# SSM配置文件详解及各种配置归纳

## web.xml配置

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

<web-app version="3.0"

xmlns="http://java.sun.com/xml/ns/javaee"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd">

<display-name></display-name>

<!--

通过contextConfigLocation配置spring，contextConfigLocation参数定义了要装入的 Spring 配置文件。

如果想装入多个配置文件，可以在 <param-value>标记中用逗号作分隔符。

在web.xml里需配置ContextLoaderListener

-->

<context-param>

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

<param-value>classpath:spring.xml;classpath:spring-mybatis.xml</param-value>

</context-param>

<!-- 过滤通过用于处理项目中的乱码问题，该过滤器位于org.springframework.web.filter包中，指向类CharacterEncodingFilter -->

<filter>

<description>字符集过滤器</description>

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

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

<init-param>

<description>字符集编码</description>

<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>

<!--

自动装配ApplicationContext的配置信息。

因为它实现了ServletContextListener这个接口，在web.xml配置这个监听器，启动容器时，就会默认执行它实现的方法。

-->

<listener>

<description>spring监听器</description>

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

</listener>

<!--

servlet标准不允许在web容器内自行做线程管理

主要负责处理由　JavaBeans Introspector的使用而引起的缓冲泄露。清除Introspector的唯一方式是刷新整个缓冲

-->

<listener>

<listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>

</listener>

<!--

DispatcherServlet是前端控制器设计模式的实现，提供Spring Web MVC的集中访问点，而且负责职责的分派而且与Spring IoC容器无缝集成，从而可以获得Spring的所有好处。

DispatcherServlet主要用作职责调度工作，本身主要用于控制流程，主要职责如下：

1、文件上传解析，如果请求类型是multipart将通过MultipartResolver进行文件上传解析；

2、通过HandlerMapping，将请求映射到处理器（返回一个HandlerExecutionChain，它包括一个处理器、多个HandlerInterceptor拦截器）；

3、通过HandlerAdapter支持多种类型的处理器(HandlerExecutionChain中的处理器)；

4、通过ViewResolver解析逻辑视图名到具体视图实现；

5、本地化解析；

6、渲染具体的视图等；

7、如果执行过程中遇到异常将交给HandlerExceptionResolver来解析。

-->

<servlet>

<description>spring mvc servlet</description>

<servlet-name>springMvc</servlet-name>

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

<init-param>

<description>spring mvc 配置文件</description>

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

<param-value>classpath:spring-mvc.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>springMvc</servlet-name>

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

</servlet-mapping>

<!--

Session是由浏览器和服务器之间维护的。

Session超时理解为：浏览器和服务器之间创建了一个Session，由于客户端长时间（休眠时间）没有与服务器交互，

服务器将此Session销毁，客户端再一次与服务器交互时之前的Session就不存在了。

-->

<session-config>

<session-timeout>15</session-timeout>

</session-config>

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

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

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

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

<welcome-file-list>

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

</welcome-file-list>

<!-- 如果需要访问静态资源，如果在springmvc层中没有进行配置的话，可以在web.xml中进行配置一个servlet，该servlet的作用是为tomcat、jetty等容器提供，将静态资源映射从/改为/static/目录 -->

<servlet-maping>

<servlet-name>default</servlet-name>

</servlet-maping>

</web-app>

## spring-mvc.xml配置

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

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd

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

http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

<!-- 自动扫描controller包下的所有类，使其认为spring mvc的控制器 -->

<context:component-scan base-package="com.jsx.controller" />

1.使用sping自带的jackson包

<!--避免IE执行AJAX时，返回JSON出现下载文件 -->

<bean id="mappingJacksonHttpMessageConverter"

class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">

<property name="supportedMediaTypes">

<list>

<value>text/html;charset=UTF-8</value>

</list>

</property>

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter

<property name="messageConverters">

<list>

<ref bean="mappingJacksonHttpMessageConverter" /> <!-- JSON转换器

</list>

</property>

</bean

2.使用阿里巴巴的fastjson包

<!-- 避免IE执行AJAX时,返回JSON出现下载文件 -->

<!-- 支持JSON数据格式 -->

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="messageConverters">

<list>

<bean class="org.springframework.http.converter.ByteArrayHttpMessageConverter"/><!-- 解析导出文件byte流 -->

<ref bean="fastJsonHttpMessageConverter" />

<!--

<ref bean="mappingJacksonHttpMessageConverter" />

-->

</list>

</property>

</bean>

<!--<bean id="mappingJacksonHttpMessageConverter"

class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">

</bean>-->

<!-- 使用fastJson来支持JSON数据格式 -->

<bean id="fastJsonHttpMessageConverter" class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">

