

1. Qt 概述
  - 1.1 简介
  - 1.2 优点
2. 创建 Qt 项目
  - 2.1 mian 程序解释
3. 命名规范与快捷键
  - 3.1 pro 文件的解释 (qmake 构建才有.pro 文件)
    - 3.1.1 Qt 模块
  - 3.2 头文件
  - 3.3 命令规范
  - 3.4 快捷键
4. 组件类
  - 4.1 QPushButton(按钮控件)
    - 4.1.1 基本简介
    - 4.1.2 信号
    - 4.1.3 槽函数
    - 4.1.4 示例
  - 4.2 QUdpSocket(网络控件)
    - 4.2.1 基本简介
  - 4.3 QMainWindow(主窗口类)
    - 4.3.1 基本简介
    - 4.3.2 菜单栏(一个窗口最多一个)
    - 4.3.3 工具栏(一个窗口可以有多个)
    - 4.3.4 状态栏(最多只有一个)
    - 4.3.5 铆接部件(浮动控件 可以有多个) 和 核心控件(只能有一个)
  - 4.4 对话框(QDialog)
    - 4.4.1 模态对话框(打开以后, 不能操作其他窗口——阻塞其他窗口)——自定义对话框
    - 4.4.2 非模态对话框(打开以后, 能操作其他窗口——不阻塞其他窗口)——自定义对话框
    - 4.4.3 消息对话框(模态对话框)
    - 4.4.4 其他对话框
5. 对象树
  - 5.1 示例
  - 5.2 qt 窗口坐标系
6. 信号和槽
  - 6.1 基本概念
  - 6.2 技巧
  - 6.3 自定义信号和槽
  - 6.4 自定义信号和槽重载问题解决
  - 6.5 拓展
  - 6.6 Lambda 表达式 (用于一个信号连接多个槽函数 或者 多个信号连接一个槽函数)
7. 资源文件
  - 7.1 添加 ui 文件
    - 7.1.1 新建时添加 ui 文件
    - 7.1.2 已经创建的文件中添加 ui 文件
  - 7.2 使用 ui 文件
  - 7.3 添加资源文件
    - 7.3.1 创建资源文件
    - 7.3.2 回到资源文件界面
    - 7.3.3 添加资源
    - 7.3.4 使用
8. 界面布局
  - 8.1 控件布局

## 8.2 弹簧

## 9. ui 控件

### 9.1 按钮控件

#### 9.1.1 工具按钮

#### 9.1.2 单选按钮

#### 9.1.3 复选按钮

### 9.2 QListWidget 控件

### 9.3 QTreeWidget

### 9.4 QTableWidget

### 9.5 其他控件

#### 9.5.1 scroll Area(下拉条)

#### 9.5.2 Tool Box(类似抽屉)

#### 9.5.3 Tab Widget (类似网页切换)

#### 9.5.4 Stacked Widget (栈控件——翻页的切换页面)

#### 9.5.5 combo Box(下拉框) font Combo Box(字体下拉框)

### 9.6 自定义控件封装

#### 9.6.1 创建 ui 文件

#### 9.6.2 引入到其他窗口界面中

#### 9.6.3 给自己定义控件添加功能

## 10. QEvent(事件)

### 10.1 鼠标事件

### 10.2 定时器事件 (概念与 stm32 中学的基本类似)

### 10.3 事件分发器(event())

### 10.4 事件过滤器

### 10.5 QPainter(绘图)

#### 10.5.1 绘图事件

#### 10.5.2 绘图高级设置

#### 10.5.3 手动调用绘图事件(利用绘图类画图片中的图)

#### 10.5.4 绘图设备

## 11. QFile(对文件进行读写操作)

### 11.1 QFileInfo对文件信息的读取

## 12. 翻金币项目

### 12.1 主窗口

### 12.2 自定义按钮类

### 12.3 选择关卡类

### 12.4 翻金币场景

### 12.5 金币类

### 12.6 关卡数据类

## 13. 打包

### 13.1 步骤1

### 13.2 步骤2(只做到这一步，需要使用者有对应的qt环境)

### 13.3 步骤3(这一步之后，就可以将这个文件夹压缩，发给使用者)

### 13.4 步骤4 (可省略 使用第三方，形成安装程序 固定生成的是setup.exe的安装包)

# 1. Qt 概述

## 1.1 简介

Qt 是一个 跨平台 的 C++图形用户界面应用程序框架。

支持 WIndow、Linux、OS。

安装: <https://www.qt.io/>

安装教程: [https://blog.csdn.net/qg\\_51355375/article/details/143222677](https://blog.csdn.net/qg_51355375/article/details/143222677)

## 1.2 优点

- 跨平台
- 接口简单
- 一定程度简化内存回收机制
- 开发效率高
- 支持嵌入式开发

## 2. 创建 Qt 项目

第一步, 点击 Qt Creator

第二步, New Project 创建新工程

第二步 (其他方式), 点击 Open Project 打开一个工程

第三步, 选择 Application-》Qt Widgets Application

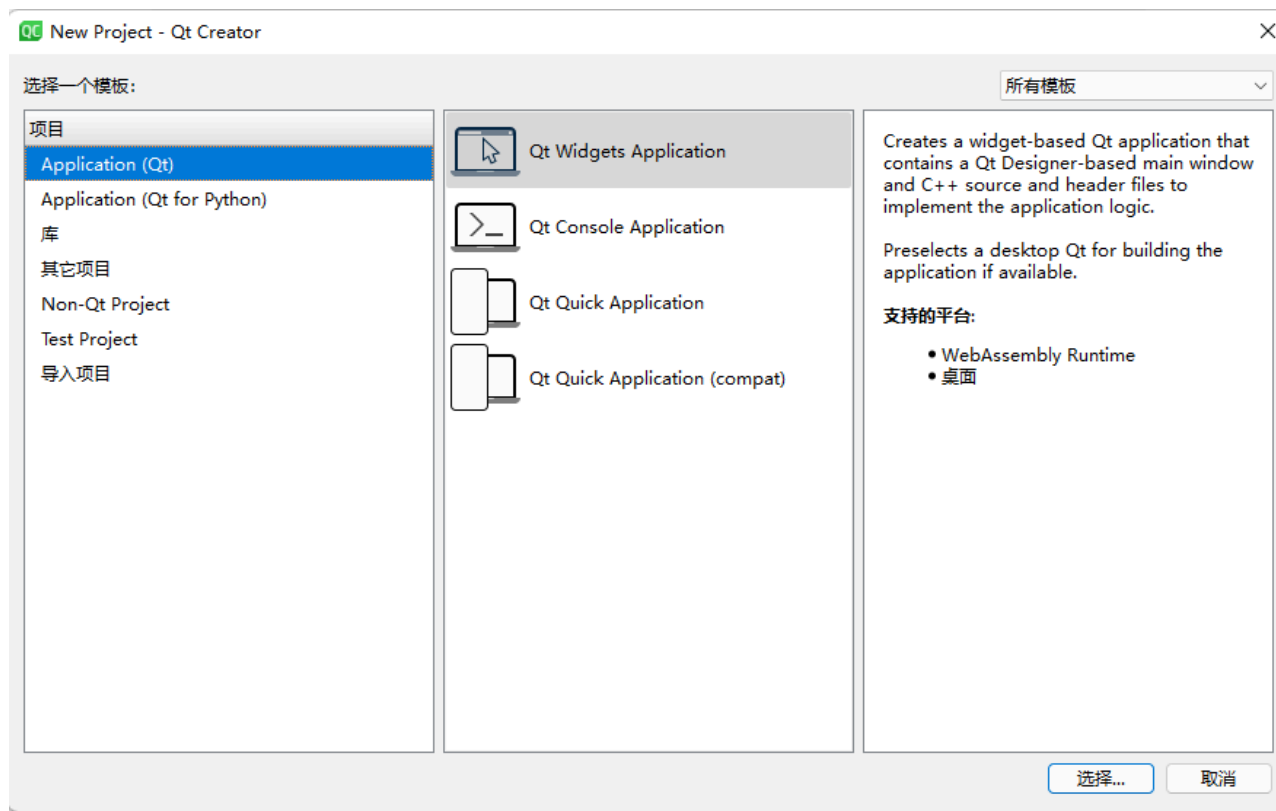


Figure 1

第四步，切记不能有中文或者空格。

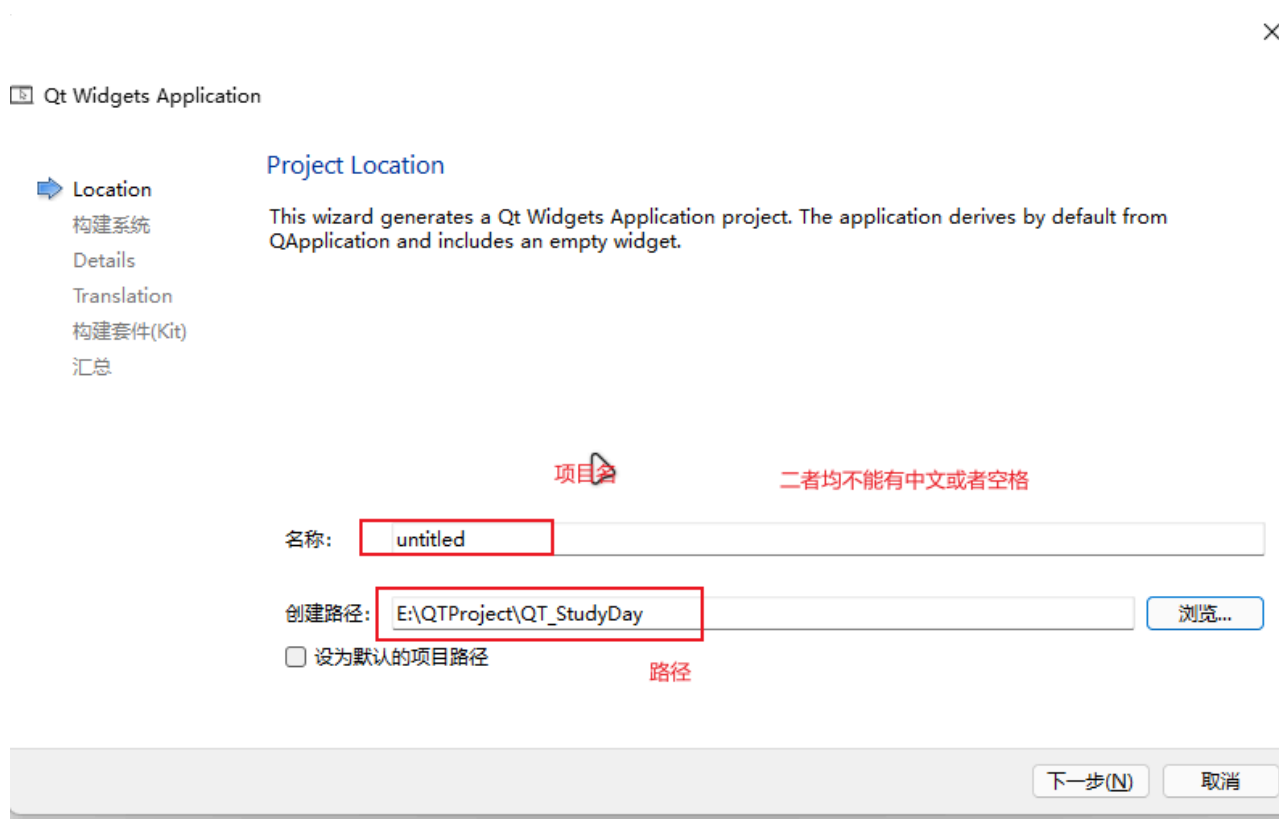
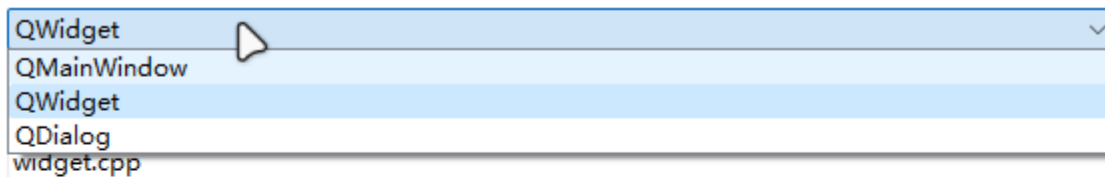
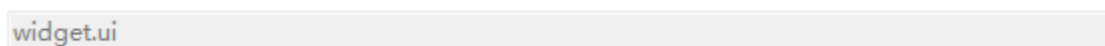


Figure 2



第五步，☐ Generate form



QMainWindow（界面会出现菜单栏、状态栏等）与QDialog（以弹出的对话框的形式出现）都继承与QWidget（空窗口）

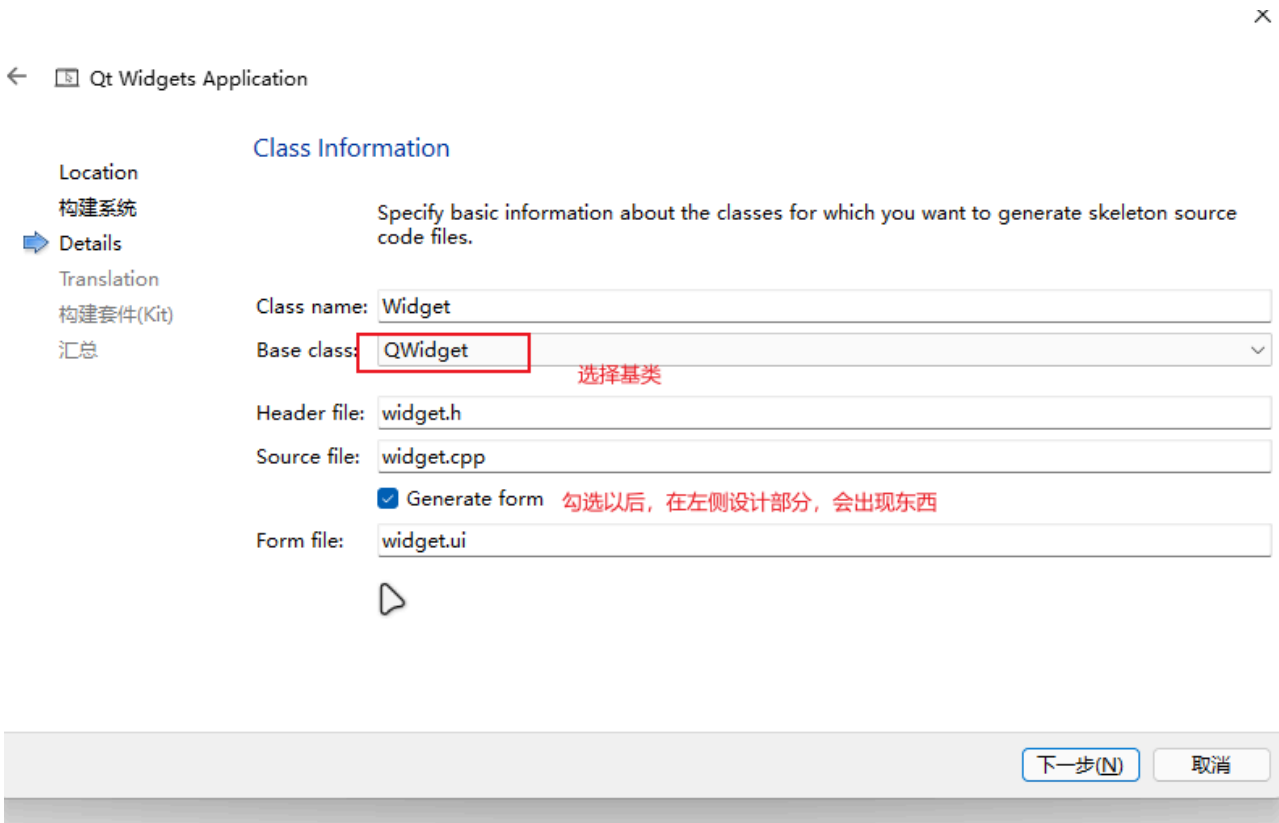


Figure 3

第六步，选择自己所需要的环境。

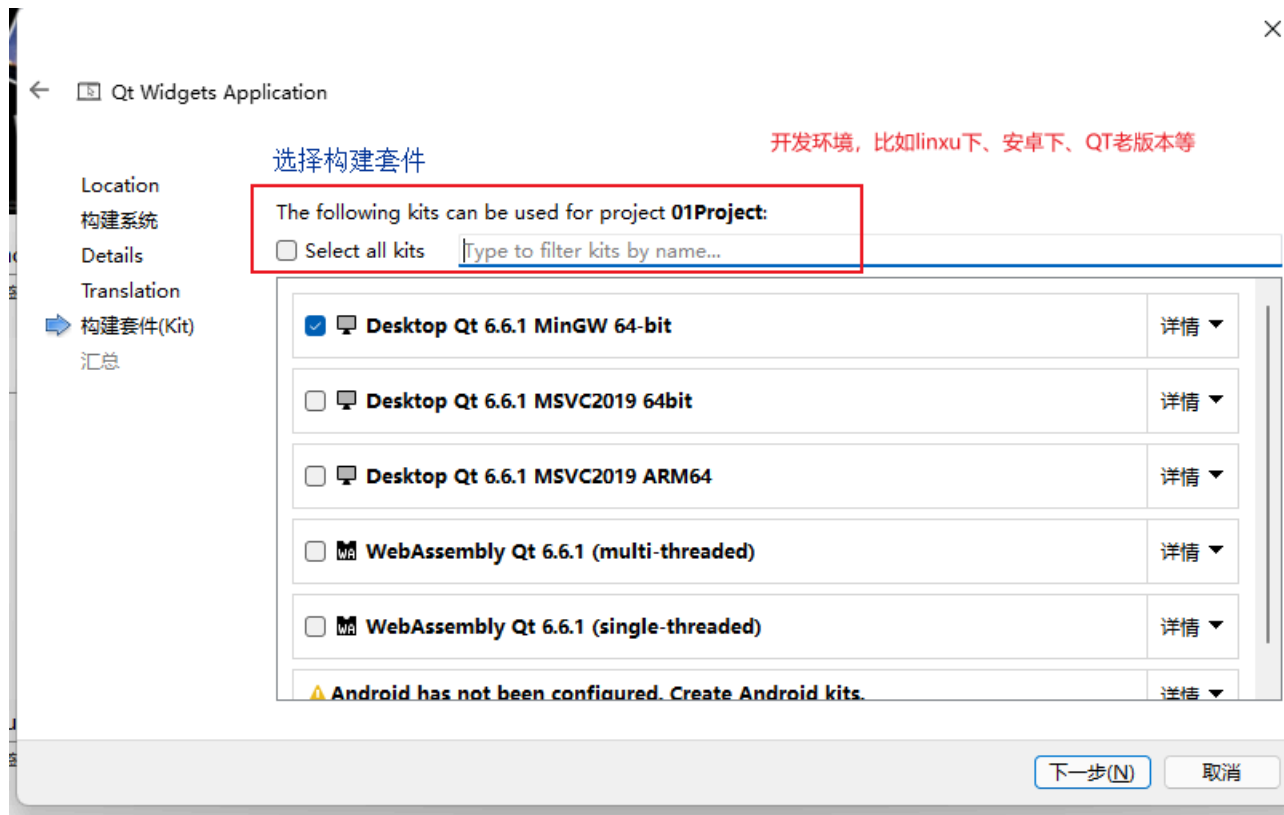


Figure 4

第七步，选择自己所需要版本控制。没有就不选。

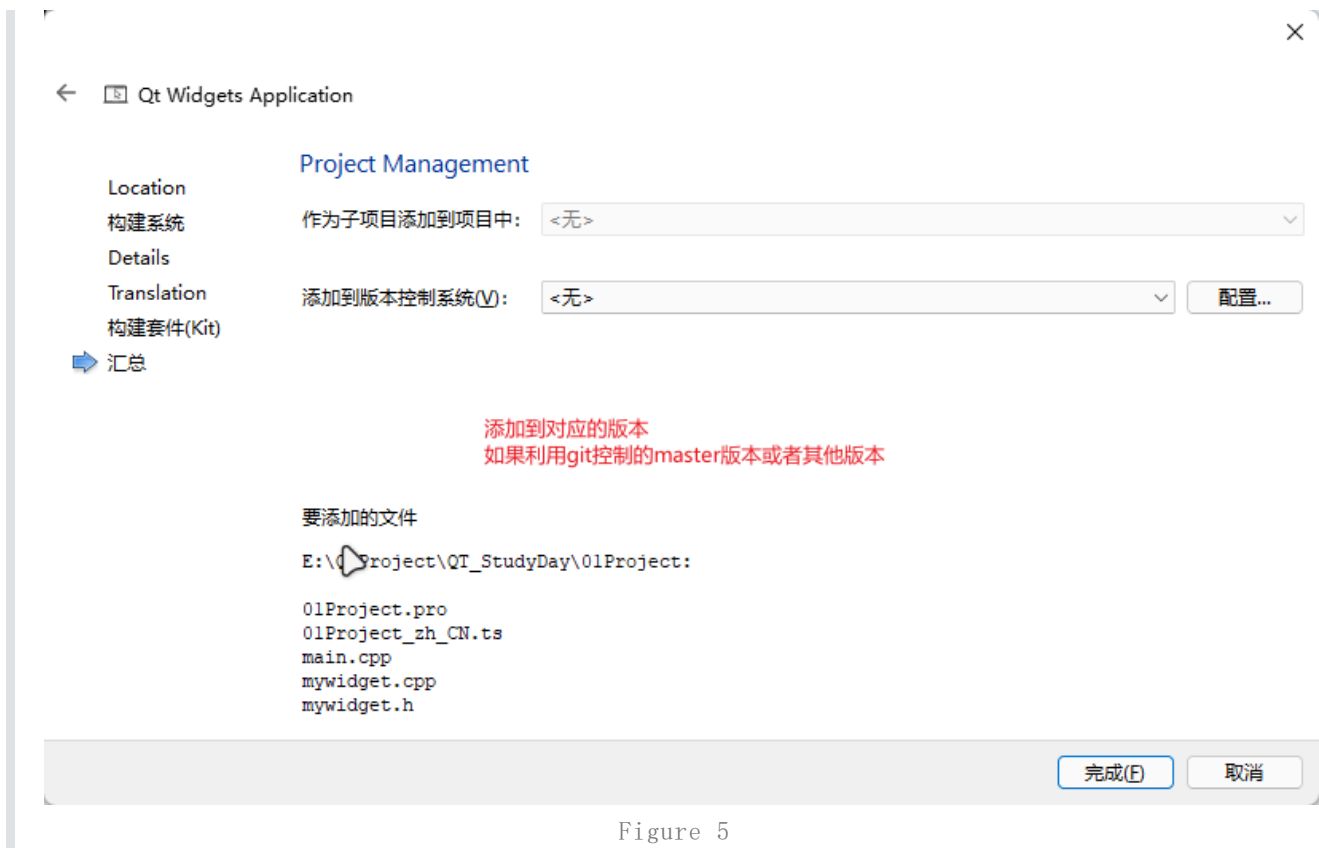


Figure 5

.pro 相应的工程文件（点击之后，能直接打开软件类似 keil5 的.uvprojx）

## 2.1 mian 程序解释

```

#include "mywidget.h"

#include <QApplication>
#include <QLocale>
#include <QTranslator>

/* 程序入口 argc: 命令行变量的数量 argv: 命令行变量数组 */
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);    //a应用程序对象，在Qt中，应用程序对象，有且只有一个

    QTranslator translator;
    const QStringList uiLanguages = QLocale::system().uiLanguages();
    for (const QString &locale : uiLanguages) {
        const QString baseName = "01Project_" + QLocale(locale).name();
        if (translator.load(":/i18n/" + baseName)) {
            a.installTranslator(&translator);
            break;
        }
    }

    MyWidget w;                    //窗口对象，MyWidget的父类-》QWidget
    w.show();                      //窗口对象 默认不会显示 需要调用show方法才能显示 类似keil5中江协OLED中的更新函数
    return a.exec();              //让应用程序对象进入消息循环机制 类似keil5的main中while的作用，不同的是，点击x会退出循环
}

```

I

Figure 6

```

1  #ifndef MYWIDGET_H
2  #define MYWIDGET_H
3
4  #include <QWidget>
5
6  class MyWidget : public QWidget
7  {
8  |   Q_OBJECT
9
10 public:
11 |   MyWidget(QWidget *parent = nullptr);
12 |   ~MyWidget();
13 };
14 #endif // MYWIDGET_H
15

```

I

Figure 7

### 3. 命名规范与快捷键

#### 3.1 pro 文件的解释（qmake 构建才有.pro 文件）

配置项	含义
QT += core gui	添加 Qt 模块 core：核心模块 gui：图形模块
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets	Qt 版本大于 4，添加 widgets 模块
CONFIG += c++17	经常用的值，release：以发布模式编译程序，debug：以调试模式编译程序，warn_on：编译器输出尽可能多的警告，C++11：启用 c++11 标准支持
TARGET	指定生成的可执行程序的名字
TEMPLATE	指定如何运行当前程序，默认值为 app（Application），表示当前程序是应用程序，可直接编译运行，常用值还有 lib，表示将当前程序编译成库文件
DEFINES	在程序中定义一个宏，如同 c 文件中的#define xxx
SOURCES	指定项目中的所有 cpp 源文件
HEADERS	指定项目中的所有 h 头文件
FORMS	指定项目的所有 ui 文件
INCLUDEPATH	指定引入外部资源的头文件路径
LIBS	指定引入外部资源的指定库文件
qnx: target.path = /tmp/\${TARGET}/bin	当前平台为QNX（嵌入式实时操作系统），路径就为 /tmp/\${TARGET}/bin
else: unix:! android: target.path = /opt/\${TARGET}/bin	当前平台是unix而不是android，路径就为/opt/\${TARGET}/bin
! isEmpty(target.path): INSTALLS += target	如果 target.path 不为空，那么就添加 target 这个规则

Table 1

```
1 QT      += core gui
2
3 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5 CONFIG += c++17
6
7 # You can make your code fail to compile if it uses deprecated APIs.
8 # In order to do so, uncomment the following line.
```



```

9  #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
   deprecated before Qt 6.0.0
10
11  SOURCES += \
12      main.cpp \
13      mywidget.cpp
14
15  HEADERS += \
16      mywidget.h
17
18  TRANSLATIONS += \
19      01Project_zh_CN.ts
20  CONFIG += lrelease
21  CONFIG += embed_translations
22
23  # Default rules for deployment.
24  qnx: target.path = /tmp/${TARGET}/bin
25  else: unix:!android: target.path = /opt/${TARGET}/bin
26  !isEmpty(target.path): INSTALLS += target

```

Fence 1

### 3.1.1 Qt 模块

Qt Widgets: Qt5 以及之后的版本，才有将 Widgets 模块独立，之前版本将 Widgets 放在 GUI 模块中。



Figure 8

## 3.2 头文件

```
1  #pragma once          //防止头文件重复声明 (C++中)
2
3  #ifndef __XX_H_       //也是防止头文件重复声明
4  #define __XX_H_
5      xxxx
6  #endif
7
```

Fence 2

```
1  /* Mywidget.h */
2  #ifndef MYWIDGET_H
3  #define MYWIDGET_H
4
5  #include <Qwidget>      //Qwidget 窗口类
6
7  class Mywidget : public Qwidget    //继承与Qwidget类
8  {
9      Q_OBJECT            //Q_OBJECT宏，允许类中使用信号和槽的机制
10
11  public:
12      Mywidget(Qwidget *parent = nullptr);    //有参构造函数
13      ~Mywidget();                            //析构函数
14  };
15  #endif // MYWIDGET_H
16
17
18
19  /* Mywidget.cpp */
20  #include "mywidget.h"
21
22  Mywidget::Mywidget(Qwidget *parent)
23      : Qwidget(parent)    //使用初始化列表，初始化Mywidge类
24  {}
25
26  Mywidget::~Mywidget() {}
```

Fence 3

## 3.3 命名规范

类名：首字母大写，单词与单词之间的首字母大写(大驼峰)

函数名、变量名：首字母小写、单词与单词之间首字母大写(小驼峰)

指针变量 -> ptr 指针名      eg. ptrstusql

成员变量 -> m\_成员变量名      eg. m\_age

### 3.4 快捷键

- 注释: CTRL+/
  - 运行: CTRL+r
  - 编译: CTRL+b
  - 字体缩放: CTRL+鼠标滚轮
  - 查找: CTRL+f
  - 整行移动: CTRL+↓ 或者 ↑
  - 帮助文档: 选中+F1
- 第一种: F1
- 第二种: 左侧帮助按钮
- 第三种: "E:\Qt\6.6.1\mingw\_64\bin\assistant.exe"
- 自动对齐: CTRL+i
- 同名之间的.h 和.cpp 切换: F4
- 选取列: alt+左键
- 快速将.h 中的函数在.cpp 中定义: alt+enter
- 重写虚函数不提示: 帮助-> 关于插件-> c++-> ClangCodeModel-> 取消勾选-> 重启 qt

## 4. 组件类

### 4.1 QPushButton(按钮控件)

#### 4.1.1 基本简介

Header:	#include
CMake:	find_package(Qt6 REQUIRED COMPONENTS Widgets) target_link_libraries(mytarget PRIVATE Qt6:: Widgets)
qmake:	QT += widgets
Inherits:	QAbstractButton
Inherited By:	QCommandLinkButton

Table 2

- **头文件:** `#include <QPushButton>` 这行代码用于在 C++源文件中包含 QPushButton 类的定义, 以便可以使用该类创建和操作按钮。
- **CMake 配置:**

- `find_package(Qt6 REQUIRED COMPONENTS widgets)` 这行代码用于在 CMake 项目中查找 Qt6 库，并指定需要使用 Widgets 模块。
- `target_link_libraries(mytarget PRIVATE Qt6::Widgets)` 这行代码则用于将 Qt6 的 Widgets 模块链接到名为 `mytarget` 的目标项目。
- **qmake 配置：** `QT += widgets` 这行代码用于在 qmake 项目文件 (.pro) 中指定需要使用 Qt 的 widgets 模块。
- **继承关系：** `QAbstractButton` 是 `QPushButton` 类的基类，这意味着 `QPushButton` 继承了 `QAbstractButton` 的所有公共和受保护的成员。
- **被继承关系：** `QCommandLinkButton` 类继承自 `QPushButton`，这意味着 `QCommandLinkButton` 类具备 `QPushButton` 类的所有功能，并且还可以在其基础上添加额外的功能。

### 4.1.2 信号

该信号全部继承于 `QAbstractButton` 类

- Signals:
  - `void clicked(bool checked = false)` //当按钮被点击时发出该信号。如果按钮是可选中的，`checked` 表示当前的选中状态。
  - `void pressed()` //当按钮被按下时发出该信号。
  - `void released()` //当按钮被释放时发出该信号（即用户松开按钮时触发）。
  - `void toggled(bool checked)` //当按钮的选中状态改变（切换）时发出该信号，`checked` 表示新的选中状态。

### 4.1.3 槽函数

- public Slots:
- 继承于 `QAbstractButton` 类
  - `void animateClick()` //触发按钮的点击动画默认持续时间（通常为 100 ms），并会触发按钮的 `clicked()` 信号。
  - `void click()` //程序模拟按钮被点击的动作，会触发 `clicked()` 信号，但不会有动画。
  - `void setChecked(bool)` //设置按钮的选中状态，仅适用于可切换的按钮（如 `QCheckBox` 或 `QRadioButton`）。
  - `void setIconSize(const QSize &size)` //设置按钮图标的显示尺寸。
  - `void toggle()` //切换按钮的选中状态（从选中到未选中或从未选中到选中）。
- 自身的槽函数
  - `void showMenu();` //用于显示与按钮关联的弹出菜单（`QMenu`）

#### 4.1.4 示例

```
1 //mywidget.h
2 #include <QPushButton> //QPushButton 按钮类
3
4
5 //mywidget.cpp
6 #include "mywidget.h"
7
8
9 MyWidget::MyWidget(QWidget *parent)
10 : QWidget(parent)
11 {
12     /* 第一种重载 */
13     QPushButton *btn = new QPushButton; //创建在堆区的按钮指针
14     // btn->show(); //show()方法是以顶层的形式创建按钮（新创建一个窗口）
15     btn->setParent(this); //btn对象 绑定父组件（依赖于mywidget窗口中）
16
17     btn->setText("这是第一个按钮"); //按钮上显示文本 设置按钮的属性一般为set...();
18
19     /* 第二种重载 */
20     QPushButton *btn01 = new QPushButton("这是第二个按钮", this);
21     //弊端：按照控件大小创建窗口
22     //新创建的控件会覆盖原控件，需要调整位置
23
24     /* 移动控件位置 */
25     btn->move(100,100);
26
27     /* 设置控件窗口标题 */
28     this->setWindowTitle("按钮");
29
30     /* 重置窗口初始大小 */
31     resize(500,400);
32
33     /* 固定窗口大小 */
34     this->setFixedSize(500,400);
35
36     /* 重置按钮大小 */
37     btn01->resize(100,50);
38
39
40
41 }
```

