班级：-------- 姓名：-------- 学号：--------

题目一 银行门户设计

设计文档

一、 设计任务描述

该门户是一个单独的程序，程序要求至少支持以下功能：

1) 注册银行卡，银行可选；

2) 登陆；

3) 修改银行卡密码；

4) 存款取款；

银行卡至少需要有卡号，密码，所属银行名称，持卡人身份证号，卡内金额等内容。

要求 1：请把所有的银行名写入文件（或数据库），注册银行卡的时候，要求只能选择已经存在的银行进行注册。

要求 2：请做好错误场景的处理，例如读银行文件错误，输入数据不合法等等。

二、 设计任务分析

由于银行门户的账户必须做到在关闭以后保存信息，因此使用数据库来存储银行所有账户信息。为了系统维护方便，没有将银行信息写在代码中而是用另一个数据库来专门存放有哪些银行，在注册时通过读取银行信息的数据库来给予用户更多选择。目前主流的流程为，

用户先进入登录界面，登录界面上有注册按钮，注册成功后可以直接回到登录界面进行登陆。而修改密码应该在登录成功后在用户设置中进行。而存取款则是通过对账户金额进行改动，在取款时需要判断取款金额是否大于当前账户余额，避免账户余额出现负数，银行出现亏帐。

至于错误处理，在设计输入界面时通过正则表达式可以规避很多常见错误（比如输入的不是数字，输入负数等等），再通过提交时对输入的二重判断即可保证用户不能提交错误信息到银行系统，处理目前能想到所有的错误场景。

三、 模块划分

银行系统按照功能分为以下几个部分：

1. 注册：负责从用户得到账户信息并写入数据库中
2. 登录：在数据库比对账户信息并决定是否进入主界面
3. 主界面：显示账户信息并提供存款等功能的入口
4. 存款：将金额存入用户账户中
5. 取款：先判断是否可以取款，然后从用户账户中取出相应金额
6. 修改密码：在数据库中修改当前用户的密码
7. 数据库：定义数据库

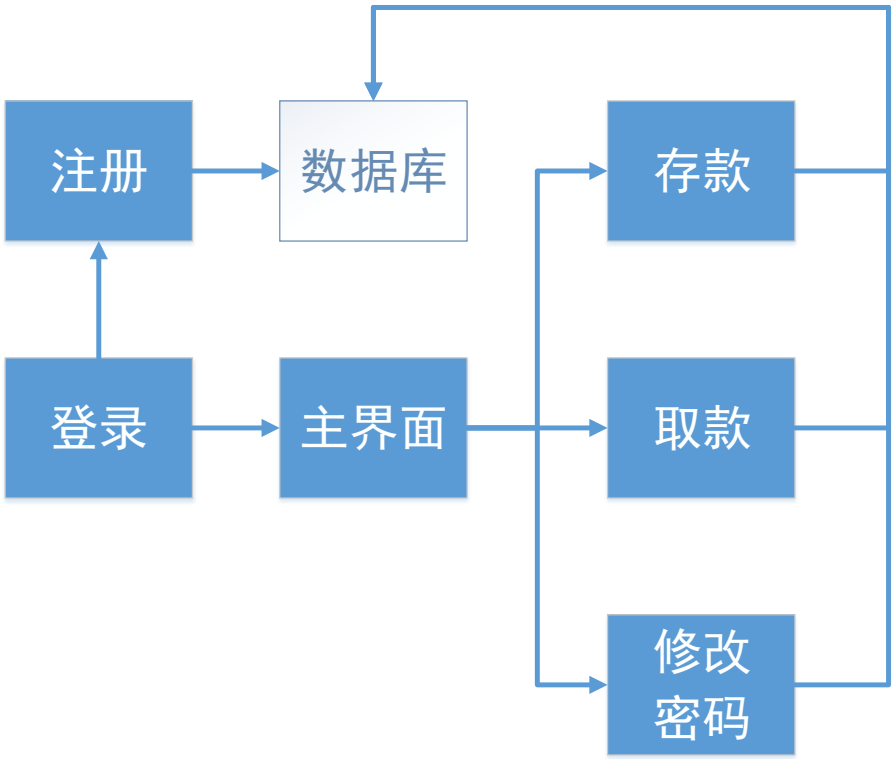


图 1 银行系统的模块划分

四、 数据结构

class account{

private:

QString num; //卡号

QString password; //密码

QString bank; //银行

QString id; //身份证号

QString contact; //联系方式

float balance; //账户余额

public:

```
    /*  构造函数  */
    account()=default;
    account(QString s1, QString s2, QString s3, QString s4, QString s5,
float f1):num(s1),password(s2),bank(s3),id(s4),contact(s5),balance(f1)
{}
    /*  私有成员的 get/set 函数  */
    QString getNum();
    void setNum(QString num);
    QString getPassword();
    void setPassword(QString password);
    QString getBank();
    void setBank(QString bank);
    QString getId();
    void setId(QString id);
    QString getContact();
    void setContact(QString contact);
    float getBalance();
    void setBalance(float balance);
    /* addBanlance 和 subBalance 提供对账户余额的直接增减  */
    void addBalance(float add);
    bool subBalance(float sub);
};
```

