

Nginx基础02：配置文件nginx.conf (Part1)

文章分类: *Server*; 标签: *Nginx*; 作者: *Hackyle*;

更新时间: *Mon Jan 16 14:15:49 CST 2023*

我们使用Nginx主要是通过其配置文件nginx.conf来实现的。按照一定的规则，编写特定的指令，可以帮助我们实现对Web服务的控制！**所以，学习Nginx的用法，几乎就是学习nginx.conf！**

如何使用本篇文章

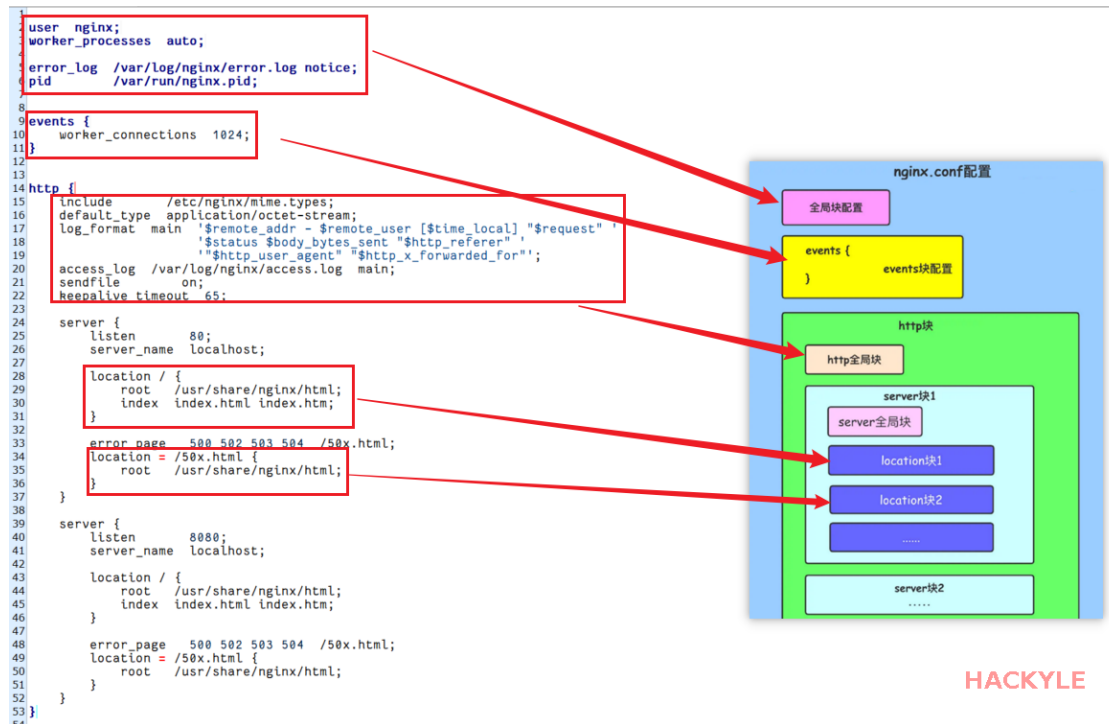
- 本文作为一篇高度总结和罗列nginx.conf中所有的基础配置项，循规蹈矩地按照文章的**顺序阅读的方式不可取**
- 笔者建议所有读者，**先看目录**，掌握Nginx都有哪些基础的配置块，再想要了解那**一个配置块**时，**再详细去看**
- 作为一篇字典类的文章，建议读者**善用浏览器的全文查找功能**，按Ctrl + F调出查找功能，搜索你感兴趣的关键词，针对性地学习

内容导览

- [nginx.conf的基本结构](#)
- [全局配置块](#)
 - [user](#)
 - [worker_processes](#)
 - [其他](#)
- [events块](#)
- [http块](#)
 - [http全局](#)
 - [公有配置](#)
 - [keepalive](#)
 - [静态资源优化配置](#)
 - [静态资源的压缩](#)
 - [综合实例](#)
 - [Gzip和sendfile共存问题](#)
 - [缓存](#)
 - [ResponseHeader中的缓存](#)
 - [清除缓存](#)
 - [不缓存](#)
 - [log](#)
 - [负载均衡](#)
 - [第七层负载均衡](#)
 - [第七层的均衡策略](#)
 - [第四层负载均衡](#)

nginx.conf的基本结构

TOP



- 配置文件一共由三部分组成，分别为全局块、events块和http块。
- 在http块中，又包含http全局块、多个server块。
- 每个server块中，可以包含server全局块和多个location块。
- 在同一配置块中嵌套的配置块，各个之间不存在次序关系。
- 牢牢把握住这张图，可以帮助初学者快速建立起对nginx.conf的初步印象。

nginx.conf文件在哪里？

- 在windows平台下，该配置文件在其数据包的conf目录下
- 在CentOS7平台，如果使用源码安装，则配置文件也在安装位置的conf目录下；如果使用yum安装，则配置文件在/etc/nginx/目录下
- 全局查找nginx.conf文件：find / -name nginx.conf

```
root@localhost:~
File Edit View Search Terminal Help
[root@localhost ~]# find / -name nginx.conf
/etc/nginx/nginx.conf
[root@localhost ~]#
```

