能会想到，应该是窗口没有显示的原因，那么就更改代码，让窗口先显示出来，再看这些值。在QWidget widget;一行代码后添加一行代码：

```
widget.show();
```

- 现在再次调试程序，这时会发现窗口只显示了一个标题栏，先不管它，继续在Qt Creator中点击“单步跳过”按钮。当我们将程序运行到最后一行代码return a.exec();时，再次按下“单步跳过”按钮后，程序窗口终于显示出来了。这是因为只有程序进入主事件循环后才能接收事件，而show()函数会触发显示事件，所以只有在完成a.exec()函数调用进入消息循环后才能正常显示。这次看到几个变量的值都有了变化，但是这时还是不清楚这些值的含义。
- **注意：**因为使用调试器进行调试要等待一段时间，而且步骤很麻烦，对于初学者来说，如果按错了按钮，还很容易出错。所以，并不推荐初学者使用。



使用qDebug() 函数

一般在程序调试过程中很常用的是qDebug()函数，它可以将调试信息直接输出到控制台，在Qt Creator中是输出到应用程序输出栏。例如：

```
QWidget widget;  
widget.resize(400, 300);           // 设置窗口大小  
widget.move(200, 100);             // 设置窗口位置  
widget.show();  
int x = widget.x();  
qDebug("x: %d", x);                // 输出x的值  
int y = widget.y();  
qDebug("y: %d", y);  
QRect geometry = widget.geometry();  
QRect frame = widget.frameGeometry();  
qDebug() << "geometry: " << geometry << "frame: " << frame;
```



要使用qDebug()函数，就要添加#include <QDebug>头文件。然后这里使用了两种输出方式：

- 方式一：直接将字符串当做参数传给QDebug()函数，例如上面使用这种方法输出x和y的值。
- 方式二：使用输出流的方式一次输出多个值，它们的类型可以不同，例如程序中输出geometry和frame的值。
- 需要说明的是，如果只使用第一种方法，那么是不需要添加<QDebug>头文件的，如果使用第二种方法就必须添加这个头文件。因为第一种方法很麻烦，所以经常使用的是第二种方法。

```
应用程序输出
mywidget2
Starting E:\app\src\03\3-2\build-mywidget2-Desktop_Qt_5_6_1_MinGW_32bit-Debug\debug\mywidget2.exe...
x: 200
y: 100
geometry: QRect(208,130 400x300) frame: QRect(200,100 416x338)
```



- 其实使用qDebug()函数的第二种方法时还可以让输出自动换行，下面来看一下其他几个函数的用法。在return a.exec();一行代码前添加如下代码：

```
qDebug() << "pos:" << widget.pos() << endl << "rect:" << widget.rect()
        << endl << "size:" << widget.size() << endl << "width:"
        << widget.width() << endl << "height:" << widget.height();
```

- 这里的“endl”就是起换行作用的。
- 根据程序的输出结果，可以很明了的看到这些函数的作用。
- 其中pos()函数返回窗口的位置，是一个坐标值，上面的x()、y()函数返回的就是它的x、y坐标值；
- rect()函数返回不包含边框的窗口内部矩形，在窗口内部，左上角是(0,0)点；
- size()函数返回不包含边框的窗口大小信息；
- width()和height()函数分别返回窗口内部的宽和高。
- 从数据可以看到，前面使用的调整窗口大小的resize()函数是设置的不包含边框的窗口大小。



3.2 对话框QDialog

本节先从对话框的介绍讲起，然后讲述两种不同类型的对话框，再讲解一个有多个窗口组成并且窗口间可以相互切换的程序，最后介绍一下Qt提供的几个标准对话框。

- 模态和非模态对话框
- 多窗口切换
- 标准对话框



模态和非模态对话框

- **模态**对话框就是在我们没有关闭它之前，不能再与同一个应用程序的其他窗口进行交互，比如新建项目时弹出的对话框。要想使一个对话框成为模态对话框，只需要调用它的`exec()`函数：

```
QDialog dialog(this);  
dialog.exec();
```

- 而对于**非模态**对话框，既可以与它交互，也可以与同一程序中的其他窗口交互，例如Microsoft Word中的查找替换对话框。要使一个对话框成为非模态对话框，我们就可以使用`new`操作来创建，然后使用`show()`函数来显示。

```
QDialog *dialog = new QDialog(this);  
dialog->show();
```



使用show()函数也可以建立模态对话框

建立模态对话框，只需在其前面使用setModal()函数即可。例如：

```
QDialog *dialog = new QDialog(this);  
dialog->setModal(true);  
dialog->show();
```

现在运行程序，可以看到生成的对话框是模态的。但是，它与用exec()函数时的效果是不一样的。这是因为调用完show()函数后会立即将控制权交给调用者，那么程序可以继续往下执行。而调用exec()函数却不是这样，它只有当对话框被关闭时才会返回。

与setModal()函数相似的还有一个setWindowModality()函数，它有一个参数来设置模态对话框要阻塞的窗口类型，可以是：

- Qt::NonModal (不阻塞任何窗口，就是非模态)，
- Qt::WindowModal (阻塞它的父窗口和所有祖先窗口以及它们的子窗口)，
- Qt::ApplicationModal (阻塞整个应用程序的所有窗口)。

而setModal()函数默认设置的是Qt::ApplicationModal。



多窗口切换

本节会涉及如下内容：

- 开始认识信号和槽
- 信号和槽的关联方式
- 从登陆对话框显示主界面的方法



开始认识信号和槽

在Qt中使用信号和槽机制来完成对象之间的协同操作。

简单来说，信号和槽都是函数，比如按下窗口上的一个按钮后想要弹出一个对话框，那么就可以将这个按钮的单击信号和我们定义的槽关联起来，在这个槽中可以创建一个对话框，并且显示它。这样，当单击这个按钮时就会发射信号，进而执行我们的槽来显示一个对话框。



关联方式一：使用connect()关联

- mywidget.h文件写上槽的声明：

public slots:

```
void showChildDialog();
```

- 在mywidget.cpp文件中将槽的实现：

```
void MyWidget::showChildDialog()
```

```
{
```

```
    QDialog *dialog = new QDialog(this);
```

```
    dialog->show();
```

```
}
```

- 在mywidget.cpp文件的MyWidget类的构造函数中使用connect()关联按钮单击信号和自定义的槽如下：

```
connect(ui->showChildButton, &QPushButton::clicked,  
        this, &MyWidget::showChildDialog);
```



