本章学习目标

- MyBatis 介绍
- MyBatis 和 JDBC 及 Hibernate 的区别
- MyBatis 入门案例
- MyBatis 框架核心架构
- MyBatis 开发 CRUD 案例
- 核心配置 sqlMapConfig 说明
- properties 属性的作用
- 别名的使用
- mapper 映射器的使用

1. MyBatis 介绍

MyBatis 本是 apache 的一个开源项目 iBatis，2010 年这个项目由 Apache Software Foundation 迁移到了 google code，并且改名为 MyBatis 。2013 年 11 月迁移到 Github。 MyBatis 是一个优秀的持久层框架，它对 JDBC 的操作数据库的过程进行封装，使开发者只需要关注 SQL 本身，而不需要花费精力去处理例如注册驱动、创建 Connection、创建 Statement、手动设置参数、结果集检索封装等繁杂的过程代码。

Mybatis 通过 xml 或注解的方式将要执行的各种 Statement（Statement、PreparedStatemnt，CallableStatement）配置起来，并通过 Java 对象和 Statement 中的 sql 进行映射生成最终执行的 sql 语句，最后由 Mybatis 框架执行 sql 并将结果映射成 java 对象并返回。

2. MyBatis 和 JDBC 及 Hibernate 的区别

2.1. 使用 JDBC 的缺点

1、数据库连接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用

数据库链接池可解决此问题。

2、Sql 语句在代码中硬编码，造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变 java 代码。

3、使用 PreparedStatement 向占有位符号传参数存在硬编码，因为 sql 语句的 where 条件不一定，可能多也可能少，修改 sql 还要修改代码，系统不易维护。

4、对结果集解析存在硬编码（查询列名），sql 变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成 pojo 对象解析比较方便。

2.2. MyBatis 和 Hibernate 的区别

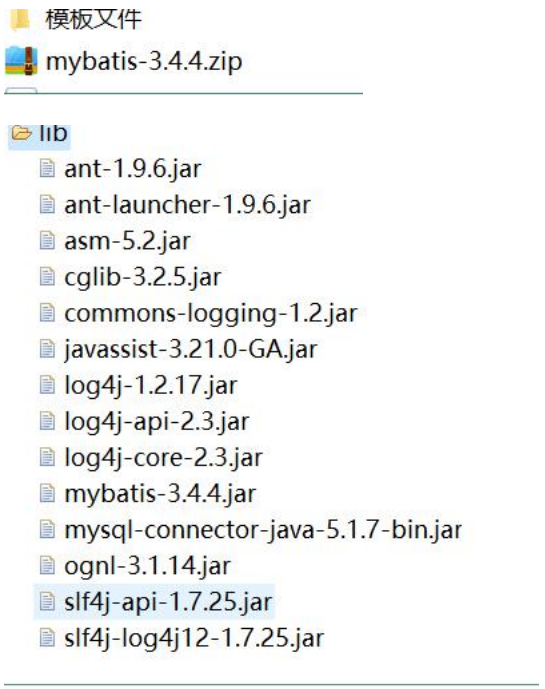
Mybatis 和 hibernate 不同，它不完全是一个 ORM 框架，因为 MyBatis 需要程序员自己编写 Sql 语句，不过 mybatis 可以通过 XML 或注解方式灵活配置要运行的 sql 语句，并将 java 对象和 sql 语句映射生成最终执行的 sql，最后将 sql 执行的结果再映射生成 java 对象。

Mybatis 学习门槛低，简单易学，程序员直接编写原生态 sql，可严格控制 sql 执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是 mybatis 无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套 sql 映射文件，工作量大。

Hibernate 对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用 hibernate 开发可以节省很多代码，提高效率。但是 Hibernate 的学习门槛高，要精通门槛更高，而且怎么设计 O/R 映射，在性能和对象模型之间如何权衡，以及怎样用好 Hibernate 需要具有很强的经验和能力才行。

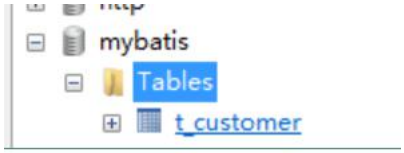
3. MyBatis 入门案例

3.1. 导入 mybatis 必须的 jar 包



3.2. 创建数据库和表

```
-- 创建客户表
CREATE TABLE t_customer(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20),
    gender CHAR(1),
    telephone VARCHAR(20)
);
```