配置文件的语法规则：

- 配置文件由指令与指令块构成；
- 每条指令以；分号结尾，指令与参数间以空格符号分隔；
- 指令块以 {} 大括号将多条指令组织在一起；
- include 语句允许组合多个配置文件以提升可维护性；
- 使用 # 符号添加注释，提高可读性；
- 使用 \$ 符号使用变量；
- 部分指令的参数支持正则表达式；

全局配置块

设置一些影响Nginx服务器整体运行的配置指令，这些指令的作用域是Nginx服务器全局。

```
nginx.conf
1 # 定义Nginx运行的用户和用户组
2 user  nobody;
3
4 # 定义Nginx运行时的进程数
5 worker_processes 1;
6
7 # 全局错误日志定义类型, [ debug | info | notice | warn | error | crit ]
8 error_log  logs/error.log  info;
9
10 # 进程pid文件
11 pid      logs/nginx.pid;
12
13 # 指定进程可以打开的最大描述符的数目
14 worker_rlimit_nofile 65535;
```

user

user user [group];

- user: 指定可以运行Nginx的用户
- group: 指定可以运行Nginx的用户组 (可选项)

如果user指令不配置或者配置为user nobody nobody, 则默认所有用户都可以启动Nginx进程。

该属性也可以在编译的时候指定, 语法如下`./configure --user=user --group=group`,如果两个地方都进行了设置, 最终生效的是配置文件中的配置。

worker_processes

master_process on|off (默认on)

- 是否以master/worker方式进行工作, 在实际的环境中 nginx是以一个master进程管理多个worker进程的方式运行的。
- 关闭后nginx就不会fork出worker子进程来处理请求, 而是用master进程自身来处理请求, 即使使用"worker_processes number"所指定进程数;

worker_processes 个数;

- 指定工作进程的个数, 默认是1个。
- 具体可以根据服务器cpu数量进行设置, 比如cpu有4个, 可以设置为4。
- 如果不知道cpu的数量, 可以设置为auto, nginx会自动判断服务器的cpu个数, 并设置相应的进程数。

```
[root@localhost location1]# ps -aux | grep nginx
root      5138  0.0  0.0 20628 1412 ?        Ss   22:03   0:00 nginx: master process /usr/
nobody    5202  0.0  0.0 23076 1672 ?        S    22:09   0:00 nginx: worker process
nobody    5203  0.0  0.0 23076 1440 ?        S    22:09   0:00 nginx: worker process
nobody    5204  0.0  0.0 23076 1672 ?        S    22:09   0:00 nginx: worker process
root      5226  0.0  0.0 112668 972 pts/0    S+   22:10   0:00 grep --color=auto nginx
@掘金技术社区
```

当worker_process设置为3时

其他

daemon on|off (默认值on)

- 是否以守护进程(脱离Terminal在后台运行)的方式运行nginx, 关闭守护进程执行的方式可以让我们方便调试nginx

pid PID文件路径 (例如: logs/nginx.pid) ;

- 用来配置Nginx当前master进程的进程号ID存储的文件路径
- 该属性可以通过`./configure --pid-path=PATH`来指定

error_log 文件路径 [日志级别];

- 记录错误信息的日志
- 日志级别的值有: debug|info|notice|warn|error|crit|alert|emerg, 翻译过来为试|信息|通知|警告|错误|临界|警报|紧急
- 建议大家设置的时候不要设置成info以下的等级, 因为会带来大量的磁盘I/O消耗, 影响Nginx的性能
- 该属性可以通过`./configure --error-log-path=PATH`来指定

events块

events块的主要功能:

- 配置Nginx服务器与用户的网络连接。
- 这一部分的指令对Nginx服务器的性能影响较大，在实际配置中应该根据实际情况灵活调整。

accept_mutex on | off;

- 当某一时刻只有一个连接到来时，多个睡眠进程会被同时叫醒，但只有一个进程可获得连接。如果每次唤醒的进程数目太多，会影响一部分系统性能。
- 默认是开启状态，开启后将会对多个Nginx进程接收连接进行序列化，防止多个进程对连接的争抢

multi_accept on|off;

- 设置是否允许同时接收多个网络连接
- 如果multi_accept被禁止了，nginx一个工作进程只能同时接受一个新的连接

worker_connections 数字;

- 设置允许每一个worker process（工作进程）的最大连接数，当每个工作进程接受的连接数超过这个值时将不再接收连接
- 默认值为512
- 当所有的工作进程都接收满时，连接进入logback，logback满后连接被拒绝

use 网络IO模型;

- method可选择的内容有：select、poll、kqueue、epoll、rtsig、/dev/poll以及eventport
- 使用linux内核在6以上，就是为了能使用epoll，提高Nginx的性能

http块

http块主要定义与http服务相关的配置

http全局

include mime.types;

- 包含进HTTP的ContentType
- 可以自定义一些ContentType，包含进来

default_type application/octet-stream;

