# Nginx学习

相关链接：

https://segmentfault.com/a/1190000015006377?utm\_source=index-hottest --未完成

https://www.cnblogs.com/yaun1498078591/p/9268591.html

Nginx基本功能：

http服务器（包含动静分离）

反向代理服务器

正向代理

负载均衡

pop3/imap4协议反向代理服务器

FastCGI(LNMP)，uWSGI(python)等协议

模块化(非DSO)，如zip，SSL模块

## 概述

Nginx（engine x）是开源、高性能的HTTP和反向代理服务器，邮件代理服务器（IMAP/POP3），通用TCP/UDP代理服务器。

解决了c10K问题（所谓c10K问题，指服务器同时支持成千上万个客户端连接）

Nginx有一个主进程和多个工作进程。主进程的作用只有一个，即通过读取配置文件（nginx.conf）来维持工作进程正常工作；工作进程对应的才是实际的请求（request），nginx会自动将客户端的请求高效地分发给工作进程。工作进程的数量在配置文件中定义，或者通过cpu的核心数自动调整。

Nginx是支持热启动的，即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。

## 服务器比较Apache、Toncat、Nginx、lighttpd

## 应用场景

-1）http服务器：

Nginx是一个轻量级的高性能http服务器，可以将服务器上的静态文件（如html、img）通过http协议展现给客户端。

-2）虚拟主机：

可以实现在一台服务器虚拟出多个网站。例如个人网站使用的虚拟主机。

例如将www.aaa.com和www.bbb.com两个网站部署在同一台服务器上，两个域名解析到同一个IP地址，但是用户通过两个域名却可以打开两个完全不同的网站，互相不影响，就像访问两个服务器一样，所以叫两个虚拟主机。

虚拟主机的原理是通过HTTP请求头中的Host是否匹配server\_name来实现的

-3）反向代理&负载均衡：

当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群，可以使用nginx做反向代理，并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况。

-4）容错技术：

容错就是当由于种种原因在系统中出现了数据、文件损坏或丢失时，系统能够自动将这些损坏或丢失的文件和数据恢复到发生事故前的状态，是系统能够连续正常运行的技术。

## 安装教程

-1）下载并解压nginx.rar

-2）启动：

双击nginx.exe（方式一）：

直接双击nginx.exe出现屏幕一闪而过，证明已经启动成功了。任务管理器会有两个nginx.exe进程，其中1个是守护进程，1个工作进程。

测试：浏览器输入http://127.0.0.1后，出现welcome to nginx！

命令行启动（方式二）：

打开：（呼出cmd输入）

> cd d:/nginx

> start nginx

关闭：> nginx –s stop

修改端口号：

打开nginx目录下的conf/nginx.conf文件，然后找到listen 80,修改为9090，关闭nginx再启动，就可以通过如下地址访问了:http://127.0.0.1:9090

## Nginx配置文件（nginx.conf）

### 文件结构

打开配置文件：vim /usr/local/nginx/conf/nginx.conf

Nginx配置文件(nginx.conf)结构如下：

... #全局块

events { ... } #events块

http { #http块

... #http全局块

server { #server块

... #server全局块

location [pattern] { ... } #location块1

location [pattern] { ... } #location块2

}

server { ... } #server块

}

说明：

-1）全局块：配置影响nginx全局的指令。一般有运行nginx服务器的用户组，nginx进程pid存放路径，日志存放路径，配置文件引入，允许生成worker process数等。

-2）events块：配置影响nginx服务器或与用户的网络连接。有每个进程的最大连接数，选取哪种事件驱动模型处理连接请求，是否允许同时接受多个网络连接，开启多个网络连接序列化等。

-3）http块：可以嵌套多个server，配置代理，缓存，日志定义等绝大多数功能和第三方模块的配置。如文件引入，mine-type定义，日志自定义，是否使用sendfile传输文件，连接超时时间，单连接请求数等。

