UI 文件设计与运行机制

原创 十月的天空 Qt 学习 2019-03-03

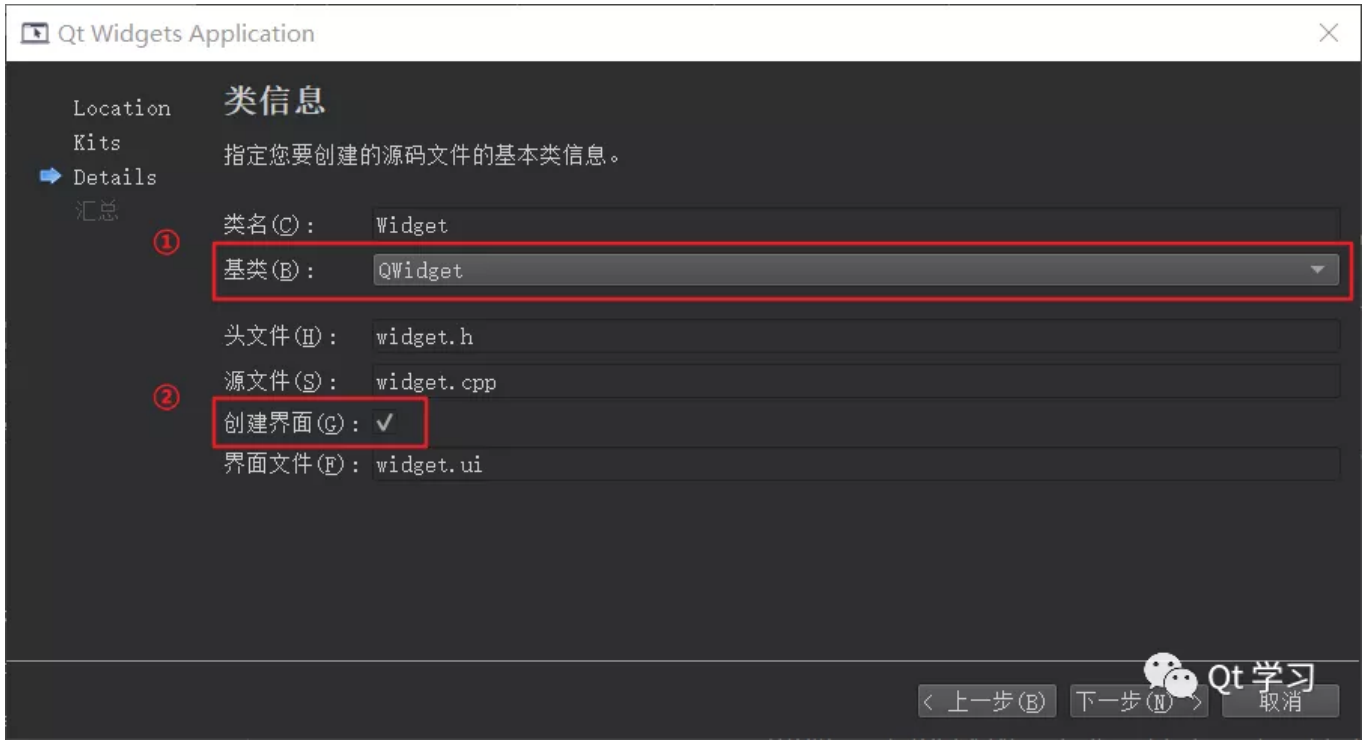
上一篇通过一个 “Hello World” 实例，演示了在 Qt Creator 里创建应用程序、设计窗体界面、编译和运行程序的基本过程。这一篇将介绍可视化设计的 UI 界面文件的原理和运行机制。

