Web CRUD&项目-MVC

课程目标

- 理解并掌握 Web CRUD 执行流程。
- 理解并掌握 Web CRUD 功能实现。
- 理解并掌握请求分发器的作用和设计。
- 理解 MVC 思想。

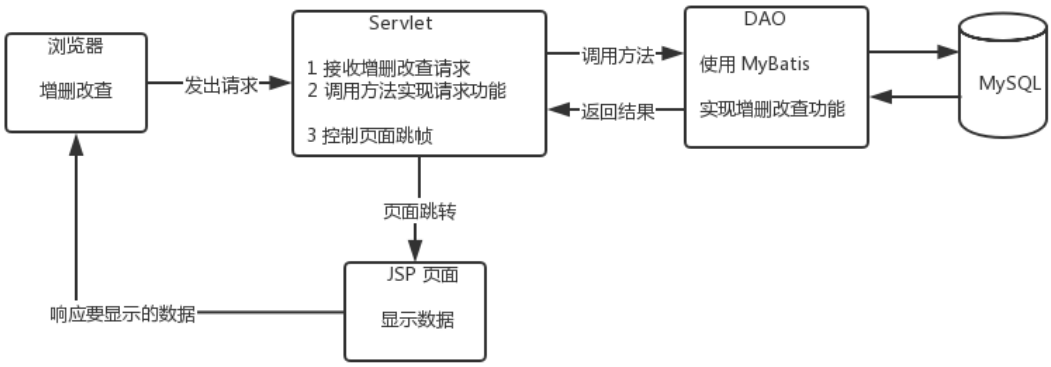
一、Web CRUD 需求及技术架构

需求: 完成商品表中数据的 CRUD（Web），让用户能够通过浏览器的相关操作，完成商品信息的增删改查。

1、技术架构

- 页面显示：JSP
- 接受用户请求：Servlet
- 和数据库交互：MyBatis

2、技术交互思路图

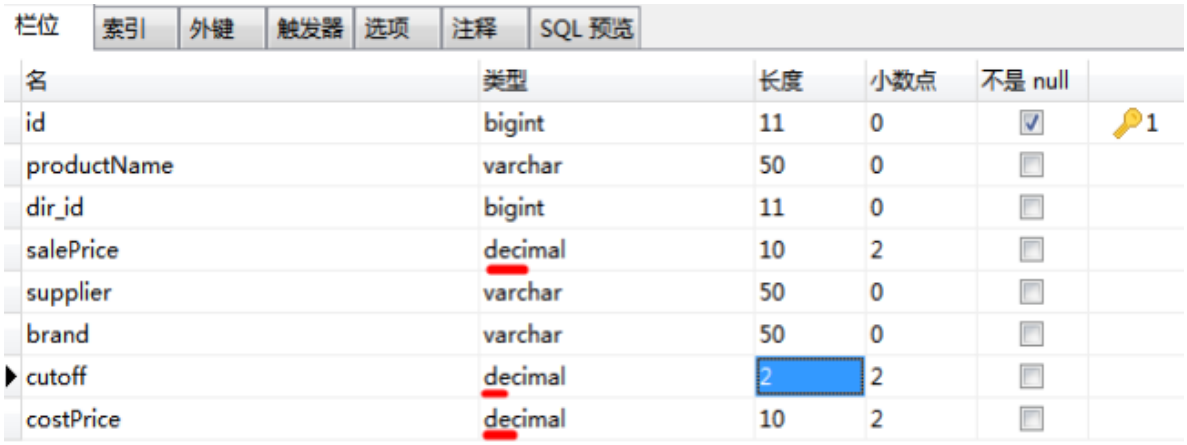


二、项目搭建

1、创建 Web 项目

导入项目需要依赖的 jar 包。

2、创建 product 表




```
CREATE TABLE `product` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `productName` varchar(255) DEFAULT NULL,
  `dir_id` bigint(20) DEFAULT NULL,
  `salePrice` decimal(10,2) DEFAULT NULL,
  `supplier` varchar(255) DEFAULT NULL,
  `brand` varchar(255) DEFAULT NULL,
  `cutoff` decimal(10,2) DEFAULT NULL,
  `costPrice` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3、准备配置文件

3.1、db.properties

在 resources 目录下新建此文件。

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/webcrud
username=root
password=admin
```

3.2、log4j.properties

在 resources 目录下新建此文件。

```
log4j.rootLogger=ERROR, stdout
log4j.logger.cn.wolfcode.mapper=TRACE
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

3.3、ProductMapper.xml

在 cn.wolfcode.mapper 包下新建此文件，但文件中先不编写 SQL。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.wolfcode.mapper.ProductMapper">
</mapper>
```

