第5章 MyBatis基础知识

CHAPTER 5

本章目标

1. 了解ORM的概念及与MyBatis的关系。
2. 掌握MyBatis中常用的API的使用方法。
3. 正确编写MyBatis的配置文件。
4. 正确编写MyBatis中的映射文件。

MyBatis本是Apache的一个开源项目iBatis，2010年这个项目由Apache Software Foundation迁移到Google Code，并且改名为MyBatis，2013年11月迁移到GitHub。

MyBatis是一款优秀的持久层框架，它支持定制化SQL、存储过程以及高级映射。MyBatis将JDBC进行了封装，避免了直接使用JDBC操作数据库。MyBatis可以使用简单的XML或注解配置映射类和表之间的关系，将接口和Java的POJO(Plain Ordinary Java Object，普通的Java对象)映射成数据库中的记录。

5.1 ORM与MyBatis

MyBatis框架是ORM的一种实现。所谓的ORM(Object-Relational Mapping，对象-关系映射)就是为了解决面向对象程序设计与数据库之间不匹配的技术，它通过描述Java对象与数据库表之间的映射关系，自动将Java应用程序中的对象持久化到关系数据库的表中。

对象-关系映射是随着面向对象的软件开发方法发展产生的。面向对象的开发方法是当今企业级应用开发环境中的主流开发方法，关系数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和关系是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据无法直接表达多对多关联和继承关系。因此，对象-关系映射系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

当开发一个应用程序的时候，如果不使用 ORM，则可能会写不少数据访问层的代码，用来从数据库保存、删除、读取对象信息等。通常在数据库编程中写很多方法以完成读取对象数据、改变状态对象等任务，这些代码不可避免会有重复。

ORM 解决的主要问题是对象关系的映射。域模型和关系模型都建立在概念模型基础上。域模型是面向对象的，而关系模型是面向关系的。一般情况下，一个持久化类和一个表对应，类的每个实例对应表中的一条记录，类的每个属性对应表的每个字段。

ORM 技术的特点如下：

（1）提高了开发效率。由于 ORM 可以自动对实体对象与数据库中的表进行字段与属性的映射，所以我们已经不需要一个专用的、庞大的数据访问层。

（2）ORM 提供了对数据库的映射，不用 SQL 直接编码，就能像操作对象一样从数据库获取数据。

从系统结构上看，采用 ORM 的系统一般都是多层系统，系统的层次多了，效率就会降低。ORM 是一种完全的面向对象的做法，而面向对象的做法也会对性能产生一定的影响。

当前的 ORM 框架产品有很多，常见的主要有 Hibernate 和 MyBatis 两个框架，这两个框架的对比见表 5-1。

表 5-1 Hibernate 框架和 MyBatis 框架的对比

对比项目	Hibernate 框架	MyBatis 框架
自动化程度	Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，建立对象与数据库表的映射，是一个全自动的、完全面向对象的持久层框架	MyBatis 是一个开源对象关系映射框架，是一个半自动化的持久层框架
性能	Hibernate 自动生成 SQL，有些语句较为烦琐，会多降低性能	MyBatis 手动编写 SQL，可以避免不需要的查询，提高系统性能
对象管理	Hibernate 是完整的对象-关系映射的框架，在开发工程中无须过多关注底层实现，只要管理对象即可	MyBatis 需要自行管理对象与表的映射关系
缓存管理	Hibernate 的二级缓存配置在 SessionFactory 生成的配置文件中进行详细配置，然后再在具体的表-对象映射中配置是哪种缓存	MyBatis 的二级缓存配置都是在每个具体的表-对象映射中进行详细配置，这样针对不同的表可以自定义不同的缓存机制
学习难易	学习门槛较高	容易学习和掌握

5.2 MyBatis 的开发环境

5.2.1 MyBatis 框架的 JAR 包下载

想在 Web 开发项目中添加 MyBatis 的支持非常简单，只将 MyBatis 的 JAR 包加入构建路径即可。编写本书时，MyBatis 的最新版本是 mybatis-3.5.2，本书中的所有

