**从 iBatis 到 MyBatis，你准备好了吗？**

对于从事 Java EE 的开发人员来说，iBatis 是一个再熟悉不过的持久层框架了，在 Hibernate、JPA 这样的一站式对象 / 关系映射（O/R Mapping）解决方案盛行之前，iBaits 基本是持久层框架的不二选择。即使在持久层框架层出不穷的今天，iBatis 凭借着易学易用、轻巧灵活等特点，也仍然拥有一席之地。尤其对于擅长 SQL 的开发人员来说，iBatis 对 SQL 和存储过程的直接支持能够让他们在获得 iBatis 封装优势的同时而不丧失 SQL 调优的手段，这是 Hibernate/JPA 所无法比拟的。具体而言，使用 iBatis 框架的主要优势主要体现在如下几个方面：

首先，iBatis 封装了绝大多数的 JDBC 样板代码，使得开发者只需关注 SQL 本身，而不需要花费精力去处理例如注册驱动，创建 Connection，以及确保关闭 Connection 这样繁杂的代码。

其次，iBatis 可以算是在所有主流的持久层框架中学习成本最低，最容易上手和掌握的框架。虽说其他持久层框架也号称门槛低，容易上手，但是等到你真正使用时会发现，要想掌握并用好它是一件非常困难的事。在工作中我需要经常参与面试，我曾听到过很多位应聘者描述，他们所在的项目在技术选型时选择 Hibernate，后来发现难以驾驭，不得不将代码用 JDBC 或者 iBatis 改写。

iBatis 自从在 Apache 软件基金会网站上发布至今，和他的明星兄弟们（Http Server，Tomcat，Struts，Maven，Ant 等等）一起接受者万千 Java 开发者的敬仰。然而在今年六月中旬，几乎是发布 3.0 版本的同时，iBatis 主页上的一则 “Apache iBATIS has been retired” 的声明在社区引起了一阵不小的波澜。在 Apache 寄居六年之后，iBatis 将代码托管到 Google Code。在声明中给出的主要理由是，和 Apache 相比，Google Code 更有利于开发者的协同工作，也更能适应快速发布。于此同时，iBatis 更名为 MyBatis。

从 iBatis 到 MyBatis，不只是名称上的变化，MyBatis 提供了更为强大的功能，同时并没有损失其易用性，相反，在很多地方都借助于 JDK 的泛型和注解特性进行了简化。iBatis 确实该退休了，因为一个更为出色的继任者经过 10 个 Beta 版本的蜕变已然出现在我们的面前。

本文将主要针对 MyBatis 和 iBatis 的变化之处进行讨论，以便于读者顺利从 iBatis 向 MyBatis 过渡。

[**回页首**](http://www.ibm.com/developerworks/cn/opensource/os-cn-mybatis/#ibm-pcon)

**由一个 MyBatis 示例开始**

如果读者接触过一些常用的 Java EE 框架，应该都知道这些框架需要提供一个全局配置文件，用于指定程序正常运行所需的设置和参数信息。而针对常用的持久层框架而言（Hibernate、JPA、iBatis 等），则通常需要配置两类文件：一类用于指定数据源、事务属性以及其他一些参数配置信息（通常是一个独立的文件，可以称之为全局配置文件）；另一类则用于指定数据库表和程序之间的映射信息（可能不止一个文件，我们称之为映射文件）。MyBatis 也不例外，虽然其中的一部分可以通过注解的形式进行，但是这两部分内容本身仍是必不可少的。

根据 iBatis 的习惯，我们通常把全局配置文件命名为 sqlMapConfig.xml，文件名本身并没有要求，在 MyBatis 中，也经常会将该文件命名为 Configuration.xml （读完全文后读者也许会发现，在 iBatis 中经常出现的 “sqlMap” 在 MyBatis 中被逐渐淡化了，除了此处，还比如 iBatis 配置文件的根元素为 <sqlMapConfig>，指定映射文件的元素为 <sqlMap>，以及 SqlMapClient 等等，这个变化正说明，iBatis 仅是以 SQL 映射为核心的框架，而在 MyBatis 中多以 Mapper、Session、Configuration 等其他常用 ORM 框架中的名字代替，体现的无非是两个方面：首先是为了减少开发者在切换框架所带来的学习成本；其次，MyBatis 充分吸收了其他 ORM 框架好的实践，MyBatis 现在已不仅仅是一个 SQL 映射框架了）。在全局配置文件中可以配置的信息主要包括如下几个方面：

