# 1 MyBatis简介

## 1.1 MyBatis

MyBatis 本是[apache](http://baike.baidu.com/view/28283.htm)的一个开源项目[iBatis](http://baike.baidu.com/view/628102.htm), 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis，实质上Mybatis对ibatis进行一些改进。

MyBatis是一个优秀的持久层框架，它对jdbc的操作数据库的过程进行封装，使开发者只需要关注 SQL 本身，而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。

对jdbc的封装框架有哪些：Hibernate,dbutils,jdbcTemplate[spring]，mybatis

原理：Mybatis通过**xml或注解**的方式将要执行的各种statement（statement、preparedStatemnt、CallableStatement）配置起来，并通过java对象和statement中的sql进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回。

## 1.2 jdbc程序代码

|  |
| --- |
| **public** **class** TestDemo {  **public** **static** **void** main(String[] args) {  Connection conn=**null**;  PreparedStatement preparedStatement=**null**;  ResultSet resultSet=**null**;  **try** {  //1.加载数据库驱动  Class.*forName*("com.mysql.jdbc.Driver");  //2.通过驱动获取连接  conn=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/wsshop", "root","root");  //3.定义sql语句  String sql="select \* from ws\_user where username=?";  //4.获取预处理statement  preparedStatement=conn.prepareStatement(sql);  //5.设置参数  preparedStatement.setString(1, "zhangfei");  //6.向数据库发送SQL，查询得到结果集  resultSet=preparedStatement.executeQuery();  //7.遍历查询结果集  **while**(resultSet.next()) {  System.***out***.println(resultSet.getString("username")+":");  }    } **catch** (Exception e) {    }**finally** {  //9.释放资源  **try** {  preparedStatement.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  } |

**存在的问题**

* 数据库连接频繁开启和关闭，会严重影响数据库的性能。
* 代码中存在硬编码，分别是数据库部分的硬编码和SQL执行部分的硬编码。

## 1.3 MyBatis的框架核心

|  |
| --- |
| 1. mybatis配置文件，包括**Mybatis全局配置文件和Mybatis映射文件**，其中全局配置文件配置了数据源、事务等信息；映射文件配置了SQL执行相关的 信息。 2. mybatis通过读取配置文件信息（全局配置文件和映射文件），构造出**SqlSessionFactory，**即会话工厂。 3. 通过SqlSessionFactory，可以创建**SqlSession**即会话。Mybatis是通过SqlSession来操作数据库的。 4. SqlSession本身不能直接操作数据库，它是通过底层的**Executor**执行器接口来操作数据库的。Executor接口有两个实现类，一个是普通执行器，一个是**缓存执行器（默认）**。 5. Executor执行器要处理的SQL信息是封装到一个底层对象**MappedStatement**中。该对象包括：SQL语句、输入参数映射信息、输出结果集映射信息。其中输入参数和输出结果的映射类型包括**HashMap集合对象、POJO对象类型**。 |

# Mybatis入门

## 2.1 环境准备

### 下载MyBatis

mybaits的代码由github.com管理，下载地址：<https://github.com/mybatis/mybatis-3/releases>

### 创建项目导包

导入下面的包

|  |
| --- |
| <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.4</version>  </dependency>    <!-- mybatis/spring包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.2.2</version>  </dependency>  <!-- 导入Mysql数据库链接jar包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.30</version>  </dependency>  <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency> |
|  |

### 添加log4j.properties

Mybatis使用的日志包是log4j的，所以需要添加log4j.properties。

在classpath下创建log4j.properties如下：【文件内容可以从mybatis-3.2.7.pdf中拷贝】

|  |
| --- |
| <!-- 日志文件管理包 -->  <!-- log start -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.12</version>  </dependency> |

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

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

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

日志级别在开发阶段设置成DEBUG，在生产阶段设置成INFO或者ERROR。

## 2.2 开发步骤

|  |
| --- |
| 1. 创建PO（model）类，根据需求创建； 2. 创建全局配置文件SqlMapConfig.xml； 3. 编写映射文件； 4. 加载映射文件，在SqlMapConfig.xml中进行加载； 5. 编写测试程序，即编写Java代码，连接并操作数据库。   思路：   1. 读取配置文件； 2. 通过SqlSessionFactoryBuilder创建SqlSessionFactory会话工厂。 3. 通过SqlSessionFactory创建SqlSession。 4. 调用SqlSession的操作数据库方法。 5. 关闭SqlSession。 |

