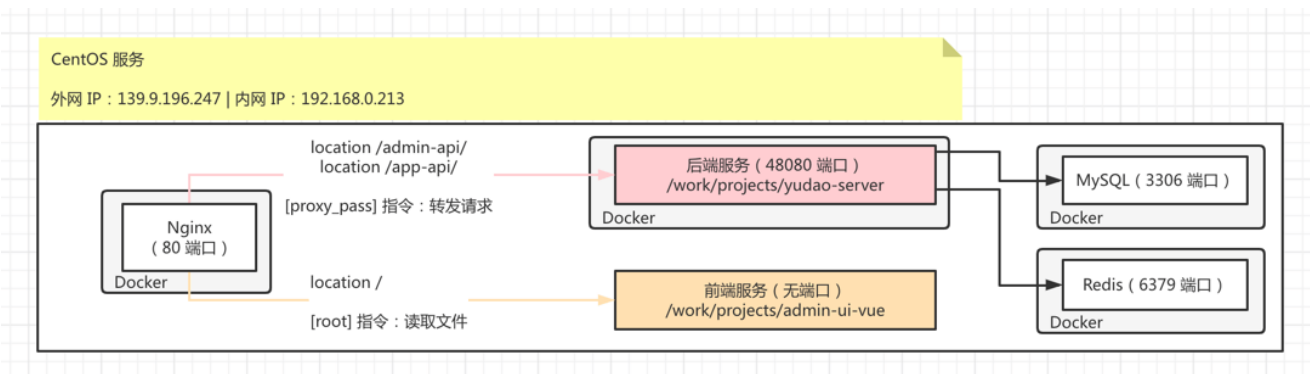


Docker 部署

友情提示：目前是 Boot 项目的部署，后续会调整成 Cloud 项目的部署

本小节，讲解如何将前端 + 后端项目，使用 Docker 容器，部署到 dev 开发环境下的一台 Linux 服务器上。如下图所示：



注意：服务器的 IP 地址。

- 外网 IP: 139.9.196.247
- 内网 IP: 192.168.0.213

下属所有涉及到 IP 的配置，需要替换成你自己的。

1. 安装 Docker

执行如下命令，进行 Docker 的安装。

```
## ① 使用 DaoCloud 的 Docker 高速安装脚本。参考 https://get.daocloud.io/#install-dc
curl -sSL https://get.daocloud.io/docker | sh

## ② 设置 DaoCloud 的 Docker 镜像中心，加速镜像的下载速度。参考 https://www.daocloud
curl -sSL https://get.daocloud.io/daotools/set_mirror.sh | sh -s http://f1361db2

## ③ 启动 Docker 服务
systemctl start docker
```

2. 配置 MySQL

2.1 安装 MySQL (可选)

友情提示：使用 Docker 安装 MySQL 是可选步骤，也可以直接安装 MySQL，或者购买 MySQL 云服务。

① 执行如下命令，使用 Docker 启动 MySQL 容器。

```
docker run -v /work/mysql:/var/lib/mysql \
-p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 \
--restart=always --name mysql -d mysql
```

- 数据库文件，挂载到服务器的 `/work/mysql/` 目录下
- 端口是 3306，密码是 123456

② 执行 `ls /work/mysql` 命令，查看 `/work/mysql/` 目录的数据库文件。

```
[root@ecs-156278 ~]# ls /work/mysql/
#ib_16384_0.dblwr  #innodb_temp  binlog.000001  binlog.index  ca.pem  client-key.pem  ib_logfile0  ibdata1  mysql  performance_schema
#ib_16384_1.dblwr  auto.cnf  binlog.000002  ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1  mysql.ibd  private_key.pem
```