* properties --- 用于提供一系列的键值对组成的属性信息，该属性信息可以用于整个配置文件中。
* settings --- 用于设置 MyBatis 的运行时方式，比如是否启用延迟加载等。
* typeAliases --- 为 Java 类型指定别名，可以在 XML 文件中用别名取代 Java 类的全限定名。
* typeHandlers --- 在 MyBatis 通过 PreparedStatement 为占位符设置值，或者从 ResultSet 取出值时，特定类型的类型处理器会被执行。
* objectFactory --- MyBatis 通过 ObjectFactory 来创建结果对象。可以通过继承 DefaultObjectFactory 来实现自己的 ObjectFactory 类。
* plugins --- 用于配置一系列拦截器，用于拦截映射 SQL 语句的执行。可以通过实现 Interceptor 接口来实现自己的拦截器。
* environments --- 用于配置数据源信息，包括连接池、事务属性等。
* mappers --- 程序中所有用到的 SQL 映射文件都在这里列出，这些映射 SQL 都被 MyBatis 管理。

上面提及的大多数元素都不是必需的，通常 MyBatis 会为没有显式设置的元素提供缺省值。一个简单的全局配置文件示例如下：

**清单 1. 简单的全局配置文件示例**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--iBatis 和 MyBatis 的全局配置文件使用不同的 DTD 约束，在将应用由  iBatis 升级至 MyBatis 时需要注意（两者的映射文件 DTD 约束也不相同）-->  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 配置数据源相关的信息 -->  <environments default="demo">  <environment id="demo">  <transactionManager type="JDBC"/>  <dataSource type="POOLED">  <property name="driver" value= … />  <property name="url" value= … />  <property name="username" value="root"/>  <property name="password" value="root"/>  </dataSource>  </environment>  </environments>  <!-- 列出映射文件 -->  <mappers>  <mapper resource="footmark/mybatis/demo/UserInfoMapper.xml"/>  </mappers>  </configuration> |

有了这些信息，MyBatis 便能够和数据库建立连接，并应用给定的连接池信息和事务属性。MyBatis 封装了这些操作，最终暴露一个 SqlSessionFactory 实例供开发者使用，从名字可以看出来，这是一个创建 SqlSession 的工厂类，通过 SqlSession 实例，开发者能够直接进行业务逻辑的操作，而不需要重复编写 JDBC 相关的样板代码。根据全局配置文件生成 SqlSession 的代码如下：

|  |
| --- |
| Reader reader = Resources.getResourceAsReader("Configuration.xml");  SqlSessionFactory sqlSessionFactory =  new SqlSessionFactoryBuilder().build(reader);  SqlSession sqlSession = sqlSessionFactory.openSession(); |

可以把上面的三行代码看做是 MyBatis 创建 SqlSession 的样板代码。其中第一行代码在类路径上加载配置文件，Resources 是 MyBatis 提供的一个工具类，它用于简化资源文件的加载，它可以访问各种路径的文件，不过最常用的还是示例中这种基于类路径的表示方式。如果读者对 Hibernate 有所了解，一定会发现 MyBatis 不论是使用风格还是类名都和 Hibernate 非常相像，笔者曾今多次在国内外 Java 社区看到有人说 MyBatis 在向 Hibernate/JPA 靠拢。暂且不论这是否属实，持久化技术在经过一番蓬勃的竞争和发展，最终在社区形成统一的认识并被广泛接受，这对开发者而言未必不是一件好事，MyBatis 在这一点上只是向事实上的标准靠近了一步。

在完成全局配置文件，并通过 MyBatis 获得 SqlSession 对象之后，便可以执行数据访问操作了。对于 iBatis/MyBatis 而言，要执行的操作其实就是在映射文件中配置的 SQL 语句。两者的配置基本相同，如下所示：

**清单 2. 在映射文件中配置 SQL 语句**

|  |
| --- |
| <?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="mybatis.demo.UserInfoMapper">  <select id="selectUser" parameterType="int"  resultType="mybatis.demo.UserInfo">  select \* from UserInfo where userid =#{userid}  </select>  </mapper> |