本篇目录：

1. 项目文件组成
2. 项目管理文件
3. 界面文件
4. 主函数文件
5. 窗体相关的文件

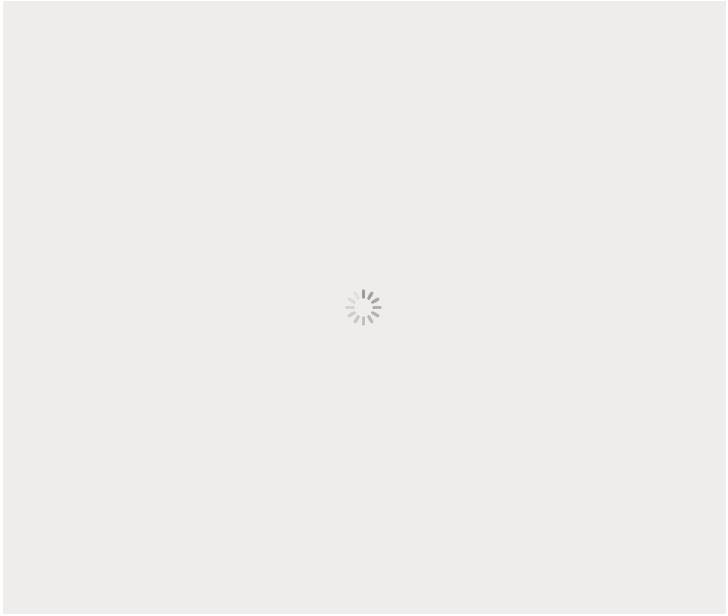
1. 项目文件组成

在 Qt Creator 中新建一个 Widget Application 项目，选择 QWidget 作为窗体基类，并选中 “创建界面” 复选框。



选择基类界面

创建后的项目文件目录树以及各文件说明如下图所示。



项目文件的目录树

说明：在 C++ 里，任何窗体或界面组件都是用类封装的，一般情况下，一个类包含一个头文件 (.h) 和一个源程序文件 (.cpp)。

2. 项目管理文件

项目管理文件 (UImechanism.pro) 用于记录项目的一些设置，以及项目包含文件的组织管理，其内容如下：（可以左右滑动以查看完整代码，下同）

```
#-------------------------------------------------
#
# Project created by QtCreator 2019-03-03T14:41:29
#
```

```
#-------------------------------------------------

QT       += core gui # 在项目中加入用于 GUI 设计的 core gui 模块

