Mybatis考点

1. 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。

2. 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，影响性能。可以使用连接池解决这个问题。但是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，需要修改代码重新编译部署。不好维护。

- 使用preparedStatement向占有位符号传参数存在硬编码，修改sql语句需要修改代码，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。

3. JDBC编程有哪些不足之处，MyBatis是如何解决的?

- ✅ 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。
- ✅ 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。
- ✅ 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。
- ✅ 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。

4. MyBatis编程步骤?

1）创建SqlSessionFactory 。

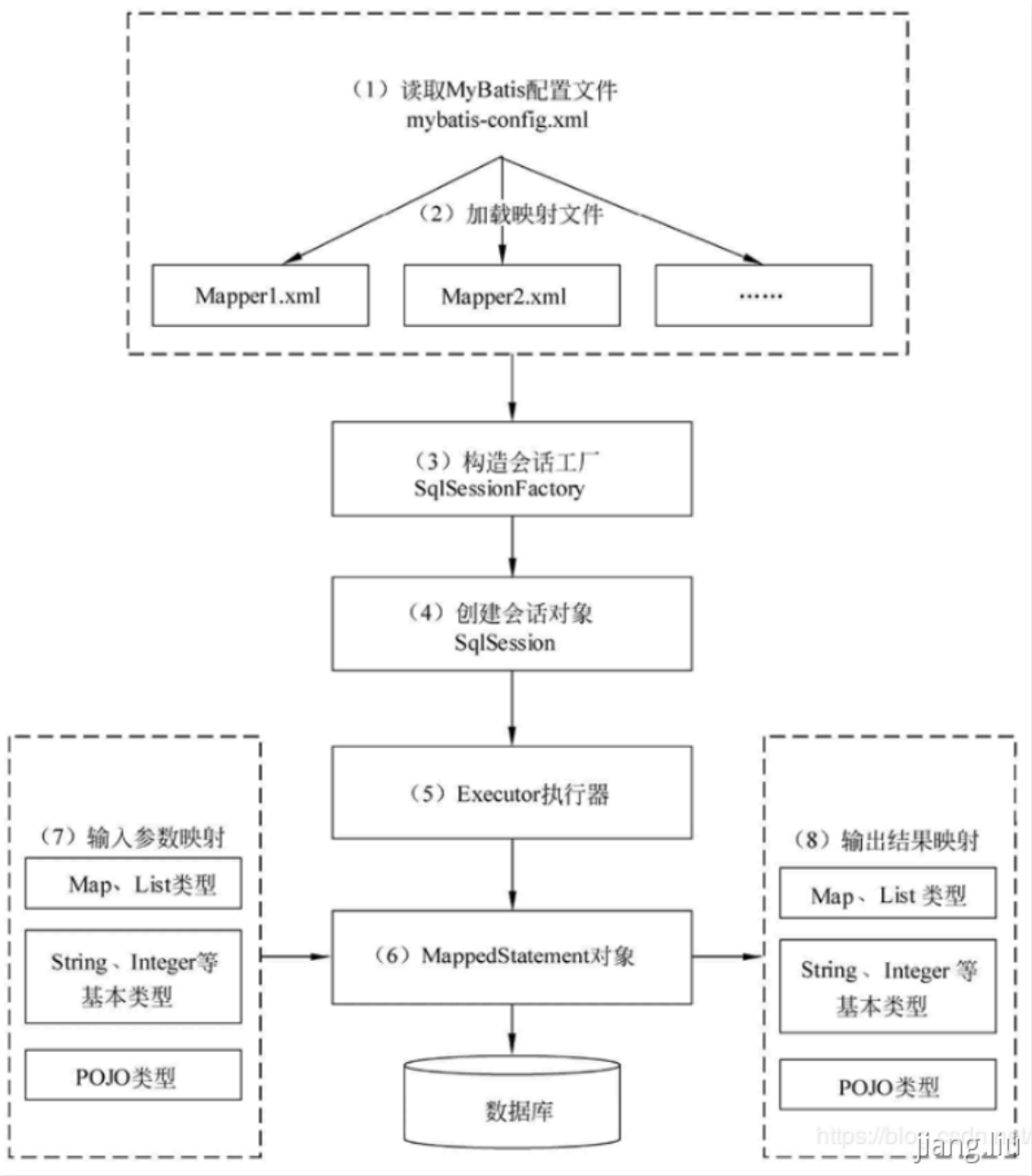
2）通过SqlSessionFactory创建SqlSession 。

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。

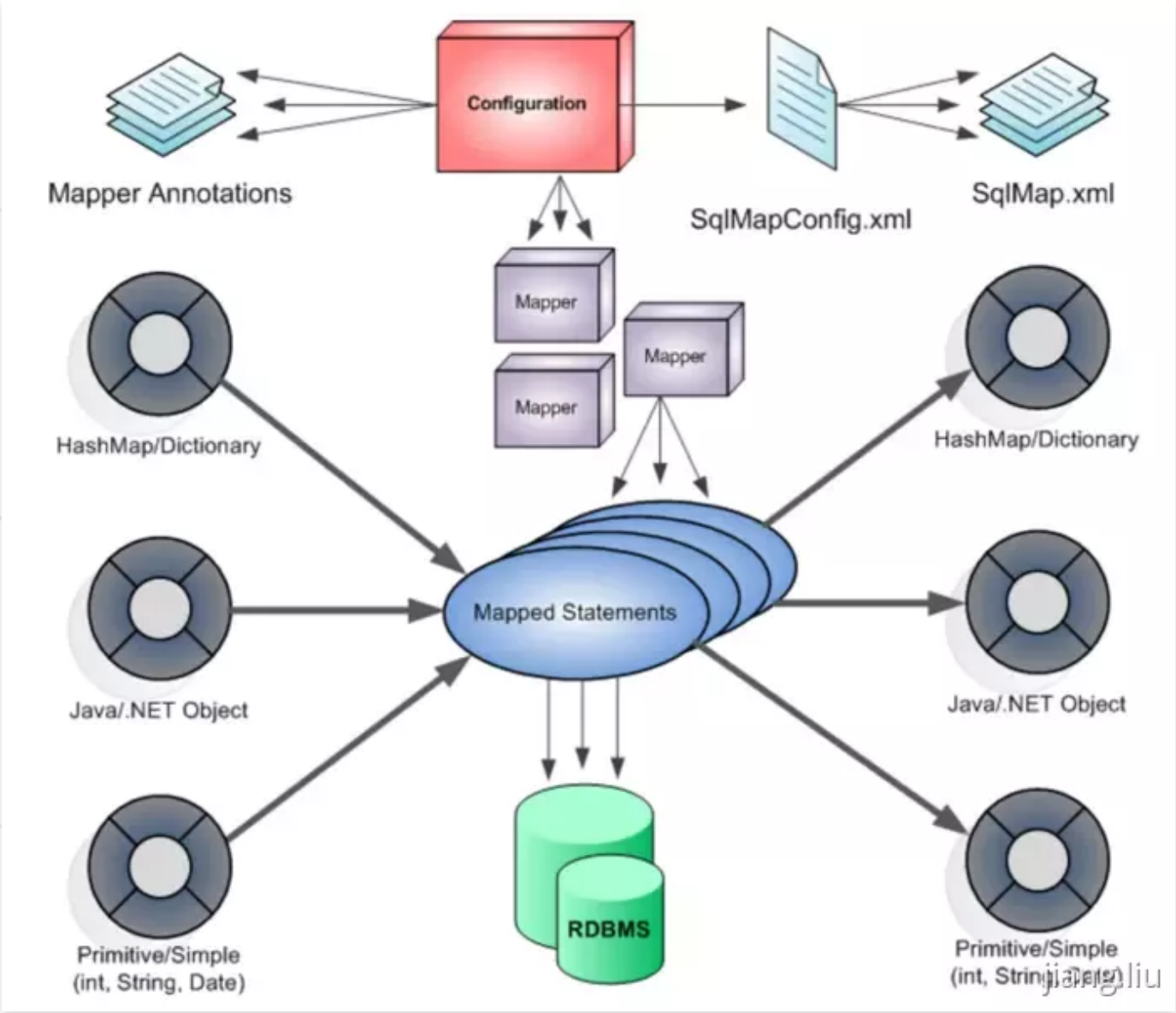
5. MyBatis的工作原理?



