**内蒙古科技大学**

**《Java Web实训报告》**

题 目：《垃圾淘宝》

姓名：刘鹏

学号：1667159125

专业：软件工程

班级：16-1

指导教师 ：周李涌

目录

[（一）项目介绍 6](#_Toc12310863)

[项目中的技术选型 6](#_Toc12310864)

[项目的内容 6](#_Toc12310865)

[（二）数据库设计 7](#_Toc12310866)

[（三）项目的环境搭建 7](#_Toc12310867)

[建立一个Project 7](#_Toc12310868)

[新建一个空项目 7](#_Toc12310869)

[新建一个父级项目 8](#_Toc12310870)

[创建子工程 10](#_Toc12310871)

[测试环境的安装和配置 16](#_Toc12310872)

[目录图 16](#_Toc12310873)

[实体 16](#_Toc12310874)

[与实体相对应的数据表 17](#_Toc12310875)

[在子项目各自的pom.xml文件中确定打包类型 17](#_Toc12310876)

[抽取公共依赖 17](#_Toc12310877)

[在Manager中先集成SpringMVC 18](#_Toc12310878)

[配置Tomcat 25](#_Toc12310879)

[建立新的文件目录 26](#_Toc12310880)

[配置Springmvc.xml 27](#_Toc12310881)

[设置web.xml 29](#_Toc12310882)

[新建controller目录 30](#_Toc12310883)

[新建Controller类 31](#_Toc12310884)

[设置返回目录 32](#_Toc12310885)

[测试Spring MVC环境 32](#_Toc12310886)

[配置DAO 33](#_Toc12310887)

[添加mybatis依赖 38](#_Toc12310888)

[配置service 38](#_Toc12310889)

[建立Service接口 45](#_Toc12310890)

[实现Service接口 46](#_Toc12310891)

[配置后台[manager] 47](#_Toc12310892)

[依赖-MyBatis-Spring整合包 49](#_Toc12310893)

[Manager 其他 配置文件 50](#_Toc12310894)

[web.xml 中配置，开始支持Spring 58](#_Toc12310895)

[修改数据库连接池 59](#_Toc12310896)

[有关数据库连接池连接不上去的问题 59](#_Toc12310897)

[打包问题 60](#_Toc12310898)

[（四）项目的业务分析 62](#_Toc12310899)

[用户注册 62](#_Toc12310900)

[业务描述 62](#_Toc12310901)

[时序图 63](#_Toc12310902)

[用户登陆 63](#_Toc12310903)

[业务描述 63](#_Toc12310904)

[时序图 65](#_Toc12310905)

[服务员注册 65](#_Toc12310906)

[业务描述 65](#_Toc12310907)

[时序图 66](#_Toc12310908)

[服务员登陆 67](#_Toc12310909)

[业务描述 67](#_Toc12310910)

[时序图 68](#_Toc12310911)

[获取所有服务员信息 68](#_Toc12310912)

[业务描述 68](#_Toc12310913)

[时序图 69](#_Toc12310914)

[通过服务员ID查询服务员 69](#_Toc12310915)

[业务描述 69](#_Toc12310916)

[时序图 70](#_Toc12310917)

[服务员修改密码 70](#_Toc12310918)

[业务描述 70](#_Toc12310919)

[时序图 72](#_Toc12310920)

[服务员修改用户名 72](#_Toc12310921)

[业务描述 72](#_Toc12310922)

[时序图 74](#_Toc12310923)

[获取所有的登陆记录 74](#_Toc12310924)

[业务描述 74](#_Toc12310925)

[时序图 75](#_Toc12310926)

[获取所有服务员的登陆记录 75](#_Toc12310927)

[业务描述 75](#_Toc12310928)

[时序图 76](#_Toc12310929)

[（五）核心代码 76](#_Toc12310930)

[用户注册 76](#_Toc12310931)

[Controller 76](#_Toc12310932)

[Service 78](#_Toc12310933)

[DAO层 78](#_Toc12310934)

[用户登陆 79](#_Toc12310935)

[Controller 79](#_Toc12310936)

[服务员注册 80](#_Toc12310937)

[Controller 80](#_Toc12310938)

[Service 82](#_Toc12310939)

[DAO层 83](#_Toc12310940)

[服务员登陆 83](#_Toc12310941)

[Controller 83](#_Toc12310942)

[Service 86](#_Toc12310943)

[DAO层 87](#_Toc12310944)

[获取所有服务员 87](#_Toc12310945)

[Controller 87](#_Toc12310946)

[Service 88](#_Toc12310947)

[DAO层 89](#_Toc12310948)

[通过服务员ID查询服务员 90](#_Toc12310949)

[Controller 90](#_Toc12310950)

[Service 91](#_Toc12310951)

[DAO层 92](#_Toc12310952)

[服务员修改密码 93](#_Toc12310953)

[Controller 93](#_Toc12310954)

[Service 97](#_Toc12310955)

[DAO层 97](#_Toc12310956)

[服务员修改用户名 98](#_Toc12310957)

[Controller 98](#_Toc12310958)

[Service 102](#_Toc12310959)

[DAO层 102](#_Toc12310960)

[获取所有的登陆记录 103](#_Toc12310961)

[Controller 103](#_Toc12310962)

[Service 104](#_Toc12310963)

[DAO层 104](#_Toc12310964)

[通过服务员ID查询服务员 105](#_Toc12310965)

[Controller 105](#_Toc12310966)

[Service 106](#_Toc12310967)

[DAO层 107](#_Toc12310968)

# （一）项目介绍

学校针对我们培训了有关JavaWeb项目实战，具体做法就是看视频，敲代码，没有什么挑战性，代码都是现成的，一个SSM框架，具体没有讲什么特别多的内涵，就是在不停的改BUG，我就从这个培训老师给的项目里面学习了一些内容，然后自己写一个项目出来。

## 项目中的技术选型

|  |  |
| --- | --- |
| **未找到图形项目表。** | 版本 |
| Spring | 5.1.7.RELEASE |
| SpringMVC | 5.1.7.RELEASE |
| MyBaits | 3.5.1 |
| 数据库 | 版本 |
| MySQL | 8.0.13 |
| 数据库连接池 | 版本 |
| Druid | 1.1.11 |
| 服务器 | 版本 |
| Tomcat | 9.0.21 |

## 项目的内容

项目的原型叫：《闲鱼》，但是我没有把它全部写完，只是实现了一部分的功能，因为工期太短，需求太多。只实现了为数不多的功能。

1. 用户注册
2. 用户登陆
3. 服务员注册
4. 服务员登陆
5. 获取所有服务员
6. 通过ID查询服务员
7. 服务员修改密码
8. 服务员修改用户名
9. 记录服务员的登陆
10. 获取所有的登陆记录
11. 通过ID查询服务员的登陆记录

