# SSM框架整合（Spring+springmvc+mybatis）

## 一、基本概念

### 1.1、Spring

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。 简单来说，Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

Springboot,其他组件交给spring来同一管理

Spring事物管理 ：

编程式事务：写在java代码里面（用的不多），

申明式事务：配置在xml文件里面

通过注解实现

通过配置xml实现

Spring事务的特性，事务的隔离级别

### 1.2、SpringMVC

Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

### 1.3、MyBatis

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。MyBatis是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO）MyBatis 消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。MyBatis 使用简单的 XML或注解用于配置和原始映射，将接口和 Java 的POJOs（Plain Old Java Objects，普通的 Java对象）映射成数据库中的记录。

## 二、SSM框架整合

### 2.1使用的相关版本

使用框架都是较新的版本：

Spring 4.3.3 RELEASE

Spring MVC 4.3.3 RELEASE

MyBatis 3.2.7

### 2.2 整合思路：

1.优先整合sping框架和mybatis框架

2.然后进行spirng和mybatis框架的测试

3.测试成功之后整合spring mvc

### 2.3整个步骤：

#### 第一步：导入相关的maven依赖包

<properties>

<!-- spring版本号 -->

<spring.version>4.3.3.RELEASE</spring.version>

<!-- mybatis版本号 -->

<mybatis.version>3.2.7</mybatis.version>

<!-- log4j日志文件管理包版本 -->

<slf4j.version>1.7.12</slf4j.version>

<log4j.version>1.2.17</log4j.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<!-- 表示开发的时候引入，发布的时候不会加载此包 -->

<scope>test</scope>

</dependency>

<!-- spring核心包 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</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-oxm</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</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-webmvc</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

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

</dependency>

<!-- mybatis核心包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

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

</dependency>

<!-- mybatis/spring包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.2</version>

</dependency>

<!-- 导入java ee jar 包 -->

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-api</artifactId>

<version>7.0</version>

</dependency>

<!-- 导入Mysql数据库链接jar包 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.39</version>

</dependency>

<!-- 导入dbcp的jar包，用来在applicationContext.xml中配置数据库 -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<!-- https://mvnrepository.com/artifact/c3p0/c3p0 -->

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

<!-- JSTL标签类 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- 日志文件管理包 -->

<!-- log start -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<!-- 格式化对象，方便输出日志 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.1.41</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>${slf4j.version}</version>

</dependency>

<!-- log end -->

<!-- 映入JSON -->

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.13</version>

</dependency>

<!-- 上传组件包 -->

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

</dependency>

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.9</version>

</dependency>

</dependencies>

#### 第二步：配置好jdbc.properties,log4j.properties配置文件

1. jdbc.properties

除了用户信息，数据库连接池的配置

jdbc.driverClassName=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/p\_manage?useUnicode=true&characterEncoding=UTF-8

jdbc.username=root

jdbc.password=root

#初始化连接数

jdbc.initialPoolSize=20

#保留的最大连接数

jdbc.maxPoolSize=100

#保留的最小连接数

jdbc.minPoolSize=10

#最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。 Default:0

jdbc.maxIdleTime=600

#当连接池中的连接耗尽的时候c3p0一次同时获取的连接数. Default:3

jdbc.acquireIncrement=5

#JDBC的标准,用以控制数据源内加载的PreparedStatements数量

jdbc.maxStatements=5

#每60秒检查所有连接池中的空闲连接

jdbc.idleConnectionTestPeriod=60

1. log4j.properties

#定义LOG输出级别

log4j.rootLogger=DEBUG,Console,File

#定义日志输出目的地为控制台

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

#可以灵活地指定日志输出格式，下面一行是指定具体的格式

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

log4j.appender.Console.layout.ConversionPattern=[%c] - %m%n

#文件大小到达指定尺寸的时候产生一个新的文件

log4j.appender.File = org.apache.log4j.RollingFileAppender

#指定输出目录

log4j.appender.File.File = logs/springmvcMybist/ssm.log

#定义文件最大大小

log4j.appender.File.MaxFileSize = 10MB

#输出所以日志，如果换成DEBUG表示输出DEBUG以上级别日志

log4j.appender.File.Threshold = ALL

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

log4j.appender.File.layout.ConversionPattern=[%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m%n

#### 第三步：创建mybatis核心配置文件SqlMapConfig.xml

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>

<typeAliases>

<!-- 给整个包取别名,别名对应的文件名 -->

<package name=*"com.hzit.ssm.bean"* />

</typeAliases>

<!-- 使用自动扫描器时，mapper.xml文件如果和mapper.java接口在一个目录则此处不用定义mappers -->

<!--<mappers> -->

<!--<package name="cn.itcast.ssm.mapper" /> -->

<!--</mappers> -->

</configuration>

#### 第四步：将mybatis对象交给sping统一管理。

配置包括数据源获取数据源；

得到SqlSessionFactory；

扫描接口以及mapper映射文件

Spring的事物管理配置

扫描基础包信息

具体配置信息如下：

<?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:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:context=*"http://www.springframework.org/schema/context"*

