目录

1 第七天：订单管理

1.1 查询一个订单

1.1.1 改造OrderMapper.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">

<!-- 订单的映射文件
namespace：接口的全路径
```

```
-->
<mapper namespace="cn.tedu.dao.OrderMapper">

    <!-- 映射不规则字段
        type：要把数据封装给哪个对象
        id：该resultMap的唯一标志
    -->
    <resultMap type="Order"
                id="OrderRM">

<!-- id属性 column是字段名  property是属性名-->
        <id column="id" property="id"/>

<!-- 其他属性 column是字段名  property是属性名-->
        <result column="door_id" property="doorId"/>
        <result column="order_no" property="orderNo"/>
        <result column="order_type" property="orderType"/>
        <result column="person_num" property="personNum"/>
        <result column="cashier" property="cashier"/>
        <result column="create_time" property="createTime"/>
        <result column="end_time" property="endTime"/>
        <result column="payment_type" property="paymentType"/>
        <result column="price" property="price"/>
        <result column="created" property="created"/>
        <result column="updated" property="updated"/>
    </resultMap>

    <!-- 查询所有订单
        resultMap：引用一个id值
    -->
    <select id="SelectAll"
            resultMap="OrderRM">
        select * from tb_order
    </select>

    <!-- 查询一个订单
        resultMap：引用哪个
    -->
    <select id="SelectOne"
            resultMap="OrderRM">
        select * from tb_order
        where id=#{id}
    </select>


</mapper>
```

1.1.2 改造OrderMapper接口

```
package cn.tedu.dao;

import java.util.List;
```

```
import cn.tedu.pojo.Order;

public interface OrderMapper {

    //查询所有订单
    //<select id="SelectAll"
    //resultType="cn.tedu.pojo.Order">
    public List<Order> SelectAll();

    //<!-- 查询一个订单
    //<select id="SelectOne"
    //resultMap="OrderRM">
    public Order SelectOne(Integer id);



}
```

1.1.3 改造OrderService接口

```
package cn.tedu.dao;

import java.util.List;

import cn.tedu.pojo.Order;

public interface OrderMapper {

    //查询所有订单
    //<select id="SelectAll"
    //resultType="cn.tedu.pojo.Order">
    public List<Order> SelectAll();

    //<!-- 查询一个订单
    //<select id="SelectOne"
    //resultMap="OrderRM">
    public Order SelectOne(Integer id);



}
```

1.1.4 改造OrderServiceImpl

```
    package cn.tedu.service;

import java.util.List;

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

import cn.tedu.dao.OrderMapper;
```

```
import cn.tedu.pojo.Order;

//作用1：交给spring管理对象
//作用2：代表是业务层代码
@Service
public class OrderServiceImpl implements OrderService {


    @Autowired//自动注入dao层
    private OrderMapper orderMapper;


    //查询所有订单
    @Override
    public List<Order> SelectAll() {
        //调用dao层干活
        return orderMapper.SelectAll();
    }

    //<!-- 查询一个订单
    @Override
    public Order SelectOne(Integer id) {
        return orderMapper.SelectOne(id);
    }

}
```

1.1.5 改造OrderController

```
package cn.tedu.controller;

import java.util.List;

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

import cn.tedu.pojo.Order;
import cn.tedu.service.OrderService;

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

    @Autowired //自动注入service层
    private OrderService orderService;

    //查询所有订单
    @RequestMapping("list")
    public String list(Model model){
        //查询所有数据
```

```
        List<Order> list =
                orderService.SelectAll();

        //给页面准备数据
        model.addAttribute("orderList",list);

        //跳转页面
        return "order";

    }

    //查询一个订单
    @RequestMapping("findOne")
    @ResponseBody//返回json串
    public Order findOne(Integer id){
        return orderService.SelectOne(id);
    }


}
```

1.1.6 测试

1.2 新增订单

1、 点击列表页面的【新增】，跳转到新增页面。
2、 新增页面需要准备数据（所有门店）
3、

1.2.1 OrderController添加跳转

在订单列表页面，点击【新增】，转到新增页面，需要准备门店数据的下拉列表

```
@RequestMapping("toAdd")
```

```
public String toAdd(Model model){
   //1、准备数据（所有门店）
   List<Door> list =
          doorService.SelectAll();
   model.addAttribute("doorList", list);

