Nginx 从入门到实战

海岛 Java后端 2019-10-27

作者 | 海岛

来源 | sf.gg/a/1190000014893012

上篇 | 10 个让你笑的合不拢嘴的 GitHub 项目

基础篇

一、环境

服务器版本：CentOS 7.2

为了保证**学习阶段**不遇到奇怪的事情，请保证以下四点（大神选择性无视）

1. 确认系统网络
2. 确认yum可用
3. 确认关闭iptables
4. 确认停用selinux

```
#查看iptables状态
systemctl status firewalld.service
#关闭防火墙（临时关闭）
systemctl stop firewalld.service
#查看SELinux状态
getenforce
#临时关闭SELinux
setenforce 0
```

安装一些系统基本工具，正常情况系统都会自带(没有在装哦)

```
yum -y install gcc gcc-c++ autoconf pcre pcre-devel make automake
yum -y install wget httpd-tools vim
```

二、Nginx是什么?

Nginx是一个开源且高性能、可靠的HTTP中间件、代理服务

其他的HTTP服务：

1. HTTPD-Apache基金会
2. IIS-微软
3. GWS-Google(不对外开放)

近几年，Nginx的市场占有率越来越高，一度飙升，为什么呢?接下来我们就知道了！

三、我们为什么选择Nginx?

1. IO多路复用epoll（IO复用）

如何理解呢?举个例子吧！

有A、B、C三个老师，他们都遇到一个难题，要帮助一个班级的学生解决课堂作业。

老师A采用从第一排开始一个学生一个学生轮流解答的方式去回答问题，老师A浪费了很多时间，并且有的学生作业还没有完成呢，老师就来了，反反复复效率极慢。

老师B是一个忍者，他发现老师A的方法行不通，于是他使用了影分身术，分身出好几个自己同一时间去帮好几个同学回答问题，最后还没回答完，老师B消耗光了能量累倒了。

老师C比较精明，他告诉学生，谁完成了作业举手，有举手的同学他才去指导问题，他让学生主动发声，分开了“并发”。

这个老师C就是Nginx。

2. 轻量级

- 功能模块少 - Nginx仅保留了HTTP需要的模块，其他都用插件的方式，后天添加
- 代码模块化 - 更适合二次开发，如阿里巴巴Tengine

3. CPU亲和

把CPU核心和Nginx工作进程绑定，把每个worker进程固定在一个CPU上执行，减少切换CPU的cache miss，从而提高性能。

三、安装与目录

本人使用了鸟哥的lnmp集成包 https://lnmp.org，简单方便-推荐！

```
#执行这句语句，根据指引，将安装 nginx php mysql 可进入lnmp官网查看更详细的过程
#默认安装目录/usr/local
wget -c http://soft.vpser.net/lnmp/lnmp1.4.tar.gz && tar zxf lnmp1.4.tar.gz && cd lnmp1.4 && ./install.sh lnmp

#默认安装目录
/usr/local
```

四、基本配置

```
#打开主配置文件，若你是用lnmp环境安装
vim /usr/local/nginx/conf/nginx.conf

----------------------------------------

user #设置nginx服务的系统使用用户
worker_processes #工作进程数 一般情况与CPU核数保持一致
error_log #nginx的错误日志
pid #nginx启动时的pid

events {
  worker_connections #每个进程允许最大连接数
  use #nginx使用的内核模型
}
```

我们使用 nginx 的 http 服务，在配置文件 nginx.conf 中的 http 区域内，配置无数个 server ，每一个 server 对应这一个虚拟主机或者域名

```
http {
  ... ... #后面再详细介绍 http 配置项目

  server {
    listen 80 #监听端口;
    server_name localhost #地址

    location / { #访问首页路径
      root /xxx/xxx/index.html #默认目录
      index index.html index.htm #默认文件
    }

    error_page 500 504 /50x.html #当出现以上状态码时从新定义到50x.html
    location = /50x.html { #当访问50x.html时
      root /xxx/xxx/html #50x.html 页面所在位置
    }
  }

  server {
    ... ...
  }
}
```

一个 server 可以出现多个 location ，我们对不同的访问路径进行不同情况的配置

我们再来看看 http 的配置详情

```
http {
  sendfile on #高效传输文件的模式 一定要开启
  keepalive_timeout 65 #客户端服务端请求超时时间
  log_format main XXX #定义日志格式 代号为main
  access_log /usr/local/access.log main #日志保存地址 格式代码 main
}
```

五、模块

查看 nginx 已开启和编联进去的模块，模块太多了，就不在这长篇大论，有需要自行百度吧~

```
#大写V查看所有模块，小写v查看版本
nginx -V
# 查看此配置文件 是否存在语法错误
nginx -tc /usr/local/nginx/conf/nginx.conf
```

场景实现篇

一、静态资源WEB服务

1.静态资源类型

非服务器动态运行生成的文件，换句话说，就是可以直接在服务器上找到对应文件的请求

1. 浏览器端渲染：HTML,CSS,JS
2. 图片：JPEG,GIF,PNG
3. 视频：FLV,MPEG
4. 文件：TXT，任意下载文件

2.静态资源服务场景-CDN

什么是CDN?例如一个北京用户要请求一个文件，而文件放在的新疆的资源存储中心，如果直接请求新疆距离太远，延迟久。使用nginx静态资源回源，分发给北京的资源存储中心，让用户请求的动态定位到北京的资源存储中心请求，实现传输延迟的最小化

2.nginx静态资源配置

```
配置域：http、server、location
#文件高速读取
http {
    sendfile on;
}
#在 sendfile 开启的情况下，开启 tcp_nopush 提高网络包传输效率
#tcp_nopush 将文件一次性一起传输给客户端，就好像你有十个包裹，快递员一次送一个，来回十趟，开启后，快递员讲等待你十个包裹都
http {
    sendfile on;
    tcp_nopush on;
}
#tcp_nodelay 开启实时传输，传输方式与 tcp_nopush 相反，追求实时性，但是它只有在长连接下才生效
http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
}

#将访问的文件压缩传输 （减少文件资源大小，提高传输速度）
#当访问内容以gif或jpg结尾的资源时
location ~ .*\.(gif|jpg)$ {
    gzip on; #开启
    gzip_http_version 1.1; #服务器传输版本
    gzip_comp_level 2; #压缩比，越高压缩越多，压缩越高可能会消耗服务器性能
    gzip_types text/plain application/javascript application/x-javascript text/javascript text/css application/xml application/xml+r
    root /opt/app/code; #对应目录（去该目录下寻找对应文件）
}

#直接访问已压缩文件
#当访问路径以download开头时，如www.baidu.com/download/test.img
#去/opt/app/code目录下寻找test.img.gz文件，返回到前端时已是可以浏览的img文件
location ~ load^/download {
    gzip_static on #开启;
    tcp_nopush on;
    root /opt/app/code;
}
```

二、浏览器缓存

HTTP协议定义的缓存机制（如：Expires; Cache-control等 ）

减少服务端的消耗，降低延迟

1.浏览器无缓存

浏览器请求 -> 无缓存 -> 请求WEB服务器 -> 请求相应 -> 呈现

在**呈现**阶段会根据缓存的设置在浏览器中生成缓存

2.浏览器有缓存

浏览器请求 -> 有缓存 -> 校验本地缓存时间是否过期 -> 没有过期 -> 呈现

若过期从新请求WEB服务器

3.语法配置

```
location ~ .*\.(html|htm)$ {
  expires 12h; #缓存12小时
}
```

服务器响应静态文件时，请求头信息会带上 etag 和 last_modified_since 2个标签值，浏览器下次去请求时，头信息发送这两个标签，服务器检测文件有没有发生变化，如无,直接头信息返 etag 和last_modified_since，状态码为 **304** ，浏览器知道内容无改变,于是直接调用本地缓存，这个过程也请求了服务，但是传着的内容极少

三、跨站访问

开发nginx跨站访问设置

```
location ~ .*\.(html|htm)$ {
   add_header Access-Control-Allow-Origin *;
   add_header Access-Control-Allow-Methods GET,POST,PUT,DELETE,OPTIONS;
   #Access-Control-Allow-Credentials true #允许cookie跨域
}
```

在响应中指定 Access-Control-Allow-Credentials 为 true 时，Access-Control-Allow-Origin 不能指定为 *，需要指定到具体域名

相关跨域内容可参考 Laravel 跨域功能中间件 使用代码实现跨域，原理与nginx跨域配置相同

四、防盗链

防止服务器内的静态资源被其他网站所套用

此处介绍的 nginx 防盗链为基础方式，其它更加深入的方式将在之后的文章介绍

首先，需要理解一个nginx变量

```
$http_referer #表示当前请求上一次页面访问的地址，换句话说，访问 www.baidu.com 主页，这是第一次访问，所以 $http_referer 为空
```

然后配置

```
location ~ .*\.(jpg|gif)$ {
   #valid_referers 表示我们允许哪些 $http_referer 来访问
   #none 表示没有带 $http_referer，如第一次访问时 $http_referer 为空
   #blocked 表示 $http_referer 不是标准的地址，非正常域名等
   #只允许此ip
   valid_referers none blocked 127.xxx.xxx.xx
   if ($invalid_referer) { #不满足情况下变量值为1
     return 403;
   }
}
```

五、HTTP代理服务

Nginx可以实现多种代理方式

- HTTP
- ICMPPOPIMAP
- HTTPS
- RTMP

1. 代理区别

区别在于代理的对象不一样

正向代理代理的对象是客户端

反向代理代理的对象是服务端

2. 反向代理

```
语法：proxy_pass URL
默认：——
位置：loaction

#代理端口
#场景：服务器80端口开放，8080端口对外关闭，客户端需要访问到8080
#在nginx中配置proxy_pass代理转发时，如果在proxy_pass后面的url加/，表示绝对根路径；如果没有/，表示相对路径，把匹配的路径部
server {
  listen 80;
  location / {
    proxy_pass http://127.0.0.1:8080/;
    proxy_redirect default;

    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr; #获取客户端真实IP

    proxy_connect_timeout 30; #超时时间
    proxy_send_timeout 60;
    proxy_read_timeout 60;

    proxy_buffer_size 32k;
    proxy_buffering on; #开启缓冲区,减少磁盘io
    proxy_buffers 4 128k;
    proxy_busy_buffers_size 256k;
    proxy_max_temp_file_size 256k; #当超过内存允许储蓄大小，存到文件
  }
}
```

负载均衡和缓存服务

一、负载均衡

负载均衡的实现方法就是我们上章介绍的**反向代理** 。将客户的请求通过 nginx 分发（反向代理）到一组多台不同的服务器上

这一组服务器我们称为 **服务池（upstream server）**，池内的每一个服务器称为一个 **单元**，服务池内将对每一个单元进行请求轮训，实现负载均衡

```
#配置
语法：upstream name ...
默认：——
位置：http

upstream #自定义组名 {
  server x1.baidu.com; #可以是域名
  server x2.baidu.com;
  #server x3.baidu.com
  #down 不参与负载均衡
  #weight=5; 权重，越高分配越多
  #backup; 预留的备份服务器
   #max_fails 允许失败的次数
   #fail_timeout 超过失败次数后，服务暂停时间
    #max_coons 限制最大的接受的连接数
    #根据服务器性能不同，配置适合的参数

  #server 106.xx.xx.xxx; 可以是ip
  #server 106.xx.xx.xxx:8080; 可以带端口号
  #server unix:/tmp/xxx; 支出socket方式
}
```

假设我们有三台服务器，并且假设它们的IP地址，前端负载均衡服务器A（127.0.0.1），后台服务器B（127.0.0.2），后台服务器C（127.0.0.3）

新建文件 proxy.conf，内容如下，上一章介绍的反向代理配置

```
proxy_redirect default;
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_connect_timeout 30;
proxy_send_timeout 60;
proxy_read_timeout 60;
proxy_buffer_size 32k;
proxy_buffering on;
proxy_buffers 4 128k;
proxy_busy_buffers_size 256k;
proxy_max_temp_file_size 256k;

#服务器A的配置
http {
  ...
  upstream xxx {
    server 127.0.0.2;
    server 127.0.0.3;
  }
  server {
    liseten 80;
    server_name localhost;
    location / {
      proxy_pass http://xxx #upstream 对应自定义名称
      include proxy.conf;
    }
  }
}

#服务器B、服务器C的配置
server {
  liseten 80;
  server_name localhost;
  location / {
     index index.html
  }
}
```

调度算法

- 轮训：按时间顺序逐一分配到不同的后端服务器
- 加权轮训：weight值越大，分配到的几率越高
- ip_hash：每个请求按访问IP的hash结果分配，这样来自同一个IP固定访问一个后端服务器
- least_conn：最少链接数，哪个机器连接数少就分发给谁
- url_hash：按照访问的URL的hash结果来分配请求，每一个URL定向到同一个后端服务器
- hash关键数值：hash自定义key

ip_hash 配置

```
upstream xxx {
    ip_hash;
    server 127.0.0.2;
    server 127.0.0.3;
 }
```

ip_hash存在缺陷，当前端服务器再多一层时，将获取不到用户的正确IP，获取的将是前一个前端服务器的IP，因此nginx1.7.2版本推出了 url_hash

url_hash 配置

```
upstream xxx {
    hash $request_uri;
    server 127.0.0.2;
    server 127.0.0.3;
 }
```

二、缓存服务

1. 缓存类型

- 服务端缓存：缓存存储在后端服务器，如redis,memcache
- 代理缓存：缓存存储在代理服务器或者中间件上，它的内容是从后端服务器获取的，但是保存在自己本地
- 客户端缓存：缓存在浏览器内的

2. nginx 代理缓存

客户端请求nginx，nginx查看本地是否有缓存数据，若有直接返回给客户端，若没有再去后端服务器请求

```
http {
  proxy_cache_path /var/www/cache #缓存地址
                levels=1:2 #目录分级
                keys_zone=test_cache:10m #开启的keys空间名字:空间大小(1m可以存放8000个key)
                max_size=10g #目录最大大小(超过时，不常用的将被删除)
                inactive=60m #60分钟内没有被访问的缓存将清理
                use_temp_path=pff; #是否开启存放临时文件目录，关闭默认存储在缓存地址

  server {
    ...
    location / {
      proxy_cache test_cache; #开启缓存对应的名称，在keys_zone命名好
      proxy_cache_valid 200 304 12h; #状态码为200 304的缓存12小时
      proxy_cache_valid any 10m; #其他状态缓存10小时
      proxy_cache_key $host$uri$is_args$args; #设置key值
      add_header Nginx-Cache "$upstream_cache_status";
    }
  }
}
```

当有个特定请求我们不需要缓存的时候，在上面配置的内容中加入以下配置

```
server {
  ...
  if ($request_uri ~ ^/(login|register) ) { #当请求地址有login或register时
    set $nocache = 1; #设置一个自定义变量为true
  }
  location / {
    proxy_no_cache $nocache $arg_nocache $arg_comment;
    proxy_no_cache $http_pragma $http_authoriztion;
  }
}
```

3. 分片请求

早期版本 nginx 对大文件的分片请求不支持缓存，1.9版本后slice模块实现了这个功能

前端发起请求，nginx去获取这个请求文件的大小，若超过我们的定义slice的大小，会进行切片，分割成多个小的请求去请求后端，到前端就成为一个一个独立的缓存文件

tips：欢迎关注微信公众号：Java后端，获取每日技术博文推送。

优势：每个子请求收到的数据都会形成独立文件，一个请求中断了，其他请求不受影响，原本情况请求中断，再次请求文件将

从头开始，而开启分片请求，就接下去获取未请求的小文件

劣势：当文件很大或者slice很小时，可能会导致文件描述符耗尽等情况

```
语法：slice size; #当大文件请求时，设置size为每个小文件的大小
默认：slice 0;
位置：http/server/location
```

常见问题

一、相同 server_name 多个虚拟主机优先级

```
#当出现虚拟主机域名相同的情况，重启nginx时，会出现警告⚠处理，但是并不不会阻止nginx继续使用

server {
  listen 80;
  server_name www.baidu.com
  ...
}

server {
  listen 80;
  server_name www.baidu.com
  ...
}

...

优先选择最新读取到的配置文件，当多个文件是通过include时，文件排序越靠前，越早被读取
```

二、location 匹配优先级

```
= #进行普通字符精确匹配，完全匹配
^~ #进行普通字符匹配，当前表示前缀匹配
~\~* #表示执行一个正则匹配()

#当程序使用精确匹配时，一但匹配成功，将停止其他匹配
#当正则匹配成功时，会继续接下来的匹配，寻找是否还有更精准的匹配
```

三、try_files的使用

按顺序检查文件是否存在

```
location / {
  try_files $uri $uri/ /index.php;
}

#先查找$uri下是否有文件存在，若存在直接返回给用户
#若$url下没有文件存在，再次访问$uri/的路径是否有文件存在
#还是没有文件存在，交给index.php处理

例:
location / {
  root /test/index.html
  try_files $uri @test
}

location @test {
  proxy_pass http://127.0.0.1：9090;
}

#访问 / 时，查看 /test/index.html 文件是否存在
#若不存在，让9090端口的程序去处理这个请求
```

四、alias和root的区别

```
location /request_path/image/ {
  root /local_path/image/;
}

#当我们访问 http://xxx.com/request_path/image/cat.png时
#将访问 http://xxx.com/request_path/image/local_path/image/cat.png 下的文件

location /request_path/image/ {
  alias /local_path/image/;
}

#当我们访问 http://xxx.com/request_path/image/cat.png时
#将访问 http://xxx.com/local_path/image/cat.png 下的文件
```

五、如果用户真实IP

当一个请求通过多个代理服务器时，用户的IP将会被代理服务器IP覆盖

```
#在第一个代理服务器中设置
  set x_real_ip=$remote_addr
#最后一个代理服务器中获取
  $x_real_ip=IP1
```

六、Nginx 常见错误码

```
413 Request Entity Too Large #上传文件过大，设置 client_max_body_size
503 bad gateway #后端服务无响应
504 Gateway Time-out        #后端服务执行超时
```

-END-

推荐阅读

1. 10 个让你笑的合不拢嘴的 GitHub 项目
2. 当我遵循了这 16 条规范写代码
3. 理解 IntelliJ IDEA 的项目配置和 Web 部署
4. Java 开发中常用的 4 种加密方法
5. 团队开发中 Git 最佳实践

