**武汉大学计算机学院**

**本科生课程设计报告**

**网络及分布式计算实验**

**基于Wireshark的网络协议实践**

专 业 名 称 ：软件工程卓越工程师班

课 程 名 称 ：网络及分布式计算实验

指 导 教 师 ：胡继承 教授

学 生 学 号 ：2017302580305

学 生 姓 名 ：祝捷

二○二○年五月

**郑 重 声 明**

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名： 祝捷 日期： 2020年5月14号

摘要

网络及分布式计算实验的实验目的是将所学知识进行综合运用，通过Wireshark软件实验在不同的场景中运行各种网络应用程序并观察其所使用的网络协议，了解它们通过协议进行交互和交换信息的细节。通过这种实战动手，我们不仅观察到细节，更能得到印象深刻的学习。实验设计主要通过Wireshark软件、QT5和Visual Studio，结合上课所学的网络协议知识，深入理解协议的细节，并熟练使用协议传递消息。

在实验项目选择上，我选择完成老师提供的lab0的四个实验，在第一次课的实验里，我熟悉了Wireshark软件的操作，并进行一些简单的分组捕获和观察。有关实验的具体细节请参考实验报告。

……

……

……

**关键词：**Wireshark；网络协议；

**目录**

**1.实验概述**

1.1 实验选题 …………………………………………………………………………………4

1.2 运行环境 …………………………………………………………………………………5

**2.实验一 基于Wireshark的简单分组捕获和观察**

2.1实验准备与背景知识………………………………………………………………………6

2.2实验设计与步骤……………………………………………………………………………9

2.3实验结果与评估…………………………………………………………………………16

2.4难点解决与实验总结……………………………………………………………………18

**3.** **数据库应用程序的设计与实现**

3.1系统架构概述与前端设计 ……………………………………………………………20

3.2功能逻辑实现……………………………………………………………………………21

3.3实现效果与功能…………………………………………………………………………30

3.4难点解决与总结…………………………………………………………………………36

**4实验总结**……………………………………………………………………………………39

**结论** ……………………………………………………………………………………………41

**参考文献** ……………………………………………………………………………………42

**附录** ……………………………………………………………………………………………43

**一、实验概述**

**1.1实验选题**

本实验共有四个选题选择，最后我选择了第一个项目和第三个项目完成，也就是计算器程序和数据库应用程序，它们的项目需求如下：

**项目1：计算器程序**

项目需求：

（1） 实现加、减、乘、除四种基本运算功能

（2） 上述四种运算均要求使用dll实现，其中dll使用c++或c#编程语言来创建

（3） 上述四种运算输入、操作和运算结果均要求直接在界面中进行显示，其中界面可以选用MFC、Winform、WPF 技术实现

（4） 上述四种运算的实现均要求输入数据兼容数据类型int、double，输出结果也兼容数据类型int、double

（5） \*对于输入的非法数据或无效数据要进行相关提示；

（6） \*对于输入的数据有回退删除功能

**项目3：数据库应用程序**

项目需求

（1） 实现增、删、改、查四种业务操作

（2） 上述四种业务的底层操作，使用c++或c#编程语言来实现

（3） 上述四种业务操作，可以自由选定特定的数据库（文件），如Excel、Sqlite、Mysql、Mssql、Oracle。需要自己提供数据库文件或数据库脚本文件进行测试。

（4） 上述四种操作，需要针对至少一个特定的业务数据管理对象，实现输入、操作、显示均直接在界面中进行显示，其中界面可以选用MFC、Winform、WPF技术实现。

（5） \*对界面中输入的非法数据或无效数据或重复数据要进行相关提示。

（6） \*在删除数据时，要给出相应的提示，由用户确认后，再进一步删除。

**1.2运行环境**

本次实验的两个项目主要基于Microsoft公司的Visual Studio 2017进行开发，主要语言是C#（在DLL创建中使用C++），操作系统是Windows10。两个项目分别利用了Winform和WPF两个不同的框架进行开发，最终都完整的实现了实验要求的全部的必做和选做的要求功能。

其中数据库应用程序是基于Postgresql开发的，它的版本号是11.2：

PostgreSQL 11.2, compiled by Visual C++ build 1914, 64-bit

**二、LAB1 - Observe the Network Protocols via Wireshark**

**2.1 Preparation and Background Knowledge**

**2.1.1 Preparation and Software Installation**

In lab1, as it is the first experiment of our course, we mainly glance through the outline of our experiment course, get our computer prepared for the course by installing necessary libraries and software and finish some simple experiments of observing the network protocols via Wireshark “in actions”.

First, I download the Wireshark software and install Npcap in my computer. When I download and install Wireshark, it will automatically lead us to install NPcap as the basic implementation of upper layers instead of WinPcap. According to the introduction on the official website of Npcap, it is the Nmap Project's packet sniffing (and sending) library for Windows, which is based on the discontinued WinPcap library, but with improved speed, portability, security, and efficiency. In particular. This is probably why Wireshark takes this as its default library.

Besides, I also install QT5 and update my Visual Studio 2017 to enable it with more operations on computer networks. But this is not so closely related with this lab so I will put that section in the next chapter.

**2.1.2 Background Knowledge**

In our lectures recently, we learn that end systems exchange messages with each other in a network application. To send a message from a source end system to a destination end system, the source breaks long messages into smaller chunks of data known as packets. Between source and destination, each packet travels through communication links and packet switches (for which there are two predominant types, routers and link-layer switches). Packets are transmitted over each communication link at a rate equal to the full transmission rate of the link.

Therefore, observing (or sniffing) the packets seems to be an efficient way to learn more about the protocols and networks. In our lab course, we use Wireshark to capture these packets in a passive process, that is, it will only observe the packets through the applications on our computer with no impact on the transmission. And this is implemented with the packet capture library which is Npcap in Wireshark nowadays and WinPcap before.

In addition, the “packets” captured could be called in a more accurate way as “frames” or “link-layer packets” because the sniffer (Wireshark) will receive a copy of every link-layer frame that is sent from or received by your computer. And they are handled by different link-layer protocols at different links. (So the implementation of the packet capture library Npcap should take different strategies on different devices to capture frames?).

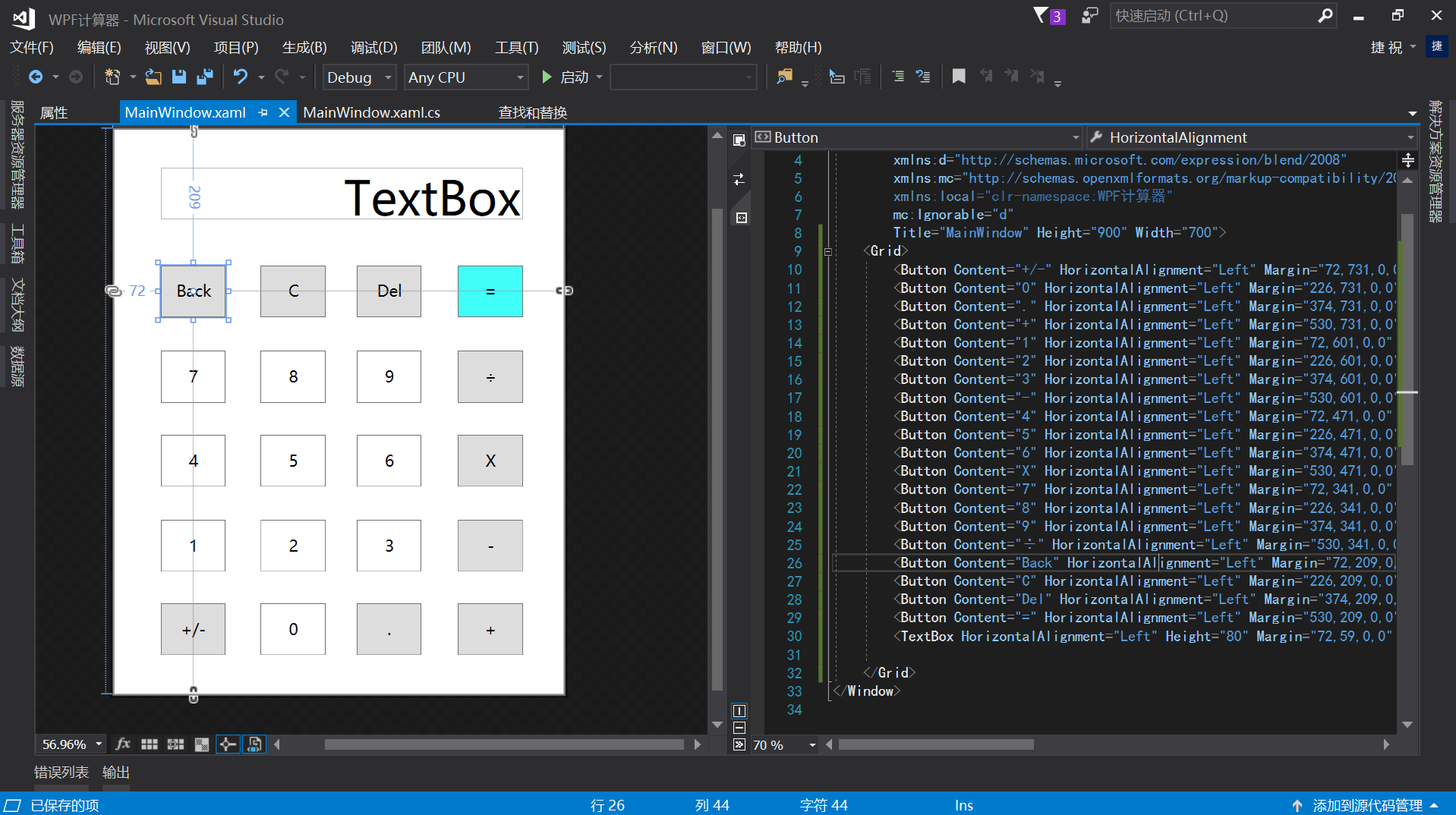


图2.1 计算器前端开发效果图

**2.2实验设计与步骤**

**2.2.1 DLL的创建和调用**

下面对于创建DLL的过程进行具体描述：

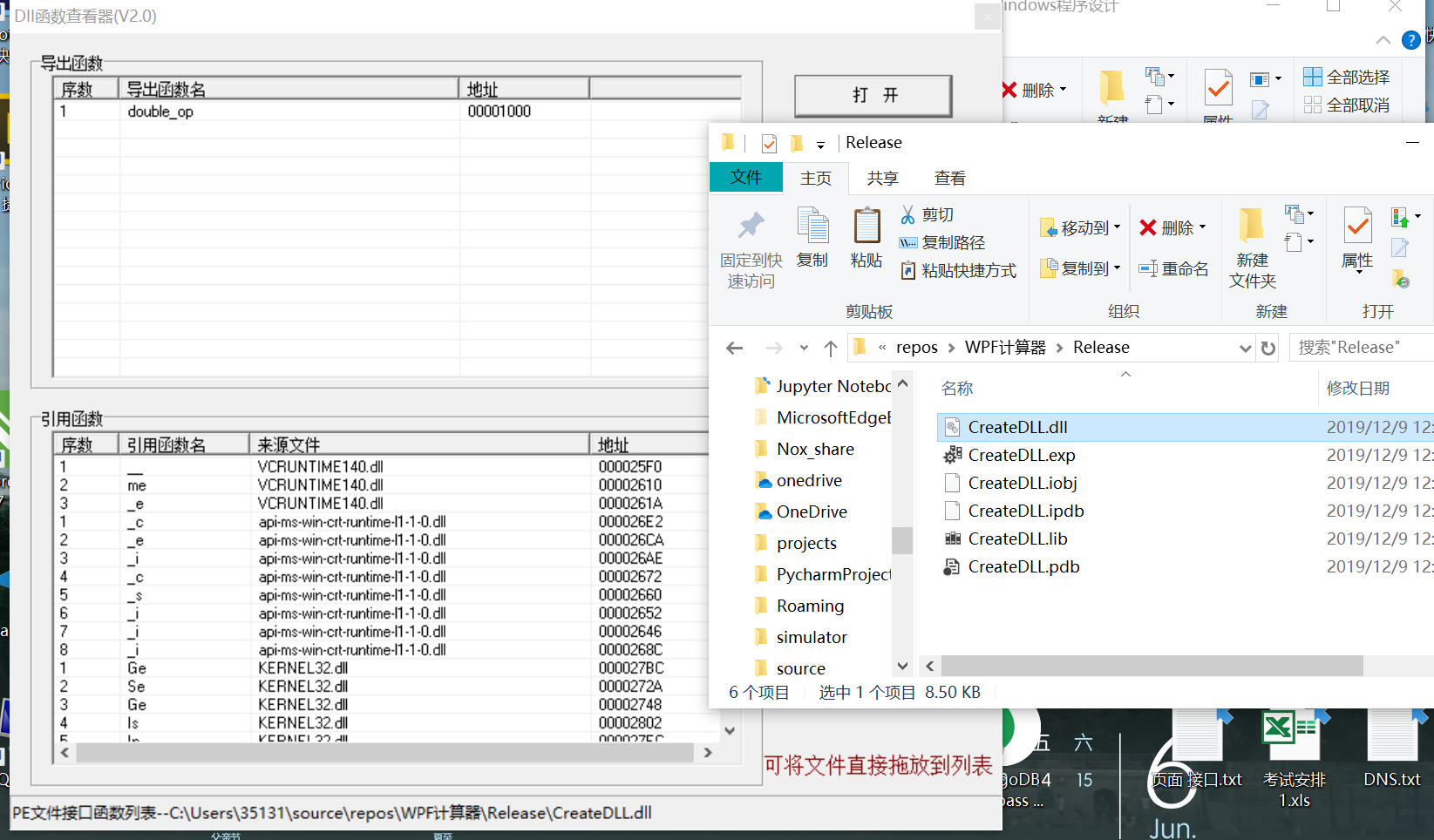


图2.2.3 DLL函数查看器中查看

至此，我们的DLL创建过程就完成了，下面将在我们的计算器WPF程序中完成对它的调用来实现计算器的主要功能。

**2.2.2 DLL的调用**

在计算器程序中，我将程序分为了三层，分别是页面交互层，业务处理层和底端实现层，同时进行了相应的异常处理，因此我调用DLL的位置主要在我们的业务处理层，也就是Calculator.cs文件中。

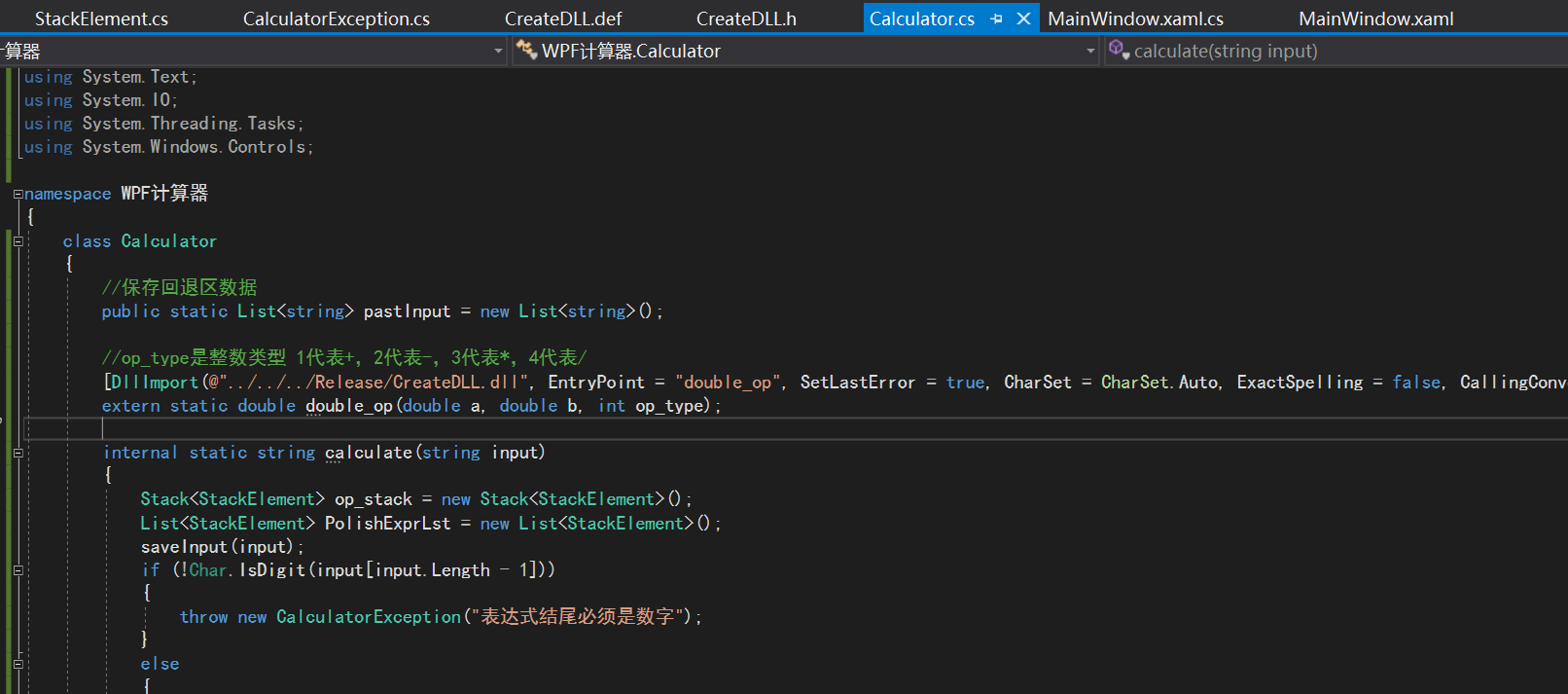


图2.2.4 DLL的调用

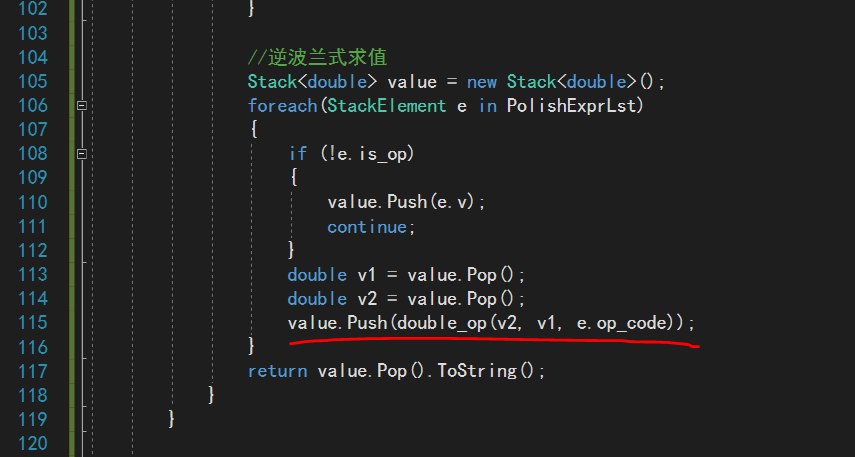


图2.2.5 使用DLL定义的函数实现四种运算

**2.2.3 表达式求值的逆波兰算法流程说明**

算术表达式的求值主要是基于中缀表达式转逆波兰式，以及逆波兰式求值的两个算法步骤完成的，下面是他们的核心代码的截图：

这个部分的代码都在Calculator，也就是我们的业务处理层中进行的，通过交互层的MainWindow触发点击事件，调用Calculator类的函数进行处理，最后使用DLL中定义的函数进行运算。

