Mybatis

1. Mybatis简介：

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

特点：

1. MyBatis 是支持定制化 SQL、存储过程以及高级映射（sql语句的返回值不能直接传到一个对象或者集合中时，利用高级映射）的优秀的持久层框架

2） MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集（手写一大堆重复无意义的代码）

3） MyBatis可以使用简单的XML或注解用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录

1. MyBatis 是一个 半自动的ORM（Object Relation Mapping）框架

Mybatis相比于JDBC和Hibernate的优点：

轻量级，性能出色

SQL 和 Java 编码分开，功能边界清晰，比JDBC优异。Java代码专注业务、SQL语句专注数据

开发效率稍逊于HIbernate，但是完全能够接受

二．搭建环境（增删改操作）

**1.环境（创建项目，修改jdk，maven版本）**

**创建maven工程：导入相关依赖**

<dependencies>

<!-- Mybatis核心 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.5.7</version>

</dependency>

<!-- junit测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<!-- MySQL驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.3</version>

</dependency>

</dependencies>

**2.配置Mybatis核心配置文件**

习惯上命名为mybatis-config.xml，找官方文档，然后粘贴配置文件configuration

environments标签用来配置数据库的环境；

<?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>

<!--配置数据库环境-->

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="${driver}"/>

<property name="url" value="${url}"/>

<property name="username" value="${username}"/>

<property name="password" value="${password}"/>

</dataSource>

</environment>

</environments>

<!--引入映射文件-->

<mappers>

<mapper resource="org/mybatis/example/BlogMapper.xml"/>

</mappers>

</configuration>

**3.创建mapper接口：**

MyBatis中的mapper接口相当于以前的dao。但是区别在于，mapper仅仅是接口，我们不需要提供实现类。

**4.创建Mybatis的映射文件**

ORM（Object Relationship Mapping）对象关系映射

分别指对象，关系，映射

对象：Java的实体类对象

关系：关系型数据库

映射：二者之间的对应关系

Java类与数据库表的映射关系如下：

类 表

属性 字段/列

对象 记录/行

1、映射文件的命名规则：

表所对应的实体类的类名+Mapper.xml

例如：表t\_user，映射的实体类为User，所对应的映射文件为UserMapper.xml

2.具体步骤：建立一个数据库表，然后建立它的实体类，然后建立表和实体类之间的映射关系，映射关系写法：建立一个映射接口，调用里面的方法来完成数据库操作，但是将方法中的sql语句写在它的xml配置文件里，这需要保持两个一致：

/\*\*

\* Mybatis面向接口编程的两个一致：

\* 1.映射文件的namespace和当前接口的名一致

\* 2.映射文件中sql语句的id与方法名一致

\*/

**5.将映射关系引入核心配置文件**

注意Resources文件夹下面的目录访问是路径格式，用/不用.

**6.写测试类，测试**

@Test

public void testMybatis() throws IOException {

//mybatis提供了一个操控数据库的会话对象sqlsession

//获取核心配置文件

InputStream is =Resources.getResourceAsStream("mybatis-config.xml");

//获取SqlSessionFactoryBuilder构建SqlSession对象

SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

//获取SqlSessionFactory,由SqlSessionFactoryBuilder创建的工厂类

SqlSessionFactory sqlSessionFactory=sqlSessionFactoryBuilder.build(is);

//从SqlSessionFactory工厂中获取操控数据库的会话对象sqlsession

//openSession(true)方法里面设置true就可以自动提交事务

SqlSession sqlSession=sqlSessionFactory.openSession(true);

//获取Mapper接口对象,getMapper方法可以返回接口的实现类的对象，该方法采用代理模式

UserMapper userMapper=sqlSession.getMapper(UserMapper.class);

//测试功能

int result=userMapper.insertUser();

// //使用JDBC方式提交事务

// sqlSession.commit();

System.out.println("result:"+result);

}

**7.加入log4j日志功能，并创建日志配置文件**

导入依赖：

<!-- log4j日志 -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

配置文件：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

<appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">

<param name="Encoding" value="UTF-8" />

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%-5p %d{MM-dd HH:mm:ss,SSS} %m (%F:%L) \n" />

</layout>

</appender>

<logger name="java.sql">

<level value="debug" />

</logger>

<logger name="org.apache.ibatis">

<level value="info" />

</logger>

<root>

<level value="debug" />

<appender-ref ref="STDOUT" />

</root>

</log4j:configuration>

日志的级别

FATAL(致命)>ERROR(错误)>WARN(警告)>INFO(信息)>DEBUG(调试)

从左到右打印的内容越来越详细，后边的包含前边的



复习搭建mybatis环境：

建立maven工程之后，

1.首先导入mybatis所需要的依赖，包含（mybatis核心，junit测试，mysql驱动，log4j日志）

2.配置mybatis核心配置文件，包含两部分，（连接数据库，使用连接池的方式，映射文件的位置），位置在resources目录下，访问时注意Resources文件夹下面的目录访问是路径格式，用/不用.

