**nginx**

[第 1 章 部署基础知识 1](#_Toc975873)

[1.1 部署基础 1](#_Toc975874)

[1.1.1 项目生命周期 1](#_Toc975875)

[1.1.2 项目部署 1](#_Toc975876)

[第 2 章 Nginx进阶 2](#_Toc975877)

[2.1 Nginx快速入门 2](#_Toc975878)

[2.1.1 Nginx简介 2](#_Toc975879)

[2.1.2 Nginx部署 3](#_Toc975880)

[2.1.3 配置详解 4](#_Toc975881)

[2.2 Nginx进阶知识 8](#_Toc975882)

[2.2.1 反向代理 8](#_Toc975883)

[2.2.2 负载均衡 10](#_Toc975884)

[2.2.3 日志解析 13](#_Toc975885)

[2.2.4 URL重写 14](#_Toc975886)

**第 1 章 部署基础知识**

**1.1 部署基础**

**1.1.1 项目生命周期**

世间万物皆有其生命，软件项目也是如此。随着互联网的发展，软件项目的生命周期也发生了很大的变化，为了更好的让大家理解软件项目，项目生命周期有狭义[具体]、广义[缘起/缘灭]之分，我们一般所说的项目生命周期主要指的是狭义的项目生命周期，我们以传统的软件项目为例进行介绍。

**传统项目生命周期**

对于传统软件项目来说，它主要包含以下五个阶段：

调研阶段

目的：居安思危

人员：相关人员，侧重于产品经理

节点：多角度思路/方案，最后领导拍板

设计阶段

目的：方案可视化

人员：产品团队主导，开发、测试、运维参与

节点：产品需求文档、项目里程表

开发阶段

目的：方案运行

人员：开发团队为主，运维团队参与

节点：阶段项目正常运行

测试阶段

目的：保证项目功能完善

人员：测试团队为主，运维、开发参与

节点：项目功能符合要求

运营阶段

目的：项目部署+运行维护

人员：运营团队为主、开发/产品团队参与

节点：项目终止、功能迭代等

**运维团队全过程参与**

**1.1.2 项目部署**

对于一个软件项目的部署来说，我们主要考虑两方面：

部署流程--基于项目功能，设计合理的部署方案；部署项目--结合部署方案，实现顺畅的项目部署

我们在这一节主要关注部署流程。

关于部署流程，主要包含以下两个方面：部署方案、部署环境

**部署方案**

为了实现顺畅的项目部署，我们一般会在部署操作的时候，按照既定的部署方案有计划的实施下去。那么部署方案是怎么设计出来的呢？

1 分析项目的产品需求文档，获取项目的主旨，定好部署方案的方向

2 分析项目开发文档，按照功能边界，设计部署的结点

3 分析边界功能，调研功能软件，合理的取舍，选符合当前业务场景的

4 梳理项目部署涉及到的部署软件实现方案，根据2确定的结点，确定初版部署方案

5 根据项目实际情况，调整优化并确定项目部署方案

**部署环境**

根据我们对项目生命周期的理解，一个项目需要经历多个团队的工作，才能最终形成一个成品，也就是说开发好的项目需要在多个阶段环境中按照我们的部署方案部署项目，而每个阶段的环境都是有特殊作用的，那么接下来我们来学习一下项目发布过程中所涉及到的五个环境：

个人开发环境

工作人员：自己

工作平台：个人笔记本、公司配的电脑

平台特点：环境是自己配的，团队中不同的个人开发环境可以不一样

工作内容：项目的子模块，子功能

完成标准：完成领导安排的内容[项目的功能子模块开发]

公司开发环境

工作人员：开发团队

工作平台：公司内部服务器

平台特点：服务器环境和线上的服务器环境完全一致

工作内容：项目子模块间的功能联调

完成标准：项目阶段开发、调试完成

项目测试环境

工作人员：测试团队

工作平台：公司内部服务器

平台特点：服务器环境和线上的服务器环境完全一致

工作内容：项目功能/非功能/探索等测试

完成标准：项目阶段功能正常运行

项目预发布环境

工作人员：运维团队

工作平台：公司线上服务器组中的一台

平台特点：服务器环境和线上的服务器环境完全一致

工作内容：特殊功能测试(比如支付)、数据压力测试、其他安全测试等

完成标准：项目阶段功能正常运行，最后一道防线

项目线上环境

工作人员：运维团队

工作平台：公司线上服务器组

平台特点：标准线上的服务器环境

工作内容：代码部署和维护

完成标准：项目正常运行

**第 2 章 Nginx进阶**

**2.1 Nginx快速入门**

**2.1.1 Nginx简介**

Nginx(发音同 engine x)是一款基于异步框架的轻量级/高性能的Web 服务器/反向代理服务器/缓存服务器/电子邮件(IMAP/POP3)代理服务器，并在一个BSD-like 协议下发行。由俄罗斯的程序设计师Igor Sysoev(伊戈尔·赛索耶夫)所开发，最初供俄国大型网站Rambler.ru及搜寻引擎Rambler使用。

