

第 3 章

Nginx 的基本配置与优化

3.1 Nginx 的完整配置示例

Nginx 的配置文件默认在 Nginx 程序安装目录的 conf 二级目录下，主配置文件为 nginx.conf，假设您的 Nginx 安装在 /usr/local/webserver/nginx/ 目录下，那么默认的主配置文件则为 /usr/local/webserver/nginx/nginx.conf，代码 3-1 是 Nginx 作为 Web Server 的完整配置示例。

代码 3-1

```
# 使用的用户和组
user www www;
# 指定工作进程数（一般等于 CPU 的总核数或总核数的两倍，例如两个四核 CPU，则总核数为 8）
worker_processes 8;
# 指定错误日志存放的路径，错误日志记录级别可选项为：[ debug | info | notice | warn | error
| crit ]
error_log /data1/logs/nginx_error.log crit;
# 指定 pid 存放的路径
pid /usr/local/webserver/nginx/nginx.pid;

# 指定文件描述符数量
worker_rlimit_nofile 51200;

events
{
    # 使用的网络 I/O 模型，Linux 系统推荐采用 epoll 模型，FreeBSD 系统推荐采用 kqueue 模型
    use epoll;
```

```
# 允许的连接数
worker_connections 51200;
}

http
{
    include      mime.types;
    default_type application/octet-stream;
    # 设置使用的字符集, 如果一个网站有多种字符集, 请不要随便设置, 应让程序员在 HTML 代码中通过 Meta
    标签设置
    #charset gb2312;

    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;

    # 设置客户端能够上传的文件大小
    client_max_body_size 8m;

    sendfile on;
    tcp_nopush on;

    keepalive_timeout 60;

    tcp_nodelay on;

    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;
    # 开启 gzip 压缩
    gzip on;
    gzip_min_length 1k;
    gzip_buffers 4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 2;
    gzip_types      text/plain application/x-javascript text/css application/xml;
    gzip_vary on;

    #limit_zone crawler $binary_remote_addr 10m;

    server
    {
        listen      80;
        server_name www.yourdomain.com yourdomain.com;
        index index.html index.htm index.php;
    }
}
```

```

root /data0/htdocs;

#limit_conn crawler 20;

location ~ .*\. (gif|jpg|jpeg|png|bmp|swf)$
{
    expires      30d;
}

location ~ .*\. (js|css)?$
{
    expires      1h;
}

log_format access '$remote_addr - $remote_user [$time_local] "$request" '
    '$status $body_bytes_sent "$http_referer" '
    '"$http_user_agent" $http_x_forwarded_for';
access_log /data1/logs/access.log access;
}
}

```

通过上面的 nginx.conf 配置文件示例，可以看出，nginx.conf 的配置文件结构主要由以下几部分构成（见代码 3-2）。

代码 3-2

```

.....
events
{
    .....
}

http
{
    .....
    server
    {
        .....
    }

    server
    {
        .....
    }
    .....
}

```

下一节笔者会详细介绍 Nginx 虚拟主机的配置。

3.2 Nginx 的虚拟主机配置

3.2.1 什么是虚拟主机

虚拟主机使用的是特殊的软硬件技术，它把一台运行在因特网上的服务器主机分成一台台“虚拟”的主机，每台虚拟主机都可以是一个独立的网站，可以具有独立的域名，具有完整的 Internet 服务器功能（WWW、FTP、Email 等），同一台主机上的虚拟主机之间是完全独立的。从网站访问者来看，每一台虚拟主机和一台独立的主机完全一样。

利用虚拟主机，不用为每个要运行的网站提供一台单独的 Nginx 服务器或单独运行一组 Nginx 进程。虚拟主机提供了在同一台服务器、同一组 Nginx 进程上运行多个网站的功能。

在 Nginx 配置文件（nginx.conf）中，一个最简化的虚拟主机配置如代码 3-3 所示。

代码 3-3

```
http
{
    server
    {
        listen      80 default;
        server_name _ *;
        access_log  logs/default.access.log combined;
        location / {
            index index.html;
            root /data0/htdocs/htdocs;
        }
    }
}
```

