# KiroScarlet / PromoProject

github.com/KiroScarlet/PromoProject

KiroScarlet

## 项目环境：IDEA，maven，MySQL5.x

- 项目运行方式：从IDEA导入项目，更新maven依赖，然后在MySQL数据库中运行miaosha.sql文件生成数据库。

- 项目入口为：com.miaoshaproject.App，使用IDEA启动后，若端口被占用，修改application.properties中的端口配置。

- 项目采用前后端分离，直接在浏览器打开resources目录下的getotp.html即可。

我的博客地址https://blog.csdn.net/m0_37657841/article/details/90524410

# 第一章 课程介绍

**电商秒杀应用简介**

> - 商品列表页获取秒杀商品列表
> - 进入商品详情页获取秒杀商品详情
> - 秒杀开始后进入下单确认页下单并支付成功

# 第二章 应用SpringBoot完成基础项目搭建

## 2.1 使用IDEA创建maven项目

1.new->project->maven项目->选择maven-archetype-quickstart

以jar包方式对外输出

稍等一会，可能会有点慢

2.新建一个resources目录，作为资源文件目录，指定为Resource root

## 2.2 引入SpringBoot依赖包实现简单的Web项目

进入官方文档https://spring.io/guides/gs/rest-service/

**Building a RESTful Web Service**

1.引入父pom

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.4.RELEASE</version>
</parent>
```

### 2.引入依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

### 3.maven Reimport刷新一下，会自动下载相应jar包（注：可以把idea设定为自动导入maven依赖）

### 4.SpringBoot的Web项目

```
@EnableAutoConfiguration//可以启动tomcat，自动加载默认配置，比如mysql
@RestController //配置
public class App
{

    @RequestMapping("/")
    public String home() {
        return "hello World!"; //当用户访问登陆进页面，输出一个helloworld字符串
    }
    public static void main( String[] args )
    {
        System.out.println("Hello World!");
        SpringApplication.run(App.class,args);//启动
    }
}
```

再次启动App，访问localhost:8080

## 2.3 Mybatis接入SpringBoot项目

### 1.SpringBoot的默认配置

在resources目录下新建SpringBoot的默认配置文件application.properties

通过一行简单的属性就能更改tomcat的端口

```
server.port=8090
```

### 2.配置pom文件

```
<!--数据库-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</version>
</dependency>
<!--数据库连接池-->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.1.3</version>
</dependency>
<!--Mybatis依赖-->
<dependency>          //将springboot对mybatis的支持
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>1.3.1</version>
</dependency>
```

### 3.配置文件application.properties，设置 mybatis的一些必需配置

```
mybatis.mapper-locations=classpath:mapping/*.xml
```

### 然后在resources目录下新建mapping目录

### 4.自动生成工具，生成数据库文件的映射

### 引入插件

```
<!--自动生成工具，生成数据库文件的映射-->
<plugin>
  <groupId>org.mybatis.generator</groupId>
  <artifactId>mybatis-generator-maven-plugin</artifactId>
  <version>1.3.5</version>
  <dependencies>  生成对应的plugin需要的一些dependencies
    <dependency>
      <groupId>org.mybatis.generator</groupId>
      <artifactId>mybatis-generator-core</artifactId>
      <version>1.3.5</version>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.41</version>
    </dependency>
  </dependencies>
  <executions>
    <execution>
      <id>mybatis generator</id>
      <phase>package</phase>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <!--允许移动生成的文件-->
    <verbose>true</verbose>
    <!--允许自动覆盖文件（生产环境中千万不要这样做）-->
    <overwrite>true</overwrite>
    <configurationFile>      mybatisgeneration的配置文件路径
      src/main/resources/mybatis-generator.xml
    </configurationFile>
  </configuration>
</plugin>
```

## 2.4 Mybatis自动生成器的使用方式

### 1.新建文件src/main/resources/mybatis-generator.xml，从官网下载xml配置文件

### http://www.mybatis.org/generator/configreference/xmlconfig.html

### 2.新建数据库

### 新建一个miaosha的数据库，并建立两张表，分别是user_info和user_password

创建数据库：用户信息表和密码表

密码是加密的字符串，不与主表创建在一起企业级密码的信息是另一个系统，在项目中分表存放，密码要以密文的方式存储

### 3.修改配置文件

```
<?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>

    <context id="DB2Tables" targetRuntime="MyBatis3">
        <!--数据库链接地址账号密码-->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
                        connectionURL="jdbc:mysql://localhost:3306/miaosha"
                        userId="root"
                        password="123456">
        </jdbcConnection>

        <!--生成DataObject类存放位置-->
        <javaModelGenerator targetPackage="com.miaoshaproject.dataobject"
targetProject="src/main/java">
            <property name="enableSubPackages" value="true" />
            <property name="trimStrings" value="true" />
        </javaModelGenerator>

        <!--生成映射文件存放位置-->
        <sqlMapGenerator targetPackage="mapping"  targetProject="src/main/resources">
            <property name="enableSubPackages" value="true" />
        </sqlMapGenerator>

        <!--生成Dao类存放位置-->
        <javaClientGenerator type="XMLMAPPER" targetPackage="com.miaoshaproject.dao"
targetProject="src/main/java">
            <property name="enableSubPackages" value="true" />
        </javaClientGenerator>

        <!--生成对应表及类名-->
        <!--  enableCountByExample="false"
               enableUpdateByExample="false"
               enableDeleteByExample="false"
               enableSelectByExample="false"
               selectByExampleQueryId="false"
               这些属性是为了使得只生成简单查询的对应文件，去掉复杂查询的生成文件，因为一般开发中不太用的到--
>
        <table tableName="user_info" domainObjectName="UserDO"
               enableCountByExample="false"
               enableUpdateByExample="false"
               enableDeleteByExample="false"
               enableSelectByExample="false"
               selectByExampleQueryId="false"></table>
        <table tableName="user_password" domainObjectName="userPasswordDO"
               enableCountByExample="false"
               enableUpdateByExample="false"
               enableDeleteByExample="false"
               enableSelectByExample="false"
               selectByExampleQueryId="false" ></table>

    </context>
</generatorConfiguration>
```

## 4.生成文件

在终端运行 mvn mybatis-generator:generate 命令 自动生成dao类和对应的数据库文件

### 5.接入mysql数据源

```
spring.datasource.name=miaosha
spring.datasource.url=jdbc:mysql://localhost:3306/miaosha
spring.datasource.username=root
spring.datasource.password=123456

#使用druid数据源
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

### 6.测试数据库

修改App类 //使用@SpringbootApplication注解 可以解决根类或者配置类头上注解过多的问题，一个@SpringbootApplication相当于
//@Configuration,@EnableAutoConfiguration和 @ComponentScan 并具有他们的默认属性值

```
@SpringBootApplication(scanBasePackages = {"com.miaoshaproject"})
@RestController
@MapperScan("com.miaoshaproject.dao")
public class App {

    @Autowired
    private UserDOMapper userDOMapper;

    @RequestMapping("/")
    public String home() {
        UserDO userDO = userDOMapper.selectByPrimaryKey(1);
        if (userDO == null) {
            return "用户对象不存在";
        } else {
            return userDO.getName();
        }
    }
}
```

启动测试

# 第三章 用户模块开发

## 3.1 使用SpringMVC方式开发用户信息

1.增加controller层、dao层 package

创建UserController
new java的class：

```
@Controller("user") 标记，用来被spring扫描到
@RequestMapping("/user") 在url上使用/user访问到
public class UserController {

    @Autowired  引入userService
    private UserService userService;

    @RequestMapping("/get")
    @ResponseBody
    public UserModel getUser(@RequestParam(name = "id") Integer id) {
        //调用service服务获取对应id的用户对象并返回给前端
        UserModel userModel = userService.getUserById(id);
        return userModel;
    }
}
```

dataobject只是对数据库的一个映射，model层菜是真正处理业务的

userController需要UserModel

## 2.在service层增加UserModel

在service层增加UserService的接口和其实现UserServiceImpl

```
package com.miaoshaproject.service.model;

/**
 * @author KiroScarlet
 * @date 2019-05-15  -16:50
 */
public class UserModel {
    private Integer id;
    private String name;
    private Byte gender;
    private Integer age;
    private String telphone;
    private String regisitMode;
    private Integer thirdPartyId;
    private String encrptPassword;
}
```