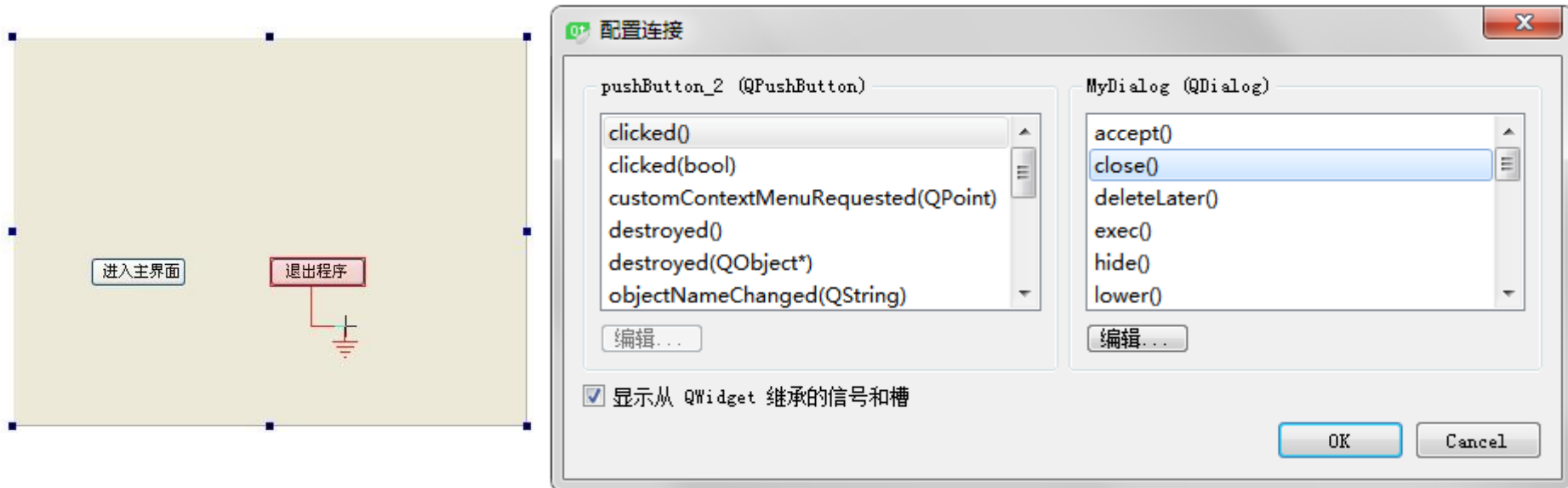
说明：

- 这里自定义了一个槽，槽一般使用slots关键字进行修饰（Qt 4中必须使用，Qt 5使用新connect语法时可以不用，为了与一般函数进行区别，建议使用），这里使用了public slots，表明这个槽可以在类外被调用。
- clicked()信号在QPushButton类中进行了定义，而connect()是QObject类中的函数，因为我们的类继承自QObject，所以可以直接使用它。
- connect()函数中的四个参数分别是：发送信号的对象、发送的信号、接收信号的对象和要执行的槽。



关联方式二：在设计模式关联

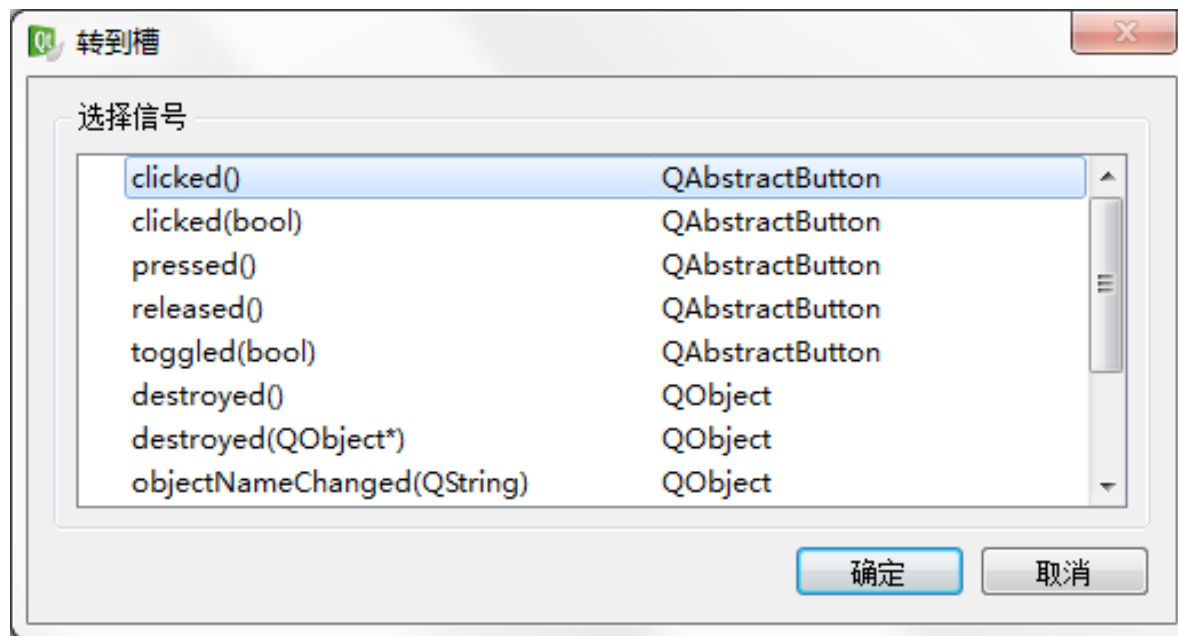
- 首先添加自定义对话框类MyDialog。在设计模式中向窗口上添加两个Push Button，并且分别更改其显示文本为“进入主界面”和“退出程序”。
- 点击设计器上方的“编辑信号/槽”图标，或者按下快捷键F4，这时便进入了部件的信号和槽的编辑模式。在“退出程序”按钮上按住鼠标左键，然后拖动到窗口界面上，这时松开鼠标左键。
- 在弹出的配置连接对话框中，选中下面的“显示从QWidget继承的信号和槽”选项，然后在左边的QPushButton栏中选择信号clicked()，在右边的QDialog栏中选择对应的槽close()，完成后按下“确定”。



关联方式三：自动关联

在“进入主界面”按钮上右击，在弹出的菜单上选择“转到槽”，然后在弹出的对话框中选择clicked()信号，并按“确定”。这时便会进入代码编辑模式，并且定位到自动生成的on_pushButton_clicked()槽中。在其中添加代码：

```
void MyDialog::on_pushButton_clicked()
{
    accept();
}
```



- 自动关联就是将关联函数整合到槽命名中。
- 例如on_pushButton_clicked()就是由字符“on”和发射信号的部件对象名，还有信号名组成。这样就可以去掉那个connect()关联函数了。每当pushButton被按下，就会发射clicked()信号，然后就会执行on_pushButton_clicked()槽。
- 这里accept()函数是QDialog类中的一个槽，对于一个使用exec()函数实现的模态对话框，执行了这个槽，就会隐藏这个模态对话框，并返回QDialog::Accepted值，我们就是要使用这个值来判断是哪个按钮被按下了。与其对应的还有一个reject()槽，它可以返回一个QDialog::Rejected值。其实，前面的“退出程序”按钮也可以关联这个槽。



