springmvc+mybatis项目构建

# maven项目构建测试

## 导入所需的jar包

<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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.ssm</groupId>

<artifactId>ssmmaven</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>ssmmaven Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<spring.version>3.2.0.RELEASE</spring.version>

<jackson.version>2.5.0</jackson.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<!-- spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

<scope>test</scope>

</dependency>

<!-- mybatis 包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.2.8</version>

</dependency>

<!--mybatis spring 插件 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.2</version>

</dependency>

<!-- mysql连接 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.34</version>

</dependency>

<!-- 数据源 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.12</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.4</version>

</dependency>

<!-- log4j -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<!-- servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>3.0-alpha-1</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- json -->

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.11</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>${jackson.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>${jackson.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.version}</version>

</dependency>

<!-- 文件上传 -->

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

</dependency>

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.2.2</version>

</dependency>

</dependencies>

<build>

<finalName>ssm</finalName>

<plugins>

<!-- 编译的时候使用JDK6和UTF8编码 -->

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.6</source>

<target>1.6</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<!-- 配置Maven插件(mvn jetty:run可以运行项目) -->

<plugin>

<groupId>org.mortbay.jetty</groupId>

<artifactId>maven-jetty-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

## 建立相关文件

### 新建对象TestUser

**package** com.ssm.po;

**public** **class** TestUser {

**private** String id;

**private** String username;

**private** Integer group;

**private** String course;

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** Integer getGroup() {

**return** group;

}

**public** **void** setGroup(Integer group) {

**this**.group = group;

}

**public** String getCourse() {

**return** course;

}

**public** **void** setCourse(String course) {

**this**.course = course;

}

}

### 新建TestUser.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">

<mapper namespace="com.ssm.po.TestUser">

<resultMap type="TestUser" id="testUserMap">

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

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

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

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

</resultMap>

<select id="selectUserById" parameterType="string" resultMap="testUserMap">

select \* from testuser where id=#{id}

</select>

</mapper>

### 新建文件夹config并创建配置文件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>

<!-- 定义一个别名 -->

<typeAliases>

<typeAlias type="com.ssm.po.TestUser" alias="TestUser"/>

</typeAliases>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

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

<property name="url" value="jdbc:mysql://localhost/test" />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<!-- 映射文件的位置 -->

<mapper resource="com/ssm/po/TestUser.xml"/>

</mappers>

</configuration>

### 测试

在src/test/java下新建包com.ssm.test并新建测试类TestPro.java

**package** com.ssm.test;

**import** java.io.InputStream;

**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.Before;

**import** org.junit.Test;

**import** com.ssm.po.TestUser;

**public** **class** TestPro {

SqlSessionFactory sf;

@Before

**public** **void** initSF()**throws** Exception{

String resource = "config/sqlMapConfig.xml";

InputStream inputStream = Resources.getResourceAsStream(resource);

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

}

@Test

**public** **void** testSelect() **throws** Exception{

SqlSession session = sf.openSession();

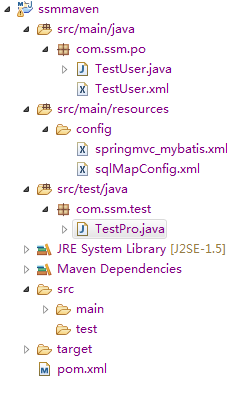
TestUser u = session.selectOne("com.ssm.po.TestUser.selectUserById", "0001");

System.**out**.println(u.getUsername());

}

}

### 包结构



# Mapper,Service层建立与测试

## Mapper层实现方法(原始开发)

/java下创建包com.ssm.mapper

### 建立TestUserMapper和TestUserMapperImpl

**package** com.ssm.mapper;

**import** com.ssm.po.TestUser;

**public** **interface** TestUserMapper {

**public** TestUser selectUserById(String id );

}

**package** com.ssm.mapper;

**import** org.mybatis.spring.support.SqlSessionDaoSupport;

**import** com.ssm.po.TestUser;

**public** **class** TestUserMapperImpl **extends** SqlSessionDaoSupport **implements** TestUserMapper{

**public** TestUser selectUserById(String id) {

TestUser testUser=**this**.getSqlSession().selectOne("com.ssm.po.TestUser.selectUserById", "0001");