-4）server块：配置虚拟主机的相关配置参数，一个http中可以有多个server。

-5）location块：配置请求的路由，以及各种页面的处理情况。

### 实例：

#user administrator administrators; #配置用户或组，默认位nobody nobody

#worker\_processes 2; #允许生成的进程数，默认根据CPU核数设置。

#pid /nginx/pid/nginx.pid; #指定存储nginx主进程pid的文件路径。作用：当执行nginx -s stop时，会找到此文件存放的进程编号然后调用kill命令将其终止

指定nginx进程运行文件存放地址

error\_log log/error.log debug; #指定日志路径，级别。可在全局块、http块、server块配置。级别：debug,info,notice,warn,error

events {

accept\_mutex on; #设置网络连接序列化，防止惊群现象发生，默认为on

multi\_accept on; #设置一个进程是否同时接受多个网络连接，默认为off

#use epoll; #事件驱动模型，select|poll|kqueue|epoll|resig|/dev/poll|eventport

worker\_connections 1024; #最大连接数，默认512

}

http {

include mine.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，默认为text/plain

#access\_log off; #取消服务日志

log\_format myFormat '$remote\_addr–$remote\_user [$time\_local] $request $status $body\_bytes\_sent $http\_referer $http\_user\_agent $http\_x\_forwarded\_for'; #日志格式

access\_log log/access.log myFormat; #日志部分内容

sendfile on; #允许sendfile方式传输文件，默认为off，可以在http块，server块，location块。

sendfile\_max\_chunk 100k; #每个进程每次调用传输数量不能大于设定的值，默认为0，即不设上限。

keepalive\_timeout 65; #连接超时时间，默认为75s，可以在http&server&location块。

upstream myStrean {

server 127.0.0.1:8081;

server 127.0.0.1:8082 backup; #热备

}

error\_page 404 https://www.baidu.com; #错误页，当状态码为404时导向百度

server {

keepalive\_requests 120; #单连接请求上限次数

listen 8080; #监听端口

server\_name 127.0.0.1; #监听地址

location ~\*^.+$ { #请求url过滤，正则匹配，~为区分大小写，~\*为不区分大小写

#root path; #根目录

#index vv.txt; #设置默认页

proxy\_pass http://myStrean; #请求转向myStrean定义的服务器列表

deny 127.0.0.1; #拒绝的ip

allow 172.18.5.54; #允许的ip

}

}

}

附：

配置简单的反向代理服务器，只需匹配到路由的location模块中，使用proxy\_pass来转到真实的服务器。如：location / { proxy\_pass://www.baidu.com; }

惊群现象：一个网路连接到来，多个睡眠的进程被同事叫醒，但只有一个进程能获得链接，这样会影响系统性能

### server元素说明

server相当于一个服务器，可配置多个。listen表示该服务器监听的端口，默认80。server\_name表示匹配主机名，比如www.abc.com，当通过[www.abc.com](http://www.abc.com)来访问时就是匹配该服务器。

### location元素

location就是匹配的路由，root是返回给客户端资源文件的根目录，它相对nginx路径的，也可指定绝对路径（如：/usr/local/nginx/www）。

在一个server中location配置段可存在多个，用于实现从uri到文件系统的路径映射。

实例：

server {

server\_name www.test.com;

root /data/www/

location /images/ {

root /app/test/;

}

}

# 当访问 http://www.test.com/images -> /app/test/images/

# 但当访问 http://www.magedu.com/时不添加子路径又会访问/data/www/下的文件

### 日志文件说明

作用：

通过访问日志，你可以得到用户ip、跳转来源、使用终端、某个url访问量等相关信息。

通过错误日志，你可以得到系统某个服务或server的性能瓶颈等。

配置如下：

log\_format myFormat '$remote\_addr–$remote\_user [$time\_local] $request $status $body\_bytes\_sent $http\_referer $http\_user\_agent $http\_x\_forwarded\_for'; #日志格式

