用户登录实现

1.1 规划需要执行的 SQL 语句

用户登录功能需要执行的 SQL 语句是根据用户名查询用户数据，再判断密码是否正确。SQL 语句大致是：

说明：以上 SQL 语句对应的后台开发已经完成，无需再次开发。

1.2 接口与抽象方法

说明：无需再次开发。

1.3 配置 SQL 映射

说明：无需再次开发。

2 用户-登录-业务层

2.1 规划异常

1.如果用户名不存在则登录失败，抛出 com.cy.store.service.ex.UserNotFoundException 异常，并从父类生成子类的五个构造方法。

```
package com.fx.store.service.ex;

/**
 * 用户数据不存在异常
 */
public class UserNotFoundException extends ServiceException{
    public UserNotFoundException() {
        super();
    }

    public UserNotFoundException(String message) {
        super(message);
    }

    public UserNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }

    public UserNotFoundException(Throwable cause) {
        super(cause);
    }

    protected UserNotFoundException(String message, Throwable cause, boolean
```

```
enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

2.如果用户的 isDelete 字段的值为 1，则表示当前用户数据被标记为“已删除”，需进行登录失败操作同时抛出 UserNotFoundException。

3. 如果密码错误则进行登录失败操作，同时抛出 com.cy.store.service.ex.PasswordNotMatchException 异常。

```
package com.fx.store.service.ex;

/**
 * 密码验证失败的异常
 */
public class PasswordNotMatchException extends ServiceException{
    public PasswordNotMatchException() {
        super();
    }

    public PasswordNotMatchException(String message) {
        super(message);
    }

    public PasswordNotMatchException(String message, Throwable cause) {
        super(message, cause);
    }

    public PasswordNotMatchException(Throwable cause) {
        super(cause);
    }

    protected PasswordNotMatchException(String message, Throwable cause, boolean
enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

4.创建以上 UserNotFoundException 和 PasswordNotMatchException 异常类，以上异常类应继承自 ServiceException 类。

2.2 接口与抽象方法

在 IUserService 接口中添加登录功能的抽象方法。

```
/**
 * 用户登录方法
 * @param username 用户名
 * @param password 密码
 * @return 登录成功的用户数据
 */
User login(String username,String password);
```

当登录成功后需要获取该用户的 id，以便于后续识别该用户的身份，并且还需要获取该用户的用户名、头像等数据，用于显示在软件的界面中，需使用可以封装用于 id、用户名和头像的数据的类型来作为登录方法的返回值类型。

2.3 实现抽象方法

1.在 UserServiceImpl 类中添加 login(String username, String password)方法并分析业务逻辑。
2.login(String username, String password)方法中代码的具体实现。

```
@Override
public User login(String username, String password) {
    //根据用户名称来查询用户的数据是否存在，如果不存在则抛出异常
    User result = userMapper.findByUserName(username);
    //判断用户查询结果是否 null，是，抛出 UserNotFoundException 异常
    if (result==null){
        throw new UserNotFoundException("用户数据不存在的错误");
    }
    //判断用户查询结果中的 isDelete 是否为 1，是，抛出 UserNotFoundException
    if (result.getIsDelete()==1){
        throw new UserNotFoundException("用户数据不存在的错误");
    }
    //判断密码是否输入错误
    //1、从查询结果中获取盐值
    String salt = result.getSalt();
    //调用 getMd5Password（）方法，将参数 password 和 salt 结合起来进行加密
    String newMd5Password = getMd5Password(password, salt);
    //判断查询结果中的密码，与以上加密得到的密码是否一致
    if (!result.getPassword().equals(newMd5Password)){
        throw new PasswordNotMatchException("密码验证失败的错误");
    }
    //创建新的 user 对象
    User user = new User();
    //将查询结果中的 uid、username、avata 封装到新的 user 对象中，提升系统性能
    user.setUid(result.getUid());
    user.setUsername(result.getUsername());
    user.setAvatar(result.getAvatar());
```

```
    //返回新的 user 对象
    return user;
}
```

3.完成后在 UserServiceTests 中编写并完成单元测试。

```
@Test
public void login(){
    try {
        String username = "lower02";
        String password = "123";
        User user = iUserService.login(username, password);
        System.out.println("登录成功！ ");
    } catch (ServiceException e) {
        System.out.println("登录失败！ "+e.getClass().getSimpleName());
        System.out.println(e.getMessage());
    }
```

用户-登录-控制器

3.1 处理异常

处理用户登录功能时，在业务层抛出了 UserNotFoundException 和 PasswordNotMatchException 异常，而这两个异常均未被处理过。则应在 BaseController 类的处理异常的方法中，添加这两个分支进行处理。

```
else if(e instanceof UserNotFoundException){
    result.setState(4001);
    result.setMessage("用户数据不存在");
}else if (e instanceof PasswordNotMatchException){
    result.setState(4002);
    result.setMessage("密码错误！ ");
}
```

3.2 设计请求

设计用户提交的请求，并设计响应的方式：

请求路径：/users/login

请求参数：String username, String password

请求类型：POST

响应结果：JsonResult<User>

3.3 处理请求

1.在 UserController 类中添加处理登录请求的 login(String username, String password)方法。

```
@RequestMapping("login")
public JsonResult<User> login(String username,String password){
    //调用业务对象的方法执行登录，并获取返回值
    User data = iUserService.login(username, password);
    //将以上返回值和状态码 ok 封装到响应结果中并返回
    return new JsonResult<User>(ok,data);
}
```

2.完成后启动项目，访问 http://localhost:8080/users/login?username=Tom&password=1234 请求进行登录。

4 用户-登录-前端页面

1.在 login.html 页面中 body 标签内部的最后，添加 script 标签用于编写 JavaScript 程序。\

```
<script type="text/javascript">
    $("#btn-login").click(function () {
        $.ajax({
            url:"/users/login",
            type:"POST",
            data:$("#form-login").serialize(),
            dataType:"json",
            success:function (json) {
                if (json.state==200){
                    alert("登录成功！ ");
                    location.href="index.html";
                }else {
                    alert("登录失败！ "+json.message);
                }
            }
        });
    });
</script>
```

2.完成后启动项目，打开浏览器访问 http://localhost:8080/web/login.html 页面并进行登录。