<property name="supportedMediaTypes">

<list>

<value>text/html;charset=UTF-8</value>

<value>application/json</value>

</list>

</property>

<property name="features">

<list>

<value>WriteMapNullValue</value>

<value>QuoteFieldNames</value>

</list>

</property>

</bean>

<注册各种消息转换类使用的时第三方jar包，进行设置>

<mvc:annotation-driven>

<mvc:message-converters>

<bean class="org.springframework.http.converter.ByteArrayHttpMessageConverter" />

<bean class="org.springframework.http.converter.FormHttpMessageConverter" />

<bean class="org.springframework.http.converter.xml.SourceHttpMessageConverter" />

<!--<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />-->

<bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter" />

</mvc:message-converters>

</mvc:annotation-driven>

<注册各种消息转换类使用的时使用自定义继承转换器的方式>

<!-- 对模型视图名称的解析，即在模型视图名称添加前后缀 controller方法返回值+.jsp-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" p:prefix="/" p:suffix=".jsp" />

<第一种开启静态资源不拦截功能>

<mvc:default-servlet-servlet-handler/>

<第二种开启静态资源不拦截功能>

<mvc:resources mapping=”/js/\*\*”>

</beans>

## spring-mybatis配置

<?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:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop" xsi:schemaLocation="

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

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

http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

">

<!-- 自动扫描(自动注入) -->

<context:component-scan base-package="com.jsx.service..\*" />

<!-- 第一种方法引入配置文件 -->

<bean id="propertyConfigurer"

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

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

</bean>

<!-- 第二种方法引入c3p0配置文件 -->

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

<!-- 配置数据源 使用c3p0的数据源>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="close">

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

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

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

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

需要注意的是出c3p0默认不支持事物自动提交，最好自己手动开启设置为true,具体参数可以自行百度

<!-- 初始化连接大小 -->

<property name="initialSize" value="${initialSize}"></property>

<!-- 连接池最大数量 -->

<property name="maxActive" value="${maxActive}"></property>

<!-- 连接池最大空闲 -->

<property name="maxIdle" value="${maxIdle}"></property>

<!-- 连接池最小空闲 -->

<property name="minIdle" value="${minIdle}"></property>

<!-- 获取连接最大等待时间 -->

<property name="maxWait" value="${maxWait}"></property>

</bean>

<!-- 配置数据源 使用阿里巴巴的数据源>

Druid是Java语言中最好的数据库连接池。Druid能够提供强大的监控和扩展功能。

-->

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

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

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

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

<!-- 初始化连接大小 -->

<property name="initialSize" value="0" />

<!-- 连接池最大使用连接数量 -->

<property name="maxActive" value="20" />

<!-- 连接池最大空闲 -->

<property name="maxIdle" value="20" />

<!-- 连接池最小空闲 -->

<property name="minIdle" value="0" />

<!-- 获取连接最大等待时间 -->

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

<!-- mysql 不支持 poolPreparedStatements-->

<!--

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

<property name="maxPoolPreparedStatementPerConnectionSize" value="33" />

-->

<!-- 验证数据库连接的查询语句，这个查询语句必须是至少返回一条数据的SELECT语句。 -->

<property name="validationQuery" value="${validationQuery}" />

<!-- 申请连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能。 -->

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

<!-- 归还连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能 -->

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

<!-- 建议配置为true，不影响性能，并且保证安全性。申请连接的时候检测，

如果空闲时间大于timeBetweenEvictionRunsMillis，执行validationQuery检测连接是否有效。 -->

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

<!-- 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="25200000" />

<!-- 打开removeAbandoned功能 -->

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

<!-- 1800秒，也就是30分钟 -->

<property name="removeAbandonedTimeout" value="1800" />

<!-- 关闭abanded连接时输出错误日志 -->

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

<!-- 开启Druid的监控统计功能 监控数据库

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

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

</bean>

<!-- myBatis文件

创建工厂 bean

SqlSessionFactoryBean 实现了 Spring 的 FactoryBean 接口

注意的是如果使用mybatis的时候讲classgai改成mybatis-plus的类，且加入包的时候只需要mybatis-plus一个第三方包就可以

-->

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

<!-- 可以是任意 的 DataSource,其配置应该和其它 Spring 数据库连接是一样的。-->

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

<!-- 自动扫描entity目录, 省掉Configuration.xml里的手工配置 --可以加入mybatisd的配置文件，配置文件可以为空>

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

</bean>

<!--

Mybatis MapperScannerConfigurer 自动扫描 将Mapper接口生成代理注入到Spring

MapperFactoryBean 创建的代理类实现了 UserMapper 接口,并且注入到应用程序中。

因为代理创建在运行时环境中(Runtime,译者注) ,那么指定的映射器必须是一个接口,而 不是一个具体的实现类。

缺点有很多的配置文件时 全部需要手动编写。

没有必要在 Spring 的 XML 配置文件中注册所有的映射器。相反,你可以使用一个 MapperScannerConfigurer ,

它 将 会 查 找 类 路 径 下 的 映 射 器 并 自 动 将 它 们 创 建 成 MapperFactoryBean。

MapperScannerConfigurer 支 持 过 滤 由 指 定 的 创 建 接 口 或 注 解 创 建 映 射 器 。 annotationClass 属性指定了要寻找的注解名称。

