**SPRING day02 笔记**

Authorized by DoHit

# SPRING aop入门

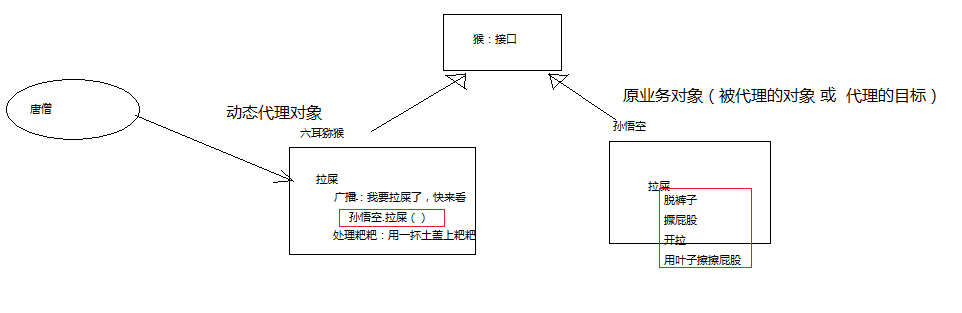
Aop 面向切面编程

在一个大型的系统中，会写很多的业务类--业务方法

同时，一个大型的系统中，还有很多公共的功能：比如事务管理、日志处理、缓存处理.....

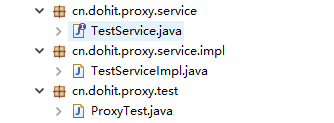
## 动态代理机制复习

### 动态代理的含义示意图



### 动态代理编程实例

工程结构如下：



#### 原业务接口TestService

|  |
| --- |
| **public** **interface** TestService {    **public** String lababa(String babaleixing);  **public** String chifan(String shenmefan);  } |

#### 原业务类TestServiceImpl

|  |
| --- |
| **public** **class** TestServiceImpl **implements** TestService{  @Override  **public** String lababa(String babaleixing) {  System.***out***.println("脱虎皮裙了。。。。");  System.***out***.println("撅屁股了。。。。");  System.***out***.println("开拉了。。。。" +babaleixing);  System.***out***.println("擦屁屁了");    **return** "拉完了";  }  @Override  **public** String chifan(String shenmefan) {  System.***out***.println("上桌。。。。。");  System.***out***.println("抓饭。。。。。");  System.***out***.println("开吃。。。。。" +shenmefan);  System.***out***.println("擦嘴嘴。。。。。。");    **return** "吃完了";  }    } |

#### 动态代理的测试类

ProxyTest

|  |
| --- |
| **public** **class** ProxyTest {    **public** **static** **void** main(String[] args) {  // 首先获取原业务对象的动态代理对象，并且定义动态代理对象中的增强处理逻辑  TestService testServiceImpl = (TestService) Proxy.*newProxyInstance*(TestServiceImpl.**class**.getClassLoader(), TestServiceImpl.**class**.getInterfaces(), **new** InvocationHandler() {  /\*\*  \* 代理对象被调用时的真正处理逻辑所在  \*/  @Override  **public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {  //前置增强  System.***out***.println("开启事务...........");    //调用原业务对象的原方法  TestServiceImpl testService = **new** TestServiceImpl();  // 根据代理对象被调用的方法，去调用相应的原业务对象的原方法  String name = method.getName();  System.***out***.println("当前被调用的是这个方法： " + name);  //调用一下原业务对象的该方法  Object invokeResult = method.invoke(testService, args);    //后置增强  System.***out***.println("提交事务...........");    **return** invokeResult;  }  });    // 调用一下动态代理对象的业务方法  String lababa = testServiceImpl.lababa("稀粑粑");  System.***out***.println("动态代理对象invoke完毕，结果为：" + lababa);  String chifan = testServiceImpl.chifan("满汉全席");  System.***out***.println("动态代理对象invoke完毕，结果为：" + chifan);    }  } |

## aspectJ面向切面编程

### AOP编程框架简介

aspectJ是一个AOP组织提供的面向切面编程框架

它的使用方法是：

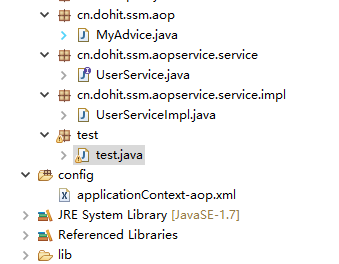
1/ 用户自己开发自己的业务类和业务方法

2/ 用户自己开发自己的增强逻辑，增强逻辑可以写在一个普通Advice类中

3/ 配置一个配置文件，告诉aspectJ，为哪些业务方法增加哪些增强逻辑

### AOP框架编程示例

示例如下：



#### 用户自己的原业务接口

|  |
| --- |
| **public** **interface** UserService {  User findUserById(**int** id);  User findUserByName(String name);  } |

