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

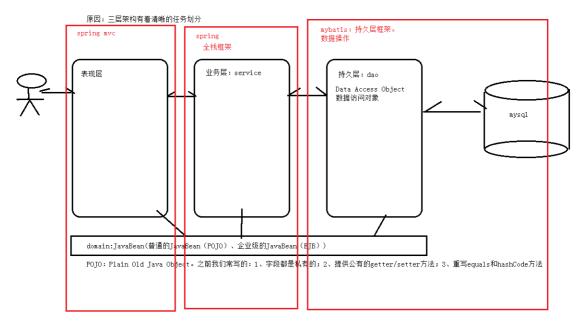
1.1. 三层架构

表现层: spring mvc。struts1、struts2.

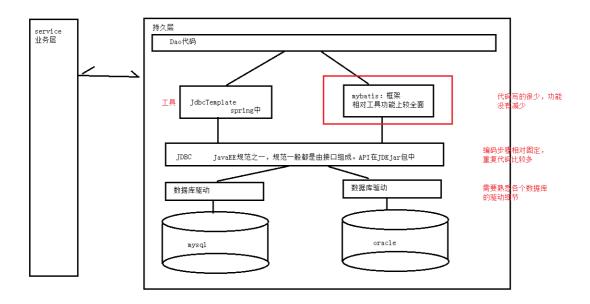
业务层:spring。很多企业都在使用。全栈框架(各层都有)

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

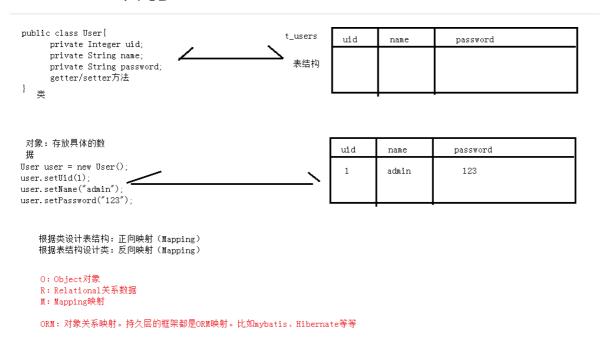


工具也可以叫做框架,只是功能相对较小,所有工具的称为比较合适。功能比较全面的叫做框架。

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



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

```
aday46_mybatis_01quickStart > src > test > java
Project

O v day46
                   ▼ ③ ‡ 🕸 🗠 m day46_mybatis_01quickStart ×

    laga day46_mybatis_01quickStart D:\IdeaProj

                                           <?xml version="1.0" encoding="UTF-8"?>
    ∨ III src
                                           cproject xmlns="http://maven.apache.org/POM/4.0.0"
      ∨ 🖿 main
                                                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           iava java
                                                     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://i
           resources
                                                <modelVersion>4.0.0</modelVersion>
                                                <groupId>com.itheima.ee100
      aday46_mybatis_01quickStart.im
                                    8
                                                <artifactId>day46_mybatis_01quickStart</artifactId>
      m pom.xml
                                                <version>1.0-SNAPSHOT</version>
  > |||| External Libraries
                                   10
                                           </project>
```

