mybatis（持久层框架）jdbc

**第一天**

# 学习目标

* 能够独立搭建Mybatis开发环境
* 能够独立使用Mybatis完成对数据库中单表进行查询
* 能够独立使用Mybatis完成对数据库中的单表进行增加记录
* 能够独立使用Mybatis对单表修改数据
* 能够独立使用Mybatis对单表删除数据
* 能够在Mybatis核心配置文件SqlMapConfig.xml中使用Properties标签引入属性资源文件
* 能够在Mybatis核心配置文件SqlMapConfig.xml中使用typeAliases标签配置pojo的别名
* 能够在Mybatis核心配置文件SqlMapConfig.xml中使用Mappers标签引入po的映射文件
* 能够说出Mybatis和Hibernate的不同使用场景

# 总结jdbc开发的问题

## jdbc开发的步骤

1. 加载驱动
2. 创建数据库连接对象
3. 定义sql
4. 创建Statement语句对象
5. 设置参数
6. 执行
7. 处理结果集
8. 释放资源

## jdbc示例程序

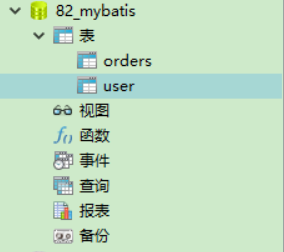
### 准备环境

jdk：1.7

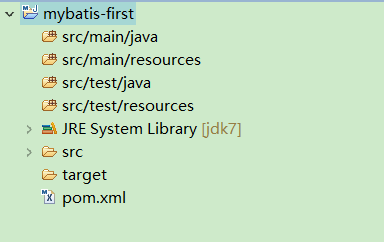
ide：eclipse

数据库：mysql

### 准备数据



### 创建项目



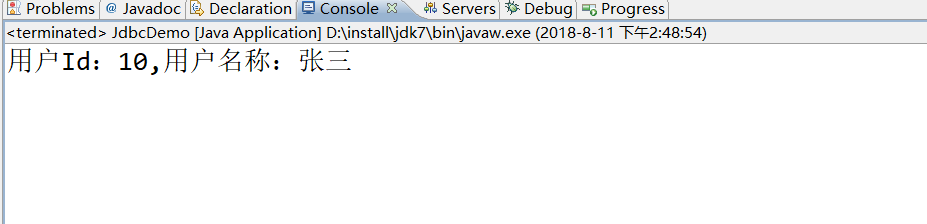
### 配置pom.xml，加入依赖包

* 数据库驱动包

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>    <groupId>cn.itheima</groupId>  <artifactId>mybatis-first</artifactId>  <version>0.0.1-SNAPSHOT</version>    <packaging>jar</packaging>  <name>mybatis-first</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <!-- mysql版本 -->  <mysql.version>5.1.30</mysql.version>  <!-- junit版本 -->  <junit.version>4.12</junit.version>  </properties>  <dependencies>  <!-- mysql数据库依赖 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- junit依赖 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

### 编写案例代码

|  |
| --- |
| **package** cn.itheima.jdbc;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  /\*\*  \* **@ClassName**: JdbcDemo  \* **@Description**: jdbc案例程序  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午2:42:10  \*  \*/  **public** **class** JdbcDemo {  /\*\*  \*  \*/  **public** **static** **void** main(String[] args) {  Connection con = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;  **try**{  // 1.加载驱动  Class.*forName*("com.mysql.jdbc.Driver");  // 2.创建数据库连接对象  con = DriverManager  .*getConnection*("jdbc:mysql://127.0.0.1:3306/82\_mybatis", "root", "admin");  // 3.定义sql  String sql = "select \* from `user` where id=?";  // 4.创建Statement语句对象  psmt = con.prepareStatement(sql);  // 5.设置参数  psmt.setInt(1, 10);  // 6.执行  rs = psmt.executeQuery();  // 7.处理结果集  **while**(rs.next()){  System.*out*.println("用户Id："+rs.getInt("id")+",用户名称："+rs.getString("username"));  }  }**catch**(Exception e){  e.printStackTrace();  }**finally**{  // 8.释放资源  **try**{  **if**(rs != **null**) rs.close();  **if**(psmt != **null**) psmt.close();  **if**(con != **null**) con.close();  }**catch**(Exception e){  e.printStackTrace();  }    }  }  } |



## 总结jdbc开发的问题

1. 频繁创建数据库连接对象和释放，容易造成系统资源的浪费，影响系统性能。在企业项目中可以使用连接池解决这个问题。但是使用jdbc需要自己来实现连接池。mybatis内部提供了连接池。
2. sql语句的定义、参数设置、结果集处理存在硬编码。在企业项目中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。
3. 结果集处理存在重复代码，处理麻烦。如果可以自动映射成java对象会比较方便。

# mybatis介绍

mybatis是Apache软件基金会下的一个开源项目，前身是Ibatis框架。2010年这个项目由apache 软件基金会迁移到google code下，改名为mybatis。2013年11月又迁移到了github（https://github.com/mybatis/mybatis-3/releases）。

mybatis是一个持久层的框架，是对JDBC操作数据库的封装，使开发者只需要关注业务本身，不需要花费精力去处理加载驱动、创建数据库连接对象、创建statement语句对象、参数设置、结果集处理等一系列繁杂的过程代码。

mybatis通过xml或注解进行配置，将java对象与sql语句中的参数自动映射生成最终执行的sql语句，并将sql语句执行结果自动映射成java对象，返回给业务层（service）应用。

三句话：

1. mybatis框架早期版本叫做Ibatis。目前托管在github
2. mybatis是对jdbc的封装，是一个持久层的框架
3. mybatis是通过xml或者注解进行配置，实现java对象与sql语句的对应关系（映射）

# mybatis入门程序

## 需求

实现用户表(user)的增删改查操作：

1. 根据用户Id查询用户
2. 根据用户名称模糊查询用户
3. 新增一个用户
4. 根据用户Id修改用户
5. 根据用户Id删除用户

## 需求实现（根据用户Id查询用户）

### 配置pom.xml，导入mybatis依赖

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>    <groupId>cn.itheima</groupId>  <artifactId>mybatis-first</artifactId>  <version>0.0.1-SNAPSHOT</version>    <packaging>jar</packaging>  <name>mybatis-first</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <!-- mysql版本 -->  <mysql.version>5.1.30</mysql.version>  <!-- junit版本 -->  <junit.version>4.12</junit.version>  <!-- mybatis版本号 -->  <mybatis.version>3.4.5</mybatis.version>  <!-- log4j日志包版本 -->  <slf4j.version>1.7.7</slf4j.version>  <log4j.version>1.2.17</log4j.version>  </properties>  <dependencies>  <!-- mysql数据库依赖 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- log4j日志包 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <!-- junit依赖 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

### 准备配置文件

#### sqlMapConfig.xml

说明：它是mybatis框架的主配置文件。文件名称是可以修改的。

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 运行环境配置,说明：  default="development"：指定使用哪一个运行环境  -->  <environments default=*"development"*>  <!-- 环境配置，id属性：唯一标识名称 -->  <environment id=*"development"*>  <!-- transactionManager：事务管理器的配置，  type="JDBC" ：目前使用的是jdbc事务  -->  <transactionManager type=*"JDBC"* />  <!-- dataSource：数据源配置，  type="POOLED"：mybatis框架内部提供的连接池  -->  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"com.mysql.jdbc.Driver"* />  <property name=*"url"* value=*"jdbc:mysql://127.0.0.1:3306/82\_mybatis"* />  <property name=*"username"* value=*"root"* />  <property name=*"password"* value=*"admin"* />  </dataSource>  </environment>  </environments>    </configuration> |