#### 用户自己的原业务类

|  |
| --- |
| **public** **class** UserServiceImpl **implements** UserService {  @Override  **public** User findUserById(**int** id){  System.***out***.println("执行原方法findUserById.......");  User user = **new** User();  user.setId(id);  **return** user;  }    @Override  **public** User findUserByName(String name){  System.***out***.println("执行原方法findUserByName.......");  User user = **new** User();  user.setUsername(name);  **return** user;  }  } |

#### 用户提供的增强逻辑类

|  |
| --- |
| **public** **class** MyAdvice {    **public** **void** before(JoinPoint joinPoint){  System.***out***.println("前增强处理。。。。。。");  }  **public** **void** after(JoinPoint joinPoint){  System.***out***.println("后增强处理。。。。。。");  }  } |

#### AOP切面配置文件

|  |
| --- |
| <bean id=*"userService"* class=*"cn.dohit.ssm.aopservice.service.impl.UserServiceImpl"* />  <bean id=*"myadvice"* class=*"cn.dohit.ssm.aop.MyAdvice"*></bean>  <aop:config>  <aop:aspect ref=*"myadvice"*>  <aop:pointcut expression=*"execution(\* cn.dohit.ssm.aopservice.service.\*.\*(..))"*  id=*"mypoint"* />  <aop:before method=*"before"* pointcut-ref=*"mypoint"* />  </aop:aspect>  </aop:config> |

#### 测试类

|  |
| --- |
| **public** **class** test {  **public** **static** **void** main(String[] args) {  ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("classpath:applicationContext-aop.xml");  // 拿到的这个bean不是原业务类的对象，而是一个动态代理对象  UserService bean = (UserService) context.getBean("userService");  // 调用业务方法时，走的是代理对象中的处理逻辑  User findUserById = bean.findUserById(100);  System.***out***.println(findUserById.getId());    User findUserByName = bean.findUserByName("张三");  System.***out***.println(findUserByName.getUsername());    }  } |

# SSM整合

整合目标：控制层采用springmvc、持久层使用mybatis实现。

各层的BEAN都交给spring管理

本质： 让mybatis层的Mapper类的对象交给spring来构造，在service层的类中需要Mapper对象时，直接从spring注入即可

整个项目的各种业务逻辑的数据库事务管理，也交给spring来处理（aop）

## 需求

实现商品查询列表，从mysql数据库查询商品信息。

## jar包

包括：spring（包括springmvc）、mybatis、mybatis-spring整合包、数据库驱动、第三方连接池。

参考：“mybatis与springmvc整合全部jar包”目录

## 工程搭建

### 整合思路

Dao层：

1. SqlMapConfig.xml，空文件即可。需要文件头。
2. applicationContext-dao.xml。
   1. 数据库连接池
   2. SqlSessionFactory对象，需要spring和mybatis整合包下的。
   3. 配置mapper文件扫描器。

Service层：

1. applicationContext-service.xml包扫描器，扫描@service注解的类。
2. applicationContext-trans.xml配置事务。

表现层：

Springmvc.xml

1. 包扫描器，扫描@Controller注解的类。
2. 配置注解驱动。
3. 视图解析器

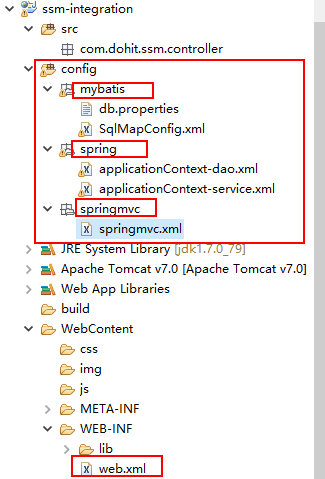
Web.xml

配置前端控制器。

### 整合步骤

#### 导入三大框架的所有jar包

#### 建各类配置文件



#### 配置spring应用容器的方式

1. 修改web.xml——关键点：在web项目中如何启动spring的applicationContext容器

思路：

由于整个工程中所有的controller对象，service对象，dao对象都交给spring框架来构造

但是web工程中，并没有一个main方法来启动spring的applicationContext容器

所以，我们需要让tomcat在启动的时候去启动spring的applicationContext容器

而spring中有两个类（DispatcherServlet和ContextListener）可以被tomcat所加载，并在初始化时创建spring的应用容器

从而，具体做法上有两种：

* 第一种：用两个类去创建两个应用容器来构造项目中的类

把controller对象交给DispatcherServlet去构造

把service和dao层的对象交给ContextListener去构造

DispatcherServlet加载springmvc.xml文件，该文件中只扫描controller层的包

ContextListener加载applicationContext-service.xml,applicationContext-dao.xml两个配置文件，两个配置文件中分别配置service层和dao层的对象构造

