框架阶段学习：

1、mybatis：3天的时间。mysql和jdbc

2、spring：4天的时间。动态代理

3、spring mvc：3天的时间。javaweb（监听器、Servlet映射）

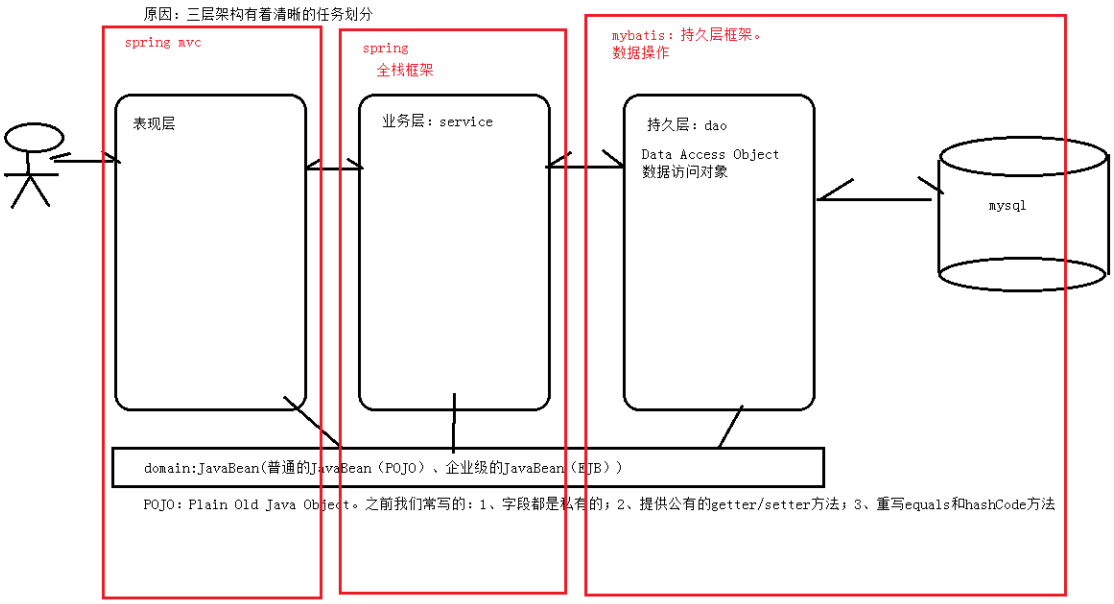
1. 三层架构

1.1. 三层架构

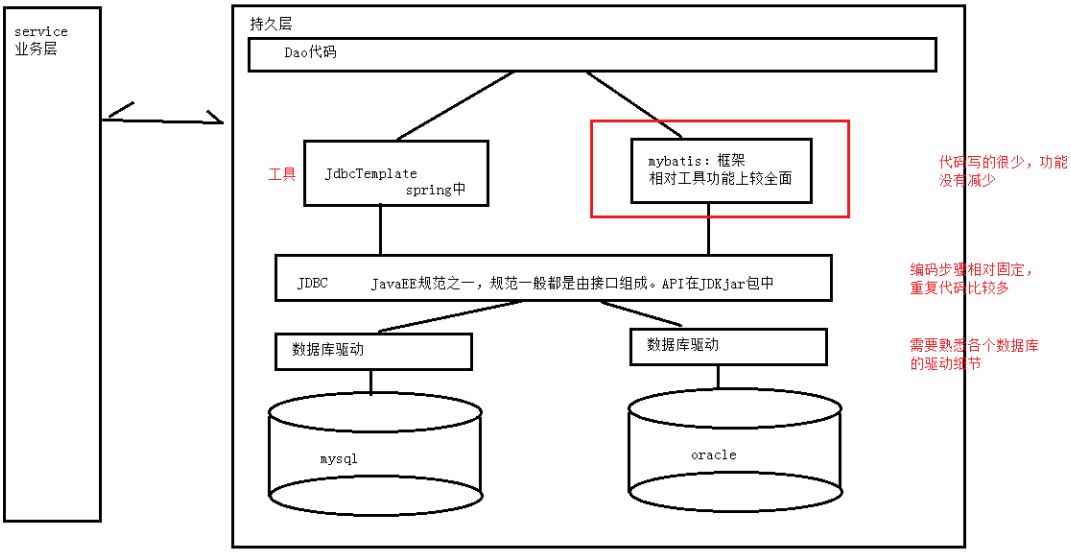
表现层：**spring mvc。**struts1、struts2.

业务层：**spring。**很多企业都在使用。全栈框架（各层都有）

持久层：**mybatis。**Hibernate、**spring jdbc、JPA（Java Persistence API，oracle提供的持久层标准）**等



1.2. 什么是框架

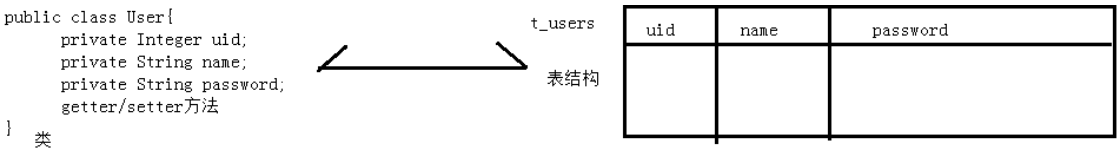


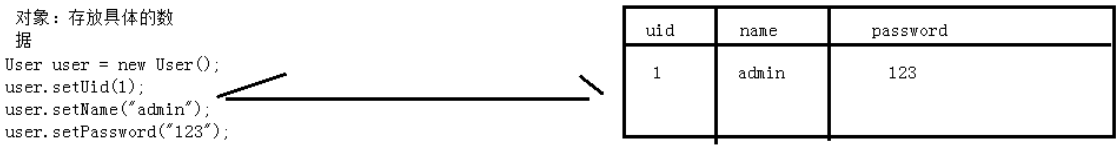
工具也可以叫做框架，只是功能相对较小，所有工具的称为比较合适。功能比较全面的叫做框架。

