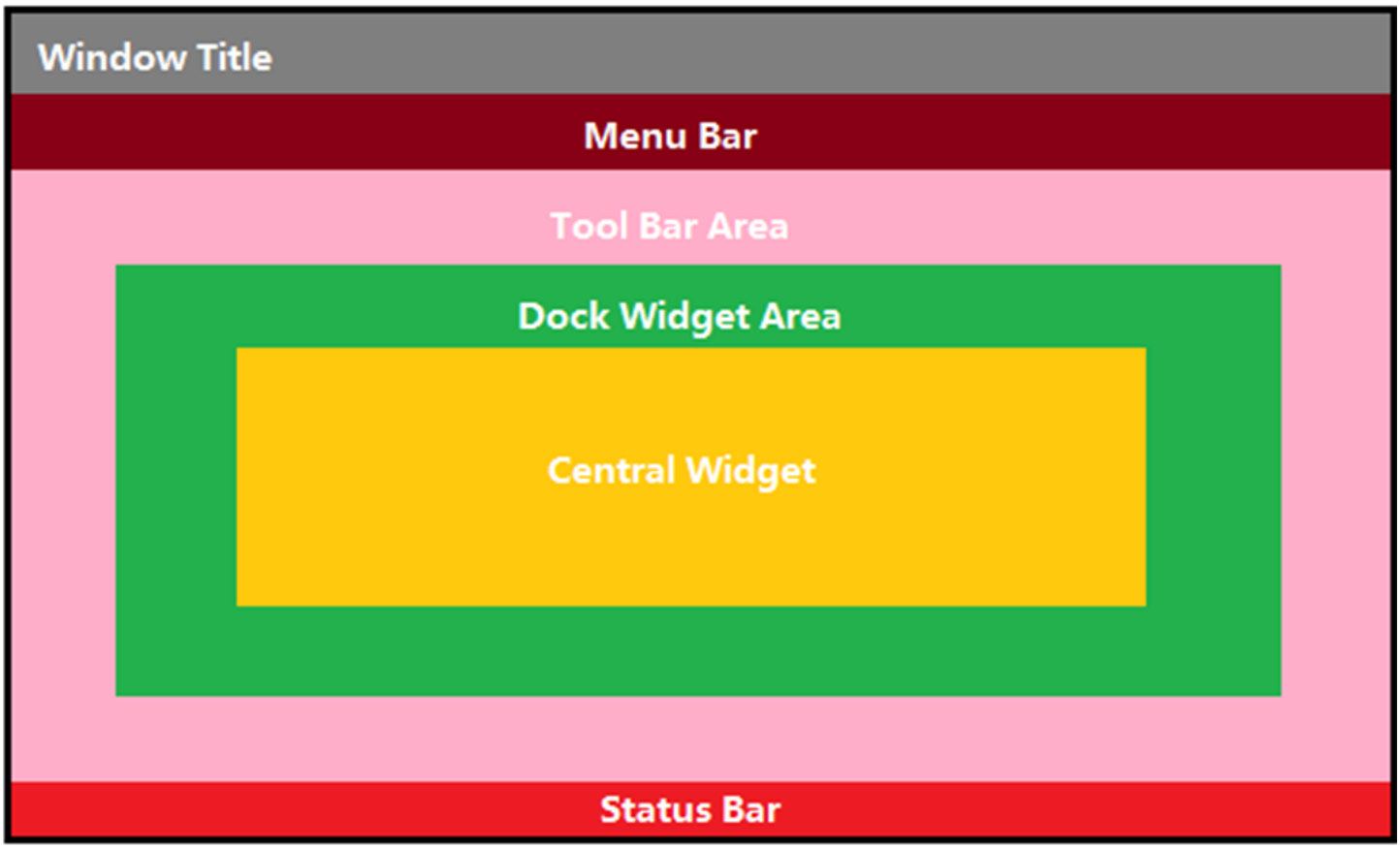
第四章 Qt窗口

Qt 窗口 是通过 QMainWindow类 来实现的。

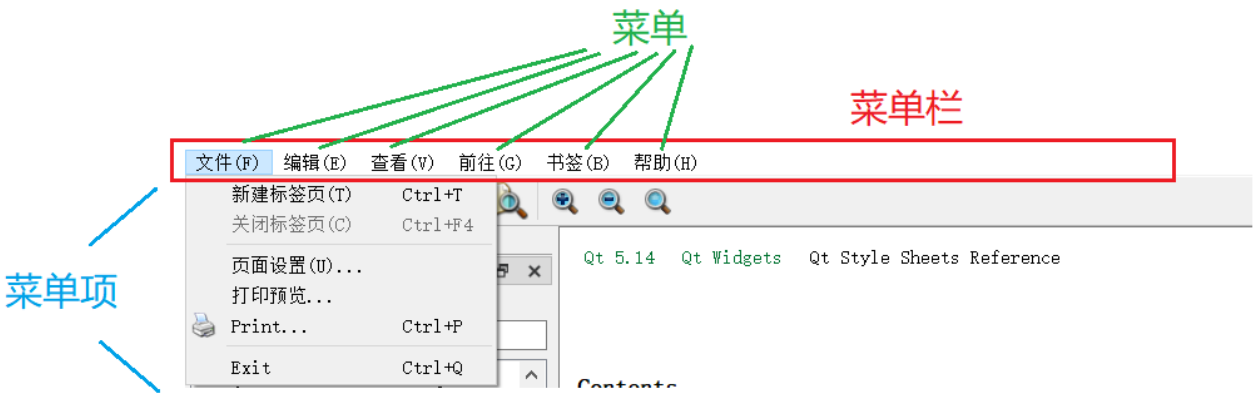
QMainWindow 是一个为用户提供主窗口程序的类，继承自 **QWidget** 类，并且提供了一个预定义的布局。**QMainWindow** 包含 **一个菜单栏（menu bar）、多个工具栏(tool bars)、多个浮动窗口（铆接部件）(dock widgets)、一个状态栏(status bar)** 和一个 **中心部件(central widget)**，它是许多应用程序的基础，如文本编辑器，图片编辑器等。如下图为 **QMainwindow** 中 各组件所处的位置：



1. 菜单栏

Qt 中的**菜单栏**是通过 **QMenuBar** 这个类来实现的。**一个主窗口最多只有一个菜单栏**。位于主窗口顶部、主窗口标题栏下面。

菜单栏中包含菜单. 菜单中包含菜单项.



1.1 创建菜单栏

方式一：菜单栏的创建可以借助于 **QMainWindow类** 提供的 **menuBar() 函数**来实现。menubar()函数原型如下：

```
QMenuBar * menuBar() const
```

```
//创建菜单栏
QMenuBar *menubar = menuBar();

//将菜单栏放入窗口中
this->setMenuBar(menubar);
```

方式二：在堆上动态创建；

```
QMenuBar *menuBar = new QMenuBar(this);

//将菜单栏放入窗口中
this->setMenuBar(menuBar);
```

使用 `setMenuBar` 把菜单栏放到窗口中.

1.2 在菜单栏中添加菜单

创建菜单，并通过 `QMenu` 提供的 `addMenu()` 函数 来添加菜单。

示例：

```
//创建菜单栏
QMenuBar *menubar = menuBar();

//将菜单栏放入窗口中
this->setMenuBar(menubar);

//创建菜单
QMenu *menu1 = new QMenu("文件");
QMenu *menu2 = new QMenu("编辑");
QMenu *menu3 = new QMenu("构建");

//添加菜单到菜单栏中
menubar->addMenu(menu1);
menubar->addMenu(menu2);
menubar->addMenu(menu3);
```

1.3 创建菜单项

在 Qt 中，并没有专门的菜单项类，可以通过 `QAction` 类，抽象出公共的动作。如在菜单中添加菜单项.

> 💡 QAction 可以给菜单栏使用, 也可以给工具栏使用.

示例：

```
//创建菜单项
QAction *act1 = new QAction("open");
QAction *act2 = new QAction("close");
QAction *act3 = new QAction("create");

//将菜单项添加到菜单上
menu1->addAction(act1);
menu1->addAction(act2);
menu1->addAction(act3);
```

1.4 在菜单项之间添加分割线

在菜单项之间可以添加分割线。分割线如下图所示，添加分割线是通过 `QMenu` 类 提供的 `addSeparator()` 函数来实现；

示例：

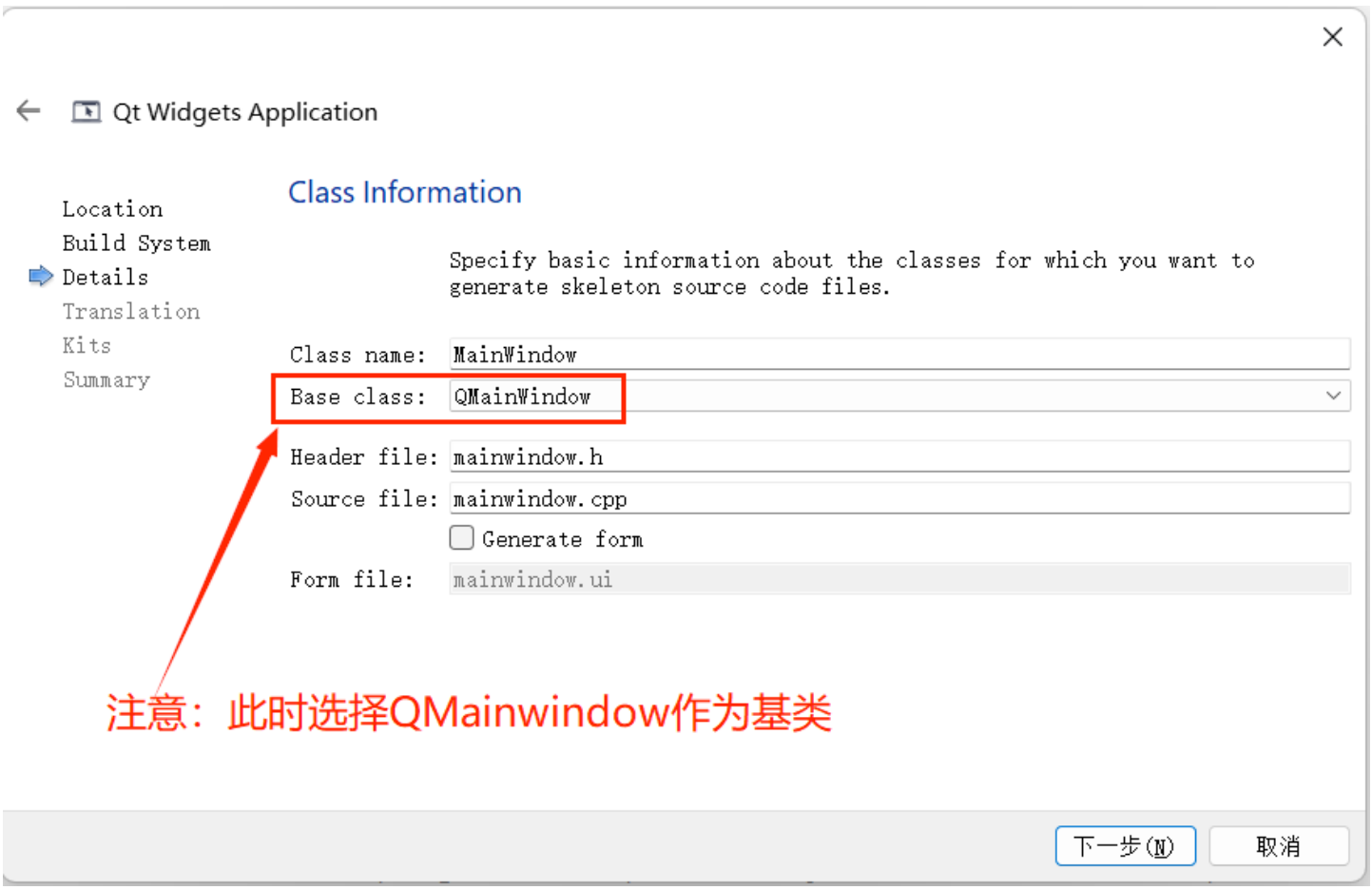
```
//创建菜单项
QAction *act1 = new QAction("open");
QAction *act2 = new QAction("close");
QAction *act3 = new QAction("create");

//将菜单项添加到菜单上
menu1->addAction(act1);
menu1->addSeparator(); //在“open”和“close”中间添加分割线
menu1->addAction(act2);
menu1->addAction(act3);
```

1.5 综合示例

在窗口上创建一个菜单栏，在菜单栏中添加一些菜单，在某一个菜单中添加一些菜单项。

1、新建 Qt 项目

> 注意：此时新建项目时选择的基类 `QMainwindow`，如下图示：



2、在 "mainwindow.cpp" 文件中创建菜单和中央控件

- 创建一个菜单栏, 一个菜单.
- 两个菜单项: 保存, 加载
- 创建一个 `QTextEdit` 作为窗口的中央控件.

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // 设置标题
    this->setWindowTitle("我的记事本");

    // 创建菜单栏
    QMenuBar* menuBar = new QMenuBar(this);
    this->setMenuBar(menuBar);

    // 创建菜单
    QMenu* menu = new QMenu("文件");
    menuBar->addMenu(menu);

    // 创建菜单项
    QAction* action1 = new QAction("保存");
    QAction* action2 = new QAction("加载");
    menu->addAction(action1);
    menu->addAction(action2);

    // 创建中央控件
    edit = new QTextEdit(this);
    this->setCentralWidget(edit);
    edit->setPlaceholderText("此处编写文本内容...");
}
```

3、给 action 添加一些动作

```
// 连接信号槽, 点击 action 时触发一定的效果.
connect(action1, &QAction::triggered, this, &MainWindow::save);
connect(action2, &QAction::triggered, this, &MainWindow::load);
```

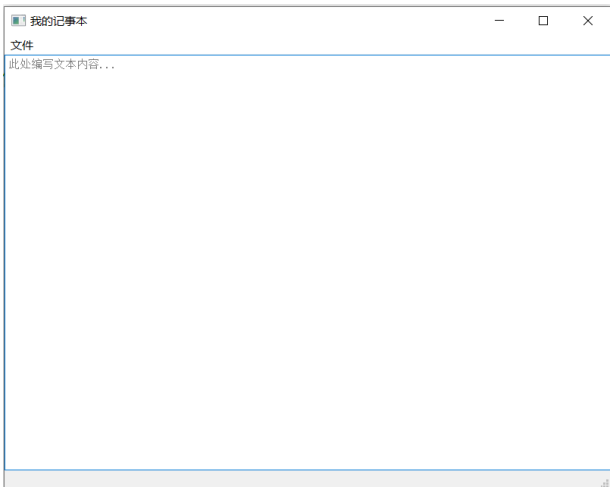
实现这两个槽函数

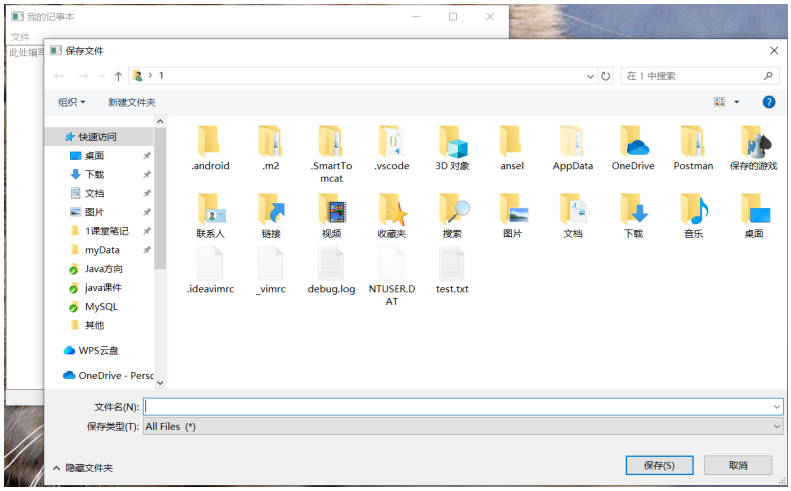
- 使用 `QFileDialog` 来实现选择文件的效果.
  - `getSaveFileName` 用于保存文件的场景. 此时的对话框可以输入文件名.
  - `getOpenFileName` 用于打开文件的场景. 此时的对话框可以获取到鼠标选择的文件名.
- 搭配 C++ 标准库的文件操作实现文件读写.