MyBatis 案例都基于这个版本。MyBatis 的 JAR 包下载地址为 https://github.com/mybatis/mybatis-3/releases。单击网址，可打开如图 5-1 所示的页面。

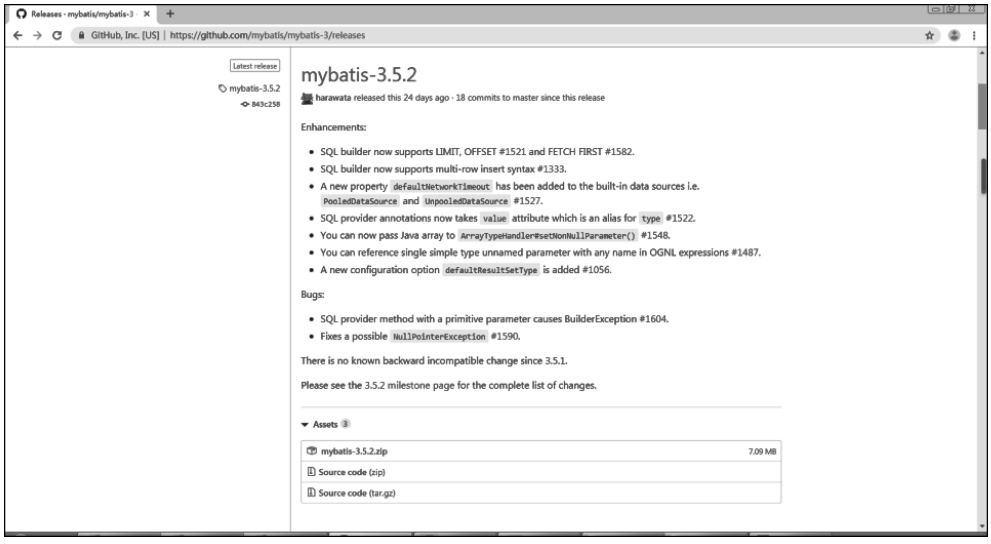

图 5-1　MyBatis 框架 JAR 包下载页面

在网页中单击 mybatis-3.5.2.zip 超链接即可下载 JAR 包。下载后将其中的 mybatis-3.5.2.jar 文件复制到工程的 lib 目录中即可，同时不要忘记添加 MySQL 的数据库驱动程序。此处，我们添加的驱动程序是 mysql-connector-java-5.1.40-bin.jar。除了上面的网站还有一个网站，对开发者会有帮助，网址为 http://www.mybatis.org/mybatis-3/zh/getting-started.html。

这个网站为开发者提供了最新的 MyBatis 相关技术内容，同时提供了 MyBatis 框架所需的一些基本知识，网站提供了 5 种语言，包括简体中文，非常便于查阅。MyBatis 中文帮助网站如图 5-2 所示。

