

轻量级 J2EE 框架应用

E 6 A Simple Controller with DAO pattern & O/R mapping

学号：SA17225052

姓名：戴赛

报告撰写时间：2017/1/7

1.主题概述

理论上这次作业应该是在上次作业的基础上实现的.

Conversation 负责将类的信息翻译成 SQL 语句,再调用 DAO 类,实现 CRUD

但是上次作业中 UserDao 是在 UserSC 而不是在 Controller 中,这么实现的话会将 SQL 语句暴露出来,不符合逻辑.

所以我在写这次作业时将 DAO 的功能全部放在了 Conversation 中,将其作为一个高级的 DAO 来使用

在有关读取 xml 类文件的时候,为了以后的可扩展性,将 if 类语句都改为了 switch 语句

2.假设

上上次作业 E4

TomCat

MySQL

3.实现或证明

1. 2.

编写 or_mapping.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<OR-Mappings>
  <jdbc>
    <!--配置部分-->
    <property>
      <name>driver_class</name>
      <value>com.mysql.jdbc.Driver</value>
    </property>
    <property>
      <name>url_path</name>
      <value>jdbc:mysql://127.0.0.1:3306/USTC</value>
    </property>
    <property>
      <name>db_username</name>
      <value>USTC</value>
    </property>
    <property>
      <name>db_userpassword</name>
      <value>123</value>
    </property>
  </jdbc>
  <class>
    <!--映射文件-->
    <!--JavaBean名称,-->
    <name>UserBean</name>
    <!--对应数据库中的表名-->
    <table>user3</table>
    <id>
      <!--主键属性名称-->
      <name>userID</name>
    </id>
    <property>
      <name>userName</name>
      <column>user_name</column>
      <type>String</type>
      <!--是否使用延迟加载-->
      <lazy>false</lazy>
    </property>
    <property>
      <name>userPass</name>
      <column>user_pass</column>
      <type>String</type>
      <lazy>true</lazy>
    </property>
  </class>
</OR-Mappings>
```

因为对 Table 的创建方法进行了多次的测试,所以 table 的名字改为了会有各种变化

3.

编写 Configuration

在其构造方法中执行 xml 的读取操作,并对相应的属性进行初始化,因为 Conversation 与 Configuration 在同一包下,所以这里的很多方法都写成了包私有.

```
public class Configuration {  
  
    private String driver_class;  
    private String url_path;  
    private String db_username;  
    private String db_userpassword;  
  
    // 保存class信息  
    private ArrayList<OrClassConfiguration> classList = new ArrayList<>();  
  
    Configuration() { this.readMapping(); }
```

读取方法 readMapping()

```
private void readMapping()  
{  
    try {  
        InputStream inputStream = this.getClass().getResourceAsStream("or_mapping.xml");  
        SAXReader saxReader = new SAXReader();  
        Document document = saxReader.read(inputStream);  
  
        // 获取(0)根节点  
        Element rootElement = document.getRootElement();  
        System.out.println("根节点名称:" + rootElement.getName());  
  
        // 获得(1)层节点,可能有多种  
        List<Element> baseElementsList = rootElement.elements();  
        for(Element baseElement:baseElementsList)  
        {  
            switch (baseElement.getName())  
            {  
                case "jdbc":  
                {  
                    // 读取配置信息  
                    isJDBC(baseElement);  
                    break;  
                }  
                case "class":  
                {  
                    // 读取映射信息  
                    isClass(baseElement);  
                    break;  
                }  
            }  
        }  
    }  
    catch (DocumentException e) {  
        System.out.println("Configuration文件读取失败"+e);  
    }  
}
```

读取到 jdbc 节点,即数据库配置信息时

执行 isJDBC()方法对数据库属性进行赋值

```
private void isJDBC(Element jdbcElement)
{
    List<Element> propertiesList = jdbcElement.elements();
    for(Element property:propertiesList)
    {
        // System.out.println(property.getName());
        Element nameElement = property.element("name");
        String nameText = nameElement.getTextTrim();
        Element valueElement = property.element("value");
        String valueText = valueElement.getTextTrim();

        System.out.println(nameText);
        switch (nameText)
        {
            case "driver_class":
            {
                driver_class = valueText;
                break;
            }
            case "url_path":
            {
                url_path = valueText;
                break;
            }
            case "db_username":
            {
                db_username = valueText;
                break;
            }
            case "db_userpassword":
            {
                db_userpassword = valueText;
                break;
            }
        }
    }
    // 检查用
    // System.out.println(driver_class+" "+url_path+" "+db_username+" "+db_userpassword);
}
```


读取的是 class 节点,即 bean 的映射信息时

执行 isClass 方法对一个 List 进行赋值来保存信息,考虑到实际工作中应该会有多个 Bean 类,所以这里用 List 来保存信息.List 中的数据为一个专门的类 OrClassConfiguration