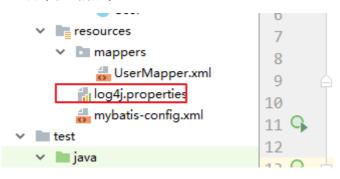
step2:导入jar包(pom.xml)

```
<dependencies>
 2
           <!--mybatis的jar包-->
 3
           <dependency>
 4
               <groupId>org.mybatis
               <artifactId>mybatis</artifactId>
 6
               <version>3.5.0
           </dependency>
           <!--数据库驱动-->
8
9
           <dependency>
10
               <groupId>mysql
11
               <artifactId>mysql-connector-java</artifactId>
               <version>5.1.47</version>
12
13
           </dependency>
           <!--单元测试-->
14
15
           <dependency>
               <groupId>junit
16
               <artifactId>junit</artifactId>
18
               <version>4.12</version>
19
               <scope>test</scope>
20
           </dependency>
           <!-- 日志: 学习期间可以打印一些内容在控制台。
21
           便于理解和学习-->
23
           <dependency>
               <groupId>log4j
24
25
               <artifactId>log4j</artifactId>
26
               <version>1.2.17
           </dependency>
27
28
       </dependencies>
```

```
/46_mybatis_01quickStart > lim src > lim main > lim resources
               ▼ 😌 ‡ 👫 I<sup>←</sup> M day46_mybatis_01quickStart × 🚜 mybatis-config.xml ×
ay46_mybatis_01quickStart D:\IdeaProj
                                     <?xml version="1.0" encoding="UTF-8" ?>
∨ IIII src
                                                - 下: xml 约束的引入-->
  ∨ I main
                                     <!DOCTYPE configuration
                                                                                                    引入约束
      java
                                              PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
                                              "http://mybatis.org/dtd/mybatis-3-config.dtd">
                               6
                                     <configuration>
                                          <1-- 配置环境-->
      iava java
                                          <environments default="mysql"</pre>
                               8
  aday46_mybatis_01quickStart.iml
                                              <environment id="mysql">
                                                                                           事务管理器。用JDBC的
  m pom.xml
                                                                               "JDBC"/> 事务管理
                              10
                                                 <transactionManager type=</pre>
External Libraries
                                                                                                    连接池 , mybatis内置的
                                                   <dataSource type="POOLED">
                                                       cproperty name="driver" value="${driver}"/>
                                                       cproperty name="url" value="${url}"/>
                                                       cproperty name="username" value="${username}"/>
                              14
                                                       cproperty name="password" value="${password}"/>
                              15
                                                   </dataSource>
                              17
                                               </environment>
                              18
                                          </environments>
                              19
                                          <mappers>
                                              <mapper resource="org/mybatis/example/BlogMapper.xml"/>
                              20
                                          </mappers>
                                     </configuration>
         <?xml version="1.0" encoding="UTF-8" ?>
```

```
2
    <!--复习一下: xml约束的引入-->
    <!DOCTYPE configuration
 3
4
            PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
            "http://mybatis.org/dtd/mybatis-3-config.dtd">
 5
    <configuration>
        <!--配置环境-->
 7
        <environments default="mysql">
8
9
            <environment id="mysql">
                <transactionManager type="JDBC"/>
10
                 <dataSource type="POOLED">
11
12
                    cproperty name="driver" value="com.mysql.jdbc.Driver"/>
                     cproperty name="url" value="jdbc:mysql:///ee100"/>
13
                    cproperty name="username" value="root"/>
14
15
                     cproperty name="password" value="sorry"/>
                 </dataSource>
16
17
            </environment>
        </environments>
18
19
    </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 🗵
1
       package com.itheima.domain;
2
3
       public class User {
4
            private Integer uid;
5
            private String name;
            private String password;
6
7
8
            public Integer getUid() {
9
                 return uid;
0
1
2
            public void setUid(Integer uid) {
3
                 this.uid = uid;
4
5
            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
 public class User {
    private Integer uid; -
    private String name;
                                                    开发中一般字段名和属性保持一致
    private String password;
    public Integer getId() {
       return uid;
                                      属性:指getter/setter方法。
                                          属性名:去掉get和set,首字母小写。
    public void setId(Integer uid) {
                                      getId():id。读属性
      this.uid = uid;
                                      setId():id。写属性
    public String getName() {
    create database ee100;
 2
    use ee100;
 3
    create table users(
 4
       uid int primary key auto_increment,
 5
       name varchar(100),
 6
       password varchar(100)
    );
```

