mybatis（持久层框架）jdbc

**第一天**

# 课程计划

1. ：mybatis基础知识
2. 总结Jdbc开发的问题
3. mybatis介绍
4. mybatis入门程序
5. mybatis框架原理
6. mybatis两种开发方法
7. mybatis核心配置文件
8. mybatis与hibernate比较

# 总结jdbc开发的问题

## jdbc开发步骤

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

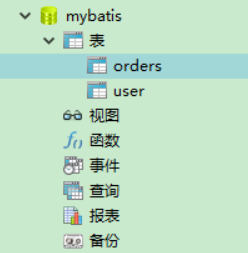
### 准备环境

jdk：1.7

ide：eclipse

数据库：mysql

### 准备数据



### 创建工程

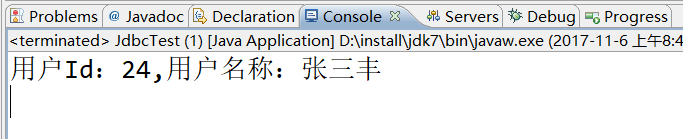


### 配置pom.xml文件，加入数据库驱动依赖

|  |
| --- |
| <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>  </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> |

### 编写代码

|  |
| --- |
| **public** **class** JdbcTest {  /\*\*  \* **@Title**: main  \* **@Description**: **TODO**(这里用一句话描述这个方法的作用)  \* **@param** args 设定文件  \* **@return** void 返回类型  \* **@throws**  \*/  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  Connection con = **null**;  PreparedStatement psmt = **null**;  ResultSet set = **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/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, 24);    // 6.执行  set = psmt.executeQuery();    // 7.处理结果集  **while**(set.next()){  System.*out*.println("用户Id："+set.getInt("id")+",用户名称："+set.getString("username"));  }    }**catch**(Exception e){  e.printStackTrace();  }**finally**{  // 8.释放资源  **try**{  **if**(set != **null**) set.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、释放，容易造成系统资源浪费，影响系统性能。企业项目中可以使用连接池解决这个问题，但是使用Jdbc需要自己实现连接池。mybatis已经提供了连接池。
2. sql语句定义、参数设置、结果集处理存在硬编码。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。如果可以映射为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。

# mybatis入门程序

## 需求

实现用户表（user）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

### 配置pom.xml文件，加入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：mybatis框架核心配置文件。

|  |
| --- |
| <?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 -->  <!-- default属性：指定使用哪一个运行环境 -->  <environments default=*"development"*>    <!-- id属性：唯一标识一个运行环境 -->  <environment id=*"development"*>  <!-- 事务管理器配置，JDBC：mybatis默认使用jdbc事务 -->  <transactionManager type=*"JDBC"* />    <!-- 数据源配置，POOLED：mybatis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/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 |

### 准备用户pojo

|  |
| --- |
| **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 + "]";  }      } |

### 准备sql语句

|  |
| --- |
| select \* from `user` where id=24 |

### 准备mapper映射文件

说明：用于配置java对象，与sql语句的对应关系。

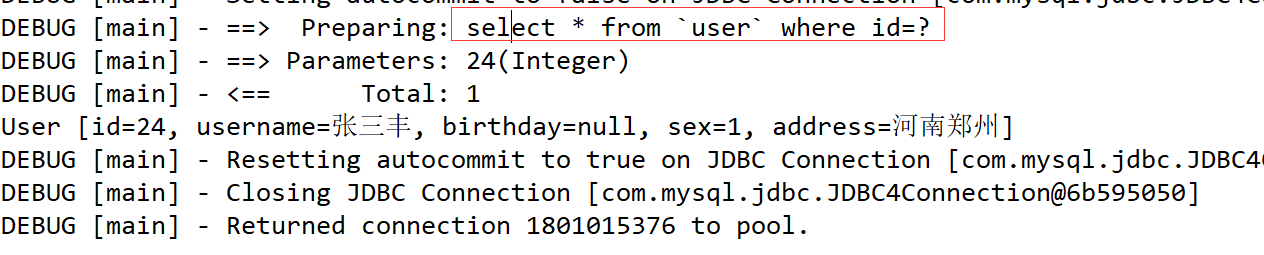
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- namespace属性：名称空间，相当于java中的package，用于放置sql语句冲突 -->  <mapper namespace=*"test"*>  <!--说明：mybatis框架针对每一种sql语句：新增/修改/删除/查询，提供了对应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

### 在sqlMapcConfig.xml文件中，加载User.xml文件

|  |
| --- |
| <?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 -->  <!-- default属性：指定使用哪一个运行环境 -->  <environments default=*"development"*>    <!-- id属性：唯一标识一个运行环境 -->  <environment id=*"development"*>  <!-- 事务管理器配置，JDBC：mybatis默认使用jdbc事务 -->  <transactionManager type=*"JDBC"* />    <!-- 数据源配置，POOLED：mybatis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/mybatis"* />  <property name=*"username"* value=*"root"* />  <property name=*"password"* value=*"admin"* />  </dataSource>  </environment>  </environments>    <!-- 加载mapper映射文件 -->  <mappers>  <!--加载User.xml文件，说明：  resource属性：指定配置文件的位置-->  <mapper resource=*"sqlmap/User.xml"*/>    </mappers>    </configuration> |