```
void MainWindow::save()
{
    // 弹出对话框, 选择写入文件的路径
    QFileDialog* dialog = new QFileDialog(this);
    QString fileName = dialog->getSaveFileName(this, "保存文件", "C:/Users/1/");
    qDebug() << "fileName: " << fileName;

    // 写入文件
    std::ofstream file(fileName.toStdString().c_str());
    if (!file.is_open()) {
        qDebug() << "文件保存失败!";
        return;
    }
    const QString& text = edit->toPlainText();
    file << text.toStdString();
    file.close();
}

void MainWindow::load()
{
    // 弹出对话框, 选择打开的文件
    QFileDialog* dialog = new QFileDialog(this);
    QString fileName = dialog->getOpenFileName(this, "加载文件", "C:/Users/1/");
    qDebug() << "fileName: " << fileName;

    // 读取文件
    std::ifstream file(fileName.toStdString().c_str());
    if (!file.is_open()) {
        qDebug() << "文件加载失败!";
        return;
    }
    std::string content;
    std::string line;
    while (std::getline(file, line)) {
        content += line;
        content += "\n";
    }
    file.close();
```

```
        // 显示到界面上
        QString text = QString::fromStdString(content);
        edit->setPlainText(text);
    }
```

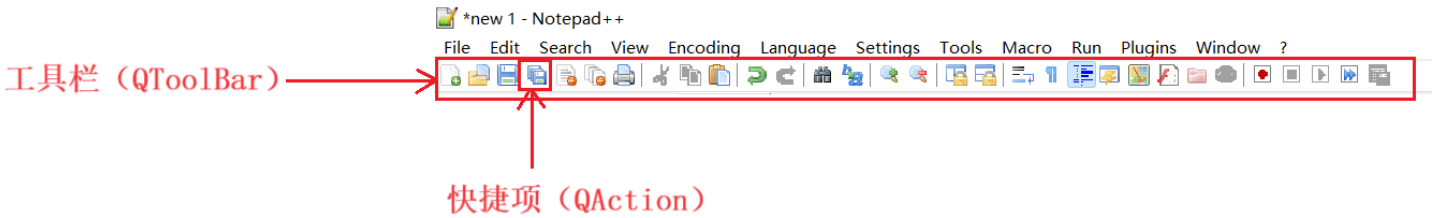
执行程序, 可以看到此时就可以通过程序来保存/加载文件了. 并且对文件进行编辑.





2. 工具栏

工具栏是应用程序中集成各种功能实现快捷键使用的一个区域。可以有多个，也可以没有，它并不是应用程序中必须存在的组件。它是一个可移动的组件，它的元素可以是各种窗口组件，它的元素通常以图标按钮的方式存在。如下图为工具栏的示意图：



2.1 创建工具栏

调用 **QMainWindow类** 的 **addToolBar()** 函数来创建工具栏，每增加一个工具栏都需要调用一次该函数。

如添加两个工具栏：

```
QToolBar *toolBar1 = new QToolBar(this);
QToolBar *toolBar2 = new QToolBar(this);
this->addToolBar(toolBar1);
this->addToolBar(toolBar2);
```

2.2 设置停靠位置

工具栏停靠位置的设置有两种方式。一种是在创建工具栏的同时指定停靠的位置，另一种是通过**QToolBar类** 提供的 **setAllowedAreas()函数** 来设置。

方式一：创建工具栏的同时指定其停靠的位置。

在创建工具栏的同时，也可以设置工具栏的位置，其默认位置是在窗口的最上面；如上述代码，默认在最上面显示。工具栏允许停靠的区域由 **QToolBar类** 提供的 **allowAreas()函数** 决定，其中可以设置的位置包括：

- **Qt::LeftToolBarArea　　停靠在左侧**
- **Qt::RightToolBarArea　　停靠在右侧**
- **Qt::TopToolBarArea　　停靠在顶部**
- **Qt::BottomToolBarArea　停靠在底部**
- **Qt::AllToolBarAreas　　以上四个位置都可停靠**

示例：

```
QToolBar *toolBar1 = new QToolBar(this);
QToolBar *toolBar2 = new QToolBar(this);

//创建工具栏的同时，指定工具栏在左侧显示
this->addToolBar(Qt::LeftToolBarArea,toolBar1);

//创建工具栏的同时，指定工具栏在右侧显示
this->addToolBar(Qt::RightToolBarArea,toolBar2);
```

方式二：使用 QToolBar类 提供的 setAllowedAreas()函数 设置停靠位置。如下示例：

```
QToolBar *toolBar1 = new QToolBar(this);
QToolBar *toolBar2 = new QToolBar(this);

this->addToolBar(toolBar1);
this->addToolBar(toolBar2);

//只允许在左侧停靠
toolBar1->setAllowedAreas(Qt::LeftToolBarArea);

//只允许在右侧停靠
toolBar2->setAllowedAreas(Qt::RightToolBarArea);
```

说明：

在创建工具栏的同时指定其停靠的位置，指的是程序运行时工具栏默认所在的位置；而使用setAllowedAreas()函数设置停靠位置，指的是工具栏允许其所能停靠的位置。

2.3 设置浮动属性

工具栏的浮动属性可以通过 **QToolBar类** 提供的 **setFloatable()函数** 来设置。**setFloatable()函数原型为：**

```
void setFloatable (bool floatable)
```

参数：

true：浮动

false：不浮动

示例：

```
QToolBar *toolBar1 = new QToolBar(this);
QToolBar *toolBar2 = new QToolBar(this);

this->addToolBar(Qt::LeftToolBarArea,toolBar1);
this->addToolBar(Qt::RightToolBarArea,toolBar2);

toolBar1->setFloatable(true);  //允许工具栏浮动
toolBar2->setFloatable(false); //不允许工具栏浮动
```

2.4 设置移动属性

设置工具栏的移动属性可以通过 **QToolBar类** 提供的 **setMovable()函数** 来设置。**setMovable()函数原型为：**

```
void setMovable(bool movable)
```

参数：

true：移动

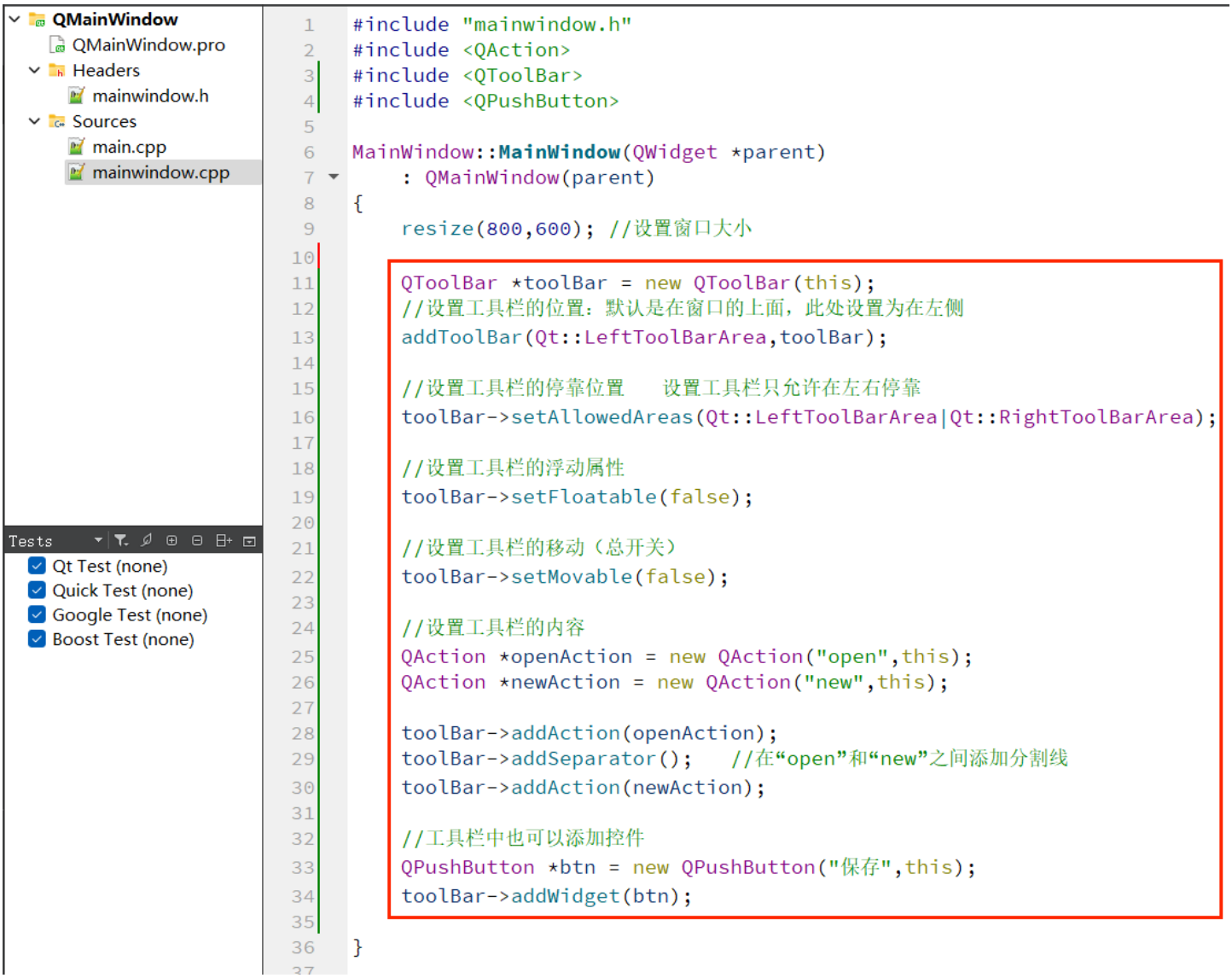
false：不移动

> 📌 **说明：若设置工具栏为不移动状态，则设置其停靠位置的操作就不会生效，所以设置工具栏的移动属性类似于总开关的效果。**

示例：

```
QToolBar *toolBar1 = new QToolBar(this);
QToolBar *toolBar2 = new QToolBar(this);

this->addToolBar(Qt::LeftToolBarArea,toolBar1);
this->addToolBar(Qt::RightToolBarArea,toolBar2);

toolBar1->setMovable(true);  //允许移动
toolBar2->setMovable(false); //不允许移动
```

2.5 综合示例



3. 状态栏

状态栏是应用程序中输出简要信息的区域。一般位于主窗口的最底部，一个窗口中最多只能有一个状态栏。在 Qt 中，状态栏是通过 **QStatusBar类** 来实现的。 在状态栏中可以显示的消息类型有：

- 实时消息：如当前程序状态
- 永久消息：如程序版本号，机构名称
- 进度消息：如进度条提示，百分百提示

3.1 状态栏的创建

状态栏的创建是通过 **QMainWindow 类** 提供的 **statusBar()** 函数来创建；示例如下：

```
//创建状态栏
QStatusBar *stbar = statusBar();

//将状态栏置于窗口中
setStatusBar(stbar);
```

3.2 在状态栏中显示实时消息

在状态栏中显示实时消息是通过 **showMessage() 函数**来实现，示例如下：

```
//状态栏中显示大约2秒的"Hello_Qt"
stbar->showMessage("Hello_Qt",2000);
```

3.3 在状态栏中显示永久消息

在状态栏中可以显示永久消息，此处的永久消息是通过 **标签** 来显示的；示例如下：

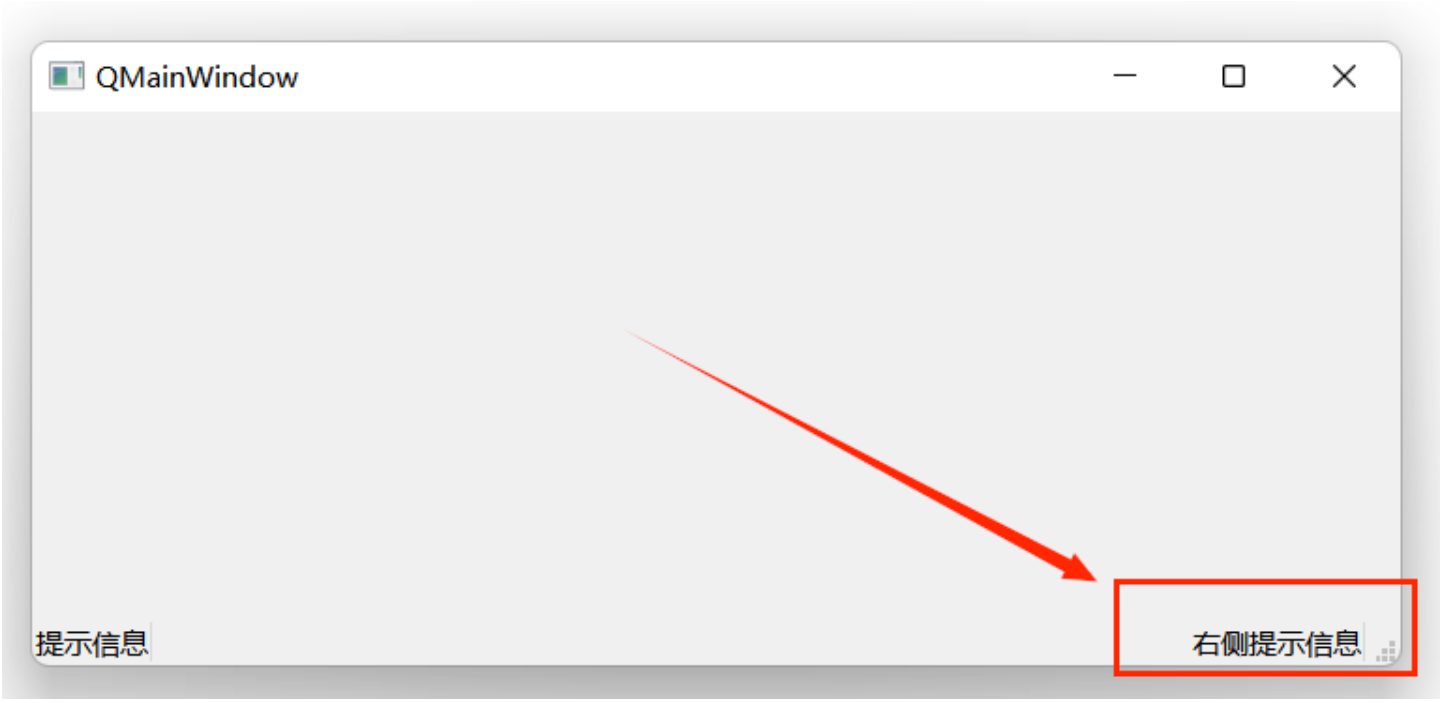
```
//创建状态栏
QStatusBar *stbar = statusBar();

//将状态栏置于窗口中
setStatusBar(stbar);

//创建标签
QLabel *label = new QLabel("提示信息",this);

//将标签放入状态栏中
stbar->addWidget(label);
```

显示效果如下：

调整显示消息的位置

```
//将创建的标签放入到状态栏的右侧
QLabel *label2 = new QLabel("右侧提示信息",this);
stbar->addPermanentWidget(label2);
```

显示效果如下：



4. 浮动窗口

在 Qt 中，浮动窗口也称之为铆接部件。浮动窗口是通过 **QDockWidget类** 来实现浮动的功能。浮动窗口一般是位于核心部件的周围，**可以有多个。**

4.1 浮动窗口的创建

浮动窗口的创建是通过 **QDockWidget类** 提供的构造方法 **QDockWidget()函数** 动态创建的；示例如下：

```
//浮动窗口
QDockWidget *dockwidget = new QDockWidget("浮动窗口",this);

//将浮动窗口置于当前窗口中
addDockWidget(Qt::BottomDockWidgetArea,dockwidget);
```

4.2 设置停靠的位置

浮动窗口是位于中心部件的周围。可以通过 **QDockWidget类** 中提供 **setAllowedAreas()** 函数设置其允许停靠的位置。其中可以设置允许停靠的位置有：

- Qt::LeftDockWidgetArea　　停靠在左侧
- Qt::RightDockWidgetArea　　停靠在右侧
- Qt::TopDockWidgetArea　　停靠在顶部
- Qt::BottomDockWidgetArea　停靠在底部
- Qt::AllDockWidgetAreas　　以上四个位置都可停靠

示例如下：**设置浮动窗口只允许上下停靠**

