Nginx介绍

Nginx简介

- Nginx是Igor Sysoev为俄罗斯访问量第二的rambler.ru站点设计开发的。从2004年发布至今，凭借开源的力量， 已经接近成熟与完善。
- Nginx功能丰富，可作为HTTP服务器，也可作为反向代理服务器，邮件服务器。支持FastCGI、SSL、Virtual Host、URL Rewrite、Gzip等功能。并且支持很多第三方的模块扩展。
- 官方：http://www.nginx.org/

Nginx特点

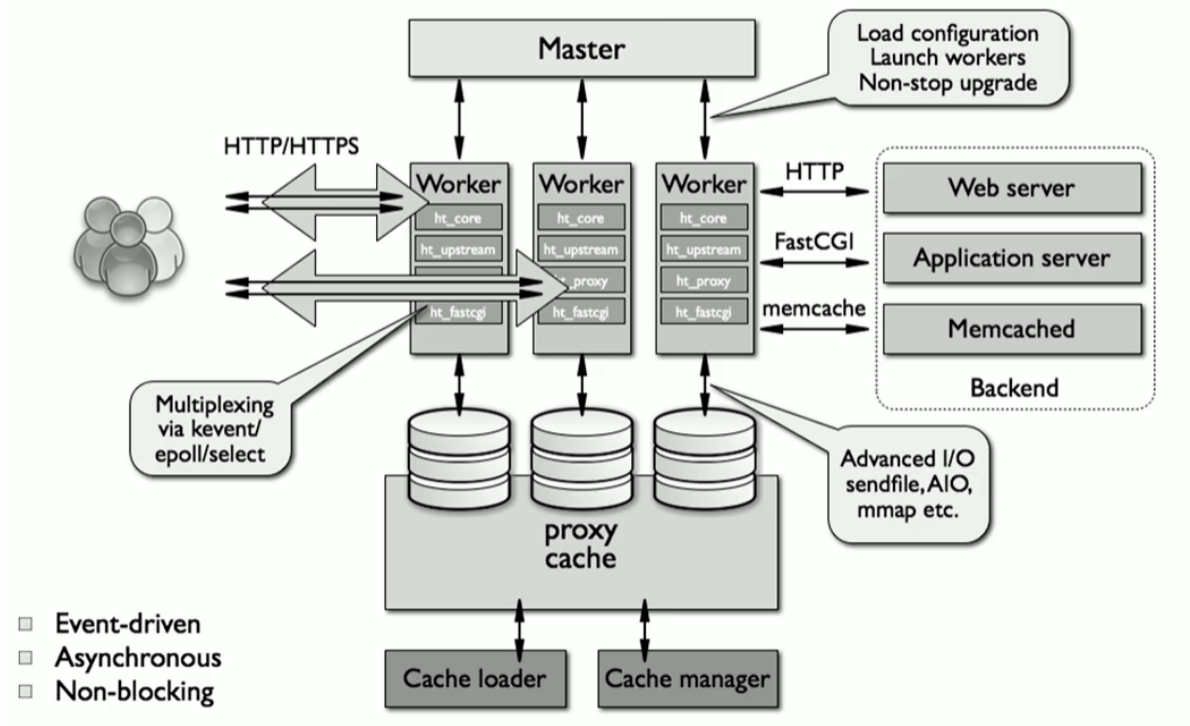
- 支持高并发，消耗内存资源少
- 具有多种功能
- - 网站web服务功能
  - 网站负载均衡功能
  - 正向代理反向代理
- 网站缓存功能
- 在多种系统平台都可以部署
- nginx实现网络通讯时使用的是异步网络IO模型：epoll模型，参考博客：https://segmentfault.com/a/1190000003063859#item-3-13

工作原理

- Nginx由内核和一系列模块组成，内核提供web服务的基本功能,如启用网络协议,创建运行环境,接收和分配客户端请求,处 理模块之间的交互。Nginx的各种功能和操作都由模块来实现。Nginx的模块从结构上分为核心模块、基础模块和第三方 模块。
- 1）核心模块： HTTP模块、EVENT模块和MAIL模块
- 2）基础模块： HTTP Access模块、HTTP FastCGI模块、HTTP Proxy模块和HTTP Rewrite模块
- 3）第三方模块： HTTP Upstream Request Hash模块、Notice模块和HTTP Access Key模块及用户自己开发的模块

