天津大学

《C++程序设计》实验报告

学　　院　国际工程师学院

专　　业　计算机技术

年　　级　2017

姓　　名　杨毅

2018 年　6 月　10 日

实验一 简易计算器

一、实验内容

使用 Qt 设计并实现带用户界面的简易计算器。计算器支持运算的数据类型为整数和小数。基本功能和 Windows10 自带的简易计算器相似。支持基本的加、减、乘、除四则运算。还支持一些单目运算：开方，平方，取倒数和取相反数。另外，还有两个主要的控制功能：清除和退格。本次实验中，一些错误提示及单目运算的使用方法本人完全参照 Window10 自带的简易计算器。

二、代码结构

实现过程中没有添加新类，所有的功能都在 MainWindow 主类中实现。类中添加的属性和方法如表 1 所示：

属性/方法	功能、作用
bool lastIsNum	上一个输入是否为数字
bool lastIsSinOpt	上一个输入是否为单目操作符
bool lastIsLBracket	记录上一个输入是否为左括号
bool lastIsRBracket	上一个输入是否为右括号
bool expClean	下一次输入是否需要清空 expression label
bool inputClean	下一次输入是否需要清空 input label
int lBracketNum	当前左括号数
int index	左括号和弹幕操作符的位置
QStack <double> operands	操作数栈
QStack <char> operators	操作符栈
void inputNumver(int)	输入数字，修改 input label
void inputOperators(QString)	输入操作符，修改 expression label
void inputSingleOpt(QString)	输入单目操作符，修改 expression label
void numberPush()	输入数字的运算操作（数字进栈）
void operatorPush(char)	输入操作符的运算操作
void singalOptPush(char)	输入弹幕操作符的运算操作
void optPriority(char)	运算符优先级
double calculateExp(double,double,char)	双目运算计算

表 1 属性方法及其功能简介

三、算法介绍

本实验使用的算法为经典的带括号表达式求值算法，算法流程如下：

（1）首先初始化两个栈，一个 double 类型的操作数栈(operands)，一个 char 类型的操作符栈(operators)。并且为了简化之后的实现过程，将'#'压入操作符栈中。

（2）初始化操作符的优先级:

操作符	优先级
#	-1
(	0
+ / -	1
× / ÷	2

表 2 操作符优先级表

（3）若读取到一个数值 num，则直接将其压入操作数栈中。operands.push(num)。

（4）若读取到一个操作符，则分为三种情况：

若是双目操作符（加、减、乘、除），则与操作符栈顶操作符 opt 的优先级进行比较，若 optPriority(opt) > operators.top(),则直接将该操作符压入操作符栈，否则弹出栈顶操作符，并且从操作数栈中弹出两个操作数 num2 和 num1，进行计算 calculateExp(num1,num2,opt)。将计算结果重新压入操作数栈中。循环该步骤，直到 optPriority(opt) > operators.top()，将 opt 压入操作符栈中。

若是左括号，则直接压入操作符栈中。若是右括号，则不断弹出操作符栈中的操作符，与两个操作数栈中的操作数，进行运算，直到弹出的操作符为左括号为止。并将运算结果压入操作数栈。

若是单目运算，则其优先级一定是最高的，所以直接弹出操作数栈的栈顶操作数 num，进行运算 opt(num)。并将运算结果压入操作数栈中。

（5）当所有的操作数和操作符读取完毕后，若操作符栈顶元素不为'#'，则弹出操作符栈栈顶操作符 opt，并从操作数栈中弹出两个操作数 num2 和 num1，进行计算 calculateExp(num1,num2,opt)并将结果压入操作符栈。循环步骤（5）直到操作符栈顶元素为'#'，则此时操作数栈中的数即为表达式运算结果。

