# 1. SSM框架整合

所谓SSM框架整合，就是利用Spring将SpringMVC和Mybatis整合到一起，这是现在比较流行的方式。

## 1.1 创建Maven工程

首先新建Maven工程，使用如下pom.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>  <**groupId**>com.zhang</**groupId**>  <**artifactId**>ssm\_demo</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  <**packaging**>war</**packaging**>  <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  *<!-- 定义使用的Spring版本号，因为下面会多次使用到 -->* <**spring-version**>5.0.2.RELEASE</**spring-version**>  </**properties**>  <**dependencies**>  *<!-- SpringMVC的依赖。会自动依赖Spring核心包 -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-webmvc</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- 需要加上Spring的JDBC和事务支持包 -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-jdbc</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>compile</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-tx</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- MySQL的JDBC驱动 -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.42</**version**>  <**scope**>runtime</**scope**>  </**dependency**>  *<!-- druid数据库连接池 -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid</**artifactId**>  <**version**>1.1.6</**version**>  </**dependency**>  *<!-- Mybatis -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis</**artifactId**>  <**version**>3.4.4</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- Mybatis与Spring的整合包 -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis-spring</**artifactId**>  <**version**>1.3.1</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- fastjson -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**>  <**version**>1.2.33</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- servlet-api -->* <**dependency**>  <**groupId**>org.apache.tomcat</**groupId**>  <**artifactId**>tomcat-servlet-api</**artifactId**>  <**version**>8.5.20</**version**>  </**dependency**>  *<!-- jstl -->* <**dependency**>  <**groupId**>javax.servlet</**groupId**>  <**artifactId**>jstl</**artifactId**>  <**version**>1.2</**version**>  </**dependency**>  *<!-- log4j2日志组件 -->* <**dependency**>  <**groupId**>org.apache.logging.log4j</**groupId**>  <**artifactId**>log4j-core</**artifactId**>  <**version**>2.10.0</**version**>  <**scope**>runtime</**scope**>  </**dependency**>  *<!-- junit -->* <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  <**scope**>test</**scope**>  </**dependency**>  *<!-- Spring的test包，为了测试用 -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-test</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>test</**scope**>  </**dependency**>  </**dependencies**>  <**build**>  <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  *<!-- 包括src目录下的xml配置文件 -->* <**include**>\*\*/\*.xml</**include**>  </**includes**>  </**resource**>  *<!-- 既然配置了resources，则还要说明包含的文件夹 -->* <**resource**>  <**directory**>src/main/resources</**directory**>  </**resource**>  <**resource**>  <**directory**>src/main/webapp</**directory**>  </**resource**>  </**resources**>  <**pluginManagement**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**version**>3.6.1</**version**>  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  <**encoding**>UTF-8</**encoding**>  </**configuration**>  </**plugin**>  </**plugins**>  </**pluginManagement**>  </**build**> </**project**> |

注意以上引入的相关包和一些设置。（上述log4j的日志组件是runtime，不能是provided）

接下来就是利用Spring来集成Mybatis和SpringMVC了，重点在于集成Mybatis，SpringMVC的集成和之前类似。目的就是用Spring管理对象的创建，以供不同层使用。

（一）在src/main/java下新建如下包：

1. com.zhang.domain：实体类包；

2. com.zhang.mapper：mapper接口和映射文件所在包；

3. com.zhang.service：业务层包；

4. com.zhang.service.impl：业务层实现包；

5. com.zhang.controller：Controller所在包。

（二）在src/main/resources下新建：

1. spring/applicationContext.xml文件：用于进行Spring的配置；

2. db.properties文件：基本的数据库配置文件；

3. mybatis/config.xml文件：该文件就是之前的Mybatis主配置文件SqlMapConfig.xml，采用Spring配置后，该文件有所精简。

下面进行具体的配置。

## 1.2 Spring和Mybatis的配置（Service层及其下层）

（1）首先，db.properties文件内容如下：

|  |
| --- |
| **driverClassName**=**com.mysql.jdbc.Driver username**=**root password**=**123qwe!@# url**=**jdbc:mysql://114.55.86.230:3306/mytest connectionProperties**=**useUnicode=false;characterEncoding=UTF8;useSSL=false** |