**return** testUser;

}

}

### 创建spring配置文件springmvc\_dao.xml

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.2.xsd ">

<!-- 配置数据源 -->

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

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

<property name="url" value="jdbc:mysql:///test" />

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

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

</bean>

<!-- 配置mybatis的工厂 -->

<bean id="sf" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="configLocation" value="classpath:config/sqlMapConfig.xml"/>

<property name="mapperLocations">

<value>classpath:com/ssm/po/TestUser.xml</value>

</property>

</bean>

<!-- 配置一个dao -->

<bean id="testUserMapper" class="com.ssm.mapper.TestUserMapperImpl">

<property name="sqlSessionFactory" ref="sf" />

</bean>

</beans>

建立配置文件后就可以将sqlMapConfig中到相关和映射去掉,只留别名

### 测试

@Test

**public** **void** TestDaoSelect() **throws** Exception{

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_mybatis.xml");

TestUserMapper testUserMapper= (TestUserMapper) ac.getBean("testUserMapper");

TestUser testUser=testUserMapper.selectUserById("0001");

System.**out**.println(testUser.getUsername());

}

## Mapper层改进(自动注入)

删除TestUserMapperImpl并创建TestUserMapper.xml文件用来替换TestUser.xml文件

### 修改TestUserMapper.xml文件中内容

更改命名空间与接口一致 <mapper namespace="com.ssm.mapper.TestUserMapper">

### 使用MapperScannerConfigurer自动扫描

1 在springmvc\_dao.xml中增加配置

<!--

MapperScannerConfigurer：mapper的扫描器，将包下边的mapper接口自动创建代理对象，

自动创建到spring容器中，bean的id是mapper的类名（首字母小写）

-->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<!-- 配置扫描包的路径

如果要扫描多个包，中间使用半角逗号分隔

要求mapper.xml和mapper.java同名且在同一个目录

-->

<property name="basePackage" value="com.ssm.mapper"/>

<!-- 使用sqlSessionFactoryBeanName -->

<property name="sqlSessionFactoryBeanName" value="sf"/>

</bean>

</beans>

2 删除在SqlSessionFactoryBean中的配置项 mapperLocations

## 测试程序

测试仍然用上面的

## Service层实现

### 创建TestUserService接口和TestUserServiceImpl实现类

实现mapper中的方法

### dao.xml中增加注解扫描

<context:component-scan base-package="com.ssm"></context:component-scan>

### 创建测试类进行测试

@Test

**public** **void** TestServiceSelect(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestUserService testService=(TestUserService) ac.getBean("testUserServiceImpl");

TestUser user=testService.selectUserById("0001");

System.**out**.println(user.getUsername());

}

# Controller层搭建

## maven项目的部署

build部署maven方式.1使用本地tomcat路径部署(部署项目需要去掉servlet相关jar包,不然会冲突)

<build>

<finalName>springmvc</finalName>

<plugins>

<plugin>

<groupId>org.codehaus.cargo</groupId>

<artifactId>cargo-maven2-plugin</artifactId>

<version>1.2.3</version>

<configuration>

<container>

<containerId>tomcat7x</containerId>

<home>F:/Tomcat8.0</home>

</container>

<configuration>

<type>existing</type>

<home>F:/Tomcat8.0</home>

</configuration>

</configuration>

<executions>

<execution>

<id>cargo-run</id>

<phase>install</phase>

<goals>

<goal>run</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

使用这种方式部署直接install即可

方法2:使用maven进行部署

<build>

<plugins>

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<path>/</path>

<port>8080</port>

<url>http://localhost:8080/manager/html</url>

<uriEncoding>UTF-8</uriEncoding>

<server>tomcat7</server>

<contextReloadable>false</contextReloadable>

<useTestClasspath>true</useTestClasspath>

</configuration>

</plugin>

</plugins>

</build>

使用这种方法不需要输入项目名称

3 <!-- 配置Maven插件(mvn jetty:run可以运行项目) -->

<plugin>

<groupId>org.mortbay.jetty</groupId>

<artifactId>maven-jetty-plugin</artifactId>

</plugin>

使用这种方法install后可以直接add在tomcat中运

行

## 新建springmvc.xml进行springmvc配置

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.2.xsd ">

<!-- 注解扫描 -->

<context:component-scan base-package="com.ssm.controller"></context:component-scan>