2.2 导入 SQL 脚本

创建一个名字为 `ruoyi-vue-pro` 数据库，执行数据库对应的 `sql` 目录下的 SQL 文件，进行初始化。



3. 配置 Redis

友情提示：使用 Docker 安装 Redis 是可选步骤，也可以直接安装 Redis，或者购买 Redis 云服务。

执行如下命令，使用 Docker 启动 Redis 容器。

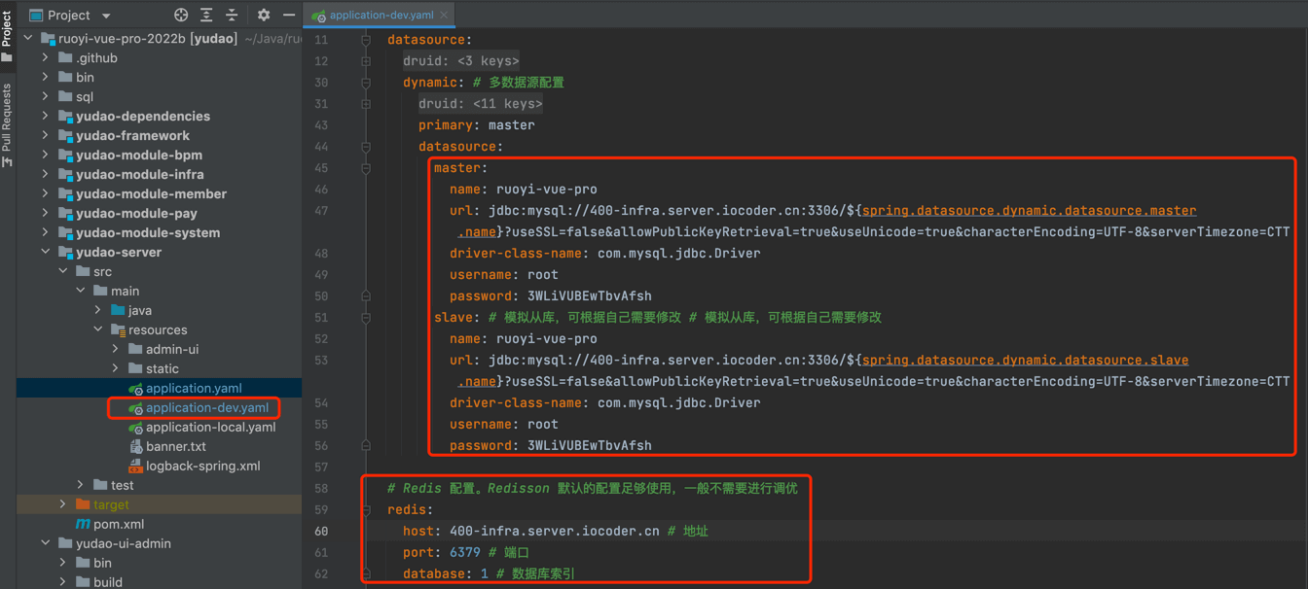
```
docker run -d --name redis --restart=always -p 6379:6379 redis:5.0.14-alpine
```

- 端口是 6379，密码未设置

4. 部署后端

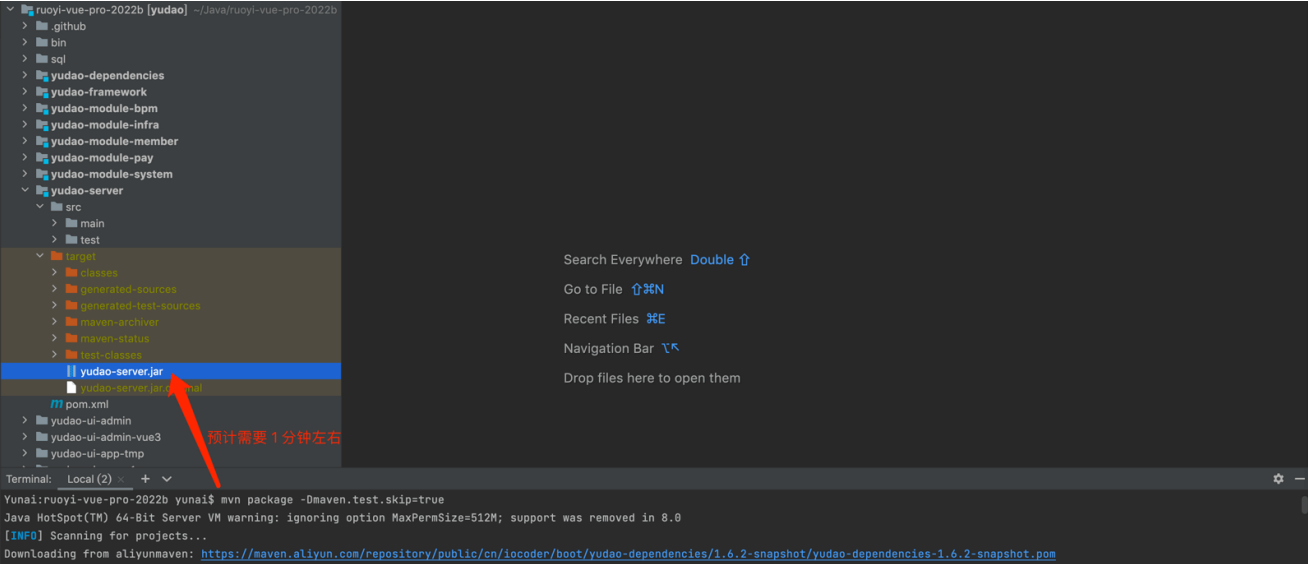
4.1 修改配置

后端 dev 开发环境对应的是 `application-dev.yaml` 配置文件，主要是修改 MySQL 和 Redis 为你的地址。如下图所示：



4.2 编译后端

在项目的根目录下，执行 `mvn clean package -Dmaven.test.skip=true` 命令，编译后端项目，构建出它的 Jar 包。如下图所示：

