# 面向对象程序设计与实践(C++)2

## 购物系统程序说明书

- **班级:2015211306**
- **姓名:魏晓**
- **学号:2015211301**
- **班内序号: 18**
- **日期:2017.12.31**

---

---



## 设计背景

- 我们经常会在淘宝、京东等电商平台购买产品，这次编程的任务是做一个类似的电商交易平台。此外，为了实现交易功能，我们还要设计一个银行系统，用于产品的支付。
    - 注：请将所有的类中的数据置为私有，请将不需要被该类外部访问的函数设置为私有；如有良好的界面设计有额外加分

# 第一部分:银行系统的设计

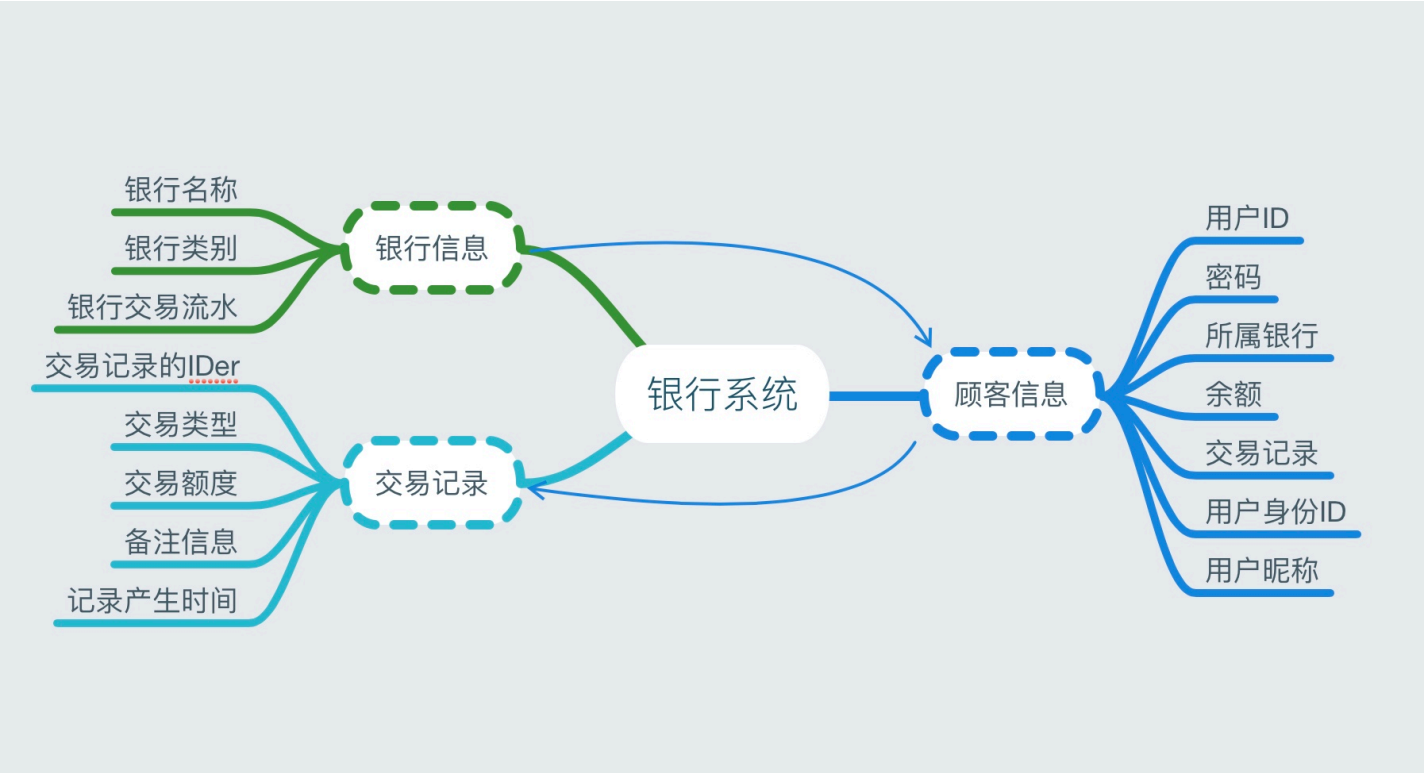
## 设计要求

- 银行:该门户是一个单独的程序，程序要求至少支持以下功能：
    - 注册银行卡，银行可选；
    - 修改银行卡密码；
    - 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
    - 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。
    - 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。

## 类设计

- 根据设计要求,银行至少要包含以下信息
    - 顾客信息
    - 交易记录
    - 银行信息

- 由此画出银行人员结构




- **设置人员父类,又设置顾客子类,方便代码重构** 在实际代码过程中,需求会随着代码的不同阶段产生不同的要求,如果简单只有一个人员类的话重构时就要改动类结构,进而动全身.
- **添加交易记录功能,方便顾客行为记录** 在实际的socket通信中,可能接受到具有危险行为的信息,除了处理不同的异常,还应该记录在记录中,方便管理
- **记录的存储**
  - 记录的构成由操作ID,交易金额,发生时间,附加信息组成.信息要精简内容要全
  - 不同记录存储在vector中,给予stl提供的功能可以有效的降低代码复杂度,同时给代码重构的人提供更加简明的信息
- 使用**stringstream**类进行单行文件的读取和保存,简介有效
- 使用**typedef**使得不同类内的数据类型的可读性与有效性,这种做法保证了当不同对象的数组下标方位不同时,能够自适应没个对象的特征

