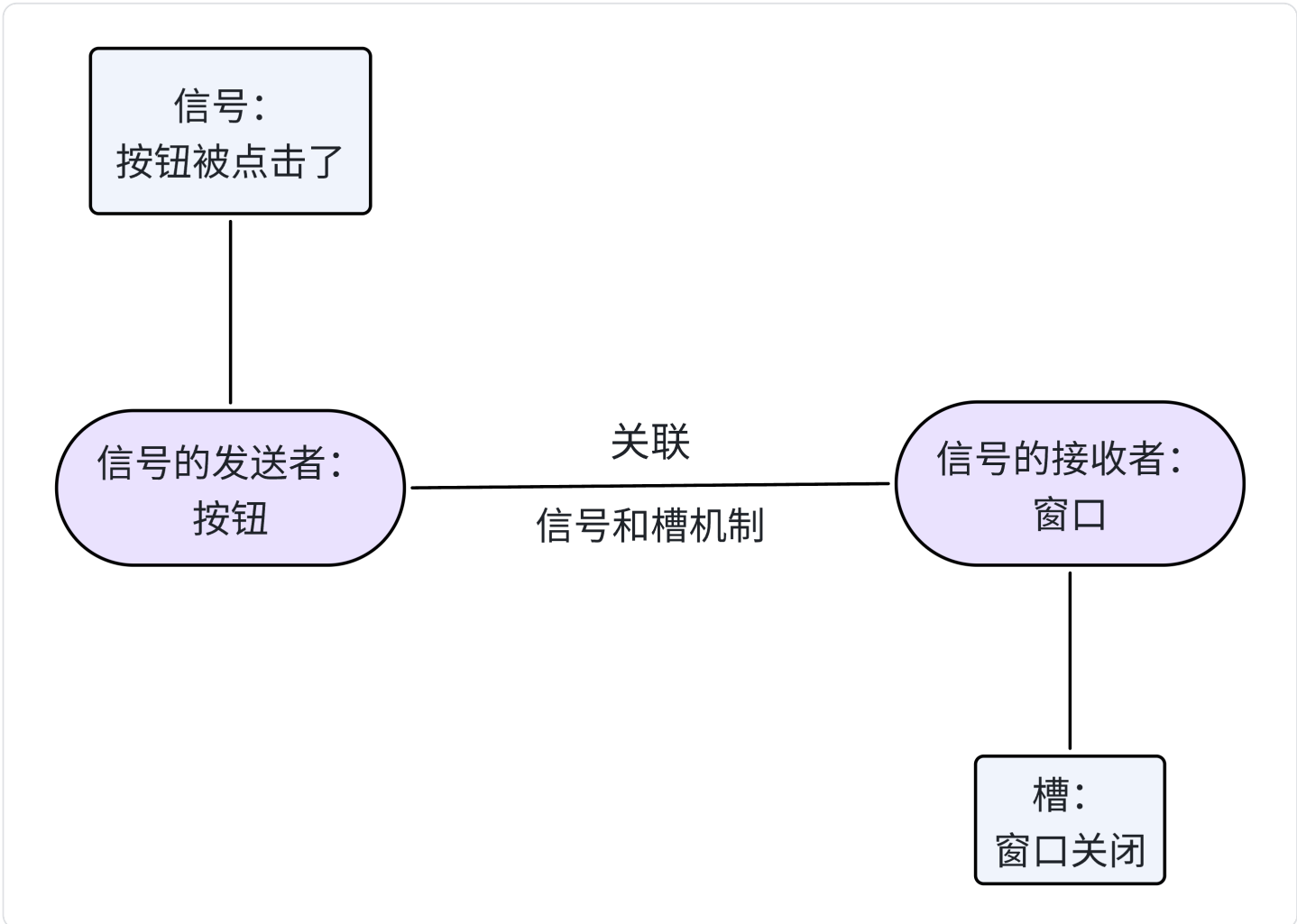
第二章 信号与槽

1. 信号和槽概述

在 Qt 中，用户和控件的每次交互过程称为一个事件。比如 "用户点击按钮" 是一个事件，"用户关闭窗口" 也是一个事件。**每个事件都会发出一个信号**，例如用户点击按钮会发出 "按钮被点击" 的信号，用户关闭窗口会发出 "窗口被关闭" 的信号。

Qt 中的所有控件都具有接收信号的能力，一个控件还可以接收多个不同的信号。对于接收到的每个信号，控件都会做出相应的响应动作。例如，按钮所在的窗口接收到 "按钮被点击" 的信号后，会做出 "关闭自己" 的响应动作；再比如输入框自己接收到 "输入框被点击" 的信号后，会做出 "显示闪烁的光标，等待用户输入数据" 的响应动作。**在 Qt 中，对信号做出的响应动作就称之为槽。**

信号和槽是 Qt 特有的消息传输机制，它能将相互独立的控件关联起来。比如，"按钮" 和 "窗口" 本身是两个独立的控件，点击 "按钮" 并不会对 "窗口" 造成任何影响。通过信号和槽机制，可以将 "按钮" 和 "窗口" 关联起来，实现 "点击按钮会使窗口关闭" 的效果。



信号的本质

信号是由于用户对窗口或控件进行了某些操作，导致窗口或控件产生了某个特定事件，这时 Qt 对应的窗口类会发出某个信号，以此对用户的操作做出反应。因此，**信号的本质就是事件**。如：

- 按钮单击、双击
- 窗口刷新
- 鼠标移动、鼠标按下、鼠标释放
- 键盘输入

那么在 Qt 中信号是通过什么形式呈现给使用者的呢?

- 我们对哪个窗口进行操作, 哪个窗口就可以捕捉到这些被触发的事件。
- 对于使用者来说触发了一个事件我们就可以得到 Qt 框架给我们发出的某个特定信号。
- **信号的呈现形式就是函数**， 也就是说某个事件产生了， Qt 框架就会调用某个对应的信号函数， 通知使用者。

在 Qt 中信号的发出者是某个实例化的类对象。

槽的本质

槽（Slot）就是对信号响应的函数。槽就是一个函数，与一般的 C++ 函数是一样的，可以定义在类的任何位置（ public、protected 或 private ），可以具有任何参数，可以被重载，也可以被直接调用（但是不能有默认参数）。**槽函数与一般的函数不同的是：槽函数可以与一个信号关联，当信号被发射时，关联的槽函数被自动执行。**

说明

（1）信号和槽机制底层是通过函数间的相互调用实现的。**每个信号都可以用函数来表示，称为信号函数；每个槽也可以用函数表示，称为槽函数。**例如: "按钮被按下" 这个信号可以用 clicked() 函数表示，"窗口关闭" 这个槽可以用 close() 函数表示，假如使用信号和槽机制-

实现："点击按钮会关闭窗口" 的功能，其实就是 clicked() 函数调用 close() 函数的效果。

（2）信号函数和槽函数通常位于某个类中，和普通的成员函数相比，它们的特别之处在于：

- 信号函数用 **signals** 关键字修饰，槽函数用 **public slots、protected slots** 或者 **private slots** 修饰。**signals** 和 **slots** 是 Qt 在 C++ 的基础上扩展的关键字，专门用来指明信号函数和槽函数；
- **信号函数只需要声明，不需要定义（实现），而槽函数需要定义（实现）。**

> 💡 信号函数的定义是 Qt 自动在编译程序之前生成的. 编写 Qt 应用程序的程序猿无需关注.
>
> 这种自动生成代码的机制称为 **元编程 (Meta Programming)** . 这种操作在很多场景中都能见到.

2. 信号和槽的使用

2.1 连接信号和槽

在 Qt 中，QObject 类提供了一个静态成员函数 **connect()** ，该函数专门用来关联指定的信号函数和槽函数。

> 💡 **关于 QObject**
>
> QObject 是 Qt 内置的父类. Qt 中提供的很多类都是直接或者间接继承自 QObject.
>
> 这一点的设定和 Java 是非常相似的.

connect() 函数原型：

```
connect (const QObject *sender,
         const char *  signal   ,
         const QObject *  receiver   ,
         const char *  method   ,
         Qt::ConnectionType    type    = Qt::AutoConnection )
```

参数说明：

- sender：信号的发送者；
- signal：发送的信号（信号函数）；
- receiver：信号的接收者；
- method：接收信号的槽函数；
- type： 用于指定关联方式，默认的关联方式为 Qt::AutoConnection，通常不需要手动设定。

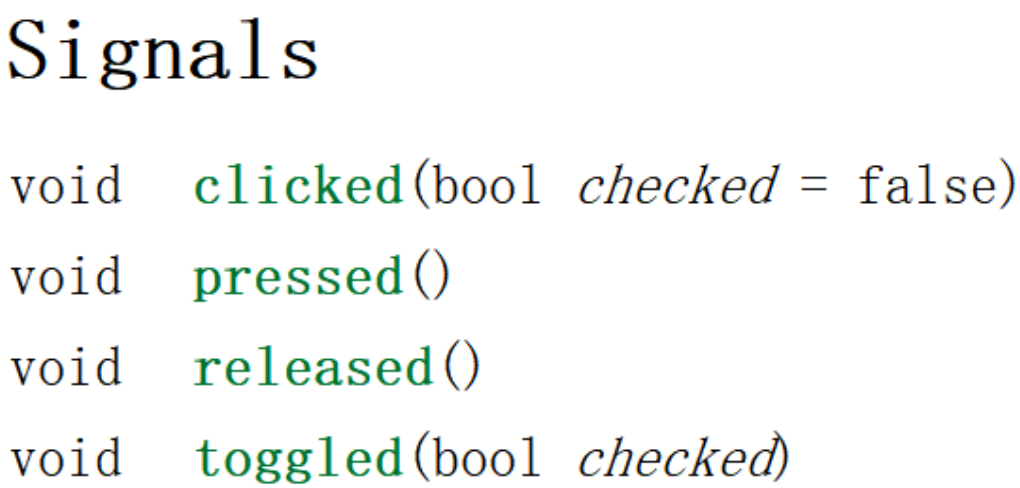
代码示例: 在窗口中设置一个按钮，当点击 "按钮" 时关闭 "窗口" .

2.2 查看内置信号和槽

系统自带的信号和槽通常是通过 **"Qt 帮助文档"** 来查询。

如上述示例，要查询 **"按钮"** 的信号，在帮助文档中输入：**QPushButton**，

- 首先可以在 **"Contents"** 中寻找关键字 **signals**，
- 如果没有找到, 继续去父类中查找. 因此我们去他的父类 **QAbstractButton** 中继续查找关键字 **signals**，



这里的 **clicked()** 就是要找的信号。槽函数的寻找方式和信号一样，只不过它的关键字是 **slot** 。

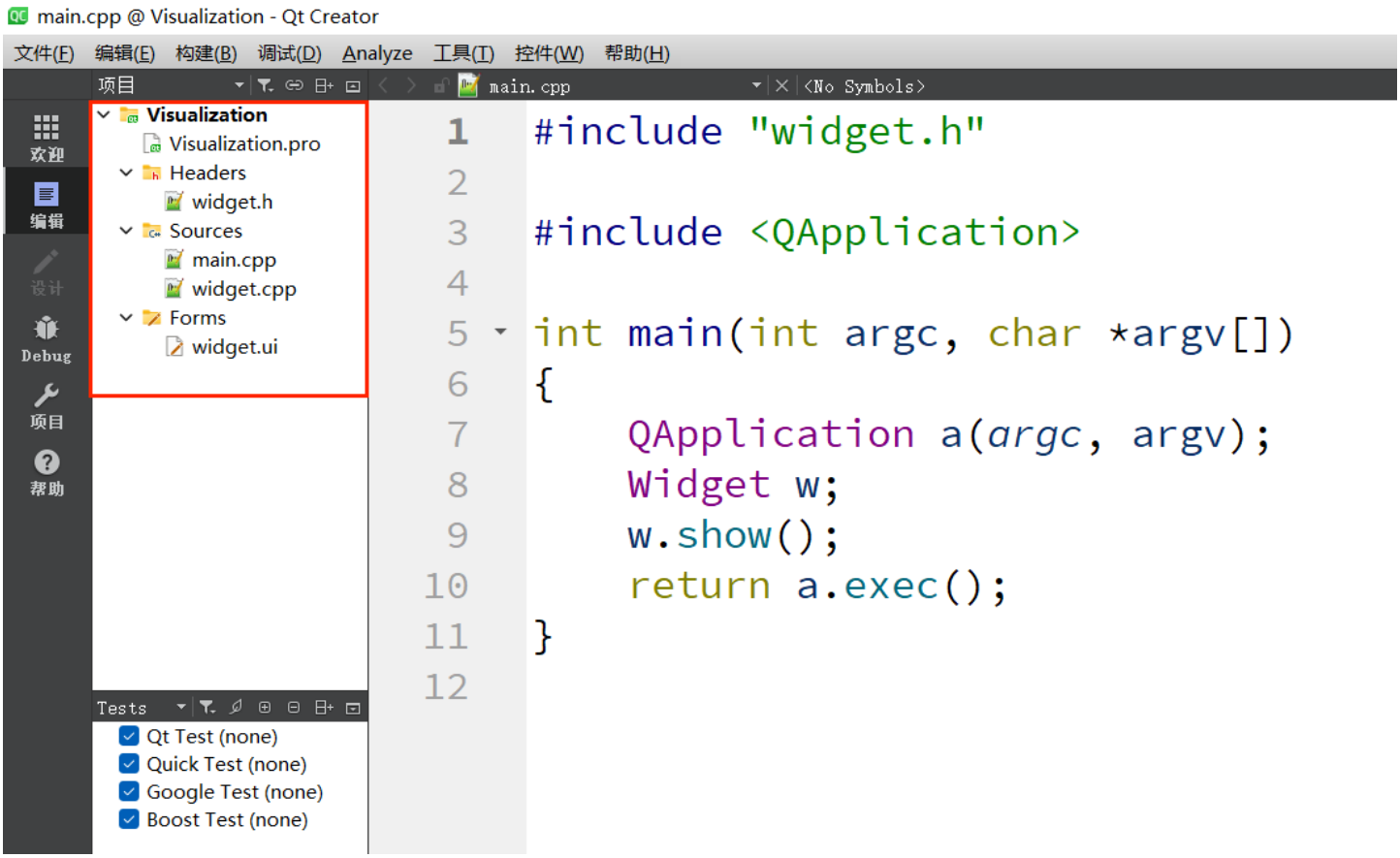
2.3 通过 Qt Creator 生成信号槽代码

Qt Creator 可以快速帮助我们生成信号槽相关的代码.

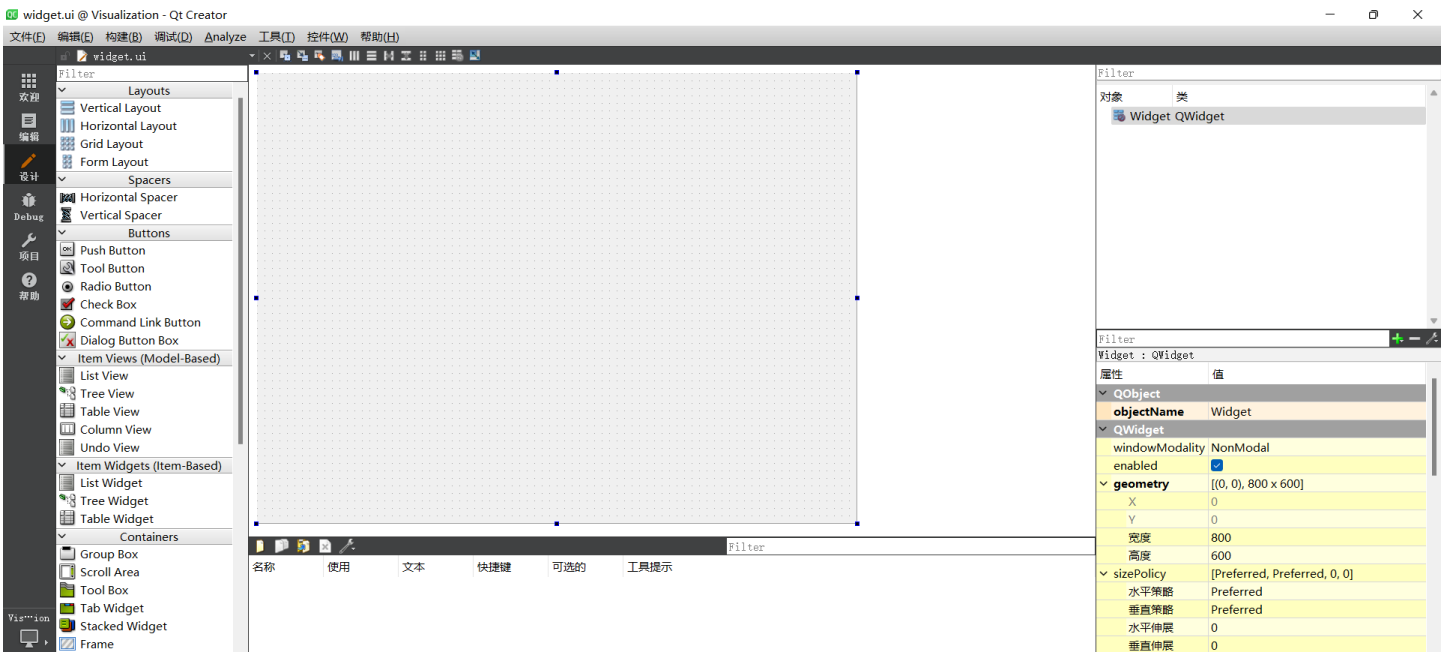
代码示例: 在窗口中设置一个按钮，当点击 "按钮" 时关闭 "窗口" .