3.4、mybatis-config.xml

在 resources 目录下新建此文件。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

    <properties resource="db.properties"/>
    <!-- 类型别名配置 -->
    <typeAliases>
        <!-- 包名范围不要太大，一般到 domain，在没有注解的情况下，会使用实体类的首字母小写的非限定类名来作为它的别名 -->
        <package name="cn.wolfcode.domain"/>
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${driver}" />
                <property name="url" value="${url}" />
                <property name="username" value="${username}" />
                <property name="password" value="${password}" />
            </dataSource>
        </environment>
    </environments>
    <!-- 关联 Mapper 文件 -->
    <mappers>
        <mapper resource="cn/wolfcode/mapper/ProductMapper.xml"/>
    </mappers>
</configuration>
```

三、后台 CRUD 实现

在 DAO 实现中通过 MyBatis API 完成 Product 表的 CRUD。

1、编写 MyBatisUtil 类

一是减少代码重复，二是 SqlSessionFactory 对象应用只需要一个。

```
package cn.wolfcode.util;

public abstract class MyBatisUtil {
    private static SqlSessionFactory sqlSessionFactory;
    static {
```

```
        InputStream inputStream;
        try {
            inputStream = Resources.getResourceAsStream("mybatis-config.xml");
            sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static SqlSession getSession() {
        return sqlSessionFactory.openSession();
    }
}
```

2、根据表编写 Product 类

```
package cn.wolfcode.domain;

@Data
public class Product {
    private Long id;
    private String productName;
    private Long dir_id;
    private BigDecimal salePrice;
    private String supplier;
    private String brand;
    private BigDecimal cutoff;
    private BigDecimal costPrice;
}
```

3、编写 IProductDAO 及 ProductDAOImpl

```
package cn.wolfcode.dao;

public interface IProductDAO {
    void save(Product product);
    void update(Product product);
    void delete(Long id);
    Product get(Long id);
    List<Product> listAll();
}
```

```
package cn.wolfcode.dao.impl;

public class ProductDAOImpl implements IProductDAO {

    @Override
    public void save(Product product) {
        SqlSession session = MyBatisUtil.getSession();
        session.insert("cn.wolfcode.mapper.ProductMapper.save", product);
        session.commit();
        session.close();
    }
```

```
    @Override
    public void update(Product product) {
        SqlSession session = MyBatisUtil.getSession();
        session.update("cn.wolfcode.mapper.ProductMapper.update", product);
        session.commit();
        session.close();
    }

    @Override
    public void delete(Long id) {
        SqlSession session = MyBatisUtil.getSession();
        session.delete("cn.wolfcode.mapper.ProductMapper.delete", id);
        session.commit();
        session.close();
    }

    @Override
    public Product get(Long id) {
        SqlSession session = MyBatisUtil.getSession();
        Product product =
session.selectOne("cn.wolfcode.mapper.ProductMapper.get", id);
        session.close();
        return product;
    }

    @Override
    public List<Product> listAll() {
        SqlSession session = MyBatisUtil.getSession();
        List<Product> products =
session.selectList("cn.wolfcode.mapper.ProductMapper.listAll");
        session.close();
        return products;
    }
}
```

4、修改 ProductMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.wolfcode.mapper.ProductMapper">
    <insert id="save" useGeneratedKeys="true" keyProperty="id">
        INSERT INTO product(productName, dir_id, salePrice, supplier, brand,
cutoff, costPrice)
        VALUES (#{productName}, #{dir_id}, #{salePrice}, #{supplier}, #{brand},
#{cutoff}, #{costPrice})
    </insert>

    <update id="update">
        UPDATE product SET
            productName = #{productName},
            dir_id = #{dir_id},
            salePrice = #{salePrice},
            supplier = #{supplier},
            brand = #{brand},
```

```
            cutoff = #{cutoff},
            costPrice = #{costPrice}
        WHERE id = #{id}
    </update>

    <delete id="delete">
        DELETE FROM product WHERE id = #{id}
    </delete>

    <select id="get" resultType="Product">
        SELECT * FROM product WHERE id = #{id}
    </select>

    <select id="listAll" resultType="Product">
        SELECT * FROM product
    </select>
</mapper>
```

5、编写单元测试

```
package cn.wolfcode.dao;

public class ProductDAOTest {

    private IProductDAO productDAO = new ProductDAOImpl();

    @Test
    public void testSave() {
        Product product = new Product();
        product.setProductName("罗技无线鼠标");
        product.setDir_id(1L);
        product.setBrand("罗技");
        product.setSupplier("罗技");
        product.setSalePrice(new BigDecimal("60"));
        product.setCutoff(new BigDecimal("0.8"));
        product.setCostPrice(new BigDecimal("48"));
        productDAO.save(product);
    }

    @Test
    public void testUpdate() {
        Product product = new Product();
        product.setProductName("罗技无线鼠标");
        product.setDir_id(1L);
        product.setBrand("罗技");
        product.setSupplier("罗技");
        product.setSalePrice(new BigDecimal("50"));
        product.setCutoff(new BigDecimal("0.8"));
        product.setCostPrice(new BigDecimal("40"));
        productDAO.save(product);
    }

    @Test
    public void testDelete() {
        productDAO.delete(1L);
    }
```

```
    @Test
    public void testGet() {
        System.out.println(productDAO.get(1L));
    }

    @Test
    public void testListAll() {
        System.out.println(productDAO.listAll());
    }
}
```

