# 尚马教育 JAVA 课程

# Mybatis

文档编号：C01

创建日期： 2017-07-07

最后修改日期：2021-03-10

版 本 号：V3.6

电子版文件名：尚马教育-第三阶段-1.mybatis基础.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Mybatis版本更新 | V2.0 |
| 2019-08-09 | 冯勇涛 | 课件格式以及课程深度加深 | V3.0 |
| 2019-11-11 | 冯勇涛 | 格式调整以及多参数 | V3.1 |
| 2021-01-18 | 冯勇涛 | 课程顺序调整，细节 | V3.5 |
| 2021-03-10 | 冯勇涛 | Mysql连接url完整 | V3.6 |

目录

[尚马教育 JAVA 课程 1](#_Toc18)

[Mybatis 1](#_Toc26216)

[1. Mybaits介绍 4](#_Toc1491)

[1.1. 认识mybatis 4](#_Toc25339)

[1.2. jdbc与mybatis对比 5](#_Toc16695)

[2. 快速入门mybatis 7](#_Toc3480)

[2.1. 步骤 7](#_Toc23209)

[2.1.1. 添加Mybaits与数据库驱动包 7](#_Toc26242)

[2.1.2. 创建数据库表 8](#_Toc3267)

[2.1.3. 创建数据库表对应的实体类 11](#_Toc5069)

[2.1.4. 创建实体类对应的xml映射文件 11](#_Toc19390)

[映射文件中的注意事项： 12](#_Toc19829)

[2.1.5. 类路径下创建xml核心配置文件 12](#_Toc22942)

[2.1.6. mybatis的api调用 13](#_Toc5988)

[2.2. 重点 15](#_Toc13505)

[3. SqlSession对象的selectList方法 15](#_Toc8271)

[3.1. 修改映射文件 15](#_Toc31654)

[3.2. 添加测试方法 15](#_Toc5330)

[3.3. 重点 16](#_Toc30761)

[4. SqlSession对象的增删改方法 16](#_Toc28736)

[4.1. 修改映射文件 16](#_Toc21910)

[4.2. 添加测试方法 17](#_Toc24819)

[4.3. 重点 20](#_Toc24625)

[5. SqlSession对象的getMapper方法(重要) 20](#_Toc17445)

[5.1. 创建mapper接口 20](#_Toc25132)

[5.2. 修改sysuserMapper.xml 21](#_Toc7583)

[5.3. 添加测试方法 21](#_Toc4385)

[5.4. 重点 22](#_Toc20920)

[6. 小单元总结 22](#_Toc22667)

[6.1. MyBatis的核心对象 22](#_Toc23679)

[7. xml核心配置认识 23](#_Toc28880)

[7.1. properties标签 23](#_Toc22276)

[7.1.1. 在类路径下定义jdbc.properties文件 23](#_Toc28638)

[7.1.2. 引入properties资源文件 23](#_Toc25630)

[7.2. Settings标签 24](#_Toc10087)

[7.3. typeAliases标签 25](#_Toc10269)

[7.4. Environments标签 27](#_Toc1598)

[7.5. mappers标签 28](#_Toc19267)

[8. xml 映射文件认识 29](#_Toc7527)

[8.1. mapper元素 29](#_Toc17677)

[9. 多参数传递 30](#_Toc11990)

[9.1. 封装实体 30](#_Toc24191)

[9.2. 封装map 30](#_Toc6894)

[9.3. @Param注解 31](#_Toc3089)

[10. 复杂查询返回Map 31](#_Toc27460)

[10.1. Mysql创建sysrole表 31](#_Toc1026)

[10.2. 修改sysuser表的数据 32](#_Toc27702)

[10.3. 映射文件中添加select标签 32](#_Toc23558)

[10.4. mapper接口添加方法 32](#_Toc5187)

[11. 模糊查询 32](#_Toc4530)

[12. 总结 33](#_Toc12126)

