1.MyBatis简介

1.1.MyBatis是什么?

MyBatis 是一款优秀的持久层框架，一个半 ORM（对象关系映射）框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 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 可以使用简单的 XML 或注解来配置和映射原生类型、接口和 Java 的 POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

1.2.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。

1.3.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。

1.4.传统JDBC开发存在的问题

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。

1.5.JDBC编程有哪些不足之处，MyBatis是如何解决这些问题的?

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。

1.6.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：

```
1    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
2
3    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:

```
1    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
2
3    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
```

1.7.MyBatis框架适用场景

```
1    MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。
2
3    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。
```

1.8.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。
```

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

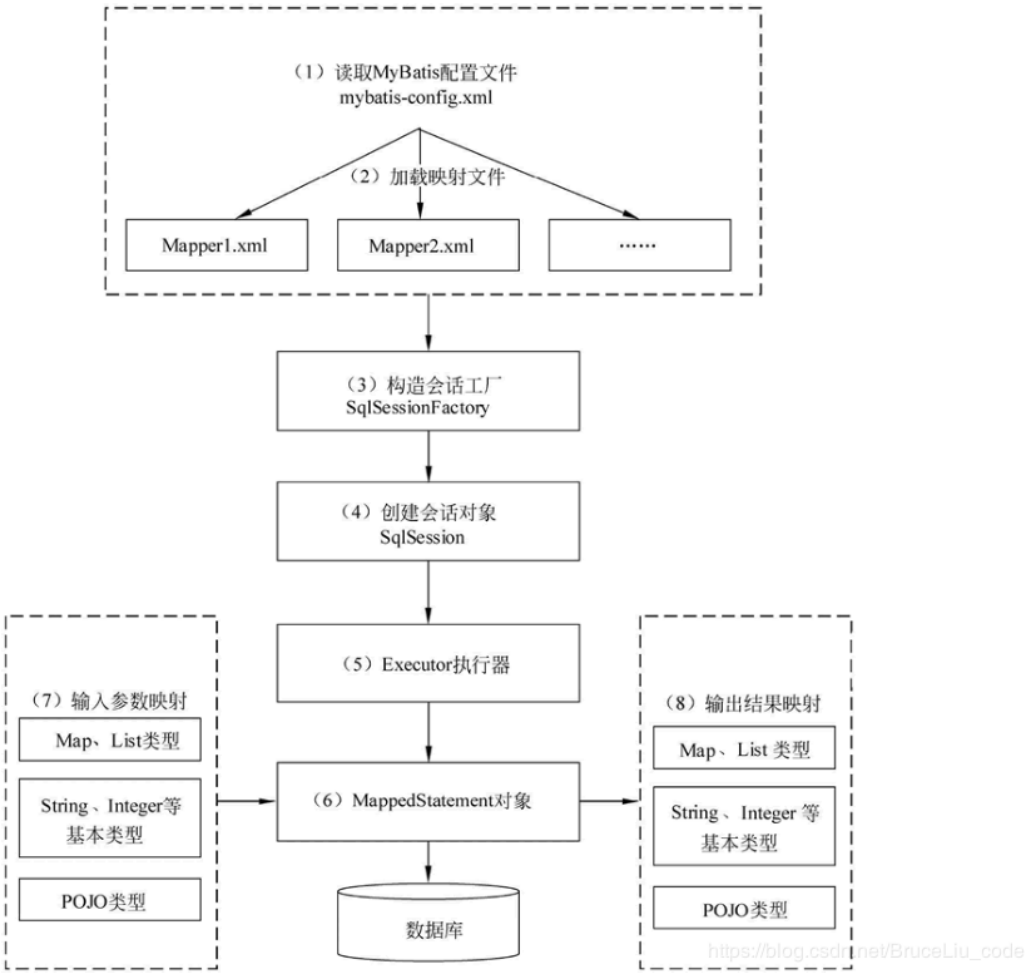
> MyBatis 是一个小巧、方便、高效、简单、直接、半自动化的持久层框架，
> Hibernate 是一个强大、方便、高效、复杂、间接、全自动化的持久层框架。

