# 单个工程中Spring+Mybatis连接多个数据库的配置（个人分享版本）

上一篇博客说到同一个问题，经过和朋友的研究已经参考网上的资料，现在给出一份更简洁的配置。

情景：现在单个工程中需要连接两个库，这两个库在同一个mysql中，两个库都需要进行读写。

解决：

## 第一步：将spring和mybatis整合，这个过程就不具体演示了，在这个过程中创建了直接使用的五个配置文件、两个类、一个标识。

五个配置文件：

jdbc.propertis（数据库连接信息）

applicationContext.xml（spring的核心配置文件）

applicationContext-transaction.xml（spring事务管理配置文件）

mybatis-config.xml（mybatis的核心配置文件）

applicationContext-mybatis.xml（spring和mybatis整合的配置文件）

两个类：

DataSourceContextHolder.java

DynamicDataSource.java

一个标识：

实体类和数据库表的对应。

## 第二步：jdbc.properties数据连接信息配置

1. #users用户信息对应的数据库
2. users.jdbc.driver=com.mysql.jdbc.Driver
3. users.jdbc.url=jdbc:mysql://127.0.0.1:3306/users?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true
4. users.jdbc.username=root
5. users.jdbc.password=123456
7. #product商品信息对应的数据库
8. product.jdbc.driver=com.mysql.jdbc.Driver
9. product.jdbc.url=jdbc:mysql://127.0.0.1:3306/product?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true
10. product.jdbc.username=root
11. product.jdbc.password=123456

## 第三步：applicationContext.xml核心配置文件内容如下

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:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7. http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
8. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
10. *<!-- 使用spring自带的占位符替换功能 -->*
11. <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
12. *<!-- 允许JVM参数覆盖 -->*
13. <property name="systemPropertiesModeName" value="SYSTEM\_PROPERTIES\_MODE\_OVERRIDE" />
14. *<!-- 忽略没有找到的资源文件 -->*
15. <property name="ignoreResourceNotFound" value="true" />
16. *<!-- 配置资源文件 -->*
17. <property name="locations">
18. <list>
19. <value>classpath:jdbc.properties</value>
20. </list>
21. </property>
22. </bean>
24. <context:component-scan base-package="com.millery" />
26. *<!-- 数据库连接池 :usersDataSource -->*
27. <bean id="usersDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">
28. *<!-- 数据库驱动 -->*
29. <property name="driverClass" value="${users.jdbc.driver}" />
30. *<!-- 相应驱动的jdbcUrl -->*
31. <property name="jdbcUrl" value="${users.jdbc.url}" />
32. *<!-- 数据库的用户名 -->*
33. <property name="username" value="${users.jdbc.username}" />
34. *<!-- 数据库的密码 -->*
35. <property name="password" value="${users.jdbc.password}" />
36. *<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->*
37. <property name="idleConnectionTestPeriod" value="60" />
38. *<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->*
39. <property name="idleMaxAge" value="30" />
40. *<!-- 每个分区最大的连接数 -->*
41. <property name="maxConnectionsPerPartition" value="150" />
42. *<!-- 每个分区最小的连接数 -->*
43. <property name="minConnectionsPerPartition" value="5" />
44. </bean>
45. *<!-- 数据库连接池 :productDataSource -->*
46. <bean id="productDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">
47. *<!-- 数据库驱动 -->*
48. <property name="driverClass" value="${product.jdbc.driver}" />
49. *<!-- 相应驱动的jdbcUrl -->*
50. <property name="jdbcUrl" value="${product.jdbc.url}" />
51. *<!-- 数据库的用户名 -->*
52. <property name="username" value="${product.jdbc.username}" />
53. *<!-- 数据库的密码 -->*
54. <property name="password" value="${product.jdbc.password}" />
55. *<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->*
56. <property name="idleConnectionTestPeriod" value="60" />
57. *<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->*
58. <property name="idleMaxAge" value="30" />
59. *<!-- 每个分区最大的连接数 -->*
60. <property name="maxConnectionsPerPartition" value="150" />
61. *<!-- 每个分区最小的连接数 -->*
62. <property name="minConnectionsPerPartition" value="5" />
63. </bean>


67. *<!-- 数据库连接池 -->*
68. <bean id="dynamicDataSource" class="com.millery.utils.DynamicDataSource">
69. <property name="targetDataSources">
70. <map key-type="java.lang.String">
71. <entry value-ref="usersDataSource" key="usersDataSource" />
72. <entry value-ref="productDataSource" key="productDataSource" />
73. </map>
74. </property>
76. *<!-- 默认使用productDataSource的数据源 -->*
77. <property name="defaultTargetDataSource" ref="productDataSource" />
78. </bean>

