http:hypertext transfer protocol:超文本传输协议
超本文:带有超级链接的链接
超级链接:能够实现在不同的文档跳转

http/0.9版本:只支持纯文本的传输(带有超级链接),ASCII码,只支持GET功能
HTML:hypertext mark language:超本文标记语言
<h2>Welcome To Gooann</h2>

Browser:浏览器

1.1.1.1 a.html
2.2.2.2 a.html

URI:uniform resource indentifier:统一资源标识符
URL:uniform resource locator:统一资源定位符
统一:路径格式的统一
protocol://address/to/resource_path
web资源:用URL标识,并且让用户客户端代理(浏览器)能够访问的文件
www.gooann.com/a.logo
www.gooann.com/a.jpeg

html:把多种web资源整合成一个html文档,并能够让浏览器访问显示的一种语言

http/1.0版本:引入MIME机制(为了实现能够传输分文本信息)
post:
缓存机制:加快速度
支持长连接
动态网页:web服务器存储的文档是非html文档,而是动态语言,动态语言生成的脚本能够接受用户的参数后形成html文档,把生成的文档返回给客户端

MIME:multi internet mail extension:多用途互联网邮件交换协议;将非文本数据在传输之前重新编码为文本格式,接受方能够用相反的方式将其重新还原为原来的格式,还能调用相应的应用程序显示此文件
SMTP:simple mail transfer protocol:简单邮件传输协议
image/jpeg image/gif

http协议头部:

www.baidu.com/index.php

HTTP个版本区别:

HTTP 0.9是第一个版本的HTTP协议,已过时.它的组成极其简单,只允许客户端发送GET这一种请求,且不支持请求头.由于没有协议头,造成了HTTP 0.9协议只支持一种内容,即纯文本.不过网页仍然支持用HTML语言格式化,同时无法插入图片.HTTP 0.9具有典型的无状态性,每个事务独立进行处理,事务结束时就释放这个连接.由此可见,HTTP协议的无状态特点在其第一个版本0.9中已经成型.一次HTTP 0.9的传输首先要建立一个由客户端到Web服务器的TCP连接,由客户端发起一个请求,然后由Web服务器返回页面内容,然后连接会关闭.如果请求的页面不存在,也不会返回任何错误码.

HTTP 1.0:HTTP协议的第二个版本,第一个在通讯中指定版本号的HTTP协议版本,至今仍被广泛采用.相对于HTTP 0.9 增加了如下主要特性:

请求与响应支持头域

响应对象以一个响应状态行开始

响应对象不只限于超文本

开始支持客户端通过POST方法向Web服务器提交数据,支持GET、HEAD、POST方法

支持长连接(但默认还是使用短连接),缓存机制,以及身份认证

HTTP 1.1:是目前使用最广泛的协议版本.HTTP1.1引入了许多关键性能优化:keepalive连接,chunked编码传输,字节范围请求,请求流水线等

Persistent Connection(keepalive连接):允许HTTP设备在事务处理结束之后将TCP连接保持在打开的状态,以便未来的HTTP请求重用现在的连接,直到客户端或服务器端决定将其关闭为止.在HTTP1.0中使用长连接需要添加请求头 Connection:Keep-Alive,而在HTTP 1.1 所有的连接默认都是长连接,除非特殊声明不支持(HTTP请求报文首部加上Connection:close)

HTTP 2.0是下一代HTTP协议,目前应用还非常少.主要特点有:

多路复用(二进制分帧):HTTP 2.0最大的特点:不会改动HTTP的语义,HTTP 方法、状态码、URI 及首部字段,等等这些核心概念上一如往常,却能致力于突破上一代标准的性能限制,改进传输性能,实现低延迟和高吞吐量.而之所以叫2.0,是在于新增的二进制分帧层.在二进制分帧层上, HTTP 2.0将所有传输的信息分割为更小的消息和帧,并对它们采用二进制格式的编码 ,其中HTTP1.x的首部信息会被封装到Headers帧,而我们的request body则封装到Data帧里面.HTTP 2.0 通信都在一个连接上完成,这个连接可以承载任意数量的双向数据流.相应地,每个数据流以消息的形式发送,而消息由一或多个帧组成,这些帧可以乱序发送,然后再根据每个帧首部的流标识符重新组装.

头部压缩:当一个客户端向相同服务器请求许多资源时,像来自同一个网页的图像,将会有大量的请求看上去几乎同样的,这就需要压缩技术对付这种几乎相同的信息.

随时复位:HTTP1.1一个缺点是当HTTP信息有一定长度大小数据传输时,你不能方便地随时停止它,中断TCP连接的代价是昂贵的.使用HTTP2的RST_STREAM将能方便停止一个信息传输,启动新的信息,在不中断连接的情况下提高带宽利用效率.
服务器端推流:Server Push:客户端请求一个资源X,服务器端判断也许客户端还需要资源Z,在无需事先询问客户端情况下将资源Z推送到客户端,客户端接受到后,可以缓存起来以备后用.
优先权和依赖:每个流都有自己的优先级别,会表明哪个流是最重要的,客户端会指定哪个流是最重要的,有一些依赖参数,这样一个流可以依赖另外一个流.优先级别可以在运行时动态改变,当用户滚动页面时,可以告诉浏览器哪个图像是最重要的,你也可以在一组流中进行优先筛选,能够突然抓住重点流.

