综合实验：物流管理系统设计与实现报告

滕宇航 2020211659

目录

1 引言

在本次课程设计中，我将实践面向对象编程的设计思想，设计并实现综合物流管理系统，并且实现如下基本功能：

- 实现描述用户，快递员的各类状态的类结构体系，模拟用户对系统的不同操作，如登录，充值，修改密码等。
- 实现物流管理系统的基本功能，如发送快递，快递揽收，接收快递等。
- 实现一个客户端/服务端模型，实现用户的登录，注册，保存用户和快递信息。

在课程设计中，我实现了题目要求的物流管理系统（以下简称“系统”）。系统除了实现了题目所要求的所有基本功能以外，还实现了如下额外功能：

- 使用Qt实现的简洁的TUI命令行界面和简明扼要的操作手册，如图所示。
- 自实现了简单的http解析并用于客户端-服务端的通信中，适用于低并发低负载场景，并且实现了简明美观的API接口

- 使用TCP实现了客户端-服务端的通信，使客户端和服务端的连接更加稳定，不容易出现丢包等情况；使用上述http解析API进行http通信，提升了开发效率。
- 实现了基于JWT（Json Web Token）的用户鉴权方式。在用户注册和登录模块中，使用Hash存储用户密码，保证了用户安全性。
- 在类设计方面，采用了层次化的类的设计方法，将网络相关模块抽象为一个独立的类，其他的类进行自上而下层次性的设计，提高了代码的可复用性。

```
06:31:34 ❯ .\delivery.exe
Usage: E:\school\OOP-lesson\delivery-3\out\build\x64-Debug\delivery.exe [options]

快递任务管理系统

Options:
  -?, -h, --help     Displays help on commandline options.
  --help-all         Displays help including Qt specific options.
  -v, --version      Displays version information.
  -s, --server       server mode
  -c, --client       client mode
  -l, --local        local mode
  -p, --port <port>  set port for server. Default 3010
```

在本报告中，我将首先对系统实现的各种功能进行展示，随后对各个功能模块的设计和实现进行展示和说明，之后说明系统开发的环境和工具以及测试正确性的方式，最后对本次课程设计进行总结和展望。

2 功能展示

2.1 基本功能

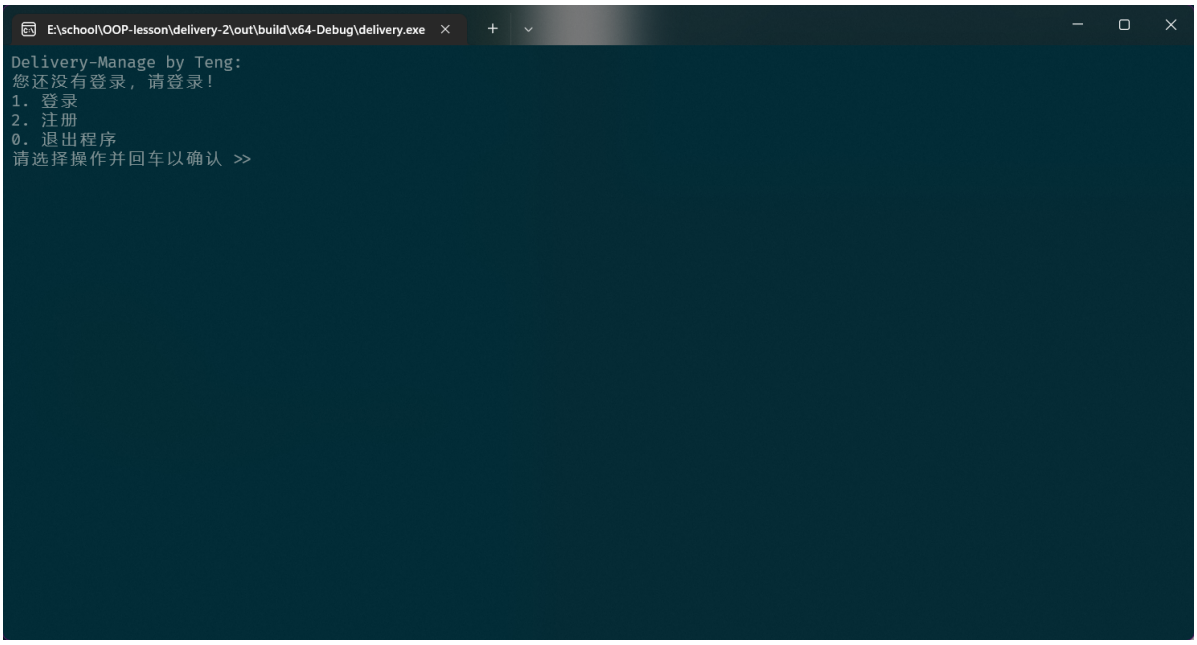
2.1.1 命令行界面


```
06:31:34 ❯ .\delivery.exe
Usage: E:\school\OOP-lesson\delivery-3\out\build\x64-Debug\delivery.exe [options]

快递任务管理系统

Options:
  -?, -h, --help     Displays help on commandline options.
  --help-all         Displays help including Qt specific options.
  -v, --version      Displays version information.
  -s, --server       server mode
  -c, --client       client mode
  -l, --local        local mode
  -p, --port <port>  set port for server. Default 3010
```