Fence 4

- 注意：在 Qt 中，如果你通过 `setParent(this)` 将按钮设置为某个父组件（例如 `mywidget`）的子组件，那么你通常不需要手动删除这个按钮对象。Qt 的内存管理机制会自动处理这种情况：当父组件被销毁时，所有子组件也会被自动销毁。但是，如果你在没有设置父组件的情况下创建了一个按钮对象，并且希望在某个时刻手动删除它，那么你需要调用 `delete` 来释放内存。

- 查看编码格式: 工具->外部->配置->文本编辑器->行为->编码格式

## 4.2 QUdpSocket(网络控件)

### 4.2.1 基本简介

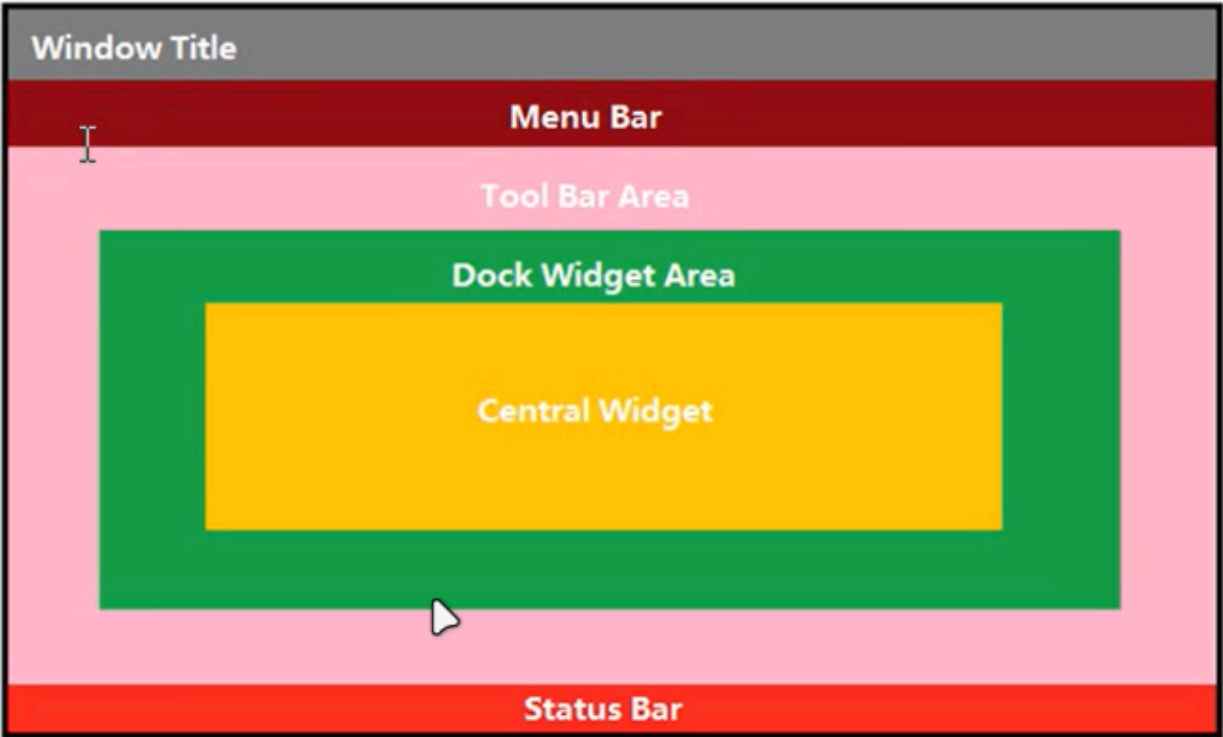
Header:	#include
CMake:	find_package(Qt6 REQUIRED COMPONENTS Network) target_link_libraries(mytarget PRIVATE Qt6::Network)
qmake:	QT += network
Inherits:	QAbstractSocket

Table 3

- **头文件:** `#include <QUdpSocket>` 用于在项目中包含 QUdpSocket 类的定义。
- **CMake 配置:**
  - `find_package(Qt6 REQUIRED COMPONENTS Network)` 用于查找 Qt6 库中的 Network 模块, 该模块包含 QUdpSocket 等网络相关的类。
  - `target_link_libraries(mytarget PRIVATE Qt6::Network)` 用于将 Qt6 的 Network 模块链接到你的项目目标 `mytarget`, 以便在项目中使用 QUdpSocket 类。
- **qmake 配置:** `QT += network` 用于在 qmake 项目文件.pro 中添加 Network 模块, 使 QUdpSocket 类可以被项目使用。
- **继承关系:** QUdpSocket 继承自 QAbstractSocket, 这意味着 QUdpSocket 类具有 QAbstractSocket 类的所有功能, 并且可以用于处理 UDP 套接字的特定需求。

### 4.3 QMainWindow(主窗口类)

- **QMainWindow** 是一个为用户提供主窗口程序的类，包含一个菜单栏 (**menu bar**)、多个工具栏 (**tool bars**)、多个锚接部件 (**dock widgets**)、一个状态栏 (**status bar**) 及一个中心部件 (**central widget**)，是许多应用程序的基础，如文本编辑器、图片编辑器等。



#### 4.3.1 基本简介

Header:	#include
CMake:	find_package(Qt6 REQUIRED COMPONENTS Widgets) target_link_libraries(mytarget PRIVATE Qt6:: Widgets)
qmake:	QT += widgets
Inherits:	QWidget

Table 4

- **头文件:** `#include <QMainWindow>` 用于在项目中包含 QMainWindow 类的定义，使得可以创建主窗口应用程序。
- **CMake 配置:**
  - `find_package(Qt6 REQUIRED COMPONENTS Widgets)` 用于查找 Qt6 库中的 Widgets 模块，该模块提供了构建图形用户界面的基础类。
  - `target_link_libraries(mytarget PRIVATE Qt6::Widgets)` 用于将 Qt6 的 Widgets 模块链接到你的项目目标 `mytarget`，以便在项目中使 用 QMainWindow 及其他相关类。
- **qmake 配置:** `QT += widgets` 用于在 qmake 项目文件.pro 中添加 Widgets 模块，使 QMainWindow 类可以被项目使用。

- **继承关系**: QMainWindow 继承自 QWidget, 这意味着 QMainWindow 类拥有 QWidget 类的所有功能, 并且提供更多用于构建应用程序主窗口的特性, 如工具栏、状态栏和菜单栏等。

### 4.3.2 菜单栏(一个窗口最多一个)

```
1 //窗口菜单栏
2     QMenuBar *bar = menuBar();
3 //将菜单栏放入窗口中
4 setMenuBar(bar);
5
6 //创建菜单
7 QMenu *fileMenu = bar->addMenu("文件");
8 QMenu *editMenu = bar->addMenu("编辑");
9
10 //创建菜单项
11 fileMenu->addAction("新建");
12 fileMenu->addAction("打开");
13
14 //创建分隔线
15 fileMenu->addSeparator();
16
17 fileMenu->addAction("退出");
18
19
```

Fence 5

### 4.3.3 工具栏(一个窗口可以有多个)

```
1 /* 工具栏 可以有多个 */
2 //创建工具栏 QMainWindow中没有实现工具栏的方法
3 QToolBar *toolBar = new QToolBar(this);
4 //添加工具栏 设定工具生成的初始位置
5 addToolBar(Qt::LeftToolBarArea,toolBar);
6
7 //设置可以移动访问 只能停在左边或者右边
8 toolBar->setAllowedAreas(Qt::LeftToolBarArea | Qt::RightToolBarArea);
9
10 //设置是否浮动 (如果停靠的边不合规, 将放回上一次的位置)
11 toolBar->setFloatable(false);
12
13 //设置能否移动 (移动总开关)
14 toolBar->setMovable(true);
15
16 //创建工具项
17 toolBar->addAction("运行");
18 toolBar->addSeparator(); //添加分割线
19 toolBar->addAction(newAct); //和菜单栏的新建 与 打开 共用一套数据
20 toolBar->addAction(openAct);
21
22 //添加控件
23 QPushButton *btn = new QPushButton("你好",toolBar);
```



```
24 | toolbar->addWidget(btn);
```

Fence 6

#### 4.3.4 状态栏(最多只有一个)

```
1 | /* 状态栏 最多只有一个 */
2 | //创建状态栏 有成员函数可以实现
3 | QStatusBar *stb = statusBar();
4 | //放入窗口
5 | setStatusBar(stb);
6 |
7 | //添加标签
8 | QLabel *lab = new QLabel("提示信息",this);
9 | QLabel *lab1 = new QLabel("提示信息2",this);
10 |
11 | //默认左侧
12 | stb->addWidget(lab);
13 | //设置在右侧
14 | stb->addPermanentWidget(lab1);
```

Fence 7

#### 4.3.5 铆接部件(浮动控件 可以有多个) 和 核心控件(只能有一个)

```
1 | /* 铆接部件（浮动控件） 可以有多个 核心部件 只能有一个*/
2 | //创建铆接部件
3 | QDockWidget *dow = new QDockWidget("浮动",this);
4 | //添加到窗口 同时指定位置(位置是以核心部件为中心)
5 | addDockWidget(Qt::BottomDockWidgetArea,dow);
6 |
7 | //固定位置移动范围 只能上或者下
8 | dow->setAllowedAreas(Qt::BottomDockWidgetArea | Qt::TopDockWidgetArea);
9 |
10 | //创建文本部件
11 | QTextEdit *te = new QTextEdit(this);
12 | //添加成核心部件
13 | setCentralWidget(te);
```

Fence 8

小技巧：只能有一个的用 set 添加 可以有多个用 add 添加 父对象提供实现的成员函数，用成员函数 没有提供用 new

## 4.4 对话框(QDialog)

标准对话框，是 Qt 内置的一系列对话框，用于简化开发。实际上，有很多对话框都是通用的，比如打开文件、设置颜色、打印设置等。这些对话框在所有程序中几乎都相应，因此没有必要在每一个程序中都实现这一个对话框。

Qt 的内置对话框大致分为以下几类：

- **QColorDialog**: 选择颜色；
- **QFileDialog**: 选择文件或目录；
- **QFontDialog**: 选择字体；
- **QInputDialog**: 允许用户输入一个值，并将其返回；
- **QMessageBox**: 消息对话框，用于显示信息、询问问题等；
- **QPageSetupDialog**: 为打印机提供纸张相关的选项；
- **QPrintDialog**: 打印机设置；
- **QPrintPreviewDialog**: 打印预览；
- **QProgressDialog**: 显示操作进程。

### 4.4.1 模态对话框(打开以后，不能操作其他窗口——阻塞其他窗口)——自定义对话框

```
1 //连接新建触发的信号 与 创建窗口
2 connect(ui->actionNew,&QAction::triggered,[=]() {
3     //模态对话框 会阻塞原窗口进程
4     //阻塞了进程 生命周期暂停 点击x的时候才释放 不要new
5     QDialog dlg(this);
6     dlg.setWindowTitle("新建");
7     dlg.exec();
8
9     qDebug() << "新窗口创建了" ;
10
11 });
12
```

Fence 9

### 4.4.2 非模态对话框(打开以后，能操作其他窗口——不阻塞其他窗口)——自定义对话框

```
1 connect(ui->new2_2,&QAction::triggered,[=]() {
2     //非模态对话框 不会阻塞原窗口进程
3     //需要创建到堆区 才能保证窗口不会只出现一瞬间 释放在父对象x掉以后释放
4     QDialog *dlg = new QDialog(this);
5     dlg->setWindowTitle("新建");
6     dlg->setAttribute(Qt::WA_DeleteOnClose); // 值为55 保证点击关闭时，释放小控
7     件内存 如果不做这个处理，连续的创建会导致内存泄露
8     dlg->show();
9
10     qDebug() << "新窗口创建了" ;
11 });
```

底层封装的函数为静态函数（不需要对象，能直接调用）

```
1 connect(ui->actionNew,&QAction::triggered,[=]() {
2     /* 消息对话框 */
3     //错误对话框
4     // QMessageBox::critical(this,"错误提示框","xxx内容错误");
5
6     //信息提示框
7     // QMessageBox::information(this,"信息提示框","xxx内容");
8
9     //提问提示框
10    //参数1 父对象    参数2 窗口标题    参数3 提示内容    参数4 按键内容    参数5 默认关联回车
    的按键类型
11    //返回值为StandardButton 用于确定返回值类型
12    // QMessageBox::question(this,"提问提示框","是否取消");
13    //QMessageBox::question(this,"提问提示框","是否保存",QMessageBox::Save |
    QMessageBox::Cancel,QMessageBox::Cancel);
14    // if(QMessageBox::Save == QMessageBox::question(this,"提问提示框","是否保
    存",QMessageBox::Save | QMessageBox::Cancel,QMessageBox::Cancel))
15    // {
16    //     qDebug() << "保存";
17    // }
18
19
20    //警告提示框
21    // QMessageBox::warning(this,"警告提示框","xxx警告");
22
23
24
25    //自定义提示框
26    QMessageBox qmb(this);
27    qmb.setWindowTitle("自定义提示框");
28    qmb.setText("是否保存");
29    // qmb.setIcon(); //设置图标
30
31    //添加自定义按钮
32    QPushButton *p1 = qmb.addButton("保存",QMessageBox::AcceptRole);
33    QPushButton *p2 = qmb.addButton("取消",QMessageBox::RejectRole);
34
35    qmb.setDefaultButton(p2); //回车关联取消
36    qmb.exec(); //显示并设置成模态
37
38    if(qmb.clickedButton() == p1){
39        qDebug() << "保存";
40    }
41
42
43    });
44
45
```

#### 4.4.4 其他对话框

```

1      connect(ui->actionNew,&QAction::triggered,[=]() {
2          //其他对话框
3          //颜色对话框
4          // QColor color = QColorDialog::getColor(QColor(255,55,0));
//QColor()还有第四个参数,为透明度
5          // qDebug() << "r=" << color.red() << "g=" << color.green() << "b=" <<
color.blue();
6          // qDebug() << "html =" << color.name();
7
8          //文本对话框
9          //参数1 父对象 参数2 窗口标题 参数3 文件路径(需要转义字符\ 参数4 过滤文本格式
10         // QString fileName = QFileDialog::getOpenFileName(this,"打开文件","E:\\笔记","
(*.txt)");
11         // qDebug() << fileName;
12
13         //字体对话框
14         bool flag;          //是否选择字体
15         QFont font = QFontDialog::getFont(&flag,QFont("宋体",20,1),this,"字体选择");
16         qDebug() << "字体: " << font.family().toUtf8().data() << "字号: " <<
font.pointSize() << "是否加粗: " << font.bold() << "是否倾斜:" << font.italic();
17         qDebug() << flag;
18     });

```

## 5. 对象树

- 在 Qt 中创建对象的时候会传递一个 Parent 对象指针，下面来解释这个 parent 到底是子什么的。
  - QObject 是以对象树的形式组织起来的。
    - 当前创建一个 QObject 对象时，会看到 QObject 的构造函数接收一个 QObject 指针作为参数，这个参数就是 parent，也就是父对象指针。
    - 这样当于，在创建 QObject 对象时，可以提供一个其父对象。我们创建的这个 QObject 对象会自动添加到其父对象的 children() 列表。
  - 当父对象析构的时候，这个列表中的所有对象也会被析构。（注意，**这里的父对象并不是继承意义上的父类！**）
    - 这种机制在 GUI 程序设计中相当有用。例如，一个按钮有一个 QShortcut（快捷键）对象作为其子对象。当我们删除按钮的时候，这个快捷键是应该删除。这是合理的。
    - 析构时，先调用自己（最大的父对象）的析构函数，再看有无子对象，有调用子对象的析构函数，子对象再检查自己有无子对象；无释放自己。（内存释放顺序为孙子对象-》子对象-》父对象。析构调用顺序为父对象-》子对象-》孙子对象）。
    - 对象放入对象树中，自动释放内存，——》一定程度简化了内存回收机制。

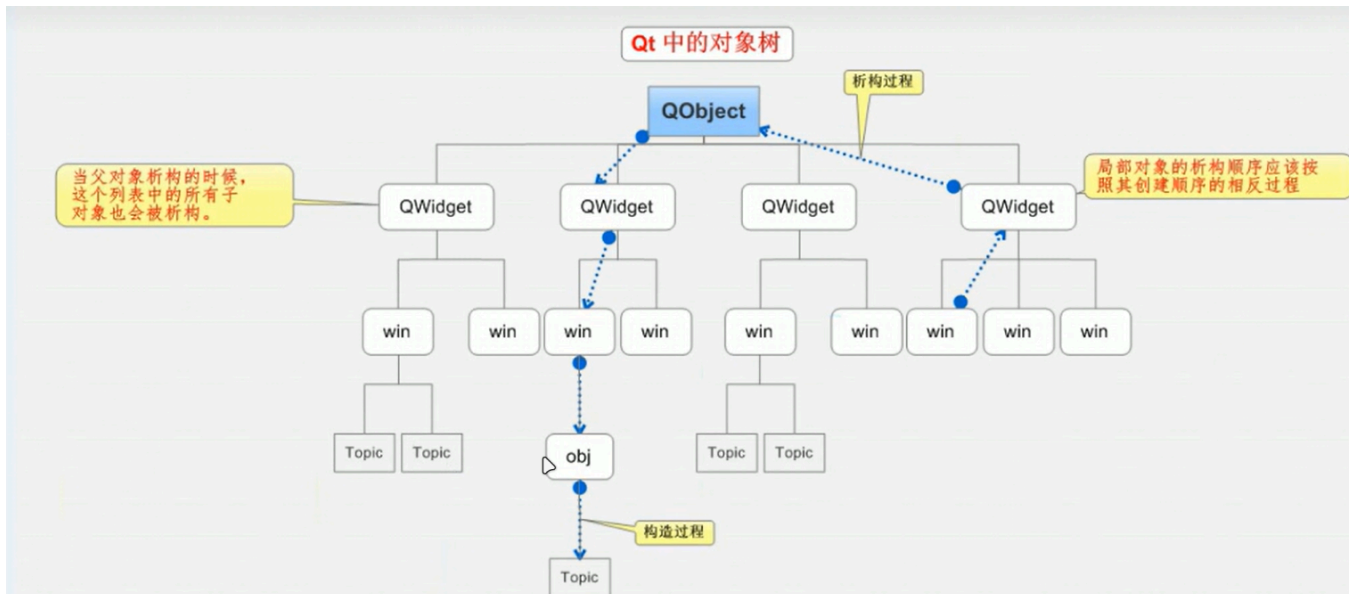


Figure 9

- QWidget 是能够定屏幕上显示的一切图形化的父类。
  - QWidget 继承自 QObject，因此也继承了这种对象关系链。一个子对象自动地成为父组件的一个子组件。因此，它会显示在父组件的坐标系之中，覆盖父组件的起示对象。例如，当用户关闭这个窗体对象，应用程序应该关闭那个控件，那么，我们现在只需要关闭一个父对象，这样这个控件就会被删除了。因为这些对象是及其相关的。

## 5.1 示例

```

1  //mypushbutton.h
2  #ifndef MYPUSHBUTTON_H
3  #define MYPUSHBUTTON_H
4
5  #include <QPushButton>
6  #include <QDebug>                                //控制台打印
7
8  class MyPushBotton : public QPushButton
9  {
10 public:
11     explicit MyPushBotton(QWidget *parent = nullptr);    //explicit 声明构造
12     函数
13     ~MyPushBotton();
14
15 signals:
16 };
17
18 #endif // MYPUSHBUTTON_H
19
20
21 //mypushbutton.cpp
22 #include "mypushbutton.h"
23

```

```

24  MyPushBotton::MyPushBotton(QWidget *parent)
25      : QPushButton{parent}                                //QPushButton{parent} 传入父类进行
构造
26  {
27
28      qDebug() << "MyPushBotton的构造函数";
29  }
30
31  MyPushBotton::~MyPushBotton()
32  {
33      qDebug() << "MyPushBotton的析构函数";
34  }
35  }
36
37
38  //mypushbutton.cpp
39  #include "mywidget.h"
40  #include "mypushbutton.h"
41  #include <QDebug>
42
43
44  MyWidget::MyWidget(QWidget *parent)
45      : QWidget(parent)                                    //使用初始化列表，初始化Mywidge类
46  {
47      /* 验证对象树的构造和析构过程 */
48      MyPushBotton *myBtn = new MyPushBotton();
49      myBtn->setParent(this);                               //绑定父对象为Mywidget
50
51      myBtn->setText("自己的按钮");
52
53  }
54
55  MyWidget::~MyWidget() {
56      qDebug() << "Mywidget的析构函数";
57  }
58

```

## 5.2 qt 窗口坐标系

坐标体系：

以左上角为原点  $(0,0)$ ，X 向右增加，Y 向下增加。

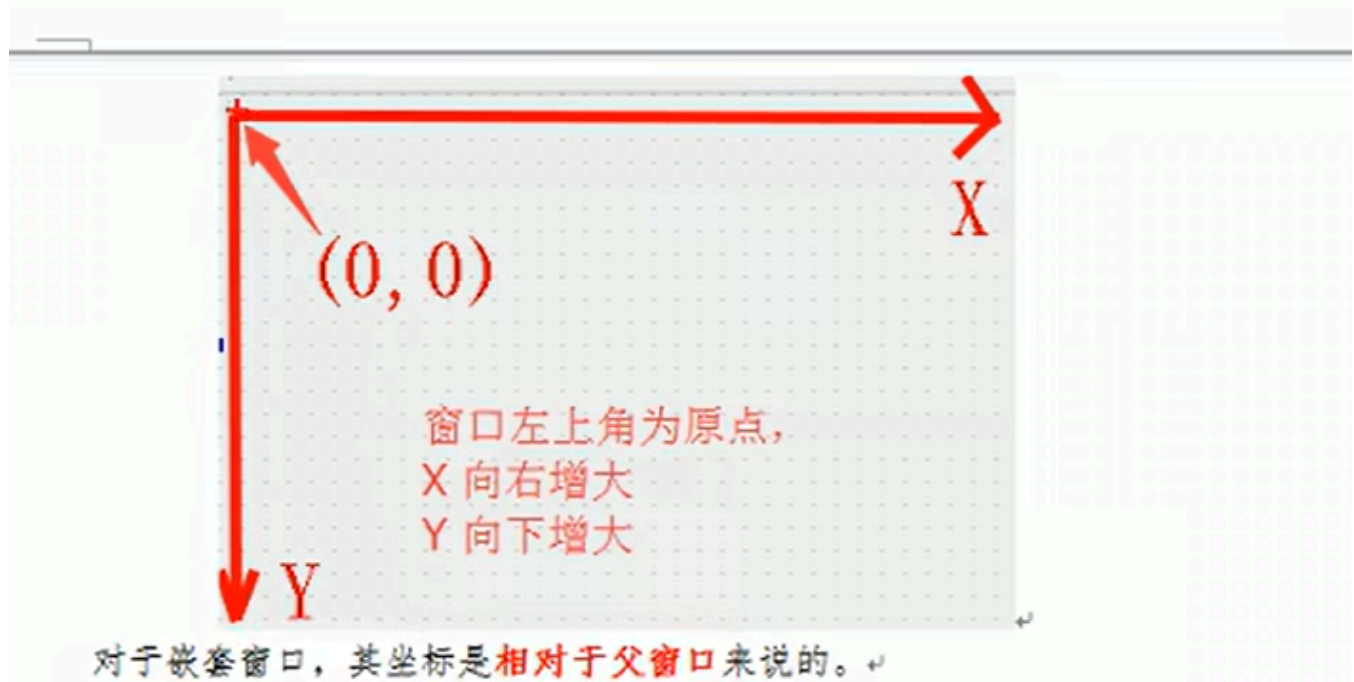


Figure 10

## 6. 信号和槽

### 6.1 基本概念

- 本质：信号-》事件 槽-》处理事件的方法或者函数 二者通过一定关系连接
- 优点：松散耦合。（信号发送方 与 信号处理方 本身没有联系 通过 connect 连接在一起）
- 使用：
  - `connect(信号发送方, 发送的信号（给函数的地址）, 信号接受方, 处理的槽函数（给函数地址）);`
  - eg. `connect(myBtn,&MyPushButton:: clicked, this,&MyWidget:: close);` // MyPushButton 可以换成继承的父类

## 6.2 技巧

- 使用 Qt 助手查找时，注意 Signals（信号）和 Public Slots（槽），如果没有找一下最后的函数，看一下继承与什么父类，在通过父类查找。

### Public Slots

```
void animateClick()
void click()
void setChecked(bool)
void setIconSize(const QSize &size)
void toggle()
```

### Signals

```
void clicked(bool checked = false)
void pressed()
void released()
void toggled(bool checked)
```

Figure 11

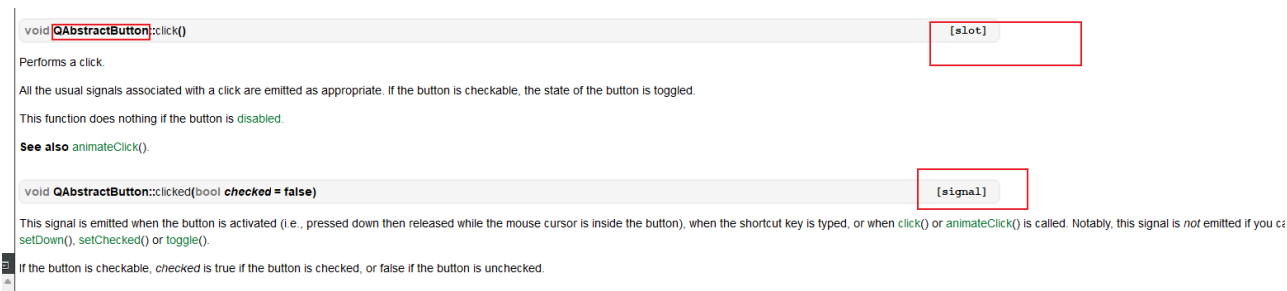


Figure 12

## 6.3 自定义信号和槽

- 自定义信号
  - 需要放到类中的 signals 下面
  - 无返回值 需要声明 不需要实现
  - 可有参数 可重载
- 自定义槽
  - 需要放到类中的 public 或者 public slots（低版本必须放在 public slots）下 可以私有 private slots
  - 无返回值 需要声明 需要实现
  - 可有参数 可重载
- 自定义触发
  - emit 关键字 调用信号 一般封装到函数中
  - 信号连接信号 触发信号



示例:

```
1 //student.h
2 #ifndef STUDENT_H
3 #define STUDENT_H
4
5 #include <QObject>
6
7 class Student : public QObject
8 {
9     Q_OBJECT
10 public:
11     explicit Student(QObject *parent = nullptr);
12
13     /* 槽函数 */
14     //有参 无返回 需实现 可重载
15     void dining();
16
17
18 signals:
19 };
20
21 #endif // STUDENT_H
22
23
24 //student.cpp
25 #include "student.h"
26 #include <QDebug>
27
28 Student::Student(QObject *parent)
29     : QObject{parent}
30 {}
31
32 void Student::dining(){
33
34     qDebug() << "冲出教室 吃饭! ";
35 }
36
37
38 //teacher.h
39 #ifndef TEACHER_H
40 #define TEACHER_H
41
42 #include <QObject>
43
44 class Teacher : public QObject
45 {
46     Q_OBJECT
47 public:
48     explicit Teacher(QObject *parent = nullptr);
49
50 signals:
51     // 有参无返回 不需要实现 可重载
```

```

52     void classIsOver();
53
54 };
55
56 #endif // TEACHER_H
57
58
59 //teacher.cpp
60 #include "teacher.h"
61
62 Teacher::Teacher(QObject *parent)
63     : QObject{parent}
64 {}
65
66
67 //widget.h
68 #ifndef WIDGET_H
69 #define WIDGET_H
70
71 #include <QWidget>
72 #include "student.h"
73 #include "teacher.h"
74
75 class widget : public QWidget
76 {
77     Q_OBJECT
78
79 public:
80     widget(QWidget *parent = nullptr);
81     ~widget();
82
83 private:
84     void By12();
85
86     Student *m_st;
87     Teacher *m_th;
88 };
89 #endif // WIDGET_H
90
91
92 //widget.cpp
93 #include "widget.h"
94
95
96 //12:00(触发条件)    老师 发出 下课信号        学生 冲向食堂
97
98 widget::widget(QWidget *parent)
99     : QWidget(parent)
100 {
101     //创建对象 同时将父对象告知
102     this->m_th = new Teacher(this);
103     this->m_st = new Student(this);
104

```

```

105     connect(this->m_th,&Teacher::classIsOver,this->m_st,&Student::dining);
106
107     this->By12();
108     //emit this->m_th->classIsOver();
109 }
110
111 void widget::By12()
112 {
113     //触发信号
114     emit this->m_th->classIsOver();
115
116 }
117
118 widget::~~widget() {}
119

```

Fence 14

注：先连接，在触发，才能生效

## 6.4 自定义信号和槽重载问题解决

- 连接时，使用函数指针 void(指针名) (参数列表) = 函数地址

示例：

```

1  //student.h
2  #ifndef STUDENT_H
3  #define STUDENT_H
4
5  #include <QObject>
6
7  class Student : public QObject
8  {
9      Q_OBJECT
10 public:
11     explicit Student(QObject *parent = nullptr);
12
13     /* 槽函数 */
14     //有参 无返回 需实现 可重载
15     void dining();
16     //重载
17     void dining(QString food);
18
19
20 signals:
21 };
22
23 #endif // STUDENT_H

```

```

24
25
26 //student.cpp
27 #include "student.h"
28 #include <QDebug>
29
30 Student::Student(QObject *parent)
31     : QObject{parent}
32 {}
33
34 void Student::dining(){
35
36     qDebug() << "冲出教室 吃饭! ";
37 }
38 void Student::dining(QString food)
39 {
40
41     //QString类型打印会出现""
42     //解决"" 先转换成QByteArray 在使用QByteArray的成员函数data()转换成char *
43     qDebug() << "冲出教室 吃饭! 吃:" << food.toUtf8().data();
44     //先转换成std::string 在换成 c_str
45     qDebug() << "冲出教室 吃饭! 吃:" << food.toStdString().c_str();
46
47 }
48
49 //teacher.h
50 #ifndef TEACHER_H
51 #define TEACHER_H
52
53 #include <QObject>
54
55 class Teacher : public QObject
56 {
57     Q_OBJECT
58 public:
59     explicit Teacher(QObject *parent = nullptr);
60
61 signals:
62     // 有参无返回 不需要实现 可重载
63     void classIsOver();
64     //重载
65     void classIsOver(QString food);
66 };
67
68 #endif // TEACHER_H
69
70
71 //teacher.cpp
72 #include "teacher.h"
73
74 Teacher::Teacher(QObject *parent)
75     : QObject{parent}
76 {}

```

```

77
78
79 //widget.h
80 #ifndef WIDGET_H
81 #define WIDGET_H
82
83 #include <QWidget>
84 #include "student.h"
85 #include "teacher.h"
86
87 class Widget : public QWidget
88 {
89     Q_OBJECT
90
91 public:
92     Widget(QWidget *parent = nullptr);
93     ~Widget();
94
95 private:
96     void By12();
97
98     Student *m_st;
99     Teacher *m_th;
100 };
101 #endif // WIDGET_H
102
103
104 //widget.cpp
105 #include "widget.h"
106
107
108 //12:00(触发条件)    老师 发出 下课信号    学生 冲向食堂
109
110 Widget::Widget(QWidget *parent)
111     : QWidget(parent)
112 {
113     //创建对象 同时将父对象告知
114     this->m_th = new Teacher(this);
115     this->m_st = new Student(this);
116
117     //出现重载版本后 编译器不知道用哪一个信号和槽
118     //使用函数指针 void(指针名) (参数列表) = 函数地址
119     void(Teacher:: *teacher)(QString) = &Teacher::classIsOver;
120     void(Student:: *student)(QString) = &Student::dining;
121     connect(this->m_th, teacher, this->m_st, student);
122
123     this->By12();
124 }
125
126 void Widget::By12()
127 {
128     //触发信号
129     emit this->m_th->classIsOver();

```

```

130
131     }
132
133     widget::~~widget() {}
134

```

Fence 15

## 6.5 拓展

- 信号链（信号连接信号）  
connect(发送方 1, 发送信号 1, 发送方 2, 发送信号 2);  
connect(发送方 2, 发送信号 2, 接受方, 槽函数);
- 断开信号  
disconnect(发送方, 发送信号, 接受方, 槽函数); //参数和 connect 一样
- 一个信号 连接 多个槽
- 多个信号 连接 同一个槽
- 信号和槽的参数 类型必须一一对应
- 信号和槽的参数 数量可以不对应 但只能信号的参数个数 **多于** 槽的参数个数（且信号参数列表中存在槽的所有参数类型）
  - clicked () 参数列表只有 bool 所以不能直接连接一个及以上参数的槽函数 可以通过自定义函数进行间接连接 或者 Lambda 表达式
- Qt4 版本之前的连接方式  
connect(发送方, SIGNAL(信号函数()), 接受方, SLOT(槽函数())); //缺点：不会做类型检测，会在运行后才能报错 **不推荐**