access\_log log/access.log myFormat; #日志部分内容

说明：

$remote\_addr和$http\_x\_forwarded\_for：记录客户端的ip地址

$remote\_user：记录客户端用户名称

$time\_local：记录访问时间与时区

$request：记录请求的url与http协议

$status：记录请求状态；成功是200

$body\_bytes\_sent：记录发送给客户端文件主体内容大小

$http\_referer：记录从那个页面链接访问过来的

$http\_user\_agent：记录客户端浏览器的相关信息

### 加载从配置文件

include file | mask #指明包含进来的其他配置文件片段

load\_module file #模块加载配置文件：/usr/share/nginx/modules/\*.conf指明要装载的动态模块路径

## 常用命令

Linux常用命令：

查看安装目录：whereis nginx

查找配置文件：find / | grep nginx.conf

查看进程：ps -ef|grep nginx

启动：nginx /usr/local/nginx/sbin/nginx

访问地址：http://192.168.2.52/ #端口号默认为80

重启nginx： ./nginx -s reload

启动nginx： nginx -c /etc/nginx/nginx.conf

退出和停止：nginx -s quit 和 nginx -s stop

重新打开日志文件：nginx -s reopen

nginx命令的选项：

-h #查看帮助选项

-V #查看版本和配置选项

-t #测试nginx语法错误

-c filename #指定配置文件(default: /etc/nginx/nginx/conf)

-s signal #发送信号给master进程，signal取值：stop/quit/reopen/reload

如：nginx -s stop #关闭nginx

nginx -s reload #重新加载配置文件

-g directives #在命令行中指明全局指令

nginx 常用命令：

nginx -s stop #快速关闭Nginx，可能不保存相关信息，并迅速终止web服务

nginx -s quit #平稳关闭Nginx，保存相关信息，有安排的结束web服务

nginx -s reload #重新加载（Nginx配置问卷改变时需重新加载）

nginx -s reopen #重新打开日志文件

nginx -c filename #为Nginx指定一个配置文件，来替代缺省的

nginx -t #不运行，仅测试配置文件。

nginx -v #显示nginx版本

nginx -V #显示nginx版本

## 配置虚拟主机

-1）配置搭建三个虚拟主机的各自的配置文件

# vim /etc/nginx/conf.d/test.conf

server {

listen 80;

server\_name www.a.com;

root /data/test1;

}

server {

listen 8080;

server\_name www.b.com;

root /data/test2;

}

server {

listen 80;

server\_name www.c.com;

root /data/test3;

}

-2）建立三个主机各自的主页面文件

mkdir /data/test1

mkdir /data/test2

mkdir /data/test3

echo web1>/data/test1/index.html

echo web2 >/data/test2/index.html

echo web3 >/data/test3/index.html

-3）重新加载nginx服务

# nginx -s reload

## 反向代理

### 相关概念

-1）正向代理

指我们通过中间代理去访问另一个服务器。比如我们一般都通过连接一个代理服务器，通过代理服务器连接要访问的国外某网站，这网络访问形式就是正向代理。

比如说要访问youtube，但是不能直接访问，只能先找个翻墙软件，通过翻墙软件才能访问youtube。该翻墙软件就叫正向代理

-2）反向代理

代理服务器nginx来接受Internet上的连接请求，然后将请求转发给内部某一服务器（多个服务器运行着同一个项目），并将内部服务器得到的结果返回给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

比如说用户要访问youtube，但是youtube悄悄把这个请求交给bilibili来做。该bilibili就是反向代理。

再比如说我们访问nginx，但nginx把请求交给tomcat来做。

反向代理服务器具有以下功能：

-1）保护网站安全：任何来自Internet的请求都必须先经过代理服务器；

-2）通过配置缓存功能加速Web请求，减轻真实Web服务器的负载压力；

-3）实现负载均衡：均衡地分发请求，平衡集群中各个服务器的负载压力。

### 实例：

-1）修改nginx.conf，主要在30~42行

server {

listen 80; #监听端口80

server\_name localhost; #服务器名

location / { #表示处理所有请求

proxy\_pass http://127.0.0.1:8111; #表示把请求交给它处理

}

}

-2）启动Tomcat和nginx

启动Nginx: nginx –s reload

-3）测试：

在浏览器输入 http://localhost ,发现访问的就是tomcat的主页，此时它会反向代理到tomcat中。

-4）意义：

既然直接通过127.0.0.1:8111/login.jsp 可以访问，那么干嘛要通过127.0.0.1/login.jsp 去反向代理到tomcat呢？ 为什么不直接使用tomcat呢？