（2）下面配置applicationContext.xml，其中主要配置：

1. 数据源dataSource

2. Mybatis的sqlSessionFactory，即Mybatis的sqlSessionFactory是由Spring管理的。该配置需要利用Mybatis-Spring的包，因此我们之前在pom.xml声明了该依赖包，该包是Mybatis官方提供的。

3. Service和Controller的bean配置。使用注解方式，使得Service能成功使用Mapper对象；Controller能成功使用Service对象。

4. 事务配置。采用注解方式，Mybatis中事务就是使用JDBC提供的事务，在Service层方法中配置事务。

5. SpringMVC配置。

新建spring文件夹，其中的applicationContext.xml具体配置如下：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?> <!-- 注意beans节点中使用的名称空间要正确 -->* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"**>  *<!-- 一、配置数据源 -->  <!-- 1. 加载db.properties文件 -->* <**context:property-placeholder location="classpath:db.properties"** />  *<!-- 2. 使用druid配置数据源 -->* <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  <**property name="url" value="${url}"** />  <**property name="connectionProperties" value="${connectionProperties}"** />  <**property name="username" value="${username}"** />  <**property name="password" value="${password}"** />  <**property name="initialSize" value="1"** />  <**property name="maxActive" value="10"** />  <**property name="minIdle" value="1"** />  <**property name="maxWait" value="60000"** />  <**property name="timeBetweenEvictionRunsMillis" value="60000"** />  <**property name="minEvictableIdleTimeMillis" value="300000"** />  <**property name="testWhileIdle" value="true"** />  <**property name="testOnBorrow" value="false"** />  <**property name="testOnReturn" value="false"** />  <**property name="poolPreparedStatements" value="true"** />  <**property name="maxOpenPreparedStatements" value="20"** />  <**property name="asyncInit" value="true"** />  </**bean**>  *<!-- 二、Mybatis的整合配置 -->  <!-- 1. 配置会话工厂。 -->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  *<!-- 引用数据源 -->* <**property name="dataSource" ref="dataSource"** />  *<!-- Mybatis的配置文件位置 -->* <**property name="configLocation" value="classpath:mybatis/config.xml"** />  </**bean**>  *<!-- 2. 配置mapper。这里使用MapperScannerConfigurer扫描mapper包 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  *<!-- 指定mapper所在包 -->* <**property name="basePackage" value="com.zhang.mapper"** />  *<!-- 指明sqlSessionFactory配置的bean的名称，注意这里不是引用sqlSessionFactory -->* <**property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"** />  </**bean**>  *<!-- 三、配置Service -->  <!-- 只要开启注解扫描即可。注解扫描这里实际上就是为Service服务的，因为需要把他们加入到容器管理中 -->* <**context:component-scan base-package="com.zhang.service"** />  *<!-- 四、事务配置 -->* <**bean id="tx" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  *<!-- 引用dataSource -->* <**property name="dataSource" ref="dataSource"** />  </**bean**>  *<!-- 使用事务注解 -->* <**tx:annotation-driven transaction-manager="tx" proxy-target-class="true"** /> </**beans**> |

配置的说明：

（1）关于数据源，这里提醒一下：在开发时应连接数量应设置地较小，够用即可。避免多台开发电脑连接同一个局域网的数据库时连接数太多。

（2）上述注解扫描只扫描了sevice层，这是因mapper接口根本无需扫描，也不用加上@Repository注解。因为mapper接口已经通过mapper扫描器MapperScannerConfigurer注入Spring中了，注入的mapper代理类的bean名称就是类名（首字母小写，例如employeeMapper）。

这样，mybatis目录下的config.xml文件。由于数据库环境、Mapper类等已经在applicationContext.xml中配置好了，所以该Mybatis主配置文件只需进行“特有”的配置，例如，下面配置了“别名”：

|  |
| --- |
| *<?***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**>  <**typeAliases**>  <**package name="com.zhang.entity"** />  </**typeAliases**> </**configuration**> |

