

# Web应用开发 之 JDBC数据库访问



# 五、 DAO设计模式

- 5.1 设计持久对象
- 5.2 设计DAO对象
- 5.3 使用DAO对象

## 五、DAO设计模式

- DAO（Data Access Object）称为数据访问对象。  
DAO设计模式可以在使用数据库的应用程序中实现业务逻辑和数据访问逻辑分离，从而使应用的维护变得简单。
- 它通过将数据访问实现（通常使用JDBC技术）封装在DAO类中，提高应用程序的灵活性。

## 5.1 设计持久对象

- 在分布式Web应用中，经常需要把数据从表示层传输到业务层，或者从业务层传输到表示层。跨层传输数据最好的方法是使用持久对象（Persistent Object）。
- 持久对象只包含数据元素，不包含任何业务逻辑，业务逻辑由业务对象实现。
- 持久对象必须是可序列化的，也就是它的类必须实现`java.io.Serializable`接口。

## 5.1 设计持久对象

- Customer类的对象就是持久对象。
- 该持久对象用于在程序中保存应用数据，并可实现对象与关系数据的映射，它实际上是一个可序列化的JavaBeans。

```
package com.model;
import java.io.Serializable;
public class Customer implements Serializable{
    private String cust_id;
    private String cname;
    private String email;
    private double balance;
    //setter和getter方法
}
```

## 5.2 设计DAO对象

- 先定义一个基类BaseDao连接数据库，通过该类可以获得一个连接对象
- 然后定义CustomerDao类，定义添加客户、查找客户、查找所有客户等方法。

# BaseDAO.java

```
1. package com.dao;
2. import java.sql.*;
3. import javax.sql.DataSource;
4. import javax.naming.*;
5. public class BaseDao {
6.     DataSource dataSource;
7.     public BaseDao () {
8.         try {
9.             Context context = new InitialContext();
10.            dataSource = (DataSource)context.lookup("java:comp/env/jdbc/sampleDS");
11.        }catch(NamingException ne){
12.            System.out.println("Exception:"+ne);
13.        }
14.    }
15.    public Connection getConnection()throws Exception{
16.        return dataSource.getConnection();
17.    }
18. }
```

## 5.2 设计DAO对象

### ➤ 程序CustomerDao.java

- CustomerDao类继承了BaseDao类并实现了添加客户、查询客户、查询所有客户的方法。
- 该类没有给出修改记录和删除记录的方法，可自行补充完整。



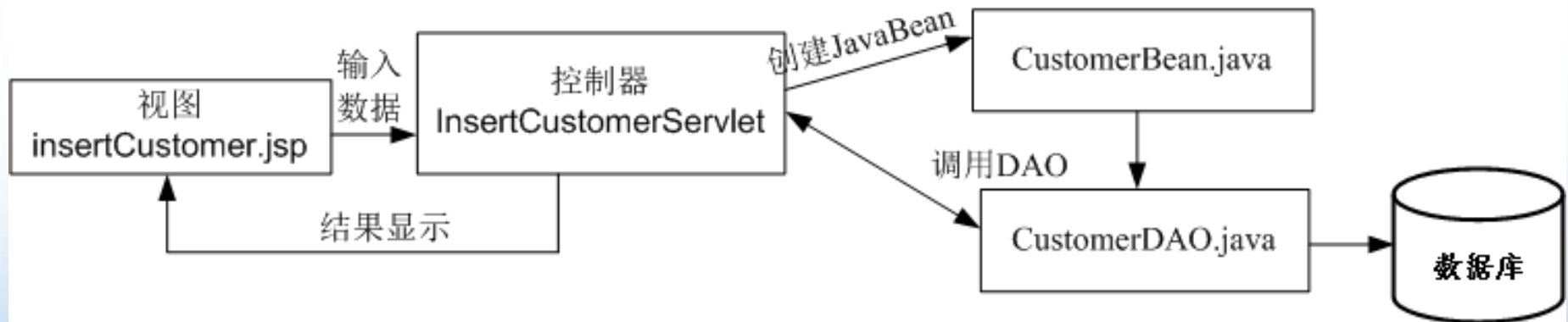
# DAO类: CustomerDao.java

```
package com.dao;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import java.util.ArrayList;
import com.model.CustomerBean;
public class CustomerDAO{
    private Connection conn = null;
    public CustomerDAO(){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            connectSQL="jdbc:mysql://localhost:3306/bank?user=root&password=111111";
        }catch(ClassNotFoundException e1){ }
    }
    public ArrayList<CustomerBean> selectCustomer(){
        //..... 查询所有客户信息
    }
    public boolean insertCustomer(CustomerBean customer){
        //.....插入一条客户记录
    }

    public CustomerBean searchCustomer(String custName){
        //..... 按姓名检索客户记录
    }
}
```