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets # 当 Qt 主版本大于 4 时，才加入 widgets 模块

TARGET = UImechanism # 生成的目标可执行文件的名称，即 UImechanism.exe
TEMPLATE = app       # 项目使用的模板是 app，是一般的应用程序

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

CONFIG += c++11

SOURCES += \
        main.cpp \
        widget.cpp

HEADERS += \
```

```
        widget.h

FORMS += \
        widget.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

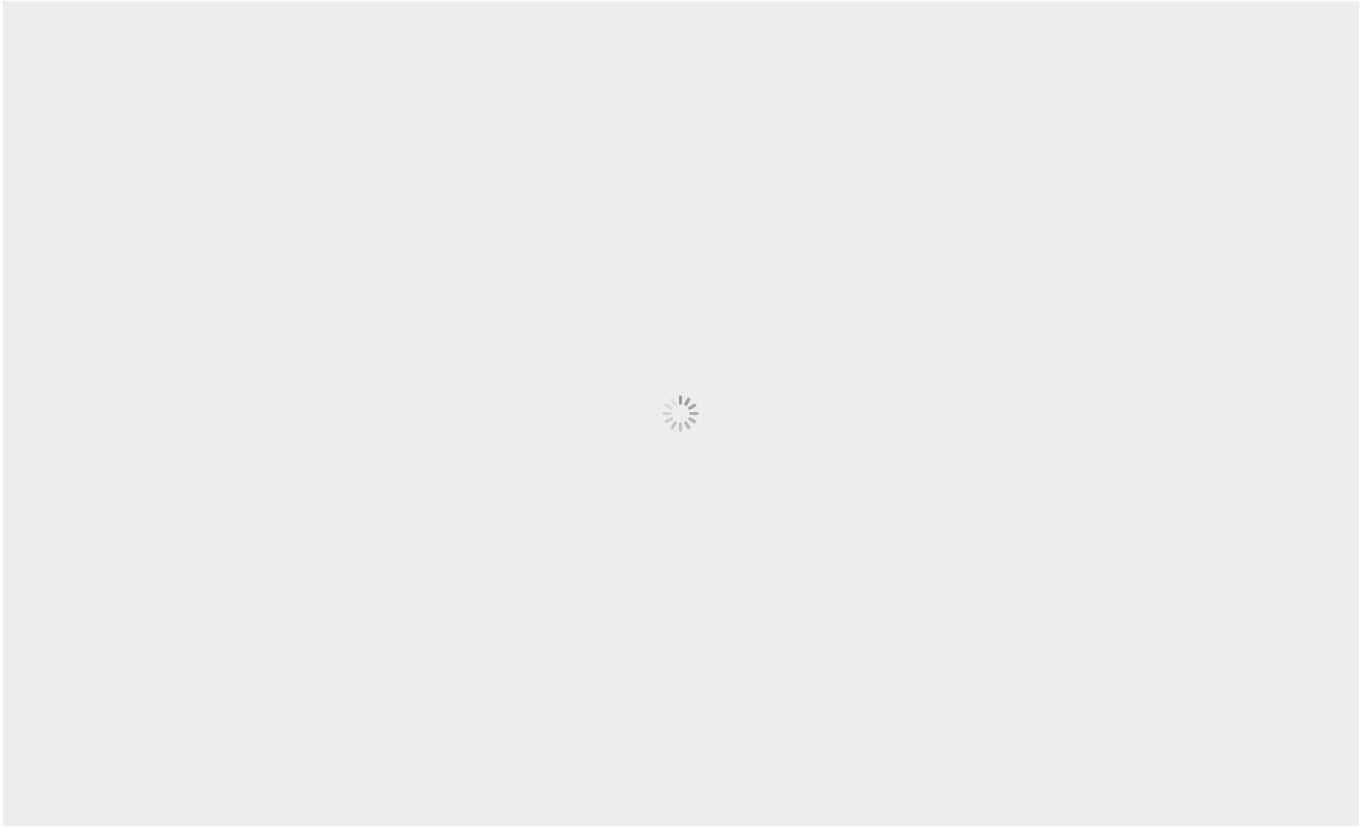
Qt 类库以模块的形式组织各种功能的类，根据项目涉及的功能需求，在项目中添加适当的类库模块支持。例如，如果项目中使用到了涉及数据库操作的类就需要用到 sql 模块，在 .pro 文件中需要增加如下一行：