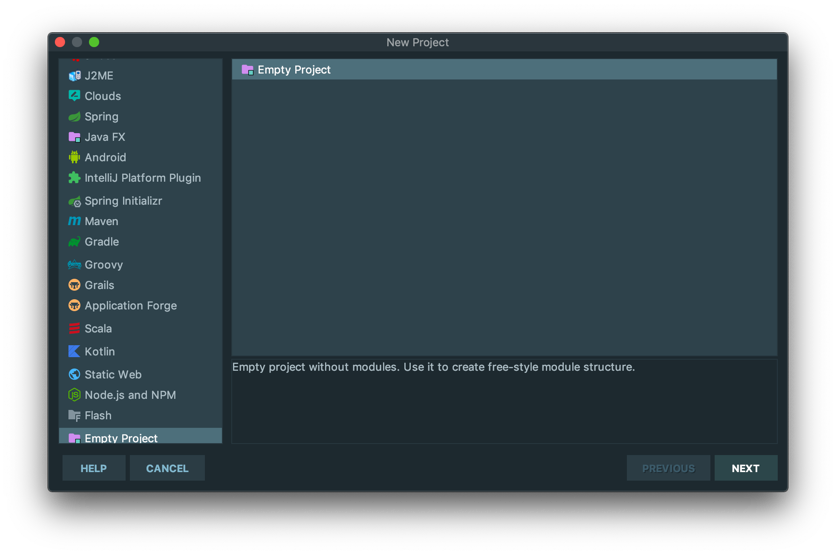
# （二）数据库设计

详情见数据库设计。

# （三）项目的环境搭建

## 建立一个Project

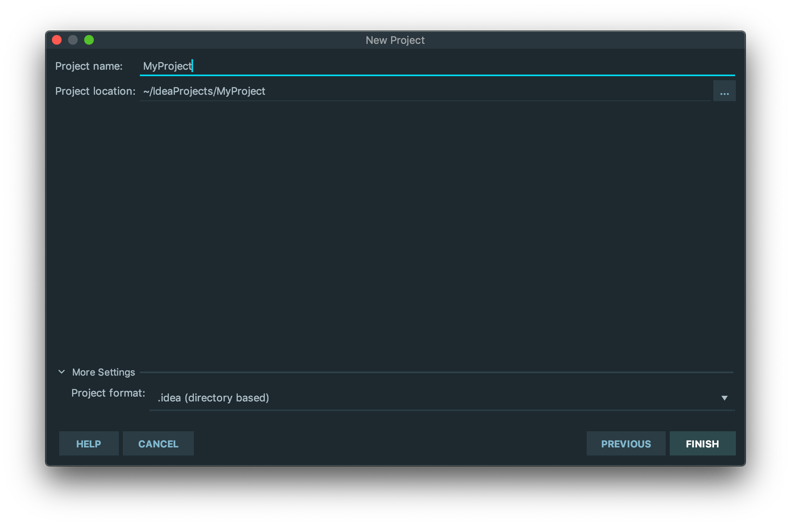
### 新建一个空项目



3· 1 建立一个空项目

￼

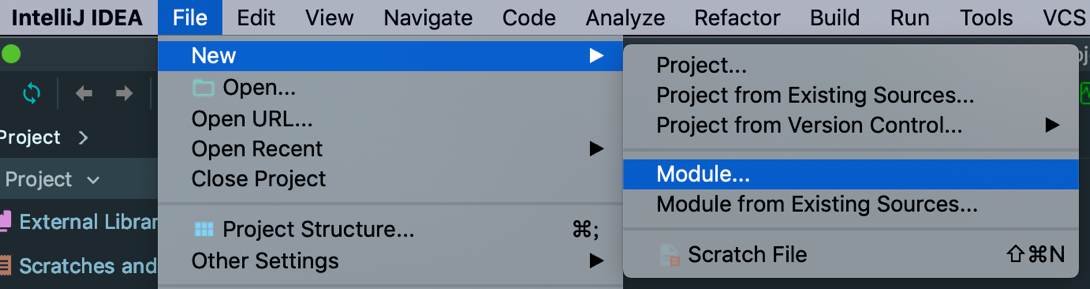
#### 项目名称 ： MyProject



3· 2 项目名称￼

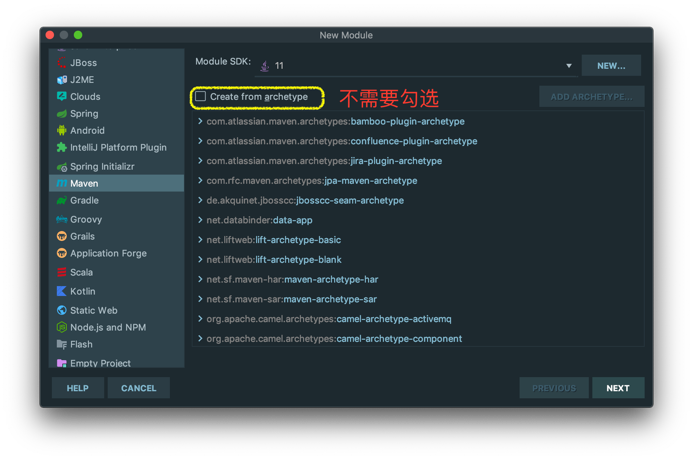
### 新建一个父级项目

第一步



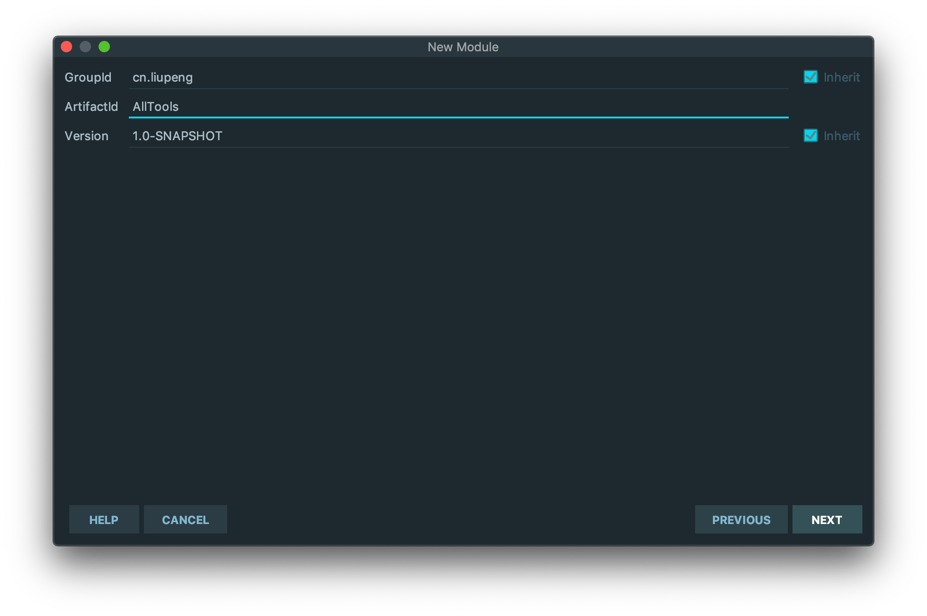
3· 3 新建一个父级项目￼

第二步



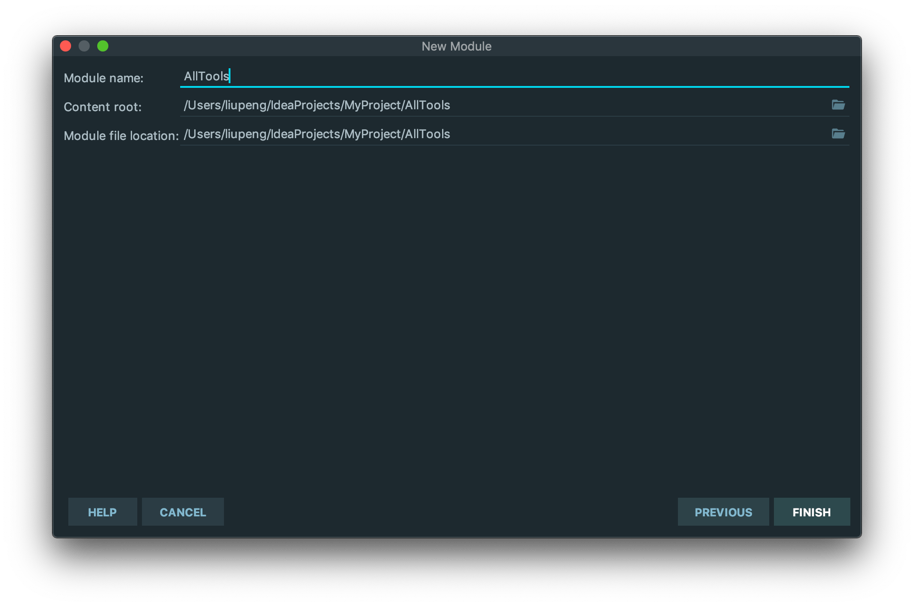
3· 4 新建一个父级项目 01

第三步



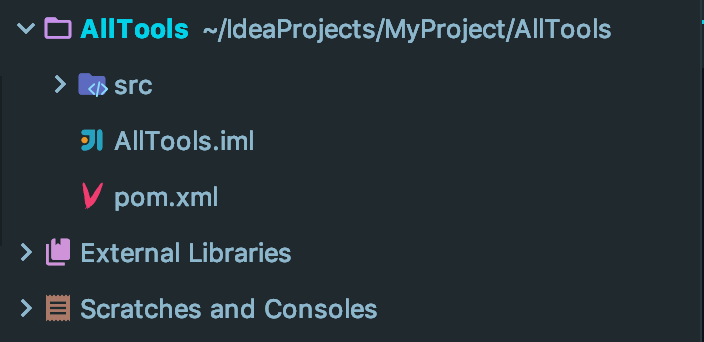
3· 5 新建一个父级项目 02

第四部



3· 6 新建一个父级项目 03

项目创建完成

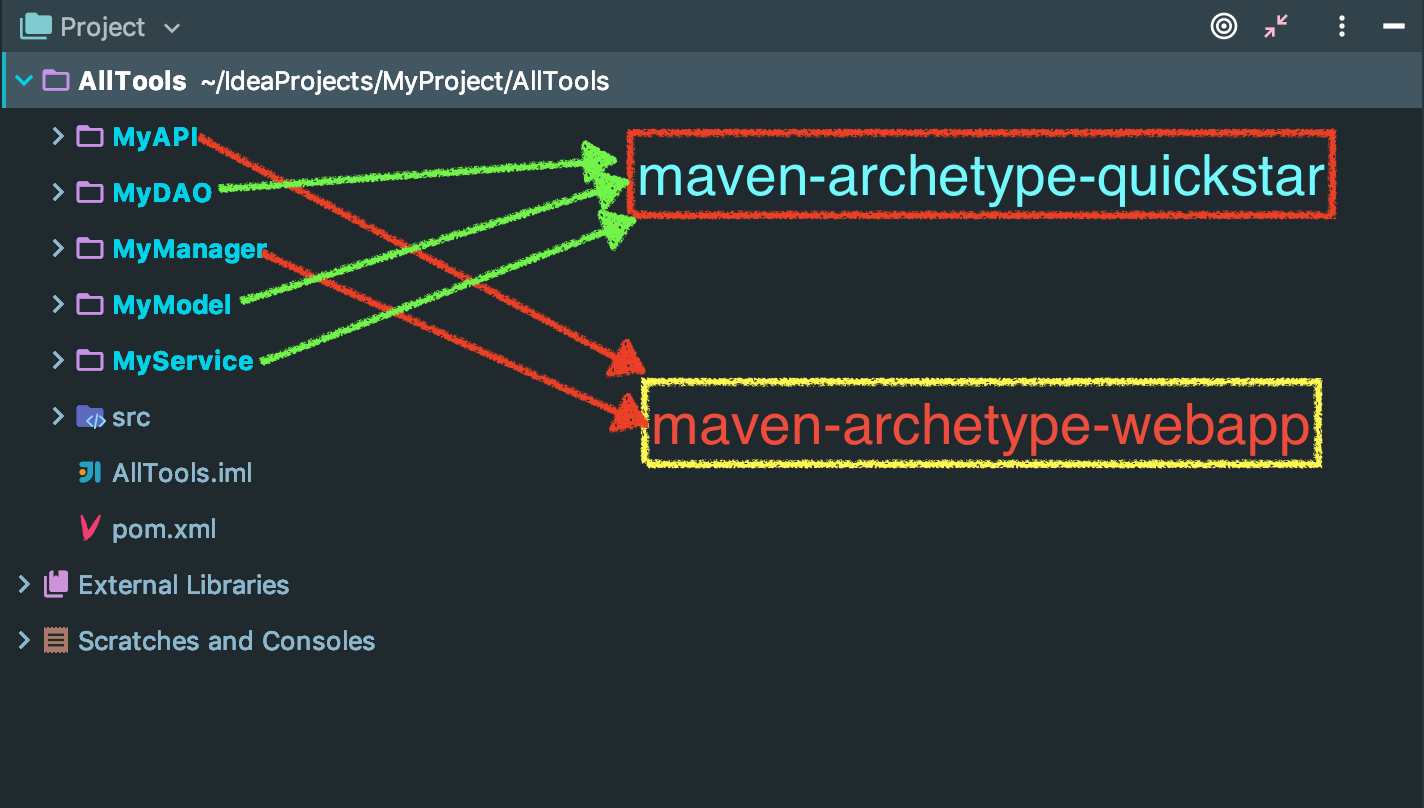


3· 7 新建一个父级项目 04

￼

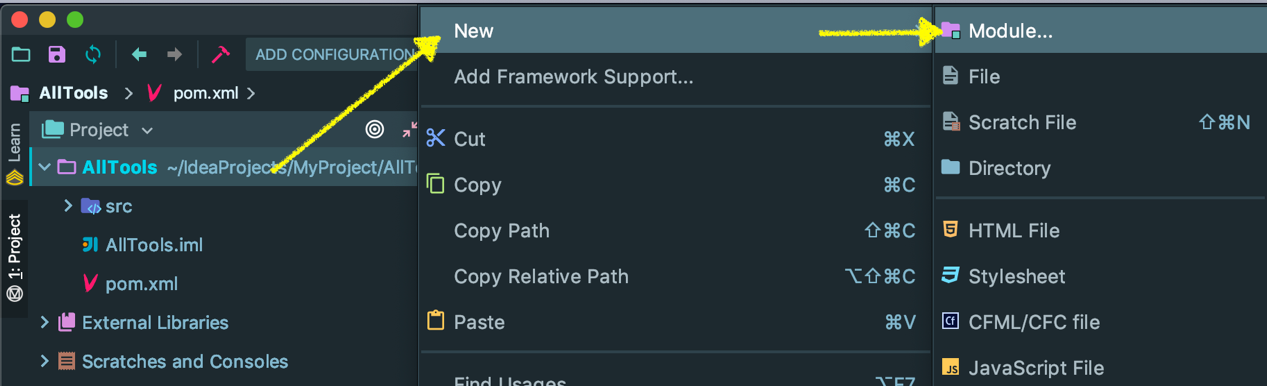
### 创建子工程

#### 子工程类型



3· 8 创建子工程 00

#### 如何创建子工程



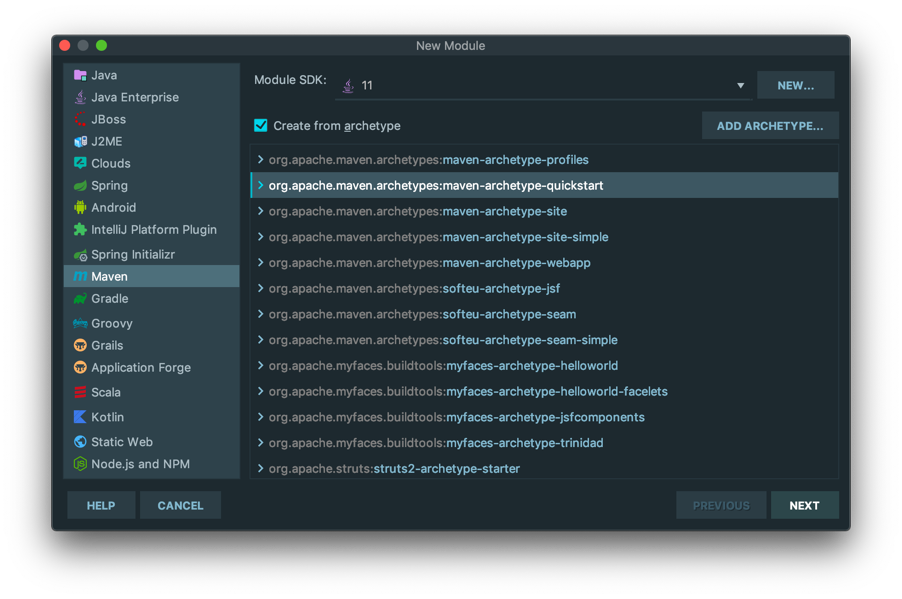
3· 9 创建子工程 01

￼

#### 子工程名称

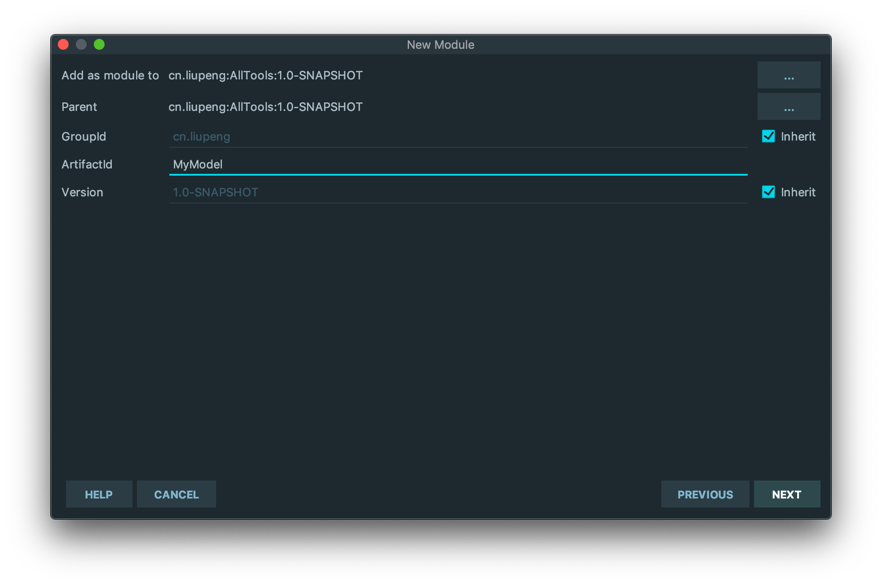
**MyModel**

*选中****maven-archetype-quickstart***



3· 10 创建子工程 02

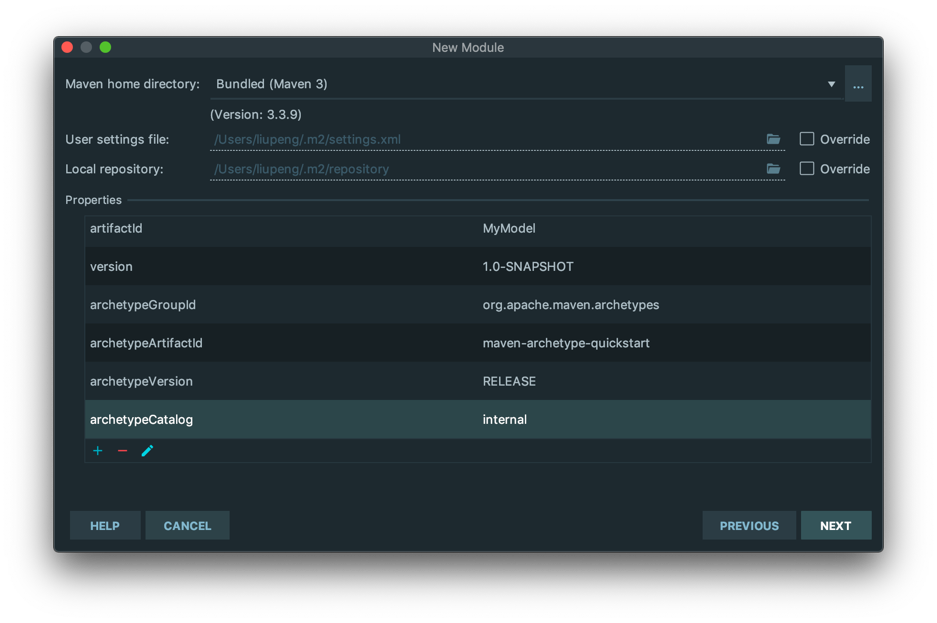
**填写项目名称**



3· 11 创建子工程 03

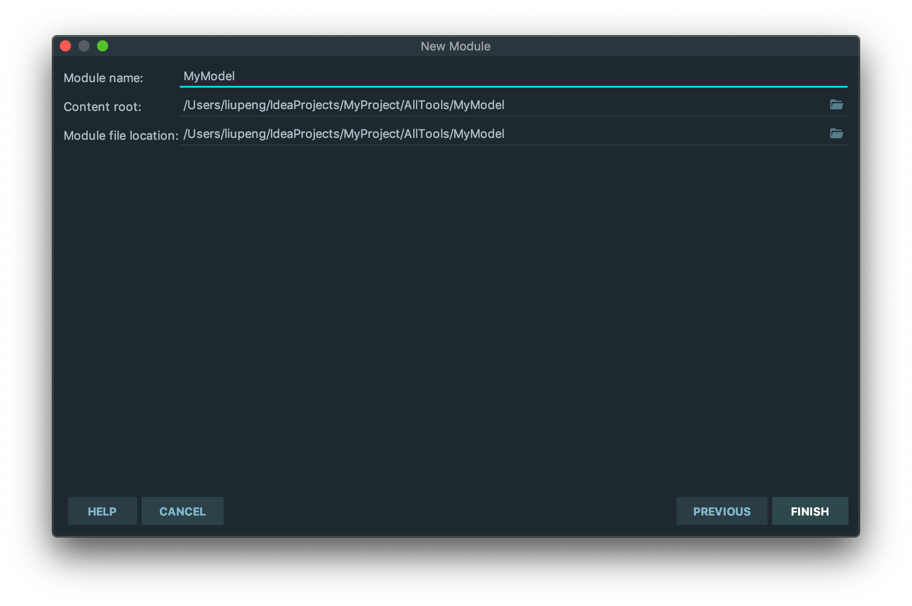
解决Maven下载过慢的问题--加一组键值对

* Name：**archetypeCatalog**
* Value：**internal**



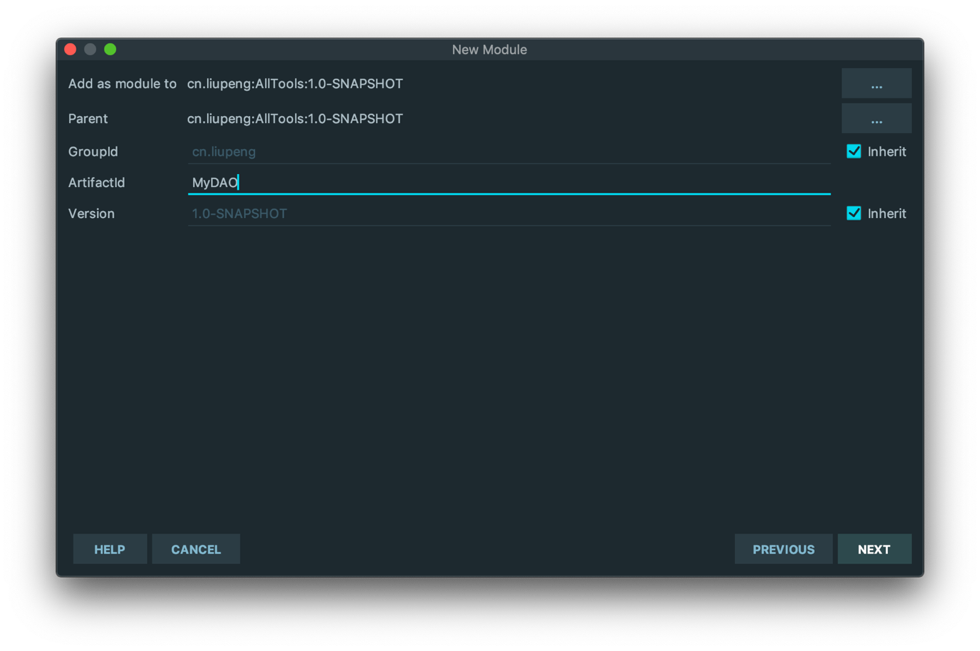
3· 12 创建子工程04￼

完成项目的构建



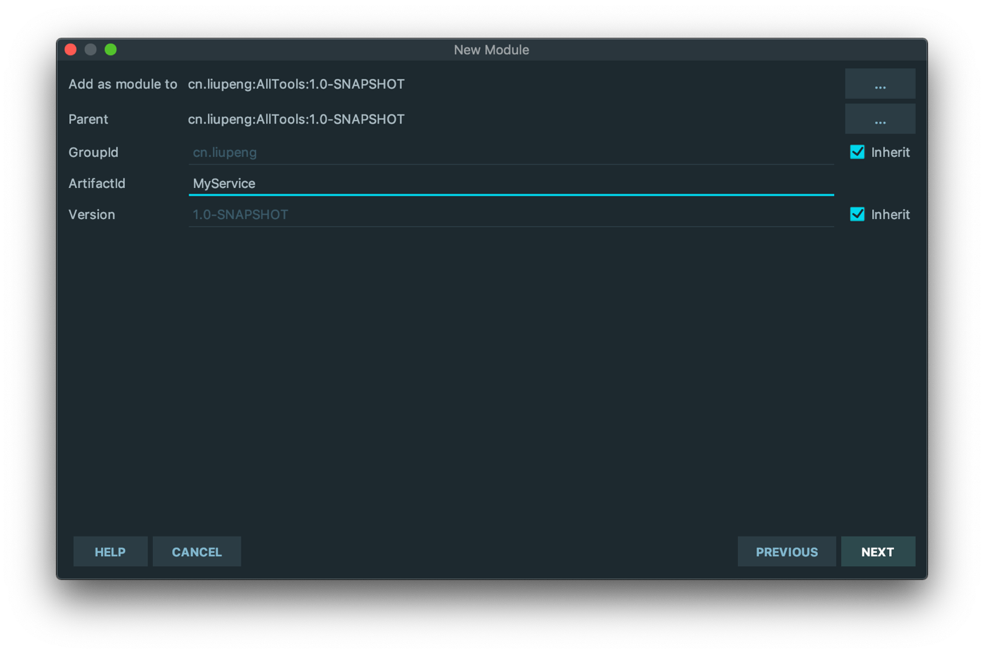
3· 13 创建子工程 05￼

**MyDAO**



3· 14 创建子工程 06

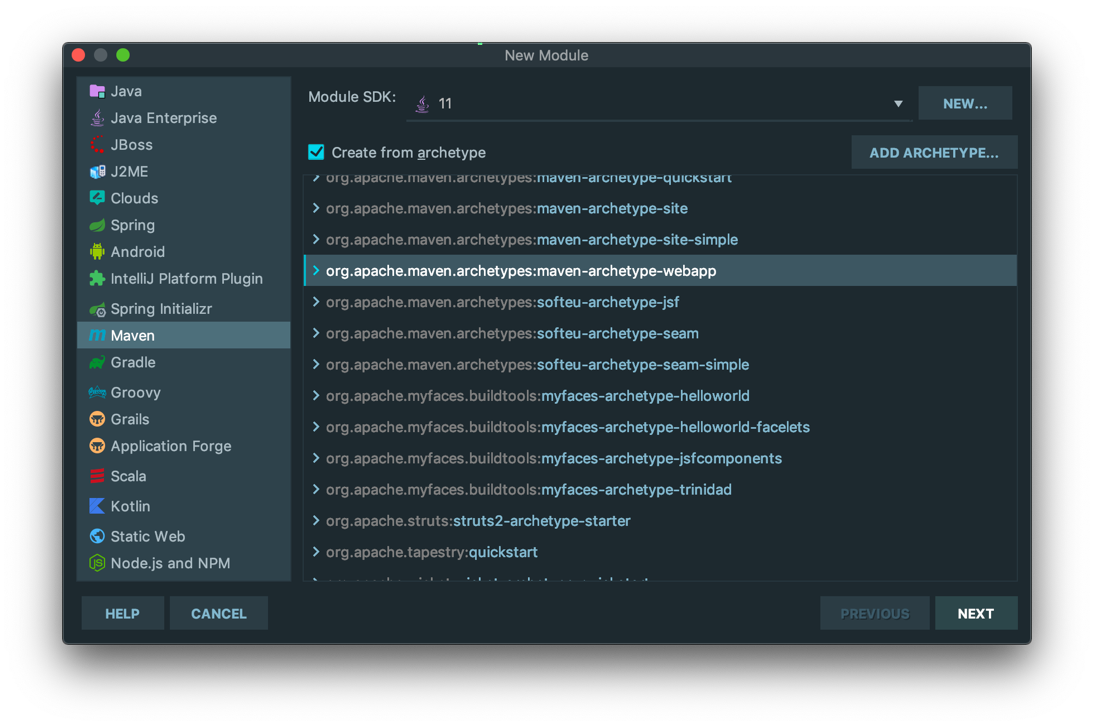
**MyService**



3· 15 创建子工程 07

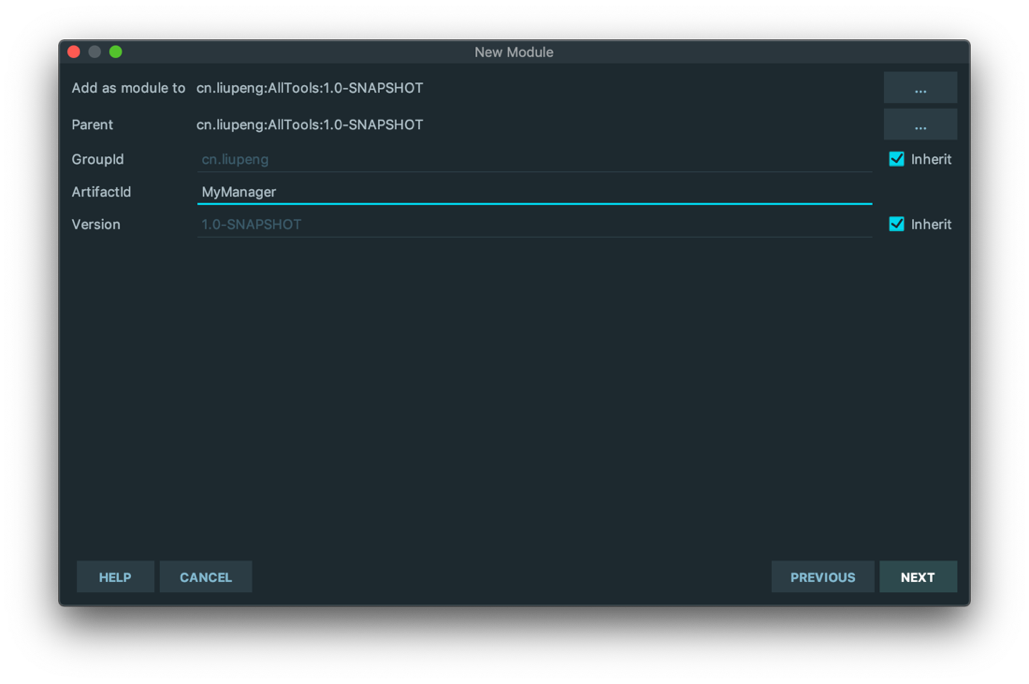
**MyManager**

选中**maven-archetype-webapp**



3· 16 创建子工程 08

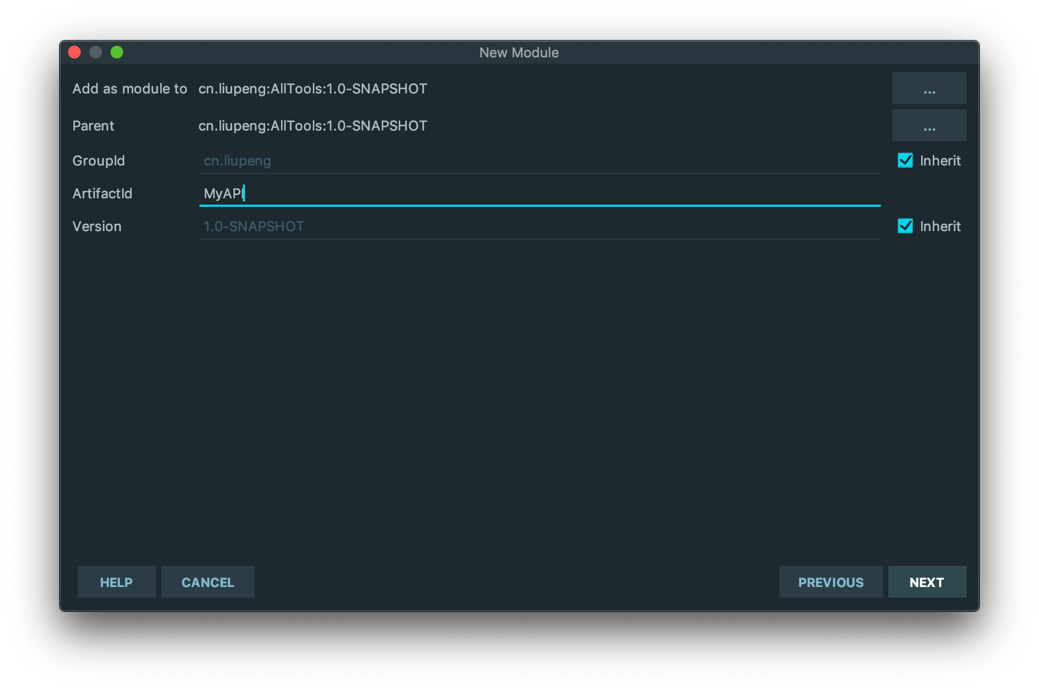
写下项目名称



3· 17 创建子工程 09

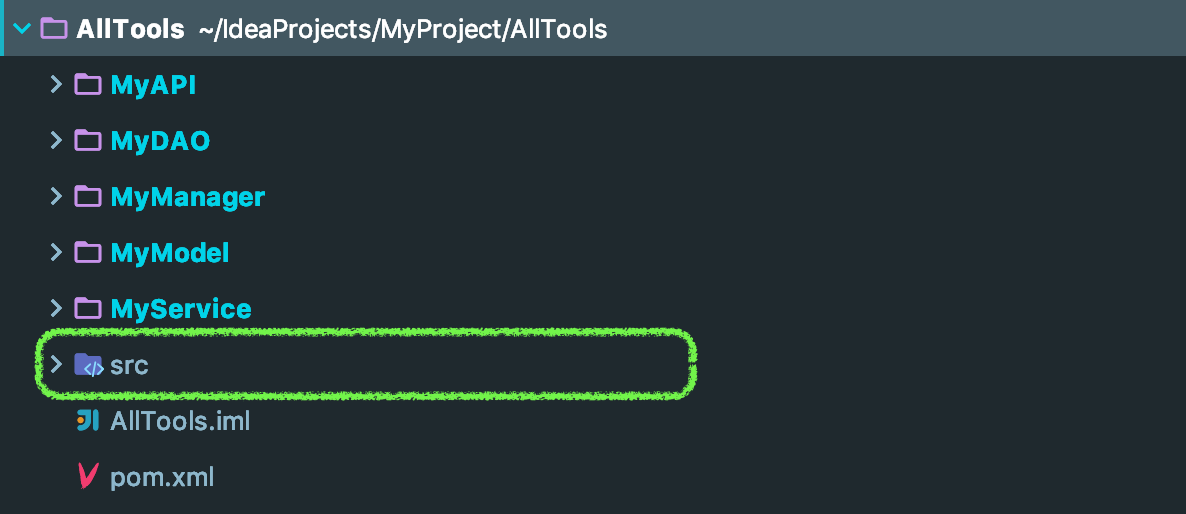
￼

**MyAPI**



3· 18 创建子工程 10

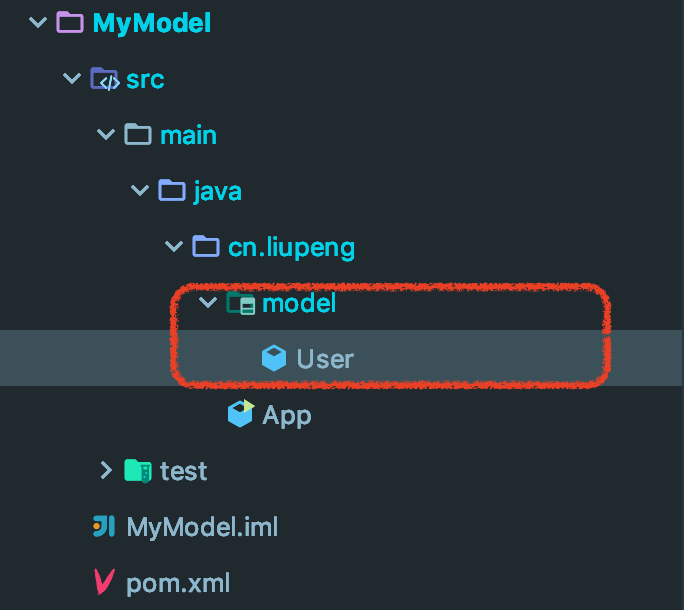
#### 删除一个没有用处的目录



3· 19 创建子工程 11￼

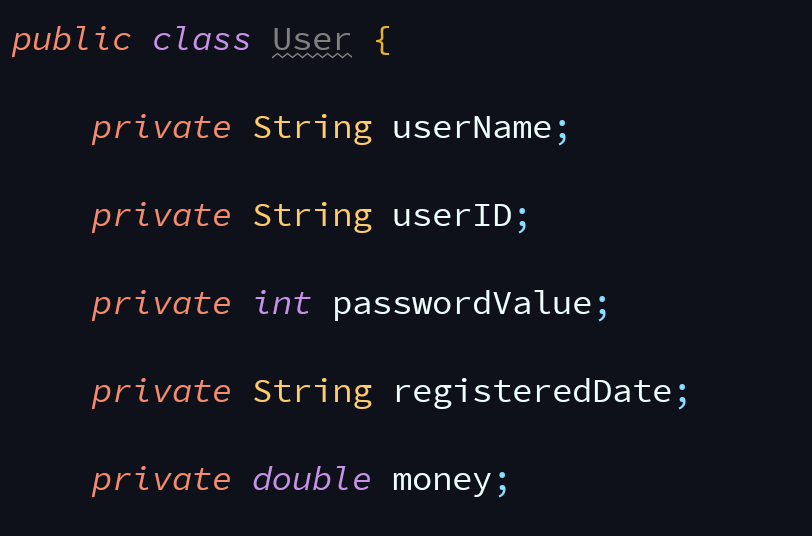
## 测试环境的安装和配置

### 目录图



3· 20 创建子工程 12

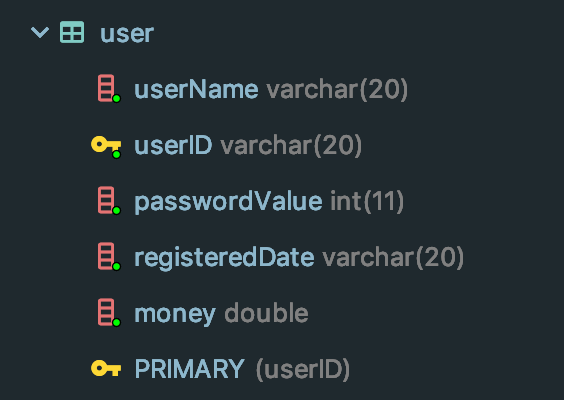
### 实体



3· 21 创建子工程 13

￼

### 与实体相对应的数据表



3· 22 创建子工程 14

### 在子项目各自的pom.xml文件中确定打包类型

* MyModel
  + <packaging>jar</packaging>
* MyDAO
  + <packaging>jar</packaging>
* MyService
  + <packaging>jar</packaging>
* MyManager
  + <packaging>war</packaging>
* MyAPI
  + <packaging>war</packaging>

### 抽取公共依赖

#### 父项目的pom.xml文件中设置

<!-- 公共版本号依赖 -->

<properties>

<junit.version>4.12</junit.version>

</properties>

<!-- 公共依赖 -->

<dependencies>

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

</dependency>

</dependencies>

### 在Manager中先集成SpringMVC

#### 父项目的pom.xml文件中设置版本控制

<spring.version>5.1.7.RELEASE</spring.version>

<spring.oxm.version>5.1.4.RELEASE</spring.oxm.version>

<spring.aspect.version>5.1.4.RELEASE</spring.aspect.version>

#### MyManager项目的pom.xml文件中设置SpringMVC的依赖

<dependencies>

<!-- 后台[manager]依赖service -->

<dependency>

<artifactId>MyService</artifactId>

<groupId>cn.liupeng</groupId>

<version>1.0-SNAPSHOT</version>

</dependency>

<!-- ===================================================== servlet start ============ -->

<!-- Session -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

</dependency>

<!-- ===================================================== servlet start ============ -->

<!-- ===================================================== spring / spring-mvc start ============ -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.1.7.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>${spring.aspect.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-expression</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.1.7.RELEASE</version>

</dependency>

