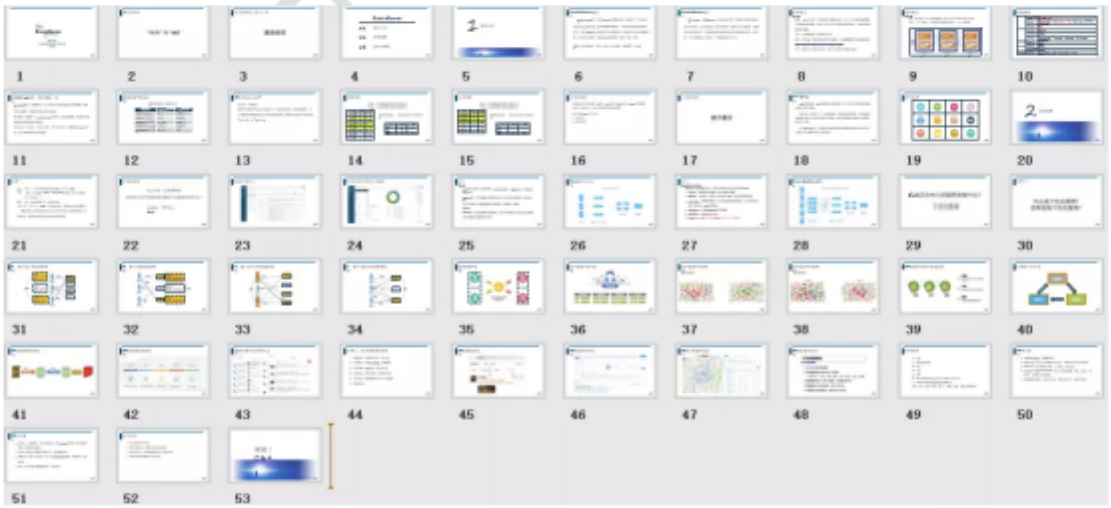
第二版：MyBatis 40 道

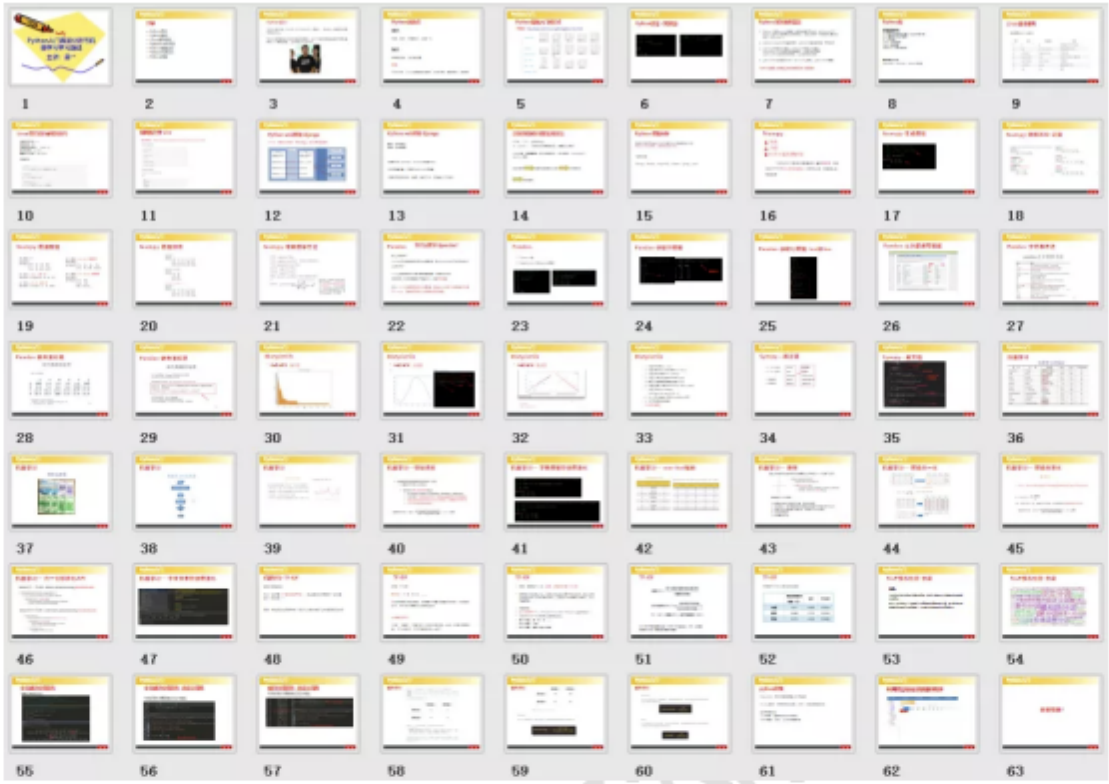
目录

46 页《JVM 深度调优，演讲 PPT》

53 页《Elasticsearch 调优演讲 PPT》



63 页《Python 数据分析入门 PPT》



MyBatis 是什么?