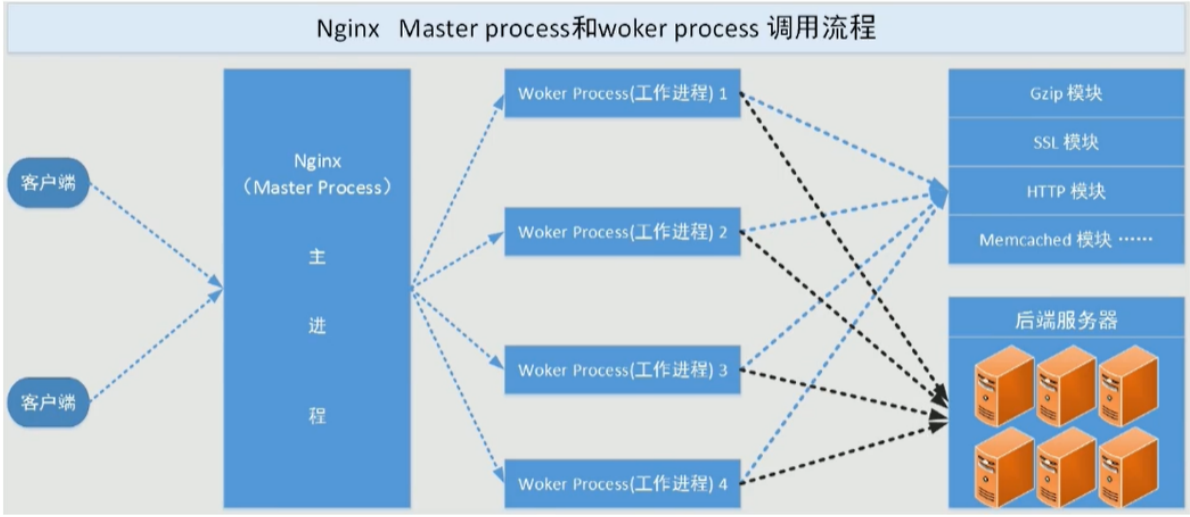
Nginx架构和进程

Nginx架构



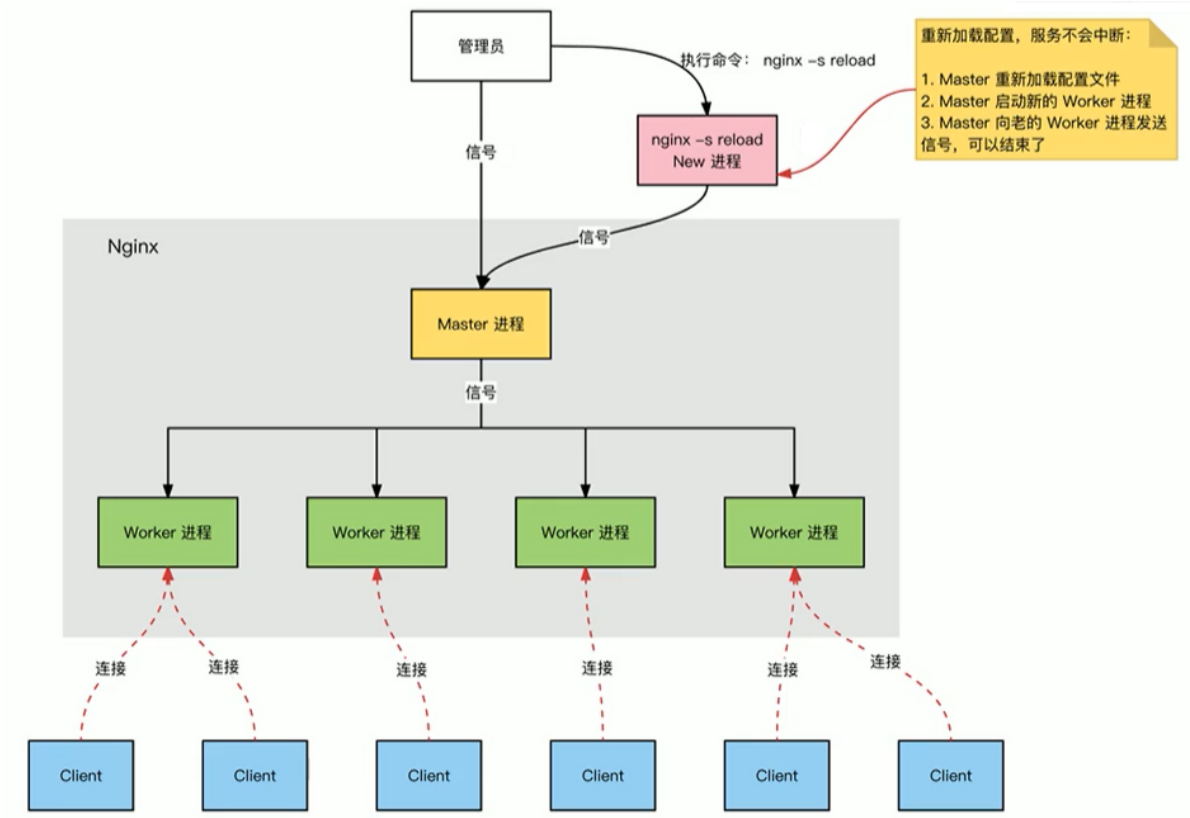
Nginx进程结构

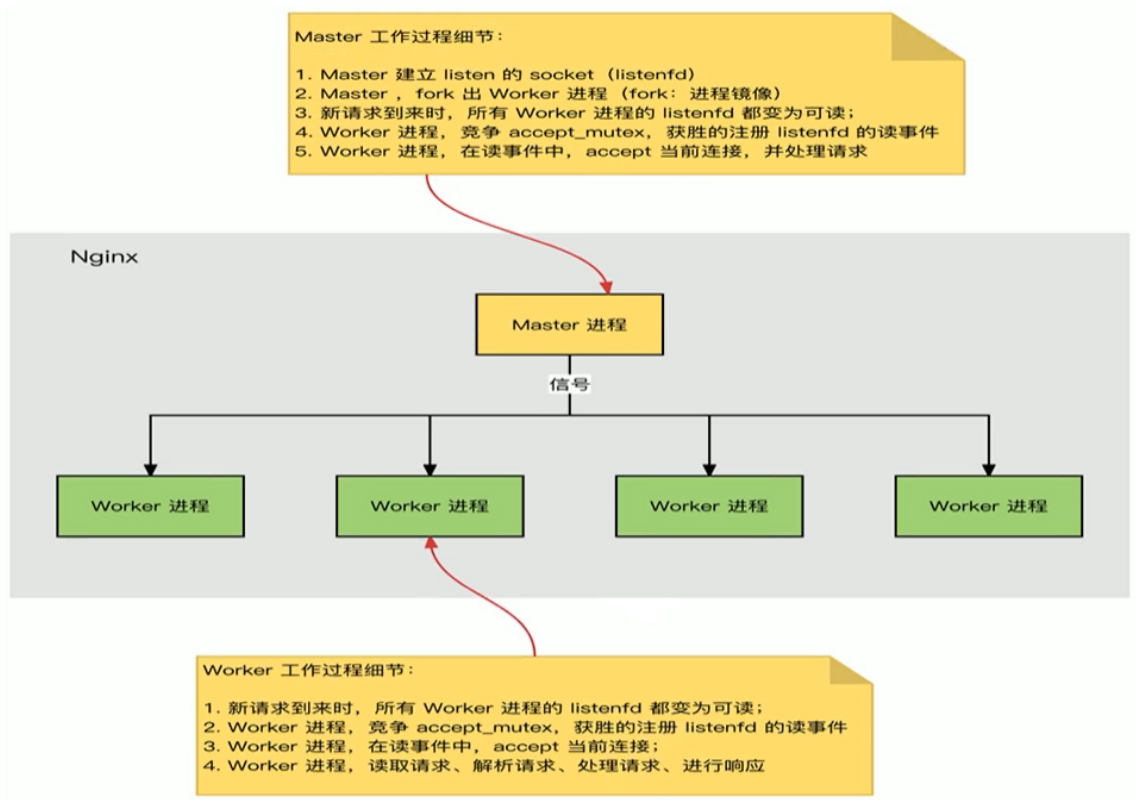
- web请求处理机制
  - 多进程方式:服务器每接收到一个客户端请求就有服务器的主进程生成一个子进程响应客户端，直到用户关闭连接，这样的优势是处理速度快。子进程之间相互独立，但是如果访问过大会导致服务器资源耗尽而无法提供请求。
  - 多线程方式:与多进程方式类似，但是每收到一个客户端请求会有服务进程派生出一个线程来个客户方进行交互，一个线程的开销远远小于一个进程，因此多线程方式在很大程度减轻了web服务器对系统资源的要求，但是多线程也有自己的缺点。即当多个线程位于同一个进程内工作的时候，可以相互访问同样的内存地址空间，所以他们相互影响，一旦主进程挂掉则所有子线程都不能工作了，IIS服务器使用了多线程的方式，需要间隔一段时间就重启一次才能稳定。
- Nginx是多进程组织模型，而且是一个由Master主进程和Worker工作进程组成。



