# Mybatis第一天课堂笔记

学习过的持久层框架：DBUtils , Hibernate

Mybatis就是类似于hibernate的orm持久层框架。

**为什么学Mybatis?**

1. 目前最主流的持久层框架为hibernate与mybatis，而且国内目前情况使用Mybatis的公司比hibernate要多。
2. Hibernate学习门槛不低，要精通门槛更高。门槛高在怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用好Hibernate缓存与数据加载策略方面需要你的经验和能力都很强才行。国内目前前的情况精通hibernate技术大牛非常少。
3. sql优化方面，Hibernate的查询会将表中的所有字段查询出来，这一点会有性能消耗。当然了，Hibernate也可以自己写SQL来指定需要查询的字段，但这样就破坏了Hibernate开发的简洁性。说得更深入一些，如果有个查询要关联多张表，比如5张表，10张表时，而且，我们要取的字段只是其中几张表的部分字段。这时用hibernate时就会显得非常力不从心。就算用hibernate的sqlquery，后续的维护工作也会让人发狂。

# JDBC编程回顾与存在的问题分析

## 开发步骤：

1. 导入数据脚本，在课前资料中有
2. 创建工程，导入mysql jar包
3. 编码

## Jdbc访问数据库的过程：

1. //加载数据库驱动
2. //创建数据库连接
3. //创建statement
4. //设置sql语句
5. //设置查询参数
6. //执行查询，得到ResultSet
7. //解析结果集ResultSet
8. //释放资源

## Jdbc存在的问题：

1. 频繁创建和打开、关闭数据连接，太消耗资源
2. Sql语句存在硬编码，不利于维护
3. Sql参数设置硬编码，不利于维护
4. 结果集获取与遍历复杂，存在硬编码，不利于维护，期望能够查询后返回一个java对象

# Mybatis介绍

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。

Mybatis是面向sql的持久层框架，他封装了jdbc访问数据库的过程，我们开发，只需专注于sql语句本身的拼装，其它复杂的过程全部可以交给mybatis去完成。

# Mybaits入门

## 需求列表

根据用户ID查询用户信息

根据用户名查找用户列表

添加用户

修改用户

删除用户

## 工程搭建

1. 导入依赖jar包，在课前资源中有
2. 配置SqlMapConfig.xml，可参考课前资料
3. 配置log4j.properties，可参考课前资料
4. 在课前资料复制pojo到工程目录下
5. 配置sql查询的映射文件，可参考课前资料
6. 加载映射文件

## 完成需求

### 需求完成步骤

1. 编写sql语句
2. 配置user映射文件
3. 编写测试程序

### 根据用户ID查询用户信息

#### 映射文件与sql

<!-- id:statementId

resultType:查询结果集的数据类型

parameterType:查询的入参

-->

<select id=*"getUserById"* parameterType=*"int"* resultType=*"com.itheima.mybatis.pojo.User"* >

SELECT \* FROM USER WHERE id = #{id1}

</select>

#### MyBatis访问数据库代码

@Test

**public** **void** testGetUserByid() **throws** IOException {

// 创建SqlSessionFactoryBuilder对象

SqlSessionFactoryBuilder sfb = **new** SqlSessionFactoryBuilder();

// 查找配置文件创建输入流

InputStream inputStream = Resources.*getResourceAsStream*("SqlMapConfig.xml");

// 加载配置文件，创建SqlSessionFactory对象

SqlSessionFactory sqlSessionFactory = sfb.build(inputStream);

// 创建SqlSession对象

SqlSession sqlSession = sqlSessionFactory.openSession();

// 执行查询，参数一：要查询的statementId ,参数二：sql语句入参

User user = sqlSession.selectOne("user.getUserById", 1);

// 输出查询结果

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

// 释放资源

sqlSession.close();

}

### 抽取SqlSessionFactoryUtils工具类，共享SqlSessionFactory创建过程

/\*\*

\* SqlSessionFactory工具类

\*

\* **@author** Steven

\*

\*/

**public** **class** SqlSessionFactoryUtils {

/\*\*

\* 单例SqlSessionFactory

\*/

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

**static** {

// 创建SqlSessionFactoryBuilder对象

SqlSessionFactoryBuilder sfb = **new** SqlSessionFactoryBuilder();

**try** {

// 查找配置文件创建输入流

InputStream inputStream = Resources.*getResourceAsStream*("SqlMapConfig.xml");

// 加载配置文件，创建SqlSessionFactory对象

*sqlSessionFactory* = sfb.build(inputStream);

} **catch** (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 获取单例SqlSessionFactory