这样，Spring和Mybatis的配置就完成了，接下来进行Service层及其下层的编写和测试工作。

为了演示，我们只先做一个“用户登录”的系统，用户登录后可以查看产品。其余的功能不考虑，按照最简单的来。首先创建user和product表和数据，SQL如下：

|  |
| --- |
| *# 创建user表* **CREATE TABLE user**(  **id VARCHAR**(40),  **account VARCHAR**(40),  **username VARCHAR**(40),  **password VARCHAR**(60),  **age SMALLINT**,  **gender VARCHAR**(20) ); *# 添加一个用于登录的账户，密码用MD5加密* **INSERT INTO user VALUES** (1, **'zs'**, **'张三'**, *md5*(**'123456'**), 12, **'男'**);  *# 创建简单的产品表* **CREATE TABLE** product(  **id VARCHAR**(40),  **name VARCHAR**(40),  **price DECIMAL**(12, 2) ); *# 添加两条产品数据* **INSERT INTO** product **VALUES** (1, **'手机'**, 1699); **INSERT INTO** product **VALUES** (2, **'电池'**, 0.5); |

接下来就开始domain、Service和DAO的开发工作。先完成用户登录的功能。

（1）在com.zhang.domain中编写User类：

|  |
| --- |
| **package** com.zhang.domain;  **import** java.io.Serializable;  *// 用户信息类* **public class** User **implements** Serializable {  *// 用户编号* **private** String **id**;  *// 用户账号* **private** String **account**;  *// 用户姓名* **private** String **username**;  *// 用户密码* **private** String **password**;  *// 年龄* **private** Integer **age**;  *// 性别* **private** String **gender**;  *// 其他属性......    // getter、setter和toString等方法* } |

（2）Service中目前要编写“根据账号拿到用户对象”和“检验用户名和密码是否能正确登录”的方法，因此了解了这个需求后先写Service接口。在com.zhang.service中编写IUserService接口类：

|  |
| --- |
| **package** com.zhang.service;  **import** com.zhang.domain.User;  **public interface** IUserService {   */\*\*  \* 根据账号查询用户  \** ***@param account*** *\** ***@return*** *\*/* User getUserByAccount(String account);   */\*\*  \* 根据账号和密码查询是否能登录  \** ***@param account*** *账号  \** ***@param pwd*** *密码  \** ***@return*** *是否能登录  \*/* **boolean** login(String account, String pwd);  } |

（3）接着编写好DAO层。在com.zhang.mapper中写好UserMapper接口：

|  |
| --- |
| **package** com.zhang.mapper;  **import** com.zhang.domain.User;  **public interface** UserMapper {  */\*\*  \* 根据账号account查询用户  \** ***@param account*** *用户账号account  \** ***@return*** *和账号相关的用户  \*/* User getUserByAccount(String account); } |

然后编写好对应的UserMapper.xml文件：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** <**mapper namespace="com.zhang.mapper.UserMapper"**>  <**select id="getUserByAccount" parameterType="string" resultType="com.zhang.domain.User"** >  SELECT *\** FROM user WHERE account = #{account}  </**select**> </**mapper**> |

（4）最后再回过头来完成UserService的实现。

首先要出注意，DAO中查出的User对象信息，其中的密码是MD5加密的，因此Service中判断能否登录成功时，也需要将调用者传递过来的密码进行MD5加密，再与DAO返回的User对象中的密码信息比对，这样才能最终判断能否登录成功。因此我们先在项目的com.zhang.utils包下引入我们自行开发的工具类MessageDigestUtils，这在以前讲解MD5时使用过，可回顾一下。工具类代码如下：

|  |
| --- |
| **package** com.zhang.utils;  **import** java.security.MessageDigest; **import** java.security.NoSuchAlgorithmException;  **public class** MessageDigestUtils {  **private** MessageDigestUtils() {}   */\*\*  \* 加密。支持MD5和SHA  \** ***@param algorithm*** *加密类型，如MD5、SHA、SHA-256、SHA-512.  \** ***@param data*** *要加密的数据。是一个字节数组，不仅仅可加密字符串资源。  \** ***@param isUpperCase*** *是否将其中的十六进制符号转换成大写的。  \** ***@return*** *加密的字符串值  \*/* **public static** String encrypt(String algorithm, **byte**[] data, **boolean** isUpperCase) {  MessageDigest messageDigest = **null**;  StringBuilder sb = **new** StringBuilder();  **try** {  messageDigest = MessageDigest.*getInstance*(algorithm);  } **catch** (NoSuchAlgorithmException e) {  **throw new** RuntimeException(e);  }  **byte**[] encByte = messageDigest.digest(data);  **for** (**byte** b : encByte) {  **int** i = 0xff & b;  **if** (i < 16) {  sb.append(**"0"**);  }  sb.append(Integer.*toHexString*(i));  }  **return** isUpperCase ? sb.toString().toUpperCase() : sb.toString(); *// 这样写，因为默认十六进制就是小写的* } } |