### 编写测试代码

|  |
| --- |
| **public** **class** MybatisTest {    // 1.测试根据用户Id查询用户  @Test  **public** **void** queryUserByIdTest() **throws** IOException{    // 1.加载核心配置文件：sqlMapConfig.xml  // resource属性：指定核心配置文件位置  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  // SqlSessionFactory对象：mybatis框架的核心对象，一个项目中只需要一个（单例设计模式）  SqlSessionFactory sqlSessionFactory = builder.build(inputStream);    // 3.使用SqlSessionFactory对象，创建sqlSession对象  // sqlSession对象：  // 3.1相当于jdbc中的connection对象  // 3.2 sqlSession对象是线程不安全，每一个执行操作的方法，都需要创建一个sqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession();    // 4.使用sqlSession对象，调用方法执行  // selectOne方法：查询单条记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  Object user = sqlSession.selectOne("test.queryUserById", 24);  System.*out*.println(user);    // 5.释放资源  sqlSession.close();    }  } |



## 其他需求实现

### 根据用户名称模糊查询用户

#### 准备sql语句

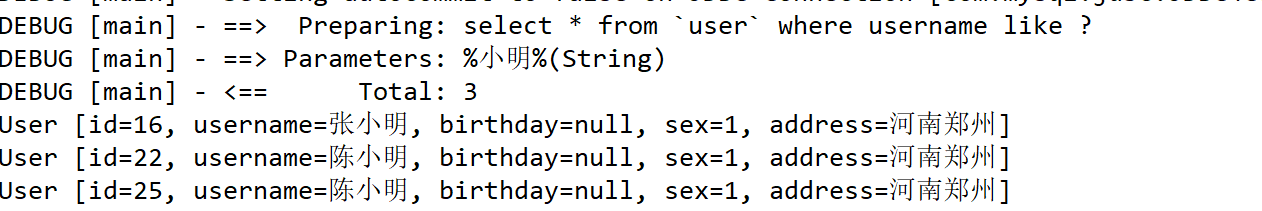
|  |
| --- |
| select \* from `user` where username like '%小明%' |

#### 配置映射文件

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

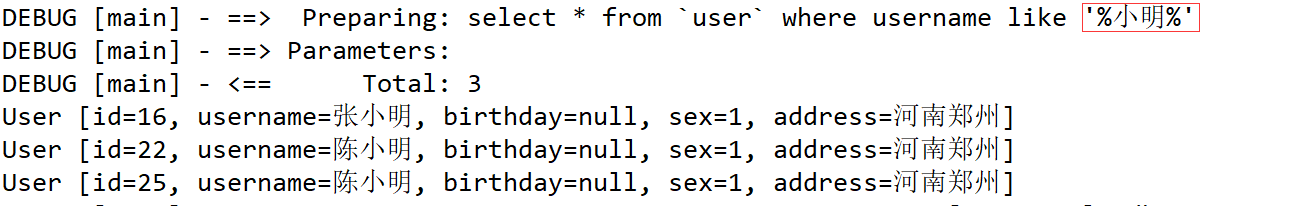
#### 编写测试代码

|  |
| --- |
| // 2.测试根据用户名称模糊查询用户  @Test  **public** **void** queryUserByNameTest() **throws** IOException{  // 1.加载核心配置文件  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  SqlSessionFactory sqlSessionFactory = builder.build(inputStream);    // 3.使用sqlSessionFactory对象，创建sqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession();    // 4.使用sqlSession对象，调用方法执行  // selectList方法：查询多条记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  List<Object> list = sqlSession.selectList("test.queryUserByName", "%小明%");  **for**(Object o :list){  System.*out*.println(o);  }    // 5.释放资源  sqlSession.close();  } |



#### 使用字符串拼接符${}，拼接参数

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



#### 占位符#{}与字符串拼接符${}区别

1. 占位符#{}，相当于jdbc中的问号?，当参数传递的是java简单类型的时候，花括号中的内容可以是任意字符串。
2. 字符串拼接符 ${}，当参数传递的是java简单类型的时候，花括号中的内容只能是：value
3. 在sql语句中使用字符串拼接符${}，可能引起sql注入的问题。但是mybatis中字符串拼接符可以使用，原因是是后端dao层的开发，项目中参数传递经过表现层（action），业务层（service）已经处理好。

### 新增用户

#### 准备sql语句

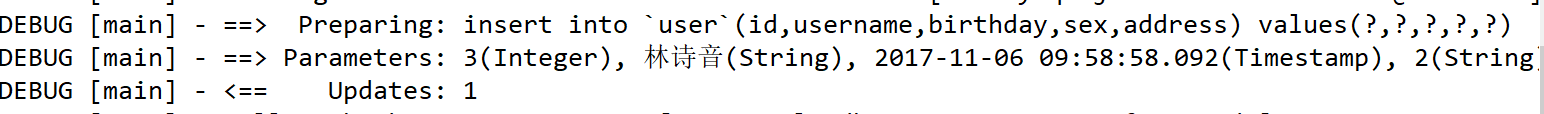
|  |
| --- |
| insert into `user`(id,username,birthday,sex,address) values(2,'小李飞刀','2017-11-06','1','来自明朝') |

#### 配置映射文件

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

#### 编写测试代码

|  |
| --- |
| // 3.测试新增用户  @Test  **public** **void** insertUserTest(){  // 1.创建sqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession();    // 2.使用sqlSession对象，调用方法执行  // insert方法：新增记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  // 创建用户对象  User user = **new** User();  user.setId(3);  user.setUsername("林诗音");  user.setBirthday(**new** Date());  user.setSex("2");  user.setAddress("来自大明朝");    sqlSession.insert("test.insertUser", user);    // 3.释放资源  sqlSession.close();    } |



