

# 一、用户中心（上）

---

- 完整了解做项目的思路，接触一些企业级的开发技术
- （尽量少写代码）目标：大家之后都能轻松做出管理系统

## 1.1.项目源码

---

- 用户中心前端项目源码：<http://gitlab.code-nav.cn/root/user-center-frontend>
- 用户中心后端项目源码：<http://gitlab.code-nav.cn/root/user-center-backend>

## 1.2.企业做项目流程

---

- 需求分析 => 设计（概要设计、详细设计）=> 技术选型 => 初始化 / 引入需要的技术 => 写 Demo  
=> 写代码（实现业务逻辑）=> 测试（单元测试）=> 代码提交 / 代码评审 => 部署=> 发布

## 1.3.需求分析

---

1. 登录 / 注册
2. 用户管理（仅管理员可见）对用户的查询或者修改
3. 用户校验（仅星球用户）

## 1.4.技术选型

---




- 前端：三件套 + React + 组件库 Ant Design + Umi + Ant Design Pro（现成的管理系统）
- 后端：
  - java
  - spring（依赖注入框架，帮助你管理 Java 对象，集成一些其他的内容）
  - springmvc（web 框架，提供接口访问、restful接口等能力）
  - mybatis（Java 操作数据库的框架，持久层框架，对 jdbc 的封装）
  - mybatis-plus（对 mybatis 的增强，不用写 sql 也能实现增删改查）
  - springboot（快速启动 / 快速集成项目。不用自己管理 spring 配置，不用自己整合各种框架）
  - junit 单元测试库
  - mysql
- 部署：服务器 / 容器（平台）

## 1.5.计划



---

### 1. 初始化项目

#### 1. 前端初始化 20 min

1. 初始化项目 ★
2. 引入一些组件之类的 ★
3. 框架介绍 / 瘦身 ★

#### 2. 后端初始化 20 min

1. 准备环境（MySQL 之类的）验证 MySQL 是否安装成功 - 连接一下 ★
2. 初始化后端项目，引入框架（整合框架） ★

### 2. 数据库设计

### 3. 登录 / 注册 20min

1. 前端

2. 后端

### 4. 用户管理（仅管理员可见） 20 min

1. 前端

2. 后端

## 1.6.前端初始化

---

### 1.6.1.下载并安装node.js

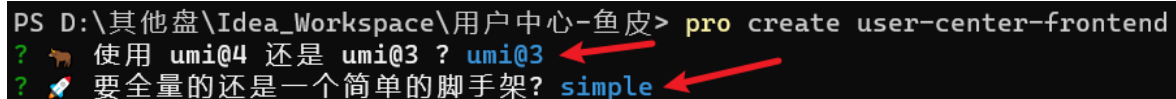
### 1.6.2.初始化[Ant Design Pro](#)脚手架

#### (1) 初始化前端项目

使用 npm

```
npm i @ant-design/pro-cli -g
```

```
pro create user-center-frontend
```



```
PS D:\其他盘\Idea_Workspace\用户中心-鱼皮> pro create user-center-frontend
? 🐾 使用 umi@4 还是 umi@3 ? umi@3
? 🚀 要全量的还是一个简单的脚手架? simple
```

注意：需要先安装 yarn

#### (2) 安装项目所需依赖包

```
yarn
```

#### (3) 安装 [Umi UI](#) (这个工具可以帮我们自动生成代码)

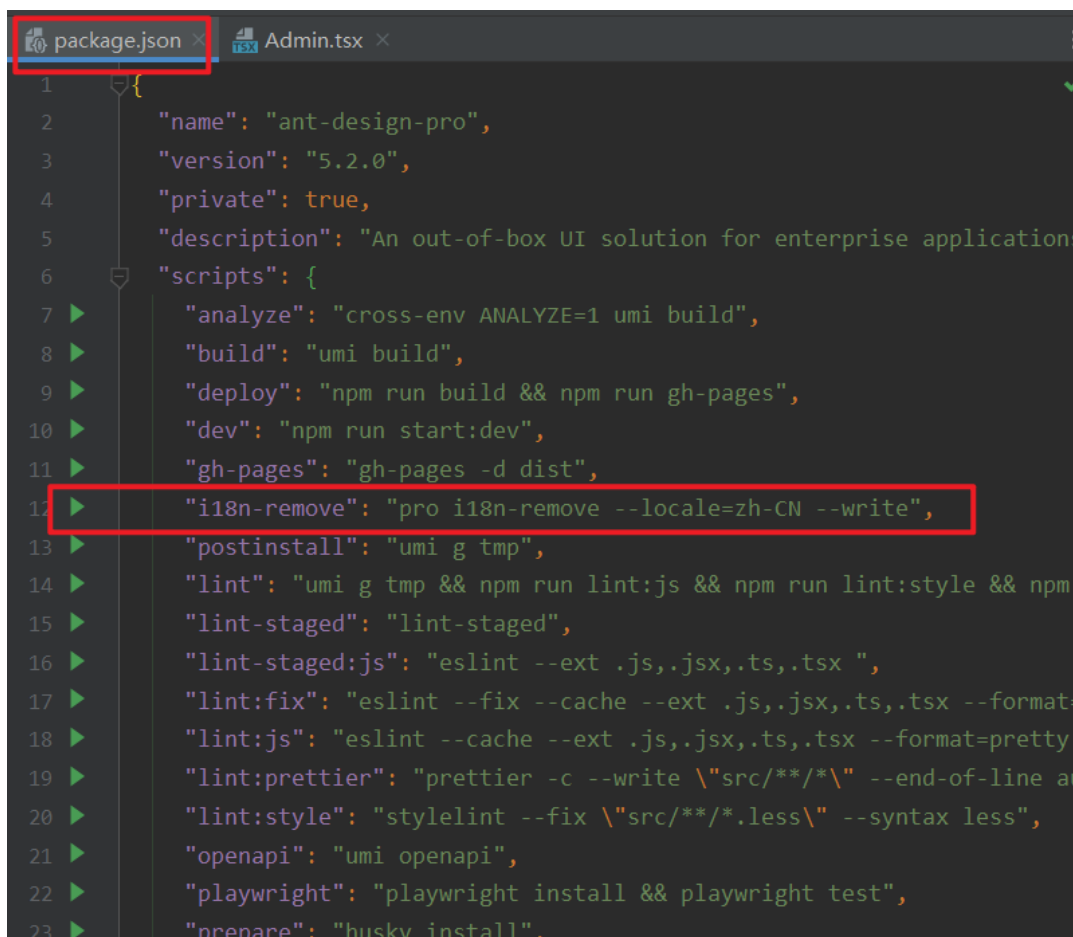
```
yarn add @umijs/preset-ui -D
```

- 使用该工具可以帮助我们快速生成页面

#### (4) 项目瘦身

##### 1. 删除国际化

- 执行 `i18n-remove` 脚本



```
1 {
2   "name": "ant-design-pro",
3   "version": "5.2.0",
4   "private": true,
5   "description": "An out-of-box UI solution for enterprise applications",
6   "scripts": {
7     "analyze": "cross-env ANALYZE=1 umi build",
8     "build": "umi build",
9     "deploy": "npm run build && npm run gh-pages",
10    "dev": "npm run start:dev",
11    "gh-pages": "gh-pages -d dist",
12    "i18n-remove": "pro i18n-remove --locale=zh-CN --write",
13    "postinstall": "umi g tmp",
14    "lint": "umi g tmp && npm run lint:js && npm run lint:style && npm",
15    "lint-staged": "lint-staged",
16    "lint-staged:js": "eslint --ext .js,.jsx,.ts,.tsx ",
17    "lint:fix": "eslint --fix --cache --ext .js,.jsx,.ts,.tsx --format=",
18    "lint:js": "eslint --cache --ext .js,.jsx,.ts,.tsx --format=pretty",
19    "lint:prettier": "prettier -c --write \"src/**/*\" --end-of-line al",
20    "lint:style": "stylelint --fix \"src/**/*.less\" --syntax less",
21    "openapi": "umi openapi",
22    "playwright": "playwright install && playwright test",
23    "prepare": "husky install",
```

- 删除项目路径下 src/locales 文件夹
- 2. 删除项目路径下 src/e2e 文件夹
  - e2e 文件夹里面定义的是一系列测试流程
- 3. 若前面有使用 umi ui 添加页面，可把该页面删除
  - 注意：需要在路径为 config/routes.ts 的文件夹下，删除对应的路由规则
- 4. 删除项目路径下 src/services/swagger 文件夹
  - swagger 文件夹里面定义了一系列后台接口程序
- 5. 删除项目路径下 config/oneapi.json 文件
  - oneapi.json 定义了整个项目用到的一些接口
- 6. 删除项目根路径下 tests 文件夹
  - tests 文件夹主要是和测试相关的
- 7. 删除项目根路径下 jest.config.js 文件
  - jest.config.js 测试相关的配置文件
- 8. 删除项目根路径下 playwright.config.ts 文件
  - playwright.config.ts -> 自动化测试工具，帮你在火狐或谷歌自动测试，不用真实地打开浏览器就能测试