## Mybaits介绍

### 认识mybatis

* MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github，通俗说法Ibatis3 = MyBatis
* iBATIS一词来源于“internet”和“abatis”的组合，是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO）
* MyBatis是一个数据持久层(ORM)框架。把实体类和SQL语句之间建立了映射关系，是一种半自动化的ORM实现
* MyBatis的优点：
  + 减少代码量
  + 基于SQL语法，简单易学
  + 能了解底层组装过程
  + SQL语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度
  + 程序调试方便



### jdbc与mybatis对比

* Jdbc代码：

|  |
| --- |
| Class.forName(“com.mysql.jdbc.Driver”);  Connection conn =DriverManager.getConnection(“url”,”username”,”password”);  String sql = “select \* from m\_user where userId=?”;  PrepareStatement pst = conn.prepareStatement(sql);  Pst.setInt(1,1);  ResultSet rs = pst.executeQuery();  while(rs.next()){  System.out.println(rs.getString(1));  } |

* Mybatis代码：

|  |
| --- |
| <select id="getUser" parameterType="int" resultType="com.javasm.entity.UserInfo">  select \* from m\_user where id=#{id}  </select> |

## 快速入门mybatis

### 步骤

|  |
| --- |
| 1. 创建java工程 2. 添加mybatis与mysql数据库驱动包。 3. 在mysql数据库下创建数据库表。 4. 在entity包下创建数据库表对应的实体类。 5. 在mapper包下创建实体类对应的xml映射文件。 6. 在类路径下创建mybatis的xml核心配置文件,注意引入映射文件。 7. 使用mybatis的核心api对象加载配置文件。 |

#### 具体细节步骤如下：

#### 添加Mybaits与数据库驱动包

* 创建java工程，并加入mybatis的jar包以及数据库驱动包。

|  |
| --- |
|  |

#### 创建数据库表

系统用户表DDL语句：

|  |
| --- |
| CREATE TABLE `sysuser` (  `id` int(11) NOT NULL,  `uname` varchar(30) NOT NULL COMMENT '用户名',  `upwd` varchar(20) NOT NULL COMMENT '密码',  `uphone` char(11) NOT NULL COMMENT '手机号',  `uimg` varchar(50) DEFAULT NULL COMMENT '头像',  `uwechat` varchar(20) DEFAULT NULL COMMENT '微信号',  `uemail` varchar(20) DEFAULT NULL COMMENT '邮箱',  `ubirthday` date DEFAULT NULL COMMENT '生日',  `deleted` char(1) DEFAULT '0' COMMENT '是否删除',  `create\_time` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,  `update\_time` timestamp NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='系统用户表'; |

系统用户表设计图：

|  |
| --- |
|  |

#### 创建数据库表对应的实体类

|  |
| --- |
| **package** com.javasm.sys.entity;  **public class** Sysuser {  **private** Integer **id**;  **private** String **uname**;  **private** String **upwd**;  **private** String **uphone**;  **private** String **uimg**;  **private** String **uwechat**;  **private** String **uemail**;  **private** String **ubirthday**;  **private** String **deleted**;  **private** String **createTime**;  **private** String **updateTime**;  *//****TODO 生成 setter,getter,toString*** } |

#### 创建实体类对应的xml映射文件

创建package包：com.javasm.sys.mapper，在该包下创建映射文件sysuserMapper.xml。

|  |
| --- |
| <**mapper namespace="userMapper"**>  <**select id="getUserById" parameterType="java.lang.Integer" resultType="com.javasm.sys.entity.Sysuser"**>  select *\** from sysuser where id=#{id}  </**select**> </**mapper**> |

**映射文件中的注意事项：**

* 注意点1：namespace:表示当前映射文件的唯一命名空间,确保不同映射文件之间的namespace不要重复。
* 注意点2：进行查询的sql语句，标签名必须是select标签。
* 注意点3：select标签的id，在当前映射文件中必须唯一。
* 注意点4：parameterType表示查询参数类型；resultType表示数据库查询结果的每行记录封装类型。
* 注意点5：#{}括号中的内容在当前查询参数是int时，可以随意定义。