1、新建项目，如下图为新建完成之后所包含的所有文件；

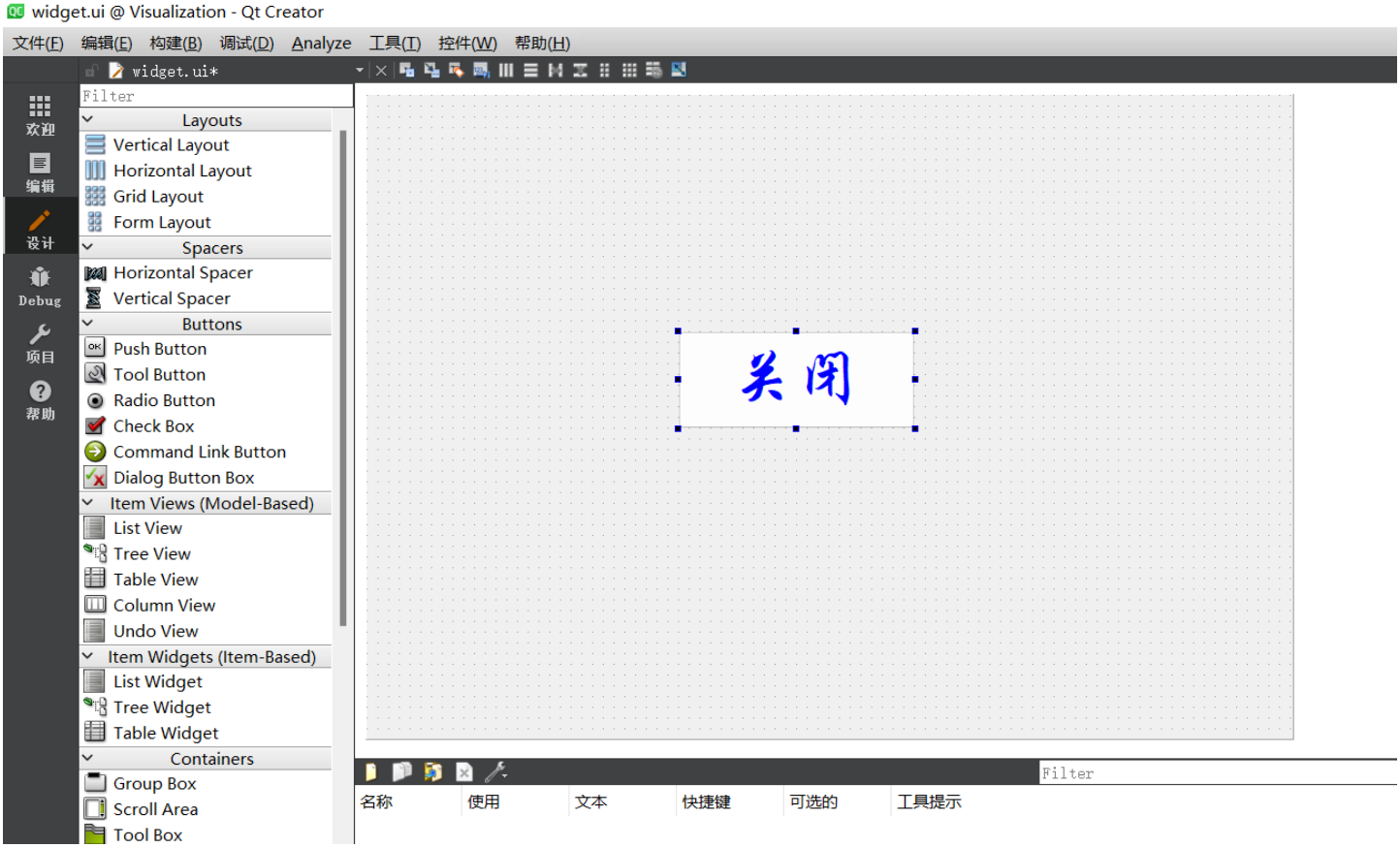
> 注意：创建时要生成 UI 设计文件；



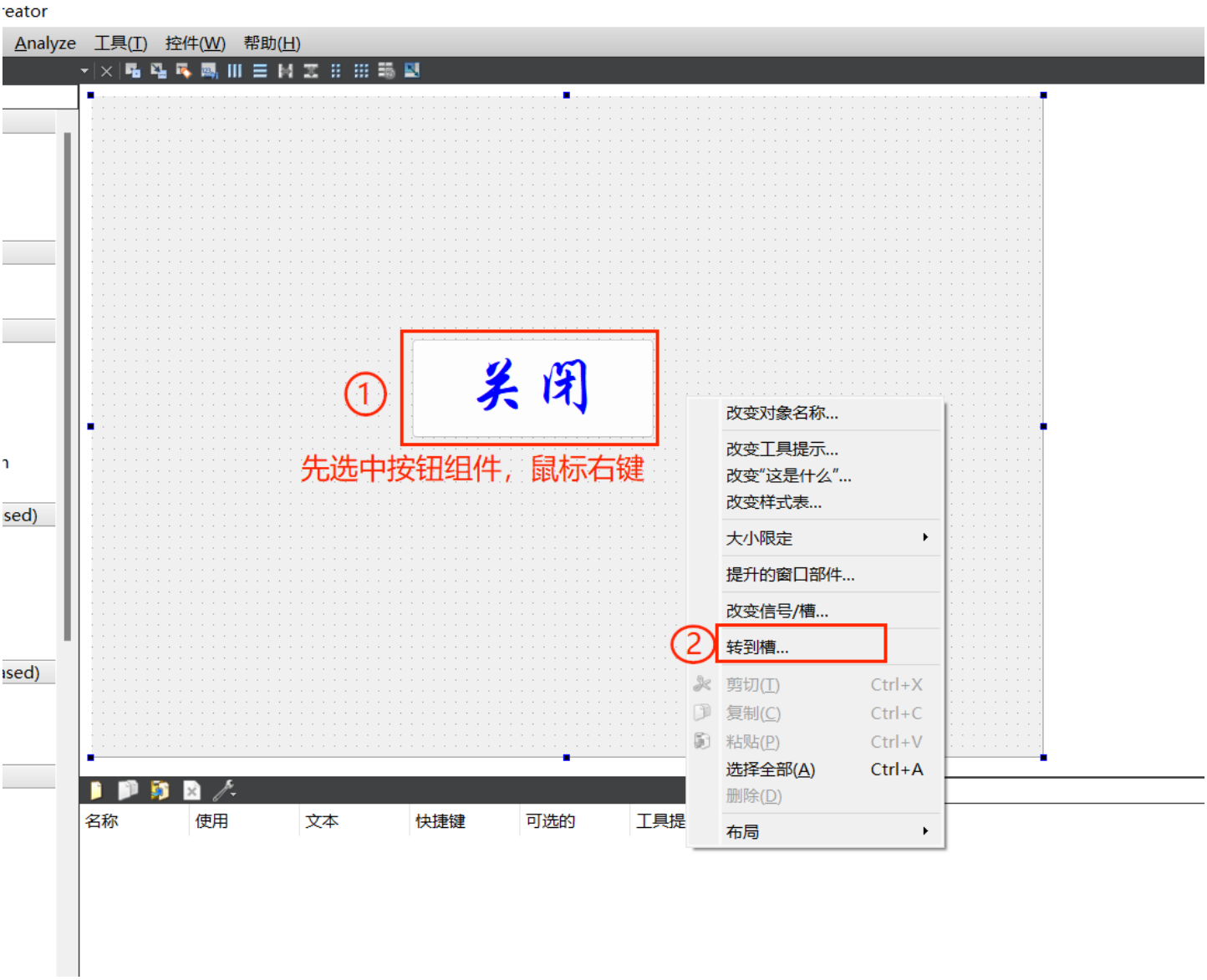
2、双击 widget.ui 文件，进入 UI 设计界面；



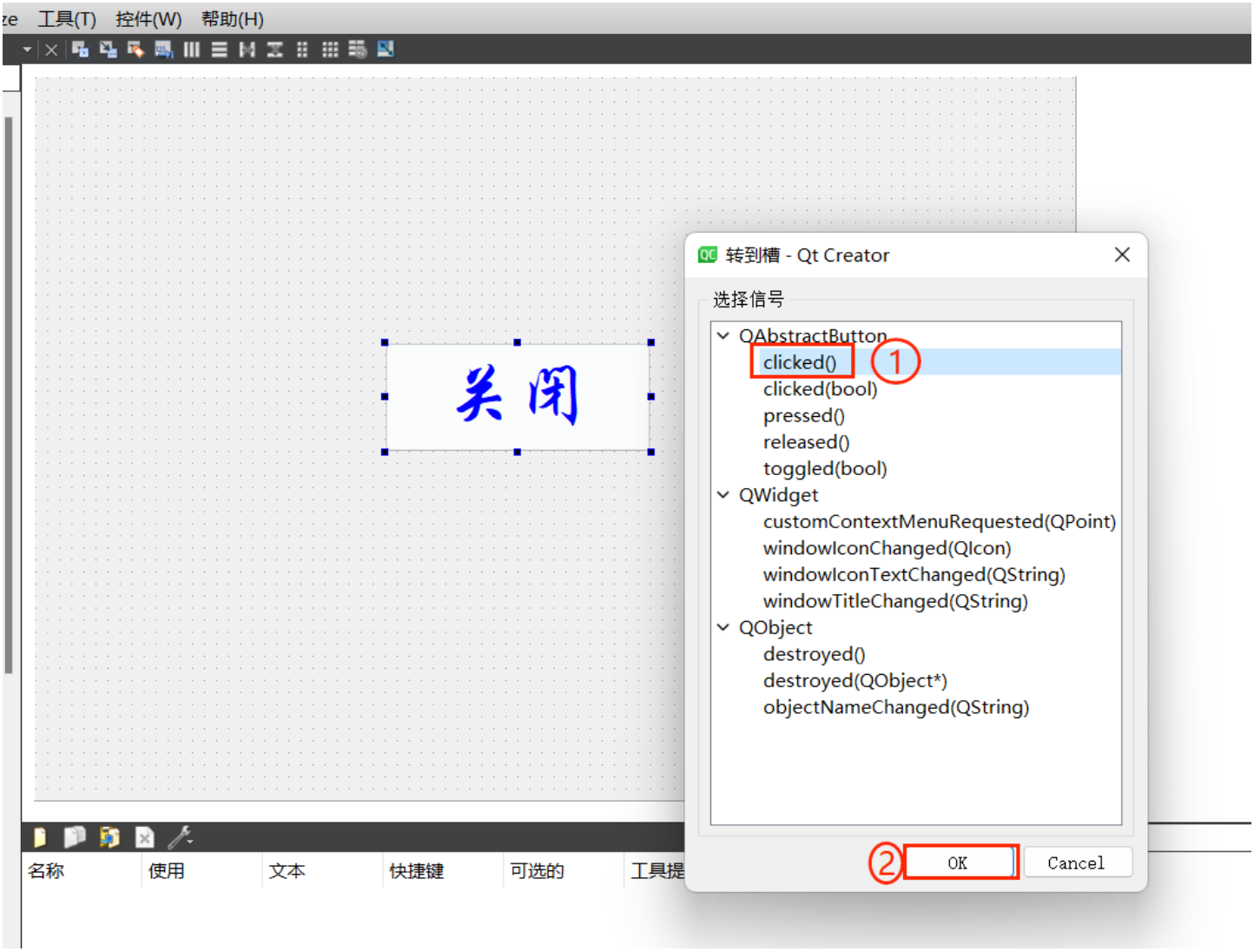
3、在 UI 设计窗口中拖入一个 "按钮" ，并且修改 "按钮" 的名称及字体大小等；


widget.ui @ Visualization - Qt Creator
文件(F) 编辑(E) 构建(B) 调试(D) Analyze 工具(T) 控件(W) 帮助(H)
widget.ui*
欢迎
编辑
设计
Debug
项目
帮助
Filter
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Dialog Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Undo View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Scroll Area
Tool Box
关闭
Filter
名称
使用
文本
快捷键
可选的
工具提示

4、可视化生成槽函数；



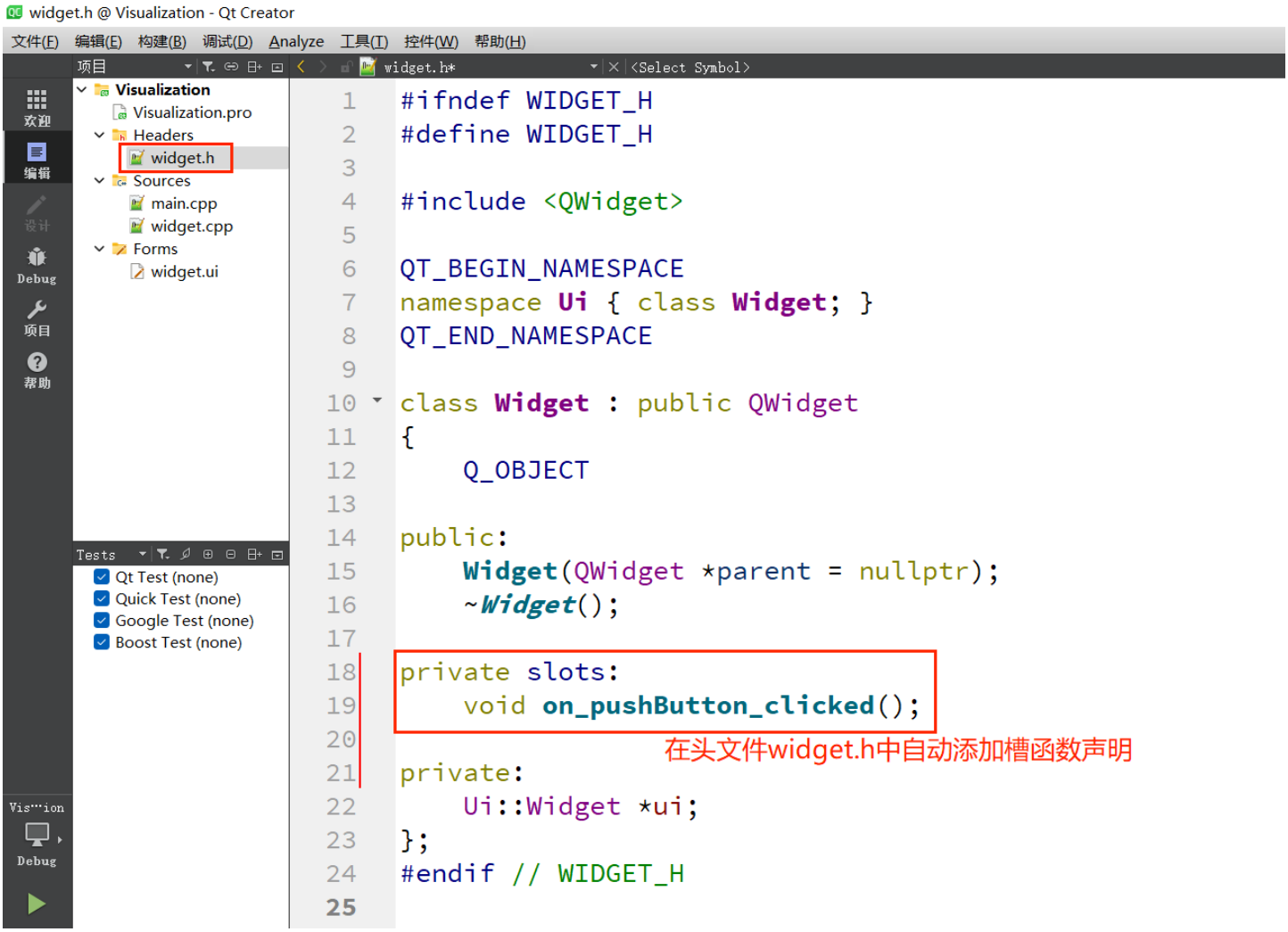
当单击 "转到槽..." 之后，出现如下界面：对于按钮来说，当点击时发送的信号是：clicked()，所以此处选择：clicked()



> 对于普通按钮来说, 使用 `clicked` 信号即可. `clicked(bool)` 没有意义的. 具有特殊状态的按钮(比如复选按钮)才会用到 `clicked(bool)` .

5、自动生成槽函数原型框架；

（1）在 "widget.h" 头文件中**自动添加槽函数的声明**；



说明：

自动生成槽函数的名称有一定的规则。槽函数的命名规则为：on_XXX_SSS，其中：

1、以 **" on "** 开头，中间使用下划线连接起来；

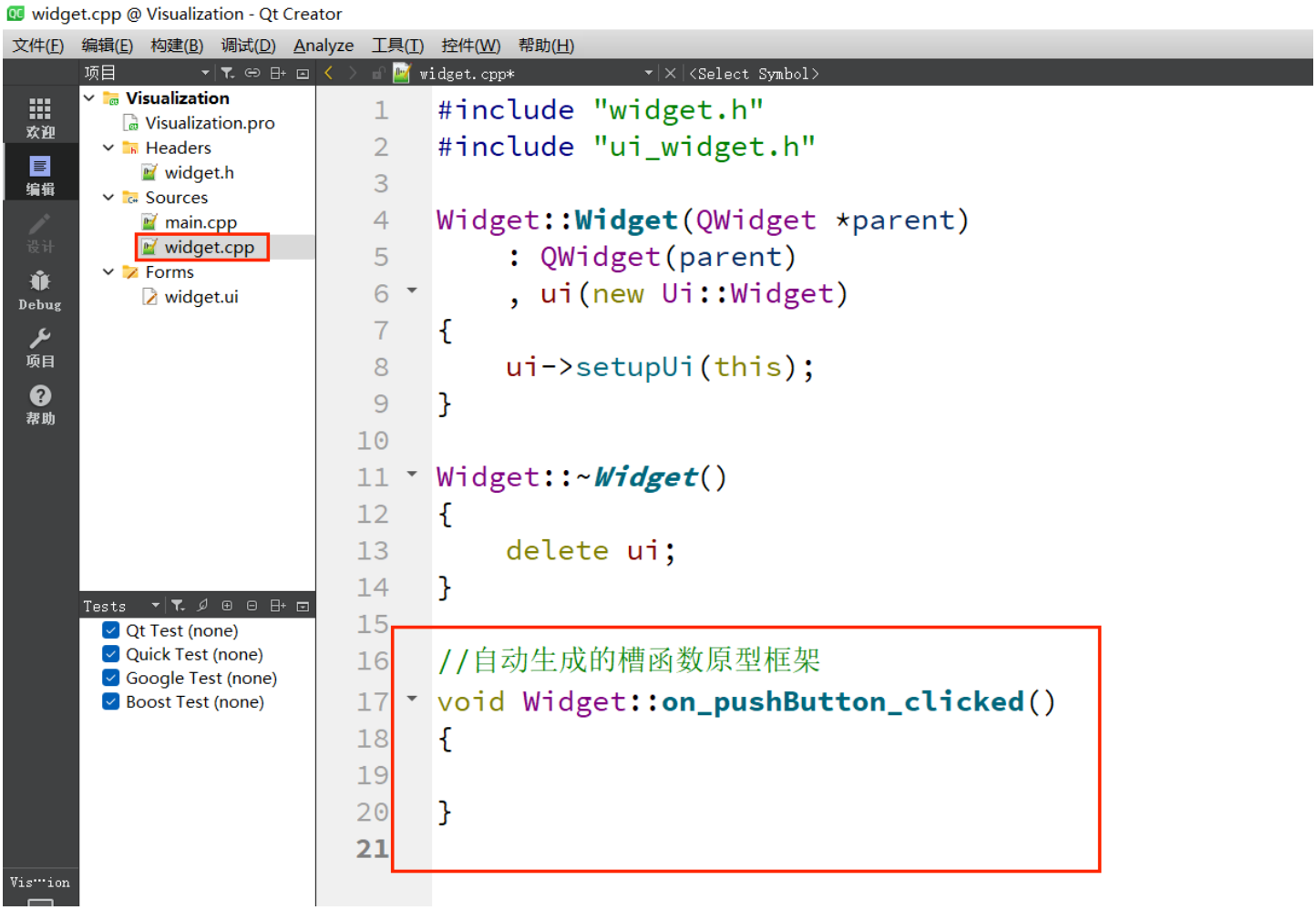
2、**" XXX "** 表示的是对象名(控件的 `objectName` 属性)。

3、**" SSS "** 表示的是对应的信号。

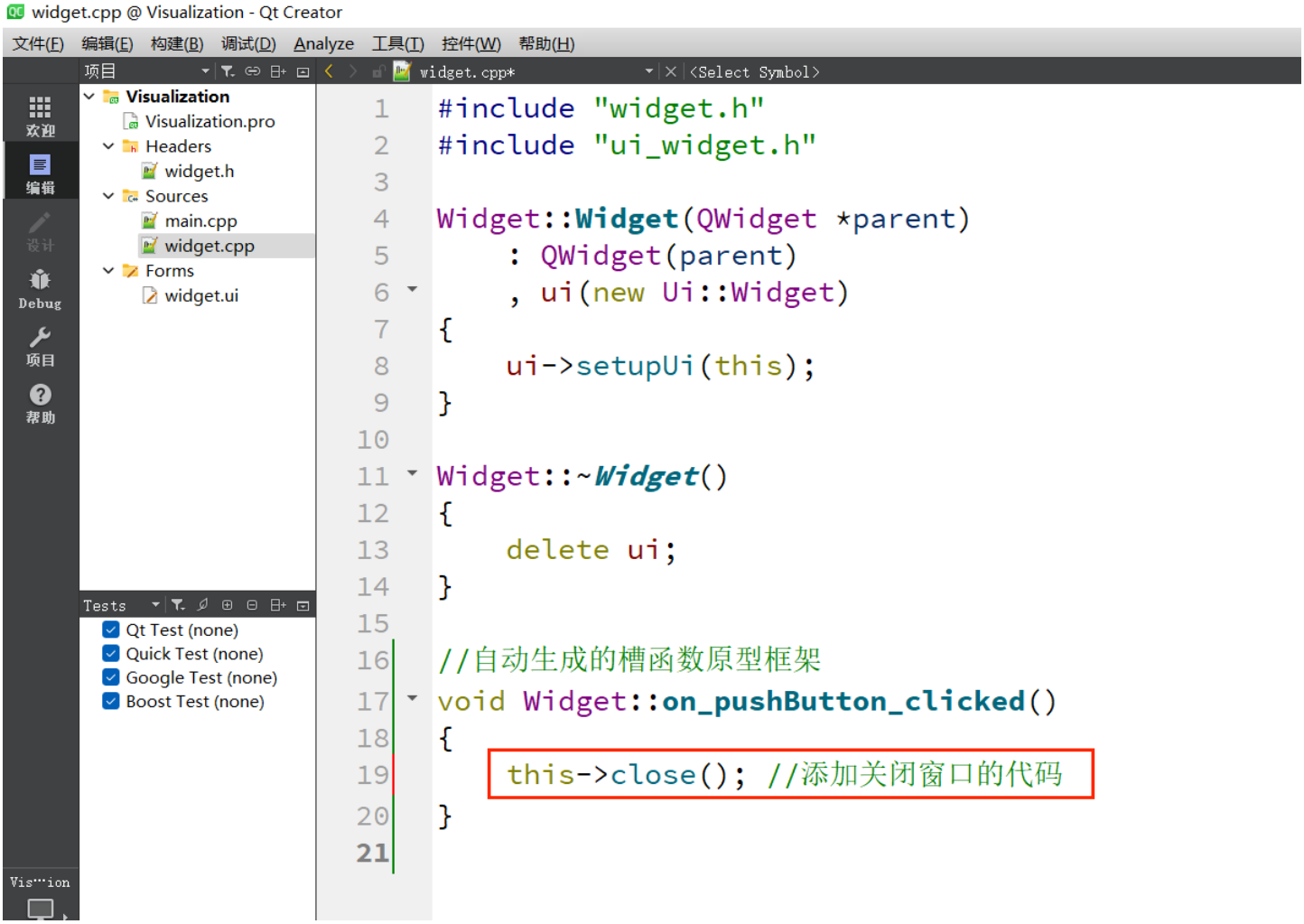
如：**" on_pushButton_clicked() "**，pushButton 代表的是对象名，clicked 是对应的信号。

> 按照这种命名风格定义的槽函数, 就会被 Qt 自动的和对应的信号进行连接.
>
> 但是咱们日常写代码的时候, 除非是 IDE 自动生成, 否则最好还是不要依赖命名规则, 而是显式使用 connect 更好.
>
> 一方面显式 connect 可以更清晰直观的描述信号和槽的连接关系.
>
> 另一方面也防止信号或者槽的名字拼写错误导致连接失效.
>
> (当然, 是配置大于约定, 还是约定大于配置, 哪种更好, 这样的话题业界尚存在争议. 此处我个人还是更建议优先考虑显式 connect)

（2）在 "widget.cpp" 中自动生成槽函数定义.