- Mybatis 是一个半 ORM（对象关系映射）框架，它内部封装了 JDBC，开发时只需要关注 SQL 语句本身，不需要花费精力去处理加载驱动、创建连接、创建 statement 等繁杂的过程。程序员直接编写原生态 sql，可以严格控制 sql 执行性能，灵活度高。
- MyBatis 可以使用 XML 或注解来配置和映射原生信息,将 POJO 映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。

Mybatis 优缺点

优点

与传统的数据库访问技术相比，ORM 有以下优点：

- 基于 SQL 语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL 写在 XML 里，解除 sql 与程序代码的耦合，便于统一管理；提供 XML 标签，支持编写动态 SQL 语句，并可重用
- 与 JDBC 相比，减少了 50%以上的代码量，消除了 JDBC 大量冗余的代码，不需要手动开关连接
- 很好的与各种数据库兼容（因为 MyBatis 使用 JDBC 来连接数据库，所以只要 JDBC 支持的数据库 MyBatis 都支持）
- 提供映射标签，支持对象与数据库的 ORM 字段关系映射；提供对象关系映射标签，支持对象关系组件维护
- 能够与 Spring 很好的集成

缺点

- SQL 语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写 SQL 语句的功底有一定要求
- SQL 语句依赖于数据库，导致数据库移植性差，不能随意更换数据库

Hibernate 和 MyBatis 的区别

相同点

- 都是对 jdbc 的封装，都是持久层的框架，都用于 dao 层的开发。

不同点

- 映射关系
- MyBatis 是一个半自动映射的框架，配置 Java 对象与 sql 语句执行结果的对应关系，多表关联关系配置简单
- Hibernate 是一个全表映射的框架，配置 Java 对象与数据库表的对应关系，多表关联关系配置复杂

SQL 优化和移植性

- Hibernate 对 SQL 语句封装，提供了日志、缓存、级联（级联比 MyBatis 强大）等特性，此外还提供 HQL（Hibernate Query Language）操作数据库，

数据库无关性支持好，但会多消耗性能。如果项目需要支持多种数据库，代码开发量少，但 SQL 语句优化困难。

- MyBatis 需要手动编写 SQL，支持动态 SQL、处理列表、动态生成表名、支持存储过程。开发工作量相对大些。直接使用 SQL 语句操作数据库，不支持数据库无关性，但 sql 语句优化容易。

|ORM 是什么

- ORM（Object Relational Mapping），对象关系映射，是一种为了解决关系型数据库数据与简单 Java 对象（POJO）的映射关系的技术。简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系型数据库中。

为什么说 Mybatis 是半自动 ORM 映射工具？它与全自动的区别在哪里？

- Hibernate 属于全自动 ORM 映射工具，使用 Hibernate 查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。
- 而 Mybatis 在查询关联对象或关联集合对象时，需要手动编写 sql 来完成，所以，称之为半自动 ORM 映射工具。

传统 JDBC 开发存在什么问题？

- 频繁创建数据库连接对象、释放，容易造成系统资源浪费，影响系统性能。可以使用连接池解决这个问题。但是使用 jdbc 需要自己实现连接池。
- sql 语句定义、参数设置、结果集处理存在硬编码。实际项目中 sql 语句变化的可能性较大，一旦发生变化，需要修改 java 代码，系统需要重新编译，重新发布。不好维护。
- 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。
- 结果集处理存在重复代码，处理麻烦。如果可以映射成 Java 对象会比较方便。

JDBC 编程有哪些不足之处，MyBatis 是如何解决的?

- 1、数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库连接池可解决此问题。
- 解决：在 mybatis-config.xml 中配置数据链接池，使用连接池管理数据库连接。
- 2、Sql 语句写在代码中造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变 java 代码。-
- 解决：将 Sql 语句配置在 XXXXmapper.xml 文件中与 java 代码分离。
- 3、向 sql 语句传参数麻烦，因为 sql 语句的 where 条件不一定，可能多也可能少，占位符需要和参数一一对应。
- 解决： Mybatis 自动将 java 对象映射至 sql 语句。
- 4、对结果集解析麻烦，sql 变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成 pojo 对象解析比较方便。
- 解决：Mybatis 自动将 sql 执行结果映射至 java 对象。

MyBatis 和 Hibernate 的适用场景?