## 实现方法

- 代码结构-*去除了简单逻辑的成员函数*

```
class sBankinterface{//银行类
```

```
private:
    vector<std::string> bankname;//设置可以选择的银行名单
    std::map<std::string,banker> bankerlist;//id/人员记录
    vector<bankrecord> BTrecord;//银行记录
    const string nowlocal;//当前登陆的顾客
};
```

- 在此基础上构造函数和析构函数的作用就是把运行期间内存中的信息毫无保留的保存,并在程序运行开始的时候进行还原,在文件中对信息进行优化,方便管理员可以修改文件
- 设置单独的run函数方便代码重构,在未来添加服务器端的时候就可以进行选择,并且能对特定的客户进行服务,使程序在实现阶段可以用多种入口,在程序构建的时候提供了更多的借口
- 查看自己的信息修改信息,为了保障银行安全,标示id由根据身份ID,姓名系统自动生成,可以保障
- 完善的**try-catch**机制可以在运行的过程中持续关注代码运行的质量,并且在发生错误的时候有完善的提示信息

## 设计效果

- 实现了所有基础设计要求功能
- 完善了银行的基础存取款功能
- 增加了交易记录分析功能,用户可以查看自己的记录,管理员可以查看所有的记录
- 增加一个额外属性,增强了系统的鲁棒特性

# 第二部分:电商平台的设计

## 设计要求

- 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。）
    - 浏览平台产品信息。
    - 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。
    - 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。
        - 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。
    - 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（<> 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。
    - 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）。