#### log4j.properties

|  |
| --- |
| # Global logging configuration  log4j.rootLogger=DEBUG, stdout  # Console output...  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n |

### 编写用户实体类

|  |
| --- |
| **package** cn.itheima.po;  **import** java.util.Date;  /\*\*  \* **@ClassName**: User  \* **@Description**: 用户实体类  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午3:15:43  \*  \*/  **public** **class** User {    **private** Integer id; // int(11) NOT NULL AUTO\_INCREMENT,  **private** String username; // varchar(32) NOT NULL COMMENT '用户名称',  **private** Date birthday; // date DEFAULT NULL COMMENT '生日',  **private** String sex; // char(1) DEFAULT NULL COMMENT '性别',  **private** String address; // varchar(256) DEFAULT NULL COMMENT '地址',  /\*\*  \* **@return** the id  \*/  **public** Integer getId() {  **return** id;  }  /\*\*  \* **@param** id the id to set  \*/  **public** **void** setId(Integer id) {  **this**.id = id;  }  /\*\*  \* **@return** the username  \*/  **public** String getUsername() {  **return** username;  }  /\*\*  \* **@param** username the username to set  \*/  **public** **void** setUsername(String username) {  **this**.username = username;  }  /\*\*  \* **@return** the birthday  \*/  **public** Date getBirthday() {  **return** birthday;  }  /\*\*  \* **@param** birthday the birthday to set  \*/  **public** **void** setBirthday(Date birthday) {  **this**.birthday = birthday;  }  /\*\*  \* **@return** the sex  \*/  **public** String getSex() {  **return** sex;  }  /\*\*  \* **@param** sex the sex to set  \*/  **public** **void** setSex(String sex) {  **this**.sex = sex;  }  /\*\*  \* **@return** the address  \*/  **public** String getAddress() {  **return** address;  }  /\*\*  \* **@param** address the address to set  \*/  **public** **void** setAddress(String address) {  **this**.address = address;  }  /\* (non-Javadoc)  \* @see java.lang.Object#toString()  \*/  @Override  **public** String toString() {  **return** "User [id=" + id + ", username=" + username + ", birthday="  + birthday + ", sex=" + sex + ", address=" + address + "]";  }  } |

### 编写sql语句

|  |
| --- |
| select \* from `user` where id=10 |

### 编写mapper映射文件

说明：该配置文件就是用户建立java对象，与sql语句的对应关系的。

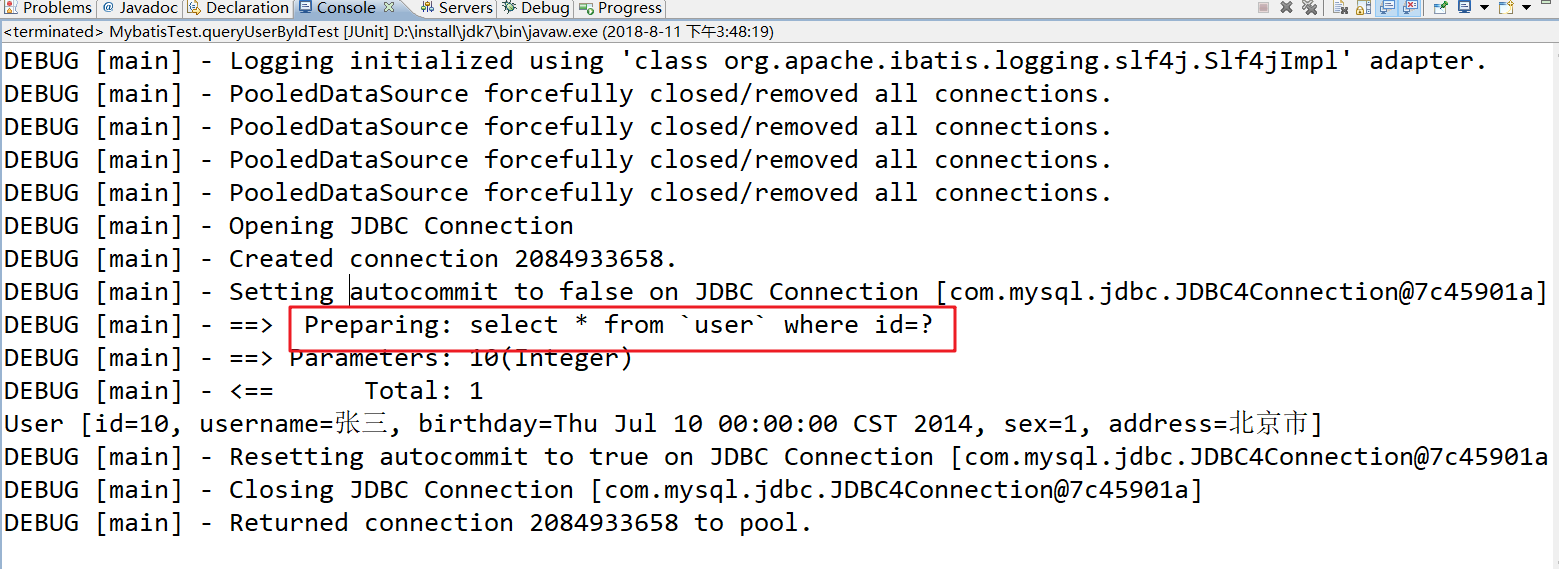
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--namespace：名称空间，相当于java中的package，用于防止sql语句名称冲突（sql语句隔离） -->  <mapper namespace=*"test"*>  <!-- mybatis框架针对每一种sql语句：新增/修改/删除/查询，提供了对应的标签：  insert/update/delete/select来放置  -->    <!-- 根据用户Id查询用户，说明：  select：放置查询的sql语句  id：唯一标识一条sql语句  parameterType：输入参数类型  resultType：返回值类型  #{id}：占位符，相当于jdbc中的问号  -->  <select id=*"queryUserById"* parameterType=*"int"* resultType=*"cn.itheima.po.User"*>  select \* from `user` where id=#{id}  </select>  </mapper> |

### 在sqlMapConfig.xml，加载mapper映射文件

|  |
| --- |
| <!-- 加载Mapper映射文件 -->  <mappers>  <!-- 加载User.xml，说明：  resource：指定文件的位置  -->  <mapper resource=*"sqlmap/User.xml"*/>  </mappers> |