框架的作用：抽取了重复的代码，让程序员们集中精力放在要实现的核心代码上，提高开发效率。

实际开发：技术能力+业务能力

2. ORM映射





根据类设计表结构：正向映射（Mapping）
根据表结构设计类：反向映射（Mapping）

O：Object对象
R：Relational关系数据
M：Mapping映射

ORM：对象关系映射。持久层的框架都是ORM映射。比如mybatis、Hibernate等等

3. mybatis简介

mybatis有关的类包名以：org.apache.ibatis开头。

mybatis最大的亮点是：

1、可以通过编写sql语句（xml文件中）的形式指挥框架的运行。原汁原味的sql语句执行效率是很高的（Hibernate无法比拟的）。

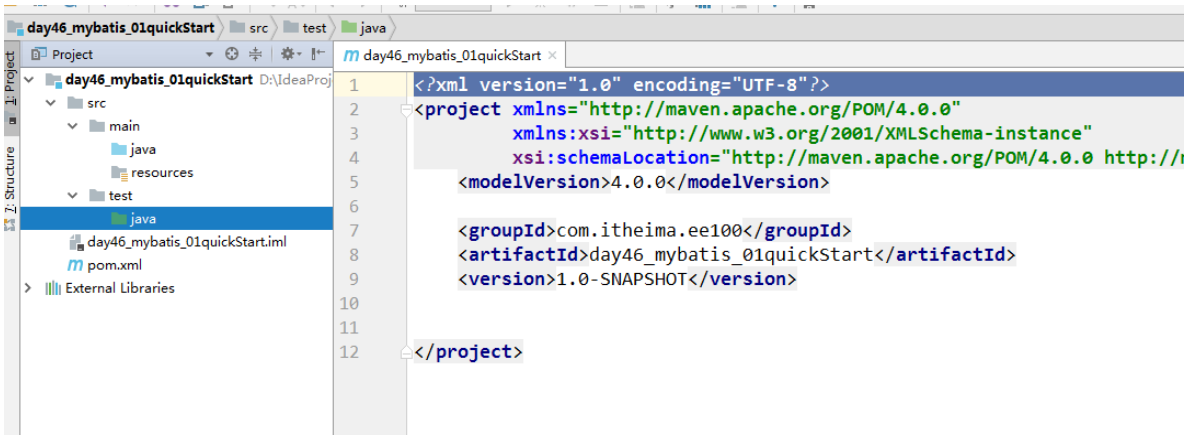
2、开发中只需要编写dao接口就可以了，自动生成该接口的代理对象。

4. mybatis的入门案例（重点）

4.1. 搭建开发环境

mybatis的网站：http://www.mybatis.org/mybatis-3/zh/index.html

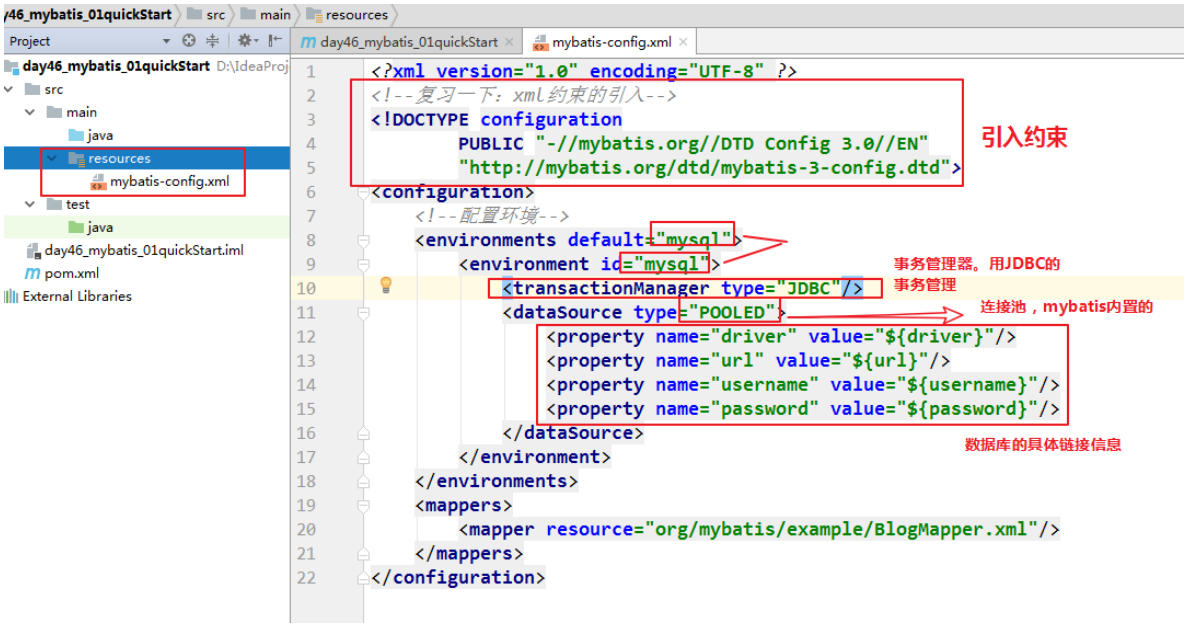
step1：建立一个新的javase工程即可



step2：导入jar包（pom.xml）

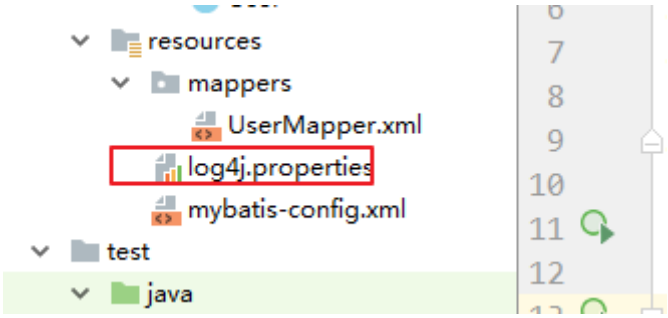
```
<dependencies>
        <!--mybatis的jar包-->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.0</version>
        </dependency>
        <!--数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--单元测试-->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <!--日志：学习期间可以打印一些内容在控制台。
        便于理解和学习-->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
    </dependencies>
```