Nginx特点

优点：

高并发量：基于 epoll/kqueue 模型开发，支持高并发量，官方说其支持高达 5w 并发连接数的响应

内存消耗少：善于处理静态文件，相较于其他web(比如：apache),占用更少的内存及资源

简单稳定：配置简单(一个conf文件),运行简单(nginx命令)，而且运行稳定

模块化程度高：功能模块插件化设计，可以自由配置相应的功能。

支持Rwrite重写规则：能够根据域名、URL等请求关键点，实现定制化的高质量分发。

低成本：Nginx的负载均衡功能很强大而且免费开源，相较于几十万的硬件负载均衡器成本相当低。

支持多系统：Nginx代码完全用C语言从头写成，可以在各系统上编译并使用。

缺点：

动态处理差：nginx善于处理静态文件，但是处理动态页面相较于Apache之类重量级的web软件能力稍欠缺。

rewrite弱：虽然nginx支持rewrite功能多，但是相较于Apache之类重量级的web软件能力稍欠缺。

**2.1.2 Nginx部署**

安装Nginx软件

apt-get install -y build-essential libssl-dev libtool libpcre3 libpcre3-dev make openssl zlib1g-dev

apt-get install nginx -y

检查效果：

sudo netstat -tnulp | grep nginx

在浏览器中通过指定网址查看

服务相关命令

启动：sudo nginx

停止：sudo nginx -s stop

平滑重启：sudo nginx -s reload

查看版本：nginx -V

查看和nginx相关的软件

dpkg --get-selections|grep nginx

移除nginx，包括相关文件

apt-get --purge remove nginx

apt-get --purge remove nginx-common

apt-get --purge remove nginx-core

**Nginx配置简介**

nginx软件目录：

配置目录：/etc/nginx

执行文件: /usr/sbin/nginx

日志目录：/var/log/nginx

启动文件：/etc/init.d/nginx

web目录：/var/www/html/，首页文件是index.nginx-debian.html

 /usr/share/nginx/html/ 首页文件是index.html

nginx配置文件：

默认文件：

/etc/nginx/nginx.conf

其他目录：

/etc/nginx/{sites-available/sites-enabled/conf.d}

文件结构：

全局配置段

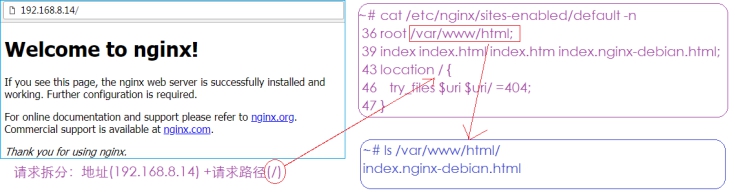
http配置段

server配置段 项目或者应用

**location配置段** url配置



**Nginx访问原理**



**2.1.3 配置详解**

**全局配置段**

主要是全局性的和服务级别的属性配置，常见的主要有以下几种设置：

user 设置使用用户(worker)

worker\_processes 进行增大并发连接数的处理 跟cpu保持一致 八核设置八个

error\_log nginx的错误日志

pid   nginx服务启动时候pid

events 定义事件相关的属性

  worker\_connections 一个进程允许处理的最大连接数

**http配置段**

主要配置server通用的一些配置

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

sendfile on; # 开启高效文件传输模式。

autoindex on; # 开启目录列表访问，合适下载服务器，默认关闭。

tcp\_nopush on; # 防止网络阻塞

tcp\_nodelay on; # 防止网络阻塞

keepalive\_timeout 120; # 长连接超时时间，单位是秒

gzip on; # 开启gzip压缩输出

**Server常见配置属性**

常见样式

server {

    listen 端口;

    server\_name 主机名;

    ...