### 编写测试代码

|  |
| --- |
| **package** cn.itheima.test;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** org.apache.ibatis.io.Resources;  **import** org.apache.ibatis.session.SqlSession;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.apache.ibatis.session.SqlSessionFactoryBuilder;  **import** org.junit.Test;  **import** cn.itheima.po.User;  /\*\*  \* **@ClassName**: MybatisTest  \* **@Description**: mybatis入门程序测试  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午3:38:50  \*  \*/  **public** **class** MybatisTest {    /\*\*  \* 测试根据用户Id查询用户  \* **@throws** IOException  \*/  @Test  **public** **void** queryUserByIdTest() **throws** IOException{    // 1.加载主配置文件（sqlMapConfig.xml）  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  /\*\*  \* sqlSessionFactory:  \* 1.它是mybatis框架的核心对象  \* 2.它是线程安全的  \* 3.一个项目中通常只需要一个（单例设计模式的应用）  \* 4.它给我们创建SqlSession对象  \*/  SqlSessionFactory sqlSessionFactory = builder.build(inputStream);    // 3.创建SqlSession对象  /\*\*  \* sqlSession：  \* 1.相当于jdbc中的connection对象，提供了操作数据库的CRUD方法  \* 2.它是线程不安全的，每一个执行的方法，都需要创建一个新的  \*/  SqlSession sqlSession = sqlSessionFactory.openSession();    // 4.执行数据库操作  /\*\*  \* selectOne：查询单条记录  \* 参数：  \* 参数一：执行的sql语句(名称空间+"."+sql语句的Id)  \* 参数二：传递参数  \* Object user = sqlSession.selectOne("test.queryUserById", 10);  \*/  User user = sqlSession.selectOne("test.queryUserById", 10);  System.*out*.println(user);    // 5.放资源  sqlSession.close();    }  } |



## 其它需求实现

### 根据用户名称模糊查询用户

#### 编写sql语句

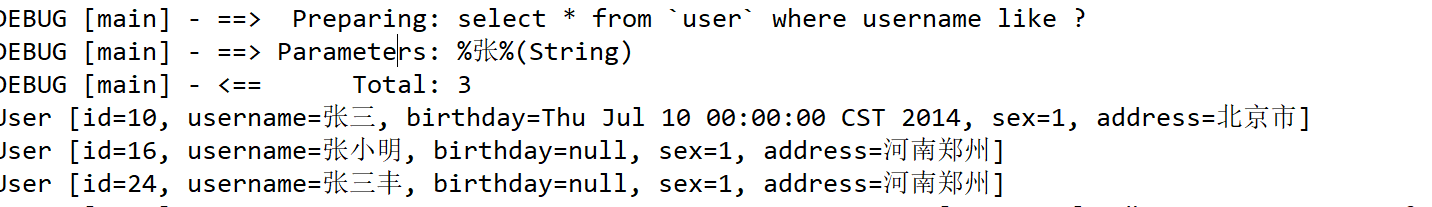
|  |
| --- |
| select \* from `user` where username like '%张%' |

#### 配置mapper映射文件

|  |
| --- |
| <!-- 根据用户名称模糊查询用户 -->  <select id=*"queryUserByName"* parameterType=*"string"* resultType=*"cn.itheima.po.User"*>  select \* from `user` where username like #{username}  </select> |

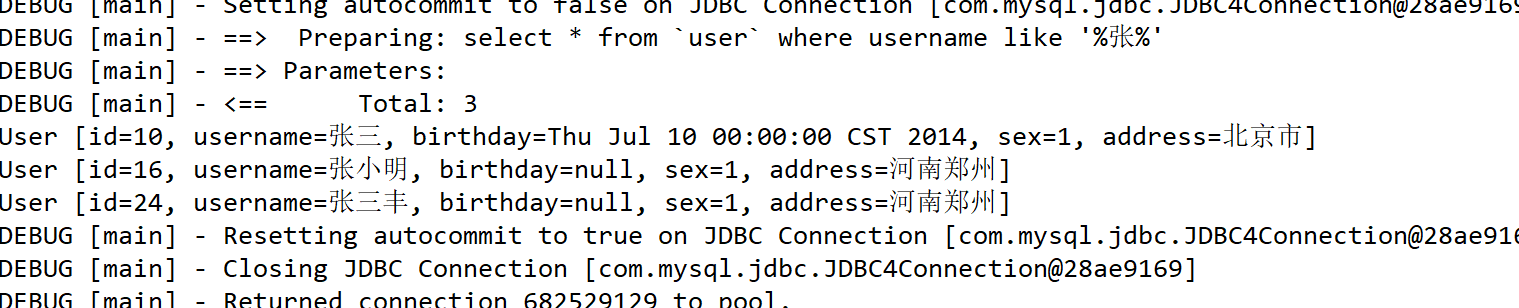
#### 编写测试代码

|  |
| --- |
| /\*\*  \* 测试根据用户名称模糊查询用户  \* **@throws** IOException  \*/  @Test  **public** **void** queryUserByNameTest() **throws** IOException{  // 1.加载主配置文件  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取主配置文件内容，获取到SqlSessionFactory  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  SqlSessionFactory sqlSessionFactory = builder.build(inputStream);    // 3.创建SqlSession  SqlSession sqlSession = sqlSessionFactory.openSession();    // 4.操作数据库  /\*\*  \* selectList：查询多条记录  \*/  List<Object> list = sqlSession.selectList("test.queryUserByName", "%张%");  **for**(Object o:list){  System.*out*.println(o);  }    // 5.释放资源  sqlSession.close();  } |



#### 使用字符串拼接符${}处理参数

|  |
| --- |
| <!-- 根据用户名称模糊查询用户 -->  <select id=*"queryUserByName"* parameterType=*"string"* resultType=*"cn.itheima.po.User"*>  <!-- select \* from `user` where username like #{username} -->    <!-- 使用字符串拼接符，处理参数：  where username like '%张%'    字符串拼接符：${value}  -->  select \* from `user` where username like '%${value}%'  </select> |



#### 站位符#{}与字符串拼接符${}区别【掌握】

1. 占位符#{}，相当于jdbc中的问号。当参数传递的是java的简单类型（八种基本类型+字符串String）的时候，花括号中的内容可以是任意字符串。
2. 字符串拼接符${}，当参数传递的是java简单类型（八种基本类型+字符串String），花括号中的内容只能是value。
3. 占位符#{}与字符串拼接符${}，当参数传递的pojo类型的时候，花括号中的内容都是pojo的属性
4. 在sql语句中使用字符串拼接符${}，可能引起sql注入的问题。在mybatis框架中我们可以考虑使用字符串拼接符。原因是J2EE项目有三层：表现层-业务层-持久层。参数传递先经过表现层和业务层已经处理过了。

