# 尚马教育 JAVA 高级课程

# SSM整合

文档编号：C07

创建日期： 2017-07-07

最后修改日期：2019-10-18

版 本 号：V3.5

电子版文件名：尚马教育-第三阶段-7.ssm框架整合.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-07-25 | 王绍成 | Mybatis版本更新 | V2.0 |
| 2019-10-18 | 冯勇涛 | 课件格式以及课程深度加深 | V3.0 |
| 2021-02-22 | 冯勇涛 | 调整课件 | V3.5 |

目录

[尚马教育 JAVA 高级课程 1](#_Toc4983)

[SSM整合 1](#_Toc7425)

[1. 整合步骤 3](#_Toc7495)

[2. 导入JAR包 3](#_Toc566)

[3. 建立开发目录 4](#_Toc11474)

[4. 了解两个容器 4](#_Toc16568)

[5. 配置SpringMVC子容器 5](#_Toc10760)

[6. 配置Spring父容器 5](#_Toc7812)

[7. 配置监听器与前段控制器 6](#_Toc4472)

[8. 配置MyBatis 7](#_Toc25167)

[8.1. 主配置 7](#_Toc25061)

[8.2. Druid连接池完整配置 9](#_Toc32581)

[8.3. 配置事务管理器 10](#_Toc15212)

[8.3.1. 注解式事务 10](#_Toc20494)

[8.3.2. 配置式事务 10](#_Toc25880)

[9. Mybatis Generator配置 11](#_Toc25244)

[9.1. MyEclipse 11](#_Toc21459)

[9.1.1. 安装 11](#_Toc9173)

[9.1.2. 使用 12](#_Toc26128)

[9.1.3. 运行 14](#_Toc28944)

[9.2. IDEA 15](#_Toc25000)

[第三步：从官网复制生成代码如下： 15](#_Toc1794)

[10. PageHelper分页插件 16](#_Toc28097)

[10.1. 配置dao.xml文件 16](#_Toc16527)

[10.2. Mapper.xml 17](#_Toc20657)

[10.3. Handler 17](#_Toc28053)

[10.4. 属性介绍 18](#_Toc14911)

## 整合步骤

添加spring，springmvc，mybatis的相关jar支持

建立开发目录（handler，service，mapper，entity等）

配置springmvc

配置spring

配置web.xml

配置mybatis

## 导入JAR包

* 导入Spring和SpringMVC相关的Jar
* 导入JackSon相关的Jar(或者fastjson)
* 导入文件上传的两个Jar
* 导入AOP的Jar
* 导入Mybatis的Jar包
* 其他Jar包支持
  + 连接池 :druid.jar(获取其他连接池)
  + 日志相关 :commons-logging-1.1.1.jar;
  + mybatis和Spring事务整合 : mybatis-spring-1.3.0.jar;spring-tx;spring-jdbc
  + 数据库链接驱动: mysql-connector-java-5.1.40-bin.jar

## 建立开发目录

|  |
| --- |
|  |

## 了解两个容器

在spring-webmvc包下的DispatcherServlet对象中创建的WebApplicationContext对象持有了spring-web包下的ContextLoaderListener对象创建的WebApplicationContext对象。

* 把DispatcherServlet内创建的容器称为springmvc子容器，保存控制层bean对象。
* 把ContextLoaderListener内创建的容器称为spring父容器，保存service、dao、其他层bean对象。

在源码中发现子容器持有父容器，即子容器中的bean对象可以依赖父容器中的bean。换言之，controller层依赖service层。这种设计目的是springMVC作为一个控制层框架，职责清晰化，仅管理控制层bean对象。

## 配置SpringMVC子容器

在resources目录下新建SpringMVC配置文件:springmvc.xml

|  |
| --- |
| *<!--spring注解识别，只识别Controller注解，springmvc控制层框架职责清晰化-->* <**context:component-scan base-package="com.javasm" use-default-filters="false"**>  <**context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"**></**context:include-filter**> </**context:component-scan**> *<!--springmvc注解识别-->* <**mvc:annotation-driven**></**mvc:annotation-driven**>  *<!--文件上传-->* <**bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver"**>  <**property name="defaultEncoding" value="utf-8"**></**property**> </**bean**> |

## 配置Spring父容器

在resources目录下新建SpringMVC配置文件:spring.xml

|  |
| --- |
| <**context:component-scan base-package="com.javasm"**>  <**context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"**></**context:exclude-filter**> </**context:component-scan**> *<!--如果我们要向service层的对象织入切面，这里做aop-->* <**aop:aspectj-autoproxy**></**aop:aspectj-autoproxy**> |