在这种方法中，整个项目运行时，会存在两个spring的容器，这两个容器有一个父子关系：

DispatcherServlet的容器是子容器

ContextListener的容器是父容器

子容器可以获取父容器中的对象

* 第二种：用一个容器去加载项目中类

把controller对象、service对象、dao对象全都交给DispatcherServlet去构造

具体做法就是让DispatcherServlet加载所有的bean配置文件

#### 配置web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>ssm-integration</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  </welcome-file-list>    <!-- 加载spring容器 -->  <!--  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-service.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  -->  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc/springmvc.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>\*.action</url-pattern>  </servlet-mapping>  </web-app> |

#### 整合service层

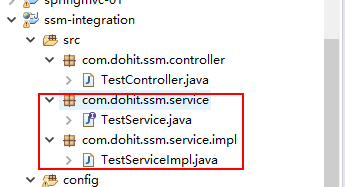
1. 可以让dispatcherServlet创造的容器来构建service层对象
2. 也可以让ContextListener创造的容器来构建service层对象

C、修改applicationContext-service.xml文件：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 组件扫描父包路径 -->  <context:component-scan base-package=*"com.dohit.ssm.service"* />    </beans> |

#### 整合测试controller+service

1、开发一个service层的测试类：



TestServiceImple.java

|  |
| --- |
| @Service  **public** **class** TestServiceImpl **implements** TestService {  @Override  **public** String helloService(String name) {    **return** name+" sb";  }  } |

2、开发表现层controller类：

写一个controller方法，在方法中调用TestServiceImpl中的一个方法

然后在页面上请求一下这个controller的方法

TestController.java

|  |
| --- |
| @Controller  **public** **class** TestController {    @Autowired  **private** TestService testService;    @RequestMapping("/hello")  @ResponseBody  **public** String hello(String name){  **return** "你好："+name;  }    @RequestMapping("/hello2")  @ResponseBody  **public** String hello2(String name){  String res = testService.helloService(name);  **return** "hello： "+res;  }  } |

#### 整合dao层

核心思想：让spring去帮我们构造一个sqlsessionfactory，并且让它去自动扫描mapper接口和xml文件，生成mapper接口的实例对象

做法：

1. 在applicationContext-dao.xml中，配置sqlsessionfactory的bean
2. 先配置一个连接池bean

classpath:mybatis/db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8  jdbc.username=root  jdbc.password=root |

##### applicationContext-dao.xml中

|  |
| --- |
| <!-- 加载properties配置文件 -->  <context:property-placeholder location=*"classpath:mybatis/db.properties"* />  <!-- 数据库连接池 -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*  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}"* />  <property name=*"maxActive"* value=*"10"* />  <property name=*"maxIdle"* value=*"5"* />  </bean> |

##### 配置SqlSessionFactoryBean

|  |
| --- |
| <!-- 让spring管理sqlsessionfactory 使用mybatis和spring整合包中的 -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <!-- 数据库连接池 -->  <property name=*"dataSource"* ref=*"dataSource"* />  <!-- 加载mybatis的全局配置文件 -->  <property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"* />  </bean> |

并且准备一个*"classpath:mybatis/SqlMapConfig.xml"文件*

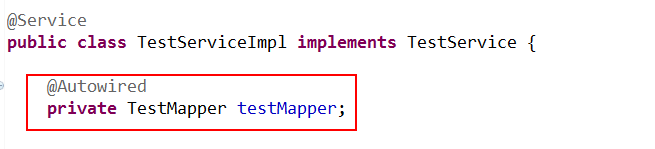
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  </configuration> |

##### 配置mapper扫描器

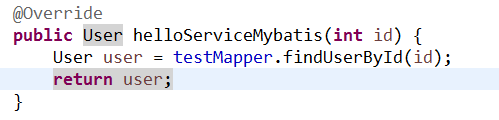
|  |
| --- |
| <!-- 配置Mapper扫描器,mapper扫描器会自动扫描mapper包，生成各种mapper的实例对象，就可以在service中注入这些mapper实例对象了-->  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.dohit.ssm.mapper"/>  </bean> |

