Mybatic的缓存模块使用了装饰者模式的变体

# Mybatis

## $ 跟 # 的使用场景 ？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将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。

使用#{}可以有效的防止SQL注入，提高系统安全性。

$一般用于传递变量（如传递数据库名称，和order by后面的字段）。#一般用于传值。

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。

## Mybatis工作原理？

通过SqlSessionFactoryBuilder从mybatis-config.xml配置文件中构建出SqlSessionFactory。

SqlSessionFactory开启一个SqlSession，通过SqlSession实例获得Mapper对象并且运行Mapper映射的Sql语句。

完成数据库的CRUD操作和事务提交，关闭SqlSession。

## 什么是Mybatis的接口绑定，有什么好处？

Mybatis实现了DAO接口与xml映射文件的绑定，自动为我们生成接口的具体实现，使用起来变得更加省事和方便。

## 什么情况用注解，什么情况用xml绑定？

注解使用情况：Sql语句简单时

xml绑定使用情况：xml绑定 (@RequestMap用来绑定xml文件)

## Mybatis在核心处理类叫什么?

SqlSession

## 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，如果使用数据库链接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

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

MyBatis与Hibernate有哪些不同？

Mybatis和hibernate不同，它不完全是一个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，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。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，对于关系模型要求高的软件（例如需求固定的定制化软件）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。

## 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。

5.SqlMapConfig.xml中配置有哪些内容？

SqlMapConfig.xml中配置的内容和顺序如下：

properties（属性）

settings（配置）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

## 简单的说一下MyBatis的一级缓存和二级缓存？

Mybatis首先去缓存中查询结果集，如果没有则查询数据库，如果有则从缓存取出返回结果集就不走数据库。Mybatis内部存储缓存使用一个HashMap，key为hashCode+sqlId+Sql语句。value为从查询出来映射生成的java对象

Mybatis的二级缓存即查询缓存，它的作用域是一个mapper的namespace，即在同一个namespace中查询sql可以从缓存中获取数据。二级缓存是可以跨SqlSession的。

Mybaits的优点：

（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。

（2）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；

（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）。

（4）能够与Spring很好的集成；

（5）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。

## MyBatis框架的缺点：

（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）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。

## MyBatis框架适用场合：

（1）MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。

（2）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。

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接口即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Mapper的Statement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MapperStatement。在Mybatis中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MapperStatement对象。

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 MapperStatement。

Mapper接口里的方法，是不能重载的，因为是使用 全限名+方法名 的保存和寻找策略。Mapper 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Mapper接口生成代理对象proxy，代理对象会拦截接口方法，转而执行MapperStatement所代表的sql，然后将sql执行结果返回。

# Hibernate

## Hibernate的核心类和接口

Configuration ，SessionFactory，Session，Transaction，Query

Hibernate pojo的3种状态

瞬态 持久化 托管（离线）

一个PO有三种状态：

1、未被持久化的vO==pojo

此时就是一个内存对象VO，由JVM管理生命周期

2、已被持久化的PO，并且在Session生命周期内

此时映射数据库数据，由数据库管理生命周期

3、曾被持久化过，但现在和Session已经detached了，以VO的身份在运行这种和Session已经detached的PO还能够进入另一个Session，继续进行PO状态管理，此时它就成为PO的第二种状态了。这种PO实际上是跨了Session进行了状态维护的。在传统的JDO1.x中，PO只有前面两种状态，一个PO一旦脱离PM，就丧失了状态了，不再和数据库数据关联，成为一个纯粹的内存VO，它即使进入一个新的PM，也不能恢复它的状态了

## Hibernate的缓存分为

一级缓存(内置缓存)->session级别缓存,是Hibernate 的内置缓存 一级缓存在Hibernate中对应的即为session范围的缓存，也就是当session关闭时缓存即被清除，一级缓存在Hibernate中是不可配置的部分

二级缓存(应用级缓存)：

二级缓存也称进程级的缓存或SessionFactory级的缓存，二级缓存可以被所有的session共享

## Hibernate 延迟加载(lazy)

1、lazy的概念，指在需要数据的时候才发出sql

2、lazy策略只是在session打开期间才是有效的

hibernate 抓取策略 ( 查询抓取 子查询抓取 连接抓取 批量抓取)

连接抓取 fetch="join" 通过select语句使用外连接来加载其关联实体或集合 此时lazy会失效

查询抓取 fetch="select" 另外发送一条select语句抓取当前对象关联实体或集合

子查询抓取 fetch="subselect" 另外发送一条select语句抓取在前面查询到的所有实对象的关联集合 通过子查询in完成

批量抓取 batch-size设置

## Hibernate使用乐观锁

：只需在在class标签上 optimistic-lock="version"，等配置。

悲观锁的实现，通常依赖于数据库机制，在整个过程中将数据锁定，其它任何用户都不能读取或修改 利用LockMode.UPGRADE。p=(Person)session.load(Person.class, 6,LockMode.UPGRADE);