- MyBatis 专注于 SQL 本身，是一个足够灵活的 DAO 层解决方案。
- 对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis 将是不错的选择。

开发难易程度和学习成本

- Hibernate 是重量级框架，学习使用门槛高，适合于需求相对稳定，中小型的项目，比如：办公自动化系统
- MyBatis 是轻量级框架，学习使用门槛低，适合于需求变化频繁，大型的项目，比如：互联网电子商务系统

总结

- MyBatis 是一个小巧、方便、高效、简单、直接、半自动化的持久层框架，
- Hibernate 是一个强大、方便、高效、复杂、间接、全自动化的持久层框架。

MyBatis 的架构

MyBatis 编程步骤是什么样的?

- 1、 创建 SqlSessionFactory

- 2、 通过 SqlSessionFactory 创建 SqlSession
- 3、 通过 sqlsession 执行数据库操作
- 4、 调用 session.commit()提交事务
- 5、 调用 session.close()关闭会话

请说说 MyBatis 的工作原理

- 在学习 MyBatis 程序之前，需要了解一下 MyBatis 工作原理，以便于理解程序。MyBatis 的工作原理如下图

1、 读取 MyBatis 配置文件：mybatis-config.xml 为 MyBatis 的全局配置文件，配置了 MyBatis 的运行环境等信息，例如数据库连接信息。

2、 加载映射文件。映射文件即 SQL 映射文件，该文件中配置了操作数据库的 SQL 语句，需要在 MyBatis 配置文件 mybatis-config.xml 中加载。mybatis-config.xml 文件可以加载多个映射文件，每个文件对应数据库中的一张表。

3、 构造会话工厂：通过 MyBatis 的环境等配置信息构建会话工厂 SqlSessionFactory。

4、 创建会话对象：由会话工厂创建 SqlSession 对象，该对象中包含了执行 SQL 语句的所有方法。

5、 Executor 执行器：MyBatis 底层定义了一个 Executor 接口来操作数据库，它将根据 SqlSession 传递的参数动态地生成需要执行的 SQL 语句，同时负责查询缓存的维护。

6、 MappedStatement 对象：在 Executor 接口的执行方法中有一个 MappedStatement 类型的参数，该参数是对映射信息的封装，用于存储要映射的 SQL 语句的 id、参数等信息。

7、 输入参数映射：输入参数类型可以是 Map、List 等集合类型，也可以是基本数据类型和 POJO 类型。输入参数映射过程类似于 JDBC 对 preparedStatement 对象设置参数的过程。

8、 输出结果映射：输出结果类型可以是 Map、 List 等集合类型，也可以是基本数据类型和 POJO 类型。输出结果映射过程类似于 JDBC 对结果集的解析过程。

MyBatis 的功能架构是怎样的

- 我们把 Mybatis 的功能架构分为三层：
- API 接口层：提供给外部使用的接口 API，开发人员通过这些本地 API 来操纵数据库。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理。
- 数据处理层：负责具体的 SQL 查找、SQL 解析、SQL 执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作。
- 基础支撑层：负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑。

MyBatis 的框架架构设计是怎么样的

- 这张图从上往下看。MyBatis 的初始化，会从 mybatis-config.xml 配置文件，解析构造成 Configuration 这个类，就是图中的红框。

1、 加载配置：配置来源于两个地方，一处是配置文件，一处是 Java 代码的注解，将 SQL 的配置信息加载成为一个个 MappedStatement 对象（包括了传入参数映射配置、执行的 SQL 语句、结果映射配置），存储在内存中。

2、 SQL 解析：当 API 接口层接收到调用请求时，会接收到传入 SQL 的 ID 和传入对象（可以是 Map、JavaBean 或者基本数据类型），Mybatis 会根据 SQL 的 ID 找到对应的 MappedStatement，然后根据传入参数对象对 MappedStatement 进行解析，解析后可以得到最终要执行的 SQL 语句和参数。