\* **@return**

\*/

**public** **static** SqlSessionFactory getSqlSessionFactory() {

**return** *sqlSessionFactory*;

}

}

### 根据用户名查找用户列表

#### 映射文件与sql

<!-- resultType:如果要返回数据集合，只需设定为每一个元素的数据类型 -->

<select id=*"getUserByName"* parameterType=*"string"* resultType=*"com.itheima.mybatis.pojo.User"*>

<!-- SELECT \* FROM USER WHERE username LIKE #{name} -->

SELECT \* FROM USER WHERE username LIKE '%${value}%'

</select>

#### MyBatis访问数据库代码

@Test

**public** **void** getUserByName() {

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

//List<User> users = sqlSession.selectList("user.getUserByName", "%张%");

List<User> users = sqlSession.selectList("user.getUserByName", "张");

**for** (User user : users) {

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

}

// 释放资源

sqlSession.close();

}

### 添加用户

#### 映射文件与sql

<insert id=*"insertUser"* parameterType=*"com.itheima.mybatis.pojo.User"*>

INSERT INTO USER

(`username`,

`birthday`,

`sex`,

`address`)

VALUES (#{username},

#{birthday},

#{sex},

#{address});

</insert>

#### MyBatis访问数据库代码

@Test

**public** **void** testInsertUser() **throws** IOException {

// 创建SqlSession对象

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

User user = **new** User();

user.setUsername("张飞");

user.setAddress("深圳市黑马");

user.setBirthday(**new** Date());

user.setSex("1");

// 执行插入

sqlSession.insert("user.insertUser", user);

// 提交事务

sqlSession.commit();

// 释放资源

sqlSession.close();

}

#### Mysql自增返回

<!-- useGeneratedKeys:标识插入使用自增id

keyProperty:与useGeneratedKeys配套使用，用于绑定主键接收的pojo属性

-->

<insert id=*"insertUserKey"* parameterType=*"com.itheima.mybatis.pojo.User"* useGeneratedKeys=*"true"* keyProperty=*"id"*>

<!-- selectKey:用于配置主键返回

keyProperty：要绑定的pojo属性

resultType:属性数据类型

order:指定什么时候执行，AFTER之后

-->

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

SELECT LAST\_INSERT\_ID()

</selectKey> -->

INSERT INTO USER

(`username`,

`birthday`,

`sex`,

`address`,

`uuid2`)

VALUES (#{username},

#{birthday},

#{sex},

#{address},

#{uuid2});

</insert>

#### Mysql的uuid返回主键

**注：在使用uuid之前数据库user表要先加上uuid2字段、user的pojo也要加上相应属性**

<!-- useGeneratedKeys:标识插入使用自增id

keyProperty:与useGeneratedKeys配套使用，用于绑定主键接收的pojo属性

-->

<insert id=*"insertUserUUID"* parameterType=*"com.itheima.mybatis.pojo.User"*>

<!-- selectKey:用于配置主键返回

keyProperty：要绑定的pojo属性

resultType:属性数据类型

order:指定什么时候执行，*BEFORE*之前

-->

<selectKey keyProperty=*"uuid2"* resultType=*"string"* order=*"BEFORE"*>

SELECT UUID()

</selectKey>

INSERT INTO USER

(`username`,

`birthday`,

`sex`,

`address`,

`uuid2`)

VALUES (#{username},

#{birthday},

#{sex},

#{address},

#{uuid2});