3.3. 编写实体类

```
public class Customer {


    private Integer id;
    private String name;
    private String gender;
    private String telephone;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    public String getTelephone() {
        return telephone;
    }
```

```
    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }

}
```

3.4. 配置全局 sqlMapConfig.xml

建议放在 classpath 下

这个文件是 mybatis 全局的配置（例如连接池，连接参数，sql 映射文件信息）

```
<?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>
    <!-- 数据库环境 -->
    <environments default="develop">
        <environment id="develop">
            <!-- jDBC 事务管理 -->
            <transactionManager type="JDBC"/>
            <!-- 连接池配置 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
```

```
    </environments>


    <!-- sql 映射文件 -->
    <mappers>
        <mapper resource="Customer.xml"/>
    </mappers>


</configuration>
```

3.5. 配置 Customer.xml（SQL 文件）

```
<?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">
<!-- 该文件存放 CRUD 的 sql 语句 -->
<mapper namespace="test">
    <!-- 添加 -->
    <insert id="insertCustomer"
parameterType="cn.sm1234.domain.Customer">
        INSERT INTO t_customer(NAME,gender,telephone)
VALUES(#{name},#{gender},#{telephone})
    </insert>
</mapper>
```

3.6. 编写测试代码

```
/**
```

```
 * 演示 MyBatis 的入门程序
 * @author lenovo
 *
 */
public class Demo1 {

    public static void main(String[] args) throws Exception {
        //1.创建 SqlSessionFactoryBuilder
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

        //2.创建 SqlSessionFactory
        InputStream inputStream =
Resources.getResourceAsStream("sqlMapConfig.xml");
        SqlSessionFactory factory = builder.build(inputStream);

        //3.创建 SqlSession  （SqlSession 可用于执行 CRUD 操作）
        SqlSession sqlSession = factory.openSession();

        //4.执行操作
        Customer customer = new Customer();
        customer.setName("张三");
        customer.setGender("男");
        customer.setTelephone("13455556666");

        sqlSession.insert("insertCustomer",customer);

        //5.如果是更新操作，则需要提交事务
        sqlSession.commit();
```

```
        //6.关闭连接
        sqlSession.close();
    }
}
```

这时发现执行成功后，控制没有日志的，可以在 classpath 下放 log4j.properties 文件：

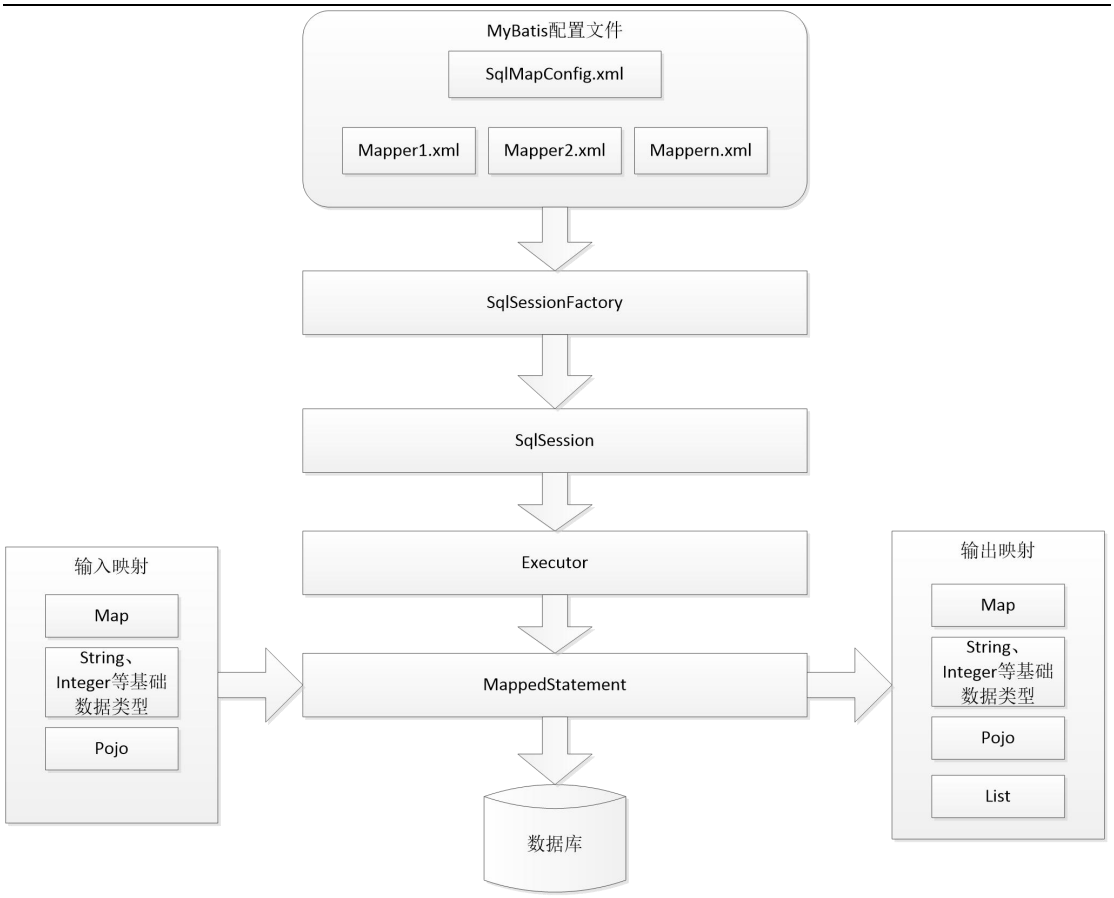
```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.err
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L
- %m%n

### direct messages to file mylog.log ###
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=c\:mylog.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L
- %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=debug, stdout
```