}

server配置段最重要的属性是listen和server\_name。它们都是用于匹配并处理请求的。

listen属性

作用：定义Server监听的ip和port，当ip/port匹配时候才进行下一步匹配

表现形式：

|  |  |  |  |
| --- | --- | --- | --- |
| 形式 | 描述 | 示例 | 完整示例 |
| IP:Port | 地址精确表示样式 | listen 10.10.10.10:99 | listen 10.10.10.10:99 |
| IP | 自动监听 IP:80地址 | listen 10.10.10.10 | listen 10.10.10.10:80 |
| Port | 自动监听 全地址:Port | listen 99或 [::]:99 | listen 0.0.0.0:99 |
| default\_server | 自动使用默认的地址 | listen default\_server | listen localhost:80 |

使用原则：

首先将所有样式补全成IP:Port,然后匹配,匹配Server多，那么接着使用Server\_name匹配

server\_name属性

作用：定义Server监听的域名，当域名匹配时候才进行下一步操作

表现形式：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 格式 | 完整样式 | 前缀正则样式 | 后缀正则样式 | 禁止非法域名或IP |
| 形式 | www.example.com | \*.example.com | www.example.\* | \_ |

使用原则：

优先使用完整样式,然后使用前缀正则样式,最后使用后缀正则样式,如果正则样式相同的时候,匹配最长,否则就走非法规则。

非法域名/IP，表示请求到该主机上一个不存在的IP或者域名

root属性

作用：定义Server相应请求的html文件所在路径

表现形式：

root /var/www/html;

index属性

作用：定义响应请求后返回的文件名称或格式

表现形式：

index index.html index.htm index.nginx-debian.html;

return属性

作用：定义响应请求后返回的http状态码

表现形式：

return       444;

**location常见配置属性**

location主要是根据Server匹配到的请求路径和关键字去响应和处理。

语法：

location optional\_modifier location\_match { ... }

location @name { ... }

其中：optional\_modifier是匹配条件，location\_match是匹配的样式，{}是要执行的操作。匹配条件主要有两种:正则/前缀字符。

匹配规则

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 含义 | 匹配方式 | 优先级 | 样式 |
| =/路径 | 精确匹配 | 前缀 | 1 | location = /image {} |
| ^~ | 优先匹配 | 前缀 | 2 | location ^~ /page {} |
| ~|!~ | 普通正则-敏感|不敏感 | 正则符号 | 3 | location ~ .(jpe?g)$ {} |
| @ | 内部重定向 | 前缀 |  | location @name {} |
| /路径 | 通用匹配 | 前缀 | 4 | location / {} |