5.2.2　日志信息配置

使用 MyBatis 的时候，有时需要查看 SQL 语句执行的详细信息，为此 MyBatis 提供了非常简单有效的方法——日志。

MyBatis 的内置日志工厂提供日志功能。内置日志工厂将日志交给以下其中一种工具作代理：

- SLF4J；
- Apache Commons Logging；
- Log4j 2；
- Log4j；
- JDK logging。

MyBatis 内置日志工厂基于运行时自省机制选择合适的日志工具。它会使用第一个查找得到的工具（按上文列举的顺序查找）。如果一个都未找到，日志功能就会被

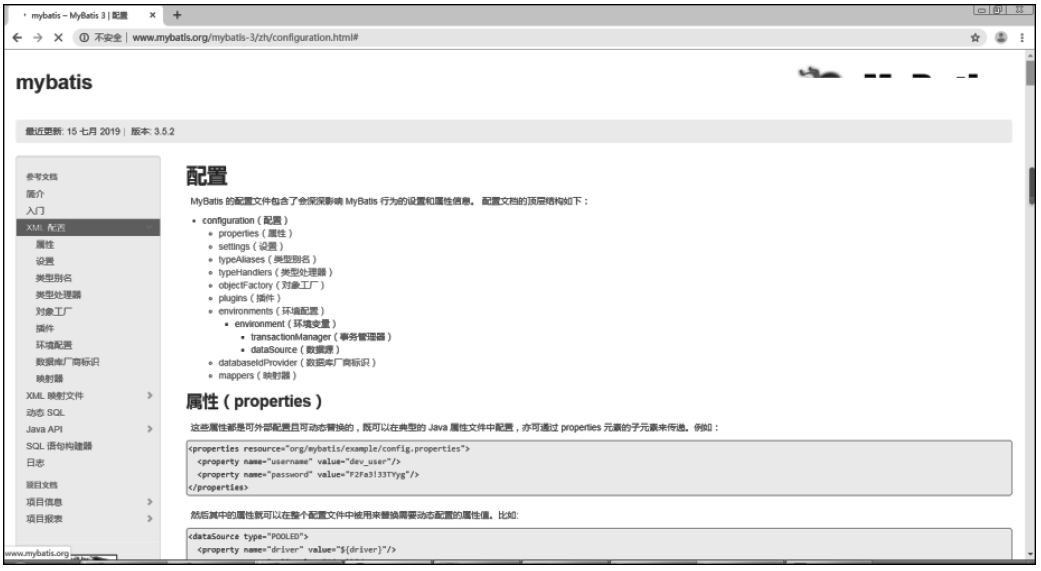

图 5-2　MyBatis 中文帮助网站

禁用。

不少应用服务器（如 Tomcat 和 WebShpere）的类路径中已经包含 Commons Logging，所以在这种配置环境下的 MyBatis 会把它作为日志工具。这将意味着，在诸如 WebSphere 的环境中，它提供了 Commons Logging 的私有实现，你的 Log4J 配置将被忽略。如果你的应用部署在一个类路径已经包含 Commons Logging 的环境中，而你又想使用其他日志工具，可以通过在 MyBatis 配置文件 mybatis-config.xml 里添加一项 setting 选择别的日志工具，如下面的代码所示。

```
<configuration>
  <settings>
    …
    <setting name="logImpl" value="LOG4J"/>
    …
    </settings>
</configuration>
```

logImpl 可选的值有 SLF4J、LOG4J、LOG4J2、JDK_LOGGING、COMMONS_LOGGING、STDOUT_LOGGING、NO_LOGGING，或者是实现了接口 org.apache.ibatis.logging.Log 且构造方法是以字符串为参数的类的完全限定名（可以参考 org.apache.ibatis.logging.slf4j.Slf4jImpl.java 的实现）。

下面的例子将使用 Log4J 配置完整的日志服务，共两个步骤。

步骤 1：添加 Log4J 的 JAR 包。

因为我们使用的是 Log4J，就要确保它的 JAR 包在应用中是可用的。要启用 Log4J，只要将 JAR 包添加到应用的类路径中即可。对于 Web 应用或企业级应用，需要

将 log4j.jar 添加到 WEB-INF/lib 目录下；对于独立应用，可以将它添加到 JVM 的 -classpath 启动参数中。

步骤 2：配置 Log4J。

配置 Log4J 比较简单，假如需要记录下面这个映射器接口的日志：

```
Package com.mapper;
public interface CustomerMapper {
    @Select("SELECT * FROM customer WHERE id = #{id}")
    Customer selectCustomer(int id);
}
```

在应用的类路径中创建一个名称为 log4j.properties 的文件，文件的具体内容如下：

```
#Global logging configuration
log4j.rootLogger=ERROR, stdout
#MyBatis logging configuration...
log4j.logger.com.mapper.CustomerMapper=TRACE
#Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [% t] -% m% n
```