UserModel需要增加 用户的密码，其通过userPasswordDOMapper从userPasswordDO得到

## 3.修改userPasswordDOMapper.xml和.java文件

### 增加方法

```
<select id="selectByUserId" parameterType="java.lang.Integer" resultMap="BaseResultMap">
  select
  <include refid="Base_Column_List" />
  from user_password
  where user_id = #{userId,jdbcType=INTEGER}
</select>
```

```
userPasswordDO selectByUserId(Integer UserId);
```

## 4.编写UserService

```
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDOMapper userDOMapper;

    @Autowired
    private userPasswordDOMapper userPasswordDOMapper;

    @Override
    public UserModel getUserById(Integer id) {
        //调用UserDOMapper获取到对应的用户dataobject
        UserDO userDO = userDOMapper.selectByPrimaryKey(id);
        if (userDO == null) {
            return null;
        }

        //通过用户id获取对应的用户加密密码信息
        userPasswordDO userPasswordDO = userPasswordDOMapper.selectByUserId(userDO.getId());

        return convertFromDataObject(userDO, userPasswordDO);
    }

    private UserModel convertFromDataObject(UserDO userDO,userPasswordDO userPasswordDO) {
        if (userDO == null) {
            return null;
        }

        UserModel userModel = new UserModel();
        BeanUtils.copyProperties(userDO, userModel);

        if (userPasswordDO != null) {
            userModel.setEncrptPassword(userPasswordDO.getEncrptPassword());
        }
        return userModel;

    }
}
```

### 5.这种方式存在的问题

> 直接给前端用户返回了UserModel，使得攻击者可以直接看到密码
>
> 需要在controller层增加一个viewobject模型对象

**只需要这些信息：**

```
public class UserVO {
    private Integer id;
    private String name;
    private Byte gender;
    private Integer age;
    private String telphone;
}
```

### 6.改造controller

```
public UserVO getUser(@RequestParam(name = "id") Integer id) {
   //调用service服务获取对应id的用户对象并返回给前端
   UserModel userModel = userService.getUserById(id);

   //将核心领域模型用户对象转化为可供UI使用的viewobject
   return convertFromModel(userModel);
}

private UserVO convertFromModel(UserModel userModel) {
   if (userModel == null) {
      return null;
   }
   UserVO userVO = new UserVO();
   BeanUtils.copyProperties(userModel, userVO);
   return userVO;

}
```

## 3.2 定义通用的返回对象——返回正确信息

**之前的程序一旦出错，只会返回一个白页，并没有错误信息，需要返回一个有意义的错误信息。**

### 1.增加一个response包。创建CommonReturnType类

200表示正常

```
public class CommonReturnType {

   //表明对应请求的返回处理结果“success”或“fail”
   private String status;

   //若status=success，则data内返回前端需要的json数据
   //若status=fail，则data内使用通用的错误码格式
   private Object data;

   //定义一个通用的创建方法
   public static CommonReturnType create(Object result) {
      return CommonReturnType.create(result, "success");
   }

   public static CommonReturnType create(Object result,String status) {
      CommonReturnType type = new CommonReturnType();
      type.setStatus(status);
      type.setData(result);
      return type;
   }
}
```

### 2.改造返回值

```
public CommonReturnType getUser(@RequestParam(name = "id") Integer id) {
    //调用service服务获取对应id的用户对象并返回给前端
    UserModel userModel = userService.getUserById(id);

    //将核心领域模型用户对象转化为可供UI使用的viewobject
    UserVO userVO = convertFromModel(userModel);

    //返回通用对象
    return CommonReturnType.create(userVO);
}
```

## 3.3 定义通用的返回对象——返回错误信息

### 1.创建error包

返回data值里面的error code和error message，方面前端处理不同的错误

### 2.创建commonError接口

```
public interface CommonError {
    public int getErrCode();

    public String getErrMsg();

    public CommonError setErrMsg(String errMs);
}
```

### 3.创建实现类

```
                    枚举类型
public enum EmBusinessError implements CommonError {
    //通用错误类型00001
    PARAMETER_VALIDATION_ERROR(00001, "参数不合法"),
                                                  不需要再定义无数的无效输入参数，比如用户名未传，邮箱未传等


    //10000开头为用户信息相关错误定义  全局需要一个统一的状态码流转
    USER_NOT_EXIST(10001, "用户不存在")
    ;

    private EmBusinessError(int errCode, String errMsg) {
        this.errCode = errCode;
        this.errMsg = errMsg;
    }

    private int errCode;    枚举可以拥有全局变量属性
    private String errMsg;

    @Override
    public int getErrCode() {
        return this.errCode;
    }

    @Override
    public String getErrMsg() {
        return this.errMsg;
    }

    @Override
    public CommonError setErrMsg(String errMsg) {
        this.errMsg = errMsg;
        return this;
    }
}
```

## 4.包装器模式实现BusinessException类

处理异常

```
/包装器业务异常实现
public class BusinessException extends Exception implements CommonError {

    private CommonError commonError;

    //直接接受EmBusinessError的传参用于构造业务异常
    public BusinessException(CommonError commonError) {
        super(); 注意要调用，因为Exception中会有自己的异常
        this.commonError = commonError;
    }

    //接收自定义errMsg的方式构造业务异常
    public BusinessException(CommonError commonError, String errMsg) {
        super();
        this.commonError = commonError;
        this.commonError.setErrMsg(errMsg); 二次改写setErrMsg
    }

    @Override
    public int getErrCode() {
        return this.commonError.getErrCode();
    }

    @Override
    public String getErrMsg() {
        return this.commonError.getErrMsg();
    }

    @Override
    public CommonError setErrMsg(String errMsg) {
        this.commonError.setErrMsg(errMsg);
        return this;
    }
}
```

### 5.抛出异常类

```
public CommonReturnType getUser(@RequestParam(name = "id") Integer id) throws
BusinessException {
    //调用service服务获取对应id的用户对象并返回给前端
    UserModel userModel = userService.getUserById(id);

    //若获取的对应用户信息不存在
    if (userModel == null) {
        throw new BusinessException(EmBusinessError.USER_NOT_EXIST);
    }
    //将核心领域模型用户对象转化为可供UI使用的viewobject
    UserVO userVO = convertFromModel(userModel);

    //返回通用对象
    return CommonReturnType.create(userVO);
}
```

上面只是定义了异常而没有处理异常

## 3.4 定义通用的返回对象——异常处理

### 1.定义exceptionHandler解决未被controller层吸收的exception

```
public class BaseController {

    //定义exceptionHandler解决未被controller层吸收的exception
    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.OK) 捕获到controller抛出的异常并返回.ok
    @ResponseBody
    public Object handlerException(HttpServletRequest request, Exception ex) {
        Map<String, Object> responseData = new HashMap<>();
        if (ex instanceof BusinessException) {                强转成BusinessException
            BusinessException businessException = (BusinessException) ex;
            responseData.put("errCode", businessException.getErrCode());
            responseData.put("errMsg", businessException.getErrMsg());
        } else {
            responseData.put("errCode", EmBusinessError.UNKNOWN_ERROR.getErrCode());
            responseData.put("errMsg", EmBusinessError.UNKNOWN_ERROR.getErrMsg());
        }
        return CommonReturnType.create(responseData, "fail");
    }

}
```