<!-- 默认的注解映射的支持，自动注册DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter -->

<mvc:annotation-driven />

<!-- 配置模型视图解析器 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/page/" />

<property name="suffix" value=".jsp" />

<!-- 如果使用jstl的话，配置下面的属性 -->

<property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />

</bean>

</beans>

## web.xml中更改头文件和配置相关信息

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

<web-app version="2.5"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<display-name></display-name>

<filter>

<filter-name>CharactorEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharactorEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<display-name>Archetype Created Web Application</display-name>

<!-- Spring MVC 上下文监听 -->

<servlet>

<servlet-name>MVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

classpath:config/springmvc\*.\*

</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- Spring MVC 普通页面拦截 -->

<servlet-mapping>

<servlet-name>MVC</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<listener>

<listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>

</listener>

</web-app>

## 新建TestUserController

@Controller

**public** **class** TestUserController {

@Autowired

TestUserService testUserService;

@RequestMapping("login")

**public** String login(HttpServletRequest request){

TestUser testUser=testUserService.selectUserById("0001");

request.setAttribute("user", testUser);

**return** "login";

}

}

## WEB-INF下新建page页面测试

# 一对多多对一关系建立

## 新建TestClass.java类

**package** com.ssm.po;

**import** java.util.List;

**public** **class** TestClass {

**private** Integer id;

//班级年纪

**private** String name;

//班级数

**private** Integer num;

//hibernate中一对多

**private** List<TestUser> testUsers;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Integer getNum() {

**return** num;

}

**public** **void** setNum(Integer num) {

**this**.num = num;

}

**public** List<TestUser> getTestUsers() {

**return** testUsers;

}

**public** **void** setTestUsers(List<TestUser> testUsers) {

**this**.testUsers = testUsers;

}

}

## 新建TestClassMapper.xml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">

<mapper namespace="com.ssm.mapper.TestClassMapper">

<resultMap type="TestClass" id="testClassMap">

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

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

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

<!-- 配置一对多的属性 -->

<collection property="testUsers" ofType="TestUser" >

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

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

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

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

</collection>

</resultMap>

<select id="selectById" parameterType="int" resultType="TestClass">

select \* from testclass where classId=#{id}

</select>

<!-- 查询多个对象 -->

<select id="selectAllOrderById" parameterType="int" resultMap="testClassMap">

select p.\*,o.\* from testclass o,testuser p where p.classId=o.classId and o.classId=#{id}

</select>

</mapper>

## 修改TestUserMapper.xml文件增加多对1关系

<!-- 配置多对一的关系 -->

<association property="testClass" javaType="TestClass">

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

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

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

</association>

## 修改TestUser.java类新增对象

//hibernate中多对1,,数据库中为classId,程序中为对象

**private** TestClass testClass;

## 新建TestClassMapper接口

**package** com.ssm.mapper;

**import** com.ssm.po.TestClass;

**public** **interface** TestClassMapper {

**public** TestClass selectAllOrderById(Integer id);

**public** TestClass selectById(Integer id);

}

## 新建测试程序测试

@Test

**public** **void** selectAllOrderById(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestClassMapper testClassMapper=(TestClassMapper) ac.getBean("testClassMapper");

TestClass testClass=testClassMapper.selectAllOrderById(1);

System.**out**.println(testClass.getTestUsers().get(0).getUsername());

}

# 对mybatis注解的研究说明

## @SelectProvide和 @ResultMap("")

一般连起来用., SelectProvide作用可以声明调用某个类的某些方法:

**定义拼装sql的类**

@SelectProvide中type参数指定的Class类，必须要能够通过无参的构造函数来初始化。

@SelectProvide中method参数指定的方法，必须是public的，返回值必须为String，可以为static。

1 public class SqlProvider {  
2     public String selectUser(long userId) {  
3         return "select \* from user where userId=" + userId;  
4     }  
5 }

如何使用:在需要使用的接口方法上写上注解

 2   @SelectProvider(type = SqlProvider.class, method = "selectUser")  
3     @ResultMap("userMap")  
4     public User getUser(long userId);