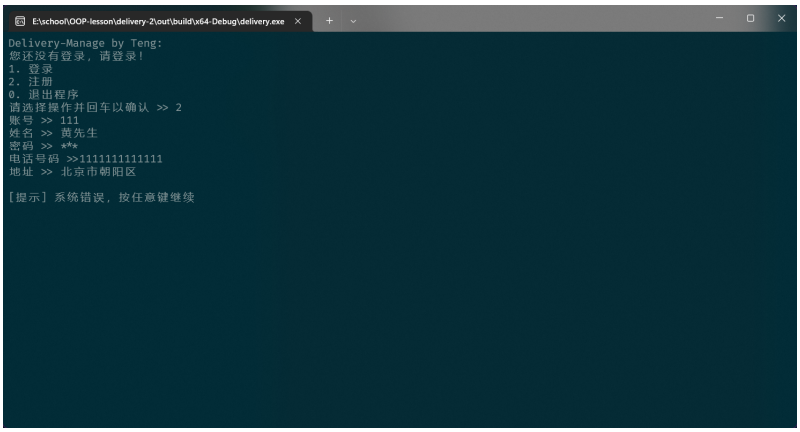

如图所示，当运行该系统且未添加任何选项时，程序会默认打印出命令行帮助界面并退出程序。该系统提供了如下命令行选项供使用者进行调用：

- `-h --help` 打印帮助界面
- `-v --version` 打印版本信息
- `-s --server` 以服务端模式运行
- `-c --client` 以客户端模式运行
- `-l --local` 以本地模式运行(没有添加网络功能时实现的模块)
- `-p --port <port>` 给服务器指定端口号

