[深入浅出Mybatis-与Spring集成](http://blog.csdn.net/hupanfeng/article/details/21454847)

标签： [MyBatis](http://www.csdn.net/tag/MyBatis)[spring](http://www.csdn.net/tag/spring)[集成](http://www.csdn.net/tag/%e9%9b%86%e6%88%90)

2014-03-18 13:48 39052人阅读 [评论](http://blog.csdn.net/hupanfeng/article/details/21454847#comments)(12) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/hupanfeng/article/details/21454847#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Mybatis（8） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/hupanfeng/article/details/21454847)

单独使用mybatis是有很多限制的（比如无法实现跨越多个session的事务），而且很多业务系统本来就是使用spring来管理的事务，因此mybatis最好与spring集成起来使用。

**前置要求**

**版本要求**

|  |  |  |  |
| --- | --- | --- | --- |
| 项目 | 版本 | 下载地址 | 说明 |
| mybatis | 3.0及以上 | https://github.com/mybatis/mybatis-3/releases |  |
| spring | 3.0及以上 | http://projects.spring.io/spring-framework/ |  |
| mybatis-spring | 1.0及以上 | https://github.com/mybatis/spring/releases |  |

**spring事务配置**

**[html]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. <!-- 自动扫描业务包 -->
2. **<context:component-scan** base-package="com.xxx.service" **/>**
4. <!-- 数据源 -->
5. **<jee:jndi-lookup** id="jndiDataSource" jndi-name="java:comp/env/jdbc/datasource" **/>**
7. <!-- 配置事务 -->
8. **<bean** id="txManager"
9. class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
10. **<property** name="dataSource" ref="jndiDataSource" **/>**
11. **</bean>**
12. <!-- 配置基于注解的事物aop -->
13. **<tx:annotation-driven** transaction-manager="txManager" proxy-target-class="true"**/>**

**单个集成**

**[html]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. <!-- 集成mybatis -->
2. **<bean** id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>**
3. **<property** name="dataSource" ref="jndiDataSource" **/>**
4. **<property** name="configLocation" value="classpath:/mybatis/mybatis-config.xml" **/>**
5. <!-- 自动配置别名 -->
6. **<property** name="typeAliasesPackage" value="com.xxx.dto" **/>**
7. **</bean>**
9. <!--创建dao bean（只需提供接口不需提供实现类 ）-->
10. **<bean** id="userDao" class="org.mybatis.spring.mapper.MapperFactoryBean"**>**
11. **<property** name="mapperInterface" value="com.xxx.dao.UserDao" **/>**
12. **<property** name="sqlSessionFactory" ref="sqlSessionFactory" **/>**
13. **</bean>**

我们不但要明白如何使用，更要明白为什么要这么使用。

*SqlSessionFactoryBean*是一个工厂bean，它的作用就是解析配置（数据源、别名等）。

*MapperFactoryBean*是一个工厂bean，在spring容器里，工厂bean是有特殊用途的，当spring将工厂bean注入到其他bean里时，它不是注入工厂bean本身而是调用bean的getObject方法。我们接下来就看看这个getObjec方法干了些什么：

**[java]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. **public** T getObject() **throws** Exception {
2. **return** getSqlSession().getMapper(**this**.mapperInterface);
3. }

看到这里大家应该就很明白了，这个方法和我们之前单独使用Mybatis的方式是一样的，都是先获取一个Sqlsession对象，然后再从Sqlsession里获取Mapper对象（再次强调Mapper是一个代理对象，它代理的是mapperInterface接口，而这个接口是用户提供的dao接口）。自然，最终注入到业务层就是这个Mapper对象。

实际的项目一般来说不止一个Dao，如果你有多个Dao那就按照上面的配置依次配置即可。

**如何使用批量更新**

前一节讲了如何注入一个mapper对象到业务层， mapper的行为依赖于配置，mybatis默认使用单个更新（即ExecutorType默认为SIMPLE而不是BATCH），当然我们可以通过修改mybatis配置文件来修改默认行为，但如果我们只想让某个或某几个mapper使用批量更新就不得行了。这个时候我们就需要使用模板技术：