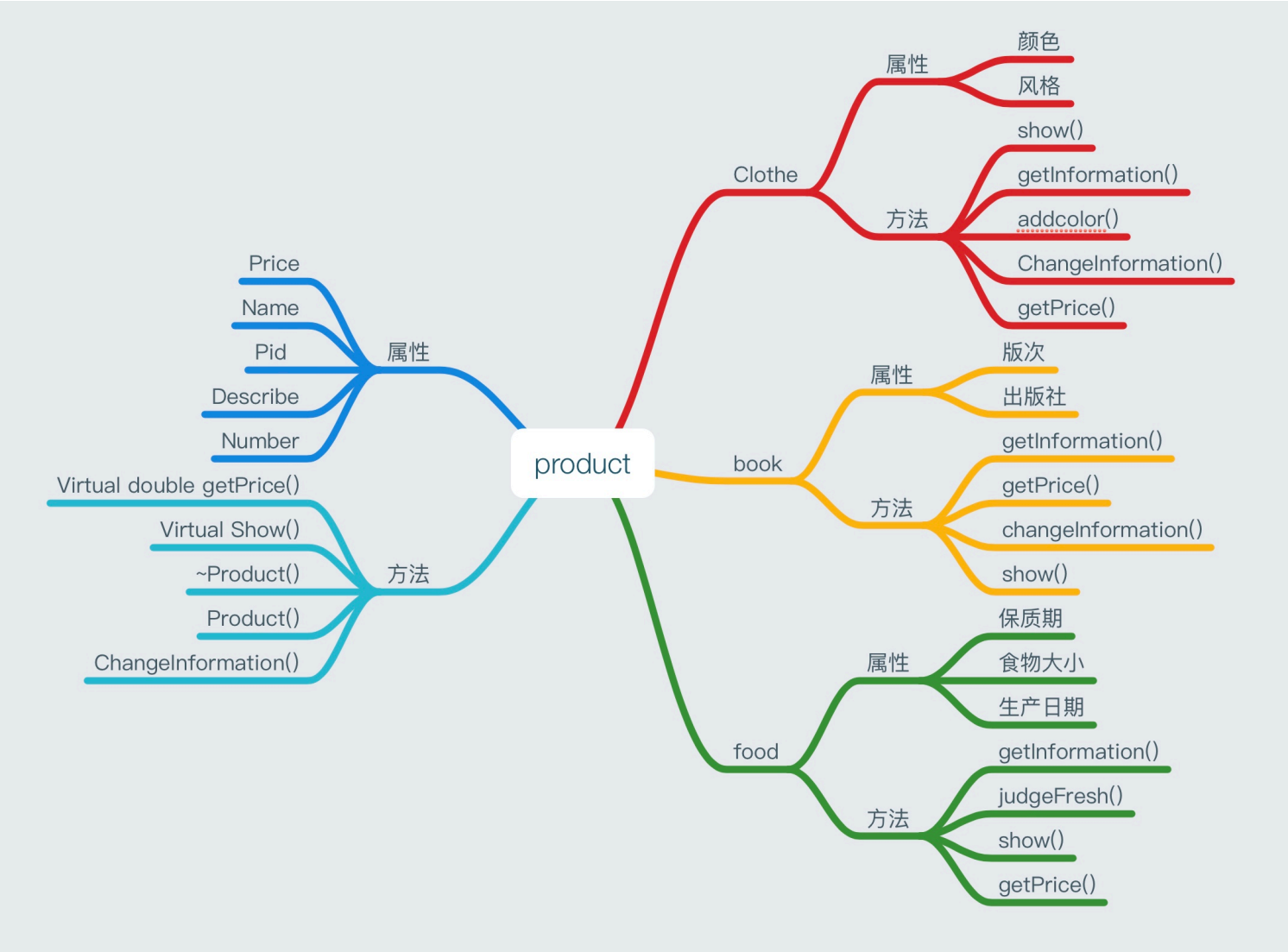
# 类设计

- 电商的基础就是商品,所以商品类的设计是这个题目的基石,商品主要包含三个种类,食物/书籍/衣服,他们之间共同的元素是
    - 商品名称
    - 商品价格
    - 商品ID
    - 商品描述
    - 商品数量
    - 有如下方法
        - 获得每一个属性的值和改变每一个属性的值的方法
        - 虚函数getPrice()
        - 构造函数和析构函数
- 食物产品设计:
    - 继承商品父类
    - 有如下特殊属性
        - 生产日期
        - 保质期
        - 食物大小
    - 有如下特殊方法
        - 获得每一个属性的值和改变每一个属性的值的方法
        - 判断食物是不是超过保质期
        - show()函数,展示食物信息
- 衣服产品设计:
    - 继承商品父类
    - 有如下特殊属性
        - 衣服风格
        - 衣服颜色

    - 有如下特殊方法
        - 获得每一个属性的值和改变每一个属性的值的方法
        - 添加和显示颜色
        - show()函数,展示衣服信息
- 书籍产品设计:
    - 继承商品父类
    - 有如下特殊属性
        - 出版社
        - 版本号
    - 有如下特殊方法
        - 获得每一个属性的值和改变每一个属性的值的方法
        - show()函数,展示书籍信息