HTTP/1.1协议中共定义了八种方法(动作)来表明Request-URI指定的资源的不同操作方式:
1.OPTIONS:返回服务器针对特定资源所支持的HTTP请求方法.也可以利用向Web服务器发送'*'的请求来测试服务器的功能性
2.HEAD:向服务器索要与GET请求相一致的响应,只不过响应体将不会被返回.这一方法可以在不必传输整个响应内容的情况下,就可以获取包含在响应消息头中的元信息
3.GET:向特定的资源发出请求.注意:GET方法不应当被用于产生“副作用”的操作中,例如在web app.中.其中一个原因是GET可能会被网络蜘蛛等随意访问
4.POST:向指定资源提交数据进行处理请求(例如提交表单或者上传文件).数据被包含在请求体中.POST请求可能会导致新的资源的建立和/或已有资源的修改
5.PUT:向指定资源位置上传其最新内容
6.DELETE:请求服务器删除Request-URI所标识的资源
7.TRACE:回显服务器收到的请求,主要用于测试或诊断
8.CONNECT:HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器

HTTP报文:请求报文(http request)和响应报文(http response)

请求报文语法:
<method><request-URL><version>
<headers>

<entity-body>
响应报文语法:
<version><status><reason-phrase>
<headers>

<entity-body>

响应报文的状态代码:
1xx:纯信息,已经弃用
2xx:"成功"类的信息
3xx:重定向类的信息
4xx:客户端错误类信息
5xx:服务器端错误类的信息

Web服务器的主要操作:
1.建立连接:接受和拒绝客户端连接请求
2.接收请求:通过网络读取HTTP请求报文
3.处理请求:解析请求报文并作出相应的动作
4.访问资源:访问请求报文中相关的资源
5.构建响应:使用正确的首部生成HTTP响应报文
6.发送相应:向客户端发送生成的响应报文
7.记录日志:把已经完成的HTTP事务记录进日志文件

Web服务器处理并发连接请求的工作模型:
1.单线程web服务器(Single-threaded web servers):此种架构方式中,web服务器一次处理一个请求,结束后读取并处理下一个请求.在某请求处理过程中,其它所有的请求将被阻塞,因此,在并发请求较多的场景中将会出现严重的性能问题.(即一次只能处理一个请求)
2.多进程/多线程web服务器:此种架构方式中,web服务器生成多个进程或线程并行处理多个用户请求,进程或线程可以按需或事先生成.有的web服务器应用程序为每个用户请求生成一个单独的进程或线程来进行响应,不过,一旦并发请求数量达到成千上万时,多个同时运行的进程或线程将会消耗大量的系统资源.(即每个进程只能响应一个请求,并且一个进程对应一个线程)
3.I/O多路复用web服务器: 为了能够支持更多的并发用户请求,越来越多的web服务器正在采用多路复用的架构--即同步监控所有的连接请求的活动状态,当一个连接的状态发生改变时(如数据准备完毕或发生某错误),将为其执行一系列特定操作；在操作完成后,此连接将重新变回暂时的稳定态并返回至打开的连接列表中,直到下一次的状态改变.由于其多路复用的特性,进程或线程不会被空闲的连接所占用,因而可以提供高效的工作模式.(这种架构可以理解为一个进程可以生成多个线程,每个请求交给一个线程进行处理).

NGINX
nginx的主要着眼点就是其高性能以及对物理计算资源的高密度利用,因此其采用了不同的架构模型.受启发于多种操作系统设计中基于“事件”的高级处理机制,nginx采用了模块化、事件驱动、异步、单线程及非阻塞的架构,并大量采用了多路复用及事件通知机制.在nginx中,连接请求由为

数不多的几个仅包含一个线程的进程worker以高效的回环(run-loop)机制进行处理,而每个worker可以并行处理数千个的并发连接及请求.

Nginx会按需同时运行多个进程:一个主进程(master)和几个工作进程(worker),配置了缓存时还会有缓存加载器进程(cache loader)和缓存管理器进程(cache manager)等.所有进程均是仅含有一个线程,并主要通过“共享内存”的机制实现进程间通信.主进程以root用户身份运行,而worker、cache loader和cache manager均应以非特权用户身份运行.

主进程主要完成如下工作:
1. 读取并验正配置信息;
2. 创建、绑定及关闭套接字;
3. 启动、终止及维护worker进程的个数;
4. 无须中止服务而重新配置工作特性;

5. 控制非中断式程序升级,启用新的二进制程序并在需要时回滚至老版本;
6. 重新打开日志文件;
7. 编译嵌入式perl脚本;

worker进程主要完成的任务包括:
1. 接收、传入并处理来自客户端的连接;
2. 提供反向代理及过滤功能;
3. nginx任何能完成的其它任务;
如果负载以CPU密集型应用为主,如SSL或压缩应用,则worker数应与CPU数相同;如果负载以IO密集型为主,如响应大量内容给客户端,则worker数应该为CPU个数的1.5或2倍

nginx:www.nginx.org
FastCGI:快速通用网关接口 连接php

nginx cache(disk)