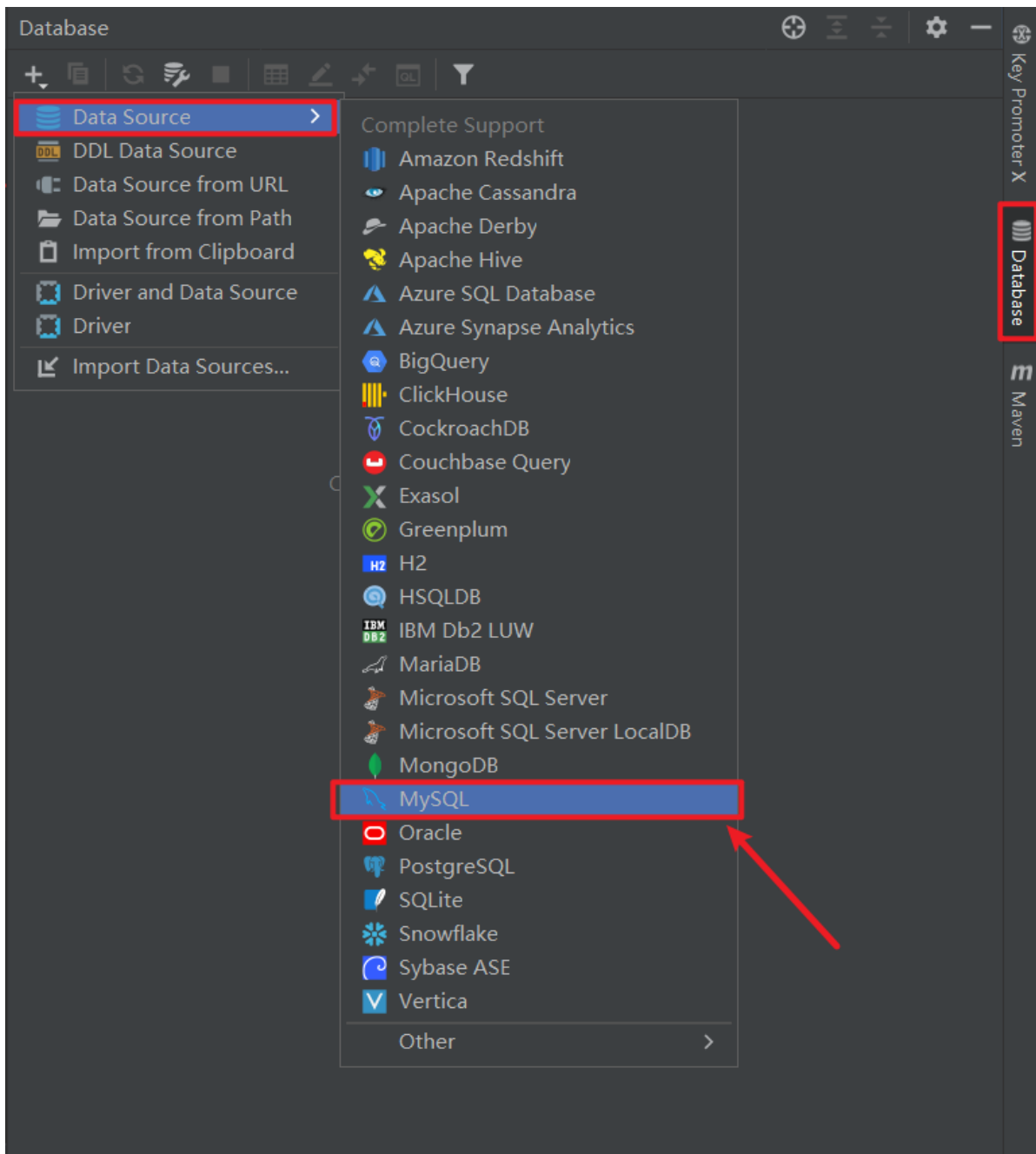
## 1.7.后端初始化

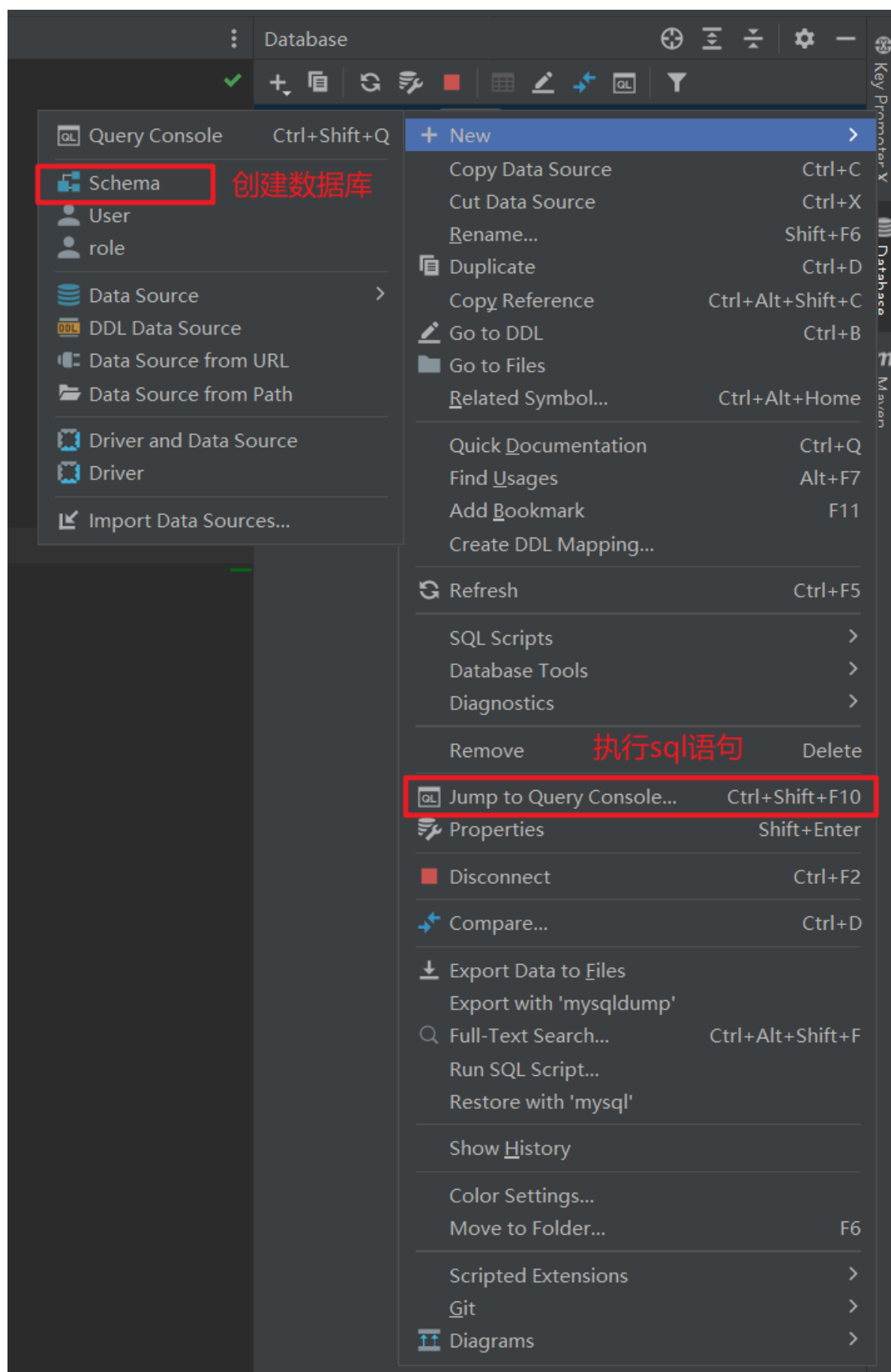
### 1.7.1.mysql的前期准备

(1) 下载并安装

- <https://dev.mysql.com/downloads/mysql/5.7.html>

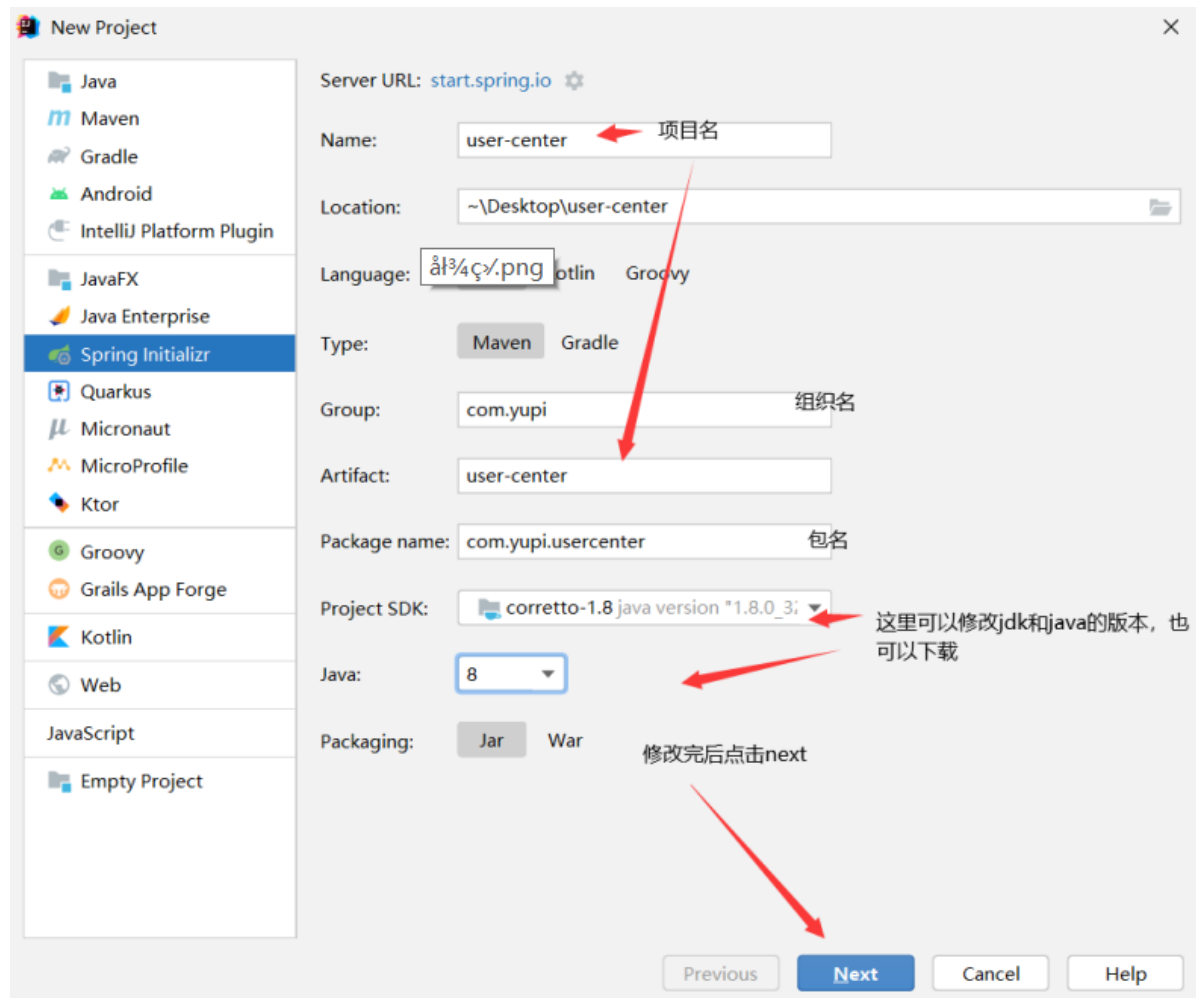
(2) 验证 MySQL 是否安装成功

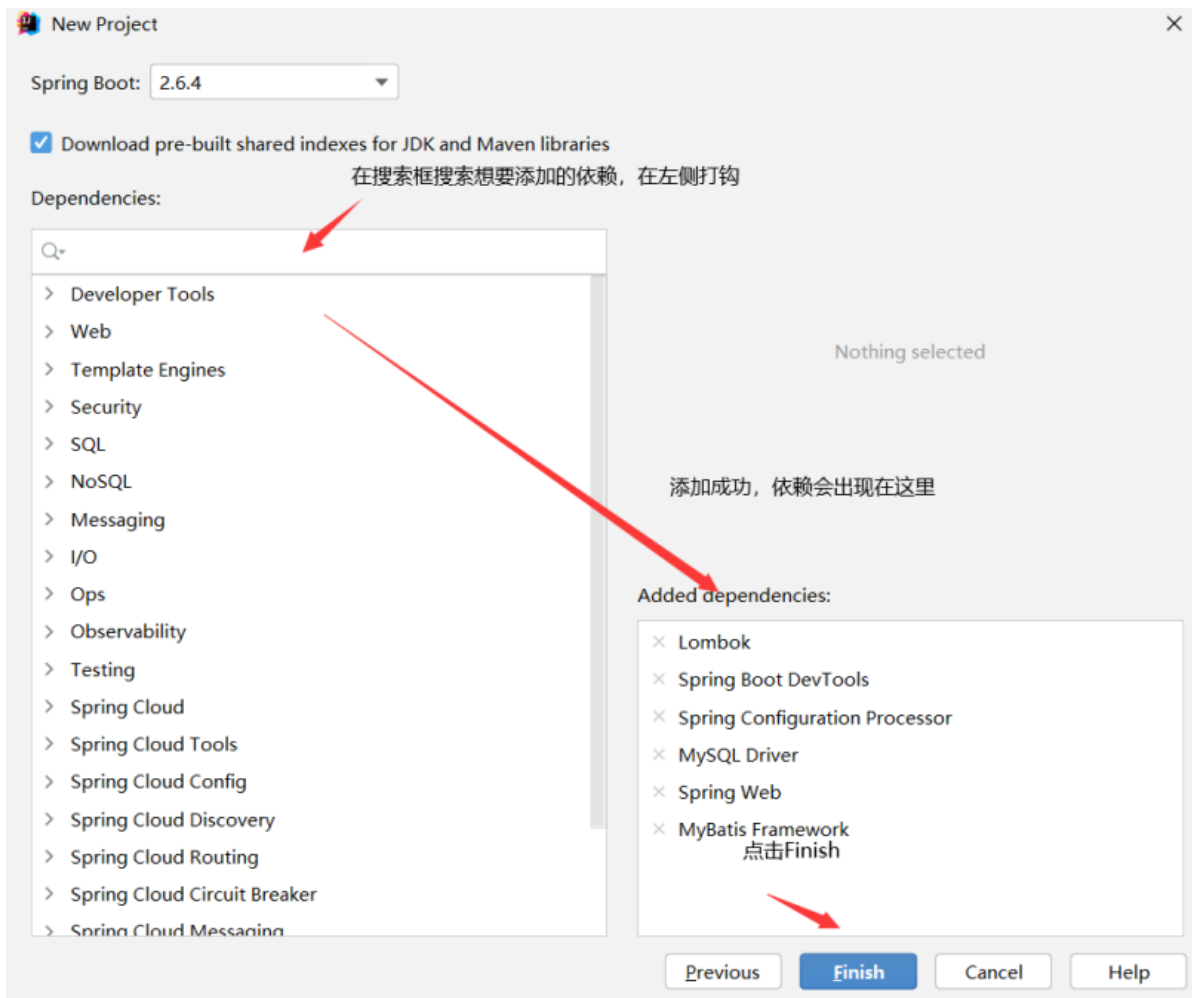




## 1.7.2.初始化springboot项目

- (1) 第一种方式去 github 上搜索 springboot 模板 , 在 github 上拉现成的模板(不推荐使用)
- (2) SpringBoot 官方的模板生成器 (<https://start.spring.io/>)
- (3) 直接在 IDEA 开发工具中生成 (推荐)





### 1.7.3.引入mybatis-plus

- 通过查看官方文档，书写 [mabatis-plus](#) demo 完成引入

## 1.8.问题及解决方案

- (1) 在单元测试中，需要添加 `@RunWith(SpringRunner.class)` 才可完成测试

```

package com.daxia.usercenterbackend;

import com.daxia.usercenterbackend.Mapper.UserMapper;
import com.daxia.usercenterbackend.model.User;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.util.List;

@SpringBootTest
@RunWith(SpringRunner.class)
public class SampleTest {

    @Resource
    private UserMapper userMapper;

    @Test
    public void testSelect() {
        System.out.println("----- selectAll method test -----");
        List<User> userList = userMapper.selectList(queryWrapper: null);
        Assert.assertEquals(expected: 5, userList.size());
        userList.forEach(System.out::println);
    }
}

```

- 原因：这里使用的 @Test 是 `org.junit.Test` 下的，需配合 `@RunWith(SpringRunner.class)` 才可完成单元测试
- 代替方法：@Test 可以使用 `org.junit.jupiter.api.Test`

```

package com.daxia.usercenterbackend;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class UserCenterBackendApplicationTests {

    @Test
    void contextLoads() {
    }

}

```

## 二、用户中心（中）

### 2.1.计划

1. 初始化项目



1. 前端初始化 20 min
  1. 初始化项目 ✓
  2. 引入一些组件之类的 ✓
  3. 框架介绍 / 瘦身 ✓
2. 后端初始化 20 min
  1. 准备环境 (MySQL 之类的) 验证 MySQL 是否安装成功 - 连接一下 ✓
  2. 初始化后端项目, 引入框架 (整合框架) ✓
2. 数据库设计 ✓★
3. 登录 / 注册 20min
  1. 后端 20min
    1. 规整项目目录 2 min ✓★
    2. 实现基本数据库操作 (操作 user 表) ✓★
      1. 模型 user 对象 => 和数据库的字段关联, 自动生成 ✓★
    3. 写注册逻辑 ✓★
  2. 前端 20min
4. 用户管理 (仅管理员可见) 20 min
  1. 前端
  2. 后端

## 2.2.数据库设计 30 min

---

### 2.2.1.数据库设计的简介

1. 什么是数据库? 存数据的
2. 数据库里有什么? 数据表 (理解为 excel 表格)
3. java 操作数据库, 代替人工
4. 什么是设计数据库表?
5. 有哪些表 (模型)? 表中有哪些字段? 字段的类型? 数据库字段添加索引?
6. 表与表之间的关联
7. 性别是否需要加索引?
  - 不需要。区分度不大的字段没必要加索引。

### 2.2.2.用户表设计

- 可选字段

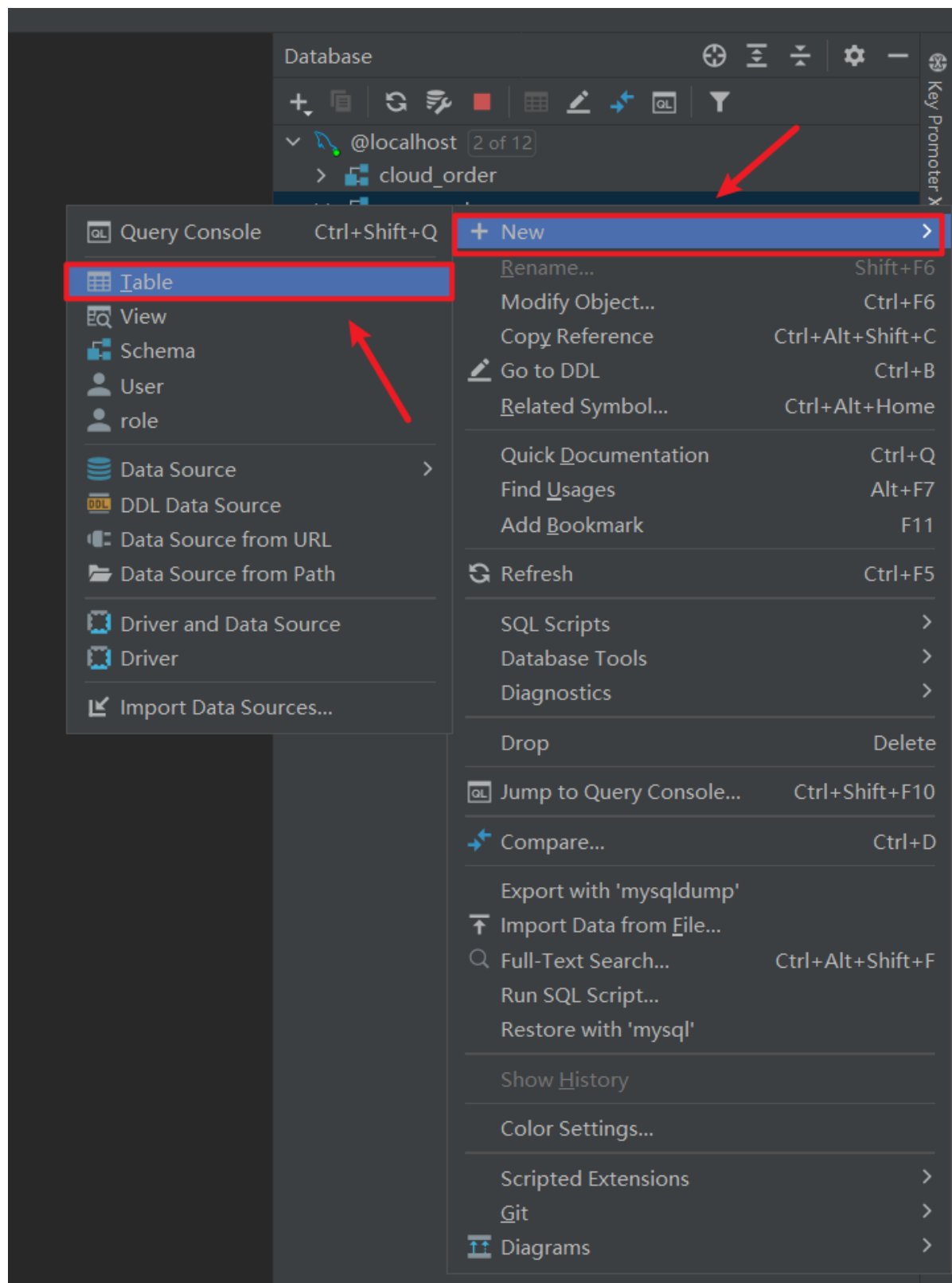
字段	说明	类型
id	主键，唯一标识	bigint
username	昵称	varchar
userAccount	登录账号	varchar
avatarUrl	头像	varchar
gender	性别	tinyint
userPassword	密码	varchar
phone	电话	varchar
email	邮箱	varchar
userStatus	用户状态；0 - 正常	int
userRole	用户权限 0 - 普通用户 1 - 管理员	int
planetCode	星球编号	varchar