3、 SQL 执行：将最终得到的 SQL 和参数拿到数据库进行执行，得到操作数据库的结果。

4、 结果映射：将操作数据库的结果按照映射的配置进行转换，可以转换成 HashMap、JavaBean 或者基本数据类型，并将最终结果返回。

什么是 DBMS

- DBMS：数据库管理系统(database management system)是一种操纵和管理数据库的大型软件，用于建立、使用和维护数 zd 据库，简称 dbms。它对数据库进行统一的管理和控制，以保证数据库的安全性和完整性。用户通过 dbms 访问数据库中的数据，数据库管理员也通过 dbms 进行数据库的维护工作。它可使多个应用程序和用户用不同的方法在同时版或不同时刻去建立，修改和询问数据库。DBMS 提供数据定义语言 DDL（Data Definition Language）与数据操作语言 DML（Data Manipulation Language），供用户定义数据库的模式结构与权限约束，实现对数据的追加权、删除等操作。

为什么需要预编译

- 定义：

- SQL 预编译指的是数据库驱动在发送 SQL 语句和参数给 DBMS 之前对 SQL 语句进行编译，这样 DBMS 执行 SQL 时，就不需要重新编译。
- 为什么需要预编译
- JDBC 中使用对象 PreparedStatement 来抽象预编译语句，使用预编译。预编译阶段可以优化 SQL 的执行。预编译之后的 SQL 多数情况下可以直接执行，DBMS 不需要再次编译，越复杂的 SQL，编译的复杂度将越大，预编译阶段可以合并多次操作为一个操作。同时预编译语句对象可以重复利用。把一个 SQL 预编译后产生的 PreparedStatement 对象缓存下来，下次对于同一个 SQL，可以直接使用这个缓存的 PreparedState 对象。Mybatis 默认情况下，将对所有的 SQL 进行预编译。
- 还有一个重要的原因，复制 SQL 注入

Mybatis 都有哪些 Executor 执行器？它们之间的区别是什么？

- Mybatis 有三种基本的 Executor 执行器，SimpleExecutor、ReuseExecutor、BatchExecutor。
- **SimpleExecutor**：每执行一次 update 或 select，就开启一个 Statement 对象，用完立刻关闭 Statement 对象。
- **ReuseExecutor**：执行 update 或 select，以 sql 作为 key 查找 Statement 对象，存在就使用，不存在就创建，用完后，不关闭 Statement 对象，而是放置于 Map<String, Statement>内，供下一次使用。简言之，就是重复使用 Statement 对象。
- **BatchExecutor**：执行 update（没有 select，JDBC 批处理不支持 select），将所有 sql 都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个 Statement 对象，每个 Statement 对象都是 addBatch()完毕后，等待逐一执行 executeBatch()批处理。与 JDBC 批处理相同。

作用范围：Executor 的这些特点，都严格限制在 SqlSession 生命周期范围内。

Mybatis 中如何指定使用哪一种 Executor 执行器?

- 在 Mybatis 配置文件中，在设置（settings）可以指定默认的 ExecutorType 执行器类型，也可以手动给 DefaultSqlSessionFactory 的创建 SqlSession 的方法传递 ExecutorType 类型参数,如 SqlSession openSession(ExecutorType execType)。
- 配置默认的执行器。SIMPLE 就是普通的执行器；REUSE 执行器会重用预处理语句（prepared statements）； BATCH 执行器将重用语句并执行批量更新。

Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么?

- Mybatis 仅支持 association 关联对象和 collection 关联集合对象的延迟加载，association 指的就是一对一，collection 指的就是一对多查询。在 Mybatis 配置文件中，可以配置是否启用延迟加载 lazyLoadingEnabled=true|false。
- 它的原理是，使用 CGLIB 创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 a.getB().getName()，拦截器 invoke()方法发现 a.getB()是 null 值，那么就会单独发送事先保存好的查询关联 B 对象的 sql，把 B 查询上来，然后调用 a.setB(b)，于是 a 的对象 b 属性就有值了，接着完成 a.getB().getName()方法的调用。这就是延迟加载的基本原理。
- 当然了，不光是 Mybatis，几乎所有的包括 Hibernate，支持延迟加载的原理都是一样的。

映射器

#{}和${}的区别

- #{}是占位符，预编译处理；${}是拼接符，字符串替换，没有预编译处理。
- Mybatis 在处理#{}时，#{}传入参数是以字符串传入，会将 SQL 中的#{}替换为?号，调用 PreparedStatement 的 set 方法来赋值。
- #{} 可以有效的防止 SQL 注入，提高系统安全性；${} 不能防止 SQL 注入
- #{} 的变量替换是在 DBMS 中；${} 的变量替换是在 DBMS 外