/var/log/:日志

HTTP referer:在请求首部中过滤非法字符(防盗链)
平滑升级:不中断服务的情况下升级nginx版本

Nginx:

nginx解决C10K

解压: ssl/tsl

```
  # yum -y install pcre-devel openssl-devel zlib-devel gd gd-devel
  # tar xvzf nginx-1.9.5.tar.gz -C /usr/src
  # useradd -r nginx
  # cd /usr/src/nginx-1.9.5
  # ./configure \
 --prefix=/usr/local/nginx \
 --sbin-path=/usr/sbin/nginx \
 --conf-path=/etc/nginx/nginx.conf \
 --error-log-path=/var/log/nginx/error.log \
 --http-log-path=/var/log/nginx/access.log \
 --pid-path=/var/run/nginx.pid  \
 --lock-path=/var/lock/nginx.lock \
 --user=nginx \
 --group=nginx \
 --with-http_ssl_module \
 --with-http_flv_module \
 --with-http_stub_status_module \
 --with-http_gzip_static_module \
 --http-client-body-temp-path=/var/tmp/nginx/client/ \
 --http-proxy-temp-path=/var/tmp/nginx/proxy/ \
 --http-fastcgi-temp-path=/var/tmp/nginx/fcgi/ \
 --http-uwsgi-temp-path=/var/tmp/nginx/uwsgi \
 --http-scgi-temp-path=/var/tmp/nginx/scgi \
 --with-pcre \
 --with-file-aio \
 --with-http_image_filter_module
  make &&   make install
```

nginx命令:

```
  -h  显示帮助信息
  -v  查看版本信息
  -V  显示版本和配置选项
  -t  测试nginx配置是否正确
  -s  向主进程发送信号(stop reopen reload quit)
```

-c 启动时指定主配文件

nginx.conf:主配文件
mime.types:
fastcgi:连接php

nginx主配结构:所有语句都以分号结束
全局部分
events
http
server:定义虚拟主机
location

worker_processes 1;工作进程个数(如果是CPU密集型,如SSL或压缩等,设置为和CPU核心数相等;如果是IO密集型,设置为CPU的1.5到2倍)

worker_connections 1024;每个worker_processes能够并发接受的连接数

ip访问限制:

```
location / {
    allow 192.168.10.0/24;
    allow 192.168.10.10;
    deny 192.168.10.0.24;
    deny 192.168.10.1;
    deny all;
}
```

用户名的访问控制:

```
    location / {
auth_basic          "closed site";
auth_basic_user_file conf/htpasswd;
}
```

nginx:stub_status_module状态模块

```
location /status {
    stub_status;
}
```

状态模块结果解析:

Active connections

当前的活动客户端连接数量,包括Waiting连接。

accepts

接受客户端连接的总数。

handled

处理的连接总数。一般来说,accepts 除非达到某些资源限制(例如, worker_connections限制),参数值是相同的。

requests

客户端请求的总数。

Reading

nginx正在读请求头的当前连接数。

Writing

nginx正在将响应写回客户端的当前连接数。

Waiting

当前空闲客户端连接数等待一个请求。

location:站点路径

Nginx启动控制脚本:

```
#!/bin/sh
#
# nginx - this script starts and stops the nginx daemon
#
# chkconfig:   - 85 15
# description:  Nginx is an HTTP(S) server, HTTP(S) reverse \
#               proxy and IMAP/POP3 proxy server
# processname: nginx
# config:      /etc/nginx/nginx.conf
# config:      /etc/sysconfig/nginx
# pidfile:     /var/run/nginx.pid
```

```
# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ "$NETWORKING" = "no" ] && exit 0

nginx="/usr/sbin/nginx"
prog=$(basename $nginx)

NGINX_CONF_FILE="/etc/nginx/nginx.conf"

[ -f /etc/sysconfig/nginx ] && . /etc/sysconfig/nginx

lockfile=/var/lock/subsys/nginx

make_dirs() {
   # make required directories
   user=`nginx -V 2>&1 | grep "configure arguments:" | sed 's/[^*]*--user=\([^
]*\).*/\1/g' -`
   options=`$nginx -V 2>&1 | grep 'configure arguments:'`
   for opt in $options; do
       if [ `echo $opt | grep '.*-temp-path'` ]; then
           value=`echo $opt | cut -d "=" -f 2`
           if [ ! -d "$value" ]; then
               # echo "creating" $value
               mkdir -p $value && chown -R $user $value
           fi
       fi
   done
}

start() {
    [ -x $nginx ] || exit 5
```

```
    [ -f $NGINX_CONF_FILE ] || exit 6
    make_dirs
    echo -n $"Starting $prog: "
    daemon $nginx -c $NGINX_CONF_FILE
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}

stop() {
    echo -n $"Stopping $prog: "
    killproc $prog -QUIT
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}

restart() {
    configtest || return $?
    stop
    sleep 1
    start
}

reload() {
    configtest || return $?
    echo -n $"Reloading $prog: "
    killproc $nginx -HUP
    RETVAL=$?
    echo
}

force_reload() {
    restart
}
```

```
configtest() {
  $nginx -t -c $NGINX_CONF_FILE
}

rh_status() {
    status $prog
}

rh_status_q() {
    rh_status >/dev/null 2>&1
}

case "$1" in
    start)
        rh_status_q && exit 0
        $1
        ;;
    stop)
        rh_status_q || exit 0
        $1
        ;;
    restart|configtest)
        $1
        ;;
    reload)
        rh_status_q || exit 7
        $1
        ;;
    force-reload)
        force_reload
        ;;
    status)
        rh_status
        ;;
    condrestart|try-restart)
        rh_status_q || exit 0
```

```
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|try-restart|reload|force-reload|configtest}"
        exit 2
esac
```

调整内核参数:

```
fs.file-max = 999999
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_max_tw_buckets = 5000
net.ipv4.ip_local_port_range = 1024 61000
net.ipv4.tcp_rmem = 4096 32768 262142
net.ipv4.tcp_wmem = 4096 32768 262142
net.core.netdev_max_backlog = 8096
net.core.rmem_default = 262144
net.core.wmem_default = 262144
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog=1024
```

