

MyBatis 编程步骤

1. 创建 SqlSessionFactory 对象。
2. 通过 SqlSessionFactory 获取 SqlSession 对象。
3. 通过 SqlSession 获得 Mapper 代理对象。
4. 通过 Mapper 代理对象，执行数据库操作。
5. 执行成功，则使用 SqlSession 提交事务。
6. 执行失败，则使用 SqlSession 回滚事务。
7. 最终，关闭会话。

`#{}` 和`${}` 的区别是什么？

`${}` 是 Properties 文件中的变量占位符，它可以用于 XML 标签属性值和 SQL 内部，属于字符串替换。例如将`${driver}` 会被静态替换为`com.mysql.jdbc.Driver` ：

```
<dataSource type="UNPOOLED">
  <property name="driver" value="${driver}"/>
  <property name="url" value="${url}"/>
  <property name="username" value="${username}"/>
</dataSource>
```

`${}` 也可以对传递进来的参数原样拼接在 SQL 中。代码如下：

```
<select id="getSubject3" parameterType="Integer" resultType="Subject">
  SELECT * FROM subject
  WHERE id = ${id}
</select>
```

- 实际场景下，不推荐这么做。因为，可能有 SQL 注入的风险。

`#{}` 是 SQL 的参数占位符，Mybatis 会将 SQL 中的`#{}` 替换为`?` 号，在 SQL 执行前会使用 PreparedStatement 的参数设置方法，按序给 SQL 的`?` 号占位符设置参数值，比如`ps.setInt(0, parameterValue)` 。所以，`#{}` 是预编译处理，可以有效防止 SQL 注入，提高系统安全性。

另外，`#{}` 和`${}` 的取值方式非常方便。例如：`#{}` 的取值方式，为使用反射从参数对象中，获取`item` 对象的`name` 属性值，相当于`param.getItem().getName()` 。

当实体类中的属性名和表中的字段名不一样，怎么办？


第一种，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。代码如下：

```
<select id="selectOrder" parameterType="Integer" resultType="Order">
  SELECT order id AS id, order_no AS orderno, order_price AS price
  FROM orders
  WHERE order id = #{}
</select>
```

- 这里，芬芳还有几点建议：
 - 1、数据库的关键字，统一使用大写，例如：`SELECT`、`AS`、`FROM`、`WHERE`。
 - 2、每 5 个查询字段换一行，保持整齐。
 - 3、`,` 的后面，和`=` 的前后，需要有空格，更加清晰。
 - 4、`SELECT`、`FROM`、`WHERE` 等，单独一行，高端大气。

第二种，是第一种的特殊情况。大多数场景下，数据库字段名和实体类中的属性名差，主要是前者为下划线风格，后者为驼峰风格。在这种情况下，可以直接配置如下，实现自动的下划线转驼峰的功能。

```
<setting name="logImpl" value="LOG4J"/>
<setting name="mapUnderscoreToCamelCase" value="true" />
</settings>
```

 也就是说，约定大于配置。非常推荐！

第三种，通过`<resultMap>` 来映射字段名和实体类属性名的一一对应的关系。代码如下：

```
<resultMap type="me.gacl.domain.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 为数据表中的属性 -->
  <result property="orderNo" column="order_no" />
  <result property="price" column="order_price" />
</resultMap>

<select id="getOrder" parameterType="Integer" resultMap="OrderResultMap">
  SELECT *
  FROM orders
  WHERE order id = #{}
</select>
```

- 此处 `SELECT *` 仅仅作示例只用，实际场景下，千万千万千万不要这么干。用多少字段，查询多少字段。
- 相比第一种，第三种的重用性会一些。

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

如下部分，可见 [《MyBatis 文档 —— Mapper XML 文件》](#)：

- `<cache />` 标签，给定命名空间的缓存配置。
 - `<cache-ref />` 标签，其他命名空间缓存配置的引用。
- `<resultMap />` 标签，是最复杂也是最强大的元素，用来描述如何从数据库结果集中来加载对象。
- `<parameterMap />` 标签，已废弃！老式风格的参数映射。内联参数是首选，这个元素可能在将来被移除，这里不会记录。
- `<sql />` 标签，可被其他语句引用的可重用语句块。
 - `<include />` 标签，引用 `<sql />` 标签的语句。
- `<selectKey />` 标签，不支持自增的主键生成策略标签。

如下部分，可见 [《MyBatis 文档 —— 动态 SQL》](#)：

- `<if />`
- `<choose />`、`<when />`、`<otherwise />`
- `<trim />`、`<where />`、`<set />`
- `<foreach />`
- `<bind />`

