



北京大学
PEKING UNIVERSITY

Qt 编程

Qt 基本介绍



- 官网: <https://www.qt.io/>
- 下载: <https://download.qt.io/>
- 跨平台可视化开发工具 (Winows / Mac / Linux / Android / IOS)

Design, Develop & Deploy User Interfaces and Applications

Target embedded, desktop, and mobile platforms with the same code base for all.

DESIGN TOOLS →

Visual 2D/3D UI Editor with ready-made UI components. Design from prototype to production.

DEVELOPMENT TOOLS →

Cross-platform IDE with a large variety of tools and extensions that enhance the user experience of Qt developers.

THE FRAMEWORK →

The most complete set of libraries for UI development, business logic and machine-to-machine communication using imperative C++ and other declarative approaches.

SUPPORTED PLATFORMS →

High-end support of operating systems for



Qt 基本介绍



- 官网: <https://www.qt.io/>
- 下载: <https://download.qt.io/>
- 跨平台可视化开发工具 (Winows / Mac / Linux / Android / IOS)
- 有收费版和免费的开源版 (Open source distribution under an LGPL or GPL license)
- 免费版开发的桌面应用, 发布时不要求开源, 须带一些动态链接库, 无法静态绑定到一个 .exe 文件

Qt 基本介绍



- 官网: <https://www.qt.io/>
- 下载: <https://download.qt.io/>
- **跨平台可视化开发工具** (Winows / Mac / Linux / Android / IOS)
- 有收费版和免费的开源版 (Open source distribution under an LGPL or GPL license)
- 免费版开发的桌面应用, 发布时不要求开源, 须带一些动态链接库, 无法静态绑定到一个 .exe 文件
- **Qt 是核心库, 类比Python**
- **Qt Creator 是IDE**, 用于开发桌面应用 (如 **WPS office**), 类比Pycharm
- **Qt Quick** 可用于开发移动应用

Qt 基本介绍

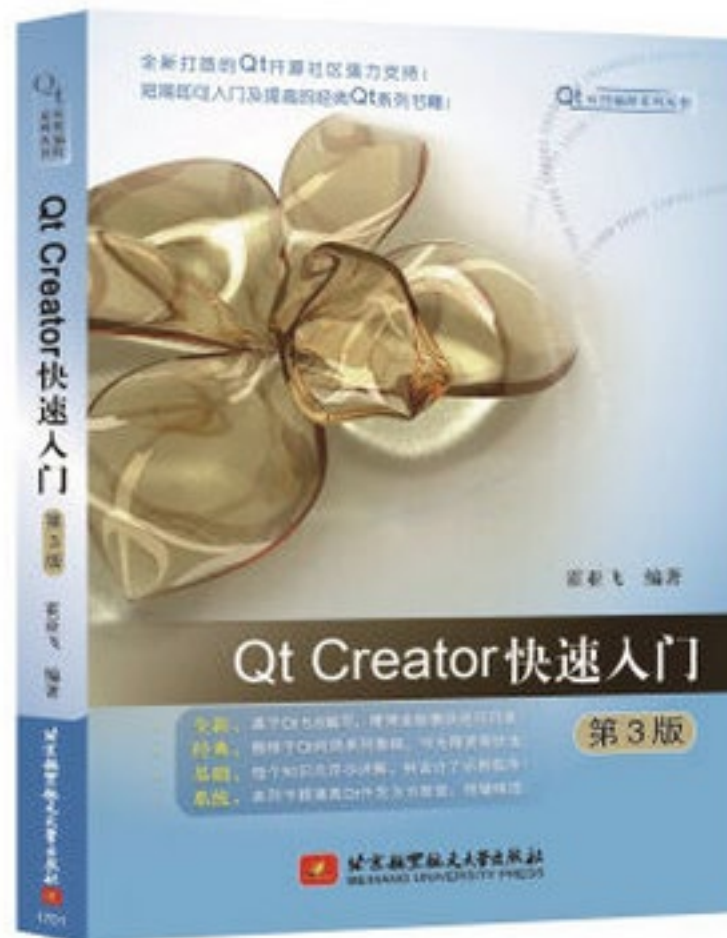
下载地址:

https://download.qt.io/new_archive/qt/5.6/5.6.1-1/

Qt Creator快速入门（第3版）霍亚飞 著

•资源网站 (包括此书源码)

<http://www.qter.org/>



Qt 基本介绍

用Qt创建桌面应用工程

- 1) 运行 Qt Creator
- 2) 新建Qt Widgets项目（即带窗口界面的项目）
- 3) 指定文件夹和项目名称
- 4) 指定一个源文件中的类名以及其基类 (QMainWindow或QDialog ...)
- 5) 工程创建完成

Qt New File or Project



选择一个模板:

所有模板 ▾

项目

Application

Library

其他项目

Non-Qt Project

Import Project

文件和类

C++

Modeling

Qt

GLSL

General

Java

Python

Qt Widgets Application

Qt Console Application

Qt Quick Application

Qt Quick Controls 2 Application

Qt Quick Controls Application

Qt Canvas 3D Application

Qt Labs Controls Application

创建一个桌面Qt应用，包含一个基于Qt设计师的主窗体。

预选一个可用的Qt桌面版本用于编译程序。

支持的平台：桌面

Choose...

Cancel



Qt Widgets Application

➡ Location

Kits

Details

汇总

项目介绍和位置

This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.

名称: test1|

创建路径: C:\diskd\qtthing\test1


浏览...

☐ 设为默认的项目路径

下一步(N)

取消



←  Qt Widgets Application

Kit Selection

Location


➔ Kits

Details

汇总

Qt Creator can use the following kits for project **test1**:

☒ Select all kits


☒  Desktop Qt 5.6.1 MinGW 32bit

详情 ▼

下一步(N)

取消



←  Qt Widgets Application

类信息

Location

Kits

➔ Details

汇总

指定您要创建的源码文件的基本类信息。

类名(C):

基类(B):

头文件(H):

源文件(S):

创建界面(G): ☒

界面文件(F):

下一步(N)

取消

Qt 基本介绍

- Qt项目文件组成

<code>helloworld.pro</code>	项目文件
<code>hellodialog.h</code>	用户自定义类的头文件
<code>hellodialog.cpp</code>	用户自定义类的源文件
<code>main.cpp</code>	程序的入口文件, 包括main函数
<code>hellodialog.ui</code>	程序的界面文件 (XML格式, 只能可视化编辑)

Qt 文件示例

`hellodialog.h`

用户自定义类的头文件

```
#ifndef HELLODIALOG_H
#define HELLODIALOG_H

namespace Ui {    // 界面的名字空间
    class HelloDialog;
}

class HelloDialog : public QDialog {
    Q_OBJECT;    // 宏定义
public:
    explicit HelloDialog(QWidget * parent = 0);
    ~HelloDialog();
```

Qt 文件示例

```
private:
```

```
    Ui::HelloDialog * ui;
```

```
};
```

```
#endif // HELLODIALOG_H
```

Qt 文件示例

- 发布Qt编写的可执行程序

- 1) 在左下角选择**release**版并编译,在**release**版文件夹下面得到 **.exe**文件
- 2) 将以下 **.dll**文件和 **.exe**文件放在同一文件夹下一起发布

`libgcc_s_dw2-1.dll`

`libstdc++-6.dll`

`libwinpthread-1.dll` (也可能不要)

`Qt5Core.dll`

`Qt5Gui.dll`

`Qt5Widgets.dll`

Qt 文件示例

- 发布Qt编写的可执行程序

最保险的带上全部.dll的方法:

1) 启动 Qt 5.6 for desktop 命令行窗口

2) 将 .exe 文件放到一个空文件夹, 如 c:\tmp, 然后运行:

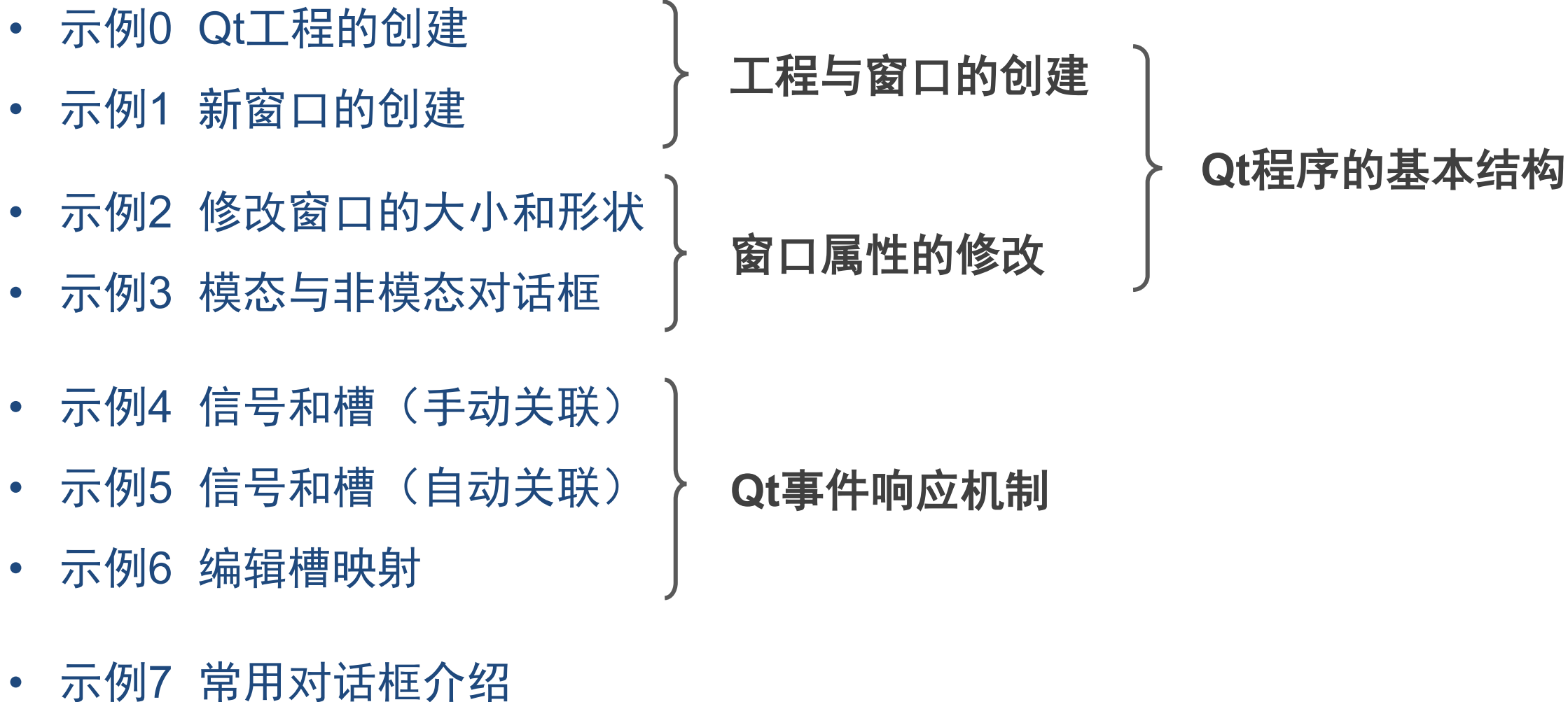
```
windeployqt c:\tmp
```

则在c:\tmp 文件下生成所有需要的文件 (约**40M**, 未必每个文件都需要), 将这些文件和 .exe一起打包发布

目录

- 示例0 Qt工程的创建
- 示例1 新窗口的创建
- 示例2 修改窗口的大小和形状
- 示例3 模态与非模态对话框
- 示例4 信号和槽（手动关联）
- 示例5 信号和槽（自动关联）
- 示例6 编辑槽映射
- 示例7 常用对话框介绍

示例关系



示例0 创建Qt工程

演示如何创建一个简单的Qt工程

- 基类的选择与区别
- Qt工程的文件结构
- 各种文件的代码解读
- 窗口界面的设计与修改

示例0 创建Qt工程

- 创建工程类选择

← Qt Widgets Application

Location
Kits
Details
汇总

类信息
指定您要创建的源码文件的基本类信息。

类名 (C): MyHelloDialog

基类 (B): QDialog

头文件 (H): myhellodialog.h

源文件 (S): myhellodialog.cpp

创建界面 (G): ☒

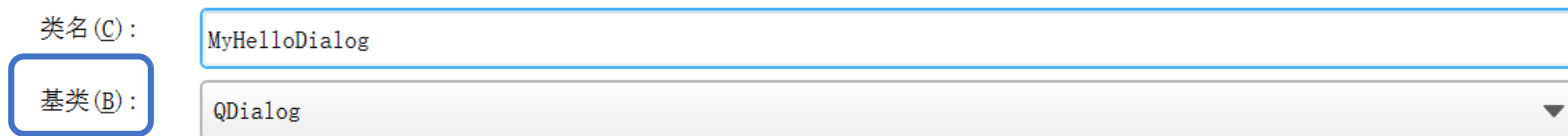
界面文件 (F): myhellodialog.ui

从基类QDialog派生而来

由Qt自动生成

示例0 创建Qt工程

- 不同“基类”的差别



A screenshot of the Qt Creator 'Create Class' dialog. The 'Class Name' (类名(C)) field contains 'MyHelloDialog'. The 'Base Class' (基类(B)) dropdown menu is open, showing 'QDialog' as the selected option. The 'Base Class' label and the dropdown menu are highlighted with blue boxes.

基类 有QMainWindow, QDialog, QWidget可选

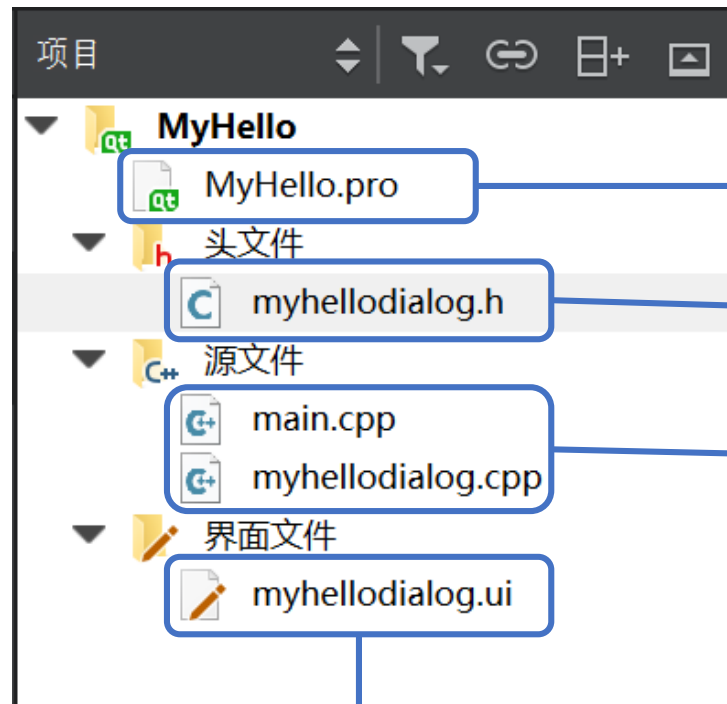
- QWidget是其他两个类的基类, 较为通用
- QMainWindow是有菜单栏的窗口
- QDialog显示一个临时的对话框

参考信息:

<https://stackoverflow.com/questions/3298792/whats-the-difference-between-qmainwindow-qwidget-and-qdialog>

示例0 创建Qt工程

- QT工程文件结构



自动生成的工程文件, 一般不改动

头文件, 存放类定义

源代码文件, 存放代码执行逻辑

界面文件, 控制界面布局

示例0 创建Qt工程

- 头文件myhellodialog.h解读

```
< > myhellodialog.h <选择符号>
1  #ifndef MYHELLODIALOG_H
2  #define MYHELLODIALOG_H
3
4  #include <QDialog>
5
6  namespace Ui {
7      class MyHelloDialog;
8  }
9
10 class MyHelloDialog: public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     explicit MyHelloDialog(QWidget *parent = 0);
16     ~MyHelloDialog();
17
18 private:
19     Ui::MyHelloDialog *ui;
20 };
21
22 #endif // MYHELLODIALOG_H
23
```

在不同名字空间中，不是同一个类

与事件响应相关的宏

当前窗口的父窗口，实现子窗口与父窗口之间的通信，缺省值为0 → 没有父窗口

指向ui名字空间中的MyHelloDialog类，用于访问界面中的各种控件

示例0 创建Qt工程

- 主函数main.cpp解读

```
< > 🔒 G+ main.cpp ⚙️ ✕ <选
1  #include "myhellodialog.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MyHelloDialog w;
8      w.show();
9
10     return a.exec();
11 }
12
```

Qt自动生成, 根据命令行
参数创建应用

创建对话框并显示, 阻塞直到关闭窗口

示例0 创建Qt工程

- 类函数myhellodialog.cpp解读



```
< > myhellodialog.cpp <选择符号>
1  #include "myhellodialog.h"
2  #include "ui_myhellodialog.h"
3
4  MyHelloDialog::MyHelloDialog(QWidget *parent) :
5      QDialog(parent),
6      ui(new Ui::MyHelloDialog)
7  {
8      ui->setupUi(this);
9  }
10
11  MyHelloDialog::~MyHelloDialog()
12  {
13      delete ui;
14  }
15
```