上面的参数意义解释如下：

·file-max：这个参数表示进程(比如一个worker进程)可以同时打开的最大句柄数,这个参数直接限制最大并发连接数,需根据实际情况配置。

·tcp_tw_reuse：这个参数设置为1,表示允许将TIME-WAIT状态的socket重新用于新的TCP连接,这对于服务器来说很有意义,因为服务器上总会有大量TIME-WAIT状态的连接。

·tcp_keepalive_time：这个参数表示当keepalive启用时,TCP发送keepalive消息的频度。默认是2小时,若将其设置得小一些,可以更快地清理无效的连接。

·tcp_fin_timeout：这个参数表示当服务器主动关闭连接时,socket保持在FIN-WAIT-2状态的最大时间。

·tcp_max_tw_buckets：这个参数表示操作系统允许TIME_WAIT套接字数量的最大值,如果超过这个数字,TIME_WAIT套接字将立刻被清除并打印警告信息。该参数默认为180000,过多的

TIME_WAIT套接字会使Web服务器变慢。
·tcp_max_syn_backlog：这个参数表示TCP三次握手建立阶段接收SYN请求队列的最大长度,默认为1024,将其设置得大一些可以使出现Nginx繁忙来不及accept新连接的情况时,Linux不至于丢失客户端发起的连接请求。
·ip_local_port_range：这个参数定义了在UDP和TCP连接中本地(不包括连接的远端)端口的取值范围。
·net.ipv4.tcp_rmem：这个参数定义了TCP接收缓存(用于TCP接收滑动窗口)的最小值、默认值、最大值。
·net.ipv4.tcp_wmem：这个参数定义了TCP发送缓存(用于TCP发送滑动窗口)的最小值、默认值、最大值。
·netdev_max_backlog：当网卡接收数据包的速度大于内核处理的速度时,会有一个队列保存这些数据包。这个参数表示该队列的最大值。
·rmem_default：这个参数表示内核套接字接收缓存区默认的大小。
·wmem_default：这个参数表示内核套接字发送缓存区默认的大小。
·rmem_max：这个参数表示内核套接字接收缓存区的最大大小。
·wmem_max：这个参数表示内核套接字发送缓存区的最大大小。
注意　滑动窗口的大小与套接字缓存区会在一定程度上影响并发连接的数目。每个TCP连接都会为维护TCP滑动窗口而消耗内存,这个窗口会根据服务器的处理速度收缩或扩张。
参数wmem_max的设置,需要平衡物理内存的总大小、Nginx并发处理的最大连接数量(由nginx.conf中的worker_processes和worker_connections参数决定)而确定。当然,如果仅仅为了提高并发量使服务器不出现Out Of Memory问题而去降低滑动窗口大小,那么并不合适,因为滑动窗口过小会影响大数据量的传输速度。rmem_default、wmem_default、rmem_max、wmem_max这4个参数的设置需要根据我们的业务特性以及实际的硬件成本来综合考虑。
·tcp_syncookies：该参数与性能无关,用于解决TCP的SYN攻击。

安装文件:/usr/local/nginx
  html目录:nginx存放网页目录
  logs目录:存放nginx日志目录
  sbin目录:存放二进制可执行文件目录(比如启动脚本)
  conf目录:存放配置文件目录
    fastcgi*:支持fastcgi配置文件
      koi*(Kernel object information)
      mime*:(mutil internet mail extend):转换定义非字符串数据
      nginx.conf:nginx的主配文件

nginx主配文件解释:

```
worker_processes  4;默认启动工作进程数量,最好是CPU个数(核心数相同)
events {
 worker_connections  1024;
}                                   //每个worker_processes最大的连接数

 http {                                 //设置站点
include       mime.types;     //文件类型
default_type  application/octet-stream;   //默认应用程序类型
sendfile        on;   //开启高速传输文件
server {                        //网页站点
    listen       80;
    server_name  www.baidu.com

    location / {
        root   /www/baidu;                存放网页目录
        index  index.html index.htm;
    }
          }

 server {                       //网页站点
    listen       80;
    server_name  www.sina.com

    location / {
        root   /www/sina;                存放网页目录
        index  index.html index.htm;
    }
          }
```

长连接:

tcp:传输控制协议,在传输data之前,先建立控制连接

http:每请求一个类型的data,都要进行三次握手

js html css

jd:十种类型/小类型

apache工作进程方式
prefork:一个进处理一个请求
worker:一个进程处理多个请求
event:一个进程启动多个线程,每个线程 处理一个请求

nginx参数:
--prefix= 指向安装目录
--sbin-path 指向(执行)程序文件(nginx)
--conf-path= 指向配置文件(nginx.conf)
--error-log-path= 指向错误日志目录
--pid-path= 指向pid文件(nginx.pid) /var/run
--lock-path= 指向lock文件(nginx.lock)(安装文件锁定,防止安装文件被别人利用,或自己误操作。)
--user= 指定程序运行时的非特权用户
--group= 指定程序运行时的非特权用户组
--builddir= 指向编译目录
--with-rtsig_module 启用rtsig模块支持(实时信号)
--with-select_module 启用select模块支持(一种轮询模式,不推荐在高载环境下使用)禁用：--without-select_module
--with-poll_module 启用poll模块支持(功能与select相同,与select特性相同,为一种轮询模式,不推荐在高载环境下使用)
--with-file-aio 启用file aio支持(一种APL文件传输格式)
--with-ipv6 启用ipv6支持
--with-http_ssl_module 启用ngx_http_ssl_module支持(使支持https请求,需已安装openssl)
--with-http_realip_module 启用ngx_http_realip_module支持(这个模块允许从请求标头更改客户端的IP地址值,默认为关)
--with-http_addition_module 启用ngx_http_addition_module支持(作为一个输出过滤器,支持不完全缓冲,分部分响应请求)
--with-http_xslt_module 启用ngx_http_xslt_module支持(过滤转换XML请求)
--with-http_image_filter_module 启用ngx_http_image_filter_module支持(传输JPEG/GIF/PNG 图片的一个过滤器)(默认为不启用。gd库要用到)
--with-http_geoip_module 启用ngx_http_geoip_module支持(该模块创建基于与MaxMind GeoIP二进制文件相配的客户端IP地址的ngx_http_geoip_module变量)
--with-http_sub_module 启用ngx_http_sub_module支持(允许用一些其他文本替换nginx响应中的一些文本)

--with-http_dav_module 启用ngx_http_dav_module支持(增加PUT,DELETE,MKCOL：创建集合,COPY和MOVE方法)默认情况下为关闭,需编译开启
--with-http_flv_module 启用ngx_http_flv_module支持(提供寻求内存使用基于时间的偏移量文件)
--with-http_gzip_static_module 启用ngx_http_gzip_static_module支持(在线实时压缩输出数据流)
--with-http_random_index_module 启用ngx_http_random_index_module支持(从目录中随机挑选一个目录索引)
--with-http_secure_link_module 启用ngx_http_secure_link_module支持(计算和检查要求所需的安全链接网址)
--with-http_degradation_module 启用ngx_http_degradation_module支持(允许在内存不足的情况下返回204或444码)
--with-http_stub_status_module 启用ngx_http_stub_status_module支持(获取nginx自上次启动以来的工作状态)
--without-http_charset_module 禁用ngx_http_charset_module支持(重新编码web页面,但只能是一个方向--服务器端到客户端,并且只有一个字节的编码可以被重新编码)
--without-http_gzip_module 禁用ngx_http_gzip_module支持(该模块同-with-http_gzip_static_module功能一样)
--without-http_ssi_module 禁用ngx_http_ssi_module支持(该模块提供了一个在输入端处理处理服务器包含文件(SSI)的过滤器,目前支持SSI命令的列表是不完整的)
--without-http_userid_module 禁用ngx_http_userid_module支持(该模块用来处理用来确定客户端后续请求的cookies)
--without-http_access_module 禁用ngx_http_access_module支持(该模块提供了一个简单的基于主机的访问控制。允许/拒绝基于ip地址)
--without-http_auth_basic_module禁用ngx_http_auth_basic_module(该模块是可以使用用户名和密码基于http基本认证方法来保护你的站点或其部分内容)
--without-http_autoindex_module 禁用disable ngx_http_autoindex_module支持(该模块用于自动生成目录列表,只在ngx_http_index_module模块未找到索引文件时发出请求。)
--without-http_geo_module 禁用ngx_http_geo_module支持(创建一些变量,其值依赖于客户端的IP地址)
--without-http_map_module 禁用ngx_http_map_module支持(使用任意的键/值对设置配置变量)
--without-http_split_clients_module 禁用ngx_http_split_clients_module支持(该模块用来基于某些条件划分用户。条件如：ip地址、报头、cookies等等)
--without-http_referer_module 禁用disable ngx_http_referer_module支持(该模块用来过滤请求,拒绝报头中Referer值不正确的请求)

--without-http_rewrite_module 禁用ngx_http_rewrite_module支持(该模块允许使用正则表达式改变URI,并且根据变量来转向以及选择配置。如果在server级别设置该选项,那么他们将在location之前生效。如果在location还有更进一步的重写规则,location部分的规则依然会被执行。如果这个URI重写是因为location部分的规则造成的,那么 location部分会再次被执行作为新的URI。 这个循环会执行10次,然后Nginx会返回一个500错误。)
--without-http_proxy_module 禁用ngx_http_proxy_module支持(有关代理服务器)
--without-http_fastcgi_module 禁用ngx_http_fastcgi_module支持(该模块允许Nginx 与FastCGI 进程交互,并通过传递参数来控制FastCGI 进程工作。)FastCGI一个常驻型的公共网关接口。
--without-http_uwsgi_module 禁用ngx_http_uwsgi_module支持(该模块用来医用uwsgi协议,uWSGI服务器相关)
--without-http_scgi_module 禁用ngx_http_scgi_module支持(该模块用来启用SCGI协议支持,SCGI协议是CGI协议的替代。它是一种应用程序与HTTP服务接口标准。它有些像FastCGI但他的设计 更容易实现。)
--without-http_memcached_module 禁用ngx_http_memcached_module支持(该模块用来提供简单的缓存,以提高系统效率)
-without-http_limit_zone_module 禁用ngx_http_limit_zone_module支持(该模块可以针对条件,进行会话的并发连接数控制)
--without-http_limit_req_module 禁用ngx_http_limit_req_module支持(该模块允许你对于一个地址进行请求数量的限制用一个给定的session或一个特定的事件)
--without-http_empty_gif_module 禁用ngx_http_empty_gif_module支持(该模块在内存中常驻了一个1*1的透明GIF图像,可以被非常快速的调用)
--without-http_browser_module 禁用ngx_http_browser_module支持(该模块用来创建依赖于请求报头的值。如果浏览器为modern ,则$modern_browser等于modern_browser_value指令分配的值；如 果浏览器为old,则$ancient_browser等于 ancient_browser_value指令分配的值；如果浏览器为 MSIE中的任意版本,则 $msie等于1)
--without-http_upstream_ip_hash_module 禁用ngx_http_upstream_ip_hash_module支持(该模块用于简单的负载均衡)
--with-http_perl_module 启用ngx_http_perl_module支持(该模块使nginx可以直接使用perl或通过ssi调用perl)
--with-perl_modules_path= 设定perl模块路径
--with-perl= 设定perl库文件路径
--http-log-path= 设定access log路径
--http-client-body-temp-path= 设定http客户端请求临时文件路径
--http-proxy-temp-path= 设定http代理临时文件路径
--http-fastcgi-temp-path= 设定http fastcgi临时文件路径
--http-uwsgi-temp-path= 设定http uwsgi临时文件路径