- 配置默认响应类型，如果不加此指令，默认值为text/plain
- 此指令还可以在http块、server块或者location块中进行配置

公有配置

add_header name value [always];

- 功能：添加指定的响应头和响应值。
- 位置：http, server, location

keepalive

keepalive_timeout timeout [header_timeout]

- 配置连接超时时间，Nginx与用户建立会话连接后，超过多少时间后断开
- 为什么要使用keepalive?
 - HTTP是一种无状态协议，客户端向服务端发送一个TCP请求，服务端响应完毕后断开连接。
 - 如果客户端在短时间内向服务端发送多个请求，对于每个请求都建立一个TCP链接，那么将会产生TCP连接爆炸
 - Keepalive指定了在一段时间内都保持连接状态，可以复用TCP链路
- 例如：keepalive_timeout 120s 100s: 下面配置的含义是，在服务器端保持连接的时间设置为120 s，发给用户端的应答报文头部中Keep-Alive域的超时时间设置为100 s。
- 位置：http, server, location

keepalive_requests number;

- 设置一个keep-alive连接使用的次数
- 默认是100
- 位置：http, server, location

resolver_timeout time;

TOP

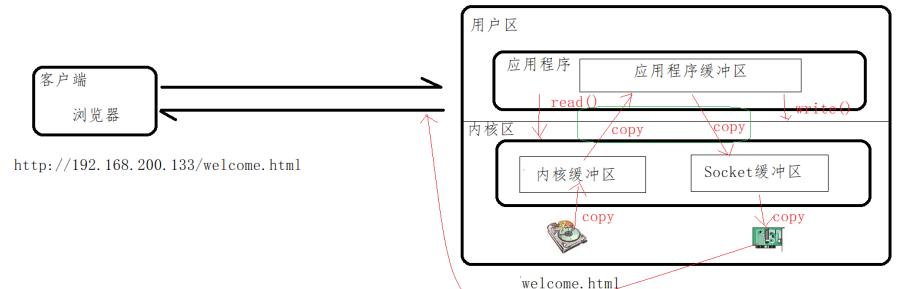
- 解析超时时间
- 默认值：30s
- 使用字段：http, server, location
- 例子：resolver_timeout 5s;

静态资源优化配置

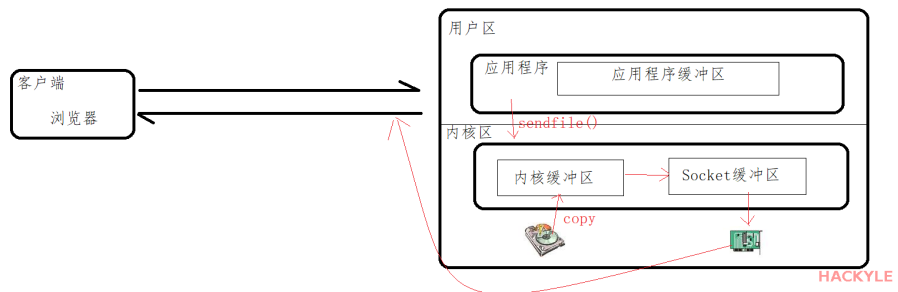
sendfile on | off（默认值）；

- 开启高效的文件传输模式
- 等价于“0拷贝思想”
- 位置：http, server, location

未使用sendfile的处理流程



使用sendfile的处理流程



sendfile_max_chunk 数据量大小size（单位kb）；

- 设置sendfile最大数据量
- size值如果大于0，Nginx进程的每个worker process每次调用sendfile()传输的数据量最大不能超过这个值(这里是128k，所以每次不能超过128k)；如果设置为0，则无限制。默认值为0。

tcp_nopush on|off（默认值）；

- 该指令必须在sendfile打开的状态下才会生效，主要是用来提升网络包的传输'效率'
- 主要思想：要发送的数据先放在缓冲区（no push），等缓冲区满了再发
- 位置：http, server, location

tcp_nodelay on（默认值）|off;

- 该指令必须在keep-alive连接开启的情况下才生效，来提高网络包传输的'实时性'
- 主要思想：一有数据就发送，没有任何延时（no delay）
- 位置：http, server, location



- "tcp_nopush"和"tcp_nodelay"“看起来是”互斥的”，那么为什么要把这两个值都打开，这个大家需要知道的是在5.9以后的版本中两者是可以兼容的
- 三个指令都开启的好处是，sendfile可以开启高效的文件传输模式，tcp_nopush开启可以确保在发送到客户端之前数据包已经充分“填满”，这大大减少了网络开销，并加快了文件发送的速度。

- 然后，当它到达最后一个可能因为没有“填满”而暂停的数据包时，Nginx会忽略tcp_nopush参数，然后，tcp_nodelay强制套接字发送数据。
- 由此可知，tcp_nopush可以与tcp_nodelay一起设置，它比单独配置tcp_nodelay具有更强的性能。

最佳实践：在http共有配置参数中全部开启

- sendfile on;
- tcp_nopush on;
- tcp_nodelay on;