四、前台 CRUD 实现

前台 CRUD 指如何处理用户在页面操作产品的 CRUD 所涉及到的相关代码：

- JSP：展示数据及提供操作；
- Servlet: 处理用户在页面操作的发送的请求，及对请求做响应。

1、请求分发器设计和实现

1.1、存在的问题

用户所发起的每个请求，按目前所学需要编写一个 Servlet，覆盖其中的 service 方法来处理，那么就会造成如下结果：

- 查询商品列表：需要一个 Servlet 来处理，ProductListServlet；
- 删除指定的商品：需要一个 Servlet 来处理，ProductDeleteServlet；
- 点击编辑和添加都是进入到可编辑的界面: 需要一个 Servlet 来处理，ProductInputServlet；
- 在编辑界面，点击保存：需要一个 Servlet 来处理，ProductSaveOrUpdateServlet。

以上设计问题：

- 每个操作都需要一个 Servlet 来处理请求，100 张表的 CRUD，需要 400 个 Servlet 来处理。
- 类文件爆炸式增长，不利后期归类管理维护。

1.2、解决方案

使用一个 Servlet 类来处理一张表的所有请求操作。

思考：如何请求到同一个Servlet ,但是又能区分是不同的请求

答案：增加一个参数来区分，请求时候多携带一个 cmd 参数来区分是哪种请求，不同的值，表示不同的请求：

- cmd=delete: 删除操作；
- cmd=input: 去新增或去修改操作；
- cmd=saveOrUpdate: 保存或修改操作；
- cmd=list 或者没参数 cmd：列表查询操作。

1.3、代码实现

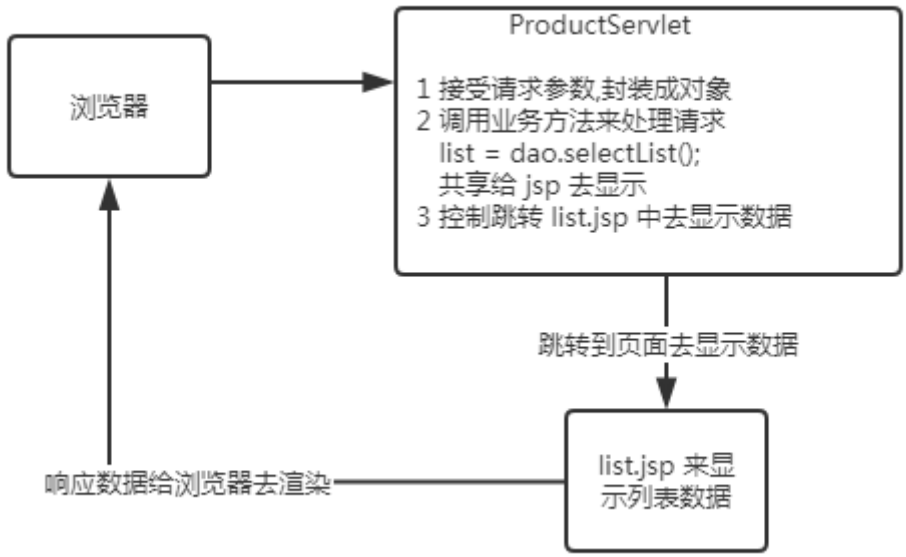
写完启动 Tomcat，访问测试一下。

```
package cn.wolfcode.web.servlet;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/product")
public class ProductServlet extends HttpServlet {

    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 处理请求编码
        req.setCharacterEncoding("UTF-8");
        String cmd = req.getParameter("cmd");
        // 针对参数 cmd 值不同，分发到不同的方法去处理
        if("delete".equals(cmd)){
            delete(req, resp);
        }else if("input".equals(cmd)){
            input(req, resp);
        }else if("saveOrUpdate".equals(cmd)){
            saveOrUpdate(req, resp);
        }else{
            list(req, resp);
        }
    }

    // 处理列表查询
    protected void list(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("ProductServlet.list");
    }

    // 处理删除
    protected void delete(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("ProductServlet.delete");
    }

    // 处理去新增或修改
    protected void input(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("ProductServlet.input");
    }

    // 处理新增或修改
    protected void saveOrUpdate(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        System.out.println("ProductServlet.saveOrUpdate");
    }
}
```

五、查询产品列表

1、流程分析



2、代码实现

2.1、修改 ProductServlet 的 list 方法

```
private IProductDAO productDAO = new ProductDAOImpl();

// 处理列表查询
protected void list(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    // 1 接受请求参数封装成对象
    // 2 调用业务方法去处理请求
    List<Product> products = productDAO.listAll();
    // 把数据共享给 list.jsp 去显示
    req.setAttribute("products", products);
    // 转发到 list.jsp
    req.getRequestDispatcher("/Web-INF/views/product/list.jsp").forward(req,
resp);
}
```

2.2、编写 list.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>产品列表</title>
</head>
<body>
    <a href="#">添加</a>
    <table border="1" cellspacing="0" cellpadding="0" width="80%">
        <tr>
            <th>编号</th>
            <th>货品名</th>
            <th>分类编号</th>
            <th>零售价</th>
            <th>供应商</th>
            <th>品牌</th>
```

