MyBatis

# 第一章 MyBatis框架概述

## 1-1 框架在开发中的作用以及常见框架概述

### 1-1-1 前端框架

a. 前端开发用的框架

Angular.js

React.js

Vue.js

b. 前端UI框架

Extjs

jquery ui

easy ui

bootstrap

lay ui

### 1-1-2 后端框架

**表现层框架（Controller）：servlet**

struts , xwork , struts2 , springMVC

**持久层框架（Dao）：JDBC**

Hibernate ：hql

ibatis ：xml

Mybatis

**整合框架**

EJB

Spring

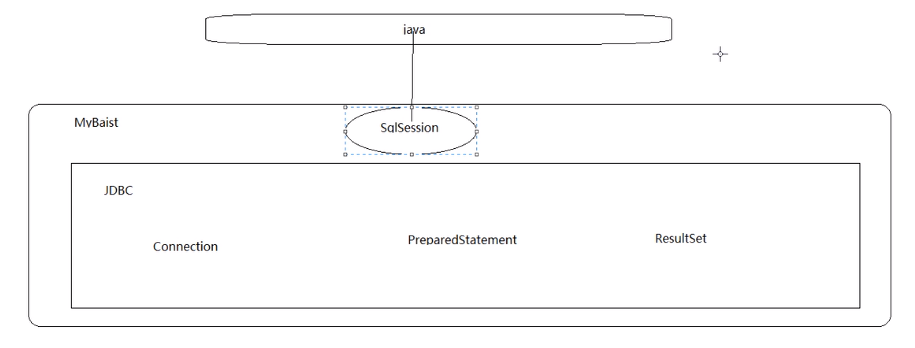
SSH：struts/strust2 spring hibernate

SSM：springmvc spring mybatis

## 1-2 MyBatis框架概述

MyBatis本是apache的一个开源项目iBatis，2010年这个项目由apache software foundation 迁移到了google code, 并且改名为MyBatis。

MyBatis是一个优秀的持久层框架。它对jdbc几乎所有的数据库操作进行了封装(包括加载驱动、创建connection、创建statement、手动设置参数、结果集检索等繁琐操作)，使开发者只需要关注SQL本身。



# 第二章 MyBatis框架搭建

## 2-1 MyBatis初识搭建开始--资源目录结构



## 2-2 MyBatis的应用

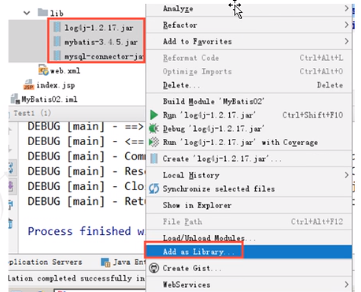
准备工作，创建学生表，创建基础数据

（1）创建项目，搭建包结构。

（2）导入mybatis相关jar包和mysql驱动包

导入log4j相关jar包

注意：导入jar包后需要在idea中构建一下才可以使用



（3）在src根下创建mybatis主配置文件mybatis-cdafig.xml ，搭建配置文件结构。

mybatis文档第三页：dbcp 、 c3p0 、 druid alibaba

（4）创建实体层domain

（5）创建mapper包结构，创建sql映射文件XxxMapper.xml。

模板在文档第四页

（6）在src根下引入log4j属性文件。

（7）搭建测试程序，测试根据ID查单条

模板在文档第二页

关键语句。

session.selectOne("test1.getByld", "A0001")。

注意：

a. 找到主配置文件的路径直接定位src下的文件即可。

b. mapper文件需要注册在主配置文件中才会生效

### 2-2-1查询单条记录

**Jdbc的操作：**

Result rs = 执行sql语句;

Student s = new Student();

if(rs.nest()){

s.setId(rs.getString(1));

s.setName(rs.getString(2));

s.setAge(rs.getInt(3));

}

**Mybatis操作：**

Student s = 执行sql语句;

### 2-2-2查询所有记录

**Jdbc操作：**

List<Student> sList = new ArrayList;

Result rs = 执行sql语句;

While（rs.next()）{

Student s = new Student();

s.setId(rs.getString(1));

s.setName(rs.getString(2));

s.setAge(rs.getInt(3));

sList.add(s);

}

**Mybatis操作:**

List<Student> sList = 执行sql语句;