静态资源的压缩

功能：将Nginx服务中的静态资源进行压缩，再发送给Client（例如Chrome），减少网络传输量，提高传输速率和效率

基于ngx_http_gzip_module模块（已自动安装）的静态资源压缩

gzip on|off（默认）；

- 开启或者关闭gzip功能
- 以下的gzip相关指令，只有gzip为on，才有效果

gzip_types mime-type ...;

- 指定对那种类型的文件进行压缩
- 多个类型之间使用空格隔开，可以使用通配符（*：表示全部类型）
- 默认值：gzip_types text/html;
- 例如：

```
1 http{
2     gzip on; #开启
3     gzip_types application/javascript; #指定对JS文件进行压缩
4 }
```

gzip_comp_level level;

- 设置Gzip压缩程度，级别从1-9。数字越低，压缩程度越低，压缩效率越高
- 默认值：gzip_comp_level 1;

gzip_vary on|off（默认值）；

- 用于设置使用Gzip进行压缩发送是否携带“Vary:Accept-Encoding”头域的响应头部。
- 主要是告诉接收方，所发送的数据经过了Gzip压缩处理



gzip_buffers number size;

- 指定处理请求压缩的缓冲区数量和大小。
- 默认值：gzip_buffers 32 4k | 16 8k;
- number：指定Nginx服务器向系统申请缓存空间个数，size指的是每个缓存空间的大小。
- 主要实现的是申请number个每个大小为size的内存空间。这个值的设定一般会和服务器的操作系统有关，所以建议此项不设置，使用默认值即可。

gzip_disable regex ...;

- 针对不同种类客户端发起的请求，可以选择性地开启和关闭Gzip功能。
- 用来排除一些明显不支持Gzip的浏览器，例如排除IE：gzip_disable "MSIE [1-6]\.";

gzip_http_version 1.0|1.1（默认值）；

- 设定gzip支持的HTTP协议版本

TOP

gzip_min_length length;

- 要发送的数据量超过了length, 才会开启Gzip压缩功能
- 默认值: gzip_min_length 20; #单位KB

gzip_proxied off (默认值) | expired | no-cache | no-store | private | no_last_modified | no_etag | auth|any;

- 设置是否对服务端返回的结果进行Gzip压缩
- off - 关闭Nginx服务器对后台服务器返回结果的Gzip压缩
- expired - 启用压缩, 如果header头中包含 "Expires" 头信息
- no-cache - 启用压缩, 如果header头中包含 "Cache-Control:no-cache" 头信息
- no-store - 启用压缩, 如果header头中包含 "Cache-Control:no-store" 头信息
- private - 启用压缩, 如果header头中包含 "Cache-Control:private" 头信息
- no_last_modified - 启用压缩,如果header头中不包含 "Last-Modified" 头信息
- no_etag - 启用压缩,如果header头中不包含 "ETag" 头信息
- auth - 启用压缩,如果header头中包含 "Authorization" 头信息
- any - 无条件启用压缩

综合实例

```

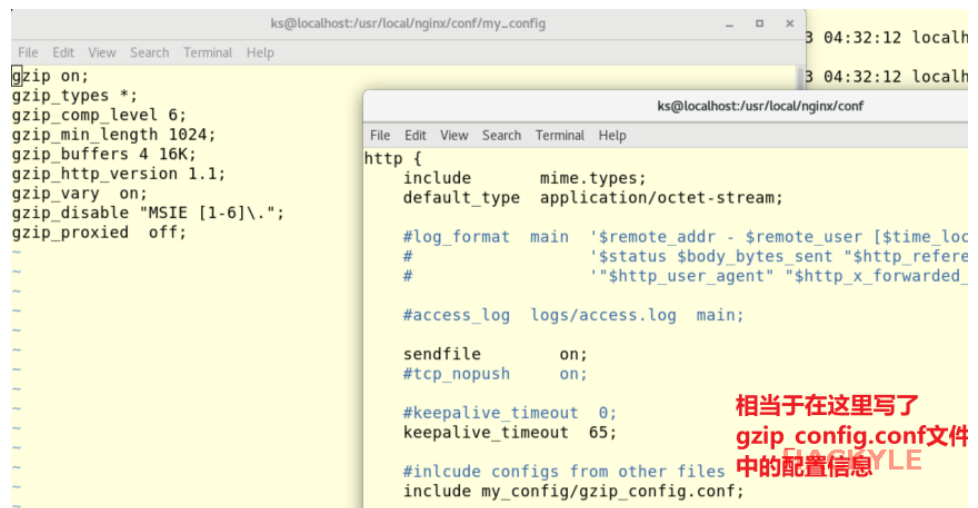
1 | gzip on;                #开启gzip功能
2 | gzip_types *;          #压缩源文件类型,根据具体的访问资源类型设定
3 | gzip_comp_level 6;    #gzip压缩级别
4 | gzip_min_length 1024; #进行压缩响应页面的最小长度,content-length
5 | gzip_buffers 4 16K;  #缓存空间大小
6 | gzip_http_version 1.1; #指定压缩响应所需要的最低HTTP请求版本
7 | gzip_vary on;        #往头信息中添加压缩标识
8 | gzip_disable "MSIE [1-6]\."; #对IE6以下的版本都不进行压缩
9 | gzip_proxied off;    #nginx作为反向代理压缩服务端返回数据的条件