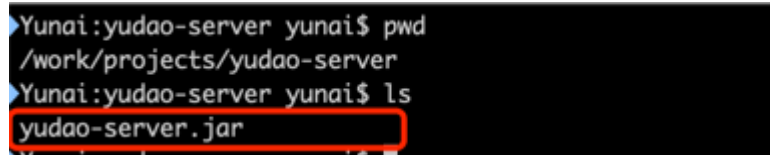


疑问：-Dmaven.test.skip=true 是什么意思？

跳过单元测试的执行。如果你项目的单元测试写的不错，建议使用 `mvn clean package` 命令，执行单元测试，保证交付的质量。

4.3 上传 Jar 包

在 Linux 服务器上创建 `/work/projects/yudao-server` 目录，使用 `scp` 命令或者 FTP 工具，将 `yudao-server.jar` 上传到该目录下。如下图所示：

A terminal window showing the following commands and output:

```
Yunai:yudao-server yunai$ pwd
/work/projects/yudao-server
Yunai:yudao-server yunai$ ls
yudao-server.jar
```

The file `yudao-server.jar` is highlighted with a red box.

4.4 构建镜像

① 在 `/work/projects/yudao-server` 目录下，新建 `Dockerfile` 文件，用于制作后端项目的 Docker 镜像。编写内容如下：

```
## AdoptOpenJDK 停止发布 OpenJDK 二进制，而 Eclipse Temurin 是它的延伸，提供更好的稳定性
## 感谢复旦核博士的建议！灰子哥，牛皮！
FROM eclipse-temurin:8-jre

## 创建目录，并使用它作为工作目录
RUN mkdir -p /yudao-server
WORKDIR /yudao-server

## 将后端项目的 Jar 文件，复制到镜像中
COPY yudao-server.jar app.jar

## 设置 TZ 时区
## 设置 JAVA_OPTS 环境变量，可通过 docker run -e "JAVA_OPTS=" 进行覆盖
ENV TZ=Asia/Shanghai JAVA_OPTS="-Xms512m -Xmx512m"

## 暴露后端项目的 48080 端口
EXPOSE 48080

## 启动后端项目
ENTRYPOINT java ${JAVA_OPTS} -Djava.security.egd=file:/dev/./urandom -jar app.jar
```

② 执行如下命令，构建名字为 `yudao-server` 的 Docker 镜像。

```
cd /work/projects/yudao-server
docker build -t yudao-server .
```

```
[root@ecs-156278 yudao-server]# docker build -t yudao-server .
Sending build context to Docker daemon 189.4MB
Step 1/7 : FROM eclipse-temurin:8-jre
8-jre: Pulling from library/eclipse-temurin
e0b25ef51634: Pull complete
d1bd2bc15eb1: Pull complete
d8fe65b7bd97: Pull complete
aff312975bbf: Pull complete
Digest: sha256:3800c6fb7431696aa8f105416e65270134dfb1a8e6ba8ebf3444033b7f7bfc72
Status: Downloaded newer image for eclipse-temurin:8-jre
--> 2a126d4fbb0e
Step 2/7 : RUN mkdir -p /yudao-server
--> Running in c1db670c3113
Removing intermediate container c1db670c3113
--> e37fadb8a606
Step 3/7 : WORKDIR /yudao-server
--> Running in 05700bfa8bb7
Removing intermediate container 05700bfa8bb7
--> 5438a53fc6c8
Step 4/7 : COPY yudao-server.jar app.jar
--> 8ae8fc7dc4e5
Step 5/7 : ENV TZ=Asia/Shanghai JAVA_OPTS="-Xms512m -Xmx512m"
--> Running in 18a550a7061b
Removing intermediate container 18a550a7061b
--> 0c4cf29353ef
Step 6/7 : EXPOSE 48080
--> Running in 6f39898ea3a3
Removing intermediate container 6f39898ea3a3
--> fb3a8e581e17
Step 7/7 : ENTRYPOINT java ${JAVA_OPTS} -Djava.security.egd=file:/dev/./urandom -jar app.jar
--> Running in ed7e7d859986
Removing intermediate container ed7e7d859986
--> 6dabefb7d4a9
Successfully built 6dabefb7d4a9
Successfully tagged yudao-server:latest
```

③ 在 `/work/projects/yudao-server` 目录下, 新建 Shell 脚本 `deploy.sh` , 使用 Docker 启动后端项目。编写内容如下:

```
#!/bin/bash
set -e

## 第一步: 删除可能启动的老 yudao-server 容器
echo "开始删除 yudao-server 容器"
docker stop yudao-server || true
docker rm yudao-server || true
echo "完成删除 yudao-server 容器"

## 第二步: 启动新的 yudao-server 容器 \
echo "开始启动 yudao-server 容器"
docker run -d \
--name yudao-server \
-p 48080:48080 \
-e "SPRING_PROFILES_ACTIVE=dev" \
-v /work/projects/yudao-server:/root/logs/ \
yudao-server
echo "正在启动 yudao-server 容器中, 需要等待 60 秒左右"
```

