ssm练习第二天

第一章：ssm框架整合

第一节：数据库与表结构

产品表信息描述

序号	字段名称	字段类型	字段描述
1	id	number(9)	无意义，主键
2	productNum	varchar(50)	产品编号
3	productName	varchar(50)	产品名称（路线名称）
4	cityName	varchar(50)	出发城市
5	DepartureTime	timestamp	出发时间
6	productPrice	nubmer(8,2)	产品价格
7	productDesc	varchar(500)	产品描述
8	productStatus	number(2)	状态(0 关闭 1 开启)

创建表sql

```
创建表空间语句
create tablespace ssm datafile 'c:\ssm.dbf' size 100m autoextend on next 10m;
创建用户授予权限
create user ssm identified by ssm default tablespace ssm;
grant dba to ssm;
创建序列语句
create sequence common_sequence;
创建表的sql语句
CREATE TABLE product(
  id number(9) PRIMARY KEY ,
  productNum VARCHAR2(50) ,
  productName VARCHAR2(50),
  cityName VARCHAR2(50),
  DepartureTime TIMESTAMP(0),
  productPrice NUMBER(8,2),
  productDesc VARCHAR2(500),
  productStatus number(2)
)
插入数据语句
insert into PRODUCT

values (common_sequence.nextval, 'itcast-002', '北京三日游', '北京', to_timestamp('10-
```

```
10-2018 10:10:00.000000', 'dd-mm-yyyy hh24:mi:ss.ff'), 1200, '不错的旅行', 1);
insert into PRODUCT
values (common_sequence.nextval, 'itcast-003', '上海五日游', '上海', to_timestamp('25-
04-2018 14:30:00.000000', 'dd-mm-yyyy hh24:mi:ss.ff'), 1800, '魔都我来了', 0);
insert into PRODUCT
values (common_sequence.nextval, 'itcast-001', '北京三日游', '北京', to_timestamp('10-
10-2018 10:10:00.000000', 'dd-mm-yyyy hh24:mi:ss.ff'), 1200, '不错的旅行', 1);
```

第二节：创建maven工程

1. 创建maven的工程
    1. 创建ssm_parent父工程（打包方式选择pom，必须的）
    2. 创建ssm_web子模块（打包方式是war包）
    3. 创建ssm_service子模块（打包方式是jar包）
    4. 创建ssm_dao子模块（打包方式是jar包）
    5. 创建ssm_domain子模块（打包方式是jar包）
    6. 创建ssm_utils子模块（打包方式是jar包）
    7. web依赖于service，service依赖于dao，dao依赖于domain，domain依赖utils
    8. 在ssm_parent的pom.xml文件中引入坐标依赖

```
<properties>
        <spring.version>5.0.2.RELEASE</spring.version>
        <slf4j.version>1.6.6</slf4j.version>
        <log4j.version>1.2.12</log4j.version>
        <mybatis.version>3.4.5</mybatis.version>
        <spring.security.version>5.0.1.RELEASE</spring.security.version>
    </properties>

    <dependencies>
        <!-- spring -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.6.8</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>ojdbc</groupId>
            <artifactId>ojdbc</artifactId>
            <version>14</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>

            <artifactId>javax.servlet-api</artifactId>
```

```
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
        <!-- log start -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>${log4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <!-- log end -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
           <groupId>com.alibaba</groupId>
           <artifactId>druid</artifactId>
           <version>1.0.9</version>
       </dependency>
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper</artifactId>
            <version>5.1.2</version>
        </dependency>


        <dependency>
```

```
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                        <encoding>UTF-8</encoding>
                        <showWarnings>true</showWarnings>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
```

9. 在ssm_web项目中导入静态页面
10. 部署ssm_web的项目，只要把ssm_web项目加入到tomcat服务器中即可

第三节：编写domain、dao和service

1. domain

```
public class Product {
    private Long id;
    private String productNum;
    private String productName;

    private String cityName;
```

```
    private Date departureTime;
    private String departureTimeStr;
    private double productPrice;
    private String productDesc;
    private Integer productStatus;
    private String productStatusStr;

   //省略getter/setter方法
}
```

2. dao

```
public interface ProductDao {

    List<Product> findAll() throws Exception;
    void save(Product product) throws Exception;
}
```

3. service

```
public interface ProductService {

    List<Product> findAll() throws Exception;

    void save(Product product) throws Exception;
}
```

第四节：编写Spring的配置文件

1. 在ssm_web项目中创建applicationContext.xml的配置文件，编写具体的配置信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">



