目录

Nginx 配置讲解 2

1.nginx 简介 2

2.nginx 和 apache 对比 2

3.nginx 的主要特征 5

4.nginx 的编译安装 7

1.下载地址 7

2.解压 7

3.编译参数详解 7

4.重点参数归类详解： 15

5.编译安装 19

5.nginx 配置 20

1.配置文件详解 20

2.nginx 日志格式 25

3.location 语法 31

4.nginx 负载均衡 36

5.nginx 反向代理 41

6.proxy_cache 43

7.nginx 开启 ssl 48

8.nginx 常用的优化 55

9.nginx 内置的变量 68

10.Rewrite 规则示例参考.....73

Nginx 配置讲解

1.nginx 简介

Nginx 是俄罗斯人编写的十分轻量级的 HTTP 服务器,Nginx ,它的发音为“engine X”，是一个高性能的 HTTP 和反向代理服务器 ,同时也是一个 IMAP/POP3/SMTP 代理服务器。英文主页：http://nginx.org。

Nginx 以事件驱动的方式编写 ,所以有非常好的性能 ,同时也是一个非常高效的反向代理、负载平衡。其拥有匹配 Lighttpd 的性能，同时还没有 Lighttpd 的内存泄漏问题，而且 Lighttpd 的 mod_proxy 也有一些问题并且很久没有更新。

现在 ,Igor 将源代码以类 BSD 许可证的形式发布。Nginx 因为它的稳定性、丰富的模块库、灵活的配置和低系统资源的消耗而闻名．业界一致认为它是 Apache2.2 + mod_proxy_balancer 的轻量级代替者，不仅是因为响应静态页面的速度非常快，而且它的模块数量达到 Apache 的近 2/3。对 proxy 和 rewrite 模块的支持很彻底 ,还支持 mod_fcgi、ssl、vhosts ，适合用来做 mongrel clusters 的前端 HTTP 响应。

2.nginx 和 apache 对比

Apache 与 Nginx 的优缺点比较

1、nginx 相对于 apache 的优点：

轻量级，同样起 web 服务，比 apache 占用更少的内存及资源

抗并发 ,nginx 处理请求是异步非阻塞的 ,而 apache 则是阻塞型的 ,在高并发下 nginx 能

保持低资源低消耗高性能

高度模块化的设计，编写模块相对简单

社区活跃，各种高性能模块出品迅速

apache 相对于 nginx 的优点：

rewrite ，比 nginx 的 rewrite 强大

模块超多，基本想到的都可以找到

少 bug ，nginx 的 bug 相对较多 （现在 bug 方面应该没多大区别）

超稳定

存在就是理由，一般来说，需要性能的 web 服务，用 nginx 。如果不需要性能只求稳定，那就 apache 吧。后者的各种功能模块实现得比前者，例如 ssl 的模块就比前者好，可配置项多。这里要注意一点，epoll(freebsd 上是 kqueue)网络 IO 模型是 nginx 处理性能高的根本理由，但并不是所有的情况下都是 epoll 大获全胜的，如果本身提供静态服务的就只有寥寥几个文件，apache 的 select 模型或许比 epoll 更高性能。当然，这只是根据网络 IO 模型的原理作的一个假设，真正的应用还是需要实测了再说的。

2、作为 Web 服务器：相比 Apache，Nginx 使用更少的资源，支持更多的并发连接，体现更高的效率 ,这点使 Nginx 尤其受到虚拟主机提供商的欢迎。在高连接并发的情况下，Nginx 是 Apache 服务器不错的替代品: Nginx 在美国是做虚拟主机生意的老板们经常选择的软件平台之一. 能够支持高达 50,000 个并发连接数的响应, 感谢 Nginx 为我们选择了 epoll and kqueue 作为开发模型.

Nginx 作为负载均衡服务器: Nginx 既可以在内部直接支持 Rails 和 PHP 程序对外进行服务, 也可以支持作为 HTTP 代理 服务器对外进行服务. Nginx 采用 C 进行编写, 不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好很多.

作为邮件代理服务器: Nginx 同时也是一个非常优秀的邮件代理服务器(最早开发这个产品的目的之一也是作为邮件代理服务器), Last.fm 描述了成功并且美妙的使用经验.

Nginx 是一个安装非常的简单 , 配置文件非常简洁 (还能够支持 perl 语法), Bugs 非常少的服务器: Nginx 启动特别容易, 并且几乎可以做到 7*24 不间断运行 , 即使运行数个月也不需要重新启动. 你还能够不间断服务的情况下进行软件版本的升级 .

3、Nginx 配置简洁, Apache 复杂

Nginx 静态处理性能比 Apache 高 3 倍以上

Apache 对 PHP 支持比较简单 , Nginx 需要配合其他后端用

Apache 的组件比 Nginx 多

现在 Nginx 才是 Web 服务器的首选

4、最核心的区别在于 apache 是同步多进程模型 , 一个连接对应一个进程 ; nginx 是异步的 , 多个连接 (万级别) 可以对应一个进程

5、nginx 处理静态文件好,耗费内存少.但无疑 apache 仍然是目前的主流,有很多丰富的特性.所以还需要搭配着来.当然如果能确定 nginx 就适合需求,那么使用 nginx 会是更经济的方式.

6、从个人过往的使用情况来看 , nginx 的负载能力比 apache 高很多。最新的服务器也改用 nginx 了。而且 nginx 改完配置能-t 测试一下配置有没有问题 , apache 重启的时候发现配置出错了 , 会很崩溃 , 改的时候都会非常小心翼翼现在看有好多集群站 , 前端 nginx 抗并发 , 后端 apache 集群 , 配合的也不错。

7、nginx 处理动态请求是比较弱 , 一般动态请求要 apache 去做 , nginx 只适合静态和反向。

8、nginx 是很不错的前端服务器 , 负载性能很好 , apache 对 php 等语言的支持很好 , 此

外 apache 有强大的支持网路，发展时间相对 nginx 更久，bug 少但是 apache 有先天不支持多核心处理负载的缺点，建议使用 nginx 做前端，后端用 apache。大型网站建议用 nginx 自代的集群功能

9、Nginx 优于 apache 的主要两点：

1.Nginx 本身就是一个反向代理服务器

2.Nginx 支持 7 层负载均衡；其他的当然，Nginx 可能会比 apache 支持更高的并发。

10、你对 web server 的需求决定你的选择。大部分情况下 nginx 都优于 APACHE，比如说静态文件处理、PHP-CGI 的支持、反向代理功能、前端 Cache、维持连接等等。在 Apache+PHP（prefork）模式下，如果 PHP 处理慢或者前端压力很大的情况下，很容易出现 Apache 进程数飙升，从而拒绝服务的现象。

11、可以看一下 nginx lua 模块：https://github.com/chaoslaw...apache 比 nginx 多的模块，可直接用 lua 实现 apache 是最流行的，why？大多数人懒得更新到 nginx 或者学新事物

12、对于 nginx，我喜欢它配置文件写的很简洁，正则配置让很多事情变得简单运行效率高，占用资源少，代理功能强大，很适合做前端响应服务器

13、Apache 在处理动态有优势，Nginx 并发性比较好，CPU 内存占用低，如果 rewrite 频繁，那还是 Apache 吧

3.nginx 的主要特征

nginx 做为 HTTP 服务器，有以下几项基本特性：

处理静态文件，索引文件以及自动索引；打开文件描述符缓冲．

无缓存的反向代理加速，简单的负载均衡和容错．

FastCGI，简单的负载均衡和容错．

模块化的结构。包括 gzipping, byte ranges, chunked responses,以及 SSI-filter 等 filter。如果由 FastCGI 或其它代理服务器处理单页中存在的多个 SSI，则这项处理可以并行运行，而不需要相互等待。

支持 SSL 和 TLSSNI．

Nginx 专为性能优化而开发，性能是其最重要的考量,实现上非常注重效率 。它支持内核 Poll 模型，能经受高负载的考验,有报告表明能支持高达 50,000 个并发连接数。

Nginx 具有很高的稳定性。其它 HTTP 服务器，当遇到访问的峰值，或者有人恶意发起慢速连接时 ,也很可能会导致服务器物理内存耗尽频繁交换 ,失去响应 ,只能重启服务器。例如当前 apache 一旦上到 200 个以上进程，web 响应速度就明显非常缓慢了。而 Nginx 采取了分阶段资源分配技术，使得它的 CPU 与内存占用率非常低。nginx 官方表示保持 10,000 个没有活动的连接，它只占 2.5M 内存，所以类似 DOS 这样的攻击对 nginx 来说基本上是毫无用处的。就稳定性而言,nginx 比 lighthttpd 更胜一筹。

Nginx 支持热部署。它的启动特别容易，并且几乎可以做到 7*24 不间断运行，即使运行数个月也不需要重新启动。你还能够在不间断服务的情况下，对软件版本进行进行升级。

Nginx 采用 master-slave 模型,能够充分利用 SMP 的优势，且能够减少工作进程在磁盘 I/O 的阻塞延迟。当采用 select()/poll()调用时，还可以限制每个进程的连接数。

Nginx 代码质量非常高，代码很规范，手法成熟， 模块扩展也很容易。特别值得一提的是强大的 Upstream 与 Filter 链。Upstream 为诸如 reverse proxy,与其他服务器通信模块的编写奠定了很好的基础。而 Filter 链最酷的部分就是各个 filter 不必等待前一个 filter 执行完毕。它可以把前一个 filter 的输出做为当前 filter 的输入，这有点像 Unix 的管线。这

意味着，一个模块可以开始压缩从后端服务器发送过来的请求，且可以在模块接收完后端服务器的整个请求之前把压缩流转向客户端。

Nginx 采用了一些 os 提供的最新特性如对 sendfile (Linux2.2+)，accept-filter (FreeBSD4.1+)，TCP_DEFER_ACCEPT (Linux 2.4+)的支持，从而大大提高了性能。