数据库： **accountInfo：**

num NVARCHAR PRIMARY KEY　　//账号

password NVARCHAR　　//密码

bank NVARCHAR　　//银行

id NVARCHAR　　//身份证号

balance REAL　　//账户余额

contact NVARCHAR　　//联系方式

数据库：**bankInfo：**

name NVARCHAR PRIMARY KEY　　//银行名称

num INT　　//序号

五、 测试结果

1. 存入 1000.56 元，此处限定只能输入 10000 元以下的含两位小数的数字，与目前自动取款机相符。

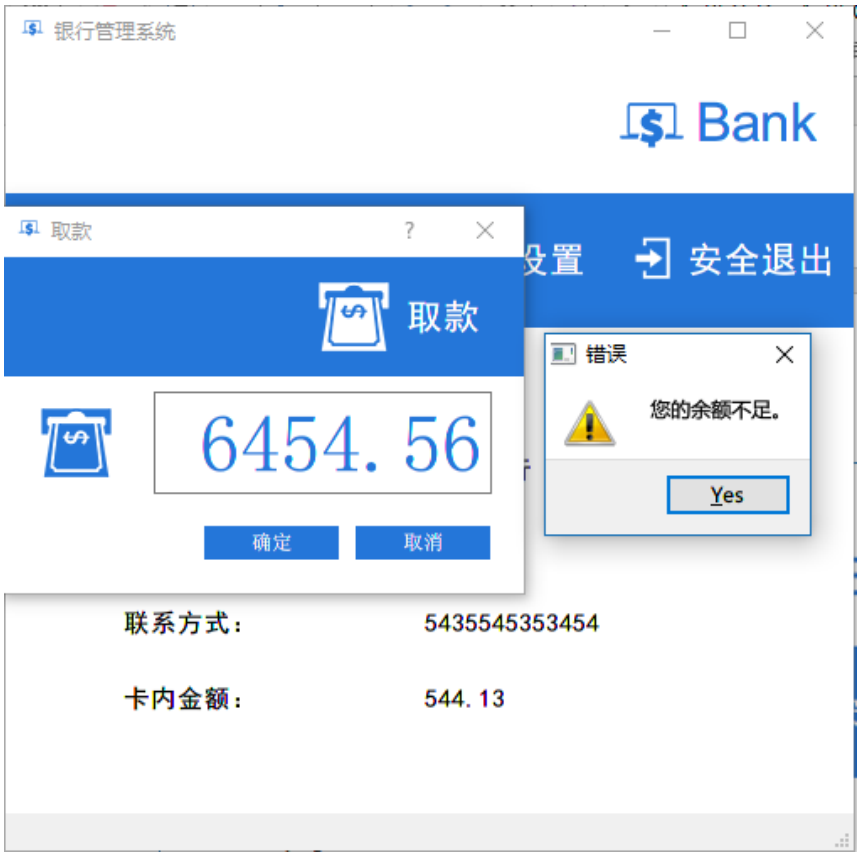
可以看到，账户余额已经变化：

2. 取出 456.43 元:

可以看到，余额已经变化：


银行管理系统
Bank
存款
取款
设置
安全退出
卡号：
123
银行：
中国银行
身份证号：
5435
联系方式：
5435545353454
卡内金额：
544.13

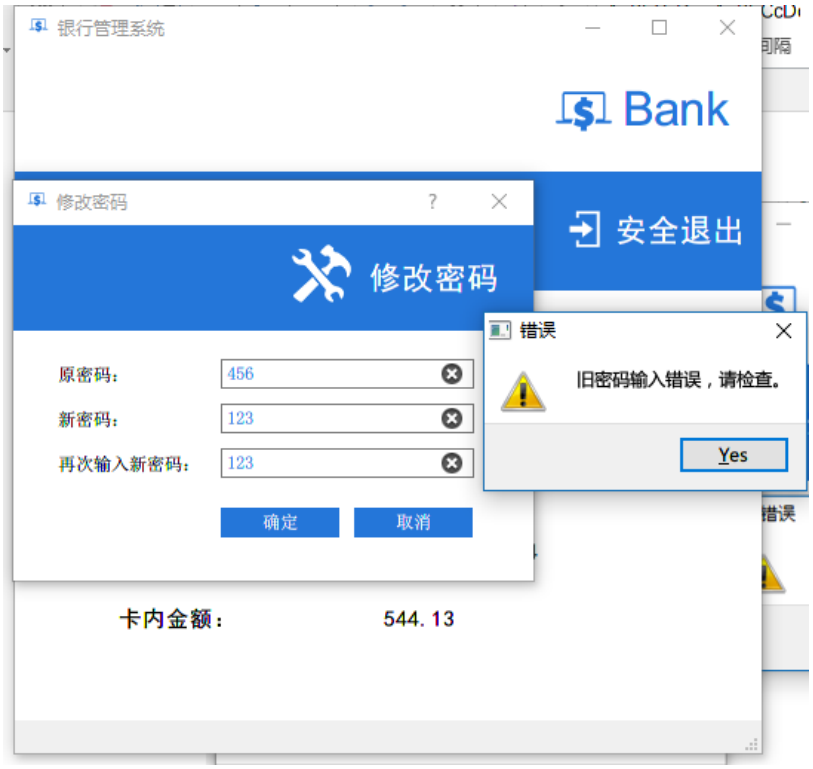
当余额不足时：


银行管理系统
Bank
取款
取款
6454.56
确定
取消
设置
安全退出
错误
您的余额不足。
Yes
联系方式：
5435545353454
卡内金额：
544.13

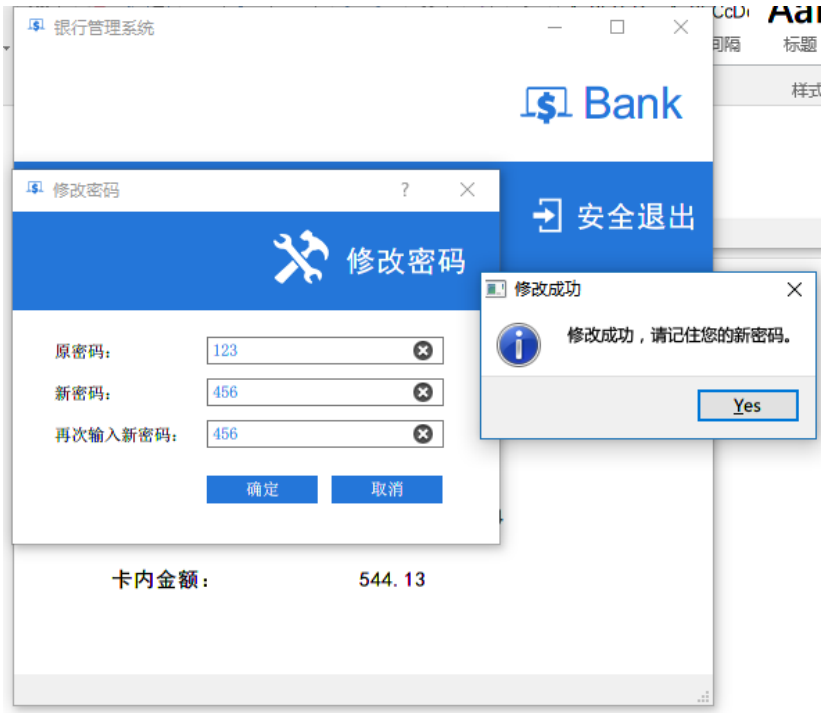
3. 修改密码：

当未填写信息时：

当输入密码错误时：



当填写正确时，修改成功:


银行管理系统
Bank
安全退出
修改密码
修改密码
原密码：
123
新密码：
456
再次输入新密码：
456
确定
取消
修改成功
修改成功，请记住您的新密码。
Yes
卡内金额：
544.13

题目二 电商平台设计

设计文档

一、 设计任务描述

1) 注册&登录：支持新用户注册平台账号，已注册用户用平台账号登录平台。（要求已注册用户的信息长久保留。）
2) 主界面浏览平台产品信息。
3) 设置优惠活动：支持对同一品类下所有产品打折的活动，支持单笔订单满 X 减 Y 的活动。
4) 购物车购买产品：支持用户添加产品到购物车，查看实际应付的产品价格，提交订单。

