

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | B/S体系软件设计 |
| 姓 名： | 周炜 |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机科学与技术系 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3210103790 |
| 指导教师： | 胡晓军 |

2023年 10月 10日

**浙江大学实验报告**

课程名称： B/S体系软件设计 实验类型： 课程大作业

实验项目名称： 物联网设备管理平台

学生姓名： 周炜 专业： 3210103790 学号： 3210103790

同组学生姓名： 指导老师： 胡晓军

实验地点： 实验日期： 年 月 日

目录

[1 环境配置 3](#_Toc153703799)

[1.1 客户端 3](#_Toc153703800)

[1.2 前端 5](#_Toc153703801)

[1.3 后端 6](#_Toc153703802)

[1.4 数据库 7](#_Toc153703803)

[2 开发日志 9](#_Toc153703804)

[2.1 客户端 10](#_Toc153703805)

[2.2 前端 12](#_Toc153703806)

[2.3 后端 20](#_Toc153703807)

[2.4 Docker打包 20](#_Toc153703808)

[2.5 Github日志记录 36](#_Toc153703809)

# 环境配置

## 客户端

**Mosquitto**

1. **下载和安装 Mosquitto**：首先，从 Mosquitto官方网站<https://mosquitto.org/download/> 下载并安装 Mosquitto

2. **添加到 PATH**：安装完成后，您需要确保 Mosquitto 的安装目录已添加到系统的 PATH 环境变量中，以便可以在命令提示符中访问它。要执行此操作，您可以按照以下步骤进行操作：

- 在 Windows 桌面上，右键单击“此电脑”或“计算机”，然后选择“属性”。

- 点击“高级系统设置”。

- 在“高级”选项卡上，点击“环境变量”按钮。

- 在“系统变量”部分，找到 PATH 变量，然后点击“编辑”。

- 在变量值中，添加 Mosquitto 安装目录的路径，例如：`C:\Program Files\Mosquitto`。确保不要删除现有的路径，只需在最后加上分号（;）然后添加 Mosquitto 的路径。

- 点击“确定”来保存更改。

更为具体的配置操作，可以参考[网上的资料](https://www.cnblogs.com/dissun/p/10505007.html)

3. **启动 Mosquitto 服务**：在命令提示符中，输入以下命令来启动 Mosquitto 服务：

|  |
| --- |
| mosquitto -v |

这应该会启动 Mosquitto 代理并显示其日志输出。如果一切设置正确，您将看到代理的版本信息和其他日志。

**MVN**

1. **安装 Maven**： 从 Maven 的官方网站下载并安装。安装完成后，确保你知道 Maven 的安装路径。

2. **配置环境变量**： 一旦 Maven 安装完成，你需要配置系统的环境变量，以便系统能够找到 Maven 的执行文件。以下是配置环境变量的步骤：

- 右键点击“此电脑”（或“计算机”），然后选择“属性”。

- 在弹出的窗口中，点击“高级系统设置”。

- 在系统属性窗口中，点击“环境变量”按钮。

- 在“系统变量”部分，找到名为 `Path` 的变量，然后点击“编辑”按钮。

- 在编辑环境变量窗口中，点击“新建”，然后添加 Maven 的 `bin` 目录的路径，例如：`C:\apache-maven-3.6.3\bin`（根据你的 Maven 安装路径进行调整）。

- 确保保存所有更改，然后关闭窗口。

3. **验证Maven安装**：打开一个新的命令提示符窗口，并运行以下命令来验证 Maven 是否正确安装：

|  |
| --- |
| mvn -version |

如果 Maven 已经正确安装和配置，它会显示 Maven 的版本信息和其他相关信息。

4. 这个时候应该就能正确使用老师提供的client程序了

|  |
| --- |
| mvn clean package |

**MySQL workbench**

[下载链接](https://dev.mysql.com/downloads/mysql/)

在本项目中，需要调用SQL文件进行数据库和数据表的建立，因此需要SQL开发环境。

MySQL Workbench是一款用于管理和操作MySQL数据库的集成开发环境（IDE）。它提供了直观的用户界面，使数据库管理员和开发人员能够轻松地执行各种数据库任务，包括创建、修改、删除数据库对象，编写SQL查询，进行性能优化，以及备份和恢复数据。MySQL Workbench还支持数据库建模和设计，使用户能够可视化地设计数据库架构。这个工具是一个强大的数据库管理工具，有助于简化MySQL数据库的管理和开发工作。

**Maven**

[下载链接](https://maven.apache.org/download.cgi)

老师提供的客户端的Java程序依赖于Maven环境， Maven用于构建、管理和维护Java应用程序。它能够自动处理项目依赖关系、构建过程和项目部署，使开发人员能够更轻松地管理复杂的Java项目。Maven的核心思想是约定优于配置，它提供了一套标准的项目结构和构建生命周期，简化了项目配置和管理的过程。

## 前端

**Node.js 19**

建议使用和我相同的nodejs环境，[下载链接](https://nodejs.org/en/download)， 在我的代码调试过程中，我已经发现nodejs16， nodejs17， nodejs18都存在目前我的前端框架不支持的库

**NVM**

[NVM（Node Version Manager）](https://www.freecodecamp.org/news/node-version-manager-nvm-install-guide/)是一个用于管理Node.js版本的工具。它允许用户在同一台计算机上轻松地安装、切换和管理不同版本的Node.js。通过NVM，您可以方便地在项目之间切换Node.js版本，以确保项目的兼容性和稳定性

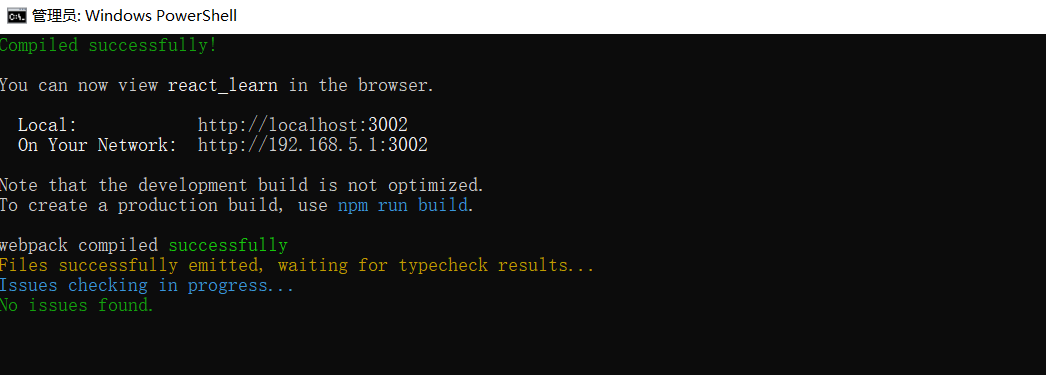
**Yarn**

最后以来于yarn来启动

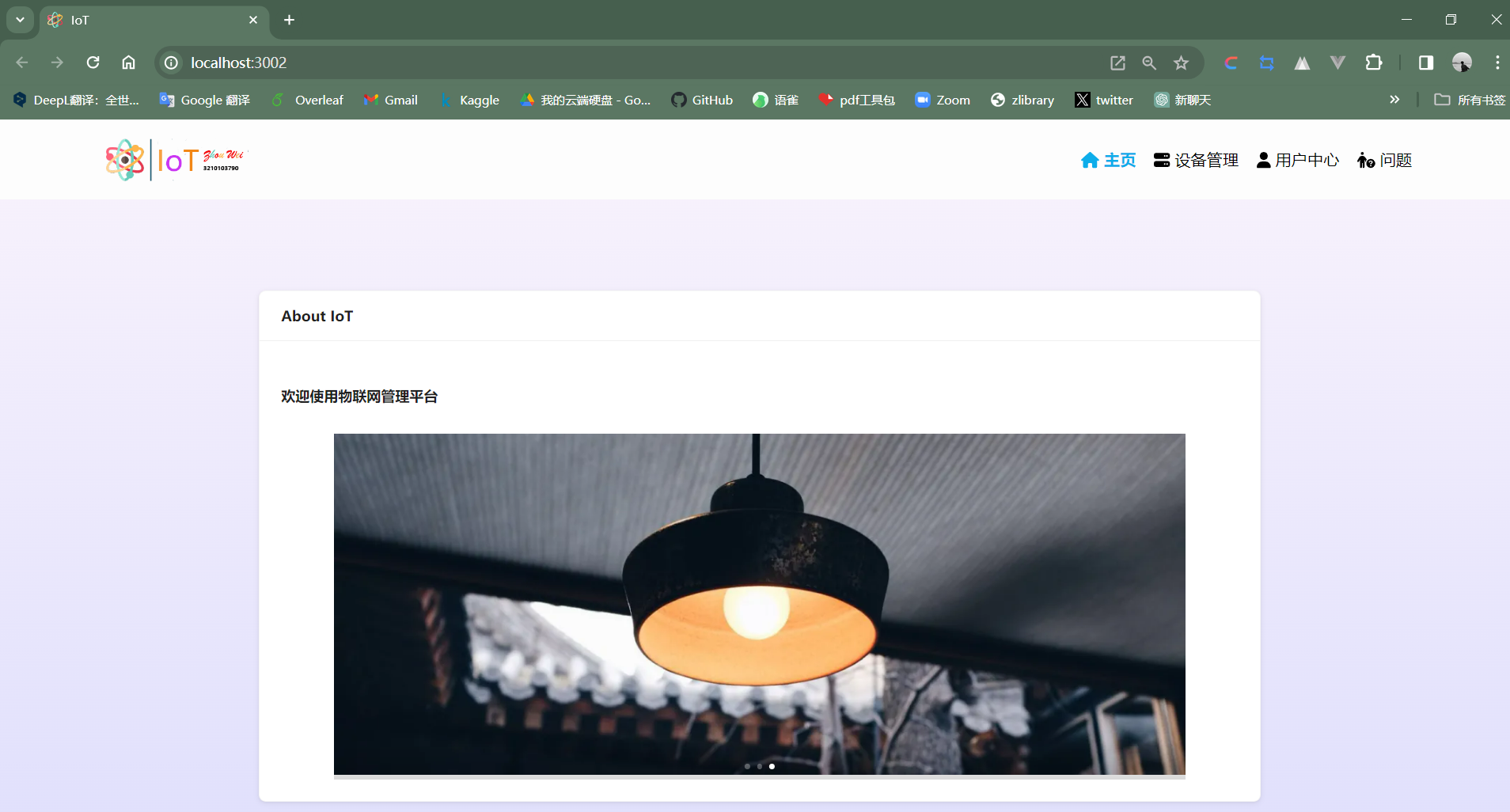
Yarn是一个用于管理JavaScript包依赖的快速、可靠和安全的包管理工具。它是由Facebook开发并维护的，旨在解决npm（Node Package Manager）在性能和安全性方面的一些问题。



编译完成后显示如下



此时输入对应的网页地址即可打开前端程序



## 后端

**Python 3.8**

Python 3.8是一种高级编程语言，它强调代码的可读性和简洁性。它具有丰富的标准库和活跃的社区支持，使得开发人员能够轻松编写功能丰富的应用程序。它的特点包括动态类型、垃圾回收、模块化编程、面向对象编程、以及丰富的第三方库支持，让开发人员能够快速构建各种类型的应用，从命令行工具到Web应用和数据科学项目。Python 3.8引入了一些新特性和改进，使得编程更加便捷和高效。

本程序依赖的包主要有

blinker cffi click colorama cryptography Flask Flask-SQLAlchemy greenlet itsdangerous Jinja2 MarkupSafe paho-mqtt pycparser PyMySQL SQLAlchemy Werkzeug

为了便于安装，我提供了setup.sh文件来进行操作

**Conda**

Conda是一个跨平台的开源包管理和环境管理工具，它用于管理不同版本的软件包和创建独立的Python环境。这使得开发人员能够轻松地组织、安装和管理项目所需的软件包，而不受特定Python版本或操作系统的限制。Conda还支持自动解决依赖关系，使得构建和共享复杂的计算环境变得简单。它广泛用于数据科学、机器学习和科学计算领域，以及其他各种Python应用程序的开发和部署。

可以用下列命令在linux系统上自动进行创建

|  |
| --- |
| wget https://repo.anaconda.com/archive/Anaconda3-2023.07-1-Linux-x86\_64.sh  chmod +x Anaconda3-2023.07-1-Linux-x86\_64.sh  ./Anaconda3-2023.07-1-Linux-x86\_64.sh  echo 'export PATH="/home/zhouwei/anaconda3/bin:$PATH"' >> ~/.bashrc  source ~/.bashrc  conda --version  conda create --name iot python=3.8  conda activate iot  pip install --upgrade pip |

