**MyBatis搭建步骤**

导入jar包

编写核心配置文件

创建持久化类和SQL映射文件

**MyBatis编程步骤**

创建SqlSessionFactory

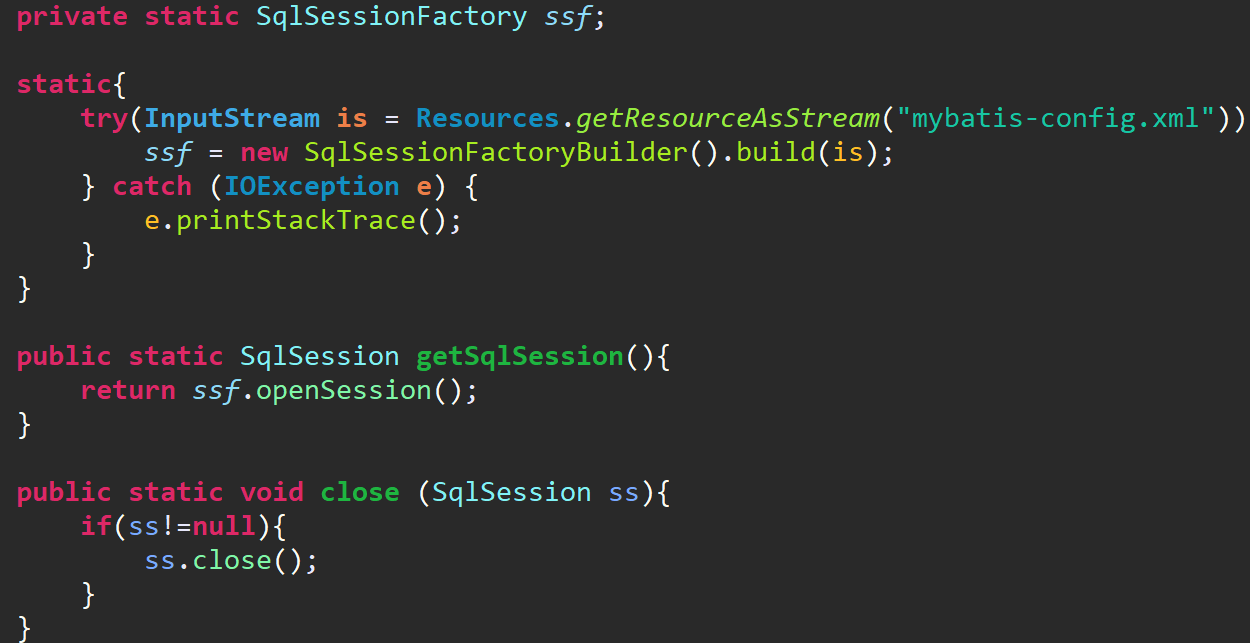
通过SqlSessionFactory创建SqlSession

通过sqlsession执行数据库操作

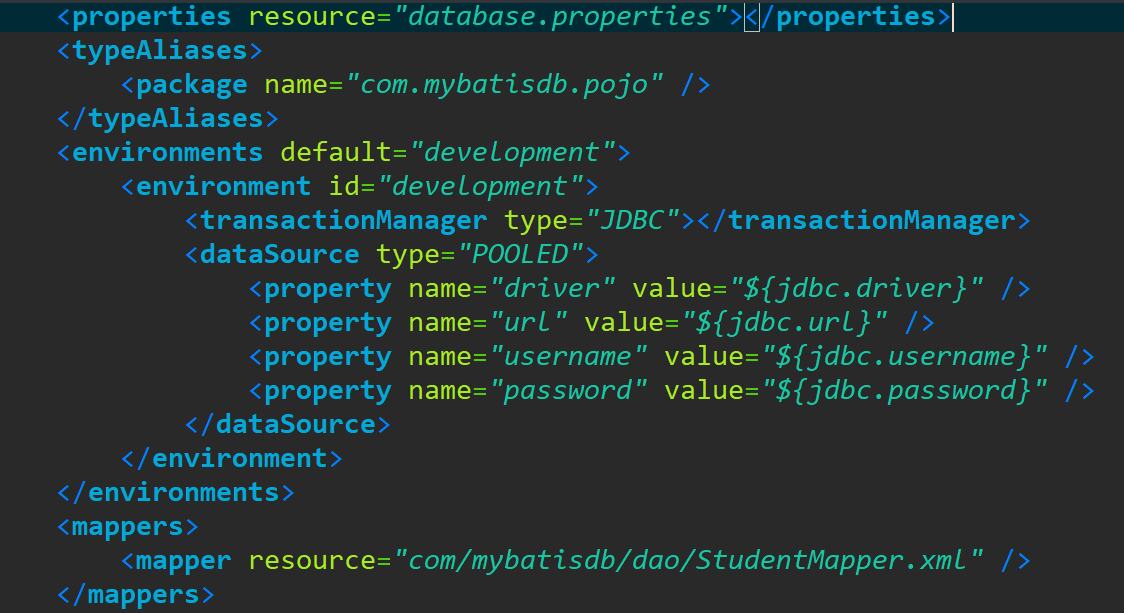
调用session.commit()提交事务

调用session.close()关闭会话

**MyBatisUtil**



**MyBatis-config.xml**



**什么是MyBatis?**

MyBatis是一个优秀的基于java的持久层框架，它内部封装了jdbc，使开发者只需要关注sql语句本身，而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。

**MyBatis核心对象及生命周期**

**SqlSessionFactoyBuilder 用过即丢**

SqlSessionFactoryBuilder这个类的作用就是为了创建SqlSessionFactory的，一旦SqlSessionFactory创建完毕，SqlSessionFactoryBuilder就没有存在的价值了，就应该被销毁。所以SqlSessionFactoryBuilder最好的作用域就是方法体内（及作为一个本地方法变量），用完即销毁。生命周期也就是调用方法的开始到结束。