```
Qt += sql
```

后面的 SOURCES, HEADERS, FORMS 记录了项目中包含的源程序文件、头文件和窗体文件（.ui 文件）的名称。这些文件列表是 Qt Creator 自动添加到项目管理文件里面的，用户不需要手动修改。当添加一个文件到项目，或从项目里删除一个文件时，项目管理文件里的条目会自动修改。

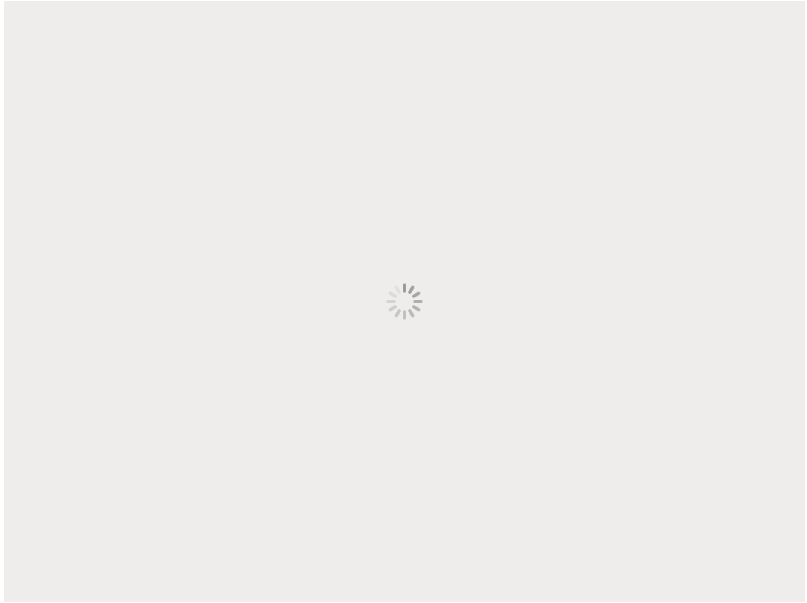
3. 界面文件

双击项目文件目录树中的窗体界面文件 (widget.ui)，打开集成在 Qt Creator 中的 UI 设计器 (Qt Designer)，其功能区域如下图所示。



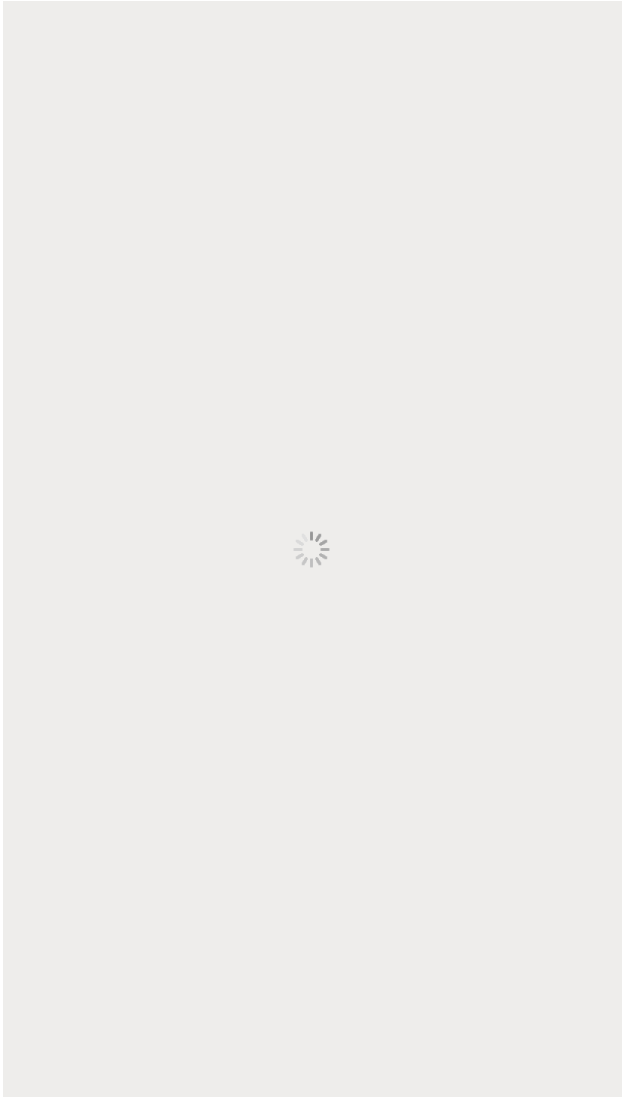
集成在 Qt Creator 中的 UI 设计器

将一个 Label 组件和一个 Push Button 组件拖放到设计的窗体上面，调整好位置。设置 Label 组件的 “text“ 属性为 “Hello, World”，“font“ -> ”点大小” 属性为 20；设置 Push Button 组件的 ”text“ 属性为 “关闭”。