step3:建立mybatis的主配置文件




```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!--复习一下：xml约束的引入-->
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!--配置环境-->
    <environments default="mysql">
        <environment id="mysql">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///ee100"/>
                <property name="username" value="root"/>
                <property name="password" value="sorry"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

step4：加入log4j的配置文件（学习阶段）



4.2. 入门案例（了解，做一遍）

向数据库的Users表中添加一条记录

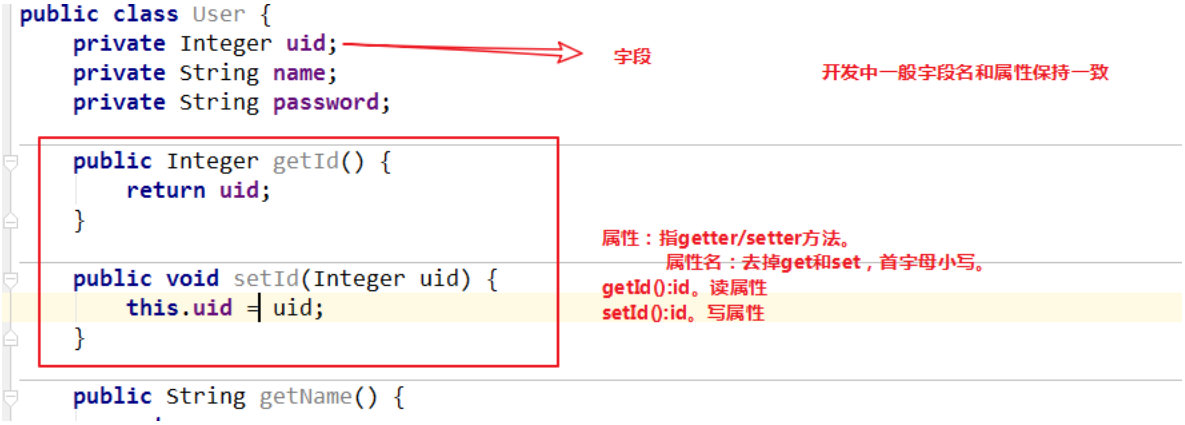
step1:编写实体类

day46_mybatis_01quickStart | mybatis-config.xml | User.java

```java
package com.itheima.domain;

public class User {
    private Integer uid;
    private String name;
    private String password;

    public Integer getUid() {
        return uid;
    }

    public void setUid(Integer uid) {
        this.uid = uid;
    }

    public String getName() {
```

step2：创建数据库和表结构

最佳实践：类中的属性名和数据库表的字段名保持一致



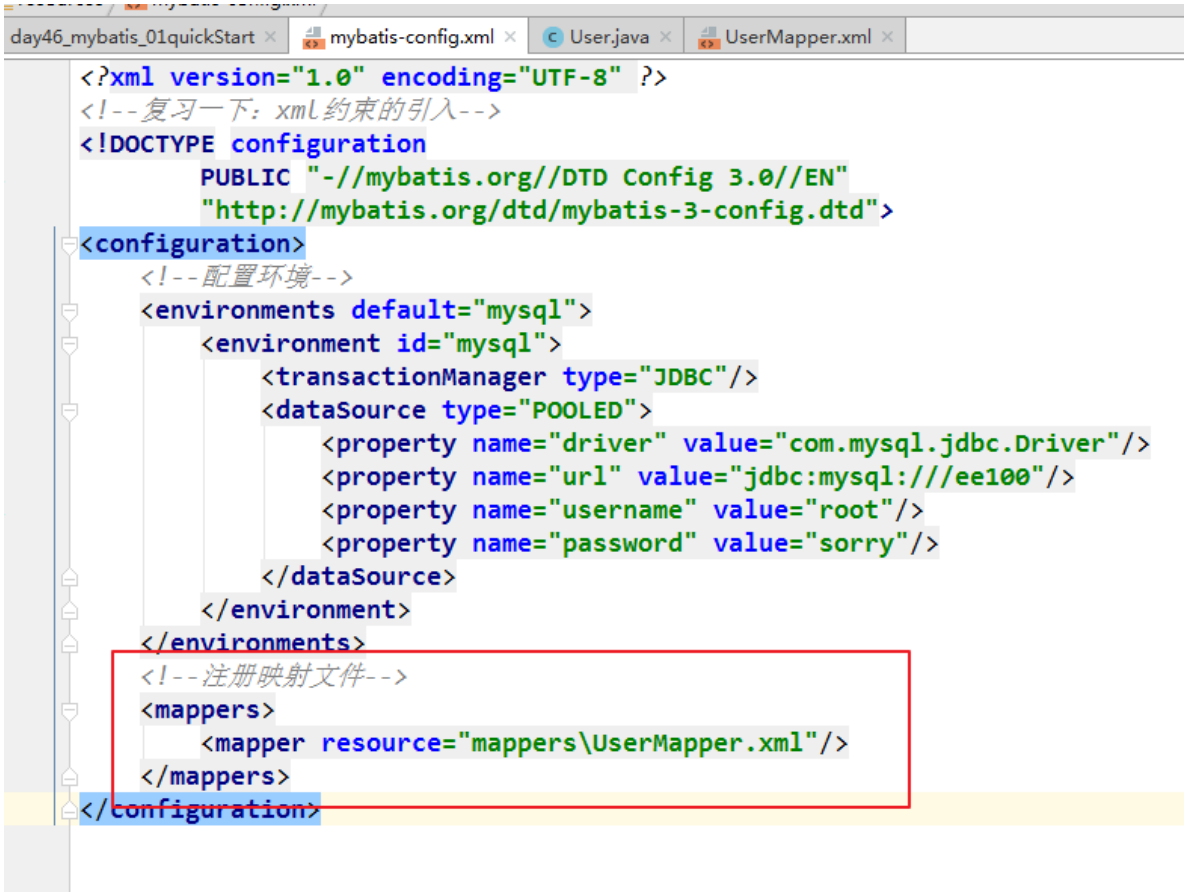

```sql
create database ee100;
use ee100;
create table users(
    uid int primary key auto_increment,
    name varchar(100),
    password varchar(100)
);
```

step3:编写映射文件

```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">
<!--namespace类似java中的package-->
<mapper namespace="usermapper">
    <!--id：当前文件中要唯一。
        parameterType:存放的数据的参数类型
        sql语句：写在标签内部
    -->
    <insert id="addUser">
```