跟 Apache 一样，Nginx 也可以配置多种类型的虚拟主机：一是基于 IP 的虚拟主机，二是基于域名的虚拟主机，三是基于端口的虚拟主机。

3.2.2 配置基于 IP 的虚拟主机

Linux、FreeBSD 操作系统都允许添加 IP 别名。IP 别名背后的概念很简单：可以在一块物理网卡上绑定多个 IP 地址。这样就能够在使用单一网卡的同一个服务器上运行多个基于 IP 的虚拟主机。设置 IP 别名也非常容易，只须配置系统上的网络接口，让它监听额外的 IP 地址。在 Linux 系统上，可以使用标准的网络配置工具（比如 ifconfig 和 route 命令）添加 IP 别名。以下是添加 IP 别名的示例：

先用 `ifconfig` 命令查看该服务器的 IP 地址。下面这台服务器有一块物理网卡设备 `eth0` 和本地回环设备 `lo`, `eth0` 的 IP 地址为 192.168.8.42, 本地回环 `lo` 的 IP 地址为 127.0.0.1。

本地回环代表设备的本地虚拟接口, 所以默认被看作是永远不会宕掉的接口。它的主要作用有两个: 一是测试本机的网络配置, 能 PING 通 127.0.0.1 说明本机的网卡和 IP 协议安装都没有问题; 另一个作用是某些 `SERVER/CLIENT` 的应用程序在运行时须调用服务器上的资源, 一般要指定 `SERVER` 的 IP 地址, 但当该程序要在同一台机器上运行且没有别的 `SERVER` 时, 就可以把 `SERVER` 的资源装在本机上, `SERVER` 的 IP 地址设为 127.0.0.1 也同样可以运行, 如代码 3-4 所示。

代码 3-4

```
[root@localhost ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:48:34:FF:96
          inet addr:192.168.8.42  Bcast:192.168.8.255  Mask:255.255.255.0
          inet6 addr: fe80::230:48ff:fe34:ff96/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8397714 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2435863 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:797326757 (760.3 MiB)  TX bytes:229377030 (218.7 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1286 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1286 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:150843 (147.3 KiB)  TX bytes:150843 (147.3 KiB)
```

如果要在 `eth0` 网卡设备上添加两个 IP 别名 192.168.8.43 和 192.168.8.44, 可以通过以下的 `ifconfig` 和 `route` 命令来进行:

```
/sbin/ifconfig eth0:1 192.168.8.43 broadcast 192.168.8.255 netmask 255.255.255.0 up
/sbin/route add -host 192.168.8.43 dev eth0:1

/sbin/ifconfig eth0:2 192.168.8.44 broadcast 192.168.8.255 netmask 255.255.255.0 up
/sbin/route add -host 192.168.8.44 dev eth0:2
```

这时, 再执行 `ifconfig` 命令, 就可以看到 `eth0` 网卡设备上绑定了两个 IP 别名, 如代码 3-5 所示。

代码 3-5

```
[root@xoyo42 ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:48:34:FF:96
          inet addr:192.168.8.42  Bcast:192.168.8.255  Mask:255.255.255.0
          inet6 addr: fe80::230:48ff:fe34:ff96/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8407611  errors:0  dropped:0  overruns:0  frame:0
          TX packets:2437149  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:798160555 (761.1 MiB)  TX bytes:229521778 (218.8 MiB)

eth0:1    Link encap:Ethernet  HWaddr 00:30:48:34:FF:96
          inet addr:192.168.8.43  Bcast:192.168.8.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:2    Link encap:Ethernet  HWaddr 00:30:48:34:FF:96
          inet addr:192.168.8.44  Bcast:192.168.8.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1290  errors:0  dropped:0  overruns:0  frame:0
          TX packets:1290  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:151235 (147.6 KiB)  TX bytes:151235 (147.6 KiB)
```