#### 类路径下创建xml核心配置文件

* 在项目src目录中创建mybatis-config.xml文件
* 配置当前数据库的数据源：

|  |
| --- |
| *<?***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**>  *<!--配置数据库环境，支持多数据库环境-->* <**environments default="dev"**>  <**environment id="dev"**>  *<!--事务管理器,JDBC表示启用JDBC事务管理-->* <**transactionManager type="JDBC"**></**transactionManager**>  *<!--连接池：POOLED表示初始化PooledDataSource连接池对象-->* <**dataSource type="POOLED"**>  <**property name="driver" value="com.mysql.jdbc.Driver"**/>  <**property name="url" value="jdbc:mysql://127.0.0.1:3306/720A?useUnicode=true&characterEncoding=utf8&useSSL=true&serverTimezone=UTC"**/>  <**property name="username" value="root"**/>  <**property name="password" value="root"**/>  </**dataSource**>  </**environment**>  </**environments**>  *<!--引入xml映射文件-->*  <**mappers**>  <**mapper resource="com/javasm/sys/mapper/sysuserMapper.xml"**></**mapper**>  </**mappers**> </**configuration**> |

#### mybatis的api调用

1. 创建SqlSessionFactoryUtil工具类

|  |
| --- |
| **public class** SqlSessionFactoryUtil {  **private static** SqlSessionFactory *ssf*=**null**;  **static**{  InputStream input = **null**;  **try** {  input = Resources.*getResourceAsStream*(**"mybatis-config.xml"**);  } **catch** (IOException e) {  e.printStackTrace();  }  *ssf* = **new** SqlSessionFactoryBuilder().build(input);  }  **public static** SqlSessionFactory getSqlSessionFactory(){  **return** *ssf*;  } } |

1. 创建junit测试类

|  |
| --- |
| **public class** TestStart {  **private static** SqlSessionFactory *ssf* = **null**;  **private** SqlSession **session** = **null**;  @BeforeClass  **public static void** initFactory() {  *ssf* = SqlSessionFactoryUtil.*getSqlSessionFactory*();  }  @Before  **public void** openSession() {**session** = *ssf*.openSession();  }  @Test  **public void** testGetUserById() **throws** IOException {  Object result = **session**.selectOne(**"userMapper.getUserById"**, 1);  System.***out***.println(result);  }  @After  **public void** closeSession() {  **session**.close();  }  @AfterClass  **public static void** closeFactory() {  *ssf* = **null**;  } } |

### 重点

* SqlSessionFactoryBuilder构建器对象生命周期很短暂，加载xml完成后即销毁。
* SqlSessionFactory会话工厂对象生命周期全局唯一。
* SqlSession会话对象生命周期很短暂，需要操作数据库则打开，操作完成后必须关闭释放连接。
* SqlSession的selectOne方法的第一个参数是namespace.id组成的字符串，第二个参数是查询参数，且当前只能传入一个参数。
* 映射文件的sql语句动态参数使用#{}表达式获取，查询参数是String，int，double等简单类型时，括号中内**容随意定义，一般写为数据库字段名**。

## SqlSession对象的selectList方法

### 修改映射文件

在sysuserMapper.xml映射文件中添加新的select标签。

|  |
| --- |
| <**select id="selectUsers" resultType="com.javasm.sys.entity.Sysuser"**>  select *\** from sysuser order by update\_time </**select**> |

### 添加测试方法

在junit测试类中添加新的测试方法

|  |
| --- |
| @Test **public void** testSelectUsers() **throws** IOException {  List<Object> objects = **session**.selectList(**"userMapper.selectUsers"**);  System.***out***.println(objects); } |

### 重点

注意点1：当selectUsers查询无参数时，parameterType可以省略(其实一直可以省略).

