**每日作业卷答案**

**MyBatis第1天**

极客营

# 关卡1

## 训练案例1

### 训练描述

使用准备的工程MyBatis01-study实现以下五个功能

1. 实现根据id查询用户，使用#{}
2. 实现根据用户名模糊查询用户，使用${}
3. 实现新增用户
4. 实现更新用户
5. 实现删除用户
6. 编写接口的实现类，使用SqlSession的用户方法完成

### 操作步骤答案

1. 在MySQL客户端中编写以上五个功能的sql语句

SELECT \* FROM `user` WHERE id = 1

SELECT \* FROM `user` WHERE username LIKE '%王%'

INSERT INTO `user` (username,birthday,sex,address) VALUES ('黄忠','2016-07-26','1','三国')

UPDATE `user` SET username = '赵云' WHERE id = 26

DELETE FROM `user` WHERE id = 47

1. 在已经准备好的User.xml页面中编写对应的sql配置

<?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=*"test"*>

<!-- 根据id查询用户 -->

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

resultType=*"com.igeek.mybatis.pojo.User"*>

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

</select>

<!-- 根据用户名查询用户 -->

<select id=*"queryUserByUserName"* parameterType=*"string"*

resultType=*"com.igeek.mybatis.pojo.User"*>

select \* from user where username like '%${value}%'

</select>

<!-- 新增用户 -->

<insert id=*"saveUser"* parameterType=*"com.igeek.mybatis.pojo.User"*>

insert into user

(username,birthday,sex,address) values

(#{username},#{birthday},#{sex},#{address})

</insert>

<!-- 更新用户 -->

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

UPDATE `user` SET

address = #{address}

WHERE username = #{username}

</update>

<!-- 删除用户 -->

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

DELETE FROM `user`

WHERE id=#{id}

</delete>

</mapper>

1. 在UserDaoImpl中使用SqlSession实现接口方法

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

**private** SqlSessionFactory sqlSessionFactory;

**public** UserDaoImpl(SqlSessionFactory sqlSessionFactory) {

**super**();

**this**.sqlSessionFactory = sqlSessionFactory;

}

@Override

**public** User queryUserById(**int** id) {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

sqlSession.close();

**return** user;

}

@Override

**public** List<User> queryUserByUserName(String userName) {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

List<User> list = sqlSession.selectList("queryUserByUserName", userName);

sqlSession.close();

**return** list;

}

@Override

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

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

sqlSession.insert("saveUser", user);

sqlSession.commit();

sqlSession.close();

}

@Override

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

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

sqlSession.update("updateUser", user);

sqlSession.commit();

sqlSession.close();

}

@Override

**public** **void** deleteUserById(**int** id) {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

sqlSession.insert("deleteUserById", 38);

sqlSession.commit();

sqlSession.close();

}

}

1. 在SqlMapConfig.xml中配置User.xml的加载

<mapper resource=*"sqlmap/User.xml"* />

1. 在UserDaoTest编写测试类，测试五个方法

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

**private** UserDao userDao;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

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

SqlSessionFactory sqlSessionFactory = builder.build(inputStream);

**this**.userDao = **new** UserDaoImpl(sqlSessionFactory);

}

@Test

**public** **void** testQueryUserById() {

User user = **this**.userDao.queryUserById(1);

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

}

@Test

**public** **void** testQueryUserByUserName() {

List<User> list = **this**.userDao.queryUserByUserName("明");

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

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

}

}

@Test

**public** **void** testSaveUser() {

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

user.setUsername("张小三");

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

user.setSex("1");

user.setAddress("湖北");

**this**.userDao.saveUser(user);

}

@Test

**public** **void** testUpdateUser() {

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

user.setUsername("张小三");

user.setAddress("北京");

**this**.userDao.updateUser(user);

}

@Test

**public** **void** testDeleteUser() {

**this**.userDao.deleteUserById(1);

}

}

## 训练案例1

### 训练描述

使用准备的工程MyBatis01-study实现以下五个功能

1. 实现根据id查询用户，使用#{}
2. 实现根据用户名模糊查询用户，使用${}
3. 实现新增用户，并能返回自增的id
4. 实现更新用户
5. 实现删除用户

### 操作步骤答案

1. 在MySQL客户端中编写以上五个功能的sql语句

SELECT \* FROM `user` WHERE id = 1

SELECT \* FROM `user` WHERE username LIKE '%王%'

INSERT INTO `user` (username,birthday,sex,address) VALUES ('黄忠','2016-07-26','1','三国')