使用自定义对话框登陆主界面

在main()函数中:

```
QApplication a(argc, argv);  
MyWidget w;  
MyDialog dialog;           // 新建MyDialog类对象  
if(dialog.exec() == QDialog::Accepted){ // 判断dialog执行结果  
    w.show();               // 如果是按下了“进入主界面”按钮，则显示主界面  
    return a.exec();        // 程序正常运行  
}  
else return 0;
```

在主函数中建立了MyDialog对象，然后判断其exec()函数的返回值，如果是按下了“进入主界面”按钮，那么返回值应该是QDialog::Accepted，就显示主界面，并且正常执行程序。如果不是，则直接退出程序。



标准对话框

Qt提供了一些常用的对话框类型，它们全部继承自QDialog类，并增加了自己的特色功能，比如获取颜色、显示特定信息等。

- 颜色对话框
- 文件对话框
- 字体对话框
- 输入对话框
- 消息对话框
- 进度对话框
- 错误信息对话框
- 向导对话框



颜色对话框

例如：

```
QColor color = QColorDialog::getColor(Qt::red, this, tr("颜色对话框"));  
QDebug() << "color: " << color;
```

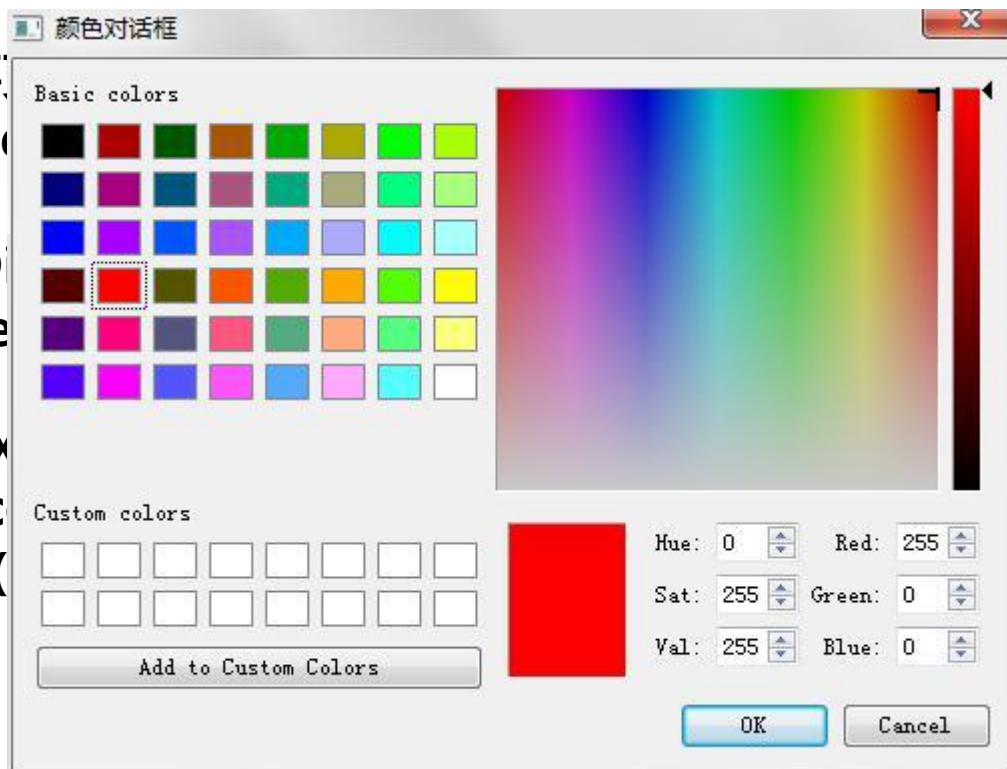
这里使用了QColorDialog的静态函数getColor()来获取颜色，它的三个参数的作用分别是：设置初始颜色、父窗口和对话框标题。这里的Qt::red，是Qt预定义的颜色对象。

➤ 如果想要：

```
void MyWidget::getColor()  
{
```

```
    QColorDialog dialog;  
    dialog.setCurrentColor(Qt::red);  
    dialog.setWindowTitle(tr("颜色对话框"));  
    QColor color = dialog.getColor();  
    qDebug() << "color: " << color;
```

```
}
```



对话框设置：

颜色对象

dialog.setCurrentColor(Qt::red); // 显示alpha选择

颜色对话框

当前颜色

颜色信息



自学内容：

学生自学其他标准对话框的使用。



其他窗口部件

Qt提供了一些常用的窗口部件：

➤ QFrame类族

➤ 按钮部件

➤ 行编辑器

➤ 数值设定框

➤ 滑块部件



QFrame类族

➤ QFrame
部件Q
QTool

➤ 带边框
的主要
(Sha

0				1				2				3				lineWidth()
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	midLineWidth()
																Box, Plain
																Box, Raised
																Box, Sunken
																Panel, Plain
																Panel, Raised
																Panel, Sunken
																WinPanel, Plain
																WinPanel, Raised
																WinPanel, Sunken
																HLine, Plain
																HLine, Raised
																HLine, Sunken
																VLine, Plain
																VLine, Raised
																VLine, Sunken
																StyledPanel, Plain
																StyledPanel, Raised
																StyledPanel, Sunken

为常用的标签
edWidget、

果。QFrame类
白边框形状



例：QLabel

- 除了最常用的显示文本外，还可以显示图片：

```
ui->label->setPixmap(QPixmap("F:/logo.png"));
```

- 还可以显示gif动态图片：

```
QMovie *movie = new QMovie("F:/donghua.gif");  
ui->label->setMovie(movie);      // 在标签中添加动画  
movie->start();
```



按钮部件

QAbstractButton类是按钮部件的抽象基类，提供了按钮的通用功能。它的子类包括：

- **复选框QCheckBox**
- **标准按钮QPushButton**
- **单选框按钮QRadioButton**
- **工具按钮QToolButton。**



例：QPushButton

在下面代码里为三个按钮改变了显示文本，这里在一个字母前加上“&”符号，那么就可以将这个按钮的加速键设置为Alt加上这个字母。如果我们要在文本中显示“&”符号本身，那么可以使用“&&”。我们也可以使用setIcon()函数来给按钮添加图标，这里图片文件使用了相对路径（当然这个也可以在设计模式通过更改icon属性来实现）。对于pushBtn3，我们为其添加了下拉菜单，当然，现在这个菜单什么功能也没实现。

```
ui->pushBtn1->setText(tr("&nihao"));           // 这样便指定了Alt+N为加速键
ui->pushBtn2->setText(tr("帮助(&H)"));
ui->pushBtn2->setIcon(QIcon("../image/help.png"));
ui->pushBtn3->setText(tr("z&oom"));
QMenu *menu = new QMenu(this);
menu->addAction(QIcon("../image/zoom-in.png"), tr("放大"));
ui->pushBtn3->setMenu(menu);
```