2.MyBatis的解析和运行原理

2.1.MyBatis编程步骤是什么样的?

```
创建SqlSessionFactory
通过SqlSessionFactory创建SqlSession
通过sqlsession执行数据库操作
调用session.commit()提交事务
调用session.close()关闭会话
```

2.2.请说说MyBatis的工作原理



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。

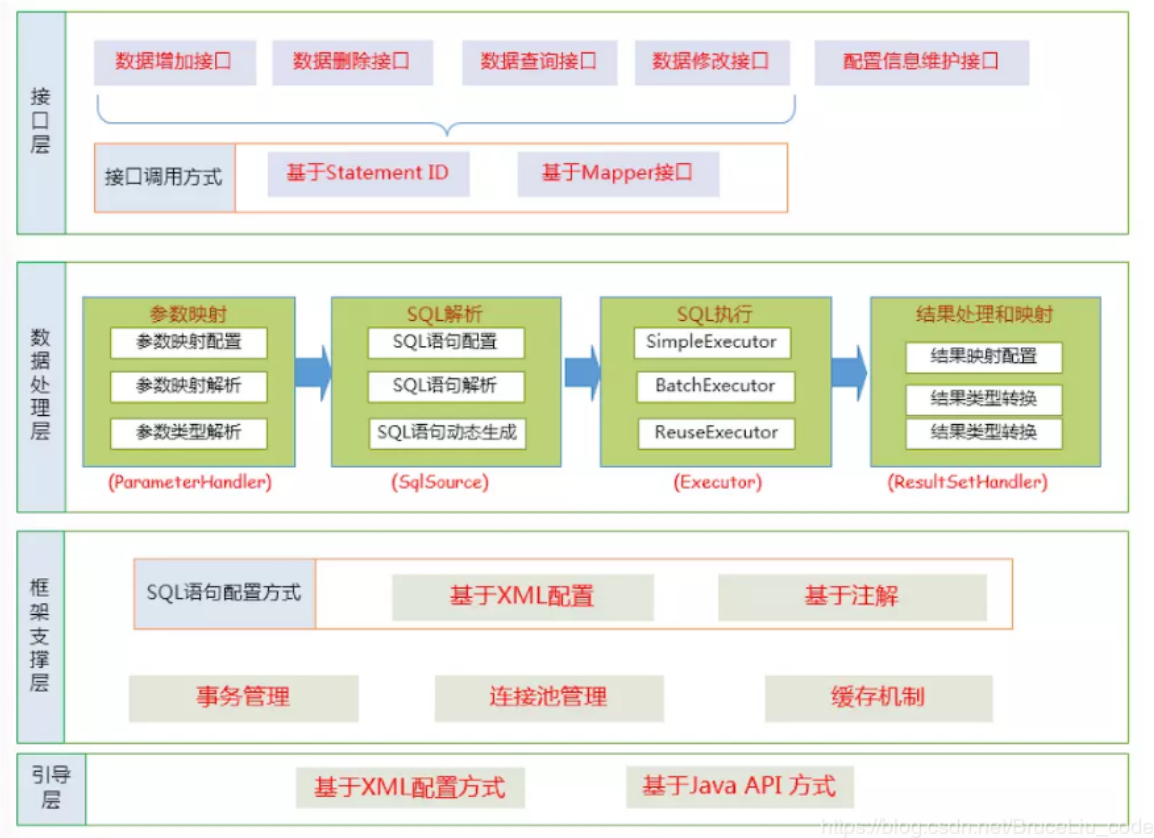
5）Executor 执行器：MyBatis 底层定义了一个 Executor 接口来操作数据库，它将根据 SqlSession 传递的参数动态地生成需要执行的 SQL 语句，同时负责查询缓存的维护。