6、在槽函数函数定义中添加要实现的功能. 实现关闭窗口的效果.



3. 自定义信号和槽

3.1 基本语法

在 Qt 中，允许自定义信号的发送方以及接收方，即可以自定义信号函数和槽函数。但是对于自定义的信号函数和槽函数有一定的书写规范。

1、自定义信号函数书写规范

（1）自定义信号函数必须写到 "signals" 下；

（2）返回值为 void，只需要声明，不需要实现；

（3）可以有参数，也可以发生重载；

2、自定义槽函数书写规范

（1）早期的 Qt 版本要求槽函数必须写到 "public slots" 下，但是现在高级版本的 Qt 允许写到类的 "public" 作用域中或者全局下；

（2）返回值为 void，需要声明，也需要实现；

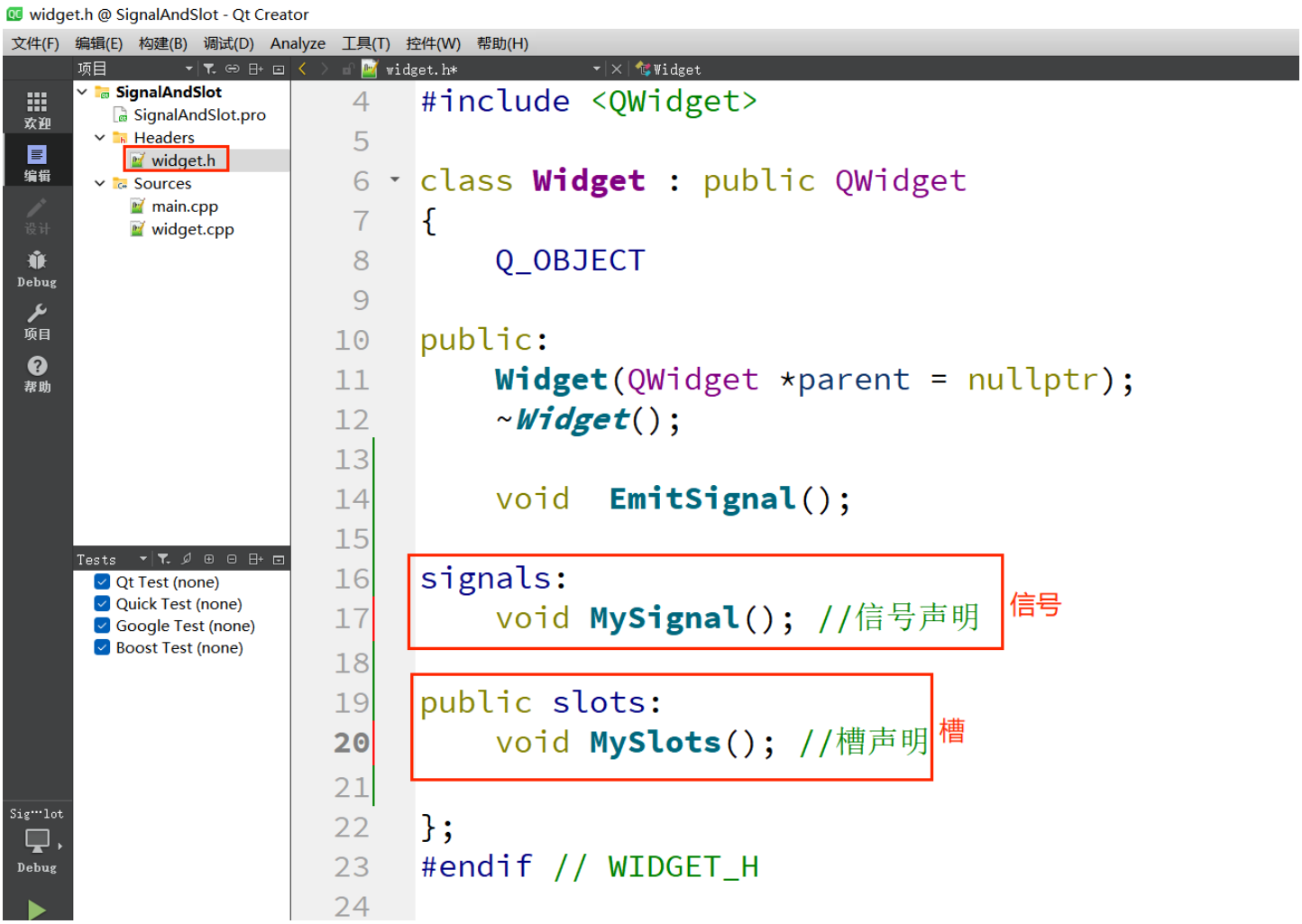
（3）可以有参数，可以发生重载；

3、发送信号

使用 "emit" 关键字发送信号 。"emit" 是一个空的宏。"emit" 其实是可选的，没有什么含义，只是为了提醒开发人员。

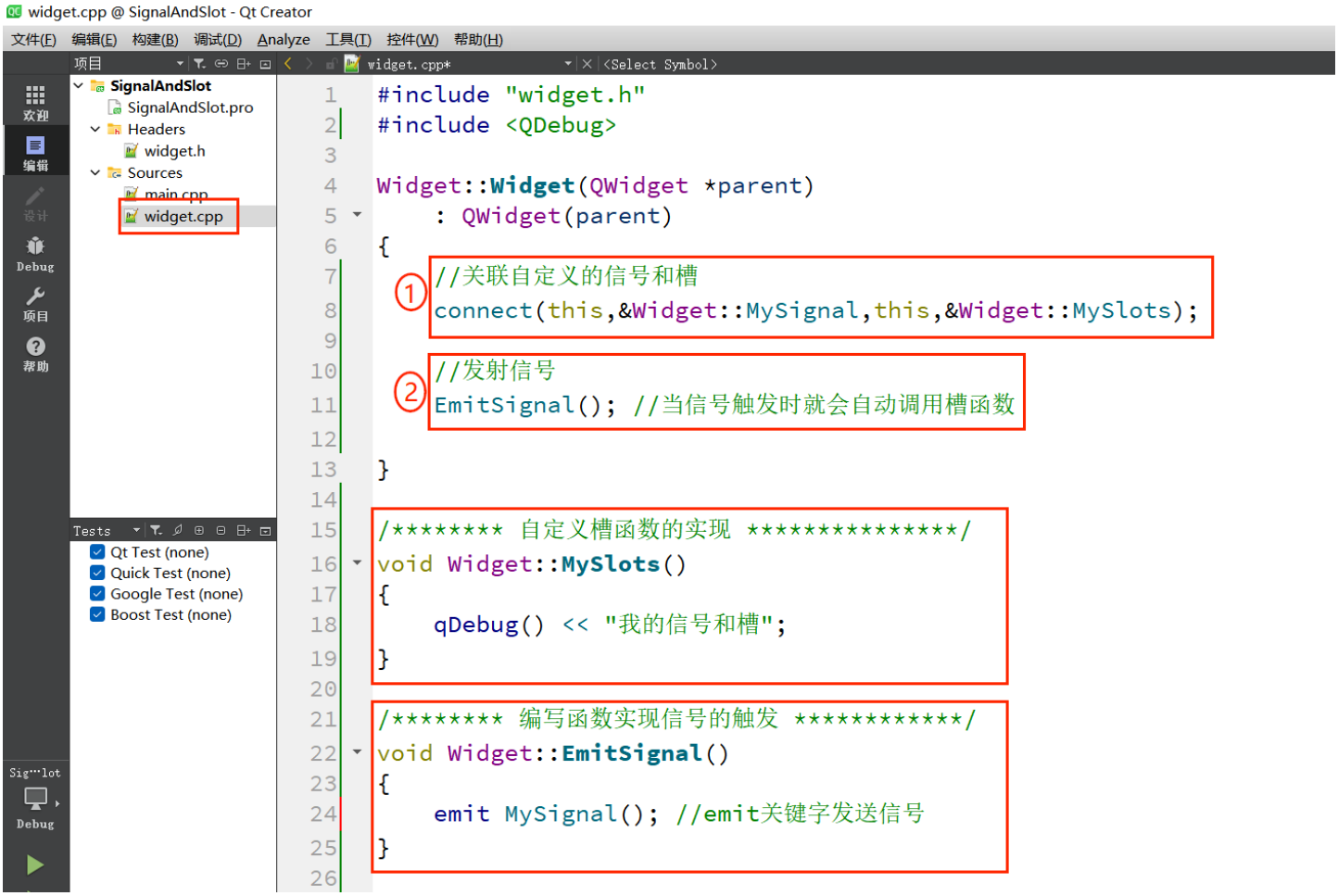
示例1：

1、在 widget.h 中声明自定义的信号和槽，如图所示；



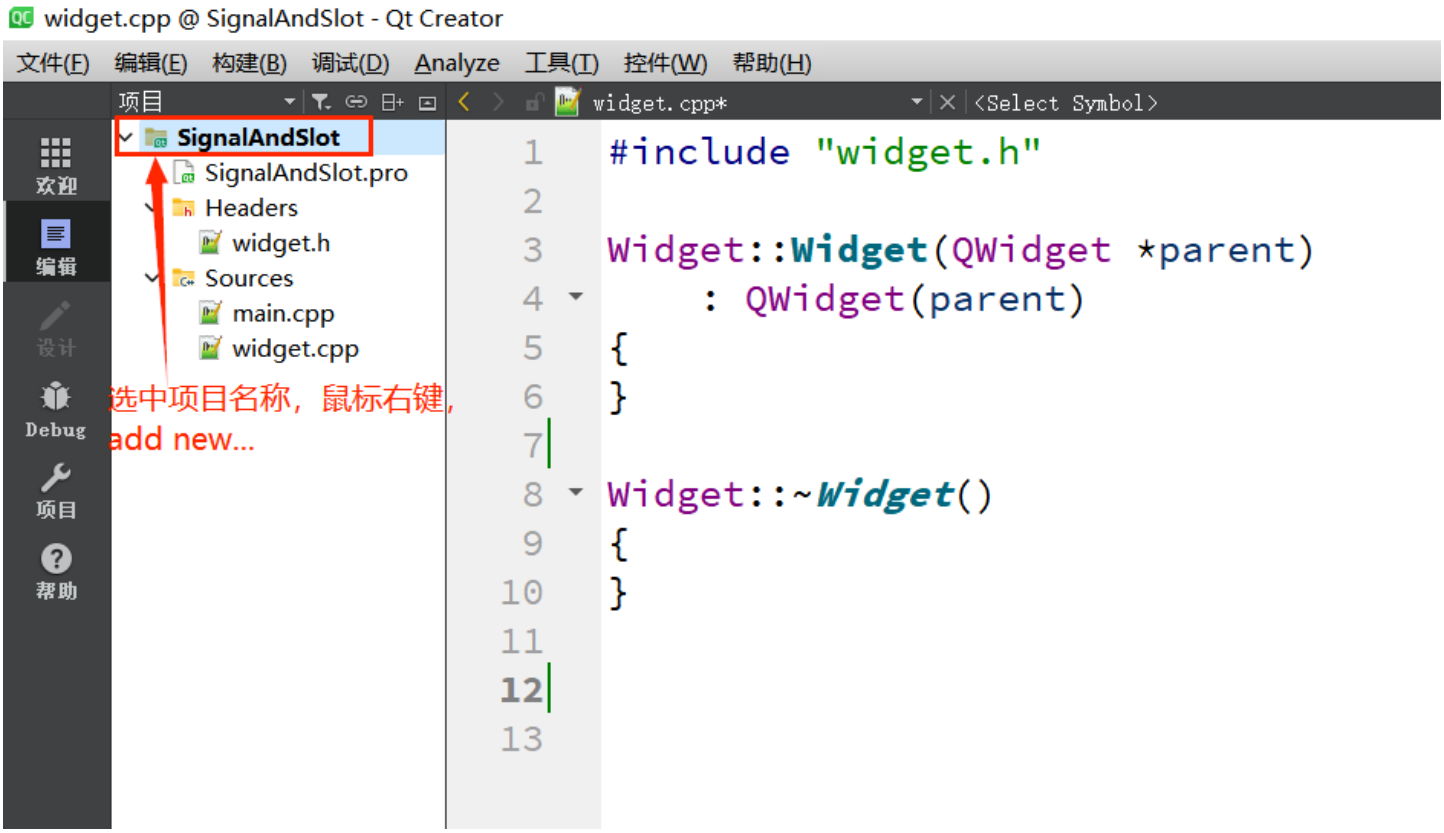
2、在 widget.cpp 中实现槽函数，并且关联信号和槽

> **注意：图中的 ① 和 ② 的顺序不能颠倒。**
>
> 原因是, 首先关联信号和槽，一旦检测到信号发射之后就会立马执行关联的槽函数。反之，若先发射信号，此时还没有关联槽函数，当信号发射之后槽函数不会响应.

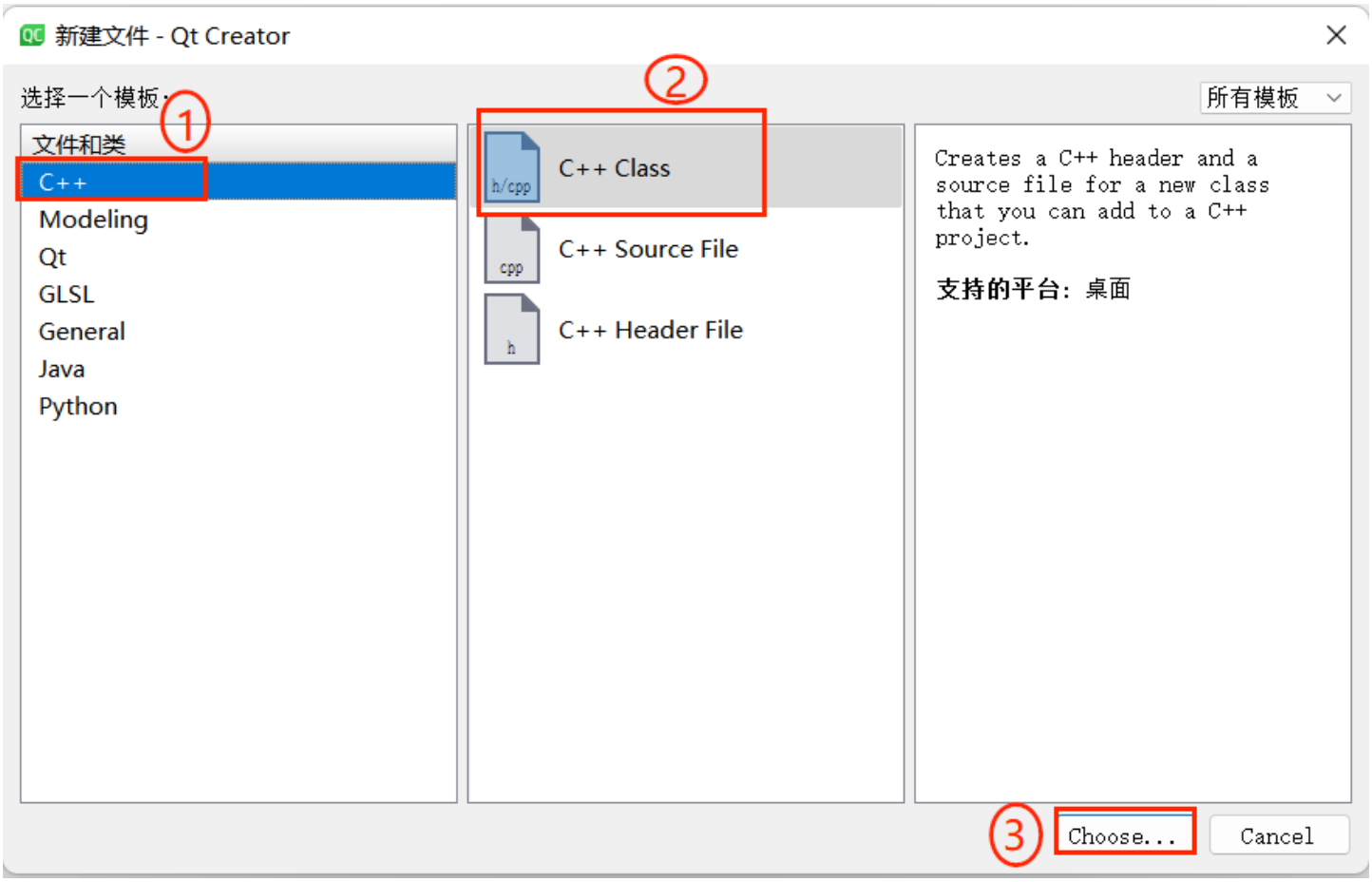


示例2：当老师说 "上课了"，学生们就 "回到座位，开始学习"。

1、在源文件中新建两个类，一个是老师类，一个是学生类；首先选中项目名称，鼠标右键 -----> "add new..."