行编辑器

行编辑器QLineEdit部件是一个单行的文本编辑器，它允许用户输入和编辑单行的纯文本内容，而且提供了一系列有用的功能，包括撤销与恢复、剪切和拖放等操作。例如：


- 显示模式
- 输入掩码
- 输入验证
- 自动补全



例：自动补全功能

在QLineEdit中也提供了强大的自动补全功能，这是利用QCompleter类实现的：

自动完成：



q
Qt
Qt Creator

```
QStringList wordList;
```

```
wordList << "Qt" << "Qt Creator";
```

```
// 新建自动完成器
```

```
QCompleter *completer = new QCompleter(wordList, this);
```

```
// 设置大小写不敏感
```

```
completer->setCaseSensitivity(Qt::CaseInsensitive);
```

```
ui->lineEdit4->setCompleter(completer);
```



数值设定框

QAbstractSpinBox类是一个抽象基类，它提供了一个数值设定框和一个行编辑器来显示设定值。它有三个子类：

- **QDateTimeEdit（日期时间设定）**
- **QSpinBox（整数设定）**
- **QDoubleSpinBox（浮点数的设定）**



例： QDateTimeEdit

下面的代码设置了dateTimeEdit中的日期和时间。这里简单说明一下：y表示年；M表示月；d表示日，而ddd表示星期；H表示小时，使用24小时制显示，而h也表示小时，如果最后有AM或者PM的，则是12小时制显示，否则使用24小时制；m表示分；s表示秒；还有一个z可以用来表示毫秒。

// 设置时间为现在的系统时间

```
ui->dateTimeEdit-
```

```
>setDateTime(QDateTime::currentDateTime());
```

// 设置时间的显示格式

```
ui->dateTimeEdit->setDisplayFormat(
```

```
tr("yyyy年MM月dd日ddd HH时mm分ss秒"));
```



滑块部件

QAbstractSlider类提供了一个区间内的整数值，它有一个滑块，可以定位到一个整数区间的任意值。这个类是一个抽象基类，它有三个子类**QScrollBar**，**QSlider**和**QDial**。其中：

- **滚动条QScrollBar更多的是用在QScrollArea类中来实现滚动区域；**
- **而QSlider是我们最常见的音量控制或多媒体播放进度等滑块；**
- **QDial是一个刻度表盘。**



小结

本章讲述了众多常用的窗口部件的使用方法，其中还涉及了程序调试和信号和槽等知识。学习本章，一定要多实践，多运用各个部件进行编程，才能真正掌握！



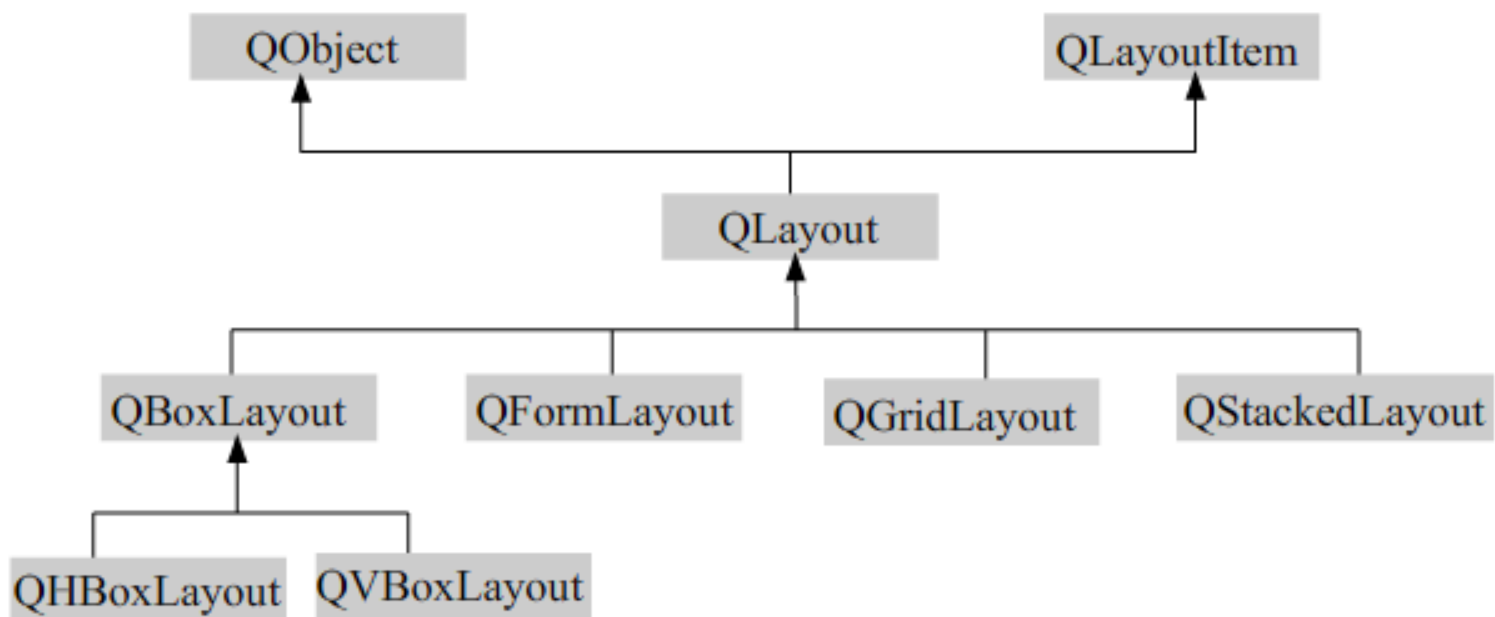
- 如何设置窗口类型，获取窗口信息的函数有哪些？
- 如何使用调试模式，`qDebug()`函数的用法？
- 什么是信号和槽，有哪些关联方式？
- Qt中标准对话框、常用部件有哪些，都有哪些特色功能？





布局管理

对于一个完善的软件，布局管理却是必不可少的。无论是想要界面中部件有一个很整齐的排列，还是想要界面能适应窗口的大小变化，都要进行布局管理。Qt中主要提供了QLayout类及其子类来作为布局管理器，它们可以实现常用的布局管理功能



主要内容

- 布局管理系统
- 设置伙伴
- 设置Tab键顺序
- 小结



4.1 布局管理系统

Qt的布局管理系统提供了简单而强大的机制来自动排列一个窗口中的部件，确保它们有效的使用空间。Qt包含了一组布局管理类来描述怎样在应用程序的用户界面中对部件进行布局，比如QLayout的几个子类，我们这里将它们称作布局管理器。所有的QWidget类的子类的实例（对象）都可以使用布局管理器来管理位于它们之中的子部件，QWidget::setLayout()函数可以在一个部件上应用布局管理器。一旦一个部件上设置了布局管理器，那么它会完成以下几种任务：