```
private void isClass(Element classElement)
{
    OrClassConfiguration orClassConfigurationTemp = new OrClassConfiguration();
    Element name = classElement.element("name");
    Element table = classElement.element("table");
    Element id_name = classElement.element("id").element("name");
    List<Element> classFeaturesList = classElement.elements();
    for(Element classFeatureElement:classFeaturesList)
    {
        switch (classFeatureElement.getName())
        {
            case "name":
            {
                // orClassConfigurationTemp.getClassName(classFeatureElement.getTextTrim());
                String textTrim = classFeatureElement.getTextTrim();
                orClassConfigurationTemp.setClassName(textTrim);
                break;
            }
            case "table":
            {
                String textTrim = classFeatureElement.getTextTrim();
                orClassConfigurationTemp.setTable(textTrim);
                break;
            }
            case "id":
            {
                Element id_name = classFeatureElement.element("name");
                String textTrim = id_name.getTextTrim();
                orClassConfigurationTemp.setClassId(textTrim);
                break;
            }
            case "property":
            {
                Element p_name = classFeatureElement.element("name");
                Element p_column = classFeatureElement.element("column");
                Element p_type = classFeatureElement.element("type");
                Element p_lazy = classFeatureElement.element("lazy");

                orClassConfigurationTemp.setProperty(p_name.getTextTrim(),p_column.getTextTrim(),p_type.getTextTrim(),p_lazy.getTextTrim());
            }
        }
    }
    classList.add(orClassConfigurationTemp);
}
```

OrClassConfiguration 包含了 property 节点下 Bean 类的四种属性,其中 property 是一个 HashMap 组成的 List

```
class OrClassConfiguration {

    private String name;
    private String table;
    private String id;

    private ArrayList<HashMap> propertiesList = new ArrayList<>();

    // 包私有
    void setProperty(String p_name,String p_column,String p_type,String p_lazy)
    {
        HashMap<String, String> tempProperty = new HashMap<>();
        tempProperty.put("name",p_name);
        tempProperty.put("column",p_column);
        tempProperty.put("type",p_type);
        tempProperty.put("lazy",p_lazy);
        // 直接加到List中
        propertiesList.add(tempProperty);
    }

    ArrayList<HashMap> getPropertiesList() { return propertiesList; }
    String getClassName() { return name; }

    void setClassName(String name) { this.name = name; }

    String getTable() { return table; }

    void setTable(String table) { this.table = table; }

    String getClassId() { return id; }

    void setClassId(String id) { this.id = id; }

}
```

4.

4.1 Conversation 类,内有开关数据库的基础方法

在当中实现了增和查操作,对应登陆和注册

之后又添加了创建表和检查表是否存在的方法用于辅助增操作

又添加了一个方法在懒加载时使用

Conversation 的属性与构造方法,这里构造方法的参数是传入 Bean 类的完整名称,如"water.ustc.bean.UserBean",在 Bean 中可使用"this.getClass.getName"来直接获取这个 String

```
// JDBC部分
private String driver_class;
private String url_path;
private String db_username;
private String db_userpassword;
private Configuration configuration;
// Class部分
private String class_Name;
private String class_table;
private String class_ID; // 主键名
private ArrayList<HashMap> propertiesList;
// 传进来Bean名,用来与xml做对比并且决定了查询的Object返回类型
private String classNameFromBean;
private String classNamePath; // 完整的名称,用来反射

public Conversation(String classPathFromBean)
{
    System.out.println("~~~~~以下是Conversation~~~~~");
    // water.ustc.bean.UserBean,完整类名,用于反射
    this.classNamePath = classPathFromBean;
    // 路径中截取类名
    String nameFromPath = classPathFromBean.substring(classPathFromBean.lastIndexOf(".") + 1);
    System.out.println("传入的类名为" + nameFromPath);
    this.classNameFromBean = nameFromPath;
    this.configuration = new Configuration();
    setJDBC();
    setClass();
}
```

setClass 会遍历 Configuration 中的存放 Class 信息的 classList,将从 bean 那里获取到的名称与 xml 中的配置信息进行匹配.propertiesList 存放的是 class 下的 property 信息