```
            <th>折扣</th>
            <th>进货价</th>
            <th>操作</th>
        </tr>
        ${products}
        <c:forEach var="product" items="${products}" varStatus="status">
            <tr>
                <td>${status.count}</td>
                <td>${product.productName}</td>
                <td>${product.dir_id}</td>
                <td>${product.salePrice}</td>
                <td>${product.supplier}</td>
                <td>${product.brand}</td>
                <td>${product.cutoff}</td>
                <td>${product.costPrice}</td>
                <td>
                    <a href="#">删除</a>
                    <a href="#">编辑</a>
                </td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

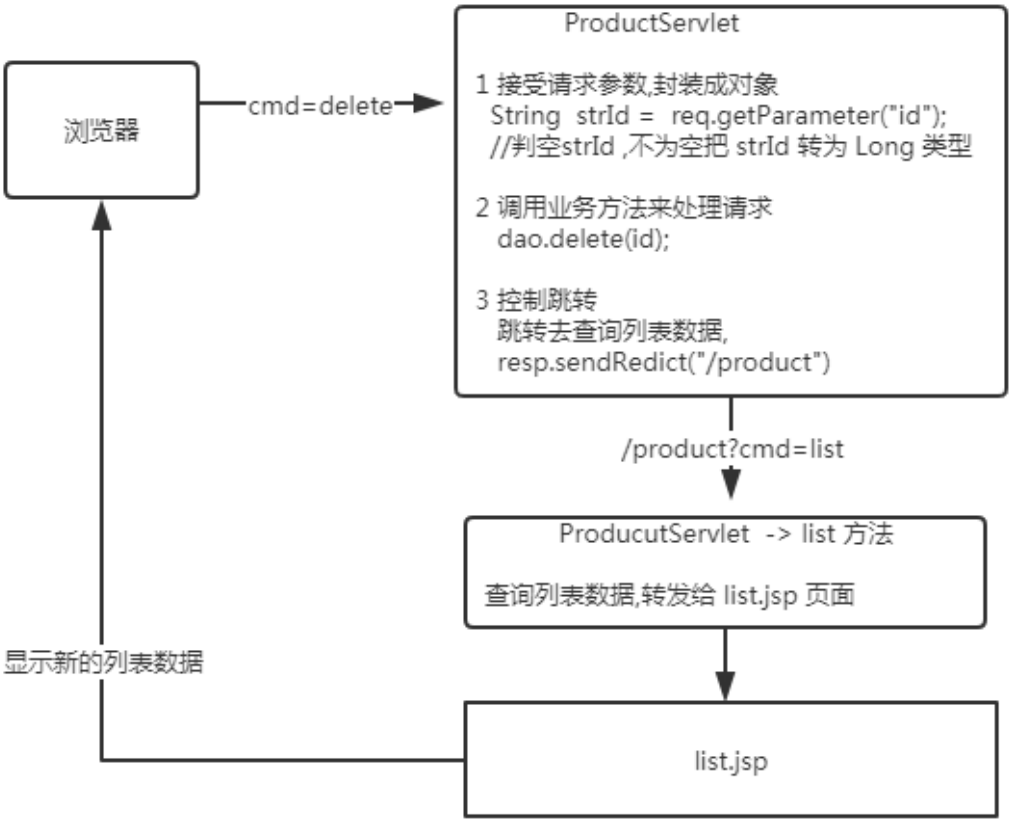
2.3、鼠标悬停效果

给表格 tr 元素（表格头除外）添加一个属性 class="trClassName"，之后在 list.jsp 中增加如下代码来实现鼠标悬停效果。

```
<script type="text/javascript">
    window.onload = function () {
    var trClzs = document.getElementsByClassName("trClassName");
    for(var i = 0; i < trClzs.length; i++){
        trClzs[i].onmouseover = function () {
            this.style.backgroundColor = "gray";
        }
        trClzs[i].onmouseout = function () {
            this.style.backgroundColor = "";
        }
    }
}
</script>
```

六、删除产品

1、流程分析



2、代码实现

2.1、修改 list.jsp

```
<a href="/product?cmd=delete&id=${product.id}">删除</a>
```

2.2、修改 ProductServlet 的 delete 方法

```
protected void delete(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    // 1 接受请求参数
    String strId = req.getParameter("id");
    // 判空
    if(StringUtil.hasLength(strId)){
        Long id = Long.valueOf(strId);
        // 2 调用业务方法来处理请求
        productDAO.delete(id);
    }
    // 3 控制跳转
    resp.sendRedirect("/product");
}
```

2.3、编写 StringUtil

```
package cn.wolfcode.util;

public abstract class StringUtil {
    public static boolean hasLength(String val){
        return val != null && !"".equals(val.trim());
    }
}
```