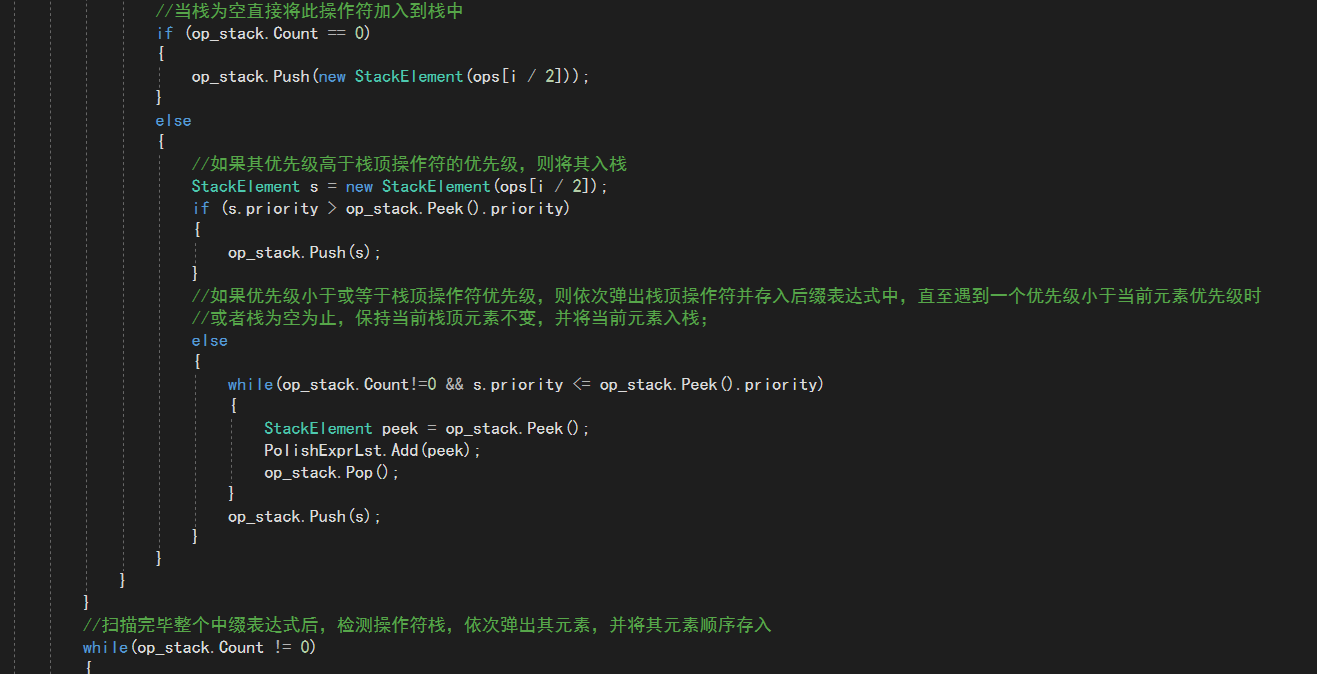


图2.2.6 中缀表达式转后缀表达式（逆波兰式）



图2.2.7 逆波兰式求值（调用DLL定义的double\_op运算）

**2.2.4 回退区功能的实现**

回退区功能的实现时我考虑到了实际使用的需求，综合思考后认为主要有以下三种情况需要用到回退区：

1. 删除Del操作之后
2. 清空Clear操作之后
3. 运算=（计算结果）操作之后

因此，我在设计Calculator类的时候就考虑了这个问题，设计了一个List专门存储这三种操作前的结果，如果需要回退Back就调取最近一次的缓存结果，显示到前端，并把该缓存从List中删去，并且我还设计了当List长度大于size=100时就会自动删去List第一个的元素，从而保证运行内存不会占用太多。

Calculator.cs类中定义静态变量

//保存回退区数据

public static List<string> pastInput = new List<string>();

//回退区最大空间

public static int size = 100;

以下是回退区save和back的两个封装函数

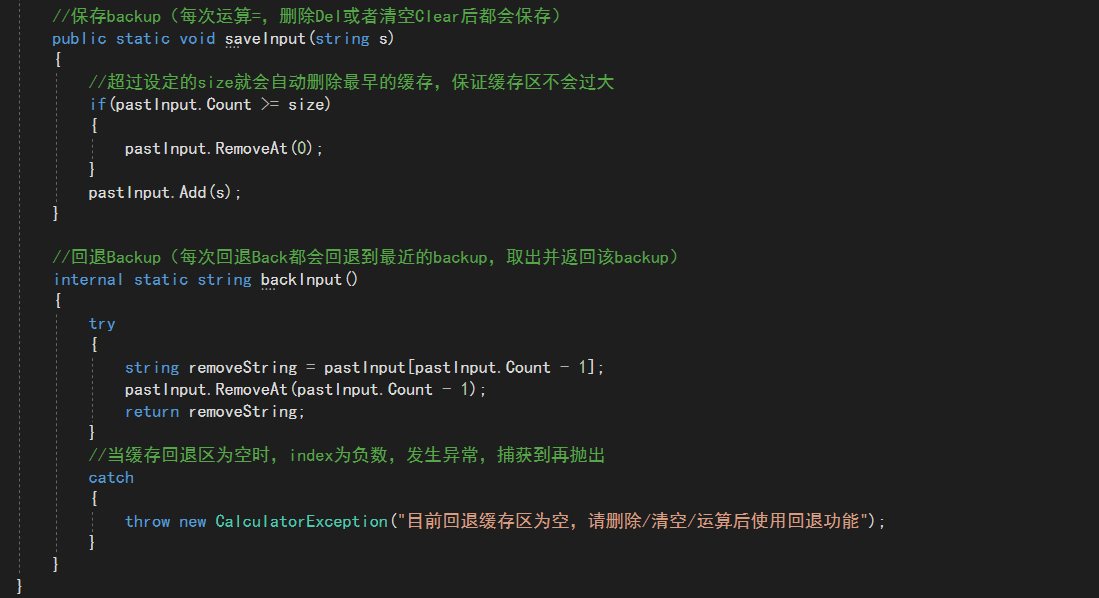


图2.2.8回退区save和back的实现

**2.2.5 异常处理、无效数据信息提示与报错**

我定义了一个自定义的异常类CalculatorException用于捕捉一部分在Calculator类中调用Calculate发现的不合理输入的错误，另外在页面交互层也就是MainWindow中的button点击事件处理函数我也以MessageBox.Show对不合理/无效的输入进行了提示，具体效果请见报告的 2.3效果展示。

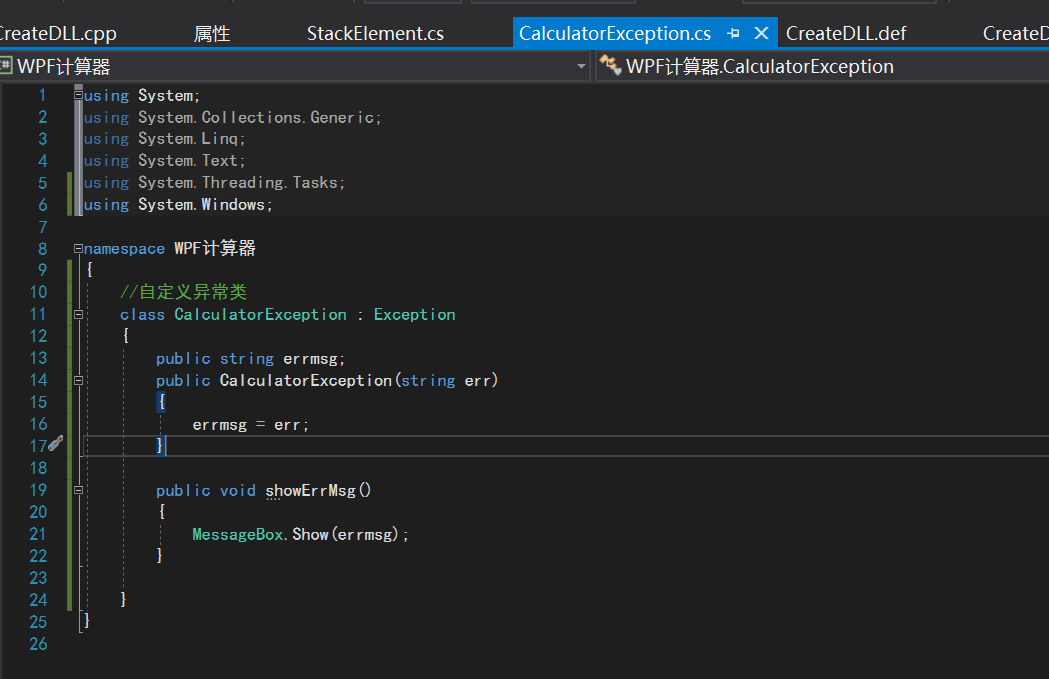


图2.2.9 自定义异常类CalculatorException



图2.2.10 自定义异常的捕获

具体说来，无效/错误信息的提示的可能情况主要包括：

1.不合理的正负号使用

2.同数字出现多个小数点

3.运算符的不合理使用

4.表达式结尾不是数字

5.回退缓存区为空的时候回退

6.错误的输入数字形式

7.除0错误

实现上，一些情况并不会报错，比如什么都不输入就去Del删除，那么我参考了微软的计算器设计，直接把TextBox值归0，总体上情况基本都考虑的比较完全，对于int/double类型的处理我是采取了C#直接转换的方式，输出结果依然能保持小数和整数的形式，对于除0错误等逻辑错误也予以一并纳入考虑，在Caculator中采取CaculatorException自定义异常报错。