# DAO对象应用

- insertCustomer.jsp页面：通过一个表单提供向数据库中插入的数据
- InsertCustomerServlet.java：使用DAO对象和传输对象，通过JDBC API实现将数据插入到数据库中



## 5.3 使用DAO对象

- addCustomer.jsp页面通过一个表单提供向数据库中插入的数据。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html><head> <title>Input a Customer</title></head>
<body>
<font color=red>${result}</font>
<p>请输入一条客户记录</p>
<form action = "addCustomer.do" method = "post">
<table>
<tr><td>客户号: </td> <td><input type="text" name="cust_id" ></td></tr>
<tr><td>客户名: </td> <td><input type="text" name="cname" ></td></tr>
<tr><td>Email: </td><td><input type="text" name="email"></td></tr>
<tr><td>余额: </td><td><input type="text" name="balance" ></td></tr>
<tr><td><input type="submit" value="确定" ></td>
<td><input type="reset" value="重置" ></td>
</tr>
</table>
</form>
</body></html>
```

## 5.3 使用DAO对象

- **AddCustomerServlet.java**使用了DAO对象和传输对象，通过JDBC API实现将数据插入到数据库中。
- 从请求对象中获得请求参数并进行编码转换，创建一个**Customer**对象
- 调用**CustomerDao**对象的**insertCustomer()**将客户对象插入数据库中
- 根据该方法执行结果将请求再转发到**addCustomer.jsp**页面

# AddCustomerServlet.java

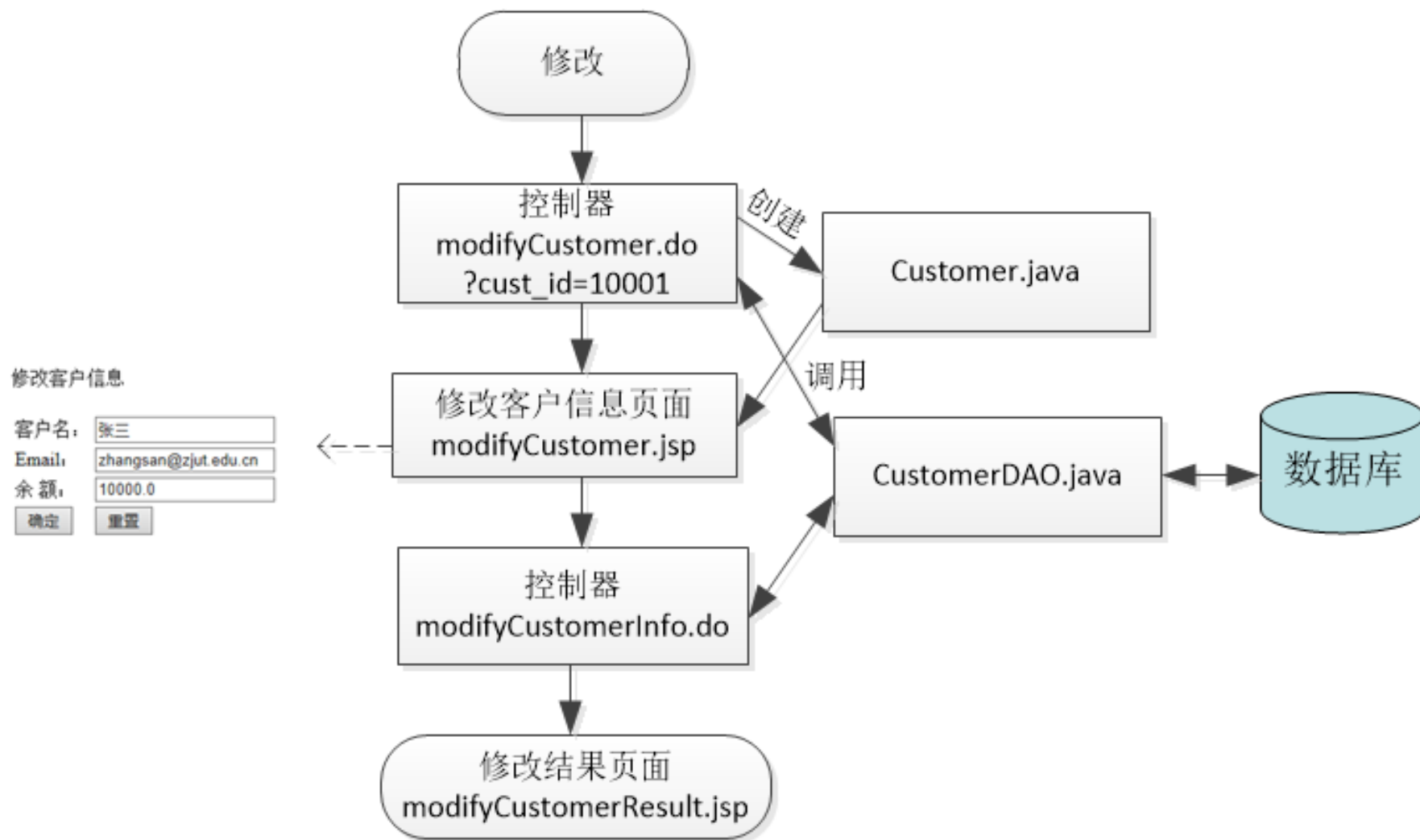
```
@WebServlet("/addCustomer.do")
public class AddCustomerServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        CustomerDao dao = new CustomerDao();
        Customer customer = new Customer();
        String message = null;
        try {
            customer.setCust_id(request.getParameter("cust_id"));
            customer.setName(request.getParameter("cname"));
            customer.setEmail(request.getParameter("email"));
            customer.setBalance(Double.parseDouble(request.getParameter("balance")));
            boolean success = dao.addCustomer(customer);
            if (success) {
                message = "<li>添加成功! </li>";
            } else {
                message = "<li>添加失败</li>";
            }
        } catch (Exception e) {
            message = "<li>出现异常</li>";
        }
        request.setAttribute("result", message);
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/addCustomer.jsp");