这时候, 从另外一台服务器 Ping 192.168.8.43 和 192.168.8.44 两个 IP, 如果能够 Ping 通, 则证明配置无误。但是, 通过 `ifconfig` 和 `route` 配置的 IP 别名在服务器重启后会消失, 不过可以将这两条 `ifconfig` 和 `route` 命令添加到 `/etc/rc.local` 文件中, 让系统开机时自动运行, 以下是相关命令:

```
vi /etc/rc.local
```

在文件末尾增加以下内容, 然后保存即可:

```
/sbin/ifconfig eth0:1 192.168.8.43 broadcast 192.168.8.255 netmask 255.255.255.0 up
/sbin/route add -host 192.168.8.43 dev eth0:1
/sbin/ifconfig eth0:2 192.168.8.44 broadcast 192.168.8.255 netmask 255.255.255.0 up
/sbin/route add -host 192.168.8.44 dev eth0:2
```

下面开始配置基于 IP 的虚拟主机。无论是通过 IP 别名在一台服务器上配置多个 IP 地址, 还是通过多块网卡在服务器上配置多个 IP 地址, 在 Nginx 中都能将其配置成为基于 IP 的虚拟主机。

接下来在 Nginx 配置文件 (`nginx.conf`) 中, 分别对 192.168.8.43、192.168.8.44、192.168.8.45 三个 IP 配置三个纯静态 HTML 支持的虚拟主机, 如代码 3-6 所示。

代码 3-6

```
http
{
    # 第一个虚拟主机
    server
    {
        # 监听的 IP 和端口
        listen      192.168.8.43:80;
        # 主机名称
        server_name 192.168.8.43;
        # 访问日志文件存放路径
        access_log   logs/server1.access.log combined;
        location /
        {
            # 默认首页文件, 顺序从左到右, 如果找不到 index.html 文件, 则查找 index.htm 文件作为
            # 首页文件
            index index.html index.htm;
            # HTML 网页文件存放的目录
            root /data0/htdocs/server1;
        }
    }
    # 第二个虚拟主机
    server
    {
        # 监听的 IP 和端口
        listen      192.168.8.44:80;
        # 主机名称
        server_name 192.168.8.44;
        # 访问日志文件存放路径
        access_log   logs/server2.access.log combined;
        location /
        {
            # 默认首页文件, 顺序从左到右, 如果找不到 index.html 文件, 则查找 index.htm 文件作为
            # 首页文件
            index index.html index.htm;
            # HTML 网页文件存放的目录
            root /data0/htdocs/server2;
        }
    }
    # 第三个虚拟主机
    server
    {
        # 监听的 IP 和端口
        listen      192.168.8.45:80;
        # 主机名称
        server_name 192.168.8.45;
        # 访问日志文件存放路径
        access_log   logs/server3.access.log combined;
        location /
        {
```

```
# 默认首页文件，顺序从左到右，如果找不到 index.html 文件，则查找 index.htm 文件作为
# 首页文件
index index.html index.htm;
# HTML 网页文件存放的目录
root /data0/htdocs/server3.com;
}
}
}
```

从上面的配置文件中可以看出，一段 `server{.....}` 就是一个虚拟主机，如果要配置多个虚拟主机，建立多段 `server{}` 配置即可，非常方便。监听的 IP 和端口也可以不写 IP 地址，只写端口，把它配置成“`listen 80`”，则表示监听该服务器上所有 IP 的 80 端口，可通过 `server_name` 区分不同的虚拟主机。

3.2.3 配置基于域名的虚拟主机

基于域名的虚拟主机是最常见的一种虚拟主机。只需配置你的 DNS 服务器，将每个主机名映射到正确的 IP 地址，然后配置 Nginx 服务器，令其识别不同的主机名就可以了。这种虚拟主机技术，使很多虚拟主机可以共享同一个 IP 地址，有效解决了 IP 地址不足的问题。所以，如果没有特殊要求使你必须用一个基于 IP 的虚拟主机，最好还是使用基于域名的虚拟主机。