- 主进程(master process)的功能:
  - 对外接口:接收外部的操作(信号)
  - 对内转发:根据外部的操作的不同，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
  - 读取Nginx配置文件并验证其有效性和正确性

    - 建立、绑定和关闭socket连接
    - 按照配置生成、管理和结束工作进程
    - 接受外界指令，比如重启、升级及退出服务器等指令
    - 不中断服务，实现平滑升级，重启服务并应用新的配置
    - 开启日志文件，获取文件描述符
    - 不中断服务，实现平滑升级，升级失败进行回滚处理
    - 编译和处理perl脚本
- 工作进程(worker process的功能:
    - 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 中配置，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
    - 上下文切换的损耗
    - 接受处理客户的请求
    - 将请求依次送入各个功能模块进行处理
    - I/O调用，获取响应数据
    - 与后端服务器通信，接收后端服务器的处理结果
    - 缓存数据，访问缓存索引，查询和调用缓存数据
    - 发送请求结果，响应客户的请求
    - 接收主程序指令，比如重启、升级和退出等



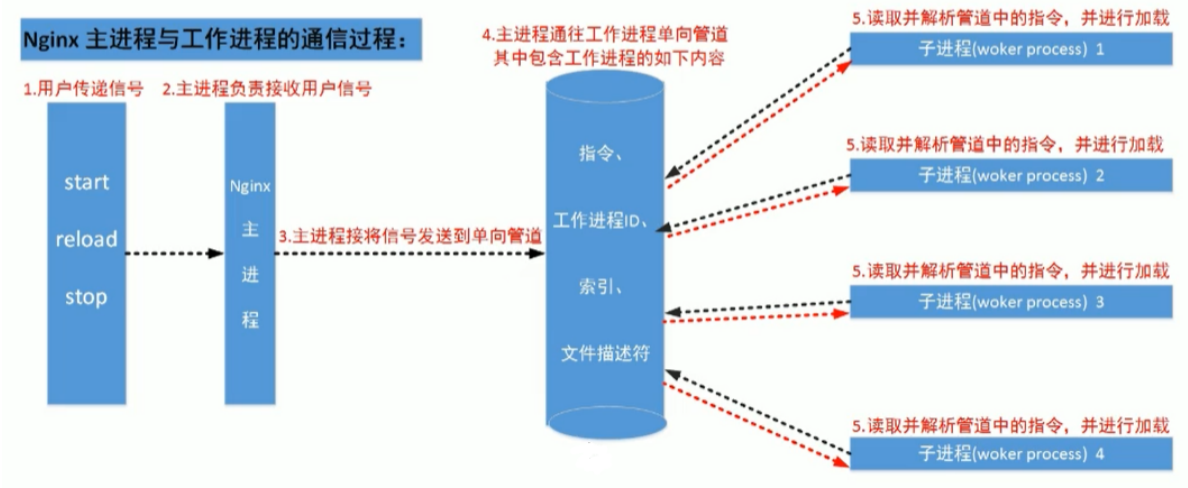


Nginx进程间通信

工作进程是有主进程生成的，主进程由root启用，主进程使用fork()函数，在Nginx服务器启动过程中主进程根据配置文件决定启动工作进程的数量，然后建立一张全局的工作表用于存放当前未退出的所有的工作进程，主进程生成工作进程后会将新生成的工作进程加入到工作进程表中，并建立一个单向的管道并将其传递给工作进程，该管道与普通的管道不同，它是由主进程指向工作进程的单项通道，包含了主进程向工作进程发出的指令、工作进程ID、工作进程在工作进程表中的索引和必要的文件描述符等信息，单向管道，工作进程只能监听内容之后读取指令。 主进程与外界通过信号机制进行通信，当接收到需要处理的信号时，它通过管道向相关的工作进程发送正确的指令，每个工作进程都有能力捕获管道中的可读事件，当管道中有可读事件的时候，工作进程就会从管道中读取并解析指令，然后采取相应的执行动作，这样就完成了主进程与工作进程的交互。

工作进程之间的通信原理基本上和主进程与工作进程之间的通信是一样的，只要工作进程之间能够取得彼此的信息，建立管道即可通信，但是由于工作进程之间是完全隔离的，因此一个进程想要知道另外一个进程的状态信息就只能通过主进程来设置了。

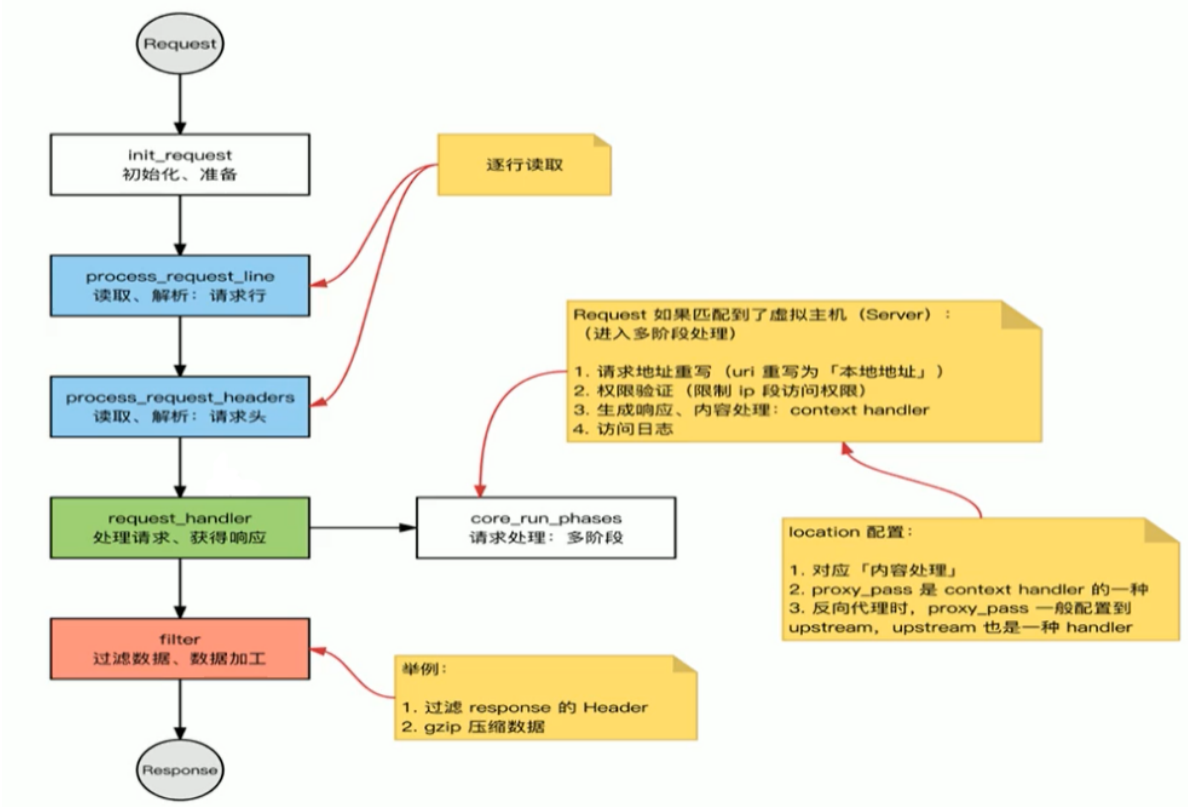
为了实现工作进程之间的交互，主进程在生成工作进程之后，在工作进程表中进行遍历，将该新进程的ID以及针对该进程建立的管道句柄传递给工作进程中的其他进程，为工作进程之间的通信做准备，当工作进程1向工作进程2发送指令的时候，首先在主进程给它的其他工作进程工作信息中找到2的进程ID，然后将正确的指令写入指向进程2的管道，工作进程2捕获到管道中的事件后，解析指令并进行相关操作，这样就完成了工作进程之间的通信。



