

[首页](#) [资讯](#) [精华](#) [论坛](#) [问答](#) [博客](#) [专栏](#) [群组](#) [更多 ▼](#)  
[您还未登录！](#) [登录](#) [注册](#)

## java 职业生涯

- [博客](#)
- [微博](#)
- [相册](#)
- [收藏](#)
- [留言](#)
- [关于我](#)





### 基于注解的mybatis


博客分类:

- [数据库](#)

首先当然得下载mybatis-3.0.5.jar和mybatis-spring-1.0.1.jar两个JAR包，并放在WEB-INF的lib目录下(如果你使用maven，则jar会根据你的pom配置的依赖自动下载，并存放在你指定的maven本地库中，默认是~/.m2/repository)，前一个是mybatis核心包，后一个是和spring整合的包。

使用mybatis，必须有个全局配置文件configuration.xml，来配置mybatis的缓存，延迟加载等一系列属性，该配置文件示例如下：

Java代码  



Java代码 


```
1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE configuration
3.   PUBLIC "-//ibatis.apache.org//DTD Config 3.0//EN"
4.   "http://ibatis.apache.org/dtd/ibatis-3-config.dtd">
5. <configuration>
6.   <settings>
7.     <!-- 全局映射器启用缓存 -->
8.     <setting name="cacheEnabled" value="true" />
9.     <!-- 查询时，关闭关联对象即时加载以提高性能 -->
10.    <setting name="lazyLoadingEnabled" value="true" />
11.    <!-- 设置关联对象加载的形态，此处为按需加载字段(加载字段由SQL指定)，不会加载关联表的所有
    字段，以提高性能 -->
12.    <setting name="aggressiveLazyLoading" value="false" />
13.    <!-- 对于未知的SQL查询，允许返回不同的结果集以达到通用的效果 -->
14.    <setting name="multipleResultSetsEnabled" value="true" />
15.    <!-- 允许使用列标签代替列名 -->
16.    <setting name="useColumnLabel" value="true" />
17.    <!-- 允许使用自定义的主键值(比如由程序生成的UUID 32位编码作为键值)，数据表的PK生成策略将被覆
    盖 -->
18.    <setting name="useGeneratedKeys" value="true" />
19.    <!-- 给予被嵌套的resultMap以字段-属性的映射支持 -->
20.    <setting name="autoMappingBehavior" value="FULL" />
21.    <!-- 对于批量更新操作缓存SQL以提高性能 -->
22.    <setting name="defaultExecutorType" value="BATCH" />
23.    <!-- 数据库超过25000秒仍未响应则超时 -->
24.    <setting name="defaultStatementTimeout" value="25000" />
25.  </settings>
26.  <!-- 全局别名设置，在映射文件中只需写别名，而不必写出整个类路径 -->
27.  <typeAliases>
28.    <typeAlias alias="TestBean"
29.      type="com.wotao.taotao.persist.test.dataobject.TestBean" />
30.  </typeAliases>
31.  <!-- 非注解的sql映射文件配置，如果使用mybatis注解，该mapper无需配置，但是如果mybatis注解中包
    含@resultMap注解，则mapper必须配置，给resultMap注解使用 -->
32.  <mappers>
33.    <mapper resource="persist/test/orm/test.xml" />
34.  </mappers>
```

## 35. &lt;/configuration&gt;

该文件放在资源文件的任意classpath目录下，假设这里就直接放在资源根目录，等会spring需要引用该文件。

查看ibatis-3-config.dtd发现除了settings和typeAliases还有其他众多元素，比如properties,objectFactory,environments等等，这些元素基本上都包含着一些环境配置，数据源定义，数据库事务等等，在单独使用mybatis的时候非常重要，比如通过以构造参数的形式去实例化一个sqlSessionFactory，就像这样：



Java代码  


Java代码 

1. SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader);
2. SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, properties);
3. SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, environment, properties);

而typeHandlers则用来自定义映射规则，如你可以自定义将Character映射为varchar，plugins元素则放了一些拦截器接口，你可以继承他们并做一些切面的事情，至于每个元素的细节和使用，你参考mybatis用户指南即可。

现在我们用的是spring，因此除settings和typeAliases元素之外，其他元素将会失效，故不在此配置，spring会覆盖这些元素的配置，比如在spring配置文件中指定c3p0数据源定义如下：

