MyBatis

MyBatis 是一个开源、轻量级的数据持久化框架，是 JDBC 和 Hibernate 的替代方案。MyBatis 内部封装了 JDBC，简化了加载驱动、创建连接、创建 statement 等繁杂的过程，开发者只需要关注 SQL 语句本身。

MyBatis 支持定制化 SQL、存储过程以及高级映射，可以在实体类和 SQL 语句之间建立映射关系，是一种半自动化的 ORM 实现

ORM（Object Relational Mapping，对象关系映射）是一种数据持久化技术，它在对象模型和关系型数据库之间建立起对应关系，并且提供了一种机制，通过 JavaBean 对象去操作数据库表中的数据。

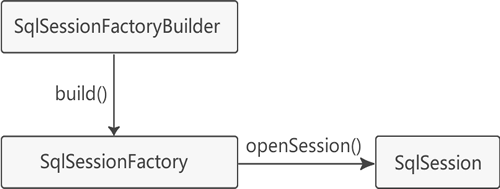
MyBatis 有三个基本要素：

1，核心接口和类

2，MyBatis核心配置文件（mybatis-config.xml）

3，SQL映射文件（mapper.xml）

MyBatis 的核心接口和类：



每个MyBatis 应用程序都以一个SqlSessionFactory 对象的实例为核心。

首先获取 SqlSessionFactoryBuilder 对象，可以根据 XML 配置文件或者 Configuration 类的实例构建该对象。

然后获取 SqlSessionFactory 对象，该对象实例可以通过 SqlSessionFactoryBuilder 对象来获取。

有了 SqlSessionFactory 对象之后，就可以进而获取 SqlSession 实例。SqlSession 对象中完全包含以数据库为背景的所有执行 SQL 操作的方法，用该实例可以直接执行已映射的 SQL 语句。

MyBatis 的工作流程可以分为几个主要步骤，从创建 SqlSessionFactory 到执行 SQL 查询并返回结果。以下是 MyBatis 的详细工作流程：

1. 配置 MyBatis 环境

首先，你需要配置 MyBatis 的核心组件，包括数据库连接、事务管理、映射文件等。

MyBatis 配置文件（mybatis-config.xml）：这是 MyBatis 的全局配置文件，通常包括数据库连接信息、事务管理器配置、日志配置、缓存配置等。

示例：

xml

<configuration>

<properties resource="jdbc.properties"/>

<settings>

<setting name="mapUnderscoreToCamelCase" value="true"/>

</settings>

<typeAliases>

<typeAlias alias="User" type="com.example.model.User"/>

</typeAliases>

<mappers>

<mapper resource="com/example/mapper/UserMapper.xml"/>

</mappers>

</configuration>

2. 创建 SqlSessionFactory

在 MyBatis 中，SqlSessionFactory 是执行 SQL 语句的核心，它是从 mybatis-config.xml 配置文件加载 MyBatis 的配置信息并创建的。

使用 SqlSessionFactoryBuilder 来构建 SqlSessionFactory 实例。

示例：

java

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

SqlSessionFactory 会读取 MyBatis 配置文件，加载数据库连接池信息、映射器等配置。

3. 获取 SqlSession

通过 SqlSessionFactory 获取 SqlSession，SqlSession 是 MyBatis 执行数据库操作的核心对象。

SqlSession 提供了与数据库交互的所有方法，如增、删、改、查等。

示例：java

SqlSession session = sqlSessionFactory.openSession();

通常会在 SqlSession 上执行事务提交和关闭操作，事务管理是 MyBatis 的一个重要功能。

4. 执行 SQL 查询

MyBatis 通过 Mapper 接口与 SQL 映射文件（Mapper XML）进行映射。你可以在 Mapper 接口中定义方法，在 XML 文件中配置对应的 SQL 语句。

4.1 Mapper 接口与 XML 映射文件

你可以创建一个 Java 接口（Mapper），然后在映射文件中为这个接口的方法定义 SQL 语句。

Mapper 接口：

java

public interface UserMapper {

User getUserById(int id);

}

Mapper XML 文件：

xml

<mapper namespace="com.example.mapper.UserMapper">

<select id="getUserById" resultType="com.example.model.User">

SELECT \* FROM users WHERE id = #{id}

</select>

</mapper>

4.2 执行 SQL

在 SqlSession 中调用 getMapper() 方法，获得接口的实现，然后调用该方法执行 SQL 查询。

通过 getMapper() 获取 Mapper 接口的代理对象，并执行映射的方法：

java

UserMapper userMapper = session.getMapper(UserMapper.class);

User user = userMapper.getUserById(1);

System.out.println(user.getName());

5. MyBatis 执行 SQL 语句

当调用 getUserById(1) 时，MyBatis 会：

・从 UserMapper.xml 中找到对应的 SQL 语句 <select id="getUserById">。

・使用传入的参数（1）替换 SQL 中的 #{id} 占位符。

・执行 SQL 查询并获取数据库结果集。

・将结果集转换成对应的 Java 对象（在这个例子中是 User 对象）。

6. 事务管理

在 MyBatis 中，事务是通过 SqlSession 来管理的。默认情况下，MyBatis 使用数据库的事务管理，事务是手动提交和回滚的。

在执行数据库操作时，如果没有显式提交事务，MyBatis 会在 SqlSession 关闭时自动提交。

如果需要控制事务，可以手动提交或回滚事务。

示例：java

try {

// 执行 SQL

session.commit(); // 提交事务

} catch (Exception e) {

session.rollback(); // 回滚事务

} finally {

session.close(); // 关闭 session

}

7. 返回结果

MyBatis 会根据 resultType 或 resultMap 的配置，将查询结果映射到 Java 对象。例如，getUserById 方法会返回一个 User 对象，它会自动将数据库查询结果的字段映射到 User 类的属性。

