nginx实践

只记录实践知识点，具体详细内容参考

nginx学习，看这一篇就够了：下载、安装。使用：正向代理、反向代理、负载均衡。常用命令和配置文件,很全-CSDN博客

其他中文文档

Nginx安装 | Nginx 中文官方文档

ubuntu安装nginx参考该文章

Ubuntu系统下Nginx安装_ubuntu安装nginx-CSDN博客

1. 安装nginx

1.1 apt 自动安装 nginx

经过多次尝试后，建议使用`apt`包管理器来安装nginx，**即自动安装，自动配置环境变量**，只需知道常用的文件路径位置即可

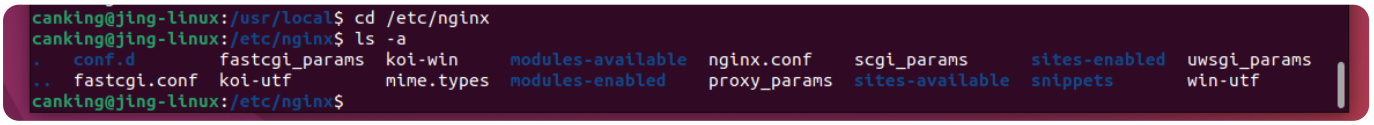
Ubuntu系统下Nginx安装_ubuntu安装nginx-CSDN博客

手动解压压缩包安装首先需要卸载`apt`安装的nginx，否则环境会冲突。然后要进入到nginx的bin目录下运行程序，或者配置环境变量，`nginx`命令才能使用

自动安装后nginx对应文件位置：

```
/usr/sbin/nginx # 启动nginx程序的文件位置
/etc/nginx  # 存放配置文件
/var/www/html #  存放项目目录
/var/log/nginx # 存放日志
```

进入`/etc/nginx`中即可查看配置文件的结构



更多详情目录信息如下：

- `/usr/sbin/nginx`：**主程序，启动文件**
- `/etc/nginx`：**存放配置文件**
  - **主配置文件**`/etc/nginx/nginx.conf`
  - **自定义的配置文件**通常放在`/etc/nginx/conf.d`目录下
  - 网站的server blocks配置通常存储在`/etc/nginx/sites-available`目录中
  - 而`/etc/nginx/sites-enabled`目录包含链接到`sites-available`目录的配置文件
- `/var/www/html`：存放项目目录
  - **默认的网站文件**存放在`/var/www/html`目录下，这里包含了Nginx的默认欢迎页面
  - **可以通过修改Nginx配置文件来改变网站文件的位置**
- `/var/log/nginx`：**存放日志**
  - 包括`access.log`和`error.log`
- **临时文件目录**：
  - Nginx的临时文件目录可能包括`client_body_temp`、`fastcgi_temp`、`proxy_temp`、`scgi_temp`和`uwsgi_temp`等，这些目录通常位于Nginx的安装目录下

```
/usr/sbin/nginx
├── /etc/nginx
│    ├── nginx.conf
│    ├── conf.d/
```

```
│   │   └── [自定义配置文件]
│   ├── sites-available/
│   │   └── [网站配置文件]
│   └── sites-enabled/
│       └── [指向 sites-available 的链接]
├── /var/www/html
│   └── [默认网站文件]
├── /var/log/nginx
│   ├── access.log
│   └── error.log
└── [临时文件目录]
    ├── client_body_temp/
    ├── fastcgi_temp/
    ├── proxy_temp/
    ├── scgi_temp/
    └── uwsgi_temp/
```

1.2 手动安装nginx

我们在官网中下载好安装包后，这里下载的是`nginx-1.23.4.tar.gz`，传到ubuntu上

这里我们新建一个`~/web`文件夹来存放相关的文件

```
canking@jing-linux:~$ cd web/
canking@jing-linux:~/web$ ls -a
.  ..  nginx-1.23.4  nginx-1.23.4.tar.gz  nginx-cluster  nginx-project  tomcat_1  tomcat_2
canking@jing-linux:~/web$ ls -a nginx-1.23.4
.  ..  auto  CHANGES  CHANGES.ru  conf  configure  contrib  html  LICENSE  Makefile  man  nginx  objs  README
canking@jing-linux:~/web$
```

具体目录结构如下：

```
# 手动解压安装包下的目录结构
nginx-版本号/
├── CHANGES # Nginx的变更日志文件
├── CONTRIBUTING.md # 贡献指南
├── LICENSE # Nginx的许可证文件。
├── README.md # Nginx的自述文件。
├── configure # Nginx的配置脚本，用于生成config.h和auto/目录。
├── contrib/ # 包含一些额外的脚本和配置文件，例如geo2nginx用于生成GeoIP模块的配置
│   ├── geo2nginx
```

```
│   ├── nginx.conf
│   ├── nginx.vim
│   ├── pcre-config.cache
│   └── pcre-config.in
├── html/ # 包含Nginx默认的静态文件
│   ├── 50x.html
│   ├── index.html
│   └── nginx-logo.png
├── man/ # 包含Nginx的手册页。
│   ├── nginx.8
│   ├── nginx.conf.5
│   └── README
├── src/ # 包含Nginx的源代码，按模块和功能组织。
│   ├── core
│   ├── event
│   ├── http
│   ├── mail
│   ├── misc
│   ├── os
│   ├── stream
│   └── test
├── zlib/ # 包含Nginx使用的Zlib源代码
└── modules/ # 包含Nginx的模块源代码。
```

1.3 使用docker 拉取nginx镜像

参考文章使用docker搭建nginx集群,实现负载均衡 - 知乎

1.3.1拉取镜像

```
docker pull nginx:latest # 这里我们直接拉取最新镜像即可
```

1.3.2 准备一个工作目录

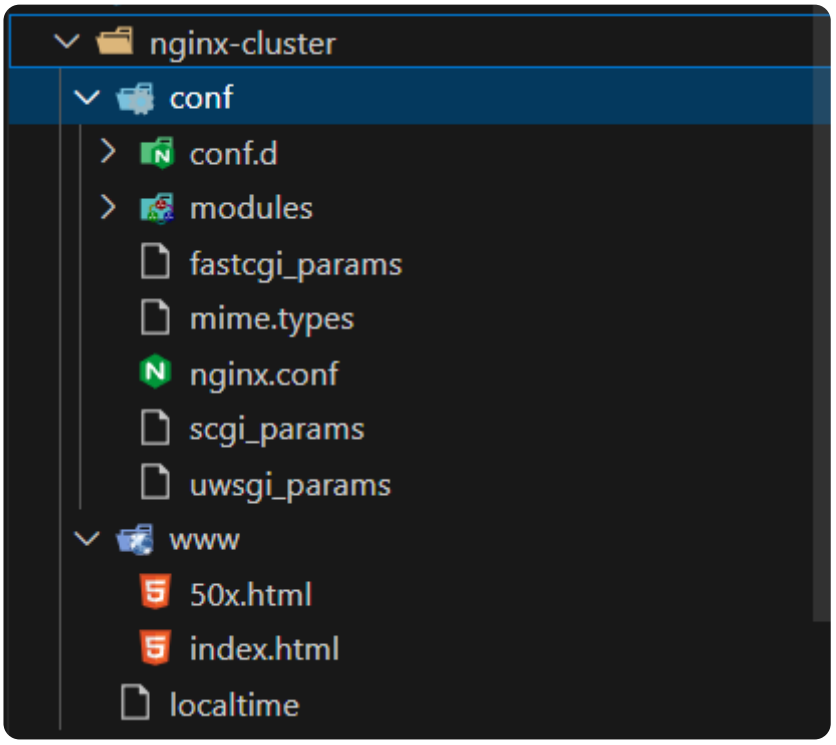
创建`~/web/nginx-cluster`目录，用于存放配置文件以及前端文件

1.3.3 先创建一个简单的容器

- 运行`docker run --name nginx-test -p 8080:80 -d nginx`
- 拷贝nginx配置文件和html文件到宿主机

```
# 执行拷贝命令
docker cp nginx-test:/etc/nginx ~/web/nginx-cluster/conf
docker cp nginx-test:/usr/share/nginx/html ~/web/nginx-cluster/www
```