2.4、增加删除前确认

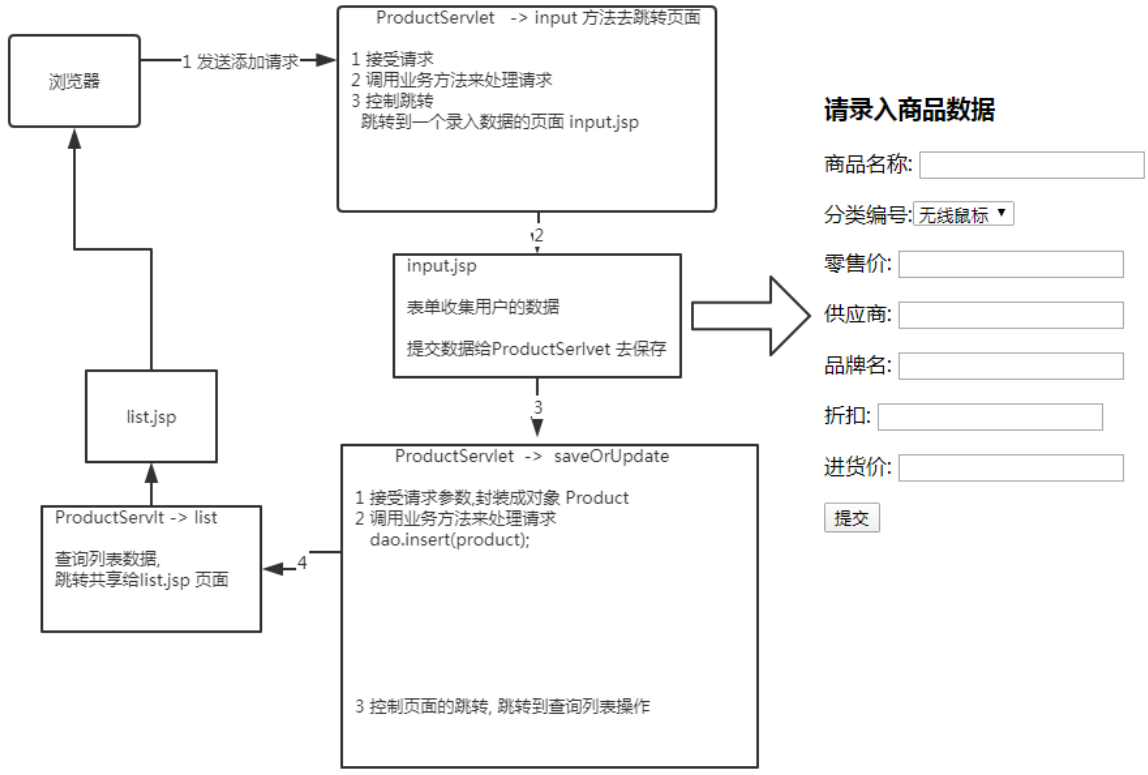
防止误删除。

```
<a href="#" onclick="deleteTr(${product.id})">删除</a>
```

```
function deleteTr(id) {
    // 弹出确认框
    var flag = window.confirm("您确定删除吗？");
    if(flag){
        // 修改浏览器地址栏中地址，并请求
        window.location.href = "/product?cmd=delete&id=" + id;
    }
}
```

七、保存产品

1、流程分析



2、代码实现

2.1、修改 list.jsp

添加一个 a 标签。

```
<a href="/product?cmd=input">添加</a>
```

2.2、修改 ProductServlet 的 input 方法

```
protected void input(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    // 控制跳转
    req.getRequestDispatcher("/WEB-INF/views/product/input.jsp").forward(req,
resp);
}
```

2.3、编写 input.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>产品新增与修改</title>
</head>
<body>
<h3>请录入商品数据</h3>
<form action="/product?cmd=saveOrUpdate" method="post">
    <p>商品名称: <input type="text" name="productName"></p>
    <p>分类编号:<select name="dirId">
        <option value="2">无线鼠标</option>
        <option value="3">有线鼠标</option>
        <option value="4">游戏鼠标</option>
    </select>
    </p>
    <p>零售价: <input type="number" name="salePrice"></p>
    <p>供应商: <input type="text" name="supplier"></p>
    <p>品牌名: <input type="text" name="brand"></p>
    <p>折扣: <input type="number" name="cutoff" step="0.1"></p>
    <p>进货价: <input type="number" name="costPrice"></p>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

