[nginx反向代理配置](http://blog.csdn.net/physicsdandan/article/details/45667357)

标签： [nginx](http://www.csdn.net/tag/nginx)[配置](http://www.csdn.net/tag/%e9%85%8d%e7%bd%ae)[代理](http://www.csdn.net/tag/%e4%bb%a3%e7%90%86)[网络](http://www.csdn.net/tag/%e7%bd%91%e7%bb%9c)

2015-05-12 11:36 7669人阅读 [评论](http://blog.csdn.net/physicsdandan/article/details/45667357#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/physicsdandan/article/details/45667357#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

系统管理（3） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/physicsdandan/article/details/45667357)

nginx反向代理配置

什么是代理

代理在普通生活中的意义就是本来应该你做的事情，你让别人代你做了，那么那个帮你做的人就是你的代理。而在计算机网络中代理的概念差不多，就是本来要客户端要做的网络访问，现在移交给另外一个机器做，那么那个机器就被称为代理服务器，代理服务器帮你来访问。过程如下：

正常情况：   
client —(send request)—> server

代理情况：   
client —(send request)—> clinet proxy –(send request)—> server

什么又是反向代理

那什么又是反向代理呢？反向代理可不是说本来代理你事务的人，反过来代理别人。反向代理在计算机网络中是指这么一个过程。一般来说正向代理是客户机找人来代理把自己的请求转发给服务端，但是如果反向代理，找代理的人不再是客户机，而是服务器这边把自己接受的请求转发给背后的其他机器。其主要区别：

* 正向代理中代理的过程是客户端，代理机器是作为一个访问客户的身份的；而在反向代理中代理机器是作为服务身份。
* 正向代理中代理的过程是服务端，服务端对代理的存在无感知；而在反向代理中客户机对代理的存在无感知。

反向代理情况：   
clinet –(send request)–> server proxy –(send request)–>other   
server

先让我们看看一个示例

#① part start

#运行nginx进程的账户

user www;

#

worker\_process 1;

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

pid /var/run/nginx.pid;

events{

use epoll;

worker\_connections 1024;

}

http{

include /etc/nginx/mime.types;

default\_type application/octet-stream;

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

#

sendfile on;

#

keepalive\_timeout 65;

gzip on;

index index.html index.htm;

include /etc/nginx/conf.d/\*.conf;

include /etc/nginx/sites-enabled/\*;

#② part start

# 定义上游服务器列表组

upstream web1 {

server 127.0.0.1:111 weight=1;

server 127.0.0.1:222 weight=1;

}

upstream web2 {

server 127.0.0.2:111 weight=1;

server 127.0.0.2:222 weight=6;

server 127.0.0.2:333 weight=7;

}

#定义一个服务器，其监听80端口，配置的域名是www.company.com

server{

listen 80;

# using www domain to access the main website

server\_name www.company.com;

access\_log /var/log/nginx/www.log

location / {

root /home/website\_root;

}

}

#③ part start

#定义第二个服务器，其同样监听80端口，但是匹配域名是web.company.com

server{

listen 80;

# using web sub domain to access

server\_name web.company.com;

access\_log /var/log/nginx/web\_access.log

location / {

root /home/web2\_root;

proxy\_pass http://127.0.0.1:8080/web/;

proxy\_read\_timeout 300;

proxy\_connect\_timeout 300;

proxy\_redirect off;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

#定义第三个服务器，其同样监听80端口，但是匹配域名是web1.company.com，并把请求转发到web1上游服务

server{

listen 80;

# using web1 sub domain to access

server\_name web1.company.com;

access\_log /var/log/nginx/web1\_access.log

location / {

root /home/web1\_root;

proxy\_pass http://web1;

proxy\_read\_timeout 300;

proxy\_connect\_timeout 300;

proxy\_redirect off;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

#定义第三个服务器，其同样监听80端口，但是匹配域名是web2.company.com，并把请求转发到web2上游服务

server{

listen 80;

# using web2 sub domain to access

server\_name web2.company.com;

access\_log /var/log/nginx/web2\_access.log

location / {

root /home/web2\_root;

proxy\_pass http://web2;

proxy\_read\_timeout 300;

proxy\_connect\_timeout 300;

proxy\_redirect off;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108

这个示例都做了什么

1. 第①部分，定义nginx通用规则，包括运行账户，处理进程个数等
2. 第②部分，开始定义上游服务器组
3. 第③部分，定义server，并指定怎么使用第②部分定义的upstream