添加以上配置后，Log4J 就会记录 com.mapper.Customer 的详细执行操作，且仅记录应用中其他类的错误信息。

也可以将日志的记录方式从接口级别切换到语句级别，从而实现更细粒度的控制。下面的配置只对 selectCustomer 语句记录日志：

```
log4j.logger. com.mapper.CustomerMapper.selectCustomer=TRACE
```

与此相对，可以对一组映射器接口记录日志，只要对映射器接口所在的包开启日志功能即可。

```
log4j.logger.com.mapper=TRACE
```

某些查询可能会返回庞大的结果，此时只想记录其执行的 SQL 语句，而不想记录结果该怎么办？MyBatis 中 SQL 语句的日志级别被设为 DEBUG(JDK 日志设为 FINE)，结果的日志级别为 TRACE(JDK 日志设为 FINER)。所以，只要将日志级别调整为 DEBUG 即可达到目的：

```
log4j.logger.com.mapper=DEBUG
```

要记录日志的是类似下面的映射器文件，而不是映射器接口又该怎么做呢？

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <!DOCTYPE mapper
3.      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
```

```
4.      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5.  <mapper namespace="com.mapper.ClazzMapper">
6.    <select id="selectClazzById" resultType="Clazz">
7.      select * from tb_clazz where id =#{id}
8.    </select>
9.  </mapper>
```

如需对 XML 文件记录日志，只对命名空间增加日志记录功能即可。

```
log4j.logger.com.mapper.ClazzMapper =TRACE
```

要记录具体语句的日志可以这样做：

```
log4j.logger.com.mapper.ClazzMapper.selectClazzById=TRACE
```

从上面的配置可以看到，为映射器接口和 XML 文件添加日志功能的语句没有差别。

5.3 MyBatis 中的 API

使用 MyBatis 的主要 Java 接口就是 SqlSession。可以通过这个接口执行命令，获取映射器和管理事务。我们会概括讨论一下 SqlSession 本身，但是首先还是要了解如何获取一个 SqlSession 实例。SqlSessions 是由 SqlSessionFactory 实例创建的。SqlSessionFactory 对象包含创建 SqlSession 实例的所有方法。而 SqlSessionFactory 本身是由 SqlSessionFactoryBuilder 创建的，它可以从 XML、注解或手动配置 Java 代码创建 SqlSessionFactory。

5.3.1 SqlSessionFactoryBuilder

SqlSessionFactoryBuilder 有 9 个 build() 方法，每种都允许从不同的资源中创建一个 SqlSession 实例。SqlSessionFactoryBuilder 的部分源码如下。

```
1.  public class SqlSessionFactoryBuilder {
2.
3.      public SqlSessionFactory build(InputStream inputStream) {
4.          return build(inputStream, null, null);
5.      }
6.      public SqlSessionFactory build(Configuration config) {
7.         return new DefaultSqlSessionFactory(config);
8.      }
9.    public SqlSessionFactory build(InputStream inputStream, String
      environment, Properties properties) {
10.  try {
11.       XMLConfigBuilder parser = new XMLConfigBuilder(inputStream,
          environment, properties);
```

```
12.     return build(parser.parse());
13.     } catch (Exception e) {
14.       throw ExceptionFactory.wrapException("Error building SqlSession.", e);
15.       } finally {
16.          ErrorContext.instance().reset();
17.          try {
18.                inputStream.close();
19.          } catch (IOException e) {
20.           //Intentionally ignore. Prefer previous error.
21.          }
22.       }
23.     }
24.   }
```

在上面的代码中，XMLConfigBuilder 是一个解析 XML 的工具类，如果调用有参数 environment 的 build() 方法，MyBatis 将会使用 configuration 对象配置这个 environment。当然，如果指定了一个不合法的 environment，就会得到错误提示。如果调用了不带 environment 参数的 build() 方法，就使用默认的 environment（在上面的示例中指定为 default="development" 的代码）。

如果调用了有参数 properties 实例的方法，MyBatis 就会加载那些 properties（属性配置文件）。那些属性可以用 ${propName} 语法形式多次用在配置文件中。

5.3.2　SqlSessionFactory

这里给出的是 3 个比较常用的生成 SqlSessionFactory 的 build()方法，下面的代码演示了第 3 行的 build()方法生成 SqlSessionFactory 示例的应用。

