

Qt6 C++开发指南学习笔记

第2章 GUI程序设计基础

2.1 GUI程序结构与运行机制

2.1.1 创建GUI窗口工程

2.1.2 项目管理文件

2.1.3 主函数文件

2.1.4 窗体相关文件

2.1.4.1 widget.h文件

2.1.4.2 widget.cpp文件

2.1.4.3 ui_widget.h文件

2.1.4.4 ui_widget.cpp文件

2.1.5 创建第一个Qt GUI工程

2.1.6 UI图像设计界面Qt Designer简介

2.1.7 在UI图像设计界面创建简单的信号与槽链接

第3章Qt框架功能概述

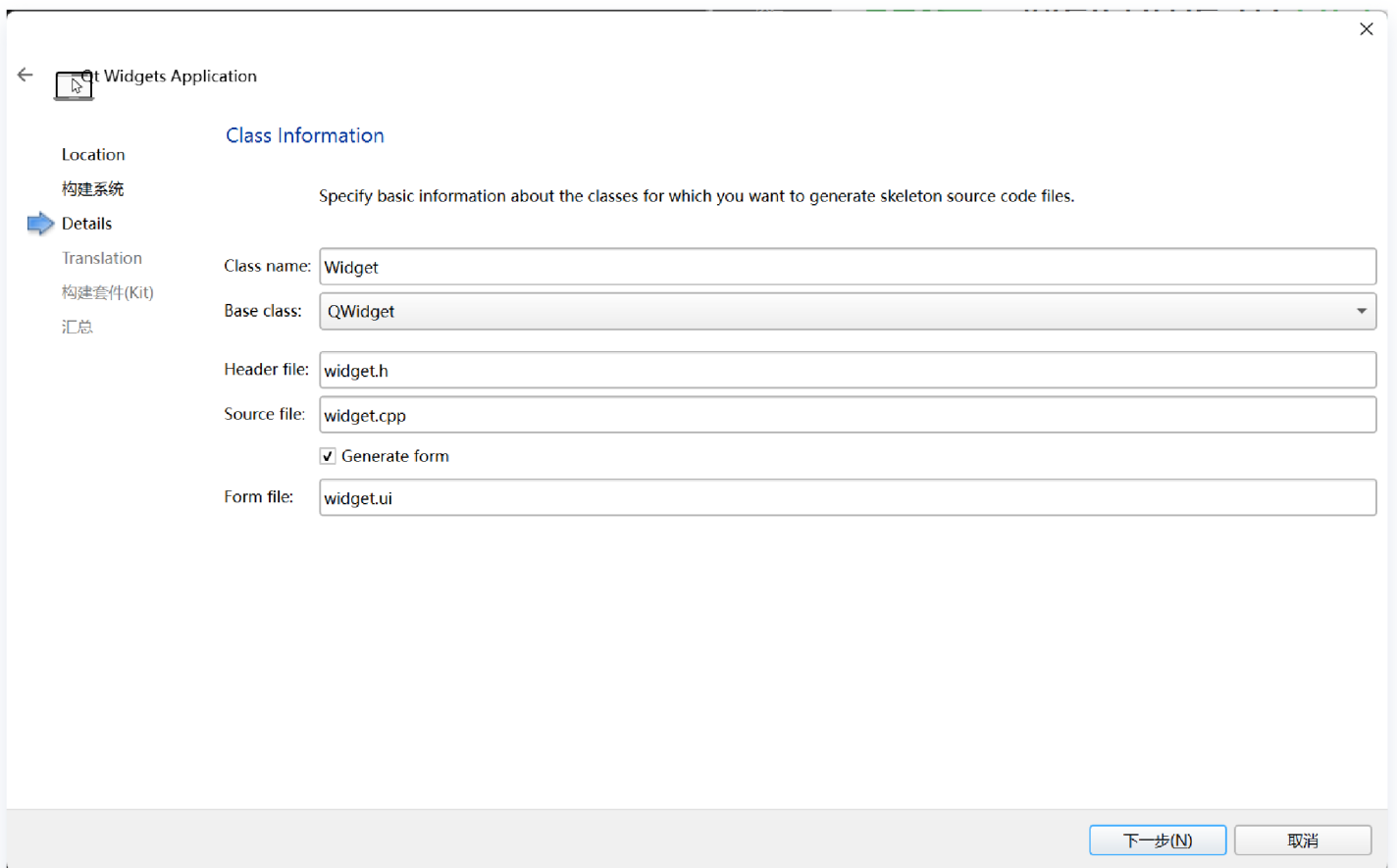
第4章常用界面组件的使用

第2章 GUI程序设计基础

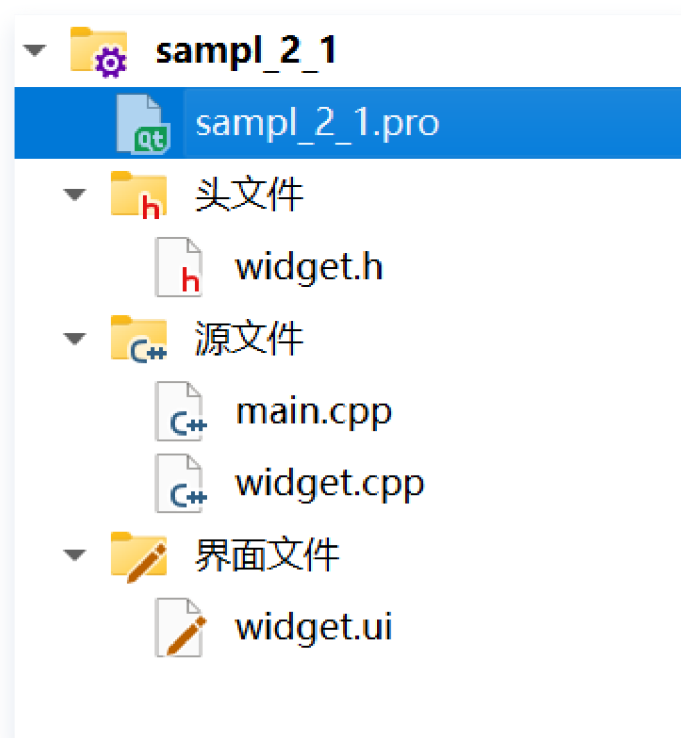
2.1 GUI程序结构与运行机制

2.1.1 创建GUI窗口工程

新建一个Widget Application项目：QWidget作为窗口基类，选中Generate form复选框：



如此，Qt Creator会生成下面这些文件



- **sampl_2_1.pro**: qmake构建系统的项目配置文件，存储了各种项目设置。
 - 项目管理文件：后缀为 **.pro**

- qmake是构建项目的软件，它根据.pro文件生成 **Makefile文件**，然后 **C++编译器** 可以根据Makefile文件进行 **编译和链接**
- qmake还会自动生成 **MOC** (meta-object compiler)和 **UIC** (user interface compiler))文件
- **main.cpp**: 主程序入口文件，实现 **main函数**
- **widget.ui**: 使用 **XML格式** 描述 **元件及布局** 的界面文件
- **widget.h**: **窗口类定义头文件**，用到了.ui文件
- **widget.cpp**: **窗口类定义实现文件**

2.1.2 项目管理文件

```
1  # Qt模块列表,在这里添加需要的模块
2  QT      += core gui
3
4  # 条件判断,如果Qt版本大于4,则添加widgets模块
5  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
6
7  # 通用配置选项
8  CONFIG += c++17
9
10 # 预处理定义列表,下面的例子是添加一个Qt6.0.0之前的所有API的宏定义
11 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
    deprecated before Qt 6.0.0
12
13 # 源文件列表,在这里添加需要的源文件,以符号\衔接下一行
14 SOURCES += \
15     main.cpp \
16     widget.cpp
17
18 # 头文件列表,在这里添加需要的头文件,以符号\衔接下一行
19 HEADERS += \
20     widget.h
21
22 # UI文件列表,在这里添加需要的UI文件,以符号\衔接下一行
23 FORMS += \
24     widget.ui
25
26 # 模板,在这里定义该项目是app(应用程序)还是lib(库),默认是app
27 TEMPLATE = app
28
29 # 默认部署规则,在这里定义部署规则,默认是在当前目录下创建一个bin目录
30 # $$为替换函数的前缀,如$$TARGET,表示替换TARGET变量的值
```

```
31 qnx: target.path = /tmp/${TARGET}/bin
32 else: unix:!android: target.path = /opt/${TARGET}/bin
33 isEmpty(target.path): INSTALLS += target
34
35 # 更多内容参见Qt文档(qmake Manual) https://doc.qt.io/qt-5/qmake-manual.html
36
```

常用的

Variable	Contents
CONFIG	项目的一般配置选项。
DESTDIR	可执行文件或二进制文件将放置在其中的目录。
FORMS	要由用户界面编译器（uic）处理的UI文件列表。
HEADERS	构建项目时使用的头文件（.h文件）的文件名列表。
QT	项目中使用的Qt模块列表。
RESOURCES	要包含在最终项目中的资源（.qrc文件）列表。有关这些文件的更多信息，请参阅Qt资源系统。
SOURCES	构建项目时要使用的源代码文件列表。
TEMPLATE	项目要使用的模板。这将决定构建过程的输出是应用程序、库还是插件。

2.1.3 主函数文件

```
1  #include "widget.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv); // 应用程序对象,管理整个程序的生命周期,一个程序有且只能有一个
8      Widget w; // 声明并创建一个窗口
9      w.show(); // 显示窗口
10     return a.exec(); // 应用程序开始运行,进入消息循环,等待事件处理
11     // a.exec()会阻塞在这里,直到窗口关闭,返回值是程序的退出码
12 }
```

主函数文件解释：

- `#include "widget.h"`：包含自定义的窗口类头文件，以便在主函数中使用。

- `#include <QApplication>` : 包含Qt应用程序类的头文件，以便在主函数中创建和管理应用程序对象。
- `int main(int argc, char *argv[])` : 主函数的入口点，接收命令行参数 `argc` 和 `argv[]`。
- `QApplication a(argc, argv);` : 创建一个Qt应用程序对象 `a`，用于管理整个程序的生命周期。在一个程序中只能有一个应用程序对象。
- `Widget w;` : 声明并创建一个名为 `w` 的窗口对象，使用了自定义的窗口类 `Widget`。
- `w.show();` : 显示窗口，使其可见。
- `return a.exec();` : 启动应用程序的事件处理循环，并等待事件的发生。这会阻塞程序，直到窗口被关闭。`a.exec()` 的返回值是应用程序的退出码。

注：`a.exec()` 是一个阻塞调用，意味着程序会在此处等待，直到窗口关闭为止。

2.1.4 窗体相关文件

- **widget.h:**
 - `widget.h` 文件通常包含窗口类的声明，其中包括类的成员变量和方法的声明。
 - 这个文件定义了窗口的外观和行为，包括窗口的标题、大小、布局等。
- **widget.cpp:**
 - `widget.cpp` 文件包含了窗口类的实现，其中包括了类成员函数的具体实现。
 - 在这个文件中，你可以找到窗口类中各个函数的具体实现，例如构造函数、析构函数以及其他自定义的函数。
- **widget.ui:**
 - `widget.ui` 文件是Qt Designer创建的窗口布局文件，以 `XML格式` 保存。
 - 它包含了 `窗口的布局信息`、`控件的位置和大小` 等设计相关的信息。
 - 这个文件通常使用 `Qt的可视化设计器工具` 进行编辑和设计。
- **ui_widget.h:**
 - `ui_widget.h` 文件是Qt的用户界面文件自动生成的，它是通过 `uic`（用户界面编译器）从 `widget.ui` 文件生成的。
 - 这个文件定义了窗口的用户界面，其中包含了窗口中各个控件的声明和初始化代码。
 - 通常情况下，`不需要手动修改这个文件`，因为它会自动由Qt工具生成和更新。

