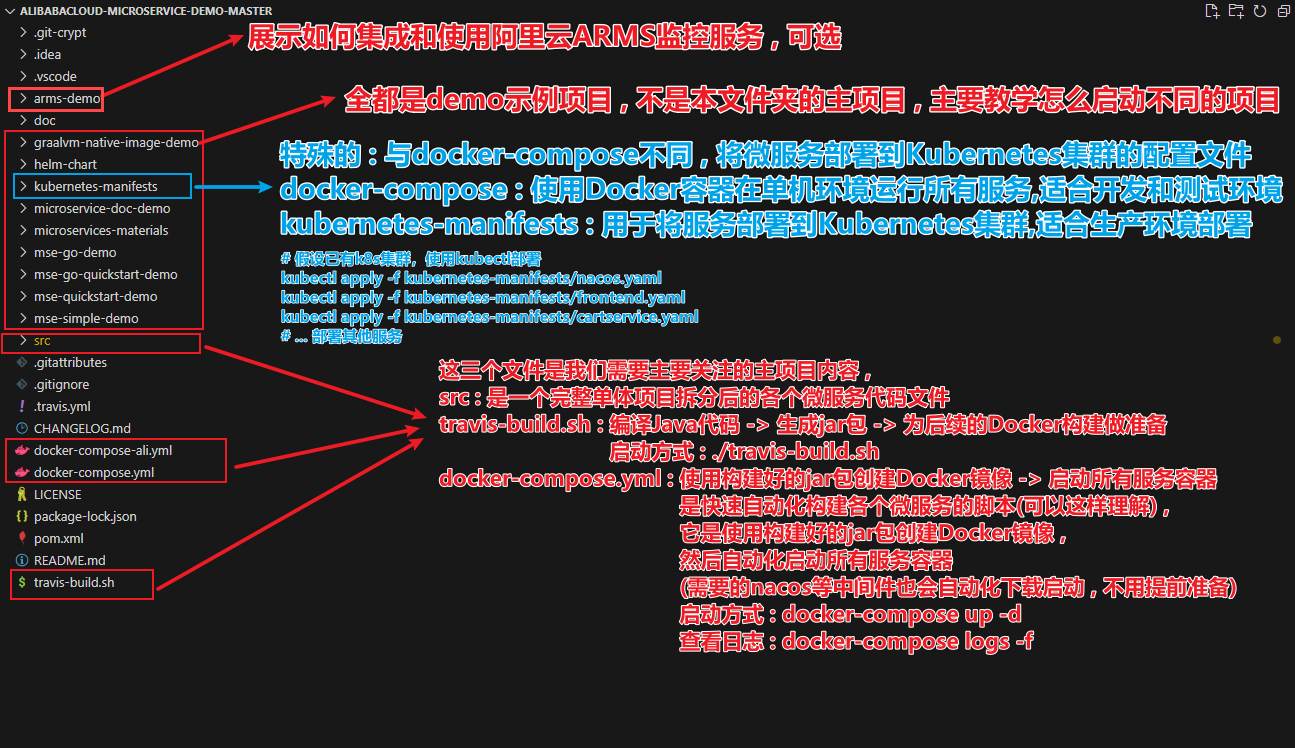
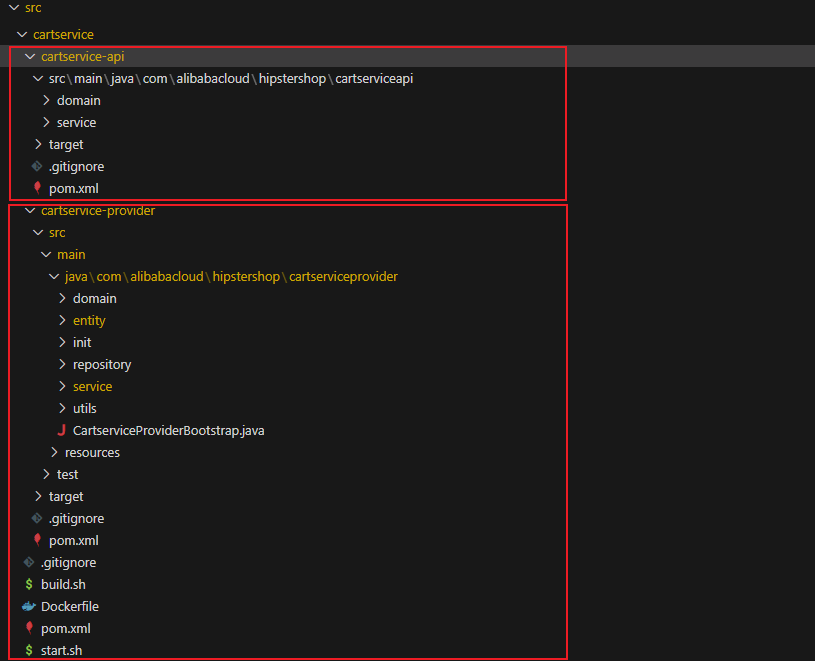
**剖析微服务代码结构与启动:**