对于更多的测试例子和具体效果，请见下一节2.3运行效果

**2.3实验结果与评估**

计算器程序启动时，输入框编辑区会自动设定为0，根据实验指导要求，我采取C++创建DLL，并在WPF程序中调用实现了数值加减乘除计算的过程

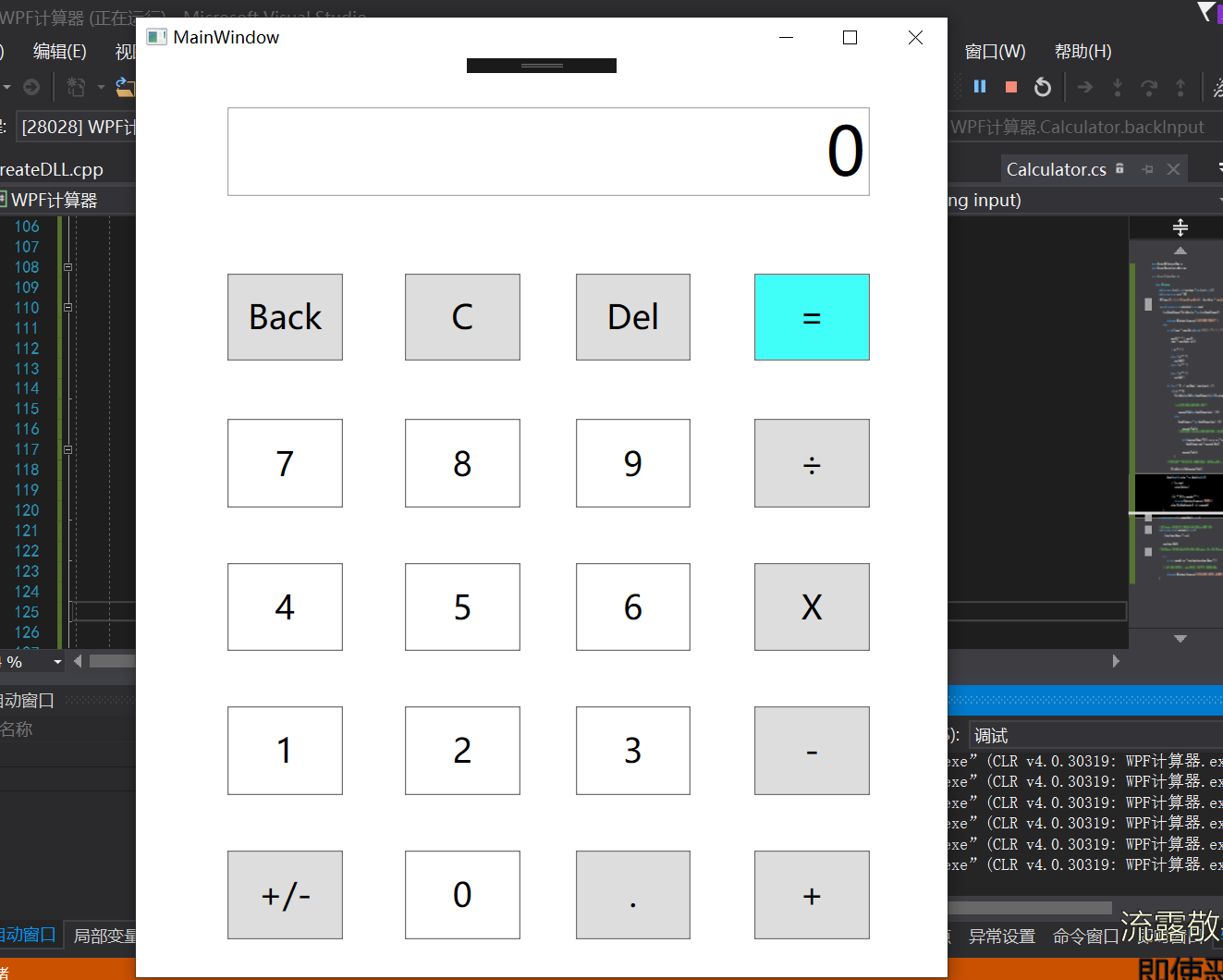


图2.3.1 启动效果

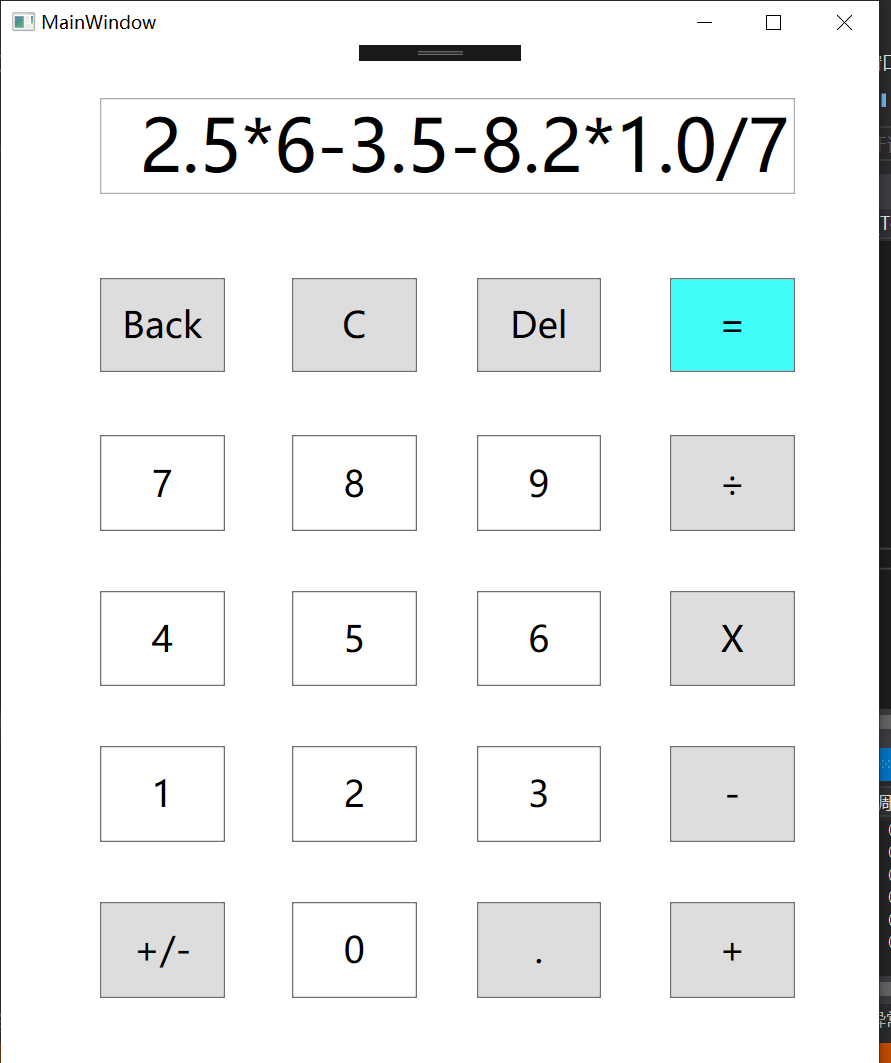


图2.3.2 输入任意一个表达式

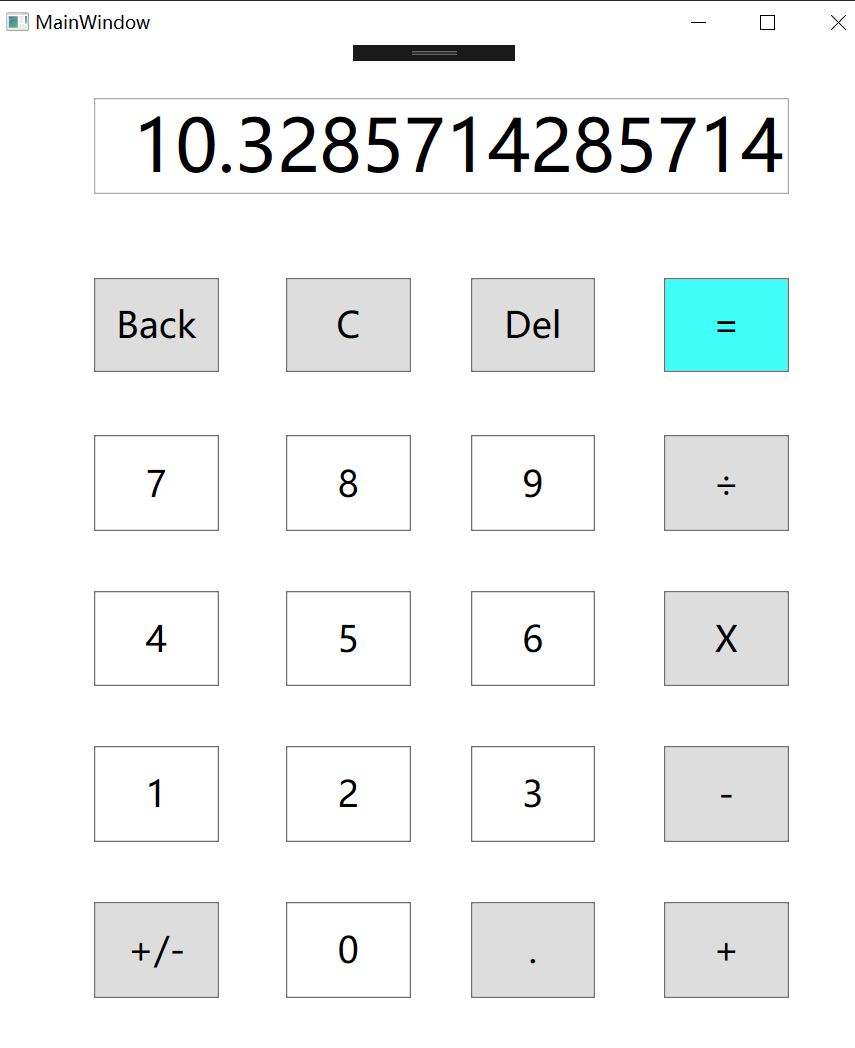
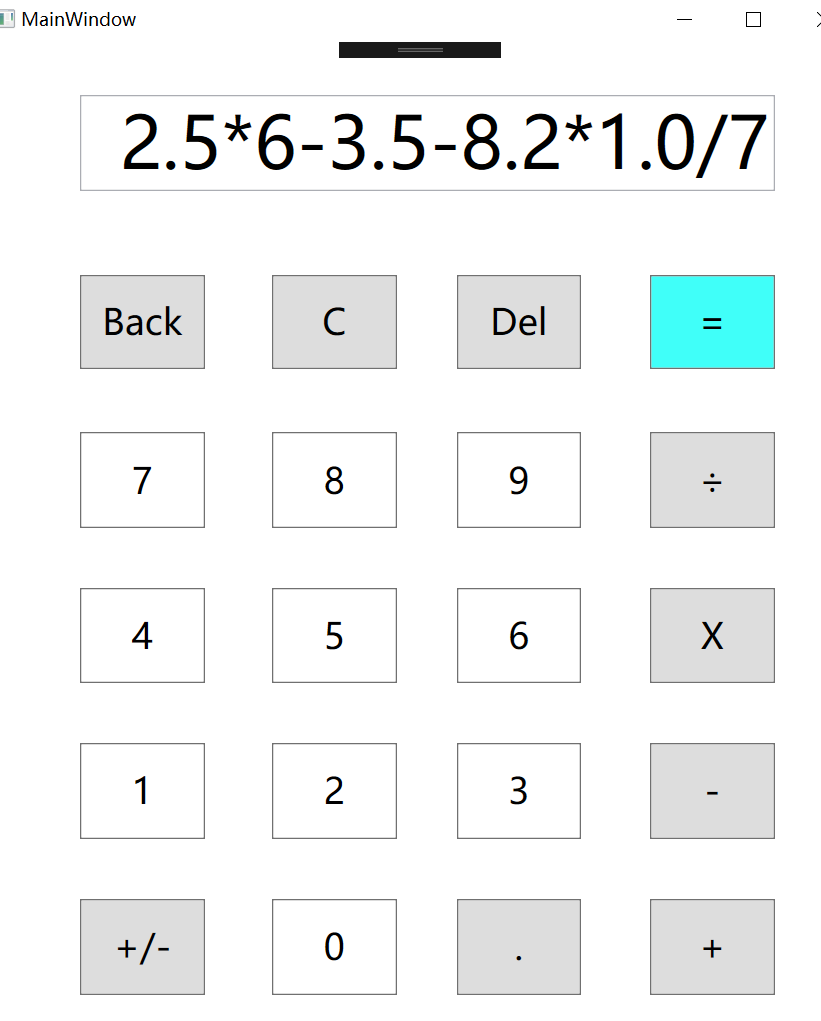
 

图2.3.3点击=计算结果 图2.3.4点击Back回退

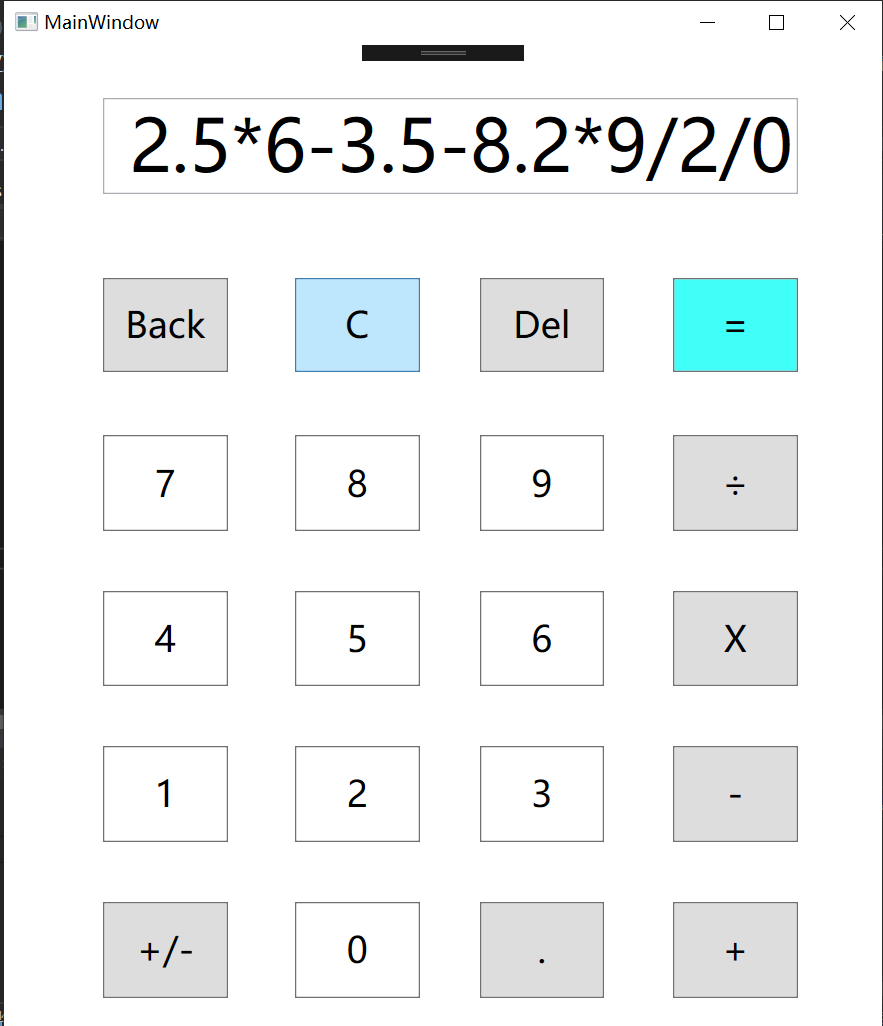
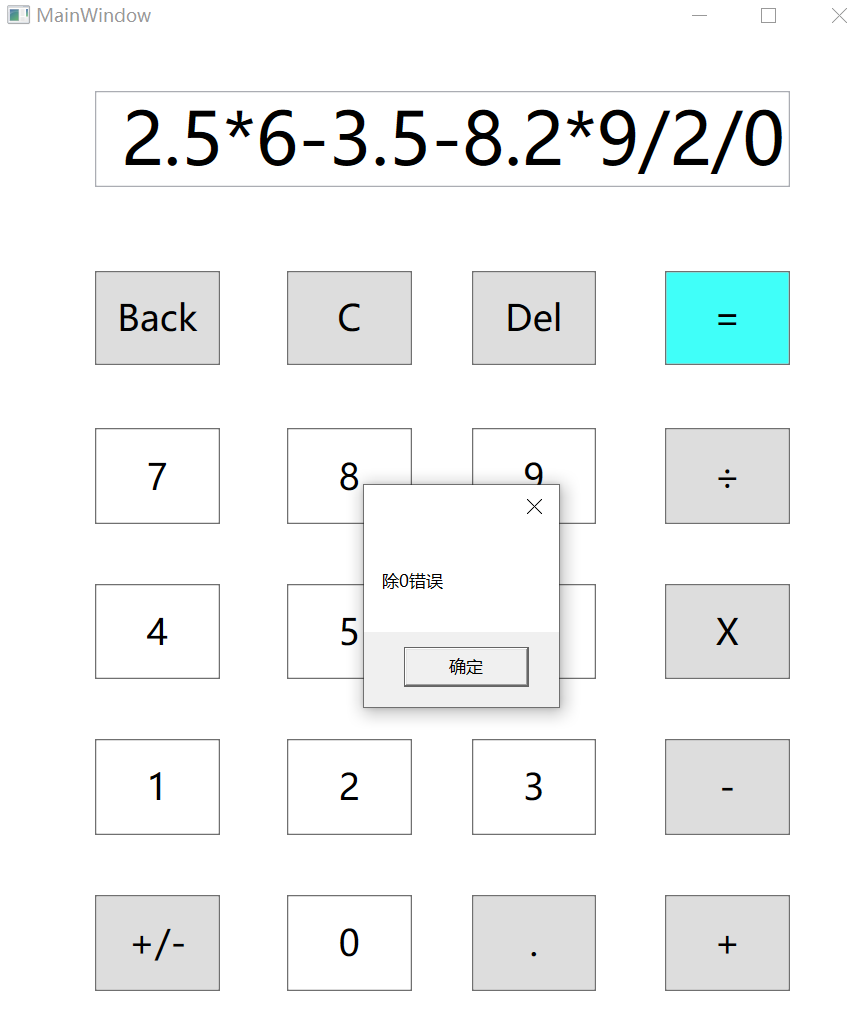
 

图2.3.5 Del删除几个字符后又输入 图2.3.6 除0错误检测

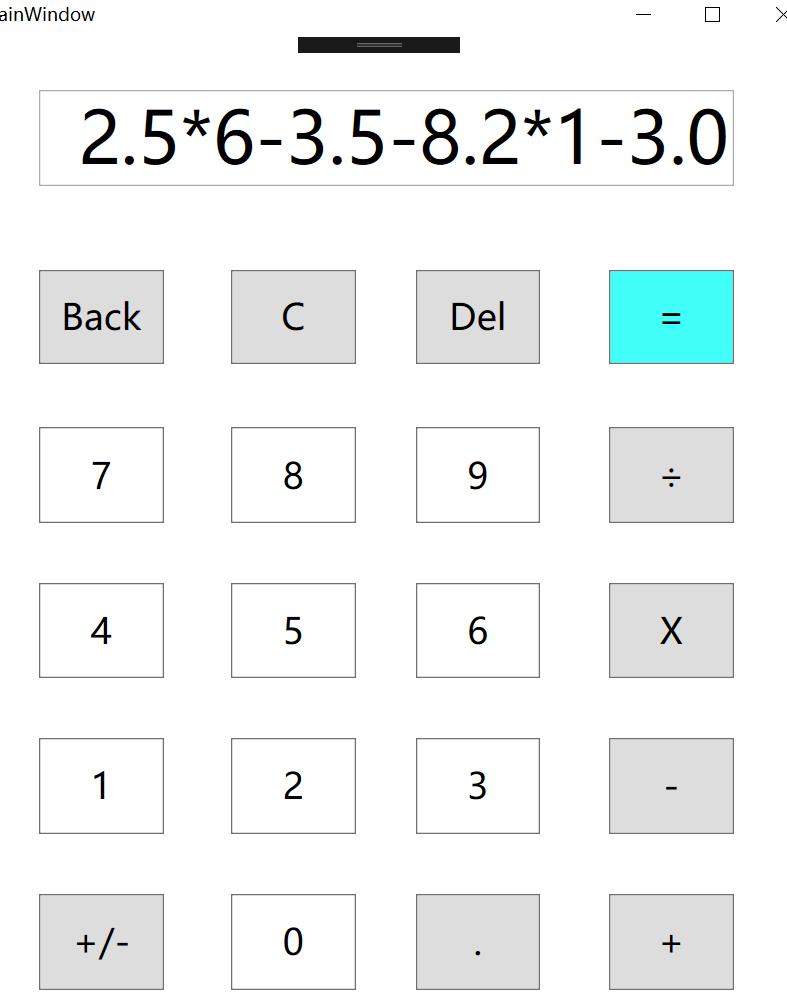
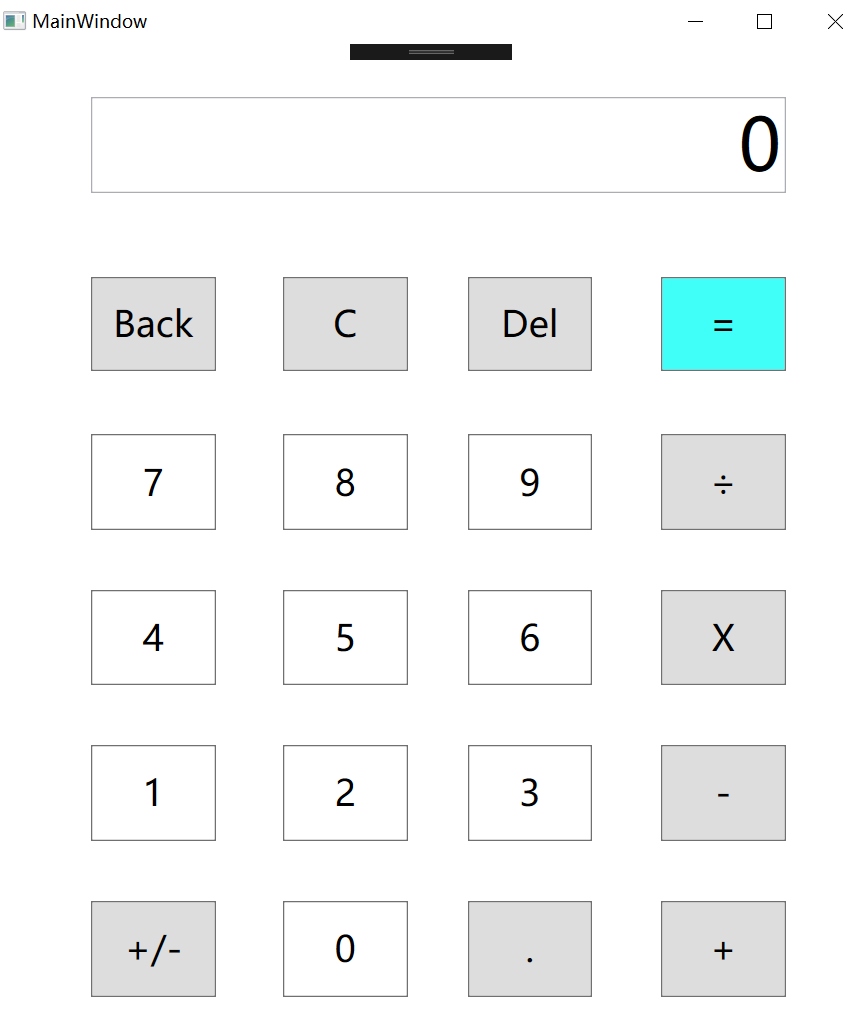
 

图2.3.7 Back回退 图2.3.8 Clear清空

**2.4 难点解决与实验总结**

总体来说，我的计算器程序**实现了要求的选做必做全部功能**，四种运算均使用C++创建的dll实现，并在WPF程序中调用，输入、操作和运算结果均直接在界面中进行显示，输入输出兼容int和double。选做功能：对于输入的非法数据或无效数据要进行相关提示，对于输入的数据有回退删除功能，两个选做功能也均已充分完成。

下面结合编写程序过程中遇到的难点谈一谈具体实现中自己是如何解决的，以及一些对于计算器程序的小结补充：

**1.采取合适的数据结构配合封装的方法来实现回退功能**

我把回退的功能浓缩在save和back的两个封装函数和一个数据结构List<string>中，当我们进行了一些可能需要回退的操作比如Del,C,Back，我们就会调用save自动保存信息到回退区，当需要Back回退时，就调取最近一次的缓存结果，显示到前端，并把该缓存从List中删去。

考虑到del，=运算等等操作可能是经常会出现的，那么如果List过长，最后可能会出现内存占用过大的情况，降低程序运行效率，因此就调取最近一次的缓存结果，显示到前端，并把该缓存从List中删去，100个缓存区的size既能满足日常使用的需要，也不至于会造成程序运行效率过低的影响。

目前的遗憾是，因为时间有限，没有实现一个可视化的列表进行展示选择（只可以按back往回回退），也没有采取增量更新的方法（提高时空运行效率）

目前的优点是，实现使用的回退效果良好，充分满足要求，封装在两个函数内也增强了代码的可读性和可维护性，有利于后期继续改进功能。

**2.自定义CalculatorException异常类对错误信息进行提示**

我自定义了一个CalculatorException异常类对于Calculator类中出现的异常信息进行了更好的封装，有利于后期实现系统架构的分层，更好的体现了面向对象职责分配的思想。

**3.系统架构清晰、封装程度高**

计算器程序主要分为三个层面，用户交互层MainWindow，业务逻辑层Calculator，底层模型层。其中交互层将绝大多数的输入都集中到了add\_Something\_To\_Input中进行处理；业务逻辑层封装了save/back方法，并采取逆波兰式算法实现了不同优先级的表达式求值，尽可能减少了代码的复杂度和耦合度，整体架构清晰；最后在calculate的逆波兰式求值计算中调用了DLL的double\_op方法，根据不同的op\_code进行相应计算并返回结果。

