# 本篇是本人看完所有Mybatis教程视频后的总结笔记

# Mybatis框架学习1——Mybatis入门

## 主要学习内容：

Mybatis框架的配置文件实现

Mybatis框架的注解实现

自定义类来实现Mybatis框架

## Mybatis框架概述

Mybatis是一个基于java的优秀的持久层框架，所谓持久层框架，即负责与数据库的数据交互的框架。

mybatis 通过 xml 配置文件或注解的方式将要执行的各种 statement 配置起来，并通过 java 对象和 statement 中 sql 的动态参数进行映射生成最终执行的sql语句，最后由 mybatis框架执行 sql 并将结果映射为 java 对象并返回。

同时，mybatis采用 ORM 思想解决了程序实体和数据库映射的问题，对 jdbc 进行了封装，使我们不用与 jdbc api 打交道，就可以完成对数据库的持久化操作，即实现实体类与数据库表的一一对应与转化。

mybatis的优点：

1. mybatis使开发者更专注于sql语句本身这也是视频中老师强调的最多的一点。sql写在xml里，便于统一管理和优化。开发者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。
2. mybatis提供映射标签，支持对象与数据库的ORM字段关系映射,而不用开发者手动转化数据。
3. mybatis提供了各种各样的缓存、延迟加载机制，提高对数据库查询的效率，以提高应用的性能。

## 2、Mybatis的xml配置实现

**2.1传统jdbc**

传统jdbc程序执行sql语句的步骤是：

加载数据库驱动Class.forName("com.mysql.jdbc.Driver");

-通过驱动管理类获取数据库连接connection = DriverManager .getConnection("jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8","root", "root");

-定义sql语句String sql = "select \* from user where username = ?";

-获取预处理statement preparedStatement = connection.prepareStatement(sql);

-设置sql参数preparedStatement.setString(1, "王五");

-向数据库发出sql执行语句resultSet = preparedStatement.executeQuery();

-遍历查询结果集

while(resultSet.next()){ System.out.println(resultSet.getString("id")+" "+resultSet.getString("username")); }

-释放资源resultSet.close(); preparedStatement.close(); connection.close();

可以看出，传统的jdbc方法有很多问题：

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。

2、Sql 语句在代码中硬编码，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。

3、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。 4、对结果集解析存在硬编码（查询列名），sql 变化导致解析代码变化，系统不易维护，如果能将数据库记 录封装成 pojo 对象解析比较方便。

**2.2 mybatis框架配置实现**

**2.2.1 持久层接口与映射文件**

在下载了mybatis的jar包并添加到pom.xml后，编写持久层的接口

**public** **interface** **IUserDao** {

**List**<**User**> **findAll**();

}

这是一个查询所有用户的操作。

对应这个持久层接口，同时创建这个接口的映射文件

<mapper namespace="com.itheima.dao.IUserDao">

<!--配置查询所有-->

<select id="findAll" resultType="com.itheima.domain.User">

select \* from user

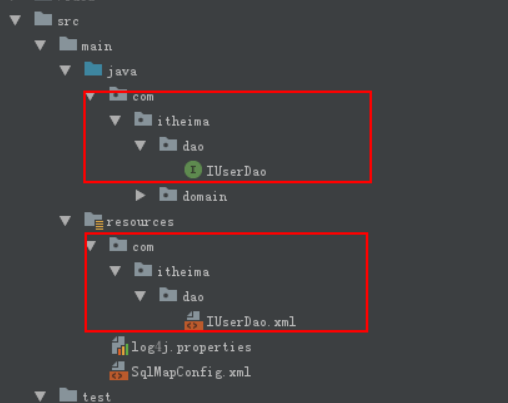
</select>

</mapper>

id为方法名称，resultType为返回类型的全限定类名。

其中，namespace的值即为持久层接口的位置。

注意，该映射文件必须和持久层接口在相同的包中，即二者的路径必须相同。其次，必须以持久层接口名称命名文件名，扩展名是.xml。



**2.2.2 编写SqlMapConfig.xml配置文件**

<!-- 配置环境 -->

<environments default="mysql">

<!-- 配置mysql的环境-->

<environment id="mysql">

<!-- 配置事务的类型-->

<transactionManager type="JDBC"></transactionManager>

<!-- 配置数据源（连接池） -->

<dataSource type="POOLED">

<!-- 配置连接数据库的4个基本信息 -->

<property name="driver" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/eesy\_mybatis"/>