在题目二我们暂时不考虑提交订单后支付等后续需求。

要求 1：电商平台上至少有三类产品：如食物、服装、图书等，每类产品中至少有三个具体的产品（如图书中可以有《C++ Primer》、《Effertive C++》等），每个具体的产品请至少包含产品描述，产品原价，产品剩余量等数据。所有的产品信息需要存储在数据库或文件中，不能写在代码中，平台管理员通过直接修改数据库或文件，管理本平台上的产品，包括产品的增加和删除，修改数量以及具体产品的属性信息等。

要求 2：请至少设计一层继承体系（产品基类-产品子类），设计一个产品基类，然后让图书类、电子产品类和服装类等产品子类类继承它，具体的产品是产品子类的实例对象（《C++ Primer》是图书类的

实例对象）。产品基类请至少具有一个虚函数 getPrice()用于计算具体产品的价格。

要求 3：请通过为每个产品子类定义“品类折扣系数”来支持对同产品子类下所有产品打折扣的活动（如图书全场 5 折，则图书类这一产品子类的折扣系数为 0.5）。

要求 4：请做好错误场景的处理。

二、 设计任务分析

首先针对电商平台应当先将产品信息对象化。设计类时根据要求将所有产品的共性设计在基类中，然后将每类产品不同之处设计在子类之中，通过继承体系将产品信息的不同部分明确表现出来。因此将产品名称、价格等信息封装在“产品”这一基类中，然后让“图书”“食品”等子类继承基类的信息，将折扣、产品类型各个类型不同的信息封装在子类中。由于 getPrice()这一函数在所有子类中都有，但实现方式不同，因此设为虚函数。将所有成员变量都设置为私有，然后通过 get/set 函数访问私有成员，保证安全性。

因为管理员可以通过修改数据库来增加或者修改产品信息，因此将类的成员全部一一对应到数据库中。接下来要考虑如何实现购物车功能。由于目前主流电商平台的购物车都是可以保存的，因此为每个用户建立一个专门的购物车数据库，每当用户将商品加入购物车时，也将商品信息存入数据库中，这样可以保证购物车不会在程序结束时丢失。那么如何实现不同分类物品打折以及满减活动呢?对于前者，通过在子类中设置 rate 属性，在 getPrice()中令实际价格=原价×折扣

实现。对于后者，专门设置全局变量 isEvent 决定是否开始满减活动，如果有满减活动，则满足要求后减去对应金额即可。

之后就是完善其他功能了，注册和登录功能与银行系统基本一致，在产品信息表现形式上，由于要求管理员可以增删物品信息，因此使用列表而非目前电商平台主流的图文表格形式来展示商品。使用列表还有一点好处在于可以方便地选择只显示某一分类的商品。在支付界面为之后的支付功能留好调用接口即可。值得注意的是支付后要对商品信息中的库存进行修改。

三、 模块划分

电商平台按照功能分为以下几个部分：

1. 注册和登录：读写账户信息
2. 主界面：显示全部或某一分类的商品信息，在商品信息上提供“添加购物车”功能，并提供购物车和用户设置等功能的入口。
3. 购物车：存放用户想要购买的商品，提供修改功能和支付功能（支付功能将在题目三中实现）。
4. 用户设置：修改当前用户的密码，提供绑定银行卡功能（将在题目三中实现）。
5. 数据库：定义产品信息、用户信息、购物车等数据库。

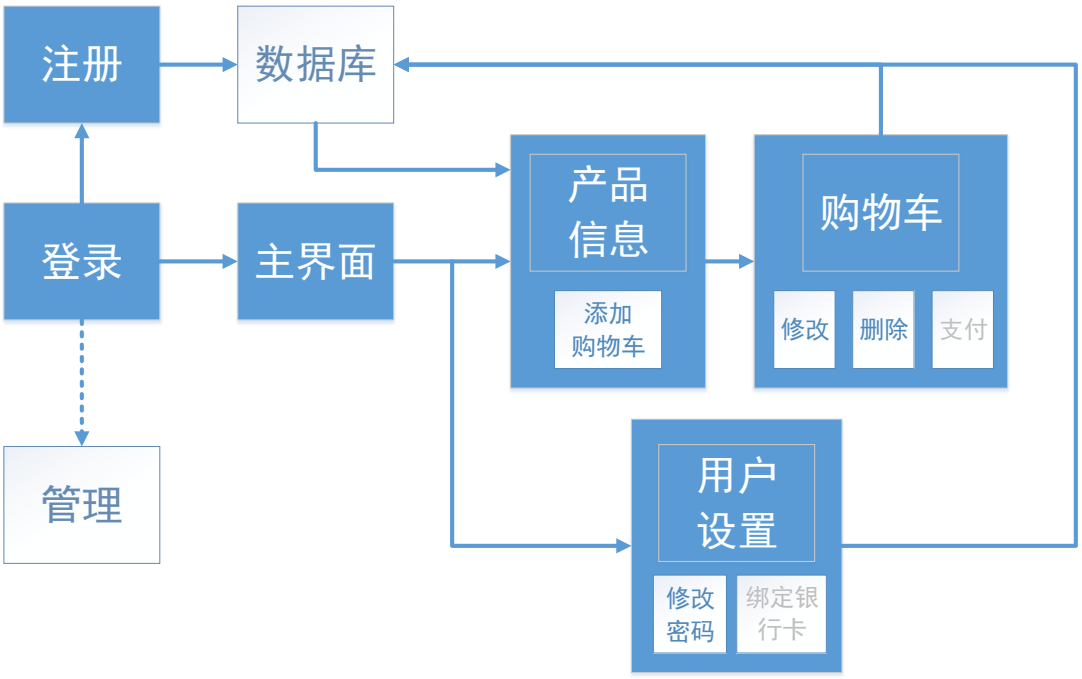


图 1 电商平台的模块划分

四、 数据结构

```
class product{                                  //产品信息
protected:
    QString name;                               //产品名称
    QString description;                        //产品描述
    float fullPrice;                            //价格
    int stock;                                  //库存

public:
    /* 构造方法 */
    product()=default;
    product(QString s1,QString s2,float f, int i):
```

```
    name(s1),description(s2),fullPrice(f),stock(i){
    }

    /*  私有属性的 get/set 属性  */
    QString getName();
    void setName();
    QString getDescription();
    void setDescription(QString description);
    float getFullPrice();
    void setFullPrice(float price);
    int getStock();
    void setStock(int stock);
    virtual float getPrice();
};

/*  继承自 product 的子类 book  （以 book 为例）*/
class book : public product{
private:
    float rate;                                    //优惠比率
    static const int type=BOOK;                    //类型
    QString author;                                //作者
```

```
/* 构造函数 */
public:
    book()=default;
    book(QString s1,QString s2,float f1, int i1, float f2,QString s3):
    product(s1,s2,f1,i1),rate(f2),author(s3){
    }
    void setRate(float);
    int getType();
    float getPrice();
};
```