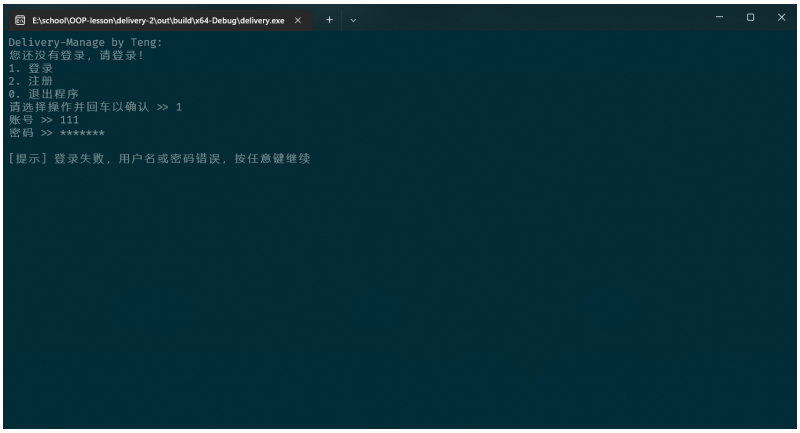
2.1.2 注册登录界面



上图给出了没有登录时可以进行的操作，登录后可以进入下面不同角色所显示的对应界面。



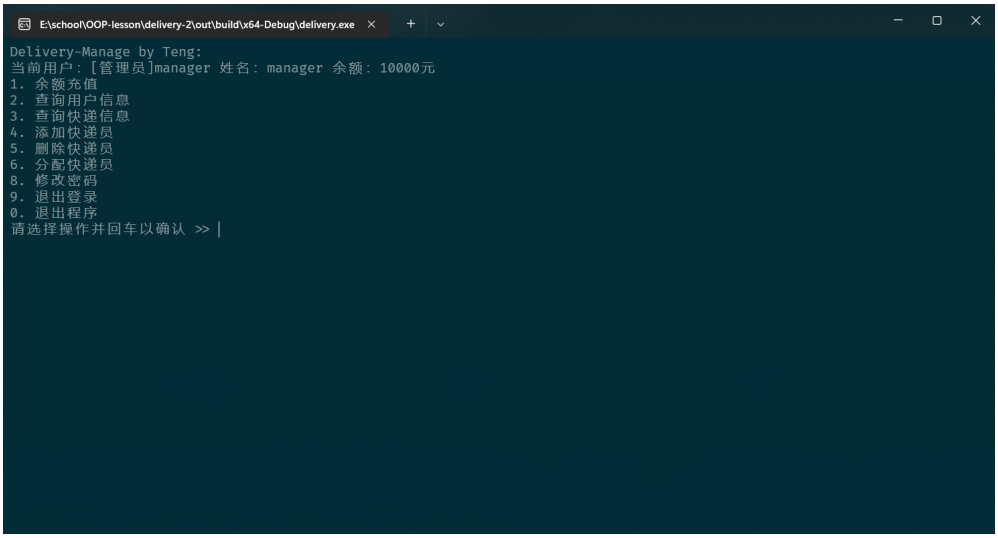
注册失败：用户名重复



登录失败：密码错误

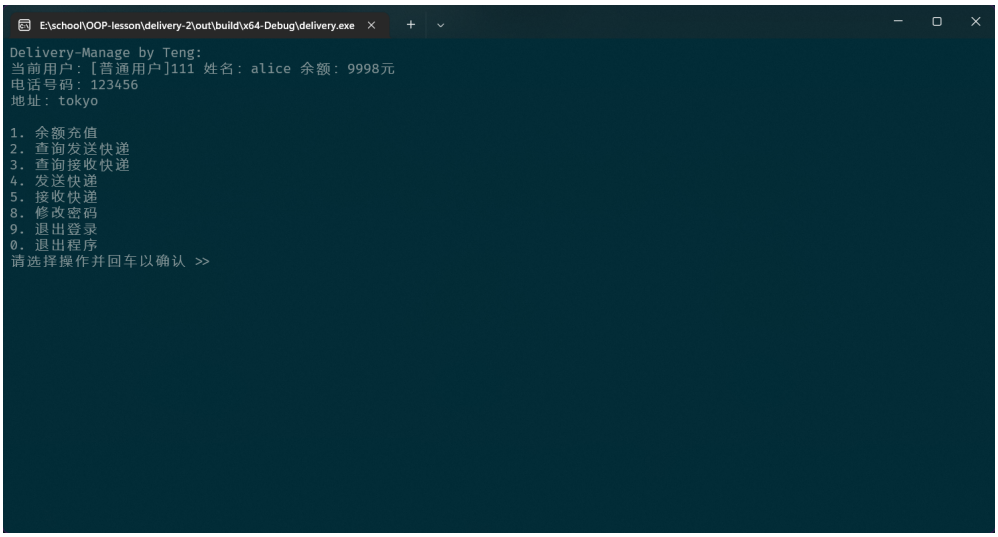
上图给出了一些异常提示：当用户注册重复用户名或用户登录时密码错误时，对应操作都将失败并且给出对应提示。当注册或登录成功后，会跳转到下文所示的相关界面。

2.1.3 管理员界面



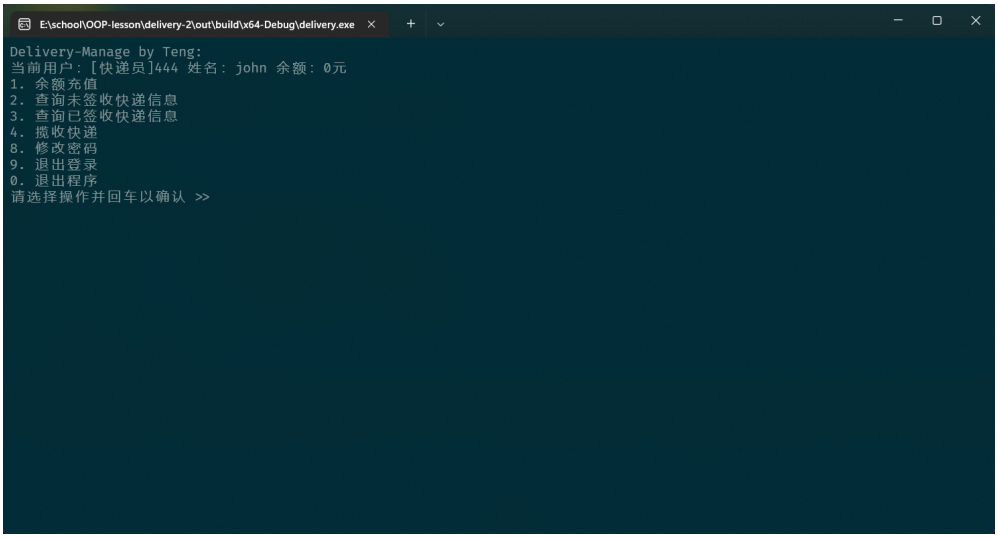
图中显示了以管理员身份登录时所进入的界面，对应功能均为题目所要求的功能且均已正确实现。选择不同的操作即可进入不同的功能。