<property name="username" value="root"/>

<property name="password" value="1234"/>

</dataSource>

</environment>

</environments>

<!-- 指定映射配置文件的位置，映射配置文件指的是每个dao独立的配置文件 -->

<mappers>

<mapper resource="com/itheima/dao/IUserDao.xml"/>

</mappers>

注意，该处的<mappers>中resource代表使用xml文件映射的方法来实现持久层接口，还有两种写法(class和package)将会在之后阐述。

**2.2.3 初步实现Mybatis**

//1.读取配置文件

InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

//2.创建SqlSessionFactory工厂

SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

SqlSessionFactory factory = builder.build(in);

//3.使用工厂生产SqlSession对象

SqlSession session = factory.openSession();

//4.使用SqlSession创建Dao接口的代理对象

IUserDao userDao = session.getMapper(IUserDao.class);

//5.使用代理对象执行方法

List<User> users = userDao.findAll();

for(User user : users){

System.out.println(user);

}

//6.释放资源

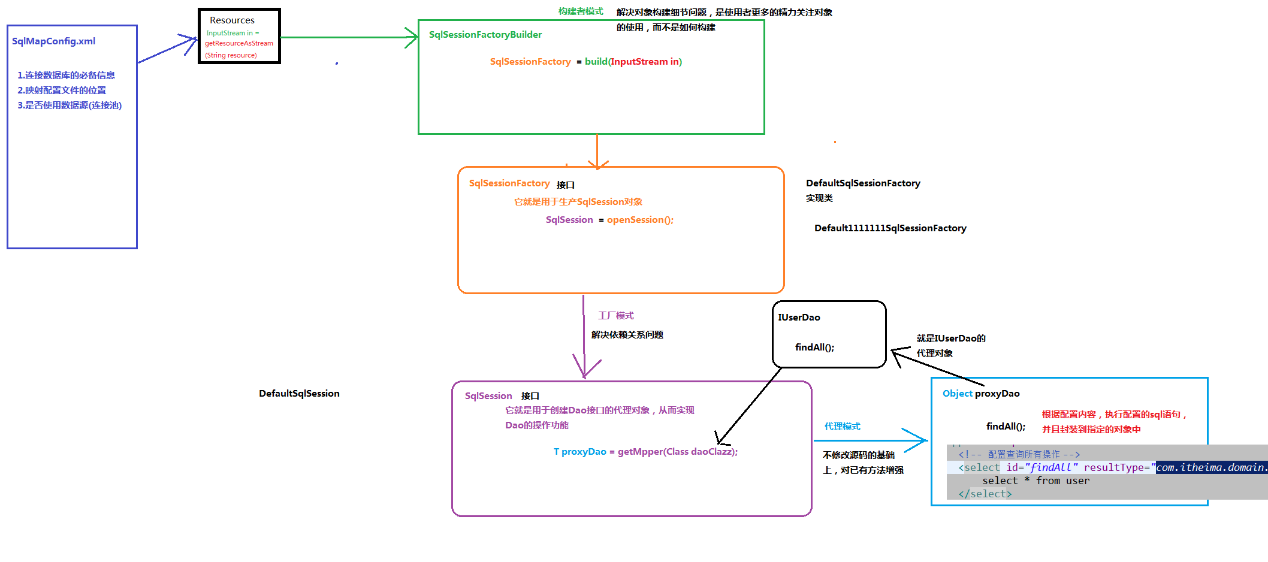
session.close();

in.close();