配置好环境后，后端的运行效果如下图



## 数据库

在使用老师所给的Java程序进行收发信息之前，我先建立了数据库， 代码如下，其中device\_info数据表与老师的发送格式完全一致，但是

额外增加了一项userId用来区分所属于的不同user

create database iot;

use iot;

create table device\_info

(

    id          int unsigned auto\_increment primary key,

    user        varchar(128) default '' not null,

    clientId        varchar(128) default '' not null,

    name        varchar(128) default '' not null,

    description varchar(128),

    create\_time datetime     default CURRENT\_TIMESTAMP not null

);

create table device\_message

(

    id        int unsigned auto\_increment   primary key,

    alert     int           default 0                 not null,

    clientId  varchar(128)  default ''                not null,

    info      varchar(128)  default ''                not null,

    lat       float         default 0.0000            not null,

    lng       float         default 0.0000            not null,

    timestamp datetime      default CURRENT\_TIMESTAMP not null on update CURRENT\_TIMESTAMP,

    value     int           default 0                 not null,

    userId    int           default 0                 not null

)charset = utf8;

CREATE TABLE user\_info

(

    id       INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

    email    VARCHAR(128) DEFAULT '' NOT NULL UNIQUE,

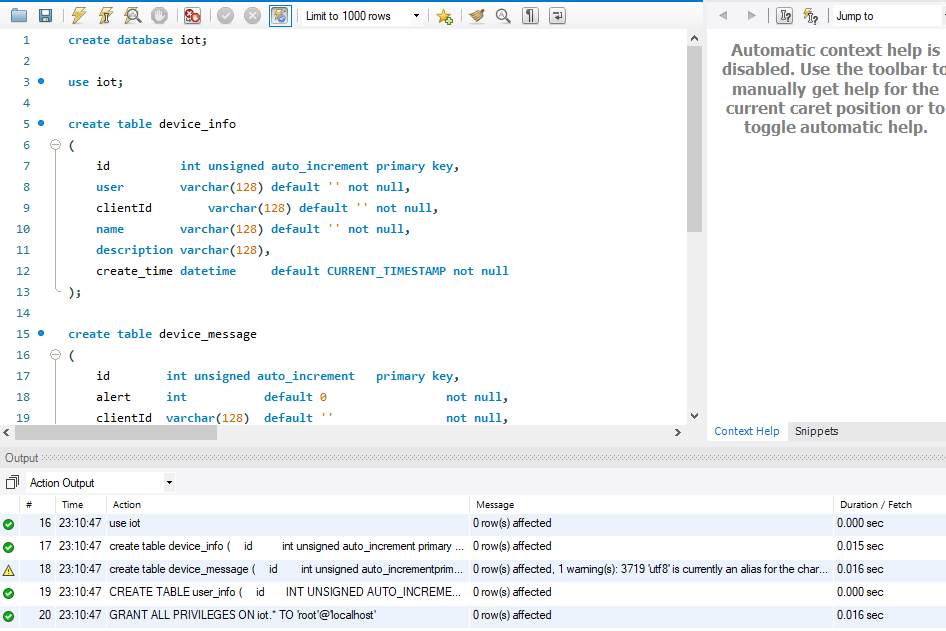
    name     VARCHAR(128) DEFAULT '' NOT NULL UNIQUE,

    password VARCHAR(128) DEFAULT '' NOT NULL

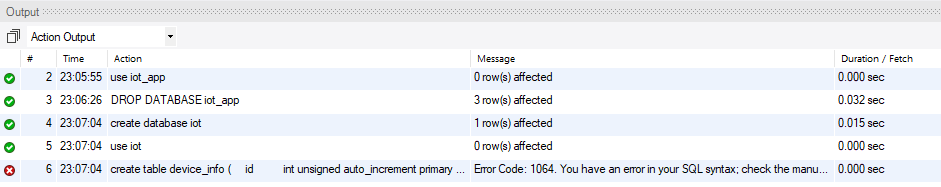
);

GRANT ALL PRIVILEGES ON iot.\* TO 'root'@'localhost';

将上述SQL命令直接复制粘贴进MySQL workbench进行建表**(注意需要在root下进行建表)**



如果不小心建立错误

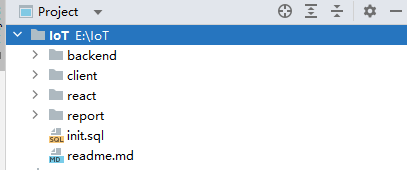


则删除数据库重来



# 开发日志

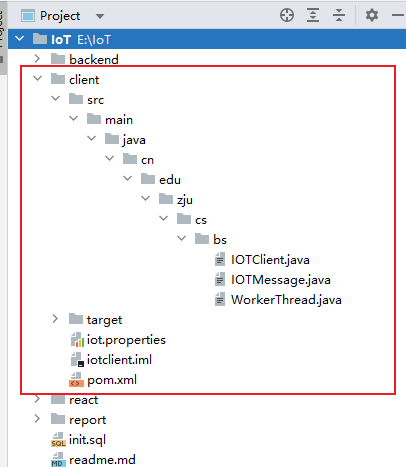
文件整体建构如下



**下列的开发日志，我主要介绍了文件建构和主要的修改细节，其余细节不再列出**

## 客户端

客户端主要使用了老师提供的java的程序用于模拟发送message信息

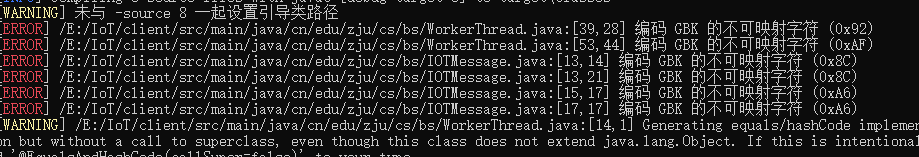


一些主要的修改细节如下

**11.19**

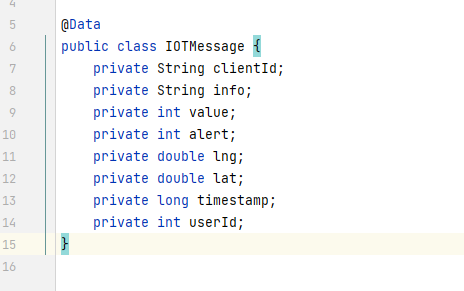
根据老师提供的代码进行了一定的修改，主要如下，解决了一个java编译过程中的报错