8. 关闭 SqlSession

完成数据库操作后，需要关闭 SqlSession，这是 MyBatis 的资源管理的一部分。关闭 SqlSession 会释放数据库连接和其他相关资源。

java

session.close();

9. 执行其他操作（可选）

MyBatis 还支持批量操作、缓存机制、动态 SQL 等功能。这些可以根据实际需求进一步使用。

一个项目的文件路径总成如下：

src/

├── main/

│ ├── java/

│ │ └── com/

│ │ └── example/

│ │ └── mybatis/

│ │ ├── Main.java <-- 这个是执行类

│ │ ├── mapper/

│ │ │ └── UserMapper.java <-- 这个是接口文件

│ │ └── model/

│ │ └── User.java <-- 这个是持久化类（注）

│ └── resources/

│ └── mybatis-config.xml <-- 这个是配置文件

│ └── mapper/

│ └── UserMapper.xml <-- 这个是映射文件

└── pom.xml

注意：持久化类也是JavaBean类

注意：SqlSessionFactoryBuilder() 是 构造器，用于构建 SqlSessionFactory 对象。

在 MyBatis 中，SqlSessionFactoryBuilder 是一个帮助类，用于通过读取 MyBatis 配置文件来构建 SqlSessionFactory。你通过调用 SqlSessionFactoryBuilder 的构造器来创建它的实例，然后使用 build() 方法来根据配置文件生成一个 SqlSessionFactory 对象。

SqlSessionFactoryBuilder 类

SqlSessionFactoryBuilder 是一个没有复杂构造参数的简单构造器。它的构造方法是默认的，也就是不带参数的构造器：

java

public class SqlSessionFactoryBuilder {

public SqlSessionFactoryBuilder() {

// 默认构造器，不做任何初始化工作

}

public SqlSessionFactory build(InputStream inputStream) {

// 根据输入流（MyBatis配置文件）构建 SqlSessionFactory

}

}

构造器的使用

你通常会看到它用来创建 SqlSessionFactoryBuilder 的实例，然后调用 build() 方法来根据传入的配置文件（例如 mybatis-config.xml）生成 SqlSessionFactory。

java

// 创建 SqlSessionFactoryBuilder 的实例

SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

// 通过配置文件构建 SqlSessionFactory

SqlSessionFactory sqlSessionFactory = builder.build(config);

或者更简洁的方式是直接使用：

java

// 直接创建并使用构造器

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(config);

总结

SqlSessionFactoryBuilder 是一个 构造器 类，提供了构建 SqlSessionFactory 的功能。

你可以通过 new SqlSessionFactoryBuilder() 创建它的实例，并通过调用 build() 方法生成 SqlSessionFactory。

该类没有特殊的构造函数，你只需要调用默认构造函数就可以实例化它。

MyBatis执行SQL的两种方式

MyBatis 有两种执行 SQL 语句的方式，如下：

1，通过 SqlSession 发送 SQL

2，通过 SqlSession 获取 Mapper 接口，通过 Mapper 接口发送 SQL

**SqlSession发送SQL**

有映射器之后就可以通过 SqlSession 发送 SQL 了。我们以 getWebsite 这条 SQL 为例看看如何发送 SQL。

Website website =

(Website)sqlSession.selectOne("net.biancheng.mapper.WebsiteMapper.getWebsite",1);

1）selectOne

selectOne 方法表示使用查询并且只返回一个对象，必须指定查询条件。只能查询 0 或 1 条记录，大于 1 条记录则运行错误。常用格式如下（也有其它重载方法，根据需要选择）。

sqlSession.selectOne(String arg0, Object arg1)

2）selectList

selectList 方法表示使用查询并且返回一个列表。可以查询 0 或 N 条记录。常用格式如下。

sqlSession.selectOne(String arg0)

也可指定参数：

sqlSession.selectList(String arg0, Object arg1)

**Mapper接口发送 SQL（主要使用）**

SqlSession 还可以获取 Mapper 接口，通过 Mapper 接口发送 SQL，如下所示。

WebsiteMapper websiteMapper =

sqlSession.getMapper(WebsiteMapper.class);

Website website = websiteMapper.getWebsite(1);

select标签

在 MyBatis 中，select 标签是最常用也是功能最强大的 SQL 语言，用于执行查询操作。

select 示例语句如下。

<select id="selectAllWebsite" resultType="net.biancheng.po.Website" parameterType="string">

SELECT id,NAME,url FROM website WHERE NAME LIKE CONCAT ('%',#{name},'%')