接下来配置基于域名的虚拟主机。在以下的示例中，配置了三个虚拟主机，第一个虚拟主机表示所有对域名 `aaa.domain.com` 的访问都由它来处理，第二个虚拟主机表示所有对域名 `bbb.otherdomain.com` 的访问都由它来处理，第三个虚拟主机表示对域名 `www.domain.com`、`domain.com`，以及除了 `aaa.domain.com` 之外的所有 `*.domain.com` 二级域名的访问都由它来处理。每个虚拟主机的网页文件分别存放在不同的目录中，每个虚拟主机使用了不同的日志文件来记录访问日志，如代码 3-7 所示。

代码 3-7

```
http
{
    # 第一个虚拟主机
    server
    {
        # 监听的端口
        listen 80;
        # 主机名称
        server_name aaa.domain.com;
        # 访问日志文件存放路径
        access_log logs/aaa.domain.com.access.log combined;
        location /
        {
```



```
# 默认首页文件，顺序从左到右，如果找不到 index.html 文件，则查找 index.htm 文件作为
# 首页文件
index index.html index.htm;
# HTML 网页文件存放的目录
root /data0/htdocs/aaa.domain.com;
}
}
# 第二个虚拟主机
server
{
    # 监听的 IP 和端口
    listen 80;
    # 主机名称
    server_name bbb.otherdomain.com;
    # 访问日志文件存放路径
    access_log logs/bbb.otherdomain.com.access.log combined;
    location /
    {
        # 默认首页文件，顺序从左到右，如果找不到 index.html 文件，则查找 index.htm 文件作为
        # 首页文件
        index index.html index.htm;
        # HTML 网页文件存放的目录
        root /data0/htdocs/bbb.otherdomain.com;
    }
}
# 第三个虚拟主机
server
{
    # 监听的 IP 和端口
    listen 80;
    # 主机名称
    server_name www.domain.com domain.com *.domain.com;
    # 访问日志文件存放路径
    access_log logs/bbb.domain.com.access.log combined;
    location /
    {
        # 默认首页文件，顺序从左到右，如果找不到 index.html 文件，则查找 index.htm 文件作为
        # 首页文件
        index index.html index.htm;
        # HTML 网页文件存放的目录
        root /data0/htdocs/domain.com;
    }
}
```

3.3 Nginx 的日志文件配置与切割

在上一节的 Nginx 虚拟主机配置中，已经使用 `access_log` 进行了日志记录，这一节中，笔者将详细介绍 Nginx 访问日志文件的配置。

与 Nginx 日志相关的指令主要有两条，一条是 `log_format`，用来设置日志的格式，另外一条是 `access_log`，用来指定日志文件的存放路径、格式和缓存大小。两条指令在 Nginx 配置文件中的位置可以在 `http{.....}` 之间，也可以在虚拟主机之间，即 `server{.....}` 两个大括号之间。

3.3.1 用 `log_format` 指令设置日志格式

`log_format` 指令用来设置日志的记录格式，它的语法如下：

```
log_format name format [format ...]
```

其中 `name` 表示定义的格式名称，`format` 表示定义的格式样式。`log_format` 有一个默认的、无须设置的 `combined` 日志格式设置，相当于 Apache 的 `combined` 日志格式，其具体参数如下：

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

您也可以自定义一份日志的记录格式，不过要注意，`log_format` 指令设置的 `name` 名称在 Nginx 配置文件中是不能重复的。

假设将 Nginx 服务器作为 Web 服务器，位于负载均衡设备、Squid、Nginx 反向代理之后，就不能获取到客户端的真实 IP 地址了。原因是经过反向代理后，由于在客户端和 Web 服务器之间增加了中间层，因此 Web 服务器无法直接拿到客户端的 IP，通过 `$remote_addr` 变量拿到的将是反向代理服务器的 IP 地址。但是，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，用以记录原有的客户端 IP 地址和原来客户端请求的服务器地址。