示例：

```

1 //同上
2 widget::widget(QWidget *parent)
3     : QWidget(parent)
4 {
5     //创建对象 同时将父对象告知
6     this->m_th = new Teacher(this);
7     this->m_st = new Student(this);
8
9     //出现重载版本后 编译器不知道用哪一个信号和槽
10    //使用函数指针 void(指针名) (参数列表) = 函数地址
11    void(Teacher:: *teacher)(QString) = &Teacher::classIsOver;
12    void(Student:: *student)(QString) = &Student::dining;
13    connect(this->m_th, teacher, this->m_st, student);
14
15    this->By12();
16    //emit this->m_th->classIsOver();
17

```

```

18 //信号连接信号
19 QPushButton *btn = new QPushButton("12: 00",this);
20
21 void(Teacher:: *teacher1)(void) = &Teacher::classIsOver;
22 void(Student:: *student1)(void) = &Student::dining;
23 connect(this->m_th,teacher1,this->m_st,student1);
24 connect(btn,&QPushButton::clicked,this->m_th,teacher1);
25
26 //断开信号
27 disconnect(btn,&QPushButton::clicked,this->m_th,teacher1);
28 }
29

```

Fence 16

## 6.6 Lambda 表达式（用于一个信号连接多个槽函数 或者 多个信号连接一个槽函数）

- 匿名函数 或者 闭包
- 用于定义并创建匿名的函数对象
- 低版本使用需要在 .pro 文件中 加 CONFIG += c++11
- [函数对象参数]（操作符重载函数参数）mutable -> 返回值类型{函数体}
  - 后面变成 最后加上()变为调用
  - 函数对象参数（捕获参数）形式：
    - 空：没有使用任何函数对象参数。
    - [=]：函数体内可以使用 Lambda 所在作用范围内所有可见的局部变量（包括 Lambda 所在类的 this），并且是 **值传递方式**（相当于编译器隐式为我们按值传递了所有局部变量）。
    - [&]：函数体内可以使用 Lambda 所在作用范围内所有可见的局部变量（包括 Lambda 所在类的 this），并且是 **引用传递方式**（相当于编译器隐式为我们按引用传递了所有局部变量）。
    - [this]：函数体内可以使用 Lambda 所在类中的成员变量。引用捕获
    - [\*this]：值捕获
    - [a]：将 a 按值传递传递。按值传递传递，函数体内不能修改传递进来的 a 的拷贝，可以标记该传参 const 的。**要修改传递进来的 a 的拷贝，可以添加 mutable 修饰符。**
    - [&a]：将 a 按引用传递。
    - [a, &b]。将 a 按值进行传递，b 按引用进行传递。
    - [=, &a, &b]。除 a 和 b 按引用进行传递外，其他参数都按值进行传递。
    - [&, a, b]。除 a 和 b 按值进行传递外，其他参数都按引用进行传递。
  - 操作符重载函数参数：标识重载的操作符的参数，没有参数时，这部分可以省略。参数可以通过按值（如：(a, b)）和按引用（如：(&a, &b)）两种方式进行传递。
  - 可修改标示符：mutable 关键词，这部分可以省略。按值传递函数对象参数时，加上 mutable 修饰符后，可以修改按值传递进来的拷贝（注意是能修改拷贝，而不是原本的参数值）。

```

1 QPushButton * myBtn = new QPushButton(this);
2 QPushButton * myBtn2 = new QPushButton(this);
3 myBtn2->move(100, 100);
4
5 int m = 10;
6
7 //mutable 修改拷贝值，而不会改变原来的值
8 connect(myBtn, &QPushButton::clicked, this, [m]() mutable { m = 100 + 10;
  qDebug() << m; });
9
10 connect(myBtn2, &QPushButton::clicked, this, [=] { qDebug() << m; });
11
12 qDebug() << m;

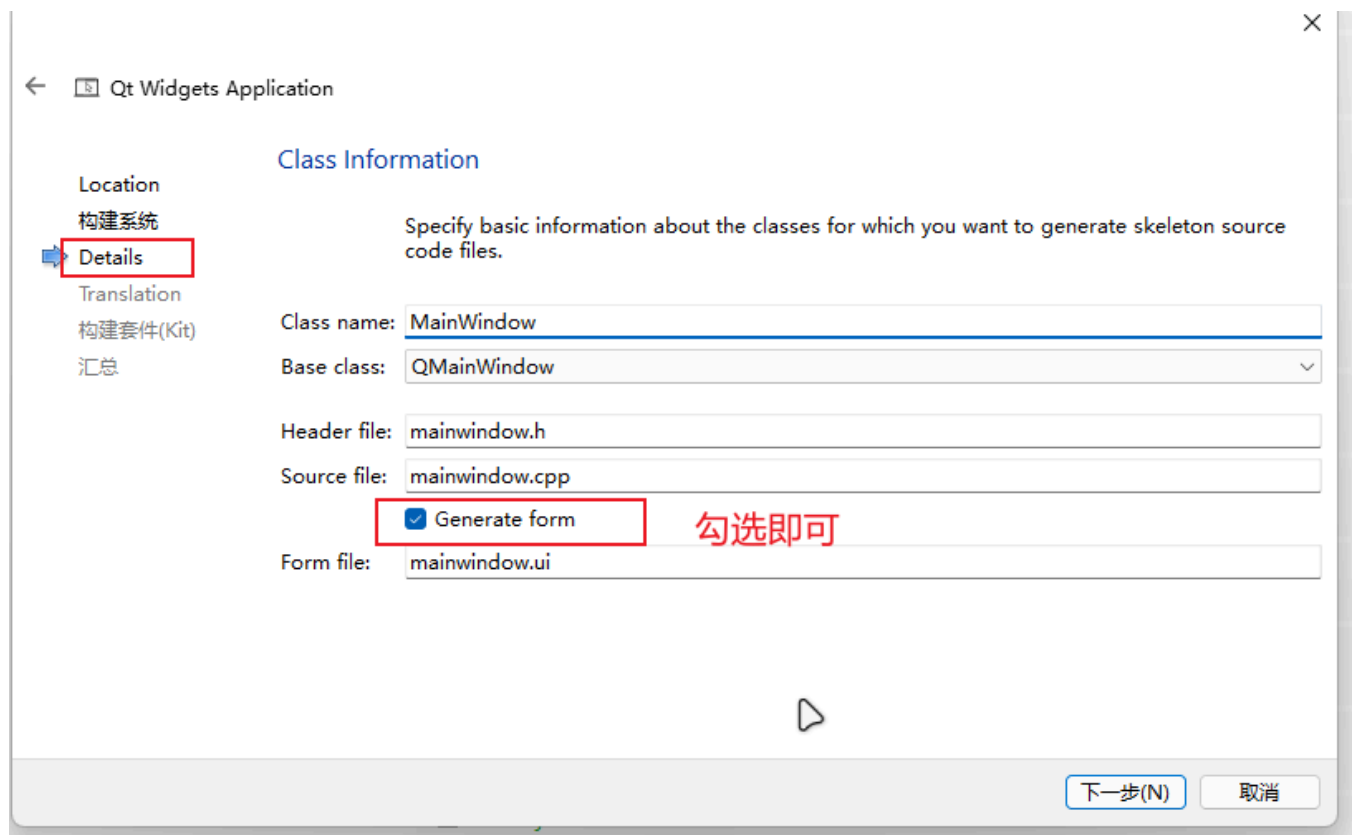
```

- 函数返回值：-> 返回值类型，标识函数返回值的类型。当返回值为 void，或者函数体中只有一处 return 的地方（比如编译器可以自动推断出返回值类型）时，这部分可以省略。如果有返回值，可以在-> 后面写一个返回值类型
- 是函数体：{}，标识函数的实现，这部分不能省略，但函数体可以为空。

## 7. 资源文件

### 7.1 添加 ui 文件

#### 7.1.1 新建时添加 ui 文件



Qt Widgets Application

Location

构建系统

Details

Translation

构建套件(Kit)

汇总

Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name: MainWindow

Base class: QMainWindow

Header file: mainwindow.h

Source file: mainwindow.cpp

☒ Generate form

Form file: mainwindow.ui

勾选即可

下一步(N) 取消

Figure 13



### 7.1.2 已经创建的文件中添加 ui 文件

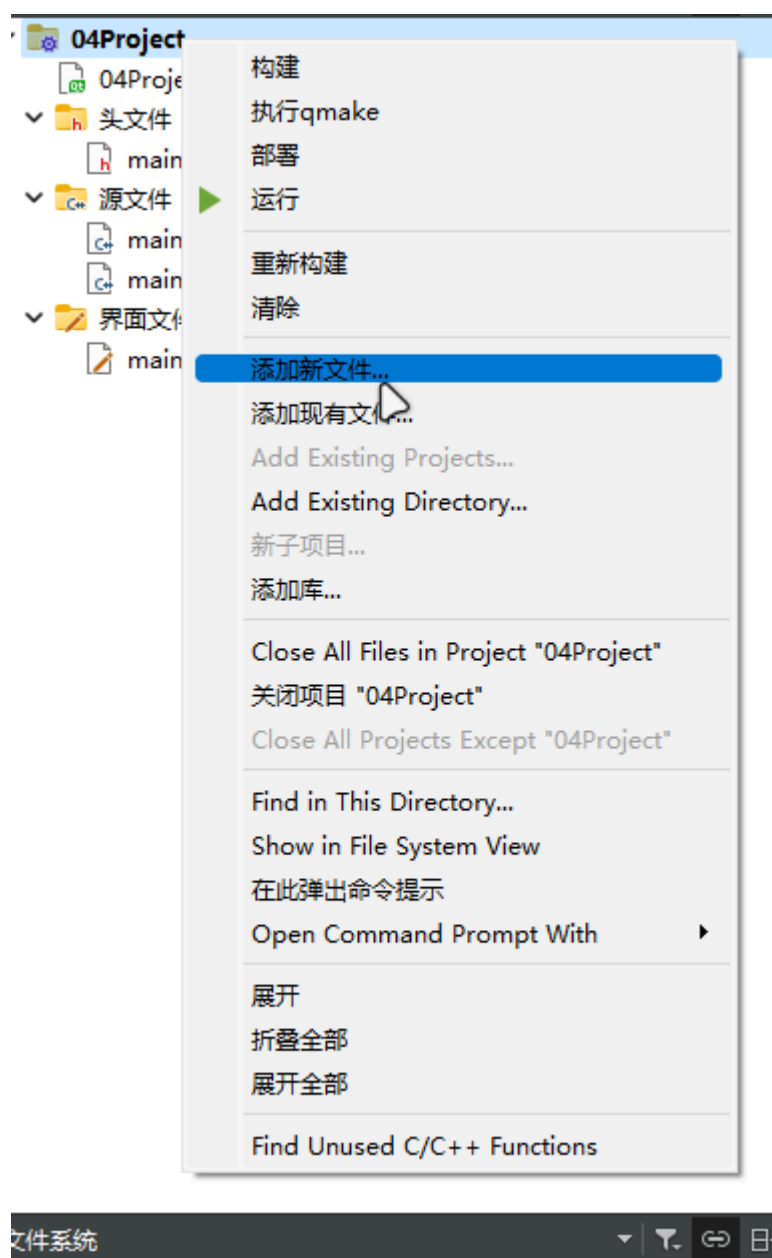


Figure 14

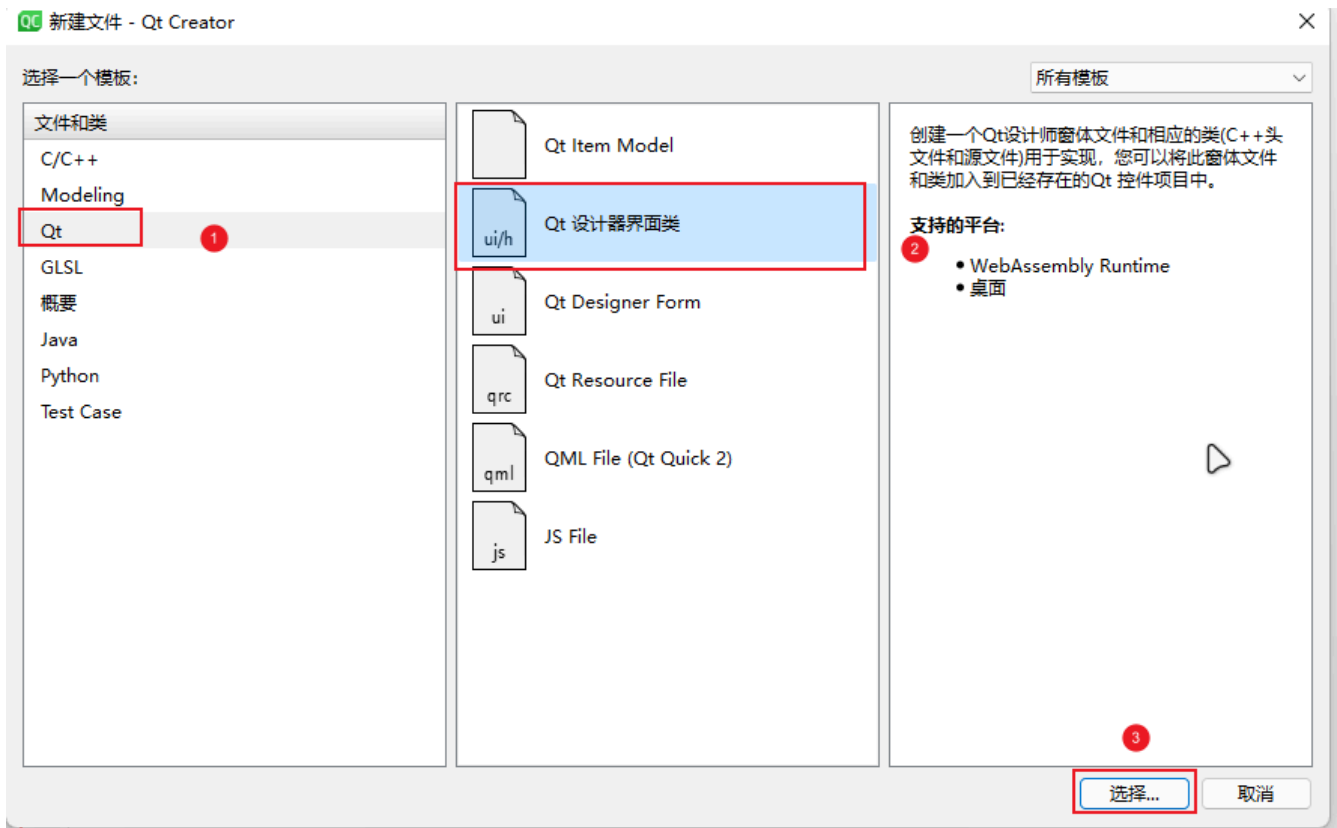


Figure 15

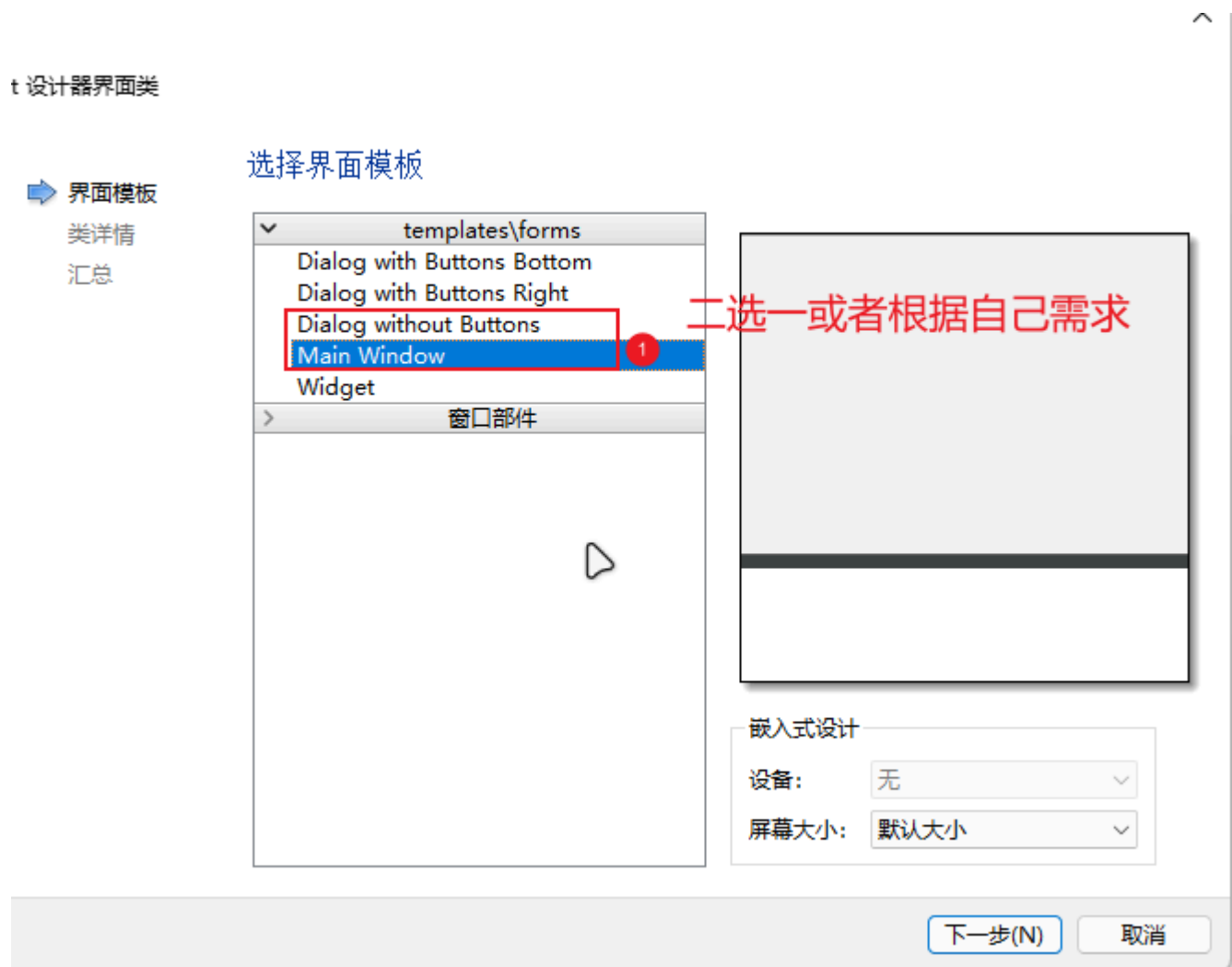


Figure 16

## 7.2 使用 ui 文件

使用时，注意有些地方需要敲回车确定

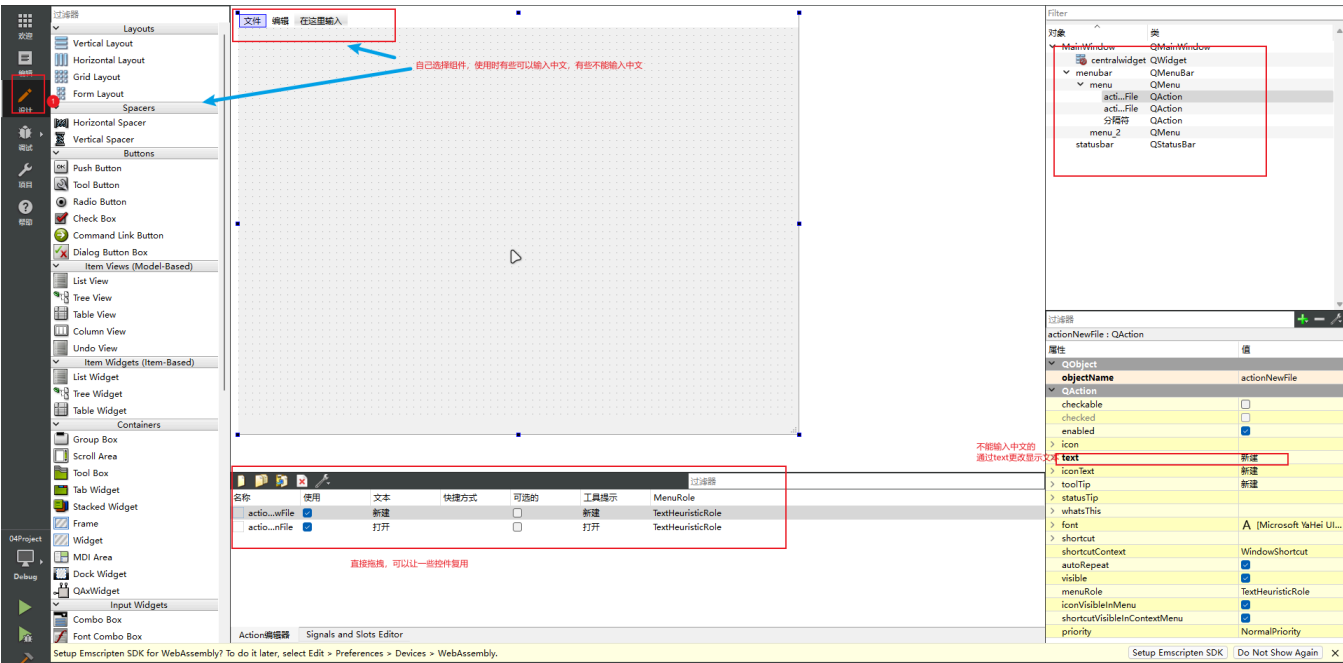


Figure 17

## 7.3 添加资源文件

### 7.3.1 创建资源文件

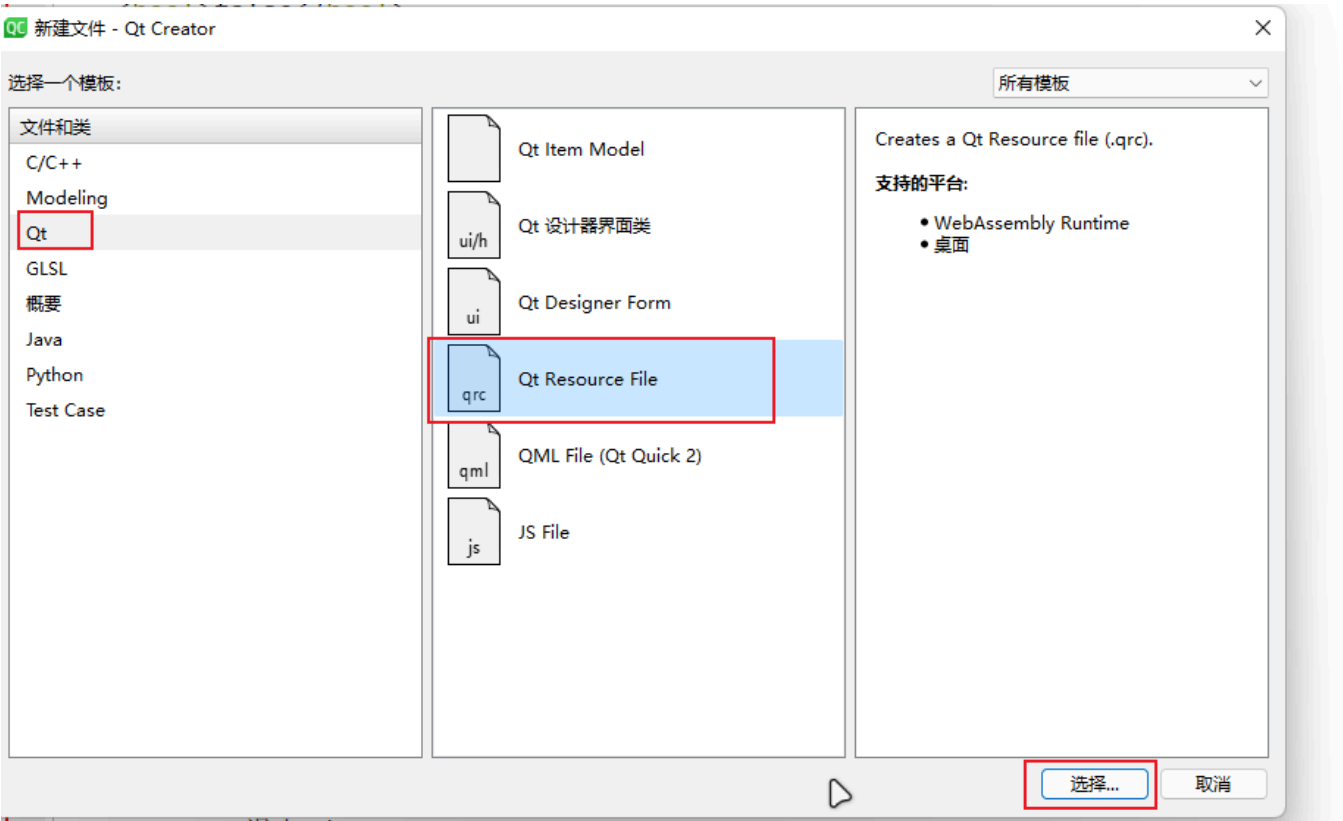


Figure 18

### 7.3.2 回到资源文件界面

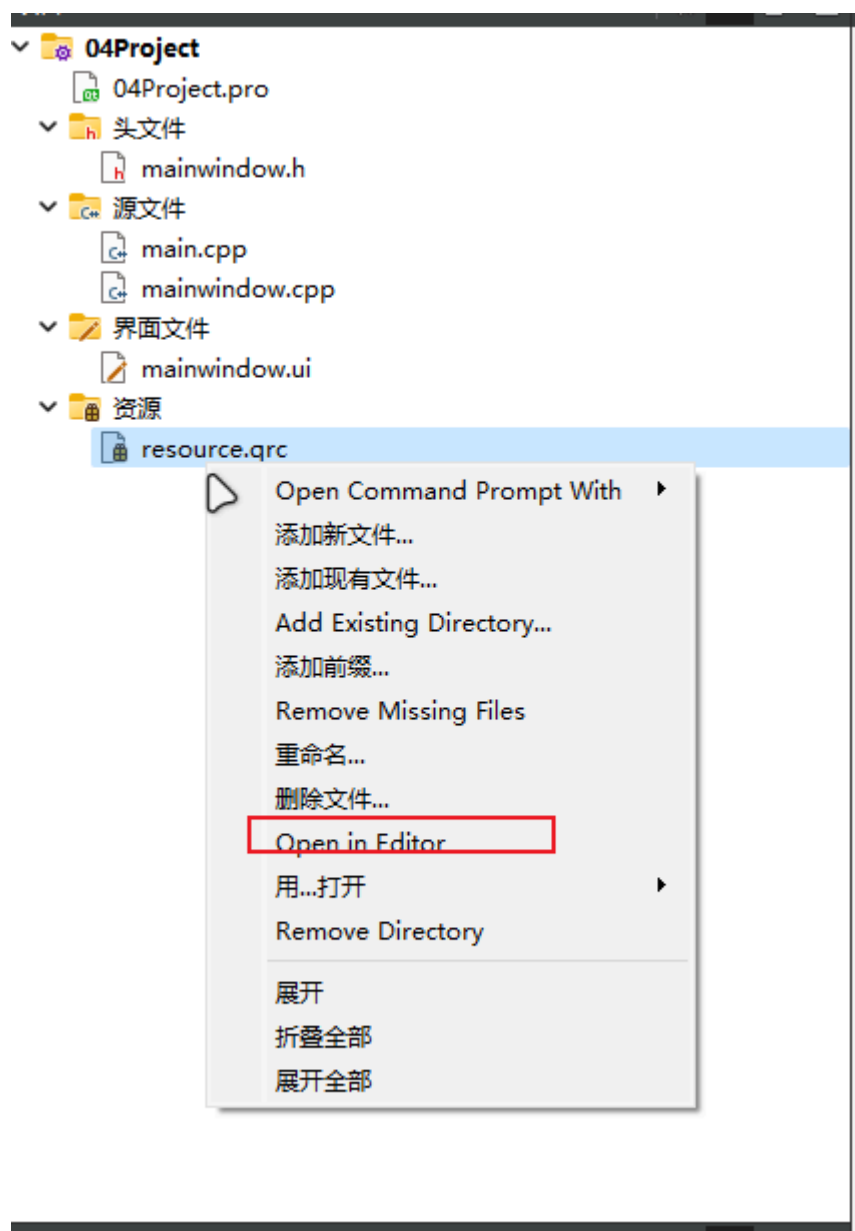


Figure 19

### 7.3.3 添加资源

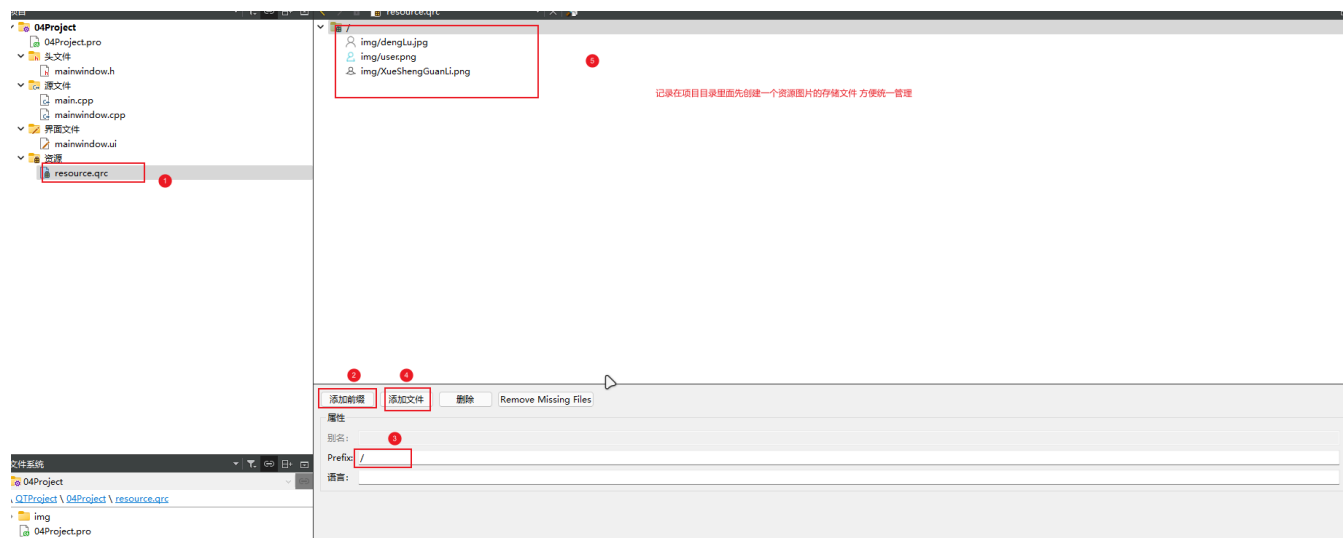


Figure 20

### 7.3.4 使用

```
1 //使用Qt资源文件 :+前缀名+资源名
2 //使用的对象-> setIcon(QIcon( ":+前缀名+资源名"));
3 ui->actionNewFile->setIcon(QIcon(":/img/dengLu.jpg"));
4 ui->actionOpenFile->setIcon(QIcon(":/img/user.png"));
```