2.1.4 用户界面



图中显示了以用户身份登录时所进入的界面，对应功能均为题目所要求的功能且均已正确实现。选择不同的操作即可进入不同的功能。

2.1.5 快递员界面

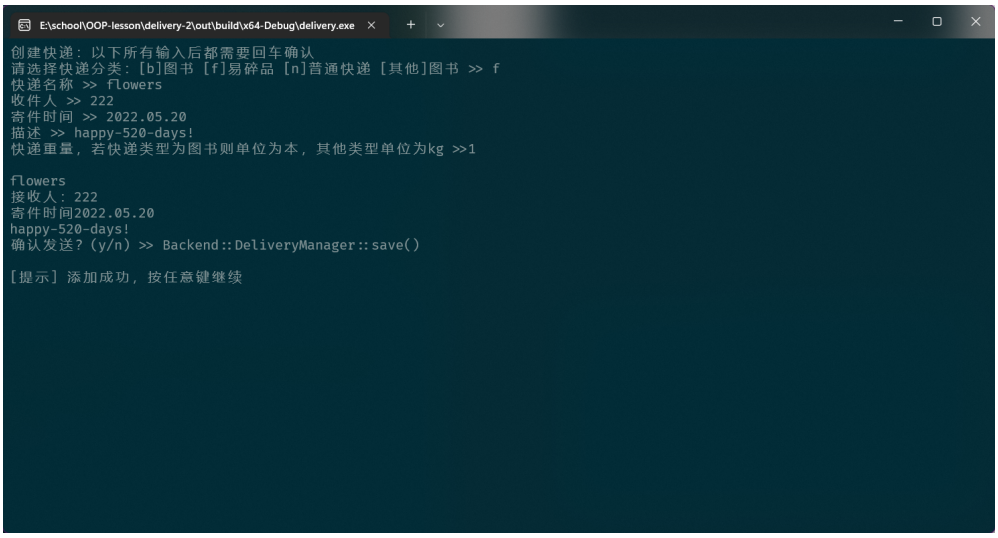


图中显示了以快递员身份登录时所进入的界面，对应功能均为题目所要求的功能且均已正确实现。选择不同的操作即可进入不同的功能。

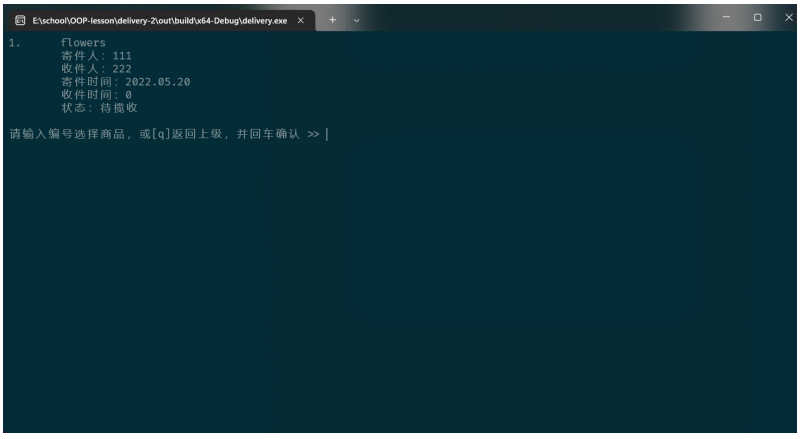
2.2 物流系统功能

现在以一个用户发出快递，另一个用户签收快递为情景进行相关物流系统功能的演示：

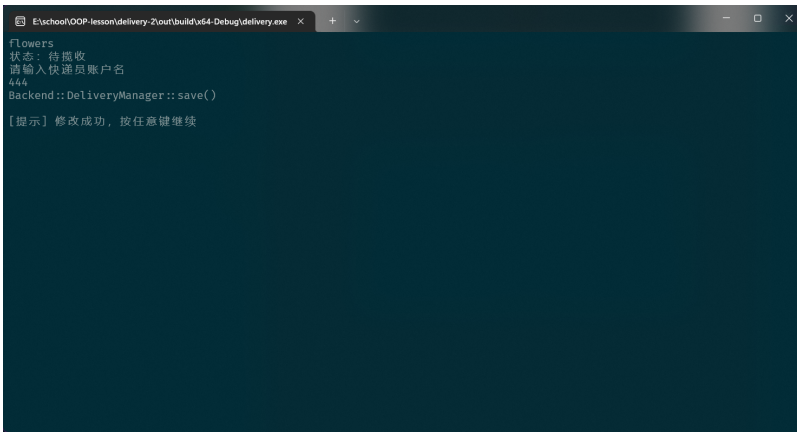
- 首先用户登录并选择发送快递操作，进入如下界面；填写好对应信息并确认发送快递后快递发送成功



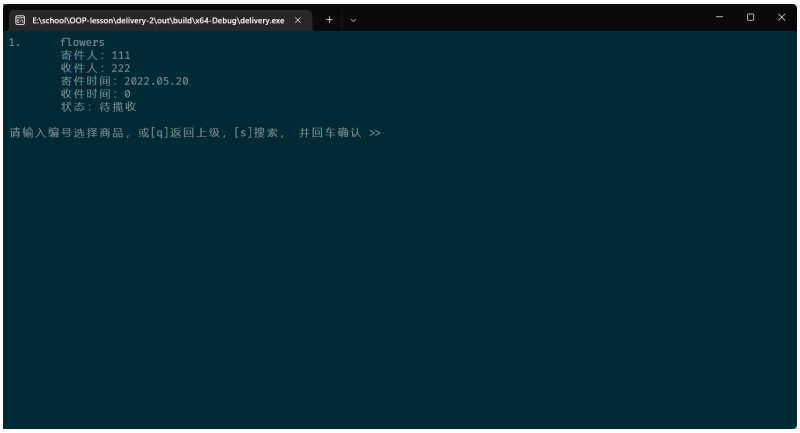
- 当用户发送快递后管理员选择分配快递员操作则系统给管理员展示未揽收的快递信息，并分配给快递员进行揽收操作。