### 新增一个用户

#### 编写sql语句

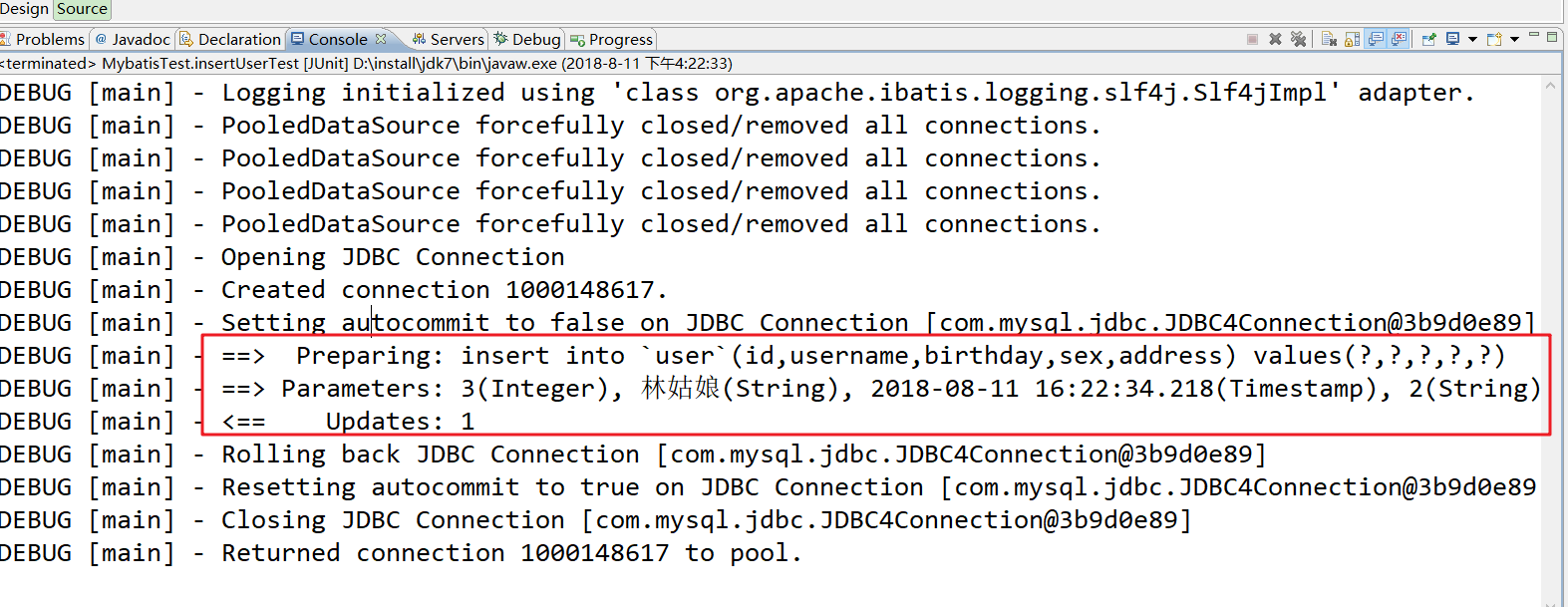
|  |
| --- |
| insert into `user`(id,username,birthday,sex,address) values(2,'小李飞刀','2018-08-11','1','明朝人') |

#### 配置映射文件

|  |
| --- |
| <!--新增一个用户，说明：  insert：放置新增的sql语句  #{id}:站位符，当参数传递的是pojo的时候，花括号中的内容是pojo的属性  -->  <insert id=*"insertUser"* parameterType=*"cn.itheima.po.User"*>  insert into `user`(id,username,birthday,sex,address)  values(#{id},#{username},#{birthday},#{sex},#{address})  </insert> |

#### 编写测试代码

|  |
| --- |
| /\*\*  \* 测试新增用户  \*/  @Test  **public** **void** insertUserTest(){    // 1.创建SqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.执行数据库操作  /\*\*  \* insert：新增记录  \*/  // 创建用户对象  User user = **new** User();  user.setId(3);  user.setUsername("林姑娘");  user.setSex("2");  user.setBirthday(**new** Date());  user.setAddress("明朝人");    sqlSession.insert("test.insertUser", user);    // 3.释放资源  sqlSession.close();    } |



问题：java程序执行成功，但是数据库中没有记录，原因是没有提交事务。

#### 提交事务

##### 方式一：手动提交

|  |
| --- |
| /\*\*  \* 测试新增用户  \*/  @Test  **public** **void** insertUserTest(){    // 1.创建SqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.执行数据库操作  /\*\*  \* insert：新增记录  \*/  // 创建用户对象  User user = **new** User();  user.setId(3);  user.setUsername("林姑娘");  user.setSex("2");  user.setBirthday(**new** Date());  user.setAddress("明朝人");    sqlSession.insert("test.insertUser", user);    /\*\*  \*提交事务：数据库关键词commit  \*/  sqlSession.commit();    // 3.释放资源  sqlSession.close();    } |

##### 方式二：自动提交

|  |
| --- |
| /\*\*  \* 测试新增用户  \*/  @Test  **public** **void** insertUserTest(){    // 1.创建SqlSession对象  //SqlSession sqlSession = this.sqlSessionFactory.openSession();    /\*\*  \* 创建sqlSession对象，指定是否自动提交事务。true：提交；fasle：不提交。默认false  \*/  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.执行数据库操作  /\*\*  \* insert：新增记录  \*/  // 创建用户对象  User user = **new** User();  user.setId(4);  user.setUsername("林姑娘");  user.setSex("2");  user.setBirthday(**new** Date());  user.setAddress("明朝人");    sqlSession.insert("test.insertUser", user);    /\*\*  \*提交事务：数据库关键词commit  \*/  //sqlSession.commit();    // 3.释放资源  sqlSession.close();    } |

注意事项：自动提交方式的特点，执行数据库操作立即提交。如果我们同一个事务中有多个操作，需要使用手动提交的方式。

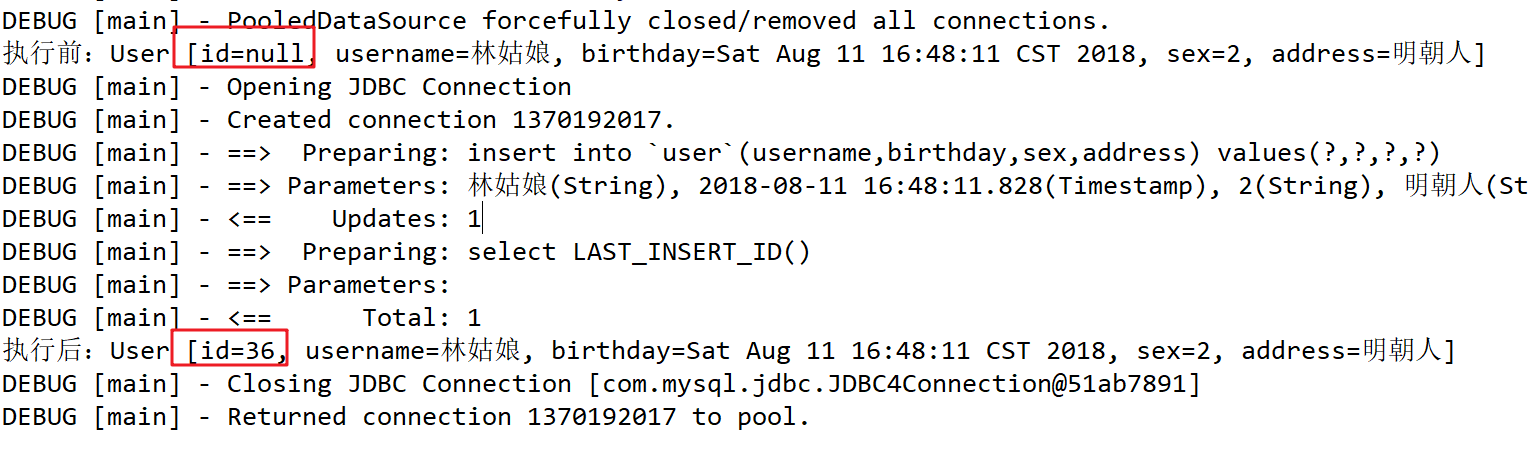
需求问题：数据库维护的自动增长的主键值，不需要我们传递。但是我们用到的时候如何查询主键值？？？

