

Java 作业三：移动服务大厅 2

——连接数据库和 DAO 模式封装

一、移动服务大厅的框架

1.1 服务大厅的整体框架

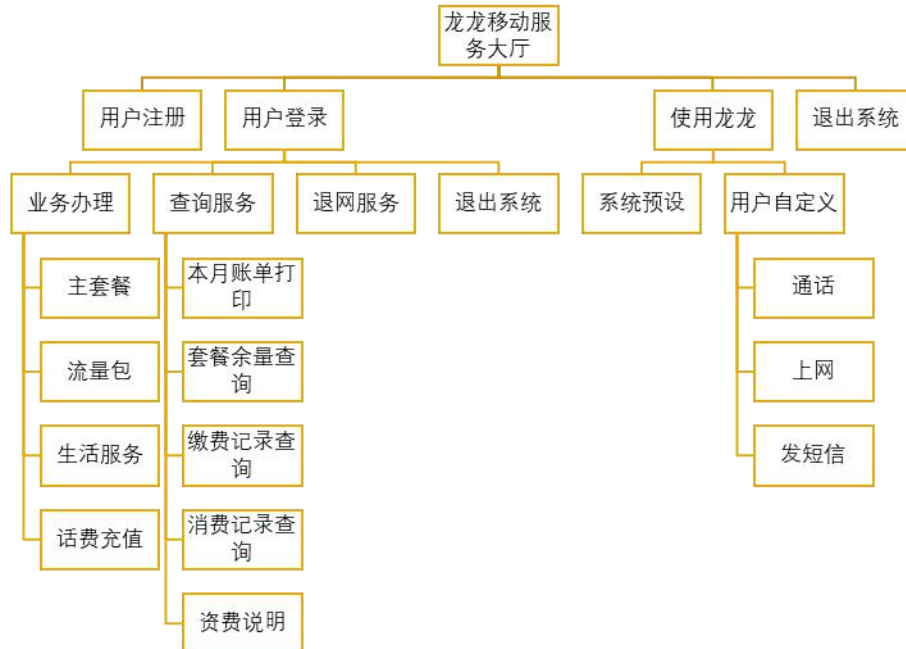


图 1-1 移动服务大厅的框架

1.2 数据库建构的框架

在移动服务大厅中，需要进行存储的数据主要有用户信息、用户的消费记录、用户的充值记录、用户的套餐使用情况、套餐的基本信息。这也是数据库中需要建立的五个表格，名字分别为 `userInfo`、`consumptionRecord`、`depositRecord`、`packageInfo`、`packageUse`。其中 `userInfo` 中存储的是用户的电话号码、用户名、密码、余额、套餐包、生活服务，以用户电话号码为主键；`consumptionRecord` 中存储的是用户的电话号码、消费类型、消费量、消费时间，这里以字段标号为主键，自动递增，作为计数；`depositRecord` 中存储的是用户的电话号码、充值的钱数和充值时间，同样以标号为主键，自动递增；`packageInfo` 存储的是每一个套餐包的名字、包含的通话时长、流量、短信数和月套餐费用，以套餐名为主键；`packageUse` 存储的是电话号码、套餐名字、通话时长、超出套餐的通话时长、流量、超出套餐的流量、短信数、超出套餐的短信数、花费、超出套餐的花费。

往这五个表格插入数据的存储对象主要是存储个人信息的 `SimCard` 类，存储用户套餐使用情况的 `MonthlyPackage` 类，以及存储所有用户的消费和充值信息的 `Customer` 类，而 `SimCard` 类和 `Customer` 类可以统一于 `ServiceHall` 类这个应用场景下，因此我针对用户套餐的增删改查封装了 `PackageInfoDAO` 接口，针对在营业服务大厅里面需要实现的打印用户各种信息，对用户信息进行增加删除修改等操作封装成了 `ServiceHallDAO` 接口。针对 MySQL 的语言特点，具体实现了 `MySQL_PackageInfo` 和 `MySQL_ServiceHall` 这两个类。

