VUE配置代理解决跨域问题

vue.config.js

```
module.exports = {
    publicPath: "./",
    devServer: {
        proxy: { //代理设置仅在本地开发有效
            '/api':{
                target: 'http://localhost:3000', //目标地址
                pathRewrite: {"^/api": ""},  //重写目标地址路径
                changeOrigin: true //是否跨域
            }
        }
    }
}
```

Nginx反向代理解决跨域

一.概述

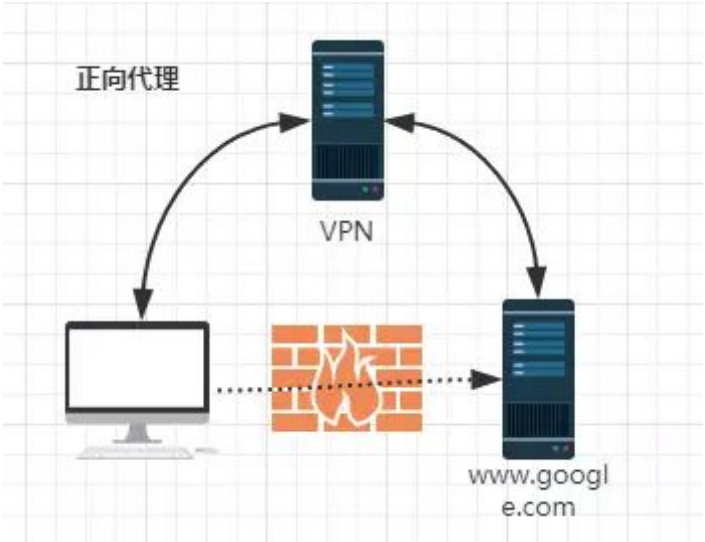
什么是nginx?

Nginx (engine x) 是一款轻量级的Web 服务器 、反向代理服务器及电子邮件（IMAP/POP3）代理服务器。

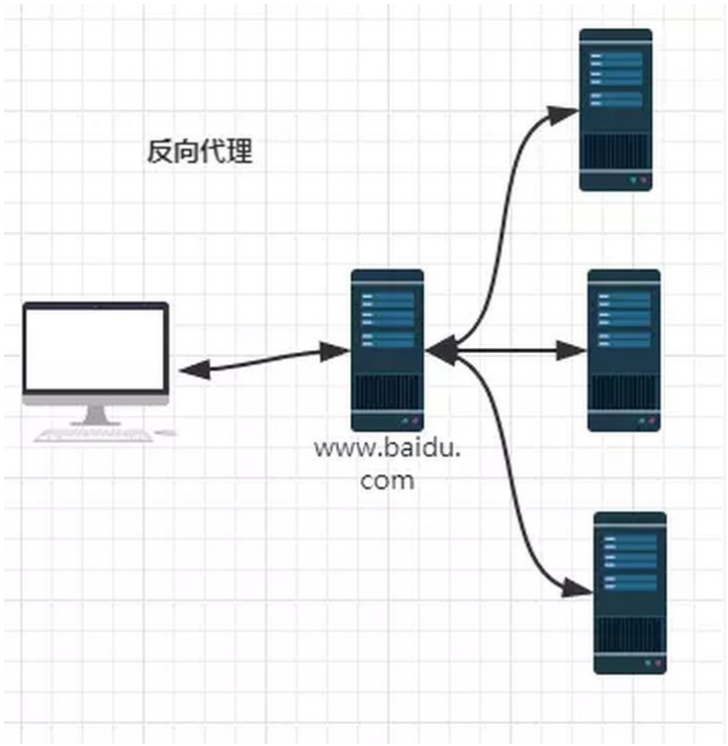
什么是反向代理?