学Java，请关注公众号：Java后端

喜欢文章，点个在看

阅读原文

Nginx 反向代理、负载均衡图文教程！

chenhongdong Java后端 2019-12-10

作者 | chenhongdong

juejin.im/post/5b01336af265da0b8a67e5c9

学到老活到老

前端圈一直很新，一直要不停的学习，而且在进入大厂的路上，还要求熟悉一门后台语言等等。用一句别人开玩笑的话来说，java十年前的技术现在还能用，而前端的技术就不是这样的了

突然想起了deno项目发布的时候，一个搞笑的issue，“求别更新了，老子学不动了”。虽然看起来是一个玩笑的issue，但却道出了前端们不得不表现出来的疲态，知识点越来越庞大，学习的内容越来越多

也听到一些朋友们说，换成现在再面试阿里，恐怕不好进了啊。当然很多都是随便一说的玩笑话，听过一笑便可，不必当真，也不必抱怨了

好了，今天就直接来说一下主题吧，前端要了解一些运维的Nginx用法，内容不多，简单看看就好，这两个功能在工作当中就够用了，那么首先来看个问题，什么是反向代理与负载均衡

什么是反向代理与负载均衡

什么是反向代理

当我们有一个服务器集群，并且服务器集群中的每台服务器的内容一样的时候，同样我们要直接从个人电脑访问到服务器集群服务器的时候无法访问，必须通过第三方服务器才能访问集群

这个时候，我们通过第三方服务器访问服务器集群的内容，但是我们并不知道是哪一台服务器提供的内容，此种代理方式称为反向代理

什么是负载均衡

公司会建立很多的服务器，这些服务器组成了服务器集群，然后，当用户访问网站的时候，先访问一个中间服务器，再让这个中间服务器在服务器集群中选择一个压力较小的服务器，然后将该访问请求引入选择的服务器

所以，用户每次访问，都会保证服务器集群中的每个服务器压力趋于平衡，分担了服务器压力，避免了服务器崩溃的情况

一句话：nginx会给你分配服务器压力小的去访问。可以关注这个公众号查看Nginx的基础教程，点击跳转 一文读懂并实操 Nginx 反向代理

Nginx反向代理与负载均衡的实现

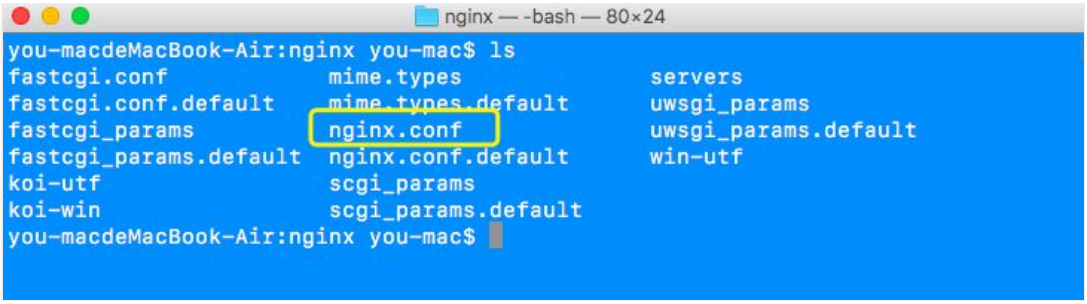
用户访问网站的时候首先会访问nginx服务器，然后nginx服务器再从服务器集群中选择压力较小的服务器，将该访问请求引向该服务器

nginx配置

下面修改配置方面我就从mac系统下来进行简单的演示，如何安装的话也暂以mac为主了，windows系统直接去Nginx官网下载安装即可

```
安装nginx
  1-进到homebrew官网，然后复制命令，预安装需要的东西
  2-brew install nginx 安装nginx
  3-nginx -v 显示版本号
进入nginx
cd /usr/local/etc/nginx
```

下图为进入nginx文件夹下的文件内容

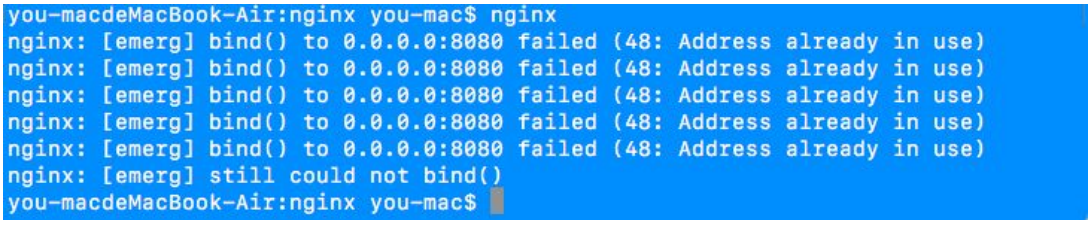


当进到这个目录下，我们就可以操作nginx了，接下来就列举一些非常非常有用的命令，多敲几遍，一定要记住

nginx常用命令

- 启动nginx
- nginx
- 当你敲完nginx这5个键的时候，并没有任何反应，此时你只需访问localhost:8080(默认)即可

- 关闭nginx
- 如果出现下图情况，不要惊慌，是因为之前nginx被启动过了
- 只需**nginx -s stop**，停止nginx服务
- 然后再次启动nginx即可



- 重启nginx
- nginx -s reload
- 每次修改完.conf文件就需要重启nginx
- 检查配置
- 检查修改的nginx.conf配置是否正确
- **nginx -t**
- 如果出现下面ok和successfull就代表正确了，其他的都不对

nginx:the configuration file/usr/local/etc/nginx/nginx.conf syntaxisok

nginx:configuration file/usr/local/etc/nginx/nginx.conf testissuccessful

对于我们前端来说正常工作当中，倒是不需要过多的修改nginx的。我们之所以修改nginx配置，是为了做一些反向代理罢了

proxy_pass

nginx反向代理主要通过proxy_pass来配置，将你项目的开发机地址填写到proxy_pass后面，正常的格式为proxy_pass URL即可

```
server {
  listen 80;
  location / {
    proxy_pass http://10.10.10.10:20186;
  }
}
```

Upstream模块实现负载均衡

- ip_hash指令
- server指令
- upstream指令及相关变量

上面写的三个指令，我们直接通过代码来一一分析

```
// 修改nginx.conf
worker_processes 1;
events {
worker_connections 1024;
}
http {
upstream firstdemo {
server 39.106.145.33;
server 47.93.6.93;
}
server {
listen 8080;
location / {
proxy_pass http://firstdemo;
}
}
}
```

上面修改的nginx.conf就是上图中花圈的那个文件，nginx配置的主要修改就在这里。化繁为简，把原本nginx.conf里的内容直接替换为上面的不到20行的代码了

既然不到20行，那就把里面对应的内容统统解释一下吧，有个了解就好

- worker_processes
- worker_connections
- upstream模块
- 负载均衡就靠它
- 语法格式：upstream name {}
- 里面写的两个server分别对应着不同的服务器

- server模块
- 实现反向代理
- listen监督端口号
- location / {}访问根路径
- proxy_pass http://firstdemo，代理到firstdemo里两个服务器上

上面修改了nginx.conf之后，别忘了最重要的一步重启nginx

那么再次访问localhost:8080，会看到如下图页面

还有另一个页面

每次刷新都会访问不同的服务器，这样就做到了**负载均衡**处理

不过，更应该做到的是当用户第一次访问到其中一台服务器后，下次再访问的时候就直接访问该台服务器就好了，不用总变化了。那么就发挥了ip_hash的威力了

```
// 省略...
upstream firstdemo {
ip_hash;
server 39.106.145.33;
server 47.93.6.93;
}
```

ip_hash它的作用是如果第一次访问该服务器后就记录，之后再访问都是该服务器了，这样比如第一次访问是33服务器，那

之后再访问也会分配为33服务器访问了

工作中的简单使用

在公司开发项目的时候，遇到设计，产品走查环节的时候，不能每次都让他们去配一个host，毕竟这样不友好，走查起来有麻烦。所以更应该给他们直观的感受，既给一个访问地址就可以看到样子

下面给大家看一下，我正常在公司时nginx做的反向代理配置，和咱们上面的如出一辙，只是加了一个server_name，用指定的域名去访问即可

```
server {
  listen 80;
  server_name  chd.news.so.m.qss.test.so.com ;
  auth_basic off;
  location / {
    proxy_pass   http://10.10.10.10:20186;
    proxy_set_header Host $host;
    proxy_redirect off;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_connect_timeout 60;
    proxy_read_timeout 600;
    proxy_send_timeout 600;
  }
}
```

每次修改完nginx配置后不要忘记重启nginx才能生效，这样只需要访问chd.news.so.m.qss.test.so.com这个地址就可以查看我的开发环境，进行走查了。

这就是nginx最大的功能，反向代理我也接触的不是很多，毕竟不是专业运维出身，可比性差了很多。略知一二，也只是方便大家工作中使用吧，再次感谢大家的收看了，哈哈

作者：chenhongdong

https://juejin.im/post/5b01336af265da0b8a67e5c9

【END】

推荐阅读

1. 面试字节跳动，我被怼了。
2. Java 并发编程 73 道面试题及答案
3. 说实话：前后端分离后，比从前更痛苦了
4. 一个女生不主动联系你还有机会吗？
5. 团队开发中 Git 最佳实践

阅读原文

Nginx 搭建图片服务器

ITDragon龙 Java后端 2019-11-15

作者 | ITDragon龙

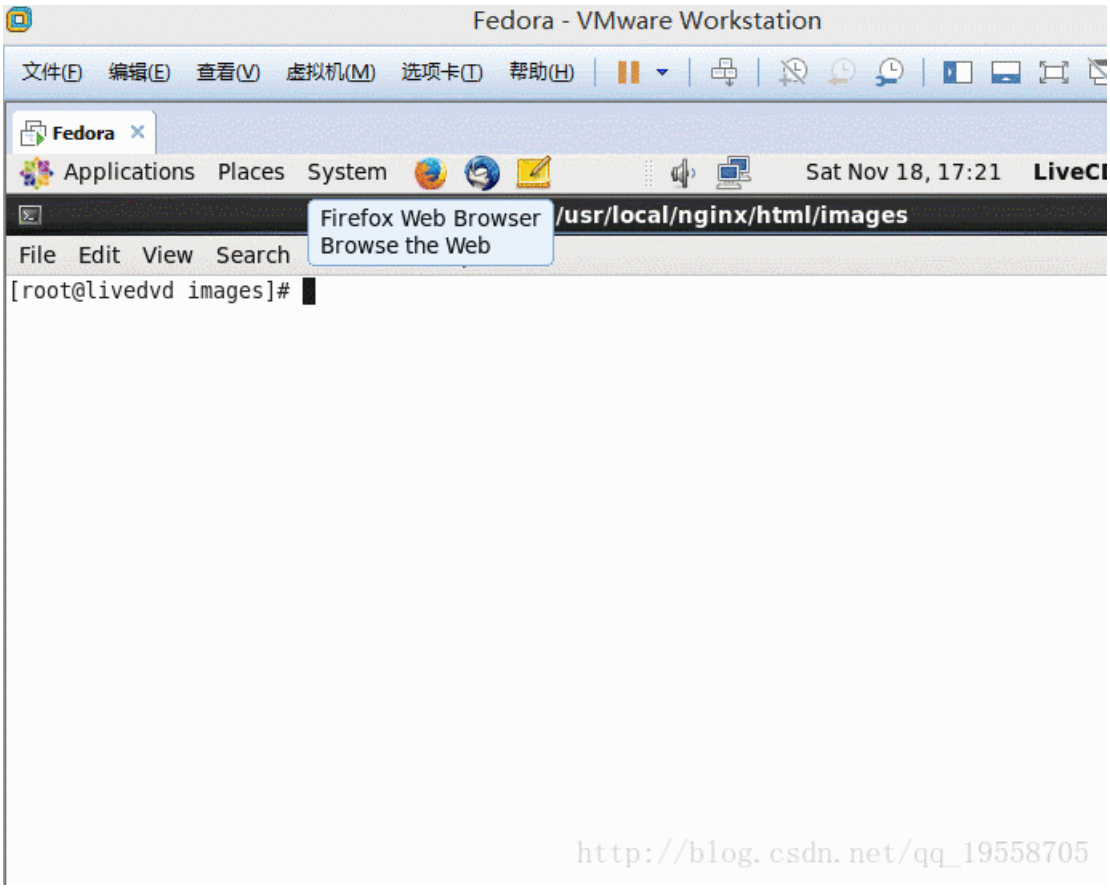
链接 | cnblogs.com/itdragon/p/7864916.html

本章内容通过Nginx 和 FTP 搭建图片服务器。在学习本章内容前，请确保您的Linux 系统已经安装了Nginx和Vsftpd。

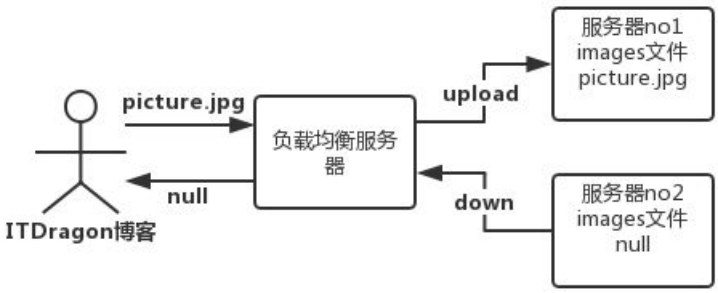
> Nginx 安装
> http://www.cnblogs.com/itdragon/p/7850985.html
>
> Vsftpd 安装
> http://www.cnblogs.com/itdragon/p/7857649.html

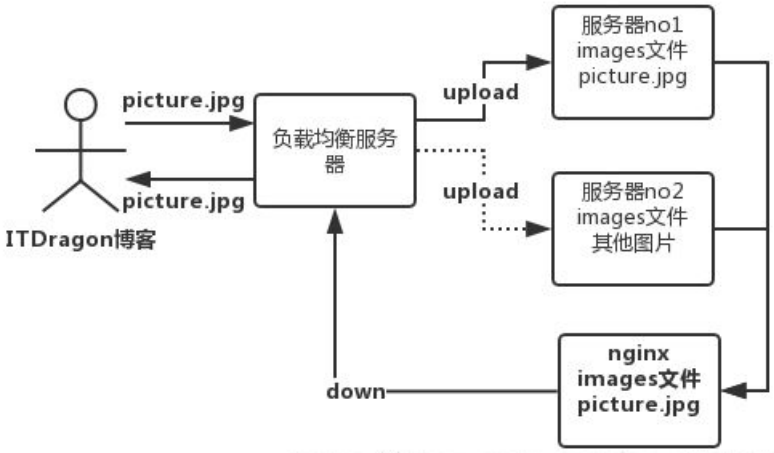
本章知识点

效果图：

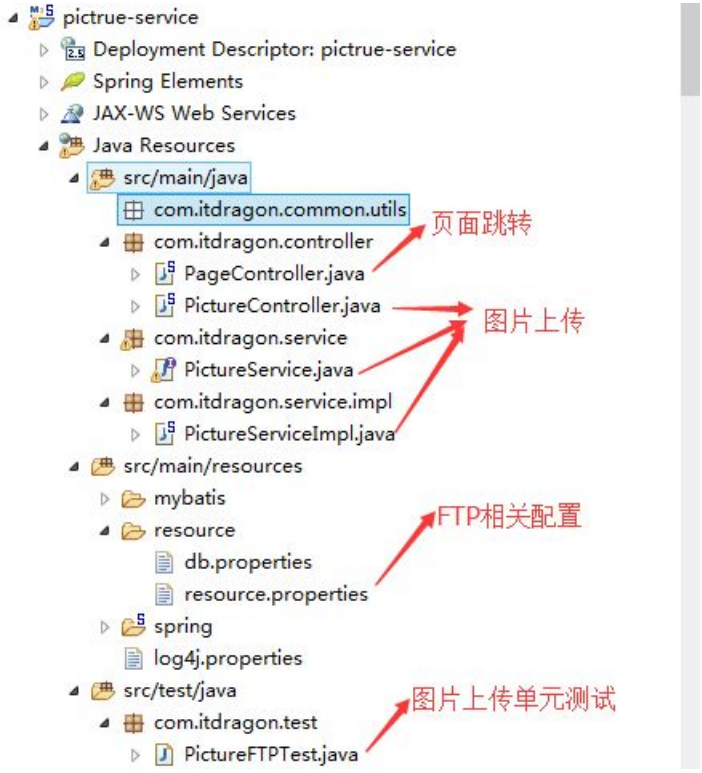


- 需求：实现图片的上传和批量上传
- 技术：Nginx，Vsftpd，Spring，SpringMVC，KindEditor，CentOS
- 说明：本章节内容主要是实现图片的上传功能。使用 KindEditer 是为了更好的演示图片的上传，回显，批量效果。后台代码与KindEditer没有直接关系，放心阅读。另外源码中有Mybatis的jar，不用理会，本章内容用不到，是为后续内容做准备！
- 源码：见文章底部
- 场景：用户将图片上传到 tomcat 服务器上，再由 tomcat 服务器通过FTP上传到 Nginx 服务器上。这有篇 Nginx 教程，欢迎关注这个订阅号：Nginx 从入门到实战





项目结构：



单元测试

首先要攻破核心技术。通过单元测试实现图片上传的功能。

```
public class PictureFTPTest {

  //测试 ftp 上传图片功能
  @Test
  public void testFtpClient() throws Exception {
    //1. 创建一个FtpClient对象
    FTPClient ftpClient = new FTPClient();
    //2. 创建 ftp 连接
    ftpClient.connect("192.168.0.11", 21);
    //3. 登录 ftp 服务器
    ftpClient.login("ftpuser", "root");
    //4. 读取本地文件
    FileInputStream inputStream = new FileInputStream(new File("F:\\hello.png"));
    //5. 设置上传的路径
    ftpClient.changeWorkingDirectory("/usr/local/nginx/html/images");
    //6. 修改上传文件的格式为二进制
    ftpClient.setFileType(FTP.BINARY_FILE_TYPE);
    //7. 服务器存储文件，第一个参数是存储在服务器的文件名，第二个参数是文件流
    ftpClient.storeFile("hello.jpg", inputStream);
    //8. 关闭连接
    ftpClient.logout();

  }

}
```

说明：这里的ip地址，端口，ftp用户名，密码，本地文件路径，以及Nginx服务器图片路径等，这些字符串参数都要根据自己实际设置的来填写的。如果你的Nginx和Vsftpd安装是按照我提供的链接来做的。那你只需要改ip地址即可。