调出界面

删除界面对象

示例0 创建Qt工程

- 窗口界面编辑

The screenshot displays the Qt Designer interface for editing a window interface. The main window is titled 'myhellodialog.ui'. On the left, a sidebar contains icons for '欢迎' (Welcome), '编辑' (Edit), '设计' (Design), 'Debug', '项目' (Project), and '帮助' (Help). The '设计' (Design) mode is selected, which is highlighted by a blue box and a blue arrow pointing to the '可视化编辑模式, 较常用' (Visual editing mode, commonly used) text. The '编辑' (Edit) mode is also highlighted by a blue box and a blue arrow pointing to the '文本化编辑模式, 较少用' (Text-based editing mode, less commonly used) text. The main area shows a grid of widgets and layouts. A 'Filter' dropdown is visible. Below the 'Layouts' section, there are 'Spacers' and 'Buttons' sections. The 'Buttons' section includes 'OK Push Button' and 'Tool Button'. A 'Text Editor' window is open, showing the XML code for the UI file. The code is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3   <class>MyHelloDialog</class>
4   <widget class="QDialog" name="MyHelloDialog">
5     <property name="geometry">
6       <rect>
7         <x>0</x>
8         <y>0</y>
9         <width>400</width>
```

A blue box labeled '文本化编辑界面' (Text-based editing interface) points to the 'Text Editor' window.

名称	使用	文本	快捷键	可选的
OK				
Tool Button				

示例0 创建Qt工程

- 窗口添加新组件

The screenshot shows the Qt Designer interface for a file named `myhellodialog.ui*`. The interface is divided into several panels:

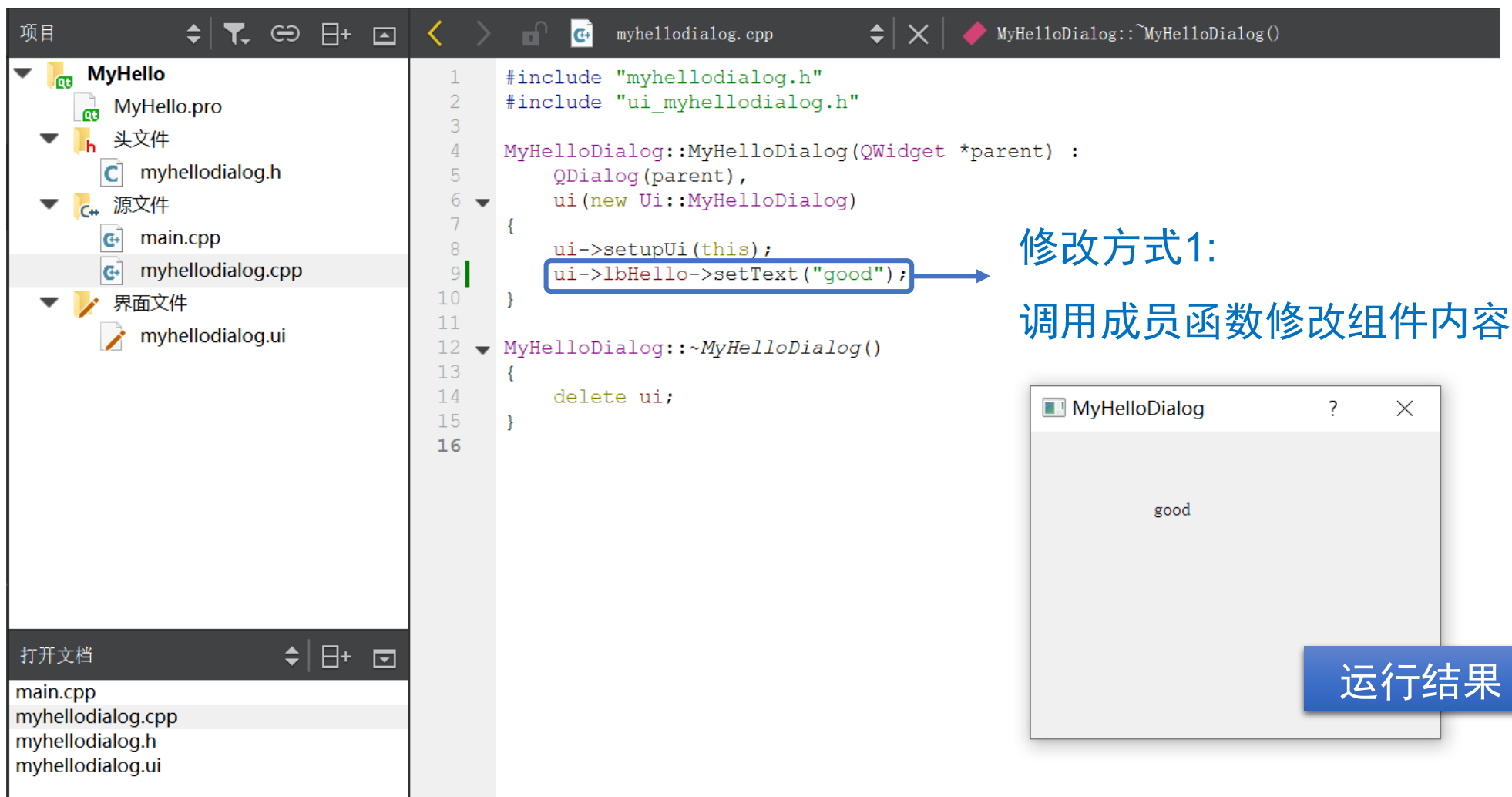
- Filter**: A search bar at the top of the widget palette.
- Display Widgets**: A list of available widgets. The **Label** widget is highlighted with a blue box. A blue arrow points from this box to the text: "选择label (标签: 文本/图片), 并拖动到界面上".
- Central Canvas**: A grid-based workspace where a `TextLabel` widget has been placed. A blue box surrounds the widget, and a blue arrow points to it with the text: "新添加的组件".
- Object Inspector**: Located on the right, it shows the hierarchy of objects. Under `MyHelloDialog` (class `QDialog`), there is an object `lbHello` of class `QLabel`.
- Property Inspector**: Also on the right, it shows the properties of the selected `lbHello` object. The `objectName` property is highlighted with a blue box, and a blue arrow points to it with the text: "修改label对象的名字".

名称	使用	文本	快捷键	可选的
名称	使用	文本	快捷键	可选的

属性	值
QObject	
objectName	lbHello
QWidget	
enabled	<input checked="" type="checkbox"/>
geometry	[(120, 60), 81 x 31]
sizePolicy	[Preferred, Preferre...]

示例0 创建Qt工程

- 修改组件内容 (通过代码)



The screenshot displays the Qt Creator IDE interface. On the left, the 'Project' pane shows the file structure of the 'MyHello' project, including 'MyHello.pro', header files, source files, and UI files. The 'myhellodialog.cpp' file is selected. The main editor window shows the C++ code for 'MyHelloDialog.cpp'. The code includes headers for 'myhellodialog.h' and 'ui_myhellodialog.h'. The constructor 'MyHelloDialog::MyHelloDialog(QWidget *parent)' initializes a 'QDialog' and a 'Ui::MyHelloDialog' object. Line 9, which is highlighted with a blue box, shows the call to 'ui->lbHello->setText("good");'. A blue arrow points from this line to the text '修改方式1: 调用成员函数修改组件内容'. The destructor 'MyHelloDialog::~MyHelloDialog()' calls 'delete ui;'. At the bottom left, the 'Open Documents' pane lists the project files. On the right, a preview of the application window titled 'MyHelloDialog' is shown, displaying the text 'good'. A blue button labeled '运行结果' (Run Result) is positioned below the window preview.