接下来在com.zhang.service.impl中编写UserServiceImpl实现类（注意使用@Transactional注解）：

|  |
| --- |
| **package** com.zhang.service.impl;  **import** com.zhang.domain.User; **import** com.zhang.mapper.UserMapper; **import** com.zhang.service.IUserService; **import** com.zhang.utils.MessageDigestUtils; **import** org.springframework.stereotype.Service; **import** org.springframework.transaction.annotation.Transactional;  **import** javax.annotation.Resource;  @Service(**"userService"**) **public class** UserServiceImpl **implements** IUserService {   @Resource(name = **"userMapper"**)  **private** UserMapper **userMapper**;   @Override  @Transactional(readOnly = **true**)  **public** User getUserByAccount(String account) {  **return userMapper**.getUserByAccount(account);  }   @Override  @Transactional(readOnly = **true**)  **public boolean** login(String account, String pwd) {  *// 如果某个为null，直接返回false* **if** (account == **null** || pwd == **null**) {  **return false**;  }   **boolean** flag = **false**; *// 是否能成功登录的标记  // 先找到用户* User user = **userMapper**.getUserByAccount(account);  **if** (user == **null**) {  flag = **false**;  } **else** {  *// 判断传递过来的密码使用MD5加密后，能否和查询出的用户密码匹配  // 因为数据库中密码就是以MD5存储的。  // 如果匹配就表示能够登录成功* String pwdMd5 = MessageDigestUtils.*encrypt*(**"MD5"**, pwd.getBytes(), **false**);  flag = pwdMd5.equalsIgnoreCase(user.getPassword());  }  **return** flag;  } } |

至此，Spring与Mybatis的集成，以及用于示例的简单的用户的Service及其以下部分已经编写好了，现在，就需要对上面的整合步骤进行测试，确保上述工作没有问题后，再来整合Web层以跑通Web应用，这是较好的办法，测试非常重要，否则一股脑写完如果产生问题将带来很大麻烦，也不易查找。

因此现在于src/test/java目录中进行上述全部工作的单元测试。新建com.zhang.test包，编写如下ServiceTest类测试代码：

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.domain.User; **import** com.zhang.mapper.UserMapper; **import** com.zhang.service.IUserService; **import** org.junit.Test; **import** org.junit.runner.RunWith; **import** org.springframework.test.context.ContextConfiguration; **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner; **import** org.springframework.transaction.annotation.Transactional; **import** javax.annotation.Resource;  @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(locations={**"classpath:spring/applicationContext.xml"**}) **public class** ServiceTest {  *// 注入需要的属性* @Resource  **private** IUserService **userService**;  @Resource  **private** UserMapper **userMapper**; *// 也测试DAO的Mapper* @Test  @Transactional  **public void** testService() { *// 测试Service  // 测试getUserByAccount* User user = **userService**.getUserByAccount(**"zs"**);  System.***out***.println(user); *// 由于User类重写了toString方法，因此可直接输出查看信息  // 测试* **boolean** login = **userService**.login(**"zs"**, **"123456"**);  System.***out***.println(login);  }   @Test  @Transactional  **public void** testDao() { *// 测试Mapper* User user = **userMapper**.getUserByAccount(**"zs"**);  System.***out***.println(user);  } } |

但使用JUnit运行上述代码却死循环出现下述错误：

|  |
| --- |
| 14:17:14.903 ERROR com.alibaba.druid.support.logging.Log4j2Impl 53 error - create connection SQLException, url: jdbc:mysql://114.55.86.230:3306/mytest, errorCode 1045, state 28000 java.sql.SQLException: Access denied for user 'zmm'@'127.0.0.1' (using password: YES) |