- 必备字段

字段	说明	类型
createTime	创建时间（数据插入时间）	datetime
updateTime	更新时间（数据更新时间）	datetime
isDelete	是否删除（逻辑删除）	tinyint

### 2.2.3.创建用户表

(1) 方式一：傻瓜式创建



<div>  <span>Modify Table</span> </div>																													
Table:	<input type="text" value="user"/>																												
Comment:	<input type="text"/>																												
Columns (14)	<div> <div>Keys (1)</div> <div>Indexes</div> <div>Foreign Keys</div> <div>Grants</div> </div>																												
<div> <div> <div>+</div> <div>-</div> <div>▲</div> <div>▼</div> </div> <table> <tr> <td>id</td> <td>bigint(20) -- part of primary key /*id*/</td> </tr> <tr> <td>username</td> <td>varchar(256) /*用户昵称*/</td> </tr> <tr> <td>userAccount</td> <td>varchar(256) /*账号*/</td> </tr> <tr> <td>avatarUrl</td> <td>varchar(1024) /*用户头像*/</td> </tr> <tr> <td>gender</td> <td>tinyint(4) /*性别*/</td> </tr> <tr> <td>userPassword</td> <td>varchar(512) /*密码*/</td> </tr> <tr> <td>phone</td> <td>varchar(128) /*电话*/</td> </tr> <tr> <td>email</td> <td>varchar(512) /*邮箱*/</td> </tr> <tr> <td>userStatus</td> <td>int(11) default 0 /*状态 0 - 正常*/</td> </tr> <tr> <td>createTime</td> <td>datetime default CURRENT_TIMESTAMP /*创建时间*/</td> </tr> <tr> <td>updateTime</td> <td>datetime default CURRENT_TIMESTAMP /*更新时间*/</td> </tr> <tr> <td>isDelete</td> <td>tinyint(4) default 0 /*是否删除（逻辑删除）*/</td> </tr> <tr> <td>userRole</td> <td>int(11) default 0 /*用户权限 0 - 普通用户 1 - 管理员*/</td> </tr> <tr> <td>planetCode</td> <td>varchar(512) /*星球编号*/</td> </tr> </table> </div>		id	bigint(20) -- part of primary key /*id*/	username	varchar(256) /*用户昵称*/	userAccount	varchar(256) /*账号*/	avatarUrl	varchar(1024) /*用户头像*/	gender	tinyint(4) /*性别*/	userPassword	varchar(512) /*密码*/	phone	varchar(128) /*电话*/	email	varchar(512) /*邮箱*/	userStatus	int(11) default 0 /*状态 0 - 正常*/	createTime	datetime default CURRENT_TIMESTAMP /*创建时间*/	updateTime	datetime default CURRENT_TIMESTAMP /*更新时间*/	isDelete	tinyint(4) default 0 /*是否删除（逻辑删除）*/	userRole	int(11) default 0 /*用户权限 0 - 普通用户 1 - 管理员*/	planetCode	varchar(512) /*星球编号*/
id	bigint(20) -- part of primary key /*id*/																												
username	varchar(256) /*用户昵称*/																												
userAccount	varchar(256) /*账号*/																												
avatarUrl	varchar(1024) /*用户头像*/																												
gender	tinyint(4) /*性别*/																												
userPassword	varchar(512) /*密码*/																												
phone	varchar(128) /*电话*/																												
email	varchar(512) /*邮箱*/																												
userStatus	int(11) default 0 /*状态 0 - 正常*/																												
createTime	datetime default CURRENT_TIMESTAMP /*创建时间*/																												
updateTime	datetime default CURRENT_TIMESTAMP /*更新时间*/																												
isDelete	tinyint(4) default 0 /*是否删除（逻辑删除）*/																												
userRole	int(11) default 0 /*用户权限 0 - 普通用户 1 - 管理员*/																												
planetCode	varchar(512) /*星球编号*/																												

(2) 方式二：使用 sql 语句

```
-- auto-generated definition
create table user
(
    id          bigint auto_increment comment 'id'
        primary key,
    username    varchar(256)
        null comment '用户昵称',
    userAccount varchar(256)
        null comment '账号',
    avatarUrl   varchar(1024) default
'https://img1.baidu.com/it/u=1966616150,2146512490&fm=253&fmt=auto&app=138&f=JPEG?w=751&h=500' null comment '用户头像',
    gender      tinyint
        null comment '性别',
    userPassword varchar(512)
        not null comment '密码',
    phone       varchar(128)
        null comment '电话',
    email       varchar(512)
        null comment '邮箱',
    userStatus  int          default 0
        not null comment '状态 0 - 正
常',
    createTime  datetime     default CURRENT_TIMESTAMP
        null comment '创建时间',
```

```

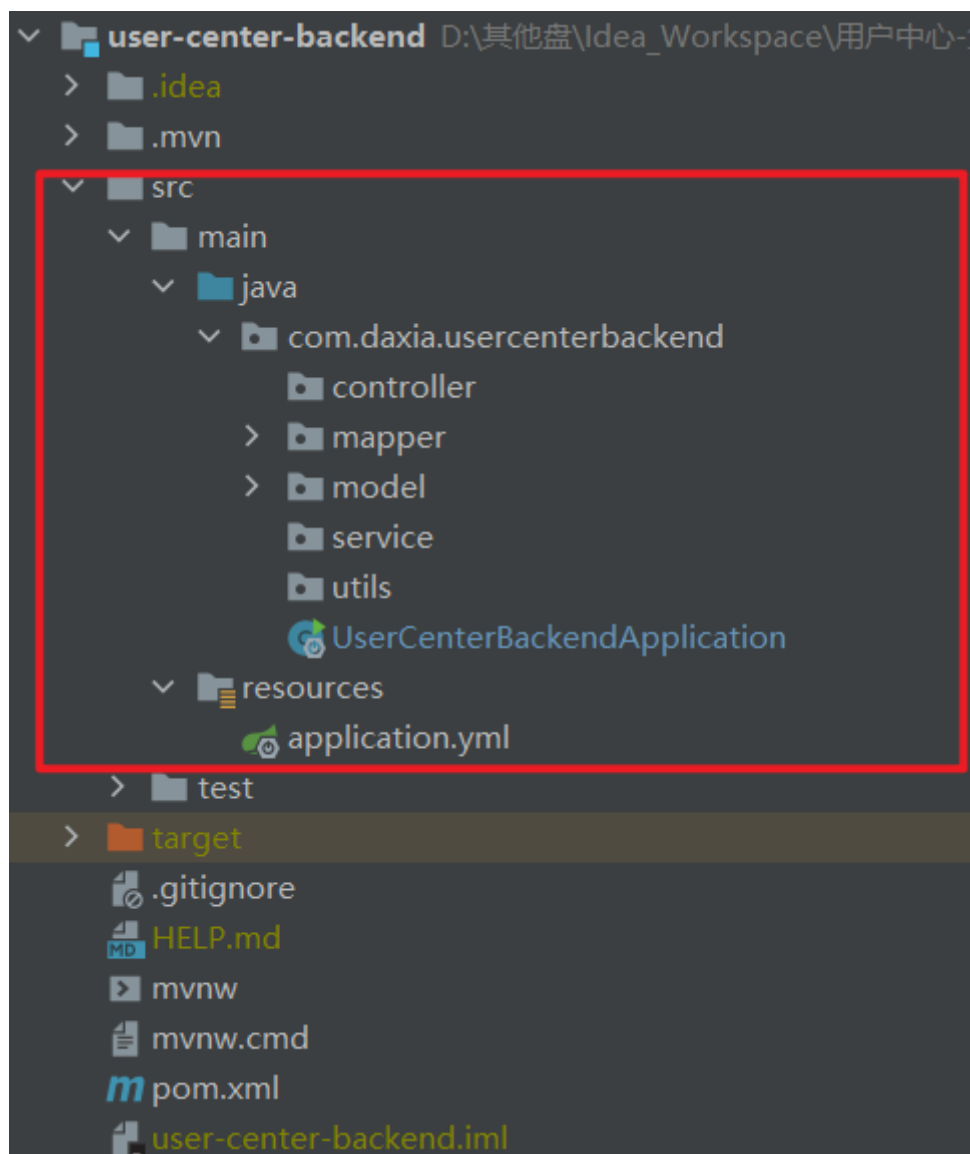
        updateTime    datetime    default CURRENT_TIMESTAMP
                                   null on update

        CURRENT_TIMESTAMP comment '更新时间',
        isDelete      tinyint     default 0
                                   not null comment '是否删除（逻辑删除）',
        userRole      int         default 0
                                   not null comment '用户权限 0
        - 普通用户 1 - 管理员',
        planetCode    varchar(512)
                                   null comment '星球编号'
    );

```

## 2.3.登录 / 注册功能

### 2.3.1.规整项目目录

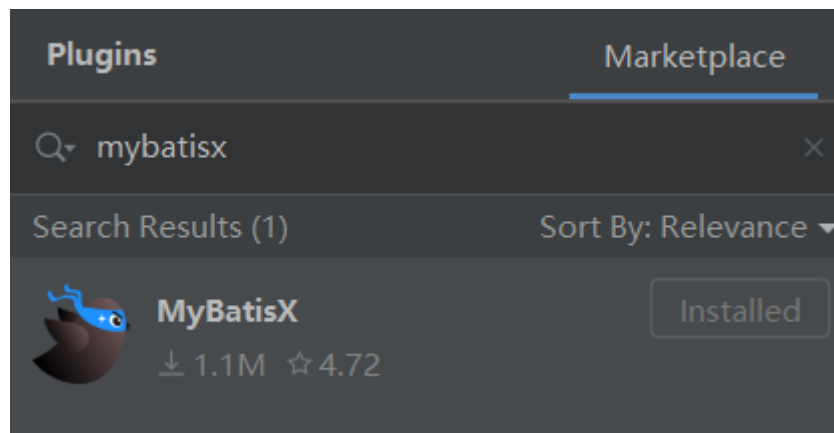


### 2.3.2.实现基本数据库操作（操作 user 表）

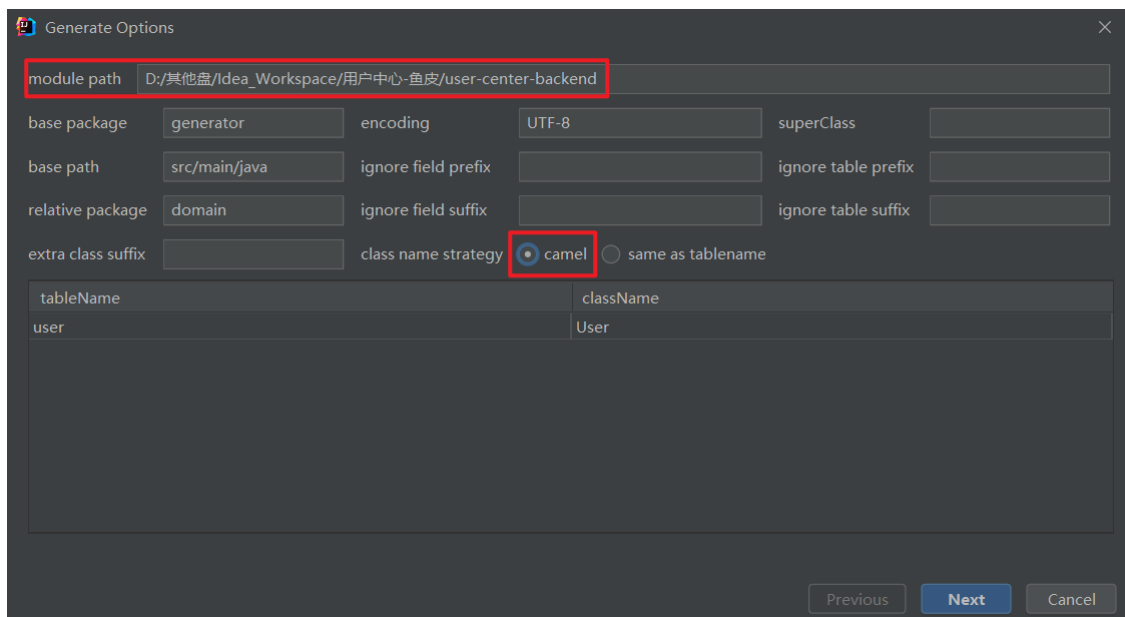
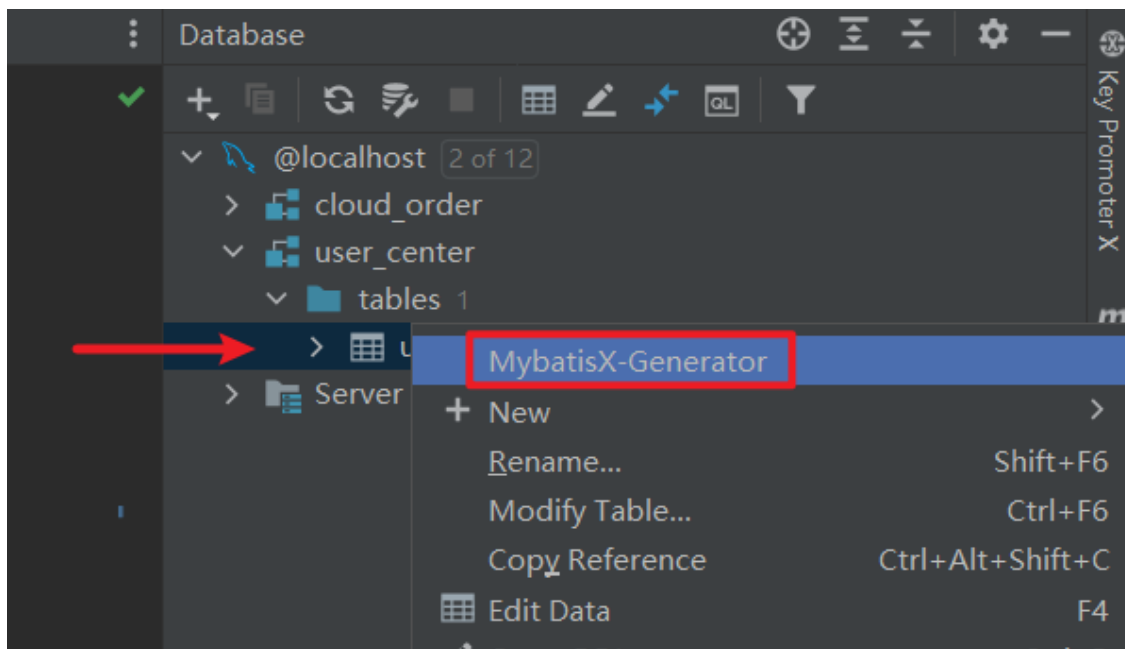
(1) 模型 user 对象 => 和数据库的字段关联，自动生成