这时候，就要用 `log_format` 指令来设置日志格式，让日志记录 X-Forwarded-For 信息中的 IP 地址，即客户的真实 IP。例如，创建一个名为 `mylogformat` 的日志格式，再用 `$http_x_forwarded_for` 变量记录用户的 X-Forwarded-For IP 地址：

```
log_format mylogformat '$http_x_forwarded_for - $remote_user [$time_local] '
                    '$request' $status $body_bytes_sent '
                    '$http_referer' '$http_user_agent';;
```

在日志格式样式中，变量 `$remote_addr` 和 `$http_x_forwarded_for` 用于记录 IP 地址；`$remote_user` 用于记录远程客户端用户名；`$time_local` 用于记录访问时间与时区；`$request` 用于记录请求 URL 与 HTTP 协议；`$status` 用于记录请求状态，例如成功时状态为 200，页面找到时状态为 404；`$body_bytes_sent` 用于记录发送给客户端的文件主体内容大小；`$http_referer` 用于记录是从哪个页面链接访问过来的；`$http_user_agent` 用于记录客户端浏览器的相关信息。

采用日志格式 `combined` 记录的日志格式如代码 3-8 所示。

代码 3-8

```
124.42.4.194 - - [12/Mar/2009:02:18:23 +0800] "GET / HTTP/1.1" 200 36179 "-"
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1) ; CIBA; .NET CLR 2.0.50727)"
124.42.4.194 - - [12/Mar/2009:02:18:24 +0800] "GET /attachment.php?fid=12 HTTP/1.1"
302 5 "http://blog.sl35.com/" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; CIBA; .NET CLR 2.0.50727)"
124.42.4.194 - - [12/Mar/2009:02:18:24 +0800] "GET /attachment.php?fid=2 HTTP/1.1"
302 5 "http://blog.sl35.com/" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) ; CIBA; .NET CLR 2.0.50727)"
```

3.3.2 用 access_log 指令指定日志文件存放路径

用 log_format 指令设置了日志格式之后，需要用 access_log 指令指定日志文件存放路径。
access_log 指令的语法如下：

```
access_log path [format [buffer=size | off]]
```

其中 path 表示日志文件的存放路径，format 表示使用 log_format 指令设置的日志格式的名称，
buffer=size 表示设置内存缓冲区的大小，例如可以设置 buffer=32k。

(1) 如果不想记录日志，可以使用以下指令关闭日志记录：

```
access_log off;
```

(2) 如果想使用默认的 combined 格式的日志记录，可以使用以下示例：

```
access_log /data1/logs/filename.log;
```

或者

```
access_log /data1/logs/filename.log combined;
```

(3) 如果想使用自定义格式的日志记录，可以使用以下示例，其中的 mylogformat 是日志格式名称：

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

```
access_log /data1/logs/access.log mylogformat buffer=32k;
```

(4) 在 Nginx 0.7.4 之后的版本中，access_log 指令中的日志文件路径可以包含变量，例如：

```
access_log /data1/logs/$server_name.log combined;
```

假设 server_name 指令设置的虚拟主机名称为 test.domain.com，那么 access_log 指令将把访

问日志记录在/data1/logs/test.domain.com.log 文件中。

如果日志文件路径中含有变量，将存在以下一些限制：

(1) Nginx 进程设置的用户和组必须有对该路径创建文件的权限。假设 Nginx 的 user 指令设置的用户名和用户组都是 www，而/data1/logs/目录的用户名和用户组为 root，日志文件/data1/logs/test.domain.com.log 将无法被 Nginx 创建；

(2) 缓冲将不会被使用；

(3) 对于每一条日志记录，日志文件都将先打开文件，再写入日志记录，然后马上关闭。为了提高包含变量的日志文件存放路径的性能，须要使用 open_log_file_cache 指令设置经常被使用的日志文件描述符缓存。

open_log_file_cache 指令主要用来设置含有变量的日志路径的文件描述符缓存，它的语法如下：

```
open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time] [off]
```