</insert>

### 修改用户

<update id=*"updateUser"* parameterType=*"com.itheima.mybatis.pojo.User"*>

UPDATE USER SET username = #{username} WHERE id = #{id}

</update>

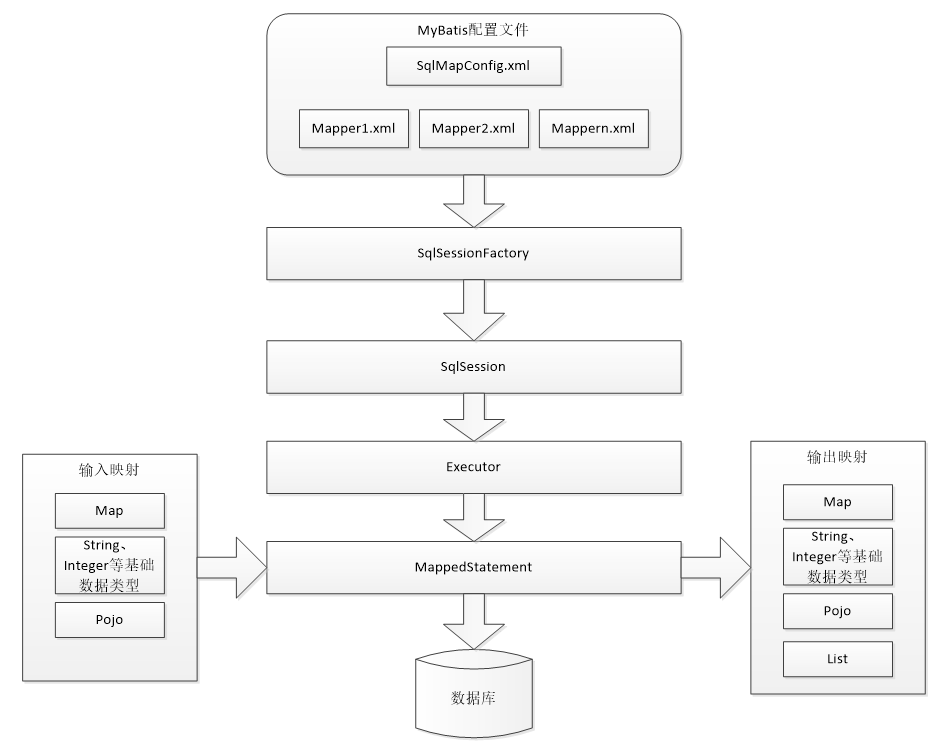
### 删除用户

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

DELETE FROM `user` WHERE `id` = #{id1}

</delete>

## Mybatis入门小结与Mybatis架构图



# Mybatis Dao开发方式

## Dao需求

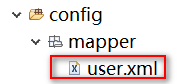
根据用户ID查询用户信息

根据用户名查找用户列表

添加用户

## 原始Dao开发方法

### 使用原有的user映射文件，不需修改



### 新建个UserDao接口

**public** **interface** UserDao {

/\*\*根据用户ID查询用户信息

\* **@param** id

\* **@return**

\*/

User getUserById(Integer id);

/\*\*

\* 根据用户名查找用户列表

\* **@param** name

\* **@return**

\*/

List<User> getUserByUserName(String name);

/\*\*

\* 添加用户

\* **@param** user

\*/

**void** insertUser(User user);

}

### 新建个UserDaoImpl接口实现类

**public** **class** UserDaoImpl **implements** UserDao {

@Override

**public** User getUserById(Integer id) {

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

User user = sqlSession.selectOne("user.getUserById", id);

sqlSession.close();

**return** user;

}

@Override

**public** List<User> getUserByUserName(String name) {

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

List<User> list = sqlSession.selectList("user.getUserByName", name);

sqlSession.close();

**return** list;

}

@Override

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

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

sqlSession.insert("user.insertUser", user);

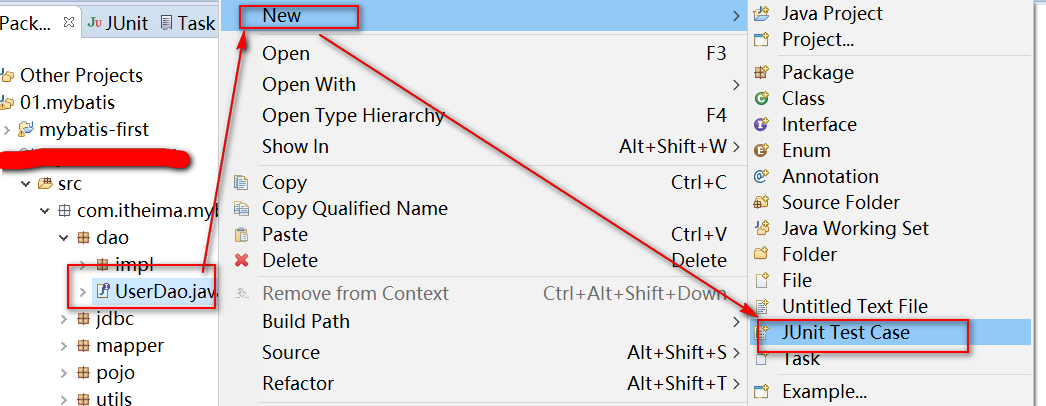
sqlSession.commit();

sqlSession.close();

}

}

### 使用dao测试



**public** **class** UserDaoTest {

@Test

**public** **void** testGetUserById() {

UserDao userDao = **new** UserDaoImpl();

User user = userDao.getUserById(30);

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

}

@Test

**public** **void** testGetUserByUserName() {

UserDao userDao = **new** UserDaoImpl();

List<User> list = userDao.getUserByUserName("张");

**for** (User user : list) {

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

}

}

@Test

**public** **void** testInsertUser() {

UserDao userDao = **new** UserDaoImpl();

User user = **new** User();

user.setUsername("张飞3");

user.setAddress("深圳市黑马");

user.setBirthday(**new** Date());

user.setSex("1");

userDao.insertUser(user);