```
private void setJDBC()
{
    this.driver_class = configuration.getDriver_class();
    this.url_path = configuration.getUrl_path();
    this.db_username = configuration.getDb_username();
    this.db_userpassword = configuration.getDb_userpassword();
}

private void setClass()
{
    ArrayList<OrClassConfiguration> classList = configuration.getClassList();
    for(OrClassConfiguration classConfiguration:classList)
    {
        // 从xml中找出相应的class配置
        if(classConfiguration.getClassName().equals(classNameFromBean))
        {
            this.class_Name = classConfiguration.getClassName();
            this.class_table = classConfiguration.getTable();
            this.class_ID = classConfiguration.getClassId();
            this.propertiesList = classConfiguration.getPropertiesList();
        }
    }
    if(!this.class_Name.equals(classNameFromBean))
    {
        System.out.println("or_mapping中未找到名为" + classNameFromBean + "的配置文件");
    }
}
```


开关数据库方法

```
private Connection openDBConnection()
{
    Connection conn = null;
    System.out.println("创建数据库连接,加载驱动:");
    try {
        try {
            // System.out.println("000000"+url+";"+userName+";"+userPassword+"000000");
            Class.forName(driver_class);
            conn = DriverManager.getConnection(url_path,db_username,db_userpassword);
        } catch (SQLException e) {
            System.out.println("    创建链接失败:"+e);
            // e.printStackTrace();
        }
    } catch (ClassNotFoundException e) {
        System.out.println("    加载驱动失败:" + e);
    }
    return conn;
}

// 关闭数据库连接
private boolean closeDBConnection(Connection connection, PreparedStatement pstmt, ResultSet rs)
{
    // conn数据库链接对象
    // rs数据返回结果,有可能为空
    // pstmt数据库命令执行对象
    System.out.println("关闭数据库连接");
    try {
        if (rs!=null)
        {
            rs.close();
            // rs = null;
        }
        if (pstmt!=null)
        {
            pstmt.close();
            // pstmt = null;
        }
        if (connection!=null)
        {
            connection.close();
            // connection = null;
        }
        System.out.println("    数据库连接关闭成功");
    } catch (SQLException e)
    {
        System.out.println("    数据库关闭失败,错误为:"+e);
        return false;
    }
    return true;
}
```

4.2 insert()插入方法,其形参是 bean 对象

首先利用完整的类名反射找出 bean 对象的类名并以此和 or_mapping 中的信息(已经存放在 propertiesList 中)找到其对应的各个属性的 getter 方法
然后执行 getter 方法,将 bean 中的各个属性存入到一个 List 中.这个 list 会在后面拼接 SQL 语句中用的上

```
public boolean insert(Object beanObject)
{
    // 数据库是否插入成功
    boolean insertResult = false;

    try {
        // 反射找到Bean对象
        Class beanClass = Class.forName(classNamePath);

        ArrayList<String> valueList = new ArrayList<>();
        valueList.add("0"); // 第一个为ID,写死为0
        // 1.获取各个getter,对于这个需要得到getUserName,getUserPass
        // 1.1.利用getter得到value(即userName的值,userPass的值)
        // 1.2.构建ArrayList(valueList) [id的值*,userName的值,userPass的值]
        for (HashMap<String,String> property:propertiesList)
        {
            System.out.println("*****");
            String propertyName = property.get("name");
            // 拼接出getter方法的完整方法名
            String propertyGetter = "get" + captureName(propertyName);
            System.out.println(property+"的getter为"+propertyGetter);
            // 反射出getter方法,并执行方法,得到属性值
            Method getterMethod = beanClass.getDeclaredMethod(propertyGetter);
            String beanProperty = (String) getterMethod.invoke(beanObject);
            System.out.println("其反射结果为:"+beanProperty);
            valueList.add("'" + beanProperty + "'");
        }
    }
}
```

同理,将 xml 中定义的表的列名(column)从 List 中拿出,存到一个 List 中

```
// 2.查找到对应的column名:user_name,user_pass
ArrayList<String> columnList = new ArrayList<>();
columnList.add(class_ID);
// 2.1 构建ArrayList [userID*,user_name,user_pass]
for (HashMap<String,String> property:propertiesList)
{
    columnList.add(property.get("column"));
}
}
```

根据属性 tableName 判断这个表是否存在

```
// 3.根据tableName判断表是否存在
if (!isTableExist(class_table))
{
    // 不存在
    // 3.1.建表
    System.out.println("表格不存在,建表");
    createTable();
}
}
```

建表的方法 createTable()

