跟我学spring3（8.3）：对ORM的支持之集成iBATIS

2016/03/01 | 分类： [教程](http://www.importnew.com/cat/tutorial) | [0 条评论](http://www.importnew.com/18222.html#respond) | 标签： [IBATIS](http://www.importnew.com/tag/ibatis), [ORM](http://www.importnew.com/tag/orm), [SPRING](http://www.importnew.com/tag/spring)

分享到：**4**

原文出处： [张开涛](http://sishuok.com/forum/blogPost/list/0/2498.html#7216)

8.3  集成iBATIS

iBATIS是一个半自动化的ORM框架，需要通过配置方式指定映射SQL语句，而不是由框架本身生成（如Hibernate自动生成对应SQL来持久化对象），即Hibernate属于全自动ORM框架。

Spring提供对iBATIS 2.X的集成，提供一致的异常体系、一致的DAO访问支持、Spring管理事务支持。

**Spring 2.5.5+版本支持iBATIS 2.3+版本，不支持低版本。**

8.3.1  如何集成

Spring通过使用如下Bean进行集成iBATIS：

* SqlMapClientFactoryBean：用于集成iBATIS。

configLocation和configLocations：用于指定SQL Map XML配置文件，用于指定如数据源等配置信息；

mappingLocations：用于指定SQL Map映射文件，即半自动概念中的SQL语句定义；

sqlMapClientProperties：定义iBATIS 配置文件配置信息；

dataSource：定义数据源。

如果在Spring配置文件中指定了DataSource，就不要在iBATIS配置文件指定了，否则Spring配置文件指定的DataSource将覆盖iBATIS配置文件中定义的DataSource。

接下来示例一下如何集成iBATIS：

**1、准备需要的jar包，从spring-framework-3.0.5.RELEASE-dependencies.zip中拷贝如下jar包：**

**com.springsource.com.ibatis-2.3.4.726.jar**

**2、 对象模型定义，此处使用第七章中的UserModel；**

**3、**iBATIS**映射定义（chapter8/sqlmaps/UserSQL.xml）：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"      "http://ibatis.apache.org/dtd/sql-map-2.dtd">  <sqlMap namespace="UserSQL">    <statement id="createTable">      <!--id自增主键从0开始 -->      <![CDATA[          create memory table test(            id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,            name varchar(100))      ]]>    </statement>    <statement id="dropTable">      <![CDATA[  drop table test  ]]>    </statement>    <insert id="insert" parameterClass="cn.javass.spring.chapter7.UserModel">      <![CDATA[        insert into test(name) values (#myName#)      ]]>      <selectKey resultClass="int" keyProperty="id" type="post">        <!-- 获取hsqldb插入的主键 -->        call identity();        <!-- mysql使用select last\_insert\_id();获取插入的主键 -->      </selectKey>    </insert>  </sqlMap> |

**4、 iBATIS配置文件（chapter8/sql-map-config.xml）定义：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE sqlMapConfig PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"      "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">    <sqlMapConfig>      <settings enhancementEnabled="true" useStatementNamespaces="true"          maxTransactions="20" maxRequests="32" maxSessions="10"/>      <sqlMap resource="chapter8/sqlmaps/UserSQL.xml"/>  </sqlMapConfig> |

**5、 数据源定义，此处使用第7章的配置文件，即“chapter7/applicationContext-resources.xml”文件。**

**6、 SqlMapClient配置（chapter8/applicationContext-ibatis.xml）定义：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <bean id="sqlMapClient"      class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">      <!-- 1、指定数据源 -->      <property name="dataSource" ref="dataSource"/>      <!-- 2、指定配置文件 -->      <property name="configLocation" value="chapter8/sql-map-config.xml"/>  </bean> |

**7、 获取SqlMapClient：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package cn.javass.spring.chapter8;  //省略import  public class IbatisTest {      private static SqlMapClient sqlMapClient;      @BeforeClass      public static void setUpClass() {        String[] configLocations = new String[] {            "classpath:chapter7/applicationContext-resources.xml",            "classpath:chapter8/applicationContext-ibatis.xml"};        ApplicationContext ctx = new ClassPathXmlApplicationContext(configLocations);        sqlMapClient = ctx.getBean(SqlMapClient.class);    }  } |