Maven 的Web 项目

搭建Maven的Web 项目，之前有写过。这里就不过多描述。

项目核心配置文件

首先是 Maven 的核心文件 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.itdragon.upload</groupId>
  <artifactId>pictrue-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <!-- 集中定义依赖版本号 -->
  <properties>
    <junit.version>4.12</junit.version>
    <spring.version>4.1.3.RELEASE</spring.version>
    <mybatis.version>3.2.8</mybatis.version>
    <mybatis.spring.version>1.2.2</mybatis.spring.version>
    <mybatis.paginator.version>1.2.15</mybatis.paginator.version>
    <mysql.version>5.1.6</mysql.version>
    <slf4j.version>1.6.4</slf4j.version>
    <jackson.version>2.4.2</jackson.version>
    <druid.version>1.0.9</druid.version>
    <httpclient.version>4.3.5</httpclient.version>
    <jstl.version>1.2</jstl.version>
    <servlet-api.version>2.5</servlet-api.version>
    <jsp-api.version>2.0</jsp-api.version>
    <joda-time.version>2.5</joda-time.version>
```

```
    <commons-lang3.version>3.3.2</commons-lang3.version>
    <commons-io.version>1.3.2</commons-io.version>
    <commons-net.version>3.3</commons-net.version>
    <pagehelper.version>3.4.2</pagehelper.version>
    <jsqlparser.version>0.9.1</jsqlparser.version>
    <commons-fileupload.version>1.3.1</commons-fileupload.version>
    <jedis.version>2.7.2</jedis.version>
    <solrj.version>4.10.3</solrj.version>
  </properties>
  <dependencies>
    <!-- 时间操作组件 -->
    <dependency>
      <groupId>joda-time</groupId>
      <artifactId>joda-time</artifactId>
      <version>${joda-time.version}</version>
    </dependency>
    <!-- Apache工具组件 -->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>${commons-lang3.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-io</artifactId>
      <version>${commons-io.version}</version>
    </dependency>
    <dependency>
      <groupId>commons-net</groupId>
      <artifactId>commons-net</artifactId>
      <version>${commons-net.version}</version>
    </dependency>
    <!-- Jackson Json处理工具包 -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>${jackson.version}</version>
    </dependency>
    <!-- httpclient -->
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>${httpclient.version}</version>
    </dependency>
    <!-- 单元测试 -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
    <!-- 日志处理 -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>${slf4j.version}</version>
    </dependency>
    <!-- Mybatis -->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>${mybatis.version}</version>
    </dependency>
    <dependency>
```

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>${mybatis.spring.version}</version>
</dependency>
<dependency>
  <groupId>com.github.miemiedev</groupId>
  <artifactId>mybatis-paginator</artifactId>
  <version>${mybatis.paginator.version}</version>
</dependency>
<dependency>
  <groupId>com.github.pagehelper</groupId>
  <artifactId>pagehelper</artifactId>
  <version>${pagehelper.version}</version>
</dependency>
<!-- MySql -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>${mysql.version}</version>
</dependency>
<!-- 连接池 -->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>${druid.version}</version>
</dependency>
<!-- Spring -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>${spring.version}</version>
</dependency>
<!-- JSP相关 -->
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>${jstl.version}</version>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>${servlet-api.version}</version>
  <scope>provided</scope>
```

```xml
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jsp-api</artifactId>
      <version>${jsp-api.version}</version>
      <scope>provided</scope>
    </dependency>
    <!-- 文件上传组件 -->
    <dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>${commons-fileupload.version}</version>
    </dependency>
    <!-- Redis客户端 -->
    <dependency>
      <groupId>redis.clients</groupId>
      <artifactId>jedis</artifactId>
      <version>${jedis.version}</version>
    </dependency>
    <!-- solr客户端 -->
    <dependency>
      <groupId>org.apache.solr</groupId>
      <artifactId>solr-solrj</artifactId>
      <version>${solrj.version}</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <!-- 资源文件拷贝插件 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-resources-plugin</artifactId>
        <version>2.7</version>
        <configuration>
          <encoding>UTF-8</encoding>
        </configuration>
      </plugin>
      <!-- java编译插件 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.2</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
          <encoding>UTF-8</encoding>
        </configuration>
      </plugin>
    </plugins>
    <pluginManagement>
      <plugins>
        <!-- 配置Tomcat插件 -->
        <plugin>
          <groupId>org.apache.tomcat.maven</groupId>
          <artifactId>tomcat7-maven-plugin</artifactId>
          <version>2.2</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

说明：和文件上传有直接关系的是：

```
<dependency>
   <groupId>commons-fileupload</groupId>
   <artifactId>commons-fileupload</artifactId>
</dependency>
```

然后是 Web 项目的核心文件 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="taotao" version="2.5">
  <display-name>pictrue-service</display-name>
  <!-- 加载spring容器 -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring/applicationContext-*.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <!-- 解决post乱码 -->
  <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>utf-8</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!-- springmvc的前端控制器 -->
  <servlet>
    <servlet-name>pictrue-service</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring/springmvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>pictrue-service</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

再是 SpringMVC 配置文件 springmvc.xml，需要添加文件上传解析器

```
<!-- 定义文件上传解析器 -->
  <bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!-- 设定默认编码 -->
    <property name="defaultEncoding" value="UTF-8"></property>
    <!-- 设定文件上传的最大值5MB，5*1024*1024 -->
    <property name="maxUploadSize" value="5242880"></property>
  </bean>
```

最后是 Ftp 配置文件 resource.properties

```
FTP_ADDRESS=192.168.0.11
FTP_PORT=21
FTP_USERNAME=ftpuser
FTP_PASSWORD=root
FTP_BASE_PATH=/usr/local/nginx/html/images
IMAGE_BASE_URL=http://192.168.0.11/images
```

Service 层

上传图片的接口 PictureService.java

```
/**
   * 上传，批量上传接口
   * @param uploadFile
   * @return
   */
  Map uploadPicture(MultipartFile uploadFile);
}
```

上传图片接口实现类 PictureServiceImpl.java

```
@Service
@SuppressWarnings({"rawtypes", "unchecked"})
public class PictureServiceImpl implements PictureService {

  // 通过 Spring4 的 Value注解，获取配置文件中的属性值
  @Value("${FTP_ADDRESS}")
  private String FTP_ADDRESS; // ftp 服务器ip地址
  @Value("${FTP_PORT}")
  private Integer FTP_PORT; // ftp 服务器port，默认是21
  @Value("${FTP_USERNAME}")
  private String FTP_USERNAME; // ftp 服务器用户名
  @Value("${FTP_PASSWORD}")
  private String FTP_PASSWORD; // ftp 服务器密码
  @Value("${FTP_BASE_PATH}")
  private String FTP_BASE_PATH; // ftp 服务器存储图片的绝对路径
  @Value("${IMAGE_BASE_URL}")
  private String IMAGE_BASE_URL; // ftp 服务器外网访问图片路径

  @Override
  public Map uploadPicture(MultipartFile uploadFile) {
    Map resultMap = new HashMap<>();
    try {
      // 1. 取原始文件名
      String oldName = uploadFile.getOriginalFilename();

      // 2. ftp 服务器的文件名
      String newName = oldName;
      //图片上传
      boolean result = uploadFile(FTP_ADDRESS, FTP_PORT, FTP_USERNAME, FTP_PASSWORD,
```

```
            uploadFile.getInputStream(), FTP_BASE_PATH, newName);
        //返回结果
        if(!result) {
          resultMap.put("error", 1);
          resultMap.put("message", "upload Fail");
          return resultMap;
        }
        resultMap.put("error", 0);
        resultMap.put("url", IMAGE_BASE_URL + "/" + newName);
        return resultMap;

      } catch (Exception e) {
        e.printStackTrace();
        resultMap.put("error", 1);
        resultMap.put("message", "upload Fail");
        return resultMap;
      }
    }

    /**
     * ftp 上传图片方法
     * @param ip ftp 服务器ip地址
     * @param port ftp 服务器port，默认是21
     * @param account ftp 服务器用户名
     * @param passwd ftp 服务器密码
     * @param inputStream 文件流
     * @param workingDir ftp 服务器存储图片的绝对路径
     * @param fileName 上传到ftp 服务器文件名
     * @throws Exception
     *
     */
    public boolean uploadFile(String ip, Integer port, String account, String passwd,
        InputStream inputStream, String workingDir, String fileName) throws Exception{
      boolean result = false;
      // 1. 创建一个FtpClient对象
      FTPClient ftpClient = new FTPClient();
      try {
        // 2. 创建 ftp 连接
        ftpClient.connect(ip, port);
        // 3. 登录 ftp 服务器
        ftpClient.login(account, passwd);
        int reply = ftpClient.getReplyCode(); // 获取连接ftp 状态返回值
        System.out.println("code : " + reply);
        if (!FTPReply.isPositiveCompletion(reply)) {
          ftpClient.disconnect(); // 如果返回状态不再 200 ~ 300 则认为连接失败
          return result;
        }
        // 4. 读取本地文件
// FileInputStream inputStream = new FileInputStream(new File("F:||hello.png"));
        // 5. 设置上传的路径
        ftpClient.changeWorkingDirectory(workingDir);
        // 6. 修改上传文件的格式为二进制
        ftpClient.setFileType(FTP.BINARY_FILE_TYPE);
        // 7. 服务器存储文件，第一个参数是存储在服务器的文件名，第二个参数是文件流
        if (!ftpClient.storeFile(fileName, inputStream)) {
          return result;
        }
        // 8. 关闭连接
        inputStream.close();
        ftpClient.logout();
        result = true;
      } catch (Exception e) {
        e.printStackTrace();
      }finally {
```

```
      // FIXME 听说，项目里面最好少用try catch 捕获异常，这样会导致Spring的事务回滚出问题？？？难道之前写的代码都是假代码！！
      if (ftpClient.isConnected()) {
        try {
          ftpClient.disconnect();
        } catch (IOException ioe) {
        }
      }
    }
    return result;
  }

}
```

说明：

- @Value 注解是Spring4 中提供的，@Value("${XXX}")
- 返回值是一个Map，并且key有error，url，message。这是根据KindEditer的语法要求来的。详情见链接。

  http://kindeditor.net/docs/upload.html

Controller 层

负责页面跳转的 PageController.java

```
@Controller
public class PageController {

  /**
   * 打开首页
   */
  @RequestMapping("/")
  public String showIndex() {
    return "index";
  }

  @RequestMapping("/{page}")
  public String showpage(@PathVariable String page) {
    System.out.println("page : " + page);
    return page;
  }
}
```

负责图片上传的 PictureController.java

```java
@RestController
public class PictureController {

  @Autowired
  private PictureService pictureService;

  @RequestMapping("pic/upload")
  public String pictureUpload(@RequestParam(value = "fileUpload") MultipartFile uploadFile) {
    String json = "";
    try {
      Map result = pictureService.uploadPicture(uploadFile);
      // 浏览器擅长处理json格式的字符串，为了减少因为浏览器内核不同导致的bug，建议用json
      json = new ObjectMapper().writeValueAsString(result);
    } catch (JsonProcessingException e) {
      e.printStackTrace();
    }
    return json;
  }
}
```

说明：

- @RestController 也是Spring4 提供的，是 @Controller + @ResponseBody 的组合注解。
- Controller层的返回值是一个json格式的字符串。是考虑到浏览器对json解析兼容性比较好。

Views视图层

负责上传图片的jsp页面 pic-upload.jsp

```
<%@ page language="java" contentType="text/html; UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>ITDragon 图片上传</title>
</head>
  <link href="/js/kindeditor-4.1.10/themes/default/default.css" type="text/css" rel="stylesheet">
  <script type="text/javascript" src="js/jquery.min.js"></script>
  <script type="text/javascript" charset="utf-8" src="/js/kindeditor-4.1.10/kindeditor-all-min.js"></script>
  <script type="text/javascript" charset="utf-8" src="/js/kindeditor-4.1.10/lang/zh_CN.js"></script>
</head>
<body>
  <h3>测试上传图片功能接口的form表单</h3>
  <form action="pic/upload" method="post" enctype="multipart/form-data">
    <input type="file" name="fileUpload" />
    <input type="submit" value="上传文件" />
  </form>
  <hr />
  <h3>借用KindEditor富文本编辑器实现批量上传图片</h3>
  <textarea id="kindEditorDesc" style="width:800px;height:300px;visibility:hidden;"></textarea>
  <script type="text/javascript">
    $(function(){
      //初始化富文本编辑器
      KindEditor.create("#kindEditorDesc", {
        // name值，必须和Controller 的参数对应，不然会提示 400 的错误
        filePostName : "fileUpload",
        // action值，
        uploadJson : '/pic/upload',
        // 设置上传类型，分别为image、flash、media、file
        dir : "image"
      });
    });
  </script>
</body>
</html>
```

说明：pic-upload.jsp 分为两个部分，第一个部分是为了测试上传图片功能的form表单。第二个部分是为了更好的体验上传，批量上传，回显功能的KindEditer 富文本编辑器。

总结

- Nginx 搭建服务器的思维
- Java实现 Ftp上传图片的功能
- KindEditer 上传图片的功能

> 源码：https://github.com/ITDragonBlog/daydayup/tree/master/Nginx

Nginx 搭建图片服务器到这里就结束了，有什么不足的地方，请赐教。如果觉得不错，可以点个赞哦！

Nginx 配置参数中文说明

Ably Java后端 2019-12-04

作者 | Ably

来源 | segmentfault.com/a/1190000005789137

Nginx配置参数中文详细说明：

```
#定义Nginx运行的用户和用户组
user www www;
#
#nginx进程数,建议设置为等于CPU总核心数.
worker_processes 8;
#
#全局错误日志定义类型,[ debug | info | notice | warn | error | crit ]
error_log /var/log/nginx/error.log info;
#
#进程文件
pid /var/run/nginx.pid;
#
#一个nginx进程打开的最多文件描述符数目,理论值应该是最多打开文件数（系统的值ulimit -n）与nginx进程数相除,但是nginx分配请求并不均匀,所
worker_rlimit_nofile 65535;
#
#工作模式与连接数上限
events
{
  #参考事件模型,use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型是Linux 2.6以上版本内核中的高性能网络I/O模型,如果跑在FreeB
  use epoll;
  #单个进程最大连接数（最大连接数=连接数*进程数）
  worker_connections 65535;
}
#
#设定http服务器
http
{
  include mime.types; #文件扩展名与文件类型映射表
  default_type application/octet-stream; #默认文件类型
  #charset utf-8; #默认编码
  server_names_hash_bucket_size 128; #服务器名字的hash表大小
  client_header_buffer_size 32k; #上传文件大小限制
  large_client_header_buffers 4 64k; #设定请求缓
  client_max_body_size 8m; #设定请求缓

  # 开启目录列表访问,合适下载服务器,默认关闭.
  autoindex on; # 显示目录
  autoindex_exact_size on; # 显示文件大小 默认为on,显示出文件的确切大小,单位是bytes 改为off后,显示出文件的大概大小,单位是kB或者MB或者
  autoindex_localtime on; # 显示文件时间 默认为off,显示的文件时间为GMT时间 改为on后,显示的文件时间为文件的服务器时间

  sendfile on; # 开启高效文件传输模式,sendfile指令指定nginx是否调用sendfile函数来输出文件,对于普通应用设为 on,如果用来进行下载等应用磁
  tcp_nopush on; # 防止网络阻塞
  tcp_nodelay on; # 防止网络阻塞

  keepalive_timeout 120; # (单位s)设置客户端连接保持活动的超时时间,在超过这个时间后服务器会关闭该链接

  # FastCGI相关参数是为了改善网站的性能：减少资源占用,提高访问速度.下面参数看字面意思都能理解.
  fastcgi_connect_timeout 300;
```

```
fastcgi_send_timeout 300;
fastcgi_read_timeout 300;
fastcgi_buffer_size 64k;
fastcgi_buffers 4 64k;
fastcgi_busy_buffers_size 128k;
fastcgi_temp_file_write_size 128k;

# gzip模块设置
gzip on; #开启gzip压缩输出
gzip_min_length 1k; #允许压缩的页面的最小字节数,页面字节数从header偷得content-length中获取.默认是0,不管页面多大都进行压缩.建议设置
gzip_buffers 4 16k; #表示申请4个单位为16k的内存作为压缩结果流缓存,默认值是申请与原始数据大小相同的内存空间来存储gzip压缩结果
gzip_http_version 1.1; #压缩版本（默认1.1,目前大部分浏览器已经支持gzip解压.前端如果是squid2.5请使用1.0）
gzip_comp_level 2; #压缩等级.1压缩比最小,处理速度快.9压缩比最大,比较消耗cpu资源,处理速度最慢,但是因为压缩比最大,所以包最小,传输速度快
gzip_types text/plain application/x-javascript text/css application/xml;
#压缩类型,默认就已经包含text/html,所以下面就不用再写了,写上去也不会有问题,但是会有一个warn.
gzip_vary on;#选项可以让前端的缓存服务器缓存经过gzip压缩的页面.例如:用squid缓存经过nginx压缩的数据

#开启限制IP连接数的时候需要使用
#limit_zone crawler $binary_remote_addr 10m;

##upstream的负载均衡,四种调度算法(下例主讲)##

#虚拟主机的配置
server
{
  # 监听端口
  listen 80;
  # 域名可以有多个,用空格隔开
  server_name ably.com;
  # HTTP 自动跳转 HTTPS
  rewrite ^(.*) https://$server_name$1 permanent;
}

server
{
  # 监听端口 HTTPS
  listen 443 ssl;
  server_name ably.com;

  # 配置域名证书
  ssl_certificate     C:\WebServer\Certs\certificate.crt;
  ssl_certificate_key  C:\WebServer\Certs\private.key;
  ssl_session_cache    shared:SSL:1m;
  ssl_session_timeout  5m;
  ssl_protocols SSLv2 SSLv3 TLSv1;
  ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
  ssl_prefer_server_ciphers  on;

  index index.html index.htm index.php;
  root /data/www/;
  location ~ .*\.(php|php5)?$
  {
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include fastcgi.conf;
  }

  # 配置地址拦截转发，解决跨域验证问题
  location /oauth/{
    proxy_pass https://localhost:13580/oauth/;
    proxy_set_header HOST $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }
```

```
    # 图片缓存时间设置
    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$ {
      expires 10d;
    }

    # JS和CSS缓存时间设置
    location ~ .*\.(js|css)?$ {
      expires 1h;
    }

    # 日志格式设定
    log_format access '$remote_addr - $remote_user [$time_local] "$request" '
    '$status $body_bytes_sent "$http_referer" '
    '"$http_user_agent" $http_x_forwarded_for';
    # 定义本虚拟主机的访问日志
    access_log /var/log/nginx/access.log access;