使用原则：

前提：根据请求url，获取uri即除了域名/IP之外的部分，用于location匹配

如果有精确匹配，即 =/路径，找到匹配项后，结束匹配。

location = 路径 {}  或者  location 完整路径 {}

如果有优先匹配，即 ^~，找到匹配项后，结束匹配。

location ^~ 路径

如果有正则匹配，即 ~|!~|~\*|!~\*，找到匹配项后，不会终止继续匹配，直到找到合适的

location ~\* 正则字符 {}

如果匹配到多个，则使用location\_match最长的。

匹配示例

常见示例：

location = / {

   #精确规则A

}

location ^~ /static/ {

   #优先规则B

}

location ~ \.(gif|jpg|png|js|css)$ {

    #正则规则C

}

location ~\* \.png$ {

    #正则规则D

}

location / {

#通用规则E

}

访问效果如下：

访问根目录/， 比如http://a.com/ 将匹配规则A

访问 http://a.com/static/a.html 将匹配规则B

访问 http://a.com/a.gif, http://a.com/b.png 规则C和D均适合，按顺序优先使用规则C

访问 http://a.com/a.PNG 则匹配规则D，因为规则D不区分大小写

访问 http://a.com/category/id/1111 则最终匹配到规则E

@name示例

@用来定义一个命名location。主要用于内部重定向，不能用来处理正常的请求。其用法如下：

location / {

    try\_files $uri $uri/ @custom

}

location @custom {

    # ...do something

    # custom 命名的 location中不能再嵌套其它的命名location

}

关于URL尾部的/有如下注意事项：

1 location中的location\_match字符有无"/"不受影响。/user/等同/user。

2 对于访问网站域名(http://sswang.com/)，尾部有无"/"不受影响。因为浏览器会自动补全"/"。

3 对于访问网站域名后面的路径(http://sswang.com/other/)。尾部的"/"很重要。

URL尾部的"/"表示目录，没有"/"表示文件，而且文件找不到的话，会发生重定向。

/other/：表示服务器会自动去该目录下找对应的默认文件。

/other：表示服务器会先去找other文件，找不到的话会将other当成目录，重定向到/other/，去该目录下找默认文件。

**location常见动作：**

在location内部常用的功能属性非常多，常见的基本属性、临时跳转、访问控制、目录列表等。

基本属性

location / {

     root   /var/www/html; # 指定响应请求的文件所在路径

     index  index.php index.html index.htm; # 指定响应请求的默认文件名称

     expires 7d; # 指定响应请求的文件过期时间，一般用于静态文件

     **try\_files $uri $uri/ =404;** # 如果root指定的路径下有查找的文件，就返回，否则报错

}

临时跳转

location = /test/ {

    return 302 http://sswang.com/; # 访问旧url的时候，临时跳转到新url，两个url均不失效

}

目录列表

location /upload {

    alias   /var/www/upload; # 指定查看文件列表路径(绝对路径)

    autoindex on;              # 开启目录自动索引

    autoindex\_exact\_size off; # 默认on，显示文件确切大小(bytes)。off表示显示文件的大概大小(kB/MB/...)

    autoindex\_localtime on;    # 默认off，显示的文件时间为GMT时间。on表示显示文件的服务器时间

}

注意：

  该alias指定的目录下，不允许出现index属性指定的文件

root VS alias

root 和 alias 所起的作用都是指定响应请求所用文件的路径，只是他们有些许的区别

  root 表示 location 匹配内容的相对路径

  alias 表示 一个绝对路径,而且必须以"/"结尾

一般情况下，在location /中配置root，在location /other中配置alias

效果一： 效果二：

location /img/ {                        location /img/ {

alias /var/www/image/;                 root /var/www/image;

}                                           }

效果一：访问http://localhost/img/，nginx找/var/www/image/目录下的文件

效果二：访问http://localhost/img/，nginx找/var/www/image/img/目录下的文件

**location核心动作**

Nginx的配置语法灵活，可控制度非常高。在0.7以后的版本中加入了一个try\_files指令，配合命名location，可以部分替代原本常用的rewrite配置方式，提高解析效率。

指令语法

try\_files file ... uri

try\_files file ... =code

作用：

响应时按顺序查找file，找到则返回file内容，否则的话进行内部重定向(uri)或返回状态码(code)。

常见示例

如果能找到指定的uri那么就返回相应的内容，否则的话返回错误状态码404

try\_files $uri $uri/ =404;

如果能找到指定的文件1/2.html那么就返回相应的内容，否则的话返回6.html文件内容

try\_files 1.html 2.html /6.html;

如果能找到指定的uri那么就返回相应的内容，否则的话就内部重定向到后端名称为@backup的location

try\_files $uri @backup;

注意：

如果最后一个地址是一个uri的话，那么这个uri必须是存在的，否则的话就出事了

**2.2 Nginx进阶知识**

**2.2.1 反向代理**

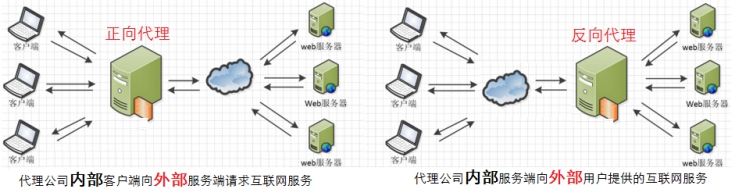
**代理是什么？**

简单来说，我找一个中间人，代替我去做一件事情，只要他给我结果就可以。

代理一般分为两种：正向代理、反向代理

**正向代理&反向代理**

示意图



区别

从用途上来讲：

        正向代理-为局域网客户端向外访问Internet服务。可以使用缓冲特性减少网络使用率。

反向代理-为局域网服务器向外提供Internet服务。可以使用负载平衡提高客户访问量。还可以基于高级URL策略和管理技术对服务进行高质量管控。

从安全性来讲：

正向代理-必须采取安全措施确保内网客户端通过它访问外部网站。隐藏客户端的身份

反向代理-对外提供服务是透明的，客户端并不知道自己访问的是一个代理。隐藏服务端的身份

**nginx 代理模块**

官方介绍

官方资料：http://www.nginx.cn/doc/standard/httpproxy.html

官方的代理属性很多，我们主要介绍proxy\_pass和proxy\_set\_header属性

官方代码示例

location / {

: proxy\_pass        http://localhost:8000; # 设定请求跳转后的地址，可以使用hostname或IP:Port形式

: proxy\_set\_header  X-Real-IP  $remote\_addr; # 后端请求携带原始请求的真实IP地址

}

属性详解：

proxy\_pass 指令设置被代理服务器的地址和被映射的URI，地址可以使用主机名或IP加端口号的形式

proxy\_pass关键点

proxy\_pass后面的路径最后的/作用很重要!!!

