# Mybatis 学习

## 什么是MyBatis

MyBatis是支持普通SQL查询，存储过程和高级映射的优秀持久层框架。MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检 索。MyBatis使用简单的XML或注解用于配置和原始映射，将接口和Java的POJOs（Plan Old Java Objects，普通的Java对象）映射成数据库中的记录.

## 开发环境的搭建

### jar包

Myeclipse2014，jdk1.7，mysql5.7



### 数据库

创建项目Mybais，把mybatis-3.2.0.jar和mysql-connector-java-5.1.7-bin.jar放入lib目录下，创建数据库和用户表。

Create TABLE `user` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`userName` varchar(50) DEFAULT NULL,

`userAge` int(11) DEFAULT NULL,

`userAddress` varchar(200) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

Insert INTO `user` VALUES ('1', 'summer', '100', 'shanghai,pudong');

Create TABLE `article` (

`id` int(11) NOT NULL auto\_increment,

`userid` int(11) NOT NULL,

`title` varchar(100) NOT NULL,

`content` text NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;

-- ----------------------------

-- 添加几条测试数据

-- ----------------------------

Insert INTO `article` VALUES ('1', '1', 'test\_title', 'test\_content');

Insert INTO `article` VALUES ('2', '1', 'test\_title\_2', 'test\_content\_2');

Insert INTO `article` VALUES ('3', '1', 'test\_title\_3', 'test\_content\_3');

Insert INTO `article` VALUES ('4', '1', 'test\_title\_4', 'test\_content\_4');

### 工程

#### Java

##### User

**package** com.model;

**public** **class** User {

**private** String id;

**private** String userName;

**private** String userAge;

**private** String userAddress;

**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** String getUserAge() {

**return** userAge;

}

**public** **void** setUserAge(String userAge) {

**this**.userAge = userAge;

}

**public** String getUserAddress() {

**return** userAddress;

}

**public** **void** setUserAddress(String userAddress) {

**this**.userAddress = userAddress;

}

}

##### Article

**package** com.model;

**public** **class** Article {

**private** String id;

**private** User user;//外键直接定义定向，而不是字段

**private** String title;

**private** String content;

**public** String getId() {

**return** id;

}

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

**this**.id = id;

}

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

**public** String getContent() {

**return** content;

}

**public** **void** setContent(String content) {

**this**.content = content;

}

}

##### Test

**package** com.test;

**import** java.io.Reader;

**import** java.util.List;

**import** java.util.UUID;

**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** com.inter.UserInter;

**import** com.model.Article;

**import** com.model.User;

**public** **class** Test {

**private** **static** SqlSessionFactory sqlSessionFactory;

**Private static** Reader reader;

**static** {

**try**{

// 读取配置文件

reader = Resources.getResourceAsReader("Configuration.xml");

// 封装成session

sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

} **catch**(Exception e) {

e.printStackTrace();

}

}

**public** **static** SqlSessionFactory getSession() {

return sqlSessionFactory;

}

**public** **static void** main(String[] args) {

SqlSession session = sqlSessionFactory.openSession();

**try**{

// 执行已经映射的sql语句

// User user = (User)

// session.selectOne("com.model.UserMapper.selectUserById", "1");

// 接口实现

UserInter userInter = session.getMapper(UserInter.**class**);

// 查询指定用户

/\*

\* User user = userInter.findUserById("1");

\* System.out.println(user.getUserAddress());

\* System.out.println(user.getUserName());

\*/

// 根据名称模糊查询

/\*

\* List<User> userList = userInter.getUserListByName("%c%");

\* System.out.println(userList.size()); for (User user : userList) {

\* System.out.println("name:"+user.getUserName()); }

\*/

// 创建

/\*

\* User user = new User(); user.setId(UUID.randomUUID().toString());

\* user.setUserName("wy"); user.setUserAge("99");

\* user.setUserAddress("山西运城"); userInter.addUser(user);

\* session.commit(); System.out.println(user.getId());

\*/

// 修改

/\*

\* User user = new User();

\* user.setId("ccae2c09-6d76-42a3-a5a8-9af3fcb1f9c8");

\* user.setUserAddress("shanxi,yuncheng");

\* userInter.updateUser(user); session.commit();

\*/

// 删除

/\*

\* userInter.deleteUser("ccae2c09-6d76-42a3-a5a8-9af3fcb1f9c8");

\* session.commit();

\*/

// 级联查询

List<Article> articleList = userInter.getUserArticles("1");

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

**for** (Article article : articleList) {

System.*out*.println(article.getTitle() + "的作者是:"

+ article.getUser().getUserName());

}

} **finally** {

session.close();