## 2-3 MyBatis解决JDBC存在的问题

（1）获取连接、得到statement、处理rs、 关闭资源非常繁琐。

解决：使用SqlSession搞定一切。

（2）将sql语句写死到java代码中，如果修改sql语句，须要修改java代码，须要重新编译。程序可维护性不高。

解决：将Sql语句配置在Mapper.xml文件中与java代码分离。

（3）向PreparedStatement 对占位符的位置设置参数时，非常繁琐。

解决：Mybatis自动将java 对象映射至sql 语句，通过statement中的parameterType定义输入参数的类型。

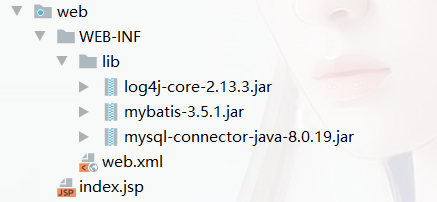
（4）解析结果集时需要把字段的值设置到相应的实体类属性名中。

解决：Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

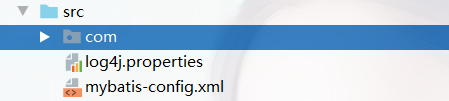
# 第三章 MyBatis基本应用

## 3-1 准备工作

**1、导入需要的jar库：**log4j-core 2.13.3.jar、mybatis-3.5.1.jar、mysql-connector-java-8.0.19.jar



**2、添加mybatis-config.xml、log4j.properties文件并设置属性**



**a、log4j.properties文件属性**

*# Global logging configuration*

1og4j.rootLogger=DEBUG, stdout

*# MyBatis logging configuration...*

*##1og4j.1ogger.org.mybatis.example.B1ogMapper=TRACE*

*# Console output. . .*

*#1og4j.appender.stdout=org.apache.1og4j.ConsoleAppender*

*#log4j.appender.stdout.1ayout=org.apache.1og4j.PatternLayout*

*#1og4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n*

*### 配置输出到控制台 ###*

log4j.appender.console = org.apache.log4j.ConsoleAppender

log4j.appender.console.Target = System.out

log4j.appender.console.layout = org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern = %d{ABSOLUTE} %5p %c{ 1 }:%L - %m%n

**b、mybatis-config.xml配置**

*<?*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>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://localhost:3306/test?serverTimezone=GMT%2B8"/>

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

<property name="password" value="123456"/>

</dataSource>

</environment>

</environments>

<mappers>

*<!--<mapper resource="org/mybatis/examp1e/BlogMapper.xm1"/>-->*

<mapper resource="com/bjpowernode/mapper/StudentMapper.xml"/>

</mappers>

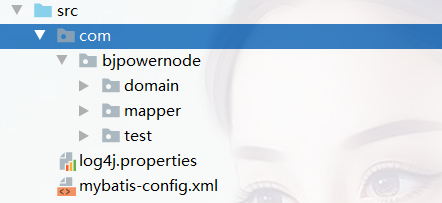
</configuration>

**3、创建需要的包**

Domain包：实体类

Mapper包：实体映射xml

Test包：测试类



## 3-2 完成根据ID查单条操作

**1、创建Student实体类**

public class Student {

private String id;

private String name;

private Integer age;

public String getId() {return id; }

public void setId(String id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) {this.name = name;}

public Integer getAge() { return age; }

public void setAge(Integer age) { this.age = age; }

@Override

public String toString() {

return "Student{" +"id='" + id + '\'' +", name='" + name + '\'' +

", age=" + age +'}';

}

}

**2、书写配置文件**

*<?*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:命名空间*

*不同的mapper映射文件使用namespace来做区分*

*不同的mapper映射文件所使用的namespace的命名不允许出现重复*

*使用命名空间 .sqlId的形式来找到我们想要执行的sql语句*

*test1.getById1*

*-->*

<mapper namespace="test1">

*<!--*

*sql语句必须要写在相应的的标签中*

*<insert>在标签中写insert开头的语句，处理添加操作*

*<update>在标签中写update开头的语句，处理修改操作*

*<delete>在标签中写delete开头的语句，处理删除操作*

*<select>在标签中写select开头的语句，处理查询操作*

*parameterType:为sql语句传递的参数类型*

*-->*

<select id="getById" parameterType="java.lang.String" resultType="com.bjpowernode.domain.Student">

select *\** from tbl\_student where id =#{id}

</select>

</mapper>

**3、书写测试类**

public class Test1 {

public static void main(String[] args) {

String resource = "mybatis-config.xml";

*//输入流*

InputStream inputStream = null;

try {

*//通过加载Mybatis的主配置文件mybatis-config.xml，创建输入对象*

inputStream = Resources.*getResourceAsStream*(resource);

}catch (IOException e){

e.printStackTrace();

}

*/\**

*SqlSessionFactoryBuilder：SqlSessionFactory的创建者*

*通过该建造者对象调用建造方法，为我们创建一个SqlSessionFactory对象*

*SqlSessionFactory对象唯一的作用技术为我们创建SqlSession对象*

*\*/*

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

*//我们未来所有的操作，使用的都是SqlSession对象*

*//例如增删改查，处理事务等等，都是统一使用session来完成*

SqlSession session = sqlSessionFactory.openSession();

*/\**

*需求：根据id查单条*

*如果取得的是单条记录，我们请用select方法*

*参数1：根据命名空间 .sqlId的形式找到我们需要使用的sql语句*

*参数2：我们要为sql语句中传递的参数*

*\*/*

Student student = session.selectOne("test1.getById","A0001");

System.*out*.println(student);

session.close();

}

}

## 3-3 完成查询信息列表操作

**1、创建Student实体类（见上面）**

**2、书写配置文件**

*<?*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:命名空间*

*不同的mapper映射文件使用namespace来做区分*

*不同的mapper映射文件所使用的namespace的命名不允许出现重复*

*使用命名空间 .sqlId的形式来找到我们想要执行的sql语句*

*test1.getById1*

*-->*

<mapper namespace="test1">

*<!--*

*如果返回的是多条记录，那么resultType返回值类型，应该写为集合的泛型*

*-->*

<select id="getAll" resultType="com.bjpowernode.domain.Student">

select *\** from tbl\_student

</select>

</mapper>

**3、书写测试类**

public class Test1 {

public static void main(String[] args) {

String resource = "mybatis-config.xml";