当然，nginx 还很年轻，多多少少存在一些问题，比如：Nginx 是俄罗斯人创建，虽然前几年文档比较少，但是目前文档方面比较全面，英文资料居多，中文的资料也比较多，而且有专门的书籍和资料可供查找。

nginx 的作者和社区都在不断的努力完善 ,我们有理由相信 nginx 将继续以高速的增长率来分享轻量级 HTTP 服务器市场，会有一个更美好的未来。

4.nginx 的编译安装

1.下载地址

Wget http://nginx.org/download/nginx-1.8.0.tar.gz

2.解压

tar zxvf nginx-1.8.0.tar.gz

cd nginx-1.8.0

3.编译参数详解

--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）

--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）

（示例：

```
location /ngx_status
    {
```

```
        stub_status on;
        access_log off;


    }
```

Active connections: 36

server accepts handled requests

9649029 9649029 32008160

Reading: 0 Writing: 1 Waiting: 35

active connections – 活跃的连接数量

server accepts handled requests — 总共处理了 11989 个连接，成功创建 11989 次握手，总共处理了 11991 个请求

reading — 读取客户端的连接数.

writing — 响应数据到客户端的数量

waiting — 开启 keep-alive 的情况下,这个值等于 active – (reading+writing), 意思就是 Nginx 已经处理完正在等候下一次请求指令的驻留连接.）

--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

4.重点参数归类详解：

1. 通用配置选项：

--prefix=<path> 指定 Nginx 的安装路径 ,所有其他的路径都要依赖于该选项

--sbin-path=<path> 指定 Nginx 二进制文件的路径。如果没有指定，那么这个路径将依赖于--prefix 选项

--conf-path=<path> 指定 Nginx 的配置文件的路径，如果在命令行没有指定配置文件，那么将会通过这里指定路径。

--error-log-path=<path> 指定错误日志文件路径，Nginx 将会往里面写入错误日志文件，除非有其它的配置。

--pid-path=<path> 指定的 Nginx master 进程的 PID 文件位置，通常在/var/run 下

--lock-path=<path> 共享存储器互斥锁文件的路径

--user=<user> worker 进程运行的用户

--group=<group> worker 进程运行的组

--with-file-aio. 为 FreeBSD4.3 +和 linux 2.6.22 +系统启用异步 I/O

--with-debug 这个选项用于启用调试日志，在生产环境的系统中不推荐使用

2. 邮件代理的配置选项：

--with-mail 该选项用于启用 Mail 模块，该模块默认没有被激活

--with-mail_ssl_module 为了代理任何一种类型的使用SSL/TLS的Mail,激活该模块

--without-mail_pop3_module 在启用 Mail 模块后，单独地禁用 pop3 模块

--without-mail_imap_module 再启用 mail 模块后，单独地禁用 IMAP 模块

--without-mail_smtp_module 在启用 mail 模块后，单独地禁用 smtp 模块

--without--http 该选项将完全禁用 http 模块，如果你只想支持

3. 指定路径的配置选项：

--without-http_perl_module Nginx 配置能够使用扩展使用 Perl 代码，这个选项启用这个模块（此模块会降低性能）

--without-perl_module_path=<path> 对于额外嵌入的 Perl 模块，使用该选项指定该 Perl 解析器的路径，也可以通过配置选项来指定 Perl 模块解析器的位置

--without—perl=<path> 如果在默认路径中没有找到 Perl，那么指定 Perl 的路径

--http-log-path=<path> Http 访问日志的默认路径

--http-client-body-temp-path=<path> 从客户端收到请求后，该选项设置的目录用于作为请求体临时存放的目录。如果 WebDAV 模块启用，那么推荐设置该路径为同一文件系统上的目录作为最终的目的地

--http-proxy-temp-path=<path> 在使用代理后，通过该选项设置存放临时文件路径

--http-fastcgi-temp-path=<path> 设置 FastCGI 临时文件的目录

--http-uwsgi-temp-path=<path> 设置 uWSGI 临时文件的目录

--http-scgi-temp-path=<path> 设置 SCGI 临时文件的目录

4. 各种模块配置选项：

--with-http_ssl_module 如果需要对流量进行加密，可以使用该选项，在 URLs 中开始部分将会是 https（需要 OpenSSL 库）

--with-http_realip_module 如果你的 Nginx 在七层负载均衡器或者是其他设备之后，它们将 http 头中的客户端 IP 地址传递，那么你将会需要启用这个模块。在多个客户处于一个 IP 地址的情况下使用

--with-http_addition_module 这个模块作为输出过滤器，使你能够在请求经过一个 location 前或者后时在该 location 本身添加内容

--with-http_xslt_module 该模块用于处理 XML 响应转换 ,基于一个或者多个 XSLT 格式（需要 libxml2 和 libxslt 库）

--with-http_image_filter_module 该模块被作为图像过滤器使用 ,在将图形投递到客户之前进行处理（需要 libgd 库）

--with-http_geoip_module 使用该模块，能够设置各种变量以便在配置文件中区段使用，基于地理位置查找客户端 IP 地址（需要 MaxMfind GeoIP 库和相应的预编译数据库文件）

--with-http_sub_module 该模块实现了替代过滤，在响应中用一个字符串替代另一个字符串

--with-http_dav_module 启用这个模块将激活使用 WebDAV 的配置指令。注意：这个模块也只在有需要使用的基础上启用，如果配置不正确，它将带来安全问题。

--with-http_flv_module 如果需要提供 Flash 流媒体视频文件，那么该模块将会提供伪流媒体

--with-http_mp4_module 这个模块支持 H.264/AAC 文件伪流媒体

--with-http_gzip_module 当被调用的资源没有.gz 结尾格式的文件时，如果想支持发送预压缩版本的静态文件，那么使用该模块

--with-http_gunzio_module 对应不支持 gzip 编码的客户，该模块用于为客户解压缩预压缩内容

--with-http_random_index_module 如果你想提供从一个目录中随机选择文件的索引文件，那么这个模块需要被激活

--with-http_secure_link_module 该模块提供了一个机制，它会将一个哈希值链接到一个 URL 中，因此，只有那些使用正确的密码能够计算链接

--with-http_stub_status_module 启用这个模块后会收集 Nginx 自身的状态信息。输出的状态信息可以使用 RRDtool 或类似的东西来绘制成图

5.编译安装

编译前需要安装的软件包

yum –y install zlib zlib-devel gzip pcre pcre-devel openssl openssl-devel gzip-devel

./configure –prefix=/usr/local/nginx

make

make install

[root@solr1 nginx]# pwd

/usr/local/nginx

[root@solr1 nginx]# ls

client_body_temp conf fastcgi_temp html logs proxy_temp sbin scgi_temp uwsgi_temp

设置开机启动：

nginx.txt

/etc/init.d/nginx

Chkconfig nginx on

Service nginx start/stop/restart/reload

/usr/local/nginx/sbin/nginx

定时分割 nginx 日志脚本：

#nginx 日志切割脚本

#!/bin/bash

logs_path="/usr/localinx/logs/"

pid_path="/usr/localinx/logsinx.pid"

itime=`date -d yesterday +%Y%m%d`

#重命名日志文件

mv ${logs_path}shuyuanaccess.log ${logs_path}shuyuan/shuyuanaccess_$itime.log

mv ${logs_path}access.log ${logs_path}xuezhiezhiaccess_$itime.log

find ${logs_path}/shuyuan/ -mtime +7 -exec rm -f {} \;

#向 nginx 主进程发信号重新打开日志

kill -USR1 `cat ${pid_path}`

设置计划任务：

Crontab –e

0 0 * * * /bin/sh /usr/local/nginx/logs/nginx_log.sh

5.nginx 配置

1.配置文件详解

#运行用户

user www;

#启动进程,通常设置成和 cpu 的数量相等

worker_processes 1;

#全局错误日志及 PID 文件

error_log /usr/local/nginx/logs/error.log;

pid /usr/local/nginx/logs/nginx.pid;

log_format access '$remote_addr - $remote_user [$time_local] "$request" '

'$status $body_bytes_sent "$http_referer" '

'"$http_user_agent" $http_x_forwarded_for';

$remote_addr 与$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 ：记录客户毒啊浏览器的相关信息；

#工作模式及连接数上限

events {

use epoll; #epoll 是多路复用 IO(I/O Multiplexing)中的一种方式,但是仅用于 linux2.6 以上内核,可以大大提高 nginx 的性能

worker_connections 1024;#单个后台 worker process 进程的最大并发链接数

multi_accept on;

}

#设定 http 服务器，利用它的反向代理功能提供负载均衡支持

```
http {
    #设定 mime 类型,类型由 mime.type 文件定义
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    #设定日志格式
    access_log    /usr/local/nginx/logs/access.log;
    #sendfile 指令指定 nginx 是否调用 sendfile 函数（zero copy 方式）来输出文件，对于普通应用，
    #必须设为 on,如果用来进行下载等应用磁盘 IO 重负载应用，可设置为 off，以平衡磁盘与网络 I/O 处理速度，降低系统的 uptime.
    sendfile        on;
    #tcp_nopush     on;
    #连接超时时间
    #keepalive_timeout  0;
    keepalive_timeout  65;
    tcp_nodelay        on;
    #开启 gzip 压缩
    gzip  on;
    gzip_disable "MSIE [1-6]\.(?!.*SV1)";
    #设定请求缓冲
    client_header_buffer_size    1k;
    large_client_header_buffers  4 4k;
```

```
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
#设定负载均衡的服务器列表
 upstream mysvr {
#weigth 参数表示权值，权值越高被分配到的几率越大
server 192.168.8.1:3128 weight=5;
server 192.168.8.2:80   weight=1;
server 192.168.8.3:80   weight=6;
}
server {
#侦听 80 端口
    listen          80;
    #定义使用 www.xx.com 访问
    server_name   www.xx.com;
    #设定本虚拟主机的访问日志
    access_log   logs/www.xx.com.access.log   access;

#默认请求
location / {
        root    /root;        #定义服务器的默认网站根目录位置
        index index.php index.html index.htm;    #定义首页索引文件的名称
    }
```

```
        # 定义错误提示页面
        error_page     500 502 503 504 /50x.html;
        location = /50x.html {
            root     /root;
        }
        #静态文件，nginx 自己处理
        location ~ ^/(images|javascript|js|css|flash|media|static)/ {
            root /var/www/virtual/htdocs;
            #过期 30 天，静态文件不怎么更新，过期可以设大一点，如果频繁更新，则可以设置得小一点。
            expires 30d;
        }
        #PHP 脚本请求全部转发到 FastCGI 处理. 使用 FastCGI 默认配置.
        location ~ \.php$ {
            root /root;
            fastcgi_pass 127.0.0.1:9000;
            fastcgi_index index.php;
            fastcgi_param                                    SCRIPT_FILENAME /home/www/www$fastcgi_script_name;
            include fastcgi_params;
        }
        #设定查看 Nginx 状态的地址
```

```
        location /NginxStatus {
            stub_status                 on;
            access_log                   on;
            auth_basic                   "NginxStatus";
            auth_basic_user_file    conf/htpasswd;
        }
        #禁止访问 .htxxx 文件
        location ~ /\.ht {
            allow host;
            deny all;
        }
         }
    }
```