问题：java程序执行成功，但是数据库中并没有林诗音的数据，原因是没有提交事务。

#### 提交事务

##### 方式一

|  |
| --- |
| // 3.测试新增用户  @Test  **public** **void** insertUserTest(){  // 1.创建sqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession();    // 2.使用sqlSession对象，调用方法执行  // insert方法：新增记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  // 创建用户对象  User user = **new** User();  user.setId(3);  user.setUsername("林诗音");  user.setBirthday(**new** Date());  user.setSex("2");  user.setAddress("来自大明朝");    sqlSession.insert("test.insertUser", user);    // 数据库提交事务：commit  sqlSession.commit();    // 3.释放资源  sqlSession.close();    } |

##### 方式二

|  |
| --- |
| // 3.测试新增用户  @Test  **public** **void** insertUserTest(){  // 1.创建sqlSession对象  //SqlSession sqlSession = sqlSessionFactory.openSession();    // 创建sqlSession对象，指定自动提交事务。true：表示提交；false：表示不提交。默认是false  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.使用sqlSession对象，调用方法执行  // insert方法：新增记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  // 创建用户对象  User user = **new** User();  user.setId(4);  user.setUsername("林诗音");  user.setBirthday(**new** Date());  user.setSex("2");  user.setAddress("来自大明朝");    sqlSession.insert("test.insertUser", user);    // 数据库提交事务：commit  //sqlSession.commit();    // 3.释放资源  sqlSession.close();    } |

问题：用户表中id是自增长，不需要传递，但是mybatis中如何获取到新增以后的用户Id？？？

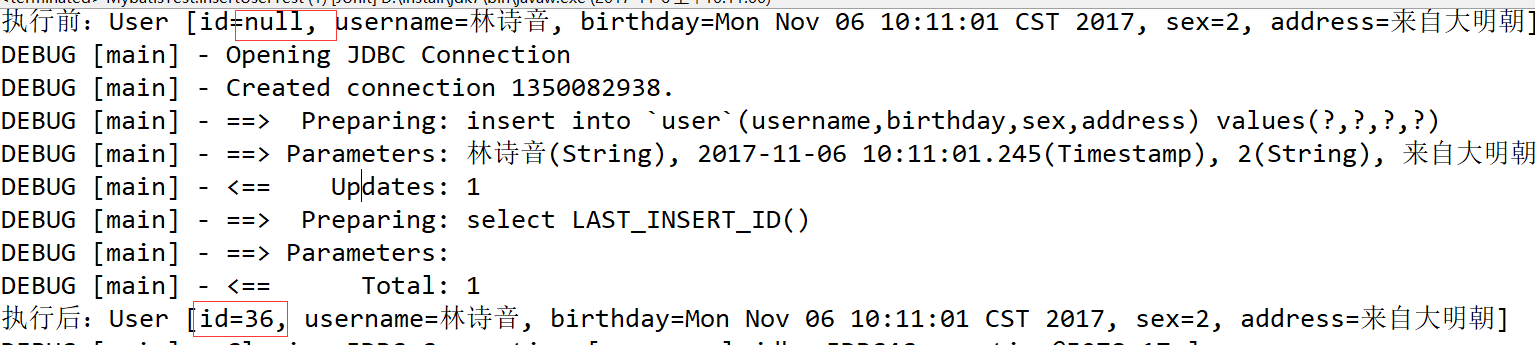
mysql数据库查询新增Id：

select LAST\_INSERT\_ID()

#### 获取新增的用户Id

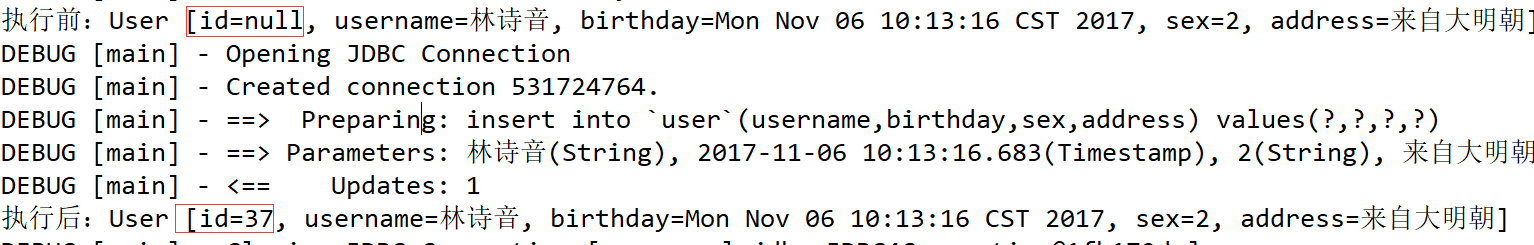
##### 方式一

|  |
| --- |
| <!-- 新增用户，说明  insert标签：用于放置新增sql语句  #{id}：占位符，当参数传递的是pojo的时候，换括号中的内容是pojo的属性-->  <insert id=*"insertUser"* parameterType=*"cn.itheima.mybatis.po.User"*>  <!-- insert into `user`(id,username,birthday,sex,address)  values(#{id},#{username},#{birthday},#{sex},#{address}) -->    <!-- selectKey标签：查询主键id，说明  keyColumn属性：主键字段（表）  keyProperty属性：主键属性（pojo）  resultType属性：指定主键字段类型  order属性：指定在insert语句之前，还是之后获取主键值.  BEFORE:在insert语句前；AFTER:在insert语句之后-->  <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

|  |
| --- |
| <!-- 新增用户，说明  insert标签：用于放置新增sql语句  #{id}：占位符，当参数传递的是pojo的时候，换括号中的内容是pojo的属性-->  <insert id=*"insertUser"* parameterType=*"cn.itheima.mybatis.po.User"*  useGeneratedKeys=*"true"* keyColumn=*"id"* keyProperty=*"id"*>  <!-- insert into `user`(id,username,birthday,sex,address)  values(#{id},#{username},#{birthday},#{sex},#{address}) -->    <!-- selectKey标签：查询主键id，说明  keyColumn属性：主键字段（表）  keyProperty属性：主键属性（pojo）  resultType属性：指定主键字段类型  order属性：指定在insert语句之前，还是之后获取主键值.  BEFORE:在insert语句前；AFTER:在insert语句之后-->  <!-- <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> |