    # 设定查看Nginx状态的地址.StubStatus模块能够获取Nginx自上次启动以来的工作状态，此模块非核心模块，需要在Nginx编译安装时手工指定.
    location /NginxStatus {
      stub_status on;
      access_log on;
      auth_basic "NginxStatus";
      auth_basic_user_file conf/htpasswd;
      #htpasswd文件的内容可以用apache提供的htpasswd工具来产生.
    }
  }
}
```

Nginx多台服务器实现负载均衡：

1.Nginx负载均衡服务器：

IP：192.168.0.4（Nginx-Server）

2.Web服务器列表：

Web1:192.168.0.5（Nginx-Node1/Nginx-Web1）；Web2:192.168.0.7（Nginx-Node2/Nginx-Web2）

3.实现目的：用户访问Nginx-Server（“http://mongo.demo.com:8888”）时，通过Nginx负载均衡到Web1和Web2服务器

Nginx负载均衡服务器的`nginx.conf`配置注释如下：

```
events
{
  use epoll;
  worker_connections 65535;
}
http
{
  ##upstream的负载均衡,四种调度算法##
  #调度算法1:轮询.每个请求按时间顺序逐一分配到不同的后端服务器,如果后端某台服务器宕机,故障系统被自动剔除,使用户访问不受影响
  upstream webhost {
    server 192.168.0.5:6666 ;
    server 192.168.0.7:6666 ;
  }
  #调度算法2:weight(权重).可以根据机器配置定义权重.权重越高被分配到的几率越大
  upstream webhost {
    server 192.168.0.5:6666 weight=2;
    server 192.168.0.7:6666 weight=3;
  }
  #调度算法3:ip_hash. 每个请求按访问IP的hash结果分配,这样来自同一个IP的访客固定访问一个后端服务器,有效解决了动态网页存在的session共享
  upstream webhost {
    ip_hash;
    server 192.168.0.5:6666 ;
    server 192.168.0.7:6666 ;
  }
  #调度算法4:url_hash(需安装第三方插件).此方法按访问url的hash结果来分配请求,使每个url定向到同一个后端服务器,可以进一步提高后端缓存服务
  upstream webhost {
    server 192.168.0.5:6666 ;
    server 192.168.0.7:6666 ;
    hash $request_uri;
  }
  #调度算法5:fair(需安装第三方插件).这是比上面两个更加智能的负载均衡算法.此种算法可以依据页面大小和加载时间长短智能地进行负载均衡,也就
  #
  #虚拟主机的配置(采用调度算法3:ip_hash)
  server
  {
    listen 80;
    server_name mongo.demo.com;
    #对 "/" 启用反向代理
    location / {
      proxy_pass http://webhost;
      proxy_redirect off;
      proxy_set_header X-Real-IP $remote_addr;
      #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      #以下是一些反向代理的配置,可选.
      proxy_set_header Host $host;
      client_max_body_size 10m; #允许客户端请求的最大单文件字节数
      client_body_buffer_size 128k; #缓冲区代理缓冲用户端请求的最大字节数,
      proxy_connect_timeout 90; #nginx跟后端服务器连接超时时间(代理连接超时)
      proxy_send_timeout 90; #后端服务器数据回传时间(代理发送超时)
      proxy_read_timeout 90; #连接成功后,后端服务器响应时间(代理接收超时)
      proxy_buffer_size 4k; #设置代理服务器（nginx）保存用户头信息的缓冲区大小
      proxy_buffers 4 32k; #proxy_buffers缓冲区,网页平均在32k以下的设置
      proxy_busy_buffers_size 64k; #高负荷下缓冲大小（proxy_buffers*2）
      proxy_temp_file_write_size 64k;
      #设定缓存文件夹大小,大于这个值,将从upstream服务器传
    }
  }
}
```

负载均衡操作演示如下：

操作对象：192.168.0.4（Nginx-Server）

```
# 创建文件夹准备存放配置文件
$ mkdir -p /opt/confs
$ vim /opt/confs/nginx.conf

# 编辑内容如下：
events
{
 use epoll;
 worker_connections 65535;
}

http
{
  upstream webhost {
    ip_hash;
    server 192.168.0.5:6666 ;
    server 192.168.0.7:6666 ;
  }

  server
  {
    listen 80;
    server_name mongo.demo.com;
    location / {
      proxy_pass http://webhost;
      proxy_redirect off;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header Host $host;
      client_max_body_size 10m;
      client_body_buffer_size 128k;
      proxy_connect_timeout 90;
      proxy_send_timeout 90;
      proxy_read_timeout 90;
      proxy_buffer_size 4k;
      proxy_buffers 4 32k;
      proxy_busy_buffers_size 64k;
      proxy_temp_file_write_size 64k;
    }
  }
}
# 然后保存并退出

# 启动负载均衡服务器192.168.0.4（Nginx-Server）
docker run -d -p 8888:80 --name nginx-server -v /opt/confs/nginx.conf:/etc/nginx/nginx.conf --restart always nginx
```

操作对象：192.168.0.5（Nginx-Node1/Nginx-Web1）

```
# 创建文件夹用于存放web页面
$ mkdir -p /opt/html
$ vim /opt/html/index.html

# 编辑内容如下：
<div>
 <h1>
  The host is 192.168.0.5(Docker02) - Node 1!
 </h1>
</div>
# 然后保存并退出

# 启动192.168.0.5（Nginx-Node1/Nginx-Web1）
$ docker run -d -p 6666:80 --name nginx-node1 -v /opt/html:/usr/share/nginx/html --restart always nginx
```

操作对象：192.168.0.7（Nginx-Node2/Nginx-Web2）

```
# 创建文件夹用于存放web页面
$ mkdir -p /opt/html
$ vim /opt/html/index.html

# 编辑内容如下：
<div>
 <h1>
  The host is 192.168.0.7(Docker03) - Node 2!
 </h1>
</div>
# 然后保存并退出

# 启动192.168.0.7（Nginx-Node2/Nginx-Web2）
$ docker run -d -p 6666:80 --name nginx-node2 -v $(pwd)/html:/usr/share/nginx/html --restart always nginx
```

测试：

域名:`mongo.demo.com`，这里是用Windows系统主机访问服务器，要在当前主机的`hosts`中添加解析“`mongo.demo.com 192.168.0.4`”，`hosts`文件所在的路径为“`C:\Windows\System32\drivers\etc`”。这里在Windows主机上通过浏览器访问“`http://mongo.demo.com:8888`”这个站点的时候，Nginx会根据来访的主机的ip_hash值，负载均衡到192.168.0.5（Nginx-Node1/Nginx-Web1）和192.168.0.7（Nginx-Node2/Nginx-Web2）服务器上。如果其中一个Web服务器无效后，负载均衡服务器会自动将请求转发到正常的Web服务器。

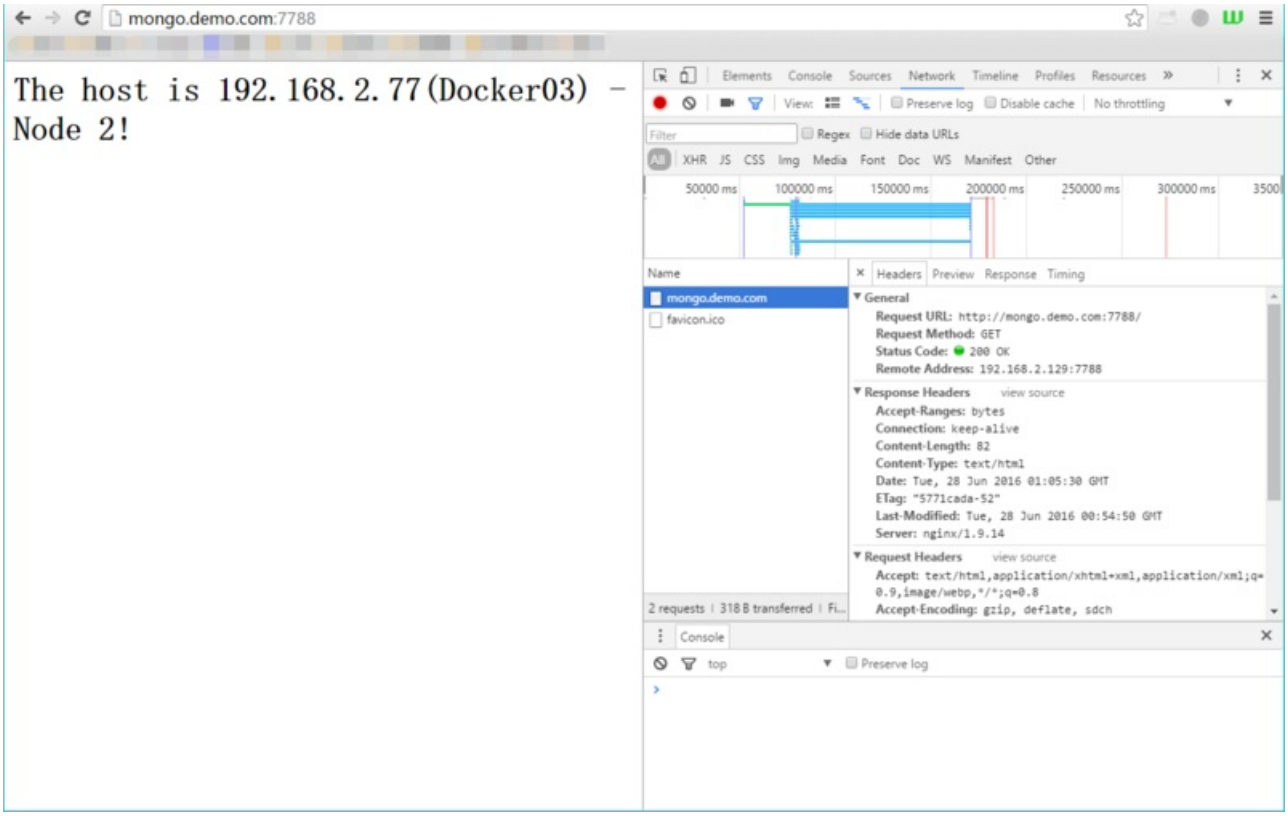
```
# Copyright (c) 2014-2016, racaljk.
# https://github.com/racaljk/hosts
# Last updated: 2016-06-04

# This work is licensed under a CC BY-NC-SA 4.0 International License.
# https://creativecommons.org/licenses/by-nc-sa/4.0/

# http://laod.cn/hosts/2016-google-hosts.html

# Localhost (DO NOT REMOVE)
127.0.0.1 localhost
255.255.255.255 broadcasthost
::1    localhost
fe80::1%lo0  localhost

192.168.2.129 mongo.demo.com
```

下图是另外做的一组demo的访问效果图，而且容器的端口和IP不同（所有信息都做了相应修改）：

1.Nginx-Server：192.168.2.129（Docker01）；

2.Nginx-Node1：192.168.2.56（Docker02）；

3.Nginx-Node2：192.168.2.77（Docker03）；



参考链接：

http://www.cnblogs.com/xcloudbiz/articles/5234373.html

http://wangying.sinaapp.com/archives/931

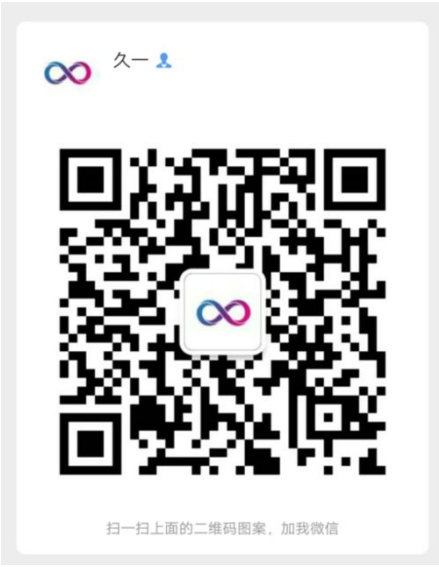
http://www.php100.com/html/program/nginx/2013/0905/5525.html

https://hub.docker.com/_/nginx/

【END】

如果看到这里，说明你喜欢这篇文章，请**转发、点赞**。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓扫描二维码进群↓



推荐阅读

1. 一键下载 Pornhub 视频！
2. 硬核文章：扛住 100 亿次请求？我们来试一试
3. 程序员开发了一款软件，完成了舔狗的绝地反杀
4. Spring Boot 多模块项目实践（附打包方法）
5. 团队开发中 Git 最佳实践

喜欢文章，点个在看

阅读原文

Redis + Tomcat + Nginx 集群实现 Session 共享

蕃薯耀 Java后端 2019-11-03

作者 | 蕃薯耀

链接 | www.cnblogs.com/fanshuyao

上篇 | 35 个小细节，提升 Java 代码运行效率

一、Session共享使用tomcat-cluster-redis-session-manager插件实现

> 插件地址见：
>
> https://github.com/ran-jit/tomcat-cluster-redis-session-manager

该插件支持Tomcat7、Tomcat8、Tomcat9

或者直接在附件中下载（版本为2.0.2，2017-11-27日前最新版本）

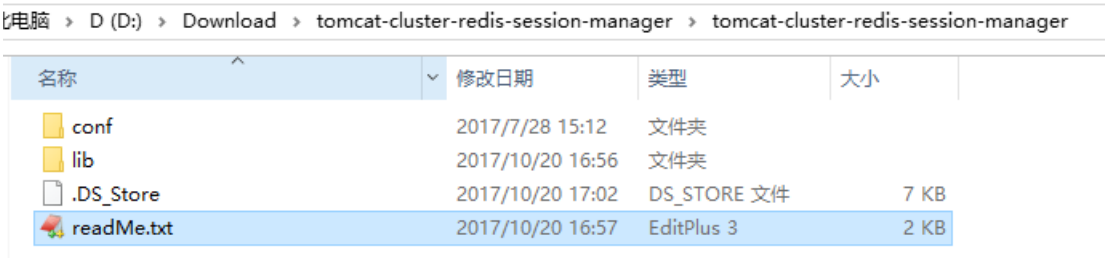
> http://dl.iteye.com/topics/download/d9fffd9d-84dd-385b-b10e-6376eaf0c815

这里有是一个只支持Tomcat7的，不支持tomcat8，暂时不见新的维护：

> https://github.com/jcoleman/tomcat-redis-session-manager

二、tomcat-cluster-redis-session-manager详解

1、解压后的文件如下：



conf目录下有一个redis-data-cache.properties：

Redis的配置文件

```
#-- Redis data-cache configuration

#- redis hosts ex: 127.0.0.1:6379, 127.0.0.2:6379, 127.0.0.2:6380, ....
redis.hosts=127.0.0.1:6379

#- redis password (for stand-alone mode)
#redis.password=

#- set true to enable redis cluster mode
redis.cluster.enabled=false

#- redis database (default 0)
#redis.database=0

#- redis connection timeout (default 2000)
#redis.timeout=2000
```

ib目录下有4个jar包，如下：

1. commons-logging-1.2.jar
2. commons-pool2-2.4.2.jar
3. jedis-2.9.0.jar
4. tomcat-cluster-redis-session-manager-2.0.1.jar

三、使用方法：

压缩文件中有使用方法，见readMe.txt 文件：

第一步：