因为nginx在处理静态文件的吞吐量上面比tomcat好很多，通常他们俩配合，不会把所有的请求都如本例所示的交给tomcat, 而是把静态请求交给nginx，动态请求，如jsp, servlet,ssm， struts等请求交给tomcat. 从而达到动静分离的效果。

## 多个tomcat使用

描述：

nginx是一个web服务器，很快速，但是不能作为Servlet容器独立运行，所以通常的工作方式是nginx配合tomcat来协同工作。

提供多个tomcat原因：后续的负载均衡和session共享教程里需要多个tomcat配合演示。

启动tomcat：

解压tomcat.rar，并运行其中bin/startup.bat文件。在conf/server.xml文件中配置端口

在浏览器输入：http://127.0.0.1:8080

## 负载均衡

### 相关概念

定义：

就是一个业务请求可以交给多个服务器来处理。但要根据每个服务器的负载情况，很忙的服务器不去处理该业务请求，仅交给比较闲的服务器去处理。

当访问量很大时，一个tomcat吃不消，这时候准备多个tomcat，由nginx按照权重来对请求进行分配，从而缓解单独一个tomcat受到的压力。

先到ngix，然后根据，tomcat的负载情况，去访问某个tomcat。

问题描述：

互联网早期，业务流量比较小并且业务逻辑比较简单，单台服务器便可以满足基本的需求。但随着发展，业务流量越来越大并且业务逻辑也越来越复杂，因此需要多台服务器进行性能的水平扩展以及避免单点故障。但如何将不同用户的流量分发到不同的服务器上面呢？

早期的方法：

使用DNS做负载，通过给客户端解析不同的IP地址，让客户端的流量直接到达各个服务器。缺点是延时性问题。

负载均衡原理：

客户端的流量首先会到达负载均衡服务器（nginx），由负载均衡服务器通过一定的调度算法将流量分发到不同的应用服务器上面，同时负载均衡服务器也会对应用服务器做周期性的健康检查，当发现故障节点时会将它从服务器集群中剔除，以此来保证应用的高可用。

负载均衡分类：

1）四层负载均衡工作在OSI模型的传输层，主要工作是转发，它在接收到客户端的流量以后通过修改数据包的地址信息将流量转发到应用服务器。

2）七层负载均衡工作在OSI模型的应用层。

### 实现

--1）修改nginx.conf配置

http {

...

upstream tomcats {

server 192.168.0.100:8080 weight=2; # 2/6次

server 192.168.0.101:8080 weight=3; # 3/6次

server 192.168.0.102:8080 weight=1; # 1/6次

}

server {

listen 80;

location / {

proxy\_pass http://tomcats;

}

}

...

}

--2）在上游配置代理服务器组，并定位集proxy\_pass上游组名。当请求到来时，Nginx会轮询服务器在指定的上游过程。

步骤1：首先增加一个upstream，用来指向这两个tomcat

-- -weight表示权重，值越大，被分配到的几率越大。

upstream tomcat\_8111\_8222 {

server 127.0.0.1:8111 weight=1;

server 127.0.0.1:8222 weight=2;

}

步骤2：然后修改location，反向代理到上述配置

location ~\*^.+$ {

proxy\_pass http://tomcat\_8111\_8222;

}

### 负载均衡轮询策略

-1）热备：如果你有2台服务器，当一台服务器宕机时，另一服务器才提供服务

upstream mysvr {

#服务器请求顺序：AAA突然A挂了，BBB...

server 127.0.0.1:8081;

server 127.0.0.1:8082 backup; #热备

}

-2）轮询：默认，每次请求逐个分发到后端服务器

upstream mysvr {

# 服务器请求顺序为:ABABAB...

server 127.0.0.1:8081;

server 127.0.0.1:8082;