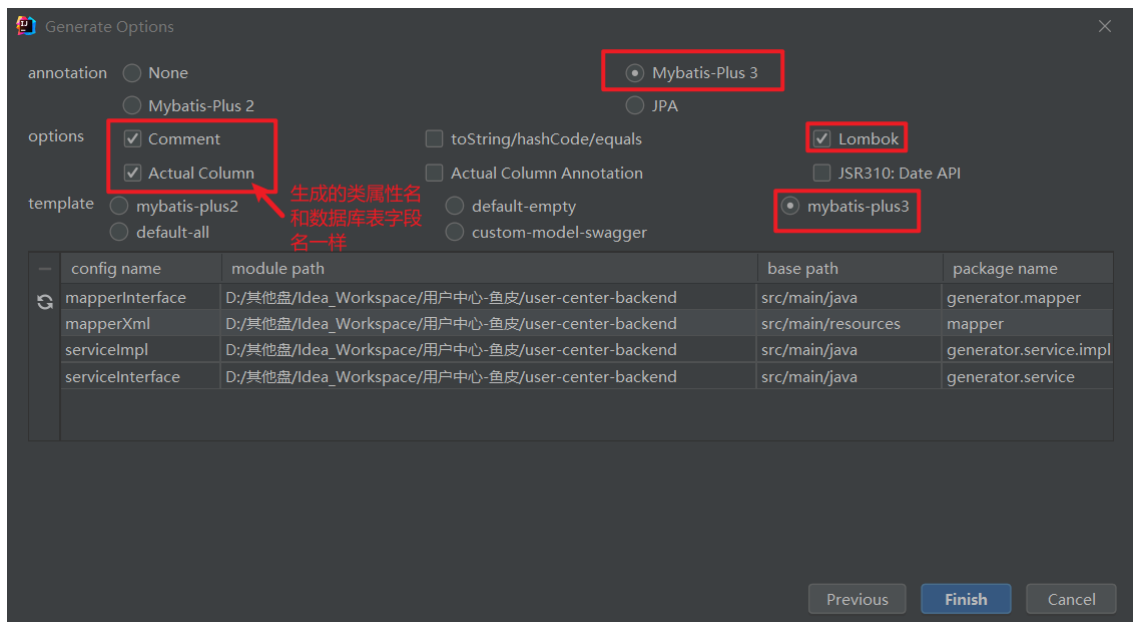
- 使用插件 mybatisX
  - MyBatisX 插件，自动根据数据库生成：

- domain 实体对象
- mapper (操作数据库的对象)
- mapper.xml (定义了 mapper对象和数据库的关联, 可以在里面自己写 SQL)
- service (包含常用的增删改查)
- serviceImpl (具体实现 service)



- 步骤:





- 测试使用 mybatisX 生成的代码

```
package com.daxia.usercenterbackend.service;

import com.daxia.usercenterbackend.mapper.UserMapper;
import com.daxia.usercenterbackend.model.User;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import javax.annotation.Resource;
import java.util.Date;

@SpringBootTest
class UserServiceTest {

    @Resource
    public UserMapper userMapper;

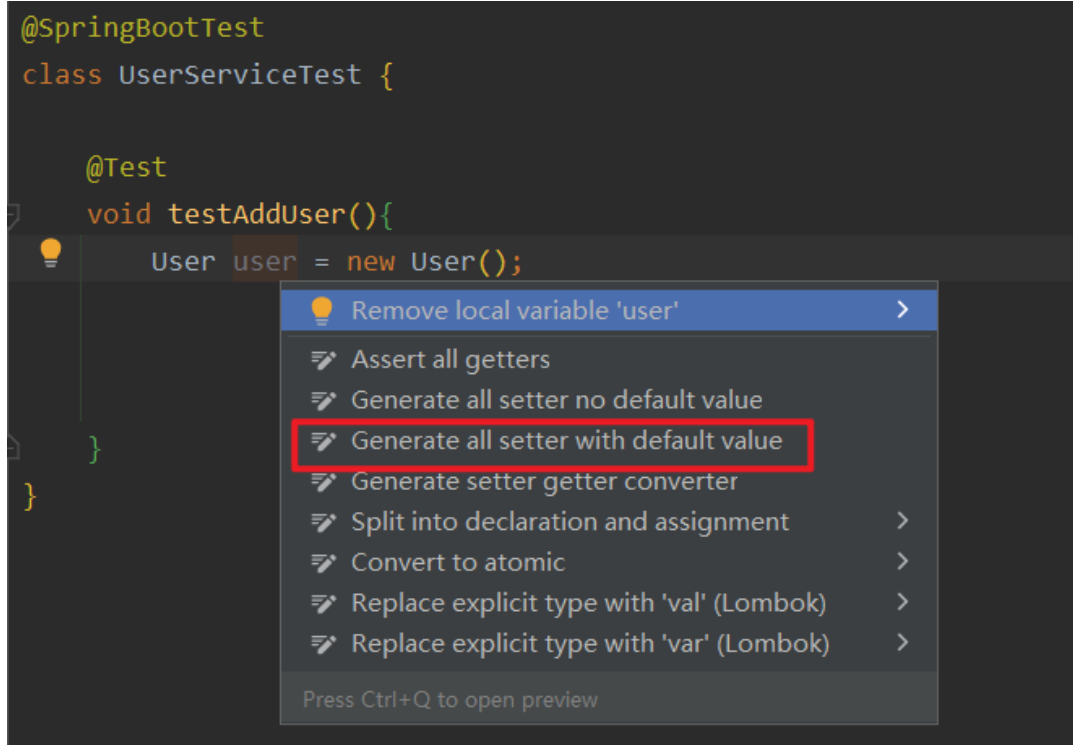
    @Test
    void testAddUser(){
        User user = new User();
        user.setUsername("daxia");
        user.setUserAccount("daxia123");

        user.setAvatarUrl("https://portrait.gitee.com/uploads/avatars/user/3031/9094632_DXdaxia_1620607641.png!avatar30");
        user.setGender(0);
        user.setUserPassword("");
        user.setPhone("111111");
        user.setEmail("222222@qq.com");
        user.setUserStatus(0);
        user.setCreateTime(new Date());
        user.setUpdateTime(new Date());
        user.setIsDelete(0);

        int result = userMapper.insert(user);
        Assertions.assertEquals(1, result);
    }
}
```

```
}  
}
```

- tips: 使用插件 GenerateAllSetter 生成对象的 set 方法



- 问题及解决方案

```
org.springframework.jdbc.BadSqlGrammarException:  
### Error updating database. Cause: java.sql.SQLException: Unknown column 'user_account' in 'field list'  
### The error may exist in com/daxia/usercenterbackend/mapper/UserMapper.java (best guess)  
### The error may involve com.daxia.usercenterbackend.mapper.UserMapper.insert-Inline  
### The error occurred while setting parameters  
### SQL: INSERT INTO user ( username, user_account, avatar_url, gender, user_password, phone, email, user_status, create_time, update_time, is_delete ) VALUES ( ?  
?, ?, ?, ?, ?, ?, ?, ?, ?, ? )  
### Cause: java.sql.SQLException: Unknown column 'user_account' in 'field list'  
; bad SQL grammar []; nested exception is java.sql.SQLException: Unknown column 'user_account' in 'field list'
```

- 原因

### mapUnderscoreToCamelCase

- 类型: `boolean`
- 默认值: `true`

是否开启自动驼峰命名规则 (camel case) 映射, 即从经典数据库列名 A\_COLUMN (下划线命名) 到经典 Java 属性名 aColumn (驼峰命名) 的类似映射。

#### 注意

此属性在 MyBatis 中原默认值为 false, 在 MyBatis-Plus 中, 此属性也将用于生成最终的 SQL 的 select body 如果您的数据库命名符合规则无需使用 `@TableField` 注解指定数据库字段名

- 解决方案

在配置文件 application.yml 中添加如下配置:

```
# mybatis-plus的配置  
mybatis-plus:  
  configuration:  
    # 解决 mybatis-plus 框架中 “ Unknown column 'user_account' in 'field list' ”  
    map-underscore-to-camel-case: false
```



## 2.4.注册-后端

### 2.4.1.注册逻辑

1. 用户在前端输入账户和密码、以及校验码 (todo)
2. 校验用户的账户、密码、校验密码，是否符合要求
  1. 非空
  2. 账户长度 **不小于** 4 位
  3. 密码就 **不小于** 8 位
  4. 账户不能重复
  5. 账户不包含特殊字符
  6. 密码和校验密码相同
3. 对密码进行加密 (密码千万不要直接以明文存储到数据库中)
4. 向数据库插入用户数据

### 2.4.2.后端实现-注册逻辑

1. 校验用户的账户、密码、校验密码，是否符合要求
    1. 非空
    2. 账户长度 **不小于** 4 位
    3. 密码就 **不小于** 8 位
    4. 账户不能重复
    5. 账户不包含特殊字符
    6. 密码和校验密码相同
  2. 对密码进行加密 (密码千万不要直接以明文存储到数据库中)
  3. 向数据库插入用户数据
- 代码实现：

```
/**
 * @author 大虾
 * @description 针对表【user】的数据库操作Service实现
 * @createDate 2022-07-07 21:58:50
 */
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User>
    implements UserService{

    @Resource
    public UserMapper userMapper;

    private final String SALT = "daxiaDaxia";

    @Override
    public Long userRegister(String userAccount, String userPassword, String
checkPassword) {
        // 1.校验用户的账户、密码、校验密码，是否符合要求
        // 1.1.非空校验
        if(StringUtils.isAnyBlank(userAccount, userPassword, checkPassword))
        {
            return (long) -1;
        }
    }
}
```

```

    }
    // 1.2. 账户长度不小于4位
    if(userAccount.length() < 4) {
        return (long) -1;
    }
    // 1.3. 密码就不小于8位
    if(userPassword.length() < 8) {
        return (long) -1;
    }
    // 1.4. 账户不包含特殊字符
    String validRule = "[~!@#$%^&*()+=|{}':;',\\\\\\\\[\\\\\\\\].<.>/?~!@#¥%.....
    &* () --+|{}【】‘；：”“”。、？]";
    Matcher matcher = Pattern.compile(validRule).matcher(userAccount);
    // 如果包含非法字符，则返回
    if(matcher.find()){
        return (long) -1;
    }
    // 1.5. 密码和校验密码相同
    if(!userPassword.equals(checkPassword)) {
        return (long) -1;
    }
    // 1.6. 账户不能重复
    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("userAccount", userAccount);
    Long count = userMapper.selectCount(queryWrapper);
    if(count > 0) {
        return (long) -1;
    }

    // 2.对密码进行加密（密码千万不要直接以明文存储到数据库中）
    String verifyPassword = DigestUtils.md5DigestAsHex((SALT +
    userPassword).getBytes(StandardCharsets.UTF_8));

    // 3. 向数据库插入用户数据
    User user = new User();
    user.setUserAccount(userAccount);
    user.setUserPassword(verifyPassword);
    int res = userMapper.insert(user);
    if(res < 0){
        return (long) -1;
    }

    return user.getId();
}
}

```

## 三、用户中心（下）

### 3.1.计划

#### 1. 初始化项目

##### 1. 前端初始化 20 min

##### 1. 初始化项目 ✓

- 2. 引入一些组件之类的 ✓
- 3. 框架介绍 / 瘦身 ✓
- 2. 后端初始化 20 min
  - 1. 准备环境 (MySQL 之类的) 验证 MySQL 是否安装成功 - 连接一下 ✓
  - 2. 初始化后端项目, 引入框架 (整合框架) ✓
- 2. 数据库设计 ✓
- 3. 登录 / 注册 20min
  - 1. 后端 20min
    - 1. 规整项目目录 2 min ✓
    - 2. 实现基本数据库操作 (操作 user 表) ✓
      - 1. 模型 user 对象 => 和数据库的字段关联, 自动生成 ✓
    - 3. 写注册逻辑 ✓
    - 4. 写登录逻辑 ✓★
  - 2. 前端 20min
    - 1. 删除多余代码 ✓★
    - 2. 完成登录页面 ✓★
- 4. 用户管理 (仅管理员可见) 20 min
  - 2. 后端 ✓★
  - 3. 前端

## 3.2.后端-登录

---

### 3.2.1.登录接口

- 接受参数: 用户账户、密码
- 请求类型: POST
- 请求体: JSON 格式的数据

请求参数很长时不建议用 get

- 返回值: 用户信息 (脱敏)

### 3.2.2.登录逻辑

- 1. 校验用户账户和密码是否合法
    - 1. 非空
    - 2. 账户长度 **不小于** 4 位
    - 3. 密码就 **不小于** 8 位吧
    - 4. 账户不包含特殊字符
  - 2. 校验密码是否输入正确, 要和数据库中的密文密码去对比
  - 3. 用户信息脱敏, 隐藏敏感信息, 防止数据库中的字段泄露
  - 4. 我们要记录用户的登录态 (session), 将其存到服务器上 (用后端 SpringBoot 框架封装的服务器 tomcat 去记录)  
cookie
  - 5. 返回脱敏后的用户信息
- 代码实现

```

public User doLogin(String userAccount, String userPassword,
HttpServletRequest request) {
    // 1.校验用户的账户、密码是否符合要求
    // 1.1.非空校验
    if(StringUtils.isAnyBlank(userAccount, userPassword)) {
        return null;
    }
    // 1.2. 账户长度不小于4位
    if(userAccount.length() < 4) {
        return null;
    }
    // 1.3. 密码就不小于8位
    if(userPassword.length() < 8) {
        return null;
    }
    // 1.4. 账户不包含特殊字符
    String validRule = "[`~!@#$$%^&*()+=|{}':;',\\\\\\\\[\\\\\\\\].<./?~!@#¥%.....
    &*()--+|{}【】‘；：”“’。、？]";
    Matcher matcher = Pattern.compile(validRule).matcher(userAccount);
    // 如果包含非法字符，则返回
    if(matcher.find()){
        return null;
    }

    // 2.校验密码是否输入正确，要和数据库中的密文密码对比去
    String encodePassword = DigestUtils.md5DigestAsHex((SALT +
userPassword).getBytes(StandardCharsets.UTF_8));
    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    // 这里存在bug: 会把逻辑删除的用户查出来
    queryWrapper.eq("userAccount", userAccount);
    queryWrapper.eq("userPassword", encodePassword);
    User user = userMapper.selectOne(queryWrapper);
    if(user == null){
        log.info("user login failed, userAccount Cannot match
userPassword");
    }

    // 3.用户信息脱敏，隐藏敏感信息，防止数据库中的字段泄露
    User newUser = new User();
    newUser.setId(user.getId());
    newUser.setUsername(user.getUsername());
    newUser.setUserAccount(user.getUserAccount());
    newUser.setAvatarUrl(user.getAvatarUrl());
    newUser.setGender(user.getGender());
    newUser.setPhone(user.getPhone());
    newUser.setEmail(user.getEmail());
    newUser.setUserStatus(user.getUserStatus());
    newUser.setCreateTime(user.getCreateTime());

    // 4.记录用户的登录态（session），将其存到服务器上
    request.getSession().setAttribute(USER_LOGIN_STATE, newUser);

    // 5.返回脱敏后的用户信息
    return newUser;
}