该指令默认是禁止的，等同于：

```
open_log_file_cache off;
```

open_log_file_cache 指令的各项参数说明如下：

max: 设置缓存中的最大文件描述符数量。如果超过设置的最大文件描述符数量，则采用 LRU (Least Recently Used) 算法清除“较不常使用的文件描述符”。LRU (Least Recently Used) 算法的基本概念是：当内存缓冲区剩余的可用空间不够时，缓冲区尽可能地先保留使用者最常使用的数据，将最近未使用的数据移出内存，腾出空间来加载另外的数据。

inactive: 设置一个时间，如果在设置的时间内没有使用此文件描述符，则自动删除此描述符。此参数为可选参数，默认的时间为 10 秒钟。

min_uses: 在参数 inactive 指定的时间范围内，如果日志文件超过被使用的次数，则将该日志文件的描述符记入缓存。默认次数为 1。

valid: 设置多长时间检查一次，看一看变量指定的日志文件路径与文件名是否仍然存在。默认时间为 60 秒。

off: 禁止使用缓存。

open_log_file_cache 指令的设置示例如下：

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

3.3.3 Nginx 日志文件的切割

生产环境中的服务器, 由于访问日志文件增长速度非常快, 日志太大会严重影响服务器效率。同时, 为了方便对日志进行分析计算, 须要对日志文件进行定时切割。定时切割的方式有按月切割、按天切割、按小时切割等。最常用的是按天切割。

Nginx 不支持像 Apache 一样使用 cronolog 来轮转日志, 但是可以采用以下方式来实现日志文件的切割:

```
mv /data1/logs/access.log /data1/logs/20090318.log  
kill -USR1 Nginx 主进程号
```

首先通过 mv 命令将日志文件重命名为/data1/logs/20090318.log, 然后发送 kill -USR1 信号给 Nginx 的主进程号, 让 Nginx 重新生成一个新的日志文件/data1/logs/access.log。如果 nginx.conf 配置文件中使用了“pid /usr/local/webserver/nginx/nginx.pid;”指令, 指定了 pid 文件的存放路径, 我们可以通过 cat 这个 pid 文件获得 Nginx 的主进程号, 命令如下:

```
kill -USR1 `cat /usr/local/webserver/nginx/nginx.pid`
```

如果想每天定时切割日志, 还须要借助 crontab。我们可以写一个按天切割的日志, 按年、月份目录存放日志的 shell 脚本:

```
vi /usr/local/webserver/nginx/sbin/cut_nginx_log.sh
```

输入以下内容并保存 (见代码 3-9):

代码 3-9

```
#!/bin/bash  
# 这个脚本须在每天的 00:00 运行  
  
# Nginx 日志文件的存放路径  
logs_path="/data1/logs/"  
  
mkdir -p ${logs_path}${date -d "yesterday" +%Y}/${date -d "yesterday" +%m}/  
mv ${logs_path}access.log ${logs_path}${date -d "yesterday" +%Y}/${date -d "yesterday" +%m}/access_${date -d "yesterday" +%Y%m%d}.log  
kill -USR1 `cat /usr/local/webserver/nginx/nginx.pid`
```

另外, 配置 crontab 每天凌晨 00:00 定时执行这个脚本:

```
crontab -e
```

输入以下内容并保存:

```
00 00 * * * /bin/bash /usr/local/webserver/nginx/sbin/cut_nginx_log.sh
```

这个 shell 脚本和 crontab 配置主要实现的功能为：假设今天的日期为 2009 年 5 月 19 日，Nginx 当前的日志文件为/data1/logs/access.log，2009 年 5 月 20 日 00:00 会执行 cut_nginx_log.sh 脚本，cut_nginx_log.sh 脚本首先创建一个目录/data1/logs/2009/05/，然后将/data1/logs/access.log 文件移动并重命名为/data1/logs/2009/05/access_20090519.log，再发送 kill -USR1 信号给 Nginx 主进程号，告诉 Nginx 重新生成一个/data1/logs/access.log 文件，2009 年 5 月 20 日的日志记录在这个新生成的日志文件中。而/data1/logs/2009/05/access_20090519.log 文件，就是 2009 年 5 月 19 日的日志文件。