Fence 18

## 8. 界面布局

## 8.1 控件布局

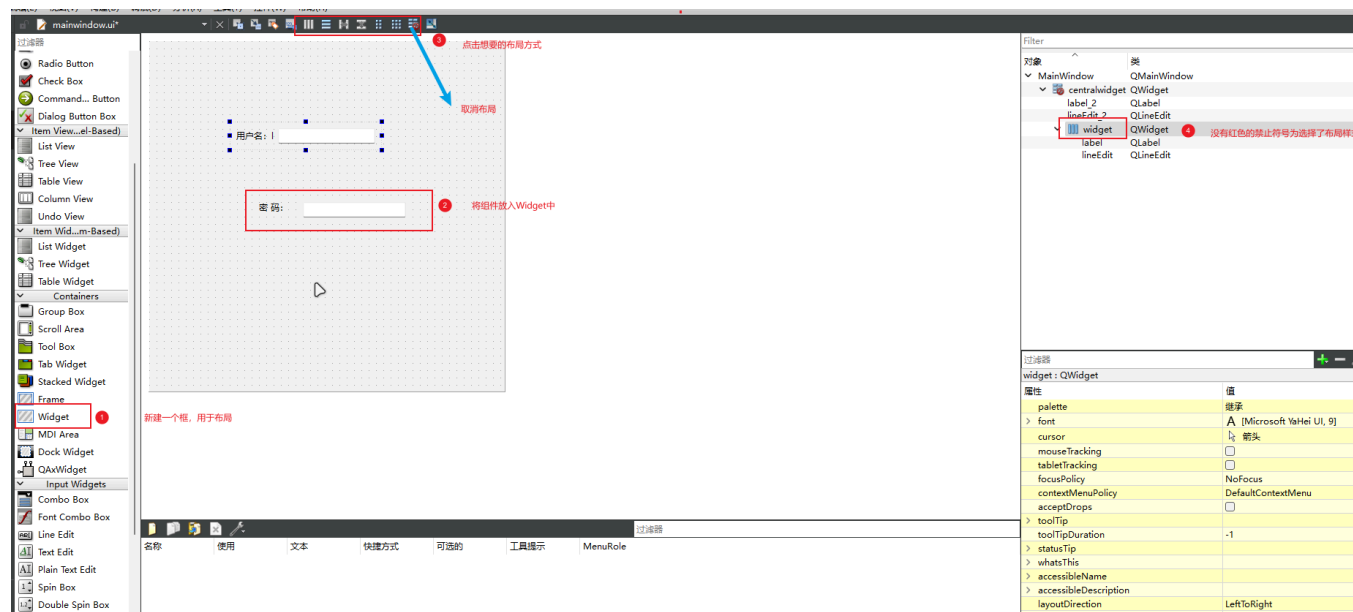


Figure 21

# 8.2 弹簧

用于自适应控件之间的距离。



Figure 22

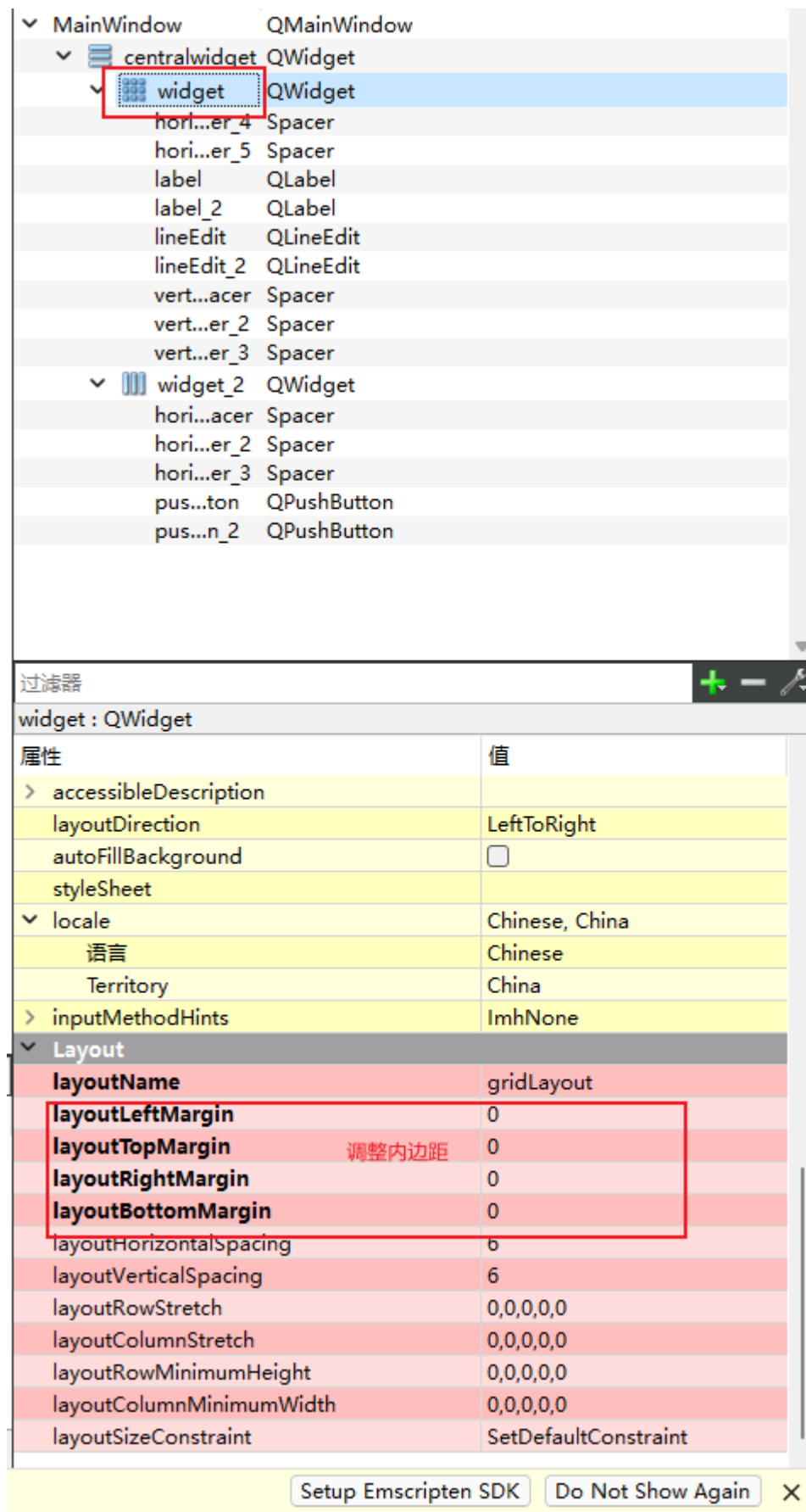


Figure 23

label	QLabel
label_2	QLabel
lineEdit	QLineEdit
lineEdit_2	QLineEdit
vert...acer	Spacer
vert...er_2	Spacer
vert...er_3	Spacer
▼ widget_2	QWidget
hori...acer	Spacer
hori...er_2	Spacer
hori...er_3	Spacer
pus...ton	QPushButton
pus...n_2	QPushButton

过滤器		+	-
lineEdit_2 : QLineEdit			
属性	值		
▼ QLineEdit			
> inputMask			
▼ text			
可翻译的	<input checked="" type="checkbox"/>		
澄清			
注释			
maxLength	32767 修改成密码框		
frame	<input checked="" type="checkbox"/>		
<b>echoMode</b>	Password		
cursorPosition	0		
▼ alignment	左对齐, 垂直中心对齐		
水平的	左对齐		
垂直的	垂直中心对齐		
dragEnabled	<input type="checkbox"/>		
readOnly	<input type="checkbox"/>		
▼ placeholderText			
可翻译的	<input checked="" type="checkbox"/>		
澄清			
注释			
cursorMoveStyle	LogicalMoveStyle		
clearButtonEnabled	<input type="checkbox"/>		

Figure 24

## 9. ui 控件

### 9.1 按钮控件

#### 9.1.1 工具按钮



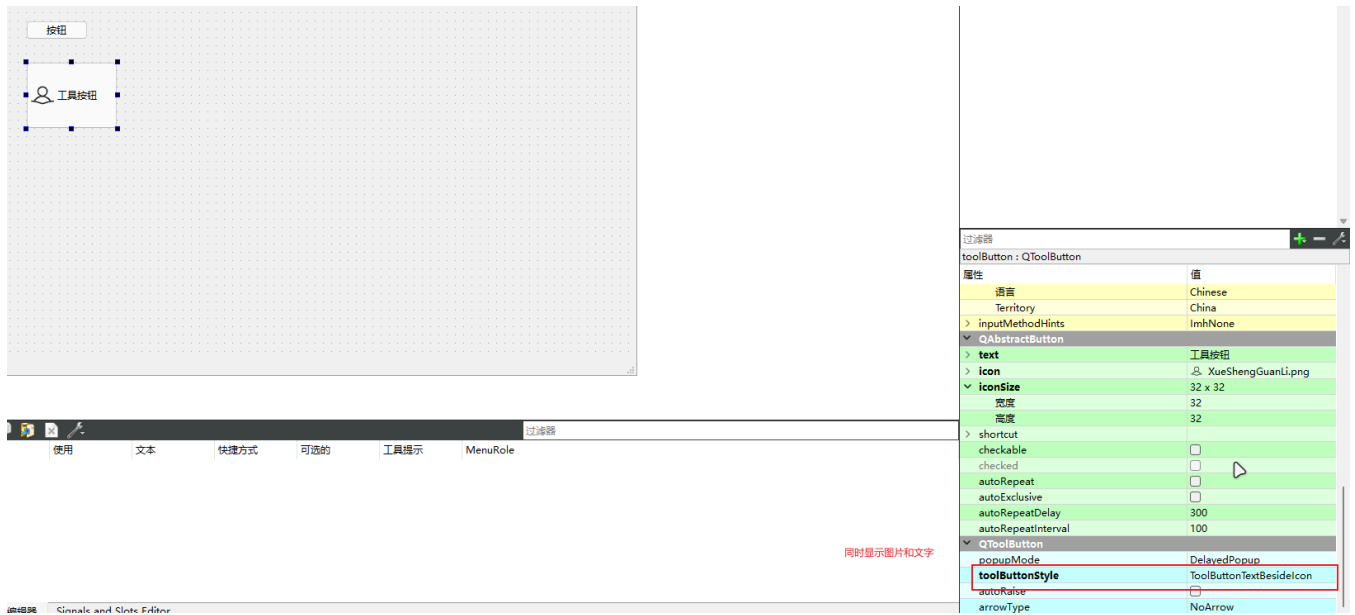


Figure 25

下面的 autoRaise 选上 变成突出风格

### 9.1.2 单选按钮



Figure 26

```

1 //默认男选择
2 ui->radioButton->setChecked(true);
3
4 connect(ui->radioButton_3,&QRadioButton::clicked,[=]() {
5     qDebug() << "多性别人士";
6 });

```

Fence 19

### 9.1.3 复选按钮

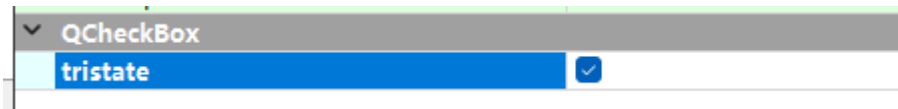


Figure 27

```

1 //stateChanged 获取按钮状态 1为半选 2为全选 0为不选
2 connect(ui->checkBox_3,&QCheckBox::stateChanged,[=](int state){
3     qDebug() << state;
4 });

```

Fence 20

## 9.2 QListWidget 控件

```

1 //设置List widget
2 //添加一行 但是可以设置文本位置
3 QListWidgetItem *item = new QListWidgetItem("锄禾日当午");
4 ui->listwidget->addItem(item);
5 item->setTextAlignment(Qt::AlignHCenter); //居中
6
7 //添加一个文本 但是不可以设置文本位置
8 QStringList stl; //qt风格的List<String>
9 stl << "汗滴留下土" << "谁知盘中餐" << "粒粒皆辛苦";
10 ui->listwidget->addItem(stl);

```

Fence 21

## 9.3 QTreeWidget 控件

```

1 //treewidget 控件
2 //添加表头
3 ui->treewidget->setHeaderLabels(QStringList() << "英雄" << "英雄简介");
4
5 //添加头节点
6 QTreeWidgetItem * i1 = new QTreeWidgetItem(QStringList() << "打野");
7 QTreeWidgetItem * i2 = new QTreeWidgetItem(QStringList() << "射手");
8 QTreeWidgetItem * i3 = new QTreeWidgetItem(QStringList() << "法师");
9 ui->treewidget->addTopLevelItem(i1);
10 ui->treewidget->addTopLevelItem(i2);
11 ui->treewidget->addTopLevelItem(i3);
12
13 //添加子节点

```

```

14     QTreeWidgetItem *i11 = new QTreeWidgetItem(QStringList() << "兰陵王" << "隐身与爆发输出，兰陵王拥有强大的隐身能力，可以快速接近敌人并造成高额伤害。他的技能设计使得他在团战中也能发挥重要作用，是灵活多变的刺客型打野。");
15     QTreeWidgetItem *i12 = new QTreeWidgetItem(QStringList() << "韩信" << "灵活位移与控制，韩信的位移技能多样，能够快速穿梭战场，避开敌人的攻击。他擅长控制敌人位置，为队友创造输出机会，是极具机动性的打野英雄。");
16     QTreeWidgetItem *i13 = new QTreeWidgetItem(QStringList() << "李白" << "飘逸位移与群体削弱，李白以其高超的位移技巧和飘逸的战斗风格著称。他能够对多个敌人造成伤害，同时保持自身的灵活性，是团战中的重要输出点。");
17     i1->addChild(i11);
18     i1->addChild(i12);
19     i1->addChild(i13);
20
21     QTreeWidgetItem *i21 = new QTreeWidgetItem(QStringList() << "后羿" << "高爆发伤害，拥有远程消耗与控制能力，适合前期压制对手");
22     QTreeWidgetItem *i22 = new QTreeWidgetItem(QStringList() << "孙尚香" << "高机动性与输出，能够快速切入战场并造成大量伤害。");
23     QTreeWidgetItem *i23 = new QTreeWidgetItem(QStringList() << "公孙离" << "独特技能机制，集高伤害输出与灵活走位于一体，擅长爆发式击杀。");
24     i2->addChild(i21);
25     i2->addChild(i22);
26     i2->addChild(i23);
27
28     QTreeWidgetItem *i31 = new QTreeWidgetItem(QStringList() << "妲己" << "高爆发伤害，技能控制能力强，适合团队中消耗敌人和提供控制。");
29     QTreeWidgetItem *i32 = new QTreeWidgetItem(QStringList() << "安琪拉" << "持续输出能力强，拥有位移技能，能够灵活走位躲避敌人攻击。");
30     QTreeWidgetItem *i33 = new QTreeWidgetItem(QStringList() << "小乔" << "高伤害输出，技能带有范围伤害，适合在团战中造成大量AOE伤害。");
31     i3->addChild(i31);
32     i3->addChild(i32);
33     i3->addChild(i33);

```

Fence 22

## 9.4 QTableWidget

```

1 //tablewidget 控件
2 //设置列数
3 ui->tablewidget->setColumnCount(3);
4
5 //设置表头
6 ui->tablewidget->setHorizontalHeaderLabels(QStringList() << "姓名" << "性别" << "年龄");
7
8 //设置行数
9 ui->tablewidget->setRowCount(5);
10
11 //设置单元格
12 //0,0开始
13 //ui->tablewidget->setItem(0,0,new QTableWidgetItem("111"));
14 QString str[] = {

```

```

15         "姐己", "女", "20", "后羿", "男", "22", "鲁班大师", "男", "25", "孙尚香", "女", "23", "安琪
    拉", "女", "21"
16     };
17     for(int i = 0; i < 5; i++)
18     {
19         int t = i * 3;
20         ui->tablewidget->setItem(i,0,new QTableWidgetItem(str[t]));
21         ui->tablewidget->setItem(i,1,new QTableWidgetItem(str[t + 1]));
22         ui->tablewidget->setItem(i,2,new QTableWidgetItem(str[t + 2]));
23
24         //int 转 string QString::number()
25     }

```

Fence 23

## 9.5 其他控件

### 9.5.1 scroll Area(下拉条)

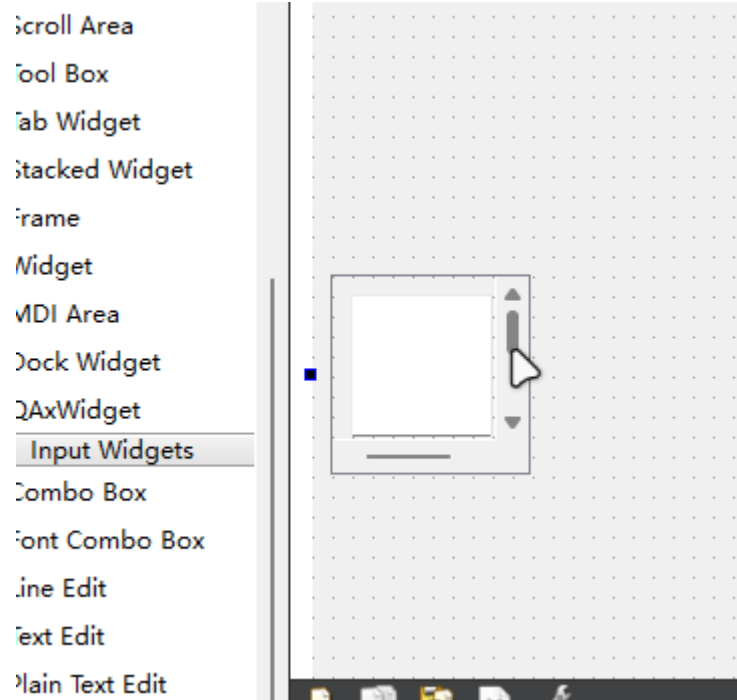


Figure 28

9.5.2 Tool Box(类似抽屉)

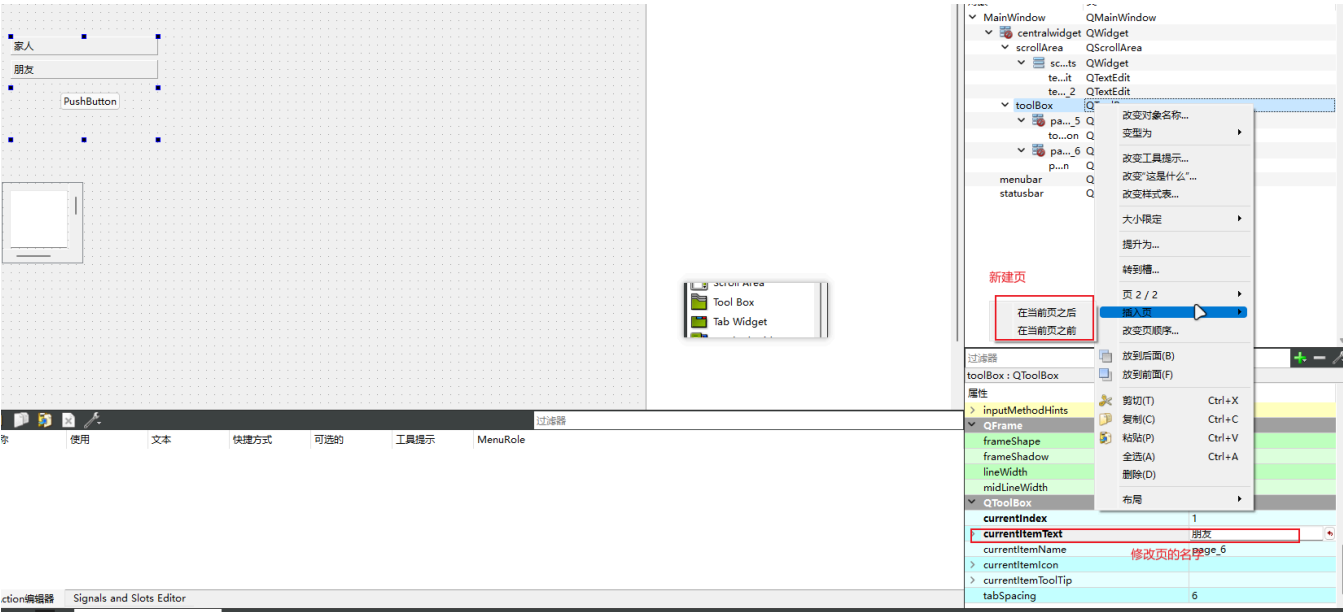


Figure 29

9.5.3 Tab Widget (类似网页切换)

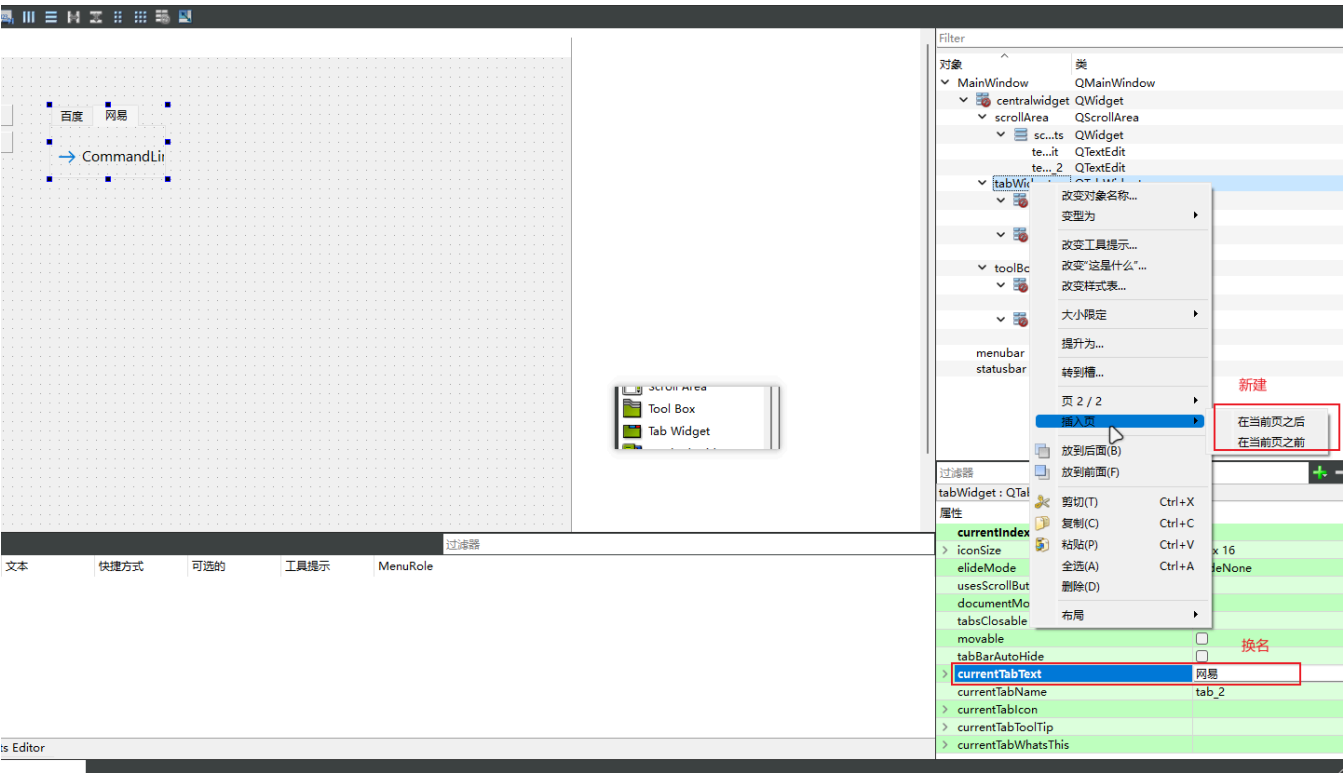


Figure 30

## 9.5.4 Stacked Widget (栈控件——翻页的切换页面)



Figure 31

```
1 //栈控件的控制 查看ui->stackedwidget中的currentIndex属性，才能知道其他控件在多少页
2 //设置默认第一个页面
3 ui->stackedwidget->setCurrentIndex(2);
4 //Tool Box
5 connect(ui->pushButton_3,&QPushButton::clicked,[=]() {
6     ui->stackedwidget->setCurrentIndex(2);
7
8 });
9
10 //Scroll Area
11 connect(ui->pushButton_4,&QPushButton::clicked,[=]() {
12     ui->stackedwidget->setCurrentIndex(1);
13 });
14
15 //Tab widget
16 connect(ui->pushButton_5,&QPushButton::clicked,[=]() {
17     ui->stackedwidget->setCurrentIndex(0);
18 });
```

### 9.5.5 combo Box(下拉框) font Combo Box(字体下拉框)

```
1 //下拉框
2 ui->comboBox->addItem("1份宫保鸡丁");
3 ui->comboBox->addItem("2份宫保鸡丁");
4 ui->comboBox->addItem("3份宫保鸡丁");
5 ui->comboBox->addItem("没有宫保鸡丁");
6
7 connect(ui->pushButton_6,&QPushButton::clicked,[=]() {
8     ui->comboBox->setCurrentIndex(1);
9     // ui->comboBox->setCurrentText("2份宫保鸡丁");
10 });
```

Fence 25

- Text Edit 与 Plain Text Edit 都是文本框,前者可以修改文字颜色、倾斜,后者纯文本
- Spin Box 数字加减控件(类似快速跳转翻页)
- Double Spin Box 带小数的数字加减控件
- Time Edit 时间的加减控件
- Date Edit 日期加减控件
- Date/Time Edit 日期/时间加减控件
- Horizontal Scroll Bar 水平滑动条
- Vertical Scroll Bar 垂直滑动条
- Horizontal Slider 水平滑块
- Vertical Slider 垂直滑块
- Label 标签

- 显示图片 `ui->label->setPixmap(QPixmap(":/img/user.png"));`
- 显示动图 `QMovie *movie = new QMovie(".gif资源"); ui->label->setMovie(movie); movie->start(); //播放`

## 9.6 自定义控件封装

### 9.6.1 创建 ui 文件

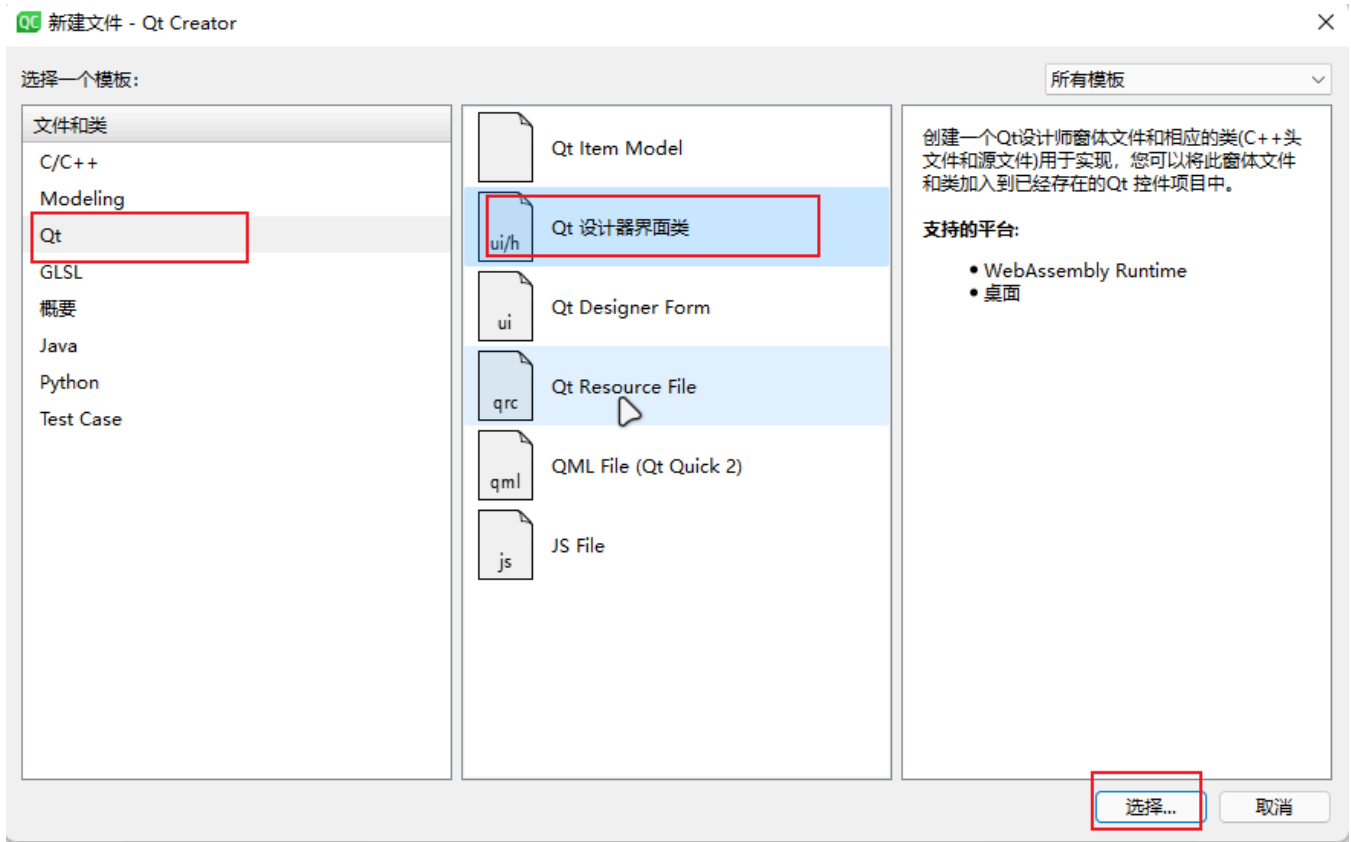


Figure 32

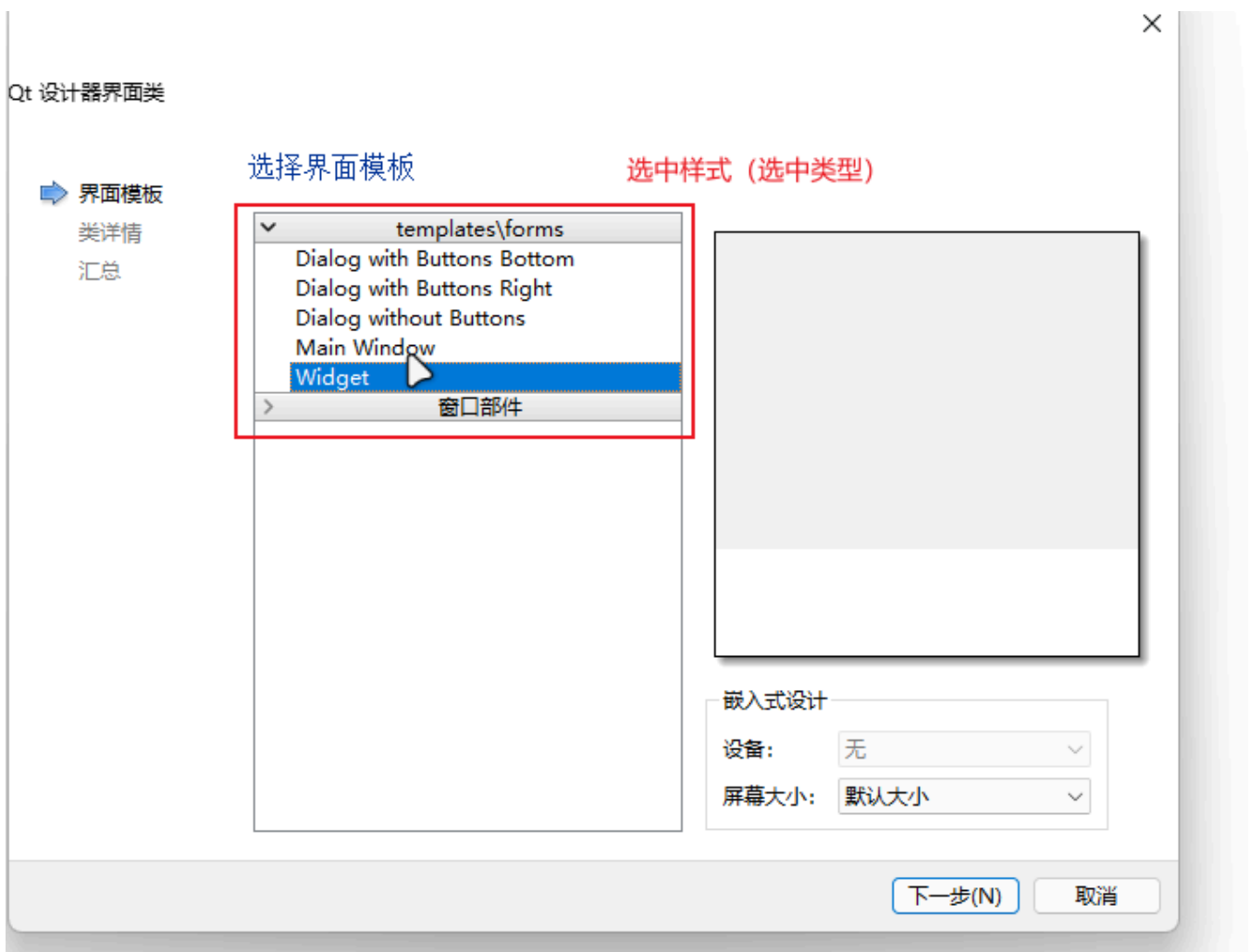


Figure 33





Figure 34

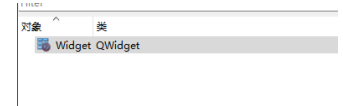
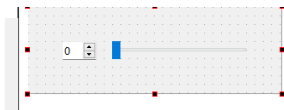