出现这个问题看似是MySQL有问题，可实际上在Mybatis中经过下述的处理即可解决问题：

只要把db.properties中的所有配置键，加上一些“前缀”（例如为“jdbc\_”或“jdbc.”等），再重新在applicationContext.xml中引用就能解决问题。例如这里把db.properties文件修改为：

|  |
| --- |
| **jdbc\_driverClassName**=**com.mysql.jdbc.Driver jdbc\_username**=**root jdbc\_password**=**123qwe!@# jdbc\_url**=**jdbc:mysql://114.55.86.230:3306/mytest jdbc\_connectionProperties**=**useUnicode=false;characterEncoding=UTF8;useSSL=false** |

修改applicationContext.xml对应的项，再运行就成功了，因此说这个可能是Mybatis的BUG，这是一个大坑。

测试结果是，运行testService测试方法输出：

|  |
| --- |
| User{id='1', account='zs', username='张三', password='e10adc3949ba59abbe56e057f20f883e', age=12, gender='男'}  true |

运行testDao测试方法输出：

|  |
| --- |
| User{id='1', account='zs', username='张三', password='e10adc3949ba59abbe56e057f20f883e', age=12, gender='男'} |

上述确保了整合和使用的正确性，下面开展Web的整合和使用工作。

## 1.3 集成SpringMVC

先在src/main/resources/spring目录下配置springmvc.xml文件：

|  |
| --- |
| *<?***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:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"**>  *<!-- SpringMVC的配置 -->  <!-- 1. 扫描controller类的注解 -->* <**context:component-scan base-package="com.zhang.controller"** />  *<!-- 2. SpringMVC驱动配置 -->* <**mvc:annotation-driven**>  <**mvc:message-converters register-defaults="true"**>  *<!--配置fastjson支持-->* <**bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"**>  <**property name="supportedMediaTypes"**>  <**list**>  <**value**>text/html;charset=utf-8</**value**>  <**value**>application/json</**value**>  </**list**>  </**property**>  </**bean**>  </**mvc:message-converters**>  </**mvc:annotation-driven**> </**beans**> |

说明：

（1）上述springmvc中又增加了对controller的扫描，使得能扫描到Controller类。

（2）实际上完全能把springmvc.xml中的配置写到applicationContext.xml中，但这样分开写也比较清晰。也不用在applicationContext.xml文件中直接扫描“com.zhang”下的所有类，这样分成两个包扫描也挺好的，减少了扫描耗时。

最后，只需要在src/main/webapp/WEB-INF/web.xml中配置加载Spring和SpringMVC即可：

|  |
| --- |
| **<!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" *>*** <**web-app**>  *<!-- 设置Spring配置文件 -->* <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:spring/applicationContext.xml</**param-value**>  </**context-param**>  *<!-- 设置Spring监听器 -->* <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  </**listener**>  *<!-- SpringMVC的DispatcherServlet配置 -->* <**servlet**>  <**servlet-name**>springmvc</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  *<!-- SpringMVC配置文件 -->* <**param-value**>classpath:spring/springmvc.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**>  </**servlet**>  *<!-- DispatcherServlet映射 -->* <**servlet-mapping**>  <**servlet-name**>springmvc</**servlet-name**>  <**url-pattern**>\*.action</**url-pattern**>  </**servlet-mapping**> </**web-app**> |

现在，编写Controller和JSP测试Web的整合。

（1）登录页login.jsp代码如下：

|  |
| --- |
| <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> *<%-- 登录页面 --%>* <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" **isELIgnored**="**false**" %> <**c:set var="basePath" value="${**pageContext.request.contextPath**}"** /> <**html**> <**head**>  <**title**>登录</**title**> </**head**> <**body**> <**form action="${**basePath**}/user/login.action" method="POST"**>  账号：<**input name="account" type="text"** />  密码：<**input name="password" type="password"** />  <**input type="submit"** /> </**form**> </**body**> </**html**> |