# 3.5 用户模型管理——otp验证码获取

```
public class UserController extends BaseController{

    @Autowired
    private UserService userService;

    @Autowired
    private HttpServletRequest httpServletRequest;

    //用户获取otp短信接口
    @RequestMapping("/getotp")
    @ResponseBody                      浏览器穿入参数 http://localhost:8090/user/getotp?telphone="12345"
    public CommonReturnType getOtp(@RequestParam(name = "telphone") String telphone) {
        //需要按照一定的规则生成OTP验证码
        Random random = new Random();
        int randomInt = random.nextInt(99999);
        randomInt += 10000;
        String otpCode = String.valueOf(randomInt);

                                key-value，telphone-otpCode，企业级一般用redisa：分布式，可重复覆盖
        //将OTP验证码同对应用户的手机号关联，使用httpsession的方式绑定手机号与OTPCDOE
        httpServletRequest.getSession().setAttribute(telphone, otpCode);

        //将OTP验证码通过短信通道发送给用户，省略
        System.out.println("telphone=" + telphone + "&otpCode=" + otpCode);
                                    实际千万能这么写，会把用户的otpCode传入日志中

        return CommonReturnType.create(null);
    }
```

### 测试，在控制台打印数据

## 3.6 用户模型管理——Metronic模板简介

采用前后端分离的思想，建立一个html文件夹，引入static文件夹

前端文件保存在本地的哪个盘下都可以，因为是通过ajax来异步获取接口

## 3.7 用户模型管理——getotp页面实现

1.getotp.html：

```
<html>
<head>
    <meta charset="UTF-8">        引入jquery的版本
    <script src="static/assets/global/plugins/jquery-1.11.0.min.js" type="text/javascript"></script>
    <title>Title</title>
</head>
<body>
    <div>
        <h3>获取otp信息</h3>
        <div>                                页面
            <label>手机号</label>
            <div>
                <input type="text" placeholder="手机号" name="telphone" id="telphone"/>
            </div>
        </div>
        <div>
            <button id="getotp" type="submit">
                获取otp短信
            </button>
        </div>
    </div>

</body>

<script>        交互
   jQuery(document).ready(function () {

       //绑定otp的click事件用于向后端发送获取手机验证码的请求
       $("#getotp").on("click",function () {
                                             判空处理
           var telphone=$("#telphone").val();
           if (telphone==null || telphone=="") {
               alert("手机号不能为空");
               return false; 捕获onclick事件，使其不传入上一层
           }


           //映射到后端@RequestMapping(value = "/getotp", method = {RequestMethod.POST},
consumes = {CONTENT_TYPE_FORMED}) 在basecontroller中声明一个静态成员 public static final String  CONTENT_TYPE_FORMED="
           $.ajax({                            application/x-www-form-urlencoded";
               type:"POST",
               contentType:"application/x-www-form-urlencoded",
               url:"http://localhost:8080/user/getotp", POST请求要发送的地址
```

```
                data:{          传参
                    "telphone":$("#telphone").val(),
                },                              定义两个回调：ajax请求成功，ajax请求失败
                success:function (data) {
                    if (data.status=="success") {  业务逻辑请求成功
                        alert("otp已经发送到了您的手机，请注意查收");
                    }else {
                        alert("otp发送失败，原因为" + data.data.errMsg);
                    }
                },
                error:function (data) {
                    alert("otp发送失败，原因为"+data.responseText);
                }
            });
        });
    });
</script>
</html>
```

### 2.指定controller的method

@RequestMapping(value = "/getotp", method = {RequestMethod.POST}, consumes = {CONTENT_TYPE_FORMED})

### 3.提示发送失败，使用chrome调试，发现报错为

getotp.html?_ijt=cqdae6hmhq9069c9s4muooakju:1 Access to XMLHttpRequest at 'http://localhost:8080/user/getotp' from origin 'http://localhost:63342' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

**跨域请求错误，只需要在UserController类上加一个注解 @CrossOrigin 即可**

## 3.8 用户模型管理——getotp页面美化

### 1.引入样式表

引入文件

<link href="static/assets/global/plugins/bootstrap/css/bootstrap.min.css" rel="stylesheet" type="text/css"/>
<link href="static/assets/global/~~plugins~~/css/component.css" rel="stylesheet" type="text/css"/>
<link href="static/assets/admin/pages/css/login.css" rel="stylesheet" type="text/css"/>

### 2.使用样式

```
<body class="login">
    <div class="content">
        <h3 class="form-title">获取otp信息</h3>
        <div class="form-group">
            <label class="control-label">手机号</label>
            <div>
                <input class="form-control" type="text" placeholder="手机号" name="telphone"
id="telphone"/>
            </div>
        </div>
        <div class="form-actions">
            <button class="btn blue" id="getotp" type="submit">
                获取otp短信
            </button>
        </div>
    </div>

</body>
```

## 3.9 用户模型管理——用户注册功能实现

### 1.实现方法：用户注册接口

```
    //用户注册接口
    @RequestMapping(value = "/register", method = {RequestMethod.POST}, consumes =
{CONTENT_TYPE_FORMED})
    @ResponseBody
    public CommonReturnType register(@RequestParam(name = "telphone") String telphone,
                                     @RequestParam(name = "otpCode") String otpCode,
                                     @RequestParam(name = "name") String name,
                                     @RequestParam(name = "gender") String gender,
                                     @RequestParam(name = "age") String age,
                                     @RequestParam(name = "password") String password) throws
BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {

        //验证手机号和对应的otpCode相符合
        String inSessionOtpCode = (String) this.httpServletRequest.getSession().getAttribute(telphone);
        if (!com.alibaba.druid.util.StringUtils.equals(otpCode, inSessionOtpCode)) {
            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "短信验证码
不符合");
        }
```

为什么要用类库里的equals，因为先会判断a==null

首先要在UserService中定义register方法

```
        //用户的注册流程
        UserModel userModel = new UserModel();
        userModel.setName(name);
        userModel.setAge(Integer.valueOf(age));
        userModel.setGender(Byte.valueOf(gender));
        userModel.setTelphone(telphone);
        userModel.setRegisitMode("byphone");

        //密码加密
        userModel.setEncrptPassword(this.EncodeByMd5(password));

        userService.register(userModel);
        return CommonReturnType.create(null);

    }

    //密码加密
    public String EncodeByMd5(String str) throws NoSuchAlgorithmException,
UnsupportedEncodingException {
        //确定计算方法
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        BASE64Encoder base64en = new BASE64Encoder();
        //加密字符串
        String newstr = base64en.encode(md5.digest(str.getBytes("utf-8")));
        return newstr;
    }
```

将异常抛给上层

## 2.引入做输入校验的依赖

加入pom文件中

```
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.7</version>
</dependency>
```

### 3.UserServiceImpl的register方法

```
@Override
@Transactional//声明事务使得用户信息和密码属于同一事物，避免一半成功一半不成功
public void register(UserModel userModel) throws BusinessException {
    //校验
    if (userModel == null) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
    }
    if (StringUtils.isEmpty(userModel.getName())
            || userModel.getGender() == null
            || userModel.getAge() == null
            || StringUtils.isEmpty(userModel.getTelphone())) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
    }

    //实现model->dataobject方法
    UserDO userDO = convertFromModel(userModel);
    //insertSelective相对于insert方法，不会覆盖掉数据库的默认值
    userDOMapper.insertSelective(userDO);

    userModel.setId(userDO.getId());
```

有的字段是唯一索引但是有时候是null（都是null不算重复），比如，手机号只能有一个，但是通过第三方账号登录时没有手机号，除非强绑定

在数据库尽量避免使用null字段
1.java中null很脆弱，
2.null在前端没有意义

```
    userPasswordDO userPasswordDO = convertPasswordFromModel(userModel);
    userPasswordDOMapper.insertSelective(userPasswordDO);
```

拿到了userDO后，setId，但是insertSelective没有指定数据库的keyProperty（id)。因此需要将id取出后赋给userModel再传给password

```
    return;
}

private userPasswordDO convertPasswordFromModel(UserModel userModel) {
    if (userModel == null) {
        return null;
    }
    userPasswordDO userPasswordDO = new userPasswordDO();
    userPasswordDO.setEncrptPassword(userModel.getEncrptPassword());
    userPasswordDO.setUserId(userModel.getId());

    return userPasswordDO;
}

private UserDO convertFromModel(UserModel userModel) {
    if (userModel == null) {
        return null;
    }
    UserDO userDO = new UserDO();
    BeanUtils.copyProperties(userModel, userDO);
    return userDO;
}
```