**整体:**



===============================================================================

**Src中具体微服务文件 -> 以cartservice为例 (其他微服务都类似)**



分析**cartservice-api文件夹**的结构和功能：

cartservice-api/

├── src/main/java/com/alibaba/hipstershop/cartserviceapi/

│   ├── domain/         # API相关的领域模型

│   └── service/        # 服务接口定义

├── target/            # 编译输出目录

├── .gitignore        # Git忽略文件配置

└── pom.xml           # Maven项目配置文件

详细功能分析：

· domain 目录

· 定义了购物车服务的数据传输对象(DTO)

· 包含接口间传递的请求和响应模型

· 这些对象在服务提供方和消费方之间共享

· 例如：

· CartDTO：购物车数据传输对象

· CartItemDTO：购物车商品项数据传输对象

· CartRequest：购物车操作请求对象

· CartResponse：购物车操作响应对象

· service 目录

· 定义了购物车服务的接口规范

· 包含所有对外暴露的服务方法定义

· 通常包括：

  public interface CartService {

·       *// 添加商品到购物车*

·       CartResponse addItem(CartRequest *request*);

·

·       *// 从购物车移除商品*

·       CartResponse removeItem(CartRequest *request*);

·

·       *// 获取购物车内容*

·       CartDTO getCart(String *userId*);

·

·       *// 清空购物车*

·       void emptyCart(String *userId*);

·

·       *// 更新购物车商品数量*

·       CartResponse updateItemQuantity(CartRequest *request*);