--http-scgi-temp-path= 设定http scgi临时文件路径
-without-http 禁用http server功能
--without-http-cache 禁用http cache功能
--with-mail 启用POP3/IMAP4/SMTP代理模块支持
--with-mail_ssl_module 启用ngx_mail_ssl_module支持
--without-mail_pop3_module 禁用pop3协议(POP3即邮局协议的第3个版本,它是规定个人计算机如何连接到互联网上的邮件服务器进行收发邮件的协议。是因特网电子邮件的第一个离线协议标 准,POP3协议允许用户从服务器上把邮件存储到本地主机上,同时根据客户端的操作删除或保存在邮件服务器上的邮件。POP3协议是TCP/IP协议族中的一员,主要用于 支持使用客户端远程管理在服务器上的电子邮件)
--without-mail_imap_module 禁用imap协议(一种邮件获取协议。它的主要作用是邮件客户端可以通过这种协议从邮件服务器上获取邮件的信息,下载邮件等。IMAP协议运行在TCP/IP协议之上, 使用的端口是143。它与POP3协议的主要区别是用户可以不用把所有的邮件全部下载,可以通过客户端直接对服务器上的邮件进行操作。)
--without-mail_smtp_module 禁用smtp协议(SMTP即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则,由它来控制信件的中转方式。SMTP协议属于TCP/IP协议族,它帮助每台计算机在发送或中转信件时找到下一个目的地。)
--with-google_perftools_module 启用ngx_google_perftools_module支持(调试用,剖析程序性能瓶颈)
--with-cpp_test_module 启用ngx_cpp_test_module支持
--add-module= 启用外部模块支持
--with-cc= 指向C编译器路径
--with-cpp= 指向C预处理路径
--with-cc-opt= 设置C编译器参数(PCRE库,需要指定–with-cc-opt=”-I /usr/local/include”,如果使用select()函数则需要同时增加文件描述符数量,可以通过–with-cc- opt=”-D FD_SETSIZE=2048”指定。)
--with-ld-opt= 设置连接文件参数。(PCRE库,需要指定–with-ld-opt=”-L /usr/local/lib”。)
--with-cpu-opt= 指定编译的CPU,可用的值为: pentium, pentiumpro, pentium3, pentium4, athlon, opteron, amd64, sparc32, sparc64, ppc64
--without-pcre 禁用pcre库
--with-pcre 启用pcre库
--with-pcre= 指向pcre库文件目录
--with-pcre-opt= 在编译时为pcre库设置附加参数
--with-md5= 指向md5库文件目录(消息摘要算法第五版,用以提供消息的完整性保护)
--with-md5-opt= 在编译时为md5库设置附加参数
--with-md5-asm 使用md5汇编源
--with-sha1= 指向sha1库目录(数字签名算法,主要用于数字签名)

--with-sha1-opt= 在编译时为sha1库设置附加参数
--with-sha1-asm 使用sha1汇编源
--with-zlib= 指向zlib库目录
--with-zlib-opt= 在编译时为zlib设置附加参数
--with-zlib-asm= 为指定的CPU使用zlib汇编源进行优化,CPU类型为pentium, pentiumpro
--with-libatomic 为原子内存的更新操作的实现提供一个架构
--with-libatomic= 指向libatomic_ops安装目录
--with-openssl= 指向openssl安装目录
--with-openssl-opt 在编译时为openssl设置附加参数
--with-debug 启用debug日志

Location语法规则： location [=|~|~*|^~] /uri/ { … }
= 开头表示精确匹配,只匹配当前目录
^~ 开头表示uri以某个常规字符串开头,理解为匹配 url路径即可。nginx不对url做编码,因此请求为/static/20%/aa,可以被规则^~ /static/ /aa匹配到。
~ 开头表示区分大小写的正则匹配
~* 开头表示不区分大小写的正则匹配
/ 通用匹配,任何请求都会匹配到。
多个location配置的情况下匹配顺序为
首先匹配 =,其次匹配^~, 其次是按文件中顺序的正则匹配,最后是交给 / 通用匹配。当有匹配成功时候,停止匹配,按当前匹配规则处理请求。
例子,有如下匹配规则：

```
location = / {
  #规则A
}
location = /login {
  #规则B
}
location ^~ /static/ {
  #规则C
}
location ~ \.(gif|jpg|png|js|css|)$ {
  #规则D
}
location ~* \.png$ {
```

```
  #规则E
}
location !~ \.xhtml$ {
  #规则F
}
location !~* \.xhtml$ {
  #规则G
}
location / {
  #规则H
}
```