### 4.前端界面

### 首先在getotp界面添加注册成功的跳转界面

```
success:function (data) {
    if (data.status=="success") {
        alert("otp已经发送到了您的手机，请注意查收");
        window.location.href="register.html";
    }else {
        alert("otp发送失败，原因为" + data.data.errMsg);
    }
},
```

### 模仿之前写的界面，新建一个register.html

```
<body class="login">
    <div class="content">
        <h3 class="form-title">用户注册</h3>
        <div class="form-group">
            <label class="control-label">手机号</label>
            <div>
                <input class="form-control" type="text" placeholder="手机号" name="telphone"
id="telphone"/>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label">验证码</label>
            <div>
                <input class="form-control" type="text" placeholder="验证码" name="otpCode"
id="otpCode"/>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label">用户昵称</label>
            <div>
                <input class="form-control" type="text" placeholder="用户昵称" name="name"
id="name"/>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label">性别</label>
            <div>
                <input class="form-control" type="text" placeholder="性别" name="gender"
id="gender"/>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label">年龄</label>
            <div>
                <input class="form-control" type="text" placeholder="年龄" name="age" id="age"/>
            </div>
        </div>
        <div class="form-group">
            <label class="control-label">密码</label>
            <div>
                <input class="form-control" type="password" placeholder="密码" name="password"
id="password"/>
            </div>
        </div>
        <div class="form-actions">
            <button class="btn blue" id="register" type="submit">
                提交注册
            </button>
        </div>
    </div>

</body>

<script>
    jQuery(document).ready(function () {

        //绑定otp的click事件用于向后端发送获取手机验证码的请求
        $("#register").on("click",function () {

            var telphone=$("#telphone").val();
            var otpCode=$("#otpCode").val();
            var password=$("#password").val();
            var age=$("#age").val();
            var gender=$("#gender").val();
            var name=$("#name").val();
            if (telphone==null || telphone=="") {
                alert("手机号不能为空");
                return false;
            }
            if (otpCode==null || otpCode=="") {
                alert("验证码不能为空");
                return false;
            }
            if (name==null || name=="") {
                alert("用户名不能为空");
                return false;
            }
            if (gender==null || gender=="") {
                alert("性别不能为空");
                return false;
            }
            if (age==null || age=="") {
                alert("年龄不能为空");
                return false;
            }
            if (password==null || password=="") {
                alert("密码不能为空");
                return false;
            }

            //映射到后端@RequestMapping(value = "/register", method = {RequestMethod.POST},
consumes = {CONTENT_TYPE_FORMED})
            $.ajax({
                type:"POST",
                contentType:"application/x-www-form-urlencoded",
                url:"http://localhost:8080/user/register",
                data:{
```

**校验必须发生在离用户最近的地方，（为什么在后端校验了一遍，前端也要校验一遍）这样用户可以及时知道输入的错误而不需要去服务器绕一圈**

**debug**

```
                    "telphone":telphone,
                    "otpCode":otpCode,
                    "password":password,
                    "age":age,
                    "gender":gender,
                    "name":name
                },
                //允许跨域请求
                xhrFields:{withCredentials:true},
                success:function (data) {
                    if (data.status=="success") {
                        alert("注册成功");
                    }else {
                        alert("注册失败，原因为" + data.data.errMsg);
                    }
                },
                error:function (data) { 比如由于网络的原因失败
                    alert("注册失败，原因为"+data.responseText);
                }
            });
            return false;
        });
    });
</script>
```

5.调试

发现报错，获取不到验证码 insession中的otpcode为null

跨域请求问题

在UserController上添加如下注解：

这里其实没明白

首先指定crossOrigin的范围，接着在前端（getotp和register）中指定xhrFields

```
//跨域请求中，不能做到session共享
@CrossOrigin(allowCredentials = "true",allowedHeaders = "*")
```

DEFAULT_ALLOWED_HEADERS:允许跨域传输苏所有的header参数，将用于使用token放入header域做session共享的跨域请求

6.注册成功，但是查看数据库，发现password表中并没有user_id

在UserDOMapper的insertSelective方法中添加如下代码：

```
 <insert id="insertSelective" parameterType="com.miaoshaproject.dataobject.UserDO"
keyProperty="id" useGeneratedKeys="true">
```

通过这样的方式将自增id取出之后复制给对应的UserDO

7.修改UserServiceImpl

```
UserDO userDO = convertFromModel(userModel);
//insertSelective相对于insert方法，不会覆盖掉数据库的默认值
userDOMapper.insertSelective(userDO);

userModel.setId(userDO.getId());

userPasswordDO userPasswordDO = convertPasswordFromModel(userModel);
userPasswordDOMapper.insertSelective(userPasswordDO);

return;
```

**重新测试成功**

8.**上面并没有做手机号的唯一性验证**

**首先，在数据库中添加索引：**

**索引名称为：**telphone_unique_index，**索引字段选择**telphone，**索引类型为**UNIQUE，**索引方法为**BTREE

**然后修改以下代码：**

```
try {
    userDOMapper.insertSelective(userDO);
} catch (DuplicateKeyException ex) {
    throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "手机号已注册");
}
```

**为了给用户一个良好的提示**

## 3.9 用户模型管理——用户登录功能实现

1.UserController**中的用户登录接口**

```
    //用户登录接口
    @RequestMapping(value = "/login", method = {RequestMethod.POST}, consumes =
{CONTENT_TYPE_FORMED})
    @ResponseBody
    public CommonReturnType login(@RequestParam(name = "telphone") String telphone,
                                  @RequestParam(name = "password") String password) throws
BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {
        //入参校验
        if (StringUtils.isEmpty(telphone) || StringUtils.isEmpty(password)) {
            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
        }

        //用户登录服务，用来校验用户登录是否合法
        //用户加密后的密码
        UserModel userModel = userService.validateLogin(telphone, this.EncodeByMd5(password));
      在没有抛出异常的前提下才会进入下一步
        //将登陆凭证加入到用户登录成功的session内   目前是假设用户是单点登录，实际：分布式
        this.httpServletRequest.getSession().setAttribute("IS_LOGIN", true);
        this.httpServletRequest.getSession().setAttribute("LOGIN_USER", userModel);

        return CommonReturnType.create(null);

    }
```

## 2.UserService中的校验登录方法

```
    /*
    telphone:用户注册手机
    encrptPassowrd:用户加密后的密码
     */
    UserModel validateLogin(String telphone, String encrptPassword) throws BusinessException;
```

## 3.UserServiceImpl的登录方法实现

```
    @Override
    public UserModel validateLogin(String telphone, String encrptPassword) throws BusinessException {
        //通过用户手机获取用户信息
        UserDO userDO = userDOMapper.selectByTelphone(telphone);
        if (userDO == null) {
            throw new BusinessException(EmBusinessError.USER_LOOGIN_FAIL);
        }                                   USER_LOOGIN_FAIL(20002, "用户手机号或密码不正确"),
        userPasswordDO userPasswordDO = userPasswordDOMapper.selectByUserId(userDO.getId());
        UserModel userModel = convertFromDataObject(userDO, userPasswordDO);

        //比对用户信息内加密的密码是否和传输进来的密码相匹配
        if (StringUtils.equals(encrptPassword, userModel.getEncrptPassword())) {
            throw new BusinessException(EmBusinessError.USER_LOOGIN_FAIL);
        }

        return userModel;
    }
```

## 4.UserDOMapper.xml中的新建方法

```
<select id="selectByTelphone" resultMap="BaseResultMap">
   select
   <include refid="Base_Column_List"/>
   from user_info
   where telphone = #{telphone,jdbcType=VARCHAR}
</select>
```

