

# 《软件设计高级实践》

## 软件部署文档



学院：智能与计算学院

专业：软件工程

年级：2018

姓名：潘柯文

学号：3018216079

姓名：林煜烜

学号：3018216074

姓名：马辰

学号：3018216106

姓名：姚毅强

学号：3018216089

# 第一章 环境配置

## 1.1 编译环境

名称	版本
jdk	1.8
mysql	5.7
tomcat	8
vue	3.0.4

## 1.2 编译平台

类型	软件
数据库	PHPstudy
后端	IDEA
前端	vscode, webstorm

## 1.3编程语言

类型	语言
数据库	SQL
后端	java
前端	htm,css,js,vue

## 1.4运行平台

类型	平台
前端	Chrome浏览器
后端	springboot

# 第二章 数据库配置

## 2.1 建库及建表

## 2.2 详细建表

### 2.2.1 用户表

用户表表格创建字段如下图所示:

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	user_id			unique	
字段 (7)					
 user_id	int(11)	否	<auto_increment>		
 user_name	varchar(64)	否	"		
 user_age	int(11)	否	0		
 user_gender	int(11)	否	0		
 user_phone	varchar(11)	否	"		
 user_worker	int(11)	否	0		
 user_login	int(11)	否	0		








### 2.2.2 文章表

文章表创建字段如下图所示:

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	article_id			unique	
字段 (5)					
 article_id	int(11)	否	<auto_increment>		
 article_title	varchar(128)	否	"		
 article_content	varchar(255)	否	"		
 article_date	varchar(255)	否	1970-01-01		
 user_id	int(11)	否	0		





### 2.2.3. 文章标签表

文章标签表创建字段如下图所示:

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	articletag_id			unique	
字段 (8)					
 articletag_id	int(11)	否	<auto_increment>		
 article_id	int(11)	否	0		
 article_title	varchar(128)	否	"		
 article_content	varchar(255)	否	"		
 article_date	varchar(255)	否	1970-01-01		
 tag_id	int(11)	否	0		
 tag_name	varchar(255)	否	"		
 user_id	int(11)	否	0		

### 2.2.4. 用户密码表

用户密码表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	usercode_id			unique	
字段 (3)					
 usercode_id	int(11)	否	<auto_increment>		
 user_password	varchar(255)	否	"		
 user_id	int(11)	否	0		



### 2.2.5. 文章点赞表

文章点赞表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	articlelike_id			unique	
字段 (3)					
 <b>articlelike_id</b>	int(11)	否	<auto_increment>		
 article_id	int(11)	否	0		
 user_id	int(11)	否	0		

## 2.2.6. 公司表

公司表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	company_id			unique	
字段 (2)					
 <b>company_id</b>	int(11)	否	<auto_increment>		
 company_intro	varchar(255)	否	"		

## 2.2.7. 评论表

评论表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	comment_id			unique	
字段 (5)					
 <b>comment_id</b>	int(11)	否	<auto_increment>		
 comment_content	varchar(255)	否	"		
 comment_date	varchar(255)	否	1970-01-01		
 user_id	int(11)	否	0		
 article_id	int(11)	否	0		




## 2.2.8. 商品表

商品表创建字段如下图所示

Field Types				
#	Field	Schema	Table	Type
1	id	shop	goods	INT
2	name	shop	goods	VARCHAR
3	lib	shop	goods	VARCHAR
4	region	shop	goods	INT
5	state	shop	goods	VARCHAR
6	price	shop	goods	INT
7	description	shop	goods	VARCHAR
8	actualPrice	shop	goods	INT

2.2.9. 标签表

标签表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	tag_id			unique	
字段 (2)					
 tag_id	int(11)	否	<auto_increment>		
 tag_name	varchar(128)	否	"		

2.2.10. 订单表

订单表创建字段如下图所示

Field Types				
#	Field	Schema	Table	Type
1	userId	shop	order	INT
2	id	shop	order	INT
3	price	shop	order	INT
4	goodsId	shop	order	INT
5	date	shop	order	VARCHAR
6	actualPrice	shop	order	INT
7	totalPrice	shop	order	INT
8	num	shop	order	INT

2.2.11. 公司领导表

公司领导表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	leader_id			unique	
字段 (4)					
 leader_id	int(11)	否	<auto_increment>		
 leader_name	varchar(64)	否	"		
 leader_post	varchar(128)	否	"		
 leader_intro	varchar(255)	否	"		

## 2.2.12. 报告表

报告表创建字段如下图所示

名称	类型	空	默认值	属性	备.
索引 (1)					
 主索引	report_id			unique	
字段 (4)					
 report_id	int(11)	否	<auto_increment>		
 report_title	varchar(128)	否	"		
 report_content	varchar(255)	否	"		
 report_date	varchar(255)	否	1970-01-01		

# 第三章 后端配置

## 3.1 IDEA 项目创建

1. 创建maven项目，SDK选用1.8，并选择Create from archetype绑定默认配置文件。
2. 将项目命名为school，并安装在同名目录下，完成新项目创建

## 3.2 添加SpringBoot 依赖

修改pom.xml文件

增加如下代码:

```
<parent>
<groupId>org.springframework.boot</ groupId>  <artifactId>spring-boot-starter -parent</ artifactId> <version>2.0.
</ parent>

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

## 3.3 接入MyBatis

### 3.3.1 添加依赖

在pom.xml中添加如下依赖：



```

<!--mysql jdbc 配置-->
<dependency>
    <groupId>mysql</ groupId> <artifactId>mysql-connector-java</ artifactId> <version>5.1.41</ version>
</ dependency>
<!--阿里巴巴德鲁伊连接池-->
<dependency>
<groupId>com. alibaba</ groupId>
<artifactId>druid</ artifactId>
<version>1.1.3</ version>
</ dependency> 13 <!--spring boot 对 mybatis 的支持-->
<dependency>
    <groupId>org.mybatis.spring.boot</ groupId> <artifactId>mybatis-spring-boot-starter</ artifactId>
    <version>1.3.1</ version>
</ dependency>
<!--mybatis 自动生成工具用来生成对应的数据库文件的映射-->
<plugin>
    <groupId>org.mybatis.generator</ groupId> <artifactId>mybatis-generator-maven-plugin</ artifactId>
    <version>1.3.5</ version>
    <dependencies>
    <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>mybaits 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>
    <!--允许文件自动覆盖, 实际开发中千万不要设置为 true-->
    <overwrite>true</ overwrite> 52 <!--generator 配置文件路径-->
    <configurationFile>
        src / main / resources / mybatis-generator .xml
    </ configurationFile>
    </ configuration>
    </ plugin>

```

### 3.3.2 配置数据源

在resources文件夹中新建application.properties，并添加如下配置：