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

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

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

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

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

<context:component-scan base-package=*"com.hzit.ssm.service,com.hzit.ssm.dao"* />

<!-- 引入配置文件 -->

<bean id=*"propertyConfigurer"*

class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"location"* value=*"classpath:jdbc.properties"* />

</bean>

<!-- 配置数据库连接池 c3p0 -->

<bean id=*"dataSource"* name=*"dataSource"*

class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<!-- 指定连接数据库的驱动 -->

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

<!-- 指定连接数据库的URL -->

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

<!-- 指定连接数据库的用户名 -->

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

<!-- 指定连接数据库的密码 -->

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

<!-- 指定连接池中保留的最大连接数. Default:15 -->

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

<!-- 指定连接池中保留的最小连接数 -->

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

<!-- 指定连接池的初始化连接数 取值应在minPoolSize 与 maxPoolSize 之间.Default:3 -->

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

<!-- 最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。 Default:0 -->

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

<!-- 当连接池中的连接耗尽的时候c3p0一次同时获取的连接数. Default:3 -->

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

<!-- JDBC的标准,用以控制数据源内加载的PreparedStatements数量。 但由于预缓存的statements属于单个connection而不是整个连接池所以设置这个参数需要考虑到多方面的因数.如果maxStatements与maxStatementsPerConnection均为0,则缓存被关闭。Default:0 -->

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

<!-- 每60秒检查所有连接池中的空闲连接.Default:0 -->

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

</bean>

<!-- spring和MyBatis完美整合，不需要mybatis的配置映射文件 -->

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

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

*<!-- 加载mybatis的全局配置文件 -->*

*<!-- <property name="configLocation" value="classpath:SqlMapConfig.xml"*

*/> -->*

*<!-- 配置实体类的别名 -->*

<property name=*"typeAliasesPackage"* value=*"com.hzit.ssm.bean"* />

<!-- 如果想自定义mapper和dao接口的位置，可以在此配置 value:指定的位置 -->

<!-- <property name="mapperLocations" value="classpath:com/ssm/mapping/\*.xml"></property> -->

</bean>

<!-- DAO接口所在包名，Spring会自动查找其下的类 接口和mapper的路径，名称一致，可以自动去查询 -->

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

<property name=*"basePackage"* value=*"com.hzit.ssm.dao"* />

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>

</bean>

<!-- (事务管理)transaction manager, use JtaTransactionManager for global tx -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

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

</bean>

<!-- 基于注解的申明式事物管理 -->

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

</beans>

#### 第五步：Junit测试

1. 创建Sql语句

Sys\_role表：

CREATE TABLE `sys\_role` (

`ROLE\_ID` varchar(32) NOT NULL COMMENT '角色ID',

`ROLE\_NAME` varchar(32) NOT NULL COMMENT '角色名称',

`ROLE\_DESC` varchar(100) DEFAULT NULL COMMENT '角色描述',

`ROLE\_ACTIVE` int(1) NOT NULL COMMENT '是否可用 0---可用 1---不可用',

PRIMARY KEY (`ROLE\_ID`)

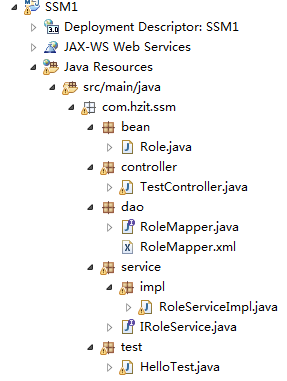
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

添加测试数据：

INSERT INTO `sys\_role` VALUES ('1', '超级管理员', '权限最多的角色', '0');

INSERT INTO `sys\_role` VALUES ('2', '管理员', '很不错的角色', '0');

1. 创建对应的实体类



实体类：Role.java

public class Role {

private String roleId;

private String roleName;

private String roleDesc;

private String roleActive;

//get set

}

1. 创建Mybatis 映射文件，及对应的DAO接口

RoleMapper.java

public interface RoleMapper {

public List<Role> findRole();

}

RoleMapper.xml

<?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.hzit.ssm.dao.RoleMapper"*>

<select id=*"findRole"* resultType=*"Role"*>

SELECT role\_id roleId,role\_name

roleName,role\_desc roleDesc,role\_active

roleActive from sys\_role

</select>

</mapper>

1. 创建service接口及实现类

IRoleService.java

public interface IRoleService {

public List<Role> findRole();

}

RoleServiceImpl.java

//Service 和controller作用一样，表示的是service层

@Service

public class RoleServiceImpl implements IRoleService {

//自动注入对象

@Autowired

private RoleMapper roleMapper;

public List<Role> findRole() {

return roleMapper.findRole();