<!-- 事务 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-testTable</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.oxm.version}</version>

</dependency>

<!-- ===================================================== spring / spring-mvc end ============ -->

<!-- ===================================================== 数据库驱动 start ===================== -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.connector.verssion}</version>

</dependency>

<!-- ===================================================== 数据库驱动 end ======================= -->

<!-- ===================================================== 数据库连接池 start =================== -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>${druid.version}</version>

</dependency>

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>${c3p0.versioon}</version>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<!-- ===================================================== 数据库连接池 end ===================== -->

<!-- ===================================================== JSTL start ========================= -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>${jstl.version}</version>

</dependency>

<!-- ===================================================== JSTL end =========================== -->

<!-- ===================================================== 动态代理 start ====================== -->

<!-- -->

<!-- https://mvnrepository.com/artifact/taglibs/standard -->

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>${taglibs.version}</version>

</dependency>

<!-- ===================================================== 动态代理 end ======================== -->

<!-- ===================================================== 日志 start ========================== -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version></version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.25</version>

</dependency>

<!-- ===================================================== 日志 end ============================ -->

<!-- ===================================================== MyBatis-Spring整合 start ============ -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>2.0.1</version>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.5.0-M1</version>

<scope>testTable</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.5.0-M1</version>

<scope>testTable</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

</dependency>

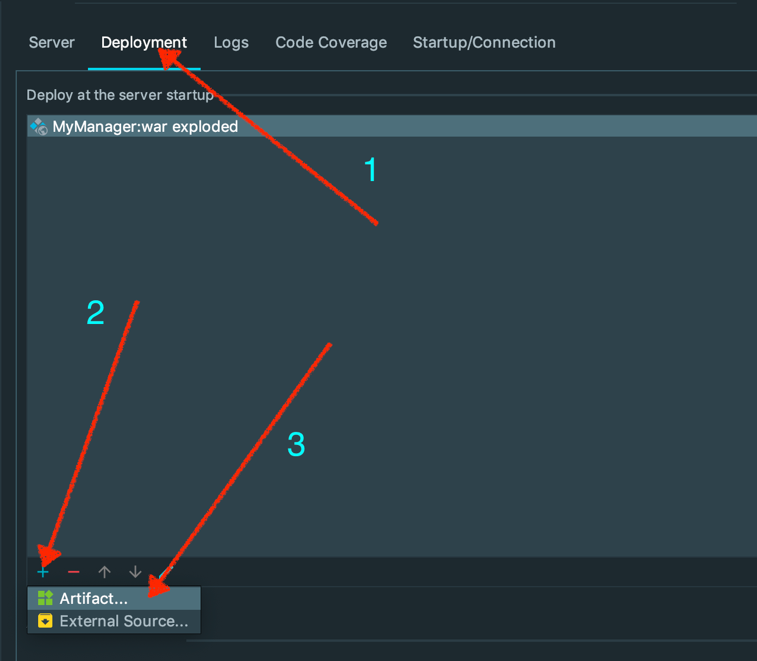
<!-- ===================================================== MyBatis-Spring整合 end =============== -->

</dependencies>

### 配置Tomcat

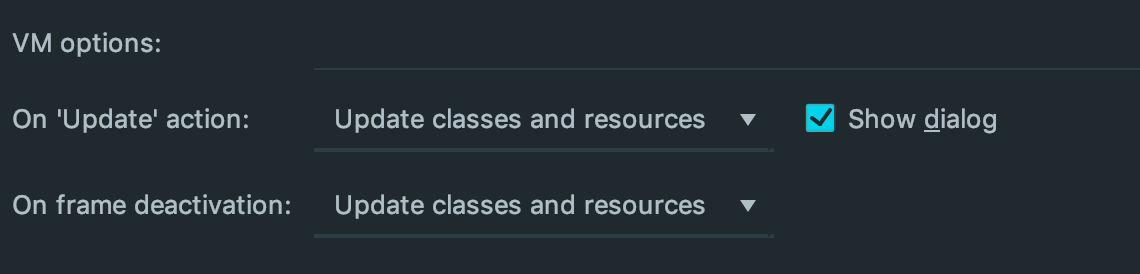
* tomcat-version：9.0.12
* java-version:12

#### 添加war包



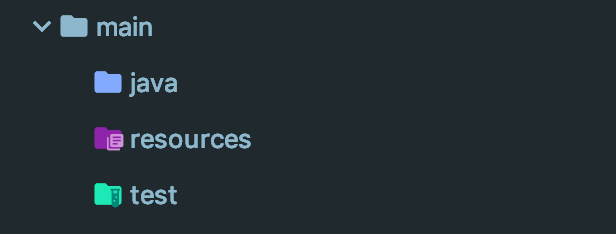
1· 23 创建子工程 15

#### 设置自动刷新JSP页面



3· 24 创建子工程 16

## 建立新的文件目录



3· 25 创建子工程 17

1. 添加Java文件夹到src目录下
   1. 设置为源代码目录
2. 添加resources文件夹到src目录下
   1. 设置为配置文件目录
3. 添加test文件夹到src目录下
   1. 设置为单元测试目录

### 配置Springmvc.xml

#### 完整代码

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:task="http://www.springframework.org/schema/task"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/task

http://www.springframework.org/schema/task/spring-task.xsd">

<!-- 使用注解开发，不用配置controller，需要配置一个组件扫描器 -->

<context:component-scan base-package="cn.liupeng.controller"/>

<!-- 视图解析器 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!-- 配置从项目根目录到指定目录一端路径 ,建议指定浅一点的目录-->