**11. 31**

为了区分不同用户的设备，我在老师的java程序中额外增加了一项userID



和value一样，由随即数产生

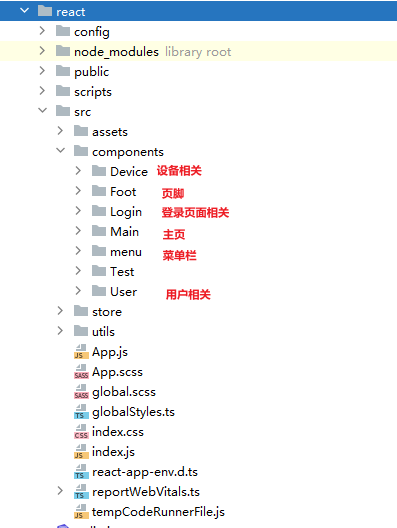


## 前端

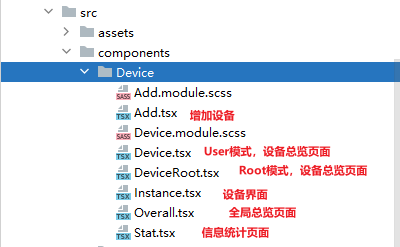


前端的代码文件结构如上图所示，是一个标准的React的架构

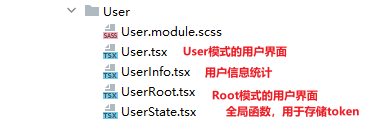
接下来让我们看一下具体的元素



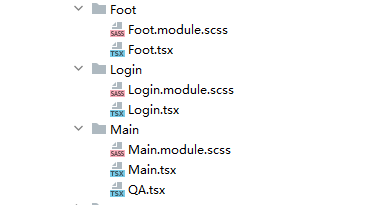
首先介绍Device相关的内容



然后是用户相关的内容



页脚，登录和主页则非常简单，由一个scss文件控制格式，以及一个主文件



导航栏则是由一个主导航和侧边导航组成

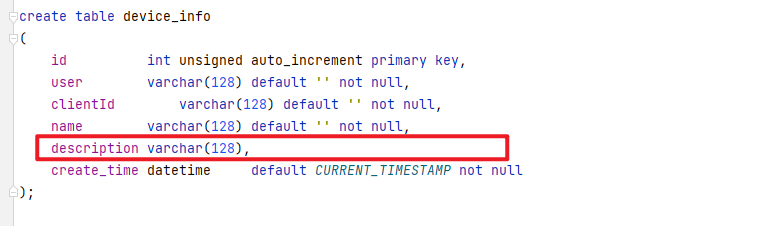


**11. 15**

修改代码使得能够支持物联网设备描述为空的情况



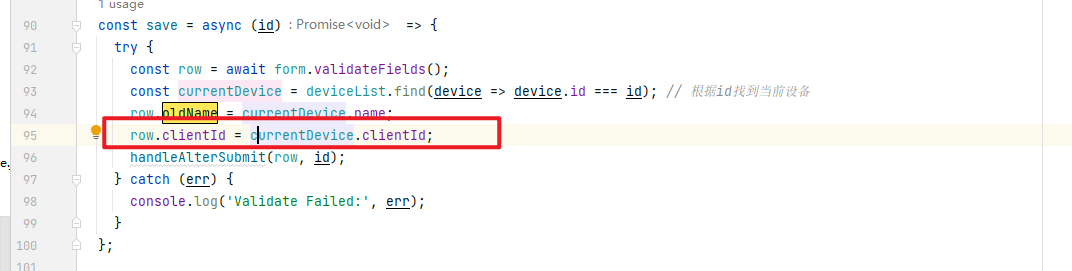
使之符合SQL中的定义



**11. 16**

修正一个bug，原先忘记加了row.clientId，导致点击修改后找不到修改对象





另外一个bug是token已经被我事先拆分了，因此实际上需要组合一下



修改为如下

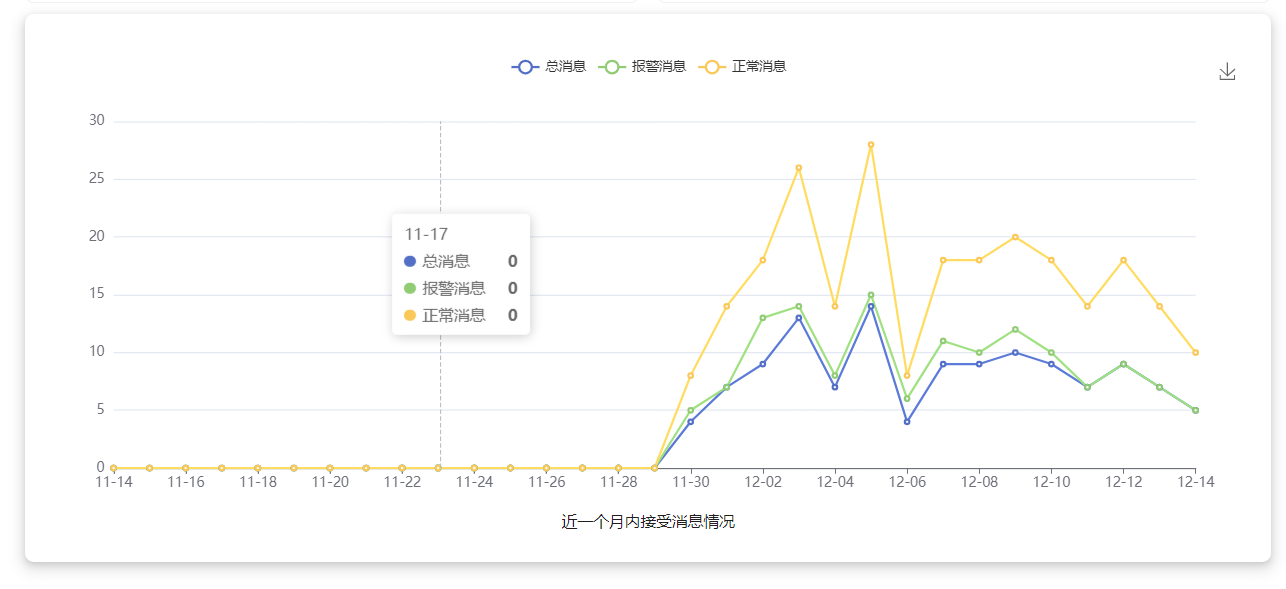


**12. 14**

Root模式下不显示消息



但是用户却能显示



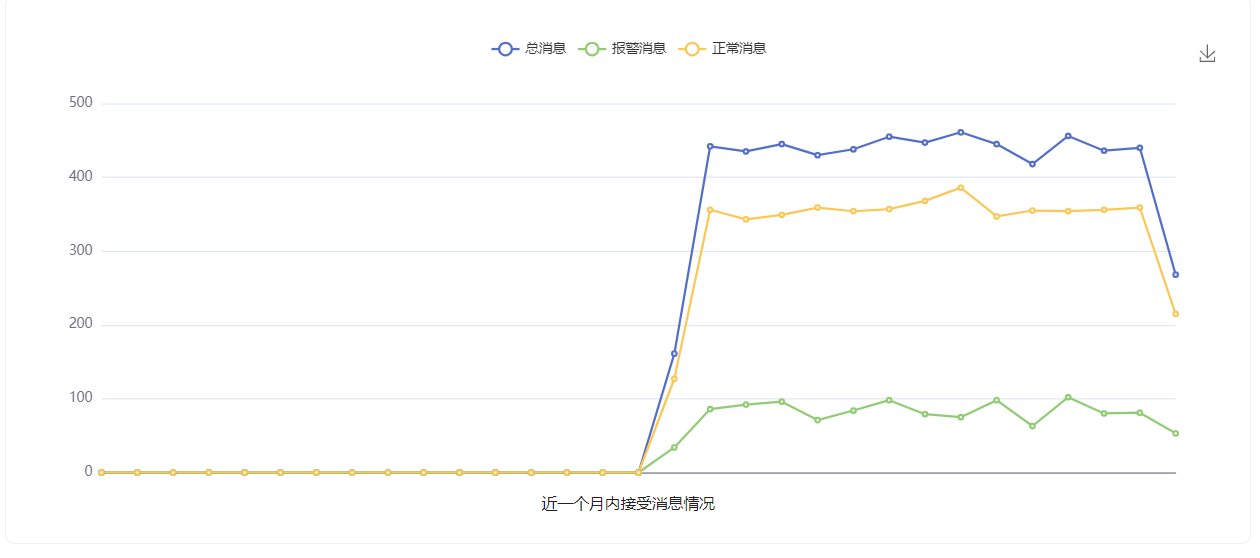
最后定位问题出现在代码中没有按照日期来筛选



应该修改为



解决了这个问题



**12. 15**

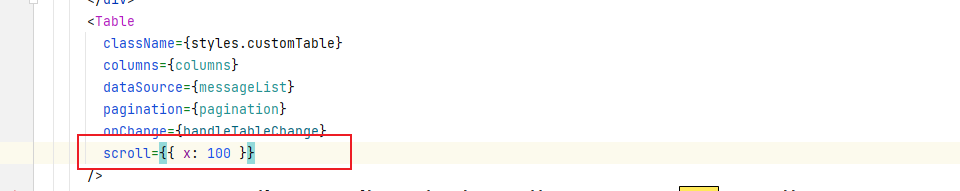
增加自动退出功能，当时间满60分钟，token失效之后，网站的登录状态自动退出



发现手机端表格会突出去一些，调试了字体和百分比后都没用



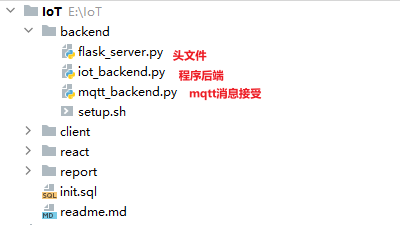
最后通过添加压缩解决了这一问题





## 后端

后端的代码文件结构如下， 详细的介绍可以见[**使用手册**]， 我在其中写了详细的接口说明



**11. 1**

完成mqtt服务器的设置，用于接受老师所发的消息



## Docker打包

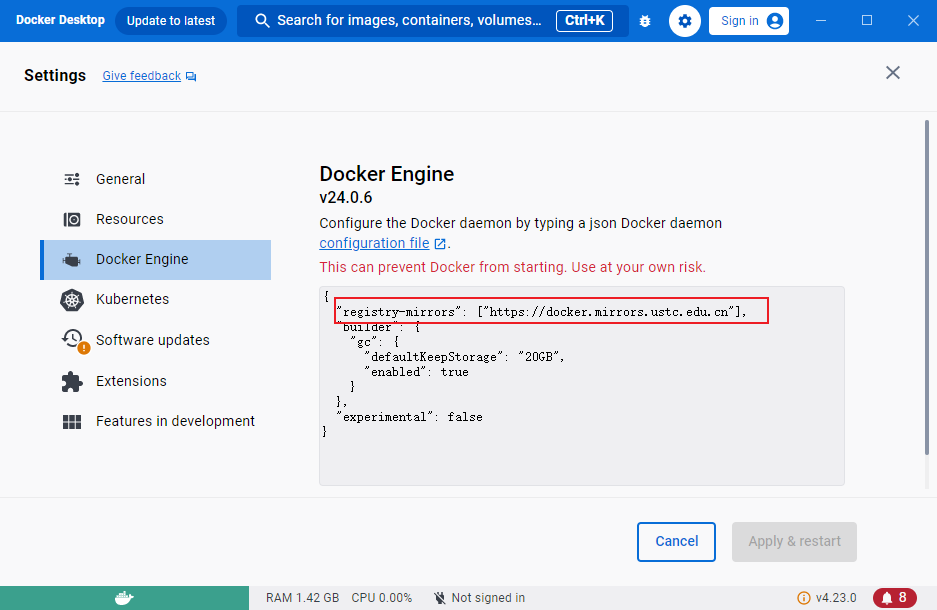
这里使用了docker-compose对项目进行打包，首先需要编写docker-compose.yml文件

Docker打包的时候可能会出现无法连接的情况，需要如下图所示进行换源

无法连接：

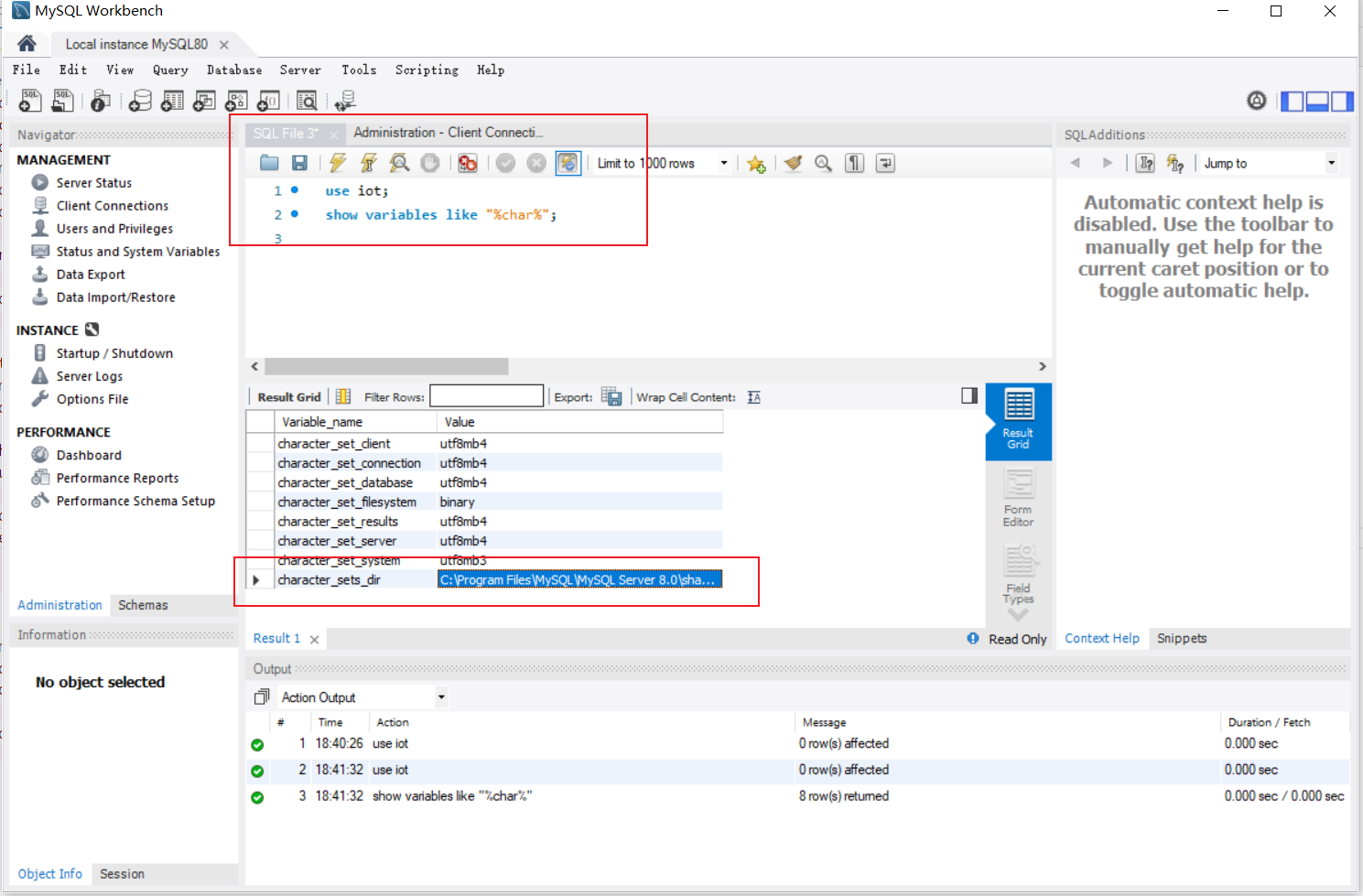


在docker desktop中进行换源



**MySQL打包**

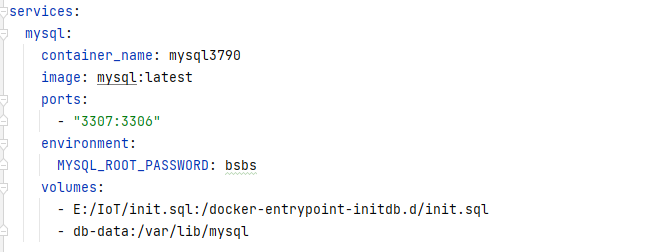
在MySQL Workbench中，用如下命令来查看本地的数据库位置，用于建立映射关系



并且要注意把sql文件中的权限改为所有主机可以访问



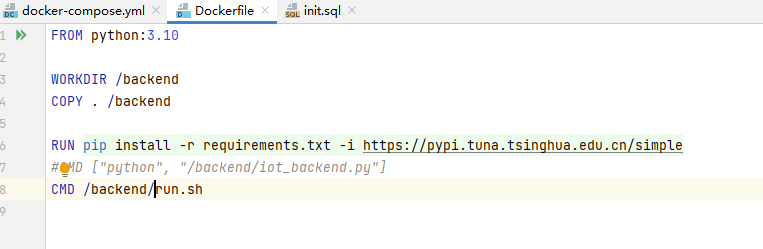
docker-compose.yml中编写为



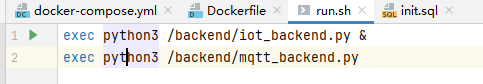
**Python打包**

首先可以使用where python来找到本地的python环境，然后列出requirements

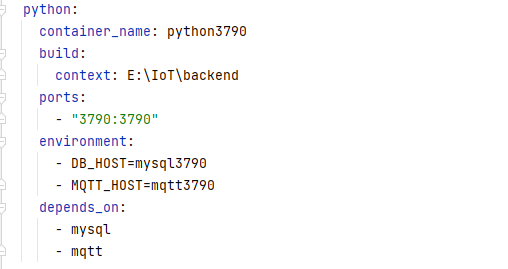
然后由于我的后端有mqtt和对应react的两个后端，但是CMD只能运行一个脚本，因此我把Dockerfile写成了如下形式：