注意点2：selectUsers下的sql语句查询出列表数据，但resultType仍然是实体类，绝对不会 出现list，切记resultType表示查询结果**单行记录**封装类型。

## SqlSession对象的增删改方法

### 修改映射文件

|  |
| --- |
| <**insert id="addUser" parameterType="com.javasm.sys.entity.Sysuser" useGeneratedKeys="true" keyProperty="id"**>  insert into sysuser(uname, upwd, uphone, uemail) values (#{uname},#{upwd},#{uphone},#{uemail}) </**insert**>  <**update id="updateUser" parameterType="com.javasm.sys.entity.Sysuser"**>  update sysuser set uname=#{uname},upwd=#{upwd},uphone=#{uphone},uemail=#{uemail} where id=#{id} </**update**>  <**delete id="delUserByKey" parameterType="java.lang.Integer"**>  delete from sysuser where id=#{id} </**delete**> |

注意点：在addUser与**updateUser**标签内，传入的参数是Sysuser实体类，则#{}括号的内容不能随意定义，必须是**实体类的属性名**。

### 添加测试方法

在junit测试类中添加新的测试方法

|  |
| --- |
| @Test **public void** test1\_addUser(){  Sysuser suser = **new** Sysuser();  suser.setUname(**"尚马学生李某"**);  suser.setUemail(**"liXxx@163.com"**);  suser.setUpwd(**"123456"**);  suser.setUphone(**"15516122134"**);  **int** rows= **session**.insert(**"userMapper.addUser"**,suser);  System.***out***.println(**"受影响的行数："**+rows);  System.***out***.println(**"获取自增id:"**+suser.getId()); }  @Test **public void** test3\_updateUser(){  **int** id = 1;  Sysuser suser = **new** Sysuser();  suser.setId(id);  suser.setUname(**"尚马学生李某人"**);  suser.setUemail(**"liXxx@163.com"**);  suser.setUpwd(**"123456"**);  suser.setUphone(**"15516122134"**);  **int** rows = **session**.update(**"userMapper.updateUser"**,suser);  System.***out***.println(**"结果："**+rows); }  @Test **public void** test4\_delUserByKey(){  **int** rows = **session**.delete(**"userMapper.delUserByKey"**,1);  System.***out***.println(**"结果："**+rows); } |

### 重点

重点1：insert，update，delete标签对应的是数据库增删改操作，不能混用。

重点2：insert添加语句，如果主键id是自增，则可以使用useGeneratedKeys="true" keyProperty="id"得到数据库生成的自增主键值，keyProperty的值是**实体类的id属性名**。

重点3：当parameterType的值是实体类自定义对象时，#{}内容必须是实体类属性名。

重点4：insert，delete，update标签没有resultType返回值。默认返回int受影响行数。

## SqlSession对象的getMapper方法(重要)

### 创建mapper接口

|  |
| --- |
| **package** com.javasm.sys.mapper;  **import** com.javasm.sys.entity.Sysuser; **import** java.util.List;  **public interface** SysuserMapper {  **public** Sysuser getUserById(Integer uid); } |

### 修改sysuserMapper.xml

修改mapper标签的namespace为mapper接口名称

|  |
| --- |
| *<?***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.javasm.sys.mapper.SysuserMapper"**>  <**select id="getUserById" parameterType="java.lang.Integer" resultType="com.javasm.sys.entity.Sysuser"** >  select *\** from sysuser where id=#{id}  </**select**> </**mapper**> |

### 添加测试方法

在以上junit测试类中加入新的测试方法。

|  |
| --- |
| @Test **public void** testGetUserByKey() **throws** IOException {  SysuserMapper mapper = **session**.getMapper(SysuserMapper.**class**);  Sysuser sysuser = mapper.getUserById(1);  System.***out***.println(sysuser); } |

### 重点

重点1：xml映射文件中的namespace不再随意，必须是mapper接口的名称（含包名）。

注意点2：调用SqlSession会话对象方法时，通过getMapper方法返回mapper接口类型的对象是当前接口的代理对象，代理对象的细节后续分析。

## 小单元总结

### MyBatis的核心对象

1. SqlSessionFactoryBuilder：

SqlSessionFactoryBuilder用过即丢，其生周期只存在于方法体内，可重用。 SqlSessionFactoryBuilder来创建多个SqlSessionFactory 实例。