markerInterface 属性指定了要寻找的父 接口。如果两者都被指定了,加入到接口中的映射器会匹配两种标准。

默认情况下,这两个 属性都是 null,所以在基包中给定的所有接口可以作为映射器加载。

-->

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

<property name="basePackage" value="com.jsx.dao" />

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

</bean>

<!-- 配置事务管理器 -->

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

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

</bean>

<!-- 注解方式配置事物 -->

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

<!-- 拦截器方式配置事物

spring有很多事物管理，其中很多都是被淘汰的了，企业一直在用，最好配置方法如下，配置事务之后，用切面直接切入所有servic层

-->

<tx:advice id="transactionAdvice" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="add\*" propagation="REQUIRED" />

<tx:method name="append\*" propagation="REQUIRED" />

<tx:method name="insert\*" propagation="REQUIRED" />

<tx:method name="save\*" propagation="REQUIRED" />

<tx:method name="update\*" propagation="REQUIRED" />

<tx:method name="modify\*" propagation="REQUIRED" />

<tx:method name="edit\*" propagation="REQUIRED" />

<tx:method name="delete\*" propagation="REQUIRED" />

<tx:method name="remove\*" propagation="REQUIRED" />

<tx:method name="repair" propagation="REQUIRED" />

<tx:method name="delAndRepair" propagation="REQUIRED" />

<tx:method name="get\*" propagation="SUPPORTS" />

<tx:method name="find\*" propagation="SUPPORTS" />

<tx:method name="load\*" propagation="SUPPORTS" />

<tx:method name="search\*" propagation="SUPPORTS" />

<tx:method name="datagrid\*" propagation="SUPPORTS" />

<tx:method name="\*" propagation="SUPPORTS" />

</tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut id="transactionPointcut" expression="execution(\* com.jsx.service..\*Impl.\*(..))" />

<aop:advisor pointcut-ref="transactionPointcut" advice-ref="transactionAdvice" />

</aop:config>

<!-- 配置druid监控spring jdbc

至于Druid是一个用于大数据实时查询和分析的高容错、高性能开源分布式系统，旨在快速处理大规模的数据，并能够实现快速查询和分析。

尤其是当发生代码部署、机器故障以及其他产品系统遇到宕机等情况时，Druid仍能够保持100%正常运行。

创建Druid的最初意图主要是为了解决查询延迟问题，当时试图使用Hadoop来实现交互式查询分析，但是很难满足实时分析的需要。

而Druid提供了以交互方式访问数据的能力，并权衡了查询的灵活性和性能而采取了特殊的存储格式

Druid是为OLAP工作流的探索性分析而构建，它支持各种过滤、聚合和查询等类；

Druid的低延迟数据摄取架构允许事件在它们创建后毫秒内可被查询到；

Druid的数据在系统更新时依然可用，规模的扩大和缩小都不会造成数据丢失；

Druid已实现每天能够处理数十亿事件和TB级数据;

-->

<bean id="druid-stat-interceptor" class="com.alibaba.druid.support.spring.stat.DruidStatInterceptor"></bean>

<bean id="druid-stat-pointcut" class="org.springframework.aop.support.JdkRegexpMethodPointcut" scope="prototype">

<property name="patterns">

<list>

<value>com.jsx.service.\*</value>

</list>

</property>

</bean>

<aop:config>

<aop:advisor advice-ref="druid-stat-interceptor" pointcut-ref="druid-stat-pointcut" />

</aop:config>

<!-- 第一种利用spring自带功能实现DAO接口所在包名，Spring会自动查找其下的类 -->

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

<property name="basePackage" value="com.cn.hnust.dao" />

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

</bean>

<!-- 第二种利用mybatis自带功能实现DAO接口所在包名，Spring会自动查找其下的类 -->

<!--maybatis：scan会扫描dao包中所有的接口，当做spring的bean配置，之后就可以注入DI -->

<mybatis-spring:scan base-package="com.chinasoft.dao"/>

</beans>

## 数据源配置：

阿里巴巴的配置

#hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect

#driverClassName=com.mysql.jdbc.Driver

#来验证数据库连接的有效性

validationQuery=SELECT 1

jdbc\_url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull

jdbc\_username=root

jdbc\_password=root

！

C3p0的配置

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://10.221.10.111:8080/db\_zsl

username=demao

password=demao

#定义初始连接数

initialSize=0

#定义最大连接数

maxActive=20

#定义最大空闲

maxIdle=20

#定义最小空闲

minIdle=1

#定义最长等待时间

maxWait=60000