窗体设计界面

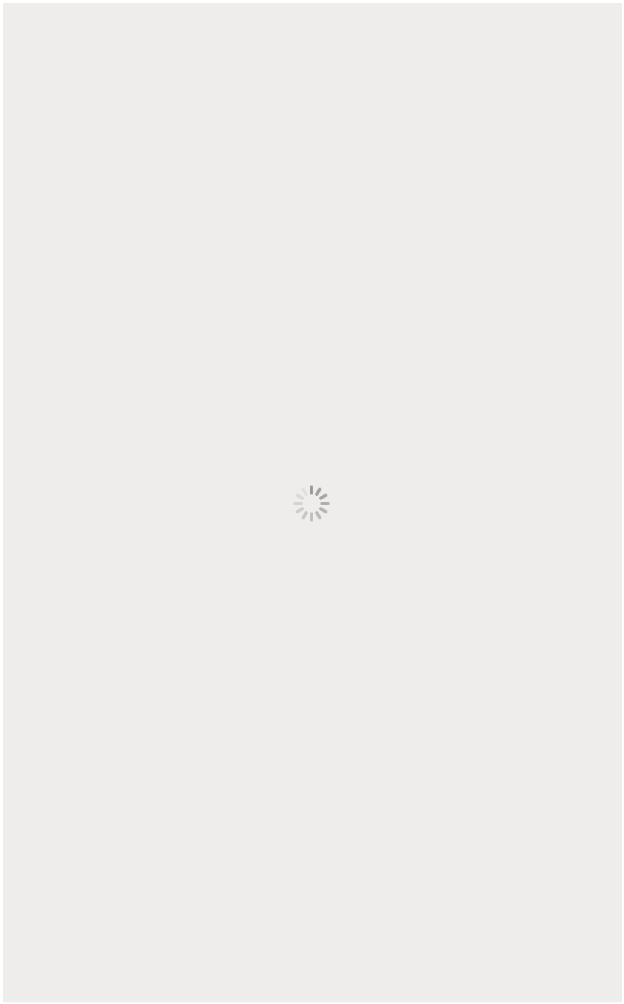
选中窗体上的 Label 组件，属性编辑器的内容如下：



界面组件 Label 的属性编辑器

（属性的多个分组表示了类的继承关系，图中 QLabel 的继承关系是 QObject→QWidget→QFrame→QLabel）

选中窗体上的 Push Button 组件，属性编辑器的内容如下：

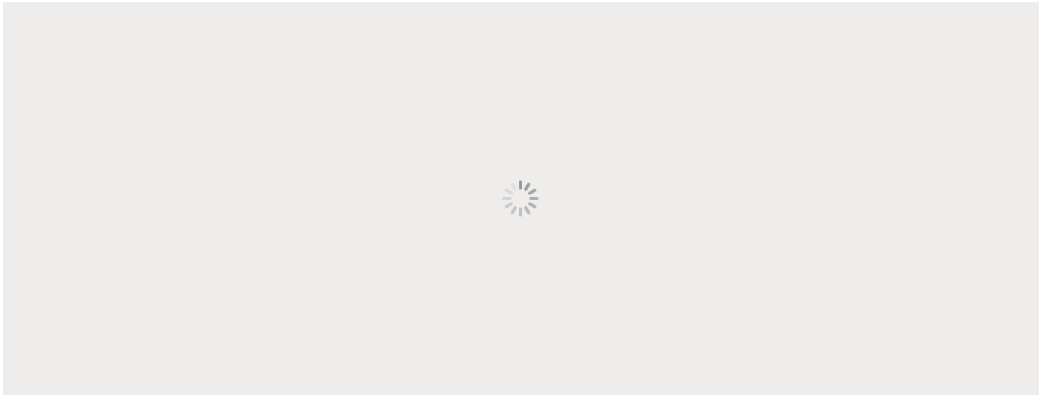


界面组件 Push Botton 的属性编辑器

编辑完两个组件的属性之后，再为 pushButton 按钮添加一个功能：单击此按钮时，关闭窗口，退出程序。使用 "信号与槽" 编辑器完成这个功能，步骤如下：

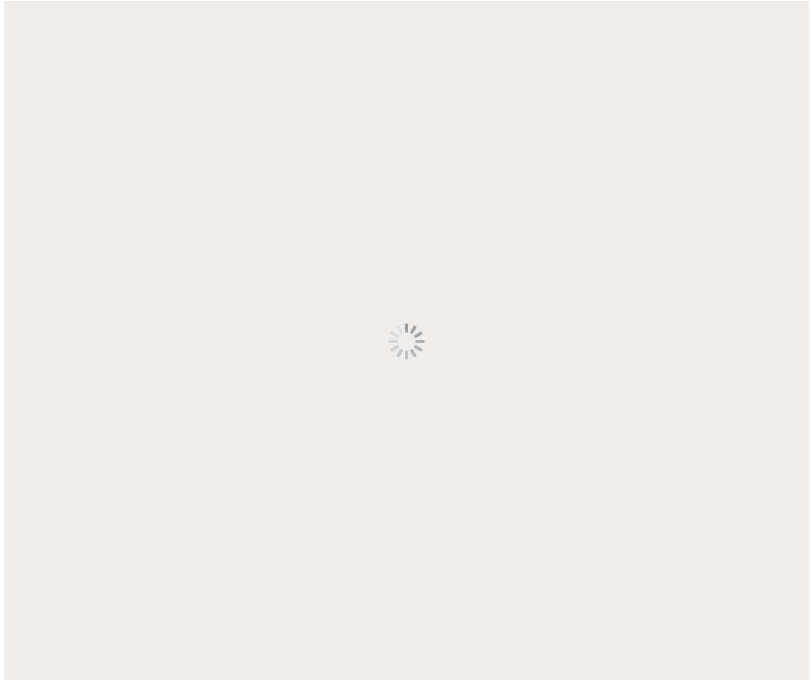
i. 单击 "信号与槽" 编辑器左上角的 "+" 按钮；
ii. 在出现的条目中，

<发送者> 选择 <pushButton>
<信号> 选择 <clicked()>