}

}

1. 新建HelloTest.java测试类

Controller--->service-->dao

@RunWith(SpringJUnit4ClassRunner.class) // 表示继承了SpringJUnit4ClassRunner类

@ContextConfiguration(locations = { "classpath:applicationContext.xml" })

public class HelloTest {

//得到日志文件对象

private static Logger *logger* = Logger.*getLogger*(HelloTest.class);

@Autowired

private IRoleService roleService = null;

@Test

public void test() {

List<Role> roleList = roleService.findRole();

// System.out.println(user.getUserName());

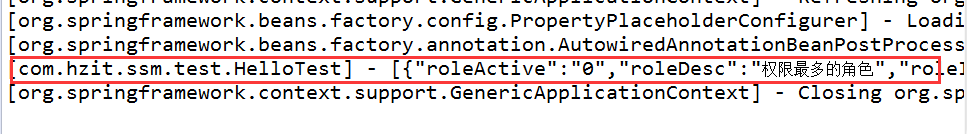
// logger.info("值："+user.getUserName());

*logger*.info(JSON.*toJSONString*(roleList));

}

}

1. 如果测试成功，表示Spring和Mybatis已经整合成功了。输出信息使用的是Log4j打印到控制台。



#### 第六步：整合springmvc

上面已经完成了2大框架的整合，SpringMVC的配置文件单独放，然后在web.xml中配置整合。

1. 在resource目录下面创建springmvc.xml配置文件

包括扫描Controller；

加载驱动信息；

设置试图解析器；

过滤静态文件；

配置文件上传信息；

<?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:mvc=*"http://www.springframework.org/schema/mvc"*

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 "*>

<!-- 扫描controller所在的包 -->

<context:component-scan base-package=*"com.hzit.ssm.controller"* />

<!-- 加载驱动信息 -->

<mvc:annotation-driven />

<!-- 配置视图解析器 -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"viewClass"*

value=*"org.springframework.web.servlet.view.JstlView"* />

<!-- 设置前缀的位置 -->

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

<!-- 设置后缀 -->

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

</bean>

<!--过滤静态文件 -->

<mvc:resources mapping=*"/resources/\*\*"* location=*"/resources/"* />

<mvc:default-servlet-handler />

<!-- 文件上传 -->

<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 设置上传文件的最大尺寸为5MB -->

<property name=*"maxUploadSize"*>

<value>5242880</value>

</property>

</bean>

</beans>

1. 配置web.xml配置文件

监听sping监听器；

加载applicationContext.xml配置文件

配置springmvc 中央控制器

配置POST乱码处理

具体配置如下：

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\_3\_0.xsd"*

version=*"3.0"*>

<display-name>Archetype Created Web Application</display-name>

<!-- Spring和mybatis的配置文件 -->

<context-param>

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

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

</context-param>

<!-- Spring监听器 -->

<listener>

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

</listener>

<!-- Spring MVC servlet -->

<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.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>SpringMVC</servlet-name>

<!-- 此处可以可以配置成\*.do，对应struts的后缀习惯 -->

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

</servlet-mapping>

<!-- 编码过滤器 -->

<filter>

<filter-name>encodingFilter</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>encodingFilter</filter-name>

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

</filter-mapping>

<welcome-file-list>

<welcome-file>/index.html</welcome-file>

<welcome-file>/index.jsp</welcome-file>

</welcome-file-list>

</web-app>

#### 第七步：整体测试

已经完成了SSM三大框架的整合了，接下来测试一下

创建TestController.java类测试

@Controller

public class TestController {

@Autowired

private IRoleService roleService;

@RequestMapping("hello")

public String helloController(Model model) {

List<Role> findRole = roleService.findRole();

model.addAttribute("role", findRole.get(0));

return "Hello";

}

}

在WEB-INF/jsp/ 下面创建Hello.jsp



Hello.jsp

<body>

角色ID:${role.roleId} 角色名称:${role.roleName}

</body>

运行：<http://localhost:8080/SSM1/hello> 显示相关信息



SpringMVC中四个基本注解：

[@Component、@Repository   @Service、@Controller](http://hi.baidu.com/huahua035/item/57746f2510e342d10e37f9f2)

@Autowired:自动注入属性值，例如写在empService上面，代表注入这个属性

@Autowired

private EmpService empService;

看字面含义，很容易却别出其中三个：

@Controller   控制层，就是我们的action层

@Service        业务逻辑层，就是我们的service或者manager层

@Repository  持久层，就是我们常说的DAO层

@Resource 资源

而@Component  （字面意思就是组件），它在你确定不了是哪一个层的时候使用。

其实，这四个注解的效果都是一样的，Spring都会把它们当做需要注入的Bean加载在上下文中；

但是在项目中，却建议你严格按照除Componen的其余三个注解的含义使用在项目中。这对分层结构的web架构很有好处！！

数据库连接池