四、UML 图

如图 4-1 为计算器用例图，为用户提供的功能主要有：清除（进行下一次运算），退格（当前输入回退一格），操作数输入（输入小数，整数），双目运

算（加、减、乘、除），单目运算（平方，开方，取倒，取相反数）。类图如图 4-2 所示，只有一个 QMainWindow 类，所有功能都在其中实现。

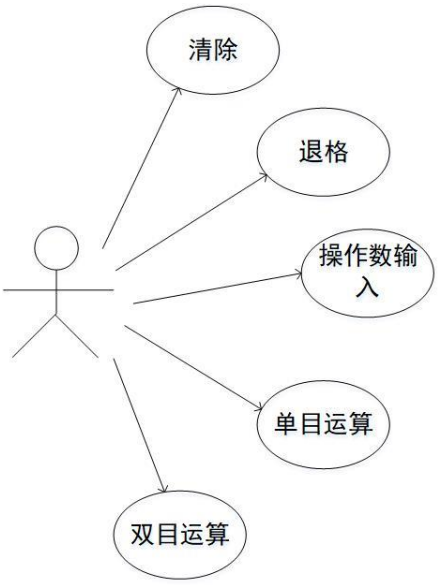

图 4-1 用例图

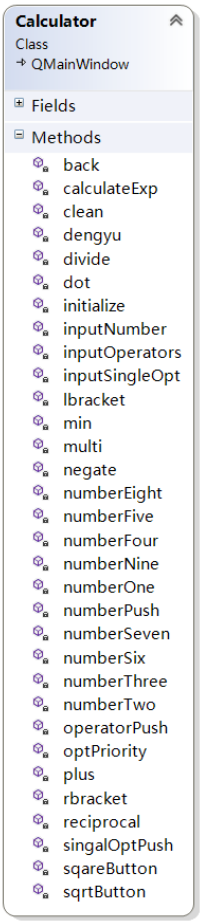

图 4-2 类图

五、信号和槽

1. 信号与槽函数的连接

（1）数字和小数点：

```
connect(ui.zeroButton, SIGNAL(clicked()), this, SLOT(numberZero()));
connect(ui.oneButton, SIGNAL(clicked()), this, SLOT(numberOne()));
connect(ui.twoButton, SIGNAL(clicked()), this, SLOT(numberTwo()));
connect(ui.threeButton, SIGNAL(clicked()), this, SLOT(numberThree()));
connect(ui.fourButton, SIGNAL(clicked()), this, SLOT(numberFour()));
connect(ui.fiveButton, SIGNAL(clicked()), this, SLOT(numberFive()));
connect(ui.sixButton, SIGNAL(clicked()), this, SLOT(numberSix()));
connect(ui.sevenButton, SIGNAL(clicked()), this, SLOT(numberSeven()));
connect(ui.eightButton, SIGNAL(clicked()), this, SLOT(numberEight()));
connect(ui.nineButton, SIGNAL(clicked()), this, SLOT(numberNine()));
connect(ui.dotButton, SIGNAL(clicked()), this, SLOT(dot()));
```

将数字和小数点 button 的 clicked()信号与其对应的槽函数连接，若释放 clicked()信号，则触发相应的槽函数。其槽函数都基本相同。

（2）双目操作符：

```
connect(ui.plusButton, SIGNAL(clicked()), this, SLOT(plus()));
connect(ui.minButton, SIGNAL(clicked()), this, SLOT(min()));
connect(ui.mulButton, SIGNAL(clicked()), this, SLOT(multi()));
connect(ui.divideButton, SIGNAL(clicked()), this, SLOT(divide()));
connect(ui.equalButton, SIGNAL(clicked()), this, SLOT(dengyu()));
```

（3）单目操作符：

```
connect(ui.sqareButton, SIGNAL(clicked()), this, SLOT(sqareButton()));
connect(ui.sqrtButton, SIGNAL(clicked()), this, SLOT(sqrtButton()));
connect(ui.reciprocalButton, SIGNAL(clicked()), this, SLOT(reciprocal()));
connect(ui.negateButton, SIGNAL(clicked()), this, SLOT(negate()));
```

（4）括号：

```
connect(ui.leftParentheseButton, SIGNAL(clicked()), this,SLOT(lbracket()));
connect(ui.rightParentheseButton, SIGNAL(clicked()), this, SLOT(rbracket()));
```

（5）控制 button：

```
connect(ui.CButton, SIGNAL(clicked()), this, SLOT(clean()));
connect(ui.backButton, SIGNAL(clicked()), this, SLOT(back()));
```