```
mybatis.mapper-locations=classpath:mapping/*.xml
spring.datasource.name=seckill
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/seckill
spring.datasource.username=root
spring.datasource.password=root

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

### 3.3.3 绑定mybaits

添加配置文件mybatis-generator.xml，写入如下代码：

```

<?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">
<!--http://www.mybatis.org/generator/configreference/xmlconfig.html--> <generatorConfiguration>
<context id="MySQLTables" targetRuntime="MyBatis3">
    <jdbcConnection driverClass="com.mysql.jdbc.Driver" 12 connectionURL="jdbc:mysql://127.0.0.1:3306/school"
        userId="root" 14 password="root">
    </jdbcConnection>
<!--生成 DataObject 类存放位置-->
<javaModelGenerator targetPackage="com.schoolproject.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-->
<!--客户端代码, 生成易于使用的针对 Model 对象和 XML 配置文件的代码-->
<!--type="ANNOTATEDMAPPER", 生成 Java Model 和基于注解的 Mapper 对象-->
<!--type="MIXEDMAPPER", 生成基于注解的 Java Model 和相应的 Mapper 对象-->
<!--type="XMLMAPPER", 生成 SQLMap XML 文件和独立的 Mapper 接口-->
    <javaClientGenerator type="XMLMAPPER" targetPackage="com.schoolproject.dao" targetProject="src/main/java">
    <property name="enableSubPackages" value="true" />
    </javaClientGenerator>
<!--生成对应表及类名-->
<!--*Example 禁用自动生成的复杂查询-->
<table tableName="user" domainObjectName="UserDO">
    enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false"
</table> 43 <table tableName="lecture" domainObjectName="LectureDO" enableCountByExample="false" enableUpdateByExample="false"
</table>
<table tableName="problem" domainObjectName="ProblemDO">
    enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false"
</table>
<table tableName="userlec" domainObjectName="UserlecDO">
    enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false"
</table>
<table tableName="userpro" domainObjectName="UserproDO">
    enableCountByExample="false" enableUpdateByExample="false"
    enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false">
</table>
<table tableName="consult" domainObjectName="ConsultDO">
    enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false"
</table>
<table tableName="ask" domainObjectName="AskDO" 64 enableCountByExample="false" enableUpdateByExample="false"
    enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false">
</table>
<table tableName="admin" domainObjectName="AdminDO">
    enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false"
</table>

```

```

<table tableName=" secret " domainObjectName=" SecretDO"
    enableCountByExample=" false " enableUpdateByExample=" false "
    enableDeleteByExample=" false " enableSelectByExample=" false " selectByExampleQueryId=" false ">
</ table>
</ context>
</ generatorConfiguration>

```

### 3.3.4 新建Maven命令

1.Run -> Edit Configurations -> 新增 -> Maven:

(1)Name: mybatis-generator

(2)Command line: mybatis-generator: generate

2.在resources中新建mapping文件夹，用于存储通过mybatis与数据库交互的SQL语句。

3.在java文件夹中的com.schoolproject中新建dao和dataobject两个文件夹， 分别存储与数据库交互的SQL命令函数和属性及其setter getter方法。

4.运行'mybatis-generator'， 完成从数据库传输数据表信息到后端功能

## 3.4 启动后端项目

修改App.java如下， 运行App

```

package com.star;

import com.star.dao.UserDOMapper;
import com.star.dataobject.UserDO;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication(scanBasePackages = {"com.star"})
@RestController
@MapperScan("com.star.dao")
public class App {
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );

        SpringApplication.run(App.class, args);
    }
}

```

在浏览器输入：“localhost : 8080”，完成从后端到前端JSON数据的传输

## 3.5搭建完整后端项目

以用户管理模块为例从底层与数据库对接到顶层与前端交互逐步搭建完整 后端项目。

### 1. DAO层

DAO层主要是和数据库交互，其使用的实体类主要是DataObject。该层数据在运行mybatis操作时会有

端调取已建立的数据库表格自动生成。

### 2. Service层

Service层主要处理后端的核心业务处理逻辑，其使用的实体类主要是ModelObject。

首先在com.schoolproject下新建service文件夹，新建UserService.java的接口，编写如下代码：

```
package com.star.service;

import com.star.error.BusinessException;
import com.star.service.model.UserModel;

public interface UserService {
    UserModel getUserById(Integer id);
    void register(UserModel userModel) throws BusinessException;
    UserModel login(String userPhone, String userPassword) throws BusinessException;
    void updateWorker(Integer userWorker, Integer id);
    void updateLogin(Integer userLogin, String userPhone);
    void updatePassword(String userPssword, Integer id);
}
```

同时在service文件夹下新建impl和model两个文件夹，分别存储该service接口的具体实现以及所需的实体类。

在impl中新建UserServiceImpl.java类，继承自接口用于实现具体方法，代码如下：

```

package com.star.service.impl;

import com.alibaba.druid.util.StringUtils;
import com.star.dao.UserCodeDOMapper;
import com.star.dao.UserDOMapper;
import com.star.dataobject.UserCodeDO;
import com.star.dataobject.UserDO;
import com.star.error.BusinessException;
import com.star.error.EmBusinessError;
import com.star.service.UserService;
import com.star.service.model.UserModel;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DuplicateKeyException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDOMapper userDOMapper;

    @Autowired
    private UserCodeDOMapper userCodeDOMapper;

    @Override
    public UserModel getUserById(Integer id){
        UserDO userDO = userDOMapper.selectByPrimaryKey(id);

        if(userDO == null){
            return null;
        }

        UserCodeDO userCodeDO = userCodeDOMapper.selectByUserId(userDO.getUserId());

        return convertFromDataObject(userDO, userCodeDO);
    }

    @Override
    @Transactional // 事务操作
    public void register(UserModel userModel) throws BusinessException {
        if(userModel == null){
            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
        }

        if(StringUtils.isEmpty(userModel.getUserPhone())
            || StringUtils.isEmpty(userModel.getUserName())
            || userModel.getUserGender() == null
            || userModel.getUserAge() == null
            || StringUtils.isEmpty(userModel.getUserPassword())){