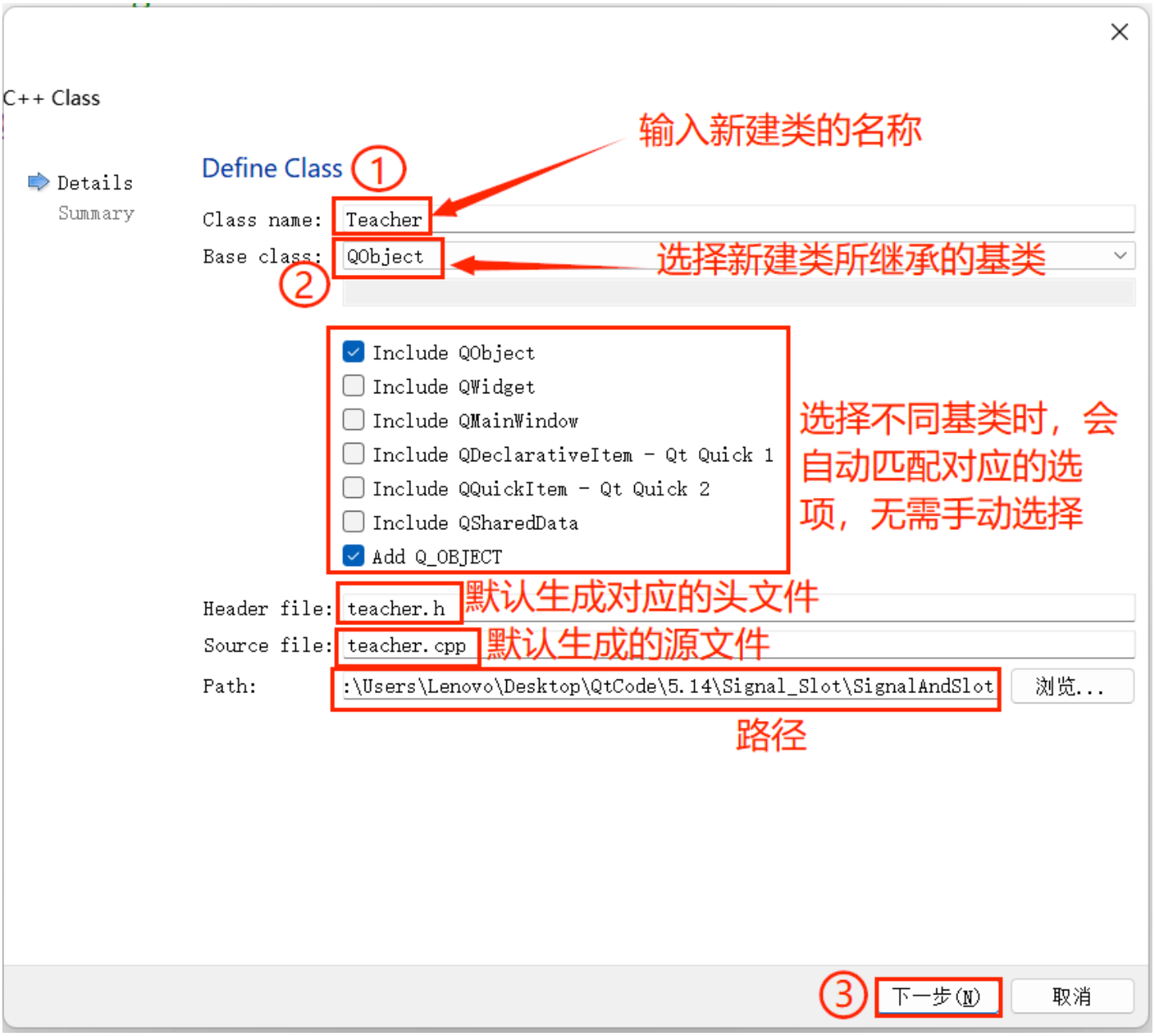
点击 "add new..." 之后，出现如下界面：



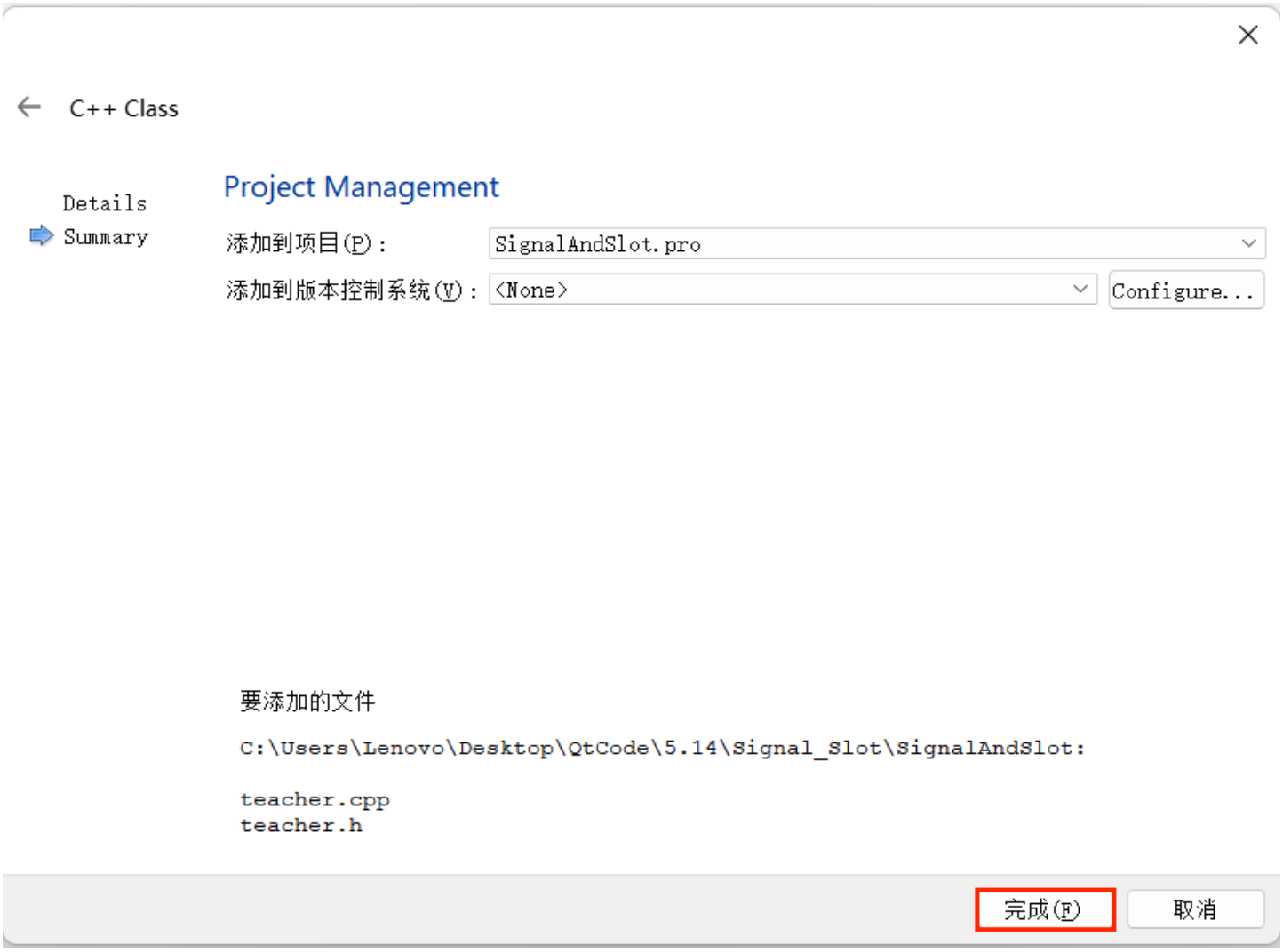
选择 "choose" 出现如下界面.

> **注意：**
>
> 在 Qt 中新建类时, 要选择新建类的父类.

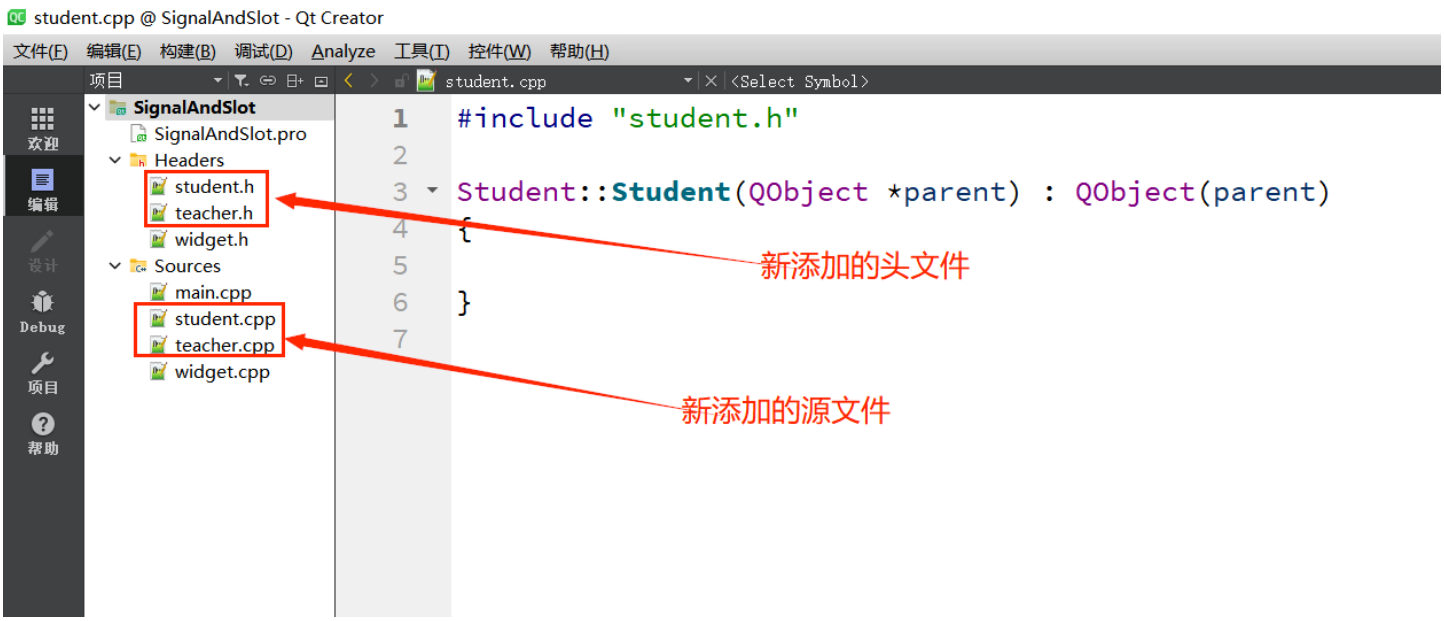
> 显然，当前项目中还没啥类适合做新类的父类, 同时新的类也不是一个 "窗口" 或者 "控件". 这种情况一般选择 QObject 作为基类.
>
> 这样做的好处是这个新类的对象可以搭配 Qt 的对象树机制. 便于对象的正确释放.



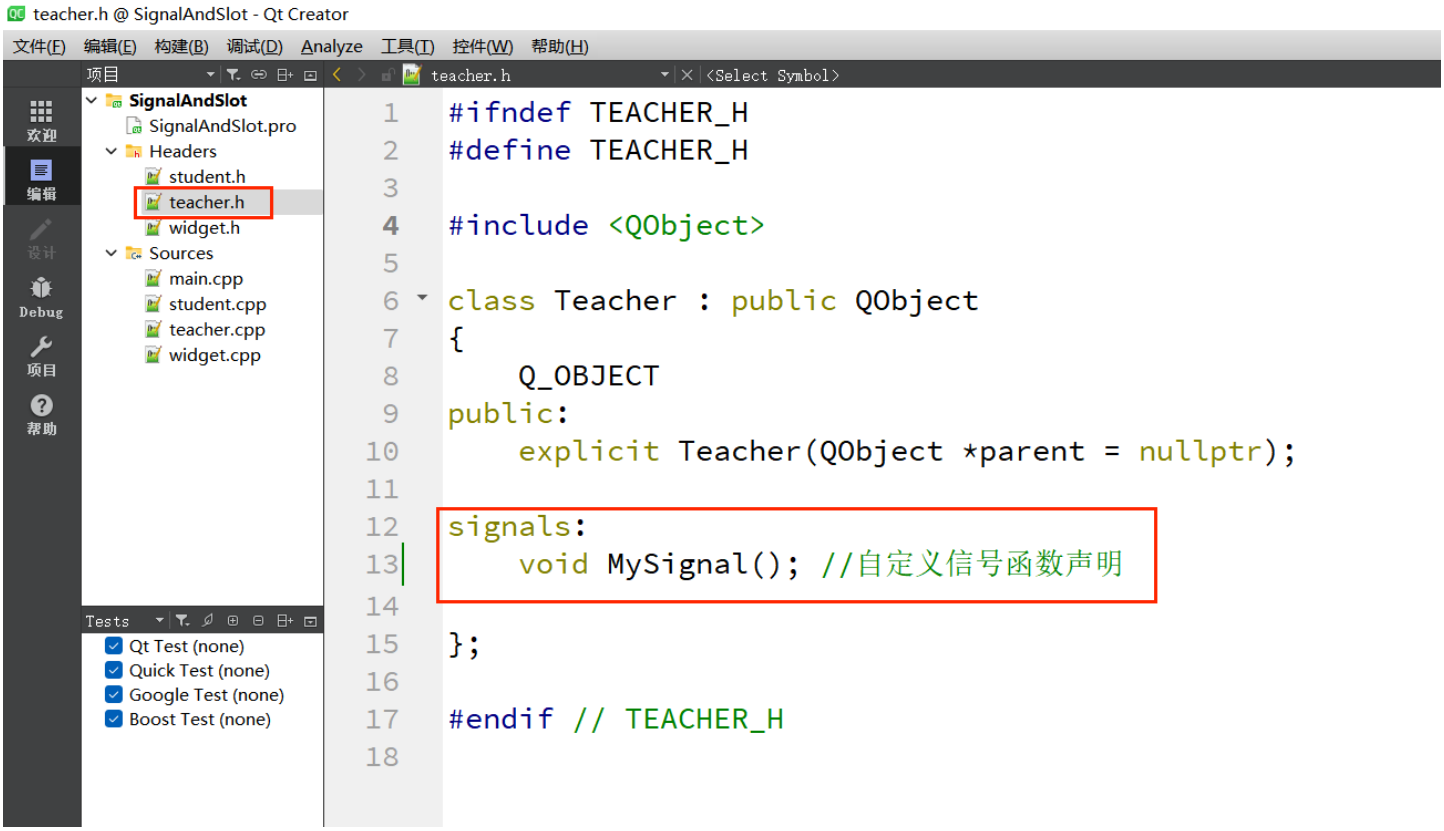
选择 "下一步"，出现如下界面：



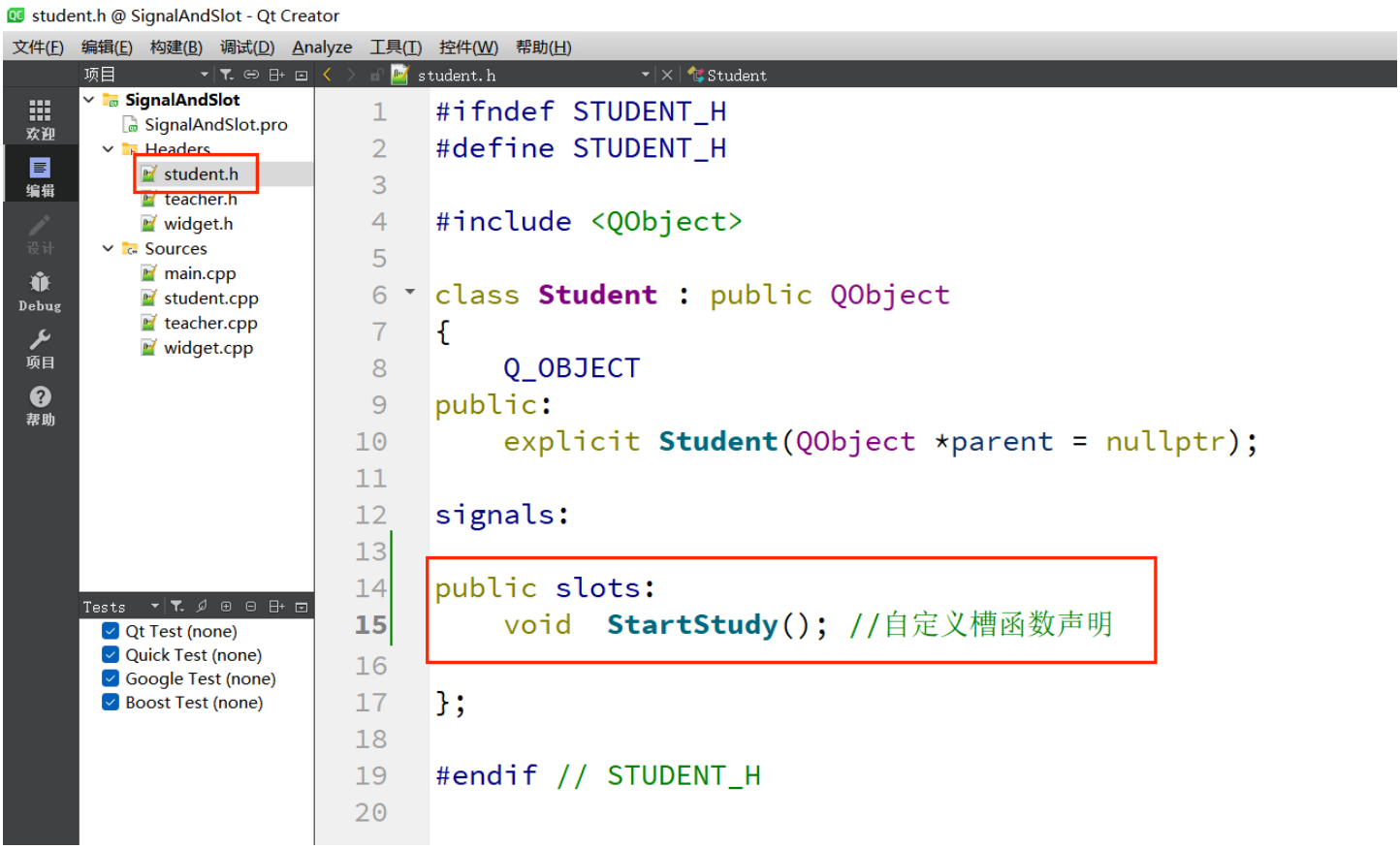
对于 "学生类" 以上述同样的方式进行添加，添加完成之后，项目目录新增文件如下：