那么产生的效果如下：
访问根目录/, 比如http://localhost/ 将匹配规则A
访问 http://localhost/login 将匹配规则B,http://localhost/register 则匹配规则H
访问 http://localhost/static/a.html 将匹配规则C
访问 http://localhost/a.gif, http://localhost/b.jpg 将匹配规则D和规则E,但是规则D顺序优先,规则E不起作用,而 http://localhost/static/c.png 则优先匹配到 规则C
访问 http://localhost/a.PNG 则匹配规则E,而不会匹配规则D,因为规则E不区分大小写。
访问 http://localhost/a.xhtml 不会匹配规则F和规则G,http://localhost/a.XHTML不会匹配规则G,因为不区分大小写。规则F,规则G属于排除法,符合匹配规则但是不会匹配到

访问 http://localhost/category/id/1111 则最终匹配到规则H,因为以上规则都不匹配,这个时候应该是nginx转发请求给后端应用服务器,比如FastCGI(php),tomcat(jsp),nginx作为方向代理服务器存在。
所以实际使用中,通常至少有三个匹配规则定义,如下：

```
#直接匹配网站根,通过域名访问网站首页比较频繁,使用这个会加速处理
#这里是直接转发给后端应用服务器了,也可以是一个静态首页
# 第一个必选规则
location = / {
    proxy_pass http://tomcat:8080/index
}

# 第二个必选规则是处理静态文件请求,这是nginx作为http服务器的强项
# 有两种配置模式,目录匹配或后缀匹配,任选其一或搭配使用
location ^~ /static/ {
```

```
    root /webroot/static/;
}
location ~* \.(gif|jpg|jpeg|png|css|js|ico)$ {
    root /webroot/res/;
}

#第三个规则就是通用规则,用来转发动态请求到后端应用服务器
#非静态文件请求就默认是动态请求,自己根据实际把握
#毕竟目前的一些框架的流行,带.php,.jsp后缀的情况很少了
location / {
    proxy_pass http://tomcat:8080/
 }
```

Nginx防盗链做法:

一：一般的防盗链如下：

```
location ~* \.(gif|jpg|png|swf|flv)$ {
  root /www/image;
  valid_referers none blocked *.baidu.com;
  if ($invalid_referer) {
    rewrite ^/ http://www.baidu.com/retrun.html;
    #return 403;
  }
}
```

第一行：gif|jpg|png|swf|flv

表示对gif、jpg、png、swf、flv后缀的文件实行防盗链

第二行： 表示对www.baidu.com这2个来路进行判断

if{}里面内容的意思是,如果来路不是指定来路,如果来路不是指定来路就跳转到http://www.baidu.com/retrun.html页面,当然直接返回403也是可以的。

二：针对图片目录防止盗链

```
location /images/ {
  alias /data/images/;
  valid_referers none blocked server_names test.com;
```

```
  if ($invalid_referer) {return 403;}
}
```

三：使用第三方模块ngx_http_accesskey_module实现Nginx防盗链

实现方法如下：

实现方法如下：

1. 下载NginxHttpAccessKeyModule模块文件：http://wiki.nginx.org/File:Nginx-accesskey-2.0.3.tar.gz；

2. 解压此文件后,找到nginx-accesskey-2.0.3下的config文件。编辑此文件：替换其中的”$HTTP_ACCESSKEY_MODULE”为”ngx_http_accesskey_module”；

3. 用一下参数重新编译nginx：

```
./configure --add-module=path/to/nginx-accesskey
<<pestd add
```

上面需要加上原有到编译参数,然后执行: make && make install

4. 修改nginx的conf文件,添加以下几行：

```
location /download {
  accesskey on;
  accesskey_hashmethod md5;
  accesskey_arg "key";
  accesskey_signature "mypass$remote_addr";
}
```

其中：

accesskey为模块开关；

accesskey_hashmethod为加密方式MD5或者SHA-1；

accesskey_arg为url中的关键字参数；

accesskey_signature为加密值,此处为mypass和访问IP构成的字符串。

访问测试脚本download.php：

```
<?
$ipkey= md5("mypass".$_SERVER['REMOTE_ADDR']);
$output_add_key="<a href=http://www.jzxue.com/download/G3200507120520LM.rar?key=".$ipkey.">download_add_key</a><br />";
$output_org_url="<a href=http://www.jzxue.com/download/G3200507120520LM.rar>download_org_path</a><br />";
echo $output_add_key;
echo $output_org_url;
```

?>
访问第一个download_add_key链接可以正常下载,第二个链接download_org_path会返回403 Forbidden错误。

LNMP架构:
nginx和php

nginx主配文件:
全局段:
worker_process
error_logs
user
group

events {
worker_connents
事件驱动设置
}

http {
web服务器设置
}

server {
虚拟主机:
linsten 80;
server_name
}

location [= | ~ | ~* | ^~] URI { URI的访问属性
root
index
allow
deny
auth_basic
}

=:精确匹配
~ ~*:匹配正则表达式
^~:不匹配正则表达式

```
location ~* \.php$ {
  fastcgi_pass http://192.168.10.20:9000;
  fastcgi_index;
}
```

web和mail服务器
反向代理:工作在服务器的前端(前端服务器)
正向代理:工作在客户端的前端(为客户端做代理)
http://192.16810.20/bbs

```
server {
 listen 80;
 server_name www.gooann.com;
}
location /bbs {
proxy_pass http://192.168.10.20/bbs;
}
```

pc:www.gooann.com/bbs
--->http://192.168.10.20/bbs--->192.168.10.20:/var/www/html/bbs/index.html

apache:/var/www/html/

```
location ~* ^/bbs {
proxy_pass http://192.168.10.20/bbs;
proxy_set_header X-Real-IP $remote_addr;
}
```

pc:www.gooann.com/bbs
--->http://192.168.10.20/bbs/bbs

$remote_addr:远程主机
$remote_user

nginx:负载均衡 后端健康状况检查
轮询算法:
round_robin:加权负载平衡
ip_hash:粘连性(pc1--192.168.10.20)
least_conn:最少连接
192.168.10.20
192.168.10.30