```
//设置浮动窗口的停靠区域，只允许上下停靠
dockwidget->setAllowedAreas(Qt::TopDockWidgetArea | Qt::BottomDockWidgetArea);
```

5. 对话框

5.1 对话框介绍

对话框是 GUI 程序中不可或缺的组成部分。一些不适合在主窗口实现的功能组件可以设置在对话框中。对话框通常是一个顶层窗口，出现在程序最上层，用于实现短期任务或者简洁的用户交互。Qt常用的内置对话框有：**QFiledialog（文件对话框）、QColorDialog（颜色对话框）、QFontDialog（字体对话框）、QInputDialog （输入对话框）和 QMessageBox（消息框）** 。

5.2 对话框的分类

对话框分为 **模态对话框** 和 **非模态对话框。**

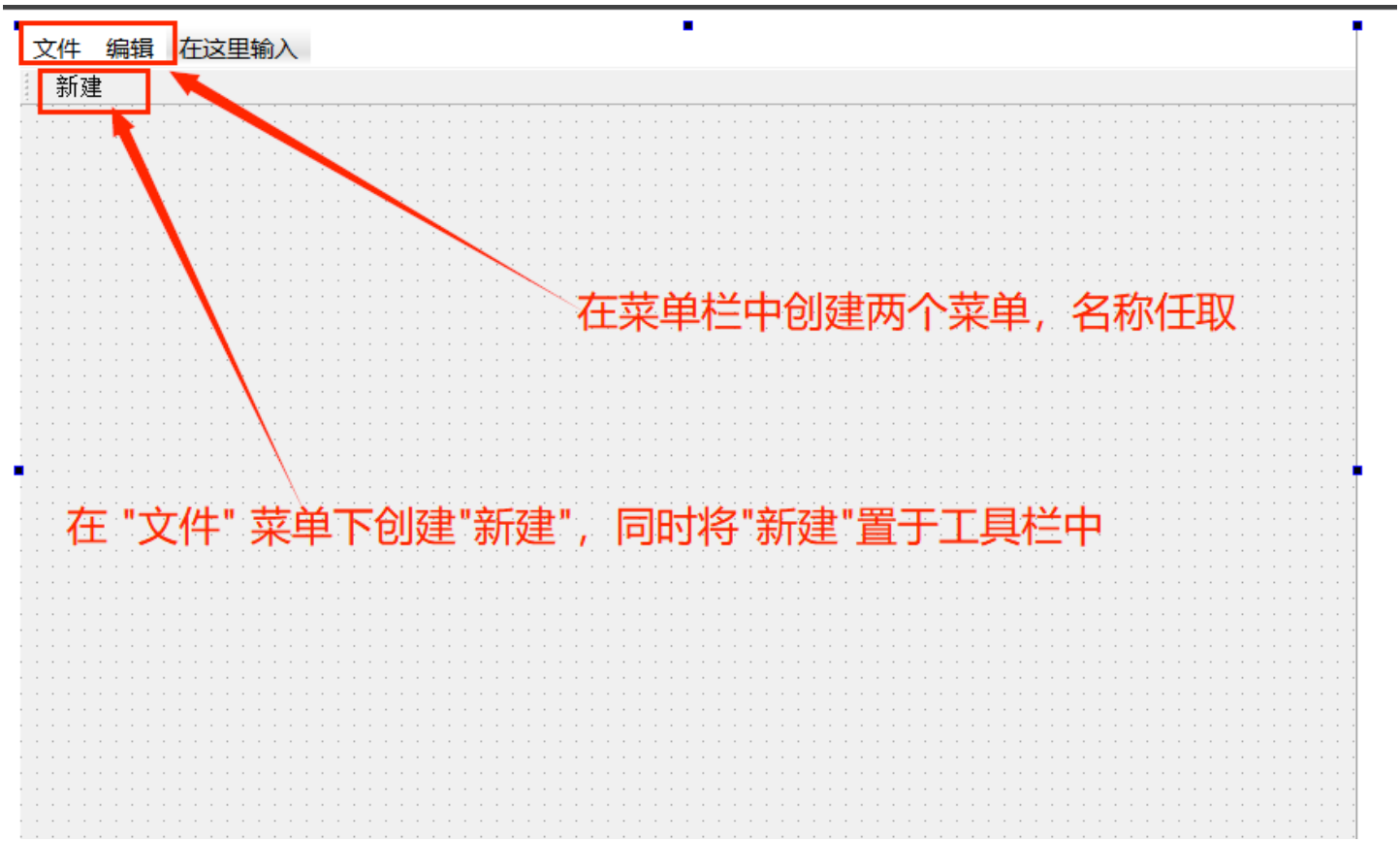
5.2.1 模态对话框

模态对话框指的是：**显示后无法与父窗口进行交互，**是一种阻塞式的对话框。使用 QDialog::exec() 函数 调用。

模态对话框适用于必须依赖用户选择的场合，比如消息显示，文件选择，打印设置等。

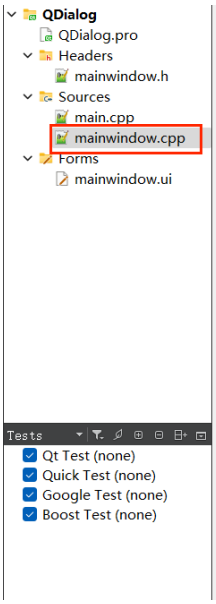
示例：

1、新建 Qt 项目，在 ui 文件中的菜单栏中设置两个菜单：**"文件"** 和 **"编辑"**，在 菜单 **"文件"** 下新建菜单项：**"创建" 并将菜单项 "新建" 置于工具栏中；** 如下图示：



2、在 mainwindow.cpp 文件中实现：当点击 **"新建"** 时，弹出一个模态对话框；

> 说明：在菜单项中，点击菜单项时就会触发 triggered() 信号。




```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDialog>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //当点击“新建”时，弹出一个模态对话框  在菜单项中，当点击之后就会触发triggered信号
    connect(ui->actionnew,&QAction::triggered,[=](){

        QDialog dlg(this);
        dlg.resize(200,100);
        dlg.exec();
    });
}
```

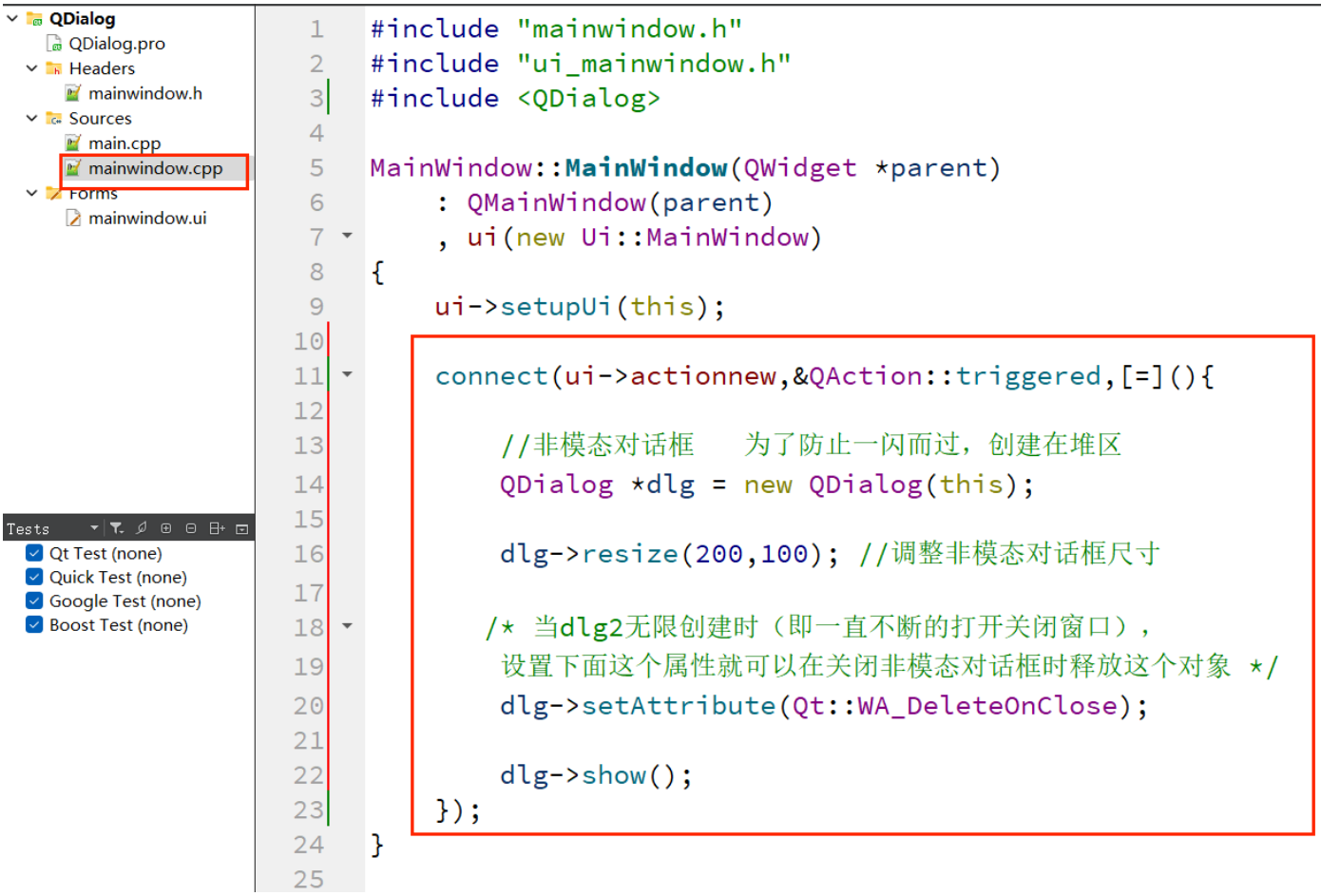
5.2.2 非模态对话框

非模态对话框显示后独立存在，**可以同时与父窗口进行交互**，是一种非阻塞式对话框，使用QDialog::show()函数调用。

> 非模态对话框一般在堆上创建，这是因为如果创建在栈上时，弹出的非模态对话框就会一闪而过。同时还需要设置 Qt:WA_DeleteOnClose 属性，目的是：当创建多个非模态对话框时（如打开了多个非模态窗口），为了避免内存泄漏要设置此属性。

非模态对话框适用于特殊功能设置的场合，比如查找操作，属性设置等。

示例：



5.2.3 混合属性对话框

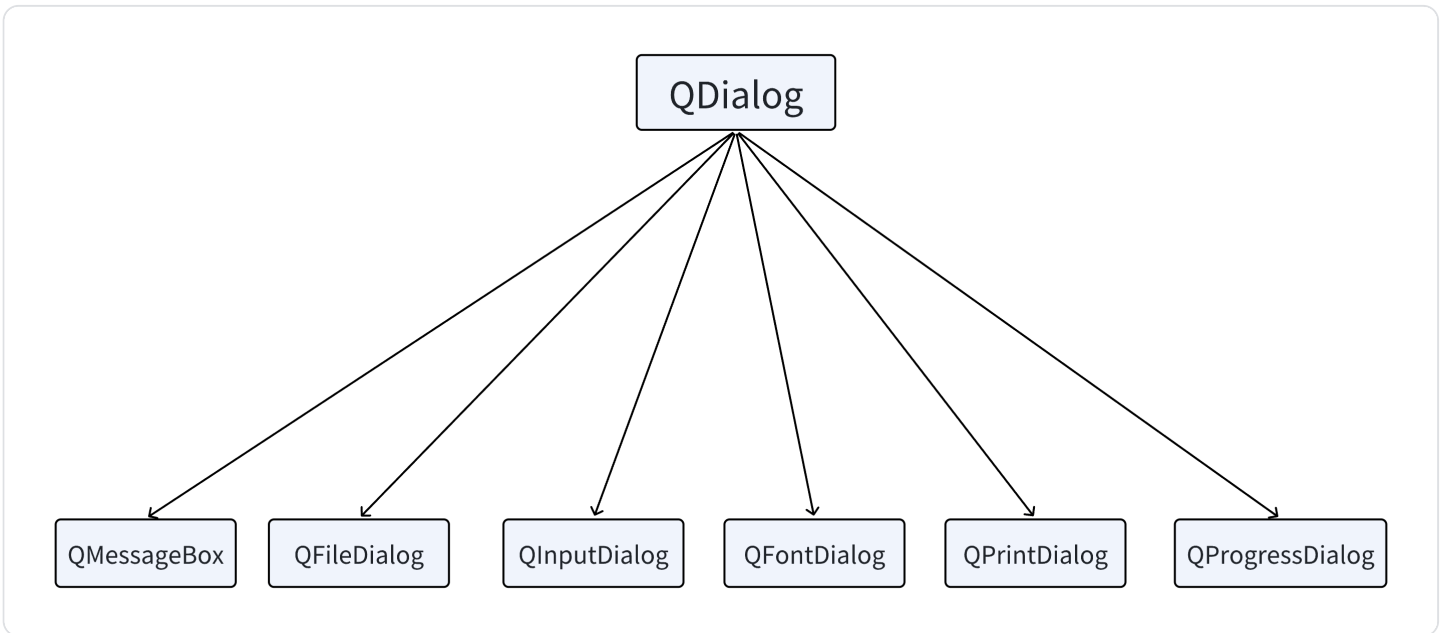
混合属性对话框同时具有模态对话框和非模态对话框的属性，对话框的生成和销毁具有非模态对话框属性，功能上具有模态对话框的属性。

使用 **QDialog::setModal() 函数** 可以创建混合特性的对话框。通常，创建对话框时需要指定对话框的父组件。

示例：

5.3 Qt 内置对话框

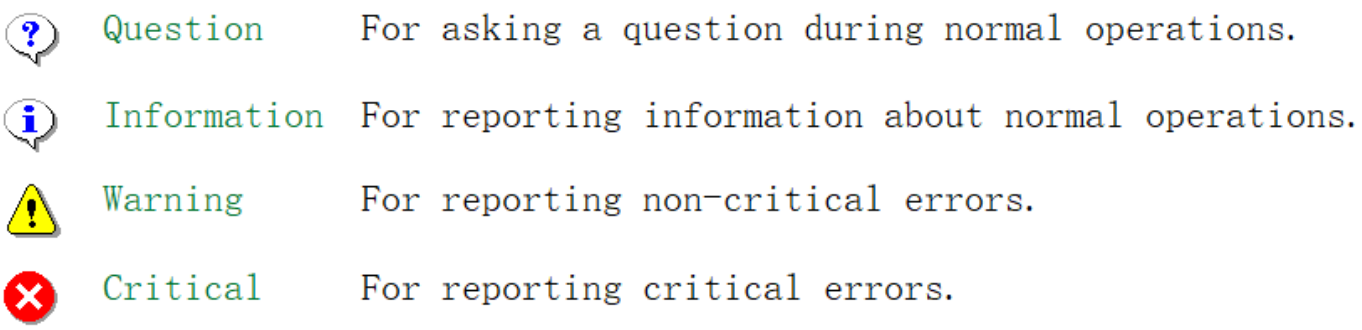
Qt 提供了多种可复用的对话框类型，即 **Qt 标准对话框**。**Qt** 标准对话框全部继承于 **QDialog类**。常用标准对话框如下：



5.3.1 消息对话框 QMessageBox

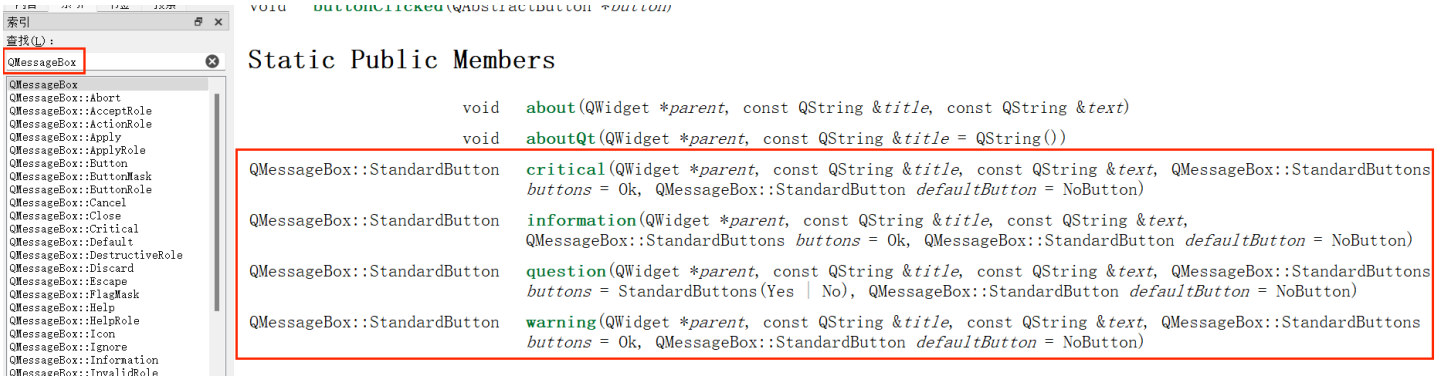
消息对话框是应用程序中最常用的界面元素。消息对话框主要用于为用户提示重要信息，强制用户进行选择操作。

QMessageBox类 中定义了静态成员函数，可以直接调用创建不同风格的消息对话框，其中包括：



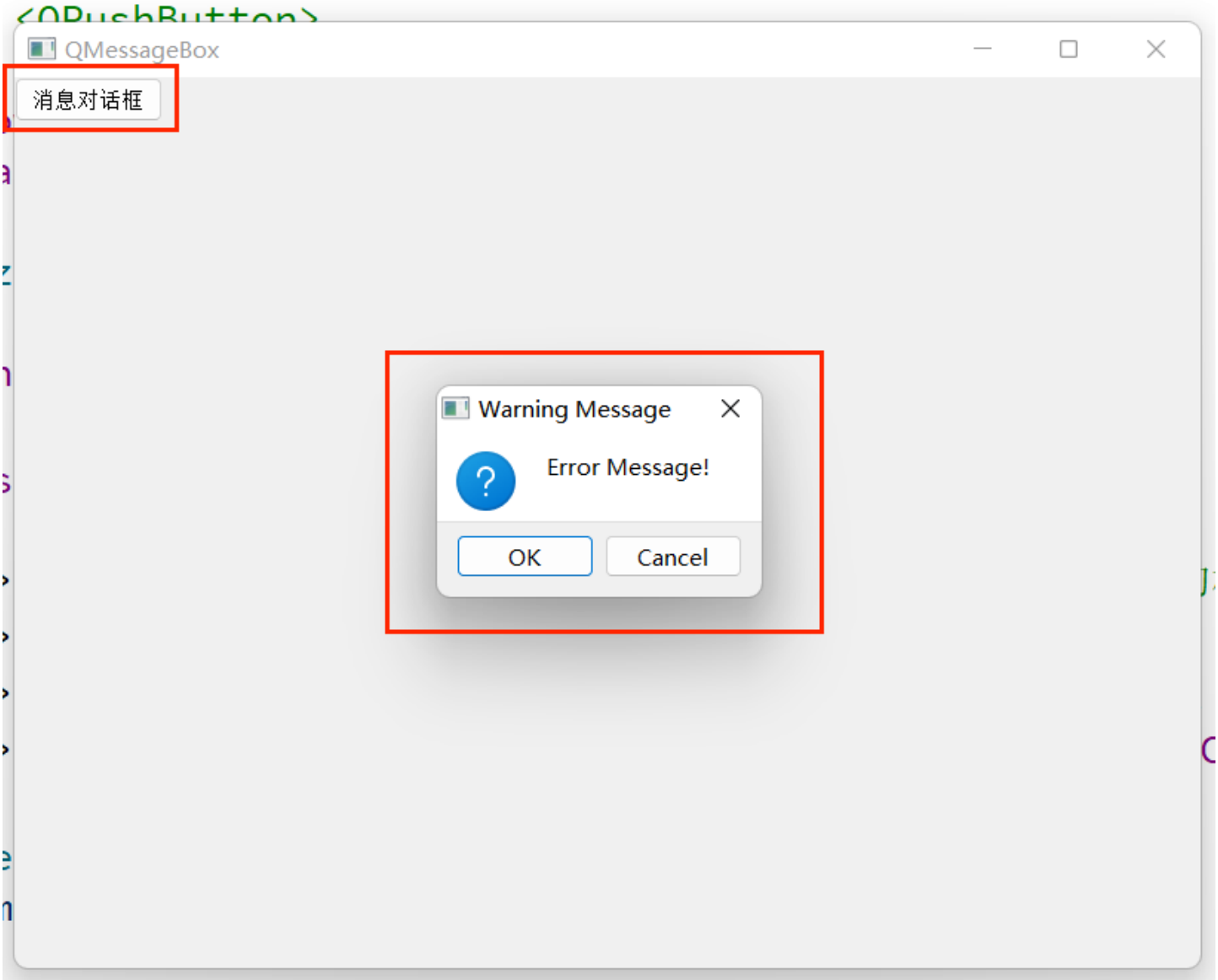
Question	**用于正常操作过程中的提问**
Information	**用于报告正常运行信息**
Warning	**用于报告非关键错误**
Critical	**用于报告严重错误**

其对应的函数原型如下：



示例1：问题提示消息对话框

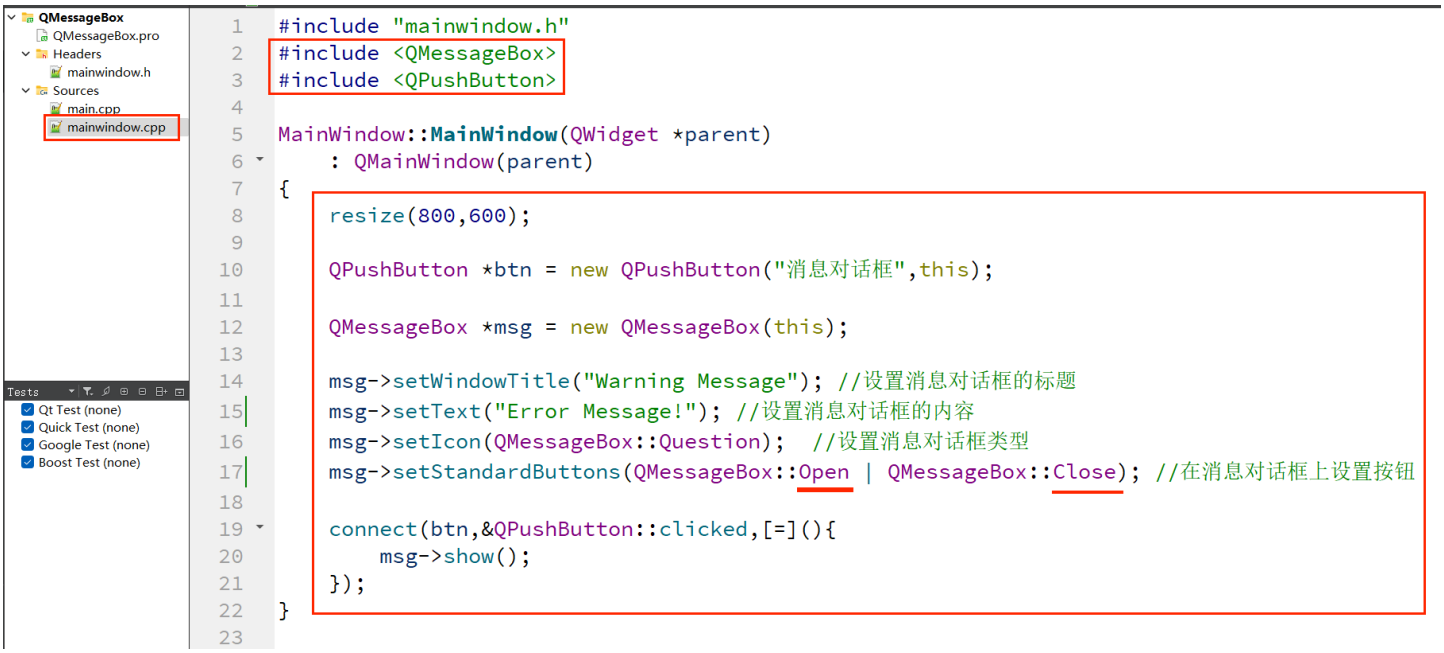
实现效果如下：



其中可以设置的按钮的类型如下：

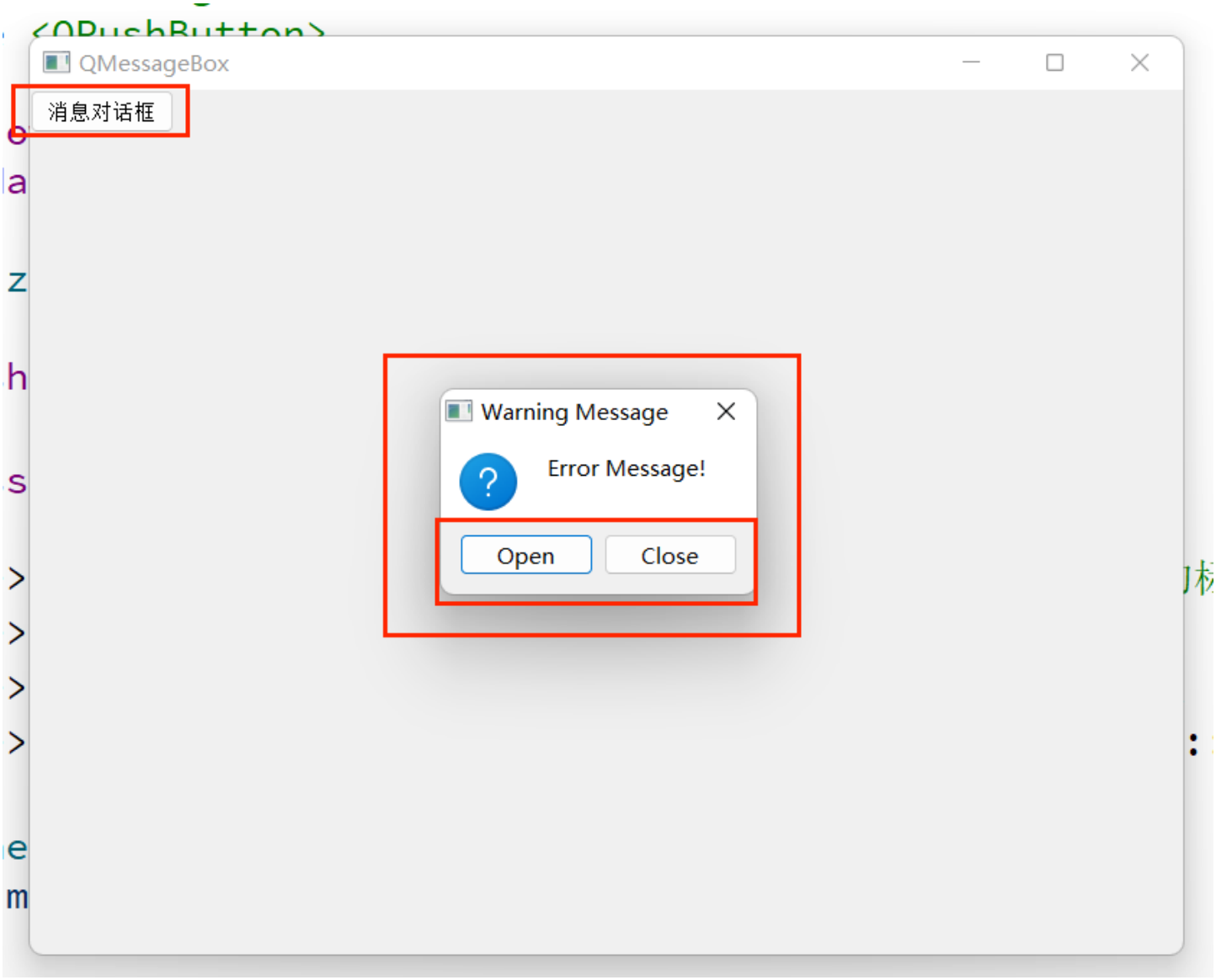
Constant	Value	Description
QMessageBox::Ok	0x00000400	An "OK" button defined with the AcceptRole.
QMessageBox::Open	0x00002000	An "Open" button defined with the AcceptRole.
QMessageBox::Save	0x00000800	A "Save" button defined with the AcceptRole.
QMessageBox::Cancel	0x00400000	A "Cancel" button defined with the RejectRole.
QMessageBox::Close	0x00200000	A "Close" button defined with the RejectRole.
QMessageBox::Discard	0x00800000	A "Discard" or "Don't Save" button, depending on the
QMessageBox::Apply	0x02000000	An "Apply" button defined with the ApplyRole.
QMessageBox::Reset	0x04000000	A "Reset" button defined with the ResetRole.
QMessageBox::RestoreDefaults	0x08000000	A "Restore Defaults" button defined with the ResetRo
QMessageBox::Help	0x01000000	A "Help" button defined with the HelpRole.
QMessageBox::SaveAll	0x00001000	A "Save All" button defined with the AcceptRole.
QMessageBox::Yes	0x00004000	A "Yes" button defined with the YesRole.
QMessageBox::YesToAll	0x00008000	A "Yes to All" button defined with the YesRole.
QMessageBox::No	0x00010000	A "No" button defined with the NoRole.
QMessageBox::NoToAll	0x00020000	A "No to All" button defined with the NoRole.
QMessageBox::Abort	0x00040000	An "Abort" button defined with the RejectRole.
QMessageBox::Retry	0x00080000	A "Retry" button defined with the AcceptRole.
QMessageBox::Ignore	0x00100000	An "Ignore" button defined with the AcceptRole.
QMessageBox::NoButton	0x00000000	An invalid button.

更改消息对话框中的按钮类型：

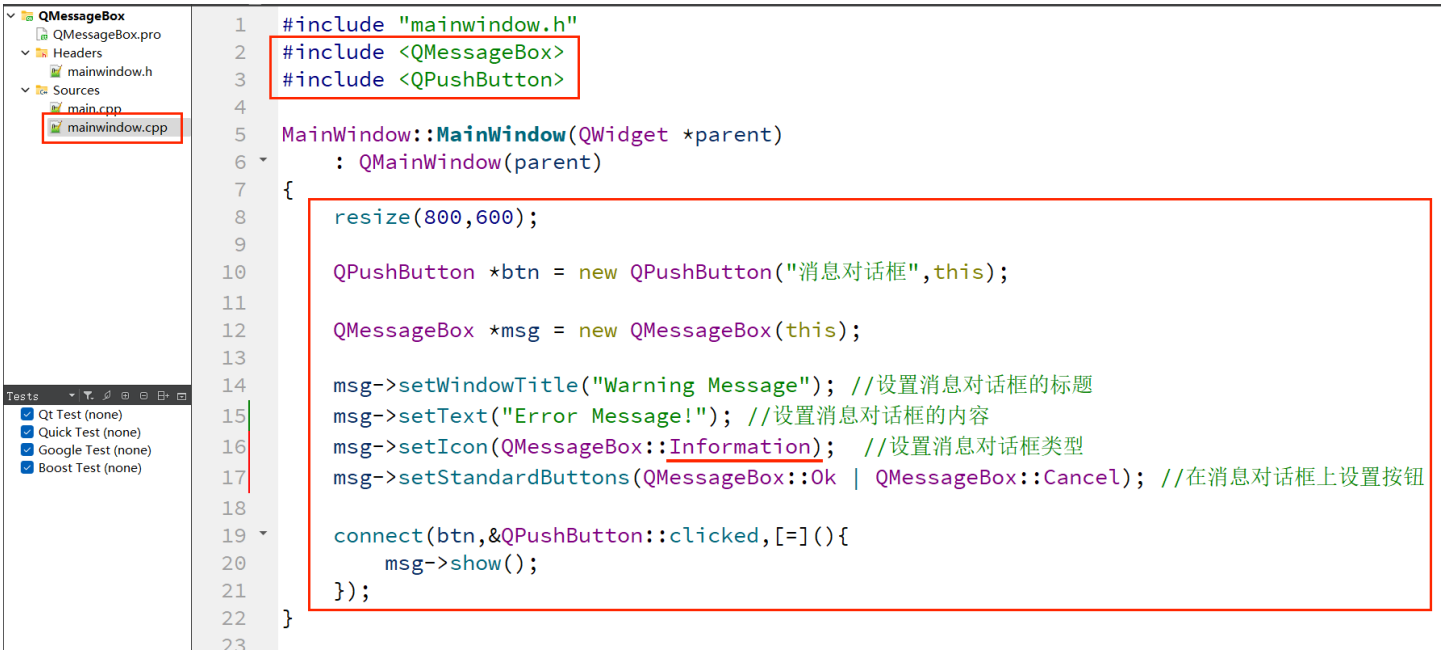