2. 槽函数执行功能

槽函数	执行功能
void numberZero()	调用 inputNumber(0)，修改 inputLabel
void numberOne()	调用 inputNumber(1)，修改 inputLabel
void numberTwo()	调用 inputNumber(2)，修改 inputLabel
void numberThree()	调用 inputNumber(3)，修改 inputLabel
void numberFour()	调用 inputNumber(4)，修改 inputLabel
void numberFive()	调用 inputNumber(5)，修改 inputLabel
void numberSix()	调用 inputNumber(6)，修改 inputLabel
void numberSeven()	调用 inputNumber(7)，修改 inputLabel
void numberEight()	调用 inputNumber(8)，修改 inputLabel
void numberNine()	调用 inputNumber(9)，修改 inputLabel
void dot()	修改 inputLabel，添加小数点
void plus()	修改 expressLabel，并进行运算
void min()	修改 expressLabel，并进行运算
void multi()	修改 expressLabel，并进行运算
void divide()	修改 expressLabel，并进行运算
void dengyu()	逐次弹出操作符栈和操作数栈进行运算，直到栈空

void sqareButton()	修改 expressLabel，调用运算函数，进行平方运算
void sqrtButton()	修改 expressLabel，调用运算函数，进行开方运算
void reciprocal()	修改 expressLabel，调用运算函数，进行倒数运算
void negate()	修改 expressLabel，调用运算函数，相反数运算
void lbracket()	修改 expressLabel，添加左括号
void rbracket()	修改 expresslabel，并调用运算函数运算
void clean()	清空所有的栈，初始化变量，初始化 label
void back()	修改 inputLabel，删除字符串结尾元素

表 3 槽函数功能表

六、实验亮点：

本次实验中，自己考虑到了几乎所有可能导致程序奔溃的情况，使得计算器对各种错误情况都能做出正确“回应”。项目有比较强的鲁棒性，具体表现如下：

1、除数为 0 的情况：（如图 6-1 和 6-2 所示）

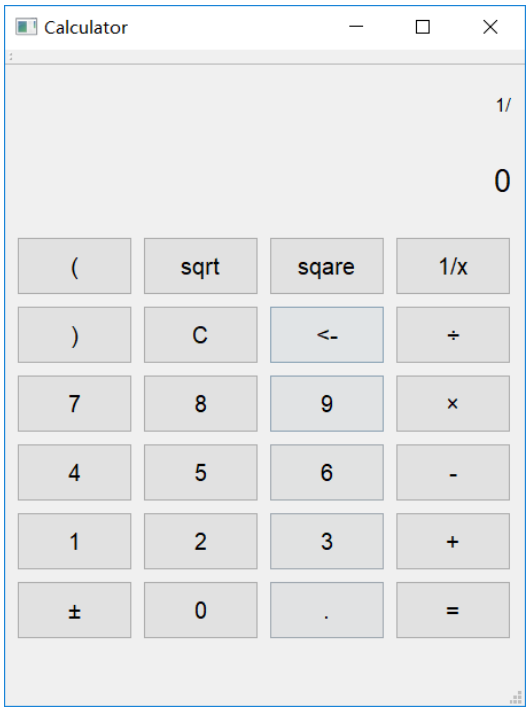

图 6-1

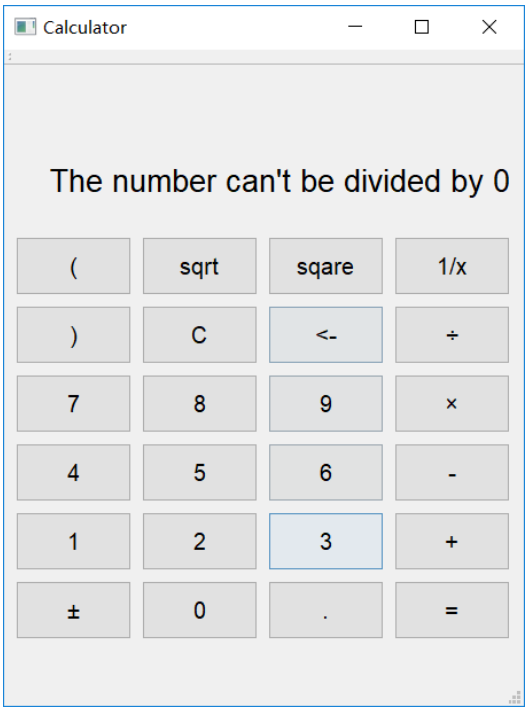

图 6-2

除数为 0 的情况实现比较简单，只需要在每次进行除法运算时候判断一下其除数是否为 0 即可。假如除数为 0，则将 inputLabel 设置为“The number can’t be divided by 0”，将 expressionLabel 设置为空。并且清空操作符栈和操作数栈，并且恢复全局变量的初始值。