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 XML 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：`<if />`、`<choose />`、`<when />`、`<otherwise />`、`<trim />`、`<where />`、`<set />`、`<foreach />`、`<bind />`。
- 其执行原理为，使用 **OGNL** 的表达式，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。

如上的内容，更加详细的话，请看 [《MyBatis 文档 —— 动态 SQL》](#) 文档。

最佳实践中，通常一个 XML 映射文件，都会写一个 **Mapper** 接口与之对应。请问，这个 **Mapper** 接口的工作原理是什么？**Mapper** 接口里的方法，参数不同时，方法能重载吗？

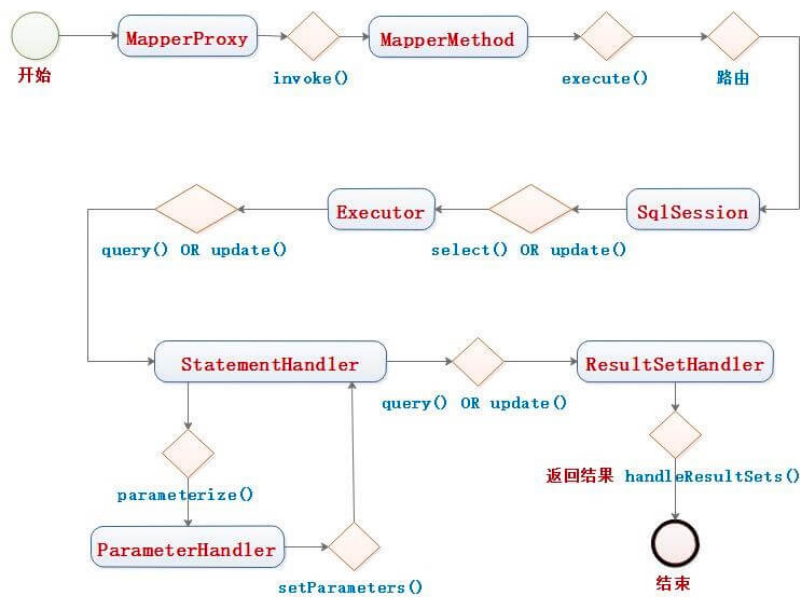
Mapper 接口，对应的关系如下：

- 接口的全限名，就是映射文件中的 `"namespace"` 的值。
- 接口的方法名，就是映射文件中 `MappedStatement` 的 `"id"` 值。
- 接口方法内的参数，就是传递给 SQL 的参数。

Mapper 接口是没有实现类的，当调用接口方法时，接口全限名 + 方法名拼接字符串作为 key 值，可唯一定位一个对应的 `MappedStatement`。举例：`com.mybatis3.mappers.StudentDao.findStudentById`，可以唯一找到 `"namespace"` 为 `com.mybatis3.mappers.StudentDao` 下面 `"id"` 为 `findStudentById` 的 `MappedStatement`。

总结来说，在 Mybatis 中，每一个 `<select />`、`<insert />`、`<update />`、`<delete />` 标签，都会被解析为一个 `MappedStatement` 对象。

另外，Mapper 接口的实现类，通过 MyBatis 使用 **JDK Proxy** 自动生成其代理对象 Proxy，而代理对象 Proxy 会拦截接口方法，从而“调用”对应的 `MappedStatement` 方法，最终执行 SQL，返回执行结果。整体流程如下图：



- 其中, SqlSession 在调用 Executor 之前, 会获得对应的 MappedStatement 方法。例如: `DefaultSqlSession#select(String statement, Object parameter, RowBounds rowBounds, ResultHandler handler)` 方法, 代码如下:

```

// DefaultSqlSession.java
@Override
public void select(String statement, Object parameter, RowBounds rowBounds,
    ResultHandler handler) {
    try {
        // 获得 MappedStatement 对象
        MappedStatement ms = configuration.getMappedStatement(statement);
        // 执行查询
        executor.query(ms, wrapCollection(parameter), rowBounds, handler);
    } catch (Exception e) {
        throw ExceptionFactory.wrapException("Error querying database. Cause: " + e,
            e);
    } finally {
        ErrorContext.instance().reset();
    }
}

```

- 完整的流程, 胖友可以慢慢撸下 MyBatis 的源码。

Mapper 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。🐼 所以有时, 想个 Mapper 接口里的方法名, 还是蛮闹心的, 嘿嘿。

Mapper 接口绑定有几种实现方式, 分别是怎么实现的?