```

```

        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
    }

    UserDO userDO = convertFromModel(userModel);
    try{
        userDOMapper.insertSelective(userDO);
    }catch (DuplicateKeyException ex){
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "手机号码已注册");
    }

    userModel.setUserId(userDO.getUserId()); // 获取自增 id

    UserCodeDO userCodeDO = convertCodeFromModel(userModel);
    userCodeDOMapper.insertSelective(userCodeDO);

    return;
}

@Override
public UserModel login(String userPhone, String userPassword) throws BusinessException {
    UserDO userDO = userDOMapper.selectByUserPhone(userPhone);
    if(userDO == null){
        throw new BusinessException(EmBusinessError.USER_LOGIN_FAIL);
    }
    UserCodeDO userCodeDO = userCodeDOMapper.selectByUserId(userDO.getUserId());
    UserModel userModel = convertFromDataObject(userDO, userCodeDO);
    if(!StringUtils.equals(userPassword, userModel.getUserPassword())){
        throw new BusinessException(EmBusinessError.USER_LOGIN_FAIL);
    }
    return userModel;
}

@Override
public void updateWorker(Integer userWorker, Integer userId){
    userDOMapper.updateWorker(userWorker, userId);
}

@Override
public void updateLogin(Integer userLogin, String userPhone){
    userDOMapper.updateLogin(userLogin, userPhone);
}

@Override
public void updatePassword(String userPassword, Integer userId){
    userCodeDOMapper.updatePassword(userPassword, userId);
}

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

```

```

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

        if(userCodeDO != null){
            userModel.setUserPassword(userCodeDO.getUserPassword());
        }
        return userModel;
    }

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

    private UserCodeDO convertCodeFromModel(UserModel userModel){
        if(userModel == null){
            return null;
        }
        UserCodeDO userCodeDO = new UserCodeDO();
        userCodeDO.setUserPassword(userModel.getUserPassword());
        userCodeDO.setUserId(userModel.getUserId());
        return userCodeDO;
    }
}

```

在model文件夹中新建userModel.java实体类，拷贝来自于userDO.java中的 属性及其setter getter方法，代码如下：



```
package com.star.service.model;

public class UserModel {
    private Integer userId;
    private String userName;
    private Integer userAge;
    private Integer userGender;
    private String userPhone;
    private Integer userWorker;
    private Integer userLogin;

    private String userPassword;

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName == null ? null : userName.trim();
    }

    public Integer getUserAge() {
        return userAge;
    }

    public void setUserAge(Integer userAge) {
        this.userAge = userAge;
    }

    public Integer getUserGender() {
        return userGender;
    }

    public void setUserGender(Integer userGender) {
        this.userGender = userGender;
    }

    public String getUserPhone() {
        return userPhone;
    }

    public void setUserPhone(String userPhone) {
        this.userPhone = userPhone == null ? null : userPhone.trim();
    }
}
```

```

}

public Integer getUserWorker() {
    return userWorker;
}

public void setUserWorker(Integer userWorker) {
    this.userWorker = userWorker;
}

public Integer getUserLogin() {
    return userLogin;
}

public void setUserLogin(Integer userLogin) {
    this.userLogin = userLogin;
}

public String getUserPassword() {
    return userPassword;
}

public void setUserPassword(String userPassword) {
    this.userPassword = userPassword == null ? null : userPassword.trim();
}
}

```

### 3. Controller层

Controller层主要处理与前端交互的接口，封装好Service层的方法方便前端的使用。由于需要用视图屏蔽部分前端用户无法查看的数据库信息，因此其使用的实体类主要是ViewObject。

首先在project文件夹下新建controller文件夹，并新建BaseController.java类，用于处理异常，代码如下：

```

package com.star.controller;

import com.star.error.BusinessException;
import com.star.error.EmBusinessError;
import com.star.response.CommonReturnTypes;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

import javax.servlet.http.HttpServletRequest;
import java.util.HashMap;
import java.util.Map;

public class BaseController {
    public static final String CONTENT_TYPE_FORMED = "application/x-www-form-urlencoded";
    // 定义 exceptionhandler 解决未被 controller 层吸收的 exception
    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.OK)
    @ResponseBody
    public Object handlerException(HttpServletRequest request, Exception ex){

        Map<String, Object> responseData = new HashMap<>();

        if(ex instanceof 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 CommonReturnTypes.create(responseData, "fail");
    }
}

```

其次在controller文件夹中新建UserController.java类继承自BaseController.java以便于和前端交互时的json数据有通用的异常处理，代码如下：

```

package com.star.controller;

import com.alibaba.druid.util.StringUtils;
import com.star.controller.viewobject.UserVO;
import com.star.error.BusinessException;
import com.star.error.EmBusinessError;
import com.star.response.CommonReturnTypes;
import com.star.service.UserService;
import com.star.service.model.UserModel;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import sun.misc.BASE64Encoder;

import javax.servlet.http.HttpServletRequest;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;

@Controller("user")
@RequestMapping("/user")
@CrossOrigin(allowCredentials = "true", allowedHeaders = "*")
public class UserController extends BaseController{

    @Autowired
    private UserService userService;

    @Autowired
    private HttpServletRequest httpServletRequest;

    @RequestMapping("/get")
    @ResponseBody
    public CommonReturnTypes getUser(@RequestParam(name="userId") Integer userId) throws BusinessException{
        UserModel userModel = userService.getUserById(userId);

        if(userModel == null){
            throw new BusinessException(EmBusinessError.USER_NOT_EXIST);
        }
        UserVO userVO = convertFromModel(userModel);
        return CommonReturnTypes.create(userVO);
    }

    //生成验证码
    @RequestMapping(value = "/getotp", method = {RequestMethod.POST})
    @ResponseBody
    public CommonReturnTypes getOtp(@RequestParam(name="userPhone") String userPhone){
        // 生成 otp 验证码
        Random random = new Random();
        int randomInt = random.nextInt(99999);
    }