UPDATE `user` SET username = '赵云' WHERE id = 26

DELETE FROM `user` WHERE id = 47

1. 在已经准备好的UserMapper.xml页面中编写对应的sql配置

<?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.igeek.mybatis.dao.UserMapper"*>

<!-- 根据id查询用户 -->

<select id=*"queryUserById"* parameterType=*"InT"*

resultType=*"com.igeek.mybatis.pojo.User"*>

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

</select>

<!-- 根据用户名查询用户 -->

<select id=*"queryUserByUserName"* parameterType=*"string"*

resultType=*"com.igeek.mybatis.pojo.User"*>

select \* from user where username like '%${value}%'

</select>

<!-- 新增用户 -->

<insert id=*"saveUser"* parameterType=*"com.igeek.mybatis.pojo.User"*>

insert into user

(username,birthday,sex,address) values

(#{username},#{birthday},#{sex},#{address})

</insert>

<!-- 更新用户 -->

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

UPDATE `user` SET

address = #{address}

WHERE username = #{username}

</update>

<!-- 删除用户 -->

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

DELETE FROM `user`

WHERE id=#{id}

</delete>

</mapper>

1. 在UserDao中编写接口方法

**public** **interface** UserMapper {

**public** User queryUserById(**int** id);

**public** List<User> queryUserByUserName(String userName);

**public** **void** saveUser(User user);

**public** **void** updateUser(User user);

**public** **void** deleteUserById(**int** id);

}

1. 在SqlMapConfig.xml中配置UserMapper.xml的加载

<mapper resource=*"sqlmap/UserMapper.xml"* />

1. 在UserMapperTest编写测试类，测试五个方法

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

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

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

**this**.sqlSessionFactory = builder.build(inputStream);

}

@Test

**public** **void** testQueryUserById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

User user = userMapper.queryUserById(1);

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

sqlSession.close();

}

@Test

**public** **void** testQueryUserByUserName() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

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

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

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

}

sqlSession.close();

}

@Test

**public** **void** testSaveUser() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

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

user.setUsername("拉莫斯");

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

user.setSex("1");

user.setAddress("披甲龙龟");

userMapper.saveUser(user);

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testUpdateUser() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

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

user.setUsername("拉莫斯");

user.setAddress("披甲龙龟");

userMapper.updateUser(user);

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testDeleteUser() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

userMapper.deleteUserById(39);

sqlSession.commit();

sqlSession.close();

}

}

# 关卡2

## 训练案例1

### 训练描述

使用准备的工程MyBatis01-study实现以下五个功能

1. 实现根据id查询订单，使用${}
2. 实现根据备注note模糊查询订单，使用#{}
3. 实现新增订单，并能返回自增的id
4. 实现根据id更新订单note
5. 实现根据id删除订单

### 操作步骤答案

1. 在MySQL客户端中编写以上五个功能的sql语句

SELECT \* FROM `order` WHERE id = 1

SELECT \* FROM `order` WHERE note LIKE '%价%'

INSERT INTO `order` (`user\_id`, `number`, `createtime`, `note`) VALUES ('1', '1000010', '2015-02-04 13:22:35','活动价');

UPDATE `order` SET note = '活动价1' WHERE id = 6

DELETE FROM `order` WHERE id = 6

1. 在OrderMapper.xml页面中编写对应的sql配置

<?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.igeek.mybatis.dao.OrderMapper"*>

<!-- 根据id查询 -->

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

resultType=*"com.igeek.mybatis.pojo.Order"*>

select \* from `order` where id = ${value}

</select>

<!-- 根据用note查询 -->

<select id=*"queryOrderByNote"* parameterType=*"string"*

resultType=*"com.igeek.mybatis.pojo.Order"*>

select \* from `order` where note like #{note}

</select>

<!-- 新增 -->

<insert id=*"saveOrder"*parameterType=*"com.igeek.mybatis.pojo.Order"*>

<selectKey keyColumn=*"id"* keyProperty=*"id"* order=*"AFTER"*

resultType=*"int"*>

SELECT LAST\_INSERT\_ID()

</selectKey>

INSERT INTO `order`

(`user\_id`, `number`, `createtime`, `note`) VALUES

(#{user\_id},#{number},#{createtime},#{note})