其中run.sh为



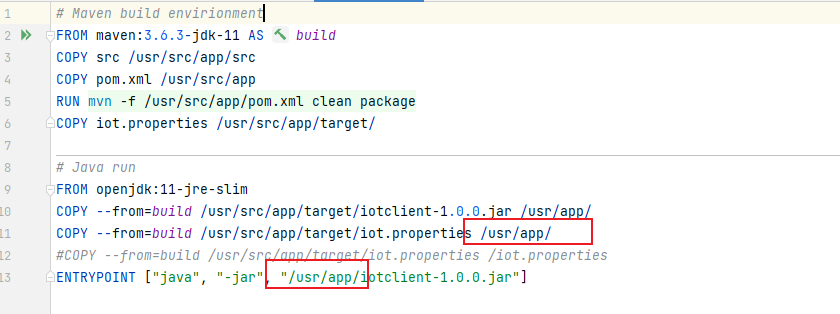
docker-compose.yml中编写为



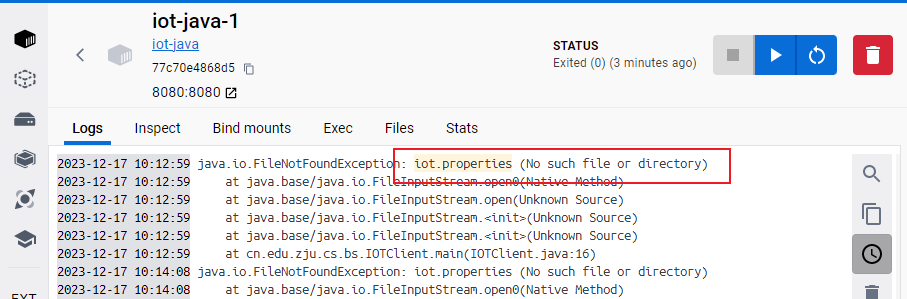
**Mqtt和java打包**

注意iot.properties存放的位置

如下图所示的简单的把iotclient-1.0.0.jar, iot.properties两个文件放在同一/usr/app/目录下会报错，因为Java程序默认在其启动目录（通常是容器的根目录）中查找iot.properties文件