```
        insert into users values(null,"admin","123")
    </insert>
</mapper>
```

step4、注册映射文件到mybatis的主配置文件中



step5、编写测试类

```
public class UserTest {
    @Test
    public void testAddUser() throws Exception{//学习方法：记
        //加载mybatis的主配置文件
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");//从类路径中加载配置文件
        //获取SqlSessionFactory对象：根据配置文件获取的
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        //获取SqlSession的对象：操作数据库主要接口
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //操作数据即可
        sqlSession.insert( s: "usermapper.addUser");//insert(String s):不是sql语句，而是配置文件中的namespace+id
        //提交事务
        sqlSession.commit();        不是自动提交事务的
        //关闭SqlSession对象
        sqlSession.close();
    }
}
```

5. mybatis基于Dao实现类的编码方式（了解）

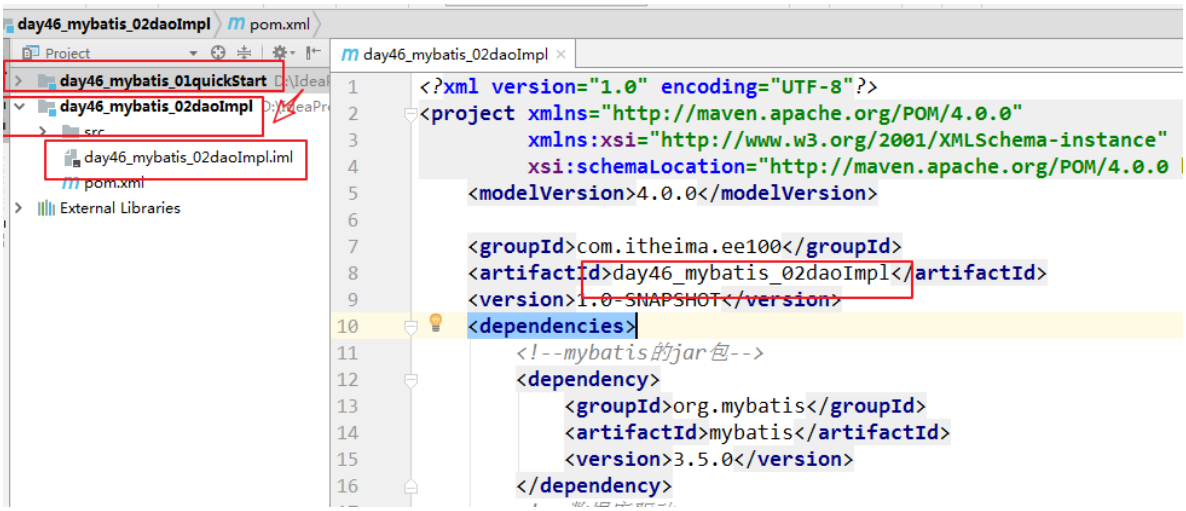
step1:拷贝建立一个新的模块

a、更改pom中的

b、更改模块的名称

c、删除多余的iml文件

d、删除target目录



step2：编写一个dao的接口

```java
public interface UserDao {
    void saveUser(User user);
}
```

step3：编写dao的实现

```java
public class UserDaoImpl implements UserDao {
    public void saveUser(User user) {
        try {
            //加载mybatis的主配置文件
            InputStream in = Resources.getResourceAsStream("mybatis-config.xml");//从类路径中加载配置文件
            //获取SqlSessionFactory对象：根据配置文件获取的
            SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
            //获取SqlSession的对象：操作数据库主要接口
            SqlSession sqlSession = sqlSessionFactory.openSession();
            //操作数据即可
            sqlSession.insert( s: "usermapper.addUser");//insert(String s):不是sql语句，而是配置文件中的namespace+id
            //提交事务
            sqlSession.commit();
            //关闭SqlSession对象
            sqlSession.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

方法和类写得较多的时候，此部分代码会变为重复劳动，开发效率低下

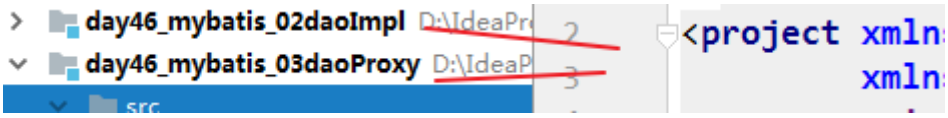
step4：测试

```java
public class UserTest {
    @Test
    public void testAddUser() throws Exception{//学习方法：记
        UserDao userDao = new UserDaoImpl();
        userDao.saveUser(null);
    }
}
```