step3:编写映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>
 2
    <!DOCTYPE mapper
 3
           PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
           "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5
    <!--namespace类似java中的package-->
6
    <mapper namespace="usermapper">
       <!--id: 当前文件中要唯一。
           parameterType:存放的数据的参数类型
8
           sql语句:写在标签内部
9
10
11
       <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

```
mybatis-config.xml × C User.java × 🐉 UserMapper.xml >
day46_mybatis_01quickStart ×
     <?xml version="1.0" encoding="UTF-8" ?>
     <!--复习一下: xml约束的引入-->
     <!DOCTYPE configuration</pre>
              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">
                       cproperty name="driver" value="com.mysql.jdbc.Driver"/>
                       cproperty name="url" value="jdbc:mysql:///ee100"/>
                       cproperty name="username" value="root"/>
                       cproperty name="password" value="sorry"/>
                   </dataSource>
              </environment>
          </environments>
          <!--注册映射文件-->
          <mappers>
              <mapper resource="mappers\UserMapper.xml"/>
          </mappers>
     </body></bre>
step5、编写测试类
public class useriest {
   public void testAddUser() throws Exception{//学习方法: 记
      //加载mybatis的主配置文
      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
      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();
      salSession.insert(s: "usermapper.addUser");//insert(String s):不是sql语句,而是配置文件中的namespace+id
      sqlSession.commit();
                           不是自动提交事务的
      sqlSession.close();
```

5. mybatis基于Dao实现类的编码方式(了解)

- b、更改模块的名称
- c、删除多余的iml文件
- d、删除target目录

```
day46_mybatis_02daoImpl ) m pom.xml
             <u>+ ⊕ ‡ | ‡</u>- ⊩
                         m day46_mybatis_02daoImpl >
> 📭 day46_mybatis_01quickStart D:\Idea
                               <?xml version="1.0" encoding="UTF-8"?>
 aPr
                               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   🚛 day46_mybatis_02daoImpl.iml
                          4
                                        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   m pom.xmi
                          5
                                   <modelVersion>4.0.0</modelVersion>
> || External Libraries
                         6
                                   <groupId>com.itheima.ee100
                         8
                                   <artifactid>day46_mybatis_02daoImpl/artifactId>
                         9
                                   <version>1.0-SNAPSHOT
                                   <dependencies>
                         11
                                       <!--mybatis的jar包-->
                                       <dependency>
                         13
                                           <groupId>org.mybatis
                         14
                                           <artifactId>mybatis</artifactId>
                         15
                                           <version>3.5.0
                                       </dependency>
                         16
```

step2:编写一个dao的接口

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

step3:编写dao的实现

```
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:根据配置文件获现
           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();
           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/
           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

```
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 🗵
       package com.itheima.util;
1
3
       import org.apache.ibatis.io.Resources;
4
       import org.apache.ibatis.session.SqlSession;
5
       import org.apache.ibatis.session.SqlSessionFactory;
       import org.apache.ibatis.session.SqlSessionFactoryBuilder;
6
7
8
       import java.io.IOException;
9
       import java.io.InputStream;
10
11
       //和SqlSessionFactory和SqlSession有关
       public class MyBatisUtil {
12
13
           private static SqlSessionFactory sqlSessionFactory;
14
           static{
15
16
                   InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
17
                   sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);
               } catch (IOException e) {
19
                   e.printStackTrace();
20
21
22
           public SqlSession openSession(){
23
               return sqlSessionFactory.openSession();
24
25
       }
```

8.2. 抽取一个单元测试的基类 (Junit)

```
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();
    }
}

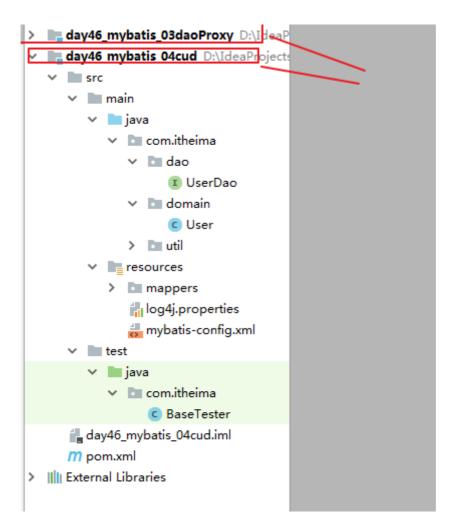
使用:

public class UserTest extends BaseTester {
        OTest
        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