## 5.UserDOMapper中建立映射

.java

```
//根据电话号码取得用户对象
UserDO selectByTelphone(String telphone);
```

## 6.新建前端界面：login.html

```
<body class="login">
   <div class="content">
      <h3 class="form-title">用户登录</h3>
      <div class="form-group">
         <label class="control-label">手机号</label>
         <div>
            <input class="form-control" type="text" placeholder="手机号" name="telphone"
id="telphone"/>
         </div>
      </div>
      <div class="form-group">
         <label class="control-label">密码</label>
         <div>
            <input class="form-control" type="password" placeholder="密码" name="password"
id="password"/>
         </div>
      </div>
      <div class="form-actions">
         <button class="btn blue" id="login" type="submit">
            登录
         </button>
         <button class="btn green" id="register" type="submit">
            注册
         </button>
      </div>
   </div>

</body>

<script>
   jQuery(document).ready(function () {

      //绑定注册按钮的click事件用于跳转到注册页面
      $("#register").on("click",function () {
         window.location.href = "getotp.html";
      });

      //绑定登录按钮的click事件用于登录
      $("#login").on("click",function () {
        var telphone=$("#telphone").val();
        var password=$("#password").val();
        if (telphone==null || telphone=="") {
          alert("手机号不能为空");
          return false;
        }
        if (password==null || password=="") {
          alert("密码不能为空");
          return false;
        }

        //映射到后端@RequestMapping(value = "/login", method = {RequestMethod.POST}, consumes
= {CONTENT_TYPE_FORMED})
        $.ajax({
          type:"POST",
          contentType:"application/x-www-form-urlencoded",
          url:"http://localhost:8080/user/login",
          data:{
            "telphone":telphone,
            "password":password
          },
          //允许跨域请求
          xhrFields:{withCredentials:true},
          success:function (data) {
            if (data.status=="success") {
              alert("登录成功");
            }else {
              alert("登录失败，原因为" + data.data.errMsg);
            }
          },
          error:function (data) {
            alert("登录失败，原因为"+data.responseText);
          }
        });
        return false;
      });
    });
```

比如age目前只判断是否为null，没有判断是否小于0等

## 3.10 优化校验规则

### 1.查询maven仓库中是否由可用类库

pom.xml

```
<!--校验-->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.2.4.Final</version>
</dependency>
```

### 2.对validator进行一个简单的封装

新建validator的目录

新建一个ValidationResult的类

```
public class ValidationResult {
    //校验结果是否有错
    private boolean hasErrors = false;

    //存放错误信息的map
    private Map<String, String> errorMsgMap = new HashMap<>();

    public boolean isHasErrors() {
        return hasErrors;
    }

    public void setHasErrors(boolean hasErrors) {
        this.hasErrors = hasErrors;
    }

    public Map<String, String> getErrorMsgMap() {
        return errorMsgMap;
    }

    public void setErrorMsgMap(Map<String, String> errorMsgMap) {
        this.errorMsgMap = errorMsgMap;
    }

    //实现通用的通过格式化字符串信息获取错误结果的msg方法
    public String getErrMsg() {
        return StringUtils.join(errorMsgMap.values().toArray(), ",");
    }
}
```

**可能很多个字段会有错误**

## 新建一个ValidatiorImpl的类

```
@Component 类扫描时会扫描到
public class ValidatorImpl implements InitializingBean {

    private Validator validator;

    //实现校验方法并返回校验结果
    public ValidationResult validate(Object bean) {
        final ValidationResult result = new ValidationResult();
        Set<ConstraintViolation<Object>> constraintViolationSet = validator.validate(bean);
        if (constraintViolationSet.size() > 0) {
            //有错误
            result.setHasErrors(true);      遍历，lamda表达式
            constraintViolationSet.forEach(constraintViolation ->{
                String errMsg = constraintViolation.getMessage(); 存放了它所违背的信息
                String propertyName = constraintViolation.getPropertyPath().toString(); 得到是哪个字段发生了什么错误
                result.getErrorMsgMap().put(propertyName, errMsg);
            });
        }
        return result;
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        //将hibernate validator通过工厂的初始化方式使其实例化
        this.validator = Validation.buildDefaultValidatorFactory().getValidator();
    }                                                            得到一个校验器
}
```

## 3.修改UserModel，基于注解的校验方式

```
@NotBlank(message = "用户名不能为空")
private String name;

@NotNull(message = "性别不能填写")
private Byte gender;           不

@NotNull(message = "年龄不能不填写")
@Min(value = 0, message = "年龄必须大于0岁")
@Max(value = 150, message = "年龄必须小于150岁")
private Integer age;

@NotBlank(message = "手机号不能为空")
private String telphone;
private String regisitMode;
private Integer thirdPartyId;

@NotBlank(message = "密码不能为空")
private String encrptPassword;
```

## 4.在UserServiceImpl中使用validator

引入bean

```
@Autowired
private ValidatorImpl validator;

        //校验
//        if (userModel == null) {
//            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
//        }
//        if (StringUtils.isEmpty(userModel.getName())
//                || userModel.getGender() == null
//                || userModel.getAge() == null
//                || StringUtils.isEmpty(userModel.getTelphone())) {
//            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
//        }

        ValidationResult result = validator.validate(userModel);
        if (result.isHasErrors()) {
            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR,
result.getErrMsg());
        }
```

**以后做校验时只需要在model的属性上做注解即可**

# 第四章 商品模块开发

## 4.1 商品模型管理——商品创建

**设计到字段的分离，所以首先不要定数据库**

### 1.首先设计商品领域模型

**使用mybatis-generator生成数据库表的数据结构**

```
public class ItemModel {
    private Integer id;

    //商品名称
    private String title;

    //商品价格
    private BigDecimal price;
```

**因为double传到前端可能会有一个精度问题**

```
    //商品的库存
    private Integer stock;

    //商品的描述
    private String description;

    //商品的销量
    private Integer sales;

    //商品描述图片的url
    private String imgUrl;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public Integer getStock() {
        return stock;
    }

    public void setStock(Integer stock) {
        this.stock = stock;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Integer getSales() {
        return sales;
    }

    public void setSales(Integer sales) {
        this.sales = sales;
    }

    public String getImgUrl() {
        return imgUrl;
    }

    public void setImgUrl(String imgUrl) {
        this.imgUrl = imgUrl;
    }
}
```

## 2.设计数据库

两张表：商品表和库存表

## 3.修改pom文件

```
<!--允许移动生成的文件-->
<verbose>true</verbose>
<!--允许自动覆盖文件（生产环境中千万不要这样做）-->
<overwrite>false</overwrite>
```

否则生成新的mapper文件会覆盖旧的，比如item覆盖user

## 4.修改mybatis-generator配置文件

注释掉不需要重新生成的DO

```
<table tableName="item" domainObjectName="ItemDO"
enableCountByExample="false"            enableUpdateByExample="false"
enableDeleteByExample="false"            enableSelectByExample="false"
selectByExampleQueryId="false" ></table>

<table tableName="item_stock" domainObjectName="ItemStockDO"
enableCountByExample="false"            enableUpdateByExample="false"
enableDeleteByExample="false"            enableSelectByExample="false"
selectByExampleQueryId="false" ></table>
```

添加两张表

运行 `mvn mybatis-generator:generate`

删除无用的文件

## 5.修改mapper的xml文件

把insert和insertSelective方法后添加属性 `keyProperty="id" useGeneratedKeys="true""` ，使其保持自增

## 6.创建ItemService接口

```
public interface ItemService {

    //创建商品
    ItemModel createItem(ItemModel itemModel);

    //商品列表浏览
    List<ItemModel> listItem();


    //商品详情浏览
    ItemModel getItemById(Integer id);
}
```

## 7.ItemServiceImpl实现类

入参校验

ItemModel里

```
//商品名称
@NotBlank(message = "商品名称不能为空")
private String title;

//商品价格
@NotNull(message = "商品价格不能为空")
@Min(value = 0,message = "商品价格必须大于0")
private BigDecimal price;

//商品的库存
@NotNull(message = "库存不能不填")
private Integer stock;

//商品的描述
@NotBlank(message = "商品描述信息不能为空")
private String description;

//商品的销量 不是创建时传入，是通过统计得到
~~@NotBlank(message = "商品图片信息不能为空")~~
private Integer sales;

//商品描述图片的url
private String imgUrl;
```

