SSM搭建

SSM搭建	1
1. lib	5
2. web.xml	5
2.1. 配置启动 Spring IOC 容器的 Listener,启动spring容器 <context-< th=""><th></th></context-<>	
param> <param-name>contextConfigLocation</param-name> <param-< td=""><td></td></param-<>	
value>classpath:applicationContext.xml	
stener> stener-	
class>org.springframework.web.context.ContextLoaderListener	
	5
2.2. 表单提交controller获得中文参数后乱码解决方案 注意:</th <th></th>	
jsp页面编码设置为UTF-8	
form表单提交方式为必须为post,get方式下面spring编码过滤器不起效果>	
<filter> <filter-name>characterEncodingFilter</filter-name> <filter-< th=""><th></th></filter-<></filter>	
class>org.springframework.web.filter.CharacterEncodingFilter <init-< th=""><th></th></init-<>	
param> <param-name>encoding</param-name> <param-value>UTF-8<td></td></param-value>	
value> <init-param> <param-name>forceEncoding</param-name></init-param>	>
<param-value>true</param-value> <filter-mapping></filter-mapping>	
<filter-name>characterEncodingFilter</filter-name> <url-pattern>/*</url-pattern>	
	6
2.3. 可以把 POST 请求转为 DELETE 或 PUT 请求 <filter> <filter-< td=""><td></td></filter-<></filter>	
name>HiddenHttpMethodFilter <filter-< td=""><td></td></filter-<>	
class>org.springframework.web.filter.HiddenHttpMethodFilter <td>(></td>	(>
<pre><filter-mapping> <filter-name>HiddenHttpMethodFilter</filter-name> <url-< pre=""></url-<></filter-mapping></pre>	7
pattern>/*	/
2.4. 配置 DispatcherServlet <servlet> <servlet-< td=""><td></td></servlet-<></servlet>	
name>dispatcherServlet <servlet-< th=""><th></th></servlet-<>	
class>org.springframework.web.servlet.DispatcherServlet <init-< td=""><td></td></init-<>	
param> <param-name>contextConfigLocation</param-name> <param-value>classpath:springmvc.xml</param-value> <load-on-< td=""><td></td></load-on-<>	
startup>1	
name>dispatcherServlet <url-pattern>/</url-pattern> <td></td>	
mapping>	7
3. applicationContext.xml	
3.1. xml version="1.0" encoding="UTF-8"? <beans< td=""><td></td></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:mybatis-	-
spring="http://mybatis.org/schema/mybatis-spring"	
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://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-
spring-1.2.xsd http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd"> <!--
Spring管理所有的业务逻辑组件.不扫描@Controll控制器 --> <context:component-
scan base-package="com.zkx"> <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller" /> </context:component-
       <!-- 引入数据库的配置文件 --> <context:property-placeholder
location="classpath:dbconfig.properties" />
                                          <!-- 数据源 c3p0 --> <bean
id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
name="driverClass" value="${orcl.driver}"></property> <property name="user"
value="${orcl.username}"></property> <property name="password"
value="${orcl.password}"></property> </bean> <!-- Spring内置数据源配置 -->
<!-- <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
驱动类
           cproperty name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
                                                  url地址
                                                             property
name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"></property>
<!-- spring事务管理 --
name="password" value="123"></property>
                                         </bean> -->
> <bean id="dataSourceTransactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
开启基于注解的事务 --> <tx:annotation-driven transaction-
manager="dataSourceTransactionManager"/>
整合mybatis,spring管理mapper的实现,spring用来管理事务 --> <!--
创建出SqlSessionFactory对象 --> <bean id="sqlSessionFactoryBean"
ref="dataSource"></property> <!-- configLocation指定全局配置文件的位置 -->
configLocation" value="classpath:mybatisconfig.xml">
<!--mapperLocations: 指定mapper文件的位置-->                                                                                                                                                                                                                                                                                                                                               <pr