### 创建PO类

|  |
| --- |
| **public** **class** User {  **private** Integer id;  **private** String username;  **private** String password;  **private** String email;  **private** String phone;  **private** String question;  **private** String answer;  **private** Integer role;  **private** Date createTime;  **private** Date updateTime; |

### 创建SqlMapConfig.xml

在classpath(src)下，创建SqlMapConfig.xml文件【SqlMapConfig.xml（文件头可以从mybatis-3.2.7.pdf文档的2.1.2小节中拷贝）】

|  |
| --- |
| <?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mybatis的环境信息 -->  <environments default=*"development"*>  <environment id=*"development"*>  <!-- 配置JDBC事务控制，由mybatis进行管理 -->  <transactionManager type=*"JDBC"*></transactionManager>  <!-- 配置数据源，采用dbcp连接池 -->  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"com.mysql.jdbc.Driver"*/>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/wsshop?useUnicode=true&amp;characterEncoding=utf8"*/>  <property name=*"username"* value=*"root"*/>  <property name=*"password"* value=*"root"*/>  </dataSource>  </environment>  </environments>  </configuration> |

### 映射文件

在classpath下，创建sqlmap文件夹。在sqlmap目录下，创建User.xml映射文件。

**【Mybatis的映射文件头（可以从mybatis-3.2.7.pdf文件中拷贝）】**

|  |
| --- |
| <?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，它的作用就是对SQL进行分类化管理，可以理解为SQL隔离  注意：使用mapper代理开发时，namespace有特殊且重要的作用  -->  <mapper namespace=*"test"*>  <!--  [id]：statement的id，要求在命名空间内唯一  [parameterType]：入参的java类型  [resultType]：查询出的单条结果集对应的java类型  [#{}]： 表示一个占位符?  [#{id}]：表示该占位符待接收参数的名称为id。注意：如果参数为简单类型时，#{}里面的参数名称可以是任意定义  -->  <select id=*"findUserById"* parameterType=*"int"* resultType=*"com.neuedu.mybatis.demo.User"*>  select \* from ws\_user where id=#{id}  </select></mapper> |

### 配置文件加载映射文件

|  |
| --- |
| <mappers>  <mapper resource=*"User.xml"*/>  </mappers> |

### 测试类

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** IOException {  //a)读取配置文件  InputStream is = Resources.*getResourceAsStream*("SqlMapConfig.xml");  SqlSessionFactory ssf = **new** SqlSessionFactoryBuilder().build(is);  SqlSession session = ssf.openSession();  User user=session.selectOne("findUserById",1);  System.***out***.println(user.getQuestion());  session.close();  } |

## 2.3 更多案例讲解

### 模糊查询用户信息

|  |
| --- |
| <!--  [${}]：表示拼接SQL字符串  [${value}]：表示要拼接的是简单类型参数。  注意：  1、如果参数为简单类型时，${}里面的参数名称必须为value  2、${}会引起SQL注入，一般情况下不推荐使用。但是有些场景必须使用${}，比如order by ${colname}  -->  <select id=*"findUserByName"* parameterType=*"string"* resultType=*"com.neuedu.mybatis.demo.User"*>  select \* from ws\_user where username like '%${value}%'  </select> |
| List<User> users=session.selectList("findUserByName","o");  **for** (User user : users) {  System.***out***.println(user.getUsername());  } |

### 插入用户信息

|  |
| --- |
| <insert id=*"insertUser"* parameterType=*"com.neuedu.mybatis.demo.User"*>  insert into ws\_user(username,password,role,create\_time,update\_time) values  (#{username},#{password},#{role},#{create\_time},#{update\_time})  </insert> |
| User user=**new** User();  user.setUsername("dddd");  user.setPassword("12344");  user.setRole(2);  user.setCreate\_time(**new** Date());  user.setUpdate\_time(**new** Date());  session.insert("insertUser",user);  session.commit(); |