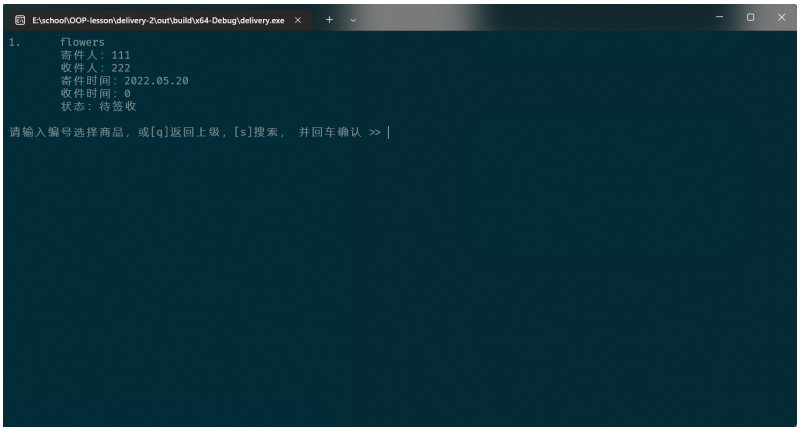
- 若输入快递员名称不存在则输出对应错误信息。



- 若对应快递员登录，快递员将查看自己未揽收的快递并进行揽收



- 在这之后若接收方用户登录，接收方会进行对应快递的签收



2.3 服务器功能

系统的服务端也具有非常丰富的功能，现列举如下：

- 完善的用户认证系统：系统的服务端设计了完善的用户认证系统。首先系统在文件中保存的密码时基于`SHA256`算法生成的哈希值，保证了用户密码的安全；同时服务器也会给用户密码加验，防止弱密码带来的可预测哈希值。由于使用TCP进行通信，服务端借助JWT生成用户令牌实现无状态的通信。
- 基于TCP和Json进行通信：系统的客户端和服务端通过TCP进行通信，使传播消息更加稳定，不容易出现丢包等情况；同时系统使用Json来储存信息，提升了开发效率，降低调试成本。
- 自实现了一个简单的Http解析库用于编码报文，约定报文格式。Http报文作为经典的通信报文协议，用其进行客户端和服务端的通信使得消息的传递更加稳定和便捷。