3.4 Nginx 的压缩输出配置

gzip (GNU-ZIP) 是一种压缩技术。经过 gzip 压缩后页面大小可以变为原来的 30% 甚至更小。这样，用户浏览页面的时候速度会快得多。gzip 的压缩页面需要浏览器和服务器双方都支持，实际上就是服务器端压缩，传到浏览器后浏览器解压并解析。浏览器那里不需要我们担心，因为 IE、Firefox、Opera、谷歌 Chrome 等绝大多数浏览器都支持解析 gzip 过的页面。

Nginx 的压缩输出由一组 gzip 压缩指令来实现。我们从示例入手，来学习 gzip 压缩输出。gzip 压缩输出的相关指令位于 http{.....} 两个大括号之间：

```
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.1;
gzip_comp_level 2;
gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;
```

gzip 各指令的语法和使用方法请查阅本书第 13 章第 13.11 节的 gzip 模块。

3.5 Nginx 的自动列目录配置

我们经常会看到一些开源软件的下载页面是能够自动列目录的，这一功能 Apache 可以实现，Nginx 同样可以实现，前提条件是当前目录下不存在用 index 指令设置的默认首页文件。如果须要在某一虚拟主机的 location / {.....} 目录控制中配置自动列目录，只须加上如下代码：

```
location / {
    autoindex on;
}
```

另外，还有两项跟自动列目录相关的指令，分别为：

```
autoindex_exact_size [ on|off ]
```

设定索引时文件大小的单位（B、KB、MB 或 GB）

```
autoindex_localtime [ on|off ]
```

开启以本地时间来显示文件时间的功能。默认为关（GMT 时间）。

3.6 Nginx 的浏览器本地缓存设置

浏览器缓存（Browser Caching）是为了加速浏览，浏览器在用户磁盘上对最近请求过的文档进行存储，当访问者再次请求这个页面时，浏览器就可以从本地磁盘显示文档，这样就可以加速页面的阅览。缓存的方式节约了网络的资源，提高了网络的效率。

浏览器缓存可以通过 expires 指令输出 Header 头来实现，expires 指令的语法如下：

语法：expires [timeepoch|max|off]

默认值：expires off

作用域：http, server, location

用途：使用本指令可以控制 HTTP 应答中的“Expires”和“Cache-Control”的 Header 头信息（起到控制页面缓存的作用）。

可以在 time 值中使用正数或负数。“Expires”头标的值将通过当前系统时间加上您设定的 time 值来获得。

epoch 指定“Expires”的值为 1 January, 1970, 00:00:01 GMT。

max 指定“Expires”的值为 31 December 2037 23:59:59 GMT，“Cache-Control”的值为 10 年。-1 指定“Expires”的值为服务器当前时间-1s，即永远过期。

“Cache-Control”头标的值由您指定的时间来决定。

负数：Cache-Control: no-cache。

正数或零：Cache-Control: max-age = #, # 为您指定时间的秒数。

“off”表示不修改“Expires”和“Cache-Control”的值。

假设一个 HTML 页面中会引用一些 JavaScript 文件、图片文件，而这些格式的文件很少会被修改，则可以通过 expires 设置浏览器缓存。

例：对常见格式的图片、Flash 文件在浏览器本地缓存 30 天，对 js、css 文件在浏览器本地缓存 1 小时，如代码 3-10 所示。



代码 3-10

```
location ~ .*\. (gif|jpg|jpeg|png|bmp|swf)$
{
    expires 30d;
}

location ~ .*\. (js|css)?$
{
    expires 1h;
}
```