1. SqlSessionFactory：

SqlSessionFactory是每个MyBatis应用的核心；

单例，存在于整合应用运行时，并且同时只存在一个对象实例；

1. SqlSession

包含了执行sql所需的所有方法，可以通过SqlSession实例直接运行映射的sql语句；

SqlSession对应着一次数据库会话，它的生命周期不是永久的，在每次访问数据库时都 需要创建它每个线程都有自己的 SqlSession 实例，SqlSession实例是不能被共享，也不 是线程安全的。

调用insert,update,selectList,selectOne,delete等方法执行增、删、查、改等操作；

调用getMapper(xxx.class)来实例化接口执行增删查改；

注意：并不是说在一个SqlSession会话中只能执行一次Sql，可以执行多次，当关闭了SqlSession就需要重新创建。同时关闭是非常重要，必须确保正常关闭。

## xml核心配置认识

以上配置文件中配置了数据库连接信息，引入了映射文件，该文件中可以有其他标签。详细如下：

### properties标签

#### 在类路径下定义jdbc.properties文件

|  |
| --- |
| jdbc.user = root  jdbc.password = root  jdbc.url = jdbc:mysql://127.0.0.1:3306/704a?characterEncoding=utf8&useSSL=true  jdbc.driver = com.mysql.jdbc.Driver |

#### 引入properties资源文件

在xml核心配置文件中通过properties标签引入jdbc.properties

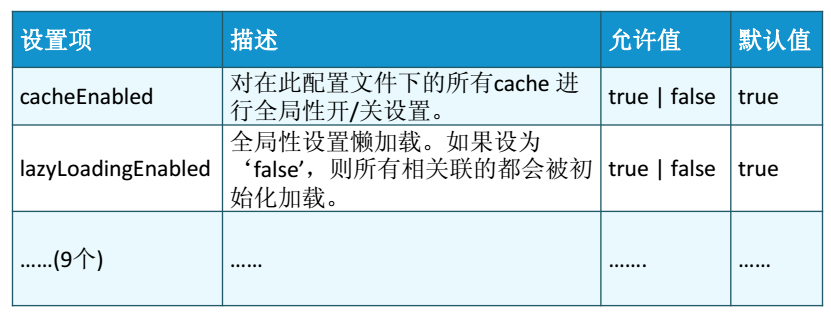
|  |
| --- |
| <properties resource="jdbc.properties"></properties> |

修改原有的配置信息,采用变量引入式声明

|  |
| --- |
| <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> |

### Settings标签

用来修改MyBatis在运行时的行为方式，主要是Mybatis的一些全局配置属性的设置。



|  |
| --- |
| <**settings**>  *<!--把数据库中的\_分割的列名，映射到实体类的驼峰属性名 -->* <**setting name="mapUnderscoreToCamelCase" value="true"**/>  *<!--mybatis的运行日志输出位置-->* <**setting name="logImpl" value="STDOUT\_LOGGING"**></**setting**> </**settings**> |

### typeAliases标签

类型别名是Java 类型的简称,它仅仅只是关联到XML 配置，简写冗长的JAVA 类名。同时没有mybatis对常用简单类型内建了别名，比如java.lang.Integer-int,java.util.String-String

|  |
| --- |
| <typeAliases>  <typeAlias alias="User" type="com.javasm.entity.User"/>  </typeAliases> |
| <**typeAliases**>  <**package name="com.javasm"**></**package**> </**typeAliases**> |

加入typeAlias配置后，映射文件中出现的类名可以写别名：

|  |
| --- |
| <**select id="getUserById" parameterType="int" resultType="Sysuser"** >  select *\** from sysuser where uid=#{aa} </**select**> |

Mybatis内嵌别名

|  |  |
| --- | --- |
| 别名 | **映射的类型** |
| string | String |
| long | Long |
| Integer | Integer |
| Int | Integer |
| double | Double |
| List | ArrayList |
| .... | .... |

注意点1：要求不同包下类不能同名，类名做别名，忽略大小写。

注意点2：以上两个typeAlias一般选择配置包名。

注意点3：别名影响的是映射文件中的parameterType与resultType属性。

### Environments标签

MyBatis 可以配置多套运行环境，将SQL映射到多个数据库上。

虽然可以配置多个运行环境，但是每个SqlSessionFactory 实例只能选择一个运行环境。即：每个数据库对应一个SqlSessionFactory 实例。



dataSource

dataSource元素使用基本的JDBC数据源接口来配置JDBC连接对象的资源。有三种内建的数据源类型 <dataSource type=“[UNPOOLED|POOLED|JNDI]"/>