*//输入流*

InputStream inputStream = null;

try {

*//通过加载Mybatis的主配置文件mybatis-config.xml，创建输入对象*

inputStream = Resources.*getResourceAsStream*(resource);

}catch (IOException e){

e.printStackTrace();

}

*/\**

*SqlSessionFactoryBuilder：SqlSessionFactory的创建者*

*通过该建造者对象调用建造方法，为我们创建一个SqlSessionFactory对象*

*SqlSessionFactory对象唯一的作用技术为我们创建SqlSession对象*

*\*/*

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

*//我们未来所有的操作，使用的都是SqlSession对象*

*//例如增删改查，处理事务等等，都是统一使用session来完成*

SqlSession session = sqlSessionFactory.openSession();

*//查询学生表中的所有数据信息*

List<Student> studentlist = session.selectList("test1.getAll");

for (Student student:studentlist){

System.*out*.println(studentlist);

}

session.close();

}

}

## 3-4 完成添加操作

**1、创建Student实体类（见上面）**

**2、书写配置文件**

*<?*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:命名空间*

*不同的mapper映射文件使用namespace来做区分*

*不同的mapper映射文件所使用的namespace的命名不允许出现重复*

*使用命名空间 .sqlId的形式来找到我们想要执行的sql语句*

*test1.getById1*

*-->*

<mapper namespace="test1">

*<!--*

*注意：*

*在未来实际项目开发中*

*所有的标签都必须要写id属性*

*<select>标签中parameterType可以省略不写*

*resultType属性必须得写*

*对于<insert><update><delete>这3个标签*

*通常只写id属性，其他属性一概不写*

*-->*

<insert id="save" parameterType="com.bjpowernode.domain.Student">

insert into tbl\_student(id,name,age) values(#{id},#{name},#{age})

</insert>

</mapper>

**3、书写测试类**

public class Test1 {

public static void main(String[] args) {

String resource = "mybatis-config.xml";

*//输入流*

InputStream inputStream = null;

try {

*//通过加载Mybatis的主配置文件mybatis-config.xml，创建输入对象*

inputStream = Resources.*getResourceAsStream*(resource);

}catch (IOException e){

e.printStackTrace();

}

*/\**

*SqlSessionFactoryBuilder：SqlSessionFactory的创建者*

*通过该建造者对象调用建造方法，为我们创建一个SqlSessionFactory对象*

*SqlSessionFactory对象唯一的作用技术为我们创建SqlSession对象*

*\*/*

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

*//我们未来所有的操作，使用的都是SqlSession对象*

*//例如增删改查，处理事务等等，都是统一使用session来完成*

SqlSession session = sqlSessionFactory.openSession();

*/\**

*MyBatis默认情况下是手动提交事务*

*添加、修改、删除都需要提交事务*

*\*/*

*//添加操作*

Student student = new Student();

student.setId("A0006");

student.setName("cxk");

student.setAge(29);

session.insert("test1.save",student);

session.commit();*//手动提交事务*

session.close();

}

}

## 3-5 完成修改操作

**1、创建Student实体类（见上面）**

**2、书写配置文件**

*<?*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:命名空间*

*不同的mapper映射文件使用namespace来做区分*

*不同的mapper映射文件所使用的namespace的命名不允许出现重复*

*使用命名空间 .sqlId的形式来找到我们想要执行的sql语句*

*test1.getById1*

*-->*

<mapper namespace="test1">

<update id="update">

update tbl\_student set name=#{name},age=#{age} where id=#{id}

</update>

</mapper>

**3、书写测试类**

public class Test1 {

public static void main(String[] args) {

String resource = "mybatis-config.xml";

*//输入流*

InputStream inputStream = null;

try {

*//通过加载Mybatis的主配置文件mybatis-config.xml，创建输入对象*

inputStream = Resources.*getResourceAsStream*(resource);

}catch (IOException e){

e.printStackTrace();

}

*/\**

*SqlSessionFactoryBuilder：SqlSessionFactory的创建者*

*通过该建造者对象调用建造方法，为我们创建一个SqlSessionFactory对象*

*SqlSessionFactory对象唯一的作用技术为我们创建SqlSession对象*

*\*/*

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

*//我们未来所有的操作，使用的都是SqlSession对象*

*//例如增删改查，处理事务等等，都是统一使用session来完成*

SqlSession session = sqlSessionFactory.openSession();

*//修改操作*

Student student = new Student();

student.setId("A0006");

student.setName("cxk1");

student.setAge(30);

session.update("test1.update",student);

session.commit();

session.close();

}

}

## 3-6 完成删除操作

**1、创建Student实体类（见上面）**

**2、书写配置文件**

*<?*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:命名空间*

*不同的mapper映射文件使用namespace来做区分*

*不同的mapper映射文件所使用的namespace的命名不允许出现重复*

*使用命名空间 .sqlId的形式来找到我们想要执行的sql语句*

*test1.getById1*

*-->*

<mapper namespace="test1">

<delete id="delete">

delete from tbl\_student where id=#{id}

</delete>

</mapper>

**3、书写测试类**

public class Test1 {

public static void main(String[] args) {

String resource = "mybatis-config.xml";

*//输入流*

InputStream inputStream = null;

try {

*//通过加载Mybatis的主配置文件mybatis-config.xml，创建输入对象*

inputStream = Resources.*getResourceAsStream*(resource);