在 `MySQL_PackageInfo` 这个类里面，具体实现了两大类的函数。一类是针对套餐资费的，主要功能是将资费说明存储到数据库、将存储在数据库的资费说明更新到程序的各套餐类中、允许运营商在程序运行时进行资费更改。这类的函数的作用对象是 `MonthlyPackage`

及其子类，数据库中链接的表格是 packageInfo。另一类是针对用户的，包括用户套餐的初始化、更新用户使用电话后的套餐余量变化、记录用户的套餐变更、更新余额、打印套餐请使用情况 and 套餐余量。这类函数的传参数据实类是 SimCard，作用的数据库里的表格涉及 packageUse, userInfo。

```
public interface PackageInfoDAO {
    void CreatePackageInformation(File f) throws Exception; //将资费说明加入到数据库中
    void UpdatePackageInformation(File f) throws Exception; //更改资费说明(运营高价格调整)
    void ModifyPackageInformation(MonthlyPackage p) throws Exception; //更改后台程序中的内容, 更改套餐的基本信息
    void UpdatePackageInformation(SimCard s) throws Exception; //更新用户的套餐使用情况, 往数据库里更新
    LinkedList<String> PrintOutPackageInformation(SimCard s) throws Exception; //将用户的套餐情况打印
    void UpdateUserPackage(SimCard s) throws Exception; //将当前用户的套餐数据传到用户的package中
    void UpdateLifeService(SimCard s, String life, double cost) throws Exception; //更新用户选择的生活服务
    void UpdateBalance(SimCard s, double b) throws Exception; //更新余额
    void ChangePackage(SimCard s, String packName) throws Exception;
    // 打印套餐余量
}
```

图 1-2 PackageInfoDAO 的函数

在 MySQL_ServiceHall 这个类里面，具体实现了创建新用户，删除用户，增加充值和消费记录，查找用户信息，根据不同权限返回用户信息，打印充值和消费记录等函数，主要是以 SimCard 类作为数据传输载体。但是这个类涉及到所有数据库表格，但主要在 Customer 类里面被调用实现功能。简而言之，就是这些封装的函数是通过 Customer 类的对象实例接收到存储了 SimCard 用户信息的对象实例，有一个间接的关系，之所以这么处理，是因为考虑到真正应用部署时空间和时间的一个权衡，这会在后面进行进一步详细地解释。

```
public interface ServiceHallDAO {
    void init() throws Exception;
    void CreateNewCustomer(SimCard s) throws Exception;
    void UpdateDeposit(SimCard s, String deposit) throws Exception;
    void UpdateConsumption(SimCard s, String type) throws SQLException;
    void Delete(String s) throws SQLException;
    boolean FindCustomer(String phone) throws Exception;
    SimCard Customer_info(String Phone, String password) throws Exception;
    SimCard Customer_use(String phone) throws Exception;
    LinkedList<String> PrintOutMonthlyConsumption(SimCard s) throws SQLException;
    LinkedList<String> PrintOutMonthlyDeposit(SimCard s) throws SQLException;
}
```

图 1-3 ServiceHallDAO 的函数

二、数据库操作的封装与实现

2.1 MySQL 数据库通用操作的封装

由于数据库的连接需要非常多重复的代码，因此除了对服务大厅的实体类数据操作进行封装之外，还专门写了一个 DatabaseUtil 类，负责对数据库进行连接，创建 statement，关闭资源，以及进行防 SQL 语句注入漏洞的数据库语句执行。每一次要进行数据库连接的时候，都会调用这个类的实例化对象，实现从数据库中读数据以及从数据库中写数据。