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。

2.3.MyBatis的功能架构是怎样的



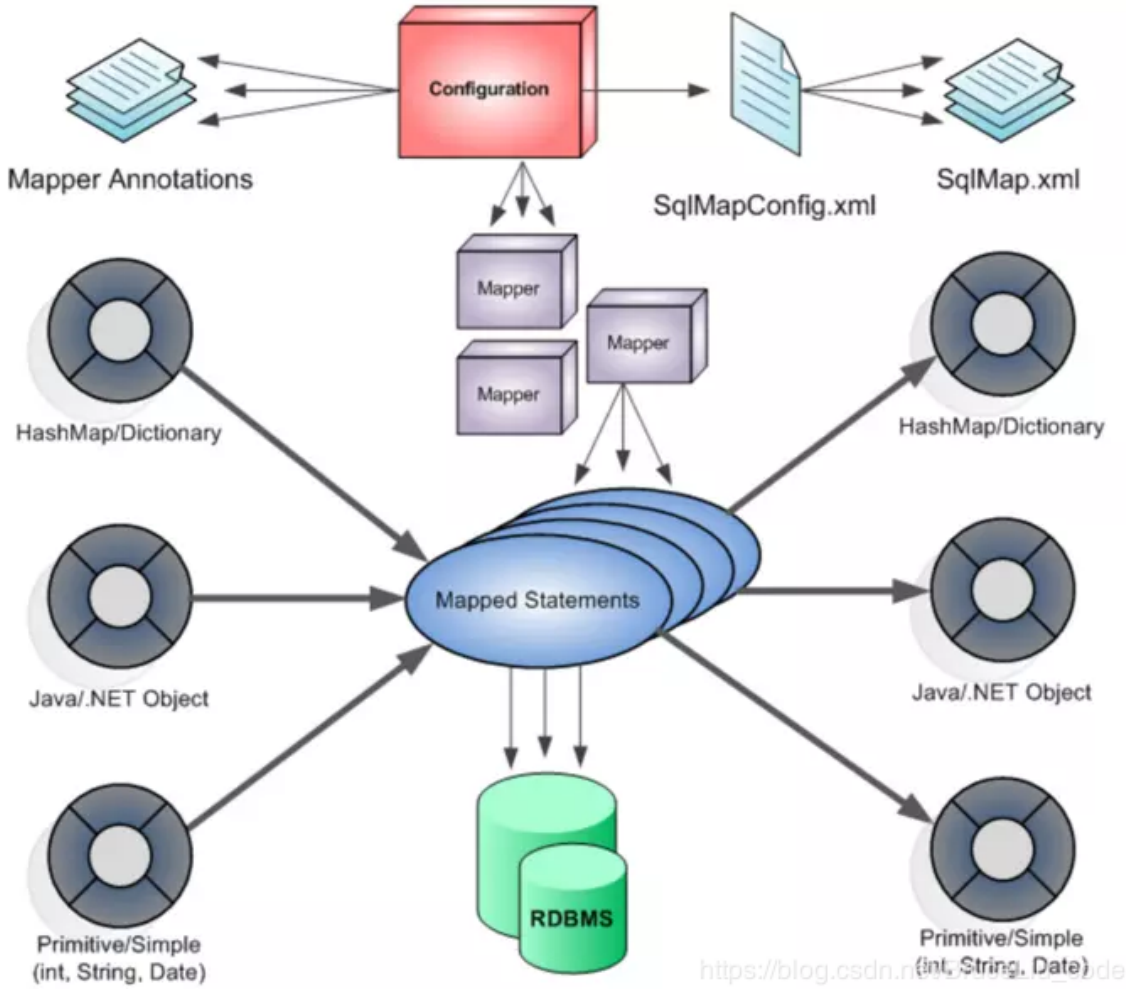
我们把Mybatis的功能架构分为三层：

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。

2.4.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。

2.5.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。

2.6.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。

2.7.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。

> 默认值可以通过`<setting name="defaultExecutorType" value="REUSE"/>`配置，不配置则为SIMPLE，或者编写代码的时候通过`org.apache.ibatis.session.SqlSessionFactory#openSession(org.apache.ibatis.session.ExecutorType)`指定类型。

2.8.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。

3.映射器

3.1.#{}和${}的区别

1）#{}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。

2）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。

3）Mybatis在处理时，是原值传入，就是把 {}时，是原值传入，就是把时，是原值传入，就是把{}替换成变量的值，相当于JDBC中的Statement编译

4）变量替换后，#{} 对应的变量自动加上单引号 ‘’；变量替换后，${} 对应的变量不会加上单引号 ‘’

5）#{} 可以有效的防止SQL注入，提高系统安全性；${} 不能防止SQL 注入

6）#{} 的变量替换是在DBMS中；${} 的变量替换是在 DBMS 外

3.2.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

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

3.3.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。

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。**

3.4.Mybatis如何执行批量操作

3.4.1.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“)”。
```

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>
```

```xml
<!-- 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