}

-3）加权轮询：按权重大小分发请求

upstream mysvr {

# 服务器请求顺序为:ABBABB...

server 127.0.0.1:8081 weight=1;

server 127.0.0.1:8082 weight=2;

}

-4）ip\_hash：同一客户端ip请求相同的服务器

upstream mysvr {

server 127.0.0.1:8081;

server 127.0.0.1:8082;

ip\_hash;

}

## Session共享

问题描述：

通过负载均衡，我们可以把请求分发到不同的tomcat来缓解服务器的压力。但存在一个问题：当同一个用户第一次访问tomcat\_8111并登录成功，而第二次却被分配到tomcat\_8222，这里并没有记录他的登录状态，那么就会呈现未登录状态。

解决方法：

用Redis来存取session.当tomcat1需要保存session值的时候，就可以把它放在Redis上，需要取的时候，也从Redis上取。（借助Jedis对tomcat做session共享）

## 动静分离

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。

动静分离，即css、js、images等都交给nginx来处理，nginx处理不了的，如jsp就交给tomcat来处理。

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。

好处：nginx处理静态内容的吞吐量很高，比tomcat高多了，这样无形中提升了性能。

实例：

upstream mystream {

server localhost:8081;

server localhost:8082;

}

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

# 把html&图片等放到wwwroot目录下，而tomcat只负责处理jsp和请求

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

root D:/wwwroot;

}

# 所有动态请求都转发给tomcat处理

location ~ \.(jsp|do)$ {

proxy\_pass http://mystream;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root e:/wwwroot;

}

}

## Nginx缓存配置

--开启简单的缓存配置，只需要两个指令：proxy\_cache\_path和proxy\_cache。

proxy\_cache\_path #配置缓存的存放地址和其他的一些常用配置;

proxy\_cache #启动缓存

实例：

proxy\_cache\_path /path/to/cache levels=1:2 keys\_zone=mycache:10m max\_size=10g inactive=60m use\_temp\_path=off;

server {

　　# ...

　　location / {

　　　　proxy\_cache mycache;

　　　　proxy\_pass http://my\_upstream;

　　}

}

## 搭建nginx图片服务器

1）创建存放图片目录

mkdir /usr/local/nginx/html/images

2）修改nginx.conf

--在默认的server里再添加一个location并指定实际路径

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

location /images/ {

root /home/ftpuser/www; #存放图片的服务器路径上一级

autoindex on

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

## 其它配置

设置404页面导向地址：

error\_page 404 https://www.runnob.com; #错误页

proxy\_intercept\_errors on; #若被代理服务器返回的状态码为400及以上，设置的error\_page配置起作用。默认为off。

设置客户端的请求方法：

proxy\_method get; #支持客户端的请求方法。post/get

设置支持的http协议版本：

proxy\_http\_version 1.0; #Nginx服务器提供代理服务的http协议版本1.0，1.1，默认设置为1.0版本

# Nginx优化

优化如下：

worker\_processes auto; #优化1，进程数

events { #优化2，事件驱动模型

worker\_connections 102400; #每个工作进程允许的最大连接数

accept\_mutex on;

multi\_accept on; #打开同时接受多个新网络连接请求的功能

use epoll; #使用高性能的epoll事件驱动，处理效率高

}

http {

include mine.types;

default\_type application/octet-stream;

# 优化3，日志优化

log\_format myFormat '$remote\_addr – $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

# 优化4，高性能传输模式

sendfile on; #开启高效文件传输模式

tcp\_nopush on; #需要在sendfile开启模式才有效，防止网路阻塞，积极的减少网络报文段的数量。将响应头和正文的开始部分一起发送，而不一个接一个的发送。

# 优化5，开启传输压缩

gzip on; #开启传输压缩功能，可以压缩传输的页面、js、json数据、图片等，占用更小的带宽，提高传输性能

gzip\_min\_length 1k;

gzip\_buffers 4 32k;