---

## 产品类

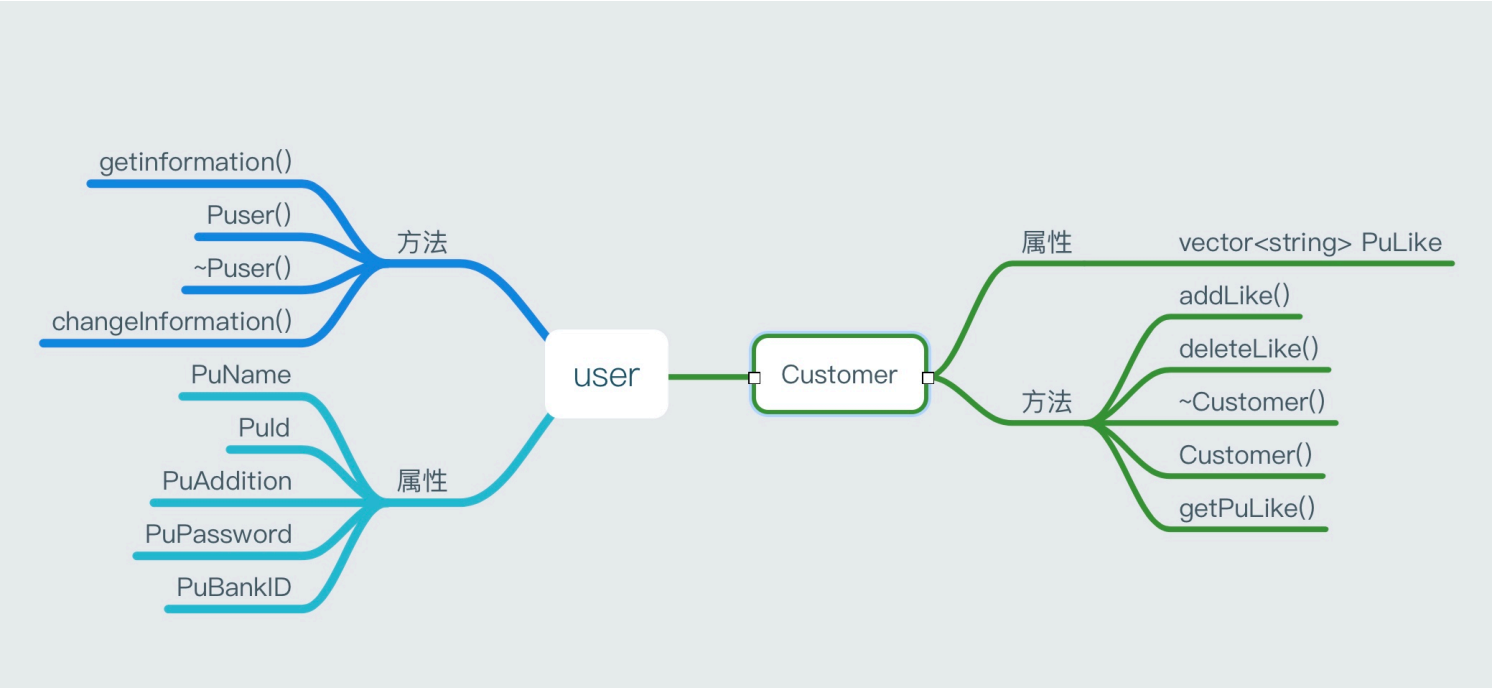


- 在父类product中声明了价格,名字,身份标示,描述,数量这些商品的必须信息,并且**全部数据采用了private属性**,再实现所有元素的get和set函数,这样既做到了良好的数据隐藏,又使程序有了很好的访问性
- 这是基础的物品的结构,在设计结构的时候遵循的原则就是尽量的数据隐藏,在具体实现的过程中,除了设置基础的增删改查外还额外增加了一些特殊的固有方法,比如food属性的方法
    - void Pfood::judgeFresh()
- 在基类中写了**虚函数getPrice()**,因为衣服和book都是一件件售卖的,食物是根据foodsize灵活售卖的,这也符合实际生产中大部分食物的售卖方式,在具体的实现中,在food子类中重写getPrice(),增强了通用性
- 在构造函数中,使用了无参数构造和有参数构造,方便在最开始读区文件的时候首先无参数声明,然后逐步读取参数,并进行有参数构造,最后压入vector
- 在衣服子类中添加了颜色属性和风格属性,在实现上使用String存储,方便一次性读写保存,降低了复杂度O(n)到O(1),在实际使用的时候预先声明一个vector temp,然后使用split函数,增强了系统的鲁棒性能,同时这样就在降低复杂度的同时增强了程序的可扩展性

```
std::vector<std::string>temp;
StringSplit(color,temp,"/");
```

- 在Pid的产生中使用了**RSHash**函数,如果发生冲突就再hash,直到无冲突.使得pid起到身份标示的作用
- 在基类中构造**纯虚函数show()**因为各个子类的展示特性不同,比如衣服要展示颜色,要展示风格,食物要展示生产日期,保质期,书要展示出版社,版次信息.在父类中声明纯虚函数,要求每一个子类都必须实现show(),增强了通用性

## 员工类




- 代码结构

```
    class Puser
{
    public:
        Puser();
        Puser(string na,string id,string ba,string ad,string pa);
        virtual ~Puser();
    private:
        string Puname;
        string Puid;
        string PuBankid;
        string Puaddition;
        string Pupassword;
};
class customer:public Puser
{
private:
    vector <string> Pulike;
public:
    customer(vector <string> &li,string na,string id,string ba,string ad,string
 pa);
    ~customer();
    void addlike(string productid);
    void deletelike(string productid);
    void getPulike(vector <string> &ans);
};
```

- 在产生姓名,Password,银行卡的时候都使用了正则检查,确保输入进系统的数据都是合法数据,在产生Id的时候使用的是Hash姓名的方式产生,防止因为id而泄漏信息
- 单独写员工子类,就是提高在后期重构的时候方便添加特殊属性,方便对关键信息的隐藏
- 在前期使用用户的过程中,可以在附加信息备注是不是管理员