Java代码  

Java代码 



```


1. <!-- c3p0 connection pool configuration -->
2. <bean id="testDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
3.     destroy-method="close">
4.     <!-- 数据库驱动 -->
5.     <property name="driverClass" value="{db.driver.class}" />
6.     <!-- 连接URL串 -->
7.     <property name="jdbcUrl" value="{db.url}" />
8.     <!-- 连接用户名 -->
9.     <property name="user" value="{db.username}" />
10.    <!-- 连接密码 -->
11.    <property name="password" value="{db.password}" />
12.    <!-- 初始化连接池时连接数量为5个 -->
13.    <property name="initialPoolSize" value="5" />
14.    <!-- 允许最小连接数量为5个 -->
15.    <property name="minPoolSize" value="5" />
16.    <!-- 允许最大连接数量为20个 -->
17.    <property name="maxPoolSize" value="20" />
18.    <!-- 允许连接池最大生成100个PreparedStatement对象 -->
19.    <property name="maxStatements" value="100" />
20.    <!-- 连接有效时间，连接超过3600秒未使用，则该连接丢弃 -->
21.    <property name="maxIdleTime" value="3600" />
22.    <!-- 连接用完时，一次产生的新连接步进值为2 -->
23.    <property name="acquireIncrement" value="2" />
24.    <!-- 获取连接失败后再尝试10次，再失败则返回DAOException异常 -->
25.    <property name="acquireRetryAttempts" value="10" />
26.    <!-- 获取下一次连接时最短间隔600毫秒，有助于提高性能 -->
27.    <property name="acquireRetryDelay" value="600" />
28.    <!-- 检查连接的有效性，此处小弟不是很懂什么意思 -->
29.    <property name="testConnectionOnCheckin" value="true" />
30.    <!-- 每个1200秒检查连接对象状态 -->
31.    <property name="idleConnectionTestPeriod" value="1200" />
32.    <!-- 获取新连接的超时时间为10000毫秒 -->
33.    <property name="checkoutTimeout" value="10000" />
34. </bean>

```

配置中的\${}都是占位符，在你指定数据库驱动打war时会自动替换，替换的值在你的父pom中配置，至于c3p0连接池的各种属性详细信息和用法，你自行参考c3p0的官方文档，这里要说明的是checkoutTimeout元素，记得千万要设大一点，单位是毫秒，假如设置太小，有可能会没等数据库响应就直接超时了，小弟在这里吃了不少苦头，还是基本功太差。



数据源配置妥当之后，我们就要开始非常重要的sessionFactory配置了，无论是hibernate还是mybatis，都需要一个sessionFactory来生成session， sessionFactory配置如下：


Java代码  

Java代码 

```
1. <bean id="testSqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
2.   <property name="configLocation" value="classpath:configuration.xml" />
3.   <property name="dataSource" ref="testDataSource" />
4. </bean>
```

testSqlSessionFactory有两处注入，一个就是前面提到的mybatis全局设置文件configuration.xml，另一个就是上面定义的数据源了（注：hibernate的sessionFactory只需注入hibernate.cfg.xml，数据源定义已经包含在该文件中），好了， sessionFactory已经产生了，由于我们用的mybatis3的注解，因此spring的sqlSessionTemplate也不用配置了， sqlSessionTemplate也不用注入到我们的BaseDAO中了，相应的，我们需要配置一个映射器接口来对应sqlSessionTemplate，该映射器接口定义了你自己的接口方法，具体实现不用关心，代码如下：



Java代码  


Java代码 

```
1. <!-- data OR mapping interface -->
2. <bean id="testMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
3.   <property name="sqlSessionFactory" ref="testSqlSessionFactory" />
4.   <property name="mapperInterface" value="com.wotao.taotao.persist.test.mapper.TestMapper" />
5. </bean>
```

对应于sqlSessionTemplate， testMapper同样需要testSqlSessionFactory注入，另外一个注入就是你自己定义的Mapper接口，该接口定义了操作数据库的方法和SQL语句以及很多的注解，稍后我会讲到。到此， mybatis和spring整合的文件配置就算OK了（注：如果你需要开通spring对普通类的代理功能，那么你需要在spring配置文件中加入<aop:aspectj-autoproxy />），至于其他的如事务配置，AOP切面注解等内容不在本文范围内，不作累述。