- 应用日志文件, 挂载到服务器的 `/work/projects/yudao-server` 目录下
- 通过 `SPRING_PROFILES_ACTIVE` 设置为 `dev` 开发环境

4.5 启动后端

① 执行 `sh deploy.sh` 命令，使用 Docker 启动后端项目。日志如下：

```
开始删除 yudao-server 容器
yudao-server
yudao-server
完成删除 yudao-server 容器
开始启动 yudao-server 容器
0dfd3dc409a53ae6b5e7c5662602cf5dcb52fd4d7f673bd74af7d21da8ead9d5
正在启动 yudao-server 容器中，需要等待 60 秒左右
```

② 执行 `docker logs yudao-server` 命令，查看启动日志。看到如下内容，说明启动完成：

友情提示：如果日志比较多，可以使用 `grep` 进行过滤。

例如说：使用 `docker logs yudao-server | grep 48080`

```
2022-04-15 00:34:19.647 INFO 8 --- [main] [TID: N/A] o.s.b.w.embedded.tomcat.Tc
```

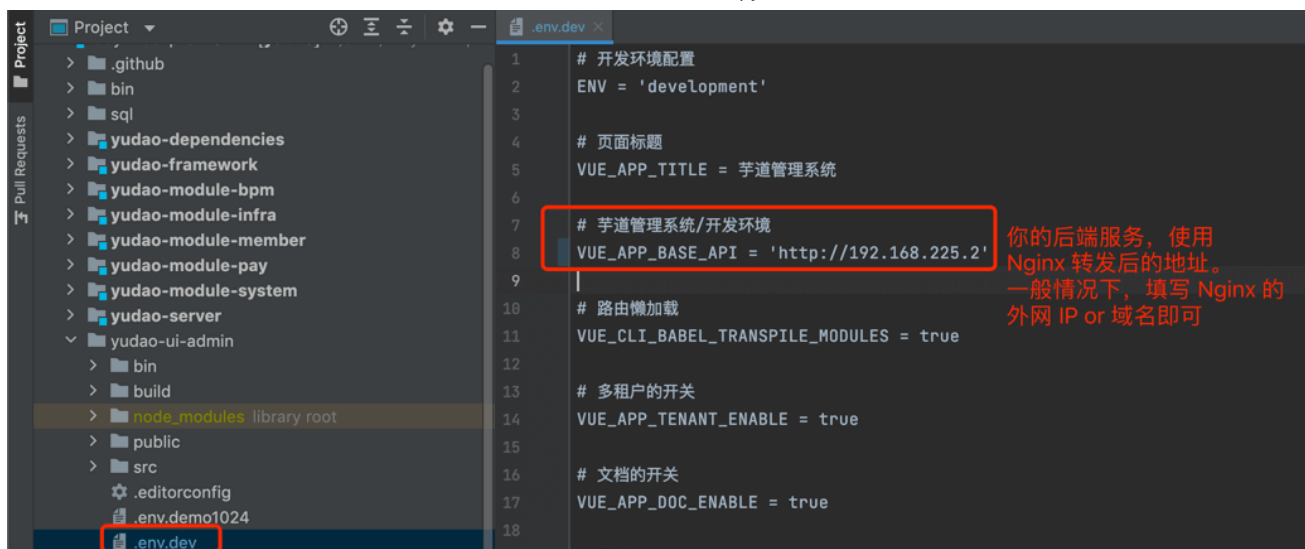
5. 部署前端

友情提示：

本小节的内容，和《开发指南 —— Linux 部署》的「部署前端」是基本一致的。

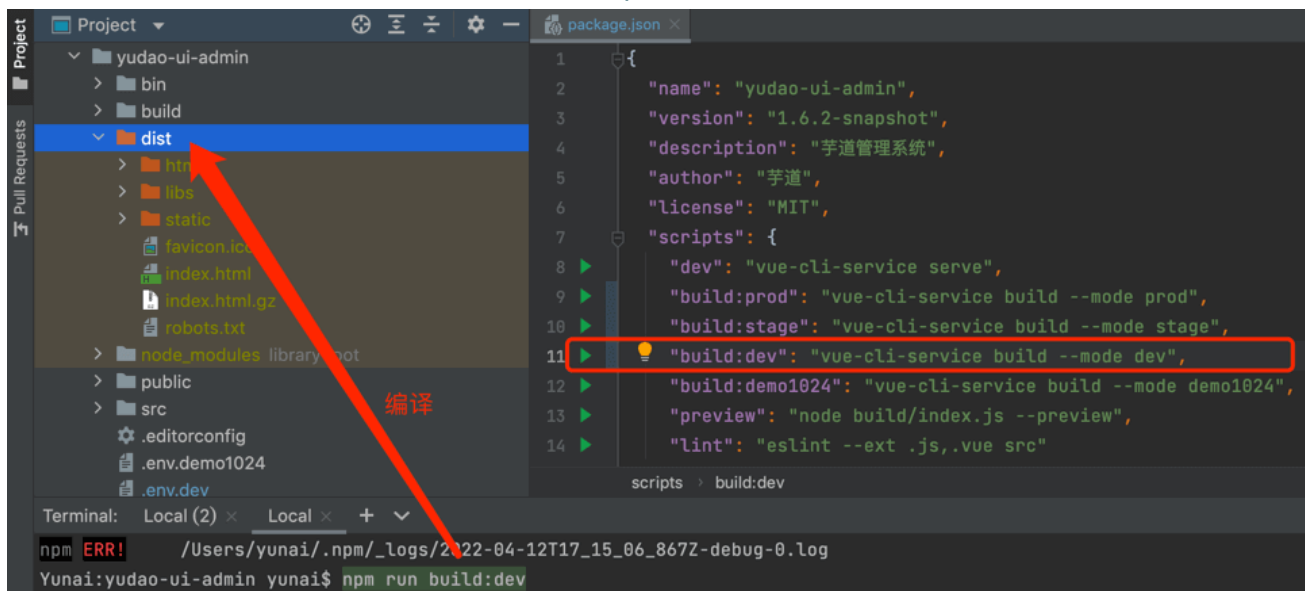
5.1 修改配置

前端 dev 开发环境对应的是 `.env.dev` 配置文件，主要是修改 `VUE_APP_BASE_API` 为你的后端项目的访问地址。如下图所示：



5.2 编译前端

在 `yudao-ui-admin` 目录下，执行 `npm run build:dev` 命令，编译前端项目，构建出它的 `dist` 文件，里面是 HTML、CSS、JavaScript 等静态文件。如下图所示：