## 记录结构

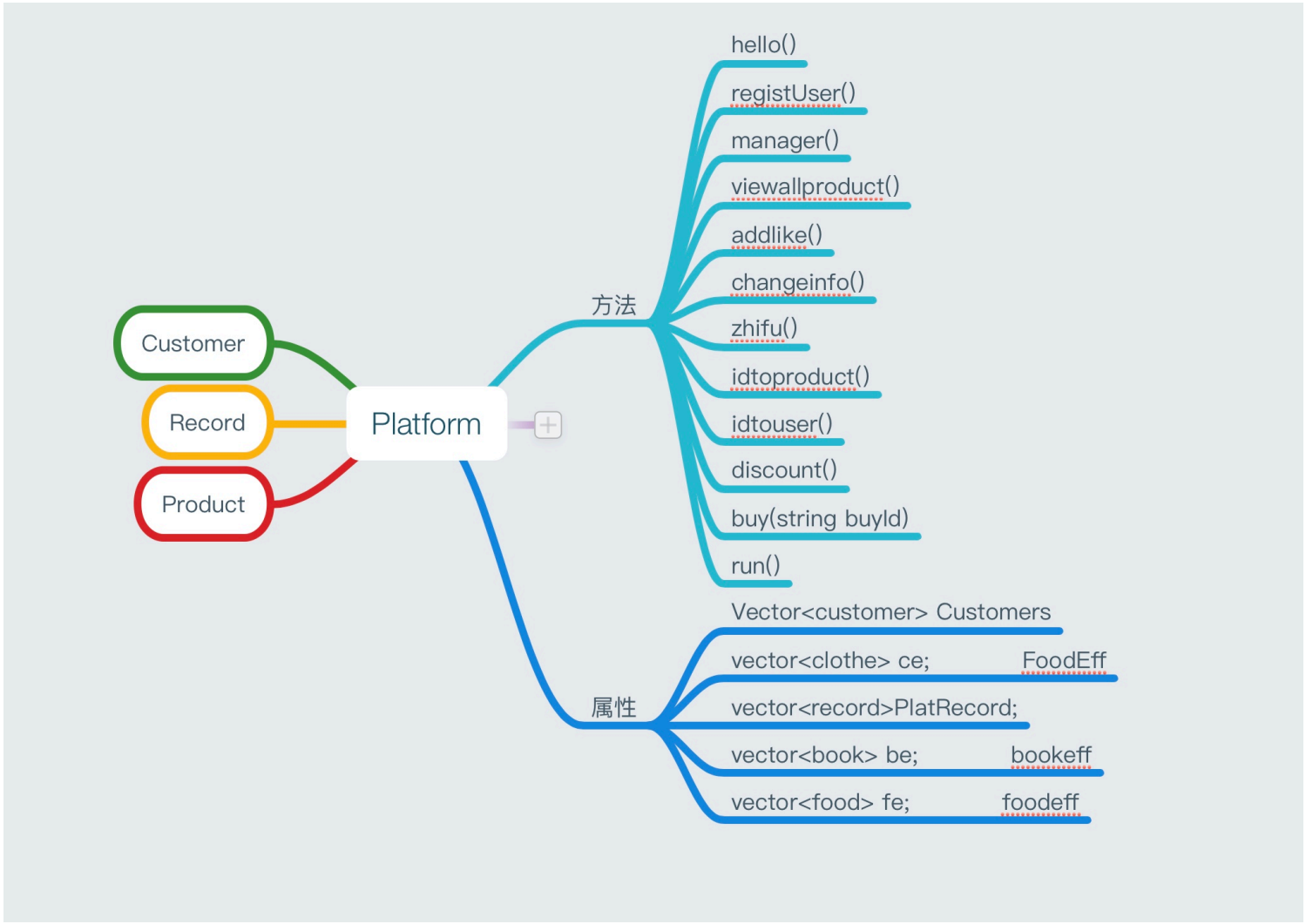

- 代码结构

```
typedef struct jiaoyirecord{
    string Userid;
    string goodsid;
    time_t recordtime;
    string addi;
    jiaoyirecord(string u,string g,time_t t,string a)
    {
        Userid=u;
        goodsid=g;
        recordtime=t;
        addi=a;
    }
```

```
    bool operator <(const jiaoyirecord &a)
    {
        if(a.recordtime<recordtime)
            return false;
        else
            return true;
    }
}record;
```

- 记录采取结构存储的方式,主要考虑到两点事实.
  - 在程序运行的时候时刻产生记录,必须时刻创建记录结构,并进行保存.使用结构存储符合我们对记录的具体要求
  - 在记录列表中,我们会经常遍历列表来检索出特定的记录,进行处理,做出数据分析.如果使用构建对象的方法,会大大增加访问时间
- 在记录的属性中严格记录了发生动作的id,交易的规模,发生的时间由系统自动生成,增加了记录的安全性,有效性,减少了假记录的产生

## 平台类




- 代码结构

```
class Pplatform
{
    public:
        Pplatform();
        ~Pplatform();
        void registuser();
        void hello();
        void run();
        void buy(string buyid);
        void manager();//ZHIYOU GUANLI YUAN YOU NENG LI
        void discount();
        void ViewAllProduct();
        void addlike();
        int idtouser(string id);
        int idtoproduct(string id);
        void changeinfo();
        bool zhifu(double localjine);
```

```
    private:
        vector<customer> customers;

        vector<Pfood> fe;// ≥Œ̂
        double Pfoodeff;//'€ø€œμ ˝
        vector<Pclother> ce;//"¬∑˛
        double Pclothereff;
        vector<Pbook> be;// È
        double Pbookeff;

        vector<record> platrecord;
        int local;
};
```

- 设计思路
  - 平台类必须统筹所有资源,所有的交易动作都是在这个平台上完成的,不管是最开始的注册还是以后的文件存取,都是在平台上完成的
  - 平台是直接面向main函数的,其他的类和数据结构都可能通过平台被main函数访问,所以在确定属性的时候必须考虑清楚属性的作用域,做到尽量的数据隐藏
  - 在platform的构造函数中从文件中读取数据并且保存在内存中,在析构函数的时候再把内存中的信息按照一定的规则保存在文件中,在文件保存中添加部分简单的结构,在不增加复杂度的情况下增加了文件的可读性
  - 在文件读取的时候使用**stringstream**简单有效,有效避免了无限读入的情况
  - 封装部分简单的结构,比如hello函数的作用就是完成循环以前的登陆部分,简化了run函数结构,增强了代码的可读性
  - 增加管理部分,因为整个程序运行的过程中会产生大量的记录,增加管理类可以管理这些记录,并得到每月统计.扩展了程序的功能
  - 除了统计功能还增加了上架商品,管理顾客的功能