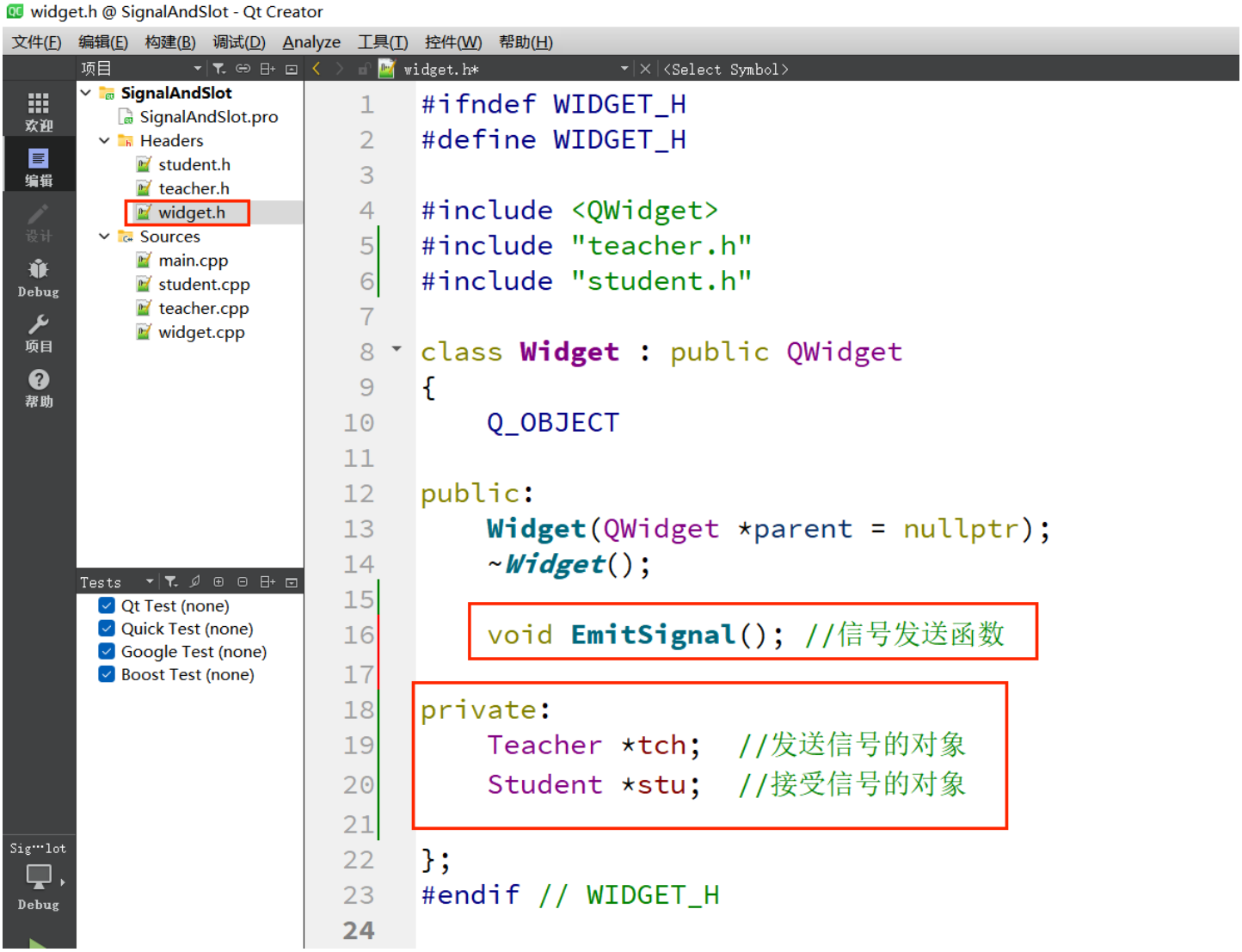
在 teacher.h 中声明信号函数：



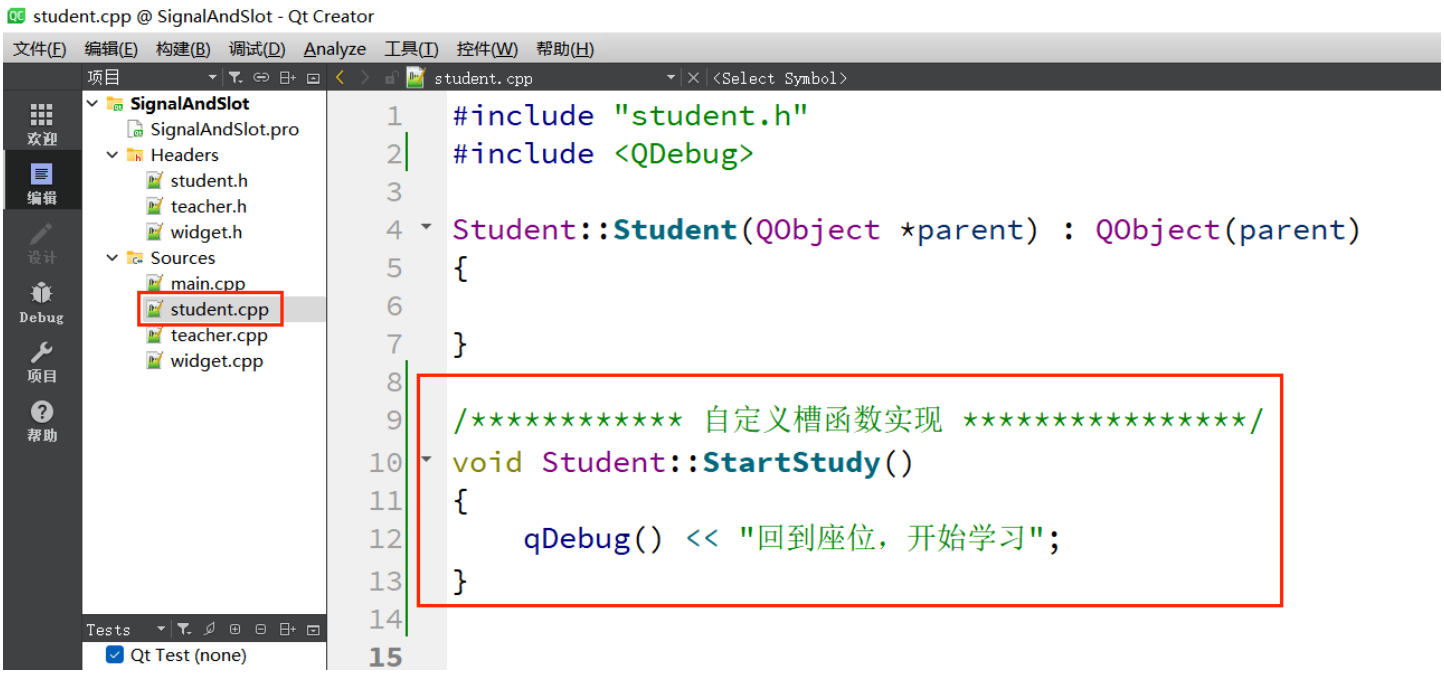
在 student.h 中声明槽函数：



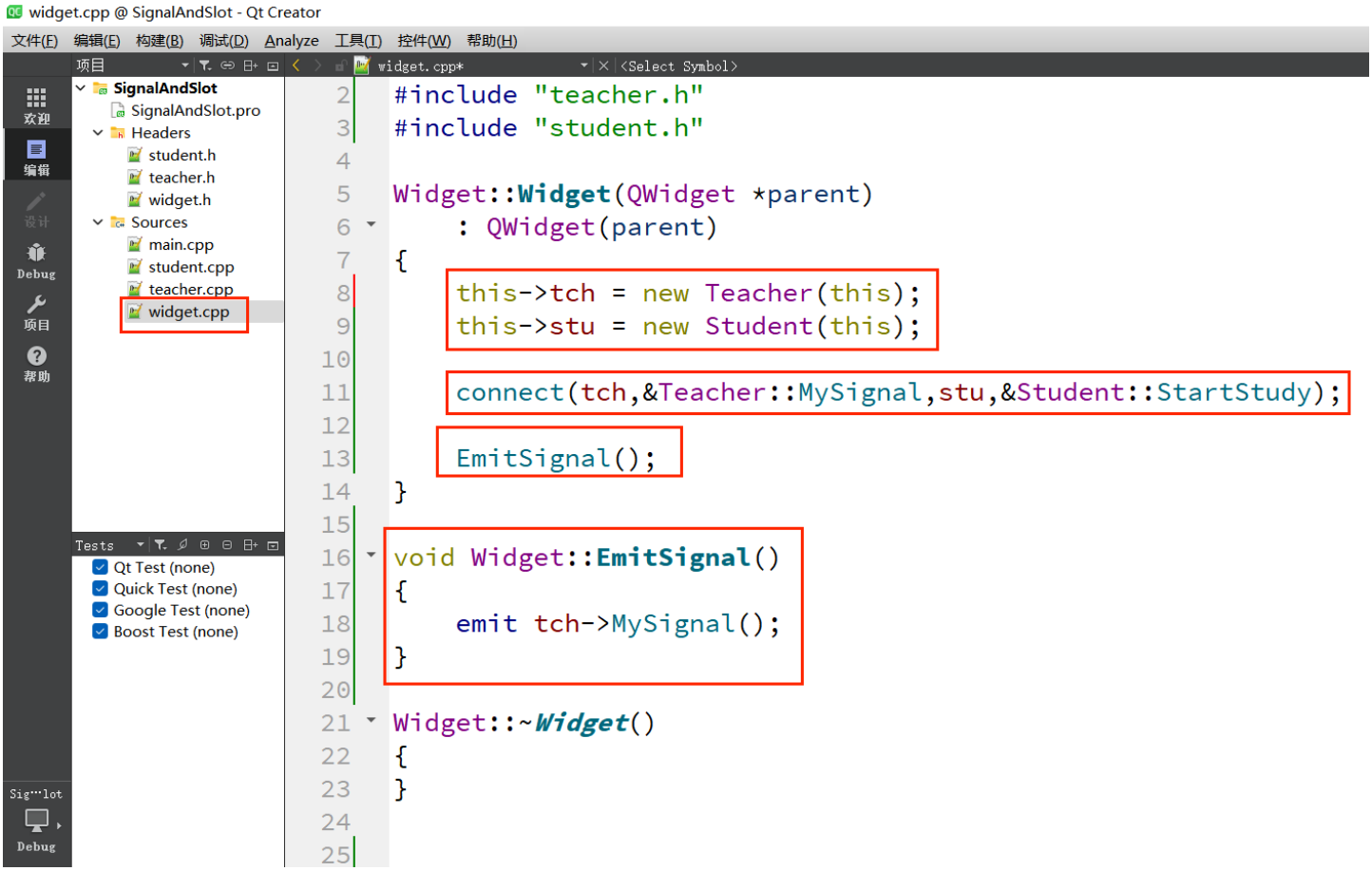
在 widget.h 中实例化 "老师类对象" 和 "学生类对象";



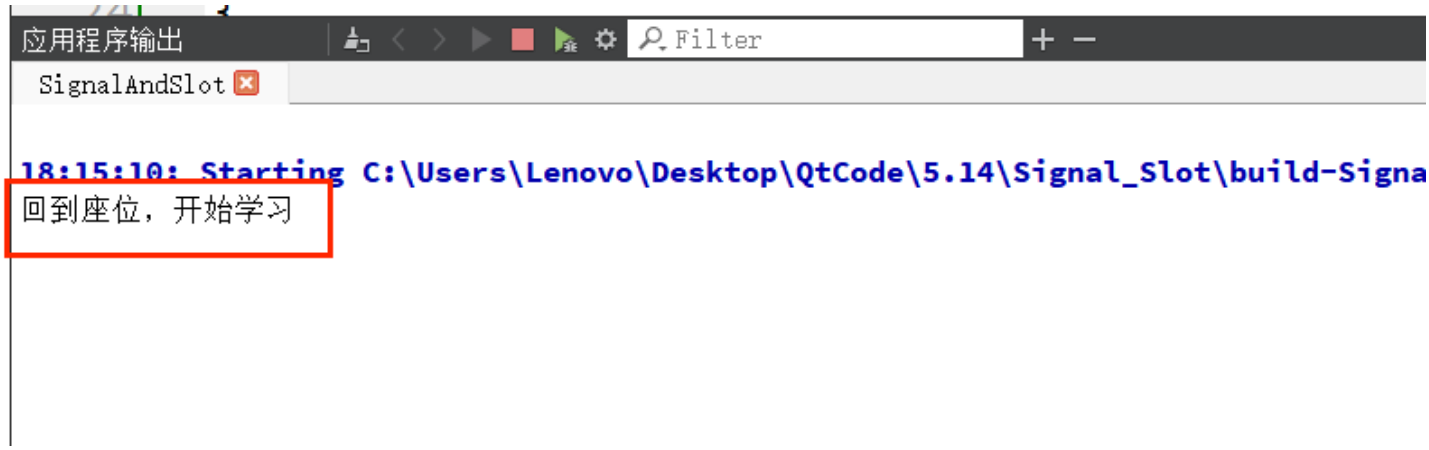
在 student.cpp 中实现槽函数：



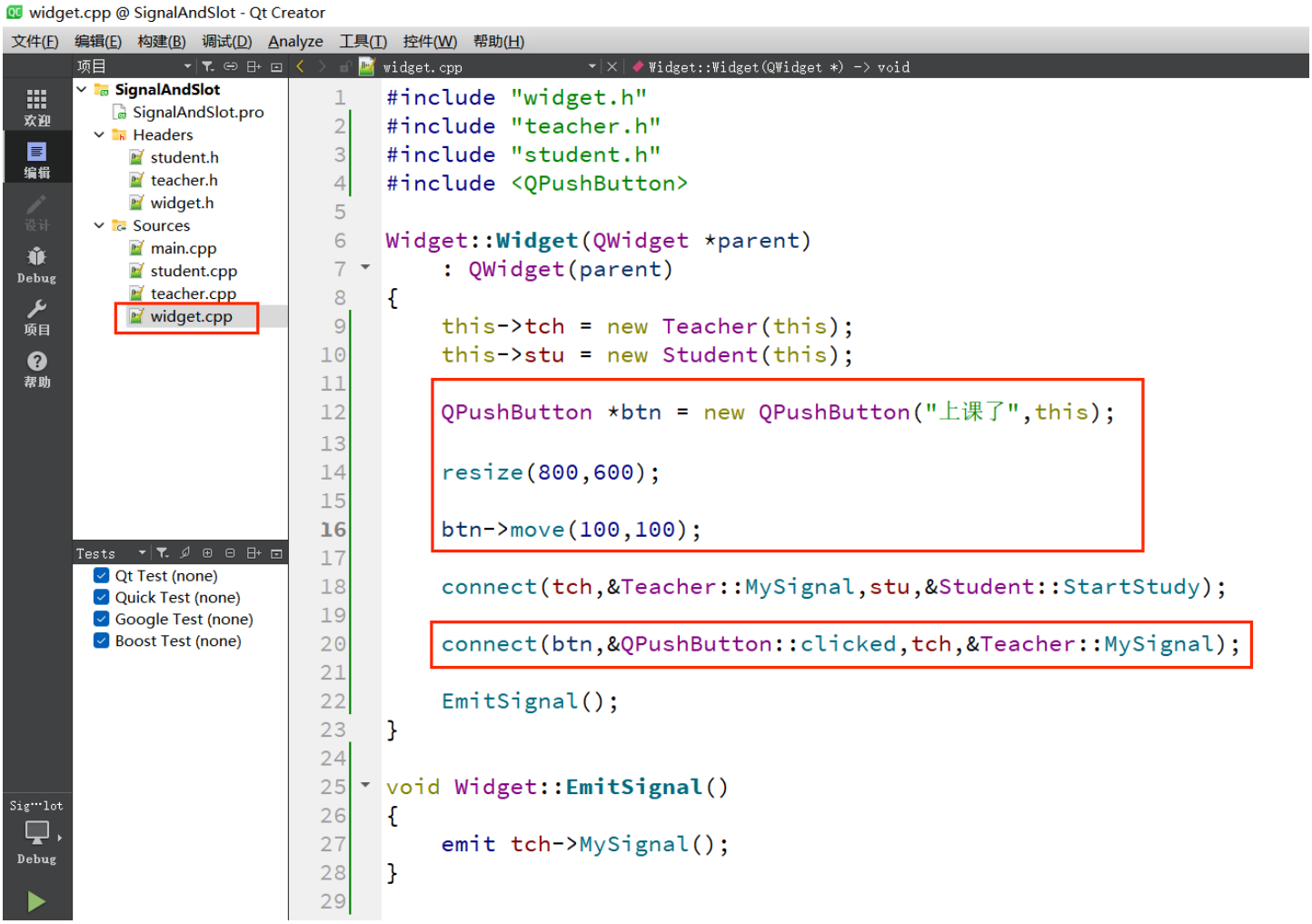
在 widget.cpp 中连接自定义信号和槽；



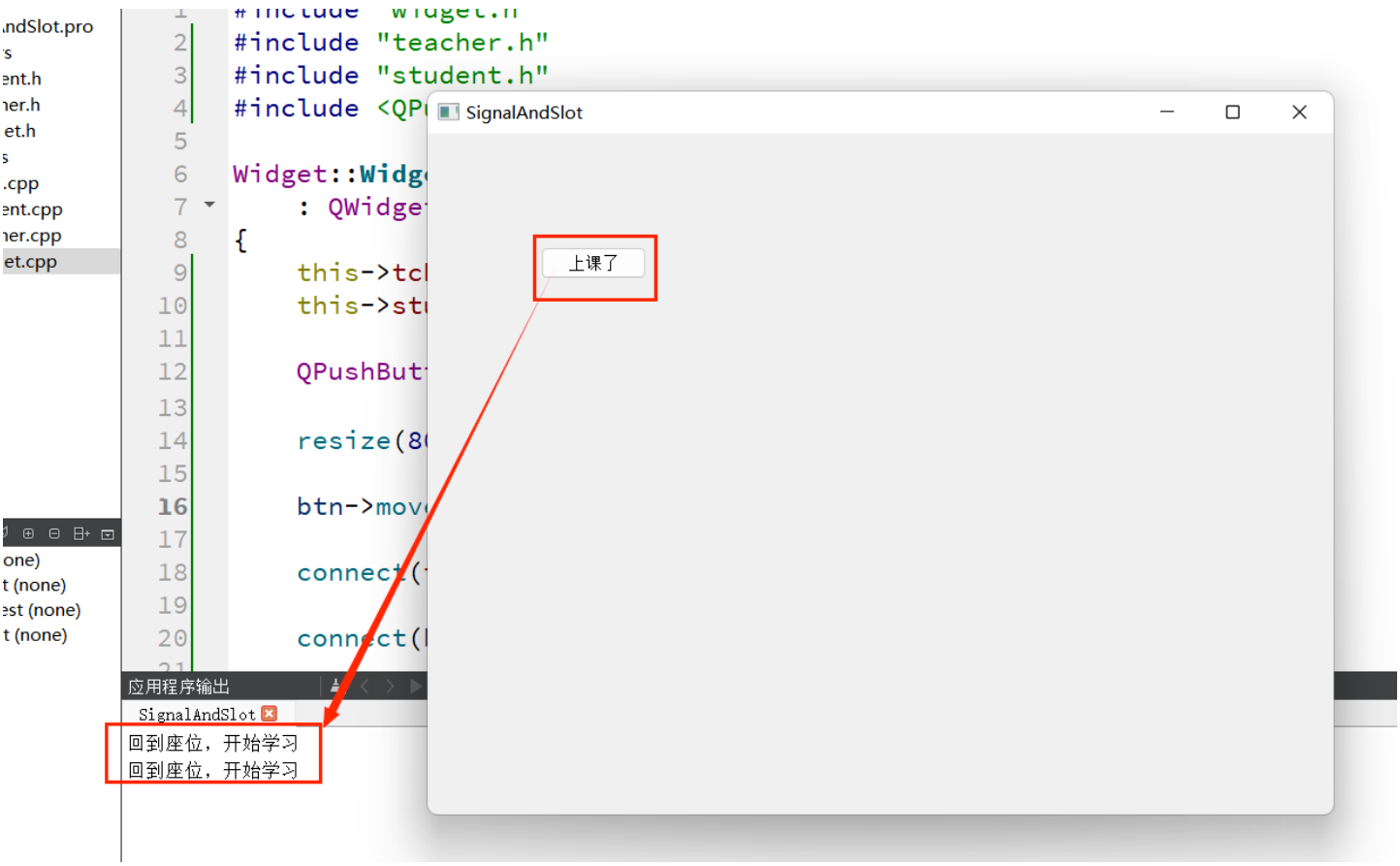
运行结果如下图示：



示例3：老师点击 "按钮" 触发学生上课；


widget.cpp @ SignalAndSlot - Qt Creator
文件(F) 编辑(E) 构建(B) 调试(D) Analyze 工具(T) 控件(W) 帮助(H)
项目
SignalAndSlot
SignalAndSlot.pro
Headers
student.h
teacher.h
widget.h
Sources
main.cpp
student.cpp
teacher.cpp
widget.cpp
Tests
Qt Test (none)
Quick Test (none)
Google Test (none)
Boost Test (none)
欢迎
编辑
设计
Debug
项目
帮助
Debug
Widget::Widget(QWidget *) -> void
#include "widget.h"
#include "teacher.h"
#include "student.h"
#include <QPushButton>
Widget::Widget(QWidget *parent)
: QWidget(parent)
{
this->tch = new Teacher(this);
this->stu = new Student(this);
QPushButton *btn = new QPushButton("上课了",this);
resize(800,600);
btn->move(100,100);
connect(tch,&Teacher::MySignal,stu,&Student::StartStudy);
connect(btn,&QPushButton::clicked,tch,&Teacher::MySignal);
EmitSignal();
}
void Widget::EmitSignal()
{
emit tch->MySignal();
}

运行结果如下图示：


SignalAndSlot
上课了
应用程序输出
SignalAndSlot
回到座位，开始学习
回到座位，开始学习

3.2 带参数的信号和槽

Qt 的信号和槽也支持带有参数, 同时也可以支持重载.

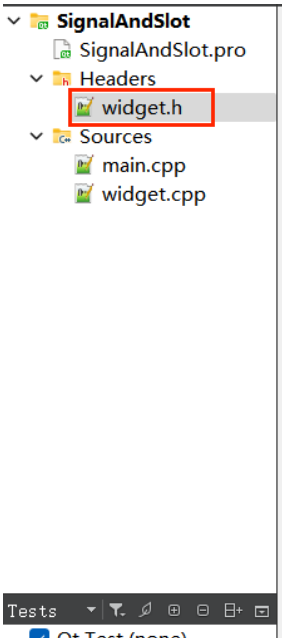
此处我们要求, **信号函数的参数列表要和对应连接的槽函数参数列表一致**.

此时信号触发, 调用到槽函数的时候, 信号函数中的实参就能够被传递到槽函数的形参当中.

> 通过这样的机制, 就可以让信号给槽传递数据了.