·   }

·

· · pom.xml

· Maven项目配置文件

· 定义项目依赖

· 包含构建配置

· 主要包括：

· 项目基本信息

· 依赖管理

· 编译配置

· 打包配置

重要特点：

· 接口设计

· 采用面向接口编程

· 定义清晰的服务边界

· 支持服务解耦

· 版本管理

· 独立的版本控制

· 支持接口的演进

· 向后兼容性维护

· 依赖隔离

· 最小化依赖

· 只包含必要的接口定义

· 避免实现细节泄露

· 跨服务通信

· 定义服务间通信契约

· 支持RPC调用

· 确保服务兼容性

这个API模块的主要作用是：

· 定义服务接口规范

· 提供数据传输对象

· 实现服务解耦

· 支持分布式部署

· 便于服务治理

通过将API单独抽取为一个模块，可以：

· 更好地管理服务版本

· 简化服务集成

· 提高代码复用性

· 降低服务间耦合

===============================================================================

分析**cartservice-provider文件夹**中的文件结构和功能：

cartservice-provider/

├── domain/         # 领域模型层

├── api/            # API接口定义

├── repository/     # 数据访问层

├── service/        # 业务逻辑层

├── util/           # 工具类

├── CartServiceProviderBootstrap.java  # 启动类

└── test/           # 测试代码

详细解释每个部分的功能：

· domain 目录

· 包含购物车领域的核心业务实体

· 定义了购物车相关的数据模型和业务规则

· 例如 Cart、CartItem 等领域对象

· api 目录

· 定义了购物车服务对外暴露的接口

· 包含接口方法定义和请求/响应对象

· 其他服务通过这些接口与购物车服务交互

· repository 目录

· 负责数据持久化层

· 定义了与数据库交互的接口和实现

· 处理购物车数据的存储和查询操作

· service 目录

· 包含核心业务逻辑的实现

· 实现购物车的添加、删除、更新等操作

· 协调领域对象和数据访问层

· util 目录

· 包含通用工具类

· 可能包含日期处理、字符串处理等辅助功能

· 提供各种通用的工具方法

· CartServiceProviderBootstrap.java

· 服务的启动入口类

· 配置服务启动参数

· 初始化Spring容器和服务注册

· test 目录

· 包含单元测试和集成测试代码

· 验证各个组件的功能正确性

· 确保代码质量和可靠性

主要功能：

· 购物车数据管理

· 创建新购物车

· 添加/删除商品

· 更新商品数量

· 清空购物车

· 数据持久化

· 保存购物车状态

· 读取用户购物车信息

· 管理购物车数据的生命周期

服务集成

· 与商品服务集成

· 与用户服务集成

· 提供RPC接口供其他服务调用

· 性能和可靠性

· 缓存机制

· 异常处理

· 事务管理

· 并发控制

===============================================================================

**下面实践如何部署+运行微服务代码:**

**(经过多次失败 , 得出经验 : 最好使用境外服务器 , 国内网络环境会导致很多地方失败 , 而且最好是弹性服务器 , 配置最少4核8 G 并且 2核4 G服务器也跑不了这个微服务项目)**

**我的服务器配置:**



5.1使用MobaXterm上传代码文件到Ubuntu服务器

可能会报错 , 原因是权限不够 , 可以执行



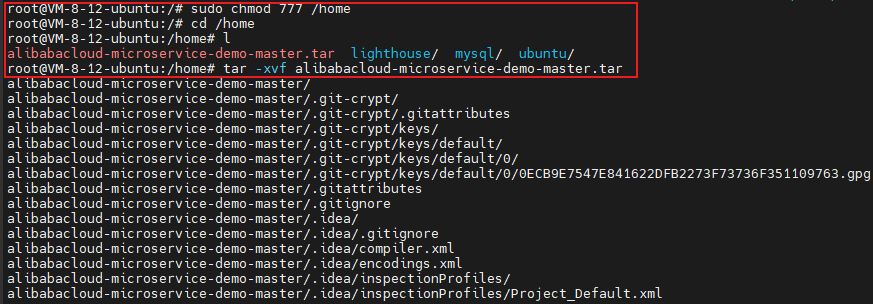
/home 换成自己上传代码的文件夹

再上传代码就可以了

附 : 常见文件权限



5.1.1上传可能比较慢 , 还需耐心等待 , 也可以先在windows上将代码文件打一个tar包 , 上传tar包到服务器上 , 在服务器上执行tar包解压命令即可



亲测 - 打tar包上传再解压的方式超级快 !

5.2 搭建环境并启动微服务项目

使用Docker Compose方式（用于开发/测试）：

5.2.1 搭建环境

因为是新服务器 , 需要先执行一些命令 , 搭建好运行环境 (先使用su命令获取最高权限 , 或者在每次使用命令的时候在命令前面加上sudo)

下面是提前需要执行的命令和一些详细信息/注意事项:

apt-get update

apt-get install openjdk-21-jre : 此处必须使用高版本的java , 亲测openjdk-8-jdk会在之后运行这个微服务项目的时候报错

apt-get install maven : 参考教程[maven 的安装与配置（Command ‘mvn‘ not found）修改配置文件后新终端依旧无法识别到 mvn 命令\_command not found: mvn-CSDN博客](https://blog.csdn.net/Myon5/article/details/136558685)

apt-get install docker.io

apt-get install docker-compose

5.2.2 启动微服务项目

5.2.2.1 操作文件时可能会有权限问题



解决方法(递归设置文件权限):