```

这个配置可能再多处都要使用, 我们将其抽取为独立的配置文件, 方便在多处进行 “include”

Step1: vim %nginx所在目录%/config/my_config/gzip_config.conf

Step2: 在Nginx主配置目录导入该文件: vim %nginx所在目录%/config/nginx.conf



Gzip和sendfile共存问题

背景

- 在开启sendfile以后, 在读取磁盘上的静态资源文件的时候, 可以减少拷贝的次数, 可以不经用户进程将静态文件通过网络设备发送出去, 但是Gzip要想对资源压缩, 是需要经过用户进程进行操作的。
- 如何解决两个设置的共存问题。
- 解决方案
 - 可以使用ngx_http_gzip_static_module模块的gzip_static指令来解决。
 - 开启了gzip_static后, 检查与访问资源同名的.gz文件时, response中以gzip相关的header返回gzip文件的内容。

安装ngx_http_gzip_static_module

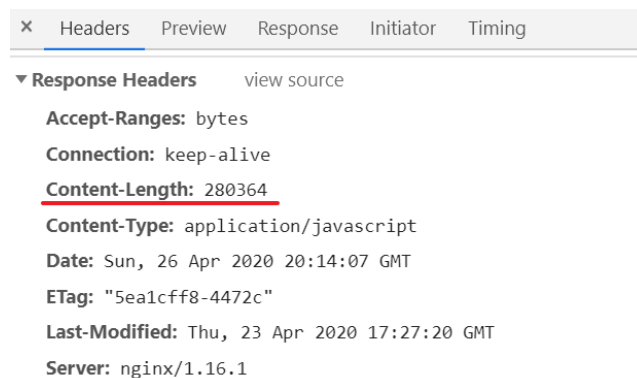
- 默认安装是没有安装ngx_http_gzip_static_module的, 所以我们需要指定该模块重新安装。
- 操纵步骤类似于Nginx的平滑升级
- **主要思想: 重新编译**一个指定了该模块的后的二进制文件, **替换**掉原来的二进制文件即可
- 具体步骤
 1. 查看以前Nginx的编译参数: `nginx -V`
 2. 备份以前安装目录中sbin下的二进制文件, 防止安装失败后进行恢复: `mv nginx nginx_old`
 3. 进入数据包目录: `cd /home/ks/nginx-版本号`
 4. 清空之前编译的内容: `make clean`
 5. 指定安装参数: `./configure --with-http_gzip_static_module`
 6. 遍历: `make`
 7. 将objs目录下的nginx二进制执行文件移动到nginx安装目录下的sbin目录中: `mv objs/nginx /usr/local/nginx/sbin`
 8. 执行更新命令: `make upgrade` #注意当前所在目录是在Nginx数据包目录下

语法: `gzip_static on | off (默认值) | always;`

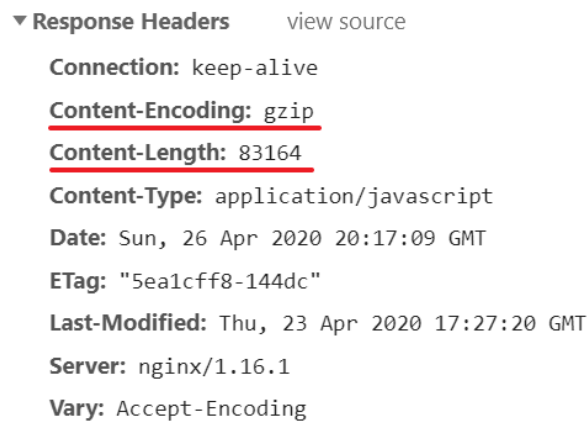
位置: `http、server、location`

测试

- 直接访问`http://192.168.200.133/jquery.js`



- 使用gzip命令进行压缩: `gzip jqueryjs`
- 再次访问`http://192.168.200.133/jquery.js`



缓存

Nginx的web缓存服务主要是使用`ngx_http_proxy_module`模块相关指令集来完成

proxy_cache_path path [levels=number] keys_zone=zone_name:zone_size [inactive=time] [max_size=size];

- 设置缓存文件的存放路径
- 位置: `http`
- 参数释义
 - path: 缓存路径, 例如: `/usr/local/proxy_cache`
 - levels: 指定该缓存空间对应的目录, 最多可以设置3层
 - levels=1:2 缓存空间有两层目录, 第一次是1个字母, 第二次是2个字母
 - 举例说明:
 - itheima[key]通过MD5加密以后的值为 43c8233266edce38c2c9af0694e2107d
 - levels=1:2 最终的存储路径为/usr/local/proxy_cache/d/07
 - levels=2:1:2 最终的存储路径为/usr/local/proxy_cache/7d/0/21
 - levels=2:2:2 最终的存储路径为??/usr/local/proxy_cache/7d/10/e2