   //2、跳转页面
   return "orderAdd";
}
```

1.2.2 orderAdd.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
  pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>永和大王门店管理系统</title>
    </head>
    <body style="padding:20px;">
        <div style="padding: 5px;">
            <h1>永和大王门店管理系统-订单添加</h1>
        </div>
        <form action="add" method="post">
        <div>
            <table border="1" cellspacing="0">
                <tr>
                    <td>所属门店：</td>
                    <td>
                        <select name="doorId">
                            <option value="">--请选择--</option>
                            <c:forEach items="${doorList}" var="door">
                                <option value="${door.id}">${door.name}</option>
                            </c:forEach>
                        </select>
                    </td>
                    <td> </td>
                    <td> </td>
                </tr>
                <tr>
                    <td>订单号：</td>
                    <td><input type="text" name="orderNo" value=""/></td>
                    <td>订单类型：</td>
                    <td><input type="text" name="orderType" value=""/></td>
                </tr>
                <tr>
                    <td>人数：</td>
                    <td><input type="text" name="personNum" value=""/></td>
                    <td>收银员：</td>
                    <td><input type="text" name="cashier" value=""/></td>
                </tr>
                <tr>
                    <td>开单时间：</td>
```

```
                    <td><input type="text" name="createTime" value=""/></td>
                    <td>结账时间：</td>
                    <td><input type="text" name="endTime" value=""/></td>
                </tr>
                <tr>
                    <td>支付方式：</td>
                    <td><input type="text" name="paymentType" value=""/></td>
                    <td>支付金额：</td>
                    <td><input type="text" name="price" value=""/></td>
                </tr>
            </table>

            <br/>
              
            <input type="submit" name="ok" value="提交"/>
            <input type="button" onclick="history.go(-1);" name="back" value="返回"/>
        </div>
    </form>
    </body>
</html>
```

1.2.3 修改OrderMapper.xml

```
<!-- 新增订单 -->
    <insert id="add">
        insert into tb_order values(
            #{id},
            #{doorId},
            #{orderNo},#{orderType},
            #{personNum},#{cashier},
            #{createTime},#{endTime},
            #{paymentType},#{price},
            #{created},#{updated}
        )
    </insert>
```

1.2.4 修改OrderMapper接口

```
//<!-- 新增订单 -->
    //<insert id="add">
    public void add(Order order);
```

1.2.5 修改OrderService接口

```
//<!-- 新增订单 -->
    //<insert id="add">
    public void add(Order order);
```

1.2.6 修改OrderServiceImpl

```
//<!-- 新增订单 -->
```

```
@Override
public void add(Order order) {
    orderMapper.add(order);
}
```

1.2.7 OrderController中添加新增方法

当在订单新增页面，点击【提交】，会访问controller插入数据库，并跳转到列表方法，列出新数据。

```
//<!-- 新增订单 -->
//目的：执行SQL
@RequestMapping("add")
public String add(Order order){
    //执行SQL
    orderService.add(order);

    //刷新列表(访问列表方法)
    return "redirect:list";
}
```

1.2.8 注意：日期字段

默认是使用/，但是可以在日期类型的属性上面加注解来改变格式

@DateTimeFormat(pattern="yyyy-MM-dd")

如：

```
//create_time字段
    @DateTimeFormat(pattern="yyyy-MM-dd")
    private Date createTime;

    //end_time字段
    @DateTimeFormat(pattern="yyyy-MM-dd")
    private Date endTime;
```

1.3 修改订单

1、 点击列表页面的【修改】跳转到修改页面。
2、 给修改页面准备数据（根据id查询数据）
3、 点击修改页面的【提交】，向服务器发起请求执行SQL
4、 刷新列表

1.3.1 OrderController中添加跳转

在订单列表页面，点击【修改】，转到修改页面

```
//1、点击列表页面的【修改】跳转到修改页面。
//2、给修改页面准备数据（根据id查询数据）
@RequestMapping("toUpdate")
public String toUpdate(
                Integer id,
                Model model){
   //查询订单数据给页面准备数据
   Order order =
          orderService.SelectOne(id);
   model.addAttribute("order",order);

   //查询门店数据给页面准备下拉框数据
   List<Door> list =
          doorService.SelectAll();
   model.addAttribute("doorList", list);

   //跳转页面
   return "orderUpdate";

}
```

1.3.2 orderUpdate.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
  pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>永和大王门店管理系统</title>
</head>
<body style="padding:20px;">
<div style="padding: 5px;">
        <h1>永和大王门店管理系统-订单修改</h1>
    </div>
<form action="update" method="post">
    <input type="hidden" name="id" value="${order.id}"/>