        rd.forward(request, response);
    }
}
```

# 思考？

如何采用基于DAO的MVC设计模式实现查询、列出所有客户信息、修改和删除和功能？

- 查询：输入客户姓名查询，支持模糊查询？
- 列出所有客户信息：按客户号、姓名、Email、余额和操作，其中操作包括修改和删除链接；
- 修改：根据客户号修改该客户的信息；
- 删除：根据客户号删除该客户的信息；

# 修改功能业务流程



# 修改页面modifyCustomer.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html><head> <title>修改客户信息</title></head>
<body>
<font color=red>${result}</font>
<p>修改客户信息</p>
<form action = "modifyCustomerInfo.do" method = "post">
  <table>
    <tr><td></td> <td><input type="hidden" name="cust_id"
value="${customer.cust_id}"></td></tr>
    <tr><td>客户名: </td> <td><input type="text" name="cname"
value="${customer.cname}"></td></tr>
    <tr><td>Email: </td><td><input type="text" name="email"
value="${customer.email}"></td></tr>
    <tr><td>余 额: </td><td><input type="text" name="balance"
value="${customer.balance}"></td></tr>
    <tr><td><input type="submit" value="确定" ></td>
      <td><input type="reset" value="重置" ></td>
    </tr>
  </table>
</form>
</body></html>
```



# 修改控制器ModifyCustomerInfo.java

```
@WebServlet("/modifyCustomerInfo.do")
public class ModifyCustomerInfo extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String cust_id=request.getParameter("cust_id");
        String cname=request.getParameter("cname");
        String email=request.getParameter("email");
        String balance=request.getParameter("balance");
        Customer customer=new Customer();
        customer.setCust_id(cust_id);
        customer.setCname(cname);
        customer.setEmail(email);
        customer.setBalance(Double.parseDouble(balance));
        CustomerDao dao = new CustomerDao();
        String message="";
        if(dao.modifyCustomer(customer)) {
            message="客户信息修改成功! ";
        }
        else {
            message="客户信息修改失败! ";
        }
        request.setAttribute("message", message);
        RequestDispatcher rd =
        getServletContext().getRequestDispatcher("/modifyCustomerResult.jsp");
        rd.forward(request, response);
    }
}
```