4. MyBatis 框架核心架构



1、mybatis 配置

SqlMapConfig.xml，此文件作为 mybatis 的全局配置文件，配置了 mybatis 的运行环境等信息。

mapper.xml 文件即 sql 映射文件，文件中配置了操作数据库的 sql 语句。此文件需要在 SqlMapConfig.xml 中加载。

2、通过 mybatis 环境等配置信息构造 SqlSessionFactory 即会话工厂

3、由会话工厂创建 sqlSession 即会话，操作数据库需要通过 sqlSession 进行。

4、mybatis 底层自定义了 Executor 执行器接口操作数据库，Executor 接口有两个实现，一个是基本执行器、一个是缓存执行器。

5、Mapped Statement 也是 mybatis 一个底层封装对象，它包装了 mybatis 配置信息及 sql 映射信息等。mapper.xml 文件中一个 sql 对应一个 Mapped Statement 对象，sql 的 id 即是 Mapped statement 的 id。

6、Mapped Statement 对 sql 执行输入参数进行定义，包括 HashMap、基本类型、

pojo，Executor 通过 Mapped Statement 在执行 sql 前将输入的 java 对象映射至 sql 中，输入参数映射就是 jdbc 编程中对 preparedStatement 设置参数。

7、Mapped Statement 对 sql 执行输出结果进行定义，包括 HashMap、基本类型、pojo，Executor 通过 Mapped Statement 在执行 sql 后将输出结果映射至 java 对象中，输出结果映射过程相当于 jdbc 编程中对结果的解析处理过程。

5. MyBatis 开发 CRUD 案例

5.1. 抽取 MyBatis 工具类

```
public class SessionUtils {


    private static SqlSessionFactoryBuilder builder;
    private static SqlSessionFactory factory;


    /**
     * 初始化 SqlSessionFactory
     */
    static {
        try {
            builder = new SqlSessionFactoryBuilder();
            InputStream inputStream =
Resources.getResourceAsStream("sqlMapConfig.xml");
            factory = builder.build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    /**
     * 获取 sqlSession
     */
    public static SqlSession getSession(){
        return factory.openSession();
    }
}
```

5.2. 添加案例

```
    <!-- 添加 -->
    <insert id="insertCustomer"
parameterType="cn.sm1234.domain.Customer">
        INSERT INTO t_customer(NAME,gender,telephone)
VALUES(#{name},#{gender},#{telephone})
    </insert>
```

```
//添加操作
    @Test
    public void test1(){
        SqlSession sqlSession = SessionUtils.getSession();
        Customer c = new Customer();
        c.setName("李四");
        c.setGender("女");
        c.setTelephone("14355557777");
        sqlSession.insert("insertCustomer", c);
        //必须添加事务
        sqlSession.commit();
        sqlSession.close();
```

```
    }
```