1. 建立与数据库中的表格相对应的实体类，创建一个映射接口用来写要执行的操作的方法
2. 写映射接口的配置文件，里面执行接口中的方法的具体内容，写log4j的日志配置文件

5.最后写测试类进行测试

三．进行查询功能

<!--查询功能必须设置标签，

resultType设置默认的映射关系，即设置之后，就会将当前处理的结果返回成当前resultType类型

resultMap设置自定义的映射关系，当字段名和对应的实体类中的属性名不一致时使用

-->

<select id="getUserById" resultType="com.libaba.mybatis.pojo.User">

select \* from t\_user where id='3';

</select>

<select id="getAllUser" resultType="com.libaba.mybatis.pojo.User">

select \* from t\_user;

</select>

测试类：

List<User> list=userMapper.getAllUser();

list.forEach(user1 -> System.out.println(user1));

1. 核心配置文件的解读

transactionManager：设置事务管理方式

属性：

type：设置事务管理方式，type="JDBC|MANAGED"

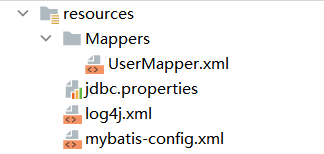
type="JDBC"：设置当前环境的事务管理都必须手动处理

type="MANAGED"：设置事务被管理，例如spring中的AOP

dataSource：设置数据源 属性： type：设置数据源的类型，type="POOLED|UNPOOLED|JNDI" type="POOLED"：使用数据库连接池，即会将创建的连接进行缓存，下次使用可以从 缓存中直接获取，不需要重新创建 type="UNPOOLED"：不使用数据库连接池，即每次使用连接都需要重新创建 type="JNDI"：调用上下文中的数据源

**使用properties文件引入方式：**

新建一个properties文件：



然后编辑：

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8

jdbc.username=root

jdbc.password=libaba

然后在mybatis-config.xml配置文件中引入该文件，：

<properties resource="jdbc.properties"></properties>

<!--配置数据库环境-->

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="${jdbc.driver}"/>

<property name="url" value="${jdbc.url}"/>

<property name="username" value="${jdbc.username}"/>

<property name="password" value="${jdbc.password}"/>

</dataSource>

</environment>

</environments>

**类型别名：**

<!--

mybatis核心配置文件中的标签顺序

properties?,settings?,typeAliases?,

typeHandlers?,objectFactory?,objectWrapperFactory?,

reflectorFactory?,plugins?,environments?,databaseIdProvider?,mappers?

-->

<!--设置类型别名,不区分大小写-->

<typeAliases>

<typeAlias type="com.libaba.mybatis.pojo.User" alias="User"></typeAlias>

<!--type:设置别名的类

alias：设置的别名，默认别名为类名-->

<!--以包名为单位设置统一的别名，即为类名且不区分大小写-->

<package name="com.libaba.mybatis.pojo"/>

</typeAliases>

然后在UserMapper.xml映射文件中就可以使用类型别名：

<select id="getAllUser" resultType="user">

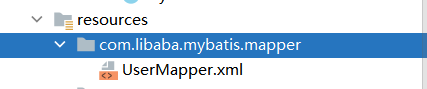
select \* from t\_user;