**三、数据库应用程序的设计与实现**

**3.1系统架构概述与前端设计**

**3.1.1 系统总体框架综述**

本次实验的数据库应用程序主要采用了基于微软的Winform并以C#为主要语言进行开发，数据库采取的是PostgreSQL 11.2。然后我的数据库程序的增删改查全部采取DataGridView的可视化实现，页面提供了直观的文件目录树形式的数据库结构的显示，很好的实现了对于PostgreSQL中存储的数据的访问与修改，还对异常输入和不正确的表达式进行了提示，在删除数据时给出相应的提示，由用户确认后，再进一步删除，完整实现了要求的必做选做的全部功能。

**3.1.2 系统前端设计与框架**

本次实验的解释器系统前端主要采用了基于微软的Winform框架以C#为主要语言的开发模式，设计上提供了包括数据库登陆页面以及主页面，和分页面进行具体的数据读写和结构的管理：

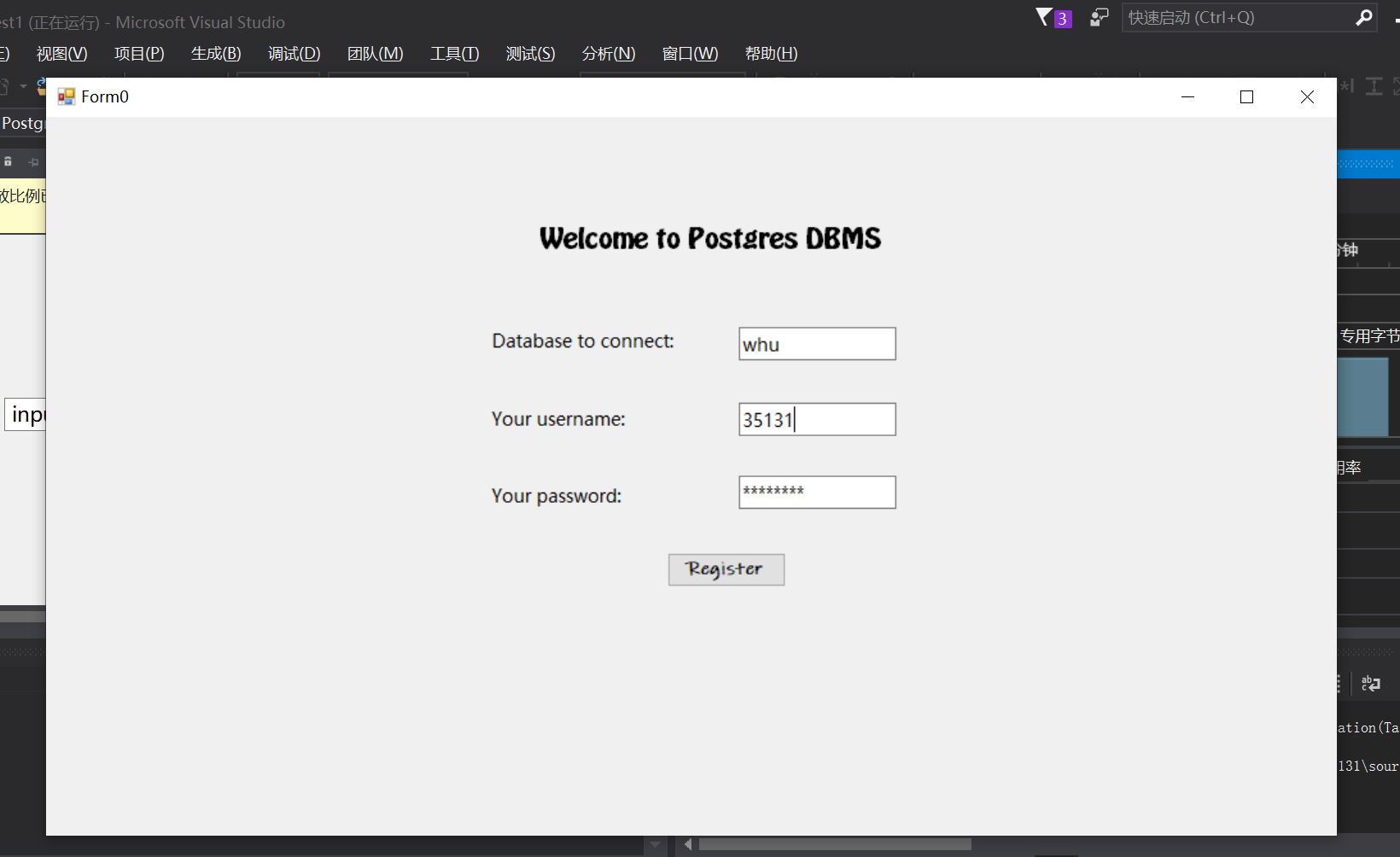


图3.1 数据库程序首页登录页图

**3.2功能逻辑实现**

**3.2.1 数据库连接与登录**

下面对于数据库程序连接数据库，登录的过程进行具体描述：

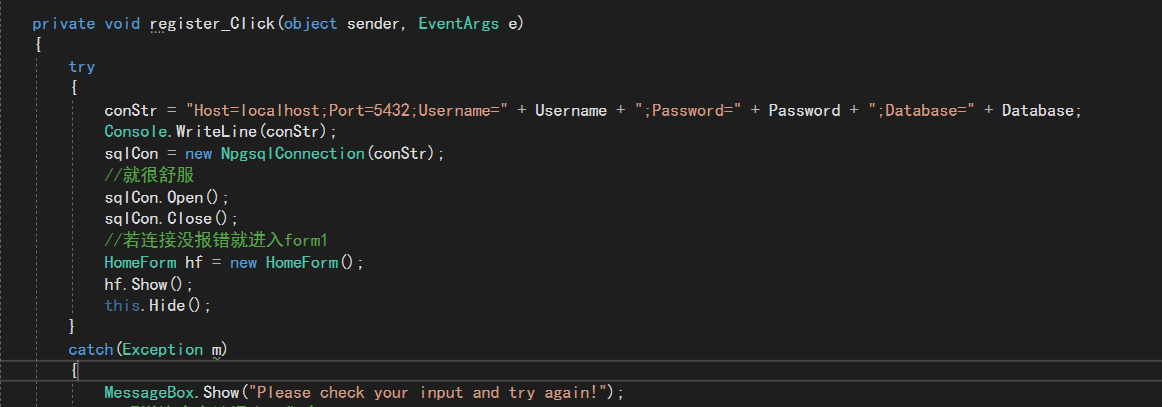


图3.2.1登录部分的代码

可以看到，这里如果登陆失败，会提示用户检查输入，因为数据库连接失败会抛出异常，所以我直接利用这一点写try catch捕获，再提示用户重新输入。

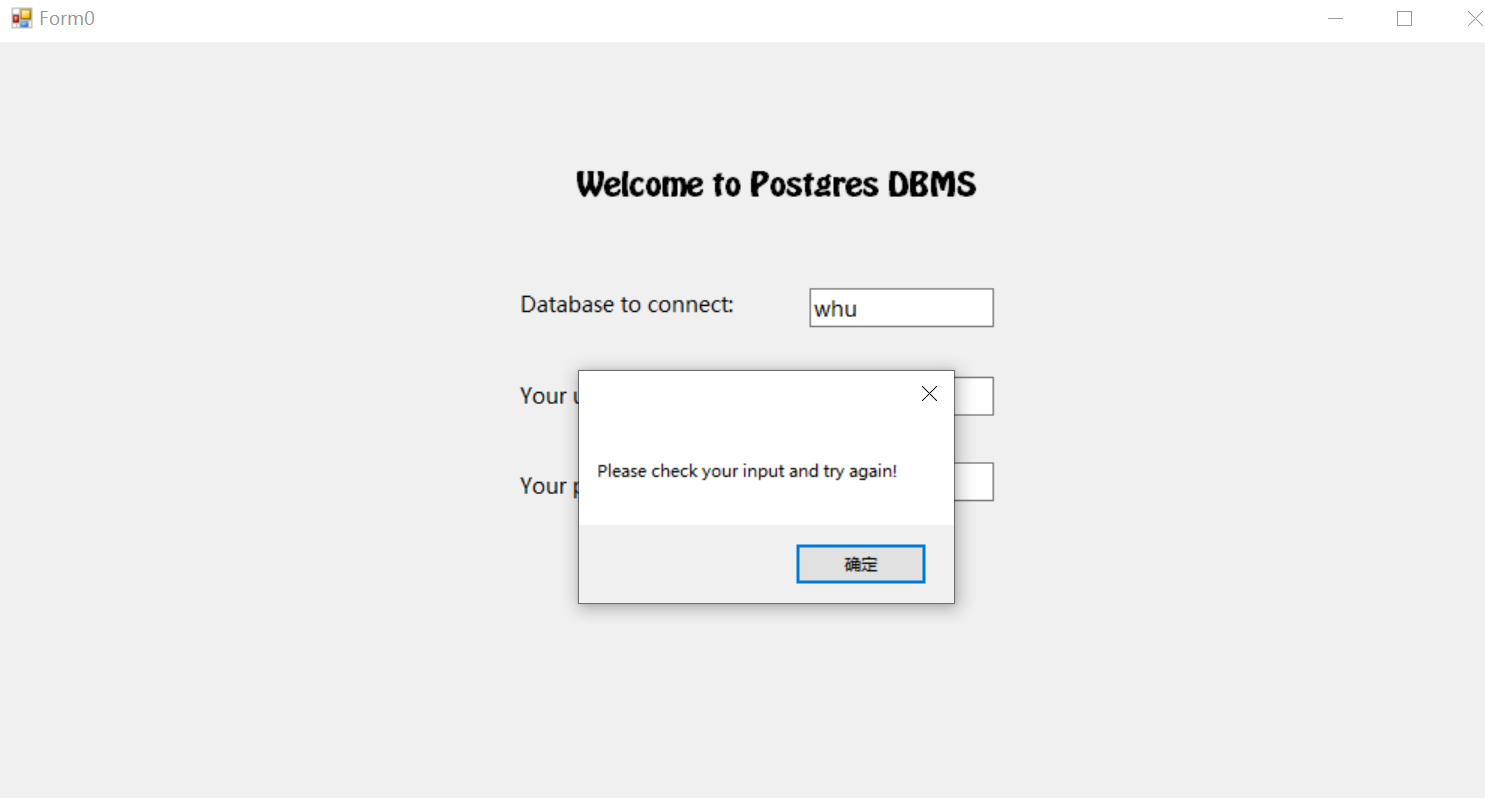


图3.2.2 登陆失败提示（用户名/密码错误）

具体的登录是通过Nuget安装Npgsql提供的接口实现。

**3.2.2 首页数据库表的目录树结构**

在数据库程序中，登陆后，会自动跳转到HomeForm也就是主页面中，左侧有Winform提供的TreeView组件，来实现文件目录树的效果，这里我用来显示数据库中的表结构。

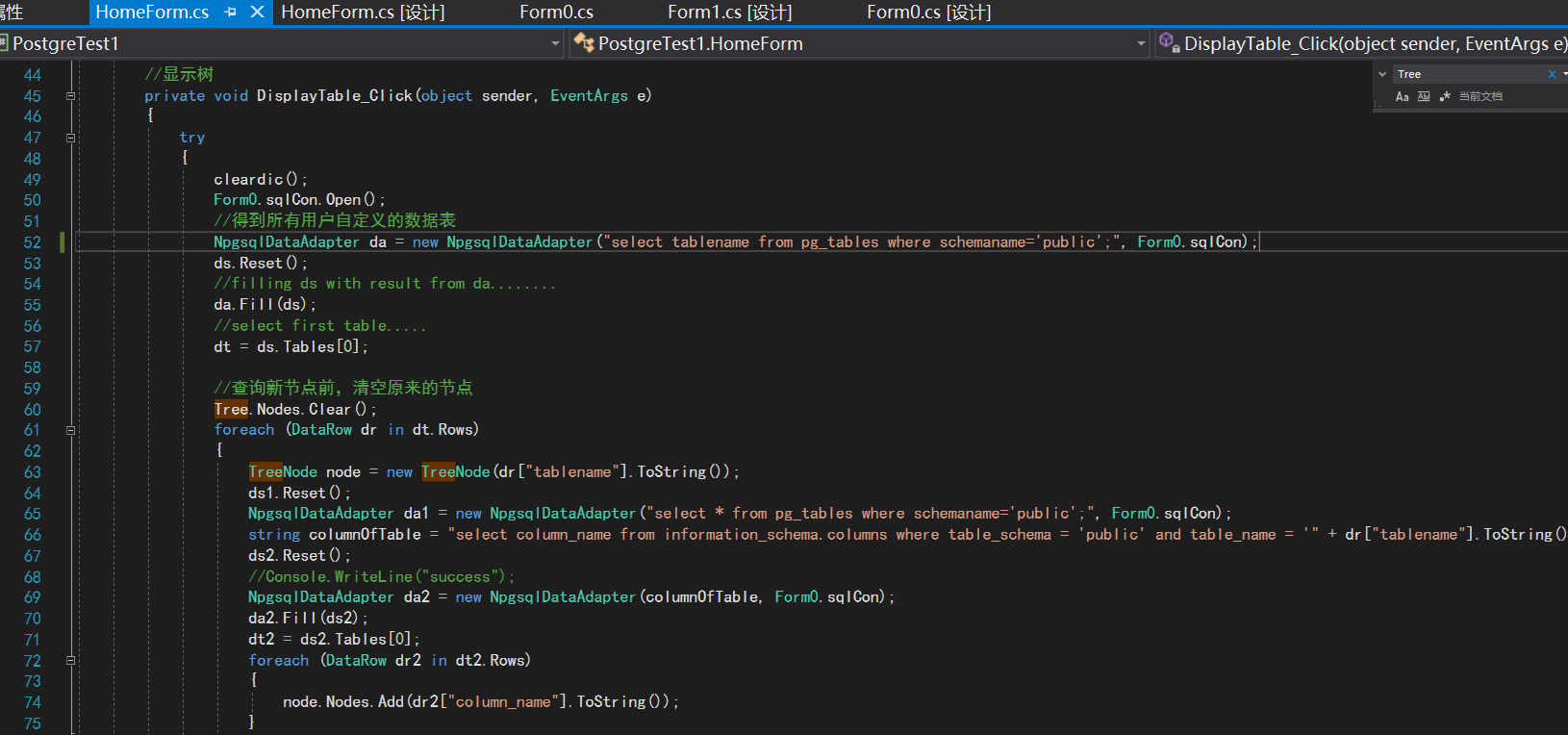


图3.2.3 显示文件目录树的代码

这里采取PostgreSQL提供的select tablename from pg\_tables where schemaname='public';查找到当前用户的全部数据表，再递归查找到对应的columnName，最后以文件目录树的形式显示在TreeView组件中。

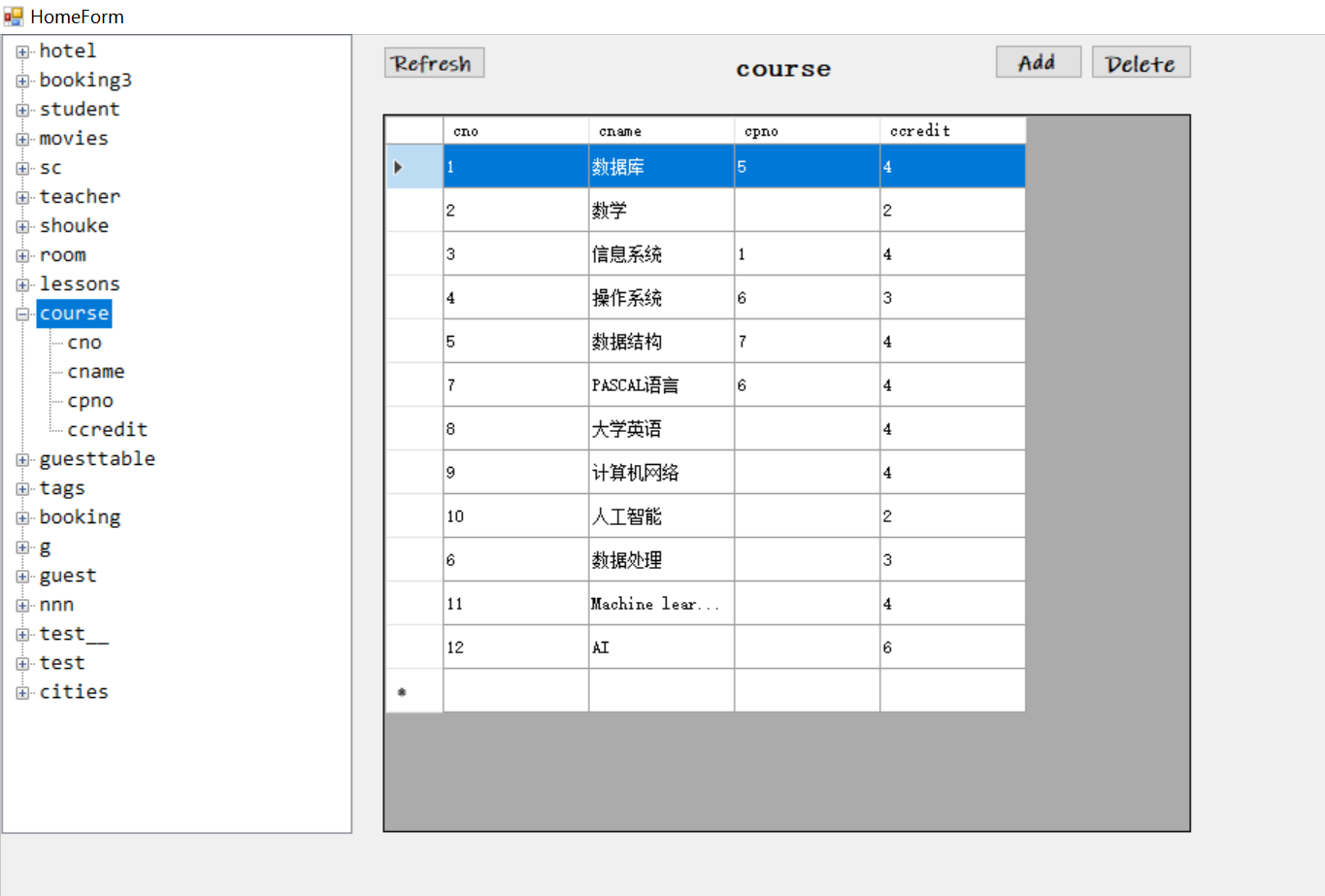


图3.2.4 显示文件目录树

**3.2.3 目录树节点右键展开菜单栏**

为了提供更丰富的功能，我在数据库程序的数据库表目录树的右键点击触发函数中增加了菜单栏的设计，采取组件是contextMenuStrip，右键点击数据表table节点，与column节点根据一级二级的关系分别打开不同的contextMenuStrip，并提供不同的功能：