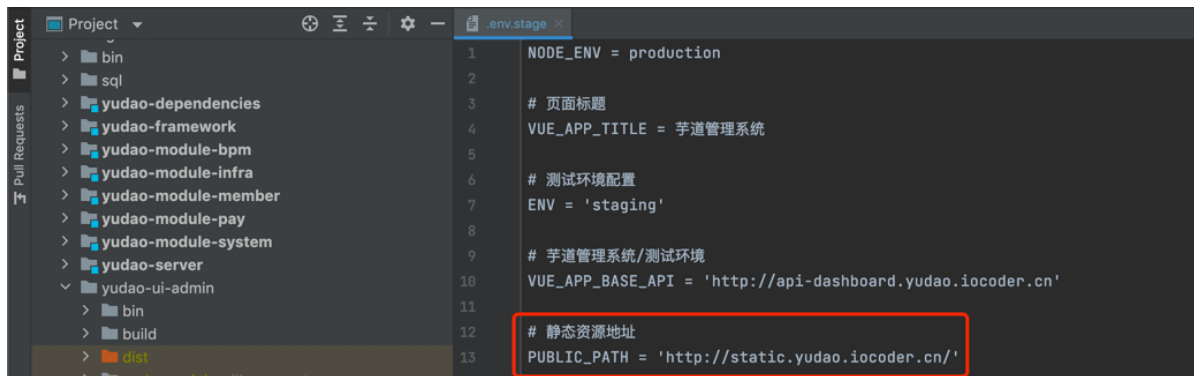
如下想要打包其它环境，可使用如下命令：

`npm run build:prod` ## 打包 prod 生产环境

`npm run build:stage` ## 打包 stage 预发布环境

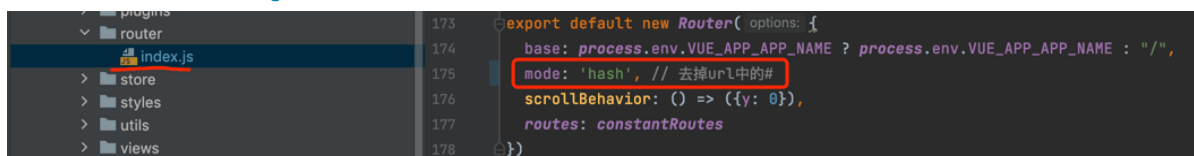
其它高级参数说明【可暂时不看】：

① `PUBLIC_PATH`：静态资源地址，可用于七牛等 CDN 服务回源读取前端的静态文件，提升访问速度，建议 prod 生产环境使用。示例如下：



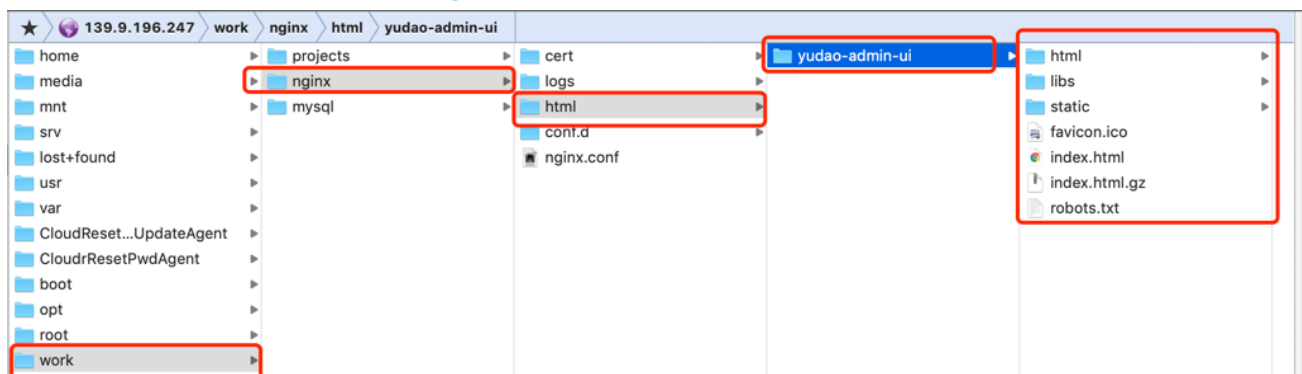
② `VUE_APP_APP_NAME` : 二级部署路径, 默认为 `/` 根目录, 一般不用修改。

③ `mode` : 前端路由的模式, 默认采用 `history` 路由, 一般不用修改。可以通过修改 `router/index.js` 来设置为 `hash` 路由, 示例如下:



5.3 上传 dist 文件

在 Linux 服务器上创建 `/work/projects/yudao-ui-admin` 目录, 使用 `scp` 命令或者 FTP 工具, 将 `dist` 上传到 `/work/nginx/html` 目录下。如下图所示:



5.4 启动前端?

前端无法直接启动, 而是通过 Nginx 转发读取 `/work/projects/yudao-ui-admin` 目录的静态文件。

6. 配置 Nginx

6.1 安装 Nginx

Nginx 挂载到服务器的目录:

- `/work/nginx/conf.d` 用于存放配置文件
- `/work/nginx/html` 用于存放网页文件
- `/work/nginx/logs` 用于存放日志

- `/work/nginx/cert` 用于存放 HTTPS 证书

① 创建 `/work/nginx` 目录, 并在该目录下新建 `nginx.conf` 文件, 避免稍后安装 Nginx 报错。内容如下:

```
user  nginx;
worker_processes  1;

events {
    worker_connections  1024;
}

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