    <!-- 开启注解扫描，要扫描的是service和dao层的注解，要忽略web层注解，因为web层让SpringMVC框架
```

```
去管理 -->
    <context:component-scan base-package="cn.itcast">
        <!-- 配置要忽略的注解 -->
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

</beans>
```

第五节：编写SpringMVC框架的配置文件

1. 在web.xml中配置DispatcherServlet前端控制器

```
<!-- 配置前端控制器：服务器启动必须加载，需要加载springmvc.xml配置文件 -->
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 配置初始化参数，创建完DispatcherServlet对象，加载springmvc.xml配置文件 -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <!-- 服务器启动的时候，让DispatcherServlet对象创建 -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

2. 在web.xml中配置DispatcherServlet过滤器解决中文乱码

```
<!-- 配置解决中文乱码的过滤器 -->
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>characterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

3. 创建springmvc.xml的配置文件，编写配置文件

```
   <?xml version="1.0" encoding="UTF-8"?>
   <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="

       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 扫描controller的注解，别的不扫描 -->
    <context:component-scan base-package="cn.itcast.controller">
    </context:component-scan>

    <!-- 配置视图解析器 -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- JSP文件所在的目录 -->
        <property name="prefix" value="/pages/" />
        <!-- 文件的后缀名 -->
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- 设置静态资源不过滤 -->
    <mvc:resources location="/css/" mapping="/css/**" />
    <mvc:resources location="/img/" mapping="/img/**" />
    <mvc:resources location="/js/" mapping="/js/**" />
    <mvc:resources location="/plugins/" mapping="/plugins/**" />

    <!-- 开启对SpringMVC注解的支持 -->
    <mvc:annotation-driven />

   </beans>
```

第六节：Spring整合SpringMVC的框架

1. 在项目启动的时候，就去加载applicationContext.xml的配置文件，在web.xml中配置ContextLoaderListener监听器（该监听器只能加载WEB-INF目录下的applicationContext.xml的配置文件）。

```
<!-- 配置Spring的监听器 -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <!-- 配置加载类路径的配置文件 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
```

2. 在controller中注入service对象，调用service对象的方法进行测试

第七节：Spring整合MyBatis框架

1. 目的：把SqlMapConfig.xml配置文件中的内容配置到applicationContext.xml配置文件中

```
    <!-- 配置C3P0的连接池对象 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="oracle.jdbc.driver.OracleDriver"></property>
        <property name="jdbcUrl" value="jdbc:mysql:///ssm" />
        <property name="user" value="root" />
        <property name="password" value="root" />
    </bean>

    <!-- 配置SqlSession的工厂 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置扫描dao的包 -->
    <bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.itcast.dao"/>
    </bean>
```

2. 在AccountDao接口中添加@Repository注解
3. 在service中注入dao对象，进行测试

第八节：配置Spring声明式事务管理

```
<!-- 配置事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

        <property name="dataSource" ref="dataSource"/>

    </bean>
 <!--开启事务的注解驱动-->
   <tx:annotation-driven></tx:annotation-driven>
```

第二章：产品模块功能

第一节：查询所有的产品

```
@Controller
@RequestMapping("/product")
public class ProductController {

@Autowired
private ProductService productService;

@RequestMapping("/findAllProduct")
    public String findAllProduct(Model model){
        model.addAttribute("productList",productService.findAllProduct());
        return "product/productList";
    }
}

@Service
public class ProductServiceImpl implements ProductService {



@Autowired
private ProductDao productDao;

public List<Product> findAll() {
     return productDao.findAll();
}


}

@Repository
public interface ProductDao {
```

```
@Select("select * from product")
public List<Product> findAll();


}
```

```
<!--数据列表-->
<table id="dataList"
class="table table-bordered table-striped table-hover dataTable">
<thead>
<tr>
<th class="" style="padding-right: 0px;"><input
id="selall" type="checkbox" class="icheckbox_square-blue">
</th>

<th class="sorting">产品编号</th>
<th class="sorting">产品名称</th>
<th class="sorting">出发城市</th>
<th class="sorting">出发日期</th>
<th class="sorting">价格</th>
<th class="sorting">描述</th>

<th class="text-center">操作</th>
</tr>
</thead>
<tbody>

<c:forEach items="${ plist }" var="p">
<tr>
<td><input name="ids" type="checkbox"></td>

<td>${ p.productNum }</td>
<td>${ p.productName }</td>
<td>${ p.cityName }</td>
<td>${ p.departureTime }</td>
<td>${ p.productPrice }</td>
<td>${ p.productDesc }</td>