- 执行完之后就可以看到目录下出现了一些配置文件，这些文件是从容器中复制出来的



- 之后可以将这个简单的容器`nginx-test`停止并且删除了

1.3.4 正式创建一个nginx

有了这些配置文件，我们在创建一个nginx容器时需要用`-v`关联文件，方便我们修改

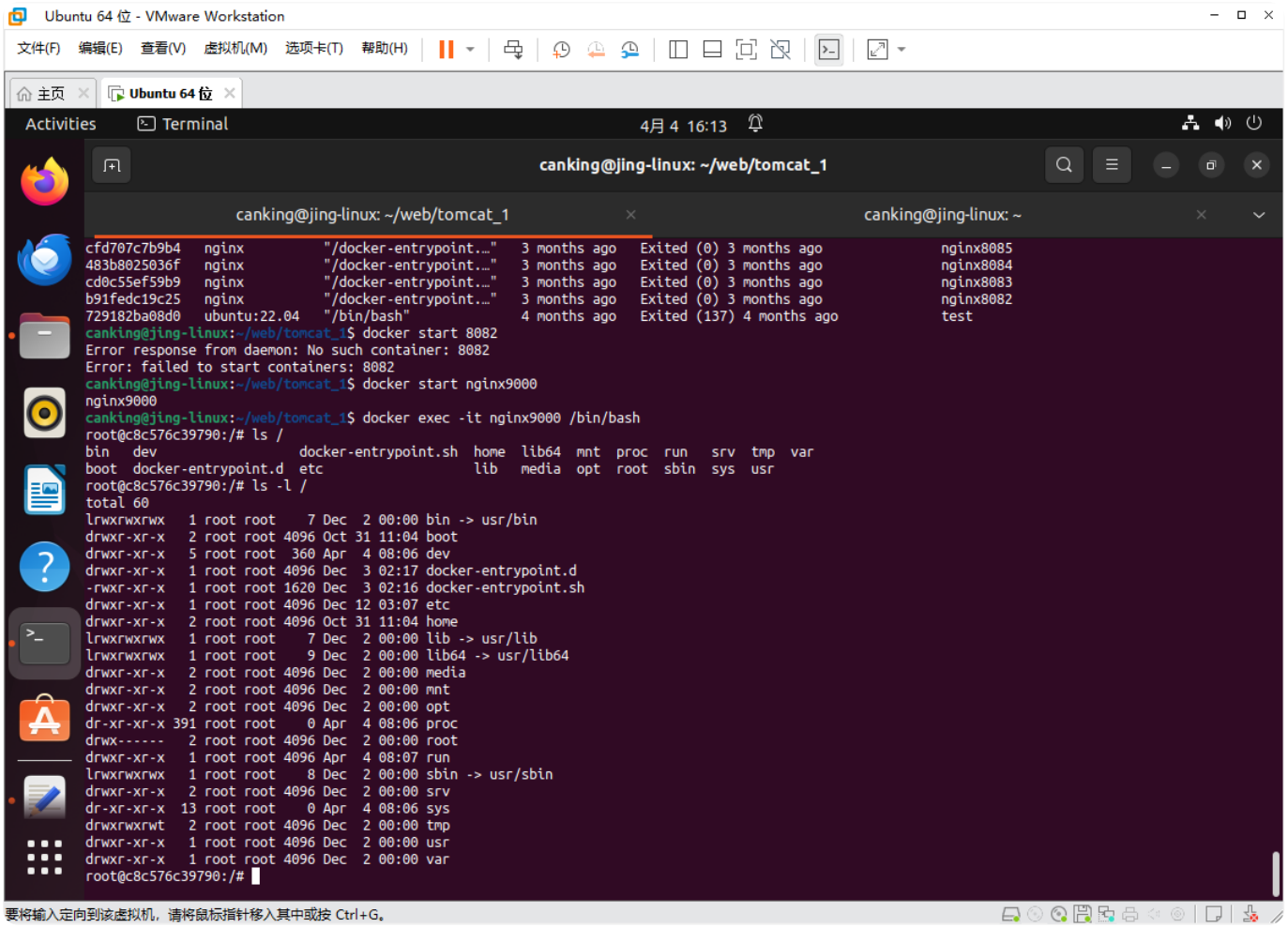
```
# 注意这里的工作路径是 ~/web/nginx-cluster/
docker run -d -it -p 9000:80 --name nginx9000 -v ./conf:/etc/nginx -v
./www:/usr/share/nginx/html nginx
```

- 9000端口是ubuntu上的端口，80端口是容器内部的端口。意思就是当我访问ubuntu 9000端口，docker会自动转发到该容器的80端口，容器内的nginx监听80端口的转发，再响应回去

- `-v`选项是绑定文件，将`~/web/nginx-cluster/conf`关联到容器内部的`/etc/nginx`。

1.3.5 进入容器内部查看目录结构

```
docker exec -it nginx9000 /bin/bash # 进入容器内部

ls -l / # 查看目录
```



1.3.6 测试nginx能否启用

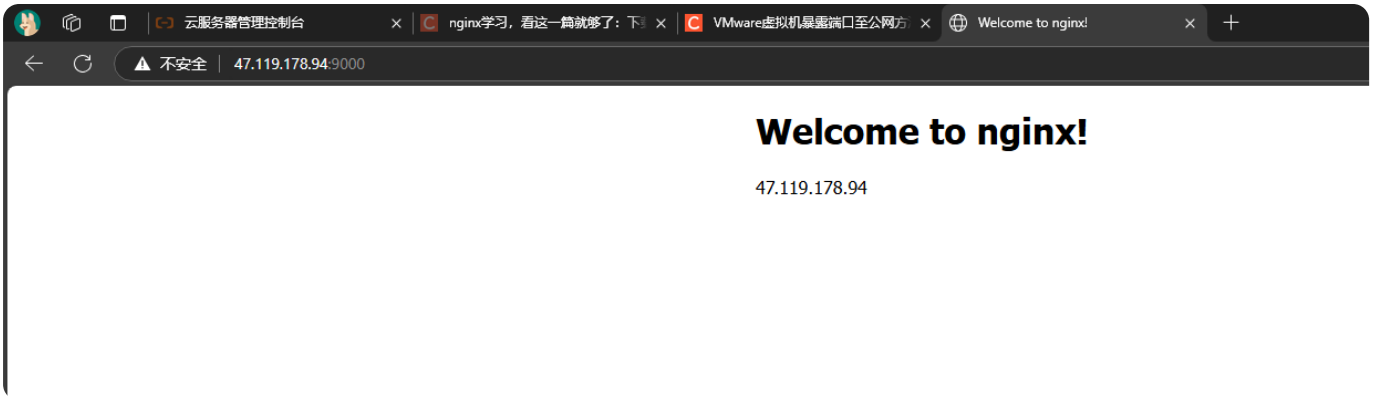
配置`~/web/nginx-cluster/conf/conf.d/default.conf`即可修改容器中对应的配置文件内容

配置`nginx-cluster/www/index.html`看nginx是否可以启用，注意要启动容器

```
nginx-cluster/www/index.html 内容：
<!DOCTYPE html>
<html>
```

```
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>47.119.178.94</p>
</body>
</html>
```

访问的是9000端口，而不是80端口



> **Tip**
>
> 如果忘记一个docker容器的 创建命令 和 关联的文件，可以运行`dokcer inspect container_name`来查看具体的配置

1.4 nginx管理命令

下面是常用的nginx命令，适用于安装了nginx的ubuntu。如果是用docker 拉取nginx镜像再运行镜像，这命令不适用。

```
nginx -v # 查看版本号

service nginx start

service nginx restart

service nginx stop

sudo nginx -t 检查配置文件是否正确

sudo nginx 默认启动nginx

sudo nginx -s [stop, reload, ...] 停止、重启等
```

接下来了解一个nginx 的配置文件的知识

2. nginx 配置文件知识学习

具体内容查看配置详解，下面只记录常用配置

Nginx中 配置文件 nginx.conf 详解-CSDN博客

nginx学习：配置文件详解，负载均衡三种算法学习，上接nginx实操篇_负载均衡如何处理文件-CSDN博客

多查官方文档或其他文档

三个部分：

1. **全局块**：`worker_processes` 为并发处理量

> **Note**
>
> 从配置文件开始到 events 块之间的内容，主要会设置一些影响nginx 服务器整体运行的配置指令，主要包括配 置运行 Nginx 服务器的用户（组）、允许生成的 worker process 数，进程 PID 存放路径、日志存放路径和类型以 及配置文件的引入等。

2. **events块**

> **Note**
>
> events 块涉及的指令**主要影响 Nginx 服务器与用户的网络连接，常用的设置包括是否开启对多 work process 下的网络连接进行序列化，是否 允许同时接收多个网络连接，选取哪种事件驱动模型来处理连接请求，每个 word process 可以同时支持的最大连接数等。**
> 上述例子就表示每个 work process 支持的最大连接数为 1024.
> 这部分的配置对 Nginx 的性能影响较大，在实际中应该灵活配置。