（2）在com.zhang.controller中编写的UserController类如下（主要就是处理登录，并把登录信息保存在session中，以便用户在该会话中浏览其他页面时能获取信息）：

|  |
| --- |
| **package** com.zhang.controller;  **import** com.zhang.domain.User; **import** com.zhang.service.IUserService; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.RequestMapping;  **import** javax.annotation.Resource; **import** javax.servlet.http.HttpSession;  @Controller @RequestMapping(**"user"**) **public class** UserController {   @Resource(name = **"userService"**)  **private** IUserService **userService**;   *// 登录功能* @RequestMapping(**"login"**)  **public** String login(HttpSession session, String account, String password) {  String returnView = **null**;  *// 检查是否登录成功* **if** (**userService**.login(account, password)) {  *// 登录成功，查出用户信息，并保存到session中* User user = **userService**.getUserByAccount(account);  session.setAttribute(**"user"**, user);  *// 返回功能页面* returnView = **"/funcpage.jsp"**;  } **else** {  *// 否则直接返回登录页面* returnView = **"/login.jsp"**;  }  **return** returnView;  } } |

上述登录成功后返回了一个“funcpage.jsp”页面，该页面用于显示用户的功能菜单，例如查看产品，下面是funcpage.jsp页面：

|  |
| --- |
| <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> *<%-- 功能页面 --%>* <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" **isELIgnored**="**false**" %> <**c:set var="basePath" value="${**pageContext.request.contextPath**}"** /> <**html**> <**head**>  <**title**>功能页面</**title**> </**head**> <**body**>  <**div**>欢迎你，**${**user.username**}**</**div**>  <**ul**>  <**li**>  <**a href="${**basePath**}/product/getAllProducts.action"**>查看员工信息</**a**>  </**li**>  </**ul**> </**body**> </**html**> |

关于产品部分代码还未写，但现在可启动Tomcat运行应用，测试Web应用能否正常跑通运行。结果是应用成功运行，访问应用的login.jsp页面就能完成登录功能。至此SSM示例项目就基本完成了。

至于查看产品的功能，此处就不再贴代码了，实现很简单，只不过是把上述的开发步骤再走一遍，最终只要能简单地查看产品即可。这部分功能已经被我完成，且该项目被我放在github上，地址是：

https://github.com/zhang13690/ssm\_demo

可供参考使用。该示例项目并没有做权限控制，任何人只要知道访问地址就能进入查看数据，这在实际中是不允许的。解决该问题可以通过编写过滤器来实现（我们之前也做过），但完成一个良好的权限模块还是需要考虑很多因素的，实际中公司可能会使用一些安全框架（例如shiro等）来解决权限的问题。因此，在后续的教程中，将基于本项目介绍安全框架shiro的使用，具体见后续教程。

# 2. Mybatis逆向工程（Mybatis Generator）

Mybatis也有逆向工程，就是由数据库表生成POJO类、mapper类和映射文件的过程。Mybatis官方就提供了“Mybatis Generator”工具来生成Mybatis基本文件和代码。Mybatis Generator的介绍和下载可进入Mybatis的GitHub账号寻找。

我们使用下载包中提供的“mybatis-generator-core-1.3.5.jar”工具来自动生成文件。首先我们需要编写一个配置文件，如generatorConfig.xml，该配置文件用来指示Mybatis Generator如何根据数据库表生成对应的实体类、映射文件和mapper接口文件。