### 根据用户Id修改用户

#### 准备sql语句

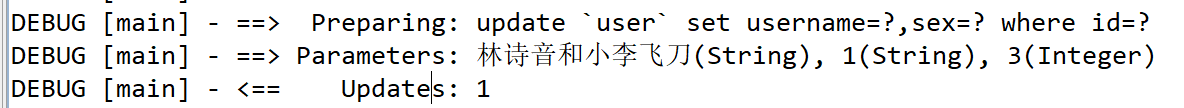
|  |
| --- |
| update `user` set username='小李飞刀和林诗音',sex='2' where id=2 |

#### 配置映射文件

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

#### 编写测试代码

|  |
| --- |
| // 4.测试根据用户Id修改用户  @Test  **public** **void** updateUserByIdTest(){  // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.使用sqlSession对象，调用方法执行  // update方法：修改记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  // 创建用户对象  User user = **new** User();  user.setId(3);  user.setUsername("林诗音和小李飞刀");  user.setSex("1");    sqlSession.update("test.updateUserById", user);    // 3.释放资源  sqlSession.close();  } |



### 根据用户Id删除用户

#### 准备sql语句

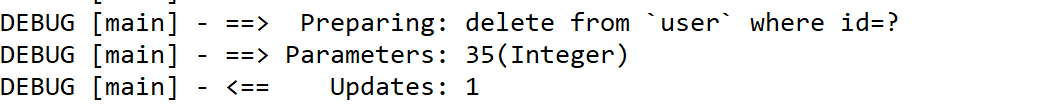
|  |
| --- |
| delete from `user` where id=4 |

#### 配置映射文件

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

#### 编写测试代码

|  |
| --- |
| // 5.测试根据用户Id删除用户  @Test  **public** **void** deleteUserByIdTest(){  // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.使用sqlSession对象，调用方法执行  // delete方法：删除记录  // statement参数：执行的sql语句（名称空间+sql语句Id）  // parameter参数：传入的参数值  sqlSession.delete("test.deleteUserById", 35);    // 3.释放资源  sqlSession.close();  } |



# mybatis框架原理

sqlMapConfig.xml：配置数据源，mapper映射文件。配置mybatis框架的运行时环境。

SqlSessionFactoryBuilder：读取核心配置文件，返回框架核心对象：SqlSessionFactory

SqlSessionFactory：创建SqlSession对象

SqlSession：发起数据库的CRUD操作

Executor对象：执行数据库的CRUD操作

MappedStatement：负责输入输出的映射

数据库：mysql

输入映射：

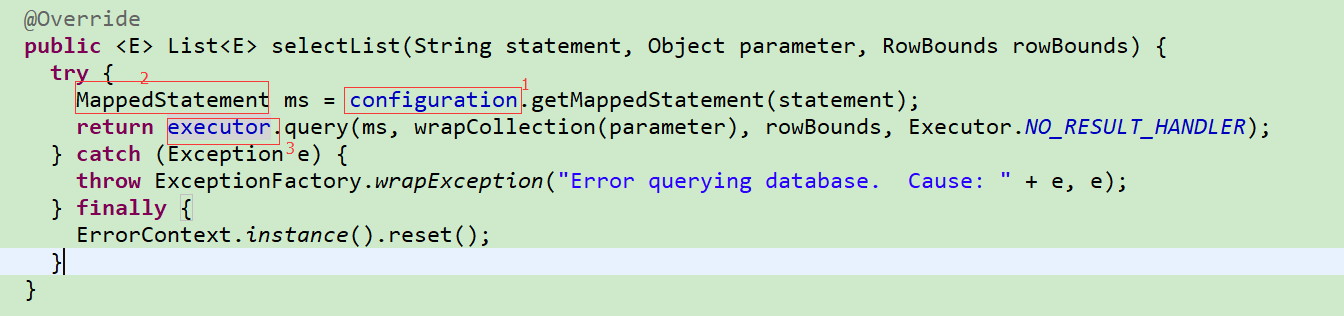
java简单类型

pojo

输出映射：

java简单类型

pojo



1. Configuration配置对象：sqlMapConfig.xl，User.xml......
2. MappedStatement对象：负责输入输出映射，ParameterType参数输入，ResultType返回值输出
3. Executor执行器对象：负责执行数据库的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