3. **http 块**： Nginx 服务器配置中最频繁的部分

> **Note**
>
> **http全局块：**
> http全局块配置的指令包括文件引入、MIME-TYPE 定义、日志自定义、连接超时时间、单链接请求数上限等。
>
> **server 块：**
> 这块和虚拟主机有密切关系，虚拟主机从用户角度看，和一台独立的硬件主机是完全一样的，该技术的产生是为了 节省互联网服务器硬件成本。
> 每个 http 块可以包括多个 server 块，而每个 server 块就相当于一个虚拟主机。
> 而每个 server 块也分为全局 server 块，以及可以同时包含多个 locaton 块。
>
> **全局 server 块:**
> 最常见的配置是本虚拟机主机的监听配置和本虚拟主机的名称或IP配置。
>
> **location 块:**
> 一个 server 块可以配置多个 location 块。
> 这块的主要作用是基于 Nginx 服务器接收到的请求字符串（例如 server_name/uri-string），对虚拟主机名称 （也可以是IP 别名）之外的字符串（例如 前面的 /uri-string）进行匹配，对特定的请求进行处理。 地址定向、数据缓 存和应答控制等功能，还有许多第三方模块的配置也在这里进行。

```
# 去掉注释后简洁的nginx.conf
worker_processes  1;

events {
```

```
    worker_connections S 1024;
}


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

    server {
        listen       80;
        server_name  localhost;

        location / {
            root   html;
            index  index.html index.htm;
        }

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
```

2.1 全局块

```
# 用户组
user myUsr myGroup;

# 工作进程数
worker_processes  1;

# 进程文件路径
pid /user/local/nginx/nginx.pid;

# 日志路径和日志级别
error_log logs/error.log debug;
```

2.2 event块

```
events {
    # 设置网路连接序列化
    accept_mutex on;

    # 一个进程是否同时接受多个网络连接
    multi_accept on;

    # 事件驱动模型
    use epoll;

    # 最大连接数
    worker_connections  1024;
}
```

2.3 http块

```
# 1. http_proxy模块
http {
```

```
    ...
    # 配置 https_proxy 反向代理
    proxy_connect_timeout  75; # 连接超时时间
    proxy_read_timeout  75; # 表示 Nginx 与代理服务器两个成功的响应操作之间超时时间
    proxy_send_timeout 100; # 表示 Nginx 传输文件至代理服务器的超时时间；
    proxy_buffer_size  4k; # 用于设置从代理服务器读取并保存用户头信息的缓冲区大小；
    proxy_buffers  4  32k; # 设置代理缓冲区大小
    proxy_busy_buffers_size  64k;  # 设置高负荷下的缓冲大小
    proxy_max_temp_file_size  64k;
    proxy_temp_file_write_size 64k;
    proxy_temp_path  proxy_temp; # 用于指定临时文件所在的目录；
    ...
}

# 2. http_gzip模块
http {
    ...
    # 配置 http_gzip 模块
    gzip on; # on表示开启gzip压缩输出，可减少网络传输
    gzip_min_length 1K; # 设置允许压缩的页面最小字节数（到达这个大小才进行压缩）
    gzip_buffers  4  16k; # 设置系统获取多少个单位的缓存用于存储 gzip 的压缩结果数据流
    gzip_http_version 1.0; # 设置 http 协议的版本

    # zip 压缩比，为 1 时，压缩比最小处理速度最快；为 9 时，压缩比最大但处理速度最慢；
    gzip_comp_level 6;

    # 匹配 mime 类型进行压缩，无论是否指定，text/html 类型总是会被压缩的；
    gzip_types text/plain text/css application/json;

    gzip_proxied any;
    gzip_vray on;
    ...
}

# 3. 负载均衡
upstream backend {
    server 192.168.56.10:8080 max_fails=2 fail_timeout=30s backup;
    server 192.168.56.11:8080 max_fails=2 fail_timeout=30s;
}
```

2.4 server块

```
server {
    # 监听端口
    listen 8080;

    # 监听服务器地址
    server_name 192.168.56.10;

    # 每个连接请求上限次数
    keepalive_requests 120;

    # 字符集
    charset utf-8;

    # 服务日志所在目录以及日志格式
    access_log logs/host80.log myLogFormat;

    # 错误页，对应错误代码显示对应的网页
    error_page  404  /404.html;
    error_page  500 502 503 504  /50x.html;
}
```

2.5 location 块

正则表达式的匹配规则，以及一些全局变量信息

nginx.conf location匹配规则讲解_nginx location匹配规则-CSDN博客

```
~ # 表示采用正则匹配，区分大小写
~* # 表示采用正则匹配，不区分大小写
^~ # 表示普通字符匹配，如果该选项匹配，只匹配该选项，不匹配其他。一般用来匹配目录
= # 精确匹配
@ # 定义一个命名的location，使用在内部定向时
```

```
# 1. 精确匹配 精确匹配请求的URI完全等于/的请求。
location = / {
    # 配置
}

# 2. 前缀匹配
# 匹配所有以/images/开头的请求，例如/images/logo.png、/images/background.jpg
等。
location /images/ {
    # 配置
}

# 3. 正则表达式匹配
# 使用正则表达式匹配所有以.jpg、.jpeg、.png或.gif结尾的请求
# \ 表示转义字符
location ~* \.(jpg|jpeg|png|gif)$ {
    # 配置
}

# 4. 正则表达式匹配（区分大小写)
# 只会匹配.php结尾的请求
location ~ \.php$ {
    # 配置
}

# 5. 最长非正则匹配:
# 当请求/app/api/时，Nginx会匹配最长的前缀，即/app/api/而不是/app/。
location /app/ {
    # 配置
}
location /app/api/ {
    # 配置
}

# 6. 命名location:
# 这是一个命名的location，可以通过error_page指令引用。
location @fallback {
    # 配置
}
# 7. 匹配空路径
#一个location匹配精确的/index.html请求
```

```
# 个location尝试匹配请求的URI，如果找不到，则使用@fallback
location = /index.html {
    # 配置
}
location / {
    try_files $uri $uri/ @fallback;
}

# 8. 匹配特定HTTP方法：
# 这个location匹配所有以/api/开头的请求，但限制除了GET方法以外的所有方法。
location /api/ {
    limit_except GET {
        deny all;
    }
}

# Nginx会根据配置文件中的顺序来匹配location，优先级从高到低依次是：精确匹配、最长非正则匹配、正则表达式匹配、前缀匹配
```

Important

实现前端路由通配，这样文件的目录路径即是url路径

```
# 使用通配来配置前端路由
location / {
 try_files $uri $uri.html $uri/ =404;
}
```

`try_files $uri $uri.html $uri/ =404`

- 当访问 `/about` 时，Nginx 会按顺序检查：
    1. 是否存在 `/about` 文件 → 无
    2. 是否存在 `/about.html` 文件 → **有！直接返回**
    3. 是否存在 `/about/` 目录 → 无
    4. 返回 404（如果前 3 步均失败）
-

```
# 1. 基本使用
location  ~*  ^.+$ {
    # 服务器的默认网站根目录位置
```

```
    root /var/www/html;

    # 默认访问的文件名
    index  index.html index.htm index.jsp;

    # 拒绝的 IP
    deny 192.168.56.21;
    deny all;

    # 允许的 IP
    allow 192.168.56.10;
    allow all;

    # 反向代理服务器地址
    proxy_pass  http://192.168.56.33;
    # 是否重定向代理服务器地址
    proxy_redirect off;
}

# 2. 设置向代理服务器发送请求时的请求头数据：
location {

    # cookie
    proxy_pass_header  Set-Cookie;
    # 主机名
    proxy_set_header Host $host;
    # 真实 IP
    proxy_set_header X-Real-Ip $remote_addr;
    # 表示 HTTP 请求端真实 IP
    proxy_set_header X-Forwarded-For $remote_addr;
}

# 3. 访问控制和处理跨域问题
location {
    # 设置允许跨域类型
    add_header Access-Control-Allow-Origin * always;

    # 是否允许信任证书
    add_header Access-Control-Allow-Credentials 'true' always;

    # 允许的请求头类型
    add_header Access-Control-Allow-Headers * always;

    # 设置允许的请求方式
```

```
    add_header Access-Control-Allow-Methods 'PUT, GET, POST, DELETE,
OPTIONS' always;

    # 处理 OPTIONS 请求
    if ($request_method = 'OPTIONS') {
        return 204;
    }
}
```

2.6 upstream块

查该网址

nginx学习：配置文件详解，负载均衡三种算法学习，上接nginx实操篇_负载均衡如何处理文件-CSDN博客nginx学习：配置文件详解，负载均衡三种算法学习，上接nginx实操篇_负载均衡如何处理文件-CSDN博客