模糊查询 like 语句该怎么写

- 1 ‘ %${question}%’ 可能引起 SQL 注入，不推荐
- 2 "%"#{question}"%" 注意：因为#{...}解析成 sql 语句时候，会在变量外侧自动加单引号‘ ’，所以这里 % 需要使用双引号" "，不能使用单引号 ‘ ’，不然会查不到任何结果。
- 3 CONCAT(‘ %’,#{question},’ %’) 使用 CONCAT()函数， （推荐）
- 4 使用 bind 标签（不推荐）

```
<select id="listUserLikeUsername"
resultType="com.jourwon.pojo.User">
    <bind name="pattern" value="'%' + username + '%'" />
    select id,sex,age,username,password from person where username
LIKE #{pattern}
```

```
</select>
```

在 mapper 中如何传递多个参数

方法 1：顺序传参法

```
public User selectUser(String name, int deptId);

<select id="selectUser" resultMap="UserResultMap">
select * from user
where user_name = #{0} and dept_id = #{1}
</select>
```

- #{}里面的数字代表传入参数的顺序。
- 这种方法不建议使用，sql 层表达不直观，且一旦顺序调整容易出错。

方法 2：@Param 注解传参法

```
public User selectUser(@Param("userName") String name, int
@Param("deptId") deptId);

<select id="selectUser" resultMap="UserResultMap">
select * from user
where user_name = #{userName} and dept_id = #{deptId}
</select>
```

- #{}里面的名称对应的是注解@Param 括号里面修饰的名称。
- 这种方法在参数不多的情况还是比较直观的，（推荐使用）。

方法 3：Map 传参法

```
public User selectUser(Map<String, Object> params);

<select id="selectUser" parameterType="java.util.Map"
resultMap="UserResultMap">
select * from user
where user_name = #{userName} and dept_id = #{deptId}
</select>
```

- #{}里面的名称对应的是 Map 里面的 key 名称。
- 这种方法适合传递多个参数，且参数易变能灵活传递的情况。（推荐使用）。

方法 4：Java Bean 传参法

```
public User selectUser(User user);

<select id="selectUser" parameterType="com.jourwon.pojo.User"
resultMap="UserResultMap">
select * from user
where user_name = #{userName} and dept_id = #{deptId}
</select>
```

- #{}里面的名称对应的是 User 类里面的成员属性。
- 这种方法直观，需要建一个实体类，扩展不容易，需要加属性，但代码可读性强，业务逻辑处理方便，推荐使用。（推荐使用）。

Mybatis 如何执行批量操作

- **使用 foreach 标签**
- foreach 的主要用在构建 in 条件中，它可以在 SQL 语句中进行迭代一个集合。foreach 标签的属性主要有 item，index，collection，open，separator，close。
- item　　表示集合中每一个元素进行迭代时的别名，随便起的变量名；
- index　　指定一个名字，用于表示在迭代过程中，每次迭代到的位置，不常用；
- open　　表示该语句以什么开始，常用“(”；
- separator　表示在每次进行迭代之间以什么符号作为分隔符，常用“,”；
- close　　表示以什么结束，常用“)”。
- 在使用 foreach 的时候最关键的也是最容易出错的就是 collection 属性，该属性是必须指定的，但是在不同情况下，该属性的值是不一样的，主要有一下 3 种情况：

1. 如果传入的是单参数且参数类型是一个 List 的时候，collection 属性值为 list
2. 如果传入的是单参数且参数类型是一个 array 数组的时候，collection 的属性值为 array
3. 如果传入的参数是多个的时候，我们就需要把它们封装成一个 Map 了，当然单参数也可以封装成 map，实际上如果你在传入参数的时候，在 MyBatis 里面也是会把它封装成一个 Map 的，
   map 的 key 就是参数名，所以这个时候 collection 属性值就是传入的 List 或 array 对象在自己封装的 map 里面的 key

- 具体用法如下：

```
<!-- 批量保存(foreach 插入多条数据两种方法)
    int addEmpsBatch(@Param("emps") List<Employee> emps); -->
<!-- MySQL 下批量保存,可以 foreach 遍历  mysql 支持 values(),(),()语法 -->
//推荐使用

<insert id="addEmpsBatch">
INSERT INTO emp(ename,gender,email,did)
VALUES
<foreach collection="emps" item="emp" separator=",">
    (#{emp.eName},#{emp.gender},#{emp.email},#{emp.dept.id})
</foreach>
</insert>
```

```
<!-- 这种方式需要数据库连接属性 allowMutiQueries=true 的支持
如jdbc.url=jdbc:mysql://localhost:3306/mybatis?allowMultiQueries=true
-->

<insert id="addEmpsBatch">
<foreach collection="emps" item="emp" separator=";">
    INSERT INTO emp(ename,gender,email,did)

VALUES(#{emp.eName},#{emp.gender},#{emp.email},#{emp.dept.id})
</foreach>
</insert>
```