**SqlSessionFactory Application**

SqlSessionFactory一旦被创建，就会存在于程序的整个生命周期，也就是只要程序不结束，它就会一直存在，你不会去主动销毁它，因为干嘛要销毁它呢？所以，我们要做的就是不要重复的去创建SqlSessionFactory！单例模式很适合这种情况！

**SqlSession Session**

SqlSession实例是不能共享的，并且不是线程安全的，所以它的最好的作用域应该是方法体内，或者是一次请求时，即method或者request，也就是说再收到一次Http请求时，就应该创建一次一个SqlSession实例，那么也就是说在每次使用完之后我们都需要把它关闭，即调用.close（）；

**什么是Hibernate?**

Hibernate 是一个开源的对象关系映射框架，它对 JDBC 进行了封装，它将 pojo 与数据库表建立映射关系，是一个全自动的 ORM框架，Hibernate 可以自动生成 SQL 语句，自动执行。

**ORM（Object - Relationship - Mapping）即对象关系映射，他是一种思想，他的实质就是将关系数据库中的业务数据用对象的形式表示出来，并通过面向对象的方式将这些对象组织出来，实现系统的业务逻辑。说到底就是 Java 实体对象跟数据库数据的映射关系。**

**JPA（Java - Persistence - API）是 Java EE 关于 ORM 思想的标准接口，仅仅是一套规范和接口，不是实现。**

**Hibernate的优缺点**

**优点**

可以让开发人员以面相对象的思想来操作数据库

映射的灵活性, 它支持各种关系数据库, 从一对一到多对多的各种复杂关系.