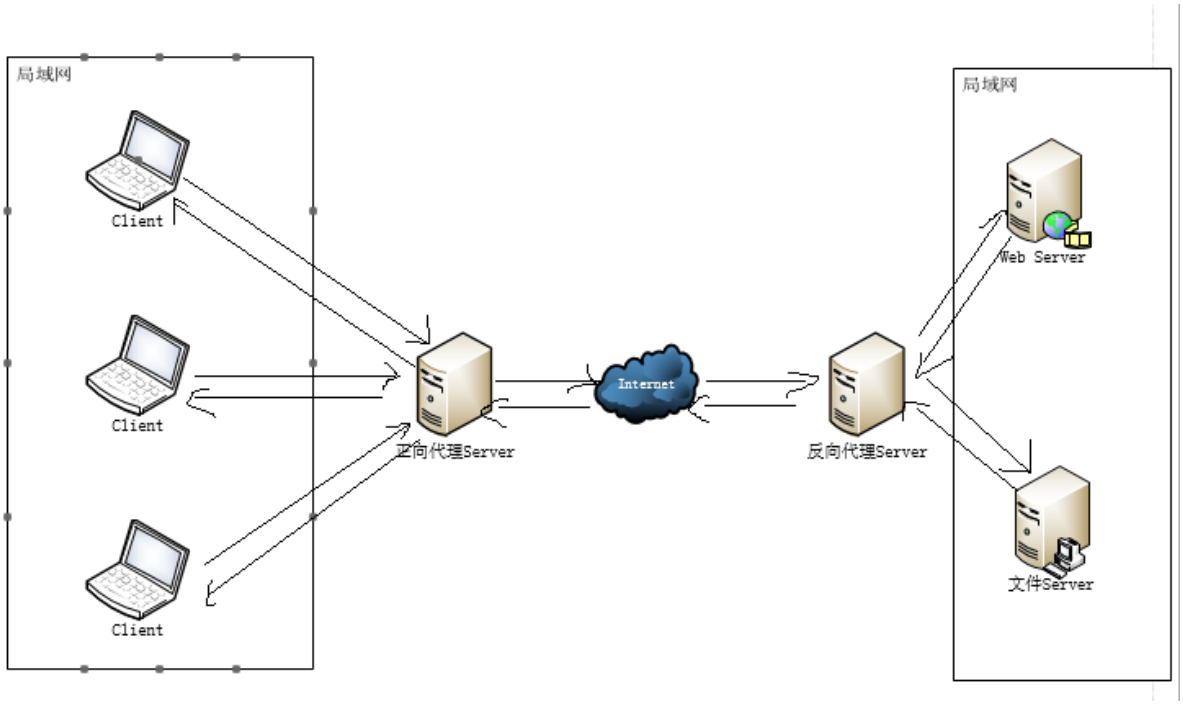
反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

正向代理



反向代理





二、常用命令

nginx 的使用比较简单，就是几条命令。

```
nginx -s stop       # 快速关闭Nginx，可能不保存相关信息，并迅速终止web服务。
nginx -s quit       #平稳关闭Nginx，保存相关信息，有安排的结束web服务。
nginx -s reload     #因改变了Nginx相关配置，需要重新加载配置而重载。
nginx -v            #显示 nginx 的版本。
nginx -V            #显示 nginx 的版本，编译器版本和配置参数。
```