- 定位子部件；
- 感知窗口默认大小；
- 感知窗口最小大小；
- 改变大小处理；
- 当内容改变时自动更新：
 - 字体大小，文本或子部件的其他内容随之改变；
 - 隐藏或显示子部件；
 - 移除一个子部件。



布局管理器

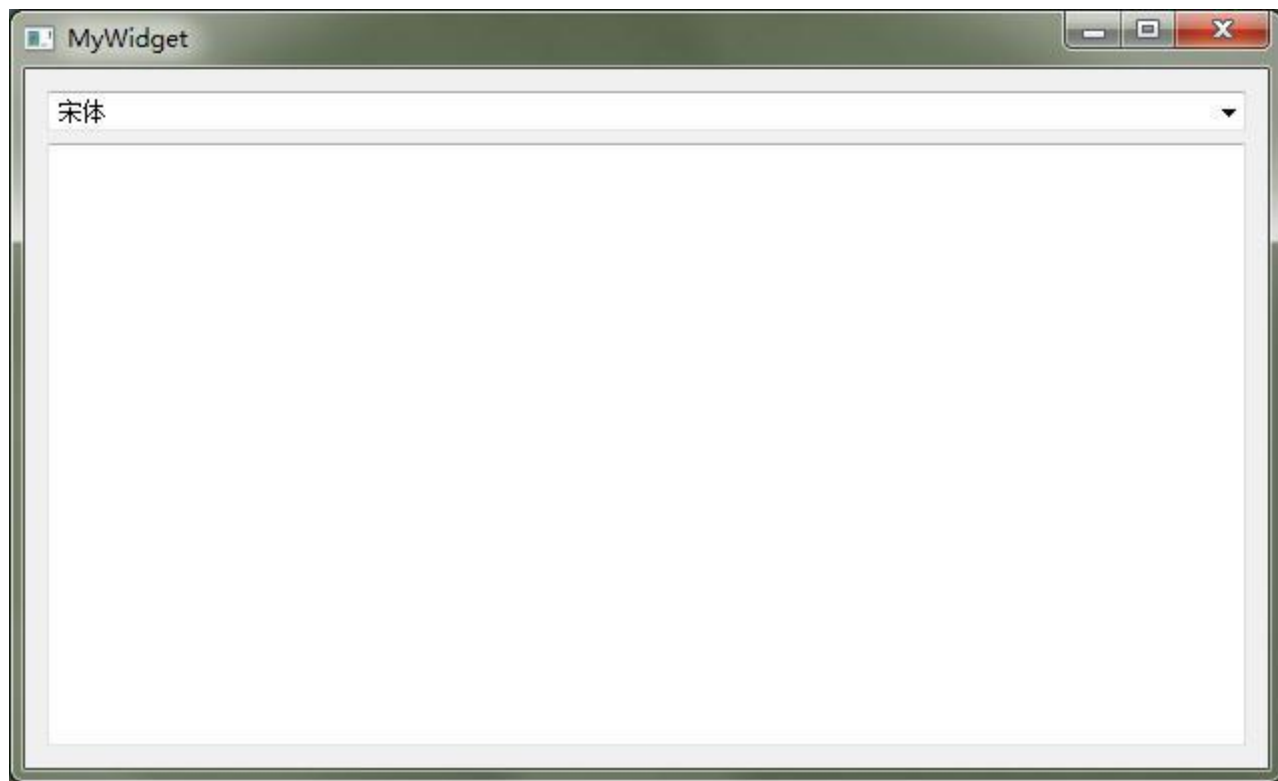
QLayout类是布局管理器的基类，它是一个抽象基类。该类继承自QObject和QLayoutItem类，而QLayoutItem类提供了一个供QLayout操作的抽象项目。QLayout和QLayoutItem都是在设计自己的布局管理器时才使用的，一般只需要使用QLayout的几个子类就可以了，它们分别是：

- **QBoxLayout（基本布局管理器）**
- **QGridLayout（栅格布局管理器）**
- **QFormLayout（窗体布局管理器）**
- **QStackedLayout（栈布局管理器）**



部件随窗口变化大小

在设计模式中向界面上拖入一个字体选择框Font Combo Box和一个文本编辑器Text Edit部件。然后点击主界面，并按下Ctrl+L快捷键，这样便设置了垂直布局管理器，可以看到两个部件已经填满了整个界面。这时运行程序，然后拉伸窗口，两个部件会随着窗口的大小变化而变化，这就是布局管理器在起作用。



基本布局管理器（QBoxLayout）

基本布局管理器QBoxLayout类可以使子部件在水平方向或者垂直方向排成一行，它将所有的空间分成一行盒子，然后将每个部件放入一个盒子中。它有两个子类QHBoxLayout水平布局管理器和QVBoxLayout垂直布局管理器。布局管理器的几个属性如下表所示。

属性	说明
layoutName	现在所使用的布局管理器的名称
layoutLeftMargin	设置布局管理器到界面左边界的距离
layoutTopMargin	设置布局管理器到界面上边界的距离
layoutRightMargin	设置布局管理器到界面右边界的距离
layoutBottomMargin	设置布局管理器到界面下边界的距离
layoutSpacing	布局管理器中各个子部件间的距离
layoutStretch	伸缩因子
layoutSizeConstraint	设置大小约束条件



使用代码实现水平布局

```
QHBoxLayout *layout = new QHBoxLayout;           // 新建水平布局管理器

layout->addWidget(ui->fontComboBox);              // 向布局管理器中添加部件

layout->addWidget(ui->textEdit);

layout->setSpacing(50);                            // 设置部件间的间隔

layout->setContentsMargins(0, 0, 50, 100); // 设置布局管理器到边界的距离,
                                           // 四个参数顺序是左, 上, 右, 下

setLayout(layout);
```



栅格布局管理器 (QGridLayout)