## 实现方法

```
@Service
public class ItemServiceImpl implements ItemService {

    @Autowired
    private ValidatorImpl validator;

    @Autowired
    private ItemDOMapper itemDOMapper;

    @Autowired
    private ItemStockDOMapper itemStockDOMapper;

    @Override
    @Transactional
    public ItemModel createItem(ItemModel itemModel) throws BusinessException {

  1     //校验入参
        ValidationResult result = validator.validate(itemModel);
        if (result.isHasErrors()) {
            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR,
result.getErrMsg());
        }
        //转化itemmodel->dataobject
        ItemDO itemDO = this.convertItemDOFromItemModel(itemModel);

  2     //写入数据库
        itemDOMapper.insertSelective(itemDO);
        itemModel.setId(itemDO.getId());
        ItemStockDO itemStockDO = this.convertItemStockDOFromItemModel(itemModel);
        itemStockDOMapper.insertSelective(itemStockDO);

3
        //返回创建完成的对象
        return this.getItemById(itemModel.getId());
                          通过service层的getItemById
    }

    private ItemDO convertItemDOFromItemModel(ItemModel itemModel) {
        if (itemModel == null) {
            return null;
        }
        ItemDO itemDO = new ItemDO();
        BeanUtils.copyProperties(itemModel, itemDO);
        return itemDO;                             itemDO.setPrice()
    }

    private ItemStockDO convertItemStockDOFromItemModel(ItemModel itemModel) {
        if (itemModel == null) {
            return null;
        }
        ItemStockDO itemStockDO = new ItemStockDO();
        itemStockDO.setItemId(itemModel.getId());
        itemStockDO.setStock(itemModel.getStock());

        return itemStockDO;
    }

    @Override
    public List<ItemModel> listItem() {
        return null;
    }

    @Override
    public ItemModel getItemById(Integer id) {
        ItemDO itemDO = itemDOMapper.selectByPrimaryKey(id);
        if (itemDO == null) {
            return null;
        }
                                                           添加方法
        //操作获得库存数量
        ItemStockDO itemStockDO = itemStockDOMapper.selectByItemId(itemDO.getId());

        //将dataobject-> Model
        ItemModel itemModel = convertModelFromDataObject(itemDO, itemStockDO);
        return itemModel;
    }

    private ItemModel convertModelFromDataObject(ItemDO itemDO, ItemStockDO itemStockDO) {
        ItemModel itemModel = new ItemModel();
        BeanUtils.copyProperties(itemDO, itemModel);
        itemModel.setStock(itemStockDO.getStock());
        return itemModel;
    }
```

```
}
```

## 8.ItemController

item里的字段都可以暴露给前端，所以itemVO和itemController相同

```
@Controller("/item")
@RequestMapping("/item")
//跨域请求中，不能做到session共享
@CrossOrigin(origins = {"*"}, allowCredentials = "true")
public class ItemController extends BaseController {

    @Autowired
    private ItemService itemService;

    //创建商品的controller
    @RequestMapping(value = "/create", method = {RequestMethod.POST}, consumes =
{CONTENT_TYPE_FORMED})
    @ResponseBody
    public CommonReturnType createItem(@RequestParam(name = "title") String title,
                                       @RequestParam(name = "description") String description,
                                       @RequestParam(name = "price") BigDecimal price,
                                       @RequestParam(name = "stock") Integer stock,
                                       @RequestParam(name = "imgUrl") String imgUrl) throws BusinessException
{
        //封装service请求用来创建商品
        ItemModel itemModel = new ItemModel();
        itemModel.setTitle(title);
        itemModel.setDescription(description);
        itemModel.setPrice(price);
        itemModel.setStock(stock);
        itemModel.setImgUrl(imgUrl);

        ItemModel itemModelForReturn = itemService.createItem(itemModel);
        ItemVO itemVO = convertVOFromModel(itemModelForReturn);
        return CommonReturnType.create(itemVO);

    }

    private ItemVO convertVOFromModel(ItemModel itemModel) {
        if (itemModel == null) {
            return null;
        }
        ItemVO itemVO = new ItemVO();
        BeanUtils.copyProperties(itemModel, itemVO);
        return itemVO;
    }

}
```

sales与创建商品无关

尽可能使controller层简单，把复杂的都放在service层

前端页面 createItem.html

【1】 创建存放校验信息的类ValidationResult　　【2】创建一个用于校验的类ValidateImpl，提供方法返回校验信息的方法，里面包括校验的具体实现

## 9.商品详情页浏览

```
@RequestMapping(value = "/get", method = {RequestMethod.GET}) 去掉consumes
@ResponseBody
public CommonReturnType getItem(@RequestParam(name = "id") Integer id) {
    ItemModel itemModel = itemService.getItemById(id);

    ItemVO itemVO = convertVOFromModel(itemModel);

    return CommonReturnType.create(itemVO);
}
```

## 4.2 商品模型管理——商品列表

*假设我们的需求是按照销量从高到低显示所有商品*

### 1.创建sql语句

### 在ItemDOMapper.xml中新建方法

没有参数

```
<select id="listItem"  resultMap="BaseResultMap">

  select
  <include refid="Base_Column_List" />
  /*通过销量倒序排序*/
  from item ORDER BY sales DESC;
</select>
```

### 2.在ItemDOMapper中创建方法

```
List<ItemDO> listItem();
```

### 3.在ItemServiceImpl中实现方法

```
@Override
public List<ItemModel> listItem() {
    List<ItemDO> itemDOList = itemDOMapper.listItem();

    //使用Java8的stream API
    List<ItemModel> itemModelList = itemDOList.stream().map(itemDO -> {
        ItemStockDO itemStockDO = itemStockDOMapper.selectByItemId(itemDO.getId());
        ItemModel itemModel = this.convertModelFromDataObject(itemDO, itemStockDO);
        return itemModel;
    }).collect(Collectors.toList());

    return itemModelList;
}
```

### 4.controller层

```
//商品列表页面浏览
@RequestMapping(value = "/list", method = {RequestMethod.GET})
@ResponseBody
public CommonReturnType listItem() {
    List<ItemModel> itemModelList = itemService.listItem();
    List<ItemVO> itemVOList = itemModelList.stream().map(itemModel -> {
        ItemVO itemVO = this.convertVOFromModel(itemModel);   将itemModel转化成itemVO
        return itemVO;
    }).collect(Collectors.toList());

    return CommonReturnType.create(itemVOList);
}
```

## 4.3 商品模型管理——商品列表页面

## 4.4 商品模型管理——商品详情页面

# 第五章 交易模块开发

## 5.1 交易模型管理——交易模型创建

### 1.先设计用户下单的交易模型 **用户下单流程**

```
//用户下单的交易模型
public class OrderModel {
    //交易单号，例如2019052100001212，使用string类型
    private String id;

    //购买的用户id
    private Integer userId;

    //购买的商品id
    private Integer itemId;

    //购买时商品的单价
    private BigDecimal itemPrice;

    //购买数量
    private Integer amount;

    //购买金额
    private BigDecimal orderPrice;

    ...
}
```

### 2.设计数据库

非自增

```
CREATE TABLE `order_info`  (
  `id` varchar(32) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `user_id` int(11) NOT NULL DEFAULT 0,
  `item_id` int(11) NOT NULL DEFAULT 0,
  `item_price` decimal(10, 2) NOT NULL DEFAULT 0.00,
  `amount` int(11) NOT NULL DEFAULT 0,
  `order_price` decimal(40, 2) NOT NULL DEFAULT 0.00,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_bin ROW_FORMAT = Compact;
```

### 3.修改配置

domai nname

```
<table tableName="order_info" domainObjectName="OrderDO"
       enableCountByExample="false"
       enableUpdateByExample="false"
       enableDeleteByExample="false"
       enableSelectByExample="false"
       selectByExampleQueryId="false" ></table>
```