```
1. Move the downloaded jars to tomcat/lib directory
    * tomcat/lib/
```

就是把lib目录下的Jar包全复制到tomcat/lib目录下

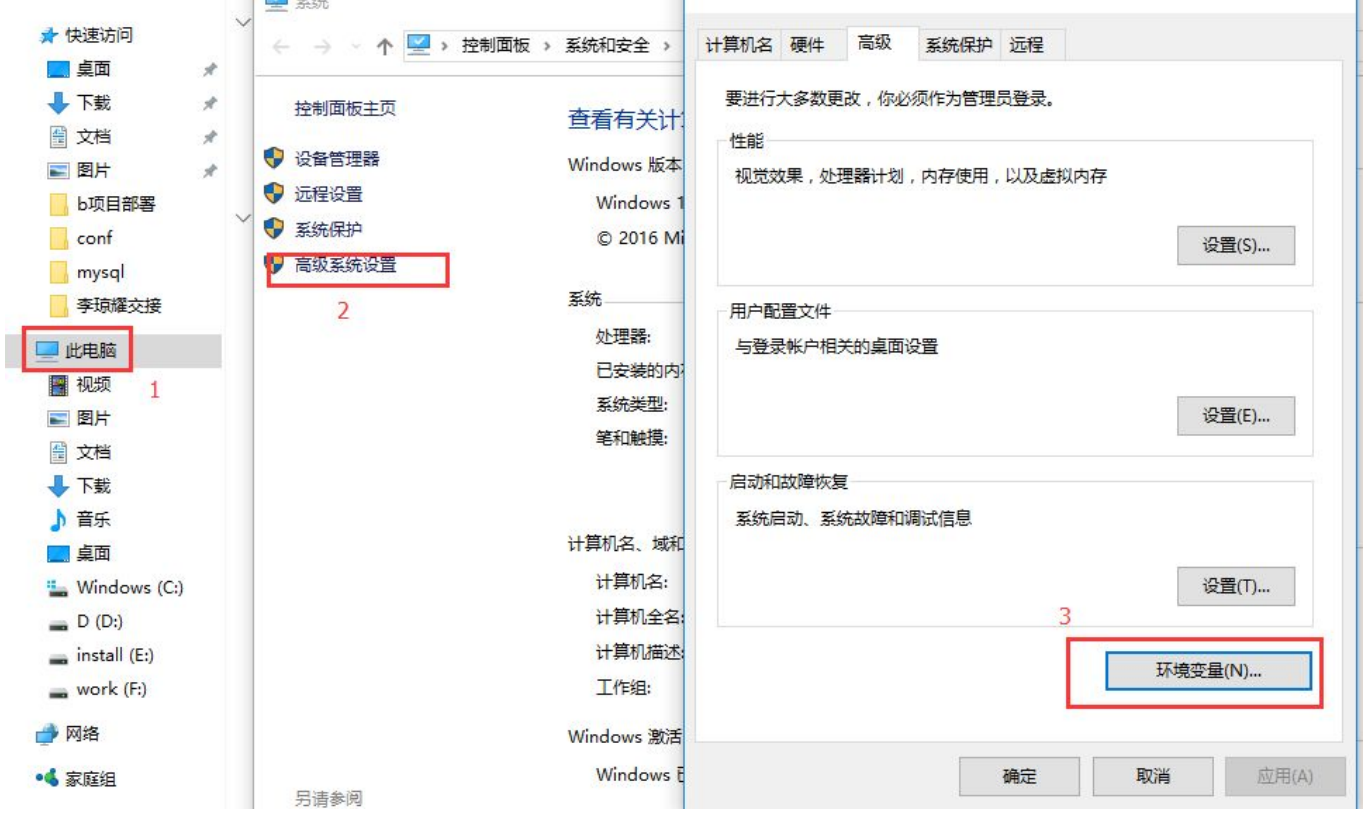
> （一般来说tomcat是集群，至少有2个tomcat，所以先配置好一个tomcat，复制完文件后，再将tomcat文件重新复制一份，这样省事，但需要修改tomcat相应的端口）

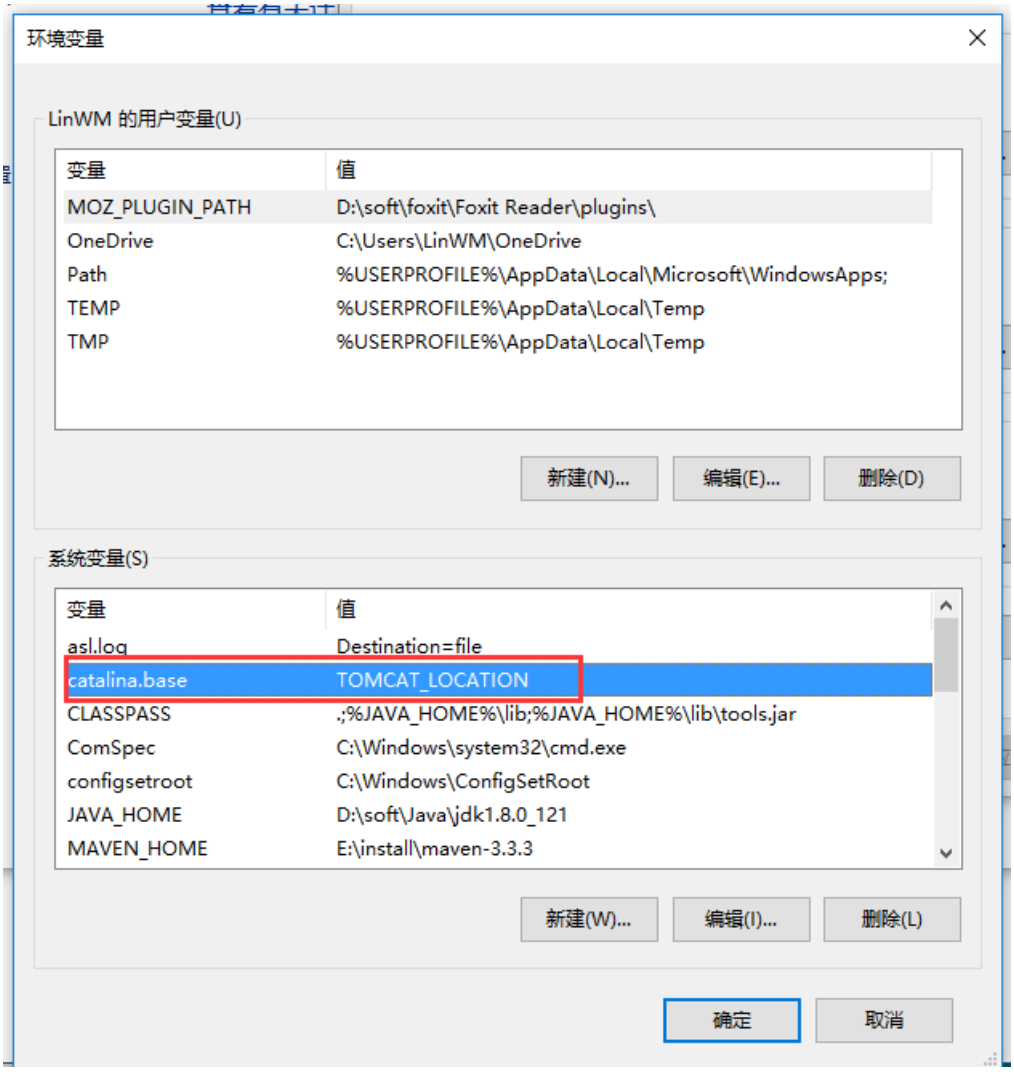
第二步：

```
2. Add tomcat system property "catalina.base"
    * catalina.base="TOMCAT_LOCATION"
```

就是配置一个环境变量，和Jdk配置的环境变量一样，需要配置一个catalina.base的环境变量，值为TOMCAT_LOCATION

如下：





第三步：

3. Extract downloaded package (tomcat-cluster-redis-session-manager.zip) **to** configure Redis credentials in redis-data-cache.pr
    * tomcat/**conf**/redis-data-cache.properties

把conf目录下的配置文件redis-data-cache.properties复制到tomcat/conf/目录下

第四步：

```
4. Add the below two lines in tomcat/conf/context.xml
    <Valve className="tomcat.request.session.redis.SessionHandlerValve" />
    <Manager className="tomcat.request.session.redis.SessionManager" />
```

在tomcat/conf/目录下的context.xml文件，加上相应的配置，如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
 Licensed to the Apache Software Foundation (ASF) under one or more
 contributor license agreements. See the NOTICE file distributed with
 this work for additional information regarding copyright ownership.
 The ASF licenses this file to You under the Apache License, Version 2.0
 (the "License"); you may not use this file except in compliance with
 the License. You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
--><!-- The contents of this file will be loaded for each web application --><Context>

  <!-- Default set of monitored resources. If one of these changes, the -->
  <!-- web application will be reloaded. -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

  <!-- Uncomment this to disable session persistence across Tomcat restarts -->
  <!--
  <Manager pathname="" />
  -->

  <!-- Uncomment this to enable Comet connection tacking (provides events
       on session expiration as well as webapp lifecycle) -->
  <!--
  <Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
  -->
  <Valve className="tomcat.request.session.redis.SessionHandlerValve"/>
  <Manager className="tomcat.request.session.redis.SessionManager"/>

</Context>
```

第五步：

```
5. Verify the session expiration time (minutes) in tomcat/conf/web.xml
    <session-config>
      <session-timeout>60<session-timeout>
    <session-config>
```

修改session的过期时间，默认是30分钟，可以不需要此步骤。

session集群的配置至此结束。

四、Nginx集群

1、下载Nignx：

> http://nginx.org/en/download.html

本人练习时使用windows，所以下载的windows版本：

> http://nginx.org/download/nginx-1.13.7.zip

2、下载后解压：D:\soft\nginx-1.12.2

> （之前使用的是1.12.2的版本，现在最新版是1.13.7，但都一样，附件中有1.12.2版本提供下载）

3、修改Nginx配置文件nginx.conf

进入conf目录（D:\soft\nginx-1.12.2\conf），找到nginx.conf配置文件，打开编辑：

3.1在http{……}里加上upstream，如下：

```
upstream myTomcatCluster{# tomcatCluster和proxy_pass保持一样
    #解决session的问题
    #ip_hash;#加上这个，解决Session每次访问页面都不一样，加上就一样了。

    #这里是tomcat的地址，weight越大，访问机率越大。
    server 127.0.0.1:9300 weight=1 fail_timeout=5s max_fails=1;
    server 127.0.0.1:9400 weight=1 fail_timeout=5s max_fails=1;
  }
```

server：配置tomcat服务器请求的地址，2台Tomcat服务就配置2个server，分别对应9300，9400端口

weight 表示权重，权重越大，访问到的机率越大。

3.2、修改location / {……}

默认是这个的：

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

修改成这样：

```
location / {
        #root html;
    proxy_pass http://myTomcatCluster;
        #index index.html index.htm;
    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_connect_timeout     1;
        proxy_read_timeout        1;
        proxy_send_timeout        1;
    }
```

最简单的配置就是：

```
location / {
   proxy_pass http://myTomcatCluster;
}
```

myTomcatCluster 对应upstream后的命名。

下面的配置可以解决2个Tomcat服务器集群，当一台服务器挂掉（宕机）后，请求变得很慢的问题。

> （Tomcat集群一台服务器挂掉后请求变慢解决方案）

```
proxy_connect_timeout    1;
proxy_read_timeout       1;
proxy_send_timeout       1;
```

3.3、启动Nginx服务器

使用Windows命令行启动

（1）进入D盘：d:

（2）进入D:\soft\nginx-1.12.2目录：

```
cd D:\soft\nginx-1.12.2
```

（3）启动服务：（启动一闪而过，但打开进程管理器能看到是已经启动的）

```
start nginx
```

关闭服务的命令：nginx -s stop

重新加载的命令：nginx -s reload，修改配置文件后，可以使用该命令直接加载，不需要重启。

五、测试集群：

1、tomcat准备

将已经配置好的一个tomcat复制一份，修改端口，然后再修改一下tomcat的配置文件（server.xml）

我的一个tomcat在：

> D:\soft\apache-tomcat-8.0.45-9300\conf

另一个是：

> D:\soft\apache-tomcat-8.0.45-9400\conf

修改：

```
<Engine defaultHost="localhost" name="Catalina">
```

其中tomcat 9300端口的修改如下：

```
<Engine defaultHost="localhost" jvmRoute="jvm9300" name="Catalina">
```

tomcat 9400端口的修改如下：

```
<Engine defaultHost="localhost" jvmRoute="jvm9400" name="Catalina">
```

2、项目准备：

新建立一个web项目，然后新建立一个index.jsp的文件，如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>首页redis-session</title>
</head>
<body>
  <div>tomcat 集群测试</div>
  <div>
    <%
     //HttpSession session = request.getSession(true);
     System.out.println(session.getId());
     out.println("<br> SESSION ID:" + session.getId()+"<br>");
    %>
  </div>
</body>
</html>
```

主要是在打印页面输出sessionId的信息：

```
out.println("<br> SESSION ID:" + session.getId()+"<br>");
```

然后把这个项目分别部署到9300、9400端口的2个tomcat中，分别启动，记得也启动Nginx和redis哦

然后打开浏览器通过地址访问项目：http://localhost/redis-session/ （使用Nginx集群分发，不需要端口号访问），显示如下：

```
tomcat 集群测试

SESSION ID:B837ECA85B47081EAA2FEFCD7E579CD2.jvm9400
```

无论怎么刷新访问（打开新的标签页也是（非新窗口））的都是jvm9400，也就是端口号为9400的tomcat

后缀.jvm9400就是前面配置的：

```
<Engine defaultHost="localhost" jvmRoute="jvm9400" name="Catalina">
```

打开新的隐身窗口访问：

```
tomcat 集群测试

SESSION ID:83BBA58F4EB7B2EFF90AE05D4A0629FD.jvm9300
```

这时访问的是端口号为9300的tomcat，通过后缀.jvm9300判断知道。

新窗口每次访问的是都是tomcat9300，session也不会变。

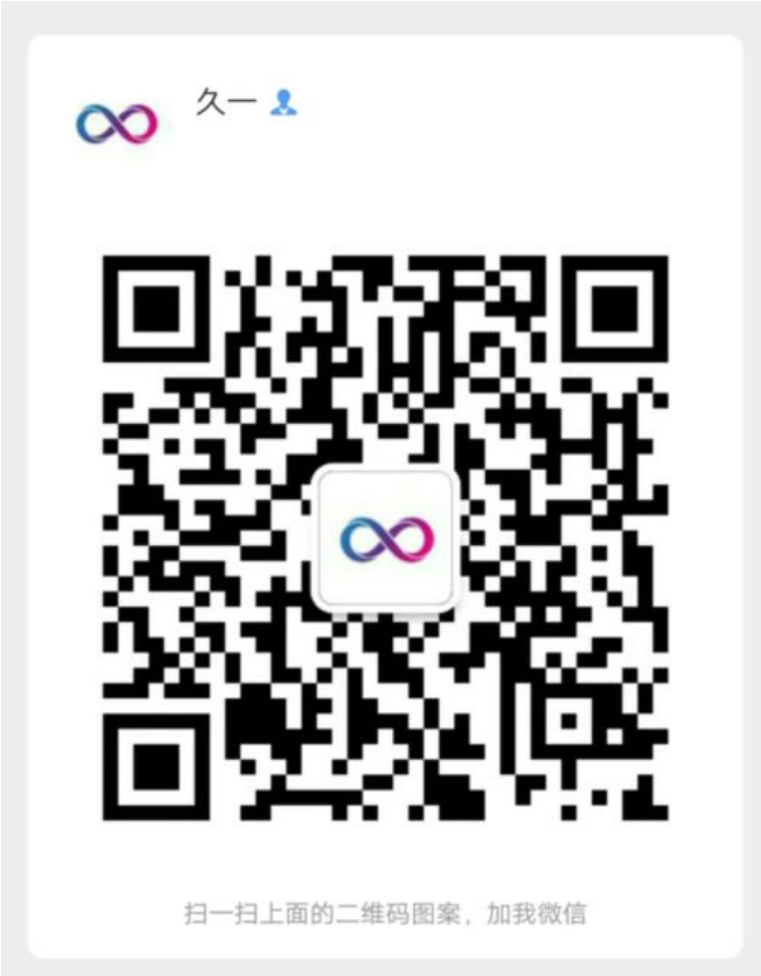
在访问后缀为.jvm9400时，把端口9400的tomcat关掉，再次刷新访问，sessionId一样不变，由此可见，2个tomcat的sessionId是共享的。

使用Redis实现session共享的好处就是，把session管理放在redis中，如果服务器重启或挂机，sessionId保存在redis中，下次重启后一样生效，避免sessionId失效，同样redis最好也做集群，避免redis重启或挂机。

-END-

如果看到这里，说明你喜欢这篇文章，请**转发、点赞**。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓扫描二维码进群↓



推荐阅读

1. 盘点阿里巴巴 33 个牛逼的开源项目
2. 为什么我不建议你去外包公司?
3. 理解 IntelliJ IDEA 的项目配置和 Web 部署
4. Java 开发中常用的 4 种加密方法
5. 团队开发中 Git 最佳实践

学Java，请关注公众号：Java后端

喜欢文章，点个在看

一文读懂并实操 Nginx 反向代理

圈圈的圈 Java后端 2019-11-19

作者 | 圈圈的圈

链接 | juejin.im/post/5c0e6d606fb9a049f66bf246

前言

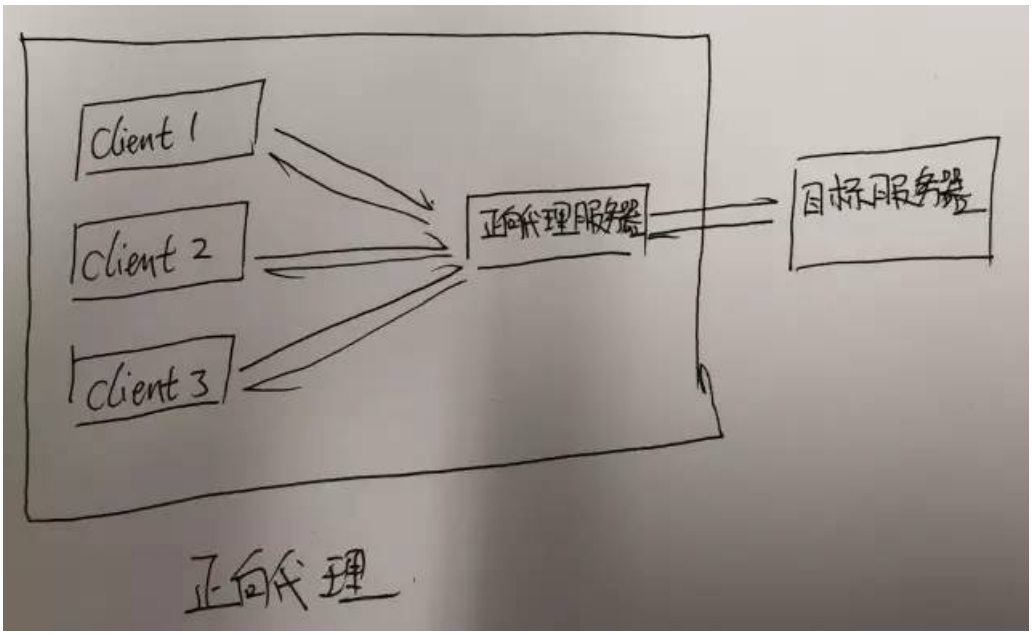
代理是个啥

既然要聊反向代理, 那首先得知道代理是个啥吧? 嗯.

正向代理

比如, 你买束花, 想要给隔壁工位的测试妹子小丽表白. 但是又怕被人家直面拒绝太没面子. 于是你把鲜花委托给平时和小丽一起的测试小伙伴小红. 让她帮忙把花送给小丽. 这就是一个简单的代理过程, 小红作为代理帮你把花送给了小丽, 当然这种情况在现实中并不推荐使用, 因为难以避免中间商赚差价 😂.

在上面的例子中, 你作为客户端(请求方), 想要向服务方(小丽)发起请求. 但是碍于面子你主动找到了第三方(小红)作为代理向服务方发送请求, 这种情况就是常说的正向代理. 正向代理在互联网中的使用主要是科学上网, 你想访问谷歌但是碍于防火墙你只能通过vpn服务器作为代理才能访问. 这个时候一般也要找值得信赖的vpn厂商, 避免中间商赚差价 😄.

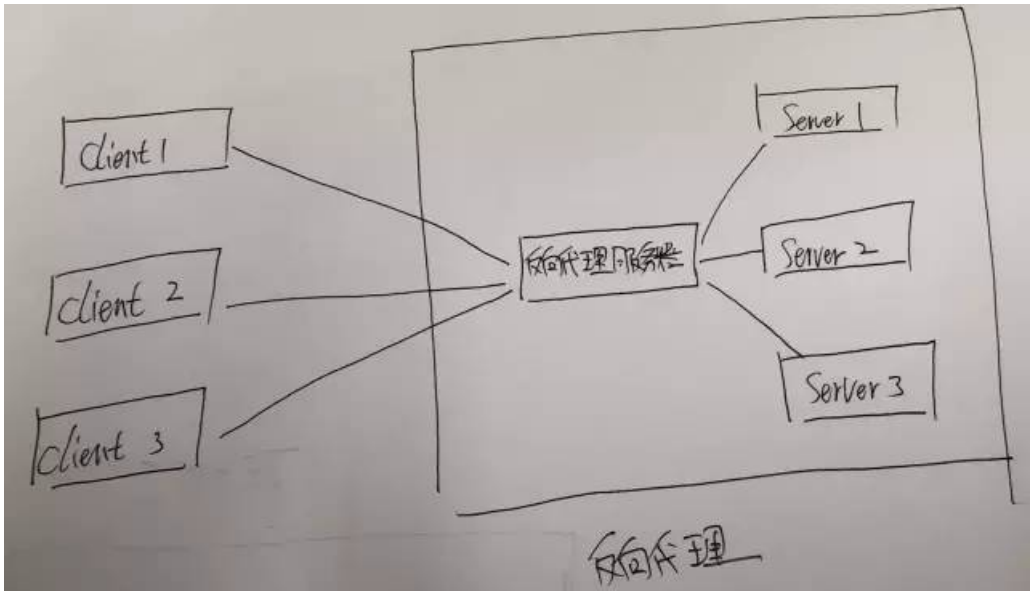


反向代理

关于反向代理的例子, 那就比较多啦. 比如, 孤独的你躺在床上夜不能寐. 于是乎, 拿出手机, 点亮了屏幕, 拨通10086, 中国移动就会随机分配一个当前处于空闲的客服MM, 你可以和客服MM聊聊天, 问问她家住哪里, 有没有男朋友, 她的微信号, 她的手机号, 星座, 八字.....

在这个例子中, 中国移动就充当了反向代理的角色. 你只需要拨打10086. 至于会不会分配到 MM 会分配到哪个 MM 在接通之前你都是不知道的. 反向代理在互联网中的使用主要是实现负载均衡. 当你访问某个网站的时候, 反向代理服务器会从当前网站的所有服

务器中选择一个空闲的服务器为你响应. 用于均衡每台服务器的负载率.



修改 hosts 完成域名绑定

mac 用户直接执行 vim /private/etc/hosts 在 hosts 文件最后添加一行:

```
127.0.0.1 a.com
```

这一句是什么意思呢? 就是告诉我们的电脑访问 a.com 的时候, 无需请求 DNS, 直接指向我们本机.

ps: win 环境下, hosts 文件在C:\Windows\System32\drivers\etc 文件夹下. 如果没有权限修改, 把 hosts 文件先拷贝到别的位置, 通过编辑器打开并添加最后一行内容以后再剪切到原来的位置替换即可.

验证: 打开命令行窗口执行 ping a.com, 如果访问的 ip 为 127.0.0.1 说明我们的域名绑定就完成啦 ^_^

Tips：可以微信搜索：Java后端，关注后加入咱们自己的交流群。

安装 nginx

> 要做 NGINX 反向代理, 肯定要安装 nginx, 本文安装步骤示例环境为 mac, win 的小伙伴, 可以百度一下嗷, 这个东西大同小异.

- 安装 brew 命令, 执行ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
- 安装 nginx, 执行 brew install nginx
- 启动 nginx nginx, 如果报没有权限, 执行sudo nginx

nginx 启动后, 浏览器打开 localhost:8080, 即可验证. 出现以下界面说明安装成功.

nginx 配置初探

配置完 hosts 域名已经能够成功绑定. 现在如果我们访问a.com 实际上是会访问到我们的自己的电脑辣. 那还不抓紧试一下?

浏览器访问 a.com

这是什么鬼????

为什么会 **无法访问此网站** 呢? 我们下载安装完 nginx 还没有做任何配置. 接下来, 我们稍微配置一下就 OK:

- 命令行切换到 nginx 配置目录下cd /usr/local/etc/nginx/servers
- 创建并编辑配置文件vim test.conf, 在配置文件中粘贴以下内容

```
server {
  # 监听80端口号
  listen 80;

  # 监听访问的域名
  server_name a.com;

  # 根据访问路径配置
  location / {
    # 把请求转发到 https://www.baidu.com
    proxy_pass https://www.baidu.com;
  }
}
```

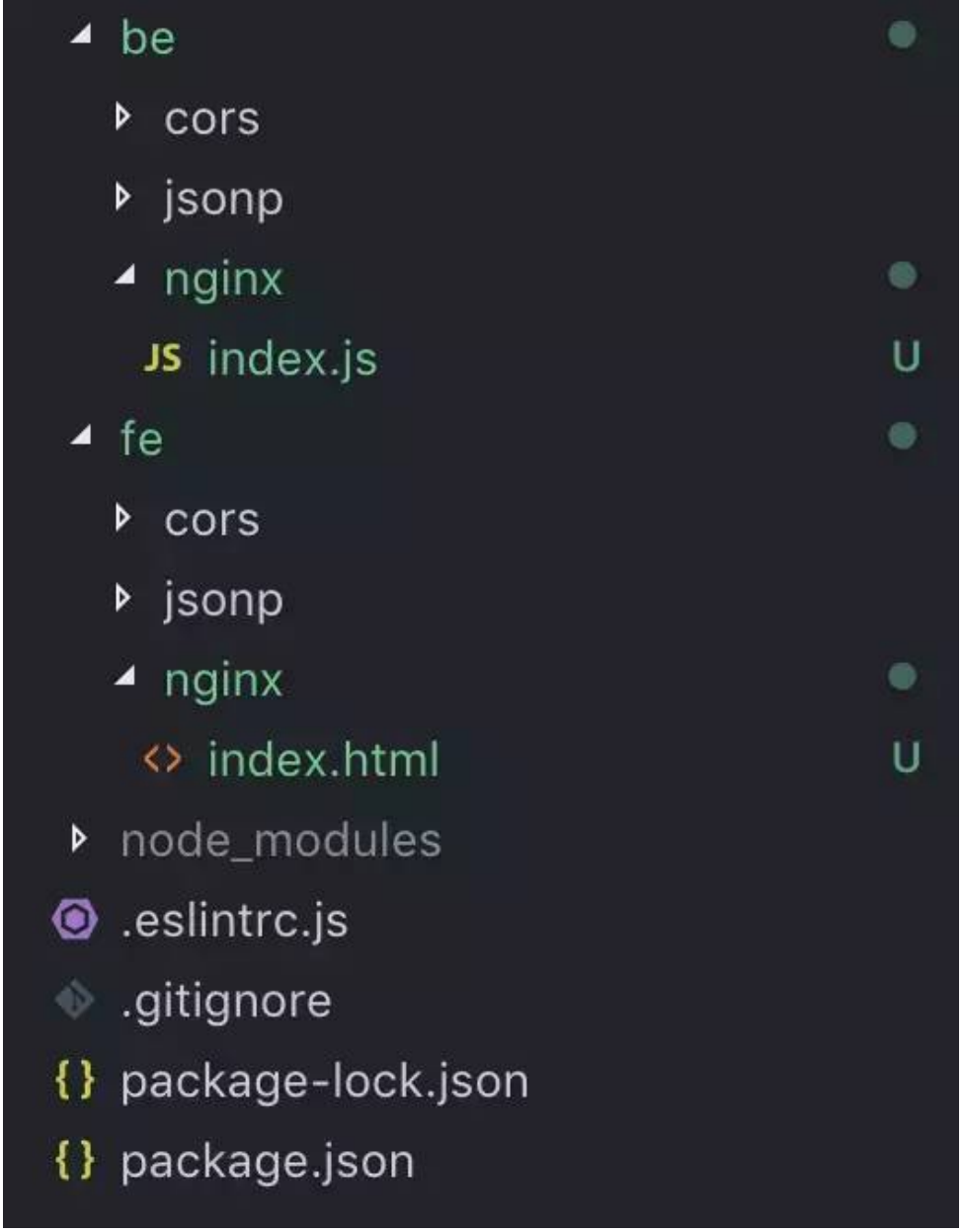
- 保存文件, 并执行nginx -s reload 重启 nginx.
- 回到浏览器, 打开 a.com 的页签, 强制刷新.

恭喜你已经完成了第一个 nginx 配置.

创建跨域环境

> 通过一系列的折腾, 我们已经可以通过 nginx 将a.com 转发到百度. 完成了第一步, 接下来我们创建跨域的 Case 并一步一步通过 nginx 配置实现跨域.

首先, 项目前后端添加 nginx 目录, 用户存放前后端代码. 代码结构如下图所示.

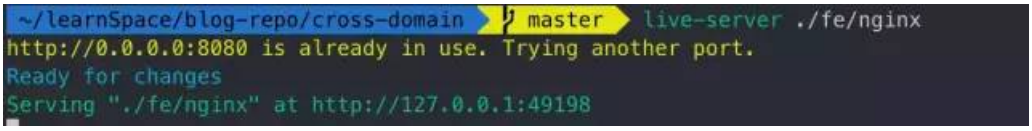


其次编写前后端代码:

前端代码(./fe/nginx/index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>CORS 实现跨域</title>
</head>
<body>
  <h3>CORS 实现跨域</h3>

  <script>
    var xhr = new XMLHttpRequest()
    xhr.open('GET', 'http://localhost:8888/api/getFriend')
    xhr.setRequestHeader('token', 'quanquanbunengshuo')
    xhr.withCredentials = true;
    xhr.onreadystatechange = function() {
      if(xhr.readyState === 4 && xhr.status === 200) {
        console.log(xhr.responseText)
        console.log(xhr.getAllResponseHeaders())
      }
    }
    xhr.send()
  </script>
</body>
</html>
```

编写完前端代码以后, 启动前端 web 容器.live-server ./fe/nginx



命令行中出现了黄色警告, 通知我们 8080 端口已经被占用, 这又是为什么呢? 大家请思考一哈.

我们重新指定一个端口live-server ./fe/nginx --port=9999 哈哈, 换一个指令, 依旧是那么顺畅. ^_^

后端代码:

```
const http = require('http');

const PORT = 8888;


const server = http.createServer((request, response) => {
 console.log(request.headers)
 response.end("{name: 'quanquan', friend: 'guiling'}");
});


server.listen(PORT, () => {
 console.log('服务启动成功, 正在监听: ', PORT);
});
```

启动后端服务 node ./be/nginx/index.js

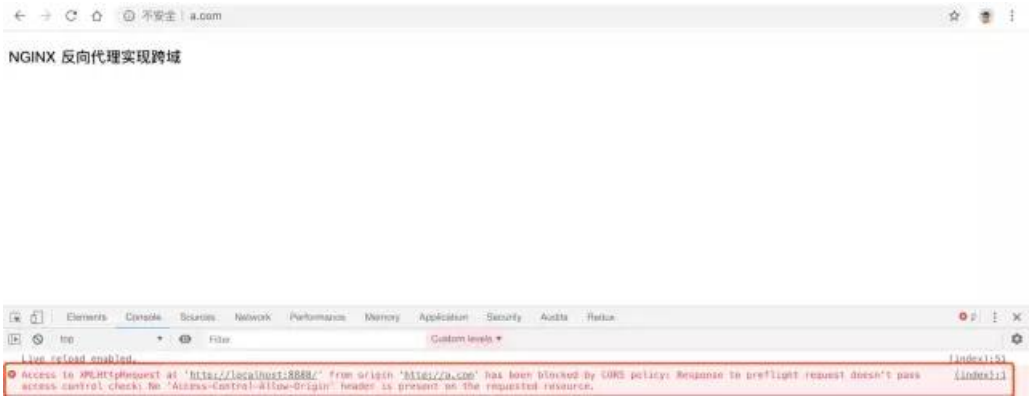
完善 nginx 配置

> 前后端代码已经准备完成, 这一步我们就来点干货. 完成最后的配置.

- 首先, 修改 nginx 配置, 把百度地址替换成本地的前端地址

```
server {
  # 监听80端口号
  listen 80;

  # 监听访问的域名
  server_name a.com;

  # 根据访问路径配置
  location / {
    # 把请求转发到 http://127.0.0.1:9999
    proxy_pass http://127.0.0.1:9999;
  }
}
```

- 修改完成 nginx 配置文件以后, 切记执行 nginx -s -reload 重启 nginx.
- 访问a.com

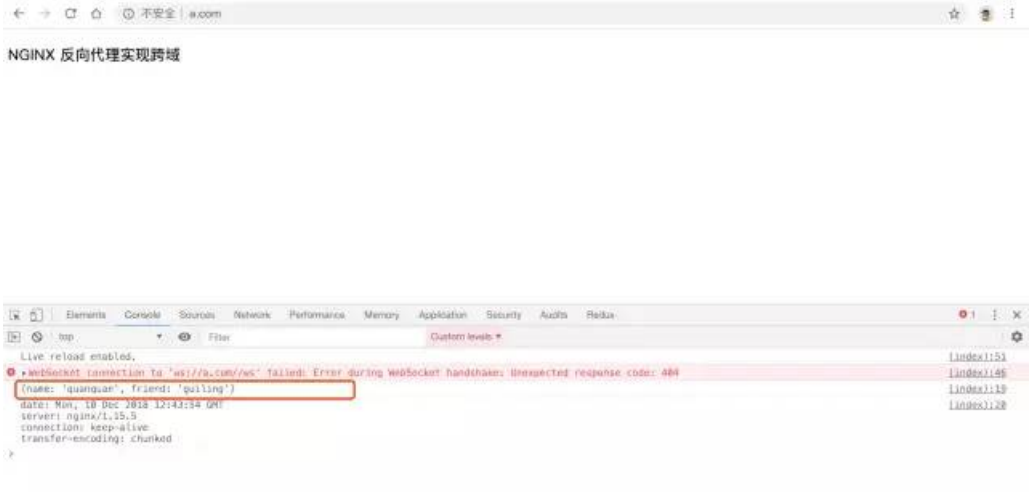


熟悉的报错又出现了...

- 修改前端项目中的接口地址：xhr.open('GET', '/api/getFriend')
- 修改 nginx 配置文件

```
server {
  # 监听80端口号
  listen 80;

  # 监听访问的域名
  server_name a.com;

  # 根据访问路径配置
  location / {
    # 把请求转发到 http://127.0.0.1:9999
    proxy_pass http://127.0.0.1:9999;
  }

  # 监听根目录下的 /api 路径
  location /api/ {
    # 把请求转发到 http://127.0.0.1:8888
    proxy_pass http://127.0.0.1:8888;
  }
}
```

新加的对于 api 路径的监听的意思就是把关于后端 api 的请求转发到后端项目上(哈哈, 当然这就是为啥好多后端接口都是要有 /api 开头的啦). 重启 nginx 以后, 再次刷新浏览器, 后端返回的结果已经成功的打印到了控制台, 本次跨域访问任务完成.



细心的小伙伴肯定发现了, 控制台还有一个报错. 这个是因为我们的项目中用到了 live-server 这个工具需要 websocket 导致的. 我们可以通过添加以下配置解决.

```
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
```

报错消失 😄, 此时完整的 nginx 配置文件为

```
server {
  # 监听80端口号
  listen 80;

  # 监听访问的域名
  server_name a.com;

  # 根据访问路径配置
  location / {
    # 把请求转发到 http://127.0.0.1:9999
    proxy_pass http://127.0.0.1:9999;

    # 兼容websocket
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
  }

  # 监听根目录下的 /api 路径
  location /api/ {
    # 把请求转发到 http://127.0.0.1:8888
    proxy_pass http://localhost:8888;
  }
}
```

前后端代码地址为:

https://github.com/luoquanquan/cross-domain/commit/f38f56689fdac1526244ecadaa979a52c9c4a7ea

总结

至此, 我们已经通过 nginx 反向代理的方式实现了跨域访问 api, 在系列文章第一篇对于跨域的解释为: 跨域源于同源策略, 是浏览器保证用户安全的行为. 我们使用的 nginx 反向代理实际上是对浏览器的一种 "哄骗", 让它认为自己访问到的是同域的 api. 实际上是在服务端做了个调包, 这个道理就如同你拨打 10086 你就认定了给你分配到的一定是中国移动的客服MM(客服GG也是有可能出现的 😄)而中国移动的客服MM就是一个很安全的聊天对象, 没有必要再进行限制...

下集预告: 终于蹩脚的码完了最后一行, 作为生产环境中最常用的 nginx 反向代理, 比我想象的要简单很多很多. 由于涉及到诸多配置的步骤. 有写的不明白的地方还望小伙伴们评论区一起讨论. 下一节预计聊聊服务端代理 ServerProxy 这个也是我要做的接口测试工具需要用到的技术方案, See you.