示例1：重载信号槽

（1）在 "widget.h" 头文件中声明重载的信号函数以及重载的槽函数；如下图所示：

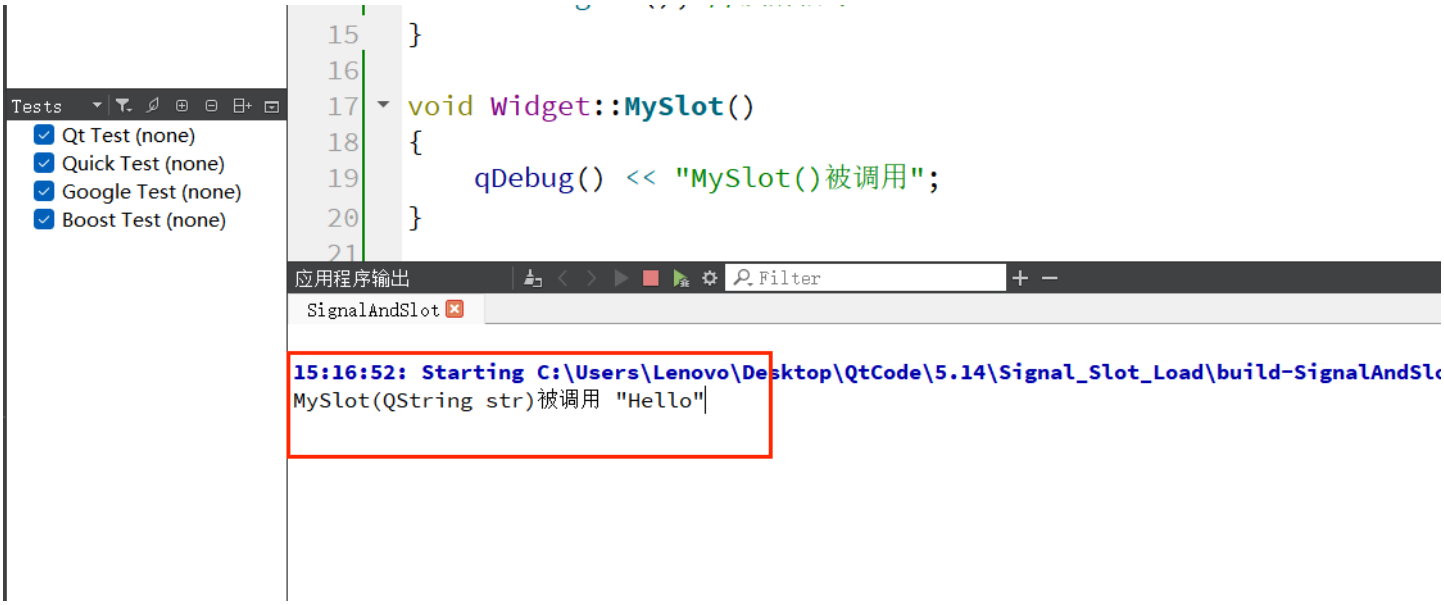



```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

    void EmitSignal(); //信号发射函数

signals:
    void MySignal();    //信号函数声明
    void MySignal(QString); //发生重载的信号函数声明

public slots:
    void MySlot();  //槽函数声明
    void MySlot(QString); //发生重载的槽函数声明

};
#endif // WIDGET_H
```

（2）在 "Widget.cpp" 文件实现重载槽函数以及连接信号和槽。

> 注意：在定义函数指针时要指明函数指针的作用域。

（3）执行结果如下图所示：



示例2：信号槽参数列表匹配规则

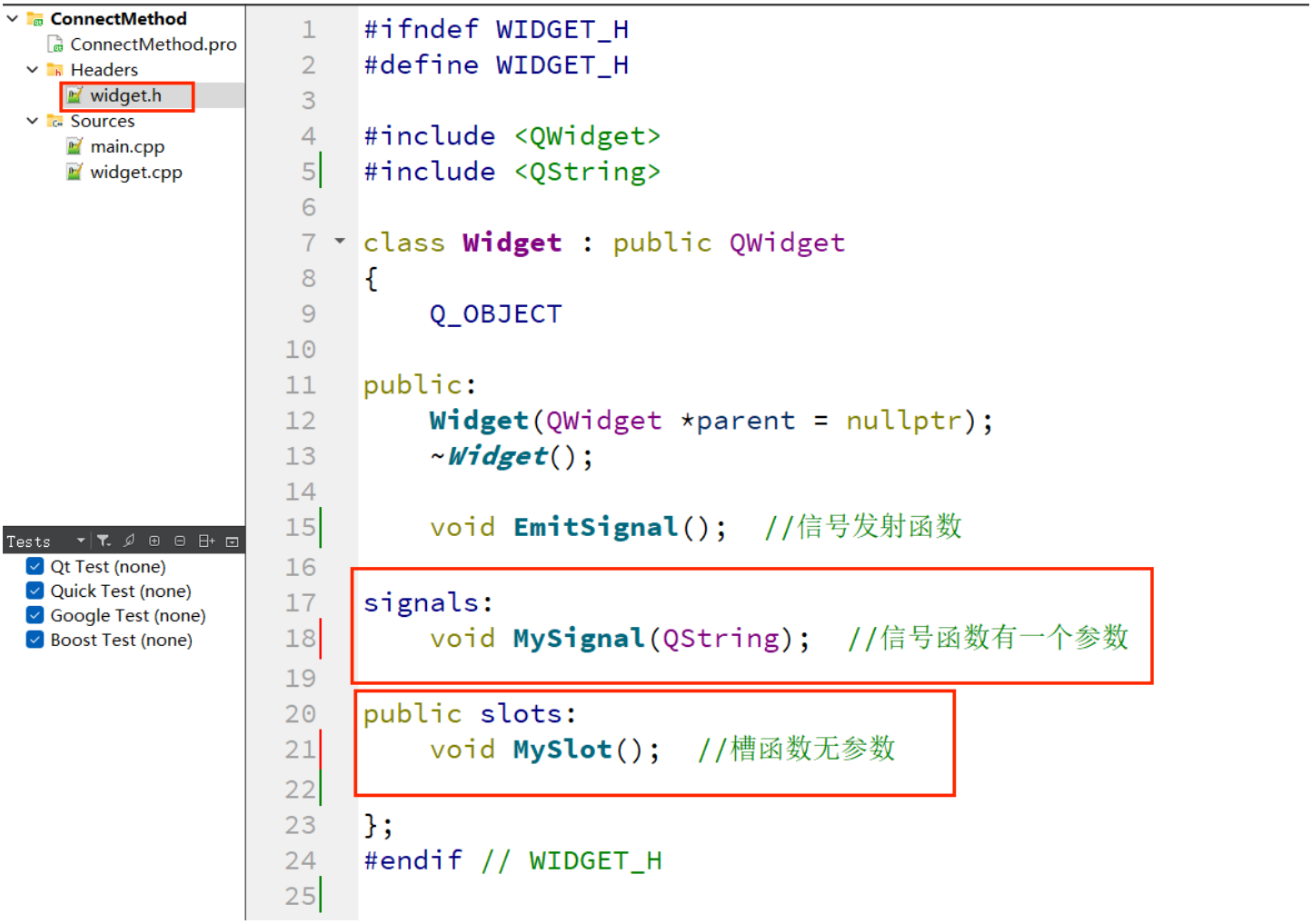
1、在 "widget.h" 头文件中声明信号和槽函数；

2、在 "widget.cpp" 文件中实现槽函数以及连接信号和槽；

> 其实信号的参数个数可以**多于槽函数的参数个数**，但是槽的参数个数不能多于信号参数个数.
>
> 但是实际开发中最好还是保持参数个数也能匹配一致.

示例3：

1、在 "widget.h" 头文件中声明信号和槽函数；



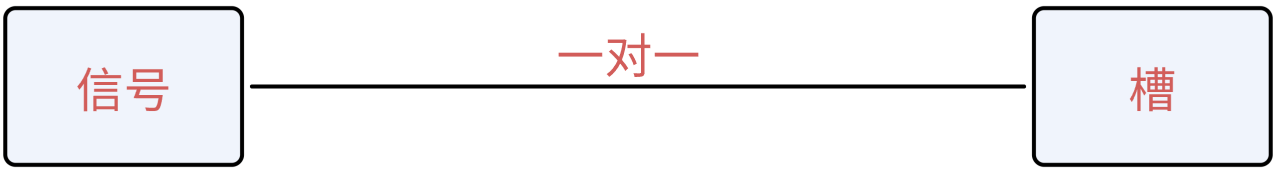
2、在 "widget.cpp" 文件中实现槽函数以及连接信号和槽；

4. 信号与槽的连接方式

4.1 一对一

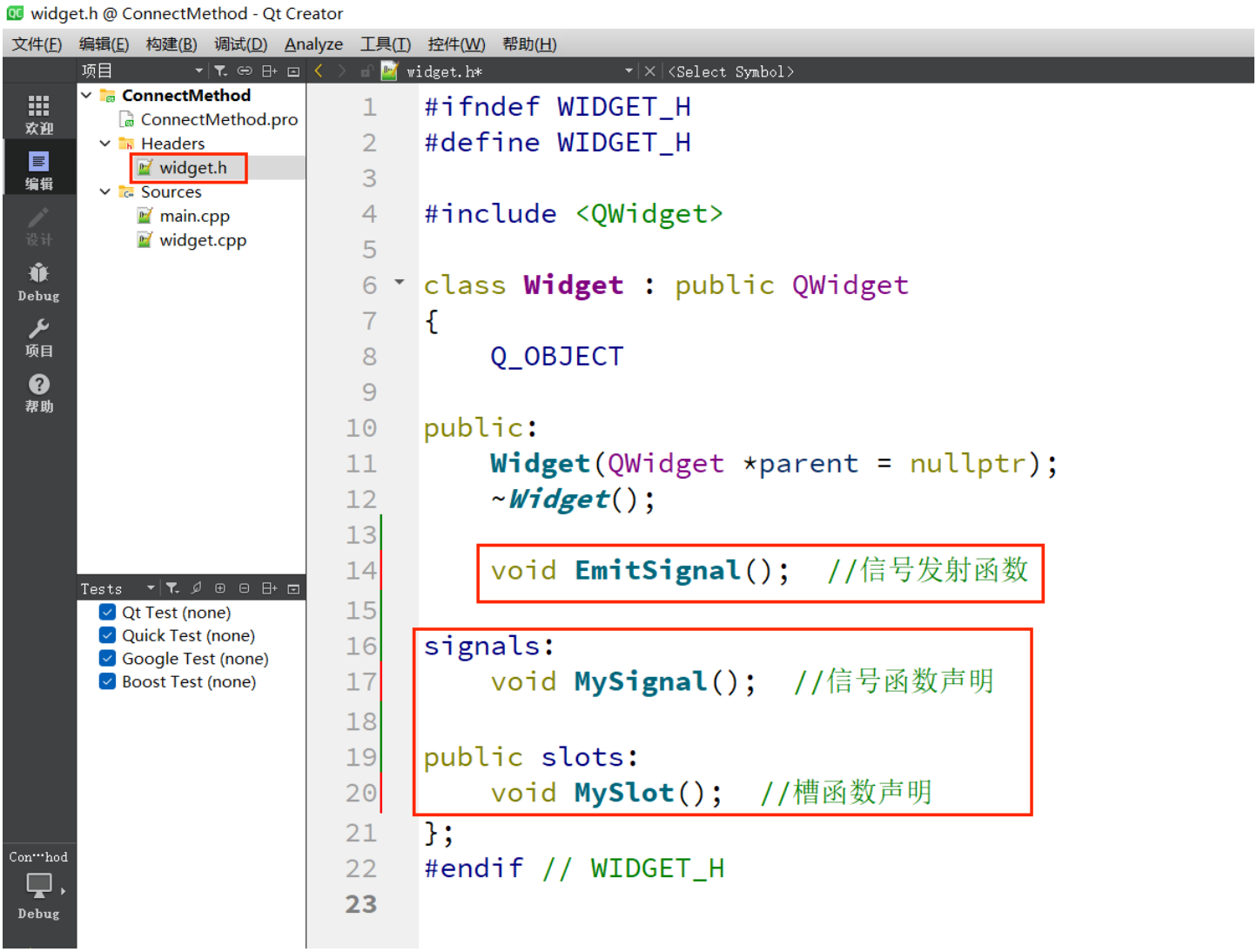
主要有两种形式，分别是：**一个信号连接一个槽** 和 **一个信号连接一个信号**。

（1）一个信号连接一个槽

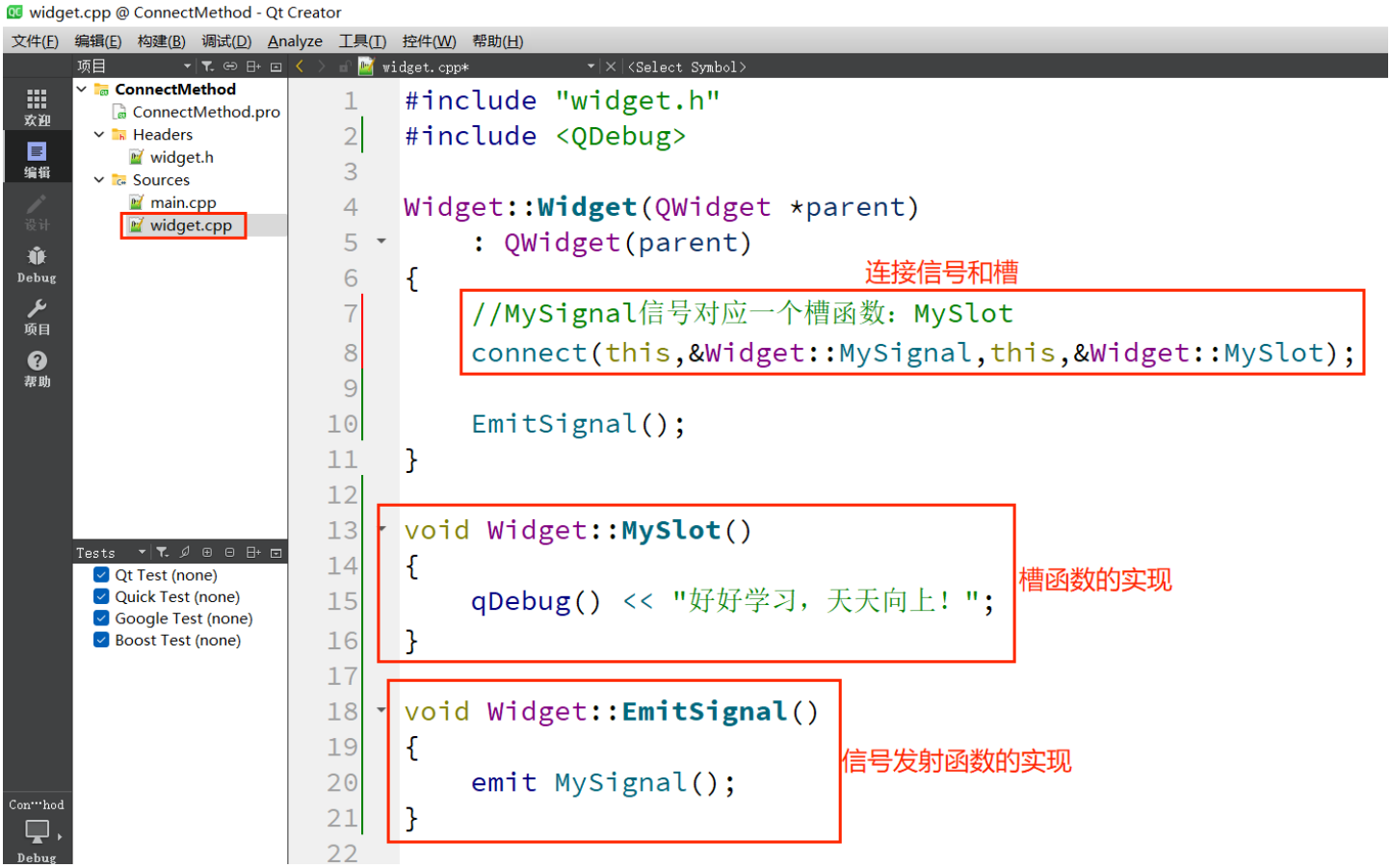


示例：

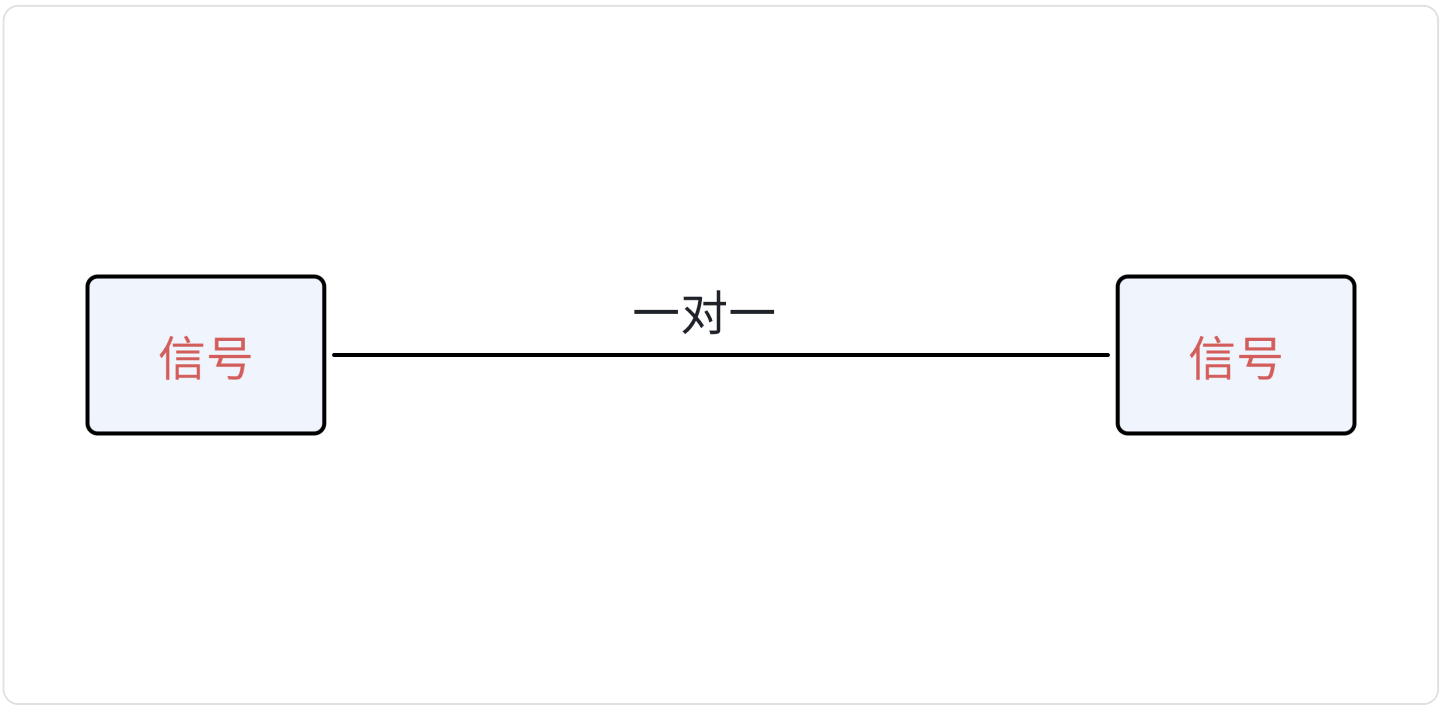
1、在 "widget.h" 中声明信号和槽以及信号发射函数；



2、在 "widget.cpp" 中实现槽函数，信号发射函数以及连接信号和槽；



（2）一个信号连接另一个信号



示例：

在上述示例的基础上，在 "widget.cpp" 文件中添加如下代码：

widget.cpp @ ConnectMethod - Qt Creator

文件(F) 编辑(E) 构建(B) 调试(D) Analyze 工具(T) 控件(W) 帮助(H)