非侵入性、移植性会好

缓存机制，提供一级缓存和二级缓存

简洁的HQL编程

**缺点**

无法对 SQL 进行优化

框架中使用ORM原则, 导致配置过于复杂

执行效率和原生的 JDBC 相比偏差，特别是在批量数据处理的时候

**MyBatis和Hibernate的区别**

**Hibernate是全自动，而MyBatis是半自动**

hibernate完全可以自动生成sql。而MyBatis仅有基本的字段映射，仍然需要通过手写sql来实现和管理。

**Hibernate数据库移植性远大于MyBatis。**

hibernate通过它强大的映射结构和hql语言，大大降低了对象与数据库（oracle、mysql等）的耦合性，而MyBatis由于需要手写sql，移植性也会随之降低很多，成本很高。

**Hibernate拥有完整的日志系统，MyBatis则欠缺一些。**

hibernate日志系统非常健全，涉及广泛，而MyBatis则除了基本记录功能外，功能薄弱很多。

**Hibernate配置要比MyBatis复杂的多，学习成本也比MyBatis高**

**MyBatis相比Hibernate需要关心很多细节**

因为MyBatis使用简单，才导致它要比hibernate关心很多技术细节。MyBatis由于不用考虑很多细节，开发模式上与传统jdbc区别很小，hibernate则正好与之相反。但是如果使用hibernate很熟练的话，实际上开发效率甚至超越MyBatis。

**sql直接优化上，MyBatis要比Hibernate方便很多**

由于MyBatis的sql都是写在xml里，因此优化sql比hibernate方便很多。而hibernate的sql很多都是自动生成的，无法直接维护sql；总之写sql的灵活度上hibernate不及MyBatis。

**Hibernate三种状态**

临时状态：刚new出来一个对象，还没有被保存到数据库中

持久化状态：已经被保存到数据库中

游离状态：数据库中有，但是session中不存在该对象

**transactionManager标签**

该标签用于指定MyBatis所使用的事务管理器。 MyBatis 支持两种事务管理器类型：

**JDBC**

该事务器就是我们之前通过Connection的commit()方法提交，通过rollback()方法回滚，默认是需要手动提交的。

**MANAGED**

由容器来管理事务的整个事务的生命周期，默认情况下会关闭连接，将来学习spring框架之后，就不用在配置事务管理器了，spring会使用自带的管理器。

**dataSource标签**

该项主要用于配置数据源和数据库连接基本属性，有以下三种内建的数据源类型：

**UNPOOLED**

该配置表示不使用连接池，每次请求都会创建一个数据库连接，使用完毕后再关闭。当项目对数据库性能要求不大的时候，可以使用该配置。

**POOLED**

改配置表示使用MyBatis自带的数据库连接池，可以在dataSource下的property属性中设置数据库连接池的基本信息，该部分可以在MyBatis的官网中看到。

**JNDI**

配置外部数据源

**parameterType:传入参数的类型，MyBatis会推断出传入参数的类型，因此可以不用配置。**

**{ }：里面需要填写javabean中的属性，其底层是通过反射机制，调用bean类相关属性的get方法来获取值的。**

**sqlSessionFactory.openSession(true)自动提交事务**

**获取主键**

<selectKey resultType="int" keyProperty="id" order="AFTER">

SELECT @@identity

</selectKey>

**$和#的区别**

#其实是占位符，他是以?进行占位的，类似JDBC的PreparedStatement，可以防止SQL注入的问题， #{}里面的内容可以写成任意字符串，这里只是起着占位符的作用，MyBatis会将sqlSession.selectList或sqlSession.selectOne等方法中的第二个参数赋值进去。因此如果sql语句需要获取用户的输入从而进行动态拼接的话，就需要使用#{}。

$是字符串拼接，参数会被直接拼接到SQL语句中，该方式会有SQL注入问题，如果SQL语句有我们程序员直接写好，不需要用户输入的话，可以使用${}，不过一般还是建议使用#{}。

**resultMap中property是实体类中定义的对象的名称，column表示的是数据库表中的字段名称**

**什么时候使用resultType？**

当去select一张表时，可以使用resultType，这些情况下，MyBatis会在幕后自动创建一个ReusltMap，基于属性名来映射到JavaBean属性上