```

- 问题以及解决方案
  - 原因：以上第二步检验密码是否正确时，查询数据库时，会把逻辑删除的用户数据也一并查出来，所以需要对 mybatis-plus 框架进行设置
  - 解决方案：
    - 在 application.yml 配置文件中配置，可避免这种情况出现

```
mybatis-plus:
  global-config:
    db-config:
      logic-delete-field: isDelete # 全局逻辑删除的实体字段名(since
3.3.0,配置后可以忽略不配置步骤2)
      logic-delete-value: 1 # 逻辑已删除值(默认为 1)
      logic-not-delete-value: 0 # 逻辑未删除值(默认为 0)
```

- 实体类字段上加上@TableLogic注解

```
@TableLogic
private Integer isDelete;
```

- 参考链接：<https://baomidou.com/pages/6b03c5/#%E4%BD%BF%E7%94%A8%E6%96%B9%E6%B3%95>

### 3.2.3.如何知道是哪个用户登录了？

1. 连接服务器端后，得到一个 session 状态（匿名会话），返回给前端
2. 登录成功后，得到了登录成功的 session，并且给该session设置一些值（比如用户信息），返回给前端一个设置 cookie 的“命令”

**session => cookie**

3. 前端接收到后端的命令后，设置 cookie，保存到浏览器内
4. 前端再次请求后端的时候（相同的域名），在请求头中带上cookie去请求
5. 后端拿到前端传来的 cookie，找到对应的 session
6. 后端从 session 中可以取出基于该 session 存储的变量（用户的登录信息、登录名）

###

### 3.2.4.控制层 Controller 封装请求

#### (1) application.yml 指定接口全局 api

```
servlet:
  context-path: /api
```

#### (2) 注册接口

<http://localhost/8080/api/user/register>

#### (3) 登录接口

<http://localhost/8080/api/user/login>

#### (4) 拓展

1. @RestController 适用于编写 restful 风格的 api, 返回值默认为 json 类型
2. controller 层校验和 service 层校验的区别
  - controller 层倾向于对请求参数本身的校验, 不涉及业务逻辑本身 (越少越好)
  - service 层是对业务逻辑的校验 (有可能被 controller 之外的类调用)

### 3.3.写代码流程

- 先做设计
- 代码实现
- 持续优化!!! (复用代码、提取公共逻辑 / 常量)

### 3.4.后端-用户管理 (仅管理员可见)

- 接口设计关键: 必须鉴权!!!
  1. 查询用户 (允许根据用户名查询)
  2. 删除用户
- controller 层

```
/**
 * 管理员查询
 * @param username 用户昵称
 * @param request springboot内置请求对象, 用于存储用户session
 * @return 查到得到的所有用户信息
 */
@GetMapping("/search")
List<User> searchUsers(String username, HttpServletRequest request){
    return userService.searchUsers(username, request);
}

/**
 * 管理员删除
 * @param id 用户id
 * @param request springboot内置请求对象, 用于存储用户session
 * @return 是否删除用户, true表示删除; false表示删除失败
 */
@PostMapping("/delete")
boolean deleteUser(@RequestBody long id, HttpServletRequest request){
    System.out.println(id);
    return userService.deleteUser(id, request);
}
```

- service 层

```
@Override
public List<User> searchUsers(@RequestBody String username,
    HttpServletRequest request) {
    // 鉴权: 只有管理员才能查询用户
    if(!isAdmin(request)){
        return new ArrayList<>();
    }
    QueryWrapper<User> queryWrapper = new QueryWrapper<>();
    // 若模糊查询的字段为空, 则忽略此操作
```

```

        if(StringUtils.isNotBlank(username)){
            queryWrapper.like("username", username);
        }
        List<User> userList = userMapper.selectList(queryWrapper);
        return userList.stream().map(user ->
            getsafteUser(user)).collect(Collectors.toList());
    }

    @Override
    public boolean deleteUser(long id, HttpServletRequest request) {
        // 鉴权: 只有管理员才能删除用户
        if(!isAdmin(request) || id < 0){
            return false;
        }
        return userMapper.deleteById(id) > 0;
    }
}

```

## 3.5.前端-登录

(1) 先对项目代码进行精简 -> 将登录页面修改成自己心仪样子

(2) 实现前后端的交互

- 这里有跨域问题需要解决
  - tips: 跨域问题虽然在后端也可以解决, 但是这样不安全, 一般都是在前端解决
  - 解决方法: 正向代理
  - 修改 config/proxy.ts 下的文件

当发起 `http://localhost:8000/api/user/login` 时, 正向代理到  
`http://localhost:8080/api/user/login`, 从而解决跨域问题

```

dev: {
    // localhost:8000/api/** -> https://preview.pro.ant.design/api/**
    '/api': {
        // 要代理的地址
        target: 'http://localhost:8080',
        // 配置了这个可以从 http 代理到 https
        // 依赖 origin 的功能可能需要这个, 比如 cookie
        changeOrigin: true,
    },
},

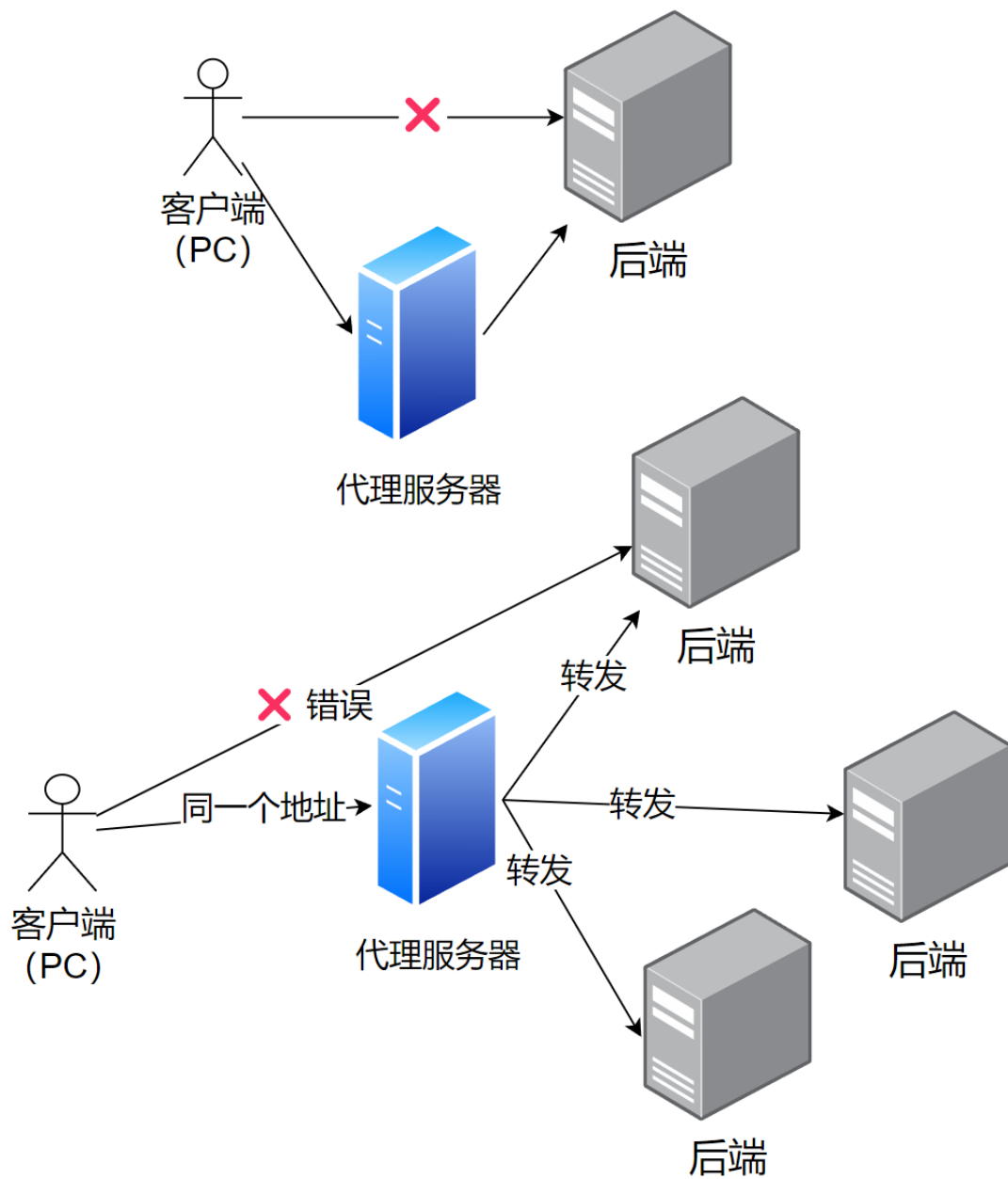
```

- 前后端交互
  - 前端需要向后端发送请求才能获取数据 / 执行操作。
- 怎么发请求: 前端使用 ajax 来请求后端
- 前端请求库及封装关系
  - axios 封装了 ajax
- request 是 ant design 项目又封装了一次
- 追踪 request 源码: 用到了 umi 的插件、requestConfig 配置文件
- 代理

- 正向代理：替客户端向服务器发送请求，可以解决跨域问题
- 反向代理：替服务器统一接收请求。
- 怎么实现代理？
  - Nginx 服务器
  - Node.js 服务器

## • 举例

- 原本请求：<http://localhost:8000/api/user/login>
- 代理到请求：<http://localhost:8080/api/user/login>



## 四、用户中心（终）

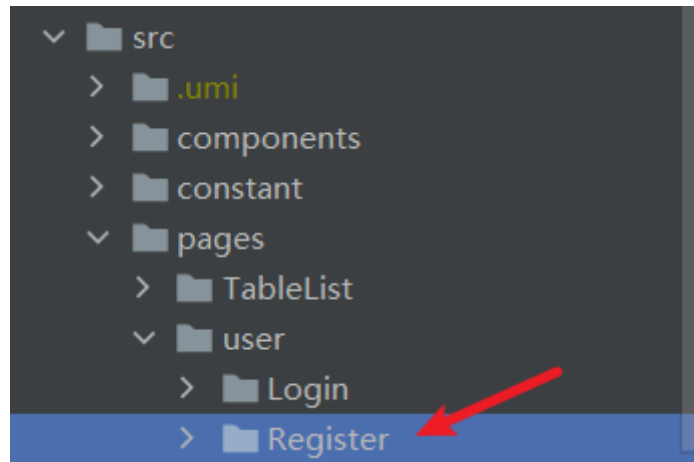
### 4.1.计划



1. 完成注册功能的前端开发 15 - 20min
2. 获取用户的登录态，**获取当前登录用户信息接口**
3. 完成用户管理后台的前端开发 20 - 30 min

## 4.2.前端-注册

(1) 复制登录页面，修改为注册页面



(2) 添加路由规则

```
{ name: '注册', path: '/user/register', component: './user/Register' },
```

```
export default [
  {
    path: '/user',
    layout: false,
    routes: [
      { name: '登录', path: '/user/login', component: './user/Login' },
      { name: '注册', path: '/user/register', component: './user/Register' },
      { component: './404' },
    ],
  },
  { path: '/welcome', name: '欢迎', icon: 'smile', component: './Welcome' },
  {
    path: '/admin',
    name: '管理页',
    icon: 'crown',
    access: 'canAdmin',
    routes: [
      { path: '/admin/sub-page', name: '二级管理页', icon: 'smile', component: './Welcome' },
      { component: './404' },
    ],
  },
  { name: '查询表格', icon: 'table', path: '/list', component: './TableList' },
  { path: '/', redirect: '/welcome' },
  { component: './404' },
];
```

(3) 在添加完组件以及路由之后，输入 `http://localhost:8000/user/register`，发现被强制路由至登录页面。此时想到 ant design pro 是一个后台管理系统，在未登录情况下想操作其它页面，势必会被强制路由到登录页面。所以需要修改此逻辑。

- 解决方案：修改项目入口文件 `src/app.tsx`

```

35   const fetchUserInfo = async () => {
36       try {
37           const msg = await queryCurrentUser();
38           return msg.data;
39       } catch (error) {
40           // history.push(loginPath);
41       }
42       return undefined;
43   };

```

```

onPageChange: () => {
    const { location } = history;
    // 添加白名单，登录页面以及注册页面不需要进行强制路由
    if(location.pathname === "/user/register"){
        return ;
    }
    // 如果没有登录，重定向到 login
    if (!initialState?.currentUser && location.pathname !== loginPath) {
        history.push(loginPath);
    }
},

```

#### (4) 修改注册逻辑

```

const handleSubmit = async (values: API.RegisterParams) => {
    try {
        const { userPassword, checkPassword } = values
        // 简单校验
        if( userPassword !== checkPassword){
            const defaultLoginFailureMessage = '两次输入密码不一致，请重新输入';
            message.error(defaultLoginFailureMessage);
            return ;
        }
        // 注册
        const userId = await register(values);

        if (userId > 0) {
            const defaultLoginSuccessMessage = '注册成功! ';
            message.success(defaultLoginSuccessMessage);

            /** 此方法会跳转到 redirect 参数所在的位置 */
            if (!history) return;
            const { query } = history.location;
            const { redirect } = query as {
                redirect: string;
            };

            history.push("/user/login?redicet=" + redirect);
            return;
        }
        throw new Error(`register is fail, Id is ${userId}`);
    }
}

```

```

    } catch (error) {
      const defaultLoginFailureMessage = '注册失败，请重试!';
      message.error(defaultLoginFailureMessage);
    }
  };
};

