MyBatis

# HelloWorld

## 1.1 导入mybatis的依赖包

|  |
| --- |
| <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.1</version>  </dependency> |

## 1.2 生成dbconfig.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/test  jdbc.username=root  jdbc.password=123456 |

## 1.3 生成mybatis配置文件mybatis-config.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>  <properties resource=*"dbconfig.properties"*>  </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>  <!-- 将我们写好的sql映射文件（UserMapper.xml）一定要注册到全局配置文件（mybatis-config.xml）中 -->  <mappers>  <mapper resource=*"UserMapper.xml"* />  </mappers>  </configuration> |

## 1.4 生成UserMapper.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=*"org.softcits.dao.UserMapper"*>  <!--  namespace:名称空间;指定为接口的全类名  id：唯一标识并和接口函数名一致  resultType：返回值类型  #{username}：从传递过来的参数中取值  UserMapper 接口如下定义  public User getUserByUsername(String username);  -->  <select id=*"getUserByUsername"* resultType=*"org.softcits.model.User"*>  <!-- 数据库字段名如何model属性名不一样,可以如下起别名使其统一 -->  select id, user\_name username,password,phone from tbl\_user where user\_name=#{username};  </select>  </mapper> |

## 1.5 生成UserMapper接口

接口式编程可以规范返回类型和传入参数类型

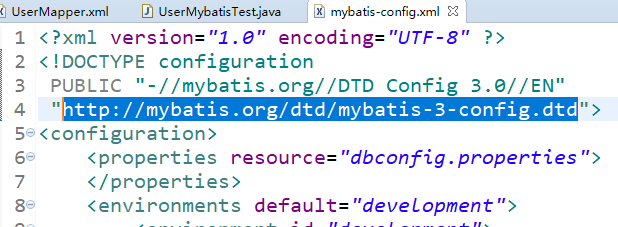
|  |
| --- |
| **package** org.softcits.dao;  **import** org.softcits.model.User;  **public** **interface** UserMapper {  **public** User getUserByUsername(String username);  } |

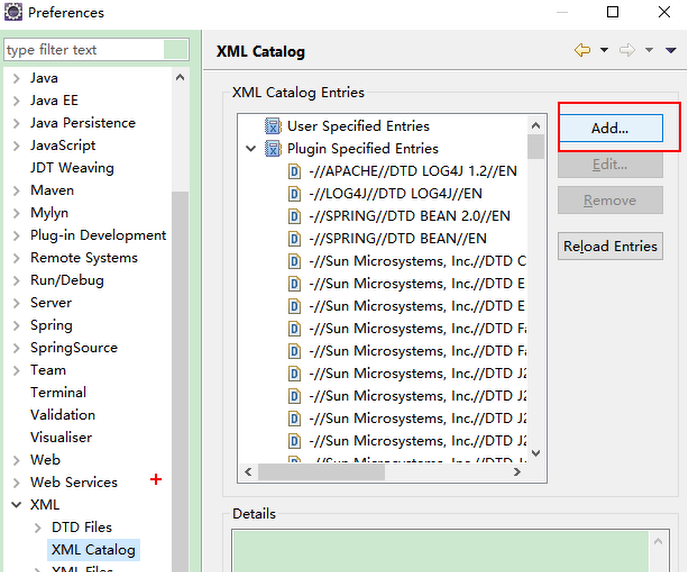
## 1.6 生成测试类

|  |
| --- |
| **package** com.mybatis.test;  **import** java.io.IOException;  **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.Test;  **import** org.softcits.dao.UserMapper;  **import** org.softcits.model.User;  /\*\*  \* 1、接口式编程  接口 ====> 实现类  \* 原生： Dao ====> DaoImpl  \* mybatis： Mapper ====> xxMapper.xml是对Mapper接口的实现  \*  \* 2、SqlSession代表和数据库的一次会话；用完必须关闭；  \* 3、SqlSession和connection一样她都是非线程安全。每次使用都应该去获取新的对象。  \* 4、mapper接口没有实现类，但是mybatis会为这个接口生成一个代理对象。  \* （将接口和xml进行绑定）  \* UserMapper user = sqlSession.getMapper(UserMapper.class);  \* 5、两个重要的配置文件：  \* mybatis的全局配置文件：包含数据库连接池信息，事务管理器信息等...系统运行环境信息  \* sql映射文件：保存了每一个sql语句的映射信息：  \* 将sql抽取出来。  \*  \*/  **public** **class** UserMybatisTest {  **public** SqlSessionFactory getSqlSessionFactory() **throws** IOException{  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.*getResourceAsStream*(resource);  **return** **new** SqlSessionFactoryBuilder().build(inputStream);  }    @Test  **public** **void** test01(){  SqlSessionFactory sqlSessionFactory = **null**;  SqlSession openSession = **null**;  **try** {  sqlSessionFactory = getSqlSessionFactory();  openSession = sqlSessionFactory.openSession();  //获取接口的实现类对象  //会为接口自动的创建一个代理对象，代理对象去执行增删改查方法  UserMapper umapper = openSession.getMapper(UserMapper.**class**);    User user = umapper.getUserByUsername("Jimmy");    System.***out***.println(user.getId() + " " + user.getUsername() + " " + user.getPassword() + " " + user.getPhone());  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally**{  openSession.close();  }  }  } |