Figure 35

## 9.6.2 引入到其他窗口界面中

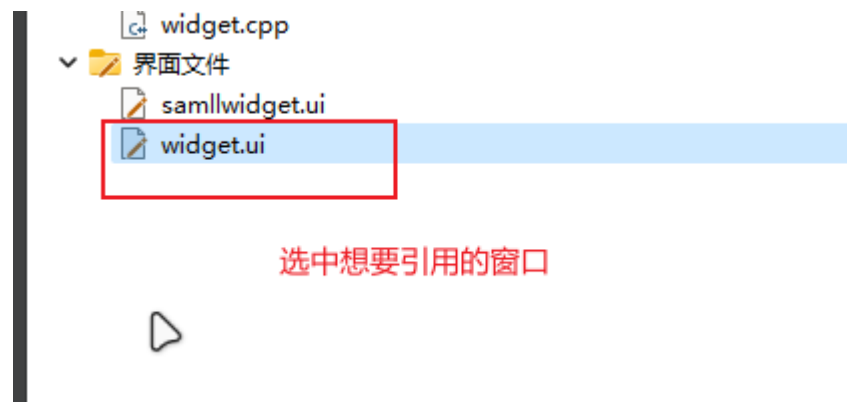


Figure 36

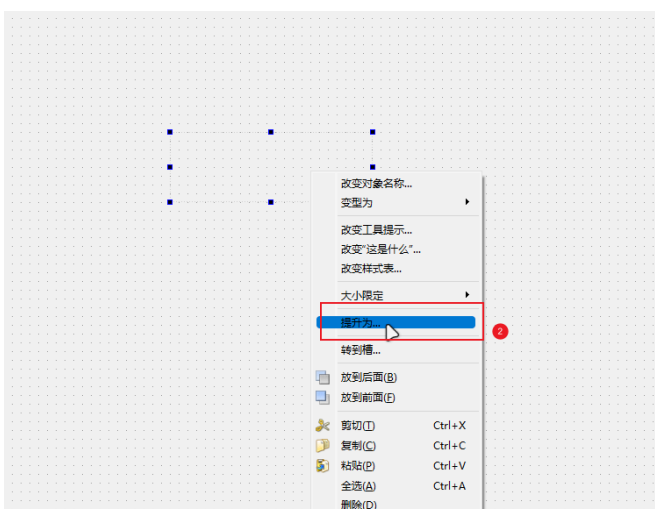


Figure 37



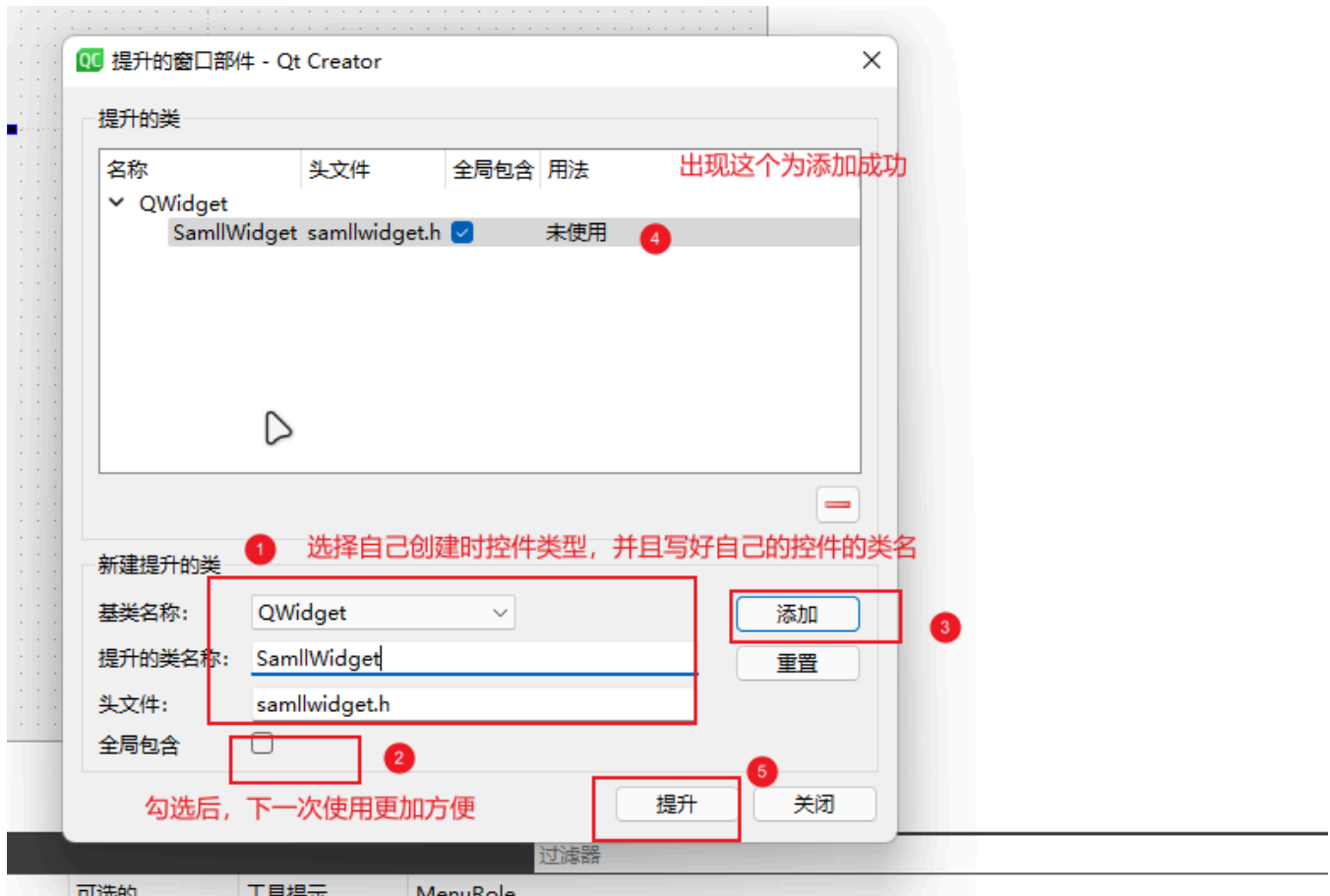


Figure 38

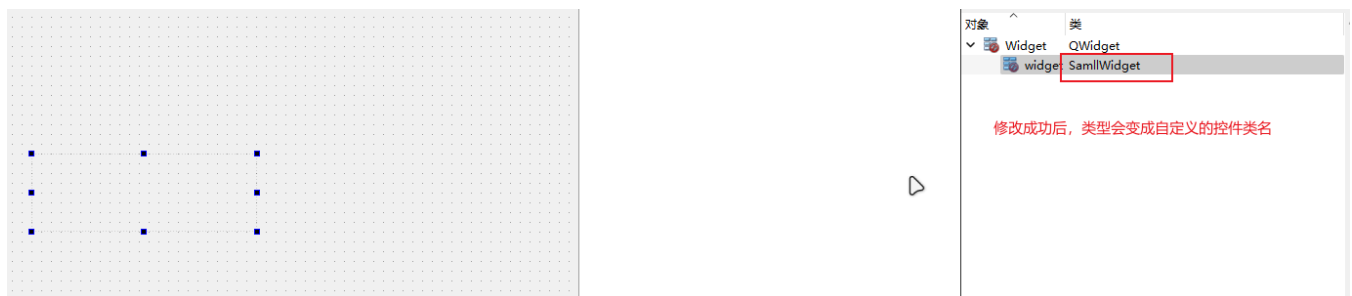


Figure 39

### 9.6.3 给自己定义控件添加功能

```

1 //自定义控件.h
2 namespace Ui {
3 class SamllWidget;
4 }
5
6 class SamllWidget : public QWidget
7 {
8     Q_OBJECT
9
10 public:
11     explicit SamllWidget(QWidget *parent = nullptr);
12     ~SamllWidget();
13 
```

```

14     //获取num
15     int getNum();
16
17     //设置num
18     void setNum(int num);
19
20     //触发信号
21     // void change(int num);
22
23 signals:
24     // 自定义信号
25     void numChanged(int num);
26
27 public slots:
28     // 触发信号
29     void change(int num);
30
31
32
33 private:
34
35     Ui::SamllWidget *ui;
36 };
37
38 //自定义控件.cpp
39 SamllWidget::SamllWidget(QWidget *parent)
40     : QWidget(parent)
41     , ui(new Ui::SamllWidget)
42 {
43     ui->setupUi(this);
44
45     //横向滑块设置范围
46     ui->horizontalSlider->setMaximum(100);
47     ui->horizontalSlider->setMinimum(0);
48
49     //设置spinBox的最大为100
50     ui->spinBox->setMaximum(100);
51
52     //给自定义控件设置功能    spinBox单向绑定horizontalSlider
53     connect(ui->spinBox,&QSpinBox::valueChanged,ui-
54 >horizontalSlider,&QSlider::setValue);
55
56     //horizontalSlider单向绑定spinBox
57     connect(ui->horizontalSlider,&QAbstractSlider::sliderMoved,ui-
58 >spinBox,&QSpinBox::setValue);
59 }
60
61 //获取num
62 int SamllWidget::getNum()
63 {
64     return ui->spinBox->value();
65 }

```

```

65 //设置num
66 void SamllWidget::setNum(int num)
67 {
68     ui->spinBox->setValue(num);
69 }
70
71 // 触发信号
72 void SamllWidget::change(int num)
73 {
74     emit numChanged(num); // 触发自定义信号
75 }
76

```

Fence 26

```

1 //主窗口
2 Widget::Widget(QWidget *parent)
3     : QWidget(parent)
4     , ui(new Ui::Widget)
5 {
6     ui->setupUi(this);
7
8     ui->lineEdit->setText(QString::number(ui->widget->getNum()));
9
10    //+
11    connect(ui->pushButton_2,&QPushButton::clicked,[=]() {
12        int num = 0;
13        num = ui->widget->getNum();
14        num++;
15        if(num >= 100)
16            num = 0;
17        ui->widget->setNum(num);
18        ui->lineEdit->setText(QString::number(num));
19    });
20
21    //-
22    connect(ui->pushButton_3,&QPushButton::clicked,[=]() {
23        int num = 0;
24        num = ui->widget->getNum();
25        num--;
26        if(num <= 0)
27            num = 100;
28        ui->widget->setNum(num);
29        ui->lineEdit->setText(QString::number(num));
30    });
31
32
33    connect(ui->widget,&SamllWidget::numChanged,[=](int num){
34        ui->lineEdit->setText(QString::number(num));
35    });
36 }

```

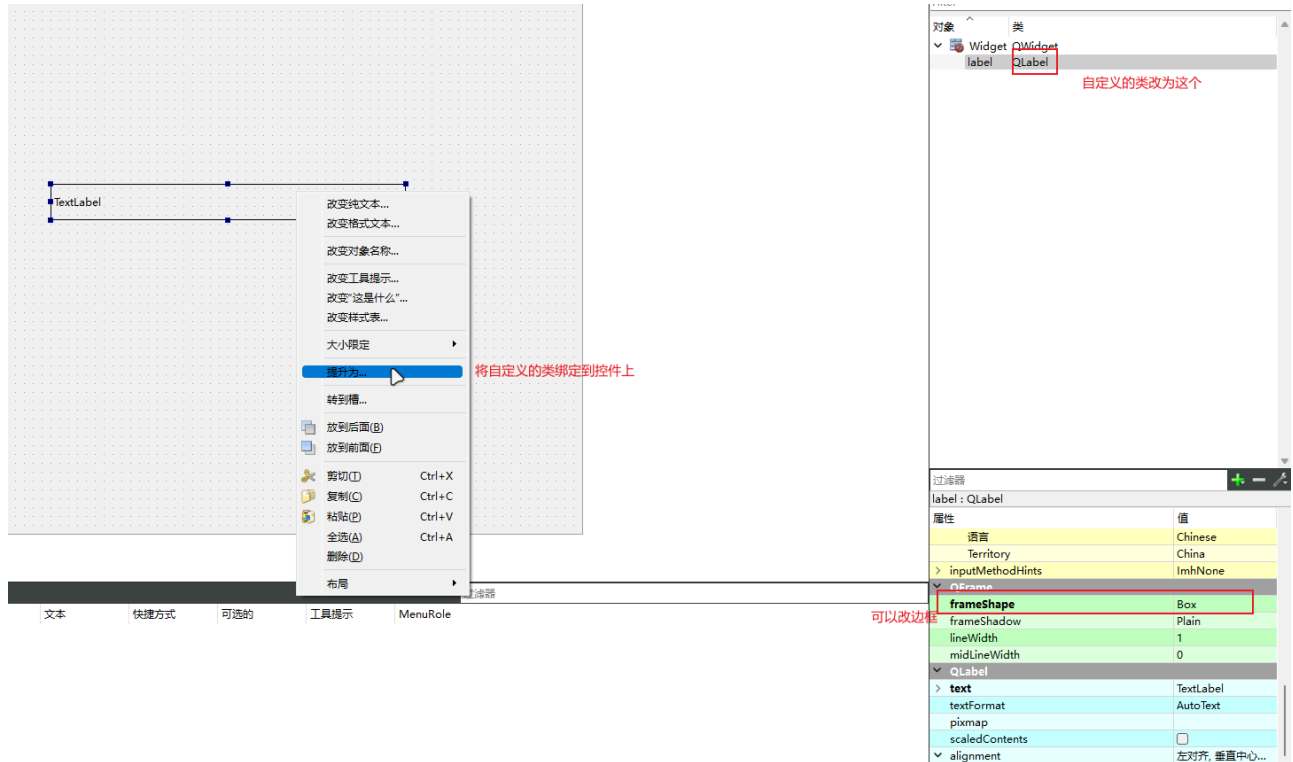
Fence 27

## 10. QEvent(事件)

事件都要重新实现，qt 文档中大多数在 Reimplemented Protected Functions 下面

### 10.1 鼠标事件

- 创建类，并且重写 enterEvent 和 leaveEvent 方法
- 改继承类型，改为和要操作的控件类型一致，如：我要操作 QLabel 控件，将继承改为 QLabel
- 在 ui 界面提升要操作的控件。



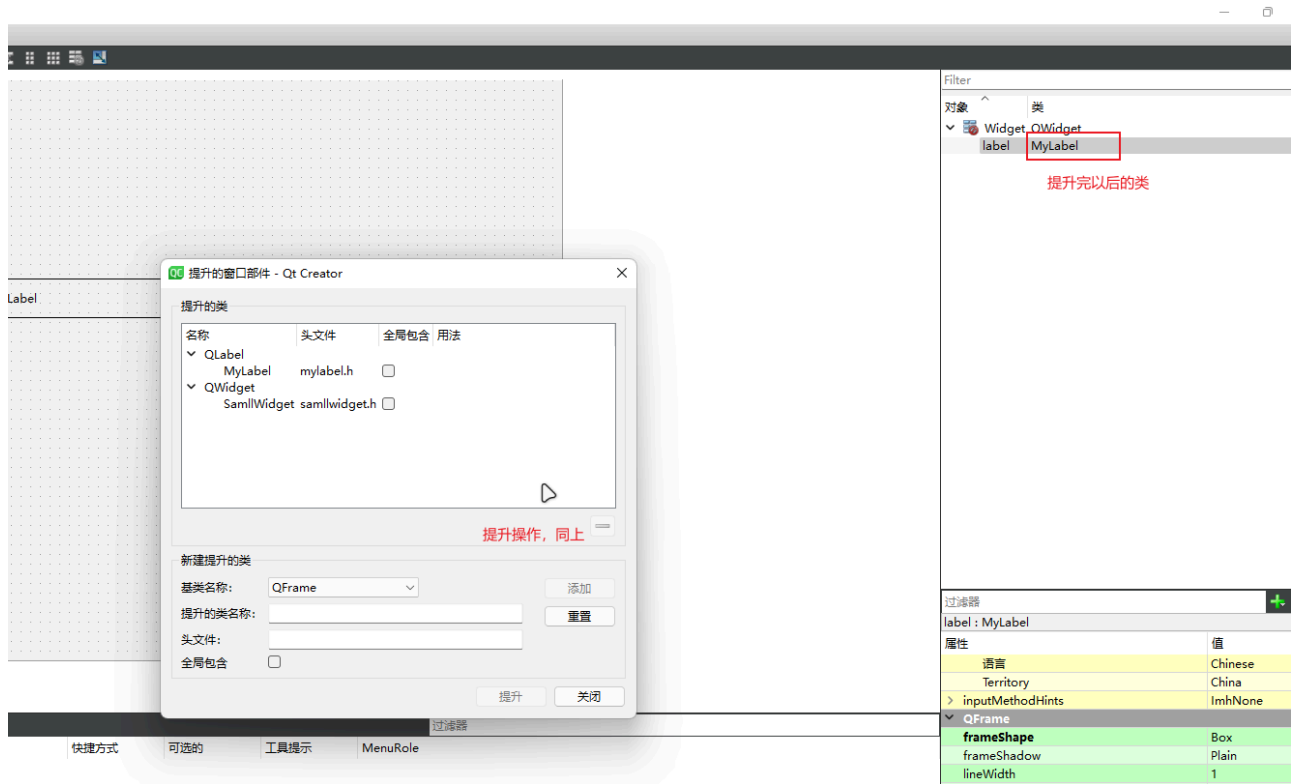


Figure 40

```

1 //MyLabel.h
2 #include <QLabel>
3
4 class MyLabel : public QLabel
5 {
6 public:
7     explicit MyLabel(QWidget *parent = nullptr);
8
9     //重写鼠标进入事件
10    void enterEvent(QEnterEvent *event);
11    //重写鼠标离开事件
12    void leaveEvent(QEvent *event);
13
14    //重写鼠标移动事件
15    void mouseMoveEvent(QMouseEvent *ev);
16    //重写鼠标按下事件
17    void mousePressEvent(QMouseEvent *ev);
18    //重写鼠标释放事件
19    void mouseReleaseEvent(QMouseEvent *ev);
20
21 signals:
22 };
23
24
25 //MyLabel.cpp
26 #include "mylabel.h"
27 #include <QDebug>
28 #include <QMouseEvent>
29 #include <QPointF>
30

```

```
31 MyLabel::MyLabel(QWidget *parent)
32     : QLabel{parent}
33 {
34     //设置鼠标追踪 不需要按下以后才有移动事件的触发
35     setMouseTracking(true);
36
37 }
38
39
40 //重写鼠标进入事件
41 void MyLabel::enterEvent(QEnterEvent *event)
42 {
43     qDebug() << "鼠标进入";
44     qDebug();
45 }
46
47 //重写鼠标离开事件
48 void MyLabel::leaveEvent(QEvent *event)
49 {
50     qDebug() << "鼠标离开";
51     qDebug();
52 }
53
54
55 //重写鼠标移动事件（过程值）
56 void MyLabel::mouseMoveEvent(QMouseEvent *ev)
57 {
58     QPointF f = ev->globalPosition();
59     //1为第一个arg()的占位符
60     //globalPosition是绝对位置 整个屏幕中的位置
61     //scenePosition是相对位置 主窗口中的位置
62     qDebug() << QString("鼠标移动到, x=%1,y=%2,globalx=%3,globaly=%4").arg(ev-
>scenePosition().x())
63         .arg(ev->scenePosition().y()).arg(f.x()).arg(f.y());
64     //qDebug() << f.rx() << " " << f.ry(); //和y() x()没有差别 只是rx和ry只读
65     qDebug();
66 }
67
68 //重写鼠标按下事件（瞬间值）
69 void MyLabel::mousePressEvent(QMouseEvent *ev)
70 {
71     // if(ev->button() == Qt::LeftButton)
72     // {
73     //     qDebug() << "左键按下";
74     // }
75     // else if(ev->button() == Qt::MiddleButton)
76     // {
77     //     qDebug() << "中键按下";
78     // }
79     // else if(ev->button() == Qt::RightButton)
80     // {
81     //     qDebug() << "右键按下";
82     // }
```

```

83
84     if(ev->buttons() == (Qt::LeftButton | Qt::MiddleButton))
85     {
86         qDebug() << "左键和中间按下";
87     }
88     qDebug();
89 }
90
91 //重写鼠标释放事件（瞬间值）
92 void MyLabel::mouseReleaseEvent(QMouseEvent *ev)
93 {
94     //buttons 用于检测按键组合 能用于检测按下事件和移动事件 释放事件中返回值依然是按下按键的值（排除引起该事件的按钮）
95     //button 用于检测按键
96     if(ev->buttons() == (Qt::LeftButton | Qt::MiddleButton))
97     {
98         qDebug() << "三键按下后，左键和中间按下,右键释放";
99     }
100    qDebug();
101 }
102
103

```

Fence 28

## 10.2 定时器事件（概念与 stm32 中学的基本类似）

```

1  //widget.h
2  #include <Qwidget>
3
4  QT_BEGIN_NAMESPACE
5  namespace Ui {
6  class widget;
7  }
8  QT_END_NAMESPACE
9
10 class widget : public Qwidget
11 {
12     Q_OBJECT
13
14 public:
15     widget(Qwidget *parent = nullptr);
16     ~widget();
17
18     //重写定时器事件
19     void timerEvent(QTimerEvent *e);
20
21 private:
22     int id1;           //定时器1的id
23     int id2;           //定时器2的id
24     int num3 = 1;      //定时器3的计数值
25     Ui::widget *ui;

```



```

26     };
27
28     //widget.cpp
29     #include "widget.h"
30     #include "ui_widget.h"
31     #include <QTimer>
32     #include <QPushButton>
33
34     Widget::Widget(QWidget *parent)
35         : QWidget(parent)
36         , ui(new Ui::Widget)
37     {
38         ui->setupUi(this);
39
40         //启动一个定时器
41         id1 = startTimer(1000); //参数1 时间间隔 (ms) 每隔xx毫秒调用定时器事
件 返回值为定时器的唯一id
42
43         id2 = startTimer(2000);
44
45         //第二种定时器方式 (信号和槽, 非事件)
46         QTimer *t = new QTimer(this);
47         t->start(500); //0.5秒
48         connect(t, &QTimer::timeout, [=]() {
49             ui->label_4->setText(QString::number(this->num3++));
50         });
51
52         //暂停
53         connect(ui->pushButton, &QPushButton::clicked, [=]() {
54             if(t->isActive()) //ui->pushButton->text() == "暂停" isActive()
是否在运行
55             {
56                 t->stop(); //暂停
57                 ui->pushButton->setText("继续");
58             }
59             else //ui->pushButton->text() == "继续"
60             {
61                 t->start(); //重启
62                 ui->pushButton->setText("暂停");
63             }
64         });
65
66         //清零
67         connect(ui->pushButton_2, &QPushButton::clicked, [=]() {
68             // t->interval(); //获取当前的毫秒数
69             this->num3 = 1;
70             ui->label_4->setText(QString::number(this->num3));
71         });
72     }
73
74     //重写定时器事件
75     void Widget::timerEvent(QTimerEvent *e)
76     {

```

```

77     //每隔1秒
78     if(e->timerId() == id1)
79     {
80         static int num1 = 1;                //静态变量 让函数第一次调用的时候才赋值，后面
调用就用之前的值
81         ui->label_2->setText(QString::number(num1++));
82     }
83
84     //每隔2秒
85     if(e->timerId() == id2)
86     {
87         static int num2 = 1;                //静态变量 让函数第一次调用的时候才赋值，后面
调用就用之前的值
88         ui->label_3->setText(QString::number(num2++));
89     }
90 }
91
92 Widget::~Widget()
93 {
94     delete ui;
95 }
96

```

Fence 29

## 10.3 事件分发器(event())

- `bool event(QEvent *ev);`
- 返回值为 `bool` 类型。如果返回 `true`，代表用户自己处理了该事件，不在向下分发（不发给系统或者父类）。
- 事件分发器 介于 用户与事件之间，用户触发事件后，由事件分发器来告诉系统该事件是用户 自己处理（重写的 `event` 中处理） 还是系统处理（默认的对应该事件函数）。

```

1  //mylabel.h
2  //重写事件分发器event
3  bool event(QEvent *e);
4
5
6  //mylabel.cpp
7  //重写事件分发器 用于拦截所有事件
8  bool MyLabel::event(QEvent *e)
9  {
10     //处理拦截下的鼠标按下事件
11     if(QEvent::MouseButtonPress == e->type())
12     {
13         //类型转换
14         // QMouseEvent *ev = static_cast<QMouseEvent *>(e);
//static_cast<>() 基本类型之间的转换、上转型（从派生类到基类 子类到父类） 且不会进行继承关系的检查
15         QMouseEvent *ev = dynamic_cast<QMouseEvent *>(e);
//dynamic_cast<>() 向下转型（从基类到派生类 父类到子类） 且会进行继承检查
16
17

```

```

18         qDebug() << QString("用户自己的处理鼠标移动到，
x=%1,y=%2,globalx=%3,globaly=%4").arg(ev->scenePosition().x()).arg(ev-
>scenePosition().y());
19
20         return true;
21     }
22
23     //其他事件调用 父类的方法 默认处理
24     return QLabel::event(e);
25 }

```

Fence 30

## 10.4 事件过滤器

- 在 event 之前可以通过事件过滤器，进行一次高级过滤
- 使用步骤
  - 给控件安装事件过滤器（控件调用）
  - 重写 eventFilter()

```

1 //widget.h
2 //重写事件过滤器
3 bool eventFilter(QObject *obj, QEvent *ev);
4
5
6 //widget.cpp
7 Widget::Widget(QWidget *parent)
8     : QWidget(parent)
9     , ui(new Ui::Widget)
10 {
11
12     //给控件设置过滤器
13     //参数 父对象
14     ui->label->installEventFilter(this);
15
16 }
17 //重写事件过滤器
18 //obj 控件类型
19 //e 事件类型
20 bool Widget::eventFilter(QObject *obj, QEvent *e)
21 {
22     //过滤出label控件
23     if(ui->label == obj)
24     {
25         if(QEvent::MouseButtonDbClick == e->type())
26         {
27             //类型转换
28             QMouseEvent *ev = dynamic_cast<QMouseEvent *>(e);
29             //dynamic_cast<>() 向下转型（从基类到派生类 父类到子类）且会进行继承检查

```

```

30         qDebug() << QString("过滤器过滤的鼠标事件, 移动到,
x=%1,y=%2,globalx=%3,globaly=%4").arg(ev->scenePosition().x()).arg(ev-
>scenePosition().y());
31
32         return true;
33     }
34 }
35
36 //其他事件 默认处理 交给父类
37 return QWidget::eventFilter(obj,e);
38
39 }

```

Fence 31

## 10.5 QPainter(绘图)

### 10.5.1 绘图事件

```

1 //widget.h
2 #include <QWidget>
3
4 QT_BEGIN_NAMESPACE
5 namespace Ui {
6 class widget;
7 }
8 QT_END_NAMESPACE
9
10 class widget : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15     widget(QWidget *parent = nullptr);
16     ~widget();
17
18     //重写绘图事件
19     void paintEvent(QPaintEvent *event);
20
21 private:
22     Ui::widget *ui;
23 };
24
25
26 //widget.cpp
27 #include "widget.h"
28 #include "ui_widget.h"
29 #include <QPainter> //回画家类
30
31 widget::widget(QWidget *parent)
32 : QWidget(parent)

```

```

33     , ui(new Ui::Widget)
34     {
35         ui->setupUi(this);
36     }
37
38 Widget::~Widget()
39 {
40     delete ui;
41 }
42
43 //重写绘图事件
44 void Widget::paintEvent(QPaintEvent *event)
45 {
46     //实例化画家对象 这里this 是告诉在哪里画（画画的地方）
47     QPainter painter(this);
48
49     //画线
50     painter.drawLine(QPoint(0,0),QPoint(100,100));
51
52     //画圆 本质是画椭圆 参数1: 圆心 参数2: x轴的焦点 参数3: y轴的焦点
53     painter.drawEllipse(QPoint(100,100),20,20);
54
55     //画矩形
56     //QRect构造重载版本1 QRect(QPoint(100,100),QPoint(200,200)) 给两个对角的点
57     //QRect构造重载版本2 QRect(QPoint(100,100),QSize(20,10)) 给一个左上角的点，通过
58     //QRect构造重载版本3 QRect(100,100,100,100) 给一个左上角的点，直接给
59     //QRectF 使用浮点精度画矩形
60     painter.drawRect(QRect(100,100,100,100));
61
62
63     QPen pen;
64
65     //要想同时设置样式、颜色等，需要先设置笔的样式
66     pen.setColor(QColor(255,0,0));
67     pen.setStyle(Qt::DotLine);
68
69
70     //改变线条颜色（换笔） 什么时候使用，那之后的线条就会变颜色
71     painter.setPen(pen);
72
73     //改变线条样式 但是不会改变文字的线条样式
74     painter.setPen(pen);
75
76
77     //设置文字样式
78     QFont font;
79     font.setBold(true); // 设置为粗体
80     font.setItalic(true); // 设置为斜体
81     font.setFamily("Microsoft YaHei"); // 设置字体为微软雅黑
82     font.setPointSize(14); // 设置字体大小为14
83     painter.setFont(font);

```

```

84
85
86 //设置画刷 填充封闭图形的颜色 注意字和填充的顺序
87 QBrush brush(QColor(255, 192, 203));
88 brush.setStyle(Qt::HorPattern); //设置画刷风格
89 painter.setBrush(brush);
90
91 painter.drawRect(QRect(300,300,100,100)); //显示外面的框
92
93 //画文字 在一个矩形框里写字
94 painter.drawText(QRect(300,300,100,100),"你好世界，你好中国！");
95 }

```

Fence 32

## 10.5.2 绘图高级设置

```

1 //widget.cpp
2 void Widget::paintEvent(QPaintEvent *event)
3 {
4     //高级设置
5     painter.drawEllipse(QPoint(100,400),50,50);
6
7     //设置高锯齿 更精细 但是效率更低
8     painter.setRenderHint(QPainter::Antialiasing);
9
10    painter.drawEllipse(QPoint(200,400),50,50);
11
12    //移动画家(将原点(0,0)移动到(100,0))
13    painter.translate(100,0);
14
15    painter.drawEllipse(QPoint(200,400),50,50); //不会重叠
16
17    //保存画家状态
18    painter.save();
19    painter.translate(100,0);
20
21
22    painter.drawEllipse(QPoint(200,400),50,50);
23
24    //还原画家状态
25    painter.restore();
26    // painter.translate(200,0);
27
28    painter.drawEllipse(QPoint(200,400),50,50); //与上一个重叠
29 }

```

Fence 33

### 10.5.3 手动调用绘图事件(利用绘图类画图片中的图)

```
1 //widget.cpp
2
3 #include "widget.h"
4 #include "ui_widget.h"
5 #include <QPainter>           //回画家类
6 #include <QTimer>
7
8 Widget::Widget(QWidget *parent)
9     : QWidget(parent)
10     , ui(new Ui::Widget)
11 {
12     ui->setupUi(this);
13
14     QTimer *t = new QTimer(this);
15
16
17     connect(ui->pushButton,&QPushButton::clicked,[=]() {
18         if(t->isActive())
19         {
20             t->stop();
21             ui->pushButton->setText("启动");
22         }
23         else
24         {
25             t->start(500);
26             ui->pushButton->setText("暂停");
27         }
28     });
29
30
31     connect(t,&QTimer::timeout,[=]() {
32         posy += 20;
33         if(posy > this->height())
34         {
35             posy = 0;
36         }
37     });
38
39     //手动调用绘图事件 用update()更新
40     update();
41 }
42
43 //重写绘图事件
44 void Widget::paintEvent(QPaintEvent *event)
45 {
46     //实例化画家对象 这里this 是告诉在哪里画（画画的地方）
47     QPainter painter(this);
48     //绘制资源图片
49     painter.drawPixmap(QRect(500,posy,20,20),QPixmap(":/img/user.png"));
50 }
```

### 10.5.4 绘图设备

绘图设备是指继承自 **QPaintDevice** 的子类。Qt一共提供了四个这样的类，分别是：

- **QPixmap**：专门为图像在屏幕上的显示做了优化。（不同设备或者操作系统，会有不同的优化）
- **QBitmap**：是 QPixmap 的一个子类，它的色深限定为1。可以使用 QPixmap 的 **isQBitmap()** 函数来确定这个 QPixmap 是否是一个 QBitmap。（只有黑白两种颜色）
- **QImage**：专门为图像的像素级访问做了优化。（可以获取像素点，并且可以对获取的像素点进行赋值）
- **QPicture**：可以用来记录和重现 QPainter 的各类命令。

- QPixmap、QImage、QPicture的使用：

```
1  //widget.h
2  #include <QWidget>
3
4  QT_BEGIN_NAMESPACE
5  namespace Ui {
6  class widget;
7  }
8  QT_END_NAMESPACE
9
10 class widget : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15     widget(QWidget *parent = nullptr);
16     ~widget();
17
18
19     //重写绘图事件
20     void paintEvent(QPaintEvent *event);
21 private:
22     Ui::widget *ui;
23 };
24
25
26
27 //widget.cpp
28 #include "widget.h"
29 #include "ui_widget.h"
30 #include <QPixmap>
31 #include <QPainter>
32 #include <QImage>
33 #include <QPicture>
34
35 widget::widget(QWidget *parent)
```



```

36 : QWidget(parent)
37 , ui(new Ui::Widget)
38 {
39     ui->setupUi(this);
40
41     //创建 QPixmap绘图设备    如果没有指定到widget上, 那么窗口什么也不会显示
42     QPixmap pix(300,400);           //创建设备大小 300x400
43     pix.fill(Qt::white);           //画布背景默认黑色 现在填充为白色
44     //pix = pix.scaled(pix.width() * 0.5, pix.height() * 0.5);    //参数1: 原宽 * 缩
放倍数  参数2: 原高 * 缩放倍数
45
46     //创建画师
47     QPainter pain(&pix);           //指定pix设备
48     pain.setPen(Qt::green);        //设置成绿色
49     pain.drawRect(100,100,100,100); //画一个矩形
50
51     //保存设备  在该路径下创建pix.png进行保存
52     pix.save("E:\\QTPProject\\13Project\\img\\pix.png");
53
54
55
56     //创建 QImage绘图设备  可以对像素点进行访问
57     QImage img(300,400,QImage::Format_RGB32);    //宽 高 文件类型
58     img.fill(Qt::white);           //画布背景默认黑色 现在填充为白色
59     QPainter pain1(&img);
60     pain1.setPen(Qt::blue);        //设置成绿色
61     pain1.drawRect(100,100,100,100);    //画一个矩形
62
63     //保存
64     img.save("E:\\QTPProject\\13Project\\img\\img.png");
65
66
67
68     //创建 QPicture绘图设备  记录和重现 绘图指令(代码)
69     QPicture pic;
70     QPainter pain2;
71     pain2.begin(&pic);
72     pain2.setPen(Qt::yellow);
73     pain2.drawRect(100,100,100,100);
74     pain2.end();
75
76     //保存
77     pic.save("E:\\QTPProject\\13Project\\img\\pic.tj");    //文件名为自定义的(可以保证
数据隐私), 里面存储的是begin()~end() 之间的代码
78
79 }
80
81 QWidget::~Widget()
82 {
83     delete ui;
84 }
85
86 //重写绘图事件