2.nginx 日志格式

日志对于统计排错来说非常有利的。nginx 日志相关的配置如 access_log、log_format、open_log_file_cache、log_not_found、log_subrequest、rewrite_log、error_log。

nginx 有一个非常灵活的日志记录模式。每个级别的配置可以有各自独立的访问日志。日志格式通过 log_format 命令来定义。ngx_http_log_module 是用来定义请求日志格式的。

1. access_log 指令

语法: access_log path [format [buffer=size [flush=time]]];

access_log path format gzip[=level] [buffer=size] [flush=time];

access_log syslog:server=address[,parameter=value] [format];

access_log off;

默认值: access_log logs/access.log combined;

配置段: http, server, location, if in location, limit_except

gzip 压缩等级。

buffer 设置内存缓存区大小。

flush 保存在缓存区中的最长时间。

不记录日志：access_log off;

使用默认 combined 格式记录日志：access_log logs/access.log 或 access_log logs/access.log combined;

2. log_format 指令

语法: log_format name string ...;

默认值: log_format combined "...";

配置段: http

name 表示格式名称，string 表示等义的格式。log_format 有一个默认的无需设置的 combined 日志格式，相当于 apache 的 combined 日志格式，如下所示：

```
log_format combined '$remote_addr - $remote_user [$time_local] '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent" ';
```

如果 nginx 位于负载均衡器，squid，nginx 反向代理之后，web 服务器无法直接获取到客户端真实的 IP 地址了。 $remote_addr 获取反向代理的 IP 地址。反向代理服务器在转发

请求的 http 头信息中，可以增加 X-Forwarded-For 信息，用来记录 客户端 IP 地址和客户端请求的服务器地址。如下所示：

log_format porxy '$http_x_forwarded_for - $remote_user [$time_local] '

' "$request" $status $body_bytes_sent '

' "$http_referer" "$http_user_agent" ';

123.125.71.86 - - [17/Jul/2015:15:24:23 +0800] "GET /data/cache/forum_slide.js?GKl HTTP/1.1" 200 4082 "http://www.iyunv.com/forum.php?fid=54&digest=1" "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)" -

日志格式允许包含的变量注释如下：

$remote_addr, $http_x_forwarded_for 记录客户端 IP 地址

$remote_user 记录客户端用户名称

$request 记录请求的 URL 和 HTTP 协议

$status 记录请求状态

$body_bytes_sent 发送给客户端的字节数，不包括响应头的大小； 该变量与 Apache 模块 mod_log_config 里的“%B”参数兼容。

$bytes_sent 发送给客户端的总字节数。

$connection 连接的序列号。

$connection_requests 当前通过一个连接获得的请求数量。

$msec 日志写入时间。单位为秒，精度是毫秒。

$pipe 如果请求是通过 HTTP 流水线(pipelined)发送，pipe 值为“p”，否则为“.”。

$http_referer 记录从哪个页面链接访问过来的

$http_user_agent 记录客户端浏览器相关信息

$request_length 请求的长度（包括请求行，请求头和请求正文）。

$request_time 请求处理时间，单位为秒，精度毫秒； 从读入客户端的第一个字节开始，直到把最后一个字符发送给客户端后进行日志写入为止。

$time_iso8601 ISO8601 标准格式下的本地时间。

$time_local 通用日志格式下的本地时间。

发送给客户端的响应头拥有“sent_http_”前缀。 比如$sent_http_content_range。

实例如下：

```
http {
 log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                                    '"$status"          $body_bytes_sent
"$http_referer" '
                                    '"$http_user_agent"
"$http_x_forwarded_for" '
                                    '"$gzip_ratio" $request_time $bytes_sent
$request_length';
 log_format srcache_log '$remote_addr - $remote_user [$time_local] "$request" '
                                '"$status"    $body_bytes_sent    $request_time
$bytes_sent $request_length '
                                '[$upstream_response_time]
[$srcache_fetch_status] [$srcache_store_status] [$srcache_expire]';
 open_log_file_cache max=1000 inactive=60s;
```

```
 server {
   server_name ~^(www\.)?(.+)$;
   access_log logs/$2-access.log main;
   error_log logs/$2-error.log;
   location /srcache {
     access_log logs/access-srcache.log srcache_log;
   }
 }
}
```

3. open_log_file_cache 指令

语法: open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time];

open_log_file_cache off;

默认值: open_log_file_cache off;

配置段: http, server, location

对于每一条日志记录，都将是先打开文件，再写入日志，然后关闭。可以使用 open_log_file_cache 来设置日志文件缓存(默认是 off)，格式如下：

参数注释如下：

max:设置缓存中的最大文件描述符数量，如果缓存被占满，采用 LRU 算法将描述符关闭。

inactive:设置存活时间，默认是 10s

min_uses:设置在 inactive 时间段内，日志文件最少使用多少次后，该日志文件描述符记入缓存中，默认是 1 次

valid:设置检查频率，默认 60s

off：禁用缓存

实例如下：

open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;

4. log_not_found 指令

语法: log_not_found on | off;

默认值: log_not_found on;

配置段: http, server, location

是否在 error_log 中记录不存在的错误。默认是。

5. log_subrequest 指令

语法: log_subrequest on | off;

默认值: log_subrequest off;

配置段: http, server, location

是否在 access_log 中记录子请求的访问日志。默认不记录。

6. rewrite_log 指令

由 ngx_http_rewrite_module 模块提供的。用来记录重写日志的。对于调试重写规则建议开启。 Nginx 重写规则指南

语法: rewrite_log on | off;

默认值: rewrite_log off;

配置段: http, server, location, if

启用时将在 error log 中记录 notice 级别的重写日志。

7. error_log 指令

语法: error_log file | stderr | syslog:server=address[,parameter=value] [debug | info |

notice | warn | error | crit | alert | emerg];

默认值: error_log logs/error.log error;

配置段: main, http, server, location

配置错误日志。

3.location 语法

语法规则： location [=|~|~*|^~] /uri/ { … }

= 开头表示精确匹配

^~ 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（注意是空格）。

~ 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

!~和!~*分别为区分大小写不匹配及不区分大小写不匹配 的正则

/ 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 ^~ /img/ {
    #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/img/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/img/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，所以想想看实际应用中哪里会用到。

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，个人觉得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，官网如是说。
#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 ^~ /sttic/ {
```

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

小结：

1、等号优先级最高，工作中尽量把等号匹配的放在前边

2、~* \.jpg$优先级第二

3、^~ /a/优先级第三

```
#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/
}
```

三、ReWrite 语法

last – 基本上都用这个 Flag。

break – 中止 Rewirte，不在继续匹配

redirect – 返回临时重定向的 HTTP 状态 302

permanent – 返回永久重定向的 HTTP 状态 301

1、下面是可以用来判断的表达式：

-f 和!-f 用来判断是否存在文件

-d 和!-d 用来判断是否存在目录

-e 和!-e 用来判断是否存在文件或目录

-x 和!-x 用来判断文件是否可执行

2、下面是可以用作判断的全局变量

例：http://localhost:88/test1/test2/test.php

$host：localhost

$server_port：88

$request_uri：http://localhost:88/test1/test2/test.php

$document_uri：/test1/test2/test.php

$document_root：D:\nginx/html

$request_filename：D:\nginx/html/test1/test2/test.php

四、Redirect 语法

server {

listen 80;

server_name start.iyunv.cn;

index index.html index.php;

root html;

if ($http_host !~ “^star\.iyunv\.cn$&quot {

rewrite ^(.*) http://star.iyunv.cn$1 redirect;

}

}

五、防盗链

```
location ~* \.(gif|jpg|swf)$ {
valid_referers none blocked start.iyunv.cn sta.iyunv.cn;
if ($invalid_referer) {
rewrite ^/ http://$host/logo.png;
}
}
```

六、根据文件类型设置过期时间

```
location ~* \.(js|css|jpg|jpeg|gif|png|swf)$ {
if (-f $request_filename) {
expires 1h;
break;
}
}
```

七、禁止访问某个目录

```
location ~* \.(txt|doc)${
root /data/www/wwwroot/iyunv/test;
deny all;
}
```

4.nginx 负载均衡

Nginx 负载均衡一些基础知识:

Nginx 的负载均衡实现过程 :首先在 http 模块中配置使用 upstream 模块定义后台的 web

server 的池子，名为 proxy-web，在池子中我们可以添加多台后台 webserver，其中状态检查、调度算法都是在池子中配置；然后在 serverr 模块中定义虚拟主机，但是这个虚拟主机不指定自己的 web 目录站点，它将使用 location 匹配 url 然后转发到上面定义好的 web 池子中，最后根据调度策略再转发到后台 web server 上

nginx 的 upstream 分配方式

（1）rr 轮询（默认）

按照请求顺序分配到每个 RS，和 lvs 中的 rr 算法一样，如果 RS 宕机，会自动剔除，默认情况下只检测 80 端口，如果 RS 报 402、403、503、504 错误，会直接返回给客户端。

（2）weight（权重）

在 rr 的基础上再加上权重（默认是 rr+weight），权重轮询和访问成正比，值越大分配的越多，可以根据服务器的配置设置权重，可以解决服务器性能不均进行请求分配的问题

（3）ip_hash

解决动态网页 session 共享问题

每个访问请求按照 IP 地址的 hash 值进行分配，ip 的 hash 值只要相同就会被分配到同一台服务器上（lvs 负载均衡的-p 参数，keepalived 配置里的 persistence_timeout50），该调度算法可以解决动态网页 session 共享问题，但有时会导致请求分配不均，

提示：由于国内用的都是 nat 模式，所以 hash 不适合使用

ip_hash 不能和其他的算法一块使用，即不能使 weight 或 backup

（4）fair（第三方）

按照后端服务器的响应时间来配置，响应时间短的优先分配，比上面的都更智能，此种算法可以按照页面大小和加载时间长短智能的进行负载均衡，nginx 本身不支持 fair，

需要下载 nginx 的 upstrea_fair 模块

（5）url_hash（第三方）

主要应用于缓存服务器上

按照访问的 url 来分配请求，让相同的 url 定向到同一个服务器，后端服务器为缓存服务器的时候效果更显著，在 upstream 中加入 hash 语句，server 语句中不能写入 weight 等其他的参数，hash_method 是使用的 hash 算法。

缺点：如果有一台机器宕机了，那就苦了，consistent_hash 可以解决这个问题

可以提高后端缓存服务器的效率，nginx 本身不支持 url_hash 的，需要下载 hash 软件

（6）least_conn

最少连接数，哪个连接少就分配到哪台设备

（7）consistent_hash

一致性算法

配置：

在 http 节点里添加:

```
#定义负载均衡设备的 Ip 及设备状态
upstream proxy_nginx {
        server 192.168.0.254 weight=1max_fails=2 fail_timeout=10s ;
        server 192.168.0.253 weight=2 max_fails=2fail_timeout=10s;
                server192.168.0.252 backup ;
                server192.168.0.251    down ;
    }
```

server 192.168.0.254： 后台 RS，可以是域名或 IP，默认是 80 端口，也可加上:80 指定

wight = 1 权重比 默认是 1

max_fails=2 健康检查的最大失败次数，超过此次数表示该 RS 不可用，默认是 1，0 表示禁止失败尝试。生产环境一般设置 2~3 次

fail_timeout=10s 失败的超时时间，默认是 10s

backup 热备配置，当前面的 RS 全部不可用时自动启动

down 表示该服务永远不可用

注意：max_fails 设置的越低用户体验越好，但是设置低了也有个缺点，就是 proxy 可能会误判 RS 的状态，而且 RS 越少误判的几率越大，误判会对业务产生巨大影响，当 RS 的数量比较少时建议将该值设置的大点。

在需要使用负载的 Server 节点下添加

```
server{
listen 80;
server_name a.com;
location / {
    proxy_pass http:// proxy_nginx;
    proxy_set_header    Host                $host;
    proxy_set_header    X-Real-IP           $remote_addr;
    proxy_set_header    X-Forwarded-For   $proxy_add_x_forwarded_for;
}
```

Nginx 还支持多组的负载均衡,可以配置多个 upstream 来服务于不同的 Server.

配置负载均衡比较简单,但是最关键的一个问题是怎么实现多台服务器之间 session 的共享

下面有几种方法(以下内容来源于网络,第四种方法没有实践.)

1) 不使用 session，换作 cookie

能把 session 改成 cookie，就能避开 session 的一些弊端，在从前看的一本 J2EE 的书上，也指明在集群系统中不能用 session，否则惹出祸端来就不好办。如果系统不复杂，就优先考虑能否将 session 去掉，改动起来非常麻烦的话，再用下面的办法。

2) 应用服务器自行实现共享

asp.net 可以用数据库或 memcached 来保存 session，从而在 asp.net 本身建立了一个 session 集群，用这样的方式可以令 session 保证稳定，即使某个节点有故障，session 也不会丢失，适用于较为严格但请求量不高的场合。但是它的效率是不会很高的，不适用于对效率 要求高的场合。

以上两个办法都跟 nginx 没什么关系，下面来说说用 nginx 该如何处理：

3) ip_hash

nginx 中的 ip_hash 技术能够将某个 ip 的请求定向到同一台后端，这样一来这个 ip 下的某个客户端和某个后端就能建立起稳固的 session，ip_hash 是在 upstream 配置中定义的：

```
upstream backend {
  server 127.0.0.1:8080 ;
  server 127.0.0.1:9090 ;
   ip_hash;
}
```

5.nginx 反向代理

location / {

proxy_pass http://127.0.0.1:88;

proxy_redirect off;# proxy_redirect 其作用是对发送给客户端的 URL 进行修改。

proxy_set_header X-Real-IP $remote_addr;

#后端的 Web 服务器可以通过 X-Forwarded-For 获取用户真实 IP

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

#以下是一些反向代理的配置，可选。

proxy_set_header Host $host;

注意：

“proxy_set_header”当我们的 RS 有多个虚拟主机（相同的 ip，相同的端口）的时候如 www、bbs、blog，代理服务器怎么知道将请求发到哪呢，这个时候 nginx 代理就会查找 proxy_set_header 参数，将请求发送到相应域名的虚拟主机上。

client_max_body_size 10m; #允许客户端请求的最大单文件字节数

client_body_buffer_size 128k; #缓冲区代理缓冲用户端请求的最大字节数，

proxy_connect_timeout 90; #nginx 跟后端服务器连接超时时间(代理连接超时)

proxy_send_timeout 90; #后端服务器数据回传时间(代理发送超时)

proxy_read_timeout 90; #连接成功后，后端服务器响应时间(代理接收超时)

proxy_buffer_size 4k; #设置代理服务器（nginx）保存用户头信息的缓冲区大小

proxy_buffers 4 32k; #proxy_buffers 缓冲区，网页平均在 32k 以下的设置

proxy_busy_buffers_size 64k; #高负荷下缓冲大小（proxy_buffers*2）

proxy_temp_file_write_size 64k;

#设定缓存文件夹大小，大于这个值，将从 upstream 服务器传

}

代理转发路径案例：

nginx 配置 proxy_pass，需要注意转发的路径配置

1、location /test/ {

proxy_pass http://t6:8300;

}

2、location /test/ {

proxy_pass http://t6:8300/;

}

上面两种配置，区别只在于 proxy_pass 转发的路径后是否带 “/”

针对情况 2，如果访问 url = http://server/test/test.jsp，则被 nginx 代理后，请求路径会变为 http://proxy_pass/test.jsp，直接访问 server 的根资源

针对情况 1，如果访问 url = http://server/test/test.jsp，则被 nginx 代理后，请求路径会便问 http://proxy_pass/test/test.jsp，将 test/ 作为根路径，请求 test/路径下的资源

典型实例：

同一个域名下，根据根路径的不同，访问不同应用及资源

例如：A 应用 http://server/a ；B 应用 http://server/b

A 应用和 B 应用共同使用访问域名 http://server；

配置 nginx 代理转发时，如果采用情况 2 的配置方式，则会导致访问 http://server/a/test.jsp 时，代理到 http://proxy_pass/test.jsp，导致无法访问到正

确的资源，页面中如果有对根资源的访问，也都会以 http://server 做为根路径访问资源，导致资源失效

针对此类情况，需要采用情况 1，分别针对不用应用，设置不同的根资源路径，并保证代理后的根路径也依然有效

6.proxy_cache

1 proxy_cache

语法：proxy_cache zone_name;

默认值：None

使用字段：http, server, location

设置一个缓存区域的名称，一个相同的区域可以在不同的地方使用。

在 0.7.48 后 ,缓存遵循后端的" Expires"，"Cache-Control: no-cache"，"Cache-Control: max-age=XXX" 以及其他等。然而 ,目前 nginx 会忽略一些缓存控制指令 ,如 :" private" 和" no-store"，同样，nginx 在缓存过程中不会处理" Vary" 头，为了确保一些私有数据不被所有的用户看到，后端必须设置 "no-cache" 或者" max-age=0"头，或者 proxy_cache_key 包含用户指定的数据如$http_cookie_xxx ,在 proxy_cache_key 中使用一部分 cookie 的值可以防止缓存私有数据 ,所以可以分别指定 location 以便分开私有数据和公有数据。

缓存指令依赖代理缓冲区(buffers)，如果 proxy_buffers 设置为 off，缓存不会生效。

2 proxy_cache_key

语法：proxy_cache_key line;

默认值：$scheme$proxy_host$request_uri;

使用字段：http, server, location

指令指定了包含在缓存中的缓存关键字。

proxy_cache_key "$host$request_uri$cookie_user";

proxy_cache_key "$scheme$host$request_uri";

3 proxy_cache_path

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

默认值：None

使用字段：http

指令指定缓存的路径和一些其他参数，缓存的数据存储在文件中。缓存的文件名和 key 为代理 URL 的 MD5 码。levels 参数指定缓存的子目录数，例如：

proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;

文件名类似于：

/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c

所有活动的 key 和元数据存储在共享的内存区域中，这个区域用 keys_zone 参数指定，如果在 inactive 参数指定的时间内缓存的数据没有被请求则被删除 默认 inactive 为 10 分钟。cache manager 进程控制磁盘的缓存大小，在 max_size 参数中定义，超过其大小后最少使用数据将被删除。

区域的大小按照缓存页面数的比例进行设置，一个页面（文件）的元数据大小按照操作系统来定，FreeBSD/i386 下为 64 字节，FreeBSD/amd64 下为 128 字节，当区域满了以后 key 将按照 LRU（最近最少使用算法）进行处理。

proxy_cache_path 和 proxy_temp_path 应该使用在相同的文件系统上。

4 proxy_cache_methods

语法：proxy_cache_methods [GET HEAD POST];

默认值：proxy_cache_methods GET HEAD;

使用字段：http, server, location

GET/HEAD 用来装饰语句，即你无法禁用 GET/HEAD 即使你只使用下列语句设置：

proxy_cache_methods POST;

5 proxy_cache_min_uses

语法：proxy_cache_min_uses the_number;

默认值：proxy_cache_min_uses 1;

使用字段：http, server, location

多少次的查询后应答将被缓存，默认 1。

6 proxy_cache_valid

语法：proxy_cache_valid reply_code [reply_code ...] time;

默认值：None

使用字段：http, server, location

为不同的应答设置不同的缓存时间，例如：