**其中，SqlSessionFactory和SqlSession即为Mybatis给我们提供的编写的持久层类。SqlSessionFactory读取了SqlMapConfig.xml中的配置信息，产生真正操作数据库的SqlSession,而SqlSession则用来帮开发者生成持久层接口的代理对象，解析接口的映射文件，通过代理方式执行CRUD操作。**

**2.3 小结**

**在这一部分，可以对比传统jdbc和mybatis框架实现持久层操作的过程，显然mybatis远比jdbc方便。在这个阶段，我们只用mybatis创建了持久层接口的映射文件与sql数据库的配置文件，再使用mybatis提供的持久层类来实现自动代理，即可完成持久层操作。**



**下一阶段将会对这些Mybatis提供的类（SqlSessionFactory、Sqlsession）的工作原理进行解释，但不会写的太详细。**

## 3、自定义Mybatis框架

在这个部分，将会对Mybatis实现代理模式的原理进行简要的阐述，但由于较为复杂，不会写的太详细。

**3.1 工具类**

**3.1.1 Configuration类**

**该类用于保存连接数据库的用户名、密码、地址等等信息。**

**3.1．2 XMLConfigBuilder类**

**这个类提供一个静态方法 loadConfiguration(),方法的作用是解析主配置文件，把内容填充到DefaultSqlSession类所需要的地方，即通过一些解析技术将SqlMapConfig.xml里配置的驱动、用户名、密码等信息解析获取，获得Configuration类，同时解析持久层接口映射文件（或注解），并封装成map来使用（将接口中的方法名、返回值等信息封装）。**