图3.2.5 使用contextMenuStrip右键展开菜单栏

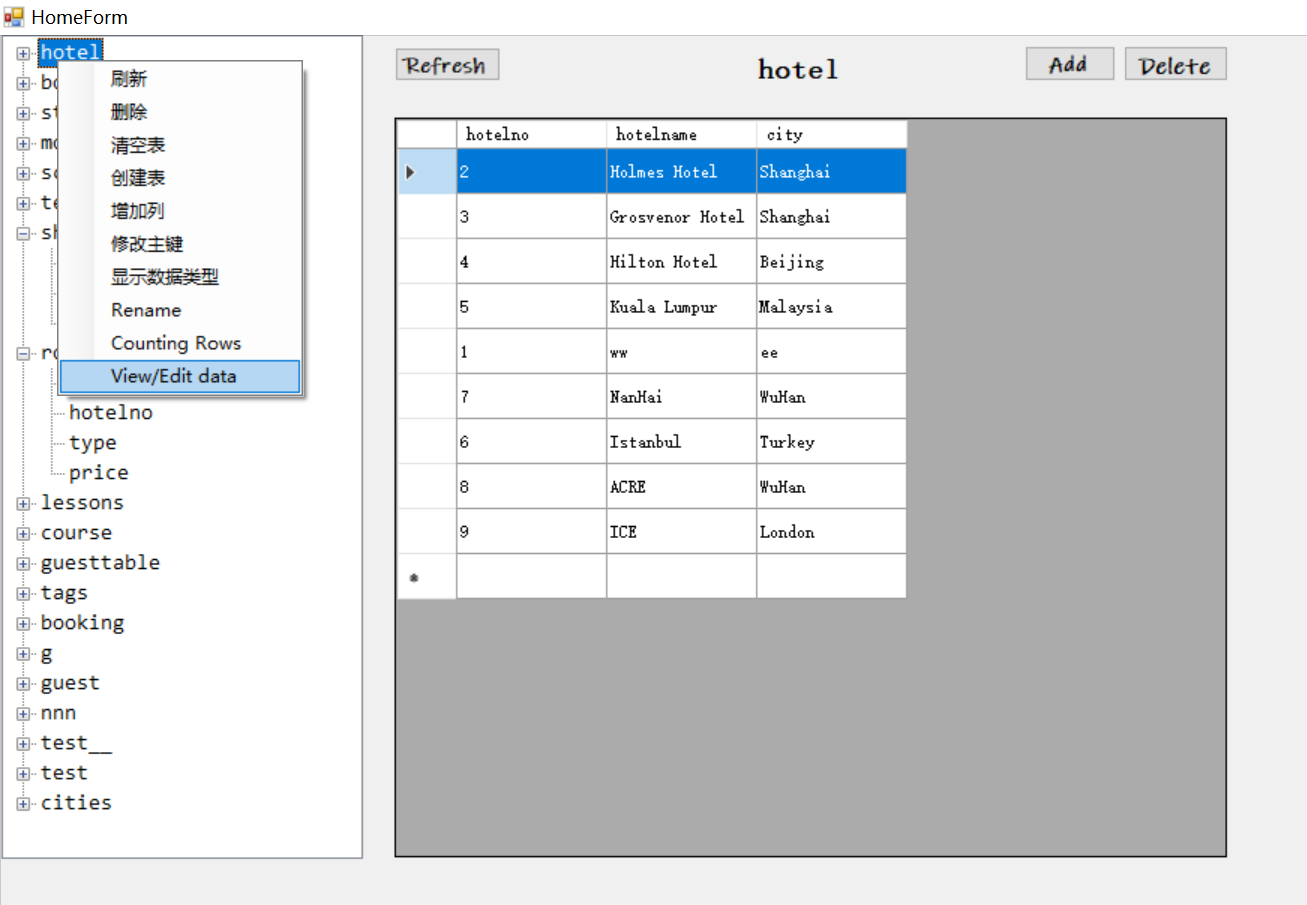


图3.2.6 菜单栏效果

**3.2.4 数据表的显示与修改（DataGridView）**

数据表在DataGridView中显示的实现比较简单：

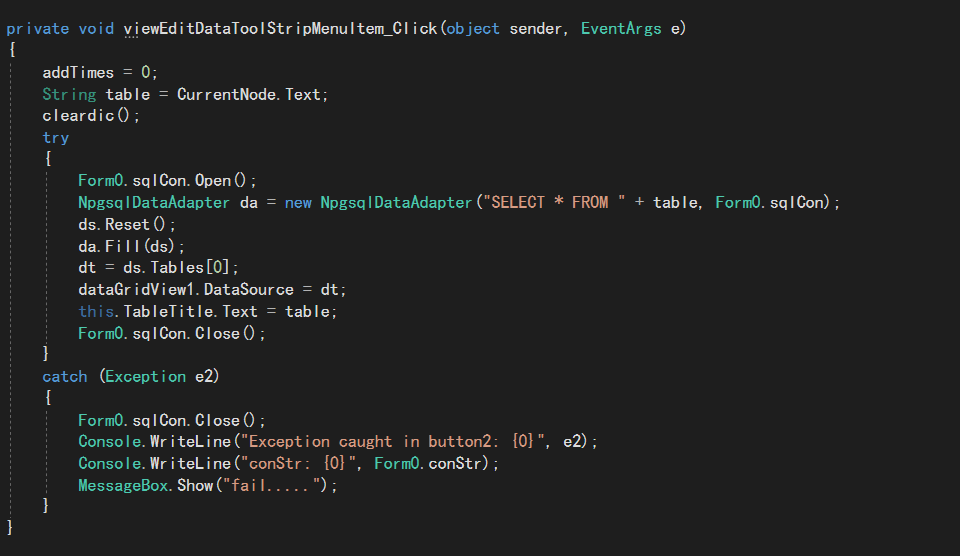


图3.2.7 在DataGridView填充datasource显示数据表

这里采取了类似我们Windows课程上讲的采取DataAdapter，DataSet以及DataTable等结构从数据库中将数据转化成DataGridView可以处理的数据形式，并提供其展现出来。效果如下图：

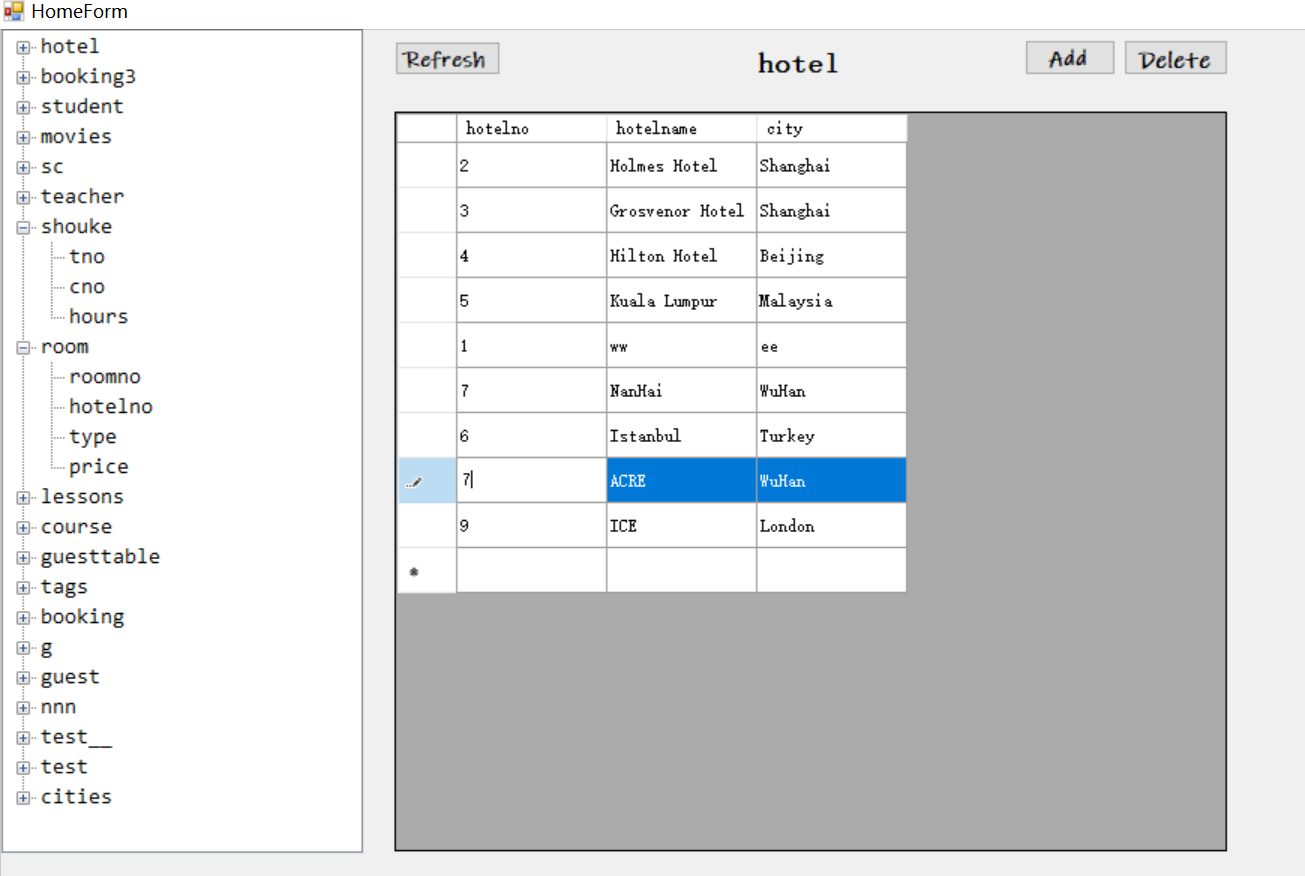


图3.2.8 数据库表的显示

当然，上方的图其实还引出了另一个问题，如何在dataGridView中直接可视化的形式来修改数据呢？

这里引出了一系列很复杂的考量，包括比如主键的考虑，你怎么知道你要改的那一行是唯一的，怎么保证修改的唯一性；又比如说添加某一行，万一有空值，那么怎么知道这个空值是否可以为空，因为数据库的表可能有些字段不能为空；又比如说修改的时候，假如有的字段是char/varchar字符型，有的就是数值型，那么update xxx set xxx = 的时候可能需要考虑到需不需要加引号，因为字符型必然需要引号，而数值型不需要，但是dataGridView触发事件的时候对此一无所知，所以这些都是我们必须要去考虑到的问题。

这一块将放在第四部分3.4难点与总结部分具体描述，这里只是带一笔，下面是关于cellEdit的事件触发的处理函数的一部分，这里对于update/add等等都直接相关。

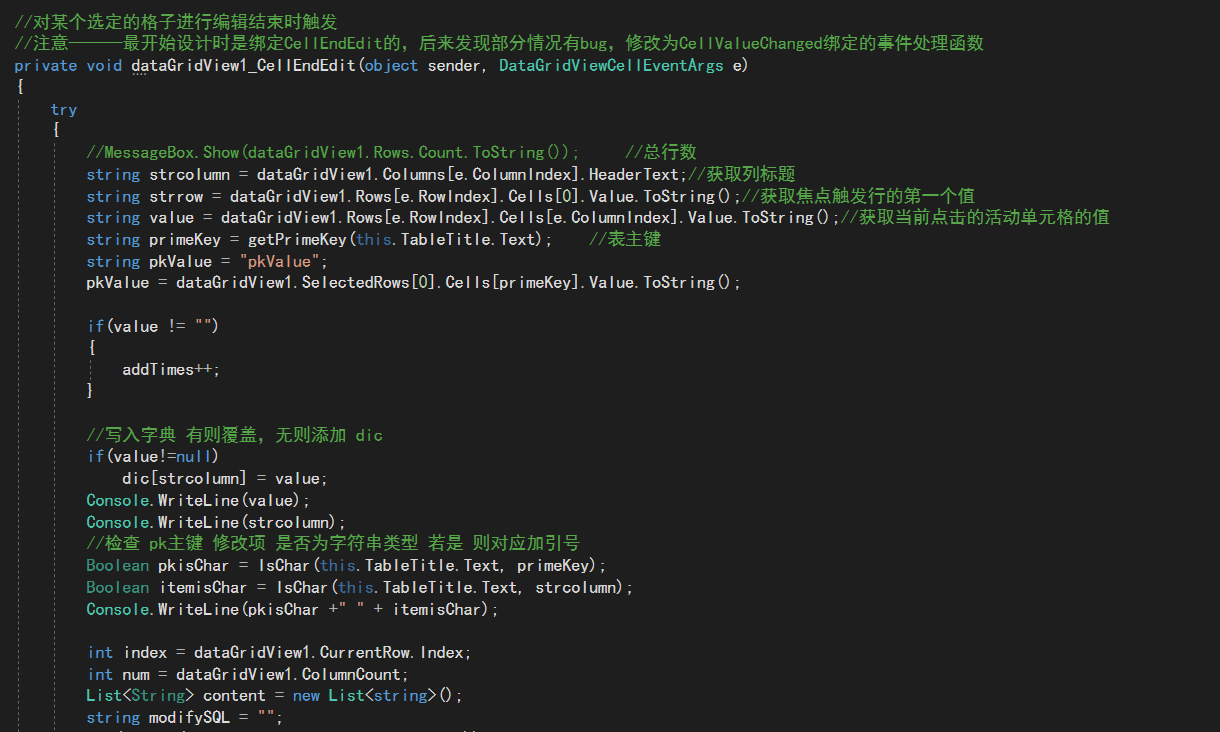


图3.2.9 CellEdit在数据表页面中直接修改数据



图3.2.10 CellEdit在数据表页面中直接修改数据

如上图所示，代码的很多部分都在做一些检查的工作，比如获取主键，比如是否是char，是否是空，最后得到了sql的字符串形式，实际执行只有短短的一行ExecuteNonQuery(modifySQL);

调用我封装的ExecuteNonQuery函数，执行非查询的sql指令（增/删/改）

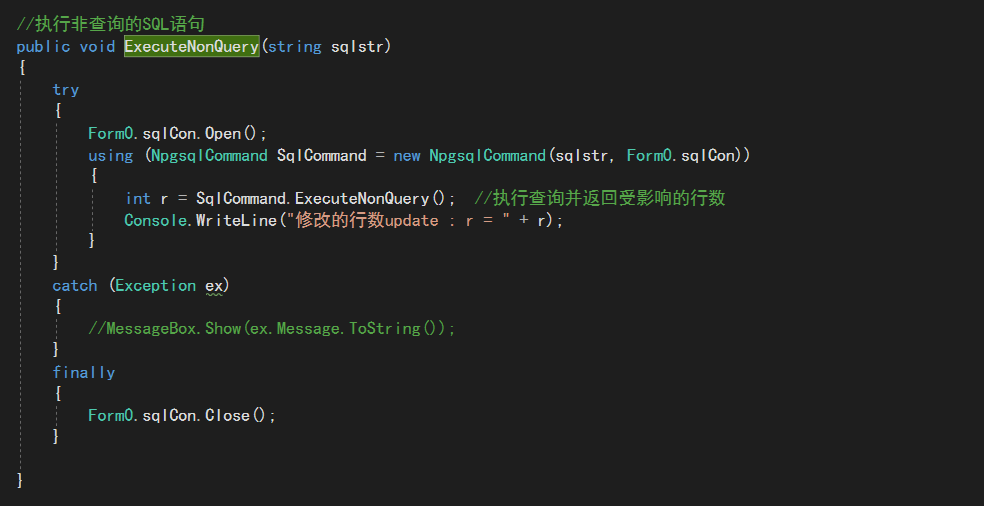


图3.2.11 ExecuteNonQuery函数

**3.2.5 异常、重复数据处理、重要信息提示（比如删除/清空表）**

在执行删除表等重要操作之前，我引入了VB里面的Interaction.InputBox，来弹出一个输入/确认框，让用户在执行删除表等重要操作之前进行再一次的确认，从而减小误操作的可能。（这也是老师提出的第六个选做要求，虽然我觉得实现起来可能是最简单的，但是实际使用意义可能比较大）



图3.2.12 删除给出提示框，用户再次确认方可删除

类似Unix的风格，再次输入确认y后才可删除表！如果取消，可以放弃删除表的指令。



图3.2.13 显示效果

除此之外，在数据库应用程序中，还有一类很常见的数据库错误，就是异常值/重复值带来的，因为主键唯一，不能有重复相同主键的数据行，而且还有一些表的字段有非空约束，那么对应的cell就不能是空，这些都要进行检查，这里我直接根据数据库接口返回的Exception信息进行捕获，重新返回给用户更友好的信息形式进行提示。