3 设计与实现

3.1 系统核心库

3.1.1 用户类和快递类的设计

首先对用户类和快递类的设计进行阐述。

由于这两个类都有一个核心特点：都需要区分用户或快递的类型，所以我先设计了一个总的用户类和快递类，再基于这个总的类实现具体类型的用户和快递类。

用户类的代码如下：

```
typedef enum USERTYPE
{
  MANAGER,
  COURIER,
  USER
} USERTYPE;

class User
{
public:
  User();
  string username;//用户名
  string passwordHash;//密码
  string name;//姓名
  string phoneNum;//电话
  string address;//地址
  int balance;//账户余额
  virtual USERTYPE getUserType() = 0;
  void setPassword(string password);
};
```

这里定义了用户的一些基本属性（参考注释），以及一个取得用户类型的虚函数作为接口；继承于这个用户总类的管理员类，快递员类和普通用户类就由此派生。下面列举快递员类的代码，该类仅定义了一个构造函数和获得用户类型的函数：

```
class Cuser : public User
{
public:
  Cuser();
  USERTYPE getUserType();
};
```

同样的道理，快递类的代码如下：

```
typedef enum DELIVERYTYPE
{
  NORMAL,//普通快递
```

```
    BOOK,//图书
    FRAGILE //易碎品
} DELIVERYTYPE;

class Delivery
{
public:
    Delivery();
    string id;//快递编号
    string name; //快递名称
    string sendUser;//寄件用户
    string receiveUser;//收件用户
    string sendTime;//寄件时间
    string receiveTime;//收件时间
    string description;//描述
    string courier; //负责派送的快递员用户名
    int pricing; //快递价格
    int state;//快递状态，0为待揽收，1为已签收，2为已揽收
    virtual DELIVERYTYPE getDeliveryType() const = 0;
    virtual int getPrice() const;
};
```

3.1.2 用户操作和快递操作类的设计

为了方便本地调用和客户端调用，我首先实现了如下虚基类：

```
class UserManager
{
    public:
        virtual ~UserManager() {};
        virtual bool reg(User* user) = 0;
        virtual bool del(User* user) = 0;
        virtual bool login(string username, string password) = 0;
        virtual bool logout() = 0;
        virtual bool addFund(int amount) = 0;
        virtual bool changePassword(string Password) = 0;
        virtual User* getUserByName(string username) const = 0;
        virtual map<string, User*> listUser() const = 0; //列出用户列表返回map
        User* currentUser;
};
```

可见，该虚基类定义了对于用户操作的各种接口，而继承于该虚基类的本地类和客户端类实现了对应于本地版本和网络版本的各个接口函数，供后文所提到的命令行界面类中函数的调用。

下面是快递操作虚基类的代码：

```
class DeliveryManager
{
  public:
    virtual ~DeliveryManager()
    virtual map<string, Delivery*> listDelivery() const = 0; //列出快递列表返回map
    virtual Delivery* getDeliveryById(string id) const = 0; //根据id获取快递
    virtual bool addDelivery(Delivery* delivery) = 0; //发送快递
    virtual bool replaceDelivery(Delivery* delivery) = 0; //修改快递
    virtual int getPrice(const Delivery* delivery) const = 0; //获取快递价格
};
```

3.2 服务器设计

3.2.1 http解析库的设计

该模块仿照常见的HTTP报文解析库进行了一些适用于系统的改动，诸如增加了发送快递接收快递等操作的关键字，接收系统传入的`url`参数等改动，代码如下：

```
class xHttp {
public:
    int code;
    HTTP_METHOD method;
    std::string url;
    std::map<std::string, std::string> headers;
    std::string body;
    std::string toString();
    static xHttp fromString(std::string str);
    static char tolower(char in);

};
```

网络模块的实现无不依赖于这个简洁高效的类HTTP操作库。

3.2.2 服务器类的设计

系统设计了一个服务器类来方便管理服务器的状态，实现服务器的功能。服务器类的定义如下：

```
class deliveryServer : public QTcpServer {
  Q_OBJECT
public:
  deliveryServer(QObject* parent = 0);
  ~deliveryServer();
protected slots:
  void onNewConnection();
  void onReadyRead();

private:
  void processData(QTcpSocket* sock);
};
```