```

```

87 void widget::paintEvent(QPaintEvent *event)
88 {
89     //指定到窗口
90     // QImage img;
91     // img.load(":/img/R-C.jpg");           //从文件中加载图片
92     // QPainter p(this);
93
94     // for(int i = 60; i < 100; i++)
95     // {
96     //     for(int j = 60; j < 100; j++)
97     //     {
98     //         QRgb va = qRgb(255,0,0);       //设置像素点的颜色
99     //         img.setPixel(i,j,va);         //访问像素点
100    //     }
101    // }
102
103    // p.drawImage(0,0,img);
104
105    //重现代码
106    QPicture pic;
107    QPainter p(this);
108    pic.load("E:\\QTProject\\13Project\\img\\pic.tj"); //加载
109    p.drawPicture(0,0,pic); //复现绘图操作 不会完全覆盖 按照原图
    //比例作画 drawPixmap(0,0,this->width(),this->height(),pic); 会拉伸图片
110 }

```

Fence 35

- QPixmap与QBitmap区别:

```

1 void PaintWidget::paintEvent(QPaintEvent *)
2 {
3     QPixmap pixmap("./Image/butterfly.png");
4     QPixmap pixmap1("./Image/butterfly1.png");
5
6     QBitmap bitmap("./Image/butterfly.png");
7     QBitmap bitmap1("./Image/butterfly1.png");
8
9     QPainter painter(this);
10    painter.drawPixmap(0, 0, pixmap);
11    painter.drawPixmap(200, 0, pixmap1);
12    painter.drawPixmap(0, 130, bitmap);
13    painter.drawPixmap(200, 130, bitmap1);
14 }

```

Fence 36

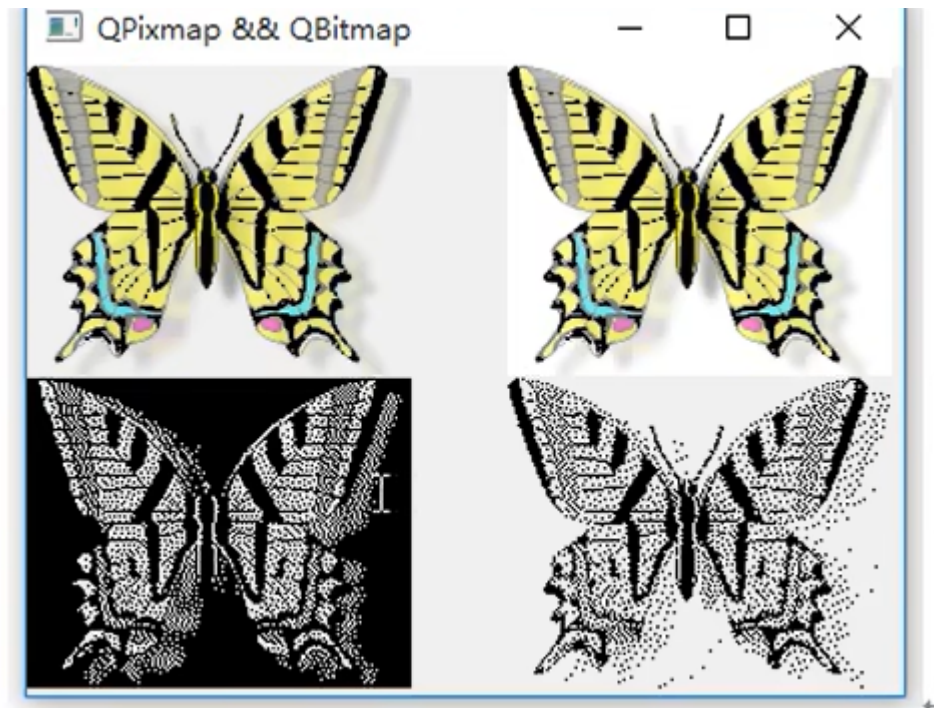


Figure 41

- 绘出了两张 .png 图片。butterfly1.png 是没有透明色的纯白背景，而 butterfly.png 是具有透明色的背景。我们分别使用 QPixmap 和 QBitmap 来加载它们。
- 注意它们的区别：
  - 白色的背景在 QBitmap 中消失了，而透明色在 QBitmap 中转换成了黑色；其他颜色则是使用点的疏密程度来体现的。

## 11. QFile(对文件进行读写操作)

```

1  //widget.cpp
2  #include "widget.h"
3  #include "ui_widget.h"
4  #include <QFile>
5  #include <QFileDialog>
6  // #include <QStringConverter>
7  // #include <QStringDecoder>
8  // #include <QMessageBox>
9
10 widget::widget(QWidget *parent)
11     : QWidget(parent)
12     , ui(new Ui::widget)
13 {
14     ui->setupUi(this);
15
16     connect(ui->pushButton, &QPushButton::clicked, [=]() {
17         //创建堆区文件对话框
18         QString path = QFileDialog::getOpenFileName(this, "打开文件",
19             "../14Project/text");
20         //放入lineEdit中

```

```

21     ui->lineEdit->setText(path);
22
23     //编码格式枚举类型 qt5 QTextCodec
24     //QTextCodec *codec = QTextCodec::codecForName("gbk");
25     //ui->textEdit->setText(codec->toUnicode(array));
26
27     //std::optional<T> 是一个在标准库中提供的模板类，用于表示一个值可以有意义（存在）或者没有
    意义（不存在）的情况。
28     //c++17提供std::optional<T> 是一个在标准库中提供的模板类，用于表示一个值可以有意义（存
    在）或者没有意义（不存在）的情况。
29     /* eg:
30     std::optional<int> findNumber(bool found) {
31     if (found) {
32         return 42; // 返回有效的值
33     }
34     return std::nullopt; // 返回无效值
35     }
36
37     int main() {
38         auto result = findNumber(true);
39         if (result) {
40             std::cout << "Found number: " << *result << "\n";
41         } else {
42             std::cout << "No number found.\n";
43         }
44         return 0;
45     }
46     */
47
48     //获取文本信息 默认获取的文本格式为utf-8（读文件）
49     QFile qf(path);
50     qf.open(QIODeviceBase::ReadOnly); //只读方式打开
51
52     QByteArray ba = qf.readAll(); //读取文件
53     //通过读取一行 qf.readLine() 读取全部
54     //QByteArray ba;
55     // while(!qf.atEnd())
56     // {
57     //     ba += qf.readLine();
58     // }
59
60
61     qf.close(); //关闭文件
62     ui->textEdit->setText(QString::fromLocal8Bit(ba)); //会进行隐式转换
    QByteArray转存QString 将GBK转为UTF-8
63     // ui->textEdit->setText(ba);
64     //utf-8 转 GBK 数据.toLocal8Bit().data()
65
66
67     //写文件
68     QFile fw(path);
69     fw.open(QIODeviceBase::Append); //追加方式写入
70     fw.write("\n"); //换行

```

```

71     QString t("哈哈哈哈哈");
72     fw.write(t.toLocal8Bit().data());           //默认写入utf-8格式
73     fw.close();
74     });
75 }

```

Fence 37

## 11.1 QFileInfo对文件信息的读取

```

1 //widget.cpp续
2 //获取文本信息 默认获取的文本格式为utf-8 (读文件)
3     QFile qf(path);
4     qf.open(QIODeviceBase::ReadOnly);           //只读方式打开
5
6     QByteArray ba = qf.readAll();               //读取文件
7     //通过读取一行 qf.readLine() 读取全部
8     //QByteArray ba;
9     // while(!qf.atEnd())
10    // {
11    //     ba += qf.readLine();
12    // }
13
14
15    qf.close();                                   //关闭文件
16    ui->textEdit->setText(QString::fromLocal8Bit(ba)); //会进行隐式转换
17    QByteArray转存QString 将GBK转为UTF-8
18    // ui->textEdit->setText(ba);
19    //utf-8 转 GBK 数据.toLocal8Bit().data()
20
21    //写文件
22    QFile fw(path);
23    fw.open(QIODeviceBase::Append);               //追加方式写入
24    fw.write("\n");                               //换行
25    QString t("哈哈哈哈哈");
26    fw.write(t.toLocal8Bit().data());             //默认写入utf-8格式
27    fw.close();
28
29
30
31    //读取文件的信息
32    QFileInfo fif(path);
33    qDebug() << "文件大小:" << fif.size() << " 后缀名:" << fif.suffix() << " 文件名
34    称" << fif.fileName() << " 文件路径:" << fif.filePath();
35
36    qDebug() << " 创建日期: " << fif.birthTime().toString("yyyy/MM/dd hh:mm:ss");
37    qDebug() << "最后修改日期:" << fif.metadataChangeTime().toString("yyyy/MM/dd
38    hh:mm:ss");

```

Fence 38

## 12. 翻金币项目

### 12.1 主窗口

```
1 //mainscene.h
2 #ifndef MAINSCENE_H
3 #define MAINSCENE_H
4
5 #include <QMainWindow>
6 #include "chooselevelscene.h"
7
8 QT_BEGIN_NAMESPACE
9 namespace Ui {
10 class MainScene;
11 }
12 QT_END_NAMESPACE
13
14 class MainScene : public QMainWindow
15 {
16     Q_OBJECT
17
18 public:
19     MainScene(QWidget *parent = nullptr);
20     ~MainScene();
21
22
23     //重写绘图事件
24     void paintEvent(QPaintEvent *event);
25
26
27
28     //创建关卡场景
29     ChooseLevelScene *m_choose = NULL;
30 private:
31     Ui::MainScene *ui;
32 };
33 #endif // MAINSCENE_H
34
35
36
37 //mainscene.cpp
38 #include "mainscene.h"
39 #include "ui_mainscene.h"
40 #include "mypushbutton.h"
41 #include <QPainter>
42 #include <QPixmap>
43 #include <QImage>
44 #include <QTimer>
45 #include <QPushButton>
46 #include <QSoundEffect>
47 #include <QUrl>
48
```

```

49  MainScene::MainScene(QWidget *parent)
50      : QMainWindow(parent)
51      , ui(new Ui::MainScene)
52  {
53      ui->setupUi(this);
54
55      //设置窗口固定大小
56      this->setFixedSize(320,588);
57
58      //设置窗口图标
59      this->setWindowIcon(QIcon(":/img/Coin0001.png"));
60
61      //设置标题
62      this->setWindowTitle("翻金币");
63
64
65      /* 退出功能 */
66      connect(ui->quit,&QAction::triggered,this,&MainScene::close);
67
68
69      //准备按钮音效
70      QSoundEffect *sound = new QSoundEffect(this);           //绑定父对象
71      sound->setSource(QUrl::fromLocalFile(":/img/TapButtonSound.wav"));
72
73
74      /* 设置start按钮 */
75      QString normalImg = ":/img/MenuSceneStartButton.png";
76      MyPushButton *startBtn = new MyPushButton(normalImg);
77      startBtn->setParent(this);                               //绑定父对象
78      int btnW = this->width() * 0.5 - startBtn->width() * 0.5;
79      int btnH = this->height() * 0.7;
80      startBtn->move(btnW,btnH);                               //移动按钮
81      this->setFocusPolicy(Qt::NoFocus);                       // 去除焦点边界 ‘虚线边框’
82
83      //实例化 关卡场景
84      this->m_choose = new ChooseLevelScene;
85
86      //监听back信号,进行返回
87      connect(this->m_choose,&ChooseLevelScene::chooseLevelSceneBack,[=]() {
88          QTimer::singleShot(500,this->m_choose,[=]() {
89              //保持前后窗口位置一致
90              this->setGeometry(this->m_choose->geometry());
91              this->show();
92              this->m_choose->hide();
93          });
94      });
95
96
97
98      /* 设置按钮按下效果 切换关卡 */
99      connect(startBtn,&MyPushButton::clicked,[=]() {
100          //弹起特效
101          startBtn->downAnim();

```

```

102         startBtn->upAnim();
103
104         //播放音频
105         sound->play();
106
107         //延时进入关卡场景
108         //参数1 延迟500ms 参数2 作用对象 参数3 延迟后进行的操作
109         QTimer::singleShot(500,this,[=]() {
110             //保持前后窗口位置一致
111             this->m_choose->setGeometry(this->geometry());
112             //自身隐藏
113             this->hide();
114
115             //显示关卡场景
116             this->m_choose->show();
117
118         });
119
120
121     });
122 }
123
124 MainScene::~MainScene()
125 {
126     delete ui;
127 }
128
129 //绘画事件
130 void MainScene::paintEvent(QPaintEvent *event)
131 {
132     /* 绘制背景图片 */
133     QPainter per(this);           //创建画家 指定在窗口作画
134     QPixmap pix;                  //创建绘画设备
135     pix.load(":/img/PlayLevelSceneBg.png"); //加载文件
136     per.drawPixmap(0,0,this->width(),this->height(),pix); //作画 void
137     QPainter::drawPixmap(0,0,pix); //不会完全覆盖 按照原图比例作画
138
139     /* 绘制左上角标题图片 */
140     // pix.load(":/img/Title.png");
141     // pix = pix.scaled(pix.width() * 0.5, pix.height() * 0.5); //参数1: 原宽 *
142     // per.drawPixmap(10,30,pix); //缩放倍数 参数2: 原高 * 缩放倍数
143
144     QImage img;
145     img.load(":/img/Title.png");
146     img = img.scaled(img.width() * 0.5, img.height() * 0.5);
147     per.drawImage(10,30,img);
148 }

```



## 12.2 自定义按钮类

```
1 //mypushbutton.h
2 #ifndef MYPUSHBUTTON_H
3 #define MYPUSHBUTTON_H
4
5 #include <QPushButton>
6
7 class MyPushButton : public QPushButton
8 {
9     Q_OBJECT
10 public:
11     //explicit MyPushButton(QWidget *parent = nullptr);
12
13     //自己的构造函数
14     MyPushButton(QString normalImg, QString pressImg = "");
15
16     //按钮向下动的动画效果
17     void downAnim();
18
19     //按钮向上动的动画效果
20     void upAnim();
21
22     //重写鼠标按下事件
23     void mousePressEvent(QMouseEvent *ev);
24     //重写鼠标释放事件
25     void mouseReleaseEvent(QMouseEvent *ev);
26
27     QString m_normalImg;           //正常按钮图片
28     QString m_pressImg;           //按下按钮切换的图片
29 signals:
30
31
32 };
33
34 #endif // MYPUSHBUTTON_H
35
36
37 //mypushbutton.cpp
38 #include "mypushbutton.h"
39 #include <QMessageBox>
40 #include <QPropertyAnimation> //动画特效类
41
42 //normalImg 正常按钮 不要显示其他图片
43 //pressImg 按下按钮后显示的图片
44 MyPushButton::MyPushButton(QString normalImg, QString pressImg)
45 {
46     this->m_normalImg = normalImg;
47     this->m_pressImg = pressImg;
48
49     //处理正常状态显示的图片
50     QPixmap pix;
```

```

51     bool ret = pix.load(this->m_normalImg);
52
53     if(!ret)
54     {
55         QMessageBox::critical(this, "错误提示", "图片加载失败! ");
56         return;
57     }
58
59     //设置按钮固定大小
60     this->setFixedSize(pix.width(), pix.height());
61
62     //设置不规则图片样式
63     this->setStyleSheet("QPushButton{border:0px;}");           //里面为css
64
65     //设置图片
66     this->setIcon(QIcon(pix));
67
68     //设置图片大小
69     this->setIconSize(QSize(pix.width(), pix.height()));
70 }
71
72
73 //按钮向下动的动画效果
74 void MyPushButton::downAnim()
75 {
76     //参数1 作用的对象  参数2 动画属性    geometry 按照按钮的几何属性动画化
77     QPropertyAnimation * animation = new QPropertyAnimation(this, "geometry");
78
79     //设置动画时间间隔 200ms
80     animation->setDuration(200);
81
82     //设置起始位置 根据几何属性 参数可以是矩形框
83     animation->setStartValue(QRect(this->x(), this->y(), this->width(), this->
84     >height()));
85
86     //设置结束位置
87     animation->setEndValue(QRect(this->x(), this->y() + 10, this->width(), this->
88     >height()));
89
90     //设置弹跳曲线
91     animation->setEasingCurve(QEasingCurve::OutBounce);
92
93     //开始动画
94     animation->start();
95 }
96
97 //按钮向上动的动画效果
98 void MyPushButton::upAnim()
99 {
100     //参数1 作用的对象  参数2 动画属性    geometry 按照按钮的几何属性动画化
101     QPropertyAnimation * animation = new QPropertyAnimation(this, "geometry");

```

```

102 //设置动画时间间隔 200ms
103 animation->setDuration(200);
104
105 //设置起始位置 根据几何属性 参数可以是矩形框
106 animation->setStartValue(QRect(this->x(),this->y() + 10,this->width(),this-
>height()));
107
108 //设置结束位置
109 animation->setEndValue(QRect(this->x(),this->y(),this->width(),this->height()));
110
111 //设置弹跳曲线
112 animation->setEasingCurve(QEasingCurve::OutBounce);
113
114 //开始动画
115 animation->start();
116 }
117
118 //重载鼠标按下事件
119 void MyPushButton::mousePressEvent(QMouseEvent *ev)
120 {
121 //传入的状态不为空 说明需要有按下状态，切换到初始图片图片
122 if(this->m_pressImg != "")
123 {
124 //处理正常状态显示的图片
125 QPixmap pix;
126 bool ret = pix.load(this->m_pressImg);
127
128 if(!ret)
129 {
130 QMessageBox::critical(this,"点击事件错误提示","图片加载失败!");
131 return;
132 }
133
134 //设置按钮固定大小
135 this->setFixedSize(pix.width(),pix.height());
136
137 //设置不规则图片样式
138 this->setStyleSheet("QPushButton{border:0px;}"); //里面为css
139
140 //设置图片
141 this->setIcon(QIcon(pix));
142
143 //设置图片大小
144 this->setIconSize(QSize(pix.width(),pix.height()));
145 }
146
147 //其他操作让父类完成
148 return QPushButton::mousePressEvent(ev);
149 }
150
151 //重载鼠标释放事件
152 void MyPushButton::mouseReleaseEvent(QMouseEvent *ev)
153 {

```

```

154 //传入的状态不为空 说明需要有按下状态，切换图片
155 if(this->m_pressImg != "")
156 {
157     //处理正常状态显示的图片
158     QPixmap pix;
159     bool ret = pix.load(this->m_normalImg);
160
161     if(!ret)
162     {
163         QMessageBox::critical(this, "释放错误提示", "图片加载失败!");
164         return;
165     }
166
167     //设置按钮固定大小
168     this->setFixedSize(pix.width(), pix.height());
169
170     //设置不规则图片样式
171     this->setStyleSheet("QPushButton{border:0px;}"); //里面为css
172
173     //设置图片
174     this->setIcon(QIcon(pix));
175
176     //设置图片大小
177     this->setIconSize(QSize(pix.width(), pix.height()));
178 }
179
180 //其他操作让父类完成
181 return QPushButton::mouseReleaseEvent(ev);
182 }
183
184

```

Fence 40

声明带默认参数，实现不需要

## 12.3 选择关卡类

```

1 //chooselevelscene.h
2 #ifndef CHOOSELEVELSCENE_H
3 #define CHOOSELEVELSCENE_H
4
5 #include <QMainWindow>
6 #include "palyscene.h"
7
8 class ChooseLevelScene : public QMainWindow
9 {
10     Q_OBJECT
11 public:
12     explicit ChooseLevelScene(QWidget *parent = nullptr);
13
14     //重写绘图事件

```

```

15     void paintEvent(QPaintEvent *event);
16
17
18     PalyScene *paly = NULL;
19
20
21 signals:
22     //发出一个返回主窗口信号
23     void chooseLevelSceneBack();
24 };
25
26 #endif // CHOOSELEVELSCENE_H
27
28
29 //chooselevelscene.cpp
30 #include "chooselevelscene.h"
31 #include "mainscene.h"
32 #include "mypushbutton.h"
33 #include <QMenuBar>
34 #include <QMainWindow>
35 #include <QPixmap>
36 #include <QPainter>
37 #include <QTimer>
38 #include <QMouseEvent>
39 #include <QLabel>
40 #include <QDebug>
41 #include <QSoundEffect>
42 #include <QUrl>
43
44 ChooseLevelScene::ChooseLevelScene(QWidget *parent)
45     : QMainWindow{parent}
46 {
47     //设置窗口固定大小
48     this->setFixedSize(320,588);
49
50     //设置窗口图标
51     this->setWindowIcon(QIcon(":/img/Coin0001.png"));
52
53     //设置标题
54     this->setWindowTitle("翻金币-关卡选择");
55
56     //创建菜单栏
57     QMenuBar *bar = new QMenuBar(this);
58     this->setMenuBar(bar); //放入到当前窗口
59
60     //创建菜单
61     QMenu *startMenu = bar->addMenu("菜单");
62
63     //创建菜单项
64     QAction *exitAction = startMenu->addAction("退出");
65
66     //退出功能
67     connect(exitAction,&QAction::triggered,this,&QWidget::close);

```

```

68
69 //创建音效
70 QSoundEffect *choo = new QSoundEffect(this);
71 choo->setSource(QUrl::fromLocalFile(":/img/TapButtonSound.wav"));
72 QSoundEffect *chooBack = new QSoundEffect(this);
73 chooBack->setSource(QUrl::fromLocalFile(":/img/BackButtonSound.wav"));
74
75 //创建back按钮
76 MyPushButton *backBtn = new
MyPushButton(":/img/BackButton.png", ":img/BackButtonSelected.png");
77 backBtn->setParent(this);
78 backBtn->move(QPoint(this->width() - backBtn->width(), this->height() - backBtn-
>height()));
79
80
81 //返回主界面
82 connect(backBtn, &MyPushButton::clicked, [=]() {
83     chooBack->play();
84     //发出信号
85     emit this->chooseLevelSceneBack();
86 });
87
88
89
90
91 //创建关卡按钮
92 for(int i = 0; i < 20; i++)
93 {
94     MyPushButton *menuBtn = new MyPushButton(":/img/LevelIcon.png");
95     menuBtn->setParent(this);
96     menuBtn->move(i % 4 * 70 + 25, i / 4 * 70 + 130); //取余控制层
数，取商控制列数，乘数控制间隔，家数控制离窗口间距
97
98 //监听按钮
99 connect(menuBtn, &MyPushButton::clicked, [=]() { //通过new存放
在堆区，虽然名字都是menuBtn，但是地址不同
100     //设置动画
101     menuBtn->downAnim();
102     menuBtn->upAnim();
103
104     //设置音效
105     choo->play();
106
107     //进入游戏场景
108
109     paly = new PalyScene(i + 1); //实例化游戏场景
110     this->hide(); //隐藏本界面
111     //设置游戏窗口初始位置
112     paly->setGeometry(this->geometry());
113
114     paly->show(); //显示游戏场景
115
116

```

```

117
118
119         //监听游戏场景的Back信号
120         connect(paly,&PalyScene::palySceneBack,[=]() {
121             //保持前后窗口位置一致
122             this->setGeometry(paly->geometry());
123             this->show();
124             delete paly;                //摧毁paly对象 让下一次进入新的关卡
125             paly = NULL;
126         });
127     });
128
129
130
131     //浮现文字
132     // menuBtn->setText(QString::number(i + 1));           //不能在图片上显示
133     QLabel *label = new QLabel(this);
134     label->move(i % 4 * 70 + 25, i / 4 * 70 + 130);
135     label->setFixedSize(menuBtn->width(),menuBtn->height()); //设置大小
136     label->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter); //设置文本对齐方式
    水平和垂直居中
137     label->setText(QString::number(i + 1));
138
139     //因为Label标签挡住了按钮，所以需要鼠标穿透事件
140     label->setAttribute(Qt::WA_TransparentForMouseEvents); //鼠标穿透属性 值为
    51
141     }
142
143
144 }
145
146
147 //重写绘图事件
148 void ChooseLevelScene::paintEvent(QPaintEvent *event)
149 {
150     //设置背景图
151     QPainter p(this);
152     QPixmap pix;
153     pix.load(":/img/otherSceneBg.png");
154     p.drawPixmap(0,0,this->width(),this->height(),pix);
155
156     //设置背景标题
157     pix.load(":/img/Title.png");
158     p.drawPixmap(((this->width() - pix.width()) *
    0.5),30,pix.width(),pix.height(),pix);
159
160 }
161
162
163

```

## 12.4 翻金市场景

```
1 //palyscene.h
2 #ifndef PALYSCENE_H
3 #define PALYSCENE_H
4
5 #include <QMainWindow>
6 #include "mycoin.h"
7
8 class PalyScene : public QMainWindow
9 {
10     Q_OBJECT
11 public:
12     //explicit PalyScene(QWidget *parent = nullptr);
13
14     PalyScene(int i);
15
16     //重写绘图事件
17     void paintEvent(QPaintEvent *event);
18
19
20
21     int m_id = 0;          //记录游戏关卡
22     int m_coinGrou[4][4];  //用于维护每关数组的二维数组
23     MyCoin * m_coinArray[4][4]; //用于维护每个按钮的属性
24
25     bool m_iswinP = false;
26
27 signals:
28     void palySceneBack();
29 };
30
31 #endif // PALYSCENE_H
32
33
34 //palyscene.cpp
35 #include "palyscene.h"
36 #include "mypushbutton.h"
37 #include "mycoin.h"
38 #include "dataconfig.h"
39 #include <QPixmap>
40 #include <QPainter>
41 #include <QMenuBar>
42 #include <QImage>
43 #include <QLabel>
44 #include <QFont>
45 #include <QTimer>
46 #include <QPropertyAnimation>
47 #include <QSoundEffect>
48 #include <QUrl>
49
50 PalyScene::PalyScene(int i)
```



```

51 {
52     this->m_id = i;
53     QString str = QString("第 %1 关翻金币").arg(m_id);
54     setWindowTitle(str);          //设置窗口标题
55
56     setFixedSize(320,588);
57     this->setWindowIcon(QIcon(":/img/Coin0001.png"));
58
59     //设置back按钮
60     MyPushButton *backBtn = new
MyPushButton(":/img/BackButton.png", ":/img/BackButtonSelected.png");
61     backBtn->setParent(this);
62     backBtn->move(QPoint(this->width() - backBtn->width(), this->height() - backBtn-
>height()));
63
64     //创建音效
65     QSoundEffect *palyBack = new QSoundEffect(this);
66     palyBack->setSource(QUrl::fromLocalFile(":/img/BackButtonSound.wav"));
67     QSoundEffect *palyCoin = new QSoundEffect(this);
68     palyCoin->setSource(QUrl::fromLocalFile(":/img/ConFlipSound.wav"));
69     QSoundEffect *palywin = new QSoundEffect(this);
70     palywin->setSource(QUrl::fromLocalFile(":/img/LevelwinSound.wav"));
71
72     //返回主界面
73     connect(backBtn,&MyPushButton::clicked,[=]() {
74         palyBack->play();
75         //发出信号
76         emit this->palySceneBack();
77     });
78
79     //创建菜单栏
80     QMenuBar *bar = new QMenuBar(this);
81     this->setMenuBar(bar);          //放入到当前窗口
82
83     //创建菜单
84     QMenu *startMenu = bar->addMenu("菜单");
85
86     //创建菜单项
87     QAction *exitAction = startMenu->addAction("退出");
88
89     //退出功能
90     connect(exitAction,&QAction::triggered,this,&QWidget::close);
91
92
93     //显示当前关卡数
94     QLabel *label = new QLabel();
95     label->setParent(this);
96     //设置字体样式, 大小
97     QFont font;
98     font.setFamily("华文新魏");
99     font.setPointSize(20);
100     label->setFont(font);
101

```

```

102     QString str2 = QString("关卡: %1").arg(m_id);
103     label->setText(str2);
104     label->setGeometry(QRect(30,this->height() - 50,120,50)); //设置QLabel形状
105
106
107     //创建好胜利图片
108     QLabel *winLabel = new QLabel();
109     QPixmap tpix;
110     tpix.load(":/img/LevelCompletedDialogBg.png");
111     winLabel->setGeometry(0,0,tpix.width(),tpix.height());
112     winLabel->setPixmap(tpix);
113     winLabel->setParent(this);
114     winLabel->move((this->width() - tpix.width())*0.5,-tpix.height()); //移
到窗口外
115
116     //初始化维护数组
117     dataConfig da;
118     for(int i = 0; i < 4; i++)
119     {
120         for(int j = 0; j < 4; j++)
121         {
122             this->m_coinGrou[i][j] = da.m_data[this->m_id][i][j];
123         }
124     }
125
126     //显示金币背景图案
127     for(int i = 0; i < 4; i++)
128     {
129         for(int j = 0; j < 4; j++)
130         {
131             QLabel *la = new QLabel();
132             QPixmap pix = QPixmap(":/img/BoardNode(1).png");
133             la->setGeometry(0,0,pix.width(),pix.height());
134             la->setPixmap(pix);
135             la->setParent(this);
136
137             la->move(57 + i * 50,200 + j * 50);
138
139             // QLabel la(this); //创建在栈上，作用域完了，就消失
140
141
142             //创建金币并显示
143             QString str3;
144             if(this->m_coinGrou[i][j] == 1)
145             {
146                 str3 = ":/img/Coin0001.png";
147             }
148             else
149             {
150                 str3 = ":/img/Coin0008.png";
151             }
152
153             MyCoin *coin = new MyCoin(str3);

```

```

154         coin->setParent(this);
155         coin->move(59 + i * 50, 204 + j * 50);
156
157         //给金币属性赋值
158         coin->m_x = i;
159         coin->m_y = j;
160         coin->m_falg = this->m_coinGrou[i][j];
161
162
163         //更新维护按钮数组
164         this->m_coinArray[i][j] = coin;
165
166         //监听金币
167         connect(coin, &MyCoin::clicked, [=]() {
168             //消除用户快速点击时，动画没完成，出现贴图错误的第二中方式
169             // if(coin->m_isAnimation)
170             //     return;
171             // else
172             // {
173             //     coin->coinTurn();
174             //     this->m_coinGrou[i][j] = this->m_coinGroup[i][j] == 0 ? 1 :
0;
175             // }
176
177
178             //防止胜利，用户快速点击其他按钮，导致出现bug
179             for(int i = 0; i < 4; i++)
180             {
181                 for(int j = 0; j < 4; j++)
182                 {
183                     this->m_coinArray[i][j]->m_judge = true;
184                 }
185             }
186
187             palyCoin->play();
188
189             coin->coinTurn();
190             //更新维护数组
191             this->m_coinGrou[i][j] = this->m_coinGrou[i][j] == 0 ? 1 : 0;
192
193
194             // QTimer::singleShot 不会阻塞进程。它用于在一定时间后执行指定的槽，所以胜利检
查需要写在这个下面 否则检查的是，周围还没有翻转的情况
195             QTimer::singleShot(300, this, [=]() {
196
197                 //翻转右侧
198                 if(coin->m_x + 1 <= 3)
199                 {
200                     this->m_coinArray[coin->m_x + 1][coin->m_y]->coinTurn();
201                     this->m_coinGrou[coin->m_x + 1][coin->m_y] = this-
>m_coinGrou[coin->m_x + 1][coin->m_y] == 0 ? 1 : 0;
202                 }
203                 //翻转左侧

```

```

204         if(coin->m_x - 1 >= 0)
205         {
206             this->m_coinArray[coin->m_x - 1][coin->m_y]->coinTurn();
207             this->m_coinGrou[coin->m_x - 1][coin->m_y] = this-
>m_coinGrou[coin->m_x - 1][coin->m_y] == 0 ? 1 : 0;
208         }
209         //翻转上面
210         if(coin->m_y - 1 >= 0)
211         {
212             this->m_coinArray[coin->m_x][coin->m_y - 1]->coinTurn();
213             this->m_coinGrou[coin->m_x][coin->m_y - 1] = this-
>m_coinGrou[coin->m_x][coin->m_y - 1] == 0 ? 1 : 0;
214         }
215         //翻转下面
216         if(coin->m_y + 1 <= 3)
217         {
218             this->m_coinArray[coin->m_x][coin->m_y + 1]->coinTurn();
219             this->m_coinGrou[coin->m_x][coin->m_y + 1] = this-
>m_coinGrou[coin->m_x][coin->m_y + 1] == 0 ? 1 : 0;
220         }
221
222         //还有状元
223         for(int i = 0; i < 4; i++)
224         {
225             for(int j = 0; j < 4; j++)
226             {
227                 this->m_coinArray[i][j]->m_judge = false;
228             }
229         }
230
231         //检查胜利
232         this->m_iswinP = true;           //假设胜利
233         for(int i = 0; i < 4; i++)
234         {
235             for(int j = 0; j < 4; j++)
236             {
237                 if(this->m_coinArray[i][j]->m_falg == false)           //存
在一个银币 没有胜利
238                 {
239                     this->m_iswinP = false;
240                     break;
241                 }
242             }
243
244             if(this->m_iswinP == false)
245                 break;
246         }
247
248         //让每个按钮下的都有胜利属性
249         if(this->m_iswinP == true)
250         {
251             palywin->play();
252