```cpp
#include "mainwindow.h"
#include <QMessageBox>
#include <QPushButton>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("消息对话框",this);

    QMessageBox *msg = new QMessageBox(this);

    msg->setWindowTitle("Warning Message"); //设置消息对话框的标题
    msg->setText("Error Message!"); //设置消息对话框的内容
    msg->setIcon(QMessageBox::Question);  //设置消息对话框类型
    msg->setStandardButtons(QMessageBox::Open | QMessageBox::Close); //在消息对话框上设置按钮

    connect(btn,&QPushButton::clicked,[=](){
        msg->show();
    });
}

```

实现效果如下：

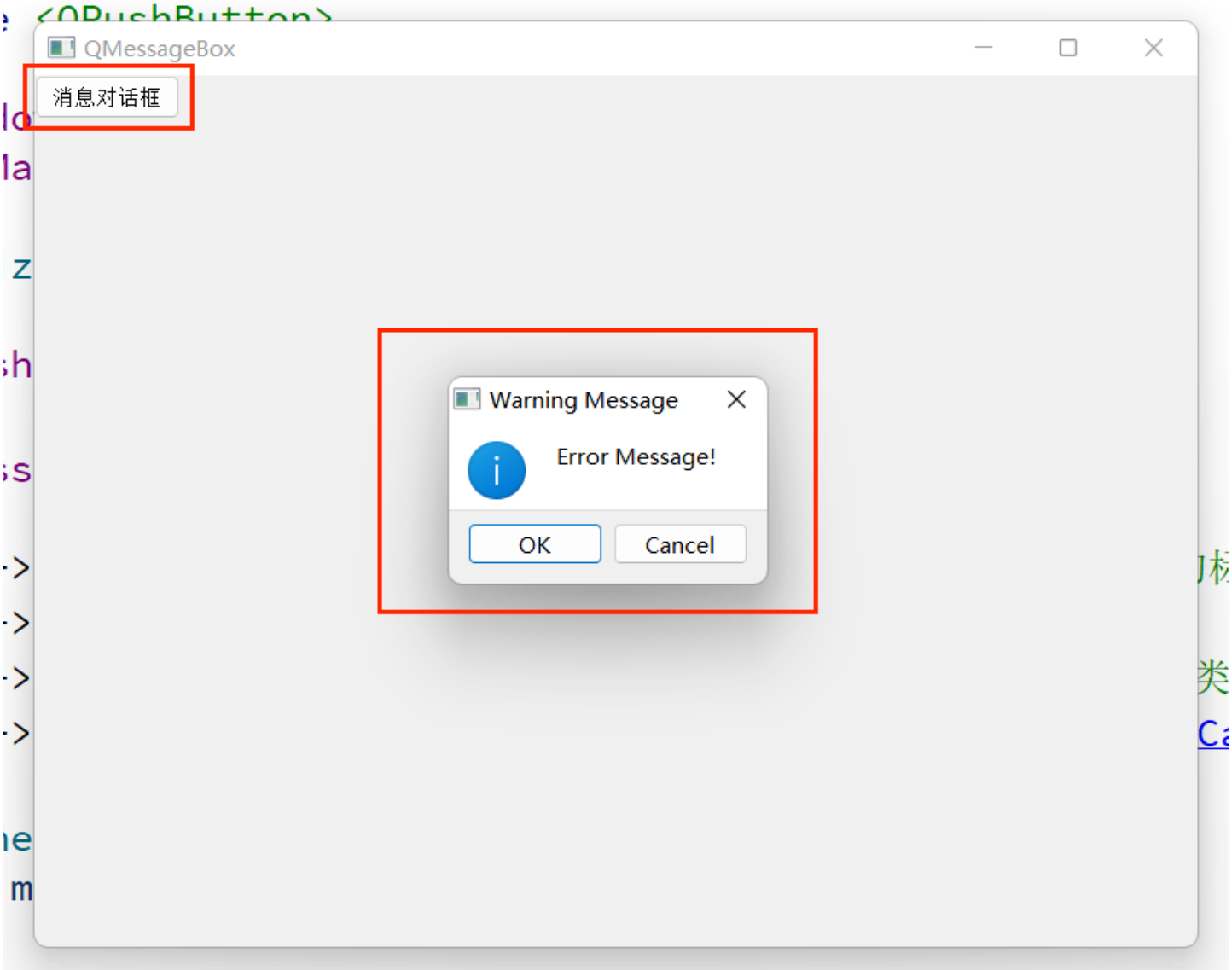


示例2：信息提示消息对话框

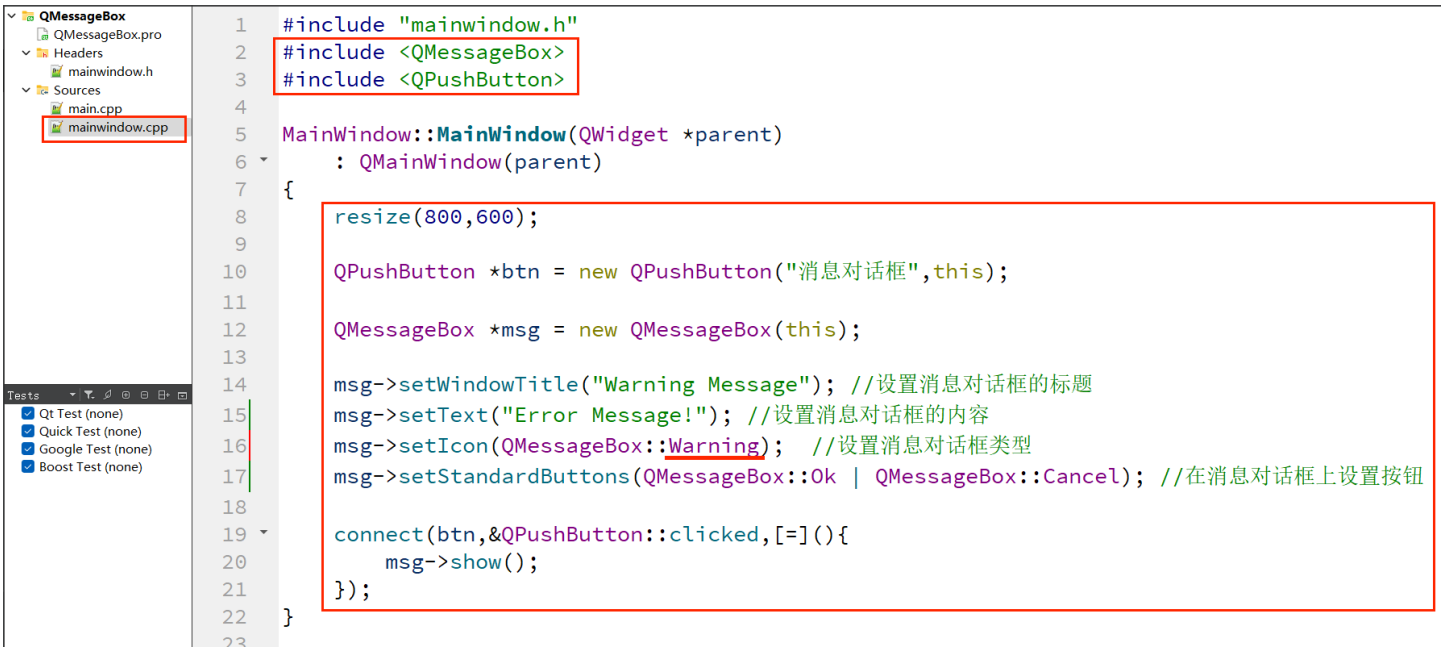