</insert>

<!-- 更新 -->

<update id=*"updateOrder"* parameterType=*"com.igeek.mybatis.pojo.Order"*>

UPDATE `order` SET

note =

#{note}

WHERE id = #{id}

</update>

<!-- 删除 -->

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

DELETE FROM `order`

WHERE id=#{id}

</delete>

</mapper>

1. 在OrderMapper中编写接口方法

**public** **interface** OrderMapper {

**public** Order queryOrderById(**int** id);

**public** List<Order> queryOrderByNote(String note);

**public** **void** saveOrder(Order user);

**public** **void** updateOrder(Order user);

**public** **void** deleteOrderById(**int** id);

}

1. 在SqlMapConfig.xml中配置OrderMapper.xml的加载

<mapper resource=*"sqlmap/OrderMapper.xml"* />

1. 在OrderMapperTest编写测试类，测试五个方法

**public** **class** OrderMapperTest {

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

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

**this**.sqlSessionFactory = builder.build(inputStream);

}

@Test

**public** **void** testQueryOrderById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);

Order order = orderMapper.queryOrderById(3);

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

sqlSession.close();

}

@Test

**public** **void** testQueryOrderByOrderName() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

List<Order> list = userMapper.queryOrderByNote("价");

**for** (Order order : list) {

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

}

sqlSession.close();

}

@Test

**public** **void** testSaveOrder() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

Order order = **new** Order();

order.setNumber("1001");

order.setUser\_id(1);

order.setCreatetime(**new** Date());

order.setNote("活动价");

userMapper.saveOrder(order);

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

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testUpdateOrder() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

Order order = **new** Order();

order.setId(7);

order.setNote("12344");

userMapper.updateOrder(order);

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testDeleteOrder() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

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

userMapper.deleteOrderById(7);

sqlSession.commit();

sqlSession.close();

}

}

## 训练案例1

### 训练描述

1. 配置properties，加载db.properties
2. 配置使用别名
3. 使用三种方式加载Mapper.xml配置文件

### 操作步骤答案

1. 在SqlMapConfig.xml中进行配置，使之加载db.properties，并使用db.properties的配置信息

<properties resource=*"db.properties"* />

<!-- 和spring整合后 environments配置将废除 -->

<environments default=*"development"*>

<environment id=*"development"*>

<!-- 使用jdbc事务管理 -->

<transactionManager type=*"JDBC"* />

<!-- 数据库连接池 -->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${jdbc.driver}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

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

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

</dataSource>

</environment>

</environments>

1. 在SqlMapConfig.xml中配置别名扫描包

<typeAliases>

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

</typeAliases>

1. 在Mapper.xml中配置使用别名

<!-- 根据id查询用户 -->

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

resultType=*"order"*>

select \* from `order` where id = ${value}

</select>

1. 在SqlMapConfig.xml中配置使用类路径的方式加载Mapper.xml

先把OrderMapper.xml和UserMapper.xml复制到com.igeek.mybatis.dao包下

再修改SqlMapConfig.xml配置如下：

<mappers>

<!-- 加载Mapper.xml配置 -->

<mapper resource=*"sqlmap/User.xml"* />

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

<mapper class=*"com.igeek.mybatis.dao.UserMapper"* />

<!-- <mapper resource="sqlmap/OrderMapper.xml" /> -->

<mapper class=*"com.igeek.mybatis.dao.OrderMapper"* />

</mappers>

1. 在SqlMapConfig.xml中配置使用包扫描的方式加载Mapper.xml

修改SqlMapConfig.xml配置如下：

<mappers>

<!-- 加载Mapper.xml配置 -->

<mapper resource=*"sqlmap/User.xml"* />

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

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

<!-- <mapper resource="sqlmap/OrderMapper.xml" /> -->

<!-- <mapper class="com.igeek.mybatis.dao.OrderMapper" /> -->

<package name=*"com.igeek.mybatis.dao"* />

</mappers>

# 关卡3

## 训练案例1

### 训练描述

先根据以下sql创建数据库表：

CREATE TABLE `tb\_order` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '订单id',

`payment` varchar(50) COLLATE utf8\_bin DEFAULT NULL COMMENT '实付金额。精确到2位小数;单位:元。如:200.07，表示:200元7分',

`paymentType` int(2) DEFAULT NULL COMMENT '支付类型，1、在线支付，2、货到付款',

`postFee` varchar(50) COLLATE utf8\_bin DEFAULT NULL COMMENT '邮费。精确到2位小数;单位:元。如:200.07，表示:200元7分',

`status` int(10) DEFAULT NULL COMMENT '状态：1、未付款，2、已付款，3、未发货，4、已发货，5、交易成功，6、交易关闭',