从 Nginx、Apache 工作原理看为什么 Nginx 比 Apache 高效！

Java后端 2月14日

来源 | 代码湾

链接 | codebay.cn/post/8557.html

Nginx才短短几年，就拿下了Web服务器大壁江山，众所周知，Nginx在处理大并发静态请求方面，效率明显高于Httpd，甚至能轻松解决C10K问题。

在高并发连接的情况下，Nginx是Apache服务器不错的替代品。Nginx同时也可以作为7层负载均衡服务器来使用。根据我的测试结果，Nginx + PHP(FastCGI) 可以承受3万以上的并发连接数，相当于同等环境下Apache的10倍。

一般来说，4GB内存的服务器+Apache（prefork模式）一般只能处理3000个并发连接，因为它们将占用3GB以上的内存，还得为系统预留1GB的内存。我曾经就有两台Apache服务器，因为在配置文件中设置的MaxClients为4000，当Apache并发连接数达到3800时，导致服务器内存和Swap空间用满而崩溃。

而这台 Nginx + PHP(FastCGI) 服务器在3万并发连接下，开启的10个Nginx进程消耗150M内存（`15M*10=150M`），开启的64个php-cgi进程消耗1280M内存（`20M*64=1280M`），加上系统自身消耗的内存，总共消耗不到2GB内存。如果服务器内存较小，完全可以只开启25个php-cgi进程，这样php-cgi消耗的总内存数才500M。

在3万并发连接下，访问Nginx+ PHP(FastCGI) 服务器的PHP程序，仍然速度飞快。

为什么Nginx在处理高并发方面要优于httpd，我们先从两种web服务器的工作原理以及工作模式说起。

一、Apache三种工作模式

我们都知道Apache有三种工作模块，分别为：prefork、worker、event。

- **prefork：**多进程，每个请求用一个进程响应，这个过程会用到select机制来通知。
- **worker：**多线程，一个进程可以生成多个线程，每个线程响应一个请求，但通知机制还是select不过可以接受更多的请求。
- **event：**基于异步I/O模型，一个进程或线程，每个进程或线程响应多个用户请求，它是基于事件驱动（也就是epoll机制）实现的。

1、prefork的工作原理

如果不用“–with-mpm”显式指定某种MPM，prefork就是Unix平台上缺省的MPM。它所采用的预派生子进程方式也是Apache1.3中采用的模式。prefork本身并没有使用到线程，2.0版使用它是为了与1.3版保持兼容性；另一方面，prefork用单独的子进程来处理不同的请求，进程之间是彼此独立的,这也使其成为最稳定的MPM之一。

2、worker的工作原理

相对于prefork，worker是2.0版中全新的支持多线程和多进程混合模型的MPM。由于使用线程来处理，所以可以处理相对海量的请求，而系统资源的开销要小于基于进程的服务器。但是，worker也使用了多进程,每个进程又生成多个线程，以获得基于进程服务器的稳定性，这种MPM的工作方 式将是Apache2.0的发展趋势。

3、event 基于事件机制的特性

一个进程响应多个用户请求，利用callback机制，让套接字复用，请求过来后进程并不处理请求，而是直接交由其他机制来处理，通过epoll机制来通知请求是否完成；在这个过程中，进程本身一直处于空闲状态，可以一直接收用户请求。可以实现一个进程程响应多个用户请求。支持持海量并发连接数，消耗更少的资源。

二、如何提高Web服务器的并发连接处理能力

有几个基本条件：

1、基于线程，即一个进程生成多个线程，每个线程响应用户的每个请求。

2、基于事件的模型，一个进程处理多个请求，并且通过epoll机制来通知用户请求完成。

3、基于磁盘的AIO（异步I/O）

4、支持mmap内存映射，mmap传统的web服务器，进行页面输入时，都是将磁盘的页面先输入到内核缓存中，再由内核缓存中复制一份到web服务器上，mmap机制就是让内核缓存与磁盘进行映射，web服务器，直接复制页面内容即可。不需要先把磁盘的上的页面先输入到内核缓存去。

刚好，Nginx 支持以上所有特性。所以Nginx官网上说，Nginx支持50000并发，是有依据的。

三、Nginx优异之处

传统上基于进程或线程模型架构的Web服务通过每进程或每线程处理并发连接请求，这势必会在网络和I/O操作时产生阻塞，其另一个必然结果则是对内存或CPU的利用率低下。

生成一个新的进程/线程需要事先备好其运行时环境，这包括为其分配堆内存和栈内存，以及为其创建新的执行上下文等。这些操作都需要占用CPU，而且过多的进程/线程还会带来线程抖动或频繁的上下文切换，系统性能也会由此进一步下降。

另一种高性能web服务器/Web服务器反向代理：Nginx，Nginx的主要着眼点就是其高性能以及对物理计算资源的高密度利用，因此其采用了不同的架构模型。受启发于多种操作系统设计中基于“事件”的高级处理机制，Nginx采用了模块化、事件驱动、异步、单线程及非阻塞的架构，并大量采用了多路复用及事件通知机制。

在Nginx中，连接请求由为数不多的几个仅包含一个线程的进程Worker以高效的回环(run-loop)机制进行处理，而每个Worker可以并行处理数千个的并发连接及请求。

四、Nginx 工作原理

Nginx会按需同时运行多个进程：一个主进程(master)和几个工作进程(worker)，配置了缓存时还会有缓存加载器进程(cache loader)和缓存管理器进程(cache manager)等。所有进程均是仅含有一个线程，并主要通过“共享内存”的机制实现进程间通信。主进程以root用户身份运行，而worker、cache loader和cache manager均应以非特权用户身份运行。

在高连接并发的情况下，Nginx是Apache服务器不错的替代品。

Nginx 安装非常的简单 , 配置文件非常简洁（还能够支持perl语法）,Bugs 非常少的服务器: Nginx 启动特别容易, 并且几乎可以做

到7*24不间断运行，即使运行数个月也不需要重新启动. 你还能够 不间断服务的情况下进行软件版本的升级 。

五、Nginx 的诞生主要解决C10K问题

最后我们从各自使用的多路复用IO模型来分析：

1、select模型：（apache使用，由于受模块等限制，用的不多）；

单个进程能够 监视的文件描述符的数量存在最大限制；

select()所维护的 存储大量文件描述符的数据结构 ，随着文件描述符数量的增长，其在用户态和内核的地址空间的复制所引发的开销也会线性增长；

由于网络响应时间的延迟使得大量TCP连接处于非活跃状态，但调用select()还是会对 所有的socket进行一次线性扫描 ，会造成一定的开销；

2、poll：poll是unix沿用select自己重新实现了一遍，唯一解决的问题是poll 没有最大文件描述符数量的限制；

3、epoll模型：（Nginx使用）

epoll带来了两个优势，大幅度提升了性能：

（1）基于事件的就绪通知方式 ，select/poll方式，进程只有在调用一定的方法后，内核才会对所有监视的文件描述符进行扫描，而epoll事件通过epoll_ctl()注册一个文件描述符，一旦某个文件描述符就绪时，内核会采用类似call back的回调机制，迅速激活这个文件描述符，epoll_wait()便会得到通知

（2）调用一次epoll_wait()获得就绪文件描述符时，返回的并不是实际的描述符，而是一个代表就绪描述符数量的值，拿到这些值去epoll指定的一个数组中依次取得相应数量的文件描述符即可，这里使用内存映射（mmap）技术， 避免了复制大量文件描述符带来的开销

（3）当然epoll也有一定的局限性， epoll只有Linux2.6才有实现 ，而其他平台都没有，这和apache这种优秀的跨平台服务器，显然是有些背道而驰了。

（4）简单来说epoll是select的升级版，单进程管理的文件描述符没有最大限制。但epoll只有linux平台可使用。作为跨平台的Apache没有使用

全面了解 Nginx 主要应用场景

RayeBlog Java后端 1月6日

作者：RayeBlog

https://raye.wang/

前言

本文只针对Nginx在不加载第三方模块的情况能处理哪些事情，由于第三方模块太多所以也介绍不完，当然本文本身也可能介绍的不完整，毕竟只是我个人使用过和了解到过得。所以还请见谅，同时欢迎留言交流

Nginx能做什么

1. 反向代理

2. 负载均衡

3. HTTP服务器（包含动静分离）

4. 正向代理

以上就是我了解到的Nginx在不依赖第三方模块能处理的事情，下面详细说明每种功能怎么做

反向代理

反向代理应该是Nginx做的最多的一件事了，什么是反向代理呢，以下是百度百科的说法：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。

简单来说就是真实的服务器不能直接被外部网络访问，所以需要一台代理服务器，而代理服务器能被外部网络访问的同时又跟真实服务器在同一个网络环境，当然也可能是同一台服务器，端口不同而已。下面贴上一段简单的实现反向代理的代码

```
server {
    listen       80;
    server_name  localhost;
    client_max_body_size 1024M;

    location / {
       proxy_pass http://localhost:8080;
       proxy_set_header Host $host:$server_port;
    }
 }
```

保存配置文件后启动Nginx，这样当我们访问localhost的时候，就相当于访问localhost:8080了

负载均衡

负载均衡也是Nginx常用的一个功能，负载均衡其意思就是分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。

简单而言就是当有2台或以上服务器时，根据规则随机的将请求分发到指定的服务器上处理，负载均衡配置一般都需要同时配置反向代理，通过反向代理跳转到负载均衡。而Nginx目前支持自带3种负载均衡策略，还有2种常用的第三方策略。

1、RR（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

简单配置

```
upstream test {
    server localhost:8080;
    server localhost:8081;
  }
  server {
    listen    81;
    server_name  localhost;
    client_max_body_size 1024M;

    location / {
      proxy_pass http://test;
      proxy_set_header Host $host:$server_port;
    }
  }
```

负载均衡的核心代码为

```
upstream test {
    server localhost:8080;
    server localhost:8081;
  }
```

这里我配置了2台服务器，当然实际上是一台，只是端口不一样而已，而8081的服务器是不存在的,也就是说访问不到，但是我们访问http://localhost 的时候,也不会有问题，会默认跳转到http://localhost:8080

具体是因为Nginx会自动判断服务器的状态，如果服务器处于不能访问（服务器挂了），就不会跳转到这台服务器，所以也避免了一台服务器挂了影响使用的情况，由于Nginx默认是RR策略，所以我们不需要其他更多的设置。

2、权重

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。例如

```
upstream test {
    server localhost:8080 weight=9;
    server localhost:8081 weight=1;
  }
```

那么10次一般只会有1次会访问到8081，而有9次会访问到8080

3、ip_hash

上面的2种方式都有一个问题，那就是下一个请求来的时候请求可能分发到另外一个服务器，当我们的程序不是无状态的时候（采用了session保存数据），这时候就有一个很大的很问题了，比如把登录信息保存到了session中，那么跳转到另外一台服务器的时候就需要重新登录了，所以很多时候我们需要一个客户只访问一个服务器，那么就需要用iphash了，iphash的每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

```
upstream test {
    ip_hash;
    server localhost:8080;
    server localhost:8081;
  }
```

4、fair（第三方）

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。

```
upstream backend {
    fair;
    server localhost:8080;
    server localhost:8081;
  }
```

5、url_hash（第三方）

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的hash算法

```
upstream backend {
    hash $request_uri;
    hash_method crc32;
    server localhost:8080;
    server localhost:8081;
  }
```

以上5种负载均衡各自适用不同情况下使用，所以可以根据实际情况选择使用哪种策略模式,不过fair和url_hash需要安装第三方模块才能使用，由于本文主要介绍Nginx能做的事情，所以Nginx安装第三方模块不会再本文介绍

HTTP服务器

Nginx本身也是一个静态资源的服务器，当只有静态资源的时候，就可以使用Nginx来做服务器，同时现在也很流行动静分离，就可以通过Nginx来实现，首先看看Nginx做静态资源服务器

```
server {
    listen       80;
    server_name  localhost;
    client_max_body_size 1024M;


    location / {
        root   e:\wwwroot;
        index  index.html;
     }
  }
```

这样如果访问http://localhost 就会默认访问到E盘wwwroot目录下面的index.html，如果一个网站只是静态页面的话，那么就可以通过这种方式来实现部署。

动静分离

动静分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路

```
upstream test{
    server localhost:8080;
    server localhost:8081;
  }

  server {
    listen     80;
    server_name  localhost;

    location / {
      root  e:\wwwroot;
      index  index.html;
    }

    # 所有静态请求都由nginx处理，存放目录为html
    location ~ \.(gif|jpg|jpeg|png|bmp|swf|css|js)$ {
      root   e:\wwwroot;
    }

    # 所有动态请求都转发给tomcat处理
    location ~ \.(jsp|do)$ {
      proxy_pass  http://test;
    }

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

这样我们就可以吧HTML以及图片和css以及js放到wwwroot目录下，而tomcat只负责处理jsp和请求，例如当我们后缀为gif的时候，Nginx默认会从wwwroot获取到当前请求的动态图文件返回，当然这里的静态文件跟Nginx是同一台服务器，我们也可以在另外一台服务器，然后通过反向代理和负载均衡配置过去就好了，只要搞清楚了最基本的流程，很多配置就很简单了，另外localtion后面其实是一个正则表达式，所以非常灵活

正向代理

正向代理，意思是一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端才能使用正向代理。

当你需要把你的服务器作为代理服务器的时候，可以用Nginx来实现正向代理，但是目前Nginx有一个问题，那么就是不支持HTTPS，虽然我百度到过配置HTTPS的正向代理，但是到最后发现还是代理不了，当然可能是我配置的不对，所以也希望有知道正确方法的同志们留言说明一下。

```
resolver 114.114.114.114 8.8.8.8;
  server {

    resolver_timeout 5s;

    listen 81;

    access_log  e:\wwwroot\proxy.access.log;
    error_log   e:\wwwroot\proxy.error.log;

    location / {
      proxy_pass http://$host$request_uri;
    }
  }
```

resolver是配置正向代理的DNS服务器，listen 是正向代理的端口，配置好了就可以在ie上面或者其他代理插件上面使用服务器

ip+端口号进行代理了。

最后说两句

Nginx是支持热启动的，也就是说当我们修改配置文件后，不用关闭Nginx，就可以实现让配置生效，当然我并不知道多少人知道这个，反正我一开始并不知道，导致经常杀死了Nginx线程再来启动……Nginx重新读取配置的命令是

```
nginx -s reload
```

windows下面就是

```
nginx.exe -s reload
```

- END -

推荐阅读

1. Github标星10.8K！Java 实战博客项目分享
2. 浅析VO、DTO、DO、PO的概念、区别和用处
3. 为什么年终奖是一个彻头彻尾的职场圈套？
4. 什么是一致性 Hash 算法？
5. 团队开发中 Git 最佳实践

后端实践：Nginx日志配置（超详细）

Java后端 2019-11-14

作者 | antwang

链接 | juejin.im/post/5aa09bb3f265da238f121b6c

前言

Nginx日志对于统计、系统服务排错很有用。

Nginx日志主要分为两种：access_log(访问日志)和error_log(错误日志)。通过访问日志我们可以得到用户的IP地址、浏览器的信息，请求的处理时间等信息。错误日志记录了访问出错的信息，可以帮助我们定位错误的原因。

本文将详细描述一下如何配置Nginx日志。

设置access_log

访问日志主要记录客户端的请求。客户端向Nginx服务器发起的每一次请求都记录在这里。客户端IP，浏览器信息，referer，请求处理时间，请求URL等都可以在访问日志中得到。当然具体要记录哪些信息，你可以通过log_format指令定义。

语法

```
access_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]]; # 设置访问日志
access_log off; # 关闭访问日志
```

- path 指定日志的存放位置。
- format 指定日志的格式。默认使用预定义的combined。
- buffer 用来指定日志写入时的缓存大小。默认是64k。
- gzip 日志写入前先进行压缩。压缩率可以指定，从1到9数值越大压缩比越高，同时压缩的速度也越慢。默认是1。
- flush 设置缓存的有效时间。如果超过flush指定的时间，缓存中的内容将被清空。
- if 条件判断。如果指定的条件计算为0或空字符串，那么该请求不会写入日志。

另外，还有一个特殊的值off。如果指定了该值，当前作用域下的所有的请求日志都被关闭。

作用域

可以应用access_log指令的作用域分别有http，server，location，limit_except。也就是说，在这几个作用域外使用该指令，Nginx会报错。

以上是access_log指令的基本语法和参数的含义。下面我们看一几个例子加深一下理解。

基本用法

```
access_log /var/logs/nginx-access.log
```

该例子指定日志的写入路径为/var/logs/nginx-access.log，日志格式使用默认的combined。

```
access_log /var/logs/nginx-access.log buffer=32k gzip flush=1m
```

该例子指定日志的写入路径为/var/logs/nginx-access.log，日志格式使用默认的combined，指定日志的缓存大小为32k，日志写入前启用gzip进行压缩，压缩比使用默认值1，缓存数据有效时间为1分钟。

使用log_format自定义日志格式

Nginx预定义了名为combined日志格式，如果没有明确指定日志格式默认使用该格式：

```
log_format combined '$remote_addr - $remote_user [$time_local] '
            '"$request" $status $body_bytes_sent '
            '"$http_referer" "$http_user_agent"';
```

如果不想使用Nginx预定义的格式，可以通过log_format指令来自定义。

语法

```
log_format name [escape=default|json] string ...;
```

- name 格式名称。在access_log指令中引用。
- escape 设置变量中的字符编码方式是json还是default，默认是default。
- string 要定义的日志格式内容。该参数可以有多个。参数中可以使用Nginx变量。

下面是log_format指令中常用的一些变量：

变量	含义
$bytes_sent	发送给客户端的总字节数
$body_bytes_sent	发送给客户端的字节数，不包括响应头的大小
$connection	连接序列号
$connection_requests	当前通过连接发出的请求数量
$msec	日志写入时间，单位为秒，精度是毫秒
$pipe	如果请求是通过http流水线发送，则其值为"p"，否则为"."
$request_length	请求长度（包括请求行，请求头和请求体）
$request_time	请求处理时长，单位为秒，精度为毫秒，从读入客户端的第一个字节开始，直到把最后一个字符发送张客户端进行日志写入为止
$status	响应状态码

$time_iso8601	标准格式的本地时间,形如"2017-05-24T18:31:27+08:00"
$time_local	通用日志格式下的本地时间，如"24/May/2017:18:31:27 +0800"
$http_referer	请求的referer地址。
$http_user_agent	客户端浏览器信息。
$remote_addr	客户端IP
$http_x_forwarded_for	当前端有代理服务器时，设置web节点记录客户端地址的配置，此参数生效的前提是代理服务器也要进行相关的x_forwarded_for设置。
$request	完整的原始请求行，如 "GET / HTTP/1.1"
$remote_user	客户端用户名称，针对启用了用户认证的请求
$request_uri	完整的请求地址，如 "https://daojia.com/"

看不清可以点开放大

下面演示一下自定义日志格式的使用：

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

我们使用log_format指令定义了一个main的格式，并在access_log指令中引用了它。假如客户端有发起请求：https://suyunfe.com/，我们看一下我截取的一个请求的日志记录:

```
112.195.209.90 - - [20/Feb/2018:12:12:14 +0800]
"GET / HTTP/1.1" 200 190 "-" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Mobile Safari/537.36" "-"
```

我们看到最终的日志记录中 `$remote_user` 、 `$http_referer` 、 `$http_x_forwarded_for` 都对应了一个 `-` ，这是因为这几个变量为空。

设置error_log

错误日志在Nginx中是通过error_log指令实现的。该指令记录服务器和请求处理过程中的错误信息。

语法

配置错误日志文件的路径和日志级别。

```
error_log file [level];
Default:
error_log logs/error.log error;
```

第一个参数指定日志的写入位置。

第二个参数指定日志的级别。level可以是debug, info, notice, warn, error, crit, alert,emerg中的任意值。可以看到其取值范围是按紧急程度从低到高排列的。只有日志的错误级别等于或高于level指定的值才会写入错误日志中。默认值是error。

基本用法

```
error_log /var/logs/nginx/nginx-error.log
```

它可以配置在：main， http, mail, stream, server, location作用域。

例子中指定了错误日志的路径为： `/var/logs/nginx/nginx-error.log` ，日志级别使用默认的error。

open_log_file_cache

每一条日志记录的写入都是先打开文件再写入记录，然后关闭日志文件。如果你的日志文件路径中使用了变量，如 `access_log /var/logs/$host/nginx-access.log` ，为提高性能，可以使用open_log_file_cache指令设置日志文件描述符的缓存。

语法