**[html]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. <!--通过模板定制mybatis的行为 -->
2. lt;bean id="sqlSessionTemplateSimple" class="org.mybatis.spring.SqlSessionTemplate"**>**
3. **<constructor-arg** index="0" ref="sqlSessionFactory" **/>**
4. <!--更新采用单个模式 -->
5. **<constructor-arg** index="1" value="SIMPLE"**/>**
6. **</bean>**
8. <!--通过模板定制mybatis的行为 -->
9. lt;bean id="sqlSessionTemplateBatch" class="org.mybatis.spring.SqlSessionTemplate"**>**
10. **<constructor-arg** index="0" ref="sqlSessionFactory" **/>**
11. <!--更新采用批量模式 -->
12. **<constructor-arg** index="1" value="BATCH"**/>**
13. **</bean>**

这里笔者定义了两个模板对象，一个使用单个更新，一个使用批量更新。有了模板之后我们就可以改变mapper的行为方式了：

**[html]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. **<bean** id="userDao" class="org.mybatis.spring.mapper.MapperFactoryBean"**>**
2. **<property** name="mapperInterface" value="com.xxx.dao.UserDao" **/>**
3. **<property** name="sqlSessionTemplate" ref=" sqlSessionTemplateBatch " **/>**
4. **</bean>**

跟上一节的mapper配置不同的是，这里不需要配置*sqlSessionFactory*属性，只需要配置*sqlSessionTemplate*（*sqlSessionFactory*属性在模板里已经配置好了）。

**通过自动扫描简化mapper的配置**

前面的章节可以看到，我们的dao需要一个一个的配置在配置文件中，如果有很多个dao的话配置文件就会非常大，这样管理起来就会比较痛苦。幸好mybatis团队也意识到了这点，他们利用spring提供的自动扫描功能封装了一个自动扫描dao的工具类，这样我们就可以使用这个功能简化配置：

**[html]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. <!-- 采用自动扫描方式创建mapper bean(单个更新模式) -->
2. **<bean** class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>**
3. **<property** name="basePackage" value="com.xxx.dao" **/>**
4. **<property** name="sqlSessionTemplateBeanName" value="sqlSessionTemplateSimple" **/>**
5. **<property** name="markerInterface" value="com.xxx.dao.SimpleDao" **/>**
6. **</bean>**
8. <!-- 采用自动扫描方式创建mapper bean(批量更新模式) -->
9. **<bean** class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>**
10. **<property** name="basePackage" value="com.xxx.dao" **/>**
11. **<property** name="sqlSessionTemplateBeanName" value="sqlSessionTemplateBatch" **/>**
12. **<property** name="markerInterface" value="com.xxx.dao.BatchDao" **/>**
13. **</bean>**

MapperScannerConfigurer本身涉及的spring的技术我就不多讲了，感兴趣且对spring原理比较了解的可以去看下它的源码。我们重点看一下它的三个属性：

*basePackage：扫描器开始扫描的基础包名，支持嵌套扫描；*

*sqlSessionTemplateBeanName：前文提到的模板bean的名称；*

*markerInterface：基于接口的过滤器，实现了该接口的dao才会被扫描器扫描，与basePackage是与的作用。*

除了使用接口过滤外，还可使用注解过滤：

**[html]** [view plain](http://blog.csdn.net/hupanfeng/article/details/21454847) [copy](http://blog.csdn.net/hupanfeng/article/details/21454847)

[在CODE上查看代码片](https://code.csdn.net/snippets/242285)

1. <!-- 采用自动扫描方式创建mapper bean(批量更新模式) -->
2. **<bean** class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>**
3. **<property** name="basePackage" value="com.xxx.dao" **/>**
4. **<property** name="sqlSessionTemplateBeanName" value="sqlSessionTemplateBatch" **/>**
5. **<property** name="annotationClass" value="com.xxx.dao.BatchAnnotation" **/>**
6. **</bean>**

*annotationClass*：*配置了该注解的dao才会被扫描器扫描，与basePackage是与的作用。*

*需要注意的是，两个过滤条件只能配一个。*

# [Spring MVC整合Mybatis实例](http://blog.csdn.net/geloin/article/details/7536968)

标签： [spring](http://www.csdn.net/tag/spring)[mvc](http://www.csdn.net/tag/mvc)[integer](http://www.csdn.net/tag/integer)[date](http://www.csdn.net/tag/date)[string](http://www.csdn.net/tag/string)[url](http://www.csdn.net/tag/url)