接口绑定有三种实现方式:

第一种, 通过 **XML Mapper** 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。

第二种, 通过**注解**绑定, 就是在接口的方法上面加上 `@Select`、`@Update`、`@Insert`、`@Delete` 注解, 里面包含 SQL 语句来绑定。

第三种, 是第二种的特例, 也是通过**注解**绑定, 在接口的方法上面加上

`@SelectProvider`、`@UpdateProvider`、`@InsertProvider`、`@DeleteProvider` 注解, 通过 Java 代码, 生成对应的动态 SQL。

实际场景下, 最最最推荐的是**第一种**方式。因为, SQL 通过注解写在 Java 代码中, 会非常杂乱。而写在 XML 中, 更加有整体性, 并且可以更加方便的使用 OGNL 表达式。

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

不同的 XML Mapper 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。如果有了 "namespace"，自然 id 就可以重复，"namespace"不同，namespace + id 自然也就不同。

如何获取自动生成的(主)键值？

不同的数据库，获取自动生成的(主)键值的方式是不同的。

MySQL 有两种方式，但是**自增主键**，代码如下：

```
// 方式一，使用 useGeneratedKeys + keyProperty 属性
<insert id="insert" parameterType="Person" useGeneratedKeys="true"
keyProperty="id">
    INSERT INTO person(name, passwd)
    VALUE ({name}, #{passwd})
</insert>

// 方式二，使用 <selectKey /> 标签
<insert id="insert" parameterType="Person" useGeneratedKeys="true"
keyProperty="id">
    <selectKey keyProperty="id" resultType="long" order="AFTER">
        SELECT LAST_INSERT_ID()
    </selectKey>
    INSERT INTO person(name, passwd)
    VALUE ({name}, #{passwd})
</insert>
```

- 其中，方式一较为常用。

Oracle 有两种方式，**序列和触发器**。因为芳芳自己不了解 Oracle，所以问了银行的朋友，他们是使用**序列**。而基于**序列**，根据 `<selectKey />` 执行的时机，也有两种方式，代码如下：

```
// 这个是创建表的自增序列
CREATE SEQUENCE student_sequence
INCREMENT BY 1
NOMAXVALUE
NOCYCLE
CACHE 10;

// 方式一，使用 <selectKey /> 标签 + BEFORE
<insert id="add" parameterType="Student">
    <selectKey keyProperty="student_id" resultType="int" order="BEFORE">
        select student_sequence.nextval FROM dual
    </selectKey>
    INSERT INTO student(student_id, student_name, student_age)
    VALUES ({student_id},#{student_name},#{student_age})
</insert>

// 方式二，使用 <selectKey /> 标签 + AFTER
<insert id="save" parameterType="com.threeti.to.ZoneTO">
    <selectKey resultType="java.lang.Long" keyProperty="id" order="AFTER">
        SELECT SEQ_ZONE.CURRVAL AS id FROM dual
    </selectKey>
    INSERT INTO TBL_ZONE (ID, NAME )
    VALUES (SEQ_ZONE.NEXTVAL, #{name,jdbcType=VARCHAR})
</insert>
```

- 他们使用第一种方式，没有具体原因，可能就没什么讲究吧。嘿嘿。

至于为什么不用**触发器**呢？朋友描述如下：

朋友：触发器不行啊，我们这边原来也有触发器，一有数据更改就会有问题了呀

芳芳：数据更改指的是？

朋友：就改线上某几条数据

芳芳：噢噢。手动改是吧？

朋友：不行~

当然，数据库还有 SQLServer、PostgreSQL、DB2、H2 等等，具体的方式，胖友自己 Google 下噢。

关于如何获取自动生成的(主)键值的**原理**，可以看看 [《精尽 MyBatis 源码分析 —— SQL 执行（三）之 KeyGenerator》](#)。

Mybatis 执行批量插入，能返回数据库主键列表吗？

能，JDBC 都能做，Mybatis 当然也能做。

在 Mapper 中如何传递多个参数？

第一种，使用 Map 集合，装载多个参数进行传递。代码如下：

```
// 调用方法
Map<String, Object> map = new HashMap();
map.put("start", start);
map.put("end", end);
return studentMapper.selectStudents(map);

// Mapper 接口
List<Student> selectStudents(Map<String, Object> map);

// Mapper XML 代码
<select id="selectStudents" parameterType="Map" resultType="Student">
    SELECT *
    FROM students
    LIMIT #{start}, #{end}
</select>
```