**3.1.3 Executor类**

**这个类的作用是通过参数获得sql语句的信息以及数据库的连接信息，执行sql操作，获得操作结果并返回。主要是作为执行类。**

**3.1.4 DataSourceUtil 类**

**该类提供一个静态方法getConnection(), 通过getConnection方法将传入的Configuration类对象进行解析，并完成注册驱动和连接操作，并返回这个连接。**

**3.1.5 Resources类**

**Resources提供一个简单的方法，将xml文件读取并得到一个流对象。**

**3.2 构建者类**

**3.2.1 SqlSessionFactoryBuilder类**

**通过传入SqlMapConfig.xml转化成的输入流，得到数据库连接的配置信息并依次创建一个相应的SqlSessionFactory，这个Factory能够得到一个SqlSession类。**

**3.2.2 DefaultSqlSessionFactory 类**

**该类实现了SqlSessionFactory接口，即实现了openSession方法，可以解析配置信息并以此返回一个SqlSession。**

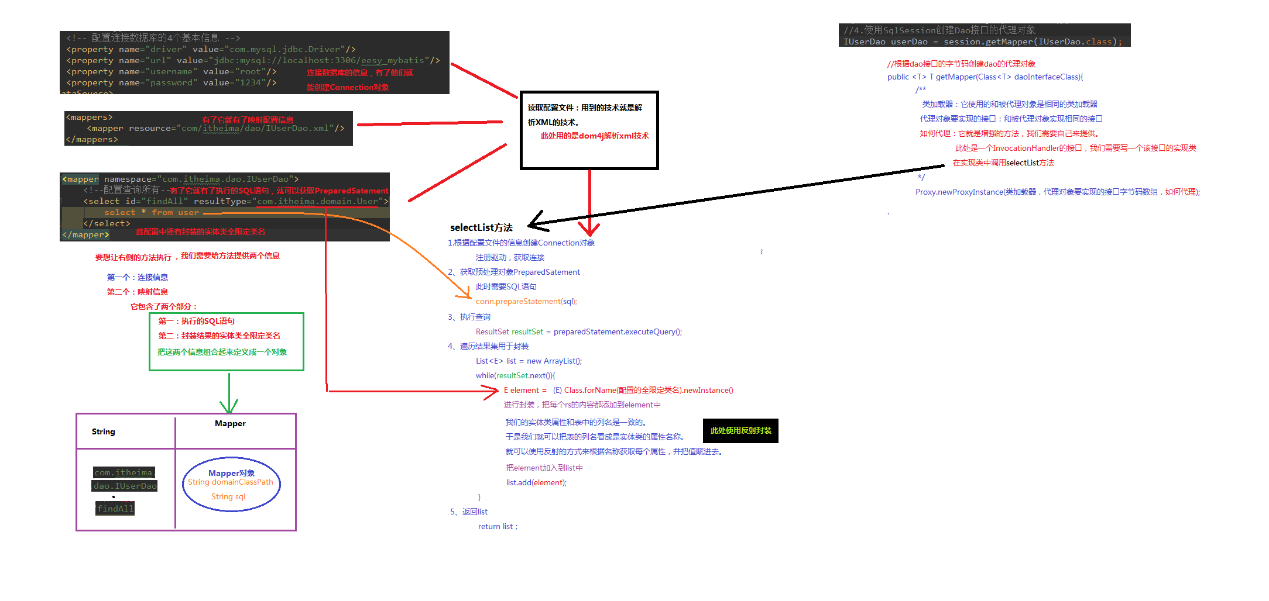
**3.2.3 DefaultSqlSession 类**

该类实现了SqlSession接口，即可释放资源，并且能通过传入的解析配置信息，调用DataSourceUtil类的方法在该类中创建持久层连接。同时我们在该类中可以调用Executor类执行查询操作，最重要的是，我们在此类中使用了动态代理，调用了MapperProxyFactory类，增强了持久层所配置的方法。

3.2.4 MapperProxyFactory类

在这个类中，我们对正在执行的方法进行增强。获取正在执行的方法的所在类名和方法名，从mapper中取出真正要执行的方法，并调用executor来执行需要执行的sql语句。

3.3 总体流程



在实现这么多类之后，总体上即可完成上一节Mybatis框架所实现的功能，最核心的步骤是：**通过创建出的session对象，调用session中的getMapper方法，完成对持久层接口的动态代理，在代理中通过连接和解析出的配置文件中的持久层方法，通过executor类来执行并返回结果。**

## 4、基于注解的Mybatis框架

**若为基于注解的Mybatis类，则不用创建持久层接口的配置文件来解析函数，而用注解的方法来获取持久层接口中的函数。**

**4.1 定义@Select注解**

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

public @interface Select

{ String value(); }

4.2 注释持久层接口

**public** **interface** **IUserDao** {

@Select("select \* from user")

**List**<**User**> **findAll**();

}

在解析dao接口获取mapper的时候，方法名直接获取，返回值则是方法名前的类型,可以解析出来。

4.3 修改SqlMapConfig.xml

<mappers>

<mapper class="com.itheima.dao.IUserDao"/>

</mappers>

在<mappers>中用class来指定被注解的dao**全限定类名**

**4.4 注解实现小结**

**如此即可完成注解实现Mybatis框架，可以对比配置方法和注解方法，发现注解方法可以通过对Dao接口的注释来替代接口的映射文件，满足方法的解析与获取操作。**

## 5、总结

总的来说，第一部分学习的内容可以分为2个部分，分别通过配置Dao接口的映射文件与给Dao接口添加注解来实现mybatis框架。而在通过Mybatis框架提供的类来完成数据库的操作之后，又通过自定义的实现Mybatis的类，来完成对数据库的操作，增强了对Mybatis实现原理的理解。总的来说，Mybatis实现的大致原理就是**通过创建出的session对象和对持久层接口的解析，调用session中的方法，完成对持久层接口的动态代理，在代理中通过连接和解析出的配置文件中的持久层方法，通过executor类来执行并返回结果。**