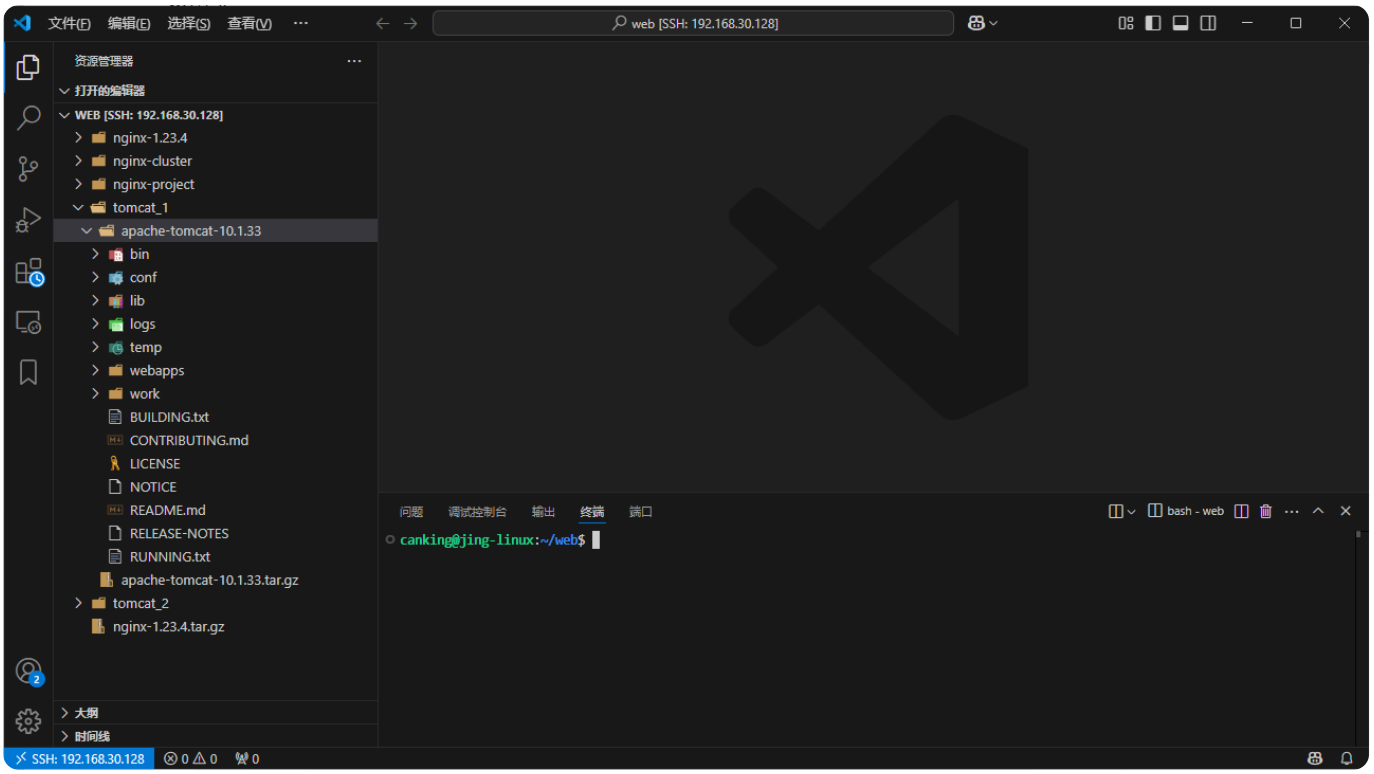
接下来进入nginx 的实战实践

3. 反向代理

3.1 准备工作

通过自动安装好`nginx`后，创建一个`~/web`目录用来存放我们相关的实践文件。

使用vscode通过ssh连接ubuntu，这里直接连接到了`~/web`目录中，**方便我们编写文件**，接着下载tomcat安装包，在ubuntu上解压

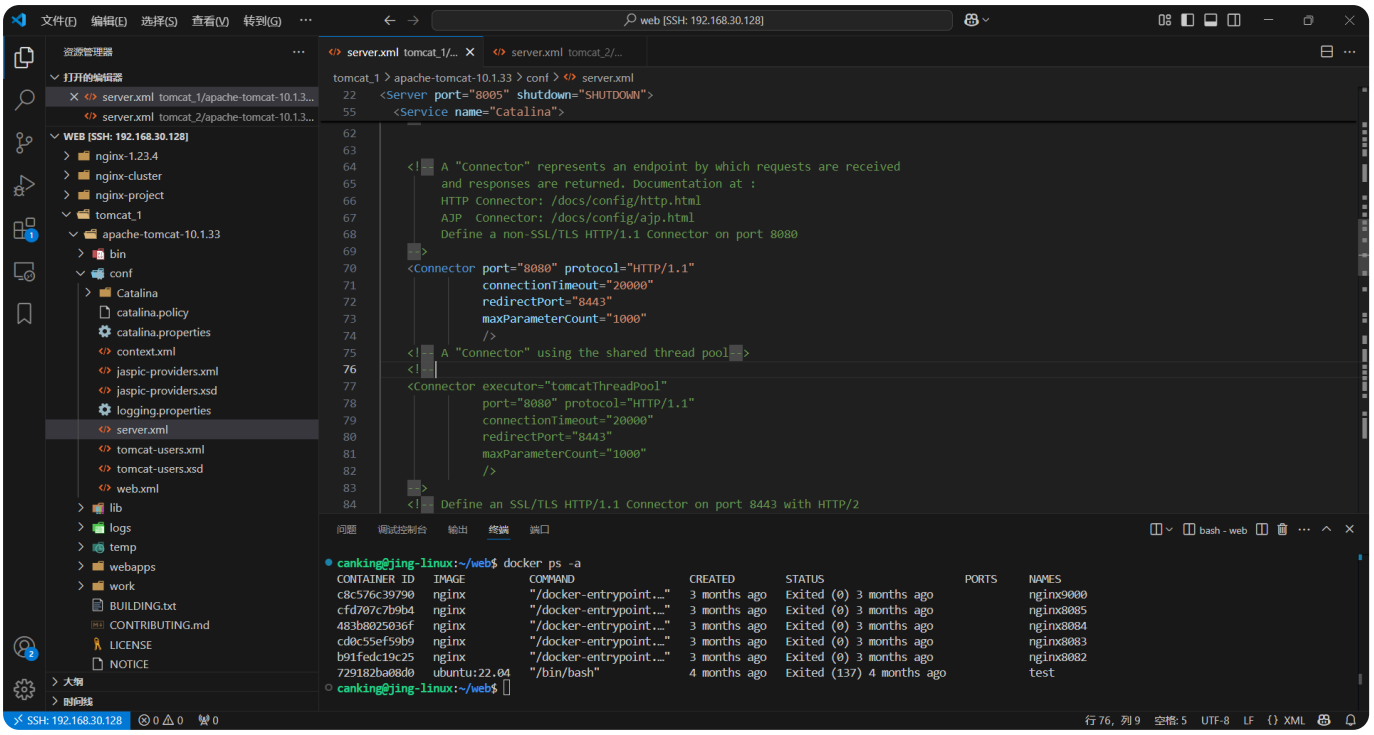


这里准备了两台tomcat，可以清晰的看到目录结构

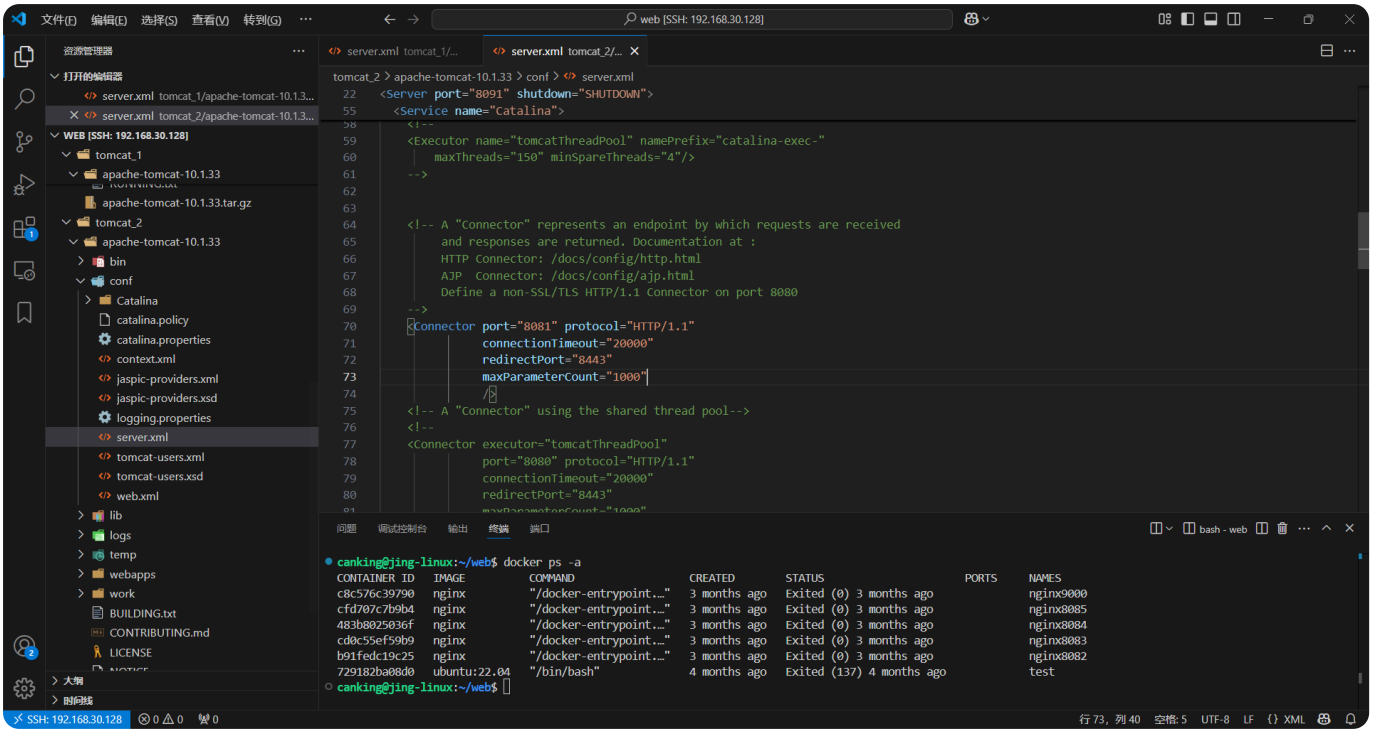
```
# 1. 如何解压
tar -xzvf apache-tomcat-10.1.33.tar.gz -C ~/web/tomcat_1/apche-tomcate-10.1.33
tar -xzvf apache-tomcat-10.1.33.tar.gz -C ~/web/tomcat_2/apche-tomcate-10.1.33

# 2. 修改解压后出现的权限问题
# 将解压安装好的tomcat的归属者改为当前用户即可
sudo chown -R canking:root ~/web/tomcat_1/apche-tomcate-10.1.33
sudo chown -R canking:root ~/web/tomcat_2/apche-tomcate-10.1.33
```