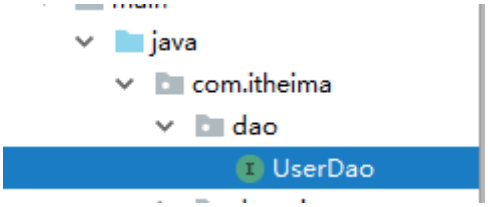
6. mybatis基于Dao动态代理的编码方式（重点）

动态代理：mybatis会使用动态代理技术，自动生成Dao接口的代理对象。我们直接用即可。

step1：拷贝建立 一个新的模块



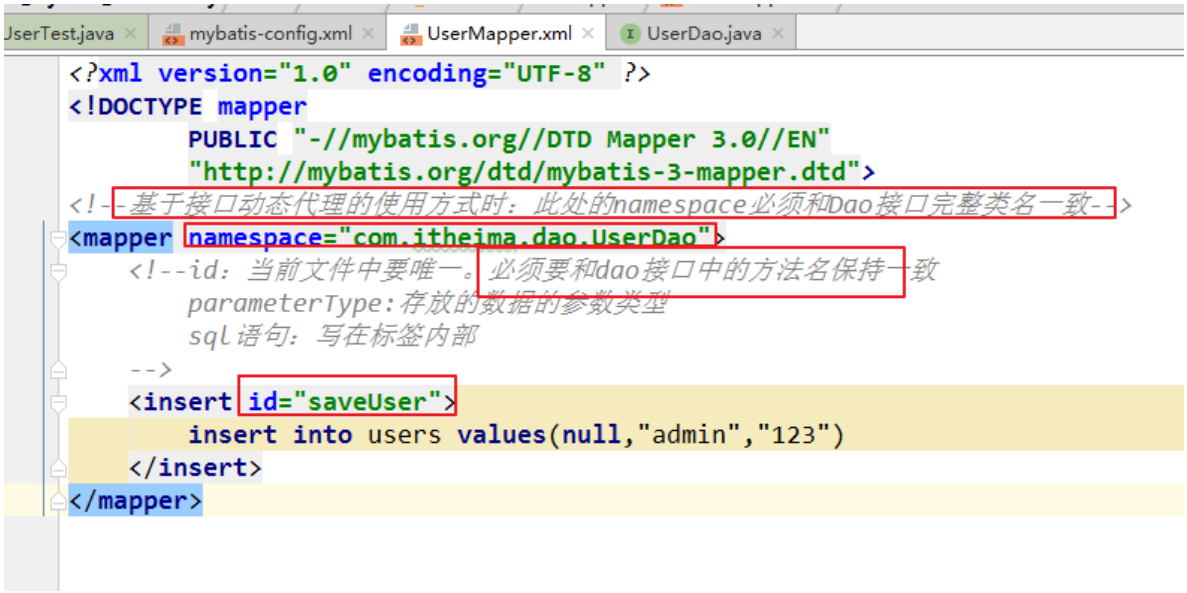
step2：删除掉dao的实现类，只留接口即可



step3：修改映射文件（很重要）

a、namespace的取值必须和Dao接口的完整类名完全一致

b、标签的id取值，必须和dao接口中的方法名保持一致



step4、测试类

```
    @Test
    public void testAddUser() throws Exception{//学习方法：记
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //获取Dao接口的代理对象
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        userDao.saveUser(null);
        sqlSession.commit();
        sqlSession.close();
    }
}
```

step5：小结

日后编写mybatis的dao遵循步骤：（重点）

a、编写Dao接口

b、编写Dao对应的映射文件：

c、测试：sqlSession.getMapper(Dao.class)

7. mybatis中常用的类或接口(重点)

- Resources：作用，加载类路径下的配置文件。

InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

- SqlSessionFactory：是mybatis中最为重要的类。她内部存放了所有与mybatis有关的信息。绝大多数情况下，一个应用只有一个SqlSessionFactory对象（单例）。相对来说是一个重量级的对象，初始化需要耗费较多的资源。线程安全的。
- SqlSession：mybatis提供给**程序员**使用的主要接口。他不是线程安全的。最佳实践：每个线程有独自的SqlSession对象。用的时候创建，用完就关闭掉。

8. 抽取工具类和测试基类（目前有用，学了spring没有用哪个了）

8.1. 抽取工具类

UserTest.java | MyBatisUtil.java

```java
package com.itheima.util;

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 java.io.IOException;
import java.io.InputStream;

//和SqlSessionFactory和SqlSession有关
public class MyBatisUtil {
    private static SqlSessionFactory sqlSessionFactory;
    static{
        try {
            InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public SqlSession openSession(){
        return sqlSessionFactory.openSession();
    }
}
```

8.2. 抽取一个单元测试的基类（Junit）

```java
public class BaseTester {
    protected SqlSession sqlSession;
    @Before//在测试方法之前执行
    public void init(){
        sqlSession = MyBatisUtil.openSession();
    }

    @After
    public void destory(){
        sqlSession.close();
    }
}
```

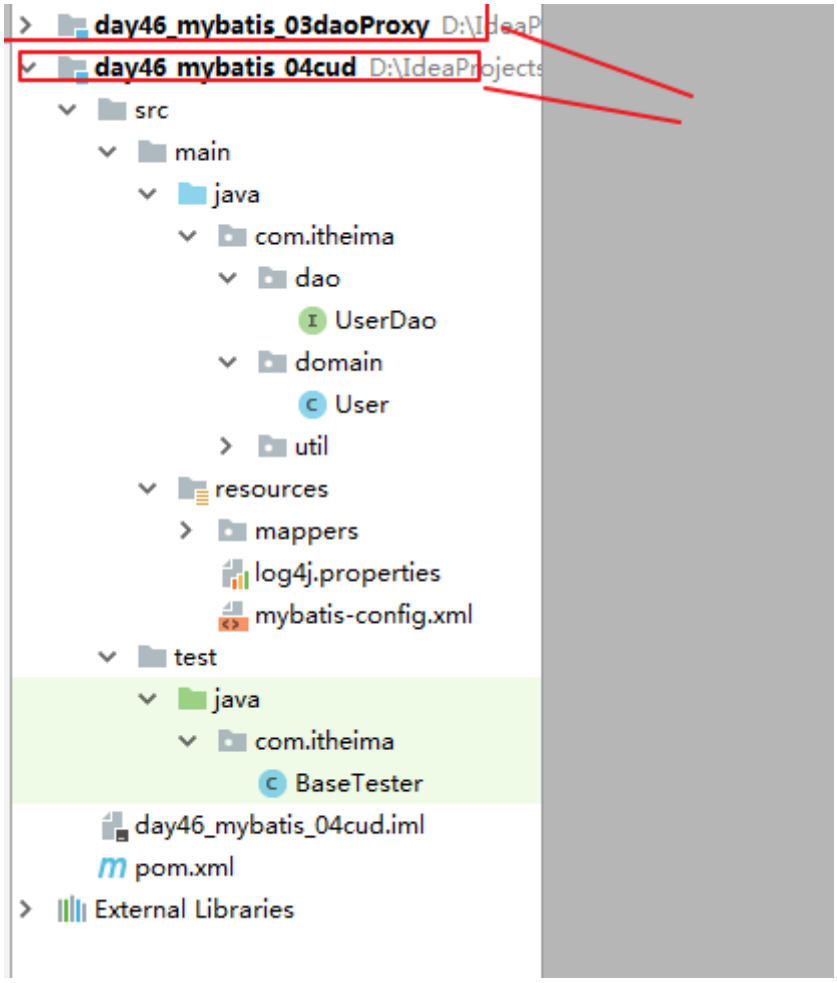
使用：

```java
public class UserTest extends BaseTester{

    @Test
    public void testAddUser1() throws Exception{//学习方法：记
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        userDao.saveUser(null);
        sqlSession.commit();
    }
```

9. 利用mybatis完成增删改操作(重点)