```cpp
#include "mainwindow.h"
#include <QMessageBox>
#include <QPushButton>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("消息对话框",this);

    QMessageBox *msg = new QMessageBox(this);

    msg->setWindowTitle("Warning Message"); //设置消息对话框的标题
    msg->setText("Error Message!"); //设置消息对话框的内容
    msg->setIcon(QMessageBox::Information);  //设置消息对话框类型
    msg->setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel); //在消息对话框上设置按钮

    connect(btn,&QPushButton::clicked,[=](){
        msg->show();
    });
}
```

实现效果如下：

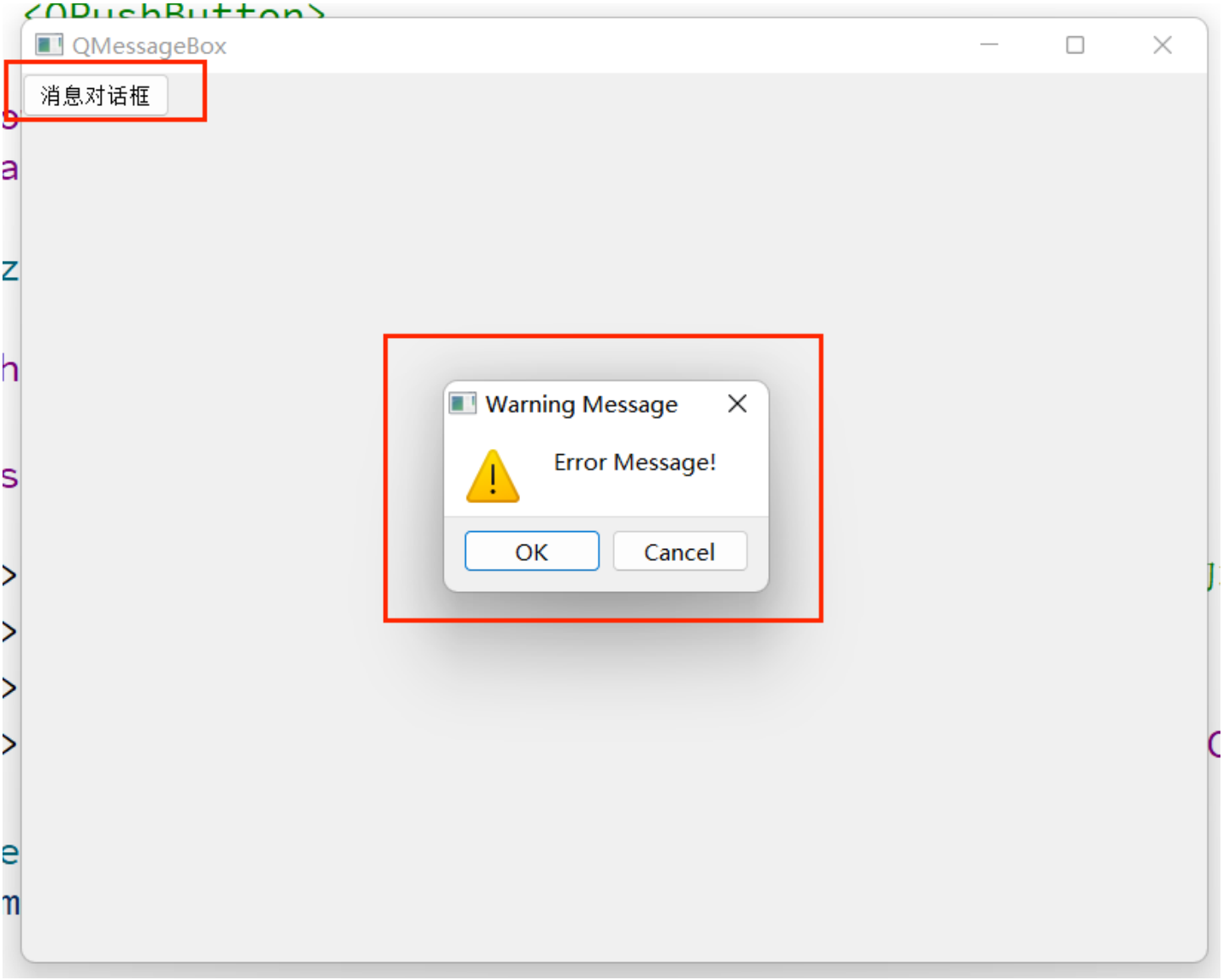


示例3：警告信息消息对话框

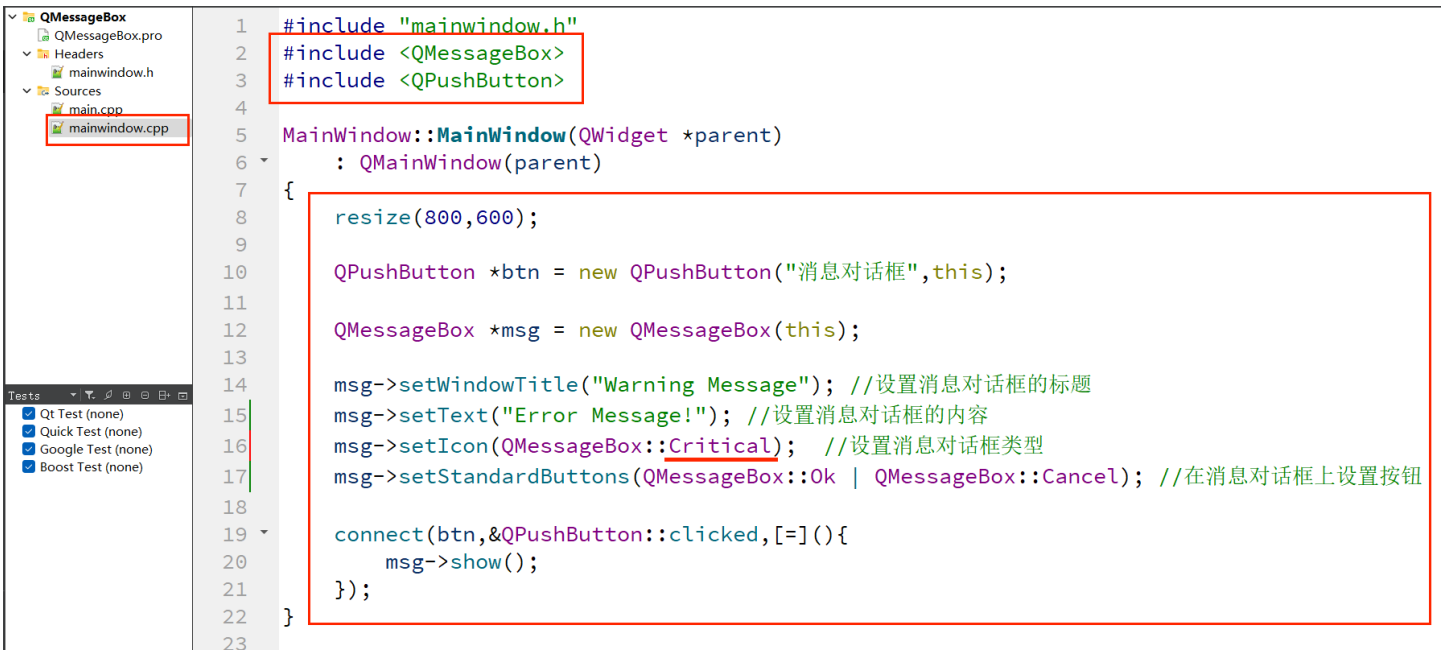



```cpp
#include "mainwindow.h"
#include <QMessageBox>
#include <QPushButton>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("消息对话框",this);

    QMessageBox *msg = new QMessageBox(this);

    msg->setWindowTitle("Warning Message"); //设置消息对话框的标题
    msg->setText("Error Message!"); //设置消息对话框的内容
    msg->setIcon(QMessageBox::Warning);  //设置消息对话框类型
    msg->setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel); //在消息对话框上设置按钮

    connect(btn,&QPushButton::clicked,[=](){
        msg->show();
    });
}
```

实现效果如下：

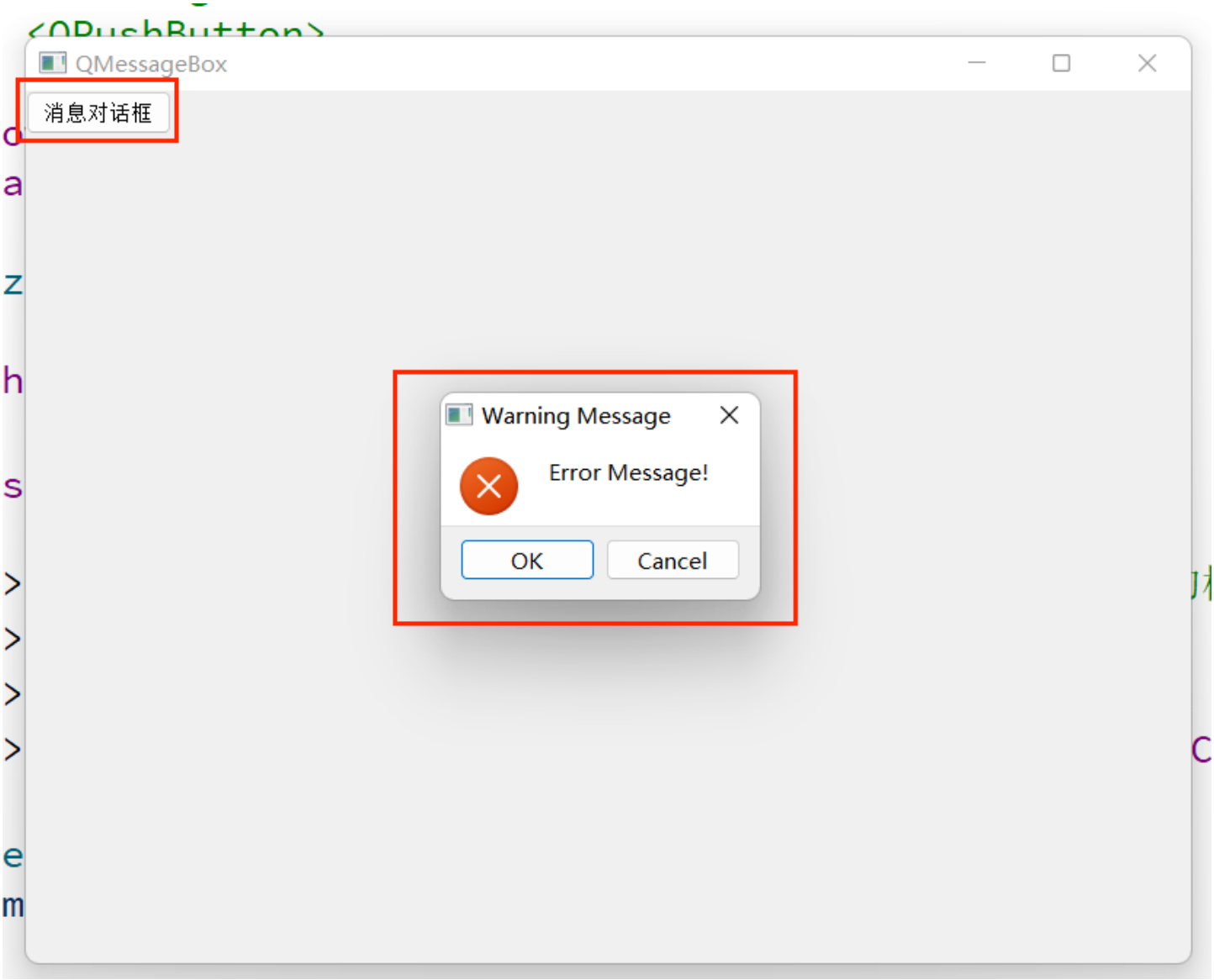


示例4：错误提示消息对话框



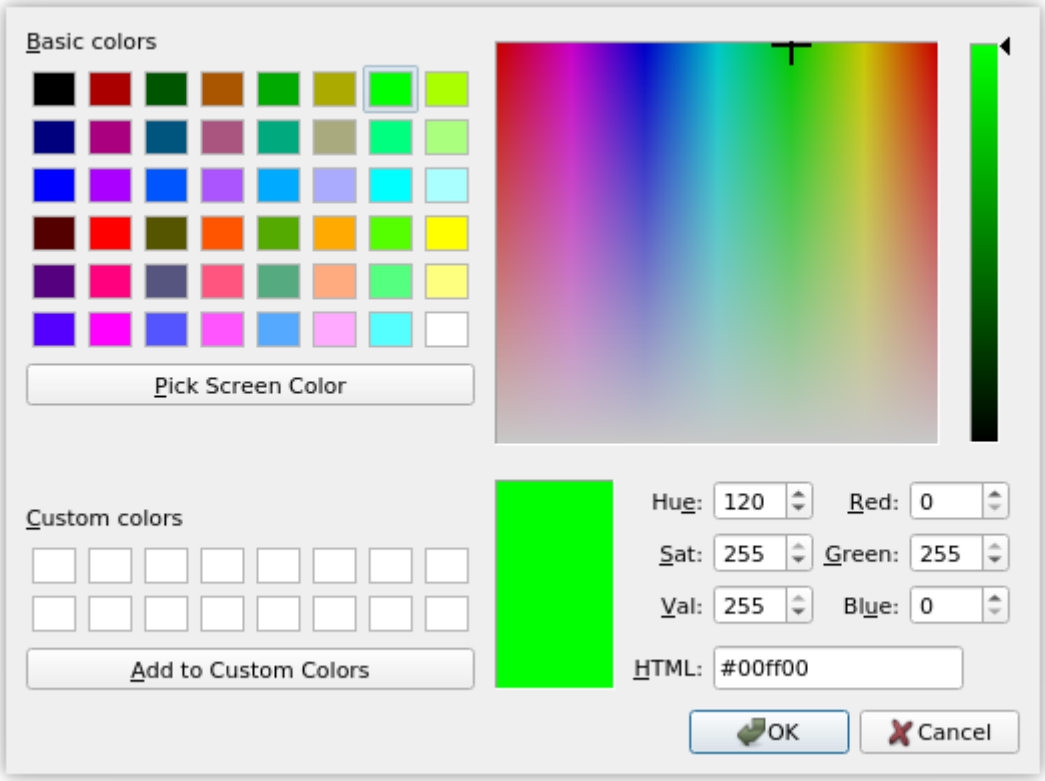

```cpp
#include "mainwindow.h"
#include <QMessageBox>
#include <QPushButton>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("消息对话框",this);