### 4.生成文件

在终端运行 `mvn mybatis-generator:generate` 命令

## 5.2 交易模型管理——交易下单

### 1.OrderService

```
public interface OrderService {

    OrderModel createOrder(Integer userId, Integer itemId, Integer amount) throws BusinessException;
}
```

### 2.OrderServiceImpl

```
@Override
@Transactional
public OrderModel createOrder(Integer userId, Integer itemId, Integer amount) throws
BusinessException {
    //1.校验下单状态，下单的商品是否存在，用户是否合法，购买数量是否正确
    ItemModel itemModel = itemService.getItemById(itemId);
    if (itemModel == null) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "商品信息不存
在");
    }

    UserModel userModel = userService.getUserById(userId);
    if (userModel == null) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "用户信息不存
在");
    }

    if (amount <= 0 || amount > 99) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "数量信息不存
在");
    }

    //2.落单减库存  支付减库存：无法保证超卖，在商家想要确保成交率时可以使用
    boolean result = itemService.decreaseStock(itemId, amount);
    if (!result) { 减库存失败
        throw new BusinessException(EmBusinessError.STOCK_NOT_ENOUGH);
    }

    //3.订单入库

    //4.返回前端
}
```

## 3.落单减库存

- ItemService

```
//库存扣减
boolean decreaseStock(Integer itemId, Integer amount) throws BusinessException;
```

- ItemServiceImpl

```
@Override
@Transactional
public boolean decreaseStock(Integer itemId, Integer amount) throws BusinessException {
    int affectedRow = itemStockDOMapper.decreaseStock(itemId, amount);
    if (affectedRow > 0) {
        //更新库存成功
        return true;
    } else {
        //更新库存失败
        return false;
    }
}
```

- ItemStockMapper

```
int decreaseStock(@Param("itemId") Integer itemId, @Param("amount") Integer amount);
```

- ItemStockMapper.xml

```
<update id="decreaseStock">

  update item_stock
  set stock = stock-#{amount}   传入的参数
  where item_id = #{item_id} and stock>=#{amount}
</update>
```

## 4.生成交易流水号

### 新建一个数据库

```
CREATE TABLE `sequence_info`  (
  `name` varchar(255) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `current_value` int(11) NOT NULL DEFAULT 0,
  `step` int(11) NOT NULL DEFAULT 0,
  PRIMARY KEY (`name`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_bin ROW_FORMAT = Compact;
```

### 插入一条语句，用来生成当前流水号

```
INSERT INTO `sequence_info` VALUES ('order_info', 0, 1);
```

### 修改mybatis-generator

```
<table tableName="sequence_info" domainObjectName="SequenceDO"
       enableCountByExample="false"
       enableUpdateByExample="false"
       enableDeleteByExample="false"
       enableSelectByExample="false"
       selectByExampleQueryId="false" ></table>
```

### 在终端运行 `mvn mybatis-generator:generate` 命令

### 修改SequenceDOMapper.xml

```
<select id="getSequenceByName" parameterType="java.lang.String" resultMap="BaseResultMap">
  select
  <include refid="Base_Column_List" />
  from sequence_info
  where name = #{name,jdbcType=VARCHAR} for update   需要上锁
</select>
```

### 添加方法

```
SequenceDO getSequenceByName(String name);
```

```
@Transactional(propagation=Propagation.REQUIRES_NEW
private String generateOrderNo() {
    //订单有16位
    StringBuilder stringBuilder = new StringBuilder();
    //前8位为时间信息，年月日
    LocalDateTime now = LocalDateTime.now();
    String nowDate = now.format(DateTimeFormatter.ISO_DATE).replace("-", "");
    stringBuilder.append(nowDate);

    //中间6位为自增序列
    //获取当前sequence
    int sequence = 0;    sequence表
    SequenceDO sequenceDO = sequenceDOMapper.getSequenceByName("order_info");

    sequence = sequenceDO.getCurrentValue();
    sequenceDO.setCurrentValue(sequenceDO.getCurrentValue() + sequenceDO.getStep());
    sequenceDOMapper.updateByPrimaryKeySelective(sequenceDO); 更新
    //拼接 凑足6位
    String sequenceStr = String.valueOf(sequence);
    for (int i = 0; i < 6 - sequenceStr.length(); i++) {
        stringBuilder.append(0);
    }
    stringBuilder.append(sequenceStr);

    //最后两位为分库分表位,暂时不考虑
    stringBuilder.append("00");    分散用户的查询

    return stringBuilder.toString();
}
```

只要执行完这部分代码块，无论外部事务成功与否，对应的事务都提交，对应的sequence都会被消耗掉

注解@Transactional的作用保证事务的一致下单函数createOrder中用到了@Transactional(propagation = Propagation.REQUIRED)表示执行代码后，不管成功与否，直接提交事务。不管该方法是否在事务中，都会开启一个新的事务，为了保证订单号的唯一性，防止下单失败后订单号的回滚

对应的sequence需要设置一个最大值：可以设置一个initvale来循环值

事物回滚也不应该回滚Sequence的值（由于tansctional会使争端代码都回滚）

## 5.销量增加

### itemDOMapper.xml

```
<update id="increaseSales">
  update item
  set sales = sales+ #{amount}
  where id = #{id,jdbcType=INTEGER}
</update>
```

### itemDOMapper

```
int increaseSales(@Param("id") Integer id, @Param("amount") Integer amount);
```

### ItemServiceImpl

```
@Override
@Transactional
public void increaseSales(Integer itemId, Integer amount) throws BusinessException {
    itemDOMapper.increaseSales(itemId,amount);
}
```

## 6.最终的OrderServiceImpl

```
@Override
@Transactional
public OrderModel createOrder(Integer userId, Integer itemId, Integer amount) throws BusinessException {
    //1.校验下单状态，下单的商品是否存在，用户是否合法，购买数量是否正确
    ItemModel itemModel = itemService.getItemById(itemId);
    if (itemModel == null) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "商品信息不存在");
    }

    UserModel userModel = userService.getUserById(userId);
    if (userModel == null) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "用户信息不存在");
    }

    if (amount <= 0 || amount > 99) {
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "数量信息不存在");
    }

    //2.落单减库存
    boolean result = itemService.decreaseStock(itemId, amount);
    if (!result) {
        throw new BusinessException(EmBusinessError.STOCK_NOT_ENOUGH);
    }

    //3.订单入库
    OrderModel orderModel = new OrderModel();
    orderModel.setUserId(userId);
    orderModel.setItemId(itemId);
    orderModel.setAmount(amount);
    orderModel.setItemPrice(itemModel.getPrice());
    orderModel.setOrderPrice(itemModel.getPrice().multiply(BigDecimal.valueOf(amount)));

    //生成交易流水号
    orderModel.setId(generateOrderNo());
    OrderDO orderDO = this.convertFromOrderModel(orderModel);
    orderDOMapper.insertSelective(orderDO);
    //加上商品的销量
    itemService.increaseSales(itemId, amount);

    //4.返回前端
    return orderModel;
}
```

## 7.controller层

```
//封装下单请求
@RequestMapping(value = "/createorder", method = {RequestMethod.POST}, consumes =
{CONTENT_TYPE_FORMED})
@ResponseBody
public CommonReturnType createOrder(@RequestParam(name = "itemId") Integer itemId,
                                    @RequestParam(name = "amount") Integer amount) throws
BusinessException {

    //获取用户登录信息                强转，否则没有这个值会报空指针错误
    Boolean isLogin = (Boolean) httpServletRequest.getSession().getAttribute("IS_LOGIN");
    if (isLogin == null || !isLogin.booleanValue()) {
        throw new BusinessException(EmBusinessError.USER_NOT_LOGIN, "用户还未登录，不能下单");
    }
    UserModel userModel = (UserModel) httpServletRequest.getSession().getAttribute("LOGIN_USER");


    OrderModel orderModel = orderService.createOrder(userModel.getId(), itemId, amount);

    return CommonReturnType.create(null);
}
```