**什么是resultMap？**

resultMap实际上是将数据库表中的字段与实体类中的属性建立一个映射关系，这样子，即使两者名字不一致，MyBatis也会根据resultMap中的映射关系正常执行

**resultMap中添加一个id的属性来指定主键，可以提高MyBatis的查询性能。resultMap中的type属性用来指定要映射的实体类**

**涉及到两张表的操作，即使你的字段名和bean属性名一致也要编写resultMap来进行关联。**

**resultType和resultMap的区别**

resultType对应的是java对象中的属性，大小写不敏感

resultMap对应的是对已经定义好了id的resultType的引用，大小写敏感；

**mapper的动态代理**

MyBatis无需编写dao实现类了，直接通过dao接口来定位到mapper中的SQL语句

**mapper的动态代理步骤**

在映射文件中的mapper标签添加namespace属性

dao中的方法名与映射文件中的id名称要一致，这样子MyBatis就可以将方法和sql语句一一对应上了

**获得sqlsession对象**

**sqlSession = MyBatisUtil.getSqlSession();**

**dao接口 = sqlSession.getMapper(dao接口.class)**

**将dao的实现类删除之后，MyBatis底层只会调用selectOne()或selectList()方法。而框架选择方法的标准是测试类中用于接收返回值的对象类型。若接收类型为 List，则自动选择 selectList()方法；否则，自动选择 selectOne()方法**

**动态SQL**

**if标签**

<select id="selectIf">

SELECT \*\*\*\*\*

FROM \*\*\*\*\*\*

WHERE 1=1

<if test="\*\*\*\*\*">

AND \*\*\*\*\*

</if>

<if test="\*\*\*\*\*">

AND \*\*\*\*\*

</if>

</select>

**where后面添加了一个1=1的条件，这样就不至于两个条件均未设定而出现只剩下一个where，这样sql语句就不正确了，所以在后面添加了1=1这个为true的条件。**

**where标签**

<select id="selectWhere">

SELECT \*\*\*\*\*

FROM \*\*\*\*\*

<where>

<if test="\*\*\*\*\*">

\*\*\*\*\*

</if>

<if test="\*\*\*\*\*">

\*\*\*\*\*

</if>

</where>

</select>

**使用where标签后，就无需再写1=1了，注意在第一个if标签中的sql可以不加and，但是其后面的if标签中必须要加and。**

**choose标签**

<select id="selectChoose">

SELECT \*\*\*\*\*

FROM \*\*\*\*\*

<where>

<choose>

<when test="\*\*\*\*\*">

\*\*\*\*\*\*

</when>

<when test="\*\*\*\*\*">

\*\*\*\*\*

</when>

<otherwise>

1 != 1

</otherwise>

</choose>

</where>

</select>

**在choose标签中可以有多个when，但是只能有一个otherwise**

**foreach标签遍历数组**

<select id="selectForeachArray">

SELECT \*\*\*\*\*

FROM \*\*\*\*\*

<if test="\*\*\*\*\*">

WHERE \*\*\*\*\*

<foreach collection="array" open="(" close=")" item="\*\*\*\*\*" separator=",">

#{\*\*\*\*\*}

</foreach>

</if>

</select>

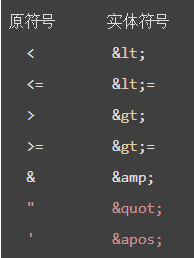
**遍历集合需要把foreach标签中collection属性改为list**

**sql标签**

sql标签可以用来定义一个可被复用的sql片段，在使用的时候写上include标签就可以将sql标签中的内容引入。

<include refid="\*\*\*\*\*"/>

**在MyBatis的动态sql中，我们可能会对一些数据进行比较大于>小于<，不过因为这些比较操作要写在xml文件中，就有可能导致xml文件解析出现问题，此时可以使用实体符号代替。**

****

**除了使用特殊符号之外，我们还可以将这些数据放到<![CDATA[ ]]>里面，这里面的内容xml是不会解析的。**

**一对多查询**

一对多关联查询指的是在查询一方对象的时候同时把跟他所关联的多方对象也查询出来

**collection：是集合的意思，即有多个对象。**

**property：指定关联属性，即实体类中的集合的属性**

**ofType：集合属性的泛型类型**

**多对一查询**

多对一指的是在查询多方对象的时候，同时将其所关联的一方对象也查询出来。由于在查询多方对象时也是一个一个查询，所以多对一关联查询，其实就是一对一关联查询。即一对一关联查询的实现方式与多对一的实现方式是相同的。