</select>

注意:为了使数据库的查询结果和返回值类型中的属性能够自动匹配，通常会对 MySQL 数据库和 JavaBean 采用同一套命名规则，即 Java 命名驼峰规则，这样就不需要再做映射了（数据库表字段名和属性名不一致时需要手动映射）

select标签常用属性

|  |  |  |
| --- | --- | --- |
| **属性名称** | **描 述** | **备注** |
| id | 它和 Mapper 的命名空间组合起来使用，是唯一标识符，供 MyBatis 调用 | 如果命名空间+id不唯一，那么 MyBatis 抛出异常 |
| parameterType | 表示传入 SQL 语句传入参数类型的全限定名或别名。它是一个可选属性，MyBatis 能推断出具体传入语句的参数 | 支持基本数据类型和 JavaBean、Map 等复杂数据类型 |
| resultType | SQL 语句执行后返回的类型（全限定名或者别名）。如果是集合类型，返回的是集合元素的类型，返回时可以使用 resultType 或 resultMap 之一 | - |
| resultMap | 它是映射集的引用，与 <resultMap> 元素一起使用，返回时可以使用 resultType 或 resultMap 之一 | 是 MyBatis 最复杂的元素，可以配置映射规则、级联、typeHandler 等 |
| flushCache | 用于设置在调用 SQL 语句后是否要求 MyBatis 清空之前查询的本地缓存和二级缓存 | 默认值为 false，如果设置为 true，则任何时候只要 SQL 语句被调用都将清空本地缓存和二级缓存 |
| useCache | 启动二级缓存的开关，默认值为 true，表示将査询结果存入二级缓存中 | - |
| timeout | 用于设置超时参数，单位是秒（s），超时将抛出异常 | - |
| fetchSize | 获取记录的总条数设定 | 默认值是数据库厂商提供的 JDBC 驱动所设置的条数 |
| statementType | 告诉 MyBatis 使用哪个 JDBC 的 Statement 工作，取值为 STATEMENT（Statement）、 PREPARED（PreparedStatement）、CALLABLE（CallableStatement） | - |
| resultSetType | 这是针对 JDBC 的 ResultSet 接口而言，其值可设置为 FORWARD\_ONLY（只允许向前访问）、SCROLL\_SENSITIVE（双向滚动，但不及时更新）、SCROLLJNSENSITIVE（双向滚动，及时更新） | - |

传递多个参数

现在需要根据 id 和 name 来模糊查询网站信息，显然这涉及到了两个参数。给映射器传递多个参数分为以下三种方法。

1，使用Map传递参数

2，使用注解传递参数

3，使用JavaBean传递参数

1. 使用Map传递参数

使用 MyBatis 提供的 Map 接口作为参数实现，如下所示。

<!-- 根据name和url模糊查询网站信息 -->

<select id="selectWebsiteByMap" resultType="net.biancheng.po.Website" parameterType="map">

SELECT id,NAME,url FROM website

WHERE name LIKE CONCAT ('%',#{name},'%')

AND url LIKE CONCAT ('%',#{url},'%')

</select>

在 WebsiteMapper 接口中，方法如下。

public List<Website> selectWebsiteByMap(Map<String, String> params);

测试代码如下。

Map<String,String> paramsMap = new HashMap<String,String>();

paramsMap.put("name","编程");

paramsMap.put("url","biancheng");

websiteMapper.selectWebsiteByMap(paramsMap);

2. 使用注解传递参数

使用 MyBatis 的注解 @Param() 传递参数，如下所示。

<!-- 根据name和url模糊查询网站信息 -->

<select id="selectWebsiteByAn" resultType="net.biancheng.po.Website">

SELECT id,NAME,url FROM website

WHERE name LIKE CONCAT ('%',#{name},'%')

AND url LIKE CONCAT ('%',#{url},'%')

</select>

WebsiteMapper 接口中方法如下。

public List<Website> selectWebsiteByAn(@Param("name") String name, @Param("url") String url);

当我们把参数传递给后台时，MyBatis 通过 @Param 提供的名称就会知道 #{name} 代表 name 参数，提高了参数可读性。但是如果这条 SQL 拥有 10 个参数的查询，就会造成可读性下降，增强了代码复杂性。

3. 使用JavaBean传递参数

在参数过多的情况下，MyBatis 允许组织一个 JavaBean，通过简单的 setter 和 getter 方法设置参数，提高可读性。如下所示。

<!-- 根据name和url模糊查询网站信息 -->

<select id="selectWebsiteByAn" resultType="net.biancheng.po.Website">

SELECT id,NAME,url FROM website

WHERE name LIKE CONCAT ('%',#{name},'%')

AND url LIKE CONCAT ('%',#{url},'%')

</select>

WebsiteMapper 接口中方法如下。

public List<Website> selectWebsiteByAn(Website website);

这就是通过 JavaBean 传递多个参数的方式。

以上 3 种方式的区别如下。

1，使用 Map 传递参数会导致业务可读性的丧失，继而导致后续扩展和维护的困难，所以在实际应用中我们应该果断废弃该方式。

2，使用 @Param 注解传递参数会受到参数个数的影响。当 n≤5 时，它是最佳的传参方式，因为它更加直观；当 n>5 时，多个参数将给调用带来困难。

3，当参数个数大于 5 个时，建议使用 JavaBean 方式。

insert标签

MyBatis insert 标签用来定义插入语句，执行插入操作。当 MyBatis 执行完一条插入语句后，就会返回其影响数据库的行数。

下面通过一个示例演示 insert 标签的具体用法。

1. 修改 WebsiteMapper.xml，增加插入语句，代码如下。

<!-- 增加网站信息 -->

<insert id="addWebsite" parameterType="string">

insert into website(name)

values(#{name})