2.1.4.1 widget.h文件

```
1  #ifndef WIDGET_H
2  #define WIDGET_H
3
```

```

4   #include <QWidget>
5
6   QT_BEGIN_NAMESPACE
7   namespace Ui { // 命名空间
8       class Widget; // ui_widget.h中定义的类,外部声明
9   }
10  QT_END_NAMESPACE
11
12  class Widget : public QWidget
13  {
14      Q_OBJECT // 宏,使用Qt信号与槽机制时必须添加
15
16  public:
17      Widget(QWidget *parent = nullptr);
18      ~Widget();
19
20  private:
21      Ui::Widget *ui; // Ui::Widget类的指针,指向可视化设计生成的窗口
22  };
23  #endif // WIDGET_H

```

widget.h文件解释:

- `#ifndef WIDGET_H` 和 `#define WIDGET_H` : 头文件保护宏, 确保头文件内容只被编译一次, 防止重复包含。
- `#include <QWidget>` : 包含QWidget类的头文件, QWidget是Qt中所有用户界面类的基类。
- `QT_BEGIN_NAMESPACE` 和 `QT_END_NAMESPACE` : Qt的命名空间开始和结束标记。
- `namespace Ui { ... }` : 命名空间Ui, 其中包含了在ui_widget.h中定义的Widget类的前向声明。
- `class Widget : public QWidget` : Widget类的声明, 继承自QWidget类。
- `Q_OBJECT` : Qt中使用信号与槽机制时必须添加的宏, 用于支持Qt的元对象系统。
- `Widget(QWidget *parent = nullptr)` : Widget类的构造函数声明, 可以接收一个QWidget类型的父对象指针, 默认为nullptr。
- `~Widget()` : Widget类的析构函数声明, 用于释放资源和清理工作。
- `Ui::Widget *ui` : 指向Ui::Widget类的指针, 用于访问在可视化设计中生成的窗口部件。
- `#endif // WIDGET_H` : 头文件结束标记, 结束头文件保护宏的定义。

2.1.4.2 widget.cpp文件

```
1  #include "widget.h"
2  #include "ui_widget.h"
3
4  Widget::Widget(QWidget *parent)
5      : QWidget(parent)
6      , ui(new Ui::Widget)
7  {
8      ui->setupUi(this); // 实现了可视化设计的窗口中设计的内容
9  }
10
11 Widget::~Widget()
12 {
13     delete ui;
14 }
```

widget.cpp文件解释：

- `#include "widget.h"` 和 `#include "ui_widget.h"`：分别包含了窗口类的头文件和ui_widget.h文件，以便在此文件中使用Widget类和Ui::Widget类。
- `Widget::Widget(QWidget *parent)`：Widget类的构造函数实现，接收一个QWidget类型的父对象指针，默认为nullptr。
 - 在构造函数中，创建了一个新的Ui::Widget对象并将其分配给ui指针。
 - 调用了ui指针的 `setupUi()` 函数，用于初始化窗口的可视化设计内容。
- `Widget::~~Widget()`：Widget类的析构函数实现，用于释放ui指针所指向的对象的内存空间。

2.1.4.3 ui_widget.h文件

- ui_widget.h和ui_widget.cpp文件都是Qt根据widget.ui 自动生成的文件，因此 无需程序员手动修改这两个文件，本段展示ui_widget相关代码旨在说明ui_widget类的实现原理。

```
1  // ...
2  class Ui_Widget // 封装可视化设计的界面,注意: Ui_Widget没有父类,不是窗口类
3  {
4  public:
5      QLabel *label;
6
7      void setupUi(QWidget *Widget)
8      {
9          // ...
```

```

10     } // setupUi
11
12     // ...
13 };
14
15 namespace Ui {
16     class Widget: public Ui_Widget {};
17 } // namespace Ui

```

2.1.4.4 ui_widget.cpp文件

```

1  #ifndef UI_WIDGET_H
2  #define UI_WIDGET_H
3
4  #include <QtCore/QVariant>
5  #include <QtWidgets/QApplication>
6  #include <QtWidgets/QWidget>
7
8  QT_BEGIN_NAMESPACE
9
10 class Ui_Widget //封装了可视化设计的窗口的类
11 {
12 public:
13
14     // 注意:Ui_Widget没有父类,它不是窗口类,只是一个可视化设计的类
15     void setupUi(QWidget *Widget)
16     {
17         // ....
18     } // setupUi
19
20     // ....
21 };
22
23 namespace Ui {
24     class Widget: public Ui_Widget {};
25 } // namespace Ui
26
27 QT_END_NAMESPACE
28
29 #endif // UI_WIDGET_H
30

```


ui_widget.cpp文件解释：

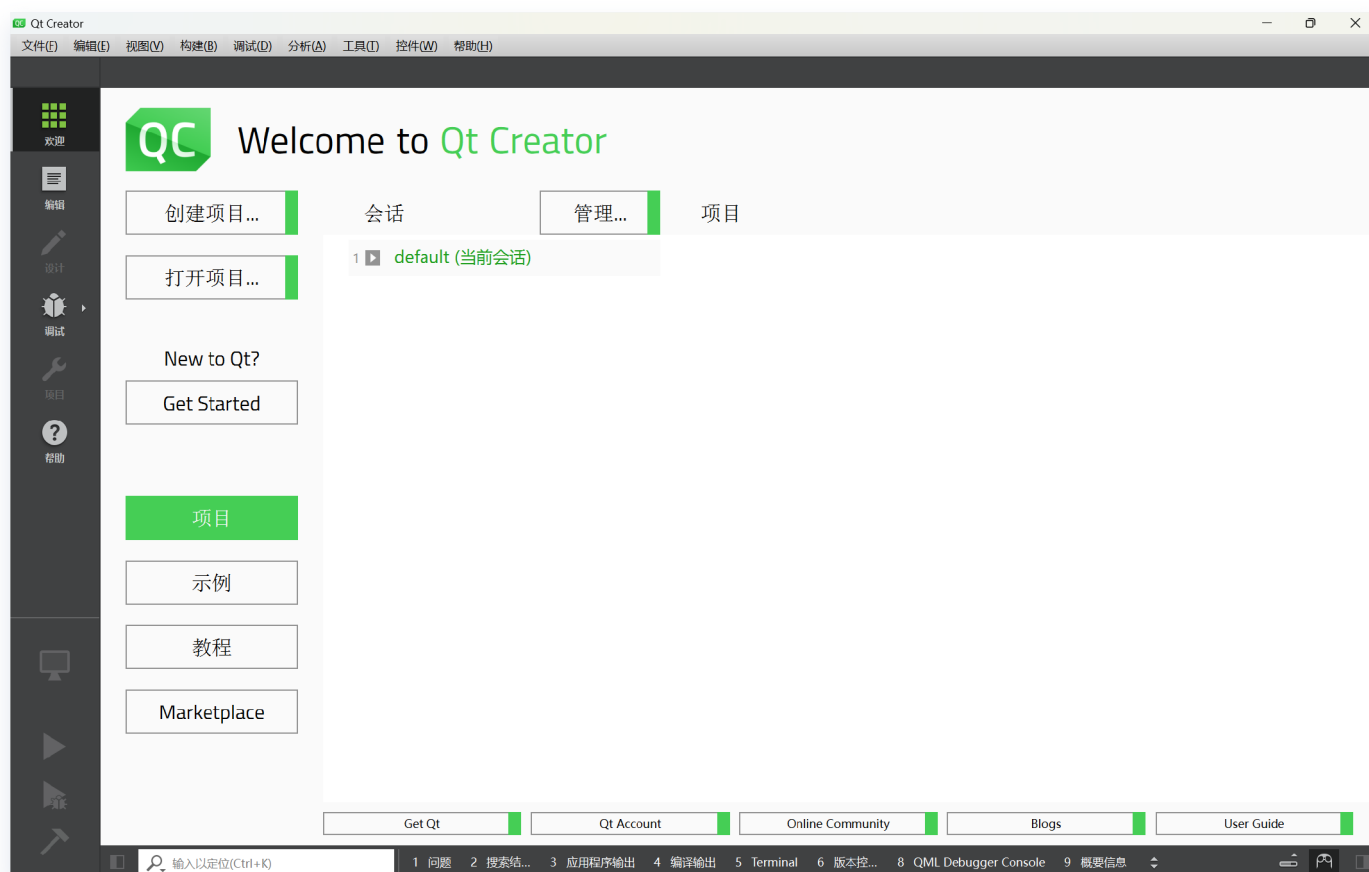
- `#ifndef UI_WIDGET_H` 和 `#define UI_WIDGET_H`：头文件保护宏，确保头文件内容只被编译一次，防止重复包含。
- `#include <QtCore/QVariant>`、`#include <QtWidgets/QApplication>` 和 `#include <QtWidgets/QWidget>`：包含了Qt的相关头文件。
- `class Ui_Widget`：封装了可视化设计的窗口的类，用于定义窗口的布局和部件。
- `void setupUi(QWidget *Widget)`：Ui_Widget类的成员函数，用于设置窗口的布局和部件。
- `namespace Ui { class Widget: public Ui_Widget {};`：Ui命名空间中的Widget类，继承自Ui_Widget类，用于访问可视化设计的窗口部件。
- `#endif // UI_WIDGET_H`：头文件结束标记，结束头文件保护宏的定义。