```

```

        randomInt += 10000;
        String otpCode = String.valueOf(randomInt);

        // 将 otp 验证码同对应的用户关联 (暂时使用 httpsession 的方式绑定 otp 与手机号)
        httpRequest.getSession().setAttribute(userPhone, otpCode);

        // 将 otp 验证码通过短信通道发送给用户 (省略, 使用控制台输出代替)
        System.out.println(String.format("userPhone = %s & otpCode = %s", userPhone, otpCode));

        return CommonReturnTypes.create(null);
    }

    //用户注册
    @RequestMapping(value = "/register", method = {RequestMethod.POST})
    @ResponseBody
    public CommonReturnTypes register(@RequestParam(name="userName") String userName,
                                      @RequestParam(name="userGender") Integer userGender,
                                      @RequestParam(name="userAge") Integer userAge,
                                      @RequestParam(name="userPassword") String userPassword,
                                      @RequestParam(name="userPhone") String userPhone,
                                      @RequestParam(name="otpCode") String otpCode)
        throws BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {

        String inSessionOtpCode = (String)this.httpServletRequest.getSession().getAttribute(userPhone);
        if(!StringUtils.equals(otpCode, inSessionOtpCode)){
            throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR, "短信验证码不正确");
        }

        UserModel userModel = new UserModel();
        userModel.setUserName(userName);
        userModel.setUserAge(userAge);
        userModel.setUserGender(new Integer(String.valueOf(userGender)));
        userModel.setUserPassword(EncodeByMd5(userPassword));
        userModel.setUserPhone(userPhone);
        userService.register(userModel);

        return CommonReturnTypes.create(null);
    }

    //MD5加密
    private String EncodeByMd5(String str) throws UnsupportedEncodingException, NoSuchAlgorithmException {
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        BASE64Encoder base64Encoder = new BASE64Encoder();
        String newstr = base64Encoder.encode(md5.digest(str.getBytes("utf-8")));
        return newstr;
    }

    //用户登录
    @RequestMapping(value = "/login", method = {RequestMethod.POST}, consumes = {CONTENT_TYPE_FORMED})
    @ResponseBody
    public CommonReturnTypes login(@RequestParam(name="userPhone") String userPhone,

```

```

        @RequestParam(name="userPassword") String userPassword)
        throws BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {

    if(StringUtils.isEmpty(userPhone) || StringUtils.isEmpty(userPassword)){
        throw new BusinessException(EmBusinessError.PARAMETER_VALIDATION_ERROR);
    }

    UserModel userModel = userService.login(userPhone, EncodeByMd5(userPassword));
    userService.updateLogin(1,userPhone);
    httpRequest.getSession().setAttribute("LOGIN", true);
    httpRequest.getSession().setAttribute("LOGIN_USER", userModel);

    return CommonReturnType.create(null);
}

//用户登出
@RequestMapping(value = "/logout", method = {RequestMethod.POST}, consumes = {CONTENT_TYPE_FORMED})
@ResponseBody
public String logout(@RequestParam(name="userPhone") String userPhone)
        throws BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {
    userService.updateLogin(0,userPhone);
    return "success";
}

//修改角色权限位
@RequestMapping(value = "/worker", method = {RequestMethod.POST}, consumes = {CONTENT_TYPE_FORMED})
@ResponseBody
public String changeWorker(@RequestParam(name="userId") Integer userId,
        @RequestParam(name="userWorker") Integer userWorker)
        throws BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {
    userService.updateWorker(userWorker,userId);
    return "success";
}

//修改密码
@RequestMapping(value = "/password", method = {RequestMethod.POST}, consumes = {CONTENT_TYPE_FORMED})
@ResponseBody
public String changePassword(@RequestParam(name="userId") Integer userId,
        @RequestParam(name="userPassword") String userPassword)
        throws BusinessException, UnsupportedEncodingException, NoSuchAlgorithmException {
    userService.updatePassword(userPassword,userId);
    return "success";
}

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

```

```
}  
}
```

然后在controller文件夹下新建viewobject文件夹存储其使用的实体类，并在 其中新建UserVO.java，主要拷贝自UserModel.java，但是屏蔽部分前端用户无法查看的重要信息，代码如下：

```
package com.star.controller.viewobject;
```

```
public class UserVO {  
    private Integer userId;  
    private String userName;  
    private Integer userAge;  
    private Integer userGender;  
    private String userPhone;  
    private Integer userWorker;  
    private Integer userLogin;  
  
    public Integer getUserId() {  
        return userId;  
    }  
  
    public void setUserId(Integer userId) {  
        this.userId = userId;  
    }  
  
    public String getUserName() {  
        return userName;  
    }  
  
    public void setUserName(String userName) {  
        this.userName = userName == null ? null : userName.trim();  
    }  
  
    public Integer getUserAge() {  
        return userAge;  
    }  
  
    public void setUserAge(Integer userAge) {  
        this.userAge = userAge;  
    }  
  
    public Integer getUserGender() {  
        return userGender;  
    }  
  
    public void setUserGender(Integer userGender) {  
        this.userGender = userGender;  
    }  
  
    public String getUserPhone() {  
        return userPhone;  
    }  
  
    public void setUserPhone(String userPhone) {  
        this.userPhone = userPhone == null ? null : userPhone.trim();  
    }  
}
```



```
public Integer getUserWorker() {  
    return userWorker;  
}  
  
public void setUserWorker(Integer userWorker) {  
    this.userWorker = userWorker;  
}  
  
public Integer getUserLogin() {  
    return userLogin;  
}  
  
public void setUserLogin(Integer userLogin) {  
    this.userLogin = userLogin;  
}  
}
```

#### 4. 返回通用返回对象

在com.star下新建response文件夹，并在其中新建CommonReturnType.java类，创建并封装json所需要的data和status，用于规约前后端交互时的json数据格式，代码如下：

```

package com.star.response;

public class CommonReturnType {

    // success, fail
    private String status;

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

    public static CommonReturnType create(Object result){

        return create(result, "success");
    }

    public static CommonReturnType create(Object result, String status){

        CommonReturnType type = new CommonReturnType();
        type.setStatus(status);
        type.setData(result);

        return type;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Object getData() {
        return data;
    }

    public void setData(Object data) {
        this.data = data;
    }
}

```

## 5. 异常处理

(1) 在com.star下新建error文件夹，并在其中新建CommonError.java类，定义错误接口，代码如下：

```
package com.star.error;

public interface CommonError {
    public int getErrCode();
    public String getErrMsg();
    public CommonError setErrMsg(String errMsg);
}
```