<td class="text-center">
<button type="button" class="btn bg-olive btn-xs"
onclick='location.href="all-order-manage-edit.html"'>订单</button>
<button type="button" class="btn bg-olive btn-xs"
onclick='location.href="all-order-manage-edit.html"'>查看</button>
</td>
</tr>
</c:forEach>

</tbody>

</table>
```

第二节：解决日期和状态字段显示的问题

1. JSP页面显示的日期为默认的英文日期，转换成字符串格式有2种方式
    1. 使用fmt标签进行转换
        1. <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
        2. undefined在JavaBean中添加属性，重写该属性的get方法进行转换

```
private String departureTimeStr;
/**
 * 重新get方法，返回字符串类型的时间
 * @return
 */
public String getDepartureTimeStr() {
    if(departureTime == null) {
        return "";
    }else {
        return DateUtils.dateToStr(departureTime, "yyyy-MM-dd HH:mm:ss");
    }
}

public class DateUtils {

    /**
     * 把日期转换成字符串
     * @param date
     * @return
     */
    public static String dateToStr(Date date,String pattern) {
        SimpleDateFormat sdf = new SimpleDateFormat(pattern);
        return sdf.format(date);
    }

}
```

2. 处理状态的问题
    1. 在JSP页面进行判断

```
<td>
    <c:if test="${ p.productStatus == 0 }">关闭</c:if>
    <c:if test="${ p.productStatus == 1 }">开启</c:if>
</td>
```

    2. 在JavaBean中添加属性，重写get方法

```
private String productStatusStr;
public String getProductStatusStr() {
    return productStatus == 0?"关闭":"开启";
}
```

第三节：保存产品

1. 跳转到新增的页面

```
    /**
     * 跳转到新增页面
     * @return
     */
    @RequestMapping("/initAdd")
    public String initAdd() {
        return "product-add";
    }
```

2. 新增产品

```
    /**
     * 保存
     * @return
     */
    @RequestMapping("/save")
    public String save(Product product) {
        productService.save(product);
        return "redirect:/product/findAll";
    }

    @Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private ProductDao productDao;

    public List<Product> findAll() {
        return productDao.findAll();
    }

    public void save(Product product) {
        productDao.save(product);
    }

}

@Repository
public interface ProductDao {


    @Select("select * from product")
```

```
    public List<Product> findAll();

    @Insert("insert into product
(productNum,productName,cityName,departureTime,productPrice,productDesc,productStatus)
values (#{productNum},#{productName},#{cityName},#{departureTime},#{productPrice},#
{productDesc},#{productStatus})")
    public void save(Product product);

}
```

3. 在进行数据绑定的时候，日期出现了异常，该格式的日期不支持默认数据类型转换
    1. 自定义类型转换器（比较麻烦，回去看第一天文档）
    2. 在departureTime属性上添加注解解决

    ```
    @DateTimeFormat(pattern="yyyy-MM-dd HH:mm")
    private Date departureTime;
    ```

    3. 在Controller类中添加方法，进行类型转换

    ```
    /**
     * 类型转换
     * @param dataBinder
     */
    @InitBinder
    public void initBinderDate(WebDataBinder dataBinder) {
        dataBinder.registerCustomEditor(Date.class, new PropertiesEditor() {
            // JSP页面传过来的数据
            public void setAsText(String text) throws IllegalArgumentException {
                // 把字符串转换成日期
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
                try {
                    Date date = sdf.parse(text);
                    // 设置值
                    super.setValue(date);
                } catch (ParseException e) {
                    e.printStackTrace();
                }
            }
        });
    }
    ```

第四节：修改产品

1. 跳转到修改页面

```
    /**
     * 跳转到修改页面
     * @return

     */
```

```
@RequestMapping("/initUpdate")
public ModelAndView initUpdate(Long id) {
    ModelAndView mv = new ModelAndView();
    // 查询
    Product product = productService.findById(id);
    mv.addObject("product", product);
    mv.setViewName("product-update");
    return mv;
}

public Product findById(Long id) {
    return productDao.findById(id);
}

@Select("select * from product where id = #{id}")
public Product findById(Long id);
```

2. 修改代码

```
    /**
     * 修改
     * @param product
     * @return
     */
    @RequestMapping("/update")
    public String update(Product product) {
        productService.update(product);
        return "redirect:/product/findAll";
    }

    public void update(Product product) {
        productDao.update(product);
    }

    @Update("update product set productNum = #{productNum},productName=#
{productName},cityName=#{cityName},departureTime=#{departureTime},productPrice=#
{productPrice},productDesc=#{productDesc},productStatus=#{productStatus} where id = #{id}")
    public void update(Product product);
```

第五节：删除产品

1. 代码