项目

- ConnectMethod
  - ConnectMethod.pro
  - Headers
    - widget.h
  - Sources
    - main.cpp
    - widget.cpp

widget.cpp*

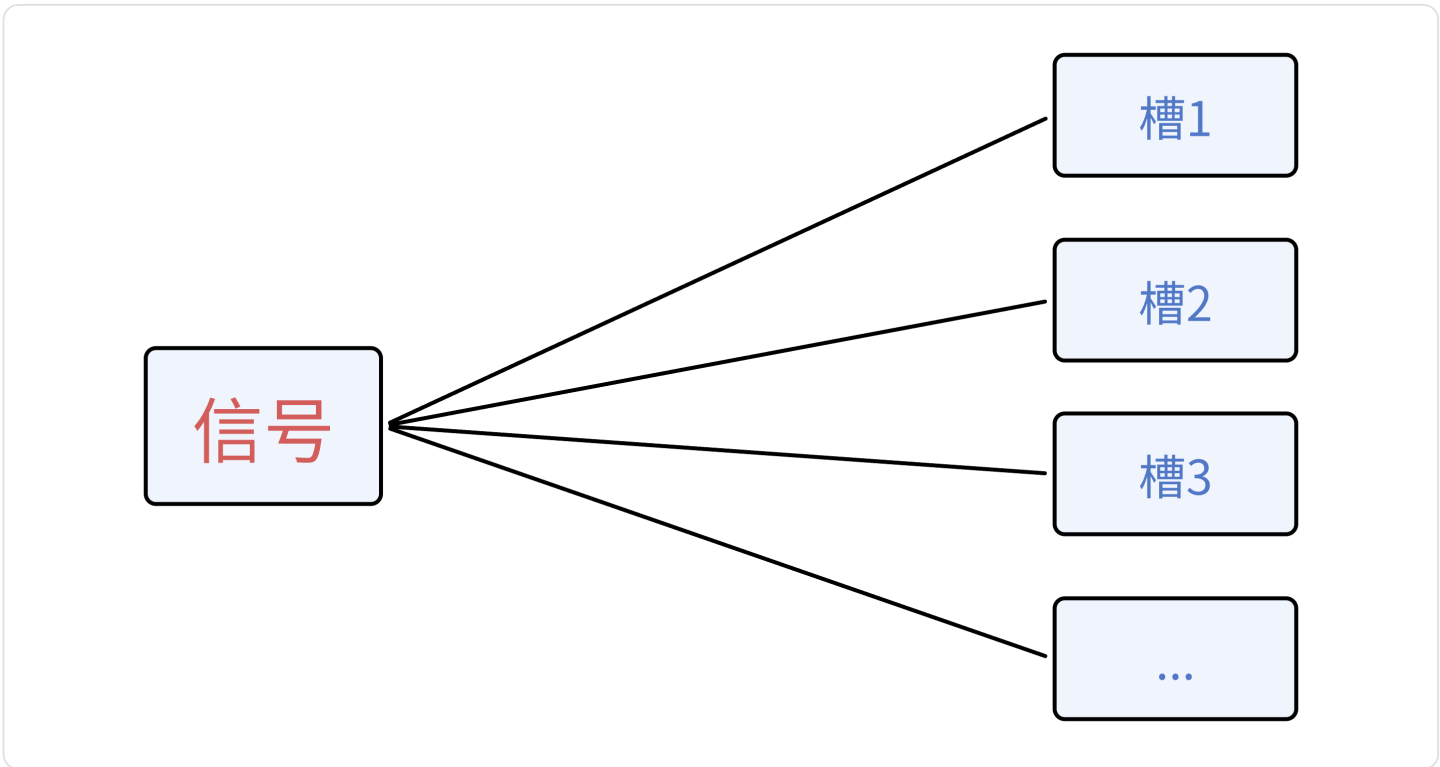
```cpp
#include "widget.h"
#include <QDebug>
#include <QPushButton>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    QPushButton *btn = new QPushButton("按钮",this);

    resize(800,600);

    connect(this,&Widget::MySignal,this,&Widget::MySlot);

    //信号与信号的连接
    connect(btn,&QPushButton::clicked,this,&Widget::MySignal);

    EmitSignal();
}

void Widget::MySlot()
{
    qDebug() << "好好学习，天天向上！";
}

void Widget::EmitSignal()
{
    emit MySignal();
}
```

Tests

- Qt Test (none)
- Quick Test (none)
- Google Test (none)
- Boost Test (none)

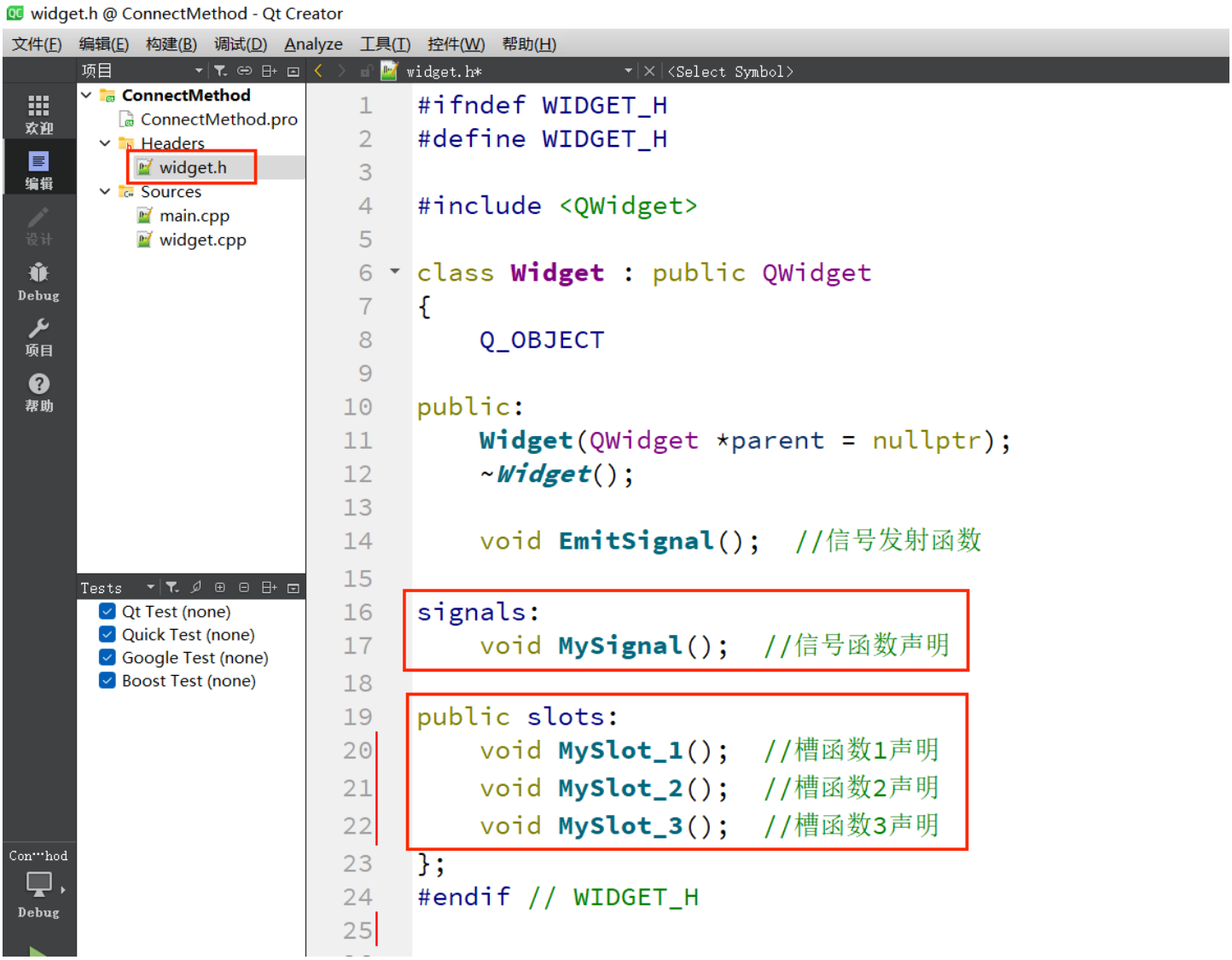
4.2 一对多

一个信号连接多个槽



示例：

（1）在 "widget.h" 头文件中声明**一个信号**和**三个槽**；



（2）在 "widget.cpp" 文件中实现槽函数以及连接信号和槽；

widget.cpp @ ConnectMethod - Qt Creator

文件(F) 编辑(E) 构建(B) 调试(D) Analyze 工具(T) 控件(W) 帮助(H)

项目 | widget.cpp* | Widget::Widget(QWidget *) -> void

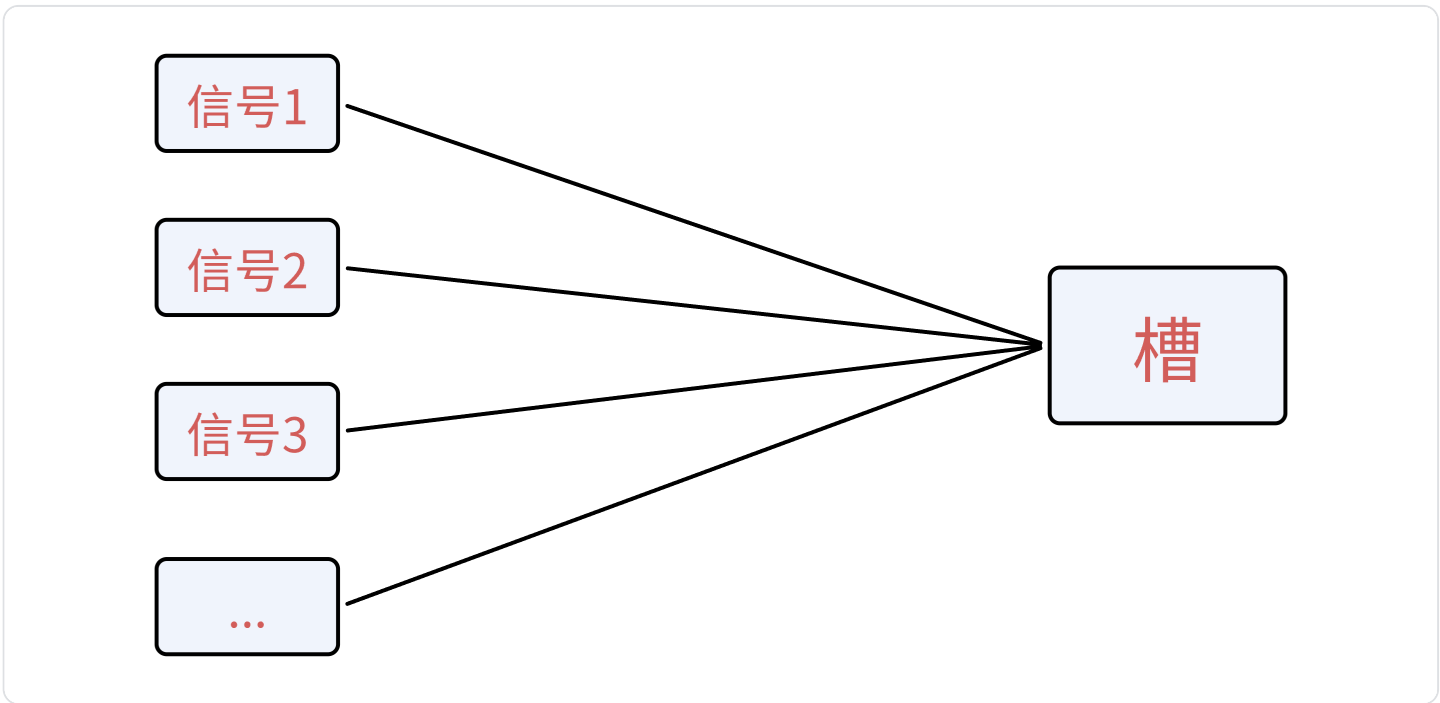
- ConnectMethod
  - ConnectMethod.pro
  - Headers
    - widget.h
  - Sources
    - main.cpp
    - widget.cpp

Tests
- Qt Test (none)
- Quick Test (none)
- Google Test (none)
- Boost Test (none)

```cpp
#include "widget.h"
#include <QDebug>
#include <QPushButton>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    QPushButton *btn = new QPushButton("按钮",this);
    btn->move(100,100);
    resize(800,600);

    connect(this,&Widget::MySignal,this,&Widget::MySlot_1); //MySignal信号 连接 槽1
    connect(this,&Widget::MySignal,this,&Widget::MySlot_2); //MySignal信号 连接 槽2
    connect(this,&Widget::MySignal,this,&Widget::MySlot_3); //MySignal信号 连接 槽3

    EmitSignal(); //发射信号
}

void Widget::MySlot_1()
{
    qDebug() << "MySlot_1";
}

void Widget::MySlot_2()
{
    qDebug() << "MySlot_2";
}

void Widget::MySlot_3()
{
    qDebug() << "MySlot_3";
}

void Widget::EmitSignal()
{
     emit MySignal();
}
```

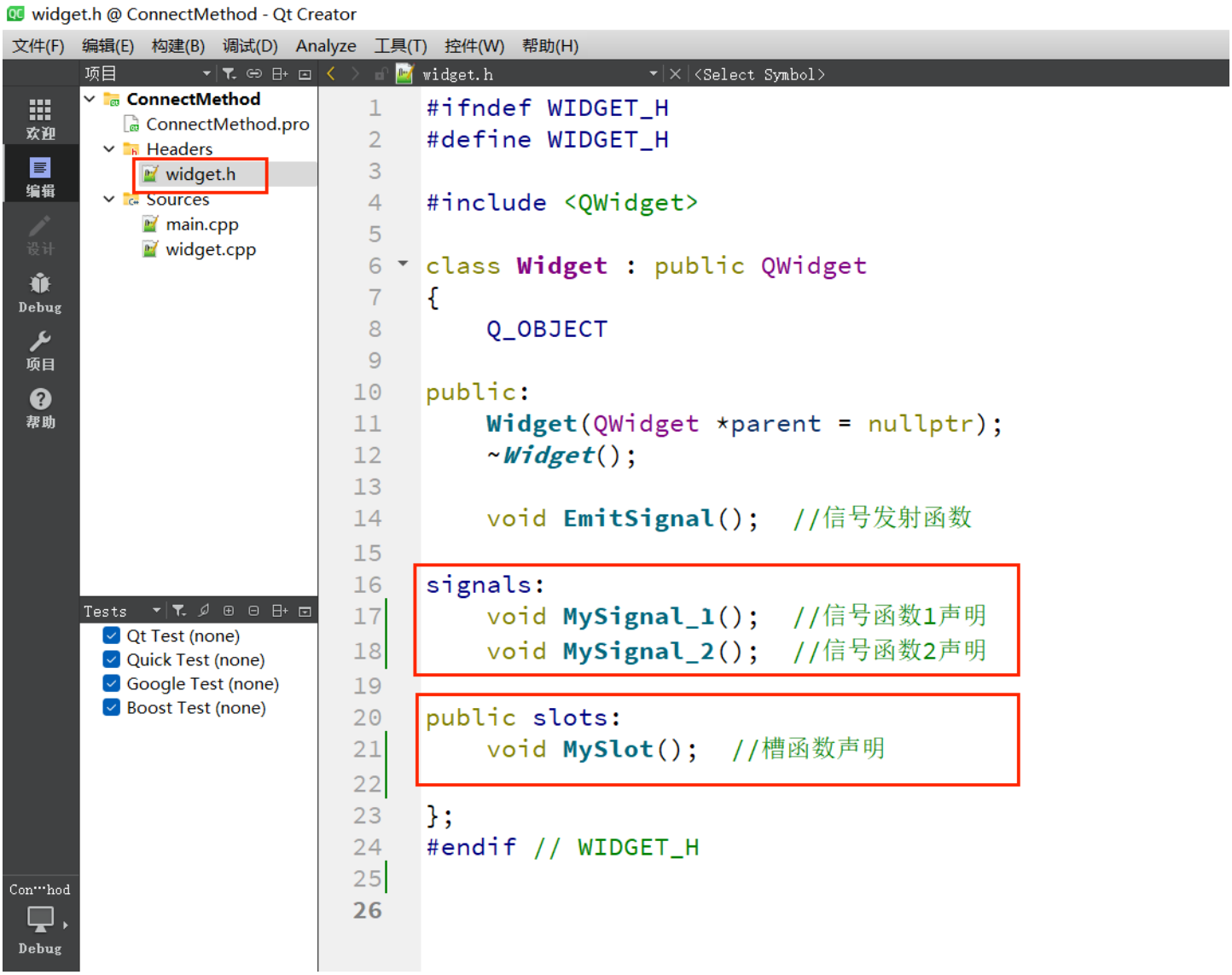
4.3 多对一

多个信号连接一个槽函数



示例：

（1）在 "widget.h" 头文件中**声明两个信号以及一个槽**；



（2）在 "widget.cpp" 文件中实现槽函数以及连接信号和槽；

5. 信号和槽的其他说明

5.1 信号与槽的断开

使用 disconnect 即可完成断开.

disconnect 的用法和 connect 基本一致.

示例:

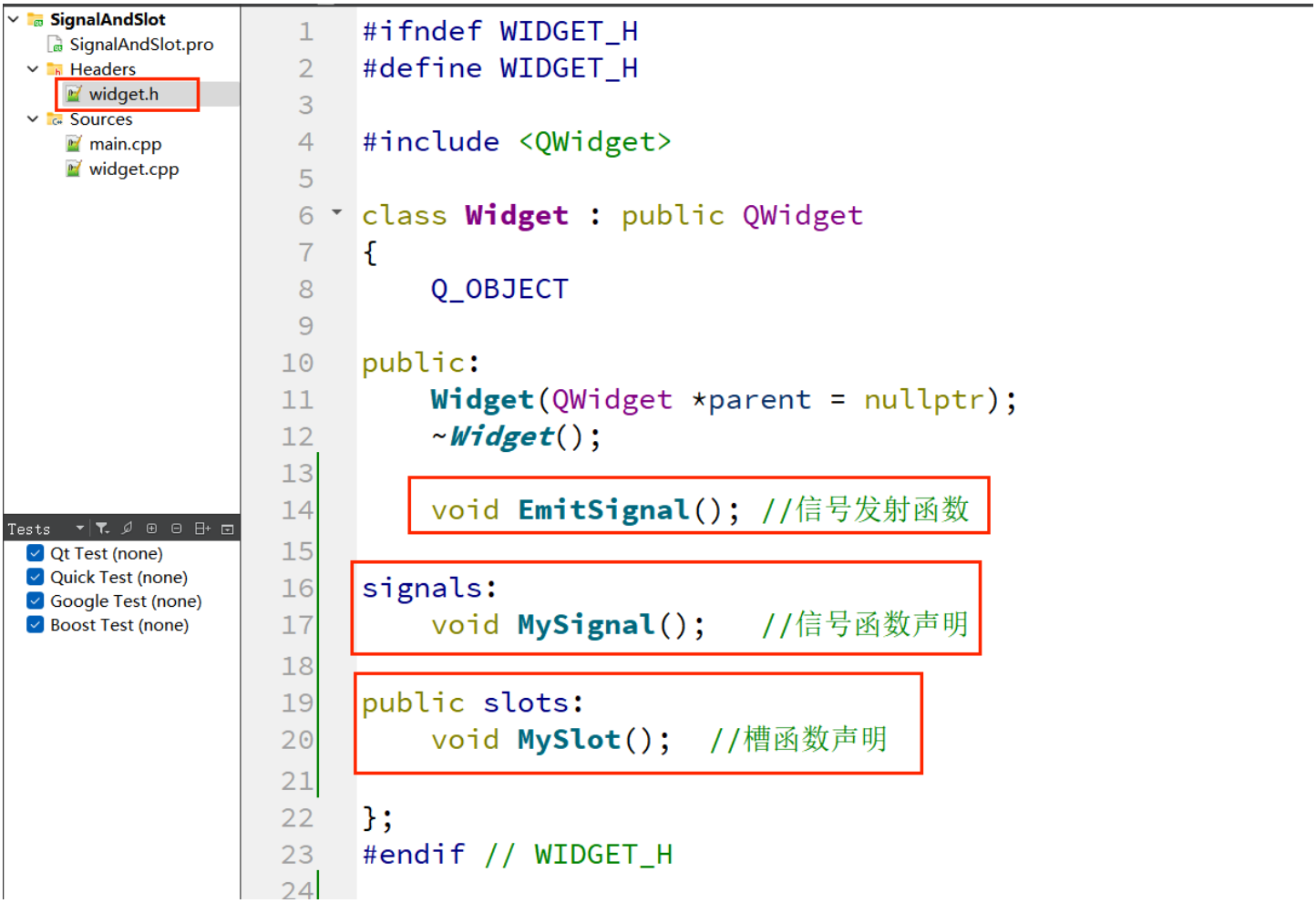
5.2 Qt4 版本信号与槽的连接

Qt4 中的 connect 用法和 Qt5 相比是更复杂的. 需要搭配 `SIGNAL` 和 `SLOT` 宏来完成.

而且缺少必要的函数类型的检查. 使代码更容易出错.