2.1.5 创建第一个Qt GUI工程

下面是用Qt Creator创建一个GUI应用程序的步骤（不同版本的Qt可能引导UI顺序不一样，但大体流程相同）：

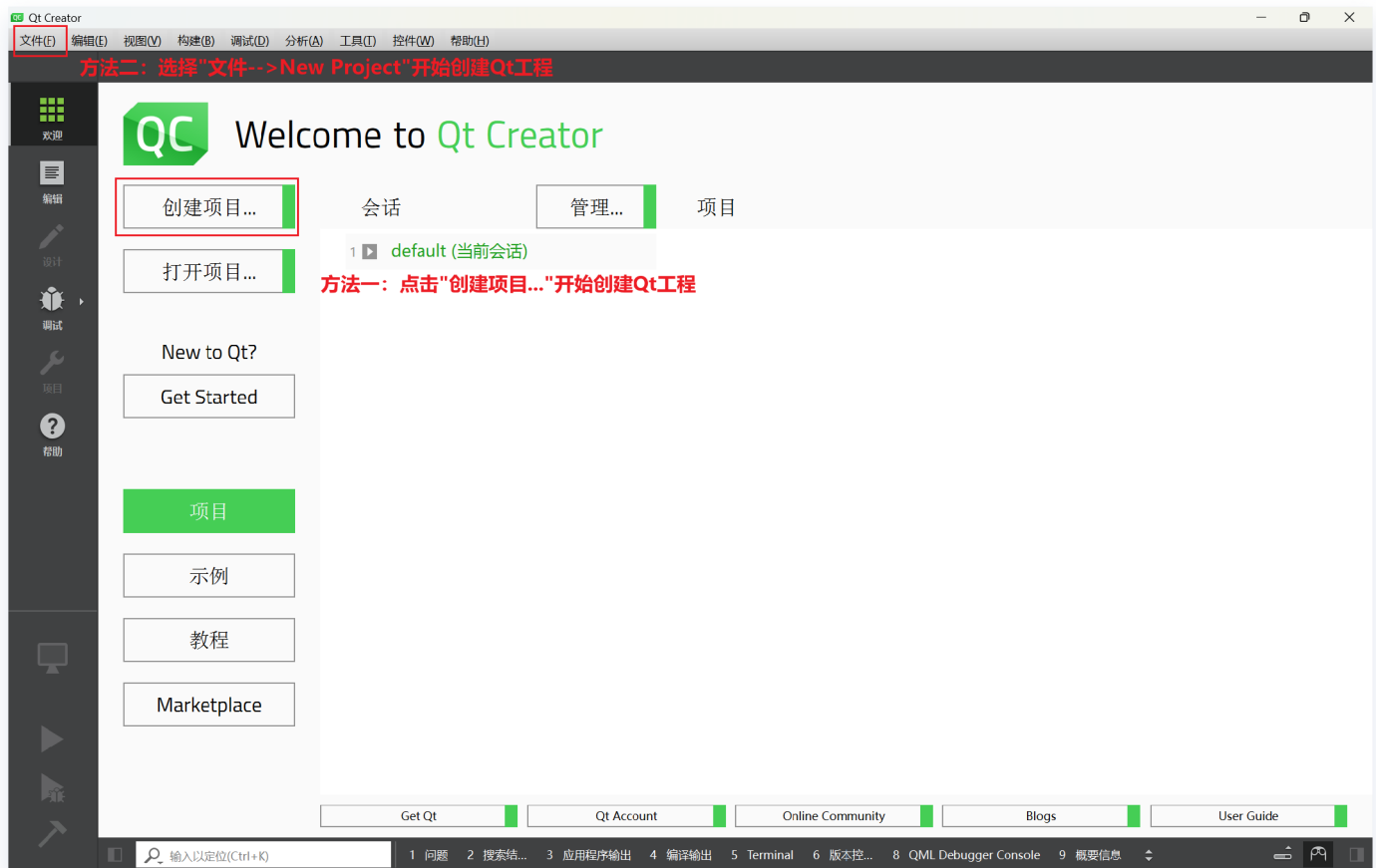
1. 打开Qt Creator：

启动Qt Creator。



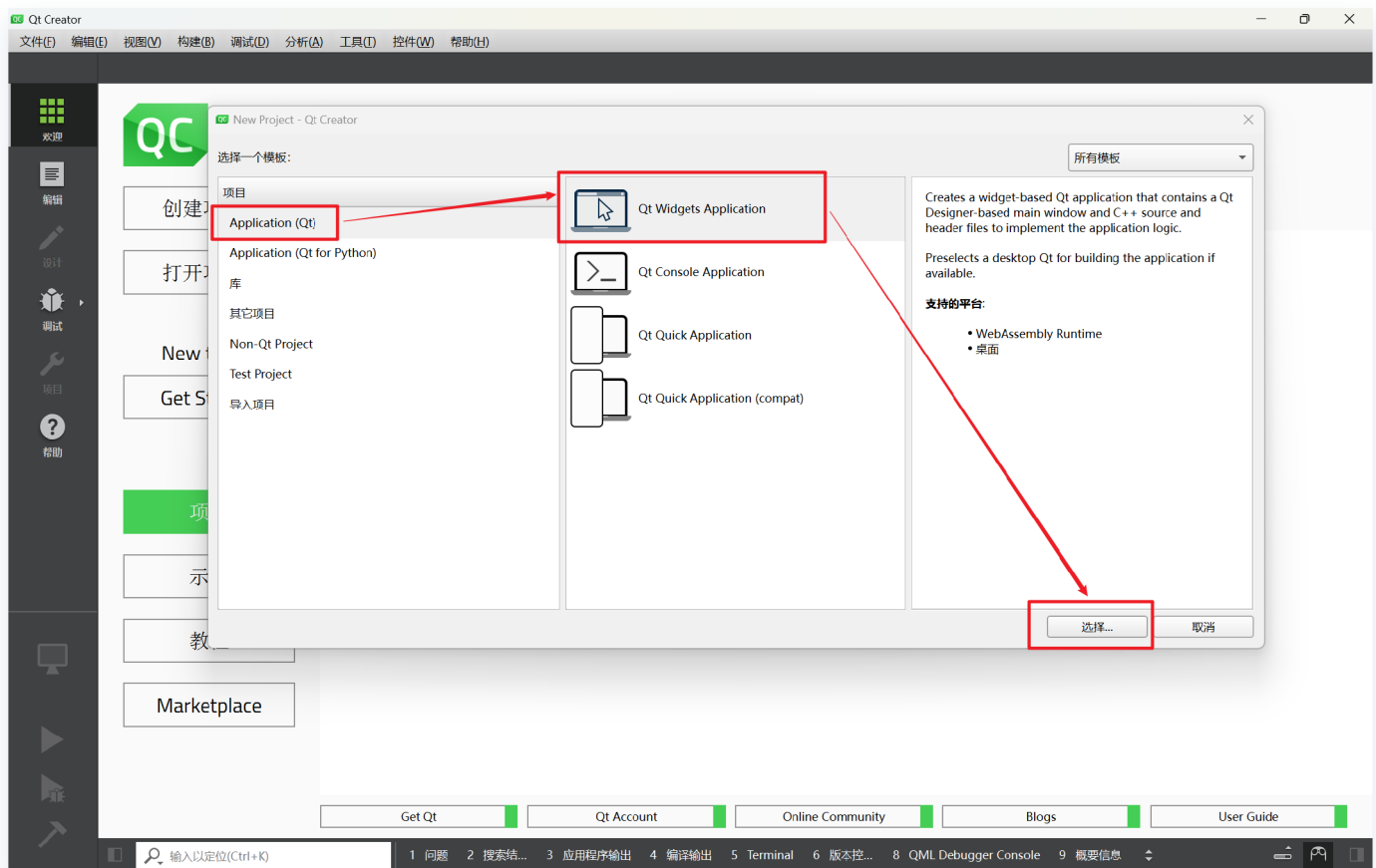
2. 创建新项目：

在欢迎界面上，点击“新建项目”（New Project），或者在菜单栏上选择“文件 --> New Project”



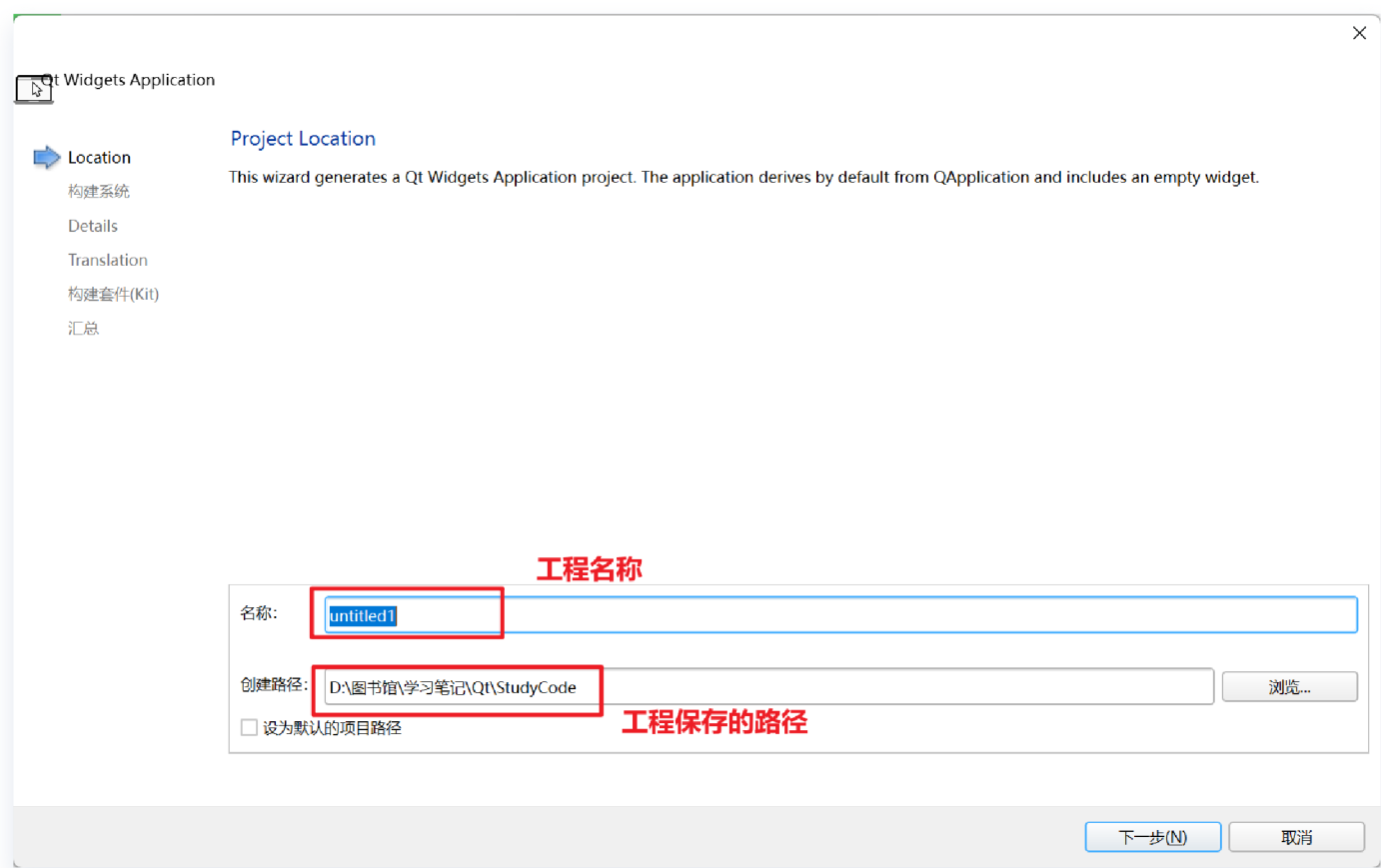
3. 选择项目类型：

在弹出的对话框中，选择“应用程序”（Application）下的“Qt Widgets应用程序”（Qt Widgets Application），然后点击“选择”（Choose）。