- **使用 ExecutorType.BATCH**
- Mybatis 内置的 ExecutorType 有 3 种，默认为 simple,该模式下它为每个语句的执行创建一个新的预处理语句，单条提交 sql；而 batch 模式重复使用已经预处理的语句，并且批量执行所有更新语句，显然 batch 性能将更优； 但 batch

模式也有自己的问题，比如在 Insert 操作时，在事务没有提交之前，是没有办法获取到自增的 id，这在某型情形下是不符合业务要求的

- 具体用法如下：

```
    //批量保存方法测试
    @Test
    public void testBatch() throws IOException{
        SqlSessionFactory sqlSessionFactory = getSqlSessionFactory();
        //可以执行批量操作的 sqlSession
        SqlSession openSession =
sqlSessionFactory.openSession(ExecutorType.BATCH);

        //批量保存执行前时间
        long start = System.currentTimeMillis();
        try {
            EmployeeMapper mapper =
openSession.getMapper(EmployeeMapper.class);
            for (int i = 0; i < 1000; i++) {
                mapper.addEmp(new
Employee(UUID.randomUUID().toString().substring(0, 5), "b", "1"));
            }

            openSession.commit();
            long end = System.currentTimeMillis();
            //批量保存执行后的时间
            System.out.println("执行时长" + (end - start));
            //批量 预编译 sql 一次==》设置参数==》10000 次==》执行 1
次    677
            //非批量    (预编译=设置参数=执行 ) ==》10000 次    1121
```

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

- mapper 和 mapper.xml 如下

```
public interface EmployeeMapper {
    //批量保存员工
    Long addEmp(Employee employee);
}
```

```
<mapper namespace="com.jourwon.mapper.EmployeeMapper"
     <!--批量保存员工 -->
    <insert id="addEmp">
        insert into employee(lastName,email,gender)
        values(#{lastName},#{email},#{gender})
    </insert>
</mapper>
```

如何获取生成的主键

- 新增标签中添加：keyProperty=" ID " 即可

```
<insert id="insert" useGeneratedKeys="true" keyProperty="userId" >
    insert into user(
    user_name, user_password, create_time)
```

```
    values(#{userName}, #{userPassword} , #{createTime, jdbcType=
TIMESTAMP})
</insert>
```

当实体类中的属性名和表中的字段名不一样 ， 怎么办

- 第 1 种： 通过在查询的 SQL 语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。

```
<select id="getOrder" parameterType="int"
resultType="com.jourwon.pojo.Order">
        select order_id id, order_no orderno ,order_price price form
orders where order_id=#{id};
</select>
```

- 第 2 种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系。

```
<select id="getOrder" parameterType="int"
resultMap="orderResultMap">
    select * from orders where order_id=#{id}
</select>

<resultMap type="com.jourwon.pojo.Order" id="orderResultMap">
    <!–用 id 属性来映射主键字段–>
    <id property="id" column="order_id">
```

```
    <!–用 result 属性来映射非主键字段，property 为实体类属性名，column 为数据库表中的属性–>
    <result property ="orderno" column ="order_no"/>
    <result property="price" column="order_price" />
</reslutMap>
```

Mapper 编写有哪几种方式?

- 第一种：接口实现类继承 SqlSessionDaoSupport：使用此种方法需要编写 mapper 接口，mapper 接口实现类、mapper.xml 文件。

1. 在 sqlMapConfig.xml 中配置 mapper.xml 的位置

```
    <mappers>
        <mapper resource="mapper.xml 文件的地址" />
        <mapper resource="mapper.xml 文件的地址" />
    </mappers>
```

2. 定义 mapper 接口
3. 实现类集成 SqlSessionDaoSupport
   mapper 方法中可以 this.getSqlSession()进行数据增删改查。
4. spring 配置

```
    <bean id=" " class="mapper 接口的实现">
        <property name="sqlSessionFactory"
        ref="sqlSessionFactory"></property>
```

```
    </bean>
```

