# mybatis入门程序

## 需求

实现用户查询：

根据用户id（主键）查询用户信息（单条记录）

根据用户名称模糊查询用户信息（多条记录）

用户添加

用户删除

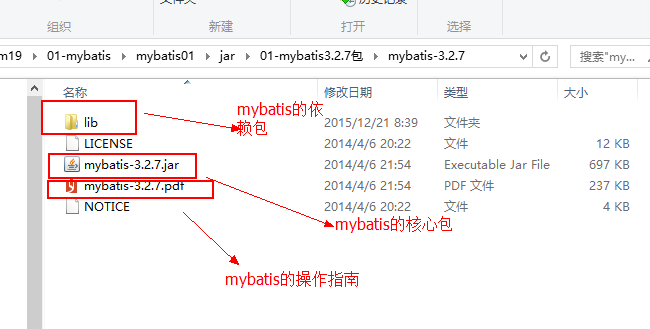
用户修改

## 环境准备

* Jdk：1.7
* Ide：eclipse indigo
* Mybatis：3.2.7
* 数据库：MySQL 5X

### 下载mybatis

mybaits的代码由github.com管理，下载地址：<https://github.com/mybatis/mybatis-3/releases>



### 数据库脚本初始化

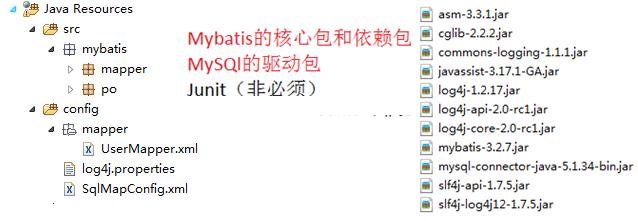


通常需要提供初始化数据的数据库脚本。

1. 执行sql\_table.sql脚本，创建数据库表；
2. 执行sql\_data.sql初始化测试数据。

## 工程搭建

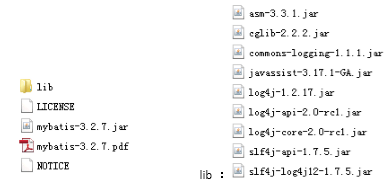
### 工程结构图



### 导入jar包

从mybatis管网下载（地址：https://github.com/mybatis/mybatis-3/releases）

mybatis-3.2.7.pdf---操作手册 ；mybatis-3.2.7.jar--核心 jar包 ；依赖的jar包



### log4j.properties（公用文件）--日志文件

Mybatis使用的日志包是log4j的，所以需要添加log4j.properties。

在classpath下创建log4j.properties如下：

文件内容可以从mybatis-3.2.7.pdf中拷贝

# Global logging configuration，建议开发环境中要用debug

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

日志级别在开发阶段设置成DEBUG，在生产阶段设置成INFO或者ERROR。

## 编程步骤

1. 创建PO类，根据需求创建；
2. 创建全局配置文件SqlMapConfig.xml；
3. 编写映射文件User.xml或UserMapper.xml；
4. 加载映射文件，在SqlMapConfig.xml中进行加载；
5. 编写测试程序，即编写Java代码，连接并操作数据库。

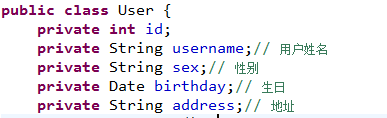
思路：

1. 读取配置文件；
2. 通过SqlSessionFactoryBuilder创建SqlSessionFactory会话工厂。
3. 通过SqlSessionFactory创建SqlSession。
4. 调用SqlSession的操作数据库方法。

关闭SqlSession。

## 代码实现

### 创建po类，（User.java）



### SqlMapConfig.xml(全局配置文件)

在classpath下，创建SqlMapConfig.xml文件

SqlMapConfig.xml（文件头可以从mybatis-3.2.7.pdf文档的2.1.2小节中拷贝）：

在config目录下，创建SqlMapConfig.xml文件，该名称不是固定不变的。

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 和spring整合后 environments配置将废除-->