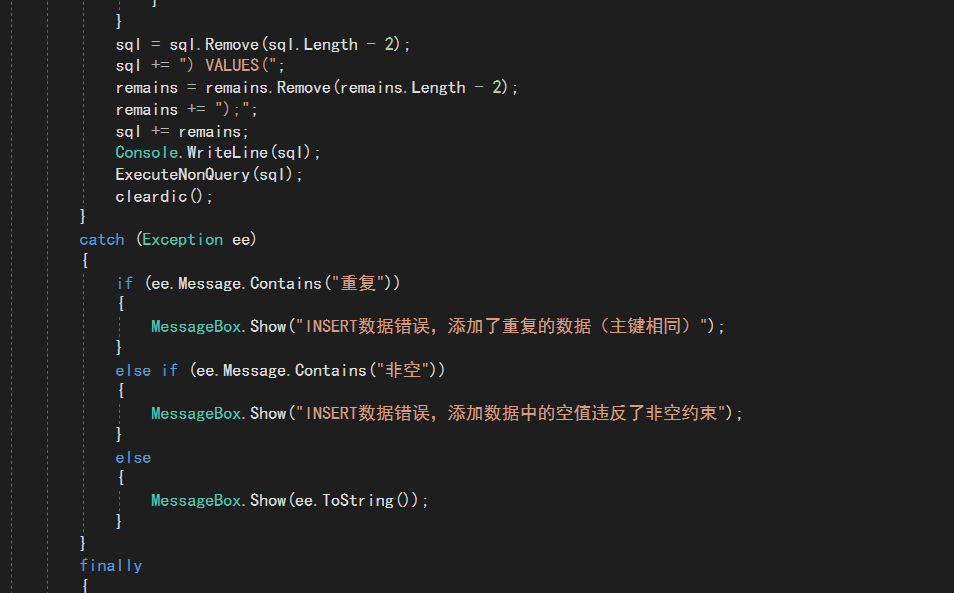


图3.2.14 利用数据库提示信息，反馈给用户以提示

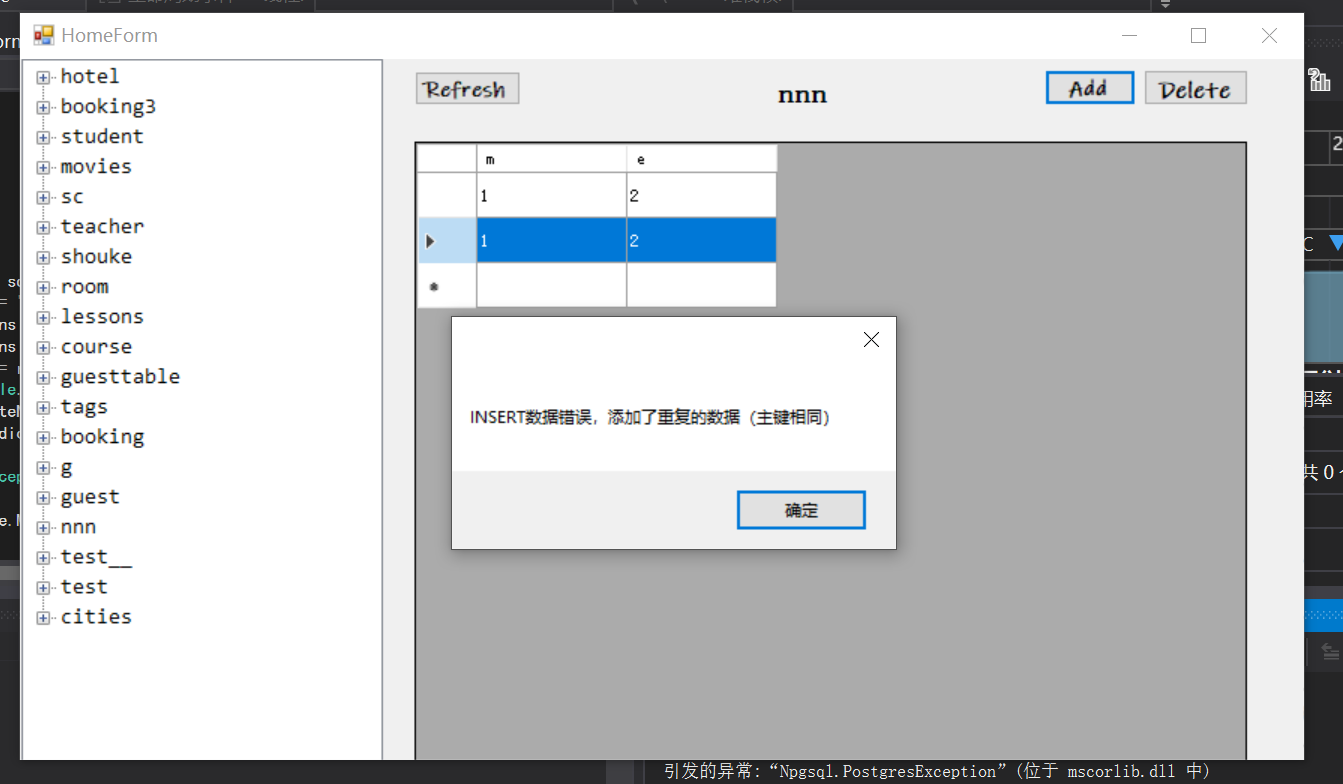


图3.2.15 重复数据错误提示



图3.2.16 异常数据错误提示信息（非空约束）

通过这样处理，极大的减轻了我们编程的负担，我们不需要去自己检查这些问题，完全可以充分利用数据库接口带来的便利，利用NpgSql抛出的异常信息进行处理分析，返回给用户以提示。