TOP

- `keys_zone`: 为这个缓存区设置名称和指定大小。例如: `keys_zone=itcast:200m` 缓存区的名称是itcast, 大小为200M, 1M大概能存储8000个keys
- `inactive`: 指定缓存的数据多次时间未被访问就将被删除。例如: `inactive=1d` #缓存数据在1天内没有被访问就会被删除
- `max_size`: 设置最大缓存空间, 如果缓存空间存满, 默认会覆盖缓存时间最长的资源。例如: `max_size=20g`
- 综合实例

```
1 http { #次配置在http块中
2     proxy_cache_path /usr/local/proxy_cache keys_zone=itcast:200m levels=1:2:1 in
3 }
```

`proxy_cache zone_name | off (默认) ;`

- 用来开启或关闭代理缓存, 如果是开启则需要指定缓存区的名称
- `zone_name`: 缓存区的名字
- 位置: http、server、location

`proxy_cache_key key;`

- 该指令用来设置web缓存的key值, Nginx会根据key值MD5哈希存缓存
- 默认值: `proxy_cache_key $scheme$proxy_host$request_uri;`
- 位置: http、server、location

`proxy_cache_valid [code ...] time;`

- 该指令用来对不同返回状态码的URL设置不同的缓存时间
- 位置: http、server、location
- 例如
 - `proxy_cache_valid 200 302 10m;` #响应为200和302的URL设置10分钟缓存
 - `proxy_cache_valid 404 1m;` #响应为404的URL设置1分钟缓存
 - `proxy_cache_valid any 1m;` #对所有响应状态码的URL都设置1分钟缓存

`proxy_cache_min_uses number;`

- 该指令用来设置资源被访问多少次后被缓存
- 默认值: `proxy_cache_min_uses 1;`
- 位置: http、server、location

`proxy_cache_methods GET|HEAD|POST;`

- 该指令用户设置缓存哪些HTTP方法
- 默认值: `proxy_cache_methods GET HEAD;` #默认缓存HTTP的GET和HEAD方法, 不缓存POST方法。
- 位置: http、server、location

ResponseHeader中的缓存

`expires [modified] time | epoch | max | off (默认值) ;`

- 功能: 控制缓存
- 参数
 - `time`: 如果为整数或0, 则表明指定有效期, Cache-Control的值为`max-age=time`; 如果是负数, 则表明即时失效, Cache-Control则为no-cache
 - `epoch`: 指定Expires的值为'1 January,1970,00:00:01 GMT'(1970-01-01 00:00:00), Cache-Control的值no-cache
 - `max`: 指定Expires的值为'31 December2037 23:59:59GMT' (2037-12-31 23:59:59), Cache-Control的值为10年
 - `off`: 默认不缓存。
- 位置: http、server、location

Cache-Control: 告诉所有的缓存机制是否可以缓存及哪种类型。

取值枚举

- `must-revalidate`: 可缓存但必须再向源服务器进行确认
- `no-cache`: 缓存前必须确认其有效性
- `no-store`: 不缓存请求或响应的任何内容
- `no-transform`: 代理不可更改媒体类型
- `public`: 可向任意方提供响应的缓存
- `private`: 仅向特定用户返回响应

TOP

- proxy-revalidate: 要求中间缓存服务器对缓存的响应有效性再进行确认
- max-age=<秒>: 响应最大Age值
- s-maxage=<秒>: 公共缓存服务器响应的最大Age值

通过add_header指控制Response Header中的数据。例子: 设定响应头中Cache-Control为public:
add_header Cache-Control public;

清除缓存

方式一: 删除对应的缓存目录

方式二: 使用第三方扩展模块 (ngx_cache_purge, 需要安装)

ngx_cache_purge安装过程

1. 下载ngx_cache_purge模块对应的资源包: ngx_cache_purge-2.3.tar.gz
2. 解压缩: tar -zxf ngx_cache_purge-2.3.tar.gz
3. 修改文件夹名称, 方便后期配置: mv ngx_cache_purge-2.3 purge
4. 查询Nginx的配置参数: nginx -V
5. 进入Nginx的安装目录, 使用./configure进行参数配置: ./configure --add-module=/root/nginx/module/purge
6. 编译: make
7. 备份以前的二进制文件: mv /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginxold
8. 将编译后的objs中的nginx拷贝到nginx的sbin目录下: cp objs/nginx /usr/local/nginx/sbin
9. 使用make进行升级: make upgrade
10. 在nginx配置文件中进行如下配置

```

1 | server{
2 |     location ~/purge(/.*) {
3 |         proxy_cache_purge itcast itheima;
4 |     }
5 | }
```

不缓存

proxy_no_cache string ...;

- 该指令是用来定义不将数据进行缓存的条件
- 位置: http、server、location
- 配置实例: proxy_no_cache \$cookie_nocache \$arg_nocache \$arg_comment;

proxy_cache_bypass string ...;

- 该指令是用来设置不从缓存中获取数据的条件。
- 位置: http、server、location
- 配置实例: proxy_cache_bypass \$cookie_nocache \$arg_nocache \$arg_comment;