```
open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time];
```

> - max 设置缓存中最多容纳的文件描述符数量，如果被占满，采用LRU算法将描述符关闭。
> - inactive 设置缓存存活时间，默认是10s。
> - min_uses 在inactive时间段内，日志文件最少使用几次，该日志文件描述符记入缓存，默认是1次。
> - valid：设置多久对日志文件名进行检查，看是否发生变化，默认是60s。
> - off：不使用缓存。默认为off。

基本用法

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

它可以配置在http、server、location作用域中。

例子中，设置缓存最多缓存1000个日志文件描述符，20s内如果缓存中的日志文件描述符至少被被访问2次，才不会被缓存关闭。每隔1分钟检查缓存中的文件描述符的文件名是否还存在。

总结

Nginx中通过access_log和error_log指令配置访问日志和错误日志，通过log_format我们可以自定义日志格式。如果日志文件路径中使用了变量，我们可以通过open_log_file_cache指令来设置缓存，提升性能。

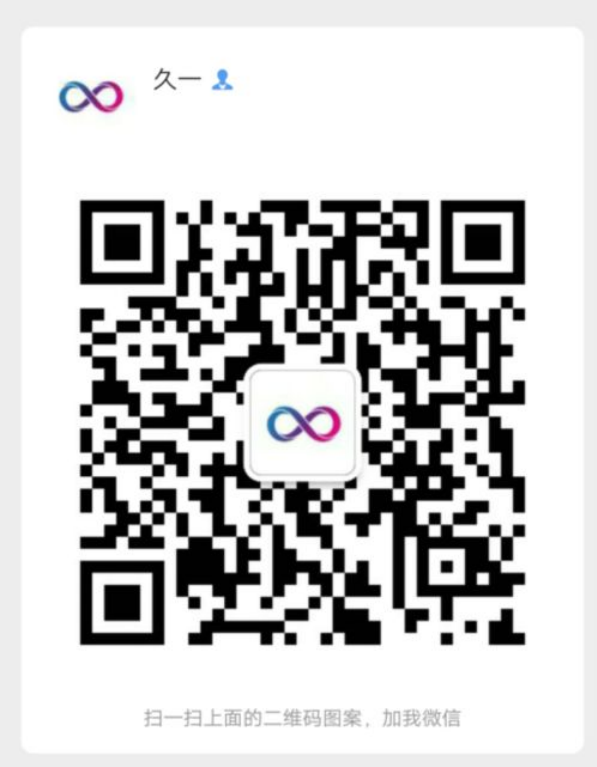
另外，在access_log和log_format中使用了很多变量，这些变量没有一一列举出来，详细的变量信息可以参考Nginx官方文档：

> http://nginx.org/en/docs/varindex.html

-END-

即可进入无广告交流群。

↓扫描二维码进群↓



推荐阅读

1. SSM 实现支付宝扫码支付功能(图文详解)
2. Spring Boot 微信点餐系统
3. Spring MVC 到 Spring Boot 的简化之路
4. 12306 的架构到底有多牛逼?
5. 团队开发中 Git 最佳实践

喜欢文章，点个在看

图解 Nginx 限流配置

程序员赵鑫 Java后端 2019-09-06

作者 | 程序员赵鑫

www.cnblogs.com/xinzhao/p/11465297.html

本文以示例的形式，由浅入深讲解Nginx限流相关配置，是对简略的官方文档的积极补充。Nginx限流使用的是leaky bucket算法，如对算法感兴趣，可移步维基百科先行阅读。不过不了解此算法，不影响阅读本文。

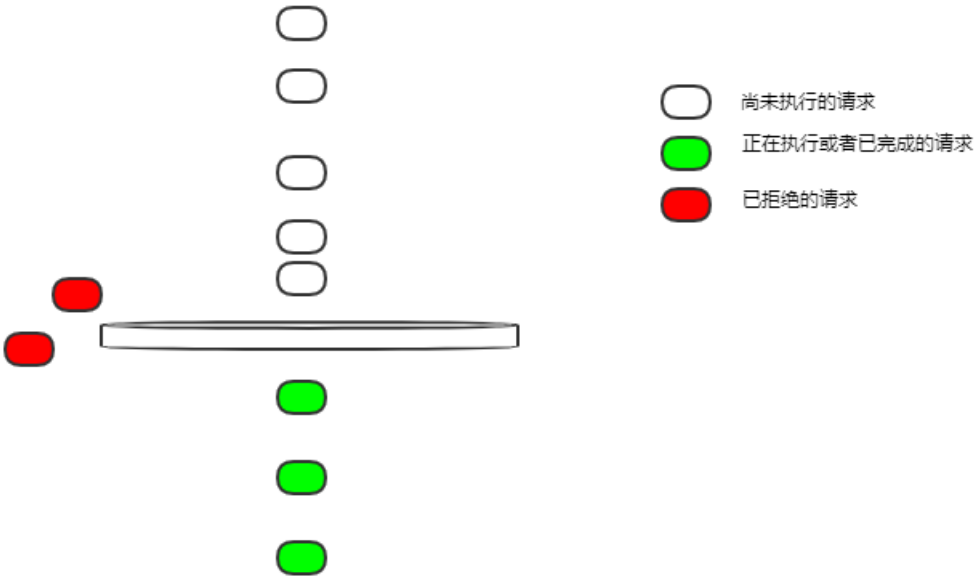
空桶

我们从最简单的限流配置开始：

```
limit_req_zone $binary_remote_addr zone=ip_limit:10m rate=10r/s;

server {
    location /login/ {
        limit_req zone=ip_limit;
        proxy_pass http://login_upstream;
    }
}
```

- $binary_remote_addr 针对客户端ip限流；
- zone=ip_limit:10m 限流规则名称为ip_limit，允许使用10MB的内存空间来记录ip对应的限流状态；
- rate=10r/s 限流速度为每秒10次请求
- location /login/ 对登录进行限流

限流速度为每秒10次请求，如果有10次请求同时到达一个空闲的nginx，他们都能得到执行吗？



漏桶漏出请求是匀速的。10r/s是怎样匀速的呢？每100ms漏出一个请求。

在这样的配置下，桶是空的，所有不能实时漏出的请求，都会被拒绝掉。

所以如果10次请求同时到达，那么只有一个请求能够得到执行，其它的，都会被拒绝。

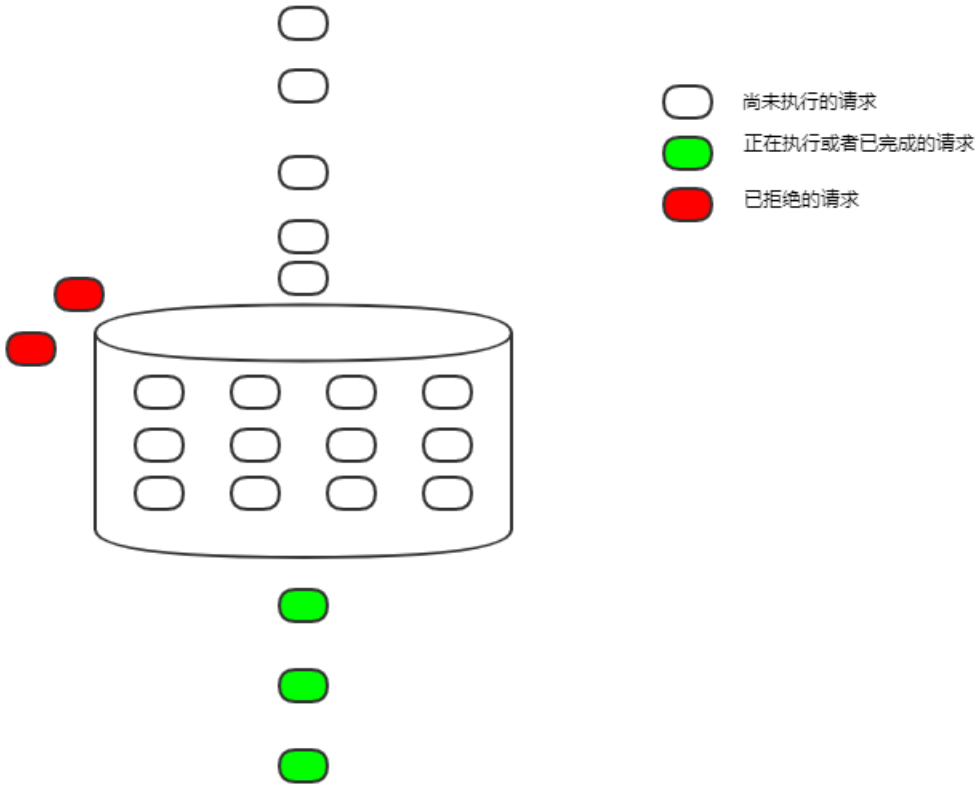
这不太友好，大部分业务场景下我们希望这10个请求都能得到执行。

Burst

我们把配置改一下，解决上一节的问题

```
limit_req_zone $binary_remote_addr zone=ip_limit:10m rate=10r/s;

server {
    location /login/ {
        limit_req zone=ip_limit burst=12
;
        proxy_pass http://login_upstream;
    }
}
```

- burst=12 漏桶的大小设置为12



逻辑上叫漏桶，实现起来是FIFO队列，把得不到执行的请求暂时缓存起来。

这样漏出的速度仍然是100ms一个请求，但并发而来，暂时得不到执行的请求，可以先缓存起来。只有当队列满了的时候，才会拒绝接受新请求。

这样漏桶在限流的同时，也起到了削峰填谷的作用。

在这样的配置下，如果有10次请求同时到达，它们会依次执行，每100ms执行1个。

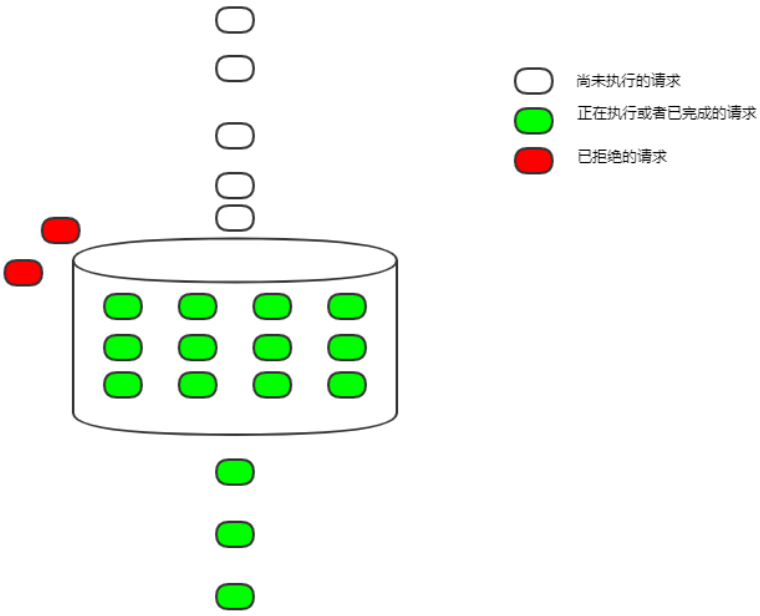
虽然得到执行了，但因为排队执行，延迟大大增加，在很多场景下仍然是不能接受的。

NoDelay

继续修改配置，解决Delay太久导致延迟增加的问题

```
limit_req_zone $binary_remote_addr zone=ip_limit:10m rate=10r/s;

server {
    location /login/ {
        limit_req zone=ip_limit burst=12 nodelay
;
        proxy_pass http://login_upstream;
    }
}
```

nodelay 把开始执行请求的时间提前，以前是delay到从桶里漏出来才执行，现在不delay了，只要入桶就开始执行



要么立刻执行，要么被拒绝，请求不会因为限流而增加延迟了。

因为请求从桶里漏出来还是匀速的，桶的空间又是固定的，最终平均下来，还是每秒执行了5次请求，限流的目的还是达到了。

但这样也有缺点，限流是限了，但是限得不那么匀速。以上面的配置举例，如果有12个请求同时到达，那么这12个请求都能够立刻执行，然后后面的请求只能匀速进桶，100ms执行1个。如果有一段时间没有请求，桶空了，那么又可能出现并发的12个请求一起执行。

大部分情况下，这种限流不匀速，不算是大问题。不过nginx也提供了一个参数才控制并发执行也就是nodelay的请求的数量。

```
limit_req_zone $binary_remote_addr zone=ip_limit:10m rate=10r/s;

server {
    location /login/ {
        limit_req zone=ip_limit burst=12 delay=4
;
        proxy_pass http://login_upstream;
    }
}
```

delay=4 从桶内第5个请求开始delay

这样通过控制delay参数的值，可以调整允许并发执行的请求的数量，使得请求变的均匀起来，在有些耗资源的服务上控制这个数量，还是有必要的。

如果喜欢本篇文章，欢迎转发、点赞。关注订阅号「Web项目聚集地」，回复「进群」即可进入无广告技术交流。

推荐阅读

1. 基于 Spring Boot 的 Restful 风格实现增删改查
2. 如何使用牛逼的插件帮你规范代码
3. IntelliJ IDEA 构建maven多模块工程项目
4. 别在 Java 代码里乱打日志了，这才是正确姿势
5. 挑战 10 道超难 Java 面试题
6. 什么时候进行分库分表？

Web项目聚集地

微信扫描二维码，关注我的公众号

喜欢文章，点个在看

阅读原文

除了负载均衡，Nginx 还可以做很多

Java后端 2019-09-01

Nginx应该是现在最火的web和反向代理服务器，没有之一。她是一款诞生于俄罗斯的高性能web服务器，尤其在高并发情况下，相较Apache，有优异的表现。

那除了负载均衡，她还有什么其他的用途呢，下面我们来看下。

一、静态代理

Nginx擅长处理静态文件，是非常好的图片、文件服务器。把所有的静态资源的放到nginx上，可以使应用动静分离，性能更好。

二、负载均衡

Nginx通过反向代理可以实现服务的负载均衡，避免了服务器单节点故障，把请求按照一定的策略转发到不同的服务器上，达到负载的效果。

```
#access_log  logs/host.access.log  main;

upstream serviceCluster{
    server 10.12.66.186:9431;
    server 10.12.66.187:9431;
}

upstream service2Cluster{
    server 10.12.66.186:9432 weight=1;
    server 10.12.66.187:9432 weight=2;
    server 10.12.66.188:9432 weight=2;
}

upstream service3Cluster{
    server 10.12.66.186:9433;
    server 10.12.66.187:9433;
}
```

常用的负载均衡策略有：

1、轮询

将请求按顺序轮流地分配到后端服务器上，它均衡地对待后端的每一台服务器，而不关心服务器实际的连接数和当前的系统负载。

2、加权轮询

不同的后端服务器可能机器的配置和当前系统的负载并不相同，因此它们的抗压能力也不相同。

给配置高、负载低的机器配置更高的权重，让其处理更多的请；而配置低、负载高的机器，给其分配较低的权重，降低其系统负载，加权轮询能很好地处理这一问题，并将请求顺序且按照权重分配到后端。

3、ip_hash（源地址哈希法）

根据获取客户端的IP地址，通过哈希函数计算得到一个数值，用该数值对服务器列表的大小进行取模运算，得到的结果便是客户端要访问服务器的序号。

采用源地址哈希法进行负载均衡，同一IP地址的客户端，当后端服务器列表不变时，它每次都会映射到同一台后端服务器进行访问。

4、随机

通过系统的随机算法，根据后端服务器的列表大小值来随机选取其中的一台服务器进行访问。

5、least_conn（最小连接数法）

由于后端服务器的配置不尽相同，对于请求的处理有快有慢，最小连接数法根据后端服务器当前的连接情况，动态地选取其中当前积压连接数最少的一台服务器来处理当前的请求，尽可能地提高后端服务的利用效率，将负责合理地分流到每一台服务器。

三、限流

Nginx的限流模块，是基于漏桶算法实现的，在高并发的场景下非常实用，如下图：

```
    server 10.12.66.187:9433;
}

limit_req_zone $binary_remote_addr zone=mylimit:10m rate=100r/s;

location /service {
    limit_req zone = mylimit burst=20 nodelay;
    proxy_pass http://serviceCluster;
}

location /service2 {
    limit_req zone = mylimit burst=20 nodelay;
    proxy_pass http://service2Cluster;
}

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

1、配置参数

1）limit_req_zone定义在http块中，$binary_remote_addr 表示保存客户端IP地址的二进制形式。

2）Zone定义IP状态及URL访问频率的共享内存区域。

zone=keyword标识区域的名字，以及冒号后面跟区域大小。16000个IP地址的状态信息约1MB，所以示例中区域可以存储160000个IP地址。

3）Rate定义最大请求速率。示例中速率不能超过每秒100个请求。

2、设置限流

burst排队大小，nodelay不限制单个请求间的时间。

四、缓存

1、浏览器缓存，静态资源缓存用expire。

```
location ~ .*\.(?:jpg|jpeg|gif|png|ico|cur|gz|svg|svgz|mp4|ogg|ogv|webm)$ {
    expires      7d;
}

location ~ .*\.(?:js|css)$ {
    expires      7d;
}
```

2、代理层缓存

```
proxy_cache_path /data/cache/nginx/ levels=1:2 keys_zone=cache:512m inactive = 1d max_size=8g;

location / {
    location ~ \.(htm|html)?$ {
        proxy_cache cache;
        proxy_cache_key    $uri$is_args$args;     //以此变量值做HASH，作为KEY
        add_header X-Cache $upstream_cache_status;
        proxy_cache_valid 200 10m;
        proxy_cache_valid any 1m;
        proxy_pass  http://real_server;
        proxy_redirect     off;
    }
    location ~ .*\.(gif|jpg|jpeg|bmp|png|ico|txt|js|css)$ {
        root /data/webapps/edc;
        expires      3d;
        add_header Static Nginx-Proxy;
    }
}
```

五、黑白名单

1、不限流白名单

```
geo $limit {
    122.16.11.0/24  0;
}                                   白名单

map $limit $limit_key {
    1 $binary_remote_addr;
    0 "";
}

limit_req_zone $limit_key zone=mylimit:10m rate=1r/s;

location / {
        limit_req zone=mylimit burst=1 nodelay;
        proxy_pass http://service3Cluster;
}
```

2、黑名单

```
location / {
    deny  10.52.119.21;
    deny 122.12.1.0/24;
    allow 10.1.1.0/16;
    allow 1001:0db8::/32;
    deny  all;
}
```

好了，上面就是nginx几个常用功能，静态分离、负载均衡、限流、缓存、黑白名单等，你都了解了吗？

作者：技术大咖秀

来源：

https://www.toutiao.com/i6692127248272589315/

如果喜欢本篇文章，欢迎转发、点赞。关注订阅号「Web项目聚集地」，回复「全栈」即可获取 2019 年最新 Java、Python、前端学习视频资源。

推荐阅读

1. 经常用 HashMap ?这 6 个问题回答下！
2. 数据库这么多锁，能锁住小姐姐吗?
3. 小白也能看懂，30 分钟搭建个人博客!
4. 快来薅当当的羊毛！
5. 聊一聊 Java 泛型中的通配符
6. 数据库不使用外键的 9 个理由