#### 查询数据库维护的主键值

mysql数据库：select LAST\_INSERT\_ID()

##### 方式一：selectKey

|  |
| --- |
| <!--新增一个用户，说明：  insert：放置新增的sql语句  #{id}:站位符，当参数传递的是pojo的时候，花括号中的内容是pojo的属性  -->  <insert id=*"insertUser"* parameterType=*"cn.itheima.po.User"*>  <!-- insert into `user`(id,username,birthday,sex,address)  values(#{id},#{username},#{birthday},#{sex},#{address}) -->    <!-- 查询数据库维护的主键值，说明：  selectKey:查询主键  keyColumn:主键字段  keyProperty:主键属性  resultType：主键字段类型  order：指定在insert语句执行前，还是执行后获取。  AFTER:在insert语句执行后  BEFORE:在insert语句执行前  -->  <selectKey keyColumn=*"id"* keyProperty=*"id"* resultType=*"int"* order=*"AFTER"*>  select LAST\_INSERT\_ID()  </selectKey>  insert into `user`(username,birthday,sex,address)  values(#{username},#{birthday},#{sex},#{address})  </insert> |



##### 方式二

|  |
| --- |
| <!--新增一个用户，说明：  insert：放置新增的sql语句  #{id}:占位符，当参数传递的是pojo的时候，花括号中的内容是pojo的属性  -->  <insert id=*"insertUser"* parameterType=*"cn.itheima.po.User"*  useGeneratedKeys=*"true"* keyColumn=*"id"* keyProperty=*"id"*>  <!-- insert into `user`(id,username,birthday,sex,address)  values(#{id},#{username},#{birthday},#{sex},#{address}) -->    <!-- 查询数据库维护的主键值，说明：  selectKey:查询主键  keyColumn:主键字段  keyProperty:主键属性  resultType：主键字段类型  order：指定在insert语句执行前，还是执行后获取。  AFTER:在insert语句执行后  BEFORE:在insert语句执行前  -->  <!-- <selectKey keyColumn="id" keyProperty="id" resultType="int" order="AFTER">  select LAST\_INSERT\_ID()  </selectKey> -->  insert into `user`(username,birthday,sex,address)  values(#{username},#{birthday},#{sex},#{address})  </insert> |

注意事项：方式二只能使用在支持自动增长的主键值 数据库（mysql）。

### 根据用户Id修改用户

#### 编写sql语句

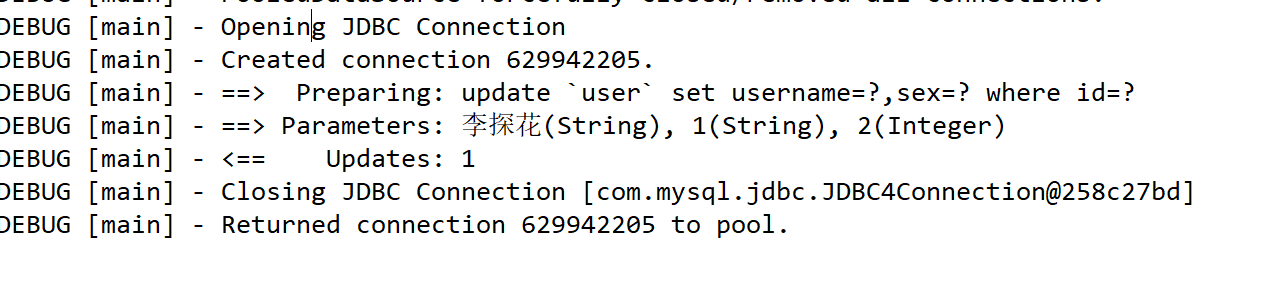
|  |
| --- |
| update `user` set username='李寻欢',sex='2' where id=2 |

#### 配置mapper映射文件

|  |
| --- |
| <!-- 根据用户Id修改用户 ，说明：  update:放置修改sql语句  -->  <update id=*"updateUser"* parameterType=*"cn.itheima.po.User"*>  update `user` set username=#{username},sex=#{sex} where id=#{id}  </update> |

#### 编写测试代码

|  |
| --- |
| /\*\*  \* 测试根据用户Id修改用户  \*/  @Test  **public** **void** updateUserByIdTest(){  // 1.创建sqlSession  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.执行数据库操作  /\*\*  \* update：更新记录  \*/  // 创建用户对象  User user = **new** User();  user.setId(2);  user.setUsername("李探花");  user.setSex("1");    sqlSession.update("test.updateUser", user);    // 3.释放资源  sqlSession.close();  } |



### 根据用户Id删除用户

#### 编写sql语句

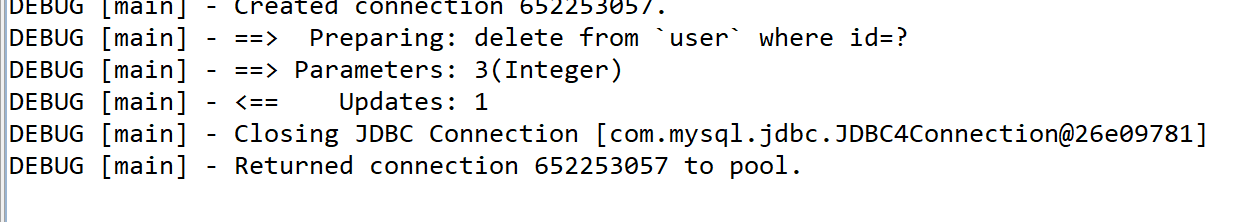
|  |
| --- |
| delete from `user` where id=4 |

#### 配置mapper映射文件

|  |
| --- |
| <!--根据用户Id删除用户，说明：  delete:放置删除sql语句  -->  <delete id=*"deleteUserById"* parameterType=*"int"*>  delete from `user` where id=#{id}  </delete> |

#### 编写测试单代码

|  |
| --- |
| /\*\*  \* 根据用户Id删除用户  \*/  @Test  **public** **void** deleteUserByIdTest(){  // 1.创建sqlSession  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.执行数据库操作  /\*\*  \* delete：删除记录  \*/  sqlSession.delete("test.deleteUserById", 3);    // 3.释放资源  sqlSession.close();  } |