    <div>
        <table border="1">
            <tr>
                <td>所属门店：</td>
                <td>
                    <select name="doorId">
                        <option value="">--请选择--</option>
                        <c:forEach items="${doorList}" var="d">
                            <option value="${d.id}"
                                <c:if test="${d.id==order.doorId}">selected</c:if>
```

```
                                >${d.name}</option>
                            </c:forEach>
                        </select>
                    </td>
                    <td> </td>
                    <td> </td>
                </tr>
                <tr>
                    <td>订单号：</td>
                    <td><input type="text" name="orderNo" value="${order.orderNo}"/></td>
                    <td>订单类型：</td>
                    <td><input type="text" name="orderType" value="${order.orderType}"/></td>
                </tr>
                <tr>
                    <td>人数：</td>
                    <td><input type="text" name="personNum" value="${order.personNum}"/>
</td>
                    <td>收银员：</td>
                    <td><input type="text" name="cashier" value="${order.cashier}"/></td>
                </tr>
                <tr>
                    <td>开单时间：</td>
                    <td><input type="text" name="createTime" value="<fmt:formatDate
value="${order.createTime}" pattern="yyyy-MM-dd HH:mm:ss"/>"/></td>
                    <td>结账时间：</td>
                    <td><input type="text" name="endTime" value="<fmt:formatDate
value="${order.endTime}" pattern="yyyy-MM-dd HH:mm:ss"/>"/></td>
                </tr>
                <tr>
                    <td>支付方式：</td>
                    <td><input type="text" name="paymentType" value="${order.paymentType}"/>
</td>
                    <td>支付金额：</td>
                    <td><input type="text" name="price" value="${order.price}"/></td>
                </tr>
            </table>

            <br/>
              
            <input type="submit" name="ok" value="提交"/>
        </div>
    </form>
    </body>
    </html>
```

1.3.3 修改OrderMapper.xml

```
<!-- 更新订单 -->
<update id="update">
   update tb_order set
       door_id=#{doorId},
       order_no=#{orderNo},
       order_type=#{orderType},
       person_num=#{personNum},
       cashier=#{cashier},
       create_time=#{createTime},
```

```
            end_time=#{endTime},
            payment_type=#{paymentType},
            price=#{price},
            created=#{created},
            updated=#{updated}
        where id=#{id}
    </update>
```

1.3.4 修改OrderMapper接口

```
//<!-- 更新订单 -->
    //<update id="update">
    public void update(Order order);
```

1.3.5 修改OrderService接口

```
//<!-- 更新订单 -->
    //<update id="update">
    public void update(Order order);
```

1.3.6 修改OrderServiceImpl

```
    @Override
    public void update(Order order) {
        orderMapper.update(order);
    }
```

1.3.7 OrderController中添加修改方法

1、点击修改页面的【提交】，向服务器发起请求执行SQL
2、刷新列表

```
    //<!-- 更新订单 -->
    //1、点击修改页面的【提交】，向服务器发起请求执行SQL
    //2、刷新列表
    @RequestMapping("update")
    public String abc(Order order){
        //1、执行SQL
        orderService.update(order);

        //2、刷新列表(访问列表方法)
        return "redirect:list";
    }
```

1.4 删除订单

1.4.1 修改OrderMapper.xml

```
<!-- 根据id删除订单 -->
    <delete id="delete">
        delete from tb_order
        where id=#{id}
    </delete>
```

1.4.2 修改OrderMapper接口

```
//根据id删除订单
public void delete(Integer id);
```

1.4.3 修改OrderService接口

```
//根据id删除订单
    public void delete(Integer id);
```

1.4.4 修改OrderServiceImpl

```
//根据id删除订单
@Override
public void delete(Integer id) {
    orderMapper.delete(id);
}
```

1.4.5 OrderController中添加删除方法

```
//根据id删除订单
@RequestMapping("delete")
public String delete(Integer id){
    //执行SQL
    orderService.delete(id);

    //刷新列表(访问列表方法)
    return "redirect:list";
}
```

1.5 拓展

1.5.1 查询关联的门店对象

1.5.1.1 修改Order实体类