```
proxy_cache_valid  200 302  10m;
proxy_cache_valid  404      1m;
proxy_cache_valid  5m;
proxy_cache_valid  200 302 10m;
proxy_cache_valid  301 1h;
proxy_cache_valid  any 1m;
```

7 proxy_cache_use_stale

为了防止缓存失效（在多个线程同时更新本地缓存时），你可以指定’updating’参数，它将保证只有一个线程去更新缓存，并且在这个线程更新缓存的过程中其他的线程只会响应当前缓存中的过期版本。

代码及 configure 配置：

在 ngx_http_proxy_module.c 里面定义了每个指令的钩子(即 callback)，它们在读取配置文件时会被调用。在 configure 的时候只需要把“HTTP_CACHE”设置为 YES(可以找到 auto/options 里面 HTTP_CACHE 那行)。“proxy_cache_purge”指令需要下载 nginx add-ons 里面的“Cache Purge”模块，并在 configure 的时候用"--add-module="来加载代码。

配置文件例子：

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';
    sendfile        on;
    tcp_nopush     on;
    keepalive_timeout  65;
    tcp_nodelay on;
    proxy_temp_path  /data/proxy_temp_path;
```

proxy_cache_path /data/proxy_path levels=1:2 keys_zone=cache_one:200m inactive=5m max_size=2m clean_time=1m;

设置 Web 缓存区名称为 cache_one，内存缓存空间大小为 200MB，

inactive 的 5m 指缓存默认时长 5 分钟，5 分钟没有被访问的内容自动清除,这里 inactive 可以设置为天 d、时 h、分 m、秒 s，

注意这个配置是在 server 标签外，levels 指定该缓存空间有两层 hash 目录，第一层目录是 1 个字母，第二层为 2 个字母

max_size 的 2m 是指单个文件超过 2m 的就不缓存；

clean_time 指定一分钟清理一次缓存。

```
upstream haha {
server 127.0.0.1:8001 weight=2;
server 10.10.10.57       weight=1;
}
server {
    listen          80;
    server_name    www.qaz.com;
location ~ \.html$ {
        proxy_pass http://127.0.0.1:8002;
}
    location / {
        proxy_cache cache_one;
        proxy_cache_valid 200 304 12h;
```

```
            proxy_cache_valid 301 302 1m;
            proxy_cache_valid any 1m;
            proxy_cache_key $host$uri$is_args$args;
            proxy_pass http://haha;
          }
        location ~ /purge(/.*)
          {
            allow          127.0.0.1;
            allow          10.10.10.67;
            deny           all;
            proxy_cache_purge      cache_one    $host$1$is_args$args;
          }
    }
```

7.nginx 开启 ssl

```
server {
        listen          443;
        server_name    localhost;
        ssl_silent_errors on; #  如果在处理 SSI 的过程中出现 "[an error occurred while processing the directive]" 错误，禁止将其输出。
        ssl_types  text/shtml;   #默认只解析 text/html 类型，这个参数可以指定其他的 MIME 类型。
```

```
        ssl                         on;
        ssl_certificate_key    /opt/nginx/ca/server/server.key;
        ssl_client_certificate /opt/nginx/ca/private/ca.crt;
        ssl_session_timeout    5m;   指 session 变量生存时间的长短，服务器默认是 20
分钟
        ssl_verify_client on;   #开户客户端证书验证
        ssl_protocols    SSLv2 SSLv3 TLSv1;
        ssl_ciphers
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
        ssl_prefer_server_ciphers     on;
       location / {
        root /home;
        index index.asp index.aspx;
    }
    }
```

关于 ssl 的一些指令：

指令

ssl

语法：ssl [on|off]

默认值：ssl off

使用字段：main, server

开启 HTTPS。

ssl_certificate

语法：ssl_certificate file

默认值：ssl_certificate cert.pem

使用字段：main, server

为这个虚拟主机指定 PEM 格式的证书文件，这个文件可以包含其他的证书和服务器私钥，同样，密钥也必须是 PEM 格式，0.6.7 版本以后，这里的路径为相对于 nginx.conf 的路径，而不是编译时的 prefix 路径。

ssl_certificate_key

语法：ssl_certificate_key file

默认值：ssl_certificate_key cert.pem

使用字段：main, server

为这个虚拟主机指定 PEM 格式的私钥，0.6.7 版本以后，这里的路径为相对于 nginx.conf 的路径，而不是编译时的 prefix 路径。

ssl_client_certificate

语法：ssl_client_certificate file

默认值：none

使用字段：main, server

指出包含 PEM 格式的 CA（root）证书，它将用于检查客户端证书。

ssl_dhparam

语法：ssl_dhparam file

默认值：none

使用字段：main, server

该指令指定了一个 PEM 格式的文件，其中包含的 Diffie-Hellman 密钥协商协议密码参数，它将用于服务器和客户端之间的会话密钥交换。（翻译来自 Google。- -!）

原文：This directive specifies a file containing Diffie-Hellman key agreement protocol cryptographic parameters, in PEM format, utilized for exchanging session keys between server and client

ssl_ciphers

语法：ssl_ciphers file

默认值：ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

使用字段：main, server

指出为建立安全连接，服务器所允许的密码格式列表，密码指定为 OpenSSL 支持的格式，如：

ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

查看当前安装的 OpenSSL 版本所支持的密码列表，可以使用下列命令：

openssl ciphers

ssl_crl

语法：ssl_crl file

默认值：none

使用字段：http, server

指令可用于 0.8.7 以后版本，用于指定一个证书吊销列表的文件名，使用 PEM 格式，它将用于检查证书吊销状态。

ssl_prefer_server_ciphers

语法：ssl_prefer_server_ciphers [on|off]

默认值：ssl_prefer_server_ciphers off

使用字段：main, server

依赖 SSLv3 和 TLSv1 协议的服务器密码将优先于客户端密码。

ssl_protocols

语法：ssl_protocols [SSLv2] [SSLv3] [TLSv1]

默认值：ssl_protocols SSLv2 SSLv3 TLSv1

使用字段：main, server

指定要开启的 SSL 协议。

ssl_verify_client

语法：ssl_verify_client on|off|optional

默认值：ssl_verify_client off

使用字段：main, server

启用客户端身份验证，参数“optional”将使用客户端提供的证书检查它的权限(0.8.7 与 0.7.63 版本之前为“ask”)。

ssl_verify_depth

语法：ssl_verify_depth number

默认值：ssl_verify_depth 1

使用字段：main, server

设置服务器按顺序检查客户端提供的证书链的长度。

ssl_session_cache

语法：ssl_session_cache off|none|builtin:size and/or shared:name:size

默认值：ssl_session_cache off

使用字段： main, server

设置储存 SSL 会话的缓存类型和大小。

缓存类型为：

off - 强制关闭：nginx 告诉客户端这个会话已经不能被再次使用。

none - 非强制关闭：nginx 告诉客户端这个会话可以被再次使用，但是 nginx 实际上并不使用它们 ,这是为某些使用 ssl_session_cache 的邮件客户端提供的一种变通方案，可以使用在邮件代理和 HTTP 服务器中。

builtin - 内建 OpenSSL 缓存，仅能用在一个工作进程中，缓存大小在会话总数中指定 ,注意 如果要使用这个类型可能会引起内存碎片问题 ,具体请查看下文中参考文档。

shared - 缓存在所有的工作进程中共享，缓存大小指定单位为字节，1MB 缓存大概保存 4000 个会话，每个共享的缓存必须有自己的名称，相同名称的缓存可以使用在不同的虚拟主机中。

可以同时使用两个缓存类型，内建和共享，如：

ssl_session_cache builtin:1000 shared:SSL:10m;

但是，使用共享缓存而不使用内建缓存将更为有效。

0.8.34 版本之前如果 ssl_verify_client 设置为'on'或者'optional'时这里不能设置为 none 或 off。

注意，为了让缓存恢复工作，你需要在服务器为 SSL socket 配置默认服务器，例如：

listen [::]:443 ssl default_server;

这是因为缓存的重用发生在任何 TLS 扩展启用之前 ,即服务器名称认证(Server Name

Identification (SNI))，来自一个指定 IP 地址（服务器）的 ClientHello 信息请求一个会话 ID，为了使其工作，默认（default）服务器必须被设置。

最好的方法是将 ssl_session_cache 放到 http 字段 ,这样下面配置的 server/虚拟主机字段都将得到相同的 SSL 缓存配置。

ssl_session_timeout

语法：ssl_session_timeout time

默认值：ssl_session_timeout 5m

使用字段： main, server

设置客户端能够反复使用储存在缓存中的会话参数时间。

ssl_engine

语法：ssl_engine

指定使用的 OpenSSL 引擎，如 Padlock，例如 PadLock，下列命令可以查看你的 OpenSSL 支持的引擎：

openssl engine

Debian 系统在版本 0.9.8o 的 OpenSSL 运行后返回：

$ openssl engine

(padlock) VIA PadLock (no-RNG, no-ACE)

(dynamic) Dynamic engine loading support

内置变量

ngx_http_ssl_module 模块支持下列内建变量：

$ssl_cipher - 返回建立的 SSL 连接中那些使用的密码字段。

$ssl_client_serial - 返回建立的 SSL 连接中客户端证书的序列号。

$ssl_client_s_dn - 返回建立的 SSL 连接中客户端证书的 DN 主题字段。

$ssl_client_i_dn - 返回建立的 SSL 连接中客户端证书的 DN 发行者字段。

$ssl_protocol - 返回建立的 SSL 连接中使用的协议。

$ssl_session_id - 需要 0.8.20 以上版本。

$ssl_client_cert

$ssl_client_raw_cert

$ssl_client_verify - 如果客户端证书审核通过，这个变量为“SUCCESS”。

非标准错误代码

这个模块支持一些非标准的 HTTP 错误代码，可以借助 error_page 指令用于调试：

495 - 检查客户端证书时发生错误。

496 - 客户端无法提供必须的证书。

497 - 传送到 HTTPS 的普通请求。

在请求完整的废除后调试完成，并取得一些内置变量，如：$request_uri, $uri, $arg 等。

8.nginx 常用的优化

Nginx 配置和优化

1.隐藏版本号默认情况下，使用 curl 命令会把 nginx 的版本信息等获取到，如：

[root@transit_machine ~]# curl -I -H"www.iyunv.com" "http://172.16.254.5/index.php"

HTTP/1.1 200 OK

Server:nginx/1.6.3

Date: Wed, 24 Jun 2015 02:50:59 GMT

Content-Type: text/html

Connection: keep-alive

X-Powered-By: PHP/5.4.30

现在通过添加 server_tokensoff 参数隐藏版本号.在 http{}模块中添加

http {

....