连接建立和请求处理过程

- Nginx启动时，Master 进程，加载配置文件
- Master进程，初始化监听的socket
- Master进程，fork 出多个Worker进程
- 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

HTTP处理过程



Nginx服务部署安装

yum安装

- yum安装，安装前需要先安装扩展源

```
[root@localhost ~]# yum install epel-release.noarch -y
[root@localhost ~]# yum install nginx -y
```

- 启动nginx

```
[root@localhost ~]# systemctl start nginx.service
[root@localhost ~]# systemctl status nginx
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; disabled; vendor preset: disabled)
   Active: active (running) since 五 2020-09-25 09:39:46 CST; 7s ago
  Process: 7195 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
  Process: 7192 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
  Process: 7190 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
 Main PID: 7197 (nginx)
```

- 关闭防火墙和selinux

```
[root@localhost ~]# systemctl stop firewalld.service
[root@localhost ~]# setenforce 0
```

- 浏览器输入IP地址即可访问

编译安装

- 从官网获取源码包，以1.18.0举例

```
[root@localhost ~]# wget http://nginx.org/download/nginx-1.18.0.tar.gz -P /usr/local/src/
[root@localhost ~]# cd /usr/local/src
[root@localhost src]# tar xzvf nginx-1.18.0.tar.gz
[root@localhost src]# cd nginx-1.18.0
[root@localhost nginx-1.18.0]# ./configure --help
```

- 编译安装

```
[root@localhost nginx-1.18.0]# yum -y install gcc pcre-devel openssl-devel zlib-devel
[root@localhost nginx-1.18.0]# useradd -r -s /sbin/nologin nginx
[root@localhost nginx-1.18.0]# ./configure --prefix=/apps/nginx \
--user=nginx \
--group=nginx \
--with-http_ssl_module \
--with-http_v2_module \
--with-http_realip_module \
--with-http_stub_status_module \
--with-http_gzip_static_module \
--with-pcre \
--with-stream \
--with-stream_ssl_module \
--with-stream_realip_module
[root@localhost nginx-1.18.0]# make -j 2 && make install
[root@localhost nginx-1.18.0]# chown -R nginx.nginx /apps/nginx
[root@localhost nginx-1.18.0]# ln -s /apps/nginx/sbin/nginx /usr/bin/
[root@localhost nginx-1.18.0]# nginx -v
```

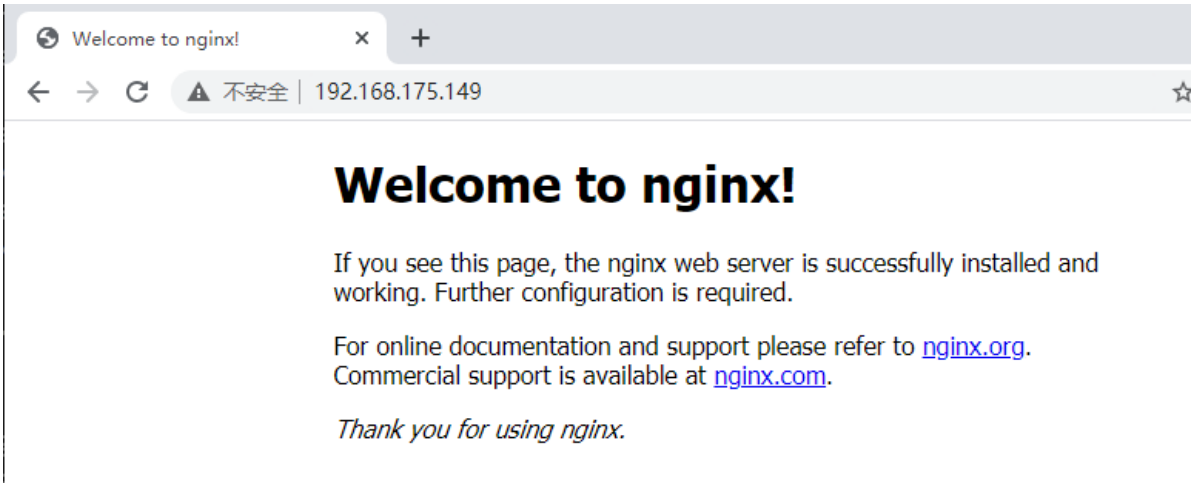
- nginx完成安装以后，有四个主要的目录
  - **conf:**保存nginx所有的配置文件，其中nginx.conf是nginx服务器的最核心最主要的配置文件，其他的.conf则是用来配置nginx相关的功能的，例如fastcgi功能使用的是fastcgi. conf和fastcgi.params两个文件，配置文件一般都有个样板配置文件，是文件名. default结尾，使用的使用将其复制为并将default去掉即可。
  - **html:**目录中保存了nginx服务器的web文件，但是可以更改为其他目录保存web文件，另外还有一个50x的web文件是默认的错误页面提示页面。
  - **logs:**用来保存ngi nx服务器的访问日志错误日志等日志，logs目录可以放在其他路径，比如/var/logs/nginx里面。
  - sbin:保存nginx二进制启动脚本，可以接受不同的参数以实现不同的功能。

```
[root@localhost nginx-1.18.0]# tree /apps/nginx -C -L 1
/apps/nginx
├── conf
├── html
├── logs
└── sbin
```

启动和停止nginx测试访问web界面

- 启动和关闭nginx

```
[root@localhost nginx-1.18.0]# nginx
[root@localhost nginx-1.18.0]# nginx -s stop
```