    QMessageBox *msg = new QMessageBox(this);

    msg->setWindowTitle("Warning Message"); //设置消息对话框的标题
    msg->setText("Error Message!"); //设置消息对话框的内容
    msg->setIcon(QMessageBox::Critical);  //设置消息对话框类型
    msg->setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel); //在消息对话框上设置按钮

    connect(btn,&QPushButton::clicked,[=](){
        msg->show();
    });
}
```

实现效果如下：



5.3.2 颜色对话框 QColorDialog

颜色对话框的功能是允许用户选择颜色。继承自 QDialog 类。颜色对话框如下图示：



常用方法介绍：

1、 QColorDialog (QWidget *parent = nullptr) //创建对象的同时设置父对象

2、 QColorDialog(const QColor &initial, QWidget *parent = nullptr) //创建对象的同时通过QColor对象设置默认颜色和父对象

3、 void setCurrentColor(const QColor &color) //设置当前颜色对话框

4、 QColor currentColor() const //获取当前颜色对话框

5、 QColor getColor(const QColor &initial = Qt::white, QWidget *parent = nullptr, const QString &title = QString(), QColorDialog::ColorDialogOptions options = ColorDialogOptions()) //打开颜色选择对话框，并返回一个QColor对象

参数说明：

initial：设置默认颜色

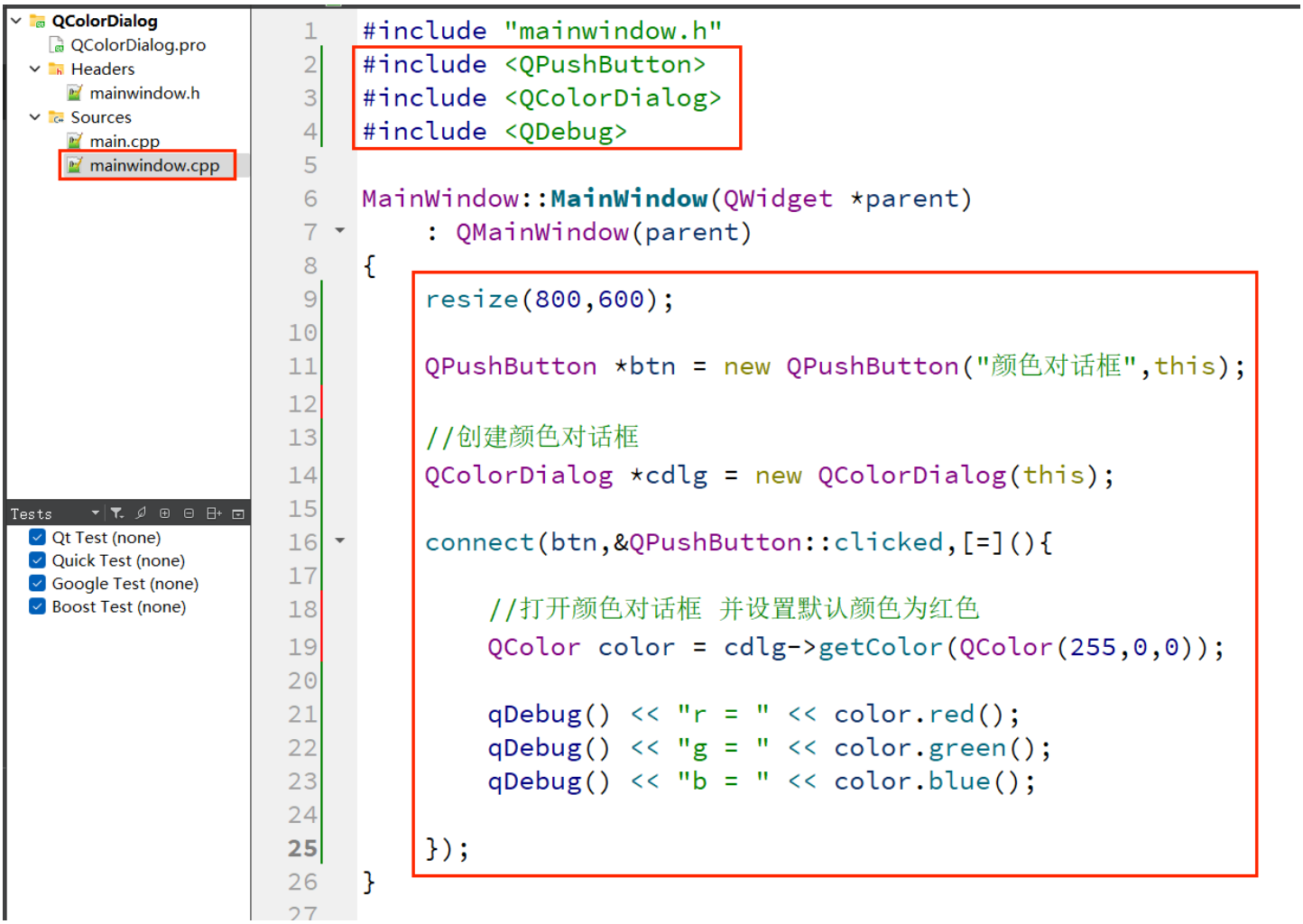
parent：设置父对象

title：设置对话框标题

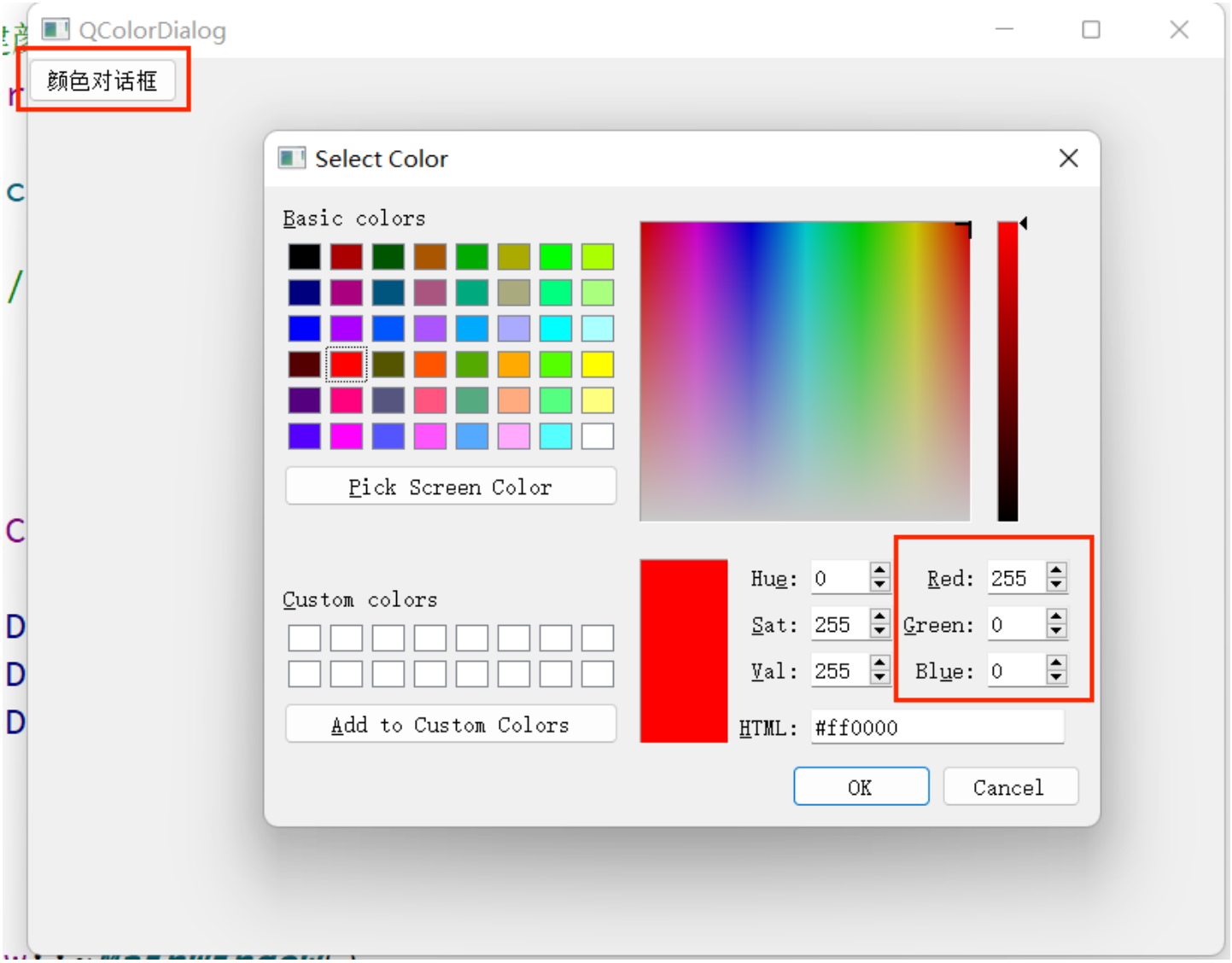
options：设置选项

6、 **void open(QObject *receiver, const char *member) //打开颜色对话框**

示例1：


QColorDialog
QColorDialog.pro
Headers
mainwindow.h
Sources
main.cpp
mainwindow.cpp
Tests
Qt Test (none)
Quick Test (none)
Google Test (none)
Boost Test (none)
#include "mainwindow.h"
#include <QPushButton>
#include <QColorDialog>
#include <QDebug>
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
{
resize(800,600);
QPushButton *btn = new QPushButton("颜色对话框",this);
//创建颜色对话框
QColorDialog *cdlg = new QColorDialog(this);
connect(btn,&QPushButton::clicked,[=](){
//打开颜色对话框 并设置默认颜色为红色
QColor color = cdlg->getColor(QColor(255,0,0));
qDebug() << "r = " << color.red();
qDebug() << "g = " << color.green();
qDebug() << "b = " << color.blue();
});
}

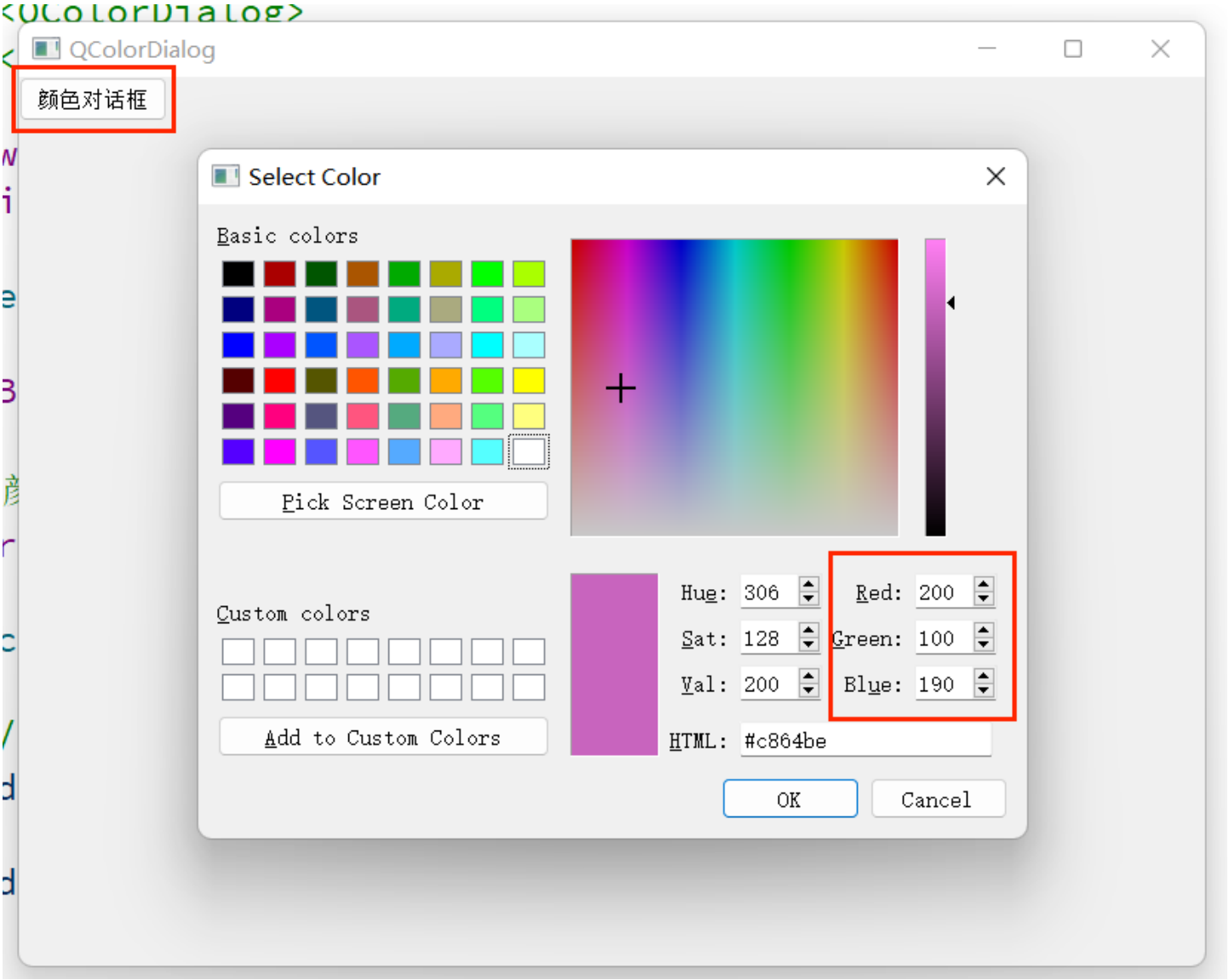
效果如下：


QColorDialog
颜色对话框
Select Color
Basic colors
Pick Screen Color
Custom colors
Add to Custom Colors
Hue: 0
Sat: 255
Val: 255
Red: 255
Green: 0
Blue: 0
HTML: #ff0000
OK
Cancel

示例2：


QColorDialog
QColorDialog.pro
Headers
mainwindow.h
Sources
main.cpp
mainwindow.cpp
Tests
Qt Test (none)
Quick Test (none)
Google Test (none)
Boost Test (none)
#include "mainwindow.h"
#include <QPushButton>
#include <QColorDialog>
#include <QDebug>
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
{
resize(800,600);
QPushButton *btn = new QPushButton("颜色对话框",this);
//创建颜色对话框
QColorDialog *cdlg = new QColorDialog(this);
connect(btn,&QPushButton::clicked,[=](){
//设置颜色对话框中的颜色
cdlg->setCurrentColor(QColor(200,100,190));
cdlg->open();
});
}

效果如下：



5.3.3 文件对话框 QFileDialog

文件对话框用于应用程序中需要打开一个外部文件或需要将当前内容存储到指定的外部文件。

常用方法介绍：

1、打开文件（一次只能打开一个文件）

QString getOpenFileName(QWidget *parent = nullptr, const QString &caption = QString(), const QString &dir = QString(), const QString &filter = QString(), QString *selectedFilter = nullptr, QFileDialog::Options options = Options())

2、打开多个文件（一次可以打开多个文件）

QStringList getOpenFileNames(QWidget *parent = nullptr, const QString &caption = QString(), const QString &dir = QString(), const QString &filter = QString(), QString *selectedFilter = nullptr, QFileDialog::Options options = Options())

3、 保存文件

QString getSaveFileName(QWidget *parent = nullptr, const QString &caption = QString(), const QString &dir = QString(), const QString &filter = QString(), QString *selectedFilter = nullptr, QFileDialog::Options options = Options())

参数说明：

参数1：parent　父亲

参数2：caption 对话框标题

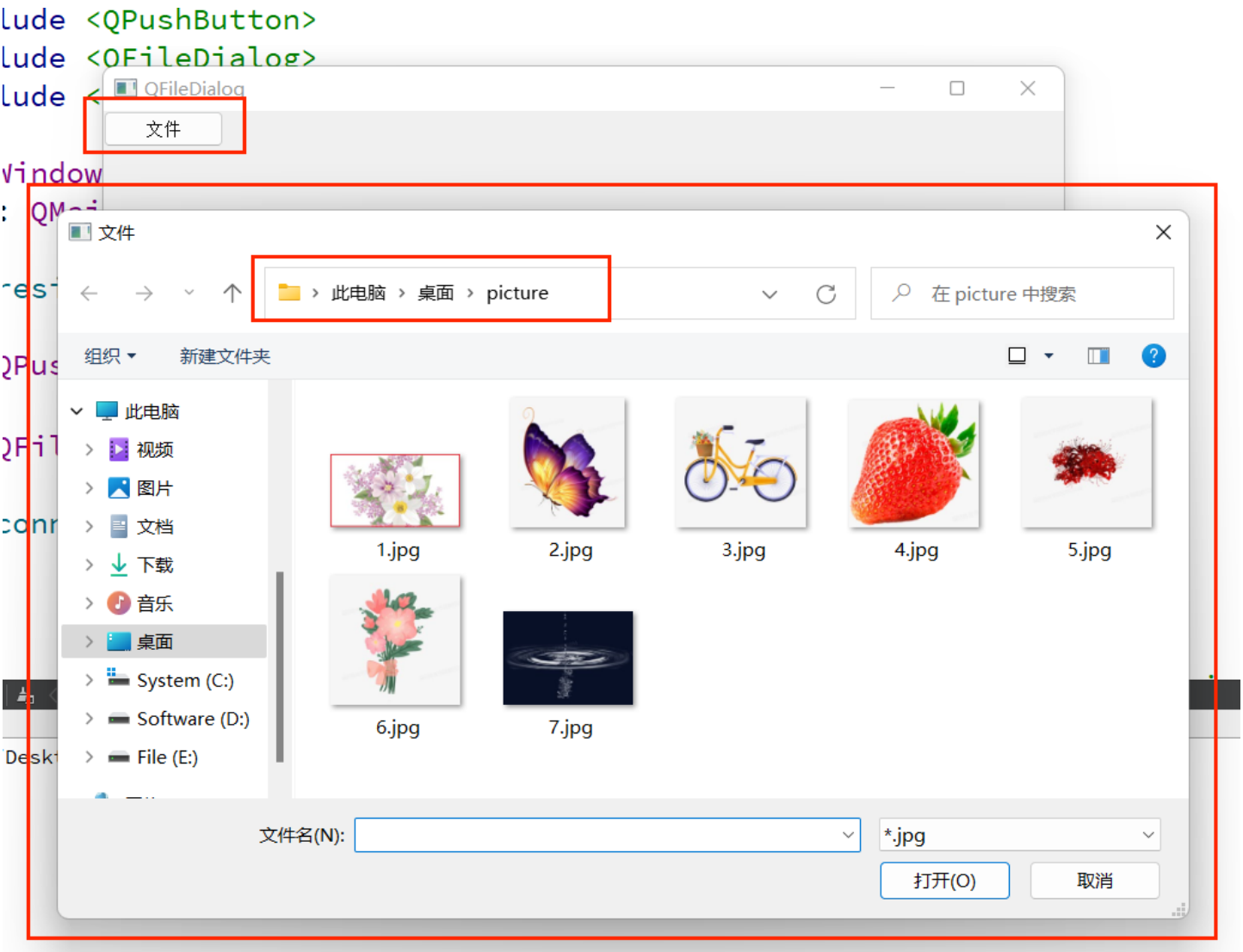
参数3：dir　默认打开的路径

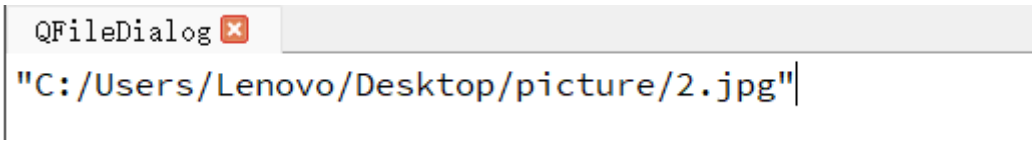
参数4：filter 文件过滤器

示例1：打开文件

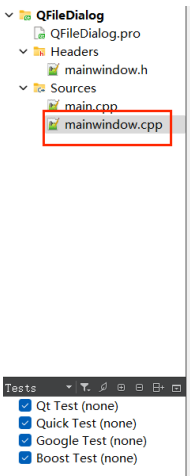