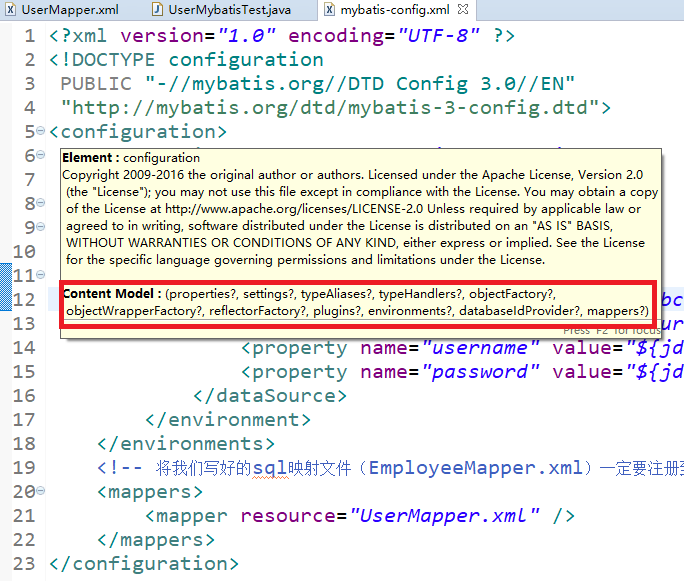
# 导入DTD

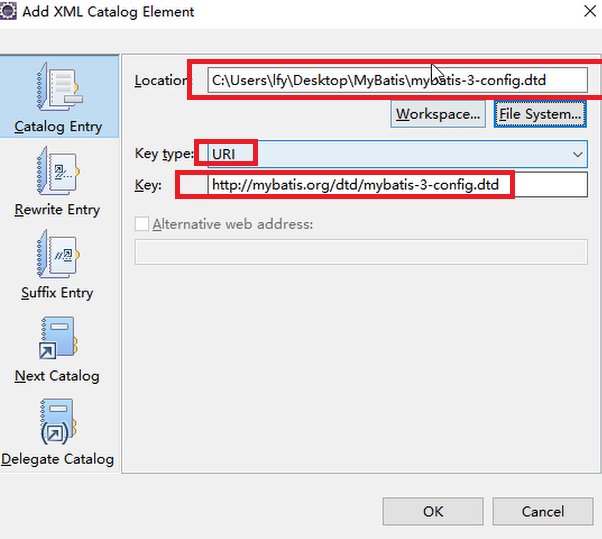
需要导入如下dtd才能再编写xml的时候有补充提示：



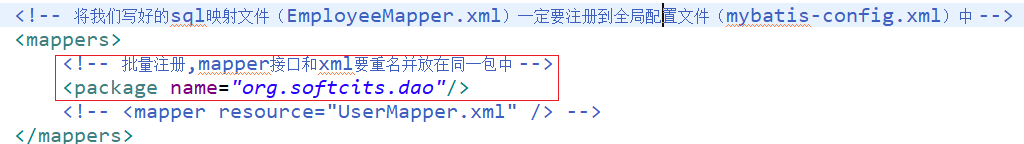


# MyBatis书写标签的顺序是固定的





# 批量注册mapper



# update/insert/delete返回类型

Long/Integer--影响的条数

Boolean--影响条数大于0时返回true

|  |
| --- |
| **public** **interface** UserMapper {  **public** User getUserByUsername(String username);    **public** Long addUser(User user);  **public** Integer updateUser(User user);  **public** Boolean deleteUser(String username);  } |