```

(5) 删除页面中冗余的代码

- 技巧：可以看代码在哪里用到，若发现对该页面无用，直接可以连带删除

(6) 在登录页面填加注解接口并调整页面格局

<Link to="/user/register">用户注册</Link>

```

131   <div
132     style={{
133       marginBottom: 24,
134     }}
135   >
136     <ProFormCheckbox noStyle name="autoLogin">
137       自动登录
138     </ProFormCheckbox>
139     <Divider type="vertical"/>
140     <Link to="/user/register">用户注册</Link>
141     <Divider type="vertical"/>
142     <a
143       style={{
144         float: 'right',
145       }}
146       href={PLANET_LINK}
147       target="_blank"
148       rel="noreferrer"
149     >
150       忘记密码请联系大虾 ?
151     </a>
152   </div>

```

## 4.3.后端添加获取当前用户信息的接口

- controller 层

```

@GetMapping("/current")
User searchUsers(HttpServletRequest request){
    return userService.getCurrentUser(request);
}

```

- service 层

```

public User getCurrentUser(HttpServletRequest request) {
    Object useObject = request.getSession().getAttribute(USER_LOGIN_STATE);
    User user = (User) useObject;
    if(user == null) {
        return null;
    }
    // TODO 校验用户是否合法
    Long userId = user.getId();
    User newUser = userMapper.selectById(userId);
    return getSaftyUser(newUser);
}

```

## 4.4.前端-完成登录之后进入后台管理系统

- 修改 src/app.tsx 目录下的初始化函数

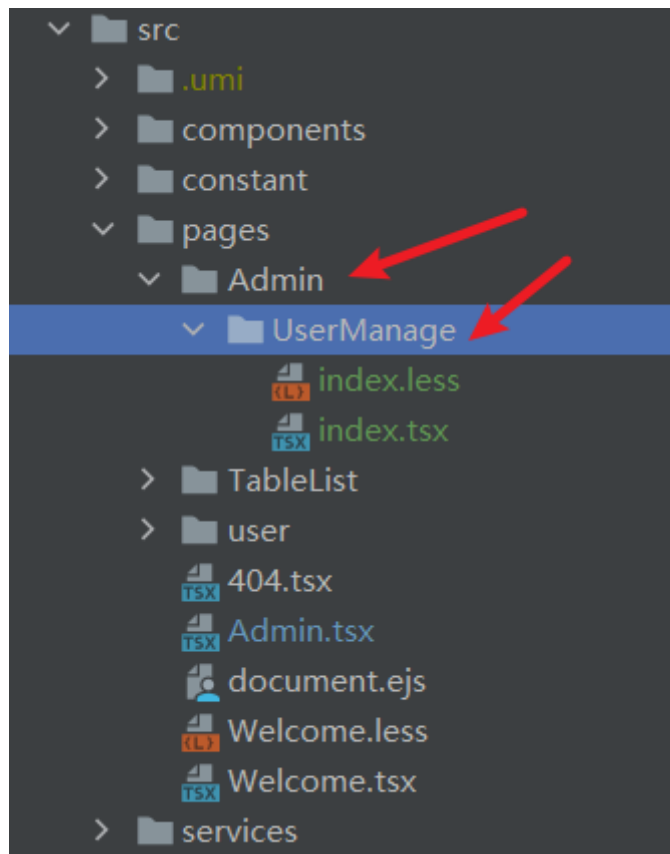
```

export async function getInitialState(): Promise<{
    settings?: Partial<LayoutSettings>;
    currentUser?: API.CurrentUser;
    loading?: boolean;
    fetchUserInfo?: () => Promise<API.CurrentUser | undefined>;
}> {
    const fetchUserInfo = async () => {
        try {
            return await queryCurrentUser();
        } catch (error) {
            history.push(loginPath);
        }
        return undefined;
    };
    // 如果是登录页面或注册页面，则执行
    if (NO_NEED_LOGIN_WHITE_LIST.includes(history.location.pathname)) {
        return {
            fetchUserInfo,
            settings: defaultSettings,
        };
    }
    const currentUser = await fetchUserInfo();
    return {
        fetchUserInfo,
        currentUser,
        settings: defaultSettings,
    };
}

```

## 4.5.前端-后台管理界面

- (1) 复制注册页面，改为 UserManage 页面



(2) 在 config/routes.ts 下添加路由规则

```
{ path: '/admin/user-manage', name: '用户管理', icon: 'smile', component: './Admin/UserManage' }
```

```
export default [
  {
    path: '/user',
    layout: false,
    routes: [
      { name: '登录', path: '/user/login', component: './user/Login' },
      { name: '注册', path: '/user/register', component: './user/Register' },
      { component: './404' },
    ],
  },
  { path: '/welcome', name: '欢迎', icon: 'smile', component: './Welcome' },
  {
    path: '/admin',
    name: '管理页',
    icon: 'crown',
    access: 'canAdmin',
    routes: [
      { path: '/admin/user-manage', name: '用户管理', icon: 'smile', component: './Admin/UserManage' },
      { component: './404' },
    ],
  },
  { name: '查询表格', icon: 'table', path: '/list', component: './TableList' },
  { path: '/', redirect: '/welcome' },
  { component: './404' },
];
```

(3) 在 src/pages/Admin/UserManage/index.tsx 下, 引入 ant design Pro [高级表格组件](#)

(4) 修改用户管理界面布局后

## 五、用户中心 (终)

## 5.1.计划

---

1. 开发用户注销 前端 7 分钟 / 后端 3 分钟 ✓
2. 补充用户注册校验逻辑 前端 10 分钟 / 后端 10 分钟 ✓
3. 后端代码优化 20 - 30 min
4. 前端代码优化 5 - 10 min
5. 项目部署上线 1h
  1. 买服务器 10 min
  2. 原生部署 20 - 30 min
  3. 容器部署 20 - 30 min
  4. 绑定域名 5 min
  5. 排查问题

## 5.2.前后端-用户退出登录

---

### 5.2.1.后端-用户退出

- controller 层

```
/**
 * 用户注销
 * @param request springboot内置请求对象，用于存储用户session
 * @return
 */
@PostMapping("/logout")
Integer userLogout(HttpServletRequest request) {
    if(request == null){
        return null;
    }
    return userService.userLogout(request);
}
```

- service 层

```
@Override
public Integer userLogout(HttpServletRequest request) {
    request.getSession().removeAttribute(USER_LOGIN_STATE);
    return 1;
}
```

### 5.2.2.前端-用户退出

- 修改请求路径

```

/** 退出登录接口 POST /api/user/logout */
export async function outLogin(options?: { [key: string]: any }) {
  return request<Record<string, any>>('/api/user/logout', {
    method: 'POST',
    ...(options || {}),
  });
}

```

- 详细退出看 src/components/RightContent/AvatarDropdown.tsx 文件

## 5.3.user 表增加 planetCode 字段

- 用户校验逻辑

仅适用于用户可信的情况

1. 先让用户自己填：2 - 5 位编号，自觉
2. 后台补充对编号的校验：长度校验、唯一性校验
3. 前端补充输入框，适配后端

后期拉取星球数据，定期清理违规用户

##

## 5.4.后端-优化

1. 通用返回对象 ✓

目的：给对象补充一些信息，告诉前端这个请求在业务层面上是成功还是失败

200、404、500、502、503

```

{
  "name": "yupi"
}

↓

// 成功
{
  "code": 0 // 业务状态码
  "data": {
    "name": "yupi"
  },
  "message": "ok",
  "description": ""
}

// 错误
{
  "code": 50001 // 业务状态码
  "data": null
  "message": "用户操作异常、xxx",
  "description": "..."
}

```

- 自定义错误码
- 返回类支持返回正常和错误

## 2. 封装全局异常处理

### 1. 定义业务异常类

- 相对于 java 的异常类，支持更多字段
- 自定义构造函数，更灵活 / 快捷的设置字段

### 2. 编写全局异常处理器

- 作用：
  - 捕获代码中所有的异常，内部消化，让前端得到更详细的业务报错 / 信息
  - 同时屏蔽掉项目框架本身的异常（不暴露服务器内部状态）
  - 集中处理，比如记录日志
- 实现：
  - Spring AOP：在调用方法前后进行额外的处理

### 3. todo：全局请求日志和登录校验

## 5.5.前端优化

- 后端定义统一的返回对象，会对前端读取数据造成影响，所以需要去适配前端接受到的数据
  - 方法一：对返回数据进行封装，后修改注册、登录等逻辑
    - 在 `src/services/ant-design-pro/typings.d.ts` 目录下，添加新类型

```
/**
 * 通用的响应模板
 */
type BaseResponse<T> = {
  code?: string;
  data?: T;
  message?: string;
  description?: string;
}
```

- 在 `src/services/ant-design-pro/api.ts` 修改请求函数的返回值类型

```
/** 注册接口 POST /api/user/register */
export async function register(body: API.RegisterParams, options?: {
  [key: string]: any }) {
  return request<BaseResponse<API.RegisterResult>>
    ('/api/user/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      data: body,
      ...(options || {}),
    });
}
```

- 修改注册、登录等逻辑



```

const handleSubmit = async (values: API.RegisterParams) => {
  try {
    const { userPassword, checkPassword } = values;
    // 简单校验
    if (userPassword !== checkPassword) {
      const defaultLoginFailureMessage = '两次输入密码不一致，请重新输入';
      message.error(defaultLoginFailureMessage);
      return;
    }
    // 注册
    const {code, data, description} = await register(values);

    // @ts-ignore
    if (code === SUCCESS && data.id > 0) {
      const defaultLoginSuccessMessage = '注册成功!';
      message.success(defaultLoginSuccessMessage);

      /** 此方法会跳转到 redirect 参数所在的位置 */
      if (!history) return;
      const { query } = history.location;
      history.push({
        pathname: '/user/login',
        query,
      });
      return;
    }
    throw new Error(description);
  } catch (error: any) {
    message.error(error.message);
  }
};

```

- 方法二：自定义全局响应处理

```

/**
 * request 网络请求工具
 * 更详细的 api 文档: https://github.com/umijs/umi-request
 * https://github.com/umijs/umi-request#interceptor
 */
import {extend} from 'umi-request';
import {NOT_LOGIN, SUCCESS} from "@/constant";
import {history} from "@@/core/history";
import {message} from "antd";

/**
 * 配置request请求时的默认参数
 */
const request = extend({
  credentials: 'include', // 默认请求是否带上cookie
});

/**
 * 所以请求拦截器
 */

```

```

request.interceptors.request.use((url, options): any => {
  console.log(`do request url = ${url}`)

  return {
    url,
    options: {
      ...options,
      headers: {},
    },
  };
});

/**
 * 所有响应拦截器
 */
request.interceptors.response.use(async (response): Promise<any> => {
  const res = await response.clone().json();
  if(res.code === SUCCESS){
    return res.data;
  }
  if(res.code === NOT_LOGIN ){
    message.error("请先登录");
    const { query } = history.location;
    history.push({
      pathname: '/user/login',
      query,
    });
  }else{
    message.error(res.description)
  }
  return response;
});

export default request;

```

- 注意：要在 `src/services/ant-design-pro/api.ts` 下引入自定一的 request

直播 0:00 - 0:35 踩坑过程，可跳过

1. 对接后端的返回值，取 data

2. 全局响应处理：

1. 应用场景：我们需要对接口的 **通用响应** 进行统一处理，比如从 response 中取出 data；或者根据 code 去集中处理错误，比如用户未登录、没权限之类的。
2. 优势：不用在每个接口请求中都去写相同的逻辑
3. 实现：参考你用的请求封装工具的官方文档，比如 umi-request (<https://github.com/umijs/umi-request#interceptor>、<https://blog.csdn.net/huantai3334/article/details/116780020>) 如果你用 axios，参考 axios 的文档。创建新的文件，在该文件中配置一个全局请求类。在发送请求时，使用我们自己的定义的全局请求类。

## 六、部署和上线

### 6.1.计划

1. 多环境 5 - 10min
2. 项目部署上线 1h ✓
  - 原始前端 / 后端项目
  - 宝塔 Linux
  - 容器
  - 容器平台
3. 前后端的联调 —— 跨域 15 min
4. 用户中心项目扩展和规划（优化点） 5 - 10 min

## 6.2.多环境

---

### 6.2.1.多环境的简介以及作用

- 参考文章: [https://blog.csdn.net/weixin\\_41701290/article/details/120173283](https://blog.csdn.net/weixin_41701290/article/details/120173283)
- 本地开发: localhost (127.0.0.1)
- 多环境: 指同一套项目代码在不同的阶段需要根据实际情况来调整配置并且部署到不同的机器上。
- 为什么需要?
  1. 每个环境互不影响
  2. 区分不同的阶段: 开发 / 测试 / 生产
  3. 对项目进行优化:
    1. 本地日志级别
    2. 精简依赖, 节省项目体积
    3. 项目的环境 / 参数可以调整, 比如 JVM 参数
- 针对不同环境做不同的事情。

### 6.2.2.多环境分类

1. 本地环境 (自己的电脑) localhost
2. 开发环境 (远程开发) 大家连同一台机器, 为了大家开发方便
3. 测试环境 (测试) 开发 / 测试 / 产品, 单元测试 / 性能测试 / 功能测试 / 系统集成测试, 独立的数据库、独立的服务器
4. 预发布环境 (体验服): 和正式环境一致, 正式数据库, 更严谨, 查出更多问题
5. 正式环境 (线上, 公开对外访问的项目): 尽量不要改动, 保证上线前的代码是“完美”运行
6. 沙箱环境 (实验环境): 为了做实验

## 6.3.前端多环境实战

---

- 请求地址
  - 开发环境: localhost:8000
  - 线上环境: user-backend.code-nav.cn (更改成实际的域名或ip地址)

```

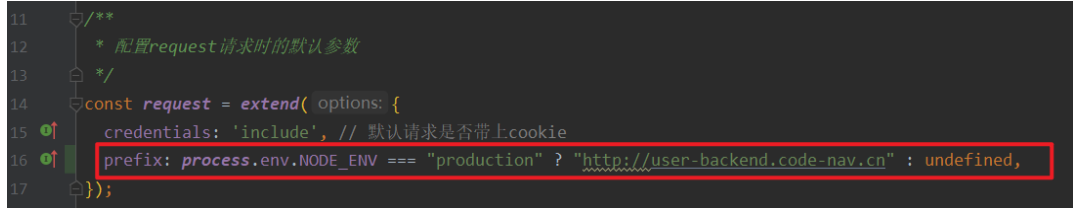
startFront(env) {
  if(env === 'prod') {
    // 不输出注释
    // 项目优化
    // 修改请求地址
  } else {
    // 保持本地开发逻辑
  }
}

```

用了 umi 框架，build 时会自动传入 NODE\_ENV == production 参数，start NODE\_ENV 参数为 development

- 启动方式
  - 开发环境：npm run start（本地启动，监听端口、自动更新）
  - 线上环境：npm run build（项目构建打包），可以使用 serve 工具启动（npm i -g serve）

修改 `src/plugins/globalResponseHandler.ts` 目录下的配置即可



```

11  /**
12   * 配置request请求时的默认参数
13   */
14   const request = extend( options: {
15     credentials: 'include', // 默认请求是否带上cookie
16     prefix: process.env.NODE_ENV === "production" ? "http://user-backend.code-nav.cn" : undefined,
17   });