\$cookie_nocache: 指的是当前请求的cookie中键的名称为nocache对应的值

\$arg_nocache和\$arg_comment: 指的是当前请求的参数中属性名为nocache和comment对应的属性值

实例: 配置不缓存的资源

```

1 | server{
2 |     listen 8080;
3 |     server_name localhost;
4 |     location / {
5 |         if ($request_uri ~ /\.js$){
6 |             set $nocache 1;
7 |         }
8 |         proxy_no_cache $nocache $cookie_nocache $arg_nocache $arg_comment;
9 |         proxy_cache_bypass $nocache $cookie_nocache $arg_nocache $arg_comment;
```

TOP

```
10 |     }  
11 | }
```

log

access_log on/off;

- 开启或关闭日志记录功能
- 位置：http, server, location

access_log path[format[buffer=size]]

- 指定access_log日志的文件格式
- 默认值：access_log logs/access.log combined;

log_format name [escape=default\|json\|none] string....;

- 指定日志的输出格式
- 默认值：log_format combined "...";

gzip on | off;

开启关闭压缩日志文件

负载均衡



实现方式

- 用户手动选择：在网站主页上面提供不同线路、不同服务器链接方式，让用户来选择自己访问的具体服务器，来实现负载均衡。

普通下载地址	
通用网络下载	电信网络下载
联通网络下载	移动网络下载

- DNS轮询方式：采用简单轮询方法

当前分配的DNS服务器是: dns19.hichina.com, dns20.hichina.com

添加记录

导入/导出

请求量统计

新手引导

全部记录

精确搜索

输入关键字

高级搜索

<input type="checkbox"/>	主机记录	记录类型	解析线路(isp)	记录值	TTL	状态	备注	操作
<input type="checkbox"/>	www	A	默认	192.168.200.146	10分钟	正常		修改 暂停 删除 备注
<input type="checkbox"/>	_dnsauth	TXT	默认	202005180000003j44frpuioy25mt1ufat91gynvjsq4r93gpewt7mx9ph5q	10分钟	正常		修改 暂停 删除 备注
<input type="checkbox"/>	www	A	默认	192.168.200.133	10分钟	正常		修改 暂停 删除 备注
<input type="checkbox"/>	@	A	默认	192.168.200.133	10分钟	正常		修改 暂停 删除 备注

HACKYLE

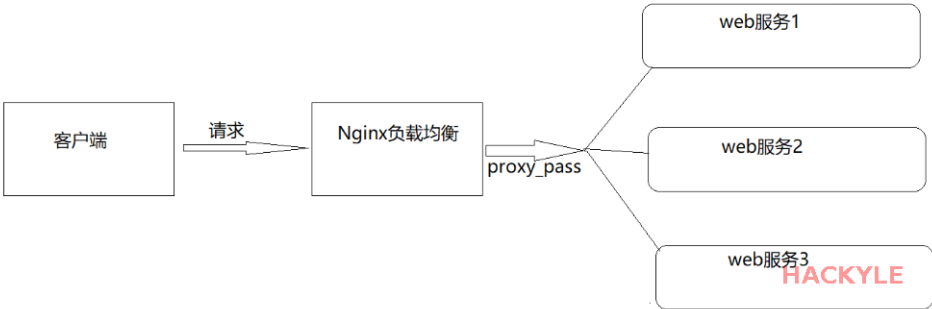
每次访问，都需要情况DNS缓存（ipconfig/flushdns），才能看到效果

- 第四层负载均衡：在OSI七层模型中的传输层，主要是基于IP+PORT的负载均衡
 - 硬件：F5 BIG-IP、Radware等
 - 软件：LVS、Nginx、Hayproxy等
- 第七层负载均衡：在应用层，主要是基于虚拟的URL或主机IP的负载均衡
 - 软件：Nginx、Hayproxy等
 - 四层负载均衡数据包是在底层就进行了分发，而七层负载均衡数据包则在最顶端进行分发，所以四层负载均衡的效率比七层负载均衡的要高。
 - 三层负载均衡不识别域名，而七层负载均衡识别域名。
- 第二层负载均衡：在数据链路层基于mac地址来实现负载均衡
- 第三层负载均衡：在网络层一般采用虚拟IP地址的方式实现负载均衡
- 最佳实践：四层负载(LVS)+七层负载(Nginx)

第七层负载均衡

```
1 upstream 取个名字 {
2     server 服务的域名或者IP地址 [paramerters]
3     server 服务的域名或者IP地址 [paramerters] ... #可以配置多个服务
4 }
```

- 定义一组用于负载均衡的服务器
- 它们可以是监听不同端口的服务器，并且也可以是同时监听TCP和Unix socket的服务器。服务器可以指定不同的权重，默认为1。
- 位置：http



paramerters-控制负载服务的状态

- down：当前的server暂时不参与负载均衡。该状态一般会对需要停机维护的服务器进行设置。

```

upstream backend{
    server 192.168.200.146:9001 down;
    server 192.168.200.146:9002
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

- backup: 预留的备份服务器。当主服务器不可用时，将用来传递请求。

```

upstream backend{
    server 192.168.200.146:9001 down;
    server 192.168.200.146:9002 backup;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