nginx 配置实战

主要有：http反向代理配置、网站有多个webapp的配置、https反向代理配置、负载均衡、静态站点配置、跨域解决方案

三、http反向代理配置

我们先实现一个小目标：不考虑复杂的配置，仅仅是完成一个 http 反向代理。

nginx.conf 配置文件如下

```
#运行用户
#user somebody;

#启动进程,通常设置成和cpu的数量相等
worker_processes  1;

#全局错误日志
error_log  D:/Tools/nginx-1.10.1/logs/error.log;
error_log  D:/Tools/nginx-1.10.1/logs/notice.log  notice;
error_log  D:/Tools/nginx-1.10.1/logs/info.log  info;

#PID文件，记录当前启动的nginx的进程ID
pid        D:/Tools/nginx-1.10.1/logs/nginx.pid;

#工作模式及连接数上限
events {
    worker_connections 1024;    #单个后台worker process进程的最大并发链接数
}

#设定http服务器，利用它的反向代理功能提供负载均衡支持
http {
    #设定mime类型(邮件支持类型),类型由mime.types文件定义
    include       D:/Tools/nginx-1.10.1/conf/mime.types;
    default_type  application/octet-stream;

    #设定日志
    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    D:/Tools/nginx-1.10.1/logs/access.log main;
    rewrite_log     on;

    #sendfile 指令指定 nginx 是否调用 sendfile 函数（zero copy 方式）来输出文件，对于普通应用，
    #必须设为 on,如果用来进行下载等应用磁盘IO重负载应用，可设置为 off，以平衡磁盘与网络I/O处理速度，降低系统的uptime.
    sendfile        on;
    #tcp_nopush     on;

    #连接超时时间
    keepalive_timeout  120;
    tcp_nodelay        on;

    #gzip压缩开关
    #gzip  on;
```

```
    #设定实际的服务器列表
    upstream zp_server1{
        server 127.0.0.1:8089;
    }

    #HTTP服务器
    server {
        #监听80端口，80端口是知名端口号，用于HTTP协议
        listen       80;

        #定义使用www.xx.com访问
        server_name  www.helloworld.com;

        #首页
        index index.html

        #指向webapp的目录
        root D:\01_Workspace\Project\github\zp\SpringNotes\spring-
security\spring-shiro\src\main\webapp;

        #编码格式
        charset utf-8;

        #代理配置参数
        proxy_connect_timeout 180;
        proxy_send_timeout 180;
        proxy_read_timeout 180;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarder-For $remote_addr;

        #反向代理的路径（和upstream绑定），location 后面设置映射的路径
        location / {
            proxy_pass http://zp_server1;
        }

        #静态文件，nginx自己处理
        location ~ ^/(images|javascript|js|css|flash|media|static)/ {
            root D:\01_Workspace\Project\github\zp\SpringNotes\spring-
security\spring-shiro\src\main\webapp\views;
            #过期30天，静态文件不怎么更新，过期可以设大一点，如果频繁更新，则可以设置得小一
点。
            expires 30d;
        }

        #设定查看Nginx状态的地址
        location /NginxStatus {
            stub_status           on;
            access_log            on;
            auth_basic            "NginxStatus";
            auth_basic_user_file  conf/htpasswd;
        }

        #禁止访问 .htxxx 文件
        location ~ /\.ht {
            deny all;
        }

        #错误处理页面（可选择性配置）
```

```
        #error_page   404              /404.html;
        #error_page   500 502 503 504  /50x.html;
        #location = /50x.html {
        #    root   html;
        #}
    }
}
```

好了，让我们来试试吧：

1. 启动 webapp，注意启动绑定的端口要和nginx中的 `upstream` 设置的端口保持一致。
2. 更改 host：在 C:\Windows\System32\drivers\etc 目录下的host文件中添加一条 DNS 记录

   ```
   127.0.0.1 www.helloworld.com
   ```

3. 启动前文中 startup.bat 的命令
4. 在浏览器中访问 www.helloworld.com，不出意外，已经可以访问了。

如果你在linux中，hosts文件在：etc\hosts

四、负载均衡配置

上一个例子中，代理仅仅指向一个服务器。

但是，网站在实际运营过程中，多半都是有多台服务器运行着同样的app，这时需要使用负载均衡来分流。

nginx也可以实现简单的负载均衡功能。

假设这样一个应用场景：将应用部署在 192.168.1.11:80、192.168.1.12:80、192.168.1.13:80 三台 linux环境的服务器上。网站域名叫 www.helloworld.com，公网IP为 192.168.1.11。在公网IP所在的服务器上部署 nginx，对所有请求做负载均衡处理。

nginx.conf 配置如下：

```
http {
     #设定mime类型,类型由mime.type文件定义
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    #设定日志格式
    access_log    /var/log/nginx/access.log;

    #设定负载均衡的服务器列表
    upstream load_balance_server {
        #weigth参数表示权值，权值越高被分配到的几率越大
        server 192.168.1.11:80   weight=5;
        server 192.168.1.12:80   weight=1;
        server 192.168.1.13:80   weight=6;
    }

   #HTTP服务器
   server {
        #侦听80端口
        listen       80;

        #定义使用www.xx.com访问
        server_name  www.helloworld.com;
```

```
        #对所有请求进行负载均衡请求
        location / {
            root        /root;                  #定义服务器的默认网站根目录位置
            index       index.html index.htm;   #定义首页索引文件的名称
            proxy_pass  http://load_balance_server/ ;#请求转向load_balance_server
定义的服务器列表

        }
    }
}
```

五、网站有多个webapp的配置

当一个网站功能越来越丰富时，往往需要将一些功能相对独立的模块剥离出来，独立维护。这样的话，通常，会有多个 webapp。

举个例子：假如 www.helloworld.com 站点有好几个webapp，finance（金融）、product（产品）、admin（用户中心）。访问这些应用的方式通过上下文(context)来进行区分:

www.helloworld.com/finance/

www.helloworld.com/product/

www.helloworld.com/admin/

我们知道，http的默认端口号是80，如果在一台服务器上同时启动这3个 webapp 应用，都用80端口，肯定是不成的。所以，这三个应用需要分别绑定不同的端口号。