### 需求

1. 根据用户Id查询用户
2. 新增用户

### 需求实现

#### 准备用户dao接口

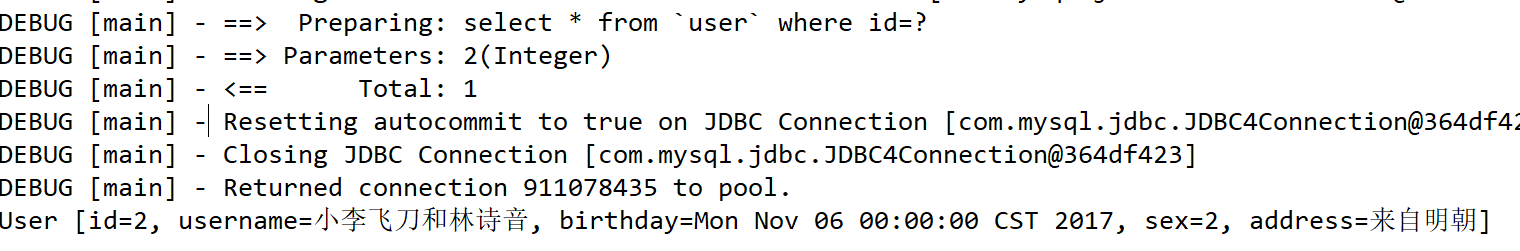
|  |
| --- |
| **public** **interface** UserDao {    // 1.根据用户Id查询用户  User queryUserById(Integer id);    // 2.新增用户  **void** insertUser(User user);  } |

#### 实现用户dao接口

|  |
| --- |
| **public** **class** UserDaoImpl **implements** UserDao {    // 定义sqlSessionFactory对象  **private** SqlSessionFactory sqlSessionFactory;  /\*\*  \*  \*/  **public** UserDaoImpl() {  **super**();  }  /\*\*  \* **@param** sqlSessionFactory  \*/  **public** UserDaoImpl(SqlSessionFactory sqlSessionFactory) {  **super**();  **this**.sqlSessionFactory = sqlSessionFactory;  }  /\* (non-Javadoc)  \* @see cn.itheima.mybatis.dao.UserDao#queryUserById(java.lang.Integer)  \*/  **public** User queryUserById(Integer id) {  // **TODO** Auto-generated method stub  // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.使用sqlSession对象，调用方法执行  Object user = sqlSession.selectOne("test.queryUserById", id);    // 3.释放资源  sqlSession.close();    **return** (User) user;  }  /\* (non-Javadoc)  \* @see cn.itheima.mybatis.dao.UserDao#insertUser(cn.itheima.mybatis.po.User)  \*/  **public** **void** insertUser(User user) {  // **TODO** Auto-generated method stub  // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.使用sqlSession对象，调用方法执行  sqlSession.insert("test.insertUser", user);    // 3.释放资源  sqlSession.close();  }  } |

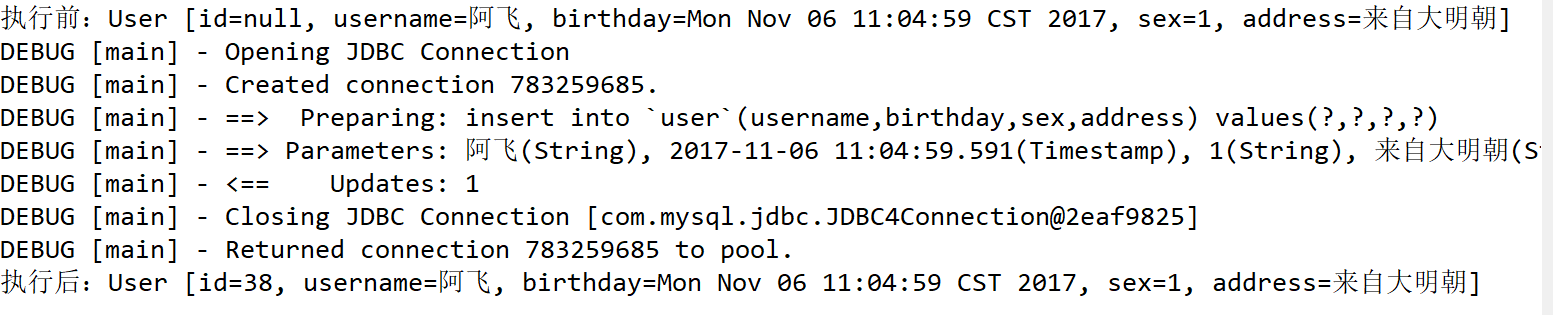
#### 编写测试代码

|  |
| --- |
| **public** **class** UserDaoTest {    **private** SqlSessionFactory sqlSessionFactory = **null**;    @Before  **public** **void** init() **throws** IOException{  // 1.加载核心配置文件：sqlMapConfig.xml  // resource属性：指定核心配置文件位置  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  // SqlSessionFactory对象：mybatis框架的核心对象，一个项目中只需要一个（单例设计模式）  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.使用userDao对象，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.创建用户dao对象  UserDao userDao = **new** UserDaoImpl(sqlSessionFactory);    // 2.使用userDao对象，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("来自大明朝");    System.*out*.println("执行前："+user);  userDao.insertUser(user);  System.*out*.println("执行后："+user);  } |



## mapper代理开发方法

### 注意事项

1. 要求mapper映射文件中namespace属性值，必须是mapper接口的全路径（包名称+接口名称）
2. 要求mapper映射文件中sql语句的声明，与mapper接口方法的声明一致