- 显然，这不是一种优雅的方式。

第二种，保持传递多个参数，使用 `@Param` 注解。代码如下：

```
// 调用方法
return studentMapper.selectStudents(0, 10);

// Mapper 接口
List<Student> selectStudents(@Param("start") Integer start, @Param("end") Integer end);

// Mapper XML 代码
<select id="selectStudents" resultType="Student">
    SELECT *
    FROM students
    LIMIT #{start}, #{end}
</select>
```

- 推荐使用这种方式。

第三种，保持传递多个参数，不使用 `@Param` 注解。代码如下：

```
// 调用方法
return studentMapper.selectStudents(0, 10);

// Mapper 接口
List<Student> selectStudents(Integer start, Integer end);

// Mapper XML 代码
<select id="selectStudents" resultType="Student">
    SELECT *
    FROM students
    LIMIT #{param1}, #{param2}
</select>
```

- 其中，按照参数在方法方法中的位置，从 1 开始，逐个为 `#{param1}`、`#{param2}`、`#{param3}` 不断向下。

Mybatis 是否可以映射 Enum 枚举类？

Mybatis 可以映射枚举类，对应的实现类为 `EnumTypeHandler` 或 `EnumOrdinalTypeHandler`。

- `EnumTypeHandler`，基于 `Enum.name` 属性(`String`)。默认。
- `EnumOrdinalTypeHandler`，基于 `Enum.ordinal` 属性(`int`)。可通过 `<setting name="defaultEnumTypeHandler" value="EnumOrdinalTypeHandler" />` 来设置。

🐿 当然，实际开发场景，我们很少使用 Enum 类型，更加的方式是，代码如下：

```
public class Dog {
    public static final int STATUS_GOOD = 1;
    public static final int STATUS_BETTER = 2;
    public static final int STATUS_BEST = 3;
    private int status;
}
```

并且，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` 类，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。

具体体现为 `#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。

关于 `TypeHandler` 的原理，可以看看 [《精尽 MyBatis 源码分析 —— 类型模块》](#)。

Mybatis 都有哪些 `Executor` 执行器？它们之间的区别是什么？

Mybatis 有四种 Executor 执行器，分别是 SimpleExecutor、ReuseExecutor、BatchExecutor、CachingExecutor。

- SimpleExecutor：每执行一次 update 或 select 操作，就创建一个 Statement 对象，用完立刻关闭 Statement 对象。
- ReuseExecutor：执行 update 或 select 操作，以 SQL 作为key 查找缓存的 Statement 对象，存在就使用，不存在就创建；用完，不关闭 Statement 对象，而是放置于缓存 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 操作（没有 select 操作，因为 JDBC 批处理不支持 select 操作），将所有 SQL 都添加到批处理中（通过 addBatch 方法），等待统一执行（使用 executeBatch 方法）。它缓存了多个 Statement 对象，每个 Statement 对象都是调用 addBatch 方法完毕后，等待一次执行 executeBatch 批处理。实际上，整个过程与 JDBC 批处理是相同。
- CachingExecutor：在上述的三个执行器之上，增加二级缓存的功能。

通过设置 <setting name="defaultExecutorType" value=""> 的 "value" 属性，可传入 SIMPLE、REUSE、BATCH 三个值，分别使用 SimpleExecutor、ReuseExecutor、BatchExecutor 执行器。

通过设置 <setting name="cacheEnabled" value=""> 的 "value" 属性为 true 时，创建 CachingExecutor 执行器。

这块的源码解析，可见 [《精尽 MyBatis 源码分析 —— SQL 执行（一）之 Executor》](#)。

MyBatis 如何执行批量插入？

首先，在 Mapper XML 编写一个简单的 Insert 语句。代码如下：

```
<insert id="insertUser" parameterType="String">
    INSERT INTO users(name)
    VALUES (#{value})
</insert>
```

然后，然后在对应的 Mapper 接口中，编写映射的方法。代码如下：

```
public interface UserMapper {
    void insertUser(@Param("name") String name);
}
```

最后，调用该 Mapper 接口方法。代码如下：

```
private static SqlSessionFactory sqlSessionFactory;

@Test
public void testBatch() {
    // 创建要插入的用户的名字的数组
    List<String> names = new ArrayList<>();
    names.add("占小狼");
    names.add("朱小厮");
    names.add("徐妈");
    names.add("飞哥");

    // 获得执行器类型为 Batch 的 SqlSession 对象，并且 autoCommit = false，禁止事务自动提交
    try (SqlSession session = sqlSessionFactory.openSession(ExecutorType.BATCH, false)) {
        // 获得 Mapper 对象
        UserMapper mapper = session.getMapper(UserMapper.class);

        // 循环插入
        for (String name : names) {
            mapper.insertUser(name);
        }

        // 提交批量操作
        session.commit();
    }
}
```