会报错为



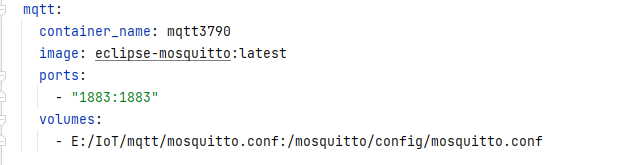
需要更改iot放置位置，并且拷贝所需依赖



还需要对老师提供的代码中的运行位置进行一定更改以适应docker环境

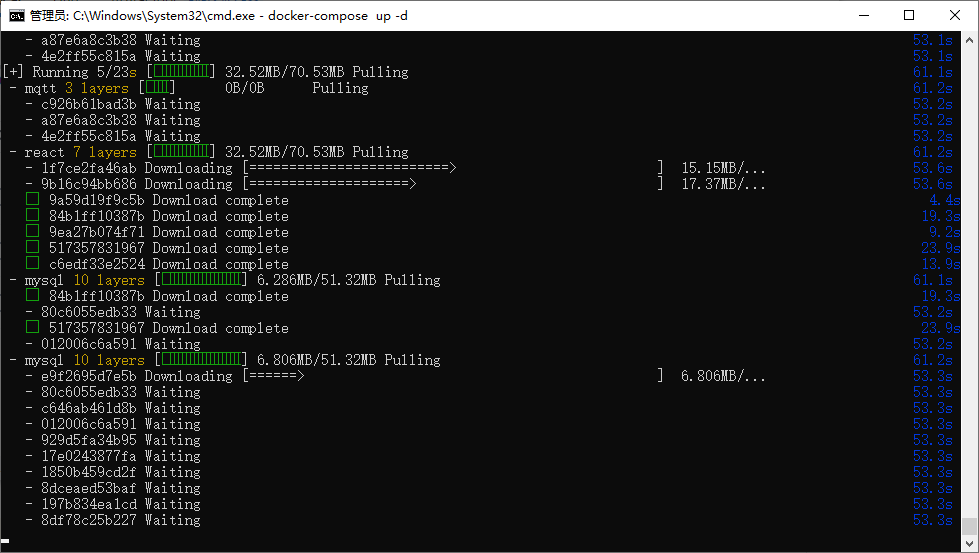


docker-compose.yml中编写为

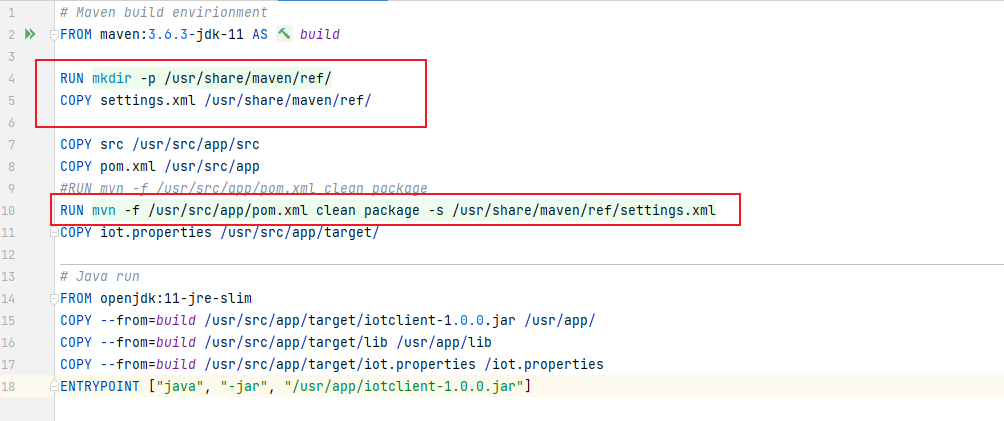




之后docker就会自动进行打包



如果maven速度过慢，还可以按照下面的方式进行换源



settings.xml为



**React打包**

首先使用yarn build生成静态文件，然后docker-compose.yml中编写为



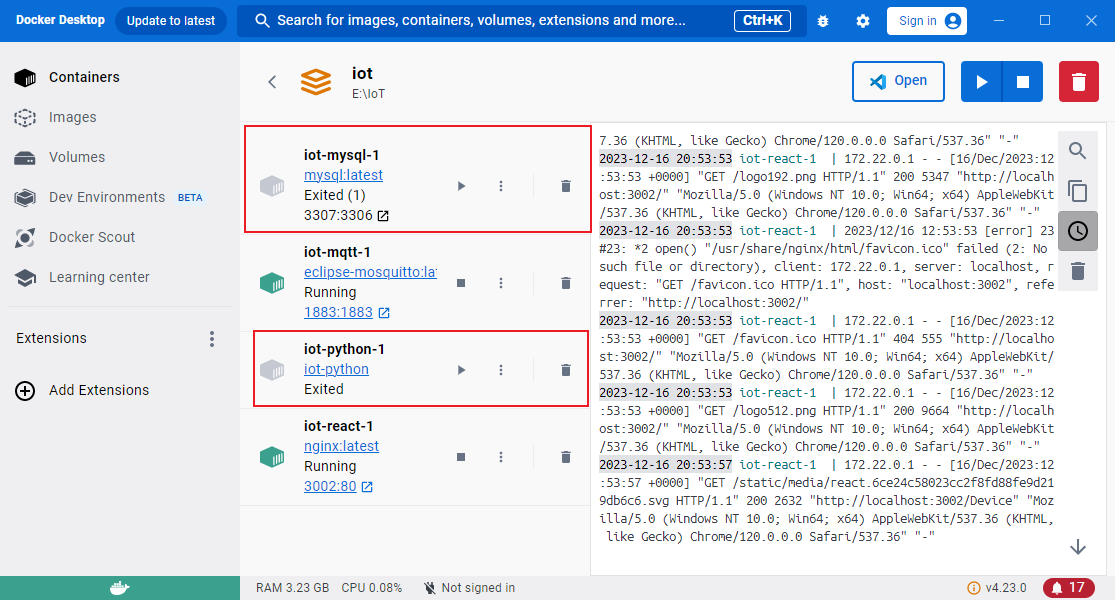
为了防止nginx路由丢失，需要编写一份配置文件



**Debug**

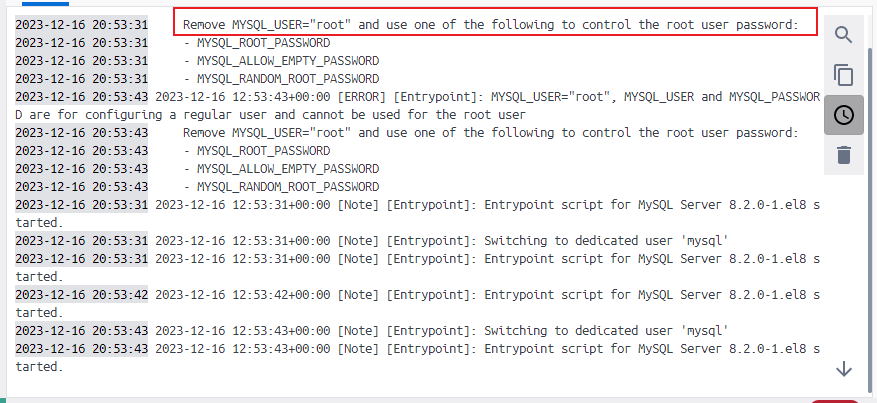
之后可以运行docker容器

但是发现其中的两个容器还有问题



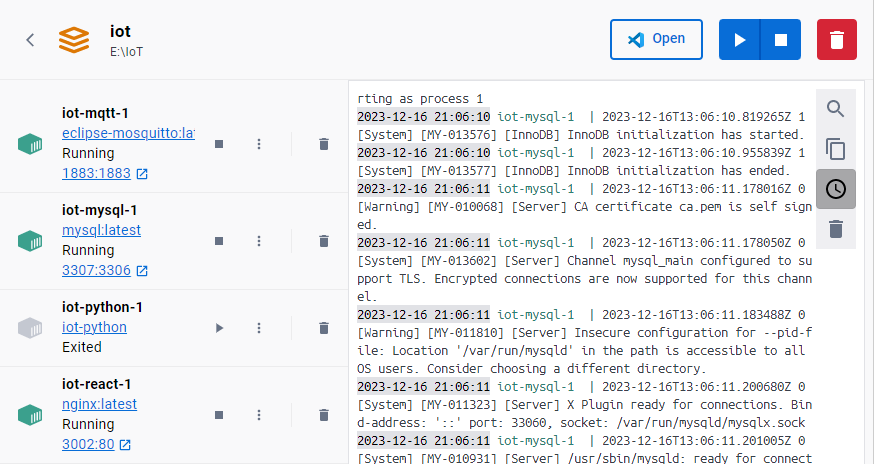
需要对yml文件进行一些修正

首先数据库中报错为



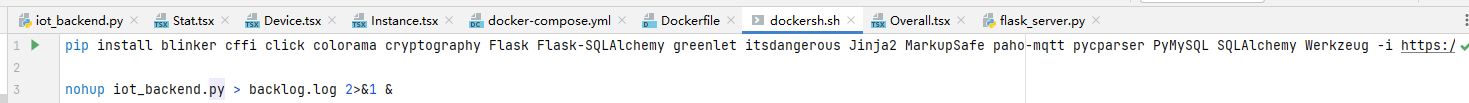
将代码改为如下形式，docker即可正常运行



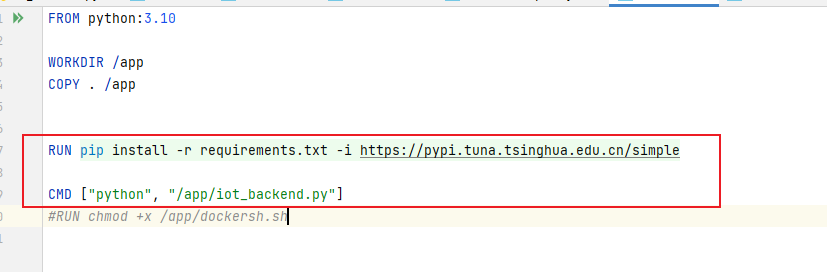


后端程序无法运行可能是使用的nohup的问题，以及使用sh命令不会让程序一直运行

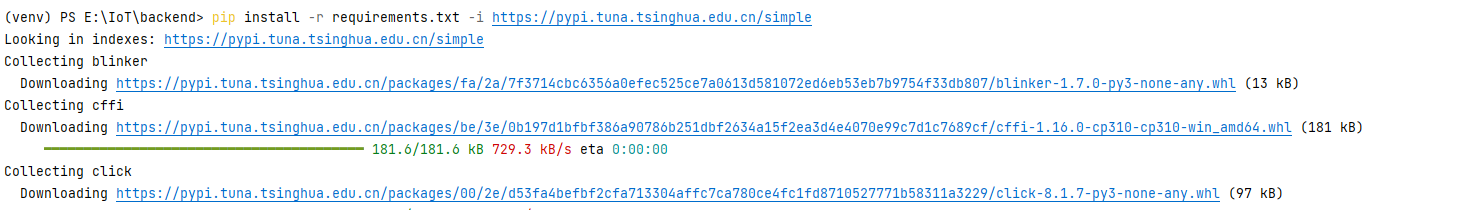
删除dockersh.sh



对Dockerfile做如下更改



并且通过本地新建一个虚拟环境测试发现

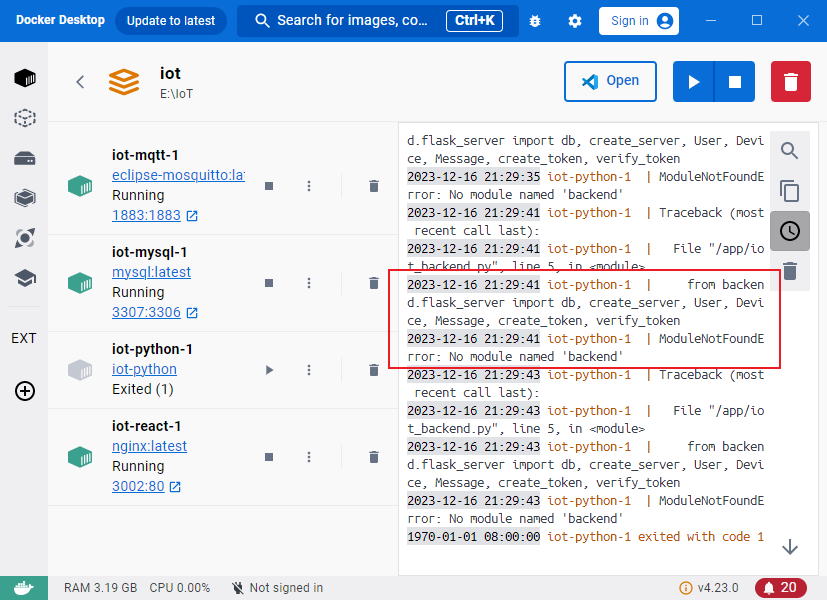


发现requirements.txt少写了一个库

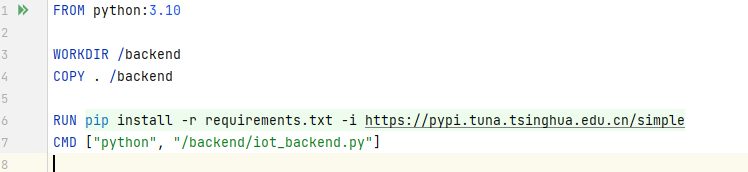


手动安装后容器运行正常