在 iBatis 中，namespace 不是必需的，且它的存在没有实际的意义。在 MyBatis 中，namespace 终于派上用场了，它使得映射文件与接口绑定变得非常自然。关于接口绑定，后面会有篇幅专门描述。使用 SqlSession 执行 SQL 的方式如下：

**清单 3. 使用 SqlSession 执行映射文件中配置的 SQL 语句**

|  |
| --- |
| try  {  UserInfo userinfo = (UserInfo) sqlSession.selectOne  ("mybatis.demo.UserInfoMapper.getUser", 2);  System.out.println(userinfo);  } finally  {  sqlSession.close();  } |

需要注意的是，SqlSession 的使用必需遵守上面的格式，即在 finally 块中将其关闭。以保证资源得到释放，防止出现内存泄露！

以上就是一个简单而完整的 MyBatis 程序。其中涉及了全局配置文件，映射文件，构建 SqlSession 对象，执行数据访问操作等四个步骤。下面将针对除构建 SqlSession 对象之外的三块内容进行分解。

[**回页首**](http://www.ibm.com/developerworks/cn/opensource/os-cn-mybatis/#ibm-pcon)

**MyBatis 全局配置文件的改变**

MyBatis 全局配置文件的各主要元素基本和 iBatis 相同，只是在用法和个别名称上做了调整。元素的意义就不再描述，下面主要讲述针对 iBatis 和 MyBatis 配置文件的主要区别之处。

首先，两个版本的 DTD 约束不同，MyBatis 的 DTD 文件已经包含在发布包下的 mybatis-3.0.x.jar 包中。这直接影响到的是，iBatis 配置文件的根元素是 <sqlMapConfig>，而 MyBatis 使用的是 <configuration>。

其次，<settings> 的用法发生了改变，之前的格式为：

**清单 4. 在 iBatis 中设置属性的方式**

|  |
| --- |
| <settings props1="value1" props2="value2"… /> |

要设置的属性直接以键值对的形式作为 <settings> 的属性。而在 MyBatis 中调整为略显复杂但却更有条理的方式：

**清单 5. 在 MyBatis 中设置属性的方式**

|  |
| --- |
| <settings>  <setting name="props1" value="value1"/>  <setting name="props2" value="value2"/>  ……  </settings> |

另外，之前配置事务管理器和数据源的方式如下：

**清单 6. 在 iBatis 中配置事务管理器和数据源的方式**

|  |
| --- |
| <transactionManager type="JDBC" >  <dataSource type="SIMPLE">  <property name="JDBC.Driver" value="${driver}"/>  <!-- 其他数据源信息省略 -->  </dataSource>  </transactionManager> |

在 MyBatis 中调整为如下的方式：

**清单 7. 在 MyBatis 中配置事务管理器和数据源的方式**

|  |
| --- |
| <environments default="demo">  <environment id="demo">  <transactionManager type="JDBC"/>  <dataSource type="POOLED">  <property name="JDBC.Driver" value="${driver}"/>  <!-- 其他数据源信息省略 -->  </dataSource>  </environment>  </environments> |

通过 <environments> 来进行数据源管理，主要是为了简化在多套数据源配置之间的切换，比如开发和发布使用不同的配置。

最后，在 iBatis 中指定映射文件的方式如下：

**清单 8. 在 iBatis 中指定映射文件的方式**

|  |
| --- |
| <sqlMap resource=... />  <sqlMap resource=... />  <sqlMap resource=... /> |

在 MyBatis 中调整为如下方式：

**清单 9. 在 MyBatis 中指定映射文件的方式**

|  |
| --- |
| <mappers>  <mapper resource=... />  <mapper resource=... />  </mappers> |

上面的这些调整，主要出发点其实并不是使得 MyBatis 功能更为强大，而是使配置更为合理，让开发者更容易阅读和理解。

到目前为止，我们主要讨论了 XML 形式的全局配置，其实这也不是唯一选择，MyBatis 还提供了通过代码来进行配置的方式：

**清单 10. 在 MyBatis 中使用代码进行配置**

|  |
| --- |
| DataSource ds = …… // 获取一个 DataSource  TransactionFactory txFactory = new JdbcTransactionFactory();  Environment env = new Environment("demo", txFactory, ds);  Configuration cfg = new Configuration(env);  cfg.addMapper(UserInfoMapper.class);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(cfg); |