<environments default=*"development"*>

<environment id=*"development"*>

<!-- 使用jdbc事务管理-->

<transactionManager type=*"JDBC"* />

<!-- 数据库连接池-->

<dataSource type=*"POOLED"*>

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

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"* />

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

<property name=*"password"* value=*"root"* />

</dataSource>

</environment>

</environments>

<!-- 加载mapper.xml -->

<mappers>

<mapper resource=*"sqlmap/User.xml"* />

</mappers>

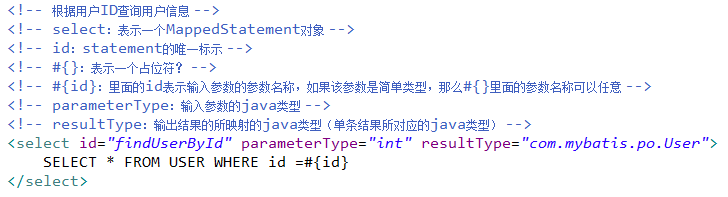
</configuration>

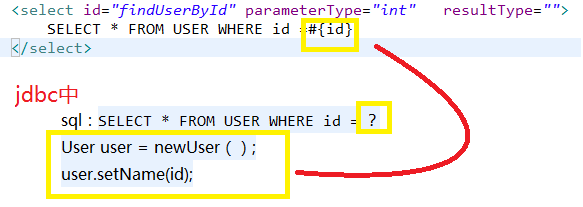
# crud

## 根据用户ID查询用户信息

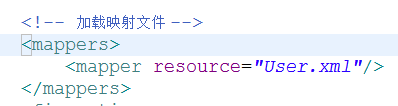
### 映射文件

在config目录下，创建User.xml（这种命名规范是由ibatis遗留下来）





### 在全局配置文件中加载映射文件

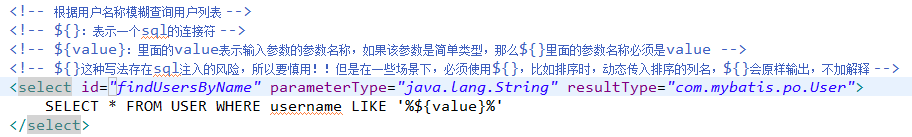


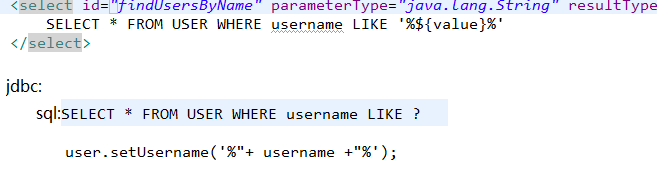
### 测试代码



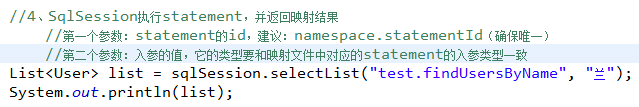
## 根据用户名称模糊查询用户列表

### 映射文件





### 测试代码



// 测试根据username模糊查询用户得到（单条数据）

@Test

**public** **void** testFindUserByName() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

// 第一个参数:statemrnt位置：namespace + statemrnt的id

// 第二个参数：传入的参数

List<User> user = **null**;

**try** {

user = sqlSession.selectList("test.findUserByName", "%小明%");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// 关闭sqlSession

sqlSession.close();

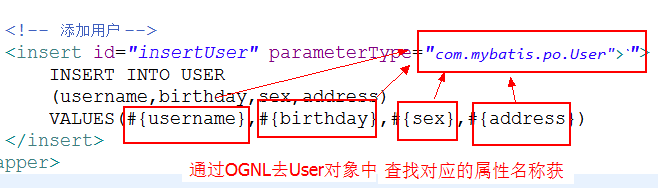
}

System.*out*.println(user);

}

## 添加用户

### 映射文件



<!-- 添加用户

parameterType：输入 参数的类型，User对象 包括 username,birthday,sex,address

#{}接收pojo数据，可以使用OGNL解析出pojo的属性值

#{username}表示从parameterType中获取pojo的属性值

-->

<insert id=*"insertUser"* parameterType=*"mybatis.po.User"*>

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

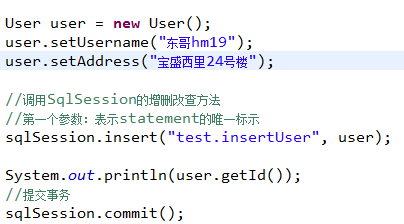
select LAST\_INSERT\_ID()

</selectKey>

INSERT INTO USER(username,birthday,sex,address) VALUES(#{username},#{birthday},#{sex},#{address})

</insert>

### 测试代码



// 插入数据

@Test

**public** **void** testInsertUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

//创建插入数据对象

User user = **new** User();

user.setUsername("小兰");

user.setAddress("四川内江");

user.setBirthday(**new** Date());

user.setSex("1");