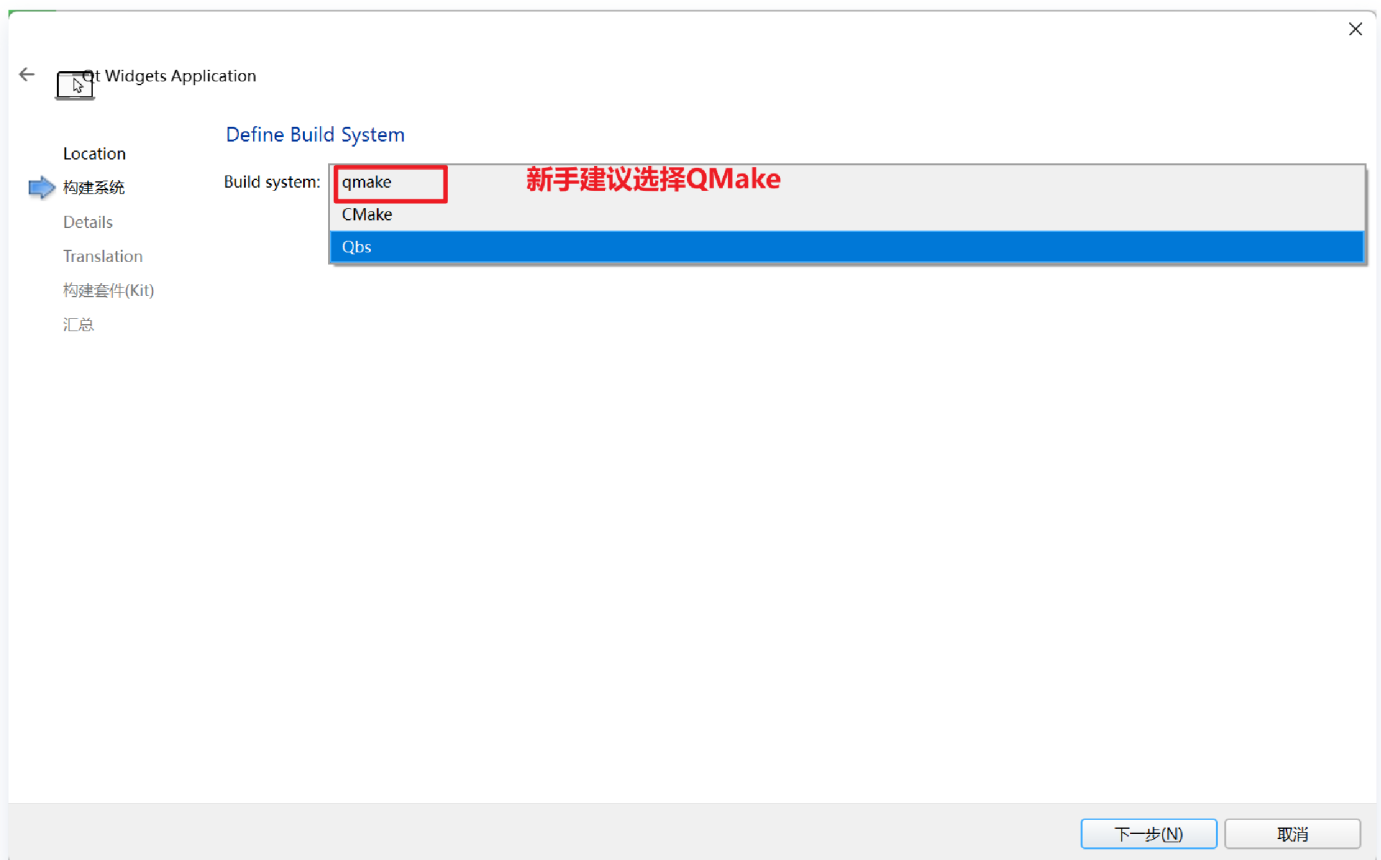
4. 设置项目名称和位置：

输入项目名称， 并选择保存位置。然后点击“下一步”（Next） ,注意Qt6以下版本不支持中文工程名。



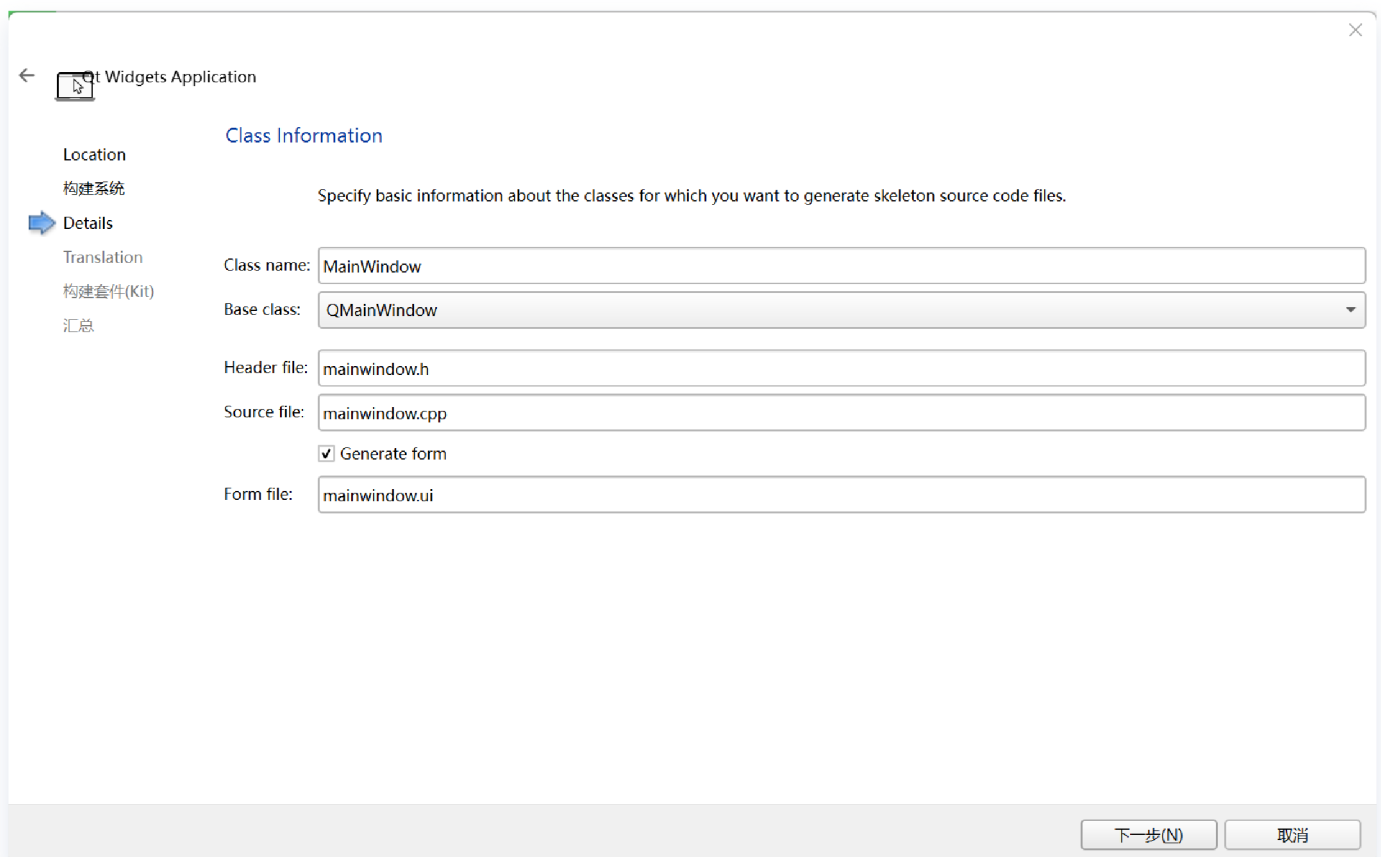
5. 选择构建工具：

选择合适的Qt版本和构建工具， 通常会自动检测到。确保选择正确的工具链， 然后点击“下一步”。



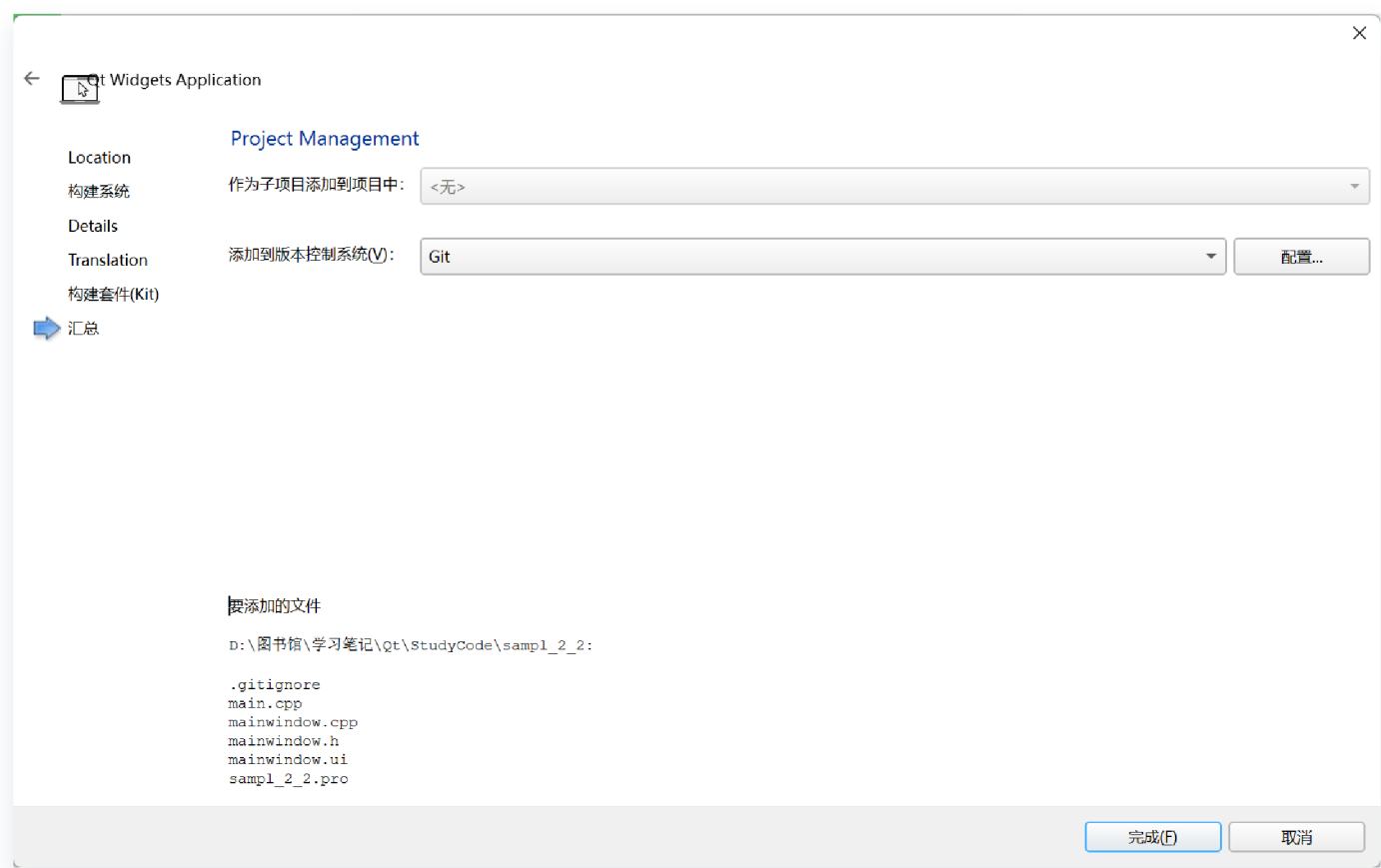
6. 配置类信息：

在下一步中，设置主窗口类的名称（如 `MainWindow` ），并选择基类为 `QMainWindow` 。然后点击“下一步”。