这里，配置的内容为：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd"*>*** <**generatorConfiguration**>  *<!-- 配置所引用jar包路径，这里要引用MySQL的JDBC驱动 -->* <**classPathEntry location="D:/DevTool/repository/mysql/mysql-connector-java/5.1.42/mysql-connector-java-5.1.42.jar"** />  *<!-- 配置context -->* <**context id="testTables" targetRuntime="MyBatis3"**>  <**commentGenerator**>  *<!-- 是否去除自动生成的注释 -->* <**property name="suppressAllComments" value="false"** />  </**commentGenerator**>  *<!-- 数据库连接信息 -->* <**jdbcConnection driverClass="com.mysql.jdbc.Driver" connectionURL="jdbc:mysql://localhost:3306/mytest" userId="root" password="123456"** />  *<!-- Java类型转换 -->* <**javaTypeResolver**>  *<!-- 配置forceBigDecimals表示是否将JDBC的DECIMAL和NUMERIC类型解析为Java的BigDecimal类型。这里设置为false，便是不转换成BigDecimal类型，而是转换成Integer类型 -->* <**property name="forceBigDecimals" value="false"** />  </**javaTypeResolver**>  *<!-- 在javaModelGenerator中配置POJO类的生成 -->  <!-- targetPackage表示目标包，targetProject表示目标路径 -->* <**javaModelGenerator targetPackage="com.zhang.domain" targetProject="./src"**>  *<!-- enableSubPackages表示是否让schema作为包的后缀 -->  <!-- schema在下面会配置。一般不用 -->* <**property name="enableSubPackages" value="false"** />  *<!-- 从数据库返回的值将被清理前后的空格 -->* <**property name="trimStrings" value="true"** />  </**javaModelGenerator**>  *<!-- 在sqlMapGenerator中配置映射文件的生成 -->* <**sqlMapGenerator targetPackage="com.zhang.mapper" targetProject="./src"**>  <**property name="enableSubPackages" value="false"** />  </**sqlMapGenerator**>  <**javaClientGenerator type="XMLMAPPER" targetPackage="com.zhang.mapper" targetProject="./src"**>  <**property name="enableSubPackages" value="false"** />  </**javaClientGenerator**>  *<!-- 指定数据库表。这里用user表作为示例 -->* <**table schema="" tableName="user"** />  </**context**> </**generatorConfiguration**> |

这样，我们再用工具和文件生成文件。首先需要在该目录下新建src目录（避免下一步报错），然后在mybatis-generator-core-1.3.5.jar所在目录执行如下命令（即使用该工具）：

|  |
| --- |
| java -jar mybatis-generator-core-1.3.5.jar -configFile generatorConfig.xml |

这样，就在src目录下生成了包文件夹和相关的文件。以后如果想重新生成，需要清空src文件，因为该工具不会清空之前的文件。

我们把生成的文件拷贝到项目中即可使用（自行做调整）。

我们发现生成的mapper中提供了基本的CRUD方法，直接使用即可。除此之外，还自动生成了一个UserExample.java文件，这样的文件是方便开发者进行有条件的选择（相当于where子句），使用方法如下：

|  |
| --- |
| **public** List<User> findById(String id) {  UserExample userExample = **new** UserExample();  userExample.clear();  *// 获得Criteria内部类。条件就封装在其中* UserExample.Criteria criteria = userExample.createCriteria();  *// 可使用多个条件* criteria.andGenderEqualTo(**"男"**);  criteria.andUsernameEqualTo(**"张三"**);  *// 最后查询* **return userMapper**.selectByExample(userExample); } |

当然，也可在update等操作中使用criteria：

|  |
| --- |
| **public void** updateUser() {  *// 修改的员工信息* User newUser = **new** User();  newUser.setUsername(**"李四的姓名"**);  *// 匹配的条件* UserExample userExample = **new** UserExample();  userExample.createCriteria().andIdEqualTo(**"1"**);  **userMapper**.updateByExampleSelective(newUser, userExample); } |

说明：

（1）除了利用“andXxxEqualTo()”方法设置某个字段“相等”的条件，还能使用其他方法如“andXxxBetween()”设置符合的“区间”。其中“Xxx”就是字段名，Mybatis Generator为各个字段都生成了这些方法，简化了开发者工作，大家可自行了解其他方法。

（2）上述更新时使用了“updateByExampleSelective()”方法，这是带有Selective的方法，还有不带Selective的方法。还有其他的很多方法也有带和不带Selective两个“版本”，他们的区别是：不带Selective的方法，会更新传入的POJO对象的所有属性，如果某属性未设置（为null或者空、0等）那么也会更新，因此我们一般会先把原有对象查询出来再更新。但是如果是使用带有Selective的方法，该方法就不会更新未设置的属性（就像例子中只会更新“Name”属性），比较好用。还有updateByPrimary和updateByPrimarySelective方法，类似的。

可以看出，自动生成的代码方便了开发者。但对于一些较复杂的查询等操作，还需要自己写SQL映射。实际中可按需结合Mybatis-Generator使用。

当然，Mybatis Generator网站上还提供了多种方式，例如集成工具到IDE中，但是我们为了不依赖任何东西，就用了上面jar包的方式。