</insert>

2. 在 WebsiteMapper 接口中定义一个 add() 方法，代码如下。

public int addWebsite(String name);

参数为 Sting 类型的字符串；返回值为 int 类型，即执行 SQL 后，插入记录的行数。

3. 测试代码如下。

//插入 name 为编程帮4 的记录

String name = "编程帮4";

int i = websiteMapper.addWebsite(name);

System.out.println("共插入了 " + i + " 条记录");

4. 执行测试代码，控制台输出如下。

共插入了 1 条记录

insert 标签常用属性

|  |  |  |
| --- | --- | --- |
| **属性名称** | **描述** | **备注** |
| id | 它和 Mapper 的命名空间组合起来使用，是唯一标识符，供 MyBatis 调用 | 如果命名空间+ id 不唯一，那么 MyBatis 抛出异常 |
| parameterType | 传入 SQL 语句的参数类型的全限定名或别名，它是一个可选属性。 | 支持基本数据类型和 JavaBean、Map 等复杂数据类型 |
| keyProperty | 该属性的作用是将插入操作的返回值赋给 PO 类的某个属性，通常为主键对应的属性。如果是联合主键，可以将多个值用逗号隔开。 | - |
| useGeneratedKe | 该属性用来设置，是否使用 JDBC 提供的 getGenereatedKeys() 方法，获取数据库内部产生的主键并赋值到 keyProperty 属性设置的请求对象的属性中，例如 [MySQL](https://c.biancheng.net/mysql/" \t "https://c.biancheng.net/mybatis/_blank)、SQL Server 等自动递增的字段，其默认值为 false。 | 该属性值设置为 true 后，会将数据库生成的主键回填到请求对象中，以供其他业务使用。 |
| flushCache | 该属性用于设置执行该操作后，是否会清空二级缓存和本地缓存，默认值为 true。 | - |
| timeout | 该属性用于设置执行该操作的最大时限，如果超时，就抛异常。 | - |
| databaseId | 取值范围 oracle、mysql 等，表示数据库厂家；元素内部可通过 <if test="\_databaseId = 'oracle'"> 来为特定数据库指定不同的 sql 语句。 | MyBatis 可以根据不同的数据库厂商执行不同的语句，这种多厂商的支持是基于映射语句中的 databaseId 属性。 MyBatis 会加载不带 databaseId 属性和带有匹配当前数据库 databaseId 属性的所有语句。 如果同时找到带有 databaseId 和不带 databaseId 的相同语句，则后者会被舍弃。 |
| keyColumn | 该属性用于设置第几列是主键，当主键列不是表中的第 1 列时，就需要设置该属性。如果是联合主键，可以将多个值用逗号隔开。 | - |

主键（自动递增）回填

我们知道，MySQL、SQL Server 等数据库表可以采用自动递增的字段作为其主键，当向这样的数据库表插入数据时，即使不指定自增主键的值，数据库也会根据自增规则自动生成主键并插入到表中。

一些特殊情况下，我们可能需要将这个刚刚生成的主键回填到请求对象（原本不包含主键信息的请求对象）中，供其他业务使用。此时，我们就可以通过在 insert 标签中添加 keyProperty 和 useGeneratedKeys 属性，来实现该功能。

下面我们通过一个示例，来演示主键（自动递增）回填功能。

1. 为 WebsiteMapper.xml 中 id 为 addWebsite 的 insert 标签添加 keyProperty 和 useGeneratedKeys 属性，具体代码如下：

<!--添加一个网站信息，成功后将主键值返回填给id(po的属性)-->

<insert id="addWebsite" parameterType="net.biancheng.po.Website" keyProperty="id" useGeneratedKeys="true">

insert into Website (name,url) values(#{name},#{url})

</insert>

2. 测试代码如下：

// 添加一个网站信息

Website addsite = new Website();

//插入的对象中不包含主键 id

addsite.setName("编程帮");

addsite.setUrl("https://www.biancheng.net/");

//执行插入

int num = websiteMapper.addWebsite(addsite);

System.out.println("添加了 " + num + " 条记录");

//获取回填的主键

System.out.println("添加记录的主键是:" + addsite.getId());

3. 执行测试代码，控制台输出如下。

添加了 1 条记录

添加记录的主键是:3

update标签

MyBatis update 标签用于定义更新语句，执行更新操作。当 MyBatis 执行完一条更新语句后，会返回一个整数，表示受影响的数据库记录的行数。

下面我们通过一个示例演示 update 标签的用法。

1. 在 WebsiteMapper.xml 中添加以下更新语句，代码如下。

<!--update 标签-->

<update id="updateWebsite" parameterType="string">

update website set name = #{name}

</update>

2. 在 WebsiteMapper 接口中增加一个 updateWebsite() 方法，代码如下。

int updateWebsite(String name);

参数为 String 类型的字符串；返回值为 int 类型，表示执行 sql 语句后受影响的记录的行数。

3. 测试代码如下。

int i = websiteMapper.updateWebsite("C语言中文网");

System.out.println("共更新了 " + i + " 条记录");