tomcat的端口配置

打开`conf/server.xml`文件，找到`Connector`标签，这里`tomcat_1`默认设置为 `port = 8080` 端口。如果8080被其他服务占用了，请自行分配其他端口



打开`tomcat_2`的`conf/server.xml`，这里端口改为8081，如果8081被占用，请自行分配端口

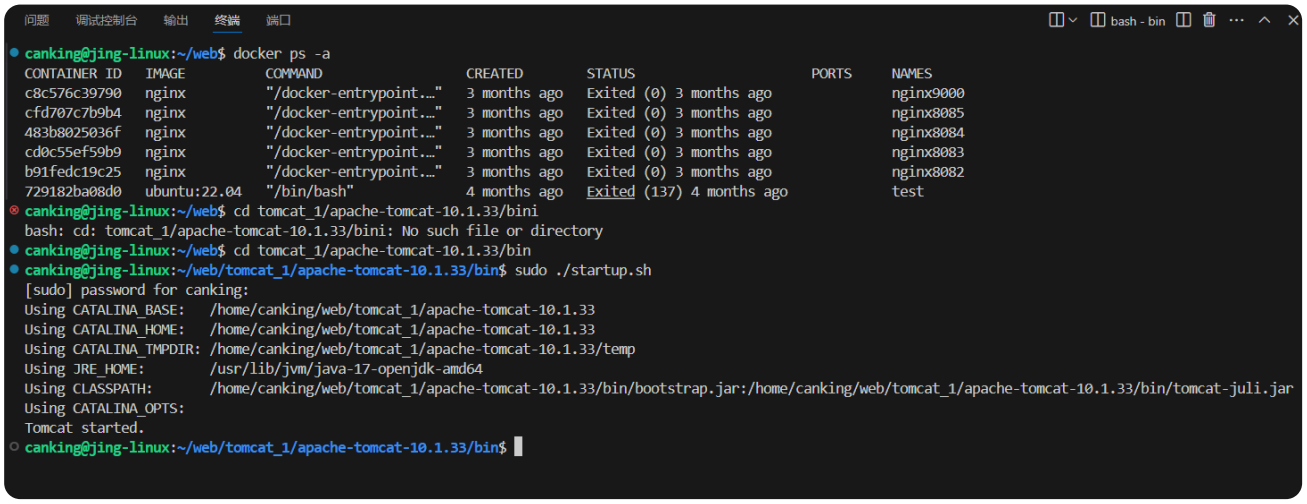


3.2 代理一台服务器

实现效果：在windows访问`www.123.com`出现tomcat页面

1. 准备好一台tomcat服务器，这里我们启动`tomcat_1`

```
# 进入tomcat_1/bin目录下启动服务
sudo ./startup.sh # 启动服务
sudo ./shutdown.sh # 关闭服务
```



2. nginx配置 `/etc/nginx/sites-available`目录下的`default`文件

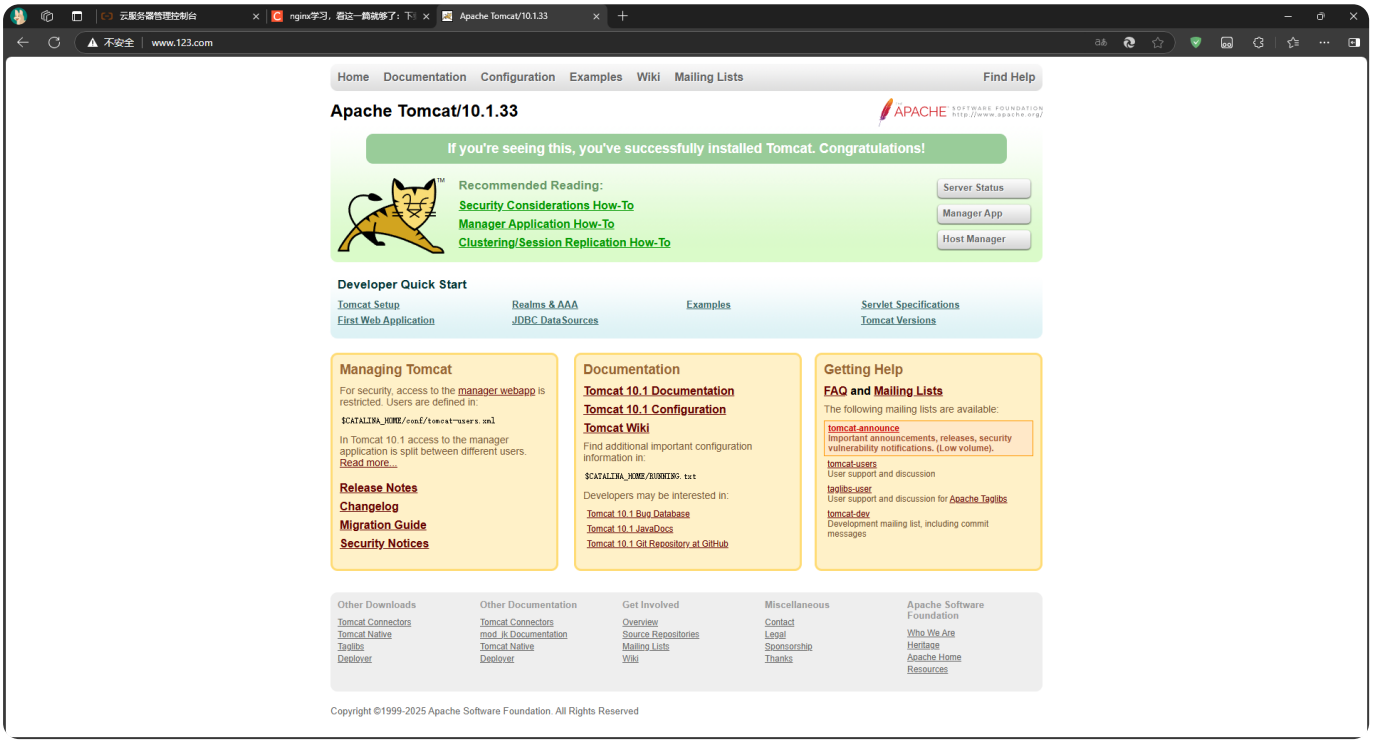
```
server {
    # 反向代理1
    listen 80;
    server_name www.123.com;

    location / {
        # tomcat 和 nginx在同一台ubuntu上，使用127.0.0.1回环地址即可
        proxy_pass  http://127.0.0.1:8080/;
        # 下面配置可以忽略
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

配置好了之后，执行`sudo service nginx start`启动 nginx

3. windows 配置：`C:/Windows/System32/drivers/etc/host`文件内添加