7. 设置项目管理信息：

如果需要，可以设置项目的版本和其他信息。点击“下一步”。



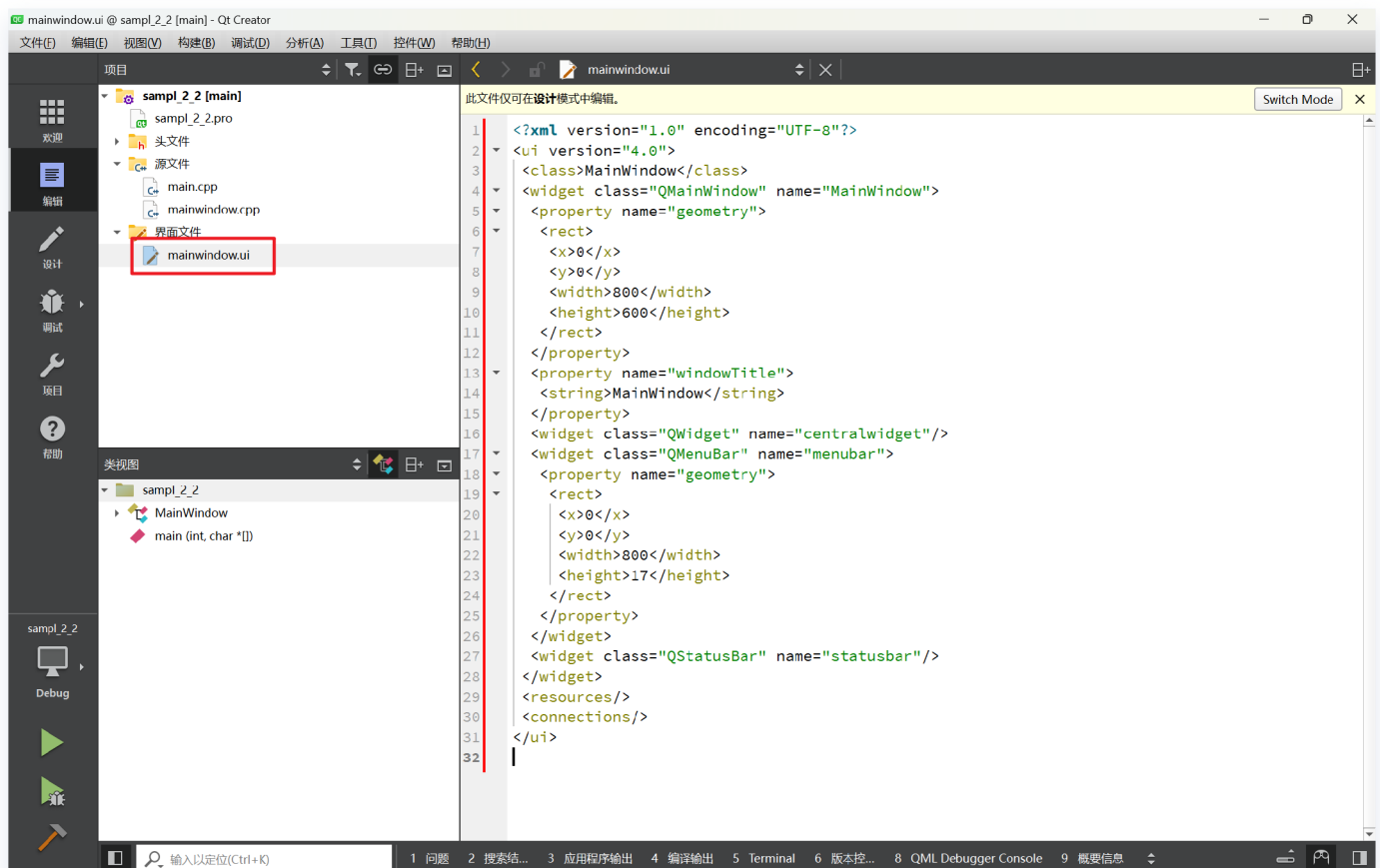
8. 完成创建：

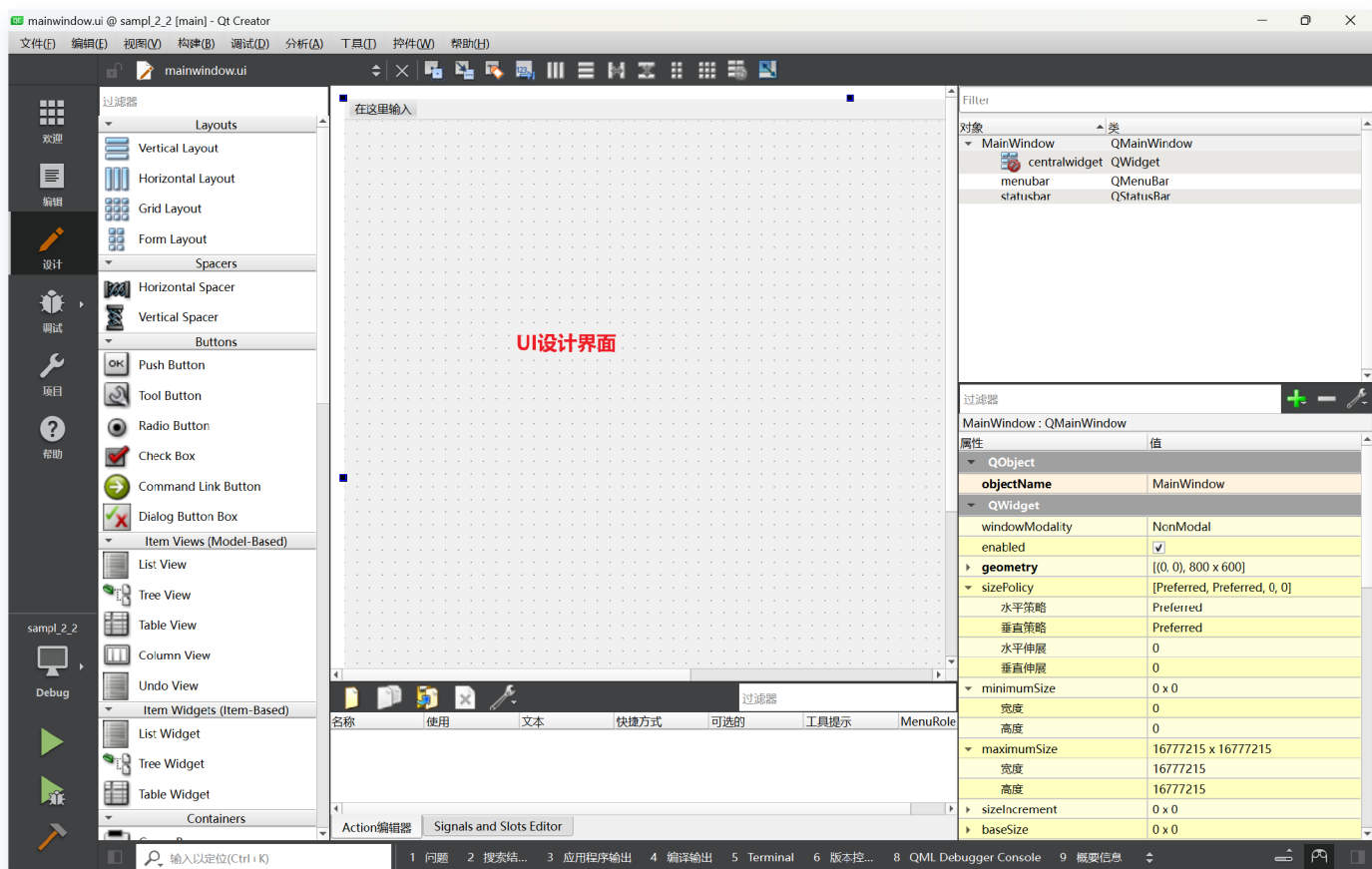
检查所有设置无误后，点击“完成”（Finish）。



9. 设计界面:

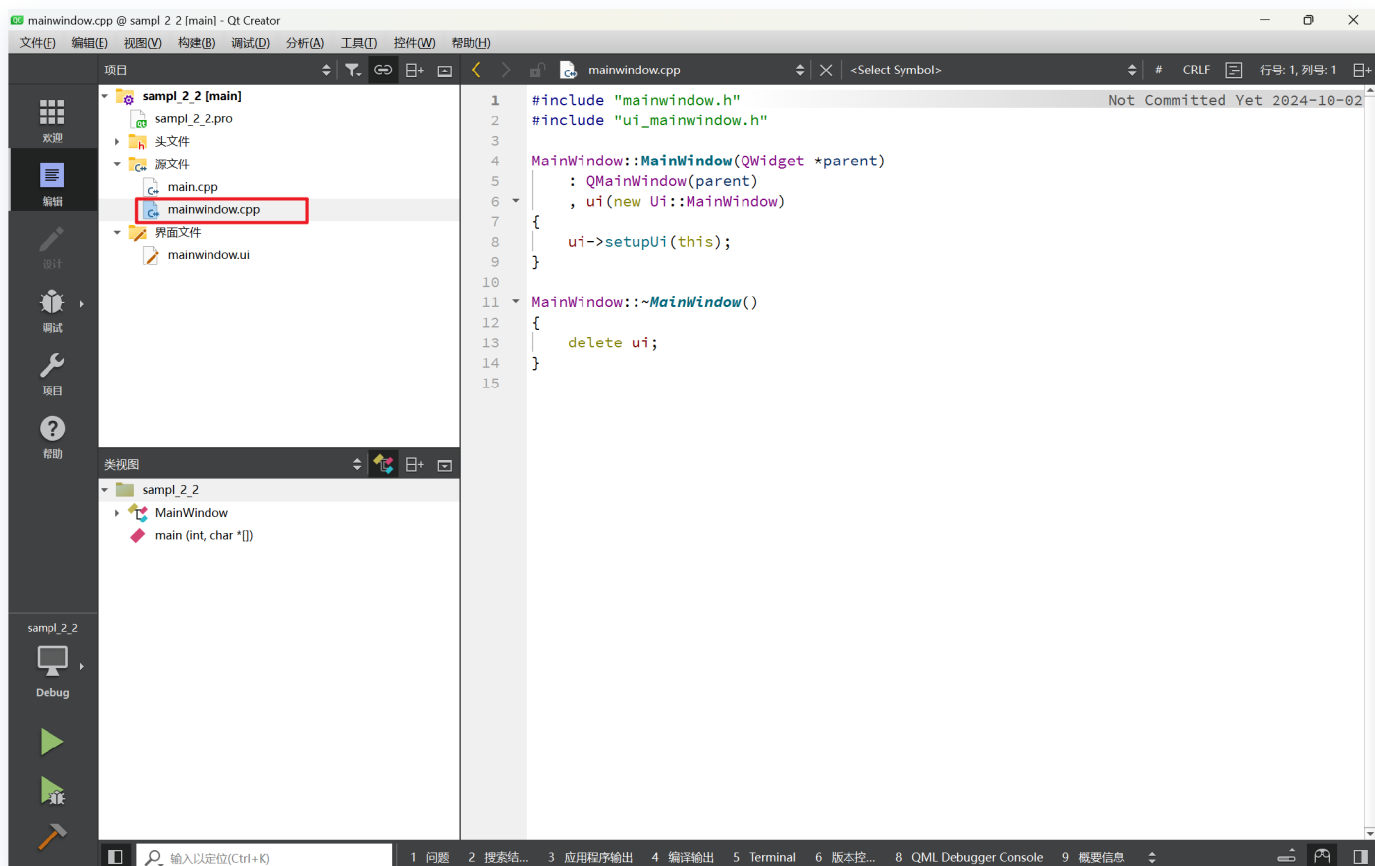
项目创建后，Qt Creator会打开项目窗口。在左侧的项目视图中，找到 `mainwindow.ui` 文件，双击打开它。使用Qt Designer来设计你的GUI界面。