**try** {

sqlSession.insert("test.insertUser", user);

//需要提交事务

sqlSession.commit();

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// 关闭sqlSession

sqlSession.close();

}

System.*out*.println(user);

}

### 主键返回之自增主键

selectKey：用于进行主键返回，定义了获取主键值的sql

order：设置selectKey中sql执行的顺序，相对于insert语句来说

keyProperty：将主键值设置到哪个属性

resultType：select LAST\_INSERT\_ID()的结果 类型

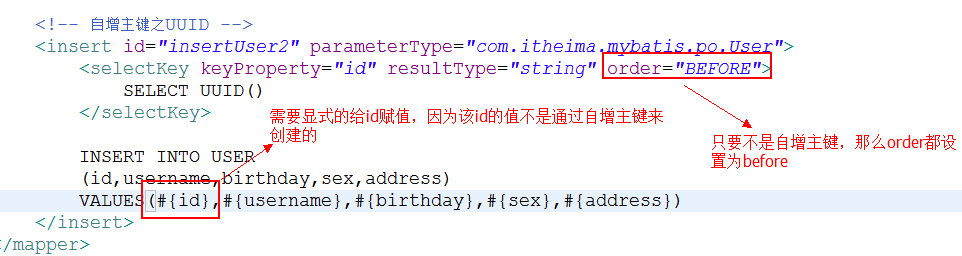


### 主键返回值UUID

UUID函数是mysql的函数

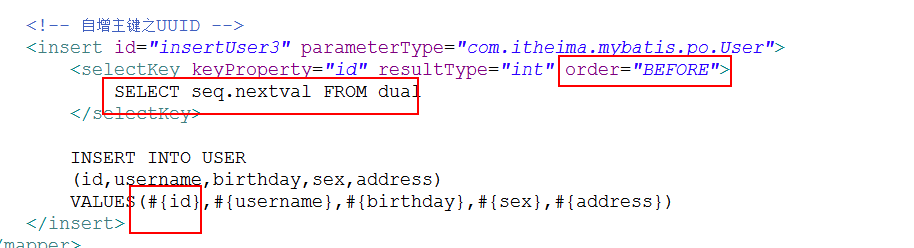
1.使用mysql的uuid机制生成主键：

使用uuid生成主键的好处是不考虑数据库移植后主键冲突问题。



### 主键返回值序列--orcale

序列也就是sequence，它是Oracle的主键生成策略



## 用户删除

### 映射文件

<!-- 用户删除 -->

<delete id=*"deleteUser"* parameterType=*"int"*>

delete from user where id=#{id}

</delete>

### 测试代码

// 测试根据id删除用户(得到单条记录)

@Test

**public** **void** testDeleteUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

**try** {

sqlSession.delete("test.deleteUser", 24);

// 需要提交事务

sqlSession.commit();

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// 关闭sqlSession

sqlSession.close();

}

}

## 用户更新

### 映射文件

<!-- 用户更新 要求：传入的user对象中包括 id属性值 -->

<update id=*"updateUser"* parameterType=*"mybatis.po.User"*>

update user set

username=#{username},birthday=#{birthday},sex=#{sex},address=#{address}

where id=#{id}

</update>

### 测试代码

// 测试根据id更新用户(得到单条记录)

@Test

**public** **void** testUpdateUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

// 创建更新数据对象，要求必须包括 id

User user = **new** User();

user.setId(1);

user.setUsername("燕青");

user.setAddress("河南郑州");

// user.setBirthday(new Date());

user.setSex("1");

**try** {

sqlSession.update("test.updateUser", user);

// 需要提交事务

sqlSession.commit();

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// 关闭sqlSession

sqlSession.close();

}

System.*out*.println("用户的id=" + user.getId());

}

# 小结

## #{}和${}



## 输入输出类型

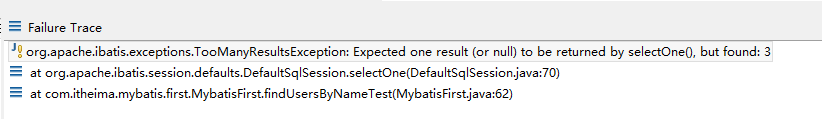


## selectOne和selectList

selectOne查询单个对象

用于查询单条记录，不能用于查询多条记录，否则异常：

org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 4



selectList查询集合对象