```
private void createTable()
{
    System.out.println("开始创建表格");
    Connection connection = this.openDBConnection();
    try {
        Statement statement = connection.createStatement();

        // sql语句拼接
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("CREATE TABLE ");
        stringBuilder.append(class_table);
        stringBuilder.append(" (");
        stringBuilder.append(class_ID);
        stringBuilder.append(" VARCHAR(255) not NULL, "); // 类型写死了
        for (HashMap<String,String> property:propertiesList)
        {
            String columnName = property.get("column");
            stringBuilder.append(columnName);
            stringBuilder.append(" VARCHAR(255), "); // 写死了,其实应该按照property的type构造的
        }
        stringBuilder.append(" PRIMARY KEY ( ");
        stringBuilder.append(class_ID);
        stringBuilder.append(" )");
        String sql = stringBuilder.toString();
        System.out.println("表格创建语句为:" + sql);

        statement.executeUpdate(sql);
        statement.close();
        this.closeDBConnection(connection, pstmt: null, rs: null);
    } catch (SQLException e) {
        System.out.println("创建表格失败"+e);
    }
}
```

之后利用 StringBuilder 拼接 sql 语句,这里使用了之前的两个 list

```
// 4.构建sql语句,需要tableName,两次循环,两次循环的次序不能错
// "INSERT INTO "+tableName+"(USERNAME,USERPASS,USERID) VALUES(?,?,"+userID+")";
StringBuilder sb = new StringBuilder();
sb.append(" INSERT INTO ");
sb.append(class_table);
sb.append(" (");
for (String column:columnList)
{
    sb.append(column);
    sb.append(",");
}
// 删去最后一个逗号
sb.deleteCharAt(sb.length()-1);
sb.append(") VALUES(");
for (String value:valueList)
{
    sb.append(value);
    sb.append(",");
}
sb.deleteCharAt(sb.length()-1);
sb.append(")");
String sql = sb.toString();
System.out.println("insert()插入语句为:"+sql);
```

执行 sql 语句

```
// 5.执行语句
Connection connection = this.openDBConnection();
PreparedStatement preparedStatement = connection.prepareStatement(sql);
int rowNum = preparedStatement.executeUpdate();
System.out.println("        插入成功,插入行数为:"+rowNum);
this.closeDBConnection(connection,preparedStatement, rs: null);

insertResult = true;
catch (Exception e) {
    System.out.println("insert()出错"+e);
}
```


将查询结果存为一个 HashMap 组成的 List,这个 List 将会是返回对象的一个属性

构造方法为从表单得到的 userName 与 userPass


```

public boolean signUp()
{
    // 1. Conversation
    Conversation conversation = new Conversation(this.getClass().getName());
    // 2. 调用insert()
    boolean insertResult = conversation.insert( beanObject this);
    // 3. 得到结果()
    return insertResult;
}

```

登陆方法

考虑到懒加载的情况

因为这里需要立刻验证密码是否正确所以如果返回的结果是懒加载的,则需要立刻对懒加载的结果执行数据库查询(调用 notLazy())

```

public boolean signIn()
{
    Conversation conversation = new Conversation(this.getClass().getName());
    String userPassFromDB = null; // 查询出的密码
    // 1. 执行查询
    QueryResult queryResult = conversation.query( beanObject this, beanPropertyName: "userName");
    // 2. 验证结果是否为懒加载
    if (queryResult.isLazy())
    {
        // 是懒加载
        System.out.println("signIn得到的查询结果是懒加载");
        // 2.1 执行懒加载方法获得结果
        ArrayList<HashMap<String, String>> hashMapsList = queryResult.notLazy();
        for(HashMap<String, String> hashMap:hashMapsList)
        {
            userPassFromDB = hashMap.get("userPass"); // 反正这个情况只会会有一个值
            System.out.println("数据库中读出的密码为:"+userPassFromDB);
        }
    }
    else {
        // 不是懒加载
        System.out.println("signIn得到的查询结果不是懒加载");
        // 2.1
        ArrayList<HashMap<String, String>> resultList = queryResult.getResultList();
        for(HashMap<String, String> hashMap:resultList)
        {
            userPassFromDB = hashMap.get("userPass"); // 反正这个情况只会会有一个值
            System.out.println("数据库中读出的密码为:"+userPassFromDB);
        }
    }
    // 3. 匹配密码
    if (userPass.equals(userPassFromDB))
        return true; // 相等
    else return false;
}

```

7. 懒加载的逻辑

其核心在查询的返回类 `QueryResult` 中,如果查询的值是懒加载的话,则将拼接出的 SQL 语句保存下来