# 设计效果

- 实现了设计文档中的**所有基础要求**
- 根据属于隐藏要求,设定私有属性,确定私有方法和公有方法
- 在设计数据元素的时候留下一个额外属性,方便以后代码重构
  - product.addition
- 使用map代替vector,使得数据结构在存储大量数据的时候复杂度降到log N
- 给三个类都添加了一个打折系数,并且在默认1,在管理员中可以设置各个品类的打折系数
- 在基础类中添加了满减金额,和起始金额,并且通过管理员功能发布不同的活动,并且管理员也具

有普通的功能

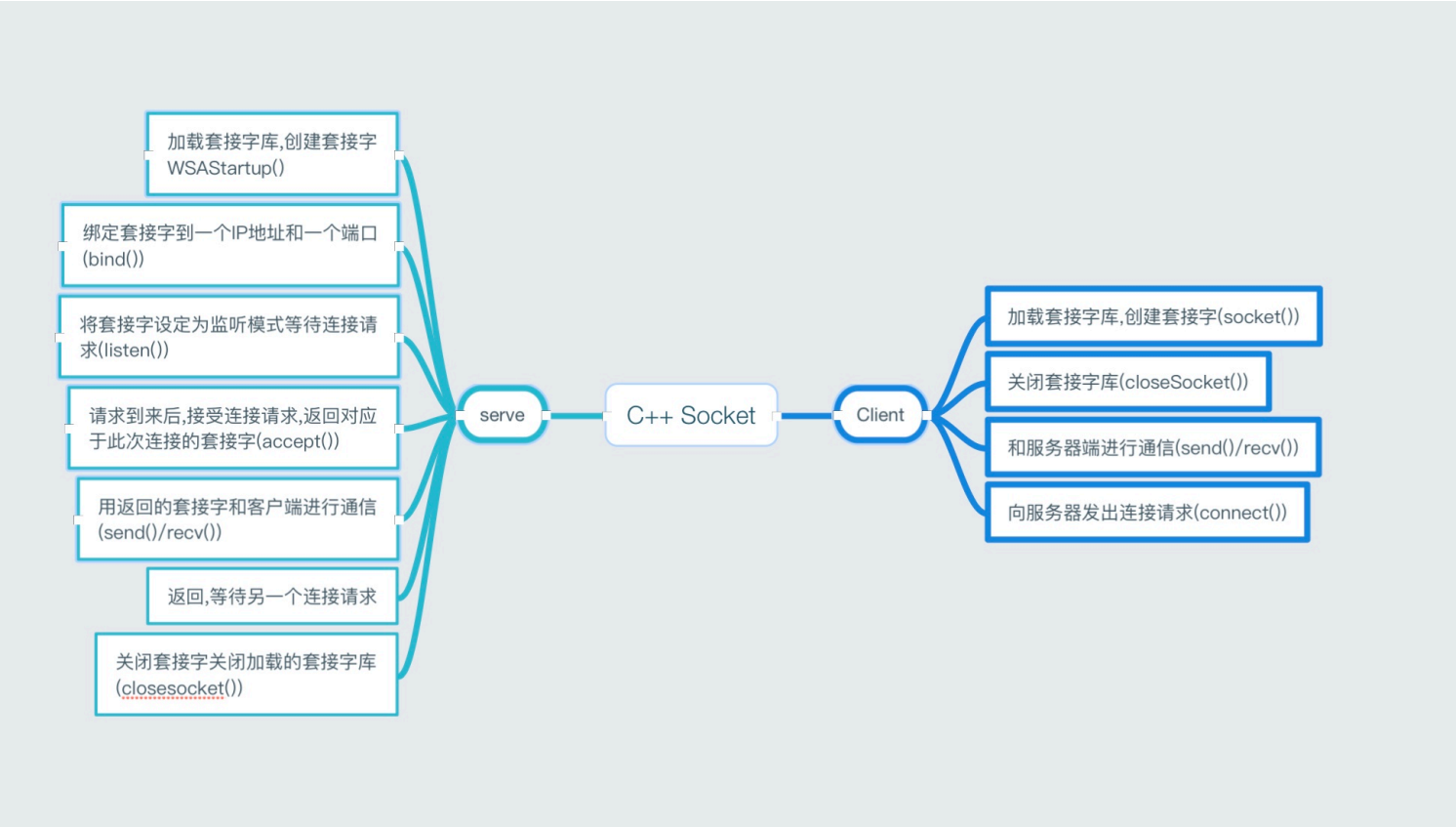
- 设置独立zhifu函数,在第二阶段把这个函数做空,在支付阶段直接建立socket链接,判断支付是不是成功
- 在完成buy函数的时候,简化接口,因为这个购物函数会有很多的用处,可能用在不同的种类上.只要输入id,这个函数就会自动判断这个物品的种类,并调用相应的信息,在支付成功之后就可以修改信息了,并且产生相关的记录.
- 显示所有产品的函数中可以显示特定的种类,使用Split函数区分,能够在提供足够强的容错能力的基础上精简了程序的结构