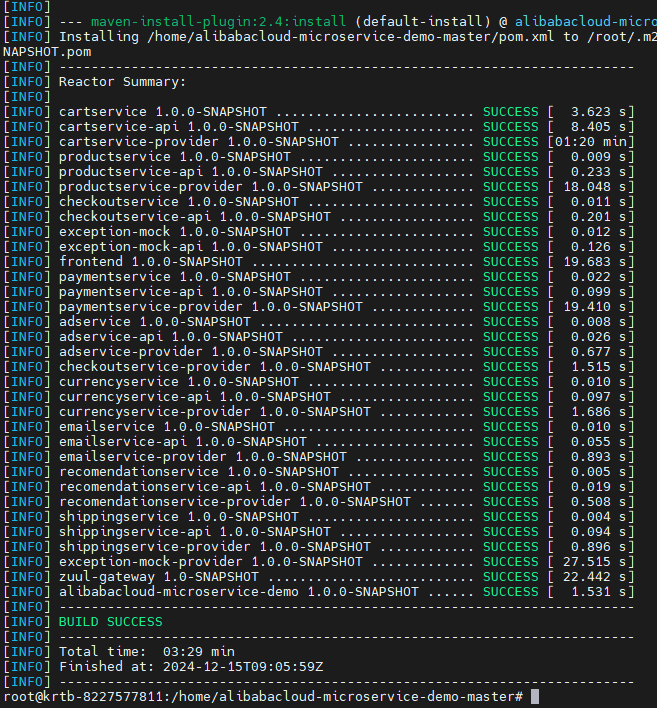
5.2.2.2 启动命令

依次执行下面两个命令

mvn clear : 可能在运行后有一些红色error , 不用管 , 接着执行下面的命令

mvn install

执行mvn install 后 :



5.3. 启动所有服务

执行下面的启动命令:

docker-compose up --build

这种方式会在单机上启动所有服务，包括：

nacos（注册中心）

frontend（前端服务，端口8080）

cartservice（购物车服务）

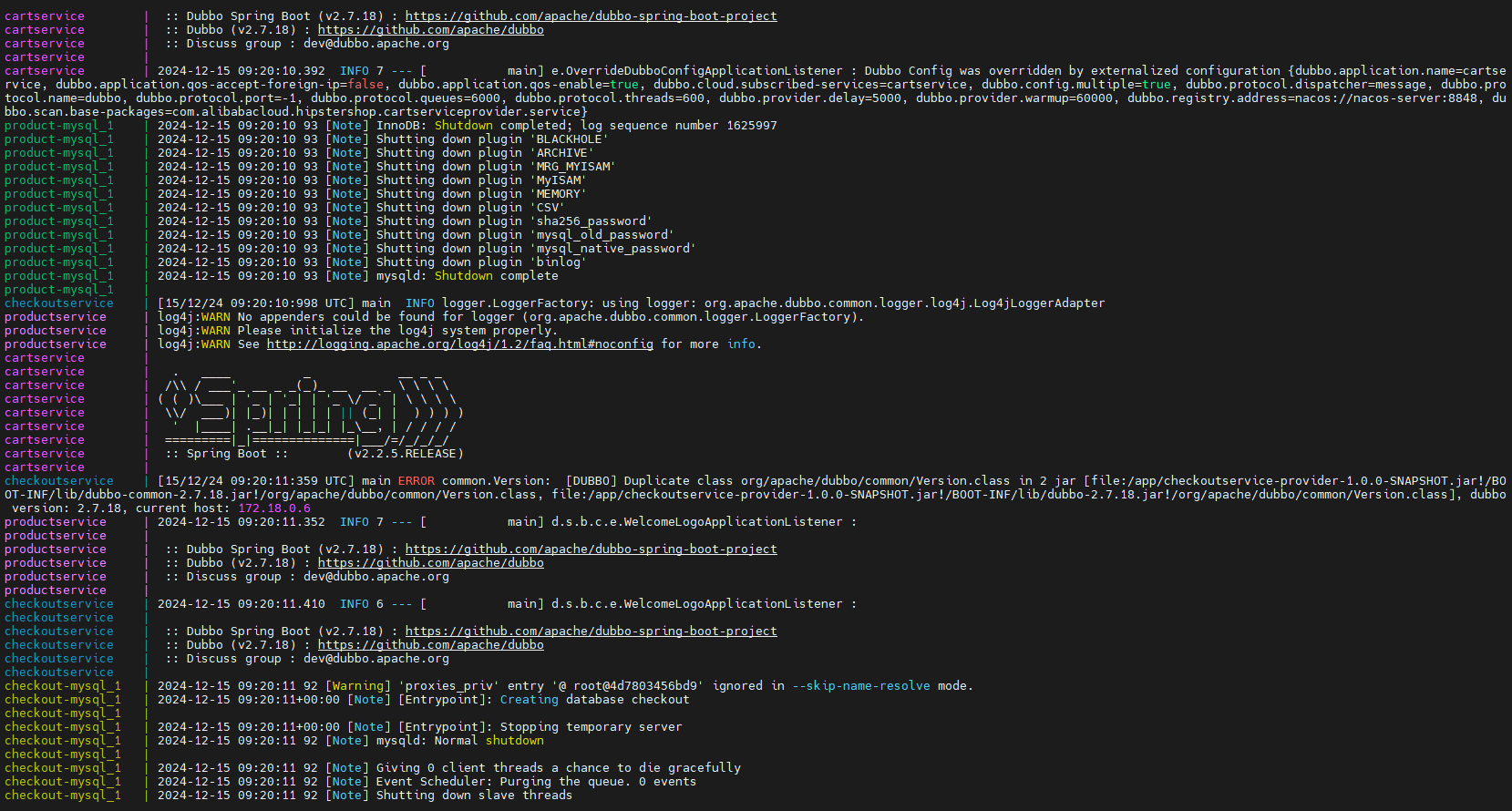
checkoutservice（结账服务）

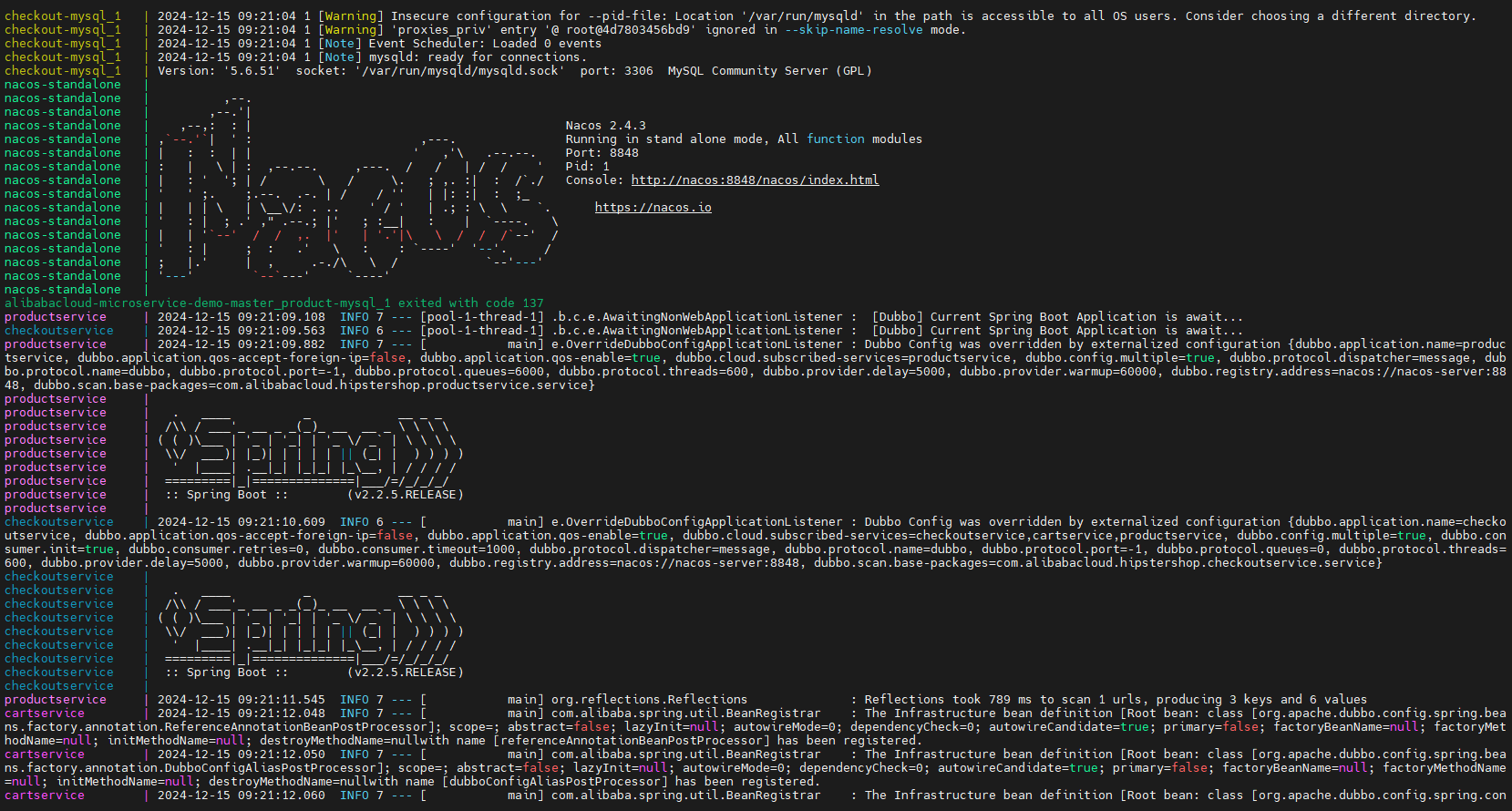
productservice（产品服务）

相关的MySQL和Redis等基础服务