至此，一个完整的mybatis整合spring的配置文件看起来应该如下所示：

Java代码  

Java代码 



```
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:context="http://www.springframework.org/schema/context"
4.   xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop"
5.   xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
6. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd
7. http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
8. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
9.
10. <!-- c3p0 connection pool configuration -->
11. <bean id="testDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
12.   destroy-method="close">
13.   <property name="driverClass" value="${db.driver.class}" />
14.   <property name="jdbcUrl" value="${db.url}" />
15.   <property name="user" value="${db.username}" />
16.   <property name="password" value="${db.password}" />
17.   <property name="initialPoolSize" value="5" />
18.   <property name="minPoolSize" value="5" />
19.   <property name="maxPoolSize" value="20" />
20.   <property name="maxStatements" value="100" />
21.   <property name="maxIdleTime" value="3600" />
22.   <property name="acquireIncrement" value="2" />
23.   <property name="acquireRetryAttempts" value="10" />
24.   <property name="acquireRetryDelay" value="600" />
25.   <property name="testConnectionOnCheckin" value="true" />
26.   <property name="idleConnectionTestPeriod" value="1200" />
27.   <property name="checkoutTimeout" value="10000" />
28. </bean>
29. <bean id="testSqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
30.   <property name="configLocation" value="classpath:configuration.xml" />
```


```

31.     <property name="dataSource" ref="testDataSource" />
32. </bean>
33. <!-- data OR mapping interface -->
34. <bean id="testMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
35.     <property name="sqlSessionFactory" ref="testSqlSessionFactory" />
36.     <property name="mapperInterface" value="com.wotao.taotao.persist.test.mapper.TestMapper" />
37. </bean>
38.
39. <!-- add your own Mapper here -->
40.
41. <!-- comment here, using annotation -->
42. <!-- <bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate"> -->
43. <!-- <constructor-arg index="0" ref="sqlSessionFactory" /> -->
44. <!-- </bean> -->
45. <!-- base DAO class, for module business, extend this class in DAO -->
46. <!-- <bean id="testBaseDAO" class="com.test.dao.TestBaseDAO"> -->
47. <!-- <property name="sqlSessionTemplate" ref="sqlSessionTemplate" /> -->
48. <!-- </bean> -->
49. <!-- <bean id="testDAO" class="com.test.dao.impl.TestDAOImpl" /> -->
50.
51. <!-- you can DI Bean if you don't like use annotation -->
52.
53. </beans>

```

到此为止，我们只讲了mybatis和spring的整合，还没有真正触及mybatis的核心：使用mybatis注解代替映射文件编程（不过官方文档也说了，如果真正想发挥mybatis功能，还是需要用到映射文件，看来mybatis自己都对mybatis注解没信心，呵呵），通过上述内容，我们知道配置搞定，但是testMapper还没有被实现，而注解的使用，全部集中在这个testMapper上，是mybatis注解的核心所在，先来看一下这个testMapper接口是个什么样的：

Java代码  

Java代码 

```

1. /**
2.  * The test Mapper interface.
3.  *
4.  * @author HuangMin <a href="mailto:minhuang@hengtiansoft.com">send email</a>
5.  *
6.  * @since 1.6
7.  * @version 1.0
8.  *
9.  *      #~TestMapper.java 2011-9-23 : afternoon 10:51:40
10. */
11. @CacheNamespace(size = 512)
12. public interface TestMapper {
13.
14.     /**
15.      * get test bean by UID.
16.      *
17.      * @param id
18.      * @return
19.      */
20.     @SelectProvider(type = TestSqlProvider.class, method = "getSql")
21.     @Options(useCache = true, flushCache = false, timeout = 10000)
22.     @Results(value = {
23.         @Result(id = true, property = "id", column = "test_id", javaType = String.class, jdbcType = JdbcType.VARCHAR),
24.         @Result(property = "testText", column = "test_text", javaType = String.class, jdbcType = JdbcType.VARCHAR) })
25.     public TestBean get(@Param("id") String id);
26.
27.     /**
28.      * get all tests.
29.      *
30.      * @return
31.      */
32.     @SelectProvider(type = TestSqlProvider.class, method = "getAllSql")
33.     @Options(useCache = true, flushCache = false, timeout = 10000)
34.     @Results(value = {
35.         @Result(id = true, property = "id", column = "test_id", javaType = String.class, jdbcType = JdbcType.VARCHAR),
36.         @Result(property = "testText", column = "test_text", javaType = String.class, jdbcType = JdbcType.VARCHAR) })