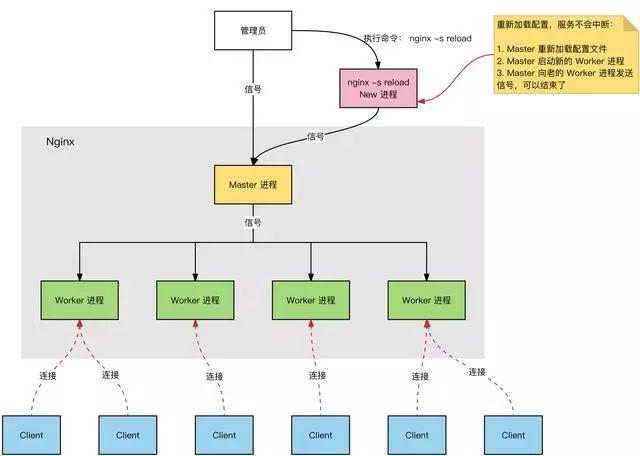
gzip\_http\_version 1.0;

gzip\_comp\_level 8;

...

}

# Nginx进程模型



Nginx服务器，正常运行过程中：

多进程：一个master进程、多个worker进程

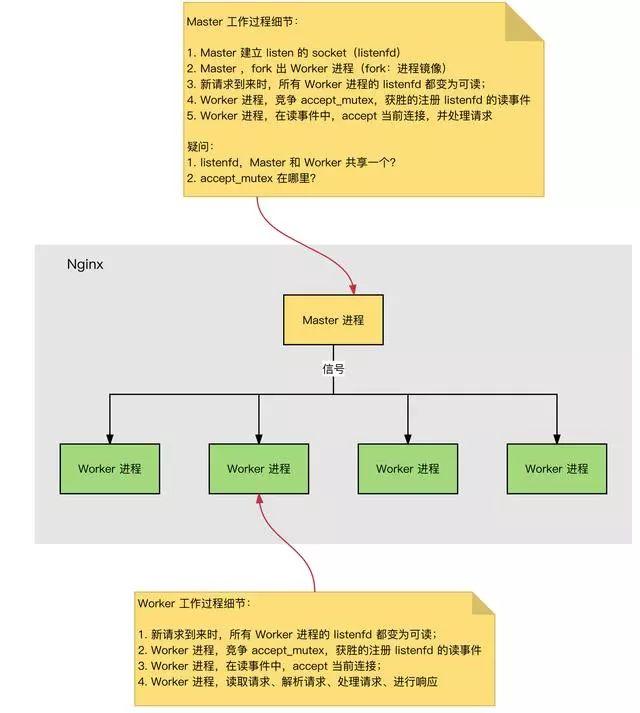
Master进程：管理worker进程。对外接口：接收外部的操作（信号）；对内转发：根据外部的操作不同，通过信号管理worker；监控：监控worker进程的运行状态，worker进程异常终止后，自动重启worker进程。

Worker进程：所有worker进程都是平等的。实际处理：网络请求，由worker进程处理。Worker进程数量：在nginx.conf配置，一般设置为核心数，充分利用CPU资源，同时避免进程数量过多，避免进程竞争CPU资源，增加上下文切换损耗。