}

}

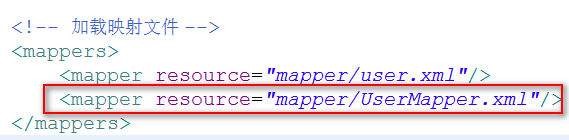
## 官方推荐，接口动态代理

### 动态代理dao开发规则

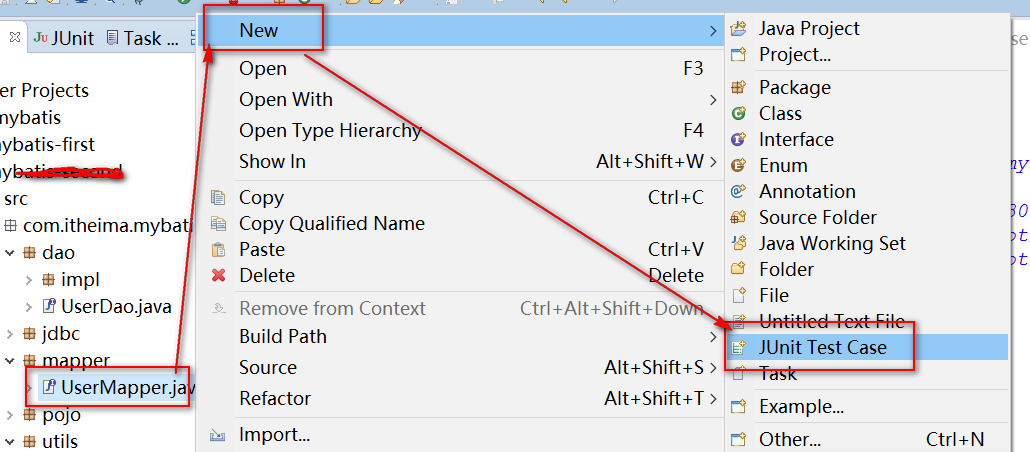
1. namespace必需是接口的全路径名
2. 接口的方法名必需与映射文件的sql id一致
3. 接口的输入参数必需与映射文件的parameterType类型一致
4. 接口的返回类型必须与映射文件的resultType类型一致

### 动态代理dao开发步骤

1. 创建UserMapper.xml映射文件(把原来的user.xml复制按开发规则要求修改一下)
2. 创建UserMapper接口(把原来的UserDao.java复制按开发规则要求修改一下)
3. 加载UserMapper.xml



1. 测试动态代理Dao



**public** **class** UserMapperTest {

@Test

**public** **void** testGetUserById() {

// 加载配置得到SqlSession

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

// 获取代理对象

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

// 查询数据

User user = userMapper.getUserById(30);

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

// 关闭资源

sqlSession.close();

}

@Test

**public** **void** testGetUserByUserName() {

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

// 获取代理对象

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

// 查询数据

List<User> list = userMapper.getUserByName("张");

**for** (User user : list) {

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

}

// 关闭资源

sqlSession.close();

}

@Test

**public** **void** testInsertUser() {

SqlSession sqlSession = SqlSessionFactoryUtils.*getSqlSessionFactory*().openSession();

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

User user = **new** User();

user.setUsername("张飞飞");

user.setAddress("深圳市黑马");

user.setBirthday(**new** Date());

user.setSex("1");

userMapper.insertUser(user);

// 提交事务

sqlSession.commit();

// 关闭资源

sqlSession.close();

}

}

# SqlMapConf.xml配置



## properties

### 属性核心文件配置

<!-- 加载规则，首先加载标签内部属性，再加载外部文件，名称相同时，会替换相同名称的内容 -->

<properties resource=*"jdbc.properties"*>

<property name=*"jdbc.username"* value=*"root1"*/>

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

</properties>

### jdbc.properties

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8

jdbc.username=root

jdbc.password=root

## typeAliases

**mybatis默认支持java基本数据类型的别名识别详细参考教案。**

**自定义别名**

<typeAliases>

<!-- 单个别名定义 -->

<!-- <typeAlias type="com.itheima.mybatis.pojo.User" alias="user"/> -->

<!-- 别名包扫描器(推荐使用此方式)，整个包下的类都被定义别名，别名为类名，不区分大小写-->

<package name=*"com.itheima.mybatis.pojo"*/>

</typeAliases>

## mappers

<mappers>

<!-- 第一种方式，加载 resource-->

<mapper resource=*"mapper/user.xml"*/>

<!-- <mapper resource="mapper/UserMapper.xml"/> -->

<!-- 第二种方式，class扫描器要求：

1、映射文件与接口同一目录下

2、映射文件名必需与接口文件名称一致

-->

<!-- <mapper class="com.itheima.mybatis.mapper.UserMapper"/> -->

<!-- 第三种方式，包扫描器要求(推荐使用此方式)：

1、映射文件与接口同一目录下

2、映射文件名必需与接口文件名称一致

-->

<package name=*"com.itheima.mybatis.mapper"*/>

</mappers>