```
1.  String resource="mybatis-config.xml";
2.  InputStream inputStream=Resources.getResourceAsStream(resource);
3.  SqlSessionFactoryBuilder builder=new SqlSessionFactoryBuilder();
4.  SqlSessionFactory factory=builder.build(inputStream);
```

以上代码是读取 mybatis-config.xml 配置文件构建一个 InputStream 输入流，然后调用 SqlSessionFactoryBuilder 的 build()方法创建一个 SqlSessionFactory 工厂类。典型的 mybatis-config.xml 配置文件如下所示。

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
3.     "http://mybatis.org/dtd/mybatis-3-config.dtd">
4.  <configuration>
5.    <!--1.配置环境 ,默认的环境 id 为 mysql-->
6.    <environments default="mysql">
7.      <!--1.2.配置 id 为 mysql 的数据库环境 -->
8.      <environment id="mysql">
```

```
9.          <!--使用 JDBC 的事务管理 -->
10.         <transactionManager type="JDBC" />
11.         <!--数据库连接池 -->
12.         <dataSource type="POOLED">
13.             <property name="driver" value="com.mysql.jdbc.Driver" />
14.             <property name ="url"
15.                             value="jdbc:mysql://localhost:3306/mybatis" />
16.             <property name="username" value="root" />
17.             <property name="password" value="root" />
18.         </dataSource>
19.       </environment>
20.     </environments>
21.     <!--2.配置 mapper 的位置 -->
22.     <mappers>
23.         <mapper resource="com/mapper/CustomerMapper.xml" />
24.     </mappers>
25.  </configuration>
```

配置文件中的<configuration>标签描述了数据库连接的信息，<mappers>标签描述了映射文件的位置信息。SqlSessionFactory 是一个接口，它有两个实现类，分别是 SqlSessionFactoryManager 和 DefaultSqlSessionFactory。前面讲到的几个类之间的关系可用如图 5-3 所示的 UML 类图表示。

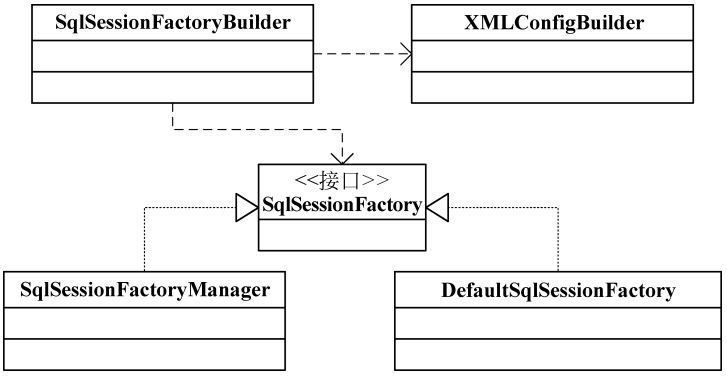

图 5-3　MyBatis 中的主要类和接口之间的关系

在基于 MyBatis 框架的应用程序中，都是以 SqlSessionFactory 为中心，它的主要作用是生产 MyBatis 的核心接口对象 SqlSession。

5.3.3　SqlSession

SqlSession 是 MyBatis 的关键对象，是执行持久化操作的对象，类似于 JDBC 中的 Connection。它是应用程序与持久层之间执行交互操作的一个单线程对象，也是 MyBatis 执行持久化操作的关键对象。SqlSession 对象完全包含对数据库所有执行 SQL

操作的方法，对 JDBC 操作数据库进行了封装，可以用 SqlSession 实例直接执行被映射的 SQL 语句。每个线程都应该有它自己的 SqlSession 实例，SqlSession 的实例不能被共享，同时也是线程不安全的。绝不能将 SqlSession 实例的引用放在一个类的静态字段，甚至是实例字段中。也绝不能将 SqlSession 实例的引用放在任何类型的管理范围中，如放在 Servlet 中的 HttpSession 对象中。使用完 SqlSession 后必须关闭，应该确保使用 finally 块来关闭它。