创建nginx自启动文件

```
# 复制同一版本的nginx的yum安装生成的service文件
[root@localhost ~]# vim /usr/lib/systemd/system/nginx.service
[Unit]
Description=The nginx HTTP and reverse proxy server
Documentation=http://nginx.org/en/docs/
After=network.target remote-fs.target nss-lookup.target
Wants=network-online.target

[Service]
Type=forking
PIDFile=/apps/nginx/run/nginx.pid
ExecStart=/apps/nginx/sbin/nginx -c /apps/nginx/conf/nginx.conf
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s TERM $MAINPID

[Install]
WantedBy=multi-user.target
[root@localhost ~]# mkdir /apps/nginx/run/
[root@localhost ~]# vim /apps/nginx/conf/nginx.conf
pid        /apps/nginx/run/nginx.pid;
```

验证Nginx自启动文件

```
[root@localhost ~]# systemctl daemon-reload
[root@localhost ~]# systemctl enable --now nginx
[root@localhost ~]# ll /apps/nginx/run
总用量 4
-rw-r--r--. 1 root root 5 5月  24 10:07 nginx.pid
```

Nginx目录结构介绍

路径信息	类型信息	作用说明
/etc/logrotate.conf	配置文件	用于日志轮询切割
/etc/nginx/ /etc/nginx/nginx.conf /etc/ngix/conf.d/ /etc/nginx/nginx.conf.default	配置文件配置目录	nginx主配置文件
/etc/nginx/fastcgi_params /etc/nginx/fastcgi_params.default /etc/nginx/scgi_params /etc/nginx/scgi_params.default /etc/nginx/uwsgi_params /etc/nginx/uwsgi_params.default	配置文件	cgi、fastcgi、uwcgi配置文件
/etc/nginx/koi-utf /etc/nginx/koi-win	配置文件	nginx编码映射文件
/etc/nginx/mime.types /etc/nginx/mime.types.default	配置文件	http协议的content-type与扩展名
/usr/lib/systemd/system/nginx.service	配置文件	nginx服务守护进程管理文件
/etc/nginx/modules	目录信息	模块目录
/usr/sbin/nginx	命令信息	nginx模块管理

日志切割

1. /etc/logrotate.d/nginx：可以实现日志切割
2. 日志切割方式一
    1. `mv /var/log/nginx/access.log /var/log/nginx/access_$(date +%F).log`
    2. 重启nginx `systemctl restart nginx`
3. 日志切割方式二，使用专用的文件切割程序--logrotate

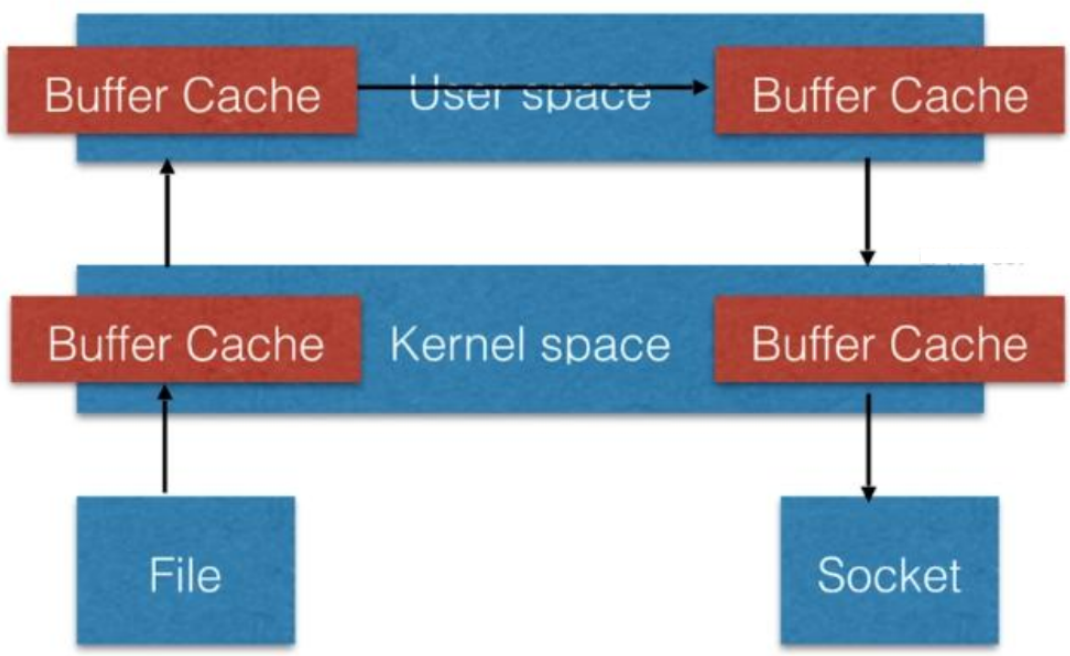
```
[root@localhost logrotate.d]# cat /etc/logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
weekly      # 定义默认日志切割周期


# keep 4 weeks worth of backlogs
rotate 4    # 定义只保留几个切割后的文件

# create new (empty) log files after rotating old ones
create      # 切割后创建出一个相同的源文件，后面可以跟上文件权限、属主、属组

# use date as a suffix of the rotated file
dateext     # 定义角标，扩展名称信息

# uncomment this if you want your log files compressed
#compress   # 是否对切割后的文件进行压缩处理

# RPM packages drop log rotation information into this directory
```

```
include /etc/logrotate.d

# no packages own wtmp and btmp -- we'll rotate them here
/var/log/wtmp {             # 当都对某个文件进行切割配置
    monthly
    create 0664 root utmp
    minsize 1M
    rotate 1
}
/var/log/btmp {
    missingok
    monthly
    create 0600 root utmp
    rotate 1
}
```

配置文件默认参数说明

```
[root@localhost ~]# cp /etc/nginx/nginx.conf{,.bak}
[root@localhost ~]# grep -Ev "#|^$" /etc/nginx/nginx.conf.bak
>/etc/nginx/nginx.conf
[root@localhost ~]# cat /etc/nginx/nginx.conf
====================第一个部分，配置文件的主区域====================
user nginx;                                                    # 定义
worker进程的管理用户
worker_processes auto;                                  # 定义worker进程数，
auto会自动调整为cpu核数
error_log /var/log/nginx/error.log;          # 定义错误日志
pid /run/nginx.pid;                                          # 定义pid文件
include /usr/share/nginx/modules/*.conf;
====================第二个部分，配置文件的事件区域====================

events {
    worker_connections 1024;         # 定义一个worker进程可以同时接受1024个请求
}
====================第三个部分，配置文件的http区域====================
http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request"
'
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
                      # 定义日志格式
    access_log  /var/log/nginx/access.log  main;
                                        # 指定日志文件路径
    sendfile            on;  # 允许sendfile方式传输文件 ，sendfile系统调用在两个文
件描述符之间直接传递数据(完全在内核中操作)，从而避免了数据在内核缓冲区和用户缓冲区之间的拷
贝，操作效率很高，被称之为零拷贝。
    tcp_nopush          on;  # 在sendfile启动下，使用TCP_CORK套接字，当有数据时，
先别着急发送，确保数据包已经装满数据，避免了网络拥塞
    tcp_nodelay         on;  # 接连接保持活动状态
    keepalive_timeout   65;  # 超时时间
    types_hash_max_size 2048;# 连接超时时间
    include             /etc/nginx/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，默认为
text/plain
    include /etc/nginx/conf.d/*.conf;
```

```
    server {
        listen       80 default_server;       # 指定监听的端口
        listen       [::]:80 default_server;
        server_name  _;                                          # 指定网站主机名
        root         /usr/share/nginx/html;  # 定义站点目录的位置
        include /etc/nginx/default.d/*.conf; # 定义首页文件
        location / {
        }
        error_page 404 /404.html;                          # 定义优雅显示页面信息
            location = /40x.html {
        }
        error_page 500 502 503 504 /50x.html;
            location = /50x.html {
        }
    }
}
```