负载平衡:
```
upstream bbs {
  server 192.168.10.20 weight=1;
  server 192.168.10.30 weight=1;
}

location ~* /bbs {
  proxy_pass http://bbs;
}
```

server:
weight:权重,默认为0,0代表不参加轮询
max_fails:最多失败次数
fail_timeout:失败超时时间
backup:备份服务器

proxy_pass:反向代理
proxy_set_header X-Real-IP $remote_addr;
upsteam:负载平衡

缓存/压缩/动静分离

缓存:nginx
cache:内存(元数据和磁盘数据对应关系)

磁盘

proxy_cache_path语法:(只能用户http段中)
path:磁盘存放缓存目录
levels:设置目录级别,levels=1:2:1,最大三级目录,数字表示每级目录名称字符数
目录:MD5加密的 c 20 a: aaaaaaaabbcd a20c
keys_zone:设置内存段名称和空间 keys_zone=bbs:10m
max_size:设置磁盘缓存目录的空间大小

proxy_cache_valid [code ...] time;
Context: http, server, location
code:http状态码 1xx 2xx 3xx 4xx 5xx

$upstream_cache_status:缓存状态
hit:命中
miss:未命中
updating:已更新
expr:过期
无效

动静分离:
浏览器:只能够解释html语言

nginx主配文件:

```
worker_process 8;

 events {
   worker_connections 1024;
   use epoll;
 }

 http{
       upstream phpserver {  //动态网站主机组
```

```
            ip_hash;
            server 192.168.10.20 weight=1 max_fails=2 fail_timeout=2;
            server 192.168.30.20 weight=1 max_fails=2 fail_timeout=2;
        }
      upstream imagesserver {
            server 192.168.10.40 weight=1 max_fails=2 fail_timeout=2;
   server 192.168.10.50 weight=1 max_fails=2 fail_timeout=2;
        }
   upstream static {
      server 192.168.10.60 weight=1 max_fails=2 fail_timeout=2;
      server 192.168.10.70 weight=1 max_fails=2 fail_timeout=2;
    }
   server {
      listen 80;
      server_name www.gooann.com
     location / {
      proxy_pass http://static;
      index index.html;
     }
     location ~* \.(jif|png|jpeg|jpg)$ {
      proxy_pass http://imagesserver;
     }
     location ~* \.(php|js|asp)$ {
      proxy_pass http://phpserver;
      index index.php index.
     }

   }

}
```

nginx压缩:

gzip on|off 是否开启gzip压缩功能;上下文:http, server, location, if in location

gzip_buffers number size:

number:nginx服务器向系统申请缓存空间的个数

size:每个缓存空间的大小

gzip_comp_level level:设置压缩比,1-9个级别

gzip_disable regex ...:
这对不同客户端浏览器类型选择关闭压缩功能
gzip_disable MSIE [4-6]\.
gzip_http_version 1.0 | 1.1;
gzip_types mime-type ...;
指定压缩文件的类型(一般压缩文本)

memcached模块
buffer:模块

读写分离/URL地址重写/防盗链

读写分离:

web:bbs服务器
192.168.10.10:nginx前端

192.168.10.20:httpd 上传(写)
192.168.10.30:httpd 读
192.168.10.40:httpd

读写分离:
httpd:启用dav模块(默认启用)
在httpd主配置文件修改(定义上传到服务器的目录/var/www/html)

<Directory "/var/www/html">
Dav on
</Directory>

nginx主配文件:
location / {
proxy_pass http://192.168.10.30;
if ($request_method = "PUT") {

```
    proxy_pass http://192.168.10.20;

    }

  }
```

if (condition) {操作}

```
  ~
  ~*
  =
```

```
rewrite ^/bbs/(.*)$ http://192.168.10.20/forum/$1;
```

www.gooann.com

www.gooann.com/bbs/index.html-->http://192.168.10.20/froum/index.html

last:本次重写完成,重新启动下一次检查(默认检查10次)

```
  rewrite ^/bbs/(.*)/images/(.*)\.jpg$ http://192.168.10.20/bbs/$2/images/$1.jpg;
```

break:本次重写完成,直接读取资源

redirect:临时重定向

permanent:永久重定向

Nginx防盗链做法:

一：一般的防盗链如下：

```
location ~* \.(gif|jpg|png|swf|flv)$ {
  root /www/image;
  valid_referers none blocked *.baidu.com;
  if ($invalid_referer) {
    rewrite ^/ http://www.baidu.com/retrun.html;
    #return 403;
```

```
  }
}
```

第一行：gif|jpg|png|swf|flv
表示对gif、jpg、png、swf、flv后缀的文件实行防盗链
第二行： 表示对www.baidu.com这2个来路进行判断
if{}里面内容的意思是,如果来路不是指定来路,如果来路不是指定来路就跳转到
http://www.baidu.com/retrun.html页面,当然直接返回403也是可以的。