数据库 **accountInfo**： //账户信息

```
    id NVARCHAR PRIMARY KEY          //账户名
    password NVARCHAR                //密码
    cartnum INT                      //购物车商品数量
```

数据库 **productInfo**： //商品信息

```
    id INTEGER PRIMARY KEY AUTOINCREMENT
    name NVARCHAR                    //商品名称
    type NVARCHAR                    //商品类型
    description NVARCHAR             //描述
    fullprice REAL                   //原价
    stock INT                        //库存
```

rate REAL //优惠比率

other NVARCHAR //其他信息

数据库 **cart+id** //购物车信息

id INTEGER PRIMARY KEY AUTOINCREMENT

name NVARCHAR //商品名称

num INT //商品在 store 中对应的下标

type NVARCHAR //商品类型

price REAL //商品原价

quantity INT //购买数量

数据库 **card+id** //绑定银行卡

num NVARCHAR PRIMARY KEY //银行卡账号

/* 在内存中存储商品信息 */

struct Store{

product * Product [100];

int ProductCount;

book * Book[100];

int BookCount;

cloth * Cloth[100];

int ClothCount;

```
    food * Food[100];
    int FoodCount;

    electro * Electro[100];
    int ElectroCount;
};
```

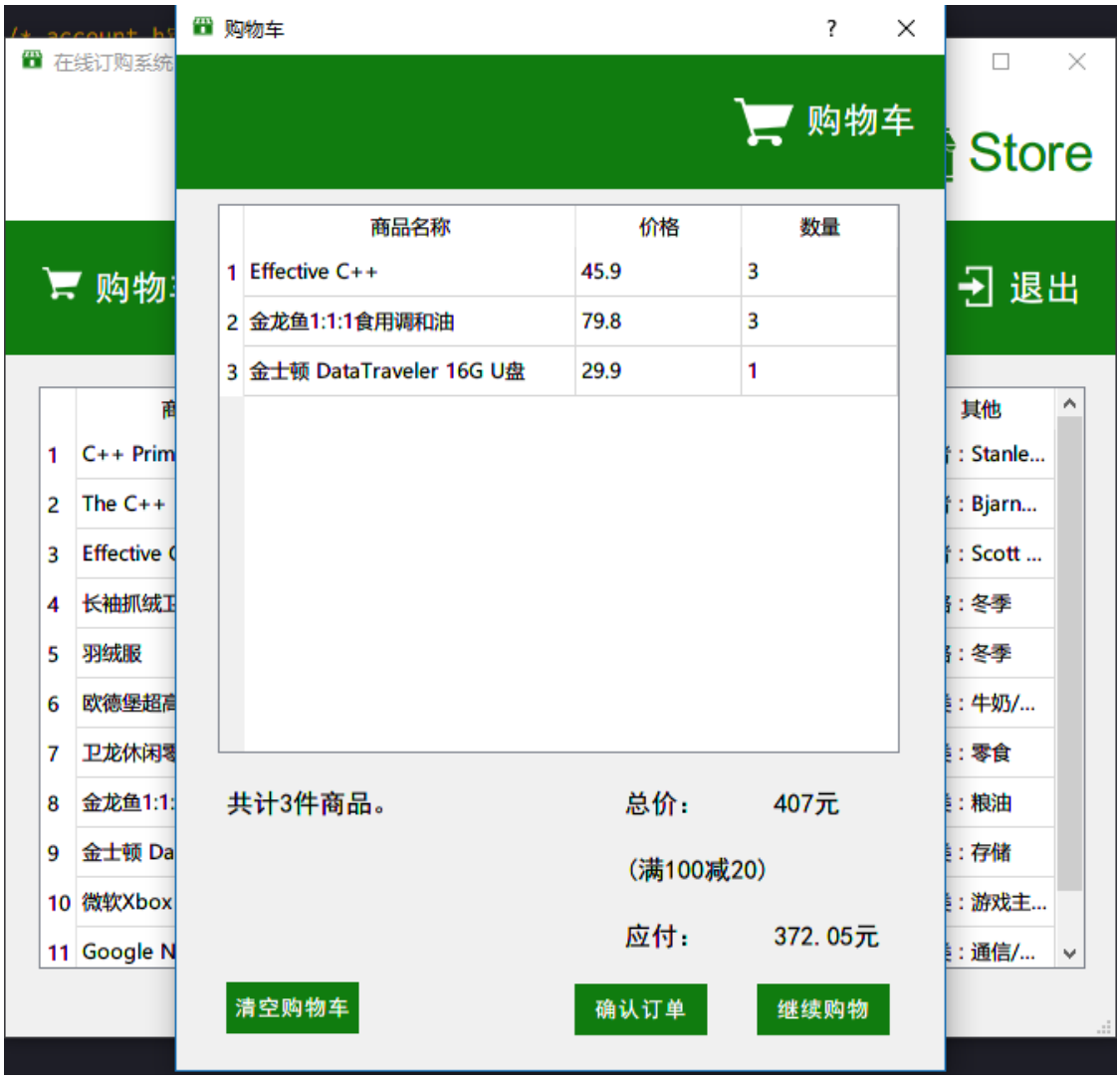
五、 测试结果

1. 在登录界面点击“管理”可以进入后台管理，设置满减活动和各分类优惠比率。

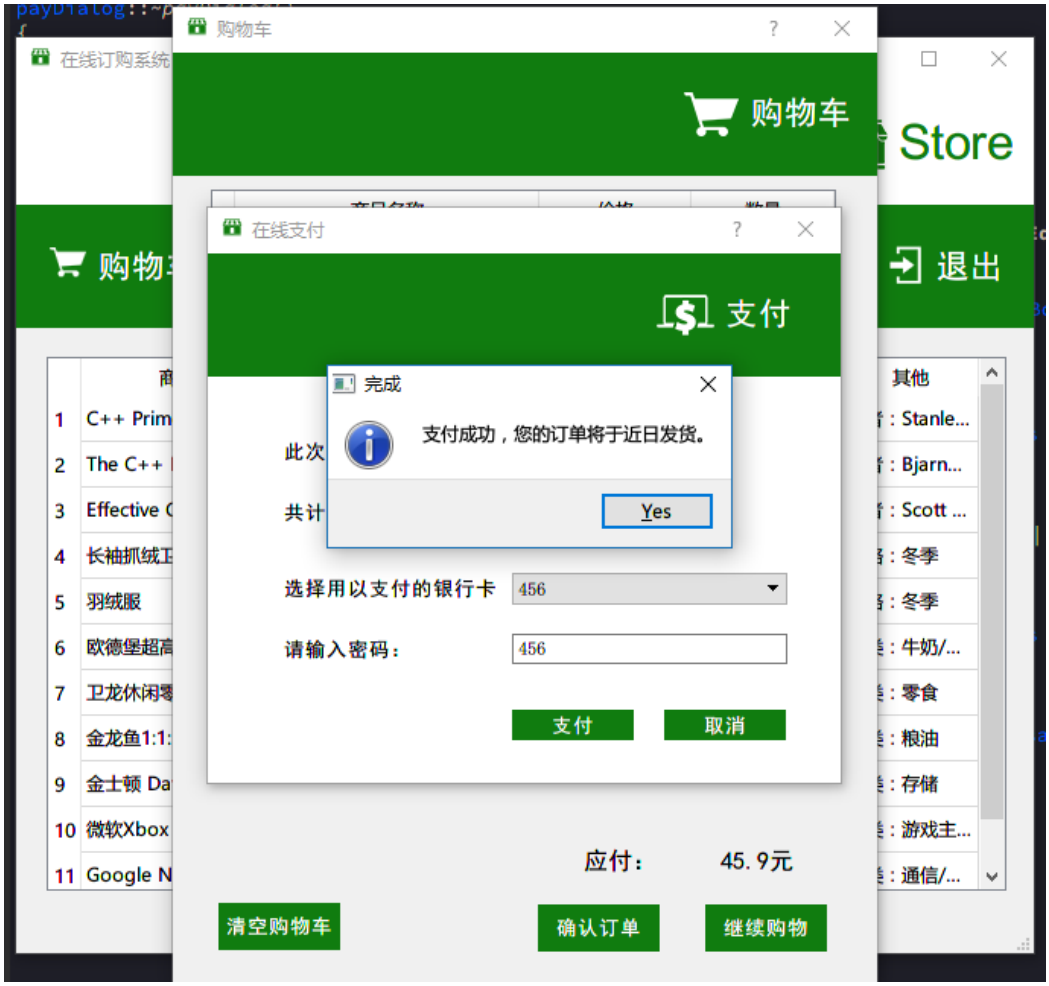
2. 在主界面可以购买商品，如果点击无货商品，将给予提示：

	商品名称	类别	描述	价格	余量	优惠	其他
1	C++ Primer	Book	一本经典的C++参考书。	90.1	36	1	作者：Stanle...
2	The C++ Programmin...	Book	C++之父经典之作。	72.2	71	1	作者：Bjarn...