server_tokens off;

}

继续 curl 一下，发现 Server:后边的 nginx 版本号已经没了

[root@transit_machine ~]# curl -I -H"www.iyunv.com" "http://172.16.254.5/index.php"

HTTP/1.1 200 OK

Server:nginx

Date: Wed, 24 Jun 2015 02:54:54 GMT

Content-Type: text/html

Connection: keep-alive

X-Powered-By: PHP/5.4.30

2.客户端 curl 某个页面，返回 404 在 server 模块中加入下面规则即可

```
if ($http_user_agent ~*"curl"){
        return 404;
    }
```

3.根据 CPU 核数进行 nginx 进程优化

四核 cpu 配置：

worker_processes 4;

worker_cpu_affinity 0001 0010 01001000;

八核 cpu 服务器参数配置：

worker_processes 8;

worker_cpu_affinity 0000000100000010 00000100 00001000 00010000 00100000 01000000 10000000;

worker_cpu_affinity 0001 0010 01001000 0001 0010 0100 1000;

官方文档说明

worker_processes 4;

worker_cpu_affinity 0001 0010 01001000;

binds each worker process to aseparate CPU, while

worker_processes 2; 4 核 2 进程

worker_cpu_affinity 0101 1010;

4.调整 nginx 事件处理模型、客户端最大连接数、最大打开的文件数

4.1 nginx 的连接处理机制在不同的操作系统上采用不用的 IO 模型，在 linux 下，nginx 使用 epoll 的 IO 多路复用模型，在 freebsd 使用 kqueue 的 IO 多路复用模型，在 solaris 使用/dev/pool 方式的 IO 多路复用模型，在 windows 使用的 icop 等等。

根据系统类型不同选择不同的事务处理模型，选择有“use [kqueue | rtsig |epool |dev/pool |select |pllo];” 我们使用的是 Centos6.4 的 linux，因此将 nginx 的事件处理模型调整为 epool 模型。

4.2 调整 nginx worker 单个进程允许的客户端最大连接数，这个值根据服务器性能和程序的内存来指定（一个进程启动使用的内存根据程序确定），这个参数是单个进程的最大链接数，实际最大链接数是 worker 进程数乘以这个数。

Max_client=worker_processes*worker_connections

4.3 配置 nginx worker 进程最大打开文件数，相当于系统 ulimit –HSn，注意配置参数不是越大越好，最好设为服务器承受的极限点，最好与 ulimit-n 的值保持一致

events {

use epoll;

worker_connections 1024;

worker_rlimit_nofile65535;

}

5.设置连接超时时间

http {

.....

keepalive_timeout 60;

###设置客户端连接保持会话的超时时间，超过这个时间，服务器会关闭该连接。

tcp_nodelay on;

####打开 tcp_nodelay，在包含了 keepalive 参数才有效

client_header_timeout 15;

####设置客户端请求头读取超时时间，如果超过这个时间，客户端还没有发送任何数据，Nginx 将返回“Request time out（408）”错误

client_body_timeout 15;

####设置客户端请求主体读取超时时间，如果超过这个时间，客户端还没有发送任何数据，Nginx 将返回“Request time out（408）”错误

send_timeout 15;

####指定响应客户端的超时时间。这个超过仅限于两个连接活动之间的时间，如果超过这个时间，客户端没有任何活动，Nginx 将会关闭连接。

.....

}

6.配置 gzip 压缩功能 Gzip 压缩是可以节省带宽，提高传输效率，但是由于是在服务器上进行压缩，会消耗服务器起源

使用的是 nginx_http_gzip_module 模块

参数：

gzip on;

###开启压缩功能

gzip_min_length 1k;

###设置允许压缩的页面最小字节数，页面字节数从 header 头的 Content-Length 中获取，默认值是 0，不管页面多大都进行压缩，建议设置成大于 1K，如果小与 1K 可能会越压越大。

gzip_buffers 4 32k;

###压缩缓冲区大小，表示申请 4 个单位为 32K 的内存作为压缩结果流缓存，默认值是申请与原始数据大小相同的内存空间来存储 gzip 压缩结果。

gzip_http_version 1.1;

###压缩版本（默认 1.1，前端为 squid2.5 时使用 1.0）用于设置识别 HTTP 协议版本，默

认是 1.1，目前大部分浏览器已经支持 GZIP 解压，使用默认即可

gzip_comp_level 9;

###压缩比例，用来指定 GZIP 压缩比，1 压缩比最小，处理速度最快，9 压缩比最大，传输速度快，但是处理慢，也比较消耗 CPU 资源。

gzip_types text/css text/xml application/javascript;

###用来指定压缩的类型，‘text/html’类型总是会被压缩。

gzip_vary on;

###vary header 支持，改选项可以让前端的缓存服务器缓存经过 GZIP 压缩的页面，例如用 Squid 缓存经过 nginx 压缩的数据。

具体配置：

没配置 gzip 压缩之前;

[root@transit_machine ~]# curl -I -H"www.iyunv.com" "http://172.16.254.5/index.php"

HTTP/1.1 200 OK

Server: nginx

Date: Wed, 24 Jun 2015 04:51:53 GMT

Content-Type: text/html

Connection: keep-alive

X-Powered-By: PHP/5.4.30

在 http 模块中添加下面参数

gzip on;

gzip_min_length 1k;

gzip_buffers 4 32k;

gzip_http_version 1.1;

gzip_comp_level 9;

gzip_types text/css text/xml application/javascript;

gzip_vary on;

重新 curl 一下，出现 vary 表示压缩已经开启

[root@transit_machine ~]# curl -I -H"www.iyunv.com" "http://172.16.254.5/index.php"

HTTP/1.1 200 OK

Server: nginx

Date: Wed, 24 Jun 2015 04:53:46 GMT

Content-Type: text/html

Connection: keep-alive

Vary:Accept-Encoding

X-Powered-By: PHP/5.4.30

7.配置 nginx expires 缓存 expire 功能优点

（1）expires 可以降低网站购买的贷款，节约成本

（2）同时提升用户访问体验

（3）减轻服务的压力，节约服务器成本，甚至可以节约人力成本，是 web 服务非常重要的功能。

expire 功能缺点：

被缓存的页面或数据更新了，用户看到的可能还是旧的内容，反而影响用户体验。

解决办法：

第一个 缩短缓存时间，例如：1 天，不彻底，除非更新频率大于 1 天

第二个 对缓存的对象改名

a.图片，附件一般不会被用户修改，如果用户修改了，实际上也是更改文件名重新传了而已

b.网站升级对于 js，css 元素，一般可以改名，把 css，js，推送到 CDN。

配置方法:

配置前 curl 一下

[root@transit_machine ~]# curl -I -H"www.iyunv.com"

"http://172.16.254.5/index.php"

HTTP/1.1 200 OK

Server: nginx

Date: Wed, 24 Jun 2015 04:53:46 GMT

Content-Type: text/html

Connection: keep-alive

Vary: Accept-Encoding

X-Powered-By: PHP/5.4.30

现在设置缓存图片 1 个月，css、javascript、flush 等缓存一天

```
location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$ {
expires 30d;
}
location ~.*\.(js|css|javascript|fluash)$ {
expires 24h;
```

}

重新 curl 一下，发现 expires 缓存到了下个月今天

[root@transit_machine ~]# curl -I -H"www.iyunv.com" "http://172.16.254.5/du.jpg"

HTTP/1.1 200 OK

Server: nginx/1.6.3

Date: Wed, 24 Jun 2015 05:20:34 GMT

Content-Type: image/jpeg

Content-Length: 27696

Last-Modified: Mon, 07 Jul 201404:26:08 GMT

Connection: keep-alive

ETag: "53ba2160-6c30"

Expires:Fri, 24 Jul 2015 05:20:34 GMT

Cache-Control: max-age=2592000

Accept-Ranges: bytes

8.nginx 错误页面友好显示当访问网站出现 400 403 404 405 408 410，500 501 502 503 504 等状态码时，可以将其重定向到一个友好页面，提升用户体验。

配置方法：

1）在 http 模块中开启：fastcgi_intercept_errors on; （注意必须开启，否则不生效）

2）在 http 或者 server 模块配置下面参数

error_page 500 501 502 503 504 /error/5-error.html;

error_page 400 403 404 405 408 410 411 412413 414 415 /error/4-error.html;

注意：这里是相对路径，‘/’表示网站根目录，如果需要自定义目录的话，需要在下面添加

location 指定该目录位置

3）确保在网站根目录下存在 error 和相关网页

效果：

```
[root@transit_machine ~]#curl   -H "www.iyunv.com""http://172.16.254.5/aaa.php"
<!DOCTYPE html>
<html>
<head>
<title>Error</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial,sans-serif;
    }
</style>
</head>
<body>
<h1>40x error</h1>
<h1>40x error</h1>
<h1>40x error</h1>
<h1>40x error</h1>
<h1>40x error</h1>
```

<h1>40x error</h1>

<h1>40x error</h1>

<h1>40x error</h1>

<h1>40x error</h1>

<h1>40x error</h1>

<h1>40x error</h1>

</body>

</html>

注意，也可以将错误重定向到其他的 url 连接

下面是天猫的 nginx 优雅显示案例

error_page 500 501 502 503 504 http://err.tmall.com/error2.html;

error_page 400 403 404 405 408 410 411 412413 414 415 http://err.tmall.com/error1.html;

9.nginx 设置防掉链防盗链就是其他网站通过 url 引用你网站上的资源，达到填充本网站的显示效果，这样会增加自己的带宽，增加服务器的压力。

解决办法：

```
location ~* \.(gif|jpg|png|swf|flv)${
valid_referers none blocked www.iyunv.comwww.founder.com;
if ($invalid_referer) {
rewrite ^/ http://www.iyunv.com/403.html;
#return 404;
}
```

}

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：www.iyunv.com www.founder.com

表示对 www.iyunv.com www.founder.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就跳转到错误页面，当然直接返回 404 也是可以的。

10.nginx 限制访问

10.1 根据扩展名限制程序和目录访问，可以防止通过访问网页，而执行网页上的脚本

Nginx 限制访问指定目录：

location ~ ^/images/.*\.(php|php5|.sh|.pl|.py)$

{

Allow 127.0.0.1;

deny all;