```
# ubuntu-ip 域名，这里可以用ifconfig查看自己ubuntu的ip地址是什么
192.168.30.128 www.123.com
```

4. 最终测试：我们在浏览器中访问`www.123.com`即可

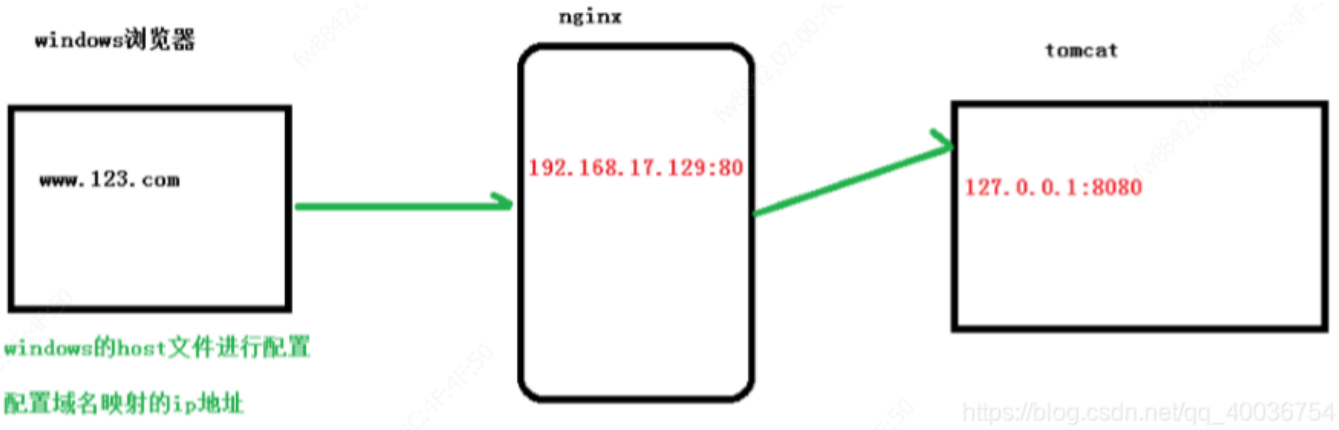


注意事项：

- tomcat加载需要一定时间，运行后windows直接访问有时候会加载不出来，需要等待一会
- 修改配置文件内容后nginx 需要重启
- 代理之前，可以先测试tomcat 和 nginx 是否正常运行

总结

这是整个代理的流程，我们浏览器访问的是`www.123.com`，这时候windows会去访问`192.168.30.128:80`端口，nginx监听的是80端口，nginx再将这个访问通过回环地址`127.0.0.1`代理到tomcat的`8080`端口上，最后tomcat再响应访问，从而能够看到tomcat的页面

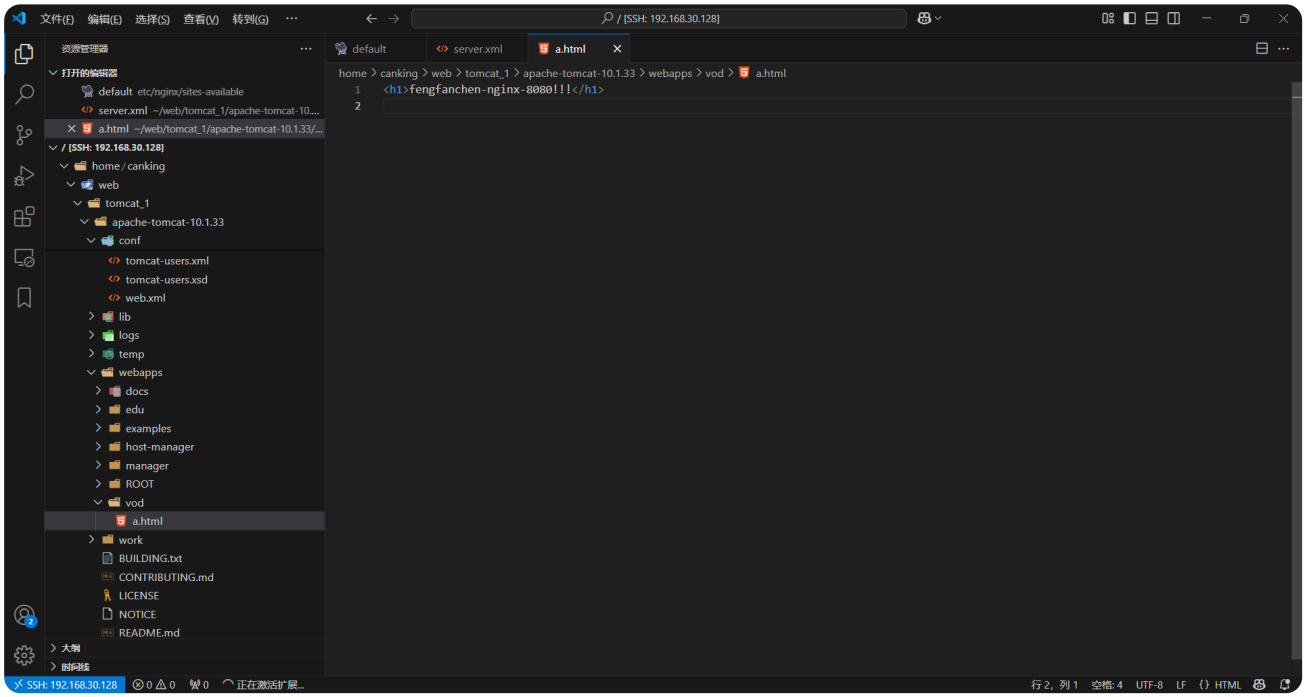


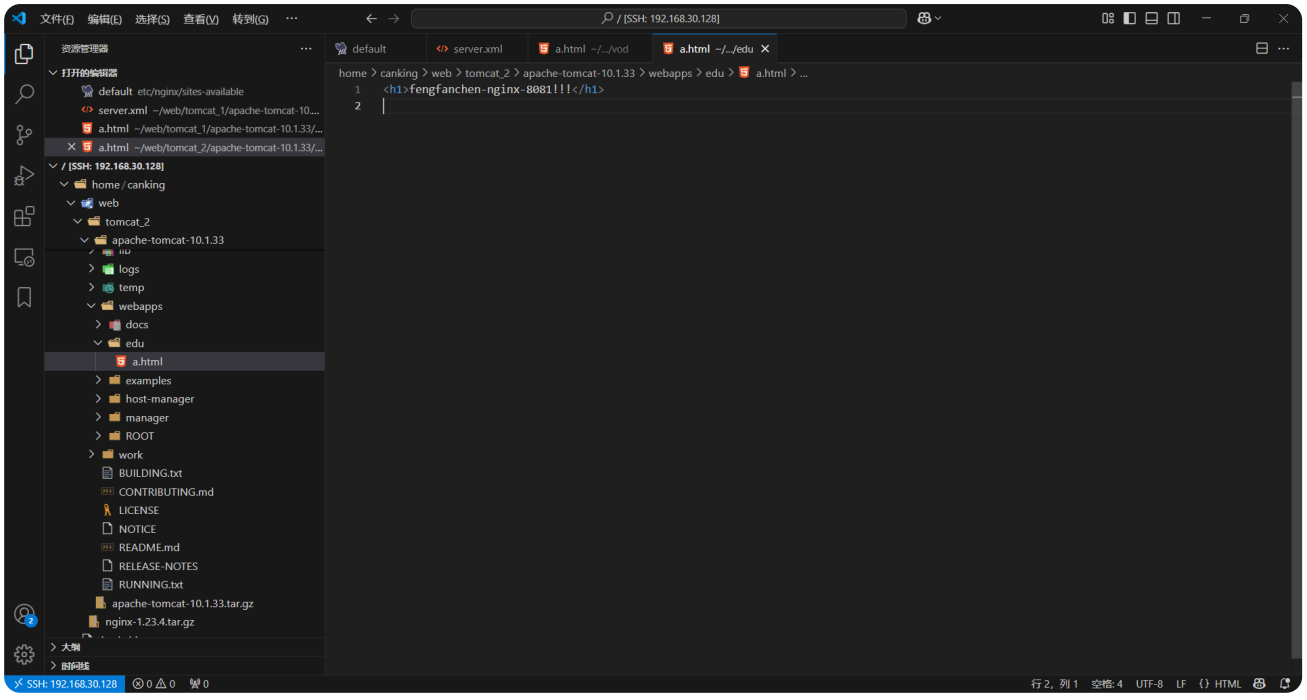
3.3 代理多台服务器

实现效果：两台tomcat服务器，用nginx代理，根据路径选择不同的服务器

1. 准备两台tomcat服务器，即前面的`tomcat_1`,`tomcat_2`
2. 在`tomcat_1`中创建`conf/webapps/vod/a.html`。在`tomcat_2`中创建`conf/webapps/edu/a.html`

```
<!--tomcat_1/apache-tomcat-10.1.33/webapps/vod/a.html -->
<h1>fengfanchen-nginx-8080!!!</h1>

<!--tomcat_2/apache-tomcat-10.1.33/webapps/edu/a.html -->
<h1>fengfanchen-nginx-8081!!!</h1>
```





3. nginx 配置。修改nginx后重新启动`sudo service nginx restart`