那么，问题来了，用户在实际访问 www.helloworld.com 站点时，访问不同 webapp，总不会还带着对应的端口号去访问吧。所以，你再次需要用到反向代理来做处理。

配置也不难，来看看怎么做吧：

```
http {
    #此处省略一些基本配置
    upstream product_server{
        server www.helloworld.com:8081;
    }
    upstream admin_server{
        server www.helloworld.com:8082;
    }
    upstream finance_server{
        server www.helloworld.com:8083;
    }

    server {
        #此处省略一些基本配置
        #默认指向product的server
        location / {
            proxy_pass http://product_server;
        }
        location /product/{
            proxy_pass http://product_server;
        }
        location /admin/ {
            proxy_pass http://admin_server;
        }
```

```
        location /finance/ {
            proxy_pass http://finance_server;
        }
    }
}
```

六、https反向代理

七、静态站点配置

有时候，我们需要配置静态站点(即 html 文件和一堆静态资源)。

举例来说：如果所有的静态资源都放在了 `/app/dist` 目录下，我们只需要在 `nginx.conf` 中指定首页以及这个站点的 host 即可。

配置如下：

```
worker_processes  1;

events {
    worker_connections  1024;
}

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

    gzip on;
    gzip_types text/plain application/x-javascript text/css application/xml
text/javascript application/javascript image/jpeg image/gif image/png;
    gzip_vary on;

    server {
        listen       80;
        server_name  static.zp.cn;

        location / {
            root /app/dist;
            index index.html;
            #转发任何请求到 index.html
        }
    }
}
```

然后，添加 HOST：

127.0.0.1 static.zp.cn

此时，在本地浏览器访问 static.zp.cn ， 就可以访问静态站点了。

八、跨域解决方案

web 领域开发中，经常采用前后端分离模式。这种模式下，前端和后端分别是独立的 web 应用程序，例如：后端是 Java 程序，前端是 React 或 Vue 应用。

各自独立的 web app 在互相访问时，势必存在跨域问题。解决跨域问题一般有两种思路：

1. **CORS**

在后端服务器设置 HTTP 响应头，把你需要运行访问的域名加入加入 `Access-Control-Allow-Origin` 中。

1. **jsonp**

把后端根据请求，构造json数据，并返回，前端用 jsonp 跨域。

这两种思路，本文不展开讨论。

需要说明的是，nginx 根据第一种思路，也提供了一种解决跨域的解决方案。

举例：www.helloworld.com 网站是由一个前端 app ，一个后端 app 组成的。前端端口号为 9000， 后端端口号为 8080。

前端和后端如果使用 http 进行交互时，请求会被拒绝，因为存在跨域问题。来看看，nginx 是怎么解决的吧：

```
            #    指定允许跨域的方法，*代表所有
            add_header Access-Control-Allow-Methods *;

            #    预检命令的缓存，如果不缓存每次会发送两次请求
            add_header Access-Control-Max-Age 3600;
            #    带cookie请求需要加上这个字段，并设置为true
            add_header Access-Control-Allow-Credentials true;

            #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
            #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支
持*号
            add_header Access-Control-Allow-Origin $http_origin;

            #    表示请求头的字段 动态获取
            add_header Access-Control-Allow-Headers
            $http_access_control_request_headers;

            #    OPTIONS预检命令，预检命令通过时才发送请求
            #    检查请求的类型是不是预检命令
            if ($request_method = OPTIONS){
                return 200;
            }
```

分环境打包

项目根目录文件.env.dev 开发环境

```
NODE_ENV = 'development'
VUE_APP_MODE = 'dev'
VUE_APP_API_URL = 'http://127.0.0.1:3000'

不同环境的.env文件配置，VUE_APP_API_URL为接口网址,可以直接调用
process.env.VUE_APP_API_URL
```

项目根目录文件.env.prod 开发环境

```
NODE_ENV = 'production'
VUE_APP_MODE = 'pord'
VUE_APP_API_URL = 'http://172.17.15.246:3001/api'
```

项目根目录文件.env.test 开发环境

```
NODE_ENV = 'test'
VUE_APP_MODE = 'dev'
VUE_APP_API_URL = 'http://localhost:3000'
```

pageage.json

```
"scripts": {
    "serve": "vue-cli-service serve --open",
    "build": "vue-cli-service build --mode dev",
    "test": "vue-cli-service build --mode test",
    "pord": "vue-cli-service build --mode pord",
    "lint": "vue-cli-service lint"
  },
```