## ResultMap说明

**使用resultType进行输出映射，只有查询出来的列名和pojo中的属性名一致，该列才可以映射成功。**

**如果查询出来的列名和pojo的属性名不一致，通过定义一个resultMap对列名和pojo属性名之间作一个映射关系**

# mybatis主键生成策略

## 自增主键返回对象中(只适合主键自增的表)

### TestClassMapper.xml中添加insert语句,返回最后添加的主键值

<insert id="insertClass" parameterType="TestClass">

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

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into testclass(name,num) value(#{name},#{num})

</insert>

### TestClassMapper接口中添加insert方法

//测试insert主键返回

**public** Integer insertClass(TestClass testClass);

### 增加测试方法看对象中是否返回主键值

@Test

**public** **void** insertClass(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestClassMapper testClassMapper=(TestClassMapper) ac.getBean("testClassMapper");

TestClass testClass=**new** TestClass();

testClass.setName("三");

testClass.setNum(5);

**int** a=testClassMapper.insertClass(testClass);

System.**out**.println(testClass.getId());

}

如果testClass.getId()返回值.说明将主键值返回

## uuid之间的生成策略(数据库主键长度必须设35位)

### 在sql语句执行之前生成uuid

<insert id="insertUser" parameterType="TestUser">

<selectKey keyProperty="id" order="BEFORE" resultType="java.lang.String">

select replace(uuid(),'-','')

</selectKey>

insert into testuser(id,username,grade,course) value(#{id,jdbcType=VARCHAR},#{username},#{grade},#{course})

</insert>

使用

select replace(uuid(),'-','')

生成.

### 测试程序

//测试uuid主键生成数据

@Test

**public** **void** insertUser(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestUserMapper testUserMapper=(TestUserMapper) ac.getBean("testUserMapper");

TestUser testUser = **new** TestUser();

testUser.setUsername("wwwwwww");

testUser.setCourse("aaaa");

testUser.setGrade(98);

testUserMapper.insertUser(testUser);

}

数据添加成功则uuid生成,

### #{}和${}

#{}表示一个占位符号，#{}接收输入参数，类型可以是简单类型，pojo、hashmap。

如果接收简单类型，#{}中可以写成value或其它名称。

#{}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...的方式获取对象属性值。

${}表示一个拼接符号，会引用sql注入，所以不建议使用${}。

${}接收输入参数，类型可以是简单类型，pojo、hashmap。

如果接收简单类型，${}中只能写成value。

${}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...的方式获取对象属性值。

### selectOne和selectList

selectOne表示查询出一条记录进行映射。如果使用selectOne可以实现使用selectList也可以实现（list中只有一个对象）。

selectList表示查询出一个列表（多条记录）进行映射。如果使用selectList查询多条记录，不能使用selectOne。

如果使用selectOne报错：

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

### mybatis和hibernate本质区别和应用场景

hibernate：是一个标准ORM框架（对象关系映射）。入门门槛较高的，不需要程序写sql，sql语句自动生成了。

对sql语句进行优化、修改比较困难的。

应用场景：

适用与需求变化不多的中小型项目，比如：后台管理系统，erp、orm、oa。。

mybatis：专注是sql本身，需要程序员自己编写sql语句，sql修改、优化比较方便。mybatis是一个不完全 的ORM框架，虽然程序员自己写sql，mybatis 也可以实现映射（输入映射、输出映射）。

应用场景：

适用与需求变化较多的项目，比如：互联网项目。

企业进行技术选型，以低成本 高回报作为技术选型的原则，根据项目组的技术力量进行选择。

## 传递pojo的包装对象

### 需求

完成用户信息的综合查询，需要传入查询条件很复杂（可能包括用户信息、其它信息，比如商品、订单的）

### 定义包装类型pojo

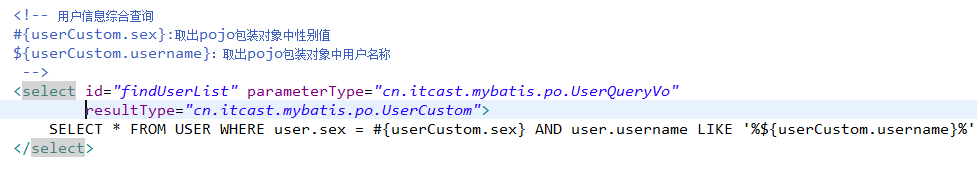
针对上边需求，建议使用自定义的包装类型的pojo。

在包装类型的pojo中将复杂的查询条件包装进去。



在UserMapper.xml中定义用户信息综合查询（查询条件复杂，通过高级查询进行复杂关联查询）。

xml配置文件中



mapper接口中方法



测试代码



# 动态sql

## 什么是动态sql

mybatis核心 对sql语句进行灵活操作，通过表达式进行判断，对sql进行灵活拼接、组装。

## 需求

自定义查询条件功能的实现

## 动态拼接sql

<select id="selectUserByCollection" parameterType="com.ssm.po.TestUser" resultMap="testUserMap">