# 第三部分:网络支付的设计

## 设计要求

- 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。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。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。
    - 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。
    - 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。
    - 请做好错误场景的处理，如绑定银行卡失败，支付失败等。

## 实验思路




- 在前两个基础程序上,增加了以下功能:
  - **基于Socket API 为软件增加了网络访问的功能,即实现C/S模型**
  - **基于Select()函数,增加多用户访问和操作结果的实时同步**
  - **提升安全性,保障服务器端数据安全**
- 在程序代码中,为了实现服务器端对客户的监督,则使用select,这样就能检测当前是谁发送了请求,如果在这个过程中产生危险动作,就可以记录
- 对于确认信息,回答的信息是可以理解的,计算机通过判定返回字符串的种类来确定支付是不是成功
- 有两种方式实现网络功能
  - 服务器端将数据传给用户,用户在本地直接调用付款与购物交互.
  - 服务器端不用把数据发送给用户,只处理结果,并传送结果
    - 显然,第一种方法易于实现,只需要在服务器端将对应的数据传给用户,用户在本地实现付款等功能但是这个功能有两个严重的问题:
    - 不易与实现多用回操作结果的实时同步
    - 无法保证用户操作的合法
  - 所以我使用第二种方式,具体的实现过程已经在上图中说明了,这里简要概括,在客户端如果需要支付请求,建立套接字链接,在远端处理数据,并以特定的形式返回处理结果
- **服务器端**
  - 服务器端部署在bank程序里,在创建套接字后,就一直监听6000端口,如果端口到来请求,接受

请求,判断来的地址,在recv到char*后把这个char*转换为string并且切割为三个子段并且分别计算正则是不是符合接受信息的规范,并且在程序中如果成功转账,就返回IP+success否则就返回IP+fail

- **客户端**
  - 服务器端部署在shop程序的zhifu()函数内,因为购物平台在平时运行的时候并不需要一直去发送信息,只有在产生交易的时候才会调用支付函数给固定IP的固定端口发送一段经过加密之后的信息,所以在具体的执行过程中,如果没有调用支付函数,那么socket连接关闭,只有在输入的信息经过信息检查和压缩之后再创建连接,发送massage,等待回执,并且在判断回执的内容进而做出不同的动作

# 代码重构

- **简化了try-catch语句**原先在每一个操作上独立的加上try-catch时候代码有太多冗余度,也不能保护输入输出部分,最后就直接加在了run()函数上,降低了冗余代码,增加了整个程序的安全性
- **借鉴MVC模型**分离了处理客户交互的子程序和进行数据处理的子程序
- 增加了管理模块,方便上架满减活动和折扣活动,进行上架处理.并且将这些功能单独分离到管理模块

# 设计效果

- 设计实例如下
  - 在创建Socket的子函数中,我复用了深入理解计算机系统一书中的Lab代码,在具体的使用环境如下

```
WORD wVersionRequested;
WSADATA wsaData;//加载
int err;

wVersionRequested = MAKEWORD( 1, 1 );

err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 )
{
    return -1;//错误码-1, err
}

if ( LOBYTE( wsaData.wVersion ) != 1 ||HIBYTE( wsaData.wVersion ) != 1 )
{
    WSACleanup( );//释放 socket
```

```
    return -2;
}
SOCKET sockSrv=socket(AF_INET,SOCK_STREAM,0);

SOCKADDR_IN addrSrv;
addrSrv.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
addrSrv.sin_family=AF_INET;
addrSrv.sin_port=htons(6000);//监听端口6000

bind(sockSrv,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));//绑定

listen(sockSrv,5);
int jianjian=1;
SOCKADDR_IN addrClient;
int len=sizeof(SOCKADDR);
while(true)
{
    SOCKET sockConn=accept(sockSrv,(SOCKADDR*)&addrClient,&len);
    char sendBuf[50];
    //sendBuf="success";
    //sprintf(sendBuf,"%s",inet_ntoa(addrClient.sin_addr));
    //send(sockConn,sendBuf,strlen(sendBuf)+1,0);
    char recvBuf[50];
    recv(sockConn,recvBuf,50,0);
    string str=recvBuf;
    vector <string> ansq;
    SplitString(str,ansq,"/");
    //cout<<str<<endl;
    string usr=ansq[0];
    cout<<"pay id:"<<usr<<endl;
    string pas=ansq[1];
    cout<<"id-password:"<<pas<<endl;
    string num=ansq[2];
    double sum=stringToNum(num);
    cout<<"account:"<<sum<<endl;
    int ans=quicktransfer(usr,pas,"2521616198",sum);
    if(ans==1)
    {
        sprintf(sendBuf,"%s-success",inet_ntoa(addrClient.sin_addr));
        send(sockConn,sendBuf,strlen(sendBuf)+1,0);
    }
    else
    {
        sprintf(sendBuf,"%s-fail",inet_ntoa(addrClient.sin_addr));
        send(sockConn,sendBuf,strlen(sendBuf)+1,0);
```

```
    }
    //printf("%s",recvBuf);
    //cout<<str<<endl;

    closesocket(sockConn);
    if(sum<0)
        break;
}
return 0;
}
```