2012-05-05 10:52 83482人阅读 [评论](http://blog.csdn.net/geloin/article/details/7536968/#comments)(68) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/geloin/article/details/7536968/#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

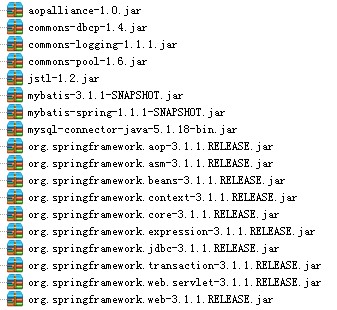
Framework（25） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

示例下载地址：<http://download.csdn.net/detail/geloin/4506640>

        本文基于[Spring 注解，让Spring跑起来](http://blog.csdn.net/geloin/article/details/7536856)。本文使用Mysql数据库。

        (1) 导入相关包，包结构如下图所示：



        (2) 修改src/applicationContext.xml文件，结果如下所示：

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:tx="http://www.springframework.org/schema/tx"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
9. http://www.springframework.org/schema/tx
10. http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
11. http://www.springframework.org/schema/context
12. http://www.springframework.org/schema/context/spring-context-3.0.xsd">
14. <!-- 引入jdbc配置文件 -->
15. <context:property-placeholder location="classpath:jdbc.properties" />
17. <!--创建jdbc数据源 -->
18. <bean id="dataSource" **class**="org.apache.commons.dbcp.BasicDataSource"
19. destroy-method="close">
20. <property name="driverClassName" value="${driver}" />
21. <property name="url" value="${url}" />
22. <property name="username" value="${username}" />
23. <property name="password" value="${password}" />
24. </bean>
26. <!-- (事务管理)transaction manager, use JtaTransactionManager **for** global tx -->
27. <bean id="transactionManager"
28. **class**="org.springframework.jdbc.datasource.DataSourceTransactionManager">
29. <property name="dataSource" ref="dataSource" />
30. </bean>
32. <!-- 创建SqlSessionFactory，同时指定数据源 -->
33. <bean id="sqlSessionFactory" **class**="org.mybatis.spring.SqlSessionFactoryBean">
34. <property name="dataSource" ref="dataSource" />
35. </bean>
37. <!-- 可通过注解控制事务 -->
38. <tx:annotation-driven />
40. <!-- Mapper接口所在包名，Spring会自动查找其下的Mapper -->
41. <bean **class**="org.mybatis.spring.mapper.MapperScannerConfigurer">
42. <property name="basePackage" value="com.geloin.spring.mapper" />
43. </bean>
45. </beans>

        (3) 在src下添加jdbc.properties

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. driver=com.mysql.jdbc.Driver
2. url=jdbc:mysql://localhost:3306/ruisystem
3. username=root
4. password=root