name="mapperLocations" value="classpath:com/zkx/dao/*.xml"></property>
</bean>
           <!--
扫描所有的mapper接口的实现,让这些mapper能够自动注入; base-
value="com.softix.dao"></property> 新版: <mybatis-spring:scan base-
package="com.softjx.dao"/> -->
                                   <bean
```

	class="org.mybatis.spring.mapper.MapperScannerConfigurer"> < property	
	name="basePackage" value="com.zkx.dao">	8
4	l. springmvc.xml	11
	4.1. xml version="1.0" encoding="UTF-8"? <beans< td=""><td></td></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: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/context http://www.springframework.org/schema/context/spring-context-4.0.xsd	
	http://www.springframework.org/schema/mvc	
	http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"> </td <td></td>	
	配置自定扫描的包> <context:component-scan base-package="com.zkx" td="" use-<=""><td></td></context:component-scan>	
	default-filters="false"> <context:include-filter <="" td="" type="annotation"><td></td></context:include-filter>	
	expression="org.springframework.stereotype.Controller"/> <td>t-</td>	t-
	scan> 配置视图解析器: 如何把 handler 方法返回值解析为实际的物理视</td <td></td>	
	> <bean< td=""><td>-4</td></bean<>	-4
	class="org.springframework.web.servlet.view.InternalResourceViewResolver">	
	<pre><pre><pre><pre><pre><pre><pre>property name="prefix" value=""></pre></pre><pre><pre>property name="suffix"</pre></pre></pre></pre></pre></pre></pre>	
	value=".jsp">	t-
	handler/> 默认配置方案。</td <td></td>	
	并提供了:数据绑定支持,@NumberFormatannotation支持,	
	@DateTimeFormat支持,@Valid支持,读写XML的支持(JAXB),	
	读写JSON的支持(Jackson)。	
	后面,我们处理响应ajax请求时,就使用到了对json的支持。>	
		11
5	i. mybatis.xml	
_	5.1. xml version="1.0" encoding="UTF-8"? configuration PUBLIC</td <td></td>	
	//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dt	
	<pre><configuration> <settings> <!--</pre--></settings></configuration></pre>	
	显式的指定每个我们需要更改的配置的值,即使他是默认的。防止版本更新·	带来
	的问题> <setting name="lazyLoadingEnabled" value="true"></setting> <setting< td=""><td></td></setting<>	
	name="aggressiveLazyLoading" value="false"/> <plugins <plugins> <</plugins>	in
	interceptor="com.github.pagehelper.PageInterceptor"> >	
		12
ϵ	5. 步骤	13
	6.1. 1.建立POJO实体对象	13
	6.2. 2.DAO层	13
	6.2.1. UserMapper.java(接口)	13
	6.2.2. UserMapper.xml	

6.3. 3.Service层	13
6.3.1. UserService.java(接口)(方法和DAO层的接口保持一致)	13
6.3.2. UserServiceImpl.java	
6.4. 4.Test测试	14
6.4.1. private static ApplicationContext context = null; private Use userService = null; @Before public void init() throws Exception { cc ClassPathXmlApplicationContext("applicationContext.xml"); userSe (UserService) context.getBean("userService"); } @Test public void }	erService ontext = new rvice =
6.5. 5.Controller层	14
6.5.1. 类注解,表明它是一个控制器,以便让springMVC扫描到: @	Controller
@RequestMapping("user") 自动注入一个Service层的对象: @Auto	wired private
UserService userService; 方法注解,提供uri的路径: @RequestMapp返回值:String 代表返回的视图路径 形参:可以传回到视图层	• • • • • • • • • • • • • • • • • • • •
6.6. 6.视图层	15
6.6.1. JSTL会自动识别对象里面的属性,EL表达式可以取出四大均	或中的对象
<c:foreach items="\${ List}" var="user"> \${user.id} <td>Each>15</td></c:foreach>	Each>15

1. lib

1.1.

- c3p0-0.9.2.1.jar
- com.springsource.net.sf.cglib-2.2.0.jar
- com.springsource.org.aopalliance-1.0.0.
- com.springsource.org.aspectj.weaver-1.
- 🖹 commons-logging-1.1.3.jar
- 剷 jsqlparser-0.9.5.jar
- 🗐 jstl-1.2.jar
- 🗐 jstl-standard.jar
- 🖹 mchange-commons-java-0.2.6.2.jar
- 剷 mybatis-3.4.5.jar
- mybatis-spring-1.3.0.jar
- mysql-connector-java-5.1.6-bin.jar
- ojdbc7.jar
- pagehelper-5.0.0-rc.jar
- 🗟 servlet-api-2.5.jar
- spring-aop-4.0.0.RELEASE.jar
- 違 spring-aspects-4.0.0.RELEASE.jar
- spring-beans-4.0.0.RELEASE.jar
- spring-context-4.0.0.RELEASE.jar
- 🗐 spring-core-4.0.0.RELEASE.jar
- spring-expression-4.0.0.RELEASE.jar
- spring-jdbc-4.0.0.RELEASE.jar
- 🗐 spring-orm-4.0.0.RELEASE.jar
- spring-tx-4.0.0.RELEASE.jar
- 🖹 spring-web-4.0.0.RELEASE.jar
- spring-webmvc-4.0.0.RELEASE.jar

2. web.xml

2.1. <!-- 配置启动 Spring IOC 容器的 Listener,启动spring容器 --> <context-param>

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