其中，DatabaseUtil 类实现数据库连接的函数，主要是通过配置 properties 文件，从文件里面读取相应的配置信息，从而建立连接，由 init() 函数进行实现。

```

public class DatabaseUtil {
    private static String driver;
    private static String url;
    private static String user;
    private static String password;
    private static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DB_URL = "jdbc:mysql://localhost:3306/javahw?serverTimezone=Asia/Shanghai";
    private PreparedStatement pstmt = null;
    private Connection con = null;
    private Statement stmt = null;
    ResultSet rs = null;
    DatabaseUtil(){
        try {
            init();//这种作用到底能不能完成我在调用getConnection之前初始化
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void init() throws Exception {...}
    int executeUpdate(String prepareSQL, Object[] objects) throws Exception {...}

    ResultSet executeQuery(String prepareSQL, Object[] objects) throws Exception {...}

    Statement getConnection()throws Exception {...}
    void closeAll() throws SQLException {...}
}

```

图 2-1 DatabaseUtil 的实现

另外，为了预防 MySQL 注入漏洞，专门实现了 executeUpdate 和 executeQuery 两个函数，通过 prestatement 的这种机制，防止出现人为的侵入数据库系统，随意修改内容的情况。具体代码实现如图 2-2、2-3。

```

int executeUpdate(String prepareSQL, Object[] objects) throws Exception {
    getConnection();
    int count = 0;
    pstmt = con.prepareStatement(prepareSQL);
    try {
        if (objects != null) {
            for (int i = 0; i < objects.length; i++) {
                pstmt.setObject(i + 1, objects[i]);
            }
            count = pstmt.executeUpdate();
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Update Failed!");
        return 0;
    }
    return count;
}

```

图 2-2 executeUpdate 的实现

```

ResultSet executeQuery(String prepareSQL, Object[] objects) throws Exception {
    getConnection();
    ResultSet rs = null;
    pstmt = con.prepareStatement(prepareSQL);
    try {
        if (objects != null) {
            for (int i = 0; i < objects.length; i++) {
                pstmt.setObject(i + 1, objects[i]);
            }
            rs = pstmt.executeQuery();
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Update Failed!");
        return null;
    }
    return rs;
}

```

图 2-3 executeQuery 的实现

2.2 MySQL_ServiceHall 的具体实现

该类的函数实现主要是对 ServiceHallDAO 的函数的一个实例化，运用 MySQL 的语言特点进行代码编写。以 UpdateDeposit 为例，如图 2-4。

```

@Override
public void UpdateDeposit(SimCard s, String deposit) throws Exception {
    int first = deposit.indexOf("值");
    int last = deposit.indexOf("元");
    double d = Double.parseDouble(deposit.substring(first+1,last));
    try {
        stmt = util.getConnection();
        String depositRecord = new String("original: insert into DepositRecord(Telephone, Money) values('"+ s.telephone + "'," + d + ")");
        stmt.executeUpdate(depositRecord);
    } catch (Exception e) {
        e.printStackTrace();
    }
    stmt.close();
    util.closeAll();
}

```

图 2-4 针对 MySQL 的接口函数实例化


值得进行细致解释的有以下几点：

①由于在每一次函数运行的时候，都要将原本程序里进行了初始化的用户信息导入数据库中，因此会存在重复的主键内容添加的情况。为避免这样的错误，我在编写插入语句的时候还添加了以下的语句“on duplicate key update ……”，如图 2-5。

```

insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('18902810777','little_wheat','152130','话费套餐',12.5,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('18902810777','话费套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('18318888999','Dududu','152126','网虫套餐',140.7,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('18318888999','网虫套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('18902285145','Lydia','152127','网虫套餐',62.5,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('18902285145','网虫套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('15815670706','Xuan','152137','超人套餐',0.0,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('15815670706','超人套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('18902810777','little_wheat','152130','话费套餐',12.5,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('18902810777','话费套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('18318888999','Dududu','152126','网虫套餐',140.7,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('18318888999','网虫套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('18902285145','Lydia','152127','网虫套餐',62.5,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('18902285145','网虫套餐',0.0,0.0,0.0,0.0,
insert into userInfo(Telephone, userName, password, packageName, balance, lifeService) values('15815670706','Xuan','152137','超人套餐',0.0,'[]')on duplicate key update
insert into packageUse(Telephone, packageName, phoneCall, OutPhoneCall, data, OutData, message, OutMessage, fee, OutFee) values('15815670706','超人套餐',0.0,0.0,0.0,0.0,