相关问题：

请求是连接到nginx，master进程负责处理和转发？

如何选定哪个worker进程处理请求？请求的处理结果，是否是要经过master进程？



http连接建立和请求处理过程：

1. nginx启动时，master进程，加载配置文件

2、master进程，初始化监听的socket

3、master进程，fork出多个worker进程

4、worker进程，竞争新的连接，获胜方通过三次握手，建立socket连接，并处理请求

问：nginx为什么拥有高性能，并能够支撑高并发？

Nginx采用多进程+异步非阻塞方式（IO多路复用Epoll）

请求的完整过程：建立连接->读取请求->解析请求->处理请求->响应请求

请求的完整过程对应到底层就是：读写socket事件。

# nginx的事件处理模型

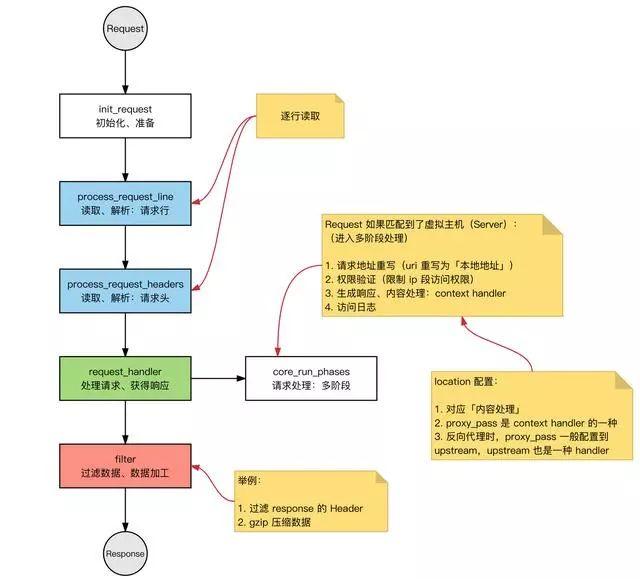
基本的http web server工作模式：

接收请求：逐行读取请求行和请求头，判断有请求体后，读取请求体

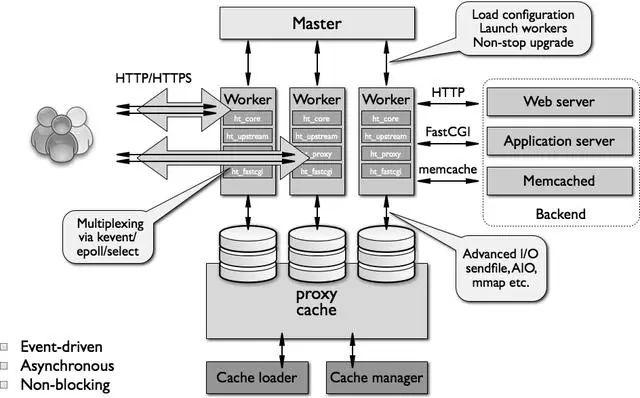
处理请求

返回响应：根据处理结果，生成相应的http请求（响应行、响应头、响应体）

Nginx工作流程：



模块化体系结构：



Nginx的模块根据其功能来分类：

1、event module：搭建了独立于操作系统的事件处理机制的框架，及提供了各具体事件的处理。包括 ngx\_events\_module，ngx\_event\_core\_module 和 ngx\_epoll\_module 等。

Nginx 具体使用何种事件处理模块，这依赖于具体的操作系统和编译选项。

2、phase handler：此类型的模块也被直接称为 handler 模块。主要负责处理客户端请求并产生待响应内容，比如 ngx\_http\_static\_module 模块，负责客户端的静态页面请求处理并将对应的磁盘文件准备为响应内容输出。

3、output filter：也称为 filter 模块，主要是负责对输出的内容进行处理，可以对输出进行修改。

例如，可以实现对输出的所有 html 页面增加预定义的 footbar 一类的工作，或者对输出的图片的 URL 进行替换之类的工作。

4、upstream：upstream 模块实现反向代理的功能，将真正的请求转发到后端服务器上，并从后端服务器上读取响应，发回客户端。

upstream 模块是一种特殊的 handler，只不过响应内容不是真正由自己产生的，而是从后端服务器上读取的。

5、load-balancer：负载均衡模块，实现特定的算法，在众多的后端服务器中，选择一个服务器出来作为某个请求的转发服务器。

Nginx和apache区别：

Nginx：

IO多路复用，epoll（freebsd上是kqueue）

高性能

高并发

占用系统资源少

Apache：

阻塞+多进程/多线程

更稳定，bug少

模块更丰富

Nginx的并发处理能力：

并发连接数，一般优化后，峰值能保持在1~3w左右（内存和CPU核心数不同，会有进一步优化空间）