(2) 新建EmBusinessError.java继承实现CommonError接口，用于定义错误类型枚举，代码如下

```

package com.star.error;

public enum EmBusinessError implements CommonError {
    // 通用错误类型 10001
    PARAMETER_VALIDATION_ERROR(10001, "参数不合法"),
    UNKNOWN_ERROR(10002, "未知错误"),

    // 20000 开头为用户信息相关错误定义
    USER_NOT_EXIST(20001, "用户不存在"),
    USER_LOGIN_FAIL(20002, "登录失败"),
    ARTICLE_NOT_EXIST(20003, "帖子不存在"),
    TAG_NOT_EXIST(20004, "标签不存在"),
    ARTICLETAG_NOT_EXIST(20005, "关系不存在"),
    COMMENT_NOT_EXIST(20006, "评论不存在"),
    ARTICLELIKE_NOT_EXIST(20007, "点赞不存在"),
    COMPANY_NOT_EXIST(20008, "公司简介不存在"),
    LEADER_NOT_EXIST(20009, "领导不存在"),
    REPORT_NOT_EXIST(20010, "报告不存在"),
    ;

    private int errCode;
    private String errMsg;

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

```

(3)新建BusinessException.java继承Exception并实现CommonError，作为包 装器业务异常类的实现，代码如下：

```
package com.star.error;

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

    private CommonError commonError;

    public BusinessException(CommonError commonError){
        super();
        this.commonError = commonError;
    }

    public BusinessException(CommonError commonError, String errMsg){
        super();
        this.commonError = commonError;
        this.commonError.setErrMsg(errMsg);
    }

    @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;
    }
}
```

## 3.6 其他模块实现

对于管理员管理模块、商城管理模块、论坛管理模块、点赞模块、评论模块、标签模块、公司管理模块、首页模块、个人中心模块一致。将代码依次按如下图目录树拷贝进项目，完成项目后端的搭建。

star > src > main > java > com > star > App

Project

star E:\javawork\star

.idea

src

main

java

com.star

controller

viewobject

ArticlelikeVO

ArticletagVO

ArticleVO

CommentVO

CompanyVO

LeaderVO

ReportVO

TagVO

UserVO

ArticleController

ArticlelikeController

ArticletagController

BaseController

CommentController

CompanyController

LeaderController

ReportController

TagController

UserController

dao

UserDO.java

```
1      package
2
3      import
11
12      @Spring
13      @RestC
14      @Mappe
15      public
16
17      pu
18      {
19
20
21
22      }
23      }
```

star > src > main > java > com > star > App

Project

ReportController

TagController

UserController

dao

ArticleDOMapper

ArticlelikeDOMapper

ArticletagDOMapper

CommentDOMapper

CompanyDOMapper

LeaderDOMapper

ReportDOMapper

TagDOMapper

UserCodeDOMapper

UserDOMapper

dataobject

ArticleDO

ArticlelikeDO

ArticletagDO

CommentDO

CompanyDO

LeaderDO

ReportDO

TagDO

UserCodeDO

UserDO

error

BusinessException

CommonError

EmBusinessError

response

CommonReturnType

service

impl

UserDO.java

UserVO.java

1 package com.st

2

3 import ...

11

12 @SpringBootApp

13 @RestController

14 @MapperScan("c

15 public class A

16

17 public sta

18 {

19 System

20

21 Spring

22 }

23 }

star > src > main > java > com > star > App

Project

CommonError

EmBusinessError

response

CommonReturnType

service

impl

ArticlelikeServiceImpl

ArticleServiceImpl

ArticletagServiceImpl

CommentServiceImpl

CompanyServiceImpl

LeaderServiceImpl

ReportServiceImpl

TagServiceImpl

UserServiceImpl

model

ArticlelikeModel

ArticleModel

ArticletagModel

CommentModel

CompanyModel

LeaderModel

ReportModel

TagModel

UserModel

ArticlelikeService

ArticleService

ArticletagService

CommentService

CompanyService

LeaderService

ReportService

TagService

UserService

UserDO.java

```

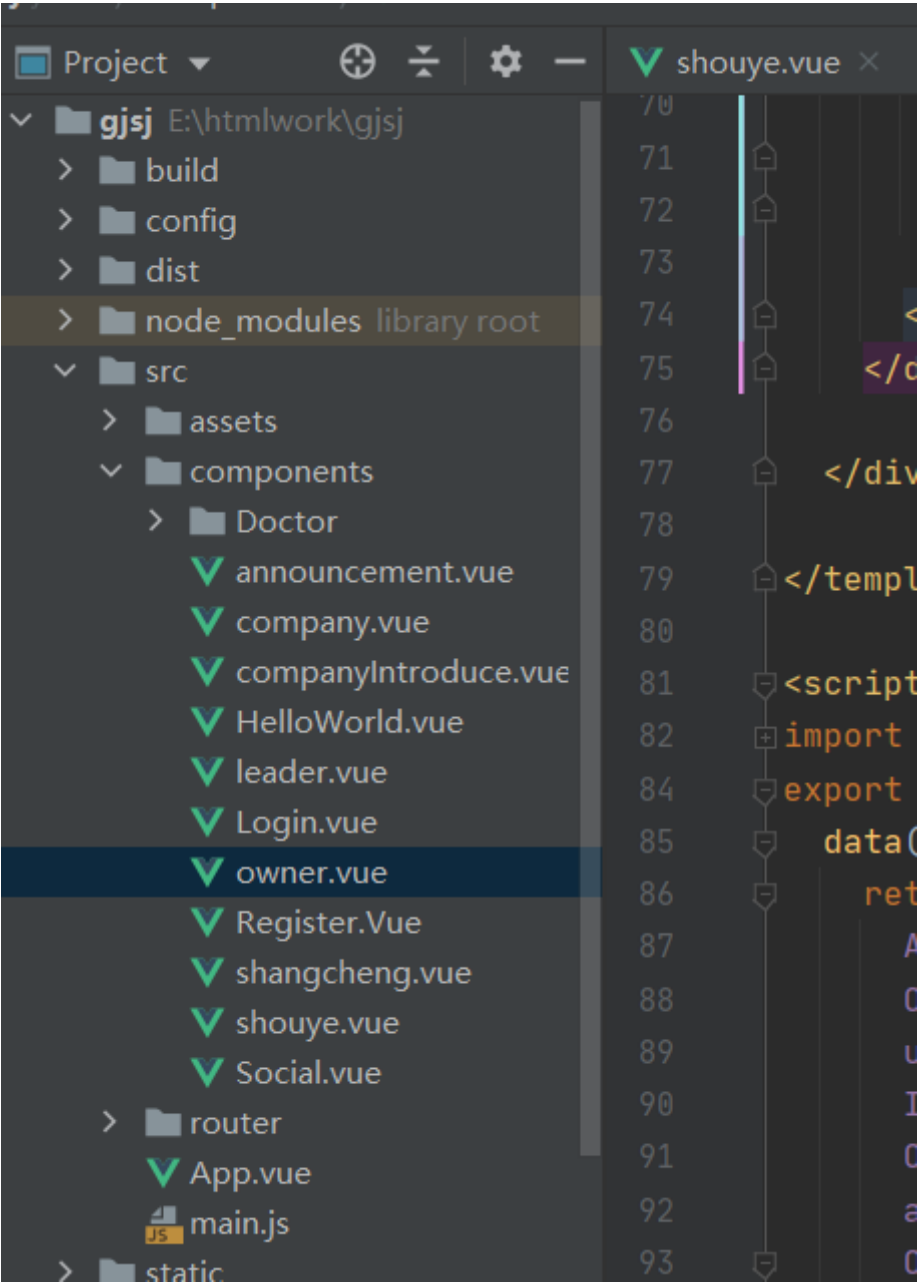
1      package
2
3      import
11
12      @Spring
13      @RestC
14      @Mappe
15      public
16
17      pu
18      {
19
20
21
22      }
23      }
```