9.1. 添加数据

step1：新拷贝一个模块



step2：编写dao接口即可

```
public interface UserDao {
    void saveUser(User user);
    void updateUser(User user);
    void deleteUser(Integer uid);
}
```

step3：添加：编写映射文件

```
<mapper namespace="com.itheima.dao.UserDao">
    <!--
    parameterType:指定方法中的参数类型。一般情况下，该属性可以省略不写
    SQL语句：使用占位符，mybaits中的占位符用#{}
            如果parameterType参数类型是一个POJO，占位符中要写POJO的属性名称
    -->
    <insert id="saveUser" parameterType="com.itheima.domain.User">
        insert into users values(null,#{name},#{password})
    </insert>
</mapper>
```

和接口保持一致

和User中的属性保持一致

step4：添加：测试

```java
@Test
public void saveUser(){
    User user = new User();
    user.setName("陈云婷");
    user.setPassword("123");

    UserDao userDao = sqlSession.getMapper(UserDao.class);
    userDao.saveUser(user);
    sqlSession.commit();
}
```

UserDaoTest › saveUser()

```
4 rows in set

mysql> select * from users;
+-----+--------+----------+
| uid | name   | password |
+-----+--------+----------+
|   2 | admin  | 123      |
|   3 | admin  | 123      |
|   4 | admin  | 123      |
|   5 | admin  | 123      |
|   6 | 陈云婷 | 123      |
+-----+--------+----------+
```

step5：修改：编写映射文件

```xml
    <update id="updateUser" parameterType="com.itheima.domain.User">
        update users set name=#{name},password=#{password} where uid=#{uid}
    </update>
</mapper>
```

step6：修改：测试

```java
@Test
public void updateUser(){
    User user = new User();
    user.setUid(2);
    user.setName("王猛");
    user.setPassword("1234");

    UserDao userDao = sqlSession.getMapper(UserDao.class);
    userDao.updateUser(user);
    sqlSession.commit();
}
```

```
|   4 | admin  | 123      |
|   5 | admin  | 123      |
|   6 | 陈云婷 | 123      |
+-----+--------+----------+
5 rows in set

mysql> select * from users;
+-----+--------+----------+
| uid | name   | password |
+-----+--------+----------+
|   2 | 王猛   | 1234     |
|   3 | admin  | 123      |
|   4 | admin  | 123      |
|   5 | admin  | 123      |
|   6 | 陈云婷 | 123      |
+-----+--------+----------+
```