栅格布局管理器QGridLayout类使得部件在网格中进行布局，它将所有的空间分隔成一些行和列。行和列的交叉处就形成了单元格，然后将部件放入一个确定的单元。

QGridLayout

// 添加部件

layout->add

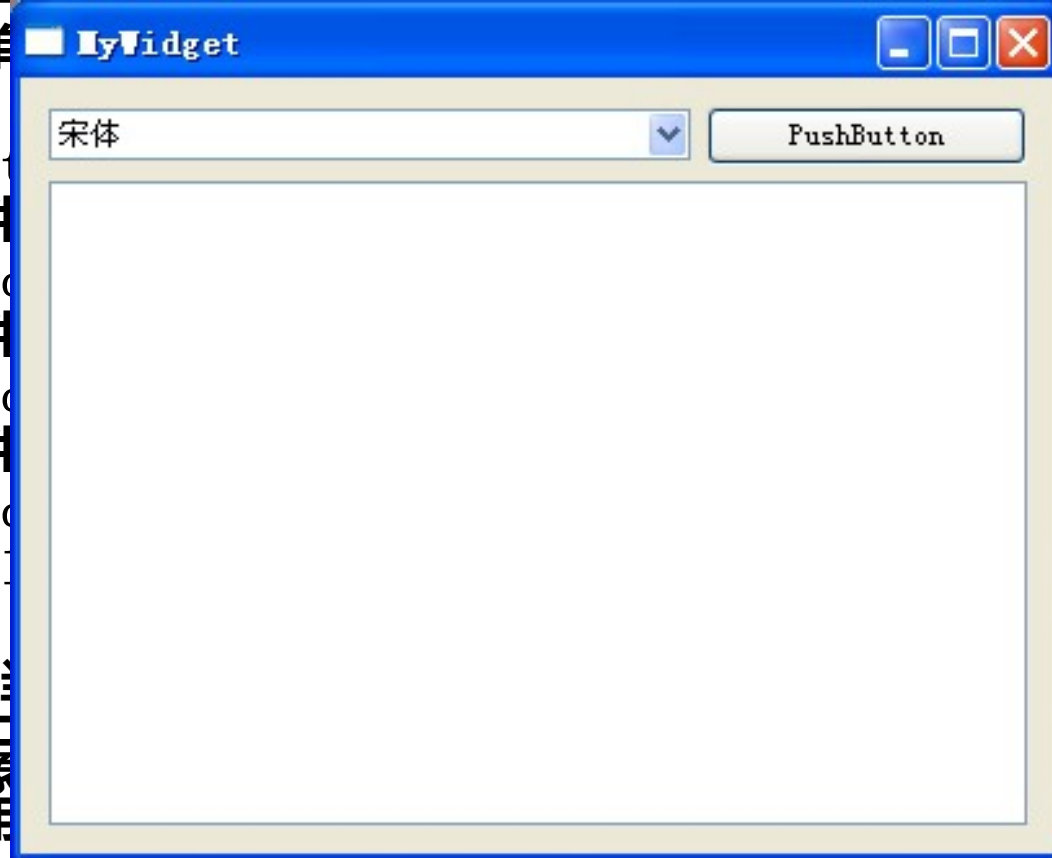
// 添加部件

layout->add

// 添加部件

layout->add

setLayout()



说明：
部件上的父对象时并不用

理器再放到一个窗
动重新定义自己
理器和其中的部件



窗体布局管理器 (QFormLayout)

窗体布局管理器QFormLayout类用来管理表单的输入部件

和与
两列
编辑器

部件分为
比如行编

在设
界面
“流
行”
自动
等，

拖入到
键，选择
表单布局
样下面便
段名称”

The screenshot shows a dialog box titled "添加表单布局行" (Add Form Layout Row) with a blue header bar and a red close button. The dialog is set against a light beige background with a dotted grid. A small label "宋体" (Songti) is visible on the left. The dialog contains the following fields and controls:

- 标签文字 (L): 姓名 (&N):
- 标签名称 (N): nLabel
- 字段类型 (T): QLineEdit (with a dropdown arrow)
- 字段名称 (F): nLineEdit
- 伙伴 (E): ☒
- 行 (R): 1 (with up/down arrows)

At the bottom of the dialog are two buttons: "确定" (OK) and "取消" (Cancel).



这里填写的标签文字中 (&N) , 要注意括号必须是英语半角的, 表明它的快捷键是Alt+N, 设置**伙伴关系**表示当按下Alt+N时, 光标会自动跳转到标签后面对应的行编辑器中。按下确定键, 便会在布局管理器中添加一个标签和一个行编辑器。按照这种方法, 再添加三行: 性别 (&S), 使用QComboBox; 年龄 (&A), 使用QSpinBox



The image shows a Qt widget window titled "MyWidget". It contains a form with the following elements:

- 姓名 (&N): A text input field.
- 性别 (&S): A QComboBox widget.
- 年龄 (&A): A QSpinBox widget with the value 0.
- 邮箱 (&M): A text input field.
- 宋体: A font dropdown menu.
- PushButton: A button.
- A large text area at the bottom.

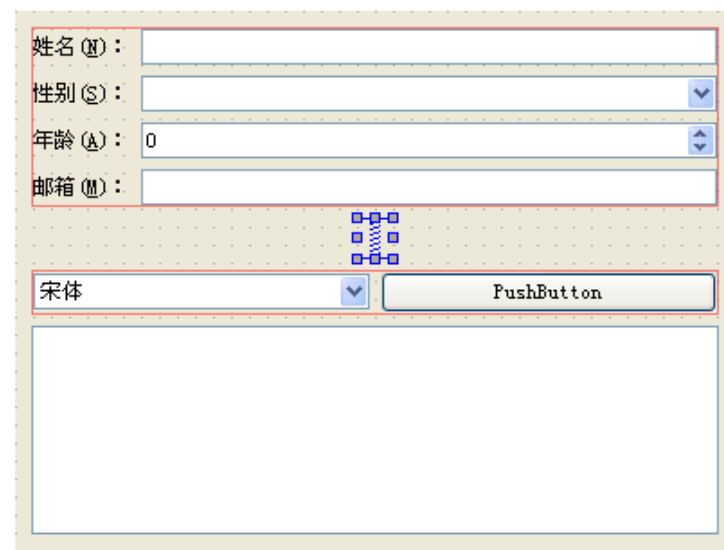
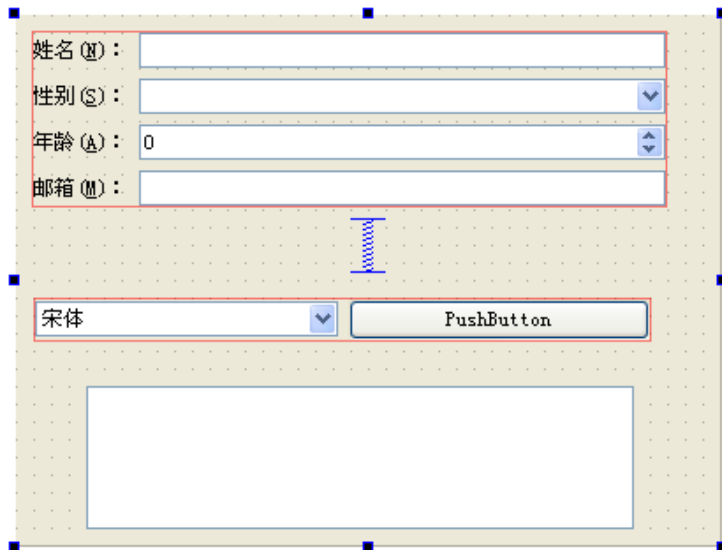
可以按下快捷键Alt+N, 这样光标就可以定位到“姓名”标签后的行编辑器中。