```
public class QueryResult {

    // 懒加载标志
    private boolean lazyFlag;
    // 返回结果,如果是懒加载则为空,因为有时可能会返回多个查询结果,所以用List
    private ArrayList<HashMap<String,String>> resultList = null;
    // 存放对应的bean名称
    private String beanPath;

    // 存放sql信息,在实际需要调用result时进行数据库查询,其实beanName好像也应该加进去
    private String sql;
    // private String driver_class;
    // private String url_path;
    // private String db_username;
    // private String db_userpassword;

    public QueryResult(boolean lazyFlag) { this.lazyFlag = lazyFlag; }

    public void setLazySQL(String sql)
    {
        // 设置懒加载相关的信息
        System.out.println("是懒加载,存放queryResult的SQL信息");
        this.sql = sql;
        // this.driver_class = driver_class;
        // this.url_path = url_path;
        // this.db_username = db_username;
        // this.db_userpassword = db_userpassword;
    }
}
```

如果不是懒加载,则保存结果属性的 `resultList` (HashMap 组成的 List) 会有相关值(query 方法利用 setter 设置的), Bean 可以直接使用它的 getter 来得到查询结果

```
public ArrayList<HashMap<String,String>> getResultList() { return resultList; }

public void setResultList(ArrayList<HashMap<String,String>> resultList) {
    // 这个setter将要在Conversation中调用
    this.resultList = resultList;
}
```

判断返回的对象是否为懒加载的方法

```
public boolean isLazy() { return lazyFlag; }
```

这个方法会创建一个 Conversation 对象,调用 Conversation 的单词查询方法
Conversation 的单词查询方法(对 SQL 语句进行查询),返回的 HashMap 组成的 List 也就是 notLazy()的返回结果

[illegible]

4. 结论

1. 对 Action 进行相关的修改

```
public class RegisterAction {  
    public String handleRegister(String userName, String pwd) {  
        UserBean userBean = new UserBean(userName, pwd);  
        if (userBean.signUp())  
        {  
            return "success";  
        }else  
        {  
            return "handleRegister()出错";  
        }  
    }  
}
```

```
public class LoginAction {  
    public String handleLogin(String name, String pwd) {  
        UserBean userBean = new UserBean(name, pwd);  
        if (userBean.signIn())  
        {  
            return "success";  
        }else {  
            return "failure";  
        }  
    }  
}
```

2. 注册结果:

用户注册

登录名：

密码：

如果您还未注册请点击[此处](#)注册！

```
开始执行表格监测方法
创建数据库连接,加载驱动:
Sun Jan 07 15:25:08 CST 2018 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL
关闭数据库连接
数据库连接关闭成功
表格监测方法执行结束:false
表格不存在,建表
开始创建表格
创建数据库连接,加载驱动:
Sun Jan 07 15:25:09 CST 2018 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL
表格创建语句为:CREATE TABLE user3 (userID VARCHAR(255) not NULL, user_name VARCHAR(255), user_pass VARCHAR(255), PRIMARY KEY ( userID ))
关闭数据库连接
数据库连接关闭成功
insert()插入语句为: INSERT INTO user3 (userID,user_name,user_pass) VALUES('0','tom','123')
创建数据库连接,加载驱动:
Sun Jan 07 15:25:09 CST 2018 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL
插入成功,插入行数为:1
关闭数据库连接
数据库连接关闭成功
反射后,获取执行对应类的对应方法的结果为success
```

如果您还未注册请点击[此处](#)注册！

可以看到这里不是懒加载的
为了测试懒加载,修改 `or_mapping` 中 `userName` 对一个的 `lazy`

用户登录

登录名: tom

密码: ...

登录 重置

如果您还未注册请点击[此处](#)注册！

Water

控制台中

[illegible]

可以看到这次的信息是懒加载的,而且没有在 `query` 中执行数据库的查询,而是到了 `UserBean` 的 `signIn` 才执行了数据库的查询

由于本次作业要求大多数的功能必须动态的实现,程序的逻辑较为复杂,作业做了很久.其中懒加载的方法没有做到使用 Hibernate 的 Invocation 的动态代理方式实现,而是使用了类似静态代理的方式来实现的.

需求 6 中的也没有做到 DBMS 的修改.

很惭愧.没有完成老师要求的作业目标.

5.参考文献

以上内容的理论知识点或技术点如果参考了网上或印刷制品，请在这里罗列出来