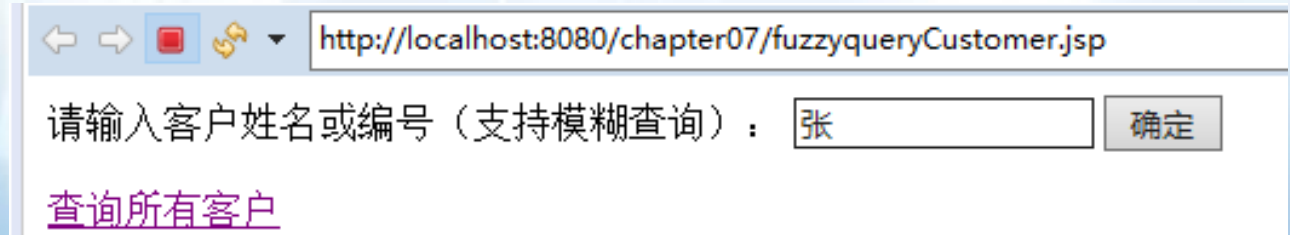
# 修改DAO类CustomerInfoDao.java

```
public class CustomerDao extends BaseDao {
    public boolean modifyCustomer(Customer customer) {
        String sql = "update customers set cname=?,email=?,balance=?
where cust_id=?";
        try (Connection conn = dataSource.getConnection();
PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, customer.getCname());
            pstmt.setString(2, customer.getEmail());
            pstmt.setDouble(3, customer.getBalance());
            pstmt.setString(4, customer.getCust_id());

            pstmt.executeUpdate();
            return true;
        } catch (SQLException se) {
            se.printStackTrace();
            return false;
        }
    }
}
```

# 示例：模糊查询功能的查询页面

```
<%@ page contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<html>
<head>
<title>模糊查询客户</title>
</head>
<body>
<form action="fuzzyqueryCustomer.do" method="post">
请输入客户姓名或编号（支持模糊查询）：
<input type="text" name="cname" size="15"> <input
type="submit" value="确定">
</form>
<p>
<a href="allCustomer.do">查询所有客户</a>
</p>
</body>
</html>
```



http://localhost:8080/chapter07/fuzzyqueryCustomer.jsp

请输入客户姓名或编号（支持模糊查询）：

[查询所有客户](#)

# 示例：模糊查询功能的控制器

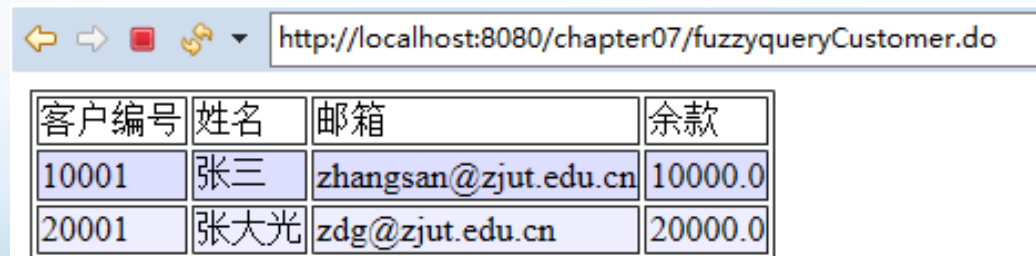
```
@WebServlet("/fuzzyqueryCustomer.do")  
  
public class FuzzyQueryCustomer extends HttpServlet {  
    public void doPost(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
        request.setCharacterEncoding("UTF-8");  
        String cname = request.getParameter("cname");  
        CustomerDao dao = new CustomerDao();  
        ArrayList<Customer> customer = dao.findByFuzzyName(cname);  
        request.setAttribute("customer", customer);  
        RequestDispatcher rd =  
getServletContext().getRequestDispatcher("/showFuzzyQueryCustomer.jsp");  
        rd.forward(request, response);  
    }  
}
```