### 删除用户

|  |
| --- |
| <delete id=*"deleteUser"* parameterType=*"int"*>  delete from ws\_user where id=#{id}  </delete> |
| session.delete("test.deleteUser", 23);  session.commit(); |

### 更新用户

|  |
| --- |
| <update id=*"updateUser"* parameterType=*"com.neuedu.mybatis.demo.User"*>  update user set username=#{username},password=#{password} where id=#{id}  </update> |
| session.update("test.updateUser", user);  session.commit(); |

|  |
| --- |
| **parameterType和resultType**  parameterType指定输入参数的java类型，可以填写别名或Java类的全限定名。  resultType指定输出结果的java类型，可以填写别名或Java类的全限定名。  **#{}和${}**  #{}：相当于预处理中的占位符？。  #{}里面的参数表示接收java输入参数的名称。  #{}可以接受HashMap、POJO类型的参数。  当接受简单类型的参数时，#{}里面可以是value，也可以是其他。  #{}可以防止SQL注入。  ${}：相当于拼接SQL串，对传入的值不做任何解释的原样输出。  ${}会引起SQL注入，所以要谨慎使用。  ${}可以接受HashMap、POJO类型的参数。  当接受简单类型的参数时，${}里面只能是value。  **selectOne和selectList**  selectOne：只能查询0或1条记录，大于1条记录的话，会报错：  selectList：可以查询0或N条记录 |

## 2.4 MyBatis的Dao编写 【一般不用，有更多好方式】

### dao

|  |
| --- |
| **public** **class** UserDaoImpl **implements** UserDao{  **private** SqlSessionFactory ssf;    **public** SqlSessionFactory getSsf() {  **return** ssf;  }  **public** **void** setSsf(SqlSessionFactory ssf) {  **this**.ssf = ssf;  }    **public** UserDaoImpl(SqlSessionFactory ssf) {  **super**();  **this**.ssf = ssf;  }  @Override  **public** **void** save(User user) {  }  @Override  **public** User findUserById(Integer id) {  SqlSession session=ssf.openSession();  User user=session.selectOne("findUserById",id);  **return** user;  }  } |
|  |

### 测试

|  |
| --- |
| **public** **class** UserDaoImplTest {  **private** SqlSessionFactory ssf;    @Before  **public** **void** setup() **throws** IOException {  InputStream resource = Resources.*getResourceAsStream*("SqlMapConfig.xml");  ssf=**new** SqlSessionFactoryBuilder().build(resource);  }  @Test  **public** **void** testSave() {  }  @Test  **public** **void** testFindUserById() {  UserDao userDao=**new** UserDaoImpl(ssf);  User user = userDao.findUserById(27);  System.***out***.println(user.getUsername());  }  } |

## 2.5 MyBatis的Dao编写【mapper代理方式实现】

Mapper代理的开发方式，程序员只需要编写mapper接口（相当于dao接口）即可。Mybatis会自动的为mapper接口生成动态代理实现类。

**不过要实现mapper代理的开发方式，需要遵循一些开发规范。**

### 开发规范

|  |
| --- |
| 1. mapper接口的全限定名要和mapper映射文件的namespace的值相同。 2. mapper接口的方法名称要和mapper映射文件中的statement的id相同； 3. mapper接口的方法参数只能有一个，且类型要和mapper映射文件中statement的parameterType的值保持一致。 4. mapper接口的返回值类型要和mapper映射文件中statement的resultType值或resultMap中的type值保持一致；   通过规范式的开发mapper接口，可以解决原始dao开发当中存在的问题：   * 模板代码已经去掉； * 剩下去不掉的操作数据库的代码，其实就是一行代码。这行代码中硬编码的部分，通过第一和第二个规范就可以解决。 |

### 编写步骤

### 第一步：【】

重新写个UserMapper配置文件和定义mapper映射文件UserMapper.xml（内容同Users.xml，**除了namespace的值**），放到新创建的目录mapper下。

|  |
| --- |
|  |

### 第二步：【添加映射配置文件】

|  |
| --- |
| <mappers>  <mapper resource=*"com/neuedu/demo/UserMapper.xml"*/>  </mappers> |