结合前面的配置文件，很容易理解这段代码的意思，故不再赘述。不过，需要注意的是 Configuration 的 addMapper() 方法，该方法的参数通常是一个接口，可以在接口里面定义若干方法，在方法上使用注解来指定映射的 SQL 语句。一个典型的接口定义以及对应的数据访问方法如下：

**清单 11. 将映射的 SQL 语句与接口中的方法绑定**

|  |
| --- |
| // 映射 SQL 绑定接口  public interface UserInfoMapper  {  @Select("select \* from userinfo where userid = #{userid}")  public UserInfo getUserInfo(int userid);  }  // 接口绑定对应的数据访问方法  try  {  //UserInfo userinfo = (UserInfo) sqlSession.selectOne  ("mybatis.demo.UserInfoMapper.selectUser", 2);  UserInfoMapper userinfoMapper =  sqlSession.getMapper(UserInfoMapper.class);  UserInfo userinfo = userinfoMapper.getUserInfo(1);  System.out.println(userinfo);  } finally  {  sqlSession.close();  } |

[**回页首**](http://www.ibm.com/developerworks/cn/opensource/os-cn-mybatis/#ibm-pcon)

**MyBatis 映射文件的改变**

MyBatis 针对映射文件进行格式调整的地方很多，但大部分仅仅只是名称上的变化，现代的 IDE 都支持联想功能，可以很方便的获取到当前位置可以有哪些元素、哪些属性等。所以这基本不会给开发者造成什么麻烦。

针对映射文件，首先是一系列的属性名称的改变，这些仅仅是名称的改变，用法和含义并没有发生变化：

* 和全局配置文件一样，由于 DTD 约束发生变化，根元素也由原来的 <sqlMap> 调整为 <mapper>。
* <select> 等元素的 parameterClass 属性改为了 parameterType 属性。
* <select> 等元素的 resultClasss 属性改为了 resultType 属性。
* <parameterMap> 等元素的 class 属性改为了 type 属性。
* <result> 元素的 columnIndex 属性被移除了。
* 嵌套参数由 #value# 改为了 #{value}。
* <parameter> 等元素的 jdbcType 属性取值中，原来的 "ORACLECURSOR" 取值改为了现在的 "CURSOR"，"NUMBER" 取值改为了 "NUMERIC"。

iBatis/MyBatis 对存储过程的支持一直是值得称道的。之前通过使用 <procedure> 元素进行存储过程的定义，示例如下：

**清单 12. iBatis 中调用存储过程的方式**

|  |
| --- |
| <procedure id="getValues" parameterMap="getValuesPM">  { ? = call pkgExample.getValues(p\_id => ?) }  </procedure> |

在 MyBatis 中，<proccedure> 元素已经被移除，通过 <select>、<insert> 和 <update> 进行定义：

**清单 13. MyBatis 中调用存储过程的方式**

|  |
| --- |
| <select id="getValues" parameterMap="getValuesPM" statementType="CALLABLE">  { ? = call pkgExample.getValues(p\_id => ?)}  </select> |

如上所示，通过 statementType 属性将该语句标识为存储过程而非普通 SQL 语句。

[**回页首**](http://www.ibm.com/developerworks/cn/opensource/os-cn-mybatis/#ibm-pcon)

**代码层面的改变**

通过前面的示例可以看出，MyBatis 在编码中的最大的改变就是将一个最常用的 API 由 SqlMapClient 改为了 SqlSessionFactory。另外，类型处理器接口也由原来的 TypeHandlerCallback 改为了 TypeHandler。最后 DataSourceFactory 也进行了调整，移动到 org.apache.ibatis.datasource 包下，其中的方法也作了微调。总之，代码层面公开的部分改动较少，不会给开发者造成较大的移植成本。

[**回页首**](http://www.ibm.com/developerworks/cn/opensource/os-cn-mybatis/#ibm-pcon)

**总结**

本文主要描述了从 iBatis 向 MyBatis 移植过程中可能遇到的问题，大部分的变化已经体现在上文中，如果希望从头开始学习 MyBatis，则建议从头开始阅读官方的 user guide 文档。