可见服务器类主要定义了这些功能：

- 监控是否有新的连接建立。根据Qt信号槽机制来与新建的TCP句柄相连接并且传递数据。
- 取得当前socket对象。根据客户端发来的信息确认客户端所属的socket对象调用相应的数据和操作。
- 数据传递函数：打印服务器现在传递的数据报文并将报文传递给相应的客户端。

由服务器类的设计可知，系统是支持多线程和一定程度的并发处理的。这也是TCP+HTTP实现网络编程模块的优越之处：相较于一般的网络编程模块能够实现更稳定的数据传输和更高效的并发处理。

3.3 用户界面设计

3.3.1 客户端类的设计

在客户端部分，系统实现了两个客户端类，分别负责不同的功能：

- 客户端连接类：封装服务器提供的API，与服务器进行通信。
- 客户端数据模型类：继承于上述快递操作和用户操作的虚基类，直接通过传入经过客户端连接类封装的HTTP报文来进行相应的快递操作和用户操作，实现系统用户界面和底层逻辑的桥接。

客户端连接类代码如下，其中`request`变量和`sendRequest`函数进行与服务端的通信，`response`变量进行客户端中具体快递操作和用户操作所需数据的封装。

```
class xHttpClient {
public:
  xHttpClient() {}
  string base;
  xHttp request;
  xHttp response;
  int sendRequest();
};
```

客户端数据模型类代码如下，通过处理封装好的数据格式具体化了客户端所需的操作。下面列举用户操作类的代码，快递操作类整体逻辑与用户操作类相似，不再赘述。

```
namespace Client {
  class UserManager : Common::UserManager
  {
  public:
    UserManager(xHttpClient* Client);
    ~UserManager();
    bool reg(User* user);
    bool del(User* user);
    bool changePassword(string Password);
    map<string, User*> listUser() const;
    bool login(string username, string password);
    bool logout();
    bool addFund(int amount);
    User* getUserByName(string username) const;
    void load();
    void loadCurrentUser();
  private:
```

```
    xHttpClient* client;
    map <string, User*> userList;
    User* _getUserByName(string username) const;
  };
}
```

3.3.2 用户界面实现

用户界面中命令行参数的传递和帮助页面的实现是通过Qt提供的QCommandLineParser库来实现的。这样设计既提升了开发效率，又给了用户较好的使用体验。

用户界面中用户的操作和选择是基于如下核心类实现的：

```
class CLI
{
public:
  CLI();
  string passwordInput();

private:
  bool mainMenu();
  /*Users*/
  void reg();
  void login();
  void passwd();
  void recharge();
  void addCourier();
  void deleteCourier();
  void deleteUser(User* user);
  /*utils*/
  void alert(string s);
  void editorInput(string* s);
  string to_string(const double val, const int n = 2);
  /*list*/
  void listUser();
  void listExpress();
  void listSendExpress(User* user);
  void listReceiveExpress(User* user);
  void listSendCourierExpress(User* user);
  void listHaveSCourierExpress(User* user);
  /*delivery*/
  void sendExpress(User* user, User* manager);
  void receiveExpress(User* user);
  void takeExpress(User* user);
  void getExpress(User* user);
  void show(User* user, Delivery* d);
  void changeState(User* user, Delivery* d);
  /*print*/
  void print(Delivery* d);
```

```
  void printState(Delivery* d);
};
```

基于这个类，我们定义了用户可以进行的各种操作：注册，登录，添加快递等；`mainMenu`函数呈现了用户的操作界面，在这个函数上通过用户的不同选择执行不同的函数，使得程序可以按要求正确执行。

该类中具体函数和上文所述用户类，快递类的耦合基于如下类：

```
typedef enum DATATYPE
{
  UNDEFINED,
  STAND,
  ONLINE
} DATATYPE;

class Data
{
private:
  Data() {
    this->user = NULL;
    this->delivery = NULL;
  };
  Data(const Data&) {};
  Data& operator=(const Data&) {};
  ~Data()
  {
    if (this->user) delete this->user;
    if (this->delivery) delete this->delivery;
  }
  static void destroy(Data* p)
  {
    delete p;
  }
  static shared_ptr<Data> instance;
  xHttpClient client;

public:
  DATATYPE type = UNDEFINED;
  UserManager* user;
  DeliveryManager* delivery;
  void* conn;
  static shared_ptr<Data> getInstance()
  {
    return instance;
  }
  /*初始化为独立模式或服务端*/
  bool init(string userFile, string deliveryFile)
  {
```

```
    if (type != UNDEFINED)
    {
      return false;
    }
    this->user = (UserManager*)new Backend::UserManager(userFile);
    this->delivery = (DeliveryManager*)new Backend::DeliveryManager(deliveryFile);
    return true;
  }
  /*初始化为联网模式客户端*/
  bool init(string serverAddr) {
    if (type != UNDEFINED)
    {
      return false;
    }
    this->client.base = serverAddr;
    this->user = (UserManager*)new Client::UserManager(&this->client);
    this->delivery = (DeliveryManager*)new Client::DeliveryManager(&this->client);
    return true;
  }
};
```

