河南大學 软件学院

Java EE技术

第9章 DAO设计模式

本章要点

- DAO设计模式的概念
- DAO设计模式的优势
- 在Java中实现DAO的步骤

1. DAO设计模式的概念

- DAO设计模式是属于Java EE数据层的操作，使用DAO设计模式可以简化大量的代码编写和增加程序的可移植性。
- 回忆之前使用JDBC操作数据库，都是在JSP页面中写JDBC代码，或者借助于Java Bean和Servlet实现，这样导致JSP页面中包含大量的HTML代码和JSP代码，显示和功能 代码混合在一起，难以维护，程序的可读性也较差。由于在JSP页面中需要JDBC，还要导入“java.sql.*”包。这样的设计非常不合理。在经典的MVC设计模式中，JSP页面仅用来作表现层，即关注数据的显示，不需要关注数据从何处而来，如何获取这些数据。
- 使用DAO设计模式可以很好地解决上述问题， DAO设计模式包括下面几部分：

  （1）DAO接口；

  （2）DAO实现类；

  （3）DAO工厂类；

  （4）VO类；

  （5）数据库连接类。

1. DAO设计模式的概念

- DAO五部分详解之数据库连接类

数据库连接类的主要功能是连接数据库并获得连接对象，以及关闭数据库等。

通过使用数据库连接类，可以大大简便开发，在需要进行数据库连接时，只需创建该类的实例，并调用其中的方法就可以获得数据库连接对象，不必再进行重复的操作。

1. DAO设计模式的概念

- DAO五部分详解之VO类

  VO类是一个包含属性与表中字段完全对应的类，在该类中提供了setter和getter方法来设置并获取该类中的属性。

1. DAO设计模式的概念

- DAO五部分详解之DAO接口

  DAO接口中定义了所有的用户操作，如添加记录、修改、删除、查找（CRUD）等，不过是接口，定义的都是抽象方法，需要实现类去具体实现这些方法。

1. DAO设计模式的概念

- DAO五部分详解之DAO实现类

  DAO实现类实现了DAO接口，并实现了DAO接口中所有抽象方法，在DAO实现类中通过数据库连接类操作数据库。

1. DAO设计模式的概念

- DAO五部分详解之DAO实现类

在没有DAO工厂类的情况下，必须通过创建DAO实现类的实例来完成数据库操作。这里就必须知道具体的子类，对于后期的修改非常不便。如后期要创建一个操作Oracle的DAO实现类，这里就必须修改所有的DAO实现类的代码。

使用DAO工厂类就能解决这个问题，可以通过该DAO工厂类的一个静态方法来获得实现类的实例。这里如果需要替换DAO实现类，只需要修改DAO工厂类中的方法代码，而不必修改所有的操作数据库代码。

2. DAO设计模式的优势

- 用户不需要知道底层的具体实现,只需要操作接 口;同时实现了前台和后后台的分离;(DAO中只需要操作对象);有利于代码的重用.(如果用户需要增加新的功能时候,我只需要在DAO这接口中添加的抽 象的方法,然后在DAOImpl中实现用户所增加的功能,就能很好的完成添加用户功能的。
- DAO的整体设计流程:
  JSP--->DAO工厂-->具 体实现类-->完成数据库的具体操作

3. 在Java中实现DAO的步骤

- 使用DAO完成数据库操作

（1）添加记录

（2）更新记录

（3）删除记录

（4）查找记录

3. 在Java中实现DAO的步骤

DAO 设计的类和页面有：

1、VO： 值对象，即POJO，普通JAVABEAN 或者是传输对象(TO)

2、接口：用来定义有哪些操作的比如：CURD

3、接口的实现类

4、数据库连接池类 通常内部采用预编译形式即：PreparedStatement

5、工厂类

6、前台页面

DAO整体设计 是采用以下模式：

调用处 ---> 工厂模式 ----->具体子类实现 ----> 完成数据库操作

|----------VO(值对象)----------|

3. 在Java中实现DAO的步骤

- 下面以一个登陆程序的例子，结合地理解DAO模式。

- 数据库的连接类 (DataBaseConnection.java)

```
package com.henu.common
import java.sql.*;
public class DataBaseConnection {
private  final String DBDRIVER="com.mysql.jdbc.Driver";
private final String DBURL="jdbc:mysql://localhost:3306/student";
private final String DBUSER="root";
private final String DBPASSWORD="root";
private Connection conn=null;
public DataBaseConnection(){
  try{
   Class.forName(DBDRIVER);
      conn=DriverManager.getConnection(DBURL,DBUSER,DBPASSWORD);
  }catch(Exception e){

  }
}
   public Connection getConnection(){
     return this.conn;
    }
    public void close() {
     try {
   this.conn.close();
  } catch (SQLException e) {
   e.printStackTrace();
  }
    }
}
```

2.DAO接口，定义要实现的功能

```
package com.henu.dao;
    import com.henu.PersonVo.vo.*;
    import java.util.*;
    public interface PersonDAO {
    public void insert(PersonVo vo) throws Exception;
    public void update(PersonVo vo) throws Exception;
    public void delete (String id)throws Exception;
    public PersonVo queryById(String id)throws Exception;
    public List queryAll()throws Exception;
    public List queryByLike(String cmd)throws Exception;

    }
```