## 配置监听器与前段控制器

|  |
| --- |
| <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:spring.xml</**param-value**> </**context-param**>  *<!--tomcat服务器的启动监听器-->* <**listener**> <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**> </**listener**>  *<!--springMVC前端控制器-->* <**servlet**>  <**servlet-name**>dispatcherServlet</**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**>dispatcherServlet</**servlet-name**>  <**url-pattern**>/</**url-pattern**> </**servlet-mapping**> |

## 配置MyBatis

### 主配置

创建jdbc.properties文件，维护数据库连接信息。

创建dao.xml的配置文件,也是一个spring的配置文件，专门存放mybatis和spring的整合配置。在spring.xml中通过import标签引入dao.xml即可。

Jdbc.properties如下：

|  |
| --- |
| **jdbc.user** = **root jdbc.password** = **root jdbc.url** = **jdbc:mysql://127.0.0.1:3306/704a?characterEncoding=utf8&useSSL=true jdbc.driver** = **com.mysql.jdbc.Driver jdbc.initSize** =**3** |

Spring.xml如下：

|  |
| --- |
| *<!--mybatis集成文件-->* <**import resource="classpath:dao.xml"**></**import**> |

dao.xml如下：

|  |
| --- |
| *<!--加载jdbc.properties文件到spring上下文-->* <**context:property-placeholder location="classpath:jdbc.properties" ignore-unresolvable="true"**></**context:property-placeholder**>  *<!--创建数据库连接池对象-->* <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  <**property name="url" value="${jdbc.url}"**></**property**>  <**property name="driverClassName" value="${jdbc.driver}"**></**property**>  <**property name="username" value="${jdbc.user}"**></**property**>  <**property name="password" value="${jdbc.password}"**></**property**>  <**property name="initialSize" value="${jdbc.initSize}"**></**property**> </**bean**>  *<!--当该bean对象被创建出来以后，立即执行afterPropertiesSet方法，初始化SqlSessionFactory到spring上下文-->* <**bean id="sessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="mapperLocations" value="classpath:com/javasm/\*/mapper/\*.xml"**> </**property**>  <**property name="typeAliasesPackage" value="com.javasm"**></**property**>  <**property name="configLocation" value="classpath:mybatis.xml"**></**property**> </**bean**>  *<!--对指定包下的接口创建代理对象，注册到spring上下文-->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="com.javasm.\*.mapper"**></**property**> </**bean**> |

### Druid连接池完整配置

<!-- 数据源 Druid-->

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

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

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

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

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

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

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

<!-- 配置初始化大小、最小、最大 -->

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

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

<property name="maxActive" value="50" />

<!-- 配置获取连接等待超时的时间 -->

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

<!-- 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" />

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

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

</bean>

### 配置事务管理器

|  |
| --- |
| *<!--配置事务管理器对象，id必须是transactionManager-->* <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  <**property name="dataSource" ref="dataSource"**></**property**> </**bean**> |

#### 注解式事务

开启事务注解*Transactional*识别，该注解使用在service层的方法上。

|  |
| --- |
| *<!--依赖于id为transactionManager的事务管理器对象，对带有@Transactional注解的方法做织入-->* <**tx:annotation-driven**></**tx:annotation-driven**> |

#### 配置式事务

|  |
| --- |
| *<!--事务通知，以及事务生效规则-->* <**tx:advice id="txAdvice"**>  <**tx:attributes**>  <**tx:method name="add\*" propagation="REQUIRED"**/>  <**tx:method name="insert\*" propagation="REQUIRED"**/>  <**tx:method name="update\*" propagation="REQUIRED"**/>  <**tx:method name="del\*" propagation="REQUIRED"**/>  <**tx:method name="select\*" propagation="SUPPORTS"**></**tx:method**>  <**tx:method name="query\*" propagation="SUPPORTS"**></**tx:method**>  </**tx:attributes**> </**tx:advice**> *<!--事务通知织入到切入点中-->* <**aop:config**>  <**aop:pointcut id="servicePointcut" expression="execution(\* com.javasm.\*.service.\*.\*(..))"**></**aop:pointcut**>  <**aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut"**></**aop:advisor**> </**aop:config**> |

## 配置跨域过滤器

在resources下创建cors.xml，并导入到spring.xml中。

### 注册CorsFilter到spring上下文

|  |
| --- |
| <**bean id="corsFilter" class="org.springframework.web.filter.CorsFilter"**>  <**constructor-arg name="configSource"**>  <**bean class="org.springframework.web.cors.UrlBasedCorsConfigurationSource"**>  <**property name="corsConfigurations"**>  <**map**>  <**entry key="/\*\*"**>  <**bean class="org.springframework.web.cors.CorsConfiguration"**>  <**property name="allowCredentials" value="true"**/>  <**property name="allowedMethods"**>  <**list**>  <**value**>GET</**value**>  <**value**>POST</**value**>  <**value**>HEAD</**value**>  <**value**>PUT</**value**>  <**value**>DELETE</**value**>  <**value**>OPTIONS</**value**>  </**list**>  </**property**>  <**property name="allowedHeaders" value="\*"**/>  <**property name="allowedOrigins" value="\*"**/>  </**bean**>  </**entry**>  </**map**>  </**property**>  </**bean**> </**constructor-arg**> </**bean**> |