- max_fails=number: 设置允许请求代理服务器失败的次数，默认为1。
- fail_timeout=time: 设置经过max_fails失败后，服务暂停的时间，默认是10秒。

```

upstream backend{
    server 192.168.200.133:9001 down;
    server 192.168.200.133:9002 backup;
    server 192.168.200.133:9003 max_fails=3 fail_timeout=15;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

HACKYLE

- max_conns: 限制最大的接收连接数

第七层的均衡策略

负载均衡策略

- 轮询: 默认方式
- weight: 权重方式
- ip_hash: 依据ip分配方式
- least_conn: 依据最少连接方式
- url_hash: 依据URL分配方式
- fair: 依据响应时间方式

weight: 权重方式

- weight=number
- number默认为1，权重数据越大，被分配到请求的几率越大；
- 该权重值，主要是针对实际工作环境中不同的后端服务器硬件配置进行调整的，所有此策略比较适合服务器的硬件配置差别比较大的情况。

TOP

```
upstream backend{
    server 192.168.200.146:9001 weight=10;
    server 192.168.200.146:9002 weight=5;
    server 192.168.200.146:9003 weight=3;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}
```

ip_hash: 依据ip分配方式

- 当对后端的多台动态应用服务器做负载均衡时，ip_hash指令能够将某个客户端IP的请求通过哈希算法定位到同一台后端服务器上。
- 这样，当来自某一个IP的用户在后端Web服务器A上登录后，在访问该站点的其他URL，能保证其访问的还是后端web服务器A。
- 注意：这样无法保证100%负载均衡，因为可能所有IP都hash到了一台服务器上。此外，服务的权重将失效。

```
upstream backend{
    ip_hash;
    server 192.168.200.146:9001;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}
```

least_conn: 依据最少连接方式

- 最少连接，把请求转发给连接数较少的后端服务器。
- 轮询算法是把请求平均的转发给各个后端，使它们的负载大致相同；但是，有些请求占用的时间很长，会导致其所在的后端负载较高。这种情况下，least_conn这种方式就可以达到更好的负载均衡效果。
- 此策略适合请求处理时间长短不一造成服务器过载的情况。

```
upstream backend{
    least_conn;
    server 192.168.200.146:9001;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}
```

url_hash: 依据URL分配方式

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

TOP


```

upstream backend{
    hash &request_uri;
    server 192.168.200.146:9001;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

fair: 依据响应时间方式

- 第三方模块提供的负载策略
- 可以根据页面大小、加载时间长短智能的进行负载均衡。
- 模块地址: <https://github.com/gnosek/nginx-upstream-fair>, 采取类似于Nginx升级的方式载入该模块到Nginx

第四层负载均衡

Nginx在1.9之后, 增加了一个stream模块, 用来实现四层协议的转发、代理、负载均衡等。stream模块的用法跟http的用法类似, 允许我们配置一组TCP或者UDP等协议的监听, 然后通过proxy_pass来转发我们的请求, 通过upstream添加多个后端服务, 实现负载均衡。

四层协议负载均衡的实现, 一般都会用到LVS、HAProxy、F5等, 要么很贵要么配置很麻烦, 而Nginx的配置相对来说更简单, 更能快速完成工作。

添加stream模块的支持

1. nginx.conf的基本结构

2. 全局配置块

1. user
2. worker_processes
3. 其他

3. events块

4. http块

1. http全局
2. 公有配置
 1. keepalive
 2. 静态资源优化配置
 3. 静态资源的压缩
 1. 综合实例
 2. Gzip和sendfile共存问题
4. 缓存
 1. ResponseHeader中的缓存
 2. 清除缓存

- Nginx默认是没有编译这个模块的, 需要使用到stream模块, 那么需要在编译的时候加上`--with-stream`。
- 完成添加`--with-stream`的实现步骤:
 1. 将原有/usr/local/nginx/sbin/nginx进行备份
 2. 拷贝nginx之前的配置信息
 3. 在nginx的安装源码进行配置指定对应模块 ./configure --with-stream
 4. 通过make模板进行编译
 5. 将objs下面的nginx移动到/usr/local/nginx/sbin下
 6. 在源码目录下执行 make upgrade进行升级, 这个可以实现不停机添加新模块的功能

```

1  stream 取个名字 {
2      server 服务的域名或者IP地址 [parameters]
3      server 服务的域名或者IP地址 [parameters] ... #可以配置多个服务
4  }

```

- 定义一组用于负载均衡的服务器
- 它们可以是监听不同端口的服务器, 并且也可以是同时监听TCP和Unix socket的服务器。服务器可以指定不同的权重, 默认为1。
- 位置: 与http同级

实例

```

1  worker_processes 1;
2
3  events {
4      worker_connections 1024;
5  }

```


TOP

版权声明： 非明确标注皆为原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上本文链接及此声明。
原文链接：<https://blog.hackyle.com/article/server/nginx-conf-part1>

16/17

Input comment, please

p

0 words 

SUBMIT

RESET

© Copy Right: 2022 HACKYLE. All Rights Reserved
Designed and Created by HACKYLE SHAWE
备案号：浙ICP备20001706号-2

TOP