<接收者> 选择 <Widget>
<槽> 选择 <close()>



信号与槽编辑器中设计信号与槽的关联

对项目进行编译和运行，可以出现如下图所示的窗口，单击“关闭”按钮可以关闭程序。



运行结果

4. 主函数文件

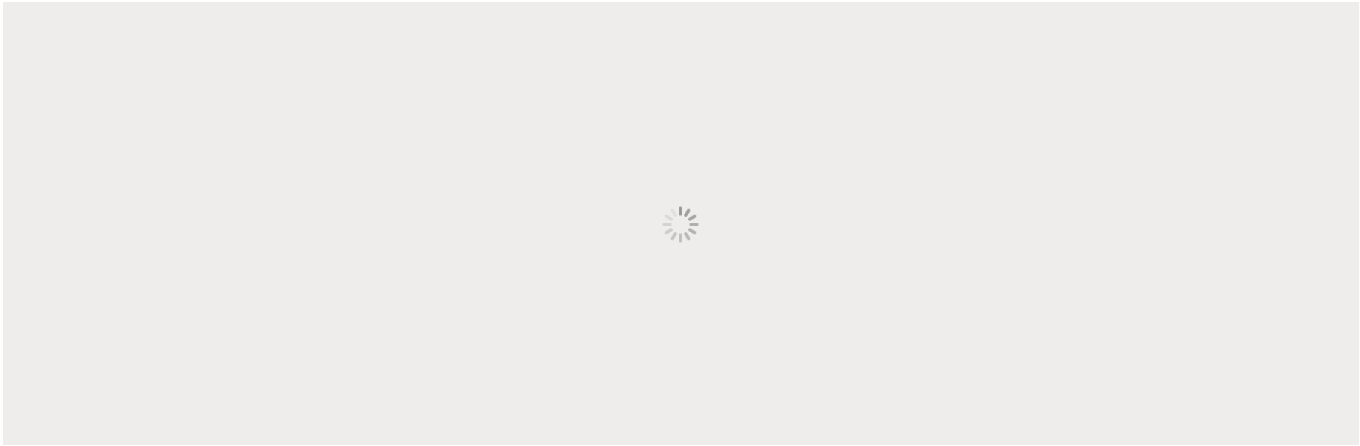
主程序入口文件 main.cpp 是实现 main() 函数的文件，它的主要功能是创建应用程序，创建窗口，显示窗口，并运行应用程序，开始应用程序的消息循环和事件处理，内容如下：

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[])
{
```

```
5      QApplication a(argc, argv); // 定义并创建应用程序
6      Widget w;                   // 定义并创建窗口
7      w.show();                   // 显示窗口
8      return a.exec();            // 应用程序运行
9  }
```