```
server {
    # 反向代理2
    listen 80;
    server_name 192.168.30.128;

    # 根据访问路径分配代理
    location ~ /vod/ {
        # 如果前缀是 /vod/ 转发到 8080
        proxy_pass http://127.0.0.1:8080;
    }
    location ~ /edu/ {
        # 如果前缀是 /edu/ 转发到 8081
        proxy_pass http://127.0.0.1:8081;
    }
}
```

4. 启动两台tomcat。分别进入到`bin/`目录下执行`sudo ./startup.sh`即可
5. 这里我们就通过 url 地址访问目标页面。访问的都是80端口

注意：

- 在实现反向代理前测试两台tomcat服务器能不能之间访问
- tomcat加载较慢，访问需等待一段时间

总结

我们访问的是nginx监听的80端口，nginx根据访问的地址，代理到不同的tomcat上，比如我们访问`vod/a.html`，就代理到了`tomcat_1`，访问`edu/a.html`，就代理到了`tomcate_2`

4. 负载均衡实践

Nginx的6种负载均衡策略【轮询/加权轮询 weight/ip_hash/least_conn/urlhash/fair】 - 天才卧龙 - 博客园

六种负载均衡策略：

1. 轮询策略：默认策略，实际上也是权重轮询，所有权重都默认为1
2. 轮询加权策略：手动设置权重，权重越大，服务器越容易被访问
3. ip_hash 策略：根据ip计算hash，每个请求按访问 ip 的 hash 结果分配，这样每个访客固定访问一个后端服务器

4. least_connn策略
5. url_hash 策略
6. fair策略：nginx默认不支持，需下载第三方模块，按后端服务器的响应时间来分配请求，响应时间短的优先分配。

4.1 通过权重实现负载均衡

我们在`tomcat_1`中创建`conf/webapps/edu/a.html`，跟`tomcat_2`的目录和文件名相同

```
<h1>fengfanchen-nginx-8080!!!</h1>
```

设置default文件，`weight`即为权重，权重越大越容易分配给该服务器

```
upstream myserver{
    server 192.168.30.128:8080 weight=5;
    server 192.168.30.128:8081 weight=3;
}
server {
    listen       80;
    server_name  192.168.30.128;
    location / {
        root   html;
        proxy_pass   http://myserver;
        index  index.html index.htm;
    }
}
```

重启nginx。`sudo service nginx restart`

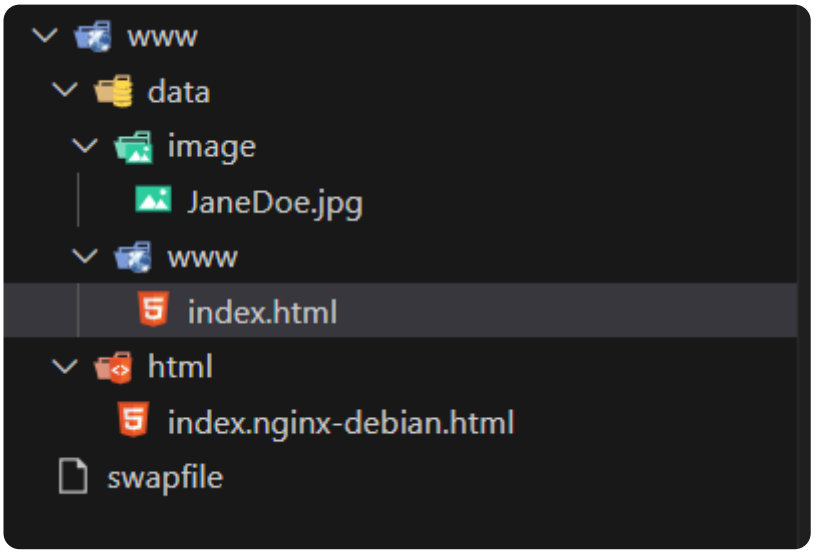
接着浏览器访问`192.168.30.128/edu/a.html`，刷新几次，会发现有时候是代理到8080，有时候是代理到8081

5. 动静分离

静态资源可以通过nginx部署，动态资源可以通过反向代理由tomcat服务器响应

1. **给nginx配置静态资源。**这里创建一个html文件，同时往image中随便放一张图片

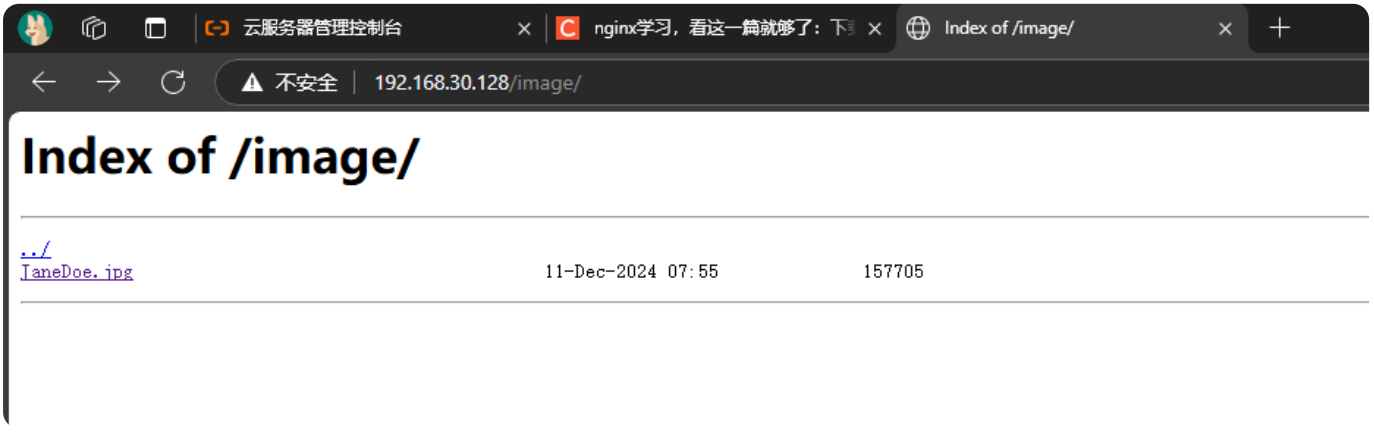
```
# 资源目录结构
/var/www/data/
    |__ www/index.html
    |__ image/JaneDoe.jpg 图片

www/index.html 文件中
<h1>fengfanchen-test-html</h1>
```

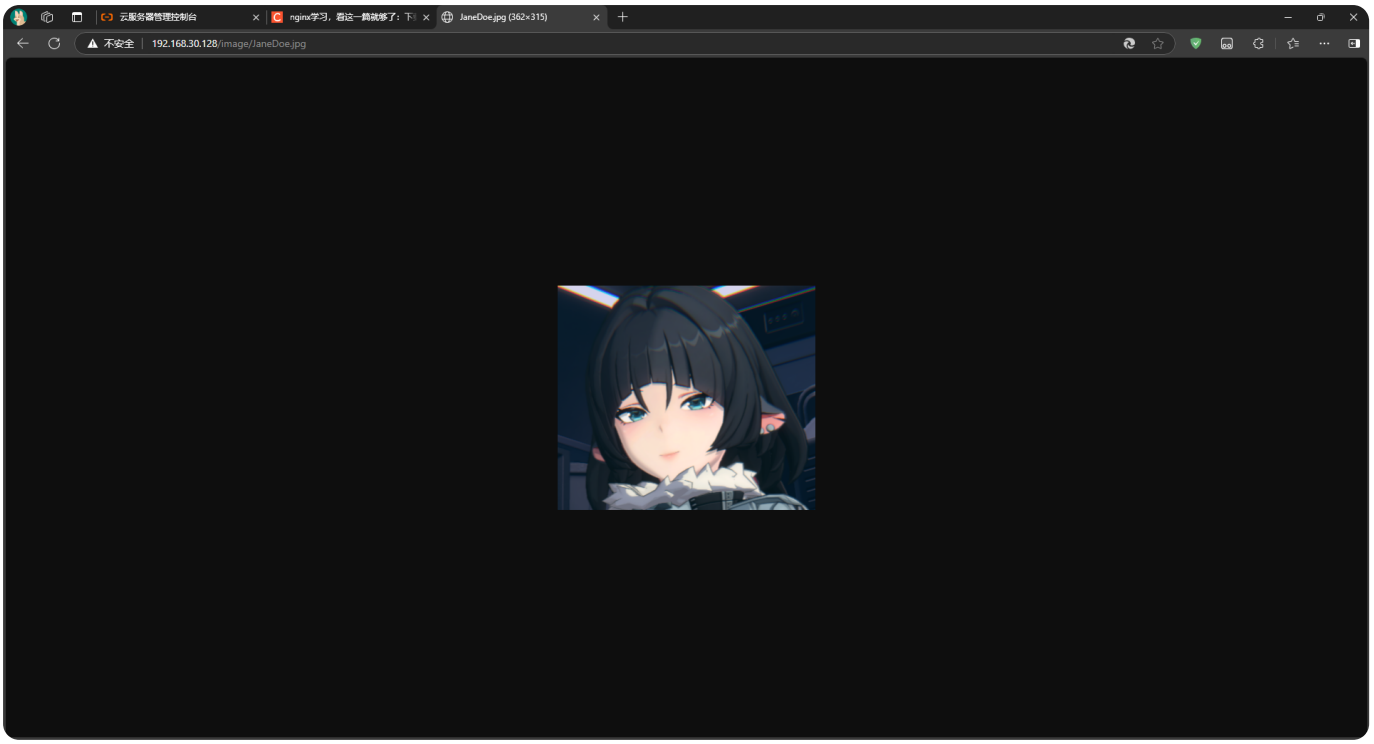


2. 配置nginx default

