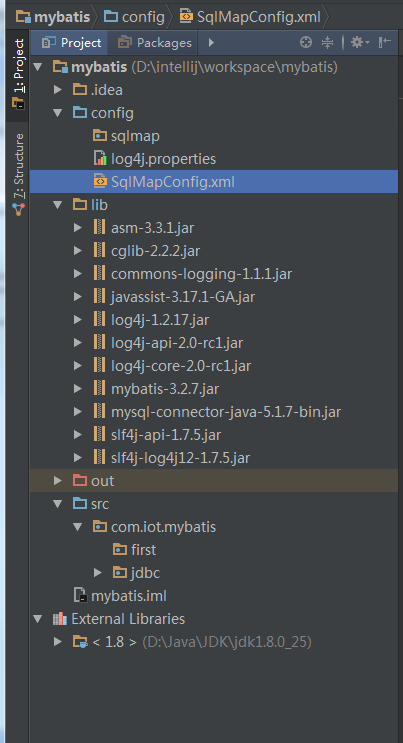
mybatis学习笔记(3)-入门程序一

## 工程结构

在IDEA中新建了一个普通的java项目，新建文件夹lib,加入jar包,工程结构如图。



* 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

* SqlMapConfig.xml

<?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>

<!-- 和spring整合后 environments配置将废除-->

<environments default="development">

<environment id="development">

<!-- 使用jdbc事务管理，事务控制由mybatis-->

<transactionManager type="JDBC" />

<!-- 数据库连接池,由mybatis管理-->

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://120.25.162.238:3306/mybatis001?characterEncoding=utf-8" />

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

<property name="password" value="123" />

</dataSource>

</environment>

</environments>

</configuration>

## 映射文件

* sqlmap/User.xml

<?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 命名空间，作用就是对sql进行分类化管理,理解为sql隔离

注意：使用mapper代理方法开发，namespace有特殊重要的作用

-->

<mapper namespace="test">

<!-- 在映射文件中配置很多sql语句 -->

<!--需求:通过id查询用户表的记录 -->

<!-- 通过select执行数据库查询

id:标识映射文件中的sql，称为statement的id

将sql语句封装到mappedStatement对象中，所以将id称为statement的id

parameterType:指定输入参数的类型

#{}标示一个占位符,

#{value}其中value表示接收输入参数的名称，如果输入参数是简单类型，那么#{}中的值可以任意。

resultType：指定sql输出结果的映射的java对象类型，select指定resultType表示将单条记录映射成java对象

-->

<select id="findUserById" parameterType="int" resultType="com.iot.mybatis.po.User">

SELECT \* FROM user WHERE id=#{value}

</select>

<!-- 根据用户名称模糊查询用户信息，可能返回多条

resultType：指定就是单条记录所映射的java对象类型

${}:表示拼接sql串，将接收到参数的内容不加任何修饰拼接在sql中。

使用${}拼接sql，引起 sql注入

${value}：接收输入参数的内容，如果传入类型是简单类型，${}中只能使用value

-->

<select id="findUserByName" parameterType="java.lang.String" resultType="com.iot.mybatis.po.User">

SELECT \* FROM user WHERE username LIKE '%${value}%'

</select>

</mapper>

在sqlMapConfig.xml中加载User.xml

<!-- 加载映射文件-->

<mappers>

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

</mappers>

## 程序代码

* po类User.java

package com.iot.mybatis.po;

import java.util.Date;

/\*\*

\* Created by Administrator on 2016/2/21.

\*/

public class User {

//属性名要和数据库表的字段对应

private int id;

private String username;// 用户姓名

private String sex;// 性别

private Date birthday;// 生日

private String address;// 地址

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

@Override

public String toString() {

return "User [id=" + id + ", username=" + username + ", sex=" + sex

+ ", birthday=" + birthday + ", address=" + address + "]";

}

}

* 测试代码

package com.iot.mybatis.first;

import com.iot.mybatis.po.User;

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 java.io.IOException;

import java.io.InputStream;

import java.util.List;

/\*\*

\* Created by Administrator on 2016/2/23.

\*/

public class MybatisFirst {

//根据id查询用户信息，得到一条记录结果

@Test

public void findUserByIdTest() throws IOException{

// mybatis配置文件

String resource = "SqlMapConfig.xml";

// 得到配置文件流

InputStream inputStream = Resources.getResourceAsStream(resource);

//创建会话工厂，传入mybatis配置文件的信息

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

// 通过工厂得到SqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过SqlSession操作数据库

// 第一个参数：映射文件中statement的id，等于=namespace+"."+statement的id

// 第二个参数：指定和映射文件中所匹配的parameterType类型的参数

// sqlSession.selectOne结果 是与映射文件中所匹配的resultType类型的对象

// selectOne查询出一条记录

User user = sqlSession.selectOne("test.findUserById", 1);

System.out.println(user);

// 释放资源

sqlSession.close();

}

// 根据用户名称模糊查询用户列表

@Test

public void findUserByNameTest() throws IOException {

// mybatis配置文件

String resource = "SqlMapConfig.xml";

// 得到配置文件流

InputStream inputStream = Resources.getResourceAsStream(resource);

// 创建会话工厂，传入mybatis的配置文件信息

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder()

.build(inputStream);

// 通过工厂得到SqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// list中的user和映射文件中resultType所指定的类型一致

List<User> list = sqlSession.selectList("test.findUserByName", "小明");

System.out.println(list);

sqlSession.close();

}

}

输出：

* findUserByIdTest()

DEBUG [main] - Logging initialized using 'class org.apache.ibatis.logging.slf4j.Slf4jImpl' adapter.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - Opening JDBC Connection

DEBUG [main] - Created connection 1857815974.

DEBUG [main] - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@6ebc05a6]

DEBUG [main] - ==> Preparing: SELECT \* FROM user WHERE id=?

DEBUG [main] - ==> Parameters: 1(Integer)

DEBUG [main] - <== Total: 1

User [id=1, username=王五, sex=2, birthday=null, address=null]

DEBUG [main] - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@6ebc05a6]

DEBUG [main] - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@6ebc05a6]

DEBUG [main] - Returned connection 1857815974 to pool.

* findUserByNameTest()

DEBUG [main] - Logging initialized using 'class org.apache.ibatis.logging.slf4j.Slf4jImpl' adapter.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - PooledDataSource forcefully closed/removed all connections.

DEBUG [main] - Opening JDBC Connection

DEBUG [main] - Created connection 1596467899.

DEBUG [main] - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@5f282abb]

DEBUG [main] - ==> Preparing: SELECT \* FROM user WHERE username LIKE '%小明%'

DEBUG [main] - ==> Parameters:

DEBUG [main] - <== Total: 3

[User [id=16, username=张小明, sex=1, birthday=null, address=河南郑州], User [id=22, username=陈小明, sex=1, birthday=null, address=河南郑州], User [id=25, username=陈小明, sex=1, birthday=null, address=河南郑州]]

DEBUG [main] - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@5f282abb]

DEBUG [main] - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@5f282abb]

DEBUG [main] - Returned connection 1596467899 to pool.

## 总结

* parameterType

在映射文件中通过parameterType指定输入参数的类型

* resultType

在映射文件中通过resultType指定输出结果的类型

* #{}和${}   
  #{}表示一个占位符号;   
  ${}表示一个拼接符号，会引起sql注入，所以不建议使用
* selectOne和selectList   
  selectOne表示查询一条记录进行映射，使用selectList也可以使用，只不过只有一个对象   
  selectList表示查询出一个列表(参数记录)进行映射，不嗯能够使用selectOne查，不然会报下面的错:

org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 3