```

```

/**
 * 配置request请求时的默认参数
 */
const request = extend({
  credentials: 'include', // 默认请求是否带上cookie
  prefix: process.env.NODE_ENV === "production" ? "http://user-backend.code-nav.cn" : undefined,
});

```

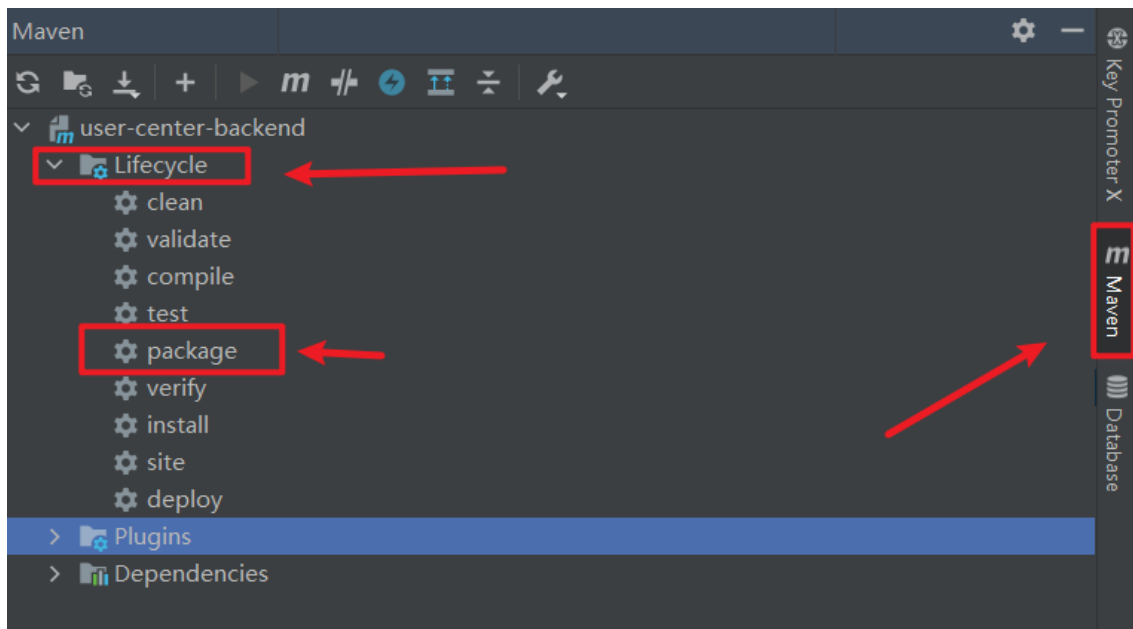
- 项目的配置
 

不同的项目（框架）都有不同的配置文件，umi 的配置文件是 config，可以在配置文件后添加对应的环境名称后缀来区分开发环境和生产环境。参考文档：<https://umijs.org/zh-CN/docs/deployment>

  - 开发环境：config.dev.ts
  - 生产环境：config.prod.ts
  - 公共配置：config.ts 不带后缀

## 6.4.后端多环境实战

- SpringBoot 项目，通过 application.yml 添加不同的后缀来区分配置文件
  - 开发环境：application-prod.yml
  - 生产环境：application-dev.yml
- 可以在启动项目时传入环境变量：



```
java -jar .\user-center-backend-0.0.1-SNAPSHOT.jar --  
spring.profiles.active=prod
```

- 主要是改：
  - 依赖的环境地址
    - 数据库地址
    - 缓存地址
    - 消息队列地址
    - 项目端口号
- 服务器配置

## 6.5.项目部署

- 参考文章: <https://www.bilibili.com/read/cv16179200>
- 需要 Linux 服务器 (建议大家用 CentOS 8+ / 7.6 以上)

### 6.5.1.原始部署

- 什么都自己装

#### (1) 部署前端

- 需要 web 服务器: **nginx**、apache、tomcat
- 安装 nginx 服务器:
  1. 用系统自带的软件包管理器快速安装, 比如 centos 的 yum
  2. 自己到官网安装 (参考文章: <https://zhuanlan.zhihu.com/p/425790769>)

```
# 下载 nginx  
curl -o nginx-1.21.6.tar.gz http://nginx.org/download/nginx-1.21.6.tar.gz  
  
# 解压 nginx  
tar -zxvf nginx-1.21.6.tar.gz  
  
# 进入 nginx 目录
```

```

cd nginx-1.21.6

# 安装 nginx 需要的环境
yum install pcre pcre-devel -y
yum install openssl openssl-devel -y
yum -y install gcc-c++

# 配置文件检查
./configure --with-http_ssl_module --with-http_v2_module --with-stream

# 编译
make
make install

# 查看是否有 nginx 的可执行文件
ls /usr/local/nginx/sbin/nginx

# 配置全局环境变量, shift+g 定位到最后一行, 再按 o 在最后一行插入
vim /etc/profile
在最后一行添加: export PATH=$PATH:/usr/local/nginx/sbin
修改环境变量后执行: source /etc/profile

# 启动nginx
nginx

# 查看启动情况
netstat -ntlp 查看启动情况

# 修改 nginx 配置后, 需要运行如下命令, 才生效
nginx -s reload

```

- 注意 nginx 权限

```

nginx path prefix: "/usr/local/nginx"
nginx binary file: "/usr/local/nginx/sbin/nginx"
nginx modules path: "/usr/local/nginx/modules"
nginx configuration prefix: "/usr/local/nginx/conf"
nginx configuration file: "/usr/local/nginx/conf/nginx.conf"
nginx pid file: "/usr/local/nginx/logs/nginx.pid"
nginx error log file: "/usr/local/nginx/logs/error.log"
nginx http access log file: "/usr/local/nginx/logs/access.log"
nginx http client request body temporary files: "client_body_temp"
nginx http proxy temporary files: "proxy_temp"
nginx http fastcgi temporary files: "fastcgi_temp"
nginx http uwsgi temporary files: "uwsgi_temp"
nginx http scgi temporary files: "scgi_temp"

```

## (2) 部署后端

- java、maven
- 安装:

```

# 安装 java8
yum install -y java-1.8.0-openjdk*

# 安装 maven

```

```
curl -o apache-maven-3.8.5-bin.tar.gz https://dlcdn.apache.org/maven/maven-3/3.8.5/binaries/apache-maven-3.8.5-bin.tar.gz

# 下载源码
git clone xxx 下载代码

# 使用 maven 工具打包项目（跳过测试）
打包构建，跳过测试
mvn package -DskipTests

# 运行 jar 包
java -jar ./user-center-backend-0.0.1-SNAPSHOT.jar --server.port=8080 --spring.profiles.active=prod

# 后台运行 jar 包
nohup java -jar ./user-center-backend-0.0.1-SNAPSHOT.jar --server.port=8080 --spring.profiles.active=prod &
# 使用 jobs 查看用 nohup 运行的项目
jobs
```

### (3) linux 命令拓展

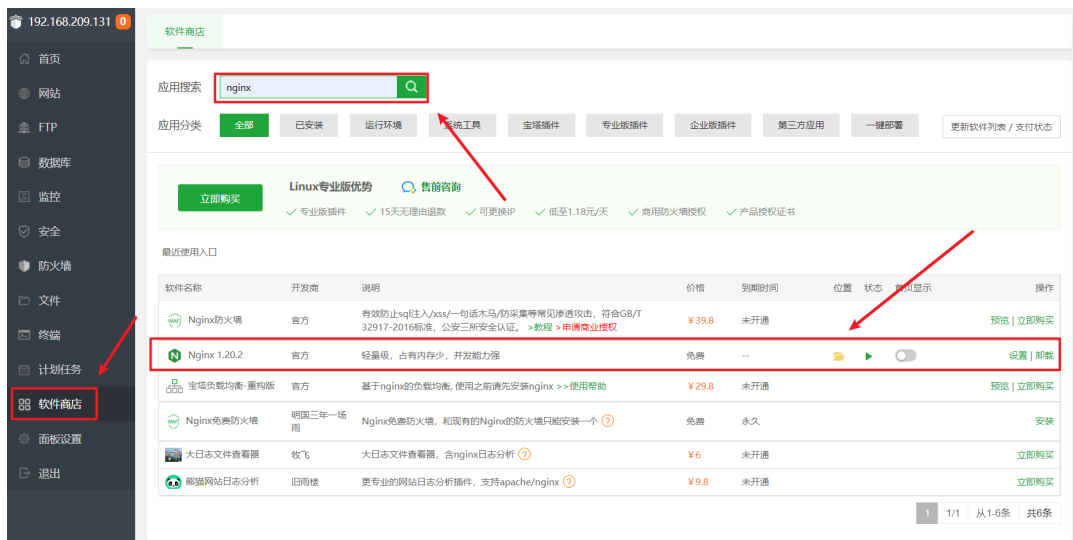
- free -h: 查看系统内存状态
- curl -o 自定义文件名 文件下载路径: 下载文件
- tar -zxvf 文件名: 解压 .tar 压缩包
- history: 得到运行过的历史命令
- unzip 待解压文件 -d 解压后存放路径: 解压 .zip 压缩包
- chmod a+x 文件名: 给文件添加可执行的权限
- jps: 查看所有运行的java程序

## 6.5.2.宝塔 Linux 部署

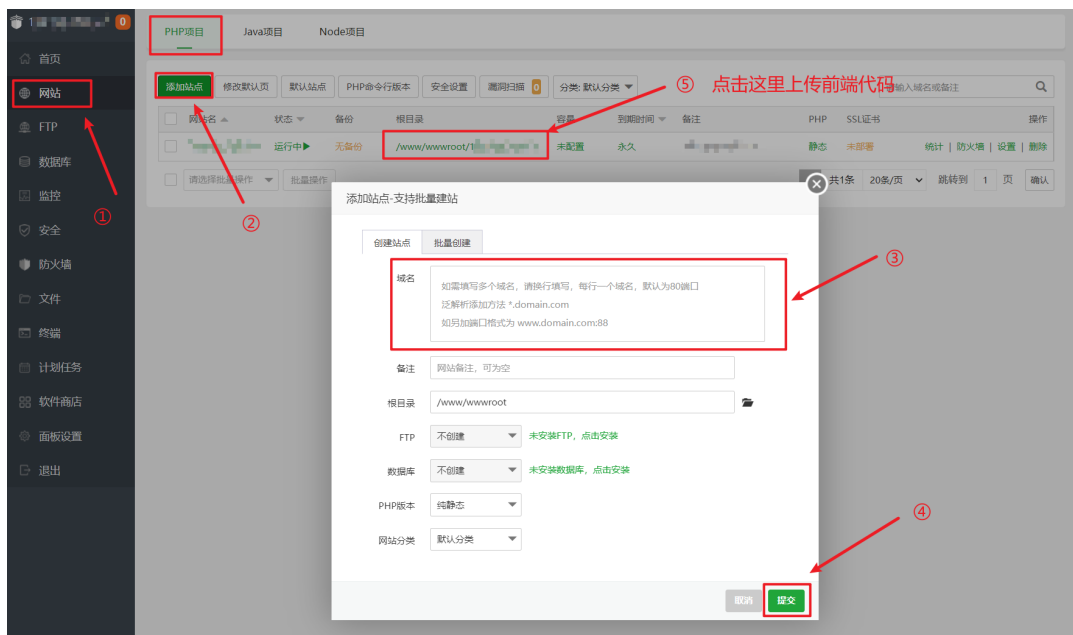
- Linux 运维面板
- 官方安装教程: <https://www.bt.cn/new/download.html>
- 方便管理服务器、方便安装软件

### (1) 前端部署方式

- 方式一: 腾讯云 web 应用托管
  - 前端腾讯云 web 应用托管 (比容器化更傻瓜式, 不需要自己写构建应用的命令, 就能启动前端项目)
  - <https://console.cloud.tencent.com/webify/new>
  - 小缺点: 需要将代码放到代码托管平台上
  - 优势: 不用写命令、代码更新时自动构建
- 方式二: 宝塔 Linux 部署
  - 在宝塔面板软件商店中搜索并安装 nginx