```
//订单对象和门店对象  一对一
    private Door door;
    public Door getDoor() {
       return door;
    }

    public void setDoor(Door door) {
       this.door = door;
    }
```

1.5.1.2 修改OrderMapper.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">

<!-- 订单的映射文件
namespace：接口的全路径
-->
<mapper namespace="cn.tedu.dao.OrderMapper">

    <!-- 映射不规则字段
       type：要把数据封装给哪个对象
       id：该resultMap的唯一标志
    -->
    <resultMap type="Order"
               id="OrderRM">

<!-- id属性 column是字段名  property是属性名-->
       <id column="id" property="id"/>

<!-- 其他属性 column是字段名  property是属性名-->
       <result column="door_id" property="doorId"/>
       <result column="order_no" property="orderNo"/>
       <result column="order_type" property="orderType"/>
       <result column="person_num" property="personNum"/>
       <result column="cashier" property="cashier"/>
       <result column="create_time" property="createTime"/>
       <result column="end_time" property="endTime"/>
       <result column="payment_type" property="paymentType"/>
       <result column="price" property="price"/>
       <result column="created" property="created"/>
       <result column="updated" property="updated"/>

       <!-- 描述一对一的门店对象信息
           association  javaType  用来描述对一的一方的数据
           javaType：将要描述哪个关联对象
           property：是主对象中的属性
       -->
       <association javaType="Door" property="door">
           <!-- 描述关联对象 -->
           <id column="id" property="id"/>
           <result column="name" property="name"/>
           <result column="tel" property="tel"/>
           <result column="updated" property="updated"/>
           <result column="created" property="created"/>
```

```
        </association>
    </resultMap>

    <!-- 查询所有订单
        resultMap: 引用一个id值
    -->
    <select id="SelectAll"
            resultMap="OrderRM">
        select * from tb_order
    </select>

    <!-- 查询一个订单
        resultMap: 引用哪个
    -->
    <select id="SelectOne"
            resultMap="OrderRM">
        select * from tb_order
        where id=#{id}
    </select>

    <!-- 新增订单 -->
    <insert id="add">
        insert into tb_order values(
            #{id},
            #{doorId},
            #{orderNo},#{orderType},
            #{personNum},#{cashier},
            #{createTime},#{endTime},
            #{paymentType},#{price},
            #{created},#{updated}
        )
    </insert>

    <!-- 更新订单 -->
    <update id="update">
        update tb_order set
            door_id=#{doorId},
            order_no=#{orderNo},
            order_type=#{orderType},
            person_num=#{personNum},
            cashier=#{cashier},
            create_time=#{createTime},
            end_time=#{endTime},
            payment_type=#{paymentType},
            price=#{price},
            created=#{created},
            updated=#{updated}
        where id=#{id}
    </update>

    <!-- 根据id删除订单 -->
    <delete id="delete">
        delete from tb_order
        where id=#{id}
    </delete>


</mapper>
```

1.5.1.3 测试

Order [id=1, doorId=1, orderNo=P000011, orderType=堂食, personNum=1, cashier=张静, createTime=Thu Apr 26 14:49:07 CST 2018, endTime=Tue Apr 17 07:24:38 CST 2018, paymentType=微支付, price=8.0, door=Door [id=1, name=永和大王（北三环西路店）, tel=110-62112313]]

OrderMapper.xml

```
<!-- 关联查询 -->
    <select id="SelectAll"
            resultMap="OrderRM">
       select * from tb_door t1
       left join tb_order t2
       on t1.id=t2.door_id
    </select>
```

Controller层

遍历list并打印，主要看关联对象信息能不能封装成功

1.5.2 工具：自动生成代码-eclipse插件

jdbcType 当这个值为null时，告诉jdbc驱动程序，这个值当null时应该按什么类型来处理。

Mysql驱动不需要，Oracle驱动底层不能识别，所以必须指定jdbcType，不指定报错。

只有在修改时，才需要指定。

利用这个工具产生insert的全字段的代码；update使用动态SQL语句的；select一般直接从insert的语句中拷贝字段；delete语句直接拷贝

1.5.2.1 安装插件

1、将给定的plugins和feaures目录直接拷贝到D:\java\ide\eclipse\dropins

2、重启eclipse。

1.5.2.2 配置sqlMapGenerator.xml

指定mysql的驱动包的路径（千万别放中文路径下）配置数据源和生成的代码所存放的位置。

创建一个配置文件sqlMapGeneractor.xml，这个文件名称随便起，它会自动识别你文件的头信息是否包含<!DOCTYPE generatorConfiguration>。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN" "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd" >
<generatorConfiguration>
    <!-- 配置属性文件，这样有变更只需改配置文件 -->
    <properties resource="generatorConfig.properties"/>
    <!-- 指定mysql的驱动包的路径 千万别放中文路径下 -->
    <classPathEntry location="d:\mysql-connector-java-5.1.40.jar" />
    <!-- 配置数据源和生成的代码所存放的位置 -->
    <context id="tarena">
        <!-- 是否生成注释 true不生成 false生成 -->
        <commentGenerator>
            <property name="suppressAllComments" value="${suppressAllComments}"/>
        </commentGenerator>
```