```

```

, '[]') on duplicate key update userName = 'Little_Wheat', password = '152130', packageName = '话务套餐', balance = 12.5, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 58.0, 0.0)
) on duplicate key update userName = 'Dududu', password = '152126', packageName = '网虫套餐', balance = 140.7, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 68.0, 0.0)
n duplicate key update userName = 'Lydia', password = '152127', packageName = '网虫套餐', balance = 62.5, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 68.0, 0.0)
duplicate key update userName = 'Xuan', password = '152137', packageName = '超人套餐', balance = 0.0, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 78.0, 0.0)
, '[]') on duplicate key update userName = 'Little_Wheat', password = '152130', packageName = '话务套餐', balance = 12.5, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 58.0, 0.0)
) on duplicate key update userName = 'Dududu', password = '152126', packageName = '网虫套餐', balance = 140.7, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 68.0, 0.0)
n duplicate key update userName = 'Lydia', password = '152127', packageName = '网虫套餐', balance = 62.5, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 68.0, 0.0)
duplicate key update userName = 'Xuan', password = '152137', packageName = '超人套餐', balance = 0.0, lifeService = '[]';
套餐', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 78.0, 0.0)

```

图 2-5 避免重复插入数据，在 insert 中添加了“on duplicate key update……”

②在更新消费记录和充值记录中，虽然在 MySQL_ServiceHall 类的函数里非常简单，但实际上在这里采取了内外存先缓存再交互的思想。

考虑到程序真正上线后，会有很多用户同时进行使用，服务器需要对每个用户的使用模式以及使用情况进行记录和计算，再写入数据库中，本身是比较耗时间的，但必须及时的更新数据库的内容，否则会发生错误。但对于充值信息和消费信息这种信息类型的，不需要更新数据的操作，就可以暂缓存入数据库中。因此，对于用户的每一条充值和消费信息，我首先是将他们存放在 Customer 类的对象中。这个类的工作本来就是将个体的 SimCard 类对象信息集合起来，再与数据库进行其他的交互，因此作为暂时的存储地方再合适不过。

而为了在此基础上减少程序内部的增加与删除，因此我将暂时存储充值和消费信息的结构设置为了循环队列，既可以满足先进先出的需求，又不需要初始化一块非常大的空间。而这个循环队列的大小需要看程序真正部署后的常用空间大小量。如图 2-6。

```

private CycQueue<Object[]> consumption = new CycQueue<>();
private CycQueue<Object[]> deposit = new CycQueue<>();

```

图 2-6 循环队列的数据结构

使用循环队列后，在 Customer 类对象中增加消费和充值信息就只需要使数据入队即可。如图 2-7。

```

public void add_deposit(SimCard cus, String de) throws MyException {
    Object[] depositRecord = {cus, de};
    boolean res = deposit.Enqueue(depositRecord);
    if(!res){
        System.out.println("CycQueue_break");
        throw new MyException("CycQueue_break");
    }
}

public void add_consumption(SimCard s, String con) throws MyException {
    Object[] consumeRecord = {s, con};
    boolean res = consumption.Enqueue(consumeRecord);
    if(!res){
        System.out.println("CycQueue_break");
        throw new MyException("CycQueue_break");
    }
}