}

}

}

#### Xml

##### User.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">

<!-- 原始的 -->

<!-- <mapper namespace="com.model.UserMapper"> -->

<!-- 接口实现，namespace必须是接口一致 -->

<mapper namespace=*"com.inter.UserInter"*>

<!-- sql语句，参数，返回类型 -->

<select id=*"findUserById"* parameterType=*"String"* resultType=*"User"*>

select \* from `user` where id = #{id}

</select>

<!-- 为了返回list 类型而定义的returnMap -->

<resultMap type=*"User"* id=*"resultListUser"*>

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

<result column=*"userName"* property=*"userName"* />

<result column=*"userAge"* property=*"userAge"* />

<result column=*"userAddress"* property=*"userAddress"* />

</resultMap>

<!-- 返回list 的select 语句，注意 resultMap 的值是指向前面定义好的 -->

<select id=*"getUserListByName"* parameterType=*"string"* resultMap=*"resultListUser"*>

select \* from user where userName like #{userName}

</select>

<!--执行增加操作的SQL语句。seGeneratedKeys设置为"true"表明要MyBatis获取由数据库自动生成的主键； keyProperty="id"指定把获取到的主键值注入

到User的id属性 -->

<insert id=*"addUser"* parameterType=*"User"* useGeneratedKeys=*"false"*

keyProperty=*"id"*>

insert into user(id,userName,userAge,userAddress)

values(#{id},#{userName},#{userAge},#{userAddress})

</insert>

<!-- 执行修改语句 -->

<update id=*"updateUser"* parameterType=*"User"*>

update user set

userAddress=#{userAddress}

where id=#{id}

</update>

<!-- 执行删除语句 -->

<delete id=*"deleteUser"* parameterType=*"String"*>

delete from user where

id=#{id}

</delete>

<!-- User 联合文章进行查询 方法之一的配置 (多对一的方式) -->

<resultMap id=*"resultUserArticleList"* type=*"Article"*>

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

<result property=*"title"* column=*"title"* />

<result property=*"content"* column=*"content"* />

<association property=*"user"* javaType=*"User"*>

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

<result property=*"userName"* column=*"userName"* />

<result property=*"userAddress"* column=*"userAddress"* />

</association>

</resultMap>

<!-- User 联合文章进行查询 方法之二的配置 (多对一的方式)

<resultMap id="resultUserArticleList-2" type="Article">

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

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

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

<association property="user" javaType="User" resultMap="resultListUser" />

</resultMap> -->

<select id=*"getUserArticles"* parameterType=*"int"*

resultMap=*"resultUserArticleList"*>

select user.id,user.userName,user.userAddress,article.id

aid,article.title,article.content from user,article

where

user.id=article.userid and user.id=#{id}

</select>

</mapper>

##### Configuration.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>

<typeAliases>

<!-- java类对应的别名 -->

<typeAlias alias=*"User"* type=*"com.model.User"* />

<typeAlias alias=*"Article"* type=*"com.model.Article"* />

</typeAliases>

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<!-- 数据库连接 -->

<dataSource type=*"POOLED"*>

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

<property name=*"url"* value=*"jdbc:mysql://127.0.0.1:3306/mybatis"* />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<!-- 映射的类配置文件 -->

<mapper resource=*"com/model/User.xml"* />

</mappers>

</configuration>

### 动态sql关键字

1. if 语句 (简单的条件判断)

<if test="title != null">

and title = #{title}

</if>

1. choose (when,otherwize) ,相当于java 语言中的 switch ,与 jstl 中的choose 很类似.  
   3. trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)  
   4. where (主要是用来简化sql语句中where条件判断的，能智能的处理 and or ,不必担心多余导致语法错误)  
   5. set (主要用于更新时)  
   6. foreach (在实现 mybatis in 语句查询时特别有用)

主要是if where foreach

### 与Spring MVC整合

1. web.xml 配置 spring dispatchservlet ,比如为:mvc-dispatcher  
   2. mvc-dispatcher-servlet.xml 文件配置  
   3. spring applicationContext.XML文件配置(与数据库相关，与mybatis sqlSessionFaction 整合，扫描所有mybatis mapper 文件等.)

##### web.xml

<context-param>

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

<param-value>classpath\*:config/applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<listener>

<listener-class>

org.springframework.web.context.ContextCleanupListener

</listener-class>

</listener>

<servlet>

<servlet-name>mvc-dispatcher</servlet-name>

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

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

</servlet>

<servlet-mapping>

<servlet-name>mvc-dispatcher</servlet-name>

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

</servlet-mapping>

##### mvc-dispatcher-Service.xml

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

<mvc:annotation-driven />

<mvc:resources mapping=*"/static/\*\*"* location=*"/WEB-INF/static/"* />

<mvc:default-servlet-handler />

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"*>

<value>/WEB-INF/pages/</value>

</property>

<property name=*"suffix"*>

<value>.jsp</value>

</property>

</bean>

##### applicationContext.xml

<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/mybatis?characterEncoding=utf8"* />

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

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

</bean>

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

<!--dataSource属性指定要用到的连接池 -->

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

<!--configLocation属性指定mybatis的核心配置文件 -->

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

<!-- 所有配置的mapper文件 -->

<!-- <property name="mapperLocations" value="classpath\*:com/model/\*.xml" /> -->

</bean>

<bean id=*"userInter"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>

<!--sqlSessionFactory属性指定要用到的SqlSessionFactory实例 -->

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

<!--mapperInterface属性指定映射器接口，用于实现此接口并生成映射器对象 -->

<property name=*"mapperInterface"* value=*"com.inter.UserInter"* />

</bean>