综合使用布局管理器

将前面的界面再进行设计：按下Ctrl键同时选中界面上的字体选择框fontComboBox和按钮pushButton，然后按下Ctrl+H快捷键将它们放入一个水平布局管理器中。然后再从部件栏中拖入一个Vertical Spacer垂直分隔符，它们是用来在部件间产生间隔的，将它放在表单布局管理器与水平布局管理器之间。

最后点击主界面，然后按下Ctrl+L快捷键，让整个界面处于一个垂直布局管理器中。这时我们可以在右上角的对象列表中选择分隔符Spacer，然后在属性栏中设置它的高度为100。



设置部件大小

凡是继承自QWidget的类都有这两个属性：大小提示（sizeHint）和最小大小提示（minimumSizeHint）。

- sizeHint属性保存了部件的建议大小，对于不同的部件，默认拥有不同的sizeHint，程序中使用sizeHint()函数来获取sizeHint的值；
- minimumSizeHint保存了一个建议的最小大小。程序中使用minimumSizeHint()函数来获取minimumSizeHint的值。

需要说明的是，如果使用minimumSize()函数设置了部件的最小大小，那么最小大小提示将会被忽略。



大小策略（sizePolicy）属性

sizePolicy属性的所有取值如下表所示：

常量	描述
QSizePolicy::Fixed	只能使用 sizeHint() 提供的值，无法伸缩
QSizePolicy::Minimum	sizeHint() 提供的大小是最小的，部件可以被拉伸
QSizePolicy::Maximum	sizeHint() 提供的是最大大小，部件可以被压缩
QSizePolicy::Preferred	sizeHint() 提供的大小是最佳大小，部件可以被压缩或拉伸
QSizePolicy::Expanding	sizeHint() 提供的是合适的大小，部件可以被压缩，不过它更倾向于被拉伸来获得更多的空间
QSizePolicy::MinimumExpanding	sizeHint() 提供的大小是最小的，部件倾向于被拉伸来获取更多的空间
QSizePolicy::Ignored	sizeHint() 的值被忽略，部件将尽可能的被拉伸来获取更多的空间



伸缩因子 (stretch factor)







它是用来设置部件间的比例的。

例如，界面上的字体选择框和一个按钮处于一个水平布局管理器中，现在想让它们的宽度比例为2:1，那么就可以点击对象栏中的horizontalLayout水平布局管理器对象，然后在它的属性栏中将layoutStretch属性设置为“2, 1”，这样这个水平布局管理器中的两个部件的宽度就是2:1的比例了。

如果要在代码中进行设置，可以在使用布局管理器的addWidget()函数添加部件的同时，在第二个参数中指定伸缩因子。



QWidget部件大小相关属性

 geometry	[(0, 0), 400 x 300]
X	0
Y	0
宽度	400
高度	300
 sizePolicy	[Preferred, Preferred, 0, 0]
水平策略	Preferred
垂直策略	Preferred
水平伸展	0
垂直伸展	0
 minimumSize	0 x 0
宽度	0
高度	0
 maximumSize	16777215 x 16777215
宽度	16777215
高度	16777215
 sizeIncrement	0 x 0
宽度	0
高度	0
 baseSize	0 x 0
宽度	0
高度	0

- 高度与宽度属性，是现在界面的大小；
- sizePolicy属性可以设置大小策略以及伸缩因子；
- minimumSize属性用来设置最小大小；
- maximumSize属性设置最大大小；
- sizeIncrement属性和baseSize属性是设置窗口改变大小的，一般不用设置。



可扩展窗口

对于一个窗口，可能有很多选项是扩充的，只有在必要的时候才显示出来，这就是所谓的可扩展窗口。对于布局管理器，当子部件重新显示时，在必要的时候才显示多余的内容，就叫做可扩展窗口。

首先在界面窗口，然后转到它的

然后转到它的

```
void MyWidget  
{  
    ui->textL  
    if(checke  
    else ui->  
}
```

“显示可扩展窗

显示隐窗口按钮

管理器的显示和隐藏
展窗口”));
);



使用按钮的按下与否两种状态来设置文本编辑器是否显示，并且相应的更改按钮的文本。为了让文本编辑器在一开始是隐藏的，还要在MyWidget类的构造函数中添加一行代码：

`ui->textEdit->hide();` // 让文本编辑器隐藏，也可以使用`setVisible(false)`函数

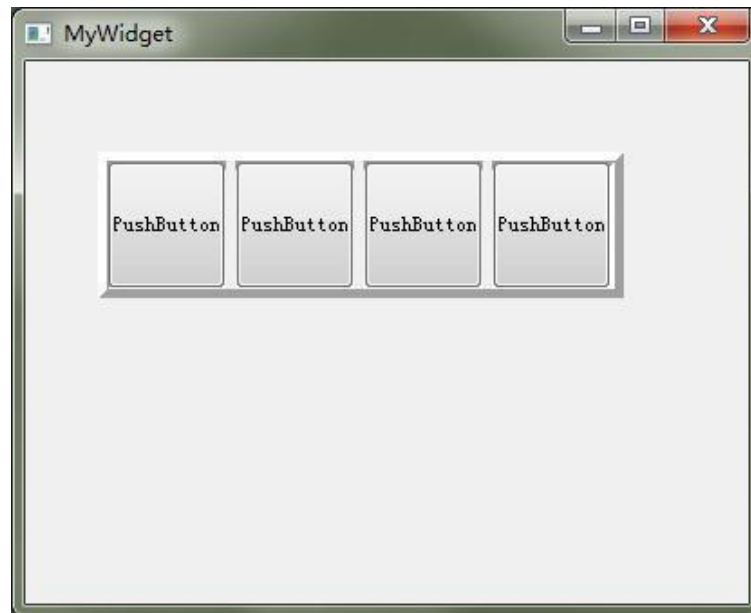


分裂器(QSplitter)

分裂器QSplitter类提供了一个分裂器部件。其实它和QBoxLayout很相似，可以完成布局管理器的功能，但是包含在它里面的部件，默认是可以随着分裂器的大小变化而进行相应大小变化的。

比如一个按钮，放在布局管理器中，它的垂直方向默认是不会被拉伸的，但是放到分裂器中就可以被拉伸。

还有一个不同就是，布局管理器是继承自QObject类的，而分裂器却是继承自QFrame类，QFrame类又是继承自QWidget类，也就是说，分裂器拥有QWidget类的特性，它是可见的，而且可以像QFrame一样设置边框。