该类管理着用户数据，根据命令行界面中用户选择操作中执行的函数调用对应的执行模式和用户类或快递类的操作，实现了程序主界面和程序后台函数的粘合。

4 开发与测试

4.1 开发与构建环境

系统在开发过程中使用的开发构建环境如下：

- windows 11
- Powershell 7
- CMake 3.20
- Visual Studio 2022
- vcpkg

在开发中使用的主要集成开发环境为Visual Studio，通过配置vcpkg和CMake引入Qt相关组件库，配置方式见附录。

在构建方面，我使用了CMake和Visual Studio相关工具链对项目进行编译，项目具体编译方法参考附录。

4.2 测试

首先我对所有实现的功能进行了测试：通过模拟用户在发送快递的过程中不同角色所处理功能的正确运作来验证了所实现功能的正确性。在验收过程中这一步已经做了演示；其次我还对一些异常情况进行了测试，列举如下：

- 输入异常：若输入与输入提示不符合的字符会强制重新开始该功能。在我处理到的地方该异常均能被正确处理。
- 用户职责异常：用户只能处理自己可以处理的数据，若由于某种原因处理到了其他数据，系统将进行报错并强制退出。测试显示该部分能被正确处理。
- http报文错误：通过使用python发送错误的http报文测试这类错误，包括报文不完全，格式不正确，内容错误，凭据错误等，这些错误都能被正确处理。

- 客户端和服务端通信错误：当客户端发出请求报文后，会被阻塞并等待来自服务端的回复；当服务端下线时，客户端也会发送服务端下线信息并被强行停止。

5 总结与展望

本次课程设计中，我设计实现了一个功能丰富的物流管理系统，具有完善的物流管理功能，并且基于物流管理系统实现了功能全面的客户端服务端系统，且具有设计良好的类的实现。

在课程设计的过程中我也遭遇了很多问题，例如在开发环境方面，由于使用 CMake 编译构建 Qt 的方法在互联网上的资源非常稀少，导致一开始在配置工程的构建环境时，就花费了许多时间进行摸索；在用户输入方面，系统中涉及到了对中文字符输入失效的问题，我通过查询资料自实现了中文输入的函数，从而很好的解决这个问题并且将这个基本功能函数用于系统各个功能的实现中。

本次课程设计中我的收获如下：

- 巩固了课堂上学习到的 C++ 面向对象知识，增强了 C++ 编程能力。
- 学习 Qt 框架的相关知识，增加对其的了解，初步掌握运用 Qt 开发的能力。
- 对面向对象编程有了更深的理解，增强面向对象架构和设计能力。
- 学习Qt和Visual Studio结合的开发环境配置，学习CMake的使用方法
- 在阅读 Qt 文档的过程中，增强信息检索能力和英文文献、文档阅读能力。

此外，若有时间，我还会对我的程序进行如下改进：

- 引入Google test等单元测试框架进行完整的单元测试。
- 完善`http`解析代码部分，加入完善的错误处理。
- 学习C++设计模式相关知识，降低程序各个模块的耦合度。
- 对网络模块进行更大量级的并发性测试并检测其性能，学习C++并发编程，尝试对网络模块实现高并发编程。

此次课程设计使我收获良多，积累了很多开发经验，对于其中的不足之处，我将在以后的学习中注重学习相关知识，争取从各方面提升自己的软件开发能力。

6 附录：系统构建的方式

6.1 构建依赖

系统的构建依赖于如下工具：

- C++ MSVC Compiler或自行修改`CMakeList.txt`改为其他编译器
- Visual Studio 2022
- CMake
- vcpkg
- Qt

6.2 构建步骤

首先通过`vcpkg`安装Qt相关的依赖（参考`CMakeList.txt`安装依赖）

随后在Visual Studio 2022里对`CMakeList.txt`进行生成操作，以产生out文件夹

最后在IDE中选中对应的可执行文件进行编译运行。