```

图 2-7 信息的入队

③短暂的存储在内存里的数据要想写入到数据库中，需要函数调用。我在这里采取的是通过定时器触发，进行每隔一段时间的自动写入，如果队列为空，则没有任何操作，如果队列不空，则将数据写入数据库中。这里的实现主要是通过 java 内置的定时器 TimerTask 父类，Timer 类对象的函数，以及重写 run()函数，达到目的。如图 2-8。

```
public void run(){
    Date date = new Date(System.currentTimeMillis());
    //System.out.println(date.toString());
    while(deposit.QueueLength()>0){
        try {
            Object[] res = deposit.DeQueue();
            SimCard s = (SimCard) res[0];
            String de = (String) res[1];
            serviceHall.UpdateDeposit(s,de);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    while(consumption.QueueLength()>0){
        try {
            Object[] res = consumption.DeQueue();
            SimCard s = (SimCard) res[0];
            String con = (String) res[1];
            serviceHall.UpdateConsumption(s,con);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if(stop){
        this.cancel();
    }
}
```

图 2-8 重写 run()函数

```
static private void RootMenu() throws Exception {
    Timer timer = new Timer();
    timer.schedule(customers, delay: 0, period: 10*1000);
}
```

图 2-9 在移动服务大厅的主函数中进行触发和隔 10 秒执行

④在打印消费记录和充值记录中，由于只是打印当月的记录，因此在输出时进行了时间的比较，如果是当月的记录就被返回，否则就不被返回和打印，如图 2-10。

```

public LinkedList<String> PrintOutMonthlyDeposit(SimCard s) throws SQLException {
    LinkedList<String> results = new LinkedList<>();
    ResultSet rs = null;
    try{
        stmt = util.getConnection();
        StringBuffer printCon = new StringBuffer("SELECT * FROM DepositRecord where Telephone = ?");
        String[] telephone = {s.telephone};
        rs = util.executeQuery(printCon.toString(), telephone);
        String temp, date, temp_month;
        while(rs.next()){
            temp = rs.getString( $: "Telephone");
            temp += " ";
            temp += rs.getString( $: "Money");
            temp+="元 ";
            date = rs.getString( $: "DepositTime");
            int month = now.get(Calendar.MONTH)+1;
            int hyphen1 = date.indexOf("-"), hyphen2 = date.lastIndexOf( str: "-");
            temp_month = date.substring(hyphen1+1,hyphen2);
            if(temp_month.equals(String.valueOf(month))){
                temp += date;
                results.add(temp);
            }
        }
    }
}

```

图 2-10 时间的对比以输出当月的充值和消费信息

2.3 MySQL_PackageInfo 的具体实现

该类的函数的具体实现，更多的是针对不同数据的处理方式进行的 MySQL 语言的编写。以 UpdateUserPackage()函数为例，如图 2-11。

```

@Override
public void UpdateUserPackage(SimCard s) throws Exception {
    try{
        stmt = util.getConnection();
        StringBuffer curr_pack = new StringBuffer("SELECT * FROM packageUse WHERE Telephone = '"+s.getTelephone()+"'");
        rs = stmt.executeQuery(curr_pack.toString());
        while(rs.next()){
            s.monthlyPackage.fee = rs.getDouble( $: "fee");
            s.monthlyPackage.extra_fee = rs.getDouble( $: "OutFee");
            s.monthlyPackage.phone_call = rs.getDouble( $: "phoneCall");
            s.monthlyPackage.extra_call = rs.getDouble( $: "OutPhoneCall");
            s.monthlyPackage.data = rs.getDouble( $: "data");
            s.monthlyPackage.extra_data = rs.getDouble( $: "OutData");
            s.monthlyPackage.message = rs.getInt( $: "message");
            s.monthlyPackage.extra_message = rs.getInt( $: "OutMessage");
        }
    }catch(Exception e){
        e.getMessage();
    }
    stmt.close();
    util.closeAll();
}