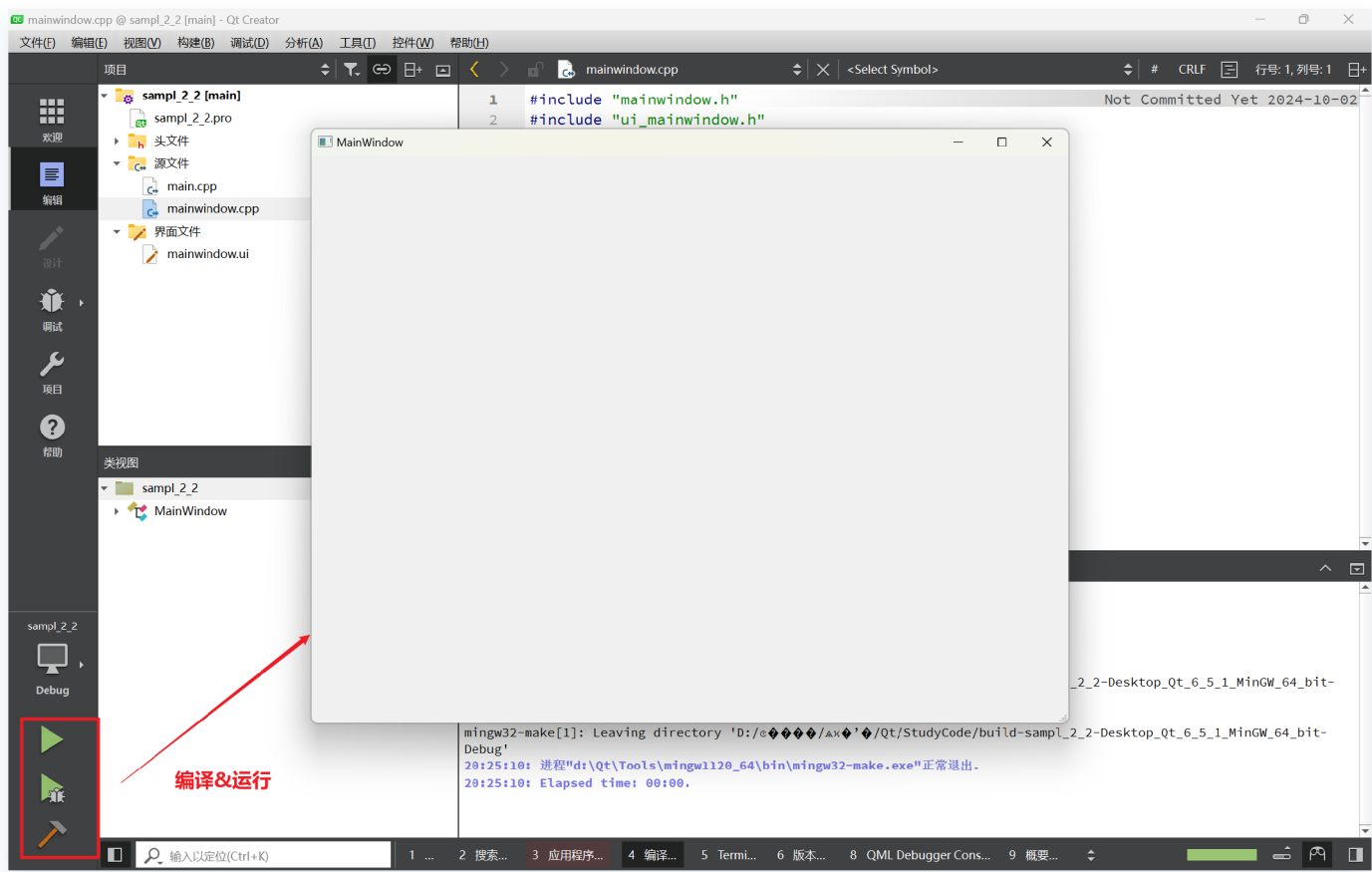
10. 编写代码：

在 `mainwindow.cpp` 文件中，可以添加逻辑代码。使用信号和槽机制来处理事件。



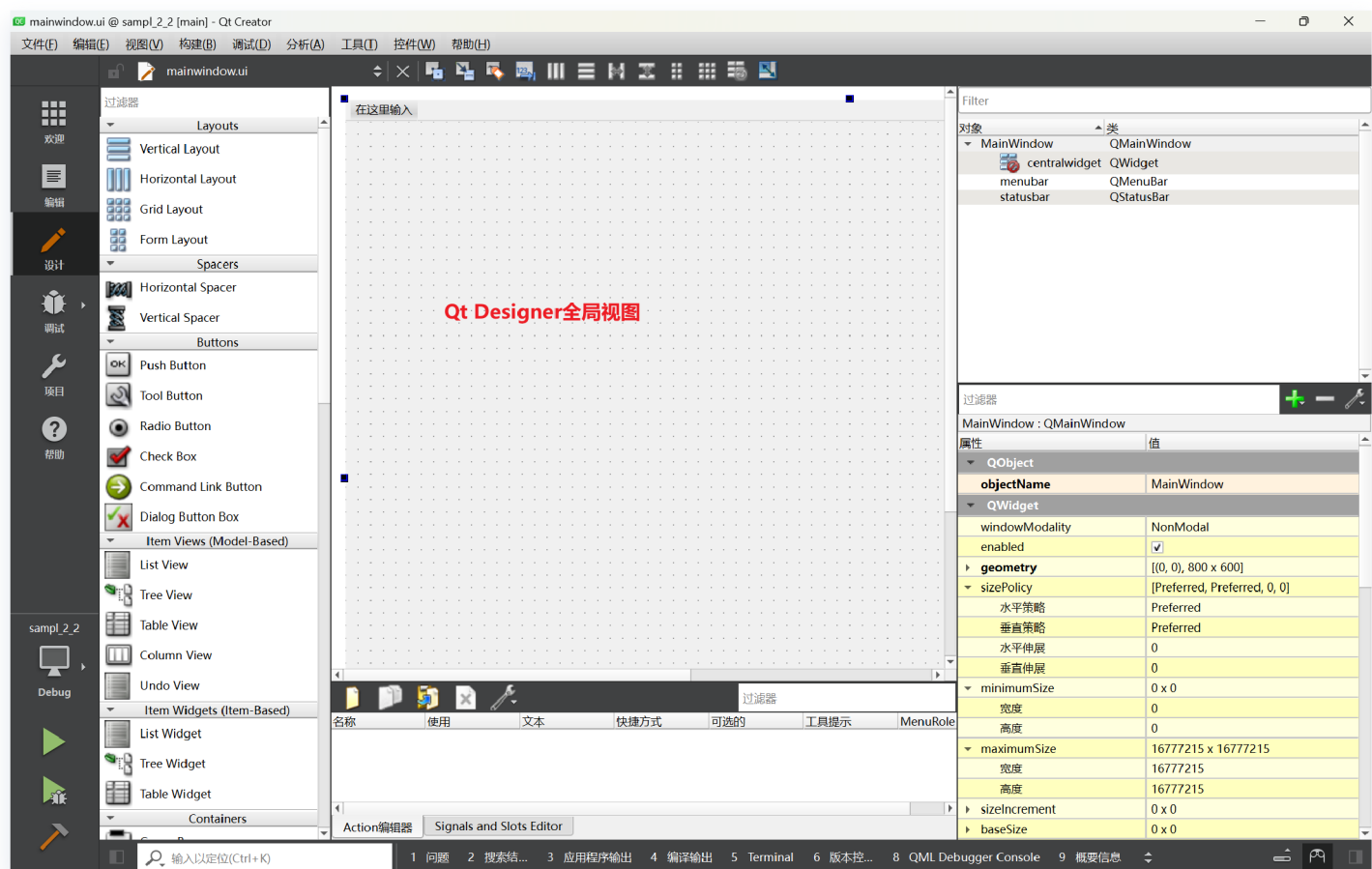
11. 构建和运行：

点击工具栏上的绿色播放按钮（Run）来编译并运行你的应用程序。确保没有错误后，程序会启动，显示你设计的界面。



以上就是用Qt Creator创建一个Qt GUI应用程序的基本步骤。

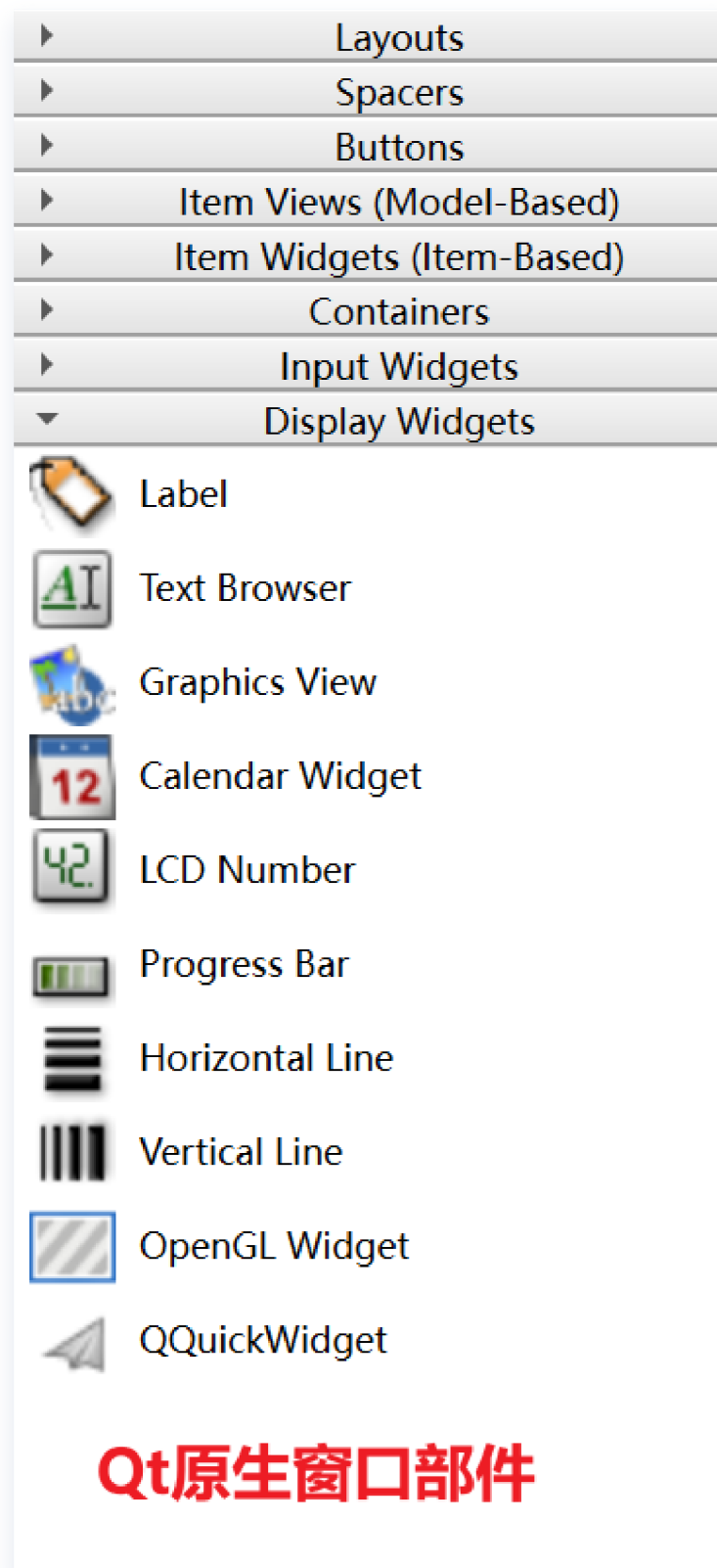
2.1.6 UI图像设计界面Qt Designer简介



Qt Designer是一个图形界面设计工具，主要用于创建和设计Qt应用程序的用户界面。它的主要特点包括：

1. 设计视图：

提供拖放功能，可以方便地将各种控件（如按钮、标签、文本框等）从左侧的工具箱拖入设计区域。




2. 控件属性：

右侧的属性编辑器允许你调整选中控件的属性，如大小、文本、颜色等。

Filter

对象列表筛选框

对象	类
▼ MainWindow	QMainWindow
 centralwidget	QWidget
menubar	QMenuBar
statusbar	QStatusBar