并且不能在mapper.xml写入resultType属性，因为mybatis已经提供了这三种返回类型支持。

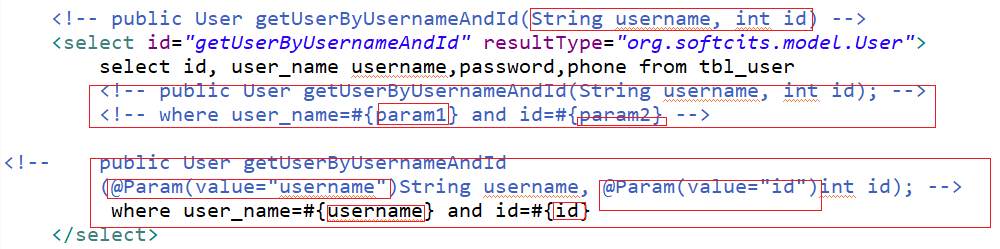
# 非POJO对象传入的多参数查询情况

MyBatis会将多个参数封装成Map

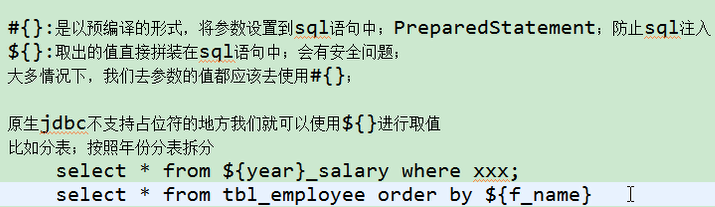
Key: param1,param2.....

Value:才是传入的参数值,在mapper中传入#{param1}，#{param2}

在实际项目经常使用命名参数的方式，即在接口方法的参数里使用@Param(“id”)注解，两种方式如下：



# $与#



# 返回List

Mapper.xml中resultType写List中的类型而不是写List

|  |
| --- |
| <!-- public List<User> getAllUsers(); -->  <select id=*"getAllUsers"* resultType=*"org.softcits.model.User"*>  select \* from tbl\_user  </select> |

# 使用ResultMap

|  |
| --- |
| <?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=*"org.softcits.dao.UserMapperPlus"*>  <resultMap type=*"org.softcits.model.User"* id=*"userDemo"*>  <!-- id定义主键会有优化 column指定那一列 property制定对应javabean的属性 -->  <id column=*"id"* property=*"id"* />  <result column=*"user\_name"* property=*"username"* />  <result column=*"password"* property=*"password"* />  <result column=*"phone"* property=*"phone"* />  </resultMap>  <!-- public User getUserByName(String name); -->  <select id=*"getUserByName"* resultMap=*"userDemo"*>  select \* from tbl\_user where user\_name=#{name}  </select>  </mapper> |

# 关联查询1-1关系

User类中有Group group;成员变量。Mapper如下

|  |
| --- |
| <resultMap type=*"org.softcits.model.User"* id=*"userAndgroup"*>  <!-- id定义主键会有优化 column指定那一列 property制定对应javabean的属性 -->  <id column=*"u\_id"* property=*"id"* />  <result column=*"u\_user\_name"* property=*"username"* />  <result column=*"u\_password"* property=*"password"* />  <result column=*"u\_phone"* property=*"phone"* />  <!-- User的group成员变量的id属性 -->  <result column=*"g\_id"* property=*"group.id"* />  <result column=*"g\_group\_name"* property=*"group.groupName"* />  </resultMap>  <select id=*"getUserAndGroup"* resultMap=*"userAndgroup"*>  select u.id u\_id, u.user\_name u\_user\_name, u.password u\_password, u.phone u\_phone, g.id g\_id, g.group\_name g\_group\_name  from tbl\_user u, tbl\_group g where u.g\_id = g.id and u.user\_name = #{username}  </select> |