3	Effective C++	Book	改善程序与设计的55个...	45.9	99	1	作者：Scott ...
4	长袖抓绒卫衣	Cloth	过冬佳选，一件即可。	199	0	0.8	风格：冬季
5	羽绒服	Cloth	纯正鸭绒，值得信赖。	499	0	0.8	风格：冬季
6	欧德堡超高温处理全脂纯...	Food	来自原产地，体验真正奶...	109	4933	1	分类：牛奶/...
7	卫龙休闲零食辣条	Food			338	1	分类：零食
8	金龙鱼1:1:1食用调和油	Food			5	1	分类：粮油
9	金士顿 DataTraveler 16...	Electro			6	0.5	分类：存储
10	微软Xbox One游戏主机	Electro				0.5	分类：游戏主...
11	Google Nexus 6P	Electro	体验原生Android的最佳...	5499	102	0.5	分类：通信/...

3. 将商品添加至购物车后，可以看到购物车应付金额已经正确地处理了满减活动和优惠活动



4. 在支付页面选择结算，支付成功，银行端扣除账户余额，购物车清空，库存相应修改。



题目三 网上支付设计

设计文档

一、 设计任务描述

实现题目三会让你设计的平台像个真正的运行在网络上的平台。题目三要求在题目一、二的基础上支持通过网上支付在电商平台上购物，请至少实现以下功能：

绑定银行卡：支持电商平台账号绑定银行卡（再建一个 database）。同一账号可以绑定多张银行卡，而且银行卡可以属于不同银行。

网上支付：选择绑定的任一银行卡支付电商平台上的订单。

要求 1：绑定银行卡和选择银行卡进行支付时都要求提供对应的银行卡密码。

要求 2：当在电商平台上绑定银行卡或者进行网上支付的时候，请不要直接打开某个文件查找银行卡的信息，因为银行卡信息文件是银行门户系统私有的，电商平台系统无法直接访问，而应该由电商平台系统去向银行门户系统对接数据。请用 socket 通信来传送数据。

要求 3：请做好错误场景的处理，如绑定银行卡失败，支付失败等。

二、 设计任务分析

题目三基于题目一和题目二，在绑定银行卡和结算支付时进行通信。根据题目要求，应当将电商平台作为客户端，而银行作为服务器

端，电商平台通过向银行系统请求数据，银行将处理结果返回至电商平台，电商平台再将结果反馈给用户。在绑定和结算时，电商平台将用户输入的信息发送给银行系统，银行系统接受数据后进行处理，与本地数据库中的数据进行比对，得到结果后将结果反馈至电商平台。而电商平台接收到结果后，根据成功与否，决定接下来的操作。这些操作已封装在题目二中。由于 TCP 连接更为稳定，因而使用了 QTcpSocket 而非 QUdpSocket。

三、 模块划分

按照分析，网上支付共分为两个部分，一部分是银行系统的服务器端，服务器端分为建立服务器、建立 TCP 连接和数据处理模块。另一部分为电商平台的客户端，客户端分为建立连接并发送消息和数据处理两个模块。