2、负数开方情况：（如图 6-3）

负数开方情况与除数为 0 情况类似，也比较简单，只需要在进行开放运算时候判断被开方数是否为 0 即可。若被开放数位 0，则将 inputLabel 设置位“Invalid input”，同时清空操作符栈和操作数栈，并且恢复全局变量的初始值。

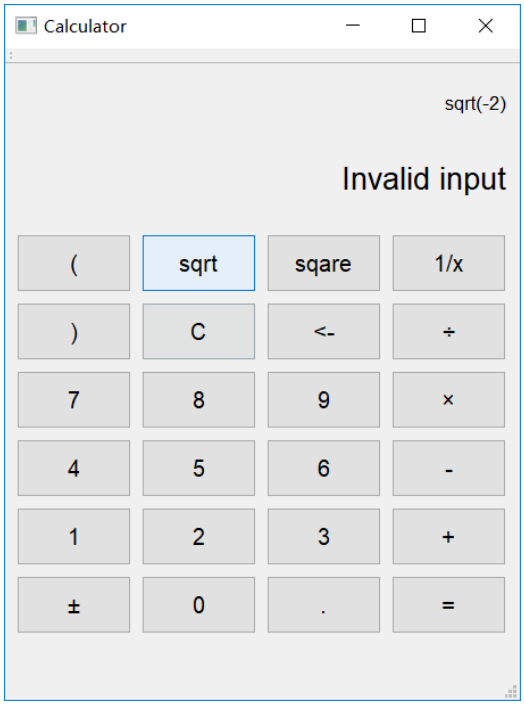

图 6-3

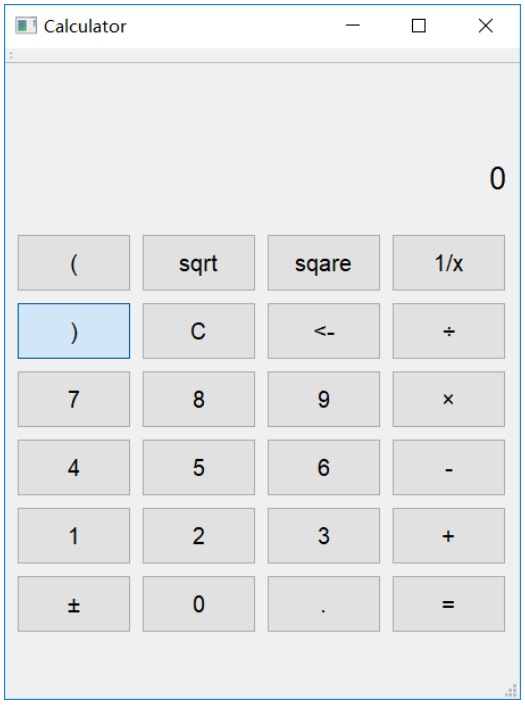

图 6-4

3、没有输入左括号，无法输入右括号（如图 6-4）

如表 1 所示，定义主类的一个 lbracketNum 属性，并且将其初始值设置为 0。每当输入一个左括号时候，执行 lbracketNum++操作将其的值加 1。之后，当输入右括号时，判断 lbracketNum 的值，若 lbracketNum 的值大于等于 1，则可以输入右括号，反之，禁止输入右括号。若可以输入右括号，则修改 expressionLabel 的值，将其末尾添加右括号，并且执行 lbracketNum--，将 lbracketNum 的值减 1。代码如下：

```
void Calculator::rbracket() {
    if ((lBracketNum) && (lastIsRBracket || lastIsNum ||lastIsSinOpt)) {
        inputOperators(")");
        operatorPush(')');
    }
}
```

由条件不难看出，当且仅当 lBracketNum 不为 0，并且上一个操作是右括号或者数字或者但目操作符时候，可以输入右括号。反之，点击右括号无效。

4、输入一个数字后，直接输入左括号，形如：3(（如图 6-5）

此处我的处理方式与 Windows 自带计算器的处理方式是统一的。出入数字后，紧接着输入一个左括号，不会发生错误，而是当下次再输入一个操作符时候，该数字会进栈并显示在左括号后面。