**3.3实现效果与功能**

数据库程序启动时，会自动打开首页，用户输入正确的用户名密码方可连接PostgreSQL成功，不成功则会提示用户重新输入。（当然我默认给出了我自己的用户名和密码）

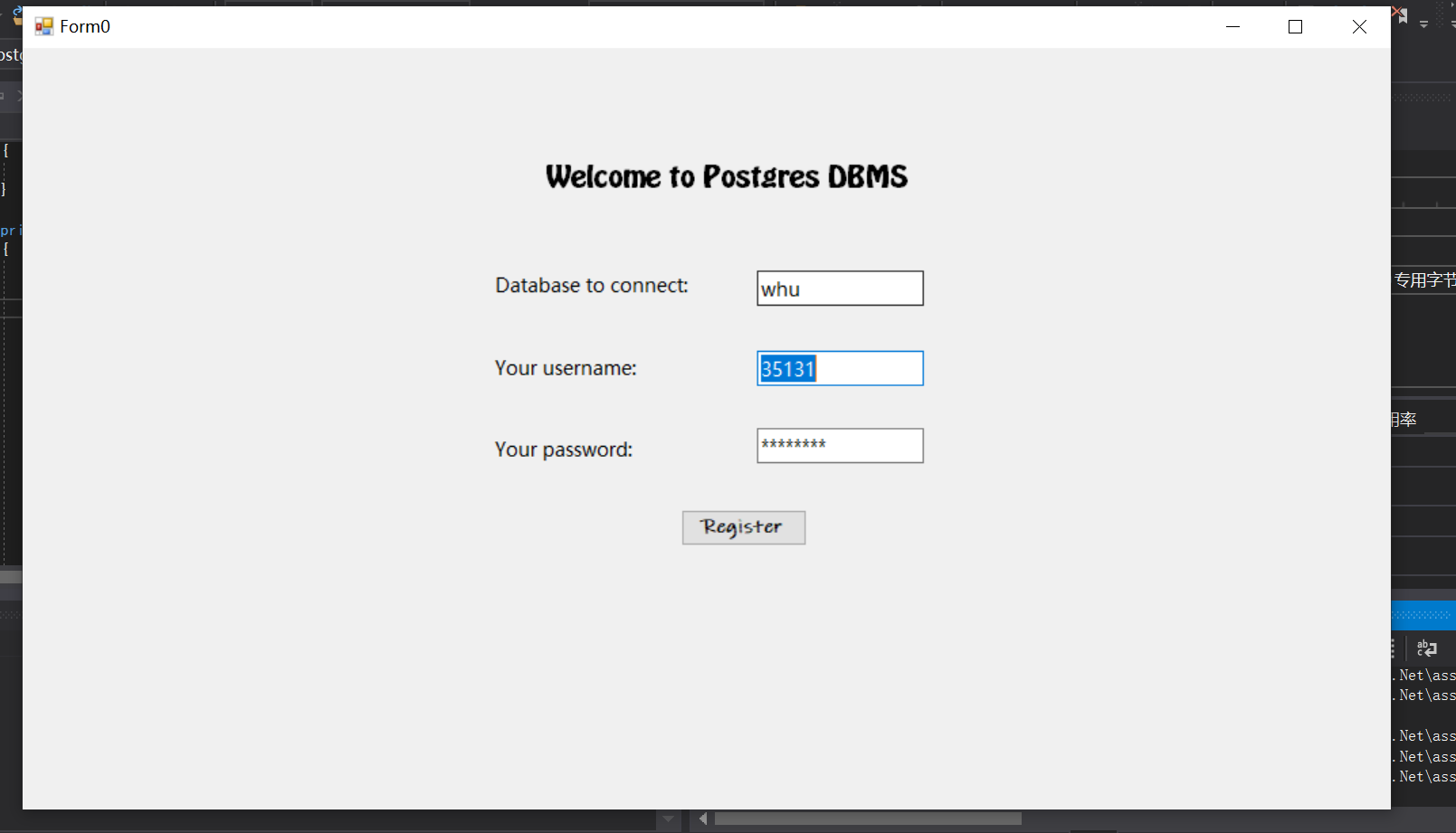


图3.3.1 启动效果

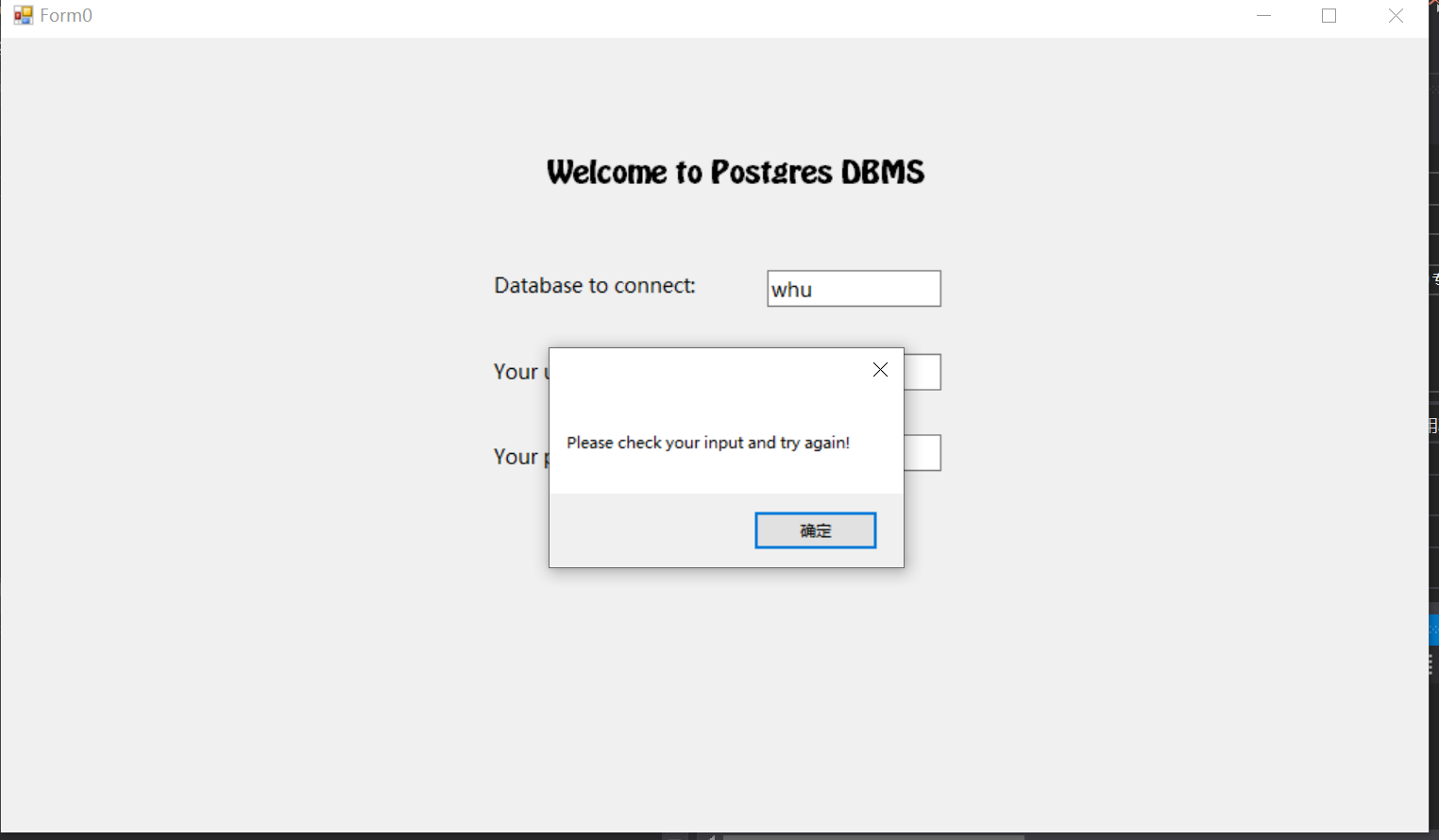


图3.3.2 输入密码/用户名/数据库名错误 提示信息

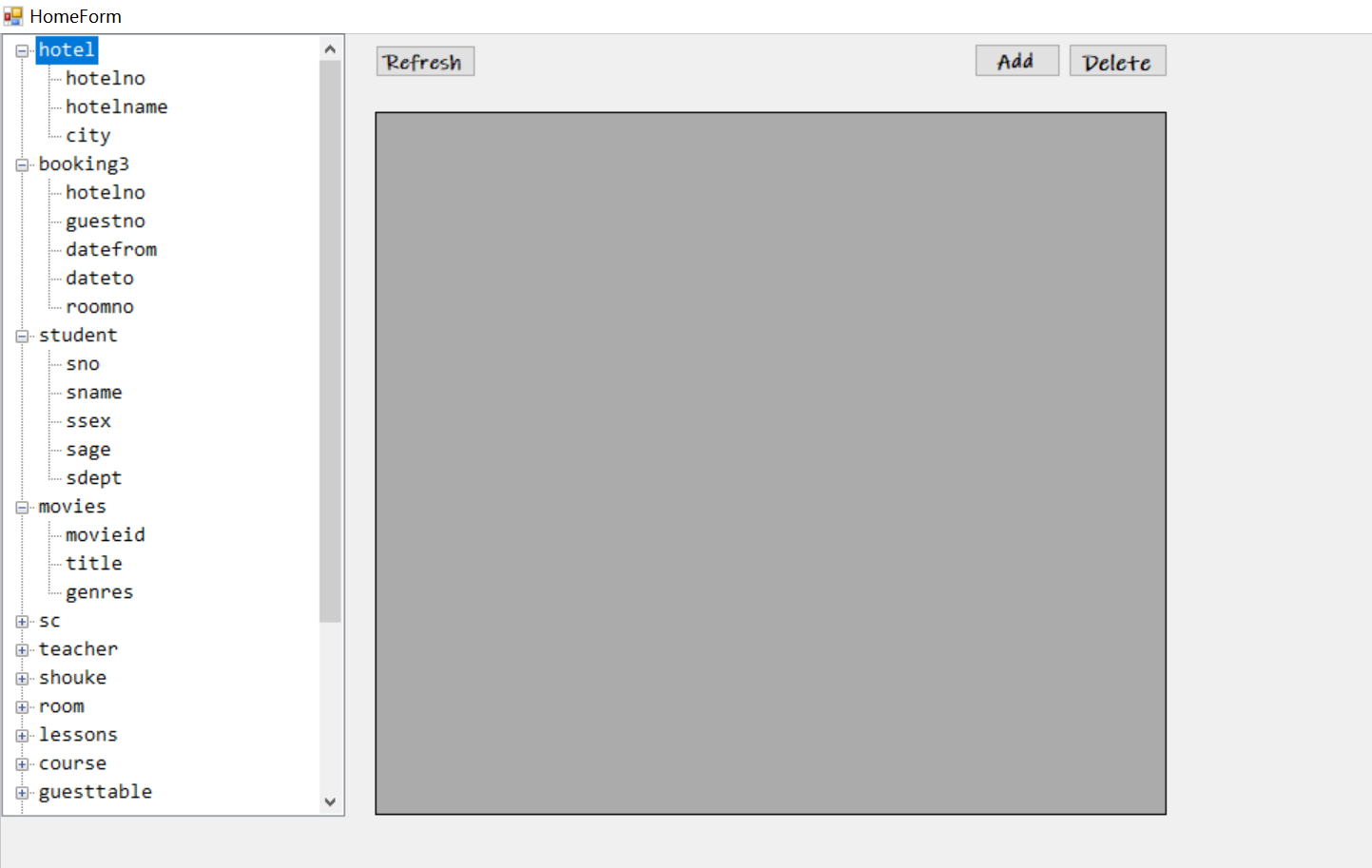


图3.3.3数据库主页面

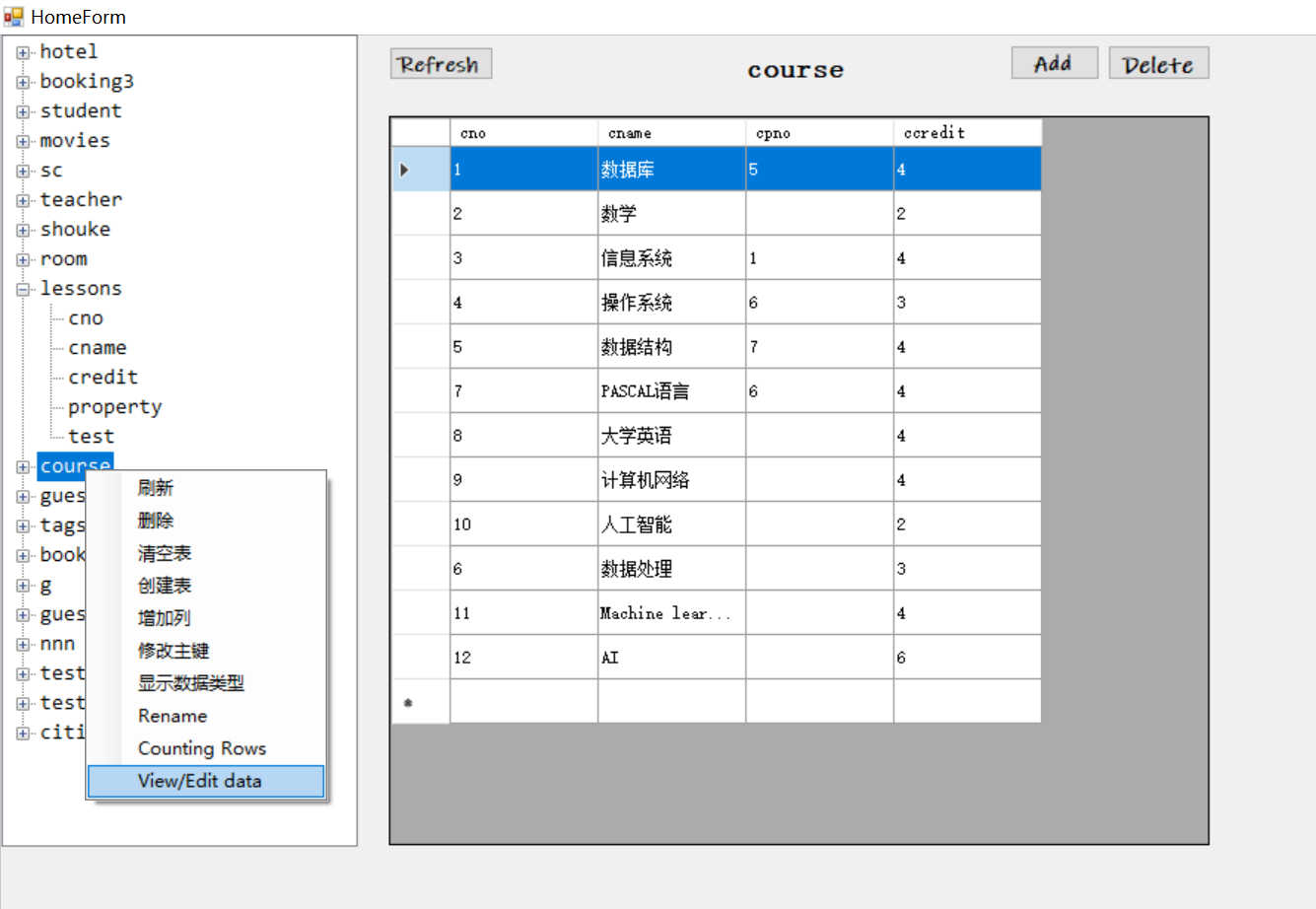


图3.3.4 右键点击数据表节点 展开菜单栏 查看数据

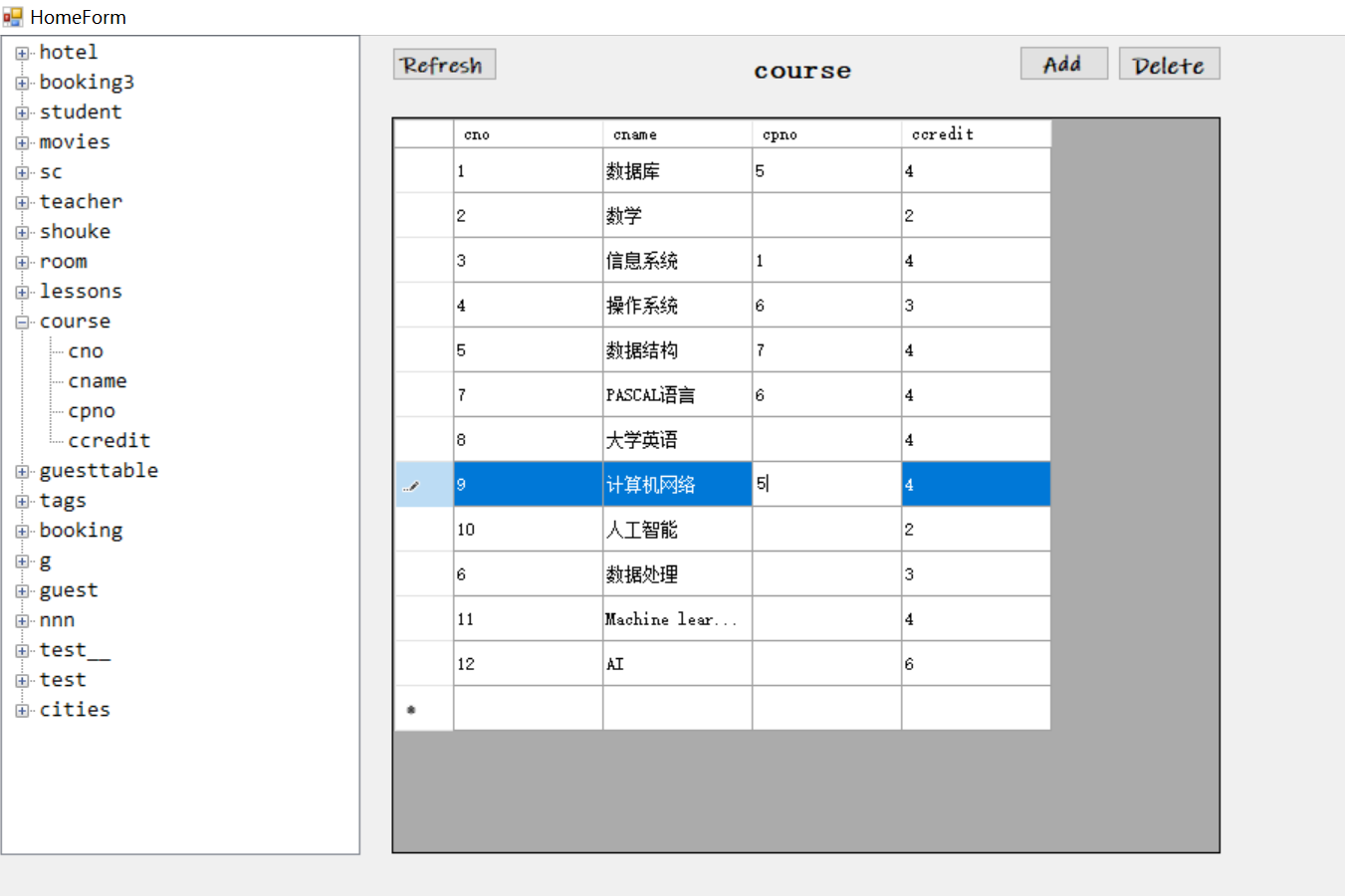


图3.3.5 DataGridView中修改数据



图3.3.6 最下方添加一行新数据Windows课程设计

（点击Add后提交到数据库）

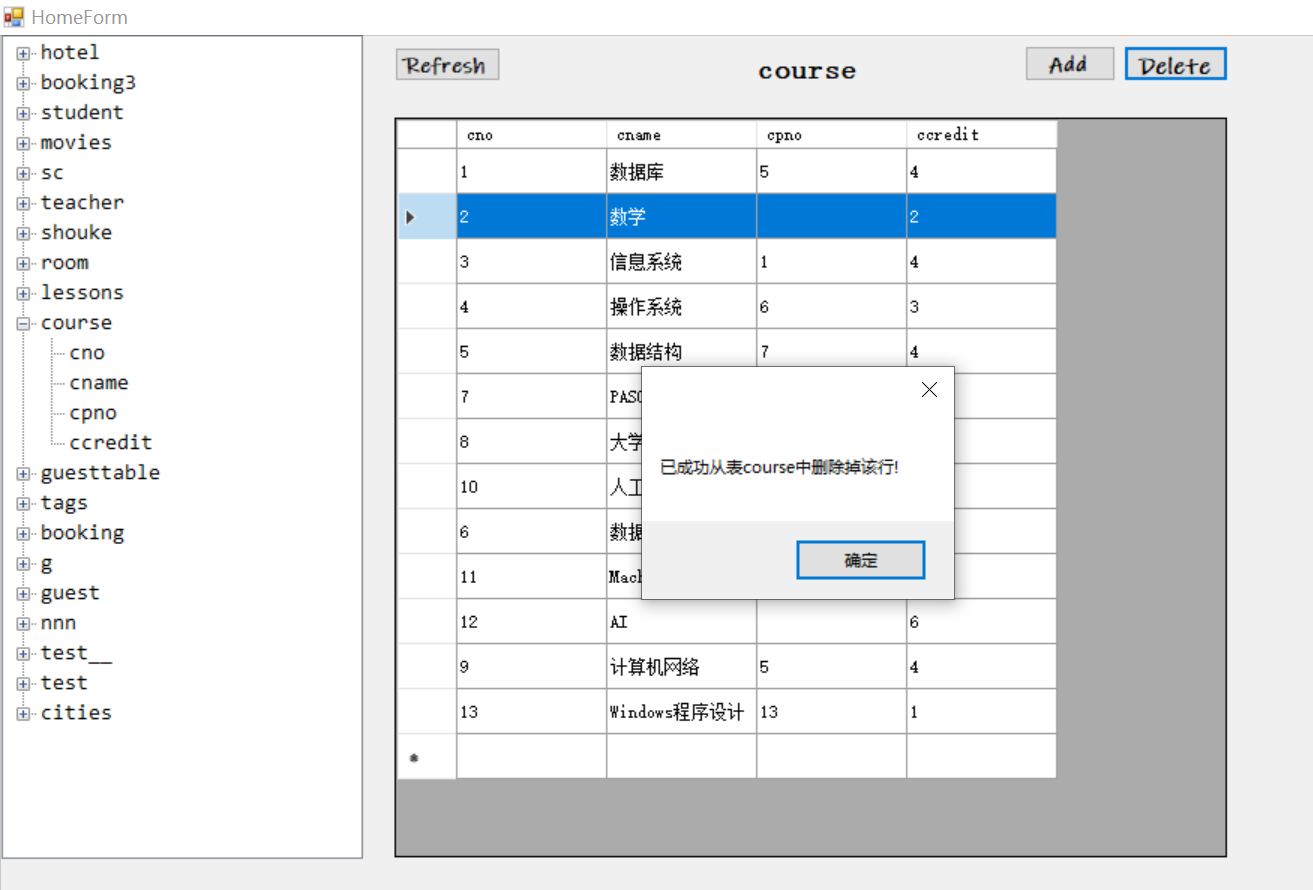


图3.3.7 从数据表Course中删除掉“数学“这一列

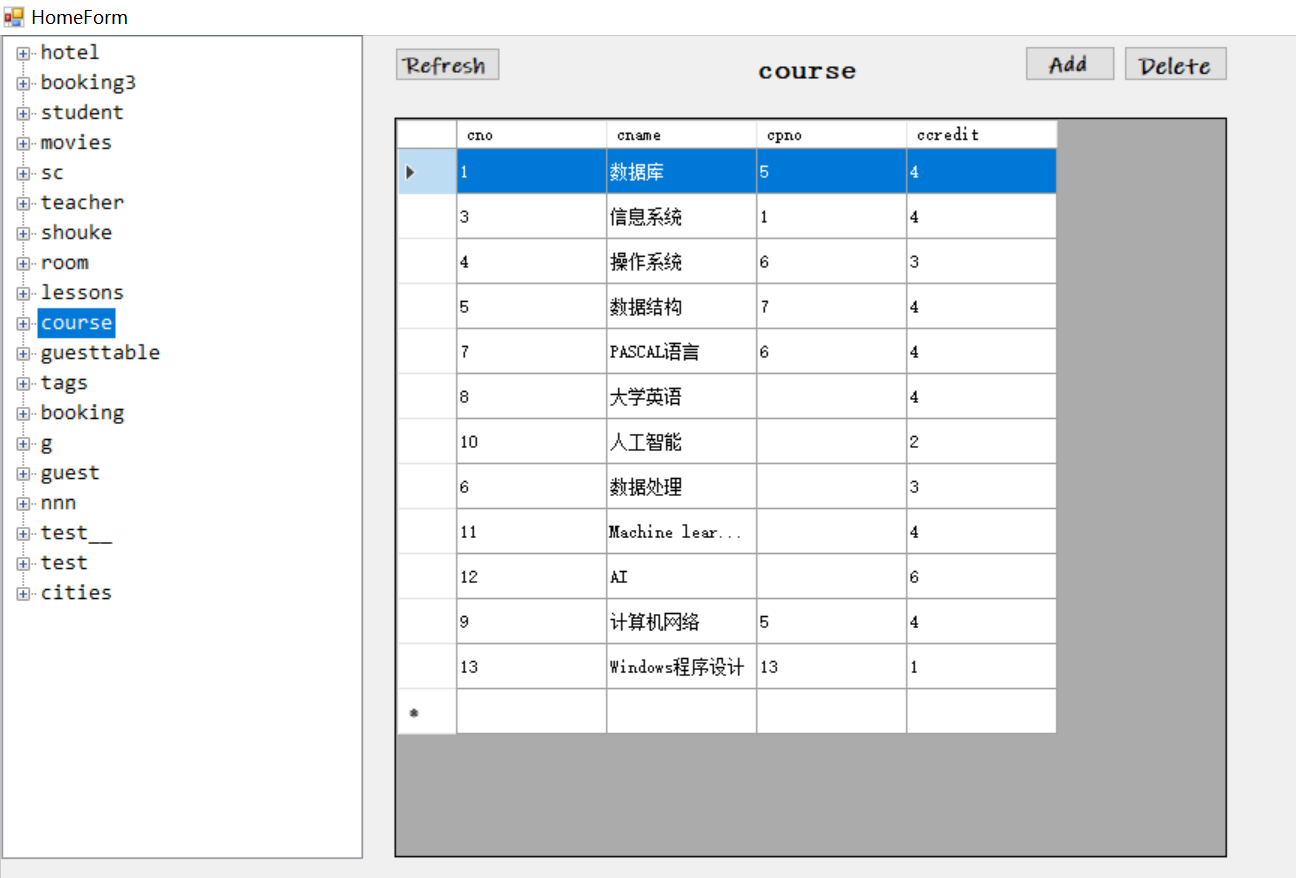


图3.3.8 刷新后，数据表显示已顺利实现刚才的增删改的操作

（“Windows程序设计”增加，“数学”删除，cpno修改成功）

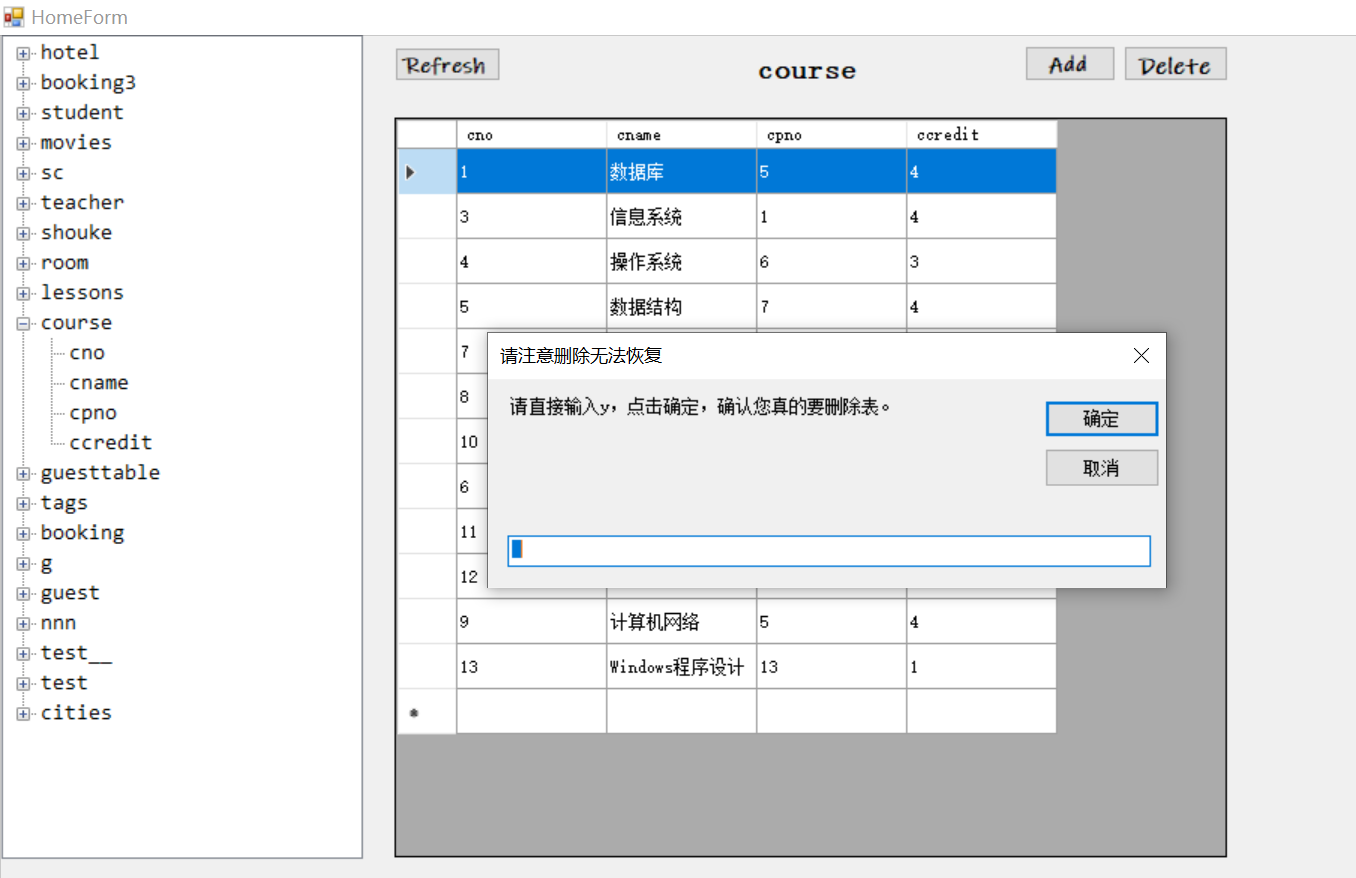


图3.3.9 删除前提示确认是否要删除表

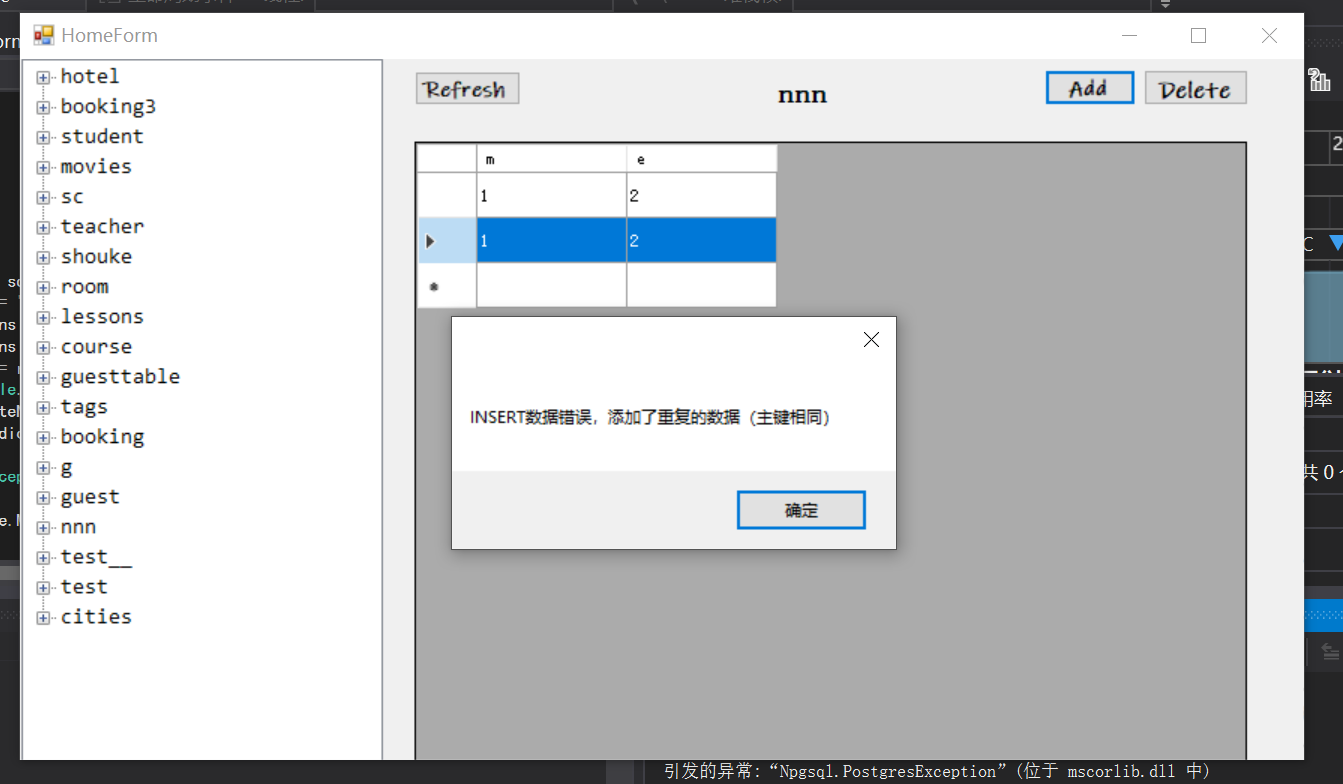


图3.3.10 重复数据错误提示



图3.3.11 异常数据错误提示信息（非空约束）

**3.4 难点解决与总结**

总体来说，我的数据库应用程序**完整的实现了实验要求的选做必做全部功能**，在DataGridView提供的可视化页面中实现增、删、改、查四种业务操作，基于NpgSql提供的接口访问我本机安装的PostgreSQL数据库，并直接在Winform的界面中利用DataGridView, ContextMenuStrip，TreeView等组件进行显示。选做功能：对界面中输入的非法数据或无效数据或重复数据要进行相关提示，在删除数据时要给出相应的提示，由用户确认后再进一步删除，这两个选做功能也已完成。

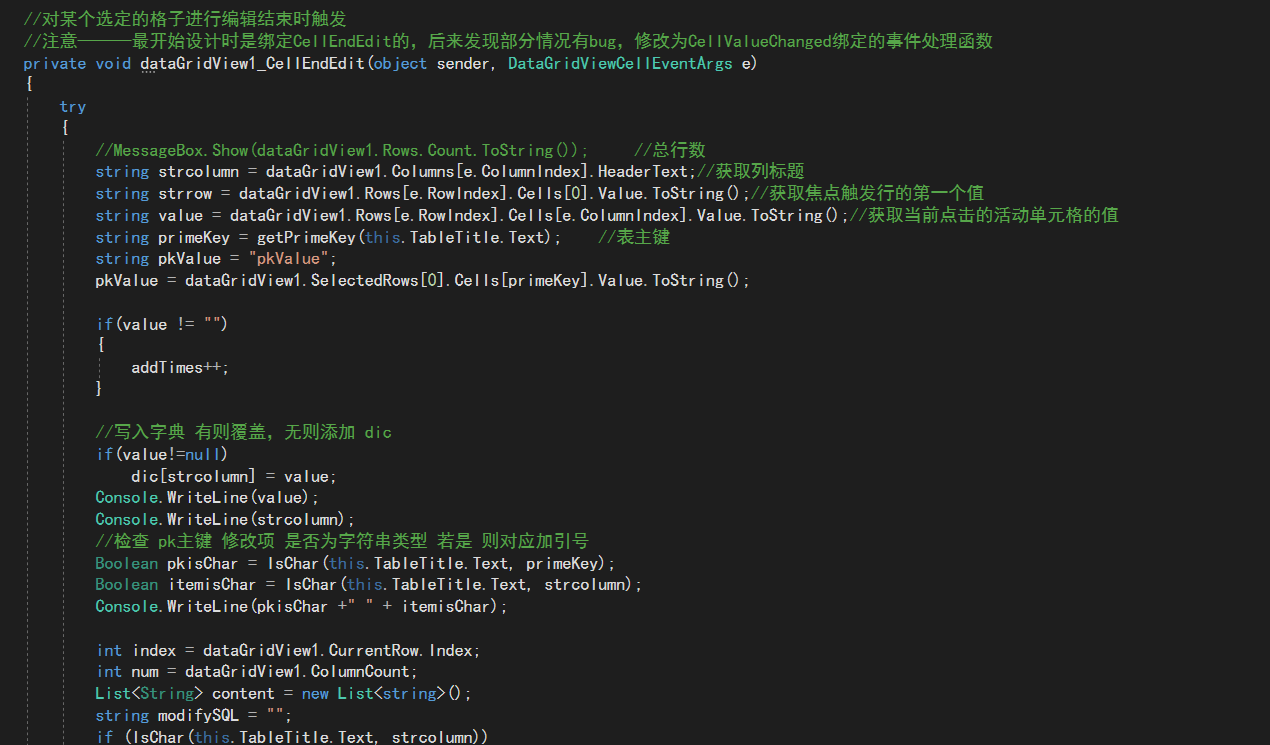
下面结合编写程序过程中遇到的核心难点，也就是增删改查的可视化操作背后的一系列问题，谈一谈具体实现中自己是如何解决的：

**如何把复杂的增加/修改/删除统一到DataGridView来进行**

这是一个非常复杂的问题，我在3.2.4有过一点点小小的讨论，可能的问题包括而不限于主键的考虑，你怎么知道你要改的那一行是唯一的，怎么保证修改的唯一性；又比如说添加某一行，万一有空值，那么怎么知道这个空值是否可以为空，因为数据库的表可能有些字段不能为空；又比如说修改的时候，假如有的字段是char/varchar字符型，有的就是数值型，那么update xxx set xxx = 的时候可能需要考虑到需不需要加引号，因为字符型必然需要引号，而数值型不需要，但是dataGridView触发事件的时候对此一无所知，所以这些都是我们必须要去考虑到的问题。

因此要解决这个问题，我们需要通过一系列的检查和异常处理来完成

**1）首先我们要定义一个CellEndEdit事件绑定函数**



因为我们修改一个cell的值的目的是很复杂的，可能是要修改一个值，可能是要为增加行做准备，而这两个目的是完全不一样的，update会要立刻修改，而insert则是等用户认为要修改好了再才会修改。

因此这个函数的工作其实并不一定会立刻修改，当我们编辑数据区的cell的时候，它会认为是要update，所以会立刻executeNonQuery，但是如果是编辑的最后一行，它会先存入缓存dic中去（dic是静态的字典）

private static Dictionary<string, string> dic = new Dictionary<string, string>();

当我们需要insert的时候，也就是用户点击Add按钮的时候，我们才会从dic中取出该行的数据，并尝试添加到数据库中去。

**2）获取主键**

因为我们要保证修改的唯一性，update xx set xx = 的时候必须要有主键来保证我们修改是唯一的准确的，不能多也不能少，所以必须要有一个函数来获取某个table的主键名：



图3.4.2 获取主键

这里PostgreSQL提供了查询表主键的SQL命令，调用返回即可，但这是很重要的，因为主键没获取就修改或者删除表的数据是很危险的。

**3）判断是否是字符类型**

检查某个表的某个字段是否是字符类型也是必要的。因为无论是增删改查的哪一种，都逃脱不了指定值的需要，而要指定某个值，在数据库中，字符类型和数值类型是完全不同的，如果是字符型，引号是必须纳入到sql指令中去的，因此我们必须把字段类型检查的函数也封装起来：

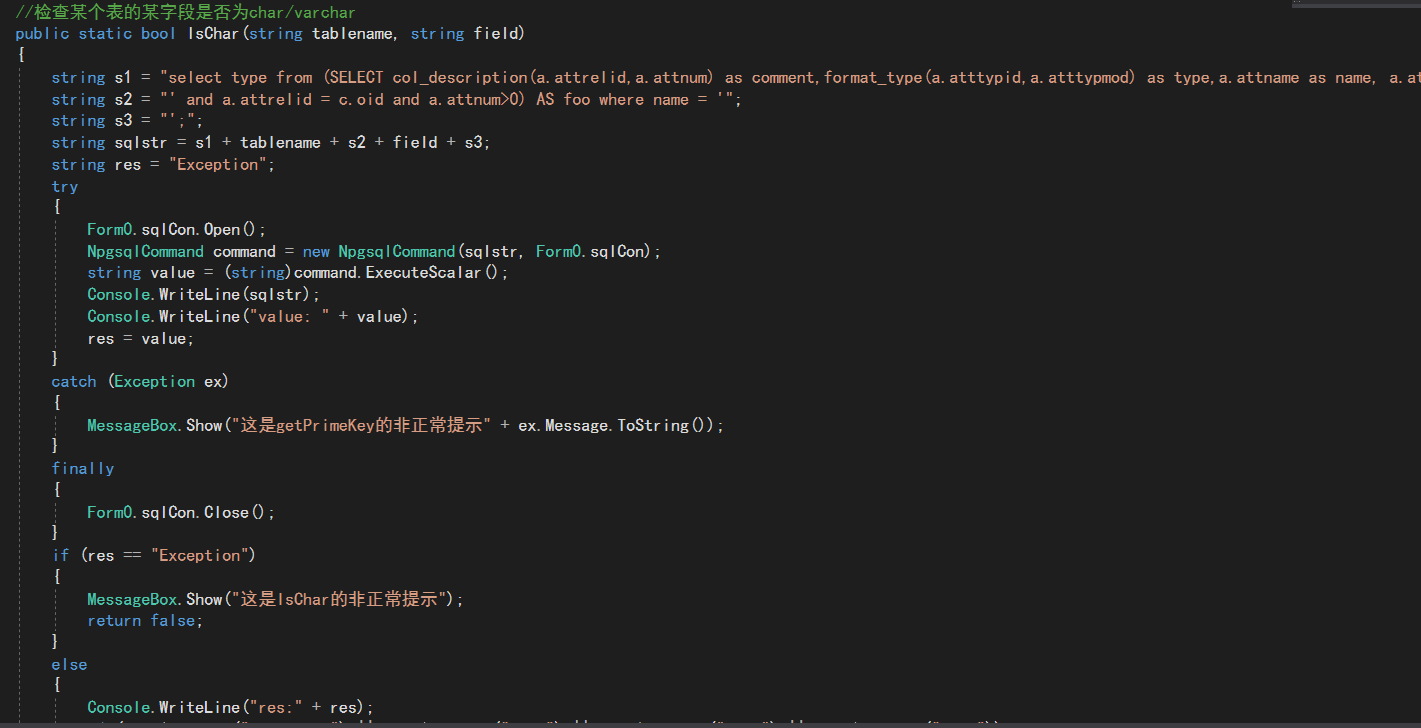


图3.4.3 检查字段是否是字符型

4）异常检测与提示

因为数据库的很多错误提示都很专业很生硬，我们也不好直接把Exception转成string直接给用户看。但如果我们自己去实现每次的异常检测又难免太过复杂，所以我采取了一个折中的方式，也就是充分利用数据库的异常信息，重新catch并返回提示给用户更友好的信息。

比如重复数据添加，我们可以判断这个异常信息是不是关于重复添加的，是则重新修改后返回给用户，又比如连接数据库，如果连接出现异常则会catch并提示用户可能是用户名或者密码输入错误。

关于删除表等操作前予以提示的实现，我采取了VB的Interaction. InputBox，这个组件不是Winform的，而是需要从VB中重新引入的，它很好的实现了弹出框让用户再次确认或取消的这个功能。