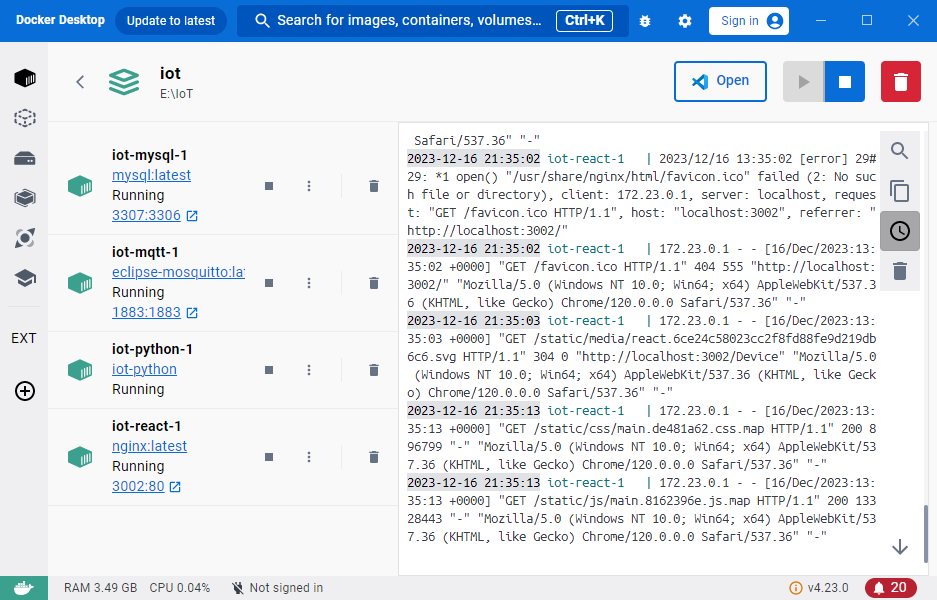
并且进一步检查后发现，文件夹名称也有错误



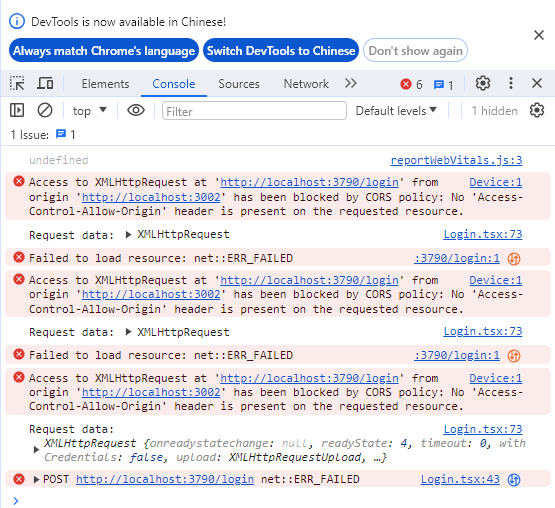
应该把工作区名称由app改为backend



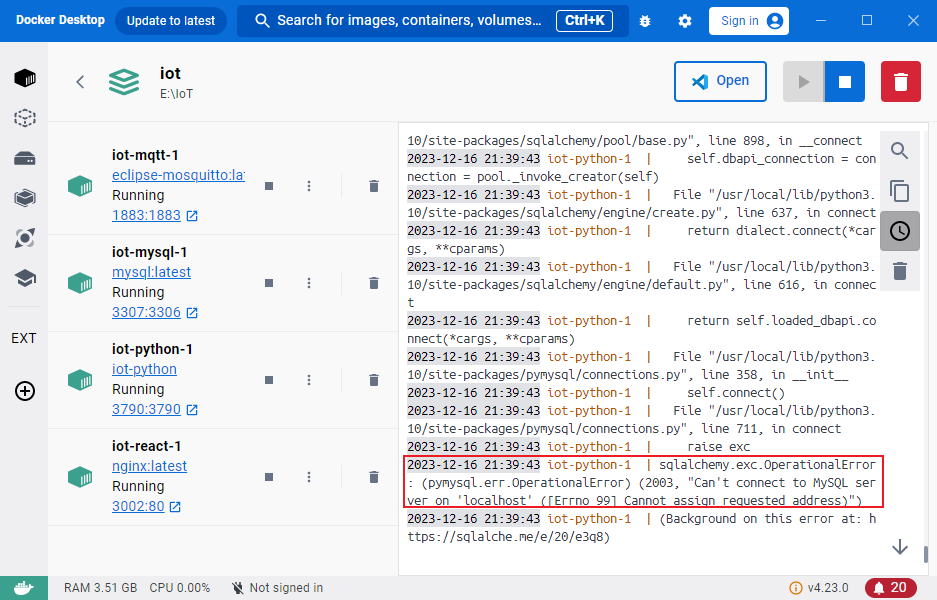
最后四个容器都能够运行了



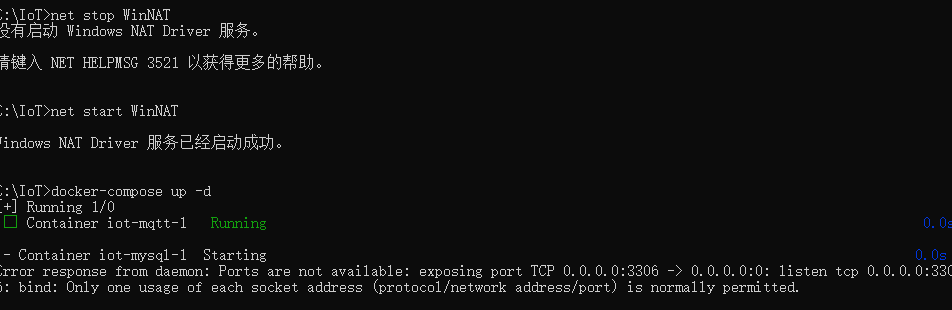
这个时候操作前端还是无法成功



观察docker可以发现问题出在sql上



由于端口3306没有进程



因此把端口改成了3307映射

另外还需要对后端和数据库的名称一致性进行更改

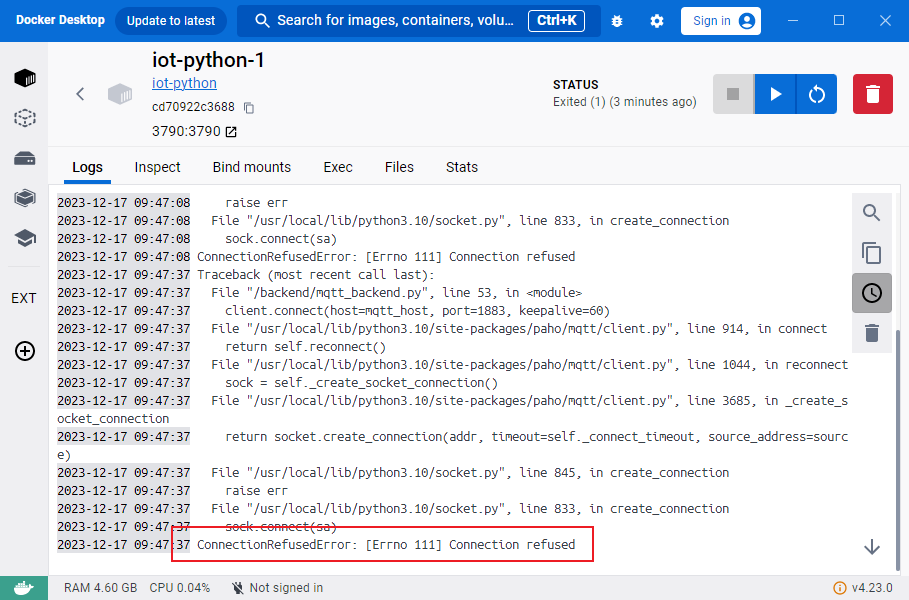


但是要改容器名称也过于繁琐，可以在docker-compose中增加network来解决这一问题

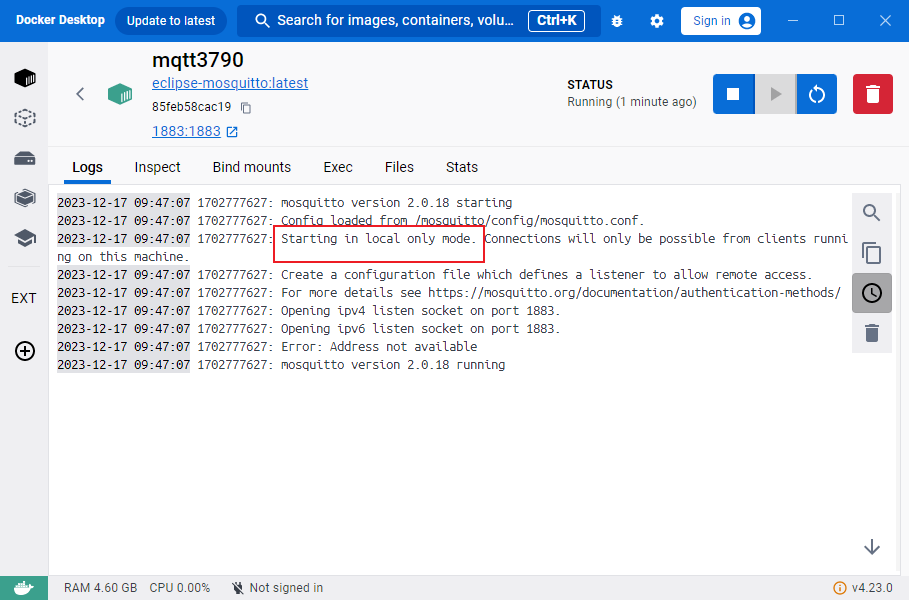


当进一步挂在mqtt服务后发现

当python程序要连接mqtt的时候，会显示访问被拒绝



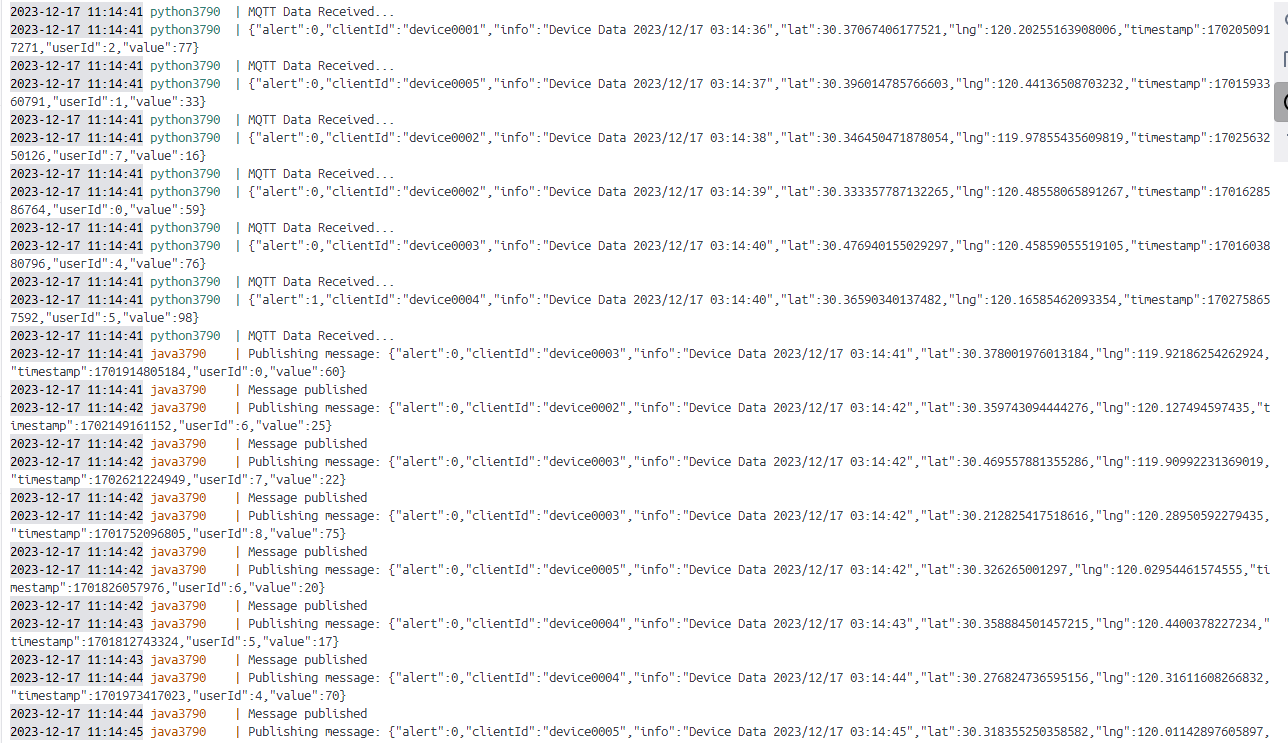
是因为mosquitto权限的问题（Starting in local only mode.）



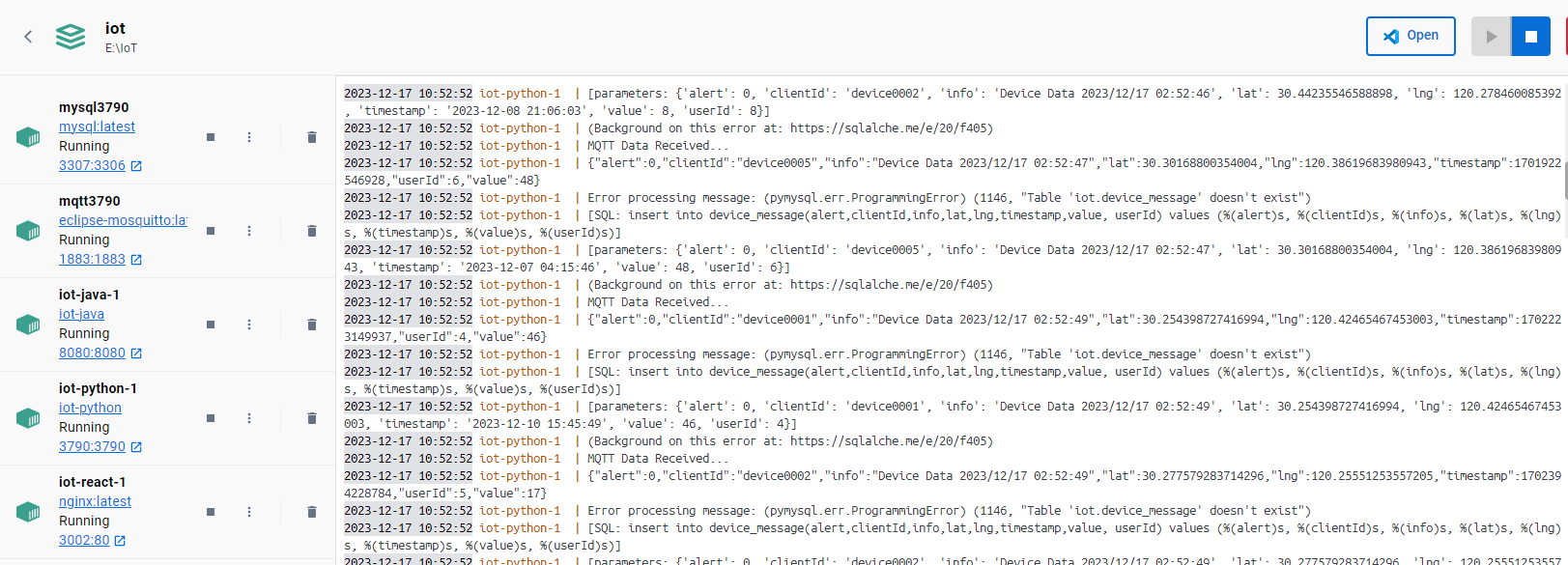
需要在mqtt的配置文件中增加



最后可以发现后端java发送程序和python以及mqtt能够正常运行（如下图消息的收发一一对应）

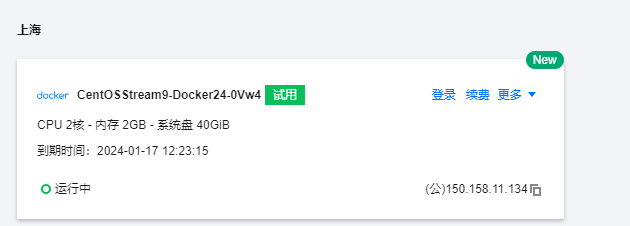


最终5个容器都正常运行，并且经过测试所有功能实现正常

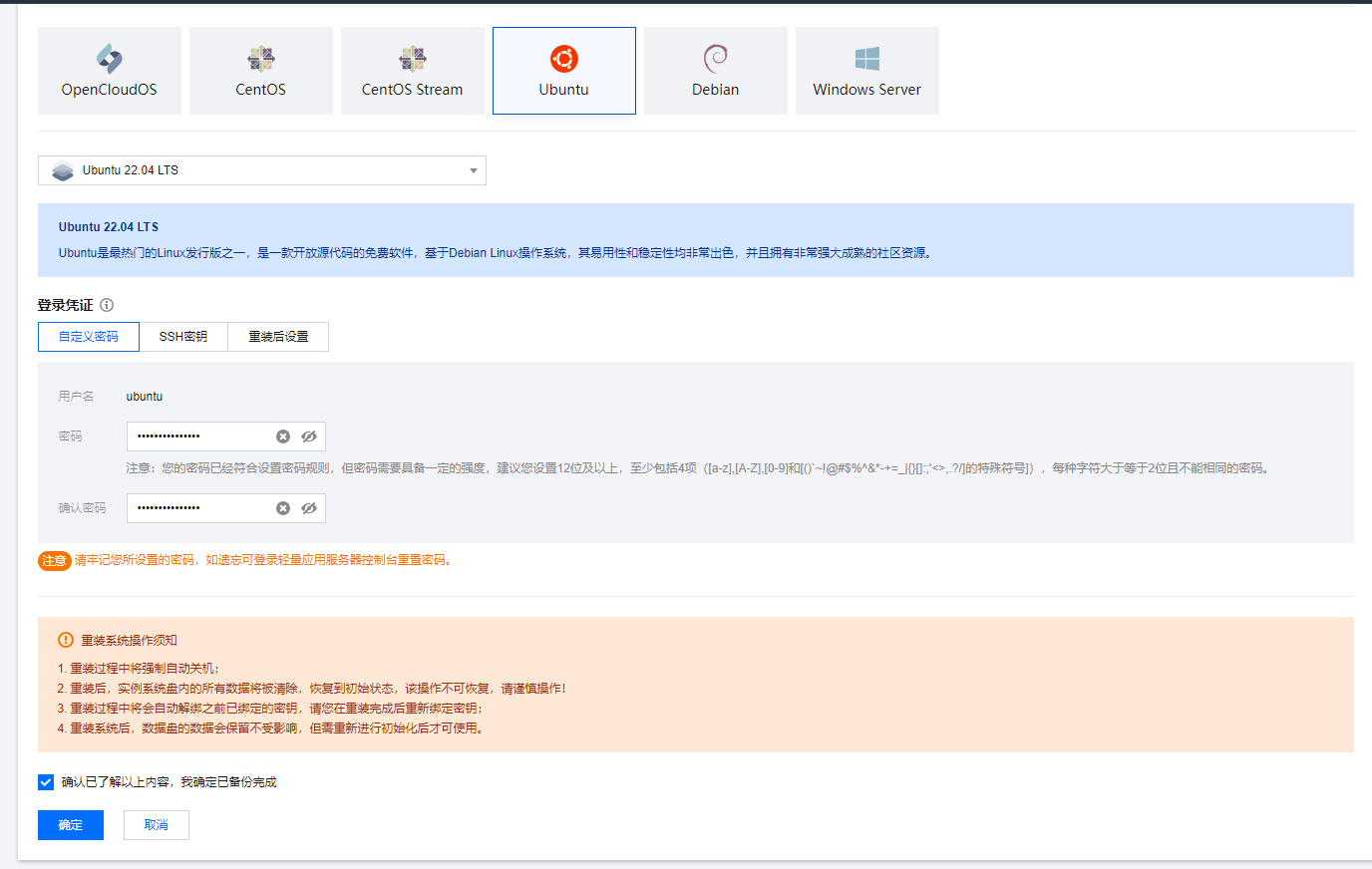


## 服务器部署

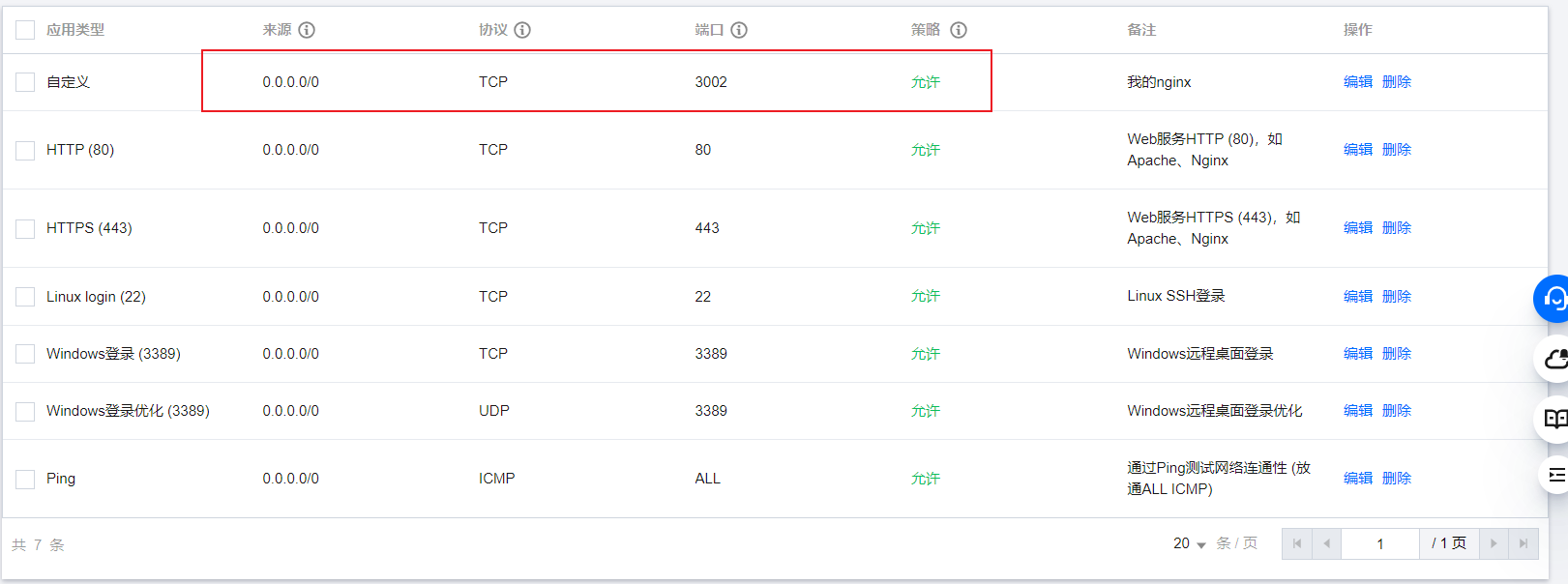
使用了腾讯云服务器，实例信息如下，按照和上面一样的建立docker容器的方法，在服务器上部署docker[[1]](#footnote-1)