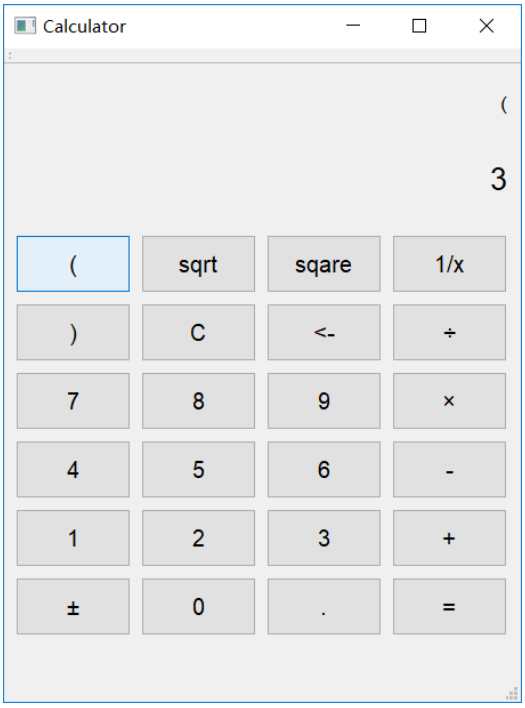

图 6-5

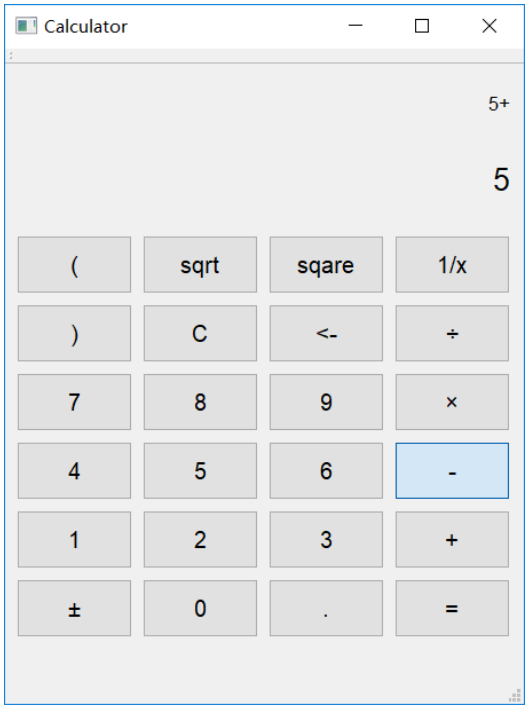

图 6-6

5、连续输入两个双目操作符，第二个操作符将无法输入。（如图 6-6）

见表 1，主类中有三个属性与此有关：lastIsNum,lastIsSinOpt 和 lastIsRBracket。当上一个输入是一个数字时，lastIsNum 为 true，当上一个输入是单操作符时，lastIsOpt 为 true，当上一个输入是右括号时，lastIsBracket 为 true。

分析不难得知，双目操作符仅可能出现在数字，单目操作符或者右括号后面。所以，实现代码如下：（以加法为例）

```
void Calculator::plus() {
    if (lastIsNum || lastIsSinOpt || lastIsRBracket) {
        inputOperators("+");
        operatorPush('+');
    }
}
```

6、在一个数字中输入大于等于 1 个小数点，第二个小数点将无法输入。（如图 6-7 所示）

在输入小数时，若输入了一个小数点后，还能再次输入小数点，例如产生类似 1.234.564 这样的数，这显然是不合法的。所以在输入一个小数点后，必须禁止小数点的输入。我采用的方法是，在每次输入小数点时，都获取 inputLabel 中的 QString，并利用 split 函数将其以“.”分割成数组保存在一个 QStringList

变量中，若该 list 长度不为 1，则说明已经存在一个小数点了，那么禁止小数点的输入，反之可以输入，代码实现如下：

```
void Calculator::dot() {
    QString nu = ui.inputLabel->text();
    QStringList list = nu.split('.');
    if (list.size() == 1) {
        nu.append(".");
        ui.inputLabel->setText(nu);
    }
}
```