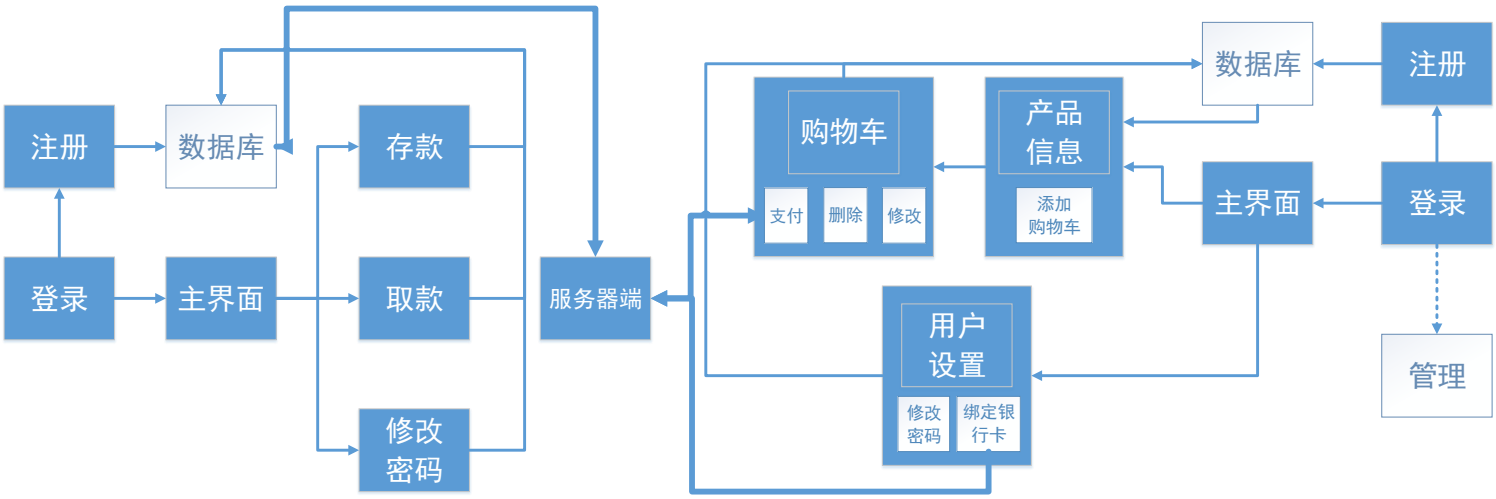


图 1 网上支付的模块划分

四、 数据结构

服务器端定义：

```
class bankServer : public QTcpServer
{
    Q_OBJECT
public:
    bankServer(QObject *parent=0);
    QTcpSocket * clientConnection;

private slots:
    void createConnection();
    void readClient();
};
```

客户端定义：

```
QTcpSocket * socket = new QTcpSocket();
socket->connectToHost(QHostAddress::LocalHost,6665);
```