location表达式

- ~ 表示执行一个正则匹配，区分大小写；
- ~* 表示执行一个正则匹配，不区分大小写；
- ^~ 表示普通字符匹配。使用前缀匹配。如果匹配成功，则不再匹配其他location；
- = 进行普通字符精确匹配。也就是完全匹配；
- @ 它定义一个命名的 location，使用在内部定向时，例如 error_page, try_files
- 优先级：=/^~/,~*/常规字符串

```
location = / {
[ configuration A ]
}
location / {
[ configuration B ]
}
location /documents/ {
[ configuration C ]
}
```

```
location ^~ /images/ {
[ configuration D ]
}
location ~* \.(gif|jpg|jpeg)$ {
[ configuration E ]
}
A: 请求 /
B: 请求 index.html
C: 请求 /documents/document.html
D: 请求 /images/1.jpg
E: 请求 /documents/document.gif
```

客户端相关配置

```
keepalive_timeout # 保持连接的超时时长
keepalive_requests # 一次连接允许请求资源的最大数量
keepalive_disable # 对某种浏览器禁用长连接
send_timeout # 向客户端发送响应报文的超时时长
client_body_buffer_size # 接收客户端请求报文的body部分的缓冲区大小
client_body_temp_path # 设定用于存储客户端请求报文的body部分的临时存储路径及子目录结构和数量
limit_rate rate # 限制响应给客户端的传输速率
limit_except # 限制对指定的请求方法之外的其它方法的使用客户端
```

搭建第一个网站

- 编写虚拟主机配置文件

```
[root@www conf.d]# vim /etc/nginx/conf.d/www.conf
server {
    listen      *:8080;
    server_name www.eagle.com;
    location / {
    root /usr/share/nginx/html;
    index ealgeslab.html;
}
}
```

- 编写网站代码（这是开发需要关心的事情，这里以简略内容代替）

```
[root@www ~]# cat /usr/share/nginx/html/ealgeslab.html
OK!第一个网站没有问题
```

- 赋予网站代码权限

```
[root@www ~]# chmod 777 /usr/share/nginx/html/ealgeslab.html
```

- 编写hosts文件，做好域名解析(如果windows需要访问也要在这个目录下的hosts文件做相同操作C:\Windows\System32\drivers\etc)

```
[root@www ~]# cat /etc/hosts
...
192.168.33.133 www.eagle.com
...
[root@www ~]# ping www.eagle.com
PING www.eagle.com (192.168.33.133) 56(84) bytes of data.
64 bytes from www.eagle.com (192.168.33.133): icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from www.eagle.com (192.168.33.133): icmp_seq=2 ttl=64 time=0.049 ms
......
```

- 重启nginx服务（使用reload平滑重启，以下两种方式不要混用）

```
[root@www html]# systemctl reload nginx.service
[root@www html]# nginx -s reload
```

- 访问测试

```
[root@www ~]# curl www.eagle.com:8080
OK!第一个网站没有问题
```

利用nginx服务搭建多个网站

- 准备好配置文件

```
[root@www conf.d]# ll
总用量 12
-rw-r--r--. 1 root root 107 9月  25 16:45 bbs.conf
-rw-r--r--. 1 root root 109 9月  25 16:45 blog.conf
-rw-r--r--. 1 root root 107 9月  25 16:44 www.conf
[root@www conf.d]# cat *.conf
server {
    listen      *:8080;
    server_name bbs.eagle.com;
    location / {
    root /html/bbs;
    index index.html;
}
}
server {
    listen      *:8080;
    server_name blog.eagle.com;
    location / {
    root /html/blog;
    index index.html;
}
}
server {
    listen      *:8080;
    server_name www.eagle.com;
    location / {
    root /html/www;
    index index.html;
}
}
```

- 准备站点目录以及首页文件

```
[root@www bbs]# mkdir -p /html/{www,bbs,blog}
[root@www bbs]# for name in {www,bbs,blog};do echo "<h1> $name </h1>" >
/html/$name/index.html;done;
```

- 赋予网站代码权限

```
[root@www ~]# chmod -R 777 /html
[root@www ~]# setenforce 0
```

- 准备域名解析

```
[root@www bbs]# cat /etc/hosts
192.168.33.133 www.eagle.com bbs.eagle.com blog.eagle.com
```

- 重启nginx

```
[root@www html]# systemctl reload nginx.service
```

- 访问测试

```
[root@www ~]# curl www.eagle.com:8080
<h1> www </h1>
[root@www ~]# curl bbs.eagle.com:8080
<h1> bbs </h1>
[root@www ~]# curl blog.eagle.com:8080
<h1> blog </h1>
```

nginx相关模块

ngx_http_access_module

- 实现基于ip的访问控制功能
- 代码实例

```
server {
    ...
    deny  192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny  all;
}
```

ngx_http_auth_basic_module模块

- 使用基于用户的访问控制，使用basic机制进行用户认证
- 代码示例

```
location / {
    auth_basic           "closed site";
    auth_basic_user_file conf/htpasswd;
}
htpasswd -bc /etc/nginx/conf/htpasswd admin 123456
将密码文件权限改为600观察会发生什么现象？
```