<property name="prefix" value="/WEB-INF/views/"></property>

<!-- 文件的后缀名 -->

<property name="suffix" value=".jsp"></property>

</bean>

<!-- 开启 Spring 框架注解的支持 -->

<mvc:annotation-driven />

<!--加载静态文件-->

<mvc:default-servlet-handler />

</beans>

### 设置web.xml

#### 核心代码

##### 配置前端控制器

<!--配置前端控制器-->

<servlet>

<servlet-name>dispatcherServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 全局初始化参数 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

##### 配置解决中文乱码的过滤器

<!--配置解决中文乱码的过滤器-->

<filter>

<filter-name>characterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncodingFilter</filter-name>

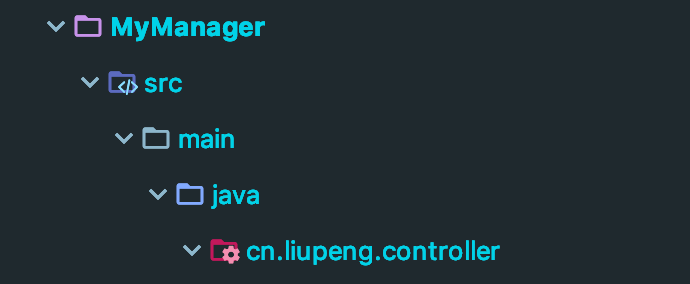
<url-pattern>/\*</url-pattern>

</filter-mapping>

### 新建controller目录

#### 文件夹目录

*cn.liupeng.controller*



3· 26 创建子工程 18

￼

### 新建Controller类

#### 路径

*cn.liupeng.controller.UserController.java*

#### 代码

package cn.liupeng.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping(path = "/user")

public class UserController {

@RequestMapping(path = "/login")

public String login() {

System.out.println("success");

return "success";

}

}

### 设置返回目录

#### 路径

* WEB-INF/views

#### 文件

* WEB-INF/views/success.jsp

#### 代码

<%--  
Created by IntelliJ IDEA.  
User: liupeng  
Date: 2019-06-21  
Time: 00:36  
To change this template use File | Settings | File Templates.  
--%>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

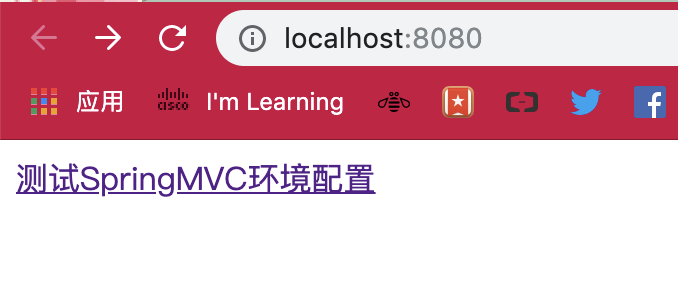
### 测试Spring MVC环境

#### 超链接

<a href="user/login">测试SpringMVC环境配置</a>

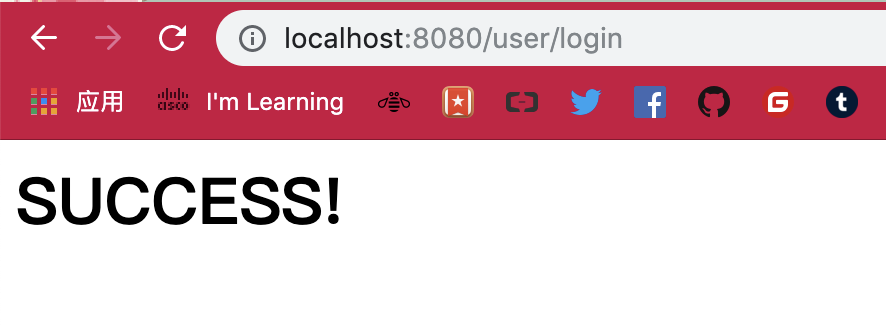
#### 效果

开始测试



3· 27 测试Spring MVC环境 01

测试效果



3· 28 测试Spring MVC环境 02

### 配置DAO

#### DAO需要用到Model

#### 修改MyDAO子项目的pom.xml文件

*添加依赖*

* 添加MyModel工程的引用

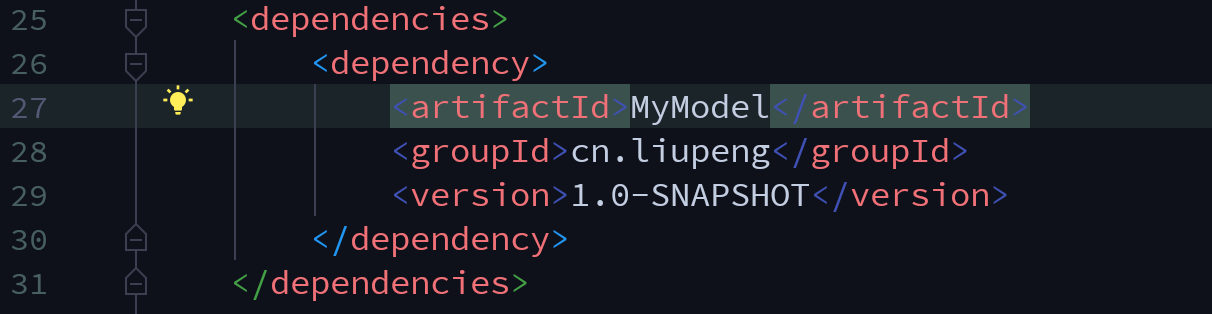
<dependency>

<artifactId>MyModel</artifactId>

<groupId>cn.liupeng</groupId>

<version>1.0-SNAPSHOT</version>

</dependency>

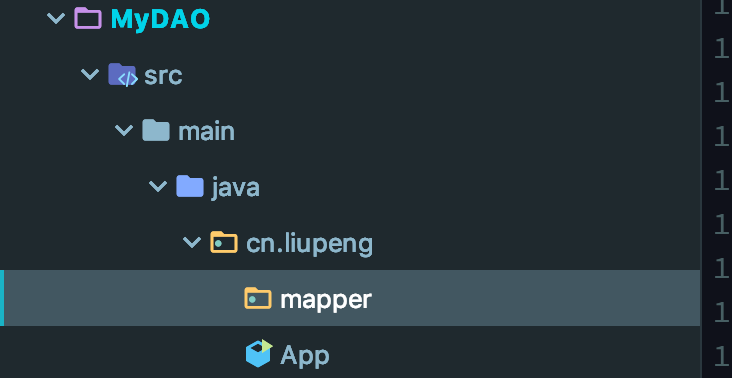


3· 29 配置DAO 01

#### 新建Mapper文件夹

##### 路径

* cn.liupeng.mapper



3· 30 新建Mapper文件夹

#### 建立\*Mapper接口

package cn.liupeng.mapper;

import cn.liupeng.model.User;

import java.util.List;

public interface UserMapper {

/\*\*

\* 添加用户

\* @param user

\*/

public void addUser(User user);

/\*\*

\* 通过用户ID删除用户

\* @param id

\*/

public void deleteUser(Integer id);

/\*\*

\* 更新用户

\* @param user

\*/

public void updateUser(User user);

/\*\*

\* 获取所有用户

\* @return

\*/

public List<User> listUser();

/\*\*

\* 通过用户ID查询用户

\* @param id

\* @return

\*/

public User findUser(Integer id);

}

#### 建立\*Mapper.xml文件

<?xml version="1.0" encoding="UTF-8" ?>

<!-- 引入dtd -->

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!-- namespace就是接口的包名加类名 -->

<mapper namespace="cn.liupeng.mapper.UserMapper">

<select id="listUser" resultType="User">

select \* from javaDataBase.user

</select>

<select id="findUser" resultType="User" parameterType="integer">

select \* from javaDataBase.user where userID = #{id}

</select>

<delete id="deleteUser" parameterType="integer">

delete \* from javaDataBase.user where userID = #{id}

</delete>

<update id="updateUser" parameterType="User">

update user set

userName = #{userName} ,

passwordValue = #{passwordValue} ,

registeredDate = #{registeredDate} ,

money = #{money}

where

userID =#{userID}

</update>

<insert id="addUser" parameterType="User">

insert user(userName, userID, passwordValue, registeredDate, money)

value #{userName}, #{userID} , #{passwordValue} , #{registeredDate} , #{money}

</insert>

</mapper>

#### 要点

*namespace: 接口的包名加类名*

如 cn.liupeng.mapper.UserMapper

*.xml文件中的 id: 必须和 接口文件中的方法名称一致*

*parameterType: 增加的时候传进来的是对象类型*

*resaultType: 是返回值类型*

### 添加mybatis依赖

#### MyDAO - pom.xml

<!-- Mybatis 依赖 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>${mybatis.version}</version>

</dependency>

#### 父项目中抽取项目依赖的版本号

<mybatis.version>3.5.1</mybatis.version>

### 配置service

**service 依赖于 DAO**

#### 修改MyService子项目的pom.xml文件

*添加依赖*

* 添加MyDAO工程的引用

<dependency>

<artifactId>MyDAO</artifactId>

<groupId>cn.liupeng</groupId>

<version>1.0-SNAPSHOT</version>

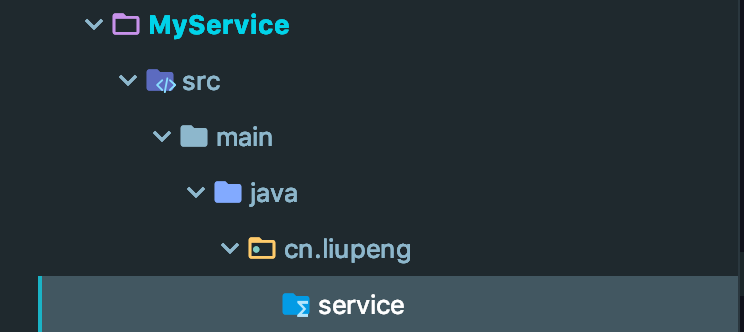
</dependency>



3· 31 配置service 01

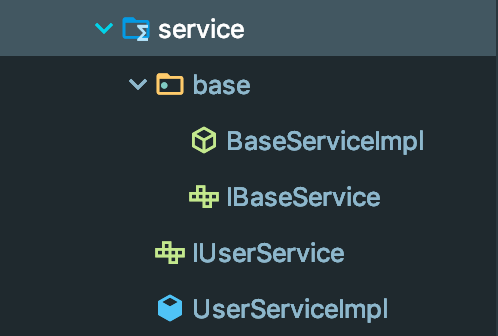
#### 新建文件夹

*文件夹目录  
cn.liupeng.service*



3· 32 配置service 02

#### 新建文件



3· 33 配置service 03

##### IBaseService

package cn.liupeng.service.base;

/\*\*

\* @author liupeng

\*/

public interface IBaseService<T> {

/\*\*

\* add or insert

\* @param t

\*/

public void insert(T t);

/\*\*

\* delete

\* @param id

\*/

public void deleteByID(Integer id);

/\*\*

\* delete by uuid

\* @param uuid

\*/

public void deleteByUUID(String uuid);

/\*\*

\* update

\* @param t

\*/

public void update(T t);

/\*\*

\* select or list or find

\* @return

\*/

public T select();

/\*\*

\* select or list or find by id

\* @param id

\* @return

\*/

public T selectByID(Integer id);

/\*\*

\* select or list or find by uuid

\* @param uuid

\* @return

\*/

public T selectByUUID(String uuid);

}

##### BaseServiceImpl

package cn.liupeng.service.base;

import cn.liupeng.mapper.UserMapper;

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

/\*\*

\* @author liupeng

\*/

public abstract class BaseServiceImpl<T> implements IBaseService<T> {

// 统一管理DAO

@Autowired

protected UserMapper userMapper;

}

##### IUserService

package cn.liupeng.service;

import cn.liupeng.model.User;

import cn.liupeng.service.base.IBaseService;

public interface IUserService extends IBaseService<User> {

// 特有的方法

public User login(String userName, String passwordValue) ;

}

##### UserServiceImpl

package cn.liupeng.service;

import cn.liupeng.model.User;

import cn.liupeng.service.base.BaseServiceImpl;

import org.springframework.stereotype.Service;

/\*\*

\* @author liupeng

\*/

@Service

public class UserServiceImpl extends BaseServiceImpl<User> implements IUserService {

@Override

public User login(String userName, String passwordValue) {

return null;

}

@Override

public void insert(User user) {

}

@Override

public void deleteByID(Integer id) {

}

@Override

public void deleteByUUID(String uuid) {

}

@Override

public void update(User user) {

}

@Override

public User select() {

return null;

}

/\*\*

\* 通过ID查找用户

\* @param id

\* @return

\*/

@Override

public User selectByID(Integer id) {

return this.userMapper.findUser(id);

}

/\*\*

\* 通过UUID查找用户

\* @param uuid

\* @return

\*/

@Override

public User selectByUUID(String uuid) {

return this.userMapper.findUserByUUID(uuid);