示例代码：

location /html/ {

　　1 proxy\_pass http://proxy.com;

    2 proxy\_pass http://proxy.com/;

}

假设我们访问的url是 http://domain.com/html/test.js，如何理解上述两种proxy\_pass的区别呢？

对于 1 来说 proxy.com后面没有"/"，表示"/html/" 请求(包括自己)后续的路径及其参数等关键字都由http://a.com/来处理，代理后的样式如下：

http://proxy.com/html/test.js

对于 2 来所 proxy.com后面有"/"，表示"/html/" 请求后续的路径及其参数等关键字都由http://a.com/来处理，代理后的样式如下：

http://proxy.com/test.js

**nginx代理实践**

代理的配置文件

~# vim /etc/nginx/conf.d/proxy.conf

server {

  listen 192.168.8.14:80;

  server\_name www.sswang.com;

  location / {

    proxy\_pass http://192.168.8.14:9999/hello/;

  }

}

后端服务配置文件

~# vim /etc/nginx/conf.d/hello.conf

server {

  listen 192.168.8.14:9999;

  location /hello/ {

    alias /var/www/html/hello/;

    try\_files $uri $uri/ =404;

  }

}

准备后端服务文件

mkdir -p /var/www/html/hello/

echo '<h1>proxy\_backend</h1>' > /var/www/html/hello/index.html

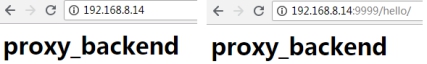
检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果



**2.2.2 负载均衡**

**负载均衡是什么？**

我们之前使用proxy\_pass的方式实现了nginx代理请求到后端的效果，随着我们的网站访问量越来越多，一个后端就不现实了，那么接下来我们应该如果在访问量日渐增大的情况下，满足线上业务的稳定呢？

解决方法就是：负载均衡

负载均衡简单说来人多力量大，打群架。

在nginx中的负载均衡主要有两种：四层负载(IP:Port)、七层负载(http://xxx)

**nginx upstream模块**

官方介绍

官方资料：http://www.nginx.cn/doc/standard/httpupstream.html

官方的代理属性很多，我们主要介绍upstream和ip\_hash属性

官方代码示例

upstream backend  {

  server backend1.example.com weight=5;

  server backend2.example.com:8080;

  server unix:/tmp/backend3;

}

server {

  location / {

    proxy\_pass  http://backend;

  }

}

属性详解：

upstream 主要是定义一个后端服务地址的集合列表，每个后端服务使用一个server命令表示

upstream {} 和 Server {} 两部分内容属于平级关系。

后端服务状态

在upstream模块中，可以使用server命令指定后端服务器的地址，同时还可以设置后端服务器在负载均衡调度中的状态，常用的状态有以下几种：

down： 表示当前server主机暂时不参与负载均衡。

backup：后备主机，当所有非backup机器出现故障或者繁忙的时候，才会请求backup机器。

max\_fails：允许请求的最大失败数，默认为1，配合fail\_timeout一起使用

fail\_timeout：经历max\_fails次失败后，暂停服务的时间，默认为10s。

**nginx负载均衡实践**

负载均衡配置文件

~# vim /etc/nginx/conf.d/upstream.conf

upstream backends {

  server 192.168.8.14:10086;

  server 192.168.8.14:10087;

  server 192.168.8.14:10088;

}

server {

  listen 80;

  server\_name localhost;

  location / {

    proxy\_pass http://backends;

  }

}

后端代理配置文件

~# vim /etc/nginx/conf.d/backend.conf

server {

  listen 192.168.8.14:10086;

  location / {

    root /var/www/html/hello/;

    try\_files $uri $uri/ =404;

  }

}

server {

  listen 192.168.8.14:10087;

  location / {

    root /var/www/html/nihao/;

    try\_files $uri $uri/ =404;

  }

}

server {

  listen 192.168.8.14:10088;

  location / {

    root /var/www/html/huanying/;

    try\_files $uri $uri/ =404;

  }

}

准备后端服务文件

mkdir -p /var/www/html/hello/

echo '<h1>backend\_hello</h1>' > /var/www/html/hello/index.html

mkdir -p /var/www/html/nihao/

echo '<h1>backend\_nihao</h1>' > /var/www/html/nihao/index.html

mkdir -p /var/www/html/huanying/

echo '<h1>backend\_huanying</h1>' > /var/www/html/huanying/index.html

检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果

~# for i in {1..100};do curl http://192.168.8.14;done

<h1>backend\_hello</h1>

<h1>backend\_nihao</h1>

<h1>backend\_huanying</h1>

...