总体来说就是这个提供了4个服务，www，web，web1，web2 4个网站。这个例子很适合只有一台机器，但是有想避免url中携带端口号，统一使用域名的方式访问。4个网站都监听80端口，但是分配不同的二级域名既可以。这就需要nginx的反向代理，把接到的请求转发给背后不同的服务。

为什么需要方向代理

为什么要反向代理？作用服务端的代理，自然就是一台服务器处理不过来了，需要转发、分散请求给其他服务器做。下面罗列些适用场景：

* 负载均衡   
  上面例子中的web1和web2使用了nginx的负载均衡技术，把请求转向一组服务器。具体转发到哪个服务器，nginx提供了多种负载策略，例子中使用的是加权重的方式，web1 upstream是2个请求中，1个请求给111服务器，1个给222服务器。关于跟多的负载均衡的策略，请参看[nginx官方文档-负载均衡](http://nginx.org/en/docs/http/load_balancing.html)
* 一个域名，多个网站。在这里反向代理倒不是为了负责存在，而是为了域名和服务的统一部署。例如一个公司的内部网站需要搭建很多服务——代码管理服、wiki服务、oa……，但是只要一个域名。这时候就可以用反向代理把不同的子域名转发到不同的服务上。下面是一个例子：
* 当然反向代理的另一大用处就是隐藏后面的实际服务，以此来达到一定的安全性。

仔细讲解每个模块

user

user 设置nginx是以什么用户来运行的，这个非常重要，**确保运行nginx的用户能有权限访问读写网站的文件**,否则会报404 not found等错误。

events

[nginx upstream](http://nginx.org/cn/docs/http/ngx_http_upstream_module.html)

upstream 直接翻译就是上游，即上游服务，其封装一组服务器列表，这些服务器可以别proxy\_pass,fastcgi\_pass,uwsgi\_pass,scgi\_pass和 memcached\_pass引用，把接到的请求转发给这些服务器组。   
引用方法就是加行[http://[upstream](http://[upstream/) module name]

> The ngx\_http\_upstream\_module module is used to define groups of servers that can be referenced by the proxy\_pass, fastcgi\_pass, uwsgi\_pass, scgi\_pass, and memcached\_pass directives.

例子：

upstream backend { #①

server backend1.example.com weight=5;

server backend2.example.com:8080;

server unix:/tmp/backend3;

server backup1.example.com:8080 backup;

server backup2.example.com:8080 backup;

}

server {

location / {

proxy\_pass http://backend; #②

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15

注意有①，和②行的写法。要引用**backend**模块，只需把它制定成[**http://backend**](http://backend/)就行。

http 意义和配置

http就是指配置关于http服务的地方，server等都是http的子模块

server 配置和匹配规则

一个http服务可以有多个server，而对server的路径匹配，反向代理都是在这里配置的。

在server中最重要的一项配置：server\_name的配置。server\_name决定了来了一个url，到底是哪个server处理该请求。nginx会依次找和url配置的第一次出现的server。server\_name可以使用通配符，也可以使用正则表达式。而且一个server的server\_name可以多个，以空格分隔。更详细的关于server\_name匹配规则，[参看这里](http://nginx.org/en/docs/http/server_names.html)

location 配置和匹配规则

server\_name是定义域名级别的规则，而location则是url中文件部分的规则的。适应例如会对图片等静态资源做单独处理等需求。

参考资料

* [nginx upstream模块介绍文档1](http://nginx.org/cn/docs/http/ngx_http_upstream_module.html)
* [nginx upstream模块介绍文档2](http://tengine.taobao.org/book/chapter_05.html)
* [nginx配置例子](http://www.cnblogs.com/xiaogangqq123/archive/2011/03/02/1969006.html)
* [nginx load balancer](http://nginx.org/en/docs/http/load_balancing.html)
* [Enhanced Load Balancing,High Availability, and Monitoring Features](http://nginx.com/blog/nginx-plus-r6-released/?_ga=1.99990941.241485225.1426074750)