}

}

### 建立Service接口

#### 建立baseService接口

* IBaseService
* 内含：增删改查基本方法

#### 实现baseService接口

* BaseServiceImpl
* 内涵：某一个实体类的Mapper接口对象

#### 某一个实体对象接口的Service操作类继承baseService接口

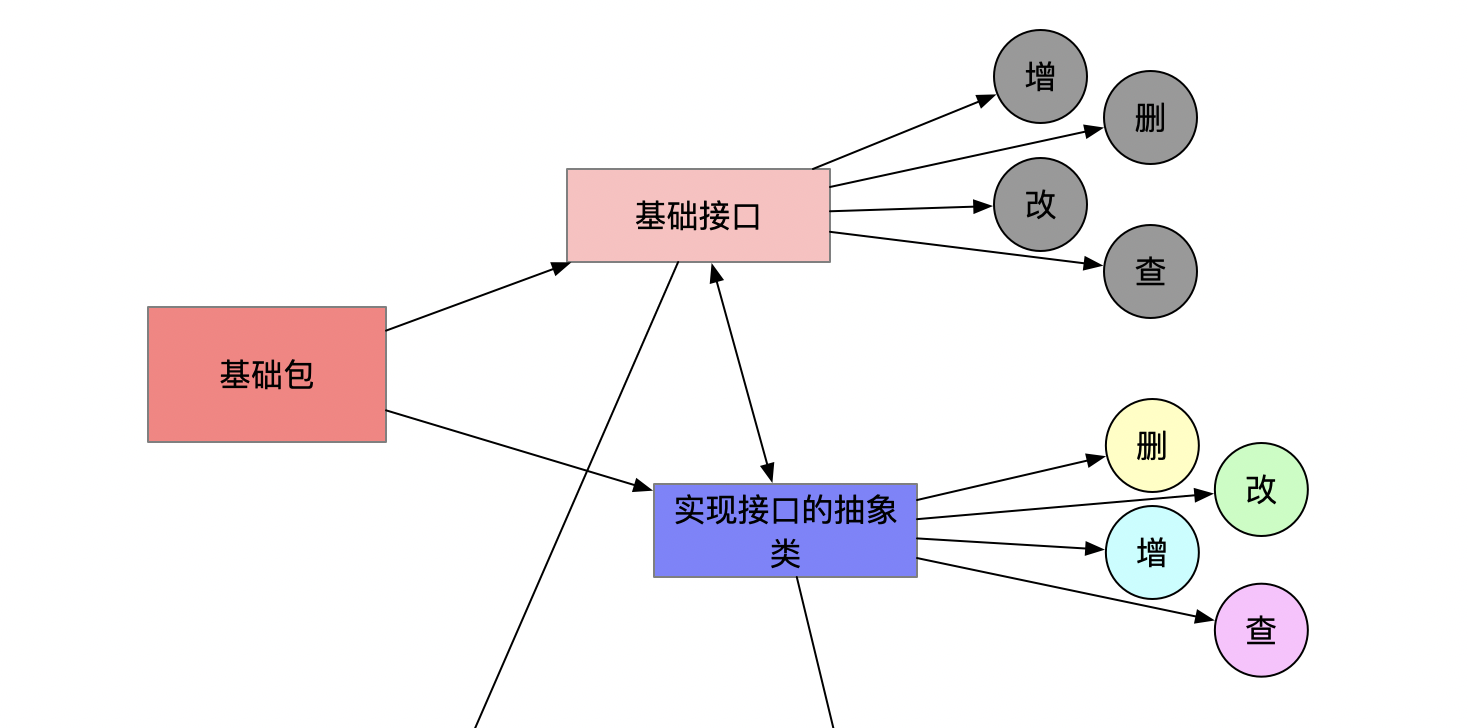
* IUserService
* 内涵：某一个实体对象的再操作

#### 继承增删改查抽象类，同时实现某个特使实体的操作接口

* UserServiceImpl
* 内涵：
  + 实现增删改查
  + 实现某个实体对象的操作方法

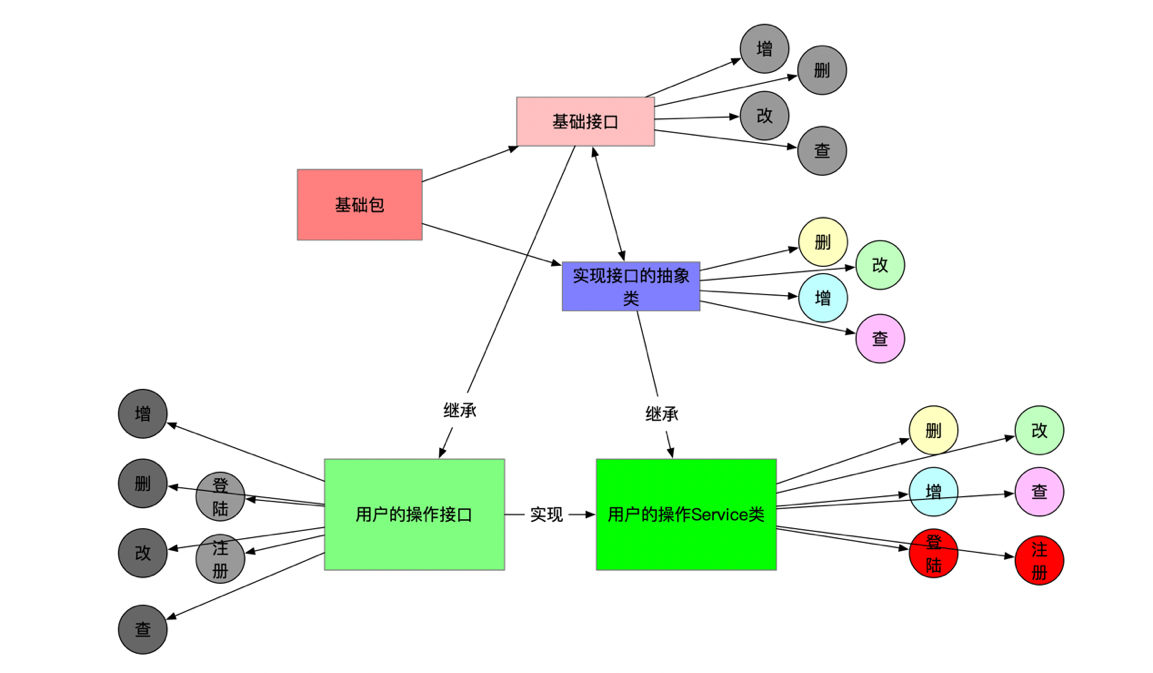
### 实现Service接口

#### 基础类的作用



3· 34 实现Service接口 01

#### 实现流程



3· 35 实现Service接口 02

￼

### 配置后台[manager]

**manager 依赖于 service**

#### 修改MyManager子项目的pom.xml文件

*添加依赖*

* 添加MyService工程的引用

<!-- 后台[manager]依赖sservice -->

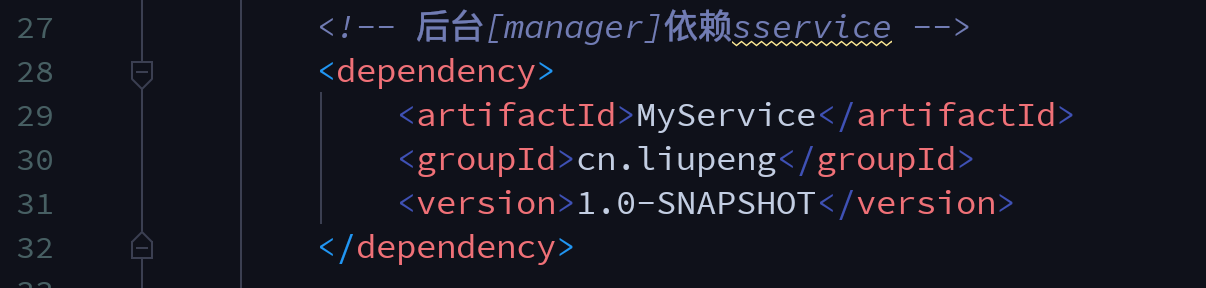
<dependency>

<artifactId>MyService</artifactId>

<groupId>cn.liupeng</groupId>

<version>1.0-SNAPSHOT</version>

</dependency>



3· 36 配置后台[manager]

#### 依赖-数据库驱动

<mysql.connector.verssion>8.0.13</mysql.connector.verssion>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.connector.verssion}</version>

</dependency>

#### 依赖-数据库连接池

<c3p0.versioon>0.9.1.2</c3p0.versioon>

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>${c3p0.versioon}</version>

</dependency>

#### 依赖-JSTL标签

<jstl.version>1.2</jstl.version>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>${jstl.version}</version>

</dependency>

#### 依赖-动态代理

<taglibs.version>1.1.2</taglibs.version>

<!-- https://mvnrepository.com/artifact/taglibs/standard -->

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>${taglibs.version}</version>

</dependency>

#### 依赖-日志

<slf4j.verssion>1.7.25</slf4j.verssion>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version></version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.25</version>

</dependency>

## 依赖-MyBatis-Spring整合包

<mybatis.spring.version>2.0.1</mybatis.spring.version>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>2.0.1</version>

</dependency>

### Manager 其他 配置文件

#### db.properties

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/javaDatabase?&amp;useSSL=false&amp;serverTimezone=UTC

jdbc.username=root

jdbc.password=\*\*\*\*\*\*

#### log4j.properties

### 设置###

log4j.rootLogger = debug,stdout

### 输出信息到控制抬 ###

log4j.appender.stdout = org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target = System.out

log4j.appender.stdout.layout = org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n

#### mybatis.xml

##### 配置别名

<typeAliases>

<!-- 批量加载别名，制定批量定义别名的类包即可，别名为类民 -->

<package name="cn.liupeng.model"/>

</typeAliases>

##### 批量加载映射文件

<!-- 批量加载映射文件 -->

<mappers>

<package name="cn.liupeng.mapper"/>

</mappers>

##### 完整代码

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 配置别名 -->

<typeAliases>

<!-- 批量加载别名，制定批量定义别名的类包即可，别名为类民 -->

<package name="cn.liupeng.model"/>

</typeAliases>

<!-- 批量加载映射文件 -->

<mappers>

<package name="cn.liupeng.mapper"/>

</mappers>

</configuration>

##### applicationContext.xml

**读取数据库连接信息**

<context:property-placeholder location="classpath:db.properties" />

#### 配置数据库连接池

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"  
init-method="init" destroy-method="close">

</bean>

#### 数据库连接

<property name="driverClassName" value="${jdbc.driver}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

#### 配置初始化大小、最小、最大

<property name="initialSize" value="1" />

<property name="minIdle" value="1" />

<property name="maxActive" value="10" />

#### 配置SessionFactory

<bean id="sessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">

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

<!--1-->

<!--<property name="configLocation" value="classpath:mybatis.xml" />-->

<!--2-->

<property name="typeAliasesPackage" value="cn.liupeng.model" />

<property name="mapperLocations" value="classpath:cn/liupeng/mapper/\*Mapper.xml" />

</bean>

#### 自动扫描Service

<context:component-scan base-package="cn.liupeng.service" />

#### 事务管理器

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

#### 开启基于注解的事务管理

<tx:annotation-driven transaction-manager="transactionManager"/>

#### 全部代码

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:util="http://www.springframework.org/schema/util"

xmlns:jdbc="http://www.springframework.org/schema/jdbc"

xmlns:cache="http://www.springframework.org/schema/cache"

xsi:schemaLocation="

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/jdbc

http://www.springframework.org/schema/jdbc/spring-jdbc.xsd

http://www.springframework.org/schema/cache

http://www.springframework.org/schema/cache/spring-cache.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/util

http://www.springframework.org/schema/util/spring-util.xsd">

<!-- 读取数据库连接信息 -->

<context:property-placeholder location="classpath:db.properties" />

<!-- 配置数据源 -->

<!--<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<property name="driverClass" value="${jdbc.driver}" />

<property name="jdbcUrl" value="${jdbc.url}" />

<property name="user" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

<property name="minPoolSize" value="2" />

<property name="maxPoolSize" value="30" />

</bean>-->

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"

init-method="init" destroy-method="close">

<property name="driverClassName" value="${jdbc.driver}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

<!-- 配置初始化大小、最小、最大 -->

<property name="initialSize" value="1" />

<property name="minIdle" value="1" />

<property name="maxActive" value="10" />

<!-- 配置获取连接等待超时的时间 -->

<property name="maxWait" value="10000" />

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->

<property name="timeBetweenEvictionRunsMillis" value="60000" />

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->

<property name="minEvictableIdleTimeMillis" value="300000" />

<property name="testWhileIdle" value="true" />

<!-- 这里建议配置为TRUE，防止取到的连接不可用 -->

<property name="testOnBorrow" value="true" />

<property name="testOnReturn" value="false" />

<!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->

<property name="poolPreparedStatements" value="true" />

<property name="maxPoolPreparedStatementPerConnectionSize"

value="20" />

<!-- 这里配置提交方式，默认就是TRUE，可以不用配置 -->

<property name="defaultAutoCommit" value="true" />

<!-- 验证连接有效与否的SQL，不同的数据配置不同 -->

<property name="validationQuery" value="select 1 " />

<property name="filters" value="stat" />

</bean>

<!-- 配置SessionFactory -->

<bean id="sessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">

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

<!--1-->

<!--<property name="configLocation" value="classpath:mybatis.xml" />-->

<!--2-->

<property name="typeAliasesPackage" value="cn.liupeng.model" />

<property name="mapperLocations" value="classpath:cn/liupeng/mapper/\*Mapper.xml" />

</bean>

<!-- 自动生成DAO，Mapper -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer" >

<!--扫描mybatis中的Mapper接口，并自动放入spring容器-->

<property name="basePackage" value="cn.liupeng.mapper" />

<property name="sqlSessionFactoryBeanName" value="sessionFactoryBean" />

</bean>

<!-- 自动扫描Service -->

<context:component-scan base-package="cn.liupeng.service" />

<!--事务管理器-->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

<!--开启基于注解的事务管理-->

<tx:annotation-driven transaction-manager="transactionManager"/>

</beans>

### web.xml 中配置，开始支持Spring

#### 加载 Spring

<!-- 加载 Spring -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

### 修改数据库连接池

#### 为什么修改？

**为什么**

* 3cp0数据库连接池不好使用

##### alibaba 的数据库连接池

* druid

##### maven坐标

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.11</version>

</dependency>

### 有关数据库连接池连接不上去的问题

我的解决方法

*修改一处配置文件*

#### 修改路径

applicationContext.xml

#### 修改内容

原先的状态

<!-- 这里建议配置为TRUE，防止取到的连接不可用 -->

<property name="testOnBorrow" value="true" />

<property name="testOnReturn" value="true" />

修改之后的状态

<!-- 这里建议配置为TRUE，防止取到的连接不可用 -->

<property name="testOnBorrow" value="true" />

<property name="testOnReturn" value="false" />

### 打包问题

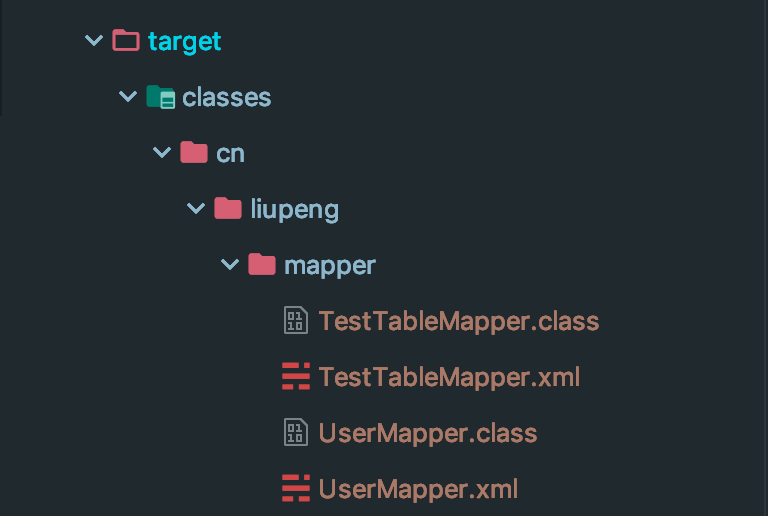
*在DAO层，打包的时候是需要将.xml文件也打包在target文件中的，但是，我的就是没有办法把它打包进去。*

解决办法

#### 方式一 直接复制进去

文件夹目录

* MyDAO/target/classes/cn/liupeng/mapper



3· 37 配置后台[manager]

#### 方式二 修改POM文件

添加代码

<resources>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

<include>\*\*/\*.tld</include>

</includes>

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<filtering>false</filtering>

</resource>

</resources>

添加位置

pom.xml

<project>

<build>

...添加在这个位置...

</build>

</project>

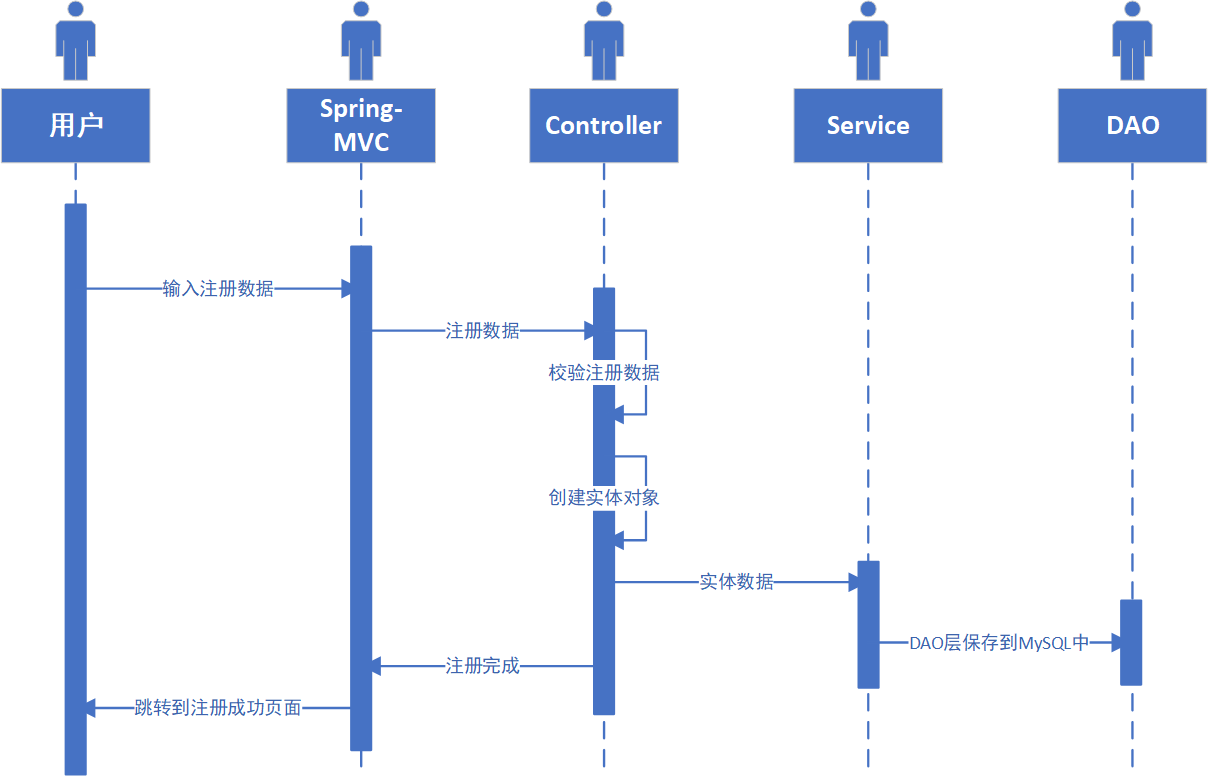
# （四）项目的业务分析

## 用户注册

### 业务描述

* 开始
* 输入注册信息
  1. 用户名
  2. 密码
  3. 确认密码
* Spring MVC 获取传入的数据
* 数据校验
  1. 密码校验
  2. 用户名校验
* 构建实体
* 调用Service层保存数据
* 结束

### 时序图



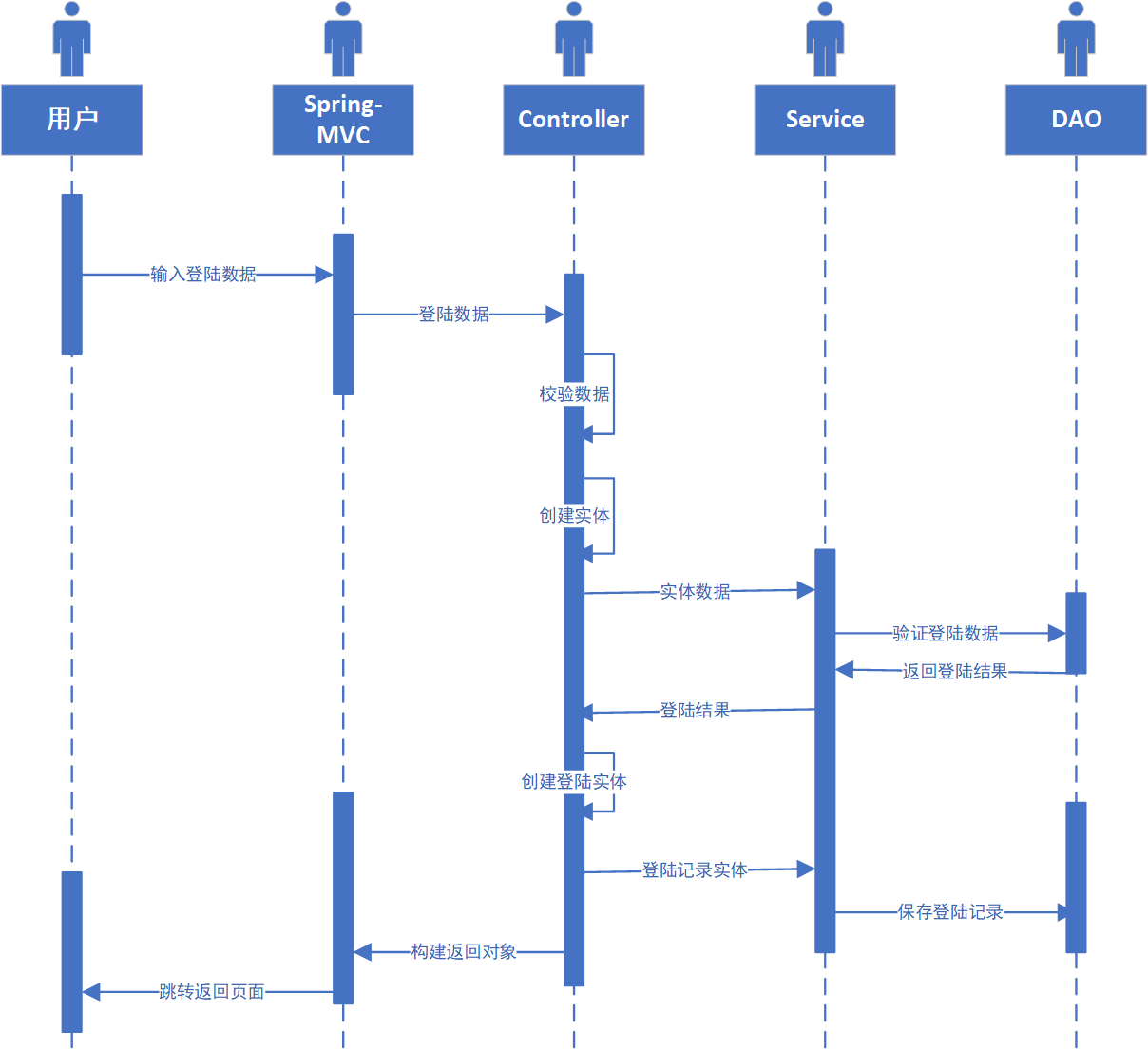
4· 1 用户注册

## 用户登陆

### 业务描述

* 开始
* 输入注册信息
  1. 用户名
  2. 密码
* Spring MVC 获取传入的数据
* 构建实体
* 调用Service层——登陆操作
* 返回登陆结果
* 构建登陆实体
* 调用Service-保存登陆记录
* 返回到登陆结果页面
* 结束