<h1>backend\_hello</h1>

<h1>backend\_nihao</h1>

<h1>backend\_huanying</h1>

**负载均衡调度算法**

官方资料：http://nginx.org/en/docs/http/ngx\_http\_upstream\_module.html#example

Nginx提供的负载均衡策略有两种：

内置策略：nginx自带的算法

雨露均沾型：轮训、加权轮训、哈希

定向服务型：ip\_hash、least\_conn、cookie、route、lean、

商业类型：ntlm、least\_time、queue、stick

扩展策略：各种结合业务场景自定义的算法或者第三方算法

自定义算法

第三方算法：**fair**、**url\_hash**

常用算法简介：

轮询(默认)：请求按顺序逐一分配到不同的后端服务器。

weight：指定轮询权重，值越大，分配到的几率就越高，适用于后端服务器性能不均衡情况。

ip\_hash：按访问IP的哈希结果分配请求，分配后访客访问固定后端服务器，有效的解决动态网页会话共享问题。

fair：基于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。

url\_hash：按访问URL的哈希结果分配请求，使同URL定向到同一台后端服务器，可提高后端缓存服务器的效率。

**加权轮训实践**

修改负载均衡配置文件

~# vim /etc/nginx/conf.d/upstream.conf

upstream backends {

  server 192.168.8.14:10086 backup;

  server 192.168.8.14:10087 weight=1;

  server 192.168.8.14:10088 weight=2;

}

...

检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果

~# for i in {1..100};do curl http://192.168.8.14;done

<h1>backend\_nihao</h1>

<h1>backend\_huanying</h1>

<h1>backend\_huanying</h1>

...

<h1>backend\_nihao</h1>

<h1>backend\_huanying</h1>

<h1>backend\_huanying</h1>

**ip\_hash实践**

修改负载均衡配置文件

~# vim /etc/nginx/conf.d/upstream.conf

upstream backends {

  ip\_hash;

  server 192.168.8.14:10086 ;

  ...

}

...

检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果

~# for i in {1..100};do curl http://192.168.8.14;done

<h1>backend\_huanying</h1>

...

<h1>backend\_huanying</h1>

**2.2.3 日志解析**

**日志功能简介**

日志简介

Nginx默认提供了两个日志文件 access.log和error.log，通过access.log可以得到用户请求的相关信息；通过error.log可以获取某个web服务故障或其性能瓶颈等信息。

而且nginx的日志支持定制化格式，这样我们就可以根据实际的业务情况更好的高效工作。最常见的场景就是获取客户端的IP，记录用户访问量。

官方介绍：http://nginx.org/en/docs/http/ngx\_http\_log\_module.html

基本配置

# cat /etc/nginx/nginx.conf -n

    40 access\_log /var/log/nginx/access.log;

    41 error\_log /var/log/nginx/error.log;

注意：

  nginx日志属性设置的完整格式是：

  属性名称 access\_log

  存储位置 /var/log/nginx/access.log

   日志格式 位置为空表示使用默认的combined 日志格式。它是通过log\_format设置的

默认日志格式

log\_format combined '$remote\_addr - $remote\_user [$time\_local] '

                    '"$request" $status $body\_bytes\_sent '

                    '"$http\_referer" "$http\_user\_agent"';

注意：

log\_format是有一批nginx内置变量组合而成的。

日志样式：

# tail /var/log/nginx/access.log

192.168.8.14 - - [12/Nov/2018:08:24:18 -0800] "GET /favicon.ico HTTP/1.0" 404 580 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36"