代码比较简单，胖友仔细看看。当然，还有另一种方式，代码如下：

```
INSERT INTO [表名] ([列名],[列名])
VALUES
([列值],[列值]),
([列值],[列值]),
([列值],[列值]);
```

- 对于这种方式，需要保证单条 SQL 不超过语句的最大限制 max_allowed_packet 大小，默认为 1 M。

这两种方式的性能对比，可以看看 [《\[实验\]mybatis批量插入方式的比较》](#)。

介绍 MyBatis 的一级缓存和二级缓存的概念和实现原理？

内容有些长，直接参见 [《聊聊 MyBatis 缓存机制》](#) 一文。

这块的源码解析，可见 [《精尽 MyBatis 源码分析 —— 缓存模块》](#)。

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。其中，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 `<setting name="lazyLoadingEnabled" value="true" />` 来启用延迟加载的功能。默认情况下，延迟加载的功能是关闭的。

它的原理是，使用 CGLIB 或 Javassist(默认) 创建目标对象的代理对象。当调用代理对象的延迟加载属性的 getting 方法时，进入拦截器方法。比如调用 `a.getB().getName()` 方法，进入拦截器的 `invoke(...)` 方法，发现 `a.getB()` 需要延迟加载时，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)` 方法，于是 a 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 在内，支持延迟加载的原理都是一样的。

这块的源码解析，可见《[精尽 MyBatis 源码分析 —— SQL 执行（五）之延迟加载](#)》文章。

Mybatis 能否执行一对一、一对多的关联查询吗？都有哪些实现方式，以及它们之间的区别。

芬芳：这道题有点难度。理解倒是好理解，主要那块源码的实现，芬芳看的有点懵逼。大体的意思是懂的，但是一些细节没扣完。

能，Mybatis 不仅可以执行一对一、一对多的关联查询，还可以执行多对一，多对多的关联查询。

芬芳：不过貌似，我自己实际开发中，还是比较喜欢自己去查询和拼接映射的数据。

- 多对一查询，其实就是一对一查询，只需要把 `selectOne(...)` 修改为 `selectList(...)` 即可。案例可见《[MyBatis: 多对一表关系详解](#)》。
- 多对多查询，其实就是一对多查询，只需要把 `#selectOne(...)` 修改为 `selectList(...)` 即可。案例可见《[【MyBatis学习10】高级映射之多对多查询](#)》。

关联对象查询，有两种实现方式：

芬芳：所有的技术方案，即会有好处，又会有坏处。很难出现，一个完美的银弹方案。

- 一种是单独发送一个 SQL 去查询关联对象，赋给主对象，然后返回主对象。好处是多条 SQL 分开，相对简单，坏处是发起的 SQL 可能会比较多。
- 另一种是使用嵌套查询，嵌套查询的含义为使用 `join` 查询，一部分列是 A 对象的属性值，另外一部分列是关联对象 B 的属性值。好处是只发一个 SQL 查询，就可以把主对象和其关联对象查出来，坏处是 SQL 可能比较复杂。

那么问题来了，`join` 查询出来 100 条记录，如何确定主对象是 5 个，而不是 100 个呢？其去重复的原理是 `<resultMap>` 标签内的 `<id>` 子标签，指定了唯一确定一条记录的 `id` 列。Mybatis 会根据 `<id>` 列值来完成 100 条记录的去重复功能，`<id>` 可以有多个，代表了联合主键的语义。

同样主对象的关联对象，也是根据这个原理去重复的。尽管一般情况下，只有主对象会有重复记录，关联对象一般不会重复。例如：下面 `join` 查询出来6条记录，一、二列是 Teacher 对象列，第三列为 Student 对象列。Mybatis 去重复处理后，结果为 1 个老师和 6 个学生，而不是 6 个老师和 6 个学生。

t_id	t_name	s_id
1	teacher	38
1	teacher	39
1	teacher	40
1	teacher	41
1	teacher	42
1	teacher	43