http {
    include      /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile      on;
    keepalive_timeout  65;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    #    access_log  /var/log/nginx/access.log  main;

    gzip on;
    gzip_min_length 1k;      # 设置允许压缩的页面最小字节数
    gzip_buffers 4 16k;      # 用来存储 gzip 的压缩结果
    gzip_http_version 1.1;   # 识别 HTTP 协议版本
    gzip_comp_level 2;        # 设置 gzip 的压缩比 1-9。1 压缩比最小但最快, 而 9 相反
    gzip_types text/plain application/x-javascript text/css application/xml appl
    gzip_proxied any;        # 无论后端服务器的 headers 头返回什么信息, 都无条件启用压

    include /etc/nginx/conf.d/*.conf; ## 加载该目录下的其它 Nginx 配置文件
}
```

② 执行如下命令, 使用 Docker 启动 Nginx 容器。

```
docker run -d \
--name nginx --restart always \
-p 80:80 -p 443:443 \
-e "TZ=Asia/Shanghai" \
```

```
-v /work/nginx/nginx.conf:/etc/nginx/nginx.conf \
-v /work/nginx/conf.d:/etc/nginx/conf.d \
-v /work/nginx/logs:/var/log/nginx \
-v /work/nginx/cert:/etc/nginx/cert \
-v /work/nginx/html:/usr/share/nginx/html \
nginx:alpine
```