```
1 #include "myhellodialog.h"
2 #include "ui_myhellodialog.h"
3
4 MyHelloDialog::MyHelloDialog(QWidget *parent) :
5     QDialog(parent),
6     ui(new Ui::MyHelloDialog)
7 {
8     ui->setupUi(this);
9     ui->lbHello->setText("good");
10 }
11
12 MyHelloDialog::~MyHelloDialog()
13 {
14     delete ui;
15 }
16
```

修改方式1:
调用成员函数修改组件内容

运行结果

示例0 创建Qt工程

- 修改组件内容 (通过属性栏)

运行结果

点击运行

修改方式2: 修改属性栏中的值

对象	类
MyHelloDialog	QDialog
lbHello	QLabel

属性	值
inputMethodHints	ImhNone
QFrame	
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
midLineWidth	0
QLabel	
text	hello
textFormat	AutoText

示例0 创建Qt工程

- 修改组件内容 (通过属性栏)

myhellodialog.ui

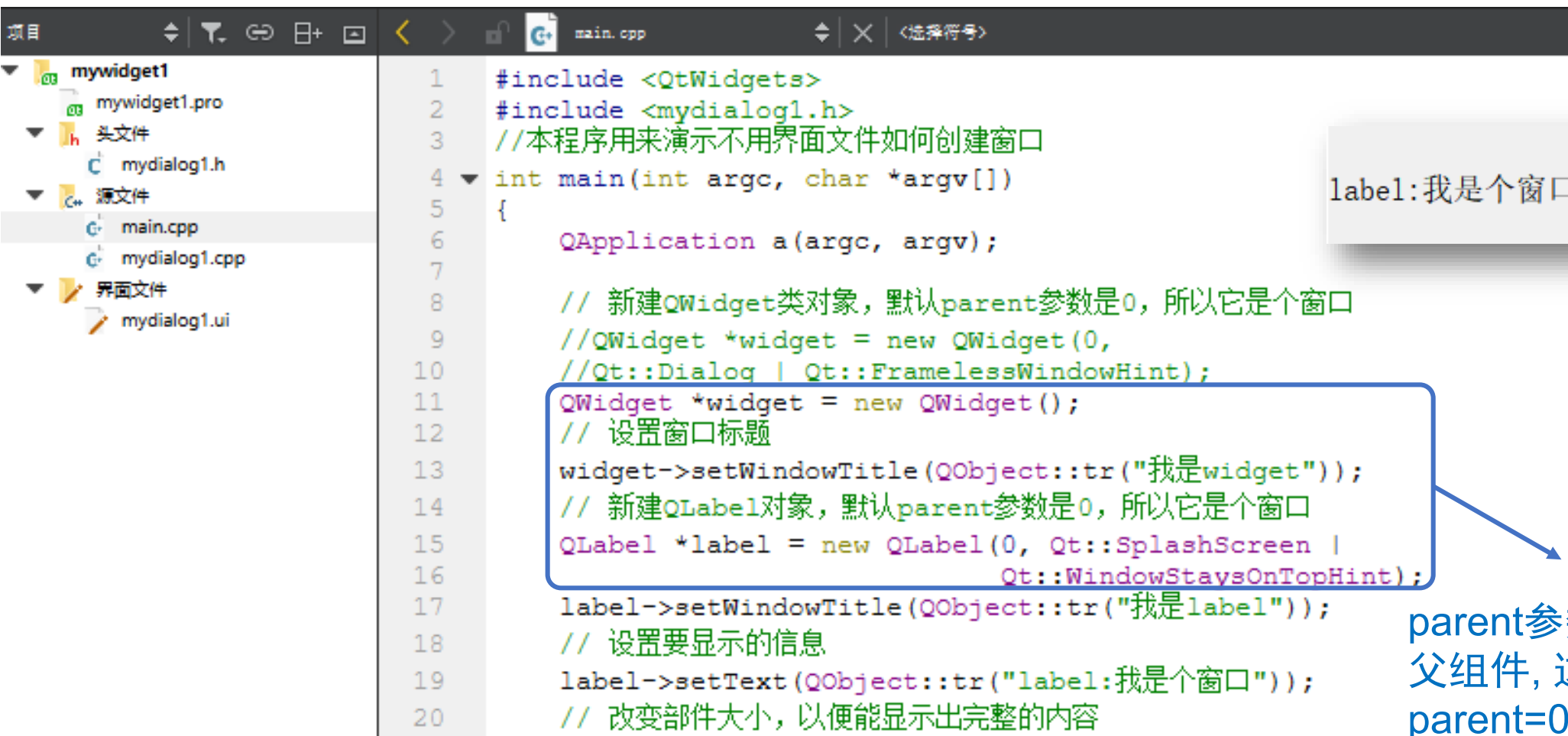
```
16 <widget class="QLabel" name="lbHello">
17   <property name="geometry">
18     <rect>
19       <x>120</x>
20       <y>60</y>
21       <width>81</width>
22       <height>31</height>
23     </rect>
24   </property>
25   <property name="text">
26     <string>hello</string>
27   </property>
28 </widget>
29 </widget>
```

在属性栏修改组件内容后，
ui文件中对应属性自动进行修改

示例1 创建窗口

- 工程创建完毕, 创建第一个Qt窗口
- 不用界面文件创建窗口
- 直接使用界面文件创建窗口

示例1 创建窗口