5.3. 修改案例

```
<!-- 修改 -->
    <update id="updateCustomer"
parameterType="cn.sm1234.domain.Customer">
        UPDATE t_customer SET NAME = #{name} WHERE id = #{id}
    </update>
```

```
// 修改操作
    @Test
    public void test2() {
        SqlSession sqlSession = SessionUtils.getSession();
        Customer c = new Customer();
        c.setId(11);
        c.setName("老王");
        sqlSession.update("updateCustomer", c);
        // 必须添加事务
        sqlSession.commit();
        sqlSession.close();
    }
```

5.4. 查询所有

```
<!-- 查询所有数据 -->
    <select id="queryAllCustomer" resultType="cn.sm1234.domain.Customer">
        SELECT * FROM t_customer
    </select>
```

```
// 查询所有操作
    @Test
    public void test3() {
        SqlSession sqlSession = SessionUtils.getSession();
        List<Customer> list = sqlSession.selectList("queryAllCustomer");
        for (Customer customer : list) {
            System.out.println(customer);
        }
        sqlSession.close();
    }
```

5.5. 查询一个（根据主键查询）

```
<!-- 根据 id 查询 -->
    <select id="queryCustomerById" parameterType="int"
resultType="cn.sm1234.domain.Customer">
        SELECT * FROM t_customer WHERE id=#{value}
    </select>
```

```
    // 根据 id 查询操作
    @Test
    public void test4() {
        SqlSession sqlSession = SessionUtils.getSession();
        Customer c = sqlSession.selectOne("queryCustomerById", 3);
        System.out.println(c);
        sqlSession.close();
    }
```

5.6. 根据指定属性查询

```
<!-- 根据 name 模糊查询 -->
    <select id="queryCustomerByName"  parameterType="string"
resultType="cn.sm1234.domain.Customer">
        SELECT * FROM t_customer WHERE NAME LIKE #{value}
    </select>
```

```
    // 根据 name 模糊查询操作
    @Test
    public void test5() {
        SqlSession sqlSession = SessionUtils.getSession();
        List<Customer> list = sqlSession.selectList("queryCustomerByName",
"%张%");
        for (Customer customer : list) {
            System.out.println(customer);
        }
        sqlSession.close();
    }
```

5.7. 删除

```
<!-- 删除 -->
    <delete id="deleteCustomer" parameterType="int">
        DELETE FROM t_customer WHERE id=#{value}
    </delete>
```

```
// 删除操作
    @Test
    public void test6() {
        SqlSession sqlSession = SessionUtils.getSession();
```

```
        sqlSession.delete("deleteCustomer", 9);
        sqlSession.commit();
        sqlSession.close();
    }
```

6. MyBatis 核心 API 介绍

6.1. SqlSessionFactoryBuilder

SqlSessionFactoryBuilder 用于创建 SqlSessionFacoty，SqlSessionFacoty 一旦创建完成就不需要 SqlSessionFactoryBuilder 了，因为 SqlSession 是通过 SqlSessionFactory 生产，所以可以将 SqlSessionFactoryBuilder 当成一个工具类使用。

6.2. SqlSessionFactory

SqlSessionFactory 是一个接口，接口中定义了 openSession 的不同重载方法，SqlSessionFactory 的最佳使用范围是整个应用运行期间，一旦创建后可以重复使用，通常以单例模式管理 SqlSessionFactory。

6.3. SqlSession

SqlSession 是一个面向用户的接口， sqlSession 中定义了数据库操作方法。

每个线程都应该有它自己的 SqlSession 实例。SqlSession 的实例不能共享使用，它也是线程不安全的。因此最佳的范围是请求或方法范围。绝对不能将 SqlSession 实例的引用放在一个类的静态字段或实例字段中。

打开一个 SqlSession；使用完毕就要关闭它。通常把这个关闭操作放到 finally 块中以确保每次都能执行关闭。如下：

```
SqlSession session = sqlSessionFactory.openSession();
try {
        // do work
```

```
    } finally {
        session.close();
    }
```