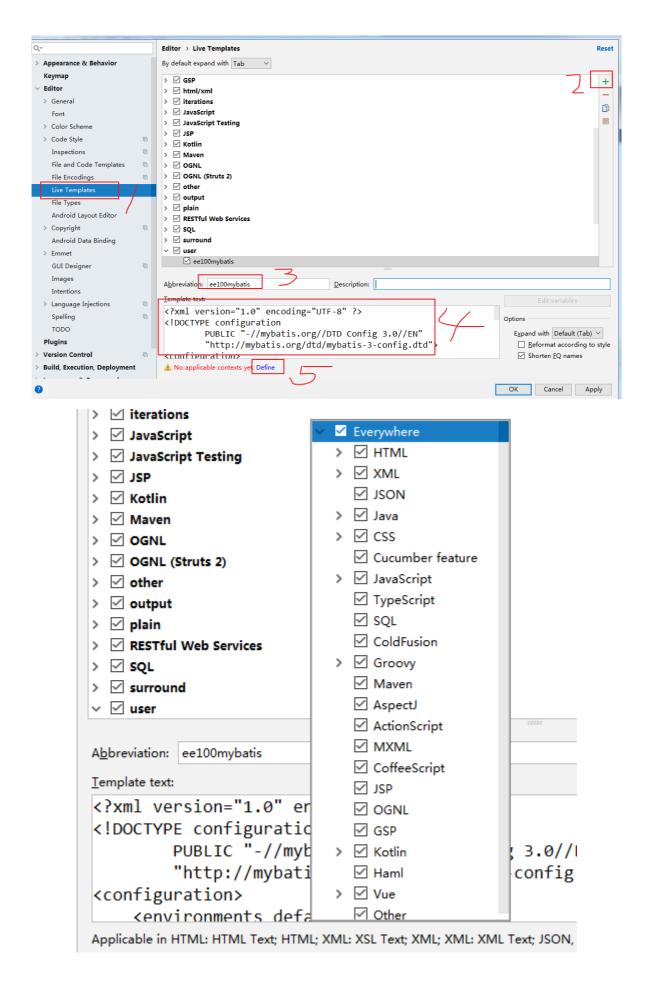
```
public interface UserDao {
    void saveUser(User user);
    void updateUser(User user);
    void deleteUser(Integer uid);
}
```

step3:添加:编写映射文件

step4:添加:测试

```
PADER CEASE OSCIDAOTOSC CACCINAS DASCICISCO |
   @Test
                                                      4 rows in set
   public void saveUser(){
      User user = new User();
                                                      mysql> select * from users;
      user.setName("陈云婷");
      user.setPassword("123");
                                                       uid name
                                                                    password
      UserDao userDao = sqlSession.getMapper(UserDao.class);
      userDao.saveUser(user);
                                                            admin
                                                                    123
      sqlSession.commit();
                                                            admin
                                                                    123
                                                            admin
                                                                    123
   }
                                                                  123
UserDaoTest → saveUser()
step5:修改:编写映射文件
    <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
                                                           aumin
admin
                                                        4
                                                                  123
  @Test
  public void updateUser(){
                                                        5
                                                        6
                                                           陈云婷
                                                                  123
    User user = new User();
     user.setUid(2);
                                                     5 rows in set
    user.setName("王猛");
     user.setPassword("1234");
                                                     mysql> select * from users;
     UserDao userDao = sqlSession.getMapper(UserDao.class);
                                                      uid name
                                                                  password
     userDao.updateUser(user);
     sqlSession.commit();
                                                           admin
                                                           admin
                                                                  123
                                                        6
                                                           陈云婷
                                                                  123
step7、删除:映射文件
  <!---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
    @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
                                                          不能省略
     <select id="findAllUsers" resultType="com.itheima.domain.User'</pre>
          select * from users
     </select>
step3:测试类
                         @lest
             41 🗣 🖯
                         public void findAll(){
             42
                              UserDao userDao = sqlSession.getMapper(UserDao.class);
                             List<User> users = userDao.findAllUsers();
             43
             44
                              for(User user:users)
             45
                                  System.out.println(user);
             46
                         }
             47
             48
per.xml
                     UserDaoTest > delUser()

    1 test passed - 1s 605ms

 DEDOO [Mail] - --> Falametels.
 TRACE [main] - <== Columns: uid, name, password

      TRACE [main] - <==</td>
      Row: 4, admin, 123

      TRACE [main] - <==</td>
      Row: 5, admin, 123

      TRACE [main] - <==</td>
      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
                 -个参数查询记录
11.2. 根据·
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