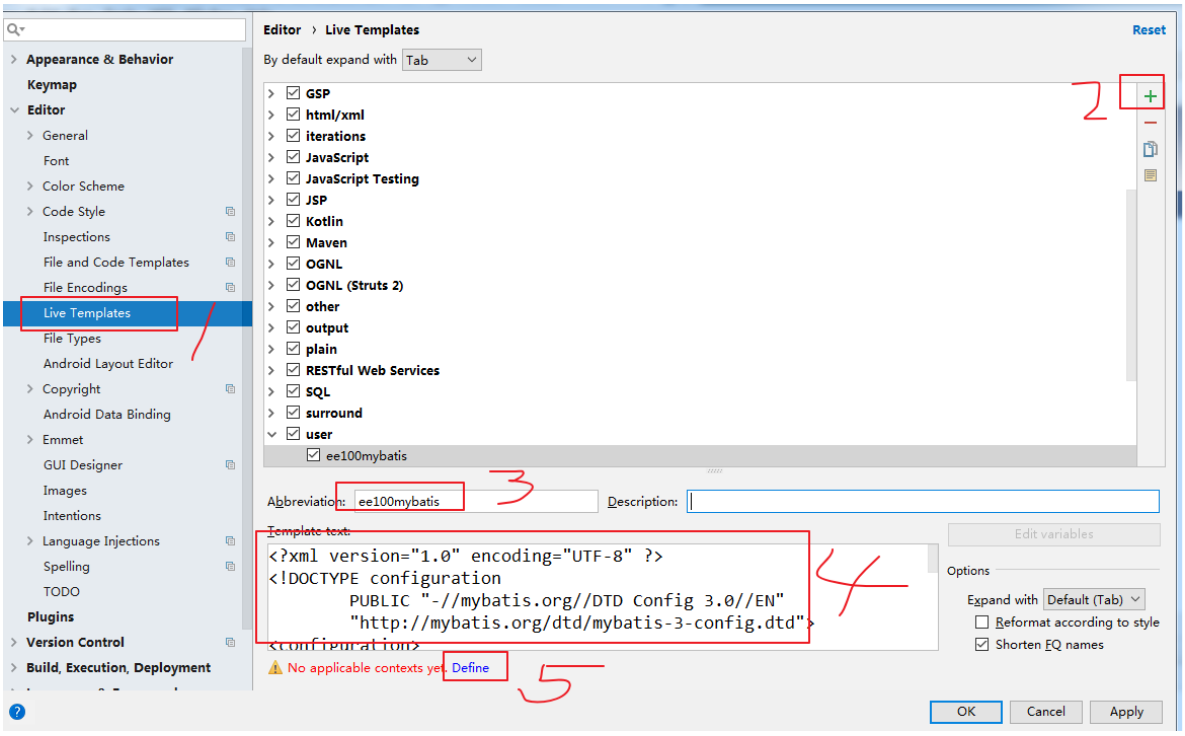
step7、删除：映射文件

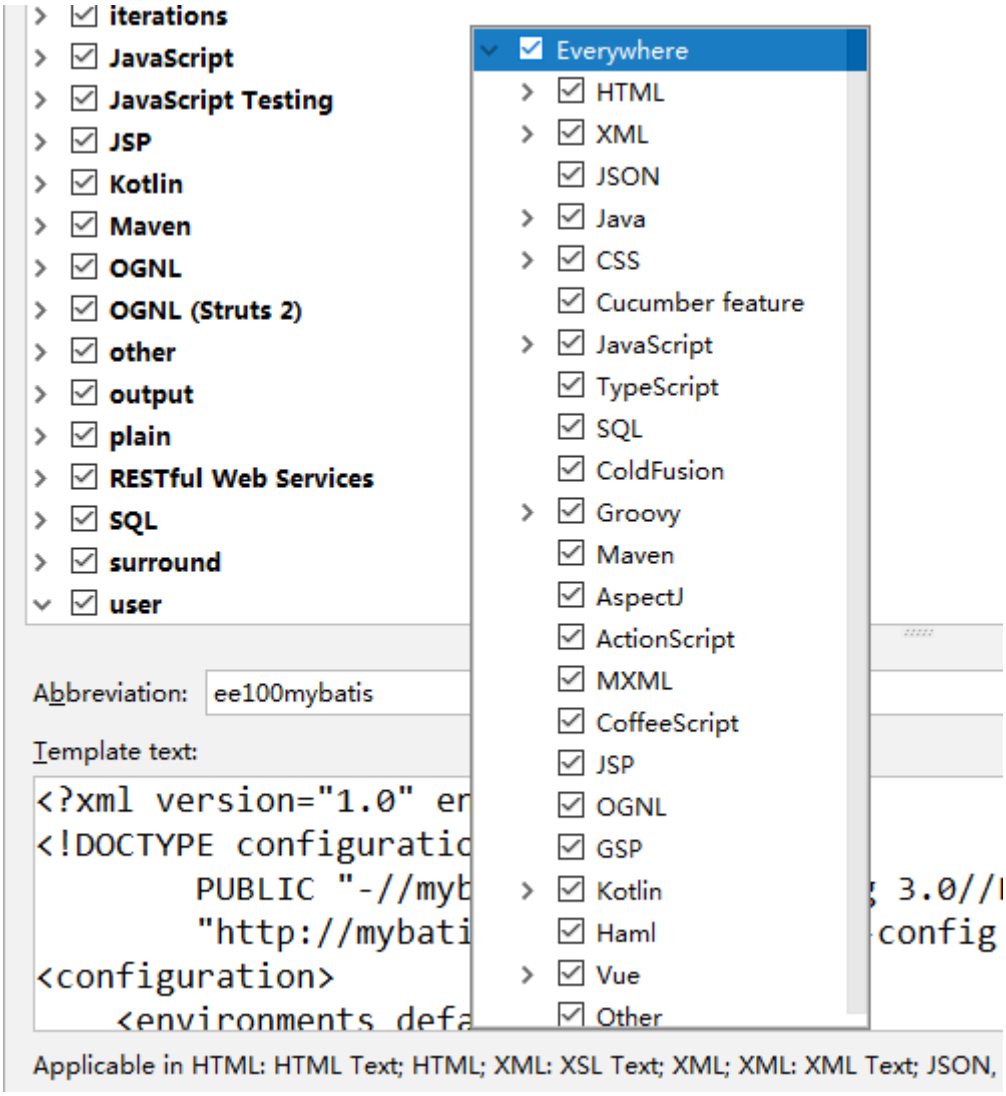
```xml
<!--如果参数只有一个且是简单（基本+String）类型，
    占位符中的内容可以随便写
-->
<delete id="deleteUser" parameterType="int">
    delete from users where uid=#{uid}
</delete>
```

step8：删除：测试

```java
@Test
public void delUser(){
    UserDao userDao = sqlSession.getMapper(UserDao.class);
    userDao.deleteUser( uid: 2);
    sqlSession.commit();
}
```

10. 配置模板





11. 利用mybatis完成简单的查询操作(重点)

11.1. 查询所有的记录

step1：编写dao接口方法

```
public interface UserDao {

    List<User> findAllUsers();
```

step2：编写映射文件

```
<!--
resultType:指定结果集要映射到的类型
-->
<select id="findAllUsers" resultType="com.itheima.domain.User">
    select * from users
</select>
```

不能省略

step3：测试类

```
@Test
public void findAll(){
    UserDao userDao = sqlSession.getMapper(UserDao.class);
    List<User> users = userDao.findAllUsers();
    for(User user:users)
        System.out.println(user);
}
}
```

UserDaoTest › delUser()

1 test passed - 1s 605ms

```
TRACE [main] - <==    Columns: uid, name, password
TRACE [main] - <==        Row: 4, admin, 123
TRACE [main] - <==        Row: 5, admin, 123
TRACE [main] - <==        Row: 6, 陈云婷, 123
DEBUG [main] - <==      Total: 3
User{uid=4, name='admin', password='123'}
User{uid=5, name='admin', password='123'}
User{uid=6, name='陈云婷', password='123'}
DEBUG [main] - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JI
```

11.2. 根据一个参数查询记录

step1：dao接口

```
User findUserByUid(Integer uid);
```

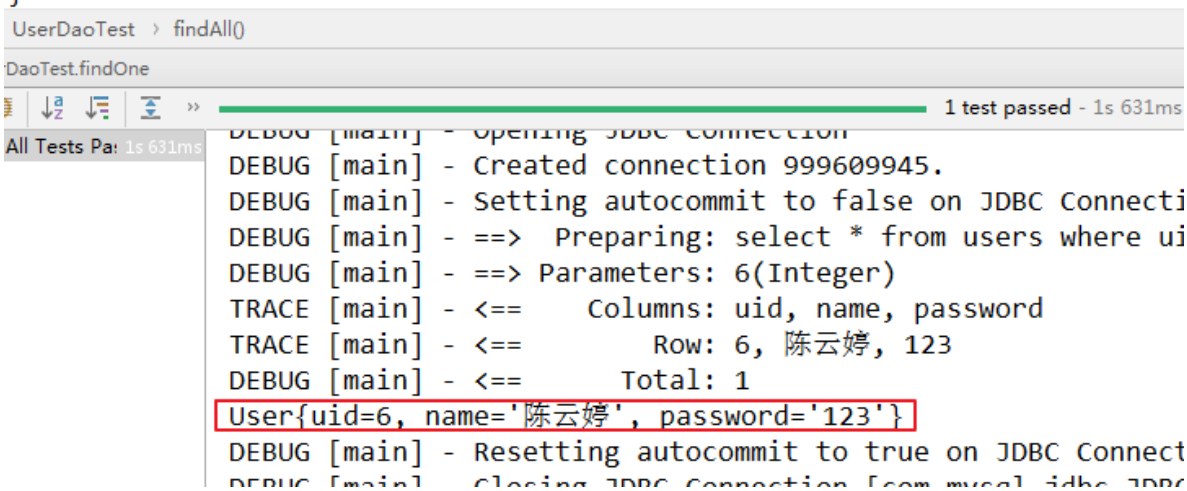
step2：映射文件

```
<select id="findUserByUid" resultType="com.itheima.domain.User">
    select * from users where uid=#{uid}
</select>

<!--
```