UNPOOLED 每次创建新的连接

POOLED 创建连接池,用完归还

JNDI 获取JNDI连接池

### mappers标签

Sql映射语句一般定义在各持久类的Mapper.xml文件中，需要在配置中引用这些映射文件

|  |
| --- |
| <!-- 将mapper文件加入到配置文件中 -->  <mappers>  <mapper resource="com/javasm/dao/UserMapper.xml"/>  </mappers> |

## xml 映射文件认识

### mapper元素

Mapper元素的namespace属性一般都与对应的dao接口全名称保持一致；

|  |
| --- |
| <mapper namespace="com.javasm.dao.UserInfoDao"></mapper > |

### select子元素

表示数据库查询，注意属性：id，parameterType，resultType

* Id与mapper接口中方法名保持一致；
* parameterType与mapper接口中方法形参类型一致,一般是String,Map,实体类,Integer,Double等；
* resultType与mapper接口中方法返回值类型一致,resultType表示mybatis对查询结果的单行记录封装类型,一般写为实体类,Map,String,Integer,Double等简单类型；

|  |
| --- |
| <select id="getUser" parameterType="int" resultType="Sysuser">  select \* from sysuser where id=#{id}  </select> |

### Update子元素

update,insert,delete子元素都没有resultType属性,默认返回受影响的行数.

|  |
| --- |
| <**update id="updateUser" parameterType="Sysuser"**>  update sysuser set **uname**=#{uname},**uphone**=#{uphone},**uemail**=#{uemail} where **uid**=#{uid} </**update**> |
| **public int** updateUser(Sysuser user); |

Dao接口的方法返回值默认是int

### Insert子元素

|  |
| --- |
| <**insert id="addUser" parameterType="Sysuser" useGeneratedKeys="true" keyProperty="uid"**>  insert into sysuser(**uname**, **upwd**, **uphone**, **uemail**) values (#{uname},#{upwd},#{uphone},#{uemail}) </**insert**> |
| **public int** addUser(Sysuser user); |

Dao接口的方法返回值默认是int

### Delete子元素

|  |
| --- |
| <**delete id="delUser" parameterType="int"**>  delete from sysuser where **uid**=#{uid} </**delete**> |
| **public int** delUser(Integer uid); |

### #{}与${}表达式

两者都是用来获取查询参数,用于sql语句中.

#### #{}表达式

mybatis内对#{}解析为?参数占位符,更加安全,因此对于条件查询参数必须使用#{}来获取数据.避免sql注入攻击.

1. 如果parameterType是简单类型的话,#{随便写}
2. 如果parameterType是实体类,#{类的成员变量名}
3. 如果parameterType是map,#{map的key}

#### ${}表达式

mybatis内对${}不解析?占位符,直接进行sql拼接,因此不适合做条件查询.如果有动态指定字段排序的场景,可以使用${}来获取排序字段以及排序方式.如下:

|  |
| --- |
| <**select id="selectUsers" parameterType="map" resultType="sysuser"**>  select *\** from sysuser order by ${soreField} ${sorted} </**select**> |
| **public** List<Sysuser> selectUsers(Map<String,Object> map); |
| @Test **public void** test2\_selectUsers() {  Map<String,Object> args = **new** HashMap<>();  args.put(**"soreField"**,**"update\_time"**);  args.put(**"sorted"**,**"asc"**);SysuserDao ud = **session**.getMapper(SysuserDao.**class**);  List<Sysuser> sysusers = ud.selectUsers(args);  System.***out***.println(sysusers); } |