2.1.mapper接口返回值类型，与sql语句标签的resultType属性指定的类型一致

2.2.mapper接口的方法名称，与sql语句id属性值一致

2.3.mapper接口的方法的形参，与sql语句的parameterType属性指定的类型一致

### 需求

1. 根据用户Id查询用户
2. 新增用户

### 需求实现

#### 准备用户mapper接口

|  |
| --- |
| **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);  } |

#### 准备用户mapper映射文件

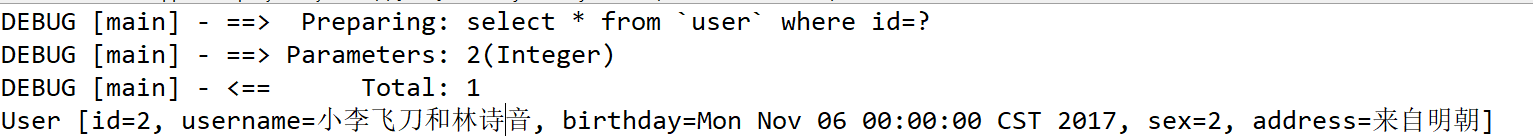
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- namespace属性：名称空间，相当于java中的package，用于放置sql语句冲突 -->  <!--mapper代理开发方法中，namespace属性值，必须是mapper接口的全路径 -->  <mapper namespace=*"cn.itheima.mybatis.mapper.UserMapper"*>  <!-- 根据用户Id查询用户，说明：  select标签：用于放置查询sql语句  id属性：唯一标识一条sql语句  parameterType属性：输入参数类型  resultType属性：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.mybatis.po.User"*>  select \* from `user` where id=#{id}  </select>    <!-- 新增用户，说明  insert标签：用于放置新增sql语句  #{id}：占位符，当参数传递的是pojo的时候，换括号中的内容是pojo的属性-->  <insert id=*"insertUser"* parameterType=*"cn.itheima.mybatis.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文件

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

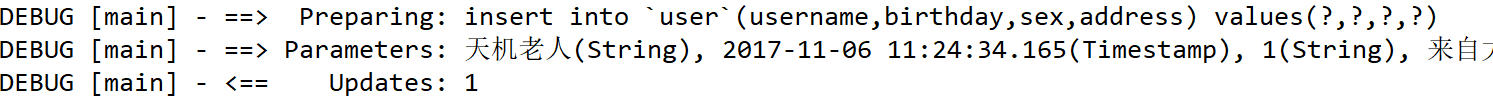
#### 编写测试代码

|  |
| --- |
| **public** **class** UserMapperTest {    **private** SqlSessionFactory sqlSessionFactory = **null**;    @Before  **public** **void** init() **throws** IOException{  // 1.加载核心配置文件：sqlMapConfig.xml  // resource属性：指定核心配置文件位置  InputStream inputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");    // 2.读取配置文件内容  SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  // SqlSessionFactory对象：mybatis框架的核心对象，一个项目中只需要一个（单例设计模式）  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.使用sqlSession对象，获取mapper代理对象  // getMapper方法：获取mapper代理对象  // type参数：mapper接口的class  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);    // 3.使用mapper代理对象，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对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession(**true**);    // 2.使用sqlSession对象，获取mapper代理对象  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);    // 3.使用mapper对象，调用方法执行  // 创建用户对象  User user = **new** User();  user.setUsername("天机老人");  user.setSex("1");  user.setBirthday(**new** Date());  user.setAddress("来自大明朝");    mapper.insertUser(user);    // 4.释放资源  sqlSession.close();  } |



## 两种开发方法总结

1. 原始的dao开发方法：需要开发接口和实现类
2. mapper代理开发方法：只需要开发接口
3. 企业项目中，推荐使用mapper代理开发

# mybatis核心配置文件

sqlMapConfig.xml：配置运行环境（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：加载属性文件（比如db.properties）和定义属性

### 准备db.properties文件

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

### 加载db.properties属性文件

|  |
| --- |
| <!--加载属性文件，说明：  1.resource属性：指定本地配置文件的位置  2.url属性：指定网络上的配置文件位置-->  <properties resource=*"db.properties"* >    </properties> |

### 使用db.propertis属性文件中的属性

|  |
| --- |
| <!-- 运行环境配置 -->  <!-- default属性：指定使用哪一个运行环境 -->  <environments default=*"development"*>    <!-- id属性：唯一标识一个运行环境 -->  <environment id=*"development"*>  <!-- 事务管理器配置，JDBC：mybatis默认使用jdbc事务 -->  <transactionManager type=*"JDBC"* />    <!-- 数据源配置，POOLED：mybatis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/mybatis" />  <property name="username" value="root" />  <property name="password" value="admin" /> -->    <!-- 使用db.properties属性文件的属性 -->  <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

|  |
| --- |
| <!--加载属性文件，说明：  1.resource属性：指定本地配置文件的位置  2.url属性：指定网络上的配置文件位置  3.使用内部property标签定义属性  4.加载顺序：首先加载内部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，属性文件中的属性，覆盖内部property标签的属性-->  <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

#### 自定义别名方式一

|  |
| --- |
| <!-- 自定义别名配置 -->  <typeAliases>  <!-- 自定义用户别名配置，说明：  1.type属性：别名的类型  2.默认使用类名称，作为别名  3.alias属性：指定别名的名称 -->  <typeAlias type=*"cn.itheima.mybatis.po.User"* alias=*"user"*/>  </typeAliases> |

#### 自定义别名方式二

|  |
| --- |
| <!-- 自定义别名配置 -->  <typeAliases>  <!-- 自定义用户别名配置，说明：  1.type属性：别名的类型  2.默认使用类名称，作为别名  3.alias属性：指定别名的名称 -->  <!-- <typeAlias type="cn.itheima.mybatis.po.User" alias="user"/> -->    <!--包扫描方式配置别名，说明：  1.name属性：指定扫描的包  2.默认使用类名称，作为别名  3.如果有多个包，配置多个package  4.企业项目中，推荐使用包扫描方式 -->  <package name=*"cn.itheima.mybatis.po"*/>  </typeAliases> |