}catch (IOException e){

e.printStackTrace();

}

*/\**

*SqlSessionFactoryBuilder：SqlSessionFactory的创建者*

*通过该建造者对象调用建造方法，为我们创建一个SqlSessionFactory对象*

*SqlSessionFactory对象唯一的作用技术为我们创建SqlSession对象*

*\*/*

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

*//我们未来所有的操作，使用的都是SqlSession对象*

*//例如增删改查，处理事务等等，都是统一使用session来完成*

SqlSession session = sqlSessionFactory.openSession();

*//删除操作*

session.delete("test1.delete","A0006");

session.commit();

session.close();

}

}

# 第四章 MyBatis结合dao层的开发

## 4-1 结合原始dao的开发

**1、导入需要的jar库**

**2、添加mybatis-config.xml、log4j.properties文件**

**3、创建需要的包**

Dao包

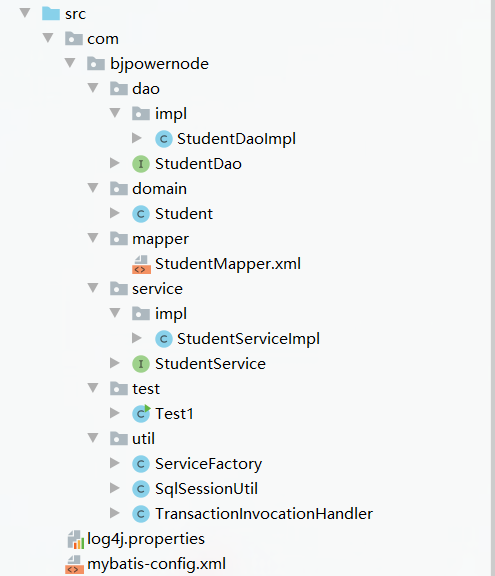
Domain包

Mapper包

Service包

Util包

Test包



**4、在工具包：util包中创建工具**

（1）创建SqlSessionUtil类

public class SqlSessionUtil {

private SqlSessionUtil() {}

private static SqlSessionFactory *sqlSessionFactory*;

static {

String resource = "mybatis-config.xml";

InputStream inputStream = null;*//输入*

try {

*//通过加载Mybatis的主配置文件mybatis-config.xml，创建输入对象*

inputStream = Resources.*getResourceAsStream*(resource);

} catch (IOException e) {

e.printStackTrace();

}

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

}

private static ThreadLocal<SqlSession> *threadLocal* = new ThreadLocal<SqlSession>();

*//取得SqlSession对象*

public static SqlSession getSession() {

SqlSession session = *threadLocal*.get();

if (session == null) {

session = *sqlSessionFactory*.openSession();

*threadLocal*.set(session);

}

return session;

}

*//关闭SqlSession对象*

public static void myClose(SqlSession session) {

if (session != null) {

session.close();

*//必须加，非常容易忘记：线程并没有销毁，而是回到了线程池中，需要手动删除*

*threadLocal*.remove();

}

}

}

（2）创建TransactionInvocationHandler代理类

*//动态代理模式:背*

public class TransactionInvocationHandler implements InvocationHandler {

*//terget:张三*

private Object target;

public TransactionInvocationHandler(Object target) {

this.target = target;

}

*//代理类的业务方法*

*//invoke ：李四的表白方法*

@Override

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

SqlSession session = null;

Object object = null;

try {

*/\*SqlSessionUtil util = new SqlSessionUtil();*

*util.getSession();\*///垃圾代码*

session = SqlSessionUtil.*getSession*();*//使用类直接调方法*

*//处理业务与逻辑*

*//method.invoke ：张三的表白方法*

object = method.invoke(target, args);

*//处理业务逻辑后，提交事务*

session.commit();

} catch (Exception e) {

session.rollback();

} finally {

SqlSessionUtil.*myClose*(session);

}

return object;

}

*//取得ls对象*

public Object getProxy() {

return Proxy.*newProxyInstance*(target.getClass().getClassLoader(), target.getClass().getInterfaces(), this);

}

}

（3）创建ServiceFactory类

public class ServiceFactory {

*//传递zs对象，得到ls对象的过程*

public static Object getService(Object service){

return

new TransactionInvocationHandler(service).getProxy();

}

}

**5、写dao层**

（1）写dao层接口

public interface StudentDao {

public Student getById(String id);

public void save(Student student);

}

（2）写dao层接口实现类：impl --> StudentDaoImpl

public class StudentDaoImpl implements StudentDao {

@Override

public Student getById(String id) {

SqlSession session = SqlSessionUtil.*getSession*();

Student student = session.selectOne("test1.getById", id);

return student;

}

@Override

public void save(Student student) {

SqlSession session = SqlSessionUtil.*getSession*();

session.insert("test1.save", student);

*//session.commit();//在业务层提交事务：Service*

}

}

**6、写Mapper映射文件**

<mapper namespace="test1">

<select id="getById" parameterType="java.lang.String" resultType="com.bjpowernode.domain.Student">

select \* from tbl\_student where id = #{id}

</select>

<insert id="save">

insert into tbl\_student(id,name,age) values(#{id},#{name},#{age})

</insert>

</mapper>

**7、业务层**

（1）写service接口

public interface StudentService {

public Student getById(String id);

public void save(Student student);