**association：标签体现出2个实体对象之间的关联关系**

**property：指定关联属性，即实体类中的属性**

**javaType：关联属性的类型**

若定义的类是双向关联，即双方的属性中均有对方对象作为域属性出现，那么它们在定义各自的 toString()方法时需要注意，只让某一方可以输出另一方即可，不要让双方的 toString()方法均可输出对方。这样会造成栈内存溢出的错误。

**多对多查询**

由两个互反的一对多关系组成，多对多关系都会通过一个中间表来建立

**自关联查询**

自关联查询就是自己充当多方，同时也充当一方，即多和一都在同一张表中，一般这样的表其实可以看做是一个树形结构，在数据库表中有一个外键，该外键表示当前数据的父节点

**延迟加载**

也称为懒加载，是指在进行关联查询时，按照设置延迟规则推迟对关联对象的select查询。延迟加载可以有效的减少数据库压力

**关联对象加载时机**

**直接加载**：执行完对主加载对象的 select 语句，马上执行对关联对象的 select 查询。

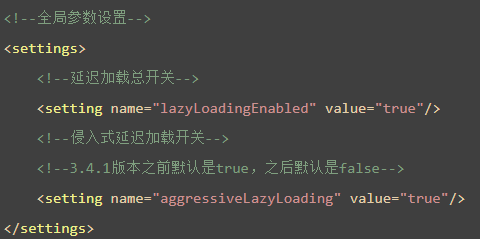
**侵入式延迟**： 执行对主加载对象的查询时，不会执行对关联对象的查询。但当要访问主加载对象的详情属性时，就会马上执行关联对象的select查询。

**深度延迟**： 执行对主加载对象的查询时，不会执行对关联对象的查询。访问主加载对象的详情时也不会执行关联对象的select查询。只有当真正访问关联对象的详情时，才会执行对关联对象的 select 查询。

**延迟加载的应用要求**

关联对象的查询与主加载对象的查询必须是分别进行的select语句，不能是使用多表连接所进行的select查询。因为，多表连接查询，其实质是对一张表的查询，对由多个表连接后形成的一张表的查询。会一次性将多张表的所有信息查询出来。

**MyBatis中对于延迟加载设置，只对于resultMap中的collection和association起作用，可以应用到一对一、一对多、多对一、多对多的所有关联关系查询中。**



**注意该内容的位置一定要在properties与typeAliases之间**

**深度延迟加载：开启总开关，将aggressiveLazyLoading关闭即可**

**若只希望某些查询支持深度延迟加载的话可以在resultMap中的collection或association添加fetchType属性，配置为lazy之后是开启深度延迟，配置eager是不开启深度延迟。fetchType属性将取代全局配置参数lazyLoadingEnabled的设置**

**缓存**

查询缓存主要是为了提高查询访问速度，即当用户执行一次查询后，会将该数据结果放到缓存中，当下次再执行此查询时就不会访问数据库了而是直接从缓存中获取该数据。 如果在缓存中找到了数据那叫做命中。 在MyBatis中有一级缓存和二级缓存两种缓存方式。

myBatis 查询缓存的作用域是根据映射文件 mapper的namespace 划分的，相同namespace 的 mapper 查询数据存放在同一个缓存区域。不同 namespace 下的数据互不干扰。无论是一级缓存还是二级缓存，都是按照 namespace 进行分别存放的。