### 时序图



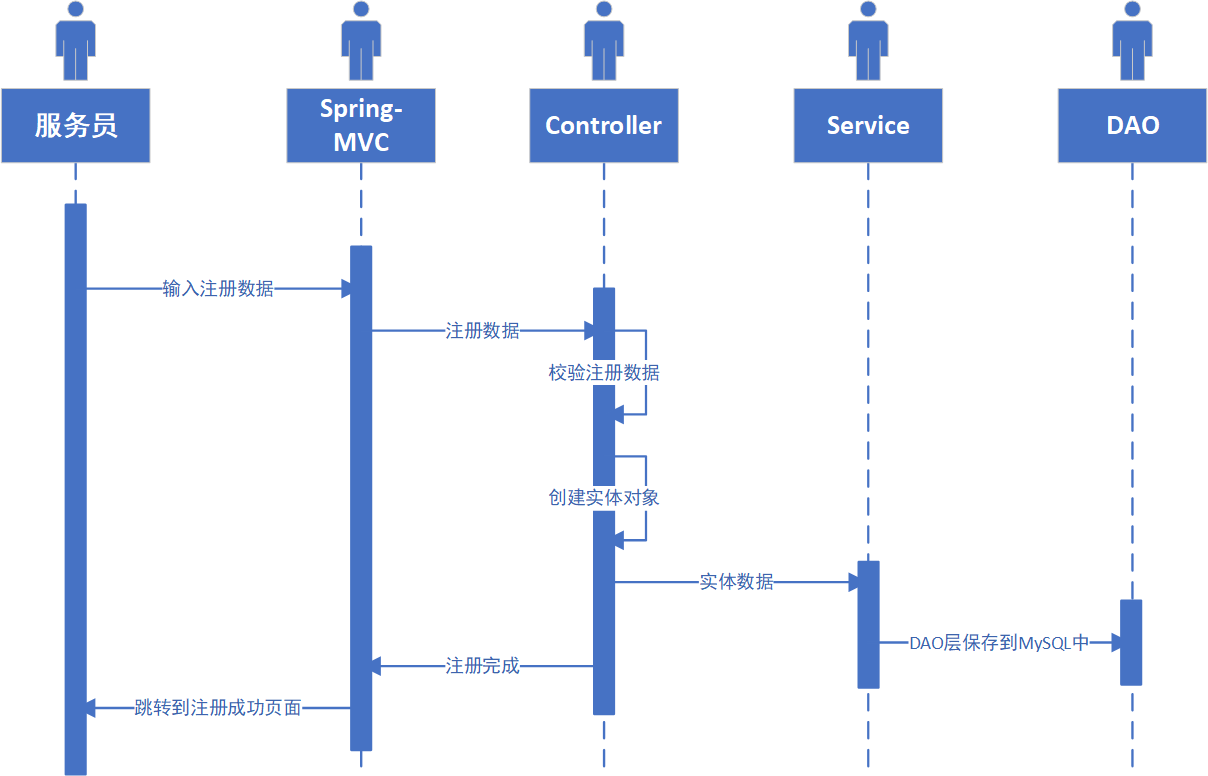
4· 2 用户登陆

## 服务员注册

### 业务描述

* 开始
* 输入注册信息
  1. 用户名
  2. 密码
  3. 确认密码
* Spring MVC 获取传入的数据
* 数据校验
  1. 密码校验
  2. 用户名校验
* 构建实体
* 调用Service层保存数据
* 结束

### 时序图



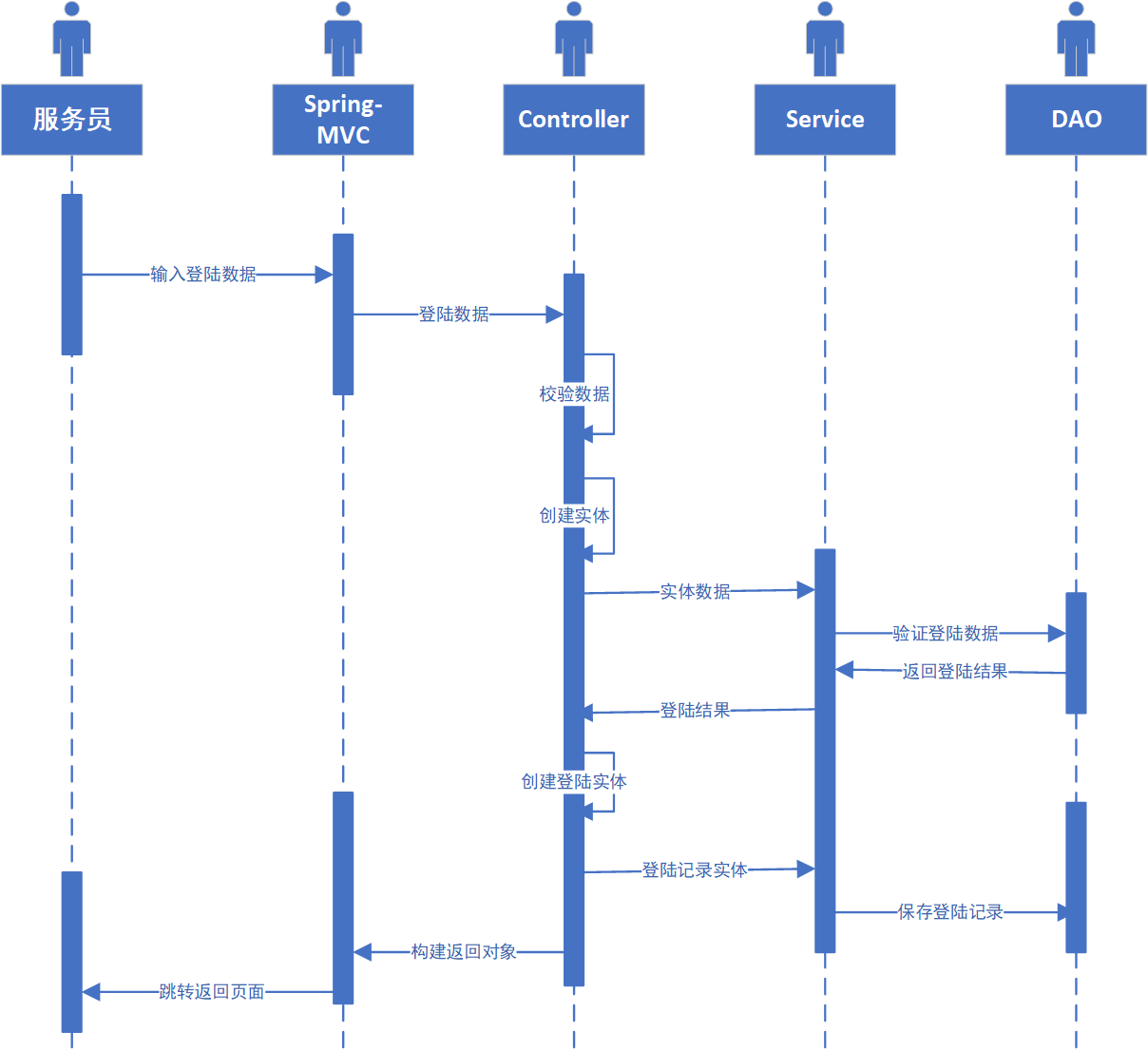
4· 3 服务员注册

## 服务员登陆

### 业务描述

* 开始
* 输入注册信息
  1. 用户名
  2. 密码
* Spring MVC 获取传入的数据
* 构建实体
* 调用Service层——登陆操作
* 返回登陆结果
* 构建登陆实体
* 调用Service-保存登陆记录
* 返回到登陆结果页面
* 结束

### 时序图



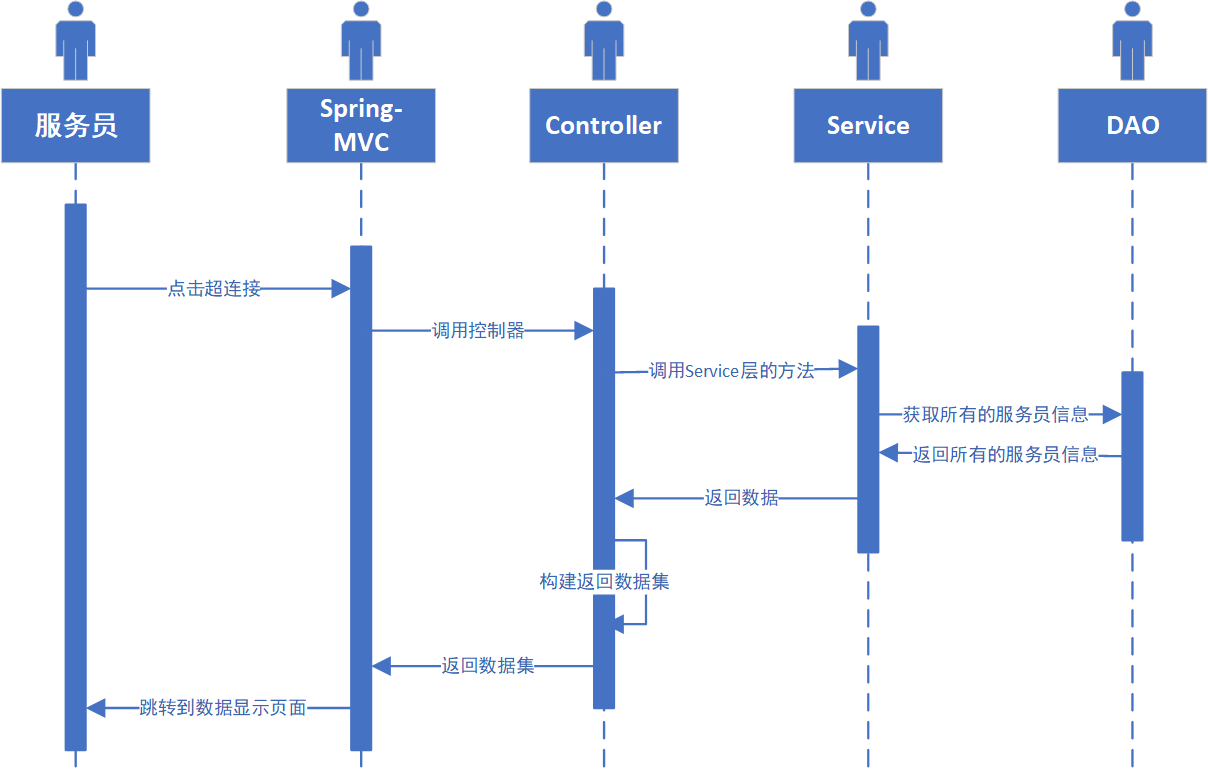
4· 4 服务员登陆

## 获取所有服务员信息

### 业务描述

* 开始
* 调用Service层-获取所有服务员信息
* 返回结果
* 构建万能返回类
* 返回数据到页面
* 结束

### 时序图



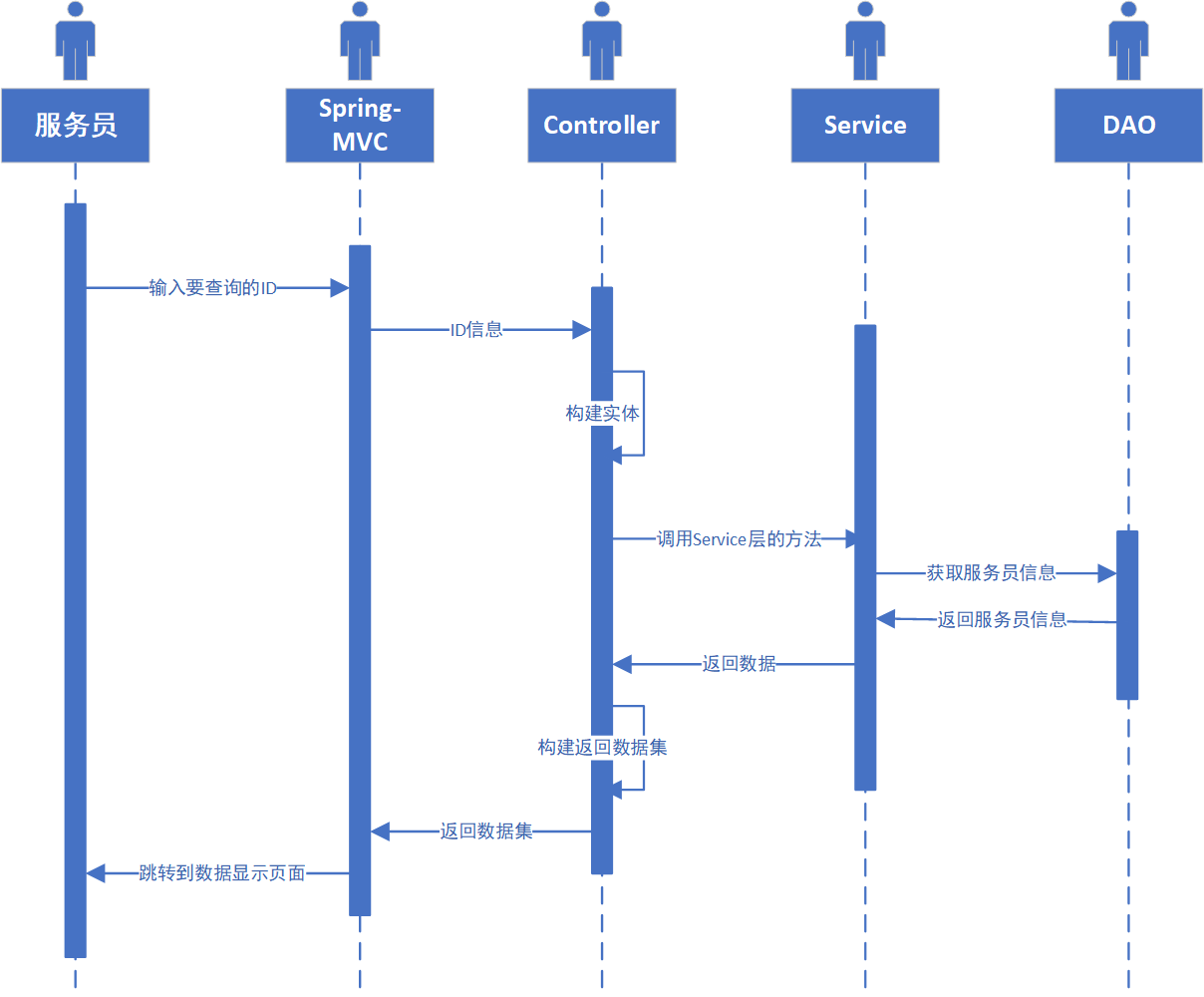
4· 5 获取所有服务员信息

## 通过服务员ID查询服务员

### 业务描述

* 开始
* 输入信息
  1. 服务员ID
* Spring MVC 获取传入的数据
* 调用Service层-获取所有服务员信息
* 返回结果
* 构建万能返回类
* 返回数据到页面
* 结束

### 时序图



4· 6 通过服务员ID查询服务员

## 服务员修改密码

### 业务描述

* 开始
* 获取服务员的
  1. 用户ID
  2. 用户旧密码
* 构建新的实体
* 校验用户是否存在
* 开始校验密码
  1. 调用Service -- 用户DI和密码是否合法
* 结束校验密码
* Spring MVC 获取数据
  1. 用户新密码
* 用户新的确认密码
* 构建新的实体
* 校验新密码的合法性
  1. 开始校验密码
     + 调用工具类--校验密码
  2. 结束校验密码
* 开始修改密码
  1. 调用Service -- 更新密码
* 结束修改密码
* 通知修改成功
* 结束