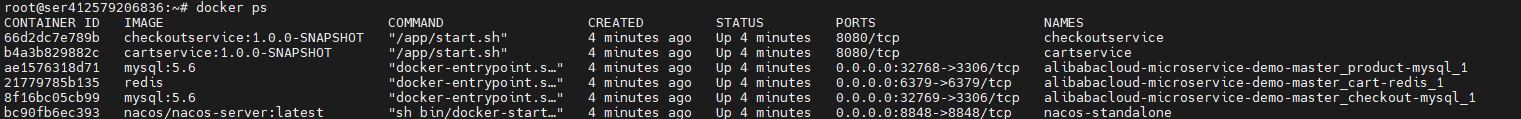
注意事项:

5.3.1. 成功自动化启动所有中间件与微服务:

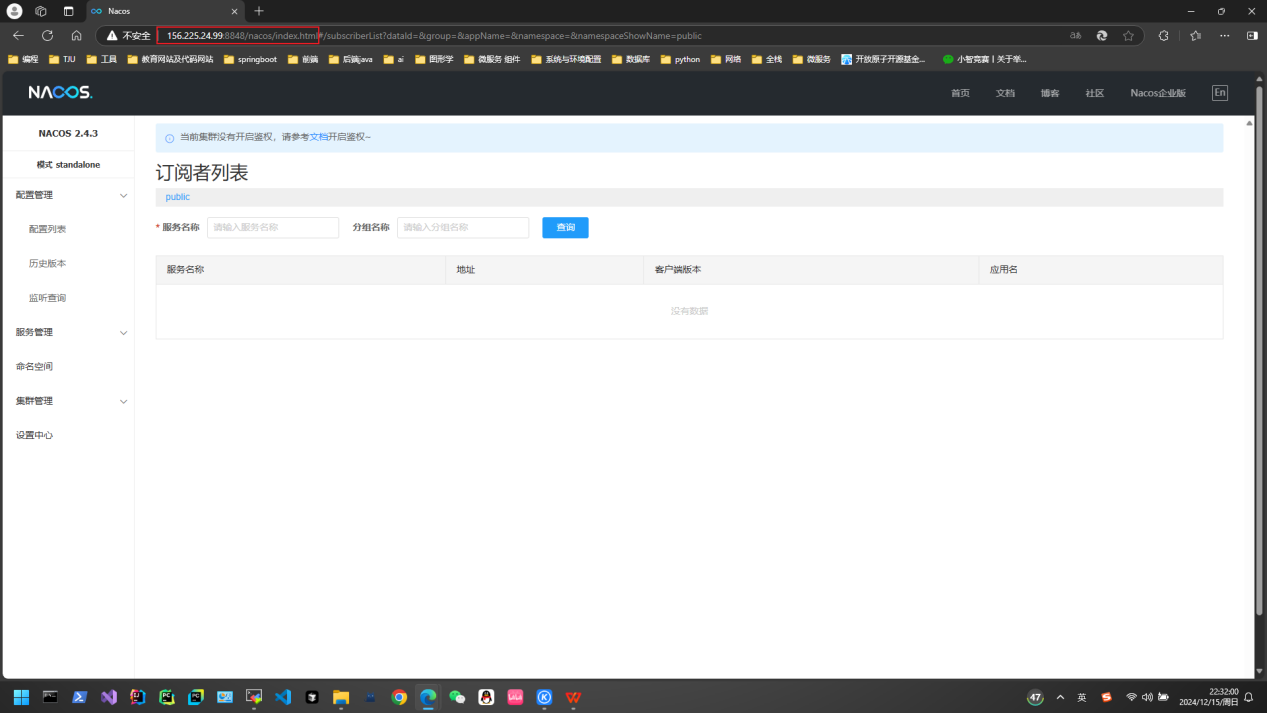




5.4 使用docker ps命令查看当前运行的容器



5.5 在浏览器可以看到部署在服务器上的nacos控制面板



可以看到 , 启动后ubuntu一共占用4.07G内存 , 这就是为什么刚才说4G系统跑不了项目(我也在这里踩了两次坑 , 内存从2G换到4G , 从4G换到8G , 才解决无法启动的问题)