③ 执行 `docker ps` 命令，查看到 Nginx 容器的状态是 `UP` 的。

下面，来看两种 Nginx 的配置，分别满足服务器 IP、独立域名的不同场景。

6.2 方式一：服务器 IP 访问

① 在 `/work/nginx/conf.d` 目录下，创建 `ruoyi-vue-pro.conf`，内容如下：

```
server {
    listen      80;
    server_name 139.9.196.247; ## 重要!!! 修改成你的外网 IP/域名

    location / { ## 前端项目
        root    /usr/share/nginx/html/yudao-admin-ui;
        index   index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /admin-api/ { ## 后端项目 - 管理后台
        proxy_pass http://192.168.0.213:48080/admin-api/; ## 重要!!! proxy_pass
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /app-api/ { ## 后端项目 - 用户 App
        proxy_pass http://192.168.0.213:48080/app-api/; ## 重要!!! proxy_pass
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

友情提示：

[root] 指令在本地文件时，要使用 Nginx Docker 容器内的路径，即 `/usr/share/nginx/html/yudao-admin-ui`，否则会报 404 的错误。

② 执行 `docker exec nginx nginx -s reload` 命令，重新加载 Nginx 配置。

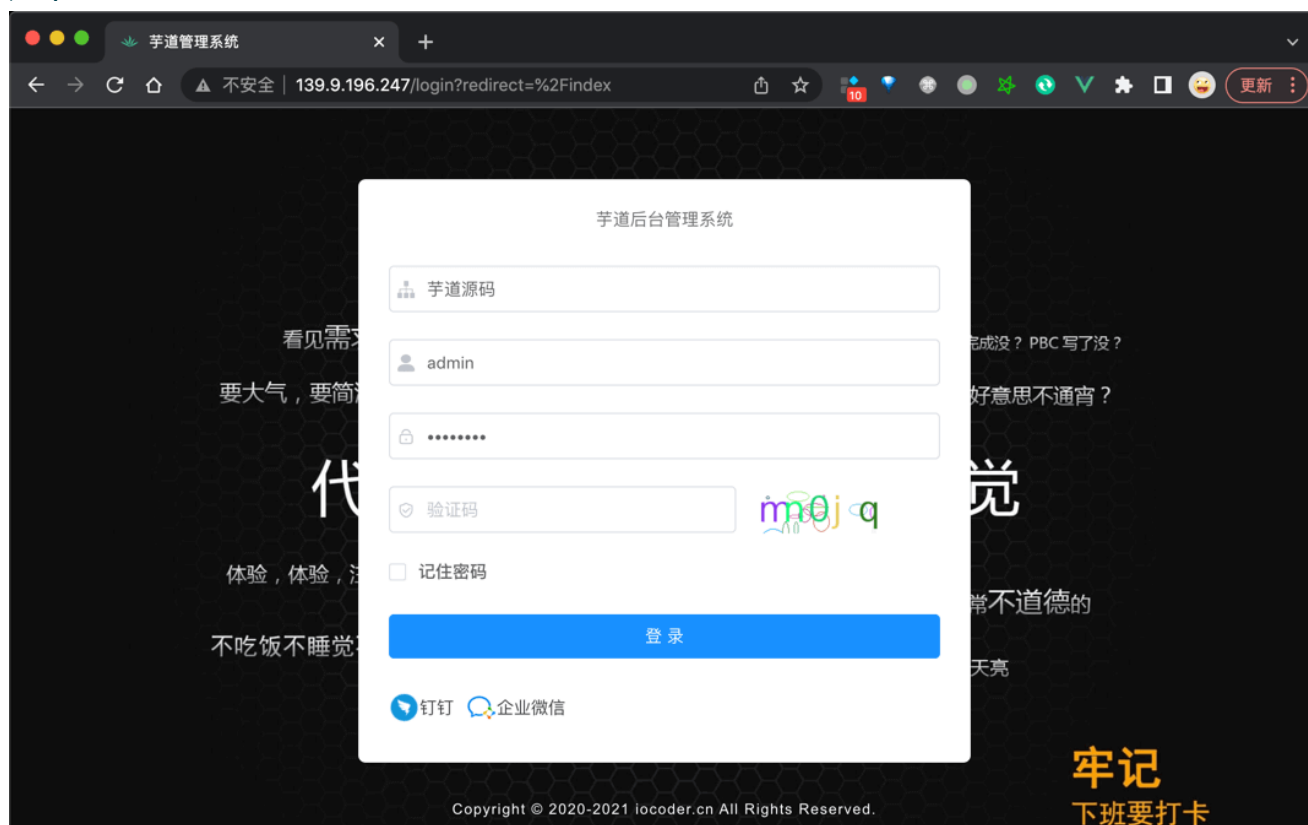
友情提示：如果你担心 Nginx 配置不正确，可以执行 `docker exec nginx nginx -t` 命令。

③ 执行 `curl http://192.168.0.213/admin-api/` 命令，成功访问后端项目的内网地址，返回结果如下：

```
{"code":401,"data":null,"msg":"账号未登录"}
```

执行 `curl http://139.9.196.247:48080/admin-api/` 命令，成功访问后端项目的外网地址，返回结果一致。

④ 请求 <http://139.9.196.247:48080> 地址，成功访问前端项目的外网地址，返回前端界面如下：



6.3 方式二：独立域名访问

友情提示：在前端项目的编译时，需要把 `VUE_APP_BASE_API` 修改为后端项目对应的域名。

例如说, 这里使用的是 <http://api.iocoder.cn>

① 在 `/work/nginx/conf.d` 目录下, 创建 `ruoyi-vue-pro2.conf`, 内容如下:

```
server { ## 前端项目
    listen      80;
    server_name admin.iocoder.cn; ## 重要!!! 修改成你的前端域名

    location / { ## 前端项目
        root    /usr/share/nginx/html/yudao-admin-ui;
        index   index.html index.htm;
        try_files $uri $uri/ /index.html;
    }
}

server { ## 后端项目
    listen      80;
    server_name api.iocoder.cn; ## 重要!!! 修改成你的外网 IP/域名

    ## 不要使用 location / 转发到后端项目, 因为 druid、admin 等监控, 不需要外网可访问。

    location /admin-api/ { ## 后端项目 - 管理后台
        proxy_pass http://192.168.0.213:48080/admin-api/; ## 重要!!! proxy_pass
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

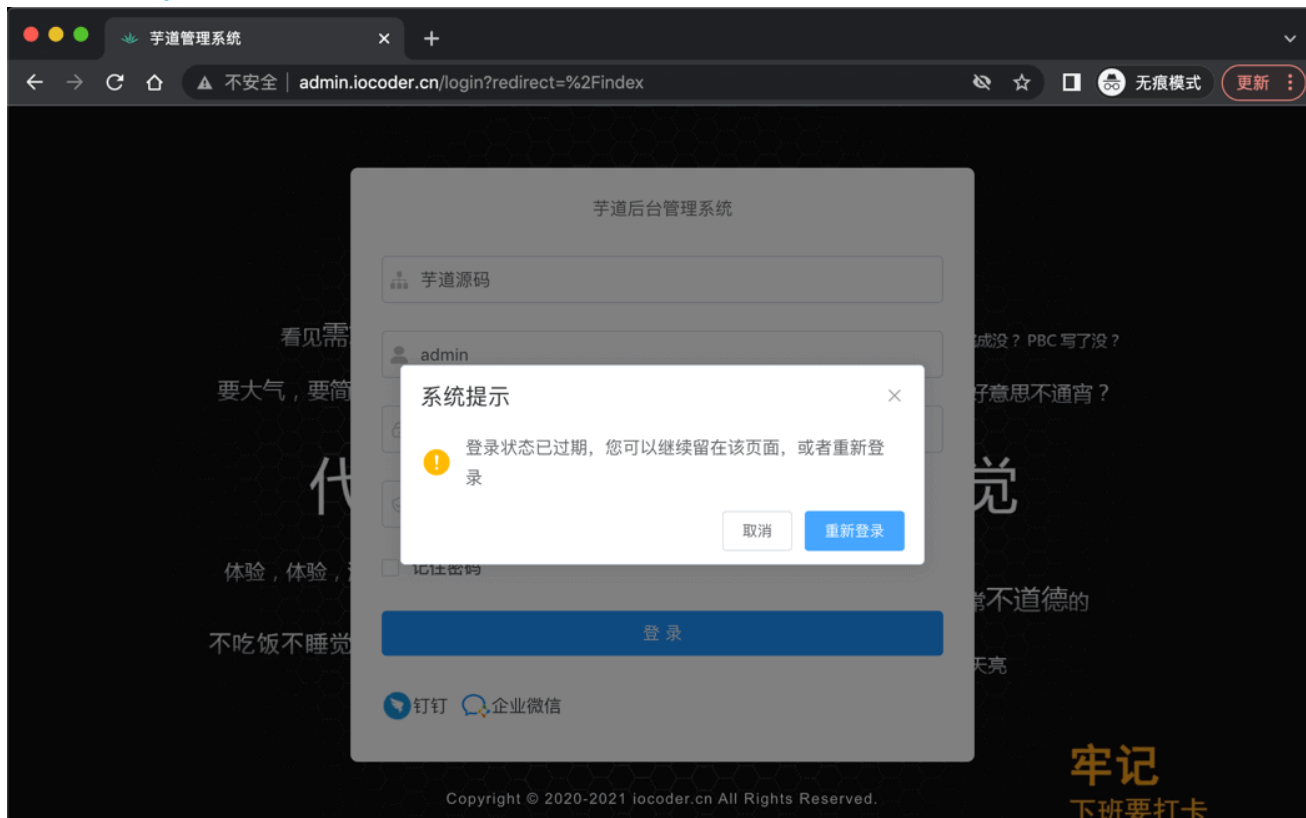
    location /app-api/ { ## 后端项目 - 用户 App
        proxy_pass http://192.168.0.213:48080/app-api/; ## 重要!!! proxy_pass
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

② 执行 `docker exec nginx nginx -s reload` 命令, 重新加载 Nginx 配置。

③ 请求 <http://api.iocoder.cn/admin-api/> 地址, 成功访问后端项目, 返回结果如下:

```
{"code":401,"data":null,"msg":"账号未登录"}
```

④ 请求 <http://admin.iocoder.cn> 地址，成功访问前端项目，返回前端界面如下：



← [Linux 部署](#)

[Jenkins 部署](#) →



Theme by [Vdoing](#) | Copyright © 2019-2023 芋道源码 | MIT License