## 多参数传递

mybatis默认api只能够接收一个动态参数，因此当需要传入多个数据变量时，需要把多个变量封装到一个对象中，比如实体对象或Map

### 封装实体

|  |
| --- |
| **Dao接口：**  **public** Sysuser selectUser(Sysuser u); |
| Xml映射文件：  <select id=*"selectUser"* parameterType=*"sysuser"* resultType=*"sysuser"*>  select \* from sysuser where uname=#{uname} and uphone=#{uphone}  </select> |
| 测试类：  ISysuserDao ud = session.getMapper(ISysuserDao.**class**);  Sysuser u = **new** Sysuser();  u.setUname("aaa");  u.setUphone("123123123");  Sysuser selectUser = ud.selectUser(u); |

注意点：传入实体类时，#{}内容为实体类属性名

### 封装map

|  |
| --- |
| **Mapper接口：**  **public** Sysuser selectUser2(Map<String,Object> m); |
| Xml映射文件：  <!-- #{map的key} -->  <select id=*"selectUser2"* parameterType=*"map"* resultType=*"sysuser"*>  select \* from sysuser where uname=#{unameKey} and uphone=#{uphoneKey}  </select> |
| 测试类：  ISysuserDao ud = session.getMapper(ISysuserDao.**class**);  Map<String,Object> u = **new** HashMap<>();  u.put("unameKey", "aaa");  u.put("uphoneKey", "1231231232");Sysuser selectUser = ud.selectUser2(u); |

注意点：传入map时，#{}内容为map的key值

### @Param注解

接口的方法形参保持多参数,由mybatis底层把多参数封装到Map对象中,通过@Param注解指定key.

|  |
| --- |
| **mapper接口：**  **public** Sysuser selectUserByNameAndPhone(@Param("un")String uname,@Param("up")String uphone); |
| Xml映射文件：  <select id=*"selectUserByNameAndPhone"* resultType=*"Sysuser"*>  select \* from sysuser where uname=#{un} and uphone=#{up}  </select> |
| 测试类：  ISysuserDao ud = session.getMapper(ISysuserDao.**class**);  Sysuser u = ud.selectUserByNameAndPhone("aaa", "123123123"); |

注意点：使用@Param注解形参时，#{}内容是Param注解的value属性值。

## 复杂查询返回Map

如果进行多表连接查询，查询结果列无法直接封装到实体类，可以封装到一个Map集合中。

### Mysql创建sysrole表

|  |
| --- |
| CREATE TABLE `sysrole` (  `id` int(11) NOT NULL,  `rname` varchar(255) DEFAULT NULL COMMENT '角色名',  `rdesc` varchar(255) DEFAULT NULL COMMENT '角色描述',  `deleted` char(1) DEFAULT NULL COMMENT '是否删除',  `create\_time` timestamp NULL DEFAULT CURRENT\_TIMESTAMP,  `update\_time` timestamp NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8; |

手动添加测试数据(人事部经理，人事部职员角色)

### 修改sysuser表的数据



### 映射文件中添加select标签

|  |
| --- |
| <**select id="selectUserAndRoleByUID" parameterType="int" resultType="map"**>  select u.*\**,r.rname from sysuser u,sysrole r where u.roleid=r.id and u.id=#{id} </**select**> |

注意：resultType属性是map，不再是一个实体类。

### mapper接口添加方法

|  |
| --- |
| **public** Map<String,Object> selectUserAndRoleByUID(Integer uid); |

## 模糊查询

|  |
| --- |
| <select id="selectUsers" parameterType="Sysuser" resultType="Sysuser">  select \* from sysuser where uname like "%"#{uname}"%"  </select> |

注意：模糊查询的%使用双引号括起来即可。

## 总结

MyBatis是一个数据持久层(ORM)框架。把实体类和SQL语句之间建立了映射关系，是一种半自动化的ORM实现，MyBatis的优点：基于SQL语法，简单易学，能了解底层组装过程，SQL语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度，程序调试方便。

MyBatis环境搭建，实现对数据表简单的查询操作。

Mybatis的getMapper方法原理，动态代理模式。

MyBatis的核心对象：SqlSessionFactoryBuilder，SqlSessionFactory，SqlSession

mybatis-config.xml 全局配置文件常用标签。

Mybatis映射文件常用标签。