• 3. 操作的对象 VO

```
package com.henu.PersonVo.vo;
public class PersonVo {
   private String id;
   private String userName;
   private String password;
   private String age;
   private String email;
public String getId() {
return id;
}
public void setId(String id) {
this.id = id;
}
public String getUserName() {
return userName;
}
public void setUserName(String userName) {
this.userName = userName;
}
…….

}
```

4.具体实现DAO接口

```
public class PersonDAOImpl implements PersonDAO {
    public void insert(PersonVo v) throws Exception{
        String sql="INSERT INTO member (id,userName,password,age,email) VALUES (?,?,?,?,?)";
        PreparedStatement st=null;
        DataBaseConnection dbc=null;
        Connection con=null;
        try{
        dbc=new DataBaseConnection();
        con=dbc.getConnection();
        st=con.prepareStatement(sql);
        st.setInt(1, Integer.parseInt(v.getId()));
        st.setString(2, v.getUserName());
        st.setString(3, v.getPassword());
        st.setString(4, v.getAge());
        st.setString(5, v.getEmail());
        st.executeUpdate();
        st.close();
        con.close();

        }catch(Exception e){
        //System.out.println("操作异常1");
         e.printStackTrace();
       }finally{
        dbc.close();
       }


    }
    public void update(PersonVo v) throws Exception{……}   ……
```

- **5.DAO工厂**

```
package com.henu.DAOFactory;
import com.henu.dao.PersonDAO;
import com.henu.PersonDAO.daoimpl.*;
public class PersonDAOFactory {
    public static PersonDAO
getPersonDAOInstance(){
     return new PersonDAOImpl();
    }
}
```

- **6.调用处（可以是Servlet，或者Struts中的Action）**

```
……
PersonVo v=new PersonVo();
        v.setId(id);
        v.setUserName(userName);
        v.setPassword(password);
        v.setAge(age);
        v.setEmail(email);
        try {
      PersonDAOFactory.getPersonDAOInstance().insert(v);
     } catch (Exception e) {

      e.printStackTrace();
     }
```

- **7.JSP页面**

```
<%@ page language="java" pageEncoding="ISO-8859-1"%>
    <%@ taglib uri="/struts-tags" prefix="s"%>

    <html>
    <head>
     <title>JSP for InsertForm form</title>
    </head>
    <body>
     <s:form action="/insert" method="post">
        ID:<s:text name="ID" />
        userName: <s:text name="userName" />
        password: <s:password name="password" />
        age:<s:text name="age" />
        email:<s:text name="email" />
      <s:submit/><s:cancel/>
     </s:form>
    </body>
    </html>
```

总结：为什么使用DAO

- **1. 数据存储逻辑的分离**
  通过对数据访问逻辑进行抽象，为上层机构提供抽象化的数据访问接口。业务层无需关心具体的select,insert,update操作，这样，一方面避免了业务代码中混杂JDBC调用语句，使得业务落实实现更加清晰，另一方面，由于数据访问几口语数据访问实现分离，也使得开发人员的专业划分成为 可能。某些精通数据库操作技术的开发人员可以根据接口提供数据库访问的最优化实现，而精通业务的开发人员则可以抛开数据曾德繁琐细节，专注于业务逻辑编码。

- **2. 数据访问底层实现的分离**

  DAO模式通过将数据访问计划分为抽象曾和实现曾，从而分离了数据使用和数据访问的地称实现细节。这意味着业务层与数据访问的底层细节无关，也就是说，我 们可以在保持上层机构不变得情况下，通过切换底层实现来修改数据访问的具体机制，常见的一个例子就是，我们可以通过仅仅替换数据访问曾实现，将我们的系统 部署在不同的数据库平台之上。

- **3. 资源管理和调度的分离**
  在数据库操作中，资源的管理和调度是一个非常值得关注的主题。大多数系统的性能瓶颈往往并非集中于业务逻辑处理本身。在系统涉及的各种资源调度过 程中，往往存在着最大的性能黑洞，而数据库作为业务系统中最重要的系统资源，自然也成为关注的焦点。DAO模式将数据访问逻辑从业务逻辑中脱离开来，使得 在数据访问层实现统一的资源调度成为可能，通过数据库连接池以及各种缓存机制(Statement Cache,Data Cache等，缓存的使用是高性能系统实现的一个关键所在)的配合使用，往往可以保持上层系统不变的情况下，大幅度提升系统性能。

- **4. 数据抽象**

在直接基于JDBC调用的代码中，程序员面对的数据往往是原始的RecordSet数据集，诚然这样的数据集可以提供足够的信息，但对于业务逻辑开发过程而言，如此琐碎和缺乏寓意的字段型数据实在令人厌倦。

DAO 模式通过对底层数据的封装，为业务曾提供一个面向对象的接口，使得业务逻辑开发员可以面向业务中的实体进行编码。通过引入DAO模式，业务逻辑更加清晰， 且富于形象性和描述性，这将为日后的维护带来极大的便利。试想，在业务曾通过Contact.getName方法获得客户姓名，相对于直接通过SQL语 句访问数据库表并从ResultSet中获得某个字符型字段而言，哪种方式更加易于业务逻辑的形象化和简洁化？