- htpasswd工具介绍
  - -c：创建一个密码文件
  - -n：不会更新文件，显示文件内容信息
  - -b：免交互式输入用户密码信息
  - -i：读取密码采用标准输入方式，并不做检查
  - -m：使用md5的加密算法
  - -B：使用bcrypt对密码进行加密
  - -C：使用bcrypt algorithm对密码进行加密
  - -d：密码加密方式
  - -s：加密方式
  - -p：不进行加密
  - -D：删除指定用户

ngx_http_stub_status_module模块

- 用于输出nginx的基本访问状态信息
- 代码示例

```
location = /basic_status {
    stub_status;
}
Active connections
The current number of active client connections including Waiting
connections.
accepts
The total number of accepted client connections.
handled
The total number of handled connections. Generally, the parameter value is
the same as accepts unless some resource limits have been reached (for
example, the worker_connections limit).
requests
The total number of client requests.
Reading
The current number of connections where nginx is reading the request header.
Writing
The current number of connections where nginx is writing the response back
to the client.
Waiting
The current number of idle client connections waiting for a request.
```

ngx_http_log_module模块

- 相关日志的设置
- 代码示例

```
log_format compression '$remote_addr - $remote_user [$time_local] '
                       '"$request" $status $bytes_sent '
```

```
                      '"$http_referer" "$http_user_agent" "$gzip_ratio"';

access_log /spool/logs/nginx-access.log compression buffer=32k;

----
$remote_addr与$http_x_forwarded_for: 用以记录客户端的ip地址;
$remote_user: 用来记录客户端用户名称;
$time_local: 用来记录访问时间与时区;
$request: 用来记录请求的url与http协议;
$status: 用来记录请求状态; 成功是200
$body_bytes_s ent: 记录发送给客户端文件主体内容大小;
$http_referer: 用来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
```

ngx_http_gzip_module模块

- 使用gzip方式压缩响应的过滤器
- 代码示例

```
gzip            on;
gzip_min_length 1000;
gzip_proxied    expired no-cache no-store private auth;
gzip_types      text/plain application/xml;
```

ngx_http_ssl_module模块

- 定义https的相关配置
- 代码示例

```
http {

    ...

    server {
        listen              443 ssl;
        keepalive_timeout   70;

        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers         AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-
MD5;
        ssl_certificate     /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 10m;

        ...
    }
```

自签名证书

- 生成ca证书

```
cd /apps/nginx
mkdir certs && cd certs
openssl req -newkey rsa:4096 -nodes -sha256 -keyout ca.key -x509 -days 3650 -
out ca.crt
```

- 生成证书请求文件

```
openssl req -newkey rsa:4096 -nodes -sha256 -keyout iproute.cn.key -out
iproute.cn.csr
```

- 签发证书

```
openssl x509 -req -days 36500 -in iproute.cn.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out iproute.cn.crt
cat iproute.cn.crt ca.crt > iproute.crt
```

- 验证证书内容

```
openssl x509 -in iproute.cn.crt -noout -text
```

Nginx证书配置

```
server {
  listen 80;
  listen 443 ssl;
  ssl_certificate /apps/nginx/certs/iproute.crt;
  ssl_certificate_key /apps/nginx/certs/iproute.cn.key;
  ssl_session_cache shared:sslcache:20m;
  ssl_session_timeout 10m;
  root /data/nginx/html;
}
```

ngx_http_rewrite_module模块

- 将用户请求的URI基于regex所描述的模式进行检查，而后完成替换

ngx_http_referer_module模块

- 定义referer首部的合法可用值

Nginx代理和缓存

介绍

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，通常使用到的http/https协议和fastgci（将动态内容和http服务器分离）

代理相关模块及配置

- nginx代理基于ngx_http_proxy_module模块的功能，该模块有很多配置指令：

正向代理

1. resolver：指定dns服务器地址
2. proxy_pass：代理到的地址
3. resolver_timeout：dns解析超时时长

案例

- 添加配置文件

```
[root@nginx1 ~]# cat /etc/nginx/conf.d/www.conf
server{
    listen *:8090;
    resolver 114.114.114.114;
    location / {
        proxy_pass http://$http_host$request_uri;
    }

}
```

- 重启nginx
- 测试

方式一：

```
[root@client ~]# curl -x 192.168.80.10:8090 "http://www.baidu.com" -I
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Wed, 21 Jul 2021 06:16:29 GMT
Content-Type: text/html
Content-Length: 277
Connection: keep-alive
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Etag: "575e1f60-115"
Last-Modified: Mon, 13 Jun 2016 02:50:08 GMT
Pragma: no-cache
```

方式二：

```
[root@client ~]# export http_proxy=http://192.168.80.10:8090
```

反向代理相关指令

1. proxy_pass指令

```
Syntax:    proxy_pass URL;
Default:    —
Context:    location, if in location, limit_except

http://localhost:8000/uri/
http://192.168.56.11:8000/uri/
http://unix:/tmp/backend.socket:/uri/

```

其中，URL为要设置的被代理服务器的地址，包含传输协议、主机名称或ip地址+端口号等元素。当URL中不包含URI时，nginx服务器将不改变源地址中的URI，当URL中包含URI时，服务器将会改变源地址中的URI；

2. proxy_set_header指令

```
Syntax:    proxy_set_header field value;
Default:    proxy_set_header Host $proxy_host;
            proxy_set_header Connection close;
Context:    http, server, location

# 用户请求的时候HOST的值是www.eagleslab.com，那么代理服务会像后端传递请求的还是eagleslab
proxy_set_header Host $http_host;
# 将$remote_addr的值放进变量X-Real-IP中，$remote_addr的值为客户端的ip
proxy_set_header X-Real-IP $remote_addr;
# 客户端通过代理服务访问后端服务，后端服务通过该变量会记录真实客户端地址
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

该指令可以更改nginx服务器接收到的客户端请求的请求头信息，然后将新的请求头发送给被代理的服务器

3. proxy_connect_timeout

```
Syntax: proxy_connect_timeout time;
Default: proxy_connect_timeout 60s;
Context: http, server, location
```

该指令配置nginx服务器与后端被代理服务器尝试建立连接的超时时间

4. proxy_read_timeout

```
Syntax:    proxy_read_timeout time;
Default:    proxy_read_timeout 60s;
Context:    http, server, location
```

该指令配置nginx服务器向后端被代理服务器发出read请求后，等待响应的超时时间

5. proxy_send_timeout

```
Syntax: proxy_send_timeout time;
Default: proxy_send_timeout 60s;
Context: http, server, location
```

该指令配置nginx服务器向后端被代理服务器发出write请求后，等待响应的超时时间

6. proxy_buffering

```
Syntax: proxy_buffering on | off;
Default: proxy_buffering on;
Context: http, server, location
```

该指令nginx会把后端返回的内容先放到缓冲区当中，然后再返回给客户端，边收边传，不是全部接收完再传给客户端

7. proxy_buffer_size

```
Syntax: proxy_buffer_size size;
Default: proxy_buffer_size 4k|8k;
Context: http, server, location
```

该指令设置nginx代理保存用户头信息的缓冲区大小

8. proxy_buffers 缓冲区