# 第四章 前端配置

## 4.1 项目创建

在vscode中新建项目，按如下目录将所有前端页面的html文件创建在同一文 件夹下。



## 4.2 搭建环境

1. 安装 node.js
2. 运行cmd,打开命令行
3. 安装cnpm

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
npm config set registry https://registry.npm.taobao.org
```

#### 4. 安装vue cli

```
cnpm install -g vue-vli
```

## 4.3 搭建项目

### 1. 第一步

vue 项目的结伴搭建命令如下：

输入：vue init webpack 项目名称

```
vue init webpack star
```

### 2. 初始化时简单配置

```
$ vue init star
```

```
# 为你的项目起个名字
```

```
<p class="mume-header " id="为你的项目起个名字"></p>
```

```
? Project name vue-web-app
```

```
# 起你的项目写一段描述
```

```
<p class="mume-header " id="起你的项目写一段描述"></p>
```

```
? Project description A guide for vue-web-app
```

```
# 作者
```

```
<p class="mume-header " id="作者"></p>
```

```
? Author jackson影琪 <*****.com>
```

```
# 选择vue种类, 第一种是运行时编译, 第二种是只运行, 建议选后者将编译交给webpack并且体积要小大约30%
```

```
<p class="mume-header " id="选择vue种类第一种是运行时编译第二种是只运行建议选后者将编译交给webpack并且体积要小大约30"></p>
```

```
? Vue build (Use arrow keys)
```

```
> Runtime + Compiler: recommended for most users
```

```
Runtime-only: about 6KB lighter min+gzip, but templates (or any Vue-specific HTML) are ONLY allowed in .vue files - render functions are required elsewhere
```

```
# 是否后编译
```

```
<p class="mume-header " id="是否后编译"></p>
```

```
? Use post-compile? Yes
```

```
# 按需引入组件还是全部引入
```

```
<p class="mume-header " id="按需引入组件还是全部引入"></p>
```

```
? Import type Partly
```

```
# 是否自定义主题, 使用后编译的情况下可用
```

```
<p class="mume-header " id="是否自定义主题使用后编译的情况下可用"></p>
```

```
? Custom theme? Yes
```

```
# rem 布局, 使用后编译的情况下可用
```

```
<p class="mume-header " id="rem-布局使用后编译的情况下可用"></p>
```

```
? Use rem layout? No
```

```
# 是否安装vue-router
```

```
<p class="mume-header " id="是否安装vue-router"></p>
```

```
? Install vue-router? Yes
```

```
# 是否用ESLint来规范你的代码
```

```
<p class="mume-header " id="是否用eslint来规范你的代码"></p>
```

```
? Use ESLint to lint your code? Yes
```

```
# 选择一个ESLint预设标准
```

```
<p class="mume-header " id="选择一个eslint预设标准"></p>
```

```
? Pick an ESLint preset Standard No
```

```
# 是否建立单元测试
```

```
<p class="mume-header " id="是否建立单元测试"></p>
```

? Set up unit tests Yes

# 是否建立端对端测试

<p class="mume-header " id="是否建立端对端测试"></p>

? Setup e2e tests with Nightwatch? No

### 3. 安装包并运行

1 # 安装依赖

2 \$ cnpm install

3 # 在本地的8080端口起一个有热刷新功能的服务

4 \$ npm start/npm run dev

## 4.4项目配置

### 1. 代码检测规范

#### (1) 安装配置文件中的依赖包

- eslint
- babel-eslint
- eslint-plugin-html
- eslint-config-standard
- eslint-plugin-standard
- eslint-plugin-promise

通过 npm install(package) --save-dev 来配置到开发环境中。

#### (2)配置.eslintrc文件

```

module.exports =
{
// 默认情况下，ESLint会在所有父级组件中寻找配置文件，一直到根目录。ESLint一旦发现配置文件中存在 "root": true，它就会停止
root: true,
// 对Babel解析器的包装使其与 ESLint 兼容。
parser: 'babel-eslint',
parserOptions: {
  // 代码是 ECMAScript 模块
  sourceType: 'module'
},
env: {
  // 预定义的全局变量，这里是浏览器环境
  browser: true,
},
// 扩展一个流行的风格指南，即 eslint-config-standard
// https://github.com/feross/standard/blob/master/RULES.md#javascript-standard-style
extends: 'vue',
// required to lint *.vue files
plugins: [
  // 此插件用来识别.html 和 .vue文件中的js代码
  'html',
  // standard风格的依赖包
  "standard",
  // standard风格的依赖包
  "promise"
],
// add your custom rules here
'rules': {
  // allow paren-less arrow functions
  'arrow-parens': 0,
  // allow async-await
  'generator-star-spacing': 0,
  // allow debugger during development
  'no-debugger': process.env.NODE_ENV === 'production' ? 2 : 0,
  "semi": [0], // 语句可以不需要分号结尾
  "no-unused-vars": [0],
  "eqeqeq": [0],
  "array-callback-return": [0],
  "quotes": [0], // 引号风格
  "spaced-comment": [0],
  'comma-spacing': [0],
  'space-before-function-paren': [0],
  'eol-last': [0],
  'space-infix-ops': [0],
  "indent": 0, // 强制一致的缩进风格
  // "key-spacing": [1, { // 对象字面量中冒号的前后空格
  //   "beforeColon": false,
  //   "afterColon": true
  // }],
  "key-spacing": [0],
  "no-trailing-spaces": [0], // 一行最后不允许有空格

```