# mybatis框架原理

sqlMapConfig.xml：运行环境配置（数据源），加载mapper映射文件。配置mybatis框架的运行时的环境

SqlSessionFactoryBuilder：读取配置文件类型。得到框架的核心对象（SqlSessionFactory）

SqlSessionFactory：它是mybatis框架的核心对象。创建SqlSession对象

SqlSession：发起数据库的CRUD操作

Executor：执行数据库CRUD操作

MappedStemented：输入输出映射对象

数据库：mysql

输入映射：

java简单类型

pojo类型

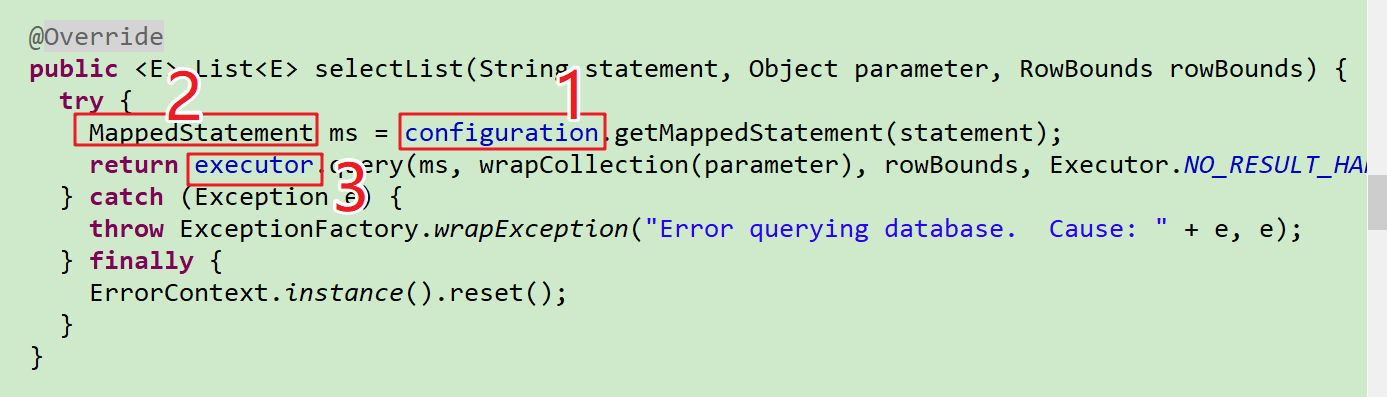
.......

输出映射：

java简单类型

pojo类型

.......



说明：

1. Configuration：配置对象，内容主配置文件（sqlMapConfig.xml），和具体mapper映射文件（User.xml......）
2. MappedStatemented：输入输出映射对象，参数输入ParameterType，返回值输出ResultType
3. Executor：执行器对象，执行数据路CRUD操作

# mybatis两种开发方法

1. 原始的dao开发方法
2. mapper代理开发方法

## 原始的dao开发方法【了解】

### 需求

1. 根据用户Id查询用户
2. 新增用户

### 需求实现

#### 编写用户dao接口

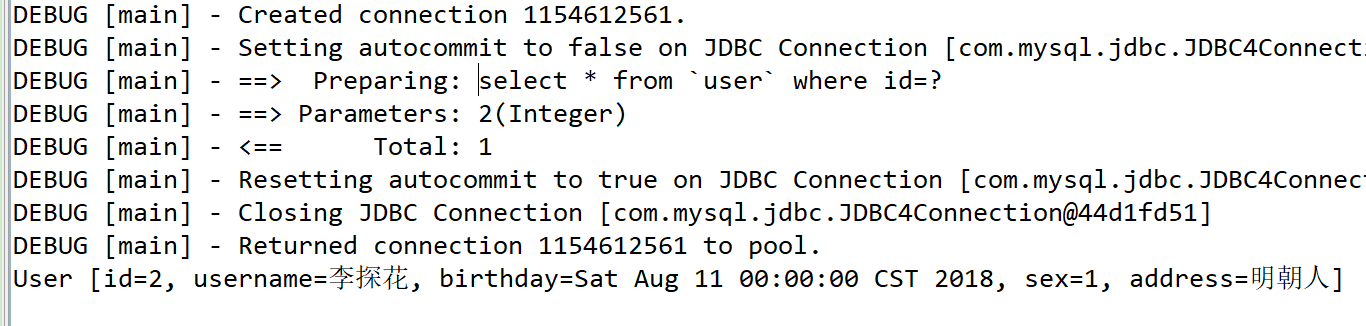
|  |
| --- |
| **package** cn.itheima.dao;  **import** cn.itheima.po.User;  /\*\*  \* **@ClassName**: UserDao  \* **@Description**: 用户dao接口  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午5:33:52  \*  \*/  **public** **interface** UserDao {    /\*\*  \* 1.根据用户id查询用户  \*/  User queryUserById(Integer id);    /\*\*  \* 2.新增用户  \*/  **void** insertUser(User user);  } |

#### 编写用户dao实现类

|  |
| --- |
| **package** cn.itheima.dao.impl;  **import** org.apache.ibatis.session.SqlSession;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** cn.itheima.dao.UserDao;  **import** cn.itheima.po.User;  /\*\*  \* **@ClassName**: UserDaoImpl  \* **@Description**: 用户dao实现类  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午5:35:14  \*  \*/  **public** **class** UserDaoImpl **implements** UserDao{    // 定义SqlSessionFactory  **private** SqlSessionFactory sqlSessionFactory;  /\*\*  \* **@param** sqlSessionFactory  \*/  **public** UserDaoImpl(SqlSessionFactory sqlSessionFactory) {  **super**();  **this**.sqlSessionFactory = sqlSessionFactory;  }  /\*\*  \* 1.根据用户id查询用户  \*/  **public** User queryUserById(Integer id) {  // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.执行数据库操作  Object user = sqlSession.selectOne("test.queryUserById", id);    // 3.释放资源  sqlSession.close();    **return** (User) user;  }  /\*\*  \* 新增用户  \*/  **public** **void** insertUser(User user) {  // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.执行数据库操作  sqlSession.insert("test.insertUser", user);    // 3.释放资源  sqlSession.close();    }  } |

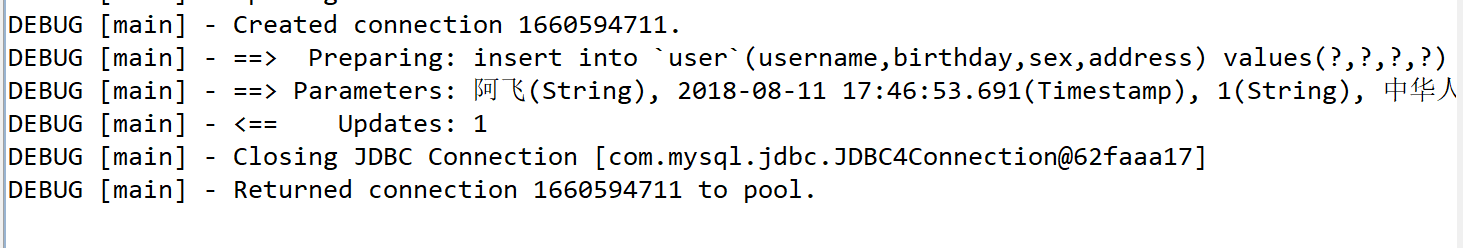
#### 编写测试代码

|  |
| --- |
| **package** cn.itheima.test;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** org.apache.ibatis.io.Resources;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.apache.ibatis.session.SqlSessionFactoryBuilder;  **import** org.junit.Before;  **import** org.junit.Test;  **import** cn.itheima.dao.UserDao;  **import** cn.itheima.dao.impl.UserDaoImpl;  **import** cn.itheima.po.User;  /\*\*  \* **@ClassName**: UserDaoTest  \* **@Description**: 原始的dao开发方法测试  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午5:40:12  \*  \*/  **public** **class** UserDaoTest {    // 定义SqlSessionFactory  **private** SqlSessionFactory sqlSessionFactory = **null**;    /\*\*  \* 初始化SqlSessionFactory  \* **@throws** IOException  \*/  @Before  **public** **void** init() **throws** IOException{  // 1.加载主配置文件（sqlMapConfig.xml）  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  /\*\*  \* sqlSessionFactory:  \* 1.它是mybatis框架的核心对象  \* 2.它是线程安全的  \* 3.一个项目中通常只需要一个（单例设计模式的应用）  \* 4.它给我们创建SqlSession对象  \*/  sqlSessionFactory = builder.build(inputStream);  }      /\*\*  \* 测试根据用户Id查询用户  \*/  @Test  **public** **void** queryUserByIdTest(){    // 1.创建用户dao对象  UserDao userDao = **new** UserDaoImpl(sqlSessionFactory);    // 2.调用方法执行  User user = userDao.queryUserById(2);  System.*out*.println(user);    }  } |