- 第二种：使用 org.mybatis.spring.mapper.MapperFactoryBean：

1. 在 sqlMapConfig.xml 中配置 mapper.xml 的位置，如果 mapper.xml 和 mappre 接口的名称相同且在同一个目录，这里可以不用配置
2. 定义 mapper 接口：

```
    <mappers>
        <mapper resource="mapper.xml 文件的地址" />
        <mapper resource="mapper.xml 文件的地址" />
    </mappers>
```

3. mapper.xml 中的 namespace 为 mapper 接口的地址
4. mapper 接口中的方法名和 mapper.xml 中的定义的 statement 的 id 保持一致
5. Spring 中定义

```
    <bean id=""
class="org.mybatis.spring.mapper.MapperFactoryBean">
        <property name="mapperInterface" value="mapper 接口地址"
/>
        <property name="sqlSessionFactory" ref="sqlSessionFactory"
/>
    </bean>
```

- 第三种：使用 mapper 扫描器：

1. mapper.xml 文件编写：
   mapper.xml 中的 namespace 为 mapper 接口的地址；
   mapper 接口中的方法名和 mapper.xml 中的定义的 statement 的 id 保持一致；
   如果将 mapper.xml 和 mapper 接口的名称保持一致则不用在 sqlMapConfig.xml 中进行配置。
2. 定义 mapper 接口：
   注意 mapper.xml 的文件名和 mapper 的接口名称保持一致，且放在同一个目录
3. 配置 mapper 扫描器：

```
    <bean
class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="mapper 接口包地址
        "></property>
        <property name="sqlSessionFactoryBeanName"
        value="sqlSessionFactory"/>
    </bean>
```

4. 使用扫描器后从 spring 容器中获取 mapper 的实现对象。

什么是 MyBatis 的接口绑定？有哪些实现方式？

- 接口绑定，就是在 MyBatis 中任意定义接口，然后把接口里面的方法和 SQL 语句绑定，我们直接调用接口方法就可以，这样比起原来了 SqlSession 提供的方法我们可以有更加灵活的选择和设置。
- 接口绑定有两种实现方式

1. 通过注解绑定，就是在接口的方法上面加上 @Select、@Update 等注解，里面包含 Sql 语句来绑定；
2. 通过 xml 里面写 SQL 来绑定， 在这种情况下，要指定 xml 映射文件里面的 namespace 必须为接口的全路径名。当 Sql 语句比较简单时候，用注解绑定，当 SQL 语句比较复杂时候，用 xml 绑定，一般用 xml 绑定的比较多。

使用 MyBatis 的 mapper 接口调用时有哪些要求?

1、 Mapper 接口方法名和 mapper.xml 中定义的每个 sql 的 id 相同。

2、 Mapper 接口方法的输入参数类型和 mapper.xml 中定义的每个 sql 的 parameterType 的类型相同。

3、 Mapper 接口方法的输出参数类型和 mapper.xml 中定义的每个 sql 的 resultType 的类型相同。

4、 Mapper.xml 文件中的 namespace 即是 mapper 接口的类路径。

这个 Dao 接口的工作原理是什么？Dao 接口里的方法，参数不同时，方法能重载吗

- Dao 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK 动态代理为 Dao 接口生成代理 proxy 对象，代理对象 proxy 会拦截接口方法，转而执行 MappedStatement 所代表的 sql，然后将 sql 执行结果返回。
- Dao 接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Mybatis 的 Xml 映射文件中，不同的 Xml 映射文件，id 是否可以重复?

- 不同的 Xml 映射文件，如果配置了 namespace，那么 id 可以重复；如果没有配置 namespace，那么 id 不能重复；毕竟 namespace 不是必须的，只是最佳实践而已。
- 原因就是 namespace+id 是作为 Map<String, MappedStatement>的 key 使用的，如果没有 namespace，就剩下 id，那么，id 重复会导致数据互相覆盖。有了 namespace，自然 id 就可以重复，namespace 不同，namespace+id 自然也就不同。

简述 Mybatis 的 Xml 映射文件和 Mybatis 内部数据结构之间的映射关系?

- 答：Mybatis 将所有 Xml 配置信息都封装到 All-In-One 重量级对象 Configuration 内部。在 Xml 映射文件中，<parameterMap>标签会被解析为 ParameterMap 对象，其每个子元素会被解析为 ParameterMapping 对象。<resultMap>标签会被解析为 ResultMap 对象，其每个子元素会被解析为 ResultMapping 对象。每一个<select>、<insert>、<update>、<delete>

标签均会被解析为 MappedStatement 对象，标签内的 sql 会被解析为 BoundSql 对象。

Mybatis 是如何将 sql 执行结果封装为目标对象并返回的？都有哪些映射形式?