发送的信息格式：

绑定银行卡时：ID:（账号）;PW:（密码）

在线支付时：PY:（支付金额）;ID:（账号）;PW:（密码）

银行向客户端返回的信息：

绑定/支付成功：YES

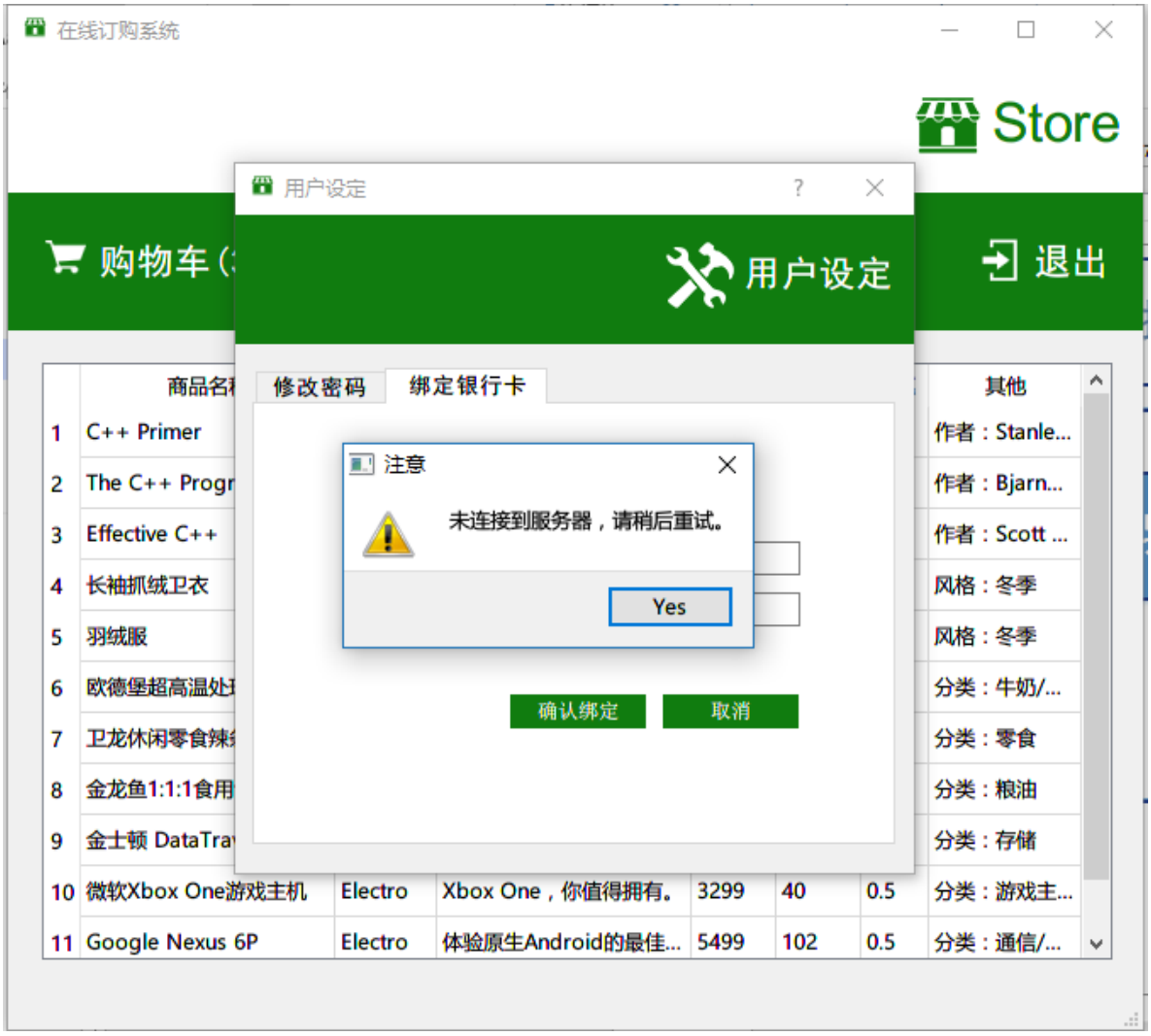
密码错误：WRPW

账号不存在：NOID

余额不足：LKMN

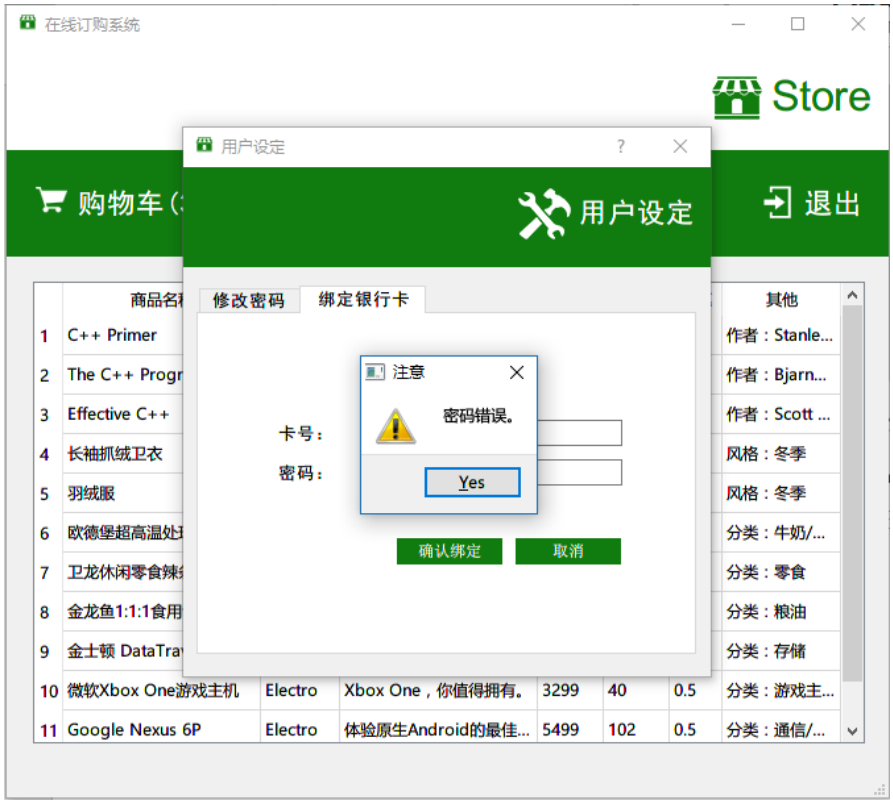
五、 测试结果

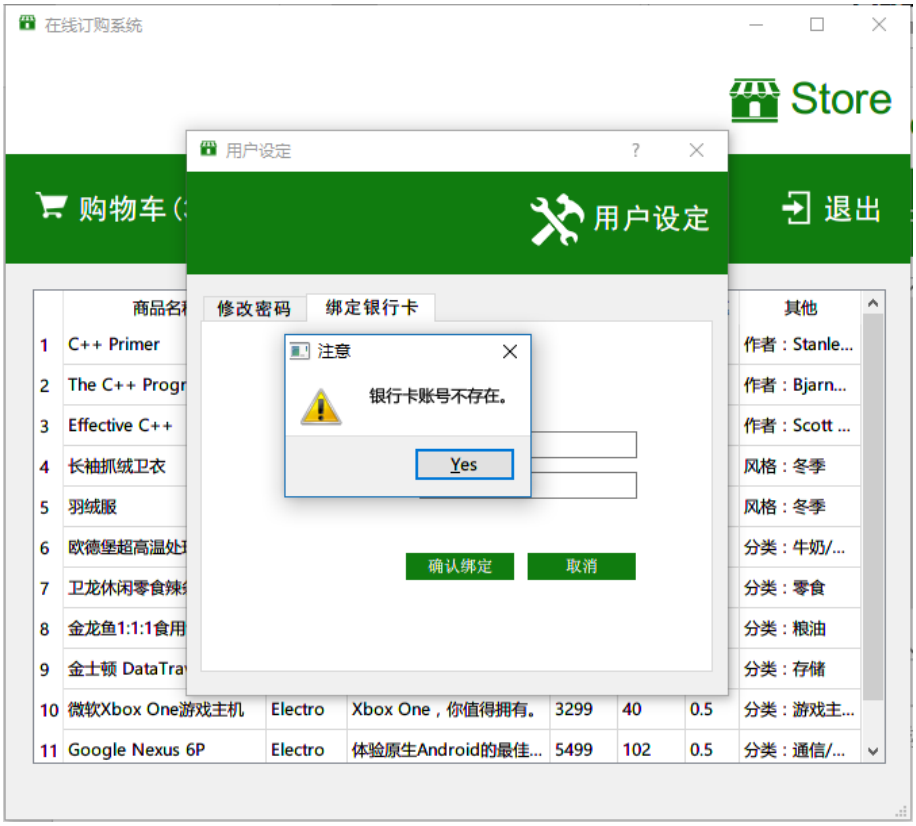
1. 在绑定银行卡/在线支付时，如果银行系统未在运行，将提示“未连接到服务器”。