测试新增用户：

|  |
| --- |
| /\*\*  \* 测试新增用户  \*/  @Test  **public** **void** insertUserTest(){  // 1.创建UserDao对象  UserDao userDao = **new** UserDaoImpl(sqlSessionFactory);    // 2.调用方法执行  // 创建用户对象  User user = **new** User();  user.setUsername("阿飞");  user.setSex("1");  user.setBirthday(**new** Date());  user.setAddress("中华人民共和国公民");    userDao.insertUser(user);  } |



## maper代理开发方法【掌握】

### mapper代理开发的原则【很重要】

1. mapper映射文件中的namespace属性值，必须是mapper接口的全限定名称（全路径）
2. mapper映射文件中sql语句的声明，与mapper接口方法的声明一致
   1. sql语句标签的resultType属性指定的类型（如果方法返回的是一个集合list，这个时候resultType指定的是集合中存放的类型），与mapper接口方法的返回值类型一致
   2. sql语句标签的id属性值，与mapper接口的方法名称一致
   3. sql语句标签的parameterType属性指定的类型，与mapper接口的方法参数类型一致

### 需求

1. 根据用户Id查询用户
2. 新增用户

### 需求实现

#### 编写mapper接口

|  |
| --- |
| **package** cn.itheima.mapper;  **import** cn.itheima.po.User;  /\*\*  \* **@ClassName**: UserMapper  \* **@Description**: 用户mapper接口  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午5:56:48  \*  \*/  **public** **interface** UserMapper {    /\*\*  \* 1.根据用户Id查询用户  \*/  User queryUserById(Integer id);    /\*\*  \* 2.新增用户  \*/  **void** insertUser(User user);  } |

#### 编写mapper映射文件

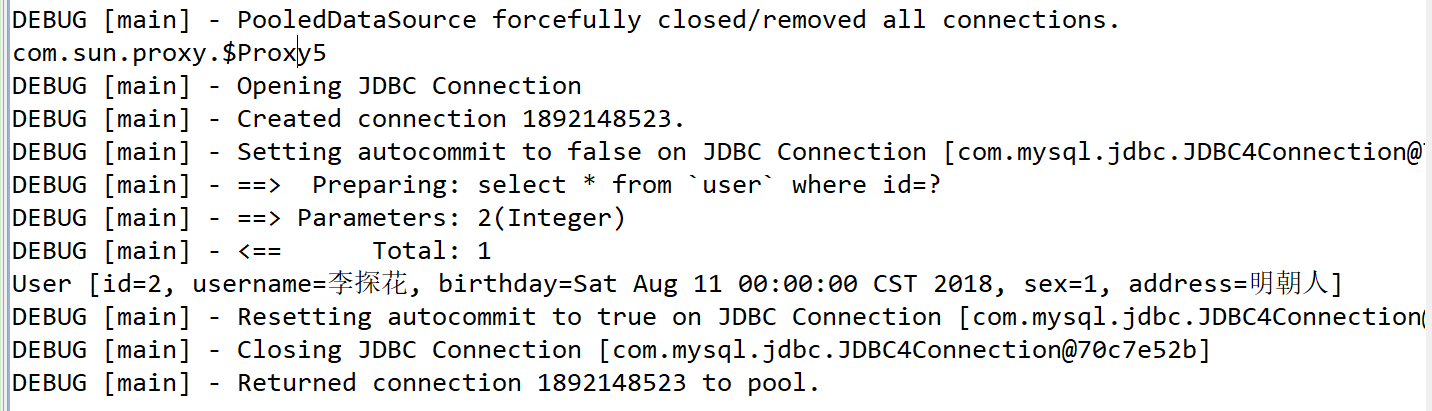
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--namespace：名称空间，相当于java中的package，用于防止sql语句名称冲突（sql语句隔离） -->  <!--mapper代理开发方法，namespace属性值必须是mapper接口全路径 -->  <mapper namespace=*"cn.itheima.mapper.UserMapper"*>  <!-- 根据用户Id查询用户 -->  <select id=*"queryUserById"* parameterType=*"int"* resultType=*"cn.itheima.po.User"*>  select \* from `user` where id=#{id}  </select>    <!-- 新增用户 -->  <insert id=*"insertUser"* parameterType=*"cn.itheima.po.User"*  useGeneratedKeys=*"true"* keyColumn=*"id"* keyProperty=*"id"*>    insert into `user`(username,birthday,sex,address)  values(#{username},#{birthday},#{sex},#{address})  </insert>  </mapper> |

#### 在sqlMapConfig.xml加载UserMapper.xml

|  |
| --- |
| <!-- 加载Mapper映射文件 -->  <mappers>  <!-- 加载User.xml，说明：  resource：指定文件的位置  -->  <mapper resource=*"sqlmap/User.xml"*/>  <!-- 加载UserMapper.xml -->  <mapper resource=*"sqlmap/UserMapper.xml"*/>  </mappers> |

#### 编写测试代码

|  |
| --- |
| **package** cn.itheima.test;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** org.apache.ibatis.io.Resources;  **import** org.apache.ibatis.session.SqlSession;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.apache.ibatis.session.SqlSessionFactoryBuilder;  **import** org.junit.Before;  **import** org.junit.Test;  **import** cn.itheima.mapper.UserMapper;  **import** cn.itheima.po.User;  /\*\*  \* **@ClassName**: UserMapperTest  \* **@Description**: mapper代理开发方法测试  \* **@author** 传智 小杨老师  \* **@date** 2018-8-11 下午6:01:06  \*  \*/  **public** **class** UserMapperTest {    // 定义SqlSessionFactory  **private** SqlSessionFactory sqlSessionFactory = **null**;    /\*\*  \* 初始化SqlSessionFactory  \* **@throws** IOException  \*/  @Before  **public** **void** init() **throws** IOException{  // 1.加载主配置文件（sqlMapConfig.xml）  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  /\*\*  \* sqlSessionFactory:  \* 1.它是mybatis框架的核心对象  \* 2.它是线程安全的  \* 3.一个项目中通常只需要一个（单例设计模式的应用）  \* 4.它给我们创建SqlSession对象  \*/  sqlSessionFactory = builder.build(inputStream);  }    /\*\*  \* 测试根据用户id查询用户  \*/  @Test  **public** **void** queryUserByIdTest(){    // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.获取接口的代理对象  /\*\*  \* getMapper:获取mapper接口代理对象  \* 参数：  \* Class：接口的字节码  \*/  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);  System.*out*.println(mapper.getClass().getName());    // 3.调用方法执行  User user = mapper.queryUserById(2);  System.*out*.println(user);    // 4.释放资源  sqlSession.close();    }  } |