```
1 #include <QtWidgets>
2 #include <mydialog1.h>
3 //本程序用来演示不用界面文件如何创建窗口
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7
8     // 新建QWidget类对象, 默认parent参数是0, 所以它是个窗口
9     //QWidget *widget = new QWidget(0,
10    //Qt::Dialog | Qt::FramelessWindowHint);
11    QWidget *widget = new QWidget();
12    // 设置窗口标题
13    widget->setWindowTitle(QObject::tr("我是widget"));
14    // 新建QLabel对象, 默认parent参数是0, 所以它是个窗口
15    QLabel *label = new QLabel(0, Qt::SplashScreen |
16                               Qt::WindowStaysOnTopHint);
17    label->setWindowTitle(QObject::tr("我是label"));
18    // 设置要显示的信息
19    label->setText(QObject::tr("label:我是个窗口"));
20    // 改变部件大小, 以便能显示出完整的内容
```

运行结果

label:我是个窗口

parent参数指定了组件的父组件, 这里默认参数parent=0, 即没有父组件, 因此创建了一个顶层窗口

示例1 创建窗口

接上一页 main.cpp

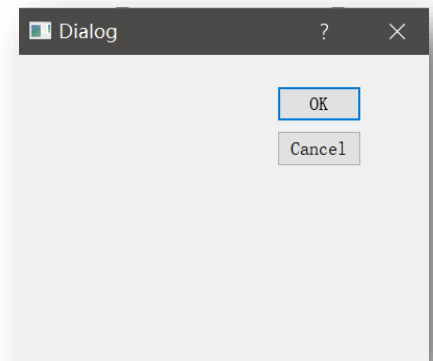
```
20 // 改变部件大小, 以便能显示出完整的内容
21 label->resize(380, 80);
22 // label2指定了父窗口为widget, 所以不是窗口
23 QLabel *label2 = new QLabel(widget);
24 label2->setText(QObject::tr("label2:我不是独立窗口, 只是widget的子部件"));
25 label2->resize(350, 30);
26 // 在屏幕上显示出来
27 label->show();
28 widget->show(); //widget默认大小: 正好包含 label2
29 //再加一个窗口, 则两个带标题栏的窗口都关闭时, 程序结束
30 QWidget *widget2 = new QWidget();
31 // 设置窗口标题
32 widget2->setWindowTitle(QObject::tr("我是widget2"));
33 widget2->show();
34 MyDialog1 * dlg = new MyDialog1();
35 dlg->show();
```

指定parent为widget,
因此是widget的组件

创建一个空窗口widget2

创建一个对话框dialog

运行结果



示例1 创建窗口

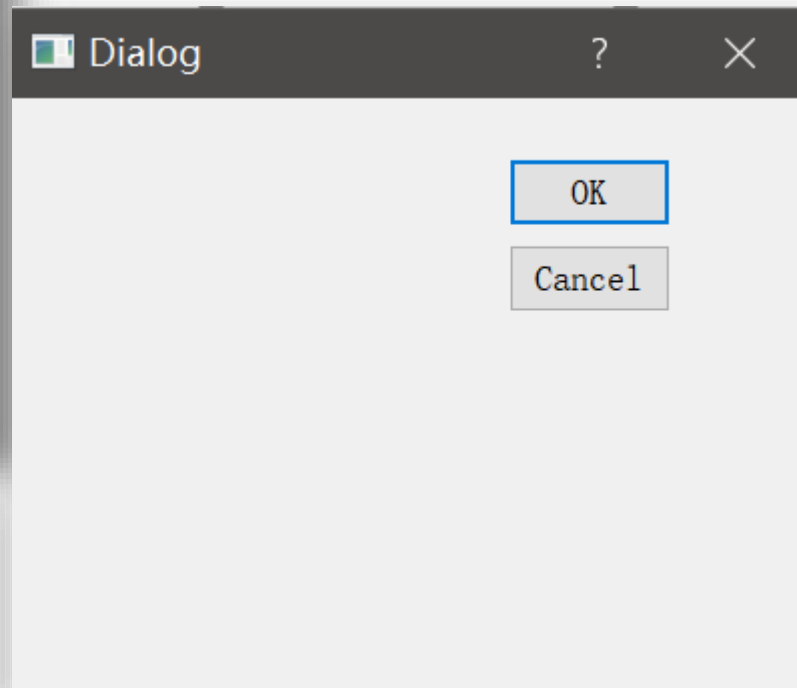
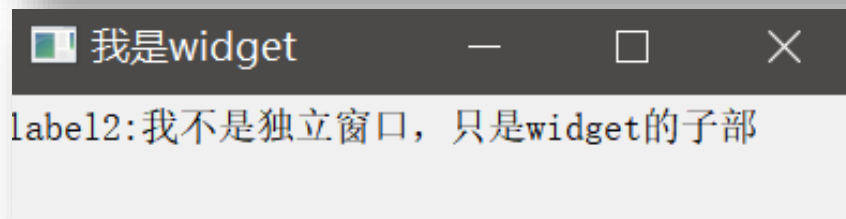
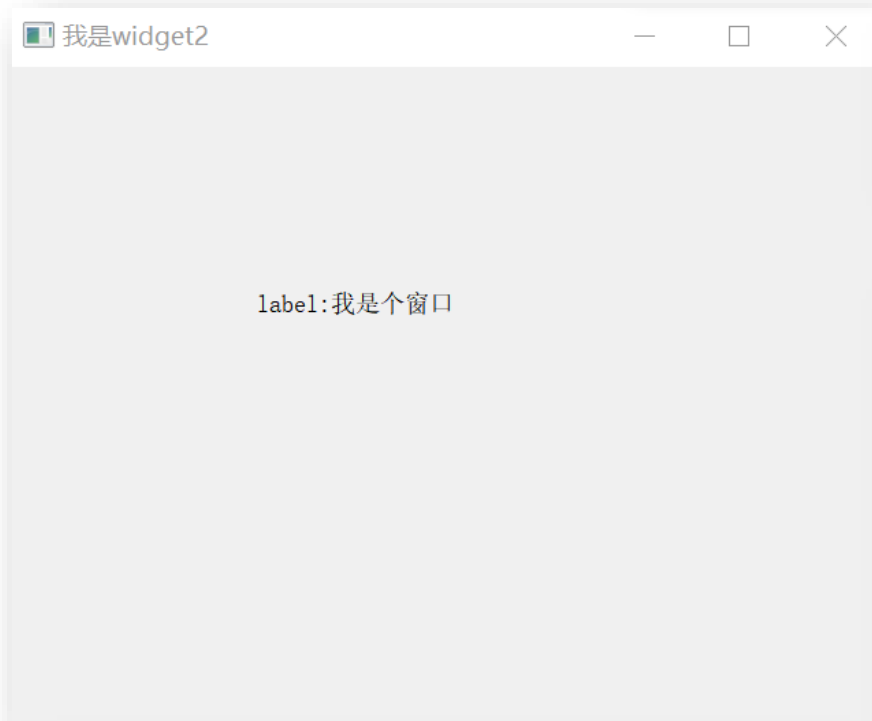
- 运行结果：

1) “Widget2” 是一个独立窗口