        (4) 在com.geloin.spring.entity包下添加实体类，实体类对应于数据表，其属性与数据表相同或多于数据表。

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. /\*\*
2. \*
3. \* @author geloin
4. \* @date 2012-5-5 上午10:24:43
5. \*/
6. **package** com.geloin.spring.entity;
8. /\*\*
9. \*
10. \* @author geloin
11. \* @date 2012-5-5 上午10:24:43
12. \*/
13. **public** **class** Menu {
14. /\*\*
15. \* 惟一标识
16. \*/
17. **private** Integer id;
18. /\*\*
19. \* 父ID
20. \*/
21. **private** Integer parentId;
22. /\*\*
23. \* 名称
24. \*/
25. **private** String name;
26. /\*\*
27. \* 对应的地址
28. \*/
29. **private** String url;
30. /\*\*
31. \* 是否显示在左侧
32. \*/
33. **private** Integer isShowLeft;
35. /\*\*
36. \*
37. \* @author geloin
38. \* @date 2012-5-5 上午10:26:19
39. \* @return the id
40. \*/
41. **public** Integer getId() {
42. **return** id;
43. }
45. /\*\*
46. \*
47. \* @author geloin
48. \* @date 2012-5-5 上午10:26:19
49. \* @param id
50. \*            the id to set
51. \*/
52. **public** **void** setId(Integer id) {
53. **this**.id = id;
54. }
56. /\*\*
57. \*
58. \* @author geloin
59. \* @date 2012-5-5 上午10:26:19
60. \* @return the parentId
61. \*/
62. **public** Integer getParentId() {
63. **return** parentId;
64. }
66. /\*\*
67. \*
68. \* @author geloin
69. \* @date 2012-5-5 上午10:26:19
70. \* @param parentId
71. \*            the parentId to set
72. \*/
73. **public** **void** setParentId(Integer parentId) {
74. **this**.parentId = parentId;
75. }
77. /\*\*
78. \*
79. \* @author geloin
80. \* @date 2012-5-5 上午10:26:19
81. \* @return the name
82. \*/
83. **public** String getName() {
84. **return** name;
85. }
87. /\*\*
88. \*
89. \* @author geloin
90. \* @date 2012-5-5 上午10:26:19
91. \* @param name
92. \*            the name to set
93. \*/
94. **public** **void** setName(String name) {
95. **this**.name = name;
96. }
98. /\*\*
99. \*
100. \* @author geloin
101. \* @date 2012-5-5 上午10:26:19
102. \* @return the url
103. \*/
104. **public** String getUrl() {
105. **return** url;
106. }
108. /\*\*
109. \*
110. \* @author geloin
111. \* @date 2012-5-5 上午10:26:19
112. \* @param url
113. \*            the url to set
114. \*/
115. **public** **void** setUrl(String url) {
116. **this**.url = url;
117. }
119. /\*\*
120. \*
121. \* @author geloin
122. \* @date 2012-5-5 上午10:26:19
123. \* @return the isShowLeft
124. \*/
125. **public** Integer getIsShowLeft() {
126. **return** isShowLeft;
127. }
129. /\*\*
130. \*
131. \* @author geloin
132. \* @date 2012-5-5 上午10:26:19
133. \* @param isShowLeft
134. \*            the isShowLeft to set
135. \*/
136. **public** **void** setIsShowLeft(Integer isShowLeft) {
137. **this**.isShowLeft = isShowLeft;
138. }
140. }

        (5) 在com.geloin.spring.mapper下添加实体类与数据表的映射关系（com.geloin.spring.mapper与applicationContext.xml中的配置一致）。

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. /\*\*
2. \*
3. \* @author geloin
4. \* @date 2012-5-5 上午10:26:34
5. \*/
6. **package** com.geloin.spring.mapper;
8. **import** java.util.List;
10. **import** org.apache.ibatis.annotations.Param;
11. **import** org.apache.ibatis.annotations.Result;
12. **import** org.apache.ibatis.annotations.Results;
13. **import** org.apache.ibatis.annotations.Select;
14. **import** org.springframework.stereotype.Repository;
16. **import** com.geloin.spring.entity.Menu;
18. /\*\*
19. \*
20. \* @author geloin
21. \* @date 2012-5-5 上午10:26:34
22. \*/
23. @Repository(value = "menuMapper")
24. **public** **interface** MenuMapper {
26. @Select(value = "${sql}")
27. @Results(value = { @Result(id = **true**, property = "id", column = "id"),
28. @Result(property = "parentId", column = "c\_parent\_id"),
29. @Result(property = "url", column = "c\_url"),
30. @Result(property = "isShowLeft", column = "c\_is\_show\_left"),
31. @Result(property = "name", column = "c\_name") })
32. List<Menu> operateReturnBeans(@Param(value = "sql") String sql);
33. }

        其中，@Repository表示这是一个被Spring管理的资源，资源名称为menuMapper；@Select表示operateReturnBeans方法为一个select方法；@Results表示返回结果，@Result将返回结果中的字段名与实体类关联；@Param表示String sql这个变量是用于Mybatis的一个变量，其名称为sql（value值），该变量在@Select中调用（通过${sql}调用）。

        (6) 在com.geloin.spring.service中添加MenuService接口

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. /\*\*
2. \*
3. \* @author geloin
4. \* @date 2012-5-5 上午10:28:42
5. \*/
6. **package** com.geloin.spring.service;
8. **import** java.util.List;
10. **import** com.geloin.spring.entity.Menu;
12. /\*\*
13. \*
14. \* @author geloin
15. \* @date 2012-5-5 上午10:28:42
16. \*/
17. **public** **interface** MenuService {
18. /\*\*
19. \* 查询所有
20. \*
21. \* @author geloin
22. \* @date 2012-5-5 上午10:28:55
23. \* @return
24. \*/
25. List<Menu> find();
26. }