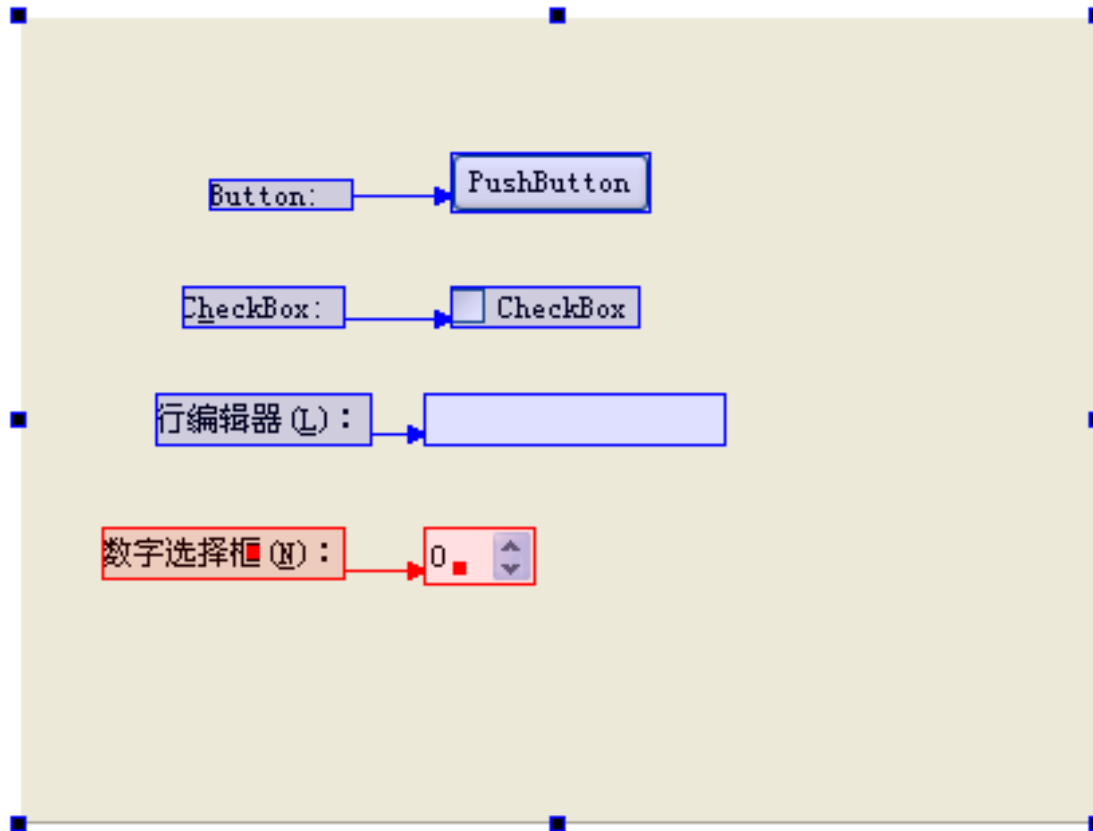
4.2 设置伙伴

- **伙伴 (buddy) 是在QLabel类中提出的一个概念。因为一个标签经常用作一个交互式部件的说明，就像表单布局管理器时看到的那样，一个lineEdit部件前面有一个标签说明这个lineEdit的作用。为了方便定位，QLabel提供了一个有用的机制，那就是提供了助记符来设置键盘焦点到对应的部件上，而这个部件就叫做这个QLabel的伙伴。**
- **其中助记符就是我们所说的加速键。在使用英文标签时，在字符串的一个字母前面添加 “&” 符号，那么就可以指定这个标签的加速键是Alt加上这个字母，而对于中文，需要在小括号中指定加速键字母。**



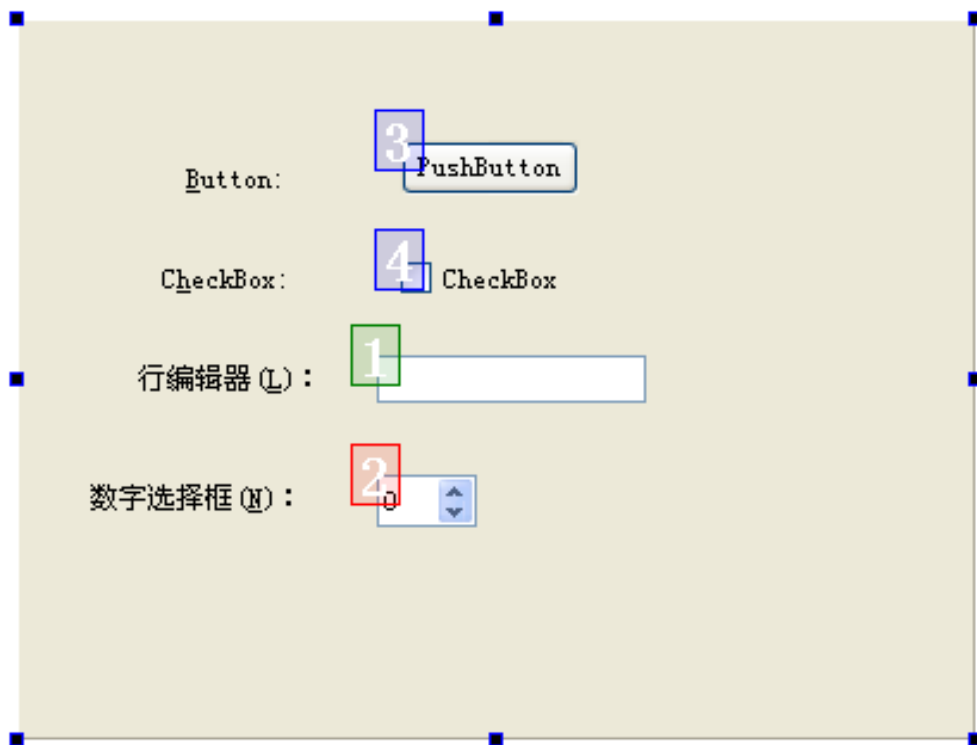
在设计器中设置伙伴

按下设计器顶部栏中的编辑伙伴图标，进入伙伴设计模式，分别将各个标签与它们后面的部件连起来。



4.3 设置Tab键顺序

对于一个应用程序，我们有时总希望使用Tab键来将焦点从一个部件移动到下一个部件。在设计模式，设计器提供了Tab键的设置功能。在上面程序的设计模式中，按下上边栏的编辑Tab顺序按钮进入编辑Tab键顺序模式，这时已经显示出了各个部件的Tab键顺序，只需要用鼠标点击这些数字，就可以更改它们。



使用代码进行设置

当程序启动时，焦点会在Tab键顺序为1的部件上。这里进行的设置等价于在构造函数中使用如下代码：

```
//lineEdit在spinBox前面
```

```
setTabOrder(ui->lineEdit, ui->spinBox);
```

```
//spinBox在pushButton前面
```

```
setTabOrder(ui->spinBox, ui->pushButton);
```

```
//pushButton在checkBox前面
```

```
setTabOrder(ui->pushButton, ui->checkBox);
```



小结

本章中讲述了一些关于界面布局的知识，学生要掌握几个布局管理器的使用方法，学会综合应用它们。



- Qt中布局管理器有哪几种？
- 布局管理器有什么作用？
- 如何设置可扩展窗口？
- 什么是伙伴关系？