}

（2）写service接口实现类：impl —> StudentServiceImpl

public class StudentServiceImpl implements StudentService {

private StudentDao studentDao = new StudentDaoImpl();*//多态*

@Override

public Student getById(String id) {

Student student = studentDao.getById(id);

return student;

}

@Override

public void save(Student student) {

studentDao.save(student);

}

}

**8、写test测试层**

public class Test1 {

public static void main(String[] args) {

*// StudentService studentService = new StudentServiceImpl();*

StudentService studentService = (StudentService) ServiceFactory.*getService*(new StudentServiceImpl());

*//测试添加操作*

*/\* Student student = new Student();*

*student.setId("A0006");*

*student.setName("cxk");*

*student.setAge(26);*

*studentService.save(student);\*/*

*//测试根据id查询单条操作*

Student student = studentService.getById("A0001");

System.*out*.println(student);

}

}

## 4-2 MyBatis对dao层动态代理的支持

我们以前用过动态代理，是在业务层使用的，在业务层使用动态代理是为了实现事务管理。业务层的动态代理是我们自己手写的。

业务层只所以要使用动态代理，是因为业务层本身就是用来处理业务逻辑的，事务相关的代码不方便放在业务层处理，所以我们想到使用代理类帮业务层去处理。

现在我们要在dao层也要加入动态代理。dao 层之所以创建代理类，是因为写dao层实现类本身就是一种不方便。在结合了MyBatis的动态代理机制后，以后的实际项目开发， dao层的impl就不写了。

MyBatis的动态代理不用我们自己手写，在MyBaits中已经集成好的一种机制，我们直接拿来使用就可以了。

开发规则：

（1）在Mapper.xml中将namespace设置为UserDao.java接口的全限定名

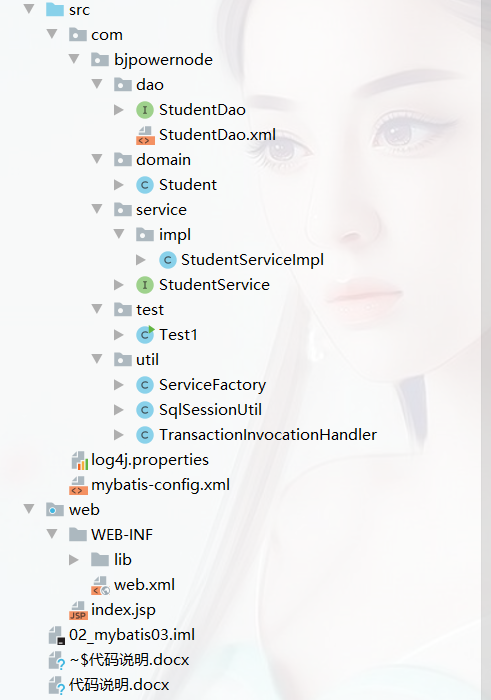
（2）将Mapper.xml中statement的id和UserDao.java接口的方法名保持一致。

步骤如下：

1、建项目

2、导入jar包

3、创建需要的包



4、在dao层删除impl，添加StudentMapper.xml，删除mapper包

（1）Studentdao接口中**方法的名字**要与Mapper映射文件中的**id的名字**必须相同；

（2）dao中有几个抽象方法，在mapper映射文件中就对应几套标签。一套标签对应一个方法。

（3）对于查询来讲，传递的参数类型和返回值类型必须跟接口中的参数类型和返回值类型保持一致。

5、业务层

（1）StudentServiceImpl

public class StudentServiceImpl implements StudentService {

*//StudentDao层的反射类型对象，语法约定*

private StudentDao studentDao = SqlSessionUtil.*getSession*().getMapper(StudentDao.class);

@Override

public Student getById(String id) {

Student student = studentDao.getById(id);

return student;

}

@Override

public void save(Student student) {

studentDao.save(student);

}

}

6、test测试

（1）测试根据id查询单条操作

public class Test1 {

public static void main(String[] args) {

*// StudentService studentService = new StudentServiceImpl();*

StudentService studentService = (StudentService) ServiceFactory.*getService*(new StudentServiceImpl());

*//测试添加操作*

*/\* Student student = new Student();*

*student.setId("A0006");*

*student.setName("cxk");*

*student.setAge(26);*

*studentService.save(student);\*/*

*//测试根据id查询单条操作*

Student student = studentService.getById("A0002");

System.*out*.println(student);

}

}

（2）查询所有记录

public class Test1 {

public static void main(String[] args) {

*// StudentService studentService = new StudentServiceImpl();*

StudentService studentService = (StudentService) ServiceFactory.*getService*(new StudentServiceImpl());

*//测试添加操作*

*/\* Student student = new Student();*

*student.setId("A0006");*

*student.setName("cxk");*

*student.setAge(26);*

*studentService.save(student);\*/*

*//测试根据id查询单条操作*

*/\*Student student = studentService.getById("A0002");*

*System.out.println(student);\*/*

*//**查询所有记录*

List<Student> studentList = studentService.getAll();*//控制层不干活，业务层拿数据*

for (Student student : studentList){

System.*out*.println(student);

}

}

}

7、在业务层创建getAll()

（1）逆向生成——在StudentService接口中生成getAll();（双击StudentService，连敲击两下shift，找到实现类）

public interface StudentService {

public Student getById(String id);

public void save(Student student);