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。

编写一个 MyBatis 插件的步骤如下：

1. 首先，实现 Mybatis 的 Interceptor 接口，并实现 `#intercept(...)` 方法。

2. 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
3. 最后，在配置文件中配置你编写的插件。

具体的，可以参考《[MyBatis 官方文档 —— 插件](#)》。

插件的详细解析，可以看看《[精尽 MyBatis 源码分析 —— 插件体系（一）之原理](#)》。

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，而非数据库分页。

所以，实际场景下，不适合直接使用 MyBatis 原有的 RowBounds 对象进行分页。而是使用如下两种方案：

- 在 SQL 内直接书写带有数据库分页的参数来完成数据库分页功能
- 也可以使用分页插件来完成数据库分页。

这两者都是基于数据库分页，差别在于前者是工程师手动编写分页条件，后者是插件自动添加分页条件。

分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义分页插件。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 * FROM student LIMIT 0, 10`。

目前市面上目前使用比较广泛的 MyBatis 分页插件有：

- [Mybatis-PageHelper](#)
- [MyBatis-Plus](#)

从现在看来，[MyBatis-Plus](#) 逐步使用的更加广泛。

关于 MyBatis 分页插件的原理深入，可以看看《[精尽 MyBatis 源码分析 —— 插件体系（二）之 PageHelper](#)》。

MyBatis 与 Hibernate 有哪些不同？

Mybatis 和 Hibernate 不同，它不完全是个 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。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。如果用 Hibernate 开发可以节省很多代码，提高效率。但是 Hibernate 的缺点是学习门槛高，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。

总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。简单总结如下：

- 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。
- Mybatis 属于半自动 ORM 映射工具，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。

另外，在《[浅析 Mybatis 与 Hibernate 的区别与用途](#)》文章，也是写的非常不错的。

当然，实际上，MyBatis 也可以搭配自动生成代码的工具，提升开发效率，还可以使用 [MyBatis-Plus](#) 框架，已经内置常用的 SQL 操作，也是非常不错的。

JDBC 编程有哪些不足之处，MyBatis 是如何解决这些问题的？

问题一：SQL 语句写在代码中造成代码不易维护，且代码会比较混乱。

解决方式：将 SQL 语句配置在 Mapper XML 文件中，与 Java 代码分离。

问题二：根据参数不同，拼接不同的 SQL 语句非常麻烦。例如 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。

解决方式：MyBatis 提供 `<where />`、`<if />` 等等动态语句所需要的标签，并支持 OGNL 表达式，简化了动态 SQL 拼接的代码，提升了开发效率。

问题三，对结果集解析麻烦，SQL 变化可能导致解析代码变化，且解析前需要遍历。

解决方式：Mybatis 自动将 SQL 执行结果映射成 Java 对象。

问题四，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，如果使用数据库链接池可解决此问题。

解决方式：在 `mybatis-config.xml` 中，配置数据库链接池，使用连接池管理数据库链接。

 当然，即使不使用 MyBatis，也可以使用数据库连接池。

另外，MyBatis 默认提供了数据库连接池的实现，只是说，因为其它开源的数据库连接池性能更好，所以一般很少使用

MyBatis 自带的连接池实现。

Mybatis 比 IBatis 比较大的几个改进是什么？

这是一个选择性了解的问题，因为可能现在很多面试官，都没用过 IBatis 框架。

1. 有接口绑定，包括注解绑定 SQL 和 XML 绑定 SQL。
2. 动态 SQL 由原来的节点配置变成 OGNL 表达式。
3. 在一对一或一对多的时候，引入了 `association`，在一对多的时候，引入了 `collection` 节点，不过都是在 `<resultMap />` 里面配置。

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？

老芬芳：这道题目，已经和源码实现，有点关系了。

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。也就是说，无需按照顺序，进行定义。

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。

可能有一些绕，胖友可以看看 [《精尽 MyBatis 源码解析 —— MyBatis 初始化（一）之加载 mybatis-config》](#)。

此处，我们在引申一个问题，Spring IOC 中，存在互相依赖的 Bean 对象，该如何解决呢？答案见 [《【死磕 Spring】—— IoC 之加载 Bean：创建 Bean（五）之循环依赖处理》](#)。

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

老芬芳：这道题目，已经和源码实现，有点关系了。

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 Mapper 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，均会被解析为一个 MappedStatement 对象，标签内的 SQL 会被解析为一个 BoundSql 对象。

666. 彩蛋

参考与推荐如下文章：

- 祖大俊 [《Mybatis3.4.x技术内幕（二十三）：Mybatis面试问题集锦（大结局）》](#)
- Java3y [《Mybatis 常见面试题》](#)
- Homiss [《MyBatis 面试题》](#)