**一级缓存**

MyBatis的一级查询缓存（也叫作本地缓存）是基于org.apache.ibatis.cache.impl.PerpetualCache 类的 HashMap本地缓存，其作用域是SqlSession，即在同一个SqlSession中两次执行相同的 sql 查询语句，第一次执行完毕后，会将查询结果写入到缓存中，第二次会从缓存中直接获取数据，而不再到数据库中进行查询，这样就减少了数据库的访问，从而提高查询效率。

**MyBatis默认一级查询缓存是开启状态，且不能关闭。**

**mybatis是如何判断某个sql语句是否在缓存中存在呢？**

因为mapper中的id具有唯一性，所以mybatis是通过这个id来判断缓存中是否存在的。如果有两个sql语句一模一样，但是两者的id不一样，此时mybatis是不会为这两个sql语句建立相同缓存的。如果一条select语句中有查询条件的话，该查询条件也会被作为特征值，即再有相同条件查询的时候，会命中。

**二级缓存**

myBatis 内置的二级缓存为 org.apache.ibatis.cache.impl.PerpetualCache。与一级缓存不同的是二级缓存的生命周期会与整个应用同步，与sqlSession是否关闭没有关系。二级缓存的使用比较简单，只需对之前的程序稍作修改即可。

**序列化javabean，将javabean实现Serializable接口，如果该javabean有显示的父类的话，让父类也实现Serializable接口。**

**在mapper配置文件中的mapper标签下添加标签：<cache/>**

**cache标签**

**eviction**：逐出策略。当二级缓存中的对象达到最大值时，就需要通过逐出策略将缓存中的对象移出缓存。默认为 LRU。常用的策略有：**FIFO**：First In First Out，先进先出；**LRU**：Least Recently Used，未被使用时间最长的

**flushInterval**：刷新缓存的时间间隔，单位毫秒。这里的刷新缓存即清空缓存。一般不指定，即当执行增删改时刷新缓存，如果长时间未刷新缓存有肯能会出现过期数据。

**readOnly**：设置缓存中数据是否只读。只读的缓存会给所有调用者返回缓存对象的相同实例，因此这些对象不能被修改的，这样性能会好一些，缺点是因为他是只读的，所以不能被修改。如果设置为false的话，读写的缓存会通过序列化返回该缓存对象的拷贝，因为会把对象进行拷贝，这会慢一些，但是安全，因此默认是 false。

**size**：二级缓存中可以存放的最多对象个数。默认为 1024 个。

**建立缓存后，下次访问前，如果对数据进行了增删改的操作，此时无论是否commit，都会清空一级缓存，二级缓存中的key是不会清空，只清空key对应的值**

**设置增删改操作的时候不清空二级缓存的话，可以在其insert或delete或update中添加属性flushCache=”false”，默认为 true。**

**关闭二级缓存**

**全局关闭**

将整个应用的二级缓存全部关闭，所有查询均不使用二级缓存。全局开关设置在mybatis.xml配置文件的全局设置中，将属性cacheEnabled设置为 false，则关闭；设置为 true，则开启，默认值为 true。即二级缓存默认是开启的。

C:\Users\hasee\Desktop\QQ浏览器截图20190107161708.png

**局部关闭**

局部关闭是只关闭某个select查询的二级缓存，在select标签中将属性useCache设置为false，那么就会关闭该select查询的二级缓存。

**二级缓存的使用注意事项**

**在一个命名空间下使用二级缓存**

二级缓存对于不同的命名空间namespace的数据是互不干扰的，倘若多个namespace中对一个表进行操作的话，就会导致这不同的namespace中的数据不一致的情况。

**在单表上使用二级缓存**

在做关联关系查询时，就会发生多表的操作，此时有可能这些表存在于多个namespace中，这就会出现上一条内容出现的问题了。

**查询多于修改时使用二级缓存**

在查询操作远远多于增删改操作的情况下可以使用二级缓存。因为任何增删改操作都将刷新二级缓存，对二级缓存的频繁刷新将降低系统性能。