```

```

37. public List<TestBean> getAll();
38.
39. /**
40.  * get tests by test text.
41.  *
42.  * @param testText
43.  * @return
44.  */
45. @SelectProvider(type = TestSqlProvider.class, method = "getByTestTextSql")
46. @Options(useCache = true, flushCache = false, timeout = 10000)
47. @ResultMap(value = "getByTestText")
48. public List<TestBean> getByTestText(@Param("testText") String testText);
49.
50. /**
51.  * insert a test bean into database.
52.  *
53.  * @param testBean
54.  */
55. @InsertProvider(type = TestSqlProvider.class, method = "insertSql")
56. @Options(flushCache = true, timeout = 20000)
57. public void insert(@Param("testBean") TestBean testBean);
58.
59. /**
60.  * update a test bean with database.
61.  *
62.  * @param testBean
63.  */
64. @UpdateProvider(type = TestSqlProvider.class, method = "updateSql")
65. @Options(flushCache = true, timeout = 20000)
66. public void update(@Param("testBean") TestBean testBean);
67.
68. /**
69.  * delete a test by UID.
70.  *
71.  * @param id
72.  */
73. @DeleteProvider(type = TestSqlProvider.class, method = "deleteSql")
74. @Options(flushCache = true, timeout = 20000)
75. public void delete(@Param("id") String id);
76. }

```

下面逐个对里面的注解进行分析：

**@CacheNamespace(size = 512)：** 定义在该命名空间内允许使用内置缓存，最大值为512个对象引用，读写默认是开启的，缓存内省刷新时间为默认3600000毫秒，写策略是拷贝整个对象镜像到全新堆（如同CopyOnWriteList）因此线程安全。

**@SelectProvider(type = TestSqlProvider.class, method = "getSql")：** 提供查询的SQL语句，如果你不用这个注解，你也可以直接使用**@Select("select \* from ....")**注解，把查询SQL抽取到一个类里面，方便管理，同时复杂的SQL也容易操作，**type = TestSqlProvider.class**就是存放SQL语句的类，而**method = "getSql"**表示get接口方法需要到TestSqlProvider类的getSql方法中获取SQL语句。

**@Options(useCache = true, flushCache = false, timeout = 10000)：** 一些查询的选项开关，比如**useCache = true**表示本次查询结果被缓存以提高下次查询速度，**flushCache = false**表示下次查询时不刷新缓存，**timeout = 10000**表示查询结果缓存10000秒。



**@Results(value = {  
@Result(id = true, property = "id", column = "test\_id", javaType = String.class, jdbcType = JdbcType.VARCHAR),  
@Result(property = "testText", column = "test\_text", javaType = String.class, jdbcType = JdbcType.VARCHAR) })：** 表示sql查询返回的结果集，@Results是以@Result为元素的数组，@Result表示单条属性-字段的映射关系，如：**@Result(id = true, property = "id", column = "test\_id", javaType = String.class, jdbcType = JdbcType.VARCHAR)**可以简写为：**@Result(id = true, property = "id", column = "test\_id")**，**id = true**表示这个test\_id字段是个PK，查询时mybatis会给予必要的优化，应该说数组中所有的@Result组成了单个记录的映射关系，而@Results则单个记录的集合。另外还有一个非常重要的注解@ResultMap也和@Results差不多，到时讲到。


**@Param("id")：** 全局限定别名，定义查询参数在sql语句中的位置不再是顺序下标0,1,2,3....的形式，而是对应名称，该名称就在这里定义。

**@ResultMap(value = "getByTestText")：** 重要的注解，可以解决复杂的映射关系，包括resultMap嵌套，鉴别器discriminator等等。注意一旦你启用该注解，你将不得不在你的映射文件中配置你的resultMap，而value = "getByTestText"即为映射文件