## mappers（映射器）

作用：加载mapper映射文件。

### 加载mapper映射文件方式一

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

### 加载mapper映射文件方式二

|  |
| --- |
| <!-- 加载mapper映射文件 -->  <mappers>  <!--加载User.xml文件，说明：  resource属性：指定配置文件的位置-->  <mapper resource=*"sqlmap/User.xml"*/>    <!-- 加载UserMapper.xml文件 -->  <!-- <mapper resource="sqlmap/UserMapper.xml"/> -->    <!-- 包扫描方式加载mapper映射文件，说明：  1.前提必须是mapper代理开发方法  2.要求mapper映射文件，与mapper接口在同一个目录  3.要求mapper映射文件的名称，与mapper接口的名称一致 -->  <package name=*"cn.itheima.mybatis.mapper"*/>  </mappers> |

# mybatis与hibernate比较

## 相同点

都是对jdbc的封装，都是持久层的框架，都用于dao层的开发。

## 不同点

1. hibernate是对sql语句做了封装，提供HQL语言操作数据库，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，代码开发量较少，sql语句优化困难；mybatis是直接使用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，代码开发量较多，sql语句优化容易。
2. hibernate是配置java对象与数据库表的对应关系，多表关联关系配置复杂。mybatis是配置java对象，与sql语句对应关系，多表关联关系配置容易。
3. hibernate是重量级的框架，学习使用门槛高，适合于需求相对稳定，中小型的项目，比如：办公自动化系统（OA）；mybatis是轻量级的框架，学习使用门槛低，适合于需求变化频繁，大型的项目，比如：互联网电子商务网站项目。目前在企业项目中，mybatis框架使用更多一些。

# 扩展

## 注解开发

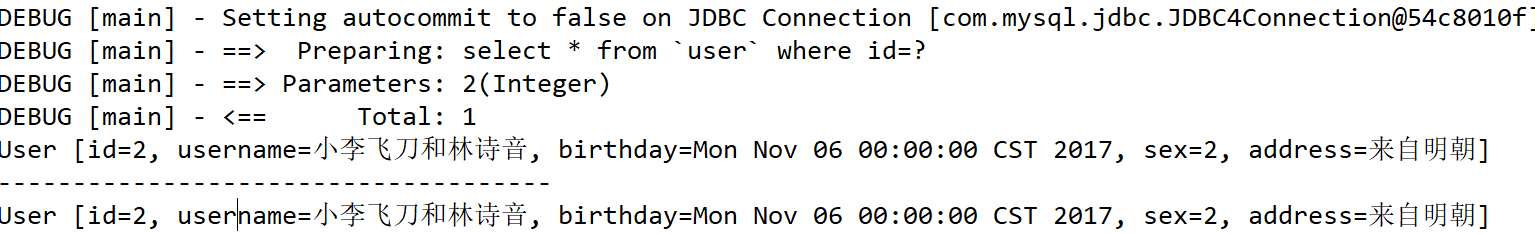
|  |
| --- |
| // 3.根据用户Id查询用户，使用注解开发实现  @Select("select \* from `user` where id=#{id}")  User queryUserById\_1(Integer id); |
| // 测试根据用户Id查询用户,测试注解开发  @Test  **public** **void** queryUserById\_1Test(){    // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.使用sqlSession对象，获取mapper代理对象  // getMapper方法：获取mapper代理对象  // type参数：mapper接口的class  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);    // 3.使用mapper代理对象，调用方法执行  User user = mapper.queryUserById\_1(2);  System.*out*.println(user);    // 4.释放资源  sqlSession.close();  } |

说明：注解开发，只做了解，企业项目中不推荐使用。

## 缓存

### 一级缓存（sqlSession级别）

|  |
| --- |
| // 测试根据用户Id查询用户,测试一级缓存  @Test  **public** **void** oneCacheTest(){    // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.使用sqlSession对象，获取mapper代理对象  // getMapper方法：获取mapper代理对象  // type参数：mapper接口的class  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);    // 3.使用mapper代理对象，调用方法执行  User user = mapper.queryUserById(2);  System.*out*.println(user);    System.*out*.println("-------------------------------------");    User user1 = mapper.queryUserById(2);  System.*out*.println(user1);      // 4.释放资源  sqlSession.close();  } |