```
'space-before-blocks': [0],//[2, "always"], //块前的空格
'keyword-spacing': [0], //关键字前后的空格
'object-curly-spacing': [0],
'arrow-spacing': [0], //关键字前后的空格
'comma-dangle': [0],//[2, "never"], // 对象字面量项尾不能有逗号
'prefer-const': [0],//
'padded-blocks': [0],//[2, "never"], //块内行首行尾是否空行
'no-multi-spaces': [0],// 不能用多余的空格
'no-unneeded-ternary': [0],
"no-multiple-empty-lines": [0],//[2, {"max": 2}], //空行最多不能超过两行
'block-spacing': [0],
'brace-style': 2,//大括号风格
"no-else-return": 1, // 如果if语句里面有return,后面不能跟else语句
}
}
```

### (3)常用的规则

```

'rules': {
  "comma-dangle": ["error", "never"], //是否允许对象中出现结尾逗号
  "no-cond-assign": 2, //条件语句的条件中不允许出现赋值运算符
  "no-console": 2, //不允许出现console语句
  "no-constant-condition": 2, //条件语句的条件中不允许出现恒定不变的量
  "no-control-regex": 2, //正则表达式中不允许出现控制字符
  "no-debugger": 2, //不允许出现debugger语句
  "no-dupe-args": 2, //函数定义的时候不允许出现重复的参数
  "no-dupe-keys": 2, //对象中不允许出现重复的键
  "no-duplicate-case": 2, //switch语句中不允许出现重复的case标签
  "no-empty": 2, //不允许出现空的代码块
  "no-empty-character-class": 2, //正则表达式中不允许出现空的字符组
  "no-ex-assign": 2, //在try catch语句中不允许重新分配异常变量
  "no-extra-boolean-cast": 2, //不允许出现不必要的布尔值转换
  "no-extra-parens": 0, //不允许出现不必要的圆括号
  "no-extra-semi": 2, //不允许出现不必要的分号
  "no-func-assign": 2, //不允许重新分配函数声明
  "no-inner-declarations": ["error", "functions"], //不允许在嵌套代码块里声明函数
  "no-invalid-regexp": 2, //不允许在RegExp构造函数里出现无效的正则表达式
  "no-irregular-whitespace": 2, //不允许出现不规则的空格
  "no-negated-in-lhs": 2, //不允许在in表达式语句中对最左边的运算数使用取反操作
  "no-obj-calls": 2, //不允许把全局对象属性当做函数来调用
  "no-regex-spaces": 2, //正则表达式中不允许出现多个连续空格
  "quote-props": 2, //对象中的属性名是否需要用引号引起来
  "no-sparse-arrays": 2, //数组中不允许出现空位置
  "no-unreachable": 2, //在return, throw, continue, break语句后不允许出现不可能到达的语句
  "use-isnan": 2, //要求检查NaN的时候使用isNaN()
  "valid-jsdoc": ["error", {
    "requireReturn": false,
    "requireParamDescription": false,
    "requireReturnDescription": true
  }], //强制JSDoc注释
  "valid-typeof": ["error", {
    "requireStringLiterals": true
  }], //在使用typeof表达式比较的时候强制使用有效的字符串
  "block-scoped-var": 2, //将变量声明放在合适的代码块里
  "complexity": 0, //限制条件语句的复杂度
  "consistent-return": 2, //无论有没有返回值都强制要求return语句返回一个值
  "curly": ["error", "all"], //强制使用花括号的风格
  "default-case": 0, //在switch语句中需要有default语句
  "dot-notation": ["error", {"allowKeywords": false, "allowPattern": ""}], //获取对象属性的时候使用点号
  "eqeqeq": ["error", "smart"], //比较的时候使用严格等于
  "no-alert": 1, //不允许使用alert, confirm, prompt语句
  "no-caller": 2, //不允许使用arguments.callee和arguments.caller属性
  "guard-for-in": 0, //监视for in循环, 防止出现不可预料的情况
  "no-div-regex": 2, //不能使用看起来像除法的正则表达式
  "no-else-return": 0, //如果if语句有return, else里的return不用放在else里
  "no-labels": ["error", {
    "allowLoop": false,
    "allowSwitch": false
  }], //不允许标签语句

```

```
"no-eq-null": 2, //不允许对null用==或者!=
"no-eval": 2, //不允许使用eval()
"no-extend-native": 2, //不允许扩展原生对象
"no-extra-bind": 2, //不允许不必要的函数绑定
"no-fallthrough": 2, //不允许switch按顺序全部执行所有case
"no-floating-decimal": 2, //不允许浮点数缺失数字
"no-implied-eval": 2, //不允许使用隐式eval()
"no-iterator": 2, //不允许使用__iterator__属性
"no-lone-blocks": 2, //不允许不必要的嵌套代码块
"no-loop-func": 2, //不允许在循环语句中进行函数声明
"no-multi-spaces": 2, //不允许出现多余的空格
"no-multi-str": 2, //不允许用\来让字符串换行
"no-global-assign": 2, //不允许重新分配原生对象
"no-new": 2, //不允许new一个实例后不赋值或者不比较
"no-new-func": 2, //不允许使用new Function
"no-new-wrappers": 2, //不允许使用new String, Number和Boolean对象
"no-octal": 2, //不允许使用八进制字面值
"no-octal-escape": 2, //不允许使用八进制转义序列
"no-param-reassign": 0, //不允许重新分配函数参数"no-prototype": 2, //不允许使用__proto__属性
"no-redeclare": 2, //不允许变量重复声明
"no-return-assign": 2, //不允许在return语句中使用分配语句
"no-script-url": 2, //不允许使用javascript:void(0)
"no-self-compare": 2, //不允许自己和自己比较
"no-sequences": 2, //不允许使用逗号表达式
"no-throw-literal": 2, //不允许抛出字面量错误 throw "error"
"no-unused-expressions": 2, //不允许无用的表达式
"no-void": 2, //不允许void操作符
"no-warning-comments": [1, {"terms": ["todo", "fixme", "any other term"]}], //不允许警告备注
"no-with": 2, //不允许使用with语句
"radix": 1, //使用parseInt时强制使用基数来指定是十进制还是其他进制
"vars-on-top": 0, //var必须放在作用域顶部
"wrap-iife": [2, "any"], //立即执行表达式的括号风格
"yoda": [2, "never", {"exceptRange": true}], //不允许在if条件中使用yoda条件
"strict": [2, "function"], //使用严格模式
"no-catch-shadow": 2, //不允许try catch语句接受的err变量与外部变量重名"no-delete-var": 2, //不允许使用delete操作
"no-label-var": 2, //不允许标签和变量同名
"no-shadow": 2, //外部作用域中的变量不能与它所包含的作用域中的变量或参数同名
"no-shadow-restricted-names": 2, //js关键字和保留字不能作为函数名或者变量名
"no-undef": 2, //不允许未声明的变量
"no-undef-init": 2, //不允许初始化变量时给变量赋值undefined
"no-undefined": 2, //不允许把undefined当做标识符使用
"no-unused-vars": [2, {"vars": "all", "args": "after-used"}], //不允许有声明后未使用的变量或者参数
"no-use-before-define": [2, "nofunc"], //不允许在未定义之前就使用变量"indent": 2, //强制一致的缩进风格
"brace-style": [2, "1tbs", {"allowSingleLine": false}], //大括号风格
"camelcase": [2, {"properties": "never"}], //强制驼峰命名规则
"comma-style": [2, "last"], //逗号风格
"consistent-this": [0, "self"], //当获取当前环境的this是用一样的风格
"eol-last": 2, //文件以换行符结束
"func-names": 0, //函数表达式必须有名字
"func-style": 0, //函数风格, 规定只能使用函数声明或者函数表达式
"key-spacing": [2, {"beforeColon": false, "afterColon": true}], //对象字面量中冒号的前后空格
```