# 关联集合封装查询 1-N关系

定义1-N的关系，Group类里有个成员List<User> users，mapper如下

|  |
| --- |
| <?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=*"org.softcits.dao.GroupMapper"*>  <resultMap type=*"org.softcits.model.Group"* id=*"groupAndusers"*>  <id column=*"g\_id"* property=*"id"* />  <result column=*"g\_group\_name"* property=*"groupName"* />    <!-- collection定义集合类型的属性封装ofType中是属性类型 -->  <collection property=*"users"* ofType=*"org.softcits.model.User"*>  <!-- 定义这个集合中元素的封装规则 -->  <id column=*"u\_id"* property=*"id"* />  <result column=*"u\_user\_name"* property=*"username"* />  <result column=*"u\_password"* property=*"password"* />  <result column=*"u\_phone"* property=*"phone"* />  </collection>  </resultMap>  <!-- public class Group {  private int id;  private String groupName;  private List<User> users; -->  <select id=*"getUsersByGroup"* resultMap=*"groupAndusers"*>  SELECT u.id u\_id, u.user\_name u\_user\_name, u.password  u\_password,u.phone u\_phone,  g.id g\_id, g.group\_name g\_group\_name  FROM tbl\_group g  left join tbl\_user u on u.g\_id = g.id  where g.id=#{id}  </select>  </mapper> |

# 动态SQL

## 12.1 if和where标签

|  |
| --- |
| <mapper namespace=*"org.softcits.dao.DynamicSQLUser"*>  <!-- 查询员工，携带哪些属性就按那些属性查 -->  <!--  User u = new User(0, null, null, "0411%");  List<User> ulist = dsu.getUsersByIf(u);  -->  <!-- public List<User> getUsersByIf(User user); -->  <select id=*"getUsersByIf"* resultType=*"org.softcits.model.User"*>  select id, user\_name username,password,phone  from tbl\_user  <!-- 使用where标签解决'and'拼接问题,但是and必须放在每个if前面 -->  <where>  <!-- OGNL表达式 可以参照apache官方文档 -->  <!-- 从参数中取属性进行判断 -->  <if test=*"username!=null"*>  user\_name=#{username}  </if>  <if test=*"phone!=null and phone!=''"*>  and phone like #{phone}  </if>  </where>  </select>  </mapper> |

## 12.2更新操作的set标签

使用set标签就不用担心条件间隔的逗号了

|  |
| --- |
| <!-- public Long updateUser(User user); -->  <update id=*"updateUser"* >  update tbl\_user  <set>  <if test=*"password!=null"* >password=#{password}, </if>  <if test=*"phone!=null"*>phone=#{phone}</if>  </set>  where user\_name=#{username}  </update> |

## 12.3 foreach遍历

|  |
| --- |
| <!--public List<User> getEmpsByForeach(List<Integer> ids); -->  <select id=*"getEmpsByForeach"* resultType=*"org.softcits.model.User"*>  select \* from tbl\_user where id in  <!--  collection：指定要遍历的集合：      1. 如果传入的是单参数且参数类型是一个List的时候，collection属性值为list     2. 如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array     3. 如果传入的参数是多个的时候，我们就需要把它们封装成一个Map了，当然单参数也可  item：将当前遍历出的元素赋值给指定的变量  separator:每个元素之间的分隔符  open：遍历出所有结果拼接一个开始的字符  close:遍历出所有结果拼接一个结束的字符  index:索引。遍历list的时候是index就是索引，item就是当前值  遍历map的时候index表示的就是map的key，item就是map的值    #{变量名}就能取出变量的值也就是当前遍历出的元素  -->  <foreach collection=*"list"* item=*"item\_id"* separator=*","*  open=*"("* close=*")"*>  #{item\_id}  </foreach>  </select> |

## 12.4 sql标签实现语句复用

|  |
| --- |
| <sql id=*"getEmpls"*>select id, user\_name username,password,phone from tbl\_user where id in</sql>    <!--public List<User> getEmpsByForeach(List<Integer> ids); -->  <select id=*"getEmpsByForeach"* resultType=*"org.softcits.model.User"*>  <include refid=*"getEmpls"*></include> |

# MyBatis缓存

## 13.1 一级缓存(本地缓存) -> 存在session中

## 13.2 二级缓存(全局缓存) -> 存在namespace中，即mapper

查出的数据会先放在当前session，只有session.close()后才能转移到二级缓存中，从而共享给其他session，所以session查询也会先看二级缓存再看一级缓存

开启二级缓存步骤

1. 在全局配置中增加

|  |
| --- |
| <settings>  <setting name=*"cacheEnabled"* value=*"true"*/>  </settings> |

1. 下载Ehcache.jar和mybatis-ehcache.jsr,配置ehcache.xml,在mapper.xml中整合EhcacheCache

|  |
| --- |
| <cache type="org.mybatis.caches.ehcache.EhcacheCache"></cache> |

1. POJO需要序列化，才能给用户每份数据拷贝

|  |
| --- |
| **public** **class** User **implements** Serializable{  **private** Integer id;  **private** String username;  **private** String password;  **private** String phone; |