示例:

（1）在 "widget.h" 头文件中声明信号和槽

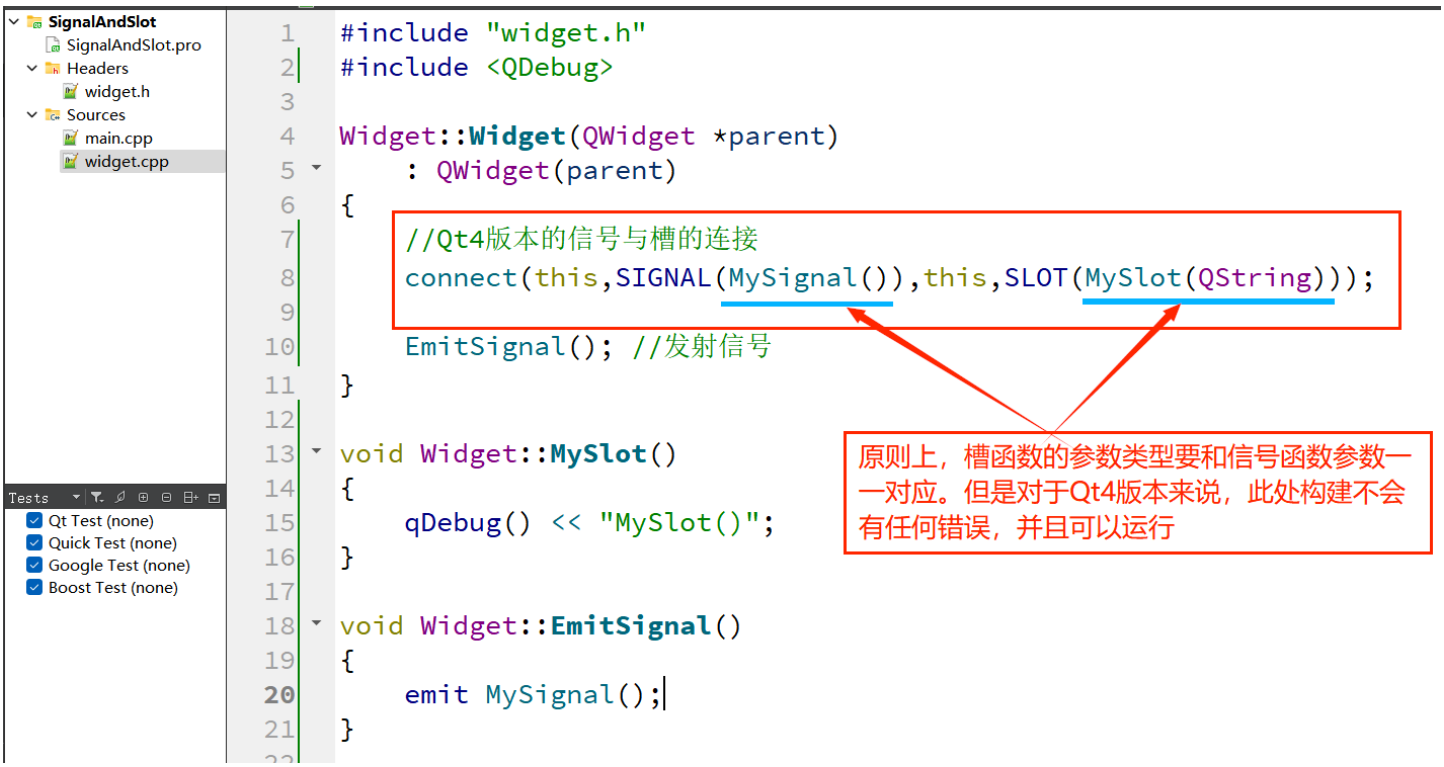


（2）在 "widget.cpp" 文件中实现槽函数以及连接信号与槽；

Qt4 版本信号与槽连接的优缺点：

- 优点：参数直观；
- 缺点：参数类型不做检测；

示例：



5.3 使用 Lambda 表达式定义槽函数

Qt5 在 Qt4 的基础上提高了信号与槽的灵活性，允许使用任意函数作为槽函数。

但如果想方便的编写槽函数，比如在编写函数时连函数名都不想定义，则可以通过 **Lambda表达式** 来达到这个目的。

Lambda表达式 是 C++11 增加的特性。C++11 中的 **Lambda表达式** 用于定义并创建匿名的函数对象，以简化编程工作。

Lambda表达式 的语法格式如下：

```
[ capture ] ( params ) opt -> ret {
    Function body;
};
```

说明：

capture	捕获列表
params	参数表
opt	函数选项
ret	返回值类型
Function body	函数体

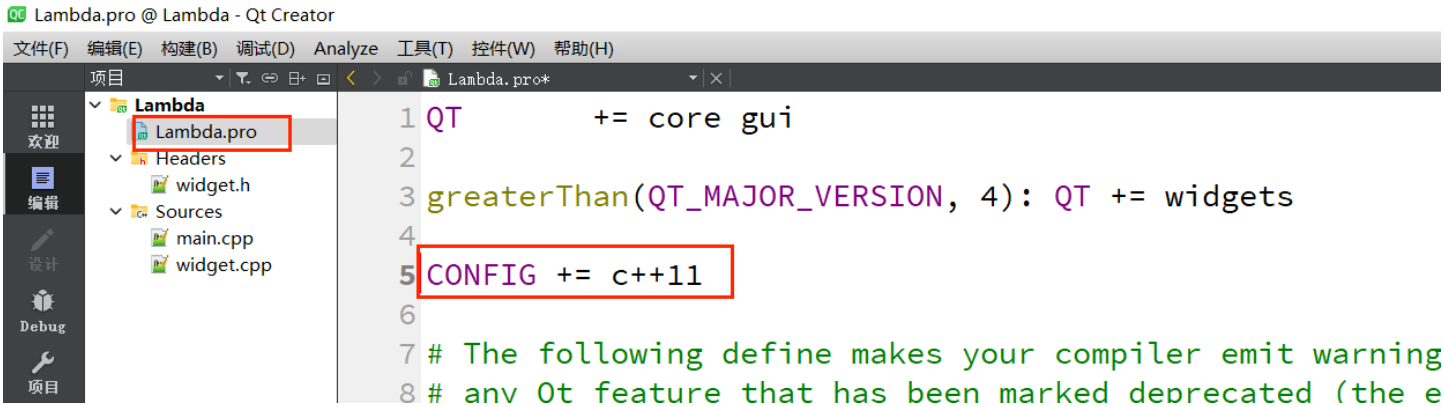
1、局部变量引入方式 [ ]

[ ] : 标识一个 **Lambda表达式 的开始。不可省略。**

符号	说明
[ ]	**局部变量捕获列表。Lambda表达式不能访问外部函数体的任何局部变量**
[a]	在函数体内部使用值传递的方式访问a变量
[&b]	在函数体内部使用引用传递的方式访问b变量
[=]	**函数外的所有局部变量都通过值传递的方式使用, 函数体内使用的是副本**
[&]	以引用的方式使用Lambda表达式外部的所有变量
[=, &foo]	foo使用引用方式, 其余是值传递的方式
[&, foo]	foo使用值传递方式，其余引用传递
[this]	在函数内部可以使用类的成员函数和成员变量，= 和 & 形式也都会默认引入

说明:

- 由于使用引用方式捕获对象会有局部变量释放了而Lambda函数还没有被调用的情况。如果执行Lambda函数，那么引用传递方式捕获进来的局部变量的值不可预知。所以绝大多数场合使用的形式为：`[=] () { }`
- 早期版本的 Qt，若要使用Lambda表达式，要在 ".pro" 文件中添加：`CONFIG += C++11`
  因为 Lambda表达式 是 C++11 标准提出的。Qt5 以上的版本无需手动添加，在新建项目时会自动添加。



示例1：Lambda表达式的使用

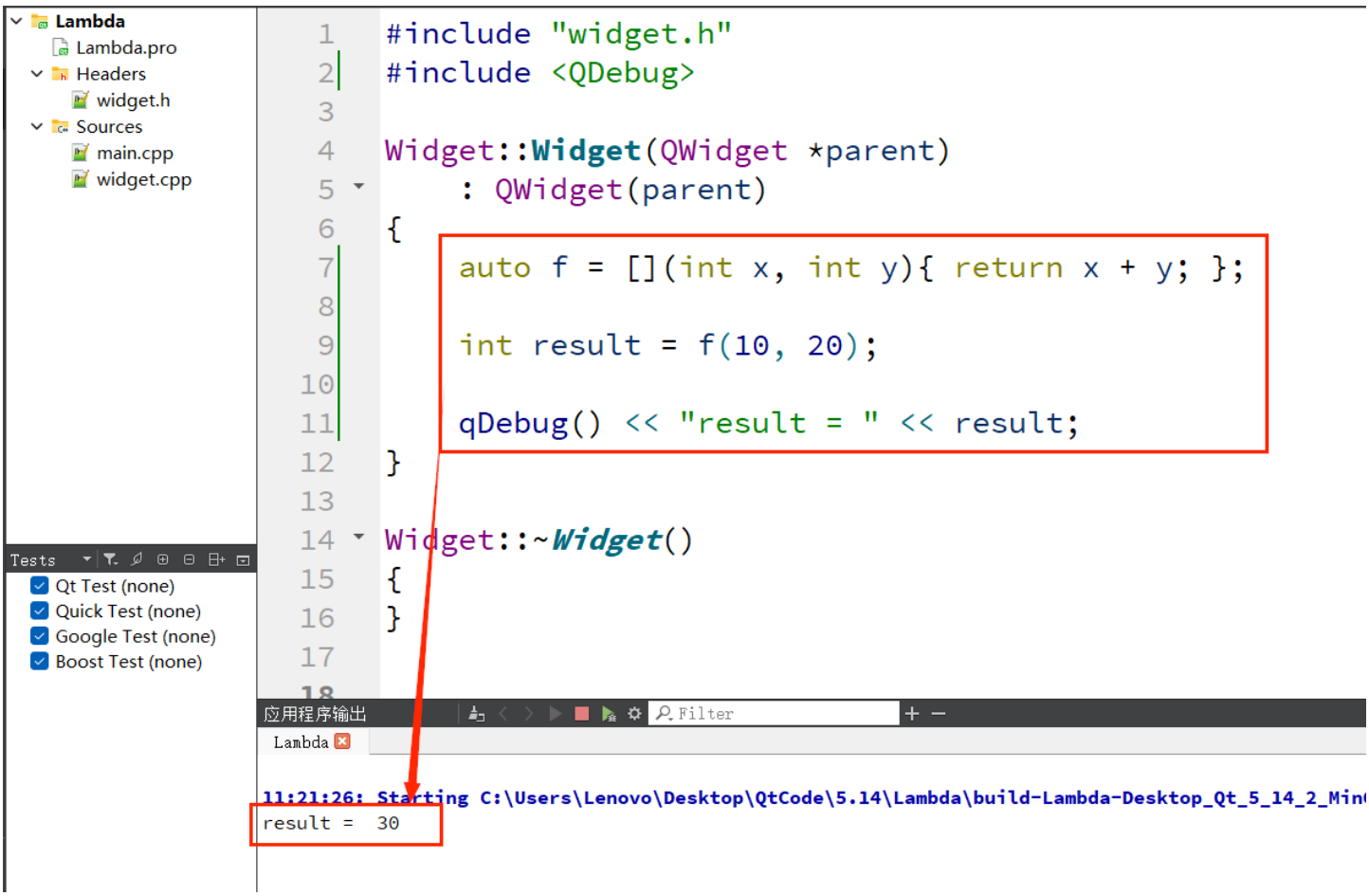
示例2：以 [=] 方式传递，外部的所有变量在Lambda表达式中都可以使用

示例3：以 [a] 方式传递，在Lambda表达式中只能使用传递进来的 a

2、函数参数 ( )

(params) 表示 Lambda函数对象接收的参数，类似于函数定义中的小括号表示函数接收的参数类型和个数。参数可以通过按值（如：(int a,int b)）和按引用（如：(int &a,int &b)）两种方式进行传递。**函数参数部分可以省略**，省略后相当于无参的函数。

示例：



3、选项Opt

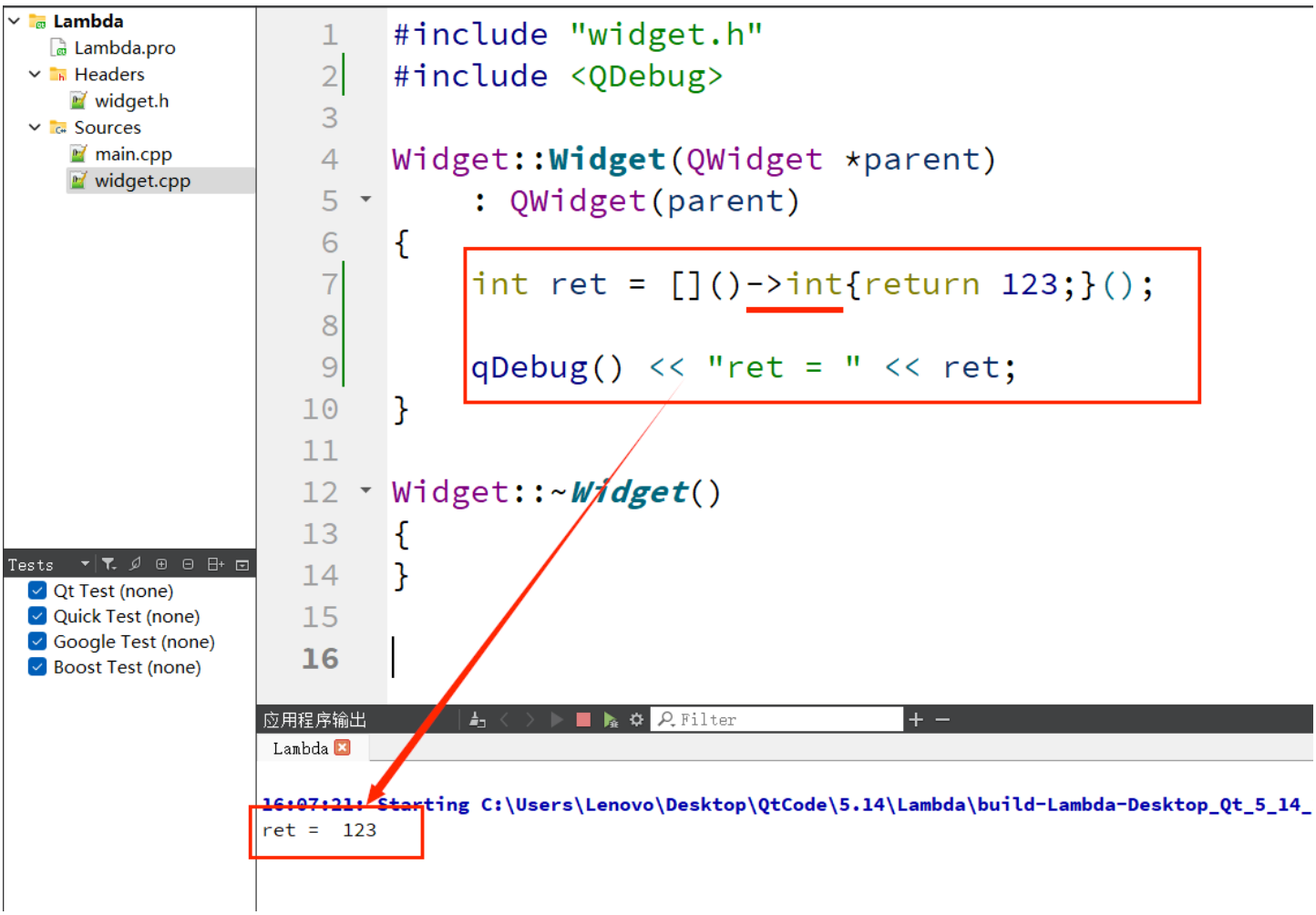
Opt 部分是可选项，最常用的是 `mutable声明` ，这部分**可以省略**。

Lambda表达式外部的局部变量通过值传递进来时，其**默认是 const**，所以**不能修改这个局部变量的拷贝**，加上mutable 就可以修改。

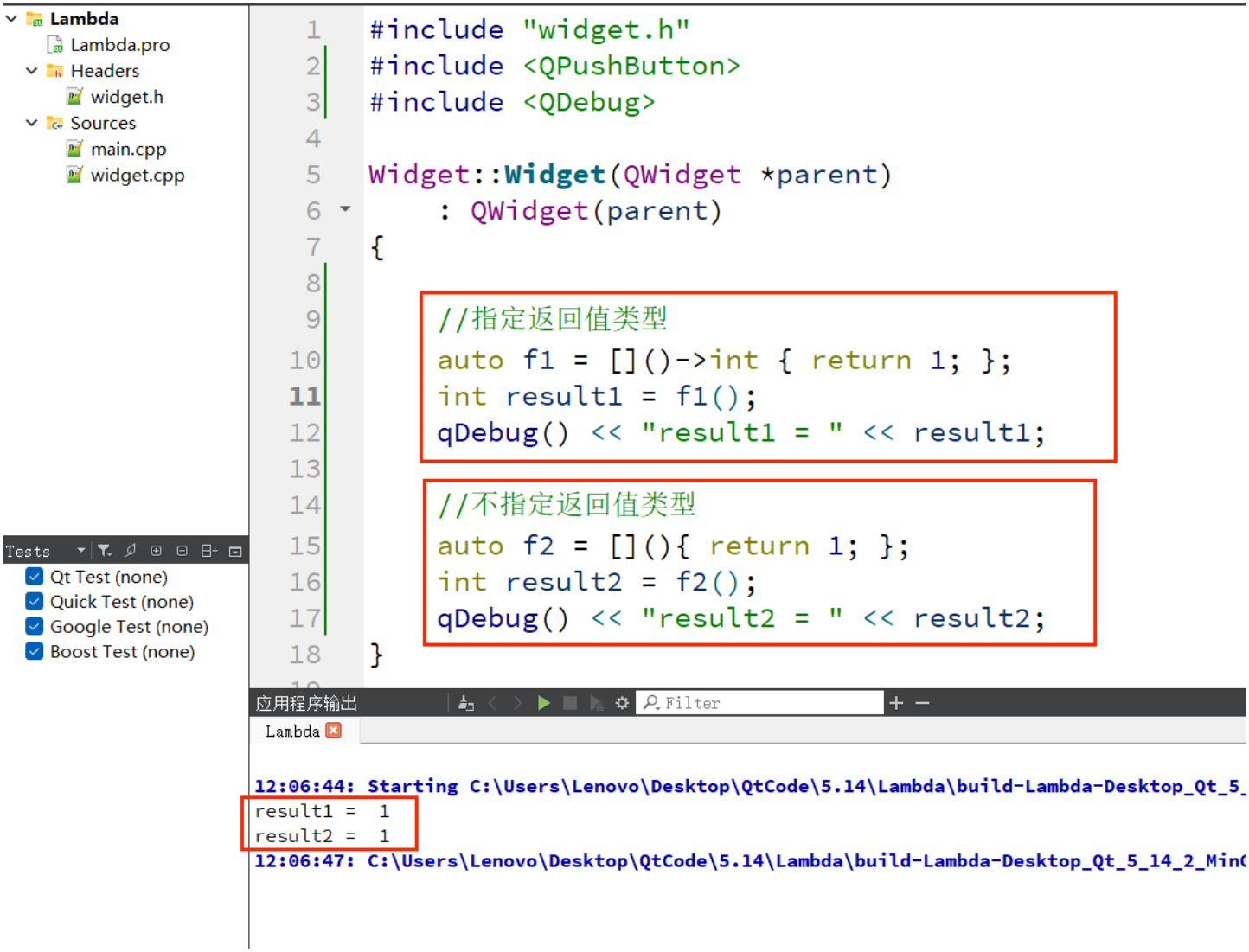
4、Lambda表达式的返回值类型 `->`

可以指定 Lambda表达式 返回值类型；如果不指定返回值类型，则编译器会根据代码实现为函数推导一个返回类型；如果没有返回值，则可忽略此部分。

示例1：



示例2：



5、Lambda表达式的函数体 `{ }`

Lambda表达式的函数体部分与普通函数体一致。用 `{ }` 标识函数的实现，**不能省略，但函数体可以为空。**

示例：

6、槽函数使用Lambda表达式来实现

示例1：点击按钮关闭窗口；

示例2：当 "connect" 函数第三个参数为 "this" 时，第四个参数使用 Lambda表达式时，可以省略掉 "this"；

5.4 信号与槽的优缺点

优点: 松散耦合

信号发送者不需要知道发出的信号被哪个对象的槽函数接收，槽函数也不需要知道哪些信号关联了自己，Qt的信号槽机制保证了信号与槽函数的调用。支持信号槽机制的类或者父类必须继承于 QObject 类。

缺点: 效率较低

与回调函数相比，信号和槽稍微慢一些，因为它们提供了更高的灵活性，尽管在实际应用程序中差别不大。通过信号调用的槽函数比直接调用的速度慢约10倍（这是定位信号的接收对象所需的开销；遍历所有关联；编组/解组传递的参数；多线程时，信号可能需要排队），这种调用速度对性能要求不是非常高的场景是可以忽略的，是可以满足绝大部分场景。

> 一个客户端程序中, 最慢的环节往往是 "人".
>
> 假设本身基于回调的方式是 10us, 使用信号槽的方式是 100us. 对于使用程序的人来说, 是感知不到的.