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。

6. MyBatis的框架架构设计是怎么的?



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。

- 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。

- 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。
- 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。
- 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。

7. 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，防止SQL注入。

8. 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。`

9. #{} 和${} 的区别?

- #{} 是预编译处理、是占位符，${}是字符串替换，是拼接符。
- Mybatis在处理#{}时，会sql中的#{}替换为？号，调用Preparedstatement的set方法来赋值。
- Mybatis在处理${}时，就是把${}替换成变量的值，调用Statement来赋值。
- #{} 的变量替换是在DBMS中，变量替换后，#{}对应的变量自动加上单引号。
- ${} 的变量替换是在DBMS之外，变量替换后，${} 对应的变量不会加单引号。
- 使用#{}可以有效防止SQL注入，提高系统安全性。

10. MyBatis 是否支持延迟加载？延迟加载的原理是什么？

MyBatis 支持延迟加载，设置 lazyLoadingEnabled=true 即可。

延迟加载的原理的是调用的时候触发加载，而不是在初始化的时候就加载信息。比如调用 a.getB().getName()，这个时候发现 a.getB() 的值为null，此时会单独触发事先保存好的关联B对象的SQL, 先查询出来B，然后再调用a.setB(b)，而这时候再调用a.getB().getName()就有值了，这就是延迟加载的基本原理。

11. 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。

12. 说一下 MyBatis 的一级缓存和二级缓存？

一级缓存：基于PerpetualCache的HashMap本地缓存，它的声明周期是和SQLSession一致的，有多个SQLSession或者分布式的环境中数据库操作，可能会出现脏数据。当Session flush或close之后，该Session中的所有Cache就将清空，默认一级缓存是开启的。

二级缓存：也是基于PerpetualCache的HashMap本地缓存，不同在于其存储作用域为Mapper级别的，如果多个SQLSession之间需要共享缓存，则需要使用到二级缓存，并且二级缓存可自定义存储源， 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

> 开启方式：在mapper类上面直接加@CacheNameSpace

二级缓存会将sql语句作为key，将结果作为value缓存到应用程序的一个全局变量中，如果下一次另一个线程请求相同的sql语句，会先到缓存中去看看有没有结果，有的话直接返回，没有再去mysql查询。线程间共享。

开启二级缓存数据查询流程：二级缓存 -> 一级缓存 -> 数据库。

> 缓存更新机制：当某一个作用域(一级缓存 Session/二级缓存 Mapper)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。

13. MyBatis 分页插件的实现原理是什么?

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。

14. MyBatis 如何编写一个自定义插件?

MyBatis 自定义插件针对 MyBatis 四大对象（Executor、StatementHandler、ParameterHandler、ResultSetHandler）进行拦截：

- Executor：拦截内部执行器，它负责调用 StatementHandler 操作数据库，并把结果集通过 ResultSetHandler 进行自动映射，另外它还处理了二级缓存的操作；
- StatementHandler：拦截 SQL 语法构建的处理，它是 MyBatis 直接和数据库执行 SQL 脚本的对象，另外它也实现了 MyBatis 的一级缓存；
- ParameterHandler：拦截参数的处理。
- ResultSetHandler：拦截结果集的处理。

15. Xml文件与DAO接口一一对应，请问这个DAO接口的工作原理？DAO接口里的方法、参数不同时，方法能重载吗?

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，唯一定位一个MappedStatement。

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接口生成代理对象 ， 代理对象会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。