```
#include "mainwindow.h"
#include <QPushButton>
#include <QFileDialog>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("文件",this);

    QFileDialog *fdlg = new QFileDialog(this);

    connect(btn,&QPushButton::clicked,[=](){

        QString str = fdlg->getOpenFileName(this,     //父亲
                                            "文件",    //文件对话框标题
                                          "C:\\Users\\Lenovo\\Desktop\\picture",  //打开的路径
                                            "*.jpg"  //只保留 .jpg 格式文件
                                            );

        qDebug() << str;

    });
}
```

运行效果：





示例2：保存文件

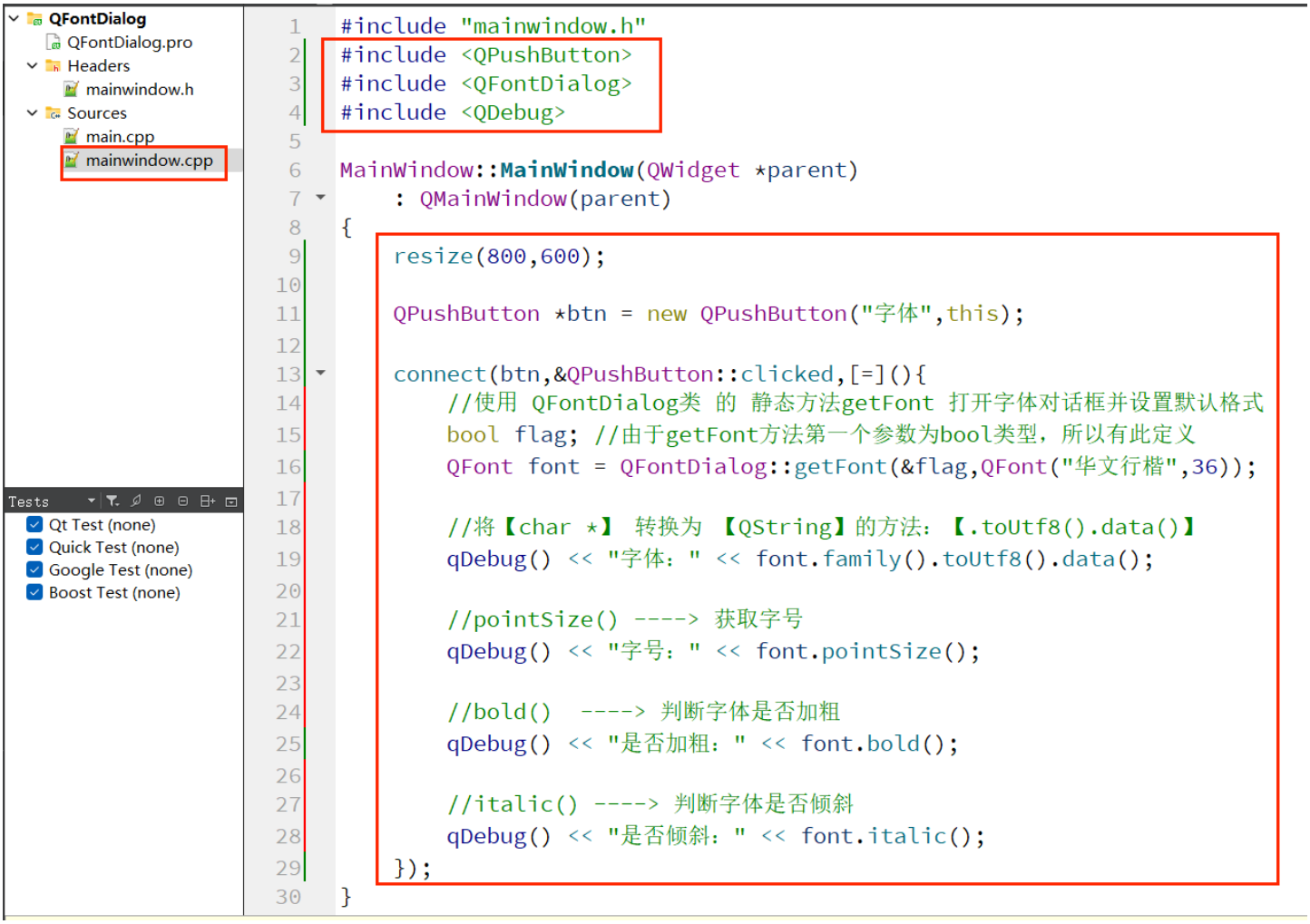



```cpp
#include "mainwindow.h"
#include <QPushButton>
#include <QFileDialog>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("文件",this);

    QFileDialog *fdlg = new QFileDialog(this);

    connect(btn,&QPushButton::clicked,[=](){

        QString str = fdlg->getSaveFileName(this,     //父亲
                                            "文件",    //文件对话框标题
                                            "C:\\Users\\Lenovo\\Desktop\\picture",  //默认保存的路径
                                            "*.jpg"  //只保留 .jpg 格式文件
                                            );

        qDebug() << str;

    });
}
```

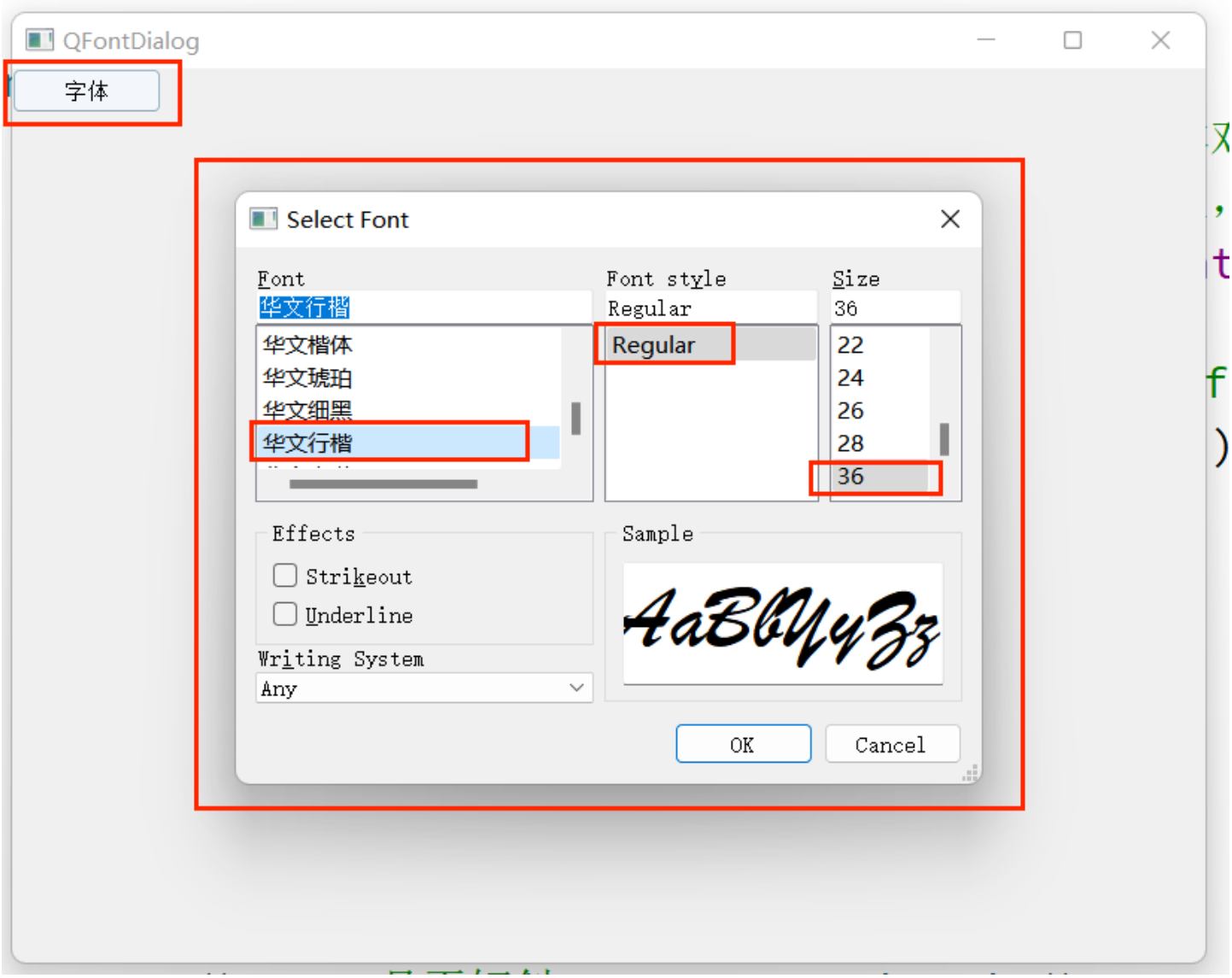
5.3.4 字体对话框 QFontDialog

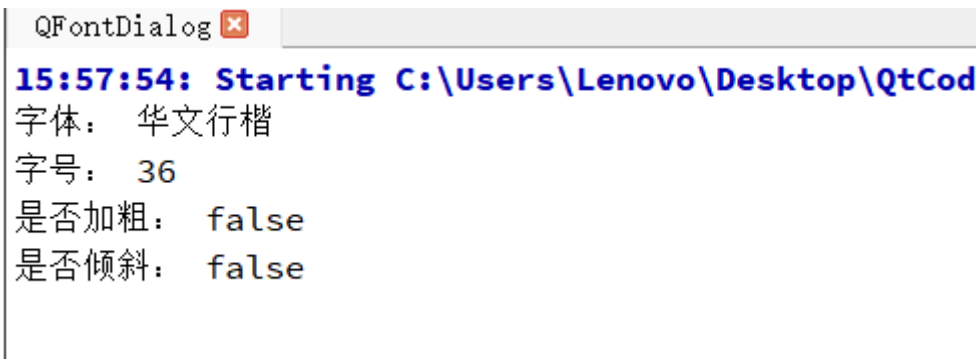
Qt 中提供了预定义的字体对话框类 QFontDialog，用于提供选择字体的对话框部件。

示例：



运行效果如下：





5.3.5 输入对话框 QInputDialog

Qt 中提供了预定义的输入对话框类：QInputDialog，用于进行临时数据输入的场合。

常用方法介绍：

1、双精度浮点型输入数据对话框

double getDouble (QWidget *parent, const QString &title, const QString &label, double value = 0, double min = -2147483647, double max = 2147483647, int decimals = 1, bool *ok = nullptr, Qt::WindowFlags flags = Qt::WindowFlags());

2、整型输入数据对话框

int getInt (QWidget *parent, const QString &title, const QString &label, int value = 0, int min = -2147483647, int max = 2147483647, int step = 1, bool *ok = nullptr, Qt::WindowFlags flags = Qt::WindowFlags());

3、选择条目型输入数据框

QString getItem (QWidget *parent, const QString &title, const QString &label, const QStringList &items, int current = 0, bool editable = true, bool *ok = nullptr, Qt::WindowFlags flags = Qt::WindowFlags(), Qt::InputMethodHints inputMethodHints = Qt::ImhNone) ;

参数说明：

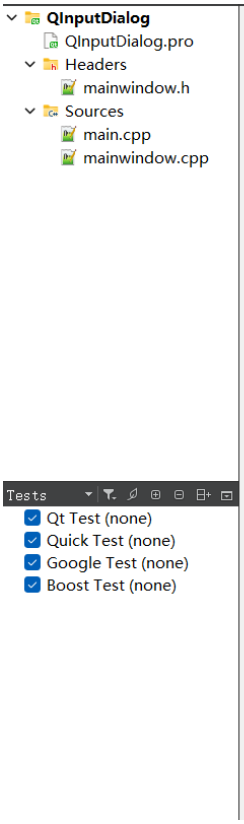
parent：父亲

title：对话框标题

label：对话框标签

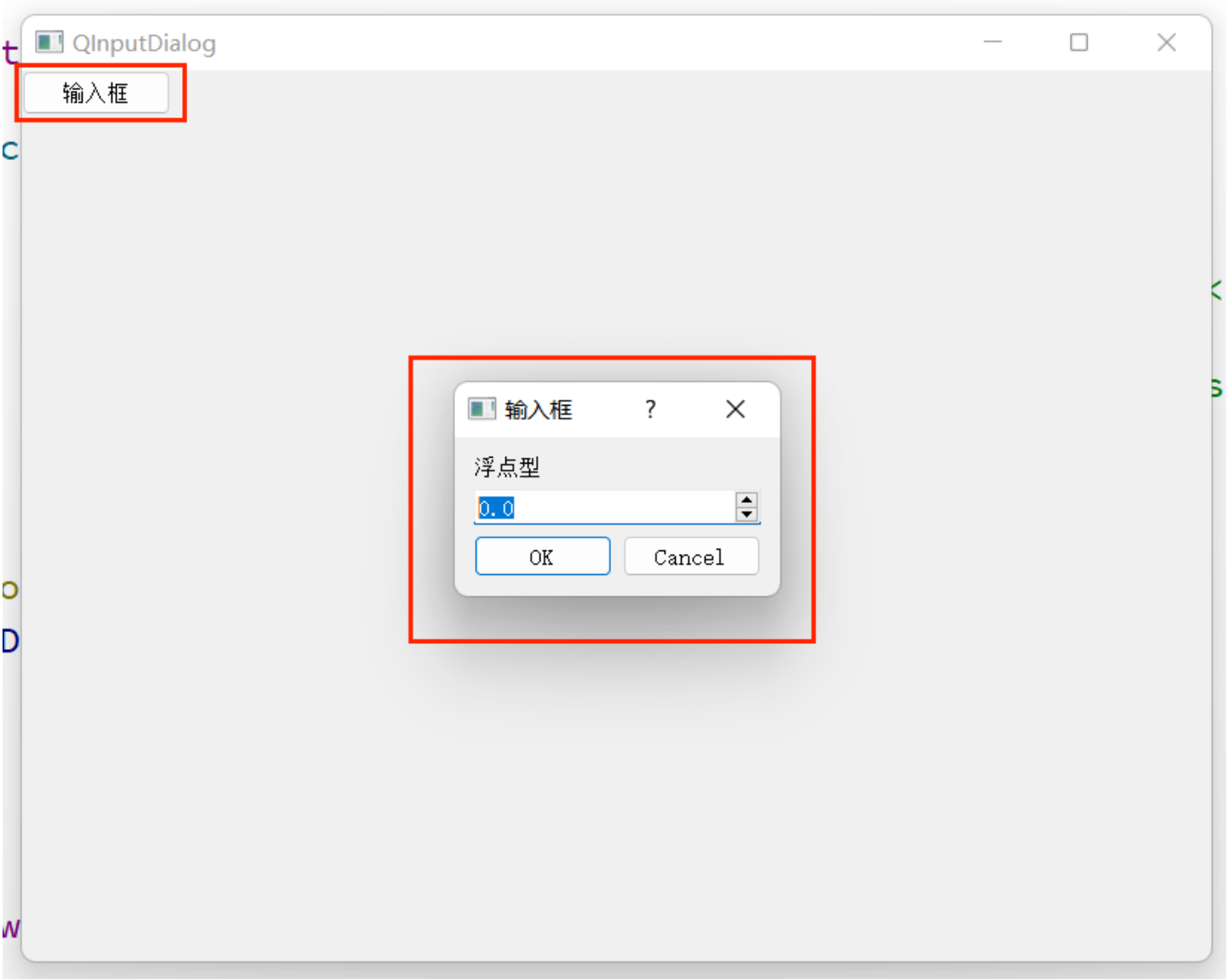
items：可供选择的条目

示例1：浮点型数据输入对话框

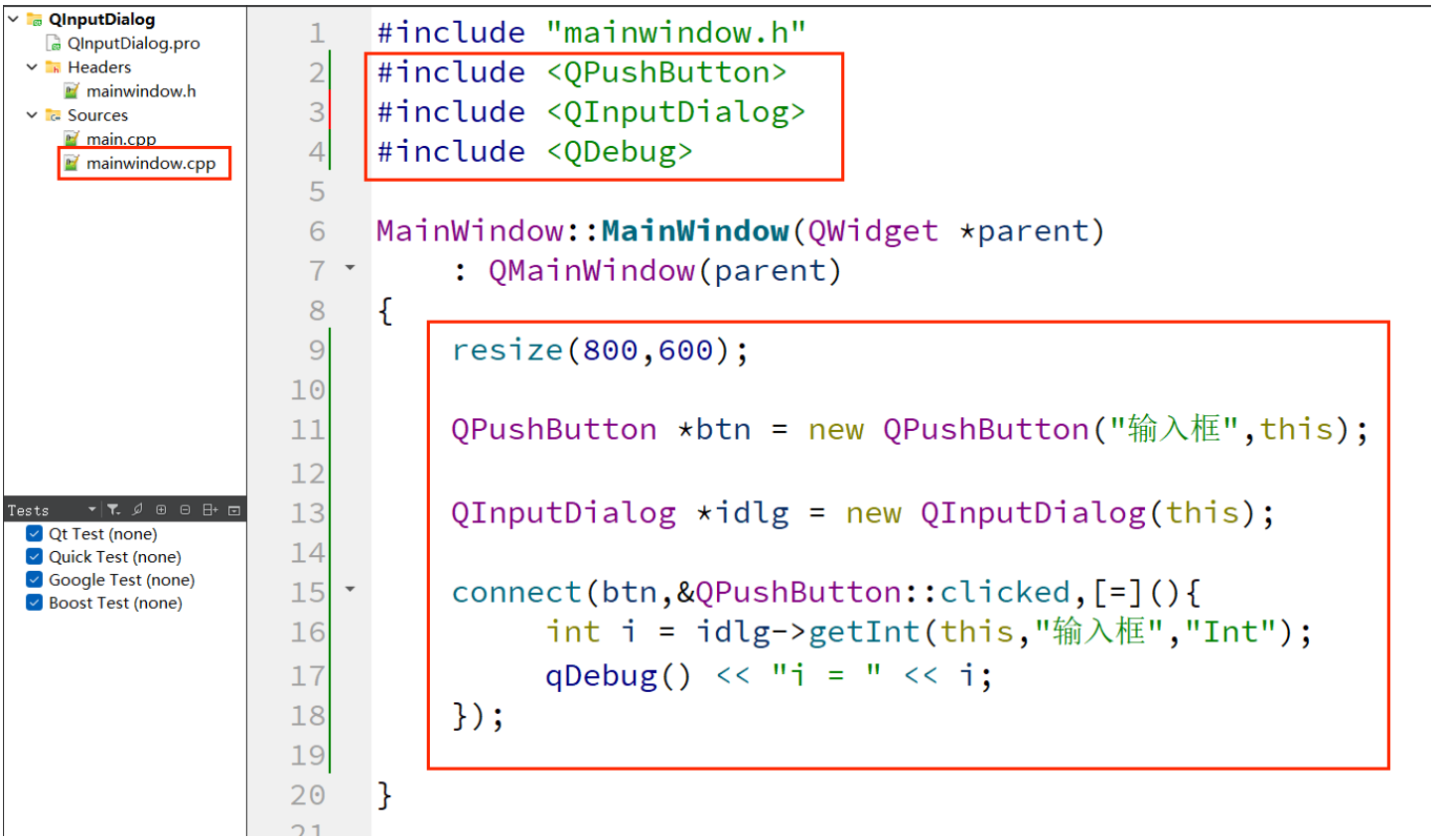