### 配置web过滤器

在web.xml中配置CorsFilter

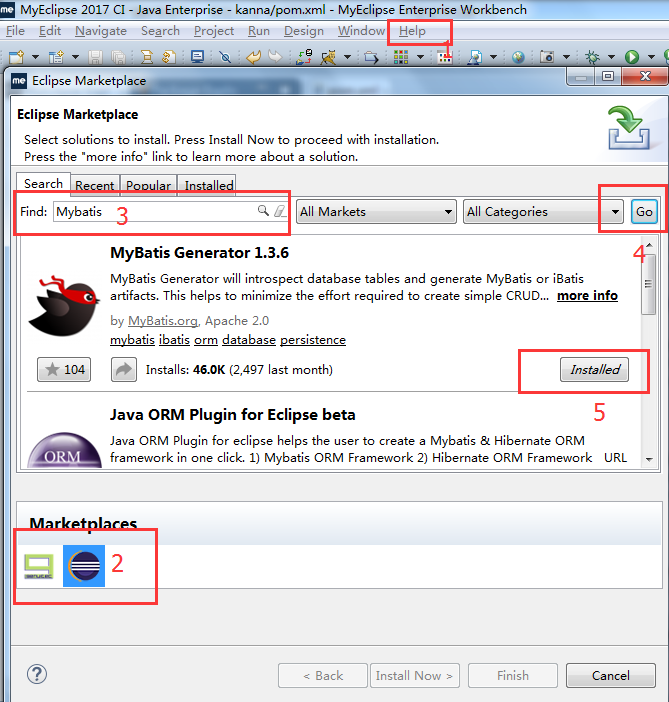
|  |
| --- |
| <**filter**>  <**filter-name**>myCorsFilter</**filter-name**>  <**filter-class**>org.springframework.web.filter.DelegatingFilterProxy</**filter-class**>  <**init-param**>  <**param-name**>targetBeanName</**param-name**>  <**param-value**>corsFilter</**param-value**>  </**init-param**> </**filter**> <**filter-mapping**>  <**filter-name**>myCorsFilter</**filter-name**>  <**url-pattern**>/\*</**url-pattern**> </**filter-mapping**> |

## Mybatis Generator配置

### MyEclipse

#### 安装

打开myeclipse--help---Install from catalog--选择eclipse应用市场---Find中填入 Mybatis (注意:区分大小写) 点go 搜索出Mybatis Generator 选择安装. 关闭myeclipse 重新打开.



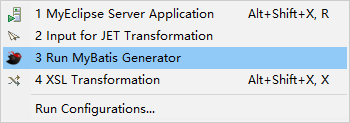
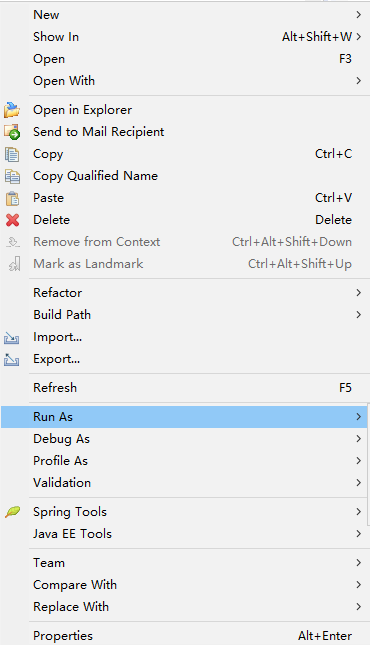
#### 使用