List<Student> getAll();

}

（2）在StudentServiceImpl实现类重写方法getAll()

public class StudentServiceImpl implements StudentService {

*//StudentDao层的反射类型对象，语法约定*

private StudentDao studentDao = SqlSessionUtil.*getSession*().getMapper(StudentDao.class);

@Override

public Student getById(String id) {

Student student = studentDao.getById(id);

return student;

}

@Override

public void save(Student student) {

studentDao.save(student);

}

@Override

public List<Student> getAll() {

List<Student> studentList = studentDao.getAll();

return studentList;

}

}

8、开发dao层

（1）在StudentDao接口中生成getAll();（双击StudentDao，连敲击两下shift，找到xml文件）

public interface StudentDao {

public Student getById(String id);

public void save(Student student);

List<Student> getAll();

}

（2）在StudentMapper.xml文件中写sql语句

<mapper namespace="com.bjpowernode.dao.StudentDao">

<select id="getAll" resultType="com.bjpowernode.domain.Student">

select *\** from tbl\_student

</select>

<select id="getById" parameterType="java.lang.String" resultType="com.bjpowernode.domain.Student">

select *\** from tbl\_student where id = #{id}

</select>

<insert id="save">

insert into tbl\_student(id,name,age) values(#{id},#{name},#{age})

</insert>

</mapper>

# 第五章 MyBatis全局配置文件

## 5-1 properties

加载数据库连接属性文件。

（1）创建db.properties数据库属性配置文件

jdbc.driver = com.mysql.cj.jdbc.Driver

jdbc.url = jdbc:mysql://localhost:3306/test?serverTimezone=GMT%2B8

jdbc.username = root

jdbc.password = 123456

（2）更改mybatis信息

*<?*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>

<properties resource="db.properties"/>

<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>

<mappers>

*<!--<mapper resource="org/mybatis/examp1e/BlogMapper.xm1"/>-->*

<mapper resource="com/bjpowernode/dao/StudentDao.xml"/>

</mappers>

</configuration>

## 5-2 settings 全局配置参数

开启延迟载等等.

详细参考： mybatis-settings.docx.

注意：设置全局参数会影响mybatis框架运行，要谨慎设置！

*<!--*

*设置数据库交互的环境*

*例如：可以在此处配置二级缓存，配置查询延迟加载策略等等。。。*

*配置的目的是为了更加有效的查询表中的记录*

*在实际项目开发中，settings的设置基本没用*

*因为settings对于查询的优化，得到的效果不明显*

*对于海量级别的数据，使用settings配置优化，起不到任何的效果*

*对于数据量较少的项目，对于查询的效率要求的比较低，也没有必要使用settings配置*

*如果遇到了海量级别的数据，我们如何去提高查询的效率呢?*

*基础操作*

*对于常用的查询条件的字段，设置索引*

*高级操作*

*使用nosql数据库，redis*

*专业操作*

*搜索引擎选择：solr和elasticsearch*

*针对于电商行业*

*select \* from tbl\_product where name like ‘%手机%’*

*-->*

*<!-- <settings>*

*<setting name="" value=""/>*

*</settings>-->*

## 5-3 typeAliases

设置类型别名。

框架默认支持的别名。

参见: Mybatis默认支持的别名.docx.

**（1）在实际项目开发中，对于别名的设置，主要是针对于pojo的设置**

*<!--*

*为mapper映射文件中的domain起别名*

*-->*

<typeAliases>

*<!--*

*方式1：*

*为指定的分别起名，别名的命名由我们自己来决定*

*type：要为哪个domain起别名 填写包.类名称*

*alias：别名的名字*

*-->*

<typeAlias type="com.bjpowernode.domain.Student" alias="stu"/>

</typeAliases>

**（2）批量定义别名的方式**

<typeAliases>

*<!--*

*方式2：*

*使用package标签批量起别名*

*别名是Mybatis默认的为我们取好的，命名不是由我们自己决定，别名为类名（类名的字母不区分大小写）*

*虽然字母不区分大小写，但是我们还是要按照约定俗成的规则填写*

*类名，类名首字母可以小写*

*name：指定一个包结构，表示在该包下，所有的domain自动起好了别名*

*-->*

*<!--*

*总结：*

*(1)未来实际项目开发中，如果公司需要使用起别名的机制，我们要使用批量起别名的方式（方式2）*

*(2)在市场上也有很多企业摒弃使用MyBatis起别名的机制*

*公司会认为将domian写成全路径，可以有效的提供代码的可读性*

*-->*

<package name="com.bjpowernode.domain"/>

</typeAliases>

**（3）使用别名（首字母不区分大小写）**

<select id="getById" parameterType="String" resultType="student">

select *\** from tbl\_student where id = #{id}

</select>

## 5-4 mapper

映射器

用来在核心配置文件中注册和加载映射文件。

**（1）resource形式**

*<!--*

*方式1：*

*使用resource属性，指定mapper映射文件*

*-->*

<mapper resource="com/bjpowernode/dao/StudentDao.xml"/>

**（2）class形式**

*<!--*

*方式2：*

*使用class属性，找到dao层接口的全路径*

*-->*

<mapper class="com.bjpowernode.dao.StudentDao"/>

**（3）使用package批量注册形式**

*<!--*

*方式3：*

*批量注册*

*name属性：指向dao层的包，表示在该dao包下，所有的mapper映射文件自动注册*

*-->*

*<!--*

*总结：*

*未来实际项目开发中，我们一定批量注册mapper映射文件*

*-->*

<package name="com.bjpowernode.dao"/>

# 第六章 MyBatis映射文件

## 6-1 parameterType

设置参数类型

parameterType用于设置输入参数的java类型。parameterType 的值为参数类型的java类型或者别名,Sql 语句获取参数的值使用#{}或者${}。使用时可省略。