### 第三步：测试

|  |
| --- |
| **public** **class** UserDaoImplTest {  **private** SqlSessionFactory ssf;    @Before  **public** **void** setup() **throws** IOException {  InputStream resource = Resources.*getResourceAsStream*("SqlMapConfig.xml");  ssf=**new** SqlSessionFactoryBuilder().build(resource);  }  @Test  **public** **void** testSave() {  }  @Test  **public** **void** testFindUserById() {  SqlSession session=ssf.openSession();  UserMapper userMapper = session.getMapper(UserMapper.**class**);  User user = userMapper.findUserById(27);  System.***out***.println(user.getPassword());  }  } |

# 3 全局配置文件其它配置

## 3.1 properties数据库文件配置

### 在src下配置个db.properties文件

|  |
| --- |
| driverClass=com.mysql.jdbc.Driver  url=jdbc:mysql://localhost:3306/wsshop?useUnicode=true&amp;characterEncoding=utf8  username=root  password=root |

### 修改全局的配置文件

|  |
| --- |
| <?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"*></properties>  <environments default=*"development"*>  <environment id=*"development"*>  <transactionManager type=*"JDBC"*/>  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"${driverClass}"*/>  <property name=*"url"* value=*"${url}"*/>  <property name=*"username"* value=*"${username}"*/>  <property name=*"password"* value=*"${password}"*/>  </dataSource>  </environment>  </environments>  <mappers>  <mapper resource=*"com/neuedu/demo/UserMapper.xml"*/>  </mappers>  </configuration> |

## 3.2 setting【了解】

|  |
| --- |
|  |

### 具体配置详解

|  |
| --- |
|  |

## 3.3 typeAliases

别名是使用是为了在映射文件中，更方便的去指定参数和结果集的类型，不再用写很长的一段全限定名。

### mybatis支持的别名

|  |  |
| --- | --- |
| 别名 | 映射的类型 |
| \_byte | byte |
| \_long | long |
| \_short | short |
| \_int | int |
| \_integer | int |
| \_double | double |
| \_float | float |
| \_boolean | boolean |
| string | String |
| byte | Byte |
| long | Long |
| short | Short |
| int | Integer |
| integer | Integer |
| double | Double |
| float | Float |
| boolean | Boolean |
| date | Date |
| decimal | BigDecimal |
| bigdecimal | BigDecimal |

### 自定义别名

|  |
| --- |
|  |

## mappers

### <mapper resource=’’/>

使用相对于类路径的资源

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

### <package name=’’/>（推荐）

注册指定包下的所有映射文件

如：<package name="com.neuedu.mybatis.mapper"/>

**注意：此种方法要求mapper接口和mapper映射文件要名称相同，且放到同一个目录下**;

# Mybatis的映射文件

## 4.1 输入映射ParameterType

指定输入参数的java类型，可以使用别名或者类的全限定名。它可以接收**简单类型,POJO对象、HashMap**。

### 传递简单类型

根据用户ID查询用户信息。

|  |
| --- |
|  |

### 传递POJO对象

|  |
| --- |
| <insert id=*"insertUser"* parameterType=*"user"*>  INSERT INTO `wsshop`.`ws\_user` (`username`,`password`,`role`,  `create\_time`,`update\_time`) VALUES(#{username},#{password},  #{role},#{create\_time},#{update\_time})  </insert> |

### 传递POJO包装对象

开发中通过pojo传递查询条件 ，查询条件是综合的查询条件，不仅包括用户查询条件还包括其它的查询条件（比如将用户购买商品信息也作为查询条件），这时可以使用包装对象传递输入参数。

**需求**

综合查询用户信息，需要传入查询条件复杂，比如（用户信息、订单信息、商品信息）。

vo:键值对对象，相对于kv

po:persist object 持久化对象

pojo:简单的java对象

entity:实体

#### 定义POJO包装类

|  |
| --- |
|  |

#### 修改UserMapper.java

|  |
| --- |
|  |

#### 修改UsrMappler.xml

|  |
| --- |
| <select id=*"findUserList"* parameterType=*"userQueryVO"* resultType=*"user"*>  select \* from ws\_user where username like "%${user.username}%" and  email like "%${user.email}%"  </select>  UserQueryVO的属性 |