```

图 2-11 MySQL_PackageInfo 的函数实现

三、数据库操作的函数调用

以上针对 MySQL 的两个类的具体函数实现都是需要调用。而函数的调用位置主要集中在 SimCard 类的余额更新，Customer 类的创建新用户、删除用户、存储和打印消费和充值记录以及获取用户信息，ServiceHall 中使用龙龙那一部分的用户套餐使用情况更新这几个地方，主要是替换了本来以程序中的数据结构进行存储信息的代码。

四、代码运行的效果

代码的界面仍然是以命令行模式实现，因此具体的功能展示与第一次作业无异，这里重点展示一下数据库表格的数据记录和数据设置。下面数据库表格的截图都是通过代码运行生成的。

Telephone	userName	password	packageName	lifeService	Balance
15815670706	Xuan	152137	超人套餐	[]	93.4
18318888999	Dududu	152126	网虫套餐	[]	140.7
18902285145	Lydia	152127	网虫套餐	[]	62.5
18902810777	Little_Wheat	152130	话痨套餐	[Entertainme	5.47
18903740847	user2	user2	网虫套餐	[]	173.75
18917225901	user1	user1	话痨套餐	[]	142.5

图 4-1 userInfo 的表格

Telephone	PackageName	phoneCall	OutPhoneCall	data	OutData	message	OutMessage	fee	OutFee
15815670706	超人套餐	0	0	0	0	100	50	83	5
18318888999	网虫套餐	0	0	1200	0	0	0	68	0
18902285145	网虫套餐	0	0	4743	1671	0	0	235.1	167.1
18902810777	话痨套餐	446	0	0	0	239	209	78.9	20.9
18903740847	网虫套餐	0	0	0	0	0	0	68	0
18917225901	话痨套餐	0	0	0	0	0	0	58	0

图 4-2 packageUse 的表格

packageName	phoneCall	data	message	fee
普通计费	0.3	0.3	2	0
网虫套餐	0	3072	0	68
话痨套餐	500	0	30	58
超人套餐	200	1024	50	78
超出套餐计费:	0.2	0.1	1	0

图 4-3 packageName 的表格

No	Telephone	Money	DepositTime
4	18318888999	200	2019-11-15 23:36:38
5	18917225901	200.5	2019-11-16 20:13:04
6	18917225901	200.5	2019-11-16 20:13:09
7	18903740847	241.75	2019-11-16 20:13:27
8	18903740847	241.75	2019-11-16 20:13:29
9	18902810777	65.67	2019-11-16 20:14:39

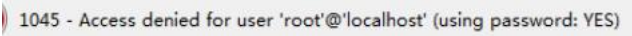
图 4-4 depositRecord 的表格，由于之前有在退网时会删除该卡号的所有信息，所以这里的主键 No 不是从 1 开始

No	Telephone	Type	Flow	ConsumeTime
10	18318888999	流量	5	2019-11-15 23:36:18
11	18902810777	短信	5	2019-11-16 20:13:59
12	18902810777	短信	234	2019-11-16 20:14:09
13	18902810777	通话	90	2019-11-16 20:14:49
14	18902810777	通话	356	2019-11-16 20:14:59

图 4-5consumeRecord 的表格

五、问题与改进

5.1 数据库的连接问题

数据的连接当时出现了 ，解决这个问题花费了比较多波折。因此以后在进行软件的安装、环境的配置、jar 包的导入都需要及时进行建立好连接，并确保自己的软件在试用期内或尝试找到破解版。

5.2 Timer 的设置和停止运行

由于设置了定时器，因此需要在程序结束前进行定时器的关闭，否则程序将无法退出。并且，关闭定时器之前，必须要把我通过设置定时器向数据库传输的各种内容全部都传输完成才正式的关闭 timer，否则会出现数据的丢失。

5.3 代码冗余度较高

代码在使用龙龙这个子菜单中重复的部分尤其多，因此针对每一次用户使用完某个功能后的数据更新都需要调用一遍写信息到数据库里的函数。这大大降低了代码调整的效率。接下来需要对框架进行更好的优化，对代码进行更好的简化和封装。