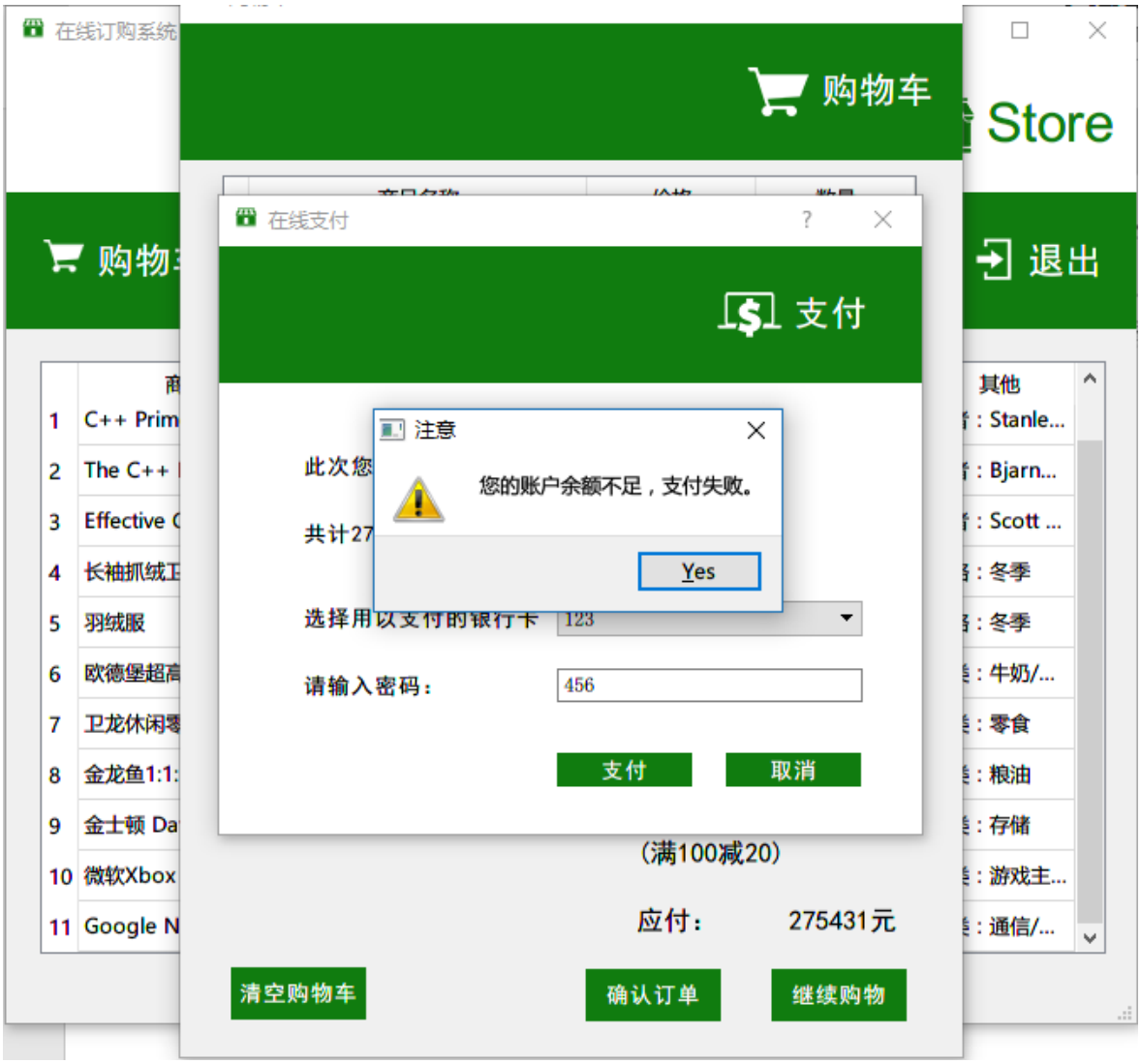
2. 当绑定已经绑定过的银行卡时，会给予提示：

3. 当密码填写错误或账号不存在时，银行系统可以给出相应反馈：

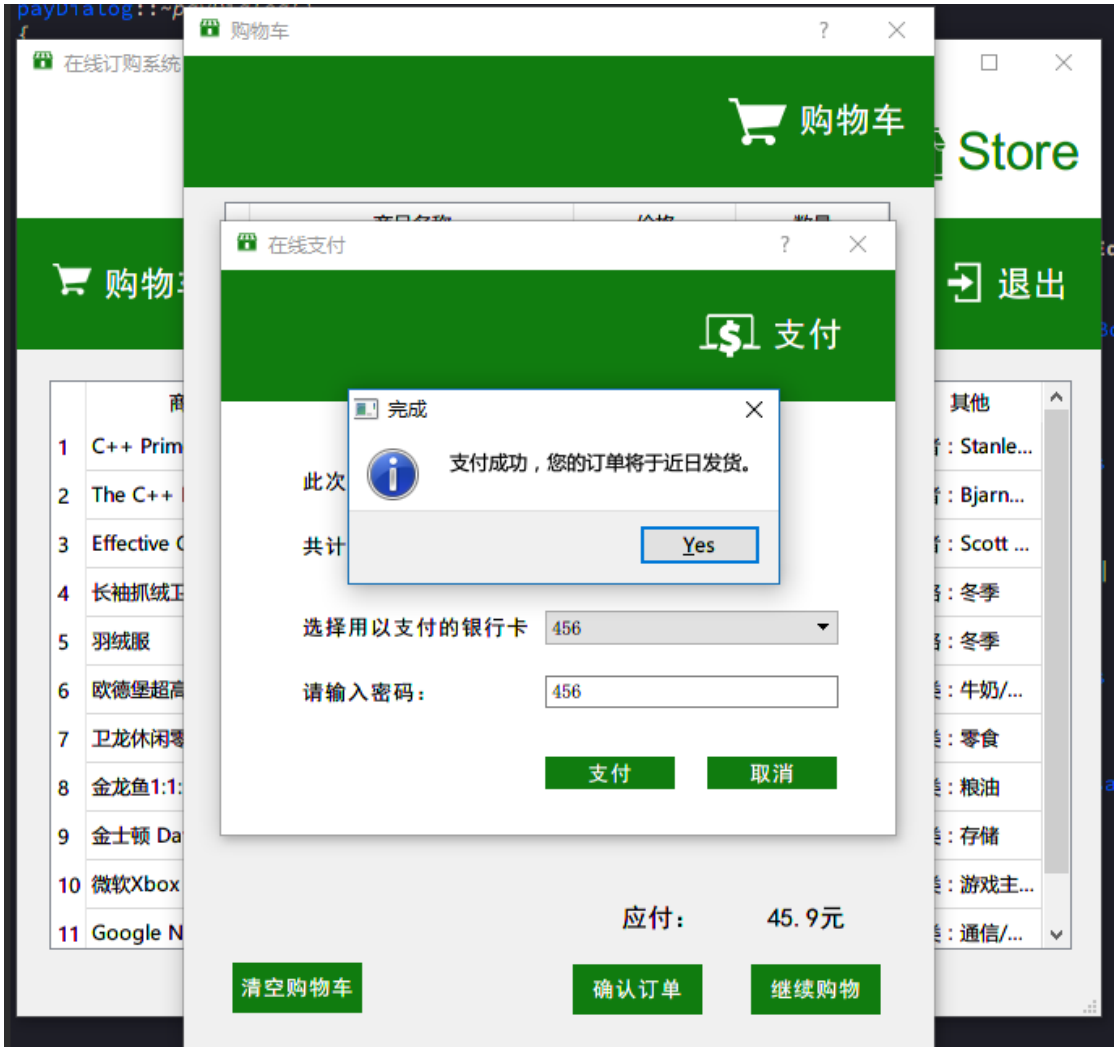




4. 在支付时，如果账户余额不足会正确给予提示：



5. 支付成功时，银行系统将从相应账户中减去购物金额，而电商平台会清空购物车并修改库存。



实验总结与感想

本次实验共分为三个部分，银行系统、电商平台和网上支付系统，三者组成了一个完整的网上购物系统。三个题目的难度逐渐递进，在开发银行系统时主要是对 Qt 图形化和数据库的学习，而电商平台重点在于实现类的继承体系以及各个数据库之间的关系，网上支付则让我们学习并实践了进程间 Socket 通信的实现。这次试验带领我们完

成了一个基础软件的开发过程，使我们通过实践加深了对 C++语言特性的理解以及对软件开发中编程规范和技巧的掌握。当然这次实验中也有一些不足，比如电商平台的界面使用列表形式而非目前电商平台主流使用的图文形式，在进程间通信和数据库存储时为了数据安全性都应该加密等等，这一次实验的不足之处将使我在之后的学习中更注重细节；不过我觉得，在实验过程中不断解决自己遇到的问题、对程序结构的不断思考使我获得了更多的经验。