5. 窗体相关的文件

对项目进行编译和运行后，在项目的目录下自动生成一个 ui_widget.h 文件，其中包含一个名称是 Ui_Widget 的类，它是根据窗体上的组件及其属性、信号与槽的关联等自动生成的一个类的定义文件。



为了搞清楚窗体类的定义，以及界面功能的实现原理，下面分别分析各个文件的内容及其功能，以及它们是如何联系在一起工作实现界面的创建与显示的。

窗体类的头文件 widget.h

在创建项目时，选择窗体基类是 QWidget，在 widget.h 中定义了一个继承自 QWidget 的类 Widget，下面是 widget.h 文件的内容与手动添加的注释：

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>

namespace Ui { // 一个命名空间 Ui，包含一个类 Widget
    class Widget;
}

class Widget : public QWidget // 继承于 QWidget 的类 Widget 的定义
{
    Q_OBJECT // 用 Qt 的信号与槽机制的类都必须加入的一个宏

public: // Widget 类的构造函数和析构函数
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private:
    // 使用 Ui::Widget 定义的一个指针，指向可视化设计的界面
    Ui::Widget *ui;
};
#endif
```

窗体类的源文件 widget.cpp

源文件 widget.cpp 是类 Widget 的实现代码，内容与手动添加的注释如下：

```
#include "widget.h"
#include "ui_widget.h" // 编译生成的与 UI 文件 widget.ui 对应的类定义文件

// 执行父类 QWidget 的构造函数，创建一个 Ui::Widget 类的对象 ui
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    // 实现窗口的生成与各种属性的设置、信号与槽的关联
    ui->setupUi(this);
}

Widget::~Widget()
{
    delete ui; // 删除用 new 创建的指针 ui
}
```

窗体界面定义文件 widget.ui

这是一个 XML 文件，定义了窗口上的所有组件的属性设置、布局，及其信号与槽函数的关联等。用 Qt Designer 可视化设计的界面都由 Qt 自动解析，并以 XML 文件的形式保存下来。在设计界面时，只需在 UI 设计器里进行可视化设计即可，而不用管 widget.ui 文件是怎么生成的。 widget.ui 文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Widget</class>
 <widget class="QWidget" name="Widget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Widget</string>
  </property>
  <widget class="QLabel" name="label">
   <property name="geometry">
    <rect>
     <x>100</x>
     <y>80</y>
     <width>211</width>
     <height>51</height>
```

```
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>20</pointsize>
     </font>
    </property>
    <property name="text">
     <string>Hello, World</string>
    </property>
   </widget>
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>160</x>
      <y>220</y>
      <width>89</width>
      <height>24</height>
     </rect>
    </property>
    <property name="text">
     <string>关闭</string>
    </property>
   </widget>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <resources/>
```

```
 <connections>
  <connection>
   <sender>pushButton</sender>
   <signal>clicked()</signal>
   <receiver>Widget</receiver>
   <slot>close()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>204</x>
     <y>231</y>
    </hint>
    <hint type="destinationlabel">
     <x>199</x>
     <y>149</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>