```
# nginx配置
upstream myserver{
    server 192.168.30.128:8080 weight=5;
    server 192.168.30.128:8081 weight=3;
}

server {
    listen 80;
    server_name 192.168.30.128;

    location / {
        root html;
        proxy_pass http://myserver;
        index index.html index.htm;
    }

    # 部署静态资源
    location /www/ {
        root /var/www/data/; # root为根路径
        autoindex on;

    }

    location /image/ {
        root /var/www/data/; # root为根路径
        autoindex on; # 列出当前目录的内容
    }
}
```

访问`:80/image/`即可看到这里显示了图片，点击后即可看到静态资源的内容





与此同时访问`:80/www/`路径可以看到我们的html页面

> **Caution**
>
> **注意：**
>
> - 浏览器访问`/image/index.html`是无法查看`data/www/index.html`的
> - 因为root 设定了根目录，访问`/image/index.html`实际上是访问`data/www/image/`目录，然后在该目录下寻找`index.html`，所以无法找到

6. nginx高可用集群

6.1 一台服务器

只有一台服务器，使用docker部署nginx实现高可用集群和负载均衡，没有使用keepalive
基于下面文章实践：

使用docker搭建nginx集群,实现负载均衡 - 知乎

6.2 两台服务器（待完善）

两台服务器，使用docker实现nginx高可用集群，通过docker使用keepalive：

docker部署Nginx+keepalive实现高可用_docker keepalive+nginx-CSDN博客

使用docker环境实现keepalived高可用_docker keepalived-CSDN博客

下面链接是keepalive介绍

keepalived工作原理-CSDN博客

简述keepalived工作原理-CSDN博客

6.2.1 一些配置信息

- 所使用的两台服务器：
    1. 本地ubuntu虚拟机，浏览器访问虚拟机本地内网ip进行测试
    2. ECS服务器，ubuntu系统，浏览器访问ECS的公网ip进行测试

- 我的nginx目录结构
    1. ecs上：

```
# nginx-cluster为自建文件夹
~/nginx-cluster/
    |__ conf/
    |   |__conf.d
    |   |   |__default
    |   |#其他文件省略
    |__ www
        |__ index.html
```

2. 本地ubuntu上：

```
~/web/nginx-cluster/
  |__ conf/
  | |__conf.d
  | |   |__default
  | |#其他文件省略
  |__ www
      |__ index.html
```

6.2.2 容器启动测试

关键配置内容：

- conf/conf.d/default 内容

```
server {
    listen       80;
    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

- www/index.html内容

```
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>192.168.30.128</p> <!--此处ip地址可自己修改 -->
</body>
</html>
```

容器启动：

```
# 这里开放9000端口，注意-v 挂载写在最后面，否则创建好的容器有bug无法允许
# ecs注意配置安全组，暴露9000端口，否则会被防火墙拦截
docker run -d -it -p 9000:80 --name nginx9000 \
-v ./conf:/etc/nginx \ # 注意./conf 是当前目录下的conf
-v ./www:/usr/share/nginx/html \
nginx # 默认nginx:latest
```

测试html是否显示:
浏览器访问 `<ip>:9000/` 测试html是否显示

6.2.3 加入keepalive

首先，windows上的虚拟机Ubuntu要配置端口转发，目的是为了让ecs服务器可用通过windows公网ip访问到ubuntu的nginx服务，而ecs服务器自带公网ip，所以不用配置。最后windows测试127.0.0.1:9000端口，出现nginx的页面。
VMware虚拟机暴露端口至公网方法流程详解_虚拟机网络暴露-CSDN博客

要在一台ECS云服务器和一台通过Windows端口转发的Ubuntu虚拟机上使用Docker实现Nginx的高可用集群，并配置虚拟IP，您可以按照以下步骤进行配置：

1. 环境准备

确保您在两台服务器上都安装了Docker。您可以通过以下命令检查Docker是否已安装：

```
docker --version
```

如果未安装Docker，请根据您的操作系统安装Docker。

2. 拉取Nginx和Keepalived镜像

在两台服务器上分别拉取Nginx和Keepalived镜像：

```
# 拉取Nginx镜像
docker pull nginx

# 拉取Keepalived镜像
docker pull osixia/keepalived
```

3. 创建Docker网络

在ECS和Ubuntu虚拟机上创建一个Docker网络，以便容器之间可以相互通信：

```
docker network create my_bridge_network
```

4. 启动Nginx容器

在两台服务器上启动Nginx容器。您可以为每个Nginx容器分配一个静态IP地址：

```
# 在ECS上启动Nginx容器
docker run -d --name nginx1 --network my_bridge_network --ip
172.18.0.2 nginx

# 在Ubuntu虚拟机上启动Nginx容器
docker run -d --name nginx2 --network my_bridge_network --ip
172.18.0.3 nginx
```

5. 配置Keepalived

5.1 编辑Keepalived配置文件

在ECS和Ubuntu虚拟机上分别创建Keepalived的配置文件。您可以使用以下示例配置：

```
# 在ECS上编辑Keepalived配置文件
sudo vi /etc/keepalived/keepalived.conf
```

配置文件示例：

```
vrrp_instance VI_1 {
 state MASTER
 interface eth0  # 根据实际情况修改
 virtual_router_id 51
 priority 100  # 主节点优先级高
 advert_int 1
 authentication {
     auth_type PASS
     auth_pass 1111  # 认证密码
 }
 virtual_ipaddress {
     192.168.0.100  # 虚拟IP地址
 }
}
```

在Ubuntu虚拟机上创建相似的配置，但将`state`设置为`BACKUP`，并将`priority`设置为一个较低的值（例如`90`）：

```
# 在Ubuntu虚拟机上编辑Keepalived配置文件
sudo vi /etc/keepalived/keepalived.conf
```

配置文件示例：

```
vrrp_instance VI_1 {
 state BACKUP
 interface eth0  # 根据实际情况修改
 virtual_router_id 51
 priority 90  # 备份节点优先级低
 advert_int 1
 authentication {
     auth_type PASS
     auth_pass 1111  # 认证密码
 }
 virtual_ipaddress {
     192.168.0.100  # 虚拟IP地址
 }
}
```

5.2 启动Keepalived容器

在ECS和Ubuntu虚拟机上启动Keepalived容器：

```
# 在ECS上启动Keepalived容器
docker run -d --name keepalived_master --net=host --cap-add=NET_ADMIN
osixia/keepalived

# 在Ubuntu虚拟机上启动Keepalived容器
docker run -d --name keepalived_backup --net=host --cap-add=NET_ADMIN
osixia/keepalived
```

6. 测试配置

1. **检查虚拟IP**：在ECS上运行以下命令，检查虚拟IP是否已分配：

   ```
   ip addr
   ```

   您应该能够看到`192.168.0.100`的虚拟IP。

2. **访问Nginx**：通过虚拟IP访问Nginx，确保请求能够成功到达主节点。
3. **故障切换测试**：可以通过停止主节点上的Keepalived容器，验证备份节点是否能够接管虚拟IP。

7. 端口转发配置

由于Ubuntu虚拟机通过Windows的端口转发暴露公网，确保Windows防火墙允许相关端口的流量（如80端口和Keepalived的VRRP协议使用的端口）。

8. 注意事项

- **网络配置**：确保ECS和Ubuntu虚拟机在同一子网中，或者能够相互通信。
- **安全性**：在生产环境中，确保使用强密码和适当的安全措施来保护Keepalived和Nginx。

通过以上步骤，您可以在ECS和Ubuntu虚拟机上成功配置Nginx的高可用集群，并实现虚拟IP的管理。

7. 其他参考网站资料

- nginx详细参数配置(史上最全) - hanease - 博客园s
- 不使用docker，直接本机安装nginx和keepalived的高可用实践：
  Keepalived + Nginx 实现高可用_nginx+keepalived高可用-CSDN博客
  ubuntu配置keepalived+nginx 高可用（二） - keepalived 配置 - 蓝迷梦 - 博客园
- 虚拟ip简介