此处我们使用了chapter7/applicationContext-resources.xml定义的“dataSource”数据源，通过ctx.getBean(SqlMapClient.class)获取SqlMapClient。

**8、 通过SqlMapClient创建和删除表：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Before  public void setUp() throws SQLException {      sqlMapClient.update("UserSQL.createTable");  }  @After  public void tearDown() throws SQLException {      sqlMapClient.update("UserSQL.dropTable");  } |

**9、 使用SqlMapClient进行对象持久化：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | @Test  public void testFirst() throws SQLException {      UserModel model = new UserModel();      model.setMyName("test");      SqlMapSession session = null;      try {          session = sqlMapClient.openSession();          beginTransaction(session);          session.insert("UserSQL.insert", model);          commitTransaction(session);      } catch (SQLException e) {          rollbackTransacrion(session);          throw e;      } finally {          closeSession(session);      }  }  private void closeSession(SqlMapSession session) {     session.close();  }  private void rollbackTransacrion(SqlMapSession session) throws SQLException {      if(session != null) {          session.endTransaction();      }  }  private void commitTransaction(SqlMapSession session) throws SQLException {      session.commitTransaction();  }  private void beginTransaction(SqlMapSession session) throws SQLException {      session.startTransaction();  } |

同样令人心烦的事务管理和冗长代码，Spring通用提供了SqlMapClientTemplate模板类来解决这些问题。

8.3.2  使用 SqlMapClientTemplate

SqlMapClientTemplate模板类同样用于简化事务管理及常见操作，类似于JdbcTemplate模板类，对于复杂操作通过提供SqlMapClientCallback回调接口来允许更复杂的操作。

接下来示例一下SqlMapClientTemplate的使用：

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @Test  public void testSqlMapClientTemplate() {  SqlMapClientTemplate sqlMapClientTemplate =  new SqlMapClientTemplate(sqlMapClient);      final UserModel model = new UserModel();      model.setMyName("myName");      sqlMapClientTemplate.insert("UserSQL.insert", model);      //通过回调允许更复杂操作      sqlMapClientTemplate.execute(new SqlMapClientCallback<Void>() {          @Override          public Void doInSqlMapClient(SqlMapExecutor session) throws SQLException {              session.insert("UserSQL.insert", model);              return null;      }});  } |

通过new SqlMapClientTemplate(sqlMapClient)创建HibernateTemplate模板类对象，通过调用模板类的save方法持久化对象，并且自动享受到Spring管理事务的好处。

而且SqlMapClientTemplate提供使用SqlMapClientCallback回调接口的方法execute用来支持复杂操作，当然也自动享受到Spring管理事务的好处。

8.3.3集成iBATIS及最佳实践

类似于JdbcDaoSupport类，Spring对iBATIS也提供了SqlMapClientDaoSupport类来支持一致的数据库访问。SqlMapClientDaoSupport也是DaoSupport实现：

接下来示例一下Spring集成iBATIS的最佳实践：

**1、 定义Dao接口，此处使用cn.javass.spring.chapter7.dao.IUserDao：**

**2、 定义Dao接口实现，此处是iBATIS实现：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package cn.javass.spring.chapter8.dao.ibatis;  //省略import  public class UserIbatisDaoImpl extends SqlMapClientDaoSupport      implements IUserDao {      @Override      public void save(UserModel model) {          getSqlMapClientTemplate().insert("UserSQL.insert", model);      }      @Override      public int countAll() {          return (Integer) getSqlMapClientTemplate().queryForObject("UserSQL.countAll");      }  } |

**3、修改iBATS映射文件（chapter8/sqlmaps/UserSQL.xml），添加countAll查询：**

java代码：

|  |  |
| --- | --- |
| 1  2  3 | <select id="countAll" resultClass="java.lang.Integer">      <![CDATA[ select count(\*) from test ]]>  </select> |

此处注意首先iBATIS实现放在dao.ibaitis包里，其次实现类命名如UserIbatisDaoImpl，即×××IbatisDaoImpl，当然如果自己有更好的命名规范可以遵循自己的，此处只是提个建议。

**4、进行资源配置，使用resources/chapter7/applicationContext-resources.xml：**

**5、dao定义配置，在chapter8/applicationContext-ibatis.xml中添加如下配置：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <bean id="abstractDao" abstract="true">      <property name="sqlMapClient" ref="sqlMapClient"/>  </bean>  <bean id="userDao"        class="cn.javass.spring.chapter8.dao.ibatis.UserIbatisDaoImpl"        parent="abstractDao"/> |