# 与SpringMVC整合

## 加入整合包

|  |
| --- |
| <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.2.2</version>  </dependency> |

## 在web.xml中的<web-app>内引入spring工厂的配置

|  |
| --- |
| <!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  <web-app>  <!--Spring配置： needed for ContextLoaderListener -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <!-- Bootstraps the root web application context before servlet initialization -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <!-- 引入springMVC的配置 -->  <servlet>  <servlet-name>demo</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  </servlet>  <!-- 如果没有精确匹配servlet，则使用以下默认servlet -->  <servlet-mapping>  <servlet-name>demo</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

## 创建springMVC的配置demo-servlet.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd">  <!-- <bean name="/welcome" class="org.softcits.controller.WelcomeController"  /> -->  <!-- 声明SpringMVC会去那些包中扫描annotation -->  <context:component-scan base-package="org.softcits.controller"  use-default-filters="false">  <!-- 只扫描控制器 -->  <context:include-filter type="annotation"  expression="org.springframework.stereotype.Controller" />  </context:component-scan>  <!-- 开启SpringMVC的Annotation机制 -->  <mvc:annotation-driven />  <!-- 静态文件映射,否则一切静态文件的请求都会被web.xml中的servlet拦截 -->  <mvc:resources location="/resources/" mapping="/resources/\*\*" />  <bean  class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/WEB-INF/jsp/" />  <property name="suffix" value=".jsp" />  </bean>  </beans> |

## 创建applicationContext.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:mybatis-spring=*"http://mybatis.org/schema/mybatis-spring"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-spring-1.2.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd"*>  <!-- Spring希望管理所有的业务逻辑组件，等。。。 -->  <context:component-scan base-package=*"org.softcits"*>  <context:exclude-filter type=*"annotation"*  expression=*"org.springframework.stereotype.Controller"* />  </context:component-scan>  <!-- 引入数据库的配置文件 -->  <context:property-placeholder location=*"classpath:dbconfig.properties"* />  <!-- 数据库连接池 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*  destroy-method=*"close"*>  <property name=*"url"* value=*"${jdbc.url}"* />  <property name=*"username"* value=*"${jdbc.username}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  <property name=*"driverClassName"* value=*"${jdbc.driver}"* />  <property name=*"maxActive"* value=*"10"* />  <property name=*"minIdle"* value=*"5"* />  </bean>  <!-- spring事务管理 -->  <bean id=*"dataSourceTransactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据连接关闭都是事务来控制 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 开启基于注解的事务 -->  <!-- transaction-manager The bean name of the PlatformTransactionManager  that is to be used to drive transactions. This attribute is not required,  and only needs to be specified explicitly if the bean name of the desired  PlatformTransactionManager is not 'transactionManager'. -->  <tx:annotation-driven transaction-manager=*"dataSourceTransactionManager"* />  <!-- 整合mybatis 目的：1、spring管理所有组件。mapper的实现类。 service==>Dao @Inject:自动注入mapper；  2、spring用来管理事务，spring声明式事务 -->  <!--创建出SqlSessionFactory对象 -->  <bean id=*"sqlSessionFactoryBean"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  <!-- configLocation指定全局配置文件的位置 -->  <property name=*"configLocation"* value=*"classpath:mybatis-springmvc.xml"*></property>  <!--mapperLocations: 指定mapper文件的位置 -->  <!-- <property name="mapperLocations" value="classpath:org/softcits/dao/\*.xml"></property> -->  </bean>  <!--配置一个可以进行批量执行的sqlSession -->  <bean id=*"sqlSession"* class=*"org.mybatis.spring.SqlSessionTemplate"*>  <constructor-arg name=*"sqlSessionFactory"* ref=*"sqlSessionFactoryBean"*></constructor-arg>  <constructor-arg name=*"executorType"* value=*"BATCH"*></constructor-arg>  </bean>  <!-- 扫描所有的mapper接口的实现，让这些mapper能够自动注入； base-package：指定mapper接口的包名 -->  <mybatis-spring:scan base-package=*"org.softcits.dao"* />  </beans> |

## 创建dbconfig.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/test  jdbc.username=root  jdbc.password=123456 |

## 创建springmvc-mybatis.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>  <settings>  <!--开启一级缓存-->  <setting name=*"cacheEnabled"* value=*"true"*/>  </settings>  </configuration> |