```cpp
#include "mainwindow.h"
#include <QPushButton>
#include <QInputDialog>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("输入框",this);

    QInputDialog *idlg = new QInputDialog(this);

    connect(btn,&QPushButton::clicked,[=](){
        double d = idlg->getDouble(this,"输入框","浮点型");
        qDebug() << "d = " << d;
    });

}
```

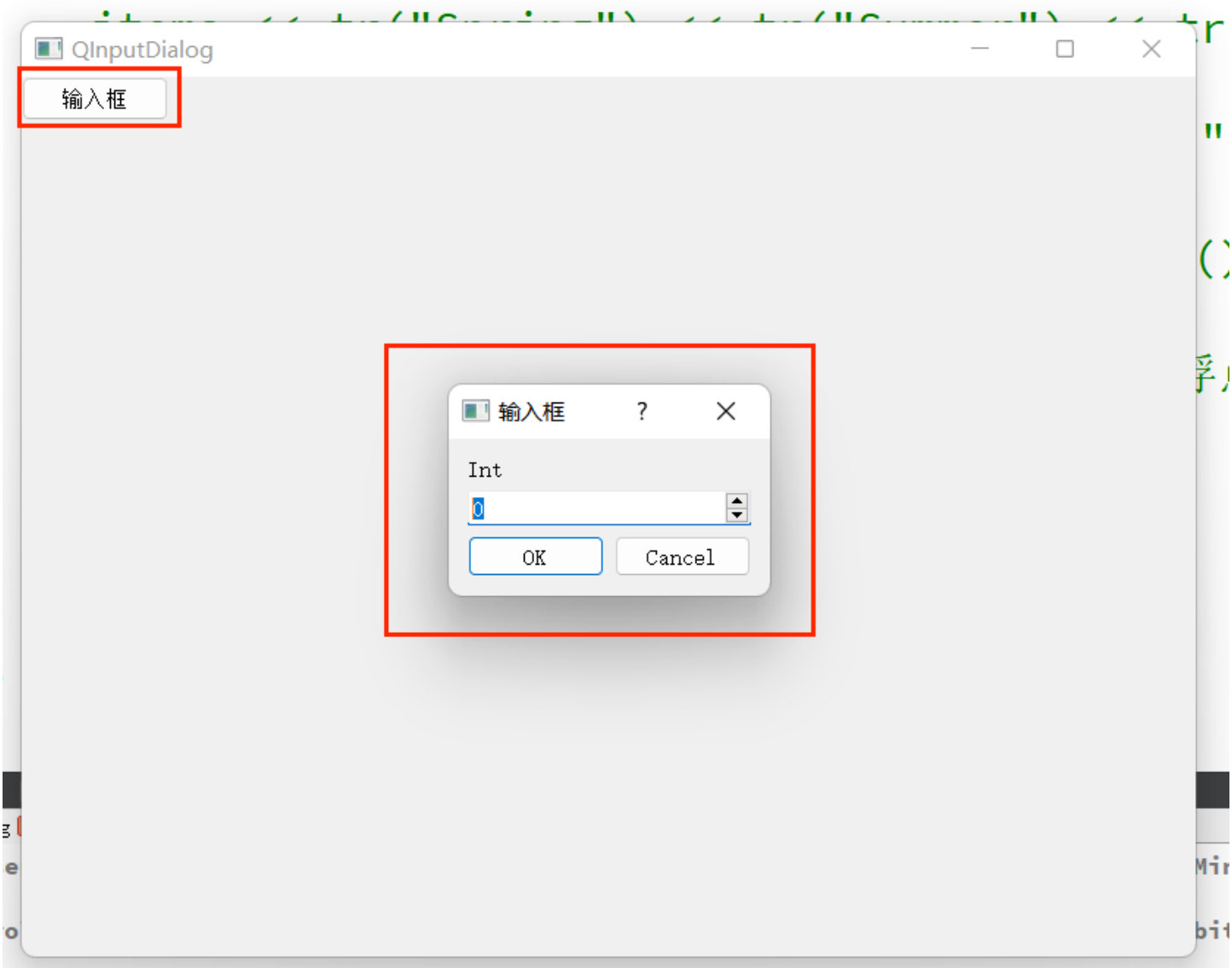
运行效果：



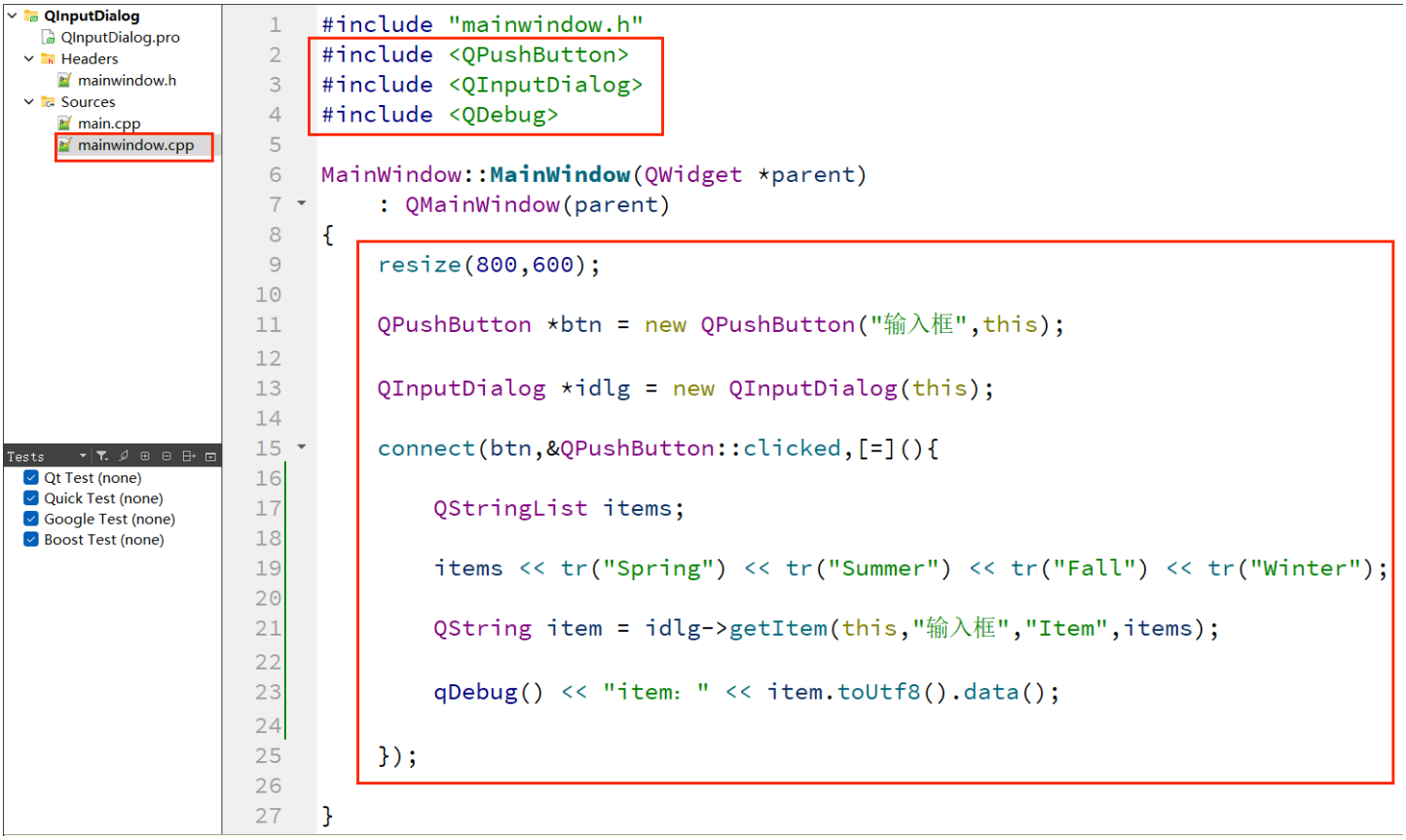
示例2：整型数据输入对话框


QInputDialog
QInputDialog.pro
Headers
mainwindow.h
Sources
main.cpp
mainwindow.cpp
Tests
Qt Test (none)
Quick Test (none)
Google Test (none)
Boost Test (none)
#include "mainwindow.h"
#include <QPushButton>
#include <QInputDialog>
#include <QDebug>
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
{
resize(800,600);
QPushButton *btn = new QPushButton("输入框",this);
QInputDialog *idlg = new QInputDialog(this);
connect(btn,&QPushButton::clicked,[=](){
int i = idlg->getInt(this,"输入框","Int");
qDebug() << "i = " << i;
});
}

运行效果：

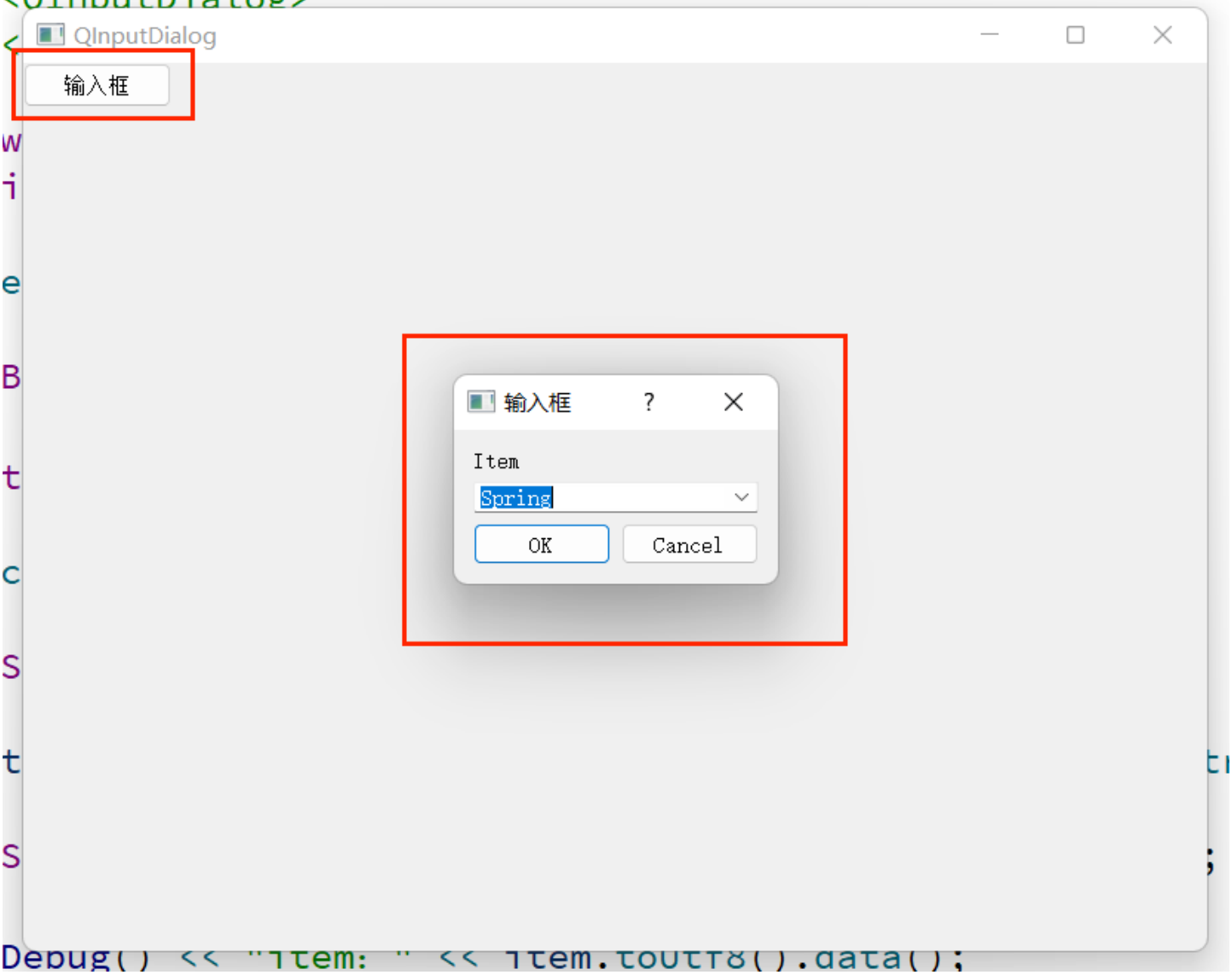

QInputDialog
输入框
输入框
Int
0
OK
Cancel

示例3：打开选择条目对话框




```cpp
#include "mainwindow.h"
#include <QPushButton>
#include <QInputDialog>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    resize(800,600);

    QPushButton *btn = new QPushButton("输入框",this);

    QInputDialog *idlg = new QInputDialog(this);

    connect(btn,&QPushButton::clicked,[=](){

        QStringList items;

        items << tr("Spring") << tr("Summer") << tr("Fall") << tr("Winter");

        QString item = idlg->getItem(this,"输入框","Item",items);

        qDebug() << "item: " << item.toUtf8().data();

    });

}
```

运行效果如下：


QInputDialog
输入框
输入框
Item
Spring
OK
Cancel