2) “label” 是一个窗口

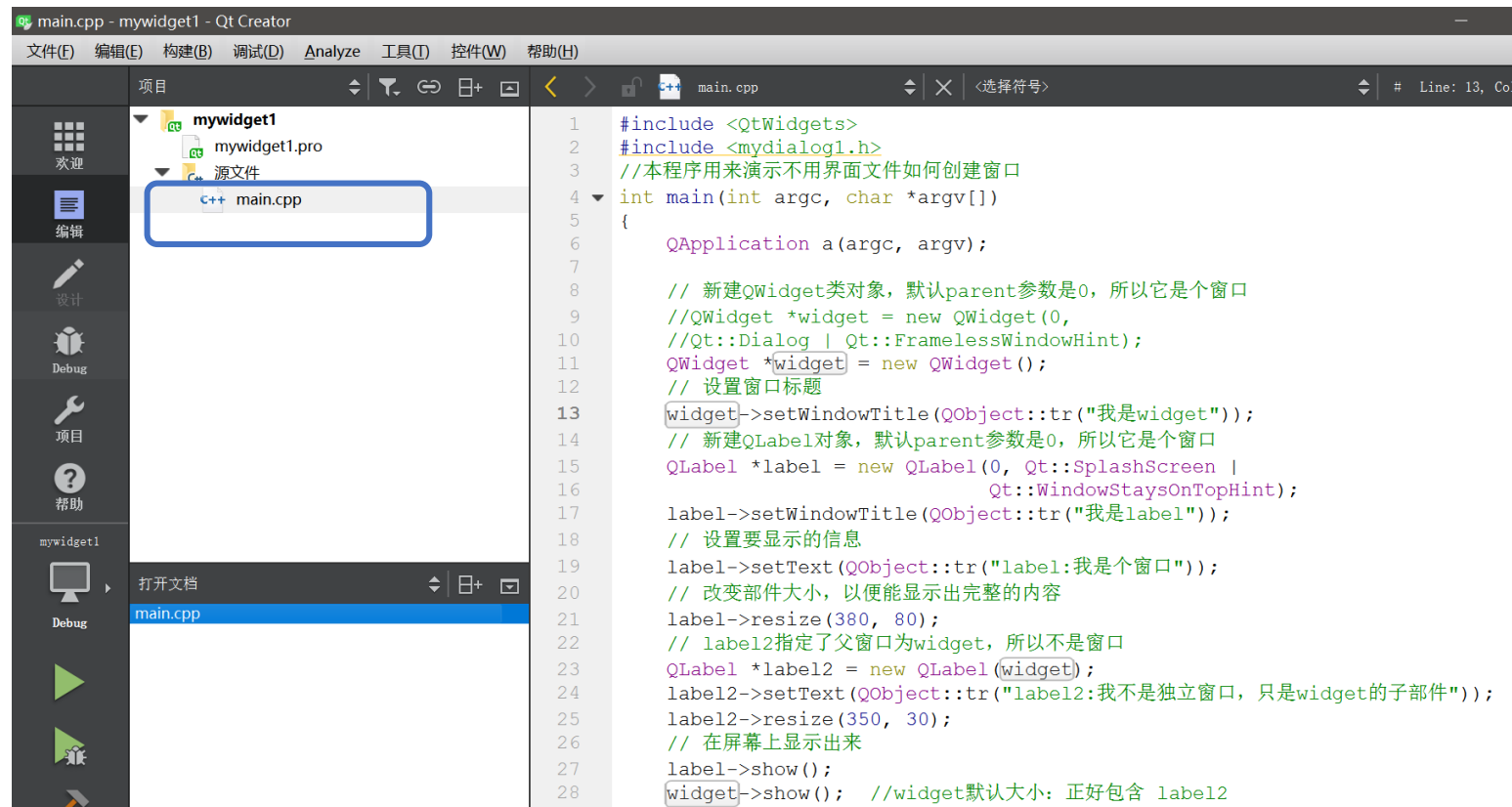
3) “我是widget” 是一个窗口, 其中label2是它的部件

4) Dialog是一个对话框

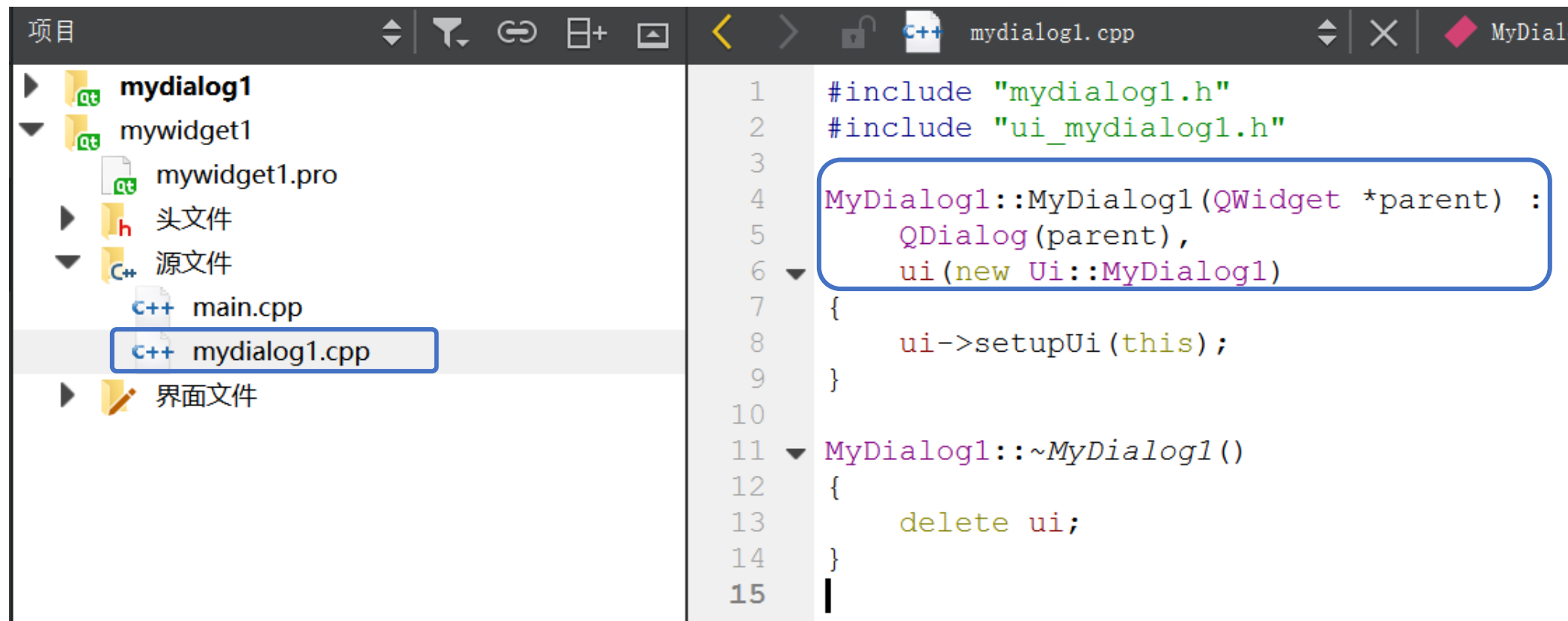


示例1 创建窗口

- **Main.cpp**
- 在这个例子中, 把窗口定义、显示的代码全部写到了 main.cpp 中
- 更常用的是定义一个自己的类, 如MyWidget, 然后在构造函数和类函数中去自定义该窗口的各种属性



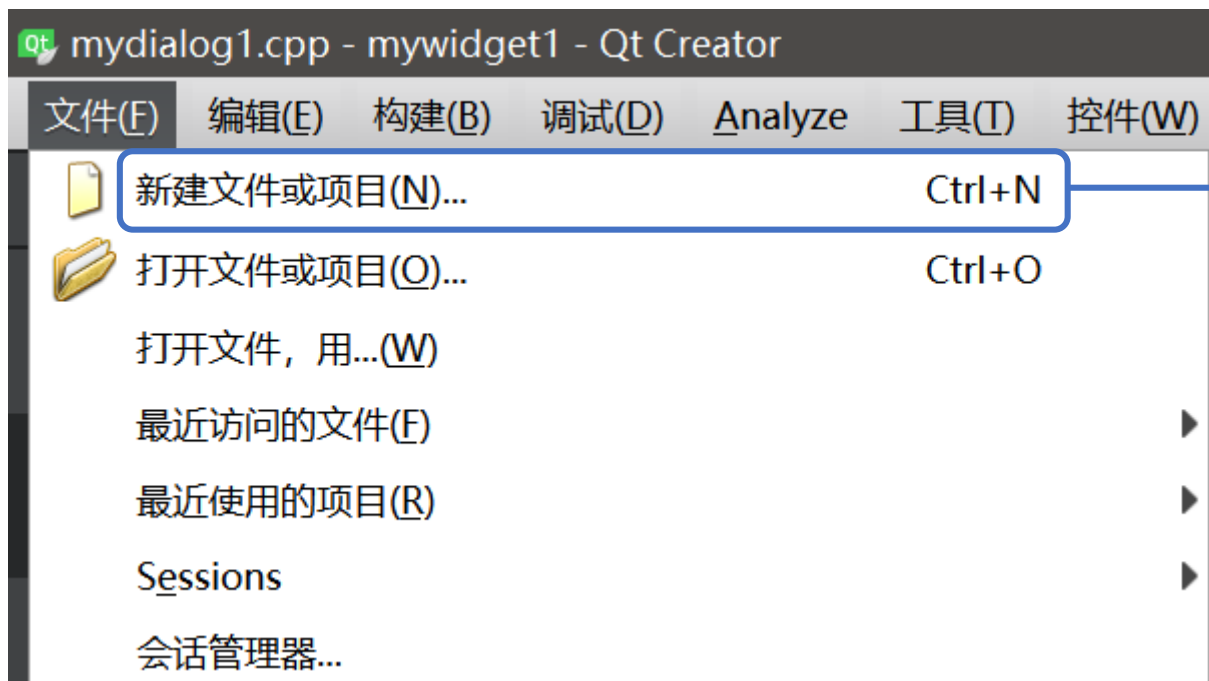
示例1 创建窗口



如上我们可以定义一个Mydialog1类, 创建使用界面,
可以直接用界面文件 (可以更方便的设计ui)

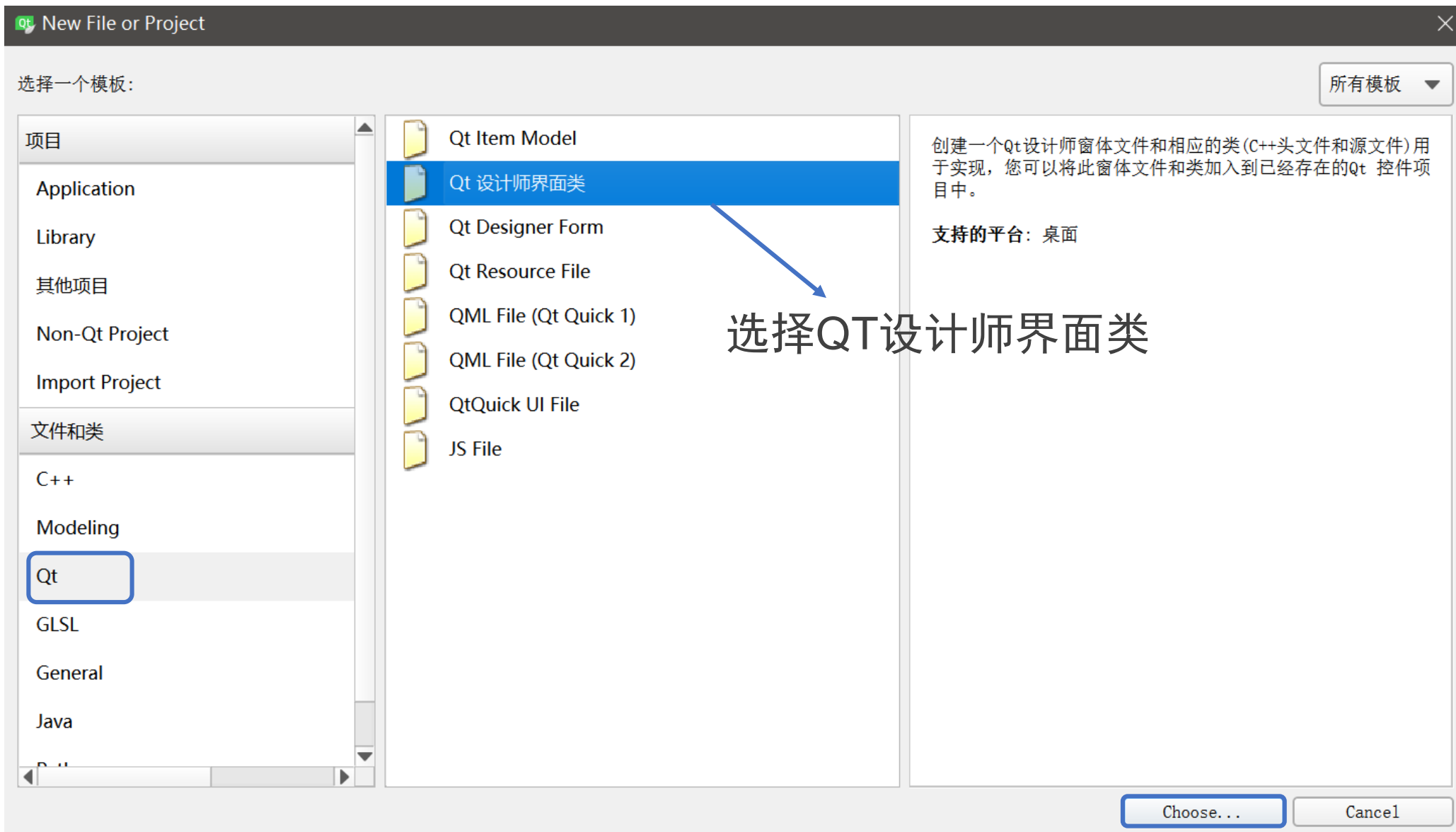
那么如何建立使用界面?

示例1 创建窗口



选择新建

示例1 创建窗口



示例1 创建窗口

Qt 设计器界面类

Form Template

Class Details

汇总

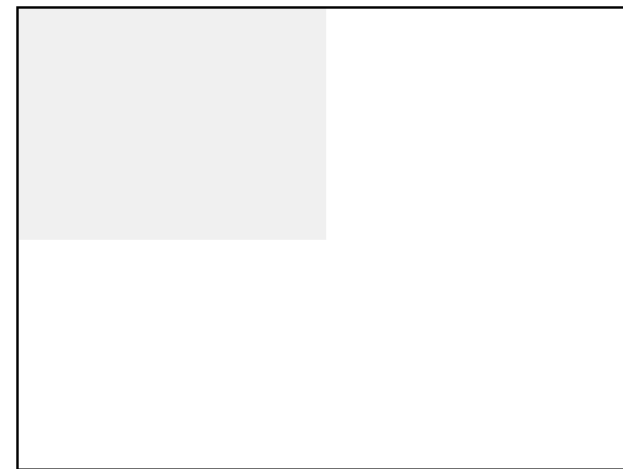
选择界面模板

templates\forms

- Dialog with Buttons Bottom
- Dialog with Buttons Right
- Dialog without Buttons
- Main Window
- Widget

窗口部件

根据需求选择即可, 一般选Dialog就行



嵌入式设计

设备:

无

屏幕大小:

默认大小

下一步(N)

取消

示例1 创建窗口

← Qt 设计器界面类

Form Template

➡ Class Details

汇总

选择类名

类

为定义的窗口取一个类名, 例如MyDialog1

类名 (C):

MyDialog1

头文件 (H):

mydialog1.h

源文件 (S):

mydialog1.cpp

界面文件 (F):

mydialog1.ui

路径 (P):

C:\Users\86157\Desktop\Code\qt_source_03\03\3-1\mywidget1

浏览...

下一步 (N)

取消

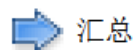
示例1 创建窗口

← Qt 设计器界面类

项目管理

Form Template

Class Details



汇总

添加到项目 (P):

mywidget1.pro

添加到版本控制系统 (V):

<None>

Configure...

要添加的文件

C:\Users\86157\Desktop\Code\qt_source_03\03\3-1\mywidget1:

mydialog1.cpp

mydialog1.h

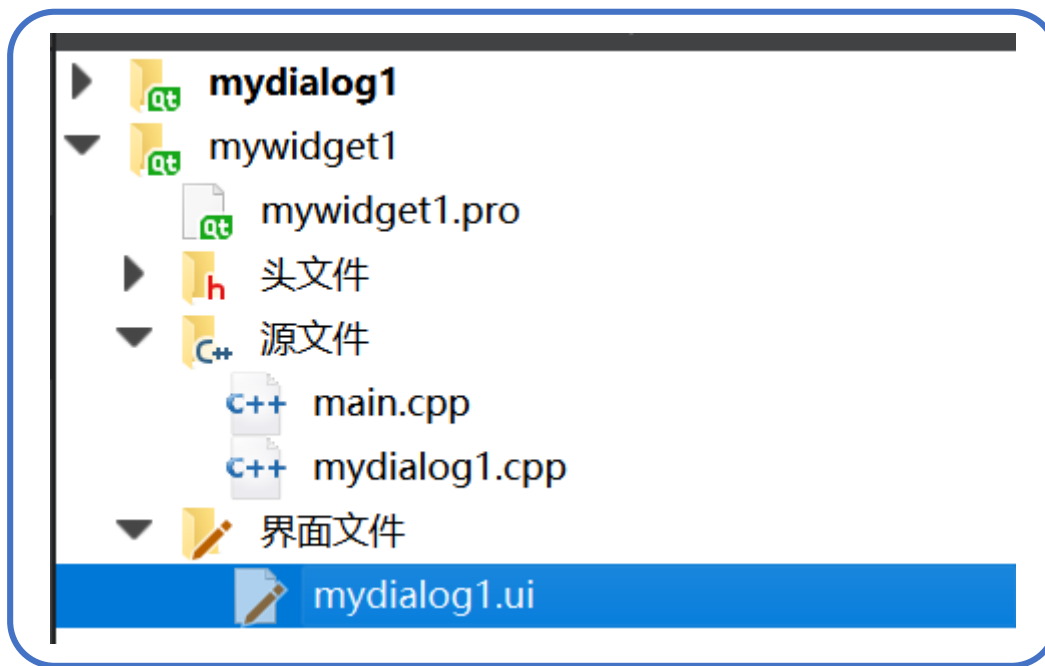
mydialog1.ui

默认设置就好, 完成创建

完成 (F)

取消

示例1 创建窗口



可以看到成功创建了

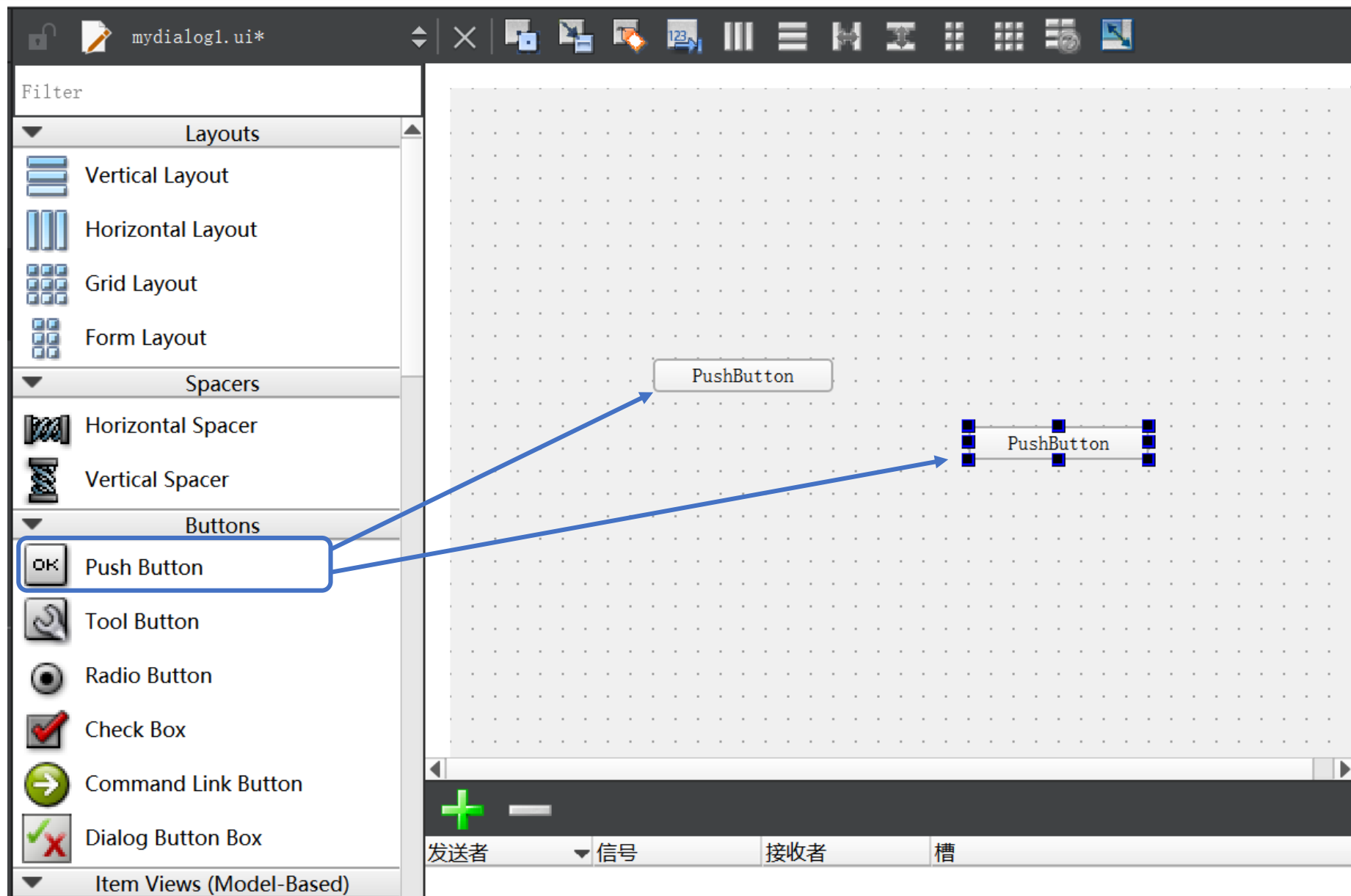
mydialog1窗口类

双击界面文件夹下的.ui文件

可以设计ui

示例1 创建窗口

- ui界面如下：
可以鼠标选择一些控件然后设计窗口
设计完毕保存即可



示例1 创建窗口

在main函数里就
可以直接用定义好的MyDialog1类了