## 第四步：application-transaction.xml，spring事务管理配置文件

1. <beans xmlns="http://www.springframework.org/schema/beans"
2. xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"
3. xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
6. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
7. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
8. http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd">
10. *<!-- 定义事务管理器 -->*
11. <bean id="transactionManager"
12. class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
13. *<!-- 使用动态的数据源dynamicDataSource -->*
14. <property name="dataSource" ref="dynamicDataSource" />
15. </bean>
17. *<!-- 定义事务策略 -->*
18. <tx:advice id="txAdvice" transaction-manager="transactionManager">
19. <tx:attributes>
20. *<!--所有以query开头的方法都是只读的 -->*
21. <tx:method name="query\*" read-only="true" />
22. *<!--其他方法使用默认事务策略 -->*
23. <tx:method name="\*" />
24. </tx:attributes>
25. </tx:advice>
27. <aop:config>
28. *<!--pointcut元素定义一个切入点，execution中的第一个星号 用以匹配方法的返回类型， 这里星号表明匹配所有返回类型。 com.abc.dao.\*.\*(..)表明匹配cn.millery.mybatis.service包下的所有类的所有*
29. *方法 -->*
30. <aop:pointcut id="myPointcut"
31. expression="execution(\* com.millery.service.\*.\*(..))" />
32. *<!--将定义好的事务处理策略应用到上述的切入点 -->*
33. <aop:advisor advice-ref="txAdvice" pointcut-ref="myPointcut" />
34. </aop:config>
36. </beans>

## 第五步：mybatis-config.xml，mybatis核心配置文件

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE configuration
3. PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4. "http://mybatis.org/dtd/mybatis-3-config.dtd">
5. <configuration>
6. <settings>
7. *<!-- 开启驼峰自动映射 -->*
8. <setting name="mapUnderscoreToCamelCase" value="true" />
9. </settings>
10. </configuration>

## 第六步：applicationContex-mybatis.xml，spring和mybatis整合配置文件

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:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7. http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
8. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
10. *<!-- 定义Mybatis的SqlSessionFactory -->*
11. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
12. *<!-- 定义数据源,数据源使用动态的数据源dynamicDataSource -->*
13. <property name="dataSource" ref="dynamicDataSource" />
14. *<!-- 指定mybatis全局配置文件 -->*
15. <property name="configLocation" value="classpath:mybatis/mybatis-config.xml" />
16. *<!-- 扫描mappers目录以及子目录下的所有xml文件 -->*
17. <property name="mapperLocations" value="classpath:mybatis/mappers/\*\*/\*.xml" />
18. *<!-- 别名扫描包 -->*
19. <property name="typeAliasesPackage" value="com.millery.pojo" />
20. </bean>
21. *<!-- 定义Mapper接口扫描器 -->*
22. <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
23. <property name="basePackage" value="com.millery.mapper" />
24. </bean>
25. </beans>

## 第七步：创建两个类，放到com.millery.utils包中

DataSourceContextHolder类：

1. package com.millery.utils;
2. public class DataSourceContextHolder {
3. private static final ThreadLocal<String> contextHolder = new ThreadLocal<String>();
4. public static void setDbType(String dbType) {
5. contextHolder.set(dbType);
6. }
7. public static String getDbType() {
8. return ((String) contextHolder.get());
9. }
10. public static void clearDbType() {
11. contextHolder.remove();
12. }
13. }

DynamicDataSource类：

1. package com.millery.utils;
2. import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;
3. public class DynamicDataSource extends AbstractRoutingDataSource {
4. @Override
5. protected Object determineCurrentLookupKey() {
6. return DataSourceContextHolder.getDbType();
7. }
8. }

## 第八步：关键的一步，一个标识，在设置动态数据源的时候，连接了两个库，但是在怎 么确定每次连接都是需要连接的数据库呢，那就要这个标识了。

假设users数据库中有一张表交user，这张表对应到java代码中有个实体类。在实体类中我们都要加注解@Table(name=”表名”)，此时的表名不能再直接写user，而是写users.user(数据库名.数据表)，这里的数据库名就是关键的一个标识。如果现在product数据库中有一个item数据表，此时同上面的一样，加上product数据库的标识即可。

## 最后一步：就是写一个测试的类进行测试，这里就偷个懒不写，读者根据自己的需求写。

注意：

1、此配置是通过讨论测试成功的方案，比之前的个人版本更优化，配置更简洁，如果有更多的数据库连接，只需要在spring核心配置文件中添加一个数据源，然后配置到动态数据源中即可；

2、配置仅供参考，直接复制，可能出现异常，毕竟每个人电脑的环境不是完全相同的。

鸣谢：此博客是经过讨论和参考朋友博客完成，特此感谢在网上共享研究成果资源的朋 友。