**nginx常用内置变量**

nginx常用的内置变量主要是用来分析日志中的http记录的，我们可以根据内置的变量精确的获取相关的信息

默认变量

$remote\_addr 前一台主机的ip地址，不一定是真实的客户端IP

$remote\_user 用于记录远程客户端的用户名称（一般为“-”）

$time\_local 用于记录访问时间和时区

$request 用于记录请求的url以及请求方法

$status 响应状态码，例如：200成功、404页面找不到等。

$body\_bytes\_sent 给客户端发送的文件主体内容字节数

$http\_referer 可以记录用户是从哪个链接访问过来的

$http\_user\_agent 用户所使用的代理（一般为浏览器）

其他常用变量

$request\_uri 包含请求参数的原始URI，不包含主机名

$uri 不带请求参数的当前URI，不包含主机名

$http\_x\_forwarded\_for 可以记录客户端IP，通过代理服务器来记录客户端的ip地址

$http\_x\_real\_ip 可以记录客户端IP，通过代理服务器来记录客户端的ip地址

$args 这个变量等于请求行中的参数，同$query\_string

$host 请求主机头字段，否则为服务器名称。

$scheme HTTP方法（如http，https）

$document\_uri 与$uri相同

$document\_root 当前请求文件配置文件中html的根目录即root值

$request\_filename 当前请求的文件路径，由root或alias指令与URI请求生成

示例：

例：http://localhost:10086/sswang1/sswang2/test.txt

$host localhost

$server\_port 10086

$request\_uri /sswang1/sswang2/test.txt

$document\_uri /sswang1/sswang2/test.txt

$document\_root /var/www/html

$request\_filename /var/www/html/sswang1/sswang2/test.txt

**2.2.4 URL重写**

经过上面反向代理和负载均衡的学习，我们知道Nginx可以基于这些方法实现比较好的分发请求的效果，而Nginx还有一个非常强大的精确分发请求的功能：Rewrite

**URL重写简介**

使用场景

根据开发功能需求定制用户浏览的URL，访问起来更规范合理，亦方便互联网搜索引擎收录网站内容

结合nginx内置变量分析用户请求信息，进行URL定制，达到用户请求的精确分发

方便企业动态调整项目URL,特别是伪静态处理和项目整体更新(域名)

Rewrite 作用

rewrite 指令的作用是实现 URL 地址重写(或跳转)。Nginx 的 rewrite 规则需要 PCRE 软件的支持，即通过 Perl 兼容正则表达式语法进行规则匹配。

Rewrite 语法

rewrite regex replacement [flag];

语法详解

rewrite 指令

regex 正则表达式，用于匹配旧的 URL 地址

replacement 重写/跳转到新的 URL 地址

[flag] 标记符号--重写类型

last   本条规则匹配完成后，继续向下匹配新的location URI规则，用于Server/if配置段

break   本条规则匹配完成即终止，不再匹配后面的任何规则，一般用于Location配置段

redirect   返回302临时重定向，浏览器地址会显示跳转后的URL地址，默认属性

permanent   返回301永久重定向，浏览器地址栏会显示跳转后的URL地址

regex 常用正则表达式说明

字符 描述 字符 描述 字符 描述

\ 转义字符 \d 匹配数字 [c] 匹配单个字符c

^ 锚定首字符 {n} 重复n次 [a-z] 匹配a-z间任一小写字母

$ 锚定尾字符 {n,} 重复>=n次 [^\/] 匹配非/之外的任意字符

\* 匹配前面的字符>=0次。如"it\*"能匹配"i"及"it"、"itt"

+ 匹配前面的字符>=1次。如"it+"能匹配"it"及"itt"、"ittt"，但不能匹配"i"

? 匹配前面的字符0/1次，例如"do(es)?"能匹配"do"或者"does"，"?"等效于"{0,1}"

. 匹配除"\n"之外的任何单个字符，若要匹配包括"\n"在内的任意字符，请使用诸如"[.\n]"之类的模式。

(pattern) 匹配括号内pattern内容，常用$0...$9属性整体获取小括号中的内容，要匹配圆括号字符需要\(Content\)

简单示例

server {

    # 访问 /last.html 的时候，页面内容重写到 /index.html 中

    rewrite /last.html /index.html last;

    # 访问 /break.html 的时候，页面内容重写到 /index.html 中，并停止后续的匹配

    rewrite /break.html /index.html break;

    # 访问 /redirect.html 的时候，页面直接302定向到 /index.html中

    rewrite /redirect.html /index.html redirect;

    # 访问 /permanent.html 的时候，页面直接301定向到 /index.html中

    rewrite /permanent.html /index.html permanent;

    # 把 /html/\*.html => /post/\*.html ，301定向

    rewrite ^/html/(.+?).html$ /post/$1.html permanent;