只有一个简单类型的参数，随便写

step3：测试

```
    @Test
    public void findOne(){
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        User user = userDao.findUserByUid(6);
        System.out.println(user);
    }
}
```



11.3. 根据多个参数查询记录

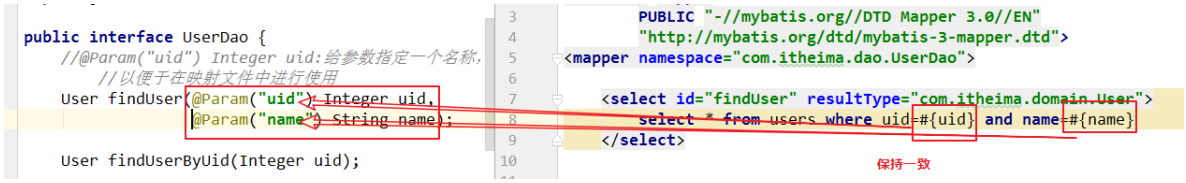
step1：编写dao接口

```
User findUser(Integer uid,String name);
```

step2：编写映射文件

```
方式一：使用固定的arg0：代表第一个参数；arg1：代表第二个参数
    <select id="findUser" resultType="com.itheima.domain.User">
        select * from users where uid=#{arg0} and name=#{arg1}
    </select>
方式二：使用固定的param1：代表第一个参数；param2：代表第二个参数
    <select id="findUser" resultType="com.itheima.domain.User">
        select * from users where uid=#{param1} and name=#{param2}
    </select>
```

推荐方式三：使用@Param注解来指定具体的名称



step3：测试

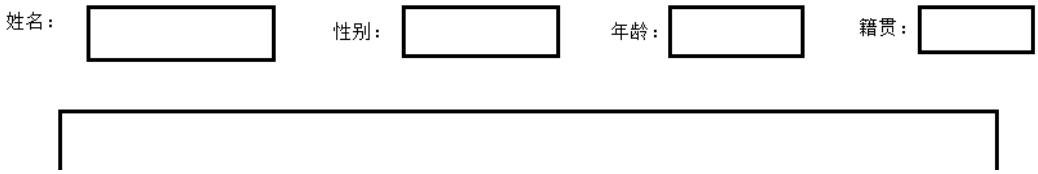
```
    @Test
    public void findOne1(){
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        User user = userDao.findUser( uid: 6, name: "陈云婷");
        System.out.println(user);
    }
```

11.4. 根据POJO参数查询记录

step1：使用场景

find(String name,String gender,String age,String address):一般参数不超过3个
find(User user):把查询条件封装到User对象中，对象中哪些才是条件：不为null的就是条件。
这就是为什么User中的id用Integer的原因。

多条件查询时



step2：dao接口

```
User findUser1(User condition);
```

step3：映射文件

```
<select id="findUser1" parameterType="com.itheima.domain.User"
        resultType="com.itheima.domain.User">
    select * from users where uid=#{uid} and name=#{name}
</select>
```

属性

step4：测试

```
@Test
public void findUser1(){
    User condition = new User();
    condition.setUid(6);
    condition.setName("陈云婷");

    UserDao userDao = sqlSession.getMapper(UserDao.class);
    User user = userDao.findUser1(condition);
    System.out.println(user);
}
```

封装查询条件

11.5. 模糊查询（重点，难点）

step1：dao接口

```
public interface UserDao {
    //支持按照名字模糊查询
    List<User> findUsers(String name);
```

step2：映射文件

方式一：仅用于mysql数据库

```xml
<select id="findUsers" resultType="com.itheima.domain.User">
    select * from users where name like concat('%',#{name},'%')
</select>
```

占位符

concat是mysql中的字符串拼接函数

方式一：测试

```java
@Test
public void findAll1(){
    UserDao userDao = sqlSession.getMapper(UserDao.class);
    List<User> users = userDao.findUsers( name: "a");
    for(User user:users)
        System.out.println(user);
}
```

```
UserDaoTest › findAll1()
aoTest.findAll1
1 test passed - 1s 445ms
ll Tests Pa: 1s 445ms
DEBUG [main] - <==      Total: 2
User{uid=4, name='admin', password='123'}
User{uid=5, name='admin', password='123'}
DEBUG [main] - Resetting autocommit to true on JDBC Connec
DEBUG [main] - Closing JDBC Connection [com.mysql.jdbc.JDB
```

方式二：测试

```java
@Test
public void findAll1(){
    UserDao userDao = sqlSession.getMapper(UserDao.class);
    List<User> users = userDao.findUsers( name: "%a%");
    for(User user:users)
        System.out.println(user);
}
```

用的时候传入%

```xml
<select id="findUsers" resultType="com.itheima.domain.User">
    select * from users where name like #{name}
</select>

<select id="findUser1" parameterType="com.itheima.domain.User
```

方式三：测试

字符串拼接，使用默认参数名：

```java
@Test
public void findAll1(){
    UserDao userDao = sqlSession.getMapper(UserDao.cla
    List<User> users = userDao.findUsers( name: "a");
    for(User user:users)
        System.out.println(user);
}
```

```xml
<select id="findUsers" resultType="com.itheima.domain.User">
    select * from users where name like '%${value}%'
</select>
```

${}:在mybatis的xml中使用时，代表的字符串拼接
${value}:因为只有一个简单类型的参数，必须使用固定值value

字符串拼接，使用自己制定的参数名：

```java
ublic interface UserDao {
    //支持按照名字模糊查询
    List<User> findUsers(@Param("name") String name);

    User findUser1(User condition);
```

```xml
<select id="findUsers" resultType="com.itheima.domain.User">
    select * from users where name like '%${name}%'
</select>
```

12. 面试题：

问：mybatis的映射文件中，#{}和${}有什么区别？

答：#{}代表占位符。能够防止sql注入

${}代表字符串拼接。有sql注入的危险。