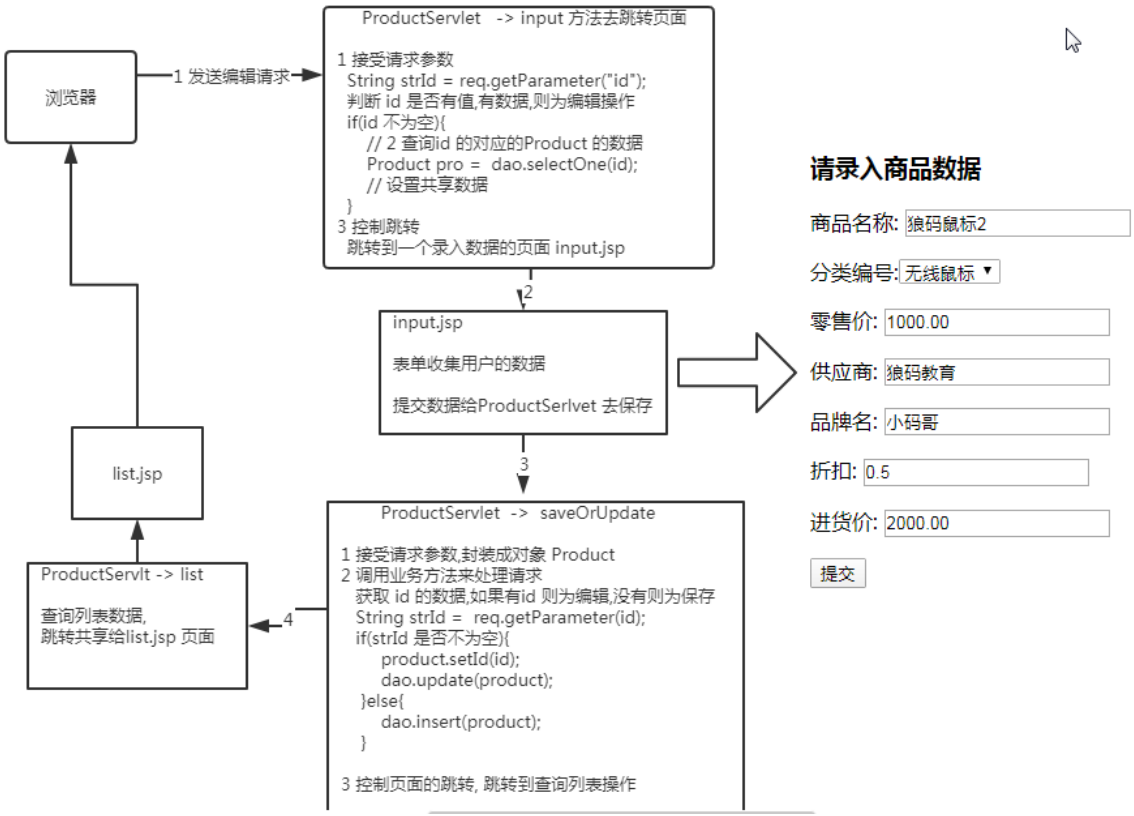
2.4、修改 ProductServlet 的 saveOrUpdate 方法

```
// 处理新增或修改
protected void saveOrUpdate(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    // 1 接受请求参数，封装成对象
    Product product = new Product();
    // 获取请求中的参数，封装到 product 对象中
    req2roduct(req, product);

    System.out.println(product);
```

```
        // 2 调用业务方法来处理请求
        productDAO.save(product);
        // 3 控制页面跳转
        resp.sendRedirect("/product");
    }

    // 获取请求中的参数，封装到 product 对象中
    private void req2roduct(HttpServletRequest req, Product product) {
        String productName = req.getParameter("productName");
        product.setProductName(productName);

        String strDirId = req.getParameter("dirId");
        if(StringUtil.hasLength(strDirId)){
            product.setDir_id(Long.valueOf(strDirId));
        }
        String strSalePrice = req.getParameter("salePrice");
        if(StringUtil.hasLength(strSalePrice)){
            product.setSalePrice(new BigDecimal(strSalePrice));
        }

        String supplier = req.getParameter("supplier");
        product.setSupplier(supplier);
        String brand = req.getParameter("brand");
        product.setBrand(brand);

        String strCutoff = req.getParameter("cutoff");
        if(StringUtil.hasLength(strCutoff)){
            product.setCutoff(new BigDecimal(strCutoff));
        }

        String strCostPrice = req.getParameter("costPrice");
        if(StringUtil.hasLength(strCostPrice)){
            product.setCostPrice(new BigDecimal(strCostPrice));
        }
    }
```

八、修改产品

1、流程分析



2、代码实现

2.1、修改 list.jsp

新增一个 a 标签。

```
<a href="/product?cmd=input&id=${product.id}">修改</a>
```

2.2、修改 ProductServlet 的 input 方法

```
protected void input(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    String strId = req.getParameter("id");
    // 判断 id 是否有值，有就查询被修改的的产品回显
    if(StringUtil.hasLength(strId)){
        // 根据 id 查询产品数据
        Product product = productDAO.get(Long.valueOf(strId));
        // 共享数据给 input.jsp 做回显
        req.setAttribute("product", product);
    }
    // 控制跳转
    req.getRequestDispatcher("/WEB-INF/views/product/input.jsp").forward(req,
resp);
}
```

2.3、修改 input.jsp

使用 EL 表达式回显数据。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
```

```
<head>
    <title>产品新增与修改</title>
</head>
<body>
<h3>请录入商品数据</h3>
<form action="/product?cmd=saveOrUpdate" method="post">
    <input type="hidden" name="id" value="${product.id}">
    <p>商品名称: <input type="text" name="productName"
value="${product.productName}"></p>
    <p>分类编号:
        <select name="dirId">
            <option value="2" ${product.dir_id == 2 ? 'selected' : ''}>无线鼠标
</option>
            <option value="3" ${product.dir_id == 3 ? 'selected' : ''}>有线鼠标
</option>
            <option value="4" ${product.dir_id == 4 ? 'selected' : ''}>游戏鼠标
</option>
        </select>
    </p>
    <p>零售价: <input type="number" name="salePrice"
value="${product.salePrice}"></p>
    <p>供应商: <input type="text" name="supplier" value="${product.supplier}">
</p>
    <p>品牌名: <input type="text" name="brand" value="${product.brand}"></p>
    <p>折扣: <input type="number" name="cutoff" step="0.1"
value="${product.cutoff}"></p>
    <p>进货价: <input type="number" name="costPrice"
value="${product.costPrice}"></p>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