### 时序图



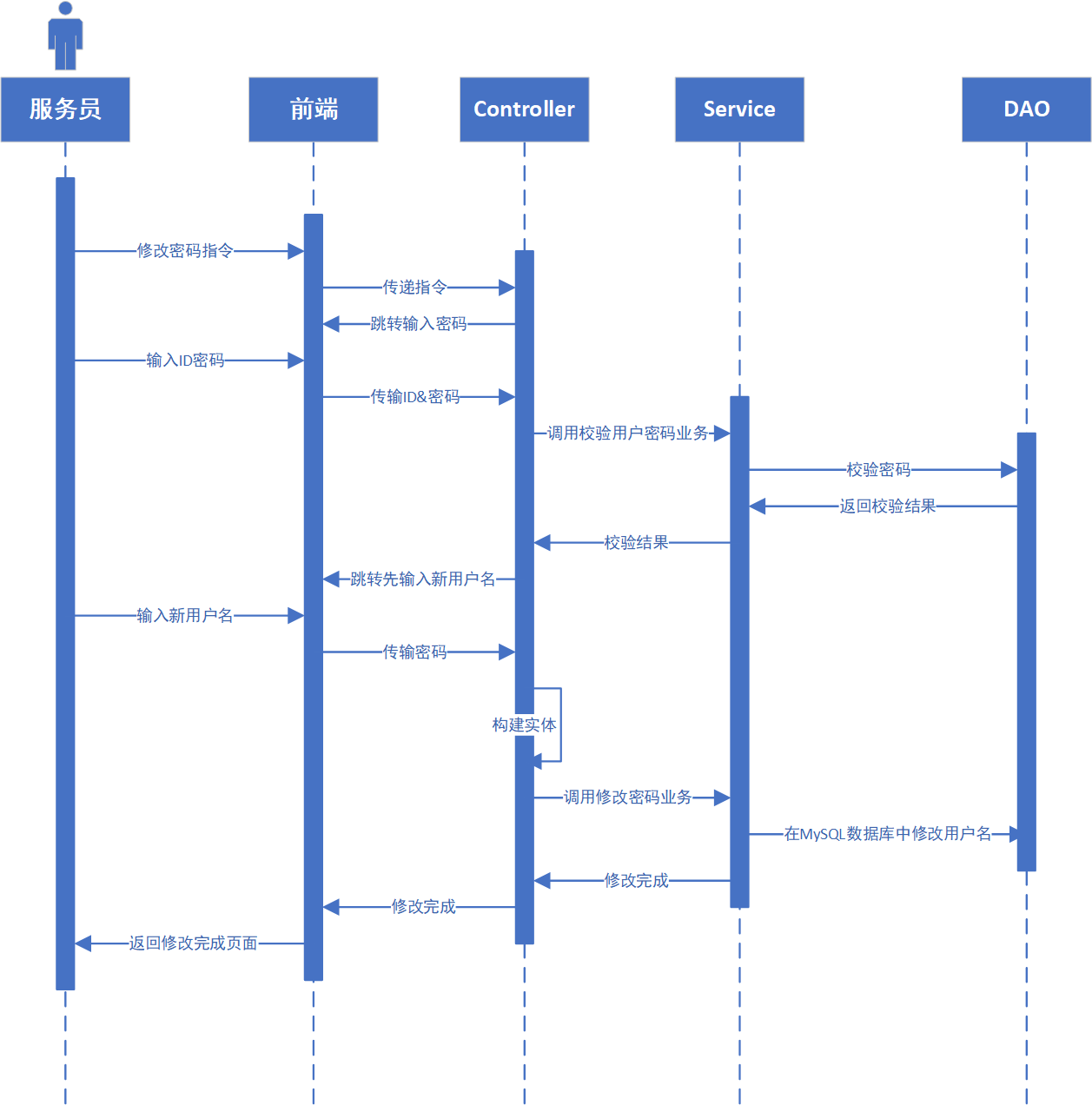
4· 7 服务员修改密码

## 服务员修改用户名

### 业务描述

* 开始
* 获取服务员的
  1. 用户ID
  2. 用户旧密码
* 构建新的实体
* 校验用户是否存在
* 开始校验密码
  1. 调用Service -- 用户DI和密码是否合法
* 结束校验密码
* Spring MVC 获取数据
  1. 用户新用户名
* 构建新的实体
* 开始修改密码
  1. 调用Service -- 更新用户名
* 结束修改密码
* 通知修改成功
* 结束

### 时序图



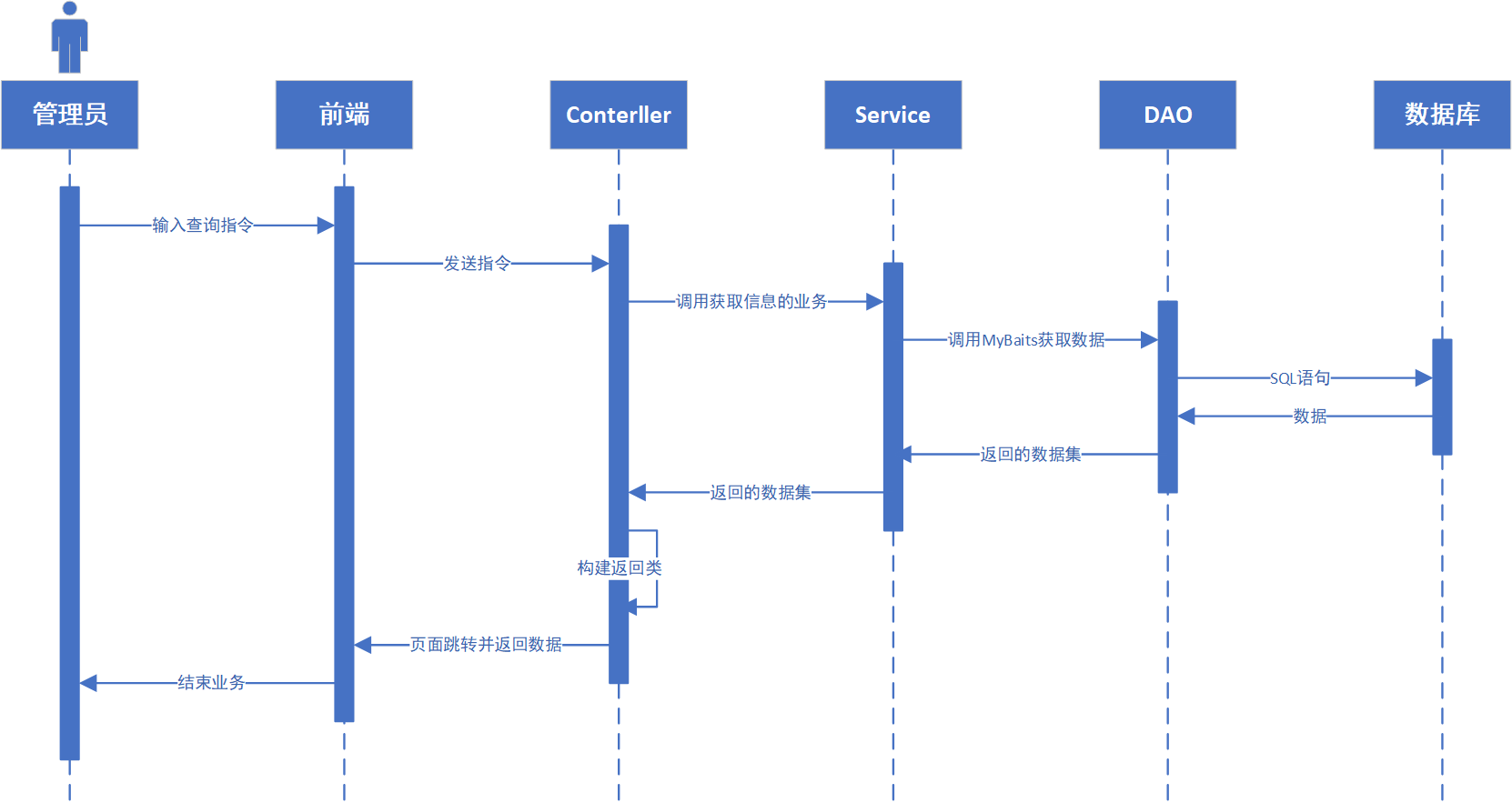
4· 8 服务员修改用户名

## 获取所有的登陆记录

### 业务描述

* 开始
* 调用Service层 -- 获取所有的用户登陆记录
* 返回记录实体集合
* 返回到返回页面
* 结束

### 时序图



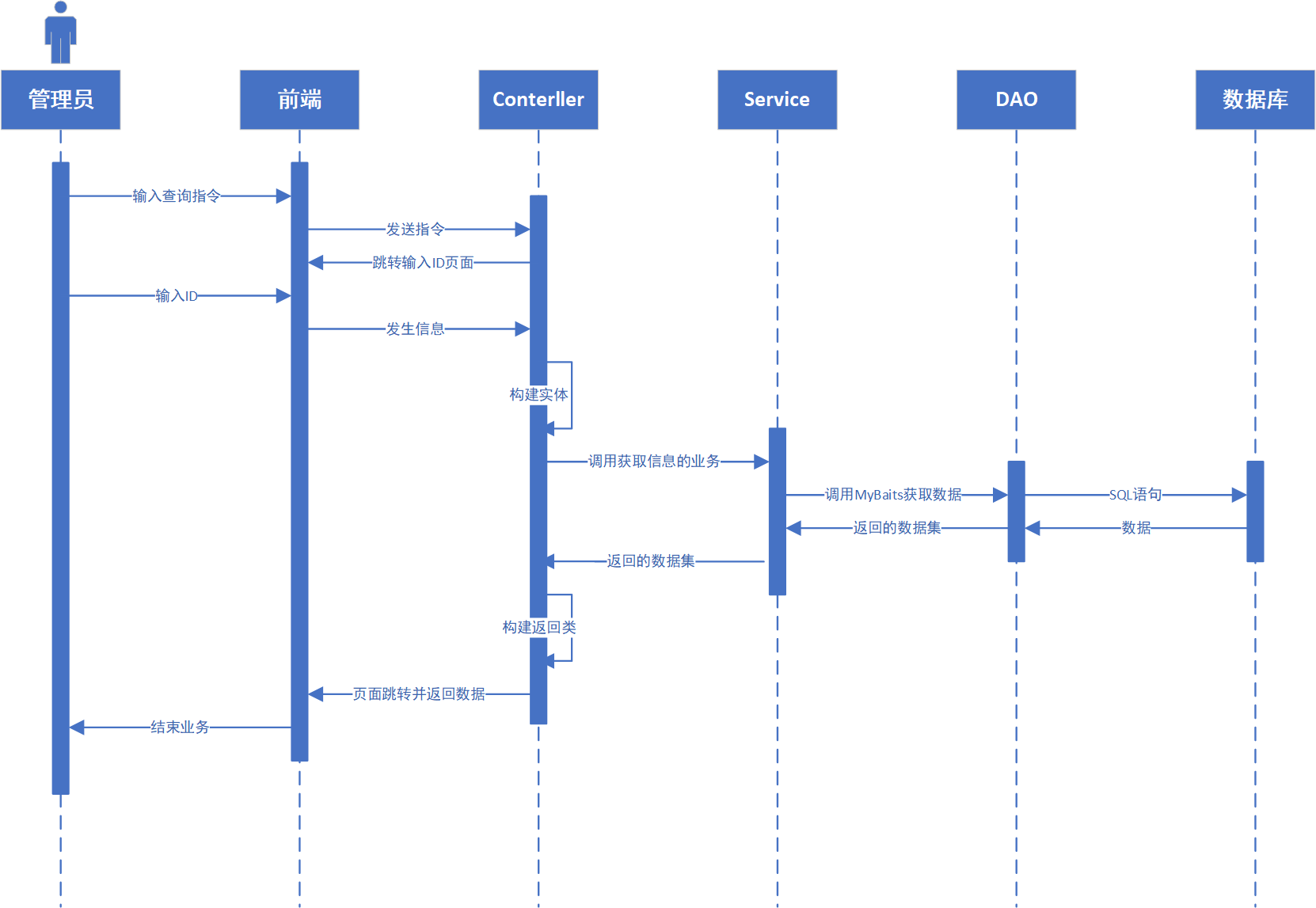
4· 9 获取所有的登陆记录

## 获取所有服务员的登陆记录

### 业务描述

* 开始
* 输入注册信息
  1. 用户ID
* Spring MVC 获取传入的数据
* 构建实体
* 调用Service--获取服务员的登陆记录
* 返回记录实体集合
* 返回到返回页面
* 结束

### 时序图



4· 10 获取所有服务员的登陆记录

# （五）核心代码

## 用户注册

### Controller

#### 跳转到数据输入页面

@RequestMapping(path = "/userRegistration")

public String userRegistration() {

String returnValue = "userRegistration";

return returnValue;

}

#### 注册时用的控制器

@RequestMapping(path = "/registration")

public String registration(String user\_name, String user\_password, String user\_confirm\_password, HttpServletRequest httpServletRequest) throws Exception {

String returnValue = null;

if ("".equals(user\_name)) {

String returninformation = TheGlobalVariable.NOUSERNAME;

} else {

CheakPasswordTool cheakPasswordTool = new CheakPasswordTool(user\_password, user\_confirm\_password);

if (cheakPasswordTool.isKey()) {

returnValue = cheakPasswordTool.getReturnInformation();

String user\_registered\_ip\_adress = new IPAdress(httpServletRequest).getIpAdress();

System.out.println("IP: " + user\_registered\_ip\_adress);

User user = new User();

user.setUser\_id(new CreateUser\_ID().getUser\_id());

user.setUser\_name(user\_name);

user.setUser\_password\_value(new PasswordToPasswordValue(user\_password).getPasswordVlaue() + "");

user.setUser\_registered\_time(new GetDateTime().getDateTime());

user.setUser\_registered\_ip\_adress(user\_registered\_ip\_adress);

System.out.println(user);

System.out.println("用户注册开始！");

userService.registration(user);

System.out.println("用户注册结束！");

} else {

returnValue = cheakPasswordTool.getReturnInformation();

}

}

return returnValue;

}

### Service

/\*\*

\* 用户注册

\* @param user

\*/

@Override

public void registration(User user) {

this.userMapper.addUser(user);

}

### DAO层

#### 接口代码

/\*\*

\* 添加用户

\* @param user

\*/

public void addUser(User user);

#### 映射文件

<insert id="addUser" parameterType="User">

insert user(user\_id, user\_name, user\_password\_value, user\_registered\_time, user\_registered\_ip\_adress)

value

(#{user\_id} , #{user\_name} , #{user\_password\_value} , #{user\_registered\_time} , #{user\_registered\_ip\_adress} );

</insert>

## 用户登陆

### Controller

#### 页面跳转控制器

@RequestMapping(path = "/userLanding")

public String userLanding() {

String returnValue = "userLanding";

return returnValue;

}

#### 登陆控制器

@RequestMapping(path = "/login")

public String login() {

System.out.println("success");

/\*List<TestTable> list = this.testService.listTest();

for (TestTable testTable : list) {

System.out.println("------------");

System.out.println(testTable);

System.out.println("------------");

}\*/

List<User> list = this.userService.listUser();

for (User user : list) {

System.out.println("------------");

System.out.println(user);

System.out.println("------------");

}

return "success";

## 服务员注册

### Controller

#### 跳转到数据输入页面

/\*\*

\* 转向-注册页面

\* @return

\*/

@RequestMapping(path = "/registration")

public ModelAndView waiterRegistration() {

String returnPageName = TheGlobalVariable.WAITERREGISTRATION;

ModelAndView modelAndView = new ModelAndView(returnPageName);

return modelAndView;

}

#### 注册控制器

/\*\*

\* 服务员注册

\* @param waiter\_name

\* @param waiter\_password

\* @param waiter\_confirm\_password

\* @param httpServletRequest

\* @return

\* @throws Exception

\*/

@RequestMapping(path = "/registration")

public String registration(String waiter\_name, String waiter\_password, String waiter\_confirm\_password, HttpServletRequest httpServletRequest) throws Exception {

String returnValue = null;

if ("".equals(waiter\_name)) {

String returninformation = TheGlobalVariable.NOUSERNAME;

} else {

CheakPasswordTool cheakPasswordTool = new CheakPasswordTool(waiter\_password, waiter\_confirm\_password);

if (cheakPasswordTool.isKey()) {

returnValue = cheakPasswordTool.getReturnInformation();

String waiter\_id = new CreateUser\_ID().getUser\_id();

String waiter\_registered\_ip\_adress = new IPAdress(httpServletRequest).getIpAdress();

String waiter\_password\_value = new PasswordToPasswordValue(waiter\_password).getPasswordVlaue() + "";

String waiter\_registered\_time = new GetDateTime().getDateTime();

Waiter waiter = new Waiter();

waiter.setWaiter\_id(waiter\_id);

waiter.setWaiter\_name(waiter\_name);

waiter.setWaiter\_password\_value(waiter\_password\_value);

waiter.setWaiter\_registered\_time(waiter\_registered\_time);

waiter.setWaiter\_registered\_ip\_adress(waiter\_registered\_ip\_adress);

System.out.println(waiter);

System.out.println("waiter注册开始！");

this.waiterService.insert(waiter);

System.out.println("waiter注册结束！");

} else {

returnValue = cheakPasswordTool.getReturnInformation();

}

}

return returnValue;

}

### Service

/\*\*

\* 服务员注册

\* 把服务员实体 通过 DAO 层 保存到数据库中

\* @param waiter

\*/

@Override

public void insert(Waiter waiter) {

this.waiterMapper.addWaiter(waiter);

}

### DAO层

#### 接口代码

/\*\*

\* 添加服务员

\* @param waiter

\*/

public void addWaiter(Waiter waiter);

#### 映射文件

<insert id="addWaiter" parameterType="Waiter">

insert into waiter(waiter\_id, waiter\_name, waiter\_password\_value, waiter\_registered\_time, waiter\_registered\_ip\_adress)

value (#{waiter\_id} , #{waiter\_name} , #{waiter\_password\_value} , #{waiter\_registered\_time} , #{waiter\_registered\_ip\_adress} )

</insert>

## 服务员登陆

### Controller

#### 跳转到数据输入页面

/\*\*

\* 转向-登陆页面

\* @return

\*/

@RequestMapping(path = "/landing")

public ModelAndView waiterLanding() {

String returnPageName = TheGlobalVariable.WAITERLANDING;

ModelAndView modelAndView = new ModelAndView(returnPageName);

return modelAndView;

}

#### 登陆控制器

/\*\*

\* 服务员登陆

\* @param waiter\_id

\* @param waiter\_password

\* @param httpServletRequest

\* @return

\* @throws Exception

\*/

@RequestMapping(path = "/landing")

public String landing(String waiter\_id, String waiter\_password, HttpServletRequest httpServletRequest) throws Exception {