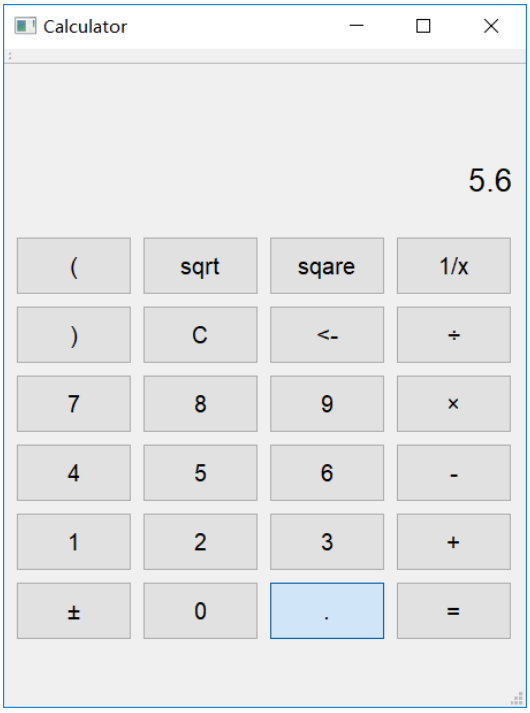

图 6-7

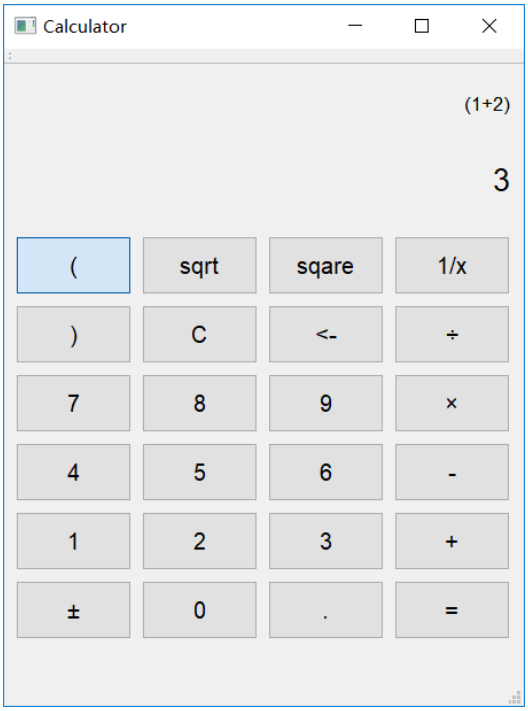

图 6-8

7、右括号存在时，直接输入左括号无法输入，形如：()(。（如图 6-8 所示）

同样地，如表 1 所示，主类中的 lastIsRBracket 属性，lastIsSinOpt 属性，lastIsLBracket 属性都与左括号的输入有关。当且仅当上一个输入不是右括号且不是但操作符或者上一个输入是左括号时候，可以输入左括号。实现代码如下：

```
void Calculator::lbracket() {
    if ((!lastIsRBracket && !lastIsSinOpt) || (lastIsLBracket)) {
        QString exp = ui.expressionLabel->text();
        index = exp.length();
        exp.append("(");
        ui.expressionLabel->setText(exp);
        operatorPush('(');
    }
    lBracketNum++;
```

```
}
```

七、总结讨论

首先感谢孙老师的教导。毫不夸张地说，这已经是我大学以来第三次做带有可视化界面的计算器了。但是这是第一次用 Qt 做该实验。第一次使用的是 VS 的 MFC，第二次使用的是 Java 做的，第三次则是本次。因此，在完成本次实验后，除了编码能力的提升外，我觉得于我而言更重要的是对于不同的语言，不同的开发框架的深入理解与体会。当时用 Java 做的计算器具有更多的功能，包括运算的历史记录，用户自定义变量的使用等等。其界面可以说完全是“画”出来的，没有 Qt 和 MFC 这种便捷式的“拖动”控件形成界面。而 Qt 和 MFC 之间，MFC 的优势在于不需要进行信号与槽函数的连接，自己设计好控件后，双击控件即可自动生成与之对应的代码函数，不需要做更多的声明。Qt 的优势在于，代码逻辑和结构更加清楚，非常符合传统的 C++项目的结构，是经典的面向对象编程结构。MFC 代码更多的是看起来“一团糟”，于我这种新入门的人而言，刚刚接触 MFC 时候是很痛苦的。

另外，不足的是，由于自己在法国，所以耽误了一些课程，导致对于信号与槽的掌握不够熟悉，从而多定义了很多冗余的槽函数，本来可以使代码更简洁。项目完成后才发现这一点，于我而言也是一个经验和教训。