# 示例：模糊查询功能的DAO类

```
public class CustomerDao extends BaseDao {  
    // 根据姓名模糊查询  
    public ArrayList<Customer> findByName(String cname) {  
        ArrayList<Customer> custList = new ArrayList<Customer>();  
        String sql = "SELECT cust_id,cname,email,balance" +  
            " FROM customers WHERE cname like ? or cust_id like ?";  
        try (Connection conn = dataSource.getConnection();  
            PreparedStatement pstmt = conn.prepareStatement(sql)) {  
            pstmt.setString(1, "%" + cname + "%");  
            pstmt.setString(2, "%" + cname + "%");  
            try (ResultSet rst = pstmt.executeQuery()) {  
  
                while (rst.next()) {  
                    Customer customer = new Customer();  
                    customer.setCust_id(rst.getString("cust_id"));  
                    customer.setCname(rst.getString("cname"));  
                    customer.setEmail(rst.getString("email"));  
                    customer.setBalance(rst.getDouble("balance"));  
                    custList.add(customer);  
                }  
            }  
        } catch (SQLException se) {  
            return null;  
        }  
        return custList;  
    }  
}
```

# 示例：模糊查询功能的结果显示页面

```
<table border="1">
<tr>
<td>客户编号</td>
<td>姓名</td>
<td>邮箱</td>
<td>余款</td>
</tr>
<c:forEach var="customer" items="${requestScope.customer}" varStatus="status">
<!--为奇数行和偶数行设置不同的背景颜色-->
    <c:if test="${status.count%2==0}">
        <tr style="background: #eeeeff">
</c:if>
    <c:if test="${status.count%2!=0}">
        <tr style="background: #dedeff">
</c:if>
<!--用EL访问作用域变量的成员-->
        <td>${customer.cust_id}</td>
        <td>${customer.cname}</td>
        <td>${customer.email}</td>
        <td>${customer.balance}</td>
    </tr>
</c:forEach>
</table>
```



客户编号	姓名	邮箱	余款
10001	张三	zhangsan@zjut.edu.cn	10000.0
20001	张大光	zdg@zjut.edu.cn	20000.0

# 作业

1、测试基于DAO的MVC设计模式实现的添加客户信息功能，包括BaseDao.java、CustomerDao.java、addCustomer.jsp、AddCustomerServlet.java,使之能正常运行。要求dao类在com.dao包下，servlet类在com.controller包下，javabeans类在com.model包下，数据库访问采用数据源实现，然后增加查询、列出所有客户信息、修改和删除等功能。

- 查询：输入客户姓名查询，支持模糊查询。
- 列出所有可出信息：按客户号、姓名、Email、余额和操作，其中操作包括修改和删除链接；
- 修改：根据客户号修改该客户的信息；
- 删除：根据客户号删除该客户的信息；

# 作业

2、实现某师生健康码管理系统的系统管理员管理功能，具体要求如下：

系统管理员登录后可设置学院、专业、班级等信息并进行查询、修改、删除等管理功能，可单独添加或批量导入教师数据和学生数据并进行查询、修改、删除等管理功能，教师数据包括姓名、身份证号、工号、学院、角色（系统管理员、校级管理员、院级管理员、普通教师），学生数据包括姓名、身份证号、学号、学院、专业、班级等信息。



# 练习

**3、采用基于DAO的MVC模式实现一个简单的银行柜台业务处理系统，包括用户申请账户、登录自己的账户、查看账户余额、取款和存款等功能，具体要求如下：**

- ① 账户信息包括账号（**19**位数字）、账户密码、姓名、身份证号码、开户银行、**email**、手机号、开户时间、账户余额（初始为0）
- ② 申请账户时，**19**位数字的账号由系统随机给定、账户密码为**6**位数字，开户时间为当前系统时间（不需录入），所有信息都要有效性判断，所有用户的账户信息都存入数据库中
- ③ 登录账户需输入账号和密码，登录成功后，将账户写入**session**，在查看账户余额、取款和存款等页面，判断账户的**session**是否存在，若存在，则实现查看账户余额、取款和存款等业务，否则重定向到登录页面。
- ④ 用户不能查看别人的账户或从别人账户上取钱。
- ⑤ 要求画出系统的功能流程图