对象列表

过滤器

属性列表筛选框



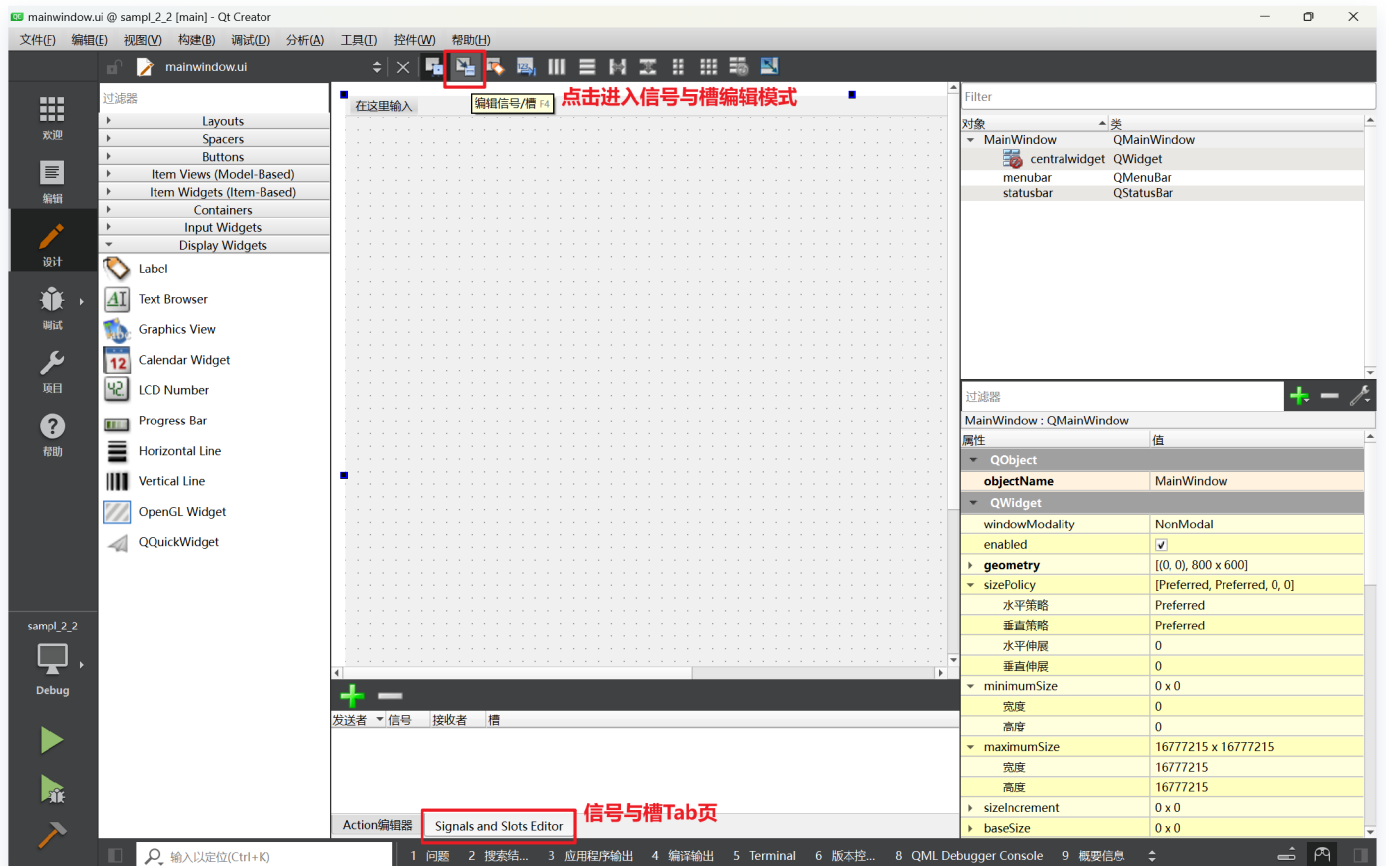
MainWindow : QMainWindow

属性	值
▼ QObject	
objectName	MainWindow
▼ QWidget	
windowModality	NonModal
enabled	<input checked="" type="checkbox"/>
▶ geometry	[(0, 0), 800 x 600]
▼ 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
▶ baseSize	0 x 0

3. 信号与槽：

支持通过简单的拖放方式连接控件之间的信号和槽，以实现交互功能。



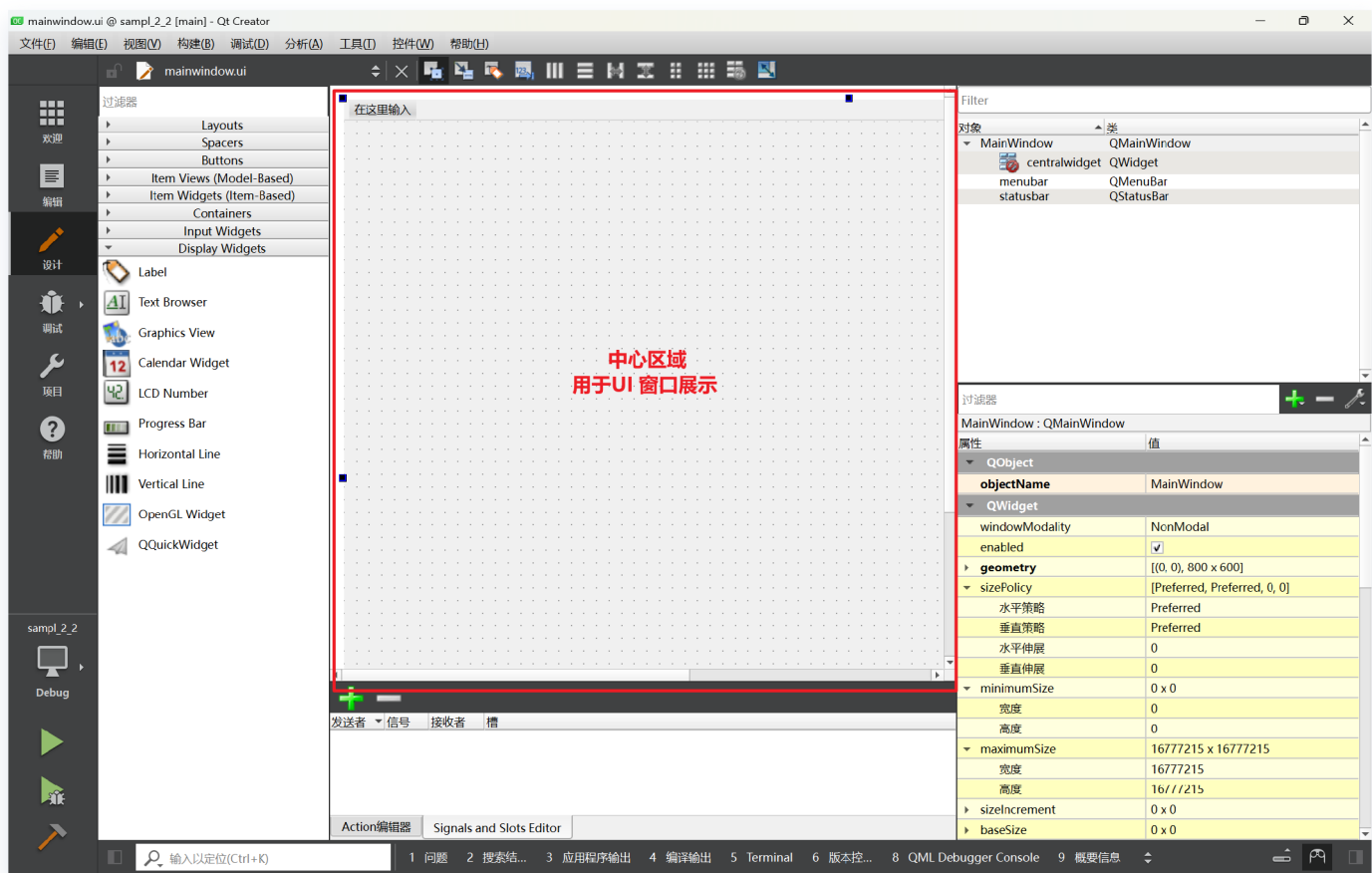
4. 布局管理：

提供多种布局选项（如网格、垂直和水平布局），帮助自动调整控件在窗口中的位置和大小。



5. 预览功能:

可以实时预览设计效果，确保界面符合预期。



6. 保存为.ui文件:

设计的界面以.ui文件格式保存，这些文件可以直接在Qt项目中使用。

Qt Designer简化了界面设计的过程，让开发者可以专注于应用程序的功能实现。

2.1.7 在UI图像设计界面创建简单的信号与槽链接

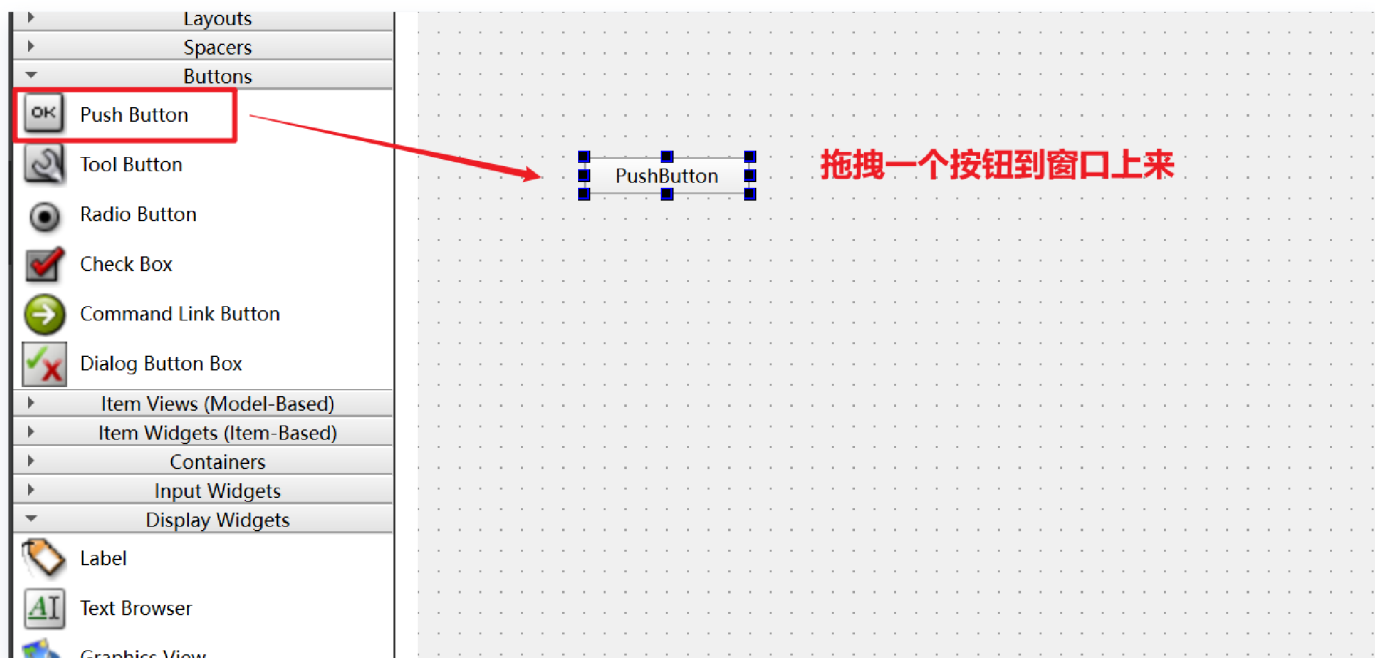
在Qt Designer中建立简单的信号与槽连接的步骤如下:

1. 打开Qt Designer:

启动Qt Designer并打开你的界面文件 (.ui)。

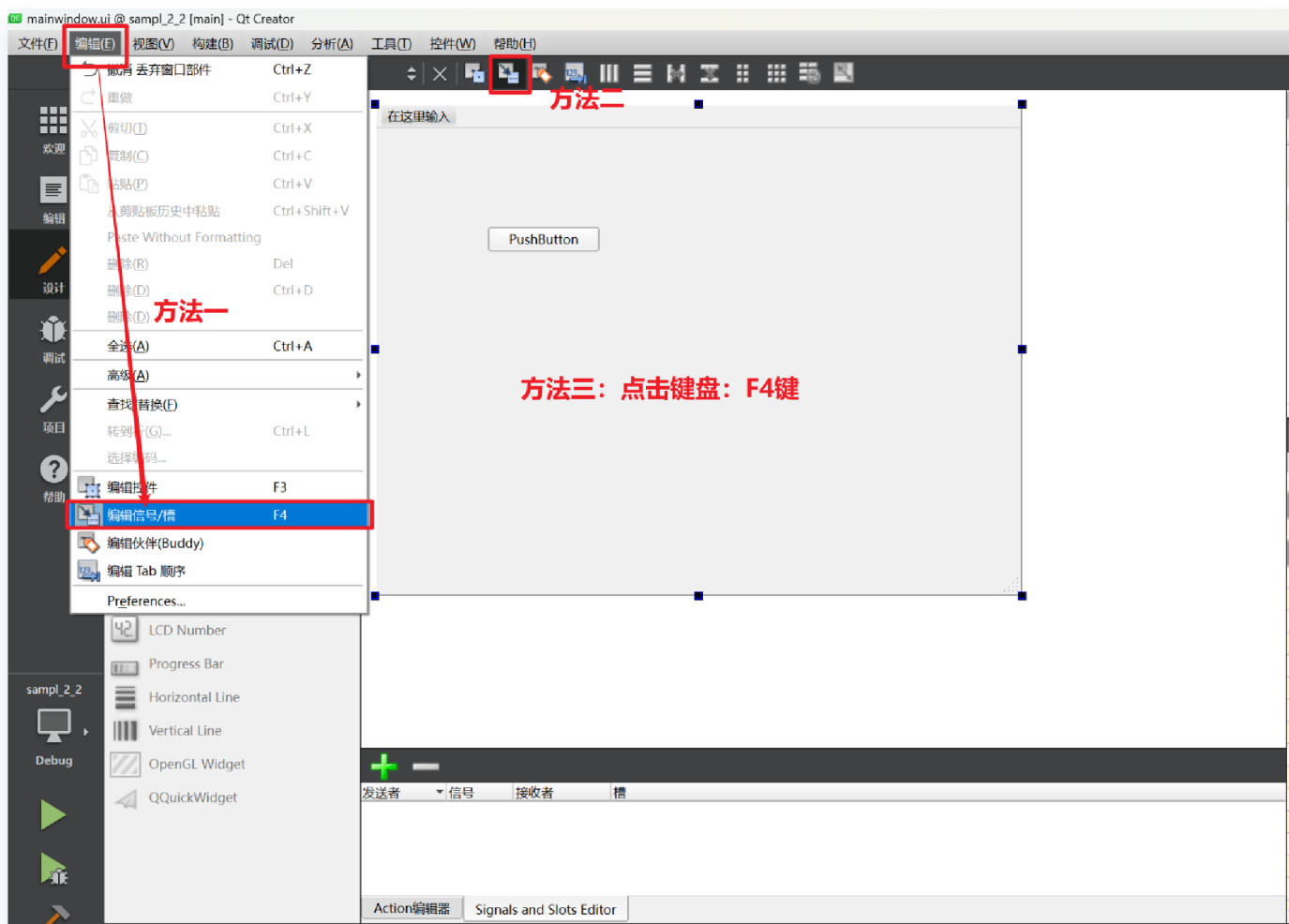
2. 选择控件：

在设计视图中，选择一个控件，例如按钮（QPushButton）。



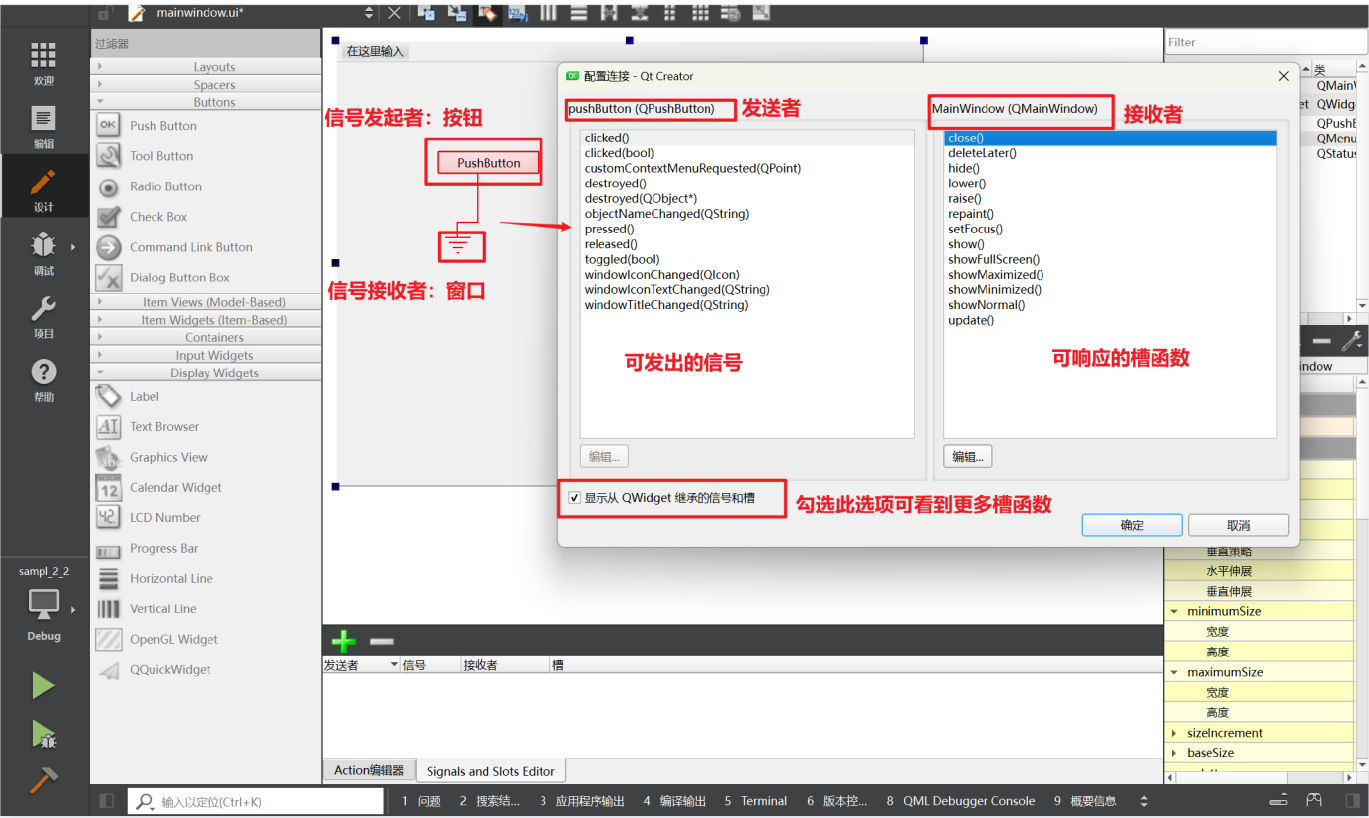
3. 打开信号与槽编辑器：

在菜单栏中，点击“编辑” > “编辑信号/槽”（Edit > Edit Signals/Slots），或者使用快捷键（通常是F4）。



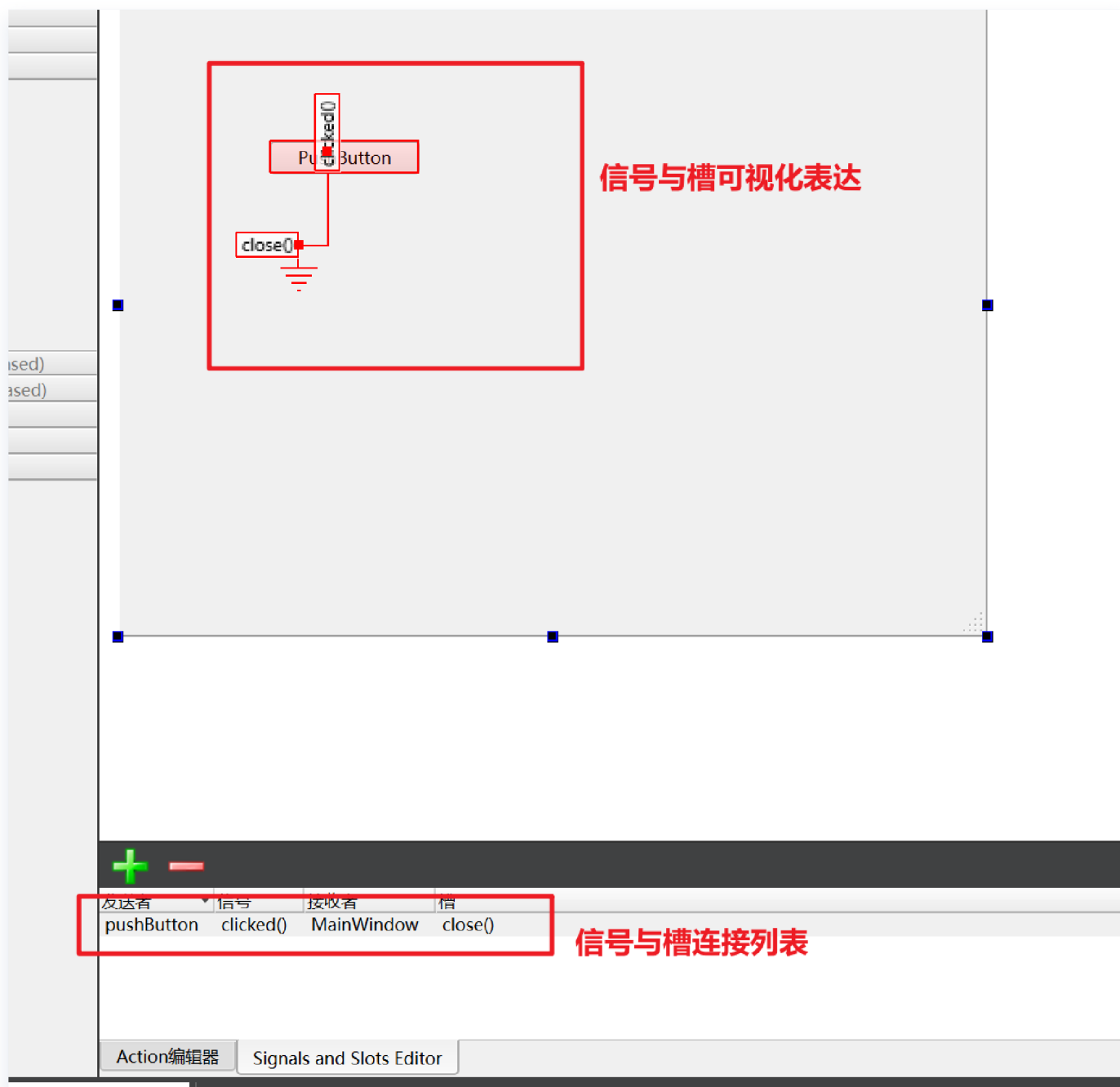
4. 连接信号与槽：

- 点击你选择的控件（如按钮），然后拖动到另一个控件（如文本框或标签）上。
- 当你释放鼠标时，会出现一个对话框，显示该控件的可用信号和槽。
- 选择一个信号（如 `clicked()` ），然后选择要连接的槽（如 `close()` ）或者选择自定义槽。



5. 完成连接：

确认连接后，点击“确定”或“OK”。连接成功后，会在设计视图中看到连接线。



6. 保存并生成代码：

保存你的.ui文件，退出信号与槽编辑模式。Qt Creator会在生成代码时自动处理信号与槽的连接。

通过以上步骤，你就可以在Qt Designer中成功建立简单的信号与槽连接，实现控件之间的交互。

第3章Qt框架功能概述

第4章常用界面组件的使用