##### 测试controller层+service层+mybatis层

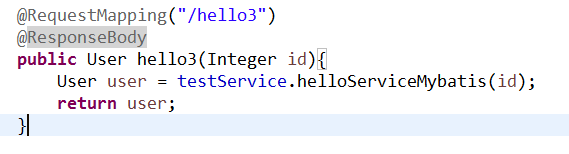
* 在testservice中，注入一个mapper对象



并且，写一个方法来调用testMapper的方法



* 在controller中，写一个方法来调service的这个方法



* 在页面上请求/hello3.action?id=1

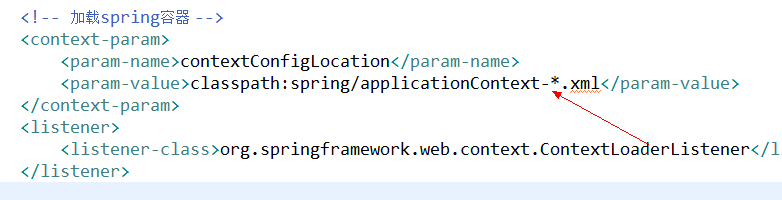
#### Spring声明式事务配置

Spring声明式事务管理配置很简单，只要加一个配置文件到工程中，并且让spring容器去加载该文件即可，配置文件如下

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知：增强 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.dohit.ssm.service.\*.\*(..))"* />  </aop:config>  </beans> |

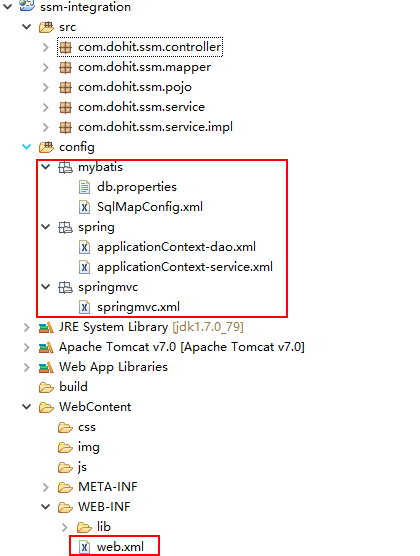
注：文件中的datasource对象引用的是在application-dao.xml中配置好的datasource对象

事务配置文件的加载应该在web.xml中由contextLoaderListener去读取加载：



## 整合完成后的所有配置文件及工程结构

### 工程结构示意图：



### 各配置文件

#### sqlMapConfig.xml

在classpath下创建*mybatis/sqlMapConfig.xml*

|  |
| --- |
| <?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>  </configuration> |

#### applicationContext-dao.xml

配置数据源、配置SqlSessionFactory、mapper扫描器。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd">  <!-- 加载properties配置文件 -->  <context:property-placeholder location="classpath:mybatis/db.properties" />  <!-- 数据库连接池 -->  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"  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}" />  <property name="maxActive" value="10" />  <property name="maxIdle" value="5" />  </bean>  <!-- mapper配置 -->  <!-- 让spring管理sqlsessionfactory 使用mybatis和spring整合包中的 -->  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <!-- 数据库连接池 -->  <property name="dataSource" ref="dataSource" />  <!-- 加载mybatis的全局配置文件 -->  <property name="configLocation" value="classpath:mybatis/SqlMapConfig.xml" />  </bean>    <!-- 配置Mapper扫描器,mapper扫描器会自动扫描mapper包，生成各种mapper的实例对象，就可以在service中注入这些mapper实例对象了-->  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.dohit.ssm.mapper"/>  </bean>  </beans> |

#### db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8  jdbc.username=root  jdbc.password=root |

#### applicationContext-service.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd">  <!-- 组件扫描父包路径 -->  <context:component-scan base-package="com.dohit.ssm.service" />    </beans> |

#### ~~applicationContext-transaction.xml~~

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* cn.dohit.springmvc.service.\*.\*(..))"* />  </aop:config>  </beans> |

#### springmvc.xml

|  |
| --- |
| <?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"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd">  <!-- 加载注解驱动 -->  <mvc:annotation-driven />  <!-- 指定需要扫描controller类的包 -->  <context:component-scan base-package="com.dohit.ssm.controller" />  <!-- 视图解析器配置：前缀和后缀 -->  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="viewClass"  value="org.springframework.web.servlet.view.JstlView" />  <property name="prefix" value="/WEB-INF/jsp/" />  <property name="suffix" value=".jsp" />  </bean>  <!-- 静态资源的自动映射 -->  <mvc:resources location="/js/" mapping="/js/\*\*" />  <mvc:resources location="/img/" mapping="/img/\*\*" />  <mvc:resources location="/css/" mapping="/css/\*\*" />  </beans> |

#### web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>ssm-integration</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  </welcome-file-list>    <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc/springmvc.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>\*.action</url-pattern>  </servlet-mapping>  </web-app> |