}

注意需要放在 php 解析 location 的下面

Nginx 禁止访问*.txt,*.doc

location ~* \.(txt|doc)$ {

if (-f $request_filename) {

root /data/www/www;

#rewrite .....可以重定向到某个 URL

break;

}

}

10.2 限制 ip 访问

location / {

deny 192.168.1.1;

allow 192.168.1.0/24;

allow 10.1.1.0/16;

deny all;

}

或者

if ($remote_addr = 10.0.0.7) {

return 403;

}

10.3 限制客户端类型：可有效防止爬虫

下面内容可以防止爬虫

if ($http_user_agent ~*
"qihoobot|Baiduspider|Googlebot|Googlebot-Mobile|Googlebot-Image|Mediapartners-Google|Adsbot-Google|Yahoo! Slurp China|YoudaoBot|Sosospider|Sogou spider|Sogou web spider|MSNBot")

{

return 403;

}

禁止某类浏览器访问

if ($http_user_agent ~* "Firefox|MSIE")

{

return 403;

rewrite ^(.*) http://blog.etiantian.org/$1 permanent;

}

11.Nginx 修改源码隐藏软件名称及版本号这个需要在编译安装之前进行修改，修改的是源码包的内容

cd nginx-1.6.2/src/core

sed -n '13,17p'nginx.h

#define NGINX_VERSION "1.6.2" 修改为想要的版本号如 2.4.3

#define NGINX_VER "nginx/" NGINX_VERSION 将 nginx 修改为想要修改的软件名称，如 Apache。

9.nginx 内置的变量

$arg_name

请求行中的 name 参数。

$args

请求行中参数字符串。

$binary_remote_addr

客户端 IP 地址的二进制形式，值的长度总是 4 字节。

$body_bytes_sent

nginx 返回给客户端的字节数，不含响应头。

$bytes_sent

nginx 返回给客户端的字节数(1.3.8, 1.2.5)。

$connection

连接的序列号(1.3.8, 1.2.5)。

$content_length

“Content-Length”请求头的值。

$content_type

“Content-Type”请求头的值。

$cookie_name

名为 name 的 cookie。

$document_root

当前请求的 root 指令或 alias 指令的配置值。

$document_uri

与$uri 相同。

$host

“Host”请求头的值，如果没有该请求头，则为与请求对应的虚拟主机的首要主机名。

$hostname

机器名称。

$http_name

任意请求头的值 ;变量名的后半部为转化为小写并且用下划线替代横线后的请求头名称。

$https

如果连接是 SSL 模块，返回“on”，否则返回空字符串。

$is_args

如果请求行带有参数，返回“?”，否则返回空字符串。

$limit_rate

允许设置此值来限制连接的传输速率。

$msec

当前时间，单位是秒，精度是毫秒。(1.3.9, 1.2.6)。

$nginx_version

nginx 版本号。

$pid

worker 进程的 PID。

$query_string

与$args 相同。

$realpath_root

按 root 指令或 alias 指令算出的当前请求的绝对路径。其中的符号链接都会解析成真是文件路径。

$remote_addr

客户端 IP 地址。

$remote_port

客户端端口。

$remote_user

为基本用户认证提供的用户名。

$request

完整的原始请求行。

$request_body

请求正文。

在由 proxy_pass 指令和 fastcgi_pass 指令处理的路径中， 这个变量值可用。

$request_body_file

请求正文的临时文件名。

处理完成时，临时文件将被删除。 如果希望总是将请求正文写入文件，需要开启 client_body_in_file_only。 如果在被代理的请求或 FastCGI 请求中传递临时文件名，就应该禁止传递请求正文本身。 使用 proxy_pass_request_body off 指令 和 fastcgi_pass_request_body off 指令 分别禁止在代理和 FastCGI 中传递请求正文。

$request_completion

请求完成时返回“OK”，否则返回空字符串。

$request_filename

基于 root 指令或 alias 指令，以及请求 URI，得到的当前请求的文件路径。

$request_method

HTTP 方法，通常为“GET”或者“POST”。

$request_time

请求处理的时间，单位为秒，精度是毫秒(1.3.9, 1.2.6)；请求处理时间从由客户端接收到第一个字节开始计算。

$request_uri

完整的原始请求行（带参数）。

$scheme

请求协议类型，为“http”或“https”。

$sent_http_name

任意的响应头字段的值。 变量名的后半部为转化为小写并且用下划线替代横线后的响应头名称。

$server_addr

接受请求的服务器地址。

为计算这个值，通常需要进行一次系统调用。为了避免系统调用，必须指定 listen 指令的地址，并且使用 bind 参数。

$server_name

接受请求的虚拟主机的首要主机名。

$server_port

接受请求的虚拟主机的端口。

$server_protocol

请求协议，通常为“HTTP/1.0”或“HTTP/1.1”。

$status

响应状态码。

$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space

客户端 TCP 连接的信息，在支持套接字选项 TCP_INFO 的系统中可用。

$uri

当前请求规范化以后的 URI。

变量$uri 的值可能随请求的处理过程而改变。 比如，当进行内部跳转时，或者使用默认页文件。

10.Rewrite 规则示例参考

Nginx Rewrite 规则相关指令

Nginx Rewrite 规则相关指令有 if、rewrite、set、return、break 等，其中 rewrite 是最关键的指令。一个简单的 Nginx Rewrite 规则语法如下：

```
rewrite ^/b/(.*)\.html /play.php?video=$1 break;
```

如果加上 if 语句，示例如下：

```
if (!-f $request_filename)
rewrite ^/img/(.*)$ /site/$host/images/$1 last;
```

Nginx 与 Apache 的 Rewrite 规则实例对比

简单的 Nginx 和 Apache 重写规则区别不大，基本上能够完全兼容。例如：

Apache Rewrite 规则：

```
RewriteRule ^/(mianshi|xianjing)/$ /zl/index.php?name=$1 [L]
RewriteRule ^/ceshi/$ /zl/ceshi.php [L]
RewriteRule ^/(mianshi)_([a-zA-Z]+)/$ /zl/index.php?name=$1_$2 [L]
RewriteRule ^/pingce([0-9]*)/$ /zl/pingce.php?id=$1 [L]
```

Nginx Rewrite 规则：

```
rewrite ^/(mianshi|xianjing)/$ /zl/index.php?name=$1 last;
rewrite ^/ceshi/$ /zl/ceshi.php last;
```

rewrite ^/(mianshi)_([a-zA-Z]+)/$ /zl/index.php?name=$1_$2 last;

rewrite ^/pingce([0-9]*)/$ /zl/pingce.php?id=$1 last;

由以上示例可以看出，Apache 的 Rewrite 规则改为 Nginx 的 Rewrite 规则，其实很简单：Apache 的 RewriteRule 指令换成 Nginx 的 rewrite 指令，Apache 的[L]标记换成 Nginx 的 last 标记，中间的内容不变。

如果 Apache 的 Rewrite 规则改为 Nginx 的 Rewrite 规则后，使用 nginx -t 命令检查发现 nginx.conf 配置文件有语法错误，那么可以尝试给条件加上引号。例如一下的 Nginx Rewrite 规则会报语法错误：

rewrite ^/([0-9]{5}).html$ /x.jsp?id=$1 last;

加上引号就正确了：

rewrite “^/([0-9]{5}).html$” /x.jsp?id=$1 last;

Apache 与 Nginx 的 Rewrite 规则在 URL 跳转时有细微的区别：

Apache Rewrite 规则：

RewriteRule ^/html/tagindex/([a-zA-Z]+)/.*$ /$1/ [R=301,L]

Nginx Rewrite 规则：

rewrite ^/html/tagindex/([a-zA-Z]+)/.*$ http://$host/$1/ permanent;

以上示例中，我们注意到，Nginx Rewrite 规则的置换串中增加了“http://$host”，这是在 Nginx 中要求的。

另外，Apache 与 Nginx 的 Rewrite 规则在变量名称方面也有区别，例如：

Apache Rewrite 规则：

RewriteRule

^/user/login/$ /user/login.php?login=1&forward=http://%{HTTP_HOST} [L]

Nginx Rewrite 规则：

rewrite ^/user/login/$ /user/login.php?login=1&forward=http://$host last;

Apache 与 Nginx Rewrite 规则的一些功能相同或类似的指令、标记对应关系：

Apache 的 RewriteCond 指令对应 Nginx 的 if 指令；

Apache 的 RewriteRule 指令对应 Nginx 的 rewrite 指令；

Apache 的[R]标记对应 Nginx 的 redirect 标记；

Apache 的[P]标记对应 Nginx 的 last 标记；

Apache 的[R,L]标记对应 Nginx 的 redirect 标记；

Apache 的[P,L]标记对应 Nginx 的 last 标记；

Apache 的[PT,L]标记对应 Nginx 的 last 标记；

允许指定的域名访问本站，其他域名一律跳转到 http://www.aaa.com

Apache Rewrite 规则：

RewriteCond %{HTTP_HOST} ^(.*?)\.domain\.com$

RewriteCond %{HTTP_HOST} !^qita\.domain\.com$

RewriteCond %{DOCUMENT_ROOT}/market/%1/index.htm -f

RewriteRule ^/wu/$ /market/%1/index.htm [L]

Nginx 的 if 指令不支持嵌套，也不支持 AND、OR 等多条件匹配，相比于 Apache 的 RewriteCond，显得麻烦一些，但是，我们可以通过下一页的 Nginx 配置写法来实现这个示例：

Nginx Rewrite 规则：

if ($host ~* ^(.*?)\.domain\.com$) set $var_wupin_city $1;

set $var_wupin ‘1’;

```
if ($host ~* ^qita\.domain\.com$)
      set $var_wupin    '0';
 if (!-f $document_root/market/$var_wupin_city/index.htm)
       set $var_wupin    '0';
if ($var_wupin ~    '1')
      rewrite ^/wu/$ /market/$var_wupin_city/index.htm last;
}
```

rewrite 的语法

语法: rewrite regex replacement flag

默认: none

作用域: server, location, if

This directive changes URI in accordance with the regular expression and the replacement string. Directives are carried out in order of appearance in the configuration file.