```
<param-value>classpath:applicationContext.xml</param-value>
 </context-param>
 stener>
   stener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
 </listener>
2.2.<!-- 表单提交controller获得中文参数后乱码解决方案 注意:
jsp页面编码设置为UTF-8
form表单提交方式为必须为post,get方式下面spring编码过滤器不起效果 -->
  <filter>
   <filter-name>characterEncodingFilter</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>
   <init-param>
     <param-name>forceEncoding</param-name>
     <param-value>true</param-value>
   </init-param>
  </filter>
 <filter-mapping>
   <filter-name>characterEncodingFilter</filter-name>
   <url-pattern>/*</url-pattern>
 </filter-mapping>
```

```
2.3.<!-- 可以把 POST 请求转为 DELETE 或 PUT 请求 -->
  <filter>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-</pre>
class>
 </filter>
  <filter-mapping>
    <filter-name>HiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
2.4. <!-- 配置 DispatcherServlet -->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-</p>
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>
```

3. applicationContext.xml

```
3.1. <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"</p>
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
 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://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-
spring-1.2.xsd
   http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">
<!-- Spring管理所有的业务逻辑组件,不扫描@Controll控制器 -->
 <context:component-scan base-package="com.zkx">
   <context:exclude-filter type="annotation"</pre>
     expression="org.springframework.stereotype.Controller" />
 </context:component-scan>
 <!-- 引入数据库的配置文件 -->
  <context:property-placeholder location="classpath:dbconfig.properties" />
  <!-- 数据源 c3p0 -->
```

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
   cproperty name="idbcUrl" value="${orcl.url}">
   cproperty name="driverClass" value="${orcl.driver}"></property>
   coperty name="user" value="${orcl.username}">
   cproperty name="password" value="${orcl.password}">
 </bean>
  <!-- Spring内置数据源配置 -->
 <!-- <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    驱动类
    property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"></property>
    url地址
    cproperty name="url"
value="jdbc:oracle:thin:@localhost:1521:orcl"></property>
   cyroperty name="username" value="C##RZ">
    property name="password" value="123"></property>
  </bean> -->
 <!-- spring事务管理 -->
 <bean id="dataSourceTransactionManager"</pre>
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
   cproperty name="dataSource" ref="dataSource" >
 </bean>
 <!-- 开启基于注解的事务 -->
 <tx:annotation-driven transaction-manager="dataSourceTransactionManager"/>
```

```
<!--
 整合mybatis,spring管理mapper的实现,spring用来管理事务
 -->
 <!--创建出SqlSessionFactory对象 -->
 <bean id="sqlSessionFactoryBean"</pre>
class="org.mybatis.spring.SqlSessionFactoryBean">
   cproperty name="dataSource" ref="dataSource" >/property>
   <!-- configLocation指定全局配置文件的位置 -->
   configLocation"
value="classpath:mybatisconfig.xml"></property>
   <!--mapperLocations: 指定mapper文件的位置-->
   property name="mapperLocations"
value="classpath:com/zkx/dao/*.xml"></property>
 </bean>
 <!-- 扫描所有的mapper接口的实现,让这些mapper能够自动注入; base-
package: 指定mapper接口的包名
 老版: 
 新版: <mybatis-spring:scan base-package="com.softjx.dao"/>
 -->
 <br/><bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
   cproperty name="basePackage" value="com.zkx.dao">
 </bean>
```

```
</beans>
```