`createTime` datetime DEFAULT NULL COMMENT '订单创建时间',

`updateTime` datetime DEFAULT NULL COMMENT '订单更新时间',

`paymentTime` datetime DEFAULT NULL COMMENT '付款时间',

`consignTime` datetime DEFAULT NULL COMMENT '发货时间',

`endTime` datetime DEFAULT NULL COMMENT '交易完成时间',

`closeTime` datetime DEFAULT NULL COMMENT '交易关闭时间',

`userId` bigint(20) DEFAULT NULL COMMENT '用户id',

`buyerMessage` varchar(100) COLLATE utf8\_bin DEFAULT NULL COMMENT '买家留言',

`buyerNick` varchar(50) COLLATE utf8\_bin DEFAULT NULL COMMENT '买家昵称',

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_bin;

使用以上数据库表，实现以下功能:

1. 根据数据库表创建pojo

2. 实现创建订单

3. 实现根据订单id查询订单

4. 实现根据订单创建的时间范围(例如查询3月26日这一天的)，查询所有支付类型为在线支付，而且已付款的订单

5. 实现根据买家留言模糊查询订单

6. 实现根据订单id修改订单

7. 实现根据订单创建的时间范围(例如查询3月26日这一天的)，修改所有支付类型是在线支付的，而且未付款的，把订单状态修改为交易关闭

8. 实现根据id删除订单

### 操作步骤答案

1. 根据数据库表创建pojo

**public** **class** TbOrder {

**private** Long id;

**private** String payment;

**private** **int** paymentType;

**private** String postFee;

**private** **int** status;

**private** Date createTime;

**private** Date updateTime;

**private** Date paymentTime;

**private** Date consignTime;

**private** Date endTime;

**private** Date closeTime;

**private** Long userId;

**private** String buyerMessage;

**private** String buyerNick;

}

2. 实现创建订单

3. 实现根据订单id查询订单

4. 实现根据订单创建的时间范围(例如查询3月26日这一天的)，查询所有支付类型为在线支付，而且已付款的订单

5. 实现根据买家留言模糊查询订单

6. 实现根据订单id修改订单

7. 实现根据订单创建的时间范围(例如查询3月26日这一天的)，修改所有支付类型是在线支付的，而且未付款的，把订单状态修改为交易关闭

8. 实现根据id删除订单

编写TbOrderMapper接口

**public** **interface** TbOrderMapper {

// 实现创建订单

**void** saveTbOrder(TbOrder tbOrder);

// 实现根据订单id查询订单

TbOrder queryTbOrderById(Long id);

// 实现根据订单创建的时间范围(例如查询3月26日这一天的)，查询所有支付类型为在线支付，而且已付款的订单

List<TbOrder> queryTbOrderByDate(Map<String, Object> map);

// 实现根据买家留言模糊查询订单

List<TbOrder> queryTbOrderByBuyerMessage(String msg);

// 实现根据订单id修改订单

**void** updateTbOrderById(TbOrder tbOrder);

// 实现根据订单创建的时间范围(例如查询3月26日这一天的)，修改所有支付类型是在线支付的，而且未付款的，把订单状态修改为交易关闭

**void** updateTbOrderByDate(Map<String, Object> map);

// 实现根据id删除订单

**void** deleteTbOrderById(Long id);

}

编写TbOrderMapper.xml配置文件

<?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=*"cn.itcast.mybatis.dao.TbOrderMapper"*>

<!-- 实现创建订单 -->

<!-- void saveTbOrder(TbOrder tbOrder); -->

<insert id=*"saveTbOrder"* parameterType=*"tbOrder"*>

INSERT INTO `tb\_order` (

`payment`,

`paymentType`,

`postFee`,

`status`,

`createTime`,

`updateTime`,

`paymentTime`,

`consignTime`,

`endTime`,

`closeTime`,

`userId`,

`buyerMessage`,

`buyerNick`

)

VALUES

(

#{payment},

#{paymentType},

#{postFee},

#{status},

#{createTime},

#{updateTime},

#{paymentTime},

#{consignTime},

#{endTime},

#{closeTime},

#{userId},

#{buyerMessage},

#{buyerNick}

)