select \* from testuser

<where>

<if test="id!=null">

and id=#{id}

</if>

<if test="username!=null">

and username=#{username}

</if>

<if test="grade!=null">

and grade=#{grade}

</if>

<if test="course!=null">

and course=#{course}

</if>

</where>

</select>

//自定义查询条件测试

@Test

**public** **void** selectUserByCollection(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestUserMapper testUserMapper=(TestUserMapper) ac.getBean("testUserMapper");

TestUser testUser = **new** TestUser();

testUser.setCourse("语文");

List<TestUser> list=testUserMapper.selectUserByCollection(testUser);

System.**out**.println(list.size());

}

## sql片段方便开发

### 需求

将上边实现的动态sql判断代码块抽取出来，组成一个sql片段。其它的statement中就可以引用sql片段。

方便程序员进行开发。

<sql id="sql\_where">

<if test="id!=null">

and id=#{id}

</if>

<if test="username!=null">

and username=#{username}

</if>

<if test="grade!=null">

and grade=#{grade}

</if>

<if test="course!=null">

and course=#{course}

</if>

</sql>

## foreach

向sql传递数组或List，mybatis使用foreach解析

### 需求

在用户查询列表和查询总数的statement中增加多个id输入查询。

sql语句如下：

两种方法：

SELECT \* FROM TESTUSER WHERE id=1 OR id=10 OR id=16

SELECT \* FROM TESTUSER WHERE id IN(1,10,16)

### 用list传递多个id

1必须要使用到代理对象

创建代理对象

**import** java.util.List;

**public** **class** TestUserVo {

TestUser testUser;

List<String> ids;

**public** TestUser getTestUser() {

**return** testUser;

}

**public** **void** setTestUser(TestUser testUser) {

**this**.testUser = testUser;

}

**public** List<String> getIds() {

**return** ids;

}

**public** **void** setIds(List<String> ids) {

**this**.ids = ids;

}

}

2 xml中写上select

<!-- 多个id 查询 -->

<select id="selectUserByOr" parameterType="com.ssm.po.TestUserVo" resultMap="testUserMap">

select \* from testuser where 1=1

<if test="ids!=null">

<foreach collection="ids" item="user\_id" open="and(" close=")" separator="or">

id=#{user\_id}

</foreach>

</if>

</select>

3Mapper接口中写上方法

//传递多个id进行查询

**public** List<TestUser> selectUserByOr(TestUserVo testUserVo);

4测试

//List数组传入实现选择查询

@Test

**public** **void** selectUserById(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestUserMapper testUserMapper=(TestUserMapper) ac.getBean("testUserMapper");

TestUserVo testUserVo=**new** TestUserVo();

List<String> ids=**new** ArrayList<String>();

ids.add("0001");

ids.add("0002");

testUserVo.setIds(ids);

List<TestUser> list = testUserMapper.selectUserByOr(testUserVo);

System.**out**.println(list.size());

}

## 一对多 多对一的动态关系

订单明细表

id 订单id 商品id

1 1 1

2 1 2

3 2 1

4 3 1

一对一

一对一

一对多

商品表

id name

1 mianbao

2 xiezi

用户表

id ame

1 zangsan

2 lisi

一对多

一对一一

订单表

id userid

1 1

2 2

3 1

一对多

## 延时加载

resultMap可以实现高级映射（使用association、collection实现一对一及一对多映射），**association、collection具备延迟加载功能。**

如果查询订单并且关联查询用户信息。如果先查询订单信息即可满足要求，当我们需要查询用户信息时再查询用户信息。把对用户信息的按需去查询就是延迟加载。

延迟加载：先从单表查询、需要时再从关联表去关联查询，大大提高 数据库性能，因为查询单表要比关联查询多张表速度要快。

### 需求

在查询班级的时候将学生的信息进行延迟加载,当需要使用的时候使用get方法,mybatis将自动调用配置的相应方法进行处理.参数为班级的id这是一对多延迟加载的表现.与上面的

一对多所配置的并不相同.