首先定义抽象的abstractDao，其有一个sqlMapClient属性，从而可以让继承的子类自动继承sqlMapClient属性注入；然后定义userDao，且继承abstractDao，从而继承sqlMapClient注入；我们在此给配置文件命名为applicationContext-ibatis.xml表示iBAITIS实现。

**5、 最后测试一下吧（cn.javass.spring.chapter8. IbatisTest）：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @Test  public void testBestPractice() {      String[] configLocations = new String[] {              "classpath:chapter7/applicationContext-resources.xml",              "classpath:chapter8/applicationContext-ibatis.xml"};      ApplicationContext ctx = new ClassPathXmlApplicationContext(configLocations);      IUserDao userDao = ctx.getBean(IUserDao.class);      UserModel model = new UserModel();      model.setMyName("test");      userDao.save(model);      Assert.assertEquals(1, userDao.countAll());  } |

和Spring JDBC框架的最佳实践完全一样，除了使用applicationContext-ibatis.xml代替了applicationContext-jdbc.xml，其他完全一样。也就是说，DAO层的实现替换可以透明化。

8.3.4Spring+iBATIS的CURD

Spring集成iBATIS进行CURD（增删改查），也非常简单，首先配置映射文件，然后调用SqlMapClientTemplate相应的函数进行操作即可，此处就不介绍了。

8.3.5集成MyBatis及最佳实践

(本笔记写于2010年底)

2010年4月份 iBATIS团队发布iBATIS 3.0的GA版本的候选版本，在iBATIS 3中引入了泛型、注解支持等，因此需要Java5+才能使用，但在2010年6月16日，iBATIS团队决定从apache迁出并迁移到Google Code，并更名为MyBatis。目前新网站上文档并不完善。

目前iBATIS 2.x和MyBatis 3不是100%兼容的，如配置文件的DTD变更，SqlMapClient直接由SqlSessionFactory代替了，包名也有com.ibatis变成org.ibatis等等。

ibatis 3.x和MyBatis是兼容的，只需要将DTD变更一下就可以了。

感兴趣的朋友可以到http://www.mybatis.org/官网去下载最新的文档学习，作者只使用过iBATIS2.3.4及以前版本，没在新项目使用过最新的iBATIS 3.x和Mybatis，因此如果读者需要在项目中使用最新的MyBatis，请先做好调研再使用。

接下来示例一下Spring集成MyBatis的最佳实践：

**1、准备需要的jar包，到MyBatis官网下载mybatis 3.0.4版本和mybatis-spring 1.0.0版本，并拷贝如下jar包到类路径：**

|  |
| --- |
| **mybatis-3.0.4\mybatis-3.0.4.jar              //核心MyBatis包**  **mybatis-spring-1.0.0\mybatis-spring-1.0.0.jar  //集成Spring包** |

**2、对象模型定义，此处使用第七章中的UserModel；**

**3、MyBatis映射定义（chapter8/sqlmaps/UserSQL-mybatis.xml）：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"       "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="UserSQL">      <sql id="createTable">      <!--id自增主键从0开始 -->      <![CDATA[        create memory table test(          id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,          name varchar(100))      ]]>      </sql>      <sql id="dropTable">      <![CDATA[ drop table test ]]>      </sql>      <insert id="insert" parameterType="cn.javass.spring.chapter7.UserModel">      <![CDATA[ insert into test(name) values (#{myName}) ]]>          <selectKey resultType="int" keyProperty="id" order="AFTER">              <!-- 获取hsqldb插入的主键 -->              call identity();              <!-- mysql使用select last\_insert\_id();获取插入的主键 -->          </selectKey>      </insert>      <select id="countAll" resultType="java.lang.Integer">      <![CDATA[ select count(\*) from test ]]>      </select>  </mapper> |

从映射定义中可以看出MyBatis与iBATIS2.3.4有如下不同：

* http://ibatis.apache.org/dtd/sql-map-2.dtd 废弃，而使用<http://mybatis.org/dtd/mybatis-3-mapper.dtd>。
* <sqlMap>废弃，而使用<mapper>标签；
* <statement>废弃了，而使用<sql>标签；
* parameterClass属性废弃，而使用parameterType属性；
* resultClass属性废弃，而使用resultType属性；
* #myName#方式指定命名参数废弃，而使用#{myName}方式。