String returnValue = null;

boolean key = false;

// ------------------------------------------------------------------------------------------------

Waiter\_landing waiter\_landing = new Waiter\_landing();

waiter\_landing.setWaiter\_id(waiter\_id);

waiter\_landing.setWaiter\_landing\_time(new GetDateTime().getDateTime());

String waiter\_landing\_ip\_adress = new IPAdress(httpServletRequest).getIpAdress();

waiter\_landing.setWaiter\_landing\_ip\_adress(waiter\_landing\_ip\_adress);

// ------------------------------------------------------------------------------------------------

String returnInformation = null;

String page = "";

if ("".equals(waiter\_id) || "".equals(waiter\_password)) {

// 填写内容不全

returnInformation = TheGlobalVariable.NOTENOUGHLANDINGINFORMATION;

} else {

// 开始操作

String waiter\_password\_value = new PasswordToPasswordValue(waiter\_password).getPasswordVlaue() + "";

Waiter waiter = new Waiter(waiter\_id, waiter\_password\_value);

String waiter\_registered\_ip\_adress = new IPAdress(httpServletRequest).getIpAdress();

waiter.setWaiter\_registered\_ip\_adress(waiter\_registered\_ip\_adress);

key = this.waiterService.landing(waiter);

}

if (key) {

waiter\_landing.setWaiter\_landing\_result(1);

page = TheGlobalVariable.SUCCESSS;

} else {

waiter\_landing.setWaiter\_landing\_result(0);

page = TheGlobalVariable.ERROR;

}

returnValue = TheGlobalVariable.ADDWAITERLANDINGRRECORDPAGE;

addWaiterLanding(httpServletRequest, page, waiter\_landing);

System.out.println(waiter\_landing);

System.out.println("returnValue" + returnValue);

return returnValue;

}

### Service

/\*\*

\* 服务员登陆是否成功

\* ------------------------------------------------------------------

\* 获取用户输入的服务员ID，密码值

\* 用户服务员ID获取数据库中的密码值

\* 匹配俩个密码值是否一致

\* 如果一致：登陆成功

\* 如果不一致：登陆不成功

\* @param waiter

\* @return

\*/

@Override

public boolean landing(Waiter waiter) {

boolean retuanValue = false;

System.out.println("校验密码：" + waiter);

String db\_waiter\_passsword\_value = this.waiterMapper.getWaiterPasswordValueByID(waiter.getWaiter\_id());

if (waiter.getWaiter\_password\_value().equals(db\_waiter\_passsword\_value)) {

retuanValue = true;

} else {

retuanValue = false;

}

return retuanValue;

}

### DAO层

#### 接口代码

/\*\*

\* 通过服务员ID获取密码

\* @param waiter\_id

\* @return

\*/

public String getWaiterPasswordValueByID(String waiter\_id);

#### 映射文件

<select id="getWaiterPasswordValueByID" resultType="string" parameterType="string">

select waiter\_password\_value from waiter where waiter\_id = #{waiter\_id} ;

</select>

## 获取所有服务员

### Controller

/\*\*

\* 获取所有的服务员信息

\* @return

\*/

@RequestMapping(path = "/allWaiter")

public ModelAndView allWaiter() {

List<Waiter> list = this.waiterService.select();

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunFunctionName("获取所有服务员信息");

information.setRunFunctionName("allWaiter()");

information.setHasReturnObject(true);

information.setObjectType("List<Waiter>");

information.setObject(list);

String returnInformation;

if (list.size() > 0) {

returnInformation = "获取到信息";

} else {

returnInformation = "没有获取到信息";

}

information.setReturnInformation(returnInformation);

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.ALLWAITERINFORMATION);

modelAndView.addObject("returnInformation", information);

modelAndView.addObject("listLength", list.size());

return modelAndView;

}

### Service

/\*\*

\* 获取所有服务员信息

\* @return

\*/

@Override

public List<Waiter> select() {

return this.waiterMapper.listWaiter();

}

### DAO层

#### 接口代码

/\*\*

\* 获取所有服务员

\* @return

\*/

public List<Waiter> listWaiter();

#### 映射文件

<select id="listWaiter" resultType="Waiter">

select \* from taobao\_db.waiter;

</select>

## 通过服务员ID查询服务员

### Controller

#### 跳转到数据输入页面

/\*\*

\* 转向-ID查询页面

\* @return

\*/

@RequestMapping(path = "/findWaiterByID")

public ModelAndView findWaiterByID() {

String returnPageName = TheGlobalVariable.FINDWAITERBYID;

ModelAndView modelAndView = new ModelAndView(returnPageName);

return modelAndView;

}

#### 查询控制器

/\*\*

\* 通过服务员ID查找服务员信息

\* @return

\*/

@RequestMapping(path = "/findWaiterByID")

public ModelAndView findWaiterByID(String waiter\_id) {

Waiter waiter = this.waiterService.selectByUUID(waiter\_id);

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunFunctionName("通过服务员ID获取服务员信息");

information.setRunFunctionName("findWaiterByID");

information.setHasReturnObject(true);

information.setObjectType("Waiter");

information.setObject(waiter);

String returnInformation;

if (waiter.equals(null)) {

returnInformation = "没有获取到信息";

} else {

returnInformation = "获取到信息";

}

information.setReturnInformation(returnInformation);

System.out.println(information);

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.FINDONEWAITER);

modelAndView.addObject("returnInformation", information);

return modelAndView;

}

### Service

/\*\*

\* 通过服务员ID查找服务员

\* @param uuid

\* @return

\*/

@Override

public Waiter selectByUUID(String uuid) {

Waiter waiter = this.waiterMapper.findWaiterByID(uuid);

//System.out.println(waiter);

return waiter;

}

### DAO层

#### 接口代码

/\*\*

\* 通过服务员ID获取服务员信息

\* @param waiter\_id

\* @return

\*/

public Waiter findWaiterByID(String waiter\_id);

#### 映射文件

<select id="findWaiterByID" resultType="Waiter" parameterType="string">

select \* from waiter where waiter\_id = #{waiter\_id} ;

</select>

## 服务员修改密码

### Controller

#### 服务员修改密码 - 输入旧密码页面

/\*\*

\* 服务员修改密码 - 输入旧密码页面

\* @return

\*/

@RequestMapping(path = "/inputOrderPassword")

public ModelAndView inputOrdPassword() {

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.WAITERINPUTORDPASSWORD);

return modelAndView;

}

#### 校验密码控制器

/\*\*

\* 校验旧密码

\* @param waiter\_id

\* @param waiter\_password

\* @param httpServletRequest

\* @return

\* @throws Exception

\*/

@RequestMapping(path = "/checkOldPassword")

public ModelAndView checkOldPassword(String waiter\_id, String waiter\_password, HttpServletRequest httpServletRequest) throws Exception {

// 开始操作

boolean key = false;

String waiter\_password\_value = new PasswordToPasswordValue(waiter\_password).getPasswordVlaue() + "";

Waiter waiter = new Waiter(waiter\_id, waiter\_password\_value);

String waiter\_registered\_ip\_adress = new IPAdress(httpServletRequest).getIpAdress();

waiter.setWaiter\_registered\_ip\_adress(waiter\_registered\_ip\_adress);

key = this.waiterService.landing(waiter);

String checkInformation = "";

String returnPage = "";

if (key) {

// 用户ID与密码校验没有问题

checkInformation = TheGlobalVariable.RIGHTIDORPASSWORD;

returnPage = TheGlobalVariable.WAITERINPUTNEWPASSWORD;

// 添加Session 中添加服务员的ID

HttpSession httpSession = httpServletRequest.getSession();

httpSession.setAttribute("waiter\_id", waiter.getWaiter\_id());

} else {

// 用户名或者密码错误

checkInformation = TheGlobalVariable.ERROEIDORPASSWORD;

returnPage = TheGlobalVariable.WAITERERRORPAGE;

}

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunfunction("服务员修改密码，校验ID和密码是否合法和配套");

information.setRunFunctionName("checkOldPassword(String waiter\_id, String waiter\_password, HttpServletRequest httpServletRequest)");

information.setHasReturnObject(false);

information.setReturnInformation(checkInformation);

//System.out.println(information);

ModelAndView modelAndView = new ModelAndView(returnPage);

return modelAndView;

}

#### 修改密码控制器

/\*\*

\* 修改新的密码

\* @param waiter\_password

\* @param waiter\_confirm\_password

\* @param httpServletRequest

\* @ret###urn

\*/

@RequestMapping(path = "/resetPassword")

public ModelAndView resetPassword(String waiter\_password, String waiter\_confirm\_password, HttpServletRequest httpServletRequest) {

HttpSession httpSession = httpServletRequest.getSession();

String waiter\_id = (String) httpSession.getAttribute("waiter\_id");

CheakPasswordTool cheakPasswordTool = new CheakPasswordTool(waiter\_password, waiter\_confirm\_password);

Information information = new Information();

String returnPage = "";

if (cheakPasswordTool.isKey()) {

// 密码合法

//密码转换为密码值

PasswordToPasswordValue passwordToPasswordValue = new PasswordToPasswordValue(waiter\_password);

String waiter\_password\_value = passwordToPasswordValue.getPasswordVlaue() + "";

Waiter waiter = new Waiter();

waiter.setWaiter\_id(waiter\_id);

waiter.setWaiter\_password\_value(waiter\_password\_value);

// 开始修改密码

this.waiterService.resetWaiterPassword(waiter);

returnPage = TheGlobalVariable.WAITERSUCCESSPAGE;

information.setRunfunction("服务员修改密码成功");

information.setReturnInformation(TheGlobalVariable.WAITERRESETPASSWORDSUCCESS);

} else {

// 密码出现了问题，操！

returnPage = TheGlobalVariable.WAITERERRORPAGE;

information.setRunfunction("服务员修改密码.校验新密码");

information.setReturnInformation(cheakPasswordTool.getReturnInformation());

}

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunFunctionName("resetPassword(String waiter\_password, String waiter\_confirm\_password, HttpServletRequest httpServletRequest)");

information.setHasReturnObject(false);

System.out.println(information);

ModelAndView modelAndView = new ModelAndView(returnPage);

modelAndView.addObject("returnInformation", information);

return modelAndView;

}

### Service

/\*\*

\* 修改服务员密码

\* @param waiter

\*/

@Override

public void resetWaiterName(Waiter waiter) {

this.waiterMapper.updateWaiterName(waiter);

}

### DAO层

#### 接口代码

/\*\*

\* 修改服务员密码

\* @param waiter

\*/

public void updateWaiterPassword(Waiter waiter);

#### 映射文件

<update id="updateWaiterPassword" parameterType="Waiter">

update waiter set waiter\_password\_value = #{waiter\_password\_value} where waiter\_id = #{waiter\_id}

</update>

## 服务员修改用户名

### Controller

#### 跳转到数据输入页面

/\*\*

\* 服务员修改用户名 - 输入旧名字页面

\* @return

\*/

@RequestMapping(path = "/inputOrderName")

public ModelAndView inputOrdWaiterName() {

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.INPUTORDNAME);

return modelAndView;

}

#### 校验用户数据的用户名与密码控制器

/\*\*

\* 校验用户数据的用户名与密码

\* @param waiter\_id

\* @param waiter\_password

\* @param httpServletRequest

\* @return

\* @throws Exception

\*/

@RequestMapping(path = "/verifyIDAndPassword")

public ModelAndView verifyIDAndPassword(String waiter\_id, String waiter\_password, HttpServletRequest httpServletRequest) throws Exception {

// 开始操作

boolean key = false;

String waiter\_password\_value = new PasswordToPasswordValue(waiter\_password).getPasswordVlaue() + "";

Waiter waiter = new Waiter(waiter\_id, waiter\_password\_value);

String waiter\_registered\_ip\_adress = new IPAdress(httpServletRequest).getIpAdress();

waiter.setWaiter\_registered\_ip\_adress(waiter\_registered\_ip\_adress);

key = this.waiterService.landing(waiter);

String checkInformation = "";

String returnPage = "";

if (key) {

// 用户ID与密码校验没有问题

checkInformation = TheGlobalVariable.RIGHTIDORPASSWORD;

returnPage = TheGlobalVariable.INPUTNEWNAME;

// 添加Session 中添加服务员的ID

HttpSession httpSession = httpServletRequest.getSession();

httpSession.setAttribute("waiter\_id", waiter.getWaiter\_id());

} else {

// 用户名或者密码错误

checkInformation = TheGlobalVariable.ERROEIDORPASSWORD;

returnPage = TheGlobalVariable.WAITERERRORPAGE;

}

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunfunction("服务员修改用户名，校验ID和密码是否合法和配套");

information.setRunFunctionName("checkOldPassword(String waiter\_id, String waiter\_password, HttpServletRequest httpServletRequest)");

information.setHasReturnObject(false);

information.setReturnInformation(checkInformation);

//System.out.println(information);

ModelAndView modelAndView = new ModelAndView(returnPage);

modelAndView.addObject("returnInformation", information);

return modelAndView;

}

#### 修改服务员用户名

/\*\*

\* 修改服务员用户名

\* @param waiter\_name

\* @param httpServletRequest

\* @return

\* @throws Exception

\*/

@RequestMapping(path = "/resetName")

public ModelAndView resetName(String waiter\_name, HttpServletRequest httpServletRequest) throws Exception {

HttpSession httpSession = httpServletRequest.getSession();

String returnInformation = "";

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunFunctionName("resetName(String waiter\_name, HttpServletRequest httpServletRequest)");

information.setRunfunction("修改服务员密码");

information.setHasReturnObject(false);

String waiter\_id = (String) httpSession.getAttribute("waiter\_id");

Waiter waiter = new Waiter();

waiter.setWaiter\_name(waiter\_name);

waiter.setWaiter\_id(waiter\_id);

if ("".equals(waiter.getWaiter\_name())) {

// 没有输入用户名

information.setReturnInformation(TheGlobalVariable.NONAME);

} else {

// 开始与数据库交互修改用户民

information.setReturnInformation("修改服务员用户名成功");

this.waiterService.resetWaiterName(waiter);

}

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.RESETWAITERNAMESUCCESS);

modelAndView.addObject("returnInformation", information);

return modelAndView;

}

### Service

/\*\*

\* 修改服务员用户名

\* @param waiter

\*/

@Override

public void resetWaiterName(Waiter waiter) {

this.waiterMapper.updateWaiterName(waiter);

}

### DAO层

#### 接口代码

/\*\*

\* 修改服务员名字

\* @param waiter

\*/

public void updateWaiterName(Waiter waiter);

#### 映射文件

<update id="updateWaiterName" parameterType="Waiter">

update waiter set waiter\_name = #{waiter\_name} where waiter\_id = #{waiter\_id}

</update>

## 获取所有的登陆记录

### Controller

#### 获取所有的服务员信息控制器

/\*\*

\* 获取所有的服务员信息

\* @return

\*/

@RequestMapping(path = "/allWaiter")

public ModelAndView allWaiter() {

List<Waiter> list = this.waiterService.select();

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunFunctionName("获取所有服务员信息");

information.setRunFunctionName("allWaiter()");

information.setHasReturnObject(true);

information.setObjectType("List<Waiter>");

information.setObject(list);

String returnInformation;

if (list.size() > 0) {

returnInformation = "获取到信息";

} else {

returnInformation = "没有获取到信息";

}

information.setReturnInformation(returnInformation);

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.ALLWAITERINFORMATION);

modelAndView.addObject("returnInformation", information);

modelAndView.addObject("listLength", list.size());

return modelAndView;

}

### Service

/\*\*

\* 获取所有服务员信息

\* @return

\*/

@Override

public List<Waiter> select() {

return this.waiterMapper.listWaiter();

}

### DAO层

#### 接口代码

/\*\*

\* 获取所有服务员

\* @return

\*/

public List<Waiter> listWaiter();

#### 映射文

<select id="listWaiter" resultType="Waiter">

select \* from taobao\_db.waiter;

</select>

## 通过服务员ID查询服务员

### Controller

#### 跳转到数据输入页面

/\*\*

\* 转向-ID查询页面

\* @return

\*/

@RequestMapping(path = "/findWaiterByID")

public ModelAndView findWaiterByID() {

String returnPageName = TheGlobalVariable.FINDWAITERBYID;

ModelAndView modelAndView = new ModelAndView(returnPageName);

return modelAndView;

}

#### 查询控制器

/\*\*

\* 通过服务员ID查找服务员信息

\* @return

\*/

@RequestMapping(path = "/findWaiterByID")

public ModelAndView findWaiterByID(String waiter\_id) {

Waiter waiter = this.waiterService.selectByUUID(waiter\_id);

Information information = new Information();

information.setRunLocation("cn.liupeng.controller.WaiterController");

information.setRunFunctionName("通过服务员ID获取服务员信息");

information.setRunFunctionName("findWaiterByID");

information.setHasReturnObject(true);

information.setObjectType("Waiter");

information.setObject(waiter);

String returnInformation;

if (waiter.equals(null)) {

returnInformation = "没有获取到信息";

} else {

returnInformation = "获取到信息";

}

information.setReturnInformation(returnInformation);

System.out.println(information);

ModelAndView modelAndView = new ModelAndView(TheGlobalVariable.FINDONEWAITER);

modelAndView.addObject("returnInformation", information);

return modelAndView;

}

### Service

/\*\*

\* 通过服务员ID查找服务员

\* @param uuid

\* @return

\*/

@Override

public Waiter selectByUUID(String uuid) {

Waiter waiter = this.waiterMapper.findWaiterByID(uuid);

//System.out.println(waiter);

return waiter;

}

### DAO层

#### 接口代码

/\*\*

\* 通过服务员ID获取服务员信息

\* @param waiter\_id

\* @return

\*/

public Waiter findWaiterByID(String waiter\_id);

#### 映射文件

<select id="findWaiterByID" resultType="Waiter" parameterType="string">

select \* from waiter where waiter\_id = #{waiter\_id} ;

</select>