4.执行测试代码，控制台输出如下。

共更新了 9 条记录

通过控制台输出可知，该更新语句执行后，共影响了 9 条数据库记录。

update 标签常用属性

|  |  |  |
| --- | --- | --- |
| **属性名称** | **描述** | **备注** |
| id | 它和 Mapper 的命名空间组合起来使用，是唯一标识符，供 MyBatis 调用 | 如果命名空间+ id 不唯一，那么 MyBatis 抛出异常 |
| parameterType | 传入 SQL 语句的参数类型的全限定名或别名，它是一个可选属性。 | 支持基本数据类型和 JavaBean、Map 等复杂数据类型 |
| flushCache | 该属性用于设置执行该操作后，是否会清空二级缓存和本地缓存，默认值为 true。 | - |
| timeout | 该属性用于设置 SQL 执行的超时时间，如果超时，就抛异常。 | - |
| statementType | 执行 SQL 时使用的 statement 类型, 默认为 PREPARED，可选值：STATEMENT，PREPARED 和 CALLABLE。 | - |

delete标签

MyBatis delete 标签用于定义 delete 语句，执行删除操作。当 MyBatis 执行完一条更新语句后，会返回一个整数，表示受影响的数据库记录的行数。

下面我们通过一个示例演示 delete 标签的用法。

1. 在 WebsiteMapper.xml 中使用 delete 标签添加一条 delete 语句，代码如下。

<delete id="deleteWebsite" parameterType="string">

delete from website where name = #{name}

</delete>

2. 在 WebsiteMapper 接口中增加一个 deleteWebsite() 方法，代码如下。

int deleteWebsite(String name);

参数为 String 类型的字符串；返回值为 int 类型，表示执行 sql 语句后，被删除记录的行数。

3. 测试代码如下。

//删除 name 为编程帮3 的记录

String name = "编程帮3";

int i = websiteMapper.deleteWebsite(name);

System.out.println("共删除了 " + i + " 条记录");

4. 执行测试代码，控制台输出如下。

共删除了 3 条记录

delete 标签常用属性

|  |  |  |
| --- | --- | --- |
| **属性名称** | **描述** | **备注** |
| id | 它和 Mapper 的命名空间组合起来使用，是唯一标识符，供 MyBatis 调用 | 如果命名空间+ id 不唯一，那么 MyBatis 抛出异常 |
| parameterType | 传入 SQL 语句的参数类型的全限定名或别名，它是一个可选属性。 | 支持基本数据类型和 JavaBean、Map 等复杂数据类型 |
| flushCache | 该属性用于设置执行该操作后，是否会清空二级缓存和本地缓存，默认值为 true。 | - |
| timeout | 该属性用于设置 SQL 执行的超时时间，如果超时，就抛异常。 | - |
| statementType | 执行 SQL 时使用的 statement 类型, 默认为 PREPARED，可选值：STATEMENT，PREPARED 和 CALLABLE。 | - |

注意：delete 标签中没有 resultType 属性，只有查询操作才需要对返回结果类型进行相应的指定。

**resultMap元素**

resultMap 是 MyBatis 中最复杂的元素，主要用于解决实体类属性名与数据库表中字段名不一致的情况，可以将查询结果映射成实体对象。下面我们先从最简单的功能开始介绍。

resultMap元素的构成

resultMap 元素还可以包含以下子元素，代码如下。

<resultMap id="" type="">

<constructor><!-- 类再实例化时用来注入结果到构造方法 -->

<idArg/><!-- ID参数，结果为ID -->

<arg/><!-- 注入到构造方法的一个普通结果 -->

</constructor>

<id/><!-- 用于表示哪个列是主键 -->

<result/><!-- 注入到字段或JavaBean属性的普通结果 -->

**<association property=""/><!-- 用于一对一关联 -->**

**<collection property=""/><!-- 用于一对多、多对多关联 -->**

<discriminator javaType=""><!-- 使用结果值来决定使用哪个结果映射 -->

<case value=""/><!-- 基于某些值的结果映射 -->

</discriminator>

</resultMap>

其中：

<resultMap> 元素的 type 属性表示需要的 POJO，id 属性是 resultMap 的唯一标识。

子元素 <constructor> 用于配置构造方法。当一个 POJO 没有无参数构造方法时使用。

子元素 <id> 用于表示哪个列是主键。允许多个主键，多个主键称为联合主键。

子元素 <result> 用于表示 POJO 和 SQL 列名的映射关系。

子元素 <association>、<collection> 和 <discriminator> 用在级联的情况下。关于级联的问题比较复杂，在《MyBatis一对一关联查询》和《MyBatis一对多关联查询》一节详细讲解。

id 和 result 元素都有以下属性

|  |  |
| --- | --- |
| **元素** | **说明** |
| property | 映射到列结果的字段或属性。如果 POJO 的属性和 SQL 列名（column元素）是相同的，那么 MyBatis 就会映射到 POJO 上 |
| column | 对应 SQL 列 |
| javaType | 配置 Java 类型。可以是特定的类完全限定名或 MyBatis 上下文的别名 |
| jdbcType | 配置数据库类型。这是 JDBC 类型，MyBatis 已经为我们做了限定，基本支持所有常用数据库类型 |
| typeHandler | 类型处理器。允许你用特定的处理器来覆盖 MyBatis 默认的处理器。需要指定 jdbcType 和 javaType 相互转化的规则 |

结果集的储存:

一条 SQL 查询语句执行后会返回结果集，结果集有两种存储方式，即使用 Map 存储和使用 POJO 存储。

使用Map存储结果集

任何 select 语句都可以使用 Map 存储，代码如下。

<!-- 查询所有网站信息存到Map中 -->

<select id="selectAllWebsite" resultType="map">

select \* from website

</select>

在 WebsiteMapper 接口中添加以下方法。

public List<Map<String,Object>> selectAllWebsite();

Map 的 key 是 select 语句查询的字段名（必须完全一样），而 Map 的 value 是查询返回结果中字段对应的值，一条记录映射到一个 Map 对象中。

使用 Map 存储结果集很方便，但可读性稍差，所以一般推荐使用 POJO 的方式。

使用POJO存储结果集

因为 MyBatis 提供了自动映射，所以使用 POJO 存储结果集是最常用的方式。但有时候需要更加复杂的映射或级联，这时就需要使用 select 元素的 resultMap 属性配置映射集合。

修改 Website 类，代码如下。

package net.biancheng.po;

import java.util.Date;

public class Website {

private int id;

private String uname;

private String url;

private int age;

private String country;

private Date createtime;

/\* setter和getter方法\*/

@Override

public String toString() {

return "Website[id=" + id + ",uname=" + uname + ",url=" + url + ",age=" + age + ",country=" + country

+ ",createtime=" + createtime + "]";

}

}

WebsiteMapper.xml 代码如下。

<!--使用自定义结果集类型 -->

<resultMap type="net.biancheng.po.Website" id="myResult">

<!-- property 是 net.biancheng.po.Website 类中的属性 -->

<!-- column是查询结果的列名，可以来自不同的表 -->

<id property="id" column="id" />

<result property="uname" column="name" />

</resultMap>

resultMap 元素的属性 id 代表这个 resultMap 的标识，type 标识需要映射的 POJO。我们可以使用 MyBatis 定义好的类的别名或自定义类的全限定名。

这里使用 property 元素指定 Website 的属性名称 uname，column 表示数据库中 website 表的 SQL 列名 name，将 POJO 和 SQL 的查询结果一 一对应。

WebsiteMapper.xml 中 select 元素配置代码如下。

<select id="selectAllWebsite" resultMap="myResult">

select id,name,url from website

</select>

可以发现 SQL 语句的列名和 myResult 中的 column 一一对应。

resultType和resultMap的区别

MyBatis 的每一个查询映射的返回类型都是 resultMap，只是当我们提供的返回类型是 resultType 时，MyBatis 会自动把对应的值赋给 resultType 所指定对象的属性，而当我们提供的返回类型是 resultMap 时，MyBatis 会将数据库中的列数据复制到对象的相应属性上，可用于复制查询。

需要注意的是，resultMap 和 resultType 不能同时使用。

MyBatis注解（3种类型）

为了简化 XML 的配置，MyBatis 提供了注解。以上注解主要分为三大类，即 SQL 语句映射、结果集映射和关系映射。下面分别进行讲解。

1. SQL 语句映射

1）@Insert：实现新增功能

@Insert("insert into user(id,name) values(#{id},#{name})")

public int insert(User user);

2）@Select：实现查询功能

@Select("Select \* from user")

@Results({

@Result(id = true, column = "id", property = "id"),

@Result(column = "name", property = "name"),

@Result(column = "sex", property = "sex"),

@Result(column = "age", property = "age")

})

List<User> queryAllUser();

3）@SelectKey：插入后，获取id的值

以 MySQL 为例，MySQL 在插入一条数据后，使用 select last\_insert\_id() 可以获取到自增 id 的值。

@Insert("insert into user(id,name) values(#{id},#{name})")

@SelectKey(statement = "select last\_insert\_id()", keyProperty = "id", keyColumn = "id", resultType = int,before = false)

public int insert(User user);

@SelectKey 各个属性含义如下。

statement：表示要运行的 SQL 语句；

keyProperty：可选项，表示将查询结果赋值给代码中的哪个对象；

keyColumn：可选项，表示将查询结果赋值给数据表中的哪一列；

resultType：指定 SQL 语句的返回值；

before：默认值为 true，在执行插入语句之前，执行 select last\_insert\_id()。值为 flase，则在执行插入语句之后，执行 select last\_insert\_id()。

4）@Insert：实现插入功能

@Insert("insert into user(name,sex,age) values(#{name},#{sex},#{age}")

int saveUser(User user);

5）@Update：实现更新功能

@Update("update user set name= #{name},sex = #{sex},age =#{age} where id = #{id}")

void updateUserById(User user);

6）@Delete：实现删除功能

@Delete("delete from user where id =#{id}")

void deleteById(Integer id);

7）@Param：映射多个参数（常用）

@Param 用于在 Mapper 接口中映射多个参数。

int saveUser(@Param(value="user") User user,@Param("name") String name,@Param("age") Int age);

@Param 中的 value 属性可省略，用于指定参数的别名。

2. 结果集映射

@Result、@Results、@ResultMap 是结果集映射的三大注解。

声明结果集映射关系代码：

@Select({"select id, name, class\_id from student"})

@Results(id="studentMap", value={

@Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),

@Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),

@Result(column="class\_id ", property="classId", jdbcType=JdbcType.INTEGER)

})

List<Student> selectAll();