</select>

**mappers标签映入映射文件包：**

在resources目录下新建一个包：

必须用com/libaba/mybatis/mapper



<mappers>

<!--引入以包为单位的文件

mapper接口所在的包必须和映射文件所在的包名一致

mapper接口和映射文件的名字一致

-->

<package name="com.libaba.mybatis.mapper"/>

</mappers>

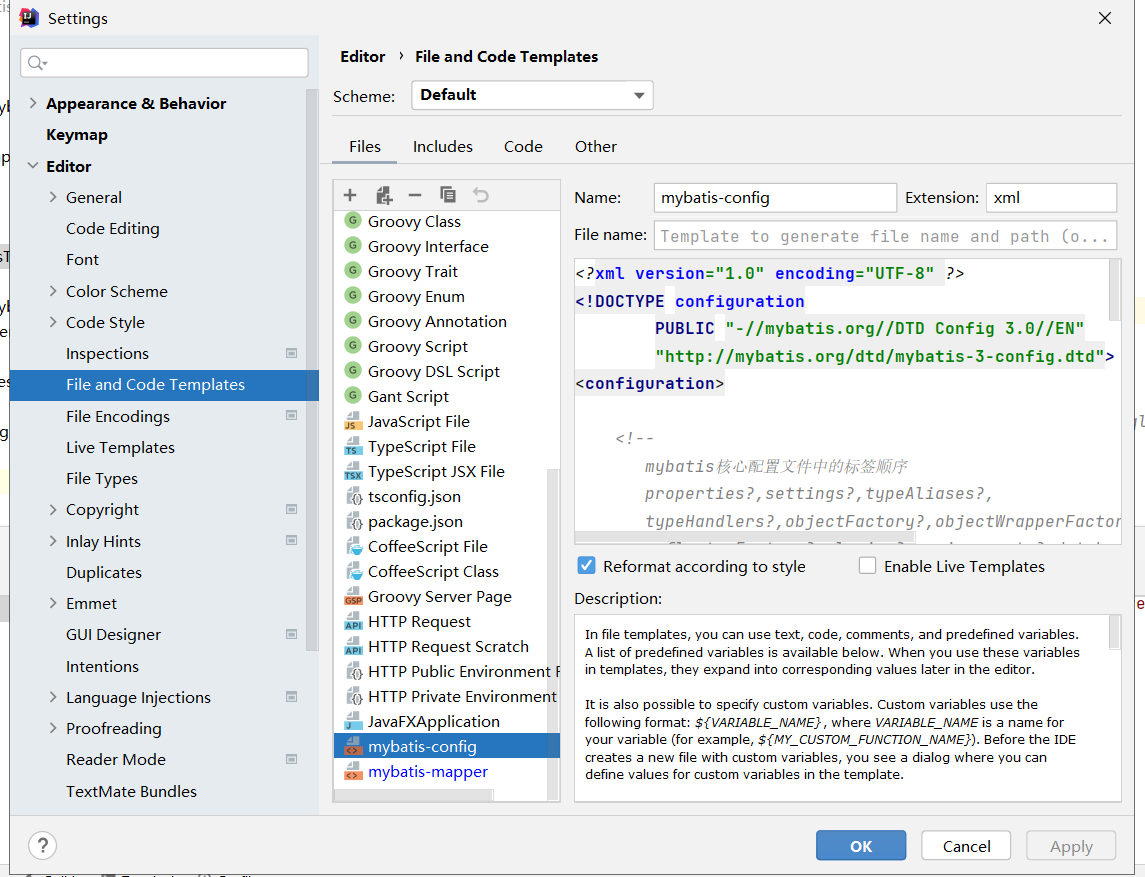
1. Mybatis获取参数的两种方式

由于每次从官方文档中获取配置文件模板都很麻烦，所以直接配置idea的模板：

配置两个模板：

mybatis-config.xml

mybatis-mapper.xml



配置完成后下次直接使用

将我们写测试类的获取SqlSession方法封装在一个SqlSessionUtils类中：

package com.libaba.mybatis.utils;

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 java.io.IOException;

import java.io.InputStream;

public class SqlSessionUtils {

public static SqlSession getSqlSession(){

SqlSession sqlSession=null;

try {

InputStream is= Resources.getResourceAsStream("mybatis-config.xml");

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

sqlSession= sqlSessionFactory.openSession(true);

} catch (IOException e) {

e.printStackTrace();

}

return sqlSession;

}

}

下次直接使用方法获取就行。。

${} ,字符串拼接，可能出现注入，处理单引号问题

#{}，占位符赋值

Mybatis获取参数值的各种情况：

1. mapper接口方法的参数为单个的字面量类型

在映射接口中写方法：

//根据用户名查询用户信息

User getUserByUserbane(String username);

然后在映射文件中写方法要执行的sql语句，这里#{username},username的值是关键，用#{aaa},也可

<select id="getUserByUserbane" resultType="User">

select \* from t\_user where username=#{username}

</select>

使用${}形式，要注意单引号问题，

<select id="getUserByUserbane" resultType="User">

<!--select \* from t\_user where username=#{aaa}-->

select \* from t\_user where username='${username}'

</select>

1. mapper接口方法的参数为多个时

Mybatis将这些参数放在一个map集合中，以arg0，arg1，为键，以param1，param2为值。

<select id="checkLogin" resultType="user">

select \* from t\_user where username=#{arg0} and password= #{arg1}

<!-- select \* from t\_user where username=#{param1} and password= #{param2}-->

</select>

此时通过${} ,#{},使用键的方式访问即可，但是需要注意${}单引号的问题

1. mapper接口方法的参数有多个时，可以手动将这些参数放在一个map集合中

然后通过${} ,#{},使用键的方式访问即可，但是需要注意${}单引号的问题

<select id="checkLoginByMap" resultType="user">

select \* from t\_user where username=#{username} and password=#{password}

</select>

1. mapper接口方法的参数是实体类类型的参数时

然后通过${} ,#{},使用对象属性的方式访问属性值即可，但是需要注意${}单引号的问题

<insert id="insertUser" >

insert into t\_user values(null,#{username},#{password},#{age},#{sex},#{email})

</insert>

1. 命名参数，使用@Param参数注解命名参数

User checkLoginByParam(@Param("username") String username, @Param("password") String password);

<select id="checkLoginByParam" resultType="user">

select \* from t\_user where username=#{username} and password=#{password}

</select>

此时，mybatis会将这些参数放在一个map集合中，

一注解的值为键，以参数为值

二以param1，param2为键，以参数位置

通过${} ,#{},使用对象属性的方式访问属性值即可，但是需要注意${}单引号的问题

1. Mybatis的各种查询功能
2. 查询一个实体类对象
3. 查询一个List集合
4. 查询单个数据
5. 查询一条数据为map集合