#### 测试

|  |
| --- |
| **public** **class** TestDemo {  **private** SqlSessionFactory ssf;  @Before  **public** **void** setup() **throws** IOException {  InputStream config = Resources.*getResourceAsStream*("SqlMapConfig.xml");  ssf=**new** SqlSessionFactoryBuilder().build(config);  }  @Test  **public** **void** testfindUserList() {  SqlSession session=ssf.openSession();  //通过session获取代理  UserMapper userMapper = session.getMapper(UserMapper.**class**);  UserQueryVO userQueryVO=**new** UserQueryVO();  User user=**new** User();  user.setUsername("o");  user.setEmail("o");  userQueryVO.setUser(user);  List<User> list = userMapper.findUserList(userQueryVO);  **for** (User u : list) {  System.***out***.println(u.getEmail());  }  }  } |

### 传递Map对象

|  |
| --- |
| 修改UserMapper.java |
|  |
| 修改UserMapper.xml |
| <select id=*"findUserListByMap"* parameterType=*"hashmap"* resultType=*"user"*>  select \* from ws\_user where username like "%${username}%" and  email like "%${email}%"  </select> |
| 测试 |
| @Test  **public** **void** testFindListByMap() {  SqlSession session = ssf.openSession();  UserMapper userMapper = session.getMapper(UserMapper.**class**);  HashMap<String,Object>map=**new** HashMap<String, Object>();  map.put("username","o");  map.put("email","o");  List<User> list = userMapper.findUserListByMap(map);  **for** (User u : list) {  System.***out***.println(u.getUsername());  }  }  } |

## 4.2 输出映射 resultType/resultMap

### resultType

使用resultType进行结果映射时，查询的列名和映射的pojo属性名完全一致，该列才能映射成功。

如果查询的列名和映射的pojo属性名全部不一致，则不会创建pojo对象；

如果查询的列名和映射的pojo属性名有一个一致，就会创建pojo对象。

#### 输出简单类型

当输出结果只有一列时，可以使用ResultType指定简单类型作为输出结果类型。

**案例：输出一个count(\*)**

|  |
| --- |
| **UserMapper.java** |
|  |
| **UserMapper.xml** |
|  |
| 测试 |
|  |

#### 输出POJO单个对象

|  |
| --- |
|  |
|  |

#### 输出POJO列表

|  |
| --- |
|  |
|  |

**总结：**

输出单个pojo对象和pojo列表时，mapper映射文件中的resultType的类型是一样的，mapper接口的方法返回值不同。

同样的mapper映射文件，返回单个对象和对象列表时，mapper接口在生成动态代理的时候，会根据返回值的类型，决定调用selectOne方法还是selectList方法。

### resultMap

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

1. 定义resultMap

2、使用resultMap作为statement的输出映射类型

|  |
| --- |
| UserMapper.java |
|  |
| UserMapper.xml |
| <resultMap type=*"user"* id=*"userResultMap"*>  <id column=*"id"* property=*"id"*/>  <result column=*"username"* property=*"username"*/>  <result column=*"password"* property=*"password"*/>  </resultMap>  <select id=*"findUserByIdResultMap"* parameterType=*"int"* resultMap=*"userResultMap"*>  select \* from ws\_user where id=#{id}  </select> |
| 测试 |
|  |

# 5.mybaties整合spring

SSM:SpringMVC + Spring + MyBatis

Spring 3.2

Mybaties 3.2.7

## 5.1 创建工程导包

### 导入mybaties包

|  |
| --- |
| mybaties核心包  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency> |
|  |