# MaBatis与SpringMVC整合遇到的问题

输出报错: Invalid bound statement(not found)

网上很多说mapper的namespace没有加入interface的全限定名，但我不是这个问题，研究了很久，才发现maven启动的时候没有将mapper.xml文件拷贝到编译后的target/classess文件夹下，所以找不到mapper

解决方法：加入maven拷贝资源文件的插件

|  |
| --- |
| <build>  <finalName>spring\_training</finalName>  <resources>  <resource>  <!--编译时拷贝maven中src/main/resources的资源文件到target/classess下-->  <directory>src/main/resources</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  <!--编译时拷贝maven中src/main/java的资源文件到target/classess下-->  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  </resources>  <pluginManagement>  <plugins>  <!-- 资源文件拷贝插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-resources-plugin</artifactId>  <version>2.7</version>  <configuration>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>${project.build.sourceEncoding}</encoding>  <compilerArguments>  <verbose />  <bootclasspath>${java.home}/lib/rt.jar;${java.home}/lib/jce.jar</bootclasspath>  </compilerArguments>  </configuration>  </plugin>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  </plugin>  </plugins>  </pluginManagement>  </build> |

# 逆向工程

## 步骤:

### 导入依赖包

|  |
| --- |
| <dependency>  <groupId>org.mybatis.generator</groupId>  <artifactId>mybatis-generator-core</artifactId>  <version>1.3.2</version>  </dependency> |

### maven项目中在src/main/resources下创建mbg.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>  <context id=*"DB2Tables"* targetRuntime=*"MyBatis3"*>  <!-- jdbcConnection：指定如何连接到目标数据库 -->  <jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*  connectionURL=*"jdbc:mysql://localhost:3306/test?allowMultiQueries=true"*  userId=*"root"*  password=*"123456"*>  </jdbcConnection>  <!-- javaModelGenerator：指定javaBean的生成策略  targetPackage="test.model"：目标包名  targetProject="\MBGTestProject\src"：目标工程  -->  <javaModelGenerator targetPackage=*"org.softcits.mbg.model"*  targetProject=*".\src\main\java"*>  <property name=*"enableSubPackages"* value=*"true"* />  <property name=*"trimStrings"* value=*"true"* />  </javaModelGenerator>  <!-- sqlMapGenerator：sql映射生成策略： -->  <sqlMapGenerator targetPackage=*"org.softcits.mbg.mapper"*  targetProject=*".\src\main\java"*>  <property name=*"enableSubPackages"* value=*"true"* />  </sqlMapGenerator>  <!-- javaClientGenerator:指定mapper接口所在的位置 -->  <javaClientGenerator type=*"XMLMAPPER"* targetPackage=*"org.softcits.mbg.mapper"*  targetProject=*".\src\main\java"*>  <property name=*"enableSubPackages"* value=*"true"* />  </javaClientGenerator>  <!-- 指定要逆向分析哪些表：根据表要创建javaBean -->  <table tableName=*"tbl\_user"* domainObjectName=*"MbgUser"*></table>  <table tableName=*"tbl\_group"* domainObjectName=*"MbgGroup"*></table>  </context>  </generatorConfiguration> |

### 在src/main/test下创建运行类

|  |
| --- |
| **package** org.softcits.mbg.test;  **import** java.io.File;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.apache.ibatis.io.Resources;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.apache.ibatis.session.SqlSessionFactoryBuilder;  **import** org.junit.Test;  **import** org.mybatis.generator.api.MyBatisGenerator;  **import** org.mybatis.generator.config.Configuration;  **import** org.mybatis.generator.config.xml.ConfigurationParser;  **import** org.mybatis.generator.internal.DefaultShellCallback;  **public** **class** MbgMybatisTest {  **public** SqlSessionFactory getSqlSessionFactory() **throws** IOException {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.*getResourceAsStream*(resource);  **return** **new** SqlSessionFactoryBuilder().build(inputStream);  }  @Test  **public** **void** testMbg() **throws** Exception {  List<String> warnings = **new** ArrayList<String>();  **boolean** overwrite = **true**;  File configFile = **new** File(".\\src\\main\\resources\\mbg.xml");  ConfigurationParser cp = **new** ConfigurationParser(warnings);  Configuration config = cp.parseConfiguration(configFile);  DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);  MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config,  callback, warnings);  myBatisGenerator.generate(**null**);  }  } |