```
                                <jdbcConnection driverClass="${driverClass}" connectionURL="${url}"
userId="${username}" password="${password}" />
            <!-- 所生成的实体类的位置默认资源包src -->
                                        <javaModelGenerator targetPackage="${modeltargetPackage}"
targetProject="${targetProject}" />
            <!-- 所生成的sqlMap的影射文件的位置,默认资源包src -->
              <sqlMapGenerator targetPackage="${sqltargetPackage}" targetProject="${targetProject}"
/>
                                        <javaClientGenerator targetPackage="${clienttargetPackage}"
targetProject="${targetProject}" type="XMLMAPPER" />

            <!-- 为哪些表生成代码 tableName:表名 schema:不用填写 -->
            <table schema="" tableName="tb_door" />
            <table schema="" tableName="tb_order" />


        </context>
    </generatorConfiguration>
```

1.5.2.3 配置generatorConfig.properties

```
suppressAllComments=false
driverClass=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/yhmisdb?characterEncoding=utf-8
username=root
password=root

modeltargetPackage=cn.tedu.pojo
targetProject=v
sqltargetPackage=cn.tedu.mapper
clienttargetPackage=cn.tedu.dao
```

1.5.2.4 生成的配置文件引入核心配置文件中

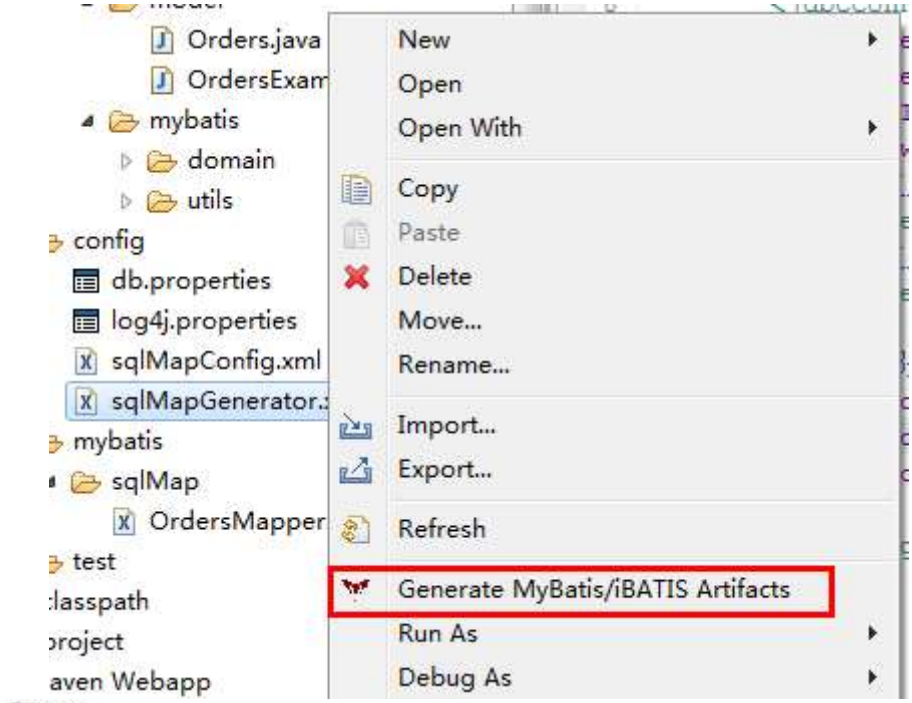
```
<mappers>
        <!-- 引入UserMapper.xml
        resource指定文件的位置，右键。copy qua...name


        -->
        <mapper resource="map/UserInfoMapper.xml"/>
</mappers>
```

1.5.2.5 测试

我们可能会担心一旦重新执行generate的时候，我们自己编写的代码会不会丢失，不会的，插件不会修改或丢弃我们自己编写的代码。

一旦掌握了插件如何使用，重要的工作就是如何使用XXXExample类了。这种方式，完全不用编写繁琐的mapper xml文件。




```
@Test
public void test(){
    SqlSession session = ssf.openSession();

    //调用工具生成的接口方法
    UserInfoMapper dao = session.getMapper(UserInfoMapper.class);
    UserInfo info = dao.selectByPrimaryKey(1);

    System.out.println(info);
    session.close();
}
```