```

```

253         for(int i = 0; i < 4; i++)
254         {
255             for(int j = 0; j < 4; j++)
256             {
257                 this->m_coinArray[i][j]->m_iswin = true;
258             }
259         }
260
261
262         //动画效果的移回胜利窗口
263         QPropertyAnimation *anim = new
QPropertyAnimation(winLabel, "geometry");
264         anim->setDuration(1000);           //设置动画时间
265         anim->setStartValue(QRect(winLabel->x(), winLabel->
>y(), winLabel->width(), winLabel->height()));    //设置开始位置
266         anim->setEndValue(QRect(winLabel->x(), winLabel->y() +
114, winLabel->width(), winLabel->height()));    //设置结束位置
267         anim->setEasingCurve(QEasingCurve::OutBounce);
        //设置缓和曲线
268
269         anim->start();
270
271
272     }
273     });
274 \
275     });
276 }
277 }
278
279
280 }
281
282 //重写绘图事件
283 void PalyScene::paintEvent(QPaintEvent *event)
284 {
285     QPixmap pix;
286     QPainter p(this);
287     pix.load(":/img/PlayLevelSceneBg.png");    //加载
288     p.drawPixmap(0,0,this->width(),this->height(),pix);
289
290     QImage img;
291     img.load(":/img/Title.png");
292     img = img.scaled(img.width() * 0.5, img.height() * 0.5);
293     p.drawImage(10,30,img);
294 }
295

```

## 12.5 金币类

```
1  //mycoin.h
2  #ifndef MYCOIN_H
3  #define MYCOIN_H
4
5  #include <QPushButton>
6  #include <QTimer>
7
8  class MyCoin : public QPushButton
9  {
10     Q_OBJECT
11 public:
12     //explicit MyCoin(QWidget *parent = nullptr);
13
14     //传入显示的图片
15     MyCoin(QString btnImg);
16
17     //改变标志函数
18     void coinTurn();
19
20     //重写 点击 事件 第一种消除用户快速点击时，动画没完成，出现贴图错误方法
21     void mousePressEvent(QMouseEvent *e);
22
23     int m_x;           //记录行
24     int m_y;           //记录列
25     bool m_falg;       //记录正方
26
27     int m_min = 1;     //图片的最小值
28     int m_max = 8;     //图片的最大值
29     QTimer *m_t1;      //定时器1 记录正翻反
30     QTimer *m_t2;      //定时器2 记录反翻正
31
32     bool m_isAnimation = false; //执行动画的标志 默认没有 防止用户快速点击时，动画没完成，
    出现贴图错误
33     bool m_iswin = false;      //胜利状态
34     bool m_judge = false;      //记录胜利状态 防止有人点快出现bug
35 signals:
36 };
37
38 #endif // MYCOIN_H
39
40 //mycoin.cpp
41 #include "mycoin.h"
42 #include <QPixmap>
43 #include <QMessageBox>
44
45 MyCoin::MyCoin(QString btnImg)
46 {
47     QPixmap pix;
48     bool rel = pix.load(btnImg);
49 }
```

```

50     if(!rel)
51     {
52         QMessageBox::critical(this, "金币 错误提示框", "图片加载失败");
53         return;
54     }
55
56     this->setFixedSize(pix.width(), pix.height());
57     this->setStyleSheet("QPushButton{border:0px;}");
58     this->setIconSize(QSize(pix.width(), pix.height()));
59     this->setIcon(QIcon(pix));
60
61     //初始化定时器
62     this->m_t1 = new QTimer(this);
63     this->m_t2 = new QTimer(this);
64
65     //监听定时器
66     connect(this->m_t1, &QTimer::timeout, [=]() {
67         QPixmap pix1;
68         pix1.load(QString(":/img/Coin000%1.png").arg(this->m_min++));
69         this->setFixedSize(pix1.width(), pix1.height());
70         this->setStyleSheet("QPushButton{border:0px;}");
71         this->setIconSize(QSize(pix1.width(), pix1.height()));
72         this->setIcon(QIcon(pix1));
73
74         this->m_t1->start(30);
75
76         //翻完
77         if(this->m_min > this->m_max)
78         {
79             this->m_min = 1;
80             this->m_t1->stop();
81             this->m_isAnimation = false;
82         }
83
84     });
85
86     connect(this->m_t2, &QTimer::timeout, [=]() {
87         QPixmap pix1;
88
89         pix1.load(QString(":/img/Coin000%1.png").arg(this->m_max--));
90         this->setFixedSize(pix1.width(), pix1.height());
91         this->setStyleSheet("QPushButton{border:0px;}");
92         this->setIconSize(QSize(pix1.width(), pix1.height()));
93         this->setIcon(QIcon(pix1));
94
95         this->m_t2->start(30);
96
97         if(this->m_min > this->m_max)
98         {
99             this->m_max = 8;
100             this->m_t2->stop();
101             this->m_isAnimation = false;
102

```

```

103         }
104
105     });
106 }
107
108 //改变标志函数
109 void MyCoin::coinTurn()
110 {
111
112     //正翻反
113     if(this->m_falg)
114     {
115         //启动定时器1
116         this->m_t1->start(30);
117         this->m_falg = !this->m_falg;
118         this->m_isAnimation = true;
119     }
120     else
121     {
122         //启动定时器2
123         this->m_t2->start(30);
124         this->m_falg = !this->m_falg;
125         this->m_isAnimation = true;
126     }
127 }
128
129 void MyCoin::mousePressEvent(QMouseEvent *e)
130 {
131     //如果进动画 或者 胜利 或者 改变周围方块时 将不能点击按钮
132     if(this->m_isAnimation || this->m_iswin || this->m_judge)
133         return;
134     else
135         QPushButton::mousePressEvent(e);
136 }
137
138
139

```

Fence 43

## 12.6 关卡数据类

```

1  //dataconfig,h
2  #ifndef DATACONFIG_H
3  #define DATACONFIG_H
4
5  #include <QObject>
6  #include <QMap>
7  #include <QVector>
8
9  class dataConfig : public QObject
10 {
11     Q_OBJECT

```



```

12 public:
13     explicit dataConfig(QObject *parent = nullptr);
14
15
16
17     QMap<int, QVector< QVector<int>>> m_data; //map中的int表示关卡号
18     QVector< QVector<int>>表示金币的二维数组
19 signals:
20 };
21 #endif // DATACONFIG_H
22
23
24 //dataconfig.cpp
25 #include "dataconfig.h"
26
27
28 dataConfig::dataConfig(QObject *parent)
29     : QObject{parent}
30 {
31     //创建第一关的模板
32     //1为金币 0为银币
33     int array1[4][4] = {{1, 1, 1, 1},
34                          {1, 1, 0, 1},
35                          {1, 0, 0, 0},
36                          {1, 1, 0, 1} } ;
37
38     QVector< QVector<int>> v;
39     for(int i = 0 ; i < 4;i++)
40     {
41         QVector<int>v1;
42         for(int j = 0 ; j < 4;j++)
43         {
44
45             v1.push_back(array1[i][j]);
46         }
47         v.push_back(v1);
48     }
49
50     m_data.insert(1,v);
51
52
53     int array2[4][4] = { {1, 0, 1, 1},
54                          {0, 0, 1, 1},
55                          {1, 1, 0, 0},
56                          {1, 1, 0, 1} } ;
57
58     v.clear();
59     for(int i = 0 ; i < 4;i++)
60     {
61         QVector<int>v1;
62         for(int j = 0 ; j < 4;j++)
63         {

```

```

64         v1.push_back(array2[i][j]);
65     }
66     v.push_back(v1);
67 }
68
69 m_data.insert(2,v);
70
71
72
73 int array3[4][4] = { {0, 0, 0, 0},
74                     {0, 1, 1, 0},
75                     {0, 1, 1, 0},
76                     {0, 0, 0, 0}} ;
77 v.clear();
78 for(int i = 0 ; i < 4;i++)
79 {
80     QVector<int>v1;
81     for(int j = 0 ; j < 4;j++)
82     {
83         v1.push_back(array3[i][j]);
84     }
85     v.push_back(v1);
86 }
87
88 m_data.insert(3,v);
89
90
91 int array4[4][4] = { {0, 1, 1, 1},
92                     {1, 0, 0, 1},
93                     {1, 0, 1, 1},
94                     {1, 1, 1, 1}} ;
95 v.clear();
96 for(int i = 0 ; i < 4;i++)
97 {
98     QVector<int>v1;
99     for(int j = 0 ; j < 4;j++)
100     {
101         v1.push_back(array4[i][j]);
102     }
103     v.push_back(v1);
104 }
105
106 m_data.insert(4,v);
107
108
109 int array5[4][4] = { {1, 0, 0, 1},
110                     {0, 0, 0, 0},
111                     {0, 0, 0, 0},
112                     {1, 0, 0, 1}} ;
113 v.clear();
114 for(int i = 0 ; i < 4;i++)
115 {
116     QVector<int>v1;

```

```
117         for(int j = 0 ; j < 4;j++)
118         {
119             v1.push_back(array5[i][j]);
120         }
121         v.push_back(v1);
122     }
123
124     m_data.insert(5,v);
125
126
127     int array6[4][4] = {    {1, 0, 0, 1},
128                             {0, 1, 1, 0},
129                             {0, 1, 1, 0},
130                             {1, 0, 0, 1}} ;
131
132     v.clear();
133     for(int i = 0 ; i < 4;i++)
134     {
135         QVector<int>v1;
136         for(int j = 0 ; j < 4;j++)
137         {
138             v1.push_back(array6[i][j]);
139         }
140         v.push_back(v1);
141     }
142
143     m_data.insert(6,v);
144
145
146     int array7[4][4] = {    {0, 1, 1, 1},
147                             {1, 0, 1, 1},
148                             {1, 1, 0, 1},
149                             {1, 1, 1, 0}} ;
150
151     v.clear();
152     for(int i = 0 ; i < 4;i++)
153     {
154         QVector<int>v1;
155         for(int j = 0 ; j < 4;j++)
156         {
157             v1.push_back(array7[i][j]);
158         }
159         v.push_back(v1);
160     }
161
162     m_data.insert(7,v);
163
164
165     int array8[4][4] = {    {0, 1, 0, 1},
166                             {1, 0, 0, 0},
167                             {0, 0, 0, 1},
168                             {1, 0, 1, 0}} ;
169
170     v.clear();
171     for(int i = 0 ; i < 4;i++)
172     {
173         QVector<int>v1;
```

```
170         for(int j = 0 ; j < 4;j++)
171         {
172             v1.push_back(array8[i][j]);
173         }
174         v.push_back(v1);
175     }
176
177     m_data.insert(8,v);
178
179     int array9[4][4] = {    {1, 0, 1, 0},
180                           {1, 0, 1, 0},
181                           {0, 0, 1, 0},
182                           {1, 0, 0, 1}} ;
183     v.clear();
184     for(int i = 0 ; i < 4;i++)
185     {
186         QVector<int>v1;
187         for(int j = 0 ; j < 4;j++)
188         {
189             v1.push_back(array9[i][j]);
190         }
191         v.push_back(v1);
192     }
193
194     m_data.insert(9,v);
195
196
197
198     int array10[4][4] = {    {1, 0, 1, 1},
199                             {1, 1, 0, 0},
200                             {0, 0, 1, 1},
201                             {1, 1, 0, 1}} ;
202     v.clear();
203     for(int i = 0 ; i < 4;i++)
204     {
205         QVector<int>v1;
206         for(int j = 0 ; j < 4;j++)
207         {
208             v1.push_back(array10[i][j]);
209         }
210         v.push_back(v1);
211     }
212
213     m_data.insert(10,v);
214
215
216     int array11[4][4] = {    {0, 1, 1, 0},
217                             {1, 0, 0, 1},
218                             {1, 0, 0, 1},
219                             {0, 1, 1, 0}} ;
220     v.clear();
221     for(int i = 0 ; i < 4;i++)
222     {
```

```

223     QVector<int>v1;
224     for(int j = 0 ; j < 4;j++)
225     {
226         v1.push_back(array11[i][j]);
227     }
228     v.push_back(v1);
229 }
230
231 m_data.insert(11,v);
232
233 int array12[4][4] = { {0, 1, 1, 0},
234                      {0, 0, 0, 0},
235                      {1, 1, 1, 1},
236                      {0, 0, 0, 0}} ;
237 v.clear();
238 for(int i = 0 ; i < 4;i++)
239 {
240     QVector<int>v1;
241     for(int j = 0 ; j < 4;j++)
242     {
243         v1.push_back(array12[i][j]);
244     }
245     v.push_back(v1);
246 }
247
248 m_data.insert(12,v);
249
250
251 int array13[4][4] = { {0, 1, 1, 0},
252                      {0, 0, 0, 0},
253                      {0, 0, 0, 0},
254                      {0, 1, 1, 0}} ;
255 v.clear();
256 for(int i = 0 ; i < 4;i++)
257 {
258     QVector<int>v1;
259     for(int j = 0 ; j < 4;j++)
260     {
261         v1.push_back(array13[i][j]);
262     }
263     v.push_back(v1);
264 }
265
266 m_data.insert(13,v);
267
268 int array14[4][4] = { {1, 0, 1, 1},
269                      {0, 1, 0, 1},
270                      {1, 0, 1, 0},
271                      {1, 1, 0, 1}} ;
272 v.clear();
273 for(int i = 0 ; i < 4;i++)
274 {
275     QVector<int>v1;

```

```

276         for(int j = 0 ; j < 4;j++)
277         {
278             v1.push_back(array14[i][j]);
279         }
280         v.push_back(v1);
281     }
282
283     m_data.insert(14,v);
284
285
286     int array15[4][4] = {    {0, 1, 0, 1},
287                             {1, 0, 0, 0},
288                             {1, 0, 0, 0},
289                             {0, 1, 0, 1}} ;
290
291     v.clear();
292     for(int i = 0 ; i < 4;i++)
293     {
294         QVector<int>v1;
295         for(int j = 0 ; j < 4;j++)
296         {
297             v1.push_back(array15[i][j]);
298         }
299         v.push_back(v1);
300     }
301
302     m_data.insert(15,v);
303
304
305     int array16[4][4] = {    {0, 1, 1, 0},
306                             {1, 1, 1, 1},
307                             {1, 1, 1, 1},
308                             {0, 1, 1, 0}} ;
309
310     v.clear();
311     for(int i = 0 ; i < 4;i++)
312     {
313         QVector<int>v1;
314         for(int j = 0 ; j < 4;j++)
315         {
316             v1.push_back(array16[i][j]);
317         }
318         v.push_back(v1);
319     }
320
321     m_data.insert(16,v);
322
323
324     int array17[4][4] = {    {0, 1, 1, 1},
325                             {0, 1, 0, 0},
326                             {0, 0, 1, 0},
327                             {1, 1, 1, 0}} ;
328
329     v.clear();
330     for(int i = 0 ; i < 4;i++)
331     {
332         QVector<int>v1;

```

```

329         for(int j = 0 ; j < 4;j++)
330         {
331             v1.push_back(array17[i][j]);
332         }
333         v.push_back(v1);
334     }
335
336     m_data.insert(17,v);
337
338
339     int array18[4][4] = { {0, 0, 0, 1},
340                           {0, 0, 1, 0},
341                           {0, 1, 0, 0},
342                           {1, 0, 0, 0}} ;
343
344     v.clear();
345     for(int i = 0 ; i < 4;i++)
346     {
347         QVector<int>v1;
348         for(int j = 0 ; j < 4;j++)
349         {
350             v1.push_back(array18[i][j]);
351         }
352         v.push_back(v1);
353     }
354
355     m_data.insert(18,v);
356
357     int array19[4][4] = { {0, 1, 0, 0},
358                           {0, 1, 1, 0},
359                           {0, 0, 1, 1},
360                           {0, 0, 0, 0}} ;
361
362     v.clear();
363     for(int i = 0 ; i < 4;i++)
364     {
365         QVector<int>v1;
366         for(int j = 0 ; j < 4;j++)
367         {
368             v1.push_back(array19[i][j]);
369         }
370         v.push_back(v1);
371     }
372
373     m_data.insert(19,v);
374
375     int array20[4][4] = { {0, 0, 0, 0},
376                           {0, 0, 0, 0},
377                           {0, 0, 0, 0},
378                           {0, 0, 0, 0}} ;
379
380     v.clear();
381     for(int i = 0 ; i < 4;i++)
382     {
383         QVector<int>v1;
384         for(int j = 0 ; j < 4;j++)

```

```

382     {
383         v1.push_back(array20[i][j]);
384     }
385     v.push_back(v1);
386 }
387
388 m_data.insert(20,v);
389
390
391 }
392

```

Fence 44

## 13. 打包

### 13.1 步骤1

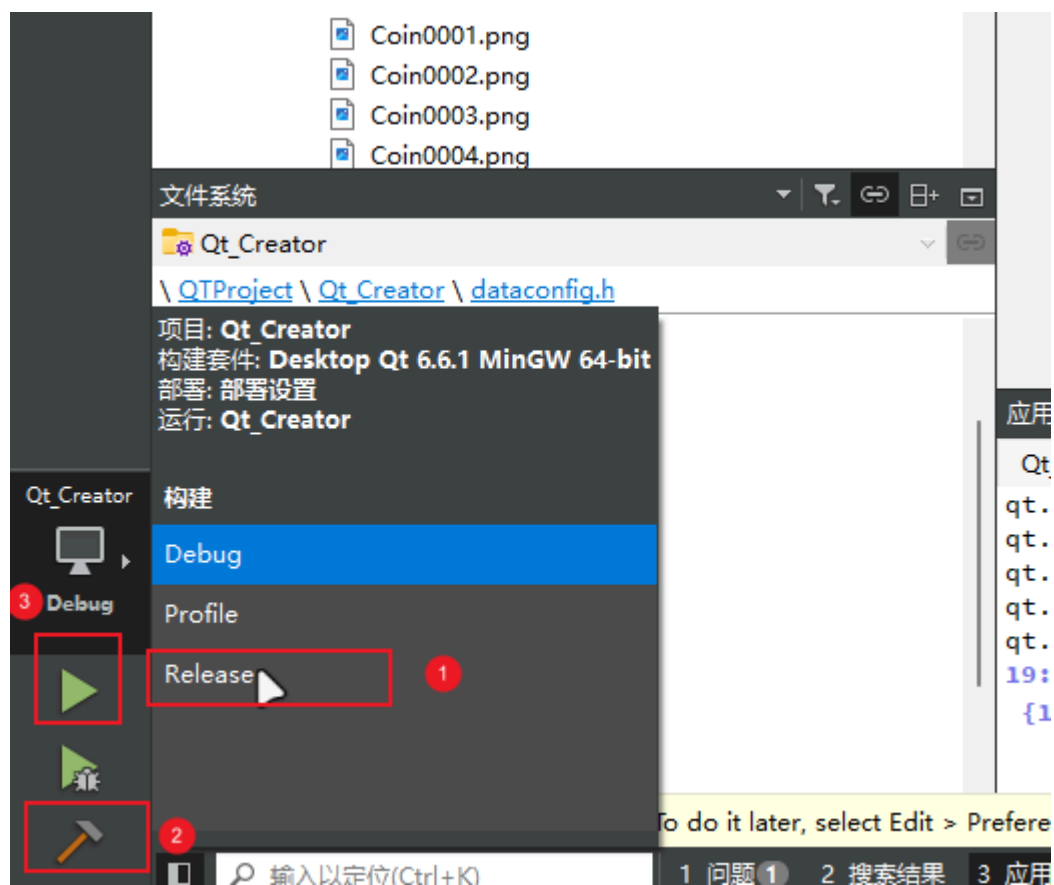


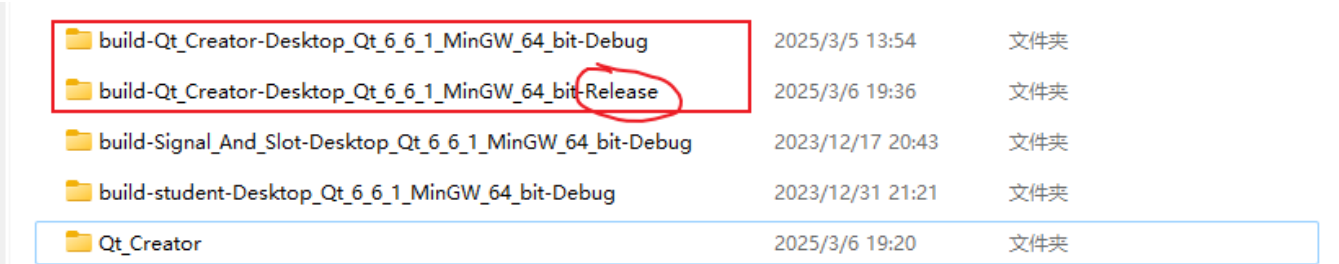
Figure 42

记得运行程序，检查是否正常。



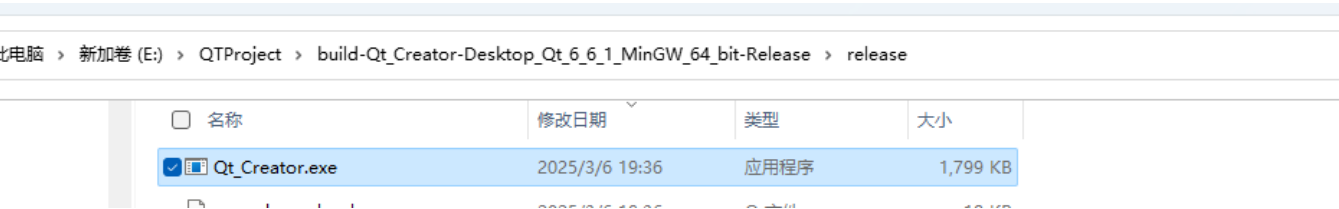
### 13.2 步骤2(只做到这一步，需要使用者有对应的qt环境)

在项目的上一级目录，有Debug版本和Release版本，复制Release文件夹里面的.exe文件



build-Qt_Creator-Desktop_Qt_6_6_1_MinGW_64_bit-Debug	2025/3/5 13:54	文件夹
build-Qt_Creator-Desktop_Qt_6_6_1_MinGW_64_bit-Release	2025/3/6 19:36	文件夹
build-Signal_And_Slot-Desktop_Qt_6_6_1_MinGW_64_bit-Debug	2023/12/17 20:43	文件夹
build-student-Desktop_Qt_6_6_1_MinGW_64_bit-Debug	2023/12/31 21:21	文件夹
Qt_Creator	2025/3/6 19:20	文件夹

Figure 43



电脑 > 新加卷 (E:) > QTProject > build-Qt\_Creator-Desktop\_Qt\_6\_6\_1\_MinGW\_64\_bit-Release > release

<input type="checkbox"/> 名称	修改日期	类型	大小
<input checked="" type="checkbox"/> Qt_Creator.exe	2025/3/6 19:36	应用程序	1,799 KB

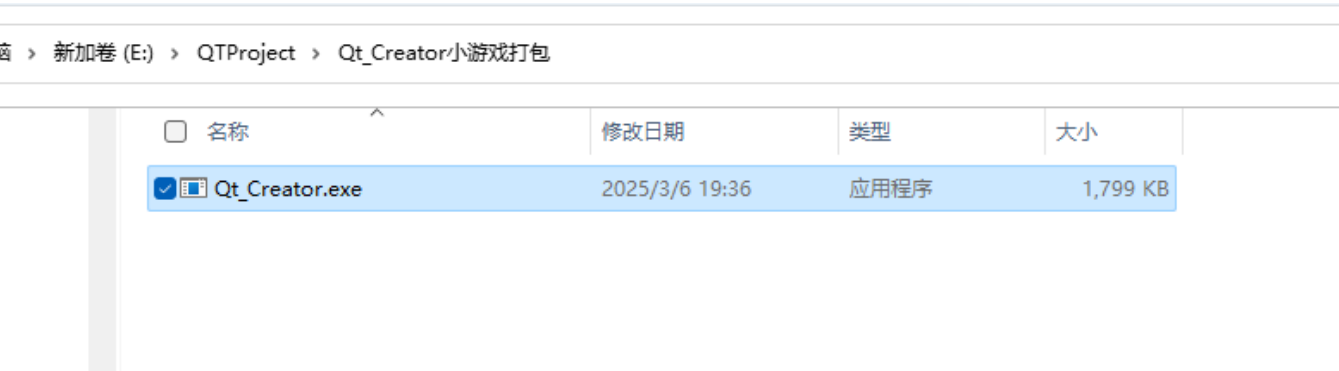
Figure 44

### 13.3 步骤3(这一步之后，就可以将这个文件夹压缩，发给使用者)

将复制的.exe文件,复制到新建的一个文件夹中

在该文件夹下，打开cmd 输入 windeployqt.exe 项目.exe 例如：windeployqt.exe Qt\_Creator.exe

确保在qt安装目录下能找到windeployqt.exe程序，E:\Qt\6.6.1\mingw\_64\bin\windeployqt.exe



商 > 新加卷 (E:) > QTProject > Qt\_Creator小游戏打包

<input type="checkbox"/> 名称	修改日期	类型	大小
<input checked="" type="checkbox"/> Qt_Creator.exe	2025/3/6 19:36	应用程序	1,799 KB