**四、实验总结**

**1）实验概述**

本次Windows实验我圆满完成了选题中的两个项目，计算器程序和数据库应用程序，不仅实现了实验要求中的全部选做和必做的目标功能，还在功能模块上进行了合理的拓展和思考，为进一步改进提供了基础。

通过这次实验，我更加熟练的掌握了WPF和Winform开发中的一些特点和区别，对于.Net环境下的开发流程有了更深的体会，也对于DLL动态链接库以及数据库应用程序中应当注意的点有了更多的了解，相信以后再遇到.Net的开发项目时会更加有经验，更加得心应手。

在本次实验中，我圆满的完成了全部选做和必做要求，并在原有基础上有一些拓展和思考。经过综合考量，我认为我的程序主要有一些几个优点和待补充完善的地方：

1） 功能相对完善

2） 封装度较高，代码可读性与可维护性较高

3） 架构清晰，尤其是WPF程序中尽可能分为了三层架构

4） 整体页面交互流畅，合理利用事件机制，ContextMenuStrip，DataGridView，TreeView等组件显示效果很好

5） 采取了相对高效的数据结构与算法，比如采取了逆波兰式求解算术表达式，采用C#提供的数据栈、列表等存储结构

6） 对于异常数据、重复数据、异常操作的信息提示清晰，对于重要的不可逆的操作提示清晰

7） 回退功能缓冲区内存控制

不完善的地方：

1）缓冲区没有采取增量更新的方法，交互上也没有显示出全部历史（只能通过回退键来回退）

2）数据库程序的迁移难度大，而且Winform写起来容易，但是页面相对WPF还不够美观

**2）实验过程中难点与收获小结**

·字符类型在C++创建的DLL和C#的WPF程序传递容易出现问题：

采用op\_code的方式，避免了字符传输的乱码的问题。

·对于string类型的复杂运算表达式求值：

充分利用C#提供的Split，Char.isDigit等函数对字符串处理，再使用C#提供的Stack/List等数据结构结合逆波兰式算法求值。

·调用DLL动态链接库的特点：

有静态调用和运行时的动态调用两种方式，静态调用是编译系统就完成对DLL的加载和卸载的编码，而动态调用方式运行时加载或卸载DLL，能更有效地使用内存。

·数据库的异常错误非常多，如果自己实现起来很复杂：

运行数据库接口抛出异常，但是我们利用try catch分别进行捕捉处理，在不同的情形下返回给用户更准确，更友好的提示信息。

·数据库修改某个单元格Cell值的目的不明确，可能是要准备增加，可能是要修改值，那么怎么处理呢：

这里比较复杂，具体情况在3.4已有说明，需要我们分位置处理，对于每个字段都要检查是否属于字符类型，每个表的主键也应当获取到，从而保证删除/更新数据的唯一性。

·数据库操作比较复杂，如何尽可能使得交互页面更加快捷直观？数据如何加载显示？

用户选择表后会把对应数据通过DataAdapter最后转化成DataTable，设定为DataGridView的datasource来实现。Winform程序页面还采取了ContextMenuStrip，TreeView等组件，使得交互形式更方便。

**结论**

本次Windows实验我圆满完成了选题中的两个项目，计算器程序和数据库应用程序，不仅实现了实验要求中的全部选做和必做的目标功能，还在功能模块上进行了合理的拓展和思考，为进一步改进提供了基础。

通过这次实验，我更加熟练的掌握了WPF和Winform开发中的一些特点和区别，对于.Net环境下的开发流程有了更深的体会，也对于DLL动态链接库以及数据库应用程序中应当注意的点有了更多的了解，相信以后再遇到.Net的开发项目时会更加有经验，更加得心应手。

在本次实验中，我圆满的完成了全部选做和必做要求，并在原有基础上有一些拓展和思考。经过综合考量，我认为我的程序主要有一些几个优点和待补充完善的地方：

1. 功能相对完善
2. 封装度较高，代码可读性与可维护性较高
3. 架构清晰，尤其是WPF程序中尽可能分为了三层架构
4. 整体页面交互流畅，合理利用事件机制，ContextMenuStrip，DataGridView，TreeView等组件显示效果很好
5. 采取了相对高效的数据结构与算法，比如采取了逆波兰式求解算术表达式，采用C#提供的数据栈、列表等存储结构
6. 对于异常数据、重复数据、异常操作的信息提示清晰，对于重要的不可逆的操作提示清晰
7. 回退功能缓冲区内存控制

不完善的地方：

1）缓冲区没有采取增量更新的方法，交互上也没有显示出全部历史（只能通过回退键来回退）

2）数据库程序的迁移难度大，而且Winform写起来容易，但是页面相对WPF还不够美观

**参考文献**

[1] 李赞，李文生.Windows程序设计——基于.Net平台[M].清华大学出版社，2016，1

[2] The PostgreSQL Global Development Group（PostgreSQL全球开发组）. PostgreSQL 11.0 官方文档[EB/OL]. http://www.pengyuwei.net/PGDOC/110/ index. html, 2019–03

[3] 博客园94cool. 在VS2015中用C++创建DLL并用C#调用且同时实现对DLL的调试 [EB/OL]. https://www.cnblogs.com/94cool/p/5772376.html, 2016–08-15

**附件**

考虑到整体两个项目的代码量比较大，所以我在实验报告中没有附上完整代码，具体代码请参考与实验报告一并提交的附件，或者实验报告中的部分关键代码的截图。

附件文件夹说明：

WPF计算器：计算器程序

PostgreTest1：数据库应用程序

|  |
| --- |
| 【结论】：  本次Windows实验我圆满完成了选题中的两个项目，计算器程序和数据库应用程序，不仅实现了实验要求中的全部选做和必做的目标功能，还在功能模块上进行了合理的拓展和思考，为进一步改进提供了基础。 |
| 【小结】：  通过这次实验，我更加熟练的掌握了WPF和Winform开发中的一些特点和区别，对于.Net环境下的开发流程有了更深的体会，也对于DLL动态链接库以及数据库应用程序中应当注意的点有了更多的了解，相信以后再遇到.Net的开发项目时会更加得心应手。 |
| 指导老师评语及成绩 |
| 【评语】：  成 绩： 指导老师签名：  批阅日期： |