```
public void findOne(){
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        User user = userDao.findUserByUid(6);
        System.out.println(user);
UserDaoTest > findAll()
DaoTest.findOne
1 test passed - 1s 631ms
              DEDOG [MAIL] - ObeLITUR SODE COLLIECTION
All Tests Par 1s
              DEBUG [main] - Created connection 999609945.
              DEBUG [main] - Setting autocommit to false on JDBC Connecti
              DEBUG [main] - ==> Preparing: select * from users where ui
              DEBUG [main] - ==> Parameters: 6(Integer)
              TRACE [main] - <==
                                     Columns: uid, name, password
                                          Row: 6, 陈云婷, 123
              TRACE [main] - <==
              DEBUG [main] - <==
                                        Total: 1
             User{uid=6, name='陈云婷', password='123'}
              DEBUG [main] - Resetting autocommit to true on JDBC Connect
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
  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
                                         PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
 public interface UserDao {
                                          'http://mybatis.org/dtd/mybatis-3-mapper.dtd">
   //@Param("uid") Integer uid:给参数指定一个名称,
                                     <mapper namespace="com.itheima.dao.UserDao";</pre>
      //以便于在<u>映射文件中进行使用</u>
                                       <select id="findUser" resultType="com.itheima.domain.User">
    select * from users where uid #{uid} and name ##{name}
   User findUser @Param("uid" Integer uid,
           @Param("name") String name
   User findUserByUid(Integer uid);
step3:测试
   @Test
   public void findOne1(){
        UserDao userDao = salSession.getMapper(UserDao.class);
        User user = userDao.findUser( uid: 6, name: "陈云婷");
        System.out.println(user);
```

wiest

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。 多条件查询时 这就是为什么User中的id用Integer的原因。 姓名: 籍贯: 性别: 年龄: step2:dao接口 User findUser1(User condition); step3:映射文件 <select id="findUser1" parameterType="com.itheima.domain.User"</pre> resultType="com.itheima.domain.User"> select * from users where uid=#{uid} and name=#{name step4:测试 @Test public void findUser1(){ User condition = new User(); 封装查询条件 condition.setUid(6); condition.setName("陈云婷"); UserDao userDao = sqlSession.getMapper(UserDao.class); User user = userDao.findUser1(condition); System.out.println(user); 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);

方式一: 仅用于mysql数据库

step2:映射文件

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

</select>

concat是mysql中的字符串拼接函数
```

方式一:测试

```
@Test
    public void findAll1(){
          UserDao userDao = sqlSession.getMapper(UserDao.class);
          List<User> users = userDao.findUsers( name: "a");
                                                                                                     16
          for(User user:users)
                                                                                                     11
                System.out.println(user);
                                                                                                     12
                                                                                                     13
                                                                                                     14
                                                                                                     15
                                                                                                     16
                                                                                                     17
                                                                                                     18
                                                                                                     19
                                                                                                     20
JserDaoTest > findAll1()
aoTest.findAll1
 ↓2 ↓E | <u>₹</u>
                                                                                  1 test passed - 1s 445ms
                                                     Total: 2
                  DEBUG [main] - <==
ll Tests Pa: 1s 445ms
                  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.mvsal.idbc.JDB)
方式二:测试
                                                           <select id="findUsers" resultType="com.itheima.domain.User">
  public void findAll1(){
   UserDao userDao = sqlSession.getMapper(UserDao.class
   List<User> users = userDao.findUsers( name: "%a%");
                                                             select * from users where name like #{name}
                                                  9
        System.out.println(user);
                                                          <select id="findUser1" parameterType="com.itheima.domain.User</pre>
```

方式三:测试

字符串拼接,使用默认参数名:

User findUser1(User condition);

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。