### 导入mysql数据库驱动

|  |
| --- |
|  |

### 数据库dbcp连接池

|  |
| --- |
|  |

### 导入spring+mvc包

|  |
| --- |
| <properties>  <!-- spring版本号 -->  <spring.version>5.0.5.RELEASE</spring.version>  <!-- mybatis版本号 -->  <mybatis.version>3.2.6</mybatis.version>  <!-- log4j日志文件管理包版本 -->  <slf4j.version>1.7.7</slf4j.version>  <log4j.version>1.2.17</log4j.version>  </properties>  <dependencies>  <!-- spring核心包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</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-oxm</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- 事务处理 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</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-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- mybatis核心包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <!-- mybatis/spring包 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.2.2</version>  </dependency>  <!-- 导入Mysql数据库链接jar包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.30</version>  </dependency>  <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency>  <!-- JSTL标签类 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  <!-- 日志文件管理包 -->  <!-- log start -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>${log4j.version}</version>  </dependency>  <!-- 格式化对象，方便输出日志 Json-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.1.41</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <!-- log end -->  <!-- 映入JSON -->  <dependency>  <groupId>org.codehaus.jackson</groupId>  <artifactId>jackson-mapper-asl</artifactId>  <version>1.9.13</version>  </dependency>  <!-- 上传组件包 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>2.4</version>  </dependency>  <!-- commons-codec是Apache开源组织提供的用于摘要运算、编码解码的包。常见的编码解码工具Base64、MD5、Hex、SHA1、DES -->  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.9</version>  </dependency>  <!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>1.4</version>  </dependency>  <dependency>  <groupId>jetty</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5-6.0.2</version>  </dependency>  <!-- junit -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.9</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  </plugins>  </build> |

### Mybatis-spring整合包

|  |
| --- |
|  |

## 5.2 配置mybatis的核心配置文件

核心配置文件、创建User模型、映射文件，

|  |
| --- |
|  |
| <?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>  <typeAliases>  <typeAlias type=*"com.neuedu.mybatis.demo.User"* alias=*"user"*/>  <typeAlias type=*"com.neuedu.mybatis.demo.UserQueryVO"* alias=*"userQueryVO"*/>  </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:3306/wsshop"*/>  <property name=*"username"* value=*"root"*/>  <property name=*"password"* value=*"root"*/>  </dataSource>  </environment>  </environments>  <mappers>  <mapper resource=*"com/neuedu/mybatis/demo/UserMapper.xml"*/>  </mappers>  </configuration> |
|  |

## 5.3 spring的数据源

|  |
| --- |
| <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.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx.xsd "*>      <!-- 1.配置数据库，dbcp数据库连接池 -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*/>  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/wsshop"*/>  <property name=*"username"* value=*"root"*/>  <property name=*"password"* value=*"root"*/>  <property name=*"maxActive"* value=*"10"*/>  <!--最大空闲数 -->  <property name=*"maxIdle"* value=*"5"*/>  </bean>  </beans> |

## 5.4 spring配置SqlSessionFactory

|  |
| --- |
| <!-- 3.配置SqlSessionFactory -->  <bean id=*"sqlSessionFactoryBean"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"SqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean> |

## 5.5 换成Mapper接口整合dao

### 创建Mapper映射文件

|  |
| --- |
| UserMapper |
| **public** **interface** UserMapper {  **public** **void** save(User user);  **public** User findUserById(**int** id);  **public** List<User> findUserListByMap(Map<String, Object>map);  **public** User findUserByIdResultMap(**int** id);  } |
| UserMapper.xml |
| <mapper namespace=*"com.neuedu.spring.mybatis.demo.UserMapper"*>  <select id=*"findById"* parameterType=*"int"* resultType=*"user"*>  select \* from ws\_user where id=#{id}  </select>  </mapper> |

### 核心配置文件加载映射文件

|  |
| --- |
| <mappers>  <!-- <mapper resource="com/neuedu/spring/mybatis/demo/UserMapper.xml"/>-->  <package name=*"com.neuedu.spring.mybatis.demo"*/>  </mappers> |

### Spring配置MapperFactoryBean

使用工厂Bean生成userMapper对象

|  |
| --- |
| <!-- 4.配置 -->  <bean id=*"userMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>  <property name=*"mapperInterface"* value=*"com.neuedu.spring.mybatis.demo.UserMapper"*/>  <property name=*"sqlSessionFactory"* ref=*"sqlSessionFactoryBean"*/>  </bean> |