```
<button type="button" class="btn bg-olive btn-xs" onclick='del(${p.id})'>删除</button>

        // 删除
        function del(id){
            if(confirm("确定删除吗？")){
                // 发送请求
                location.href = "${pageContext.request.contextPath}/product/delete?
```

```
id="+id;
            }
        }

    /**
     * 删除
     * @param id
     * @return
     */
    @RequestMapping("/delete")
    public String delete(Long id) {
        productService.delete(id);
        return "redirect:/product/findAll";
    }

    public void delete(Long id) {
        productDao.delete(id);
    }

    @Delete("delete from product where id = #{id}")
    public void delete(Long id);
```

第三章：订单模块功能

第一节：订单表与产品表的关系

1. 一个用户会产品一个订单，一个订单只能选择一个产品，因为是旅游产品。
2. 一个产品可以被多个订单所选择。
3. 订单表与产品表的关系是多对一
4. 订单表的SQL语句
5. 订单表信息描述 orders

序号	字段名称	字段类型	字段描述
1	id	number	主键
2	orderNum	varchar2(50)	订单编号 不为空 唯一
3	orderTime	timestamp	下单时间
4	peopleCount	number	出行人数
5	orderDesc	varchar2(500)	订单描述(其它信息)
6	payType	number	支付方式(0 支付宝 1 微信 2其它)
7	orderStatus	number	订单状态(0 未支付 1 已支付)
8	productId	number	产品id 外键

productId描述了订单与产品之间的关系。

创建表sql

```
CREATE TABLE orders(
    id NUMBER(9) PRIMARY KEY ,
    orderNum VARCHAR2(20) NOT NULL UNIQUE,
    orderTime TIMESTAMP(0),
    peopleCount NUMBER,
    orderDesc VARCHAR2(500),
    payType NUMBER(2),
    orderStatus  NUMBER(2),
    productId  NUMBER(9),
    FOREIGN KEY (productId) REFERENCES product(id)
)
```

第二节：搭建订单模块的环境

1. 编写Order的JavaBean

```
public class Order implements Serializable{

    private static final long serialVersionUID = 8153486211933088983L;

    private Long id;
    private String orderNum;
    private Date orderTime;
    private Integer peopleCount;
    private String orderDesc;
    private Integer payType;
    private Integer orderStatus;
    // 外键
```

```
    // private Long productId;
    private Product product;

    // get和set
}
```

2. 编写OrderDao

```
package cn.itcast.dao;

import org.springframework.stereotype.Repository;

@Repository
public interface OrderDao {

}
```

3. 编写OrderService

```
package cn.itcast.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import cn.itcast.dao.OrderDao;
import cn.itcast.service.OrderService;

@Service
public class OrderServiceImpl implements OrderService {

    @Autowired
    private OrderDao orderDao;

}
```

4. 编写OrderController

```
package cn.itcast.web;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import cn.itcast.service.OrderService;
```

```
@Controller
@RequestMapping("/order")
public class OrderController {

    @Autowired
    private OrderService orderService;

}
```

第三节：查询所有的订单

1. 具体的代码如下

```
@Controller
@RequestMapping("/order")
public class OrderController {

    @Autowired
    private OrderService orderService;

    /**
     * 查询所有
     * @return
     */
     @RequestMapping("/findAllOrder")
    public String findAllOrder(Model model){

        model.addAttribute("orderList",orderService.findAllOrder());

     return "order/orderList";
    }

}

@Service
public class OrderServiceImpl implements OrderService {

    @Autowired
    private OrderDao orderDao;

    public List<Order> findAll() {
        return orderDao.findAll();
    }

}
```

@Repository public interface OrderDao { /**

```
*  查询所有
    * @return
       */
       @Select("select * from orders")
       @Results(
               @Result(column = "productid" ,property = "product",
                       one = @One(select = "cn.itcast.dao.ProductDao.findProductById"))
       )
       List<Order> findAll();
```

}

${ o.orderNum } ${ o.peopleCount } ${ o.orderDesc } 支付宝/c:if 微信/c:if 其他/c:if 未支付/c:if 已支付/c:if ${ o.product.productName } ${ o.product.productPrice }

<button type="button" class="btn bg-olive btn-xs" onclick='location.href="${pageContext.request.contextPath}/pages/order-show.jsp"'>订单 <button type="button" class="btn bg-olive btn-xs" onclick='location.href="${pageContext.request.contextPath}/pages/order-show.jsp"'>查看 /c:forEach