7. 核心配置 sqlMapConfig 说明

SqlMapConfig.xml 中配置的内容和顺序如下：

properties（属性）

settings（全局配置参数）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

7.1. properties 属性的使用

7.1.1. 内部属性使用

直接在 sqlMapConfig.xml 定义和使用

```
<?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>
```

```
    <!-- 1、内部的属性用法 -->
    <properties>
        <property name="jdbc.driver" value="com.mysql.jdbc.Driver"/>
        <property name="jdbc.url"
value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"/>
    </properties>


    <!-- 数据库环境 -->
    <environments default="develop">
        <environment id="develop">
            <!-- jDBC 事务管理 -->
            <transactionManager type="JDBC"/>
            <!-- 连接池配置 -->
            <!--
                POOLED:代表使用 mybatis 内置连接池（*）
                UNPOOLED:代表不适用连接池
             -->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>


    <!-- sql 映射文件 -->
    <mappers>
        <mapper resource="Customer.xml"/>
```

```
    </mappers>


</configuration>
```

7.1.2. 外部属性的使用（*）

定义 jdbc.properties 文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8
```

sqlMapConfig.xml 使用

```
<?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>
    <!-- 1、内部的属性用法 -->
    <!-- <properties>
        <property name="jdbc.driver" value="com.mysql.jdbc.Driver"/>
        <property name="jdbc.url"
value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"/>
    </properties> -->


    <!-- 2.外部的属性用法 -->
    <properties resource="jdbc.properties"/>



    <!-- 数据库环境 -->
    <environments default="develop">
        <environment id="develop">
            <!-- jDBC 事务管理 -->
```

```
            <transactionManager type="JDBC"/>
            <!-- 连接池配置 -->
            <!--
                POOLED:代表使用 mybatis 内置连接池（*）
                UNPOOLED:代表不适用连接池
             -->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>

    <!-- sql 映射文件 -->
    <mappers>
        <mapper resource="Customer.xml"/>
    </mappers>

</configuration>
```

7.2. 别名的使用

7.2.1. mybatis 自身支持的别名

别名	映射的类型
_byte	byte
_long	long

_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
map	Map

7.2.2. 自定义别名

sqlMapConfig.xml 定义别名

```
<?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>
    <!-- 1、内部的属性用法 -->
    <!-- <properties>
```

```
        <property name="jdbc.driver" value="com.mysql.jdbc.Driver"/>
        <property name="jdbc.url"
value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"/>
    </properties> -->

    <!-- 2.外部的属性用法 -->
    <properties resource="jdbc.properties"/>

    <!-- 定义别名 -->
    <typeAliases>
        <!--
            type: 需要映射的类型
            alias:  别名
         -->
        <typeAlias type="cn.sm1234.domain.Customer" alias="customer"/>
    </typeAliases>

    <!-- 数据库环境 -->
    <environments default="develop">
        <environment id="develop">
            <!-- jDBC 事务管理 -->
            <transactionManager type="JDBC"/>
            <!-- 连接池配置 -->
            <!--
                POOLED:代表使用 mybatis 内置连接池（*）
                UNPOOLED:代表不适用连接池
             -->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
```

```
            <property name="url" value="${jdbc.url}"/>
            <property name="username" value="root"/>
            <property name="password" value="root"/>
        </dataSource>
    </environment>
</environments>

<!-- sql 映射文件 -->
<mappers>
    <mapper resource="Customer.xml"/>
</mappers>

</configuration>
```

Customer.xml 使用别名

```
<!-- 根据 id 查询 -->
    <select id="queryCustomerById" parameterType="Integer"
resultType="customer">
        SELECT * FROM t_customer WHERE id=#{value}
    </select>
```

7.3. mapper 映射器

方式一：直接导入 sql 映射文件

```
<!-- sql 映射文件 -->
    <mappers>
        <mapper resource="Customer.xml"/>
    </mappers>
```

方式二：导入 mapper 接口的方式

```
<mappers>
```

```
        <mapper class="mapper 接口"/>
    </mappers>
```

方式三：使用包扫描方式统一导入包下所有 mapper 接口

```
<mappers>
        <package name=""/>
    </mappers>
```