### 测试

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations={"classpath:applicationContext.xml"})  **public** **class** UserMapperTest {  @Autowired  **private** UserMapper mapper;  @Test  **public** **void** testFindUserById() {  System.***out***.println(mapper.findById(1).getUsername());  }  } |

### 用MapperScannerConfigurer批量扫描创建代理对象

【上面的代码麻烦，每一个mappler就创建一个工厂bean】

|  |
| --- |
| <!-- mapper代理开发方式之批量mapper配置 ,bean的名字默认为mapper接口类名的首字母小写  注意：  1.jdk1.8 用这种方式，bean不能创建成功 ，改成jdk1.7的即可  2.或者spring我换成spring3.2.9或以上就OK了  -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.neuedu.spring.mybatis.demo"*></property>  <property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactoryBean"*/>  </bean> |

测试与上一个测试一样

# 6.逆向工程

## 简介

**简单点说，就是通过数据库中的单表，自动生成java代码。**

Mybatis官方提供了逆向工程

可以针对单表自动生成mybatis代码（mapper.java\mapper.xml\po类）

企业开发中，逆向工程是个很常用的工具。

## 下载逆向工程

<https://github.com/mybatis/generator/releases/tag/mybatis-generator-1.3.2>

<dependency>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-maven-plugin</artifactId>  
 <version>1.4.0</version>  
</dependency>

## 使用方法

1. 创建简单的java项目
2. 导入jar包,创建generator配置文件；
3. 使用java类来执行逆向工程；
4. 把生成的代码拷贝到项目中。
5. 在正式项目中使用逆向工程生成的代码

### 第一步：创建generator配置文件

在classpath下，创建generator.xml配置文件：（文件内容可以从逆向工程的jar包中docs目录下的index.html中找到相关代码）

|  |
| --- |
| <?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=*"testTables"* targetRuntime=*"MyBatis3"*>  <commentGenerator>  <!-- 是否去除自动生成的注释 true：是 ： false:否 -->  <property name=*"suppressAllComments"* value=*"true"* />  </commentGenerator>  <!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->  <jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*  connectionURL=*"jdbc:mysql://localhost:3306/wsshop"* userId=*"root"*  password=*"root"*>  </jdbcConnection>    <javaTypeResolver>  <property name=*"forceBigDecimals"* value=*"false"* />  </javaTypeResolver>  <!-- targetProject:生成PO类的位置 -->  <javaModelGenerator targetPackage=*"com.neuedu.ws.pojo"*  targetProject=*".\src\main\java\"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  <!-- 从数据库返回的值被清理前后的空格 -->  <property name=*"trimStrings"* value=*"true"* />  </javaModelGenerator>  <!-- targetProject:mapper映射文件生成的位置 -->  <sqlMapGenerator targetPackage=*"com.neuedu.ws.mapper"*  targetProject=*".\src\main\resources\"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  </sqlMapGenerator>  <!-- targetPackage：mapper接口生成的位置 -->  <javaClientGenerator type=*"XMLMAPPER"*  targetPackage=*"com.neuedu.ws.mapper"*  targetProject=*".\src\main\java\"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  </javaClientGenerator>  <!-- 指定数据库表 -->  <table schema=*""* tableName=*"ws\_user"*></table>  </context>  </generatorConfiguration> |

### 第二步：使用java类来执行逆向工程

需要导入mysql的驱动包和mybatis的逆向工程包

|  |
| --- |
| public class Generator {  public static void main(String[] args) throws Exception{  List<String> warnings = new ArrayList<String>();  boolean overwrite = true;  File configFile = new File("config/generator.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);  }  } |

### 第三步：把生成的代码拷贝到项目中

如果正式项目中已经有po类所在的包了，那么就只需要拷贝po类到指定包下就可以。

如果正式项目中没有po包，那么就把逆向工程中整个po类的包拷贝过去。

Mapper.xml和mapper.java的拷贝与po类一样。

|  |
| --- |
|  |

### 第四步：测试

逆向工程提供了很多查询方法，可以不用写sql,这个根hibernate有点类似

|  |
| --- |
|  |