说明：一级缓存mybatis框架就已经默认使用，不需要关心。

### 二级缓存（mapper级别缓存，在多个SqlSession之间共享）

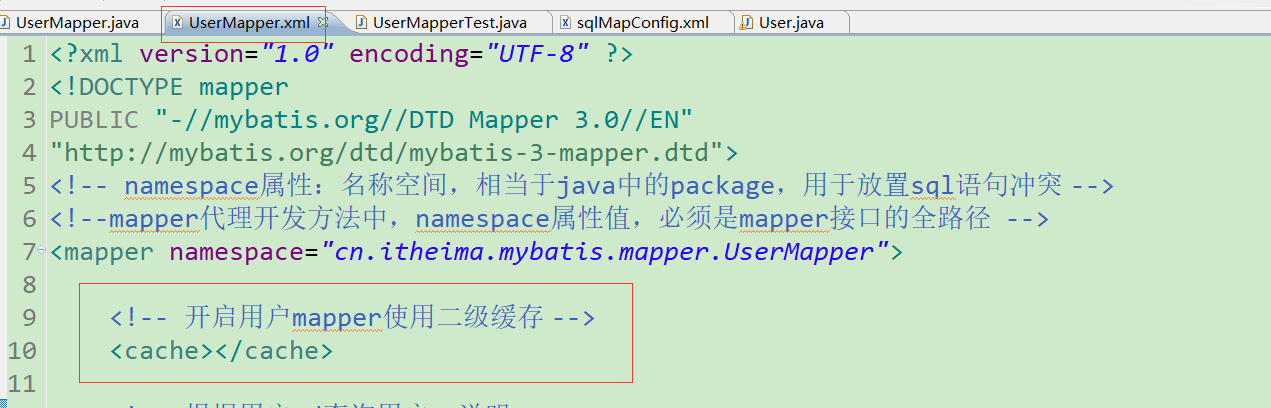
#### 在sqlMapConfig.xml中，配置开启二级缓存

|  |
| --- |
| <!-- 全局参数配置 -->  <settings>  <!-- 配置开启二级缓存 -->  <setting name=*"cacheEnabled"* value=*"true"*/>  </settings> |

#### 修改用户pojo，实现序列化接口

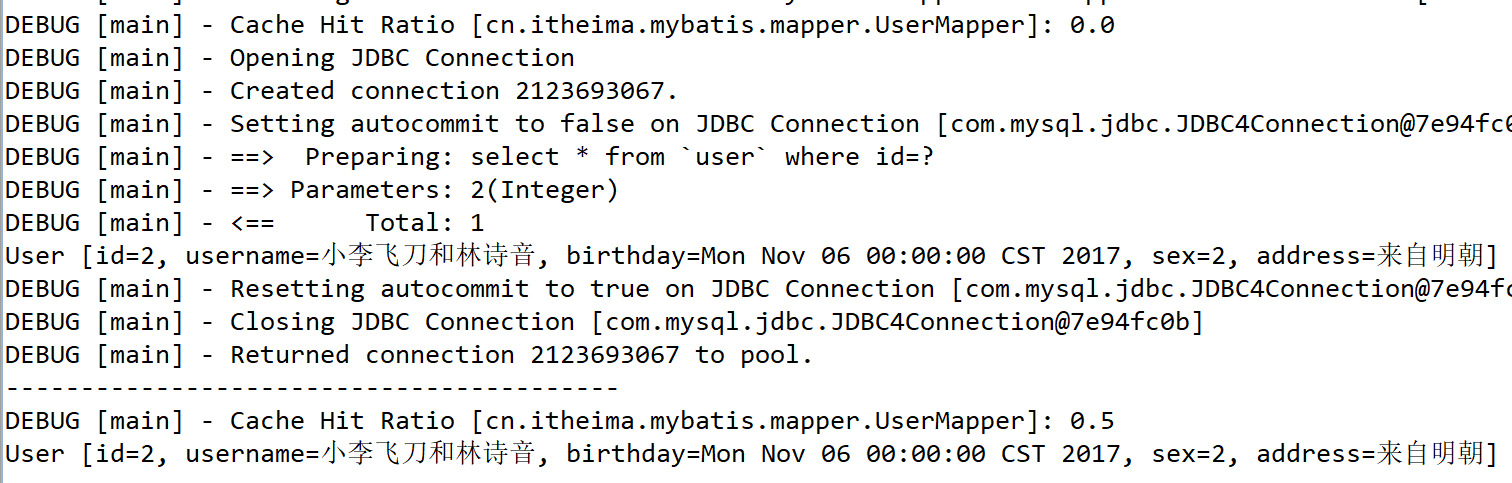
|  |
| --- |
| **public** **class** User **implements** java.io.Serializable{    **private** Integer id; // int(11) NOT NULL AUTO\_INCREMENT,  **private** String username; // varchar(32) NOT NULL COMMENT '用户名称',  **private** Date birthday; // date DEFAULT NULL COMMENT '生日',  **private** String sex; // char(1) DEFAULT NULL COMMENT '性别',  **private** String address; // varchar(256) DEFAULT NULL COMMENT '地址', |

#### 在UserMapper.xml中，开启使用二级缓存



#### 测试二级缓存

|  |
| --- |
| // 测试根据用户Id查询用户,测试二级缓存  @Test  **public** **void** twoCacheTest(){    // 1.创建sqlSession对象  SqlSession sqlSession = **this**.sqlSessionFactory.openSession();    // 2.使用sqlSession对象，获取mapper代理对象  // getMapper方法：获取mapper代理对象  // type参数：mapper接口的class  UserMapper mapper = sqlSession.getMapper(UserMapper.**class**);    // 3.使用mapper代理对象，调用方法执行  User user = mapper.queryUserById(2);  System.*out*.println(user);  // 4.释放资源  sqlSession.close();    System.*out*.println("-----------------------------------------");    // 5.创建sqlSession对象  SqlSession sqlSession1 = **this**.sqlSessionFactory.openSession();    // 6.使用sqlSession对象，获取mapper代理对象  // getMapper方法：获取mapper代理对象  // type参数：mapper接口的class  UserMapper mapper1 = sqlSession1.getMapper(UserMapper.**class**);    // 7.使用mapper代理对象，调用方法执行  User user1 = mapper1.queryUserById(2);  System.*out*.println(user1);  // 8.释放资源  sqlSession1.close();  } |



说明：mybatis框架的二级缓存，只做了解，企业项目中不推荐使用。原因是mybatis框架的二级缓存不能实现细粒度的控制。