这个指令根据表达式来更改 URI，或者修改字符串。指令根据配置文件中的顺序来执行。

Be aware that the rewrite regex only matches the relative path instead of the absolute URL. If you want to match the hostname, you should use an if condition, like so:

注意重写表达式只对相对路径有效。如果你想配对主机名，你应该使用 if 语句。

rewrite 只是会改写路径部分的东东，不会改动用户的输入参数，因此这里的 if 规则里面，你无需关心用户在浏览器里输入的参数，rewrite 后会自动添加的，因此，我们只是加上了一个？号和后面我们想要的一个小小的参数 ***https=1 就可以了。

nginx 的 rewrite 规则参考：

~ 为区分大小写匹配

~* 为不区分大小写匹配

!~和!~*分别为区分大小写不匹配及不区分大小写不匹

-f 和!-f 用来判断是否存在文件

-d 和!-d 用来判断是否存在目录

-e 和!-e 用来判断是否存在文件或目录

-x 和!-x 用来判断文件是否可执行

last 相当于 Apache 里的[L]标记，表示完成 rewrite，呵呵这应该是最常用的

break 终止匹配, 不再匹配后面的规则

redirect 返回 302 临时重定向 地址栏会显示跳转后的地址

permanent 返回 301 永久重定向 地址栏会显示跳转后的地址

$args

$content_length

$content_type

$document_root

$document_uri

$host

$http_user_agent

$http_cookie

$limit_rate

$request_body_file

$request_method

$remote_addr

$remote_port

$remote_user

$request_filename

$request_uri

$query_string

$scheme

$server_protocol

$server_addr

$server_name

$server_port

$uri

结合 QeePHP 的例子

if (!-d $request_filename) {

rewrite ^/([a-z-A-Z]+)/([a-z-A-Z]+)/?(.*)$ /index.php?namespace=user&controller=$1&action=$2&$3 last;

rewrite ^/([a-z-A-Z]+)/?$ /index.php?namespace=user&controller=$1 last;

break;

多目录转成参数

abc.domian.com/sort/2 =>

abc.domian.com/index.php?act=sort&name=abc&id=2

```
if ($host ~* (.*)\.domain\.com) {
set $sub_name $1;
rewrite ^/sort\/(\d+)\/?$ /index.php?act=sort&cid=$sub_name&id=$1 last;
}
```

目录对换

/123456/xxxx -> /xxxx?id=123456

```
rewrite ^/(\d+)/(.+)/ /$2?id=$1 last;
```

例如下面设定 nginx 在用户使用 ie 的使用重定向到/nginx-ie 目录下：

```
if ($http_user_agent ~ MSIE) {
rewrite ^(.*)$ /nginx-ie/$1 break;
}
```

目录自动加“/”

```
if (-d $request_filename){
rewrite ^/(.*)([^/])$ http://$host/$1$2/ permanent;
}
```

禁止 htaccess

```
location ~/\.ht {
deny all;
}
```

禁止多个目录

```
location ~ ^/(cron|templates)/ {
deny all;
break;
}
```

禁止以/data 开头的文件

可以禁止/data/下多级目录下.log.txt 等请求;

```
location ~ ^/data {
deny all;
}
```

禁止单个目录

不能禁止.log.txt 能请求

```
location /searchword/cron/ {
deny all;
}
```

禁止单个文件

```
location ~ /data/sql/data.sql {
deny all;
}
```

给 favicon.ico 和 robots.txt 设置过期时间;

这里为 favicon.ico 为 99 天,robots.txt 为 7 天并不记录 404 错误日志

```
location ~(favicon.ico) {
```

```
log_not_found off;
expires 99d;
break;
}
location ~(robots.txt) {
log_not_found off;
expires 7d;
break;
}
```

设定某个文件的过期时间;这里为 600 秒，并不记录访问日志

```
location ^~ /html/scripts/loadhead_1.js {
access_log     off;
root /opt/lampp/htdocs/web;
expires 600;
break;
}
```

文件反盗链并设置过期时间

这里的 return 412 为自定义的 http 状态码，默认为 403，方便找出正确的盗链的请求

“rewrite ^/ http://leech.divmy.com/leech.gif;”显示一张防盗链图片

“access_log off;”不记录访问日志，减轻压力

“expires 3d”所有文件 3 天的浏览器缓存

```
location ~* ^.+\.(jpg|jpeg|gif|png|swf|rar|zip|css|js)$ {
```

```
valid_referers none blocked *.c1gstudio.com *.c1gstudio.net localhost
208.97.167.194;
if ($invalid_referer) {
rewrite ^/ http://leech.divmy.com/leech.gif;
return 412;
break;
}
access_log off;
root /opt/lampp/htdocs/web;
expires 3d;
break;
}
```

只充许固定 ip 访问网站，并加上密码

```
root /opt/htdocs/www;
allow 208.97.167.194;
allow 222.33.1.2;
allow 231.152.49.4;
deny all;
auth_basic “C1G_ADMIN”;
auth_basic_user_file htpasswd;
```

将多级目录下的文件转成一个文件，增强 seo 效果

/job-123-456-789.html 指向/job/123/456/789.html

rewrite ^/job-([0-9]+)-([0-9]+)-([0-9]+)\.html$ /job/$1/$2/jobshow_$3.html last;

将根目录下某个文件夹指向 2 级目录

如/shanghaijob/ 指向 /area/shanghai/

如果你将 last 改成 permanent，那么浏览器地址栏显是/location/shanghai/

rewrite ^/([0-9a-z]+)job/(.*)$ /area/$1/$2 last;

上面例子有个问题是访问/shanghai 时将不会匹配

rewrite ^/([0-9a-z]+)job$ /area/$1/ last;

rewrite ^/([0-9a-z]+)job/(.*)$ /area/$1/$2 last;

这样/shanghai 也可以访问了，但页面中的相对链接无法使用，

如./list_1.html 真实地址是/area/shanghia/list_1.html 会变成/list_1.html,导至无法访问。

那我加上自动跳转也是不行咯

(-d $request_filename)它有个条件是必需为真实目录，而我的 rewrite 不是的，所以没有效果

if (-d $request_filename){

rewrite ^/(.*)([^/])$ http://$host/$1$2/ permanent;

}

知道原因后就好办了，让我手动跳转吧

rewrite ^/([0-9a-z]+)job$ /$1job/ permanent;

rewrite ^/([0-9a-z]+)job/(.*)$ /area/$1/$2 last;

文件和目录不存在的时候重定向：

if (!-e $request_filename) {

```
proxy_pass http://127.0.0.1;
}
```

域名跳转

```
server
{
listen          80;
server_name   jump.88dgw.com;
index index.html index.htm index.php;
root   /opt/lampp/htdocs/www;
rewrite ^/ http://www.88dgw.com/;
access_log   off;
}
```

多域名转向

```
server_name   www.7oom.com/   www.divmy.com/;
index index.html index.htm index.php;
root   /opt/lampp/htdocs;
if ($host ~  “c1gstudio\.net”) {
rewrite ^(.*) http://www.7oom.com$1/ permanent;
}
```

三级域名跳转

```
if ($http_host ~*  “^(.*)\.i\.c1gstudio\.com$”) {
rewrite ^(.*) http://top.88dgw.com$1/;
```

```
    break;
    }
```

域名镜向

```
    server
    {
    listen          80;
    server_name    mirror.c1gstudio.com;
    index index.html index.htm index.php;
    root    /opt/lampp/htdocs/www;
    rewrite ^/(.*) http://www.divmy.com/$1 last;
    access_log    off;
    }
```

某个子目录作镜向

```
    location ^~ /zhaopinhui {
    rewrite ^.+ http://zph.divmy.com/ last;
    break;
    }
```

discuz ucenter home (uchome) rewrite

```
    rewrite ^/(space|network)-(.+)\.html$ /$1.php?rewrite=$2 last;
    rewrite ^/(space|network)\.html$ /$1.php last;
    rewrite ^/([0-9]+)$ /space.php?uid=$1 last;
```

discuz 7 rewrite

```
    rewrite ^(.*)/archiver/((fid|tid)-[\w\-]+\.html)$ $1/archiver/index.php?$2 last;
    rewrite
^(.*)/forum-([0-9]+)-([0-9]+)\.html$ $1/forumdisplay.php?fid=$2&page=$3 last;
    rewrite
^(.*)/thread-([0-9]+)-([0-9]+)-([0-9]+)\.html$  $1/viewthread.php?tid=$2&extra=p
age\%3D$4&page=$3 last;
    rewrite ^(.*)/profile-(username|uid)-(.+)\.html$ $1/viewpro.php?$2=$3 last;
    rewrite ^(.*)/space-(username|uid)-(.+)\.html$ $1/space.php?$2=$3 last;
    rewrite ^(.*)/tag-(.+)\.html$ $1/tag.php?name=$2 last;
```

给 discuz 某版块单独配置域名

```
    server_name   bbs.c1gstudio.com news.c1gstudio.com;
    location = / {
    if ($http_host ~ news\.divmy.com$) {
    rewrite ^.+ http://news.divmy.com/forum-831-1.html last;
    break;
    }
    }
```

discuz ucenter 头像 rewrite 优化

```
    location ^~ /ucenter {
    location ~ .*\.php?$
    {
    #fastcgi_pass   unix:/tmp/php-cgi.sock;
```

```
fastcgi_pass   127.0.0.1:9000;
fastcgi_index index.php;
include fcgi.conf;
}
location /ucenter/data/avatar {
log_not_found off;
access_log     off;
location ~ /(.*)_big\.jpg$ {
error_page 404 /ucenter/images/noavatar_big.gif;
}
location ~ /(.*)_middle\.jpg$ {
error_page 404 /ucenter/images/noavatar_middle.gif;
}
location ~ /(.*)_small\.jpg$ {
error_page 404 /ucenter/images/noavatar_small.gif;
}
expires 300;
break;
}
}
```

jspace rewrite

```
location ~ .*\.php?$
```

```
{
#fastcgi_pass   unix:/tmp/php-cgi.sock;
fastcgi_pass   127.0.0.1:9000;
fastcgi_index index.php;
include fcgi.conf;
}
location ~* ^/index.php/
{
rewrite ^/index.php/(.*) /index.php?$1 break;
fastcgi_pass   127.0.0.1:9000;
fastcgi_index index.php;
include fcgi.conf;
}
```