并且重装系统为Ubuntu

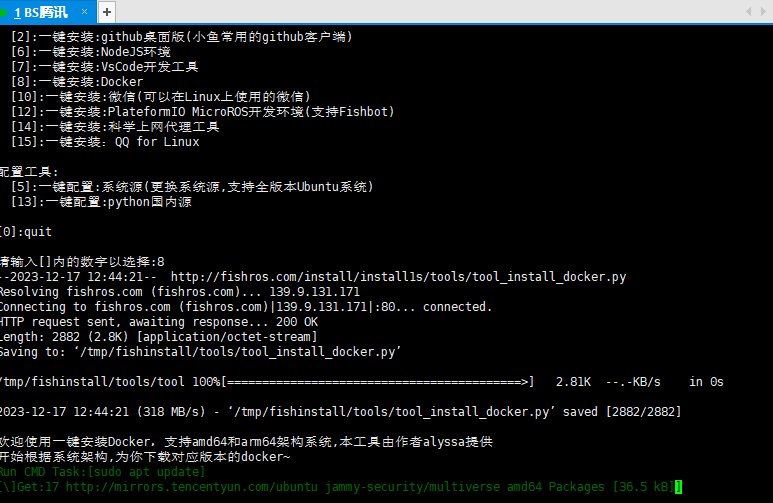


因为我们最后通过3002号端口来访问网页，而不是常规的80，因此需要在防火墙中添加3002端口



用下列命令先[一键安装docker](https://fishros.com/install/install1s/docs/index.html#/)

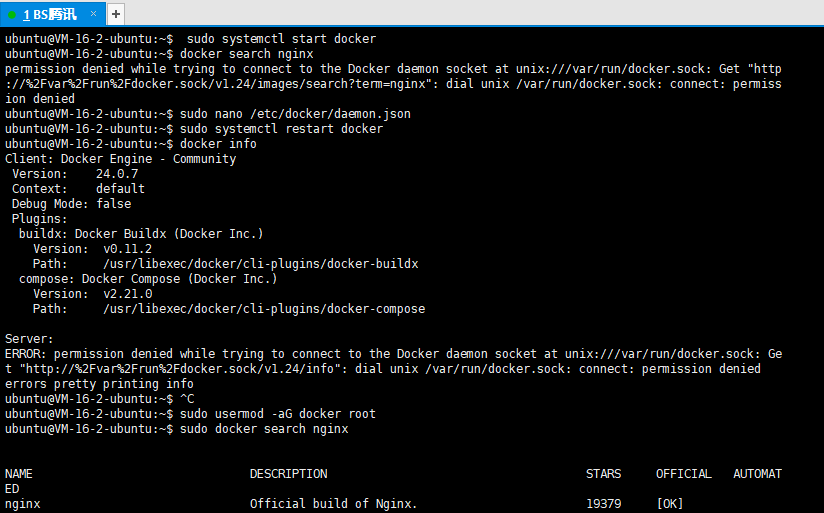
wget http://fishros.com/install -O fishros && . fishros



安装完docker之后进行换源

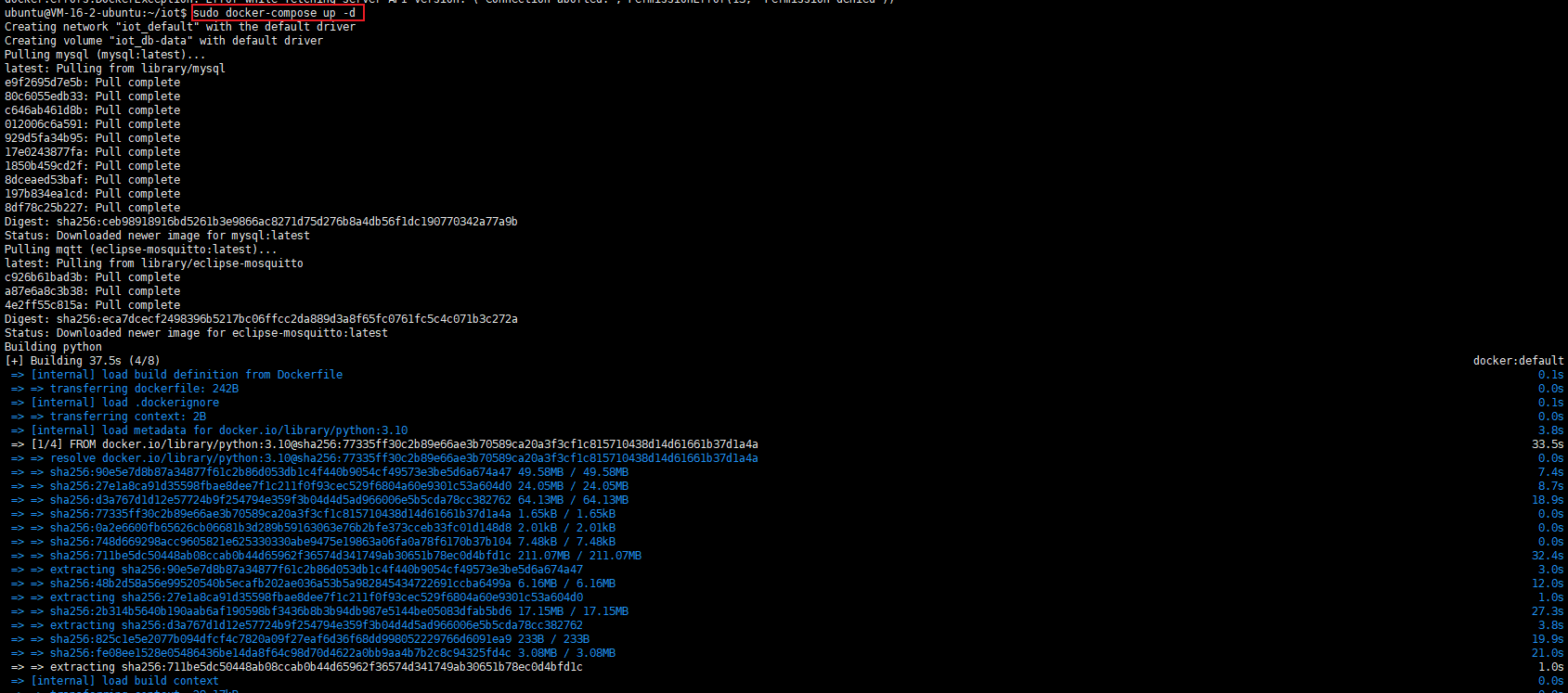


换源之后建立用户，然后可以发现docker已经能够正常搜索镜像源了

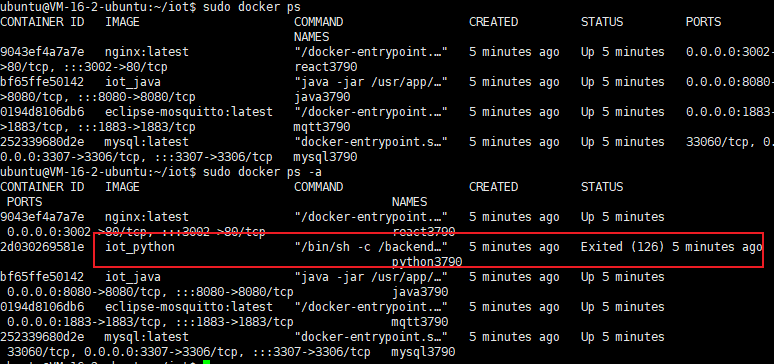


然后安装docker-compose并且生成容器

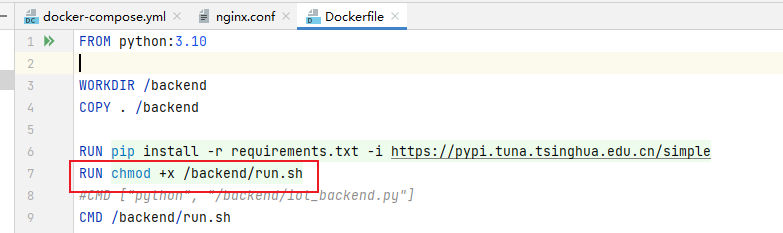
|  |
| --- |
| sudo apt install docker-compose  cd ~/iot  sudo docker-compose up -d |