</insert>

<!-- 实现根据订单id查询订单 -->

<!-- TbOrder queryTbOrderById(Long id); -->

<select id=*"queryTbOrderById"* parameterType=*"long"* resultType=*"tbOrder"*>

SELECT \* FROM tb\_order WHERE id=#{id}

</select>

<!-- 实现根据订单创建的时间范围(例如查询3月26日这一天的)，查询所有支付类型为在线支付，而且已付款的订单 -->

<!-- List<TbOrder> queryTbOrderByDate(Map<String, Object> map); -->

<select id=*"queryTbOrderByDate"* parameterType=*"map"* resultType=*"tbOrder"*>

SELECT

\*

FROM

tb\_order

WHERE

createTime BETWEEN #{startTime}

AND #{endTime}

AND paymentType = 1

AND STATUS = 2

</select>

<!-- 实现根据买家留言模糊查询订单 -->

<!-- List<TbOrder> queryTbOrderByBuyerMessage(String msg); -->

<select id=*"queryTbOrderByBuyerMessage"* parameterType=*"string"*

resultType=*"tbOrder"*>

SELECT

\*

FROM

tb\_order

WHERE

buyerMessage LIKE '%${value}%'

</select>

<!-- 实现根据订单id修改订单 -->

<!-- void updateTbOrderById(TbOrder tbOrder); -->

<update id=*"updateTbOrderById"* parameterType=*"tbOrder"*>

UPDATE `tb\_order`

SET

`payment` = #{payment},

`paymentType` = #{paymentType},

`postFee` =

#{postFee},

`status` = #{status},

`createTime` = #{createTime},

`updateTime` = #{updateTime},

`paymentTime` = #{paymentTime},

`consignTime` = #{consignTime},

`endTime` = #{endTime},

`closeTime` =

#{closeTime},

`userId` = #{userId},

`buyerMessage` = #{buyerMessage},

`buyerNick` = #{buyerNick}

WHERE

(`id` = #{id})

</update>

<!-- 实现根据订单创建的时间范围(例如查询3月26日这一天的)，修改所有支付类型是在线支付的，而且未付款的，把订单状态修改为交易关闭 -->

<!-- void updateTbOrderByDate(Map<String, Object> map); -->

<update id=*"updateTbOrderByDate"* parameterType=*"map"*>

UPDATE `tb\_order`

SET `status` = '6'

WHERE

createTime

BETWEEN #{startTime}

AND #{endTime}

AND `paymentType` = 1

AND `status`

= 1

</update>

<!-- 实现根据id删除订单 -->

<!-- void deleteTbOrderById(Long id); -->

<delete id=*"deleteTbOrderById"* parameterType=*"long"*>

DELETE FROM

tb\_order WHERE `id`=#{id}

</delete>

</mapper>

编写测试类

**public** **class** TbOrderMapperTest {

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

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

**this**.sqlSessionFactory = builder.build(inputStream);

}

@Test

**public** **void** testSaveTbOrder() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

TbOrder tbOrder = **new** TbOrder();

tbOrder.setStatus(1);

tbOrder.setCreateTime(**new** Date());

tbOrder.setPaymentType(1);

tbOrder.setBuyerMessage("好东西");

tbOrderMapper.saveTbOrder(tbOrder);

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testQueryTbOrderById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

TbOrder tbOrder = tbOrderMapper.queryTbOrderById(4l);

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

sqlSession.close();

}

@Test

**public** **void** testQueryTbOrderByDate() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

Map<String, Object> map = **new** HashMap<>();

map.put("startTime", **new** Date(0l));

map.put("endTime", **new** Date());

List<TbOrder> list = tbOrderMapper.queryTbOrderByDate(map);

**for** (TbOrder o : list) {

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

}

sqlSession.close();

}

@Test

**public** **void** testQueryTbOrderByBuyerMessage() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

List<TbOrder> list = tbOrderMapper.queryTbOrderByBuyerMessage("好");

**for** (TbOrder o : list) {

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

}

sqlSession.close();

}

@Test

**public** **void** testUpdateTbOrderById() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

TbOrder tbOrder = **new** TbOrder();

tbOrder.setId(3l);

tbOrder.setStatus(1);

tbOrderMapper.updateTbOrderById(tbOrder);

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testUpdateTbOrderByDate() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

Map<String, Object> map = **new** HashMap<>();

map.put("startTime", **new** Date(0l));

map.put("endTime", **new** Date());

tbOrderMapper.updateTbOrderByDate(map);

sqlSession.commit();

sqlSession.close();

}

@Test

**public** **void** testDeleteTbOrderByIdLong() {

SqlSession sqlSession = **this**.sqlSessionFactory.openSession();

TbOrderMapper tbOrderMapper = sqlSession.getMapper(TbOrderMapper.**class**);

tbOrderMapper.deleteTbOrderById(1l);

sqlSession.commit();

sqlSession.close();

}

}