2.4、修改 ProductServlet 的 saveOrUpdate 方法

```
protected void saveOrUpdate(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    Product product = new Product();
    // 1 获取请求中的参数，封装到 product对象中
    req2roduct(req,product);
    // 获取 id 参数值，若有值，修改，若无值，新增
    String strId = req.getParameter("id");
    if(StringUtil.hasLength(strId)){
        // 给 product 对象设置 id 值
        product.setId(Long.valueOf(strId));
        // 2 调用业务方法来处理修改请求
        productDAO.update(product);
    }else {
        // 2 调用业务方法来处理新增请求
        productDAO.save(product);
    }
    // 3 控制页面跳转
    resp.sendRedirect("/product");
}
```

九、如何找 BUG

1、常见找 BUG 操作

- 看错误信息描述，定位错误位置。
- 若没有错误信息，只是数据不对。
  - 打开浏览器 F12 查看请求数据是否正确；
  - 在 servlet 的 service方法中打断点，查看数据的变化，判断是封装问题还是业务方法问题。
- 定位到错误思考原因，解决问题。
- 收集异常，写明原因和解决方案。

2、常见的错误

- NumberFormatException：没有给字符判断是否为空，直接转为数字类型。
- PropertyNotFoundException：某类没有对应的属性。
- 做编辑操作变成了保存操作：
  - input.jsp 没有 id 值没有回显，或请求没有携带 id 参数；
  - 要么 saveOrUpdate 方法没有根据 id 来做业务处理；
  - 字符串判空有问题。

注意事项: 以后不需要数据共享的跳转统统使用重定向即可。

十、MVC 思想

软件（Web应用）开发的模式：Model1（模型一），Model2（模型二），MVC。

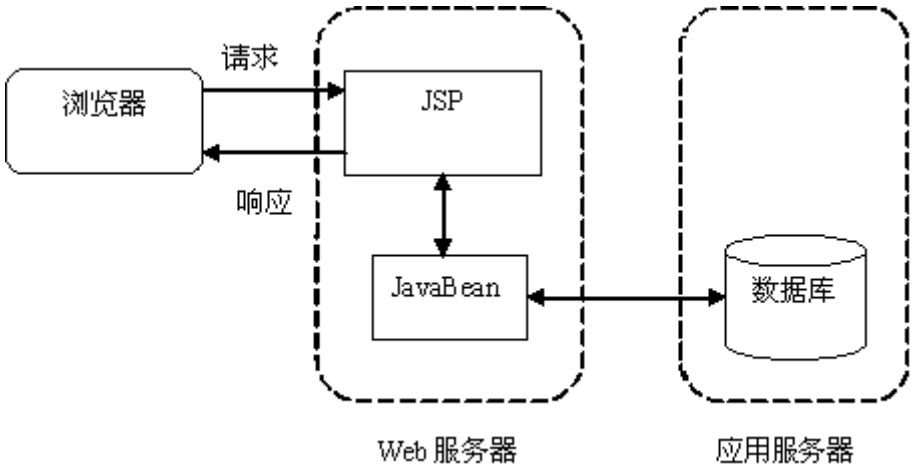
1、Model1 模型一

1.1、技术实现

JSP + JavaBean，以 JSP 为中心，JSP 的职责包含:

- 界面输出（页面渲染）；
- 接受请求参数；
- 调用业务方法，处理请求；
- 控制界面跳转。

1.2、模型图



1.3、优劣势

- 优势：适用于简单的功能,快速开发。
- 劣势：没有体现出责任分离原则。

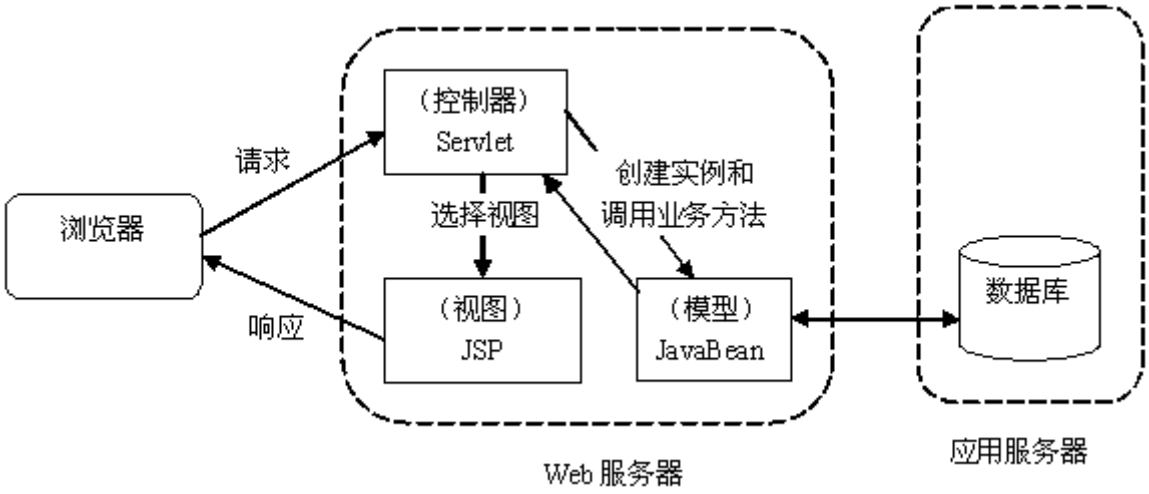
2、Model2 模型二

2.1、技术实现

JSP + Servlet + JavaBean，以 Servlet 为中心（所有请求都发送给 Servlet）。

- JSP：界面输出（页面渲染）。
- Servlet：
  接受请求参数，封装成对象；
  - 调用业务方法，处理请求；
  - 控制跳转。
- JavaBean：体现封装，封装数据，封装业务操作 API，可重复使用。

2.2、模型图



2.3、优劣势

- 优势：体现出责任分离原则，提高代码可读性和维护度。
- 劣势：实现相对 Model1 复杂一点。

3.3、MVC 设计思想（面试题）

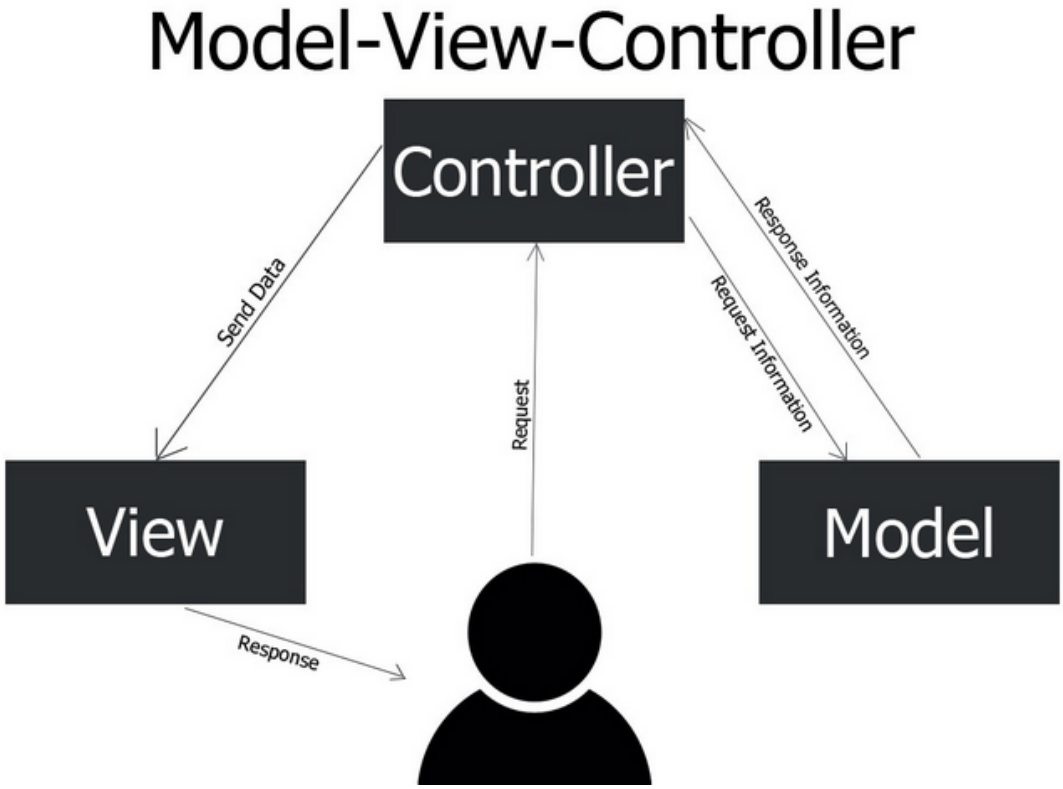
3.1、目的

责任分离，把业务代码从视图中剥离出来，早期运用于 CS 领域（桌面程序）。

3.2、组成

M : Model，模型对象（封装业务操作，算法，可重复使用，JavaBean)，如 DAO，Domain。
V : View，视图（界面），如 JSP，HTML。
C : Controller，控制器（接受请求，控制跳转），如 Servlet。

3.3、模型图



3.4、结论

- Model2 就是一个小型的 MVC 架构;
- 目前咱们就是使用 MVC 架构 JSP + Servlet + Model（Domain）；
- 跟着老师走，潜移默化中会学到很多好的设计思想，代码规范。

练习

```
CREATE TABLE `product` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `productName` varchar(255) DEFAULT NULL,
  `dir_id` bigint(20) DEFAULT NULL,
  `salePrice` decimal(10,2) DEFAULT NULL,
  `supplier` varchar(255) DEFAULT NULL,
  `brand` varchar(255) DEFAULT NULL,
  `cutoff` decimal(10,2) DEFAULT NULL,
  `costPrice` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

根据上面表结构，完成这个产品 表 Web 版本的 CRUD，且要求遵循 MVC 思想，编写处理各种请求代码之前须画出请求响应的执行流程图。