```java
//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>
```

3.5.如何获取生成的主键

对于支持主键自增的数据库（MySQL）

```
<insert id="insertUser" useGeneratedKeys="true"
keyProperty="userId" >
    insert into user(
    user_name, user_password, create_time)
    values(#{userName}, #{userPassword} , #{createTime, jdbcType=
TIMESTAMP})
</insert>
```

> parameterType 可以不写，Mybatis可以推断出传入的数据类型。如果想要访问主键，那么应当parameterType 应当是java实体或者Map。这样数据在插入之后可以通过ava实体或者Map 来获取主键值。通过 getUserId获取主键

不支持主键自增的数据库（Oracle）

对于像Oracle这样的数据，没有提供主键自增的功能，而是使用序列的方式获取自增主键。可以使用 `<selectKey>` 标签来获取主键的值，这种方式不仅适用于不提供主键自增功能的数据库，也适用于提供主键自增功能的数据库 `<selectKey>` 一般的用法.

```
<selectKey keyColumn="id" resultType="long" keyProperty="id"
order="BEFORE">
</selectKey>
```

属性	描述
keyProperty	selectKey 语句结果应该被设置的目标属性。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
keyColumn	匹配属性的返回结果集中的列名称。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表
resultType	结果的类型，MyBatis 通常可以推算出来。MyBatis 允许任何简单类型用作主键的类型，包括字符串。如果希望作用于多个生成的列，则可以使用一个包含期望属性的 Object 或一个 Map。
order	值可为BEFORE 或 AFTER。如果是 BEFORE，那么它会先执行 selectKey设置 keyProperty 然后执行插入语句。如果为AFTER 则相反。
statementType	使用何种语句类型，默认PREPARED。 有STATEMENT，PREPARED 和 CALLABLE 语句的映射类型。分别代表 PreparedStatement 和CallableStatement 类型。

```
<insert id="insertUser" >
    <selectKey keyColumn="id" resultType="long" keyProperty="userId" order="BEFORE">
        SELECT USER_ID.nextval as id from dual
    </selectKey>
    insert into user(
    user_id,user_name, user_password, create_time)
    values(#{userId},#{userName}, #{userPassword} , #{createTime, jdbcType= TIMESTAMP})
</insert>
```

此时会将Oracle生成的主键值赋予userId变量。这个userId 就是USER对象的属性，这样就可以将生成的主键值返回了。如果仅仅是在insert语句中使用但是不返回，此时keyProperty=“任意自定义变量名”，resultType 可以不写。

Oracle 数据库中的值要设置为 BEFORE ， 这是因为 Oracle中需要先从序列获取值，然后将值作为主键插入到数据库中。

> 扩展 如果Mysql 使用selectKey的方式获取主键，需要注意下面两点： order： AFTER 获取递增主键值： SELECT LAST_INSERT_ID()

3.6.如何获取生成的主键

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.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.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>
```

3.7.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

```
<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）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。

3.8.什么是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。

3.9.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。
```

3.10.最佳实践中，通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，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，就是人们常说的Mapper接口，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MappedStatement，举例：`com.mybatis3.mappers.StudentDao.findStudentById`，可以唯一找到namespace为`com.mybatis3.mappers.StudentDao`下面`id = findStudentById`的`MappedStatement`。在Mybatis中，每一个`<select>`、`<insert>`、`<update>`、`<delete>`标签，都会被解析为一个MappedStatement对象。

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。

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。

3.11.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。

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

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。

3.13.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。

3.14.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。

3.15.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。

4.高级查询

4.1.MyBatis实现一对一，一对多有几种方式，怎么操作的?

有联合查询和嵌套查询。联合查询是几个表联合查询，只查询一次，通过在resultMap里面的association，collection节点配置一对一，一对多的类就可以完成

嵌套查询是先查一个表，根据这个表里面的结果的外键id，去再另外一个表里面查询数据，也是通过配置association，collection，但另外一个表的查询通过select节点配置。

4.2.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。

5.动态SQL

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

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。

6.插件模块

6.1.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`

6.2.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。

7.缓存

7.1.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。