    # 把 /search/key => /search.html?keyword=key

    rewrite ^/search\/([^\/]+?)(\/|$) /search.html?keyword=$1 permanent;

}



Rewrite VS location

rewrite 是在同一域名内更改获取资源的路径

location 结合 proxy\_pass实现反向代理，不会对请求的资源路径进行变动

location 结合 rewrite 对某个具体的请求进行重写

这三种状况虽然能实现大致同样的效果，但是有一个优先级保证了他们的基本使用标准：

1 执行server块的rewrite指令

2 执行location匹配

3 执行选定的location中的rewrite指令

**Rewrite简单实践(1)**

配置文件

~# vim /etc/nginx/conf.d/rewrite.conf

server {

  listen 80;

  server\_name rewrite.com;

  rewrite ^/(.\*) http://www.rew.com/$1;

}

server {

  listen 80;

  server\_name www.rew.com;

  location / {

    root /var/www/html/rew/;

    index index.html index.htm;

  }

}

准备后端服务文件

mkdir -p /var/www/html/rew/

echo '<h1>Rewrite</h1>' > /var/www/html/rew/index.html

# vim /etc/hosts

192.168.8.14 rewrite.com www.rew.com

检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果

浏览器检查

curl -I rewrite.com

**Rewrite简单实践(2)**

配置文件

~# vim /etc/nginx/conf.d/last\_break.conf

server {

    listen 80;

    server\_name localhost;

    location / {

        rewrite /last/ /ok.html last;

        rewrite /break/ /ok.html break;

    }

    location = /ok.html {

        return 400;

    }

}

准备后端服务文件

echo '<h1>break\_OK</h1>' > /var/www/html/ok.html

检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果

浏览器检查

curl localhost/break/

curl localhost/last/

**if 指令简介**

if指令主要是基于nginx的内置变量值来获取请求的关键字，从而进行精确的分发功能。

if语法

if(内置变量 操作符 值){ ... }

对给定的条件(内置变量 操作符 值)进行判断。如果为真，大括号内的rewrite指令将被执行。

，if条件(conditon)可以是如下任何内容：

内置变量主要来自于nginx内部，可以参考2.2.3部分。

操作符主要有三类：

精确匹配： =、!=

正则匹配：~、~\*、!~

文件匹配：

-f、!-f 判断是否存在普通文件

-d、!-d 判断是否存在目录文件

-e、!-e 判断是否存在文件或目录

-x、!-x 判断文件是否可执行

if指令示例

如果提交方法为POST，则返回状态405（Method not allowed）。return不能返回301,302

if ($request\_method = POST) {

    return 405;

}

如果请求的文件名不存在，则反向代理到localhost 。这里的break也是停止rewrite检查

if (!-f $request\_filename){

    break;

    proxy\_pass  http://127.0.0.1;

}

如果host不是www.nihao.com，则重定向到标准的www.nihao.com中

server {

  listen 80;

  server\_name nihao.com www.nihao.com;

  if ( $host != "www.nihao.com" ){

      rewrite ^/(.\*)$ http://www.nihao.com/$1 permanent;

  }

  location / {

    try\_files $uri $uri/ =404;

  }

}

**if实践之防盗链**

需求：

对我们的图片服务器，只允许特定的网站才能进行访问，其他任意网站访问的时候，都返回403或者报错页面。

我们主要是针对$invalid\_referer 内置变量的判断

配置文件

~# vim /etc/nginx/conf.d/tupian.conf

server {

  listen 80;

  server\_name localhost;

  location ~\* \.(gif|jpg|png|webp)$ {

    valid\_referers none blocked domain.com \*.domain.com server\_names ~\.google\. ~\.baidu\.;

    if ($invalid\_referer) {

      return 403;

      #rewrite ^/ http://www.domain.com/403.jpg;

    }

    root /var/www/html/image;

  }

}

准备后端服务文件

mkdir /var/www/html/image -p && cd /var/www/html/image/

上传一个名字为a.jpg的图片

检查nginx配置后重载服务

/usr/sbin/nginx -t

systemctl reload nginx

netstat -tnulp | grep nginx

查看效果

浏览器检查

curl -I localhost/a.jpg

curl -I localhost/a.jpg -H 'Referer:http://www.qq.com'

curl -I localhost/a.jpg -H 'Referer:http://www.www.com'

curl -I localhost/a.jpg -H 'Referer:http://www.baidu.com'

tail -f /var/log/nginx/access.log