```

"max-nested-callbacks": 0, //回调嵌套深度
"new-cap": [2, {"newIsCap": true, "capIsNew": false}], //构造函数名字首字母要大写
"new-parens": 2, //new时构造函数必须有小括号
"newline-after-var": 0, //变量声明后必须空一行
"no-array-constructor": 2, //不允许使用数组构造器
"no-inline-comments": 0, //不允许行内注释
"no-lonely-if": 0, //不允许else语句内只有if语句
"no-mixed-spaces-and-tabs": [2, "smart-tabs"], //不允许混用tab和空格
"no-multiple-empty-lines": [2, {"max": 2}], //空行最多不能超过两行
"no-nested-ternary": 2, //不允许使用嵌套的三目运算符
"no-new-object": 2, //禁止使用new Object()
"fun-call-spacing": 2, //函数调用时, 函数名与()之间不能有空格
"no-ternary": 0, //不允许使用三目运算符
"no-trailing-spaces": 2, //一行最后不允许有空格
"no-underscore-dangle": 2, //不允许标识符以下划线开头
"no-extra-parens": 0, //不允许出现多余的括号
"one-var": 0, //强制变量声明放在一起
"operator-assignment": 0, //赋值运算符的风格
"padded-blocks": [2, "never"], //块内行首行尾是否空行
"quote-props": 0, //对象字面量中属性名加引号
"quotes": [1, "single", "avoid-escape"], //引号风格
"semi": [2, "always"], //强制语句分号结尾
"semi-spacing": [2, {"before": false, "after": true}], //分号前后空格
"sort-vars": 0, //变量声明时排序
"space-before-blocks": [2, "always"], //块前的空格
"space-before-function-paren": [2, {"anonymous": "always", "named": "never"}], //函数定义时括号前的空格
"space-infix-ops": [2, {"int32Hint": true}], //操作符周围的空格
"keyword-spacing": 2, //关键字前后的空格
"space-unary-ops": [2, {"words": true, "nonwords": false}], //一元运算符前后不要加空格
"wrap-regex": 2, //正则表达式字面量用括号括起来
"no-var": 0, //使用let和const代替var
"generator-star-spacing": [2, "both"], //生成器函数前后空格
"max-depth": 0, //嵌套块深度
"max-len": 0, //一行最大长度, 单位为字符
"max-params": 0, //函数最多能有多少个参数
"max-statements": 0, //函数内最多有几个声明
"no-bitwise": 0, //不允许使用位运算符
"no-plusplus": 0 //不允许使用++ --运算符
}

```

## 2. 开发代理配置

编辑config 目录下的index.js文件,在env里配置如下:

```

proxyTable: {
  "/api": {
    "target": "http://127.0.0.1:9080",
    "changeOrigin": true,
  }
},

```

### 3. 打包生产环境修改

```
build: {
  // Template for index.html
  index: path.resolve(__dirname, '../dist/index.html'),

  // Paths
  assetsRoot: path.resolve(__dirname, '../dist'),
  assetsSubDirectory: 'static',
  assetsPublicPath: './',/**/'->'./'
  /**
   * Source Maps
   */
  productionSourceMap: true,
  // https://webpack.js.org/configuration/devtool/#production
  devtool: '#source-map',

  // Gzip off by default as many popular static hosts such as
  // Surge or Netlify already gzip all static assets for you.
  // Before setting to `true`, make sure to:
  // npm install --save-dev compression-webpack-plugin
  productionGzip: false,
  productionGzipExtensions: ['js', 'css'],
  // Run the build command with an extra argument to
  // View the bundle analyzer report after build finishes:
  // `npm run build --report`
  // Set to `true` or `false` to always turn it on or off
  bundleAnalyzerReport: process.env.npm_config_report
}
```

## 4.5打包部署

### 1. 项目架构

```
| -build           //打包配置,存放打包的文件
| -config          //项目环境配置 开发 生产 测试等
| -node_modules    //项目包
| -public          //一般用于存放静态文件,打包时会被直接复制到输出目录(./dist)
| -src             //项目源代码
|   | -asserts     //用于存放静态资源,打包时会经过 webpack 处理
|   | -caches       //缓存
|   | -components   //组件 存放 Vue 组件,一般是该项目公用的无状态组件
|   | -entries       //入口
|   | -pages        //页面视图
|   | -routes       //路由 配置路由的地方
|   | -services     //服务 存放服务文件,一般是网络请求等
|   | -utils        //辅助工具 工具类库
|   | -theme.styl    //主题样式文件
| -static          //存放静态资源的地方 一般是通用样式
| -test            //测试
| -.babelrc        //vue-cli脚手架工具根目录的babelrc配置文件
| -package.json    //包管理代码
| -.eslintrc.js     //代码检测配置
| -.postcssrc.js    ///添加浏览器私缀
| -index.html      //静态文件,打包时会被直接复制到输出目录中
| -.gitignore      //Git忽略文件
| -.eslintignore    //代码检测忽略文件
```

## 2. 打包

```
npm run build
```