```
Syntax: proxy_buffers number size;
Default: proxy_buffers 8 4k|8k;
Context: http, server, location
```

以上是常见的代理配置指令，除此之外还有很多，诸位可以自行探究

缓存相关模块及配置

proxy 中的buffer和cache都是用于提高IO吞吐效率的，但是他们是一对不同的概念，翻译成中文分别是缓冲和缓存。buffer主要用于传输效率不同步或者优先级别不相同的设备之间传递数据，一般通过对一方数据进行**临时**存放，再统一发送的办法传递给另一方，以降低进程之间的等待时间，保证速度较快的进程不发生间断，临时存放的数据一旦发送给另一方，这些数据本身也就没有用处了；cache主要用于将硬盘上已有的数据在内存中建立缓存数据，提高数据的访问效率，提高数据的访问效率，对于过期不用的缓存可以随时销毁，但不会销毁硬盘上的数据。

需要注意的是proxy cache机制依赖于proxy buffer机制，只有在proxy buffer机制开启的情况下proxy cache的配置才会发挥作用

- proxy_cache_path：该指令用于设置nginx服务器存储缓存数据的路径以及缓存索引相关内容

```
Syntax: proxy_cache_path path [levels=levels] [use_temp_path=on|off]
keys_zone=name:size [inactive=time] [max_size=size] [min_free=size]
[manager_files=number] [manager_sleep=time] [manager_threshold=time]
[loader_files=number] [loader_sleep=time] [loader_threshold=time]
[purger=on|off] [purger_files=number] [purger_sleep=time]
[purger_threshold=time];
Default:    —
Context:    http
---
path：设置缓存数据存放的根路径，该路径应该是预先存在于磁盘上的
levels：设置在相对于path指定目录的第几级hash目录中缓存数据
keys_zone：nginx服务器的缓存索引重建进程在内存中为缓存数据建立索引，这一对变量用来设置存放缓存索引的内存区域的名称和大小
```

- proxy_cache_valid：针对不同的HTTP响应状态设置不同的缓存时间

```
Syntax: proxy_cache_valid [code ...] time;
Default:    —
Context:    http, server, location
```

- proxy_hide_header ： 默认情况下，nginx不会将代理服务器的响应中的头字段“Date”、“Server”、“X-Pad”和“X-Accel-...”传递给客户机。proxy_hide_header指令设置将不传递的其他字段。相反，如果需要允许传递字段，则可以使用proxy_pass_ header指令。

```
Syntax: proxy_hide_header field;
Default:    —
Context:    http, server, location
```

- proxy_connect_timeout
- proxy_read_timeout
- proxy_send_timeout

基于http协议代理实验

实验一、简单实现反向代理功能

```
[root@server conf.d]# cat proxy.conf
server {
    listen 8080;
    server_name www.server1.ccom;

    location / {
        proxy_pass http://192.168.80.128:8090;
}
}
```

实验二、proxy代理网站常用优化配置写入到新文件，调用时使用include即可

```
[root@server conf.d]# cat /etc/nginx/proxy_params
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

proxy_connect_timeout 30;
proxy_send_timeout 60;
proxy_read_timeout 60;

proxy_buffering on;
proxy_buffer_size 32k;
proxy_buffers 4 128k;

[root@server conf.d]# cat proxy.conf
server {
    listen 8080;
    server_name www.server1.com;

    location / {
        proxy_pass http://192.168.80.128:8090;
        include proxy_params;
}
}
```

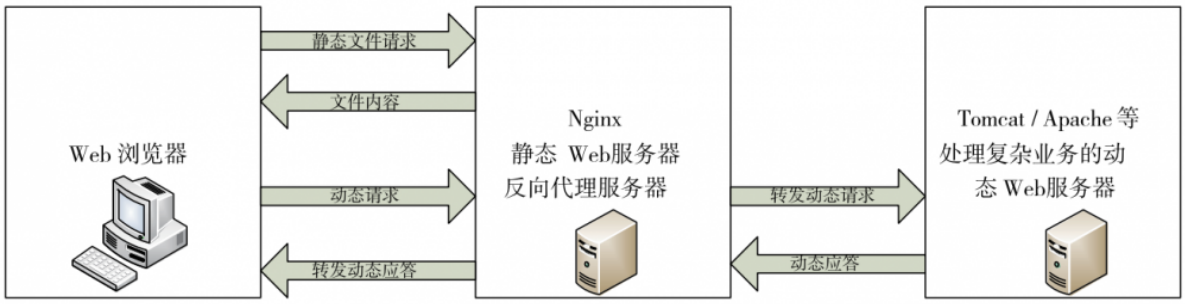
proxy_cache缓存实验

- 准备配置文件

```
http {
```

```
# 其他配置--------------
# 指定了数据存放路径在/myweb/server/proxycache目录下，它包含两级hash目录，缓存数据的总量不能超过20m，如果缓存在5分钟之内没有被访问则强制刷新，定义缓存空间mycache
proxy_cache_path /myweb/server/proxycache levels=1:2 max_size=20m inactive=5m loader_sleep=1m keys_zone=mycache:10m;
# 配置响应数据的临时存放目录
proxy_temp_path /myweb/server/tmp;
# 其他配置--------------
server {
    proxy_pass http://192.168.80.20;
    proxy_cache mycache;
    proxy_cache_valid 200 301 1h;     # 状态码为200 301的响应缓存1h
    proxy_cache_valid any 1m;         # 配置其他状态的响应数据缓存1分钟
    }
}
```

基于fastcgi协议代理和缓存实验：




```
# 官网示例
server {
    location / {
        fastcgi_pass  localhost:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param QUERY_STRING    $query_string;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

```
node1为代理服务器
node2为lnmp服务器
[root@node1 ~]# cat /etc/nginx/nginx.conf
http{
...
fastcgi_cache_path /var/cache/nginx/fastcgi_cache levels=1:2:1 keys_zone=fcgi:20m
inactive=120s;
}
[root@node1 conf.d]# cat proxy.conf
server{
server_name 172.16.0.10;
location / {
proxy_pass http://192.168.10.20;
}
```

```
# 实现动静分离
location ~ .*\.(html|txt)$ {
root /usr/share/nginx/html/;
}
# 缓存相关配置
location ~* \.php$ {
fastcgi_cache fcgi;
fastcgi_cache_key $request_uri;
fastcgi_cache_valid 200 302 10m;
fastcgi_cache_valid 301 1h;
fastcgi_cache_valid any 1m;
}
[root@node2 ~]# yum install php-fpm
[root@node2 ~]# systemctl start php-fpm.service
[root@node2 ~]# cat /etc/nginx/conf.d/php.conf
server{
listen 80;
server_name 192.168.10.20;
root /usr/share/nginx/html;
location ~* \.php$ {
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /usr/share/nginx/html$fastcgi_script_name;
include fastcgi_params;
}
}
测试:
1. 访问http://172.16.0.10/index.html 和 index.php
2. 查看node1上的cache目录下缓存内容
```