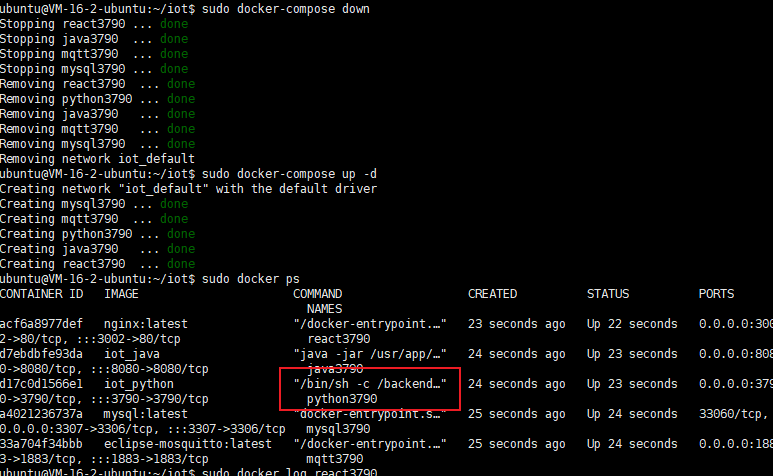
与本机环境不同的是python没有成功建立



需要在Dockerfile中增加授权

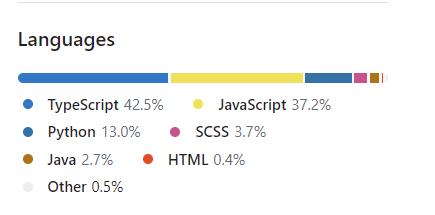


此时再重启docker就发现均已经运行正常

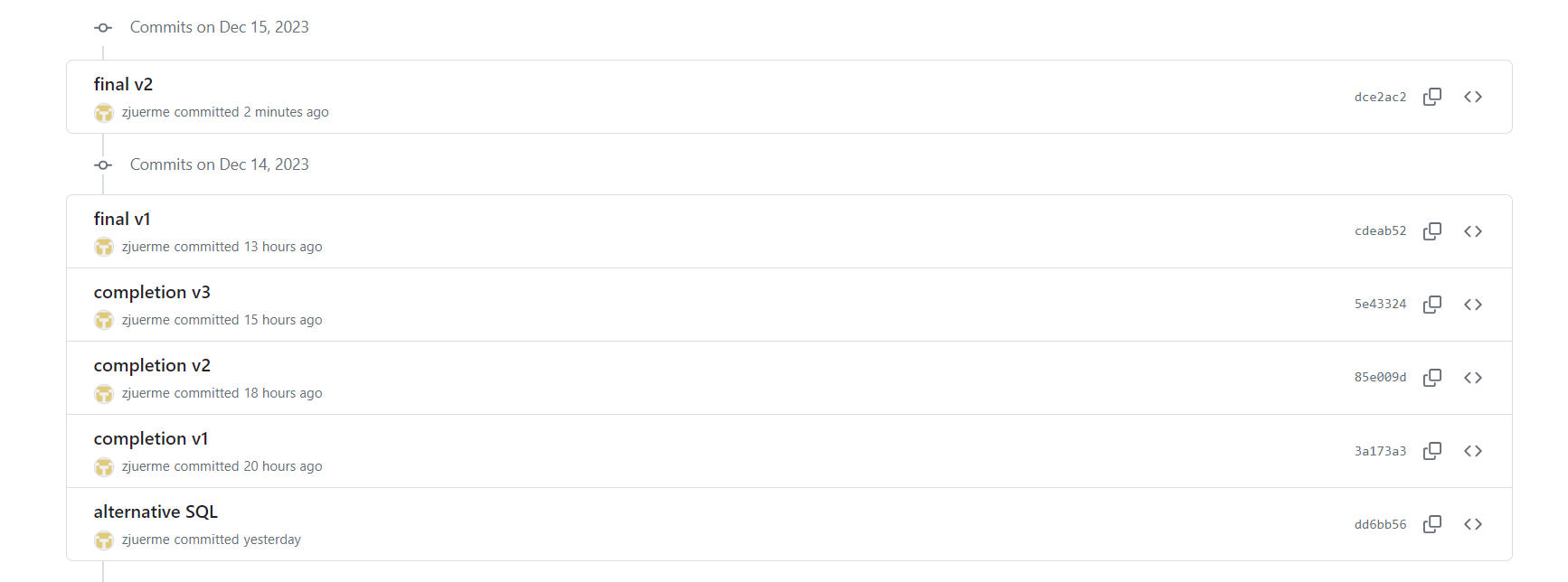


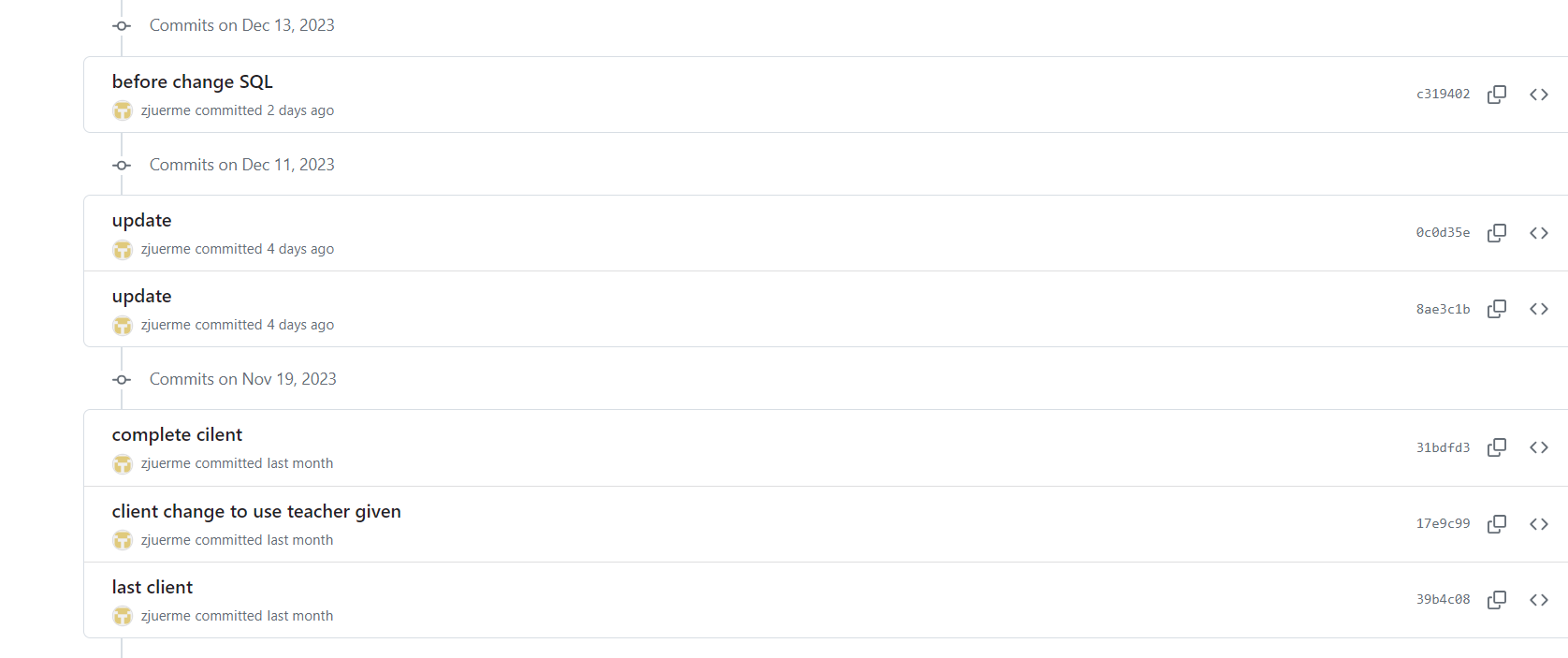
## Github日志记录

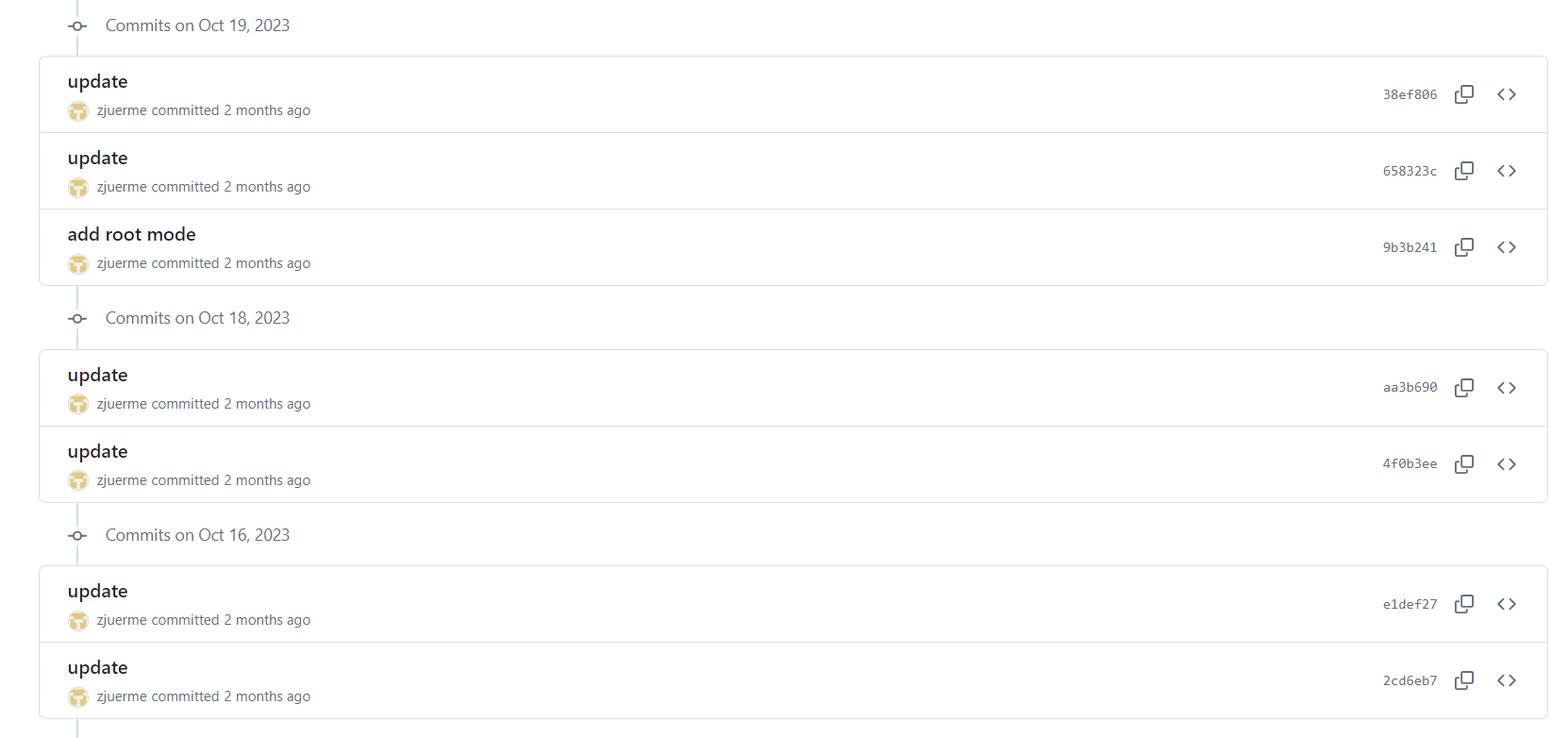
本项目的language使用情况如下

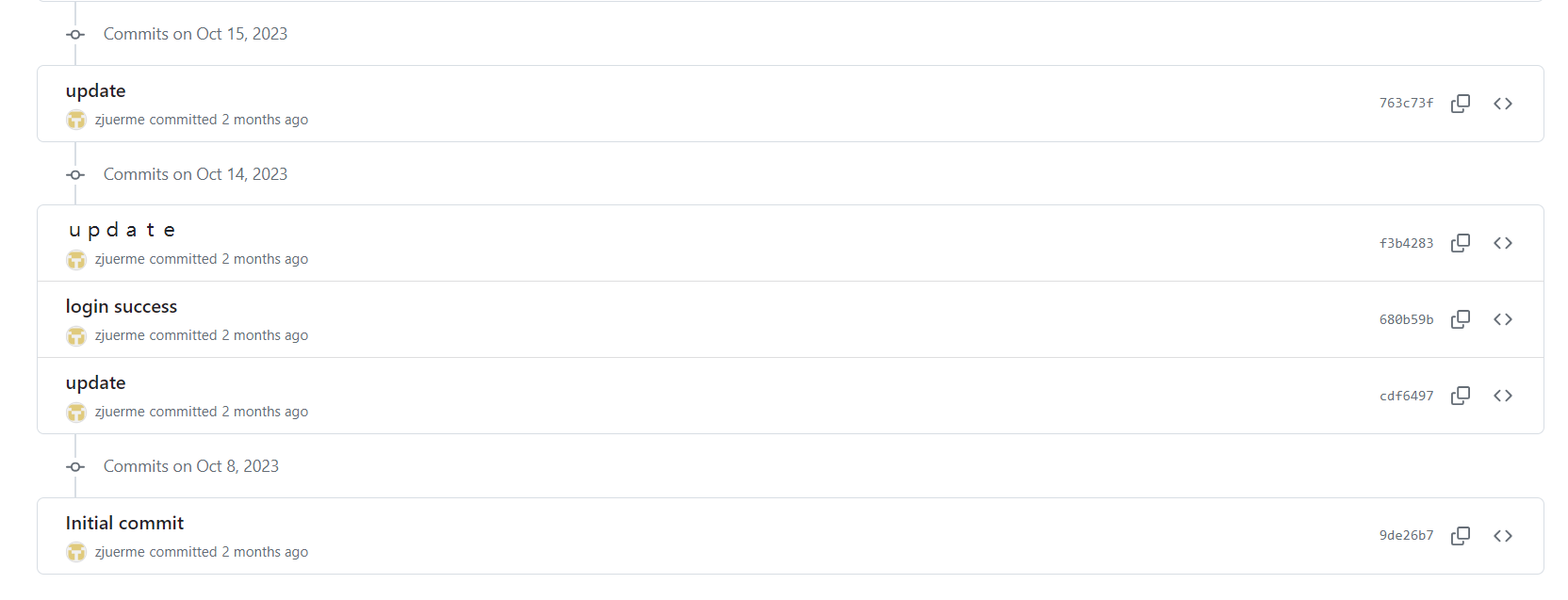


从10.8号开始本项目工程之后，一直用GitHub进行项目托管，记录了开发的相关情况，总共历时约2个半月最后完成了本项目

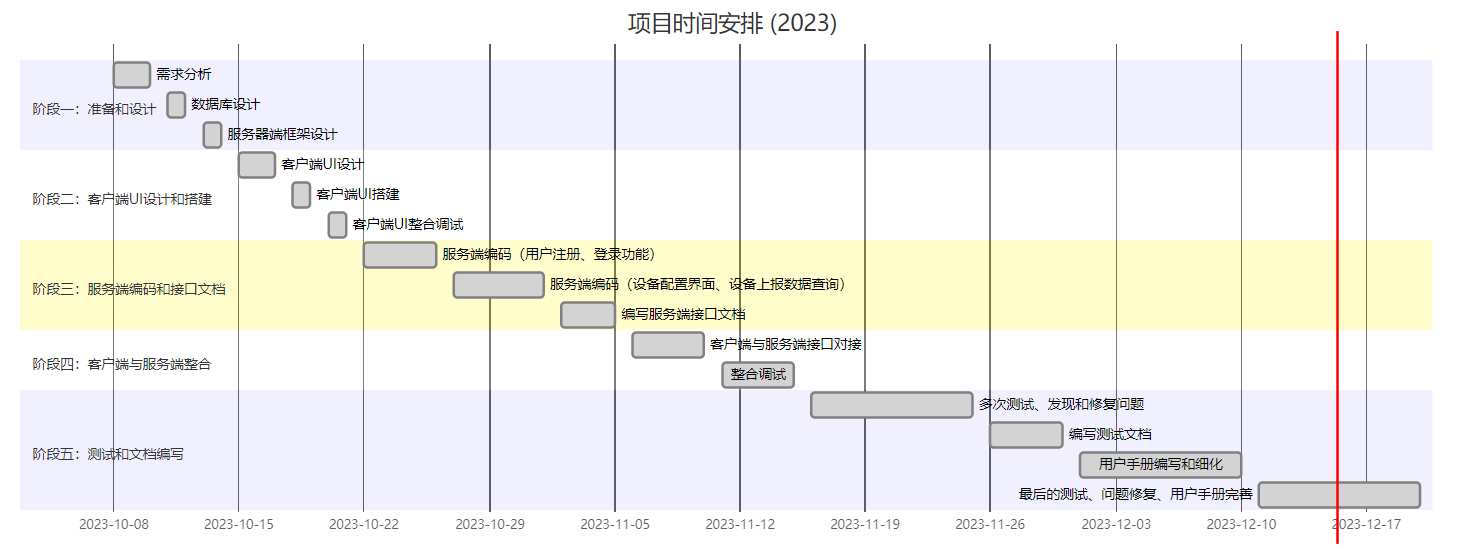








开发的时间安排也大致符合先前的安排



1. 注意：为了防止权限问题，shell操作docker的时候，最好都加上sudo [↑](#footnote-ref-1)