**3、 MyBatis配置文件（chapter8/sql-map-config-mybatis.xml）定义：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"      "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>      <settings>          <setting name="cacheEnabled" value="false"/>      </settings>      <mappers>          <mapper resource="chapter8/sqlmaps/UserSQL-mybatis.xml"/>      </mappers>  </configuration> |

从配置定义中可以看出MyBatis与iBATIS2.3.4有如下不同：

* <http://ibatis.apache.org/dtd/sql-map-config-2.dtd>废弃，而使用<http://mybatis.org/dtd/mybatis-3-config.dtd>；
* < sqlMapConfig >废弃，而使用<configuration>；
* settings属性配置方式废弃，而改用子标签< setting name=”..” value=”..”/>方式指定属性，且一些属性被废弃，如maxTransactions；
* < sqlMap>废弃，而采用<mappers>标签及其子标签<mapper>定义。

**4、 定义Dao接口，此处使用cn.javass.spring.chapter7.dao. IUserDao：**

**5、 定义Dao接口实**现，**此处是MyBatis实现：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package cn.javass.spring.chapter8.dao.mybatis;  //省略import  public class UserMybatisDaoImpl extends SqlSessionDaoSupport  implements IUserDao {      @Override      public void save(UserModel model) {          getSqlSession().insert("UserSQL.insert", model);      }      @Override      public int countAll() {          return (Integer) getSqlSession().selectOne("UserSQL.countAll");      }  } |

和Ibatis集成方式不同的有如下地方：

* 使用SqlSessionDaoSupport来支持一致性的DAO访问，该类位于org.mybatis.spring.support包中，非Spring提供；
* 使用getSqlSession方法获取SqlSessionTemplate，在较早版本中是getSqlSessionTemplate方法名，不知为什么改成getSqlSession方法名，因此这个地方在使用时需要注意。
* SqlSessionTemplate是SqlSession接口的实现，并且自动享受Spring管理事务好处，因此从此处可以推断出为什么把获取模板类的方法名改为getSqlSession而不是getSqlSessionTemplate。

**6、进行资源配置，使用resources/chapter7/applicationContext-resources.xml：**

**7、dao定义配置，在chapter8/applicationContext-mybatis.xml中添加如下配置：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">    <property name="dataSource" ref="dataSource"/><!-- 1、指定数据源 -->    <property name="configLocation" value="chapter8/sql-map-config-mybatis.xml"/>  </bean>  <bean id="abstractDao" abstract="true">     <property name="sqlSessionFactory" ref="sqlSessionFactory"/>  </bean>  <bean id="userDao"             class="cn.javass.spring.chapter8.dao.mybatis.UserMybatisDaoImpl"             parent="abstractDao"/> |

和Ibatis集成方式不同的有如下地方：

* SqlMapClient类废弃，而使用SqlSessionFactory代替；
* 使用SqlSessionFactoryBean进行集成MyBatis。

首先定义抽象的abstractDao，其有一个sqlSessionFactory属性，从而可以让继承的子类自动继承sqlSessionFactory属性注入；然后定义userDao，且继承abstractDao，从而继承sqlSessionFactory注入；我们在此给配置文件命名为applicationContext-mybatis.xml表示MyBatis实现。

**8、最后测试一下吧（cn.javass.spring.chapter8. IbatisTest）：**

java代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @Test  public void testMybatisBestPractice() {      String[] configLocations = new String[] {          "classpath:chapter7/applicationContext-resources.xml",          "classpath:chapter8/applicationContext-mybatis.xml"};       ApplicationContext ctx = new ClassPathXmlApplicationContext(configLocations);      IUserDao userDao = ctx.getBean(IUserDao.class);      UserModel model = new UserModel();      model.setMyName("test");      userDao.save(model);      Assert.assertEquals(1, userDao.countAll());  } |

和Spring 集成Ibatis的最佳实践完全一样，除了使用applicationContext-mybatis.xml代替了applicationContext-ibatis.xml，其他完全一样，且MyBatis 3.x与Spring整合只能运行在Spring3.x。

在写本书时，MyBatis与Spring集成所定义的API不稳定，且期待Spring能在发布新版本时将加入对MyBatis的支持。