4. springmvc.xml

```
4.1. <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"</pre>
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 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.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
   http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
 <!-- 配置自定扫描的包 -->
 <context:component-scan base-package="com.zkx" use-default-filters="false">
   <context:include-filter type="annotation"</pre>
     expression="org.springframework.stereotype.Controller"/>
 </context:component-scan>
 <!-- 配置视图解析器: 如何把 handler 方法返回值解析为实际的物理视图 -->
 <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
   coperty name="prefix" value="">
   content
 </bean>
```

<!-- 处理静态资源 -->

<mvc:default-servlet-handler/>

<!-- 默认配置方案。

并提供了:数据绑定支持,@NumberFormatannotation支持,

@DateTimeFormat支持,@Valid支持,读写XML的支持(JAXB),

读写JSON的支持(Jackson)。

后面,我们处理响应ajax请求时,就使用到了对json的支持。

-->

<mvc:annotation-driven></mvc:annotation-driven>

</beans>

5. mybatis.xml

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

<!DOCTYPE configuration</pre>

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<settings>

<!--

显式的指定每个我们需要更改的配置的值,即使他是默认的。防止版本更新带来

```
的问题 -->
    <setting name="lazyLoadingEnabled" value="true"/>
    <setting name="aggressiveLazyLoading" value="false"/>
   </settings>
   <!-- <plugins>
    <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>
   </plugins> -->
 </configuration>
6. 步骤
 6.1.1.建立POJO实体对象
 6.2.2.DAO层
   6.2.1. UserMapper.java(接口)
   6.2.2. UserMapper.xml
    namespace 中的要和接口名保持一致,sql 的ID 要和接口中的方法一致
    <mapper namespace="com.zkx.dao.UserMapper" >
    </mapper>
 6.3.3.Service层
   6.3.1. UserService.java(接口)(方法和DAO层的接口保持一致)
   6.3.2. UserServiceImpl.java
    类前面需要注解@Service("userService")
      @Transactional
    该类需要私有的成员变量,通过spring自动注入的方式获取DAO层的对象,改对
    象可以直接使用DAO层接口中的方法
```

```
@Autowired
     private UserMapper userMapper;
6.4.4.Test测试
 6.4.1. private static ApplicationContext context = null;
   private UserService userService = null;
 @Before
   public void init() throws Exception {
     context = new ClassPathXmlApplicationContext("applicationContext.xml");
     userService = (UserService) context.getBean("userService");
   }
 @Test
   public void TestUser() {
   }
6.5.5.Controller层
 6.5.1. 类注解,表明它是一个控制器,以便让springMVC扫描到:
   @Controller
   @RequestMapping("user")
 自动注入一个Service层的对象:
   @Autowired
   private UserService userService;
 方法注解,提供uri的路径:
   @RequestMapping("/selAll")
 返回值:String 代表返回的视图路径
 形参:可以传回到视图层
```

6.6.6.视图层

6.6.1. JSTL会自动识别对象里面的属性,EL表达式可以取出四大域中的对象
<c:forEach var="user" items="\${ List}">
 \${user.id}
</c:forEach>