中的resultMap ID(注意此处的value = "getByTestText", 必须是在映射文件中指定命名空间路径)。@ResultMap在某些简单场合可以用@Results代替, 但是复杂查询, 比如联合、嵌套查询@ResultMap就会显得解耦方便更容易管理。一个映射文件如下所示:

Java代码  

Java代码 

```
1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!DOCTYPE mapper
3. PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
4. "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">
5.
6. <mapper namespace="com.wotao.taotao.persist.test.mapper.TestMapper">
7.   <resultMap id="getByTestText" type="TestBean">
8.     <id property="id" column="test_id" javaType="string" jdbcType="VARCHAR" />
9.     <result property="testText" column="test_text" javaType="string" jdbcType="VARCHAR" />
10.   </resultMap>
11. </mapper>
```

注意文件中的namespace路径必须是使用@resultMap的类路径, 此处是TestMapper, 文件中 id="getByTestText"必须和@resultMap中的value = "getByTestText"保持一致。

@InsertProvider(type = TestSqlProvider.class, method = "insertSql") : 用法和含义@SelectProvider一样, 只不过是用来插入数据库而用的。



@Options(flushCache = true, timeout = 20000) : 对于需要更新数据库的操作, 需要重新刷新缓存flushCache = true使缓存同步。


@UpdateProvider(type = TestSqlProvider.class, method = "updateSql") : 用法和含义@SelectProvider一样, 只不过是用来更新数据库而用的。

@Param("testBean") : 是一个自定义的对象, 指定了sql语句中的表现形式, 如果要在sql中引用对象里面的属性, 只要使用testBean.id, testBean.textText即可, mybatis会通过反射找到这些属性值。

@DeleteProvider(type = TestSqlProvider.class, method = "deleteSql") : 用法和含义@SelectProvider一样, 只不过是用来删除数据而用的。

现在mybatis注解基本已经讲完了, 接下来我们就要开始写SQL语句了, 因为我们不再使用映射文件编写SQL, 那么就不得不在java类里面写, 就像上面提到的, 我们不得不在TestSqlProvider这个类里面写SQL, 虽然已经把所有sql语句集中到了一个类里面去管理, 但听起来似乎仍然有点恶心, 幸好mybatis提供SelectBuilder和SqlBuilder这两个小工具来帮助我们生成SQL语句, SelectBuilder专门用来生成select语句, 而SqlBuilder则是一般性的工具, 可以生成任何SQL语句, 我这里选择了SqlBuilder来生成, TestSqlProvider代码如下:

Java代码  

Java代码 

```
1. /*
2.  * #~ test-afternoon10:51:40
3.  */
4. package com.wotao.taotao.persist.test.sqlprovider;
5.
6. import static org.apache.ibatis.jdbc.SqlBuilder.BEGIN;
7. import static org.apache.ibatis.jdbc.SqlBuilder.FROM;
8. import static org.apache.ibatis.jdbc.SqlBuilder.SELECT;
9. import static org.apache.ibatis.jdbc.SqlBuilder.SQL;
10. import static org.apache.ibatis.jdbc.SqlBuilder.WHERE;
11. import static org.apache.ibatis.jdbc.SqlBuilder.DELETE_FROM;
12. import static org.apache.ibatis.jdbc.SqlBuilder.INSERT INTO;
13. import static org.apache.ibatis.jdbc.SqlBuilder.SET;
14. import static org.apache.ibatis.jdbc.SqlBuilder.UPDATE;
15. import static org.apache.ibatis.jdbc.SqlBuilder.VALUES;
16.
17. import java.util.Map;
18.
19. /**
20.  * The test sql Provider,define the sql script for mapping.
21.  *
22.  * @author HuangMin <a href="mailto:minhuang@hengtiansoft.com">send email</a>
```

```
23. *
24. * @since 1.6
25. * @version 1.0
26. *
27. *      #~TestSqlProvider.java 2011-9-23 : afternoon 10:51:40
28. */
29. public class TestSqlProvider {
30.
31.     /** table name, here is test */
32.     private static final String TABLE_NAME = "test";
33.
34.     /**
35.      * get test by id sql script.
36.      *
37.      * @param parameters
38.      * @return
39.      */
40.     public String getSql(Map<String, Object> parameters) {
41.         String uid = (String) parameters.get("id");
42.         BEGIN();
43.         SELECT("test_id, test_text");
44.         FROM(TABLE_NAME);
45.         if (uid != null) {
46.             WHERE("test_id = #{id,javaType=string,jdbcType=VARCHAR}");
47.         }
48.         return SQL();
49.     }
50.
51.     /**
52.      * get all tests sql script.
53.      *
54.      * @return
55.      */
56.     public String getAllSql() {
57.         BEGIN();
58.         SELECT("test_id, test_text");
59.         FROM(TABLE_NAME);
60.         return SQL();
61.     }
62.
63.     /**
64.      * get test by test text sql script.
65.      *
66.      * @param parameters
67.      * @return
68.      */
69.     public String getByTestTextSql(Map<String, Object> parameters) {
70.         String tText = (String) parameters.get("testText");
71.         BEGIN();
72.         SELECT("test_id, test_text");
73.         FROM(TABLE_NAME);
74.         if (tText != null) {
75.             WHERE("test_text like #{testText,javaType=string,jdbcType=VARCHAR}");
76.         }
77.         return SQL();
78.     }
79.
80.     /**
81.      * insert a test sql script.
82.      *
83.      * @return
84.      */
85.     public String insertSql() {
86.         BEGIN();
87.         INSERT INTO(TABLE_NAME);
88.         VALUES("test_id", "#{testBean.id,javaType=string,jdbcType=VARCHAR}");
89.         VALUES("test_text", "#{testBean.testText,javaType=string,jdbcType=VARCHAR}");
90.         return SQL();
91.     }
92.
```

```

93.  /**
94.   * update a test sql script.
95.   *
96.   * @return
97.   */
98.  public String updateSql() {
99.      BEGIN();
100.      UPDATE(TABLE_NAME);
101.      SET("test_text = #{testBean.testText,javaType=string,jdbcType=VARCHAR}");
102.      WHERE("test_id = #{testBean.id,javaType=string,jdbcType=VARCHAR}");
103.      return SQL();
104.  }
105.
106.  /**
107.   * delete a test sql script.
108.   *
109.   * @return
110.   */
111.  public String deleteSql() {
112.      BEGIN();
113.      DELETE_FROM(TABLE_NAME);
114.      WHERE("test_id = #{id,javaType=string,jdbcType=VARCHAR}");
115.      return SQL();
116.  }
117. }



```


BEGIN();表示刷新本地线程，某些变量为了线程安全，会先在本地存放变量，此处需要刷新。

SELECT，FROM，WHERE等都是sqlbuilder定义的公用静态方法，用来组成你的sql字符串。如果你在testMapper中调用该方法的某个接口方法已经定义了参数@Param()，那么该方法的参数Map<String, Object> parameters即组装了@Param()定义

的参数，比如testMapper接口方法中定义参数为@Param("testId"),@Param("testText")，那么parameters的形态就是：  
[key="testId",value=object1],[key="testText",value=object2]，如果接口方法没有定义@Param()，那么parameters的key就是参数的顺序小标：  
[key=0,value=object1],[key=1,value=object2]，SQL()将返回最终append结束的字符串，sql语句中的形如  
#{id,javaType=string,jdbcType=VARCHAR}完全可简写为#{id}，我只是为了规整如此写而已。另外，对于复杂查询还有很多标签可用，比如：JOIN，INNER\_JOIN，GROUP\_BY，ORDER\_BY等等，具体使用详情，你可以查看源码。

最后记得把你的Mapper接口注入到你的DAO类中，在DAO中引用Mapper接口方法即可。我在BaseDAO中的注解注入如下：



Java代码  


Java代码 

```

1. ....
2. @Repository("testBaseDAO")
3. public class TestBaseDAO {
4. ....

```



Java代码  

Java代码 

```

1. ....
2. /**
3.  * @param testMapper
4.  *      the testMapper to set
5.  */
6. @Autowired
7. public void setTestMapper(@Qualifier("testMapper") TestMapper testMapper) {
8.     this.testMapper = testMapper;
9. }
10. ....

```

分享到:  

[自动登录](#) | [轻松实现Apache.Tomcat集群和负载均衡](#)

- 2011-11-04 20:34
- 浏览 6681