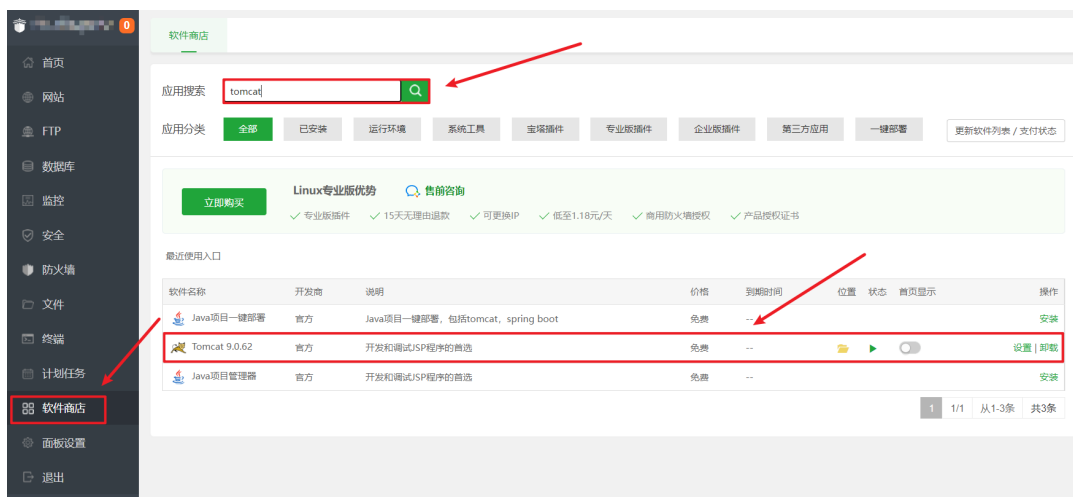


## ○ 添加项目



## (2) 后端部署

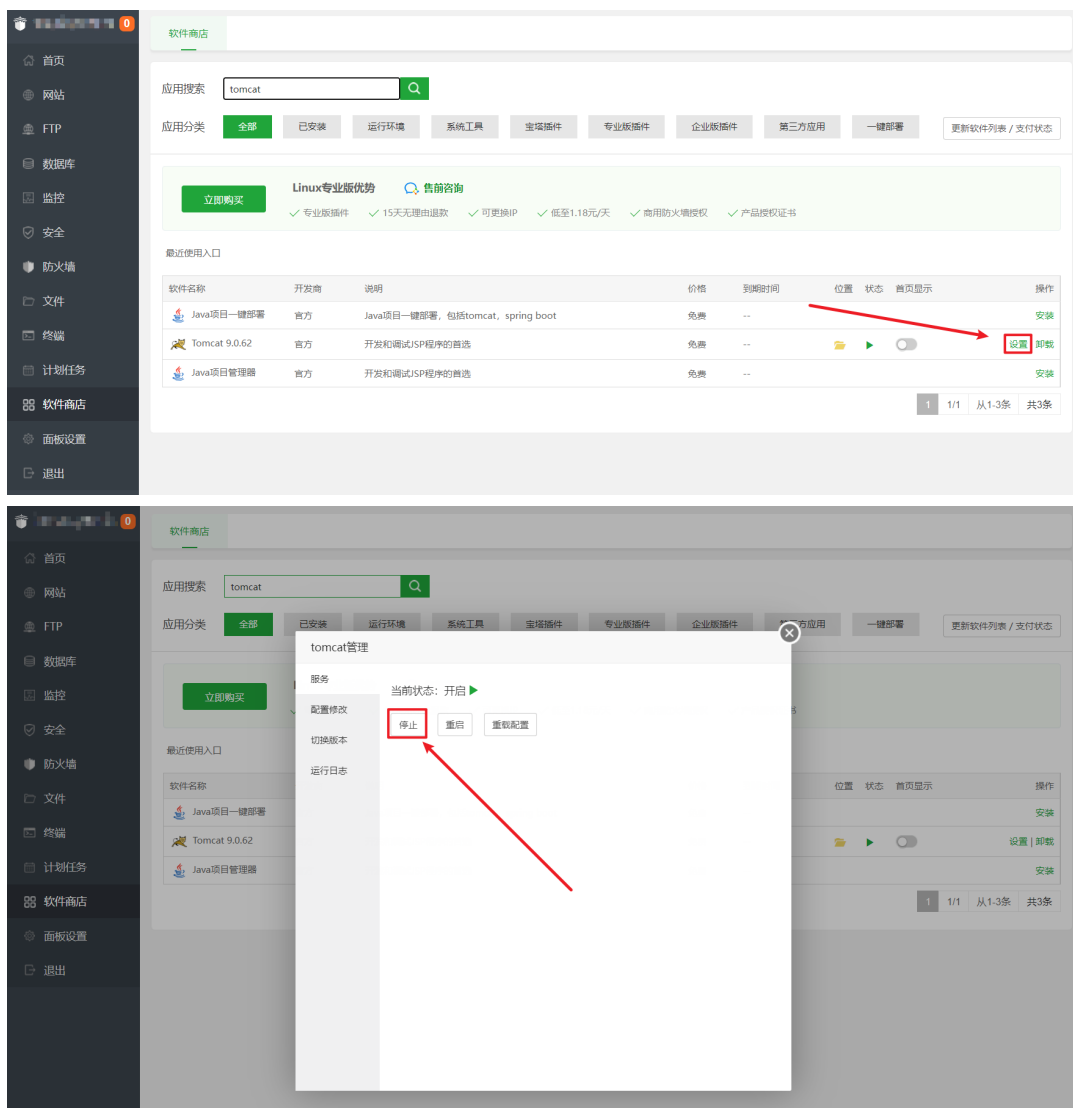
- 方式：宝塔linux
  - 在宝塔面板软件商店中搜索并安装 tomcat



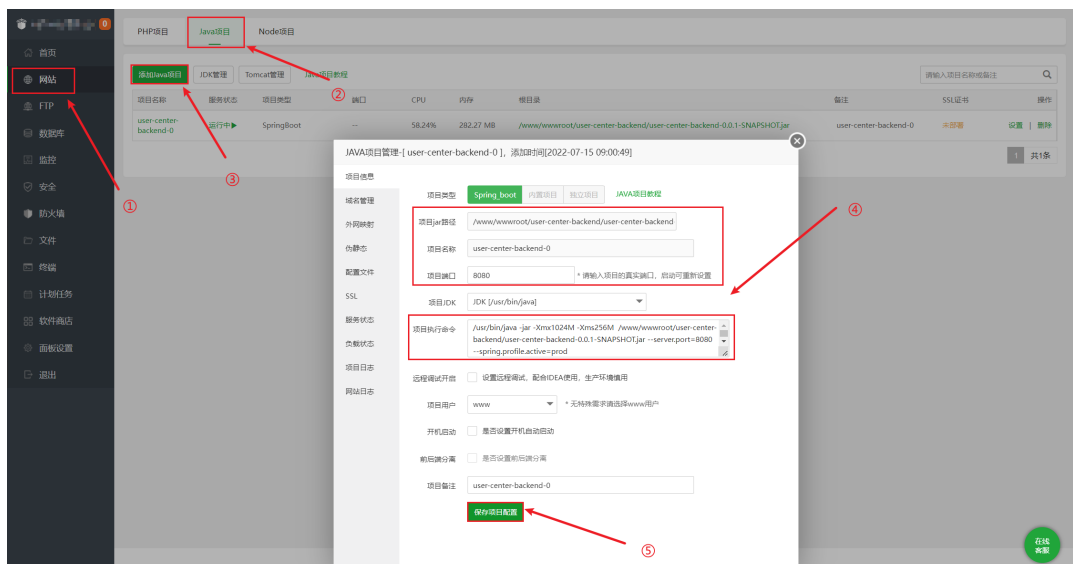
安装 tomcat 的目的是为了安装 java8

- 如果项目中有使用到8080端口，需要将 tomcat关闭





### ○ 添加 java 项目



## (3) Docker 部署

### • docker简介

- docker 是容器，可以将项目的环境（比如 java、nginx）和项目的代码一起打包成镜像，所有同学都能下载镜像，更容易分发和移植。
- 好处：再启动项目时，不需要敲一大堆命令，而是直接下载镜像、启动镜像就可以了。
- docker 可以理解为软件安装包。

- Docker 安装: <https://www.docker.com/get-started/> 或者宝塔安装
- **Dockerfile的简介**
- Dockerfile 用于指定构建 Docker 镜像的方法
- Dockerfile 一般情况下不需要完全从 0 自己写, 建议去 github、gitee 等托管平台参考同类项目 (比如 springboot)
- Dockerfile 编写:
  - FROM 依赖的基础镜像
    - WORKDIR 工作目录
    - COPY 从本机复制文件
    - RUN 执行命令
    - CMD / ENTRYPOINT (ENTRYPOINT可附加额外参数) 指定运行容器时默认执行的命令
- 根据 Dockerfile 构建镜像:
- 后端 Dockerfile

```
FROM maven:3.5-jdk-8-alpine as builder

# Copy local code to the container image.
WORKDIR /app
COPY pom.xml .
COPY src ./src

# Build a release artifact.
RUN mvn package -DskipTests

# Run the web service on container startup.
CMD ["java", "-jar", "/app/target/user-center-backend-0.0.1-SNAPSHOT.jar", "--spring.profiles.active=prod"]
```

- 前端 Dockerfile

```
FROM nginx

WORKDIR /usr/share/nginx/html/
USER root

COPY ./docker/nginx.conf /etc/nginx/conf.d/default.conf

COPY ./dist /usr/share/nginx/html/

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

- cmd 命令

```
# 后端
docker build -t user-center-backend:v0.0.1 .

# 前端
docker build -t user-center-front:v0.0.1 .
```

- Docker 构建优化：减少尺寸、减少构建时间（比如多阶段构建，可以丢弃之前阶段不需要的内容）
- docker run 启动

```
# 前端
docker run -p 80:80 -d user-center-frontend:v0.0.1

# 后端
docker run -p 8080:8080 user-center-backend:v0.0.1
```

- 虚拟化
  1. 端口映射：把本机的资源（实际访问地址）和容器内部的资源（应用启动端口）进行关联
  2. 目录映射：把本机的端口和容器应用的端口进行关联
- **docker 常用操作指令介绍**
- 进入容器

```
docker exec -i -t 容器名/容器id /bin/bash
```

- 查看进程

```
docker ps [-a]
```

- 查看日志

```
docker logs -f [容器名/容器id]
```

- 杀死容器

```
docker kill 容器名/容器id
```

- 强制删除镜像

```
docker rmi -f 容器名/容器id
```

## 6.5.3.Docker 平台部署

### (1) Docker 平台分类

1. 云服务商的容器平台（腾讯云、阿里云）
2. 面向某个领域的容器平台（前端 / 后端微信云托管）**要花钱！**

## (2) 好处

- 容器平台的好处：
  1. 不用输命令来操作，更方便省事
  2. 不用在控制台操作，更傻瓜式、更简单
  3. 大厂运维，比自己运维更省心
  4. 额外的能力，比如监控、告警、其他（存储、负载均衡、自动扩缩容、流水线）

## 6.6.绑定域名

### (1) 域名解析过程

- 前端项目访问：用户输入网址 => 域名解析服务器（把网址解析为 ip 地址 / 交给其他的域名解析服务） => 服务器 => （防火墙） => nginx 接收请求，找到对应的文件，返回文件给前端 => 前端加载文件到浏览器中（js、css） => 渲染页面
- 后端项目访问：用户输入网址 => 域名解析服务器 => 服务器 => nginx 接收请求 => 后端项目（比如 8080端口）
- nginx 反向代理：替服务器接收请求，转发请求

## 6.7.跨域问题解决

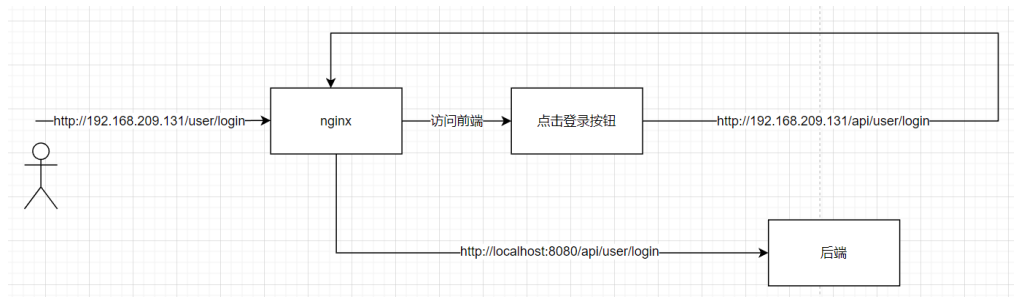
- 浏览器为了用户的安全，仅允许向 **同域名、同端口** 的服务器发送请求。
- 解决跨域：
  - 把域名、端口改成相同的
    - 即前端和后端配置相同的域名以及端口，通过nginx来进行转发。例如当访问 `user.code-nav.cn/user/login` 转发到前端，当访问 `user.code-nav.cn/api/user/login` 转发到后端
    - 修改 `src/plugins/globalResponseHandler.ts` 问价下的请求url

```
/**
 * 配置request请求时的默认参数
 */
const request = extend({
  credentials: 'include', // 默认请求是否带上cookie
  prefix: process.env.NODE_ENV === 'production' ?
    'http://192.168.209.131:80' : undefined,
});
```

- 在 nginx 配置文件中，增加如下配置

```
# 解决跨域问题方式一
location ^~ /api/ {
    proxy_pass http://127.0.0.1:8080/api/;
}
```

- 当用户访问了 `http://192.168.209.131/user/login` 时，后点击了登录按钮，则会请求 `http://192.168.209.131:80` 地址，此时匹配 nginx 中的配置，被转发访问 `http://127.0.0.1:8080/api/user/login`，从而解决跨域问题

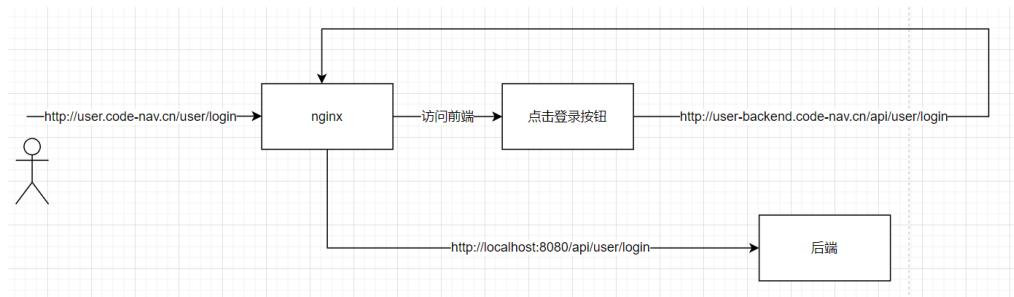


- 让服务器告诉浏览器：允许跨域（返回 Access-Control-Allow-Origin 响应头）

- 网关支持 (Nginx)

```

location ^~ /api/ {
    proxy_pass http://127.0.0.1:8080/api/;
    add_header 'Access-Control-Allow-Origin' $http_origin;
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
    add_header Access-Control-Allow-Headers '*';
    if ($request_method = 'OPTIONS') {
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Origin' $http_origin;
        add_header 'Access-Control-Allow-Methods' 'GET, POST,
OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-
Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range';
        add_header 'Access-Control-Max-Age' 1728000;
        add_header 'Content-Type' 'text/plain; charset=utf-8';
        add_header 'Content-Length' 0;
        return 204;
    }
}
  
```



- 修改后端服务

1. 配置 @CrossOrigin 注解

```

@CrossOrigin(origins = { "http://user.code-nav.cn"}, methods =
{ RequestMethod.DELETE })

@CrossOrigin(origins = { "http://user.code-nav.cn"},
allowCredentials = "true")
  
```

2. 添加 web 全局请求拦截器

```

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
  
```

```

@Override
public void addCorsMappings(CorsRegistry registry) {
    //设置允许跨域的路径
    registry.addMapping("/**")
        //设置允许跨域请求的域名
        //当**Credentials为true时，**origin不能为星号，需为
        具体的ip地址【如果接口不带cookie,ip无需设成具体ip】
        .allowedOrigins("http://localhost:9527",
            "http://127.0.0.1:9527", "http://127.0.0.1:8082",
            "http://127.0.0.1:8083")
        //是否允许证书 不再默认开启
        .allowCredentials(true)
        //设置允许的方法
        .allowedMethods("*")
        //跨域允许时间
        .maxAge(3600);
}
}

```

3. 定义新的 corsFilter Bean，参考：<https://www.jianshu.com/p/b02099a435bd>

## 七、项目优化点

### 1. 功能扩充

1. 管理员创建用户、修改用户信息、删除用户
2. 上传头像
3. 按照更多的条件去查询用户
4. 更改权限

### 2. 修改 Bug

### 3. 项目登录改为分布式 session（单点登录 - redis）

### 4. 通用性

1. set-cookie domain 域名更通用，比如改为 \*.xxx.com
2. 把用户管理系统 => 用户中心（之后所有的服务都请求这个后端）
5. 后台添加全局请求拦截器（统一去判断用户权限、统一记录请求日志）

## 八、拓展

### 8.1.Ant Design Pro（Umi 框架）

- app.tsx 项目全局入口文件，定义了整个项目中使用的公共数据（比如用户信息）
- access.ts 控制用户的访问权限
- 首次访问页面（刷新页面），进入 app.tsx，执行 getInitialState 方法，该方法的返回值就是全局可用的状态值。

### 8.2.Ant Design&Ant Design Procomponents&Ant Design Pro对比

MFSU：前端编译优化

Ant Design 组件库 => React

Ant Design Procomponents => Ant Design

Ant Design Pro 后台管理系统 => Ant Design、React、Ant Design Procomponents、其他的库

## 8.3.ProComponents 高级表单

---

1. 通过 columns 定义表格有哪些列

2. columns 属性

- dataIndex 对应返回数据对象的属性
- title 表格列名
- copyable 是否允许复制
- ellipsis 是否允许缩略
- valueType: 用于声明这一列的类型 (dateTime、select)

链接: <https://procomponents.ant.design/components/table>