下面为 @Results 各个属性的含义。

id：表示当前结果集声明的唯一标识；

value：表示结果集映射关系；

@Result：代表一个字段的映射关系。其中，column 指定数据库字段的名称，property 指定实体类属性的名称，jdbcType 数据库字段类型，id 为 true 表示主键，默认 false。

可使用 @ResultMap 来引用映射结果集，其中 value 可省略。

@Select({"select id, name, class\_id from student where id = #{id}"})

@ResultMap(value="studentMap")

Student selectById(Integer id);

这样不需要每次声明结果集映射时都复制冗余代码，简化开发，提高了代码的复用性。

**MyBatis关联（级联）查询**

通过 <resultMap> 元素的子元素 <association> 处理一对一级联关系。

通过 <resultMap> 元素的子元素 <collection> 处理一对多级联关系

示例代码如下。

<!-- 查询每门课的教师名字，以及这门课成绩大于60的同学的信息 -->

<resultMap id="score60" type="com.example.mybatis.domain.course">

<id property="CPlus" column="CPlus" />

<result property="Cname" column="Cname" />

<association property="teacher"

javaType="com.example.mybatis.domain.Teacher">

<id property="TPlus" column="TPlus" />

<result property="Tname" column="Tname" />

</association>

<collection property="sc" ofType="com.example.mybatis.domain.sc">

<id property="id" column="id" />

<result property="score" column="score" />

<association property="Student"

javaType="com.example.mybatis.domain.Student">

<id property="SPlus" column="SPlus" />

<result property="Sname" column="Sname" />

</association>

</collection>

</resultMap>

<select id="select60" parameterType="com.example.mybatis.domain.parameter" resultMap="score60">

select c.Cname,t.Tname,sc.score,s.SPlus,s.Sname

from course as c

inner join teacher as t

on c.TPlus = t.TPlus

inner join sc

on c.CPlus = sc.CPlus

inner join student as s

on sc.SPlus = s.SPlus

<trim prefix="where" prefixOverrides="and">

<if test="avg!=0">

and sc.score &lt;=#{avg}

</if>

<if test="Sname!=null">

and Sname=#{Sname}

</if>

</trim>

</select>

**MyBatis动态SQL**

MyBatis 的动态 SQL 包括以下几种元素，如下表所示。

|  |  |  |
| --- | --- | --- |
| **元素** | **作用** | **备注** |
| [if](https://c.biancheng.net/mybatis/if.html" \t "https://c.biancheng.net/mybatis/_blank) | 判断语句 | 单条件分支判断 |
| [choose（when、otherwise）](https://c.biancheng.net/mybatis/choose-when-otherwise.html" \t "https://c.biancheng.net/mybatis/_blank) | 相当于 Java 中的 switch case 语句 | 多条件分支判断 |
| [trim](https://c.biancheng.net/mybatis/trim.html" \t "https://c.biancheng.net/mybatis/_blank)、[where](https://c.biancheng.net/mybatis/where.html" \t "https://c.biancheng.net/mybatis/_blank) | 辅助元素 | 用于处理一些SQL拼装问题 |
| [foreach](https://c.biancheng.net/mybatis/foreach.html" \t "https://c.biancheng.net/mybatis/_blank) | 循环语句 | 在in语句等列举条件常用 |
| [bind](https://c.biancheng.net/mybatis/bind.html" \t "https://c.biancheng.net/mybatis/_blank) | 辅助元素 | 拼接参数 |

**if标签：条件判断**

MyBatis if 类似于 Java 中的 if 语句，是 MyBatis 中最常用的判断语句。使用 if 标签可以节省许多拼接 SQL 的工作，把精力集中在 XML 的维护上。

if 语句使用方法简单，常常与 test 属性联合使用。语法如下。

<if test="判断条件">

SQL语句

</if>

当判断条件为 true 时，才会执行所包含的 SQL 语句。

最常见的场景是在 if 语句中包含 where 子句，例如。

<select id="selectAllWebsite" resultMap="myResult">

select id,name,url from website

<if test="name != null">

where name like #{name}

</if>

</select>

以上代表表示根据网站名称去查找相应的网站信息，但是网站名称是一个可填可不填的条件，不填写的时候不作为查询条件。

**where 标签**

where 标签主要用来简化 SQL 语句中的条件判断，可以自动处理 AND/OR 条件，语法如下。

<where>

<if test="判断条件">

AND/OR ...

</if>

</where>

if 语句中判断条件为 true 时，where 关键字才会加入到组装的 SQL 里面，否则就不加入。where 会检索语句，它会将 where 后的第一个 SQL 条件语句的 AND 或者 OR 关键词去掉。

**choose 标签**

MyBatis 中动态语句 choose-when-otherwise 类似于 Java 中的 switch-case-default 语句。由于 MyBatis 并没有为 if 提供对应的 else 标签，如果想要达到<if>...<else>...</else> </if> 的效果，可以借助 <choose>、<when>、<otherwise> 来实现。

动态语句 choose-when-otherwise 语法如下。

<choose>

<when test="判断条件1">

SQL语句1

</when >

<when test="判断条件2">

SQL语句2

</when >

<when test="判断条件3">

SQL语句3

</when >

<otherwise>

SQL语句4

</otherwise>

</choose>

choose 标签按顺序判断其内部 when 标签中的判断条件是否成立，如果有一个成立，则执行相应的 SQL 语句，choose 执行结束；如果都不成立，则执行 otherwise 中的 SQL 语句。这类似于 Java 的 switch 语句，choose 为 switch，when 为 case，otherwise 则为 default。