```

ui_widget.h 文件

对 widget.ui 文件进行编译，会生成 ui_widget.h 文件。根据项目 shadow build 编译设置不同，ui_widget.h 会出现在编译后的目录下，或与 widget.ui 同目录。

注意：ui_widget.h 是对 widget.ui 文件编译后自动生成的，widget.ui 又是通过 UI 设计器可视化设计生成的。所以，对 ui_widget.h 手工进行修改没有什么意义，所有涉及界面的修改都应该直接在 Qt Designer 里进行。故 ui_widget.h 也没有必要添加到项目里。

ui_widget.h 文件内容如下：

```
/********************************************************************************
** Form generated from reading UI file 'widget.ui'
** Created by: Qt User Interface Compiler version 5.12.1
** WARNING! All changes made in this file will be lost when recompiling UI file!
********************************************************************************/

#ifndef UI_WIDGET_H
#define UI_WIDGET_H

#include <QtCore/QVariant>
#include <QtWidgets/QApplication>
#include <QtWidgets/QLabel>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QWidget>

QT_BEGIN_NAMESPACE

class Ui_Widget
{
public:
    QLabel *label;
```

```
    QPushButton *pushButton;

    void setupUi(QWidget *Widget)
{
        if (Widget->objectName().isEmpty())
            Widget->setObjectName(QString::fromUtf8("Widget"));
        Widget->resize(400, 300);
        label = new QLabel(Widget);
        label->setObjectName(QString::fromUtf8("label"));
        label->setGeometry(QRect(100, 80, 211, 51));
        QFont font;
        font.setPointSize(20);
        label->setFont(font);
        pushButton = new QPushButton(Widget);
        pushButton->setObjectName(QString::fromUtf8("pushButton"));
        pushButton->setGeometry(QRect(160, 220, 89, 24));

        retranslateUi(Widget);
        QObject::connect(pushButton, SIGNAL(clicked()), Widget, SLOT(close()));

        QMetaObject::connectSlotsByName(Widget);
    } // setupUi

    void retranslateUi(QWidget *Widget)
{
        Widget->setWindowTitle(QApplication::translate("Widget", "Widget", nullptr));
        label->setText(QApplication::translate("Widget", "Hello, World", nullptr));
```

```
49          pushButton->setText(QApplication::translate("Widget", "\345\205\263\351\227\255", nullptr));
50      } // retranslateUi
51
52  };
53
54  namespace Ui {
55      class Widget: public Ui_Widget {};
56  } // namespace Ui
57
58  QT_END_NAMESPACE
59
60  #endif // UI_WIDGET_H
61
```

文件 ui_widget.h 中主要做了以下的一些工作：

- 定义了一个 Ui_Widget 类，用于封装可视化设计的界面；
- 自动生成了界面各个组件的类成员变量定义；
- 定义了 setupUi() 函数，这个函数用于创建各个界面组件，并设置其位置、大小、文字内容、字体等属性，设置信号与槽的关联；
- 定义 namespace Ui 以及定义一个从 Ui_Widget 继承的 Widget 类。

提示： ui_widget.h 文件里实现界面功能的类是 Ui_Widget。再定义一个类 Widget 从 Ui_Widget 继承而来，并定义在 namespace Ui 里，这样 Ui::Widget 与 widget.h 里的类 Widget 同名，但是用 namespace 区分开来。所以，界面的 Ui::Widget 类与文件 widget.h 里定义的 Widget 类实际上是两个类，但是 Qt 的处理让用户感觉不到 Ui::Widget 类的存在，只需要知道在 Widget 类里用 ui 指针可以访问可视化设计的界面组件就可以了。

喜欢此内容的人还喜欢

从“多国渴望回归中国”到假瑞士专家，是哪些人在抹黑中国?

李未熟擒话

生一场病，才知道谁最爱你。

蕊希