## 测试

### 创建测试类

|  |
| --- |
| @Test  //简单测试  **public** **void** testMyBatis3Simple() **throws** IOException{  SqlSessionFactory sqlSessionFactory = getSqlSessionFactory();  SqlSession openSession = sqlSessionFactory.openSession();  **try**{  MbgUserMapper mapper = openSession.getMapper(MbgUserMapper.**class**);  List<MbgUser> list = mapper.selectByExample(**null**);  **for** (MbgUser mu : list) {  System.***out***.println(mu.getUserName());  }  }**finally**{  openSession.close();  }  }    @Test  //复杂的含两个or关系的条件测试  **public** **void** testMyBatis3() **throws** IOException{  SqlSessionFactory sqlSessionFactory = getSqlSessionFactory();  SqlSession openSession = sqlSessionFactory.openSession();  **try**{  MbgUserMapper mapper = openSession.getMapper(MbgUserMapper.**class**);  //xxxExample就是封装查询条件的  //1、查询所有  //List<MbgUsesr> users = mapper.selectByExample(null);  //2、查询员工名字中有e字母的，和员工性别是1的  //封装员工查询条件的example  MbgUserExample example = **new** MbgUserExample();  //创建一个Criteria，这个Criteria就是拼装查询条件  //select id, last\_name, email, gender, d\_id from tbl\_employee  //WHERE ( last\_name like ? and gender = ? ) or email like "%e%"  Criteria criteria = example.createCriteria();  criteria.andUserNameLike("%e%");  criteria.andGIdEqualTo(1);    Criteria criteria2 = example.createCriteria();  criteria2.andGIdIn(Arrays.*asList*(2,3));    //拼接条件  example.or(criteria2);    List<MbgUser> list = mapper.selectByExample(example);  **for** (MbgUser mu : list) {  System.***out***.println(mu.getUserName() + " " + mu.getgId());  }    }**finally**{  openSession.close();  }  } |

### 创建mybatis单机配置文件(即junit使用的未和spring集成的)"mybatis-config.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>  <properties resource=*"dbconfig.properties"*>  </properties>  <settings>  <setting name=*"cacheEnabled"* value=*"true"*/>  </settings>  <environments default=*"development"*>  <environment id=*"development"*>  <transactionManager type=*"JDBC"* />  <!-- mybatis默认数据源,也可以配置c3p0/dbcp -->  <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>  <!-- 将我们写好的sql映射文件（EmployeeMapper.xml）一定要注册到全局配置文件（mybatis-config.xml）中 -->  <mappers>  <!-- 批量注册,mapper接口和xml要重名并放在同一包中 -->  <package name=*"org.softcits.mbg.mapper"*/>  <!-- <mapper resource="UserMapper.xml" /> -->  </mappers>  </configuration> |

# 分页

## 引入jar

|  |
| --- |
| <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>3.4.2-fix</version>  </dependency> |

## 在mybatis-config.xml全局配置中添加pageHelper,注意添加标签顺序

|  |
| --- |
| <plugins>  <!-- com.github.pagehelper为PageHelper类所在包名 -->  <plugin interceptor=*"com.github.pagehelper.PageHelper"*>  <!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库 -->  <property name=*"dialect"* value=*"mysql"* />  </plugin>  </plugins> |

## 测试类

|  |
| --- |
| //测试分页插件  @Test  **public** **void** testPageHelp() **throws** IOException{  SqlSessionFactory sqlSessionFactory = getSqlSessionFactory();  SqlSession openSession = sqlSessionFactory.openSession();  **try**{  MbgUserMapper mapper = openSession.getMapper(MbgUserMapper.**class**);  MbgUserExample example = **new** MbgUserExample();    PageHelper.*startPage*(2, 2);    List<MbgUser> list = mapper.selectByExample(example);  **for** (MbgUser mu : list) {  System.***out***.println(mu.getUserName());  }    PageInfo<MbgUser> info = **new** PageInfo<>(list);  System.***out***.println("当前页码："+info.getPageNum());  System.***out***.println("总记录数："+info.getTotal());  System.***out***.println("每页的记录数："+info.getPageSize());  System.***out***.println("总页码："+info.getPages());  System.***out***.println("是否第一页："+info.isIsFirstPage());  }**finally**{  openSession.close();  }  } |