- 第一种是使用<resultMap>标签，逐一定义列名和对象属性名之间的映射关系。
- 第二种是使用 sql 列的别名功能，将列别名书写为对象属性名，比如 T_NAME AS NAME，对象属性名一般是 name，小写，但是列名不区分大小写，Mybatis 会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成 T_NAME AS NaMe，Mybatis 一样可以正常工作。

有了列名与属性名的映射关系后，Mybatis 通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

Xml 映射文件中，除了常见的 select|insert|updae|delete 标签之外，还有哪些标签?

- 还有很多其他的标签，<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态 sql 的 9 个标签，trim|where|set|foreach|if|choose|when|otherwise|bind 等，其中<sql>为 sql 片段标签，通过<include>标签引入 sql 片段，<selectKey>为不支持自增的主键生成策略标签。

Mybatis 映射文件中，如果 A 标签通过 include 引用了 B 标签的内容，请问，B 标签能否定义在 A 标签的后面，还是说必须定义在 A 标签的前面?

- 虽然 Mybatis 解析 Xml 映射文件是按照顺序解析的，但是，被引用的 B 标签依然可以定义在任何地方，Mybatis 都可以正确识别。
- 原理是，Mybatis 解析 A 标签，发现 A 标签引用了 B 标签，但是 B 标签尚未解析到，尚不存在，此时，Mybatis 会将 A 标签标记为未解析状态，然后继续解析余下的标签，包含 B 标签，待所有标签解析完毕，Mybatis 会重新解析那些被标记为未解析的标签，此时再解析 A 标签时，B 标签已经存在，A 标签也就可以正常解析完成了。

Mybatis 能执行一对多，一对一的联系查询吗，有哪些实现方法

- 能，不止可以一对多，一对一还可以多对多，一对多
- 实现方式:

1. 单独发送一个 SQL 去查询关联对象，赋给主对象，然后返回主对象
2. 使用嵌套查询，似 JOIN 查询，一部分是 A 对象的属性值，另一部分是关联对 象 B 的属性值，好处是只要发送一个属性值，就可以把主对象和关联对象查出来
3. 子查询

Mybatis 是否可以映射 Enum 枚举类?

- Mybatis 可以映射枚举类，不单可以映射枚举类，Mybatis 可以映射任何对象到表的一列上。映射方式为自定义一个 TypeHandler，实现 TypeHandler 的 setParameter()和 getResult()接口方法。
- TypeHandler 有两个作用，一是完成从 javaType 至 jdbcType 的转换，二是完成 jdbcType 至 javaType 的转换，体现为 setParameter()和 getResult()两个方法，分别代表设置 sql 问号占位符参数和获取列查询结果。

Mybatis 动态 sql 是做什么的？都有哪些动态 sql？能简述一下动态 sql 的执行原理吗?

- Mybatis 动态 sql 可以让我们在 Xml 映射文件内，以标签的形式编写动态 sql，完成逻辑判断和动态拼接 sql 的功能，Mybatis 提供了 9 种动态 sql 标签 trim|where|set|foreach|if|choose|when|otherwise|bind。
- 其执行原理为，使用 OGNL 从 sql 参数对象中计算表达式的值，根据表达式的值动态拼接 sql，以此来完成动态 sql 的功能。

Mybatis 是如何进行分页的？分页插件的原理是什么?

- Mybatis 使用 RowBounds 对象进行分页，它是针对 ResultSet 结果集执行的内存分页，而非物理分页，可以在 sql 内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

- 分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql，根据 dialect 方言，添加对应的物理分页语句和物理分页参数。
- 举例：select * from student，拦截 sql 后重写为：select t.* from (select * from student) t limit 0, 10

简述 Mybatis 的插件运行原理，以及如何编写一个插件。

- Mybatis 仅可以编写针对 ParameterHandler、ResultSetHandler、StatementHandler、Executor 这 4 种接口的插件，Mybatis 使用 JDK 的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这 4 种接口对象的方法时，就会进入拦截方法，具体就是 InvocationHandler 的 invoke()方法，当然，只会拦截那些你指定需要拦截的方法。
- 实现 Mybatis 的 Interceptor 接口并复写 intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

Mybatis 的一级、二级缓存

1、 一级缓存：基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。

2、 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态),可在它的映射文件中配置 <cache/>

3、 对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存 Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。