创建generatorConfig.xml文件

|  |
| --- |
| *<?***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**>  *<!-- 数据库驱动包位置 -->* <**classPathEntry location="D:\generator\mysql-connector-java-5.1.34.jar"** />   <**context id="DB2Tables" targetRuntime="MyBatis3"**>  <**commentGenerator**>  <**property name="suppressAllComments" value="true"** />  </**commentGenerator**>  *<!-- 数据库链接URL、用户名、密码 -->* <**jdbcConnection driverClass="com.mysql.jdbc.Driver" connectionURL="jdbc:mysql://localhost:3306/704a" userId="root" password="root"**>   </**jdbcConnection**>    <**javaTypeResolver**>  <**property name="forceBigDecimals" value="false"** />  </**javaTypeResolver**>    *<!-- 生成模型的包名和位置 -->* <**javaModelGenerator targetPackage="com.javasm.goods.entity" targetProject="D:\threeWorkspace\1111ssm\src"**>  <**property name="enableSubPackages" value="true"** />  <**property name="trimStrings" value="true"** />  </**javaModelGenerator**>    *<!-- 生成的映射文件包名和位置 -->* <**sqlMapGenerator targetPackage="com.javasm.goods.mapper" targetProject="D:\threeWorkspace\1111ssm\src"**>  <**property name="enableSubPackages" value="true"** />  </**sqlMapGenerator**>    *<!-- 生成DAO的包名和位置 -->* <**javaClientGenerator type="XMLMAPPER" targetPackage="com.javasm.goods.mapper" targetProject="D:\threeWorkspace\1111ssm\src"**>  <**property name="enableSubPackages" value="true"** />  </**javaClientGenerator**>    *<!-- 要生成那些表(更改tableName和domainObjectName就可以) -->* <**table tableName="goods" domainObjectName="Goods" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"** />  <**table tableName="goodstype" domainObjectName="GoodsType" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"** /></**context**> </**generatorConfiguration**> |

#### 运行

右键点击配置文件，选择运行，



### IDEA

#### 安装

因为IntelliJ中没有mybatis-generato对应的插件，所以需要在MAVEN中使用 mybatis-generator-maven-plugin插件来完成功能或者运行mybatis Generator代码方式来生成。

代码方式：

第一步：创建独立的工程：mybatisGenerator

第二步：工程添加依赖项：mybatis-generator-core.jar与mysql-connector-java.jar

第三步：拷贝**generator.xml配置文件到src目录**

第三步：从官网复制生成代码如下：

|  |
| --- |
| **public class** MyGenerator {  **public static void** main(String[] args) **throws** Exception {  List<String> warnings = **new** ArrayList<String>();  **boolean** overwrite = **true**;  String path = MyGenerator.**class**.getClassLoader().getResource(**"generator.xml"**).getPath();  File configFile = **new** File(path);  ConfigurationParser cp = **new** ConfigurationParser(warnings);  Configuration config = cp.parseConfiguration(configFile);  DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);  MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config, callback, warnings);  myBatisGenerator.generate(**null**);  } } |

## PageHelper分页插件

只需要引入两个jar

jsqlparser-0.9.5.jar

pagehelper-5.0.0.jar

### 配置dao.xml文件

|  |
| --- |
| <**bean id="sessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  <**property name="dataSource" ref="dataSource"**></**property**>  <**property name="mapperLocations" value="classpath:com/javasm/\*/mapper/\*.xml"**></**property**>  <**property name="typeAliasesPackage" value="com.javasm"**></**property**>  <**property name="configLocation" value="classpath:mybatis.xml"**></**property**>  <**property name="plugins"**>  <**array**>  <**bean class="com.github.pagehelper.PageInterceptor"**>  *<!--分页参数合理化-->* <**property name="properties"**>  <**value**>reasonable=true</**value**>  </**property**>  </**bean**>  </**array**>  </**property**> </**bean**> |

在原来配置文件上加入红色代码

### Mapper.xml

<select id="selectList" resultMap="BaseResultMap">

select \* from t\_student

</select>

只需要写基本的查询，不需要加入分页信息

### Handler

|  |
| --- |
| @GetMapping(**"users"**) **public** ResponseEntity selectUsers(Sysuser user, @RequestParam(defaultValue = **"1"**) Integer pageNum, @RequestParam(defaultValue = **"3"**) Integer pageSize){  *//在调用服务层的集合查询之前，进行分页开启* PageHelper.*startPage*(pageNum,pageSize);  *//ArrayList类型-->分页后-->Page类型* List<Sysuser> sysusers = **us**.selectUsers(user);  *//封装分页数据以及数据集* PageInfo info = **new** PageInfo(sysusers);  **return** ResponseEntity.*ok*(info); } |

### 属性介绍

|  |  |
| --- | --- |
| **属性名称** | **属性描述** |
| pageNum | 当前页 |
| pageSize | 每页的数量 |
| size | 当前页的数量 |
| startRow | 当前页面第一个元素在数据库中的行号(不常用) |
| endRow | 当前页面最后一个元素在数据库中的行号(不常用) |
| total | 总记录数 |
| pages | 总页数 |
| list | 结果集 |
| firstPage | 第一页 |
| prePage | 前一页 |
| isFirstPage | 是否为第一页 |
| isLastPage | 是否为最后一页 |
| hasPreviousPage | 是否有前一页 |
| hasNextPage | 是否有下一页 |
| navigatePages | 导航页码数 |
| navigatepageNums | 导航页码数组 |

任务：

使用第二阶段项目前端vue视图，实现单表的操作。