        (7) 在com.geloin.spring.service.impl中添加MenuServiceImpl作为MenuService接口的实现

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. /\*\*
2. \*
3. \* @author geloin
4. \* @date 2012-5-5 上午10:29:22
5. \*/
6. **package** com.geloin.spring.service.impl;
8. **import** java.util.List;
10. **import** javax.annotation.Resource;
12. **import** org.springframework.stereotype.Repository;
13. **import** org.springframework.transaction.annotation.Transactional;
15. **import** com.geloin.spring.entity.Menu;
16. **import** com.geloin.spring.mapper.MenuMapper;
17. **import** com.geloin.spring.service.MenuService;
19. /\*\*
20. \*
21. \* @author geloin
22. \* @date 2012-5-5 上午10:29:22
23. \*/
24. @Repository(value = "menuService")
25. @Transactional
26. **public** **class** MenuServiceImpl **implements** MenuService {
28. @Resource(name = "menuMapper")
29. **private** MenuMapper menuMapper;
31. /\*
32. \* (non-Javadoc)
33. \*
34. \* @see com.geloin.spring.service.MenuService#find()
35. \*/
36. @Override
37. **public** List<Menu> find() {
38. String sql = "select \* from tb\_system\_menu";
39. **return** **this**.menuMapper.operateReturnBeans(sql);
40. }
42. }

        其中，@Transactional表示该类被Spring作为管理事务的类，@Resource引入一个Spring定义的资源，资源名为menuMapper（name值），即为第七步定义的映射类。

        (8) 修改控制器LoginController

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. /\*\*
2. \*
3. \* @author geloin
4. \* @date 2012-5-5 上午9:31:52
5. \*/
6. **package** com.geloin.spring.controller;
8. **import** java.util.HashMap;
9. **import** java.util.List;
10. **import** java.util.Map;
12. **import** javax.annotation.Resource;
13. **import** javax.servlet.http.HttpServletResponse;
15. **import** org.springframework.stereotype.Controller;
16. **import** org.springframework.web.bind.annotation.RequestMapping;
17. **import** org.springframework.web.servlet.ModelAndView;
19. **import** com.geloin.spring.entity.Menu;
20. **import** com.geloin.spring.service.MenuService;
22. /\*\*
23. \*
24. \* @author geloin
25. \* @date 2012-5-5 上午9:31:52
26. \*/
27. @Controller
28. @RequestMapping(value = "background")
29. **public** **class** LoginController {
31. @Resource(name = "menuService")
32. **private** MenuService menuService;
34. /\*\*
35. \*
36. \*
37. \* @author geloin
38. \* @date 2012-5-5 上午9:33:22
39. \* @return
40. \*/
41. @RequestMapping(value = "to\_login")
42. **public** ModelAndView toLogin(HttpServletResponse response) **throws** Exception {
44. Map<String, Object> map = **new** HashMap<String, Object>();
46. List<Menu> result = **this**.menuService.find();
48. map.put("result", result);
50. **return** **new** ModelAndView("background/menu", map);
51. }
52. }

        通过map将从数据库中获取的值传递到jsp页面，"background/menu"值经context-dispatcher.xml转化后，变为/WEB-INF/pages/background/menu.jsp，即，方法toLogin的含义为：从数据库中获取菜单信息，然后将之存储到map中，通过map把菜单列表传递到/WEB-INF/pages/background/menu.jsp页面用于显示。

        (9) 编写/WEB-INF/pages/background/menu.jsp页面

**[java]** [view plain](http://blog.csdn.net/geloin/article/details/7536968/) [copy](http://blog.csdn.net/geloin/article/details/7536968/)

1. <%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%>
3. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
8. <title>Insert title here</title>
9. </head>
10. <body>
11. <c:forEach items="${result }" var="item">
12. ${item.id }--${item.name }--${item.parentId }--${item.url }--${item.isShowLeft }<br />
13. </c:forEach>
14. </body>
15. </html>