Figure 45

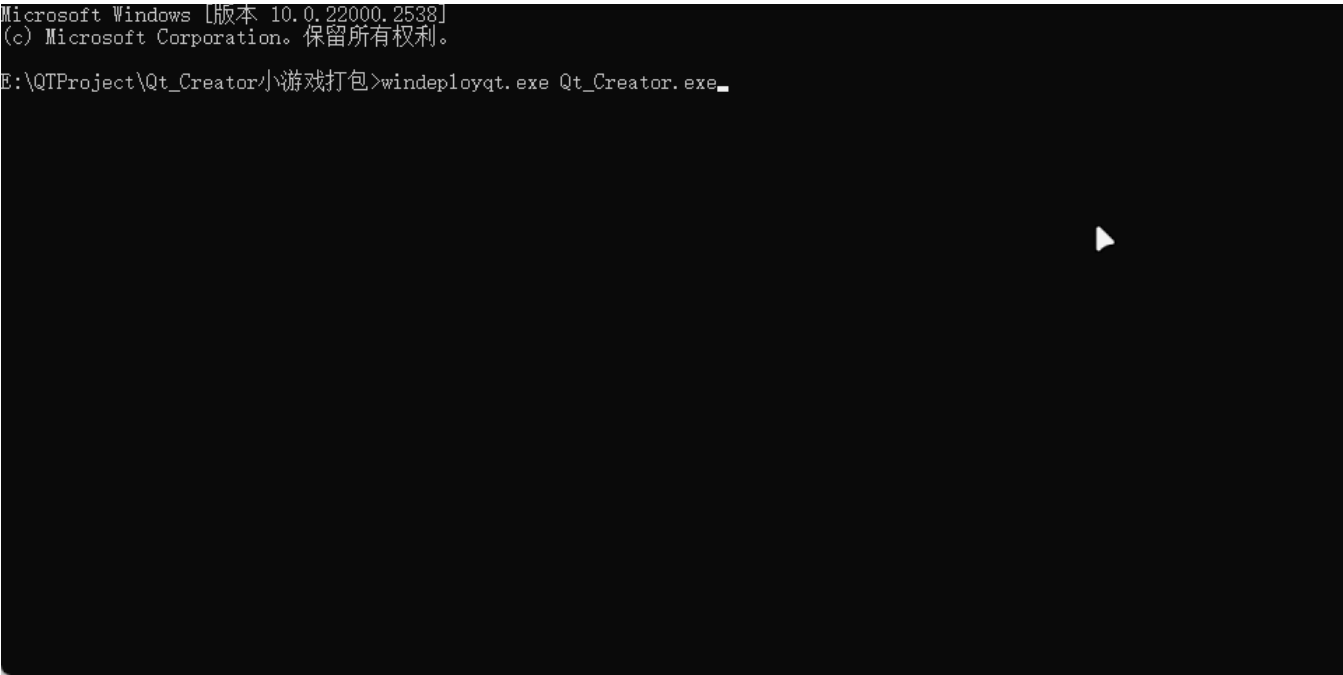


Figure 46

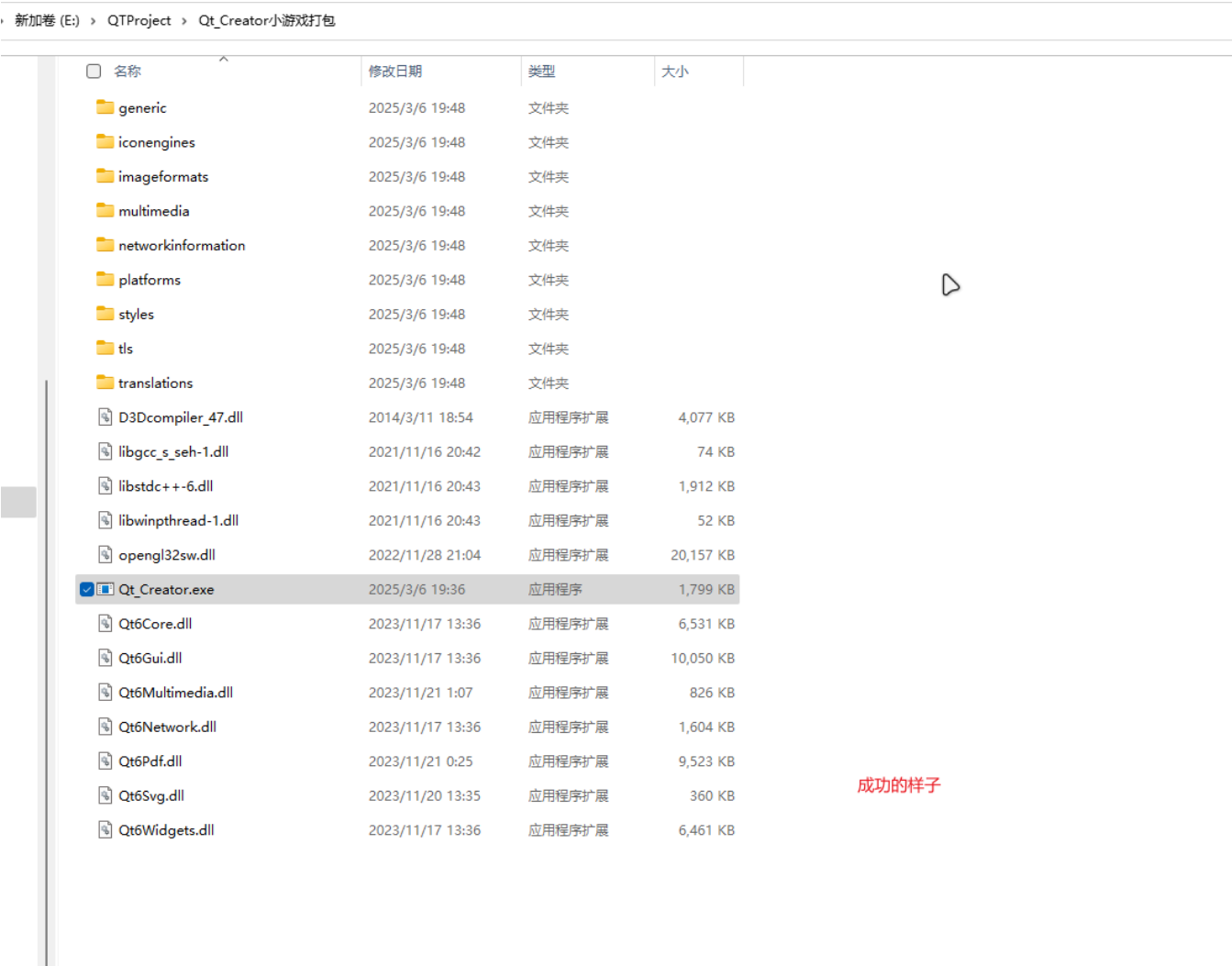


Figure 47

### 13.4 步骤4（可省略 使用第三方，形成安装程序 固定生成的是setup.exe的安装包）

- 1. 安装nsis与hm nis edit第三方打包软件
- 2. 启动hm nis edit

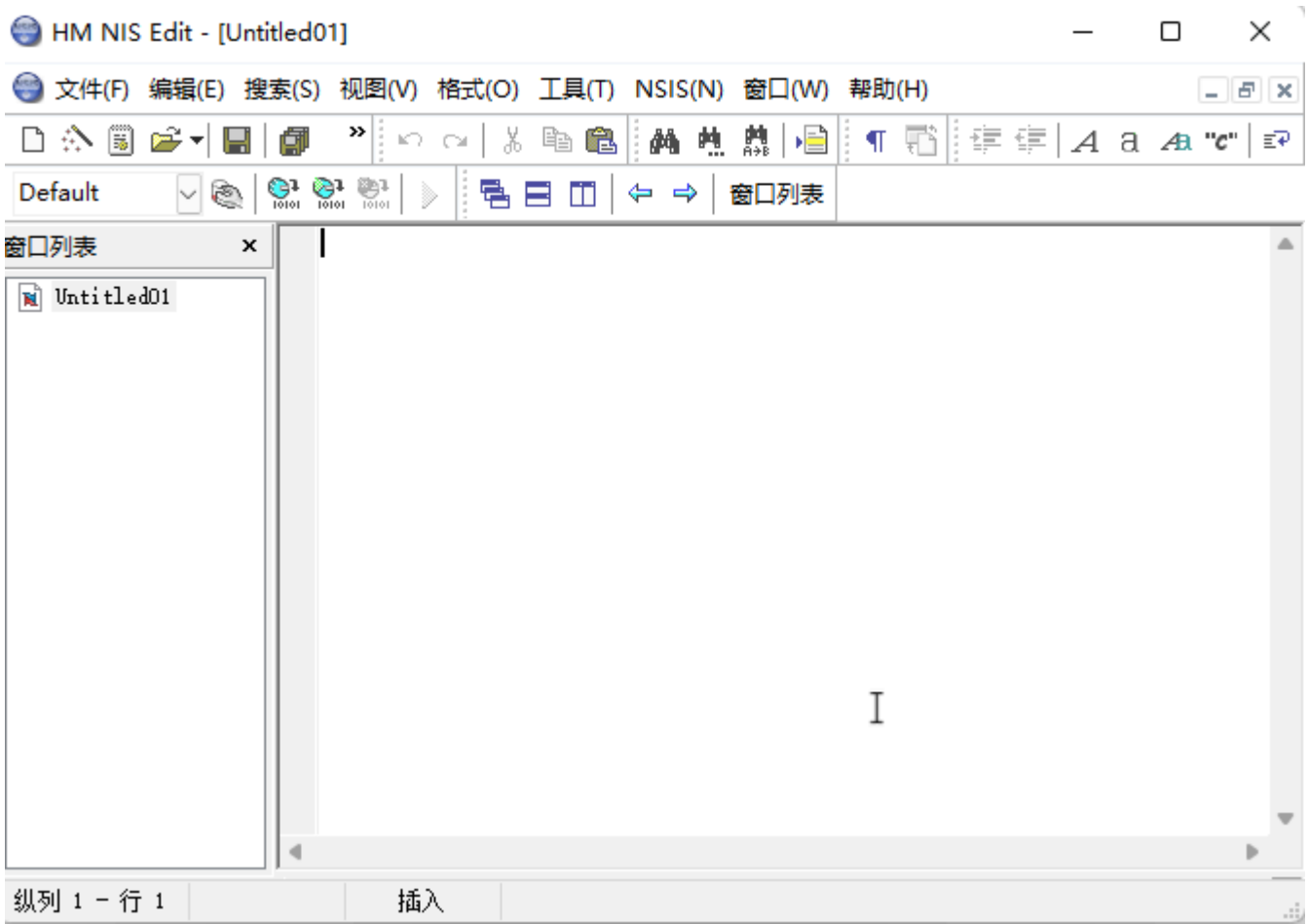


Figure 48

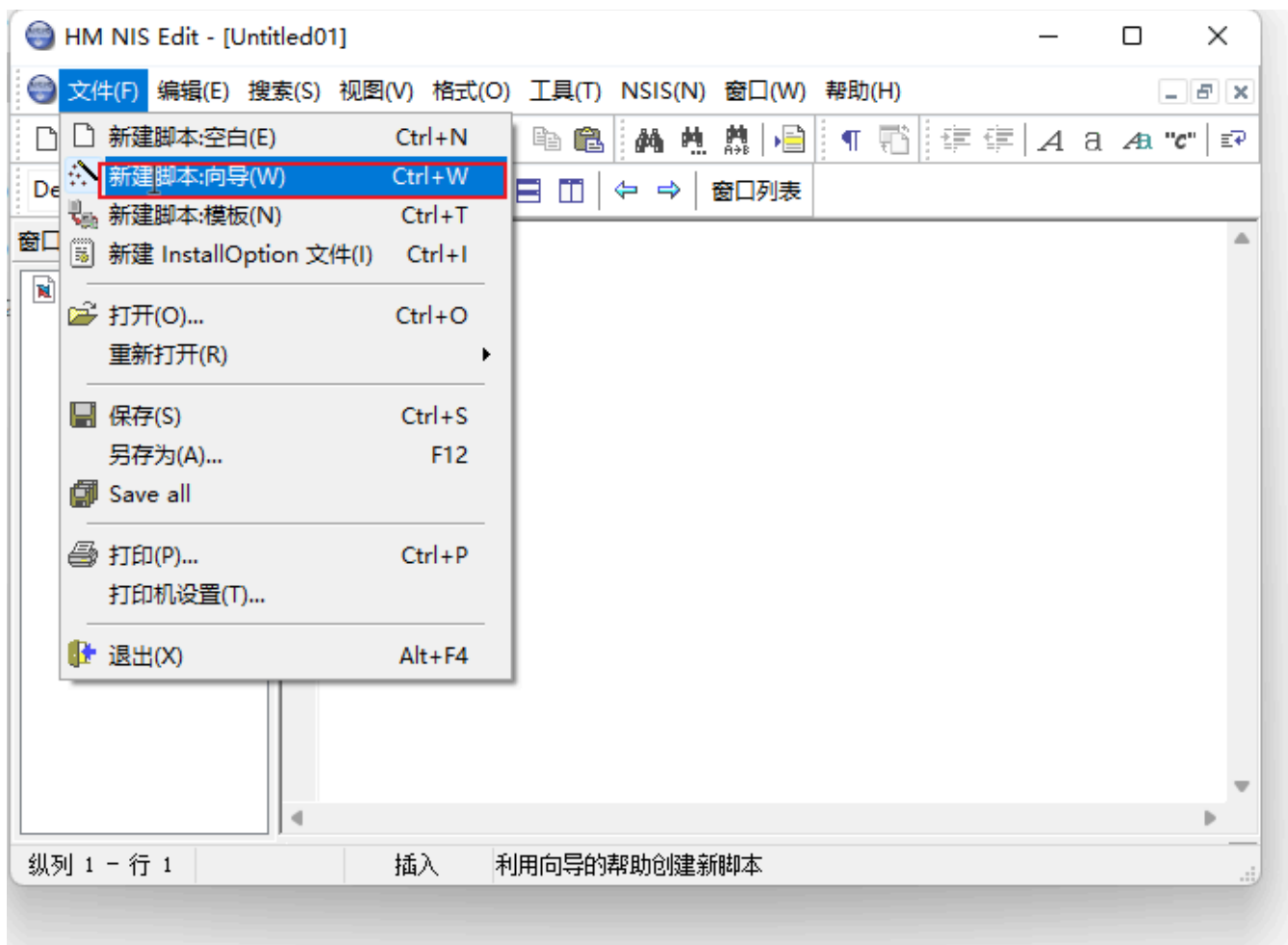


Figure 49

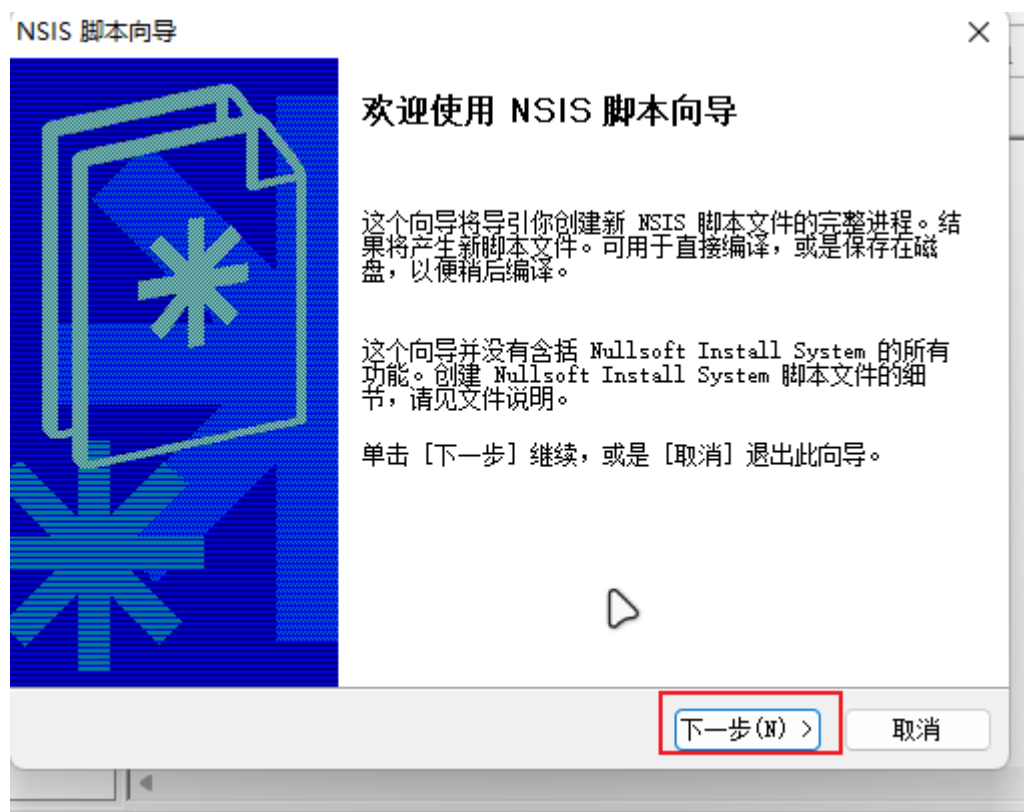


Figure 50

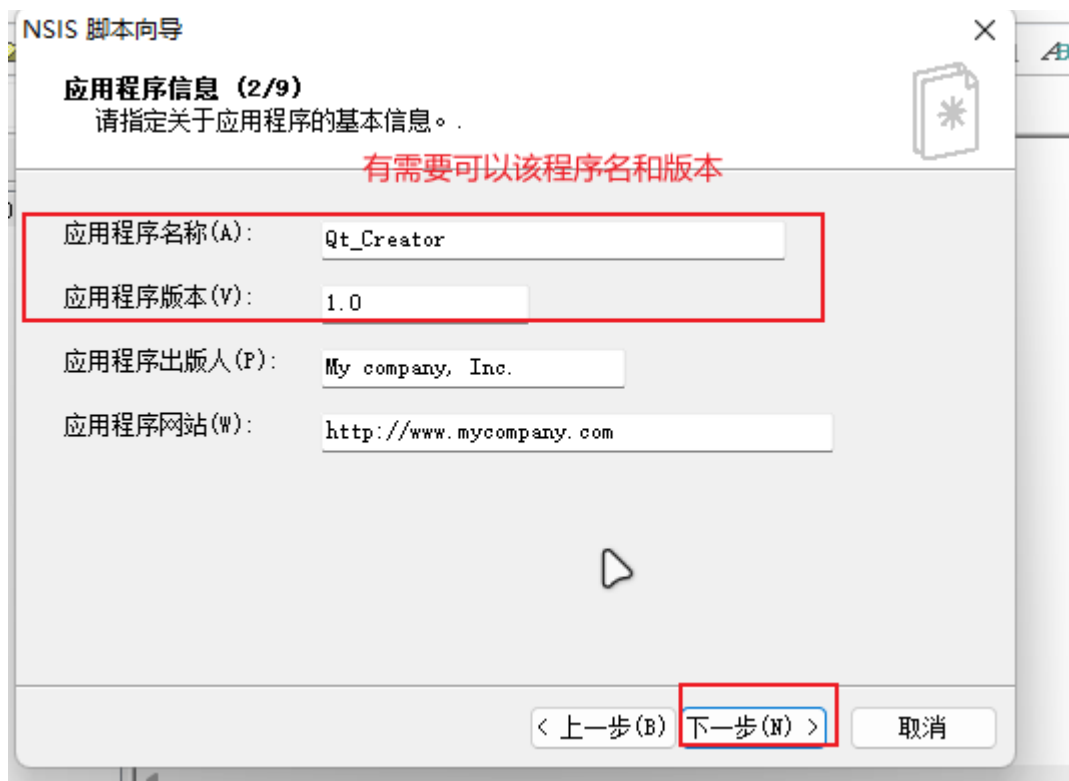


Figure 51



Figure 52

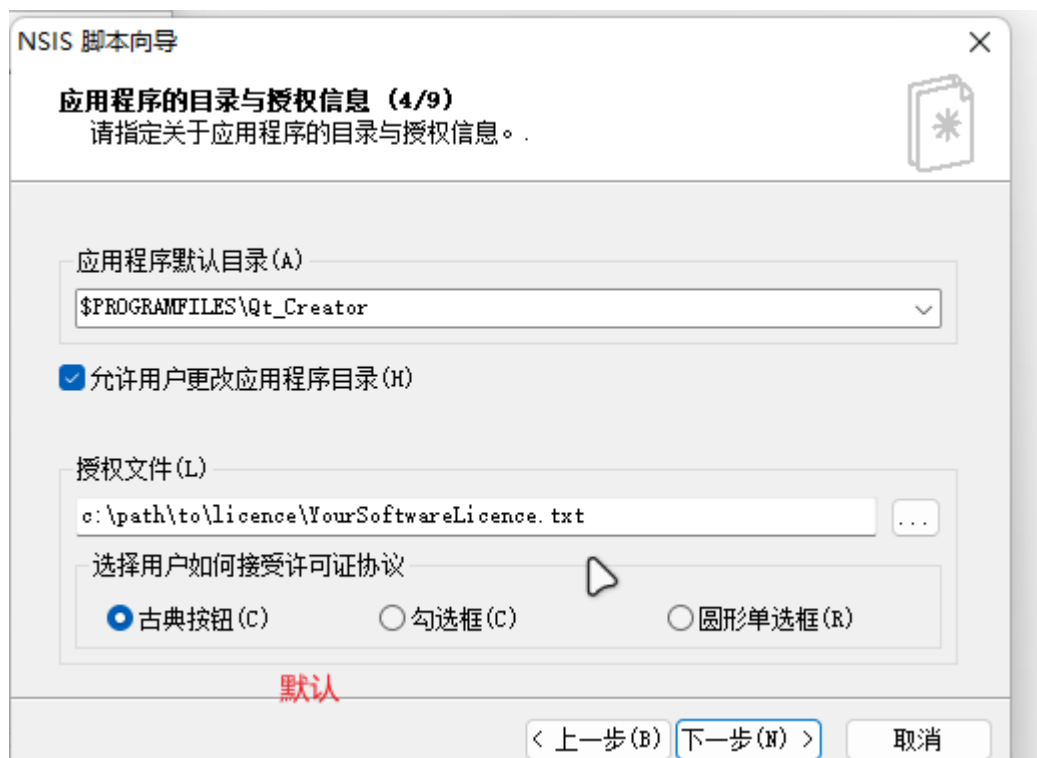


Figure 53

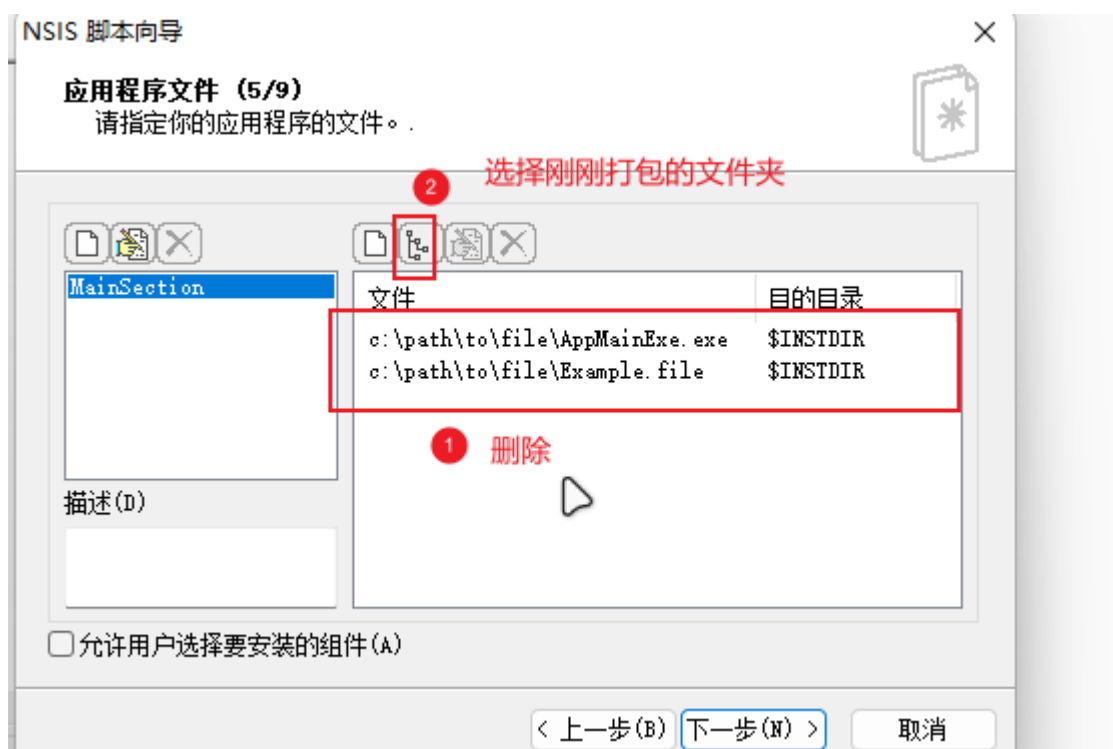


Figure 54

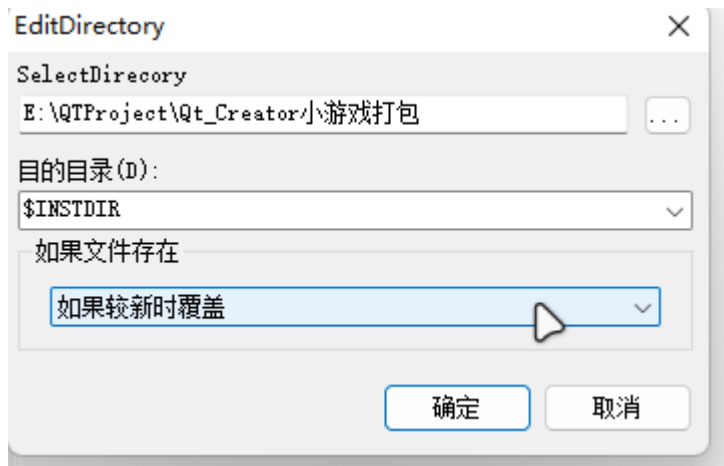


Figure 55



Figure 56



Figure 57

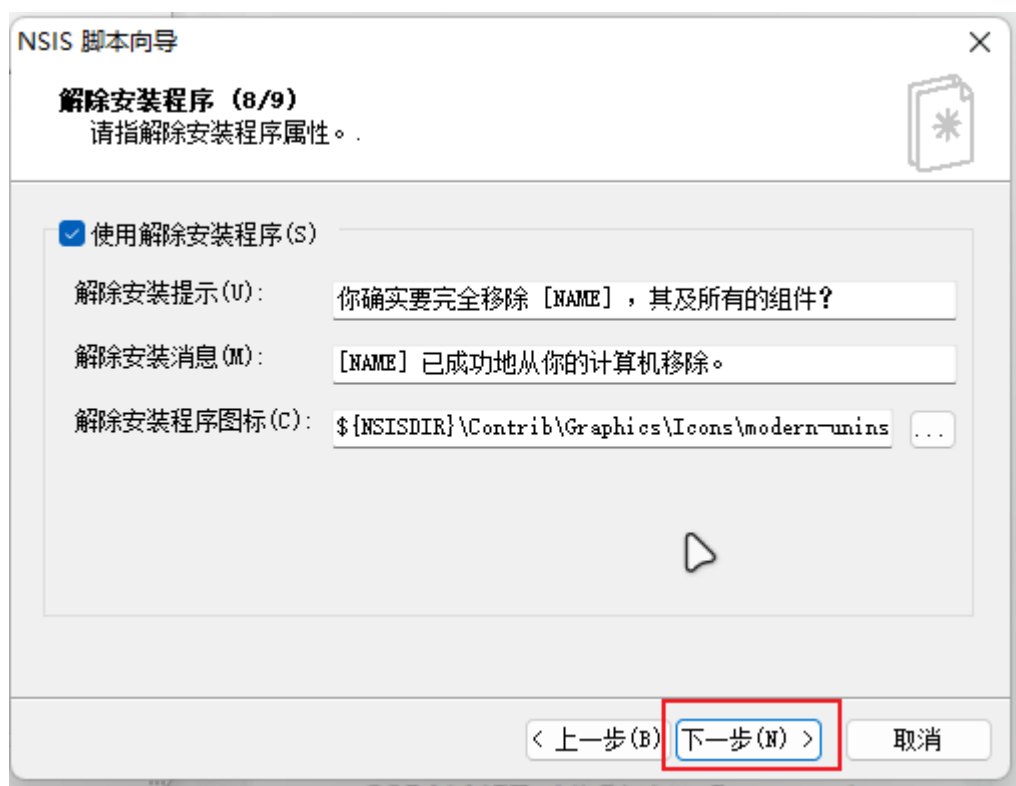


Figure 58





Figure 59

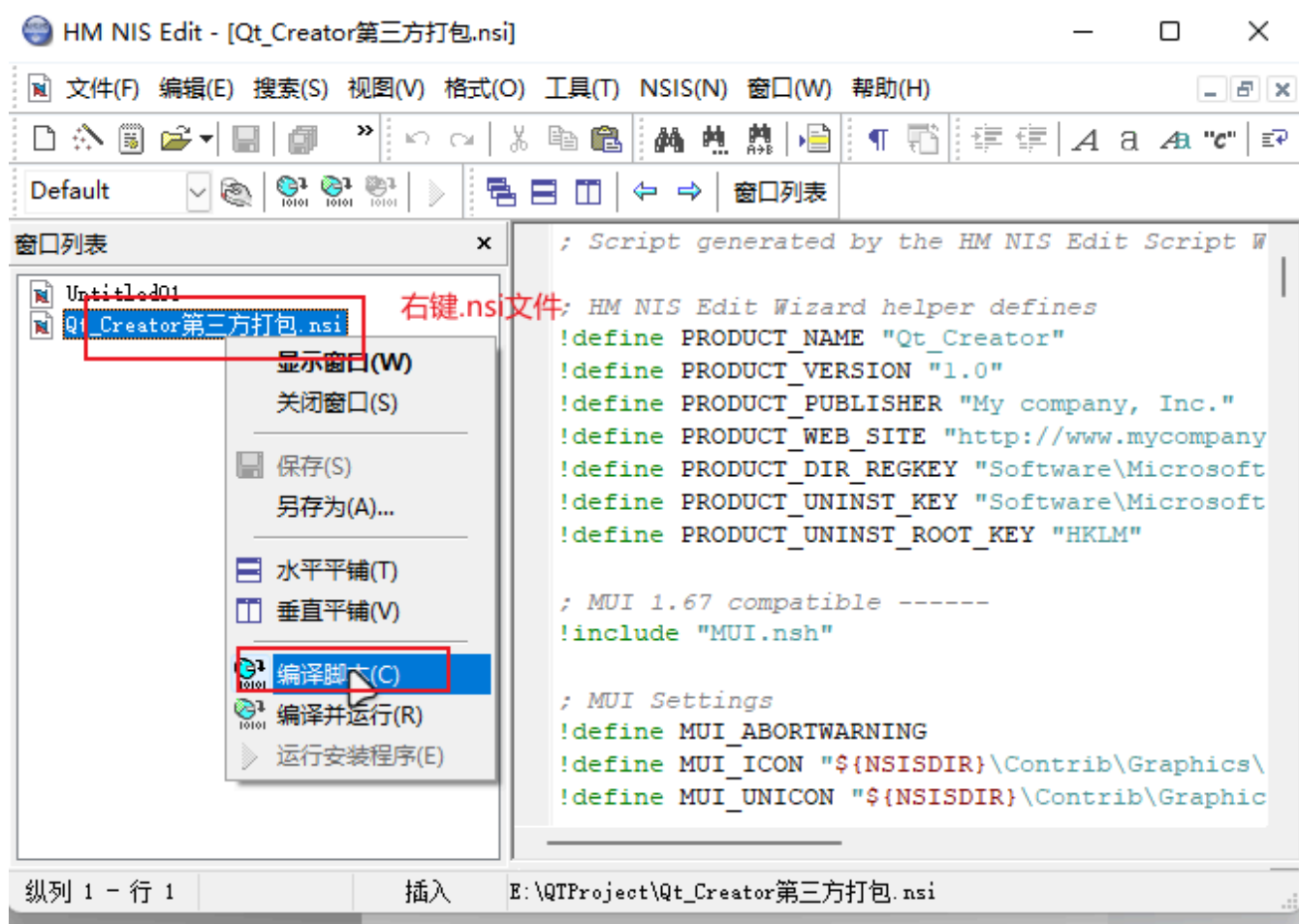


Figure 60

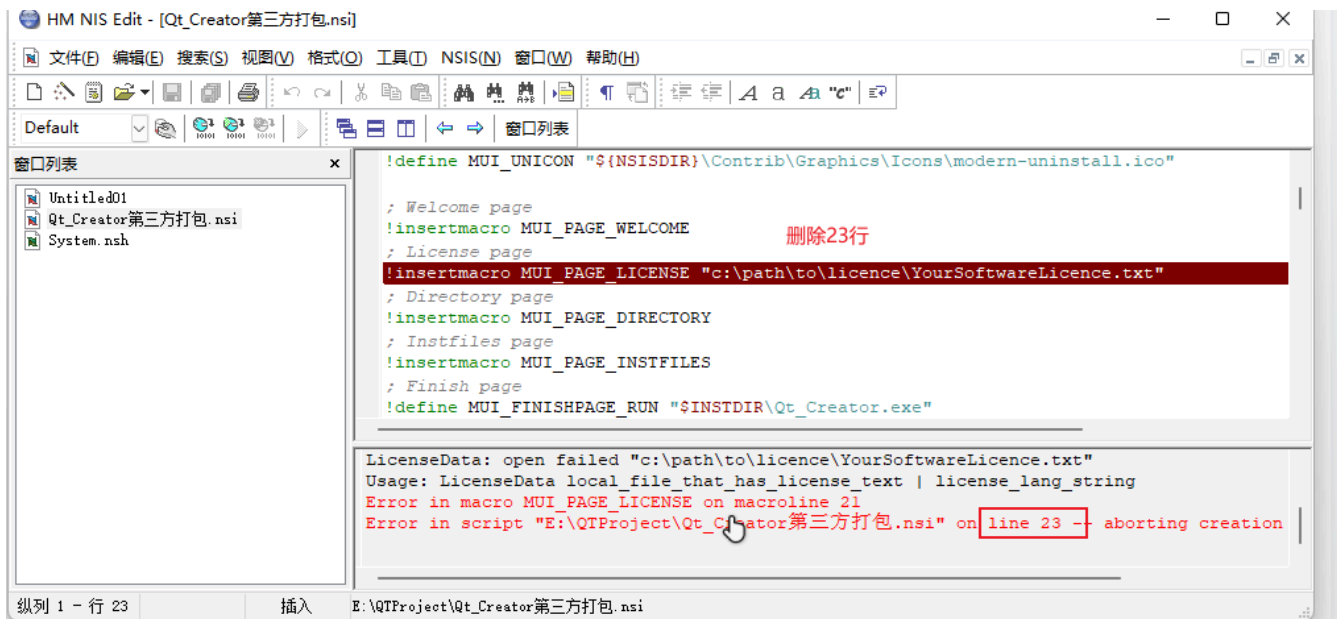


Figure 61

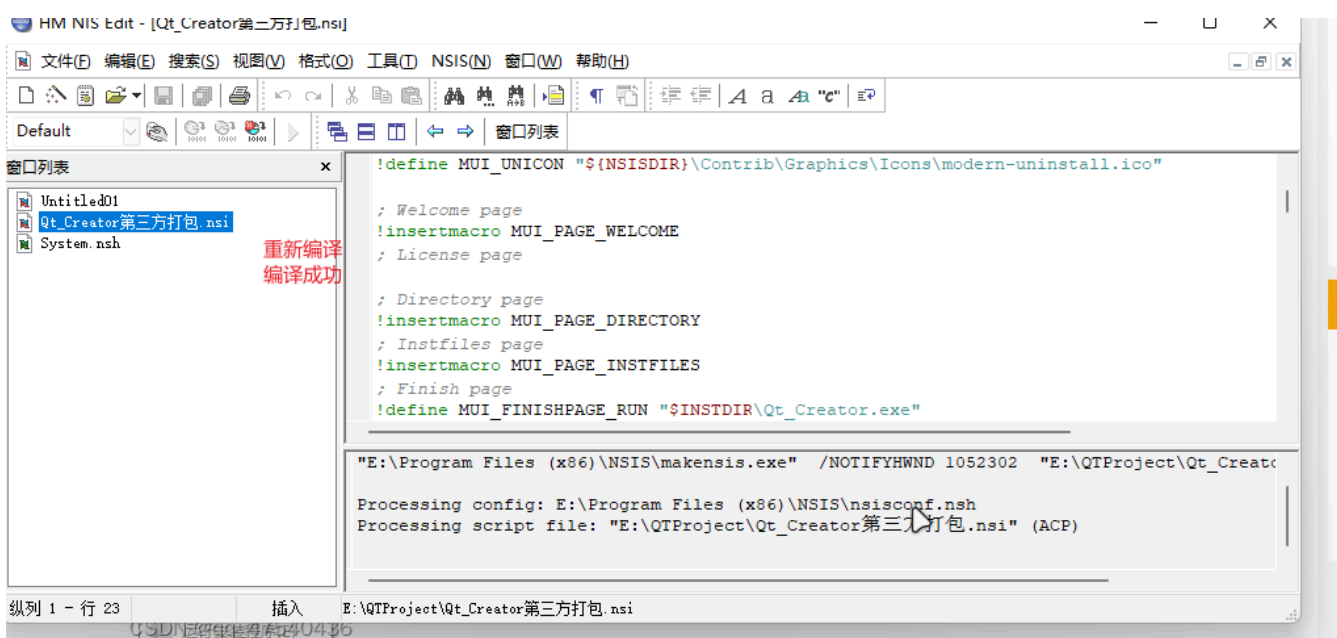


Figure 62

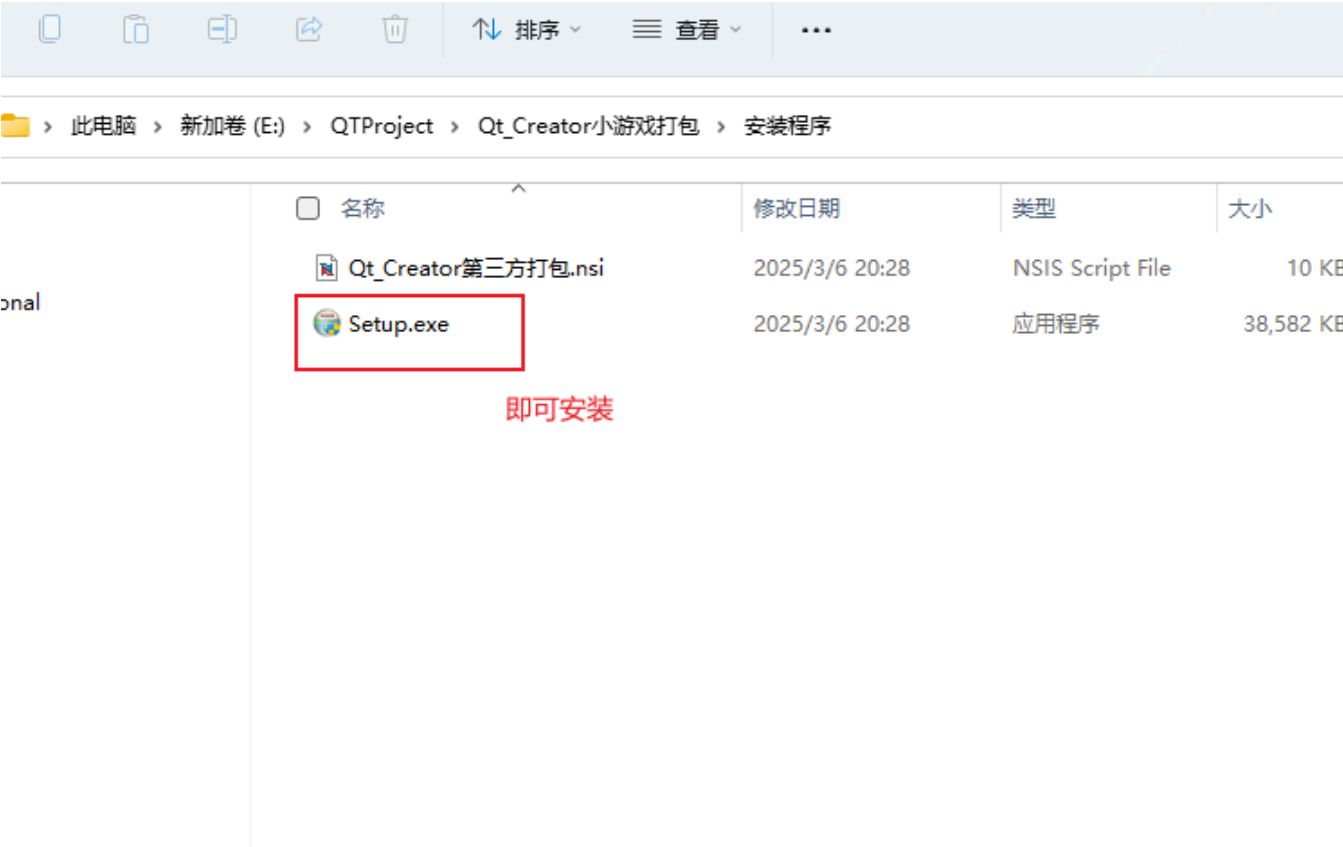


Figure 63