（1）使用简单数据类型（8基本数据类型+String）为参数

（2）使用引用数据类型为参数。

（3）使用map为参数。

## 6-2 #{}与${}

使用在sql语句中的符号

（1）#{}：表示占位符（？），可以有效防止sql注入。使用#{}设置参数无需考虑参数的类型。

（2）${}：表示拼接符，无法防止sql注入。使用${}设置参数必须考虑参数的类型。

（3）传递简单类型参数。

a. 如果获取简单类型参数，#{}中可以使用value或其它名称。

b. 如果获取简单类型参数，${}中只能使用value。

注意引号的引入

select \* from tbl student where id='${value}'

（4）在没有特殊要求的情况下，通常使用#{}占位符

（5）有些情况必须使用${} ，

比如：需要动态拼接表名，

Select \* from ${tablename}

String tbl = “tbl\_student”;

String sql = “select \* from ” + tbl

Select \* from ? where id = ?

比如：动态拼接排序字段:

select \* from tablename order by ${username} desc

（6）重点案例：

使用${}执行like模糊查询

使用#{}执行like模糊查询

## 6-3 resultType

设置返回值类

（1）返回简单类型 8+String

（2）返回pojo domain Student

（3）返回hashmap（注意返回值类型）

（4）当查询字段名和poio属性名不一致时的解决方案

a.为字段起别名,别名为类中属性名。

select id,fullname as name,age from tbl\_ student

b.使用resultMap

*<!--*

*id:resultMap标签对的唯一标识*

*将来在使用到该resultMap标签的时候，使用id来找到这组标签*

*type:指定一个类型，与数据库表一对应，建立起表字段和类属性的名字一一匹配的关系*

*-->*

<resultMap id="stuMap" type="Student">

*<!--*

*id标签：用来配置主键的对应关系的*

*result标签：用来配置普通字段对应关系的*

*对于tbl\_student表， 表结果时一个id,两个普通字段*

*所以我们需要一个id标签，两个result标签*

*property属性:配置的是类中的属性名*

*column属性:配置的是表中的字段名*

*这样就能够建立起类属性和表字段一对应的关系了*

*Student*

*id*

*name*

*age*

*tbl\_ student*

*tbl\_ id*

*tbl\_ name*

*tbl\_ age*

*-->*

<id property="id" column="id"/>

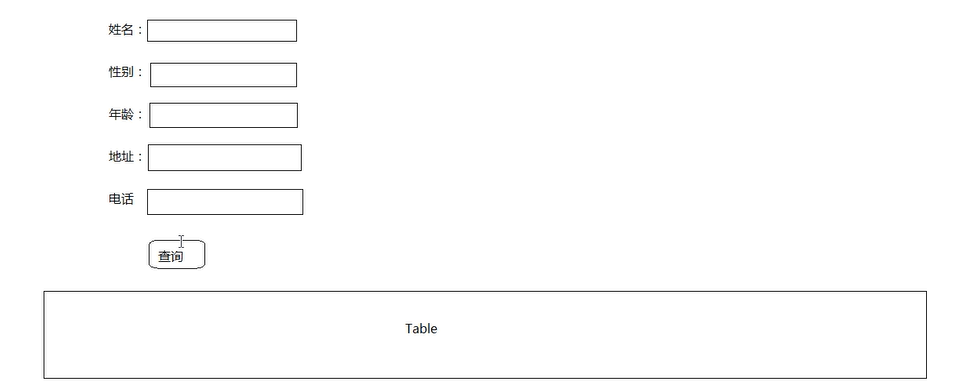
<result property="name" column="fullname"/>

<result property="age" column="age"/>

</resultMap>

# 第七章动态sql

使用动态sql，动态拼接sql语句。



情况1 ：什么都不填

String sql=“select \* from tbl";

情况2：只填写了姓名。

String sql =“select \* from tbl where name like ?”，

情况3 ：填写了姓名，性别。

String sql=“select \* from tbl where name like ? and gender=?”;

情况4：填写了姓名，性别，年龄。

String sql =“select \* from tbl where name like ? and gender=? and age=?";

在实际项目开发中，我们对于这种需求，肯定是要将sql语句写成动态的形式。动态sql 语句的核心思想是，有哪个查询条件，就动态的在where关键字后面挂载哪个查询条件。

String sql=“select \* from tbl where 1 = 1";

if(name!=null){

sql += “and name like ?”

}

if(gender!=null){

sql += “and gender = ?”