测试新增用户：

|  |
| --- |
| /\*\*  \* 测试新增用户  \*/  @Test  **public** **void** insertUserTest(){  // 1.创建sqlSession  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.获取mapper接口代理对象  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);    // 3.调用方法执行  // 创建用户对象  User user= **new** User();  user.setUsername("龙小云");  user.setBirthday(**new** Date());  user.setSex("1");  user.setAddress("明朝人");    mapper.insertUser(user);    // 4.释放资源  sqlSession.close();  } |

# mybatis核心配置文件

说明：sqlMapConfig.xml：运行环境的配置（数据源），mapper映射文件。

|  |  |  |
| --- | --- | --- |
| 顺序 | 配置标签名称 | 说明 |
| 1 | properties | 属性 |
| 2 | settings | 配置全局参数 |
| 3 | typeAliases | 类型别名 |
| 4 | typeHandlers | 类型处理器 |
| 5 | objectFactory | 对象工厂 |
| 6 | plugins | 插件 |
| 7 | environments | 环境集合属性对象 |
| 8 | databaseIdProvider | 多数据库支持 |
| 9 | mappers | 映射器 |
| 说明：   1. 红色字体内容要求掌握 2. 在sqlMapConfig.xml中必须是从上往下的配置顺序 | | |

## properties（属性）

作用：加载属性资源文件（db.properties）和定义属性。

### 编写db.properties

|  |
| --- |
| db.driver=com.mysql.jdbc.Driver  db.url=jdbc:mysql://127.0.0.1:3306/82\_mybatis  db.username=root  db.password=admin |

### 加载db.properties文件

|  |
| --- |
| <!-- 加载属性资源文件 ,说明：  1.resource：加载【本地的】资源文件  2.url:加载【网络上的】资源文件  -->  <properties resource=*"db.properties"*>  </properties> |

### 使用资源属性文件中的属性

|  |
| --- |
| <!-- 运行环境配置,说明：  default="development"：指定使用哪一个运行环境  -->  <environments default=*"development"*>  <!-- 环境配置，id属性：唯一标识名称 -->  <environment id=*"development"*>  <!-- transactionManager：事务管理器的配置，  type="JDBC" ：目前使用的是jdbc事务  -->  <transactionManager type=*"JDBC"* />  <!-- dataSource：数据源配置，  type="POOLED"：mybatis框架内部提供的连接池  -->  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"${db.driver}"* />  <property name=*"url"* value=*"${db.url}"* />  <property name=*"username"* value=*"${db.username}"* />  <property name=*"password"* value=*"${db.password}"* />  </dataSource>  </environment>  </environments> |

### 使用内部property标签定义属性

|  |
| --- |
| <!-- 加载属性资源文件 ,说明：  1.resource：加载【本地的】资源文件  2.url:加载【网络上的】资源文件  3.使用内部property标签定义属性  4.加载顺序：先加载内部property标签定义的属性，再加载属性文件中的属性。如果有相同的属性，  属性资源文件中的属性，会覆盖内部property标签定义的属性  -->  <properties resource=*"db.properties"*>  <property name=*"db.username"* value=*"root123"*/>  <property name=*"db.password"* value=*"admin"*/>  </properties> |

## typeAliases（类型别名）

作用：简称。比如中华人民共和国。简称中国。

### 内置别名

|  |  |
| --- | --- |
| 别名 | 映射类型 |
| \_byte | byte |
| \_long | long |
| \_short | short |
| \_int | int |
| \_integer | int |
| \_double | double |
| \_float | float |
| \_boolean | boolean |
| string | String |
| byte | Byte |
| long | Long |
| short | Short |
| int | Integer |
| integer | Integer |
| double | Double |
| float | Float |
| boolean | Boolean |
| date | Date |
| decimal | BigDecimal |
| bigdecimal | BigDecimal |
| object | Object |
| map | Map |
| hashmap | HashMap |
| list | List |
| arraylist | ArrayList |
| collection | Collection |
| iterator | Iterator |

说明：

1. 内置别名可以直接使用
2. 别名不区分大小写

### 自定义别名

#### 方式一

|  |
| --- |
| <!-- 配置自定义别名 -->  <typeAliases>  <!-- 配置用户的别名，说明：  1.type：指定别名的类型  2.默认使用类的名称作为别名的名称  3.alias:指定别名的名称  -->  <typeAlias type=*"cn.itheima.po.User"* alias=*"user"*/>  </typeAliases> |

#### 方式二

|  |
| --- |
| <!-- 配置自定义别名 -->  <typeAliases>  <!-- 配置用户的别名，说明：  1.type：指定别名的类型  2.默认使用类的名称作为别名的名称  3.alias:指定别名的名称    <typeAlias type="cn.itheima.po.User" alias="user"/> -->    <!-- 包扫描方式配置别名,说明：  1.name：配置要扫描的包  2.默认都使用类的名称作为别名的名称  3.如果有多个包。在同一个父包，配置父包即可  4.不在同一个父包，配置多个<package name="cn.itcast.po"/>  -->  <package name=*"cn.itheima.po"*/>    </typeAliases> |

## mappers（映射器）

作用：加载mapper映射文件。

### 方式一

|  |
| --- |
| <!-- 加载Mapper映射文件 -->  <mappers>  <!-- 加载User.xml，说明：  resource：指定文件的位置  -->  <mapper resource=*"sqlmap/User.xml"*/>  <!-- 加载UserMapper.xml -->  <mapper resource=*"sqlmap/UserMapper.xml"*/>  </mappers> |

### 方式二

|  |
| --- |
| <!-- 加载Mapper映射文件 -->  <mappers>  <!-- 加载User.xml，说明：  resource：指定文件的位置  -->  <mapper resource=*"sqlmap/User.xml"*/>  <!-- 加载UserMapper.xml  <mapper resource="sqlmap/UserMapper.xml"/> -->    <!-- 包扫描方式加载mapper映射文件，说明：  1.要求前提必须mapper代理的开发方法  2.要求mapper映射文件，与mapper接口要放在同一个目录  3.要求mapper映射文件的名称，与mapper接口的名称要一致  -->  <package name=*"cn.itheima.mapper"*/>    </mappers> |

# mybatis与hibernate比较

## 相同点

都是对jdbc操作数据库的封装，都是持久层的框架。

## 不同点

1. hibernate对sql语句做了封装，使用HQL语句操作数据库。数据库无关性支持好，在项目需要支持多种数据库的情况下，代码开发量少，sql语句性能优化困难。

mybatis使用sql语句操作数据库，不支持数据库无关性，在项目需要支持多种数据库的情况下，代码开发量相对较多。sql语句性能优化容易。

1. hibernate配置java对象，与数据库表的对应关系，多表的关联关系查询配置复杂。mybatis配置java对象，与sql语句执行结果的对应关系，多表的关联关系查询配置简单。
2. hibernate学习，使用门槛高，适合需求稳定的中小型项目，比如：办公自动化系统，财务系统。mybatis学习，使用门槛低，适合需求变化频繁，大型项目，比如：互联网项目。