SqlSession 类中有 20 多个方法，将它们按功能进行分组，便于理解。

1. 对数据库进行增、删、改、查的方法

- <T>T selectOne(String statement, Object parameter)

查询方法，查询结果是一个泛型对象或为 null。

- <E>List<E>selectList(String statement, Object parameter)

查询方法，查询结果可以是 0 到多个泛型对象的集合，也可以为 null。

- <K, V>Map<K, V>selectMap(String statement, Object parameter, String mapKey)

查询方法，查询结果是一个 Map 集合。

- int insert(String statement, Object parameter)

更新方法，执行插入该表中记录的操作。

- int update(String statement, Object parameter)

更新方法，执行更新该表中记录的操作。

- int delete(String statement, Object parameter)

更新方法，执行删除该表中记录的操作。

- seqSession.selectCursor(String statement,Object parameter);

方法中的参数 statement 是映射文件中对应的元素的 id，参数 parameter 是执行 SQL 语句所需绑定的动态参数。

selectOne 和 selectList 的区别是，selectOne 必须返回一个对象或 null 值。如果返回值多于一个，就会抛出异常。如果不能确定返回对象的数量，请使用 selectList。如果需要查看返回对象是否存在，一个好的办法是返回一个值即可(0 或 1)。selectMap 稍微特殊一点，因为它会将返回的对象的其中一个属性作为 key 值，将对象作为 value 值，从而将多结果集转为 Map 类型值。

与上面 7 个方法对应的还有 7 个重载的方法，除了 selectMap()方法有两个参数外，其他的都只有一个参数 statement。

- <T>T selectOne(String statement);
- <E>List<E>selectList(String statement);
- <T>Cursor<T>selectCursor(String statement);
- <K,V>Map<K,V>selectMap(String statement, String mapKey);
- int insert(String statement);
- int update(String statement);

- int delete(String statement)。

这 7 个方法中没有动态绑定参数 parameter，所以一般是将参数写在映射文件中，这种应用场景不是很多。

如果查询结果是一个集合，要对返回结果的数量进行控制，可以使用以下的重载方法。

- <E> List<E> selectList（String statement，Object parameter，RowBounds rowBounds）
- <K，V> Map<K，V> selectMap（String statement，Object parameter，String mapKey，RowBounds rowbounds）

这两个方法是前面 selectList 和 selectMap 两个方法的重载，其中的 rowbounds 对象参数可以设置返回结果集合的起始位置和数量。

2. 事务控制方法

控制事务作用域有 3 个方法。如果已经设置了自动提交或正在使用外部事务管理器，这 3 个方法不会起作用。然而，如果正在使用 JDBC 事务管理器，由 Connection 实例控制，那么这 3 个方法就会派上用场。

- void commit()：事务提交。
- void rollback()：回滚事务。
- void close()：关闭 SqlSession 对象。

3. 缓存管理方法

MyBatis 使用到两种缓存：本地缓存(local cache)和二级缓存(second level cache)。

- void clearCache()：清空本地缓存。

4. 获取 Mapper 接口

上述 SqlSession 的 insert、update、delete 和 select 方法都很强大，但也有些烦琐，可能会产生类型安全问题，并且对于 IDE 和单元测试也很不方便。因此，一个更通用的方式是通过使用映射器类执行映射语句。一个映射器类就是一个仅需声明与 SqlSession 方法相匹配的方法的接口类。

<T>T getMapper(Class<T>type)：获取映射器类接口的方法。

Mapper 接口的使用后面会详细介绍。

5.4 MyBatis 的配置文件

MyBatis 的配置文件是 MyBatis 框架的必备文件，包含了影响 MyBatis 行为的设置和属性信息。配置文档的顶层结构如下。

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
```