```
1  #include "mywidget.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MyDialog1 w;
8
9      ...
10
11     return a.exec();
12 }
13
```

示例2 大小位置修改

- 创建窗口完毕, 演示如何修改窗口大小、位置
- qDebug的介绍

示例2 大小位置修改

```
< > DEV main.cpp [No Symbols]
1  #include <QApplication>
2  #include <QWidget>
3  #include <QDebug>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8
9      QWidget widget;
10     widget.resize(400, 300); // 设置窗口大小, 此大小不含边框
11     widget.move(200, 100);   // 设置窗口位置, 此位置包括边框
12     widget.show();
13     int x = widget.x();      // 包括边框
14     qDebug("x: %d", x);     // 输出x的值
15     int y = widget.y();      // 包括边框
16     qDebug("y: %d", y);
17     QRect geometry = widget.geometry(); // 用户区, 不含边框
18     QRect frame = widget.frameGeometry(); // 整个窗口, 包含边框
19     qDebug() << "geometry: " << geometry << "frame: " << frame;
20 }
```

这里定义一个
QWidget类型对
象来设置窗口
的大小和位置



示例2 qDebug的使用

- 要 #include <QDebug>
- 利用QDebug在控制台console打印调试信息，类似C语言的printf和C++的cout的结合
- 注意：QDebug可输出的类型也有QString等QT中独有的类，也可以通过重载运算符打印自己定义的class

常用来程序的调试