- 在客户端创建了客户端,每次支付请求都发送套接字

```
WORD wVersionRequested;//初始化
    WSADATA wsaData;
    int err;

    wVersionRequested = MAKEWORD( 1, 1 );//协议选择

    err = WSAStartup( wVersionRequested, &wsaData );//定义异常
    if ( err != 0 ) {
        return false;
    }

    if ( LOBYTE( wsaData.wVersion ) != 1 ||HIBYTE( wsaData.wVersion ) != 1 ) {//建立连接否则返回false
        WSACleanup( );
        return false;
    }
    SOCKET sockClient=socket(AF_INET,SOCK_STREAM,0);

    SOCKADDR_IN addrSrv;//声明socket结构体
    addrSrv.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");//ip地址
    addrSrv.sin_family=AF_INET;//协议选择
    addrSrv.sin_port=htons(6000);//端口选择

    connect(sockClient,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));//建立连接

    send(sockClient,fasong,strlen(fasong)+1,0);//socket发送
    char recvBuf[50];
    recv(sockClient,recvBuf,50,0);//接受对面的返回
    printf("%s\n",recvBuf);
    closesocket(sockClient);//关闭socket端口
    WSACleanup();
```

```
    if(recvBuf[10]=='s'&&strlen(recvBuf)==17)
        return true;
    else
        return false;
}
```

## 设计成果

- 实现了**所有的基础功能**
- 通过修改inet_addr属性可以和任何地址建立连接.
- 在发送的时候首先对要发送的信息进行了压缩,只需要一次发送就可以传输密码,账号,交易金额等信息.、
- 服务端收到信息之后,首先处理收到的信息,然后在发送回执信息的时候也是发送一个压缩之后的信息,信息里面包含了是否成功和客户机的名称.
- 在服务端能够记录访问记录,访问类型,交易的具体信息,方便管理员管理.
- 每一次的信息交流都会记录在双方的记录里,方便双方检查日志,并且在后期提供管理的可能.
- 在客户端运行结束的时候会在析构函数里向服务端发送一个关闭请求,实现了关闭客户端,服务端自动挂起的功能

## 实验感想

- 这次实验于我而言是特别曲折的,最开始在mac平台上写的代码,后来遇到无法解决的问题,又把代码迁移到windows,使得我的节奏一下子放缓了好多
- 在迁移的过程中使我意识到了代码通用型的重要性.多使用typedef可以有效解决
- **代码是给人看的,只是偶尔在计算机上执行**,在开发完前两个个几周再开发第三个的时候,如果没有合理的开发日志看起来是非常缓慢的
- 在后期开发socket的过程中,深刻认识到了进程间通信的重要性,要熟悉通信协议才能在基本功能的基础上挖掘自己需要的功能

# 附录

### 开发日志

- 2017/11/17,Xcode,看要求,设计银行数据结构
- 2017/11/18,Xcode,看要求,设计银行数据结构,确定属性,写输入输出
- 2017/11/19,xcode,写输入输出,写功能函数

- 2017/11/24,xcode,写功能函数,运行,调试,失败,总是输出丢失部分数据
- 2017/11/25,Codeblocks,在win重构,调试,设计商店
- 2017/12/01,Codeblocks,调试银行,确定商店属性,代码结构
- 2017/12/02,Codeblocks,封装第一个银行,看Socket知识,写产品数据结构
- 2017/12/09,Codeblocks,写平台类,写输入输出,调试输入输出,写商店的实验报告
- 2017/12/10,Codeblocks,写输入输出,写功能函数.遇到头文件重复定义问题.
- 2017/12/15,Codeblocks,调试商店,重新看设计要求,列出了修改条目
- 2017/12/16,Codeblocks,完善功能,开始检查代码风格,不清楚的地方打注释
- 2017/12/22,Codeblocks,准备socket,开始写socket,尝试在独立文件中首先实现socket通信
- 2017/12/23,Codeblocks,把可以通信的两个socket代码段加入到事先预留好的zhifu函数和监听函数,并且调试bug
- 2017/12/26,Codeblocks,再次看要求,增加功能模块,写实验报告,重构try-catch模块
- 2017/12/28,Codeblocks,再次调试三个程序,准备基础的实验报告,一起验收.
- 2017/12/31,Codeblocks,写实验报告,准备打包

---

- *tips 在参考了老师您的要求和学长学姐的优秀实验报告后,很认同实验报告不是长篇累牍的粘代码贴图片,而是简洁有效的说明实现方法,程序功能,所以有了这遍篇幅相对比较短小的实验报告,希望得到您的肯定,如果有任何疑问都可以联系我.13051957575*