**set标签**

在 Mybatis 中，update 语句可以使用 set 标签动态更新列。set 标签可以为 SQL 语句动态的添加 set 关键字，剔除追加到条件末尾多余的逗号

<update id="updateWebsite"

parameterType="net.biancheng.po.Website">

UPDATE website

<set>

<if test="name!=null">name=#{name}</if>

<if test="url!=null">url=#{url}</if>

</set>

WHERE id=#{id}

</update>

**foreach标签**

对于一些 SQL 语句中含有 in 条件，需要迭代条件集合来生成的情况，可以使用 foreach 来实现 SQL 条件的迭代。

Mybatis foreach 标签用于循环语句，它很好的支持了数据和 List、set 接口的集合，并对此提供遍历的功能。语法格式如下。

<foreach item="item" index="index" collection="list|array|map key" open="(" separator="," close=")">

参数值

</foreach>

foreach 标签主要有以下属性，说明如下。

item：表示集合中每一个元素进行迭代时的别名。

index：指定一个名字，表示在迭代过程中每次迭代到的位置。

open：表示该语句以什么开始（既然是 in 条件语句，所以必然以(开始）。

separator：表示在每次进行迭代之间以什么符号作为分隔符（既然是 in 条件语句，所以必然以,作为分隔符）。

close：表示该语句以什么结束（既然是 in 条件语句，所以必然以)开始）。

使用 foreach 标签时，最关键、最容易出错的是 collection 属性，该属性是必选的，但在不同情况下该属性的值是不一样的，主要有以下 3 种情况：

如果传入的是单参数且参数类型是一个 List，collection 属性值为 list。

如果传入的是单参数且参数类型是一个 array 数组，collection 的属性值为 array。

如果传入的参数是多个，需要把它们封装成一个 Map，当然单参数也可以封装成 Map。Map 的 key 是参数名，collection 属性值是传入的 List 或 array 对象在自己封装的 Map 中的 key。

<select id="selectWebsite"

parameterType="net.biancheng.po.Website"

resultType="net.biancheng.po.Website">

SELECT id,name,url,age,country

FROM website WHERE age in

<foreach item="age" index="index" collection="list" open="("

separator="," close=")">

#{age}

</foreach>

</select>

**bind 标签**

每个数据库的拼接函数或连接符号都不同，例如 MySQL 的 concat 函数、Oracle 的连接符号“||”等。这样 SQL 映射文件就需要根据不同的数据库提供不同的实现，显然比较麻烦，且不利于代码的移植。幸运的是，MyBatis 提供了 bind 标签来解决这一问题。

bind 标签可以通过 OGNL 表达式自定义一个上下文变量。

比如，按照网站名称进行模糊查询，SQL 映射文件如下。

<select id="selectWebsite" resultType="net.biancheng.po.Website">

<bind name="pattern" value="'%'+\_parameter+'%'" />

SELECT id,name,url,age,country

FROM website

WHERE name like #{pattern}

</select>

bind 元素属性如下。

value：对应传入实体类的某个字段，可以进行字符串拼接等特殊处理。

name：给对应参数取的别名。

以上代码中的“\_parameter”代表传递进来的参数，它和通配符连接后，赋给了 pattern，然后就可以在 select 语句中使用这个变量进行模糊查询，不管是 MySQL 数据库还是 Oracle 数据库都可以使用这样的语句，提高了可移植性。

**trim标签**

在 MyBatis 中除了使用 if+where 实现多条件查询，还有一个更为灵活的元素 trim 能够替代之前的做法。

trim 一般用于去除 SQL 语句中多余的 AND 关键字、逗号，或者给 SQL 语句前拼接 where、set 等后缀，可用于选择性插入、更新、删除或者条件查询等操作。trim 语法格式如下。

<trim prefix="前缀" suffix="后缀" prefixOverrides="忽略前缀字符" suffixOverrides="忽略后缀字符">

SQL语句

</trim>

trim 中属性说明如下。

属性 描述

prefix 给SQL语句拼接的前缀，为 trim 包含的内容加上前缀

suffix 给SQL语句拼接的后缀，为 trim 包含的内容加上后缀

prefixOverrides 去除 SQL 语句前面的关键字或字符，该关键字或者字符由 prefixOverrides 属性指定。

suffixOverrides 去除 SQL 语句后面的关键字或者字符，该关键字或者字符由 suffixOverrides 属性指定。

**分页**

MyBatis 的分页功能是基于内存的分页，即先查询出所有记录，再按起始位置和页面容量取出结果。

修改 ...Mapper.xml 的查询语句，增加 limit 关键字，并增加了两个参数，起始位置（from）和页面容量（pageSize）来实现分页。

SQL 映射代码如下：

<select id="selectWebsite" resultType="net.biancheng.po.Website">

SELECT id,name,url,age,country

FROM website

<trim prefix="where" prefixOverrides="and">

<if test="site.name != null and site.name !=''">

AND name LIKE CONCAT ('%',#{site.name},'%')

</if>

<if test="site.url!= null and site.url !=''">

AND url LIKE CONCAT ('%',#{site.url},'%')

</if>

ORDER BY id limit #{from},#{pageSize}

</trim>

</select>