```
mywidget2/main.cpp* [No Symbols]
1  #include <QApplication>
2  #include <QWidget>
3  #include <QDebug>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8
9      QWidget widget;
10     widget.resize(400, 300);           // 设置窗口大小, 此大小不含边框
11     widget.move(200, 100);            // 设置窗口位置, 此位置包括边框
12     widget.show();
13     int x = widget.x();                // 包括边框
14     qDebug("x: %d", x);               // 输出x的值
15     int y = widget.y();                // 包括边框
16     qDebug("y: %d", y);
17     QRect geometry = widget.geometry(); // 用户区, 不含边框
18     QRect frame = widget.frameGeometry(); // 整个窗口, 包含边框
19     qDebug() << "geometry: " << geometry << "frame: " << frame;
20
21     qDebug() << "width:" << widget.width() << endl; // 用户区, 不含边框
22     qDebug() << "height:" << widget.height() << endl; // 用户区, 不含边框
23     qDebug() << "pos:" << widget.pos() << endl; // 整个窗口, 包含边框
24
25     widget.resize(500, 600); // 重设宽和高. 用户区, 不含边框
26
27     return a.exec();
28 }
```

示例2 qDebug的使用

- 在控制台中打印出了
要求的调试信息

```
应用程序输出  
mywidget1 X mywidget2 X  
Starting C:\DISK0\qtthing\booksample\03\3-2\build-mywidget2-Des  
\mywidget2.exe...  
x: 200  
y: 100  
geometry: QRect(211,145 400x300) frame: QRect(200,100 422x356  
width: 400  
height: 300  
pos: QPoint(200,100) |
```

```
mywidget2/main.cpp* <No Symbols>  
1 #include <QApplication>  
2 #include <QWidget>  
3 #include <QDebug>  
4  
5 int main(int argc, char *argv[])  
6 {  
7     QApplication a(argc, argv);  
8  
9     QWidget widget;  
10    widget.resize(400, 300); // 设置窗口大小, 此大小不含边框  
11    widget.move(200, 100); // 设置窗口位置, 此位置包括边框  
12    widget.show();  
13    int x = widget.x(); // 包括边框  
14    qDebug("x: %d", x); // 输出x的值  
15    int y = widget.y(); // 包括边框  
16    qDebug("y: %d", y);  
17    QRect geometry = widget.geometry(); // 用户区, 不含边框  
18    QRect frame = widget.frameGeometry(); // 整个窗口, 包含边框  
19    qDebug() << "geometry: " << geometry << "frame: " << frame;  
20  
21    qDebug() << "width:" << widget.width() << endl; // 用户区, 不含边框  
22    qDebug() << "height:" << widget.height() << endl; // 用户区, 不含边框  
23    qDebug() << "pos:" << widget.pos() << endl; // 整个窗口, 包含边框  
24  
25    widget.resize(500, 600); // 重设宽和高. 用户区, 不含边框  
26  
27    return a.exec();  
28 }  
29
```


示例3 模态与非模态对话框

- 对话框的模态与非模态设置
 - **模态对话框**：阻断了用户输入，当一个模态对话框打开时，用户只能与该对话框交互，而其他界面对象收不到输入信息
例如：关机时弹出的对话框，会禁止用户对其他窗口进行操作，必须取消这个对话框才能对其他对话框进行操作
 - **非模态对话框**：类似普通的windows窗口，在非模态对话框打开时，用户可同时打开其他窗口，操作完毕后，又可用鼠标或者其他方式激活该窗口
例如：操作系统中不同窗口的切换

示例3 模态与非模态对话框

The screenshot shows the Qt Creator IDE with the following components:

- Project Explorer:** Shows a project named 'mydialog1' with files 'mydialog1.pro', 'mywidget.h', 'main.cpp', 'mywidget.cpp', and 'mywidget.ui'.
- Editor:** Displays the code in 'main.cpp':

```
1 #include "mywidget.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     MyWidget w; //init构造函数显示dialog
8     w.show();  //显示 MyWidget窗口
9 }
```
- Run and Debug Console:** Shows two windows: 'mydialog1' and 'MyWidget'.

Annotations in the image:

- A blue box highlights the code lines: `QApplication a(argc, argv);` and `MyWidget w; //init构造函数显示dialog`.
- A blue arrow points from the `MyWidget w;` line to the 'mydialog1' window.
- A blue arrow points from the `w.show();` line to the 'MyWidget' window.

两行语句分别产生了两个窗口

这里第一行语句之所以会产生窗口是因为MyWidget类的构造函数中定义了创建对话框

示例3 模态与非模态对话框

The screenshot displays the Qt Creator IDE with the `mywidget.cpp` file open. The code defines the `MyWidget::MyWidget` constructor, which creates a `QDialog` object and sets its modality. A blue box highlights the dialog creation and modality settings, with arrows pointing to the resulting windows.

```
3 #include <QDialog>
4
5 MyWidget::MyWidget(QWidget *parent) :
6     QWidget(parent),
7     ui(new Ui::MyWidget)
8 {
9     ui->setupUi(this); //必须照抄，建里窗口和界面的关系
10    QDialog *dialog = new QDialog(this);
11
12    dialog->setModal(true); //模态对话框，即关闭前不能和其他窗口交互的对话框
13    //dialog->setModal(false); //非模态对话框，即关闭前可以和其他窗口交互的对话框
14    dialog->show(); //导致MyWidget窗口显示后，立即弹出对话框。show函数立即返回，
15                    //MyWidget窗口才可以显示
16
17    /*
18
19
20
21
22
23
```

Below the code, two windows are shown: `mydialog1` (a modal dialog) and `MyWidget` (the main widget). The `MyWidget` window is titled "MyWidget" and contains a text area.

MyWidget类的构造函数中定义了创建对话框

示例3 模态与非模态对话框



```
< > mywidget.cpp MyWidget::MyWidget(QWidget *) #
3  #include <QDialog>
4
5  MyWidget::MyWidget(QWidget *parent) :
6      QWidget(parent),
7      ui(new Ui::MyWidget)
8  {
9      ui->setupUi(this); //必须照抄，建里窗口和界面的关系
10     QDialog *dialog = new QDialog(this); 通过setModal设置模态/非模态
11
12     dialog->setModal(true); //模态对话框，即关闭前不能和其他窗口交互的对话框
13     //dialog->setModal(false); //非模态对话框，即关闭前可以和其他窗口交互的对话框
14     dialog->show(); //导致MyWidget窗口显示后，立即弹出对话框。show函数立即返回，
15                     //MyWidget窗口才可以显示
16
17     /*
18     setWindowModality() 函数可以设置对话框要阻塞的窗口类型, 可以为:
19
20     Qt::NonModal      不阻塞任何窗口，即非模态
21     Qt::WindowModal   阻塞所有祖先窗口及其子窗口
22     Qt::ApplicationModal 阻塞整个应用程序所有窗口
23     */
```

- (True) 模态对话框：弹出以后原窗口不能交互
- (False) 非模态对话框：弹出以后原窗口还能交互

示例3 模态与非模态对话框



```
< > mywidget.cpp MyWidget::MyWidget(QWidget *) #
3  #include <QDialog>
4
5  MyWidget::MyWidget(QWidget *parent) :
6      QWidget(parent),
7      ui(new Ui::MyWidget)
8  {
9      ui->setupUi(this); //必须照抄，建里窗口和界面的关系
10     QDialog *dialog = new QDialog(this); 通过setModal设置模态/非模态
11
12     dialog->setModal(true); //模态对话框，即关闭前不能和其他窗口交互的对话框
13     //dialog->setModal(false); //非模态对话框，即关闭前可以和其他窗口交互的对话框
14     dialog->show(); //导致MyWidget窗口显示后，立即弹出对话框。show函数立即返回，
15                     //MyWidget窗口才可以显示
16
17     /*
18     setWindowModality() 函数可以设置对话框要阻塞的窗口类型, 可以为:
19
20     Qt::NonModal      不阻塞任何窗口，即非模态
21     Qt::WindowModal   阻塞所有祖先窗口及其子窗口
22     Qt::ApplicationModal 阻塞整个应用程序所有窗口
23     */
24
25     • (True) 模态对话框：弹出以后原窗口不能交互
26     • (False) 非模态对话框：弹出以后原窗口还能交互
```

这里由于设置了dialog为模态对话框, 则关闭dialog之前不能与widget交互!!!