}

## 7-1 where 和 if

动态设置参数

<select id="select17" resultType="Student">

select *\** from tbl\_student

*<!--*

*where标签:*

*当where标签在使用的时候，必须要搭配where标签对中的if标签来使用*

*通过if标签的判断，如果有查询条件，则展现where关键字，如果没有查询条件则不展现where关键字*

*-->*

<where>

<if test="name !=null and name != ''">

name like '%' #{name} '%'

</if>

<if test="address !=null and address != ''">

and address like '%' #{address} '%'

</if>

</where>

</select>

## 7-2 使用foreach遍历数组和集合

<select id="select18" resultType="Student">

select *\** from tbl\_student

where id in

*<!--*

*foreach标签:用来遍历传递来的数组参数*

*collection:标识传递参数的类型*

*array:数组*

*list:集合*

*item:每一次遍历出来的元素，在使用该元素的时候，需要套用在#{}中*

*open:拼接循环的开始符号*

*close:拼接循环的结束符号*

*separator:元素与元素之间的分隔符*

*-->*

<foreach collection="array" item="id" open="(" close=")" separator=",">

#{id}

</foreach>

</select>

## 7-3 sql片段的应用

*<!--*

*使用sql标签制作sql片段*

*sql片段的作用是用来代替sql语句中的代码*

*如果你的mapper映射文件中的sql语句某些代码出现了大量的重复，我们可以使用sql片段来代替他们*

*id: sql片段的唯一标识，将来找到sql片段使用id来进行定位*

*将来的实际项目开发中，使用sql片段用来代替重复率高，且复杂的子查询*

*select \* from tbl*

*where name=(*

*select xxxxx*

*xxxxx*

*(*

*select .....*

*)*

*)*

*注意:*

*对于sql片段*

*在同一个mapper下，大量的出现重复的子查询的几率不高，所以一般情况下没有使用sql片段的必要*

*在实际项目开发中，如果你大量的使用sql片段，会大大的降低sql语句的可读性*

*在很多企业中，干脆摒弃使用sql片段的机制*

*-->*

<sql id="sql1">

select *\** from tbl\_student

</sql>

<select id="select19" resultType="Student">

<include refid="sql1"/> where id = #{id}

</select>

# 第八章 多表联查案例

关联学生表和班级表

（1）查询出学生姓名和班级名称

**a.测试Test2.java**

*//20.测试：多表联查 查询出学生姓名和班级名称*

List<Map<String,Object>> mapList = studentDao.select20();

for (Map<String,Object> map:mapList){

Set<String> set = map.keySet();

for (String key:set){

System.*out*.println("key:"+key);

System.*out*.println("value:"+map.get(key));

}

System.*out*.println("-------------");

}

**b. StudentDao接口**

List<Map<String, Object>> select20();

**c. StudentDao.xml映射文件**

<select id="select20" resultType="map">

select

s.name as sname,

c.name as cname

from tbl\_student s

join tbl\_classroom c

on s.classroomId = c.id

</select>

（2）查询出学生和班级所有信息，加VO（View Object：视图对象；Value Object：持久对象）

Pojo-->domain

**a.测试Test2.java**

*//21.测试：多表联查 查询出学生和班级所有信息，加VO*

*/\**

*在实际项目开发中，如果需要为前端展现的数据，使用一个domain类型不足以表现出来这些数据*

*这时我们可以考虑使用两种技术来实现*

*分别为:*

*使用map以及使用vo*

*例如 我们现在的需求*

*查询出学生和班级所有信息*

*得到的结果 使用学生的domain或者班级的domian都不能够封装这些结果*

*所以我们可以使用map去保存这些信息*

*同时我们也可以使用vo类来保存这些信息*

*vo指的是创建出来一个类，这个类中的属性是完全由我们自己去定义，属性会保存所有需要展现的信息*

*例如 我们现在的这个例子，我们可以使用vo来封装所有与学生和班级相关的信息*

*vo*

*student*

*classroom*

*\*/*

List<StudentAndClassroomVo> voList = studentDao.select21();

for (StudentAndClassroomVo vo : voList){

System.*out*.println(vo);

}

**b. StudentDao接口**

List<StudentAndClassroomVo> select21();

**c. StudentDao.xml映射文件**

<select id="select21" resultType="com.bjpowernode.vo.StudentAndClassroomVo">

select

s.id sid,

s.name sname,

s.age sage,

s.address saddress,

c.id cid,

c.name cname

from tbl\_student s

join tbl\_classroom c

on s.classroomId = c.id

</select>

（3）查询出带有字母z的学生和班级所有信息

**a.测试Test2.java**

*//22.测试：多表联查 查询出带有字母z的学生和班级所有信息*

List<StudentAndClassroomVo> voList = studentDao.select22("z");

for (StudentAndClassroomVo vo : voList){

System.*out*.println(vo);

}

*/\**

*实际项目开发中，如果要为前端同时提供多组值，那么我们应该使用map还是vo呢?*

*如果前端的需求的重复率不高，那么我们选择临时使用map就可以了*

*如果前端对于该需求的重复率较高，那么我们可以创建一个vo类来使用，非常方便*

*\*/*

**b. StudentDao接口**

List<StudentAndClassroomVo> select22(String z);

**c. StudentDao.xml映射文件**

<select id="select22" resultType="com.bjpowernode.vo.StudentAndClassroomVo">

select

s.id sid,

s.name sname,

s.age sage,

s.address saddress,

c.id cid,

c.name cname

from tbl\_student s

join tbl\_classroom c

on s.classroomId = c.id

where s.name like '%' #{name} '%'

</select>