# 第六章 秒杀模块开发

## 6.1 秒杀模型管理——活动模型创建

### 1.使用joda-time

**pom文件中，修改后需要refresh**

**推荐使用joda-time**

```
<dependency>
  <groupId>joda-time</groupId>
  <artifactId>joda-time</artifactId>
  <version>2.9.1</version>
</dependency>
```

### 2.创建活动模型

```
public class PromoModel {
    private Integer id;

    //秒杀活动状态：1表示还未开始，2表示正在进行，3表示已结束
    private Integer status;  便于外层系统直接调用


    //秒杀活动名称
    private String promoName;

    //秒杀活动的开始时间
    private DateTime startDate;

    //秒杀活动的结束时间
    private DateTime endDate;

    //秒杀活动的适用商品
    private Integer itemId;

    //秒杀活动的商品价格
    private BigDecimal promoItemPrice;
```

### 3.设计数据库

```
CREATE TABLE `promo`  (
  `id` int(100) NOT NULL AUTO_INCREMENT,
  `promo_name` varchar(255) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL DEFAULT '',
  `start_date` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',  mysql中的datatime是年月日时分秒的string类型
  `end_date` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',  `start_date` datetime NOT NULL DEFAULT
  `item_id` int(11) NOT NULL DEFAULT 0,                          CURRENT_TIMESTAMP,
                                                                 `end_date` datetime NOT NULL DEFAULT
  `promo_item_price` decimal(10, 2) NOT NULL DEFAULT 0.00,       CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8 COLLATE = utf8_bin ROW_FORMAT
= Compact;
```

### 4.mybatis逆向工程

```
<table tableName="promo" domainObjectName="PromoDO"
       enableCountByExample="false"
       enableUpdateByExample="false"
       enableDeleteByExample="false"
       enableSelectByExample="false"
       selectByExampleQueryId="false" ></table>
```

## 6.2 秒杀模型管理——活动模型与商品模型结合

### 1.service

秒杀服务根据商品id，查询得到当前的活动以及其价格

### PromoService

```
PromoModel getPromoByItemId(Integer itemId);
```

## PromoServiceImpl

```
@Service
public class PromoServiceImpl implements PromoService {

    @Autowired
    private PromoDOMapper promoDOMapper; 引入数据库



    //根据iremId获取即将开始的或者正在进行的活动
    @Override
    public PromoModel getPromoByItemId(Integer itemId) {

        //获取商品对应的秒杀信息
        PromoDO promoDO = promoDOMapper.selectByItemId(itemId);

        //dataobject->model 领域模型
        PromoModel promoModel = convertFromDataObject(promoDO);
        if (promoModel == null) {
            return null;
        }                                   如果生成DOMapper后需要添加字段，可以手动改（因为使用mybatis可
                                            能会冲掉被人的数据库）
                                            DOMapper：result，insert等中都要相应的改动
        //判断当前时间是否秒杀活动即将开始或正在进行
        DateTime now = new DateTime();
        if (promoModel.getStartDate().isAfterNow()) {
            promoModel.setStatus(1); 1还没开始
        } else if (promoModel.getEndDate().isBeforeNow()) {
            promoModel.setStatus(3); 3已经结束
        } else {
            promoModel.setStatus(2); 2正在进行
        }

        return promoModel;
    }

    private PromoModel convertFromDataObject(PromoDO promoDO) {
        if (promoDO == null) {
            return null;
        }
        PromoModel promoModel = new PromoModel();
        BeanUtils.copyProperties(promoDO, promoModel);         joda-time
        promoModel.setStartDate(new DateTime(promoDO.getStartDate()));
        promoModel.setEndDate(new DateTime(promoDO.getEndDate()));

        return promoModel;
    }
}
```

## 2.使用聚合模型，在ItemModel上添加属性

```
//使用聚合模型，如果promoModel不为空，则表示其拥有还未结束的秒杀活动
private PromoModel promoModel;                  包括还未开始和正在进行的
                        将秒杀活动和商品关联在一起
```

## 更改ItemServiceImpl

```
@Override
public ItemModel getItemById(Integer id) {
    ItemDO itemDO = itemDOMapper.selectByPrimaryKey(id);
    if (itemDO == null) {
        return null;
    }
    //操作获得库存数量
    ItemStockDO itemStockDO = itemStockDOMapper.selectByItemId(itemDO.getId());

    //将dataobject-> Model
    ItemModel itemModel = convertModelFromDataObject(itemDO, itemStockDO);

    //获取活动商品信息
    PromoModel promoModel = promoService.getPromoByItemId(itemModel.getId());
    if (promoModel != null && promoModel.getStatus().intValue() != 3) {
        itemModel.setPromoModel(promoModel);
    }
    return itemModel;
}
```

## 同时修改ItemVO **前端的展示**

```
//商品是否在秒杀活动中，以及对应的状态：0表示没有秒杀活动，1表示秒杀活动等待开始，2表示进行中
private Integer promoStatus;

//秒杀活动价格
private BigDecimal promoPrice;

//秒杀活动id
private Integer promoId;

//秒杀活动开始时间
private String startDate;
```

## 修改ItemController

```
private ItemVO convertVOFromModel(ItemModel itemModel) {
    if (itemModel == null) {
        return null;
    }
    ItemVO itemVO = new ItemVO();
    BeanUtils.copyProperties(itemModel, itemVO);
    if (itemModel.getPromoModel() != null) {    有正在进行或即将进行的秒杀活动
        itemVO.setPromoStatus(itemModel.getPromoModel().getStatus());
        itemVO.setPromoId(itemModel.getPromoModel().getId());
                    itemVO.setStartDate(itemModel.getPromoModel().getStartDate().
                    toString(DateTimeFormat.forPattern("yyyy-MM-dd HH:mm:ss")));
        itemVO.setPromoPrice(itemModel.getPromoModel().getPromoItemPrice());
    } else {
        itemVO.setPromoStatus(0);
    }
    return itemVO;
}
```

## 3.修改前端界面

## 4.修改OrderModel 对应的秒杀下单

### 增加秒杀价格字段

```
//若非空，则表示是以秒杀商品方式下单
private Integer promoId;

//购买时商品的单价,若promoId非空，则表示是以秒杀商品方式下单
private BigDecimal itemPrice;
```

### 然后在数据库中，DO中，DOMapper中增加此字段

promoId

## 5.改造下单接口

```
//1.通过url上传过来秒杀活动id，然后下单接口内校验对应id是否属于对应商品且活动已开始
//2.直接在下单接口内判断对应的商品是否存在秒杀活动，若存在进行中的则以秒杀价格下单
//倾向于使用第一种形式，因为对同一个商品可能存在不同的秒杀活动，而且第二种方案普通销售的商品也需要校验秒杀
OrderModel createOrder(Integer userId, Integer itemId, Integer promoId, Integer amount) throws BusinessException;
```

### 实现

```
//校验活动信息
        if (promoId != null) {
            //(1)校验对应活动是否存在这个适用商品
            if (promoId.intValue() != itemModel.getPromoModel().getId()) {
                throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "活动信息不正确");
                //(2)校验活动是否正在进行中
            } else if (itemModel.getPromoModel().getStatus() != 2) {
                throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "活动信息不正确");
            }
        }

        //2.落单减库存
        boolean result = itemService.decreaseStock(itemId, amount);
        if (!result) {
            throw new BusinessException(EmBusinessError.STOCK_NOT_ENOUGH);
        }

        //3.订单入库
        OrderModel orderModel = new OrderModel();
        orderModel.setUserId(userId);
        orderModel.setItemId(itemId);
        orderModel.setPromoId(promoId);
        orderModel.setAmount(amount);

        if (promoId != null) {
            orderModel.setItemPrice(itemModel.getPromoModel().getPromoItemPrice());
        } else {
            orderModel.setItemPrice(itemModel.getPrice());
        }

        orderModel.setOrderPrice(orderModel.getItemPrice().multiply(BigDecimal.valueOf(amount)));
```

## 在controller层添加参数

```
@RequestParam(name = "promoId",required = false) Integer promoId,
```

## 进行测试