### 执行步骤

1 在TestClassMapper.xml文件中增加 找所有班级的sql

<select id="selectAllClass" resultMap="testClassMap">

select \* from testclass

</select>

2 在TestUserMapper中增加根据classId返回TestUser对象的sql

<!-- 测试延迟加载 -->

<select id="selectUserByClassId" parameterType="int" resultType="TestUser">

select \* from testuser where classId=#{value}

</select>

3 配置association 或者collection关系该关系决定我传入的值和需要执行的sql

<!-- 配置一对多的属性 -->

<!-- 延迟加载 -->

<collection property="testUsers" ofType="TestUser"

select="com.ssm.mapper.TestUserMapper.selectUserByClassId" column="classId">

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

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

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

<result column="grade" property="grade"/> -->

</collection>

4 TestClassMapper中需要写上查询TestClass数组的方法

**blic** List<TestClass> selectAllClass();

5 TestUserMapper中需要写上根据classId的查询方法

//进行延迟加载测试

**public** List<TestUser> selectUserByClassId(Integer classId);

6需要在SqlMapperConfig.开启延迟加载功能

<settings>

<!-- 打开延迟加载的开关 -->

<setting name="lazyLoadingEnabled" value="true" />

<!-- 将积极加载改为消息加载即按需加载 -->

<setting name="aggressiveLazyLoading" value="false"/>

</settings>

7jar包cglib需要

8测试

//测试延迟加载

@Test

**public** **void** selectAllClass(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

TestClassMapper testClassMapper=(TestClassMapper) ac.getBean("testClassMapper");

List<TestClass> list = testClassMapper.selectAllClass();

System.**out**.println(list.get(0));

List<TestUser> listUser = list.get(0).getTestUsers();

System.**out**.println(listUser.get(0).getUsername());

}

### 工作流程

用户查询

classId

TestUserMapper.java

public List<TestUser> selectTestUser(int classId)

collection或assocation

延迟加载被调用

TestUserMapper.xml

select \*……where id=#{classId}

TestClassMapper.java

public List<TestClass> selectTestClass()

TestClassMapper.xml

select \*…………

# 逆向工程

## 新建一个测试工程,用来生成包和文件

配置文件

lib包

## 运行程序GeneratorSqlmap

## 将运行的包拷到我们的工程下

## 新建测试文件测试

**package** com.ssm.test;

**import** java.util.List;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.ssm.mapper.TestclassMapper;

**import** com.ssm.po.Testclass;

**import** com.ssm.po.TestclassExample;

**public** **class** TestCreateTable {

/\*\*

\* 使用生成的程序进行测试

\*/

TestclassMapper classMapper;

@Before

**public** **void** beforeDo(){

ApplicationContext ac=**new** ClassPathXmlApplicationContext("config/springmvc\_dao.xml");

classMapper=(TestclassMapper) ac.getBean("testclassMapper");

}

//自定义查询条件

@Test

**public** **void** testSelectByExample() **throws** Exception{

TestclassExample testclassExample= **new** TestclassExample();

TestclassExample.Criteria criteria = testclassExample.createCriteria();

criteria.andNameEqualTo("三");

List<Testclass> list=classMapper.selectByExample(testclassExample);

System.**out**.println(list);

}

//根据主键进行查询

@Test

**public** **void** testSelectById(){

Testclass testclass = classMapper.selectByPrimaryKey(1);

System.**out**.println(testclass.getName());

}

//插入

@Test

**public** **void** testInsert(){

Testclass testclass=**new** Testclass();

testclass.setName("aaa");

testclass.setNum(53);

**int** a = classMapper.insert(testclass);

System.**out**.println(a);

}

//更新数据

@Test

**public** **void** testUpdateById(){

//对所有字段进行更新，需要先查询出来再更新

Testclass testclass = classMapper.selectByPrimaryKey(1);

testclass.setName("zhouwu");

classMapper.updateByPrimaryKey(testclass);

}

}